Entwurfsbeschreibung – elatePortal

1 Allgemeines

Das elatePortal stellt eine auf Basis von J2EE entwickelte eLearning-Plattform dar, die in Teilen auf dem Quellcode des Apache Jetspeed-2-Portals basiert. Es stellt eine Weiterentwicklung des bereits in der Abteilung für Betriebliche Informationssysteme eingesetzten UebManagers dar, der mit der hohen Anzahl an zu verwaltenden Veranstaltungen nicht mehr skalierte. Die vom Uebmanager übernommenen Funktionen erlauben es bspw. Studenten einer Universität, sich zu registrieren, anzumelden und anschließend in Übungsgruppen einzuschreiben, zu Klausuren anzumelden, sich in einem Forum auszutauschen oder angebotene Dokumente herunterzuladen. Des Weiteren können sich Tutoren und Vorlesende registrieren und Veranstaltungen eintragen sowie Onlineklausuren durchführen.

2 Produktübersicht

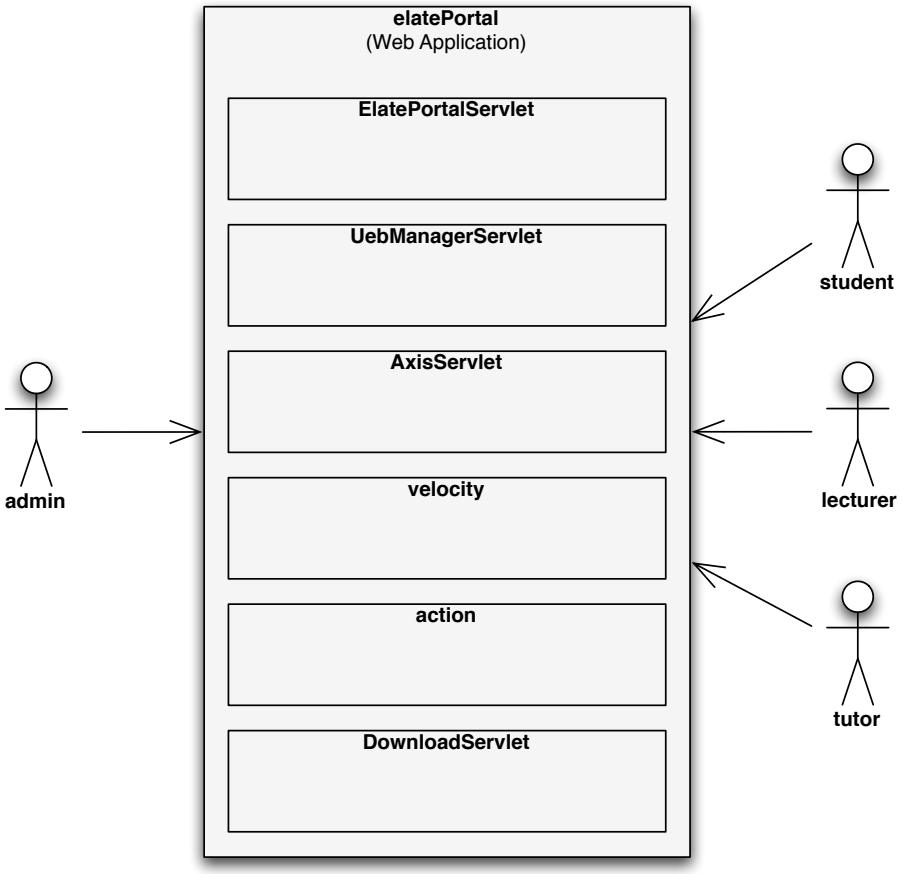
Das elatePortal ist in Form einer J2EE-Webapplikation[1] implementiert, die aus 6 Servlets besteht:

- ElatePortalServlet (das Portal),
- UebManagerServlet (Behandlung der UebManager-Templates im Aufgabenmodell),
- AxisServlet (XML-Datentransfer nach dem SOAP-Protokoll),
- velocity (Template-Compiler),
- action (Portlet Bridge zum Apache Struts-Framawork) und
- DownloadServlet.

Zudem unterstützt das Portal eine robustes Usermanagement, das über Rollen implementiert ist. So kann einfach über die Rechte einen Benutzers am Portal entschieden werden diesem dann in Abhängigkeit seiner Rechte verschiedene Seiten, Portlets und Funktionen zur Verfügung gestellt werden.

