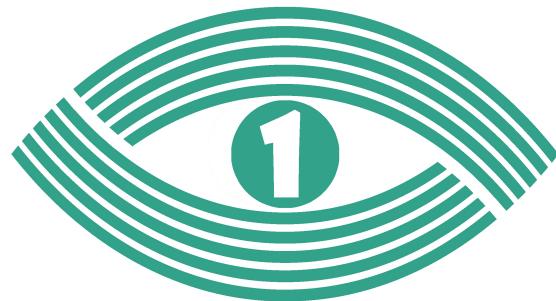


Entwurfsdokument zu

# 1Sicht



Erarbeitet von:

**Martin Zahariev, Maximilian Ackermann,  
Stanimir Bozhilov, Tihomir Georgiev, Hristo Klechorov  
und Andry Niclas**

PSE im Sommersemester 2019

IPD Reussner und IPD Koziolek  
Betreut von: Erik Burger, Sandro Koch und Jan Keim

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>5</b>
<b>2 Architektur</b>	<b>6</b>
2.1 Architektur der App . . . . .	6
2.2 Architektur des Servers . . . . .	8
2.3 Architektur der Datenbank . . . . .	11
<b>3 Dateiorganisation</b>	<b>12</b>
3.1 Pakete . . . . .	12
3.2 JSON - Dateien . . . . .	13
<b>4 Benutzte Frameworks</b>	<b>15</b>
4.1 Ktor  . . . . .	15
4.2 Android Jetpack  . . . . .	15
4.3 Retrofit  . . . . .	15
4.4 MySQL  . . . . .	15
4.5 SQLite  . . . . .	15
4.6 Exposed  . . . . .	16
4.7 Firebase  . . . . .	16
<b>5 Paketübersicht - App</b>	<b>17</b>
5.1  repository . . . . .	17
5.2  networking . . . . .	18
5.3  database . . . . .	19
5.3.1  database.entities . . . . .	20
5.3.2  database.daos . . . . .	21
5.4  ui . . . . .	22
5.4.1  ui.administrator . . . . .	23
5.4.2  ui.student . . . . .	24
5.4.3  ui.employee . . . . .	25
5.4.4  ui.shared . . . . .	26
<b>6 Klassenübersicht - App</b>	<b>27</b>
6.1  networking . . . . .	27
6.1.1  StudentService . . . . .	27
6.1.2  EmployeeService . . . . .	28
6.1.3  AdminService . . . . .	29
6.1.4  ReviewService . . . . .	29
6.1.5  RetrofitClient . . . . .	31
6.2  database . . . . .	32
6.2.1  Database . . . . .	32

6.3	database.entities . . . . .	33
6.3.1	Administrator . . . . .	33
6.3.2	Student . . . . .	33
6.3.3	Employee . . . . .	34
6.3.4	Review . . . . .	35
6.3.5	Timeslot . . . . .	36
6.3.6	StudentTimeslot . . . . .	37
6.4	repository . . . . .	38
6.4.1	AdminRepository . . . . .	38
6.4.2	EmployeeRepository . . . . .	39
6.4.3	StudentRepository . . . . .	40
6.4.4	ReviewRepository . . . . .	41
6.4.5	TimeslotRepository . . . . .	43
6.5	database.daos . . . . .	44
6.5.1	AdministratorDao . . . . .	44
6.5.2	StudentDao . . . . .	45
6.5.3	EmployeeDao . . . . .	46
6.5.4	ReviewDao . . . . .	46
6.5.5	TimeslotDao . . . . .	47
6.5.6	StudentTimeslotDao . . . . .	48
6.6	ui.shared . . . . .	49
6.6.1	LogInActivity . . . . .	49
6.6.2	LogInViewModel . . . . .	50
6.6.3	RegisterActivity . . . . .	50
6.6.4	RegisterViewModel . . . . .	51
6.6.5	SettingsActivity . . . . .	51
6.6.6	SettingsViewModel . . . . .	52
6.6.7	EmployeeAdapter . . . . .	53
6.6.8	StudentAdapter . . . . .	53
6.6.9	ReviewsAdapter . . . . .	54
6.7	ui.administrator . . . . .	55
6.7.1	HomeActivity . . . . .	55
6.7.2	ListRequestsActivity . . . . .	56
6.7.3	ListRequestsViewModel . . . . .	57
6.7.4	ListReviewActivity . . . . .	57
6.7.5	ListReviewViewModel . . . . .	58
6.7.6	ListUserActivity . . . . .	59
6.7.7	ListUserViewModel . . . . .	60
6.8	ui.employee . . . . .	61
6.8.1	HomeEmployeeActivity . . . . .	61
6.8.2	HomeEmployeeViewModel . . . . .	62
6.8.3	WaitingScreenActivity . . . . .	62
6.8.4	ReviewCreationActivity . . . . .	64
6.8.5	ReviewCreationViewModel . . . . .	65

6.8.6	(C) ReviewScreenActivity . . . . .	67
6.8.7	(C) ReviewScreenViewModel . . . . .	67
6.9	ui.student . . . . .	69
6.9.1	(C) HomeStudentActivity . . . . .	69
6.9.2	(C) HomeStudentViewModel . . . . .	70
6.9.3	(C) ReviewSignUpViewModel . . . . .	71
6.9.4	(C) ReviewScreenActivity . . . . .	72
6.9.5	(C) ReviewScreenViewModel . . . . .	72
<b>7</b>	<b>Paketübersicht - Server</b>	<b>74</b>
7.1	controllers . . . . .	74
7.2	repository . . . . .	75
7.3	db . . . . .	76
7.3.1	db.tables . . . . .	77
7.3.2	db.models . . . . .	78
<b>8</b>	<b>Klassenübersicht - Server</b>	<b>79</b>
8.1	(C) Application . . . . .	79
8.2	controllers . . . . .	79
8.2.1	(C) AdminController . . . . .	79
8.2.2	(C) EmployeeController . . . . .	79
8.2.3	(C) StudentController . . . . .	80
8.2.4	(C) ReviewController . . . . .	80
8.3	repository . . . . .	80
8.3.1	(C) AdminRepository . . . . .	80
8.3.2	(C) EmployeeRepository . . . . .	81
8.3.3	(C) StudentRepository . . . . .	82
8.3.4	(C) ReviewRepository . . . . .	83
8.3.5	(C) TimeslotRepository . . . . .	84
8.4	db . . . . .	85
8.4.1	(C) Database . . . . .	85
8.5	db.models . . . . .	86
8.5.1	(C) Admin . . . . .	86
8.5.2	(C) Employee . . . . .	86
8.5.3	(C) Student . . . . .	87
8.5.4	(C) Review . . . . .	87
8.5.5	(C) Timeslot . . . . .	88
8.5.6	(C) StudentTimeslot . . . . .	89
8.6	db.tables . . . . .	90
8.6.1	(C) Admins . . . . .	90
8.6.2	(C) Employees . . . . .	90
8.6.3	(C) Students . . . . .	91
8.6.4	(C) Reviews . . . . .	91
8.6.5	(C) Timeslots . . . . .	92



---

8.6.6	StudentsTimeslots . . . . .	93
<b>9</b>	<b>Diagramme</b>	<b>94</b>
9.1	Systemsequenzdiagramme . . . . .	94
9.2	Objektsequenzdiagramme . . . . .	99
9.3	Klassendiagramme . . . . .	102
9.3.1	Klassendiagramm der Client-App . . . . .	102
9.3.2	Klassendiagramm des Servers . . . . .	103
	<b>Glossar</b>	<b>104</b>
	<b>Akronyme</b>	<b>105</b>

## 1 Einleitung

Dieses Dokument ist das Ergebnis der Entwurfsphase zur Android App „1Sicht“ im PSE-Projekt „Write your own App“.

Im folgenden Kapitel wird die übergeordnete Architektur der Implementierung, bis hin zu angewandten Entwurfsmustern erläutert.

Die Dateiorganisation, die darauf aufbaut, wird im dritten Kapitel beschrieben und erklärt. Im weiteren Verlauf des Dokuments wird es spezifischer, hin zur Beschreibung von Methoden und Attributen. Zur Unterstützung bedienen wir uns hierbei bei Screenshots der Ordnerstrukturen, UML-Klassendiagrammen und UML-Sequenzdiagrammen.

Ein vollständiges Klassendiagramm des Projektes befindet sich am Ende des Dokuments. Es soll einen Überblick bieten und alle vorher erklärten Einzelheiten zum Entwurf im Gesamten nachvollziehbar machen.

## 2 Architektur

### 2.1 Architektur der App

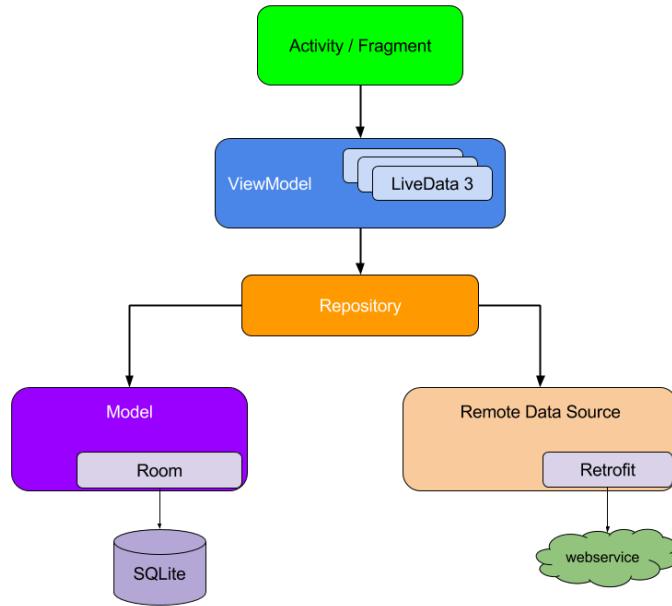


Abbildung 1: App Architektur Diagramm

Die Architektur der App orientiert sich an dem MVVM Architekturmuster, das zur Trennung von Darstellung und Logik der Benutzerschnittstelle (GUI) dient. Wir können also in der Abbildung die folgende 3 Schichten (von oben nach unten) abgrenzen:

- **View-Schicht** - Diese Schicht beinhaltet alle durch die grafische Benutzeroberfläche (GUI) angezeigten Elementen (Activities, Fragments, usw.). Es bindet sich an Eigenschaften des ViewModels, um Inhalte darzustellen, zu manipulieren und Benutzereingaben weiterzuleiten.
- **ViewModel-Schicht** - Das ViewModel dient als Bindeglied zwischen View und obigem Model. Einerseits tauscht es Information mit dem Model aus, ruft also Methoden oder Dienste auf, andererseits stellt es der View öffentliche Eigenschaften und Befehle zur Verfügung. Diese werden von der View an Steuerelemente gebunden, um Inhalte auszugeben bzw. UI-Ereignisse weiterzuleiten. Das ViewModel darf dabei keinerlei Kenntnis der View besitzen.

- **Repository-Schicht** - Die Datenzugriffsschicht für die Inhalte, die dem Benutzer angezeigt und von ihm manipuliert werden. Dazu benachrichtigt es über Datenänderungen und führt eine Validierung der vom Benutzer übergebenen Daten durch. Es enthält die gesamte Geschäftslogik und ist für sich alleine durch Unit Tests überprüfbar.

Dabei zu beachten ist, dass jede Komponente nur von der Komponente abhängt, die eine Ebene darunter liegt. Beispielsweise hängen Activities und Fragments nur vom ViewModell ab. Das Repository ist die einzige Klasse, die von mehreren anderen Klassen abhängt. In unserem Fall hängt das Repository von einem beständigen Datenmodell (lokale Datenbank) und einer Remote-Backend-Datenquelle (Server) ab.

Dieses Design schafft eine konsistente und angenehme Benutzererfahrung. Unabhängig davon, ob der Benutzer einige Minuten nach dem letzten Schließen oder einige Tage später zur App zurückkehrt, wird sofort angezeigt, dass die App lokal bestehen bleibt. Wenn diese Daten veraltet sind, aktualisiert das Repository-Modul die Daten im Hintergrund.

## 2.2 Architektur des Servers

Die Architektur des Servers orientiert sich an dem MVC Architekturstil, wobei wir zusätzlich das Repository Entwurfsmuster benutzt haben. Die Hauptkomponenten kann man folgendermaßen zusammenfassen:

- **View-Schicht** - Diese Schicht ist mehr oder weniger mit der Controller-Schicht zusammengefasst, da es sich bei unserem Server nur um JSON Format als Kommunikationsformat handelt und es werden eben keine HTML Seiten generiert, damit man eine dedizierte View-Schicht hat.
- **Controller-Schicht** - Diese Schicht zusammen mit dem Repository Entwurfsmuster sorgt dafür, die Kommunikation zwischen der API Schnittstelle und der Model-Schicht zu verwirklichen.
- **Model-Schicht** - Die Model-Schicht ist dafür verantwortlich, die Daten von der Datenbank als Kotlin Klassen darzustellen und die Kommunikation mit der Datenbank zu erleichtern.

Der Server dient als REST Schnittstelle und stellt folgende URLs als Endpunkte über HTTP zu Client Apps zur Verfügung:

**/students**

- **/create POST**
- **/ GET**
- **/studentId GET**
- **/studentId PUT**
- **/studentId DELETE**
- **/studentId/reviews GET**

**/reviews**

- **/create POST**
- **/ GET**
- **/employeeId GET**

- /employeeId PUT
- /employeeId DELETE
- /verify/employeeId PUT
- /employeeId/reviews GET
- /employeeId/reviews/reviewId GET

/admins

- /create POST
- / GET
- /adminId GET
- /adminId PUT
- /adminId DELETE

/reviews

- /create POST
- / GET
- /reviewId GET
- /?keywords GET
- /reviewId PUT
- /reviewId DELETE
- /reviewId/timeslots GET
- /reviewId/timeslots POST
- /reviewId/timeslots/timeslotId GET
- /reviewId/timeslots/timeslotId PUT

- /reviewId/timeslots/timeslotId **DELETE**
- /reviewId/signUp/timeslotId/studentId **POST**
- /reviewId/signOut/timeslotId/studentId **POST**
- /reviewId/changeTimeslot/newTimeslotId/studentId **PUT**

## 2.3 Architektur der Datenbank

Die lokale Datenbank wird mittels der Room Persistenzbibliothek verwaltet. Diese Bibliothek bietet eine Abstraktionsschicht über SQLite, um einen stabileren Datenbankzugriff zu ermöglichen und gleichzeitig die volle Leistungsfähigkeit von SQLite zu nutzen. Es gibt 3 Hauptkomponenten in der Room-Bibliothek:

- **Database** enthält den Datenbankinhaber und dient als Hauptzugriffspunkt für die zugrunde liegende Verbindung zu den dauerhaften relationalen Daten der App.
- **Entity** repräsentiert eine Tabelle in der Datenbank.
- **Data Access Object (DAO)** enthält die Methoden für den Zugriff auf die Datenbank.

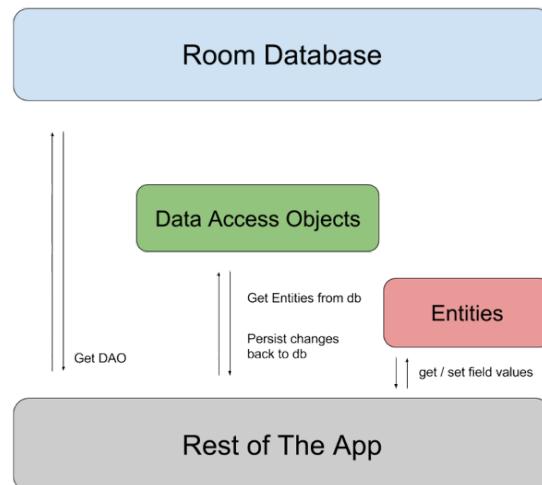
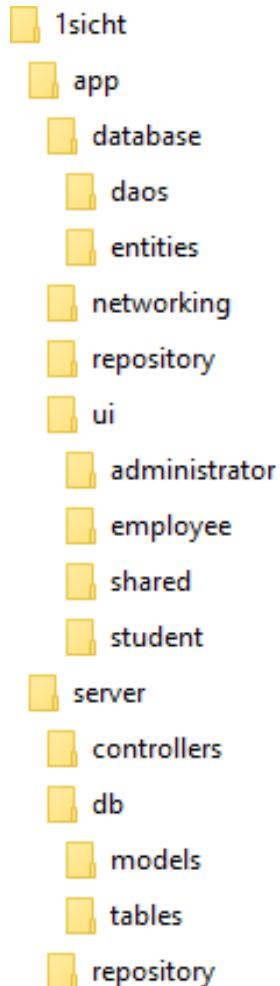


Abbildung 2: Room Architektur Diagramm

## 3 Dateiorganisation

### 3.1 Pakete

In der folgenden Grafik soll der Aufbau der Pakete anhand einer Ordnerstruktur klar gemacht werden. In den Ordner selbst befinden sich .kt-Dateien, die im Laufe dieses Entwurfsdokuments näher beschrieben werden.



### 3.2 JSON - Dateien

In diesem Abschnitt werden die JSON-Objekte, die für die Kommunikation zwischen der Client-App und dem Server benutzt werden, beispielhaft gezeigt.

Wichtig hierbei ist, dass in den JSON-Dateien nicht die Mengen an Objekten selbst, sondern nur anhand der IDs übergeben werden.

```

1 {
2   "adminId" : 12222,
3   "googleId" : "GoogleLoginToken",
4   "firstName" : "beispiel",
5   "lastName" : "administrator"
6 }
```

Beispiel 1: JSON-Datei für Administratoren

```

1 {
2   "employeeId" : 23333,
3   "googleId" : "GoogleLoginToken",
4   "firstName" : "beispiel",
5   "lastName" : "mitarbeiter",
6   "isVerified" : true,
7   "createdReviews": [
8     "reviewID" : 1234,
9     "reviewID" : 5678
10  ]
11 }
```

Beispiel 2: JSON-Datei für Mitarbeiter

```

1 {
2   "studentId" : 34444,
3   "googleId" : "GoogleLoginToken",
4   "firstName" : "beispiel",
5   "lastName" : "student",
6   "matriculationNumber" : 1234567,
7   "reviews": [
8     "reviewID" : 1234,
9     "reviewID" : 5678
10  ]
11 }
```

Beispiel 3: JSON-Datei für Studenten

```
1 {
2     "reviewID" : 1111,
3     "reviewName" : "LA I Einsicht",
4     "reviewRoom" : "Audimax",
5     "dateOfReview" :
6     {
7         "day" : 12,
8         "month" : 9,
9         "year" : 2018,
10        "hour" : 12,
11        "minute" : 30
12    },
13    "creator" : 23333,
14    "currentCountOfStudents" : 110
15 }
```

Beispiel 4: JSON-Datei für Klausureinsichten

```
1 {
2     "startOfTimeslot" :
3     {
4         "day" : 12,
5         "month" : 9,
6         "year" : 2018,
7         "hour" : 13,
8         "minute" : 30
9     },
10    "durationOfTimeslot" : 20,
11    "maxCountOfStudents" : 200,
12    "signedUpStudents" :
13    [
14        "matriculationNumber" : 1234567,
15        "matriculationNumber" : 8910111,
16    ],
17    "reviewID" : 1111
18 }
```

Beispiel 5: JSON-Datei für Timeslots

## 4 Benutzte Frameworks

In diesem Kapitel werden alle Frameworks angegeben und beschrieben, die für die Implementierung der App verwendet werden. Des weiteren wird beschrieben, warum wir uns für diese entschieden haben und welche Funktionalität von welchen Frameworks verwendet wird.

### 4.1 Ktor

Ktor ist ein Server-Framework, dass von JetBrains als Kotlin Framework bereitgestellt wird. Wir benutzen Ktor in unserem Projekt, da Ktor auf Kotlin aufbaut und es damit gut zu dem zu bedienenden Klienten, der Android-App auf den Endgeräten, passt.

### 4.2 Android Jetpack

Android Jetpack ist eine Sammlung aus verschiedenen Libraries. Für unsere App benutzen wir Room, um die Kommunikation mit SQLite leichter zu implementieren.

### 4.3 Retrofit

Wir benutzen Retrofit, um die Kommunikation vom Client zum Server zu ermöglichen und REST Aufrufe zum Server leicht und ohne viel Code durchführen zu können.

### 4.4 MySQL

MySQL ist eine relationale SQL Datenbank, die wir im Server Teil der App benutzen, da sie sehr populär und von fast allen Frameworks unterstützt ist. Außerdem ist MySQL open-source und sehr portabel zwischen vielen Architekturen und Technologischen Stacks.

### 4.5 SQLite

SQLite ist eine relationale SQL Datenbank. Wir haben uns für SQLite in Kombination mit Room für unsere lokale Datenbank entschieden, weil die Mehrheit der Android

Entwickler diese Kombination bevorzugen. Außerdem wird Room von Google angeboten und ist relativ neu. Die Kombination von SQLite und Room ist kleiner, mehr robust und mehr flexibel als die Alternativen. Auch das Multithreading ist besser unterstützt. SQLite allein war eine gute lokale Datenbank, aber mit Room ist sie die beste Option.

## 4.6 Exposed

Exposed ist ein Framework zum Abbilden von Java/Kotlin Klassen in relationalen Tabellen in einer Datenbank und zur Kommunikation mit der bestimmten Datenbank. Wir haben uns für Exposed entschieden, weil es gut entwickelt ist und weil es relativ simpel zum Einsatz in unserem Projekt zu sein scheint. Außerdem ist Exposed ein Projekt von JetBrains, die eben die Entwickler von Kotlin sind, und es ist aus diesem Grund als Framework empfohlen.

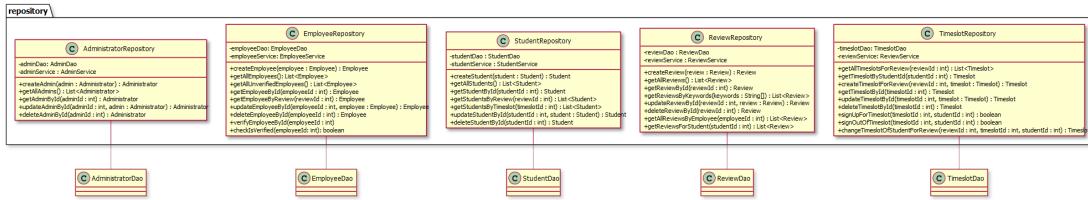
## 4.7 Firebase

Bei Firebase handelt es sich um ein Google-Framework, das Funktionalitäten zum Erstellen und Verbessern von Webanwendungen, aber auch, wie in unserem Fall, mobilen Anwendungen bereitstellt.

In unserer App wird die Funktionalität aus „Firebase Authentication“ zur Registrierung und Anmeldung mit einem Google-Account verwendet. Außerdem wird die Funktionalität aus „Cloud-Messaging“ verwendet, um Benachrichtigungen an die verschiedenen Benutzer der App zu senden.

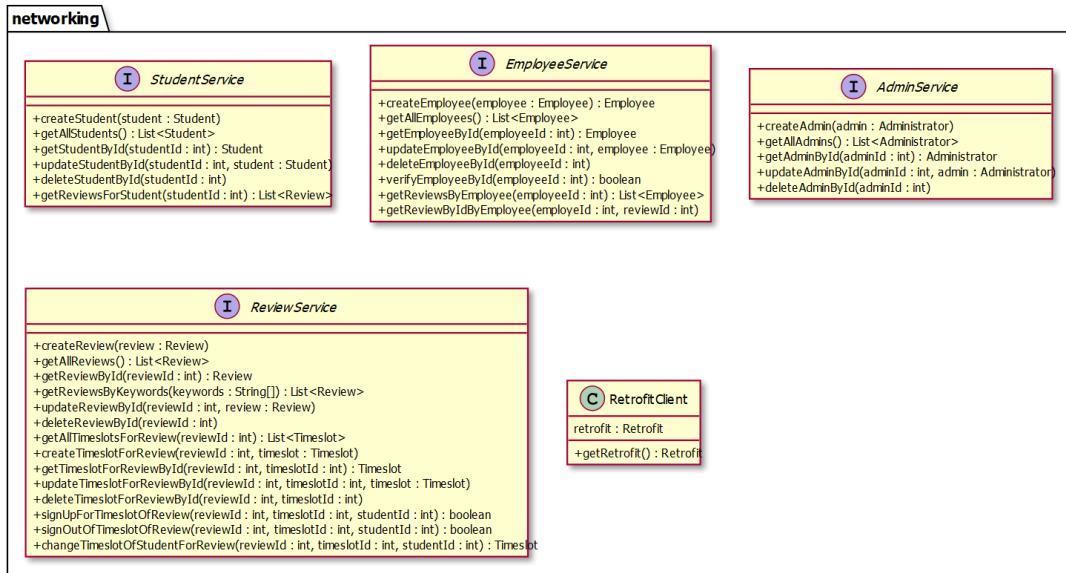
## 5 Paketübersicht - App

### 5.1 repository



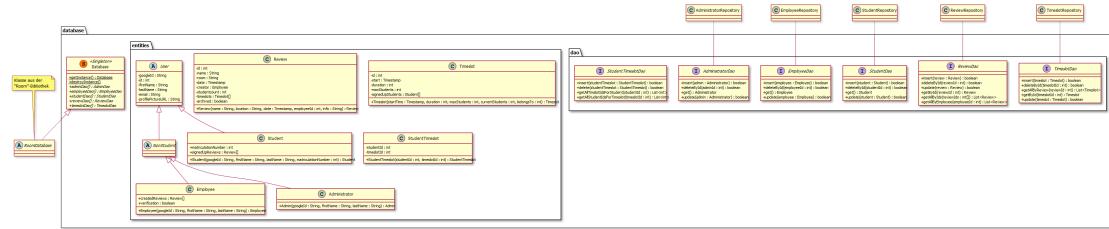
In diesem Paket befinden sich die Klassen StudentRepository, AdminRepository, EmployeeRepository, ReviewRepository und TimeslotRepository. Sie stellen eine Schnittstelle zur Kommunikation mit Datenquellen wie Server sowie mit der lokalen Datenbank zur Verfügung.

## 5.2 networking



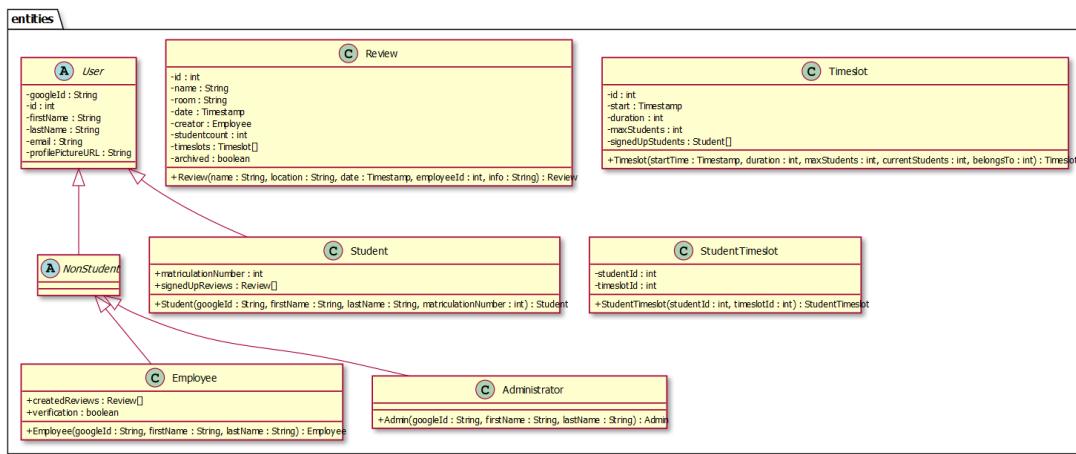
In diesem Paket befinden sich die Klassen RetrofitClient, die die Kommunikation zum Server zur Verfügung stellt, sowie die Interfaces StudentService, EmployeeService, AdminService und ReviewService, die die Server REST Endpunkte repräsentieren und als Methoden dem Client zur Verfügung stellen.

### 5.3 database



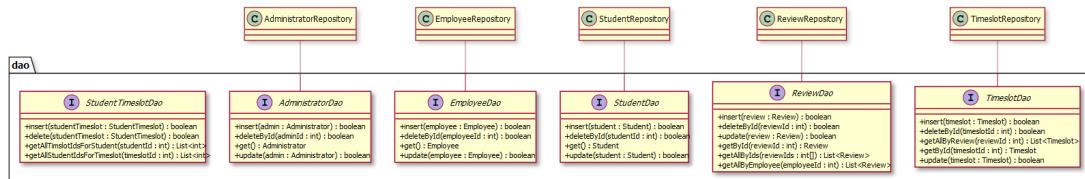
In diesem Paket befindet sich die Klasse Database, die von der Klasse RoomDatabase erbt, sowie die Unterpakete entities und daos. Die Klasse Database repräsentiert und verwaltet die lokale Datenbank der App, worin der aktuell angemeldete Nutzer, sowie die für ihn relevanten Daten gespeichert werden.

### 5.3.1 database.entities



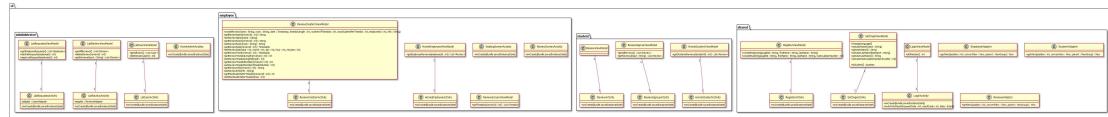
In diesem Paket befinden sich die Modelle der, von der App verwendeten, Daten. Die Klassen enthalten nur die Felder, die die entsprechende Attribute in den Tabellen repräsentieren.

### 5.3.2 database.daos



In diesem Paket befinden sich die Schnittstellen, die Methoden für Zugriff auf die Datenbank zur Verfügung stellen.

## 5.4 ui



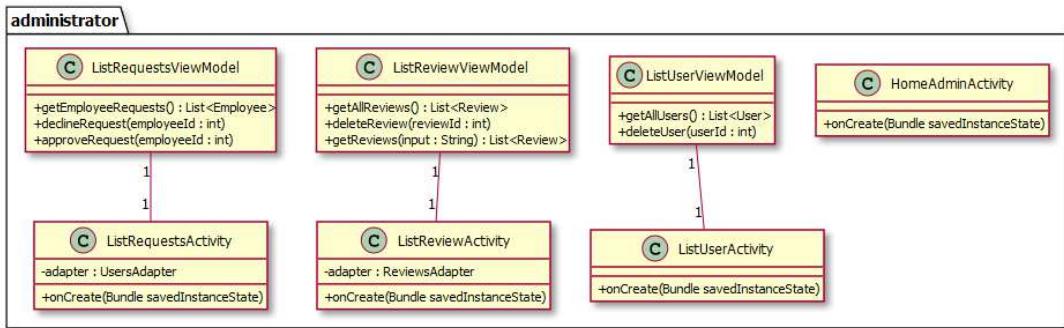
In diesem Paket befinden sich die Klassen der GUI.

Die Activity Klassen erstellen die Bildschirme, wo die App die Benutzeroberfläche darstellt. Sie erbt von der Klasse AppCompatActivity, die die grundlegende Klasse für Activity Klassen ist, die die AndroidX Bibliothek benutzen.

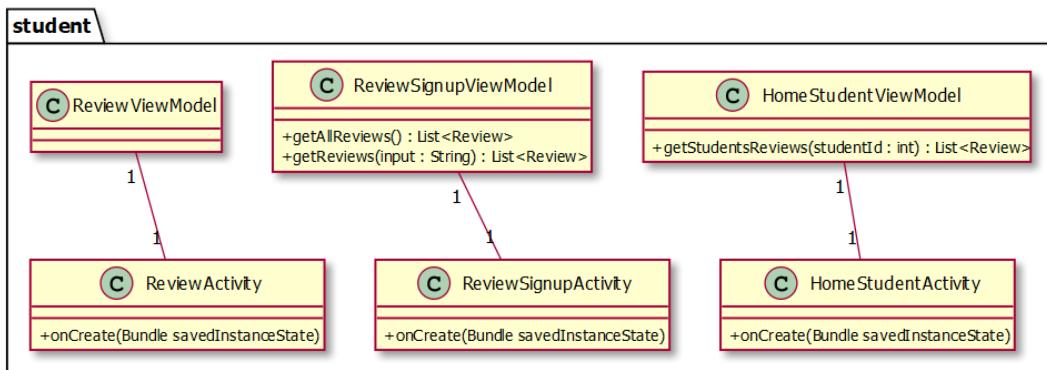
Die ViewModel Klassen dienen dazu, dass UI-bezogene Daten auf lebenszyklusbewusste Weise gespeichert und verwaltet werden. Mit der ViewModel Klasse können Daten Konfigurationsänderungen wie Bildschirmdrehungen überstehen. Sie behandelt auch die Kommunikation der Aktivität / Fragment mit dem Rest der Anwendung. Alle ViewModel Klassen erben von der ViewModel Klasse aus der AndroidX Bibliothek.

Weiterhin enthält das Paket utility Klassen, die die Logik der mehr komplizierten Komponenten der Views enthalten.

#### 5.4.1 ui.administrator

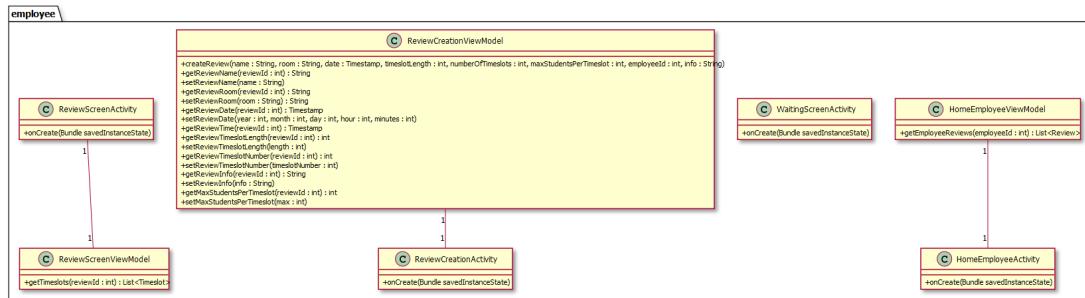


In dem administrator Paket befinden sich die Activity und ViewModel Klassen, die für das Erstellen der GUI des Administrators verantwortlich sind.

**5.4.2 ui.student**

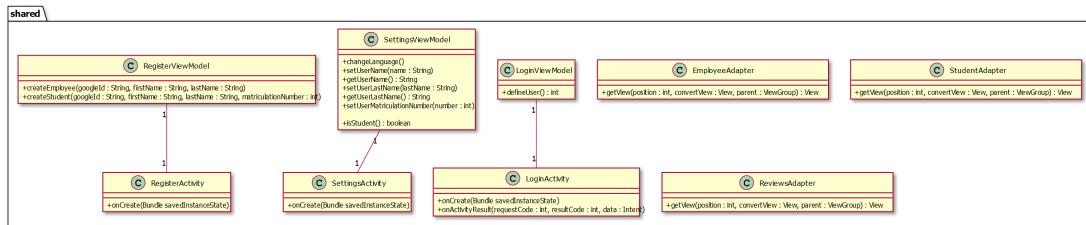
In dem student Paket befinden sich die Activity und ViewModel Klassen, die für das Erstellen der GUI des Studenten verantwortlich sind.

### 5.4.3 ui.employee



In dem `employee` Paket befinden sich die Activity und ViewModel Klassen, die für das Erstellen der GUI des Mitarbeiters verantwortlich sind.

#### 5.4.4 ui.shared



In dem shared Paket befinden sich die Activity und ViewModel Klassen, die für das Erstellen der GUI Bildschirme, die von allen drei Benutzerarten gesehen werden, und die utility Klassen.

## 6 Klassenübersicht - App

### 6.1 networking

#### 6.1.1 StudentService

Dieses Interface dient dazu, REST Aufrufe zum Server bezüglich den /student REST Endpunkten zu machen

##### Methoden

```
+ createStudent(student : Student) : Void
```

Diese Methode dient dazu, einen Aufruf zum Server zu machen, um einen neuen Studenten zu erstellen

```
+ getAllStudents() : List<Student>
```

Diese Methode bekommt und gibt eine Liste mit allen Studenten in der App zurück

```
+ getStudentById(studentId : int) : Student
```

Diese Methode liefert einen einzigen Student mit einer bestimmten ID in der App

```
+ updateStudentById(studentId : int, student : Student) : Void
```

Diese Methode modifiziert einen bereits existierenden Studenten in der App

```
+ deleteStudentById(studentId : int) : Void
```

Diese Methode löscht einen bestimmten Studenten von der App, falls er existiert.

```
+ getReviewsForStudent(studentId : int) : List<Review>
```

Diese Methode gibt eine Liste mit Klausureinsichten, für die sich ein Student angemeldet hat, zurück

### 6.1.2 EmployeeService

Dieses Interface dient dazu, REST Aufrufe zum Server bezüglich den /employee REST Endpunkten zu machen

#### Methoden

+ `createEmployee(employee : Employee) : Employee`

Diese Methode dient dazu, einen neuen Mitarbeiter in der App zu erstellen.

+ `getAllEmployees() : List<Employee>`

Diese Methode liefert eine Liste mit allen Mitarbeitern in der App

+ `getEmployeeById(employeeId : int) : Employee`

Diese Methode liefert einen bestimmten Mitarbeiter in der App

+ `updateEmployeeById(employeeId : int, employee: Employee) : Void`

Diese Methode modifiziert einen bestimmten Mitarbeiter in der App, falls er existiert

+ `deleteEmployeeById(employeeId : int) : Void`

Diese Methode löscht einen Mitarbeiter von der App, falls er existiert

+ `verifyEmployeeById(employeeId : int) : Boolean`

Diese Methode dient zur Verifizierung eines bestimmten Mitarbeiters und liefert true, falls die Verifizierung erfolgreich war, sonst false.

+ `getReviewsByEmployee(employeeId : int) : List<Review>`

Diese Methode liefert eine Liste mit allen von einem bestimmten Mitarbeiter erstellten Klausureinsichten

+ `getReviewByIdByEmployee(employeeId : int, reviewId : int) : Review`

Diese Methode liefert eine bestimmte Klausureinsicht, die von einem bestimmten Mitarbeiter erstellt wurde.

### 6.1.3 AdminService

Dieses Interface dient dazu, REST Aufrufe zum Server bezüglich den /admin REST Endpunkten zu machen

#### Methoden

```
+ createAdmin(admin : Administrator) : Void
```

Diese Methode erstellt einen neuen Admin in der App

```
+ getAllAdmins() : List<Administrator>
```

Diese Methode liefert eine Liste mit allen Admins in der App

```
+ getAdminById(adminId : int) : Administrator
```

Diese Methode liefert einen bestimmten Admin in der App

```
+ updateAdminById(adminId : int, admin : Administrator) : Void
```

Diese Methode modifiziert einen bestimmten Admin in der App, falls er existiert

```
+ deleteAdminById(adminId : int) : Void
```

Diese Methode löscht einen bestimmten Admin von der App, falls er existiert

### 6.1.4 ReviewService

Dieses Interface dient dazu, REST Aufrufe zum Server bezüglich den /review REST Endpunkten zu machen

#### Methoden

```
+ createReview(review : Review) : Void
```

Diese Methode erstellt eine neue Klausureinsicht in der App

```
+ getAllReviews() : List<Review>
```



Diese Methode liefert eine Liste mit allen Klausureinsichten in der App

+ `getReviewById(reviewId : int) : Review`

Diese Methode liefert eine bestimmte Klausureinsicht in der App

+ `getReviewsByKeywords(keywords : String[]) : List<Review>`

Diese Methode liefert eine Liste mit Klausureinsichten, die bestimmte Schlüsselwörter in ihrer Beschreibung enthalten

+ `updateReviewById(reviewId : int, review : Review) : Void`

Diese Methode modifiziert eine Klausureinsicht in der App, falls sie existiert

+ `deleteReviewById(reviewId: int): Void`

Diese Methode löscht eine Klausureinsicht von der App, falls sie existiert

+ `getAllTimeslotsForReview(reviewId : int) : List<Timeslot>`

Diese Methode liefert eine Liste mit Zeitslots, die zu einer bestimmten Klausureinsicht gehören

+ `createTimeslotForReview(reviewId : int, timeslot : Timeslot) : Void`

Diese Methode erstellt einen neuen Zeitslot für eine bestimmte Klausureinsicht

+ `getTimeslotForReviewById(reviewId : int, timeslotId : int) : Timeslot`

Diese Methode liefert einen bestimmten Zeitslot von einer bestimmten Klausureinsicht

+ `updateTimeslotForReviewById(reviewId : int, timeslotId : int, timeslot : Timeslot) : Void`

Diese Methode modifiziert einen bestimmten Zeitslot von einer bestimmten Klausureinsicht, falls er existiert

+ `deleteTimeslotForReviewById(reviewId : int, timeslotId : int) : Void`



Diese Methode löscht einen bestimmten Zeitslot von einer bestimmten Klassenübersicht, falls er existiert

+ `signUpForTimeslotOfReview(reviewId : int, timeslotId : int, studentId : int) : Boolean`

Diese Methode meldet einen bestimmten Studenten für einen bestimmten Zeitslot einer bestimmten Klausureinsicht an und gibt true zurück, falls die Anmeldung erfolgreich war, sonst false

+ `signOutOfTimeslotOfReview(reviewId : int, timeslotId : int, studentId : int) : Boolean`

Diese Methode meldet einen bestimmten Studenten von einem bestimmten Zeitslot einer bestimmten Klausureinsicht ab und gibt true zurück, falls die Abmeldung erfolgreich war, sonst false

+ `changeTimeslotOfStudentForReview(reviewId : int, timeslotId : int, studentId : int) : Timeslot`

Diese Methode meldet einen Studenten von einem bestimmten Zeitslot einer bestimmten Klausureinsicht ab und meldet ihn für einen neuen Zeitslot der gleichen Klausureinsicht an und liefert den neuen Zeitslot als Ergebnis

### 6.1.5 RetrofitClient

In dieser Klasse werden die Service Interfaces initialisiert und die Kommunikation zum Server wird zur Verfügung gestellt

#### Attribute

- `retrofit : Retrofit`

Singleton Instanz der Retrofit Klasse, die dazu dient, die Verbindung mit dem REST Server über die Endpoints zu verwirklichen

#### Methode

+ `getRetrofit() : Retrofit`

Diese Methode initialisiert die Verbindung zum Server und gibt die Singleton Instanz zurück, damit sie von anderen Klassen benutzt werden kann



## 6.2 database

### 6.2.1 Database

Database extends RoomDatabase

Diese abstrakte Klasse definiert die Liste der Entities und Datenzugriffsobjekte in der Datenbank. Es ist auch der Hauptzugriffspunkt für die zugrunde liegende Verbindung. In dieser Klasse wurde das Singleton Entwurfsmuster verwendet, womit sichergestellt wird, dass es zu keiner Duplizierung von Daten kommt.

#### Methoden

+ static getInstance() : Database

Gibt die einzige mögliche Instanz von Database zurück. Deswegen ist die Methode auch als static deklariert.

+ static destroyInstance() : void

Löscht die aktuelle Datenbank.

+ abstract adminDao() : AdminDao

Gibt die AdminDao-Instanz zurück.

+ abstract studentDao() : StudentDao

Gibt die StudentDao-Instanz zurück.

+ abstract employeeDao() : EmployeeDao

Gibt die EmployeeDao-Instanz zurück.

+ abstract reviewDao() : ReviewDao

Gibt die ReviewDao-Instanz zurück.

+ abstract timeslotDao() : TimeslotDao

Gibt die TimeslotDao-Instanz zurück.



## 6.3 database.entities

### 6.3.1 Administrator

Die **Administrator** Klasse ist das Modell eines Administrators in der Datenbank.

#### Attribute

- `googleId : String`

Repräsentiert die einzigartige Identifikationsnummer von Google des Administrators.

- `id : int`

Repräsentiert die einzigartige interne Identifikationsnummer des Administrators.

- `firstName : String`

Repräsentiert den Vornamen des Administrators.

- `lastName : String`

Repräsentiert den Nachnamen des Administrators.

#### Methoden

+ `Admin(googleId : String, firstName : String, lastName : String) : Admin`

Initialisiert ein neues Administrator Objekt mit den gegebenen Eingabedaten.

### 6.3.2 Student

Die **Student** Klasse ist das Modell eines Studenten in der Datenbank.

#### Attribute

- `googleId : String`



Repräsentiert die einzigartige Identifikationsnummer von Google des Studenten.

- `id : int`

Repräsentiert die einzigartige interne Identifikationsnummer des Studenten.

- `firstName : String`

Repräsentiert den Vornamen des Studenten.

- `lastName : String`

Repräsentiert den Nachnamen des Studenten.

- `matriculationNumber : int`

Repräsentiert die einzigartige Matrikelnummer des Studenten.

## Methoden

+ `Student(googleId : String, firstName : String, lastName : String, matriculationNumber : int) : Student`

Initialisiert einen neuen Studenten mit den gegebenen Eingabedaten.

### 6.3.3 Employee

Die `Employee` Klasse ist das Modell eines Mitarbeiters in der Datenbank.

#### Attribute

- `googleId : String`

Repräsentiert die einzigartige Identifikationsnummer von Google des Mitarbeiters.

- `id : int`

Repräsentiert die einzigartige interne Identifikationsnummer des Mitarbeiters.

- `firstName : String`

Repräsentiert den Vornamen des Mitarbeiters.

- `lastName : String`

Repräsentiert den Nachnamen des Mitarbeiters.

## Methoden

+ `Employee(googleId : String, firstName : String, lastName : String)`  
: `Employee`

Initialisiert einen neuen Mitarbeiter mit den gegebenen Eingabedaten.

### 6.3.4 Review

Die `Review` Klasse ist das Modell einer Klausureinsicht in der Datenbank.

#### Attribute

- `id : int`

Repräsentiert die einzigartige interne Identifikationsnummer der Klausureinsicht.

- `name : String`

Repräsentiert den Namen der Klausureinsicht.

- `room : String`

Repräsentiert den Ort, wo die Klausureinsicht stattfindet.

- `date : Timestamp`

Der Datum und Anfangsuhrzeit der Klausureinsicht.

- `timeslotLength : int`

Repräsentiert die Dauer eines Zeitslots während der Klausureinsicht.



- `numberOfTimeslots` : int

Repräsentiert die Anzahl von Zeitslots in der Klausureinsicht.

- `employeeId` : String

Repräsentiert die Id des Mitarbeiters, der die Klausureinsicht erstellt hat.

- `info` : String

Repräsentiert die zusätzliche Information über die Klausureinsicht.

## Methoden

+ `Review(name : String, room : String, date : Timestamp, timeslotLength : int, numberOfTimeslots : int, employeeId : String, info : String)` : Review

Initialisiert eine neue Klausureinsicht mit den gegebenen Eingabedaten.

### 6.3.5 Timeslot

Die `Timeslot` Klasse ist das Modell eines Zeitslots in der Datenbank.

#### Attribute

- `id` : int

Repräsentiert die einzigartige interne Identifikationsnummer des Zeitslots.

- `starttime` : Timestamp

Repräsentiert die Anfangszeit des Zeitslots.

- `duration` : int

Repräsentiert die Dauer des Zeitslots.

- `maxStudents` : int



Repräsentiert die maximale Anzahl von Studenten, die sich für den Zeitslot anmelden dürfen.

- `currentStudents : int`

Repräsentiert die aktuelle Anzahl von Studenten, die sich für den Zeitslot schon angemeldet haben.

- `belongsTo : int`

Repräsentiert die Id-Nummer der Klausureinsicht, zu der der Zeitslot gehört.

## Methoden

+ `Timeslot(startTime : Timestamp, duration : int, maxStudents : int, currentStudents : int, belongsTo : int)`

Initialisiert einen neuen Zeitslot mit den gegebenen Eingabedaten.

### 6.3.6 `StudentTimeslot`

#### Attribute

- `studentId : String`

Repräsentiert die Id des Studenten, der sich für diesen Zeitslot angemeldet hat.

- `timeslotId : int`

Repräsentiert die Id-Nummer des Zeitslots, für den sich der Student angemeldet hat.

#### Methoden

+ `StudentTimeslot(studentId : int, timeslotId : int)`

Initialisiert eine neue Student-Zeitslot Objekt mit den gegebenen Eingabedaten.



## 6.4 repository

### 6.4.1 AdminRepository

Diese Klasse stellt Lese- und Schreibzugriffe auf Admin Objekte sowohl in der lokalen Datenbank, als auch im Server zur Verfügung.

#### Attribute

- adminDao: AdminDao

Das DAO Objekt, das zur Kommunikation mit der lokalen Datebank dient

- adminService: AdminService

Das Interface, das zur Kommunikation mit dem Server dient

#### Methoden

+ createAdmin(admin : Administrator) : Administrator

Diese Methode ermöglicht es, einen neuen Administrator in der App zu erstellen.

+ getAllAdmins() : List<Administrator>

Diese Methode ermöglicht es, alle Admins in der App in einer Liste zu bekommen.

+ getAdminById(adminId : int) : Administrator

Diese Methode ermöglicht es, einen Admin mit einer bestimmten Id in der App zu bekommen.

+ updateAdminById(adminId : int, admin : Administrator) : Administrator

Diese Methode ermöglicht es, einen Admin mit einer Id durch einen modifizierten Admin zu ersetzen.

+ deleteAdminById(adminId : int) : Administrator

Diese Methode ermöglicht es, einen Admin mit einer bestimmten Id in der App, vollständig aus der App zu löschen.

#### 6.4.2 EmployeeRepository

Diese Klasse stellt Lese- und Schreibzugriffe auf Mitarbeiter Objekte sowohl in der lokalen Datenbank, als auch im Server zur Verfügung.

##### Attribute

- employeeDao: EmployeeDao

Das DAO Objekt, das zur Kommunikation mit der lokalen Datebank dient

- employeeService: EmployeeService

Das Interface, das zur Kommunikation mit dem Server dient

##### Methoden

+ createEmployee(employee : Employee) : Employee

Diese Methode ermöglicht es, einen neuen Mitarbeiter in der App zu erstellen.

+ getAllEmployees() : List<Employee>

Diese Methode ermöglicht es, alle Mitarbeiter in der App in einer Liste zu bekommen.

+ getAllUnverifiedEmployees() : List<Employee>

Diese Methode ermöglicht es, alle noch nicht verifizierten Mitarbeiter in der App in einer Liste zu bekommen.

+ getEmployeeById(employeeId : int) : Employee

Diese Methode ermöglicht es, einen Mitarbeiter mit einer bestimmten Id in der App, als Ergebnis zu bekommen.

+ getEmployeeByReview(reviewId : int) : Employee



Diese Methode ermöglicht es, einen Mitarbeiter, der eine Klausureinsicht mit einer Id in der App erstellt hat, als Ergebnis zu bekommen.

+ `updateEmployeeById(employeeId : int, employee : Employee) : Employee`

Diese Methode ermöglicht es, einen Mitarbeiter mit einer Id durch einen modifizierten Mitarbeiter zu ersetzen.

+ `deleteEmployeeById(employeeId : int) : Employee`

Diese Methode ermöglicht es, einen Mitarbeiter mit einer bestimmten Id in der App, vollständig aus der App zu löschen.

+ `verifyEmployeeById(employeeId : int) : Void`

Diese Methode ermöglicht es, einen, noch nicht verifizierten, Mitarbeiter mit einer bestimmten Id in der App, zu verifizieren.

+ `checkIsVerified(employeeId: int): Boolean`

Diese Methode ermöglicht es, zu überprüfen, ob ein Mitarbeiter mit einer bestimmten Id in der App bereits verifiziert ist.

#### 6.4.3 **StudentRepository**

Diese Klasse stellt Lese- und Schreibzugriffe auf Student Objekte sowohl in der lokalen Datenbank, als auch im Server zur Verfügung.

##### **Attribute**

- `studentDao: StudentDao`

Das DAO Objekt, das die Kommunikation mit der lokalen Datebank zur Verfügung stellt

- `studentService: StudentService`

Das Interface, das die Kommunikation mit dem Server zur Verfügung stellt

##### **Methoden**

```
+ createStudent(student : Student) : Student
```

Diese Methode ermöglicht es, einen neuen Studenten in der App zu erstellen.

```
+ getAllStudents() : List<Student>
```

Diese Methode ermöglicht es, alle Studenten in der App in einer Liste zu bekommen.

```
+ getStudentById(studentId : int) : Student
```

Diese Methode ermöglicht es, einen Student in der App mithilfe seiner Id als Ergebnis zu bekommen.

```
+ getStudentsByReview(reviewId : int) : List<Student>
```

Diese Methode ermöglicht es, Studenten, die sich für eine bestimmte Klausureinsicht angemeldet haben, in einer Liste zu bekommen.

```
+ getStudentsByTimeslot(timeslotId : int) : List<Student>
```

Diese Methode ermöglicht es, Studenten, die sich für einen bestimmten Zeitslot angemeldet haben, in einer Liste zu bekommen.

```
+ updateStudentById(studentId : int, student : Student) : Student
```

Diese Methode ermöglicht es, einen Studenten mit einer Id durch einen modifizierten Studenten zu ersetzen

```
+ deleteStudentById(studentId : int) : Student
```

Diese Methode ermöglicht es, einen Studenten mit einer bestimmten Id, vollständig aus der App zu löschen.

#### 6.4.4 ReviewRepository

Diese Klasse stellt Lese- und Schreibzugriffe auf Klausureinsicht Objekte sowohl in der lokalen Datenbank, als auch im Server zur Verfügung.

##### Attribute

- `reviewDao: ReviewDao`

Das DAO Objekt, das zur Kommunikation mit der lokalen Datebank dient

- `reviewService: ReviewService`

Das Interface, das zur Kommunikation mit dem Server dient

## Methoden

+ `createReview(review : Review) : Review`

Diese Methode ermöglicht es, eine neue Klausureinsicht in der App zu erstellen.

+ `getAllReviews() : List<Review>`

Diese Methode ermöglicht es, alle Klausureinsichten in der App in einer Liste zu bekommen.

+ `getReviewById(reviewId : int) : Review`

Diese Methode ermöglicht es, eine Klausureinsicht mit einer bestimmten Id in der App als Ergebnis zu bekommen.

+ `getReviewsByKeywords(keywords : String[]) : List<Review>`

Diese Methode ermöglicht es, Klausureinsichten in der App, die bestimmte Schlüsselwörter als Information enthalten, in einer Liste zu bekommen.

+ `updateReviewById(reviewId : int, review : Review) : Review`

Diese Methode ermöglicht es, eine Klausureinsicht mit einer bestimmten Id durch eine modifizierte Klausureinsicht zu ersetzen.

+ `deleteReviewById(reviewId : int) : Review`

Diese Methode ermöglicht es, eine Klausureinsicht mit einer bestimmten Id in der App vollständig aus der App zu löschen.

+ `getAllReviewsByEmployee(employeeId : int) : List<Review>`



Diese Methode ermöglicht es, alle Klausureinsichten, die von einem Mitarbeiter mit einer bestimmten Id in der App erstellt wurden, in einer Liste zu bekommen.

```
+ getReviewsForStudent(studentId : int) : List<Review>
```

Diese Methode ermöglicht es, alle Klausureinsichten, für die sich ein Student mit einer bestimmten Id in der App angemeldet hat, in einer Liste zu bekommen.

#### 6.4.5 TimeslotRepository

Diese Klasse stellt Lese- und Schreibzugriffe auf Admin Objekte sowohl in der lokalen Datenbank, als auch im Server zur Verfügung.

##### Attribute

```
- timeslotDao: TimeslotDao
```

Das DAO Objekt, das zur Kommunikation mit der lokalen Datebank dient

```
- reviewService: ReviewService
```

Das Interface, das zur Kommunikation mit dem Server dient

##### Methoden

```
+ getAllTimeslotsForReview(reviewId : int) : List<Timeslot>
```

Diese Methode ermöglicht es, alle Zeitslots von einer Klausureinsicht mit einer bestimmten Id in der App, in einer Liste zu bekommen.

```
+ getTimeslotByStudentId(studentId : int) : Timeslot
```

Diese Methode ermöglicht es, alle Zeitslots, für die sich ein Student mit einer bestimmten Id in der App angemeldet hat, in einer Liste zu bekommen.

```
+ createTimeslotForReview(reviewId : int, timeslot : Timeslot) : Timeslot
```



Diese Methode erstellt einen neuen Zeitslot einer bestimmten Klausureinsicht

+ `getTimeslotById(timeslotId : int) : Timeslot`

Diese Methode liefert einen bestimmten Zeitslot in der App

+ `updateTimeslotById(timeslotId : int, timeslot : Timeslot) : Timeslot`

Diese Methode modifiziert einen bestimmten Zeitslot in der App

+ `deleteTimeslotById(timeslotId : int) : Timeslot`

Diese Methode Löscht einen bestimmten Zeitslot von der App

+ `signUpForTimeslot(timeslotId : int, studentId : int) : Boolean`

Diese Methode meldet einen bestimmten Studenten für einen bestimmten Zeitslot einer bestimmten Klausureinsicht an und liefert true bei erfolgreicher Anmeldung, sonst false

+ `signOutOfTimeslot(timeslotId: int, studentId: int) : Boolean`

Diese Methode meldet einen bestimmten Studenten von einem bestimmten Zeitslot einer bestimmten Klausureinsicht ab und liefert true bei erfolgreicher Abmeldung, sonst false

+ `changeTimeslotOfStudentForReview(reviewId : int, timeslotId : int, studentId : int) : Timeslot`

Diese Methode meldet einen bestimmten Studenten von einem bestimmten Zeitslot einer bestimmten Klausureinsicht ab und meldet ihn für einen anderen bestimmten Zeitslot der gleichen Klausureinsicht an und liefert den neuen Zeitslot

## 6.5 database.daos

### 6.5.1 AdministratorDao

Die `AdministratorDao` Schnittstelle stellt zur Verfügung die Methoden, die die Daten in der `Administrator` Tabelle in der Datenbank verwalten.

## Methoden

+ `insert(administrator : Administrator) : boolean`

Fügt den gegebenen Administrator in die Datenbank hinzu.

+ `deleteById(administratorId : int) : boolean`

Löscht den Administrator mit der gegebenen Id-Nummer aus der Datenbank.

+ `get() : Administrator`

Gibt den aktuell angemeldeten Administrator zurück.

+ `update(administrator : Administrator) : boolean`

Aktualisiert die Information für den aktuell angemeldeten Administrator.

### 6.5.2 ⓘ StudentDao

Die `StudentDao` Schnittstelle stellt die Methoden zur Verfügung, die die Daten in der `Student` Tabelle der Datenbank verwalten.

## Methoden

+ `insert(student : Student) : boolean`

Fügt den gegebenen Studenten in die Datenbank hinzu.

+ `deleteById(studentId : int) : boolean`

Löscht den Studenten mit der gegebenen Id-Nummer aus der Datenbank.

+ `get() : Student`

Gibt den aktuell angemeldeten Student zurück.

+ `update(student : Student) : boolean`

Aktualisiert die Information für den aktuell angemeldeten Student.

### 6.5.3 EmployeeDao

Die `EmployeeDao` Schnittstelle stellt die Methoden zur Verfügung, die die Daten in der Employee Tabelle der Datenbank verwalten.

#### Methoden

+ `insert(employee : Employee) : boolean`

Fügt den gegebenen Mitarbeiter in die Datenbank hinzu.

+ `deleteById(employeeId : int) : boolean`

Löscht den Mitarbeiter mit der gegebenen Id-Nummer aus der Datenbank.

+ `get() : Employee`

Gibt den aktuell angemeldeten Mitarbeiter zurück.

+ `update(employee : Employee) : boolean`

Aktualisiert die Information für den aktuell angemeldeten Mitarbeiter.

### 6.5.4 ReviewDao

Die `ReviewDao` Schnittstelle stellt die Methoden zur Verfügung, die die Daten in der Review Tabelle der Datenbank verwalten.

#### Methoden

+ `insert(review : Review) : boolean`

Fügt die gegebene Klausureinsicht in die Datenbank hinzu.

+ `deleteById(reviewId : int) : boolean`

Löscht die Klausureinsicht mit der gegebenen Id-Nummer aus der Datenbank.

+ `updateById(reviewId : int, review : Review) : boolean`

Aktualisiert die Information für die Klausureinsicht mit der gegebenen Id-Nummer.

+ `getById(reviewId : int) : Review`

Gibt die Klausureinsicht mit der gegebenen Id-Nummer zurück.

+ `getAllByIds(reviewIds : int[]) : List<Review>`

Gibt die Menge an Klausureinsichten mit den gegebenen Id-Nummern zurück.

+ `getAllByEmployee(employeeId : int) : List<Review>`

Gibt die Menge an Klausureinsichten, die von einem gegebenen Mitarbeiter erstellt wurden zurück.

#### 6.5.5 TimeslotDao

Die `TimeslotDao` Schnittstelle stellt die Methoden zur Verfügung, die die Daten in der Timeslot Tabelle der Datenbank verwalten.

##### Methoden

+ `insert(timeslot : Timeslot) : boolean`

Fügt den gegebenen Zeitslot in die Datenbank hinzu.

+ `deleteById(timeslotId: int): boolean`

Löscht den Zeitslot mit der gegebenen Id-Nummer.

+ `getAllByReview(reviewId : int) : List<Timeslot>`

Gibt die Menge an aller Zeitslots zurück, die zu einer Klausureinsicht gehören.

+ `getById(timeslotId : int) : Timeslot`

Gibt den Zeitslot mit der gegebenen Id-Nummer zurück.

+ `insertAll(timeslots : List<Timeslot>) : boolean`

Fügt die gegebene Menge an Zeitslots in die Datenbank hinzu.

+ `updateTimeslot(timeslotId: int, timeslot: Timeslot): void`

Ersetzt den Zeitslot mit der gegebenen Id-Nummer mit der neuen Zeitslot Objekt.

#### 6.5.6 ⓘ StudentTimeslotDao

Die `StudentTimeslotDao` Schnittstelle stellt die Methoden zur Verfügung, die die Daten in der `StudentTimeslot` Tabelle der Datenbank verwalten.

##### Methoden

+ `insert(studentTimeslot : StudentTimeslot) : boolean`

Fügt den neuen Eintrag in die Datenbank hinzu.

+ `delete(studentTimeslot : StudentTimeslot) : boolean`

Löscht den gegebenen Eintrag von der Datenbank.

+ `getAllTimeslotIdsForStudent(studentId : String) : List<int>`

Gibt die Menge an allen Zeitslots zurück, für die sich der gegebene Student angemeldet hat.

+ `getAllStudentIdsForTimeslot(timeslotId : int) : List<String>`

Gibt die Menge an allen Studenten zurück, die sich für den gegebenen Zeitslot angemeldet haben.

## 6.6 ui.shared

### 6.6.1 LogInActivity



LogIn-Bildschirm

LogInActivity extends AppCompatActivity

#### Methoden

+ onCreate(Bundle savedInstanceState): Void

Die Methode zum Erstellen der Activity ohne Datenverlust

+ onActivityResult(requestCode : int, resultCode : int, data : Intent): Void

Empfängt ein GoogleSignInAccount Objekt, das für das Erstellen oder Überprüfen eines Accounts gebraucht wird, und schickt es zum Repository.

### 6.6.2 LogInViewModel

LogInViewModel extends ViewModel

#### Methoden

```
+ defineUser() : int
```

Die Methode überprüft, ob der Benutzer Student, Mitarbeiter oder Admin ist, und gibt 1 für Student, 2 für Mitarbeiter und 3 für Admin zurück, damit den entsprechenden Home Bildschirm erstellt wird.

### 6.6.3 RegisterActivity



RegisterActivity extends AppCompatActivity

#### Methoden

```
+ onCreate(Bundle savedInstanceState) : Void
```

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.6.4 RegisterViewModel

RegisterViewModel extends ViewModel

#### Methoden

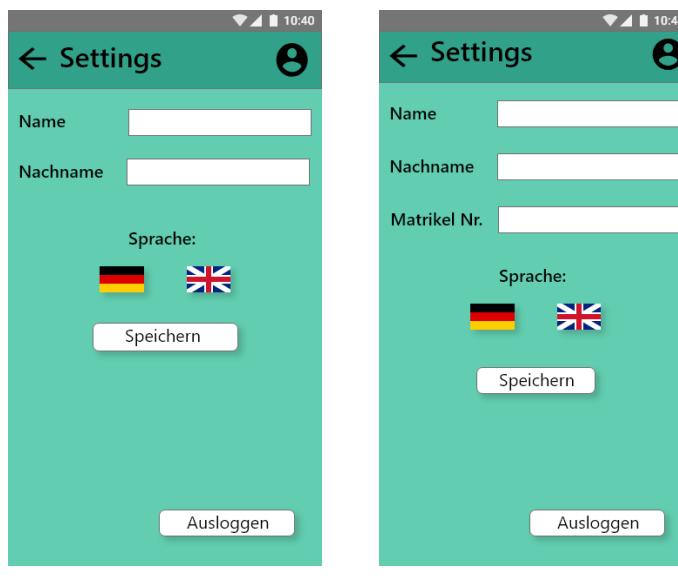
```
+ createEmployee(googleID : String, firstName : String,
lastName : String) : Void
```

Die Methode zum Initialisieren eines neuen, nicht verifizierten, Mitarbeiters. Diese Methode wird aufgerufen, wenn der Nutzer auf den Bestätigen Button klickt, nachdem er den Mitarbeiter Button gewählt hat.

```
+ createStudent(googleID : String, firstName : String,
lastName : String, matriculationNumber : int) : Void
```

Die Methode zum Initialisieren eines neuen Studenten. Diese Methode wird aufgerufen, wenn der Nutzer auf den Bestätigen Button klickt, nachdem er den Student Button gewählt hat.

### 6.6.5 SettingsActivity



Für Mitarbeiter und Admins

Für Studenten

Settings-Bildschirm



```
SettingsActivity extends AppCompatActivity
```

#### Methoden

```
+ onCreate(Bundle savedInstanceState) : Void
```

Die Methode zum Erstellen der Activity ohne Datenverlust.

#### 6.6.6 SettingsViewModel

```
SettingsViewModel extends ViewModel
```

##### Methoden

```
+ changeLanguage() : Void
```

Ändert die Sprache der App.

```
+ setUserName(name : String) : Void
```

Setzt den Vornamen des Benutzers.

```
+ getUserName() : String
```

Empfängt den Vornamen des Benutzers von Repository, damit der Vorname auf dem Settings Bildschirm angezeigt werden kann.

```
+ setUserLastName(lastName : String) : Void
```

Setzt den Nachnamen des Benutzers.

```
+ getUserLastName() : String
```

Empfängt den Nachnamen des Benutzers von Repository, damit der Nachnamen auf dem Settings Bildschirm angezeigt werden kann.

```
+ setUserMatriculationNumber(number : int) : Void
```

Setzt die Matrikelnummer des Studenten.

+ `getUserMatriculationNumber() : String`

Empfängt die Matrikelnummer des Studenten vom Repository, damit die Matrikelnummer auf dem Settings Bildschirm angezeigt werden kann.

+ `isStudent() : Boolean`

Überprüft, ob der Benutzer Student ist, damit den entsprechenden Settings Bildschirm angezeigt wird.

### 6.6.7 EmployeeAdapter

`ItemsAdapter extends ArrayAdapter<Employee>`

Die `EmployeeAdapter` Klasse füllt eine ListView mit den Daten aus dem Mitarbeiter Array. Damit wird eine vertikale Liste von scrollbaren Elementen erstellt.

#### Methode

+ `getView(position : int, convertView : View, parent : ViewGroup) : View`

Gibt das View Objekt zurück, das die Daten aus der Liste an einer bestimmten Position darstellt.

### 6.6.8 StudentAdapter

`ItemsAdapter extends ArrayAdapter<Student>`

Die `StudentAdapter` Klasse füllt eine ListView mit den Daten aus dem Studenten Array. Damit wird eine vertikale Liste von scrollbaren Elementen erstellt.

#### Methode

+ `getView(position : int, convertView : View, parent : ViewGroup) : View`

Gibt das View Objekt zurück, das die Daten aus der Liste an einer bestimmten Position darstellt.



### 6.6.9 ReviewsAdapter

ItemsAdapter extends ArrayAdapter<Review>

Die **ReviewsAdapter** Klasse füllt eine ListView mit den Daten aus dem Review Array. Damit wird eine vertikale Liste von scrollbaren Elementen erstellt.

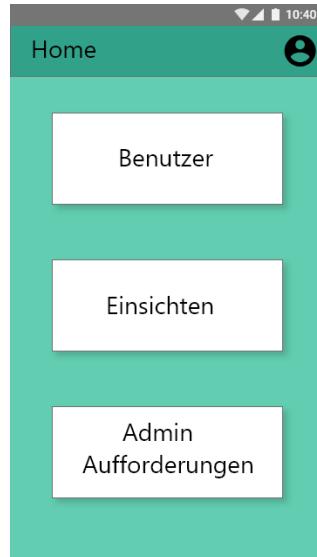
#### Methode

+ getView(position : int, convertView : View, parent : ViewGroup) : View

Gibt das View Objekt zurück, das die Daten aus der Liste an einer bestimmten Position darstellt.

## 6.7 ui.administrator

### 6.7.1 HomeActivity



HomeAdmin-Bildschirm

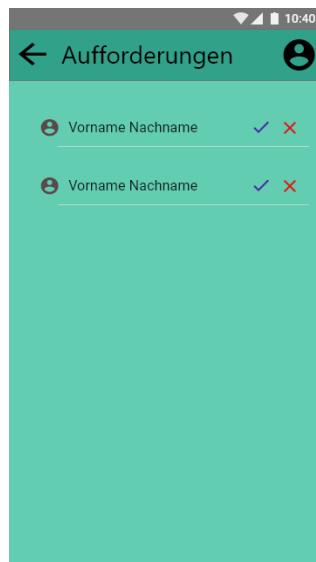
HomeActivity extends AppCompatActivity

#### Methoden

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.7.2 ListRequestsActivity



Requests-Bildschirm

ListRequestActivity extends AppCompatActivity

#### Attribute

- adapter : EmployeeAdapter

Der Adapter für die scrollbare Liste von Mitarbeiteranfragen.

#### Methoden

- + onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.7.3 ListRequestsViewModel

```
ListRequestsViewModel extends ViewModel
```

#### Methoden

```
+ getEmployeeRequests() : List<Employee>
```

Empfängt eine Liste mit Employee Objekten, die auf Verifizierung warten.

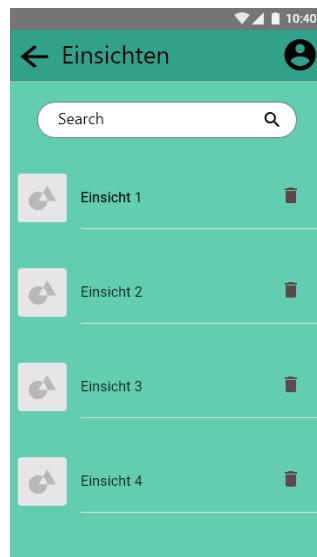
```
+ declineRequest(employeeID : int) : Void
```

Die Methode zum Ablehnen von Mitarbeiter Anfragen.

```
+ approveRequest(employeeID : int) : Void
```

Die Methode zum Akzeptieren von Mitarbeiteranfragen.

### 6.7.4 ListReviewActivity



Einsichten-Bildschirm

```
ListReviewActivity extends AppCompatActivity
```



## Attribut

- adapter : ReviewsAdapter

Der Adapter für die scrollbare Liste von Einsichten.

## Methoden

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

## 6.7.5 ListReviewViewModel

ListUserViewModel extends ViewModel

### Methoden

+ getAllReviews() : List<Review>

Empfängt eine Liste mit allen Einsichten vom Repository.

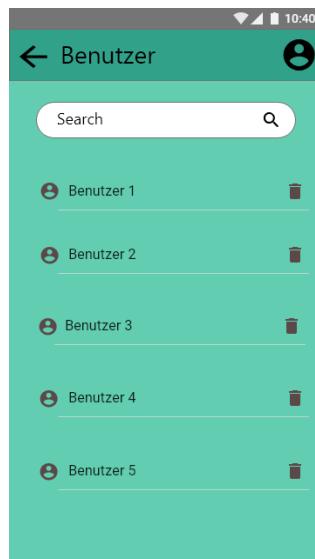
+ deleteReview(reviewID: int) : Void

Initialisiert das Löschen einer Einsicht.

+ searchReviews(input: String) : List<Review>

Sendet die Suchwörter mithilfe von Repository zum Server und empfängt eine Liste mit den Einsichten, die als Ergebnis der Suche entstehen.

### 6.7.6 ListUserActivity



Benutzer-Bildschirm

ListUserActivity extends AppCompatActivity

#### Attribut

- adapter : EmployeeAdapter

Der Adapter für die scrollbare Liste von Mitarbeitern.

- adapter : StudentAdapter

Der Adapter für die scrollbare Liste von Studenten.

#### Methode

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust



### 6.7.7 ListUserViewModel

ListUserViewModel extends ViewModel

#### Methoden

+ getAllEmployees() : List<Employee>

Empfängt eine Liste mit allen Mitarbeitern vom Repository.

+ getAllStudents() : List<Student>

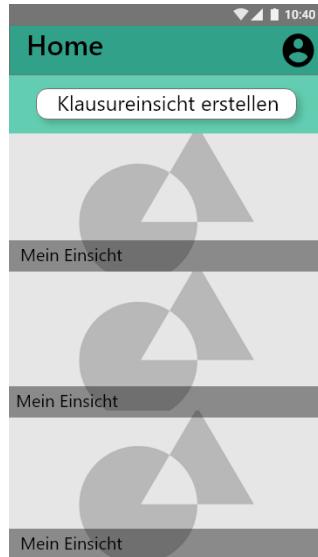
Empfängt eine Liste mit allen Studenten vom Repository.

+ deleteUser(id : int)

Initialisiert das Löschen eines Benutzers.

## 6.8 ui.employee

### 6.8.1 HomeEmployeeActivity



Home-Bildschirm

HomeEmployeeActivity extends AppCompatActivity

#### Attribut

- adapter : ReviewsAdapter

Der Adapter für die scrollbare Liste von Einsichten von Reviews

#### Methode

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.8.2 HomeEmployeeViewModel

HomeEmployeeViewModel extends ViewModel

#### Methode

+ getEmployeeReviews(employeeID: int) : List

Empfängt eine Liste aller Einsichten, die von dem Mitarbeiter erstellt wurde, vom Repository.

### 6.8.3 WaitingScreenActivity



Waiting-Bildschirm

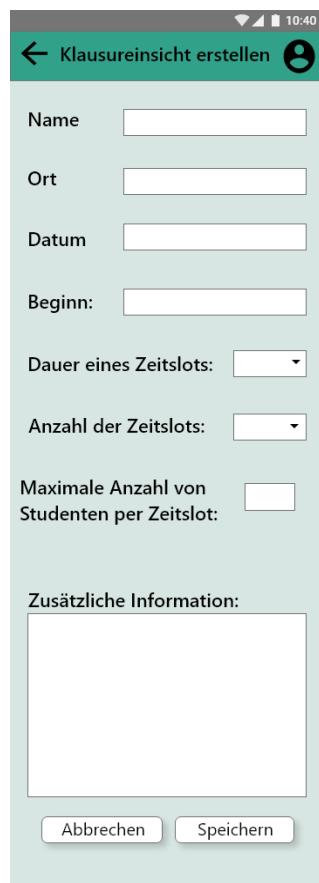
WaitingScreenActivity extends AppCompatActivity

#### Methode

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.8.4 ReviewCreationActivity



Einsicht-Bildschirm

ReviewCreationActivity extends AppCompatActivity

#### Methode

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.8.5 ReviewCreationViewModel

```
ReviewCreationViewModel extends ViewModel
```

#### Methoden

```
+ createReview(name : String, room: String, date : Timestamp,  
timeslotLength : int, numberOfWorkslots : int,  
maxStudentsPerTimeslot : int, employeeID : int, info : String)  
: Void
```

Die Methode zum Erstellens von Einsichten. Die Parameter der Methode werden an das Repository weitergeschickt, damit das Einsicht Objekt erstellt wird.

```
+ getReviewName(reviewID : int): String
```

Empfängt den Namen der Einsicht vom Repository, damit der Namen auf den Bildschirm angezeigt wird, wenn der Mitarbeiter seine Einsicht ändert will.

```
+ setReviewName(name : String) : Void
```

Setzt den Namen der Einsicht, wenn der Mitarbeiter den Namen der Einsicht ändert oder die Einsicht erstellt wird.

```
+ getReviewPlace(reviewID : int) : String
```

Empfängt das Ort der Einsicht vom Repository, damit der Ort auf dem Bildschirm angezeigt wird.

```
+ setReviewPlace(place : String) : Void
```

Setzt den Ort der Einsicht, wenn der Mitarbeiter seine Einsicht ändert oder die Einsicht erstellt wird.

```
+ getReviewDate(reviewID : int) : Timestamp
```

Gibt das Datum der Einsicht aus dem Timestamp Objekt, dass vom Repository empfängt wird, zurück, damit das Datum auf den Bildschirm angezeigt werden kann.

```
+ setReviewDate(year : int, month : int, day : int, hour : int,
```

```
minutes : int) : Void
```

Erstellt ein Timestamp Objekt, wo das Datum und die Zeit der Einsicht gespeichert werden.

```
+ getReviewTime(reviewID: int) : int
```

Gibt die Zeit der Einsicht aus dem Timestamp Objekt zurück, damit die Zeit auf dem Bildschirm angezeigt werden kann.

```
+ getReviewTimeSlotsLength(reviewID : int) : int
```

Empfängt die Dauer der Einsichten vom Repository, damit die Dauer auf dem Bildschirm angezeigt werden kann.

```
+ setReviewTimeSlotsLength(length : int) : Void
```

Setzt die Dauer der Einsichten, wenn der Mitarbeiter die Dauer ändern will oder die Einsicht erstellt wird.

```
+ getReviewTimeSlotsNumber(reviewID : int) : int
```

Empfängt die Zahl der Zeitslots vom Repository, damit die Zahl auf dem Bildschirm angezeigt werden kann.

```
+ setReviewTimeSlotsNumber(timeSlotsNumber : int) : Void
```

Setzt die Zahl der Zeitslots beim Erstellen oder bei der Änderung einer Einsicht.

```
+ getReviewInfo(reviewID : int) : String
```

Empfängt die zusätzliche Information der Einsicht vom Repository, damit die Information auf dem Bildschirm angezeigt werden kann.

```
+ setReviewInfo(info : String) : Void
```

Setzt die zusätzliche Information der Einsicht beim Erstellen oder bei der Änderung einer Einsicht.

```
+ setMaxStudentsPerTimeslot(max : int, reviewID : int) : Void
```

Setzt die maximale Zahl von Studenten per Zeitslot, wenn der Mitarbeiter diese Zahl ändern will oder die Einsicht erstellt wird.

```
+ getMaxStudentsPerTimeslot(reviewID : int) : int
```

Empfängt die maximale Zahl von Studenten per Zeitslot vom Repository, damit diese Zahl auf dem Bildschirm angezeigt werden kann.

#### 6.8.6 ReviewScreenActivity



Einsicht-Bildschirm

ReviewScreenActivity extends AppCompatActivity

#### Methode

```
+ onCreate(Bundle savedInstanceState) : Void
```

Die Methode zum Erstellen der Activity ohne Datenverlust

#### 6.8.7 ReviewScreenViewModel

ReviewScreenViewModel extends ViewModel

**Methoden**

```
+ getTimeslots(reviewID) : List
```

Empfängt eine Liste mit allen Zeitslots, die zu einer Einsicht gehören, vom Repository, damit die Zeitslots auf dem Bildschirm angezeigt werden.

## 6.9 ui.student

### 6.9.1 HomeStudentActivity



Home-Bildschirm

HomeStudentActivity extends AppCompatActivity

#### Attribut

- adapter : ReviewsAdapter

Der Adapter für die Liste von Einsichten von getStudentsReviews(studentID : int)

#### Methode

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.9.2 HomeStudentViewModel

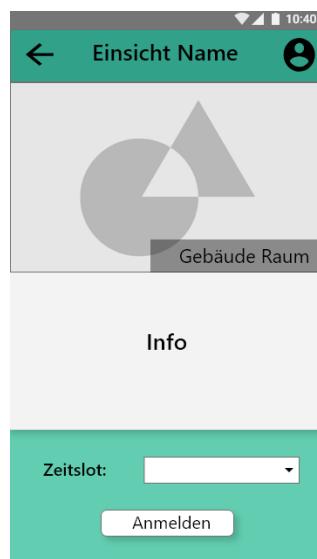
HomeStudentViewModel extends ViewModel

#### Methode

+ getStudentsReviews(studentID : int) : List

Empfängt eine Liste mit der Einsichten von Repository, für die der Student sich angemeldet hat, damit die Liste mithilfe von ReviewsAdapter auf den Bildschirm angezeigt wird.

### 6.9.2.1 ReviewSignUpActivity



Bildschirm zur Anmeldung bei einer Klausureinsicht

ReviewSignUpActivity extends AppCompatActivity

#### Methode

+ onCreate(Bundle savedInstanceState) : Void



Die Methode zum Erstellen der Activity ohne Datenverlust

### 6.9.3 ReviewSignUpViewModel

ReviewSignUpViewModel extends ViewModel

#### Methoden

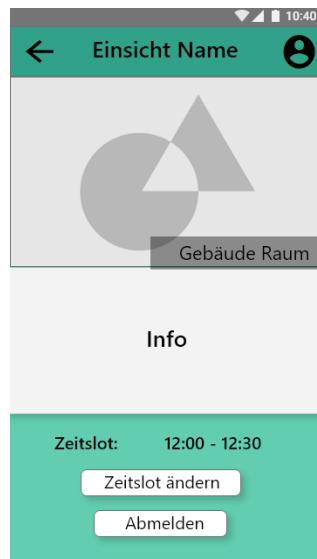
+ getAllReviews() : List<Review>

Empfängt eine Liste mit allen Einsichten von Repository, damit die Liste mithilfe dem **ReviewsAdapter** auf dem Bildschirm angezeigt werden kann.

+ getReviews(input : String) : List<Review>

Empfängt eine Liste mit den Einsichten, die als Ergebnis einer Suche entstehen, vom Repository, damit die Liste mithilfe dem **ReviewsAdapter** auf dem Bildschirm angezeigt werden kann.

#### 6.9.4 ReviewScreenActivity



Einsichtanmeldung-Bildschirm

ReviewScreenActivity extends AppCompatActivity

##### Methode

+ onCreate(Bundle savedInstanceState) : Void

Die Methode zum Erstellen der Activity ohne Datenverlust

#### 6.9.5 ReviewScreenViewModel

ReviewScreenViewModel extends ViewModel

##### Attribute

- isRegistered : Boolean

False, wenn der Student für diese Einsicht nicht angemeldet ist, und True ansonsten.

##### Methoden



+ `getReviewInfo(reviewID : int) : String`

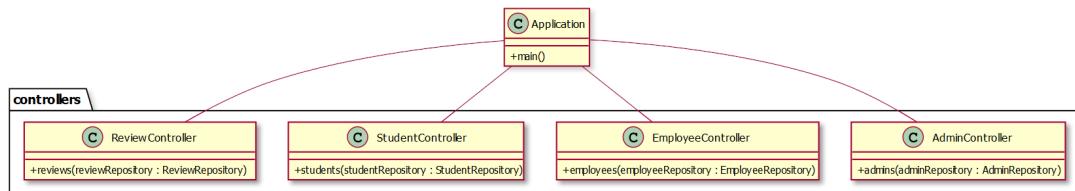
Empfängt die zusätzliche Information der Einsicht von Repository, damit die Information auf den Bildschirm angezeigt wird.

+ `getTimeslot(reviewID : int, studentID : int) : int`

Empfängt das Zeitslot, für das der Student sich angemeldet hat, von Repository, damit das Zeitslot auf den Bildschirm angezeigt wird. (Wenn `isRegistered = true`)

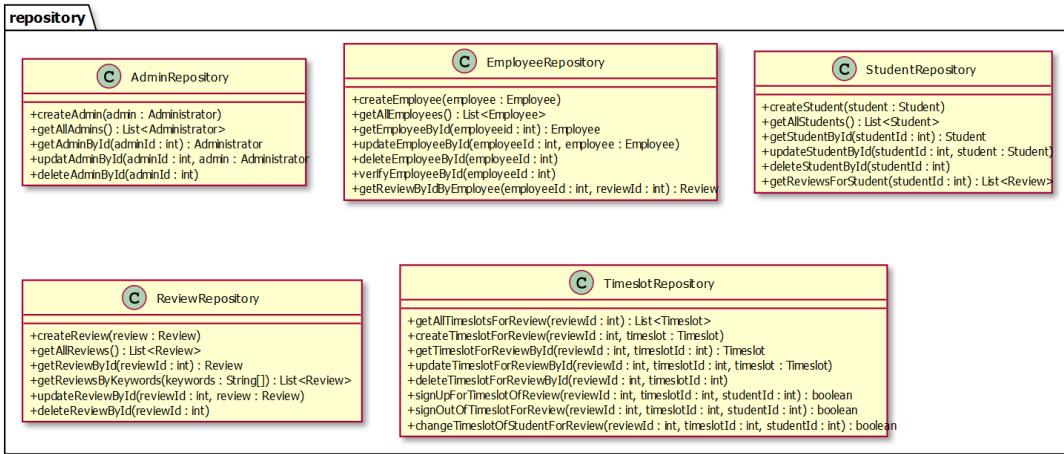
## 7 Paketübersicht - Server

### 7.1 controllers



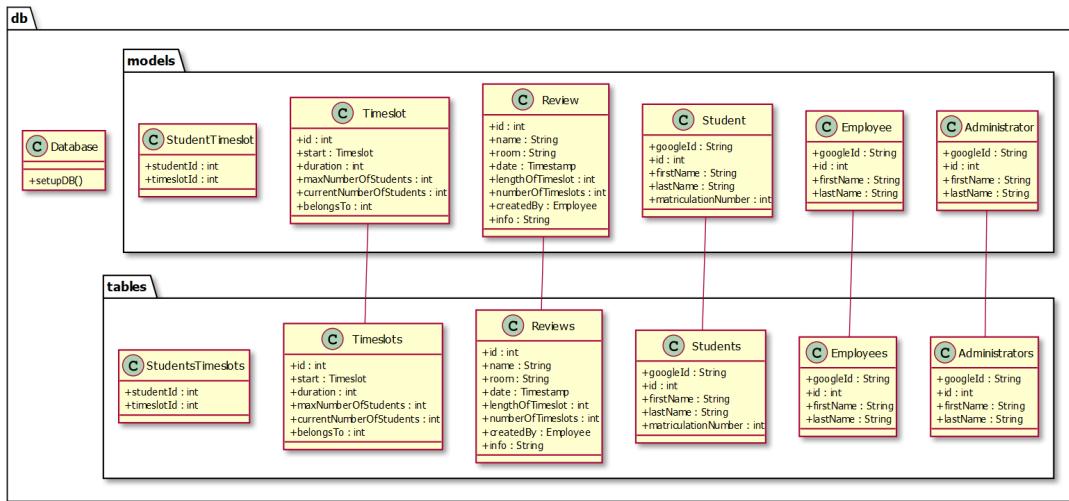
In diesem Paket befinden sich die Klassen AdminController, EmployeeController, StudentController und ReviewController, die die REST Endpunkte des Servers zu Client Apss zur Verfügung stellen. Außerdem sind diese Klassen für die Bearbeitung und Weiterleitung einkommender Anfragen von Clients verantwortlich.

## 7.2 repository



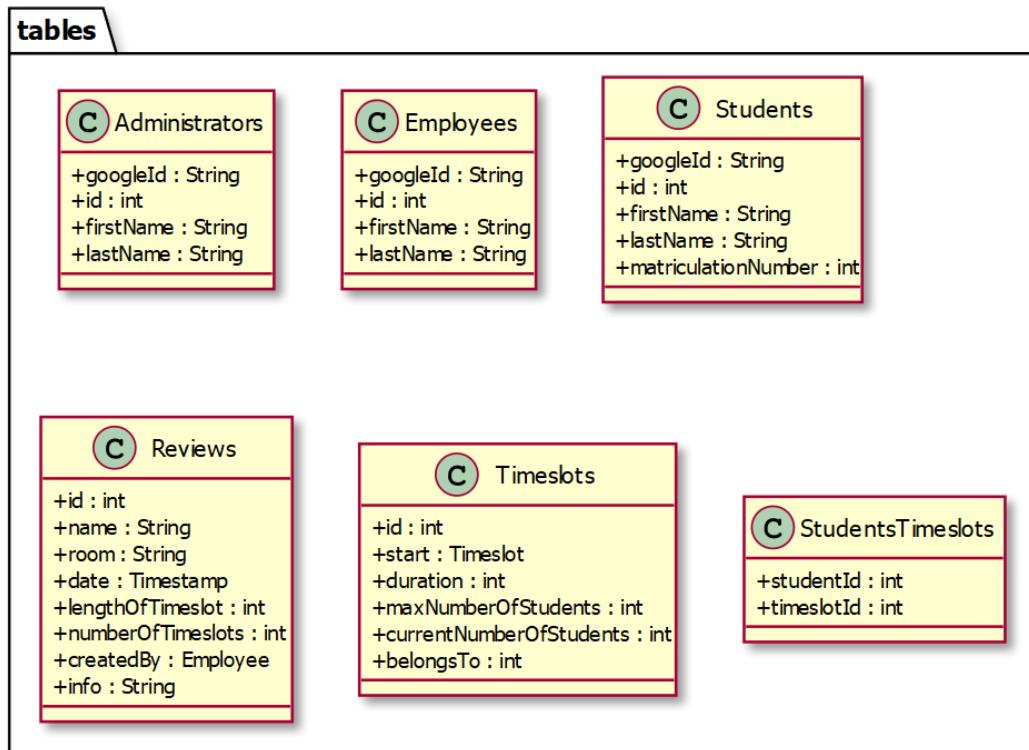
In diesem Paket sind die Klassen AdminRepository, EmployeeRepository, StudentRepository, ReviewRepository sowie TimeslotRepository zu finden. Sie dienen als Schnittstelle zwischen den Controllern und Datenquellen wie Datenbank und sind beispielsweise für Lese- und Schreibzugriffe auf die Datenbank zuständig.

### 7.3 db



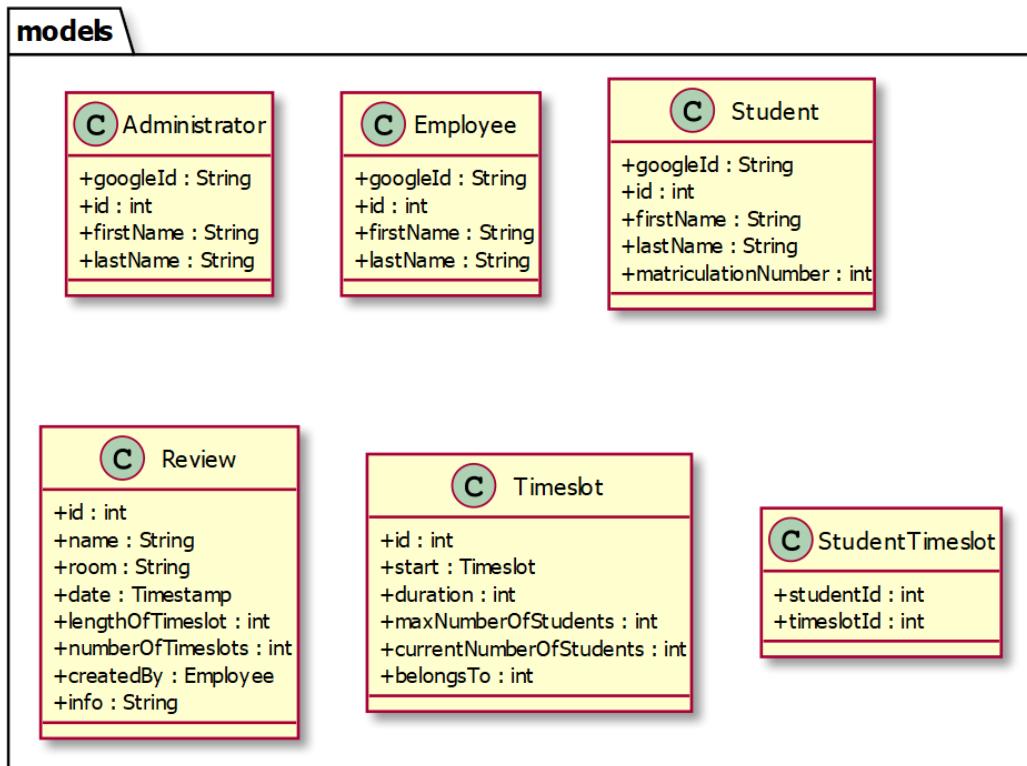
Hier befindet sich die Klasse Database, die vom Exposed Framework stammt und die die Verbindung mit der Datenbank realisiert.

## 7.3.1 db.tables



Hier befinden sich die Klassen Administrators, Employees, Students, Reviews und Timeslots, die die Tabellen in der Datenbank repräsentieren.

## 7.3.2 db.models



In diesem Paket sind die Klassen Administrator, Employee, Student, Review und Timeslot, die einzelne Einträge in der Datenbank darstellen. Sie stelle außerdem Lese- und Schreibzugriffe zu den Datenbanktabellen zur Verfügung.

## 8 Klassenübersicht - Server

### 8.1 Application

Diese Klasse ist die Hauptklasse des Server Moduls, deren Funktion vor allem darin besteht, den Server zu starten sowie den HTTP Port des Servers und die REST Endpunkten zu Client Anwendungen zur Verfügung zu stellen.

#### Methode

```
+ main()
```

Diese Methode sorgt dafür, den Server zu starten und alle für seine Funktion nötige Teile zu konfigurieren, wie z.B. REST Endpunkte sowie Serialisation bzw. Deserialisation von JSON Objekten.

### 8.2 controllers

#### 8.2.1 AdminController

Diese Klasse dient dazu, die REST Endpunkte für Operationen verbunden mit Admins zur Verfügung zu stellen.

#### Methode

```
+ admins(adminRepository: AdminRepository): Void
```

Diese Methode stellt die REST Endpunkte und die entsprechenden Operationen zur Verfügung.

#### 8.2.2 EmployeeController

Diese Klasse dient dazu, die REST Endpunkte für Operationen verbunden mit Mitarbeitern zur Verfügung zu stellen.

#### Methode

```
+ employees(employeeRepository: EmployeeRepository): Void
```

Diese Methode stellt die REST Endpunkte und die entsprechenden Operationen zur Verfügung.

### 8.2.3 **StudentController**

Diese Klasse dient dazu, die REST Endpunkte für Operationen verbunden mit Studenten zur Verfügung zu stellen.

#### **Methode**

```
+ students(studentRepository: StudentRepository): Void
```

Diese Methode stellt die REST Endpunkte und die entsprechenden Operationen zur Verfügung.

### 8.2.4 **ReviewController**

Diese Klasse dient dazu, die REST Endpunkte für Operationen verbunden mit Klausureinsichten zur Verfügung zu stellen.

#### **Methode**

```
+ reviews(reviewRepository: ReviewRepository, timeslotRepository: TimeslotRepository): Void
```

Diese Methode stellt die REST Endpunkte und die entsprechenden Operationen zur Verfügung.

## 8.3 **repository**

### 8.3.1 **AdminRepository**

Diese Klasse dient dazu, Lese- sowie Schreiboperationen für Admins zur Verfügung zu stellen.

#### **Methoden**

```
+ createAdmin(admin: Admin): Void
```

Diese Methode dient dazu, einen neuen Admin in der App zu erstellen. Das Admin Objekt kommt von der Client App und wird zur Erstellung des Admins in der Datenbank benutzt.

+ `getAllAdmins(): List<Admin>`

Diese Methode eine Liste mit allen Admins in der App zurück.

+ `getAdminById(adminId: Int): Admin`

Diese Methode bekommt ein adminId Parameter von der Client App und sucht mithilfe von diesem Parameter nach einem Admin in der App. Falls einer gefunden wird, wird er zurückgegeben.

+ `updateAdminById(adminId: Int, admin: Admin): Void`

Diese Methode bekommt ein adminId Parameter von der Client App und modifiziert den entsprechenden Admin, falls er existiert.

+ `deleteAdminById(adminId: Int): Void`

Diese Methode bekommt ein adminId Parameter von der Client App und löscht den entsprechenden Admin vollständig von der App, falls er existiert.

### 8.3.2 EmployeeRepository

Diese Klasse dient dazu, Lese- sowie Schreiboperationen für Mitarbeiter zur Verfügung zu stellen.

#### Methoden

+ `createEmployee(employee: Employee): Void`

Diese Methode dient zum Erstellen eines neuen Mitarbeiters in der App

+ `getAllEmployees(): List<Employee>`

Diese Methode liefert eine Liste mit allen Mitarbeitern in der App

+ `getEmployeeById(employeeId: Int): Employee`

Diese Methode bekommt ein employeeId Parameter und gibt den entsprechenden Mitarbeiter zurück, falls er existiert.

+ `updateEmployeeById(employeeId: Int, employee: Employee): Void`

Diese Methode bekommt ein employeeId Parameter und modifiziert den entsprechenden Mitarbeiter, falls er existiert.

+ `deleteEmployeeById(employeeId: Int): Void`

Diese Methode bekommt ein employeeId Parameter und löscht den entsprechenden Mitarbeiter, falls er existiert.

+ `verifyEmployeeById(employeeId: Int): Boolean`

Diese Methode bekommt ein employeeId Parameter und verifiziert einen Mitarbeiter in der App und gibt dabei true zurück. Sonst gibt sie false zurück.

+ `getReviewsByEmployee(employeeId: Int): List<Review>`

Diese Methode bekommt ein employeeId Parameter und gibt alle mit dem Mitarbeiter mit dieser ID assoziierten Klausureinsichten zurück.

+ `getReviewByIdByEmployee(employeeId: Int, reviewId: Int): Review`

Diese Methode bekommt ein employeeId Parameter und ein reviewId Parameter und gibt die entsprechende Klausureinsicht assoziiert mit dem entsprechenden Mitarbeiter zurück.

### 8.3.3 **StudentRepository**

Diese Klasse dient dazu, Lese- sowie Schreiboperationen für Studenten zur Verfügung zu stellen.

#### Methoden

+ `createStudent(student: Student): Void`

Diese Methode erstellt einen neuen Student in der App mithilfe vom Parameter, den sie bekommt.

```
+ getAllStudents(): List<Student>
```

Diese Methode gibt eine Liste mit allen Studenten in der App zurück.

```
+ getStudentById(studentId: Int): Student
```

Diese Methode bekommt einen studentId Parameter und gibt den entsprechenden Studenten zurück, falls er existiert.

```
+ updateStudentById(studentId: Int, student: Student): Void
```

Diese Methode bekommt einen studentId Parameter und modifiziert den entsprechenden Studenten, falls er existiert.

```
+ deleteStudentById(studentId: Int): Void
```

Diese Methode bekommt einen studentId Parameter und löscht den entsprechenden Studenten, falls er existiert.

```
+ getReviewsForStudent(studentId: Int): List<Review>
```

Diese Methode bekommt einen studentId Parameter und gibt eine Liste an Klausureinsichten, für die der entsprechende Student sich angemeldet hat.

### 8.3.4 ReviewRepository

Diese Klasse dient dazu, Lese- sowie Schreiboperationen für Klausureinsichten zur Verfügung zu stellen.

#### Methoden

```
+ createReview(review: Review): Void
```

Diese Methode dient zum erstellen einer neuen Klausureinsicht in der App.

```
+ getAllReviews(): List<Review>
```

Diese Methode gibt eine Liste mit allen Klausureinsichten in der App zurück.

```
+ getReviewById(reviewId: Int): Review
```

Diese Methode gibt eine Klausureinsicht mit der entsprechenden ID, falls sie existiert.

```
+ getReviewsByKeywords(keywords: String[]): List<Review>
```

Diese Methode gibt eine Liste mit allen Klausureinsichten in der App, die zumindest ein Wort vom Parameter als Information enthalten.

```
+ updateReviewById(reviewId: Int, review: Review): Void
```

Diese Methode modifiziert die Klausureinsicht mit der entsprechenden ID, falls sie existiert.

```
+ deleteReviewById(reviewId: Int): Void
```

Diese Methode löscht die Klausureinsicht mit der entsprechenden ID, falls sie existiert.

### 8.3.5 TimeslotRepository

Diese Klasse dient dazu, Lese- sowie Schreiboperationen für Zeitslots zur Verfügung zu stellen.

#### Methoden

```
+ getAllTimeslotsForReview(reviewId: Int): List<Timeslot>
```

Diese Methode gibt eine Liste mit allen Zeitslots, die zu der entsprechenden Klausureinsicht in der App gehören.

```
+ createTimeslotForReview(reviewId: Int, timeslot: Timeslot): Void
```

Diese Methode erstellt einen neuen Zeitslot für die Klausureinsicht mit der entsprechenden ID.

```
+ getTimeslotForReviewById(reviewId: Int, timeslotId: Int): Timeslot
```

Diese Methode gibt einen Zeitslot mit einer bestimmten ID, der zu einer bestimmten Klausureinsicht gehört, zurück.

```
+ updateTimeslotForReviewById(reviewId: Int, timeslotId: Int, timeslot: Timeslot): Void
```

Diese Methode modifiziert einen bestimmten Zeitslot, der zu einer bestimmten Klausureinsicht gehört, falls er existiert.

```
+ deleteTimeslotForReviewById(reviewId: Int, timeslotId: Int): Void
```

Diese Methode löscht einen bestimmten Zeitslot, der zu einer bestimmten Klausureinsicht gehört, falls er existiert.

```
+ signUpForTimeslotOfReview(reviewId: Int, timeslotId: Int, studentId: Int): Boolean
```

Diese Methode meldet einen Studenten für einen Zeitslot, der zu einer bestimmten Klausureinsicht gehört, an, und gibt true zurück, falls die Anmeldung erfolgreich war. Sonst gibt sie false zurück.

```
+ signOutOfTimeslotForReview(reviewId: Int, timeslotId: Int, studentId: Int): Boolean
```

Diese Methode meldet einen Studenten von einem Zeitslot, der zu einer bestimmten Klausureinsicht gehört, ab, und gibt true zurück, falls die Abmeldung erfolgreich war. Sonst gibt sie false zurück.

```
+ changeTimeslotOfStudentForReview(reviewId: Int, timeslotId: Int, studentId: Int): Timeslot
```

Diese Methode meldet einen Studenten von einem Zeitslot einer Klausureinsicht ab, und meldet ihn für einen anderen an, falls das möglich ist. Wenn die Ummeldung erfolgreich war, gibt die Methode true zurück, sonst gibt sie false zurück.

## 8.4 db

### 8.4.1 Database

#### Methode

```
+ setupDB()
```



Diese Methode sorgt dafür, eine Verbindung mit der Datenbank zu erstellen und die Kommunikation mit der Datenbank zu initialisieren.

## 8.5 db.models

### 8.5.1 Admin

Diese Klasse stellt einen Datenbankeintrag eines Admins dar.

#### Attribute

+ googleId: String

Stellt die einheitliche ID vom Google Account des Admins dar.

+ id: Int

Stellt die interne einheitliche ID eines Admins dar.

+ firstName: String

Stellt den Vornamen des Admins dar.

+ lastName: String

Stellt den Namen des Admins dar.

### 8.5.2 Employee

Diese Klasse stellt einen Datenbankeintrag eines Mitarbeiters dar.

#### Attribute

+ googleId: String

Stellt die einheitliche ID vom Google Account des Mitarbeiters dar.

+ id: Int

Stellt die interne einheitliche ID eines Mitarbeiters dar.

+ `firstName: String`

Stellt den Vornamen des Mitarbeiters dar.

+ `lastName: String`

Stellt den Namen des Mitarbeiters dar.

### 8.5.3 **Student**

Diese Klasse stellt einen Datenbankeintrag eines Studenten dar.

#### **Attribute**

+ `googleId: String`

Stellt die einheitliche ID vom Google Account des Studenten dar.

+ `id: Int`

Stellt die interne einheitliche ID eines Studenten dar.

+ `firstName: String`

Stellt den Vornamen des Studenten dar.

+ `lastName: String`

Stellt den Namen des Studenten dar.

+ `matriculationNumber: Int`

Stellt die Matrikelnummer eines Studenten dar.

### 8.5.4 **Review**

Diese Klasse stellt einen Datenbankeintrag einer Klausureinsicht dar.

#### **Attribute**

+ `id: Int`

Stellt die interne einheitliche ID einer Klausureinsicht

+ `name: String`

Stellt den Namen der Klausureinsicht dar.

+ `location: String`

Stellt den Raum, in dem die Klausureinsicht stattfindet, dar.

+ `date: Timestamp`

Stellt den Datum, an dem die Klausureinsicht stattfindet dar.

+ `lengthOfTimeslot: Int`

Stellt die Länge der Zeitslots einer Klausureinsicht dar.

+ `numberOfTimeslots: Int`

Stellt die Anzahl Zeitslots einer Klausureinsicht dar.

+ `createdBy: Int`

Stellt die ID des Mitarbeiters, der die Klausureinsicht erstellt hat.

+ `description: String`

Stellt eine Beschreibung der Klausureinsicht dar.

### 8.5.5 Timeslot

Diese Klasse stellt einen Datenbankeintrag eines Zeitslots dar.

#### Attribute

+ `id: Int`

Stellt die interne einheitliche ID eines Zeitslots



+ `startTime: Timestamp`

Stellt die Startzeit eines Zeitslots dar.

+ `duration: Int`

Stellt die Dauer eines Zeitslots dar.

+ `maxNumberOfStudents: Int`

Stellt die maximale Anzahl an Studenten, die sich für den Zeitslot anmelden können.

+ `currentNumberOfStudents: Int`

Stellt die aktuelle Anzahl an Studenten, die sich für den Zeitslot angemeldet haben.

+ `belongsTo: Int`

Stellt die ID der Klausureinsicht, zu der der Zeitslot gehört, dar.

### 8.5.6 **StudentTimeslot**

Diese Klasse stellt die Beziehung in der Datenbank zwischen Studenten und Zeitslots dar.

#### Attribute

+ `studentId: Int`

Stellt die interne einheitliche ID eines Studenten in der Beziehung dar.

+ `timeslotId: Int`

Stellt die interne einheitliche ID eines Zeitslots in der Beziehung dar.

## 8.6 db.tables

### 8.6.1 Admins

Diese Klasse stellt die Tabelle Admins in der Datenbank dar.

#### Attribute

+ googleId: String

Stellt die einheitliche ID vom Google Account des Admins dar.

+ id: Int

Stellt die interne einheitliche ID eines Admins dar.

+ firstName: String

Stellt den Vornamen des Admins dar.

+ lastName: String

Stellt den Namen des Admins dar.

### 8.6.2 Employees

Diese Klasse stellt die Tabelle Employees in der Datenbank dar.

#### Attribute

+ googleId: String

Stellt die einheitliche ID vom Google Account des Mitarbeiters dar.

+ id: Int

Stellt die interne einheitliche ID eines Mitarbeiters dar.

+ firstName: String

Stellt den Vornamen des Mitarbeiters dar.

+ lastName: String

Stellt den Namen des Mitarbeiters dar.

### 8.6.3 Students

Diese Klasse stellt die Tabelle Students in der Datenbank dar.

#### Attribute

+ googleId: String

Stellt die einheitliche ID vom Google Account des Studenten dar.

+ id: Int

Stellt die interne einheitliche ID eines Studenten dar.

+ firstName: String

Stellt den Vornamen des Studenten dar.

+ lastName: String

Stellt den Namen des Studenten dar.

+ matriculationNumber: Int

Stellt die Matrikelnummer eines Studenten dar.

### 8.6.4 Reviews

Diese Klasse stellt die Tabelle Reviews in der Datenbank dar.

#### Attribute

+ id: Int

Stellt die interne einheitliche ID einer Klausureinsicht

+ **name: String**

Stellt den Namen der Klausureinsicht dar.

+ **location: String**

Stellt den Raum, in dem die Klausureinsicht stattfindet, dar.

+ **date: Timestamp**

Stellt den Datum, an dem die Klausureinsicht stattfindet dar.

+ **lengthOfTimeslot: Int**

Stellt die Länge der Zeitslots einer Klausureinsicht dar.

+ **numberOfTimeslots**

:

Int Stellt die Anzahl Zeitslots einer Klausureinsicht dar.

+ **createdBy: Int**

Stellt die ID des Mitarbeiters, der die Klausureinsicht erstellt hat.

+ **description: String**

Stellt eine Beschreibung der Klausureinsicht dar.

### 8.6.5 **Timeslots**

Diese Klasse stellt die Tabelle Timeslots in der Datenbank dar.

#### Attribute

+ **id: Int**

Stellt die interne einheitliche ID eines Zeitslots

+ **startTime: Timestamp**



Stellt die Startzeit eines Zeitslots dar.

+ `duration: Int`

Stellt die Dauer eines Zeitslots dar.

+ `maxNumberOfStudents: Int`

Stellt die maximale Anzahl an Studenten, die sich für den Zeitslot anmelden können.

+ `currentNumberOfStudents: Int`

Stellt die aktuelle Anzahl an Studenten, die sich für den Zeitslot angemeldet haben.

+ `belongsTo: Int`

Stellt die ID der Klausureinsicht, zu der der Zeitslot gehört, dar.

### 8.6.6 StudentsTimeslots

Diese Klasse stellt die Tabelle StudentsTimeslots in der Datenbank dar.

#### Attribute

+ `studentId: Int`

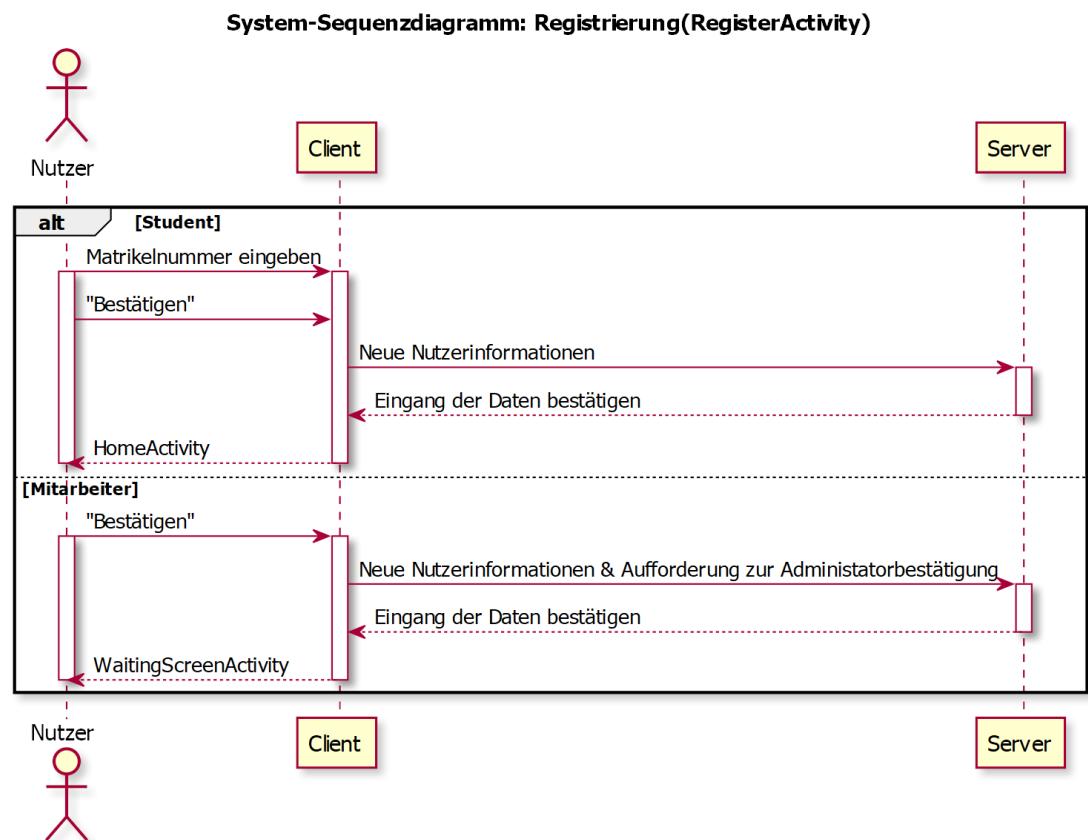
Stellt die interne einheitliche ID eines Studenten in der Beziehung dar.

+ `timeslotId: Int`

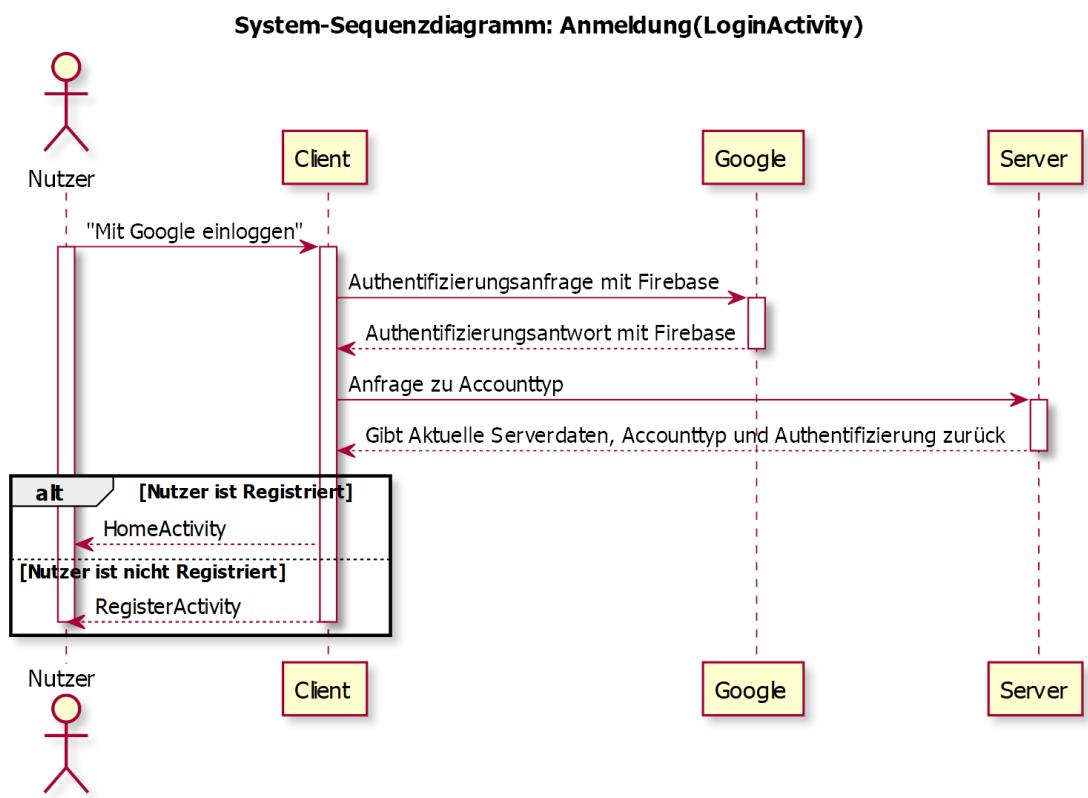
Stellt die interne einheitliche ID eines Zeitslots in der Beziehung dar.

## 9 Diagramme

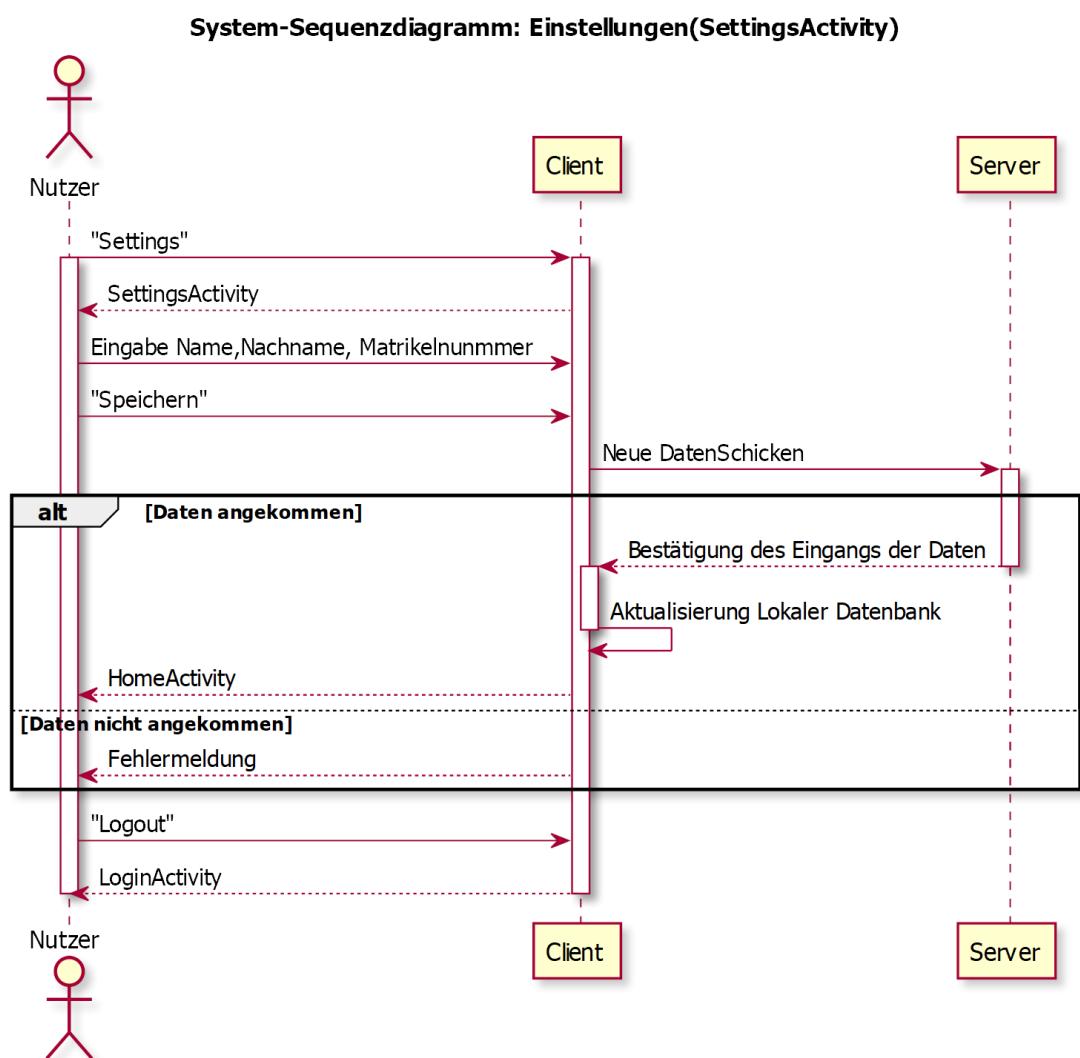
### 9.1 Systemsequenzdiagramme



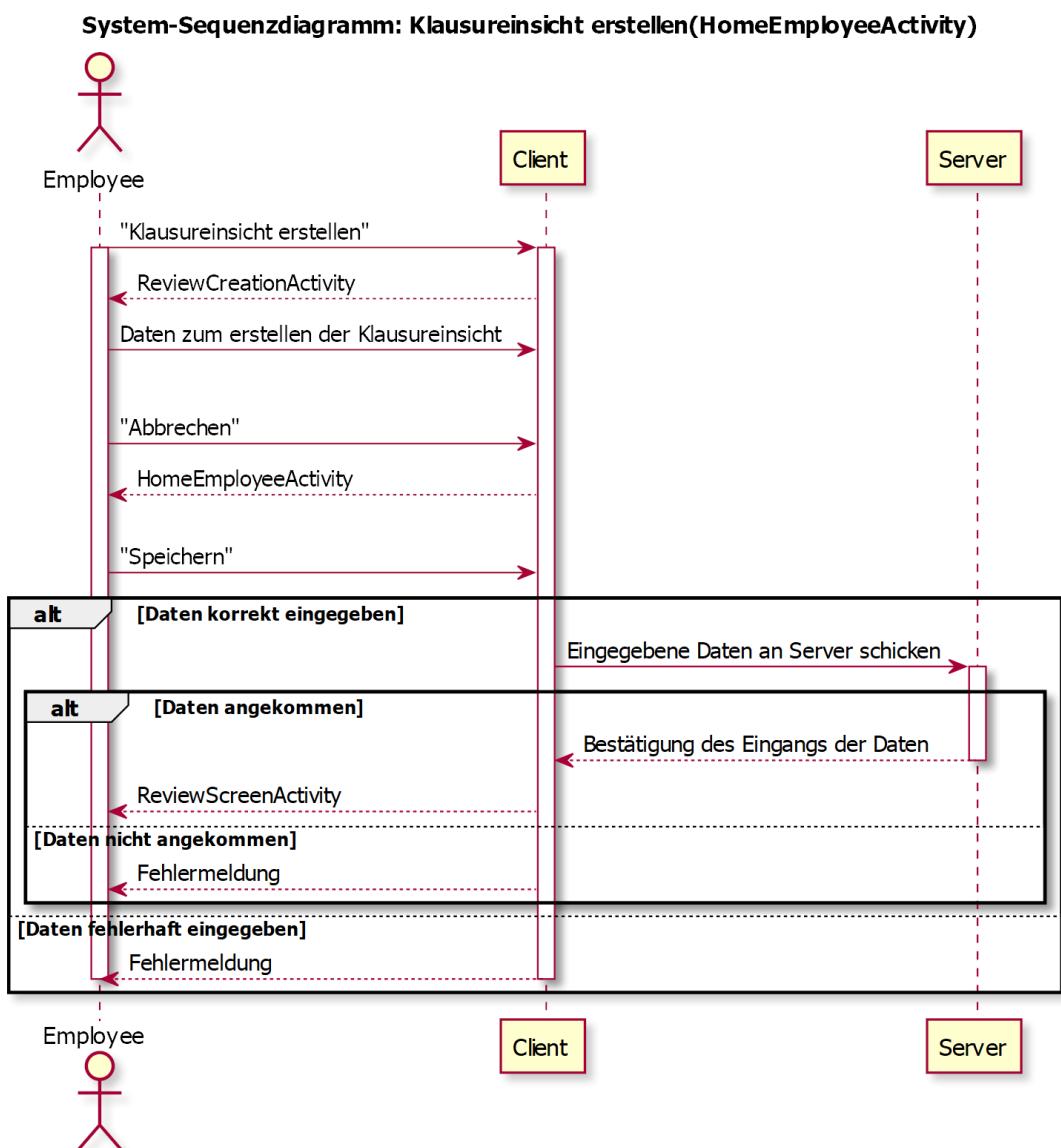
Übersichtliche Darstellung des Registrierungsvorgangs.



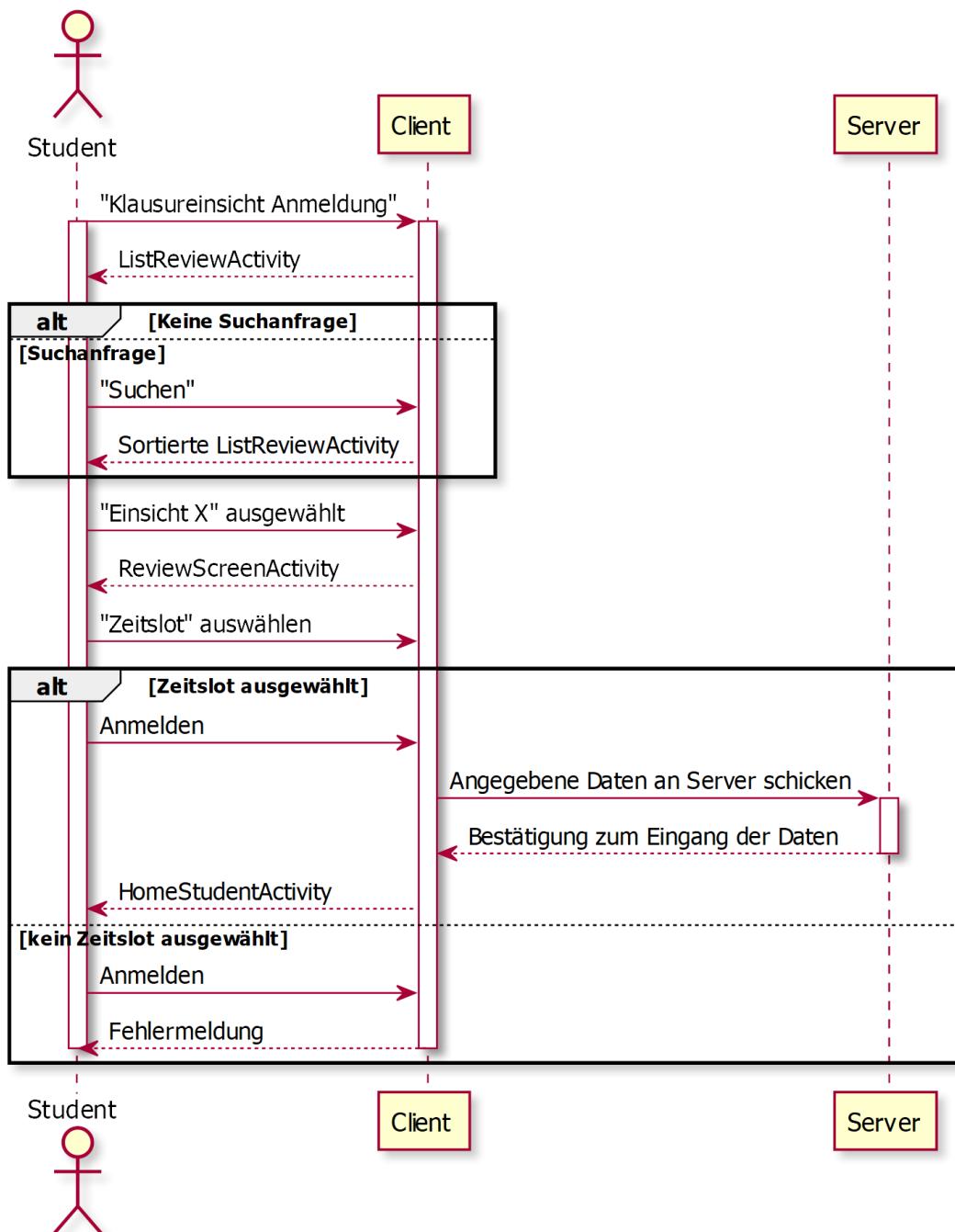
Übersichtliche Darstellung des Anmeldevorgangs.



Übersichtliche Darstellung der Einstellungsmöglichkeiten.

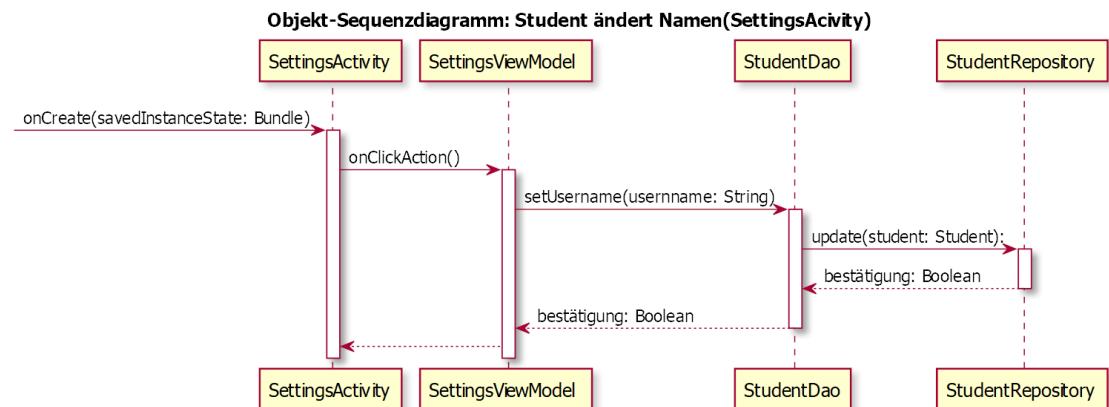


Übersichtliche Darstellung des Vorgangs, mit einem Mitarbeiter-Account, Klausureinsicht zu erstellen.

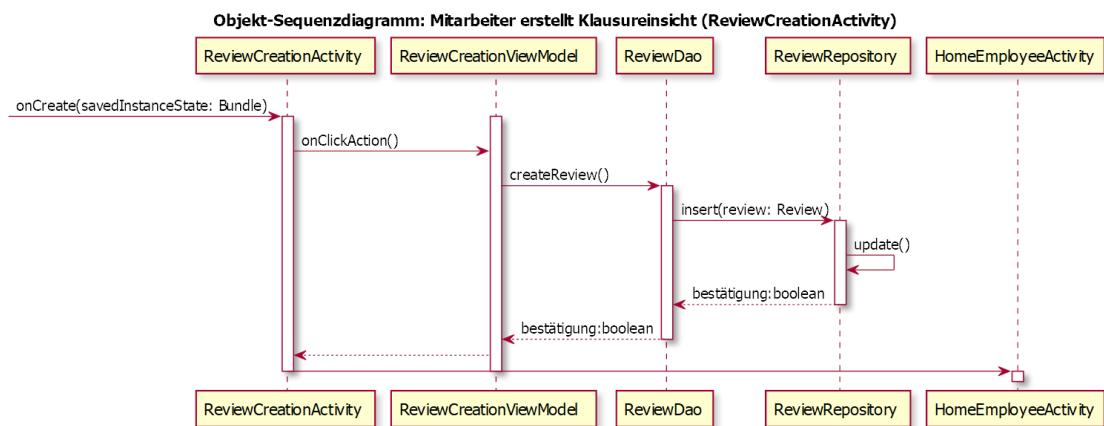
**System-Sequenzdiagramm: Klausureinsichtanmeldung(HomeStudentActivity)**


Übersichtliche Darstellung des Vorgangs sich, mit einem Studenten-Account, für Klausureinsichten anzumelden. Übersichtliche Darstellung der Einstellungsmöglichkeiten.

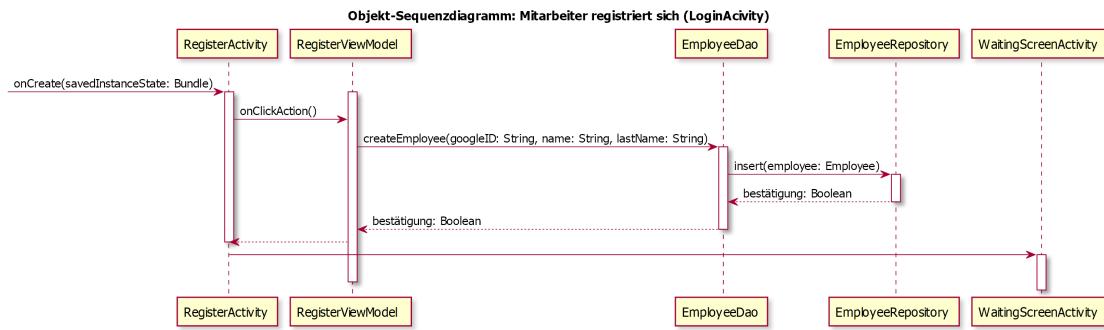
## 9.2 Objektsequenzdiagramme



Vereinfachte Darstellung der Einstellungsfunktion am Beispiel einer Namensänderung.



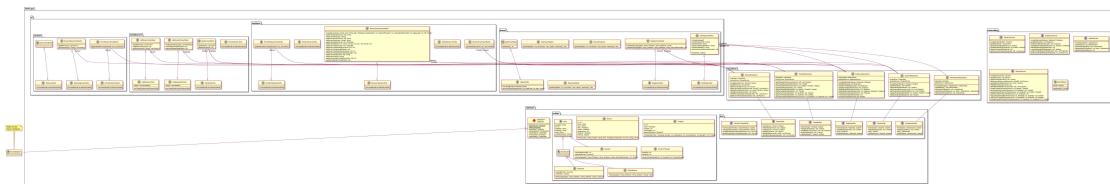
Vereinfachte Darstellung des Vorgangs, als Mitarbeiter, eine Klausureinsicht zu erstellen.



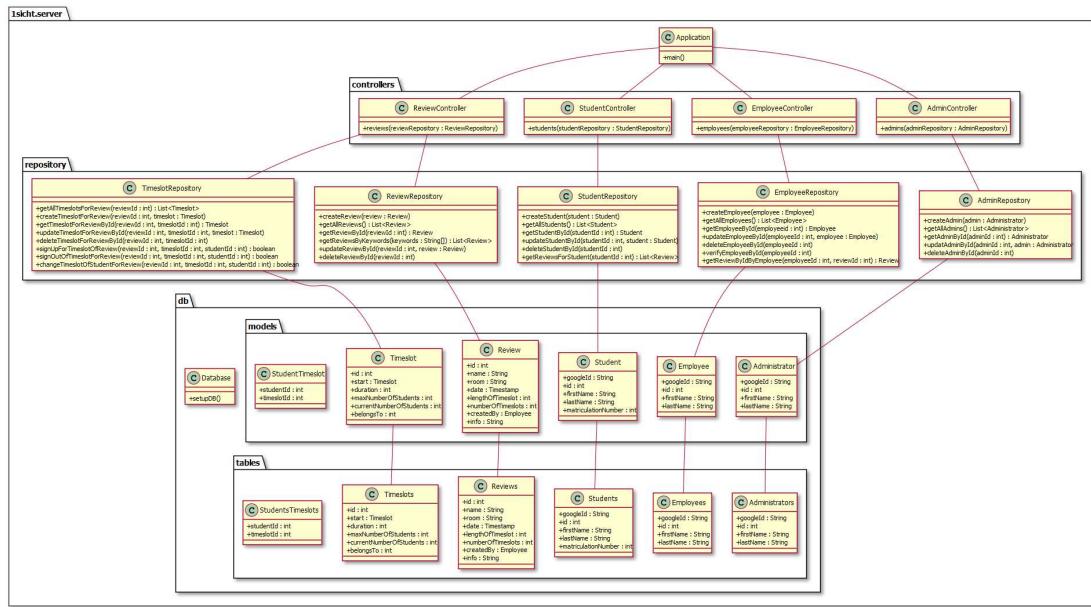
Vereinfachte Darstellung des Registrierungsvorgangs am Beispiel eines Mitarbeiters.

## 9.3 Klassendiagramme

### 9.3.1 Klassendiagramm der Client-App



### 9.3.2 Klassendiagramm des Servers



## Glossar

**Activity** Bildschirm den der Nutzer sieht und mit dem er Interagieren kann.

**Entity** Eine Entity ist eine von den drei Hauptkomponenten in der Room-Bibliothek, die die Daten des repräsentierenden Objekts beinhaltet.

**Fragment** Fragments dienen dazu z.b. bei Rotation des Handys mehrere Views darzustellen.

**Model** Enthält Daten die mit der View dargestellt werden.

**Server** Nicht-lokale Funktionalitäten und Daten werden dort gespeichert.

**Singleton** Der Singleton Entwurfsmuster stellt sicher, dass es nur eine einzige Instanz einer Klasse gibt.

**SQLite** Die SQLite Bibliothek implementiert eine kleine, schnelle, eigenständige, hochzuverlässige SQL-Datenbank-Engine mit vollem Funktionsumfang.

**View** Enthält Informationen zur Darstellung für Benutzer.

**ViewModel** Klasse in der Daten ohne Android-Kontext verarbeitet werden und die in Activities angezeigt werden.

## Akronyme

**DAO** Data Access Object.

**GUI** Graphical User Interface.

**MVVM** Model-View-ViewModel.