

Software Engineering Übung, LVNr:

Übungsleiter: Dr. Sabri Pillana

Designmodell v.1.0

Projekttitel: pandabay

Projekthomepage: <http://pandabay.googlecode.com/svn/trunk/>

Gruppenmitglieder:

MatNr:	Nachname:	Vorname:	e-mail:
0851656	Brückler	Wolfgang	a0851656@unet.univie.ac.at
1106868	Oppermann	Michael	a1106868@unet.univie.ac.at
1030657	Zörényi	Kristóf	a1030657@unet.univie.ac.at

Erstellen sie ein Designmodell gemäß *Unified Process* das zumindest folgende

Aspekte umfasst:

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

1 Klassendesign

Durch die Verwendung des Spring Web Frameworks wird die grobe Klassenstruktur bereits vorgegeben, nämlich in Form des MVC-Pattern für Webanwendungen.

In der Anwendung wird die Präsentationsschicht von den Daten getrennt. Das heißt für unser Projekt: JSP sorgt rein für die grafische Benutzeroberfläche (ohne Business Logic) und die Datenübermittlung zwischen dieser View und der Datenbank (Model) wird über den jeweiligen Controller gesteuert.

Unsere Applikation wird folgendermaßen eingeteilt:

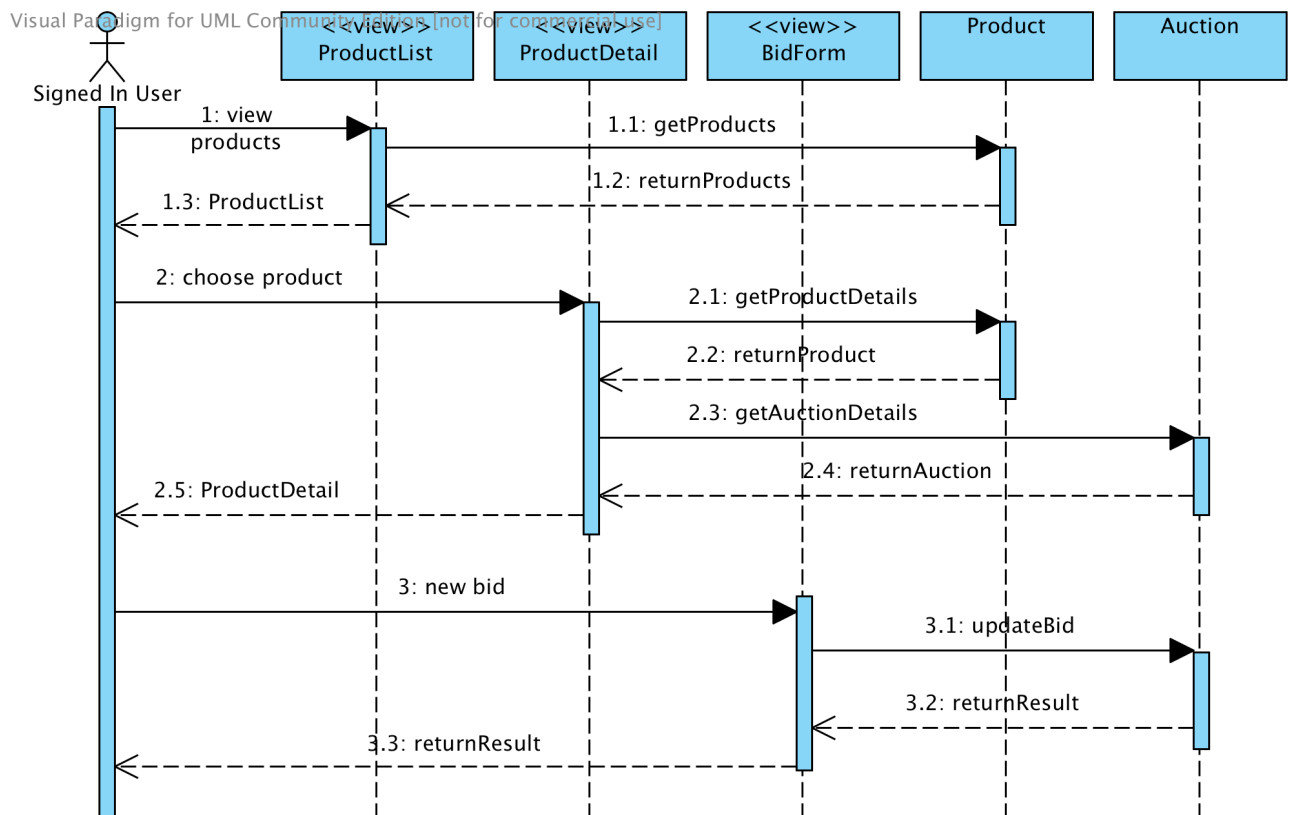
- **Model:**
 - Produkt (alle Attribute, die ein Produkt auf pandabay haben kann, und die dazugehörigen Get-/Set-Methoden)
 - Auktion (beinhaltet Daten, die nicht direkt zum Produkt gehören, eher zum Auktions-Workflow, z.B. Laufzeit, höchstes Gebot, etc., ein Vorteil dieser Trennung ist, das man zu einem späteren Zeitpunkt mehrere Produkte zu einer Auktion zusammenfassen kann)
 - Benutzer (Zugangsdaten, Adresse, usw. - alles was für ein Benutzerprofil wichtig ist - und wieder Getter/Setter, um auf diese privaten Variablen zugreifen zu können)
- **View:** zeigt die Daten des Models in einer benutzerfreundlichen Web-Applikation. Das GUI wird mittels JSP realisiert:
 - Login
 - Register
 - Home
 - ListProduct
 - DetailProduct
 - NewProduct
 - NewBid
 - uvm.
- **Controller:**
 - NewUserController
 - NewProductController
 - NewBidController
 - LoginController
 - RegisterController
 - uvm.

Um die Business Logic von den eigentlichen Daten zu trennen, greifen wir noch auf das sogenannte DAO Pattern (Data Access Object) zurück. Dabei lagern wir die Datenzugriffslogik (z.B. von Produkt oder User) in eine separate Klasse aus. Die eigentliche Model-Klasse ist nicht mehr von einer bestimmten Implementierung (Datenbankzugriff) abhängig. In unserem Fall gibt es dann austauschbare JdbcDao-Klassen, in denen die Implementierung auf eine MySQL-Datenbank erfolgt. Um die Abhängigkeiten zu minimieren, führen wir noch Interfaces für die jeweiligen Dao-Klassen ein.

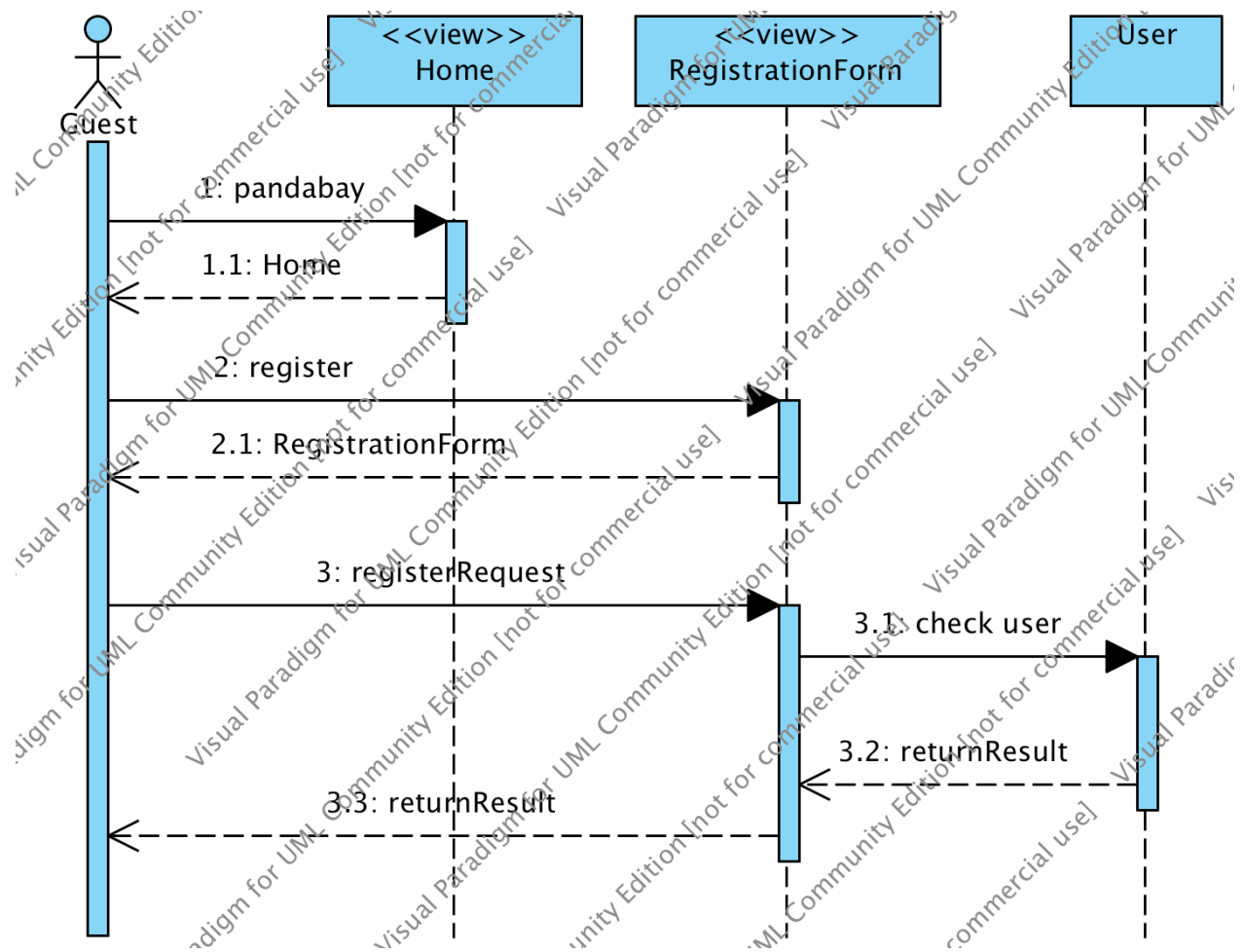
Durch das zentrale DispatcherServlet (Spring Framework) wird der MVC-Teil vom Rest der Applikation getrennt. Die Implementierung erfolgt in der Serverkonfiguration ([web.xml](#)).

2 Use Case Realization Design

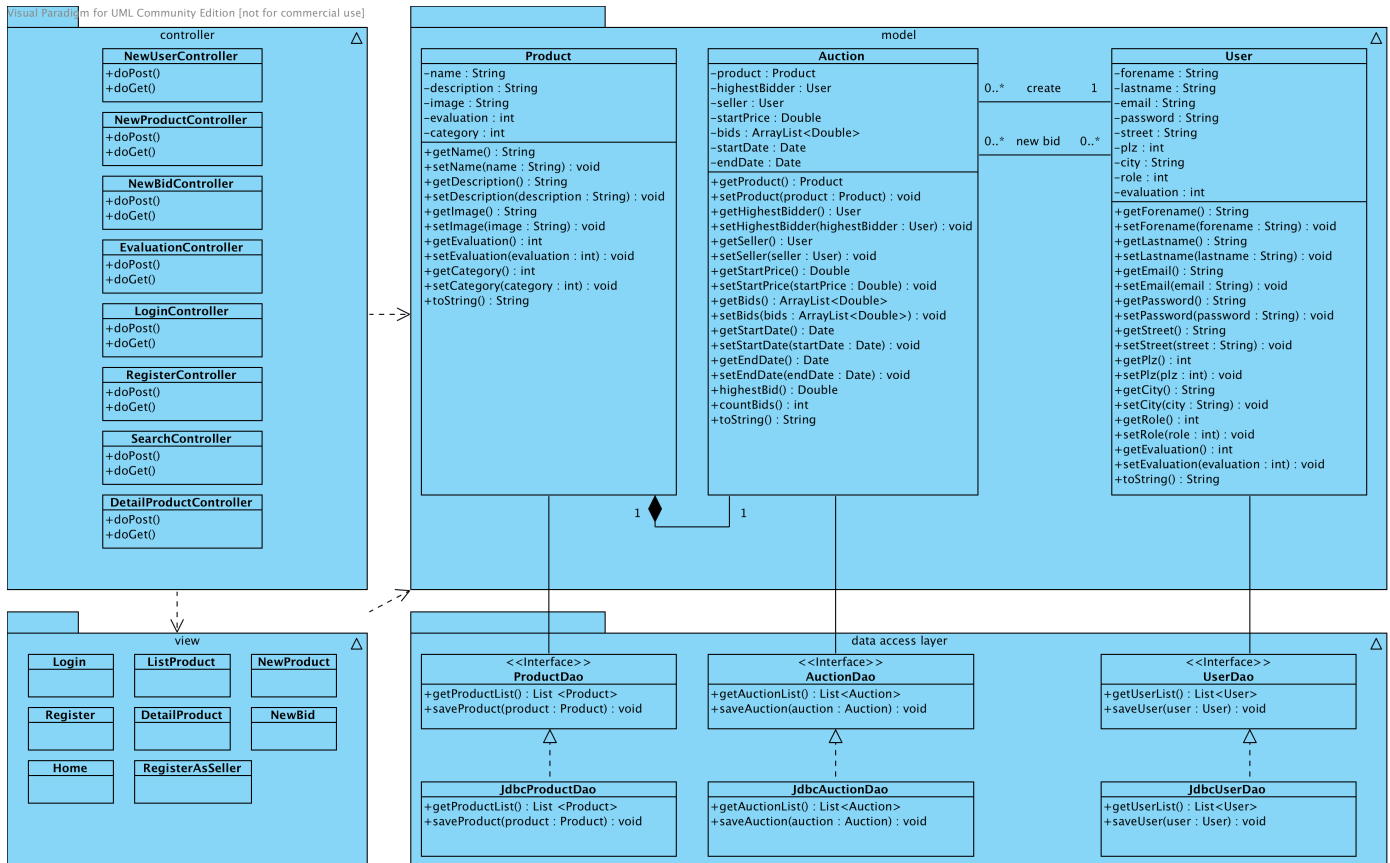
Auswahl eines bestimmten Produkts und neues Gebot.