Einen Überblick über den Aufbau der Webapplikation aus Sicht des Servlet-Containers gibt folgendes Umweltdiagramm:

[1] siehe Java Servlet Specification 2.4 (JSR 154)



3 Grundsätzliche Struktur- und Entwurfsprinzipien des Gesamtsystems

3.1 Architektur der Web-Applikation

Die Architektur des elatePortals ist in Form einer 4-Schichten-Web-Architektur implementiert. Dabei sind die einzelnen Systemkomponenten wie folgt auf die 4 Schichten (Tiers) verteilt:

Clientschicht (CT): Das Client Tier stellt den Webbrowser des Benutzers dar, mit dessen Hilfe er das vom Portal generierte Markup in Form einer HTML-Seite anzeigen kann. Diese Schicht ist somit die View-Komponente des Systems.

Geschäftsschicht (BT): Die Geschäftsschicht des Systems wird durch die Implementierung der API-Interfaces repräsentiert, wodurch eine Unabhängigkeit zwischen Schnittstelle und deren konkreter Implementierung erreicht wird. Die Implementierung kann einfach ausgetauscht werden, ohne dass durch eine Schnittstellenänderung andere Komponenten ebenfalls einer Änderung bedürfen.

Applikationsschicht (AT): Das Application Tier des Portals wird hauptsächlich durch die einzelnen Portlets und Struts Actions realisiert.

Datenhaltungsschicht (EIT): MySQL-Datenbank, XML, Dateisystem.

3.2 Eingesetzte Frameworks und Technologien

3.2.1 Axis

Apache AXIS ist eine Open-Source-Implementierung des Web Service Standards *SOAP*. SOAP widerum ist ein Protokoll, das Datenaustausch und *Remote Procedure Calls* zwischen Systemen auf XML-Basis erlaubt. Axis wird im elatePortal als Java-Servlet innerhalb des Servlet-Containers betrieben.

3.2.2 Struts

Das Struts-Framework untersützt die Umsetzung des MVC-Patterns mittels des Dreigespanns aus *Servlet* (Controller), *JSP* (View) und *JavaBean* (Model), indem es den Nachrichtenaustausch zwischen diesen Komponenten vereinheitlicht und diese somit austauschbar macht.

3.2.3 Velocity

Zur Realisierung der View-Komponente der Portelts wird in den meisten Fällen die *Java-Template-Engine Velocity* verwendet. Über eine recht einfache Syntax, können dynamisch Textersetzungen im Markup durchgeführt werden. Velocity bietet somit eine Alternative zu den sonst bei Portlets häufig anzutreffenden JSPs.

3.2.4 JAXB

Um bei der (De-)Serialisierung von Java-Objekten nicht auf das eigenständige Parsen von XML-Dateien angewiesen zu sein, wird beim elatePortal das JAXB-Framework verwendet. JAXB steht hierbei für *Java Architecture for XML Binding* und erlaubt das Marshalling (Speichern von Objekten) und Unmarshalling (Binden von XML-Daten in einem Objekt).

3.2.5 OJB

Eine konsistente und einheitliche Speicherung von Objekten in einer relationalen Datenbanke ist nicht ohne Weiteres Möglich. Zu diesem Zweck existieren so genannte *objektrelationale Mapper*, die einzelne Klassen auf Entitätsmengen (Tabellen) abbilden und somit Objekte in einer Datenbank persistent speichern können. Eine Implementierung dieser Technik ist die ObjectRelationalBridge (OJB), wie sie im elatePortal zum Einsatz kommt.

3.2.6 Spring

Das Spring-Framework ist ein Java-basiertes Komponentenframework, das es ermöglicht, komplexe externe Anhängigkeiten einfach zu verwalten. Dadurch ist es möglich einfachere Objekte (sogen. *POJOs*; Plain Old Java Objects) zu entwerfen, die Eingeschränkte Schnittstellen besitzen und somit einfacher wartbar und austauschbar sind.

3.3 Eingesetzte Entwurfsmuster

3.3.1 MVC (Model-View-Controller)

Model-View-Controller (MVC) ist ein Architekturmuster zur Aufteilung von Softwaresystemen in die drei Einheiten Datenmodell (Model), Präsentation/Darstellung (View) und Programmsteuerung (Controller). Im elatePortal findet dieses Muster durch den Einsatz von *Servlets* und Portlets Verwendung. Sowohl Servlets als auch Portlets führen serverseitig eine Anwendung aus, halten serverseitig ihre Daten und leiten ihre Darstellung auf die Clientseite weiter.

