Design Dokument für TODO

Christian Stricker Alisha Klein David Klopp
Markus Vieth

7. Januar 2016

Inhaltsverzeichnis

Teil II.

LowLevelDesign

Einleitung

In diesem Dokument wird die Implementierung unseres Systems anhand verschiedener Klassen-, sowie Sequenzdiagramme dargestellt. Hierbei wird sowohl auf die Interaktion der verschiedenen Klassen untereinander, wie auch auf die hierfür verwendeten Pattern eingegangen.
Das System, sowie alle Angaben zum System, beziehen sich dabei auf das "Architectural Design Document for TODO“ vom 8. Dezember 2015.

1. Apache und Tomcat

1.1. Warum Apache verwenden?

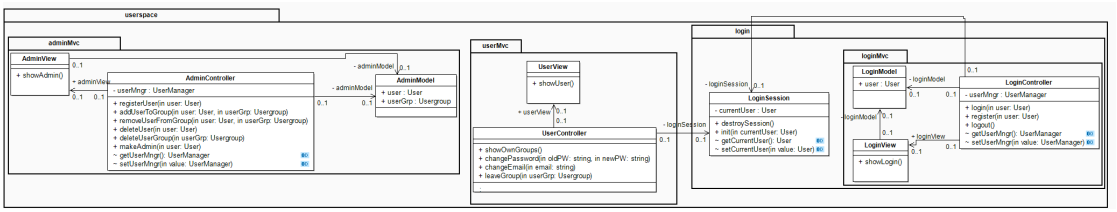
Der Apache HTTP Server ist der meist verbreitete Webserver im Internet. Wir verwenden den Apache HTTP Server, da er quelloffen ist und aktiv weiterentwickelt wird. Durch diese Quelloffenheit ist der Server auf unterschiedlichen Plattformen lauffähig und leicht installier- und konfigurierbar.

1.2. Warum Tomcat verwenden?

Apache Tomcat ist open source und auf Grund seiner Java Implementierung ebenfalls, wie der Apache HTTP Server, plattformunabhängig. Da Tomcat die offizielle Referenzimplementierung für JavaServer Pages (JSP) und Servlets ist, ist dieser stets aktuell und wird dementsprechend aktiv weiterentwickelt. Die Implementierung der Servlets ist unabhängig von der Spezifikation des Webservers und sehr effizient, weshalb sie leicht portiert werden kann. Die Schnittstellen in Tomcat stellen eine klare Trennung zwischen Logik und Sicherheit bereit. Letztere wird durch die Verwendung der JavaVM weiter erhöht. All diese Punkte machen es leichter möglich ToDo effizient und sicher zu implementieren.

2. Präsentationsebene

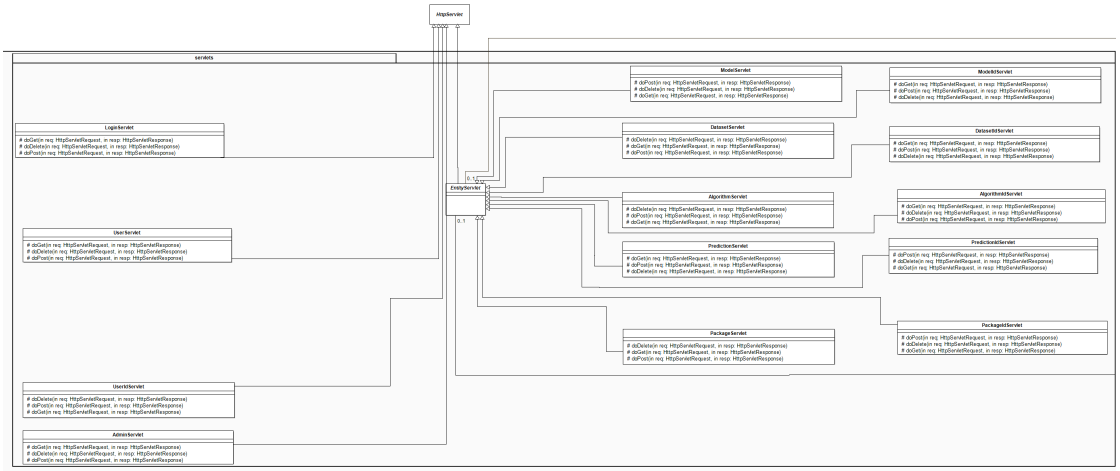
2.1. Nutzerumgebung



Die Nutzerumgebung umfasst den Login-Bereich, sowie den Benutzer- und Adminbereich. Die Bereiche sind nach dem Model-View-Controller-Prinzip entworfen. Alle drei Teile werden über eine View-Klasse repräsentiert. Weiterhin haben alle drei Bereiche jeweils einen Controller. Jeder Controller nimmt die Anfragen der entsprechenden View-Klassen entgegen und verarbeitet diese. Zu den Anfragen gehören im Login-Bereich anmelden oder abmelden oder neu registrieren. Im User-Bereich kann der angemeldete Benutzer seine Angaben ändern, sowie seine Gruppenzugehörigkeit - soweit er die Berechtigung hat - bearbeiten. Ein angemeldeter Administrator kann verschiedene Benutzeroperationen machen, wie Registrieren, Gruppen zuweisen, Benutzer oder -Gruppen löschen sowie neue Administratoren benennen. Der Adminbereich hat eine Model-Klasse, in welcher der aktuell bearbeitete Benutzer bzw. die aktuelle bearbeitete Benutzergruppe zwischengespeichert wird.

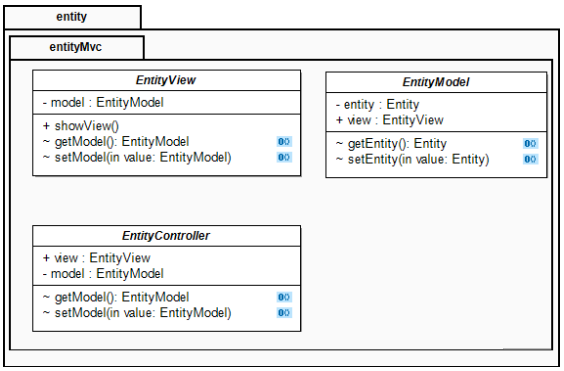
Login-Bereich und User-Bereich teilen sich als Model eine Session-Klasse, welche die Informationen zum aktuell angemeldeten Benutzer hält.

2.2. Servlet



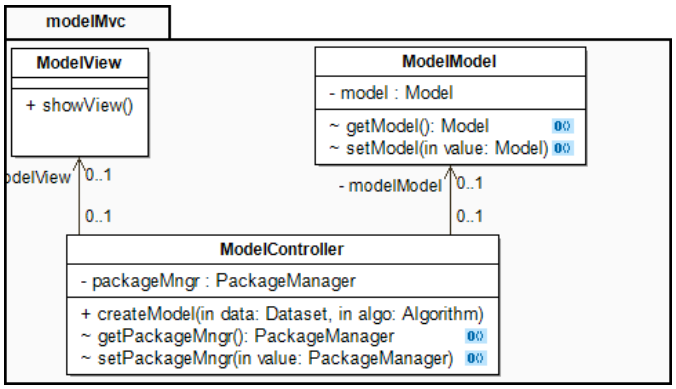
Der Server implementiert verschiedene Servlets, die die Anfragen eines Clients entgegennehmen und beantworten. Hierzu werden die Anfragen des Nutzer ausgewertet und in der Klassenhierarchie an die zuständigen Instanzen weitergeleitet. Zu jedem REST Endpoint den das System bereitstellt existiert ein entsprechendes Servlet, welches von dem javax.servlet.http.HttpServlet abgeleitet ist. Jede Subklasse überschreit hierbei die ”doGet” und ”doPost” Methode, um die gewünschte Operation entsprechend der Art der Anfrage auszuführen. Hierzu kommuniziert die Servlet Klasse mit einer Subklasse des EntityControllers, der nach dem Model-View-Controller Pattern mit dem EntityModel und dem EntityView interagiert. Der EntityView realisiert hierbei den dynamischen Web-Inhalt, der dem Benutzer auf Anfrage angezeigt wird. Für jede verfügbare Ansicht existierten entsprechende Subklassen dieser drei MVC Klassen, die den jeweils benötigten Inhalt bereitstellen.