Ein Gast registriert sich auf der Auktions-Plattform.



3 Übersichtsklassendiagramm

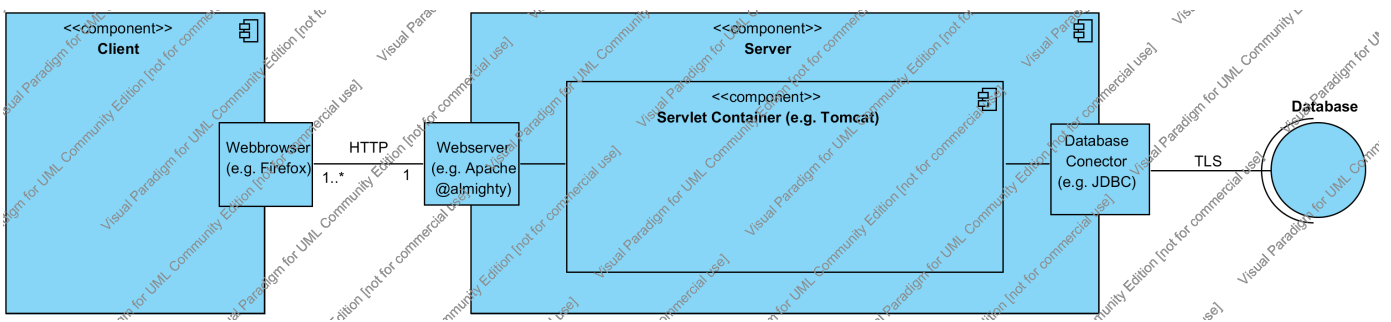


4 Architekturbeschreibung

Die genauen Spezifikationen der Architektur werden durch das **Spring MVC Framework** vorgegeben daher ist hier nur eine Übersicht der logischen Komponenten:

- Ein **Client** spricht den Webserver über den Browser an. Es wird die benötigte Seite mittels http requests angefordert und die jeweilige View (html) via http-responses zurückgegeben.
- der **Webserver** gibt den request mit Hilfe des Deployment Descriptors (xml) an das jeweilige Controller Servlet weiter.
- das **Controller Servlet** verwendet ein POJO aus dem Model um die logischen Operationen durch zu führen und lässt mittels JSP dynamisch eine View für den Web Server erstellen.
- das **Plain Old Java Object** hat Methoden zum Verbinden in die Datenbank, falls z.B. persistente Datenspeicherung benötigt wird. Vom Spring Framework wird hier Java Database Connectivity vorgegeben.

Komponentendiagramm:



Verteilungsdiagramm:

