Februar 2024

DOKUMENTATION ROOMMATE

Inhalt

Einführung und Ziele

Aufgabenstellung

Das Projekt RoomMate ist eine Web-Application, die leere Räume zur Buchung zur Verfügung stellt. Studenten sollten in der Lage sein, Arbeitsplätze zu einem gewissen Zeitpunkt mit bestimmten Equipments zu suchen, zu reservieren und gegebenenfalls zu stornieren. Ein Nutzer sollte auch in der Lage sein, alle seine Reservierungen auf einmal zu sehen und zu managen.

Admins haben die Berechtigung, Räume, Sitze und Equipments hinzuzufügen. Sie sollten auch in der Lage sein, alle Reservierungen zu sehen und zu managen. Ein Admin kann, wenn er möchte, Reservierungen einfrieren, stornieren und Sitzplätze für eine gewisse Zeit sperren.

Qualitätsziele

Folgende Qualitätsziele wurden für das Projekt definiert:

– Die Seite sollte für alle Nutzer barrierefrei sein. Das heißt, dass die Seite auch nur mit der Tastatur bedienbar sein sollte
– Die Anwendung sollte eine adäquate Repräsentation der Geschäftslogik haben
– Die Anwendung sollte wartbar sein und dafür eine Onion-Architektur verwenden
– Die Anwendung sollte sicher sein. Das heißt, dass die Authentifikation und Autorisierung gut umgesetzt sein sollte
– Das Projekt sollte sinnvoll getestet werden. Das heißt, dass die Anwendung sowohl Unit- als auch Integrationstests haben sollte

Stakeholder

Rolle	Name	Kontakt	Erwartunghaltung
Dozent	Jens Bendispost	Jens@Bendisposto@uni-duesseldorf.de	Auftraggeber. Erwartet eine funktionsfähige Anwendung, die ihren Anforderungen entspricht.

Randbedingungen

Wir benutzen folgenden Technology Stack:
- Java 21
- Spring Boot
- Thymeleaf, HTML, CSS
- PostgreSQL über Docker
- Spring Security (OAuth2)
- Version Control System: Git
- Repository: GitHub
- weitere tools, wie lombok, docker, docker-compose, junit, mockito etc.

Kontextabgrenzung

Es gibt 3 wesentliche Kontexte, die in diesem Projekt eine Rolle spielen:

Der Admin-Kontext: Hier kann ein Admin Räume, Sitze und Equipments hinzufügen. Er kann auch Reservierungen sehen und managen, sowie Sitze für eine gewisse Zeit sperren und Reservierungen einfrieren.

Der User-Kontext: Hier kann ein User nach Räumen suchen, Sitze reservieren und stornieren. Er kann auch alle seine Reservierungen sehen und managen.

Der KeyMaster-Kontext: KeyMaster observiert die Anwendung und gibt wieder, welche Zugriffe, welche User auf welche Räume haben. Falls ein User in einem Raum reserviert, wird das auch in KeyMaster gespeichert.

Lösungsstrategie

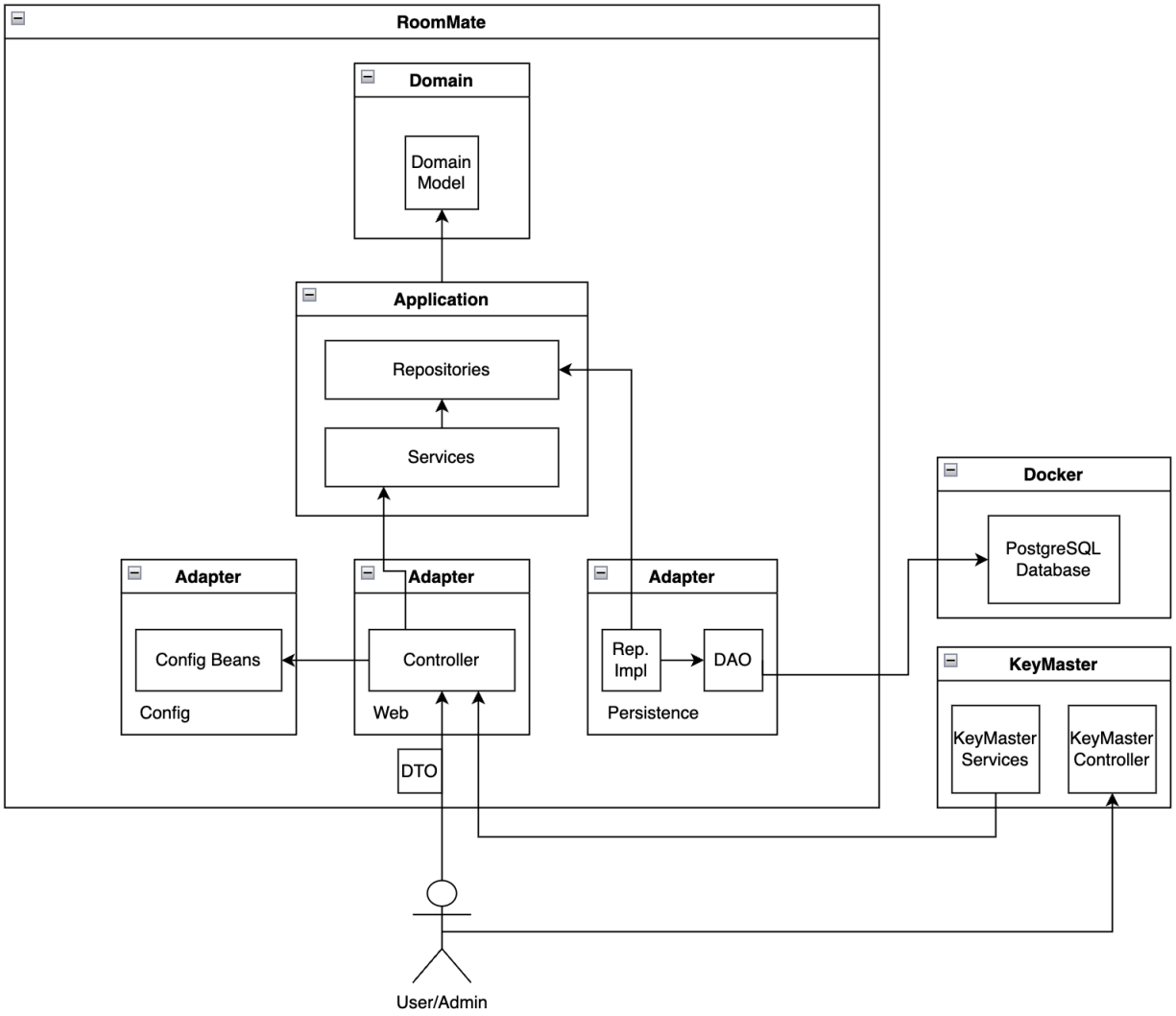
Als Lösungsstrategie für eine erfolgreiche Umsetzung des Projekts haben wir uns entschieden, zuerst einen horizontalen Prototypen zu entwickeln. Dieser Prototyp sollte die grundlegenden Funktionen der Anwendung beinhalten. Nachdem der Prototyp fertig ist, werden wir uns auf die Qualität der Anwendung konzentrieren.

Nach und nach implementieren wir die Anforderungen, die in der Aufgabenstellung definiert sind. Wir werden auch die Anwendung testen und die Qualität der Anwendung sicherstellen.

Wir implementieren immer wieder eine Funktion und testen diese. Wenn die Funktion fehlerfrei ist, implementieren wir die nächste Funktion.

Zuletzt gehen wir alle Anforderungen durch und stellen sicher, dass alle Anforderungen erfüllt sind.

Bausteinsicht



- Domain Model
- Application Services
- Repositories
- Data Transfer Objects
- Web Controller
- Configuration
- Repository Implementation
- Data Access Objects
- Docker PostgreSQL Database Container
- KeyMaster SCS

Das Programm enthält folgende Bausteine:

Bauteil 1 Domain Model

Das Domain Model bildet die Entitäten aus der Realität so gut wie möglich ab.
Sie weiss nichts von der Außenwelt mit Ausnahme von @Id und @Version Annotation, welche für uns kein Problem liefern und dessen Abhängigkeit nicht wirklich Probleme verursachen. Das ist der Kern unserer Anwendung.

Bauteil 2 Application Services

Die Application Services handeln den Großteil der Logik. Diese werden von Controllern und anderen Services aufgerufen. Sie enthält Methoden, welche den Großteil der Kalkulationen übernehmen.

Bauteil 3 Repositories

Die Repositories liegen im Application Services Package und definieren die Schnittstelle, welche unsere Repository Implementationen im Persistence Adapter implementieren soll.

Bauteil 4 Data Transfer Objects

Die View sendet mithilfe von DTOs mit den Handler-Methoden im Web Controller. Diese dienen zur Validierung der gesendeten Daten.

Bauteil 5 Web Controller

Web Controller empfangen HTTP-Requests von Usern und vom Fremdsystem KeyMaster. Die Handler-Methoden im Controller rufen Application Services auf, senden Entsprechende Views oder JSON zurück und sind auch für Error-Handling zuständig.

Bauteil 6 Configuration

Konfigurationsklassen sind hauptsächlich für Security zuständig. Sie hindern, dass unter anderem csrf-Attacken negiert werden und dass User mit OAuth2 authentifiziert werden. Sie legen zudem auch west, welche Routen eine Autorisierung benötigen und welche öffentlich sind.

Bauteil 7 Repository Implementation

Repository Implementationen injizieren DAOs und haben Methoden zur Verfügung, die pure Datenbanklogik ausführen. Diese Klassen implementieren die Interfaces aus dem Application Service Package

Bauteil 8 Data Access Objects

DAOs sind Objekte, welche unter der Haube SQL-Anfragen an die Datenbank senden.

Bauteil 9 Docker PostgreSQL Database Container

Wir holen uns einen PostgreSQL Container und binden diese an unsere Applikation an.
In der Datenbank werden unsere Daten persistiert.

Bauteil 10 KeyMaster SCS

KeyMaster ist ein Fremdsystem, das Reservierungen aus RoomMate zu Räumen und Usern angibt. Sie sendet jede Minute einen HTTP-Request an einen REST-Controller von RoomMate, um eine Access-Liste zu erhalten. RoomMate fragt minütlich auch Daten von KeyMaster an, um die adäquate Access-Liste herzustellen.

Architekturentscheidungen

Es wurde entschieden, die Onion Architektur zu verwenden. Die Onion Architektur ist eine Schichtenarchitektur, die die Geschäftslogik in den Mittelpunkt stellt. Die Geschäftslogik ist in der Mitte und wird von den anderen Schichten umgeben. Die Schichten sind so aufgebaut, dass die inneren Schichten von den äußeren Schichten unabhängig sind. Die äußeren Schichten können auf die inneren Schichten zugreifen, aber nicht umgekehrt.