2.2.1. Abstrakte MVC-Klassen



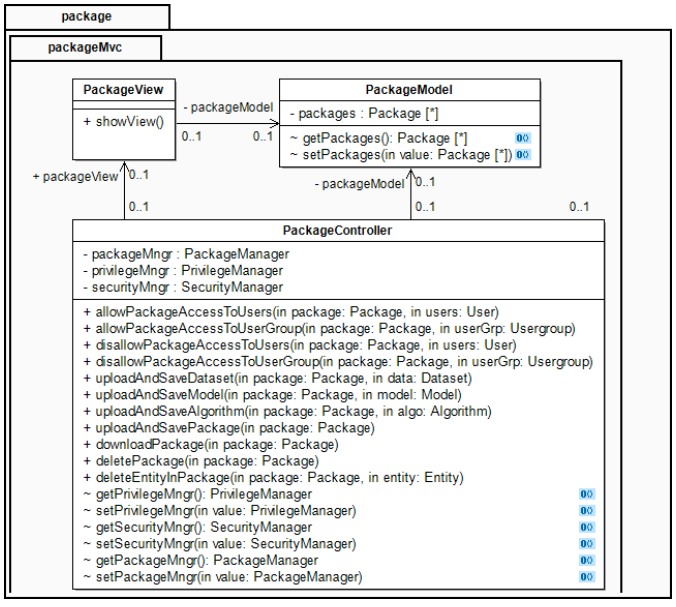
2.3. Entity MVC

2.3.1. Beispiel der Model-Subklassen



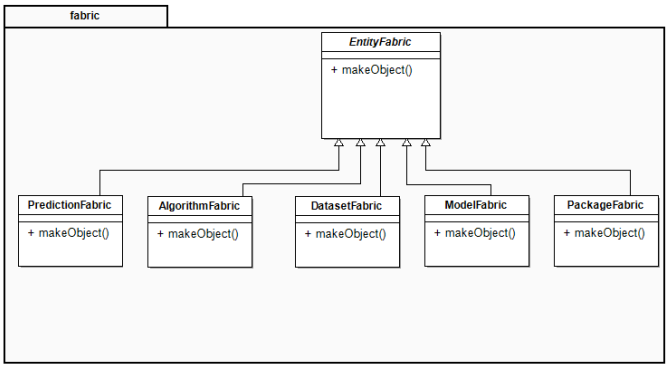
Alle drei Bereiche: Prognosen, Model, und Package, sind wie eben angesprochen nach dem Model-View-Controller-Prinzip strukturiert. Die Views leiten alle von der Entity-View ab (s.Presentation). Jeder Bereich hat einen eigenen Controller, welcher Anfragen durch die View entgegen nimmt und diese an den entsprechenden Handler im Server weiterleitet. Entsprechende Handler sind Package-Manager, Security-Manager und Privilege-Manager. Weiterhin hat jeder Controller und jedeView Zugriff auf das entsprechende Model. Der Package-Controller hat außerdem Zugriff auf die Login-Session. Somit kann der current_user immer direkt dem Handler übergeben werden.

2.3.2. Beispiel der Package-Subklassen



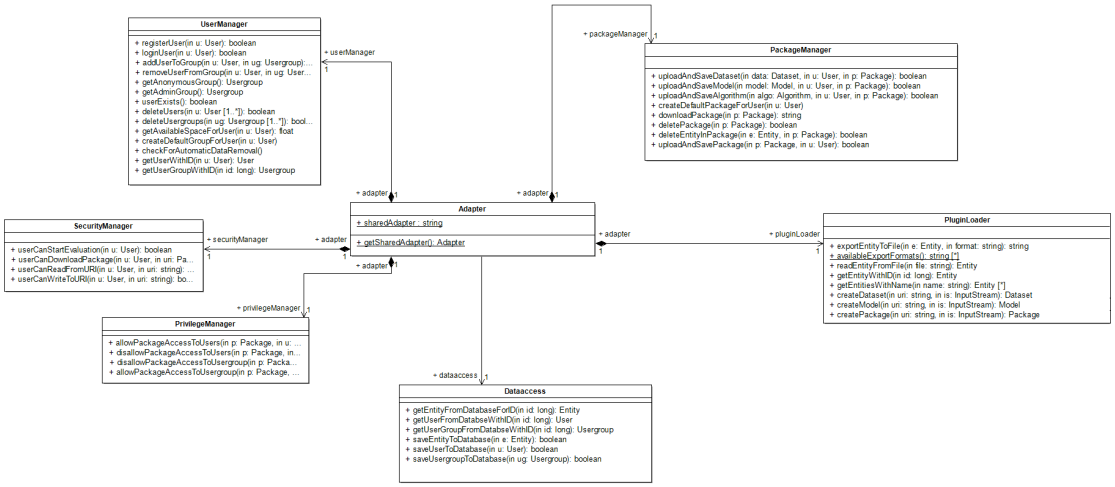
3. Logikebene

3.1. Fabrikmethode



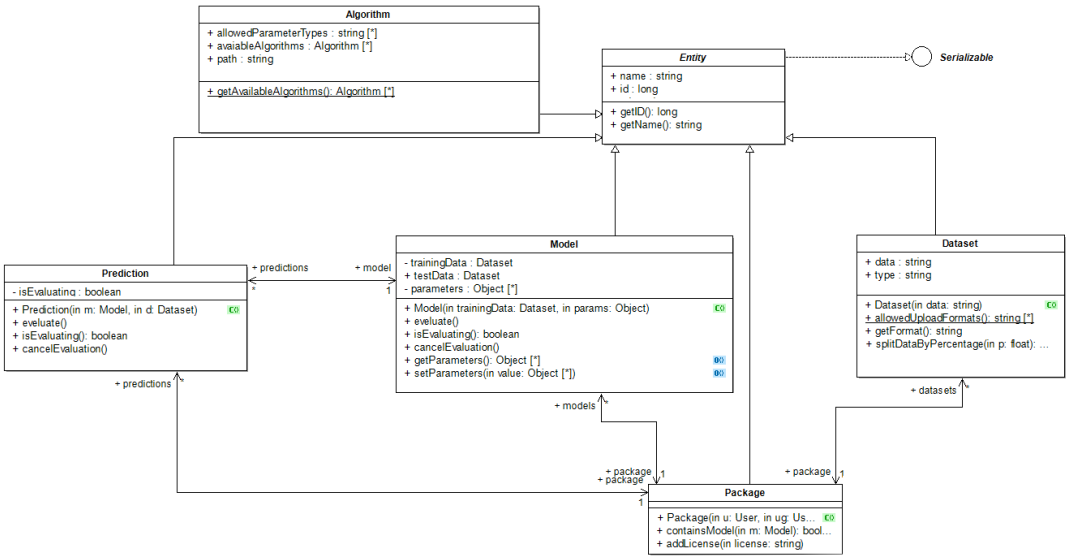
Die Erstellung der bereits ebene erwähnten Model-View-Controller-Struktur der Entitys wird über eine Abstract Factory geregelt. Dieses Pattern sorgt dafür, dass immer der richtige Controller zusammen mit der richtigen View und dem entsprechenden Model erstellt wird.