3.3.2 Factory

Das Entwurfsmuster *Factory* ist ein Erzeugungsmuster. Dieses Muster definiert eine Schnittstelle zur Erzeugung eines Objekts, wobei es den Subklassen überlassen bleibt zu entscheiden von welcher Klasse das zu erzeugende Objekt ist. Anwendung findet dieses Muster bsp. in der Klasse *ObjektFactory* aus dem Paket *de.elatePortal.db.jaxb*. Dort dient es der Programmierschnittstelle „Java Architecture for XML Binding“.

3.3.3 VO (Value Objects)

Value Objects (VO) Entwurfsmuster werden für den Transfer von Daten zwischen den Schichten einer J2EE-Anwendung eingesetzt. Value Objects Klassen beinhalten ausschließlich Public-Attribute sowie ggf. Get- und Set-Methoden. Die Verwendung eines Value Objects ermöglicht es dem Client mit Hilfe eines einzigen entfernten Methodenaufrufs (Remote Procedure Call) mehrere Attribute auf einmal zu lesen bzw. zu schreiben, was zu einem nicht unerheblichen Zuwachs an Performanz führt. Das Package *de.elatePortal.db.vo* enthält diverse Value Objects Klassen für einen Großteil der Datenhaltung, bsp. *PendingRegistration* für Registierungen oder *Subscription* für Einschreibungen.

3.3.4 DAO

Das *Data Access Objects* (DAO) Entwurfsmuster kapselt Daten- bzw. Datenbankzugriffe von der Anwendung. Der Einsatz diesen Musters minimiert den evtl. Portierungsaufwand beim Wechsel der Art der Datenhaltung. Im Package *de.elatePortal.db.dao* befindet sich die Data Access Objects API des elatePortals in Form einer Factory-Klasse *DAOFactory* und diversen Interfaces für den Zugriff auf Registrierungen, Einschreibungen usw.

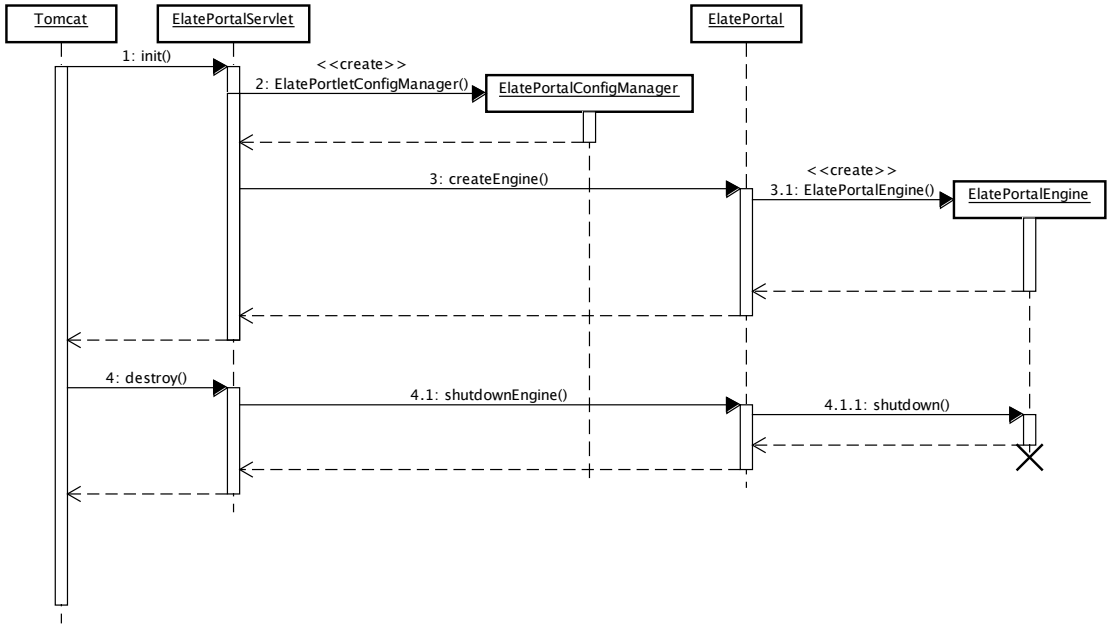
4 Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1 elatePA

4.1.1 Portalapplikation (Paket: de.elateportal.engine.*)

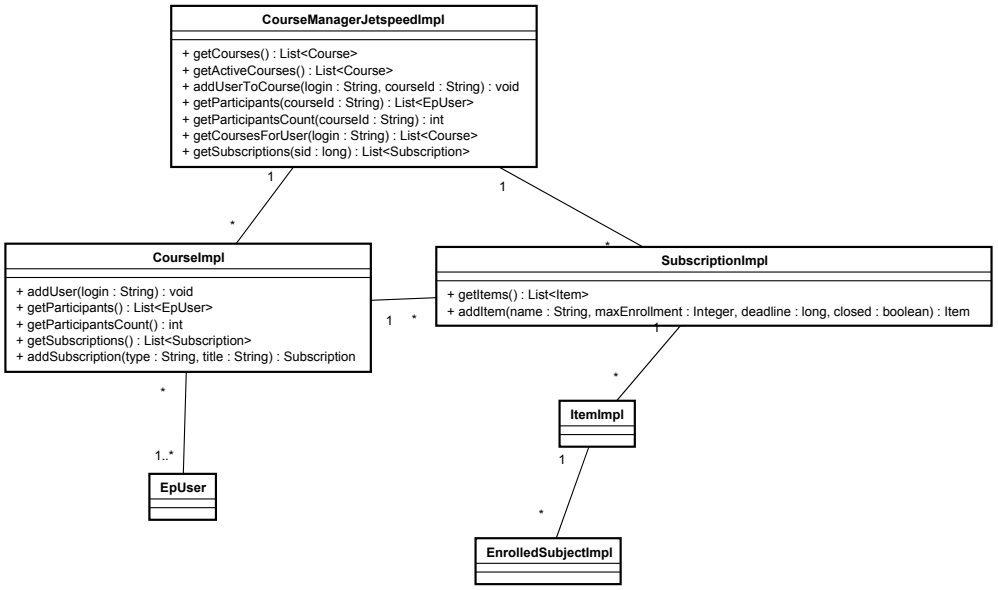
Die Kenrkomponente des Systems stellt ein Servlet dar, welches für die Portalfunktionalität und somit den Portletcontainer verantwortlich ist. Diese Komponente ähnelt in vielen Teilen

dem Jetspeed-Portal und weist auch einen ähnlichen Lebenszyklus auf (siehe Sequenzdiagramm). Dabei wurde die Grundstruktur des Portals beibehalten, große Teile des Quellcodes jedoch neu implementiert oder zumindest basierend auf den Jetspeed-Quellen erweitert (abgeleitet). So sind sämtliche direkte Aufrufe von Jetspeed-Methoden auf 8 Klassen beschränkt, die von den anderen Klassen über ein portalunabhängiges Interface angesprochen werden. Diese bilden somit ein Abstraktionsschicht zwischen den Klassen des elatePortals und Jetspeed, so dass ein eventueller Austausch des Portals oder Änderungen an dessen Methoden wiederum nur Änderungen dieser 8 Klassen nötig machen.



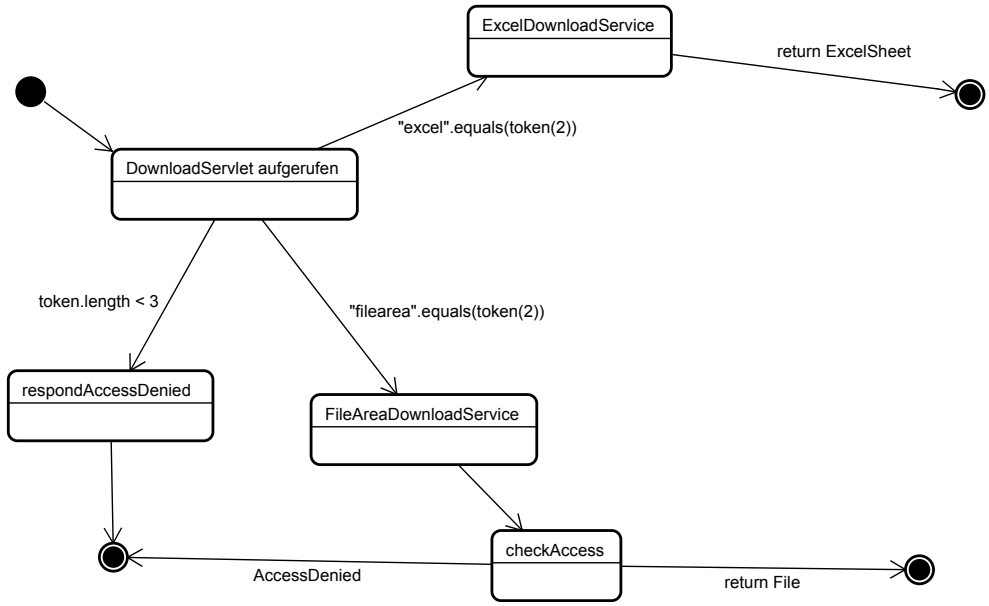
4.1.2 Komponenten (Paket: de.elateportal.components.*)

