**SLGP – Softwarelösungen für Geschäftsprozesse**

Fallstudie SLGP Group NINE - AlphaRooms

Applikationsdokumentation



Version: 1.5

Datum: 16.12.2018

**Inhaltsverzeichnis**

[1 Abbildungsverzeichnis 4](#_Toc532742510)

[2 Begriffe / Abkürzungen 5](#_Toc532742511)

[3 Einführung 5](#_Toc532742512)

[3.1 Zweck dieses Dokumentes 6](#_Toc532742513)

[3.1.1 Das Dokument umschreibt folgende Punkte: 6](#_Toc532742514)

[3.1.2 AlphaRooms enthält folgende Eigenschaften 6](#_Toc532742515)

[3.1.3 Ziele 6](#_Toc532742516)

[4 Anforderungen 7](#_Toc532742517)

[4.1 User Stories 7](#_Toc532742518)

[4.1.1 Lehrperson (Rolle User) 7](#_Toc532742519)

[4.1.2 Sekretariat (Rolle Admin) 8](#_Toc532742520)

[5 Architektur 9](#_Toc532742521)

[5.1 Übergeordnete Struktur 9](#_Toc532742522)

[5.2 Projektstruktur 10](#_Toc532742523)

[5.3 Generelle Struktur 11](#_Toc532742524)

[6 Datenstrukturen 12](#_Toc532742525)

[6.1 Applikation 12](#_Toc532742526)

[6.2 Datenbank 12](#_Toc532742527)

[7 User Interface 14](#_Toc532742528)

[7.1 Navigation 14](#_Toc532742529)

[7.2 UI für User Story Lehrer - Rolle User 15](#_Toc532742530)

[7.2.1 HomeView.fxml 15](#_Toc532742531)

[7.2.2 BookingView.fxml 16](#_Toc532742532)

[7.2.3 Menubar.fxml 18](#_Toc532742533)

[7.2.4 AddView.fxml 18](#_Toc532742534)

[7.2.5 StatView.fxml 21](#_Toc532742535)

[7.2.6 RoomView.fxml 22](#_Toc532742536)

[7.2.7 UserView.fxml 23](#_Toc532742537)

[7.3 UI für User Story Sekretariat - Rolle Admin 24](#_Toc532742538)

[7.3.1 AddView.fxml 24](#_Toc532742539)

[8 Interfaces 25](#_Toc532742540)

[9 Testen 27](#_Toc532742541)

[9.1 Unit Test 27](#_Toc532742542)

[9.2 Usability-Test - User Story 28](#_Toc532742543)

[9.2.1 Sekretariat (Rolle Admin) 29](#_Toc532742544)

[9.3 Zusatz 29](#_Toc532742545)

[10 Deployment 30](#_Toc532742546)

[10.1 Architekturbeschreibung 30](#_Toc532742547)

[10.1.1 Server 30](#_Toc532742548)

[10.1.2 Client 30](#_Toc532742549)

[11 Konfigurationsangaben 31](#_Toc532742550)

[11.1 Voraussetzungen 31](#_Toc532742551)

[11.1.1 Server 31](#_Toc532742552)

[11.1.2 Client 31](#_Toc532742553)

[11.2 Credentials 31](#_Toc532742554)

[11.2.1 Installation Server 32](#_Toc532742555)

[11.2.2 Installation Client 33](#_Toc532742556)

[11.3 Database Creator (Scrip) – Manual Install / With SampleData 34](#_Toc532742557)

[12 Gemachte Erfahrungen (lessons learned) 38](#_Toc532742558)

# Abbildungsverzeichnis

[Abbildung 1: Client / Server Architektur 9](#_Toc532742559)

[Abbildung 2: Projektstruktur 10](#_Toc532742560)

[Abbildung 3: Server Container 11](#_Toc532742561)

[Abbildung 4: Enitiy Klassen 12](#_Toc532742562)

[Abbildung 5: ERM Version 1.0 12](#_Toc532742563)

[Abbildung 6: ERM Version 2.0 13](#_Toc532742564)

[Abbildung 7: Übersicht Aufruf FXML Sichten 14](#_Toc532742565)

[Abbildung 8: AlphaRooms Menü Bar 14](#_Toc532742566)

[Abbildung 9: Anmeldemaske AlphaRooms (HomeView.fxml) 15](#_Toc532742567)

[Abbildung 10: AlphaRooms Buchungsübersicht (BookingView.fxml) – UC4 16](#_Toc532742568)

[Abbildung 11: AlphaRooms Logout (HomeView.fxml) 16](#_Toc532742569)

[Abbildung 12: Stornierung einer Buchung in AlphaRooms (BookingView.fxml) – UC5 17](#_Toc532742570)

[Abbildung 13: Stornierte Buchung in AlphaRooms (BookingView.fxml) – UC5 17](#_Toc532742571)

[Abbildung 14: Applikationssteuerung (Menubar.fmxl) 18](#_Toc532742572)

[Abbildung 15: Neue Buchung mit AlphaRooms (AddView.fxml) – UC1 18](#_Toc532742573)

[Abbildung 16: Negativtest Buchung in gebuchten Zeitbereich (AddView.fxml) – UC4 19](#_Toc532742574)

[Abbildung 17: Raumverfügbarkeit prüfen und Belegung berücksichtigen (AddView.fxml) – UC2 19](#_Toc532742575)

[Abbildung 18: Buchung erstellen (AddView.fxml) – UC1 20](#_Toc532742576)

[Abbildung 19: Statistik (StatView.fxml) – UC10 21](#_Toc532742577)

[Abbildung 20: Räume verwalten – UC8 22](#_Toc532742578)

[Abbildung 21: Benutzer verwalten – UC8 23](#_Toc532742579)

[Abbildung 22: Neue Buchung mit anderem Benutzer erstellen (Funktion der Rolle Admin) – UC9 24](#_Toc532742580)

[Abbildung 23 – RMI\_Room 25](#_Toc532742581)

[Abbildung 24 – RMI\_User 26](#_Toc532742582)

[Abbildung 25: JUnit Test Entität Room 27](#_Toc532742583)

[Abbildung 26: JUnit Test Entität User 27](#_Toc532742584)

[Abbildung 27: Config postgresql.conf 32](#_Toc532742585)

[Abbildung 28: Create Database PGAdmin 32](#_Toc532742586)

[Abbildung 29: Query Tool 33](#_Toc532742587)

[Abbildung 30: Server ready 33](#_Toc532742588)

[Abbildung 31: Konfigurationsdatei rmi\_client.properties 33](#_Toc532742589)

# Begriffe / Abkürzungen

GUI Graphical User Interface (Grafische Benutzeroberfläche)

RMI Remote Method Invokation (Aufruf entfernter Methoden)

Java FX ein Framework zur Erstellung plattformübergreifender Java-Applikationen.

SLGP Softwarelösungen für Geschäftsprozesse

DB Datenbank

JVM Java Virtual Machine (Oracle)

CUD Create Update and Delete Operations

# Einführung

Im Rahmen der auferlegten Fallstudie soll für die HSLU ein Raumverwaltungsprogramm erstellt werden. Die Applikation hat den Zweck die vorliegenden Geschäftsprozesse zu verbessern und zu optimieren.

Im Dokument AlphaRooms\_Dokumentation\_V1.4.docx wird der Ablauf der Fallstudie dokumentiert. Der Ablauf wird in diesem Dokument gemäss der Aufgabenstellung in zehn Phasen unterteilt.

1 Einführung

2 Anforderungen

3 Architektur

4 Datenstrukturen

5 User Interface

6 Interfaces

7 Testen

8 Deployment

9 Konfigurationsangaben

10 Gemachte Erfahrungen

Wir werden mit der Realisation des neuen Buchungssystem beginnen, wenn die ersten fünf Phasen erfolgreich abgeschlossen wurde.

Die Aufgabenstellung und User Stories sollen verhindern, dass fehlerhafte Interpretationen das Projektresultat verfälschen.

Die Erfolgskontrolle wird anhand des Testens vorgenommen. Anhand der Resultate der User Stories können Erfolge dokumentiert und festgehalten werden.

Da die Fallstudie von uns als Projekt wahrgenommen wird, werden wir von nun an die „Fallstudie“ mit „Projekt“ bezeichnen.

## Zweck dieses Dokumentes

Das Dokument gibt Einblick in die Applikation «AlphaRooms» und dessen Zusammenspiel zwischen Server und Client mit dem Übertragungsprotokoll RMI.

### Das Dokument umschreibt folgende Punkte:

* Architektur
* Datenstrukturen
* Klassendiagramm
* Module/Komponenten und deren Verteilung
* User Interface
* Konfigurationsangaben
* Fazit

### AlphaRooms enthält folgende Eigenschaften

**Rolle User:**

Meine Buchungen

Buchung ansehen - getBookings

Buchung stornieren - cancelBooking

Buchung erstellen

Raum suchen

Raum suchen mit Auslastung (Anzahl Personen / Uhrzeit /Datum) - getRooms

Raum gewählt / Anzahl Personen gewählt / Uhrzeit & Datum eingetragen – checkAvailability

Buchung erstellen - createBooking

**Rolle Admin:**

Alle Buchungen stornieren - getBookings

Benutzer erfassen - getUsers / createUser

Raum erfassen - getRoom / createRoom

**Statistik:**

getRoomStats (Booking Count, avg. People Utilization, Daytime)

### Ziele

Ziel ist es einen Prototyp zu erstellen. Um die Aufgabekriterien aufzuteilen wurden mehrere User Stories erstellt und analysiert. Diese wurden mit dem folgenden Prototyp «AlphaRooms» umgesetzt.

Sämtliche User Stories der Fallstudie wurden innerhalb von AlphaRooms umgesetzt.

# Anforderungen

Beschreibung von Anforderungen werden in Form von User Stories vorgenommen werden. Eine User Story ist immer einem Anwendungsfall (Use Case) zugewiesen. Der Use Case soll Verwechslungen vermeiden und zusätzlich einen Kontext für die User Story liefern.

## User Stories

### Lehrperson (Rolle User)

|  |  |
| --- | --- |
| **Use Case ID:** | **UC1** – Neue Buchung erfassen |
| User Story Description: | Als Lehrperson kann ich einen Raum suchen, damit ich den Raum buchen kann. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC2** – Raum identifizieren |
| User Story Description: | Als Lehrperson kann ich einen Raum mit meiner gewünschten Belegung suchen, damit ich einen Raum buchen kann, der meiner Gruppengrösse entspricht. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC3** – Raum reservieren |
| User Story Description: | Als Lehrperson kann ich einen Raum reservieren sollte der Raum im gewünschten Zeitraum frei sein. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC4** – Raumverfügbarkeit prüfen |
| User Story Description: | Als Lehrperson kann ich Buchungen einsehen, um zu prüfen welche Person zu welchem Zeitpunkt einen Raum gebucht hat. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC5** – Buchung stornieren |
| User Story Description: | Als Lehrperson kann ich meine Buchungen stornieren, wenn ich verhindert bin. |
| **Use Case ID:** | **UC6** – Buchung stornieren |
| User Story Description: | Als Lehrperson kann ich mich am System nur anmelden, wenn ich Username und Passwort habe, somit sind meine Daten geschützt. |

### Sekretariat (Rolle Admin)

|  |  |
| --- | --- |
| **Use Case ID:** | **UC7** – Raum verwalten |
| User Story Description: | Als Sekretariat kann in einen Raum erfassen, damit der Raum gebucht werden kann. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC8** – Benutzer verwalten |
| User Story Description: | Als Sekretariat kann in einen Benutzer erfassen, damit der Benutzer sich anmelden kann. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC9** – Umbuchen |
| User Story Description: | Als Sekretariat kann ich alle Arten von Buchungen, erstellen und stornieren, um den Abtausch von Räumen zu ermöglichen. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC10** – Statistik abrufen |
| User Story Description: | Als Sekretariat kann ich Statistiken über die Raumbelegung abrufen, um zu sehen wann Räume am wenigsten belegt sind. |

|  |  |
| --- | --- |
| **Use Case ID:** | **UC11** – Anmelden |
| User Story Description: | Als Sekretariat kann ich mich am System nur anmelden, wenn ich Username und Passwort habe, somit sind meine Daten geschützt. |

# Architektur

Da es sich bei der Fallstudie um die Entwicklung einer verteilten Applikation handelt kommen nachfolgende Strukturen zum Tragen. Die generelle Struktur ist bekannt als eine Client / Server Struktur.

## Übergeordnete Struktur

Archi te ktur 
Alpha Rooms Datenbank 
AlphaRooms Clients 
Alpha Rooms Server 
Netzwerk 

**AlphaRooms Server:**

Server Applikation basierend auf Java welcher mithilfe von RMI über das Netzwerk mit den AlphaRoom Clients kommuniziert.

**AlphaRooms Clients:**

Client Applikation basierend auf Java welche mithilfe JavaFX dem Benutzern eine Grafische Benutzeroberfläche zur Verfügung stellt.

**AlphaRooms Datenbank:**

SQL Datenbank mit Postgres

Computergenerierter Alternativtext:

insisted 
Remote Objects 
asgard 
bushes 


**Kommunikation:**

Die Kommunikation wird mithilfe von RMI bewerkstelligt. Die wichtigsten Komponenten wurden für die Verständlichkeit auf einer Zeichnung festgehalten.