3.2. Adapter



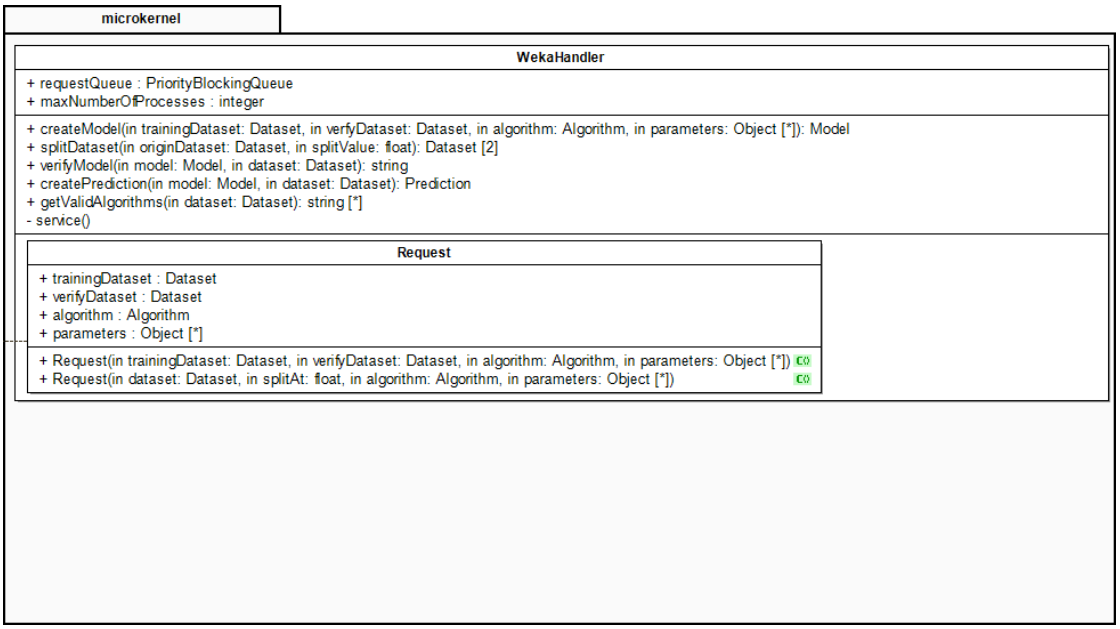
Der Adapter implementiert das Singleton-Pattern, d.h. es existiert immer nur genau eine einzige Instanz der Adapterklasse. Hierbei kennt der Adapter je eine einzige Instanz der jeweiligen Kernkomponenten des Systems. Durch diese Implementierung ist ein Zugriff auf sämtliche dieser Komponenten leicht von verschiedenen Klassen aus möglich. Stellt der Nutzer beispielsweise eine Anfrage an das System, so erhalten die Klassen der Präsentationsebene über den Adapter eine Instanz der jeweils notwendigen Managerklasse, wie z.B. dem UserManager.

3.3. Entity



Jedes Package, Dataset, Model, jeder Algorithm und jede Prediction leiten von der Klasse Entity ab und haben somit einen eindeutige id, sowie uri und einen Namen. Über die Kommunikation mit den darüber liegenden Schichten wird der PluginLoader instruiert je nach Bedarf die jeweilig angeforderten Entities zu erzeugen bzw. abzurufen. Durch die Anbindung an die Weka-Library können die Datsets oder die Models verarbeitet werden, um so neue Predictions zu generieren. Sämtliche administrativen Anfragen die den User oder Usergoups betreffen werden durch den UserManager gehandhabt. Dieser kooperiert über den Adapter mit dem PrivilegeManager, dem PackageManager und dem SecurityManager um die Anfragen zu beantworten. Der Datenzugriff auf die Datenbank erfolgt vollständig über die Klasse Dataaccess, die ebenfalls an den Adapter angebunden ist. Diese Klasse bildet hierbei einen Wrapper um die SPARQL Abfragesprache um die Anfragen an die Datenbank zu formulieren und gewährleistet somit, dass keine gleichzeitigen Zugriffe auf die Datenbank erfolgen können, die zu Beschädigung führen könnten.

3.4. Weka



Der WekaHandler stellt einen Wrapper um die bereits bestehende Weka-Library dar. Dieser Wrapper dient dazu, die von Weka implementierten Datenstrukturen und Methoden, zu den von uns erstellten Klassen kompatibel zu machen. Der Zugriff auf sämtliche für uns relevanten Funktionen der Library kann somit einfacher erfolgen, da ein tiefgehendes Verständnis der Selbigen nur für die Implementierung des WekaHandlers erforderlich ist, nicht aber für die Verwendung der Methoden in den restlichen Klassen.

4. Sequenzdiagramme