Entwurf

Praxis der Softwareentwicklung

Wintersemester 17/18

Autorisierungsmanagement für eine virtuelle Forschungsumgebung für Geodaten

Autoren:

Bachvarov, Aleksandar
Dimitrov, Atanas
Mortazavi Moshkenan, Houraalsadat
Sakly, Khalil
Slobodyanik, Anastasia
Voneva, Sonya

24.12.17

Inhaltsverzeichnis

1 Einleitung

Während der Entwurfsphase wurde das Konzept für das zu erstellende Produkt formuliert. Die in diesem Dokument definierte Softwarearchitektur legt die Struktur und das Verhalten des Produkts fest. Durch Bestimmung grundlegender Entwurfsentscheidungen haben sich einige sinnvolle Änderungen zum Pflichtenheft ergeben.
Für die Funktion „Benutzer suchen“ wurden Suchparameter erweitert: die Suche soll anhand von Name, Email sowie Domain des Benutzers möglich sein. Nach dem Löschen einer Ressource werden alle Benutzer, die Besitzerrechte für diese Ressource hatten, benachrichtigt. Als Ressourcendaten werden auch Typ und kurze Beschreibung der Ressourcen gespeichert. Administratoren des Portals sollen in der Lage sein andere Benutzer zu blockieren.

2 Architektur

2.1 Model View Template (MVT)

Beim Design des Autorisierungsmanagements wird schnell deutlich, dass eine strikte Trennung zwischen Datenbank, Applikationslogik (Code) und Benutzeroberfläche (View) von Vorteil ist. Das Model-View-Template Prinzip (MVT), das in Abbildung 1 dargestellt ist, realisiert diese Trennung und wird in der Webanwendung eingesetzt.
MVT ist eine Ableitung des Architektur-Patterns “Model-View-Presenter“ (MVP) und wird hauptsächlich zum Erstellen von Benutzerschnittstellen verwendet.

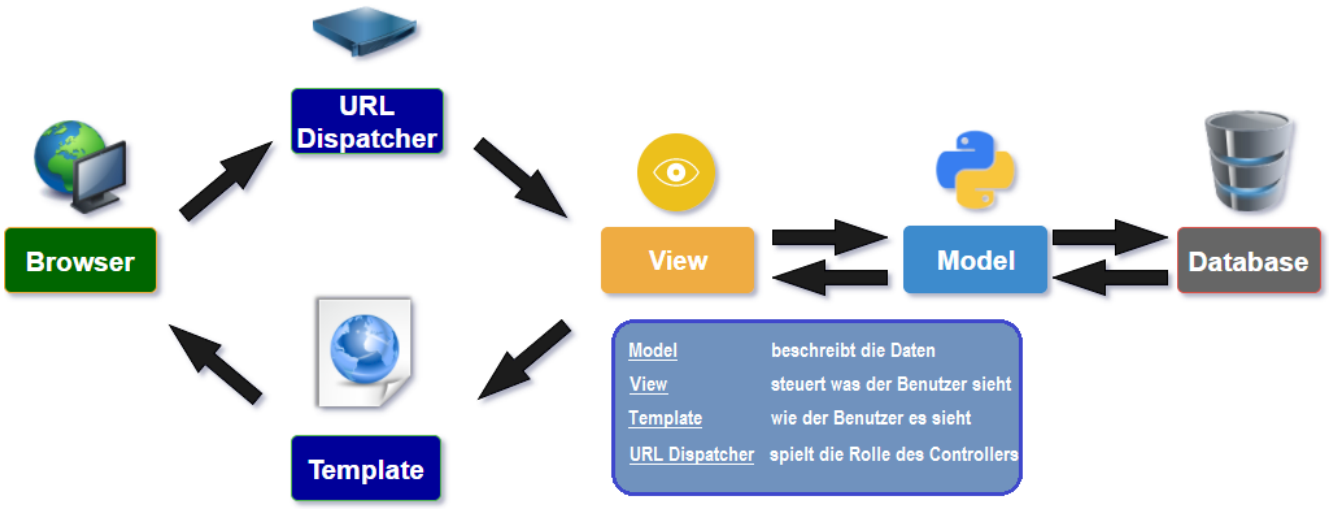

Abbildung 1: MVT Architektur

• Der **URL-Dispatcher** *(urls.py)* ordnet die angeforderte URL einer **View-Funktion** zu und ruft sie auf.
• Die **View-Funktion** *(views.py)* führt die angeforderte Aktion aus, bei der meistens mit Daten aus der Datenbank gehandelt wird. Sie kann auch andere Aufgaben beinhalten, zum Beispiel Logik der Anwendung.
• Das **Model** *(models.py)* ordnet die Daten den Tabellen der Datenbank zu und ermöglicht Interaktionen zwischen Daten und System.
• Nach der Ausführung der angeforderten Aufgaben gibt **View** ein HTTP-Antwortobjekt an den Webbrowser zurück, meistens nach der Übergabe der Daten durch ein **Template**.
• **Template** gibt üblicherweise HTML-Seiten zurück. Die Django-Template-Sprache bietet HTML-Autoren eine einfach zu erlernende Syntax und bietet gleichzeitig die gesamte für die Präsentationslogik erforderliche Leistung.

2.2 Betrieb des Django ORM

Object-Relational Mapping (ORM) Django ist ein in Python geschriebenes quelloffenes Webframework, das beispielweise folgendermaßen funktioniert:
• Django erstellt SQL-Tabellen aus den Modellen (Klassen) mit allen Klassenattributen als Tabellenspalten. Dies benötigt keine zusätzliche Implementierung der SQL-Abfragen.
• Um die Tabellen auszufüllen wird die Methode Model.save() benutzt.
• Auf die gleiche Weise ist es möglich, alle Einträge der Tabelle abzurufen. Dabei wird pro Tabelleneintrag eine Instanz von dem gewünschten Objekt zurückgegeben. Ein Beispiel für den Abruf aller Request-Objekte wird in Abbildung 2 gezeigt:

Abbildung 2: Ablauf des Transfers der Daten in MVT mit Django

2.3 Paketstruktur

Das folgende Paketdiagramm (Abbildung 3) stellt die verschiedenen Pakete, die in der Entwurfsphase gestaltet werden(inkl. Pakete die von Django vordefiniert sind - “Extern“), und ihre Beziehungen zueinander dar.

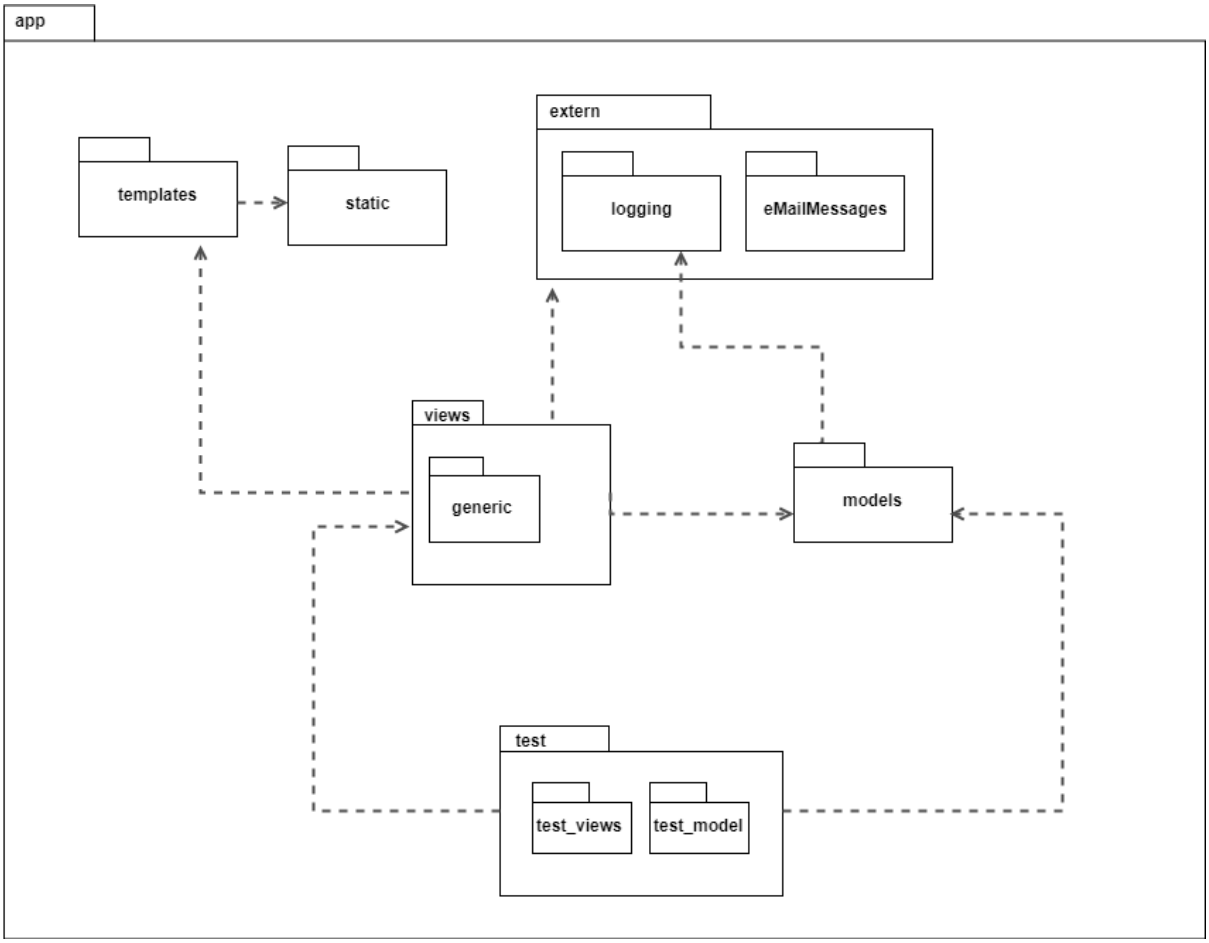

Abbildung 3: Paketdiagramm: die im Projekt zu implementierenden Pakete und Verwendungsbeziehung (Usage) zwischen denen.

Models

Ein Model enthält Informationen über Objekte und Verhalten von den Daten, die in der Datenbank gespeichert werden. Das Paket “Models“ besteht aus den Model-Klassen “User“, “Owner“, “Admin“, “Resource“, “Request“, “AccessRequest“, “DeletionRequest“ und einer Enumeration-Klasse “ResourceType“. Üblicherweise wird jede Model-Klasse zu einer Tabelle in der Datenbank übersetzt. Jedes Attribut der Klasse wird dabei als eine Spalte in der entsprechenden Tabelle repräsentiert. Systemereignisse, die im “Models“-Paket implementiert werden, werden durch Funktionalität des Pakets “‘Logging“ protokolliert.

Views

Im Paket "'Views" wird Anwendungslogik implementiert: Klassen aus dem "Models"-Paket werden hier in ein Zusammenspiel gesetzt. View ist eine Funktion, die als Eingabe ein Web-Request erfordert und eine Web-Response ausgibt. Diese Response kann prinzipiell alles beinhalten - zum Beispiel den HTML-Inhalt einer Webseite, ein Bild, eine 404 Error-Seite oder eine Weiterleitung zu einer anderen Webseite. Die View-Funktion definiert was genau angezeigt wird, nachdem eine URL angefragt wird. Zum Paket "Views" gehören alle Klassen, die eine solche Funktion spezifizieren. Dabei wird die äußere Form dieser Anzeige mit Hilfe der Pakete "'Templates" und "Static" definiert.
Das "Views"-Paket benutzt die externen Pakete "Logging" (zum Protokollieren der relevanten Systemereignissen) und "EMailMessages" (zum Senden und Empfangen von E-Mail Benachrichtigungen). Das Paket beinhaltet ein Subpaket:

- **Generic**
  Da bei der Entwicklung einer Webseite bestimmte Muster häfiger vorkommen, bietet Django eine Sammlung von solcher Views an. Diese Views haben eine abstrakte Form, damit sie mehrmals wiederverwendet werden können. Auf diese Weise spart man Zeit und Codezeilen.

Templates

"Django's template language is designed to strike a balance between power and ease. It's designed to feel comfortable to those used to working with HTML."

-The Django template language | Django documentation

Da Django ein Web Framework ist, ist es notwendig HTML-Dateien dynamisch zu generieren. Ein typisches Konzept dafür ist der Einsatz von Vorlagen (Templates). Eine Vorlage beinhaltet statische Teile vom gewünschten HTML Output, sowie Syntax, die beschreibt wie dynamischen Inhalt eingefügt wird. Um die Vorlagen graphisch zu gestalten benutzt "'Templates" das Paket "'Static".

Static

"CSS enables developers to separate content and visual elements for greater page control and flexibility. "

-CSS | Techopedia

Für die Webseite sollen zusätzliche Dateien wie Abbildungen, CSS- oder JavaScript-Dateien gespeichert werden. In Django sind diese Dateien als "Static Files" bezeichnet. Diese Dateien definieren das Aussehen des Portals. Ein vordefiniertes API von Django "django.contrib.staticfiles" hilft dabei diese Dateien zu organisieren.

Test

Das “Test“-Paket dient zum Testen des Produkts, um Qualität in jeder Phase der Entwicklung zu gewährleisten. Dieses Paket beinhaltet zwei Unterpaketen - “Test_models“ und “Test_views“ entsprechend für die Model- und die Viewklassen. Jedes Unterpaket enthält eine Testklasse spezifisch für jede Model- oder Viewklasse. Jede Testklasse hat eine setUp() Methode, die die entsprechende Objekte zum Testen erstellt und eine oder mehrere Modultestsmethoden für jede öffentliche Methode der getesteten Klassen. “Test_views“ und “Test_models“ benutzen entsprechend “Views“ und “Models“, um die Funktionen und Objekte von diesen Paketen zu testen.

Extern

- **Logging**
  Für Protokollieren der Systemereignissen bietet Django ein Logging-Modul an, das von Python vordefiniert ist. Dieses Paket beinhaltet Klassen und Funktionen, die Information über alle relevanten System- und Datenänderungen dokumentieren.

- **EmailMessages**
  Wie bei der “Logging“-Funktionalität, wird für das Senden von E-Mail Benachrichtigungen auch ein externes Paket von Django verwendet. Das Paket bietet API für automatisierte Erstatten und Absenden der E-Mail Benachrichtigungen, wobei alle Einzelheiten entsprechen dem System konfiguriert werden können.

2.4 Entwurfsmuster

In objektorientierten Programmiersprachen werden Entwurfsmuster (englisch Design Patterns) verwendet, um ein ganz bestimmtes Software-Entwurfsproblem zu lösen. Das Entwurfsmuster dient dabei als eine Art "Rezept", mit dessen Hilfe die gegebene Programmieraufgabe gelöst wird. Bei diesem Entwurf werden Entwurfsmuster verwendet, um einzelne Komponenten voneinander zu entkoppeln und das Geheimnisprinzip zu unterstützen.

Strategie-Muster

Strategiemuster hat das Ziel, eine Familie von Algorithmen zu definieren, zu kapseln und austauschbar zu machen. Dises Muster wird verwendet, wenn sich mehrere verwandte Klassen nur in ihrem Verhalten unterscheiden. Strategieobjekte ermöglichen, ein Objekt mit einem von mehreren möglichen Verhaltensalgorithmen zu konfigurieren. Dies bietet die Möglichkeit die notwendigen Funktionalitäten zu variieren.
Das Strategiemuster kommt in den Klassen "*Request*", "*AccessRequest*" und "*DeletionRequest*" bei der Funktion "*accept()*" zum Einsatz (Abbildung 4). Die vordefinierte Funktion der abstrakten Klasse wird abhängig von benötigter Funktionalität von erbenden Klassen implementiert.

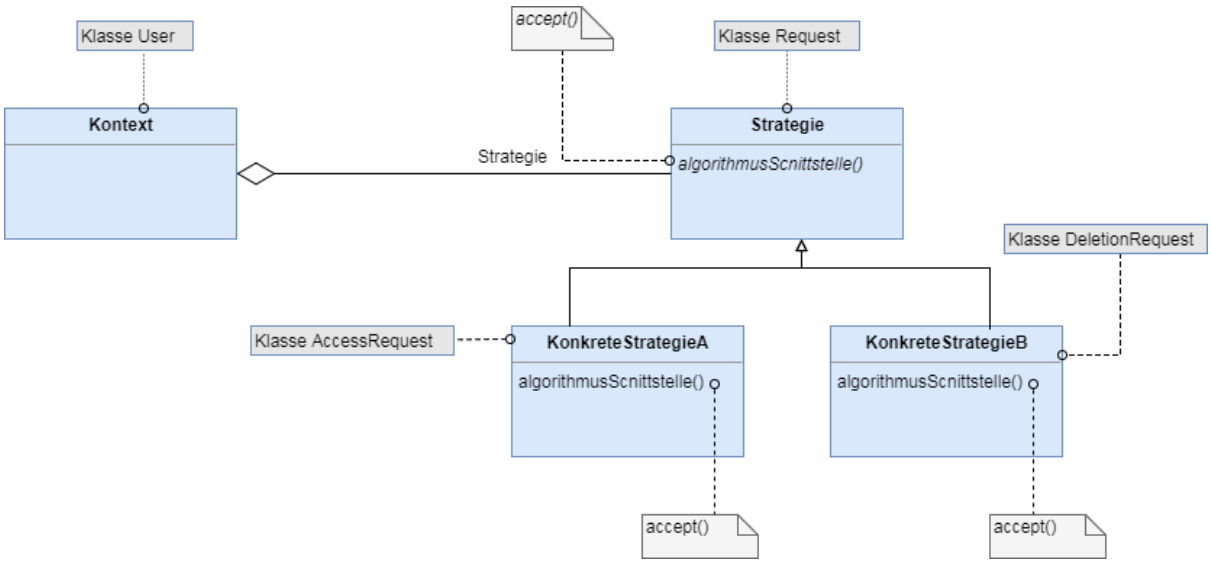

Abbildung 4: Strategiemuster Klassendiagramm. Durch die angehängten Notizen ist angezeigt, welche von zu implementierenden Klassen und ihren Funktionen der Klassen bzw. Funktionen vom Strategiemuster entsprechen.

Befehlsmuster

Das Befehlsmuster wird zur Bearbeitung von den Requests (Funktionen "*deny()*" und "*accept()*" in Klasse "*Request*") verwendet. Mit Hilfe dieses Musters wird die Bearbeitung der Requests von den Requests selbst, wie in der Abbildung 5 angezeigt ist, getrennt.
Das Befehlsmusters ermöglicht es, ein bestimmtes Ereignis in einem dafür vorgesehenen Handler zu bearbeiten und dieses damit in einem Objekt zu kapseln. Jeder Handler lässt sich eindeutig einer Aufgabe und einem Ereignis zuordnen, sodass keine Verschränkung zwischen zwei verschiedenen Handlern auftritt. Außerdem lässt sich leichter ein weiterer Handler hinzufügen oder ein bestehender Handler austauschen.

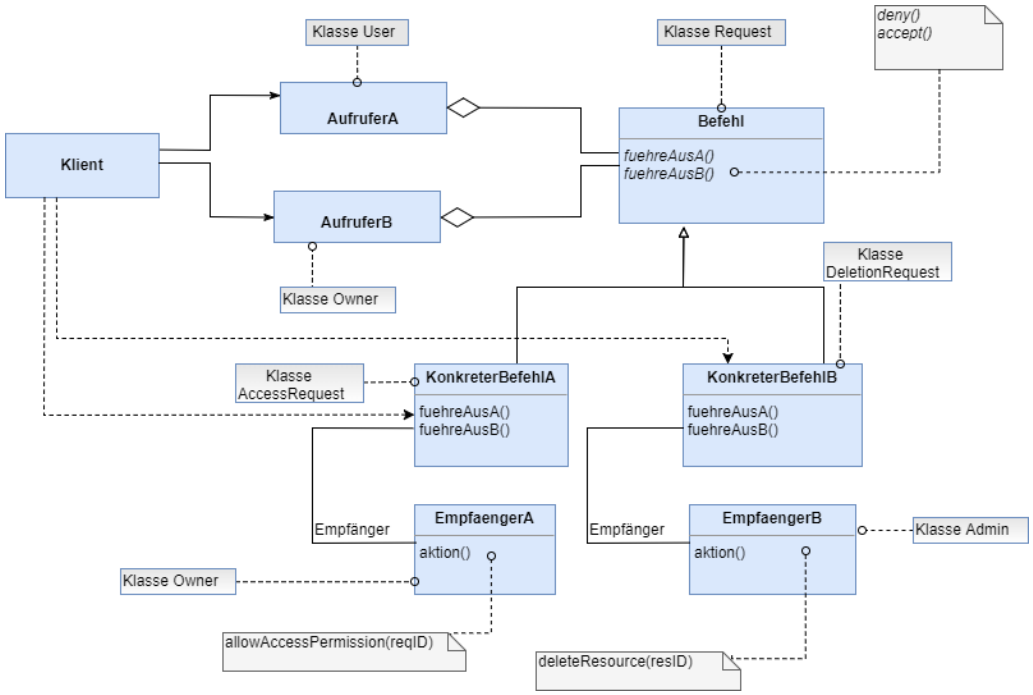

Abbildung 5: Befehlsmuster Klassendiagramm. Durch die angehängten Notizen ist angezeigt, welche von zu implementierenden Klassen und ihren Funktionen der Klassen bzw. Funktionen vom Befehlsmuster entsprechen.

2.5 URL Verzeichnis

Alle Interaktionen zwischen den Benutzern und dem Produkt werden durch eine graphische Benutzeroberfläche (GUI) unterstützt. Die GUI des Produkts wird als eine Webseite dargestellt, wobei sämtlicher Inhalt und Funktionalität unter URL Verzeichnis abgelegt werden (Abbildung 6). Eine URL lokalisiert ein oder mehrere Views, die entweder als ein Main-Fenster oder ein Dialog präsentiert werden.

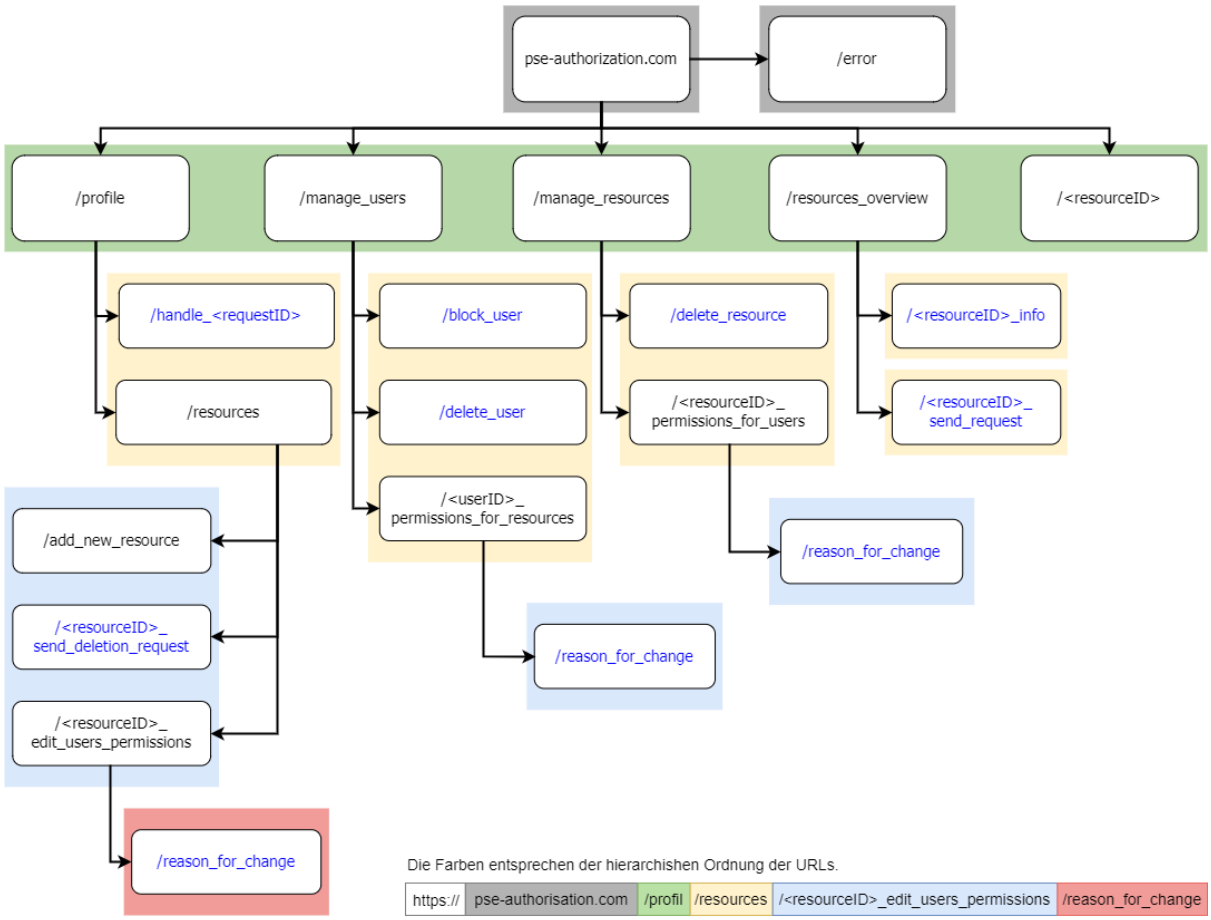

Abbildung 6: URL Struktur des Portals. Jedes URL korrespondiert mit entsprechendem View. Schwarz beschriftete URLs weisen zu Views, die im Main-Fenster angezeigt werden, blaue zu solchen, die im Dialogfenster angezeigt werden.

pse-authoritation.com – Mockup-Seite, dient zum Einloggen in das Portal, enthält Autorisierungsformular und wird nach der Entwicklung durch das Autorisierungsmanagement jenes Systems, in welches das Projekt integriert wird, ersetzt.

- **/profile** - Benutzerprofil-Seite, enthält persönliche Information des Benutzers und Übersicht der Requeste, die an den Benutzer adressiert sind.

  Benutzerprofil-Seite ist mit Subseiten versehen:

  - **/handle_<requestID>** - Dialog zur Bearbeitung des Requests, enthält "Annehmen"- und "Ablehnen"-Buttons, sowie ein Eingabefeld für kurze Begründung.

- **/resources** - Seite mit Übersicht von Ressourcen, für die der Benutzer Besitzerrechte hat. Diese Seite enthält Subseiten:
  - **/add_new_resource** - Seite zum Erstellen neuer Ressourcen. Enthält Upload-Dialog, Dialog zur Vergabe von Rechten und Eingabefelder für Namen und Beschreibung der Ressourcen.
  - **/<resourceID>_edit_users_permissions** - Dialog zu Vergabe und Entzug von Rechten für die Ressource.
  - **/<resourceID>_send_deletion_request** - Dialog zum Absenden von Löschrequest für die Ressource.
- **/manage_users** - Administratorseite zur Benutzerverwaltung, enthält Liste von Benutzern und Subseiten:
  - **/block_user** - Dialog zum Blockieren des Users.
  - **/delete_user** - Dialog zum Löschen des Users.
  - **/<userID>_permissions_for_resources** - Dialog zur Änderung von Rechten des Benutzers.
- **/manage_resources** - Administratorseite für Ressourcenverwaltung, enthält Liste von Ressourcen und Subseiten:
  - **/<resourceID>_permissions_for_users** - Dialog zur Änderung von Zugriffsrechten für die Ressource.
  - **/delete_resource** - Dialog zum Löschen der Ressource.
- **/resources_owerview** - Übersicht von Ressourcen, enthält Liste aller Ressourcen und folgende Subseiten:
  - **/<resourceID>_info** - enthält Metadaten der Ressource.
  - **/<resourceID>_send_request** - Dialog zum Absenden des Requests für die Ressource.
- **/<resourceID>** - Ressourcenseite, enthält Metadaten und Link für Zugriff auf die Ressource.

3 Datenhaltung

Relevante Daten über die Benutzer, Ressourcen, deren Beziehungen (Rechte) und Interaktionen (Requests) werden in einer Datenbank gespeichert. Das in diesem Projekt benutzte Framework Django bietet integrierte objektrelationale Abbildung für die Datenbanksysteme MySQL, Oracle, PostgreSQL und SQLite.
Da SQLite-Bibliotheken sich direkt in Anwendungen integrieren lassen, sodass keine weitere Server-Software benötigt wird, wurde entschieden, SQLite3-Syntax zu verwenden. Im Projekt wird das eingebettete Datenbanksystem (für Frontend nicht sichtbar) entworfen, sodass auf erweiterte Funktionalitäten von komplizierteren Datenbanksysteme verzichtet werden kann. Dabei entstehen auch einige Vorteile: die SQLite-Bibliothek ist nur wenige hundert Kilobyte groß und durch das Einbinden der Bibliothek wird die Anwendung um Datenbankfunktionen erweitert, ohne auf externe Softwarepakete angewiesen zu sein.

3.1 Datenbank

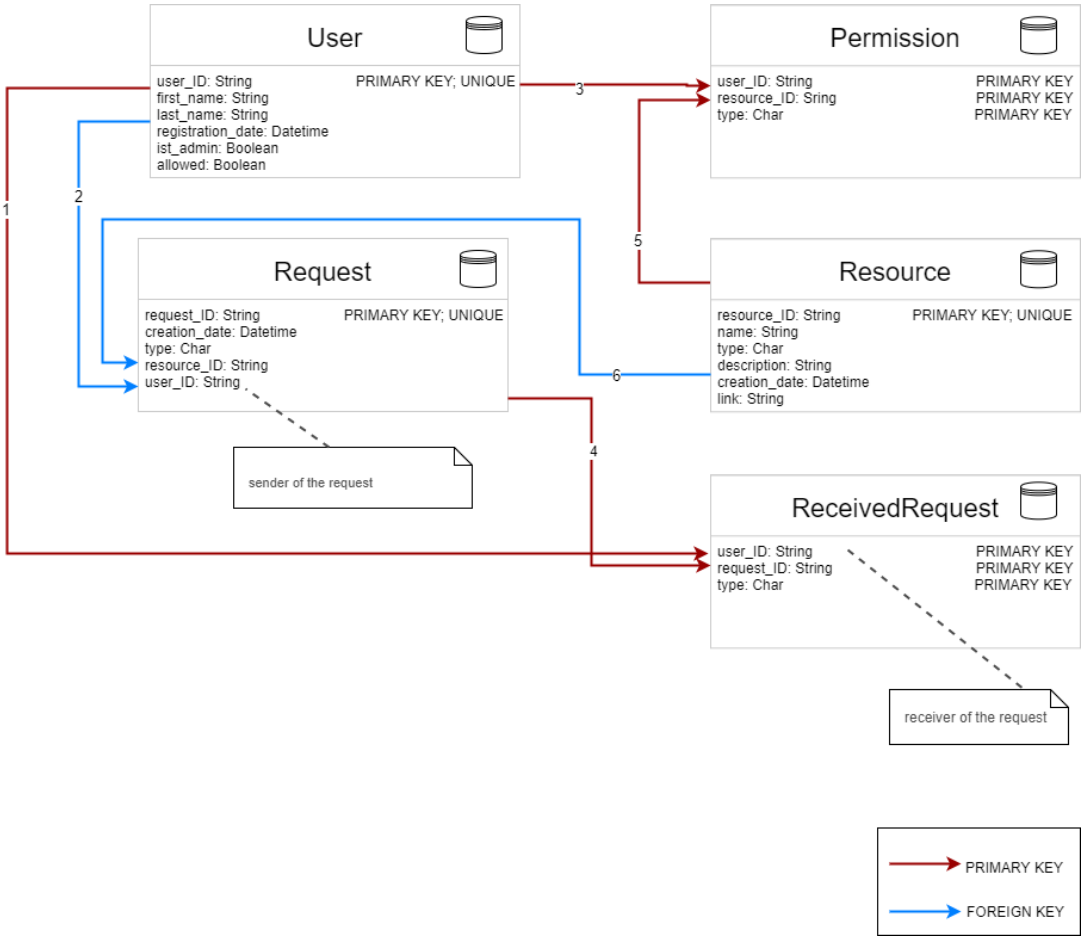

Abbildung 7: Schema der Datenbank zur Webseite. Jedes Rechteck repräsentiert eine Tabelle, jedes Attribut - eine Spalte in der entsprechenden Tabelle in der Datenbank. Die Pfeile geben Auskunft welches Attribut in welcher Tabelle als Schlüssel vorkommt. Blau steht für Fremd- und rot für Primärschlüssel.

"A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table."

—Models | Django documentation

Die in diesem Projekt zu entwerfende Datenbank hat die auf Abbildung 7 dargestellte Struktur. Die Tabelle "User" speichert einen eindeutigen Identifikator (*ID*) des Benutzers, seinen Vor- und Nachnamen, sowie das Datum, an dem er sich registriert hat. Zusätzlich hat die Tabelle eine Spalte "is_admin", die zur Unterscheidung zwischen "normalem" Benutzer und Administrator dient. In der Spalte "allowed" wird Status des Benutzers gespeichert: "true", falls der Benutzer das Portal benutzen darf, und "fakse", falls er von einem Administrator blockiert wurde.
Metadaten über die Ressourcen werden in der Tabelle "Resource" gespeichert: ID, Name, generischer Typ, kurze Beschreibung und Erstellungsdatum. Zusätzlich hat jeder Eintrag in der Tabelle eine Link zur "echten" Resource.
In der Tabelle "Request" werden relevante Daten über die Requests gespeichert: ID, Erstellungsdatum, Typ des Requests (Zugriffs- oder Löschrequest), ID der Ressource, für die der Request gesendet wird und ID des Absenders. Die letzte zwei sind Foreign Keys aus den Tabellen "Resource", bzw. "User" (Pfeile 2 und 6). Auf diese Weise werden Redundanzen vermieden und die Datenbank bleibt konsistent. Requests werden sofort nach der Erstellung gespeichert und nach der Bearbeitung wieder gelöscht.
Abgesendete, noch nicht bearbeitete Requests werden in der Tabelle "ReceivedRequest" dupliziert, dabei wird ein Eintrag pro Empfänger erstellt. Diese Tabelle enthält folgende Spalten: ID und Typ des Requests und ID des Empfängers. ID des Requests und ID des Empfängers sind Schlüssel aus den Tabellen "Request", bzw. "User" (Pfeile 1 und 4). Der Primary Key besteht aus dem Tupel $\{user\_ID, request\_ID, type\}$.
Rechte werden in der Tabelle "Permission" dargestellt. Jeder Eintrag dieser Tabelle präsentiert entweder Zugriffs- oder Besitzerrechte, dabei werden folgende Attribute gespeichert: ID des Benutzers, ID der Ressource und Typ des Eintrags (Zugriffs- oder Besitzerrechte). Die erste zwei sind Schlüssel aus den Tabellen "User", bzw. "Resource" (Pfeile 3 und 5). Alle Attribute zusammen formieren den Primary Key dieser Tabelle.

3.2 Logging

Logging dient der Aufzeichnung und Nachvollziehbarkeit von Daten- und Systemänderungen, sowie Fehlerzuständen des Produkts. Mithilfe eines externen Django-Moduls ("Logging") wird eine Log-Datei geführt. Auswertung dieser Datei ist durch Administratoren vorgesehen.
Diese Datei beinhaltet zeitlich geordnete Ereignisse des Systems. Aktivitäten wie Senden eines Requests, Bearbeitung eines Requests, Erstellen/Löschen einer Ressource und Editieren der Rechte können der Log-Datei entnommen werden. Zu jedem Ereignis wird auch den genauen Zeitpunkt und Systemzustand gespeichert.

4 Klassenübersicht

4.1 Model

Klasse User

`User extends Model`

User-Klasse bildet eine Elternklasse aller Benutzertypen. Sie stellt alle Attribute fest, die für Darstellung des Benutzers notwendig sind, und beschreibt Aktivitäten, die von allen Benutzertypen ausgeführt werden können.

Attribute

- `[final] USER_ID : String`
  Eindeutiger Identifikator des Benutzers
- `[private] name : String`
  Name und Vorname des Benutzers
- `[private] email : String`
  Email des Benutzers

Funktionen

- `[public] searchResource(searchParameters: String): List<Resource>`
  Funktion liefert `List<Resource>`, die alle Ressourcen enthält, deren Metadaten den `searchParameters` entsprechen.
- `[public] addResource( ): Resource`
  Funktion erstellt eine Instanz der Klasse `Resource`, Metadaten werden dabei in Datenbank gespeichert, wobei `User` als Besitzer dieser Ressource definiert wird.
- `[public] accessResource(resourceID: String): void`
  Falls Instanz der Klasse `User`, die diese Funktion aufruft, entsprechende Rechte hat, werden Metadaten der Ressource, deren `resourceID` als Parameter übergeben wird, von Datenbank abgerufen.
- `[public] sendAccessRequest(resourceID: String): void`
  An alle Besitzer der Ressource, deren `resourceID` als Parameter übergeben wird, wird /inlinecodeAccessRequest gesendet.
- `[public] cancelRequest(requestID: String): void`
  Der Request, dessen `requestID` als Parameter übergeben wird, wird aus der Datenbank gelöscht.

Klasse Owner

`Owner extends User`

Diese Klasse erweitert Klasse `User`. Nachdem eine Instanz der `User` Klasse Besitzerrechte für eine oder mehrere Ressourcen bekommt, wird ein neuer Eintrag in der Datenbank-Tabelle "Owner", beziehungsweise eine neue Instanz der Klasse `Owner` erstellt.

Funktionen

- `[public] allowAccessPermission(resourceID:String, userID:String): void`
  Instanz der Klasse `User` bekommt `AccessPermission` für Instanz der Klasse `Resource`. `userID` des Benutzers und `resourceID` der Ressource werden als Parameter übergeben.

- `[public] allowAccessPermission(requestID:String): void`
  Diese Funktion lädt vorherige Funktion über, dabei wird als Parameter `requestID` eines Requests übergeben. Der Absender dieses Requests bekommt `AccesPermission` für die Ressource, der `Request.RESORCE_ID` gehört.

- `[public] deleteAccessPermission(resourceID:String, userID:String): void`
  Der Instanz der Klasse `User`, die übergebenes Parameter `userID` besitzt, wird `AccessPermission` für die Ressource, deren `resourceID` jeweil als Parameter übergeben wird, entzogen.

- `[public] denyAccessPermission(requestID:String): void`
  `AccessRequest` wird abgelehnt: Instanz, der `requestID` entspricht, wird gelöscht.

- `[public] allowOwnerPermission(resourceID:String, userID:String): void`
  Instanz der Klasse `User` bekommt `OwnerPermission` für Instanz der Klasse `Resource`. `userID` und `resourceID` dieser Instanzen werden als Parameter übergeben.

- `[public] sendDeletionRequest(resourceID:String): void`
  Diese Funktion erstellt eine Instanz der Klasse `DeletionRequest`. Das übergebene Parameter `resourceID` wird als `Request.REQUEST_ID` und `userID` des `Owner` als `Request.SENDER_ID` gespeichert.

Klasse Admin

`Admin extends Owner`

`Admin` erweitert `Owner`-Klasse. Falls ein Benutzer Administratorrechte besitzt, wird ein Eintrag in der Tabelle “Admin“ der Datenbank gespeichert und eine Instanz der Klasse `Admin` erstellt.

Funktionen

- `[public] searchUser(searchParameters:String): List<User>`
  Funktion liefert die Liste `List<User>` zurück, die alle Benutzer enthält, deren Daten (Name, Email, Domain) den `searchParameters` entsprechen.

- `[public] blockUser(userID:String): void`
  Nach Ausführung dieser Funktion wird sich der Benutzer, dem die `userID` entspricht, nicht mehr beim Portal anmelden können: “allowed“-Flag des entsprechenden Eintrags der “User“-Tabelle wird auf `false` umgestellt.

- `[public] deleteUser(userID:String): void`
  Diese Funktion löscht die `userID` enthaltenden Einträge aus allen Tabellen der Datenbank: `user`, alle seine Rechte und Requests. Ressourcen, für die er Rechte hat, werden nicht gelöscht.

- `[public] deleteResource(resourceID:String): void`
  Diese Funktion löscht die mit `resourceID` Einträge aus allen Tabellen der Datenbank: Metadaten der Ressource, alle Rechte für sie und alle für sie erstellten Requests. Alle Benutzer, die Besitzerrechte für diese Ressource hatten, werden benachrichtigt.

- `[public] acceptDeletionRequest(requestID:String): void`
  `DeletionRequest`, dem die `requestID` entspricht, wird genehmigt: Instanz der `Resource`-Klasse wird gelöscht, sowie alle entsprechenden Einträge in der Datenbank: Metadaten der Ressource, alle Rechte für sie und alle für sie erstellten Requests. Alle Benutzer, die Besitzerrechte für diese Ressource hatten, werden benachrichtigt.

- `[public] denyDeletionRequest(requestID:String): void`
  Diese Funktion lehnt den Request mit `requestID` ab: entsprechende `DeletionRequest`-Instanz wird gelöscht und der Absender wird benachrichtigt.

- `[public] deleteOwnerPermission(resourceID:String, userID:String): void`
  Diese Funktion löscht den Eintrag mit `resourceID`, `userID` und Typ “Òwner“ aus der Tabelle “Permission“.

Klasse Resource

`Resource extends Model`

Die Klasse `Resource` repräsentiert Ressourcen und kapselt dabei ihre Metadaten als Attribute.

Attribute

- `[public] RESOURCE_ID : String`
  Eindeutiger Identifikator des entsprechenden Eintrags in Datenbank.
- `[private] type : ResourceType`
  Typ der Ressource.
- `[private] name : String`
  Name der Ressource, wird vom Ersteller eingegeben.
- `[private] description : String`
  Beschreibung des Inhalts der Ressource, wird bei der Erstellung eingegeben.
- `[private] CREATION_DATE : String`
  Datum der Erstellung, kann nicht geändert werden.

Funktionen

- `[public] hasAccessPermission(userID:String): boolean`
  Liefert `true` zurück, falls der Besitzer mit `uesrID` Zugriffsrechte für diese Ressource hat, und `false` wenn nicht.
- `[public] hasAccessPermission(userID:String): boolean`
  Liefert `true` zurück, falls der Besitzer mit `uesrID` Besitzerrechte für diese Ressource hat, und `false` wenn nicht.

Klasse ResourceType

`<<enumeration>> ResourceType`

Diese ist eine Aufzählungstyp-Klasse, die zur Unterscheidung von Ressourcen unterschiedlicher Typen dient. Enthält beispielsweise folgende Typen:

- Data
- Tool
- Workflow

Klasse Request

"Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods. Subclasses of an abstract class in Python are not required to implement abstract methods of the parent class."

—Abstract Classes | Python Course

`<<abstract>> Request`

Abstrakte Klasse `Request` definiert alle gemeinsamen Attribute und Methoden für alle mögliche Requesttypen. Sie selbst kann jedoch keine eigenständigen Instanzen erzeugen. Diese Klasse dient vor allem zur Erweiterbarkeit des Produkts.

Attribute

- `[final] REQUEST_ID : String`
  Eindeutiger Identifikator des entsprechenden Eintrags in der Datenbank.
- `[final] SENDER_ID : String`
  ID des Absenders (`User`)
- `[final] CREATION_DATE : String`
  Datum der Erstellung.
- `[final] RESOURCE_ID : String`
  ID der Ressource.

Funktionen

- `[abstract] deny(): void`
  Signatur der Methode, die diesen Request ablehnt.
- `[abstract] accept(): void`
  Signatur der Methode, die diesen Request annimmt.

Klasse AccessRequest

`AccessRequest extends Request`

Repräsentation des Zugriffsrequests. Wird in der Datenbank mit von der `Request`-Klasse vordefinierten Attributen gespeichert.

Funktionen

- `[public] deny(): void`
  Diese Funktion lehnt den Zugriffsrequest ab: Instanz der `AccessRequest`-Klasse wird gelöscht und der Absender benachrichtigt.

- `[public] accept(): void`
  Request wird genehmigt: in der Tabelle "Permission" wir neuer Eintrag erstellt mit `this.resourceID`, `this.userID` und Typ "Access".

Klasse DeletionRequest

`DeletionRequest extends Request`

Repräsentation des Löschrequests. Wird in der Datenbank mit von der `Request`-Klasse vordefinierten Attributen gespeichert.

Funktionen

- `[public] deny(): void`
  Diese Funktion lehnt den Löschrequest ab: Instanz der `DeletionRequest`-Klasse wird gelöscht und der Absender benachrichtigt.

- `[public] accept(): void`
  Request wird genehmigt: die Ressource mit `this.resourceID` wird aus der Datenbank gelöscht, sowie alle Rechte und Requests für sie. Die Besitzer werden benachrichtigt.

Klasse Logging

`<<external>> Logging`

Vorimplemetiertes API von Python, dient zum Protokollieren von Systemereignissen:

- Erstellung und Löschung einer Ressource
- Absenden und Stornieren eines Requests
- Absenden einer Benachrichtigung
- Genehmigung und Ablehnung eines Requests
- Änderung der Rechte
- Löschung eines Benutzers

Klasse EmailMessage

`<<external>> EmailMessage`

Vorimplentierte Phyton-Klasse, wird zur Benachrichtigungen der Benutzer des Portals benutzt. Benutzer werden über einige Systemereignisse benachrichtigt:

- Erhalten eines Requests
- Genehmigung und Ablehnung abgesendeter Requests
- Änderung der Rechte
- Löschung einer Ressource, für die der Benutzer Besitzerrechte hatte.

4.2 View

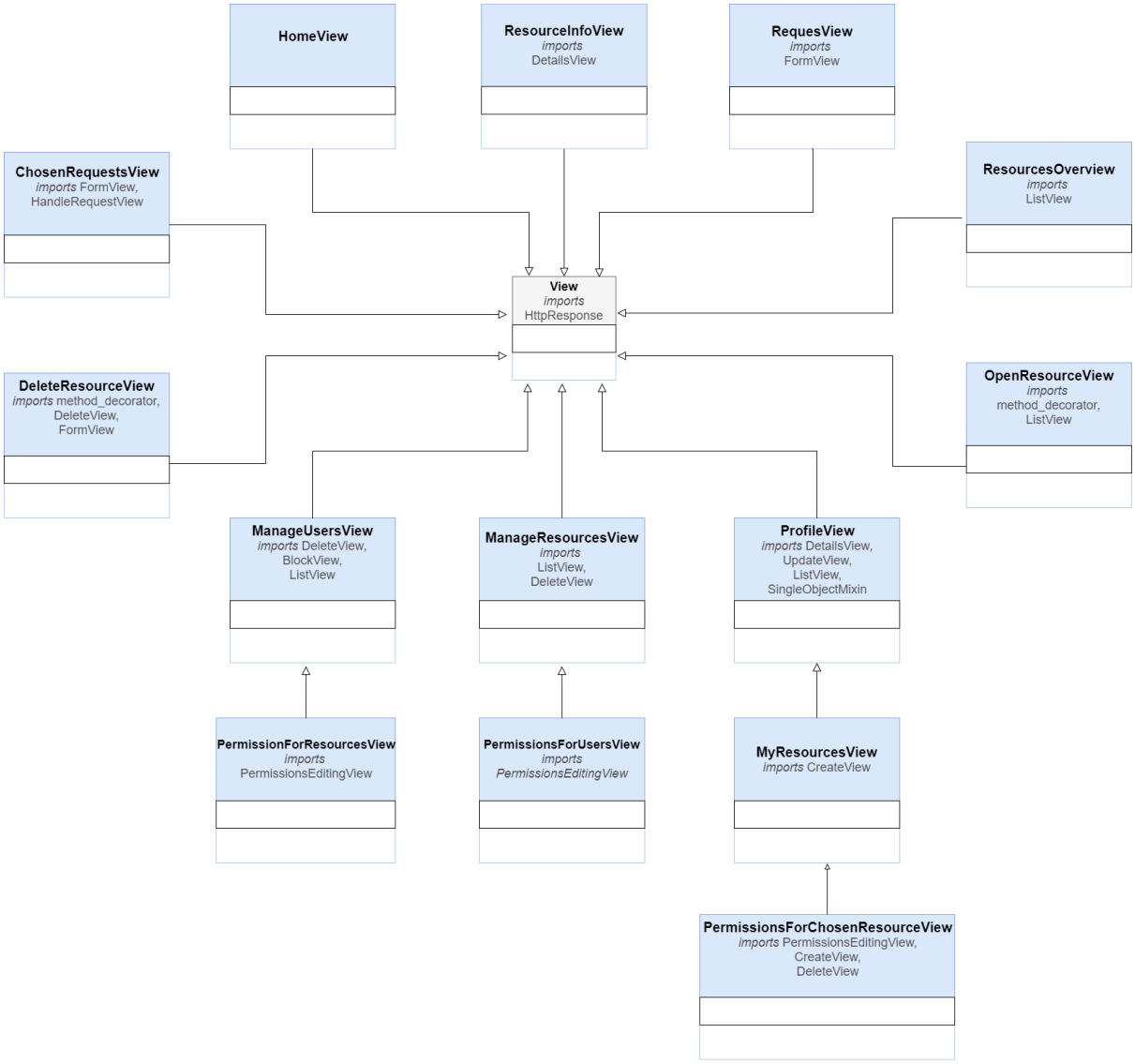

Abbildung 8: Views-Klassendiagramm

Unsere View-Klassen bestehen aus kleineren generischen Klassen, die schon in Django zur Verfügung stehen (django.views.generic; django.utils.decorators). Ausnahmen sind die Klassen HandleRequestView, PermissionsEditingView, BlockView und HomeView.

HomeView Diese Klasse bezeichnet die Homepage, auf der der Benutzer zwischen zwei Subseiten auswählen kann - ”Profile” und ”Resources Overview”.

Klasse ProfileView Diese Klasse bezeichnet die Profilseite des Benutzers.

```
from django.views.generic import DetailsView
```

Zeigt den Namen des Benutzers.

```
from django.views.generic import UpdateView
```

Zeigt ein Feld zur Änderung vom Name.

```
from django.views.generic import ListView
```

Zeigt eine Liste von Requests.

```
from django.views.generic.detail import SingleObjectMixin
```

Benutzt mit ListView um Requests eines spezifischen Benutzers zu zeigen.

Klasse ChosenRequestsView Diese Klasse bezeichnet den Dialog, in dem der Benutzer entscheidet einen Request entweder abzulehnen oder zu genehmigen.

```
from django.views.generic import FormView
```

Zeigt ein Feld zur Beschreibung der Entscheidung.

```
from django.views import HandleRequestView
```

Zeigt zwei Optionen - ”genehmigen” und ”ablehnen”.

Klasse MyResourcesView Diese Klasse bezeichnet die Subeite, auf der der Benutzer seine eigenen Ressourcen sieht.

`MyResourcesView extends ProfileView`

Alle genersiche View-Klassen aus ProfileView werden auch in MyResourcesView benutzt mit dem Unterschied, dass ListView und SingleObjectMixin eine Liste der Ressourcen spezifisch für einen Benutzer zeigen.

`from django.views.generic import CreateView`

Dient zur Erstellung von neuen Ressourcen.

Klasse DeleteResourceView Diese Klasse bezeichnet den Dialog, in dem der Benutzer einen Löschrequest sendet.

`from django.utils.decorators import method_decorator`

Dient zur Prüfung der Adminrechte.

`from django.views.generic import DeleteView`

Wenn der Benutzer Adminrechte hat, dann kann er die Ressource löschen.

`from django.views.generic import FormView`

Zeigt ein Feld zur Beschreibung des Grundes des Löschrequests, wenn der Benutzer keine Adminrechte hat.

Klasse PermissionForChosenResourceView Diese Klasse bezeichnet die Subseite, auf der der Benutzer die Rechte anderer Benutzer für seine eigene Ressourcen editieren kann.

`PermissionForChosenResourceView extends MyResourcesView`

Alle genersiche View-Klassen aus MyResourcesView werden auch in PermissionForChosenResourceView benutzt mit dem Unterschied, dass ListView und SingleObjectMixin eine Liste der Benutzer spezifisch für eine Ressource zeigen.

`from django.views import PermissionsEditingView`

Zeigt zwei Optionen(Checkboxes) - “Zugriffsrechte“ und “Besitzerrechte“.

`from django.views.generic import CreateView`

Dient dem Hinzufügen der Benutzern in dieser Liste(von Benutzer spezifisch für eine Ressource), deren Rechte mit Hilfe von die Klasse PermissionsEditingView editiert werden können.

`from django.views.generic import DeleteView`

Dient dem Löschen von Benutzern aus diese Liste(nur von Benutzern spezifisch für eine Ressource). So kann man Rechte entziehen (nur von Benutzer die keine Besitzerrechte haben).

Klasse ManageUsersView Diese Klasse bezeichnet die Seite, auf der der Administrator alle Benutzer des Systems verwalten kann.

`from django.views.generic import DeleteView`

Dient dem Löschen des Benutzers.

`from django.views import BlockView`

Dient dem Blockieren des Benutzers. Besteht aus einem Feld zur Beschreibung des Grundes des Blockierens und Dropdown zur Eingabe des Zeitraums.

`from django.views.generic import ListView`

Zeigt eine Liste aller Benutzer des Systems.

Klasse PermissionsForResourceView Diese Klasse bezeichnet die Subseite, auf der der Administrator die Rechte von bestimmten Benutzern editieren kann.

`PermissionForResourcesView extends ManageUsersView`

Alle generischen View-Klassen aus ManageUsersView werden auch in PermissionForResourceView benutzt mit demm Unterschied, dass ListView eine Liste aller Ressourcen zeigt.

`from django.views import PermissionsEditingView`

Zeigt drei Optionen - ”Zugriffsrechte”, ”Bestizerrechte”, ”Keine”.

Klasse ManageResourcesView Diese Klasse bezeichnet die Subseite, auf der der Administrator Rechte für bestimmte Ressource editieren kann.

```
from django.views.generic import ListView
```

Zeigt eine Liste aller Ressourcen im System.

```
from django.views.generic import DeleteView
```

Dient zum Löschen einer Ressource. Mit oder ohne Beschreibung.

Klasse PermissionsForUsersView Diese Klasse bezeichnet die Subseite, auf der der Administrator die Rechte für bestimmte Ressource editieren kann.

```
PermissionForUsersView extends ManageResourcesView
```

Alle generischen View Klassen aus ManageResourcesView werden auch in PermissionForUsers-View benutzt mit dem Unterschied, dass ListView eine Liste der Benutzer zeigt.

```
from django.views import PermissionsEditingView
```

Zeigt drei Optionen - "Zugriffsrechte", "Bestizerrechte", "Keine".

Klasse ResourcesOverview Diese Klasse bezeichnet die Subseite, auf der der Benutzer nach Ressourcen suchen kann, die Ressourcen zugreifen, deren Beschreibung lesen, oder Requests senden.

```
from django.views.generic import ListView
```

Zeigt eine Liste der Ressourcen.

Klasse OpenResourceView Diese Klasse bezeichnet die Subseite, auf der die Ressource sich befindet.

```
from django.utils.decorators import method_decorator
```

Dient zur Prüfung der Zugriffsrechte.

```
from django.views.generic import ListView
```

Zeigt eine Liste der Ressourcen, nach Wahl(alphabetisch, neu, enthält "a-z", usw.) sortiert.

Klasse RequestView Diese Klasse bezeichnet den Dialog, in dem der Benutzer einen Request mit Beschreibung für eine Ressource sendet.

```
from django.views.generic import FormView
```

Zeigt ein Feld zur Beschreibung des Grundes des Zugriffsrequests.

Klasse ResourceInfoView Diese Klasse bezeichnet den Dialog, in dem der Benutzer relevante Information über eine bestimmte Ressource sehen kann.

```
from django.views.generic import DetailsView
```

Zeigt relevante Information über bestimmte Ressource (Beschreibung, Erstellungsdatum, Name, Typ, usw.).

5 Sequenzdiagramme

NB: Mit blauer Farbe sind vorimplementierte Methoden von Django markiert.

5.1 Zugriffsrequest absenden

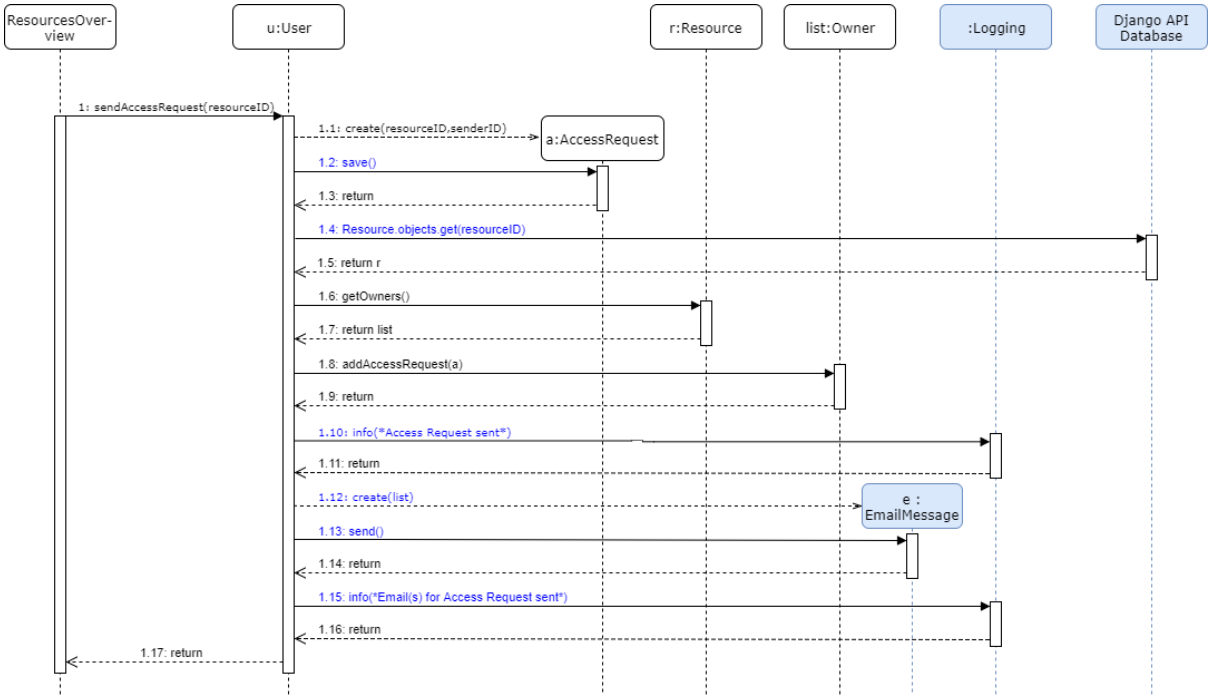

Abbildung 9: Zugriffsrequest absenden

Von der View-Klasse **ResourcesOverview** wird die Methode **sendAccessRequest(resourceID)** des Benutzers „**u**“ aufgerufen. Erstens wird eine AccesRequest-Instanz „**a**“ erstellt mit **create(resourceID, senderID)**-Methode und durch **save()**-Methode in der Datenbank gespeichert.
Danach wird die Ressource, für der der Request erstellt wurde, von der Datenbank geholt und in der Variable „**r**“ gespeichert.
Von der Ressourceninstanz „**r**“ wird dann **getOwners()**-Methode aufgerufen, die alle Besitzer des Ressource liefert. Von allen Instanzen der Besitzer dieser Ressource (auf dem Diagramm als die Variable „**list**“ bezeichnet) wird die Methode **addAccessRequest(AccessRequest)** aufgerufen. Das fügt den Request in der Liste der abgesendeten, aber noch nicht bearbeiteten Requests, hinzu.
Eine passende Log-Nachricht wird in der Log-Datei durch die Methode **info()** gespeichert.
Danach wird eine Instanz der Django-Klasse „**Emailmessage**“ erstellt. Als Parameter wir die Variable „**list**“ übergeben. Damit ist gemeint, dass die Klasse alle Emails von den Besitzern in die Liste kapselt.
Der Konstruktor der Klasse erfordert auch andere Parameter wie E-Mail Nachricht, Sender usw., die im Diagramm nicht bezeichnet sind. Nach dem erfolgreichen Absenden der Emails mit der Methode **send()** wird eine Log-Nachricht in der Log-Datei gespeichert.

5.2 Zugriffsrequest genehmigen

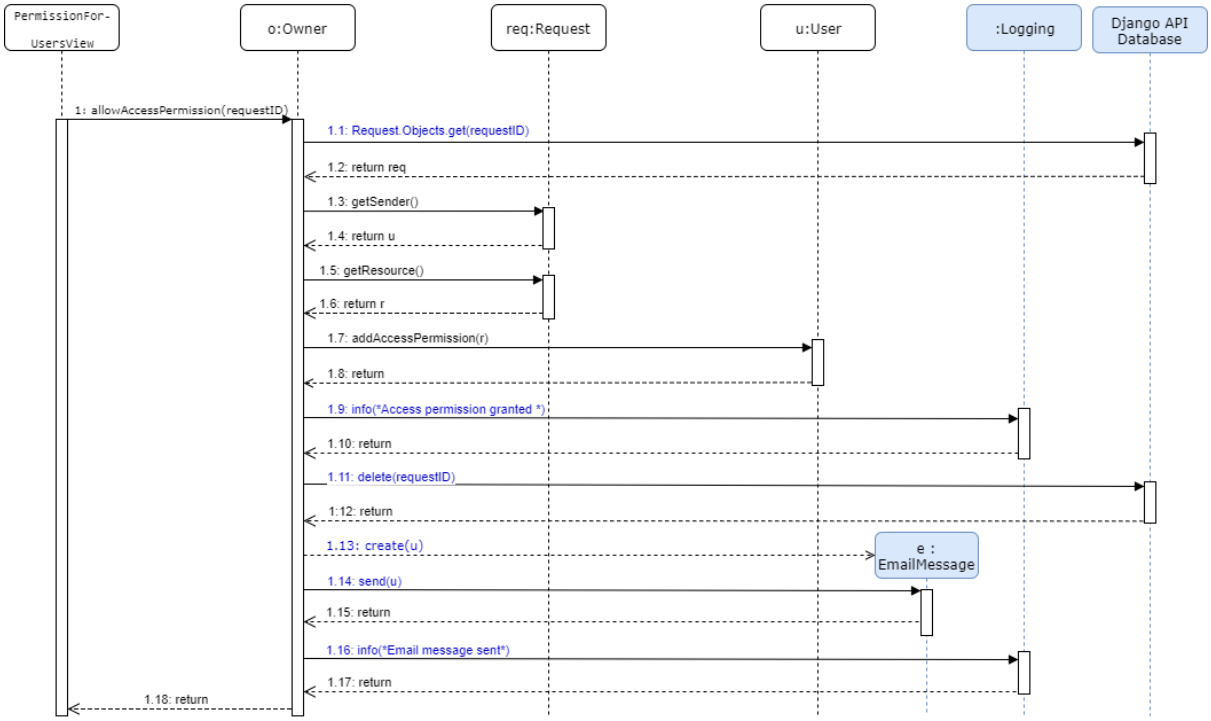

Abbildung 10: Zugriffsrequest genehmigen

Von der View-Klasse **PermissionForUsersView** wird die Methode **allowAccessPermission(requestID)** des Besitzers „**o**“ aufgerufen. Danach wird der Request von der Datenbank geholt und die Variable „**req**“ gespeichert.
Danach wird von „**req**“ die Methode **getSender()** aufgerufen und das Ergebnis(der Benutzer, der den Request geschickt hat) wird dann in „**u**“ gespeichet.
Das Gleiche passiert für das durch den Request angefragte Ressource durch die Methode **getResource()**. Von „**u**“ wird dann **addAccessPermission(resource)** aufgerufen, was intern den Benutzer „**u**“ in die Liste „**accessPermission**“ von der Ressource hinzufügt.
Eine Log-Nachricht wird in der Log-Datei gespeichert.
Die Methode **delete(requestID)** löscht den Request von der Datenbank. Von hier an ist das Diagramm ähnlich dem Diagramm Zugriffsrequest absenden.

5.3 Ressource erstellen

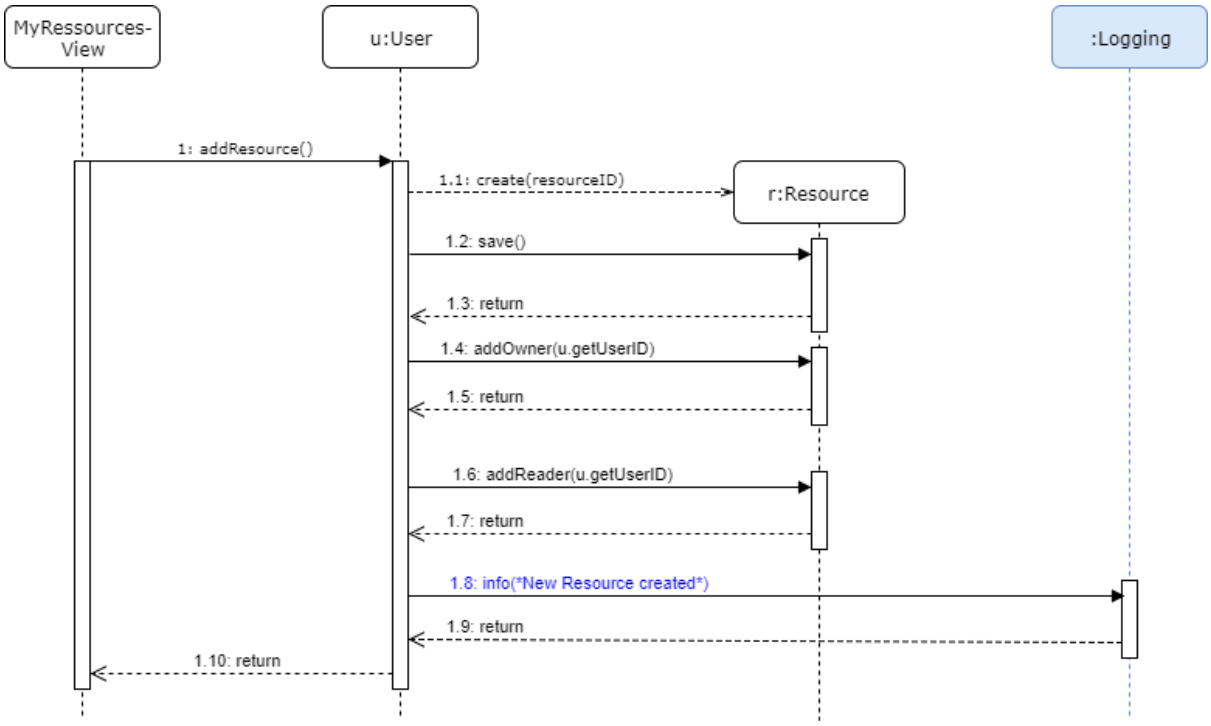

Abbildung 11: Ressource erstellen

Von der View-Klasse **MyRessourcesView** wird die Methode **addResource()** des Benutzers „**u**“ aufgerufen.
Erstens wird eine Instanz der Ressource „**r**“ mit **create(resourceID)**-Methode erstellt und in der Datenbank durch **save()**-Methode gespeichert.
Dann wird das Benutzer „**u**“ zuerst als Besitzer dieser Ressource „**r**“ durch seinen „**UserID**“ mit Aufruf der Methode **addOwner(userID)** hinzugefügt, danach als Leser dieser Ressource „**r**“ durch seinen „**userID**“ mit Aufruf der Methode **addReader(userID)**.
Am Ende wird eine Log-Nachricht über diese Ereignis in der Log-Datei gespeichert.

5.4 Ressource zugreifen

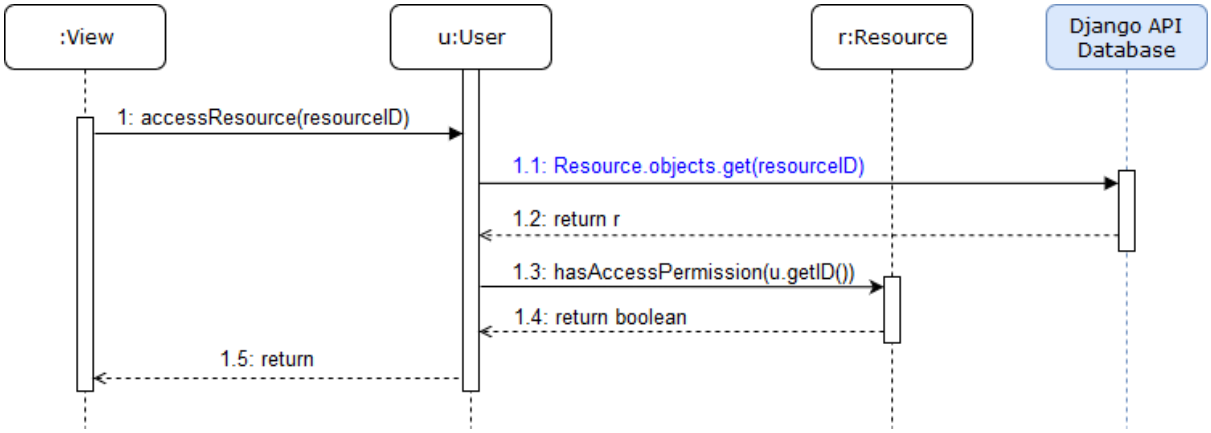

Abbildung 12: Ressource zugreifen

Als Ergebnis einer Benutzeraktion wird von einer View-Klasse die a**ccessRequest(resourceID)**-Methode des Benutzers **u** aufgerufen.
Aufruf der Methode **Resource.objects.get(resourceID)**, die von Django vorimplementiert ist, wird entsprechende Instanz der **Resource**-Klasse **r** liefern.
Mit der Methode **hasAccessPermission(u.getID())** wird überprüft, ob der Besitzer Zugriffsrechte für diese Ressource hat. ID des Benutzers wird dabei als Parameter übergeben. Negatives Ergebnis wird eine *Exception* aufrufen (Ressourcenzugriff soll von View versehen werden, nur wenn eingelogter Benutzer Zugriffsrechte für die Ressource hat).

5.5 Ressource löschen

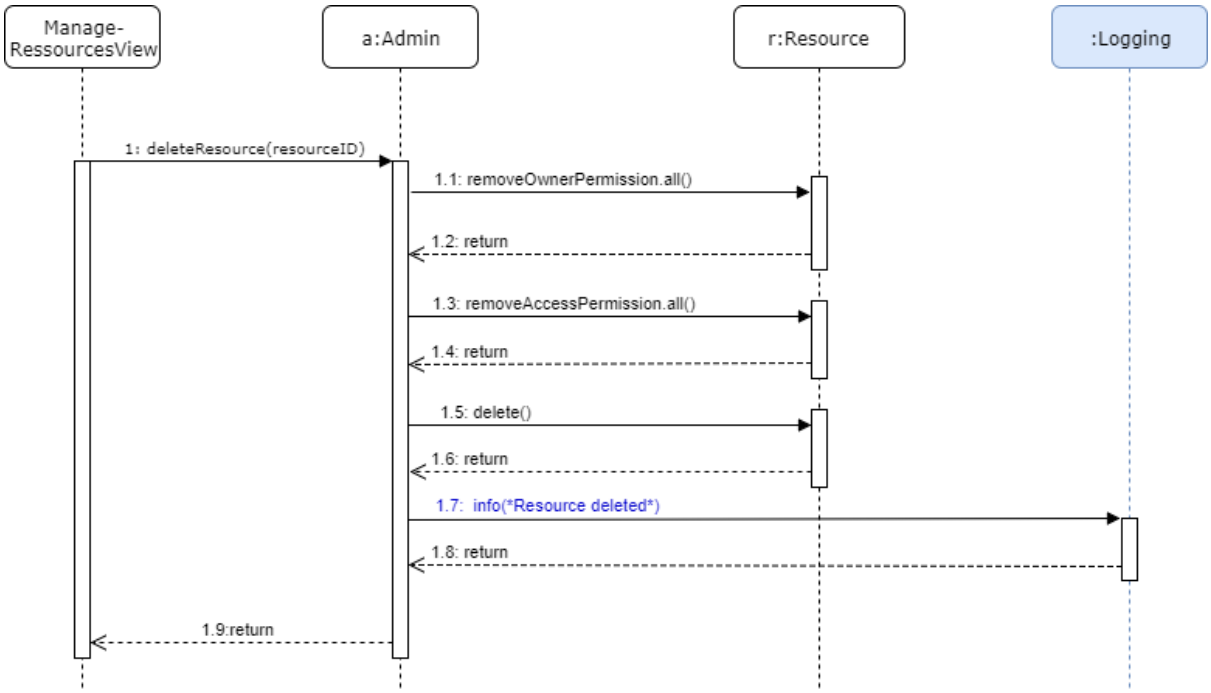

Abbildung 13: Ressource löschen

Von der View-Klasse **Manage-Resource** wird die Methode **deleteResource(ressourceID)** des Admins „**a**“ aufgerufen.
Erstens wird die Liste aller „**Owner**“ dieser Ressource vom Datenbank gelöscht, mit Aufruf der Methode **removeOwnerPermission.all()**, danach die Liste aller Leser diese Resource mit Aufruf der Methode **removeReaderPermission.all()**.
Wenn die zwei Listen von der Datenbank gelöscht sind, dann wird die Ressource „**r**“ auch gelöscht , mit Hilfe der Methode **delete()**.
Am Ende wird eine Log-Nachricht über diese Ereignis in der Log-Datei gespeichert.

6 Anhang

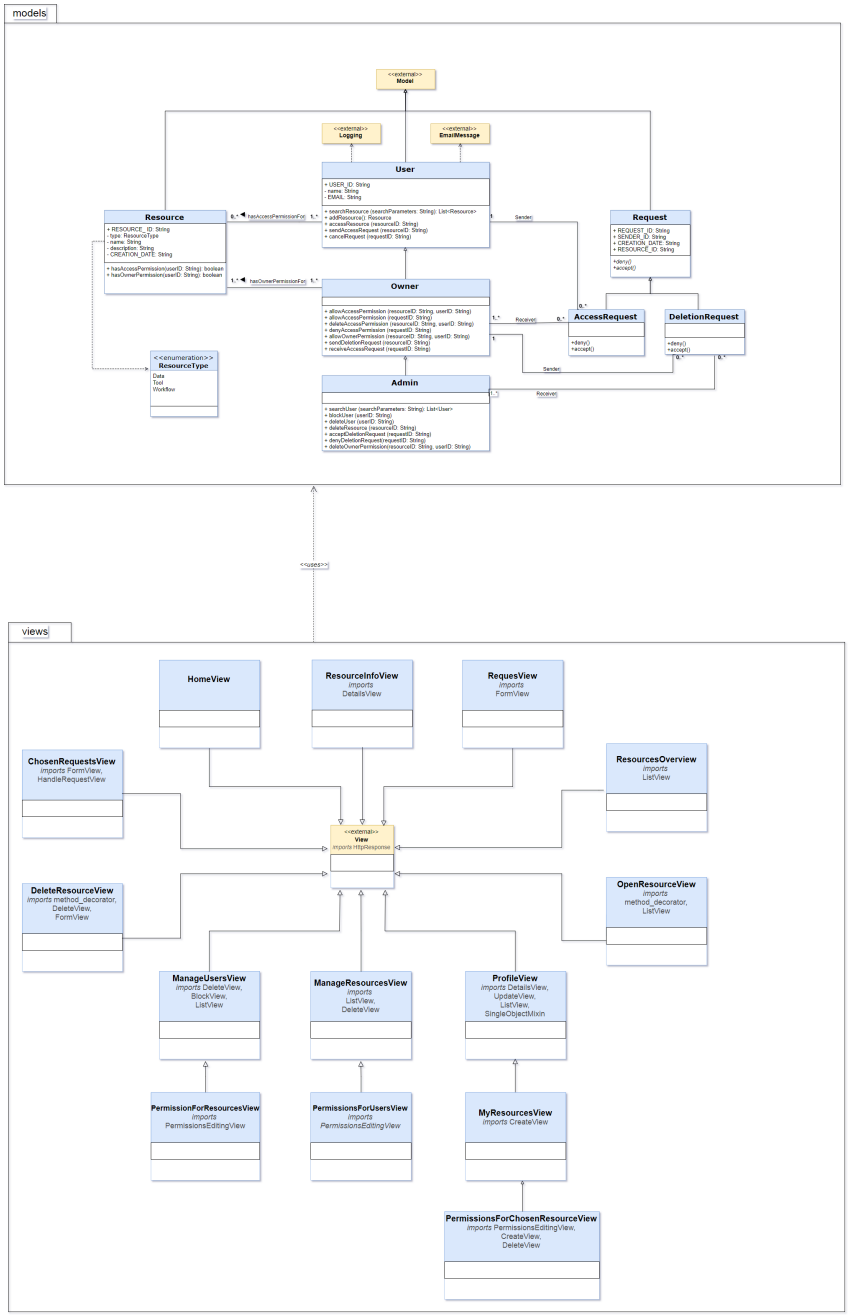

Abbildung 14: Klassendiagramm: Abbildung der Klassen-Struktur des Projekts.

Glossar

Geheimnisprinzip

Vom Innenleben einer Klasse soll der Verwender – gemeint sind sowohl die Algorithmen, die mit der Klasse arbeiten, als auch der Programmierer, der diese entwickelt – möglichst wenig wissen müssen.

GUI

Abk. GUI von englisch **G**raphical **U**ser **I**nterface, ist Grafische Benutzeroberfläche oder auch grafische Benutzerschnittstelle, über die eine Person eine Software kontrollieren kann. Im Idealfall ist sie benutzerfreundlich, so dass die Interaktion auf natürliche und intuitive Weise geschehen kann.

HTML

Abk. HTML von englisch **H**ypertext **M**arkup **L**anguage, auf Deutsch bedeutet dies „Auszeichnungssprache für verknüpften Text“. Sie ist Voraussetzung für die Programmierung und das Design von Webinhalten. Andere Standards wie PHP bauen in erheblichem Maße auf HTML auf.

ORM

Abk. ORM von englisch **O**bject-**R**elational **M**apping, ist eine Programmiertechnik zum Konvertieren von Daten zwischen inkompatiblen Systemen mit objektorientierten Programmiersprachen. Dies erzeugt in Wirklichkeit eine “virtuelle Objektdatenbank“, die innerhalb der Programmiersprache verwendet werden kann.

URL

Abk. URL von englisch **U**niform **R**esource **L**ocator, wird häufig als Webadresse bezeichnet. Diese Adresse wird verwendet, um eine im Web vorhandene Ressource durch eine ASCII-Zeichenfolge zu bezeichnen. Ressourcen können variiert werden (Webseite, Video, Ton, Bild, Animation, E-Mail-Adresse ...).