UE Software Engineering 050052 – Gruppe 10
WS 2015/16

LV-Leiter: Hans Moritsch

Designmodell

Projektname: Blue Couch - Das soziale Netzwerk

Projektteam:

Nachname	Vorname	Matrikelnummer	E-Mail-Adresse
Gazar	Mohamed	a0928951	a0928951@unet.univie.ac.at
Kolhaupt	Raphael	a1407523	a1407523@unet.univie.ac.at
Misurec	Patrik	a1325267	a1325267@unet.univie.ac.at
Pfneisl	Christian	a9525708	a9525708@unet.univie.ac.at

CEWebS-Teamseite: Team 6: SN2 - Blue Couch

Datum: 25. Nov.2015

Inhalt:

1 Klassendesign

Beginnend folgt eine Beschreibung der wichtigsten Klassen sowie ihre Methoden und Attribute.

1.1. Klasse Anwender (abstract)

Basisklasse für die unterschiedlichen Anwendertypen. Subklassen sind User, Forscher, und Admin. Stellt Funktionen zur Verfügung, die jede dieser Subklassen benötigt.

Attribute

- eMailID : String
    - eine eindeutige ID jedes Anwenders (bestehend aus seiner, beim LogIn angegebenen eMail. Hiermit wird der Anwender eindeutig Identifiziert, um unter anderem zu entscheiden in welche Rolle (Admin, Forscher, User) der Anwender fällt (stellt ihm die jeweiligen Methoden zur Verfügung)
- Password : String
    - Das Passwort des Anwenders, um auf Zugriff auf seine Daten vor anderen zu schützen. Dient neben der eMailID als zusätzliche Authentifikationsmaßnahme
- Erstellt : String
    - Das Registrierungsdatum des Anwenders

Methoden

- userSuchen(Stichwort : String) : ArrayList<String>
    - Ermöglicht dem Anwender User anhand eines Strings zu suchen. Läuft im Zusammenspiel mit AnwenderManagement ab.
- beitragSuchen(Stichwort : String, user : User) : ArrayList<String>
    - Ermöglicht dem Anwender Beiträge anhand eines Strings eines bestimmten Users zu suchen. Läuft im Zusammenspiel mit AnwenderManagement ab.

1.2. Klasse Admin (extends Anwender)

Klasse die der verwaltung anderer Nutzer und deren Beiträge dient. Erweitert Basisklasse Anwender um „Admin“ Funktionen und Attribute. Erhält außerdem Benachrichtungen bezüglich von Usern gemeldeten Beiträgen mittels eigener Klassenvariable.

Attribute

- Static gemeldeteBeitraege : ArrayList<int>
    - Liste aller gemeldeten Beiträge – für jeden Admin gleich, daher static. Wenn ein Admin einen Beitrag loescht / freigibt à wird auch aus Liste gemeldeter Beitraege geloescht

Methoden

- BeitragVonPinnwandLoeschen(id : int) : void
    - Admin erhält die Möglichkeit Beitraege, welche gemeldet wurden, zu loeschen. Hierzu benötigt er die jeweilige id des Beitrags (welche im Klassenattribut gemeldeteBeitraege gespeichert ist). Sollte der Beitrag geloescht worden sein, erhaelt dieser ein Update seines Attributes geloescht mit dem jeweiligen Datum des LoeschTages.
- Usersperren(eMail : String, Begruendung : String = unbefristet) : void
    - Admin erhaelt weiters die Option, User zu sperren. Hierzu benoetigt er nur die id des Benutzers (welche er Bspw. Durch den gemeldeten Beitrag erhaelt). Er kann weiters angeben wie lange die sperre gelten solle.
- gemeldetenBeitragFreigeben(beitrag : int)
    - Sollte ein Beitrag zu unrecht gemeldet worden sein, gibt der Admin diesen wieder frei. Ähnlich zu BeitragVonPinnwandLoeschen wird der Beitrag aus dem Klassenattribut gemeldeteBeitraege entfernt, ohne diesem jedoch ein LoeschDatum einzutragen.

1.3. Klasse Forscher (extends Anwender)

Klasse Forscher, Erweiterung der Basisklasse Anwender, beinhaltet Methoden zur statistischen Auswertung des aktuellen Zustands.

Methoden

- alleAuswertungen() : int []
    - Eine Auswertung aller Statistiken erfolgt, Ergebnisse werden per int Array retourniert.
- Anzahl_Freundschaftsbeziehungen() : int
    - Eine Auswertung der Gesamtanzahl der Freundschaftsbeziehungen erfolgt. Arbeitet unter Zuhilfenahme des AnwenderManagements. Ergebnis ist die entsprechende (int) Anzahl.
- Anzahl_Freunde_Durchschnit() : int
    - Eine Auswertung der Durchschnittsanzahl der Freundschaftsbeziehungen erfolgt. Arbeitet unter Zuhilfenahme des AnwenderManagements. Ergebnis ist die entsprechende (int) Anzahl.
- Anzahl_Freunde_pro_Zeitspanne() : int
    - Eine Auswertung der Durchschnittsanzahl innerhalb einer gegebenen Zeitspanne der Freundschaftsbeziehungen erfolgt. Arbeitet unter Zuhilfenahme des AnwenderManagements. Ergebnis ist die entsprechende (int) Anzahl.
- Anzahl_Beitraege_pro_Zeitspanne() : int
    - Eine Auswertung der Durchschnittsanzahl innerhalb einer gegebenen Zeitspanne der Beiträge erfolgt. Arbeitet unter Zuhilfenahme des AnwenderManagements. Ergebnis ist die entsprechende (int) Anzahl.
- Anzahl_Beitraege_Durchschnitt() : int
    - Eine Auswertung der Durchschnittsanzahl der Beiträge erfolgt. Arbeitet unter Zuhilfenahme des AnwenderManagements. Ergebnis ist die entsprechende (int) Anzahl.
- Anzahl_aktueller_Sperren() : int
    - Eine Auswertung der aktuell gesperrten User erfolgt. Arbeitet unter Zuhilfenahme des AnwenderManagements. Ergebnis ist die entsprechende (int) Anzahl.

1.4. Klasse User (extends Anwender)

„Herzstück“ des Projektes – Erweiterung der Basisklasse Anwender – stellt eigentliche Möglichkeit zur Verfügung User, samt aller Attribute anzulegen und zu verwalten. User legt seinerseits eigene Klassen zum Ausbau an.

Attribute

- SichtbarkeitFreundesListe : ArrayList<Gruppe>
  - Eine Liste der Gruppen dessen Mitglieder die Berechtigung besitzen die entsprechende FreundesListe des Users zu sehen.
- SichtbarkeitPinnwand : ArrayList<Gruppe>
  - Eine Liste der Gruppen dessen Mitglieder die Berechtigung besitzen die entsprechende Pinnwand des Users zu sehen.
- SichtbarkeitKontaktdaten : ArrayList<Gruppe>
  - Eine Liste der Gruppen dessen Mitglieder die Berechtigung besitzen die entsprechende Kontaktdaten des Users zu sehen.
- SichtbarkeitProfilInfo : ArrayList<Gruppe>
  - Eine Liste der Gruppen dessen Mitglieder die Berechtigung besitzen die entsprechende Profilinfo des Users zu sehen.
- SichtbarkeitProfilBild : ArrayList<Gruppe>
  - Eine Liste der Gruppen dessen Mitglieder die Berechtigung besitzen das entsprechende Profilbild des Users zu sehen.
- gesperrt : boolean = false
  - Wert der anzeigt, ob ein User aktuell eine Sperre besitzt
- gesperrtBis : Calendar
  - Wert der anzeigt, wie lange ein User gesperrt ist.

[Anmerkung: 3 Kombinationen; gesperrt false, gesperrt bis Null à User nicht gesperrt, gesperrt true, gesperrt bis Calendar à User bis Datum gesperrt, gesperrt true, gesperrt bis Null à User dauerhaft gesperrt]

- SperreNR : int = 0
  - Gibt an wie oft ein User bereits gesperrt wurde
- kontaktdaten : Kontaktdaten
  - Beinhaltet das Objekt kontaktdaten vom typ Kontaktdaten, welche eindeutig dem jeweiligen User zugeordnet ist (beidseitige Absicherung)
- profilinfo : Profilinfo

- Beinhaltet das Objekt profilinfo vom typ Profilinfo, welche eindeutig dem jeweiligen User zugeordnet ist (beidseitige Absicherung)

· beitraege : ArrayList<Beitrag>

- Beinhaltet die Objekte beitraege vom typ Beitrag, welche eindeutig dem jeweiligen User zugeordnet ist (beidseitige Absicherung)

· freunde : ArrayList<Freund>

- Beinhaltet die Objekte freunde vom typ Freund, welche eindeutig dem jeweiligen User zugeordnet ist (beidseitige Absicherung)

· nutzerbild : File

- Beinhaltet das nutzerbild des Nutzers

· gruppen : ArrayList<Gruppe>

- Beinhaltet die Objekte gruppen vom typ Gruppe, welche eindeutig dem jeweiligen User zugeordnet ist (beidseitige Absicherung)

· freundschaftsanfragen_ausgehend : ArrayList<String>

- Beinhaltet alle getaetigten ausgehenden Freundschaftsanfragen des Users

· freundschaftsanfragen_eingehend : ArrayList<String>

- Beinhaltet alle eingehenden Freundschaftsanfragen des Users die dieser beantworten muss

Methoden

· User(eMail : String, pw : String)

- Konstruktor der Klasse User. Anfangswerte werden automatisch gesetzt, beim Registrieren muss lediglich eine eindeutige eMailID sowie ein Passwort angegeben werden. Restliche Änderungen werden nachträglich vom User getätigt. (Dieser startet sozusagen mit einem „leeren“ Profil)

· anzahlFreunde() : int

- Methode um die Anzahl der gefundenen Freunde eines Users zu bekommen. Nimmt Instanzvariable freunde zur Hilfe.

· anzahlFreundeZeit() : int

- Methode um die Anzahl der geschlossenen Freundschaften eines Users innerhalb einer Zeitspanne zu bekommen. Nimmt Instanzvariable freunde zur Hilfe.

· freundLoeschen(name : String) : void

- Methode um eine Freundschafts bezüglich eines anderen Nutzers zu entfernen. Dieser Nutzer wird aus der Liste freunde entfernt. Erfolgt auf beidseitigem Niveau à Auch Instanzvariable des anderen Nutzers wird entfernt

· freundschaftsanfrage_versenden(name : String) : void

  - Methode um eine Freundschaftsanfrage an einen anderen Nutzer zu versenden. Dieser erhält eine Freundschaftseinladung (Speicherung der Einladung in der Instanzvariable freundschaftsanfragen_ausgehend). Dieser Nutzer wird der Instanzvariable freundschaftsanfragen_ausgehend des aktuellen „einladenden“ Nutzers hinzugefügt.

· freundschaftsanfrage_beantworten(name : String) : void

  - Methode um eine eingegangene Freundschaftsanfrage zu beantworten. Hierbei besteht entweder die Möglichkeit diese abzulehnen aka zu verwerfen (i.e. sie aus der Instanzvariable freundschaftsanfragen_eingehend zu löschen) oder sie anzunehmen, woraufhin sie zusätzlich zum löschen innerhalb der Instanzvariable freundschaftsanfragen_eingehend, auch mittels Erzeugung eines neuen Objektes vom Typ Freund und dem hinzufügen zur Instanzvariable freunde gespeichert wird.

· beitragLoeschen(beitrag : BeitragID) : void

  - Methode um einen verfassten Beitrag wieder zu loeschen. Hierbei wird die Instanzvariable geloescht des zu loeschendem Objekt mit dem aktuellen Datum versehen, um dieses zwar nichtmehr anzuzeigen (bzw. um anzuzeigen das es geloescht wurde, es jedoch für eine spätere Dokumentation weiterhin aufzubewahren.

· anzahlBeitrage() : int

  - liefert die aktuelle Anzahl der (nicht geloeschten) Beiträge zurück.

· anzahlBeitraegeZeit() : int

  - liefert die Anzahl der (nicht geloeschten) Beiträge zurück, welche innerhalb einer bestimmten Zeitspanne geloescht worden sind.

· beitragHinzufuegen(titel : String, text : String) : void

  - Erstellt ein neues Objekt vom Typ Beitrag und fuegt es der Instanzvariable beitraege hinzu. Dieses neue Objekt wird mit entsprechendem Erstelldatum, übergebenem Titel und übergebenen (Inhalts-)Text versehen.

· beitragMelden(user : String, beitrag : String) : void

  - Der User erhält die Möglichkeit einen Beitrag eines anderen Users zu melden. Hierbei gibt er den jeweiligen User samt Beitrag an. Der gemeldete Beiträg erhält einen Vermerk innerhalb seiner Instanzvariablen, sowie einen Eintrag in die Klassenvariable gemeldeteBeitraege der Klasse Admin

- gruppeErstellen(name : String) : void
    - Der User erhält die Möglichkeit ein neues Objekt vom Typ Gruppe zu erstellen, welches Standartmäßig einen Namen erhält. Weiters wird dieses erzeugte Objekt der Instanzvariable gruppen des Users angefügt.
- gruppeLoeschen(name : String) : void
    - Der User erhält die Möglichkeit ein Objekt vom Typ Gruppe zu loeschen. Kaskadierend wird die Instanzvariable der in der jeweiligen Gruppe angefügten Freunde in_gruppe auf null gesetzt. Das zu loeschende Objekt wird anschließend aus der Instanzvariable gruppen des Users entfernt.

1.5. Klasse Profilinfo

Die Klasse Profilinfo beinhaltet alle wesentlichen Attribute des Users bezüglich seines Profils. Sie ist einem User eindeutig anhand dessen emailID zugewiesen, wobei jeder User genau ein Objekt der Klasse Profilinfo „besitzt“.

Attribute

- vorname : String
    - Der Vorname des Users
- nachname : String
    - Der Nachname des Users
- geburtsdatum : Calendar
    - Das Geburtsdatum des Users
- hobbys : String
    - Die Hobbies des Users
- geschlecht : String
    - Das Geschlecht des Users
- interessen : String
    - Die Interessen des Users
- religion : String
    - Die Religion des Users
- familienstatus : String
    - Der Familienstatus des Users
- sexuelle_Orient : String

- Die sexuelle Orientierung des Users

· politische_Einst : String

- Die politische Einstellung des Users

· zusatzangaben : String

- Zusatzangaben, welche der User tätigen will

· user : String

- Der User zugehörig zum jeweiligen Objekt vom Typ Profilinfo ist hier gespeichert.

Methoden

· profilinfo_aendern(vname : String, nachname : String, bday : Calendar, hob : String, g : String, r : String, i : String, s : String, p : String, z : String) : void

- Dient der gemeinsamen Änderung der gesamten Profilinfo (ohne separate alle getX Anfragen versenden zu müssen)

1.6. Klasse Kontaktdaten

Die Klasse Kontaktdaten beinhaltet alle wesentlichen Attribute des Users bezüglich seiner Kontaktdaten. Sie ist einem User eindeutig anhand dessen emailID zugewiesen, wobei jeder User genau ein Objekt der Klasse Kontaktdaten „besitzt“.

Attribute

· adresse : String

- Die Adresse des Users

· wohnort : String

- Der Wohnort des Users

· telefonnr : String

- Die Telefonnummer des Users

· email : String

- Die eMail des Users

· user : String

- Der User zugehörig zum jeweiligen Objekt vom Typ Kontaktdaten

Methoden

- kontaktdaten_aendern(adresse : String, wOrt : String, tNr : String, eMail : String) : void
    - Dient der gemeinsamen Änderung der gesamten Kontaktdaten (ohne separate alle getX Anfragen versenden zu müssen)

1.7. Klasse Gruppe

Die Klasse Gruppe beinhaltet alle wesentlichen Attribute des Users bezüglich seiner Gruppen. Sie ist einem User eindeutig anhand dessen emailID zugewiesen, wobei jeder User mehere Objekte der Klasse Gruppe besitzen kann.

Attribute

- name : String
    - Der Name der Gruppe
- mitglieder : ArrayList<Freund>
    - Die Mitglieder der Gruppe (Referenz auf jeweiliges Gruppenobjekt)
- user : String
    - Der User zugehörig zu jeweiligen Objekten vom Typ Gruppe

Methoden

- freund_zu_gruppe(freund : Freund) : void
    - Eine Referenz vom Typ Freund zur Instanzvariable mitglieder der Gruppe hinzufügen. Das Objekt vom Typ Freund bekommt weiterhin einen Vermerk innerhalb seiner Instanzvariablen, zu welcher Gruppe es zugeordnet wurde
- freund_aus_gruppe(freund : Freund) : void
    - Eine Referenz aus der Instanzvariable mitglieder der Gruppe entfernen. Das Objekt vom Typ Freund bekommt weiterhin einen Vermerk innerhalb seiner Instanzvariablen, zu welcher Gruppe es zugeordnet wurde

1.8. Klasse Freund

Die Klasse Freund beinhaltet alle wesentlichen Attribute des Users bezüglich seiner Freunde. Sie ist einem User eindeutig anhand dessen emailID zugewiesen, wobei jeder User mehere Objekte der Klasse Freund besitzen kann.

Attribute

- beginn : Calender
    - Erstellungsdatum des Objekts vom Typ Freund

- in_gruppe : Gruppe
    - Vermerk, zu welcher Gruppe der jeweilige Freund zugeordnet wurde
- user : String
    - Der User zugehörig zu jeweiligen Objekten vom Typ Gruppe
- freunduser : String
    - Der User mit dem die Freundschaft eingegangen wurde

1.9. Klasse Beitrag

Die Klasse Beitrag beinhaltet alle wesentlichen Attribute des Users bezüglich seiner Beiträge. Sie ist einem User eindeutig anhand dessen emailID zugewiesen, wobei jeder User mehere Objekte der Klasse Beitrag besitzen kann.

Attribute

- user : String
    - Der User zugehörig zu jeweiligen Objekten vom Typ Beitrag
- titel : String
    - Titel des Beitrages
- erstellt : ArrayList<Calender> = today
    - Datum an dem der Beitrag erstellt wurde
- inhalt : String
    - Textueller Inhalt des jeweiligen Beitrages
- gemeldet : boolean
    - Wert der angibt ob jeweiliger Beitrag gemeldet wurde
- geloescht : Calendar
    - Datum an dem der Beitrag geloescht wurde (entweder vom jeweiligen User oder von einem Admin)
- static BeitragID : int
    - Die eindeutige BeitragsID des jeweiligen Beitrags. Fortlaufend inkrementierende Nummer, welche bei Erstellung eines Beitrag hochgezählt wird. Statisches Klassenattribut um so dem Admin zu erlauben einen Beitrag anhand seiner eindeutigen ID zu loeschen.

1.10. Klasse AnwenderManagement

Die Klasse AnwenderManagement regelt die interne Logik, sowie die weiterleitung der Befehle die die Servlets benötigen. Es überprüft ob Aktionen erlaubt sind, oder nicht, aka führt diese aus oder verweigert sie – dient also quasi gleichzeitig als Sicherheitssystem (zumindest von der Logik-Seite aus)

Attribute

- anwenderDAO : AnwenderDAO
    - AnwenderManagement benötigt Hilfe von AnwenderDAO, da diese für das persistente Verwalten der Daten verantwortlich ist, weshalb AnwenderManagement eine Referenz auf ein Objekt des Typs anwenderDAO erhaelt (bzw. genauer gesagt, ein Objekt vom Typ anwenderDAO erzeugt)

Methoden

- login(email : String, password : String) : String
    - Der Anwender muss sich vor dem ausführen zuerst mittels Methode LogIn identifizieren. Diese überprüft, unter Zuhilfename der Methoden des AnwenderDAO's ob der Anwender registriert ist, und um welchen Typ Anwender (g.e. User, Admin, Forscher) es sich handelt, und gibt Rückmeldung über ein erfolgreiches oder negatives Anmelden.
- registrieren(email : String, password : String) : String
    - Der User muss sich bei seinem ersten Besuch registrieren, wobei ein neues Objekt vom Typ User angelegt wird. Hierbei muss eine eindeutige eMailID (welche noch nicht durch einen anderen User belegt sein darf) sowie ein Passwort benötigt. Anschließend wird der neu angelegte User mittels Methoden des AnwenderDAO's persistent gespeichert.
- speichereAnwender(user : Anwender) : Boolean
    - Jeder neu erzeugt Anwender bzw. Jeder Anwender der ein Updater seiner Instanzvariablen bekommen hat muss erneut gespeichert werden. Die Methode ruft die entsprechende Methode des AnwenderDAO's auf.
- userSuchen(Stichwort : String) : ArrayList<String>
    - Die Methode ermöglicht es alle User zu suchen, deren Vor- bzw. Nachnamen dem eingebenen String entsprechen. Dies geschieht unter der Zuhilfenahme von Methoden des AnwenderDAO's
- beitragSuchen(Stichwort : String, user : User) : ArrayList<String>
    - Die Methode ermöglicht es alle Beiträge zu suchen, deren (Inhalts-)Text bzw. Titel dem eingegeben String entsprechen. Hierbei muss allerdings der jeweilige User ebenso übergeben werden, innerhalb dessen Beiträge gesucht werden soll. Dies geschieht unter der Zuhilfenahme von Methoden des AnwenderDAO's sowie den Methoden des jeweiligen Users.

- userSiteAnzeigen(user : String) : ArrayList<String>
    - Die Methode ermöglicht es, sich alle relevanten UserDaten zu holen, um eine entsprechende Ansicht dessen Profils zu ermöglichen. Dies geschieht einerseits unter Zuhilfenahme der Methoden des AnwenderDAO's, andererseits werden die Methoden des entsprechenden Users benuttzt, um an dessen relevante Daten zu kommen. Vorher muss jedoch selbstverständlich überprüft werden, ob der Benutzer, das recht hat, die jeweilige UserSite sehen zu dürfen.

1.11. Klasse AnwenderDAO

Die Klasse AnwenderDAO dient ausschließlich der persistenten Speicherung, sowie dem Auslesen der persistent gespeicherten Daten. Sie übernimmt keine logische Überprüfung – die muss bereits vor Aufruf vom AnwenderManagement stattgefunden haben.

Attribute

- dateipfad : String
    - Der Dateipfad, an dessen Stelle die entsprechende Datei zur speicherung (und natürlich zum auslesen) der Daten zu finden ist.

Methoden

- saveAnwenderList() : void
    - Die Klasse AnwenderDAO speichert die gesamte Anwenderliste persistent.
- loadAnwenderList() : ArrayList<Anwender>
    - Die Klasse AnwenderDAO liest die gesamt persistent gepspeicherte Anwenderliste aus, und retourniert diese.
- getAnwenderbyID(id : String) : Anwender
    - Die Klasse AnwenderDAO sucht innerhalb der persistent gespeicherten Anwenderliste nach einem bestimmten User und gibt diesen ggfs. zurück. Andernfalls wird null zurückgegeben.
- saveAnwender(user : Anwender) : boolean
    - Die Klasse AnwenderDAO fügt einen Anwender zur persistent gespeicherten Anwenderliste hinzu.
- getAnwenderlistbyString(text : String) : ArrayList<Anwender>
    - Die Klasse AnwenderDAO sucht innerhalb der persistent gespeicherten Anwenderliste, nach allen Usern die einem bestimmten String gleichen und gibt diese zurück.

2 Use Case Realization Design

2.1 UseCase 1: Kontaktinformationen verwalten

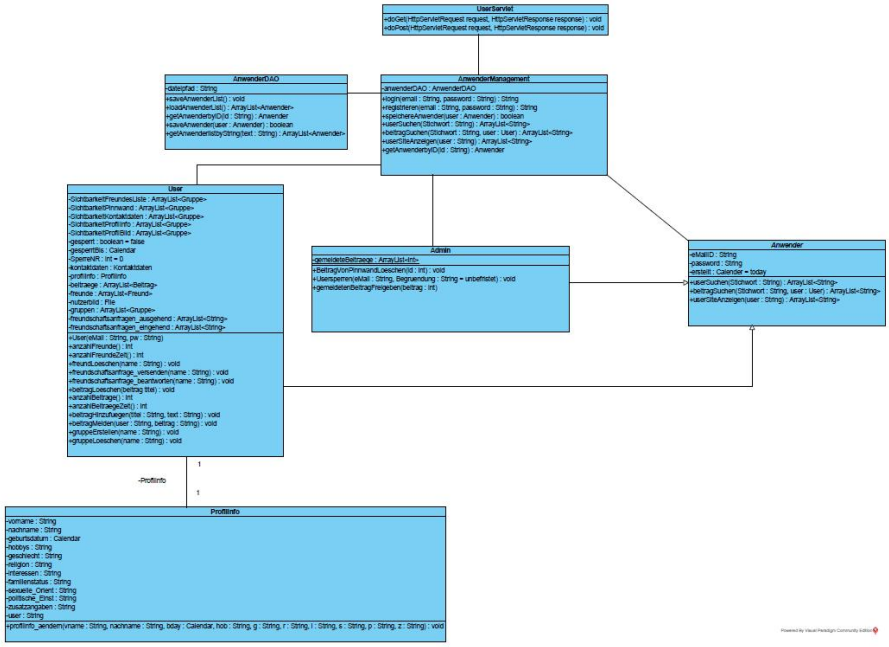
Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

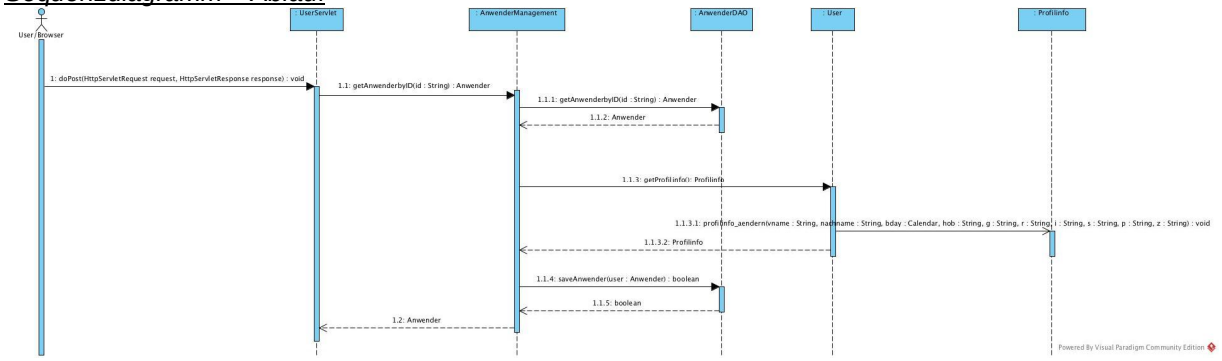
Textuelle Beschreibung – Ablauf

2.2 UseCase 2: Persönliche Information verwalten

Klassendiagramm – Ausschnitt



Sequenzdiagramm – Ablauf



Textuelle Beschreibung – Ablauf

UserServlet ruft getAnwenderbyID(Anwender) des AnwenderManagement auf. Dieser weißt selbigen Befehl an das anwenderDAO weiter. Nach erhalt des (durch Login eindeutig) als User identifizierten Benutzer, wird vom Servlet die Methode profilinfo_aendern des Objekts profilinfo des Users aufgerufen, und diesem die Daten uebergeben. Anschließend weißt das Servlet, dem Management den Befehl zu, den aktualisierten User wieder zu speichern. Dieses leitet den Befehl mittels eigener Methode speichereAnwender(Anwender) wieder dem AnwenderDAO weiter, welcher den aktualisierten User persistent mittels der Methode saveAnwender(Anwender) speichert.

2.3 UseCase 3: Benutzerbild verwalten

Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

Textuelle Beschreibung – Ablauf

2.4 UseCase 4: Sichtbarkeitsrechte festlegen

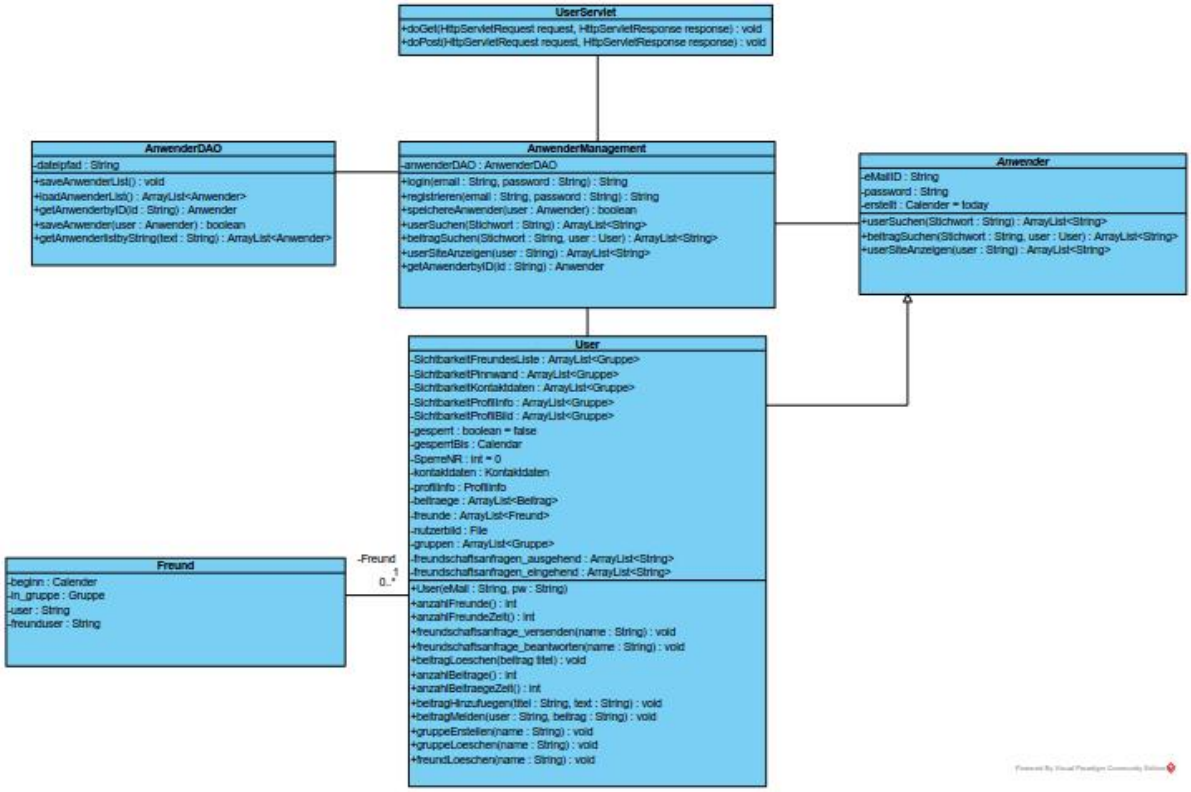
Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

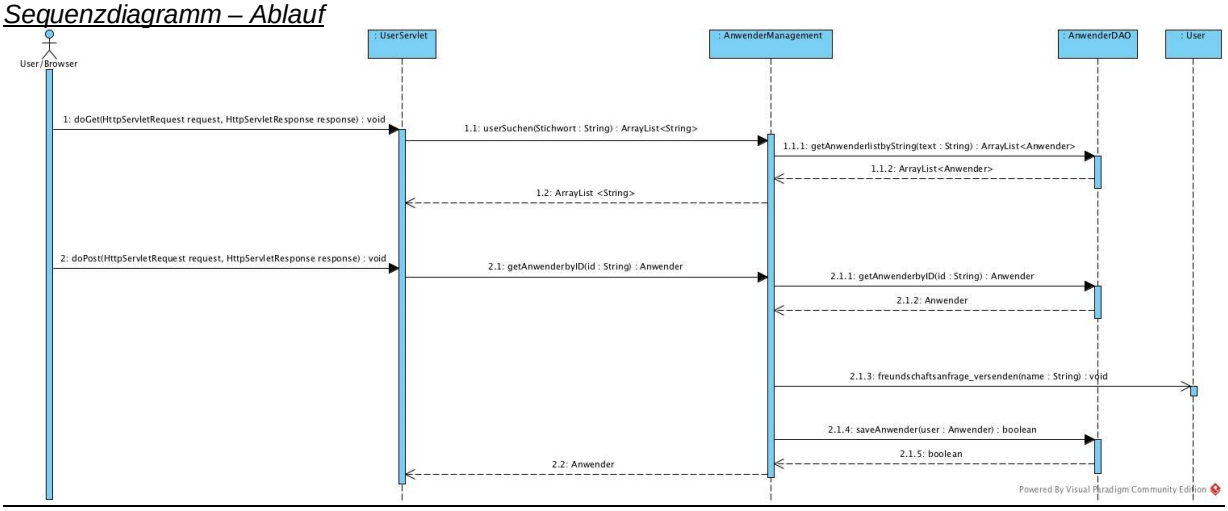
Textuelle Beschreibung – Ablauf

2.5 UseCase 5: Freund einladen

Klassendiagramm – Ausschnitt



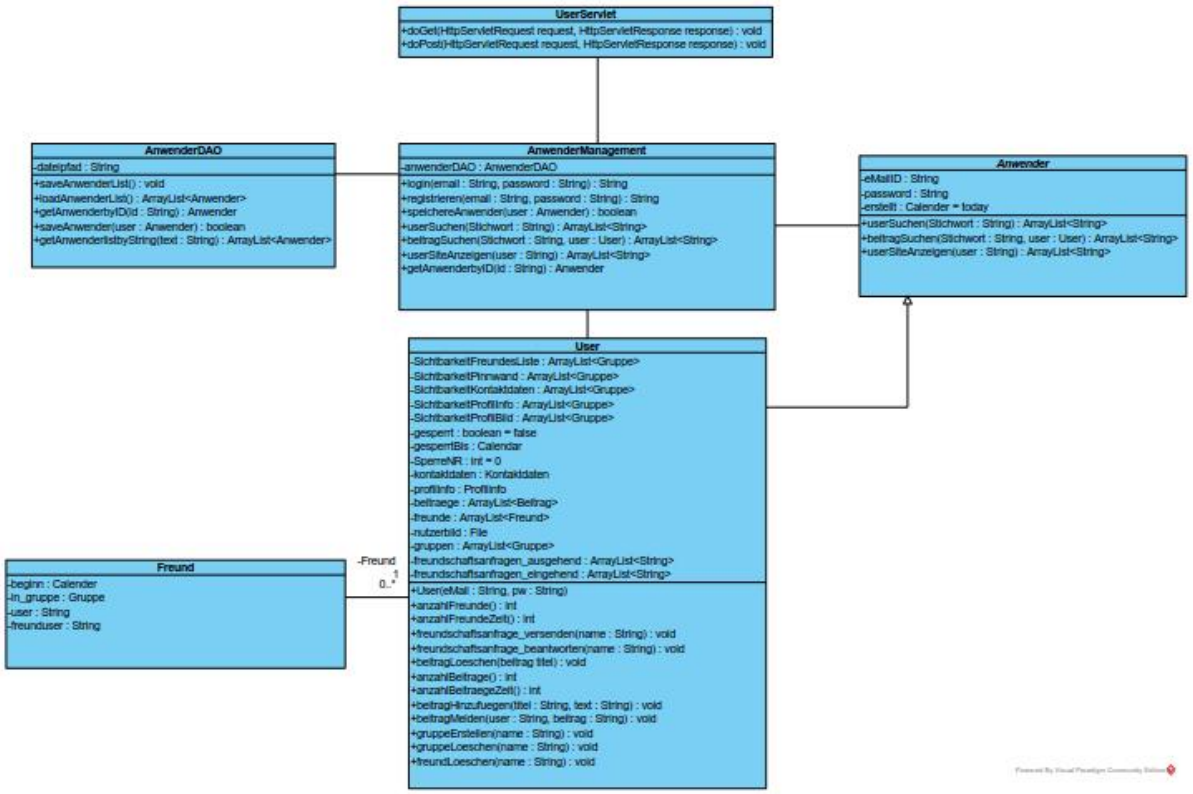
Sequenzdiagramm – Ablauf



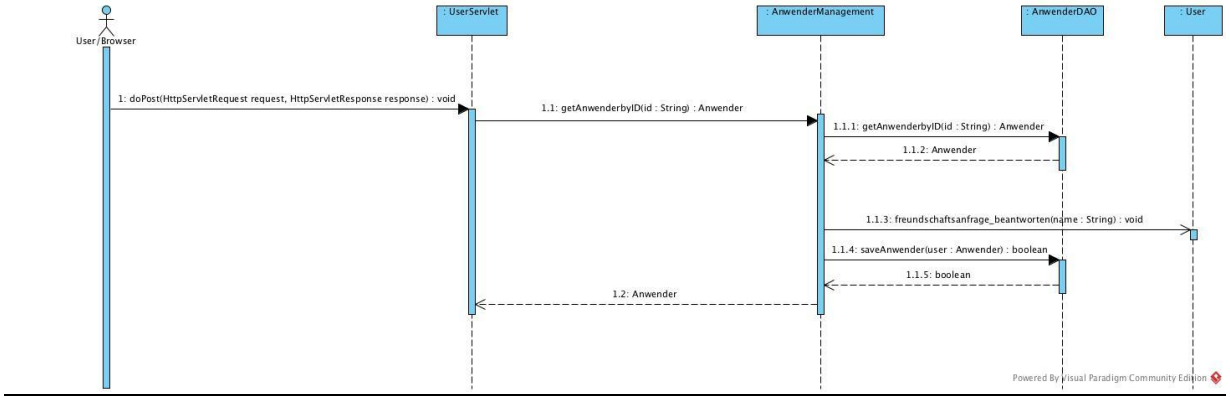
Textuelle Beschreibung – Ablauf

2.6 UseCase 6: auf Einladung reagieren

Klassendiagramm – Ausschnitt

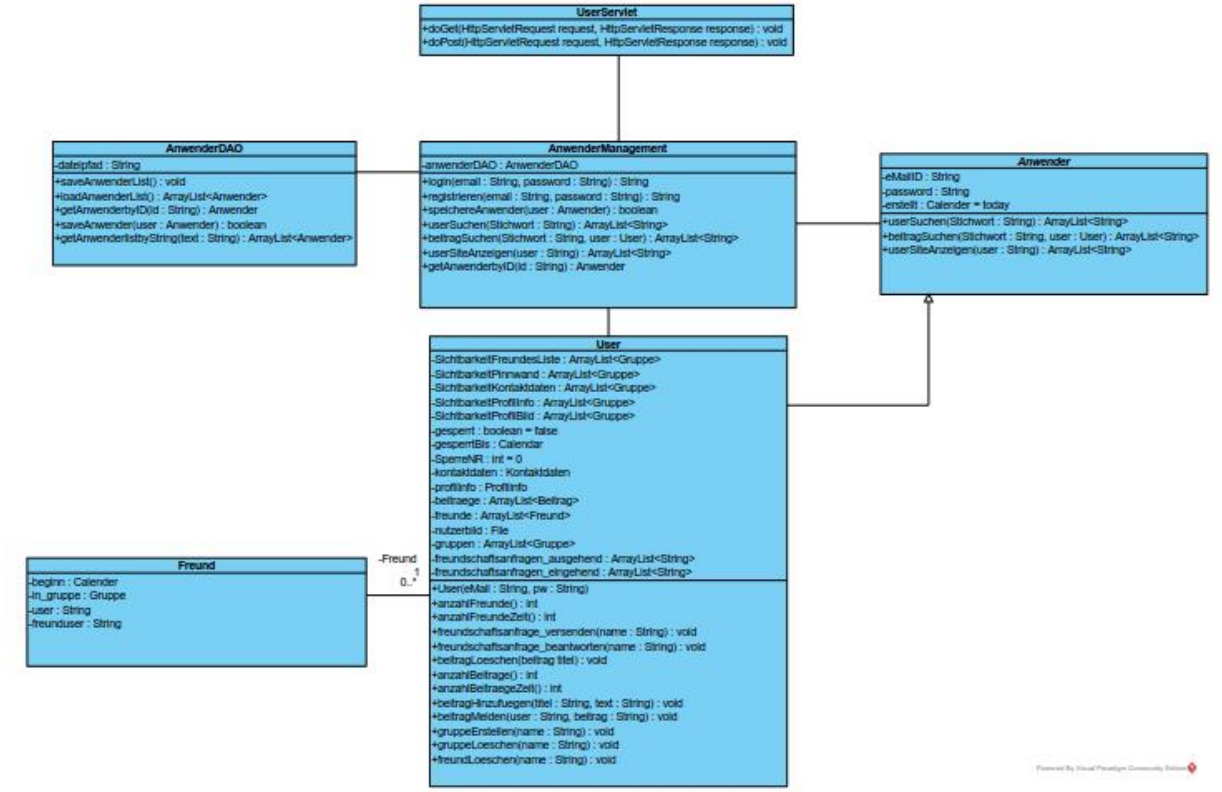


Sequenzdiagramm – Ablauf

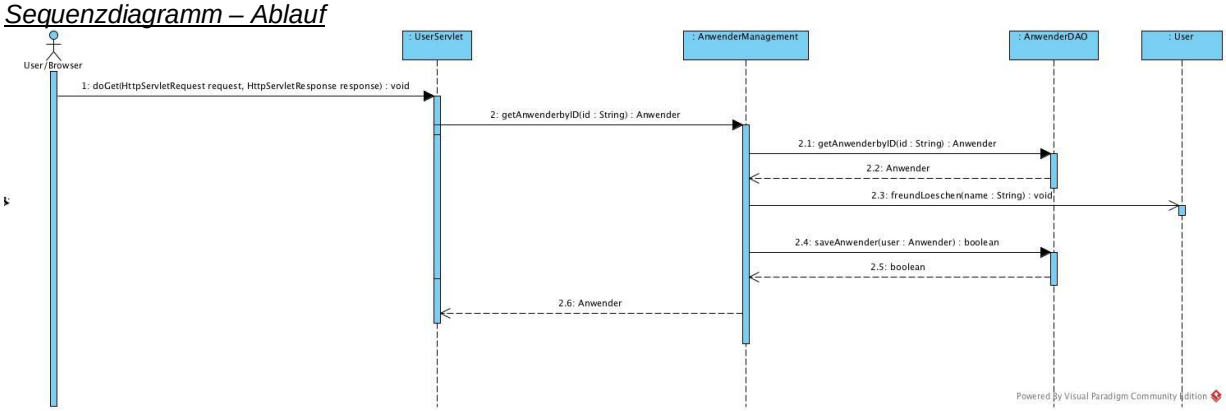


2.7 UseCase 7: Freund löschen

Klassendiagramm – Ausschnitt



Sequenzdiagramm – Ablauf



2.8 UseCase 8: Freunde gruppieren

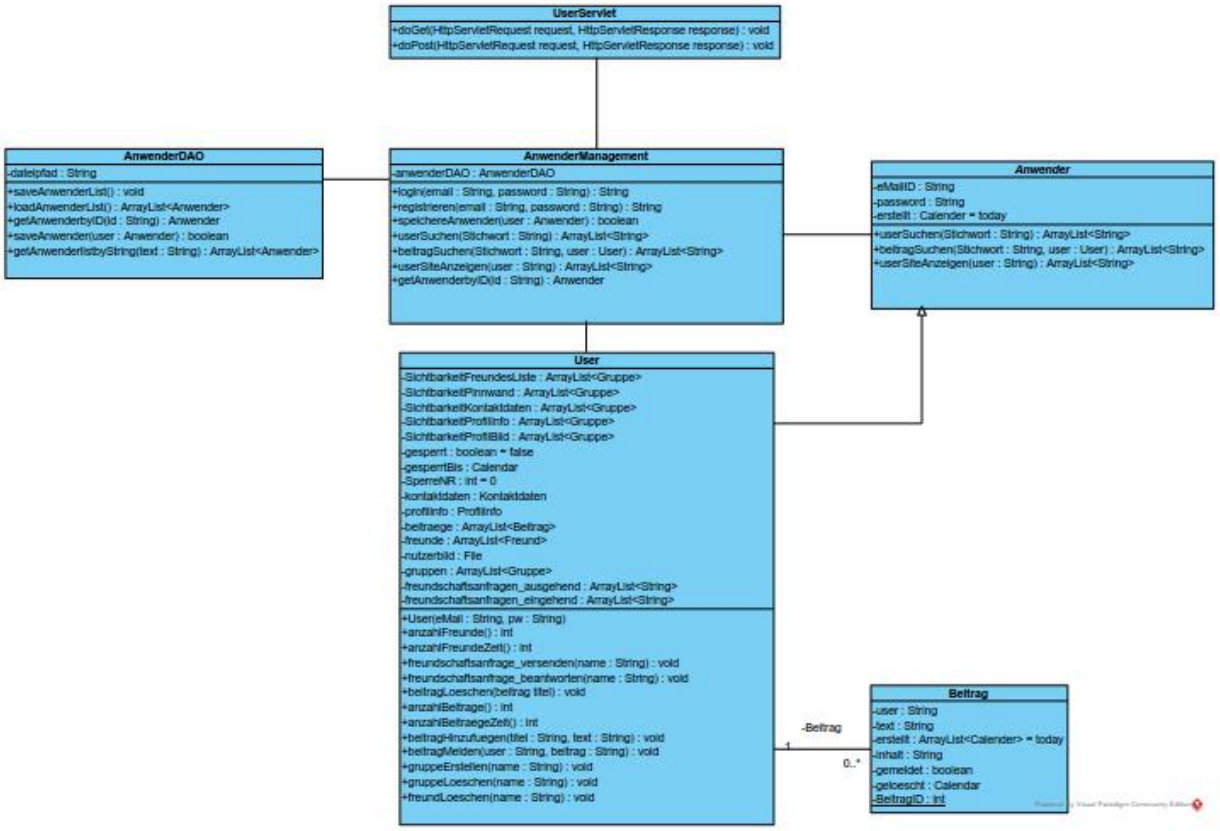
Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

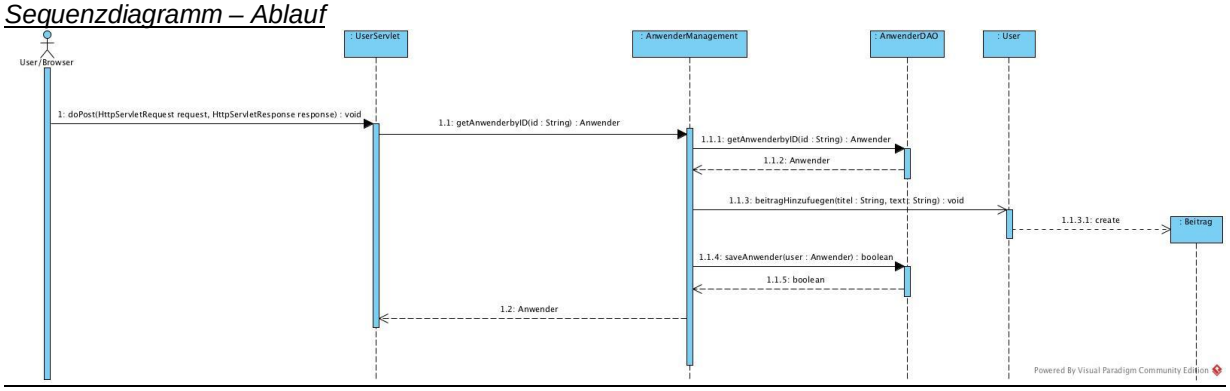
Textuelle Beschreibung – Ablauf

2.9 UseCase 9: Pinnwand Beitrag hinzufügen

Klassendiagramm – Ausschnitt



Sequenzdiagramm – Ablauf



2.10 UseCase 10: suchen

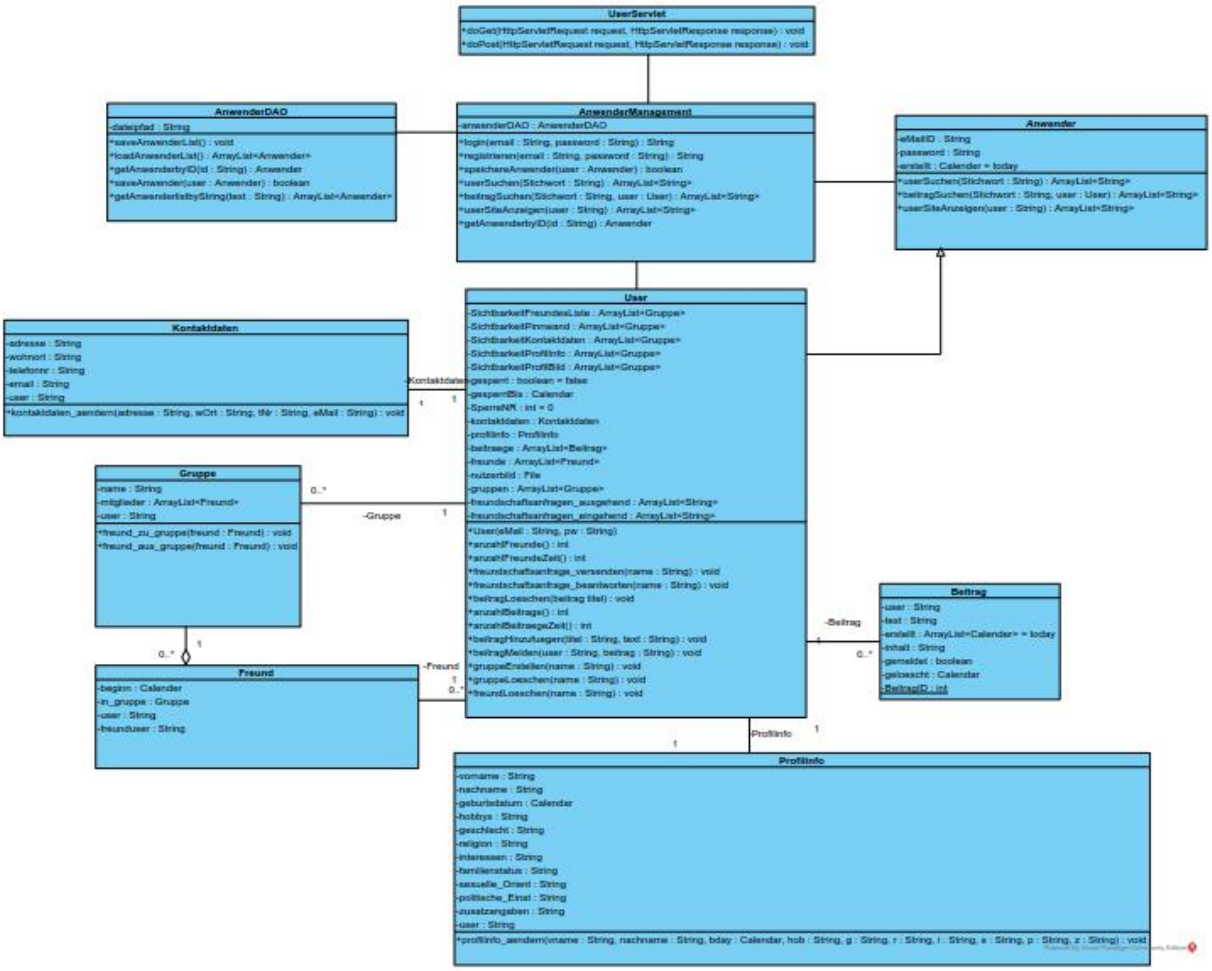
Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