Abbildung 1: Client / Server Architektur

## Projektstruktur

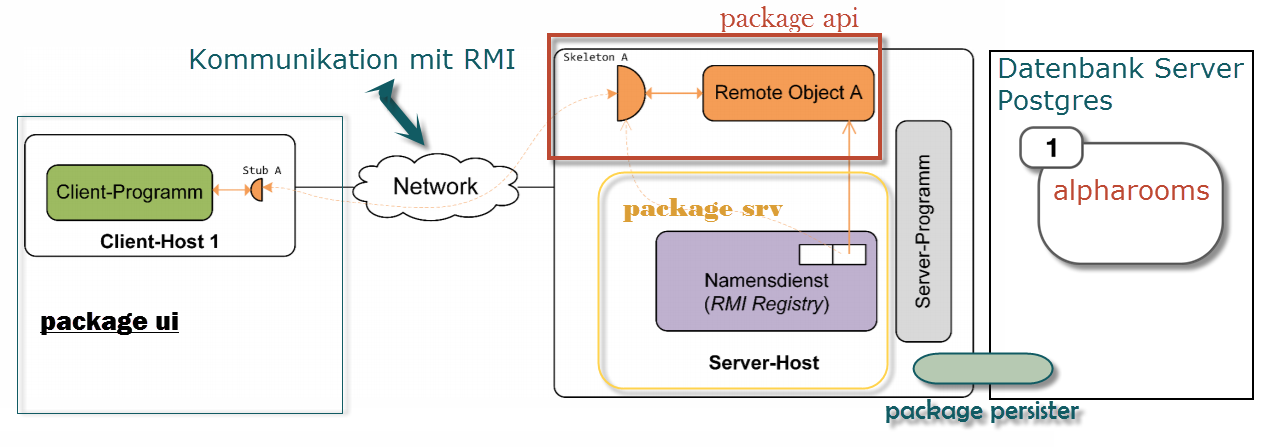
Als erstes haben wir Uns im Team die vorgegebenen Strukturen des Projektes angesehen, um uns einen Überblick zu verschaffen. Die Struktur im Verzeichnis «persister» bildet zudem die Datenbankstruktur von «AlphaRooms» ab. Ebenfalls ersichtlich ist die Unterteilung zwischen dem Client und der Server Komponente.

Computergenerierter Alternativtext:



Abbildung 2: Projektstruktur

## Generelle Struktur



Die Architektur unterteilt sich in die folgenden Bereiche:

1. srv
2. persister
3. api
4. domain
5. business

Die drei Bereiche «srv», «persister», «api» wurden genutzt. Die Bereiche «domain» und «business» wurden aus Zeitgründen nicht mitberücksichtigt.

Server Architektur 
srv 
API / RMI 
Dersister 
booking 
domain 
interface 
API 

Abbildung 3: Server Container

# Datenstrukturen

## Applikation

Die folgende Abbildung zeigt die Entitys der Applikation «AlphaRooms». Die Beziehungen unter den Objekten werden im Abschnitt «6.2 Datenbank» genauer beschrieben.

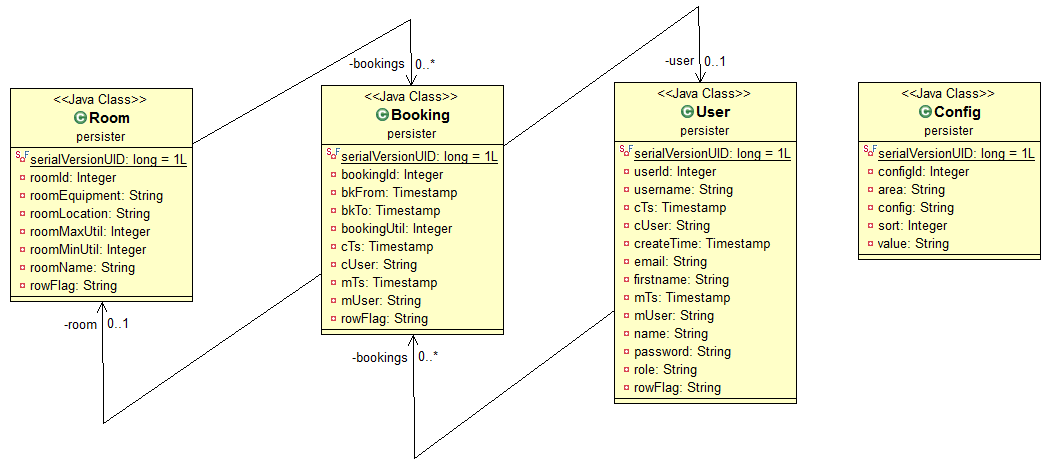


Abbildung 4: Enitiy Klassen

## Datenbank

Die Datenbank wurde bei unserem Projekt zu Beginn erstellt. Über die User Stories und Use Cases haben wir versucht sämtliche Anwendungen in einer simplen und effizienten Struktur abzubilden.

Computergenerierter Alternativtext:
config 
name V ARCHAR(IOO) 
Value V ARCHAR(IOO) 
username V ARCHAR(16) 
email VARCHAR(255) 
passvvord VARCHAR(32) 
time TIMESTAMP 
room 
room 
_name 
_locaton V ARCHAR(45) 
bocking 
room id INT 
creaÉ_user VARCHAR(16) 
modifv_user V ARCHAR(16) 

Abbildung 5: ERM Version 1.0

Das folgende Konstrukt ist aus unserer Arbeit entstanden und wurde für die Entwicklung verwendet. Alle Schlüsselfelder wurden einer Feldnotation bezeichnet. Die Notation «TABELLE\_ID» wurde für Primärschlüssel gewählt und «ID\_TABELLE» für einen Fremdschlüssel . Zudem wurde das Feld «username» als UNIQUE deklariert. Somit kann das Feld username ebenfalls als Schlüssel für die Tabelle user verwendet werden.

Computergenerierter Alternativtext:
config 
module V ARCHAR(45) 
name VARCHAR(IOO) 
Value VARCHAR(100) 
order INT 
username V ARCHAR(16) 
email VARCHAR(255) — 
passvvord VARCHAR(32) 
creaÉ time TIMESTAMP 
role 
c tsDATETIME 
c_user VARCHAR(16) 
ts DATETIME 
m 
m _user V ARCHAR(16) 
room 
room 
room 
room 
room 
room 
id 
_name 
_locaton VARCHAR(45) 
_equi pm ent V ARCHAR(45) 
min util INT 
max utl INT 
bocking 
ron_fiag V ARCHAR(8) 
booking_id INT 
booking_for VARCHAR(16) 
booking_utl IBIT 
id 
room INT 
c_user VARCHAR(16) 
c tsDATETIME 
m _user V ARCHAR(16) 
ts DATETIME 
m 
bk from DATETIME 
Obk DATETIME 

Abbildung 6: ERM Version 2.0

# User Interface

Das User Interface von AlphaRooms wurde so einfach wie möglich gehalten und lässt Platz für Erweiterungen. Die nachfolgende Abbildung gibt Aufschluss über die Struktur der einzelnen (\*.fxml) Sichten. Die Navigation wird über eine Menü Bar im linken oberen Bildrand vorgenommen. Sämtliche Sichten sind funktionstüchtig und im Prototyp einsehbar.

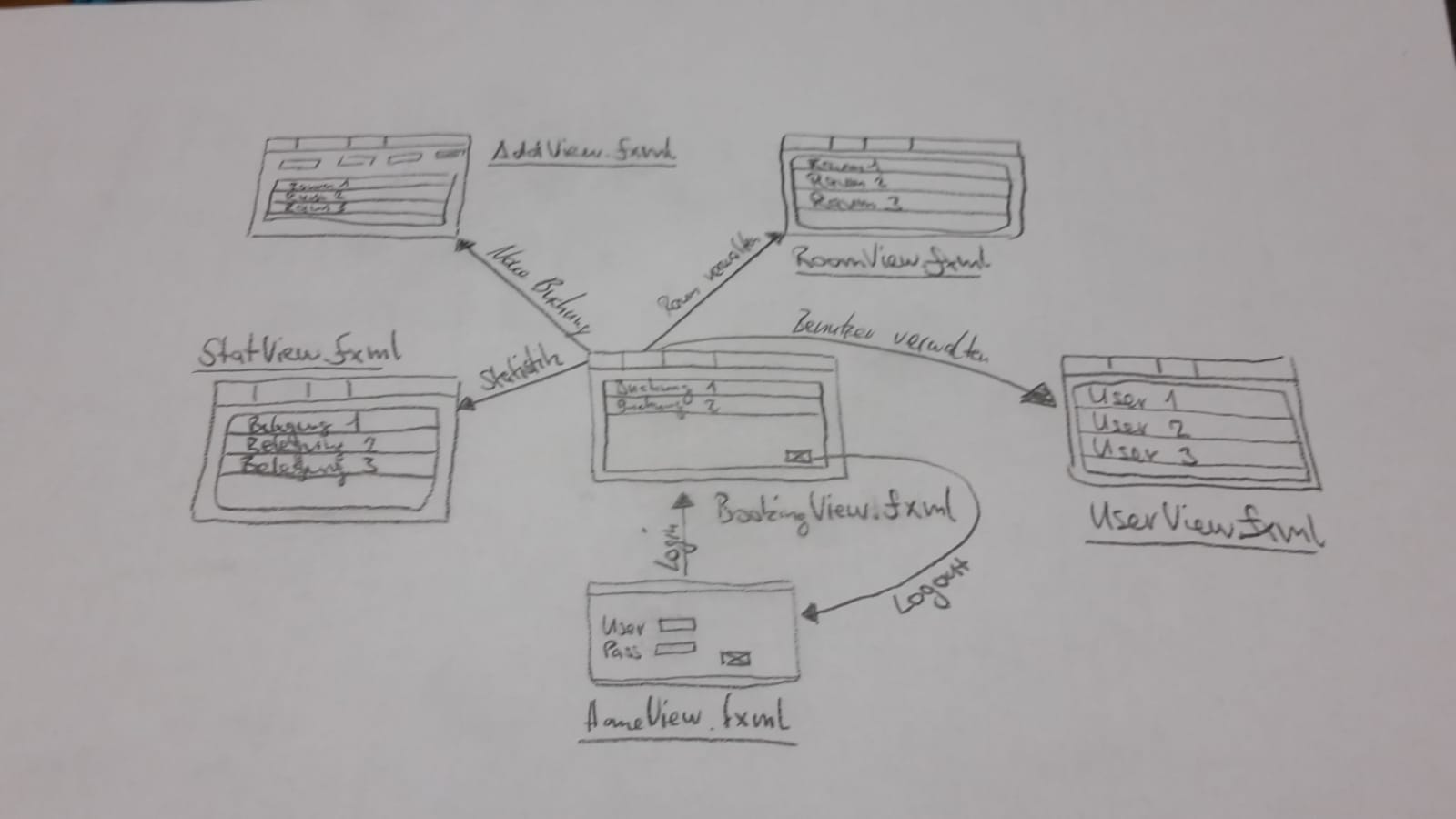


Abbildung 7: Übersicht Aufruf FXML Sichten

## Navigation

Folgende Menüeinträge sind in der Menü Bar vorhanden. Die Menüeinträge dienen der Navigation und werden benötigt um von einer Sicht zur Anderen zu Springen.

|  |  |
| --- | --- |
| C:\Users\Admin\AppData\Local\Temp\SNAGHTMLf5c7d5.PNG | C:\Users\Admin\AppData\Local\Temp\SNAGHTMLf5f02c.PNG |

Abbildung 8: AlphaRooms Menü Bar

Unterteilt wird das User Interface für die Rollen «User» und «Admin». Im nachfolgenden Kapitel werden die einzelnen Funktionalitäten der Software über das UI beschrieben.

## UI für User Story Lehrer - Rolle User

Das folgende Kapitel umfasst das UI für die Rolle User. Im Kontext der Fallstudie entspricht die Rolle User einer Lehrperson.

### HomeView.fxml

Login Maske der Applikation AlphaRooms.



Abbildung 9: Anmeldemaske AlphaRooms (HomeView.fxml)

#### Datenstruktur

Sämtliche Felder der Datenbank sind auf der Tabelle users zu finden.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feld Formular** | **Feld Datenbank** | **Beschreibung** | **Funktion** |
| **User** | username | Username des Benutzers | Identifikation des Benutzers |
| **Password** | Password | Passwort des Benutzers | Identifikation des Benutzers |

### BookingView.fxml

Die Sicht BookingView bietet dem Benutzer die Übersicht über sämtliche Buchungen und bietet ebenfalls die Möglichkeit Buchungen zu stornieren. In der Rolle Benutzer können nur eigene Buchungen storniert werden.

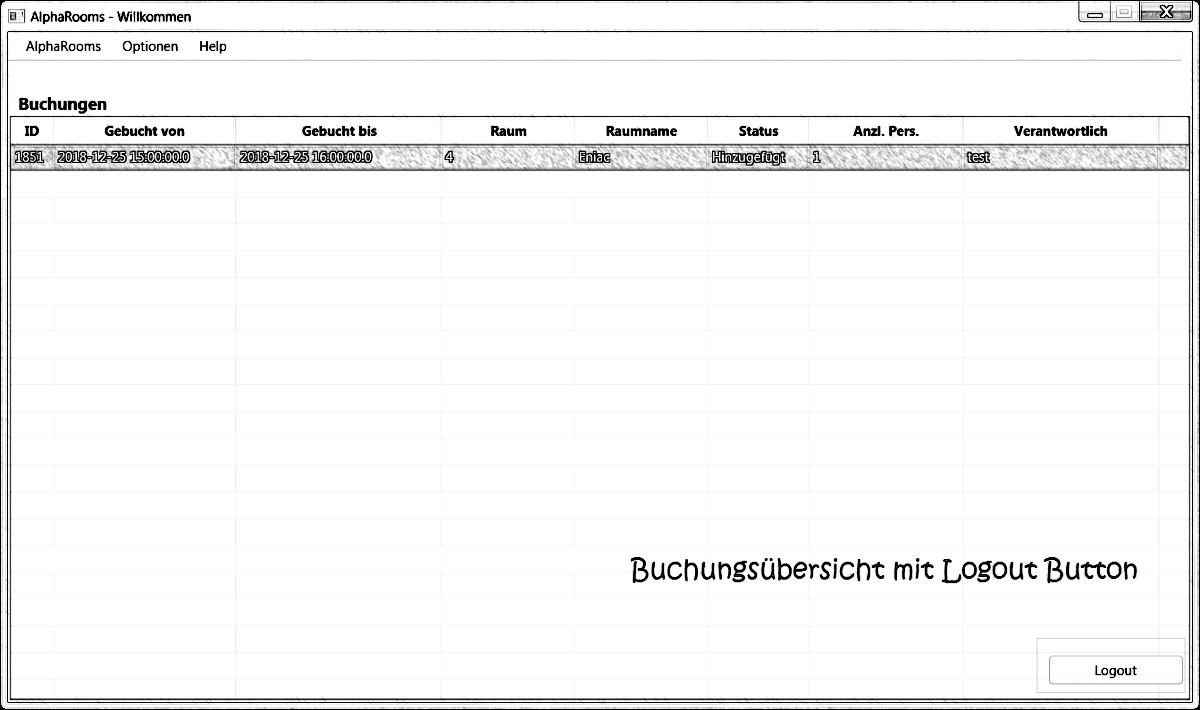


Abbildung 10: AlphaRooms Buchungsübersicht (BookingView.fxml) – UC4

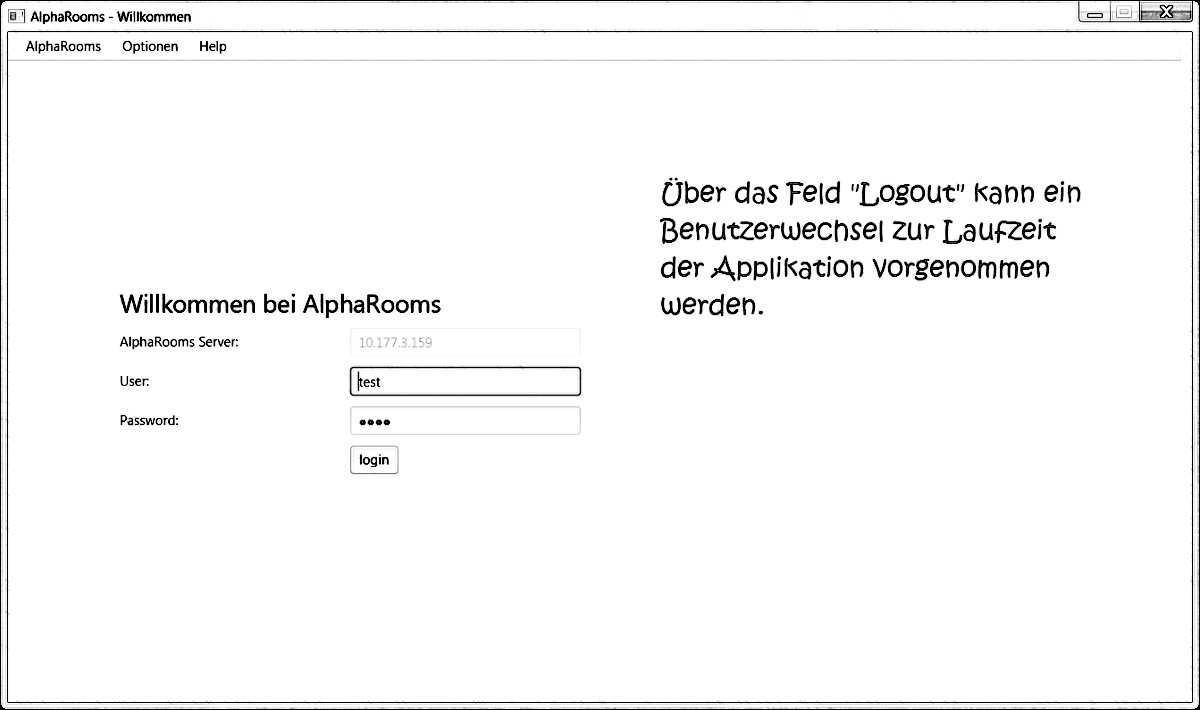


Abbildung 11: AlphaRooms Logout (HomeView.fxml)

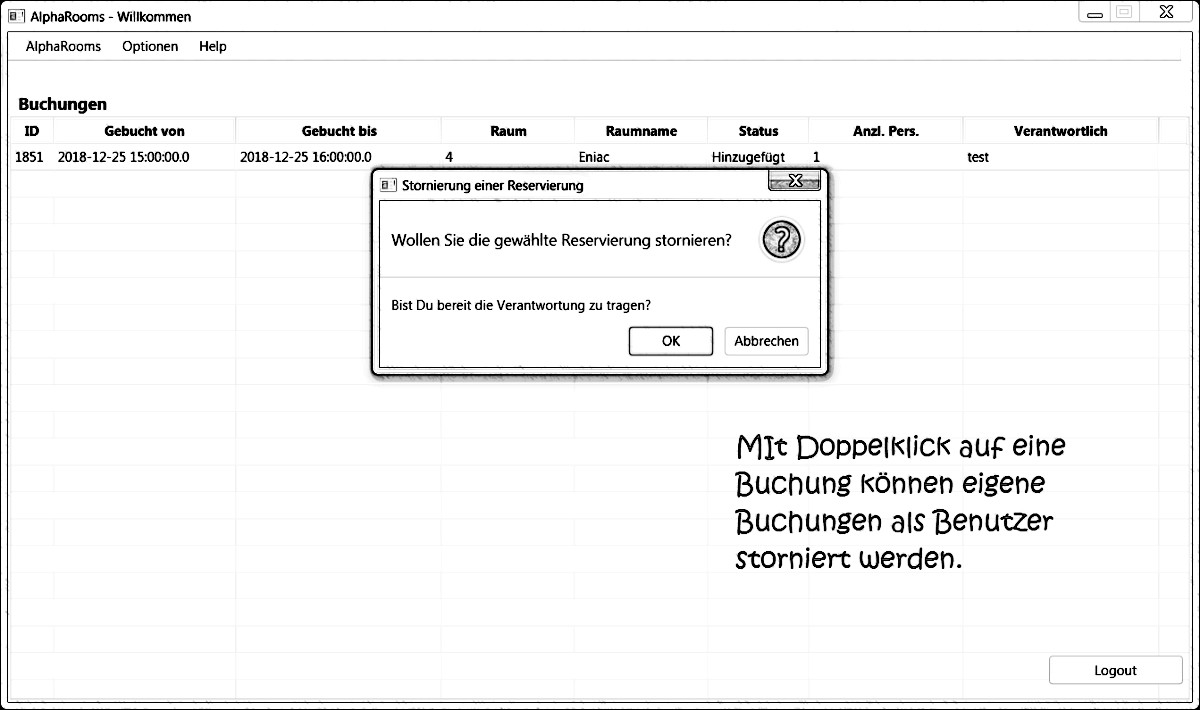


Abbildung 12: Stornierung einer Buchung in AlphaRooms (BookingView.fxml) – UC5



Abbildung 13: Stornierte Buchung in AlphaRooms (BookingView.fxml) – UC5

#### Datenstruktur

Sämtliche Werte sind in der Tabelle bookings zu finden. Für den Raum Name wird ein Lookup auf die Tabelle rooms ausgeführt. Verwendete Datenstruktur:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feld Formular** | **Feld Datenbank** | **Beschreibung** | **Funktion** |
| **ID** | Booking\_id | ID der Buchung | Identifikation der Buchung |
| **Gebucht von** | Bk\_from | Startzeit und Datum der Buchung | Beginn Zeitfenster der Buchung |
| **Gebucht bis** | BK\_to | Endzeit und Datum der Buchung | Ende Zeitfenster der Buchung |
| **Raum** | Id\_room | ID des Raums | Identifikation des Raumes |
| **Raum Name** | Room\_name | Raum Name | Erkennung des Raumes |
| **Status** | Row\_flag | Status (Hinzugefügt / STORNIERT) | Eine Buchung kann Statusabhängig behandelt werden |
| **Anzl. Pers.** | Booking\_util | Raumauslastung der Buchung in Personen | Erkennen der Auslastung |
| **Verantwortlich** | Booking\_for | Verantwortlich für die Buchung | Deklaration der Ansprechsperson |

### Menubar.fxml

Die Sicht der Menubar erlaubt es dem Benutzer die Applikation zu steuern.

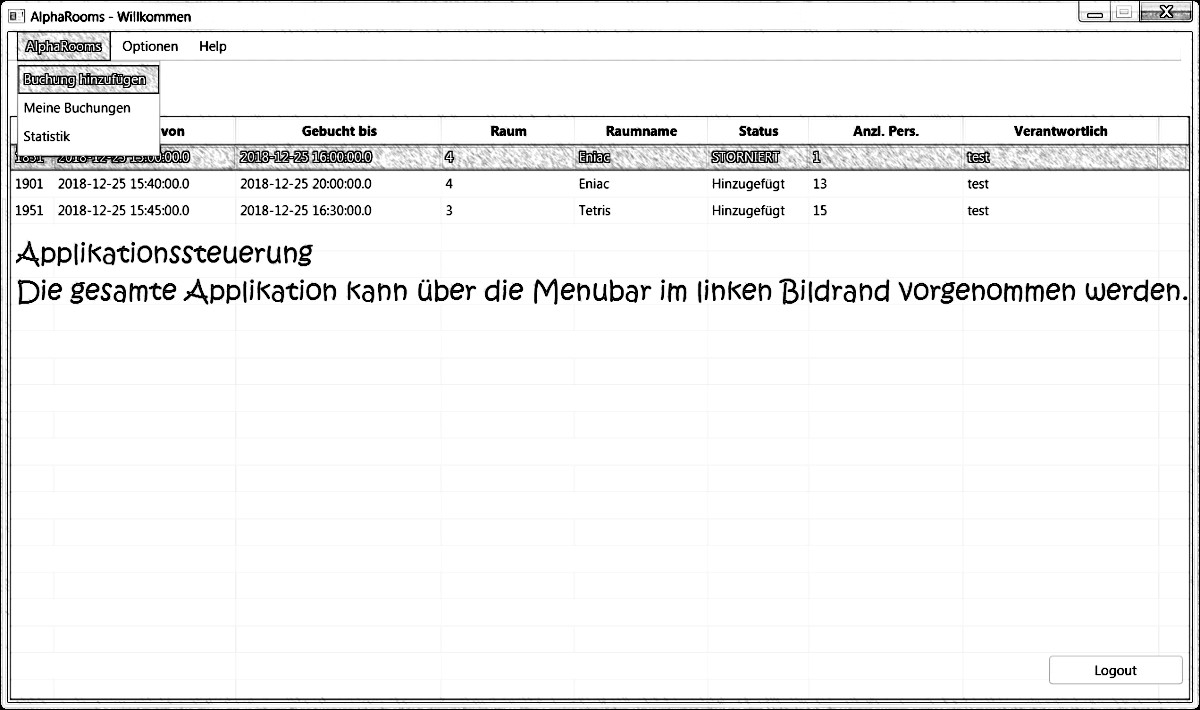


Abbildung 14: Applikationssteuerung (Menubar.fmxl)

#### Datenstruktur

Es wurden keine Datenstrukturen für die Sicht Menubar.fxml verwendet. Alle Werte sind fix codiert.

### AddView.fxml

Die Sicht AddView ist eine der wichtigsten im System und bietet die Möglichkeit einen Raum entsprechend der Verfügbarkeit wie auch Raumgrösse zu finden.

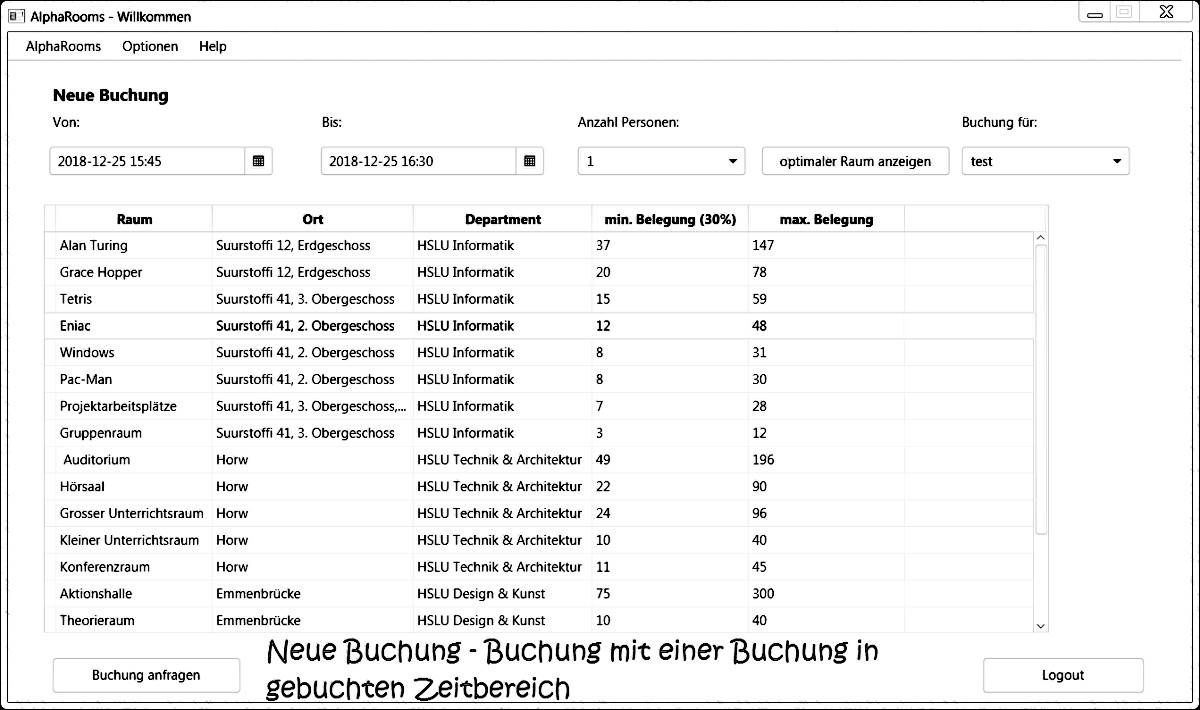


Abbildung 15: Neue Buchung mit AlphaRooms (AddView.fxml) – UC1

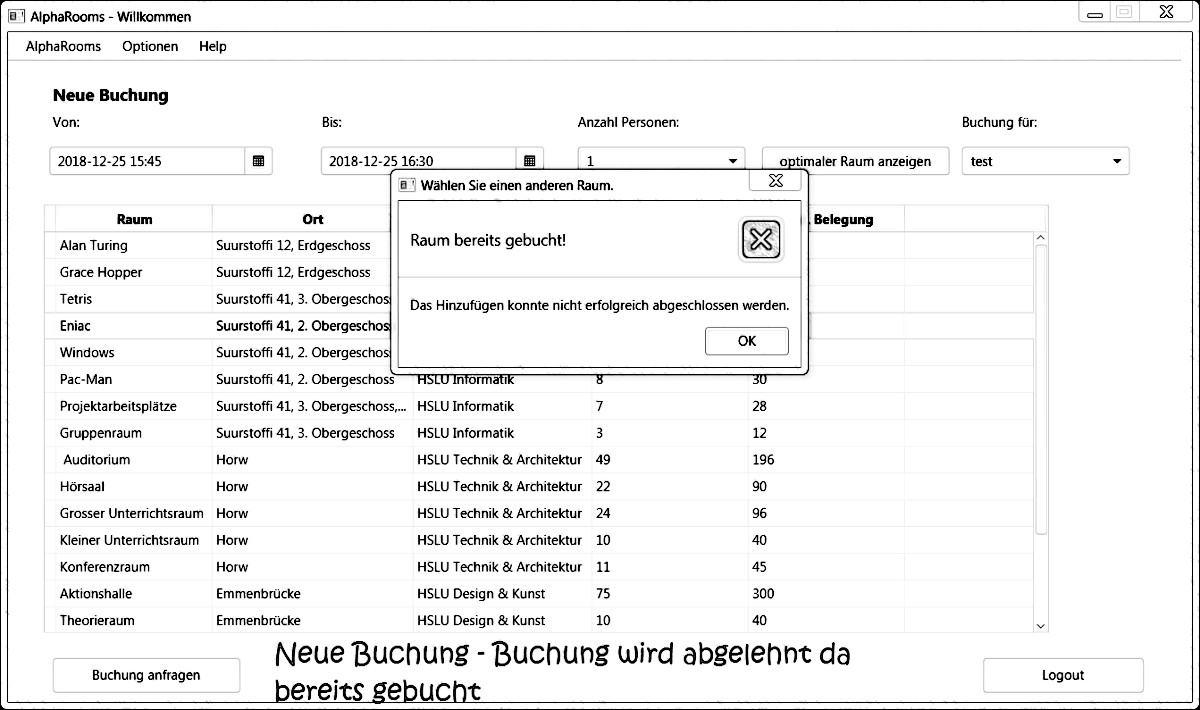


Abbildung 16: Negativtest Buchung in gebuchten Zeitbereich (AddView.fxml) – UC4

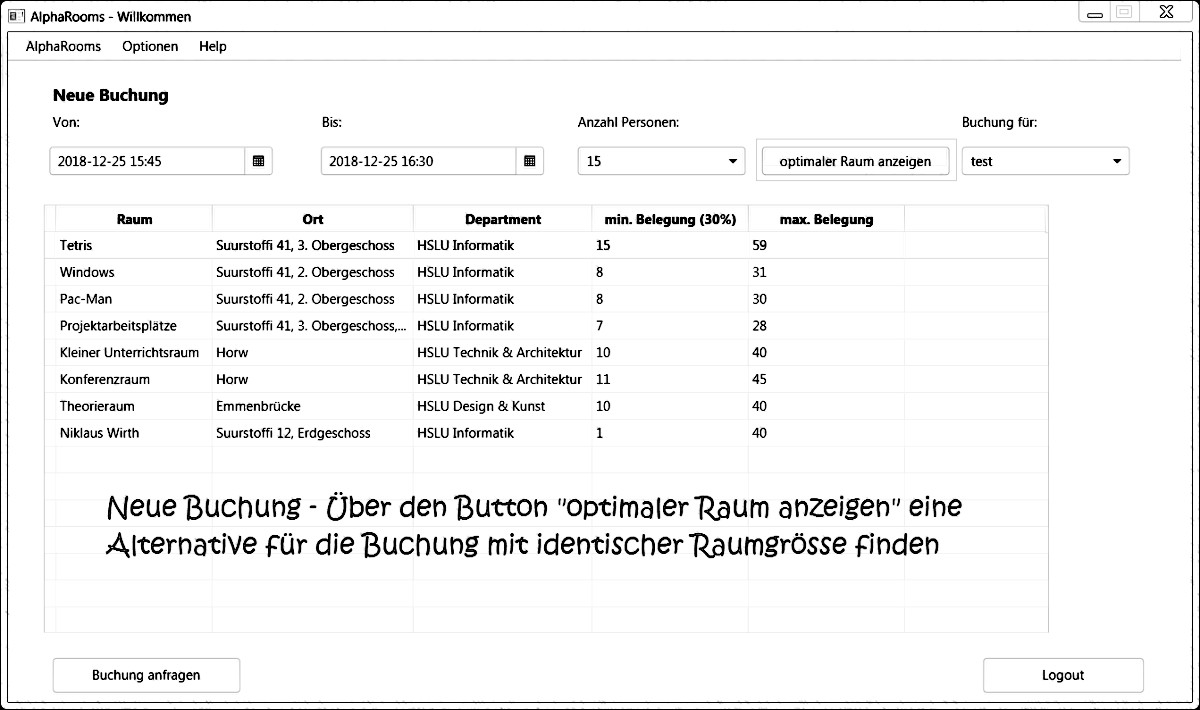


Abbildung 17: Raumverfügbarkeit prüfen und Belegung berücksichtigen (AddView.fxml) – UC2

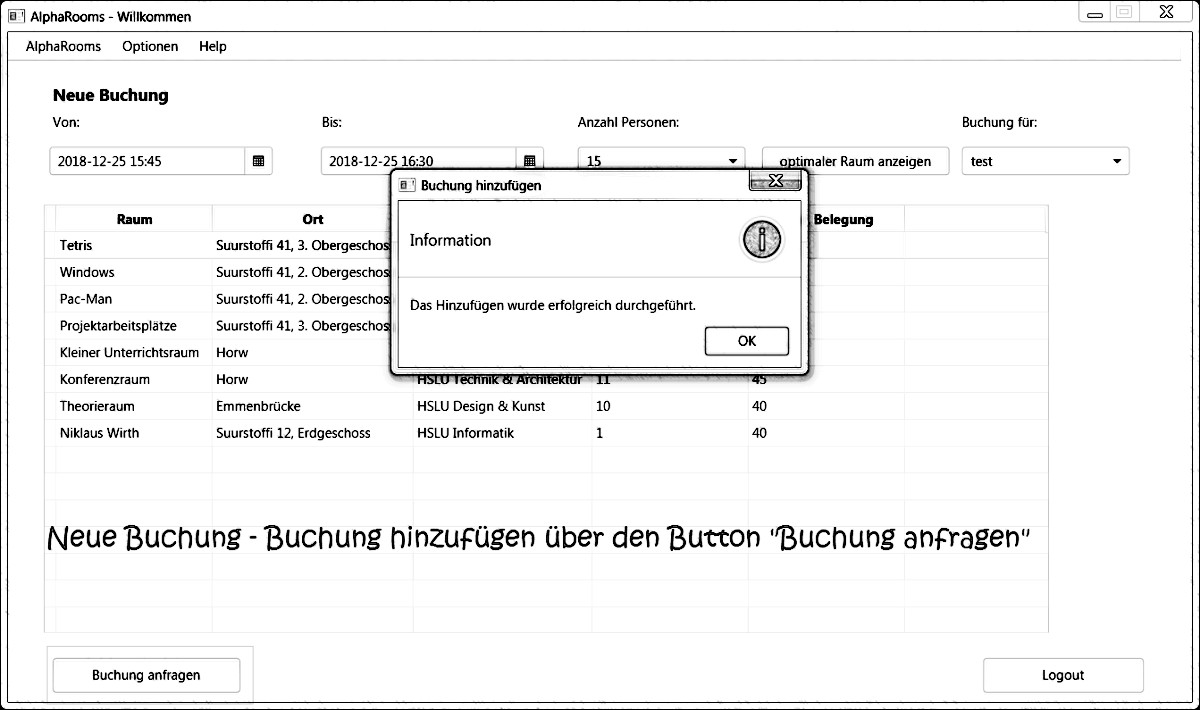


Abbildung 18: Buchung erstellen (AddView.fxml) – UC1

#### Datenstruktur

Die verwendeten Felder sind auf den Tabellen bookings und rooms zu finden. Die Datenstruktur wird als serialisiertes Java Objekte übergeben. Verwendete Datenstruktur:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feld Formular** | **Feld Datenbank** | **Beschreibung** | **Funktion** |
| **Von** | Bk\_from | Startzeit und Datum der Buchung | Beginn Zeitfenster der Buchung |
| **Bis** | BK\_to | Endzeit und Datum der Buchung | Ende Zeitfenster der Buchung |
| **Raum** | Room\_name | Raum Name | Erkennung des Raumes |
| **Anzahl Personen** | Booking\_util | Raumauslastung der Buchung in Personen | Erkennen der Auslastung und Finden der besten Räumen |
| **Buchung für** | Booking\_for | Verantwortlich für die Buchung | Deklaration der Ansprechsperson |
| **Ort** | Room\_location | Ort des Raumes | Den Raum zu finden bei einer Buchung |
| **Departement** | Room\_equipment | Departement des Raumes | Den Raum zu finden bei einer Buchung |
| **Min. Belegung (30%)** | Room\_min\_util | Die Belegung welche für einen Raum mindestens empfohlen wird. | Die Raumauslastung zu erhöhen. |
| **Max. Belegung** | Room\_max\_util | Die Belegung welcher für einen Raum maximal empfohlen wird. | Das eine Räumlichkeit nicht überlastet wird. |

### StatView.fxml

Die Sicht StatView bietet eine Systemweite Statistik über die Buchungen auf entsprechende Räume unterteilt nach der Tageszeit einer Buchung.

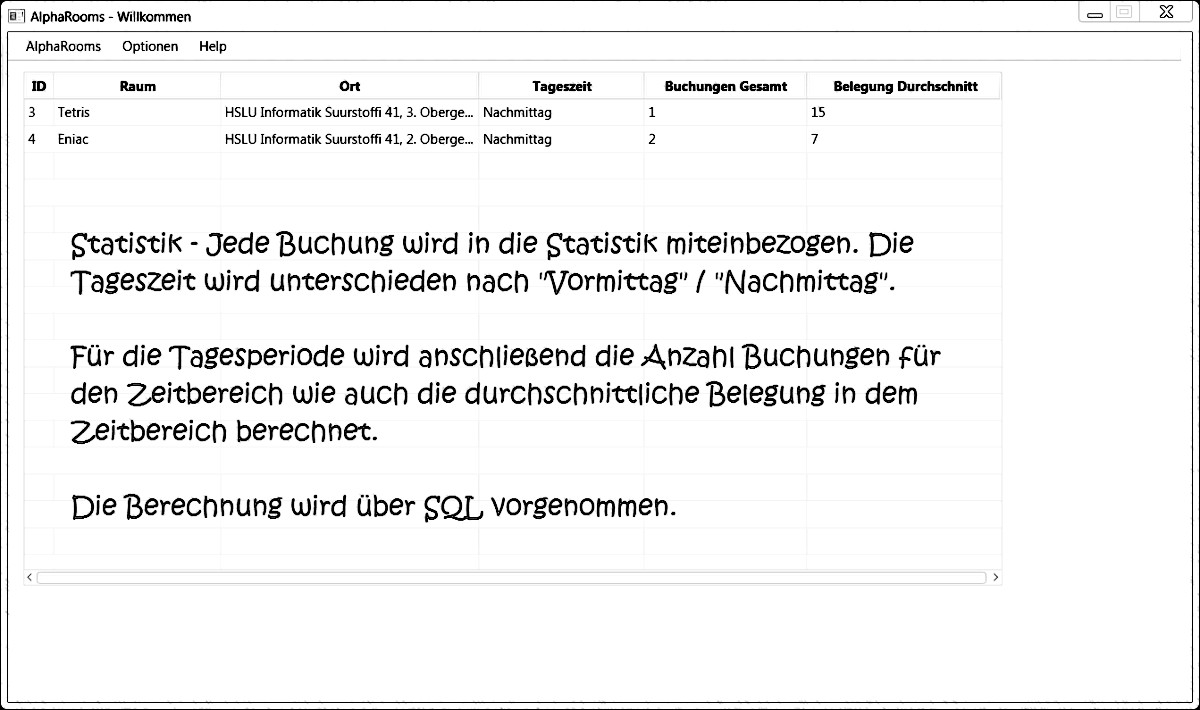


Abbildung 19: Statistik (StatView.fxml) – UC10

#### Datenstruktur

Verwendete Datenstruktur:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feld Formular** | **Feld Datenbank** | **Beschreibung** | **Funktion** |
| **ID** | Room\_id | Room Identifikation | Identifikation des Raumes |
| **Raum** | Room\_name | Raum Name | Erkennung des Raumes |
| **Ort** | Room\_location | Ort des Raumes | Den Raum zu finden bei einer Buchung |
| **Tageszeit** | STMT CALC | Tageszeit (Vormittag / Nachmittag) | Die Auslastung von verschiedenen Tagesperioden zu zeigen. |
| **Buchungen gesamt** | STMT CALC | Alle Buchungen welche zu dieser Tageszeit erstellt wurden | Abschätzen der Raumnutzung zu verschiedenen Zeiten |
| **Belegung Durchschnitt** | STMT CALC | Durchschnittliche Belegung von Buchungen auf dem Raum und Tageszeit | Abschätzen der Raumnutzung zu verschiedenen Zeiten |

### RoomView.fxml

Die Sicht RoomView erlaubt es dem Benutzer von AlphaRooms sämtliche Räumlichkeiten der Applikation zu erblicken wie auch CUD Operationen auszuführen.

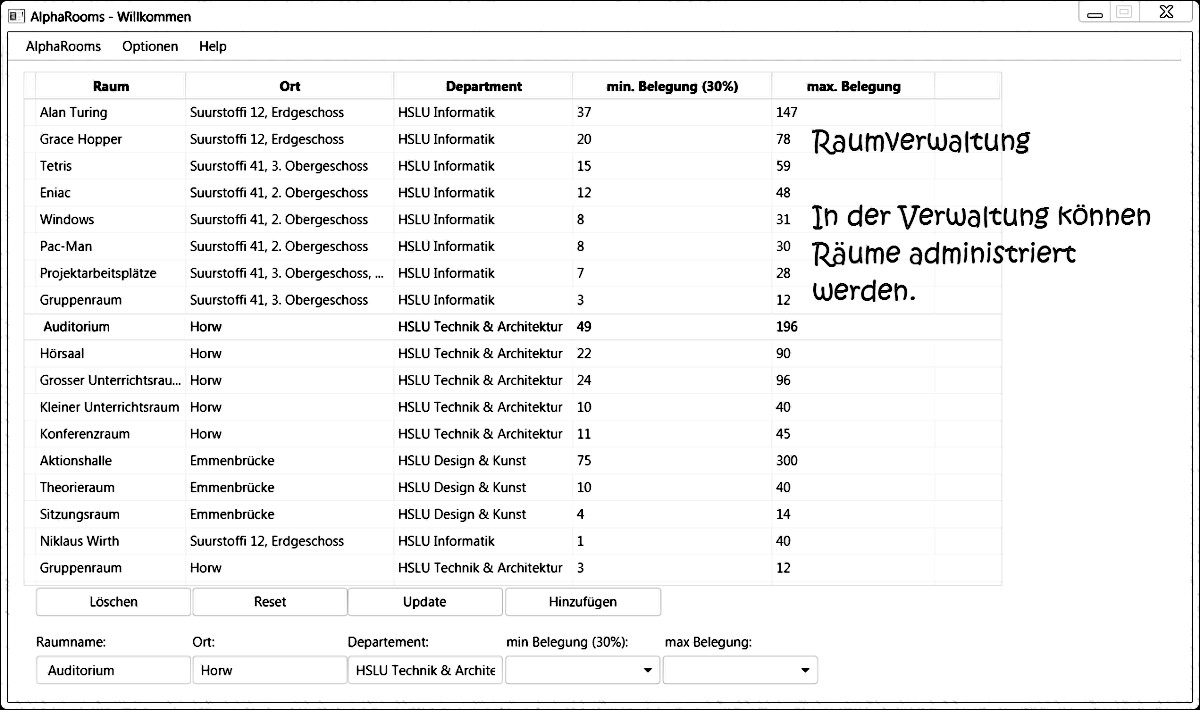


Abbildung 20: Räume verwalten – UC8

#### Datenstruktur

Alle Felder der Datenstruktur sind auf der Tabelle rooms zu finden. Verwendete Datenstruktur:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feld Formular** | **Feld Datenbank** | **Beschreibung** | **Funktion** |
| **Raum** | Room\_name | Raum Name | Erkennung des Raumes |
| **Ort** | Room\_location | Ort des Raumes | Den Raum zu finden bei einer Buchung |
| **Departement** | Room\_equipment | Departement des Raumes | Den Raum zu finden bei einer Buchung |
| **Min. Belegung (30%)** | Room\_min\_util | Die Belegung welche für einen Raum mindestens empfohlen wird. | Die Raumauslastung zu erhöhen. |
| **Max. Belegung** | Room\_max\_util | Die Belegung welcher für einen Raum maximal empfohlen wird. | Das eine Räumlichkeit nicht überlastet wird. |

### UserView.fxml

Die Sicht UserView erlaubt es dem Benutzer von AlphaRooms sämtliche Benutzer der Applikation zu erblicken wie auch CUD Operationen auszuführen.

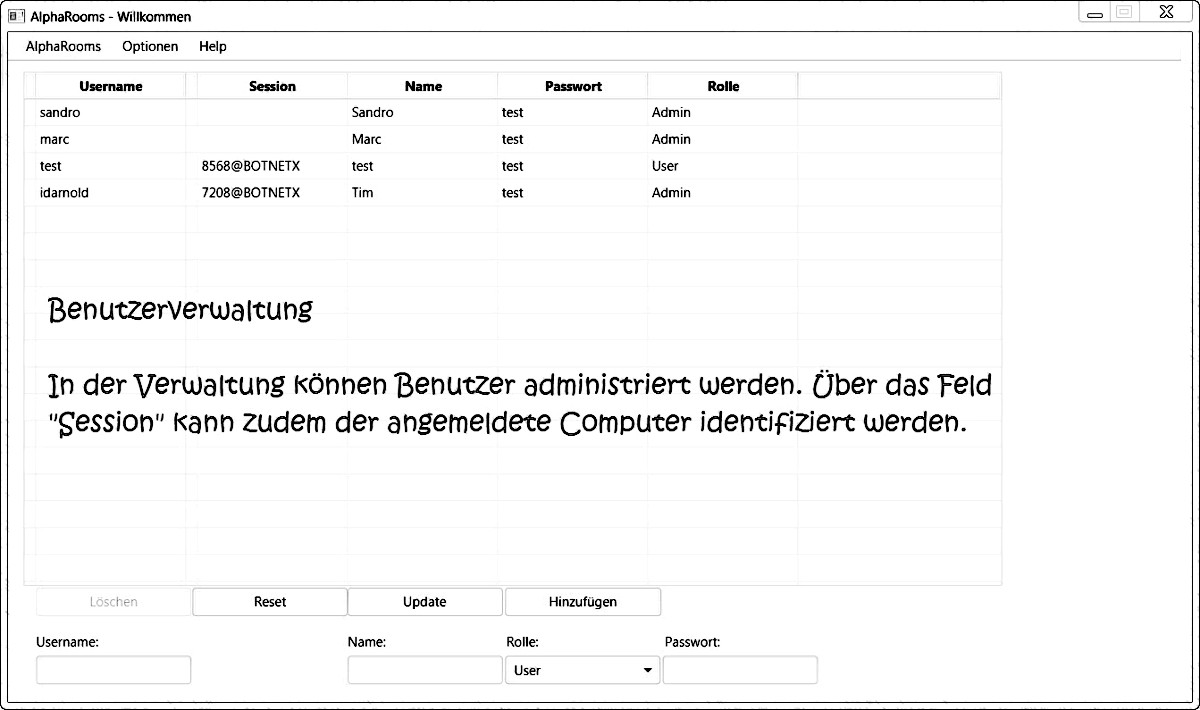


Abbildung 21: Benutzer verwalten – UC8

#### Datenstruktur

Alle Felder der Datenstruktur sind auf der Tabelle users zu finden. Verwendete Datenstruktur:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feld Formular** | **Feld Datenbank** | **Beschreibung** | **Funktion** |
| **Username** | Username | Username | Erkennung des Users |
| **Session** | Firstname | Identifikation der Session und Gerät welches die Anmeldung als letztes getätigt hat | Überblick der Anmeldungen |
| **Name** | Name | Name des Benutzers | Name des Benutzers |
| **Passwort** | Password | Passwort des Benutzers | Passwort des Benutzers |
| **Rolle** | Role | Rolle des Benutzers | Unterscheidung zwischen der Rolle Admin und Benutzer |

## UI für User Story Sekretariat - Rolle Admin

Die Rolle Admin entspricht in der Fallstudie der Sachbearbeiterin im Sekretariat. Sie hat die Möglichkeit Buchungen zu erstellen für andere Benutzer und ebenfalls für alle Benutzer Buchungen zu stornieren. Die unten gezeigte Funktionalität wird auf der AddView zu finden sein.

### AddView.fxml

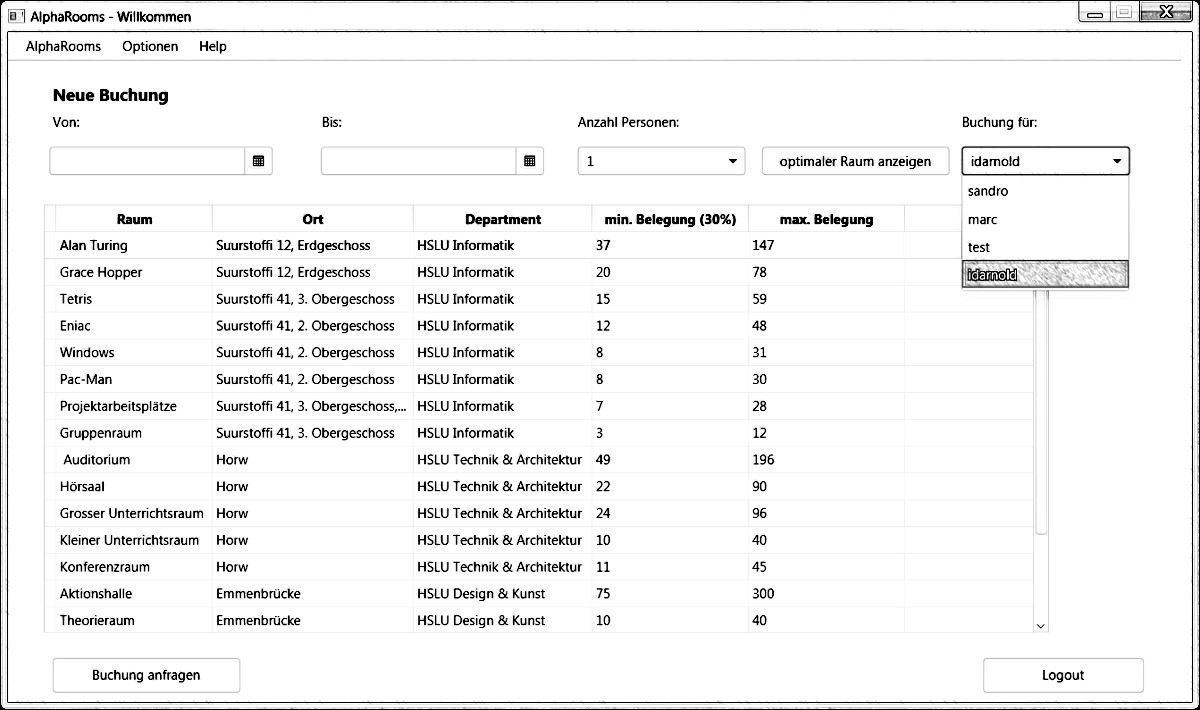


Abbildung 22: Neue Buchung mit anderem Benutzer erstellen (Funktion der Rolle Admin) – UC9

# Interfaces

Im Kapitel 8 Interfaces werden die Business-Interfaces aufgeführt und beschrieben. Es wird erläutert für welche Entitäten das Interface verwendet wird und welche Funktionalitäten das Interface zur Verfügung stellt. Die Abbildungen der Interfaces wurden in Form von UML Diagrammen gemacht.

|  |  |
| --- | --- |
| **UML** | **Beschreibung** |
| Abbildung 8 – RMI\_Booking | ***RMI\_Booking***  Das RMI Interface «RMI\_Booking» liefert sämtliche Buchungen und bietet zudem die Möglichkeit Buchungen zu stornieren. Die Rückverfolgung wird durch den Änderungszeitstempel & Benutzer bewerkstelligt, welcher bei einer Stornierung mitgegeben wird.  Die Methode «createBooking» wird verwendet um ein Booking Objekt direkt in die Datenbank zu schreiben. |
|  |  |
| Abbildung 23 – RMI\_Room | ***RMI\_Room***  Das Interface RMI\_Room bietet vier operative Methoden wie auch drei Methoden für die Raum Verwaltung (CUD Operations).  Die zwei Methoden getOptRooms / geStatRooms liefern die Informationen für die optimale Raum Wahl bzw. sämtliche Statistik Daten.  Die get Methoden dienen zum Austausch von einem Raum bzw. der Raumauflistung. |
|  |  |
| Abbildung 24 – RMI\_User | ***RMI\_User***  Das Interface RMI\_User bietet drei operative Methoden, wie auch drei Methoden für die Benutzer Verwaltung (CUD Operations).  Die Methode checkUserLogin prüft das User Login. Weiter werden der Computername und eine eindeutige Nummer für die Identifizierung der Client Applikation in die Datenbank geschrieben.  Als weitere Optimierung könnten sämtliche Operationen aller Interfaces nur noch mithilfe der Session ID ausgeführt werden. Somit wäre ein Benutzer an ein aktives Login gebunden. Die würde die Sicherheit der Software wesentlich erhöhen. |

# Testen

Das Testkonzept von AlphaRooms bestand im Wesentlichen aus drei Komponenten. Den Proof of Concept Phase einhergehend mit empirischen Tests während dem Entwicklungsvorgang, zwei durchgeführten Unit Test auf Persister Ebene und dem umfänglichen Usability-Test mithilfe der dokumentieren User Stories.

## Unit Test

Durchgeführt wurden bis anhin zwei Unit Test für die Entitäten Rooms und Users. Für eine Beta geschweige einer finalen Version von AlphaRooms mussten sicherlich noch ausführlicher getestet werden.

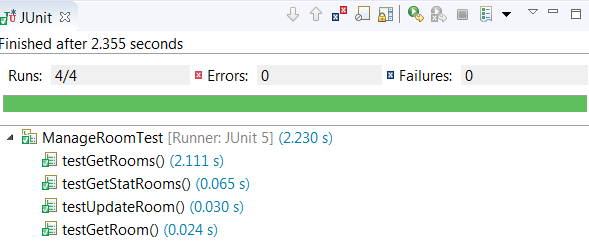


Abbildung 25: JUnit Test Entität Room

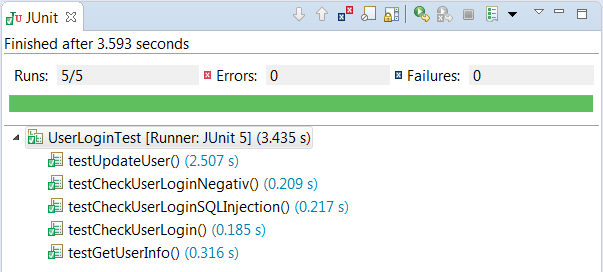


Abbildung 26: JUnit Test Entität User

## Usability-Test - User Story

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case ID:** | **UC1** – Neue Buchung erfassen | Erwartetes Ergebnis | Erhaltenes Ergebnis |
| User Story Description: | Als Lehrperson kann ich einen Raum suchen, damit ich den Raum buchen kann. | Das Suchen und Buchen verläuft fehlerfrei. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC2** – Raum identifizieren |  |  |
| User Story Description: | Als Lehrperson kann ich einen Raum mit meiner gewünschten Belegung suchen, damit ich einen Raum buchen kann, der meiner Gruppengrösse entspricht. | Es werden korrekte Raumvorschläge geliefert, die Einträge entsprechen Gruppengrösse und sind in der Periode verfügbar sind. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC3** – Raum reservieren |  |  |
| User Story Description: | Als Lehrperson kann ich einen Raum reservieren sollte der Raum im gewünschten Zeitraum frei sein. | Eine Buchung auf einen Raum ist fehlerfrei möglich solange der Raum zum Zeitpunkt der Buchung frei ist. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC4** – Raumverfügbarkeit prüfen |  |  |
| User Story Description: | Als Lehrperson kann ich Buchungen einsehen, um zu prüfen welche Person zu welchem Zeitpunkt einen Raum gebucht hat. | Buchungsübersicht wird fehlerfrei geladen. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC5** – Buchung stornieren |  |  |
| User Story Description: | Als Lehrperson kann ich meine Buchungen stornieren, wenn ich verhindert bin. | Buchungen, welche auf die Verantwortlichkeit des aktiven Benutzers laufen können mit Doppelklick storniert werden. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC6** – Buchung stornieren |  |  |
| User Story Description: | Als Lehrperson kann ich mich am System nur anmelden, wenn ich Username und Passwort habe, somit sind meine Daten geschützt. | Eine Anmeldung ist nur möglich mit korrekten Anmeldeinformationen. | Gemäss erwartetem Ergebnis. |

### Sekretariat (Rolle Admin)

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case:** | **UC7** – Raum verwalten | Erwartetes Ergebnis | Erhaltenes Ergebnis |
| User Story Description: | Als Sekretariat kann in einen Raum erfassen, damit der Raum gebucht werden kann. | Ein Raum kann erfasst werden ohne Fehlermeldung. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC8** – Benutzer verwalten |  |  |
| User Story Description: | Als Sekretariat kann in einen Benutzer erfassen, damit der Benutzer sich anmelden kann. | Ein Benutzer kann erfasst werden ohne Fehlermeldung. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC9** – Umbuchen |  |  |
| User Story Description: | Als Sekretariat kann ich alle Arten von Buchungen, erstellen und stornieren, um den Abtausch von Räumen zu ermöglichen. | Ein Admin Benutzer kann Buchungen für andere Personen ohne Probleme erstellen. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC10** – Statistik abrufen |  |  |
| User Story Description: | Als Sekretariat kann ich Statistiken über die Raumbelegung abrufen, um zu sehen wann Räume am wenigsten belegt sind. | Die Statistik ist aktuell und kann ohne Fehler angezeigt werden. | Gemäss erwartetem Ergebnis. |
| **Use Case:** | **UC11** – Anmelden |  |  |
| User Story Description: | Als Sekretariat kann ich mich am System nur anmelden, wenn ich Username und Passwort habe, somit sind meine Daten geschützt. | User und Passwort werden zuverlässig geprüft im und bei fehlerhaften Eingaben wird der Anmeldevorgang unterbrochen. | Gemäss erwartetem Ergebnis. |

## Zusatz

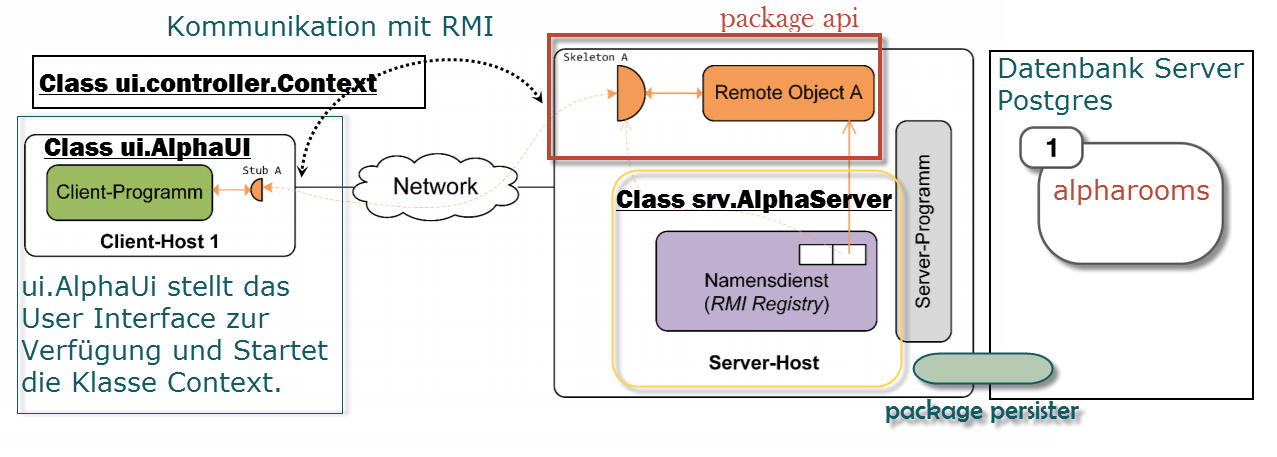
|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case:** | **UC5** – Buchung stornieren | Erwartetes Ergebnis | Erhaltenes Ergebnis |
| User Story Description: | Als Lehrperson kann ich einen Raum reservieren, der durch eine stornierte Buchung war welche nun storniert ist, weil ein Abtausch gemacht wurde. | Der Zeitbereich lässt sich fehlerfrei buchen. | Gemäss erwartetem Ergebnis. |

# Deployment

## Architekturbeschreibung

Die einzelnen Module sind wie in der Beschreibung auf einem Client, sowie auf dem Server verteilt. Die Verteilung wiederum bestehen aus mehreren Packages. Diese sind wie in der Grafik ersichtlich in mehrere Sektionen eingeteilt.

**Server**

**Client**

Im produktiven Einsatz werden die Klassen srv.AlphaServer (Server) und ui.AlphaUi (Client) gestartet.

### Server

Der Server erstellt über die «AlphaServer» Klasse einen Skeleton der Klassen im Package «srv». Das Skeleton wird anschliessend von RMI verwendet, um die über die Interfaces im Package «api» zur Verfügung gestellten Methoden, Informationen abzurufen und zu senden.

In RMI ist es nur möglich serialisierte Informationen zu übertragen. Deshalb wird diese Serialisierung im Package «api» automatisch vorgenommen.

Die von RMI über die Skeletons angesprochenen Klassen im Package «srv», welche auch die Klasse srv.AlphaServer enthält, geben anschliessend die Anfragen weiter an das Package «persister». Das Package «persister» ist ein JPA Projekt welche die Anfragen verarbeitet und die Kommunikation mit der Datenbank «alpharooms» über SQL sicherstellt.

### Client

Der Client startet über die Klasse «AlphaUi» das grafische Benutzer Interface, wie auch die «Context» Klasse im Package «ui.controller». Die Klasse «Context» baut die RMI Stubs auf, um die vom Server zur Verfügung gestellten Interfaces anzusprechen.

Sobald der Client das UI auf dem Computer aufgebaut hat nutzen die Controller Klassen die von der «Context» Klasse zur Verfügung gestellten Methoden, welche sich auf dem Server befinden.

# Konfigurationsangaben

In diesem Kapitel sind alle nötigen Angaben enthalten, die ein Administrator für die Installation und Inbetriebnahme von AlphaRooms benötigt.

## Voraussetzungen

AlphaRooms benötigt je nach Komponente (Server / Client) andere Voraussetzungen. Das Programm AlphaRooms kann über folgenden Link (**sixxt.com/build.zip**) mit Install Scripts und Konfigurationsdateien bezogen werden.

### Server

Stellen Sie sicher das folgende Software installiert ist und die weiteren Bedingungen erfüllt sind.

1. Installierte Java Version JRE 10.0.2
2. Installierter POSTGRES SQL Server Version 9.6 (Konfiguriert auf Port 5433)
3. Konfiguriertes PGAdmin Tool (Postgres Server auf Port 5433)
4. Einsatzbereite Datenbank «alpharooms» auf POSTGRES Server (Einspielt über Backup Datei / Script)
5. Statische IP Adresse
6. Beziehen Sie die AlphaRoomsServer über den oben genannten Link

### Client

Stellen Sie sicher das folgende Software installiert ist und die weiteren Bedingungen erfüllt sind.

1. Installierte Java Version JRE 10.0.2
2. Beziehen Sie die AlphaRoomsClient über den oben genannten Link
3. Konfigurierte «rmi\_client.properties» Datei im User Profil des Benutzers (statischen IP des Servers)

## Credentials

1. Verwendetes AlphaRooms SQL Server Benutzer Login:
   1. User: alpha
   2. Passwort: alpha
2. Standard Benutzer für die erste Anmeldung:
   1. User: test
   2. Passwort: test

### Installation Server

1. Installieren Sie POSTGRES in der Version 9.6
2. Installieren Sie pgAdmin4
3. Wechseln Sie in das Verzeichnis C:\Program Files\PostgreSQL\9.6\data und ändern Sie die Datei postgresql.conf
   1. Setzen Sie Postgres auf den Port **5433**
   2. Erhöhen Sie den Paramter max\_connection auf **9999**.
4. Starten Sie den Postgres Server neu. Sollten Sie nicht wissen wie können Sie auch den Server neustarten.

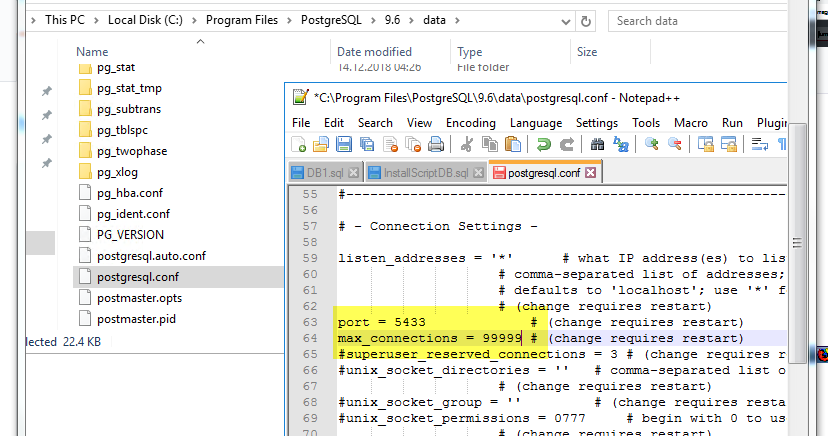


Abbildung 27: Config postgresql.conf

1. Starten Sie pgAdmin4 und erstellen Sie eine neue Datenbank «alpharooms»

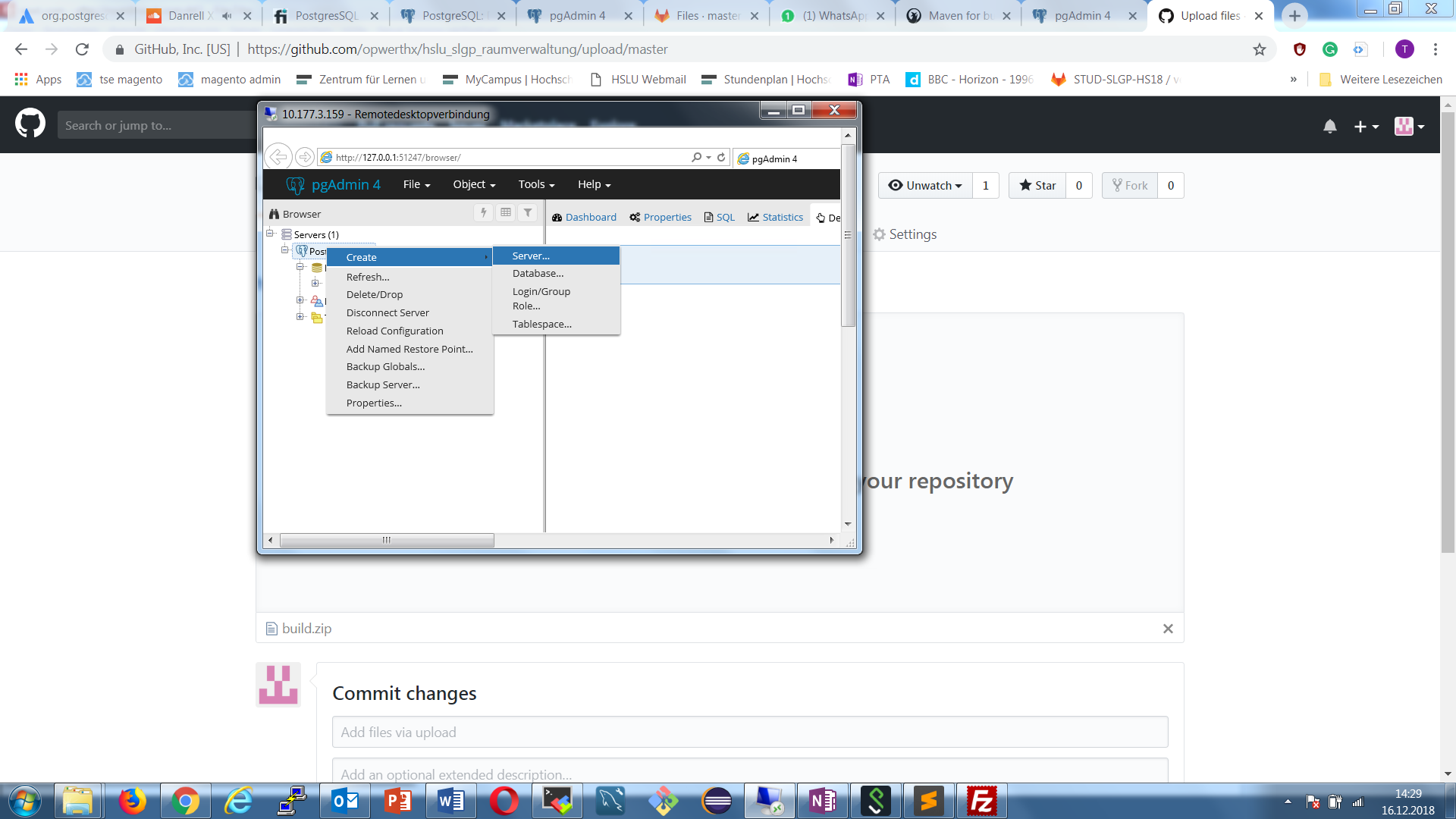


Abbildung 28: Create Database PGAdmin

1. Selektieren Sie die neue Datenbank «alpharooms» und öffnen Sie über Tools das «Query Tool»

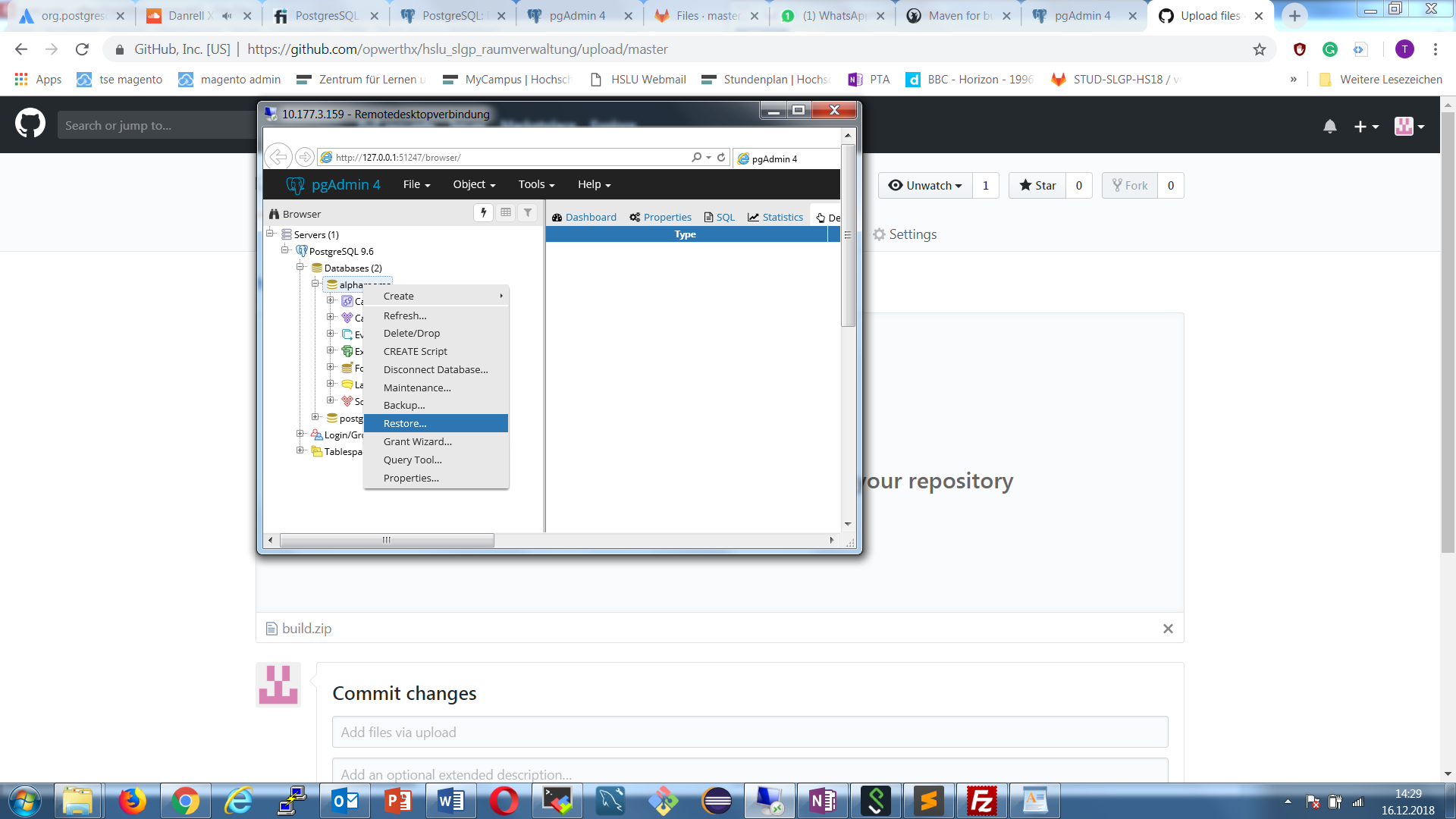


Abbildung 29: Query Tool

1. Kopieren Sie den Inhalt des Files «InstallScriptDB.sql» in das Query Window und führen Sie die Befehle über das Blitz Symbol aus.
2. Melden Sie sich von PGAdmin ab.
3. Starten Sie im heruntergeladenen Ordner «build» das Programm «LaunchServer.bat»

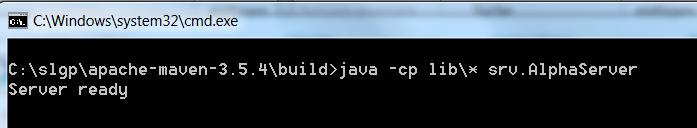


Abbildung 30: Server ready

1. Der AlphaRooms Server ist nun bereit.

### Installation Client

1. Laden Sie die Datei «build.zip» vom Link «**sixxt.com/build.zip»** herunter und entpacken Sie das Zip.
2. Ändern Sie die Datei «rmi\_client.properties» und ersetzten 127.0.0.1 mit der IP Ihres AlphaRooms Server. Speicher Sie die Datei.

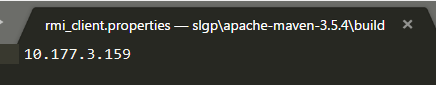


Abbildung 31: Konfigurationsdatei rmi\_client.properties

1. Ändern Sie die Datei «rmi\_client.properties» und ersetzten 127.0.0.1 mit der IP Ihres AlphaRooms Server. Speicher Sie die Datei.
2. Führen Sie das Batchscript «InstallClient.bat», es kopiert die bearbeitete Datei «rmi\_client.properties» in Ihr Benutzerverezeichnis (%USERPROFILE%).
3. Starten Sie AlphaRoomsUI über das Batchscript «LaunchClient.bat» und melden sich mit User: test und Passwort «test» an.
4. Konfigurieren Sie Benutzer und Räume.
5. Glückwunsch Ihr neues AlphaRooms System ist nun konfiguriert und bereit für Ihre Benutzer!

## Database Creator (Scrip) – Manual Install / With SampleData

|  |
| --- |
| --  -- PostgreSQL database dump  --  -- Dumped from database version 11.1  -- Dumped by pg\_dump version 11.1  -- Started on 2018-12-15 16:20:54  -- SEQUENCE 1  SET statement\_timeout = 0;  SET lock\_timeout = 0;  SET idle\_in\_transaction\_session\_timeout = 0;  SET client\_encoding = 'UTF8';  SET standard\_conforming\_strings = on;  SELECT pg\_catalog.set\_config('search\_path', '', false);  SET check\_function\_bodies = false;  SET client\_min\_messages = warning;  SET row\_security = off;  -- SEQUENCE 2  DROP DATABASE alpharooms;  --  -- TOC entry 2865 (class 1262 OID 13012)  -- Name: postgres; Type: DATABASE; Schema: -; Owner: postgres  --  -- SEQUENCE 3  CREATE DATABASE alpharooms WITH TEMPLATE = template0 ENCODING = 'UTF8' LC\_COLLATE = 'German\_Switzerland.1252' LC\_CTYPE = 'German\_Switzerland.1252';  -- SEQUENCE 4  ALTER DATABASE alpharooms OWNER TO postgres;  \connect alpharooms  SET statement\_timeout = 0;  SET lock\_timeout = 0;  SET idle\_in\_transaction\_session\_timeout = 0;  SET client\_encoding = 'UTF8';  SET standard\_conforming\_strings = on;  SELECT pg\_catalog.set\_config('search\_path', '', false);  SET check\_function\_bodies = false;  SET client\_min\_messages = warning;  SET row\_security = off;  ---- INSTALL SCRIPT USE DB  -- SEQUENCE 5  DROP USER alpha;  CREATE USER alpha WITH  LOGIN  SUPERUSER  INHERIT  NOCREATEDB  CREATEROLE  NOREPLICATION  CONNECTION LIMIT -1  PASSWORD 'alpha';  -- Table: public.sequence  DROP TABLE public.sequence;  CREATE TABLE public.sequence  (  seq\_name character varying(50) COLLATE pg\_catalog."default" NOT NULL,  seq\_count numeric(38,0),  CONSTRAINT sequence\_pkey PRIMARY KEY (seq\_name)  )  WITH (  OIDS = FALSE  )  TABLESPACE pg\_default;  ALTER TABLE public.sequence  OWNER to postgres;    CREATE TABLE public.configs  (  config\_id SERIAL not null,  area character varying(45) COLLATE pg\_catalog."default",  config character varying(100) COLLATE pg\_catalog."default",  value character varying(100) COLLATE pg\_catalog."default",  sort integer,  CONSTRAINT configs\_pkey PRIMARY KEY (config\_id)  )  WITH (  OIDS = FALSE  )  TABLESPACE pg\_default;  ALTER TABLE public.configs  OWNER to postgres;  -- Table: public.rooms  -- DROP TABLE public.rooms;  CREATE TABLE public.rooms  (  row\_flag character varying(8) COLLATE pg\_catalog."default",  room\_id SERIAL not null,  room\_name character varying(45) COLLATE pg\_catalog."default",  room\_location character varying(45) COLLATE pg\_catalog."default",  room\_equipment character varying(45) COLLATE pg\_catalog."default",  room\_min\_util integer,  room\_max\_util integer,  CONSTRAINT rooms\_pkey PRIMARY KEY (room\_id)  )  WITH (  OIDS = FALSE  )  TABLESPACE pg\_default;  ALTER TABLE public.rooms  OWNER to postgres;  -- Table: public.users  -- DROP TABLE public.users;  CREATE TABLE public.users  (  user\_id SERIAL not null,  row\_flag character varying(8) COLLATE pg\_catalog."default",  username character varying(16) COLLATE pg\_catalog."default" NOT NULL,  email character varying(255) COLLATE pg\_catalog."default",  password character varying(32) COLLATE pg\_catalog."default" NOT NULL,  firstname character varying(255) COLLATE pg\_catalog."default",  name character varying(255) COLLATE pg\_catalog."default",  create\_time timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,  role character varying(60) COLLATE pg\_catalog."default",  c\_ts timestamp without time zone,  c\_user character varying(16) COLLATE pg\_catalog."default",  m\_ts timestamp without time zone,  m\_user character varying(16) COLLATE pg\_catalog."default",  CONSTRAINT users\_pkey PRIMARY KEY (user\_id),  CONSTRAINT users\_username\_key UNIQUE (username)  )  WITH (  OIDS = FALSE  )  TABLESPACE pg\_default;  ALTER TABLE public.users  OWNER to postgres;    CREATE TABLE public.bookings  (  booking\_id SERIAL not null,  row\_flag character varying(20) COLLATE pg\_catalog."default",  booking\_for character varying(16) COLLATE pg\_catalog."default",  booking\_util integer,  id\_room integer,  c\_user character varying(16) COLLATE pg\_catalog."default",  c\_ts timestamp without time zone,  m\_user character varying(16) COLLATE pg\_catalog."default",  m\_ts timestamp without time zone,  bk\_from timestamp without time zone,  bk\_to timestamp without time zone,  CONSTRAINT bookings\_pkey PRIMARY KEY (booking\_id),  CONSTRAINT fk\_booking\_for FOREIGN KEY (booking\_for)  REFERENCES public.users (username) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT fk\_create\_user FOREIGN KEY (c\_user)  REFERENCES public.users (username) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT fk\_modify\_user FOREIGN KEY (m\_user)  REFERENCES public.users (username) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT fk\_room FOREIGN KEY (id\_room)  REFERENCES public.rooms (room\_id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  )  WITH (  OIDS = FALSE  )  TABLESPACE pg\_default;  ALTER TABLE public.bookings  OWNER to postgres;  -- Table: public.configs  -- DROP TABLE public.configs;  --  --  -- TOC entry 2852 (class 0 OID 41214)  -- Dependencies: 198  -- Data for Name: configs; Type: TABLE DATA; Schema: public; Owner: postgres  --  INSERT INTO public.configs VALUES (101, 'TEST', 'LOGINTYPE', 'PLAIN', 10);  --  -- TOC entry 2856 (class 0 OID 41236)  -- Dependencies: 202  -- Data for Name: rooms; Type: TABLE DATA; Schema: public; Owner: postgres  --  INSERT INTO public.rooms VALUES ('ACTIVE', 1, 'Alan Turing', 'Suurstoffi 12, Erdgeschoss', 'HSLU Informatik', 37, 147);  INSERT INTO public.rooms VALUES ('ACTIVE', 2, 'Grace Hopper', 'Suurstoffi 12, Erdgeschoss', 'HSLU Informatik', 20, 78);  INSERT INTO public.rooms VALUES ('ACTIVE', 3, 'Tetris', 'Suurstoffi 41, 3. Obergeschoss', 'HSLU Informatik', 15, 59);  INSERT INTO public.rooms VALUES ('ACTIVE', 4, 'Eniac', 'Suurstoffi 41, 2. Obergeschoss', 'HSLU Informatik', 12, 48);  INSERT INTO public.rooms VALUES ('ACTIVE', 5, 'Windows', 'Suurstoffi 41, 2. Obergeschoss', 'HSLU Informatik', 8, 31);  INSERT INTO public.rooms VALUES ('ACTIVE', 6, 'Pac-Man', 'Suurstoffi 41, 2. Obergeschoss', 'HSLU Informatik', 8, 30);  INSERT INTO public.rooms VALUES ('ACTIVE', 7, 'Projektarbeitsplätze', 'Suurstoffi 41, 3. Obergeschoss, Nr. 302', 'HSLU Informatik', 7, 28);  INSERT INTO public.rooms VALUES ('ACTIVE', 8, 'Gruppenraum', 'Suurstoffi 41, 3. Obergeschoss', 'HSLU Informatik', 3, 12);  INSERT INTO public.rooms VALUES ('ACTIVE', 9, ' Auditorium', 'Horw ', 'HSLU Technik & Architektur', 49, 196);  INSERT INTO public.rooms VALUES ('ACTIVE', 10, 'Hörsaal', 'Horw ', 'HSLU Technik & Architektur', 22, 90);  INSERT INTO public.rooms VALUES ('ACTIVE', 11, 'Grosser Unterrichtsraum', 'Horw ', 'HSLU Technik & Architektur', 24, 96);  INSERT INTO public.rooms VALUES ('ACTIVE', 12, 'Kleiner Unterrichtsraum', 'Horw ', 'HSLU Technik & Architektur', 10, 40);  INSERT INTO public.rooms VALUES ('ACTIVE', 13, 'Konferenzraum', 'Horw ', 'HSLU Technik & Architektur', 11, 45);  INSERT INTO public.rooms VALUES ('ACTIVE', 15, 'Aktionshalle', 'Emmenbrücke ', 'HSLU Design & Kunst ', 75, 300);  INSERT INTO public.rooms VALUES ('ACTIVE', 16, 'Theorieraum', 'Emmenbrücke ', 'HSLU Design & Kunst ', 10, 40);  INSERT INTO public.rooms VALUES ('ACTIVE', 17, 'Sitzungsraum', 'Emmenbrücke ', 'HSLU Design & Kunst ', 4, 14);  INSERT INTO public.rooms VALUES ('ACTIVE', 19, 'Niklaus Wirth', 'Suurstoffi 12, Erdgeschoss', 'HSLU Informatik', 1, 40);  INSERT INTO public.rooms VALUES ('ACTIVE', 14, 'Gruppenraum', 'Horw', 'HSLU Technik & Architektur', 3, 12);  --  -- TOC entry 2859 (class 0 OID 41270)  -- Dependencies: 205  -- Data for Name: sequence; Type: TABLE DATA; Schema: public; Owner: postgres  --  INSERT INTO public.sequence VALUES ('SEQ\_GEN\_TABLE', 4200);  --  -- TOC entry 2854 (class 0 OID 41222)  -- Dependencies: 200  -- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres  --  INSERT INTO public.users VALUES (1002, NULL, 'sandro', '', 'test', NULL, 'Sandro', NULL, 'Admin', NULL, NULL, NULL, NULL);  INSERT INTO public.users VALUES (1003, NULL, 'marc', NULL, 'test', NULL, 'Marc', '2018-12-10 00:25:35.495', 'Admin', '2018-12-10 00:25:35.495', NULL, '2018-12-10 00:25:35.495', NULL);  INSERT INTO public.users VALUES (201, 'test', 'test', 'idarnold', 'test', '3204@BOTNETX', 'test', '2018-12-11 19:04:09.442', 'User', '2018-12-11 19:04:09.442', 'idarnold', '2018-12-11 19:04:09.442', 'idarnold');  INSERT INTO public.users VALUES (1, 'test', 'idarnold', 'idarnold', 'test', '8568@BOTNETX', 'Tim', '2018-12-11 19:04:14.243', 'Admin', '2018-12-11 19:04:14.243', 'idarnold', '2018-12-11 19:04:14.243', 'idarnold');  -- TOC entry 2858 (class 0 OID 41244)  -- Dependencies: 204  -- Data for Name: bookings; Type: TABLE DATA; Schema: public; Owner: postgres  --  INSERT INTO public.bookings VALUES (1851, 'STORNIERT', 'test', 1, 4, 'test', '2018-12-12 17:43:39.619', 'test', '2018-12-13 15:39:12.508', '2018-12-25 15:00:00', '2018-12-25 16:00:00');  INSERT INTO public.bookings VALUES (1901, 'Hinzugefügt', 'test', 13, 4, 'test', '2018-12-13 15:40:29.549', 'test', '2018-12-13 15:40:29.549', '2018-12-25 15:40:00', '2018-12-25 20:00:00');  INSERT INTO public.bookings VALUES (1951, 'Hinzugefügt', 'test', 15, 3, 'test', '2018-12-13 15:43:43.706', 'test', '2018-12-13 15:43:43.706', '2018-12-25 15:45:00', '2018-12-25 16:30:00');  INSERT INTO public.bookings VALUES (2001, 'STORNIERT', 'test', 7, 17, 'test', '2018-12-15 00:12:42.635', 'test', '2018-12-15 00:12:52.051', '2018-12-10 08:00:00', '2018-12-11 17:00:00');  INSERT INTO public.bookings VALUES (2101, 'Hinzugefügt', 'test', 7, 19, 'test', '2018-12-15 00:24:16.354', 'test', '2018-12-15 00:24:16.354', '2018-12-11 08:00:00', '2018-12-16 12:00:00');  --  -- TOC entry 2872 (class 0 OID 0)  -- Dependencies: 203  -- Name: bookings\_booking\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres  --  SELECT pg\_catalog.setval('public.bookings\_booking\_id\_seq', 1, false);  --  -- TOC entry 2873 (class 0 OID 0)  -- Dependencies: 197  -- Name: configs\_config\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres  --  SELECT pg\_catalog.setval('public.configs\_config\_id\_seq', 1, false);  --  -- TOC entry 2874 (class 0 OID 0)  -- Dependencies: 201  -- Name: rooms\_room\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres  --  SELECT pg\_catalog.setval('public.rooms\_room\_id\_seq', 19, true);  --  -- TOC entry 2875 (class 0 OID 0)  -- Dependencies: 199  -- Name: users\_user\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres  --  SELECT pg\_catalog.setval('public.users\_user\_id\_seq', 1, false);  -- Completed on 2018-12-15 16:20:55  GRANT ALL ON SEQUENCE public.bookings\_booking\_id\_seq TO alpha WITH GRANT OPTION;  GRANT ALL ON SEQUENCE public.configs\_config\_id\_seq TO alpha WITH GRANT OPTION;  GRANT ALL ON SEQUENCE public.rooms\_room\_id\_seq TO alpha WITH GRANT OPTION;  GRANT ALL ON SEQUENCE public.users\_user\_id\_seq TO alpha WITH GRANT OPTION;  GRANT ALL ON TABLE public.bookings TO alpha WITH GRANT OPTION;  GRANT ALL ON TABLE public.configs TO alpha WITH GRANT OPTION;  GRANT ALL ON TABLE public.rooms TO alpha WITH GRANT OPTION;  GRANT ALL ON TABLE public.sequence TO alpha WITH GRANT OPTION;  GRANT ALL ON TABLE public.users TO alpha WITH GRANT OPTION; |

# Gemachte Erfahrungen (lessons learned)

Die technische Umsetzung war eine Herausforderung an sich. Folgende Dinge hätten den Projektablauf effizienter gestaltet:

* Spannend für eine nächste Umsetzung wäre die Verwendung von WhatsApp als UI Provider und einem Java Backend. Bei der richtigen Vorgehensweise und eventuell einer anderen Teamkonstellation hätten wir diese Funktionalität gerne der Klasse präsentiert.
* Die zur Verfügung gestellten Demoprojekte besser analysieren. Bestenfalls gleich im Team, um ein gemeinsames Verständnis zu erlangen.
* Die Nutzung der Context Klasse zu Beginn verstehen, um die Best Practice Ansätze besser zu verfolgen.
* Komplexe Abfragen in die Datenbank sind mit JPA nur aufwändig bis gar nicht möglich. Das wir hier eine String Konkatenierung zum Einsatz gebracht werden musste überraschte Uns sehr.
* RMI ist eine geniale Technologie, welche mithilfe der einfachen Serialisierung in Java bestens genutzt werden kann.
* Das Session und Connection Handling in JPA und Postgres erfordern erhöhte Aufmerksamkeit. Die Programmierung war fehlerfrei jedoch haben uns die Limitierungen, welche standardmässig sehr tief sind überrascht. Wir würden uns beim nächsten Mal nicht automatisch auf die Normaleinstellungen verlassen.