Software Engineering Übung, LVNr:

Übungsleiter: Hans Moritsch

Designmodell v.1.0

Projekttitel: ydio

Projekthomepage: ydio.sahann.at

Gruppenmitglieder:

MatNr	Nachname	Vorname	e-mail
0803000	Ilg	Benjamin	a0803000@unet.univie.ac.at
0907851	Sahann	Raphael	raphael.sahann@univie.ac.at
1126575	Mermi	Dagcan	mermi.dagcan@gmail.com
1025504	Czernecki	Mateusz	mathck@gmail.com

Erstellen sie ein Designmodell gemäß *Unified Process* das zumindest folgende Aspekte umfasst:

- Klassendesign
- Use-Case-Realization-Design
- Übersichtsklassendiagramm
- Architekturbeschreibungen

1 Klassendesign

1.1.1 AbstractUser

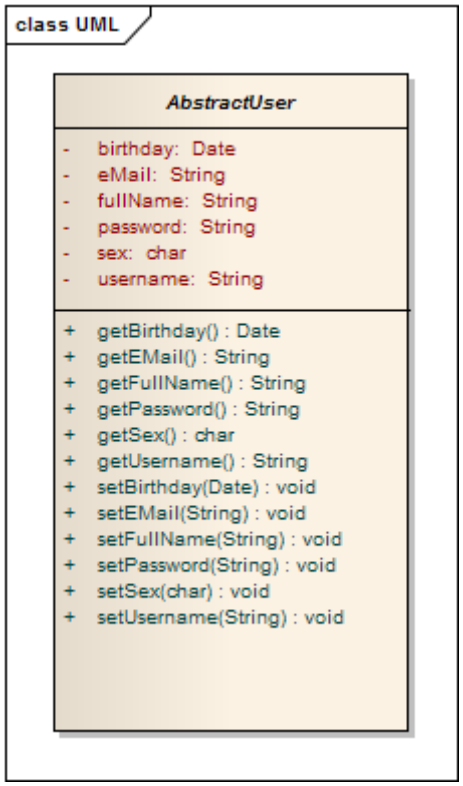
AbstractUser ist unsere abstrakte Benutzerklasse von dem alle anderen Benutzer erben.

Private Attribute:

- birthday: date
- eMail: String
- fullName: String
- password: String
- sex: char
- username: String

Da keine anderen Klassen direkt auf diese Variablen zugreifen sollen, sind sie private. Jeder dieser Attribute hat einen Setter/Getter.

- setBirthday nimmt eine Date variable entgegen und überprüft ob das Datum vor dem jetzigen Zeitpunkt liegt.
- setSex nimmt einen char entgegen und überprüft diesen (‘m’ oder ‘f’).

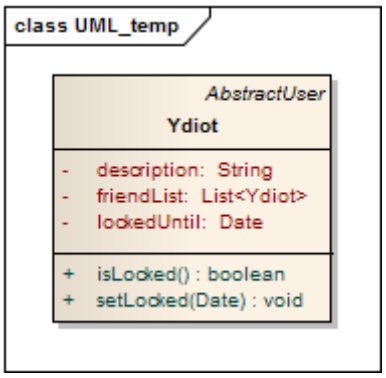


1.1.2 Ydiot

Ydioten erben vom AbstractUser und sind unsere Benutzer. Sie haben zusätzlich die privaten Variablen

- description: String
- locketUntil: Date bis zu diesem Zeitpunkt ist der User gesperrt

Die Methode [isLocked(): boolean] gibt zurück ob der User im moment gesperrt ist.
[setLocket(Date): void] darf von Moderatoren ausgeführt werden und gibt einen Zeitpunkt an bis zu welchem Zeitpunkt der Benutzer gesperrt sein soll.



1.1.3 Moderator, Administrator

Moderator erbt von AbsractUser und Administrator wiederrum von Moderator.

1.1.4 Forscher

Forscher erben von AbstractUser und haben sonst keine speziellen Eigentschaften.

1.1.5 SQL

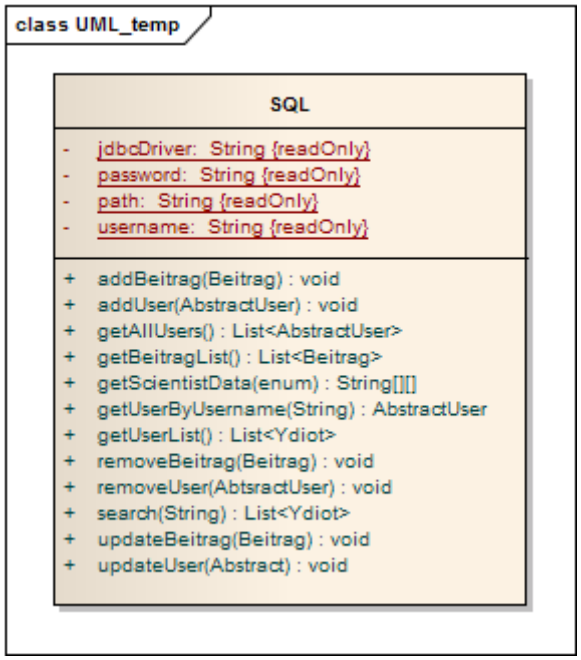
Die Klasse SQL implementiert das Interface DAO und verfügt über folgende konstanten Attribute:

- path: String Gibt an wo die SQL Datei gespeichert ist
- password: String
- jdbcDriver: String JDBC Driver Name
- username: String

Die vom Interface DAO vorgegebenen Methoden sind folgende:

- addUser(AbstractUser) : void
- getUserList(): List<Ydiot> Gibt eine Liste der in der SQL Datei gespeicherten User zurück
- getAllUsers(): List<AbstractUser> Gibt eine Liste mit allen Usern zurück
- removeUser(AbstractUser): void Löscht einen Usereintrag permanent von der SQL Datei. Darf nur von Administratoren ausgeführt werden
- getScientistData(enum): String[ ][ ] Gibt für Forscher relevante Informationen in Form von einer Tabelle.
- getUserByUsername(String):AbstractUser. Liefert den gewünschten AbstractUser aus der SQL Datei zurück, falls vorhanden.

- search(String): List<Ydiot> — Durchsucht die SQL Datei nach fullName und userName
- addBeitrag(Beitrag): void
- updateUser(AbstractUser): void — Überschreibt einen AbstractUser mit geänderten Inhalten.
- updateBeitrag(Beitrag): void — Überschreibt einen Beitrag.
- removeBeitrag(Beitrag): void — Löscht einen Beitrag permanent von der SQL Datei. Darf nur von Moderatoren ausgeführt werden.



1.1.6 Beitrag

Beitrag ist die Klassenrepräsentation eines Beitrags.

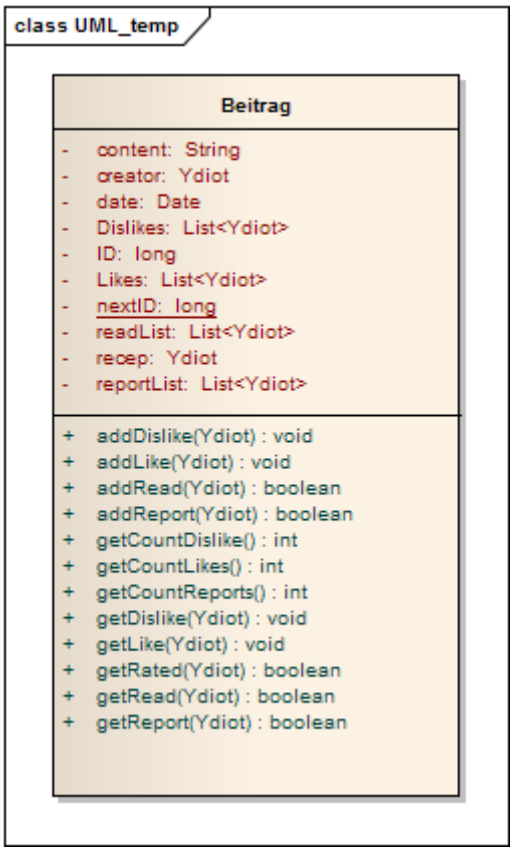
private Attribute

- nextID: static Variable, die definiert welche ID das nächste Objekt bekommen wird.
- ID: ID des Objektes in der Datenbank, für einfachere Identifikation.
- content (String): Maximal 1000 Zeichen String Freitext von User beliebig einzugeben.
- creator (Ydiot): Objekt des Benutzers der den Beitrag erstellt hat.
- date (Date): Zeitpunkt der Erstellung des Beitrags.
- Likes (List<Ydiot>): Speichert wer schon ein Like an diesem Beitrag gemacht hat.
- Dislikes (List<Ydiot>): Speichert die Ydioten, die Dislikes gemacht haben.
- reportList (List<Ydiot>): Speichert Liste aller Ydioten, welche den Beitrag schon gemeldet haben.
- readList (List<Ydiot>): Speichert welche Ydioten den Beitrag schon mindestens einmal geladen haben.

Zu beachtende Methoden:

- addDislike (Ydiot): Übergibt den aktuell eingeloggten Benutzer. Überprüft vor dem Hinzufügen ob Benutzer schon einen Dislike oder Like hinzugefügt hat.
- addLike (Ydiot): siehe addDislike(Ydiot)
- addReport (Ydiot): Übergibt aktuellen Benutzer. Überprüft vor dem Hinzufügen ob Benutzer schon einen Report abgegeben hat.

- addRead (Ydiot): Sobald ein Beitrag aus der Datenbank geladen wird überprüft er ob der aktuelle Benutzer den Beitrag schon gesehen hat. Wenn nicht wird er in der Liste hinzugefügt.
- getCountX (): gibt die Größe der jeweiligen Liste zurück.
- getX (Ydiot): Gibt zurück ob das übergebene Objekt schon in der jeweiligen Liste enthalten ist oder nicht.



1.1 Wichtige Designentscheidungen

getScientistData(enum): wir haben uns wegen der Performanz des Servers und der Internetseite für diese Variante entschieden. Alternative wäre gewesen alle möglichen Forscherdaten auf einmal auszugeben.
→ sehr viel Speicherverbrauch auf dem Java Server, hohe Auslastung der SQL Datenbank und aus der Menge der zu verarbeitenden Daten resultierende lange Ladezeit beim ersten Mal aufrufen der Webseite als Forscher.

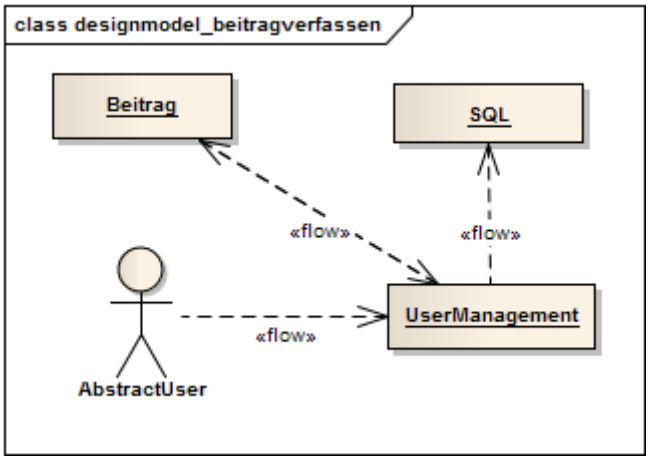
Pinnwand nicht als eigene Klasse: Ist nicht notwendig. Identifikation von Beiträgen, die auf einer bestimmten Pinnwand geladen werden kann auch einfach durch die Speicherung des Ydioten gemacht werden, an die ein Beitrag gerichtet wird.

Im Beitrag werden Likes/Dislikes nicht als Zahl sondern als Listen von Ydioten gespeichert. Dadurch haben wir einen Überblick darüber, wer diese Aktionen ausgeführt hat. Ausserdem gibt es Listen über die Ydioten, die den Beitrag gelesen und und gemeldet haben.

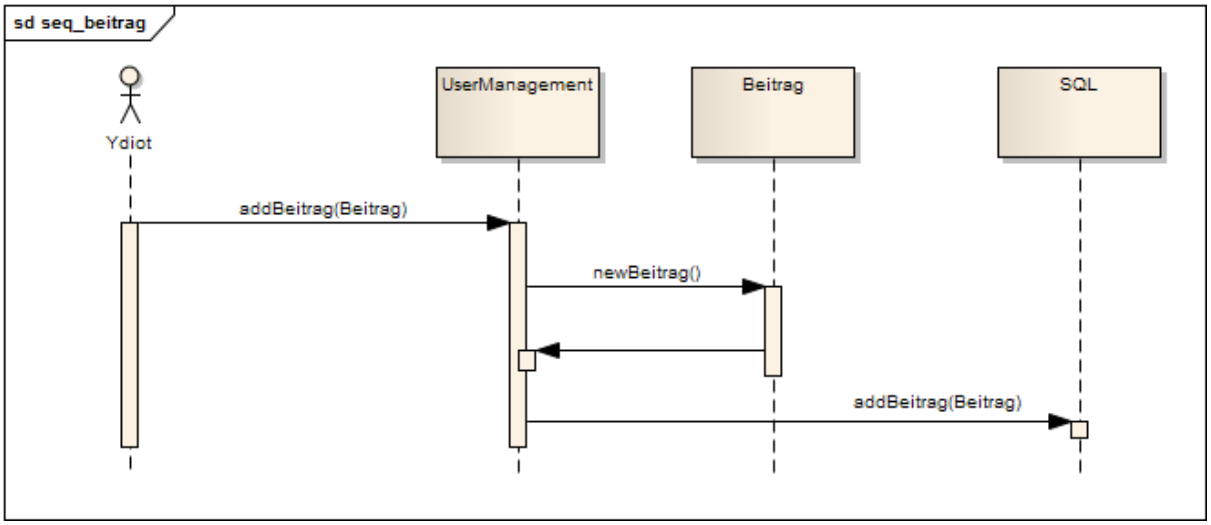
2 Use Case Realization Design

Modellieren sie, wie die zentralen Use Cases ihres Systems mittels Designklassen realisiert werden. Modellieren sie dabei die wesentlichen statischen und dynamischen Aspekte jedes Use Cases mit geeigneten Klassendiagrammen und Sequenzdiagrammen.

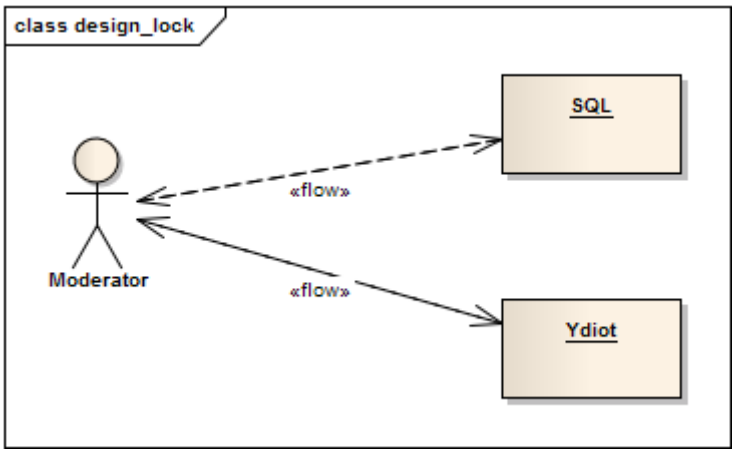
2.1 UseCase: Beitrag verfassen



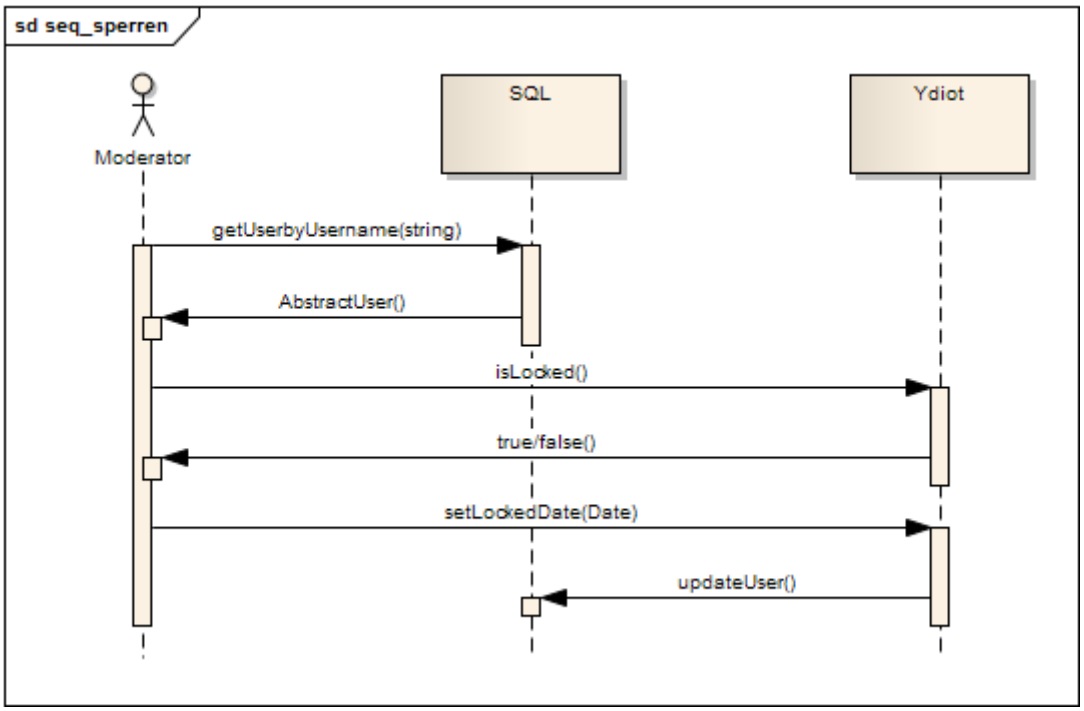
Das Verfassen eines Beitrages erfolgt über die Methode addBeitrag(). Diese Methode ist in Usermanagement implementiert und nimmt ein neuen Beitrag entgegen. Danach wird der Beitrag in der SQL Datenbank abgespeichert.



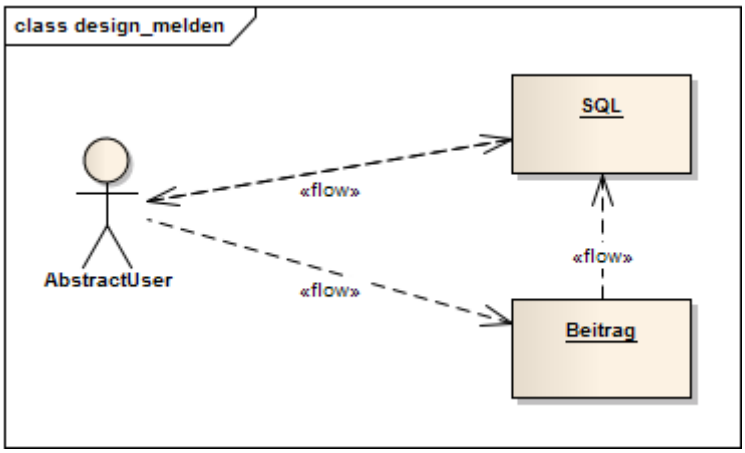
2.2 UseCase: Ydiot sperren



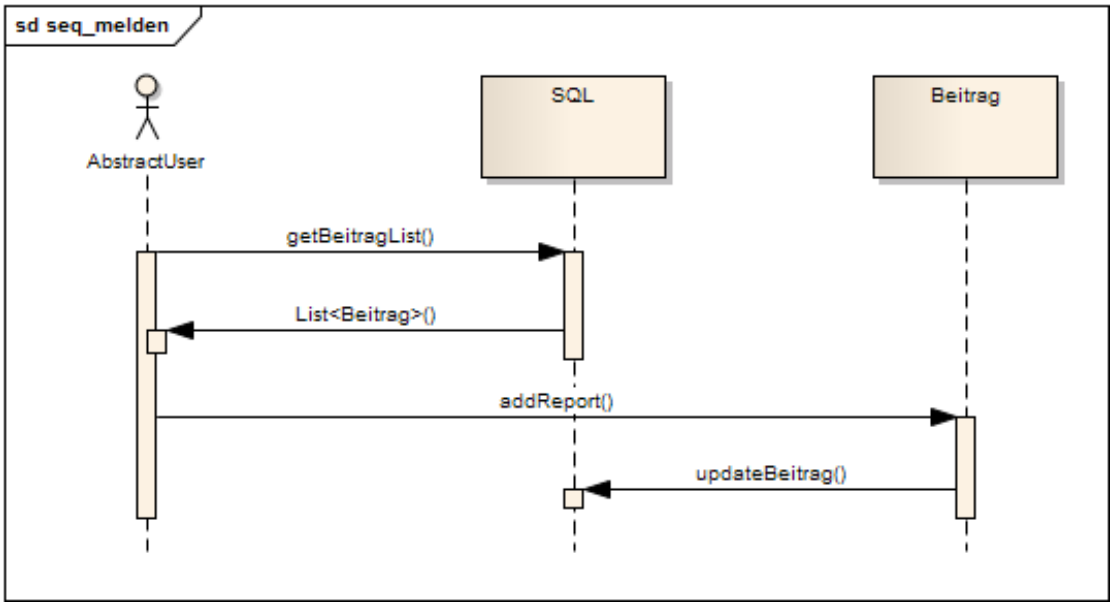
Moderatoren können Ydioten sperren. Dazu müssen sie für einen Ydioten, der in der Datenbank existiert und noch nicht gelocked ist, die Variable lockedUntil verändern. Die Suche nach dem User geschieht über die Datenbank. Danach wird über die Methode isLocked() überprüfft ob der Benutzer schon gesperrt ist. Ist er das nicht kann der Moderator über die Methode setLockedDate(Date) einen Zeitpunkt festlegen, bis zu dem der Ydiot gesperrt sein soll.



2.3 UseCase: Beitrag Melden



Um einen Beitrag zu melden, muss zuerst der entsprechende Beitrag aus der Datenbank geladen werden. Danach kann mittels der ID der gewünschte Beitrag gemeldet werden. Das erfolgt über die Methode addReport(). Dadurch wird der Benutzer, welcher diese Methode aufgerufen hat in einer Liste gespeichert. Diese Liste wird in der Klasse Beitrag gespeichert und dient dazu einen Überblick über die Ydioten zu haben, die diesen Beitrag schon gemeldet haben.



3 Übersichtsklassendiagramm

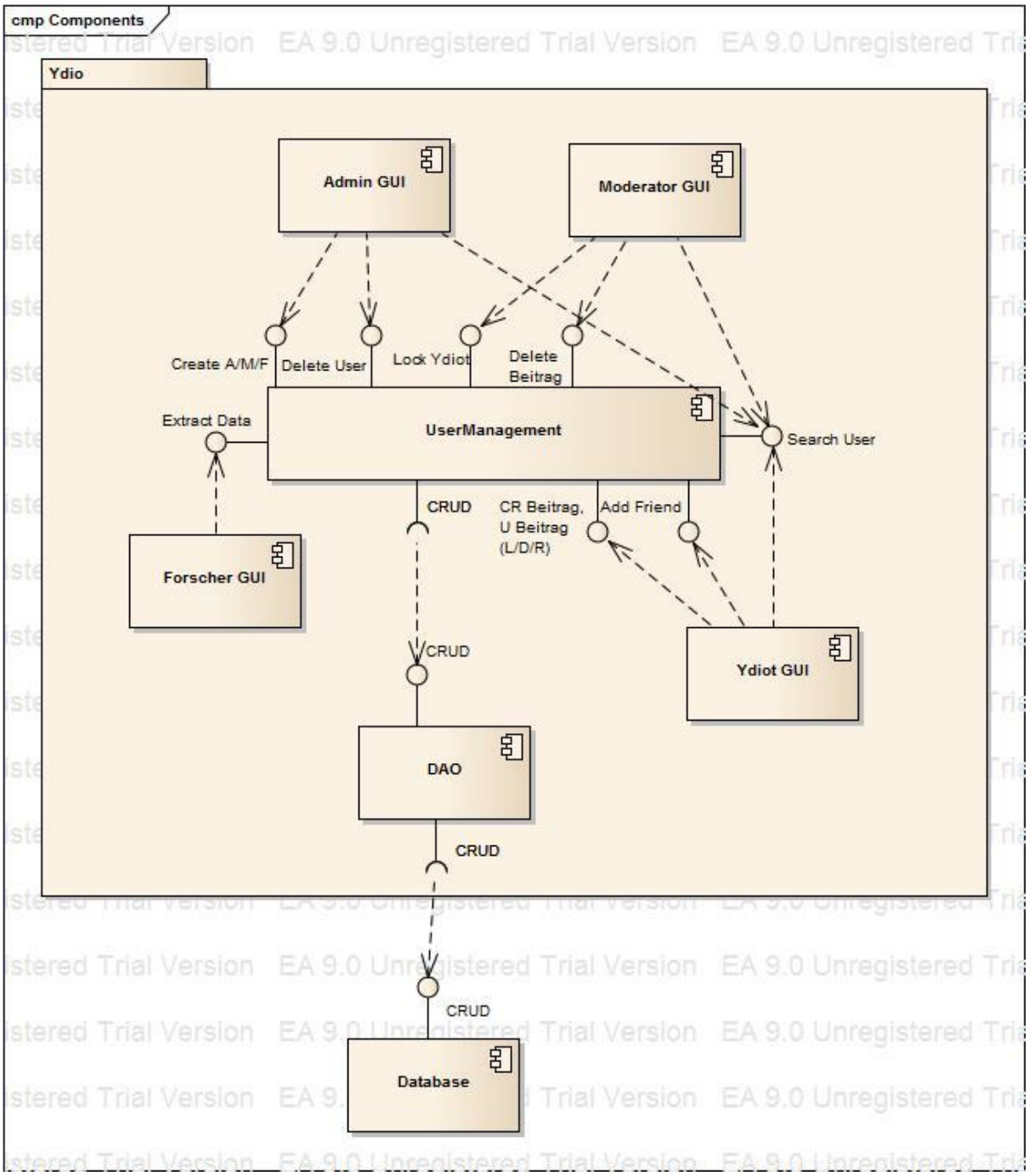
Erstellen sie ein Klassendiagram das *alle* wesentlichen Klassen ihres Systems und deren Beziehungen möglichst übersichtlich darstellt.

4 Architekturbeschreibung

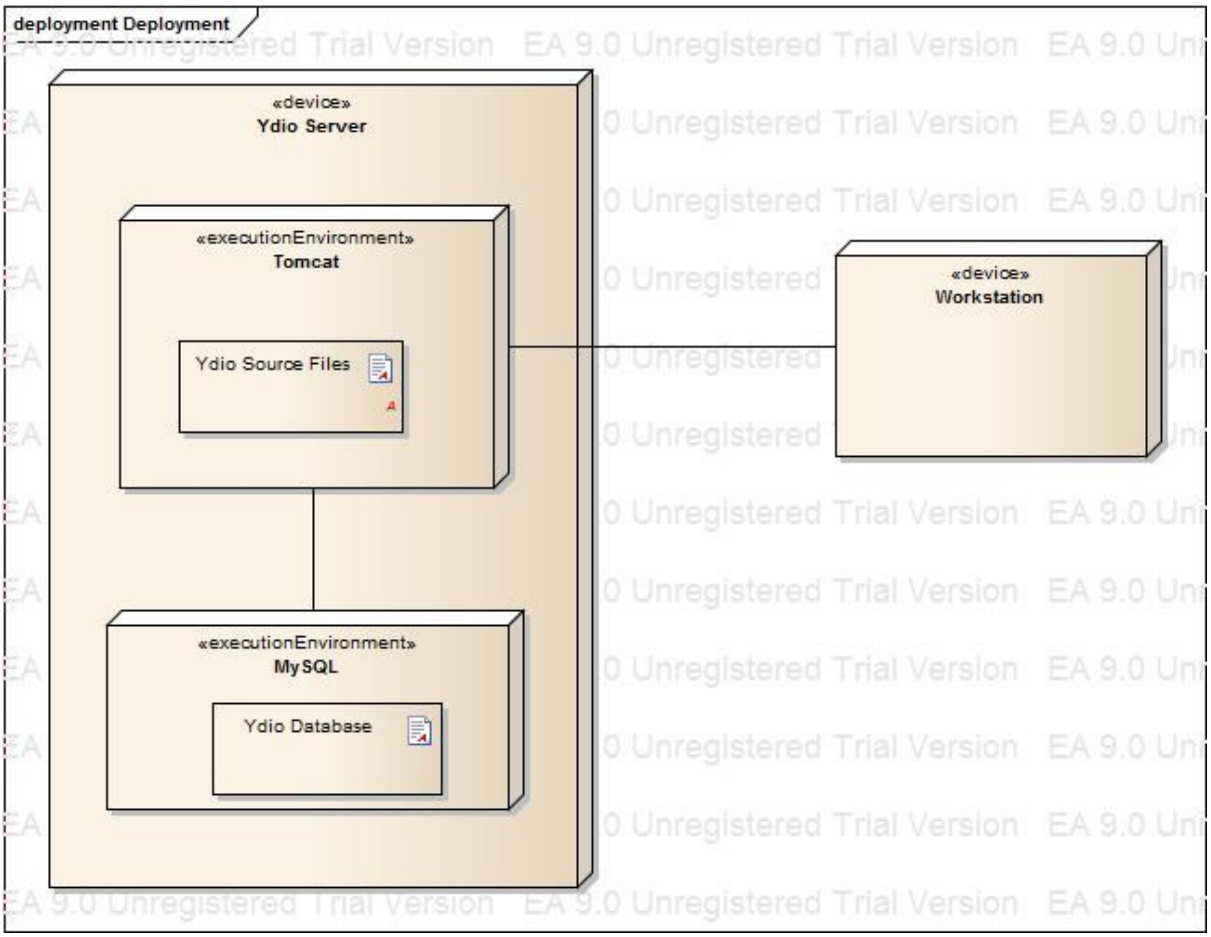
Komponentendiagramm:

Zeigt den Aufbau des Systems Ydio. Hier ist gut erkennbar, dass Ydio auf den einzelnen Userinterfaces der Benutzergruppen basiert und dabei auf UserManagement als zentrale Verwaltungskomponente basiert. Dieses Verwendet den standardisierten Datenbankzugang DAO.

Verwendete Abkürzungen:
CRUD → Create, Read, Update, Delete
L/D/R → Like/Dislike/Report
A/M/F → Admin/Moderator/Forscher



Das Deployment Diagramm zeigt die Verteilung des Programms über mehrere Geräte und Umgebungen. Da es sich um ein sehr einfach strukturiertes System handelt sind nur zwei Umgebungen auf zwei Geräten beteiligt.


deployment Deployment
«device»
Ydio Server
«executionEnvironment»
Tomcat
Ydio Source Files
«executionEnvironment»
MySQL
Ydio Database
«device»
Workstation