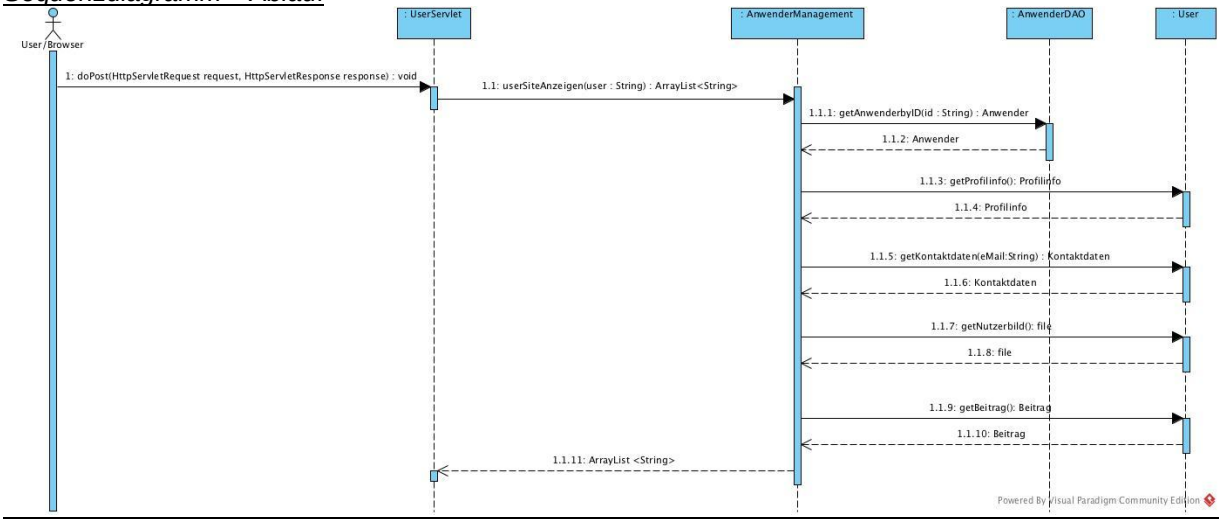
Textuelle Beschreibung – Ablauf

2.11 UseCase 11: UserSite ansehen

Klassendiagramm – Ausschnitt



Sequenzdiagramm – Ablauf



2.12 UseCase 12: Beitrag melden

Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

Textuelle Beschreibung – Ablauf

2.13 UseCase 13: gemeldeten Beitrag prüfen

Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

Textuelle Beschreibung – Ablauf

2.14 UseCase 14: Beitrag von Pinnwand löschen

Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

Textuelle Beschreibung – Ablauf

2.15 UseCase 15: User sperren