Dieses Paket liefert einige Managerkomponenten. Der CourseManager ist dafür zuständig, Lehrveranstaltungen und Einschreibungen darin zu verwalten. So bietet er Methoden zur Abfrage der in einer Lehrveranstaltung eingeschriebenen Studenten (`getParticipants()`). Weiterhin führt er die Einschreibungen neuer Studenten in eine Lehrveranstaltung durch (`addUserToCourse()`). Außerdem kann man für jeden Studenten auch dessen Einschreibungen abfragen (`getCoursesForUser()`). Ihm unterstellt sind die Klassen `CourseImpl` und `SubscriptionImpl`. `CourseImpl` stellt eine Lehrveranstaltung dar, welche durch eine `CourseId` eindeutig identifizierbar ist. Das Interface `Course`, auf dem `CourseImpl` aufbaut, bietet Methoden um Benutzer zu einem Kurs hinzuzufügen, eine Liste der Teilnehmer oder der Einschreibungen abzufragen, neue hinzuzufügen und bestehende zu löschen. Eine Subscription ist eine Einschreibung in eine Lehrveranstaltung. Sie enthält die Id der Lehrveranstaltung, die zugehörige Lehrveranstaltung (CourseId).



Weiterhin gehört `PsmlTreeBuilderImpl` zum Package `components`. Er baut anhand der in der Datenbank eingetragenen Lehrveranstaltungen eine psml-Datei auf.

Das `DownloadServlet`, welches ebenfalls im Paket zu finden ist, liefert Dateien zurück, `ExcelDownloadService` liefert Excel-Tabellen mit den Subscriptions der Lehrveranstaltungen zurück. Siehe dazu folgendes Zustandsdiagramm:



Die Komponente `UserRegistrationManagerImpl` verwaltet Neuanmeldungen. Jeder Student kann sich durch Angabe von Login, Name, Vorname, Email und Passwort registrieren.

Die Komponente `LostPasswordManagerImpl` bietet Studenten welche ihr Passwort vergessen haben, eine Möglichkeit das ElatePortal wieder zu nutzen. Dazu wird ein neues

zufälliges Passwort an die Email des Studenten geschickt mit welcher er sich zu Beginn registriert hat.

`NewsManagerXMLImpl` ist für die Verwaltung der Nachrichten verantwortlich. Es können dabei neue Nachrichten erstellt werden, bestehende editiert oder gelöscht werden.

Die Komponente `EpUserManagerJetspeedImpl` ist für die Userverwaltung verantwortlich. Sie baut hauptsächlich auf dem UserManager von Jetspeed-2 auf.

4.1.3 Portlets (Paket: de.elateportal.portlets.*)

Dieses Paket kapselt allgemein die Implementierung der Portlets und eng verbundener Funktionen. Es ist also für Teile des Views des elatePortals im Sinne des MVC-Musters verantwortlich.

Das Paket wurde an Rollen/Rechten sowie wichtigen Funktionen orientiert und enthält für die Rollen Admin, User sowie für Registrierung, Kurse und News extra Pakete. Die enthaltenen Klassen sind von `CourseVelocityPortlet` abgeleitet.

Die Ansicht des elatePortals setzt sich aus den einzelnen Portlets zusammen die mittels Portlet-API in Jetspeed eingebunden sind (das elatePortal besitzt demnach mehrere Ebenen, die in Jetspeed integriert sind). Die dynamische Markup-Erzeugung geschieht in den meisten Portlets über die Java-Template-Engine *Velocity*..

4.1.4 Datenhaltung (Paket: de.elateportal.db.*)

Die Datenhaltung im elatePortal ist durch drei unabhängige Komponenten realisiert:

- MySQL-Datenbank,
- XML-Dateien und
- Dateien im Dateisystem.

