



## Diplomarbeit

# Work-Area Booking System (W.A.B.S.)

Arbeitsplatzbuchungssystem

Patrick Bayr

Sonja Lechner

Manuel Payer

Marcel Schranz

Imst, 12. September 2023

Betreut durch:

DI Dr. Michael Netzer

DI<sup>in</sup> Dr.<sup>in</sup> Melanie Osl

# **Eidesstattliche Erklärung**

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbst verfasst und keine anderen als die angeführten Behelfe verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht. Ich bin damit einverstanden, dass meine Arbeit öffentlich zugänglich gemacht wird.

---

Imst, 23. Oktober 2022

---

Patrick Bayr

---

Sonja Lechner

---

Manuel Payer

---

Marcel Schranz

# **Vorwort**

Durch COVID-19 wurden die Mitarbeiter: innen des Klimabündnisses Tirol aufgefordert, im Homeoffice zu arbeiten. Auch nach den Lockerungen der Pandemiemaßnahmen möchte das Klimabündnis Tirol dem Wunsch der Angestellten nach Weiterführung der Wahlmöglichkeit zwischen Heimarbeit und Arbeiten im Büro nachkommen. Die Flexibilität des Arbeitsplatzes wird von den Mitarbeiter:innen als großer Mehrwert definiert.

Dieses Projekt dient der Entwicklung einer Softwareanwendung, welche von den Mitarbeiter:innen des Klimabündnisses Tirol verwendet wird, um via Remote einen Arbeitsplatz innerhalb der Büroräumlichkeiten buchen zu können. Darüber hinaus sind die Angestellten in der Lage, ebenfalls diverse Büroausstattungen sowie das firmeneigene Lastenfahrrad über diese Anwendung zu buchen.

# Kurzfassung

Die Covid-19-Pandemie hat zu erheblichen Veränderungen am Arbeitsplatz geführt, insbesondere in Bezug auf die Art und den Ort der Arbeit. Viele Unternehmen führen ein hybrides Modell ein, bei dem zwischen Fernarbeit und Büroarbeit gewählt werden kann.

Dieses Modell brachte jedoch neue Herausforderungen mit sich, wie zum Beispiel mögliche Engpässe bei Arbeitsplätzen, wenn punktuell zu viele Mitarbeiter:innen ins Büro kommen. Um den Projektpartner, Klimabündnis Tirol, bei derartigen Planungsschwierigkeiten zu unterstützen, besteht das Projektziel darin, ein webbasiertes Anwendungssystem für die einfache Verwaltung von Arbeitsplätzen zu entwickeln. Das Work-Area Booking System (WABS) ist eine umfassende Lösung, die die Herausforderungen bei der Verwaltung von Büroräumen in einer hybriden Arbeitsumgebung bewältigt. Durch die Bereitstellung einer benutzerfreundlichen Plattform für die Buchung von Arbeitsplätzen und die Verwaltung von Ressourcen ermöglicht die App dem Projektpartner eine effizientere Arbeitsplatzverteilung, größere Flexibilität bei den Arbeitszeiten der Mitarbeiter:innen, Kosteneinsparungen aufgrund verbesserter Arbeitsplatz- und Ressourcenverwaltung, eine bessere Organisation von persönlichen Meetings und die gemeinsame Nutzung von Ressourcen.

# Abstract

The Covid-19 pandemic has led to significant changes in the workplace, especially in terms of how and where work is done. Many companies have introduced a hybrid model, allowing employees to choose between remote work and office work. However, this model has brought new challenges, such as potential shortages of workspaces when too many employees come to the office. To support our project partner, Klimabündnis Tirol, in dealing with such scheduling difficulties, the project goal is to develop a web-based application system to manage workstations.

The Work-Area Booking System (WABS) is a comprehensive solution that addresses the challenges of managing office spaces in a hybrid work environment. By providing a user-friendly platform for booking workspaces and managing resources, the app enables the project partner to achieve more efficient workspace allocation, greater flexibility in employees' working hours, cost savings due to improved workspace and resource management, better organization of personal meetings and shared resources.

# Inhaltsverzeichnis

|       |  |    |
|-------|--|----|
| 1     | Einleitung .....   | 15 |
| 2     | Projektmanagement .....  | 16 |
| 2.1   | Metainformationen .....  | 16 |
| 2.1.1 | Projektteam .....  | 16 |
| 2.1.2 | Projektbetreuer .....  | 16 |
| 2.1.3 | Projektpartner .....   | 16 |
| 2.2   | Vorerhebungen.....   | 17 |
| 2.2.1 | Ist-Zustand .....  | 17 |
| 2.2.2 | Soll-Zustand .....   | 17 |
| 2.2.3 | Funktionalitäten.....  | 18 |
| 2.2.4 | Projektumfeldanalyse .....   | 19 |
| 2.3   | Problemanalyse .....   | 23 |
| 2.3.1 | User-Stories .....   | 23 |
| 2.4   | Planung .....  | 28 |
| 2.4.1 | Projektstrukturplan .....  | 28 |
| 2.4.2 | Projektlaufplan.....   | 29 |
| 2.5   | Risikoanalyse.....   | 31 |
| 2.5.1 | Risikomatrix .....   | 33 |
| 3     | Systemdokumentation.....   | 36 |
| 3.1   | Generelle Systemübersicht .....  | 36 |
| 3.1.1 | Verwendete Technologien und Werkzeuge.....                               | 37 |
| 3.1.2 | Logischer Aufbau .....   | 39 |
| 3.1.3 | Physischer Aufbau.....   | 39 |
| 3.1.4 | Konzept für Ausnahmebehandlung .....                                     | 41 |
| 3.1.5 | Klassen und Komponenten .....  | 41 |
| 3.2   | Systematische Literatursuche .....                                       | 42 |
| 3.3   | Einordnung der Technologien und Alternativen .....                       | 43 |
| 3.3.1 | Realisierung der Datenbank mit MariaDB.....                              | 43 |
| 3.3.2 | Spring Boot & Security .....   | 43 |
| 3.3.3 | Umsetzung des Frontends mit Thymeleaf .....                              | 44 |
| 3.3.4 | Darstellung und Interagierbarkeit durch JavaScript,JqueryUI,SVG.js ..... | 44 |
| 4     | Testfälle .....  | 45 |
| 5     | Technologieauswahl und Datenbankentscheidung .....                       | 47 |

|       |  |    |
|-------|--|----|
| 5.1   | Analyse von Technologieoptionen .....                            | 47 |
| 5.2   | Die Funktionen von Spring.....                                   | 48 |
| 5.3   | SpringBoot als Framework.....                                    | 49 |
| 5.4   | Sicherung von Daten.....   | 49 |
| 5.4.1 | Firebase .....   | 49 |
| 5.4.2 | Der Übergang von Firebase zu einer relationalen Datenbank .....  | 50 |
| 5.4.3 | H2-Datenbank.....  | 50 |
| 6     | Mitarbeitermodul .....   | 52 |
| 6.1   | Die Schichten des Mitarbeitermoduls.....                         | 52 |
| 6.1.1 | Domain Layer .....   | 52 |
| 6.1.2 | Repository-Layer.....  | 55 |
| 6.1.3 | Service-Layer.....   | 59 |
| 6.1.4 | Controller-Layer.....  | 63 |
| 6.1.5 | Frontend .....   | 67 |
| 7     | Spring Security im Arbeitsplatzbuchungssystem .....              | 73 |
| 7.1   | Warum Spring Security? .....                                     | 73 |
| 7.2   | Weitere Spring Security Funktionen .....                         | 73 |
| 7.3   | Implementierung der Sicherheitskonfiguration.....                | 74 |
| 7.4   | Implementierung des Login Controllers.....                       | 78 |
| 7.4.1 | Exemplarische Darstellung des Login-Vorgangs.....                | 80 |
| 8     | Ressourcenmodul .....  | 84 |
| 8.1   | Einführung: .....  | 84 |
| 8.2   | Backend-Logik des Ressourcenmoduls .....                         | 84 |
| 8.2.1 | Struktur des Backends .....                                      | 84 |
| 8.2.2 | Domain Layer .....   | 85 |
| 8.2.3 | Repository Layer .....   | 85 |
| 8.2.4 | Service Layer .....  | 85 |
| 8.2.5 | Funktionsweise .....   | 86 |
| 8.3   | Implementierung .....  | 89 |
| 8.3.1 | Technologien im Backend:.....                                    | 89 |
| 8.3.2 | Ressource-Webcontroller .....                                    | 92 |
| 8.4   | Zusammenfassung des Ressourcen- und RessourceBooking-Moduls..... | 95 |
| 9     | Front-End Modul.....   | 96 |
| 9.1   | Struktur.....  | 96 |
| 9.2   | Funktionsweise .....   | 97 |

|         |  |     |
|---------|--|-----|
| 9.3     | Technologien im Frontend.....                        | 102 |
| 9.4     | Implementierung .....                                | 102 |
| 9.5     | Zusammenfassung Front-End Modul.....                 | 112 |
| 10      | Raummodul .....                                      | 113 |
| 10.1    | Einführung .....                                     | 113 |
| 10.2    | Verwendete Technologien.....                         | 113 |
| 10.3    | Raum Modul Backend.....                              | 114 |
| 10.3.1  | Struktur des Backends .....                          | 114 |
| 10.4    | Codeauszüge und Umsetzung.....                       | 116 |
| 10.5    | Domänenlayer .....                                   | 116 |
| 10.5.1  | Klasse Room.....                                     | 118 |
| 10.5.2  | Klasse RoomBooking .....                             | 120 |
| 10.6    | Repositorylayer .....                                | 121 |
| 10.6.1  | Interface DBAccessRoom.....                          | 122 |
| 10.6.2  | Interface RoomJPARepr .....                          | 122 |
| 10.6.3  | Klasse DBAccessRoomJPHA2.....                        | 124 |
| 10.6.4  | Interface RoomBookingRepo .....                      | 126 |
| 10.6.5  | Interface RoomBookingJPARepr.....                    | 127 |
| 10.6.6  | Klasse RoomBookingRepo_JPHA2.....                    | 128 |
| 10.7    | Servicelayer.....                                    | 130 |
| 10.7.1  | RoomServiceImplementation .....                      | 131 |
| 10.7.2  | RoomBookingServiceImplementation .....               | 133 |
| 10.8    | Raum Modul Frontend.....                             | 134 |
| 10.8.1  | Struktur, Controllerlayer und Thymeleaf .....        | 134 |
| 10.8.2  | RoomWebController.....                               | 136 |
| 10.8.3  | AllRooms Template .....                              | 137 |
| 10.8.4  | Ansicht AllRooms Template im Browser .....           | 139 |
| 10.8.5  | Zusammenfassung MVC.....                             | 139 |
| 10.9    | Kernmodul Raumansicht, Raumauswahl und Buchung ..... | 141 |
| 10.10   | Codeauszüge zum Kernmodul .....                      | 143 |
| 10.10.1 | Verwendung von JavaScript.....                       | 145 |
| 11      | Deskbooking Modul .....                              | 147 |
| 11.1    | Einführung .....                                     | 147 |
| 11.2    | Struktur der WABS-Anwendung .....                    | 148 |
| 11.2.1  | Vorteile der Schichtenarchitektur.....               | 148 |

|        |   |     |
|--------|---|-----|
| 11.3   | Domäne .....                                    | 149 |
| 11.3.1 | Desk .....                                      | 149 |
| 11.3.2 | Port .....                                      | 152 |
| 11.3.3 | Booking .....                                   | 152 |
| 11.3.4 | PublicHoliday .....                             | 155 |
| 11.3.5 | Timeslot .....                                  | 156 |
| 11.3.6 | DeskBooking .....                               | 157 |
| 11.4   | Repository.....                                 | 160 |
| 11.4.1 | Desk Repository Schicht.....                    | 161 |
| 11.4.2 | DeskBooking Repository Schicht.....             | 165 |
| 11.4.3 | Andere Komponenten der Repository Schicht.....  | 173 |
| 11.5   | Service .....                                   | 173 |
| 11.5.1 | DeskServiceImplementation (Klasse) .....        | 174 |
| 11.5.2 | DeskbookingServiceImplementation (Klasse) ..... | 174 |
| 11.6   | Presentation Layer.....                         | 178 |
| 11.7   | Testen des Deskbooking-Moduls .....             | 186 |
| 12     | Evaluation .....                                | 188 |
| 12.1   | Projektevaluierung.....                         | 188 |
| 12.2   | Produktevaluierung .....                        | 189 |
| 12.2.1 | Google Firebase .....                           | 189 |
| 12.2.2 | Frontend .....                                  | 189 |
| 12.2.3 | JQuery UI .....                                 | 190 |
| 12.3   | Resümeees.....                                  | 190 |
| 12.3.1 | Resümee Bayr .....                              | 190 |
| 12.3.2 | Resümee Lechner .....                           | 192 |
| 12.3.3 | Resümee Payer .....                             | 193 |
| 12.3.4 | Resümee Schranz.....                            | 194 |
| 13     | Zusammenfassung .....                           | 196 |
| 14     | Literaturverzeichnis .....                      | 197 |
| A      | Anhang.....                                     | 200 |
| A.1.   | Testfälle .....                                 | 201 |
| A.2.   | Zeitprotokoll .....                             | 207 |

# Abbildungsverzeichnis

|   |    |
|---|----|
| Abbildung 1: Grafische Darstellung der Stakeholder .....  | 20 |
| Abbildung 2: Use Case Diagramm.....   | 24 |
| Abbildung 3: Projektstrukturplan.....   | 28 |
| Abbildung 4: Gantt Chart .....  | 30 |
| Abbildung 5: Risikomatrix .....   | 34 |
| Abbildung 6: Risikomatrix v2.....   | 35 |
| Abbildung 7: Schichtenstruktur .....  | 37 |
| Abbildung 8: Dependencies in der pom.xml .....  | 38 |
| Abbildung 9: Komponentendiagramm.....   | 40 |
| Abbildung 10: ER-Diagramm.....  | 42 |
| Abbildung 11: Dynamische Testung Ressourcen .....   | 46 |
| Abbildung 12 - @Service-Annotation der EmployeeServiceImplementation-Klasse .....   | 48 |
| Abbildung 13 - Domain Layer.....  | 52 |
| Abbildung 14 – Annotationen der Employee Entität .....  | 52 |
| Abbildung 15 - Employee Datenfeld id .....  | 53 |
| Abbildung 16 – Spezifische Employee Datenfelder.....  | 53 |
| Abbildung 17 - Konstruktor der Employee-Klasse.....   | 54 |
| Abbildung 18 - Hier wird eine Liste von Berechtigungen/Rollen zurückgegeben .....   | 54 |
| Abbildung 19 - Gibt den Spitznamen der Mitarbeiterin/des Mitarbeiters zurück .....  | 54 |
| Abbildung 20 - Spring Security UserDetails Methoden.....  | 55 |
| Abbildung 21 - Repository Layer .....   | 55 |
| Abbildung 22 - Interface für den Datenzugriff auf Mitarbeiter:innen.....  | 56 |
| Abbildung 23 - Das EmployeeJPARepo erweitert das JpaRepository mit benutzerdefinierten Methoden .....                         | 56 |
| Abbildung 24 - Konkrete Implementierung für den Mitarbeiterdatenzugriff .....   | 57 |
| Abbildung 25 - Konstruktor der EmployeeDBAccess_JPAH2-Klasse .....  | 57 |
| Abbildung 26 - Die saveEmployee()-Methode speichert eine:n Mitarbeiter:in .....   | 57 |
| Abbildung 27 - Liefert eine Liste aller Mitarbeiter:innen zurück.....   | 58 |
| Abbildung 28 - Sucht eine:n Mitarbeiter:in anhand der ID .....  | 58 |
| Abbildung 29 - Löscht eine:n Mitarbeiter:in mithilfe der "id" .....   | 58 |
| Abbildung 30 - Sucht eine:n Mitarbeiter:in unter Verwendung der E-Mail-Adresse .....  | 58 |
| Abbildung 31 - Gibt eine:n Mitarbeiter:in durch Angabe des Spitznamens zurück .....   | 59 |
| Abbildung 32 - Service-Layer .....  | 59 |
| Abbildung 33 - EmployeeService-Interface.....   | 60 |
| Abbildung 34 – Die konkrete Implementierung der Geschäftslogik für die Verwaltung von Mitarbeiterinnen und Mitarbeitern ..... | 60 |
| Abbildung 35 - Konstruktor der EmployeeServiceImplementation-Klasse .....   | 61 |
| Abbildung 36 - Speichert eine:n Mitarbeiter:in und gibt diese oder diesen zurück.....   | 61 |
| Abbildung 37 - Gibt eine Liste aller Mitarbeiter:innen zurück .....   | 61 |
| Abbildung 38 - Sucht eine Mitarbeiterin oder einen Mitarbeiter unter Verwendung der id .....                                  | 62 |
| Abbildung 39 - Aktualisiert eine:n Mitarbeiter:in durch Angabe der id und gibt diesen zurück .....                            | 62 |
| Abbildung 40 - Löscht eine:n Mitarbeiter:in anhand der ID.....  | 62 |
| Abbildung 41 - Sucht eine:n Mitarbeiter:in basierend auf dem Spitznamen und gibt diesen zurück ....                           | 63 |

|   |    |
|---|----|
| Abbildung 42 - Lädt die Benutzerdetails anhand des Mitarbeiternamens und gibt ein UserDetails-Objekt zurück ..... | 63 |
| Abbildung 43 - Controller Layer .....   | 63 |
| Abbildung 44 - EmployeeWebController-Klasse .....   | 64 |
| Abbildung 45 - Konstruktor der EmployeeWebController-Klasse .....   | 64 |
| Abbildung 46 - Controller-Endpunkt für die Ansicht aller Mitarbeiter:innen.....                                   | 64 |
| Abbildung 47- Controller-Endpunkt für die Ansicht der Admin-Startseite.....                                       | 65 |
| Abbildung 48 - Controller-Endpunkt für Ansicht der User-Startseite .....  | 65 |
| Abbildung 49 - Controller-Endpunkt zum Löschen eines Mitarbeiters .....   | 65 |
| Abbildung 50 - Controller-Endpunkt für die Ansicht zum Einfügen von Mitarbeiterinnen und Mitarbeitern .....       | 65 |
| Abbildung 51 - Controller-Endpunkt zum Validieren und Speichern von Mitarbeiterinnen und Mitarbeitern .....       | 66 |
| Abbildung 52 - Controller-Endpunkt für die Ansicht für das Bearbeiten von Mitarbeiterinnen und Mitarbeitern ..... | 66 |
| Abbildung 53 - Controller-Endpunkt für die Aktualisierung von Mitarbeiterinnen und Mitarbeitern ...               | 66 |
| Abbildung 54 - Controller-Endpunkt fü die Ansicht der Admin-Startseite .....                                      | 67 |
| Abbildung 55 - Ansicht der Admin-Startseite.....  | 67 |
| Abbildung 56 - HTML Seite für die Admin-Startseite .....  | 68 |
| Abbildung 57 - Administrator Startseite Verwaltung von Ressourcen .....   | 68 |
| Abbildung 58 - HTML-Seite für die Admin Navigationsleiste .....   | 69 |
| Abbildung 59 - Controller-Endpunkt für die Ansicht aller Mitarbeiter:innen.....                                   | 69 |
| Abbildung 60 - Methode des Employee-Service Interfaces .....  | 69 |
| Abbildung 61 - Methode zum Abrufen aller Mitarbeiter der EmployeeServiceImplementation-Klasse                     | 70 |
| Abbildung 62 - Methode zum Abrufen aller Mitarbeiter des EmployeeDBAccess-Interfaces .....                        | 70 |
| Abbildung 63 - Methode zum Abrufen aller Mitarbeiter der EmployeeDBAccess_JPAH2-Klasse.....                       | 70 |
| Abbildung 64 - HTML-Seite allemployees.html die eine Liste aller Mitarbeiter:innen zurückgibt .....               | 71 |
| Abbildung 65 - Liste aller Mitarbeiter:innen .....  | 72 |
| Abbildung 66 - WebSecurityConfig-Klasse - Enthält die Sicherheitseinstellungen.....                               | 74 |
| Abbildung 67 - Konstruktor der WebSecurityConfig-Klasse.....  | 74 |
| Abbildung 68 - Diese Methode gibt ein Bean des Typs BCryptPasswordEncoder zurück .....                            | 74 |
| Abbildung 69 - Sicherheitsregeln für HTTP-Anfragen .....  | 75 |
| Abbildung 70 - Formular-Login Konfigurationen.....  | 75 |
| Abbildung 71 - Konfiguration für das Abmelden von Mitarbeitern.....   | 76 |
| Abbildung 72 - Hier wird die Konfiguration der Sicherheitseinstellungen abgeschlossen .....                       | 77 |
| Abbildung 73 - AuthenticationManger für die Authentifizierung .....   | 77 |
| Abbildung 74- Abrufen der Benutzerdaten und Erstellung eines User-Objekts für die Authentifizierung .....         | 77 |
| Abbildung 75 - LoginWebController-Klasse .....  | 78 |
| Abbildung 76 - Konstruktor der LoginWebController-Klasse .....  | 78 |
| Abbildung 77 - Controller-Endpunkt für die Ansicht des Login-Formulars.....                                       | 78 |
| Abbildung 78 - Controller-Endpunkt für die Authentifizierung von Mitarbeitern.....                                | 79 |
| Abbildung 79 - Controller-Endpunkt für die Abmeldung eines Mitarbeiters .....                                     | 79 |
| Abbildung 80 - Controller-Endpunkt für die Login-Fehlerseite .....  | 79 |
| Abbildung 81 - Controller-Endpunkt für den Login.....   | 80 |
| Abbildung 82 - HTML-Seite für den Login-Vorgang .....   | 80 |
| Abbildung 83 - Login-Seite der W.A.B.S.-Applikation.....  | 81 |
| Abbildung 84 - Login mit korrekten Anmeldedaten.....  | 81 |

|  |     |
|--|-----|
| Abbildung 85 - Die Methode authentifiziert den Mitarbeiter .....                             | 82  |
| Abbildung 86 - Ausschnitt der securityFilterChain-Methode der WebSecurityConfig-Klasse ..... | 82  |
| Abbildung 87 - Controller-Endpunkt für die Ansicht der Administrator-Startseite .....        | 83  |
| Abbildung 88 - Administrator-Startseite der W.A.B.S.-Applikation.....                        | 83  |
| Abbildung 89 - Package Struktur des Back-Ends .....  | 85  |
| Abbildung 90 - Ressourcenliste im Front-End.....   | 86  |
| Abbildung 91 - Ressourcen GUI im Front-End.....  | 87  |
| Abbildung 92 - Liste aller Ressourcen in der Datenbank.....                                  | 88  |
| Abbildung 93 - Codesnippet aus der addRessource.html .....                                   | 88  |
| Abbildung 94 - Eingebundene Pakete der Domain Class: Ressource .....                         | 89  |
| Abbildung 95 - Konstruktor der Domain Class: Ressource .....                                 | 90  |
| Abbildung 96 - Abbildung einer Bean vor einer Methode .....                                  | 90  |
| Abbildung 97 - Struktur der Repository Schicht.....  | 91  |
| Abbildung 98 - Methode mit Validierungsprüfung .....   | 92  |
| Abbildung 99 - Methode im WebController .....  | 93  |
| Abbildung 100 - CodeSnippet aus dem WebController .....                                      | 94  |
| Abbildung 101 - Struktur der Front-end Dateien .....   | 96  |
| Abbildung 102 - Auszug des Templates addRessource.html.....                                  | 98  |
| Abbildung 103 - Front-End der Admin Maske um Ressourcen hinzuzufügen.....                    | 99  |
| Abbildung 104 - Top Bar Admin .....  | 99  |
| Abbildung 105 - Top Bar Employee.....  | 99  |
| Abbildung 106.1 - Nav.Bar Employee .....   | 100 |
| Abbildung 107 - Nav.Bar Admin.....   | 100 |
| Abbildung 108 - Aufruf des ausgelagerten Codes der navbar Admin .....                        | 103 |
| Abbildung 109 - Ausgelagerter Code der NavBar Admin .....                                    | 103 |
| Abbildung 110 - Java Script Funktion für den Dropdown Effekt .....                           | 104 |
| Abbildung 111 - CodeSnippet der JavaScript Datei die die meisten Funktionen beinhaltet ..... | 106 |
| Abbildung 112 - CSS Regeln des Delete-Buttons .....  | 109 |
| Abbildung 113 - Aufruf des Delete-Buttons.....   | 110 |
| Abbildung 114 - Media Rules .....  | 111 |
| Abbildung 115- zeigt die Schichtenarchitektur .....  | 115 |
| Abbildung 116 - Unterscheidung zwischen Domäne und Domänenbuchung .....                      | 117 |
| Abbildung 117 - Codeausschnitt aus der Domänenklasse 'Room' .....                            | 118 |
| Abbildung 118- Codeausschnitt aus der Domänenklasse 'RoomBooking' .....                      | 120 |
| Abbildung 119 - Relevante Packages im Repositorylayer des Raummoduls .....                   | 121 |
| Abbildung 120 - Codeausschnitt aus dem Repository-Interface DBAccessRoom .....               | 122 |
| Abbildung 121 - Code des Repository-Interfaces RoomJPAReport.....                            | 122 |
| Abbildung 122 - Implementierende Klasse des Repositorylayers .....                           | 124 |
| Abbildung 123 - Raumbuchungs Interface.....  | 126 |
| Abbildung 124 - Technologiespezifisches Raumbuchungs-Interface .....                         | 127 |
| Abbildung 125 - Auszug aus der implementierenden Klasse der Raumbuchungen.....               | 128 |
| Abbildung 126 - Packages der raumrelevanten Klassen im Servicelayer .....                    | 130 |
| Abbildung 127 - Code RoomServiceImplementation.....  | 131 |
| Abbildung 128-RoomBookingService Implementierung .....                                       | 133 |
| Abbildung 129- Controller-Layer.....   | 134 |
| Abbildung 130- Template/View .....   | 135 |
| Abbildung 131 – RoomWebController.....   | 136 |
| Abbildung 132- Template rooms/allRooms .....   | 137 |

|   |     |
|---|-----|
| Abbildung 133 - Benutzeroberfläche Admin der Raumansicht als Liste .....          | 139 |
| Abbildung 134 - Abbildung des Grundrisses .....                                   | 142 |
| Abbildung 135 - Ansicht nach Klick auf Raum 3 .....                               | 143 |
| Abbildung 136- Ansicht nach Klick AP1.....  | 143 |
| Abbildung 137- Umsetzung der Imagemap .....                                       | 144 |
| Abbildung 138 - Einbindung JavaScript.....  | 145 |
| Abbildung 139 - scaleImageMap-Funktion .....                                      | 146 |
| Abbildung 140 - Übersicht - Projektstruktur .....                                 | 148 |
| Abbildung 141 – Attribute der Desk Klasse.....                                    | 150 |
| Abbildung 142 – Methode der Desk Klasse .....                                     | 151 |
| Abbildung 143 – Attribute der Booking Klasse.....                                 | 153 |
| Abbildung 144 – Konstruktors der Booking Klasse.....                              | 154 |
| Abbildung 145 – Methode der Booking Klasse .....                                  | 154 |
| Abbildung 146 - PublicHoliday Klasse .....  | 155 |
| Abbildung 147 - Timeslot Klasse .....   | 157 |
| Abbildung 148 - DeskBooking Konstruktor mit start und endTime.....                | 158 |
| Abbildung 149 - DeskBooking Konstruktor mit Timeslot .....                        | 158 |
| Abbildung 150 - Deskbooking Methode .....   | 159 |
| Abbildung 151 – DeskJPARepo .....   | 161 |
| Abbildung 152 - DeskRepo Interface .....  | 162 |
| Abbildung 153 - DeskJPARepo – Teil 2.....   | 163 |
| Abbildung 154 - DeskJPARepo – Teil 3.....   | 164 |
| Abbildung 155 - DeskBookingRepo_JPAH2 addBooking Methode .....                    | 167 |
| Abbildung 156 - DeskBookingRepo_JPAH2 getAllBookings Methode.....                 | 168 |
| Abbildung 157 - DeskBookingRepo_JPAH2 searchBookings Methode .....                | 169 |
| Abbildung 158 - DeskbookingRepo_JPAH2 getBookingByBookingId Methode .....         | 170 |
| Abbildung 159 - DeskbookingRepo_JPAH2 updateBookingByld Methode .....             | 171 |
| Abbildung 160 - DeskbookingRepo_JPAH2 deleteBookingByld Methode .....             | 171 |
| Abbildung 161 - DeskbookingRepo_JPAH2 getAvailableDesks Methode .....             | 172 |
| Abbildung 162 - DeskbookingServiceImplementation - addDeskbooking Method.....     | 175 |
| Abbildung 163 - DeskbookingServiceImplementation - getAllBookings Method.....     | 175 |
| Abbildung 164 - DeskbookingServiceImplementation - searchbookings Method .....    | 176 |
| Abbildung 165 - DeskbookingServiceImplementation - getBookingByld Method .....    | 176 |
| Abbildung 166 - DeskbookingServiceImplementation - updateBookingByld Method ..... | 177 |
| Abbildung 167 - DeskbookingServiceImplementation - getAvailableDesks Method ..... | 177 |
| Abbildung 168 - ViewConstants .....   | 179 |
| Abbildung 169 - PathConstants .....   | 180 |
| Abbildung 170 - WABS - Admin View - AllDeskbookings.....                          | 182 |
| Abbildung 171 - DeskbookingController Method - updateDeskBookingForm.....         | 182 |
| Abbildung 173 - WABS - Update Deskbooking Formular .....                          | 183 |
| Abbildung 174 - Update Deskbooking Thymeleaf Template.....                        | 184 |
| Abbildung 175 - DeskbookingController Methode - updateDeskBooking .....           | 185 |
| Abbildung 176 - WABS – AllDeskbookings Ansicht .....                              | 185 |
| Abbildung 177 - testAddBoking Method.....   | 187 |

# Tabellenverzeichnis

|   |     |
|---|-----|
| Tabelle 1: Klassifizierung der Stakeholder.....         | 21  |
| Tabelle 2: Maßnahmen pro Stakeholder.....               | 22  |
| Tabelle 3: Use Case - Arbeitsplatz buchen .....         | 25  |
| Tabelle 4: Use Case - Mitarbeiter:in Konto anlegen..... | 26  |
| Tabelle 5: Änderung der Büroräumlichkeiten .....        | 27  |
| Tabelle 6: Risikoportfolio.....                         | 33  |
| Tabelle 7 - Testfall GetAllDeskBookings .....           | 201 |
| Tabelle 8: Test-Schritte getAllBookings.....            | 202 |
| Tabelle 9: Testfall updateEmployee .....                | 205 |

# 1 Einleitung

Das Klimabündnis Tirol hatte einen Antrag für eine Softwarelösung an das IT-Kolleg Imst gestellt, welches von Frau Dr.in Dorothea Schumacher (ehemalige Lehrperson an der HTL Imst) an Projektteam übermittelt wurde. Dabei ging es darum, eine technische Lösung für ein arbeitsinternes Problem zu finden und zu erstellen.

Das Arbeitsplatzbuchungssystem (Work-Area Booking System - W.A.B.S) soll genau diese Aufgaben erfüllen. Neben der unkomplizierten Handhabung erfüllt die Anwendung noch weitere Aspekte, die im Laufe der vorliegenden Arbeit genauer beschrieben werden.

# 2 Projektmanagement

Im folgenden Abschnitt werden die Methoden und die Dokumentation der Projektumfeldanalyse dargestellt. Ebenso folgen einige Informationen über die Struktur des Projektteams sowie deren Betreuer und Partner.

## 2.1 Metainformationen

Im folgenden Abschnitt wird die Organisation des Projektteams sowie die Kontaktdaten der Betreuer und des Projektpartners erfasst.

### 2.1.1 Projektteam

- Die Position des Projektleiters und die damit verbundenen Aufgaben (Überwachung und Koordinierung der Arbeitsprozesse) sowie die Raumverwaltung fällt in den Aufgabenbereich von **Herrn Patrick Bayr** sowie das Buchungsmodul für das eigenes Modul inklusive Schnittstellenanbindung
- **Frau Sonja Lechner** widmet sich der Arbeitsplatzverwaltung und der Umsetzung der Internationalisierung.
- Der Aufgabenbereich von **Herrn Manuel Payer** beschäftigt sich mit der Implementierung des Ressourcen-Moduls sowie dem Frontend und dem Corporate Design.
- Die Pflege der Benutzergruppenverwaltung sowie die Mitarbeiterverwaltung fällt in den Verantwortungsbereich von **Herrn Marcel Schranz**.

### 2.1.2 Projektbetreuer

- Herr Dr. Michael Netzer befindet sich in der Position des Projektbetreuers.
- Frau Dr.in Melanie Osl hat sich freundlicherweise als Nebenbetreuerin zur Verfügung gestellt

### 2.1.3 Projektpartner

Firma

Klimabündnis Tirol  
Müllerstraße 7, 6020 Innsbruck  
Tel.: 0512 583558 -11  
Fax.: 0512 583558 -20  
Ansprechperson: Herr Andrä Stigger  
E-Mail.: [andrae.stigger@klimabundnis.at](mailto:andrae.stigger@klimabundnis.at)

## 2.2 Vorerhebungen

Der Abschnitt „Vorerhebungen“ beschreibt den gegenwärtigen Zustand des Projektes sowie den angestrebten Ziel-Zustand (Soll-Zustand). Weiters folgt eine ausführliche Erklärung zu den identifizierten Stakeholdern und deren Relevanz bzw. Charakterisierung zum Projekt.

### 2.2.1 Ist-Zustand

Der Ist-Zustand ist jener Zustand, den das Projekt zum gegenwärtigen Zeitpunkt hat. Im weiteren Abschnitt wird der gegenwärtige Zustand genauer eruiert:

- Mitarbeiter:innen haben aktuell keine technische Möglichkeit ihren Arbeitsplatz buchen zu können.
- Es kommt öfters zu Kollisionen am Arbeitsplatz weil die Mitarbeiter:innen keine Einsicht über die verfügbaren Arbeitsplätze vor Ort haben.
- Manche Arbeitsplätze bleiben hin und wieder unbesetzt, obwohl sie verfügbar wären.
- Das Buchen des Lastenfahrrades ist derzeit nicht transparent genug.
- Mitarbeiter:innen können diverse Büroausstattung erst vor Ort organisieren, was dazu führt, dass manche Ressourcen nicht mehr zur Verfügung stehen.
- Der Besprechungsraum kann aktuell nur mühsam mündlich/schriftlich reserviert werden. Eine Softwarelösung fehlt hier zur Gänze.
- Manche Arbeitsplätze werden regelmäßig von bestimmten Personen vorgezogen, was aktuell nicht ersichtlich ist.
- Das aktuelle System ist besonders für neue Mitarbeiter schwierig zu überblicken.

### 2.2.2 Soll-Zustand

Im Gegensatz zum Ist-Zustand beschäftigt sich der Soll-Zustand mit jenem Zustand, den das Projekt am Ende erreichen soll/muss. Nachfolgenden stehen die einzelnen Zustände, die nach der Realisierung des Projektes erfüllt sein sollen/müssen.

- Mitarbeiter:innen können mittels einer Webapplikation und einer grafischen Schnittstelle einen Arbeitsplatz buchen.
- Kollisionen am Arbeitsplatz sollen durch Überbuchungen vermieden werden.
- Die Nutzung der Ressourcen wird transparenter und effizienter.
- Büroausstattungen sollen unkompliziert reserviert werden können.
- Plätze die immer wieder von den gleichen Mitarbeiter:innen gebucht werden, sollen als ihr Favoriten gekennzeichnet werden.
- Der Besprechungsraum und das Lastenfahrrad sollen über die Anwendung unkompliziert gebucht werden können.

### 2.2.3 Funktionalitäten

Die Funktionalität beschreibt die Zweckbestimmung eines Produktes und wird zwischen "Muss-Anforderungen" und "Kann-Anforderungen" unterschieden. Diese werden dann wiederum in funktionale und nicht funktionale -Anforderungen unterteilt.

Funktionale MUSS-Anforderungen:

- Die Anwender:innen müssen sich registrieren, um Zugang zum System zu erhalten.
- Die Anwender:innen müssen einen Arbeitsplatz buchen können.
- Die Anwender:innen müssen die Büroausstattung buchen können.
- Die Anwender:innen müssen ihre eigenen Buchungen ändern können.
- Die Anwender:innen müssen Home-Office als Option auswählen können.
- Die Administrator:innen müssen alle Anwender:innen verwalten können.
- Die Administrator:innen müssen alle Räumlichkeiten verwalten können.

Funktionale KANN-Anforderungen:

- Die Anwender:innen können Arbeitsplätze als Favoriten speichern.
- Die Anwender:innen können ihre Startseite nach Belieben individuell gestalten.

Nicht funktionale MUSS-Anforderungen:

- Das Corporate Design muss eingehalten werden.
- Es muss ein Responsive Design (Wikipedia, 2022) verwendet werden.
- Die Benutzeroberfläche muss intuitiv bedienbar sein.
- Die verwendeten Konzepte müssen DSGVO konform sein.

Nicht-funktionale KANN-Anforderungen:

- Anwender:innen können ihr Profil mit einem Bild versehen.
- Anwender:innen können zwischen einem dunklen- bzw. hellen Modus wechseln.
- Anwender:innen können sich selbst einen (optionalen) Usernamen geben.

## 2.2.4 Projektumfeldanalyse

Bei der Identifizierung der einzelnen Stakeholder wurde ein einfaches Brainstorming-Verfahren (Wikipedia, 2022) angewandt, um die potenziellen Stakeholder zu identifizieren und um sie, entsprechend der Umfeldanalyse, einordnen zu können. Neben der verwendeten Methode beschreibt der folgende Abschnitt auch die Charakterisierung der Stakeholder sowie deren Einstellung, Einfluss und Nähe zum Projekt und auch den zutreffenden Maßnahmen, um deren positive Einstellung zu bewahren oder erzielen.

Um das W.A.B.S visuell ansprechend und dem **Klimabündnis Tirol** entsprechend zu gestalten, stellt das Klimabündnis dem Projektteam das Corporate Designs zur Verfügung.

Das Klimabündnis Tirol, Tochterunternehmen des **Klimabündnisses Österreich**, fungiert als Projektpartner und Auftraggeber. Da die Mitarbeiter:innen des Klimabündnisses Tirol auch nach der Corona-Pandemie entscheiden können, ob sie ihrer Arbeit im Home-Office oder im Büro nachkommen möchten und die Organisation der Arbeitsplatzverwaltung nicht immer einfach ist, ist das Klimabündnis Tirol mit der Idee zur Entwicklung eines Arbeitsplatzbuchungssystems an das IT-Kolleg Imst herangetreten.

Da das **Projektteam** im Rahmen der Ausbildung und als Voraussetzung für den Abschluss eine Diplomarbeit erstellt, wurde auch das **IT-Kolleg Imst** als Stakeholder in Betracht gezogen. Weiters wird der allgemeine Teil der Diplomarbeit im Unterricht SYP erarbeitet.

Im Zuge der Mitarbeiterverwaltung, für Arbeitszeit-Aufzeichnungen und Ähnliches, verwendet das Klimabündnis Tirol eine Software der **Firma ARGE DATA**. Um später die beste Usability zu gewährleisten, soll das W.A.B.S. in die bereits bestehende Software integriert werden können. Dazu muss noch abgeklärt werden, ob lediglich eine Schnittstelle zur Verfügung gestellt werden muss oder ob die Zeitbuchungen der Arbeitsplätze an ARGE DATA übermittelt werden sollen.

Das Projektteam wird von **Herrn Dr. Michael Netzer und Frau Dr. Melanie Osl** betreut. Im Zuge der Ausarbeitung stehen sie für technische und projektspezifische Fragen zur Verfügung.

Um das W.A.B.S. genau für die Anforderungen des Auftraggebers zu erstellen, arbeitet das Projektteam intensiv mit der Ansprechperson beziehungsweise dem **Geschäftsführer des Klimabündnisses Tirol, Herrn Andrä Stigger**, zusammen. Herrn Stigger werden laufend die Fortschritte mitgeteilt und es wird laufend um ein Feedback gebeten.

Auch die schlussendlichen **Anwender:innen** der Applikation werden in die frühe Entwicklung und Testung miteinbezogen. Dies dient insbesondere dazu, bestehende Funktionen zu erweitern, zu optimieren und die Benutzerfreundlichkeit zu steigern.

Im Zuge der Stakeholder Analyse wird die Einstellung, die Einfluss und die Nähe zum Projekt genauer ausgeführt. Die folgende Abbildung 1: Grafische Darstellung der Stakeholder, stellt die Stakeholder in eine visuelle Veranschaulichung. Dabei ist zu verstehen, dass die Kreise bestimmte Eigenschaften aufweisen, welche nun genauer beschrieben werden.

Die Einstellung, also die möglichen Einflüsse, Einstellungen und die Nähe des identifizierten Stakeholders, wird mittels der Farben grün (positiv) und blau (neutral) dargestellt. Die Stakeholder sind unterschiedlich weit von dem Projekt entfernt. Der Einfluss der Stakeholder auf das gesamte Projekt wird durch die Größe der Stakeholder-Kreise dargestellt.

Die Nähe der Stakeholder zum Projekt wird mittels unterschiedlicher Positionierungen der Kreise dargestellt. Diese Positionierung soll dem Leser mitteilen, in welcher Entfernung ein Stakeholder zum Projekt steht. Je weiter entfernt ein Kreis zur Mitte ist, desto weniger gravierende Auswirkungen wird ein Stakeholder auf das Projekt im Falle einer außerplanmäßigen Veränderung haben. Ein naher Kreis symbolisiert hingegen, dass ein Stakeholder einen wichtigen Einfluss auf das Projekt hat.

Im Falle der Projektarbeit bedeutet dies zum Beispiel, dass das Klimabündnis Tirol einen **sehr nahen** Bezug zum Projekt hat, wohingegen das Klimabündnis Österreich einen **entfernteren** Bezug zu dem Projekt bezieht.

Der Grund dafür ist, dass sich die Arbeit derzeit lediglich auf die Tiroler Zweigstelle bezieht und eine österreichweite Integrierung noch nicht feststeht.

Im Gegensatz dazu steht das IT-Kolleg Imst in einem **nahen** Verhältnis zum Projekt, weil das IT-Kolleg Imst die auszubildende Institution darstellt und somit eine primär bürokratische- und keine technische Funktion bietet.

Die Größe der Kreise, die in Abbildung 1: Grafische Darstellung der Stakeholder. zu sehen sind, symbolisieren den individuellen Einfluss auf das Projekt. Je größer der Kreis desto einflussreicher ist die potenzielle Auswirkung auf das Projekt. So würde beispielsweise der Wegfall des Stakeholders „Klimabündnis Tirol“ bedeuten, dass das Projekt ohne Auftraggeber nicht weiter fortbestehen kann.

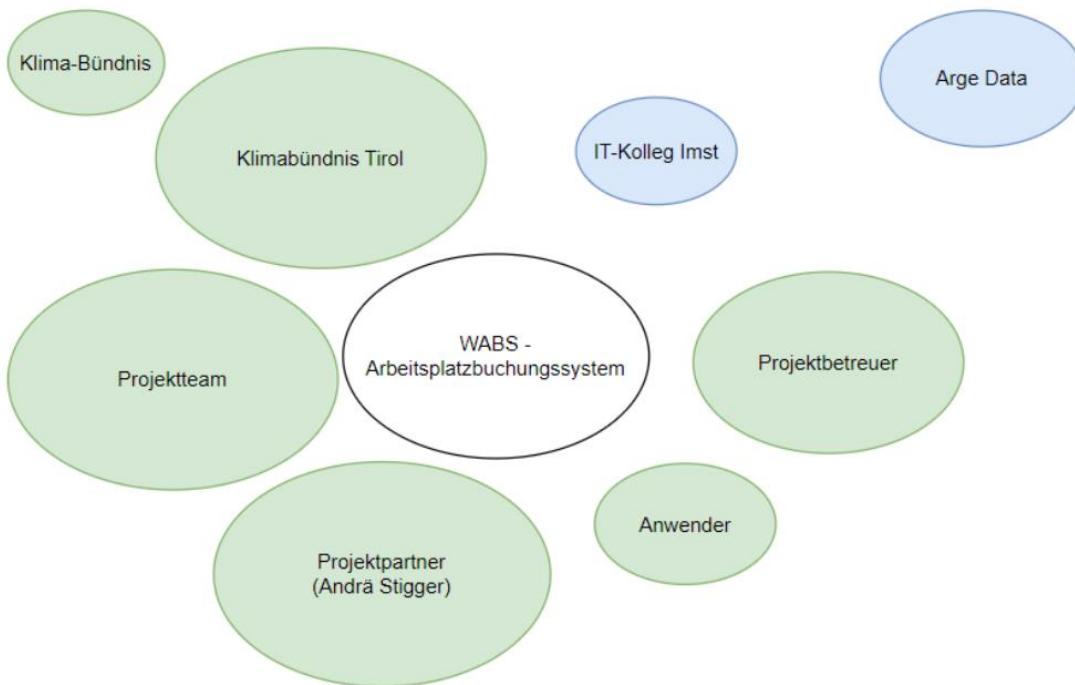


Abbildung 1: Grafische Darstellung der Stakeholder.

Um den Erfolg des Projektes zu fördern, ist eine positive Einstellung der Stakeholder in Bezug auf das Projekt von unvorstellbarer Wichtigkeit. Im Laufe der Zeit können unterschiedliche Ereignisse dazu führen, dass die Einstellung mancher Instanzen sich ändern. Das kann dann unvorhersehbare Schwierigkeiten auslösen, die das Projekt teilweise sehr stark schädigen können. Ein negativer Stakeholder ist immer eine potenzielle Gefahrenquelle, die den Fortschritt der Arbeit stagnieren oder sogar nötig werden lässt.

In der folgenden Tabelle 1: Klassifizierung der Stakeholder tabellarisch dargestellt. Weiters wird die Einstellung anhand farblicher Hervorhebungen ersichtlich.

Diese Farben unterstreichen nochmal zusätzlich, welche potenziellen Auswirkungen von dem einzelnen Stakeholder ausgeht. Die Farbe der jeweiligen Einstellung gibt an, wie der Stakeholder auf das Projekt gestimmt ist. Dabei signalisiert die Farbe Grün eine positive bzw. die Farbe Blau eine neutrale Einstellung.

| Stakeholder                         | Einstellung | Einfluss | Nähe      |
|-------------------------------------|-------------|----------|-----------|
| Klimabündnis Tirol                  | Positiv     | Groß     | Sehr nahe |
| Klimabündnis (Österreich)           | Positiv     | Groß     | Entfernt  |
| IT-Kolleg Imst                      | Neutral     | Mittel   | Nahe      |
| ARGE Data                           | Neutral     | Klein    | Fern      |
| Projektbetreuer                     | Positiv     | Groß     | Sehr nahe |
| Projektpartner (Herr Andrä Stigger) | Positiv     | Groß     | Sehr nahe |
| Anwender                            | Neutral     | Groß     | Sehr nahe |
| Projektteam                         | Positiv     | Groß     | Sehr nahe |

Tabelle 1: Klassifizierung der Stakeholder

Um einer negativen Einstellung entgegen wirken zu können bzw. um bereits positiv gestimmte Stakeholder auch positiv halten zu können, werden entsprechende Maßnahmen gesetzt.

Jene Maßnahmen werden in der folgenden Tabelle 2: Maßnahmen pro Stakeholder, aufgelistet.

| Stakeholder             | Maßnahmen   |
|-------------------------|---|
| Projektteam             | Arbeit wird zeitgerecht aufgeteilt und es wird laufend über den Zustand des Projektes diskutiert.   |
| Klimabündnis Österreich | Einhaltung des Corporate Designs, Erweiterbarkeit des Systems optimieren  |
| Klimabündnis Tirol      | Regelmäßige Informationen zum Projektstatus, strikte Einhaltung der Vorgaben, Wartung und Long-Term-Support (Wikipedia, 2022) kommunizieren |
| IT-Kolleg Imst          | Fristgerechte Ausarbeitung, Abgabe sowie regelmäßiger Austausch des IST-Zustandes   |

|                                     |   |
|-------------------------------------|---|
| ARGE DATA                           | Leichte Anbindungsmöglichkeit einer potenziellen Schnittstelle, um Zeitbuchungen der Arbeitsplätze mit den Buchungen der Arbeitszeiten zu synchronisieren |
| Projektbetreuer                     | Fristgerechte Ausarbeitung, Abgabe sowie regelmäßiger Austausch des IST-Zustands  |
| Projektpartner (Herr Andrä Stigger) | Permanente Einbindung in Informationen/Problemen/Fragen des Projektes   |
| Anwender                            | Fristgerechte Ausarbeitung, Abgabe sowie regelmäßiger Austausch des IST-Zustandes   |

Tabelle 2: Maßnahmen pro Stakeholder

## 2.3 Problemanalyse

Die Problemanalyse beschäftigt sich mit dem Problem und der damit verbundenen Ursache und wie passende Maßnahmen gefunden werden, um der Problematik entgegen wirken zu können. Diese Analyse wurde in sieben Use-Cases (Microtool.de, 2022) unterteilt und sie werden im Absatz „Use-Case Diagramm“ genauer beschrieben.

### 2.3.1 User-Stories

„Eine User Story ist eine informelle, allgemeine Erklärung eines Software-Features, die aus der Sicht des Endbenutzers verfasst wurde“ (Atlassian, 2022).

Im Zuge der Problemanalyse haben sich insgesamt Neun User-Stories ergeben, welche im folgenden Abschnitt aufgelistet werden.:

- Als Anwender:in möchte ich alle meine Buchungen ändern beziehungsweise stornieren können.
- Als Anwender:in möchte ich, über einen gesicherten Zugang, für ein bestimmten Zeitraum, sehen welche Büroausstattung zur Verfügung stehen und diese auch buchen können.
- Als Anwender:in möchte ich, über einen gesicherten Zugang, für einen bestimmten Zeitraum, sehen ob das Lastenfahrrad zur Verfügung steht und dieses auch buchen können.
- Als Anwender:in möchte ich, über einen gesicherten Zugang, sehen welche Arbeitsplätze, für einen bestimmten Zeitraum zur Verfügung stehen und diese auch buchen können.
- Als Administrator:in möchte ich sicherstellen, dass Mitarbeiter:innen die Zeitbuchungen anderer Mitarbeiter:innen nicht verändern bzw. stornieren können.
- Als Administrator:in möchte ich eine einfache Übersicht über die verliehenen Ressourcen (inkl. Anzahl und Status) haben.
- Als Administrator:in möchte ich die Anordnung und die Eigenschaften (z.B.: Höhenverstellbarkeit des Tisches, Anzahl und Typ der Schnittstellen) der Arbeitsplätze innerhalb der Räumlichkeiten verwalten können.
- Als Administrator:in möchte ich den Grundriss der Räumlichkeiten unkompliziert selbst verwalten können.
- Als Administrator:in möchte ich die Mitarbeiter in der Datenbank verwalten können.

### Use-Case Diagramm

Das Use-Case-Diagramm beschreibt die Beziehungen zwischen den Akteuren und dem System. Dabei ist zu beachten, dass die Administrator:innen im Gegensatz zum Anwender, mehr Funktionalitäten benötigen. Diese Funktionalitäten beschäftigen sich primär mit der Verwaltung menschlicher sowie materiellen Ressourcen. In Abbildung 2: Use Case Diagramm, werden diese Beziehungen bzw. Funktionalitäten grafisch dargestellt.

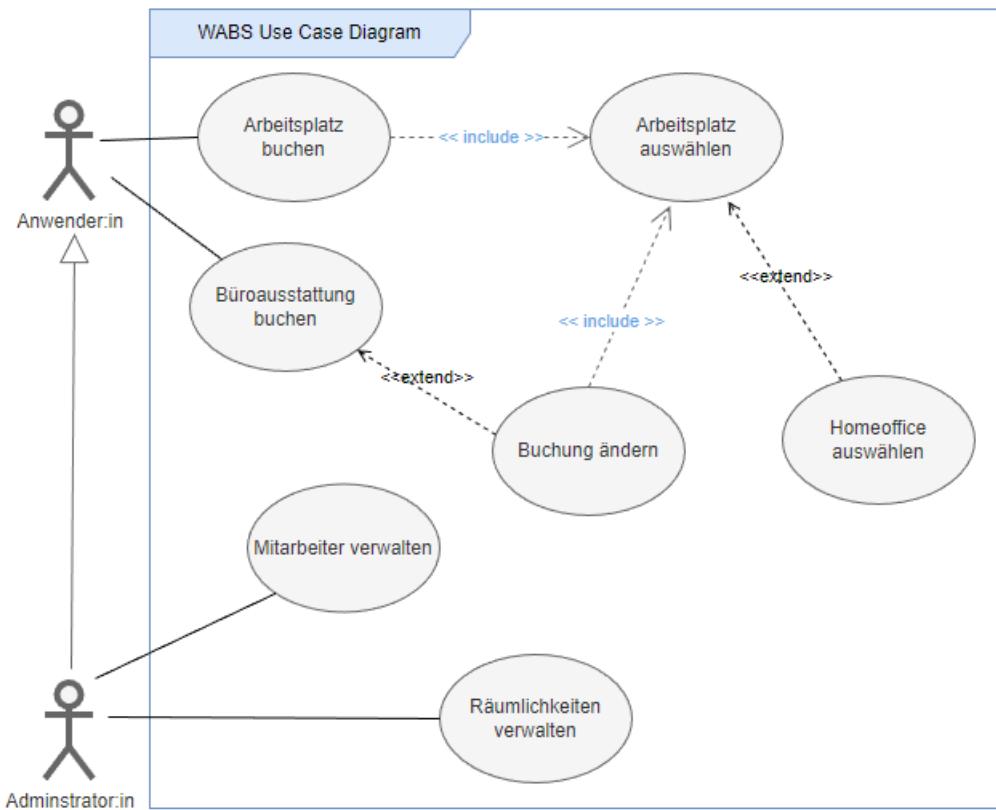


Abbildung 2: Use Case Diagramm

Eine genauere Beschreibung der Use-Cases folgt in den nachstehenden Tabellen. Dabei wurden drei spezifische Benutzerfälle aufgelistet, auf die näher eingegangen wird:

So wird in der Tabelle 3: Use Case - Arbeitsplatz buchen, beschrieben, dass die Anwender:innen eine Funktionalität wollen, die ihnen eine Buchung eines Arbeitsplatzes ermöglicht.

|                     |   |
|---------------------|---|
| Use-Case Nummer     | UC-WABS-1   |
| Kurze Beschreibung  | Einen Arbeitsplatz buchen.  |
| Akteure             | Anwender:in   |
| Auslöser            | Anwender:in will einen Arbeitsplatz im Büro buchen.   |
| Vorbedingungen      | Verfügbarer Arbeitsplatz, autorisierter Account.  |
| Standardablauf      | Anwender:in meldet sich an, wählt für einen Tag den gewünschten Arbeitsplatz und reserviert ihn für einen bestimmten Zeitraum.                            |
| Alternative Abläufe | Anwender:in meldet sich an, wählt einen Arbeitsplatz aus der Favoritenliste und bucht diesen für einen oder mehrere Tage bzw. für eine bestimmte Periode. |

Tabelle 3: Use Case - Arbeitsplatz buchen

Tabelle 4: Use Case - Mitarbeiter:in Konto anlegen, beschreibt, dass die Administrator:innen in der Lage sein müssen, neue Konten in das System einbetten zu können.

|                     |  |
|---------------------|--|
| Use-Case Nummer     | UC-WABS-2  |
| Kurze Beschreibung  | Neues Mitarbeiter:innen Konto anlegen.   |
| Akteure             | Administrator:in   |
| Auslöser            | Neue/r Mitarbeiter:in muss angelegt werden.  |
| Vorbedingungen      | Mitarbeiter:in existiert noch nicht im System  |
| Standardablauf      | <ul style="list-style-type: none"> <li>• Administrator:in legt neues Konto an.</li> <li>• System schickt Bestätigungsemail an Mitarbeiter:in und an Administrator:in.</li> </ul> |
| Alternative Abläufe | <ul style="list-style-type: none"> <li>• Mitarbeiter:in legt Benutzerkonto an.</li> <li>• Das Konto wird von Administrator:in bestätigt.</li> </ul>                              |

Tabelle 4: Use Case - Mitarbeiter:in Konto anlegen

Ebenso müssen die Administrator:innen in der Lage sein, die Räumlichkeiten ändern zu können. Dieser Fall wird in Tabelle 5: Änderung der Büroräumlichkeiten genauer beschrieben.

|                     |  |
|---------------------|--|
| Use-Case Nummer     | UC-WABS-3  |
| Kurze Beschreibung  | Räumlichkeiten ändern.   |
| Akteure             | Administrator:in   |
| Auslöser            | Umzug oder Änderung der Büroräumlichkeiten.                          |
| Vorbedingungen      | Erforderliche Änderung der Arbeitsplätze.                            |
| Standardablauf      | Administrator:in meldet sich an und erstellt/modifiziert einen Raum. |
| Alternative Abläufe | Administrator:in meldet sich an und löscht einen Raum.               |

Tabelle 5: Änderung der Büroräumlichkeiten

## 2.4 Planung

Im folgenden Abschnitt wird auf die Planung der Struktur sowie auf den Ablauf eingegangen. Dazu wurde ein Projektstruktur- als auch ein Projektablaufplan erstellt, welche im darunterliegenden Abschnitt genauer eruiert werden.

### 2.4.1 Projektstrukturplan

In der nachstehenden Abbildung 3: Projektstrukturplan, wird der Projektstrukturplan der Diplomarbeit dargestellt. Dabei wurde die Umsetzung nach dem Prinzip des „Teilen und Herrschen“ in Arbeitspakete/Module unterteilt. Aus der Analyse haben sich folgende Module ergeben:

- Projektmanagement beschäftigt sich mit der Systemplanung und umfasst die Zielformulierung, Anforderungs-, Umfeld-, Problem-, Risikoanalyse sowie die Projektplanung.
- Ressourcenverwaltung, Mitarbeiterverwaltung, Räumlichkeiten beinhalten die Konzipierung der Datenentwicklung, die Backend-, UI-Entwicklung sowie Testung und Optimierung.
- Layout und Design beinhaltet die Umsetzung des Corporate Designs. Ebenfalls wird eine Dark/Light-Theme als auch eine Internationalisierung Implementierung, im Sinne der Barrierefreiheit, als Sinnvoll betrachtet. Da die Anwendung auf unterschiedlichen Endgeräten dargestellt wird, wird darauf geachtet, dass das Responsive Design eingehalten wird.
- Zusammenführung, Testung, Optimierung beinhaltet die Zusammenführung der Module und die Optimierung/Testung des Systems.

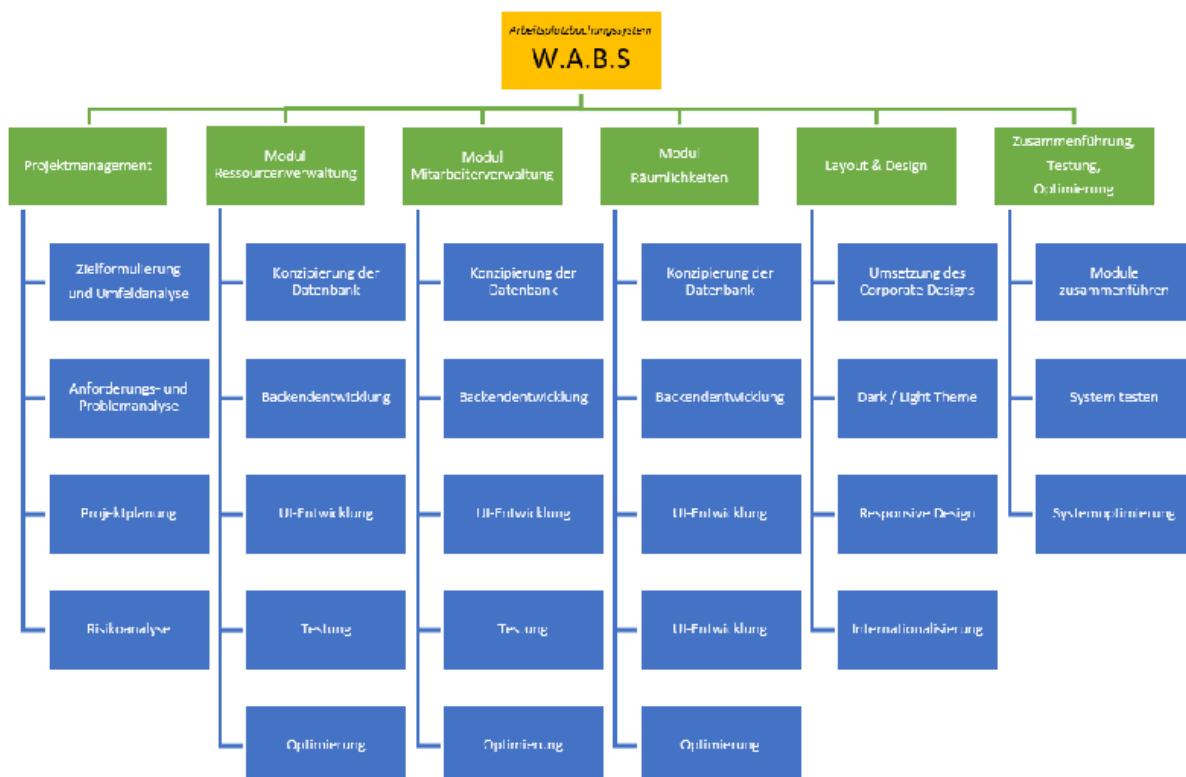
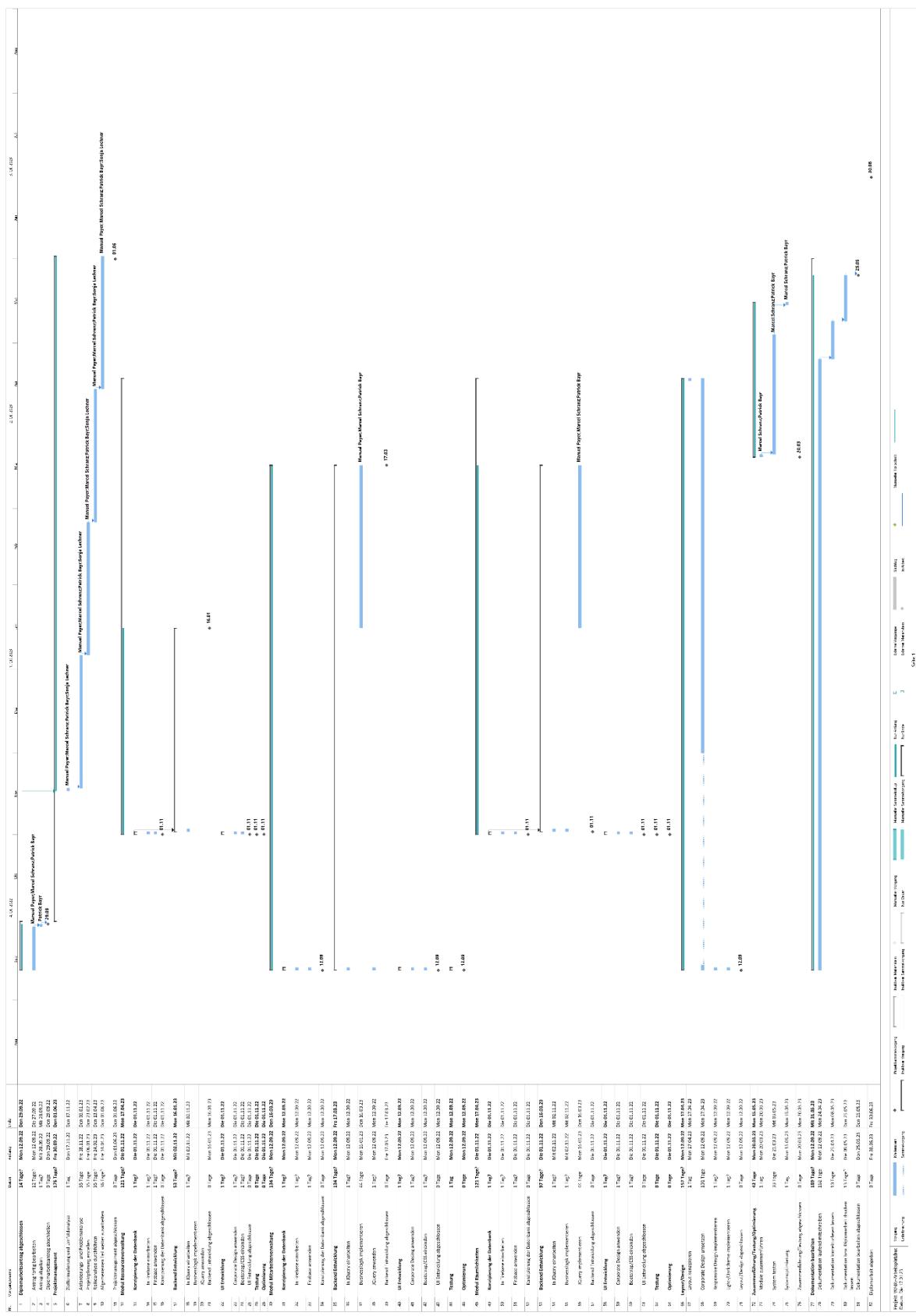


Abbildung 3: Projektstrukturplan

## 2.4.2 Projektablaufplan

Um den Projektablauf verständlicher darzustellen, wurden die einzelnen Module sowie die jeweils dazugehörigen Arbeitspakete in einem „Gantt-Chart“ visuell dargestellt. Dabei ist zu beachten, dass die Module und die Arbeitspakete jeweils mit einer Zeitspanne versehen wurde, in welcher die Arbeitspakete erledigt werden sollen. Jene Visuelle Darstellung wird im folgenden Abschnitt dargestellt.



*Abbildung 4: Gantt Chart*

## 2.5 Risikoanalyse

Im Laufe der Projektentwicklung können diverse Risiken auftreten, welche sich unterschiedlich auf bestimmte Aspekte oder sogar auf das ganze Projekt auswirken können. Diese Auswirkungen können den Projektfortschritt nicht nur stagnieren lassen, sondern auch zurückwerfen oder im schlimmsten Fall, das gesamte Projekt zu Nichte machen.

Um solchen Gefahren entgegenwirken zu können, werden im Zuge der Risikoanalyse die möglichen Risiken identifiziert, bewertet und entsprechende Gegenmaßnahmen gesetzt. Die Bewertung der Risiken und deren Auswirkung werden im Risikoportfolio hinterlegt, kategorisiert und mit einer Bewertung, welche sich aus der Multiplikation der Eintrittswahrscheinlichkeit (skaliert in einem Bereich zwischen X und Y) und der Auswirkung ergibt, hinterlegt. Daraus ergibt sich das Risikopotenzial.

