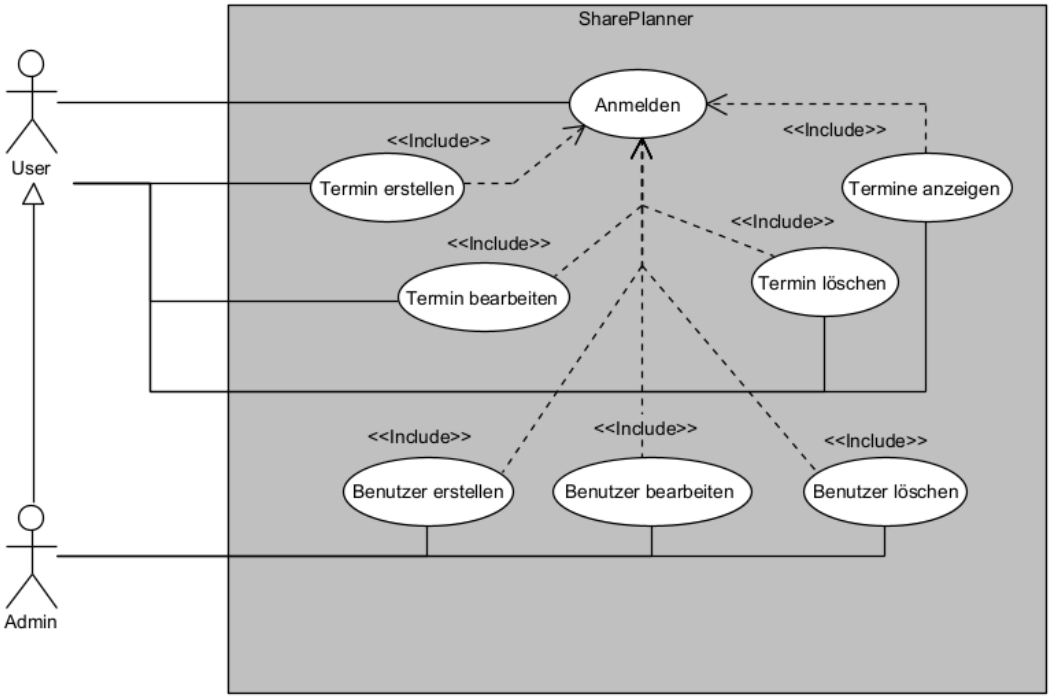
IMPLEMENTIERUNGS DOKUMENTATION

Share Planner

Marc Beyer 20%, Marco Kühn 60%, Alex Rechtin 20%

Inhalt

Use Case Diagramm



Im Diagramm kann man sehen, dass wir nur zwei Aktöre haben. Einmal den Benutzer und den Admin. Der Admin ist eine Generalisierung des Benutzers, da dieser Alles kann was der Benutzer auch kann, nur dass er die Benutzer zusätzlich verwalten kann.

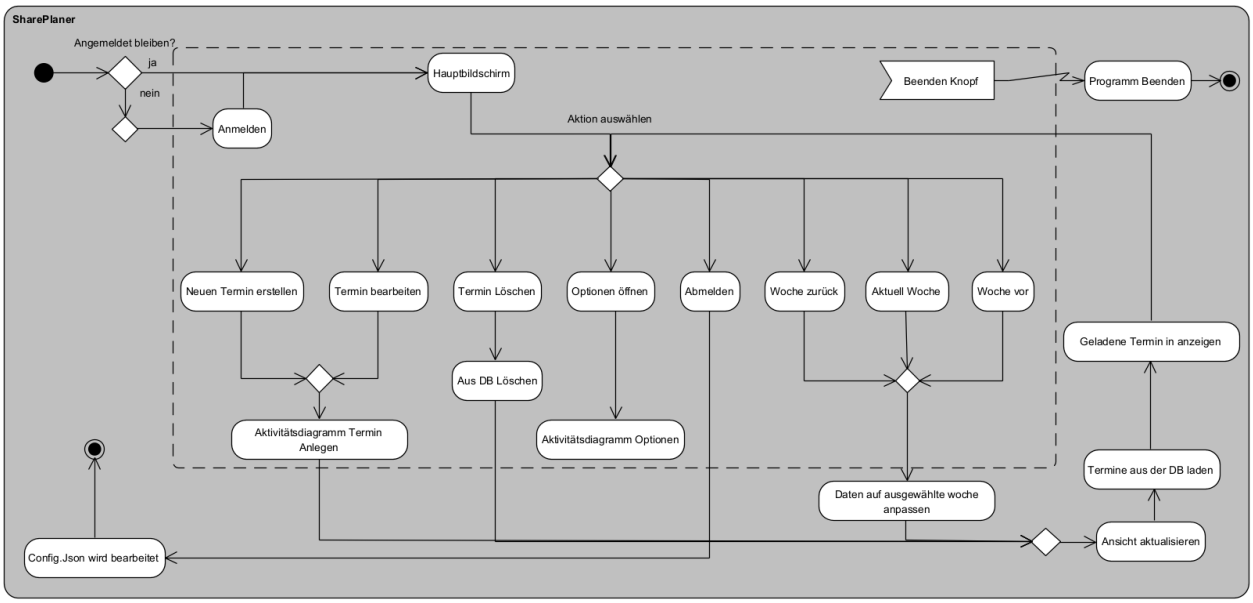
Mann kann erkennen das man angemeldet sein muss, bevor man überhaupt etwas im Programm benutzen kann.

Während der Benutzer nur seine Termine erstellen und verwalten kann, kann der Admin dies auch mit den Terminen von anderem Benutzer. Beispielsweise kann ein Admin die Privaten Termine eines anderen Nutzers sehen oder Termine bearbeiten oder Löschen.

Ein Admin ist allerdings hauptsächlich dazu da, andere Benutzer hinzuzufügen oder vorhanden zu verwalten.

Aktivitätsdiagramm

Allgemein



Dieses Diagramm ist unser Hauptdiagramm. Es beginnt mit der Anmeldung, nach welcher der Benutzer in den Hauptbildschirm gelangt. Dort hat der Nutzer Acht Möglichkeiten. Als erstes ist es möglich die angezeigte Woche zu verändern, indem der eine Woche in die Vergangenheit oder die Zukunft wechselt. Außerdem kann er auch direkt mit einem Knopf auf die aktuelle Woche zurückkehren. Bei jedem Wochen Wechsel, werden die Termine für die Woche aus der Datenbank geladen und nur die angezeigt, die der Benutzer sehen darf.

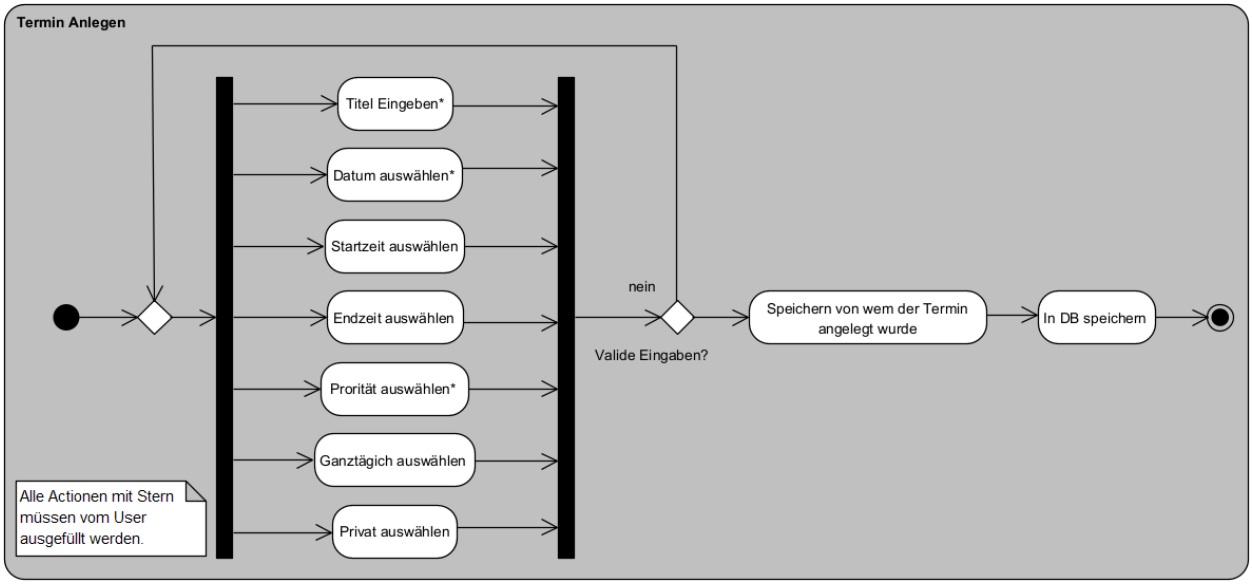
Als nächstes kommen wir zum wichtigsten. Der Benutzer kann hier einen neuen Termin anlegen. Dies wird im nächsten Diagramm genauer erläutert. Wenn dies geschehen ist, werden die Termine ebenfalls neu geladen. Falls der Benutzer nun einen angelegten Termin bearbeiten will, kann er dies ebenfalls tun. Da der Ablauf dieses Prozesses sehr ähnlich zum Anlegen ist, wird dieser ebenfalls im nächsten Diagramm beschrieben.

Nachdem nun ein paar Termine vorhanden sind, kann der Benutzer diese auch löschen. Dies geht einfach über einen Knopf am Termin. Wenn dieser gedrückt wird, wird der Termin aus der Datenbank gelöscht, und die Ansicht wird erneut aktualisiert.

Kommen wir nun zu den Optionen. Der Ablauf der Optionen wird im Diagramm Optionen genauer erläutern. Ist dieses aber abgeschlossen kommt der Benutzer zurück in den Hauptbildschirm und die Termine werden, falls nötig, neu geladen.

Zum Schluss kann der Benutzer sich Abmelden. Wenn er dies tut, werden die Einstellungen so angepasst, dass die Anmelde daten nicht mehr gespeichert sind und das Programm wird daraufhin beendet.

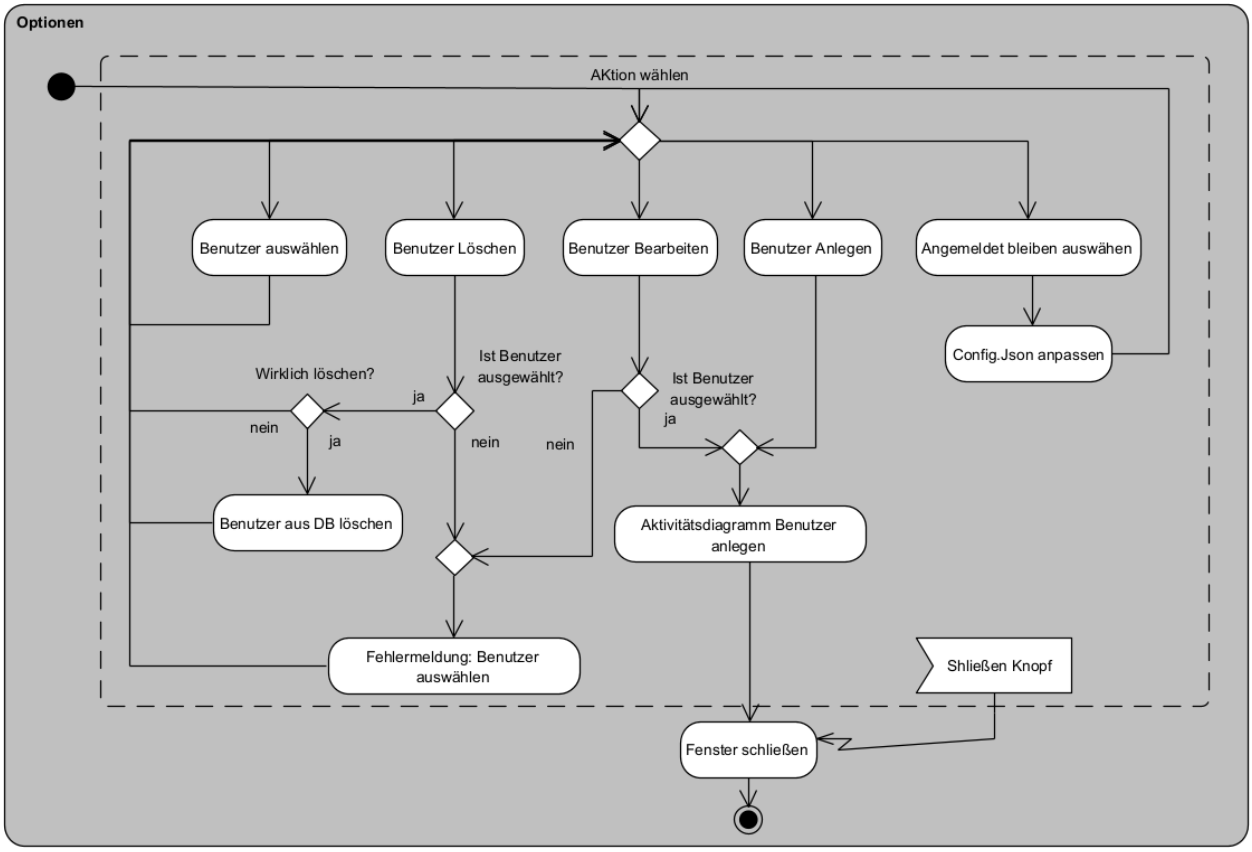
Termin Anlegen/Bearbeiten



Wenn der Benutzer vom Termin anlegen das Diagramm aufruft, sind keine der Felder mit Daten gefüllt. Falls er aber von Termin bearbeiten kommt, sind die Felder mit Daten von besagtem Termin gefüllt.

Der Benutzer hat nun die Möglichkeit alle Daten in beliebiger reinfolge anzugeben. Wenn dies geschehen ist und der Speichern Knopf gedrückt wurde, werden die Daten validiert. Falls sie nicht valide sind, bekommt der Benutzer eine passende Fehlermeldung und er muss die Daten Anpassen. Sollte aber alles Stimmen, wird noch im Hintergrund gespeichert wer diesen Termin erstellt und diese Daten werden dann in der Datenbank gespeichert. Danach ist der Prozess zu Ende.

Optionen



Bei dem Aufruf dieses Fensters, hat der Benutzer 5 Optionen.

Er kann einen Benutzer aus einer Combobox auswählen, in der alle Benutzer stehen.

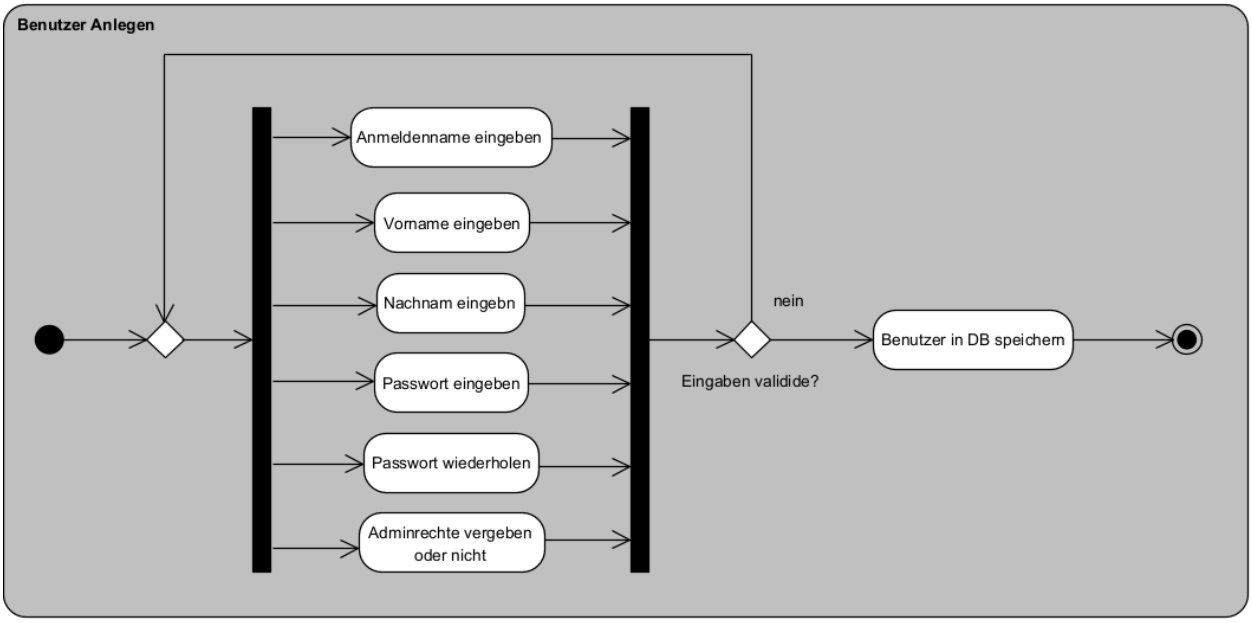
Außerdem dann ein und ausschalten, ob er angemeldet bleiben möchte. Daraufhin werden die Einstellungen passen angepasst.

Wenn der Benutzer ein anderer User löschen möchte und den Knopf drückt, wird überprüft, ob ein Nutzer ausgewählt wurde oder nicht. Falls nicht wird der Benutzer darauf hingewiesen, dass er die tun muss. Falls einer ausgewählt ist, wird nochmal abgefragt, ob diese Benutzer gelöscht werden soll. Dies kann dann bestätigt werden. Falls ja, wird der Benutzer aus der Datenbank gelöscht.

Wenn ein Benutzer bearbeitet werden muss, folgt gleiche Überprüfung, ob ein Benutzer ausgewählt wurde, wie beim Löschen. Falls einer ausgewählt wurde, kommt zu einem Fenster wo man den Benutzer bearbeiten kann. Dieser Prozess wird im nächsten Diagramm genauer beschrieben. Nachdem dies geschehen ist, wird das Optionsfenster geschlossen.

Wenn man einen Benutzer anlegen möchte, kommt man zu einem Fenster, wo man dies tun kann. Dieser Prozess ist zum Bearbeiten sehr ähnlich und wird deswegen ebenfalls im nächsten Diagramm genauer beschrieben. Wenn dieser Vorgang abgeschlossen ist, wird das Optionsfenster geschlossen.

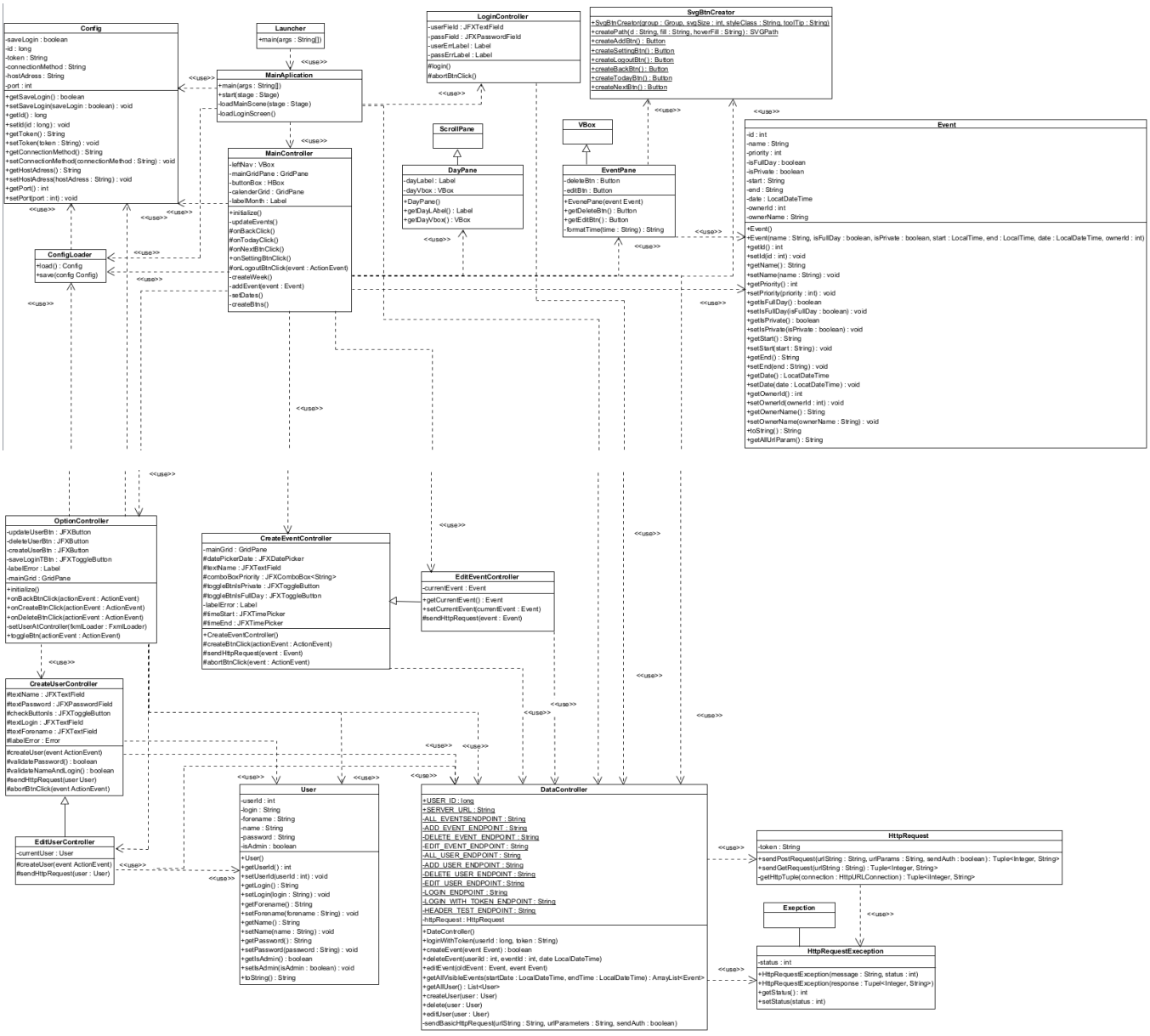
Benutzer Anlegen/Bearbeiten



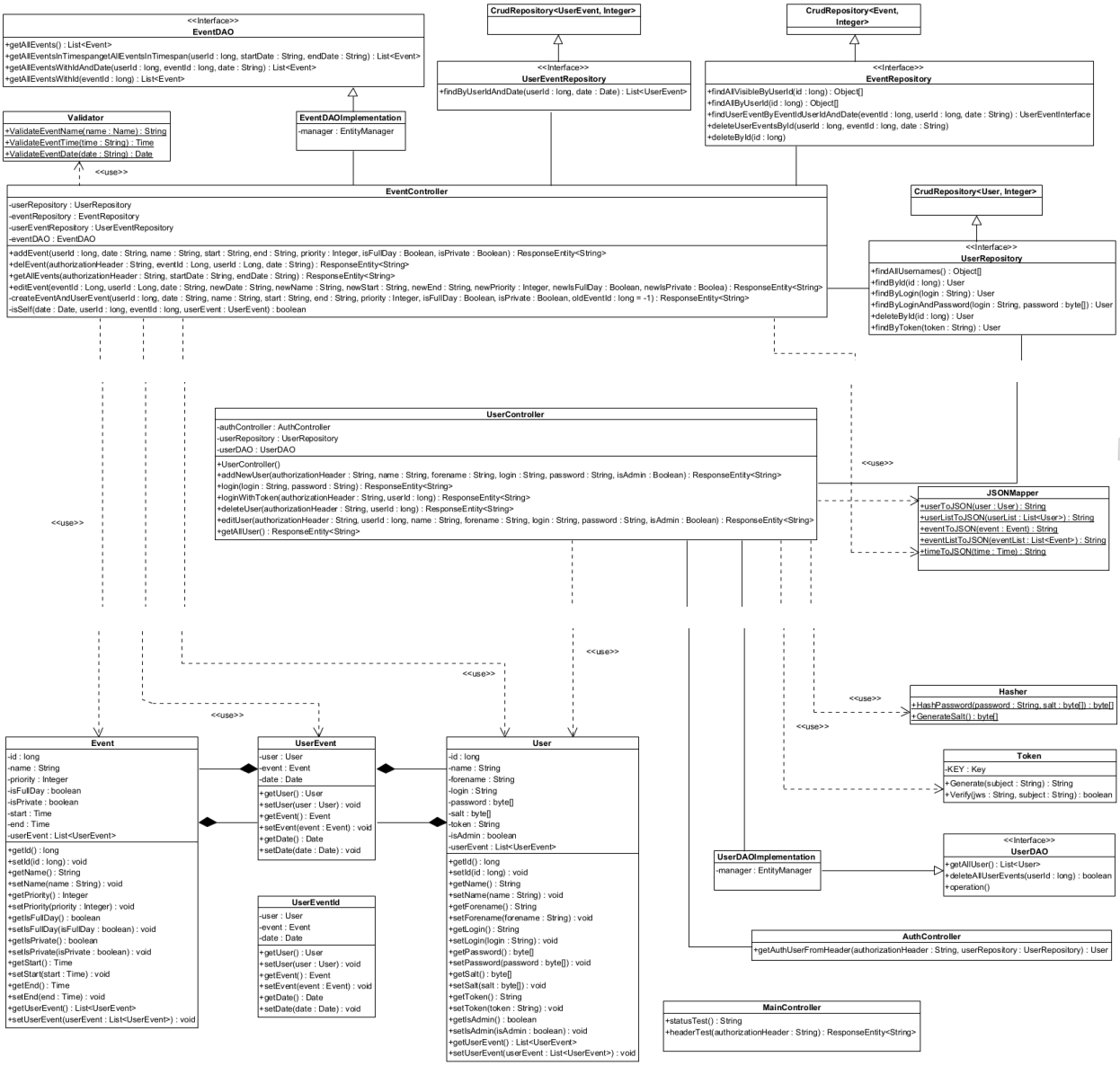
Wenn der Benutzer von Bearbeiten kommt, sind schon Daten von der zu bearbeiten Benutzer eingetragen. Andernfalls sind alle Eingabefelder leer. Nun kann der Benutzer hier der in beliebiger Reihenfolge die Eingaben ausfüllen. Wenn er nun aus speichern klickt, werden die überprüft. Sollten sie nicht valide sein, muss Benutzer sie nochmal anpassen. Andernfalls wird der Benutzer in der Datenbank gespeichert und der Prozess ist beendet.

Klassen Diagramm

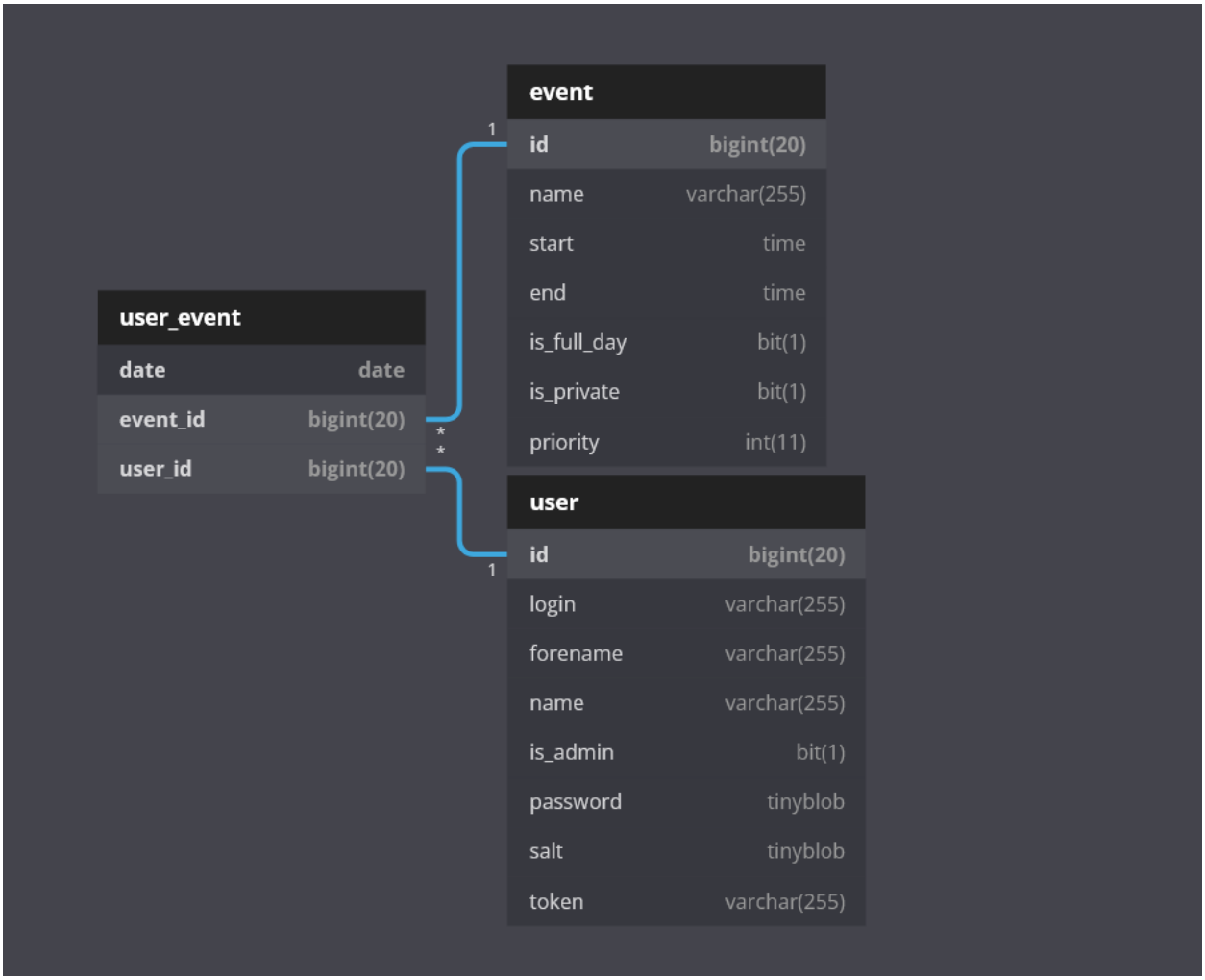
Client



Server


<<Interface>>
EventDAO
+getAllEvents() : List<Event>
+getAllEventsInTimespangetAllEventsInTimespan(userId : long, startDate : String, endDate : String) : List<Event>
+getAllEventsWithIdAndDate(userId : long, eventId : long, date : String) : List<Event>
+getAllEventsWithId(eventId : long) : List<Event>
CrudRepository<UserEvent, Integer>
<<Interface>>
UserEventRepository
+findByUserIdAndDate(userId : long, date : Date) : List<UserEvent>
CrudRepository<Event, Integer>
<<Interface>>
EventRepository
+findAllVisibleByUserId(id : long) : Object[]
+findAllByUserId(id : long) : Object[]
+findUserEventByEventIdUserIdAndDate(eventId : long, userId : long, date : String) : UserEventInterface
+deleteUserEventsById(userId : long, eventId : long, date : String)
+deleteById(id : long)
Validator
+ValidateEventName(name : Name) : String
+ValidateEventTime(time : String) : Time
+ValidateEventDate(date : String) : Date
EventDAOImplementation
-manager : EntityManager
<<use>>
EventController
-userRepository : UserRepository
-eventRepository : EventRepository
-userEventRepository : UserEventRepository
-eventDAO : EventDAO
+addEvent(userId : long, date : String, name : String, start : String, end : String, priority : Integer, isFullDay : Boolean, isPrivate : Boolean) : ResponseEntity<String>
+delEvent(authorizationHeader : String, eventId : Long, userId : Long, date : String) : ResponseEntity<String>
+getAllEvents(authorizationHeader : String, startDate : String, endDate : String) : ResponseEntity<String>
+editEvent(eventId : Long, userId : Long, date : String, newDate : String, newName : String, newStart : String, newEnd : String, newPriority : Integer, newIsFullDay : Boolean, newIsPrivate : Boolea) : ResponseEntity<String>
-createEventAndUserEvent(userId : long, date : String, name : String, start : String, end : String, priority : Integer, isFullDay : Boolean, isPrivate : Boolean, oldEventId : long = -1) : ResponseEntity<String>
-isSelf(date : Date, userId : long, eventId : long, userEvent : UserEvent) : boolean
CrudRepository<User, Integer>
<<Interface>>
UserRepository
+findAllUsernames() : Object[]
+findById(id : long) : User
+findByLogin(login : String) : User
+findByLoginAndPassword(login : String, password : byte[]) : User
+deleteById(id : long) : User
+findByToken(token : String) : User
UserController
-authController : AuthController
-userRepository : UserRepository
-userDAO : UserDAO
+UserController()
+addNewUser(authorizationHeader : String, name : String, forename : String, login : String, password : String, isAdmin : Boolean) : ResponseEntity<String>
+login(login : String, password : String) : ResponseEntity<String>
+loginWithToken(authorizationHeader : String, userId : long) : ResponseEntity<String>
+deleteUser(authorizationHeader : String, userId : long) : ResponseEntity<String>
+editUser(authorizationHeader : String, userId : long, name : String, forename : String, login : String, password : String, isAdmin : Boolean) : ResponseEntity<String>
+getAllUser() : ResponseEntity<String>
<<use>>
JSONMapper
+userToJSON(user : User) : String
+userListToJSON(userList : List<User>) : String
+eventToJSON(event : Event) : String
+eventListToJSON(eventList : List<Event>) : String
+timeToJSON(time : Time) : String
Event
-id : long
-name : String
-priority : Integer
-isFullDay : boolean
-isPrivate : boolean
-start : Time
-end : Time
-userEvent : List<UserEvent>
+getId() : long
+setId(id : long) : void
+getName() : String
+setName(name : String) : void
+getPriority() : Integer
+setPriority(priority : Integer) : void
+getIsFullDay() : boolean
+setIsFullDay(isFullDay : boolean) : void
+getIsPrivate() : boolean
+setIsPrivate(isPrivate : boolean) : void
+getStart() : Time
+setStart(start : Time) : void
+getEnd() : Time
+setEnd(end : Time) : void
+getUserEvent() : List<UserEvent>
+setUserEvent(userEvent : List<UserEvent>) : void
UserEvent
-user : User
-event : Event
-date : Date
+getUser() : User
+setUser(user : User) : void
+getEvent() : Event
+setEvent(event : Event) : void
+getDate() : Date
+setDate(date : Date) : void
UserEventId
-user : User
-event : Event
-date : Date
+getUser() : User
+setUser(user : User) : void
+getEvent() : Event
+setEvent(event : Event) : void
+getDate() : Date
+setDate(date : Date) : void
User
-id : long
-name : String
-forename : String
-login : String
-password : byte[]
-salt : byte[]
-token : String
-isAdmin : boolean
-userEvent : List<UserEvent>
+getId() : long
+setId(id : long) : void
+getName() : String
+setName(name : String) : void
+getForename() : String
+setForename(forename : String) : void
+getLogin() : String
+setLogin(login : String) : void
+getPassword() : byte[]
+setPassword(password : byte[]) : void
+getSalt() : byte[]
+setSalt(salt : byte[]) : void
+getToken() : String
+setToken(token : String) : void
+getIsAdmin() : boolean
+setIsAdmin(isAdmin : boolean) : void
+getUserEvent() : List<UserEvent>
+setUserEvent(userEvent : List<UserEvent>) : void
Hasher
+HashPassword(password : String, salt : byte[]) : byte[]
+GenerateSalt() : byte[]
Token
-KEY : Key
+Generate(subject : String) : String
+Verify(jws : String, subject : String) : boolean
UserDAOImplementation
-manager : EntityManager
<<Interface>>
UserDAO
+getAllUser() : List<User>
+deleteAllUserEvents(userId : long) : boolean
+operation()
AuthController
+getAuthUserFromHeader(authorizationHeader : String, userRepository : UserRepository) : User
MainController
+statusTest() : String
+headerTest(authorizationHeader : String) : ResponseEntity<String>

Datenbank Schema



Unsere Datenbank besteht aus drei Tabellen. Eine für den User in dem wir allen seine wichtigen Daten speichern. Wichtig hierbei ist, dass wir das Passwort als Hash Speichern, damit man aus der Datenbank das Passwort nicht lesen kann. Dafür müssen wir ebenfalls einen Salt speichern, sodass wir damit überprüfen können, ob das eingegeben Passwort stimmt. Ebenfalls speichern wir für jeden Benutzer einen Token, welcher bei jeder Anfrage vorweggeschickt wird, sodass wir eine funktionierendes Rechtesystem haben.

In der Tabelle für die Events speichern wir alle Daten eines Termins.

Und zum Schluss, haben wir noch eine Tabelle, die die beiden anderen Tabellen miteinander verbindet, so dass Jeder User Events haben kann und jedes Event einem User zugeordnet ist.

Coding Guidelines

1 Formatierung

1.1 Allgemein

- Maximal 80 bis 100 Zeichen pro Zeile
- Wir Benutzen den Standard Formatierter der in IntelliJ IDEA eingebaut ist

1.2 Klammern

- Die werden Klammern wie folgend benutzen:

```
public void example() {
}
```

1.3 Arrays

- Für Arrays werden folgende Formatierungen benutzt:

```
String[] array = new String[]{"1", "2", "3", "4"}

    ODER:

String[] array = new String[]{
        "1",
        "2",
        "3",
        "4"
}
```

2. Benennung

2.1 Allgemein

- Alle Namen beinhalten nur ASCII Zeichen
- Die dürfen nicht mit zahlen beginnen
- Alle Namen werden auf Englisch geschrieben

2.2 Klassennamen

- Namen werden im ‚UpperCamelCase' geschrieben

2.3 Methodennamen

- Namen werden im ‚lowerCamelCase' geschrieben
- Sollen die Methoden beschreiben Bsp. ‚startProcess' oder ‚log'

2.4 Konstanten Namen

- Namen werden in ‚UPPER_CASE' geschrieben

2.5 Variabel Namen

- Namen werden im ‚lowerCamelCase' geschrieben
- Die Namen sollen beschreiben was in ihnen gespeichert ist

2.6 Dateinamen

- Namen werden im ‚lowerCamelCase' geschrieben