Karlsruher Institut für Technologie

HCI ACCESSIBILITY

Kolloquium Entwurfsheft zur Solidarischen Raumnutzung

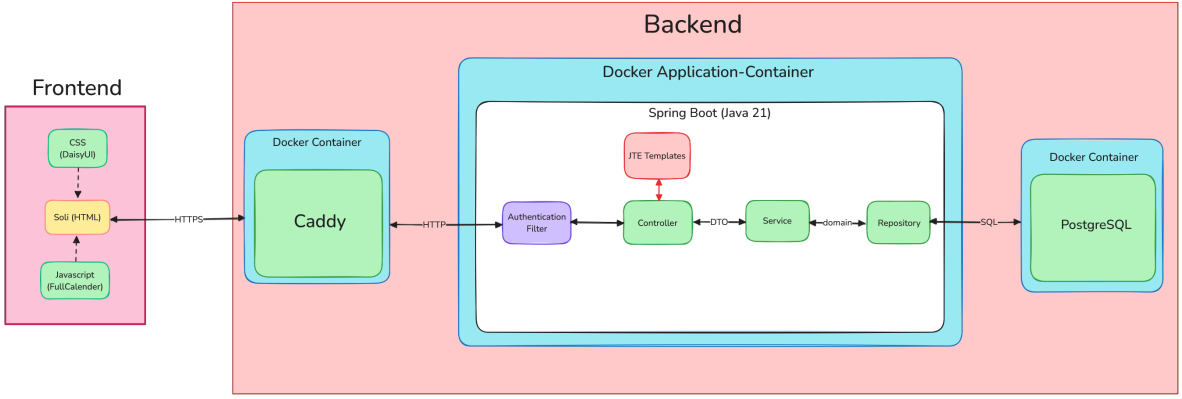
PSE-Projekt WS24/25

Alexander Klee, Jannik Hönlinger, Johannes Frohnmeyer, Ben Steinle und Antonia Ammon | 03. Januar 2025

Gliederung

1. Aufbau
   - Architektur
   - Klassendiagramm
2. View
   - Ansichten
3. Controller
   - Controller-Übersicht
     - Beispiel: BookingViewController
4. Services
   - Service-Übersicht
5. Daten
6. Data Model
7. Q&A

Architektur



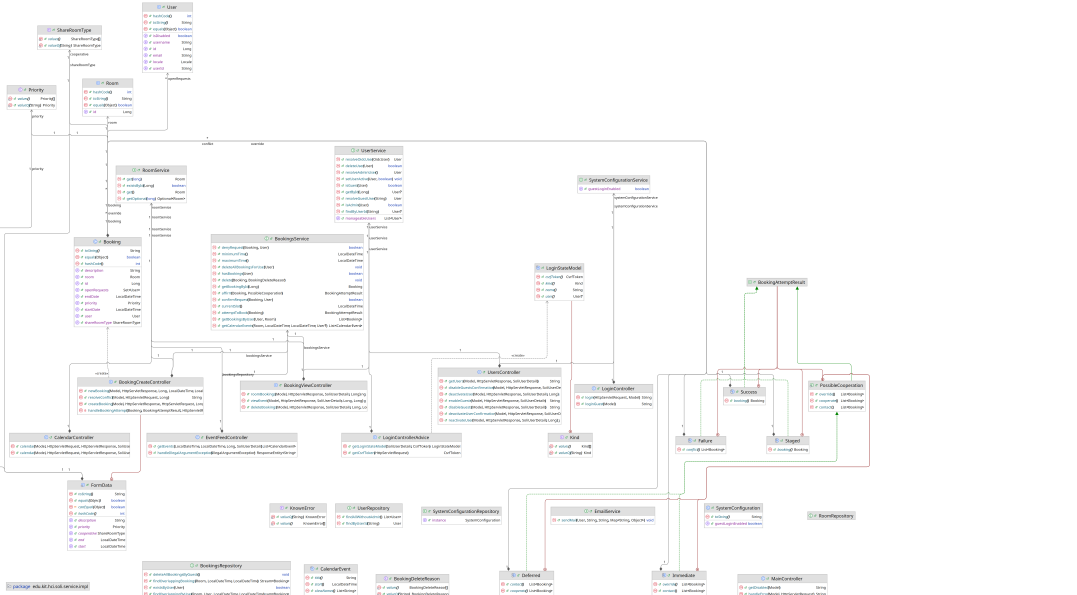
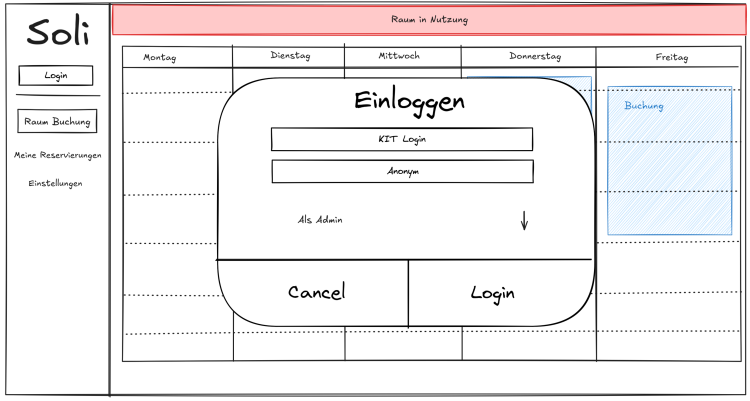