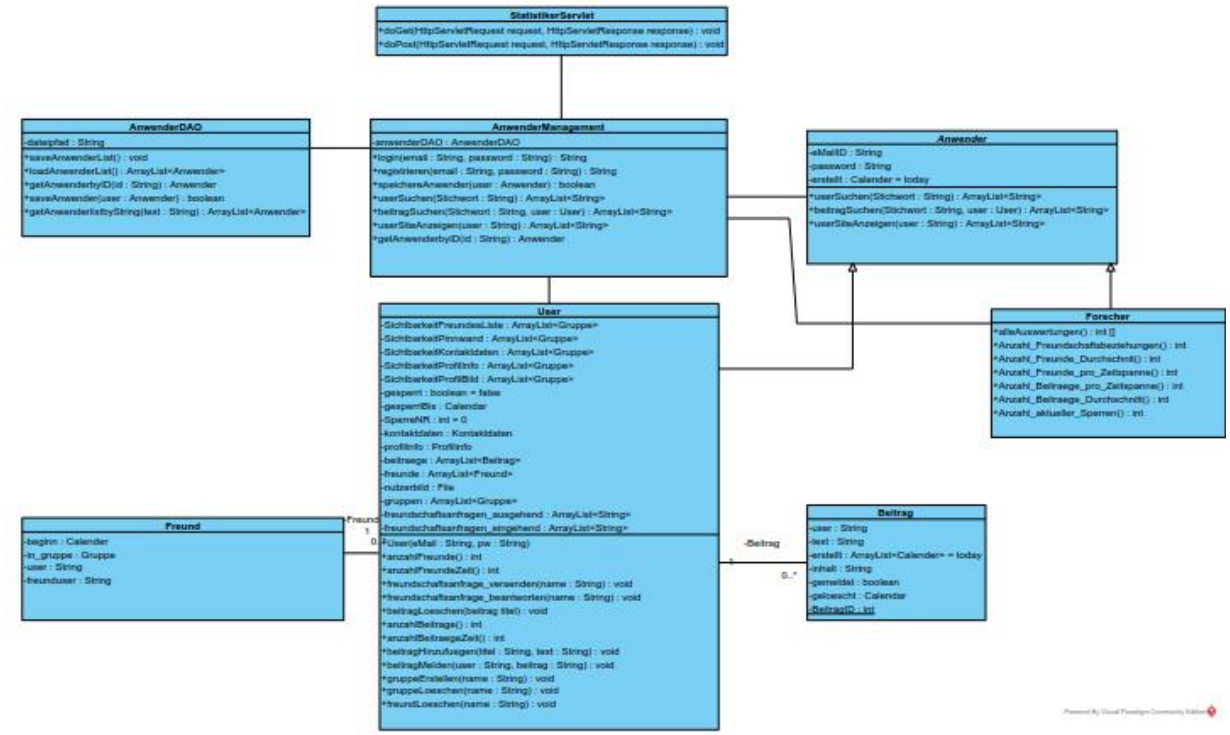
Klassendiagramm – Ausschnitt

Sequenzdiagramm – Ablauf

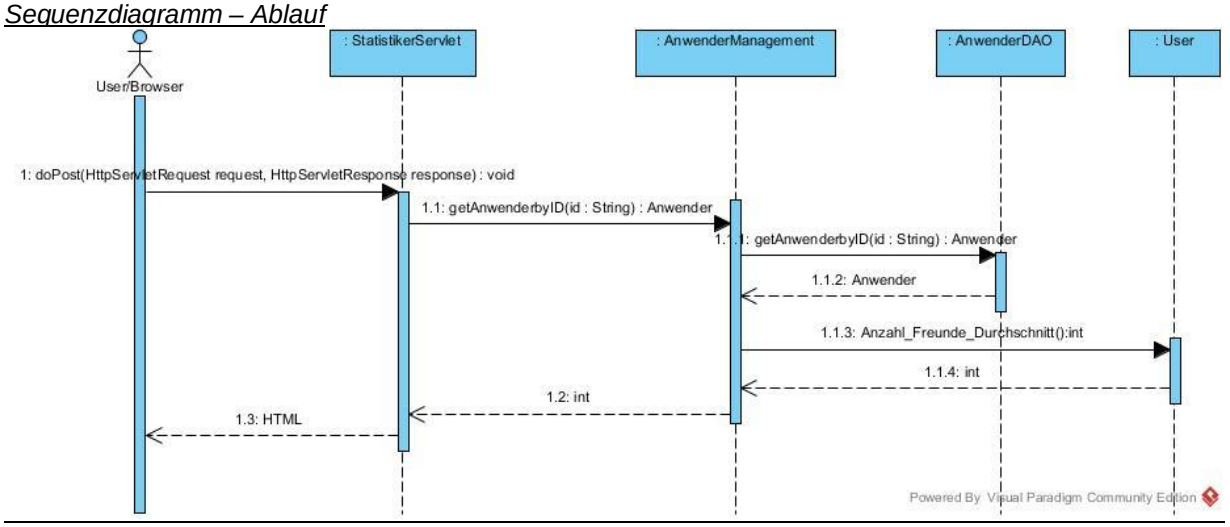
Textuelle Beschreibung – Ablauf

2.16 Use Case 16: Statistik auswerten

Klassendiagramm – Ausschnitt

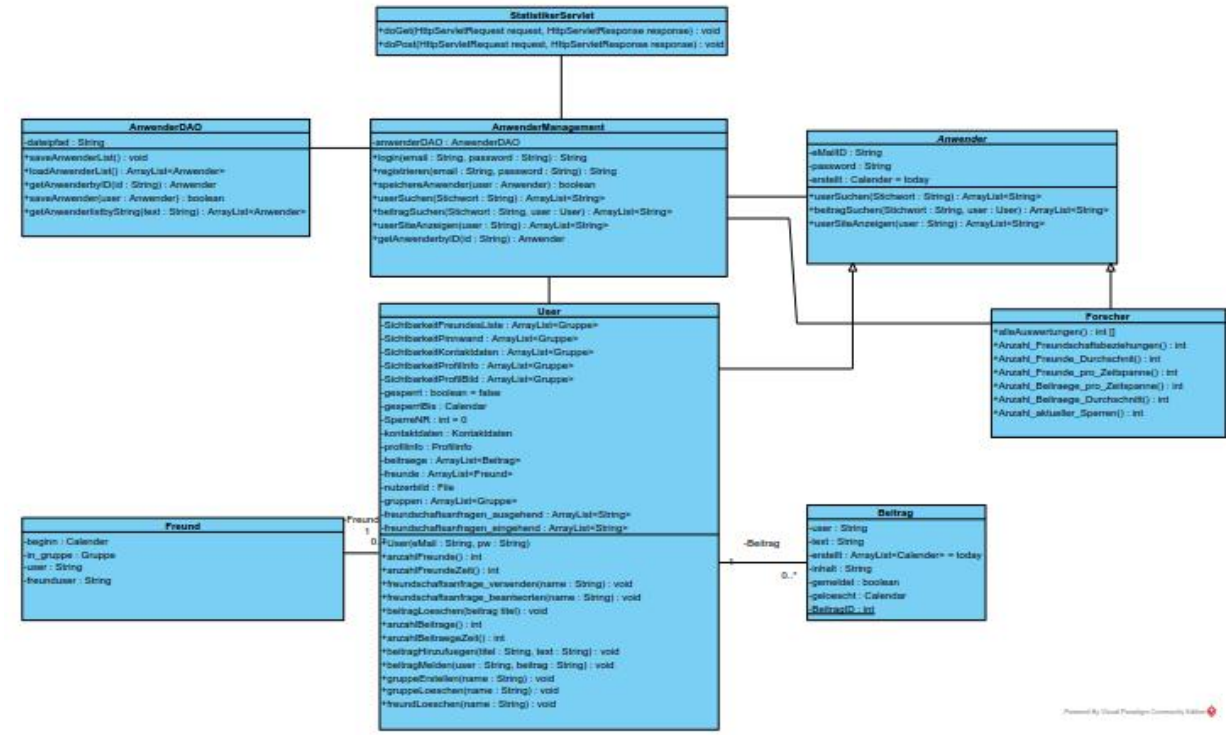


Sequenzdiagramm – Ablauf

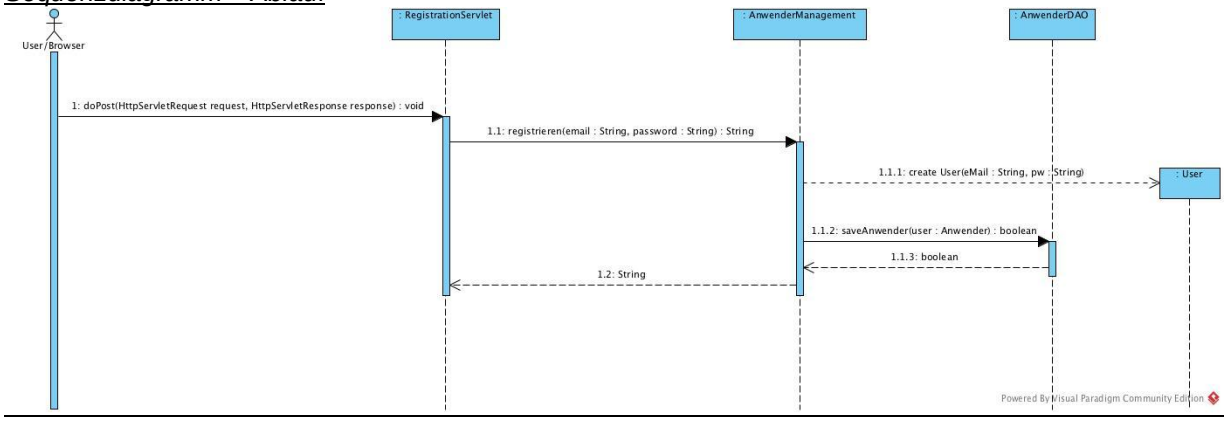


2.17 Registrieren

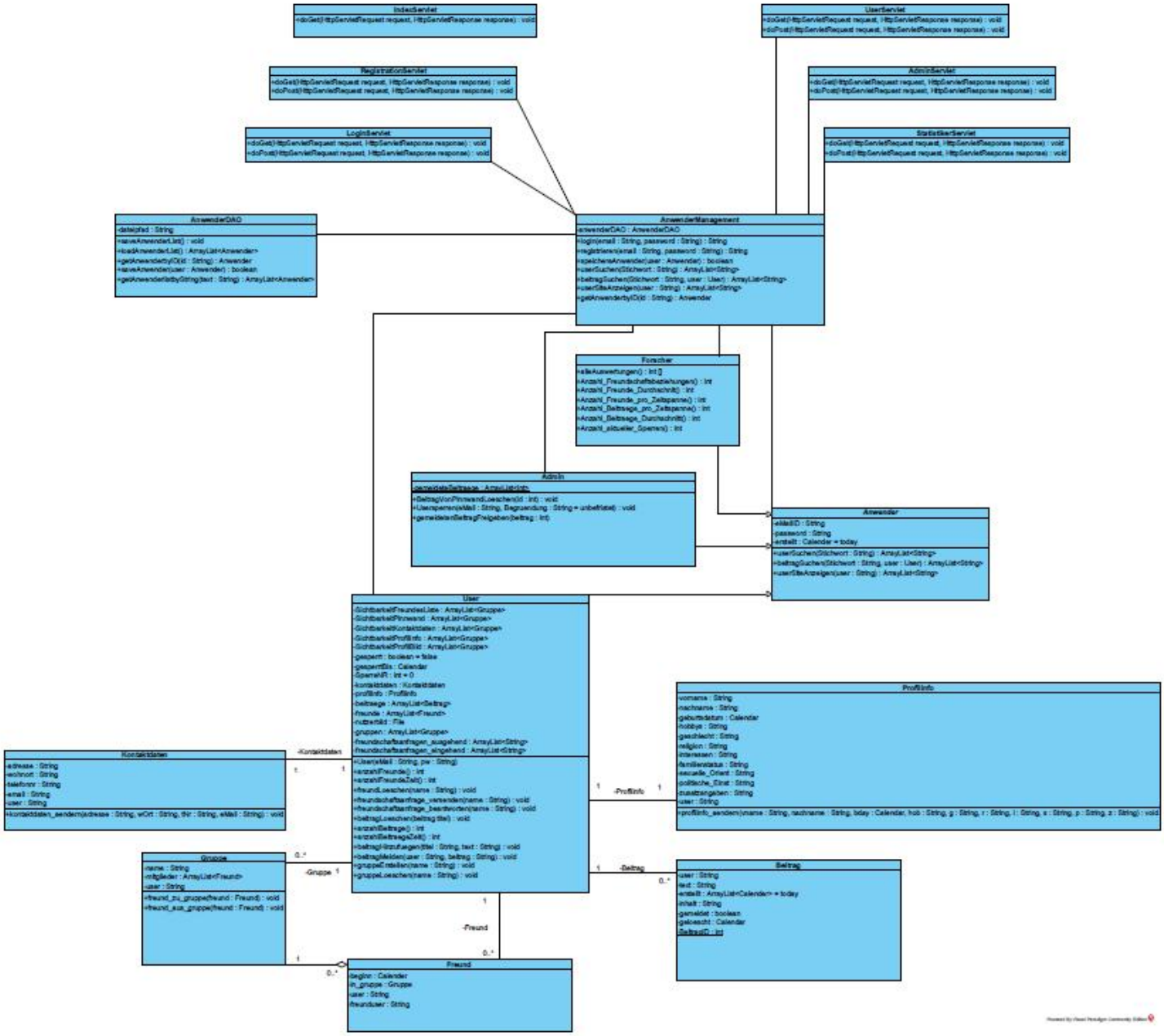
Klassendiagramm – Ausschnitt



Sequenzdiagramm – Ablauf

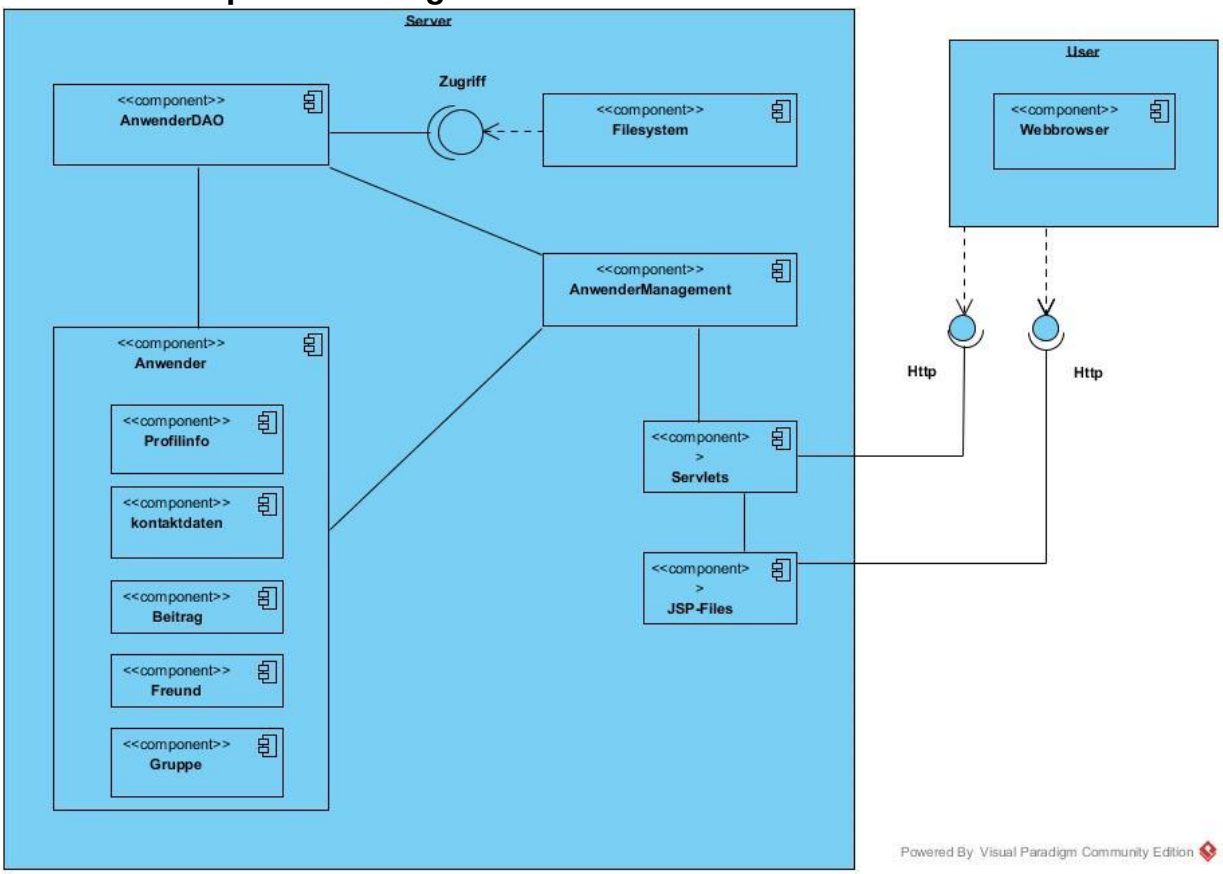


3 Übersichtsklassendiagramm



4 Architekturbeschreibung

4.1 Komponentendiagramm:



4.2 Deploymentdiagramm