Die Anbindung der MySQL-Datenbank erfolgt über das OJB-Framework, welches dafür zuständig ist, Java-Objekte konsistent in einer relationalen Datenbank zu speichern. Gleichzeitig werden hierdurch eine physikalische als auch eine logische Datenunabhängigkeit erreicht, was eine einfache Austauschbarkeit der Datenbank und Änderbarkeit des Schemas erlaubt (eine Schemaänderung hätte allerdings auch Änderungen in der Mapper-Datei von OJB zur Folge).

Der Zugriff auf XML-Dateien ist über das JAXB-Framework gelöst. Ohne XML-Parsing können so Objektdaten persistent (marshalling) gemacht bzw. XML-Daten an Objekte gebunden (unmarshalling) werden.

Dateisystemknoten werden in der Klasse `FSNode` gekapselt, welche einen abstrakten Zugriff und damit eine Unabhängigkeit bspw. von verschiedenen Pfadangaben auf UNIX- oder Windows-Systemen erreicht.

4.1.5 Hilfsklassen (Paket: de.elateportal.util.*)

Die Klassen im Package de.elatePortal.util.* stellen Mechanismen zum betreiben eines Forums bereit. Dies beinhaltet Datumsverarbeitung, Passwörter und Rechteprüfung, Parser und Syntaxüberprüfung und anderes.

Die Klassen sind streng nach Funktionalität getrennt implementiert. Diese Kapselung ermöglicht es, einzelne Utilities gezielt einzusetzen, ohne großen Overhead zu erzeugen.

Die Integration erfolgt meist durch die Aufrufende Klasse, selten benutzen die Util-Klassen selbst das Apache-Interface wie z. B. `LoggerImpl.java` oder `MailUtil.java` das Spring-Framework.

4.1.6 UebManager (Paket: de.thorstenberger.uebman.*)

Der Einsatz des UebMangers im elatePortal dient hauptsächlich dem Zweck, das für ihn entwickelte Aufgabenmodell weiter zu nutzen. Die Integration des UebManagers in das Portal ist nicht sehr weit fortgeschritten. Beim Aufruf einer Klausur bspw., wird einfach die Ansicht zum UebManager umgeschaltet, was der Nutzer auch deutlich zu sehen bekommmt. Da zu einem späteren Zeitpunkt das Aufgabenmodell ins Portal integriergt werden soll, wird die parallele UebManager-Instanz in zukünftigen Versionen des Portals nicht mehr benutzt.

4.2 API (Paket: de.elateportal.*

Die API des elatePortals befindet sich im Paket `de.elateportal` und ist in folgende Pakete untergliedert:

1. `components` – Hier befinden sich Schnittstellen, die für Verwaltungsfunktionen, wie das Registrieren von Benutzern benötigt werden.
2. `db` – Dieses Paket enthält Schnittstellen, die für die persistente Datenspeicherung genutzt werden.
3. `generic` – Enthält mehrere Klassen, welche die `Exception`-Klasse erweitern. Somit ist dieses Paket für die Fehlerbehandlung im System verantwortlich.
4. `log` – Enthält ein Interface für Protokollierungen im elatePortal.
5. `om` (Object Model) – Dieses Paket enthält Schnittstellen für die Geschäftslogik des Portals.

Das elatePortal nutzt eine eigene API, um die Schnittstelle zu Jetspeed 2 und den von diesem bereitgestellten Diensten sehr dünn zu halten. Dies gewährleistet eine gute Portierbarkeit in andere JSR-168 kompatible Portale. Würde diese Abstraktionsebene fehlen und die Klassen direkt auf die Dienste von Jetspeed 2 zugreifen, so würde eine Portierung stark erschwert, da dann eventuell nicht klar wäre wo genau Dienste der Jetspeed-API genutzt werden. Ein Beispiel dafür ist das Interface `UserRegistrationHandler` aus dem Paket `de.elatePortal.components.core.userregistration`. Erst die Implementierung dieses Interfaces durch die Klasse `JetspeedUserRegistrationHandlerImpl` bindet den Registrierungsprozess an die Jetspeed-API.

Des Weiteren bleibt das elatePortal durch diese API erweiterbar und skalierbar, da Implementierungen bestimmter Funktionen, die durch ein Interface spezifiziert sind leicht geändert oder gar ausgetauscht werden können, solange sich an den Schnittstellen selbst nichts ändert.

Da die elatePortal-API von vielen Klassen des Portals, als auch von Portlets genutzt werden kann, wird sie als jar-Paket in den Ordner für `shared-lib` deployt.

Eine Übersicht über die wichtigsten Klassen der elatePortal-API liefert das folgende Klassendiagramm: