Software Engineering Übung

Gruppe: 6
Übungsleiter: Vasko

Designmodell v.2.0

Projekttitel: OldFace

Projekthomepage: http://oldface.omaha17.at/

Gruppenmitglieder:

MatNr:	Nachname:	Vorname:	e-mail:
Balaz	Melanie	1507236	a1507236@unet.univie.ac.at
Berg	Stefan	1425065	a1425065@unet.univie.ac.at
Haag	Valentin	1425622	a1425622@unet.univie.ac.at
Kramml	Hannes	1503268	a1503268@unet.univie.ac.at

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

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

1 Klassendesign

Relevante Klassen für für Iteration 1:

- **User:** User ist die abstrakte Klasse für alle für alle verschiedenen Arten von Usern des sozialen Netzwerkes. In Iteration 1 wird nur der Senior implementiert werden. Die Klasse User hat die privaten Attribute username (String) und password (String) die bei der Erstellung festgelegt werden. Zusätzlich werden als als Datentyp Calendar das Erstellungsdatum (creation_date) und der letzte Login (last_login_date) des Users gespeichert. Er hat die öffentliche Methode authenticateUser (username: String, password : String) welche zum authentifizieren des Users beim Login mittels Username und Passwort verwendet wird.

- **Senior**: Der Senior erweitert die Klasse User und ist als der Hauptnutzer des sozialen Netzwerkes gedacht. In Iteration 1 hat er die Attribute displayname (String) und birth_date (Calendar). In Iteration 1 hat die Klasse die Methoden getWall(): Wall welche die Pinnwand eines Users liefert und getDashboard(), welche die für einen gewissen User relevanten Messages aufruft und als Dashboard wiedergibt.

- **Message**: Die Klasse Message stellt die abstrakte Methode für Post und Comment dar. Sie hat die Attribute creation_date (Calendar) und text (String). In Iteration 1 besitzt sie die Methode deleteMessage(): void zum löschen von Messages.

- **Post**: Die Klasse Post erweitert die Klasse Message und hat die zusätzlichen Methoden addComment(): void zum Erzeugen eines Kommentars welches einem spezifischen Post zugeordnet sein muss und die Methode getComments() welche die die zu einem Post gehörigen Comments angibt. Ein Post muss einem Wall zugeordnet sein.

- **Comment**: Die Klasse Comment erweitert die Klasse Message. Sie hat keine zusätzlichen Attribute und Methoden. Jedes Comment muss einem Post zugeordnet sein.

- **Wall**: Die Klasse Wall stellt die Pinnwand eines Seniors dar. Jedem Senioren ist genau eine Pinnwand zugeordnet. Sie hat die Methoden addPost(text: String) : void, welche es einem Senioren erlaubt eine Post auf einer Pinnwand zu erstellen und die Methode getPosts() welche die zu einer Pinnwand gehörigen Posts aufruft.

Relevante Klassen zusätzlich für Iteration 2:
Logikklassen

- **Message Manager:** Die MessageManager Klasse ist für die Verwaltung jeder Art von Nachrichten zuständig, die hinzugefügt, gelöscht oder geliket werden.

- **Profile Manager:** Diese Klasse ist für die Verwaltung von Profilen zuständig, über sie können Profile geblockt und gefunden, sowie gefolgt und die Follower geholt werden.

- **Visualization Manager**: Übernimmt alle visuellen Einheiten sowie das Dashboard und die Wall, sowie die Statistiken für die Statistikseite, und die letzten Posts für die Admin Seite.

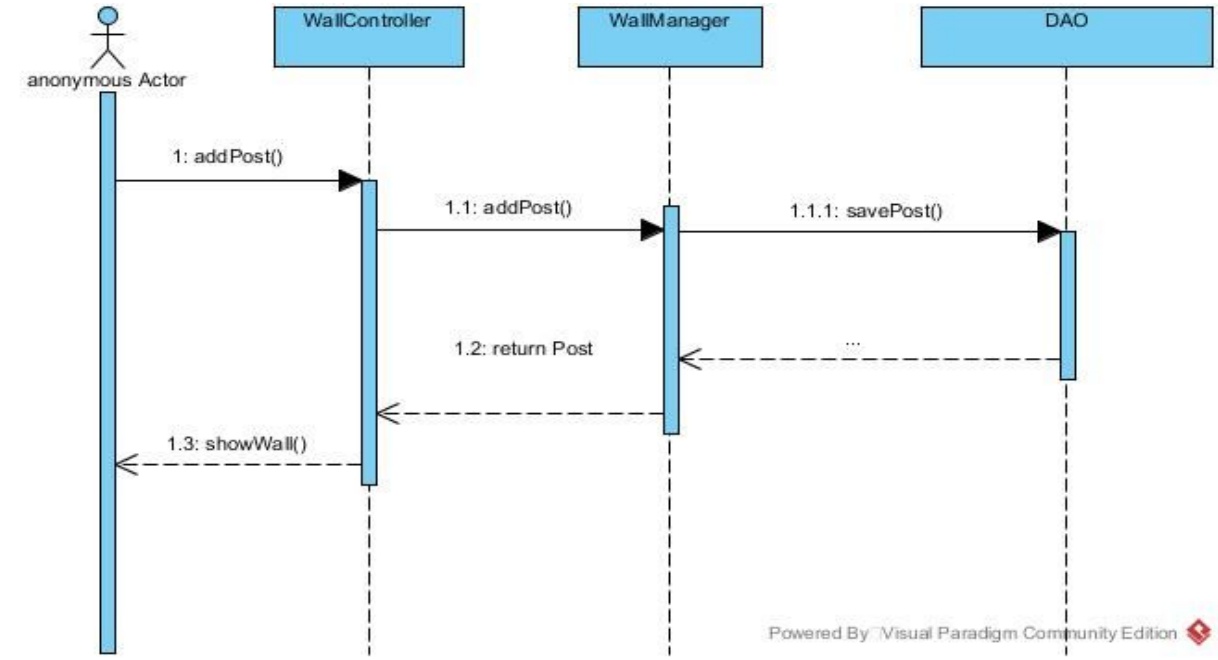
- **Authentication Manager:** Diese Klasse ist für Registrierung und Login zuständig.