Eine genaue Beschreibung der einzelnen Risiken und deren Einstufung, folgt in der Tabelle 6: Risikoportfolio. Es ist zu beachten, dass ein Risiko mit einer Nummer, einem Status, einer Kategorie, einem Titel, den Folgen, der Eintrittswahrscheinlichkeit sowie einem Ausmaß versehen wird. Der Status gibt an, ob ein Risiko überwacht, eliminiert oder eingetreten ist. Weiters wird das jeweilige Risiko in eine Kategorie untergeordnet, wobei man hier man zwischen technischer-/produktbezogener-, wirtschaftlicher-, politischer-, menschlicher-/kultureller Natur unterscheidet.

Im Anschluss wurden die Risiken betitelt und es wurde die Eintrittswahrscheinlichkeit (E) sowie die Auswirkung (A) eruiert. Es wurden jeweils Werte von eins bis vier hinterlegt, welche sich miteinander multiplizieren. Das Produkt der Berechnung ist das Gefahrenpotential. Je höher der Wert, desto größer ist das Potential einer Gefahr. Im Umkehrschluss bedeutet das ebenfalls, dass je niedriger das Produkt der Berechnung ist, desto geringer ist die Gefahr.

Im nächsten Schritt wurden Gegenmaßnahmen ermittelt, welche monetärer Natur sein können. Wenn dem so ist, werden die Kosten zur Prävention der Gefahr, hinterlegt.

| Nr. | Status    | Kategorie      | Titel  | Folgen  | E | Potential | Gegenmaßnahme  |
|-----|-----------|----------------|--|---|---|-----------|--|
| 1   | Überwacht | Menschlich     | Projektpartner Ausfall   | Das Projekt kann zwar erarbeitet, aber nicht mehr in der vorgesehenen betrieblichen Umgebung realisiert werden.       | 1 | 2         | Regelmäßige Team-Meetings sowie Gespräche mit dem Projektpartner, um sicherzustellen, dass Anforderungen erfüllt und ihre Bedenken gelöst bzw. Fragen beantwortet werden.  |
| 2   | Überwacht | Menschlich     | Ausfall eines Projektmitgliedes  | Teil des Projekts kann nicht mehr fertiggestellt werden bzw. der Arbeitsaufwand der restlichen Teammitglieder steigt. | 1 | 2         | Regelmäßige Kommunikation und gegenseitige Unterstützung innerhalb des Teams, um Präventionen zu schaffen. Zudem wird sichergestellt, dass sich alle Teammitglieder mit den jeweils anderen Modulen des Projekts auskennen, um im Notfall ohne allzu großen Aufwand diese auch übernehmen zu können. |
| 3   | Überwacht | Politisch      | Compliance Anforderung des DSGVO ändert sich   | Das Datenschutzkonzept muss kurzfristig geändert werden   | 1 | 1         | Software erweiterbar/wartbar gestalten   |
| 4   | Überwacht | Technisch      | Entstehende Projekt-Verzögerungen durch unvorhersehbare Ereignisse                         | Fristgerechtes Einhalten der Meilensteine schwer bis kaum möglich   | 2 | 6         | Regelmäßige Scrum-Meetings sowie Gespräche mit dem Projektpartner und den Projektbetreuern - dadurch ist man besser auf unvorhersehbare Ereignisse vorbereitet und kann auf diese schneller reagieren.   |
| 5   | Überwacht | Wirtschaftlich | Kostenerhöhung des Projekts bedingt durch Preisänderungen benötigter Drittanbietersoftware | Kosten-Nutzen Verhältnis rentiert sich für Auftraggeber nicht mehr  | 1 | 2         | Vorab alle möglichen Kostenfaktoren analysieren und bereits während der Planung einen Puffer im Budget einplanen.  |

|   |           |           |   |  |   |   |   |
|---|-----------|-----------|---|--|---|---|---|
| 6 | Überwacht | Technisch | Drittanbieter-Software oder Dienste Ausfall                       | Geplanter Projektablauf muss kurzfristig angepasst werden  | 1 | 3 | Alternativen suchen und sich mit diesen bereits vorher auseinandersetzen  |
| 7 | Überwacht | Technisch | Verlust notwendiger Daten   | Projekt muss neu konzipiert werden. Zeitdruck, Potenzial fehlerhafter Implementierungen und die Wahrscheinlichkeit, wichtige Elemente zu vergessen, steigen. | 1 | 4 | Bereits vor Start der Implementierung ein solides Sicherungskonzept erstellen - regelmäßige Backups und redundante Sicherung auf mehreren Medien. |
| 8 | Überwacht | Technisch | Fehler in der Implementierung von Businesslogik und sonstige Bugs | Software funktioniert nicht so wie geplant   | 1 | 3 | Während der Entwicklung des Projekts stetiges testen, optimieren und absprechen mit Auftraggeber  |

Tabelle 6: Risikoportfolio

### 2.5.1 Risikomatrix

Aus dem Risikoportfolio wurde im Anschluss eine Risikomatrix erstellt, wie in Abbildung 5: Risikomatrix zu sehen ist, welche eine grafische Darstellung der einzelnen Risiken innerhalb einer Matrix darstellt. Dabei ist zu beachten, dass ein Kreis jeweils ein eruiertes Risiko ist und sich auf den X-/ Y-Achsen befindet, welche Auskunft über die Auswirkung (x-Achse) sowie über die Eintrittswahrscheinlichkeit (Y-Achse) liefert.

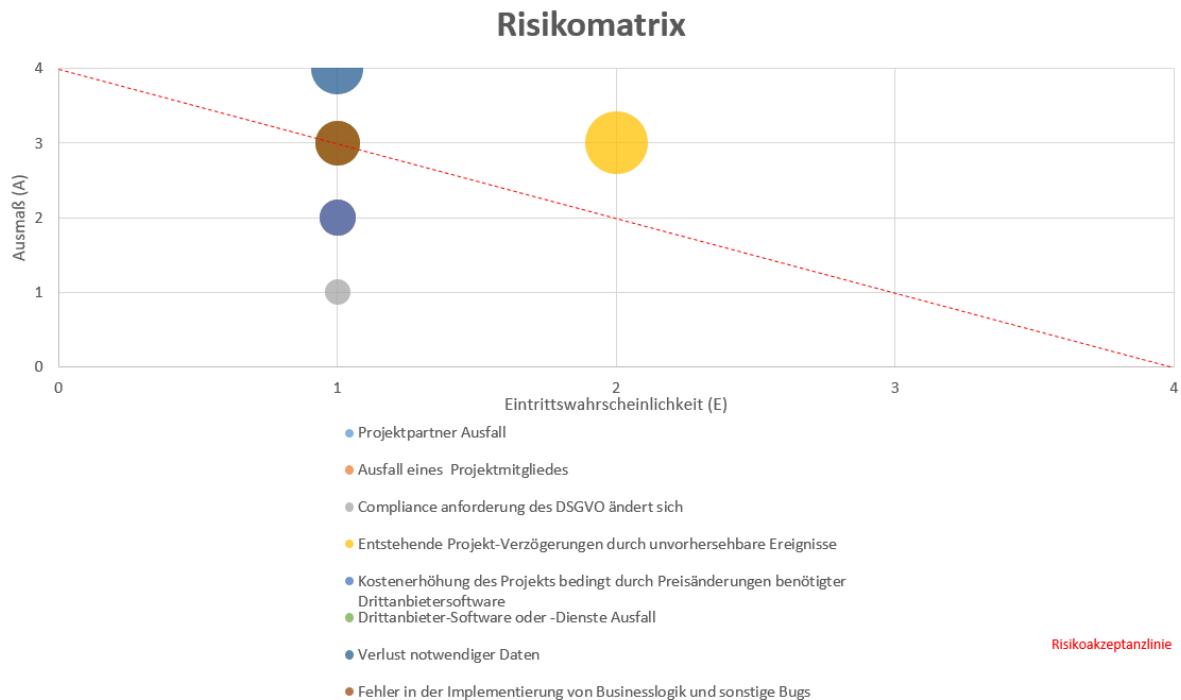


Abbildung 5: Risikomatrix

Der Nachteil dieser Darstellung ist, dass Risiken, die das selbe Risikopotential miteinander teilen, sich gegenseitig überdecken. Damit es klarer ersichtlich ist, sollte die nachstehende Abbildung zur Beurteilung hinzugezogen werden.

Die zweite Grafik, Abbildung 6: Risikomatrix v2., stellt dasselbe Prinzip, welches oben umgesetzt wurde, in einer manuell erstellten Matrix dar. Hier werden die Risiken mit ihrer Nummer in Kästchen geschrieben, welche sich von ihrer Farbe sowie ihrer Position unterscheiden. Dabei ist zu beachten, dass Risiken, welche sich im roten Bereich befinden, ein sehr hohes Risikopotential besitzen. Jene, die sich im gelben Bereich befinden, besitzen ein mittleres Gefahrenausmaß sowie eine mittlere Eintrittswahrscheinlichkeit. Je weiter sich ein Risiko im oben rechten Bereich befindet, desto bedrohlicher könnte es sein.

|            |        | hoch        |        |      |                                    |
|------------|--------|-------------|--------|------|------------------------------------|
|            | hoch   | R7          |        |      |                                    |
| Auswirkung | mittel | R6, R8      | R4     |      |                                    |
|            | mittel | R1, R2, R5, |        |      |                                    |
|            | tief   | R3          |        |      |                                    |
|            |        | gering      | mittel | hoch |                                    |
|            |        |             |        |      | <b>Eintrittswahrscheinlichkeit</b> |

Abbildung 6: Risikomatrix v2.

# 3 Systemdokumentation

Der folgende Abschnitt beschäftigt sich mit der Systemdokumentation. Dabei wird auf die angewandten Technologien und Werkzeuge sowie auf den Systementwurf eingegangen.

## 3.1 Generelle Systemübersicht

Im Folgenden wird eine Beschreibung der Systemkomponenten gegeben, einschließlich der physischen und logischen Komponenten sowie der Klassen, die im aktuellen Zustand des Produkts verwendet werden.

Die Code-Struktur wurde mithilfe des Schichtenmodells aufgebaut, das auf dem MVC-Muster (Model-View-Controller) basiert. Diese Struktur unterteilt die Anwendung in drei logische Schichten, was eine gängige Architektur für Full-Stack-Anwendungen ist und zahlreiche Vorteile bietet.

Die Klassen sind in die folgenden Schichten unterteilt:

- Domain Layer: Hier werden die Klassen definiert, die die Geschäftslogik und die Datenrepräsentation der Anwendung beinhalten. Dies ist die Herzstück der Anwendung.
- Service Layer: In dieser Schicht befinden sich die Klassen, die die Geschäftslogik steuern und verschiedene Dienste für die Anwendungslogik bereitstellen.
- Repository Layer: Hier werden die Klassen definiert, die für die Datenbankzugriffe und die Datenbankkommunikation verantwortlich sind.

Diese Schichtenstruktur wurde ausgewählt, da sie eine bewährte Methode in der Welt der Full-Stack-Anwendungen ist und dazu beiträgt, den Code übersichtlicher und wartungsfreundlicher zu gestalten."

Ziel ist es, dass die Entwicklung und Wartung des Codes durch klare Zuständigkeiten definiert sind und Änderungen nur minimale Auswirkungen auf anderen Layer haben. Im Sinne der Wartbarkeit ist dies eine sehr gute Vorgehensweise.

Wie in der nachstehenden Abbildung 7: Schichtenstruktur zu sehen ist, existiert neben den oben genannten Schichten noch ein Exception (docs.oracle, 2023) Layer (Ausnahmen Schicht) sowie ein REST-Controller-Layer (baeldung, 2023) (Representational State Transfer).

Der Exception Layer beinhaltet dabei sogenannte Exception Classes, die dazu dienen, im Falle eines Fehlers, wie zum Beispiel einer nicht funktionierenden Verbindung zur Datenbank, eine entsprechende Exception zu werfen, die im späteren Verlauf helfen soll, Fehlerquellen schneller zu orten.

Das REST-Control-Layer kümmert sich währenddessen um entgegengenommene HTTP- Anfragen (Hypertext Transfer Protocol) (Wikipedia, 2023) und liefert dementsprechend eine Antwort zurück. Der Controller selbst verwendet die Service-Klassen, um die Business-Logik zu implementieren und Daten aus der Datenbank zu holen oder zu aktualisieren.

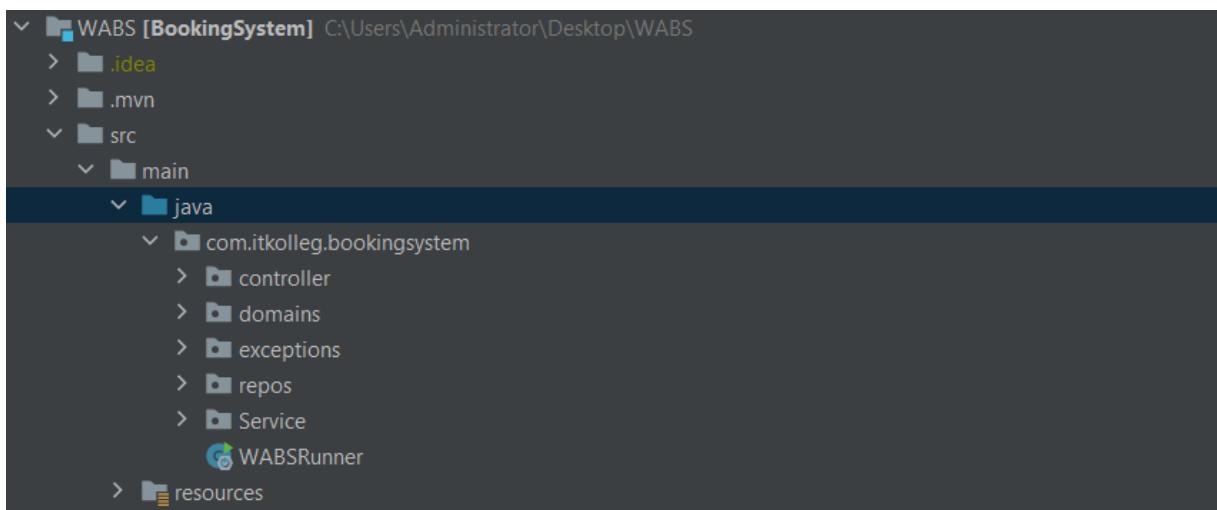


Abbildung 7: Schichtenstruktur

Eine ausführlichere Beschreibung der oben genannten Technologien lässt sich im Literaturverzeichnis finden.

Diese Technologien, Strukturen und Architekturen bilden gemeinsam das Backend (computerweekly, 2023) der Anwendung. Das Frontend (computerweekly, 2023) wird mithilfe von Thymeleaf (thymeleaf.org, 2023), einer serverseitigen Java Template-Engine, erstellt. Da Thymeleaf hervorragend mit Spring MVC, Spring Boot und anderen Spring-Modulen harmoniert, hat man sich für eben jene Technologie entschieden.

Mit Thymeleaf können Entwickler serverseitige HTML-Vorlagen (Hypertext Markup Language) (de.ryte.com, 2023) erstellen, die dynamisch mit Daten gefüllt werden können. Das bedeutet, dass eine Vorlage mit Platzhaltern für Daten erstellt werden kann und dass diese Platzhalter zur Laufzeit durch tatsächliche Daten aus der Datenbank oder einem anderen Backend-System ersetzt werden können.

### 3.1.1 Verwendete Technologien und Werkzeuge

Bei der Entwicklung der Webanwendung wurde auf verschiedene Programmiersprachen zurückgegriffen. Der Großteil der Sprache ist Java und JavaScript (Wikipedia, 2023), während für die Frontend-Darstellung HTML5 und CSS (Cascading Style Sheet) (Wikipedia, 2023) eingesetzt werden, die sich um die visuelle Darstellung der Anwendung kümmern.

Neben den genannten Sprachen wurden auch andere Technologien verwendet, die im nachstehenden Absatz aufgelistet werden. Wie bereits in der generellen Systemübersicht erwähnt, wird die Anwendung unter Verwendung von Java Spring, Thymeleaf und MVC-Modell als Technologien erstellt. Zusätzlich dazu sendet der REST-Controller Antworten in Form von JSON-Objekten (Wikipedia, 2023) an den Client, die er dann interpretieren und verarbeiten kann.

Durch die Verwendung von JQUERY UI (jqueryui, 2023) und svg.js (svgjs.dev, 2023), die beide JavaScript-Bibliotheken sind, werden der Anwendung weitere Funktionen hinzugefügt. Der Projektpartner wünschte sich, dass der Grundriss der Räumlichkeiten inklusive der Arbeitsplätze visuell dargestellt wird, um im Falle eines Umzugs jederzeit einen neuen Grundriss selbstständig zeichnen und

in die Webseite einbetten zu können. Diese Funktionalität wird durch die oben erwähnten JavaScript-Bibliotheken ergänzt.

Die Anwendung wurde mithilfe eines Build-Management-Tools entwickelt. Dieses Tool bietet eine einheitliche Möglichkeit, Abhängigkeiten zu verwalten, den Build-Prozess zu automatisieren und die Bereitstellung von Java-Software zu vereinfachen. Für dieses Tool wurde Maven hinzugezogen, da es möglich ist, mit Hilfe von Maven schnell und einfach Java-Projekte zu erstellen, indem die Projektinformationen und -abhängigkeiten in einer einzigen Konfigurationsdatei (pom.xml) angegeben werden. Maven verwendet diese Informationen, um den Build-Prozess zu automatisieren, was die Entwicklungszeit verkürzt und die Konsistenz im Code erhöht. Maven arbeitet mit sogenannten Abhängigkeiten (engl. Dependencies). Diese Abhängigkeiten werden von Maven in der pom.xml interpretiert, heruntergeladen und verwaltet. In der nachstehenden Abbildung 8: Dependencies in der pom.xml, sind die eingebundenen Abhängigkeiten zu sehen.

```
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Abbildung 8: Dependencies in der pom.xml

### 3.1.2 Logischer Aufbau

Die Kernklasse des Systems ist die abstrakte Klasse "Booking", die drei konkreten Unterklassen hat: "DeskBooking", "RoomBooking" und "EquipmentBooking". Jede Buchung hat eine eindeutige ID und ist einem Mitarbeiter zugeordnet. Eine Buchung hat einen Start- und einen Endzeitpunkt sowie eine Dauer.

Die Klasse "Employee" repräsentiert einen Mitarbeiter im System und hat eine eindeutige ID sowie einen Namen und eine E-Mail-Adresse.

Die Klasse "Desk" repräsentiert einen Schreibtisch im System und hat eine eindeutige ID sowie eine Schreibtischnummer und die Anzahl der Monitore, die an den Schreibtisch angeschlossen sind. Ein Schreibtisch kann eine Liste von Ports haben, die von Geräten genutzt werden können.

Die Klasse "Port" repräsentiert einen Anschluss an einem Schreibtisch und hat eine eindeutige ID sowie eine Beschreibung, welches Gerät an diesem Anschluss angeschlossen werden kann.

Die Klasse "Equipment" repräsentiert ein Gerät im System und hat eine eindeutige ID sowie einen Namen und eine Beschreibung.

Die Klasse "Room" repräsentiert einen Raum im System und hat eine eindeutige ID sowie eine Raumnummer, eine Liste von Vertices, Floor, Info und eine Liste von Desks, also die maximale Anzahl von Arbeitsplätzen, die den Raum beinhalten können.

Es gibt eine Service-Schicht im System, die für die Geschäftslogik verantwortlich ist. Jede Klasse hat ihre eigene Service-Klasse, z.B. "BookingService", "EmployeeService", usw.

Es gibt eine Controller-Schicht, diese Ebene behandelt die Belange der Benutzeroberfläche, wie z. B. das Anzeigen von Daten und das Akzeptieren von Benutzereingaben.

Es gibt eine Repository-Schicht im System, die für den Zugriff auf die Datenbank verantwortlich ist. Jede Klasse hat ihre DBAccess-Klasse, z.B. "DBAccessBooking", "DBAccessEmployee", usw.

### 3.1.3 Physischer Aufbau

Dieses Komponentendiagramm, wie in Abbildung 9: Komponentendiagramm zu sehen ist, zeigt das Work Area Booking System (WABS) auf der technischen Ebene. Das WABS ist eine Webanwendung, mit der Mitarbeiter Arbeitsplätze, Ressourcen oder Räume buchen können.

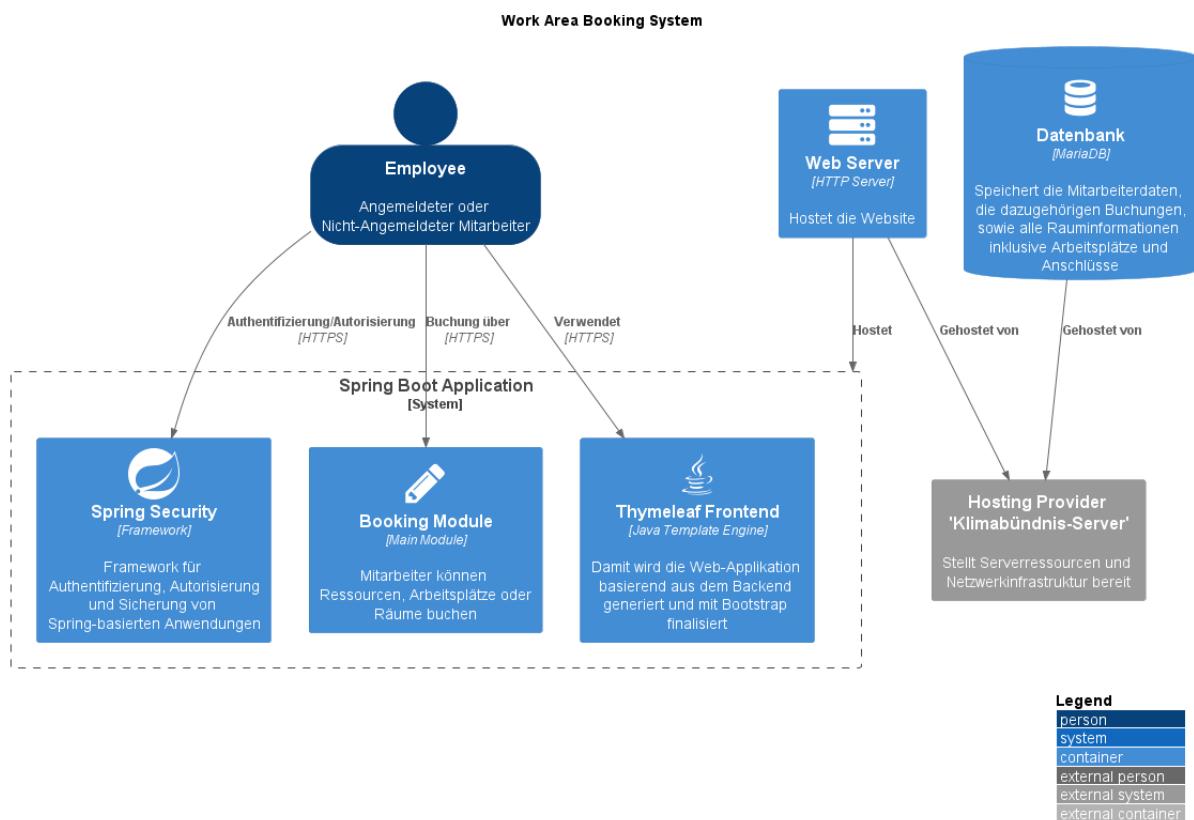


Abbildung 9: Komponentendiagramm

Die Architektur basiert auf einer Spring-Boot-Anwendung, die aus mehreren Containern besteht. Nachfolgend werden die wichtigsten Komponenten genauer beschrieben.

Der erste Container ist das Spring-Security-Framework, das für die Authentifizierung, Autorisierung und Sicherheit der Anwendung zuständig ist.

Der zweite Container ist das Booking-Modul, das die Kernfunktionalität der Anwendung darstellt. Damit können eingeloggte Mitarbeiter Arbeitsplätze, Ressourcen und Ressourcen für einen bestimmten Zeitraum buchen.

Der dritte Container ist das Thymeleaf-Frontend, das als Java-Template-Engine verwendet wird, um die Webanwendung zu generieren und mit Bootstrap (getbootstrap.com, 2023) zu finalisieren.

Die Webanwendung wird auf einem Webserver gehostet. Die Daten der Mitarbeiter, ihre Buchungen sowie alle Raumdaten einschließlich Arbeitsplätze und deren Anschlüsse werden in einer MariaDB-Datenbank gespeichert.

### 3.1.4 Konzept für Ausnahmebehandlung

Fehler in den Eingaben werden durch ausgiebige Validierung geprüft und an den Anwender gemeldet, damit er weiß, worin der Fehler lag. Führt eine Funktion eine CRUD-Operation (mindsquare.de, 2023) auf der Datenbank durch, wird diese ebenfalls auf ihre Gültigkeit geprüft. Wird beispielsweise versucht einen Mitarbeiter mit bereits erstellten Buchungen zu löschen, würde dies von der Datenbank selbst, durch entsprechende Regeln, verhindert werden. Spring Boot fängt die Abfrage jedoch schon im Vorhinein ab und gibt eine entsprechende Fehlermeldung aus. Für ein gut funktionierendes Exception Handling wurden somit für alle spezifischen Exceptions Controller erstellt. Damit wird der Anwender immer genau informiert und bekommt keine kryptischen Fehlermeldungen bezüglich der Datenbank zu sehen.

### 3.1.5 Klassen und Komponenten

Die Klassen- und Komponentenstruktur beschreibt die verschiedenen Klassen und Komponenten, die im System vorhanden sind, sowie deren Eigenschaften, Methoden und Beziehungen. In diesem Abschnitt der Systemdokumentation werden die verschiedenen Klassen und Komponenten des Systems sowie deren Beziehungen und Abhängigkeiten beschrieben, um eine umfassende Übersicht über die Struktur des Systems zu geben.

Das Work-Area Booking System besteht aus verschiedenen Klassen und Komponenten, die in einer Hierarchie angeordnet sind. Die Hauptklasse ist die Booking-Klasse, von der die konkreten Klassen DeskBooking, RoomBooking und EquipmentBooking abgeleitet sind. Jede dieser Klassen hat ihre eigenen Methoden und Eigenschaften, die es ermöglichen, Buchungen für bestimmte Ressourcen zu verwalten.

Die Employee-Klasse ist eng mit der Booking-Klasse verbunden und stellt die Beziehung zwischen dem Mitarbeiter und der Buchung her. Die Desk- und Equipment-Klassen repräsentieren die physischen Ressourcen, die für Buchungen zur Verfügung stehen, während die Room-Klasse die Räume darstellt, die für Buchungen auch reserviert werden können.

Es gibt auch andere Klassen wie Port und Role, die für die Verwaltung des Systems notwendig sind. Das Enum Role ermöglicht das Verteilen der Zugriffsrechte in Vier Kategorien, „Admin“, „Operator“, „Privileged User“ und „Normal User“. Diese Klassen haben Beziehungen und Abhängigkeiten untereinander, um sicherzustellen, dass das System reibungslos funktioniert.

Die Methoden und Eigenschaften der Klassen ermöglichen es dem System, verschiedene Aufgaben auszuführen, wie zum Beispiel die Buchung von Ressourcen, die Verwaltung von Benutzern und die Verwaltung von Buchungen. Die Beziehungen zwischen den Klassen stellen sicher, dass das System effektiv und effizient arbeitet, indem sie die Interaktion zwischen den Komponenten regeln.

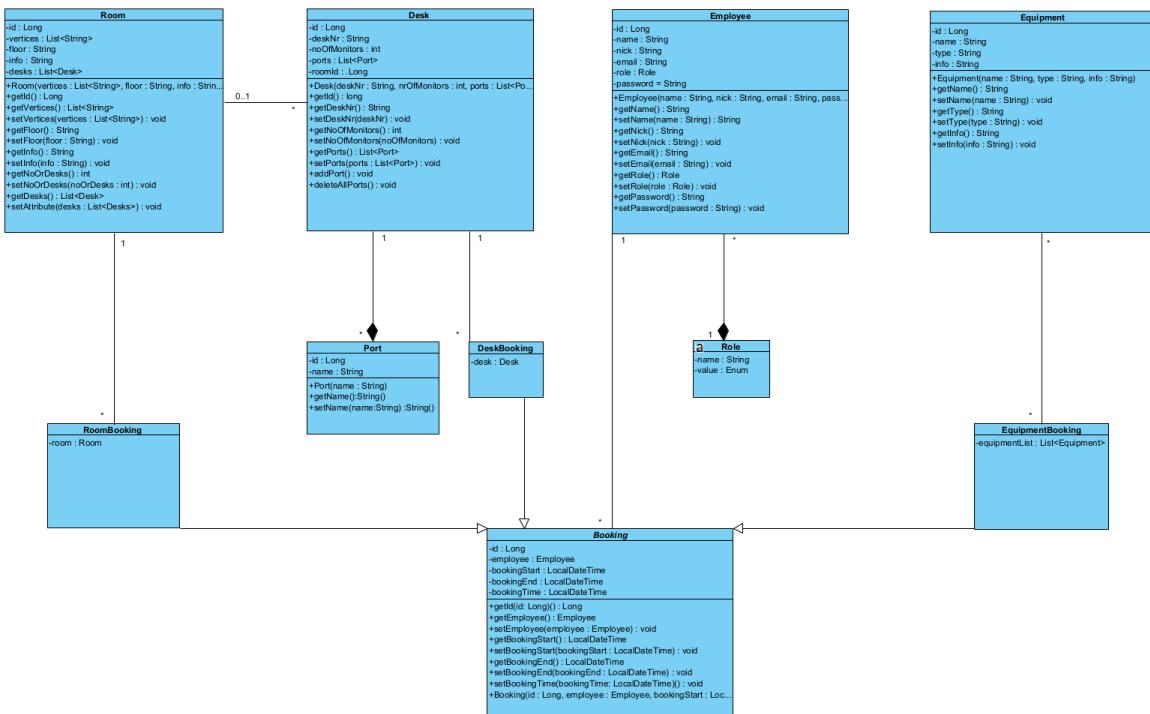


Abbildung 10: ER-Diagramm

### 3.2 Systematische Literatursuche

Für die Durchführung einer systematischen Literatursuche nach alternativen Programmen mit einer ähnlichen Funktionsweise wie die W.A.B.S.-Applikation wurde Google Scholar genutzt. Es wurden verschiedene Suchbegriffe wie "Webapplikation", "Spring Framework", "Hotelzimmerbuchungssystem" und "Raumplaner" kombiniert und die Suche wurde auf Artikel beschränkt, die innerhalb der letzten fünf Jahre veröffentlicht wurden.

Die Ergebnisse der Suche offenbarten eine Vielzahl von ähnlichen Applikationen, die von unterschiedlichen Entwicklern und Unternehmen erstellt wurden. Diese Applikationen umfassten Raumplaner, Hotelzimmerbuchungssysteme und andere webbasierte Anwendungen, welche ähnliche Funktionalitäten wie die W.A.B.S.-Applikation anbieten.

Bemerkenswerterweise wurde eine bedeutende Anzahl dieser Applikationen ebenfalls unter Verwendung des Spring Frameworks implementiert. Dies deutet darauf hin, dass die Technologien, welche für die Entwicklung der W.A.B.S.-Applikation ausgewählt wurden, für die Umsetzung ähnlicher Applikationen geeignet sind und bereits von anderen Entwicklern erfolgreich eingesetzt wurden.

Die systematische Literatursuche half dabei, ein umfassendes Verständnis für die Funktionsweise von webbasierten Anwendungen mit ähnlicher Funktionalität wie die W.A.B.S.-Applikation zu entwickeln. Zudem bestätigte sie, dass die für die Entwicklung ausgewählten Technologien geeignet sind.

### **3.3 Einordnung der Technologien und Alternativen**

Im folgenden Abschnitt werden die angewandten Technologien und der Grund, warum man sich innerhalb des Projektteams für jene Technologien entschieden hat, genauer erklärt.

#### **3.3.1 Realisierung der Datenbank mit MariaDB**

Die Wahl des Datenbanksystems für die W.A.B.S-Applikation wurde sorgfältig evaluiert. Zunächst wurde in Absprache mit dem Projektbetreuer die Sinnhaftigkeit der Auswahl von Google-Firebase ([firebase.google.com](https://firebase.google.com), 2023) als dokumentenbasierte Datenbanktechnologie eruiert. Der Hauptvorteil dieser Technologie war die Möglichkeit, die Datenbank auf Google-Servern zu hosten, sowie die Aspekte der Authentifizierung und Autorisierung zu übernehmen. Ein Prototyp wurde erstellt, jedoch wurde nach Absprache mit dem Betreuer festgestellt, dass es bessere Alternativen gibt und es kontraproduktiv wäre, die benötigten Datensätze der Applikation in einer dokumentenbasierten Datenbank zu persistieren.

Dementsprechend wurde beschlossen, dass das Team bei einem vertrauten relationalen Datenbanksystem bleibt. Die Wahl fiel auf MariaDB ([mariadb.org](https://mariadb.org), 2023), da es eine unkomplizierte Open-Source Technologie ist, deren SQL-Syntax dem Team bereits vertraut ist und die Anbindung an das Spring-Framework im Gegensatz zu Firebase sehr einfach ist. Eine Alternative wäre MySQL ([mysql.com](https://mysql.com), 2023), das ebenso gängig und ausreichend für den momentanen Gebrauch ist. Da MariaDB jedoch in Bezug auf Geschwindigkeit und Performance besser agiert, entschied sich das Team für MariaDB, um eventuellen zukünftigen Skalierungsfragen besser entgegenzutreten zu können.

#### **3.3.2 Spring Boot & Security**

Für die Entwicklung der W.A.B.S-Applikation fiel die Wahl auf das Spring-Framework, da das gesamte Team bereits eine gute Vorbildung für Java und insbesondere für das Framework für Enterprise-Applikationen Spring Boot hatte. Spring Boot bot sich auch aufgrund seiner modularen Struktur ideal für die Arbeit an der Applikation an.

In Bezug auf Authentifizierung, Autorisierung und Sicherheit bietet das Spring Security Framework alle notwendigen Funktionen, um die Sicherheit der Applikation zu gewährleisten. Mit Spring Security können Benutzer identifiziert und überprüft, Zugriffsrechte und Rollen vergeben und Daten verschlüsselt werden.

Obwohl Spring Security eine sehr leistungsfähige Lösung für die Sicherheit von Spring-Webanwendungen bietet, gibt es auch Alternativen wie Apache Shiro ([shiro.apache.org](https://shiro.apache.org), 2023) oder das Java Authentication and Authorization Service (JAAS) ([wikipedia.com](https://wikipedia.com), 2023), die in Betracht gezogen werden können. Die Entscheidung für Spring Security wurde jedoch aufgrund der bereits vorhandenen Erfahrung des Teams mit dem Spring-Framework und der modularen Struktur von Spring Boot getroffen.

### 3.3.3 Umsetzung des Frontends mit Thymeleaf

Bei der Umsetzung des Frontends wurde Thymeleaf verwendet. Dieses Framework eignet sich, wie bereits in Abschnitt 3.1 erwähnt, hervorragend in Verbindung mit dem Spring Framework und wurde speziell für dessen Verwendung entwickelt. Thymeleaf ermöglicht eine nahtlose Integration und zeichnet sich durch eine natürliche Syntax aus, die der von HTML sehr ähnelt. Dadurch wird die Lesbarkeit und Wartbarkeit des Codes erheblich erleichtert.

Neben Thymeleaf gibt es auch Alternativen wie FreeMarker ([freemarker.apache.org](https://freemarker.apache.org), 2023), Velocity ([velocity.apache.org](https://velocity.apache.org), 2023) oder Mustache ([wikipedia.com](https://en.wikipedia.org), 2023). Die Entscheidung für Thymeleaf wurde hauptsächlich getroffen, weil das gesamte Team bereits mit der Verwendung des Frameworks vertraut war. Eine Evaluation der Alternativen wurde jedoch durchgeführt, um sicherzustellen, dass die Wahl die bestmögliche Lösung für das Projekt darstellte.

### 3.3.4 Darstellung und Interagierbarkeit durch JavaScript,JqueryUI,SVG.js

Die Veranschaulichung und Interaktion von zweidimensionalen Räumen in Webanwendungen ist eine komplexe Aufgabe, die den Einsatz mehrerer Technologien erfordert. Eine wichtige Komponente hierbei ist die Darstellung der Räume im SVG-Format, da es die Darstellung von Polygonen und anderen Formen erleichtert.

Für die Projektion der Räume wird die JavaScript-Bibliothek SVG.js verwendet. Diese Bibliothek erleichtert die Erstellung, Skalierung und Manipulation von SVG-Elementen, indem sie eine umfangreiche API ([redhat.com](https://www.redhat.com), 2023) bereitstellt. Im Gegensatz zu anderen Bibliotheken bietet SVG.js zudem eine nahtlose Skalierung unabhängig vom Browser der Webanwendung.

Ein weiterer wichtiger Aspekt ist die Interaktion der Nutzer mit den SVG-Elementen. Hierfür wird die Bibliothek JqueryUI eingesetzt. JqueryUI bietet eine Vielzahl von interaktiven Elementen, darunter Drag-and-Drop-Funktionalitäten, Zoom- und Pan-Optionen sowie Tooltips. Diese Funktionen sind wichtig, um eine effektive und benutzerfreundliche Interaktion zwischen Nutzern und der Webanwendung zu gewährleisten.

Alternativen zu SVG.js wären beispielsweise D3.js ([d3js.org](https://d3js.org), 2023) oder Snap.svg. ([snapsvg.io](https://snapsvg.io), 2023) D3.js ist eine sehr mächtige Bibliothek, die auch die Visualisierung von komplexen Daten ermöglicht. Snap.svg ist ebenfalls eine starke Bibliothek, die sich jedoch auf die Manipulation von SVG-Elementen konzentriert. Die Wahl von SVG.js wurde hauptsächlich getroffen, da es eine leichte und benutzerfreundliche Bibliothek ist, die sich gut für die Projektion von polygonen Räumen eignet.

Als Alternative zu JqueryUI könnte auch die Bibliothek React ([react.dev](https://react.dev), 2023) eingesetzt werden. React ist eine JavaScript-Bibliothek, die speziell für die Erstellung von Benutzeroberflächen entwickelt wurde. Es bietet eine Vielzahl von Komponenten, die es Entwicklern erleichtern, interaktive und dynamische Benutzeroberflächen zu erstellen. Die Wahl von JqueryUI wurde jedoch getroffen, da es eine bewährte Bibliothek mit einer umfangreichen Palette an interaktiven Elementen ist, die sich gut für die Bedürfnisse des Projekts eignen.

## 4 Testfälle

Im folgenden Abschnitt wurden unterschiedliche Testungen zu den verschiedenen Modulen der Softwareanwendung erstellt. Zur Testung wurde die dynamische Methode verwendet.

Die Testungen wurden mit einer Test ID betitelt, die Informationen über das jeweilige Modul, die zu testende Methode und die Anzahl der durchgeführten Testungen bereitstellt.

Zusätzlich wurde auch eine Testbeschreibung eingefügt, welche dem Leser mitteilt, womit sich der vorliegende Test beschäftigt. Um ein tieferes Verständnis für den jeweiligen Testfall zu übermitteln, wurde auch eine Annahme, welche das erwartete Verhalten des Tests beschreibt, als auch die Voraussetzungen und die durchgeführten Schritte niedergeschrieben.

Vor der tatsächlichen Testung wurde ein erwartbares Ergebnis erstellt, welches das zu erwartende Verhalten der zu testenden Funktionalität dokumentiert. Mit anderen Worten: Der Soll Zustand.

Nach der Durchführung der Testung wurde das tatsächlich eingetretene Verhalten dokumentiert (IST Zustand).

Der Status gibt an, ob ein Test das zu erwartende Ergebnis tatsächlich erfüllt hat. Ist dem so, so ändert sich der Status auf „PASSED“. Wäre die Testung erfolglos gewesen, so wäre der Status auf „FAIL“ gesetzt worden und es hätte einen neuen Testfall gegeben.

In der nachfolgenden Abbildung 11: Dynamische Testung Ressourcen, ist ein Testfall aufgeführt. Hier wurde die Funktionalität, ob das Löschen einer Ressource funktioniert und ob die hinterlegten Informationen ebenfalls aus der Datenbank gelöscht wurden. Der Test wurde durch das aktive Aufrufen der Funktionalität innerhalb der Anwendung durchgeführt und das erwartete und tatsächliche Ergebnis wurden jeweils dokumentiert.

Da der Test erfolgreich war, hat sich der Status des Tests auf „PASSED“ umgestellt.

|  |  |   |              |        |
|--|--|---|--------------|--------|
| Testfall-ID:   | T-Ress.-DeleteMethod.001   |   | Status:      | PASSED |
| Testbeschreibung:  | Dieser Test beschäftigt sich damit, ob die Funktionalität der Delete-Method auch tatsächlich das gewünschte Resultat erzeugt.                      |   |              |        |
| Annahme:   | Testdaten:   | Schritte:   |              |        |
| Das Löschen einer Ressource mittels dem "Delete"-Button soll die gespeicherte Ressource aus der Datenbank entfernen  | Um Ressourcen verwalten zu können muss der User die Admin oder Operator Rolle besitzen. Dazu werden die Login Daten Username und Passwort benötigt | 1. Einloggen als Admin/Operator.<br>2. Navigierung zu "Ressourcen".<br>3. Alle Ressourcen anzeigen lassen.<br>4. Zu löschen Ressource auswählen.<br>5. Betätigung der "Delete"-Buttons.<br>6. Bestätigen der durchführbaren Operation.<br>7. Webseite neu laden.<br>8. Überprüfung, ob die Ressource noch vorhanden ist |              |        |
| Erwartetes Ergebnis  |  | Tatsächliches Ergebnis  |              |        |
| Es wird erwartet, dass die zu löschen Ressource nach durchführung der Schritte nicht mehr auf der Webseite aufscheint und auch aus der Datenbank entfernt wird |  | Beim durchführen des Lösch-Vorgangs wurde die zu löschen Ressource sowohl von der Webseite als auch aus der Datenbank entfernt.   |              |        |
| Date:  | 15.05.2023   | Tester:   | Payer Manuel |        |

Abbildung 11: Dynamische Testung Ressourcen

Im Anhang befinden sich weitere Testfälle, die sich jeweils auf einen Aspekt einer Funktionalität beziehen. Es wurde immer dieselben Schemata verwendet, um die Testfälle zu beschreiben. Angeführt von einer Beschreibung des Testfalls, gefolgt von den durchgeföhrten Schritten sowie dem erwarteten Ergebnis und dem tatsächlich eingetretenen Ergebnis. Technologieauswahl und Datenbankentscheidung

Im folgenden Kapitel werden potenzielle Technologien und Frameworks analysiert, die den Projektanforderungen entsprechen, und anschließend die finale Auswahl präsentiert. Zusätzlich erfolgt eine detaillierte Erläuterung der gewählten Datenbanktechnologie.

# 5 Technologieauswahl und Datenbankentscheidung

Im folgenden Kapitel werden potenzielle Technologien und Frameworks analysiert, die den Projektanforderungen entsprechen, und anschließend die finale Auswahl präsentiert. Zusätzlich erfolgt eine detaillierte Erläuterung der gewählten Datenbanktechnologie.

## 5.1 Analyse von Technologieoptionen

Um die passende Technologie für die Umsetzung des Projekts zu finden, wurden die Anforderungen an das Projekt (Funktionalitäten) genauer betrachtet. In der Java-Welt stehen Entwickler:innen zahlreiche Frameworks und Technologien zur Verfügung. Das Spring Framework (Spring Framework, 2023) beispielsweise, das in vielen Projekten eingesetzt wird, bietet Module für fast jeden Aspekt der Softwareentwicklung - von Webentwicklung mit Spring MVC über Sicherheitsfeatures mit Spring Security bis hin zu Datenzugriffsmethoden mit Spring Data und beschleunigter Anwendungsentwicklung durch Spring Boot.

Während Spring in der Java-Community hoch angesehen ist, gibt es auch den Java EE-Standard (Oracle, 2023) für Unternehmensanwendungen. Dieser umfasst eine Vielzahl von APIs für verschiedene Entwicklungsaufgaben, einschließlich JPA für Persistenz und JSF für Webentwicklung. Bei der Datenpersistenz bietet Hibernate als ORM-Framework eine objektorientierte Sicht auf relationale Datenbanken, wohingegen MyBatis (MyBatis, 2023) Entwickler:innen mehr Kontrolle über ihre SQL-Abfragen lässt.

Für moderne Web-UIs könnte man sich Vaadin (Vaadin, 2023) zuwenden, das eine serverseitige Logik bevorzugt, aber auch clientseitige Erweiterungen zulässt. Apache Struts (Apache Struts, 2023), obwohl älter als einige seiner Gegenstücke, ist immer noch ein brauchbares Web-Framework basierend auf dem MVC-Prinzip. In Sachen Sicherheit stellt Apache Shiro (Apache Shiro, 2023) ein leichtgewichtiges Framework dar, das von Authentifizierung bis Kryptographie alles abdeckt.

Die jüngsten Neuzugänge in der Java-Framework-Landschaft, Quarkus (Quarkus, 2023) und Micronaut (Micronaut, 2023), sind speziell darauf ausgelegt, Java-Anwendungen mit schnellen Startzeiten und minimalem Speicherbedarf für Cloud- und Microservices-Architekturen zu unterstützen.

Die Wahl des richtigen Frameworks oder der richtigen Technologie hängt stark von den spezifischen Anforderungen des Projekts und den Vorlieben des Entwicklerteams ab.

Schlussendlich hat sich das Projektteam für das Spring Framework entschieden. Neben den technischen Vorteilen des Frameworks war der ausschlaggebende Grund, dass Spring bereits Teil des Unterrichts war. Daher besaß das Projektteam ein Grundverständnis und eine Erfahrungsbasis mit dieser Technologie. Diese Vertrautheit machte es zu einer logischen Wahl, insbesondere da Spring alle für das Projekt erforderlichen technischen Anforderungen erfüllte.

## 5.2 Die Funktionen von Spring

In der Welt von Spring, einem weitverbreiteten Framework für die Entwicklung von Java-Anwendungen, stößt man auf verschiedene Kernkonzepte, die seine Beliebtheit und Wirksamkeit unterstreichen. Ein Schlüsselprinzip ist die Dependency Injection (DI) (Baeldung, 2023). Bei DI handelt es sich um ein Design-Prinzip, bei dem Abhängigkeiten einer Klasse von außen bereitgestellt werden, anstatt intern instanziert zu werden. Dies fördert Modularität und Testbarkeit. Spring bietet zwei Hauptmethoden zur Implementierung von DI: über Konstruktoren und Setter-Methoden.

Ein weiterer entscheidender Aspekt von Spring ist der Datenzugriff. Durch das Java Database Connectivity (JDBC)-Template wird die Interaktion mit Datenbanken erheblich vereinfacht, indem redundanter und fehleranfälliger Code vermieden wird (GeeksForGeeks, 2023). Ergänzend dazu unterstützt Spring ein deklaratives Transaktionsmanagement, das eine klare Trennung von Geschäftslogik und Transaktionsregeln ermöglicht.

Wendet man sich der Webentwicklung zu, so tritt Spring MVC in den Vordergrund (upGrad, 2023). Dieses Framework implementiert das Model-View-Controller-Prinzip und fördert so eine klare Trennung von Daten, Benutzeroberfläche und Kontrolllogik. Ob man nun eine traditionelle Webanwendung oder einen RESTful Web Service erstellen möchte, Spring MVC bietet die Werkzeuge dazu.

In einer Zeit, in der Sicherheit von größter Bedeutung ist, glänzt Spring Security durch seine robusten Authentifizierungs- und Autorisierungsmechanismen (Spring Security, 2023). Es schützt Anwendungen nicht nur vor allgemeinen Cyber-Bedrohungen, sondern bietet auch eine breite Palette von Sicherheitsfunktionen. Im Abschnitt 3.3.2 - Spring Boot & Security, wird näher auf die Funktionen von Spring Security eingegangen.

Ein nicht zu übersehendes Merkmal von Spring sind die Annotationen (Spring Framework Guru, 2023). Diese speziellen Markierungen im Code, oft vor Klassen oder Methoden platziert, geben dem Spring-Framework Anweisungen bezüglich der Behandlung dieser Komponenten. Ob es sich nun um eine "@Service"-Annotation handelt, die eine Klasse als Geschäftsservice markiert, oder um eine "@Autowired"-Annotation, die DI anfordert, Annotationen sind das Rückgrat der Konfiguration von Spring (Baeldung, 2023). In Abbildung 12 - @Service-Annotation der EmployeeServiceImplementation-Klasse, ist die Annotation "@Service" ersichtlich.

```
@Service
public class EmployeeServiceImplementation
    implements EmployeeService {
```

Abbildung 12 - @Service-Annotation der EmployeeServiceImplementation-Klasse

Durch diese weiß das Spring Framework, dass die Klasse als Service-Komponente innerhalb des Spring Application Contexts zu behandeln ist. Dies bedeutet, dass sie beim Start der Anwendung automatisch von Spring erkannt und als Bean (Baeldung, 2023) instanziert wird. Beans, die mit "@Service" annotiert sind, können dann mittels Dependency Injection in andere Komponenten der Anwendung eingebunden werden. Die "@Service"-Annotation ist eine spezielle Kennzeichnung für Komponenten, die Geschäftslogik oder Service-Aufrufe enthalten. Weitere Annotationen werden im Abschnitt 6 *Mitarbeitermodul*, genauer beschrieben.

## 5.3 SpringBoot als Framework

SpringBoot (SpringBoot, 2023), ein prominenter Vertreter des Spring-Ökosystems, hat die Art und Weise revolutioniert, wie Entwickler:innen Spring-basierte Anwendungen erstellen. Es handelt sich hierbei um ein Framework, das darauf ausgelegt ist, die Erstellung von produktionsreifen, eigenständigen Spring-Anwendungen zu vereinfachen. Anstatt Entwickler:innen mit zahlreichen Konfigurationen und Einstellungen zu belasten, minimiert Spring Boot diesen Aufwand erheblich.

Ein hervorstechendes Merkmal von Spring Boot ist seine Auto-Konfigurationsfähigkeit. Wenn Bibliotheken zum Projekt hinzugefügt werden, versucht Spring Boot, automatisch die bestmögliche Konfiguration dafür zu erstellen. Das bedeutet, dass sich Entwickler:innen weniger um spezifische Einstellungen kümmern müssen und sich mehr auf den eigentlichen Code konzentrieren können.

Ein weiterer Vorteil ist die Standalone-Natur von Spring Boot. Anwendungen müssen nicht auf externe Webserver angewiesen sein, da sie als unabhängige JAR- oder WAR-Dateien laufen können (spring.io, 2023). Dabei sind sie dennoch produktionsreif. Sie verfügen über Funktionen wie Gesundheitschecks, was die Überwachung und Wartung in einer Produktionsumgebung erleichtert.

Im Gegensatz zu einigen anderen Frameworks generiert Spring Boot keinen Code. Die Konfiguration erfolgt über Konventionen und Annotationen, was den Code sauber und verständlich hält.

## 5.4 Sicherung von Daten

Da es für die reibungslose Funktionsweise von W.A.B.S. von entscheidender Bedeutung ist, Daten von Mitarbeiter:innen und Ressourcen effektiv zu verwalten und speichern zu können, erfordert die Wahl der richtigen Datenbanktechnologie besondere Aufmerksamkeit und Sorgfalt. In den nächsten Unterabschnitten wird die Auswahl der Datenbanktechnologien näher erläutert.

### 5.4.1 Firebase

In den Anfangsstadien des Projekts war die Firebase-Datenbank (Google Firebase, 2023) die erste Wahl für die Speicherung und Verwaltung von Daten. Firebase, eine weit verbreitete und äußerst flexible Plattform von Google, bietet eine dokumentenbasierte NoSQL-Datenbank, die auf die Speicherung von Informationen in strukturierten JavaScript Object Notation (JSON)-Dokumenten abzielt. Dieses Modell schien anfangs ideal für das Projekt zu sein, da es die Datenverwaltung und Echtzeitsynchronisation ermöglichte.

Firebase verwendet ein dokumentenbasiertes Modell, bei dem Informationen in Sammlungen organisiert werden, die ähnlich wie Tabellen in einer relationalen Datenbank funktionieren. Jede Sammlung enthält Dokumente, die wiederum ähnlich wie Zeilen in JSON-Format strukturiert sind. Dieses Modell ermöglicht eine äußerst flexible Speicherung und Abfrage von Daten, was besonders nützlich ist, wenn sich die Datenstrukturen in der Anwendung häufig ändern.

Ein herausragendes Merkmal der Firebase-Datenbank ist ihre Echtzeiterfassung. Dies bedeutet, dass jede Änderung an den Daten sofort an alle verbundenen Clients übertragen wird. Diese Echtzeitsynchronisation eignet sich hervorragend für gemeinsam verwendeten Anwendungen, bei denen mehrere Benutzer:innen gleichzeitig an denselben Daten arbeiten, sowie für Echtzeitanwendungen, die von ständig aktualisierten Informationen abhängen.

Firebase bietet SDKs (Software Development Kits) für JavaScript, um Webanwendungen zu entwickeln, sowie für Android und iOS, um mobile Anwendungen zu erstellen. Diese Vielseitigkeit erleichtert die Entwicklung plattformübergreifender Anwendungen erheblich.

Darüber hinaus bietet Firebase skalierbare Cloud-Hosting-Lösungen und automatische Lastenausgleichsoptionen. Dies stellt sicher, dass die Anwendung stets die erforderliche Leistung bietet, unabhängig davon, wieviele Benutzer:innen gleichzeitig darauf zugreifen.

#### 5.4.2 Der Übergang von Firebase zu einer relationalen Datenbank

Während der Projektentwicklung wurde deutlich, dass sich die Anforderungen und Komplexitäten des Systems in eine andere Richtung bewegten. Das Projekt konzentrierte sich auf die umfassende Verwaltung von Mitarbeiter:innen und deren Arbeitszeiten, sowohl im Büro als auch im Homeoffice. Diese Komplexität erforderte eine tiefere Datenmodellierung und eine bessere Unterstützung für Beziehungen zwischen verschiedenen Datensätzen. Hier stieß man an die Grenzen der dokumentenbasierten Datenbankstruktur von Firebase.

Ein weiterer wichtiger Faktor für den Wechsel war die Integration von Sicherheitsfunktionen. Die Entscheidung fiel auf die Verwendung von Spring Security, einer leistungsstarken Lösung für die Sicherheit von Java-Anwendungen. Die Kombination von Spring Security mit einer relationalen Datenbank ermöglichte eine effiziente Verwaltung von Benutzerzugriffsrechten und eine robuste Absicherung der Anwendung.

Darüber hinaus führte die Entscheidung, die Anwendung auf einem Server des Projektpartners zu hosten, dazu, dass man nicht mehr von der Bereitstellung über Firebase abhängig war. Dies ermöglichte eine größere Flexibilität bei der Wahl der Datenbanktechnologie.

Insgesamt führten diese Faktoren zur Entscheidung, von einer dokumentenbasierten Datenbank wie Firebase zu einer relationalen Datenbank überzugehen. Diese strategische Umstellung ermöglichte es, die Anforderungen des Projekts in Bezug auf Datenmodellierung, Sicherheit und Flexibilität besser zu erfüllen und legte damit die Grundlage für ein leistungsfähiges und maßgeschneidertes Arbeitsplatzbuchungssystem. Für die Implementierungsphase wurde die H2 Datenbank (H2-Database, 2023) verwendet, die mit einer Dependency einfach in das Projekt integriert werden kann und daher ohne großen Konfigurationsaufwand verwendet werden konnte.

#### 5.4.3 H2-Datenbank

Die H2-Datenbank unterscheidet sich grundlegend von der dokumentenbasierten Firebase-Datenbank, da sie auf dem relationalen Datenbankmodell basiert. In H2 werden Daten in Tabellen mit Zeilen und Spalten gespeichert, was die Verwaltung von komplexen Beziehungen zwischen verschiedenen Datensätzen ermöglicht.

H2 zeichnet sich durch beeindruckende Geschwindigkeit und Leichtgewichtigkeit aus. Sie verarbeitet Daten äußerst schnell und erfordert nur minimale Konfiguration, was ihre Integration in Projekte erleichtert.

Die Plattformunabhängigkeit von H2 ermöglicht einen reibungslosen Einsatz auf verschiedenen Betriebssystemen, was hohe Flexibilität bei der Entwicklung und Bereitstellung von Anwendungen bietet.

Des Weiteren bietet H2 die Möglichkeit zur Durchführung von Transaktionen, die entscheidend sind, um die Konsistenz und Integrität der Daten in einem Buchungssystem sicherzustellen. Dies ist besonders wichtig, um sicherzustellen, dass alle Buchungen und Aktualisierungen korrekt und zuverlässig abgewickelt werden.

Zusätzlich zur Transaktionsunterstützung bietet H2 die Option, Daten im RAM zu speichern. Dieser In-Memory-Modus kann die Datenverarbeitung weiter beschleunigen und somit die Leistungsfähigkeit der Anwendung steigern.

# 6 Mitarbeitermodul

Das Mitarbeitermodul stellt einen grundlegenden Baustein des Arbeitsplatzbuchungssystems dar. Es ermöglicht die Verwaltung verschiedener Mitarbeiter:innen, von Büroangestellten bis hin zu Remote-Mitarbeiter:innen, und weist ihnen individuelle Benutzerrollen zu. Dieser Abschnitt wird die Employee-Klasse und die verschiedenen Schichten der Anwendung vorstellen, die für die Verwaltung von Mitarbeiter:innen zuständig sind.

## 6.1 Die Schichten des Mitarbeitermoduls

Um die Verwaltung von Mitarbeiter:innen in der Anwendung effizient zu gestalten, werden verschiedene Schichten (Layer) implementiert. Diese einzelnen Layer werden in den folgenden Unterabschnitten genauer beschrieben.

### 6.1.1 Domain Layer

Wie in Abbildung 13 - Domain Layer ersichtlich ist, enthält der "Domain"-Layer unter anderem die Employee-Entitätsklasse, die das Datenmodell für Mitarbeiter:innen darstellt. Die Employee-Entität ist mit der Datenbanktabelle verknüpft und ermöglicht das Speichern und Abrufen von Mitarbeiterdaten.

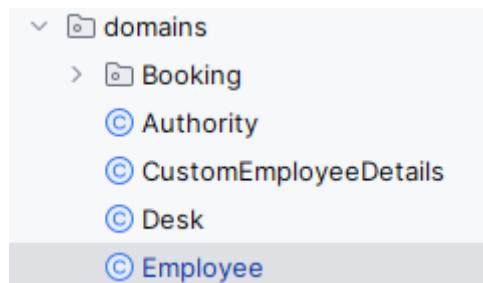


Abbildung 13 - Domain Layer

Die "Employee"-Klasse bildet das Datenmodell für Mitarbeiter:innen in der Anwendung und repräsentiert eine/n Mitarbeiter:in. Die in Abbildung 14 ersichtlichen Annotationen für die "Employee"-Entität sowie die Beschreibung der Entitäts-Klasse und ihrer Eigenschaften wird im Folgenden näher erläutert.

```
@Entity
@NoArgsConstructor
@Getter
@Setter
@ToString
public class Employee implements UserDetails {
```

Abbildung 14 – Annotationen der Employee Entität

- **@Entity:** Diese Annotation kennzeichnet die Klasse als JPA-Entity, was bedeutet, dass sie in einer Datenbank gespeichert und verwaltet wird.

- **@NoArgsConstructor:** Diese Lombok-Annotation generiert einen Standardkonstruktor ohne Parameter.
- **@Getter und @Setter:** Diese Lombok-Annotationen generieren automatisch Getter- und Setter-Methoden für die Eigenschaften der Klasse.
- **@ToString:** Diese Lombok-Annotation generiert eine `toString`-Methode.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

Abbildung 15 - Employee Datenfeld id

- **@Id und @GeneratedValue:** Diese Annotationen (Abbildung 15 - Employee Datenfeld id") kennzeichnen das "id"-Feld als Primärschlüssel und geben an, dass der Wert automatisch generiert wird.

```
@Size(min = 2, max = 50)
private String fname;
```

```
@Size(min = 2, max = 50)
private String lname;
```

```
@Size(min = 2, max = 10)
private String nick;
@Email
private String email;
```

```
@NotNull
@Size(min = 2, max = 50)
private String password;
```

```
@NotNull
private Role role;
```

Abbildung 16 – Spezifische Employee Datenfelder

Die Datenfelder, die in Abbildung 16 – Spezifische Employee Datenfelder zu sehen sind, repräsentieren die Informationen einer Mitarbeiterin oder eines Mitarbeiters, darunter Vorname ("`fname`"), Nachname ("`lname`"), Spitzname ("`nick`"), E-Mail-Adresse ("`email`"), Passwort ("`password`") und Rolle ("`role`"). Die Annotationen wie "`@Size`", "`@Email`" und "`@NotNull`" dienen der Validierung der Eingabedaten und stellen sicher, dass die Daten den Anforderungen entsprechen.

```
public Employee(String fname, String lname, String nick,
                String email, String password, Role role) {
    this.fname = fname;
    this.lname = lname;
    this.nick = nick;
    this.email = email;
    this.password = password;
    this.role = role;
}
```

Abbildung 17 - Konstruktor der Employee-Klasse

Der Konstruktor in Abbildung 17 ermöglicht das einfache Erstellen eines Mitarbeiterobjekts mit den erforderlichen Informationen.

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();
    authorities.add(new SimpleGrantedAuthority(role.name()));
    return authorities;
}
```

Abbildung 18 - Hier wird eine Liste von Berechtigungen/Rollen zurückgegeben

Die "getAuthorities()"-Methode in Abbildung 18 wird von Spring Security benötigt und gibt die Berechtigungen (Rollen) des Benutzers zurück. Die Rolle wird dann als "SimpleGrantedAuthority" zurückgegeben.

```
@Override
public String getUsername() { return this.getNick(); }
```

Abbildung 19 - Gibt den Spitznamen der Mitarbeiterin/des Mitarbeiters zurück

Die in Abbildung 19 dargestellte "getUsername()"-Methode gibt den Benutzernamen zurück, der als Spitzname ("nick") festgelegt ist.

```

@Override
public boolean isAccountNonExpired() { return true; }

@Override
public boolean isAccountNonLocked() { return true; }

@Override
public boolean isCredentialsNonExpired() { return true; }

@Override
public boolean isEnabled() { return true; }

```

Abbildung 20 - Spring Security UserDetails Methoden

Im Spring Security "UserDetails"-Interface, wie in Abbildung 20 gezeigt, sind zusätzliche Methoden verfügbar, die Informationen zum Status des Benutzerkontos liefern. Dazu gehören die Überprüfung auf Kontoablauf, Kontosperrung, Gültigkeit der Anmeldeinformationen und Aktivierung des Kontos. Alle Methoden sind so implementiert, dass das Konto immer aktiv und nicht gesperrt ist.

### 6.1.2 Repository-Layer

Im "Repository"-Layer, wie in Abbildung 21 dargestellt, werden CRUD-Operationen (Create, Read, Update, Delete) für Mitarbeiter:innen ausgeführt. Die EmployeeRepository-Schnittstelle bietet JPA-Methoden zur Verwaltung von Mitarbeiterdaten.

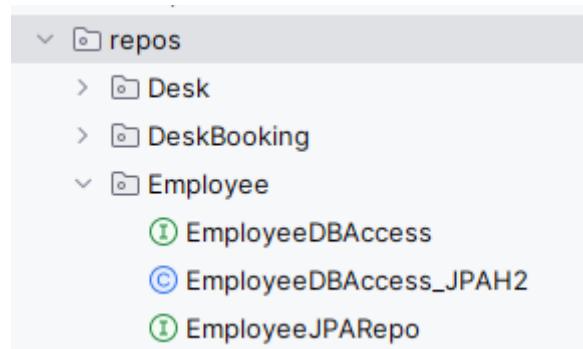


Abbildung 21 - Repository Layer

Das "EmployeeDBAccess"-Interface in Abbildung 22 enthält die definierten Methoden, die für den Datenzugriff auf Mitarbeiterdaten benötigt werden.

```

public interface EmployeeDBAccess {
    Employee saveEmployee(Employee employee)
        throws ExecutionException, InterruptedException, EmployeeAlreadyExistsException;

    List<Employee> getAllEmployees()
        throws ExecutionException, InterruptedException;

    Employee getEmployeeById(Long id)
        throws EmployeeNotFoundException, ExecutionException, InterruptedException;

    void deleteEmployeeById(Long id)
        throws EmployeeDeletionNotPossibleException;

    Employee getEmployeeByEmail(String email);

    Employee getEmployeeByNick(String nick);

}

```

Abbildung 22 - Interface für den Datenzugriff auf Mitarbeiter:innen

- **saveEmployee():** Speichert eine/n Mitarbeiter:in in der Datenbank und gibt diese oder diesen zurück. Falls die Mitarbeiterin oder der Mitarbeiter bereits existiert, kann eine "EmployeeAlreadyExistsException" ausgelöst werden.
- **getAllEmployees():** Ruft eine Liste aller Mitarbeiter:innen aus der Datenbank ab.
- **getEmployeeById():** Sucht eine/n Mitarbeiter:in anhand der "id" und gibt diese oder diesen zurück. Falls die Mitarbeiterin oder der Mitarbeiter nicht gefunden wird, kann eine EmployeeNotFoundException ausgelöst werden.
- **deleteEmployeeById():** Löscht eine:n Mitarbeiter:in anhand seiner "id". Falls das Löschen nicht möglich ist, wird eine "EmployeeDeletionNotPossibleException" ausgelöst werden.
- **getEmployeeByEmail() und getEmployeeByNick():** Sucht nach Mitarbeiter:innen anhand ihrer E-Mail-Adresse bzw. ihres Spitznamens und gibt den/die gefundenen Mitarbeiter zurück.

Das "EmployeeJPARepo"-Interface, dass in Abbildung 23 zu sehen ist, erweitert das JpaRepository von Spring Data JPA und erbt dadurch grundlegende CRUD-Operationen für die Entität Employee. Dieses Interface wird mit "@Repository" annotiert.

Die "@Repository"-Annotation in Spring ist eine Spezialisierung der "@Component"-Annotation und wird in der Regel auf Klassen angewendet, die die Datenzugriffsschicht einer Anwendung repräsentieren. Sie kennzeichnet eine Klasse als ein Repository, das für den Datenbankzugriff verantwortlich ist.

```

@Repository
public interface EmployeeJPARepo extends JpaRepository<Employee, Long> {
    Employee getEmployeeByEmail(String email);

    Employee getEmployeeByNick(String nick);
}

```

Abbildung 23 - Das EmployeeJPARepo erweitert das JpaRepository mit benutzerdefinierten Methoden

- **getEmployeeByEmail()** und **getEmployeeByNick()**: Sucht nach Mitarbeiter:innen anhand der E-Mail-Adresse bzw. des Spitznamens und gibt diese oder diesen zurück.

Die restlichen Standardmethoden wie "*findAll()*", "*findById()*", "*save()*", und "*deleteById()*" werden vom "*JpaRepository*" geerbt und können verwendet werden, um grundlegende CRUD-Operationen durchzuführen.

Die "*EmployeeDBAccess*"-Klasse (Abbildung 24) implementiert das "*EmployeeDBAccess*"-Interface und stellt die konkrete Implementierung der Datenbankzugriffsfunktionen bereit. Hier werden die Methoden aus dem Interface mit den Methoden aus dem Spring Data JPA-Repository "*EmployeeJPARepo*" verknüpft.

```
@Component
public class EmployeeDBAccess_JPAH2 implements EmployeeDBAccess {

    private final EmployeeJPARepo employeeJPARepo;
```

Abbildung 24 - Konkrete Implementierung für den Mitarbeiterdatenzugriff

Die "**@Component**"-Annotation in Spring dient dafür, eine Klasse als Spring-Komponente zu kennzeichnen. Eine Spring-Komponente ist im Grunde eine Klasse, die von Spring verwaltet und instanziert wird. Spring scannt das Klassenverzeichnis nach Klassen mit dieser Annotation und erstellt entsprechende Instanzen, die in seinem ApplicationContext verwaltet werden.

```
public EmployeeDBAccess_JPAH2(EmployeeJPARepo employeeJPARepo) {
    this.employeeJPARepo = employeeJPARepo;
}
```

Abbildung 25 - Konstruktor der *EmployeeDBAccess\_JPAH2*-Klasse

Im Konstruktor, wie in Abbildung 25 gezeigt, wird das "*EmployeeJPARepo*" als notwendige Abhängigkeit injiziert. Diese Abhängigkeit kommt bei Datenbankoperationen für Mitarbeiterdaten zum Einsatz.

```
@Override
public Employee saveEmployee(Employee employee)
    throws EmployeeAlreadyExistsException {
    return employeeJPARepo.save(employee);
}
```

Abbildung 26 - Die *saveEmployee()*-Methode speichert eine/n Mitarbeiter:in

Um eine neue Mitarbeiterin oder einen neuen Mitarbeiter in der Datenbank zu speichern, wird die "*saveEmployee()*"-Methode aus Abbildung 26 des "*employeeJPARepo*" aufgerufen. Falls ein Fehler auftritt, wird eine "*EmployeeAlreadyExistsException*" ausgelöst, die darauf hinweist, dass diese Person bereits existiert.

```
@Override
public List<Employee> getAllEmployees() {
    return this.employeeJPARepo.findAll();
}
```

Abbildung 27 - Liefert eine Liste aller Mitarbeiter:innen zurück

Die "getAllEmployees()"-Methode, wie in Abbildung 27 zu sehen, gibt eine Liste aller Mitarbeiter:innen in der Datenbank zurück, indem sie die "findAll()" -Methode des "employeeJPARepo" aufruft.

```
@Override
public Employee getEmployeeById(Long id) throws EmployeeNotFoundException {
    Optional<Employee> optEmployee = this.employeeJPARepo.findById(id);
    if (optEmployee.isPresent()) {
        return optEmployee.get();
    } else {
        throw new EmployeeNotFoundException("The Employee with the ID: "
            + employeeJPARepo.findById(id) + " was not found!");
    }
}
```

Abbildung 28 - Sucht eine/n Mitarbeiter:in anhand der ID

Bei der Suche nach einer Mitarbeiterin oder einem Mitarbeiter anhand der "id" (Abbildung 28) wird eine "EmployeeNotFoundException" ausgelöst, wenn diese oder dieser nicht gefunden wird.

```
@Override
public void deleteEmployeeById(Long id) throws EmployeeDeletionNotPossibleException {
    try {
        this.employeeJPARepo.deleteById(id);
    } catch (Exception e) {
        throw new EmployeeDeletionNotPossibleException("Mitarbeiter mit der ID "
            + id + " konnte nicht gelöscht werden!");
    }
}
```

Abbildung 29 - Löscht eine/n Mitarbeiter:in mithilfe der "id"

Falls die Löschung einer Mitarbeiterin oder eines Mitarbeiters mithilfe der "id", wie in Abbildung 29 dargestellt, aus irgendeinem Grund nicht möglich ist, wird eine "EmployeeDeletionNotPossibleException" ausgelöst.

```
@Override
public Employee getEmployeeByEmail(String email) {
    return this.employeeJPARepo.getEmployeeByEmail(email);
}
```

Abbildung 30 - Sucht eine/n Mitarbeiter:in unter Verwendung der E-Mail-Adresse

Die Methode "getEmployeeByEmail()" (Abbildung 30) des "employeeJPARepo" ermöglicht die Suche nach einer Mitarbeiterin oder einem Mitarbeiter anhand der E-Mail-Adresse.

```
@Override
public Employee getEmployeeByNick(String nick) {
    return this.employeeJPARepo.getEmployeeByNick(nick);
}
```

Abbildung 31 - Gibt eine/n Mitarbeiter:in durch Angabe des Spitznamens zurück

Wie in Abbildung 31 zu sehen ist, ermöglicht die Methode "getEmployeeByNick()" die Suche nach einer Mitarbeiterin oder einem Mitarbeiter anhand des Spitznamens ("nick"). Diese Methode verwendet die "getEmployeeByNick()"-Methode des "employeeJPARepo" und gibt diese oder diesen zurück.

### 6.1.3 Service-Layer

Im "Service"-Layer (Abbildung 32) werden Geschäftslogik und Datenverarbeitung für Mitarbeiter:innen durchgeführt. Hier werden auch die Mitarbeiterobjekte validiert, bevor sie in die Datenbank gespeichert oder aktualisiert werden.

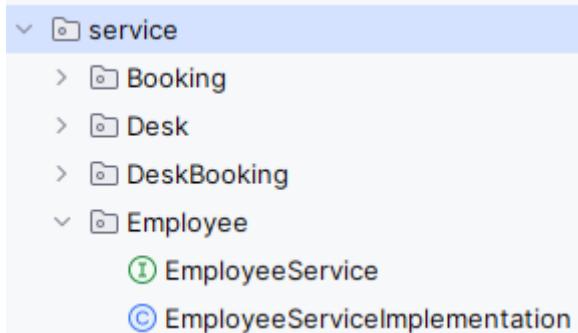


Abbildung 32 - Service-Layer

Die beiden Klassen, "EmployeeService" (Interface) und "EmployeeServiceImplementation" (Implementierungsklasse), gehören zum "Service"-Layer der Anwendung und dienen zur Abstraktion der Geschäftslogik und Zugriff auf Mitarbeiterdaten.

```

public interface EmployeeService extends UserDetailsService {
    Employee addEmployee(Employee employee)
        throws EmployeeAlreadyExistsException, ExecutionException, InterruptedException;

    List<Employee> getAllEmployees()
        throws ExecutionException, InterruptedException;

    Employee getEmployeeById(Long id)
        throws EmployeeNotFoundException, ExecutionException, InterruptedException;

    Employee updateEmployeeById(Employee employee)
        throws ExecutionException, InterruptedException, EmployeeNotFoundException,
        EmployeeAlreadyExistsException;

    void deleteEmployeeById(Long id)
        throws EmployeeDeletionNotPossibleException;

    Employee getEmployeeByNick(String nick);
}

```

Abbildung 33 - *EmployeeService*-Interface

Das "EmployeeService"-Interface, wie in Abbildung 33 zu sehen, definiert einen Satz von Methoden, die die Geschäftslogik für die Mitarbeiterverwaltung festlegen. Es fungiert als Vertrag des "Service"-Layers und dient als Schnittstelle zwischen dem Controller und dem Repository-Layer.

- **addEmployee()**: Fügt eine/n neuen Mitarbeiter:in hinzu und wirft eine "EmployeeAlreadyExistsException", wenn ein:e Mitarbeiter:in mit der gleichen E-Mail-Adresse oder dem gleichen Spitznamen bereits existiert.
- **getAllEmployees()**: Ruft alle Mitarbeiter:innen ab.
- **getEmployeeById()**: Ruft eine/n Mitarbeiter:in anhand seiner ID ab und wirft eine "EmployeeNotFoundException", wenn diese oder dieser nicht gefunden wird.
- **updateEmployeeById()**: Aktualisiert die Informationen einer Mitarbeiterin oder eines Mitarbeiters und kann in bestimmten Fällen eine EmployeeNotFoundException oder eine EmployeeAlreadyExistsException auslösen.
- **deleteEmployeeById()**: Löscht eine/n Mitarbeiter:in anhand seiner ID und wirft eine "EmployeeDeletionNotPossibleException", wenn die Löschung nicht möglich ist.
- **getEmployeeByNick()**: Ruft eine:n Mitarbeiter:in anhand des Spitznamens ab.

Die in Abbildung 34 gezeigte Klasse "EmployeeServiceImplementation" ist eine konkrete Implementierung des "EmployeeService"-Interfaces und stellt die tatsächliche Geschäftslogik für die Mitarbeiterverwaltung zur Verfügung.

```

@Service
public class EmployeeServiceImplementation implements EmployeeService {

    private final EmployeeDBAccess employeeDBAccess;

```

Abbildung 34 – Die konkrete Implementierung der Geschäftslogik für die Verwaltung von Mitarbeiter:innen

Die "@Service"-Annotation im Spring Framework wird verwendet, um eine Klasse als einen sogenannten "Service" zu kennzeichnen. Ein Service ist eine Komponente, die Geschäftslogik enthält und oft für die Verarbeitung von Anfragen und die Bereitstellung von Diensten in einer Anwendung verwendet wird.

```
public EmployeeServiceImplementation(EmployeeDBAccess employeeDBAccess) {
    this.employeeDBAccess = employeeDBAccess;
}
```

Abbildung 35 - Konstruktor der *EmployeeServiceImplementation*-Klasse

Der Konstruktor der Klasse, in Abbildung 35 dargestellt, nimmt ein "EmployeeDBAccess"-Objekt entgegen, das die Schnittstelle zum "Repository"-Layer darstellt. Dieses Objekt wird benötigt, um auf die Datenbank zuzugreifen und Mitarbeiterdaten zu verwalten.

```
@Override
public Employee addEmployee(Employee employee)
    throws EmployeeAlreadyExistsException, ExecutionException, InterruptedException {
String email = employee.getEmail();
String nick = employee.getNick();

Employee existingEmployeeByEmail = employeeDBAccess.getEmployeeByEmail(email);
Employee existingEmployeeByNick = employeeDBAccess.getEmployeeByNick(nick);

if (existingEmployeeByEmail != null) {
    throw new EmployeeAlreadyExistsException("Employee with email already exists");
}

if (existingEmployeeByNick != null) {
    throw new EmployeeAlreadyExistsException("Employee with nick already exists");
}

return this.employeeDBAccess.saveEmployee(employee);
}
```

Abbildung 36 - Speichert eine:n Mitarbeiter:in und gibt diese oder diesen zurück

Die "addEmployee()"-Methode (Abbildung 36) dient dazu, eine neue Mitarbeiterin oder einen neuen Mitarbeiter hinzuzufügen. Vor diesem Schritt erfolgt die Überprüfung der E-Mail-Adresse und des Spitznamens, um Duplikate zu vermeiden. Wenn bereits eine Mitarbeiterin oder ein Mitarbeiter mit derselben E-Mail-Adresse oder demselben Spitznamen existiert, wird eine "EmployeeAlreadyExistsException" ausgelöst. Andernfalls erfolgt die Speicherung des neuen Mitarbeiter-Objekts, gefolgt von seiner Rückgabe.

```
@Override
public List<Employee> getAllEmployees() throws ExecutionException, InterruptedException {
    return this.employeeDBAccess.getAllEmployees();
}
```

Abbildung 37 - Gibt eine Liste aller Mitarbeiter:innen zurück

Die in Abbildung 37 dargestellte "getAllEmployees()"-Methode führt eine Abfrage durch und gibt eine Liste aller Mitarbeiter:innen in der Datenbank zurück.

```
@Override
public Employee getEmployeeById(Long id)
    throws EmployeeNotFoundException, ExecutionException, InterruptedException {
    return this.employeeDBAccess.getEmployeeById(id);
}
```

Abbildung 38 - Sucht eine Mitarbeiterin oder einen Mitarbeiter unter Verwendung der id

Die Methode "getEmployeeById()" ermöglicht es, eine/n Mitarbeiter:in anhand der "id" aus der Datenbank abzurufen, wie in Abbildung 38 zu sehen ist, und löst eine "EmployeeNotFoundException" aus, wenn dieser oder diese nicht existiert.

```
@Override
public Employee updateEmployeeById(Employee employee)
    throws ExecutionException, InterruptedException,
    EmployeeNotFoundException, EmployeeAlreadyExistsException {
    Employee employeeFromDb = this.employeeDBAccess.getEmployeeById(employee.getId());
    if (employeeFromDb == null) {
        throw new EmployeeNotFoundException("The Employee with the ID: "
            + employeeDBAccess.getEmployeeById(employee.getId()) + " was not found!");
    }
    employeeFromDb.setFname(employee.getFname());
    employeeFromDb.setLname(employee.getLname());
    employeeFromDb.setNick(employee.getNick());
    employeeFromDb.setPassword(employee.getPassword());
    employeeFromDb.setEmail(employee.getEmail());
    employeeFromDb.setRole(employee.getRole());

    return this.employeeDBAccess.saveEmployee(employeeFromDb);
}
```

Abbildung 39 - Aktualisiert eine/n Mitarbeiter:in durch Angabe der ID und gibt diese oder diesen zurück

Die Methode "updateEmployeeById()" in Abbildung 39 dient zur Aktualisierung der Mitarbeiterdaten anhand ihrer ID. Sie beginnt mit einer Überprüfung, ob die Mitarbeiterin oder der Mitarbeiter in der Datenbank existiert. Falls nicht, wird eine "EmployeeNotFoundException" ausgelöst. Ansonsten werden die Mitarbeiterinformationen aktualisiert und in der Datenbank gespeichert.

```
@Override
public void deleteEmployeeById(Long id) throws EmployeeDeletionNotPossibleException {
    this.employeeDBAccess.deleteEmployeeById(id);
}
```

Abbildung 40 - Löscht eine/n Mitarbeiter:in anhand der ID

Bei der Methode "deleteEmployeeById()" (Abbildung 40) wird versucht, eine:n Mitarbeiter:in anhand der ID zu löschen. Falls die Löschung nicht möglich ist, wird eine "EmployeeDeletionNotPossibleException" ausgelöst.

```

@Override
public Employee getEmployeeByNick(String nick) {
    Employee employeeDB = this.employeeDBAccess.getEmployeeByNick(nick);
    if (employeeDB == null) {
        // Benutzer nicht gefunden, Fehlerbehandlung durchführen
        throw new IllegalArgumentException("Benutzername nicht gefunden: " + nick);
    }
    return employeeDB;
}

```

Abbildung 41 - Sucht eine/n Mitarbeiter:in basierend auf dem Spitznamen und gibt diesen zurück

Die "getEmployeeByNick()"-Methode in Abbildung 41 ermöglicht das Abrufen einer Mitarbeiterin oder eines Mitarbeiters anhand des Spitznamens und gibt diese oder diesen zurück. Sollte die Mitarbeiterin oder der Mitarbeiter nicht gefunden werden, führt dies zu einer "IllegalArgumentException".

```

@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    Employee employee = this.employeeDBAccess.getEmployeeByNick(username);
    if (employee == null) {
        throw new UsernameNotFoundException("Mitarbeiter mit dem nick "
            + username + " nicht gefunden!");
    }
    return User.builder()
        .username(employee.getNick())
        .password(employee.getPassword())
        .roles(employee.getRole().toString())
        .build();
}

```

Abbildung 42 - Lädt die Benutzerdetails anhand des Mitarbeiternamens und gibt ein UserDetails-Objekt zurück

Als Teil der Spring Security-Integration dient die Methode "loadUserByUsername()" (Abbildung 42) zum Laden von Benutzerdetails, wie Benutzernamen und Passwort, basierend auf dem Benutzernamen. Sie gibt ein "UserDetails"-Objekt zurück, welches für die Benutzerauthentifizierung genutzt wird. Im Falle eines nicht gefundenen Benutzernamens führt dies zu einer "UsernameNotFoundException".

#### 6.1.4 Controller-Layer

Der "Controller"-Layer, wie in Abbildung 43 zu sehen, handhabt HTTP-Anfragen und -Antworten für Mitarbeiteroperationen. Hier befindet sich der "EmployeeWebController", der die Kommunikation zwischen der Benutzeroberfläche und dem Backend ermöglicht.

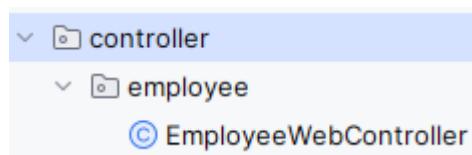


Abbildung 43 - Controller Layer

Abbildung 44 zeigt den "EmployeeWebController", eine Spring MVC-Controller-Klasse, die für die Verarbeitung von HTTP-Anfragen im Zusammenhang mit Mitarbeiter- und Benutzermanagement verantwortlich ist.

```
@Controller
@RequestMapping("/web")
public class EmployeeWebController {

    EmployeeService employeeService;
```

Abbildung 44 - EmployeeWebController-Klasse

Die Klasse ist mit der "@Controller"-Annotation gekennzeichnet, was bedeutet, dass es sich um eine Spring MVC-Controller-Klasse handelt. Sie ist für die Verarbeitung von HTTP-Anfragen und die Bereitstellung von Modellen und Ansichten verantwortlich.

Die "@RequestMapping"-Annotation auf Klassenebene gibt den Basispfad an, unter dem die Controller-Methoden erreichbar sein werden.

```
public EmployeeWebController(EmployeeService employeeService) {
    this.employeeService = employeeService;
}
```

Abbildung 45 - Konstruktor der EmployeeWebController-Klasse

Der Konstruktor der "EmployeeWebController"-Klasse, wie in Abbildung 45 gezeigt, nimmt eine Instanz des "EmployeeService" entgegen. Dies ermöglicht die Injektion des EmployeeService, um auf die Geschäftslogik für Mitarbeiter:innen zuzugreifen.

```
@GetMapping("/admin/allemployees")
public ModelAndView allEmployees() throws ExecutionException, InterruptedException {
    List<Employee> allEmployees = employeeService.getAllEmployees();
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("employee/allemployees");
    modelAndView.addObject(attributeName: "employees", allEmployees);
    return modelAndView;
}
```

Abbildung 46 - Controller-Endpunkt für die Ansicht aller Mitarbeiter:innen

Die Methode "allEmployees()" in Abbildung 46 verarbeitet HTTP-GET-Anfragen an "/admin/allemployees". Sie ruft alle Mitarbeiter:innen über den "EmployeeService" ab und gibt sie an die Ansicht "employee/allemployees" weiter.

```
@GetMapping("/admin/admin-start")
public ModelAndView home() throws ExecutionException, InterruptedException {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("employee/admin-start");
    return modelAndView;
}
```

Abbildung 47- Controller-Endpunkt für die Ansicht der Admin-Startseite

Die "home()"-Methode, wie in Abbildung 47 dargestellt, nimmt HTTP-GET-Anfragen an "/admin/admin-start" entgegen und rendert die Ansicht "employee/admin-start" für Administratoren.

```
@GetMapping("/user/start")
public ModelAndView homeUser() throws ExecutionException, InterruptedException {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("employee/start");
    return modelAndView;
}
```

Abbildung 48 - Controller-Endpunkt für Ansicht der User-Startseite

Die in Abbildung 48 gezeigte "homeUser"-Methode handhabt HTTP-GET-Anfragen an "/user/start" und zeigt die Ansicht "employee/start" für Benutzer:innen an.

```
@GetMapping("/admin/deleteemployee/{id}")
public String deleteEmployeeWithId(@PathVariable Long id, Model model) {
    try {
        this.employeeService.deleteEmployeeById(id);
        return "redirect:/web/admin/allemployees";
    } catch (EmployeeDeletionNotPossibleException e) {
        System.out.println(e.getMessage());
        model.addAttribute(attributeName: "errortitle", attributeValue: "Mitarbeiter-Löschen schlägt fehl!");
        model.addAttribute(attributeName: "errormessage", e.getMessage());
        return "myerrorpage";
    }
}
```

Abbildung 49 - Controller-Endpunkt zum Löschen eines Mitarbeiters

Die Route "/admin/deleteemployee/{id}", dargestellt in Abbildung 49, verarbeitet HTTP-GET-Anfragen, um Mitarbeiter:innen anhand der angegebenen ID zu löschen. Wenn das Löschen fehlschlägt, wird eine Fehlermeldung angezeigt.

```
@GetMapping("/admin/insertemployeeform")
public ModelAndView insertEmployeeForm() {
    return new ModelAndView(viewName: "employee/insertemployeeform", modelName: "myemployee", new Employee());
}
```

Abbildung 50 - Controller-Endpunkt für die Ansicht zum Einfügen von Mitarbeiterinnen und Mitarbeitern

Bei HTTP-GET-Anfragen an "/admin/insertemployeeform" (Abbildung 50) wird das Formular zur Erfassung neuer Mitarbeiter:innen angezeigt.

```

@PostMapping("/admin/insertemployee")
public String insertEmployee(@Valid @ModelAttribute("myemployee") Employee employee, BindingResult bindingResult, Model model)
    throws ExecutionException, InterruptedException {
    if (bindingResult.hasErrors()) {
        model.addAttribute( attributeName: "bindingResult", bindingResult);
        return "employee/insertemployeeform";
    } else {
        try {
            this.employeeService.addEmployee(employee);
            return "redirect:/web/admin/allemployees";
        } catch (EmployeeAlreadyExistsException e) {
            System.out.println(e.getMessage());
            model.addAttribute( attributeName: "errortitle", attributeValue: "Mitarbeiter kann nicht eingefügt werden!");
            model.addAttribute( attributeName: "errormessage", e.getMessage());
            return "myerrorpage";
        }
    }
}

```

Abbildung 51 - Controller-Endpunkt zum Validieren und Speichern von Mitarbeiterinnen und Mitarbeitern

Im Gegensatz dazu erfolgt bei HTTP-POST-Anfragen an "/admin/insertemployee" (siehe Abbildung 51) die tatsächliche Erfassung einer Mitarbeiterin oder eines Mitarbeiters. Hier wird die Eingabe validiert, und im Falle einer ungültigen Eingabe werden entsprechende Fehlermeldungen angezeigt.

```

@GetMapping("/admin/editemployee/{id}")
public String editEmployeeForm(@PathVariable Long id, Model model) throws EmployeeNotFoundException, ExecutionException, InterruptedException {
    Employee employee = employeeService.getEmployeeById(id);
    if (employee != null) {
        model.addAttribute( attributeName: "employee", employee);
        return "employee/editemployeeform";
    } else {
        throw new EmployeeNotFoundException("Mitarbeiter nicht gefunden");
    }
}

```

Abbildung 52 - Controller-Endpunkt für die Ansicht für das Bearbeiten von Mitarbeiterinnen und Mitarbeitern

Die "editEmployeeForm()"-Methode in Abbildung 52 rendert das Formular zur Bearbeitung einer Mitarbeiterin oder eines Mitarbeiters.

```

@PostMapping("/admin/editemployee")
public String editEmployee(@Valid @ModelAttribute("employee") Employee updatedEmployee, BindingResult bindingResult, Model model)
    throws EmployeeNotFoundException {
    System.out.println("Validierung fehlgeschlagen: " + bindingResult.getAllErrors());
    if (bindingResult.hasErrors()) {
        System.out.println("Validierung fehlgeschlagen: " + bindingResult.getAllErrors());
        return "employee/editemployeeform";
    } else {
        try {
            this.employeeService.updateEmployeeById(updatedEmployee);
            return "redirect:/web/admin/allemployees";
        } catch (Exception e) {
            System.out.println(e.getMessage());
            model.addAttribute( attributeName: "errortitle", attributeValue: "Mitarbeiter bearbeiten fehlgeschlagen!");
            model.addAttribute( attributeName: "errormessage", e.getMessage());
            return "error";
        }
    }
}

```

Abbildung 53 - Controller-Endpunkt für die Aktualisierung von Mitarbeiterinnen und Mitarbeitern

Die "editEmployee()"-Methode, die in Abbildung 53 dargestellt ist, kümmert sich um die Verarbeitung und Aktualisierung der Mitarbeiterdaten.

### 6.1.5 Frontend

Im folgenden Abschnitt wird der Prozess zur Hinzufügung einer Mitarbeiterin oder eines Mitarbeiters erläutert.

In diesem Beispiel wird das Administrator-Konto verwendet, da nur Administratoren Mitarbeiter:innen verwalten können. Standardbenutzer sehen die Option zur Mitarbeiterverwaltung nicht, da sie nach erfolgreicher Anmeldung auf eine andere Startseite weitergeleitet werden.

Nach erfolgreichem Login der Administrierenden durch den LoginWebController, nimmt der "EmployeeWebController" (Abbildung 54) die Anfrage entgegen und rendert die entsprechende HTML-Seite.

```
@GetMapping("/admin/admin-start")
public ModelAndView home() throws ExecutionException, InterruptedException {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("employee/admin-start");
    return modelAndView;
}
```

Abbildung 54 - Controller-Endpunkt für die Ansicht der Admin-Startseite

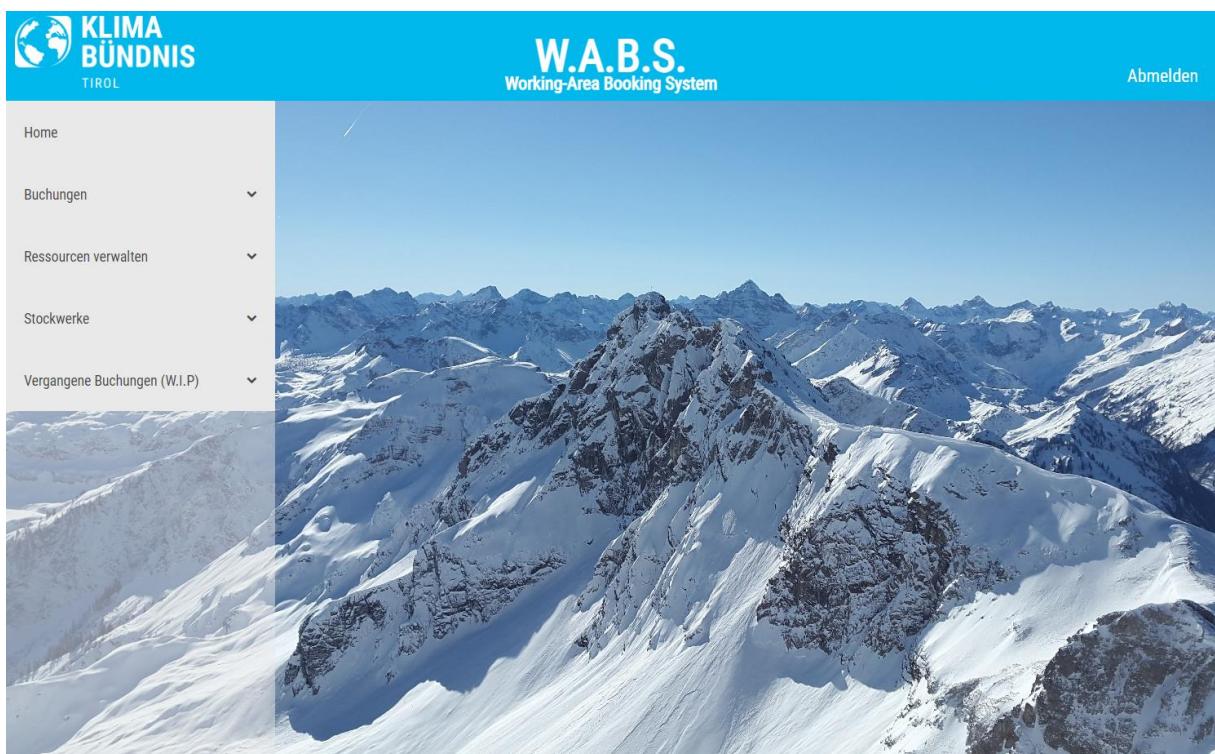


Abbildung 55 - Ansicht der Admin-Startseite

In der in Abbildung 55 gezeigten Startseite, kann sich der Administrator nun über die Navigationsbar durch verschiedene Register klicken und zum Beispiel "Ressourcen", "Räumlichkeiten" und "Mitarbeiter" verwalten.

Die zugehörige "admin-start.html" -Seite (Abbildung 56) wirkt recht minimalistisch, da Header, Top-Bar und Nav-Bar in separate HTML-Dateien ausgelagert wurden.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<div th:insert="headerAdmin :: headerAdmin" th:with="title='Admin-Startseite'"></div>
<body>
<!-- Top Bar -->
<div th:insert="topbarAdmin :: topbarAdmin"></div>
<!-- Side Navbar -->
<div th:insert="navbarAdmin :: navbarAdmin"></div>
<div class="filler"></div>
</body>
</html>
```

Abbildung 56 - HTML Seite für die Admin-Startseite

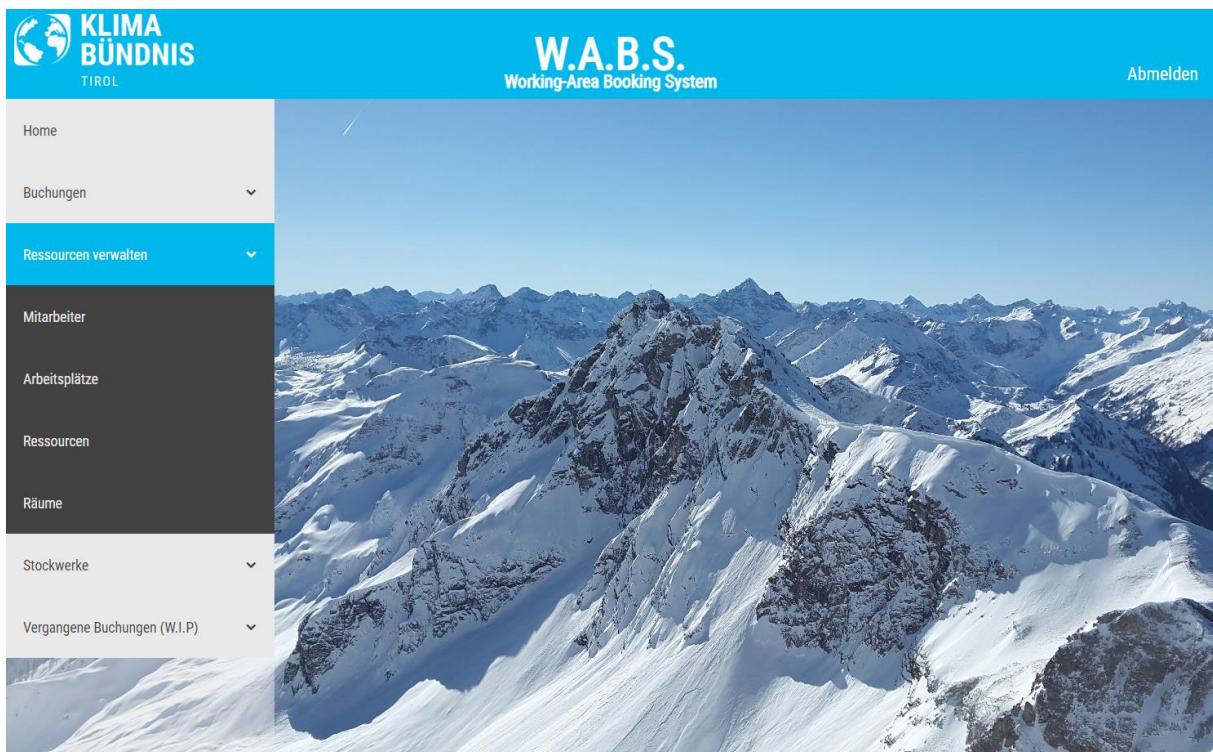


Abbildung 57 - Administrator Startseite Verwaltung von Ressourcen

Der Navigationsbereich am linken Bildschirmrand (Abbildung 57) wird ausgeklappt und der Bereich "Ressourcen verwalten" wird ausgewählt.

Sobald der Link "Mitarbeiter" betätigt wird, springt der "EmployeeWebController" mit dem Endpunkt "/web/admin/allemployees" an, nimmt die Anfrage entgegen und rendert die Ansicht (Abbildung 58).

```
<div class="contenedor-menu" th:fragment="navbarAdmin">
<!-- <a href="" class="btnMenu">Menu <i class="fa fa-bars"></i></a> -->

<ul class="menu">
    <li><a th:href="@{/web/admin/admin-start}">Home</a></li>
    <li><a href="#">Buchungen<i class="fa fa-chevron-down"></i></a>
        <ul>
            <li><a th:href="@{/web/deskbookings/admin}">Arbeitsplätze</a></li>
            <li><a th:href="@{/web/ressourceBooking/allBookings}">Ressourcen</a></li>
            <li><a th:href="@{/web/roomBooking/allBookings}">Räume</a></li>
        </ul>
    </li>
    <li><a href="#">Ressourcen verwalten<i class="fa fa-chevron-down"></i></a>
        <ul>
            <li><a th:href="@{/web/admin/allemployees}">Mitarbeiter</a></li>
            <li><a th:href="@{/web/desks}">Arbeitsplätze</a></li>
            <li><a th:href="@{/web/ressource/allRessources}">Ressourcen</a></li>
            <li><a th:href="@{/web/rooms/allRooms}">Räume</a></li>
        </ul>
    </li>
    <li><a href="#">Stockwerke<i class="fa fa-chevron-down"></i></a>
        <ul>
            <li><a th:href="@{/web/rooms/floors}">Stockwerk 1</a></li>
            <li><a th:href="@{/web/admin/admin-start}">Stockwerk 2 (W.I.P.)</a></li>
        </ul>
    </li>
    <li><a href="#">Vergangene Buchungen (W.I.P)<i class="fa fa-chevron-down"></i></a>
        <ul>
            <li><a href="@{/web/deskbookings/admin}">Arbeitsplätze (W.I.P)</a></li>
            <li><a href="@{/web/ressourceBooking/allBookings}">Ressourcen (W.I.P)</a></li>
            <li><a href="@{/web/roomBooking/allBookings}">Räume (W.I.P)</a></li>
        </ul>
    </li>
</ul>
</div>
```

Abbildung 58 - HTML-Seite für die Admin Navigationsleiste

Bevor der Controller die Ansicht rendern kann, muss er die aktuelle Mitarbeiterliste aus der Datenbank holen. Dazu ruft der Controller zunächst die "getAllEmployees()" Methode im "EmployeeService"-Interface auf (Abbildung 59).

```
@GetMapping("/admin/allemployees")
public ModelAndView allEmployees() throws ExecutionException, InterruptedException {
    List<Employee> allEmployees = employeeService.getAllEmployees();
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("employee/allemployees");
    modelAndView.addObject(attributeName: "employees", allEmployees);
    return modelAndView;
}
```

Abbildung 59 - Controller-Endpunkt für die Ansicht aller Mitarbeiter:innen

Wie in Abbildung 60 ersichtlich, gibt die Methode "allEmployees()" eine Liste mit Employees an den Aufrufer zurück. Diese Liste wird dann später auf der Website dargestellt.

```
List<Employee> getAllEmployees() throws ExecutionException, InterruptedException;
```

Abbildung 60 - Methode des Employee-Service Interfaces

Über das Interface wird die ausimplementierte Methode in der "EmployeeServiceImplementation"-Klasse aufgerufen.

```
@Override  
public List<Employee> getAllEmployees() throws ExecutionException, InterruptedException {  
    return this.employeeDBAccess.getAllEmployees();  
}
```

Abbildung 61 - Methode zum Abrufen aller Mitarbeiter:innen der EmployeeServiceImplementation-Klasse

Da es im Fall von "getAllEmployees()" keiner weiteren Geschäftslogik bedarf, wird der Aufruf (wie in Abbildung 61 ersichtlich) einfach an das EmployeeDBAccess-Interface weitergegeben.

```
List<Employee> getAllEmployees() throws ExecutionException, InterruptedException;
```

Abbildung 62 - Methode zum Abrufen aller Mitarbeiter:innen des EmployeeDBAccess-Interfaces

Auch bei der "getAllEmployees()"-Methode im EmployeeDBAccess-Interface (Abbildung 62) wird dem Aufrufer eine Liste mit Mitarbeiter:innen zurückgegeben.

```
@Override  
public List<Employee> getAllEmployees() {  
    return this.employeeJPARepo.findAll();  
}
```

Abbildung 63 - Methode zum Abrufen aller Mitarbeiter:innen der EmployeeDBAccess\_JPAH2-Klasse

Über das Interface gelangt der Aufruf schlussendlich in die Klasse "EmployeeDBAccess\_JPAH2" (Abbildung 63). Hier wird dann, mithilfe der JPA-Funktionalität, die Mitarbeiterliste aus der Datenbank geholt und zurück an die Controller-Klasse übermittelt. Die Methode "allEmployees()" (Abbildung 59) gibt die Mitarbeiterliste daraufhin an das entsprechende Template (Abbildung 64) weiter. Hier wird die Struktur der Anzeige für die Mitarbeiterdaten festgelegt.

```

<div class="content-container">
  <div class="allEmployee-Container">
    <h1>Mitarbeiterverzeichnis</h1><br><br>
    <div class="scroll-container">
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>Vorname</th>
            <th>Nachname</th>
            <th>Alias</th>
            <th>Rolle</th>
            <th>Aktion</th>
          </tr>
        </thead>
        <th:block th:each="employee : ${employees}">
          <tr>
            <td th:text="${employee.id}">...</td>
            <td th:text="${employee.fname}">...</td>
            <td th:text="${employee.lname}">...</td>
            <td th:text="${employee.nick}">...</td>
            <td th:text="#${strings.substringAfter(employee.role, '_')}">...</td>
            <td>
              <a th:href="@{/web/admin/editemployee/{id}(id=${employee.id})}" role="button" class="button">Bearbeiten</a>
              <a th:href="@{/web/admin/deleteemployee/{id}(id=${employee.id})}" role="button" class="delete-button">Löschen</a>
            </td>
          </tr>
        </th:block>
      </table>
    </div>
    <div class="button-row">
      <a class="add-button" th:href="@{/web/admin/insertemployeeform}">neuer Mitarbeiter</a>
      <button class="delete-button" onclick="goBack()">Zurück</button>
    </div>
  </div>
</div>
<div class="filler"></div>

```

Abbildung 64 - HTML-Seite alleemployees.html die eine Liste aller Mitarbeiter:innen zurückgibt

Anschließend wird eine Liste aller Mitarbeiter:innen (Abbildung 65) zurückgegeben.

The screenshot shows the W.A.B.S. (Working-Area Booking System) interface. At the top, there is a logo for "KLIMA BÜNDNIS TIROL" and the system name "W.A.B.S. Working-Area Booking System". On the right, there is a "Abmelden" (Logout) button. The left sidebar contains navigation links: Home, Buchungen, Ressourcen verwalten, Mitarbeiter (which is highlighted in blue), Arbeitsplätze, Ressourcen, Räume, Stockwerke, and Vergangene Buchungen (W.I.P.). Below the sidebar, the main content area is titled "Mitarbeiterverzeichnis" and displays a table of employees:

| ID | Vorname | Nachname | Alias     | Rolle      | Aktion   |
|----|---------|----------|-----------|------------|--|
| 1  | Patrick | Bayr     | operator  | OPERATOR   | <button>Bearbeiten</button> <button>Löschen</button> |
| 2  | Manuel  | Payr     | nemployee | N_EMPLOYEE | <button>Bearbeiten</button> <button>Löschen</button> |
| 3  | Sonja   | Lechner  | pemployee | P_EMPLOYEE | <button>Bearbeiten</button> <button>Löschen</button> |
| 4  | Marcel  | Schranz  | admin     | ADMIN      | <button>Bearbeiten</button> <button>Löschen</button> |
| 5  | Sonja   | Lechner  | sonlech   | ADMIN      | <button>Bearbeiten</button> <button>Löschen</button> |
| 6  | Jason   | Lechner  | jaslech   | P_EMPLOYEE | <button>Bearbeiten</button> <button>Löschen</button> |
| 7  | Josan   | Lechner  | joslech   | N_EMPLOYEE | <button>Bearbeiten</button> <button>Löschen</button> |
| 8  | Camil   | Lechner  | camlech   | OPERATOR   | <button>Bearbeiten</button> <button>Löschen</button> |

At the bottom of the table, there are two buttons: "neuer Mitarbeiter" (New Employee) and "Zurück" (Back).

Abbildung 65 - Liste aller Mitarbeiter:innen

# 7 Spring Security im Arbeitsplatzbuchungssystem

Die Integration von Spring Security in das Arbeitsplatzbuchungssystem wurde als strategischer Schritt vorgenommen und erwies sich als entscheidend für die Sicherheit und den Schutz der Anwendung und ihrer Ressourcen. Dieser Abschnitt erläutert die Gründe für die Verwendung von Spring Security und hebt einige seiner Hauptfunktionen und Vorteile hervor. Außerdem werden der Aufbau und die Funktion des Security-Moduls vorgestellt.

## 7.1 Warum Spring Security?

Die Entscheidung zur Verwendung von Spring Security in der Anwendung wurde aus mehreren wichtigen Gründen getroffen.

Eine Hauptaufgabe besteht darin, den Zugriff auf spezifische Teile der Anwendung zu beschränken, um sicherzustellen, dass ausschließlich autorisierte Benutzer:innen auf sensible Ressourcen zugreifen können. Dies ist besonders wichtig angesichts der vertraulichen Mitarbeiterdaten und ihrer Buchungsinformationen.

Ein weiterer essenzieller Aspekt ist die Möglichkeit der feingranularen Festlegung von Berechtigungen und Rollen für Benutzer:innen, mithilfe von Spring Security. Dies erweist sich als besonders wichtig im Arbeitsplatzbuchungssystem, in dem verschiedene Benutzer:innen unterschiedliche Berechtigungen besitzen, wie beispielsweise "*Administrator*", "*Operator*", "*privilegierte Mitarbeiter*" und "*normale Mitarbeiter*".

Darüber hinaus ermöglicht Spring Security die Implementierung eines Rechtesystems, das die Zuweisung spezifischer Rechte und Aufgaben an Benutzer:innen ermöglicht. Dieses System ist von grundlegender Bedeutung für die Verwaltung von Mitarbeiterberechtigungen und die effektive Steuerung des Zugriffs innerhalb der Anwendung.

## 7.2 Weitere Spring Security Funktionen

Neben den oben genannten Hauptfunktionen bietet Spring Security eine Vielzahl von Sicherheitsmechanismen und -funktionen:

- **Authentifizierung:** Spring Security ermöglicht die Implementierung verschiedener Authentifizierungsmethoden, darunter Benutzername/Passwort, Oauth (Open Authorization), LDAP (Lightweight Directory Access Protocol) und mehr.
- **Session-Management:** Spring Security bietet umfassende Funktionen zur Verwaltung von Benutzersitzungen, einschließlich der Kontrolle über Sitzungstimeouts und -validierungen.
- **Passwortverschlüsselung:** Das Framework bietet die Möglichkeit zur sicheren Verschlüsselung von Passwörtern, um die Sicherheit der Benutzerdaten zu gewährleisten.
- **Integration mit anderen Spring-Modulen:** Spring Security kann nahtlos in andere Spring-Module integriert werden, was die Entwicklung und Konfiguration vereinfacht.

- **Benutzerdefinierte Filter:** Die Möglichkeit, benutzerdefinierte Filter hinzuzufügen, ermöglicht die Anpassung der Sicherheitskonfiguration an spezifische Anforderungen.

### 7.3 Implementierung der Sicherheitskonfiguration

Die Klasse "WebSecurityConfig" (Abbildung 66) ist eine Spring-Konfigurationsklasse, die die Sicherheitseinstellungen für die Anwendung definiert. Sie ist verantwortlich für die Konfiguration von Spring Security, einschließlich Authentifizierung, Autorisierung und anderen Sicherheitsaspekten.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    private final EmployeeService employeeService;
```

Abbildung 66 - WebSecurityConfig-Klasse - Enthält die Sicherheitseinstellungen

Die "@Configuration"-Annotation wird auf Klassen angewendet, um anzugeben, dass sie Konfigurationsklassen sind.

Spring liest Konfigurationselemente aus Klassen mit dieser Annotation und erzeugt entsprechend Bean-Definitionen und Konfigurationen entsprechend.

```
public WebSecurityConfig(EmployeeService employeeService) {
    this.employeeService = employeeService;
}
```

Abbildung 67 - Konstruktor der WebSecurityConfig-Klasse

In Abbildung 67 erhält der Konstruktor der "WebSecurityConfig" eine Instanz des "EmployeeService", die für die Benutzerverwaltung verantwortlich ist. Diese Instanz wird zur Abfrage von Benutzerdetails genutzt.

```
@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Abbildung 68 - Diese Methode gibt ein Bean des Typs BCryptPasswordEncoder zurück

Die Methode in Abbildung 68 erstellt ein Bean, das einen "BCryptPasswordEncoder" generiert und zurückgibt. Dieser Encoder wird später zur Verschlüsselung von Benutzerpasswörtern verwendet.

```

@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .headers() HeadersConfigurer<HttpSecurity>
        .contentTypeOptions() ContentTypeOptionsConfig
        .disable() HeadersConfigurer<HttpSecurity>
        .and() HttpSecurity
        .authorizeHttpRequests(authConfig -> {
            authConfig.requestMatchers(HttpMethod.GET, "/web/login", "/web/deskbookings/add", "/error",
                "/web/login-error", "/web/logout", "/static/**", "/templates/**").permitAll();
            authConfig.requestMatchers(HttpMethod.POST, "/web/login", "/web/deskbookings/add").permitAll();
            authConfig.requestMatchers(HttpMethod.GET, "/web/deskbookings/**", "/web/deskbookings/mydeskbookings",
                "/web/deskbookings/add", "/web/deskbookings/view/**", "/web/deskbookings/add",
                "/web/deskbookings/new", "/web/deskbookings/update/**", "/web/deskbookings/deskbookinghistory/**",
                "/web/deskbookings/cancel/**", "/web/user/start", "/web/resourceBooking/allBookingsEmployee",
                "/web/resourceBooking/createBookingEmployee/**", "/web/resourceBooking/deleteBookingEmployee/**",
                "/web/deskbookings/mydeskbookings", "/web/resource/allRessourcesEmployee", "/web/rooms/allRoomsEmployee",
                "/web/roomBooking/allBookingsEmployee", "/web/roomBooking/createBookingEmployee/**")
                .hasAnyRole( ...roles: "ADMIN", "OPERATOR", "N_EMPLOYEE", "P_EMPLOYEE");
            authConfig.requestMatchers(HttpMethod.GET, "/web/**", "/web/desks/**").hasAnyRole( ...roles: "ADMIN", "OPERATOR");
            authConfig.requestMatchers(HttpMethod.POST, "/web/deskbookings/**", "/web/deskbookings/view/**",
                "/web/deskbookings/add", "/web/deskbookings/add", "/web/deskbookings/new", "/web/deskbookings/update",
                "/web/deskbookings/cancel/**", "/web/resource/**", "/web/roomBooking/**", "/web/resourceBooking/**",
                "/web/rooms/**", "/web/roomBooking/createBookingEmployee/**", "/web/roomBooking/updateBooking/**")
                .hasAnyRole( ...roles: "ADMIN", "OPERATOR", "N_EMPLOYEE", "P_EMPLOYEE");
            authConfig.requestMatchers(HttpMethod.POST, "/web/**", "/web/desks/**").hasAnyRole( ...roles: "ADMIN", "OPERATOR");
        })
}

```

Abbildung 69 - Sicherheitsregeln für HTTP-Anfragen

Der Abschnitt der "securityFilterChain()"-Methode, wie in Abbildung 69 vorgestellt, dient zur Konfiguration der Sicherheitsregeln für HTTP-Anfragen. Sie definiert, welche URLs und HTTP-Methoden für verschiedene Benutzerrollen ("ADMIN", "OPERATOR", "N\_EMPLOYEE", "P\_EMPLOYEE") zugänglich sein sollen.

- **permitAll()**: Wird verwendet, um öffentlichen Zugriff auf bestimmte URLs und Methoden zu ermöglichen.
- **hasAnyRole()**: Beschränkt den Zugriff auf bestimmte Rollen.

```

.formLogin(login -> {
    login.loginPage("/web/login")
        .failureHandler((request, response, exception) -> {
            String errorMessage = "Falsche Anmeldeinformationen.";
            response.sendRedirect( s: "/web/login-error" + errorMessage);
        })
        .successHandler((request, response, authentication) -> {
            Set<String> roles = AuthorityUtils.authorityListToSet(authentication.getAuthorities());
            if (roles.contains("ROLE_ADMIN")) {
                response.sendRedirect( s: "/web/admin/admin-start");
            } else if (roles.contains("ROLE_OPERATOR")) {
                response.sendRedirect( s: "/web/admin/admin-start");
            } else if (roles.contains("ROLE_N_EMPLOYEE")) {
                response.sendRedirect( s: "/web/user/start");
            } else if (roles.contains("ROLE_P_EMPLOYEE")) {
                response.sendRedirect( s: "/web/user/start");
            } else {
                response.sendRedirect( s: "/web/login-error");
            }
        });
}

```

Abbildung 70 - Formular-Login Konfigurationen

Der nächste Teil der Konfiguration (Abbildung 70) sorgt dafür, dass Benutzer:innen nach der Anmeldung basierend auf ihren Rollen auf die entsprechenden Seiten weitergeleitet werden. Dies ermöglicht es, verschiedenen Benutzergruppen unterschiedliche Bereiche und Funktionen der Anwendung bereitzustellen.

- **.formLogin(login -> { ... }):** Dieser Abschnitt konfiguriert das Formular-Login, bei dem Benutzer:innen ihre Anmeldeinformationen in einem Formular eingeben.
- **login.loginPage("/web/login"):** Hier wird die URL für die Login-Seite festgelegt. In diesem Fall ist die Login-Seite unter "/web/login" verfügbar. Wenn ein nicht authentifizierte:r Benutzer:in auf eine geschützte Ressource zugreift, wird er zur Login-Seite weitergeleitet.
- **login.failureHandler((request, response, exception) -> { ... }):** Hier wird ein Fehlerhandler für den Fall konfiguriert, dass die Anmeldung fehlschlägt. Wenn die Anmeldung fehlschlägt, wird die Benutzerin oder der Benutzer auf die Seite "/web/login-error" weitergeleitet, und die Fehlermeldung "Falsche Anmeldeinformationen" wird an die URL angehängt.
- **login.successHandler((request, response, authentication) -> { ... }):** Hier wird ein Erfolgshandler für den Fall konfiguriert, dass die Anmeldung erfolgreich ist. Nach einer erfolgreichen Anmeldung wird die Benutzerin oder der Benutzer basierend auf seinen Rollen (Authorities) auf verschiedene Seiten weitergeleitet.

```
.logout(logout -> {
    logout.logoutRequestMatcher(new AntPathRequestMatcher(✓ "/logout"));
    logout.logoutSuccessUrl("/web/login");
    logout.deleteCookies(...cookieNamesToClear: "JSESSIONID");
    logout.invalidateHttpSession(true);
});
```

Abbildung 71 - Konfiguration für das Abmelden von Mitarbeitern

Des Weiteren kümmert sich der Abschnitt in Abbildung 71 um das Abmelden und die Beendigung der Sitzung. Nach einer erfolgreichen Abmeldung werden die Benutzer:innen zur Login-Seite umgeleitet.

- **logout.logoutRequestMatcher(new AntPathRequestMatcher("/logout")):** Hier wird festgelegt, dass der Logout nur dann ausgelöst wird, wenn eine Anforderung an die URL "/logout" gesendet wird. Wenn ein/e Benutzer:in zu dieser URL navigiert, wird diese oder dieser automatisch ausgeloggt.
- **logout.logoutSuccessUrl("/web/login"):** Nach dem erfolgreichen Ausloggen werden die Benutzer:innen auf die Seite "/web/login" weitergeleitet. Dies ist die Seite, auf der sich Benutzer:innen erneut anmelden können.
- **logout.deleteCookies("JSESSIONID"):** Diese Zeile gibt an, dass das Cookie mit dem Namen "JSESSIONID" gelöscht werden soll, um die Sitzung zu beenden. Das JSESSIONID-Cookie wird normalerweise verwendet, um die Sitzung des/der Benutzer:in aufrechtzuerhalten.
- **logout.invalidateHttpSession(true):** Hier wird angegeben, dass die HTTP-Sitzung nach dem Ausloggen ungültig gemacht (invalidiert) werden soll. Dies stellt sicher, dass die Sitzungsinformationen nicht mehr verwendet werden können, nachdem der/die Benutzer:in sich ausgeloggt hat.

```

    return http.build();
}

```

Abbildung 72 - Hier wird die Konfiguration der Sicherheitseinstellungen abgeschlossen

Schließlich gibt der letzte Teil der Methode, wie in Abbildung 72 zu erkennen ist, ein "SecurityFilterChain()"-Objekt zurück und schließt somit die Konfiguration der Sicherheitseinstellungen ab.

```

@Bean
public AuthenticationManager authenticationManager(HttpSecurity httpSecurity, UserDetailsService userDetailsService,
                                                 BCryptPasswordEncoder bCryptPasswordEncoder) throws Exception {
    AuthenticationManagerBuilder authenticationManagerBuilder = httpSecurity.getSharedObject(AuthenticationManagerBuilder.class);
    authenticationManagerBuilder.userDetailsService(userDetailsService)
        .passwordEncoder(bCryptPasswordEncoder);
    return authenticationManagerBuilder.build();
}

```

Abbildung 73 - AuthenticationManger für die Authentifizierung

Die Methode in Abbildung 73 konfiguriert den "AuthenticationManager", der für die Authentifizierung der Benutzer:innen verantwortlich ist.

- **httpSecurity:** Ein "HttpSecurity"-Objekt, das von Spring Security verwaltet wird und die Sicherheitskonfiguration der Anwendung repräsentiert.
- **userDetailsService:** Ein "UserDetailsService"-Objekt, das für das Abrufen von Benutzerdetails aus der Datenbank oder anderen Quellen verwendet wird.
- **bCryptPasswordEncoder:** Ein "BCryptPasswordEncoder"-Objekt, das zum Verschlüsseln von Passwörtern verwendet wird.

```

@Bean
public UserDetailsService userDetailsService(BCryptPasswordEncoder bCryptPasswordEncoder) {
    return username -> {
        Employee employee = employeeService.getEmployeeByNick(username);
        if (employee == null) {
            throw new UsernameNotFoundException("Mitarbeiter mit dem Nick " + username + " nicht gefunden!");
        }
        return User.builder()
            .username(employee.getNick())
            .password(bCryptPasswordEncoder.encode(employee.getPassword()))
            .authorities(employee.getAuthorities())
            .build();
    };
}

```

Abbildung 74- Abrufen der Benutzerdaten und Erstellung eines User-Objekts für die Authentifizierung

In Abbildung 74 wird ein benutzerdefinierter "UserDetailsService" erstellt, um Benutzerdaten aus der Datenbank abzurufen. Dabei wird der "BCryptPasswordEncoder" verwendet, um die Gültigkeit des gespeicherten Passworts zu überprüfen. Wenn eine Mitarbeiterin oder ein Mitarbeiter gefunden wird, erstellt die Methode ein User-Objekt und gibt dieses zurück.

## 7.4 Implementierung des Login Controllers

Die Implementierung des Login-Controllers ist ein wichtiger Schritt bei der Verwendung von Spring Security. Hier werden Benutzeranmeldungen verarbeitet und Authentifizierungsversuche durchgeführt.

Folgender Code ist ein Beispiel für einen Spring MVC-Controller, dargestellt in Abbildung 75, der für die Verarbeitung von HTTP-Anforderungen im Zusammenhang mit der Benutzeranmeldung und Benutzeroauthentifizierung im Arbeitsplatzbuchungssystem verantwortlich ist.

```
@Controller
@RequestMapping("/web")
public class LoginWebController {
    private final AuthenticationManager authenticationManager;
```

Abbildung 75 - LoginWebController-Klasse

**@Controller** und **@RequestMapping("/web")**: Diese Annotationen deklarieren die Klasse "LoginWebController" als einen Spring MVC-Controller und legen den Basispfad ("/web") für die Verarbeitung der HTTP-Anforderungen fest.

```
public LoginWebController(AuthenticationManager authenticationManager) {
    this.authenticationManager = authenticationManager;
}
```

Abbildung 76 - Konstruktor der LoginWebController-Klasse

Der Konstruktor (Abbildung 76) erhält eine Instanz des "AuthenticationManager", der für die Benutzeroauthentifizierung verantwortlich ist. Der "AuthenticationManager" wird über Konstruktorinjektion bereitgestellt.

```
@GetMapping("/login")
public String showLoginForm(Model model) {
    model.addAttribute(attributeName: "error", attributeValue: "");
    return "login/login";
}
```

Abbildung 77 - Controller-Endpunkt für die Ansicht des Login-Formulars

In Abbildung 77 werden die HTTP-GET-Anforderungen für den Pfad "/web/login" verarbeitet. Die Methode "showLoginForm()" zeigt das Login-Formular an und gibt den Namen des HTML-Formulars ("login/login") zurück.

```

@PostMapping(value="/login")
public String processLoginForm(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(username, password);
    try {
        Authentication authentication = authenticationManager.authenticate(token);
        SecurityContextHolder.getContext().setAuthentication(authentication);
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        model.addAttribute(attributeName: "error", attributeValue: "Benutzername oder Passwort ungültig");
        return "login/login";
    }
    return null; // Rückgabewert kann null sein, Weiterleitung wird in der Security-Konfiguration festgelegt
}

```

Abbildung 78 - Controller-Endpunkt für die Authentifizierung von Mitarbeiter:innen

Die Methode "processLoginForm()", dargestellt in Abbildung 78, verarbeitet HTTP-POST-Anforderungen für den Pfad "/web/login". Sie verarbeitet das Login-Formular und authentifiziert die Benutzerin oder den Benutzer. Wenn die Authentifizierung erfolgreich ist, wird die Benutzerin oder der Benutzer auf die Begrüßungsseite weitergeleitet. Im Fehlerfall wird er auf die Login-Fehlerseite weitergeleitet.

```

@GetMapping(value="/logout")
public String logout(HttpServletRequest request) {
    request.getSession().invalidate();
    SecurityContextHolder.clearContext();
    return "login/login";
}

```

Abbildung 79 - Controller-Endpunkt für die Abmeldung eines/einer Mitarbeiter:in

Die in der Abbildung 79 gezeigte Methode "logout()", verarbeitet HTTP-GET-Anforderungen für den Pfad "/web/logout". Sie behandelt die Abmeldung des/der Nutzer:in, indem sie die Sitzung ungültig macht und den Sicherheitskontext löscht.

```

@GetMapping(value="/login-error")
public String loginError() {
    return "login/login-error";
}

```

Abbildung 80 - Controller-Endpunkt für die Login-Fehlerseite

Die Methode "loginError()" in Abbildung 80 zeigt die Login-Fehlerseite an, wenn während des Anmeldevorgangs ein Fehler aufgetreten ist.

Insgesamt ist der "LoginWebController" für die Anzeige des Login-Formulars, die Verarbeitung von Anmeldeversuchen und die Steuerung des Anmelde- und Abmeldevorgangs im Arbeitsplatzbuchungssystem verantwortlich. Er arbeitet eng mit Spring Security zusammen, um die Authentifizierung und Autorisierung der Benutzer:innen sicherzustellen.

#### 7.4.1 Exemplarische Darstellung des Login-Vorgangs

Im Folgenden wird die Login-Funktionalität der Applikation Schritt für Schritt beschrieben. Dazu wird zunächst die Login-Seite näher beleuchtet, die den Eintrittspunkt für Benutzer:innen darstellt.

Die Benutzer:innen öffnen einen Webbrowser und geben die URL "<http://localhost:8080/web/login>" in die Adressleiste ein.

Der Webbrowser sendet daraufhin eine HTTP-Anfrage an den Server, um die Login-Seite aufzurufen.

Im nächsten Schritt springt der Login-Controller mit der "showLoginForm()"-Methode (Abbildung 811) an, nimmt die Anfrage entgegen und verarbeitet sie.

```
@GetMapping("/login")
public String showLoginForm(Model model) {
    model.addAttribute("attributeName: "error", "attributeValue: "");
    return "login/login";
}
```

Abbildung 81 - Controller-Endpunkt für den Login

Der Controller rendert dann die in Abbildung 82 dargestellte "login.html" HTML-Seite.

```
<div class="content-container">
<div class="addResourceContainer">
    <h1>W.A.B.S.</h1>
    <h2>Work-Area Booking System</h2><br><br>
    <form method="post" th:action="@{/web/login}" th:error="${#fields.errors()}">
        <h3>Benutzername</h3>
        <div class="form-group">
            <label for="username"></label>
            <input type="text" class="form-control" name="username" id="username" placeholder="username" required>
        </div>
        <h3>Passwort</h3>
        <div class="form-group">
            <label for="password"></label>
            <input type="password" class="form-control" name="password" id="password" placeholder="Passwort" required>
        </div>
        <div th:if="${error}" class="alert alert-danger" role="alert">
            <span th:text="${error}"></span>
        </div>
        <div class="button-container">
            <button class="button" type="submit">Login</button>
        </div>
    </form>
</div>
</div>
```

Abbildung 82 - HTML-Seite für den Login-Vorgang

Das Formular enthält Felder für den Benutzernamen und das Passwort, sowie einen "Login"-Button.

Der vom Controller generierte HTML-Code (Abbildung 83) wird an den Browser zurückgesendet.

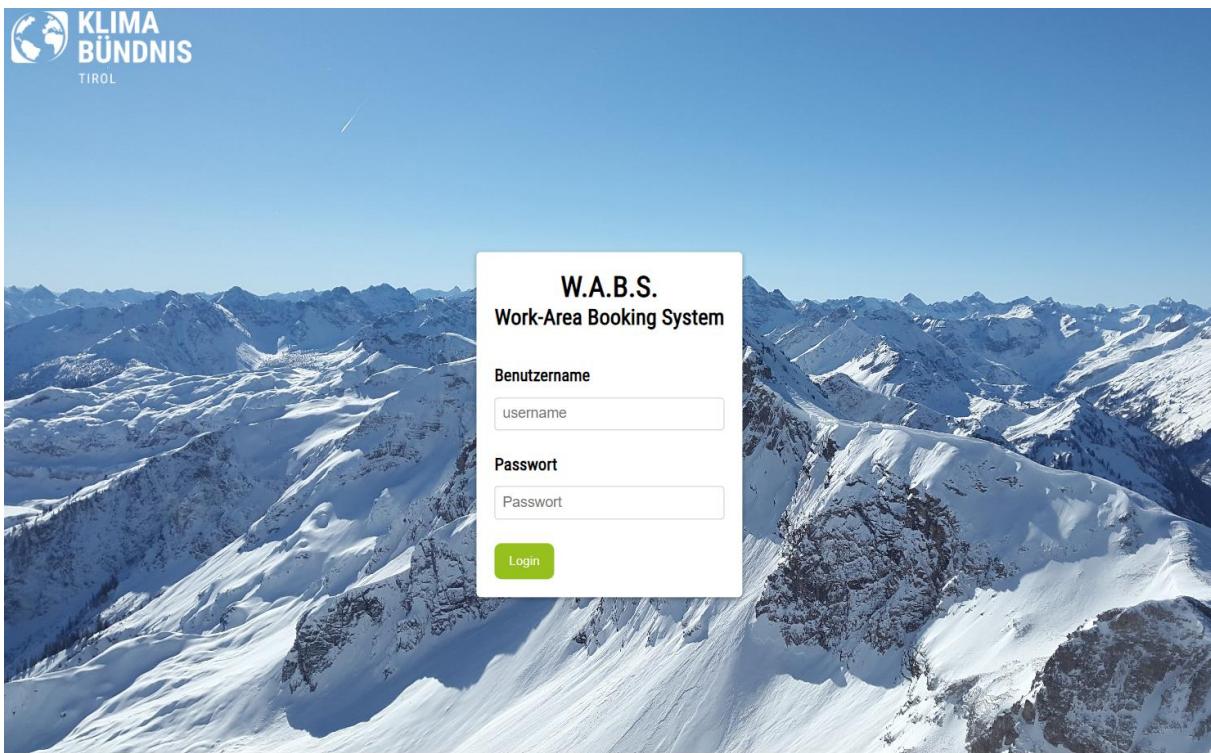


Abbildung 83 - Login-Seite der W.A.B.S.-Applikation

In Abbildung 84 wurden die Anmeldedaten in die entsprechenden Formularfelder eingegeben.

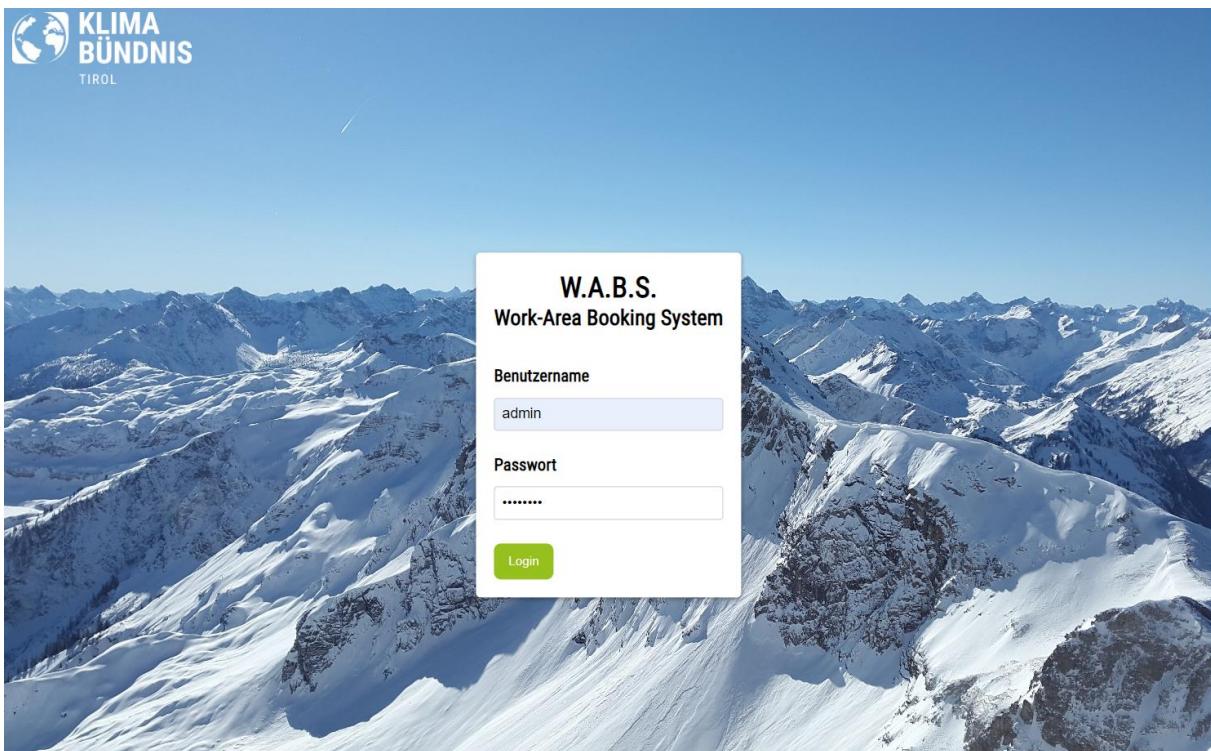


Abbildung 84 - Login mit korrekten Anmeldedaten

Sobald die Benutzerin oder der Benutzer den Login-Button betätigt, sendet der Webbrowser eine POST-Anfrage an den Server, die die eingegebenen Anmeldeinformationen enthält.

Der Login-Controller verarbeitet diese POST-Anfrage mithilfe der "processLoginForm()"-Methode (Abbildung 85) und überprüft die Gültigkeit der eingegebenen Anmeldeinformationen.

```
@PostMapping("/login")
public String processLoginForm(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(username, password);
    try {
        Authentication authentication = authenticationManager.authenticate(token);
        SecurityContextHolder.getContext().setAuthentication(authentication);
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        model.addAttribute(attributeName: "error", attributeValue: "Benutzername oder Passwort ungültig");
        return "login/login";
    }
    return null; // Rückgabewert kann null sein, Weiterleitung wird in der Security-Konfiguration festgelegt
}
```

Abbildung 85 - Die Methode authentifiziert die Mitarbeiter:innen

Bei ungültigen Anmeldeinformationen wird eine Fehlermeldung generiert und den Benutzer:innen angezeigt.

Wenn die Anmeldeinformationen gültig sind, authentifiziert der Controller die Benutzerin oder den Benutzer und gibt ihm Zugriff auf die entsprechende Startseite.

Die Konfiguration für die Weiterleitungen wurde in der "WebSecurityConfig"-Klasse (Abbildung 86) festgelegt, die auf Basis der Rolle des angemeldeten Benutzers die spezifischen Weiterleitungen ausführt.

```
.formLogin(login -> {
    login.loginPage("/web/login")
        .failureHandler((request, response, exception) -> {
            String errorMessage = "Falsche Anmeldeinformationen.";
            response.sendRedirect(s: "/web/login-error" + errorMessage);
        })
        .successHandler((request, response, authentication) -> {
            Set<String> roles = AuthorityUtils.authorityListToSet(authentication.getAuthorities());
            if (roles.contains("ROLE_ADMIN")) {
                response.sendRedirect(s: "/web/admin/admin-start");
            } else if (roles.contains("ROLE_OPERATOR")) {
                response.sendRedirect(s: "/web/admin/admin-start");
            } else if (roles.contains("ROLE_N_EMPLOYEE")) {
                response.sendRedirect(s: "/web/user/start");
            } else if (roles.contains("ROLE_P_EMPLOYEE")) {
                response.sendRedirect(s: "/web/user/start");
            } else {
                response.sendRedirect(s: "/web/login-error");
            }
        });
});
```

Abbildung 86 - Ausschnitt der securityFilterChain-Methode der WebSecurityConfig-Klasse

Die Anmeldung mit dem Administrator-Konto war erfolgreich, und die Weiterleitung zur Administrator-Startseite erfolgt. Der "EmployeeWebController" rendert die entsprechende Ansicht (Abbildung 87) und gibt sie an den Browser (Abbildung 88) zurück.

```
@GetMapping("admin/admin-start")
public ModelAndView home() throws ExecutionException, InterruptedException {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("employee/admin-start");
    return modelAndView;
}
```

Abbildung 87 - Controller-Endpunkt für die Ansicht der Administrator-Startseite

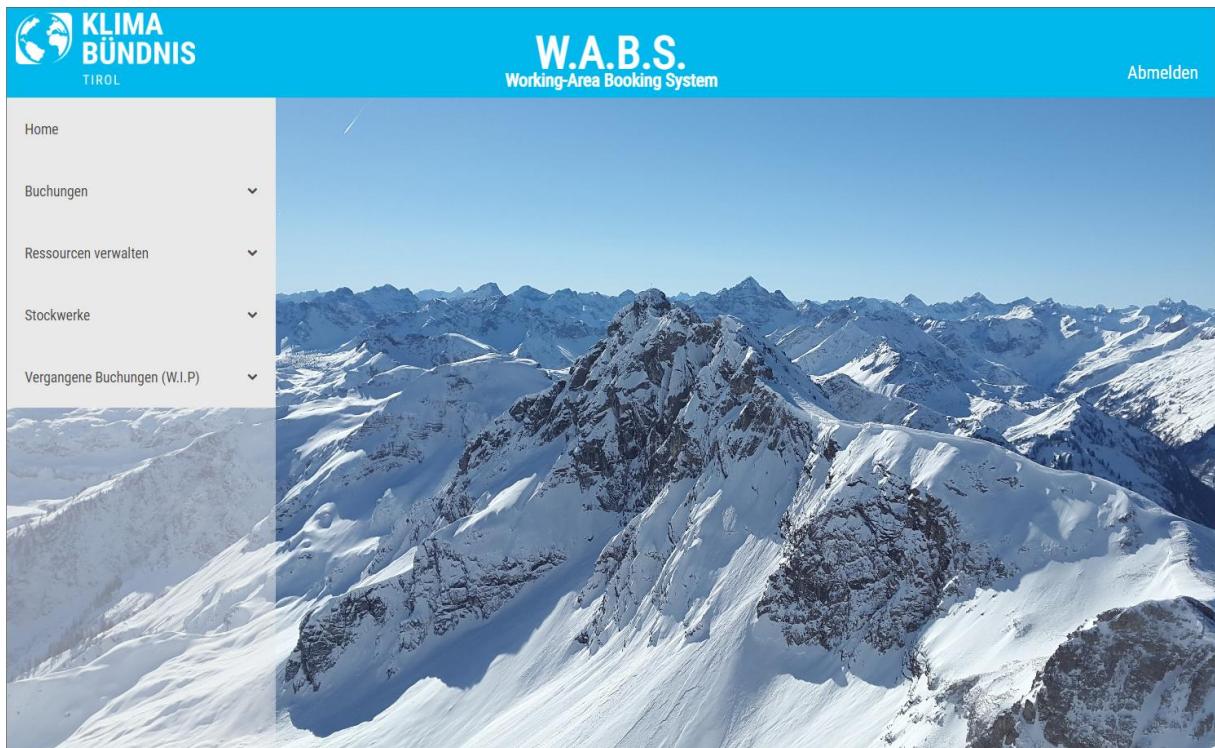


Abbildung 88 - Administrator-Startseite der W.A.B.S.-Applikation

Nach erfolgreichem Login können die Administrierenden verschiedene Ressourcen verwalten, darunter die Pflege von Mitarbeiterdaten, das Einsehen vergangener und zukünftiger Buchungen sowie die Durchführung anderer administrativer Aufgaben über die verfügbaren Links in der Navigationsleiste.

# 8 Ressourcenmodul

## 8.1 Einführung:

Der Aufgabenbereich von Herrn Payer bezieht sich auf das Ressourcenmodul, welches die Aufgabe hat, die unterschiedlichen Ressourcen bzw. die Geschäftsausstattung der Firma Klimabündnis Tirol zu verwalten. Dabei sollen Ressourcen, wie Kameras, Lastenfahrräder, Whiteboards, Beamer oder ähnlichem wie in einem Reservierungssystem von Benutzern über einen gewissen Zeitraum erstellt und gebucht werden können.

Neben dem Buchungsmodul ist das gesamte Frontend von Herrn Payer erstellt worden. Das Frontend baut auf die Nutzung von HTML5, CSS, Javascript, Springboot und Thymeleaf auf. Thymeleaf erfüllt jedoch lediglich den Zweck, dass Daten aus dem Backend an das Frontend übergeben werden. Eine genauere Beschreibung der Struktur sowie der Funktionsweise folgt in den anschließenden Abschnitten nach dem Backend.

## 8.2 Backend-Logik des Ressourcenmoduls

Nachstehend wird das Modul genau beschrieben und erklärt. Das Augenmerk liegt auf der Struktur, der Funktionsweise, markante Codeabschnitte und eine abschließende Zusammenfassung.

### 8.2.1 Struktur des Backends

Das Backend des Moduls baut auf ein Architekturmuster auf, welches unter dem Namen „Schichtenarchitektur/Layered Architecture“ bekannt ist. Dabei werden die Komponenten einer Anwendung in verschiedene Ebenen aufgeteilt, um die Struktur, Wiederverwendbarkeit und Wartbarkeit zu verbessern.

Es wurden weitere Technologien verwendet, welche im Abschnitt „8.3“ genauer erläutert werden.

Wie in Abbildung 89 - Package Struktur des Back-Ends zu sehen ist, werden die Schichten domains, repos und service verwendet. Die Pakete config sowie controller erfüllen den Aspekt der Logik und der Steuerung des Webcontrollers, welcher HTTP-Anfragen von einem Webbrower entgegennimmt, verarbeitet, koordiniert und wieder zurückleitet.

Bei Exceptions handelt es sich nicht um eine spezifische Schicht, sondern um ein Konzept der Softwareentwicklung, welches dazu dient, unerwartete oder fehlerhafte Zustände während der Laufzeit zu behandeln.

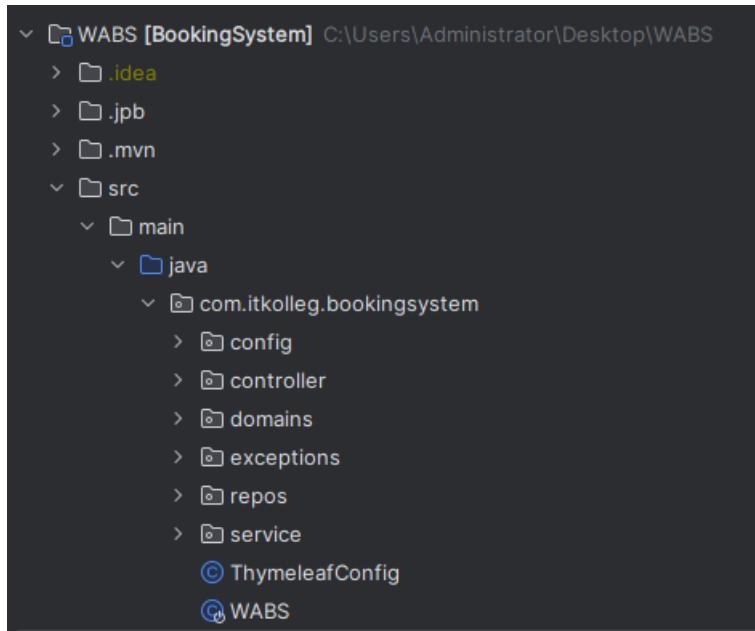


Abbildung 89 - Package Struktur des Back-Ends

Es folgt eine kurze Übersicht der Schichten:

### 8.2.2 Domain Layer

Dieser Layer, weiters auch als Data Layer umschrieben, beinhaltet die Kernlogik der Anwendung. Sie ist unabhängig von Frameworks, Datenbanken oder anderen technischen Details und bildet das Herzstück der Anwendung.

### 8.2.3 Repository Layer

Die Repository Schicht stellt eine Schnittstelle zwischen dem Data Layer und den darüber liegenden Schichten dar. Darüber liegende Schichten müssen nicht direkt mit dem Data Layer interagieren. Diese Schicht bietet Methoden zum Abfragen und Manipulieren von Daten an, ohne, dass die darunterliegende Datenbankstruktur bekannt sein muss.

### 8.2.4 Service Layer

Der Service Layer beinhaltet die Geschäfts-, sowie die Anwendungslogik. Es werden Dienste implementiert, die bestimmte Funktionalitäten der Anwendung bereitstellen. Sie koordiniert Interaktionen zwischen Domänenobjekten und verwendet dafür Daten aus der Repository Schicht.

In den jeweiligen Schichten befinden sich modulspezifische Klassen oder Interfaces. Diese Klassen/Interfaces beinhalten den Code, welcher in einem nachfolgenden Abschnitt beschrieben wird.

### 8.2.5 Funktionsweise

Es folgt eine Erklärung der Funktionsweise des Moduls. Es wird beschrieben, wie eine Benutzereingabe in der GUI an das System übergeben wird und wie es folglich durch das Ressourcen Modul verwaltet, manipuliert und weitergegeben wird.

Am besten lässt sich die Funktionalität durch eine Bilderstrecke beschreiben, beginnend mit einer Eingabe des Benutzers innerhalb der Grafischen Benutzer Oberfläche (GUI). Es wird besonders auf die Funktionalität des Hinzufügens einer Ressource eingegangen. Löschen, Bearbeiten, Buchen folgen einem ähnlichen Prinzip.

Innerhalb der Oberfläche kann der Benutzer einer Ressource, wie in der Abbildung 90 **Fehler! Verweisquelle konnte nicht gefunden werden.** zu sehen ist, erstellen. Diese Funktionalität ist jedoch nur für administrative Rollen zugänglich. Nichtadministrative Rollen können die Ressourcen lediglich buchen, aber nicht manipulieren.

| ID | Name  | Kategorie | Beschreibung | Info       | Seriennummer | Aktion                  |                             |                          |
|----|-------|-----------|--------------|------------|--------------|-------------------------|-----------------------------|--------------------------|
| 1  | Test1 | BEAMER    | TestBeamer   | InfoBeamer | BeamerSN     | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 2  | Test2 | BEAMER    | TestBeamer   | InfoBeamer | BeamerSN     | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 3  | Test3 | BEAMER    | TestBeamer   | InfoBeamer | BeamerSN     | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 4  | Test4 | BEAMER    | TestBeamer   | InfoBeamer | BeamerSN     | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |

neue Ressource   Zurück

Abbildung 90 - Ressourcenliste im Front-End

Das Betätigen des Buttons „neue Ressource“ leitet dann den/die Benutzer:in an eine andere Seite weiter, auf der dann die Daten für die neue Ressource eingegeben werden können.

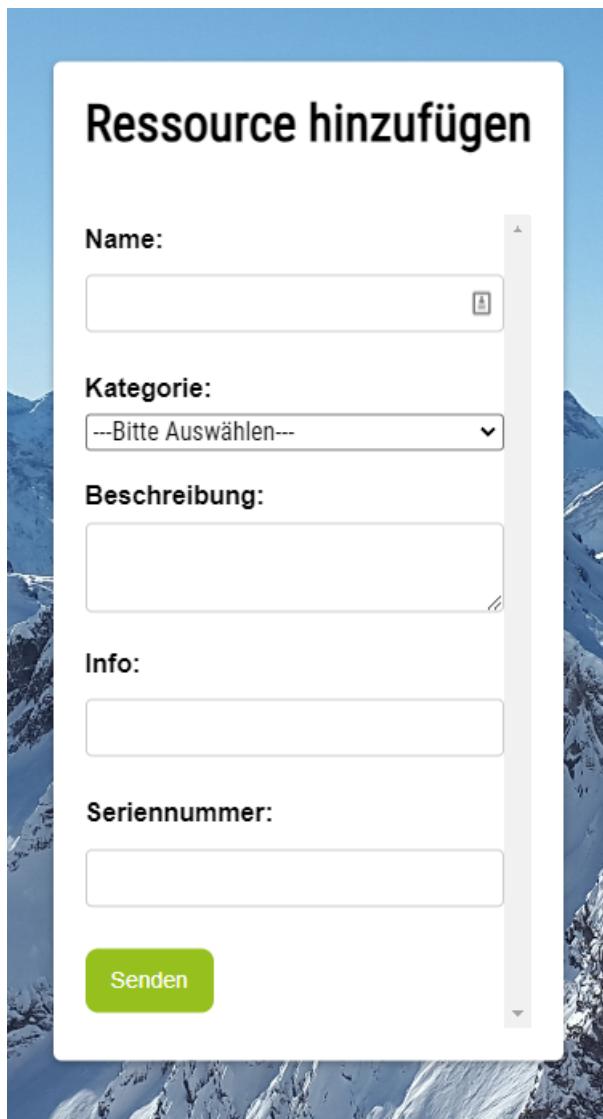


Abbildung 91 - Ressourcen GUI im Front-End

Wie in der obenstehenden Abbildung 91 zu sehen ist, werden unterschiedliche Daten erwartet. Der Name ist selbsterklärend und sollte sinnvoll gewählt werden. Die Kategorie greift auf eine ENUM-Liste im Backend zu, in der die unterschiedlichen Kategorien, die eine Geschäftsausstattung haben kann, angezeigt werden.

Derzeit ist es für den/die Benutzer:in noch nicht möglich, eine neue Kategorie über die GUI zu erstellen. Diese Funktionalität ist noch in Bearbeitung. Sie gehört allerdings nicht zu den angeforderten funktionalen Anforderungen des Partners und wird daher im Praktikum implementiert.

Auf Wunsch des Projektpartners kann man bei jeder Ressource eine optionale Beschreibung hinzufügen. Diese Beschreibung soll dazu dienen, dass man Anmerkungen zur Ressource hinterlegen kann. Ein sinnvolles Beispiel wäre das Hinterlegen der Information, ob die Ressource irgendwelche Mängel aufweist.

Das Datenfeld „Info“ ist eine weitere angeforderte Funktionalität und soll Informationen speichern, die die Ressource betreffen. Zum Beispiel: „Kamera muss bitte vor Benutzung immer auf Updates überprüft werden.“

Die Seriennummer dient dazu, die jeweilige Geschäftsausstattung einzigartig zu definieren. Es können zwar dieselben Modelle sein, aber jedes Gerät hat eine einzigartige Seriennummer/IMEI.

Wenn der/die Benutzer:in zufrieden mit den eingegebenen Daten ist, werden die Informationen über den Button „Senden“ an die Datenbank übergeben. Zu demonstrativen Zwecken wurde eine solche Ressource erstellt (siehe Abbildung 92 - Liste aller Ressourcen in der Datenbank).

| ID | Name              | Kategorie | Beschreibung      | Info       | Seriennummer       | Aktion                  |                             |                          |
|----|-------------------|-----------|-------------------|------------|--------------------|-------------------------|-----------------------------|--------------------------|
| 1  | Test1             | BEAMER    | TestBeamer        | InfoBeamer | BeamerSN           | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 2  | Test2             | BEAMER    | TestBeamer        | InfoBeamer | BeamerSN           | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 3  | Test3             | BEAMER    | TestBeamer        | InfoBeamer | BeamerSN           | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 4  | Test4             | BEAMER    | TestBeamer        | InfoBeamer | BeamerSN           | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |
| 5  | DokumentationTest | BEAMER    | Dient zur Testung | Testung    | 123456789123456789 | <button>Buchen</button> | <button>Bearbeiten</button> | <button>Löschen</button> |

Abbildung 92 - Liste aller Ressourcen in der Datenbank

Das Feld ID kann nicht manuell gefüllt werden, da die ID durch eine autoinkrementelle Funktion im Code sequenziert wird. Diese ID ist einzigartig und kann nicht verändert werden.

Von dieser Liste aus können dann weitere Operationen durchgeführt werden. Das Bearbeiten einer Ressource hat den Zweck, eine bestehende Ressource zu löschen und sie mit neuen Daten als neue Ressource zu erstellen. Das bedeutet, dass die Ressource nicht bearbeitet wird, sondern die Daten werden an ein neues Objekt übergeben und angelegt.

Folgend wird der Prozess im Backend betrachtet. Davor ist wichtig zu erwähnen, wie genau die Eingaben im Webbrowser an die Datenbank übergeben werden. Dazu bezieht sich dieser Abschnitt lediglich auf den Namen der Ressource.

In der nachfolgenden Abbildung 93 ist ein Ausschnitt des Frontends zu sehen. Das soll visuell klarstellen, wie die Eingabe der Ressource übernommen wird.

```
<div class="content-container">
  <div class="addRessourceContainer">
    <h1>Ressource hinzufügen</h1><br><br>
    <div class="scroll-container">
      <form th:action="@{/web/ressource/addRessource}" th:object="${newRessource}" method="post">
        <!-- zuerst objekt holen, mapping wohin, Übertragungsmethode -->
        <div class="form-group">
          <label for="name">Name:</label>
          <input type="text" class="form-control" id="name" name="name" th:field="*{name}" required>
        </div>
```

Abbildung 93 - Codesnippet aus der addRessource.html

Das Augenmerk liegt auf der Zeile: `<input type="text" class="form-control" id="name" name="name" th:field="*{name}" required>`.

Der Abschnitt `th:field="*{name}"` ist die Variable, die den eingegebenen Namen im Formular entgegen nimmt. Dieser Wert wird dann an die Schichten weitergegeben. Die Service Schicht prüft dabei die Geschäftslogik, gibt den Wert dann an die Repository Schicht weiter, welche die Aufgabe hat, als Schnittstelle zu agieren und die Daten dann weiter an die Datenbank zu übergeben. Dabei wird die Save-Methode aufgerufen, die diesen Schritt realisiert.

So werden die Daten des Webbrowsers an die Anwendung und folglich an die Datenbank übergeben.

## 8.3 Implementierung

Es folgen markante Codeabschnitte aus dem Modul „Ressourcenverwaltung“ sowie detailliertere Beschreibungen der implementierten Klassen. Weiters wird auf die verwendeten Technologien Bezug genommen. Vorerst wird auf die Domäenschicht eingegangen.

### 8.3.1 Technologien im Backend:

**Jakarta:** Jakarta ist eine Sammlung von Projekten, welche sich auf die Entwicklung von Open-Source-Softwares bezieht. Durch Jakarta lassen sich Webanwendungen sowie Datenbankzugriffe fördern.

In der Abbildung 94 sieht man die zum Einsatz gekommenen Pakete, die die Domain Klasse verwendet.

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;
```

Abbildung 94 - Eingebundene Pakete der Domain Class: Ressource

Die Einbindung dieser Pakete ermöglicht die Verwendung einer Java Persistence API (JPA) Annotation, wie zum Beispiel „@ID“. Diese Annotation ermöglicht eine unkomplizierte Deklaration einer Variable zu einer ID (bevorzugt vom Datentyp Long). Der Kompilierer ist dann im Stande, bei der Ausführung zu erkennen, welche der Variablen als ID bestimmt ist. Darauf aufbauend wurde ergänzend die Annotation **@GeneratedValue Sequence** verwendet. Bei der Erstellung und Vergabe der ID folgt der Algorithmus nun einer numerischen Sequenz und sorgt dafür, dass die ID vom System generiert wird, aber der Zahlenreihe folgt.

Eine ID sollte in der Regel einzigartig identifiziert sein und man sollte sie nachträglich nicht ändern können. Im Zuge der Webanwendung kann eine ID nicht manueller verändert werden.

Verantwortlich für den Inhalt: Manuel Payer

**Lombok:** Die Funktionalität von Lombok beschreibt ein Java-Framework, welches entwickelt wurde, um den Prozess des Schreibens von Codeduplikaten zu reduzieren. Dadurch wird die Anwendung effizienter und lesbarer, indem es viele typische Aufgaben, wie das Erstellen von Gettern, Settern,

Konstruktoren automatisiert. Dabei ist zu berücksichtigen, dass die Setter Methoden, welche generiert werden, lediglich eine Grundfunktionalität bieten. Eine etwaige Business Logik oder Validierungsprüfung ist allerdings nicht inbegriffen.

In der obenstehenden Abbildung 94 - Eingebundene Pakete der Domain Class: Ressource sieht man, dass üblich vorkommende Methoden, wie Getter, Setter, ToString sowie NoArgsConstructor von Lombok übernommen werden. Es ist zu erwähnen, dass einNoArgsConstructor einen parameterlosen Konstruktor erstellt. Das Gegenstück dazu wäre der AllArgsConstructor.

Wie in der Abbildung 95 zu sehen ist, existiert dennoch ein parametrisierter Konstruktor in der Klasse. Im Grunde würde das die Funktionalität eines AllArgsConstructor beschreiben, dessen Verwendung würde aber eine Redundanz erzeugen.

```
/*
 * Konstruktor der Klasse Ressource. Er nimmt folgende Parameter entgegen:
 *
 * @param id          vom Typ Long
 * @param ressourcetype vom Typ Ressourcetype
 * @param name        vom Typ String
 * @param description  vom Typ String
 * @param info         vom Typ String
 * @param serialnumber vom Typ String
 */
▲ pabayr+2
public Ressource(Long id, Ressourcetype ressourcetype, String name, String description, String info, String serialnumber) {
    this.id = id;
    this.ressourcetype = ressourcetype;
    this.description = description;
    this.info = info;
    this.name = name;
    this.serialnumber = serialnumber;
}

}
```

Abbildung 95 - Konstruktor der Domain Class: Ressource

**SpringBeans:** Beans sind verwaltete Komponenten innerhalb eines Spring-Containers. Es sind Klassen, die vom Container instanziert, verwaltet und konfiguriert werden. Sie können Dienste, Geschäftslogiken oder andere Komponenten repräsentieren. Diese Beans können über eine Dependency Injection miteinander verbunden werden, wodurch eine lose Kopplung zwischen Komponenten erreicht wird. Diese Technologie fördert die Wartbarkeit, Erweiterbarkeit sowie die Modularität von Anwendungen.

Der zugehörige Quellcode wird in der Abbildung 96 - Abbildung einer Bean vor einer Methode, dargestellt.

```
▲ manpayer+1
2  public RessourceServiceImplementation(DBAccessRessource dbAccessRessource) {
3      this.dbAccessRessource = dbAccessRessource;
4  }
```

Abbildung 96 - Abbildung einer Bean vor einer Methode

## Domäne Ressource und Ressourcetype

Zusammengefasst übernimmt die beschriebene Klasse „Ressource“ die Kernlogik der Anwendung und befindet sich in der Domänen Schicht.

In der selbigen Schicht befindet sich zusätzlich die Klasse „Ressourcetype“. Diese Klasse ist eine ENUM Klasse und dient lediglich dazu, dass bei der Auswahl der Kategorie einer Ressource, wie in der Abbildung 97 zu sehen ist, vordefinierte Werte auszuwählen. Derzeit ist die Klasse jedoch noch ausbaufähig, da sie aus Sicht des/der Benutzer:in ein Hinzufügen einer neuen Kategorie nicht unterstützt. Diese Funktionalität wird im Praktikum bei der Firma Klimabündnis Tirol überarbeitet.

In der Repository Schicht des Moduls existieren je zwei verschiedene Packages, die unterschiedliche Funktionalitäten der Anwendung erfüllen. Jeder dieser Packages sind vom Aufbau identisch und verwenden dieselbe Struktur der Klassen, jedoch mit unterschiedlichen Logiken.

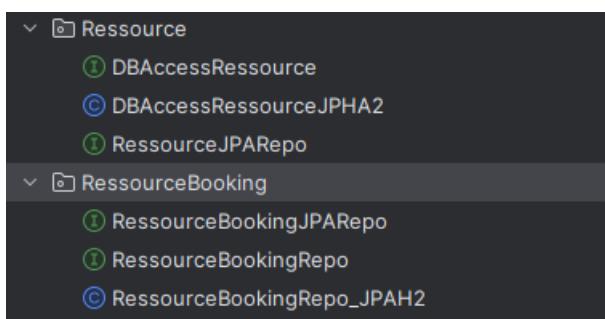


Abbildung 97 - Struktur der Repository Schicht

Dabei ist zu beachten, dass die Packages je zwei Interfaces und eine gewöhnliche Klasse beinhalten. RessourceJPARepo wird um das JPARepository erweitert und beinhaltet damit bereits eine Vielzahl von Methoden, die vom Paket bereitgestellt werden. Hier wurde lediglich eine Methode deklariert, die dazu dienen soll, eine Ressource anhand ihrer ID ausfindig machen zu können.

Im Gegensatz dazu besitzt das DBAccessRessource-Interface mehrere Methoden, die dazu dienen, CRUD-Operationen, die unmittelbar auf die Datenbank zugreifen, zu deklarieren. Dieses Interface wird von der Klasse DBAccessRessourceJPHA2 implementiert. Dort werden die deklarierten Methodenköpfe aus dem Interface mit einer Business Logik sowie einer Funktionalität ausimplementiert.

Die meisten Methoden in dieser Klasse besitzen eine Validierungsprüfung sowie ein Exceptionhandling.

Hier ein Auszug der Methode „getRessourceById“ (siehe Abbildung 98 - Methode mit Validierungsprüfung):

```

/**
 * Diese Methode gibt eine Ressource anhand der mitgegebenen ID zurück. Sie benötigt folgende Parameter:
 *
 * @param id vom Typ Long
 * @return Liste als Optional von Ressourcen
 * @throws RessourceNotFoundException Ressource nicht gefunden
 */
1 usage  ↗ manpayer +1
@Override
public Ressource getRessourceById(Long id) throws RessourceNotFoundException {
    Optional<Ressource> ressourceOptional = this.ressourceJPARepo.findById(id);
    if (ressourceOptional.isPresent()) {
        return ressourceOptional.get();
    } else {
        throw new RessourceNotFoundException();
    }
}

```

Abbildung 98 - Methode mit Validierungsprüfung

Wie auch in der Repository-Schicht besitzt die Service-Schicht ebenfalls zwei Packages, die jeweils zwei Klassen beinhalten.

Wie bereits zuvor besitzt jedes Package ein Interface, in dem die Methoden Köpfe (Byte-Welt, 2023) deklariert wurden, welche in der Klasse „RessourceServiceImplementation“/„RessourceBookingServiceImplementation“ mit einer Geschäftslogik ausimplementiert wurden.

In dieser Klasse wird die Funktionalität der Methoden ausimplementiert. Im Interface wurden bei der Deklaration der Methodenköpfe bereits auch die möglichen Fehlerbehandlungen eingebaut, die während der Laufzeit geworfen und abgefangen werden. Das ist wichtig, um sicherzustellen, dass die Anwendung aufgrund eines Fehlers nicht abbricht und neugestartet werden muss.

### 8.3.2 Ressource-Webcontroller

Diese Klasse repräsentiert einen Web Controller für das Modul Ressourcen. Der Controller ermöglicht die Interaktion mit Ressourcen über HTTP-Anfragen und Antworten.

Zusammengefasst ist diese Klasse dafür zuständig, den/die Benutzer:in beim Ausführen einer Aktion auf die entsprechende Seite weiterzuleiten, indem der Controller die HTTP-Anfragen des Webclients entgegennimmt und diese dann beantwortet.

In der nachstehenden Abbildung 99 - Methode im WebController ist zu sehen, wie eine Methode im WebController aussieht. Dabei richtet sich ein besonderes Augenmerk auf die URL, welche mit der Annotation @GetMapping hervorgehoben wird.

```

@Controller
@RequestMapping(value="/web/ressource")
public class RessourceWebController {

    7 usages
    RessourceService ressourceService;

    /**
     * Konstruktor der Klasse RessourceWebController. Sie nimmt eine Variable (ressourceService) vom Typ RessourceService entgegen und weiß den Wert
     * dem globalen Datenfeld hinzu
     *
     * @param ressourceService vom Typ RessourceService
     */
    ± pwnz
    public RessourceWebController(RessourceService ressourceService) { this.ressourceService = ressourceService; }

    /**
     * Dient dazu eine Übersicht aller Ressourcen für den/die Admin zu liefern.
     *
     * @return modelAndView
     * @throws ExecutionException  Aufführungsfehler
     * @throws InterruptedException Unterbrechungsfehler
     */
    ± pwnz
    @GetMapping(value="/allRessources")
    public ModelAndView allRessources() throws ExecutionException, InterruptedException {
        List<Ressource> allRessources = ressourceService.getAllRessource();
        return new ModelAndView(viewName: "ressource/allressources", modelName: "ressources", allRessources);
    }
}

```

Abbildung 99 - Methode im WebController

Dazu muss erwähnt werden, dass die Klasse selbst mit Annotationen versehen wurde. Diese Annotationen haben – von oben beginnend - folgende Bedeutung:

**@Controller** – markiert eine Klasse als eine Controller Klasse. Dadurch weiß der Kompilierer, wozu diese Klasse mit der dazugehörigen Annotation dient.

**@RequestMapping** – Bildet die Standard URL für das Modul Ressource/RessourceBooking. Das Aufrufen einer Seite, wie allRessources wird dann einfach an die Standard URL konkateniert. Daraus ergibt sich dann folgende URL: .../web/ressource/allRessources.

**@GetMapping** – wird speziell für HTTP-GET-Anfragen verwendet. Methoden, die mit dieser Annotation versehen sind, können folglich nur auf HTTP-GET-Anfragen reagieren. Sie akzeptiert eine oder mehrere URL-Pfade/Muster, die angeben, auf welche URLs die annotierte Methode reagieren soll.

**@PostMapping** – Ähnlich zur GetMapping-Annotation nimmt die @PostMapping Annotation nur HTTP-POST-Anfragen entgegen. Eine Abbildung mit der @PostMapping Annotation befindet sich im anschließenden Abschnitt.

Auch für die Klasse „RessourceBooking“ existiert ein WebController. Im Wesentlichen hat er dieselben Aufgaben, wie der Controller für die Klasse „Ressource“, folgt jedoch einer anderen Geschäftslogik.

```
@PostMapping(value="/updateBooking")
public String updateBooking(@Valid RessourceBooking booking, BindingResult bindingResult, RedirectAttributes redirectAttributes) throws RessourceAlreadyExistsException

    try {
        if (bindingResult.hasErrors()) {
            return "redirect:/web/ressourceBooking/updateBooking/" + booking.getId();
        } else {
            this.ressourceBookingService.updateBooking(booking);
            return "redirect:/web/ressourceBooking/allBookings";
        }
    } catch (RessourceNotAvailableException e) {
        redirectAttributes.addFlashAttribute("errorMessage", "Ressource in diesem Zeitraum nicht verfügbar");
        return "redirect:/web/ressourceBooking/updateBooking/" + booking.getId();
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("errorMessage", "Ressource konnte nicht gebucht werden");
        return "redirect:/web/ressourceBooking/updateBooking/" + booking.getId();
    }
}
```

Abbildung 100 - CodeSnippet aus dem WebController

Die gezeigte Methode ist eine CRUD-Methode, die das Updaten/Bearbeiten von entgegengenommenen HTTP-POST-Anfragen bzw. eingegebenen Parametern entgegennimmt und eine bestehende Ressourcenbuchung überschreibt.

Eine Prüfung der eingegebenen Daten des/der Benutzer:in erfolgt über BindingResults. Sie prüft auf etwaige Bindungs- und Validierungsfehler. Mit einem Try-Catch-Block wird geprüft und festgelegt, was passieren soll, wenn ein solcher Bindungs- bzw. Validierungsfehler auftritt.

Hier ist auch eine weitere Annotation zu sehen. Nämlich die @Valid-Annotation. Sie ermöglicht eine Signalisierung, dass die beantragten Daten validiert werden sollen, bevor diese in die Anwendungslogik übergeben werden.

RedirectAttributes hilft dabei, Daten sicher zwischen verschiedenen Anfragen weiterzugeben, insbesondere wenn eine Weiterleitung erfolgt. Nachrichten, Fehlermeldungen oder andere Informationen werden von einer Controller-Methode zur nächsten übertragen, ohne die Parameter in der URL sichtbar zu machen. So werden saubere und benutzerfreundliche URLs beibehalten und gleichzeitig werden wichtige Informationen übergeben.

In diesem Fall, soll die Redirection erfolgen, wenn bindingResult.hasErrors auf „true“ steht. Wenn dies der Fall ist, soll eine Weiterleitung auf eine neue/andere oder gleiche URL erfolgen.

## 8.4 Zusammenfassung des Ressourcen- und RessourceBooking-Moduls

Herr Manuel Payer ist für das Ressourcenmodul sowie für das Ressourcen-Buchungs-Modul verantwortlich, welche die Verwaltung der Geschäftsausstattung der Firma Klimabündnis Tirol ermöglichen. Die Module ermöglichen die Erstellung und Buchung von Ressourcen wie Kameras, Lastenfahrräder, Whiteboards und Beamern über einen bestimmten Zeitraum.

Das Backend der Module folgt einer Schichtenarchitektur und verwendet Technologien wie Jakarta, Lombok und SpringBeans. Die Schichten umfassen Domain, Repository und Service. Das Frontend wurde ebenfalls von Herrn Payer erstellt und nutzt HTML5, CSS, JavaScript, Spring Boot und Thymeleaf.

Die Funktionsweise des Moduls umfasst die Erfassung von Benutzereingaben über die GUI, die Verarbeitung durch das Ressourcenmodul und die Übertragung an die Datenbank. Es gibt auch eine Beschreibung der verschiedenen Datenfelder, Validierungen und Ausnahmehandlungen.

Das Backend verwendet Jakarta für die Java Persistence API (JPA) und Lombok zur Vereinfachung der Codeerstellung. SpringBeans werden für die Verwaltung von Komponenten verwendet.

Die Repository-Schicht enthält Pakete für die Verwaltung von Ressourcen und Ressourcenbuchungen. Die Service-Schicht implementiert die Geschäftslogik und der Web-Controller ermöglicht die Interaktion über http-Anfragen und -Antworten.

Es gibt auch Erklärungen zu Annotationen wie @GetMapping, @PostMapping, @Valid und RedirectAttributes, die für die Steuerung der Anfragen und die Datenübertragung verwendet werden.

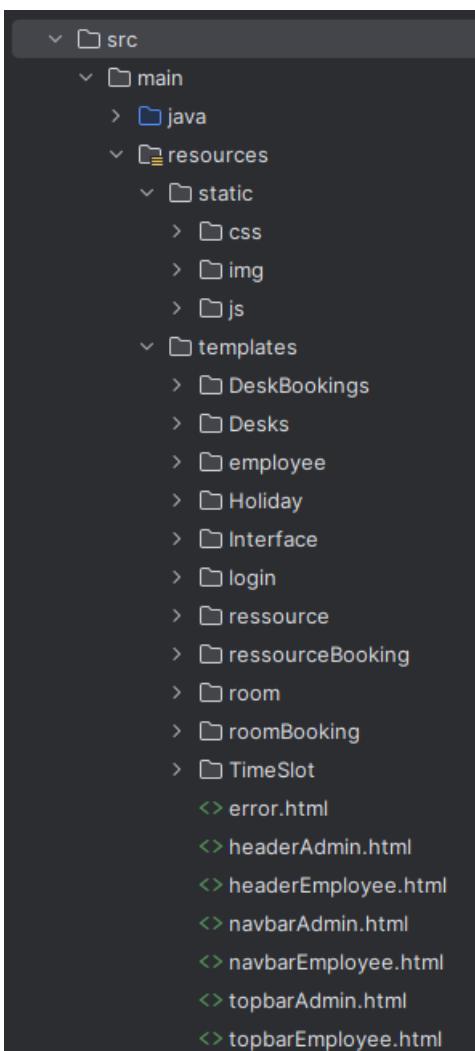
Insgesamt handelt es sich um eine umfassende Beschreibung des Ressourcen- und Ressourcenbuchungsmoduls und seiner Implementierung im Backend.

# 9 Front-End Modul

Neben den Modulen „Ressourcen“ und „RessourcenBuchung“ war Herr Payer Manuel ebenfalls für die Erstellung des Frontends zuständig. In der folgenden Dokumentation wird auf die Struktur, die Funktionsweise und dem Code des Moduls „Frontend“ eingegangen.

Das Klimabündnis Tirol hat dem Projektteam eine Anleitung des Corporate Designs zukommen lassen, auf welchem die Gestaltung der Webanwendung aufgebaut wurde. Das Corporate Design hat dabei die zu verwendenden Farben, Schriftarten und die Logoplatzierung vorgegeben.

## 9.1 Struktur



Wie in der linksstehenden Abbildung 101 - Struktur der Front-end Dateien zu sehen ist, ist das gesamte Frontend in zwei Hauptpackages eingebettet:

Static: Static besitzt weitere Unterordner, in denen sich CSS-Dateien, Bilder (img) und Java Script Code befinden.

Templates: Dort befinden sich sämtliche HTML-Templates, die in der Anwendung zum Einsatz kommen.

Um sicherzustellen, dass das Projektteam dieselben Vorlagen verwendet, hat Herr Payer zuvor Templates erstellt, die von jedem Mitglied der Gruppe befüllt wurden.

Die Mitglieder des Teams haben die Daten via Thymeleaf Annotation in die Vorlage eingebettet und Herr Payer hat mit CSS-Aufrufen für die Gestaltung der Daten gesorgt.

Es fällt auf, dass es jeweils unterschiedliche Templates für den/die Administrator:in und für den/die Mitarbeiter:in vorliegen. Da die beiden genannten Organisationsgruppen unterschiedliche Funktionalitäten innerhalb der Anwendung aufrufen müssen, wurde entschieden, dass auch eine optische Hervorhebung erfolgen soll.

Abbildung 101 - Struktur der Front-end Dateien

In der nachstehenden Dokumentation wird das Frontend immer in Bezug auf das Ressourcen Modul demonstriert. Das Frontend ähnelt sich zum Großteil in allen Modulen und so soll sichergestellt werden, dass sich ein roter Faden durch die Dokumentation des Arbeitsbereiches von Herrn Payer zieht.

## 9.2 Funktionsweise

Bevor auf den Code näher eingegangen wird, folgt eine grundsätzliche Erklärung der Funktionsweise dieses Modules. Es wird darauf eingegangen, wie das entsprechende Template vom WebController aus dem Backend heraus geladen wird. Dabei ist zu erwähnen, dass diese Vorgehensweise ein separates Template für jede funktionelle Anforderung, die eine visuelle Darstellung benötigt, erfordert.

In der nachfolgenden Abbildung 101 - Struktur der Front-end Dateien ist zu sehen, wie ein reguläres Template für eine simple CRUD-Methode aufgebaut ist. Zur Demonstration wird auf die „addRessource“-HTML-Klasse eingegangen, welche die Aufgabe hat, dem/der Anwender:in eine Grafische Benutzeroberfläche (GUI) zur Eingabe von Daten zu liefern.

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<div th:insert="headerAdmin :: headerAdmin" th:with="title='Ressource hinzufügen'"></div>
<body>
<!-- Top Bar -->
<div th:insert="topbarAdmin :: topbarAdmin"></div>
<!-- Side Navbar -->
<div th:insert="navbarAdmin :: navbarAdmin"></div>
<!-- Content -->
<div class="content-container">
    <div class="addResourceContainer">
        <h1>Ressource hinzufügen</h1><br><br>
        <div class="scroll-container">
            <form th:action="@{/web/ressource/addRessource}" th:object="${newRessource}" method="post">
                <!-- zuerst objekt holen, mapping wohin, Übertragungsmethode-->
                <div class="form-group">
                    <label for="name">Name:</label>
                    <input type="text" class="form-control" id="name" name="name" th:field="*{name}" required>
                </div>
                <div class="form-group">
                    <label for="ressourcetype">Kategorie:</label>
                    <select name="ressourcetype" id="ressourcetype" class="form-control">
                        <option disabled selected value>---Bitte Auswählen---</option> <!--disabled= nicht auswählbar-->
                        <!--<option th:each="ressource : ${newRessource.ressourcetype}">
                            th:value="${ressource}" th:text="${ressource}">-->
                        <option th:each="type : ${T(com.itkolleg.bookingsystem.domains.Ressourcetype).values()}">
                            th:value="${type}" th:text="${type}"
                            th:selected="${type == newRessource.ressourcetype}">
                        </option>
                        <!-- Testung muss noch durchgeführt werden-->
                    </select>
                </div>
                <div class="form-group">
                    <label for="description">Beschreibung:</label>
                    <textarea class="form-control" id="description" name="description"
                        th:field="*{description}"></textarea>
                </div>
                <div class="form-group">
                    <label for="info">Info:</label>
                    <input type="text" class="form-control" id="info" name="info" th:field="*{info}">
                </div>
        </div>
    </div>
</div>

```

Abbildung 102 - Auszug des Templates addRessource.html

Wenn die Anwendung ausgeführt wird und man die entsprechende URL aufruft, die für die Darstellung der Webseite zuständig ist, wird der eingegebene Code im Frontend umgewandelt und vom Webserver interpretiert. Die nachstehende **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt, wie der Code umgewandelt wurde.

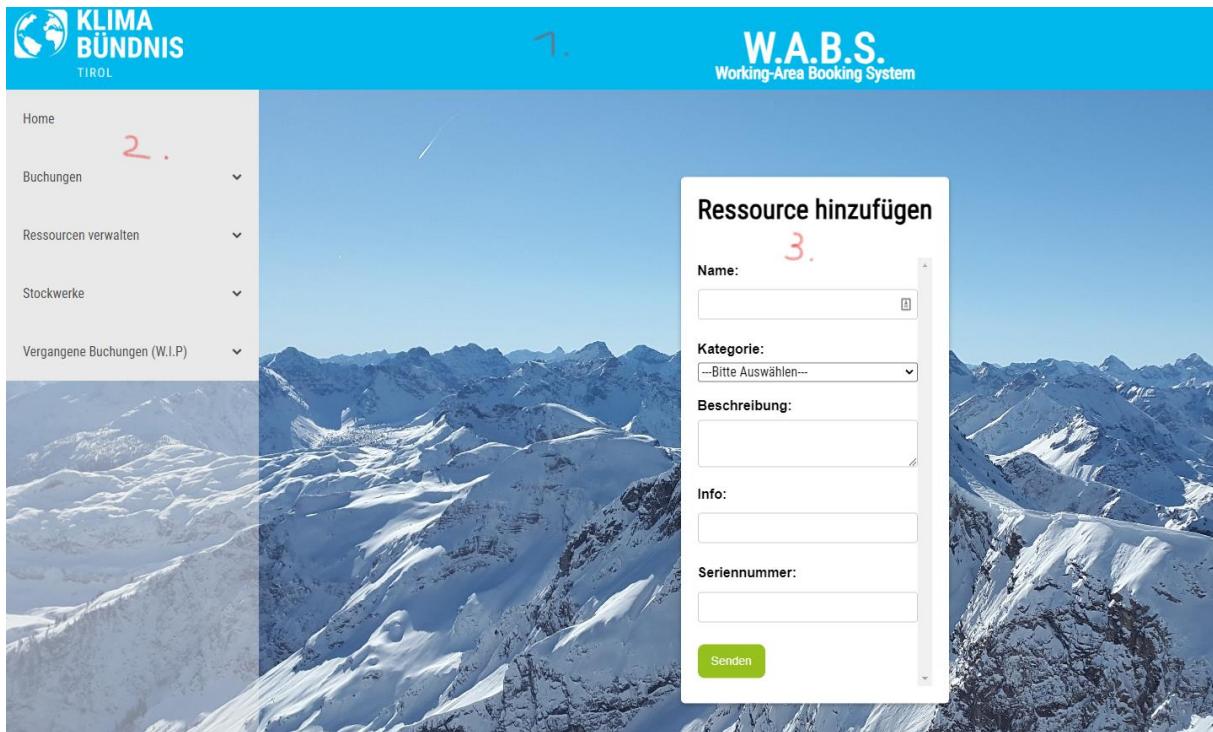


Abbildung 103 - Front-End der Admin Maske um Ressourcen hinzuzufügen

Zum besseren Verständnis wurde das Template in drei Bereiche unterteilt.

1. TOP Bar: Die Topbar ist in allen HTML-Templates, die dem/der Administrator:in gewidmet sind, identisch. Sie beinhaltet das Firmenlogo und den Namen der Anwendung sowie einen Abmelden-Button, der in den nachstehenden Abbildungen ersichtlich ist.

Das Logo in der linken oberen Ecke besitzt darüber hinaus noch eine Weiterleitungsfunktion, die den/die Anwender:in wieder auf die Startseite befördert. Da es üblich ist, dass klickbare Logos den/die Benutzer:in auf die Startseite weiterleitet, wurde diese Funktionalität ebenfalls in dieses Projekt implementiert. Das ist aber nicht die einzige Möglichkeit wieder auf die Startseite zu kommen. Es ist lediglich ein kleines Quality of Life-Feature.

Je nach Benutzergruppe ändert sich die Top-Bar. Die untenstehenden Abbildungen (Abbildung 104 - Top Bar Admin und Abbildung 105 - Top Bar Employee) zeigen den Unterschied der GUI:



Abbildung 104 - Top Bar Admin



Abbildung 105 - Top Bar Employee

2. Nav Bar: Linksbündig befindet sich auf jedem Template eine Navbar, die ihr Aussehen ebenfalls je nach Benutzergruppe ändert. Da der/die Administrator:in andere Funktionalitäten wie der/die Mitarbeiter:in besitzt, passt sich dementsprechend die Navbar auch der Benutzergruppe an.

So kann man nachstehend sehen, wo die Unterschiede, je nach Benutzergruppe, liegen. Links ist die Navbar der Administrativen Gruppe (Abbildung 106 - Nav.Bar Admin) und rechts der Mitarbeiter:innen-Schicht (Abbildung 106.1 - Nav.Bar Employee). Die Navbar besteht aus HTML, JS und CSS Code.

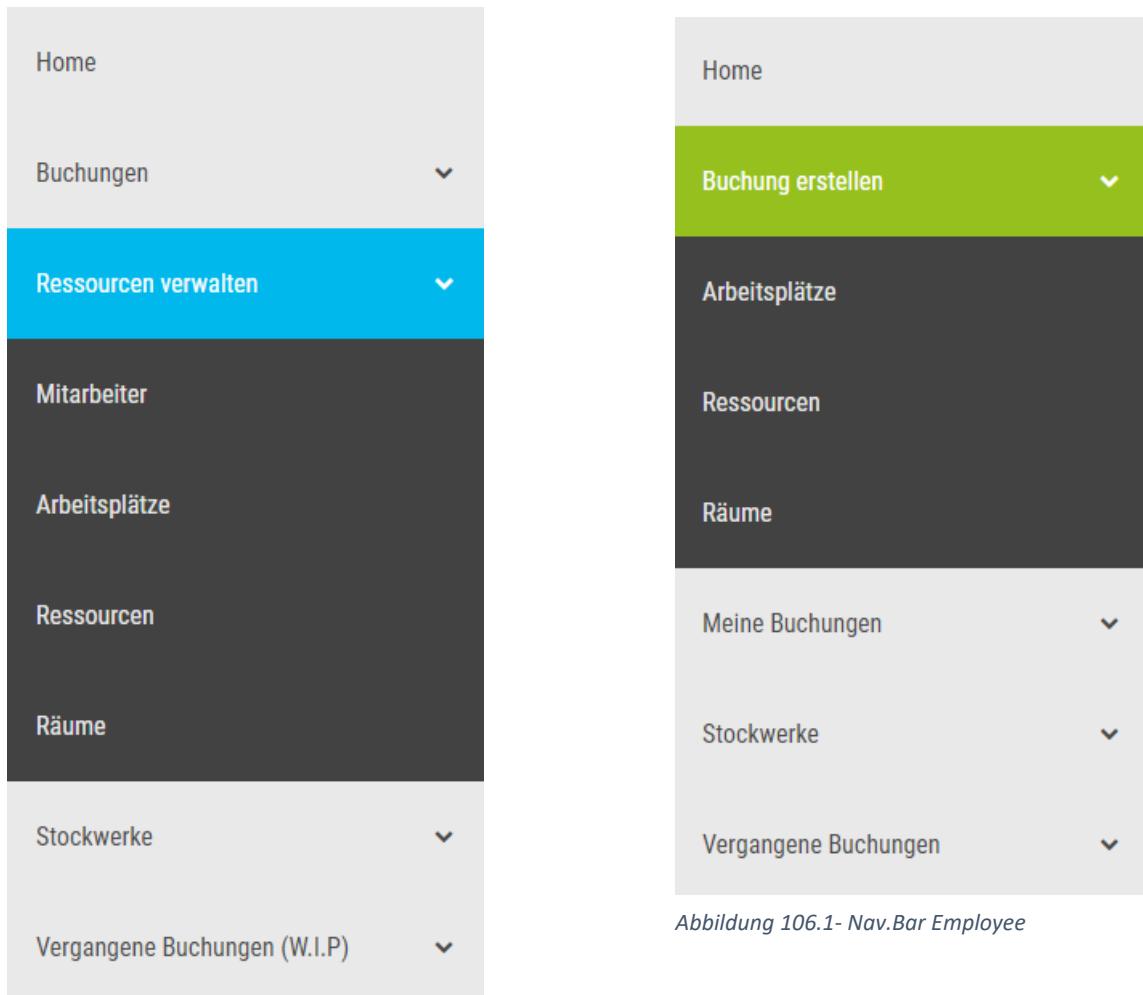


Abbildung 106 - Nav.Bar Admin

Abbildung 106.1- Nav.Bar Employee

3. Content Container: Dieser Bereich bezieht sich nicht nur auf das Eingabe-Formular mit dem Titel „Ressource hinzufügen“, sondern viel mehr auf den gesamten inneren Bereich, dem Contentbereich.

Das ist das Herzstück eines jeweiligen Templates und hat unterschiedliche Aufgaben. In diesem Fall ist die Aufgabe das Darstellen eines Formulars, welches der/die Benutzer:in benötigt, um der Datenbank eine neue Ressource mit mitgegebenen Parametern zu übergeben. Da dieser Teil je nach Modul unterschiedliche Funktionalitäten aufweist, musste eben jener Bereich von jedem Projektmitglied selbst definiert und implementiert werden. Herr Payer hat sich dann um die korrekte visuelle Darstellung gekümmert.

Eventuell ist bei der Betrachtung des Codes (zu sehen auf der Abbildung 102) aufgefallen, dass weder Header, Topbar noch die Navbar Code aufweisen. Das ist auch so gewollt. Da die drei genannten Bereiche in jedem Template identisch sind, wurde auf Hinblick der Wartbarkeit beschlossen, diesen Bereich aus den Templates in eine jeweilige Datei auszulagern.

Sollte nun eine Änderung der Navbar für den/die Administrator:in erfolgen, so muss diese Änderung nur noch in einer Datei durchgeführt werden und sie wird automatisch für alle Templates übernommen.

Eine nähere Beschreibung der Auslagerung und der verwendeten Technologien folgt im Abschnitt: „10.2 Verwendete Technologien“.

Der Header beinhaltet in der Regel Meta Daten, die für die jeweilige Datei relevant sind. Da aber manche Templates unterschiedliche Daten/Links/Skripte erfordern, wurde beschlossen, einfach alle verwendeten Meta Daten zusammen mit dem Header auszulagern. So kann es sein, dass Meta Daten in Template eingebunden wurden, welche nicht zum Einsatz kommen, aber es stellte sich heraus, dass dies eine einfache Auslagerung und das Verhindern von Codeduplikaten ermöglicht.

## 9.3 Technologien im Frontend

**Thymeleaf:** Das ist eine Template-Engine für die Erstellung von dynamischen Webseiten in Java-Anwendungen. Es hat eine enge Integration mit dem Frontend und ermöglicht es Entwickler:innen, Serverdaten nahtlos in HTML-Vorlagen einzubetten.

Mithilfe von Thymeleaf Ausdrücken ist es möglich, Serverdaten direkt in HTML-Tags und -Attributen einzubetten. Zusätzlich unterstützt Thymeleaf auch die Verwendung von Iterationen und bedingten Anweisungen sowie Internationalisierung und Lokalisierung.

Zusammengefasst ist Thymeleaf ein sehr mächtiges Tool, welches einen starken Einsatz im gesamten Projekt einnimmt.

**Bootstrap:** Dies ist ein kostenloses Open Source Frontend-Entwicklungs Framework, welches eine Kollektion von unterschiedlichen Syntaxen beinhaltet, um schnell optisch ansprechende Designs zu erstellen, die ebenfalls responsive sind. Dabei verwendet Bootstrap sowohl HTML, als auch Java Script und CSS.

Der Einsatz von Bootstrap in diesem Projekt wurde allerdings nicht stark priorisiert, da der Großteil des Designs manuell entwickelt wurde. So würde Bootstrap im Allgemeinen eine JavaScript Funktion für Dropdowns beinhalten. Jedoch wurde entschieden, das selbst in JavaScript zu implementieren.

## 9.4 Implementierung

Im nachstehenden Kapitel folgen Erklärungen zu markanten Codeausschnitten und mit welchen Hintergedanken sie implementiert wurden.

Wie bereits weiter oben erwähnt wurde, wurden diverse Codeduplikate aus den Templates ausgelagert um eine bessere Wartbarkeit und Übersichtlichkeit zu erschaffen. Dies wird durch die Thymeleaf Funktion „th:insert“ ermöglicht.

Die folgenden Abbildung 107, Abbildung 108 zeigen den Code, der vom NavBar Bereich ausgelagert wurde. Um den Code an genau diese Stelle über einen Aufruf einbinden zu können, muss man den „th:insert“ Ausdruck verwenden.

"navbarAdmin :: navbarAdmin": Dies ist der Wert des th:insert-Attributs. Hier wird definiert, welches Stück HTML-Inhalt eingefügt werden soll. In diesem Fall wird nach einem Thymeleaf-Fragment mit dem Namen "navbarAdmin" gesucht und dieses eingefügt. Das Doppelpunkt-Syntax (::) wird verwendet, um auf ein bestimmtes Fragment innerhalb eines Thymeleaf-Templates zu verweisen.

Das bedeutet, dass Thymeleaf nach einem Fragment mit dem Namen "navbarAdmin" sucht und den Inhalt dieses Fragments anstelle des <div>-Tags einfügt, in dem sich die th:insert-Anweisung befindet.

```
<body>
<!-- Top Bar --&gt;
&lt;div th:insert="topbarAdmin :: topbarAdmin"&gt;&lt;/div&gt;
<!-- Side Navbar --&gt;
&lt;div th:insert="navbarAdmin :: navbarAdmin"&gt;&lt;/div&gt;</pre>

```

Abbildung 107 - Aufruf des ausgelagerten Codes der navbar Admin

Um es einfach zusammenzufassen: Der hier gezeigte Code (Abbildung 108 - Ausgelagerter Code der NavBar Admin) wird während der Laufzeit an die oben angeführte Stelle, dank des Thymeleaf-Ausdrucks, injiziert.

```
<div class="contenedor-menu" th:fragment="navbarAdmin">
    <!-- &lt;a href="" class="btnMenu"&gt;Menu &lt;i class="fa fa-bars"&gt;&lt;/i&gt;&lt;/a&gt; --&gt;

    &lt;ul class="menu"&gt;
        &lt;li&gt;&lt;a th:href="@{/web/admin/admin-start}"&gt;Home&lt;/a&gt;&lt;/li&gt;
        &lt;li&gt;&lt;a href="#"&gt;Buchungen&lt;i class="fa fa-chevron-down"&gt;&lt;/i&gt;&lt;/a&gt;
            &lt;ul&gt;
                &lt;li&gt;&lt;a th:href="@{/web/deskbookings/admin}"&gt;Arbeitsplätze&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a th:href="@{/web/ressourceBooking/allBookings}"&gt;Ressourcen&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a th:href="@{/web/roomBooking/allBookings}"&gt;Räume&lt;/a&gt;&lt;/li&gt;
            &lt;/ul&gt;
        &lt;/li&gt;
        &lt;li&gt;&lt;a href="#"&gt;Ressourcen verwalten&lt;i class="fa fa-chevron-down"&gt;&lt;/i&gt;&lt;/a&gt;
            &lt;ul&gt;
                &lt;li&gt;&lt;a th:href="@{/web/admin/allemployees}"&gt;Mitarbeiter&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a th:href="@{/web/desks}"&gt;Arbeitsplätze&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a th:href="@{/web/ressource/allRessources}"&gt;Ressourcen&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a th:href="@{/web/rooms/allRooms}"&gt;Räume&lt;/a&gt;&lt;/li&gt;
            &lt;/ul&gt;
        &lt;/li&gt;
        &lt;li&gt;&lt;a href="#"&gt;Stockwerke&lt;i class="fa fa-chevron-down"&gt;&lt;/i&gt;&lt;/a&gt;
            &lt;ul&gt;
                &lt;li&gt;&lt;a th:href="@{/web/rooms/floors}"&gt;Stockwerk 1&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a th:href="@{/web/admin/admin-start}"&gt;Stockwerk 2 (W.I.P.)&lt;/a&gt;&lt;/li&gt;
            &lt;/ul&gt;
        &lt;/li&gt;
        &lt;li&gt;&lt;a href="#"&gt;Vergangene Buchungen (W.I.P)&lt;i class="fa fa-chevron-down"&gt;&lt;/i&gt;&lt;/a&gt;
            &lt;ul&gt;
                &lt;li&gt;&lt;a href="@{/web/deskbookings/admin}"&gt;Arbeitsplätze (W.I.P)&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a href="@{/web/ressourceBooking/allBookings}"&gt;Ressourcen (W.I.P)&lt;/a&gt;&lt;/li&gt;
                &lt;li&gt;&lt;a href="@{/web/roomBooking/allBookings}"&gt;Räume (W.I.P)&lt;/a&gt;&lt;/li&gt;
            &lt;/ul&gt;
        &lt;/li&gt;
    &lt;/ul&gt;
&lt;/div&gt;</pre>

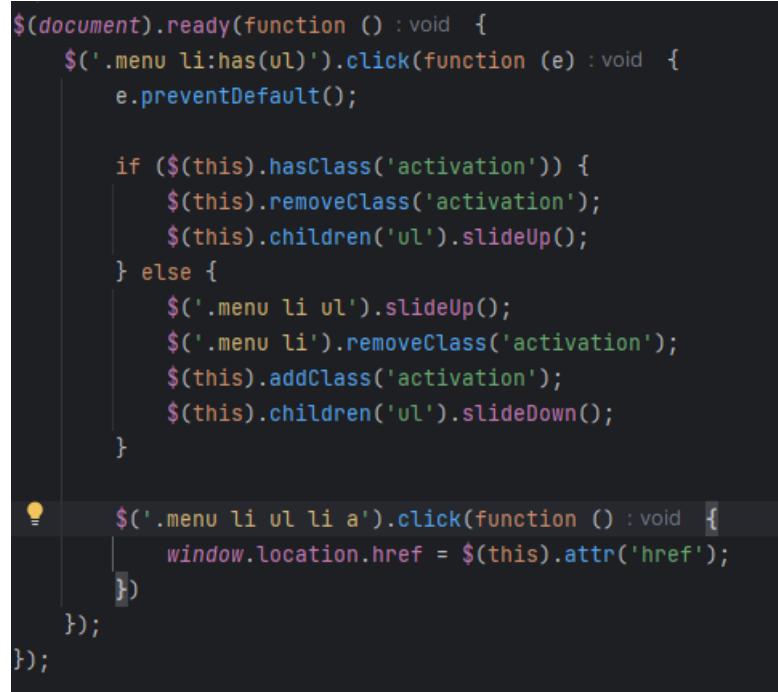
```

Abbildung 108 - Ausgelagerter Code der NavBar Admin

## JavaScript Verwendung

Wie bereits erwähnt, stützt sich das Projekt ebenfalls auf die Verwendung von JavaScript. Diese JavaScript Dateien wurden ebenfalls in eigene Klassen im Package „Static/js“ ausgelagert, um eine sauberere und wartbare Anwendung zu schaffen.

In Bezug auf den Arbeitsbereich von Herrn Payer kam JavaScript hauptsächlich im NavBar Bereich vor. Das Skript hat hier die Aufgabe, mehrere Funktionalitäten der Navbar hinzuzufügen. Das schließt eine Dropdown Funktion, farbliche Hervorhebungen, responsive Anpassung, öffnen und schließen von Dropdown-Menüs und on-klick Weiterleitung mit ein.



```

$(document).ready(function () : void {
    $('.menu li:has(ul)').click(function (e) : void {
        e.preventDefault();

        if ($(this).hasClass('activation')) {
            $(this).removeClass('activation');
            $(this).children('ul').slideUp();
        } else {
            $('.menu li ul').slideUp();
            $('.menu li').removeClass('activation');
            $(this).addClass('activation');
            $(this).children('ul').slideDown();
        }

        $('.menu li ul li a').click(function () : void {
            window.location.href = $(this).attr('href');
        })
    });
});

```

Abbildung 109 - Java Script Funktion für den Dropdown Effekt

Der gezeigte Code ist ein JavaScript-Code-Snippet, der in der Regel verwendet wird, um das Verhalten eines Dropdown-Menüs auf einer Webseite zu steuern.

`$(document).ready(function () { ... });`: Dieser Code wird ausgeführt, wenn das Dokument (die Webseite) vollständig geladen ist. Es verwendet jQuery, um sicherzustellen, dass der Code erst ausgeführt wird, wenn das DOM (Document Object Model) bereit ist.

`$('.menu li:has(ul)').click(function (e) { ... })`: Dieser Teil des Codes wählt alle <li>-Elemente in einem HTML-Element mit der Klasse "menu" aus, die Unterlisten (<ul>) enthalten. Wenn eines dieser <li>-Elemente angeklickt wird, wird die darin enthaltene Funktion ausgelöst.

`e.preventDefault();`: Dieser Aufruf verhindert, dass das Standardverhalten eines HTML-Links oder eines HTML-Formulars ausgeführt wird. In diesem Fall verhindert es, dass ein Link in einem Dropdown-Menü dazu führt, dass die Seite neu geladen wird.

`if ($(this).hasClass('activation')) { ... } else { ... };`: Hier wird überprüft, ob das angeklickte `</i>`-Element die CSS-Klasse "activation" hat. Wenn es diese hat, bedeutet das, dass das Dropdown-Menü bereits geöffnet ist, und der Code wird verwendet, um es zu schließen. Andernfalls wird es geöffnet.

`$(this).removeClass('activation');`: Wenn das Dropdown-Menü bereits geöffnet ist (hat die Klasse "activation"), wird diese Zeile verwendet, um die Klasse "activation" zu entfernen, wodurch das Dropdown-Menü geschlossen wird.

`$(this).children('ul').slideUp();`: Hier wird das `<ul>`-Element, das ein Kind des angeklickten `</i>`-Elements ist, eingefahren (versteckt), was dazu führt, dass das Dropdown-Menü geschlossen wird.

`else { ... };`: Wenn das Dropdown-Menü nicht geöffnet ist (hat nicht die Klasse "activation"), wird dieser Block ausgeführt.

`$('.menu li ul').slideUp();`: Hier werden alle Dropdown-Menüs, die sich in `<ul>`-Elementen innerhalb von `<li>`-Elementen befinden, geschlossen. Dies stellt sicher, dass nur ein Dropdown-Menü gleichzeitig geöffnet ist.

`$('.menu li').removeClass('activation');`: Alle `</i>`-Elemente in der Klasse "menu" verlieren die Klasse "activation", um sicherzustellen, dass nur das aktuell angeklickte Element die Klasse hat.

`$(this).addClass('activation');`: Das aktuell angeklickte `<li>`-Element erhält die Klasse "activation", um anzugeben, dass das Dropdown-Menü geöffnet ist.

`$(this).children('ul').slideDown();`: Das `<ul>`-Element, das ein Kind des angeklickten `</i>`-Elements ist, wird eingeschoben (angezeigt), wodurch das Dropdown-Menü geöffnet wird.

`$('.menu li ul li a').click(function () { ... });`: Dieser Abschnitt fängt Klick-Ereignisse auf den Anker-Links (`<a>`) in den Dropdown-Listen ab. Wenn ein solcher Link angeklickt wird, wird die darin enthaltene Funktion ausgeführt.

`window.location.href = $(this).attr('href');`: Diese Zeile leitet den/die Benutzer:in zur URL weiter, die im href-Attribut des angeklickten Links enthalten ist. Mit anderen Worten: Wenn ein Dropdown-Link angeklickt wird, navigiert er den/die Benutzer:in zu einer anderen Seite oder URL.

```
$document.ready(function () : void {
    $("#navbarContainer").load("/static/css/navbarEmployee.html");
});

//alert("JavaScript-Code wird ausgeführt!");

// Adjust dropdown width when expanded
let dropdowns : NodeListOf<Element> = document.querySelectorAll( selectors: ".dropdown" );
± pwnzZzZz
dropdowns.forEach( callbackfn: (dropdown : Element) : void => {
    let content : Element = dropdown.querySelector( selectors: ".dropdown-content" );
    let links : NodeListOf<HTMLAnchorElement> = content.querySelectorAll( selectors: "a" );
    let width : number = 0;
    links.forEach( callbackfn: (link : HTMLAnchorElement) : void => {
        width = Math.max(width, link.offsetWidth);
    });
    content.style.width = width + "px";
});

// Toggle dropdown content on click
let dropdownToggles : NodeListOf<Element> = document.querySelectorAll( selectors: ".dropdown > a" );
± pwnzZzZz
dropdownToggles.forEach( callbackfn: (toggle : Element) : void => {
    let content : Element = toggle.nextElementSibling;
    toggle.addEventListener( type: "click", listener: (event : Event) : void => {
        event.preventDefault();
        content.classList.toggle( token: "open" );
    });
});
}
```

Abbildung 110 - CodeSnippet der JavaScript Datei die die meisten Funktionen beinhaltet

`$(document).ready(function () { ... });`: Dieser Code verwendet jQuery und wird ausgeführt, wenn das Dokument (die Webseite) vollständig geladen ist. Es lädt den Inhalt einer HTML-Datei ("navbarEmployee.html") in ein Element mit der ID "navbarContainer". Dies dient dazu, die Navbar (Navigationsleiste) der Webseite dynamisch zu erstellen oder zu aktualisieren.

`//alert("JavaScript-Code wird ausgeführt!");`: Diese Zeile ist auskommentiert (beginnt mit //), was bedeutet, dass sie nicht aktiv ist. Es handelt sich um eine Kommentarzeile, die für Entwickler oder Debugging-Zwecke verwendet werden kann, um eine Nachricht anzuzeigen. In diesem Fall wird die Nachricht "JavaScript-Code wird ausgeführt!" ausgegeben, wenn sie aktiviert wird.

`let dropdowns = document.querySelectorAll(".dropdown");`: Dieser Code verwendet die native JavaScript-Funktion querySelectorAll, um alle Elemente auf der Seite auszuwählen, die die CSS-Klasse "dropdown" haben. Diese Elemente könnten Dropdown-Menüs repräsentieren.

`dropdowns.forEach((dropdown) => { ... });`: Hier wird eine Schleife verwendet, um durch alle ausgewählten Dropdown-Elemente zu iterieren.

`let content = dropdown.querySelector(".dropdown-content");`: Für jedes Dropdown-Element wird das darin enthaltene Element mit der Klasse "dropdown-content" ausgewählt. Dies könnte den Inhalt des Dropdown-Menüs darstellen.

`let links = content.querySelectorAll("a");`: Hier werden alle Anker-Links (`<a>`) innerhalb des Dropdown-Inhalts ausgewählt.

`let width = 0;`: Eine Variable "width" wird initialisiert, um die maximale Breite der Dropdown-Links zu verfolgen.

`links.forEach((link) => { ... });`: Eine Schleife durchläuft alle ausgewählten Dropdown-Links.

`width = Math.max(width, link.offsetWidth);`: Hier wird die Breite des aktuellen Dropdown-Links mit der aktuellen maximalen Breite verglichen und die größere der beiden wird in der "width"-Variable gespeichert. Auf diese Weise wird die maximale Breite aller Links ermittelt.

`content.style.width = width + "px";`: Schließlich wird die Breite des Dropdown-Inhalts auf die ermittelte maximale Breite gesetzt, wodurch alle Dropdown-Links die gleiche Breite haben.

`let dropdownToggles = document.querySelectorAll(".dropdown > a");`: Dieser Code wählt alle Anker-Links (`<a>`) aus, die direkte Kinder von Elementen mit der Klasse "dropdown" sind. Diese Anker-Links könnten Schaltflächen sein, die verwendet werden, um das Dropdown-Menü zu öffnen oder zu schließen.

`dropdownToggles.forEach((toggle) => { ... });`: Eine Schleife durchläuft alle ausgewählten Anker-Links.

`let content = toggle.nextElementSibling;`: Hier wird das nächste Geschwister-Element des aktuellen Anker-Links ausgewählt. Dies könnte das Dropdown-Menü-Inhaltselement sein.

`toggle.addEventListener("click", (event) => { ... });`: Hier wird ein Klick-Ereignislistener zum Anker-Link hinzugefügt. Wenn der Anker-Link geklickt wird, wird die darin enthaltene Funktion ausgelöst.

`event.preventDefault();`: Diese Zeile verhindert das Standardverhalten des Anker-Links, d.h. das Navigieren zu einer anderen Seite. Dadurch wird verhindert, dass der Link die Seite neu lädt oder zu einer anderen Seite springt.

`content.classList.toggle("open");`: Hier wird die CSS-Klasse "open" zum Dropdown-Inhalt hinzugefügt oder entfernt. Dies ermöglicht es, das Dropdown-Menü bei Klick auf den Anker-Link zu öffnen oder zu schließen, indem die Sichtbarkeit des Dropdown-Inhalts gesteuert wird.

Zusammengefasst handelt es sich bei diesem JavaScript-Code um eine Funktion, die Dropdown-Menüs auf einer Webseite steuert. Sie passt die Breite der Dropdown-Links an und ermöglicht das Öffnen und Schließen der Dropdown-Menüs bei Klick auf die zugehörigen Anker-Links.

Neben der oben erklärten Syntax beinhalten die JavaScript Dateien noch mehrere Funktionen. So wurde beschlossen, dass es Sinn macht, Funktionen für die Datumsauswahl sowie deren Verwaltung ebenfalls in dieselbe Datei wie die Navbar auszulagern.

In der Regel würde man diese Funktionen in eigene Klassen auslagern, da die Date-/Time-Picker Funktion nichts mit der Navbar zu tun hat. Dennoch wurden diese Funktionen in dieselbe Klasse implementiert, da sie ohnehin von jedem Template aufgerufen werden und man die Struktur des Projektes nicht noch größer machen wollte.

## CSS Verwendung:

Das gesamte CSS wurde für das Projekt eigens entwickelt und es folgen einige Auszüge aus dieser Datei. Das Hauptaugenmerk liegt in der ersten Abbildung auf visuelle Darstellung und Positionierung eines „Delete-Buttons“, der bei den meisten Template zum Einsatz kommt.

```
.delete-button {
    background-color: #00b8eb;
    border-radius: 8px;
    border-style: none;
    box-sizing: border-box;
    color: #FFFFFF;
    cursor: pointer;
    display: inline-block;
    font-size: 14px;
    font-weight: 500;
    height: 40px;
    line-height: 20px;
    list-style: none;
    margin: 0;
    outline: none;
    padding: 10px 16px;
    position: relative;
    text-align: center;
    text-decoration: none;
    transition: color 100ms;
    vertical-align: baseline;
    user-select: none;
    -webkit-user-select: none;
    touch-action: manipulation;
}
```

Abbildung 111 - CSS Regeln des Delete-Buttons

*background-color: #00b8eb;:* Setzt die Hintergrundfarbe des Elements auf ein Hellblau (#00b8eb).

*border-radius: 8px;:* Rundet die Ecken des Elements mit einem Radius von 8 Pixeln, was zu abgerundeten Ecken führt.

*border-style: none;:* Entfernt den Rahmen (keinen Rand) um das Element.

*box-sizing: border-box;:* Stellt sicher, dass die Größe des Elements einschließlich Padding und Border berechnet wird (kein zusätzlicher Raum für Padding und Border).

*color: #FFFFFF;:* Setzt die Textfarbe des Elements auf Weiß (#FFFFFF).

*cursor: pointer;:* Zeigt den Mauszeiger als Zeiger (wie bei einem anklickbaren Element) an, wenn die Maus über das Element bewegt wird.

*display: inline-block;:* Legt fest, dass das Element als Block-Element behandelt wird, aber wie ein Inline-Element in Zeile mit dem vorherigen Inhalt fließt.

*font-size: 14px;*: Legt die Schriftgröße des Textes im Element auf 14 Pixel fest.

*font-weight: 500;*: Setzt die Schriftgewichtung auf 500, was normalerweise einem mittleren Gewicht entspricht.

*height: 40px;*: Legt die Höhe des Elements auf 40 Pixel fest.

*line-height: 20px;*: Setzt die Zeilenhöhe (Abstand zwischen Textzeilen) auf 20 Pixel.

*list-style: none;*: Entfernt jegliche Listenpunkte oder Markierungen, die normalerweise vor dem Element angezeigt werden.

*margin: 0;*: Entfernt jeglichen Außenabstand um das Element.

*outline: none;*: Entfernt die Standard-Umrandung (Fokus-Rahmen) um das Element, wenn es aktiviert ist.

*padding: 10px 16px;*: Legt den Innenabstand des Elements auf 10 Pixel oben/unten und 16 Pixel links/rechts fest.

*position: relative;*: Setzt die Positionierung des Elements auf "relative", was bedeutet, dass es relativ zu seiner normalen Position verschoben werden kann.

*text-align: center;*: Zentriert den Text innerhalb des Elements horizontal.

*text-decoration: none;*: Entfernt jegliche Textdekoration (normalerweise Unterstreichung) für verknüpfte Texte.

*transition: color 100ms;*: Definiert eine sanfte Übergangsanimation für die Textfarbe (Farbänderung) mit einer Dauer von 100 Millisekunden.

*vertical-align: baseline;*: Legt die vertikale Ausrichtung des Elements auf die Grundlinie (Baseline) des umgebenden Texts fest.

*user-select: none;*: Verbietet die Auswahl des Texts im Element durch den Benutzer.

*-webkit-user-select: none;*: Eine ähnliche Regel wie "user-select", die spezifisch für WebKit-basierte Browser (wie Chrome und Safari) gilt.

*touch-action: manipulation;*: Legt das Verhalten des Elements für Touch-Gesten fest, in diesem Fall für "manipulation", was bedeutet, dass es für Benutzergesten wie Scrollen und Zoomen optimiert ist und nicht selektiert wird.

Diese Regeln definieren das Erscheinungsbild und Verhalten von Elementen mit der CSS-Klasse "delete-button" auf der Webseite. Wichtig ist, dass diese Regeln zwar das Verhalten und das Aussehen des „delete-Buttons“ bestimmen, seine Anwendungslogik hängt davon aber nicht ab. So kann der „delete-Button“ auch einfach als „return-Button“ funktionieren, wie man in der nachstehenden Abbildung 112 - Aufruf des Delete-Buttons sehen kann:

```
<div class="button-row">
    <a th:href="@{/web/ressource/addRessource}" class="add-button" role="button">neue Ressource</a>
    <button class="delete-button" onclick="goBack()">Zurück</button>
</div>
```

Abbildung 112 - Aufruf des Delete-Buttons

Da größtenteils die Verwendung von Bootstrap ausgeschlossen wurde, um ein besseres Verständnis der einzelnen Technologien und deren Anwendung zu erlangen, musste dementsprechend die Responsibility ebenfalls manuell hinterlegt werden. Dies geschah durch den Einsatz von Media Queries:

```
/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {
    .logo {
        width: 150px;
    }

    .content-container {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 10vh;
    }

    .top-bar-header {
        top: 30%;
    }

    .top-bar-header h1 {
        font-size: 18px;
    }

    .top-bar-sub-header {
        top: 60%;
    }

    .top-bar-sub-header h2 {
        font-size: 12px;
    }
}
```

Abbildung 113 - Media Rules

Wenn das Anwendungsfenster kleiner wird, so erkennt das System die derzeit verwendete Pixelgröße der Anwendung und setzt dementsprechend andere CSS-Regeln in Kraft. So besitzt das Logo eine Breite von 150px bei kleinen Bildschirmen bis zu einer Breite von 600px. Kommen beispielsweise Bildschirme mit mindestens 1200px Breite zum Einsatz, so wird das Logo auf 200px Breite skaliert.

Derzeit ist die Anwendung für Bildschirme zwischen 600px und 1200px und darüber optimiert. Kleinere Bildformate (Smartphones oder kleine Tablets) besitzen derzeit keine eigene Skalierung, da eine Anwendung auf Smartphones derzeit nicht vorgesehen ist.

## 9.5 Zusammenfassung Front-End Modul

In diesem Projekt war Herr Payer für das Frontend verantwortlich. Die Struktur des Frontends wurde im Zusammenhang mit dem Ressourcenmodul erläutert, obwohl sie in allen Modulen weitgehend ähnlich ist, um eine einheitliche Dokumentation sicherzustellen.

Es wurde die grundlegende Funktionsweise des Frontend-Moduls erklärt. Hierbei wird beschrieben, wie die Templates vom Backend-Controller geladen werden und warum für jede funktionale Anforderung, die eine visuelle Darstellung benötigt, ein separates Template erforderlich ist.

Im Frontend kamen zwei Haupttechnologien zum Einsatz. Zum einen Thymeleaf, eine Java-basierte Template-Engine, die es ermöglicht, Serverdaten nahtlos in HTML einzufügen. Zum anderen Bootstrap, ein Framework zur Front-End-Entwicklung, das die Erstellung ansprechender und responsiver Designs erleichtert.

Einige JavaScript-Codeausschnitte sind ebenfalls Teil des Projekts. Sie dienen der Steuerung von Dropdown-Menüs und der Anpassung der visuellen Darstellung. Darüber hinaus habe ich Codeauslagerungen und die Verwendung von Thymeleaf-Ausdrücken implementiert, um die Wartbarkeit zu verbessern.

Im Bereich CSS werden Regeln für ein Element namens "delete-button" beschrieben, einschließlich Hintergrundfarbe, Schriftgröße, Abständen und Rundung der Ecken.

Abschließend wurden Media Queries verwendet, um das Design an verschiedene Bildschirmgrößen anzupassen. Beispielsweise wurde die Größe des Logos je nach Bildschirmgröße skaliert.

Insgesamt war meine Arbeit im Frontend von großer Bedeutung für die visuelle Gestaltung und Benutzerfreundlichkeit des Projekts. Die Dokumentation und Strukturierung des Codes sowie die Nutzung verschiedener Technologien haben dazu beigetragen, ein robustes Frontend zu erstellen.

# 10 Raummodul

## 10.1 Einführung

Dieses Modul stellt eine Hauptkomponente der Applikation dar. Das gesamte Programm ist darauf ausgerichtet, den Benutzer:innen und den Mitarbeiter:innen des Klimabündnisses eine benutzerfreundlich und intuitive Möglichkeit zur schnellen und unkomplizierten Buchung der Räume und den sich darin befindlichen Arbeitsplätzen zur Verfügung zu stellen.

Die Hauptherausforderung bestand darin, die graphische Oberfläche für den Nutzer so zu gestalten, dass auch nicht programmieraffine Administrator:innen zukünftig neue Etagen und Räume hinzufügen können, um eine langfristige Nutzung trotz eventueller Anpassungen zu gewährleisten.

## 10.2 Verwendete Technologien

Um die Leistungsfähigkeit der Webapplikation und den einzelnen Modulen zu gewährleisten, wurden bestimmte Technologien ausgewählt die auch die Entwicklung ungemein vereinfachten.

Für die Entwicklung des Backend-Systems wurde Java als Programmiersprache ausgewählt, da diese Sicherheit und Stabilität mit sich bringt. Zusätzlich bringt Java den Vorteil, dass es sich gut mit Spring Boot kombinieren lässt, um komplexere Anwendungen zu entwickeln. Ein weiterer Aspekt, der für die Wahl von Java spricht, ist, dass alle Teammitglieder durch ihre Ausbildung die meiste Erfahrung mit dieser Sprache haben.

Spring Boot spielt in diesem Zusammenhang eine entscheidende Rolle. Das Framework bietet eine robuste und skalierbare Plattform für die Implementierung von Geschäftslogik, Datenbankanbindung und die Verwaltung von Benutzersitzungen. Um die Abwicklung des Projektmanagements und die Verwaltung der Abhängigkeiten zu vereinfachen, setzt das Team auf Maven. Dieses Tool standardisiert die Projektorganisation und macht die Handhabung von Abhängigkeiten unkompliziert, während es gleichzeitig die Migration zu neuen Funktionen durch mehr Transparenz erleichtert.

Um den sogenannten "Boilerplate Code" zu minimieren, kommt die Java-Bibliothek Lombok zum Einsatz. Lombok erlaubt es, mehrere Zeilen notwendigen Codes durch nur eine Zeile zu ersetzen. Weiterhin wird die Jakarta-Bibliothek genutzt, eine Sammlung von Java-Bibliotheken und APIs, die vor allem durch die JPA (Java Persistence API) die Arbeit mit Datenstrukturen und Datenbanken erleichtert.

Für die Frontend-Entwicklung wurde auf Thymeleaf gesetzt, eine moderne serverseitige Template-Engine. Sie wird oft in Verbindung mit Spring Boot verwendet und ermöglicht das einfache Einbinden dynamischer Inhalte in HTML-Templates. Ein besonderes Merkmal von Thymeleaf ist seine "natürliche Vorlage"-Fähigkeit, die es Entwicklern erlaubt, Templates als normale HTML-Dateien anzusehen, selbst ohne Durchlauf durch den Thymeleaf-Prozessor. Dies vereinfacht das Design und Debugging erheblich.

Zuletzt setzte sich für die Steuerung und Manipulation eventbasierter Aktionen der Benutzeroberfläche JavaScript durch, da es sich dafür ideal eignet.

## 10.3 Raum Modul Backend

In diesem Abschnitt wird die Struktur und Funktionsweise der Applikation anhand von Codeauszügen erläutert.

### 10.3.1 Struktur des Backends

Das Modul und die gesamte Applikation basiert auf der Verwendung einer sogenannten Schichtenarchitektur (Layered Architecture). Jede Schicht hat klare Verantwortlichkeiten und Schnittstellen, um die Abhängigkeiten zwischen den Schichten zu minimieren. Dies hat sich als bewährte Praxis in der Softwareentwicklung etabliert und bietet mehrere Vorteile:

Jede Schicht ist für einen spezifischen Aspekt der Anwendung zuständig, was eine saubere Aufgabenverteilung ermöglicht. Diese gezielte Zuteilung der Verantwortlichkeiten führt auch zu einer höheren Wiederverwendbarkeit der jeweiligen Komponenten. Da die Schichten jeweils nur für bestimmte Aufgaben konzipiert sind, können sie in anderen Kontexten oder Projekten leicht wiederverwendet werden.

Ein weiterer wichtiger Aspekt ist die Skalierbarkeit. Durch getrennte Schichten kann bei steigendem Ressourcenbedarf jeder Layer unabhängig von dem Anderen skaliert werden. Dies erlaubt eine effiziente Nutzung der Ressourcen und erleichtert die Anpassung der Software an veränderte Anforderungen.

Die Gliederung der Software in separate Komponenten hat zudem noch den Vorteil, dass die Wartbarkeit vereinfacht wird. Zukünftige Anpassungen oder Erweiterungen können in der Regel auf die betroffenen Komponenten beschränkt werden, ohne, dass der gesamte Code überarbeitet werden muss.

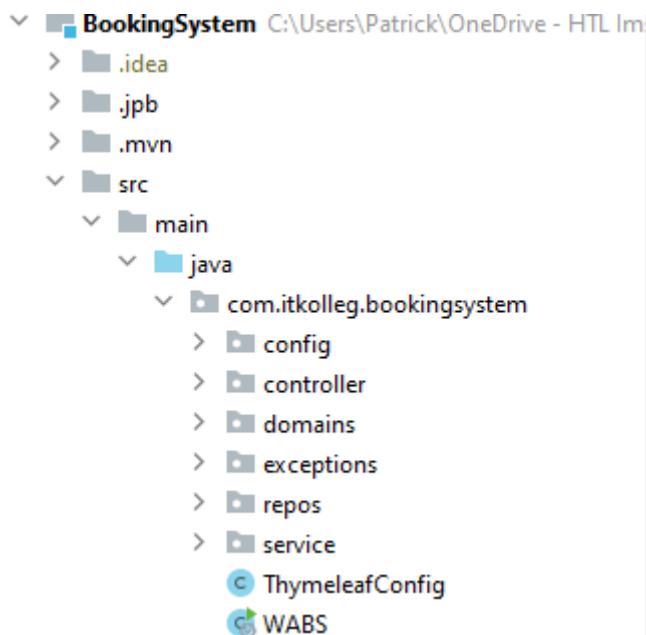


Abbildung 114- zeigt die Schichtenarchitektur

Eine Erklärung der eingesetzten Layer:

Typisch für diese Softwarearchitektur (*Abbildung 114*) ist, dass der Controller die Verantwortung für die Verarbeitung von Anfragen der Benutzer:innen oder externen Systemen übernimmt. Er empfängt HTTP-Anfragen, verarbeitet diese und leitet sie an die jeweiligen Service-Methoden weiter. Zudem ist der Controller dafür zuständig, HTTP-Antworten zu generieren und damit die Benutzeroberfläche zu manipulieren.

Die Domänen, auch Kernobjekte genannt, des Systems repräsentieren die Entitäten und sind für die Datenstrukturen und die Geschäftslogik eines Modells verantwortlich. Sie bilden sozusagen das Rückgrat der Anwendungslogik und definieren, wie Daten organisiert und manipuliert werden.

Der Repositorylayer spielt eine maßgebliche Rolle bei der Kommunikation mit der Datenbank. Er stellt die notwendigen Schnittstellen und Methoden bereit, um auf die Datenbank zuzugreifen und Daten zu lesen, zu manipulieren, zu speichern, zu löschen oder zu aktualisieren. Diese Abstraktionsebene sorgt dafür, dass andere Schichten der Anwendung nicht direkt mit der Datenbank kommunizieren müssen, was sowohl die Sicherheit als auch die Wartbarkeit der Software erhöht.

Die Serviceschicht definiert schließlich die Anwendungslogik der Software. Hier werden die verschiedenen Funktionen und Abläufe der Anwendung implementiert. Der Service-Layer interagiert über definierte Schnittstellen mit den Domänen und Repositories und dient somit als Vermittler zwischen dem Controller und dem Repository.

## 10.4 Codeauszüge und Umsetzung

Um die Struktur und die obenstehenden Technologien besser erklären zu können, folgen nun Codeauszüge und Erläuterungen anhand der einzelnen Layer der Architektur.

## 10.5 Domänenlayer

Der folgende Abschnitt beschreibt die Umsetzung des Domänenlayer.

Wichtig anzumerken ist, dass zum Modul Raum die demenstprechende Buchung der Räume gehört. Dies geschieht allerdings nicht in einer einzelnen Klasse sondern wird getrennt. In der Domänenschicht befindet sich deswegen ein eigenes Paket "*Booking*" welche die Modelle der Buchungen von beispielsweise Räumen darstellt.

Dieses Muster der Trennung von Domäne und Domänenbuchung, wie in Abbildung 115 gezeigt, zieht sich durch alle Schichten der Anwendung, so findet man in jedem Layer eine seperate Implementierung von Domäne und Domänenbuchung.

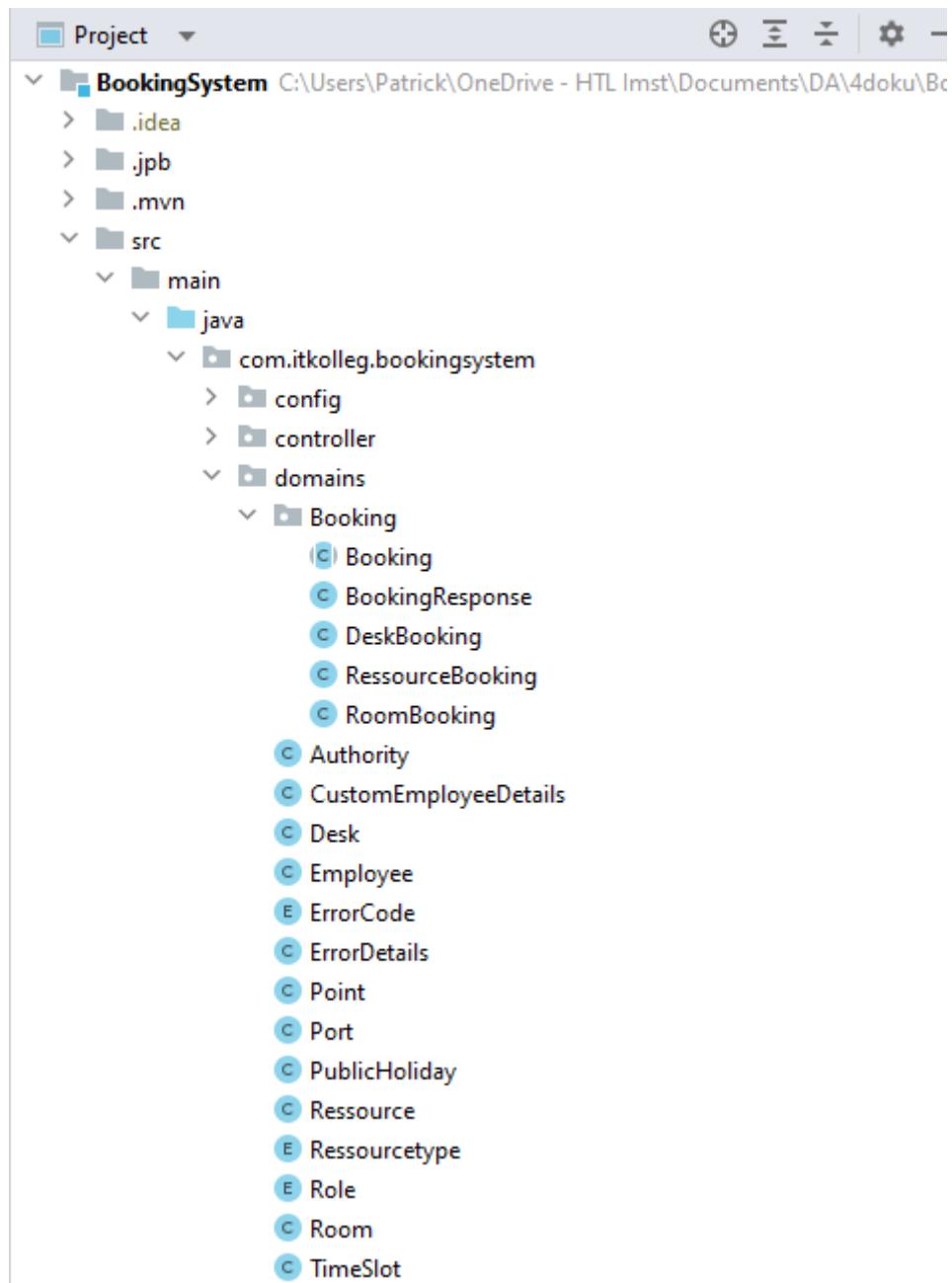


Abbildung 115 - Unterscheidung zwischen Domäne und Domänenbuchung

### 10.5.1 Klasse Room

```

3  import jakarta.persistence.*;
4  import lombok.Getter;
5  import lombok.NoArgsConstructor;
6  import lombok.Setter;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9
10
11 /**
12  * Represents a room in the booking system.
13 */
14
15 @Entity
16 @NoArgsConstructor
17 @Getter
18 @Setter
19 public class Room {
20
21     no usages
22     private static final Logger logger = LoggerFactory.getLogger(Room.class);
23
24     2 usages
25     @Id
26     @GeneratedValue(strategy = GenerationType.SEQUENCE)
27     private Long id;
28
29     2 usages
30     private String floor;
31
32     1 usage
33     private String info;
34
35     /**
36      Constructs a Room object with the specified ID and floor.
37      @param id The ID of the room.
38      @param floor The floor of the room.
39     */
40     public Room(Long id, String floor) {
41         this.id = id;
42         this.floor = floor;
43     }

```

Abbildung 116 - Codeausschnitt aus der Domänenklasse 'Room'

In den ersten Zeilen von Abbildung 116 befinden sich die Imports, welche immer, sofern benötigt, am Beginn der Klasse sind. Dies bedeutet, dass gewisse Funktionalitäten welche sich in der Klasse befinden, explizit geladen werden müssen.

Folgender Auszug stammt aus dem Domänenlayer in der Klasse Room. Dort kann man sehen, dass die bereits erwähnten Bibliotheken Jakarta und Lombok von Nöten sind. Diese wiederum werden erstmalig in Zeile 15 bis Zeile 18 in sogenannten *Annotationen* (erkennbar durch ein `@`) angewendet.

**Felder:** Die Klasse hat drei Instanzvariablen ("id", "floor", "info"). Diese repräsentieren jeweils die ID, den Stock und optionale Informationen eines Raumes.

**Konstruktor:** Neben dem parameterlosen Konstruktor (der durch die "@NoArgsConstructor"-Annotation generiert wird), gibt es auch einen expliziten Konstruktor, der id und floor als Parameter akzeptiert.

#### Jakarta Persistence

**@Entity:** Dies ist eine Jakarta Persistence Annotation, die angibt, dass die Klasse Room eine Entität ist, die in einer relationalen Datenbank gespeichert werden soll.

**@Id:** Markiert das Feld, das den Primärschlüssel der Entität repräsentiert.

**@GeneratedValue:** Gibt an, dass der Wert für das Primärschlüsselfeld automatisch generiert werden soll. Hier wird die Sequenzstrategie verwendet.

#### SLF4J (Simple Logging Facade for Java)

**Logger:** SLF4J wird hier verwendet, um einen Logger für die Klasse zu erzeugen. Dies ermöglicht das Protokollieren von Informationen, Warnungen oder Fehlern.

**Instanzvariablen:** Die Klasse hat drei Instanzvariablen (id, floor, info). Diese repräsentieren jeweils die ID, den Stock und zusätzliche Informationen eines Raumes.

### 10.5.2 Klasse RoomBooking

```

1 package com.itkolleg.booking system domains.Booking;
2 import com.itkolleg.booking system domains.Employee;
3 import com.itkolleg.booking system domains.Room;
4 import jakarta.persistence.Entity;
5 import jakarta.persistence.ManyToOne;
6 import lombok.AllArgsConstructor;
7 import lombok.Getter;
8 import lombok.NoArgsConstructor;
9 import lombok.Setter;
10
11 import java.time.LocalDate;
12 import java.time.LocalTime;
13
14 /**
15 * Represents a room booking.
16 *
17 * Extends the base class Booking.
18 */
19
20 /**
21 * @Entity
22 * @NoArgsConstructor
23 * @AllArgsConstructor
24 * @Getter
25 * @Setter
26 */
27 public class RoomBooking extends Booking {
28
29     /**
30      * Constructs a RoomBooking object with the specified employee, room, date, start time, and end time.
31      * @param employee The Employee object associated with the booking.
32      * @param room The Room object associated with the booking.
33      * @param date The date of the booking.
34      * @param start The start time of the booking.
35      * @param endTime The end time of the booking.
36      */
37
38     public RoomBooking(Employee employee, Room room, LocalDate date, LocalTime start, LocalTime endTime) {
39         super(employee, date, start, endTime);
40         this.room = room;
41     }
42
43 }
44

```

Abbildung 117- Codeausschnitt aus der Domänenklasse 'RoomBooking'

In diesem Codebeispiel (Abbildung 118) sieht man die Java-Klasse namens "RoomBooking". Mit diesem Kontext ist der Unterschied zu Abbildung 117 einfacher zu verstehen.

Es gibt einen benutzerdefinierten Konstruktor, der einen Employee, einen Room, ein LocalDate-Objekt und zwei LocalTime-Objekte als Argumente akzeptiert. Dieser Konstruktor ruft den Konstruktor der übergeordneten Booking-Klasse mit einigen dieser Parameter auf "(super(employee, date, start, endTime))" und initialisiert das room-Feld der RoomBooking-Klasse.

Dieser Code modelliert eine Raumreservierung, die spezifische Informationen wie den beteiligten Mitarbeiter, den Raum, das Datum und die Zeiten enthält. Dabei verwendet er sowohl Jakarta Persistence für die Datenpersistenz als auch Lombok zur Reduzierung von Boilerplate-Code.

**@AllArgsConstructor:** Diese Lombok-Annotation generiert einen Konstruktor mit Argumenten für alle Felder der Klasse.

**@ManyToOne:** Diese Jakarta Persistence Annotation zeigt an, dass eine "viele-zu-eins"-Beziehung zwischen der RoomBooking-Klasse und der Room-Klasse besteht. Das bedeutet, dass viele RoomBooking-Objekte auf ein einziges Room-Objekt verweisen können.

**Extends Booking:** Die RoomBooking-Klasse erweitert die Booking-Basisklasse. Dies impliziert, dass RoomBooking alle Eigenschaften und Methoden von Booking erbt.

**LocalDate** und **LocalTime**: Java 8's Date-Time-API wird verwendet, um den Buchungstag (LocalDate) und die Start-/Endzeiten (LocalTime) zu repräsentieren.

## 10.6 Repositorylayer

Der folgende Abschnitt beschreibt die Umsetzung der Architektur für den Repositorylayer.

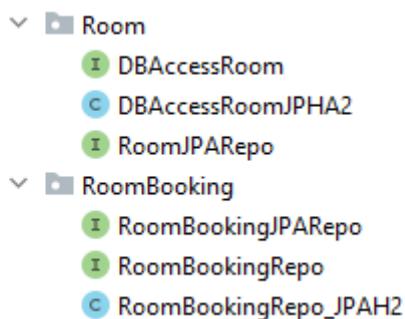


Abbildung 118 - Relevante Packages im Repositorylayer des Raummoduls

Die Packages Room und RoomBooking (*Abbildung 119*) im Repositorylayer bestehen jeweils aus 2 Interfaces und einer implementierenden Klasse. Grund dafür ist, dass man eine technologieunabhängige Erweiterbarkeit erreichen will. Dies wird realisiert, indem man die Grundfunktionalitäten der datenbankzugreifenden Ebene in einem Interface festlegt und die technologiespezifische Implementierung in der davon erbenden Klasse.

Wenn beispielsweise zukünftig eine andere Datenbanktechnologie bevorzugt werden würde, ist es nicht nötig bestehenden Code zu verändern sondern lediglich eine neue Klasse mit der gewünschten Technologie zu implementieren.

### 10.6.1 Interface DBAccessRoom

```
10 usages 1 implementation ± blauekaktus +2
public interface DBAccessRoom {
    7 usages 1 implementation ± blauekaktus
    Room addRoom(Room room) throws ExecutionException, InterruptedException;

    2 usages 1 implementation ± blauekaktus
    List<Room> getAllRooms() throws ExecutionException, InterruptedException;

    1 usage 1 implementation ± blauekaktus
    Room getRoomById(Long id) throws RoomNotFoundException, ExecutionException, InterruptedException;

    1 usage 1 implementation ± pabayr
    Room updateRoom(Room room) throws RoomNotFoundException, ExecutionException, InterruptedException;

    1 usage 1 implementation ± blauekaktus
    void deleteRoomById(Long id) throws RoomDeletionNotPossibleException;
}
```

Abbildung 119 - Codeausschnitt aus dem Repository-Interface DBAccessRoom

Wie bereits beschrieben, stellt das in Abbildung 119 gezeigte Interface lediglich sogenannte „CRUD“-Methodenköpfe zur Verfügung (**C**reate, **U**pdate, **D**elete) und gilt als technologieneutrale Anleitung der implementierenden Klassen.

### 10.6.2 Interface RoomJPARepr

```
package com.itkolleg.bookingsystem.repos.Room;

import com.itkolleg.bookingsystem.domains.Room;
import org.springframework.data.jpa.repository.JpaRepository;

5 usages ± pabayr
public interface RoomJPARepr extends JpaRepository<Room, Long> {
    1 usage ± pabayr
    Room findRoomById(Long id);
}
```

Abbildung 120 - Code des Repository-Interfaces RoomJPARepr

Dieses Java-Interface (*Abbildung 121*) dient als JPA (Java Persistence API) Repository. Es verwendet Spring Data JPA, ein Modul des Spring Frameworks, um die Arbeit mit Datenbanken zu erleichtern. Das Interface erbt von JpaRepository, welches viele Standardmethoden wie beispielsweise CRUD-Methoden bereitstellt.

**public interface RoomJPARepr:** Definiert ein öffentliches Interface namens RoomJPARepr.

**extends JpaRepository<Room, Long>:** Das Interface erweitert JpaRepository, ein generisches Interface aus dem Spring Data JPA-Modul. Das Interface JpaRepository selbst enthält Methoden für CRUD-Operationen für die Arbeit mit Datenbanken. Die generischen Typen "<Room, Long>" geben an, dass dieses Repository für die Verwaltung von Entitäten vom Typ Room mit der ID vom Typ Long verwendet wird.

**Room findRoomById(Long id):** Diese Methode ist eine benutzerdefinierte Abfragemethode. Spring Data JPA kann automatisch eine Implementierung für diese Methode bereitstellen, die ein Room-Objekt sucht, dessen id dem übergebenen Wert entspricht. Das ist möglich, weil der Methodename bestimmten Namenskonventionen folgt, die Spring Data JPA erkennt.

Insgesamt dient dieses Interface als Vertrag, den die Service-Klasse, die das Repository verwendet, erfüllen muss. Dieses Interface muss nicht manuell implementiert werden wie im nächsten Abschnitt gezeigt wird; Spring Data JPA generiert zur Laufzeit eine Implementierung.

### 10.6.3 Klasse DBAccessRoomJPHA2

```

@Component
public class DBAccessRoomJPHA2 implements DBAccessRoom {

    6 usages
    private final RoomJPARRepo roomJPARRepo;
    2 usages
    private final RoomBookingRepo roomBookingRepo;

    /**
     * no usages  ± pabayr
     */
    public DBAccessRoomJPHA2(RoomJPARRepo roomJPARRepo, RoomBookingRepo roomBookingRepo) {
        this.roomJPARRepo = roomJPARRepo;
        this.roomBookingRepo = roomBookingRepo;
    }

    /**
     * 7 usages  ± pabayr
     */
    @Override
    public Room addRoom(Room room) throws ExecutionException, InterruptedException {
        return this.roomJPARRepo.save(room);
    }

    /**
     * 2 usages  ± pabayr
     */
    @Override
    public List<Room> getAllRooms() throws ExecutionException, InterruptedException {
        return this.roomJPARRepo.findAll();
    }

    /**
     * 1 usage  ± pabayr
     */
    @Override
    public Room getRoomById(Long id) throws RoomNotFoundException, ExecutionException, InterruptedException {
        Optional<Room> roomOptional = this.roomJPARRepo.findById(id);
        if (roomOptional.isPresent()) {
            return roomOptional.get();
        } else {
            throw new RoomNotFoundException("Room not found for ID: " + id);
        }
    }

    /**
     * 1 usage  ± pabayr
     */
    @Override
    public Room updateRoom(Room room) throws RoomNotFoundException, ExecutionException, InterruptedException {
        return this.roomJPARRepo.save(room);
    }

    /**
     * 1 usage  ± pabayr
     */
    @Override
    public void deleteRoomById(Long id) throws RoomDeletionNotPossibleException {
        List<RoomBooking> bookings = this.roomBookingRepo.getBookingsByRoomId(id);

        if (!bookings.isEmpty()) {
            throw new RoomDeletionNotPossibleException("Room already booked");
        }
        this.roomJPARRepo.deleteById(id);
    }
}

```

Abbildung 121 - Implementierende Klasse des Repositorylayers

Abbildung 121 zeigt nun die eigentlich implementierte Klasse für Datenbankzugriffe der Raumobjekte, was anhand des „*implements DBAccessRoom*“ erkannt werden kann.

Die „@Component-Annotation“ ist so zu verstehen, dass das Springframework eine Instanz dieser Klasse als sogenannte *Bean* (Baeldung, 2023) im Spring-Anwendungskontext automatisch erzeugt.

```
private final RoomJPARepo roomJPARepo;  
private final RoomBookingRepo roomBookingRepo;
```

Diese Felder speichern die Repository-Objekte für Room und RoomBooking Entitäten. Über diese Objekte wird der Zugriff auf die Datenbank realisiert. Trotz strikter Trennung von Raum und Raumbuchungen ist hier für eine bestimmte Methode das „roomBookingRepo“ notwendig.

```
public DBAccessRoomJPHA2(RoomJPARepo roomJPARepo, RoomBookingRepo roomBookingRepo)
```

Der Konstruktor initialisiert die oben genannten Felder. Spring wird die erforderlichen Repository-Beans automatisch injizieren.

**public Room addRoom(Room room):** Diese Methode fügt ein Room-Objekt zur Datenbank hinzu und gibt das hinzugefügte Objekt zurück. Es verwendet die save-Methode des roomJPARepo.

**public List<Room> getAllRooms():** Diese Methode gibt eine Liste aller Room-Objekte in der Datenbank zurück. Es verwendet die findAll-Methode des roomJPARepo.

**public Room getRoomById(Long id):** Diese Methode sucht ein Room-Objekt in der Datenbank basierend auf der ID. Wenn das Objekt nicht gefunden wird, wird eine RoomNotFoundException geworfen.

**public Room updateRoom(Room room):** Diese Methode aktualisiert ein Room-Objekt in der Datenbank und gibt das aktualisierte Objekt zurück. Es verwendet ebenfalls die save-Methode des roomJPARepo.

**public void deleteRoomById(Long id):** Diese Methode löscht ein Room-Objekt aus der Datenbank basierend auf der ID. Bevor das Löschen durchgeführt wird, überprüft es, ob es Buchungen für diesen Raum gibt. Dies ist wie bei den Feldern schon beschrieben notwendig, da ansonsten die *referentielle Integrität* (wikipedia, 2023) verletzt wird.

#### 10.6.4 Interface RoomBookingRepo

```

15 ❶ public interface RoomBookingRepo {
16    ❷ implementation ± pabaygit
17    RoomBooking addBooking(RoomBooking booking) throws RoomNotAvailableException, RoomNotFoundException;
18
19    ❷ no usages 1 implementation ± pabaygit
20    List<Room> getAllRooms();
21
22    ❷ implementation ± pabaygit
23    List<RoomBooking> getAllBookings();
24
25    ❷ implementation ± pabaygit
26    Optional<RoomBooking> getBookingByBookingId(Long id);
27
28    ❷ usage 1 implementation ± pabaygit
29    List<RoomBooking> getBookingsByRoom(Room room);
30
31    ❷ usage 1 implementation ± pabaygit
32    List<RoomBooking> getBookingsByRoomId(Long roomId);
33
34    ❷ implementation ± pabaygit
35    List<RoomBooking> getBookingsByDate(LocalDate date);
36
37    ❷ implementation ± pabaygit
38    List<RoomBooking> getBookingsByEmployee(Employee employee);
39
40    ❷ implementation ± pabaygit
41    List<RoomBooking> getBookingsByEmployeeId(Long employeeId);
42
43    ❷ no usages 1 implementation ± pabaygit
44    List<RoomBooking> getBookingsByRoomAndDate(Room room, LocalDate date);
45
46    ❷ 5 usages 1 implementation ± pabaygit
47    List<RoomBooking> getBookingsByRoomAndDateAndBookingTimeBetween(Room room, LocalDate date, LocalTime start, LocalTime endTime);
48
49    ❷ implementation ± pabaygit
50    RoomBooking updateBookingById(Long id, RoomBooking updatedBooking) throws RoomNotFoundException;
51
52    ❷ implementation ± pabaygit
53    RoomBooking updateBooking(RoomBooking updatedBooking) throws RoomNotFoundException;
54
55    ❷ void deleteBookingById(Long id) throws RoomDeletionNotPossibleException;
56
57    ❷ no usages 1 implementation ± pabaygit
58    List<Room> getAvailableRooms(LocalDate date, LocalTime start, LocalTime endTime);
59
60    ❷ 2 usages 1 implementation ± pabaygit
61    boolean isRoomAvailable(Room room, LocalDate date, LocalTime start, LocalTime endTime);
62
63    ❷ 1 usage 1 implementation ± pabaygit
64    List<RoomBooking> getBookingsByDateAndByStartBetween(LocalDate date, LocalTime start, LocalTime endTime);
65
66    ❷ no usages 1 implementation ± pabaygit
67    List<RoomBooking> getBookingsByEmployeeIdAndDateAndRoomId(Long employeeId, LocalDate date, Long roomId);
68
69    ❷ implementation ± pabaygit
70    RoomBooking save(RoomBooking booking);
71
72  }
```

Abbildung 122 - Raumbuchungs Interface

Ebenso wie bei Abbildung 123 werden bei diesem Interface die Methodenköpfe definiert, welche im nächsten Schritt wieder durch die spezifische Technologie und letztendlich über "Dependency Injection" in der implementierenden Klasse zur Laufzeit generiert werden.

In dem Falle der Raumbuchungen sind allerdings mehr Methoden notwendig als wie zuvor bei den Räumen, um den Use-Cases der Mitarbeiter:innen gerecht zu werden.

## 10.7 Interface RoomBookingJPARepo

```

13
14 @Repository
15 public interface RoomBookingJPARepo extends JpaRepository<RoomBooking, Long> {
16     1 usage  ± pabayrgit
17     List<RoomBooking> getBookingsByRoom(Room room);
18
19     1 usage  ± pabayrgit
20     List<RoomBooking> getBookingsByRoomId(Long roomId);
21
22     ± pabayrgit
23     List<RoomBooking> getBookingsByDate(LocalDate date);
24
25     ± pabayrgit
26     List<RoomBooking> getBookingsByEmployee(Employee employee);
27
28     ± pabayrgit
29     List<RoomBooking> getBookingsByEmployeeId(Long employeeId);
30
31     1 usage  ± pabayrgit
32     List<RoomBooking> getBookingsByRoomAndDate(Room room, LocalDate date);
33
34     no usages  ± pabayrgit
35     List<RoomBooking> getBookingsByRoomAndEmployee(Room room, Employee employee);
36
37     ± pabayrgit
38     List<RoomBooking> getBookingsByEmployeeAndDate(Employee employee, LocalDate date);
39
40     no usages  ± pabayrgit
41     List<RoomBooking> getBookingsByEmployeeAndDateAndRoom(Employee employee, LocalDate date, Room room);
42
43     1 usage  ± pabayrgit
44     List<RoomBooking> getBookingsByDateAndStartBetween(LocalDate date, LocalTime start, LocalTime endTime);
45
46     2 usages  ± pabayrgit
47     List<RoomBooking> getBookingsByRoomAndDateAndStartBetween(Room room, LocalDate date, LocalTime start, LocalTime endTime);
48
49     1 usage  ± pabayrgit
50     List<RoomBooking> getBookingsByEmployeeIdAndDateAndRoomId(Long employeeId, LocalDate date, Long roomId);
51
52 }
```

Abbildung 123 - Technologiespezifisches Raumbuchungs-Interface

Alle relevanten Methoden werden hier in *Abbildung 123* nochmals mit Einhaltung der JPA-Namenskonvention definiert.

"`@Repository`" und "`extends JpaRepository`" in Kombination mit den Typen "`<RoomBooking,Long>`" machen die Magie möglich.

### 10.7.1 Klasse RoomBookingRepo\_JPHA2

```

1 usage  ± pabayrgit +1
29   @Component
30   @ComponentScan({"com.itkolleg.repos"})
31   public class RoomBookingRepo_JPAH2 implements RoomBookingRepo {
32       2 usages
32       private static final Logger logger = LoggerFactory.getLogger(RoomBookingRepo_JPAH2.class);
33       20 usages
33       private final RoomBookingJPARepo roomBookingJPARepo;
34       5 usages
34       private final RoomJPARepo roomJPARepo;
35       2 usages
35       private final EmployeeJPARepo employeeJPARepo;
36
37
38
39   /** Constructs a RoomBookingRepo_JPAH2 with the specified repositories. ...*/
40   no usages  ± pabayrgit
41   public RoomBookingRepo_JPAH2(RoomBookingJPARepo roomBookingJPARepo, RoomJPARepo roomJPARepo, EmployeeJPARepo employeeJPARepo) {
42       this.roomBookingJPARepo = roomBookingJPARepo;
43       this.roomJPARepo = roomJPARepo;
44       this.employeeJPARepo = employeeJPARepo;
45   }
46
47
48
49
50
51
52   /** Adds a room booking to the system. ...*/
53   ± pabayrgit
54   @Override
55   public RoomBooking addBooking(RoomBooking booking) throws RoomNotAvailableException, RoomNotFoundException {
56       if (booking == null || booking.getRoom() == null) {
57           throw new IllegalArgumentException("The RoomBooking or Room cannot be null!");
58       }
59
60       Long roomId = booking.getRoom().getId();
61       Room room = this.roomJPARepo.findRoomById(roomId);
62
63       if (!isRoomAvailable(room, booking.getDate(), booking.getStart(), booking.getEndTime())) {
64           throw new RoomNotAvailableException("Room not available for booking period!");
65       }
66
67       RoomBooking roomBooking = new RoomBooking();
68       roomBooking.setEmployee(booking.getEmployee());
69       roomBooking.setRoom(room);
70       roomBooking.setDate(booking.getDate());
71       roomBooking.setStart(booking.getStart());
72       roomBooking.setEndTime(booking.getEndTime());
73       roomBooking.setCreatedOn(LocalDateTime.now());
74       roomBooking.setUpdatedOn(LocalDateTime.now());
75
76       try {
77           return this.roomBookingJPARepo.save(roomBooking);
78       } catch (Exception e) {
79           throw new RuntimeException("Error saving the booking to the database", e);
80       }
81   }
82
83
84
85
86
87 }

```

Abbildung 124 - Auszug aus der implementierenden Klasse der Raumbuchungen

Die Klasse RoomBookingRepo\_JPAH2 (Abbildung 124) ist nun die Implementierung für die Verwaltung der Raumreservierungen, wobei das JPAH2 für die Java Persistence H2 Datenbank steht.

Die Annotation "@Component" macht diese Klasse als Spring Bean bekannt, so, dass sie von Spring verwaltet werden kann.

"@Componentscan({"com.itkolleg.repos"})" gibt an, dass Spring in dem angegebenen Paket nach weiteren Beans suchen soll, um eine fehlerhafte Ausführung zu ermöglichen.

Neu doch sehr wichtig ist hier die Angabe eines "*EmployeeJPARepo*" als Datenfeld. Der Grund dafür ist, dass eine Buchung nach Geschäftslogik mit einer Mitarbeiterin oder einem Mitarbeiter verbunden werden muss. Dementsprechend wird im Konstruktor schon automatisch das dazugehörige Repository angegeben, um Mitarbeitermethoden zu ermöglichen (z.B. Mitarbeiter:in setzen).

Der Auszug zeigt lediglich einen Bruchteil der Methoden, die diese Klasse implementiert, aber exemplarisch wird folgend die "*addBooking*" näher erläutert:

Diese Methode fügt eine neue Raumreservierung hinzu.

**Parameterüberprüfung:** Zuerst wird überprüft, ob die Buchung oder der Raum null sind. Das bedeutet, dass überprüft wird ob die Buchung oder der Raum invalide sind. Wenn ja, wird eine **IllegalArgumentException** ausgelöst.

**Verfügbarkeitsüberprüfung:** Die Methode "*isRoomAvailable*" wird aufgerufen, um zu prüfen, ob der Raum für den angegebenen Zeitraum verfügbar ist. Wenn nicht, wird eine "*RoomNotAvailableException*" ausgelöst.

**Buchungsobjekt Erstellung:** Ein neues RoomBooking-Objekt wird erstellt und mit den Daten aus der übergebenen Buchung gefüllt.

**Speichern und Rückgabe:** Das neue RoomBooking-Objekt wird in der Datenbank gespeichert, und das gespeicherte Objekt wird zurückgegeben.

Falls beim Speichern ein Fehler auftritt, wird eine *RuntimeException* ausgelöst.

Diese Methode zeigt schon sehr gut, dass über die injizierten Repositories, welche die Datenfelder dieser Klasse ausmachen, die ganzen Methoden bereit stehen und mit diesen dann weitere Logiken erfüllt werden können.

## 10.8 Servicelayer

In diesem Abschnitt wird der Servicelayer im Zuge des Raummoduls besser beschrieben.

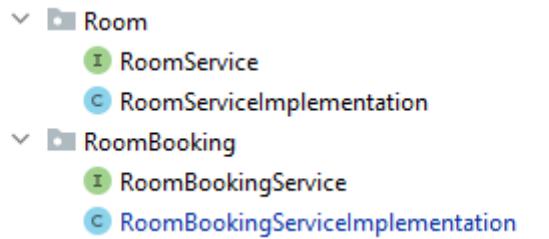


Abbildung 125 - Packages der raumrelevanten Klassen im Servicelayer

Um nicht zu repetitiv zu werden, wird ausschließlich auf die implementierenden Klassen und nicht wie in den vorherigen Teilen auch auf die vererbenden Interfaces eingegangen.

Auch wird auffallen, dass die Grundfunktionalitäten bereits im letzten Abschnitt des Repositorylayers (Abbildung 125) großteils (RoomBooking) wenn nicht vollständig (Room) erfüllt wurden. Der Servicelayer dient abermals als Abstraktionsschicht zwischen Controller- und dem Repo-Layer. Durch diese Entkopplung kann der Controller ohne Kenntnis der darunterliegenden Datenquellen oder der spezifischen Implementierungen arbeiten.

### 10.8.1 RoomServiceImplementation

```

16 no usages ± pabayr +3
17  @Service
18 public class RoomServiceImplementation implements RoomService {
19     6 usages
20     private final DBAccessRoom dbAccessRoom;
21
22     /** Constructs a new RoomServiceImplementation with the specified DBAccessRoom object. ...*/
23     no usages ± pwnzZzZz
24     public RoomServiceImplementation(DBAccessRoom dbAccessRoom) { this.dbAccessRoom = dbAccessRoom; }
25
26     /** Adds a room to the system. ...*/
27     1 usage ± pabaygit +1
28     @Override
29     public Room addRoom(Room room) throws ExecutionException, InterruptedException {
30         return this.dbAccessRoom.addRoom(room);
31     }
32
33     /** Retrieves all rooms from the system. ...*/
34     4 usages ± pabaygit +1
35     @Override
36     public List<Room> getAllRooms() throws ExecutionException, InterruptedException {
37         return this.dbAccessRoom.getAllRooms();
38     }
39
40     /** Retrieves a room by its ID. ...*/
41     3 usages ± pabaygit +1
42     @Override
43     public Room getRoomById(Long id) throws RoomNotFoundException, ExecutionException, InterruptedException {
44         return this.dbAccessRoom.getRoomById(id);
45     }
46
47     /** Updates a room in the system. ...*/
48     1 usage ± pabayr +1
49     @Override
50     public Room updateRoom(Room room) throws RoomNotFoundException, ExecutionException, InterruptedException {
51         return this.dbAccessRoom.updateRoom(room);
52     }
53
54     /** Deletes a room by its ID. ...*/
55     1 usage ± pabaygit +1
56     @Override
57     public void deleteRoomById(Long id) throws RoomDeletionNotPossibleException {
58         this.dbAccessRoom.deleteRoomById(id);
59     }
60
61 }
62

```

Abbildung 126 - Code RoomServiceImplementation

Wie bereits erwähnt in der Einleitung des Servicelayers erwähnt, wird in dieser Klasse (*Abbildung 126*) kaum neue Funktionalitäten implementiert sondern hauptsächlich eine Abstraktionsschicht geschaffen.

Erwähnenswert ist dennoch die "@Service"-Annotation, so folgt eine nähere Beschreibung dieser:

Die "@Service"-Annotation ist eine spezielle Form der "@Component"-Annotation im Spring Framework, die auf Klassenebene verwendet wird. Durch das Anbringen der "@Service"-Annotation teilt man Spring mit, dass die annotierte Klasse als Service-Komponente in einem Spring-Anwendungskontext fungieren soll.

Hauptmerkmale der "@Service" Annotation:

**Automatische Erkennung:** Spring wird automatisch eine Bean-Instanz der annotierten Klasse erstellen, wenn der Classpath gescannt wird.

**Singleton** (wikipedia, 2023): Standardmäßig wird die Klasse als Singleton behandelt, d.h., es wird nur eine Instanz der Klasse im Spring-Kontext erstellt.

**Semantische Bedeutung:** Obwohl "@Service" ähnlich wie "@Component" oder "@Repository" funktioniert, gibt sie dem Entwickler und dem Team semantische Hinweise darauf, dass die Klasse für Geschäftslogik oder Service-Aufgaben gedacht ist.

**Dependency Injection:** Diese Bean-Instanz kann in andere Spring-Komponenten wie Controller, andere Service-Klassen oder Repositories injiziert werden.

## 10.8.2 RoomBookingServiceImplementation

```

29  @Service
30  public class RoomBookingServiceImplementation implements RoomBookingService {
31
32
33      22 usages
34      private final RoomBookingRepo roomBookingRepo;
35      2 usages
36      private final DBAccessRoom dbAccessRoom;
37      1 usage
38      private final EmployeeDBAccess employeeDBAccess;
39
40
41      /** Constructs a RoomBookingServiceImplementation with the specified dependencies. ...*/
42      no usages  ± pabayrgit*
43      public RoomBookingServiceImplementation(RoomBookingRepo roomBookingRepo, DBAccessRoom dbAccessRoom,
44                                              EmployeeDBAccess employeeDBAccess) {
45          this.roomBookingRepo = roomBookingRepo;
46          this.dbAccessRoom = dbAccessRoom;
47          this.employeeDBAccess = employeeDBAccess;
48      }
49
50
51      /** Adds a new room booking. ...*/
52      2 usages  ± pabayrgit*
53      @Override
54      public RoomBooking addRoomBooking(RoomBooking roomBooking) throws RoomNotAvailableException, RoomNotFoundException {
55          List<RoomBooking> bookings = this.roomBookingRepo.getBookingsByRoomAndDateAndBookingTimeBetween(roomBooking.getRoom(),
56              roomBooking.getDate(), roomBooking.getStart(), roomBooking.getEndTime());
57          LocalDate currentDate = LocalDate.now();
58          System.out.println("Booking date: " + roomBooking.getDate());
59          System.out.println("Current date: " + LocalDate.now());
60
61          if (!bookings.isEmpty()) {
62              throw new RoomNotAvailableException("Room not available for booking period");
63          }
64
65          if (roomBooking.getDate().isBefore(currentDate)) {
66              throw new IllegalArgumentException("Cannot create booking a past date");
67          }
68          return this.roomBookingRepo.addBooking(roomBooking);
69      }
70
71      /** Retrieves all room bookings. ...*/
72      ± pabayrgit
73      @Override
74      public List<RoomBooking> getAllBookings() { return this.roomBookingRepo.getAllBookings(); }
75
76      /** Retrieves room bookings by employee ID. ...*/
77      ± pabayrgit
78      @Override
79      public List<RoomBooking> getBookingsByEmployeeId(Long employeeId) {
80          return this.roomBookingRepo.getBookingsByEmployeeId(employeeId);
81
82
83
84
85
86
87
88
89
90
91
92
93
94

```

Abbildung 127-RoomBookingService Implementierung

Ebenso wie der RoomServiceImplementation, wird auch der "RoomBookingServiceImplementation", ersichtlich in Abbildung 127, eine "@Service"-Annotation hinzugefügt um für künftige Injections die Funktionalität über Spring zu gewährleisten.

"RoomBookingRepo", "DBAccessRoom" und "EmployeeDBAccess" werden benötigt um die entsprechenden Methoden entkoppelt aufrufen zu können.

Auch wenn wieder nur ein kleiner Ausschnitt dieser Klasse gezeigt wird, ändern sich wenig Funktionalitäten zu selbigen des abstrahierten Repositories.

Anhand der "@addRoomBooking" Methode kann allerdings gesehen werden, dass Logiken noch im Servicelayer implementiert werden können. In diesem Kontext muss gesagt werden, dass nicht alle Logiken vollkommen sauber abstrahiert und entkoppelt von den Layern worden sind.

Die Methode "addRoomBooking" fügt eine neue Raum-Buchung hinzu und führt einige Überprüfungen durch:

Sie verwendet "roomBookingRepo" um zu prüfen, ob es bereits Buchungen für den gewünschten Zeitraum gibt. Wenn dies der Fall ist, wird eine "RoomNotAvailableException" geworfen.

Es wird geprüft, ob das Buchungsdatum in der Vergangenheit liegt. Wenn dies der Fall ist, wird eine "IllegalArgumentException" geworfen.

Falls beide Bedingungen erfüllt sind, wird die Buchung über "roomBookingRepo.addBooking" zur Datenbank hinzugefügt.

## 10.9 Raum Modul Frontend

In diesem Abschnitt wird näher auf die Realisierung des Frontends eingegangen. In diesem Kontext wird auch der Controllerlayer miterklärt.

### 10.9.1 Struktur, Controllerlayer und Thymeleaf

In einer Anwendung, die das Layered Architecture (Schichtenarchitektur) Muster verwendet, ist der Controller Layer oft dafür verantwortlich, Anfragen von Benutzern zu empfangen, die Geschäftslogik anzustoßen und anschließend die Ergebnisse an die View-Schicht weiterzuleiten. In Java-basierten Webanwendungen, die Spring Boot und Thymeleaf verwenden, wird dieser Ablauf in der Regel durch *Spring MVC (Model-View-Controller)* implementiert - das MVC-Muster wird nach Erklärung der Struktur später noch erläutert.

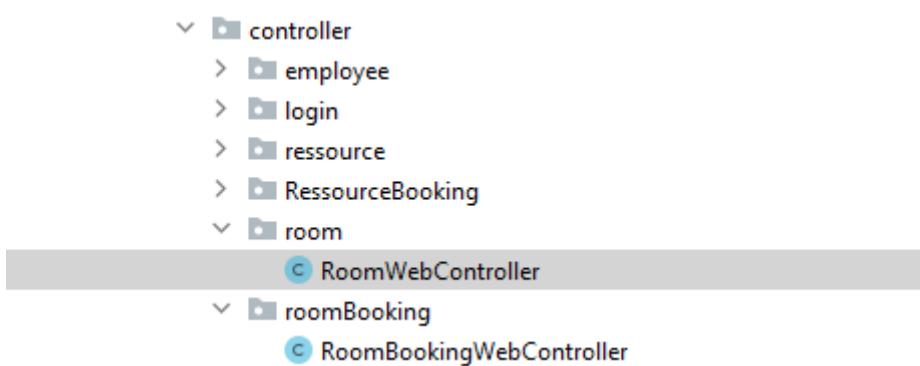


Abbildung 128- Controller-Layer

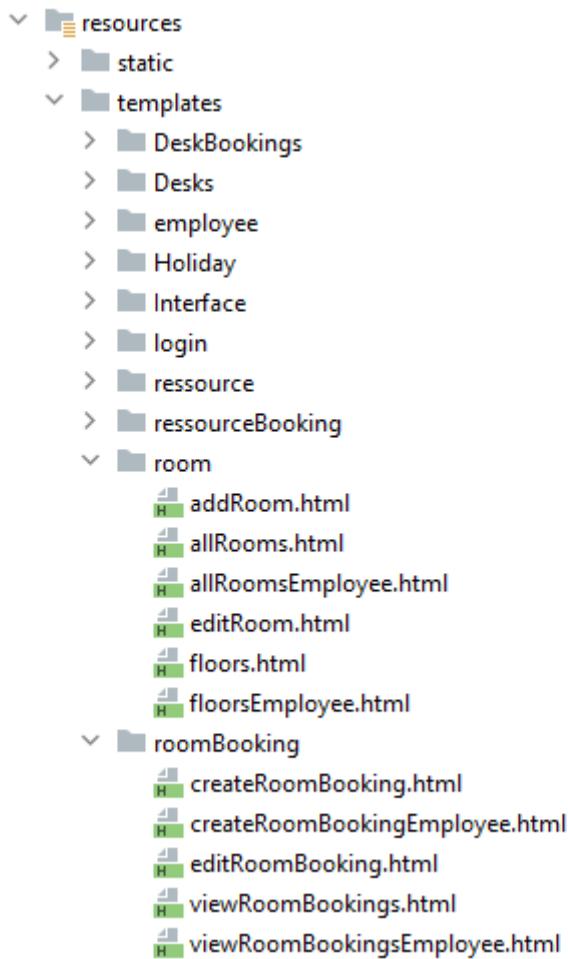


Abbildung 129- Template/View

Um nachvollziehen zu können wie diese Komponenten Abbildung 129 und Abbildung 130 miteinander interagieren folgen Codeauszüge von Controller und den Templates.

### 10.9.2 RoomWebController

```

30  @Controller
31  @RequestMapping("web/rooms")
32  public class RoomWebController {
33
34      9 usages
35      RoomService roomService;
36
37      /** Constructs a new RoomWebController with the given RoomService. ...*/
38      no usages  ± pwnz
39      public RoomWebController(RoomService roomService) { this.roomService = roomService; }
40
41      /** Handles the GET request for "/allRooms" endpoint. ...*/
42      no usages  ± pbayr +1
43      @GetMapping("allRooms")
44      public ModelAndView allRooms() throws ExecutionException, InterruptedException {
45          List<Room> allRooms = roomService.getAllRooms();
46          return new ModelAndView( viewName: "room/allRooms", modelName: "rooms", allRooms);
47      }
48
49      /** Handles the GET request for "/allRoomsEmployee" endpoint. ...*/
50      no usages  ± pbayr +1
51      @GetMapping("allRoomsEmployee")
52      public ModelAndView allRoomsEmployee() throws ExecutionException, InterruptedException {
53          List<Room> allRooms = roomService.getAllRooms();
54          return new ModelAndView( viewName: "room/allRoomsEmployee", modelName: "roomsEmployee", allRooms);
55      }
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
    }
```

Abbildung 130 – RoomWebController

Der Controller (*Abbildung 130*) ist in diesem Kontext eine Art Vermittler zwischen dem Benutzer und dem eigentlichen Service (hier "RoomService"), der die Geschäftslogik bereitstellt.

**@Controller:** Diese Annotation gibt an, dass die Klasse als Controller innerhalb des Spring Frameworks agieren soll.

**@RequestMapping("/web/rooms"):** Diese Annotation legt die Grund-*URL* für alle Methoden in dieser Controller-Klasse fest. Jede Methode im Controller, die eine *HTTP-Anfrage* behandelt, wird an diese Basis-*URL* angehängt. Zum Beispiel wird die Methode, die mit "@GetMapping("/allRooms")" annotiert ist, unter der URL "/web/rooms/allRooms" erreichbar sein.

**@GetMapping("/allRooms"):** Diese Annotation gibt an, dass die Methode allrooms() eine HTTP GET-Anfrage an der URL "/web/rooms/allRooms" verarbeiten soll. GET-Anfragen sind die häufigste Art von HTTP-Anfragen und werden in der Regel zum Abrufen von Daten verwendet.

**@GetMapping("/allRoomsEmployee"):** Ähnlich wie "@GetMapping("/allRooms")", aber für eine andere URL: "/web/rooms/allRoomsEmployee". Diese Methode wird aufgerufen, wenn ein Client eine HTTP GET-Anfrage an diese URL sendet.

Wenn also nun beispielsweise bei laufender Applikation die URL /web/rooms/allRooms aufgerufen wird, wird die Methode "allrooms" aufgerufen, was zur Folge hat, dass das Template, welches unter dem Pfad "/web/rooms/allRooms" hinterlegt ist, aufgerufen wird.

### 10.9.3 AllRooms Template

```

1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <div th:insert="headerAdmin :: headerAdmin" th:with="title='Liste aller Räume'></div>
4  <body>
5  <!-- Top Bar -->
6  <div th:insert="topbarAdmin :: topbarAdmin"></div>
7  <!-- Side Navbar -->
8  <div th:insert="navbarAdmin :: navbarAdmin"></div>
9  <!-- Content -->
10 <!-- Content -->
11 <div class="content-container">
12   <div class="allroomsContainer">
13     <h1>Raumliste</h1><br><br>
14     <div class="scroll-container">
15       <table>
16         <thead>
17           <tr>
18             <th>ID</th>
19             <th>Stockwerk</th>
20             <th> Info</th>
21             <th> Aktion</th>
22           </tr>
23         </thead>
24         <tbody>
25           <th:block th:each="room : ${rooms}">
26             <tr>
27               <td th:text="${room.id}"></td>
28               <td th:text="${room.floor}"></td>
29               <td th:text="${room.info}"></td>
30               <td>
31                 <a th:href="@{'/web/roomBooking/createBooking/' + ${room.id}}" class="button" role="button">Buchen</a>
32                 <a th:href="@{'/web/rooms/updateRoom/' + ${room.id}}" class="button" role="button">Bearbeiten</a>
33                 <a th:href="@{'/web/rooms/deleteRoom/' + ${room.id}}" class="delete-button" role="button">Löschen</a>
34               </td>
35             </tr>
36           </th:block>
37         </tbody>
38       </table>
39     </div>
40     <div class="button-row">
41       <a th:href="@{/web/rooms/addRoom}" class="add-button" role="button">neuer Raum</a>
42     </div>
43   </div>
44 </div>
45 <div class="filler"></div>
46 </body>
47 </html>

```

Abbildung 131- Template rooms/allRooms

In Bezug auf das Template, welches in Abbildung 131 zu sehen ist, wird folgend nur auf die spezifischen Besonderheiten der Thymleafengine eingegangen:

**th:insert="headerAdmin :: headerAdmin":** Das "th:insert"-Attribut fügt den Inhalt eines Fragments aus einem anderen Template in das aktuelle Template ein. Hier wird das Fragment "headerAdmin" aus dem gleichen Template eingefügt. Dem sei beigegeben, dass jede View ein Admintemplate sowie ein Employeetemplate hat, um Intuition zu fördern.

**th:with="title='Liste aller Räume'"**: Mit "th:with" können Variablen innerhalb des Blocks definiert werden, in dem sie deklariert sind. Hier wird der Variablen "title" der Wert "Liste aller Räume" zugewiesen.

**th:each="room : \${rooms}"**: Das "th:each"-Attribut wird für Iterationen verwendet. Es durchläuft die Liste "rooms" und speichert jeweils ein Element in der Variablen "room".

**th:text="\${room.id}", th:text="\${room.floor}", th:text="\${room.info}"**: Das "th:text"-Attribut setzt den Textinhalt eines HTML-Elements auf den Wert der entsprechenden Expression. Hier wird der Textinhalt der <td>-Elemente auf die Werte der Eigenschaften "id", "floor" und "info" des aktuellen "room"-Objekts gesetzt.

**th:href="@{/web/roomBooking/createBooking/' + \${room.id}}"**: "th:href" wird verwendet, um den href-Attributwert dynamisch zu setzen. Die "@{...}"-Syntax wird verwendet, um URLs zu erstellen. In diesem Fall wird die URL auf Basis der id des aktuellen "room"-Objekts dynamisch generiert.

**th:block**: "th:block" ist ein Attribut, das einen Container für Thymeleaf-Attribute bietet, der später vom HTML-Code entfernt wird. Es wird oft verwendet, um Thymeleaf-Code zu gruppieren, der auf mehrere HTML-Elemente angewendet wird, ohne zusätzliche HTML-Tags einzufügen.

Dieses Template wird dann im Browser letztendlich zu folgender Benutzeroberfläche interpretiert:

#### 10.9.4 Ansicht AllRooms Template im Browser

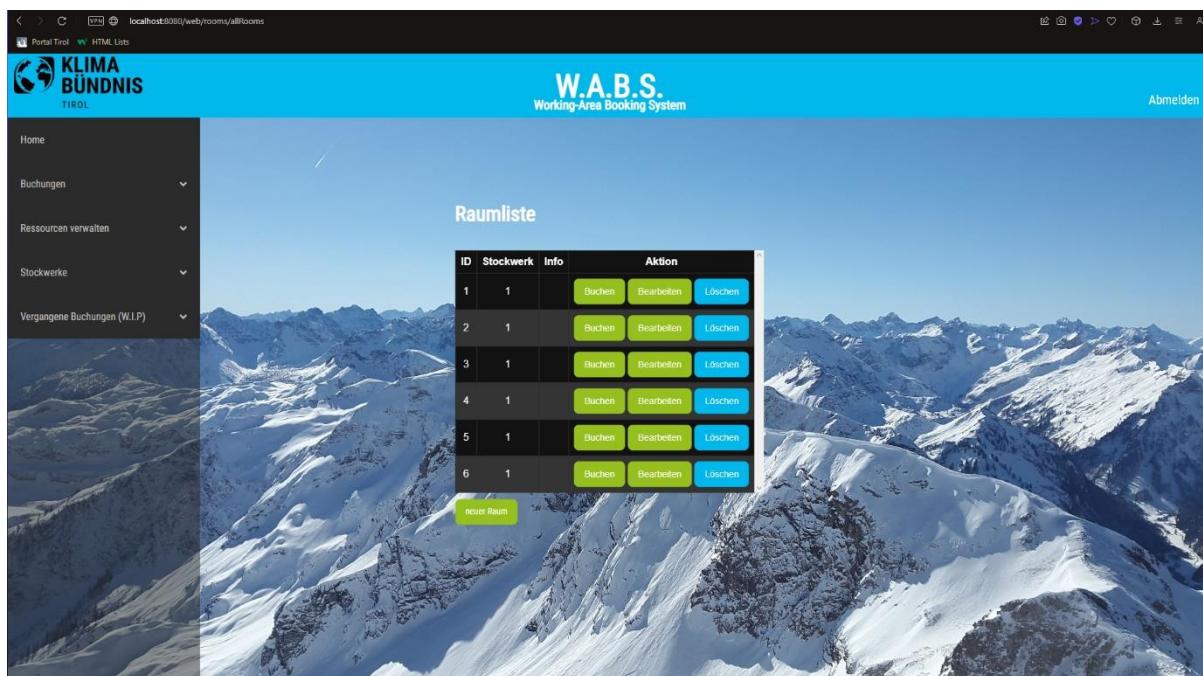


Abbildung 132 - Benutzeroberfläche Admin der Raumansicht als Liste

Wie in Abbildung 132 zu sehen ist dies eine Ansicht aller Räume im Administratormodus. Hier hat der Administrator die Möglichkeit, Räume zu bearbeiten, buchen, löschen und neue Räume zu erstellen. Das Bearbeiten, Löschen und Hinzufügen der Räume ist ausschließlich Administratoren möglich.

#### 10.9.5 Zusammenfassung MVC

Das MVC-Muster unterteilt die Anwendungslogik in drei grundlegende Komponenten:

Das "Modell" repräsentiert die Daten und die Geschäftslogik der Anwendung. Es enthält die Kernfunktionalitäten und -daten. Änderungen im Modell werden normalerweise an die View weitergeleitet, damit der Benutzer die neuesten Informationen sieht. Das Modell weiß nichts über die View und den Controller, was die Unabhängigkeit der Geschäftslogik von der Benutzeroberfläche fördert. Somit kann man sagen, dass der Domänen/Repositorylayer das Modell realisiert.

Die "View" ist verantwortlich für die Darstellung der Daten, die im Modell gespeichert sind. Sie abonniert das Modell und wird automatisch aktualisiert, wenn das Modell sich ändert. In einer Webanwendung kann die View durch HTML-Templates realisiert werden, eventuell ergänzt durch clientseitigen Code (JavaScript), mehr dazu folgend.

Der "Controller" nimmt Benutzereingaben entgegen und aktualisiert das Modell oder die View entsprechend. In einer Webanwendung könnten das beispielsweise HTTP-Requests sein, die

von Benutzern kommen. Der Controller enthält die Anwendungslogik, die nicht zum Modell gehört, und dient als Verbindung zwischen dem Modell und der View.

Zusammenarbeit der Komponenten:

Der Benutzer interagiert mit der View, welche daraufhin eine Anfrage an den Controller sendet.

Der Controller verarbeitet die Anfrage, führt die entsprechende Geschäftslogik aus (manchmal nach Änderungen am Modell) und aktualisiert dann die View.

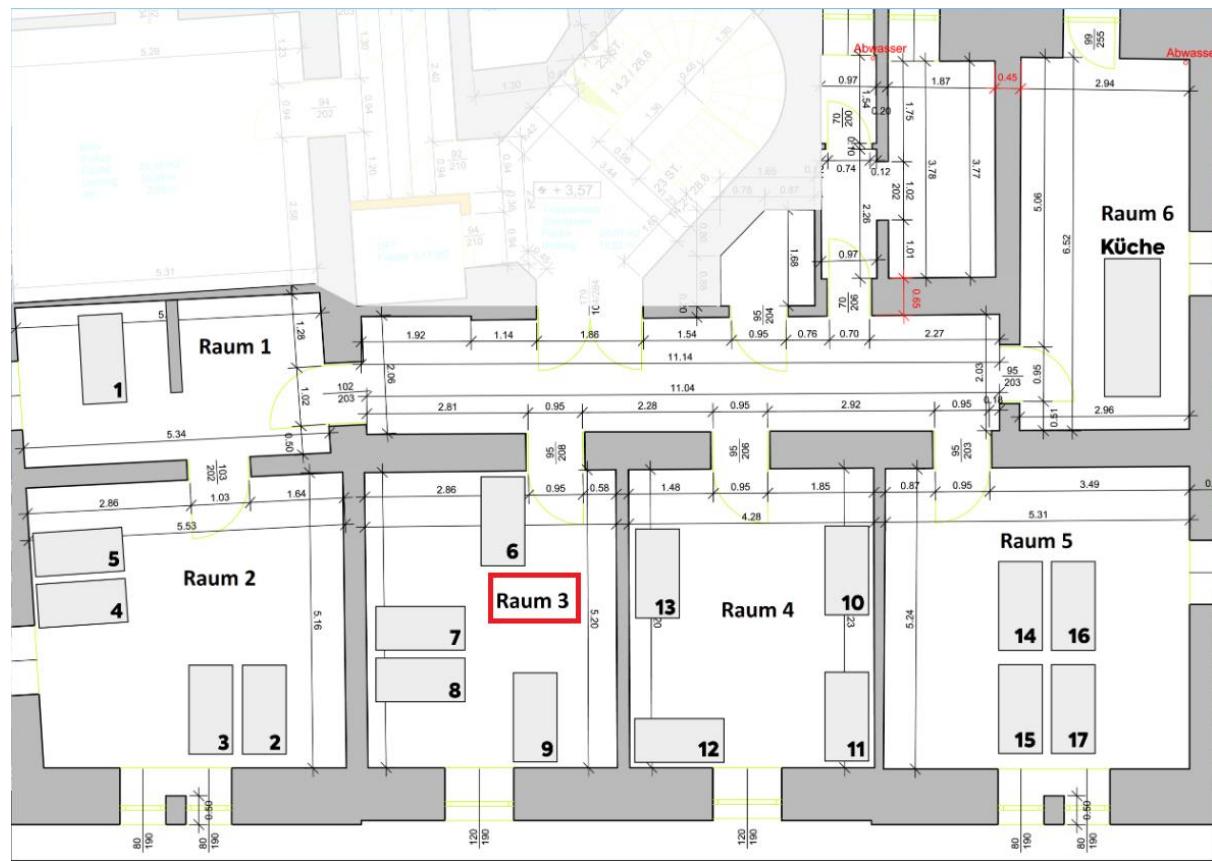
Die View holt die neuesten Daten vom Modell und zeigt sie dem Benutzer.

## 10.10 Kernmodul Raumansicht, Raumauswahl und Buchung

Nach einer ausführlichen Evaluation diverser Konzepte zur Optimierung der Raum- und Arbeitsplatzbuchung in Gebäudekomplexen wurde in Kooperation mit dem Klimabündnis eine besonders effiziente und benutzerfreundliche Lösung implementiert. Diese Initiative hatte nicht nur den Zweck, die Buchungsprozesse zu vereinfachen, sondern auch, einen nachhaltigen und ressourcenschonenden Umgang mit den verfügbaren Räumlichkeiten zu fördern.

Die ausgewählte Methode basiert auf der Verwendung eines digitalen Stockwerkplans, der mittels HTML-ImageMaps interaktiv gestaltet wurde. In dieser digitalen Darstellung sind sämtliche Räume und Arbeitsplätze des betreffenden Stockwerks präzise eingezeichnet und beschriftet. Durch die Integration von HTML-ImageMaps wurde es möglich, klickbare Bereiche direkt auf den Plan zu legen.

Der Vorteil dieser Implementierung besteht darin, dass die Nutzer eine visuelle und intuitive Übersicht über alle verfügbaren Optionen erhalten. Ein einfacher Mausklick auf den jeweiligen Raum oder Arbeitsplatz führt die Nutzer direkt zur entsprechenden Buchungsseite. Dies erhöht nicht nur die Benutzerfreundlichkeit, sondern trägt auch direkt dazu bei, die ursprüngliche Anforderung der Klimabündnisses zu erfüllen.



*Abbildung 133 - Abbildung des Grundrisses*

In Referenz auf Abbildung 135 sei angemerkt, dass der rote Rahmen um Raum 3 lediglich das klickbare Element veranschaulichen soll.

Wenn ein Nutzer auf einen Raum klickt oder auf einen Arbeitsplatz (1-17) wird er direkt weitergeleitet auf eine Buchungsseite für das ausgewählte Objekt.

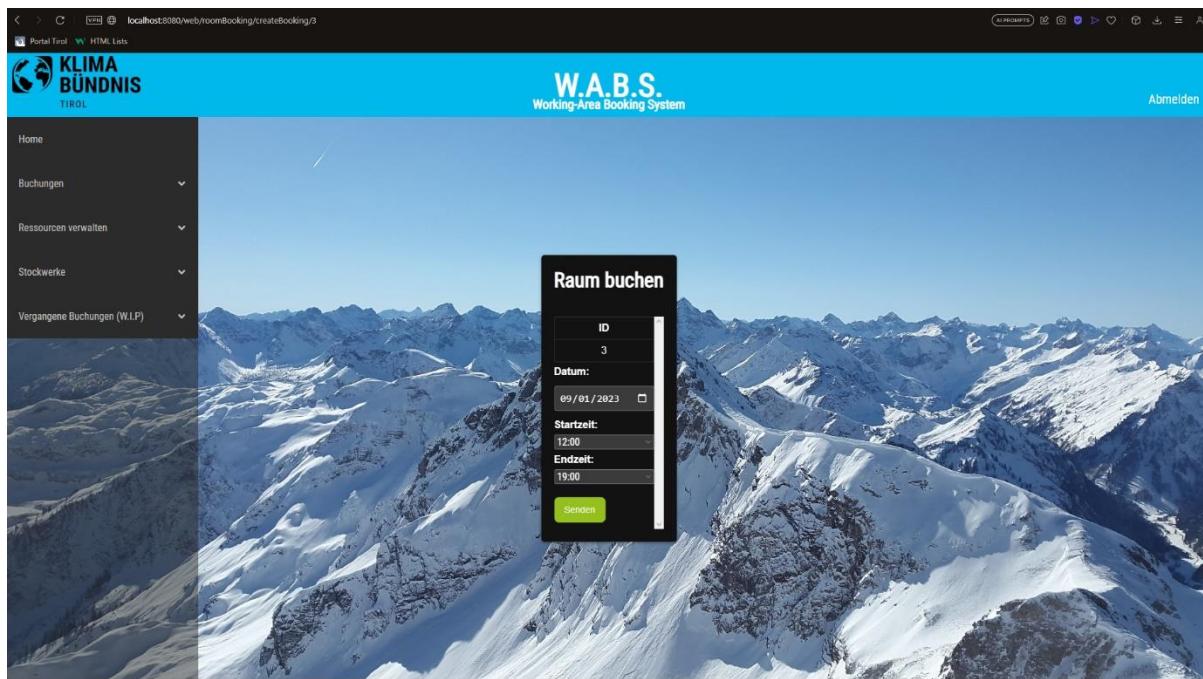


Abbildung 134 - Ansicht nach Klick auf Raum 3

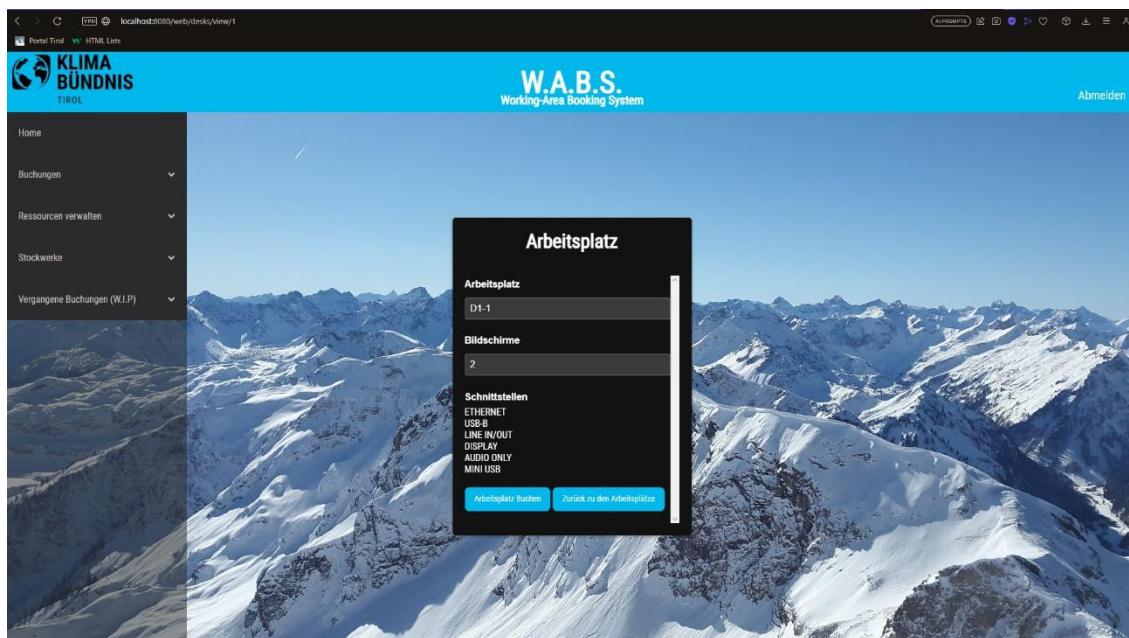


Abbildung 135- Ansicht nach Klick AP1

## 10.11 Codeauszüge zum Kernmodul

Folgend wird die Funktionsweise anhand von Codeausschnitten erklärt.

```

9   <!-- Content -->
10  <div class="content-container">
11    <div class="imgmapContainer">
12      <h1>Stockwerk 1</h1>
13      
15
16      <map name="image-map" id="image-map">
17        <!-- Image Map Generated by http://www.image-map.net/ -->
18        <area target="" alt="WorkArea1" title="WorkArea1" href="/web/desks/view/1" coords="267,1257,462,1257,471,1621,293,1630"
19          shape="poly">
20        <area target="" alt="WorkArea2" title="WorkArea2" href="/web/desks/view/2"
21          coords="951,2711,1134,2711,1134,3078,951,3078" shape="poly">
22        <area target="" alt="WorkArea3" title="WorkArea3" href="/web/desks/view/3"
23          coords="733,2711,917,2711,917,3078,733,3078" shape="poly">
24        <area target="" alt="WorkArea4" title="WorkArea4" href="/web/desks/view/4"
25          coords="92,2378,470,2355,481,2539,115,2562" shape="poly">
26        <area target="" alt="WorkArea5" title="WorkArea5" href="/web/desks/view/5"
27          coords="92,2160,458,2138,470,2321,92,2344" shape="poly">
28        <area target="" alt="WorkArea6" title="WorkArea6" href="/web/desks/view/6"
29          coords="1925,1920,2120,1931,2120,2298,1937,2310" shape="poly" >
30        <area target="" alt="WorkArea7" title="WorkArea7" href="/web/desks/view/7"
31          coords="1501,2459,1868,2470,1868,2654,1501,2654" shape="poly" >
32        <area target="" alt="WorkArea8" title="WorkArea8" href="/web/desks/view/8"
33          coords="1501,2676,1868,2676,1868,2860,1501,2860" shape="poly" >
34        <area target="" alt="WorkArea9" title="WorkArea9" href="/web/desks/view/9"
35          coords="2063,2734,2246,2734,2246,3112,2063,3112" shape="poly" >
36        <area target="" alt="WorkArea10" title="WorkArea10" href="/web/desks/view/10"
37          coords="3346,2126,3530,2126,3541,2504,3346,2504" shape="poly" >
38        <area target="" alt="WorkArea12" title="WorkArea12" href="/web/desks/view/11"
39          coords="2567,2929,2934,2929,2945,3112,2567,3112" shape="poly" >
40        <area target="" alt="WorkArea11" title="WorkArea11" href="/web/desks/view/12"
41          coords="3346,2734,3530,2734,3541,3124,3346,3101" shape="poly" >
42        <area target="" alt="WorkArea13" title="WorkArea13" href="/web/desks/view/13"
43          coords="2567,2149,2750,2149,2750,2516,2567,2516" shape="poly">
44        <area target="" alt="WorkArea14" title="WorkArea14" href="/web/desks/view/14"
45          coords="4068,2275,4068,2654,4252,2654,4252,2275" shape="poly">
46        <area target="" alt="WorkArea15" title="WorkArea15" href="/web/desks/view/15"
47          coords="4068,2699,4252,2699,4252,3078,4068,3078" shape="poly">
48        <area target="" alt="WorkArea16" title="WorkArea16" href="/web/desks/view/16"
49          coords="4286,2275,4458,2275,4469,2654,4286,2642" shape="poly">
50        <area target="" alt="WorkArea17" title="WorkArea17" href="/web/desks/view/17"
51          coords="4286,2699,4469,2699,4469,3078,4286,3078" shape="poly">
52        <area target="" alt="Raum1" title="Raum1" href="/web/roomBooking/createBooking/1" coords="770,1326,1081,1444" shape="rect">
53        <area target="" alt="Raum2" title="Raum2" href="/web/roomBooking/createBooking/2" coords="688,2272,1025,2434" shape="rect">
54        <area target="" alt="Raum3" title="Raum3" href="/web/roomBooking/createBooking/3" coords="1976,2360,2305,2514" shape="rect">
55        <area target="" alt="Raum4" title="Raum4" href="/web/roomBooking/createBooking/4" coords="2913,2375,3249,2565" shape="rect">
56        <area target="" alt="Raum5" title="Raum5" href="/web/roomBooking/createBooking/5" coords="4040,2097,4435,2258" shape="rect">
57        <area target="" alt="Küche" title="Raum6" href="/web/roomBooking/createBooking/6" coords="4148,200,4859,211,4859,1748,4160,1748"
58          shape="poly">
59      </map>
60    </div>
61  </div>

```

Abbildung 136- Umsetzung der Imagemap

In dem Codebeispiel (*Abbildung 136*) wird eine klickbare Image-Map implementiert, die das erste Stockwerk eines Gebäudes visualisiert. Diese interaktive Karte bietet Benutzern die Möglichkeit, bestimmte Arbeitsbereiche und Räume zu buchen. Die gesamte Struktur ist in einem HTML-Container eingebettet, der als "content-container" bezeichnet wird. Innerhalb dieses Hauptcontainers gibt es einen spezialisierten Untercontainer namens "imgmapContainer", der das eigentliche Bild und die Image-Map beherbergt.

Das zentrale Element der Implementierung ist das Bild, das den Grundriss des Stockwerks darstellt. Dieses Bild ist im HTML-Tag unter der ID "image" und dem Attribut "usemap" gespeichert, welches auf die eigentliche Image-Map verweist. Die Image-Map selbst wird durch den HTML-Tag `<map>` mit der ID "image-map" definiert.

Innerhalb der Image-Map gibt es mehrere "Flächen" oder "Areas", die durch den "<area>"-Tag gekennzeichnet sind. Jede dieser Flächen besitzt spezifische Attribute, die verschiedene Funktionen erfüllen. Das "target"-Attribut beispielsweise bestimmt, wie der neue Inhalt nach einem Klick geladen wird, während "alt" und "title" Textbeschreibungen für Barrierefreiheit und Tooltips bieten. Das "href"-Attribut enthält die URL, zu der der Benutzer nach dem Klicken navigiert wird. Die "coords"- und "shape"-Attribute definieren die Form und die Position der klickbaren Bereiche auf dem Bild.

### 10.11.1 Verwendung von JavaScript

Eine der großen Herausforderungen im Projekt war die Anpassung des Grundrisses an verschiedene Bildschirmgrößen und -auflösungen. Der ursprüngliche Grundriss hatte eine feste Pixelgröße von 4962 x 3509. In einem digitalen Umfeld kann dies problematisch sein, insbesondere wenn der Grundriss als interaktive Karte (ImageMap) mit festgelegten Klickbereichen dient. Diese Klickbereiche, die anhand von absoluten Pixelkoordinaten definiert sind, würden ihre Position verlieren oder ungenau werden, wenn das Bild skaliert wird, um auf verschiedenen Bildschirmen angezeigt zu werden.

Zur Lösung dieses Problems kam JavaScript zum Einsatz. Bei jedem Laden der Seite oder bei Größenänderungen des Browserfensters werden die Koordinaten und die Bildschirmgröße automatisch angepasst. Auf diese Weise funktionieren die absolut festgelegten Klickbereiche der interaktiven Karte so, als wären sie relativ zur Bildgröße. Dies wird durch spezielle Ereignisbeobachter ("eventListener") (Abbildung 137) für die "load"- und "resize"-Ereignisse ermöglicht. Dadurch bleibt die Funktionalität der interaktiven Karte erhalten, unabhängig davon, wie das Browserfenster skaliert wird.

```
<script>

    window.addEventListener('load', scaleImageMap);
    window.addEventListener('resize', scaleImageMap);

</script>
```

Abbildung 137 - Einbindung JavaScript

```

1 4 usages  ± pabayr
2 function scaleImageMap() {
3     var image = document.getElementById( elementId: 'image' );
4     var map = document.querySelector( selectors: 'map[name="image-map"]' );
5     var areas = map.getElementsByTagName( qualifiedName: 'area' );
6
7     var widthRatio = image.clientWidth / image.naturalWidth;
8     var heightRatio = image.clientHeight / image.naturalHeight;
9
10    for (var i = 0; i < areas.length; i++) {
11        var coords = areas[i].getAttribute( qualifiedName: 'coords' ).split( separator: ',' );
12        var scaledCoords = [];
13
14        for (var j = 0; j < coords.length; j++) {
15            if (j % 2 === 0) {
16                // X coordinate
17                scaledCoords.push(Math.round( x: coords[j] * widthRatio));
18            } else {
19                // Y coordinate
20                scaledCoords.push(Math.round( x: coords[j] * heightRatio));
21            }
22
23        areas[i].setAttribute( qualifiedName: 'coords', scaledCoords.join(',') );
24    }
25
26
27 }

```

Abbildung 138 - scaleImageMap-Funktion

Zu Beginn holt die Funktion (*Abbildung 138*) Informationen zum Bild und zur zugehörigen ImageMap direkt aus dem HTML-Dokument. Das Bild selbst und alle Klickbereiche innerhalb der ImageMap werden identifiziert und in Variablen gespeichert.

Daraufhin berechnet die Funktion das Verhältnis zwischen der aktuellen Größe des Bildes und seiner ursprünglichen Größe. Dies ist wichtig, weil die Klickbereiche in der ImageMap in der Regel auf die ursprüngliche Bildgröße abgestimmt sind. Um diese Bereiche korrekt zu skalieren, muss man wissen, wie stark das Bild verkleinert oder vergrößert wurde.

Anschließend geht die Funktion durch alle definierten Klickbereiche durch und passt deren Koordinaten an. Für jeden Bereich werden die alten Koordinaten mit den berechneten Skalierungsfaktoren multipliziert. Dabei wird unterschieden, ob es sich um eine X- oder eine Y-Koordinate handelt, und entsprechend der Skalierungsfaktor für die Breite oder die Höhe angewendet.

Schließlich werden die neu berechneten, skalierten Koordinaten wieder in die ImageMap eingefügt. Das Ergebnis ist eine ImageMap, deren Klickbereiche genau mit dem skalierten Bild übereinstimmen, sodass die Interaktivität der Karte unabhängig von der Bildschirmgröße erhalten bleibt.

# 11 Deskbooking Modul

## 11.1 Einführung

Das Deskbooking-Modul ermöglicht Benutzern die effiziente und benutzerfreundliche Verwaltung von Arbeitsplätzen.

Das Deskbooking-Modul verfolgt zwei Ziele: die Optimierung der Arbeitsplatznutzung und die Verbesserung der Benutzererfahrung. Es besteht aus drei Hauptelementen: Desks, Deskbookings und Hilfskomponenten. Desks sind die Arbeitsplätze, die gebucht werden können. Deskbookings sind die Buchungen von Arbeitsplätzen.

Die Hilfskomponenten erleichtern die Integration des Deskbooking-Moduls mit anderen Modulen des WABS-Systems und bieten zusätzliche Funktionen für die Benutzer, z. B. die Möglichkeit, Arbeitsplätze nur zur bestimmte Zeiträumen zu buchen oder Buchungen an bestimmten Tagen zu verhindern.

In den folgenden Abschnitten wird detailliert auf die verschiedenen Aspekte des Deskbooking-Moduls eingegangen, einschließlich Code-Ausschnitte, Domäne, Repository, Service, und den Präsentation Schichten.

## 11.2 Struktur der WABS-Anwendung

WABS basiert auf der Schichtarchitektur, einem der am weitesten verbreiteten Architekturen in der Softwareentwicklung. Diese Architektur teilt die Anwendung in mehrere horizontalen Schichten auf, wobei jede Schicht spezifische Aufgaben und Verantwortlichkeiten hat.

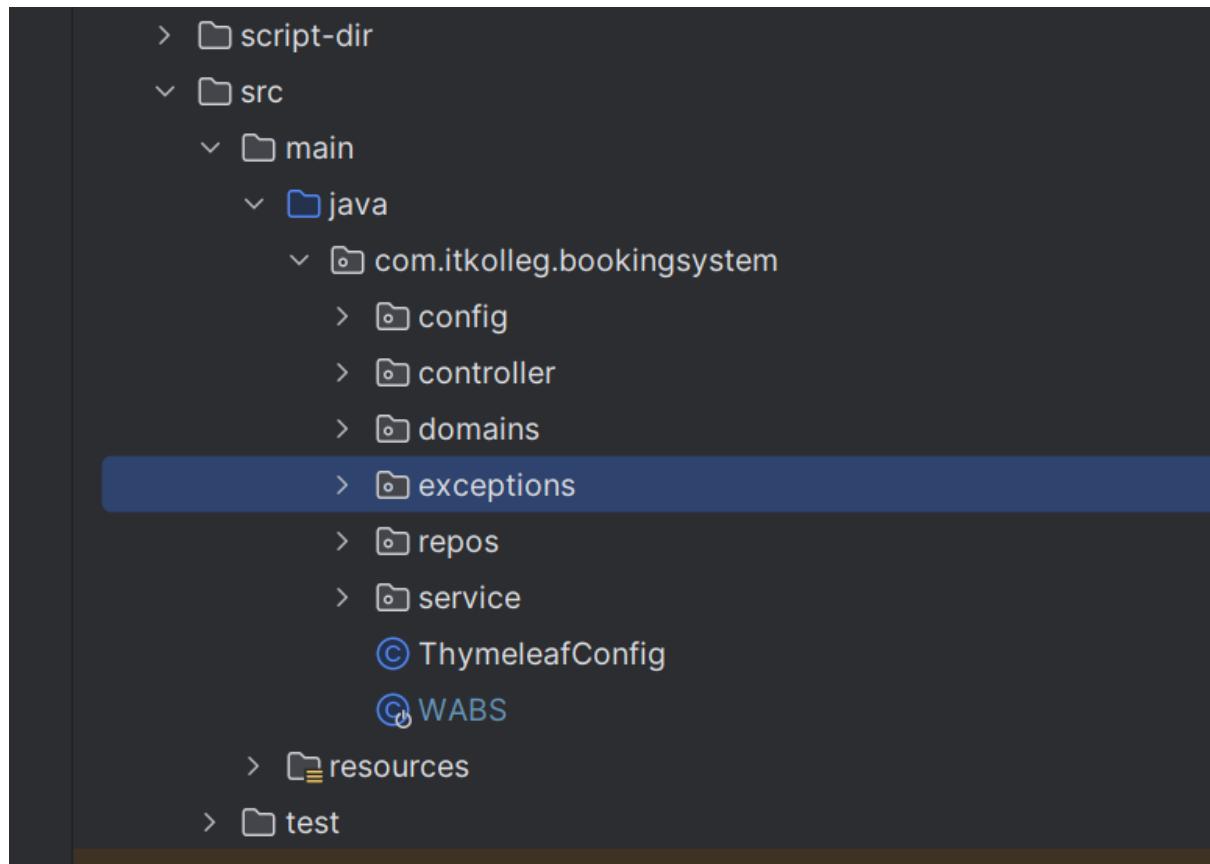


Abbildung 139 - Übersicht - Projektstruktur

### 11.2.1 Vorteile der Schichtenarchitektur

Die schichtbasierte Architektur hat viele Vorteile, die das Erstellen und Pflegen von Software einfacher macht. Sie teilt die Aufgaben klar auf, was den Code leichter lesbar und wartbar macht.

Dank dieser klaren Aufteilung ist es einfacher, Fehler zu finden und zu beheben. Das macht das System stabiler und zuverlässiger. Ein weiterer Vorteil ist, dass man Teile des Systems ändern oder erweitern kann, ohne alles neu machen zu müssen.

Die Benutzeroberfläche ist der Teil, den die Nutzer sehen und verwenden. Sie zeigt Informationen an und nimmt Eingaben entgegen. Die Geschäftslogik im Zentrum des Systems bestimmt, wie die Teile miteinander arbeiten. Die Datenzugriffsschicht kümmert sich um die Datenbank und sorgt dafür, dass Daten richtig gespeichert und abgerufen werden. Es gibt auch zusätzliche Schichten, die andere wichtige Aufgaben übernehmen, wie das Aufzeichnen von Informationen (Logging) oder das Behandeln von Fehlern.

Für Web-Programme, wie WABS, ist diese Schichtenarchitektur sehr nützlich. Sie kann leicht erweitert werden, wenn mehr Leistung benötigt wird. Die klare Struktur hilft Entwicklern, besser zusammenzuarbeiten und Fehler schnell zu finden. Weil alles so klar getrennt ist, kann man Teile des Systems leicht testen, was die Qualität der Software verbessert.

Diese Architektur stellt eine gute Basis für die Entwicklung dar. Sie sorgt dafür, dass WABS gut funktioniert und dabei flexibel und effizient bleibt.

## 11.3 Domäne

Die Domänenschicht repräsentiert die Geschäftslogik, die Geschäftsregeln und die Kernfunktionalitäten des Systems. In diesem Abschnitt liegt der Fokus auf dem Deskbooking-Modul, insbesondere auf "*Desks*", welche Arbeitsplätze repräsentieren, sowie auf "*Deskbookings*", welche eine Erweiterung des Booking-Interfaces ist und Buchungen der Arbeitsplätze darstellen sollen.

Die Domäne enthält die Businesslogik des Systems. Sie definiert, wie die verschiedenen Entitäten und Objekte im System interagieren und wie Geschäftsprozesse ablaufen. Weiteres ist diese Schicht unabhängig von anderen Schichten wie der Datenzugriffsschicht oder der Präsentationsschicht. Dies ermöglicht eine klare Trennung der Verantwortlichkeiten und macht den Code wartbarer und testbarer.

Durch die Isolierung der Geschäftslogik in der Domänenschicht kann diese Logik in verschiedenen Teilen des Systems, sogar in anderen Anwendungen wiederverwendet werden. Änderungen in der Geschäftslogik können in der Domänenschicht vorgenommen werden, ohne, dass andere Teile des Systems beeinflusst werden. Dies erleichtert die Anpassung der Funktionen an sich ändernden Geschäftsanforderungen. Ein Domänenmodell dient als gemeinsame Sprache zwischen Entwicklern und Stakeholdern. Es hilft, Missverständnisse zu vermeiden und stellt sicher, dass alle Beteiligten eine klare Vorstellung von den Geschäftsregeln und -prozessen haben.

Die Domäne stellt sicher, dass das System die Geschäftsanforderungen korrekt erfüllt und bietet eine solide Basis für die Entwicklung und Erweiterung des Systems.

### 11.3.1 Desk

Der "*Desk*" repräsentiert einen Arbeitsplatz innerhalb des Systems. Es handelt sich um eine der wichtigsten Entitäten, die verschiedene Attribute aufweist, um die spezifischen Eigenschaften und Merkmale eines Arbeitsplatzes zu definieren. Er beinhaltet Details, wie eine eindeutige Nummer (ID), Anzahl der Monitore und zugehörige Ports und bietet Hilfsmethoden zur Verwaltung von Ports und zur Überprüfung der Verfügbarkeit des Arbeitsplatzes.

Beim Betrachten der Attribute des Desk-Objekts sieht man die eindeutige Identifikationsnummer des Arbeitsplatzes, die als "*id*" bezeichnet wird. Dann gibt es die "*deskNr*", die als Nummer des Arbeitsplatzes dient und für Identifikationszwecke verwendet wird. Die "*nrOfMonitors*" zeigt, wie viele Monitore am Arbeitsplatz installiert sind. Schließlich gibt es noch "*ports*", eine Sammlung von Anschlüssen, die dem Arbeitsplatz verfügbar sind (Abbildung 140 – Attribute der Desk Klasse).

```

package com.itkolleg.bookingsystem.domains;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;
import lombok.extern.slf4j.Slf4j;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

/** Represents a desk within the system. ...*/
@Entity
@Getter
@Setter
@ToString
@NoArgsConstructor
@Slf4j
public class Desk {
    /** A list of ports available at the desk. */
    @NotEmpty(message = "Ports must not be empty")
    @Size(min = 1, message = "At least one port is required")
    @ElementCollection
    private List<Port> ports = new ArrayList<>();

    /** The identifier for the desk. */
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    /** The desk number. */
    @NotEmpty(message = "desk number must not be empty")
    @Size(max = 10, message = "desk number must not exceed 10 characters")
    private String deskNr;

    /** The number of monitors at the desk. */
    @Digits(integer = 2, fraction = 0, message = "Number of monitors must be a positive integer (0 - 99)")
    @Min(value = 0, message = "Number of monitors can't be less than 0")
    @Max(value = 10, message = "Number of monitors can't exceed 10")
    private int nrOfMonitors;
}

```

Verantwortlich für den Inhalt: Sonja Lechner

Abbildung 140 – Attribute der Desk Klasse

Unter den Methoden des Desk, gibt es einen Konstruktor, der einen neuen Arbeitsplatz mit einer bestimmten Nummer, einer bestimmten Anzahl von Monitoren und einer bestimmten Portliste erstellt. Die "equals()" Methode ist eine Standardfunktion, die Desk-Objekte anhand ihrer ID vergleicht. Mit "hashCode()" wird ein Hash-Code generiert, der auf der ID des Desks basiert. Die "addPort()" Funktion ermöglicht das Hinzufügen eines Anschlusses zur Portliste des Arbeitsplatzes, während

"`deleteAllPorts()`" alle Anschlüsse aus dieser Liste entfernt(*Abbildung 141 – Methode der Desk Klasse*).

```
/** Overrides the default equals method to compare Desk objects based on their unique identifier. ...*/
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Desk desk = (Desk) o;
    return Objects.equals(id, desk.id);
}

/** Overrides the default hashCode method to generate a hash code based on the Desk's unique identifier. ...*/
@Override
public int hashCode() {
    return Objects.hash(id);
}

/** Constructor for creating a desk with a specific desk number, number of monitors, and a list of ports. */
public Desk(@NotNull String deskNr, int nrOfMonitors, List<Port> ports) {
    this.deskNr = deskNr;
    this.nrOfMonitors = nrOfMonitors;
    if (ports != null) {
        this.ports = ports;
    }
}

/** Method to add a port to the list of ports for the desk. */
public void addPort(Port port) {
    if (port != null && !this.ports.contains(port)) {
        this.ports.add(port);
        log.info("Port " + port + " was created and added to desk: " + deskNr + ".");
    }
}

/** Method to delete all ports from the list of ports for the desk. */
public void deleteAllPorts() {
    if (!this.ports.isEmpty()) {
        this.ports.clear();
        log.info("All ports have been removed from desk: " + deskNr + ".");
    }
}
```

*Abbildung 141 – Methode der Desk Klasse*

Die Annotation "`@Slf4j`" stellt Logging-Funktionen bereit. "`@Entity`" gibt an, dass die Klasse eine Entität ist und einer Datenbanktabelle zugeordnet ist. "`@NoArgsConstructor`" ist eine Lombok-Annotation, die einen Konstruktor ohne Argumente bereitstellt. "`@Getter`" und "`@Setter`": Lombok-Annotationen, die die Getter- und Setter-Methoden für die Klassenattribute generieren. (Baeldung Lombok, 2023)

### 11.3.2 Port

Der "Port" repräsentiert eine Schnittstelle, die am Arbeitsplatz verfügbar ist. Ein Port ist eine wesentliche Komponente einer Arbeitsplatzkonfiguration und bietet Anschlussmöglichkeiten für verschiedene Geräte. Der Port hat ein Attribut "*name*", welches seinen Typ oder seine Funktionalität beschreibt (z.B. "USB", "HDMI").

Diese Klasse ist als *@Embeddable* markiert, was bedeutet, dass ihre Instanzen als Teil der Besitzentität, in diesem Fall dem Desk, eingebettet werden. Das bedeutet, dass die Attribute des Ports Teil der Desk-Tabelle im Datenbankschema sein werden und nicht eine separate Tabelle.

**Der Konstruktor** initialisiert einen Port mit einem spezifischen Namen und die *toString()* Methode gibt einen String des Ports zurück. Die *@Size*-Annotation beim *name*-Attribut stellt sicher, dass der Name des Ports mindestens 2 Zeichen lang ist. Die Annotation *@NoArgsConstructor* ist eine Lombok-Annotation, die einen Konstruktor ohne Argumente bereitstellt und *@Getter* und *@Setter* sind ebenso Lombok-Annotationen, die die Getter- und Setter-Methoden für die Klassenattribute generieren.

### 11.3.3 Booking

Die abstrakte Klasse "Booking" dient als Vorlage für die verschiedene Arten von Buchungen innerhalb des Systems. Diese Klasse kapselt gemeinsame Eigenschaften und Verhaltensweisen, die von unterschiedlichen Buchungstypen im WABS-System geteilt werden. Durch diese zentrale Abstraktion können Buchungstypen effizienter erstellt und gewartet werden.

Jede Buchung in dieser Klasse hat einen eindeutige Identifier namens "*id*". Das Attribut "*Employee*" repräsentiert eine Mitarbeiterin oder einen Mitarbeiter, der mit der Buchung verknüpft ist und darf nicht leer sein. Das "*date*"-Attribut gibt das Buchungsdatum an, welches entweder das aktuelle Datum oder ein zukünftiges sein muss. "*timeSlot*" definiert den Zeitraum der Buchung. "*createdOn*" und "*updatedOn*" sind Zeitstempel, die den Erstellungs- bzw. den letzten Aktualisierungszeitpunkt der Buchung anzeigen. Dabei dürfen beide nicht in der Zukunft liegen. "*start*" und "*endTime*" geben den Beginn und das Ende der Buchung an Abbildung 142.

```

=
/** Unique identifier for each booking. */
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
private Long id;

/** The employee associated with the booking. ...*/
@NotNull(message = "employee must not be null")
private Employee employee;

/** The date for the booking. ...*/

@NotNull
@FutureOrPresent(message = "The date must not be in the past")
@ManyToOne(fetch = FetchType.EAGER)
private LocalDate date;

/** The time slot for the booking. */
@ManyToOne(fetch = FetchType.EAGER)
private Timeslot timeSlot;

/** The timestamp of when the booking was made. ...*/

@CreatedDate
@Column(updatable = false)
@PastOrPresent(message = "The date created must not be in the future")
protected LocalDateTime createdOn;

/** The timestamp of when the booking was updated last. ...*/

@LastModifiedDate
@Column(insertable = false)
protected LocalDateTime updatedOn;

/** The start time of the booking. */
private LocalTime start;

/** The end time of the booking. */
private LocalTime endTime;

```

Abbildung 142 – Attribute der Booking Klasse

In Anbetracht der unterschiedlichen Anforderungen an Buchungen (Arbeitsplätze sollen nur zu bestimmten Zeiträumen buchbar sein, während andere jederzeit möglich sein sollen) verfügt die Klasse "Booking" über zwei Konstruktoren. Der erste Konstruktor initialisiert die Buchung basierend auf den Start- und Endzeiten ("start" und "endTime"), während der zweite dies anhand eines vorgegebenen Zeitraums ("Timeslot") macht. Diese sind in Abbildung 143 – Konstruktoren der Booking Klasse dargestellt.

Zusätzlich zur Flexibilität bei der Buchungserstellung bietet die Klasse auch eine Methode namens "isValidTime()". Diese Methode gewährleistet, dass die Startzeit stets vor der Endzeit liegt. Es gibt auch andere nützliche Methoden, die das Abrufen und Festlegen von Erstellungs- und

Aktualisierungszeitstempeln ermöglichen. Diese Methoden sind in Abbildung 144 – Method dargestellt.

```
/** Constructor for creating a booking with a specific employee, date, start and end times. */
public Booking(@NotNull Employee employee, @NotNull LocalDate date, LocalTime startTime, LocalTime endTime) {
    this.employee = employee;
    this.date = date;
    this.start = startTime;
    this.endTime = endTime;
}

/** Constructor for creating a booking with a specific employee, date, time slot, and timestamp. ...*/
public Booking(@NotNull Employee employee, @NotNull LocalDate date, Timeslot timeSlot) {
    this.employee = employee;
    this.date = date;
    this.timeSlot = timeSlot;
    this.start = timeSlot.getStartTime();
    this.endTime = timeSlot.getEndTime();
```

Abbildung 143 – Konstruktoren der Booking Klasse

```
/** This method checks if the start time is before the end time. ...*/
@AssertTrue(message = "Start time must be before end time")
public boolean isValidTime() { return start.isBefore(endTime); }

/** Returns the timestamp of when the booking was created. ...*/
public LocalDateTime getCreatedOn() { return this.createdOn; }

/** Sets the timestamp of when the booking was created. ...*/
public void setCreatedOn(LocalDateTime createdOn) { this.createdOn = createdOn; }

/** Returns the timestamp of when the booking was last updated. ...*/
public LocalDateTime getUpdatedOn() { return this.updatedOn; }

/** Sets the timestamp of when the booking was last updated. ...*/
public void setUpdatedOn(LocalDateTime updatedOn) { this.updatedOn = updatedOn; }
```

Abbildung 144 – Methode der Booking Klasse

Als abstrakte Klasse kann "Booking" abstrakte Methoden vorgeben. Jede Unterklasse, die "Booking" erweitert, muss diese Methoden implementieren. Sie kann jedoch auch zusätzliche Methoden hinzufügen, die in der "Booking"-Klasse nicht vorgegeben wurden. Die Klasse bietet Validierungen, Aufzeichnungen von Erstellungs- und Aktualisierungszeitstempeln und Flexibilität beim Erstellen von Buchungen basierend auf Start- und Endzeiten oder vordefinierten Zeiträumen.

#### 11.3.4 PublicHoliday

Ein "PublicHoliday" im Buchungssystem kennzeichnet Tage, an denen keine Buchungen vorgenommen werden dürfen. Obwohl es sich in den meisten Fällen um gesetzliche Feiertage handelt, können auch andere Tage, an denen nicht gearbeitet wird, als "PublicHoliday" definiert werden. Beispiele für solche Tage sind Neujahr, Weihnachten oder andere landesspezifische Feiertage. Jeder "PublicHoliday" verfügt über ein spezifisches Datum, eine Beschreibung des Anlasses und eine Angabe darüber, ob Buchungen an diesem Tag erlaubt sind oder nicht.

Der "PublicHoliday" besitzt verschiedene Attribute. Dazu gehört die "id", welche eine eindeutige Kennung für jeden Feiertag darstellt. Das "date"-Attribut gibt das Datum des Feiertags an, während "description" eine Beschreibung des Feiertags liefert. Das Attribut "isBookingAllowed" gibt an, ob an diesem Feiertag Buchungen erlaubt sind oder nicht. Hinsichtlich der Konstruktoren gibt es den Standardkonstruktor, der keine Argumente benötigt. Es gibt auch einen weiteren Konstruktor, der einen Feiertag mit einem spezifischen Datum, einer Beschreibung und einer Angabe initialisiert, sowie ob Buchungen an diesem Tag erlaubt sind oder nicht. (Abbildung 145 - PublicHoliday Klasse)

```

@Entity
@Getters
@Setters
@Slf4j
public class PublicHoliday {

    /** The unique identifier of the public holidays. */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /** The date of the public holidays. */
    @FutureOrPresent(message = "The date of the public holidays must not be in the past")
    private LocalDate date;

    /** The description of the public holidays. */
    @NotEmpty(message = "The description of the public holidays must not be empty")
    private String description;

    /** Specifies whether booking is allowed on this public holidays or not. */
    private boolean isBookingAllowed;

    /** Default constructor. */
    public PublicHoliday() {
    }

    /**
     * Constructor that sets up the date, description, and booking allowance for the public holidays.
     *
     * @param date      The date of the public holidays.
     * @param description  The description of the public holidays.
     * @param isBookingAllowed Specifies whether booking is allowed on this public holidays or not.
     */
    public PublicHoliday(LocalDate date, String description, boolean isBookingAllowed) {
        this.date = date;
        this.description = description;
        this.isBookingAllowed = isBookingAllowed;
    }
}

```

Abbildung 145 - PublicHoliday Klasse

### 11.3.5 Timeslot

Ein "*Timeslot*" stellt einen Zeitraum innerhalb des Buchungssystems dar. Er definiert einen vorher festgelegten Zeitraum, in dem eine Buchung möglich ist. Die Klasse "*Timeslot*" bietet standardisierte Zeitrahmen, die gängige Buchungsintervalle repräsentieren, wie Vormittag (AM), Nachmittag (PM) und den gesamten Tag (ALL\_DAY). Jeder dieser Zeiträume verfügt über eine Startzeit, eine Endzeit und eine Bezeichnung. Zu den Attributen eines Zeitraums gehören die "id" als eindeutige Kennung, die "start" als Beginn des Zeitraums, die "endTime" als Ende des Zeitraums und der "name", welcher den Timeslot beschreibt. Es gibt zwei Konstruktoren für diese Klasse. Der Konstruktor mit einem Timeslot wird mit einer festgelegten Startzeit, Endzeit und Bezeichnung initialisiert. Zusätzlich bietet die Klasse Methoden wie "getStartTimeAsString()" und "getEndTimeAsString()", die die Start- bzw. Endzeit als String zurückgeben, sowie "getStart()" und "getEnd()", die die tatsächlichen Start- und Endzeiten zurückgeben. (Abbildung 146 - Timeslot Klasse)

```

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
@Slf4j
public class Timeslot {

    /** The unique identifier for the time slot. */
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    /** The start time of the time slot. */
    private LocalTime startTime;

    /** The end time of the time slot. */
    private LocalTime endTime;

    /** The name or description of the time slot. */
    private String name;

    /** Constructor to create a Timeslot. ...*/
    public Timeslot(LocalTime startTime, LocalTime endTime, String name) {
        this.startTime = startTime;
        this.endTime = endTime;
        this.name = name;
    }

    /** Get the start time of the time slot as a string. ...*/
    public String getStartTimeAsString() { return this.startTime.toString(); }

    /** Get the end time of the time slot as a string. ...*/
    public String getEndTimeAsString() { return this.endTime.toString(); }

    /** Get the start time of the time slot. ...*/
    public LocalTime getStart() { return this.startTime; }

    /** Get the end time of the time slot. ...*/
    public LocalTime getEnd() { return this.endTime; }
}

```

Abbildung 146 - Timeslot Klasse

### 11.3.6 DeskBooking

Das **DeskBooking-Objekt** dient zur Darstellung einer spezifischen Buchung für einen Arbeitsplatz. Es baut auf der allgemeinen Buchungsklasse auf und erweitert diese um Attribute und Methoden, die für Arbeitsplatzbuchungen relevant sind. Da das DeskBooking-Objekt von der generischen Booking-Klasse erbt, übernimmt es alle deren Attribute und Methoden und fügt zusätzliche hinzu. Dieser Ansatz modularisiert den Code und erhöht die Wiederverwendbarkeit.

Die Klasse verfügt über zwei Konstruktoren, um Buchungen spezifischer Art zu initialisieren. Ein Konstruktor mit start und end Times und einen anderen mit einem Timeslot.

Dieser Konstruktor (Abbildung 147 - DeskBooking Konstruktor mit start und endTime) ermöglicht es, ein DeskBooking-Objekt mit charakteristischen Start- und Endzeiten zu erstellen.

```
/*
 * Constructor for creating a desk booking with specific employee, desk, date, start and end times, and timestamp.
 * @param employee The employee associated with the booking.
 * @param desk The desk associated with the booking.
 * @param date The date for the booking.
 * @param start The start time for the booking.
 * @param endTime The end time for the booking.
 */
public DeskBooking(Employee employee, Desk desk, LocalDate date, LocalTime start, LocalTime endTime) {
    super(employee,
        date,
        start,
        endTime);
    this.desk = desk;
}
```

Abbildung 147 - DeskBooking Konstruktor mit start und endTime

Durch die Verwendung dieses Konstruktors kann der Benutzer eine Buchung für einen bestimmten Arbeitsplatz an einem spezifischen Datum und innerhalb eines eindeutigen Zeitraums (definiert durch Start- und Endzeit) vornehmen.

Der Konstruktor (Abbildung 148 - DeskBooking Konstruktor mit Timeslot) ermöglicht es, ein DeskBooking-Objekt basierend auf einem vorgegebenen Zeitfenster (Timeslot) zu erstellen.

```
/*
 * Constructor for creating a desk booking with a specific employee, desk, date, and time slot.
 * @param employee The employee associated with the booking.
 * @param desk The desk associated with the booking.
 * @param date The date for the booking.
 * @param timeSlot The time slot for the booking.
 */
public DeskBooking(Employee employee, Desk desk, LocalDate date, Timeslot timeSlot) {
    super(employee,
        date,
        timeSlot);
    this.desk = desk;
```

Abbildung 148 - DeskBooking Konstruktor mit Timeslot

Durch die Verwendung dieses Konstruktors kann der Benutzer eine Buchung für einen bestimmten Arbeitsplatz, an einem bestimmten Datum und innerhalb eines vorgegebenen Zeitfensters vornehmen. Dies ist besonders nützlich, wenn das System feste Zeitfenster für Buchungen vorsieht.

Beide Konstruktoren sorgen dafür, dass alle notwendigen Attribute des DeskBooking-Objekts korrekt initialisiert werden, wenn ein neues Objekt erstellt wird. Dies gewährleistet die Integrität und Konsistenz der Daten innerhalb des Systems.

Das *DeskBooking*-Objekt bietet mehrere Methoden und Funktionen. Die "*equals()*"-Methode wurde überschrieben, um Deskbooking-Objekte basierend auf dem zugeordneten Arbeitsplatz zu vergleichen. Ebenso wurde die "*hashCode()*"-Methode angepasst, um einen Hashcode basierend auf dem Arbeitsplatz zu generieren. Es gibt spezielle Callback-Methoden wie "*onPrePersist()*" und "*onPostPersist()*", die vor bzw. nach dem Persistieren eines DeskBooking-Objekts aufgerufen werden und entsprechende Protokolleinträge erstellen. Ähnliche Callbacks, "*onPreUpdate()*" und "*onPostUpdate()*", werden vor bzw. nach dem Aktualisieren eines DeskBooking-Objekts aufgerufen. Wie in „Booking“, gibt es auch hier zwei Konstruktoren. Diese Methoden und Funktionen sind in Abbildung 149 - Deskbooking Methode dargestellt.

```

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    DeskBooking that = (DeskBooking) o;
    return Objects.equals(desk, that.desk);
}

@Override
public int hashCode() { return Objects.hash(desk); }

@PrePersist
public void onPrePersist() { log.info("Attempting to create a desk booking for desk: {}", desk.getDeskNr()); }

@PostPersist
public void onPostPersist() {
    log.info("Successfully created a desk booking with ID: {} for desk: {}", getId(), desk.getDeskNr());
}

@PreUpdate
protected void onPreUpdate() {
    updatedOn = LocalDateTime.now();
    log.info("Attempting to update a desk booking with ID: {} for desk: {}", getId(), desk.getDeskNr());
}

@PostUpdate
public void onPostUpdate() {
    log.info("Successfully updated a desk booking with ID: {} for desk: {}", getId(), desk.getDeskNr());
}

```

Abbildung 149 - Deskbooking Methode

## 11.4 Repository

Die Repository-Schicht, oft einfach als "Repository" bezeichnet, ist ein entscheidender Bestandteil des "Work-Area-Booking-System" (WABS). Sie dient als Brücke zwischen der Domänenschicht und der Datenzugriffsschicht und ist verantwortlich für den Zugriff auf Datenquellen, um Daten abzurufen und zu speichern. Dieser Abschnitt fokussiert sich auf die Rolle und Bedeutung der Repository-Schicht im Kontext des DeskBooking-Moduls.

Das Repository bietet eine Abstraktion des Datenzugriffs für die Domäne. Es ermöglicht der Domäne, mit Daten zu arbeiten, ohne sich um die Details der Datenpersistenz oder die spezifische Art der Datenquelle kümmern zu müssen. Dies bedeutet, dass die Geschäftslogik in der Domäne sauber von Datenbankdetails oder anderen Datenquellen getrennt bleibt. Dank der Integration von Spring Data JPA und der H2-Datenbank wird dieser Datenzugriff stark vereinfacht und standardisiert.

Die Hauptfunktionen dieses Layers sind, Daten abzufragen basierend auf bestimmten Kriterien oder Bedingungen und Daten persistent zu machen. Durch die Verwendung von Spring Data JPA bietet die Repository-Schicht CRUD-Methoden (Erstellen, Lesen, Aktualisieren, Löschen) zum Speichern, Aktualisieren und Löschen von Daten in der H2-Datenbank. Dazu versteckt es die Komplexität des Datenzugriffs und bietet eine konsistente Schnittstelle für die Domäne.

In Bezug auf das DeskBooking Modul würde die Repository-Schicht spezifische Speicher für Entitäten wie Desk, DeskBooking, PublicHoliday und Timeslot enthalten. Jedes dieser Repositorys bietet Methoden, um mit den zugehörigen Daten zu arbeiten, sei es das Abrufen von verfügbaren Arbeitsplätzen, das Speichern einer neuen Buchung oder das Aktualisieren eines Feiertags.

Die Repository-Schicht ist von entscheidender Bedeutung, da sie eine klare und konsistente Schnittstelle für den Datenzugriff bietet und sicherstellt, dass die Geschäftslogik sauber und unabhängig von Datenbankdetails bleibt. Mit der Integration von JPA und H2 wird die Entwicklung beschleunigt, der Code bleibt sauber und die Systemwartung wird erleichtert.

### 11.4.1 Desk Repository Schicht

Dieser Abschnitt beschreibt die Repository-Schicht für die Desk-Entität im "Work-Area-Booking-System" (WABS).

#### DeskJPARespo Interface:

Durch die Erweiterung der JpaRepository werden für die Desk-Entität automatisch die grundlegenden CRUD-Operationen (Create, Read, Update, Delete) bereitgestellt. Das erleichtert die Handhabung von Datenbankoperationen enorm. Im Repository sind auch manche benutzerdefinierten Methoden wie findAllDesksByPage und findDeskById definiert, die spezielle Datenabfrageanforderungen erfüllen. Die Verwendung der `@Repository-Annotation` kennzeichnet diese Klasse als ein Spring Data JPA-Repository, was die Integration in das Spring Framework erleichtert und zusätzliche Funktionen wie Transaktionsmanagement ermöglicht. (Abbildung 150 – DeskJPARespo)

```
import ...

/**
 * This interface represents the repository for desks in the booking system.
 * @author Sonja Lechner
 * @version 1.0
 * since 2023-05-24
 */
@Repository
public interface DeskJPARespo extends JpaRepository<Desk, Long> {

    /**
     *
     * @param pageable The pageable object
     * @return A page of desks
     */
    @Query("select a from Desk a")
    Page<Desk> findAllDesksByPage(Pageable pageable);

    /**
     * Retrieves a desk by its ID.
     * @param id id of the desk to be found
     * @return desk with the given id
     */
    Desk findDeskById(Long id);
}
```

Abbildung 150 – DeskJPARespo

#### DeskRepo Interface:

Das DeskRepo Interface definiert die erwarteten Operationen für das **Desk**-Repository. Es enthält Methoden für das Hinzufügen, Abrufen, Aktualisieren und Löschen von Arbeitsplätzen sowie für das

Arbeiten mit Ports, die zu einem Arbeitsplatz gehören. Weiters definiert es auch Ausnahmen, die während der Ausführung dieser Operationen auftreten können, wie "ResourceNotFoundException" und "ResourceDeletionFailureException" (Siehe Abbildung: DeskRepo Interface).

```
public interface DeskRepo {

    /** Adds a desk to the repository. ...*/
    Desk addDesk(Desk desk);

    /** Retrieves all desks from the repository. ...*/
    List<Desk> getAllDesks();

    /**      Retrieves a page of desks from the repository. ...*/
    Page<Desk> getAllDesksByPage(Pageable pageable);

    /** Retrieves a desk by its ID. ...*/
    Desk getDeskById(Long id) throws ResourceNotFoundException;

    /** Updates a desk by its ID. ...*/
    Desk updateDeskById(Long id, Desk desk) throws ResourceNotFoundException;

    /** Updates a desk. ...*/
    Desk updateDesk(Desk desk) throws ResourceNotFoundException;

    /** Deletes a desk by its ID. ...*/
    List<Desk> deleteDeskById(Long id) throws ResourceDeletionFailureException;

    /** Creates a port for a desk. ...*/
    Desk createPort(Long deskId, Port newPort);

    /** Updates a port for a desk. ...*/
    Desk updatePort(Long deskId, String portName, Port updatedPort);

    /**...*/
    Desk deletePort(Long deskId, String portName);

    /** Retrieves the ports of a desk. ...*/
    List<Port> getPorts(Long deskId);
}
```

Abbildung 151 - DeskRepo Interface

### DeskRepo\_JPAH2 Klasse:

Diese Klasse dient als konkrete Implementierung des "DeskRepo" Interfaces und nutzt dabei das "DeskJPARrepo", sowie das "DeskBookingJPARrepo", um Datenzugriffsoperationen auszuführen. Mit der @Component-Annotation wird sie als Spring-Komponente gekennzeichnet, was bedeutet, dass sie vom Spring Container instanziert und verwaltet wird. Ein integrierter Logger macht es möglich, sowohl wichtige Informationen als auch auftretende Fehler zu protokollieren. Die in dieser Klasse definierten Methoden sind für die Ausführung der eigentlichen Datenzugriffsoperationen verantwortlich. Sie behandeln dabei mögliche Ausnahmen und sorgen dafür, dass die erwarteten Ergebnisse korrekt

zurückgegeben werden. Siehe **Fehler! Verweisquelle konnte nicht** Verantwortlich für den Inhalt: Sonja Lechner  
153 - DeskJPARespo – Teil .

```
>     /** Updates a desk by its ID. ...*/
>     @Override
>     public Desk updateDeskById(Long id, Desk updatedDesk) throws ResourceNotFoundException {
>         Optional<Desk> deskOptional = this.deskJPARespo.findById(id);
>         if (deskOptional.isPresent()) {
>             Desk desk = deskOptional.get();
>             desk.setDeskNr(updatedDesk.getDeskNr());
>             desk.setNrOfMonitors(updatedDesk.getNrOfMonitors());
>             desk.setPorts(updatedDesk.getPorts());
>             return this.deskJPARespo.save(desk);
>         } else {
>             throw new ResourceNotFoundException("The desk with the ID: " + id + " was not found!");
>         }
>     }

>     /** Updates a desk. ...*/
>     public Desk updateDesk(Desk updatedDesk) throws ResourceNotFoundException {
>         try {
>             Optional<Desk> desk = this.deskJPARespo.findById(updatedDesk.getId());
>             if (desk.isPresent()) {
>                 Desk deskToUpdate = desk.get();
>                 deskToUpdate.setDeskNr(updatedDesk.getDeskNr());
>                 deskToUpdate.setNrOfMonitors(updatedDesk.getNrOfMonitors());
>                 deskToUpdate.deleteAllPorts();
>                 for (Port p : updatedDesk.getPorts()) {
>                     deskToUpdate.addPort(new Port(p.getName()));
>                 }
>                 return this.deskJPARespo.save(deskToUpdate);
>             } else {
>                 throw new ResourceNotFoundException("desk with ID " + updatedDesk.getId() + " not found!");
>             }
>
>         } catch (Exception e) {
>             throw new ResourceNotFoundException("desk not found");
>         }
>     }

>     /** Deletes a desk by its ID. ...*/
>     @Override
>     public List<Desk> deleteDeskById(Long id) throws ResourceDeletionFailureException {
>         List<DeskBooking> bookings = this.deskBookingJPARespo.findBookingsByDeskId(id);
>         if (!bookings.isEmpty()) {
>             throw new ResourceDeletionFailureException("desk with the ID: " + id + " cannot be deleted as it is part of a booking.");
>         }

>         Optional<Desk> deskOptional = this.deskJPARespo.findById(id);
>         if (deskOptional.isPresent()) {
>             this.deskJPARespo.deleteById(id);
>             return this.deskJPARespo.findAll();
>         } else {
>             throw new ResourceDeletionFailureException("Deletion of the desk with the ID: " + id + " was not possible!");
>         }
>     }
```

Abbildung 152 - DeskJPARespo – Teil 1

```

/** Creates a port for a desk. ...*/
@Override
public Desk createPort(Long deskId, Port newPort) {
    Desk desk = deskJPARepo.findById(deskId).orElse( other: null);
    if (desk != null) {
        List<Port> ports = desk.getPorts();
        ports.add(newPort);
        desk.setPorts(ports);
        return deskJPARepo.save(desk);
    }
    return null;
}

/** Updates a port for a desk. ...*/
@Override
public Desk updatePort(Long deskId, String portName, Port updatedPort) {
    Desk desk = deskJPARepo.findById(deskId).orElse( other: null);
    if (desk != null) {
        List<Port> ports = desk.getPorts();
        for (Port port : ports) {
            if (port.getName().equals(portName)) {
                port.setName(updatedPort.getName());
                break;
            }
        }
        desk.setPorts(ports);
        return deskJPARepo.save(desk);
    }
    return null;
}

/** Deletes a port from a desk. ...*/
@Override
public Desk deletePort(Long deskId, String portName) {
    Desk desk = deskJPARepo.findById(deskId).orElse( other: null);
    if (desk != null) {
        List<Port> ports = desk.getPorts();
        ports.removeIf(port -> port.getName().equals(portName));
        desk.setPorts(ports);
        return deskJPARepo.save(desk);
    }
    return null;
}

/** Retrieves the ports of a desk. ...*/
@Override
public List<Port> getPorts(Long deskId) {
    Desk desk = deskJPARepo.findById(deskId).orElse( other: null);
    if (desk != null) {
        return desk.getPorts();
    }
    return null;
}

```

Abbildung 153 - DeskJPAREpo – Teil 2

Ein wichtiger Aspekt dieser Architektur ist die Flexibilität und Skalierbarkeit. Während das System aktuell die H2-Datenbank nutzt, ermöglicht die klare Trennung und Abstraktion der Repository-Schicht

einen nahtlosen Übergang zu anderen Datenbanktechnologien in der Zukunft. Beispielsweise könnte, falls erforderlich, die H2-Datenbank leicht durch eine leistungsstärkere Datenbank wie MariaDB oder MySQL ersetzt werden. Dies wäre ohne größere Änderungen an der Geschäftslogik oder anderen Teilen des Systems möglich, da die spezifischen Datenbankoperationen innerhalb der Repository-Schicht gekapselt sind. Dies stellt sicher, dass das DeskBooking-Modul zukunftssicher und leicht an veränderte Anforderungen anpassbar ist.

#### 11.4.2 DeskBooking Repository Schicht

Im DeskBooking-Modul spiegelt sich die Struktur der Repository-Schicht wider. Es besteht aus drei zentralen Komponenten: dem "*DeskBookingRepo*", dem "*DeskBookingJPARepo*" und dem "*DeskBooking\_JPAH2Repo*". Jede dieser Komponenten hat eine spezifische Rolle im Datenzugriffsprozess.

Das *DeskBookingRepo* dient als Hauptinterface und legt die Methoden fest, die für den Datenzugriff auf DeskBooking-Entitäten erforderlich sind. Das *DeskBookingJPARepo*, das auf Spring Data JPA aufbaut, implementiert diese Methoden und bietet so eine direkte Verbindung zur Datenbank. Schließlich ist das *DeskBooking\_JPAH2Repo* speziell für die Interaktion mit der H2-Datenbank optimiert.

Durch diese Aufteilung wird sichergestellt, dass das DeskBooking-Modul effizient mit Daten arbeiten kann, während es gleichzeitig flexibel genug bleibt, um zukünftige Änderungen oder Erweiterungen zu unterstützen. Es ist ein Paradebeispiel dafür, wie die Repository-Schicht im gesamten WABS-System eingesetzt wird, um den Datenzugriff zu optimieren und gleichzeitig die Integrität und Klarheit des Codes zu gewährleisten.

##### **DeskBookingRepo**

Das *DeskBookingRepo-Interface* dient als Schnittstelle für den Datenzugriff auf Arbeitsplatzbuchungen. Es legt die notwendigen Methoden fest, um mit Buchungsdaten zu interagieren, ohne, dass sich die Anwender oder Entwickler direkt mit den Datenbankdetails auseinandersetzen müssen.

Innerhalb dieses Interfaces sind Methoden wie "*addBooking()*" vorgesehen, die es ermöglichen, eine neue Buchung zum System hinzuzufügen. Mit "*getAllBookings()*" können alle vorhandenen Buchungen abgerufen werden, während "*searchBookings()*" eine flexiblere Alternative bietet, spezifische Buchungen anhand verschiedener Kriterien zu erhalten.

Für Anforderungen bestimmter Datenfelder, gibt es Methoden wie "*getBookingByBookingId()*", die eine Buchung anhand ihrer eindeutigen ID abruft. Dies ist besonders nützlich, wenn man Details zu einer bestimmten Buchung benötigt. Sollte eine Buchung aktualisiert oder gelöscht werden müssen, stehen die Methoden "*updateBookingById()*" und "*deleteBookingById()*" zur Verfügung.

Die Methode "*getAvailableDesks()*" ist besonders nützlich, um zu überprüfen, welche Arbeitsplätze zu einem bestimmten Zeitpunkt verfügbar sind und stellt sicher, dass Doppelbuchungen vermieden werden.

## DeskBookingJPARepo

Das „*DeskBookingJPARepo-Interface*“ stellt eine spezialisierte Schnittstelle für den Datenzugriff auf Arbeitsplatzbuchungen im Kontext von Spring Data JPA dar. Durch die Erweiterung des JpaRepository-Interfaces profitiert es von einer Vielzahl von vorgefertigten Methoden für gängige Datenbankoperationen, welche die Entwicklung erheblich beschleunigt und den Code sauber hält.

Durch das Interface können Entwickler direkt auf spezifische Buchungsinformationen zugreifen, ohne komplexe Datenbankabfragen schreiben zu müssen. Zum Beispiel ermöglicht *"findBookingsById()"* das Abrufen einer Buchung anhand ihrer eindeutigen ID, während *"findBookingsByDeskId()"* alle Buchungen für einen bestimmten Arbeitsplatz zurückgibt.

Die Methode *"findBookingsByDate()"* ist besonders nützlich, wenn man alle Buchungen an einem bestimmten Datum sehen möchte. Wenn man hingegen alle Buchungen einer bestimmten Mitarbeiterin / eines bestimmten Mitarbeiter abrufen möchte, kann man *"findBookingsByEmployeeId()"* verwenden.

Ein weiteres komfortables Feature ist *"findByDateAndStartBetween()"*. Diese Methode ermöglicht es, Buchungen basierend auf einem spezifischen Datum und einem Zeitraum zu filtern, was besonders nützlich ist, um die Verfügbarkeit von Arbeitsplätzen in einem bestimmten Zeitfenster zu überprüfen.

## DeskBookingRepo\_JPAH2

Dank der Integration von JPA in diese Implementierung kann das DeskBookingRepo\_JPAH2 Datenbanktransaktionen auf einer abstrakten Ebene durchführen. Dies bedeutet, dass Entwickler nicht direkt mit SQL-Abfragen arbeiten müssen, sondern stattdessen objektorientierte Methoden verwenden können, um mit Datenbankdaten zu interagieren.

Beispielsweise ermöglicht die Methode *"addBooking()"* das Hinzufügen einer neuen Buchung zur Datenbank. Intern nutzt sie die *"save()"*-Methode von JPA, um das DeskBooking-Objekt zu persistieren. Ähnlich verwendet *"getBookingByBookingId()"* die *"findById()"* -Methode von JPA, um eine Buchung basierend auf ihrer eindeutigen ID abzurufen.

Die Methode *"getAvailableDesks()"* könnte eine der komplexeren Methoden in dieser Abschnitt sein. Sie gibt verfügbare Arbeitsplätze zurück, die anhand bestimmter Parameter abgefragt werden.

## Einige wichtige Methode erklärt

**addBooking():** Die Methode "addBooking()" stellt sicher, dass alle erforderlichen Daten vorhanden sind und, dass der gewünschte Arbeitsplatz zum gewünschten Zeitpunkt verfügbar ist. Sie übernimmt die Validierung der Eingabe, das Abrufen des zugehörigen Arbeitsplatzes und die Überprüfung der Verfügbarkeit des Arbeitsplatzes. Nach erfolgreicher Validierung wird die Buchung in der Datenbank gespeichert, wie in Abbildung 154 - DeskBookingRepo\_JPAH2 addBooking Methode abgebildet.

```
/*
 * Adds a desk booking to the repository.
 *
 * @param deskBooking The desk booking to add.
 * @return The added desk booking.
 * @throws DeskNotAvailableException If the desk is not available for the booking period or if the employee already has a booking on the specified date.
 * @throws ResourceNotFoundException If the desk or employee is not found.
 * @throws IllegalArgumentException If any of the required fields are null.
 */
@Override
public DeskBooking addBooking(DeskBooking deskBooking) throws DeskNotAvailableException, ResourceNotFoundException {
    // Validate input
    if (deskBooking.getEmployee() == null || deskBooking.getDesk() == null || deskBooking.getStart() == null || deskBooking.getEndTime() == null) {
        throw new IllegalArgumentException("Please fill in all fields");
    }

    // Load the associated desk entity from the database
    Desk desk = deskJPARepo.findDeskById(deskBooking.getDesk().getId());
    if (desk == null) {
        throw new ResourceNotFoundException("Desk with ID " + deskBooking.getDesk().getId() + " was not found");
    }

    // Check if the desk is available for the booking period
    List<Desk> availableDesks = getAvailableDesks(deskBooking.getDate(), deskBooking.getStart(), deskBooking.getEndTime(), Optional.of(desk.getId()));
    if (availableDesks.isEmpty()) {
        throw new DeskNotAvailableException("Desk with ID " + desk.getId() + " is not available for this period");
    }

    // Check if the employee already has a booking on this date
    List<DeskBooking> existingBookings = deskBookingJPARepo.findBookingsByEmployeeIdAndDate(deskBooking.getEmployee().getId(), deskBooking.getDate());
    if (!existingBookings.isEmpty()) {
        throw new DeskNotAvailableException(deskBooking.getEmployee().getFname() + " " + deskBooking.getEmployee().getLname() + " already has a booking on this date!");
    }

    // Save the booking
    try {
        return this.deskBookingJPARepo.save(deskBooking);
    } catch (Exception e) {
        throw new RuntimeException("Error saving the booking to the database", e);
    }
}
```

Abbildung 154 - DeskBookingRepo\_JPAH2 addBooking Methode

Die addBooking Methode stellt sicher, dass jedes erforderliche Feld vorhanden und verfügbar ist, der gewünschte Arbeitsplatz existiert sowie, dass die Mitarbeiterin oder der Mitarbeiter an diesem Datum keine anderen Buchungen hat, bevor sie oder er die Buchung in der Datenbank speichert.

**getAllBookings():** Die getAllBookings Methode gibt eine Liste aller zukünftigen Arbeitsplatzbuchungen zurück.

```
/**  
 * Retrieves all desk bookings from the repository.  
 *  
 * @return A list of all desk bookings in the database.  
 * @throws ResourceNotFoundException If no desk bookings are found.  
 */  
  
@Override  
public List<DeskBooking> getAllBookings() throws ResourceNotFoundException {  
    // Fetch all bookings from the database  
    List<DeskBooking> bookings = deskBookingJPARepo.findAll();  
  
    //check if the list is empty  
    if (bookings.isEmpty()) {  
        throw new ResourceNotFoundException("No desk bookings found.");  
    }  
    // Filter out bookings that are in the past  
    return bookings.stream()  
        .filter(booking -> !booking.getDate().isBefore(LocalDate.now()))  
        .collect(Collectors.toList());  
}
```

Abbildung 155 - DeskBookingRepo\_JPAH2 getAllBookings Methode

**searchBookings():** Mit der searchBookings Methode können Benutzer Arbeitsplatzbuchungen basierend auf verschiedenen Kriterien suchen. Sie bietet Flexibilität bei der Suche und gibt nur die relevanten Buchungen basierend auf den angegebenen Kriterien zurück. (Abbildung 156 - DeskBookingRepo\_JPAH2 searchBookings Methode)

```

/**
 * Searches for desk bookings based on the given parameters.
 *
 * @param employeeId The ID of the employee.
 * @param deskId      The ID of the desk.
 * @param date        The date.
 * @return The list of desk bookings that match the search criteria.
 */
public List<DeskBooking> searchBookings(Long employeeId, Long deskId, LocalDate date){

    // If the date is in the past, return an empty list
    if (date != null && date.isBefore(LocalDate.now())) {
        return new ArrayList<>();
    }

    // If all parameters are null, return all bookings
    if (employeeId == null && deskId == null && date == null) {
        return deskBookingJPARepo.findAll();
    }

    // If only employee ID and date are provided, search by employee ID and date
    if (employeeId != null && date != null) {
        return deskBookingJPARepo.findBookingsByEmployeeIdAndDate(employeeId, date);
    }

    // If only desk ID and date are provided, search by desk ID and date
    if (deskId != null && date != null) {
        return deskBookingJPARepo.findBookingsByDeskIdAndDate(deskId, date);
    }

    // If only employee ID is provided, search by employee ID
    if (employeeId != null) {
        return deskBookingJPARepo.findBookingsByEmployeeId(employeeId);
    }

    // If only desk ID is provided, search by desk ID
    if (deskId != null) {
        return deskBookingJPARepo.findBookingsByDeskId(deskId);
    }

    return new ArrayList<>();
}

```

---

Abbildung 156 - DeskBookingRepo\_JPAH2 searchBookings Methode

**getBookingByBookingId():** Die Methode „*getBookingByBookingId()*“ ermöglicht es Benutzern, eine spezifische Arbeitsplatzbuchung basierend auf ihrer eindeutigen ID abzurufen. Sie stellt sicher, dass der Benutzer informiert wird, ob die Buchung existiert oder nicht (*Abbildung 157 - DeskbookingRepo\_JPAH2 getBookingByBookingId Methode*).

```
/*
 * Retrieves a desk booking based on the provided booking ID.
 *
 * @param bookingId The unique identifier of the desired desk booking.
 * @return An Optional containing the desk booking if found, or an empty Optional otherwise.
 * @throws ResourceNotFoundException If no desk booking is found for the provided booking ID.
 */
@Override
public Optional<DeskBooking> getBookingByBookingId(Long bookingId) throws ResourceNotFoundException {
    // Fetch all bookings from the database with the Desk Booking Id provided
    Optional<DeskBooking> bookingOptional = deskBookingJPARepo.findBookingsById(bookingId);

    //check if the list is empty
    if (bookingOptional.isEmpty()) {
        String message = "Desk booking with the ID: " + bookingId + " was not found!";
        log.error(message);
        throw new ResourceNotFoundException(message);
    }
    return bookingOptional;
}
```

Abbildung 157 - DeskbookingRepo\_JPAH2 getBookingByBookingId Methode

**updateBookingById():** Die "updateBookingById"-Methode ermöglicht es Benutzern, eine bestehende Buchung zu aktualisieren. Sie stellt sicher, dass die aktualisierten Daten korrekt sind und dass die Buchung existiert, bevor sie aktualisiert wird. (Abbildung 159 Abbildung 165 - DeskbookingServiceImplementation - updateBookingById Method).

```

/**
 * Updates a desk booking by its ID.
 *
 * @param deskBookingId      The ID of the desk booking to update.
 * @param updatedDeskBooking The updated desk booking details.
 * @return The updated desk booking.
 * @throws ResourceNotFoundException If the booking or associated entities are not found.
 */
public DeskBooking updateBookingById(Long deskBookingId, DeskBooking updatedDeskBooking) throws ResourceNotFoundException {
    // Validate the provided updated booking details
    validateUpdatedBooking(updatedDeskBooking);

    // Fetch the existing booking using the provided ID
    return deskBookingJPARepo.findById(deskBookingId).map(existingBooking -> {
        // Fetch the associated desk and employee details for the updated booking
        Desk fetchedDesk;
        try {
            fetchedDesk = fetchDesk(updatedDeskBooking.getDesk().getId());
        } catch (ResourceNotFoundException e) {
            throw new RuntimeException(e);
        }
        Employee fetchedEmployee;
        try {
            fetchedEmployee = fetchEmployee(updatedDeskBooking.getEmployee().getId());
        } catch (ResourceNotFoundException e) {
            throw new RuntimeException(e);
        }

        // Update the existing booking with the new details
        existingBooking.setDesk(fetchedDesk);
        existingBooking.setEmployee(fetchedEmployee);
        existingBooking.setDate(updatedDeskBooking.getDate());
        existingBooking.setStart(updatedDeskBooking.getStart());
        existingBooking.setEndTime(updatedDeskBooking.getEndTime());
        existingBooking.setUpdatedOn(LocalDateTime.now());
        return existingBooking;
    }).orElseThrow(() -> new ResourceNotFoundException("The desk booking with the ID: " + deskBookingId + " was not found!"));
}

```

Abbildung 158 - DeskbookingRepo\_JPAH2 updateBookingById Methode

**deleteBookingById():** Mit der „*deleteBookingById*“-Methode können Benutzer eine Buchung löschen. Sie stellt sicher, dass die Buchung existiert, bevor sie gelöscht wird und informiert den Benutzer über den Status des Löschens (Siehe Abbildung 159 - DeskbookingRepo\_JPAH2 deleteBookingById Methode).

```

/**
 * Deletes a desk booking from the repository using its ID.
 *
 * @param bookingId The unique identifier of the desk booking to be deleted.
 * @throws ResourceDeletionFailureException If the specified booking is not found in the repository.
 */
public void deleteBookingById(Long bookingId) throws ResourceDeletionFailureException {
    // Retrieve the booking using the provided ID
    Optional<DeskBooking> bookingOptional = this.deskBookingJPARepo.findById(bookingId);

    // Check if the booking exists in the repository
    if (bookingOptional.isPresent()) {
        // Delete the booking from the repository
        this.deskBookingJPARepo.deleteById(bookingId);
    } else {
        // Throw an exception if the booking is not found
        throw new ResourceDeletionFailureException("The desk booking with the ID: " + bookingId + " was not found!");
    }
}

```

Abbildung 159 - DeskbookingRepo\_JPAH2 deleteBookingById Methode

**getAvailableDesks():** Die "getAvailableDesks" Methode gibt eine Liste von Arbeitsplätzen zurück, die zu einem bestimmten Zeitpunkt verfügbar sind. Sie berücksichtigt bestehende Buchungen und stellt sicher, dass sich überschneidende Buchungen vermieden werden. (Abbildung 166 - DeskbookingServiceImplementation - getAvailableDesks Method)

```

/*
 * Retrieves available desks for a specific date and time range.
 * This method checks the availability of each desk for the given date and time range
 * and returns a list of desks that are not booked during that period.
 *
 * @param date The date to check for availability.
 * @param start The start time of the time range.
 * @param end The end time of the time range.
 * @param specificDeskId (Optional) The specific desk ID to check availability for.
 * @return A list of available desks for the specified date and time range.
 *         If specificDeskId is provided, the list will contain either the specific desk (if available) or be empty.
 */
public List<Desk> getAvailableDesks(LocalDate date, LocalTime start, LocalTime end, Optional<Long> specificDeskId) {
    List<Desk> allDesks;

    // Directly query for the specific desk if ID is provided
    if (specificDeskId.isPresent()) {
        Desk specificDesk = deskJPARRepo.findById(specificDeskId.get()).orElse( null );
        allDesks = (specificDesk != null) ? Collections.singletonList(specificDesk) : Collections.emptyList();
    } else {
        allDesks = deskJPARRepo.findAll();
    }

    // Ensure allDesks is not null
    if (allDesks == null) {
        allDesks = new ArrayList<>();
    }

    List<Desk> availableDesks = new ArrayList<>();

    try {
        for (Desk desk : allDesks) {
            Long deskId = desk.getId();
            List<DeskBooking> overlappingBookings = deskBookingJPARRepo.findBookingsByDeskIdDateAndTimeRange(deskId, date, start, end);

            // Ensure overlappingBookings is not null
            if (overlappingBookings == null || overlappingBookings.isEmpty()) {
                availableDesks.add(desk);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return availableDesks;
}

```

Abbildung 160 - DeskbookingRepo\_JPAH2 getAvailableDesks Methode

### 11.4.3 Andere Komponenten der Repository Schicht

Innerhalb der Deskbooking Modul existieren weitere Hilfskomponenten wie PublicHoliday und Timeslot, die in der Repository-Schicht vorkommen. Beide Komponenten sind ähnlich aufgebaut und verfügen über vergleichbare Strukturen und Funktionen. Sie nutzen jeweils ein zentrales Interface, welches die Hauptmethoden für den Datenzugriff definiert. Ergänzt wird dieses durch ein weiteres Interface, das von Spring Data JPA bereitgestellt wird und Standard-CRUD-Operationen ermöglicht. Eine konkrete Implementierungsklasse stellt schließlich die tatsächlichen Datenbankoperationen bereit. Trotz ihrer spezifischen Anwendungsgebiete weisen PublicHoliday und Timeslot in ihrer Architektur und Funktionalität viele Parallelen auf.

## 11.5 Service

Der Service Layer, oft auch als Business-Logik-Schicht bezeichnet, ist eine Schlüsselkomponente in Schichtenarchitekturen. Er befindet sich zwischen der Präsentationsschicht (UI) und der Datenzugriffsschicht (Repositories, Datenbanken) und kapselt die Kerngeschäftslogik einer Anwendung ein. Hier befindet sich die Hauptfunktionalität der Anwendung. Der Service Layer transformiert Daten zwischen der Präsentationsschicht und der Datenzugriffsschicht, was die Umwandlung von Datenformaten, die Validierung von Daten oder die Aggregation von Daten beinhalten kann. Er verwaltet Transaktionen und stellt die Datenkonsistenz sicher. Bei einem Fehlschlag einer Operation sorgt der Service Layer dafür, dass keine inkonsistenten Daten gespeichert werden können. Insgesamt stellt der Service Layer in der WABS-Anwendung sicher, dass die Geschäftsregeln und -logik rund um das Arbeitsplatzbuchungssystem korrekt implementiert und durchgesetzt werden.

Innerhalb des Service Layers befindet sich die Klasse DeskServiceImplementation, die das DeskService-Interface implementiert. Sie stellt die tatsächliche Implementierung der Methoden bereit. Durch Dependency Injection kann auf das DeskRepo-Interface zugegriffen werden, was den Zugriff auf die Datenbankoperationen ermöglicht. Die Klasse bietet Funktionen wie das Hinzufügen eines neuen Arbeitsplatzes, das Abrufen der Gesamtzahl der Arbeitsplätze, das Abrufen einer Liste aller Arbeitsplätze und viele andere Funktionen, die für die Verwaltung von Arbeitsplätzen erforderlich sind.

Ebenso gibt es die BookingServiceImplementation, die das BookingService-Interface implementiert und konkrete Funktionalitäten für die Verwaltung der Buchungen bietet. Sie interagiert mit dem BookingRepo, um die gewünschten Aktionen zu realisieren, wie das Hinzufügen einer neuen Buchung, das Abrufen aller vorhandenen Buchungen oder das Aktualisieren einer Buchung basierend auf ihrer ID.

Die TimeslotServiceImplementation ist für die Verwaltung von Zeitfenstern verantwortlich. Sie implementiert das TimeslotService-Interface und bietet Funktionen wie das Hinzufügen eines neuen Zeitraums oder das Abrufen aller verfügbaren Zeiträume.

### 11.5.1 DeskServiceImplementation (Klasse)

Die DeskServiceImplementation implementiert das DeskService-Interface und dient als Vermittler zwischen der Präsentationsschicht und der Datenzugriffsschicht. Dank der Dependency Injection wird eine Instanzvariable namens deskRepo bereitgestellt, die eine Referenz auf das DeskRepo-Interface enthält. Dies ermöglicht der Klasse, direkt mit der Datenbank zu interagieren und die erforderlichen Datenbankoperationen durchzuführen.

Innerhalb der DeskServiceImplementation gibt es verschiedene Methoden, die spezifische Funktionen bieten. Die Methode „addDesk()“ ermöglicht das Hinzufügen eines neuen Arbeitsplatzes zur Datenbank. Nachdem der Arbeitsplatz erfolgreich hinzugefügt wurde, gibt die Methode den neu erstellten Arbeitsplatz zurück. Mit „getTotalDesks()“ kann die Gesamtzahl der in der Datenbank gespeicherten Arbeitsplätze abgerufen werden. Die Methode „getAllDesks()“ bietet eine einfache Möglichkeit, eine Liste aller Arbeitsplätze in der Datenbank abzurufen. Für Anwendungen, die eine Paginierung erfordern, gibt es die Methode „getAllDesksByPage()“. Sie gibt eine Seite von Arbeitsplätzen zurück, wobei die Anzahl der zurückgegebenen Einträge durch das Pageable-Objekt gesteuert wird. Mit „getDeskById()“ kann ein spezifischer Arbeitsplatz basierend auf seiner eindeutigen ID abgerufen werden. Die Methode „updateDeskById()“ bietet die Möglichkeit, die Details eines bestimmten Arbeitsplatzes zu aktualisieren. Wenn ein Arbeitsplatz nicht mehr benötigt wird oder aus anderen Gründen gelöscht werden soll, kann die Methode „deleteDeskById()“ verwendet werden. Zusätzlich zu den oben genannten Methoden gibt es auch Funktionen wie „createPort()“, „updatePort()“, „deletePort()“ und „getPorts()“, die speziell für die Verwaltung der Schnittstellen eines Arbeitsplatzes entwickelt wurden.

### 11.5.2 DeskbookingServiceImplementation (Klasse)

Die DeskServiceImplementation-Klasse im Deskbooking-Modul der WABS-Anwendung verknüpft Benutzeroberfläche und Datenbank durch die Umsetzung des DeskBookingService-Interfaces. Sie interagiert sowohl mit dem DeskBookingRepo für Buchungsoperationen als auch mit dem HolidayRepo, um Feiertagsbeschränkungen zu berücksichtigen. Ihre robuste Fehlerbehandlung liefert klare Rückmeldungen bei Konflikten und sie gewährleistet konsistente und effiziente Datenbankinteraktionen, wobei sie die Businesslogik des Buchungssystems beibehält.

#### Implementierung der Methoden

Die Methode "addDeskBooking()" in der DeskServiceImplementation-Klasse dient dem Hinzufügen neuer Arbeitsplatzbuchungen. Bei Aufruf dieser Methode wird werden verschiedene Überprüfungen, wie zum Beispiel Prüfung der Verfügbarkeit des gewünschten Arbeitsplatzes und die Validierung des gewählten Datums und der Uhrzeit, durchgeführt. Nach erfolgreicher Überprüfung wird die neu erstellte Buchung zur Datenbank hinzugefügt und diese als Rückgabewert geliefert (*Abbildung 161 - DeskbookingServiceImplementation - addDeskbooking Method*).

```
/** Creates a new desk booking. ...*/
@Override
public DeskBooking addDeskBooking(DeskBooking booking) throws DeskNotAvailableException, ResourceNotFoundException, IllegalArgumentException {
    LocalDate bookingDate = booking.getDate();
    LocalDate currentDate = LocalDate.now();
    Role role = booking.getEmployee().getRole();

    // Check if desk is available for the date and time chosen
    List<DeskBooking> bookings = deskBookingRepo.getBookingsByDeskAndDateAndBookingTimeRange(booking.getId(), bookingDate, booking.getStart(), booking.getEndTime());
    if (!bookings.isEmpty()) {
        throw new DeskNotAvailableException("Desk not available for booking period");
    }

    // Check if booking is for a past date
    if (bookingDate.isBefore(currentDate)) {
        throw new IllegalArgumentException("Cannot create booking for a past date");
    }

    // Check if booking is for a weekday
    DayOfWeek dayOfWeek = bookingDate.getDayOfWeek();
    if (dayOfWeek == DayOfWeek.SATURDAY || dayOfWeek == DayOfWeek.SUNDAY) {
        throw new IllegalArgumentException("Cannot create booking for a weekend");
    }

    // Check if booking is allowed on this day i.e. not a non-bookable holiday
    if (!holidayRepo.isBookingAllowedOnHoliday(bookingDate)) {
        throw new IllegalArgumentException("No work-area bookings are on this public holiday!");
    }

    // Determine maximum advance booking date based on user role
    LocalDate maxAdvanceBookingDate = (role == Role.ROLE_N_EMPLOYEE) ? currentDate.plusWeeks( weeksToAdd: 1 ) : currentDate.plusWeeks( weeksToAdd: 12 );

    // Check if booking is too far in advance
    if (bookingDate.isAfter(maxAdvanceBookingDate)) {
        throw new IllegalArgumentException("Cannot book more than " +
            (role == Role.ROLE_N_EMPLOYEE ? "1 week" : "12 weeks") + " in advance");
    }
    return this.deskBookingRepo.addBooking(booking);
}
```

Abbildung 161 - DeskbookingServiceImplementation - addDeskbooking Method

Die **getAllBookings** Methode gibt eine Liste aller Arbeitsplatzbuchungen zurück, wie in Abbildung 162 abgebildet.

```
/** Retrieves all desk bookings from the database. ...*/
@Override
public List<DeskBooking> getAllBookings() throws ResourceNotFoundException {
    // Fetch all bookings from the repository
    return deskBookingRepo.getAllBookings();
}
```

Abbildung 162 - DeskbookingServiceImplementation - getAllBookings Method

Die "searchBookings"-Methode in der DeskServiceImplementation ermöglicht eine flexible Suche nach Arbeitsplatzbuchungen. Sie kann Buchungen anhand von Kriterien wie Mitarbeiter und Datum finden. Zusätzlich gibt es eine erweiterte Version dieser Methode, die auch die Möglichkeit bietet, Buchungen anhand der Arbeitsplatz-ID zu durchsuchen. Dies bietet den Benutzern eine vielseitige Möglichkeit, spezifische Buchungsinformationen im System abzurufen (Abbildung 163 - DeskbookingServiceImplementation - searchbookings Method).

```

/** Searches for desk bookings based on provided criteria. ...*/
@Override
public List<DeskBooking> searchBookings(Long employeeId, Long deskId, LocalDate date) throws ResourceNotFoundException {
    // If all parameters are provided, search by employeeId, deskId, and date
    if (employeeId != null && deskId != null && date != null) {
        return deskBookingRepo.getBookingsByEmployeeIdAndDeskIdAndDate(employeeId, deskId, date);
    }

    // If only employeeId is provided, search by employeeId
    if (employeeId != null) {
        return deskBookingRepo.getBookingsByEmployee(employeeId);
    }

    // If only deskId is provided, search by deskId
    if (deskId != null) {
        return deskBookingRepo.getBookingByDesk(deskId);
    }

    // If only date is provided, search by date
    if (date != null) {
        return deskBookingRepo.getBookingByDate(date);
    }

    // If none of the parameters are provided, return an empty list
    return new ArrayList<>();
}

```

Abbildung 163 - DeskbookingServiceImplementation - searchbookings Method

Die "getBookingById"-Methode in der DeskServiceImplementation dient dazu, eine spezifische Arbeitsplatzbuchung im System zu lokalisieren und abzurufen. Durch die Eingabe einer Buchungs-ID kann der Benutzer genaue Informationen zu einer bestimmten Buchung erhalten, was die Effizienz und Benutzerfreundlichkeit des Systems erhöhten (*Abbildung 164 - DeskbookingServiceImplementation - getBookingById Method*).

```

/** Retrieves a desk booking by its ID. ...*/
@Override
public Optional<DeskBooking> getBookingById(Long bookingId) throws ResourceNotFoundException {
    // Fetch the booking from the repository using the provided ID
    return Optional.ofNullable(deskBookingRepo.getBookingByBookingId(bookingId))
        .orElseThrow(() -> new ResourceNotFoundException("Booking with ID " + bookingId + " not found."));
}

```

Abbildung 164 - DeskbookingServiceImplementation - getBookingById Method

Die **updateBookingById**-Methode in der DeskServiceImplementation ermöglicht es Benutzern, bestehende Arbeitsplatzbuchungen zu aktualisieren. Durch die Angabe der ID der Buchung können spezifische Details wie Datum, Zeit oder zugeordneter Arbeitsplatz geändert werden. Diese Funktion ist besonders nützlich, um Anpassungen an Buchungen vorzunehmen, ohne sie löschen und erneut erstellen zu müssen, wodurch der Buchungsprozess effizienter und mitarbeiterfreundlicher gestaltet wird (*Abbildung 165 - DeskbookingServiceImplementation - updateBookingById Method*).

```

/** Updates a desk booking by its ID. ...*/
public DeskBooking updateBookingById(Long bookingId, DeskBooking updatedBooking) throws ResourceNotFoundException, DeskNotAvailableException {
    // Fetch the existing booking from the repository
    Optional<DeskBooking> existingBooking = deskBookingRepo.getBookingByBookingId(bookingId);

    // Check if the booking exists
    if (existingBooking.isEmpty()) {
        throw new ResourceNotFoundException("Booking not found for ID: " + bookingId);
    }

    // Check if the desk is available for the updated date and time
    Desk desk = updatedBooking.getDesk();
    LocalDate date = updatedBooking.getDate();
    LocalTime start = updatedBooking.getStart();
    LocalTime endTime = updatedBooking.getEndTime();

    List<Desk> availableDesks = getAvailableDesks(date, start, endTime, Optional.of(desk.getId()));
    if (availableDesks.isEmpty() || !availableDesks.contains(desk)) {
        throw new DeskNotAvailableException("Desk is not available for the specified date and time period.");
    }

    // Set the ID for the updated booking and save it
    updatedBooking.setId(bookingId);
    return deskBookingRepo.updateBooking(updatedBooking);
}

```

Abbildung 165 - DeskbookingServiceImplementation - updateBookingById Method

**getAvailableDesks:** Gibt eine Liste der verfügbaren Arbeitsplätze für ein bestimmtes Datum und einen bestimmten Zeitraum zurück. Es gibt eine optionale Möglichkeit, die Verfügbarkeit eines spezifischen Arbeitsplatzes zu überprüfen (Abbildung 166 - DeskbookingServiceImplementation - getAvailableDesks Method).

```

/** Fetches the available desks for a given date and time range. ...*/
@Override
public List<Desk> getAvailableDesks(LocalDate date, LocalTime start, LocalTime end, Optional<Long> deskId) throws ResourceNotFoundException {
    return deskBookingRepo.getAvailableDesks(date, start, end, deskId);
}

```

Abbildung 166 - DeskbookingServiceImplementation - getAvailableDesks Method

Die DeskServiceImplementation bietet eine umfangreiche Sammlung von Methoden zur Verwaltung von Arbeitsplatzbuchungen. Die Methoden "getBookingsByEmployeeId", "getBookingsByEmployeeNick" und "getBookingsByEmployeeAndDate" ermöglichen es, Buchungen basierend auf Mitarbeiterinformationen und Datum zu filtern. Es gibt auch spezifische Abfragen, um Buchungen nach Arbeitsplatz, Datum oder einer Kombination von beiden zu erhalten, wie in "getBookingByDesk", "getBookingsByDate" und "getBookingsByDeskAndDate" gezeigt.

Die Methode "deleteBookingById" ermöglicht die Löschung einer Buchung. Für eine detailliertere Analyse bietet "getBookingHistoryByEmployeeId" einen Einblick in die Buchungshistorie einer Mitarbeiterin oder eines Mitarbeiters über einen Zeitraum von zwei Wochen. Mit "getBookingByDateAndByStartBetween" können Benutzer Buchungen für spezifische Zeiträume eines bestimmten Datums abrufen. Die save-Methode stellt sicher, dass alle Buchungsinformationen korrekt in der Datenbank gespeichert werden. Schließlich hilft "getAvailableDesks", die verfügbaren Arbeitsplätze für gegebene Zeiträume und Daten zu identifizieren, wobei auch die Möglichkeit besteht, die Verfügbarkeit eines bestimmten Arbeitsplatzes zu überprüfen. Insgesamt stellt diese Implementierung sicher, dass Benutzer alle notwendigen Tools haben, um Arbeitsplatzbuchungen effizient zu verwalten und zu analysieren.

## 11.6 Presentation Layer

Dies ist die Benutzeroberflächen-Schicht (User Interface – UI). Sie ist verantwortlich für die Anzeige von Informationen und für die Interpretation der Befehle des Benutzers. Hierfür wurde das Model-View-Controller (MVC) Muster von Spring in Verbindung mit Thymeleaf verwendet.

Das Model-View-Controller (MVC) Muster ist ein Entwurfsmuster, das häufig in Webanwendungen und in Schichtenarchitektur verwendet wird. Es teilt eine Anwendung in drei miteinander verbundene Komponenten auf:

*Thymeleaf* ist eine moderne serverseitige Java-Template-Engine, die sowohl in Web- als auch in eigenständigen Umgebungen eingesetzt wird. Sie dient zur Verarbeitung und Erzeugung von HTML, XML, JavaScript, CSS und Text.

*Spring MVC (Model-View-Controller)* ist ein Modul des Spring Frameworks, das eine Architektur für die Entwicklung von Webanwendungen bereitstellt. Es basiert auf dem MVC-Designmuster, das die Anwendung in drei Hauptkomponenten unterteilt: Model, View und Controller. Dieses Muster erleichtert die Trennung von Anwendungslogik, Benutzeroberfläche und Datenzugriff. ([www.spring.io](http://www.spring.io), 2023)

Das Model-View-Controller (MVC) Muster ist ein Entwurfsmuster, das häufig in Webanwendungen und in Schichtenarchitektur verwendet wird. Es teilt eine Anwendung in drei miteinander verbundene Komponenten auf:

**Model (Modell):** Repräsentierten die Daten und die Geschäftslogik der Anwendung. Es kommuniziert direkt mit der Datenbank und aktualisiert die Ansicht, wenn die Daten geändert werden.

**View (Ansicht):** Das, was der Benutzer sieht und mit dem er interagiert. Es stellt die Benutzeroberfläche der Anwendung dar.

**Controller (Steuerelement):** Fungiert als Schnittstelle zwischen Modell und Ansicht. Der Controller empfängt Benutzereingaben von der Ansicht, fordert das Modell auf, diese Daten zu ändern oder abzurufen, und aktualisiert anschließend die Ansicht.

Ein zentrales Element in die Implementierung ist die Verwendung von Konstanten für URL-Pfade und Ansichten, wie in der *PathConstants* (Abbildung 168 - PathConstants) und der *ViewConstants* (Abbildung 167 - ViewConstants) Klassen definiert. Durch die Zentralisierung dieser Pfade in einer einzigen Klasse wird die Wartbarkeit verbessert, da Änderungen an den Pfaden nur an einem Ort vorgenommen werden müssen. Darüber hinaus trägt dies zur Sicherheit bei, da die Verwendung von Konstanten das Risiko von Tippfehlern oder inkonsistenten Pfaden reduziert, die zu Sicherheitslücken führen könnten.

```

public final class ViewConstants {

    // Prevent instantiation
    private ViewConstants() { throw new AssertionError( detailMessage: "ViewConstants class should not be instantiated." ); }

    //ADMIN & OPERATOR DESKBOOKING VIEWS
    public static final String A_ALL_DESKBOOKINGS = "deskbookings/admin/allDeskBookings";
    public static final String A_VIEW_DESKBOOKING = "deskbookings/admin/viewDeskBooking";
    public static final String A_ADD_DESKBOOKING = "deskbookings/admin/addDeskBooking";
    public static final String A_NEW_DESKBOOKING = "deskbookings/admin/newDeskBooking";
    public static final String A_UPDATE_DESKBOOKING = "deskbookings/admin/updateDeskBooking";
    public static final String A_CANCEL_DESKBOOKING = "deskbookings/admin/cancelDeskBooking";

    //NORMAL & PRIVILEGED EMPLOYEE DESKBOOKING VIEWS
    public static final String E_MY_DESKBOOKING = "deskbookings/emp/myDeskBooking";
    public static final String E_VIEW_DESKBOOKING = "deskbookings/emp/viewDeskBooking";
    public static final String E_ADD_DESKBOOKING = "deskbookings/emp/addDeskBooking";
    public static final String E_NEW_DESKBOOKING = "deskbookings/emp/newDeskBooking";
    public static final String E_UPDATE_DESKBOOKING = "deskbookings/emp/updateDeskBooking";
    public static final String E_HISTORY_DESKBOOKING = "deskbookings/emp/myDeskBookingHistory";
    public static final String E_CANCEL_DESKBOOKING = "deskbookings/emp/cancelDeskBooking";

    //DESK VIEWS
    public static final String ALL_DESKS = "desks/allDesks";
    public static final String ALL_DESKS_EMP = "desks/allDesksEmp";
    public static final String DESK_VIEW = "desks/viewDesk";
    public static final String ADD_DESK = "desks/addDesk";
    public static final String UPDATE_DESK = "desks/updateDesk";
    public static final String DELETE_DESK = "desks/deleteDesk";

    //HOLIDAY VIEWS
    public static final String ALL_HOLIDAYS = "desks/allHolidays";
    public static final String ADD_HOLIDAY = "desks/addHoliday";
    public static final String VIEW_HOLIDAY = "desks/viewHoliday";
    public static final String UPDATE_HOLIDAY = "desks/updateHoliday";
    public static final String DELETE_HOLIDAY = "desks/deleteHoliday";

    //TIMESLOT VIEWS
    public static final String ALL_TIMESLOTS = "desks/allTimeslots";
    public static final String ADD_TIMESLOT = "desks/addTimeslot";
    public static final String VIEW_TIMESLOT = "desks/viewTimeslot";
    public static final String UPDATE_TIMESLOT = "desks/updateTimeslot";
    public static final String DELETE_TIMESLOT = "desks/deleteTimeslot";

    //ERRORVIEW VIEW
    public static final String ERRORVIEW = "/error";
}

```

Abbildung 167 - ViewConstants

```

@Slf4j
public final class PathConstants {

    // Prevent instantiation
    private PathConstants() { throw new AssertionError( detailMessage: "PathConstants class should not be instantiated." ); }

    //DESKBOOKINGS
    // ADMIN & OPERATOR URL PATHS
    public static final String ADMIN_VIEW_ALL = "/admin";
    public static final String ADMIN_VIEW_ALL_P = "/web/deskbookings/admin";
    public static final String ADMIN_VIEW_BOOKING = "/admin/view/{id}";
    public static final String ADMIN_ADD = "/admin/add";
    public static final String ADMIN_NEW = "/admin/new/{deskId}";
    public static final String ADMIN_NEW_P = "/admin/new";
    public static final String ADMIN_UPDATE = "/admin/update/{id}";
    public static final String ADMIN_UPDATE_P = "/admin/update";
    public static final String ADMIN_CANCEL = "/admin/cancel/{id}";
    public static final String ADMIN_CANCEL_P = "/admin/cancel";

    // NORMAL & PRIVILEGED EMPLOYEE URL PATHS
    public static final String EMP_View_All= "/mydeskBookings";
    public static final String EMP_View_All_P= "/web/deskbookings/mydeskBookings";
    public static final String EMP_VIEW_BOOKING = "/view/{id}";
    public static final String EMP_ADD = "/add";
    public static final String EMP_NEW = "/new/{deskId}";
    public static final String EMP_NEW_P = "/new";
    public static final String EMP_UPDATE = "/update/{id}";
    public static final String EMP_UPDATE_P = "/update";
    public static final String EMP_VIEW_HISTORY = "/deskbookinghistory/{id}";
    public static final String EMP_CANCEL = "/cancel/{id}";
    public static final String EMP_CANCEL_P = "/cancel";

    //DESKS
    // ADMIN & OPERATOR URL PATHS
    public static final String VIEW_ALL = "/admin";
    public static final String VIEW_ALL_P = "/web/desks/admin";
    public static final String VIEW_DESK = "/view/{id}";
    public static final String ADD = "/add";
    public static final String UPDATE = "/update/{id}";
    public static final String UPDATE_P = "/update";
    public static final String DELETE = "/delete/{id}";
    public static final String DELETE_P = "/delete";

    // NORMAL & PRIVILEGED EMPLOYEE URL PATHS
    public static final String VIEW_ALL_EMP = "/employees";

    // ERRORVIEW URL PATH
    public static final String ERROR = "/error";
}

```

Abbildung 168 - PathConstants

Der *DeskController* dient als Vermittler zwischen dem Benutzer und der zugrundeliegenden Geschäftslogik. Er ist verantwortlich für die Verarbeitung von Webanfragen, die sich auf Arbeitsplätze beziehen, und nutzt den *DeskService*, um die erforderlichen Operationen durchzuführen.

Wenn ein Benutzer beispielsweise alle verfügbaren Arbeitsplätze sehen möchte, ruft er die Methode *getAllDesks* auf. Diese Methode interagiert mit dem *DeskService*, um die Daten abzurufen und

präsentiert sie dann in einer geeigneten Ansicht. Ebenso ermöglicht die Methode "*addDeskForm*" dem Benutzer, ein neues Desk-Objekt zu erstellen, indem ein Formular angezeigt wird. Nach dem Ausfüllen und Absenden des Formulars wird die Methode "*addDesk*" aufgerufen, um den neuen Arbeitsplatz tatsächlich hinzuzufügen.

Es gibt auch Methoden, die spezifische Ansichten für verschiedene Benutzertypen oder in unterschiedlichen Kontexten bieten, wie beispielsweise "*getAllEDesks*", die möglicherweise eine andere Ansicht der Arbeitsplätze für Mitarbeiter:innen zeigt.

Die Methoden "*viewDesk*" und "*updateDeskForm*" ermöglichen es den Benutzern, Details zu einem bestimmten Arbeitsplatz zu sehen und zu aktualisieren. Wenn ein Benutzer einen Arbeitsplatz löschen möchte, kann er dies über die Methoden "*deleteDeskForm*" und "*deleteDesk*" tun.

Schließlich gibt es die "*getError*"-Methode, die dazu dient, dem Benutzer Informationen über unerwartete Fehler zu liefern.

Der *DeskBookingController* dient als Vermittler zwischen dem Benutzer und der zugrundeliegenden Geschäftslogik für Arbeitsplatzbuchungen. Er nutzt den *DeskBookingService*, um auf die Daten und Geschäftslogik zuzugreifen und verarbeitet Benutzeranfragen, die durch verschiedene Aktionen wie das Klicken auf Schaltflächen oder das Ausfüllen von Formularen ausgelöst werden.

Wenn ein Benutzer alle seine Buchungen sehen möchte, ruft er die Methode "*getAllBookings*" auf. Um eine neue Buchung hinzuzufügen, stellt "*addBookingForm*" ein Formular zur Verfügung, und nach dem Absenden wird die Methode "*addBooking*" verwendet, um die Buchung tatsächlich zu erstellen. Es gibt auch spezielle Methoden, die für verschiedene Benutzertypen oder in unterschiedlichen Kontexten, wie "*getAllEBookings*", bereitgestellt werden.

Der Controller ermöglicht auch das Anzeigen von Details einer bestimmten Buchung mit "*viewBooking*", das Aktualisieren mit "*updateBookingForm*" und "*updateBooking*" sowie das Löschen mit "*deleteBookingForm*" und "*deleteBooking*".

| ID | Mitarbeiter    | Arbeitsplatz | Datum              | Start    | Ende     | Angelegt Am                 | Zuletzt Bearbeitet Am | Aktion                            |
|----|----------------|--------------|--------------------|----------|----------|-----------------------------|-----------------------|-----------------------------------|
| 10 | Marcel Schranz | D2-3         | September 7, 2023  | 08:00 am | 12:30 pm | September 7, 2023, 05:20 pm |                       | Anzeigen Bearbeiten<br>Stornieren |
| 11 | Sonja Lechner  | D1-3         | September 8, 2023  | 12:30 pm | 05:00 pm | September 7, 2023, 05:20 pm |                       | Anzeigen Bearbeiten<br>Stornieren |
| 12 | Jason Lechner  | D2-3         | September 9, 2023  | 08:00 am | 05:00 pm | September 7, 2023, 05:20 pm |                       | Anzeigen Bearbeiten<br>Stornieren |
| 13 | Jason Lechner  | D1-3         | September 10, 2023 | 08:00 am | 05:00 pm | September 7, 2023, 05:20 pm |                       | Anzeigen Bearbeiten<br>Stornieren |
| 14 | Camil Lechner  | D1-2         | September 11, 2023 | 08:00 am | 05:00 pm | September 7, 2023, 05:20 pm |                       | Anzeigen Bearbeiten<br>Stornieren |

Abbildung 169 - WABS - Admin View - AllDeskbookings

Wenn ein Administrator in der Ansicht "All Deskbookings" (Abbildung 171 - WABS - Update Deskbooking Form) eine Buchung bearbeiten möchte, kann er dies tun, indem er auf den "Bearbeiten"-Button klickt. Dies ruft die Methode `updateDeskBookingForm` im `DeskBookingController` auf (Abbildung 170 - DeskbookingController Method - `updateDeskBookingForm`).

```
/** Displays the form for updating a specific desk booking in the admin view. ...*/
@GetMapping(@ADMIN_UPDATE)
public String updateDeskBookingForm(@PathVariable Long id, Model model) throws ResourceNotFoundException{
    Optional<DeskBooking> booking = this.deskBookingService.getBookingById(id);

    if (booking.isEmpty()) {
        // Log an error message indicating that the booking was not found
        log.error("Desk booking with ID {} not found.", id);
        return "redirect:"+ A_ALL_DESKBOOKINGS;
    }

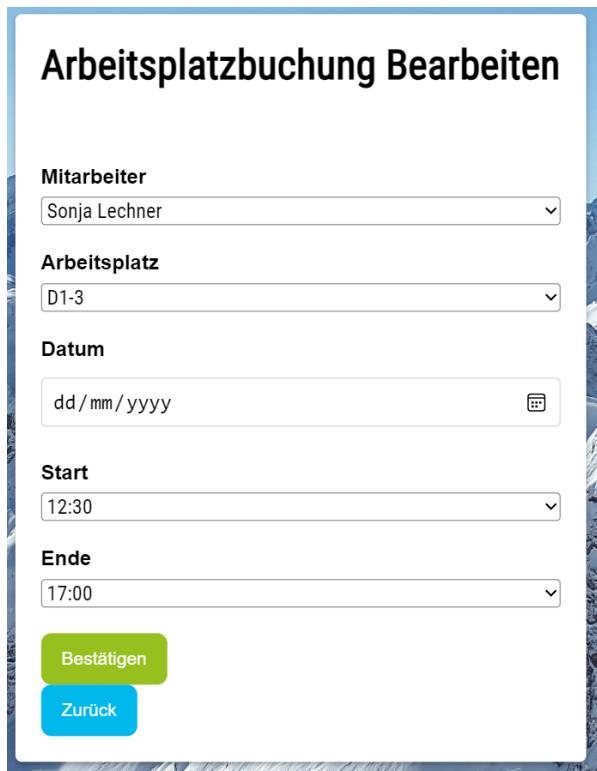
    // Convert the LocalDate to java.util.Date for the Thymeleaf template
    Date bookingDate = java.sql.Date.valueOf(booking.get().getDate());
    model.addAttribute( attributeName: "bookingDate", bookingDate);

    // Add booking, bookingDate, employee, desks, unique booking start and end times to the model
    model.addAttribute( attributeName: "booking", booking.get());
    model.addAttribute( attributeName: "bookingDate", bookingDate);
    if (!model.containsAttribute( attributeName: "errorMessage")) {
        model.addAttribute( attributeName: "errorMessage", attributeValue: null);
    }
    return A_UPDATE_DESKBOOKING;
}
```

Abbildung 170 - DeskbookingController Method - `updateDeskBookingForm`

Die Methode beginnt damit, die spezifische Arbeitsplatzbuchung über den `deskBookingService` basierend auf der übergebenen ID zu suchen. Wenn die Buchung nicht existiert, wird der Benutzer mit

einer Fehlermeldung zur Übersichtsseite weitergeleitet. Andernfalls wird das Buchungsdatum für die Verwendung im Thymeleaf-Template vorbereitet und zusammen mit anderen relevanten Daten dem Modell hinzugefügt. Die Methode endet, indem sie den Namen der Ansicht zurückgibt, die das Aktualisierungsformular anzeigt, (Abbildung 173) welches aus dem Thymeleaf Template erzeugt wurde (Abbildung 174). Falls die Buchung nicht gefunden werden sollte, wird eine Fehlermeldung protokolliert und zur Ansicht "All Deskbookings" (Abbildung 169 - WABS - Admin View - AllDeskbookings) welches aus dem Thymeleaf Template (Abbildung 175) erzeugt wurde, weitergeleitet.



The screenshot shows a web-based form titled "Arbeitsplatzbuchung Bearbeiten". The form fields are as follows:

- Mitarbeiter:** Sonja Lechner (selected in a dropdown)
- Arbeitsplatz:** D1-3 (selected in a dropdown)
- Datum:** dd/mm/yyyy (input field)
- Start:** 12:30 (selected in a dropdown)
- Ende:** 17:00 (selected in a dropdown)

At the bottom of the form are two buttons: a green "Bestätigen" (Confirm) button and a blue "Zurück" (Back) button.

Abbildung 171 - WABS - Update Deskbooking Formular

```

<div class="addRessourceContainer">
    <h1>Arbeitsplatzbuchung Bearbeiten</h1><br><br>
    <form method="post" th:action="@{/web/deskbookings/admin/update}" th:object="${booking}">
        <!-- <form method="post" th:action="@{/web/deskBookings/update/{id}({id=${booking.id}})" th:object="${booking}">-->
        <div th:if="${errorMessage}" class="alert alert-danger error-message">
            <p th:text="${errorMessage}"></p>
        </div>
        <div class="form-group">
            <label for="id" hidden="hidden"></label>
            <input class="form-control" id="id" name="id" required th:field="*{id}"
                   type="hidden">
        </div>
        <div class="form-group">
            <label for="employee.id">Mitarbeiter</label>
            <select class="form-control" id="employee.id" name="employee" th:field="*{employee.id}">
                <option selected="selected" value="">--Mitarbeiter wählen--</option>
                <option th:each="employee : ${employees}" th:text="${employee.fname} + ' ' + ${employee.lname}"
                       th:value="${employee.id}"></option>
            </select>
            <div th:errors="*{employee.id}" th:if="#{fields.hasErrors('employee.id')}"></div>
        </div>
        <div class="form-group">
            <label for="deskId">Arbeitsplatz</label>
            <select class="form-control" id="deskId" name="desk.id" th:field="*{desk.id}">
                <option selected="selected" value="">--Select--</option>
                <option th:each="desk : ${desks}" th:text="${desk.deskNr}" th:value="${desk.id}"></option>
            </select>
            <div th:errors="*{desk.id}" th:if="#{fields.hasErrors('desk.id')}"></div>
        </div>
        <div class="form-group">
            <label for="date">Datum</label>
            <input class="form-control" id="date" name="date" required
                   th:value="${bookingDate}"
                   th:field="*{date}" type="date">
            <div class="form-text" th:errors="*{date}">
                th:if="#{fields.hasErrors('date')}"></div>
            </div>
        <div class="form-group">
            <label for="start">Start</label>
            <select class="form-select" id="start" name="start" th:field="*{start}">
                <option value="--Start--">--Start--</option>
                <option th:each="time : ${startTimes}" th:text="${time}"
                       th:value="${time}"></option>
            </select>
            <div th:errors="*{start}" th:if="#{fields.hasErrors('start')}"></div>
        </div>
        <div class="form-group">
            <label for="endTime">Ende</label>
            <select class="form-select" id="endTime" name="endTime" th:field="*{endTime}">
                <option value="--Ende--">--Ende--</option>
                <option th:each="time : ${endTimes}" th:text="${time}"
                       th:value="${time}"></option>
            </select>
            <div th:errors="*{endTime}" th:if="#{fields.hasErrors('endTime')}"></div>
        </div>
        <button class="button" type="submit">Bestätigen</button>
    </form>

```

Abbildung 172 - Update Deskbooking Thymeleaf Template

```
/** Handles the POST request to update a desk booking. ...*/
@PostMapping(@v ADMIN_UPDATE_P)
public String updateDeskBooking(@Valid DeskBooking booking, BindingResult bindingResult,
    @RequestParam("id") Long id, @RequestParam("desk_id") Long deskId,
    @RequestParam("employee_id") Long employeeId, @RequestParam("date") String date,
    RedirectAttributes redirectAttributes)
throws ResourceNotFoundException, EmployeeNotFoundException, DeskNotAvailableException,
ExecutionException, InterruptedException {

    // Check if there are validation errors in the booking object.
    if (bindingResult.hasErrors()) {
        handleValidationErrors(bindingResult, redirectAttributes);
        return A_UPDATE_DESKBOOKING;
    } else {
        // Fetch the desk by its ID.
        Desk desk = deskService.getDeskById(deskId);
        if(desk == null) {
            throw new ResourceNotFoundException("Desk not found for id: " + deskId);
        }

        // Fetch the employee by its ID.
        Employee employee = employeeService.getEmployeeById(employeeId);
        if(employee == null) {
            throw new EmployeeNotFoundException("Employee not found for id: " + employeeId);
        }

        // Convert the date string to a LocalDate object.
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        LocalDate localDate = LocalDate.parse(date, formatter);

        // Set the updated attributes to the booking object.
        booking.setDate(localDate);
        booking.setDesk(desk);
        booking.setEmployee(employee);
        booking.setId(id);

        // Update the booking in the database.
        this.deskBookingService.updateBookingById(booking.getId(), booking);

        // Redirect to the view all bookings page.
        return "redirect:" + ADMIN_VIEW_ALL_P;
    }
}
```

Abbildung 173 - DeskbookingController Methode - updateDeskBooking

| ID | Mitarbeiter    | Arbeitsplatz | Datum              | Start    | Ende     | Angelegt Am                 | Zuletzt Bearbeitet Am       | Aktion   |
|----|----------------|--------------|--------------------|----------|----------|-----------------------------|-----------------------------|--|
| 10 | Marcel Schranz | D2-3         | September 7, 2023  | 08:00 am | 12:30 pm | September 7, 2023, 05:20 pm |                             | <button>Anzeigen</button> <button>Bearbeiten</button><br><button>Stornieren</button> |
| 11 | Sonja Lechner  | D1-3         | September 20, 2023 | 12:30 pm | 05:00 pm | September 7, 2023, 05:20 pm | September 7, 2023, 05:58 pm | <button>Anzeigen</button> <button>Bearbeiten</button><br><button>Stornieren</button> |
| 12 | Jason Lechner  | D2-3         | September 9, 2023  | 08:00 am | 05:00 pm | September 7, 2023, 05:20 pm |                             | <button>Anzeigen</button> <button>Bearbeiten</button><br><button>Stornieren</button> |
| 13 | Jason Lechner  | D1-3         | September 10, 2023 | 08:00 am | 05:00 pm | September 7, 2023, 05:20 pm |                             | <button>Anzeigen</button> <button>Bearbeiten</button><br><button>Stornieren</button> |
| 14 | Camil Lechner  | D1-2         | September 11, 2023 | 08:00 am | 05:00 pm | September 7, 2023, 05:20 pm |                             | <button>Anzeigen</button> <button>Bearbeiten</button><br><button>Stornieren</button> |

Abbildung 174 - WABS – AllDeskbookings Ansicht

## 11.7 Testen des Deskbooking-Moduls

Das Deskbooking-Modul ist ein zentrales Element des Systems und somit ist es von entscheidender Bedeutung, dass es korrekt funktioniert. In diesem Kapitel werden die spezifischen Testmethoden vorgestellt, die während der Entwicklung dieses Moduls verwendet wurden.

Das Testen des Deskbooking-Moduls stellt sicher, dass alle Funktionen, von der Buchung eines Schreibtisches bis zur Verwaltung von Reservierungen, wie erwartet arbeiten. Dieses Kapitel bietet einen Überblick über die verschiedenen Testansätze und -techniken, die angewendet wurden.

### Test Beispiel

Der Test "testAddDeskBooking" ist ein Einheitstest. Dies wird durch die Struktur und die Verwendung von *Mocking* deutlich. Der Test konzentriert sich auf eine spezifische Funktionalität, nämlich das Hinzufügen einer Arbeitsplatzbuchung.

Im Setup wird ein Mock-Objekt von DeskBooking namens mockBooking erstellt, das ein Datum auf einen Tag ab dem aktuellen Datum und einen Zeitbereich von 9:00 bis 17:00 Uhr festlegt. Der Test simuliert das Verhalten des deskBookingRepo, das ein Repository oder ein Datenzugriffsobjekt für Arbeitsplatzbuchungen sein soll. Wenn die Methode getBookingsByDeskAndDateAndBookingTimeRange des deskBookingRepo mit beliebigen Werten für ihre Parameter aufgerufen wird, gibt sie eine leere Liste zurück. Dies simuliert ein Szenario, in dem für den gegebenen Schreibtisch, das Datum und den Zeitbereich keine Buchungen vorhanden sind.

Wenn die Methode addBooking des deskBookingRepo mit einem DeskBooking-Objekt aufgerufen wird, gibt sie das mockBooking-Objekt zurück. Dies simuliert das erfolgreiche Hinzufügen einer Buchung. Die Methode addBooking des deskBookingRepo wird mit dem mockBooking-Objekt aufgerufen, und das Ergebnis wird in der Variablen "result" gespeichert.

Der Test überprüft, ob das Ergebnis nicht null ist, um sicherzustellen, dass eine Buchung zurückgegeben wurde. Der Test überprüft auch, ob das Datum des Ergebnisses mit dem Datum von mockBooking übereinstimmt, um sicherzustellen, dass die richtige Buchung hinzugefügt wurde.

Die erwarteten Ergebnisse sind, dass das Ergebnis nicht null sein sollte und dass das Datum des Ergebnisses mit dem Datum von mockBooking übereinstimmen sollte. Die tatsächlichen Ergebnisse würden von der Ausführung des Tests abhängen. Wenn der Test jedoch ohne Fehler besteht, bedeutet dies, dass die erhaltenen Ergebnisse den erwarteten Ergebnissen entsprechen. Wenn der Test fehlschlägt, wurden eine oder beide Behauptungen nicht erfüllt (Abbildung 175 - testAddBooking Method).

```
@Test
public void testAddDeskBooking() throws Exception {
    DeskBooking mockBooking = new DeskBooking();
    mockBooking.setDate(LocalDate.now().plusDays( daysToAdd: 1));
    mockBooking.setStart(LocalTime.of( hour: 9, minute: 0));
    mockBooking.setEndTime(LocalTime.of( hour: 17, minute: 0));

    when(deskBookingRepo.getBookingsByDeskAndDateAndBookingTimeRange(anyLong(),
        any(LocalDate.class), any(LocalTime.class), any(LocalTime.class)))
        .thenReturn(Collections.emptyList());

    when(deskBookingRepo.addBooking(any(DeskBooking.class)))
        .thenReturn(mockBooking);

    DeskBooking result = deskBookingRepo.addBooking(mockBooking);

    assertNotNull(result);
    assertEquals(mockBooking.getDate(), result.getDate());
}
```

Abbildung 175 - testAddBooking Method

# 12 Evaluation

Im nachstehenden Abschnitt wird eine Projekt-, Produktevaluierung vorgestellt, wo nachgelesen werden kann, welche Änderungen am Produkt bzw. welche Änderungen in der Projektstruktur im Laufe der Arbeit entstanden und wie sie sich auf das Projekt im gesamten ausgewirkt hat. Am Ende des Kapitels lassen sich persönliche Resümees der einzelnen Projektmitgliedern finden.

## 12.1 Projektevaluierung

### Das Team und die Modulaufteilung

Anfangs bestand das Team lediglich aus drei Personen. Das gesamte Projekt wurde in unterschiedliche Module aufgeteilt und den Mitgliedern des Projektteams zugewiesen. Kurz nach der ersten Einteilung wurde das Team um ein weiteres Mitglied erweitert. Im Nachhinein entstanden Probleme mit der Zuweisung der Module, denn es musste sichergestellt werden, dass die Aufgabenaufteilung untereinander fair verteilt war.

Das Resultat war, dass sich die Einteilung der Module im Laufe des Projektes immer wieder geändert haben. Als Beispiel:

Zu Beginn des Projektes war Herr Payer dem Buchungsmodul zugeteilt, welches dann an Frau Lechner übergeben wurde. Herr Payer hat hingegen das Ressourcenmodul von Herrn Bayr übernommen und dieser wiederum erhielt das Raumverwaltungsmodul.

Danach wurden immer wieder neue kleine Module gefunden, die auf die Mitglieder aufgeteilt wurden. Die Endgültige Aufteilung der Module war dann wie folgt:

Herr Bayr: Raumverwaltung, Dokumentation, Management des Projektes via GitHub

Herr Schranz: Mitarbeiterverwaltung, Authentifizierung und Benutzergruppen, Log-In-Funktionalität.

Frau Lechner: Arbeitsplatzmodul sowie das Arbeitsplatz-Buchungsmodul inkl. Erstellung der Grundstruktur des Projektes.

Herr Payer: Ressourcenverwaltung, Frontend, Corporate Design.

Kleinere Funktionalitäten, die nicht vom Kunden angefordert wurden, wie Internationalisierung oder Dark-/ Light-Theme wurden aufgrund von Zeitstress dann auf einen späteren Zeitpunkt gelegt.

## 12.2 Produktevaluierung

### 12.2.1 Google Firebase

Die nennenswerteste Änderung im Bezug auf das Projekt war die nachträgliche Umstellung von Google Firebase. Ziel war es ursprünglich, dass die Datenbank über Google Firebase gehostet werden würde. Somit wäre die Datenbank und der Zugriff auf das Hostsystem zu Google überlagert worden.

Nachträglich hat sich jedoch rausgestellt, dass Google Firebase nicht jene Technologie ist, die auf das Projekt optimal passt. Es gab mehrere Komplikationen, die das Weiterarbeiten mit Firebase erübrigten. So ist Firebase nicht dazu im Stande ein relationales Datenbankmodell zu erstellen, welches das Projektteam bei den eingesetzten Technologien bevorzugen würde.

Nach Rücksprache mit den Betreuern und diversen anderen Lehrpersonen wurde beschlossen, das System an den bereits bestehenden Host, den das Klimabündnis bis dato verwendet, anzubinden. Durch diese Umstellung konnte man das ursprüngliche Problem der nicht relationalen Datenbank beheben dafür ist aber eine stärkere Abhängigkeit zum bereits bestehenden Host System entstanden. Wenn das System aus technischen Gründen nicht erreichbar sein sollte, so ist auch die Anwendung W.A.B.S. nicht erreichbar.

### 12.2.2 Frontend

Zu Beginn der Arbeit wurde vorübergehend beschlossen, dass das Frontend hauptsächlich nur auf HTML5, Java Script, CSS, und Bootstrap aufbaut. Dies wurde teilweise auch so realisiert. Das Frontend besteht größtenteils aus HTML5 und CSS jedoch wurde zusätzlich zu den oben angeführten technologien auch mit Thymeleaf gearbeitet. Thymeleaf übernimmt dabei nicht nur die essenzielle Rolle des Mappings von Daten, sondern ist auch eine große Hilfe bei der Erschaffung des Frontends auf Basis des Backends. Da Thymeleaf eine wichtige Komponente innerhalb des Projektes ist wurde auch versucht so viel wie möglich diese Technologie zu verwenden. So konnte auch der Aufwand, PHP verbauen zu müssen, auf das minimale beschränkt werden.

### 12.2.3 JQuery UI

JQuery UI hätte die Java Script Bibliothek um sehr viele neue Funktionalitäten bzw. Möglichkeiten erweitert. Geplant war, dass JQuery UI dabei helfen soll, die Anforderung des Kunden, dass der Benutzer unkompliziert den Grundriss des Gebäudes selbst einzubetten und Tische und Co. Via Drag and Drop Funktionalität zuzuweisen.

Nach diversen Fachgesprächen zwischen dem Projektteam und unterschiedlichen Experten des IT-Kolleg Imst, hat man beschlossen, das Jquery nicht die geeignete Technologie ist, um diese Anforderung zu realisieren. Die Festlegung auf eine finale Technologie hat hier leider noch nicht stattgefunden. Als alternative Technologie wurde dann SVG zu Rate gezogen. SVG hätte das Ganze noch unkomplizierter als Jquery UI gemacht, aber leider war selbst diese Technologie nicht ganz auf die gewünschten Bedürfnisse zugeschnitten.

Durch die ständigen Änderungen der Technologie, stand das Modul immer wieder am Anfang und ist derzeit auch noch in der Entwicklungsphase. Durch den engen Zeitrahmen ist es fraglich, ob die Anforderung des Kunden zeitgerecht übergeben werden kann. Derzeit ist jedoch der aktuelle Grundriss mit klickbaren UI-Elementen hinterlegt. Lediglich der Aspekt der Wartbarkeit ist noch ausständig.

## 12.3 Resümeees

### 12.3.1 Resümee Bayr

#### Evaluation des Raum-Moduls in der Arbeitsplatzbuchungsanwendung

1. Beschreibung des Raum-Moduls: Das Raum-Modul in der W.A.B.S-Applikation ermöglicht eine benutzerfreundliche und intuitive Darstellung der Räume. Über dieses Modul können Buchungen für die einzelnen Arbeitsplätze in den Räumen vorgenommen werden. Das Raum-Modul spielt somit eine entscheidende Rolle in dem Projekt.
2. Grundidee zur Umsetzung der Räume mit SVGs: Anfänglich wurde in Betracht gezogen, die Raumpläne als "Scalable Vector Graphics" (SVG) darzustellen, da diese eine geräteunabhängige Skalierung ermöglichen. Allerdings stellte sich heraus, dass die Erstellung neuer Räume für Administratoren in diesem Format sehr komplex ist. Das Extrahieren der benötigten Koordinaten aus einem vorhandenen Raumplan, der meist im PNG-Format vorliegt, ist ohne Bildbearbeitungsprogramme wie z.B. "Inkscape" kaum möglich.
3. Umsetzung mit PNG durch das HTML-Element "Imagemap": Mit freundlicher Unterstützung der Lehrpersonen entstand die Idee, die Raumpläne direkt im gegebenen PNG-Format darzustellen. Nach Rücksprache mit dem Projektpartner wurde bestätigt, dass die übermittelten Grundrisse für die allgemeine Raumansicht verwendet werden sollen. Für diesen Anwendungsfall wurde das HTML-Element "Imagemap" als geeigneter Workaround identifiziert. Dadurch ist es möglich, die einzelnen Räume auf dem Grundriss durch die Angabe weniger Koordinaten hervorzuheben.

4. Herausforderung der geräteabhängigen Darstellung: Durch die Verwendung von Imagemaps konnte die Hervorhebung der Räume und die Interaktion mit ihnen sichergestellt werden. Eine Herausforderung bestand jedoch darin, die absoluten Raumkoordinaten im Vergleich zur Grundrissdimension anzugeben. Absolute Koordinaten skalieren nicht und sind fix. Es gibt auch relative Pfade, die in der Regel prozentuale Angaben zur Gesamtgröße verwenden. Allerdings stellte sich nach umfangreichen Tests heraus, dass die meisten Browser Schwierigkeiten haben, diese relativen Pfade richtig zu interpretieren.
5. Umwandlung der Koordinaten mithilfe von JavaScript: Dank eines Expertentipps wurde darauf aufmerksam gemacht, dass es möglich ist, die Bildschirmdimensionen vor dem Laden der Website zu ermitteln. Diese Information ermöglicht es, eine JavaScript-Funktion zu erstellen, die die absoluten Pfade in neue absolute Koordinaten umwandelt, abhängig von den Bildschirmdimensionen.

Die Evaluation des Raum-Moduls in der Arbeitsplatzbuchungsanwendung hat gezeigt, dass die Umsetzung der Räume durch SVGs zu einer erhöhten Komplexität bei der Erstellung neuer Räume führt. Die Verwendung von PNG-Bildern in Kombination mit dem HTML-Element Imagemap erwies sich als geeigneter Workaround für die Darstellung der Räume und ermöglichte die Interaktion mit ihnen. Allerdings stellte die geräteabhängige Darstellung der absoluten Raumkoordinaten eine Herausforderung dar, die durch die Umwandlung der Koordinaten mittels JavaScript gelöst wurde. Insgesamt hat das Raum-Modul eine benutzerfreundliche und intuitive Darstellung der Räume ermöglicht, und die vorgenommenen Anpassungen haben zu einer verbesserten Nutzererfahrung beigetragen. Es empfiehlt sich jedoch, weiterhin die Möglichkeiten zur Vereinfachung der Raum-Erstellung und zur Optimierung der geräteabhängigen Darstellung zu erforschen, um die Effizienz und Flexibilität des Raum-Moduls weiter zu verbessern.

### 12.3.2 Resümee Lechner

Im Verlauf des Projekts ist man auf verschiedenste Probleme und Herausforderungen gestoßen, aus denen wertvolle Informationen und Erkenntnisse gewonnen wurden. Die folgenden Punkte fassen kurz zusammen, was ich für zukünftige Projekte gelernt habe:

1. Zu Beginn waren manche Kundenanforderungen nicht klar. Um Missverständnisse zu klären, war es wichtig, entweder erneut Kontakt mit dem Kunden aufzunehmen oder im Team Annahmen zu treffen, die sich im Nachhinein als richtig erwiesen. Eine gründliche Anforderungsanalyse und klare Kommunikation mit dem Kunden sind entscheidend, um ein gemeinsames Verständnis der Projektziele und -anforderungen zu erreichen. Dadurch können Missverständnisse und spätere Änderungen reduziert werden.
2. Die Zeitplanung zu Beginn war idealistisch, was zu ungeplanten Ereignissen und dem Nicht-Einhaltung einiger Termine führte. Eine realistische Planung unter Berücksichtigung von Pufferzeiten und unvorhergesehenen Risiken ist wichtig, um Zeit- und Budgetüberschreitungen zu vermeiden. Eine regelmäßige Überwachung des Projektfortschritts und eine frühzeitige Erkennung von Abweichungen ermöglichen eine rechtzeitige Anpassung des Projektplans.
3. Bei der Zuteilung der Verantwortung für verschiedene Arbeitsbereiche und Zuständigkeiten ist klar geworden, dass eine andere Zuteilung möglicherweise vorteilhafter gewesen wäre. Die Kommunikation innerhalb des Teams verlief nicht immer reibungslos, und es gab Schwierigkeiten bei der Planung von Treffen, die von allen Teammitgliedern eingehalten wurden. Die Auswahl und das Management des Projektteams sind von großer Bedeutung. Es ist wichtig, Teammitglieder mit den richtigen Fähigkeiten und Erfahrungen auszuwählen und klare Rollen und Verantwortlichkeiten festzulegen. Eine effektive Kommunikation und Zusammenarbeit innerhalb des Teams fördern den Erfolg des Projekts.
4. Das Projekt lief nicht reibungslos, insbesondere im Hinblick auf Continuous Integration in GitHub und die Durchführung von Tests. Eine regelmäßige Qualitätssicherung und Überprüfung während des gesamten Projektablaufs hätte geholfen, die Einhaltung der definierten Standards und Anforderungen sicherzustellen. Frühzeitige Tests und Feedbackschleifen ermöglichen es, Probleme frühzeitig zu erkennen und zu beheben.
5. Der Projektplan war nicht detailliert genug erstellt, was zu mehreren Änderungen führte. Dadurch entstanden manchmal Schwierigkeiten und Verzögerungen im Projektverlauf. Eine gute Change-Management-Strategie ist wichtig, um mit Änderungen während des Projekts umzugehen. Änderungen sollten sorgfältig bewertet, dokumentiert und kommuniziert werden, um Auswirkungen auf den Projektumfang, die Zeitplanung und das Budget zu minimieren.
6. Eine kontinuierliche Verbesserung und retrospektive Bewertung des Projekts sind wichtig, um Lessons Learned festzuhalten und Best Practices für zukünftige Projekte zu identifizieren. Das Teilen von Erfahrungen und Wissen innerhalb des Teams trägt zur kontinuierlichen Weiterentwicklung bei.

Insgesamt hat das Projekt wertvolle Erfahrungen geliefert, die es ermöglichen, in zukünftigen Projekten effizienter und erfolgreicher zu sein. Durch Berücksichtigung dieser Erkenntnisse können zukünftige Projekte optimiert und die Kundenzufriedenheit gesteigert werden.

### 12.3.3 Resümee Payer

Evaluation des Ressourcenmoduls und des Frontend inkl. Corporate Design

1. Beschreibung des Ressourcenmoduls: Beamer, Whiteboards, Kamera, Lastenfahrrad und viele andere Ressourcen, die im Unternehmen existieren werden vom Ressourcenmodul verwaltet. Das Modul bietet die üblichen CRUD-Methoden (Create, Read, Update, Delete), die dem Administrator ermöglichen, eine Übersicht über die bestehenden Ressourcen zu behalten. Ressourcen besitzen damit auch diverse Daten, die eingegeben werden können wie z.B. Seriennummern, Info, Beschreibung, Kategorie usw...
2. Beschreibung des Frontends: Das Modul Frontend schließt nicht nur die GUI, sondern auch die Umsetzung des Corporate Designs mit ein. Es wurde von Anfang an dafür gesorgt, dass das Frontend auf HTML-Basis aufgebaut wird. Es wurde auch ein eigenes CSS entwickelt, welches ein Template dekoriert, welches wiederum die Vorlage für alle HTML Seiten im Dokument ist. Ursprünglich wurde beschlossen, dass das Frontend von 2 Personen erledigt wird. Schlussendlich ist der Großteil des Frontends von Herrn Payer alleine bearbeitet worden.
3. Grund für die gewählten Technologien: Das Frontend wurde vorwiegend nur mittels HTML5 erstellt. Dies hat sich angeboten, da HTML im Rahmen der Ausbildung am IT-Kolleg Imst behandelt wurde. Da es eine sehr simple und einfache Sprache ist, blieb die Entscheidung bei HTML5. Alternativ hätte man das Frontend auch auf Java Script Basis erstellen können. Das verwendete CSS wurde selbst erstellt. Man hat die Verwendung von Bootstrap so gering wie möglich gehalten. Das geschah lediglich aufgrund dessen, dass Herr Payer den Großteil selbst gestalten/implementieren wollte.

Das Erstellen des Frontends hat mir ein tieferes Verständnis für HTML, CSS, JS, PHP gegeben, denn das war nicht wirklich meine Stärke. Daher habe ich mich auch dafür entschieden, dass ich das Tool das Modul übernehmen möchte. Der Umgang mit den verschiedenen Technologien hat mir ein umfangreiches und solides Basiswissen beschafft, welches ich für meine künftige berufliche Laufbahn bestimmt nutzbringend einsetzen werde.

#### 12.3.4 Resümee Schranz

##### Evaluation des Employee-Moduls und der Spring Security in der Arbeitsplatzbuchungsanwendung

1. Beschreibung des Employee-Moduls: Das Employee-Modul in der Arbeitsplatzbuchungsanwendung ermöglicht die Verwaltung von Mitarbeitern. Es bietet CRUD (Create, Read, Update, Delete) Methoden, die es dem Administrator ermöglichen, neue Mitarbeiter anzulegen, bestehende Mitarbeiter zu bearbeiten und Mitarbeiter zu löschen. Das Modul ist wichtig für die korrekte Verwaltung und Organisation der Mitarbeiterdaten in der Anwendung.
2. Beschreibung der Spring Security: Die Spring Security wird in der Arbeitsplatzbuchungsanwendung für die Benutzerauthentifizierung über den USERSERVICE verwendet. Die Benutzerinformationen werden über eine H2 in-memory Datenbank gespeichert. Spring Security spielt eine wesentliche Rolle bei der Gewährleistung der Sicherheit und des Zugriffsschutzes in der Anwendung. Es ermöglicht den Benutzern, sich sicher anzumelden und gewährleistet, dass nur autorisierte Benutzer auf die Ressourcen der Anwendung zugreifen können.
3. Gründe für den Wechsel zur RDBM: Der Wechsel von Google Firebase zu einer relationalen Datenbank (RDBM) wie der H2 in-memory Datenbank wurde aus verschiedenen Gründen vorgenommen. Eine RDBM schien sinnvoller für die Arbeitsplatzbuchungsanwendung, da sie besser an die Anforderungen des Systems angepasst werden konnte. Zudem wurde das Konzept relationaler Datenbanken im Unterricht ausführlicher behandelt, was die Implementierung und Verwaltung erleichterte. Der Wechsel zur RDBM hat auch die Möglichkeit eröffnet, die Leistung und Skalierbarkeit der Anwendung zu verbessern.
4. Einrichtung von Spring Security: Die Einrichtung von Spring Security in der Arbeitsplatzbuchungsanwendung beinhaltet mehrere Schritte. Zunächst werden die erforderlichen Abhängigkeiten in den Projektbuild eingefügt. Anschließend wird die Konfiguration der Sicherheitseinstellungen vorgenommen, einschließlich der Definition von Rollen und Berechtigungen sowie der Festlegung von Zugriffsregeln für die verschiedenen Ressourcen. Die Benutzerinformationen werden in der H2 in-memory Datenbank gespeichert, und die Authentifizierung erfolgt über Benutzername und Passwort. Durch die Einrichtung von Spring Security wird sichergestellt, dass nur authentifizierte Benutzer auf die geschützten Bereiche der Anwendung zugreifen können.
5. Bewertung der CRUD-Methoden für die Administratorfunktionen: Die CRUD-Methoden im Employee-Modul, die es dem Administrator ermöglichen, neue Mitarbeiter anzulegen, zu bearbeiten und zu löschen, sind entscheidend für die effektive Verwaltung der Mitarbeiterdaten. Die Implementierung dieser Funktionen sollte benutzerfreundlich und intuitiv sein, um dem Administrator eine einfache Bedienung zu ermöglichen. Es ist wichtig sicherzustellen, dass die CRUD-Methoden ordnungsgemäß validieren, ob bestimmte Aktionen zulässig sind und ob alle erforderlichen Daten vorhanden sind, um Mitarbeiter erfolgreich anzulegen, zu bearbeiten oder zu löschen. Die Leistungsfähigkeit und Effizienz dieser Funktionen sind entscheidend, um eine reibungslose Verwaltung der Mitarbeiterdaten zu gewährleisten.

Nach intensiver Auseinandersetzung mit Google Firebase bin ich nun mit der Handhabung einer dokumentenorientierten Datenbank vertraut, kann problemlos Collections erstellen und verwalten und die Vorteile des Hostings über Google nutzen. Diese Erfahrung hat mich nicht nur fachlich bereichert, sondern auch gezeigt, wie vielseitig und leistungsfähig moderne Datenbanktechnologien sein können. Durch die umfassende Beschäftigung mit Spring Security und der Implementierung von CRUD Methoden in Spring Boot habe ich wertvolle Kenntnisse und Fähigkeiten erlangt. Ich bin nun in der Lage, die Sicherheit einer Anwendung effektiv zu gewährleisten und den Benutzern eine sichere Authentifizierung und Autorisierung zu bieten. Die Integration von Spring Security in die Arbeitsplatzbuchungsanwendung hat mir gezeigt, wie leistungsfähig und flexibel diese Bibliothek ist, wenn es darum geht, Zugriffsregeln und Berechtigungen zu definieren. Darüber hinaus ermöglicht die Implementierung von CRUD Methoden eine effiziente Verwaltung der Mitarbeiterdaten, wodurch die Anwendung benutzerfreundlich und intuitiv wird. Die Kombination aus Spring Security und den CRUD Methoden in Spring Boot hat mein Verständnis für die Entwicklung sicherer und funktionaler Anwendungen erweitert und mich auf dem Gebiet der Webentwicklung einen großen Schritt vorangebracht.

## 13 Zusammenfassung

Das Work-Area Booking System (W.A.B.S) ist eine webbasierte Java-Applikation, die mit dem Spring-Framework erstellt wurde. Diese Anwendung wurde entwickelt, um Unternehmen dabei zu unterstützen, ihre Büroflächen effizienter zu nutzen und die Zusammenarbeit ihrer Mitarbeiter:innen zu verbessern. Insbesondere in der aktuellen Covid-19-Pandemie haben viele Unternehmen ihren Mitarbeiter:innen erlaubt, von zu Hause aus zu arbeiten, was zu einem erhöhten Bedarf an flexiblen Arbeitsplätzen und -ressourcen im Büro geführt hat. Das W.A.B.S stellt hierfür eine innovative Lösung dar.

Die Applikation ermöglicht es den Mitarbeiterinnen, Arbeitsplätze und Ressourcen online zu buchen, was zu einer höheren Flexibilität bei der Arbeitszeitgestaltung führt. Gleichzeitig erhöht es die Mitarbeiterinnenzufriedenheit und steigert die Produktivität. Das System ist in der Lage, Engpässe bei Arbeitsplätzen im Büro zu beseitigen und somit das Potenzial von Büroflächen vollständig auszuschöpfen. Unternehmen können die Anwendung nutzen, um eine bessere Zusammenarbeit ihrer Mitarbeiter\*innen zu ermöglichen, indem sie beispielsweise Räume für Meetings und Projekte reservieren.

Die Authentifizierung und Autorisierung wird mithilfe von Spring Secure umgesetzt, was für eine hohe Sicherheit des Systems sorgt. Das Backend erzeugt mithilfe von Thymeleaf die Komponenten für das Frontend, während die Interagierbarkeit durch JavaScript Bibliotheken wie jQueryUI und SVG.js realisiert wird. Durch diese Technologien wird eine benutzerfreundliche Oberfläche geschaffen, die es den Mitarbeiter\*innen leicht macht, die verfügbaren Ressourcen zu finden und zu reservieren.

Das W.A.B.S wurde speziell für den Partner Klimabündnis Tirol entwickelt, der sich für nachhaltige Entwicklung und Umweltschutz einsetzt. Durch die Möglichkeit, Arbeitsplätze und Ressourcen online zu buchen, kann das System dazu beitragen, die CO2-Emissionen durch Pendelfahrten der Mitarbeiter\*innen zu reduzieren und somit einen Beitrag zum Klimaschutz leisten.

Insgesamt ist das W.A.B.S eine innovative Lösung für Unternehmen, die ihre Büroflächen effizienter nutzen und die Zusammenarbeit ihrer Mitarbeiterinnen verbessern möchten. Durch die Kombination von Java, Spring-Framework, Thymeleaf und Javascript Bibliotheken wird eine benutzerfreundliche Oberfläche geschaffen, die hohe Sicherheitsstandards erfüllt. Das System bietet Flexibilität bei der Arbeitszeitgestaltung, erhöht die Mitarbeiterinnenzufriedenheit und steigert die Produktivität.

# 14 Literaturverzeichnis

(2023). Von wikipedia: [https://de.wikipedia.org/wiki/Referentielle\\_Integrität](https://de.wikipedia.org/wiki/Referentielle_Integrität) abgerufen

(2023). Von wikipedia: [https://de.wikipedia.org/wiki/Singleton\\_\(Entwurfsmuster\)](https://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster)) abgerufen

(2023). Von wikipedia: [https://de.wikipedia.org/wiki/Dependency\\_Injection](https://de.wikipedia.org/wiki/Dependency_Injection) abgerufen

*Atlassian.* (9. 12 2022). Von Atlassian: <https://www.atlassian.com/de/agile/project-management/user-stories> abgerufen

*baeldung.* (26. 3 2023). Von baeldung: <https://www.baeldung.com/spring-controller-vs-restcontroller> abgerufen

*Baeldung.* (2023). Von <https://www.baeldung.com/spring-bean> abgerufen

*Baeldung.* (2023). *baeldung.com.* Von <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring> abgerufen

*Baeldung Lombok.* (08. 09 2023). Von Baeldung: <https://www.baeldung.com/intro-to-project-lombok> abgerufen

*Baeldung Lombok.* (08. 09 2023). Von Baeldung: <https://www.baeldung.com/intro-to-project-lombok> abgerufen

*Baeldung.com.* (09 2023). *Thymeleaf in Spring MVC.* Von [www.baeldung.com: https://www.baeldung.com/thymeleaf-in-spring-mvc](https://www.baeldung.com/thymeleaf-in-spring-mvc) abgerufen

*Byte-Welt.* (2023). Von Byte-Welt: [https://wiki.byte-welt.net/wiki/Methode\\_\(Java\)#:~:text=Eine%20Methode%20besteht%20aus%20einem,Code%2C%20der%20tats%C3%A4chlich%20ausgef%C3%BChrt%20wird.&text=Der%20Kopf%20ist%20stets%20vor,stets%20durch%20geschweifte%20Klammern%20gekennzeichnet](https://wiki.byte-welt.net/wiki/Methode_(Java)#:~:text=Eine%20Methode%20besteht%20aus%20einem,Code%2C%20der%20tats%C3%A4chlich%20ausgef%C3%BChrt%20wird.&text=Der%20Kopf%20ist%20stets%20vor,stets%20durch%20geschweifte%20Klammern%20gekennzeichnet). abgerufen

*computerweekly.* (26. 3 2023). Von computerweekly: <https://www.computerweekly.com/de/definition/Back-End-Front-End#:~:text=Frontend%20und%20Backend%20k%C3%B6nnen%20auch,ausschlie%C3%9Flich%20f%C3%BCr%C3%BCr%20authentifizierte%20Benutzer%20gelten>. abgerufen

*d3js.org.* (28. 3 2023). Von d3js.org: <https://d3js.org/> abgerufen

*de.ryte.com.* (26. 3 2023). Von de.ryte.com: <https://de.ryte.com/wiki/HTML#:~:text=HTML%20bedeutet%20Hypertext%20Markup%20Language,Informationen%2C%20die%20diese%20Inhalte%20beschreiben>. abgerufen

*docs.oracle.* (26. 3 2023). Von docs.oracle: <https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html> abgerufen

*firebase.google.com.* (28. 3 2023). Von firebase.google.com: <https://firebase.google.com/> abgerufen

Foundation, T. A. (2023). *Apache Shiro.* Von <https://shiro.apache.org/get-started.html> abgerufen

Foundation, T. A. (2023). *Apache Struts.* Von <https://struts.apache.org/getting-started/> abgerufen

*freemarker.apache.org*. (28. 3 2023). Von freemarker.apache.org: <https://freemarker.apache.org/> abgerufen

*getbootstrap.com*. (28. 3 2023). Von getbootstrap.com: <https://getbootstrap.com/> abgerufen

*Google Firebase*. (09 2023). Von <https://firebase.google.com/docs?hl=de> abgerufen

*H2-Database*. (2023). *H2 Database Engine*. Von <https://www.h2database.com/html/main.html> abgerufen

*javabeginners*. (26. 3 2023). Von javabeginners: [https://javabeginners.de/Design\\_Patterns/Model-View-Controller.php](https://javabeginners.de/Design_Patterns/Model-View-Controller.php) abgerufen

*javatpoint*. (26. 3 2023). Von javatpoint: <https://www.javatpoint.com/java-full-stack> abgerufen

*jqueryui*. (26. 3 2023). Von jqueryui: <https://jqueryui.com/> abgerufen

*jt.* (2023). *Spring Framework Guru*. Von <https://springframework.guru/spring-framework-annotations/> abgerufen

*kaalel*. (2023). *GeeksForGeeks*. Von <https://www.geeksforgeeks.org/spring-jdbc-template/> abgerufen

*mariadb.org*. (28. 3 2023). Von mariadb.org: <https://mariadb.org/> abgerufen

*Micronaut*. (2023). *Micronaut.io*. Von <https://micronaut.io/> abgerufen

*Microtool.de*. (02. 12 2022). Von <https://www.microtool.de/wissen-online/was-sind-use-cases/> abgerufen

*mindsquare.de*. (28. 3 2023). Von <https://mindsquare.de/knowhow/crud/#:~:text=Der%20Begriff%20CRUD%20kommt%20aus,Akronyme%20RUDI%20oder%20CDUR%20verwendet>. abgerufen

*mrknowing*. (26. 3 2023). Von mrknowing: <http://www.mrknowing.com/2013/11/08/wie-funktioniert-die-3-schichten-architektur/> abgerufen

*MyBatis*. (2023). *MyBatis*. Von <https://mybatis.org/mybatis-3/> abgerufen

*mysql.com*. (28. 3 2023). Von mysql.com: <https://www.mysql.com/de/> abgerufen

*Oracle*. (2023). *Java EE*. Von <https://www.oracle.com/java/technologies/java-ee-glance.html> abgerufen

*Quarkus*. (2023). *Quarkus.io*. Von <https://quarkus.io/about/> abgerufen

*react.dev*. (28. 3 2023). Von react.dev: <https://react.dev/> abgerufen

*redhat.com*. (28. 3 2023). Von redhat.com: [https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces#:~:text=APIs%20\(Application%20Programming%20Interfaces%20oder,zu%20kommenizieren%20und%20Daten%20auszutauschen](https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces#:~:text=APIs%20(Application%20Programming%20Interfaces%20oder,zu%20kommenizieren%20und%20Daten%20auszutauschen). abgerufen

*shiro.apache.org*. (28. 3 2023). Von shiro.apache.org: <https://shiro.apache.org/> abgerufen

*slf4j.org*. (kein Datum).

*SLF4J.org*. (08. 09 2023). Von SLF4J: <https://www.slf4j.org/> abgerufen

*snapsvg.io*. (28. 3 2023). Von snapsvg.io: <http://snapsvg.io/> abgerufen

*svgjs.dev*. (26. 3 2023). Von svgjs.dev: <https://svgjs.dev/docs/3.0/> abgerufen

Thai, N. N. (2023). *Baeldung*. Von <https://www.baeldung.com/spring-bean> abgerufen

*tyhmeleaf.org*. (26. 3 2023). Von tyhmeleaf.org: <https://www.thymeleaf.org/> abgerufen

Vaadin. (2023). *Vaadin*. Von <https://vaadin.com/docs/v8/framework/introduction/intro-overview#:~:text=Vaadin%20Framework%20is%20a%20Java,is%20the%20more%20powerful%20one>. abgerufen

Vats, R. (2023). *upGrad*. Von <https://www.upgrad.com/blog/spring-mvc-flow-diagram/> abgerufen

*velocity.apache.org*. (28. 3 2023). Von velocity.apache.org: <https://velocity.apache.org/> abgerufen

VMware. (2023). *Spring Framework*. Von <https://spring.io/projects/spring-framework> abgerufen

VMware. (2023). *Spring Security*. Von <https://spring.io/projects/spring-security> abgerufen

VMware. (2023). *spring.io*. Von <https://spring.io/guides/gs/convert-jar-to-war/> abgerufen

VMWare. (2023). *SpringBoot*. Von <https://spring.io/projects/spring-boot> abgerufen

*Wikipedia*. (02. 12 2022). Von Wikipedia: [https://de.wikipedia.org/wiki/Responsive\\_Webdesign](https://de.wikipedia.org/wiki/Responsive_Webdesign) abgerufen

*Wikipedia*. (02. 12 2022). Von Wikipedia: [https://de.wikipedia.org/wiki/Corporate\\_Design](https://de.wikipedia.org/wiki/Corporate_Design) abgerufen

*Wikipedia*. (23. Oktober 2022). *Wikipedia DE*. Von <https://de.wikipedia.org/wiki/Brainstorming> abgerufen

*Wikipedia*. (23. Oktober 2022). *Wikipedia DE*. Von [https://en.wikipedia.org/wiki/Long-term\\_support](https://en.wikipedia.org/wiki/Long-term_support) abgerufen

*Wikipedia*. (26. 3 2023). Von Wikipedia: [https://de.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol) abgerufen

*Wikipedia*. (26. 3 2023). Von Wikipedia: <https://de.wikipedia.org/wiki/JavaScript> abgerufen

*Wikipedia*. (26. 3 2023). Von Wikipedia: [https://de.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://de.wikipedia.org/wiki/Cascading_Style_Sheets) abgerufen

*Wikipedia*. (26. 3 2023). Von Wikipedia: [https://de.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](https://de.wikipedia.org/wiki/JavaScript_Object_Notation) abgerufen

*wikipedia.com*. (28. 3 2023). Von wikipedia.com: [https://de.wikipedia.org/wiki/Java\\_Authentication\\_and\\_Authorization\\_Service](https://de.wikipedia.org/wiki/Java_Authentication_and_Authorization_Service) abgerufen

*wikipedia.com*. (28. 3 2023). Von wikipedia.com: [https://en.wikipedia.org/wiki/Mustache\\_\(template\\_system\)](https://en.wikipedia.org/wiki/Mustache_(template_system)) abgerufen

*www.spring.io*. (2023). *www.spring.io*. Von <https://docs.spring.io/spring-framework/reference/>: <https://docs.spring.io/spring-framework/reference/> abgerufen

*www.thymeleaf.org*. (07 2023). *Thymeleaf Home*. Von www.thymeleaf.org: [www.thymeleaf.org](http://www.thymeleaf.org) abgerufen

# A Anhang

Im folgenden Abschnitt befinden sich ausgelagerte Abbildungen/Tabellen, welche aus Gründen der Leserlichkeit, an das Ende des Dokumentes verschoben wurden.

## A.1. Testfälle

In der nachstehenden Tabelle, **Fehler! Verweisquelle konnte nicht gefunden werden.**, wurde die GetAllDeskBooking Methode getestet. Zu beachten ist, dass das in den nachstehenden Absätzen ein anderes Template verwendet wurde, um den Test zu beschreiben. Die Methode der Testung bleibt jedoch gleich.

| TESTTITEL  | PRIORITÄT   | TESTFALL-ID   | TESTNUMMER   | TESTDATUM |
|--|---|---|--|-----------|
| GetAllDeskBooking  | High  | C1  | 1  | 16/05/23  |
| TESTBESCHREIBUNG   | TEST DESIGNED BY  | TEST AUSGEFÜHRT VON   | AUSFÜHRUNGSDATUM   |           |
| Alle vorhandene DeskBookings von der Datenbank hervorrufen.  | Sonja Lechner   | Sonja Lechner   | 16/05/23   |           |
| TESTBESCHREIBUNG   | ABHÄNGIGKEITEN TESTEN   | PRÜFBEDINGUNGEN   | PRÜFKONTROLLE  |           |
| Testen ob die Methode ,<br>getAllBookings()' erwartungsgemäß funktioniert. Beim Aufruf der Methode, soll eine Liste alle vorhandene DeskBookings zurück liefern. | Die Testmethode setzt voraus, dass die Datenbank Buchungen enthält. Es sollte sichergestellt werden, dass die Testdaten für die Buchungen korrekt vorbereitet sind, um genaue Ergebnisse zu erhalten. | Überprüfen Sie die Größe der zurückgegebenen Liste, um sicherzustellen, dass sie der Anzahl der vorhandenen Buchungen entspricht. Vergleichen Sie die einzelnen Buchungen in der zurückgegebenen Liste mit den erwarteten Buchungen, um sicherzustellen, dass alle relevanten Informationen korrekt sind. | <ul style="list-style-type: none"> <li>• Überprüfen Sie die Größe der zurückgegebenen Liste, um sicherzustellen, dass sie der Anzahl der vorhandenen Buchungen entspricht.</li> <li>• Vergleichen Sie die einzelnen Buchungen in der zurückgegebenen Liste mit den erwarteten Buchungen, um sicherzustellen, dass alle relevanten Informationen korrekt sind.</li> </ul> |           |

Tabelle 7 - Testfall GetAllDeskBookings

In der nachfolgenden Tabelle, Tabelle 8: Test-Schritte getAllBookings, wurden die durchgeführten Schritte, die zur Realisierung des Testfalles durchgeführt wurden, genau beschrieben sowie mit einem Zeitstempel versehen.

| SCHRI<br>T-ID | SCHRITBESCHREIBUNG   | TESTDATUM | ERWARTETE ERGEBNISSE   | TATSÄCHLICHE ERGEBNISSE   | BESTANDEN /<br>NICHT<br>BESTANDEN | ZUSÄTZL<br>ICHE<br>HINWEIS<br>E |
|---------------|--|-----------|--|---|-----------------------------------|---------------------------------|
| C1-1          | Arrange:<br>Vorhandene<br>Buchungen in der<br>Datenbank<br>vorbereiten.  | 16/05/23  |  |   |                                   |                                 |
| C1-2          | Act: Aufruf der<br>Methode<br>getAllBookings()<br>von<br>DeskBookingDBAc<br>cess_JPAH2.                          | 16/05/23  |  |   |                                   |                                 |
| C1-3          | Assert: Überprüfen,<br>ob die<br>zurückgegebene<br>Liste der<br>Buchungen der<br>erwarteten Liste<br>entspricht. | 16/05/23  | Die zurückgegebene Liste<br>enthält alle vorhandenen<br>Buchungen. | Die zurückgegebene<br>Liste entspricht den<br>erwarteten Buchungen. | Bestanden                         |                                 |

Tabelle 8: Test-Schritte getAllBookings

## TESTFALL BEISPIEL

Test: Nr C1

Name: getAllBookings()

Beschreibung: Überprüft, ob alle Buchungen erfolgreich abgerufen werden können.

Vorbedingungen: Vorhandene Buchungen in der Datenbank.

Tester: Sonja Lechner

Datum: 2023-05-20

Aktionen (Schritte:

- Arrange: Vorhandene Buchungen in der Datenbank vorbereiten.
- Act: Aufruf der Methode getAllBookings() von DeskBookingDBAccess\_JPAH2.
- Assert: Überprüfen, ob die zurückgegebene Liste der Buchungen der erwarteten Liste entspricht. - SOLL: Die zurückgegebene Liste enthält alle vorhandenen Buchungen.  
IST: Die zurückgegebene Liste entspricht den erwarteten Buchungen.

Kriterien für erfolgreiche Absolvierung: Die zurückgegebene Liste enthält alle vorhandenen Buchungen.

Es wurden weitere Testungen zu verschiedenen Funktionalitäten durchgeführt, die kurz beschrieben wurden:

testGetAllBookings(): Überprüft, ob die Methode getAllBookings() alle Buchungen zurückgibt, die in der Datenbank gespeichert sind.

testGetBookingByBookingId(Long bookingId): Überprüft, ob eine spezifische Buchung anhand ihrer ID aus der Datenbank abgerufen werden kann.

testGetBookingByDesk(Desk desk): Überprüft, ob eine Liste von Buchungen aus der Datenbank abgerufen werden kann, die für einen bestimmten Schreibtisch gemacht wurden.

testGetBookingsByEmployee(Employee employee): Überprüft, ob eine Liste von Buchungen aus der Datenbank abgerufen werden kann, die von einem bestimmten Mitarbeiter gemacht wurden.

testGetBookingsByEmployeeAndDate(Employee employee, LocalDate date): überprüft ob beim Eingabe der Desks und Employee alle aktuelle Buchungen von der Datenbank zurückgeliefert wird.

testUpdateBookingByBookingId(Long deskBookingId, DeskBooking updatedDeskBooking): überprüft ob ein aktualisiertes DeskBooking, richtig in der Datenbank gespeichert wird.

| TESTTITEL  | PRIORITÄT  | TESTFALL-ID  | TESTNUMMER  | TESTDATUM |
|--|--|--|---|-----------|
| updateEmployee   | High   | C2   | 2   | 16/05/23  |
| TESTBESCHREIBUNG   | TEST DESIGNED BY   | TEST AUSGEFÜHRT VON  | AUSFÜHRUNGSDATUM  |           |
| Den entsprechenden Employee aus der Datenbank holen.   | Marcel Schranz   | Marcel Schranz   | 16/05/23  |           |
| TESTBESCHREIBUNG   | ABHÄNGIGKEITEN TESTEN  | PRÜFBEDINGUNGEN  | PRÜFKONTROLLE   |           |
| Testen ob die Methode updateEmployee() den gewünschten Employee aus der Datenbank holt. Nach erfolgreicher Bearbeitung muss der Employee wieder in die Datenbank gespeichert werden. | Die Testmethode setzt voraus, dass die Datenbank Employees enthält. Ebenfalls sollte es nur dem eingeloggten User mit adminrechten möglich sein, Employees zu verändern. | Der anhand der ID überarbeitet Employee wird wieder in die Datenbank zurück gespeichert. | <ul style="list-style-type: none"> <li>Der bearbeitete Employee wurde ordnungsgemäß gespeichert.</li> </ul> |           |

Tabelle 9: Testfall updateEmployee

## TESTFALL BEISPIEL

Test: Nr C2

Name: updateEmployee()

Beschreibung: Überprüft, ob ein vorhandener Employee bearbeitet werden kann.

Vorbedingungen: Eingeloggter Benutzer mit administrator Rechten und vorhandene Employees in der Datenbank.

Tester: Marcel Schranz

Datum: 2023-05-20

Aktionen (Schritte:

- Arrange: User muss sich einloggen.
- Act: Aufruf der Methode updateEmployee().
- Assert: Überprüfen, ob der bearbeitete Employee korrekt gesichtet wurde. –  
SOLL: Der überarbeitete Employee wird zurückgegeben..

IST:

Kriterien für erfolgreiche Absolvierung: Der Employee wird korrekt aus der Datenbank gelesen..

## A.2. Zeitprotokoll

Im folgenden Abschnitt befinden sich die jeweiligen Zeitprotokolle der Teammitglieder.

## Zeitprotokoll - Patrick Bayr

| Datum      | Aufgabe/Beschreibung   | Zeit in h |
|------------|--|-----------|
| 13.09.2022 | Bearbeitung des Projektantrages  | 2,0       |
| 20.09.2022 | Bearbeitung des Projektantrages  | 2,0       |
| 27.10.2022 | Fertigstellung und Einreichen des Projektantrages                                    | 2,0       |
| 04.10.2022 | Beginn der Projektumfeldanalyse  | 2,0       |
| 11.10.2022 | Einführung in LaTex und weiterarbeiten and der Umfeldanalyse                         | 2,0       |
| 18.10.2022 | LaTex Theorie, weiterarbeiten an der Ummfeldanalyse                                  | 3,0       |
| 23.10.2022 | Weiterarbeiten an der Umfeldanalyse  | 1,0       |
| 24.10.2022 | Fertigstellung Umfeldanalyse   | 0,5       |
| 25.10.2022 | Einführung Anforderungs- und Problemanalyse + Einreichen der Umfeldanalyse           | 2,0       |
| 04.11.2022 | Erarbeitung von Ideen und Konzepten zum User Interface.                              | 4,5       |
| 08.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 15.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 22.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 24.11.2022 | Fertigstellung des Mockups   | 2,0       |
| 25.11.2022 | Projektbetreuer-Treffen  | 1,0       |
| 02.12.2022 | Team Besprechung, Überarbeitung des Dokuments  | 4,5       |
| 06.12.2022 | Beginn der Projektplanung  | 2,0       |
| 09.12.2022 | Team Besprechung Überarbeitung des Mockups, Vorbereitung für das Treffen mit Stigger | 2,0       |
| 13.12.2022 | Ausarbeitung des Projektstrukturplanes   | 2,0       |
| 20.12.2022 | Ausarbeitung des Projektablaufplanes   | 2,0       |
| 23.12.2022 | Treffen mit Projektpartner   | 4,0       |
| 10.01.2023 | Überarbeitung des PSP und des PAP + erstellung des Gant Charts                       | 2,0       |
| 17.01.2023 | Fertigstellung des PSP, PAP und des Gant Charts                                      | 2,0       |
| 20.01.2023 | Team Besprechung. Fixierung der Module, Ausarbeitung der Arbeitspakete               | 2,0       |
| 24.01.2023 | Ausfüllen des Statusberichtes  | 2,0       |
| 31.01.2023 | Beginn der Risikoanalyse   | 2,0       |
| 07.02.2023 | Fertigstellung der Risikoanalyse   | 2,0       |
| 14.02.2023 | Erstellung des Posters für den HTL-Infoabend   | 2,0       |
| 18.02.2023 | Überarbeitung der Methoden (löschen + hinzufügen)                                    | 3,0       |

|            |   |     |
|------------|---|-----|
| 21.02.2023 | Weiterarbeiten am Poster  | 2,0 |
| 27.02.2023 | Fertigstellung des Posters  | 4,0 |
| 03.03.2023 | Teilnahme am AbsolventInnen Abend   | 3,5 |
| 07.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 14.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 21.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 28.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 29.03.2023 | Treffen mit Projektpartner  | 5,0 |
| 04.04.2023 | Ausarbeitung der Produktbeschreibung  | 2,0 |
| 11.04.2023 | Ausarbeitung der Produktbeschreibung  | 2,0 |
| 18.04.2023 | Ausarbeitung der Produktbeschreibung  | 2,0 |
| 25.04.2023 | Fertigstellung der Produktbeschreibung  | 2,0 |
| 02.05.2023 | Ausarbeitung der Testfälle  | 2,0 |
| 09.05.2023 | Ausarbeitung der Testfälle  | 2,0 |
| 14.05.2023 | Pfadänderung der URL's innerhalb des Webcontrollers und der Templates   | 0,5 |
| 15.05.2023 | Fertigstellung der Testfälle  | 2,0 |
| 20.05.2023 | erneute Pfadänderung der URL's  | 1,0 |
| 21.05.2023 | Einarbeitung SVG.JS   | 5,0 |
| 23.05.2023 | Ausarbeitung der Evaluation + Vorbereitungen SVG  | 5,0 |
| 30.05.2023 | Ausarbeitung der Evaluation   | 2,0 |
| 05.06.2023 | Anpassung des Designs damit es Responsive ist.  | 3,0 |
| 06.06.2023 | Ausarbeitung der Evaluation + Ausbauen des Room Packages + Fehlerbehebung   | 6,0 |
| 08.06.2023 | Optische Anpassung + Tyhmeleaf Anbindung für Create und Read erstellt.  | 5,0 |
| 13.06.2023 | Ausarbeitung der Evaluation + Team Troubleshooting  | 4,0 |
| 18.06.2023 | Ausarbeitung des Front- und Back-Ends   | 3,0 |
| 19.06.2023 | Optimierung der Klassen   | 2,0 |
| 20.06.2023 | Code Optimierung  | 4,0 |
| 23.06.2023 | Create und Read Methoden fertig implementiert + Testung   | 2,0 |
| 24.06.2023 | Implementierung Update + Delete, mehrere Interfaces erstellt, Room Booking Modul implementiert, Testungen, Fehlerbehebungen | 8,0 |
| 25.06.2023 | Umarbeitung auf ImageMap<br>Dokumentation, Problemlösung Create und Update + weitere Fehlerbehebungen                       | 8,0 |
| 26.06.2023 | Umarbeitung auf ImageMap  | 9,5 |

|                             |  |              |
|-----------------------------|--|--------------|
| 27.06.2023                  | Anpassung der Templates, Auslagerung weiterer Code Duplikate,                | 7,0          |
| 28.06.2023                  | Fehlerbehebung, Optimierung, Dokumentation                                   | 5,0          |
| 29.06.2023                  | Weitere Optimierungen im Back-End, löschen nicht benötigter Methoden, Pakete | 5,5          |
| 29.08.2023                  | Beginn der Dokumentation des persönliches Teiles                             | 4,0          |
| 01.09.2023                  | Überarbeitung des persönliches Teils   | 3,0          |
| 09.09.2023                  | Kontrolle und Überarbeitung des persönliches Teils                           | 4,0          |
| 11.09.2023                  | Endgültige Finalisierung des Dokuments                                       | 6,0          |
| <b>Stundenausmaß Gesamt</b> |  | <b>202,5</b> |

## Zeitprotokoll - Sonja Lechner

| Datum      | Aufgabe                 | Beschreibung   | Zeit in h |
|------------|-------------------------|--|-----------|
| 13.09.2022 | Projektantrag           | Projektantrag Erstellen                                  | 2,0       |
| 20.09.2022 | Projektantrag           | Projektantrag Erstellen                                  | 2,0       |
| 27.09.2022 | Projektantrag           | Einreichen des Projektantrags                            | 2,0       |
| 04.10.2022 | Projektdokumentation    | Umfeldanalyse durchführung                               | 2,0       |
| 11.10.2022 | Projektdokumentation    | Latex Einfuhrung, UmfeldAnalyse                          | 2,0       |
| 18.10.2022 | Projektdokumentation    | Umfeldanalyse weiterbearbeiten                           | 2,0       |
| 24.10.2022 | Team Treffen            | Umfeldanalyse Besprechen                                 | 2,0       |
| 25.10.2022 | Projektdokumentation    | Abgabe der Umfeldanalyse, Anforderungsanalyse            | 2,0       |
| 04.11.2022 | Team Treffen            | Besprechung der User-Interface                           | 4,5       |
| 08.11.2022 | Team Treffen            | Mockups  | 4,0       |
| 15.11.2022 | Mockups                 | Rescherchieren welches Tool dafür am besten geeignet ist | 3,0       |
| 15.11.2022 | Mockups                 | Rescherchieren welches Tool dafür am besten geeignet ist | 0,5       |
| 16.11.2022 | Mockups                 | Einlesen, Erlernen von Adobe XD                          | 2,0       |
| 22.11.2022 | Mockups                 | Arbeiten am Mockup                                       | 3,0       |
| 23.11.2022 | Mockups                 | Mockup überarbeiten                                      | 3,0       |
| 24.11.2023 | Projektdokumentation    | Fertigstellung des Mockups                               | 2,0       |
| 02.12.2022 | Projektbetreuer Treffen | Besprechung des Projekts                                 | 1,0       |
| 06.12.2022 | Team Treffen            | Projektplanung   | 2,0       |
| 09.12.2022 | Team Besprechung        | Vorbereitung fur das Treffen mit dem Projekt Partner     | 2,0       |
| 10.12.2022 | Projektdokumentation    | Anderungen am Mockup                                     | 2,0       |
| 13.12.2022 | Projektdokumentation    | Entwicklung der Projektstrukturplan                      | 2,0       |
| 20.12.2022 | Projektdokumentation    | Arbeiten am Projektablaufplan                            | 2,0       |
| 23.12.2022 | Projektdokumentation    | Projekt Treffen mit dem Projektpartner                   | 2,0       |

|            |                      |   |     |
|------------|----------------------|---|-----|
| 10.01.2023 | Projektdokumentation | Arbeiten am Projektstrukturplan, Projektlaufplan und Gantt-Chart                      | 2,0 |
| 17.01.2023 | Projektdokumentation | Arbeiten am Projektstrukturplan, Projektlaufplan und Gantt-Chart                      | 2,0 |
| 24.01.2023 | Projektdokumentation | Statusbericht   | 2,0 |
| 31.01.2023 | Projektdokumentation | Risikoanalyse   | 2,0 |
| 07.02.2023 | Projektdokumentation | Risikoanalyse   | 2,0 |
| 14.02.2023 | Projekt Poster       | WABS Poster   | 2,0 |
| 17.02.2023 | WABS - Projektarbeit | Neu Aufstellung des Projekts in Github  | 1,0 |
| 21.02.2023 | Projekt Poster       | WABS Poster   | 2,0 |
| 23.02.2023 | WABS - Projektarbeit | Domainschicht - Desk, Booking, Port   | 3,0 |
| 27.02.2023 | WABS - Projektarbeit | Domain-, Repo-en, ServiceSchicht - Desk, Booking, Port                                | 5,0 |
| 28.02.2023 | Projekt Poster       | WABS Poster   | 2,0 |
| 01.03.2023 | WABS - Projektarbeit | Domainschicht - Desk, Booking   | 3,0 |
| 07.03.2023 | Absolventinnenabend  | Teilnahme am Absolventinnenabend  | 2,0 |
| 12.03.2023 | WABS - Projektarbeit | Repo and Service Layer  | 2,5 |
| 14.03.2023 | Projektdokumentation | Systemdokumentation   | 2,0 |
| 19.03.2023 | WABS - Projektarbeit | Methode Erstellen   | 1,5 |
| 21.03.2023 | Projektdokumentation | Systemdokumentation   | 2,0 |
| 28.03.2023 | Projektdokumentation | Systemdokumentation   | 2,0 |
| 29.03.2023 | Projekt Treffen      | Projekt Treffen mit dem Projektpartner & Projektbetreuern                             | 5,0 |
| 04.04.2023 | Projektdokumentation | Produktbeschreibung   | 2,0 |
| 08.04.2023 | WABS - Projektarbeit | Klassen überarbeiten, Änderungen der Attribute, Methode, Neue Klassen, Fehlerbehebung | 5,0 |
| 11.04.2023 | Projektdokumentation | Produktbeschreibung   | 2,0 |
| 16.04.2023 | WABS - Projektarbeit | Implementierung der Methode, Rest API zum Testen                                      | 3,0 |
| 18.04.2023 | Projektdokumentation | Produktbeschreibung   | 2,0 |

|            |                      |  |     |
|------------|----------------------|--|-----|
| 21.04.2023 | WABS - Projektarbeit | Implementierung der Methode, Rest API zum Testen       | 3,5 |
| 22.04.2023 | WABS - Projektarbeit | Implementierung der Methode, Rest API zum Testen       | 1,5 |
| 25.04.2023 | Projektdokumentation | Produktbeschreibung                                    | 2,0 |
| 02.05.2023 | Projektdokumentation | Ausarbeiten der Testfälle                              | 2,0 |
| 06.05.2023 | WABS - Projektarbeit | Repo and Service Layer                                 | 3,5 |
| 07.05.2023 | WABS - Projektarbeit | Repo and Service Layer                                 | 4,5 |
| 09.05.2023 | Projektdokumentation | Ausarbeiten der Testfälle                              | 2,0 |
| 16.05.2023 | Projektdokumentation | Ausarbeiten der Evaluation                             | 2,0 |
| 27.05.2023 | WABS - Projektarbeit | API & Templates  | 5,0 |
| 28.05.2023 | WABS - Projektarbeit | API & Templates  | 4,0 |
| 21.05.2023 | Team Besprechung     | Aktualisierung des WABS in github, Bug Fixes           | 3,0 |
| 23.05.2023 | Projektdokumentation | Ausarbeiten der Evaluation                             | 2,0 |
| 24.05.2023 | WABS - Projektarbeit | Fehlerbehebung Deskbooking Module                      | 3,0 |
| 25.05.2023 | WABS - Projektarbeit | addBooking Fehlerbehebung, Holiday Module implemtation | 4,0 |
| 30.05.2023 | Projektdokumentation | Ausarbeiten der Evaluation                             | 2,0 |
| 30.05.2023 | WABS - Projektarbeit | Templates  | 3,0 |
| 02.06.2023 | WABS - Projektarbeit | Exception Handling, Kommentieren, Templates            | 1,5 |
| 03.06.2023 | WABS - Projektarbeit | Exception Handling, Kommentieren, Templates            | 4,5 |
| 04.06.2023 | WABS - Projektarbeit | Fehlerbehebung   | 3,0 |
| 06.06.2023 | Projektdokumentation | Ausarbeiten der Evaluation                             | 2,0 |
| 09.06.2023 | WABS - Projektarbeit | Code Optimierung und Fehlerbehebungen                  | 5,0 |
| 10.06.2023 | WABS - Projektarbeit | Code Optimierung und Fehlerbehebungen                  | 3,5 |
| 13.06.2023 | Projektdokumentation | Ausarbeiten der Evaluation                             | 2,0 |
| 17.06.2023 | WABS - Projektarbeit | Github Merge Conflict Resolution                       | 2,5 |
| 20.06.2023 | WABS - Projektarbeit | Code Optimierung und Fehlerbehebungen                  | 4,5 |
| 20.06.2023 | Projektdokumentation | Posters und Code Optimierung                           | 2,0 |

|                       |                      |                                       |              |
|-----------------------|----------------------|---------------------------------------|--------------|
| 25.06.2023            | WABS - Projektarbeit | Code Optimierung und Fehlerbehebungen | 7,0          |
| 26.06.2023            | WABS - Projektarbeit | Code Optimierung und Fehlerbehebungen | 6,5          |
| 27.06.2023            | WABS - Projektarbeit | Fehlerbehebungen                      | 4,5          |
| 28.06.2023            | WABS - Projektarbeit | Code Kommentieren und Optimieren      | 5,5          |
| 29.06.2023            | WABS - Projektarbeit | Produkt Finalisierung                 | 4,5          |
| 30.06.2023            | WABS - Projektarbeit | Produkt Finalisierung                 | 3,0          |
| 03.07.2023            | WABS - Projektarbeit | Produkt Finalisierung                 | 4,0          |
| 01.07.2023            | WABS - Diplomarbeit  | Persönliche Teil Erstellen            | 2,0          |
| 03.07.2023            | WABS - Diplomarbeit  | Persönliche Teil Erstellen            | 4,0          |
| 22.07.2023            | WABS - Diplomarbeit  | Persönliche Teil Erstellen            | 3,0          |
| 28.07.2023            | WABS - Diplomarbeit  | Persönliche Teil Erstellen            | 0,5          |
| 12.08.2023            | WABS - Diplomarbeit  | Persönliche Teil Bearbeiten           | 2,0          |
| 13.08.2023            | WABS - Diplomarbeit  | Persönliche Teil Bearbeiten           | 1,0          |
| 25.08.2023            | WABS - Diplomarbeit  | Persönliche Teil Bearbeiten           | 3,5          |
| 09.09.2023            | WABS - Diplomarbeit  | Persönliche Teil Überarbeiten         | 5,0          |
| 10.09.2023            | WABS - Diplomarbeit  | Persönliche Teil Überarbeiten         | 2,0          |
| 11.09.2023            | WABS - Diplomarbeit  | Dokument Finalisieren                 | 4,0          |
| <b>Gesamt Stunden</b> |                      |                                       | <b>248,0</b> |

## Zeitprotokoll - Manuel Payer

| Datum      | Aufgabe/Beschreibung   | Zeit in h |
|------------|--|-----------|
| 13.09.2022 | Bearbeitung des Projektantrages  | 2,0       |
| 20.09.2022 | Bearbeitung des Projektantrages  | 2,0       |
| 27.10.2022 | Fertigstellung und Einreichen des Projektantrages                                    | 2,0       |
| 04.10.2022 | Beginn der Projektumfeldanalyse  | 2,0       |
| 11.10.2022 | Einführung in LaTex und weiterarbeiten and der Umfeldanalyse                         | 2,0       |
| 18.10.2022 | LaTex Theorie, weiterarbeiten an der Ummfeldanalyse                                  | 3,0       |
| 23.10.2022 | Weiterarbeiten an der Umfeldanalyse  | 1,0       |
| 24.10.2022 | Fertigstellung Umfeldanalyse   | 0,5       |
| 25.10.2022 | Einführung Anforderungs- und Problemanalyse + Einreichen der Umfeldanalyse           | 2,0       |
| 04.11.2022 | Erarbeitung von Ideen und Konzepten zum User Interface.                              | 4,5       |
| 08.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 15.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 22.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 24.11.2022 | Fertigstellung des Mockups   | 2,0       |
| 25.11.2022 | Projektbetreuer-Treffen  | 1,0       |
| 02.12.2022 | Team Besprechung, Überarbeitung des Dokuments  | 4,5       |
| 06.12.2022 | Beginn der Projektplanung  | 2,0       |
| 09.12.2022 | Team Besprechung Überarbeitung des Mockups, Vorbereitung für das Treffen mit Stigger | 2,0       |
| 13.12.2022 | Ausarbeitung des Projektstrukturplanes   | 2,0       |
| 20.12.2022 | Ausarbeitung des Projektablaufplanes   | 2,0       |
| 23.12.2022 | Treffen mit Projektpartner   | 4,0       |
| 10.01.2023 | Überarbeitung des PSP und des PAP + erstellung des Gant Charts                       | 2,0       |
| 17.01.2023 | Fertigstellung des PSP, PAP und des Gant Charts                                      | 2,0       |
| 20.01.2023 | Team Besprechung. Fixierung der Module, Ausarbeitung der Arbeitspakete               | 2,0       |
| 24.01.2023 | Ausfüllen des Statusberichtes  | 2,0       |
| 31.01.2023 | Beginn der Risikoanalyse   | 2,0       |
| 07.02.2023 | Fertigstellung der Risikoanalyse   | 2,0       |
| 14.02.2023 | Erstellung des Posters für den HTL-Infoabend   | 2,0       |
| 18.02.2023 | Überarbeitung der Methoden (löschen + hinzufügen)                                    | 3,0       |
| 21.02.2023 | Weiterarbeiten am Poster   | 2,0       |

|            |   |     |
|------------|---|-----|
| 27.02.2023 | Fertigstellung des Posters  | 4,0 |
| 03.03.2023 | Teilnahme am AbsolventInnen Abend   | 3,5 |
| 07.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 14.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 21.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 28.03.2023 | Ausarbeitung der Systemdokumentation  | 2,0 |
| 29.03.2023 | Treffen mit Projektpartner  | 5,0 |
| 04.04.2023 | Ausarbeitung der Produktbeschreibung  | 2,0 |
| 11.04.2023 | Ausarbeitung der Produktbeschreibung  | 2,0 |
| 18.04.2023 | Ausarbeitung der Produktbeschreibung  | 2,0 |
| 25.04.2023 | Fertigstellung der Produktbeschreibung  | 2,0 |
| 02.05.2023 | Ausarbeitung der Testfälle  | 2,0 |
| 09.05.2023 | Ausarbeitung der Testfälle  | 2,0 |
| 14.05.2023 | Pfadänderung der URL's innerhalb des Webcontrollers und der Templates   | 0,5 |
| 15.05.2023 | Fertigstellung der Testfälle  | 2,0 |
| 20.05.2023 | erneute Pfadänderung der URL's  | 1,0 |
| 21.05.2023 | Erstellung der Navbar + Java Script Funktionen für Dropdown Funktionalität eingefügt.   | 5,0 |
| 23.05.2023 | Ausarbeitung der Evaluation + Verlinkungen hinzugefügt + Fehlerbehebung Front-End   | 5,0 |
| 30.05.2023 | Ausarbeitung der Evaluation   | 2,0 |
| 05.06.2023 | Anpassung des Designs damit es Responsive ist.  | 3,0 |
| 06.06.2023 | Ausarbeitung der Evaluation + Ausbauen des Ressourcen Packages + Fehlerbeherbung  | 6,0 |
| 08.06.2023 | Optische Anpassung + Tyhmeleaf Anbindung für Create und Read erstellt.  | 5,0 |
| 13.06.2023 | Ausarbeitung der Evaluation + Team Troubleshooting  | 4,0 |
| 18.06.2023 | Ausarbeitung des Front- und Back-Ends   | 3,0 |
| 19.06.2023 | Optimierung der Klassen   | 2,0 |
| 20.06.2023 | Überarbeitung des Posters + Code Optimierung  | 4,0 |
| 21.06.2023 | Kleine Anpassungen des Frontends  | 0,5 |
| 23.06.2023 | Create und Read Methoden fertig implementiert + Testung   | 2,0 |
| 24.06.2023 | Implementierung Update + Delete, mehrere Interfaces erstellt, Ressource Booking Modul<br>implementiert, Testungen, Fehlerbehebungen                           | 8,0 |
| 25.06.2023 | Den Webcontroller weitere Methoden hinzugefügt, CSS weiter ausgebaut, Beginn der<br>Dokumentation, Problemlösung Create und Update + weitere Fehlerbehebungen | 8,0 |
| 26.06.2023 | Auslagerung von Codeduplikaten, Fehlerbehebung Update Ressourcebooking, Mapping   | 9,5 |

|            |  |              |
|------------|--|--------------|
|            | überarbeitet, Überarbeitung der Templates;   |              |
| 27.06.2023 | Anpassung der Templates, Auslagerung weiterer Code Duplikate,  | 7,0          |
| 28.06.2023 | Fehlerbehebung, Optimierung, Dokumentation   | 5,0          |
| 29.06.2023 | Weitere Optimierungen im Back-End, löschen nicht benötigter Methoden, Pakete   | 5,5          |
| 30.06.2023 | Code Verschönerung, Überarbeitung des CSS, Scrollfunktion hinzugefügt,<br>Code in Hinblick auf Wartbarkeit überarbeitet, Grundlegendes Exceptionhandling | 8,5          |
|            | Anpassung der Templates, Finalisierung.  |              |
| 01.07.2023 | Überarbeitung des Diplom-Dokuments   | 3,0          |
| 10.07.2023 | Beginn der Dokumentation des persönliches Teiles   | 4,0          |
| 01.08.2023 | Dokumentation des persönlichen Teils   | 5,0          |
| 15.08.2023 | Dokumentation des persönlichen Teils   | 5,0          |
| 20.08.2023 | Dokumentation des persönlichen Teils   | 6,0          |
| 01.09.2023 | Überarbeitung des persönliches Teils   | 3,0          |
| 09.09.2023 | Kontrolle und Überarbeitung des persönliches Teils   | 4,0          |
| 10.09.2023 | Finalisierung des Dokuments, Quellen, Abbildungen, geprüft, Formatierung angepasst   | 3,0          |
| 11.09.2023 | Entgültige Finalisierung des Dokuments   | 6,0          |
|            | <b>Stundenausmaß Gesamt</b>  | <b>233,5</b> |

## Zeitprotokoll - Marcel Schranz

| Datum      | Aufgabe/Beschreibung   | Zeit in h |
|------------|--|-----------|
| 13.09.2022 | Bearbeitung des Projektantrages  | 2,0       |
| 20.09.2022 | Bearbeitung des Projektantrages  | 2,0       |
| 27.10.2022 | Fertigstellung und Einreichen des Projektantrages                                    | 2,0       |
| 04.10.2022 | Beginn der Projektumfeldanalyse  | 2,0       |
| 11.10.2022 | Einführung in LaTex und weiterarbeiten and der Umfeldanalyse                         | 2,0       |
| 18.10.2022 | LaTex Theorie, weiterarbeiten an der Ummfeldanalyse                                  | 3,0       |
| 23.10.2022 | Weiterarbeiten an der Umfeldanalyse  | 1,0       |
| 24.10.2022 | Fertigstellung Umfeldanalyse   | 0,5       |
| 25.10.2022 | Einführung Anforderungs- und Problemanalyse + Einreichen der Umfeldanalyse           | 2,0       |
| 04.11.2022 | Erarbeitung von Ideen und Konzepten zum User Interface.                              | 4,5       |
| 08.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 15.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 22.11.2022 | Erarbeitung eines Mockups  | 2,0       |
| 24.11.2022 | Fertigstellung des Mockups   | 2,0       |
| 25.11.2022 | Projektbetreuer-Treffen  | 1,0       |
| 02.12.2022 | Team Besprechung, Überarbeitung des Dokuments  | 4,5       |
| 06.12.2022 | Beginn der Projektplanung  | 2,0       |
| 09.12.2022 | Team Besprechung Überarbeitung des Mockups, Vorbereitung für das Treffen mit Stigger | 2,0       |
| 13.12.2022 | Ausarbeitung des Projektstrukturplanes   | 2,0       |
| 20.12.2022 | Ausarbeitung des Projektablaufplanes   | 2,0       |
| 23.12.2022 | Treffen mit Projektpartner   | 4,0       |
| 10.01.2023 | Überarbeitung des PSP und des PAP + erstellung des Gant Charts                       | 2,0       |
| 17.01.2023 | Fertigstellung des PSP, PAP und des Gant Charts                                      | 2,0       |
| 20.01.2023 | Team Besprechung. Fixierung der Module, Ausarbeitung der Arbeitspakete               | 2,0       |
| 24.01.2023 | Ausfüllen des Statusberichtes  | 2,0       |
| 31.01.2023 | Beginn der Risikoanalyse   | 2,0       |
| 07.02.2023 | Fertigstellung der Risikoanalyse   | 2,0       |
| 12.02.2023 | Team Besprechung. Java Module  | 1,0       |
| 13.02.2023 | Firebase Datenbankeinrichtung/Datenbankanbindung                                     | 10,0      |
| 14.02.2023 | Erstellung Poster HTL-Infoabend/Spring Boot CRUD Implementierung                     | 5,0       |

|            |  |     |
|------------|--|-----|
| 15.02.2023 | Employee Klasse + Exceptions Firebase                  | 5,0 |
| 21.02.2023 | Weiterarbeiten am Poster                               | 2,0 |
| 22.02.2023 | Room Klasse + Exceptions Firebase                      | 3,0 |
| 23.02.2023 | Ressource Klasse + Exceptions Firebase                 | 3,0 |
| 27.02.2023 | Fertigstellung des Posters                             | 2,0 |
| 03.03.2023 | Teilnahme am AbsolventInnen Abend                      | 3,0 |
| 07.03.2023 | Ausarbeitung der Systemdokumentation                   | 2,0 |
| 12.03.2023 | Umstieg von Firebase zu H2                             | 3,0 |
| 13.03.2023 | Employee Klasse + Exceptions JPA/H2                    | 2,0 |
| 14.03.2023 | Ausarbeitung der Systemdokumentation                   | 2,0 |
| 21.03.2023 | Ausarbeitung der Systemdokumentation                   | 2,0 |
| 25.03.2023 | Spring Security  | 1,0 |
| 26.03.2023 | Implementierung Employee-Controller                    | 2,0 |
| 28.03.2023 | Ausarbeitung der Systemdokumentation                   | 2,0 |
| 29.03.2023 | Treffen mit Projektpartner                             | 5,0 |
| 04.04.2023 | Ausarbeitung der Produktbeschreibung                   | 2,0 |
| 11.04.2023 | Ausarbeitung der Produktbeschreibung                   | 2,0 |
| 18.04.2023 | Ausarbeitung der Produktbeschreibung                   | 2,0 |
| 25.04.2023 | Fertigstellung der Produktbeschreibung                 | 2,0 |
| 30.04.2023 | Spring Security Authentifizierung                      | 3,0 |
| 01.05.2023 | Spring Security Konfiguration, Login + Weiterleitungen | 8,0 |
| 02.05.2023 | Ausarbeitung der Testfälle                             | 2,0 |
| 07.05.2023 | Spring Security Full Tutorial                          | 2,0 |
| 09.05.2023 | Ausarbeitung der Testfälle                             | 2,0 |
| 13.05.2023 | Spring Boot Zusammenführung Module                     | 4,0 |
| 15.05.2023 | Fertigstellung der Testfälle                           | 2,0 |
| 21.05.2023 | Employee Modul Frontend                                | 4,0 |
| 22.05.2023 | Employee Modul Frontend                                | 6,0 |
| 23.05.2023 | Ausarbeitung der Evaluation                            | 2,0 |
| 30.05.2023 | Ausarbeitung der Evaluation                            | 2,0 |
| 06.06.2023 | Ausarbeitung der Evaluation                            | 6,0 |
| 16.06.2023 | Ausarbeitung der Evaluation                            | 2,0 |
| 19.06.2023 | Optimierung Employee Modul                             | 4,0 |

|            |   |              |
|------------|---|--------------|
| 23.06.2023 | Anpassung der Spring Security Config            | 2,0          |
| 24.06.2023 | Optimierung Employee Modul                      | 3,0          |
| 28.08.2023 | Beginn der Dokumentation des persönlichen Teils | 5,0          |
| 01.09.2023 | Überarbeitung des persönlichen Teils            | 4,0          |
| 10.09.2023 | Finalisierung des Dokuments                     | 3,0          |
| 11.09.2023 | Endgültige Finalisierung des Dokuments          | 6,0          |
|            | <b>Stundenausmaß Gesamt</b>                     | <b>193,5</b> |