Abbildung: Bei Bedarf am Ende als einzelnes PDF danach fragen

View
Ansichten

- **Loginansicht**: Der/die Nutzende muss die Anmeldedaten eingeben, um auf ein Großteil der Funktionalität zuzugreifen.
- **Kalenderansicht**: Zeigt den Kalender mit den Terminen und Admins können hier der Öffnungszeiten einstellen.
- **Termin-Erstellen-Ansicht**: Mit ihr kann der/die Nutzende einen Termin mit Zeitraum, Priorität, und optionaler Beschreibung, sowie eine Angabe zur Bereitschaft zur geteilten Raumnutzung erstellen.
- **Terminansicht**: Zeigt die Informationen des Termins, sowie die Möglichkeit zur Anfrage einer geteilten Raumnutzung oder für den Eigentümer und Admin, die Möglichkeit des Termin-Löschens.
- **Terminübersicht**: Zeigt aufgelistet alle gebuchten Termine des Nutzenden, sowie die Möglichkeit Termine zu löschen und anzusehen.
- **Kontenliste**: Nur für Admins zugänglich und bietet die Möglichkeit spezifische Nutzende zu sperren oder die Gästefunktionalität auszuschalten.

Anmeldungsansicht



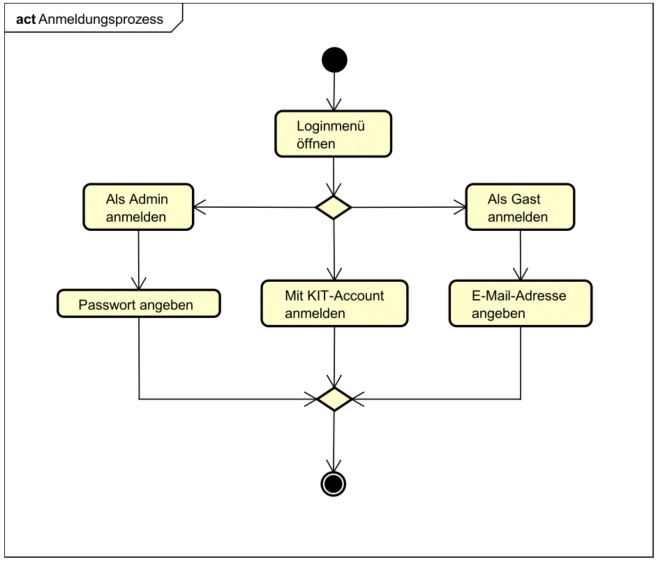


Abbildung: Mit der Anmeldung akzeptiert nun der Authentication Filter und der Großteil der Funktionalität ist nun zugänglich.

Kalenderansicht

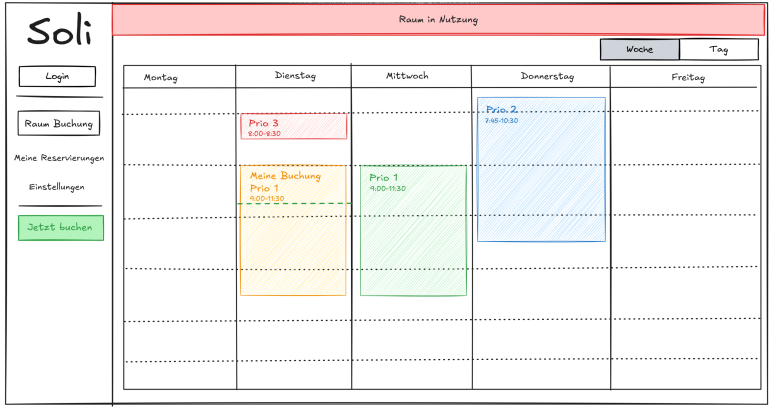


Abbildung: Zeigt alle Termine an, die in der Datenbank gespeichert sind.

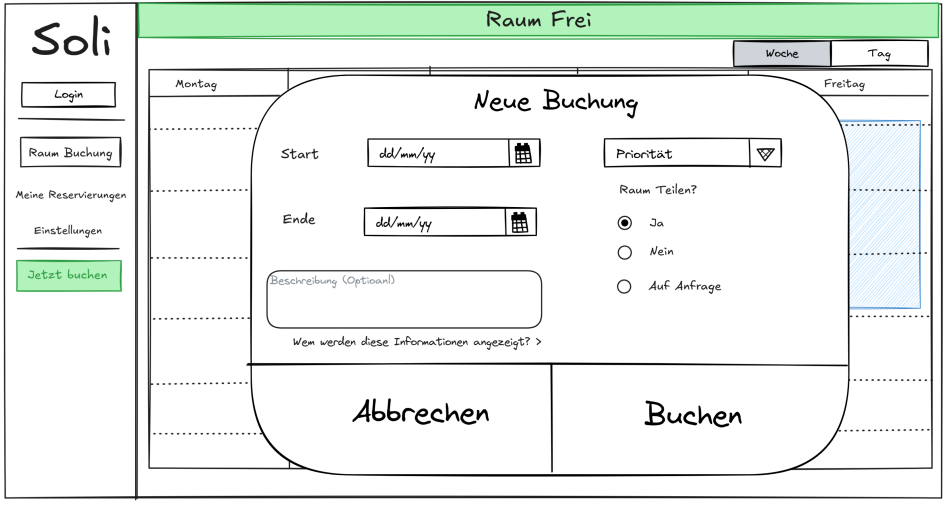


Abbildung: Diese Ansicht wird über die Kalenderansicht erreicht.

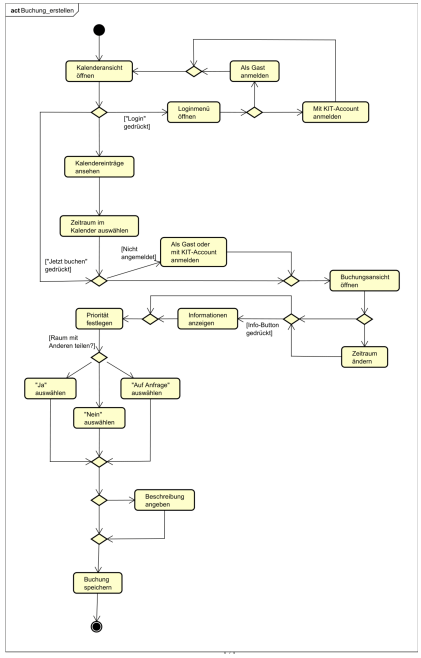
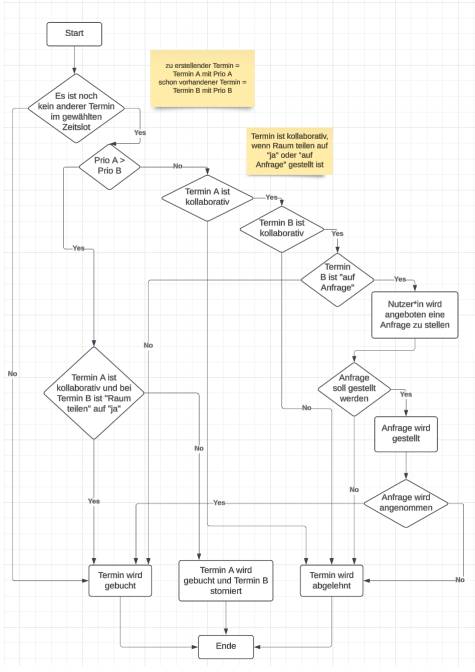
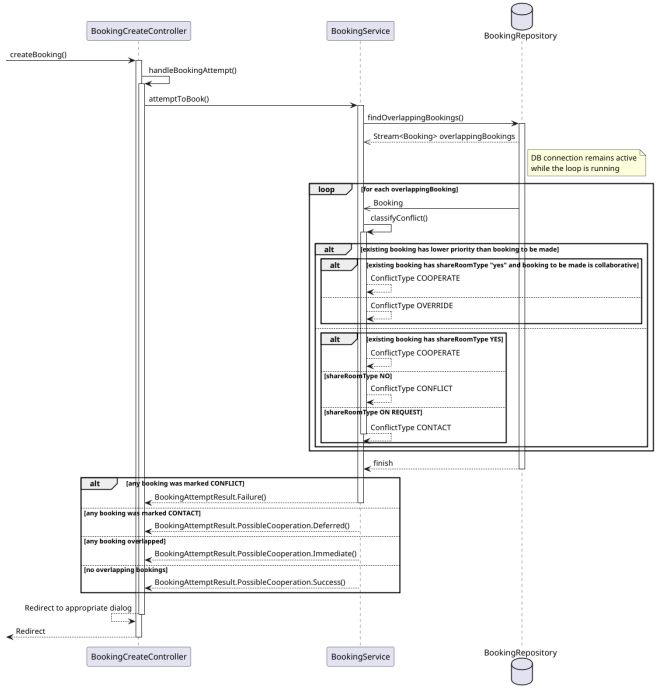


Abbildung: Der Termin wird in der Datenbank gespeichert und ist nun in der Kalenderansicht sichtbar.

Terminkonfliktlösung:




BookingCreateController
BookingService
BookingRepository
createBooking()
handleBookingAttempt()
attemptToBook()
findOverlappingBookings()
Stream<Booking> overlappingBookings
DB connection remains active while the loop is running
loop
[for each overlappingBooking]
Booking
classifyConflict()
alt
[existing booking has lower priority than booking to be made]
alt
[existing booking has shareRoomType "yes" and booking to be made is collaborative]
ConflictType COOPERATE
ConflictType OVERRIDE
alt
[existing booking has shareRoomType YES]
ConflictType COOPERATE
[shareRoomType NO]
ConflictType CONFLICT
[shareRoomType ON REQUEST]
ConflictType CONTACT
finish
alt
[any booking was marked CONFLICT]
BookingAttemptResult.Failure()
[any booking was marked CONTACT]
BookingAttemptResult.PossibleCooperation.Deferred()
[any booking overlapped]
BookingAttemptResult.PossibleCooperation.Immediate()
[no overlapping bookings]
BookingAttemptResult.PossibleCooperation.Success()
Redirect to appropriate dialog
Redirect
BookingCreateController
BookingService
BookingRepository

Terminübersicht

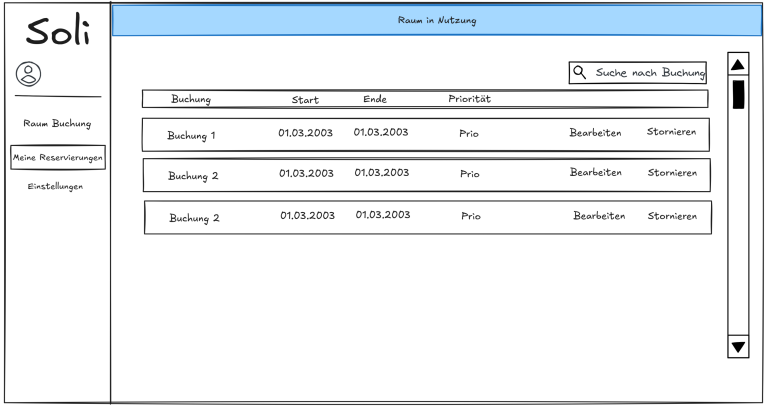
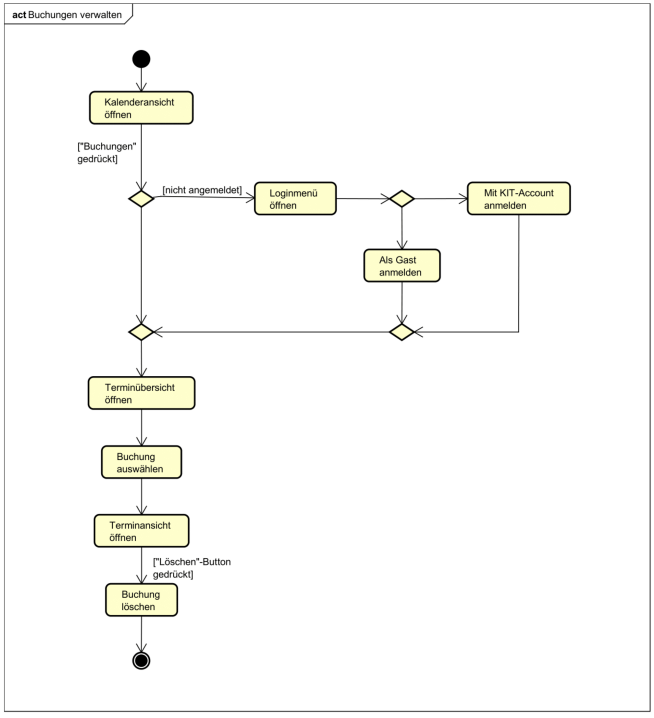


Abbildung: Zeigt alle Termine an, die der/die Nutzende erstellt hat.


act Buchungen verwalten
Kalenderansicht öffnen
["Buchungen" gedrückt]
[nicht angemeldet]
Loginmenü öffnen
Mit KIT-Account anmelden
Als Gast anmelden
Terminübersicht öffnen
Buchung auswählen
Terminansicht öffnen
["Löschen"-Button gedrückt]
Buchung löschen

Terminansicht

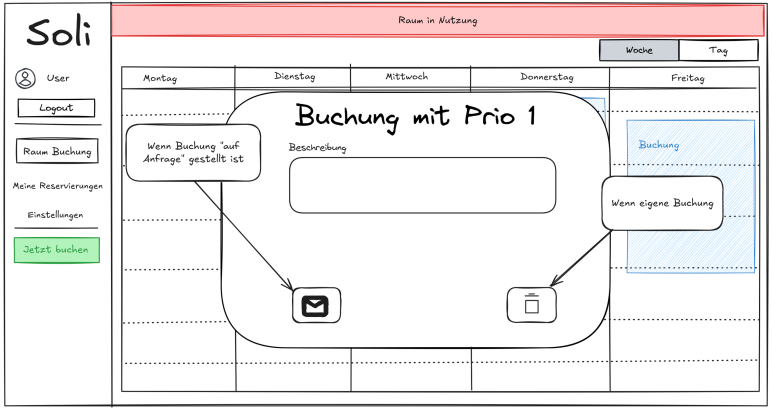
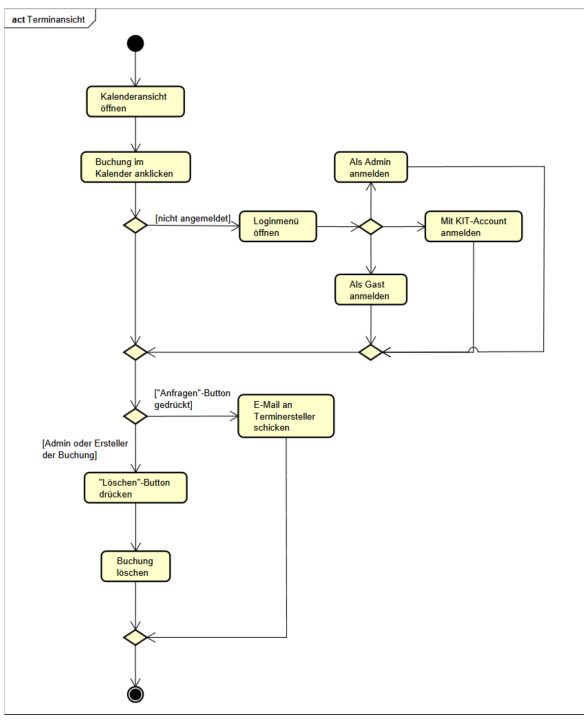
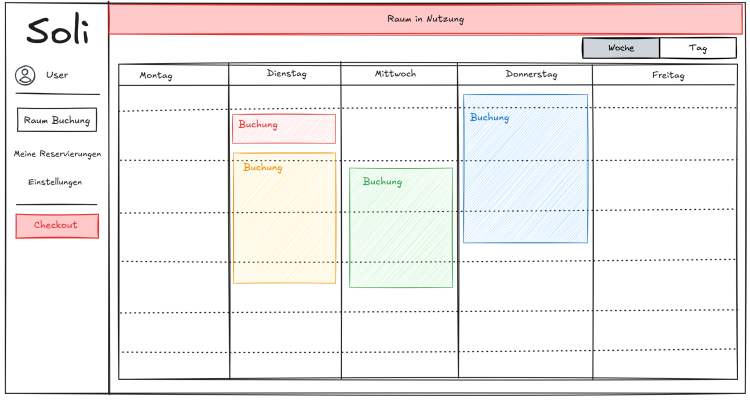
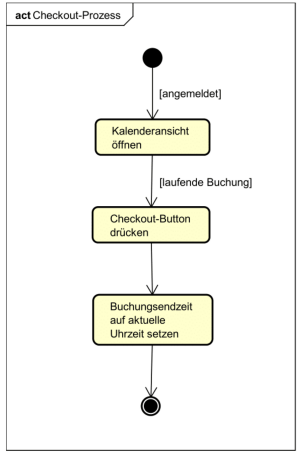


Abbildung: Zeigt alle Informationen des Termins an.


act Terminansicht
Kalenderansicht öffnen
Buchung im Kalender anklicken
[nicht angemeldet]
Loginmenü öffnen
Als Admin anmelden
Mit KIT-Account anmelden
Als Gast anmelden
["Anfragen"-Button gedrückt]
E-Mail an Terminersteller schicken
[Admin oder Ersteller der Buchung]
"Löschen"-Button drücken
Buchung löschen

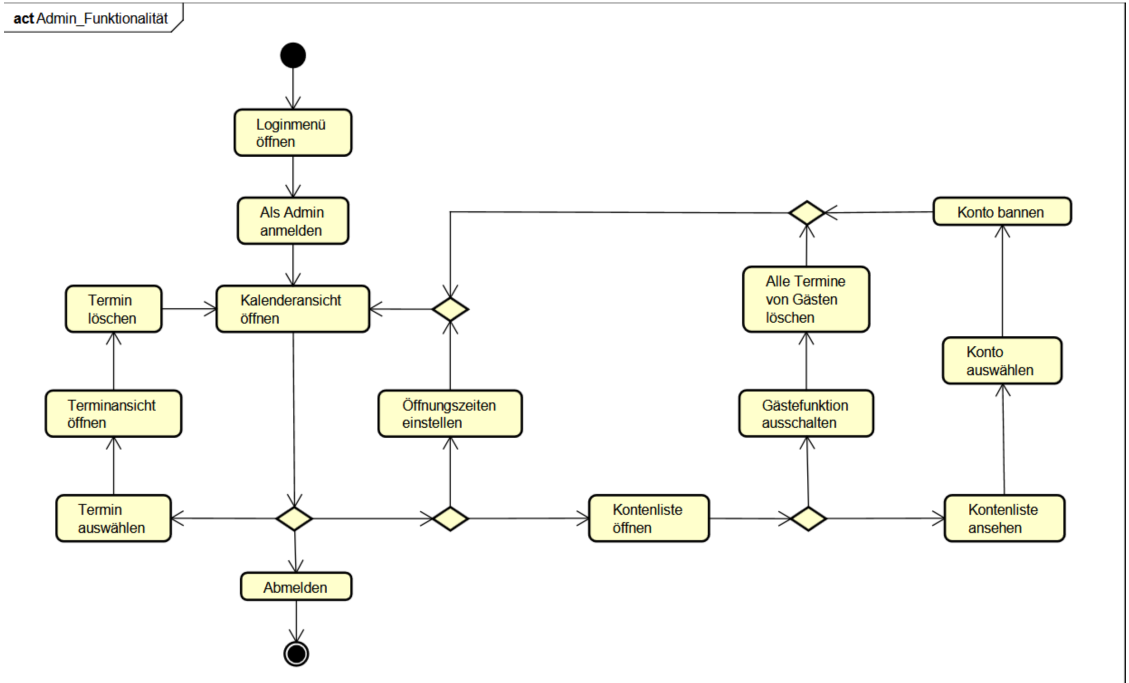
Checkout


Soli
User
Raum Buchung
Meine Reservierungen
Einstellungen
Checkout
Raum in Nutzung
Woche
Tag
Montag
Dienstag
Mittwoch
Donnerstag
Freitag
Buchung
Buchung
Buchung
Buchung


act Checkout-Prozess
[angemeldet]
Kalenderansicht öffnen
[laufende Buchung]
Checkout-Button drücken
Buchungsendzeit auf aktuelle Uhrzeit setzen

Kontenliste

Abbildung: Nur für Admins zugänglich.


act Admin_Funktionalität
Loginmenü öffnen
Als Admin anmelden
Kalenderansicht öffnen
Termin löschen
Terminansicht öffnen
Termin auswählen
Abmelden
Öffnungszeiten einstellen
Kontenliste öffnen
Gästefunktion ausschalten
Alle Termine von Gästen löschen
Konto bannen
Konto auswählen
Kontenliste ansehen

Controller

- Verarbeitung von Anfragen und Erzeugung von Antworten
- Dekodierung und Prüfung Eingaben in HTTP-Anfragen
- Generierung von Parametern für JTE-Templates der View
- Nutzung des Service-Layers zur Datenverarbeitung
- Weiterleitung unangemeldeter Nutzender zur Anmeldeseite bei geschützten Endpunkten
- Bereitstellung von Endpunkten für die Nutzenden-Interaktion, welche die HTTP-Oberfläche kapseln und Ansichten modellieren

Controller-Übersicht (Teil 1)

- **CalendarController:**
  - Kalenderansicht und Buchungsabruf
  - REST-Endpunkt für die Kalenderintegration mit FullCalendar
- **EventFeedController:**
  - REST-Endpunkt für Kalenderdaten im FullCalendar-Format
  - Verlinkung zu den Terminansichten
- **BookingViewController:**
  - Terminübersicht, Terminansicht
- **BookingCreateController:**
  - Termin-Erstellung-Ansicht zur Verarbeitung neuer Buchungen
  - Terminkonfliktlösung bei Überschneidungen von Buchungen
- **CollaborationRequestController:**
  - Formular zur Bestätigung oder Ablehnung einer Kollaboration
  - Kollaborationsanfragen verwalten

Controller-Übersicht (Teil 2)

- **LoginController:**
  - Login-Ansicht zur Verarbeitung der Anmeldungen (Admin, Gast, OIDC)
  - Gastanmeldung durch E-Mail-Link
- **UsersController:**
  - Kontoliste für Admins
- **MainController:**
  - Fehlerseite anzeigen
  - Benachrichtigungsseite bei deaktiviertem Konto

Beispiel: BookingViewController

- **GET /roomId/bookings**
  - Persönliche Terminübersicht anzeigen
- **GET /roomId/bookings/eventId**
  - Detailansicht eines Termins anzeigen
- **DELETE /roomId/bookings/eventId**
  - Termin löschen und zur Terminübersicht weiterleiten

Projektstruktur:

- **RESTful Design:** Klare Trennung von Lese- und Löschoperationen
- **Konsistente URL-Struktur:** Einheitliches Muster für alle terminbezogenen Endpunkte

Services
Service-Layer

- Koordination der Applikationslogik und Bereitstellung an die Endpunkte (z.B. Controller)
- Abkapselung der Logik der Datenverarbeitung von der Verarbeitung der HTTP-Anfragen und Antworten
- Verwendet Repository-Layer für die Abstraktion der Interaktion mit der Datenbank
- Bearbeitet komplexere Aufgaben, wie Terminbuchung, Terminkonfliktauflösung und E-Mail-Versand

Service-Übersicht (Teil 1)

- **Bookings Service:**
  - Verantwortlich für Terminverwaltung (Löschen, Buchen, Koordination der Raumteilung)
  - Bereitet Termine für die Kalenderansicht vor
  - Identifiziert potenzielle Terminkonflikte beim Buchen und bereitet eine Auflösung vor
- **E-Mail Service:**
  - Ermöglicht Senden von E-Mails an die im Konto hinterlegte E-Mail-Adresse
  - Die E-Mail-Adresse wird je nach Kontotyp vom Nutzenden oder vom OIDC-Provider bereitgestellt
- **Room Service:**
  - Verantwortlich für die Auswahl des Raums, der z.B. in einer Buchung verwendet wird
  - Ermöglicht eine einfache Erweiterung der Applikation, um mehrere Räume zu verwalten

Service-Übersicht (Teil 2)

- **System Configuration Service:**
  - Konfiguration der Applikation, z.B. der Aktivierung oder Deaktivierung von Gastkonten
  - Wenn von einem Admin die Funktion der Gastkonten deaktiviert wird, löscht dieser Service auch alle Termine der betroffenen Konten
- **User Service:**
  - Verwaltet die Konten: Deaktivieren von Konten, Überprüfen des Status (Admin oder Gast), sowie das Erstellen und Löschen von Konten

Daten

Daten-Layer (Repository-Paket)

- Sammlung an Schnittstellen für den Datenzugriff und die Verwaltung von Entitäten in der Datenbank
- Verwendet das **Spring Data JPA Framework** für die Verwaltung von CRUD-Operationen und benutzerdefinierten Abfragen
- **Hauptaufgaben:**
  - Verwaltung von Entitäten: Es definiert Schnittstellen, die von JPA-Repository erben, um CRUD-Operationen auf den Entitäten durchzuführen
  - Benutzerdefinierte Abfragen: Es definiert benutzerdefinierte Abfragen, um spezifische Daten aus der Datenbank zu extrahieren
- **Ziel:** Trennung der Datenzugriffslogik von der Anwendungslogik

Datenmodellübersicht

- Das Datenmodell von Soli ist in zwei Bereiche unterteilt:
  - **DTO (Data Transfer Objects):** Übertragen Daten zwischen Systemschichten
  - **Domain:** Repräsentiert die Datenobjekte, die in der PostgreSQL-Datenbank gespeichert werden.
- Das Datenmodell wird von der Datenbank in Tabellen mit definierten Beziehungen abgelegt. Daten werden nur bei aktiven Anfragen im Arbeitsspeicher gehalten und bei Bedarf in die Datenbank geschrieben.

Kernentitäten

- **Booking:** Repräsentiert eine Buchung mit Raum, Nutzer, Zeitspanne und optionalem Kommentar
- **User:** Speichert die OIDC-ID der Nutzenden, ergänzt um E-Mail, Name, Sprache und den Admin-Status
- **Raum:** Ermöglicht die Verwaltung mehrerer Räume

Authentifizierung

- Die Nutzenden werden über das OIDC-System des KIT identifiziert.
  KIT-Nutzer haben IDs mit dem Präfix `kit/`, Gäste mit `guest/`, der Admin besitzt die feste ID `admin`.

Sicherheitsmaßnahmen

- **ACID-Eigenschaften** der PostgreSQL-Datenbank gewährleisten Konsistenz und Schutz vor Datenverlust
- **Flyway** sorgt für automatische Datenbankmigrationen bei Modelländerungen, um Datenkonsistenz bei Systemaktualisierungen zu gewährleisten

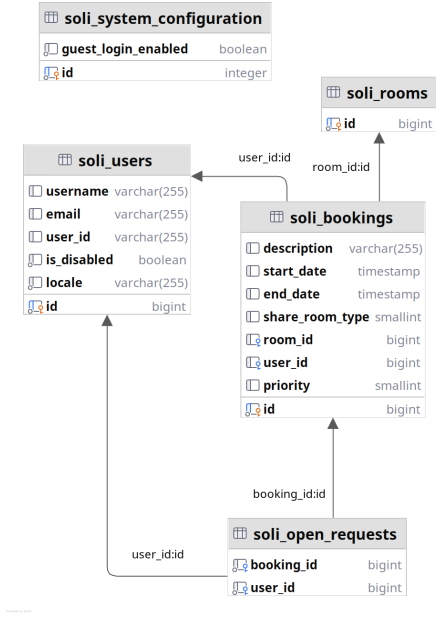


Abbildung: ER-Diagramm der Datenbank

Q&A