- **Statistic:** Die Statistic Klasse fasst die verschiedenen möglichen Statistiken zusammen und ihre Berechnungsmethoden

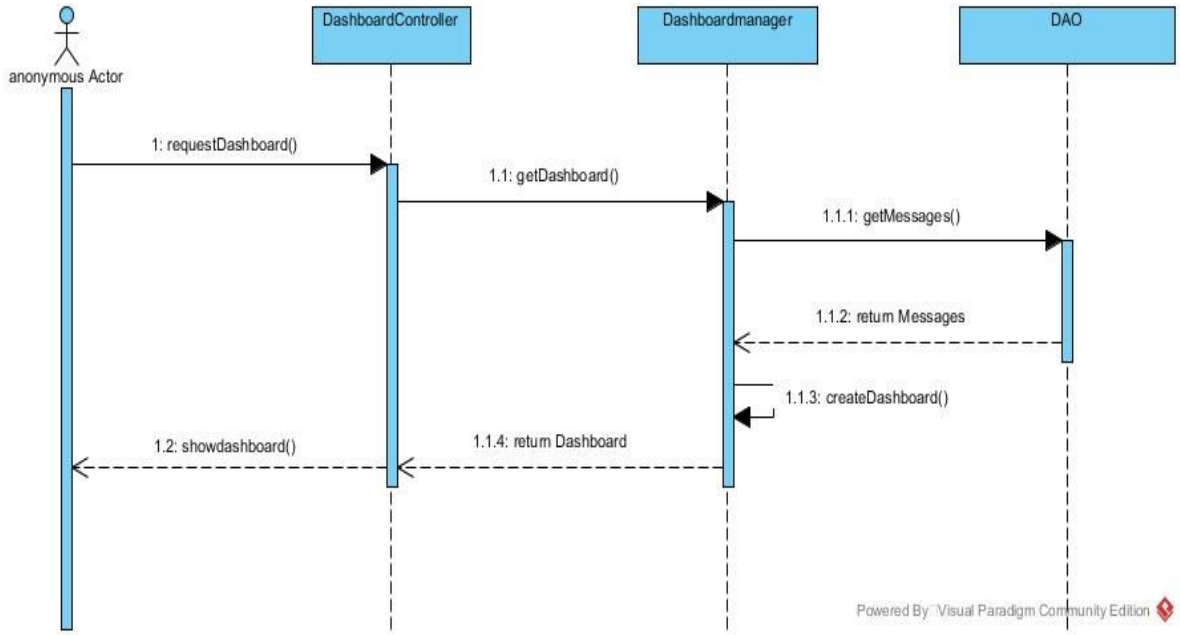
- Follower: Enthält all die Seniors denen ein Senior folgt

2 Use Case Realization Design

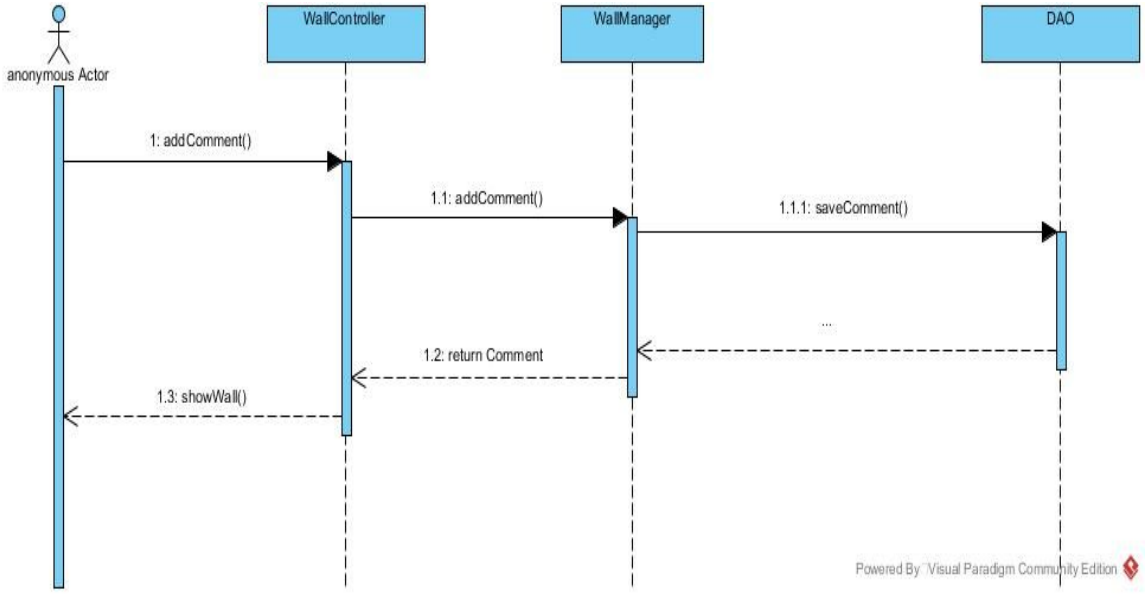
Beitrag posten



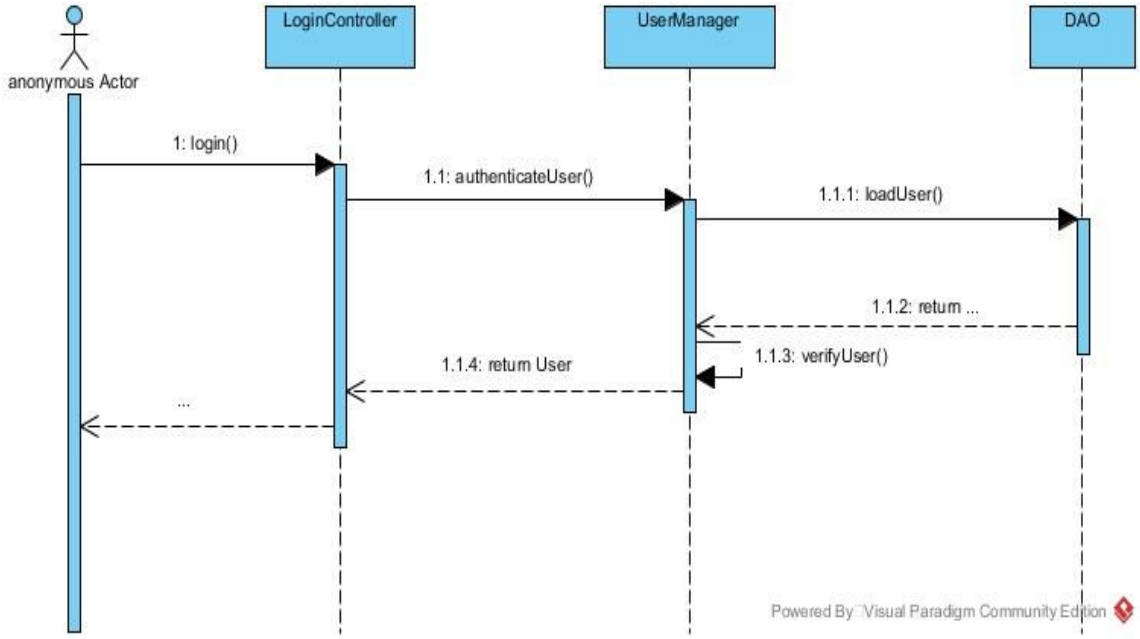
Dashboard anzeigen



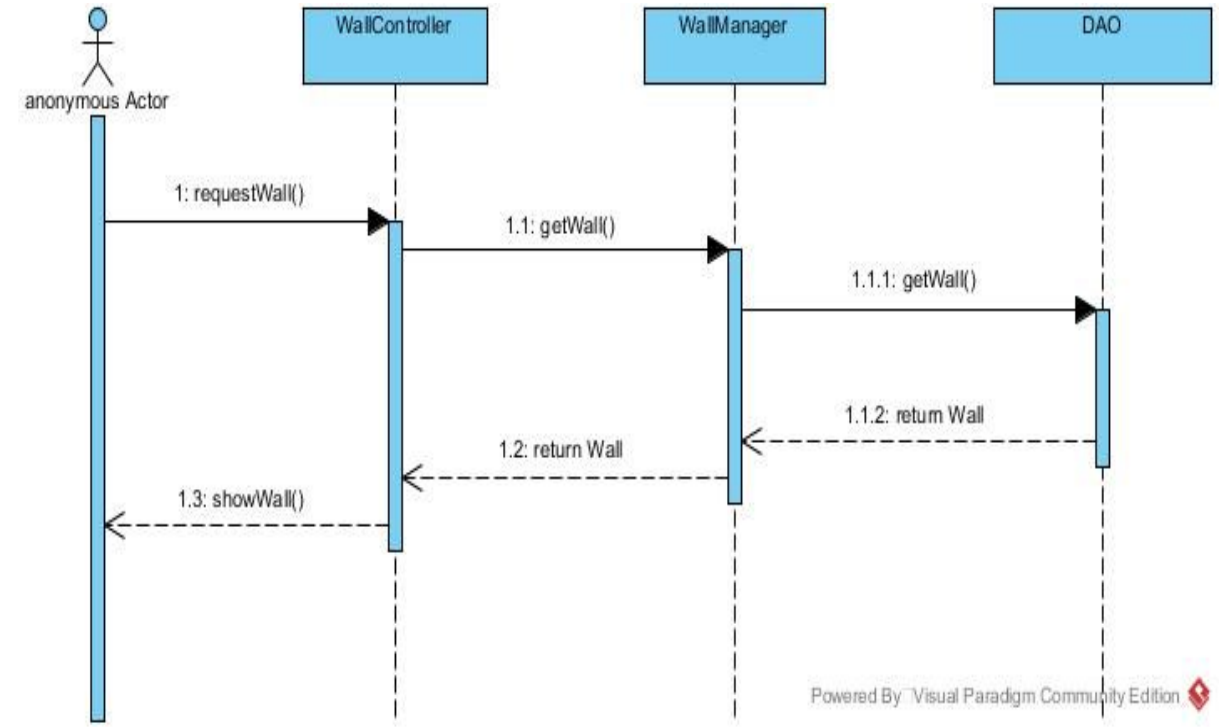
Kommentar erstellen



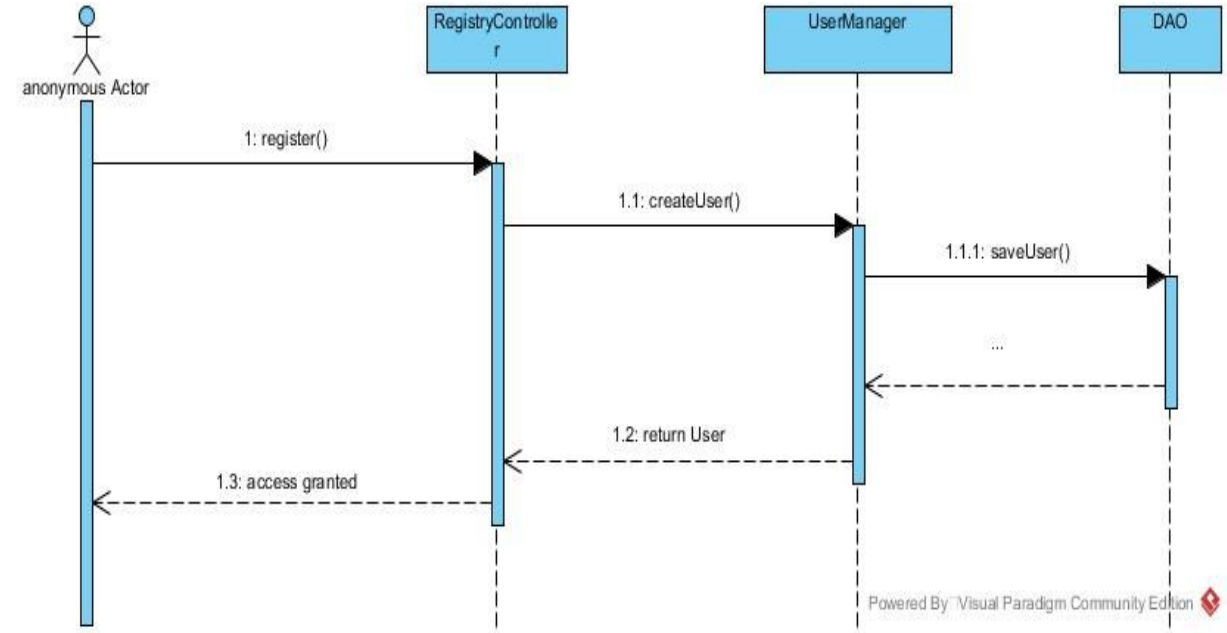
Login


anonymous Actor
LoginController
UserManager
DAO
1: login()
1.1: authenticateUser()
1.1.1: loadUser()
1.1.2: return ...
1.1.3: verifyUser()
1.1.4: return User
...
Powered By Visual Paradigm Community Edition

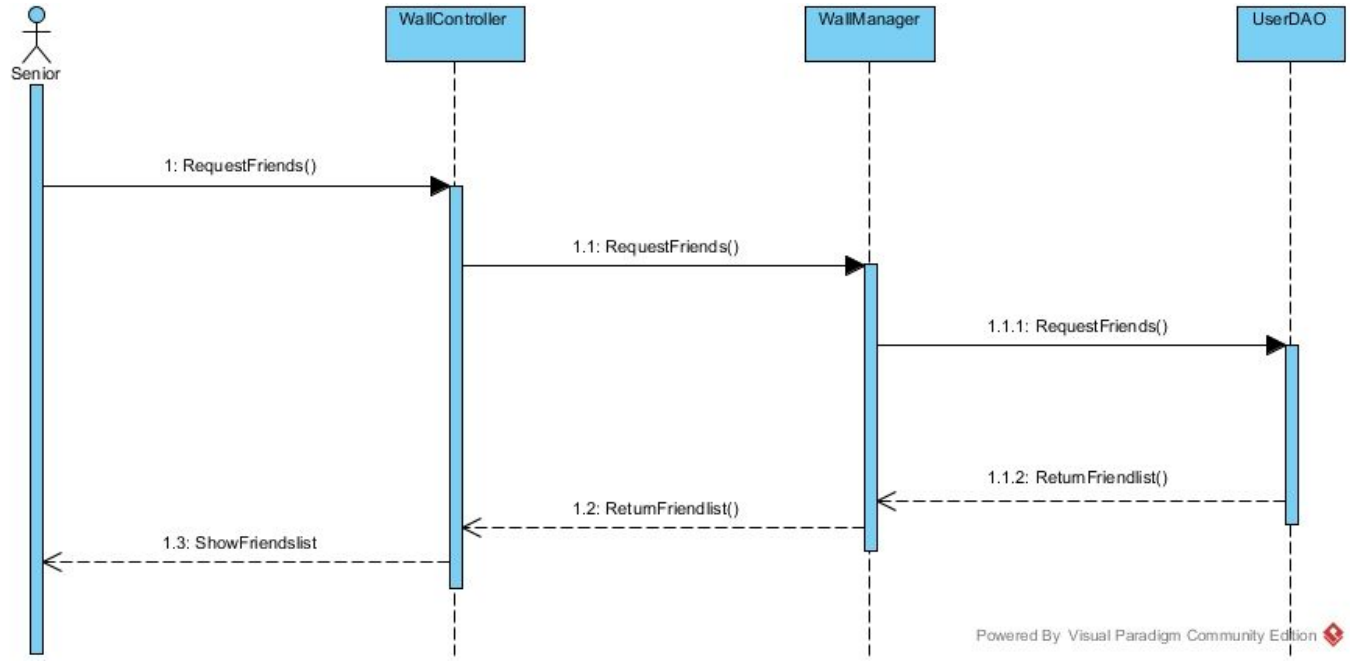
Pinnwand aufrufen



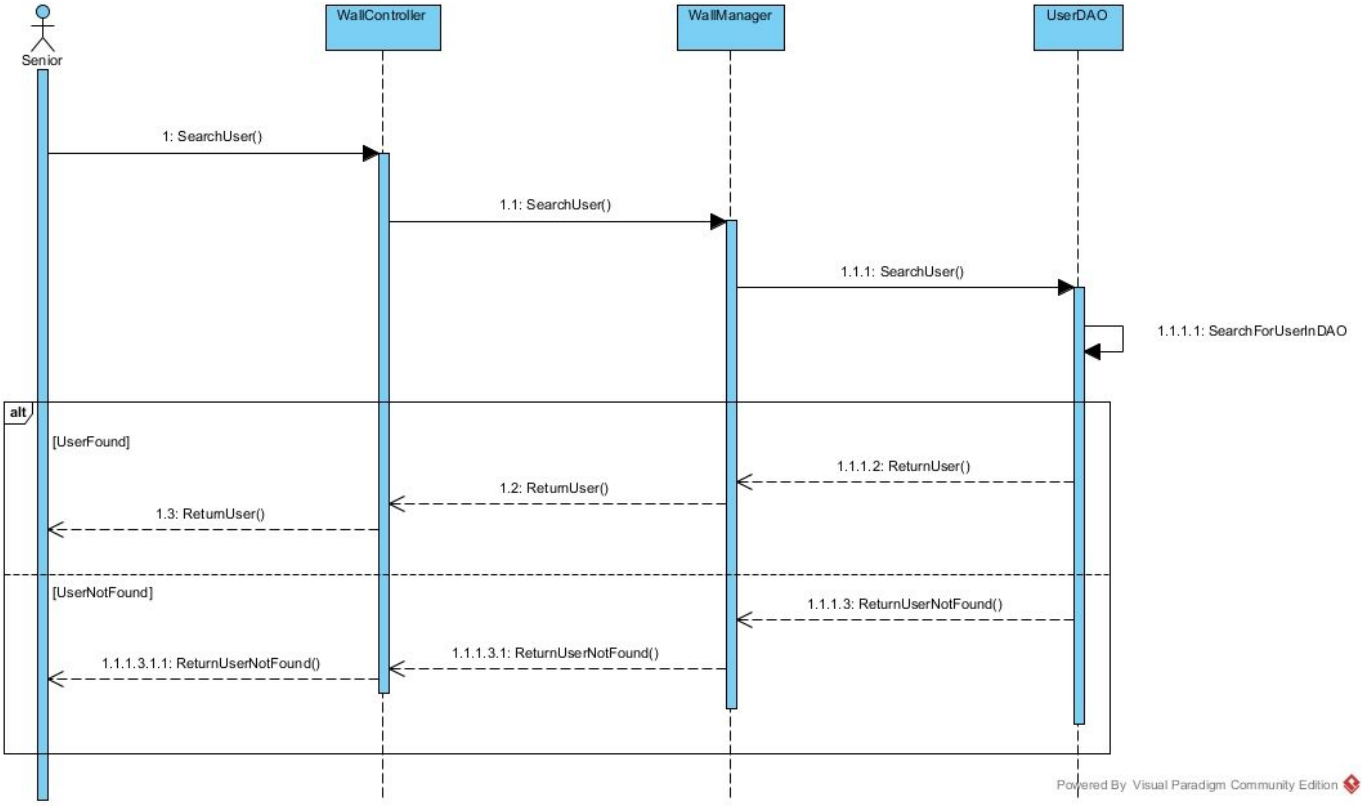
Registration



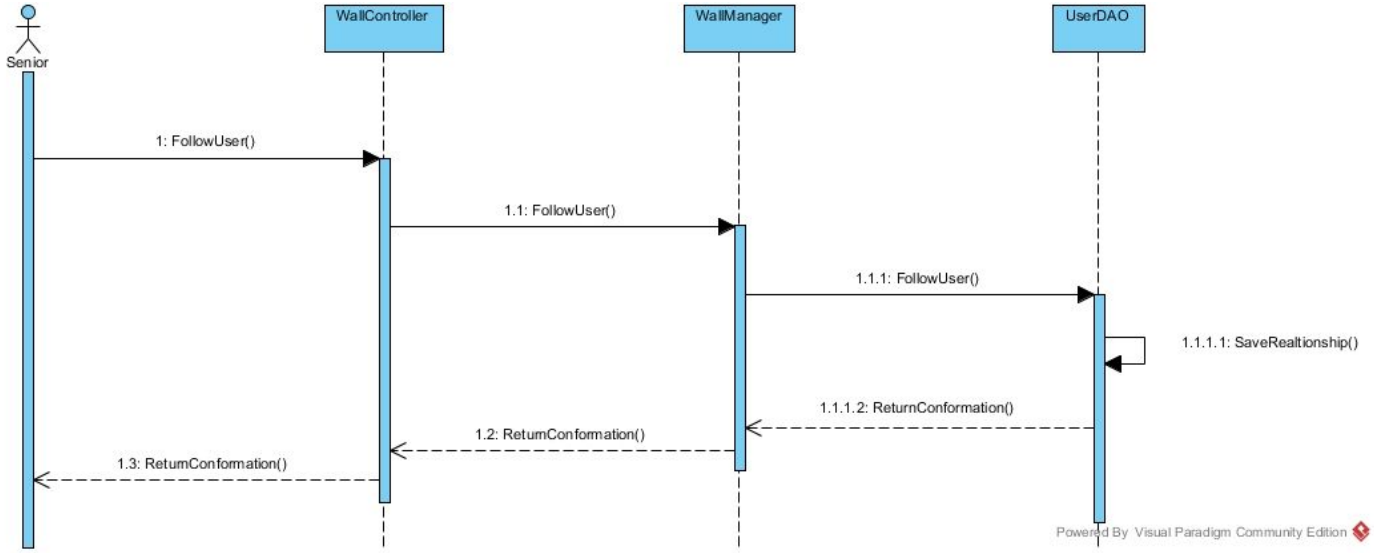
Freunde anzeigen



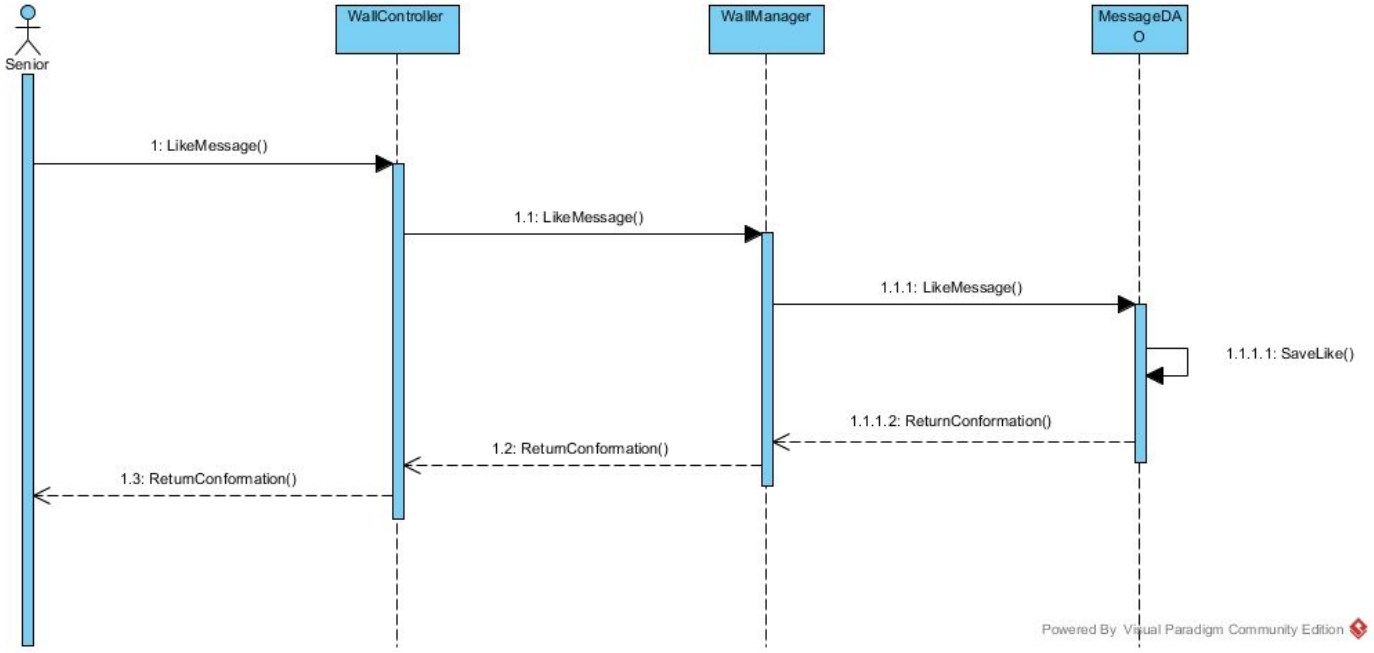
User suchen



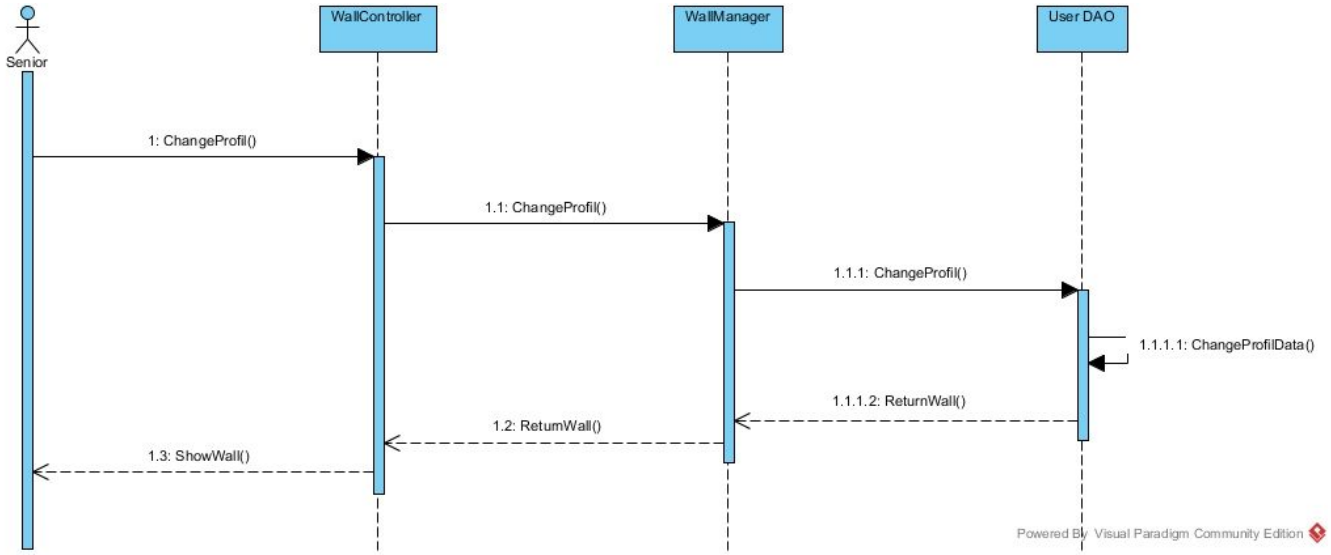
Person folgen



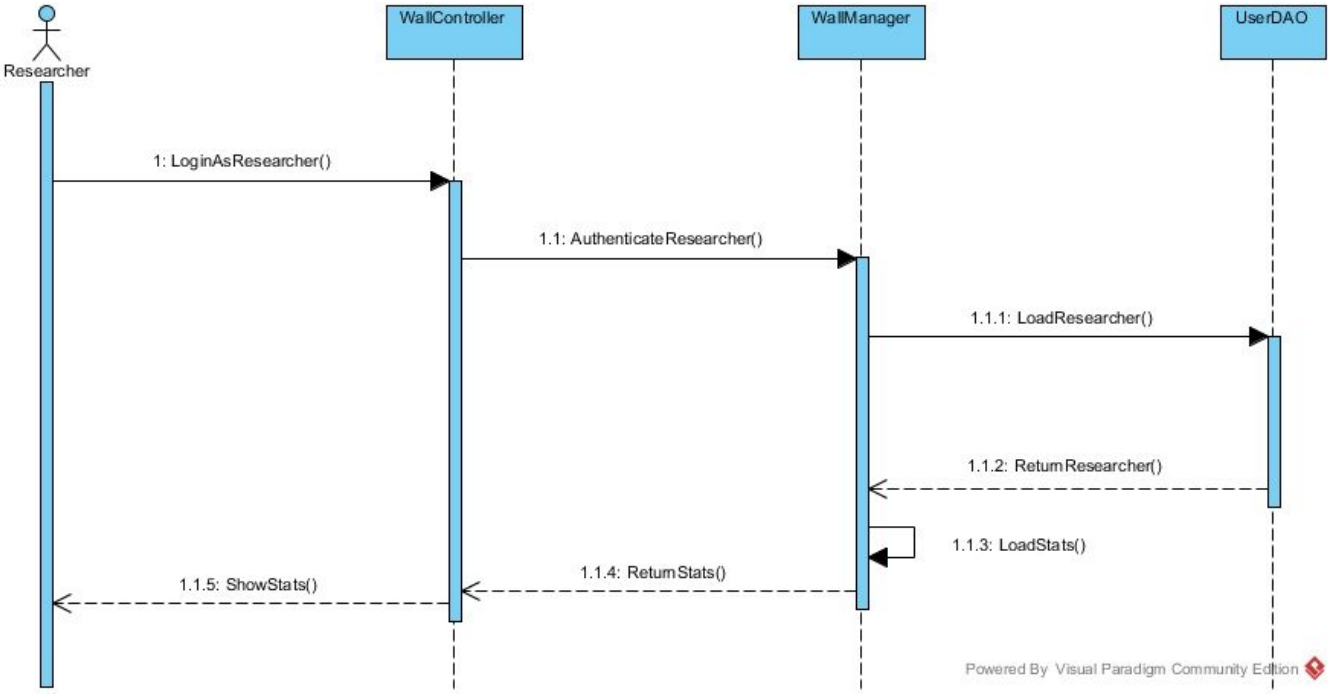
Post/Kommentar liken



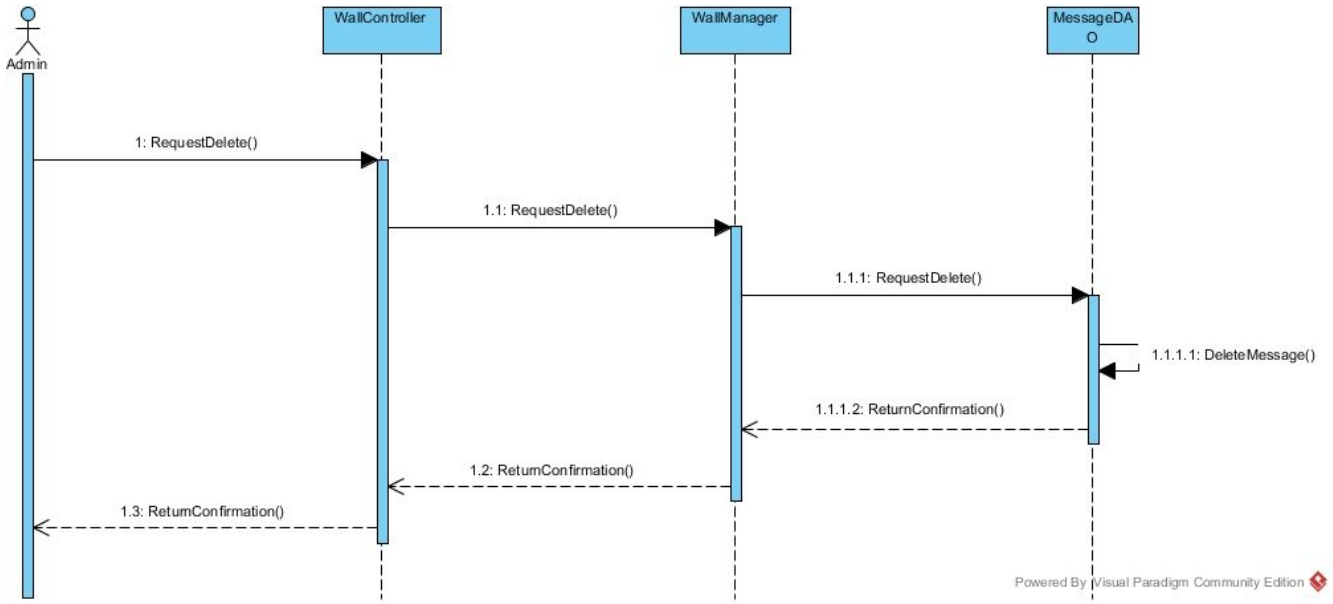
Profil bearbeiten



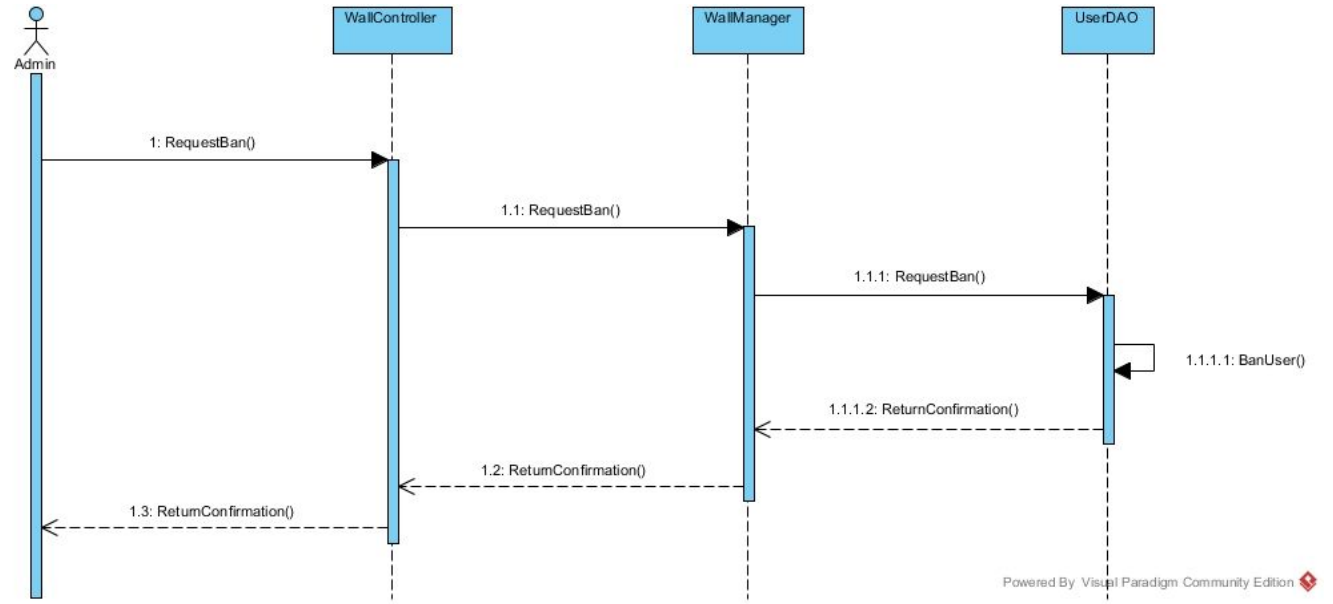
Statistik anzeigen



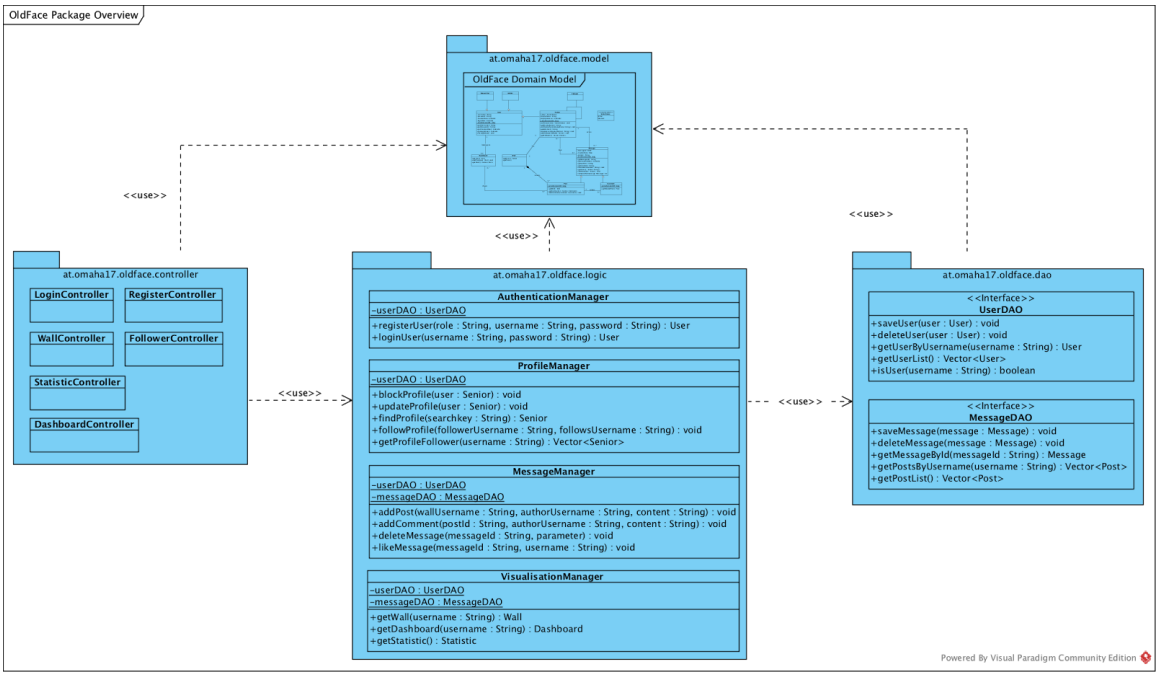
Post/Kommentar löschen



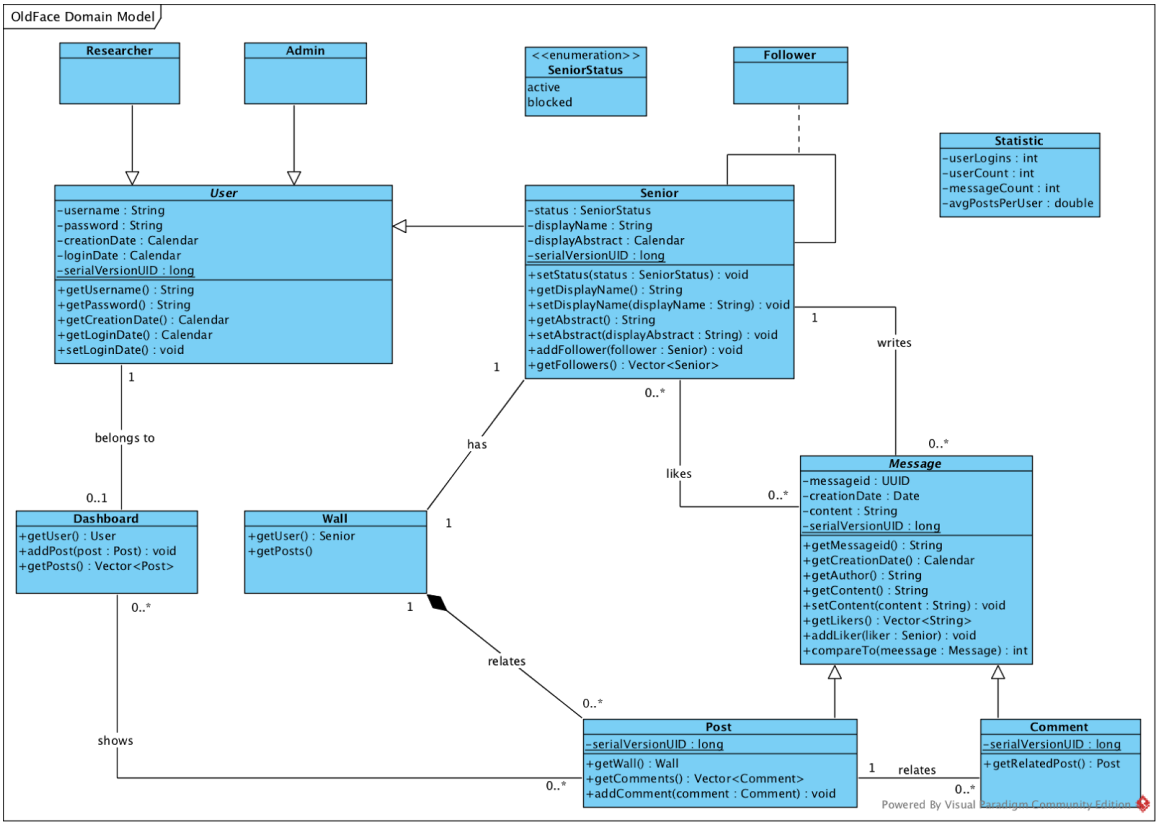
User sperren



3 Übersichtsklassendiagramm

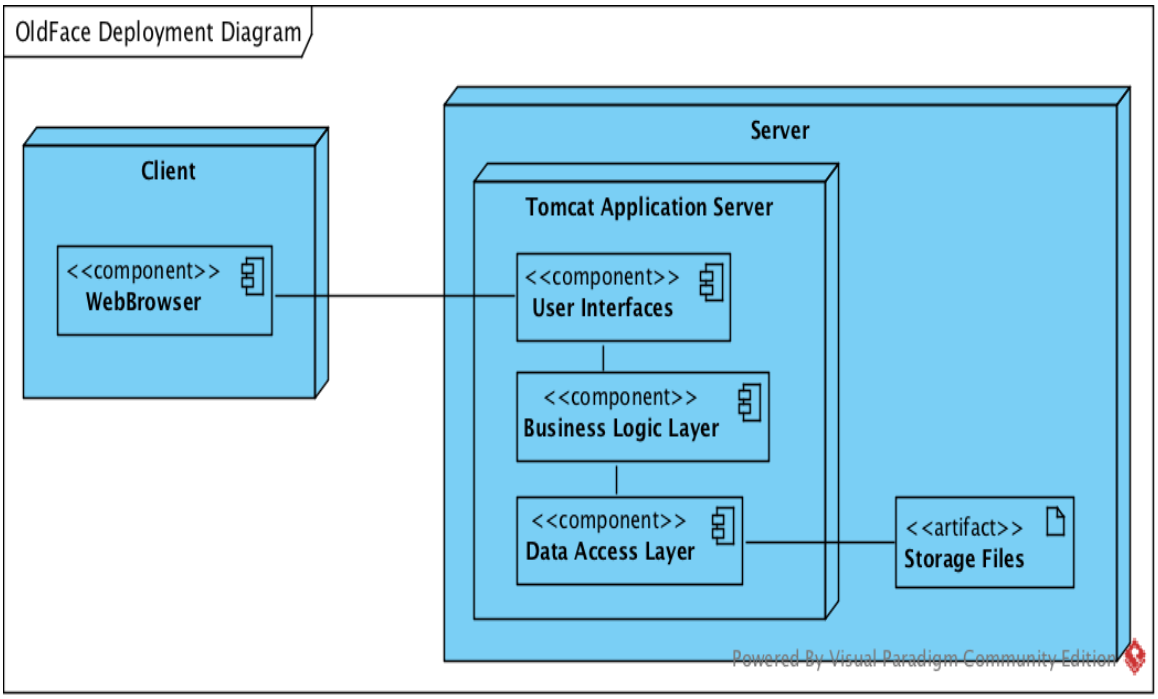


Das obige Diagramm zeigt die strukturierten Packages und deren Zusammenspiel mit weiteren Schichten/Packages. Als Designpattern fungierte das MVC Modell (Model-View-Control). Der View Teil ist hier nicht ersichtlich, dieser befindet sich in getrennten .twig Files (Umsetzung mittels TWIG Template Engine). Wie in der Grafik zu sehen ist, befindet sich die Logik/Funktionalitäten in einer separaten Business Logik Schicht mit exklusivem Zugriff auf die Datenobjekte (DAOs).
Das folgende Diagramm beschreibt das Domain Model von OldFace. Also alle für das Social Network notwendigen Objekte und deren Beziehungen zueinander (ohne Business Logik Komponenten). Aufgrund der notwendigen Datentrennung für die Java-Serialisierung werden aber manche Beziehungen nur indirekt gesetzt.



4 Architekturbeschreibung

Deploymentdiagramm:



Komponentendiagramm: