Architektur

Es gibt 2 Grundlegende Schichten.

Die View ( Frontend ) und das Backend. Diese sind in 2 getrennten Projekten untergebracht.

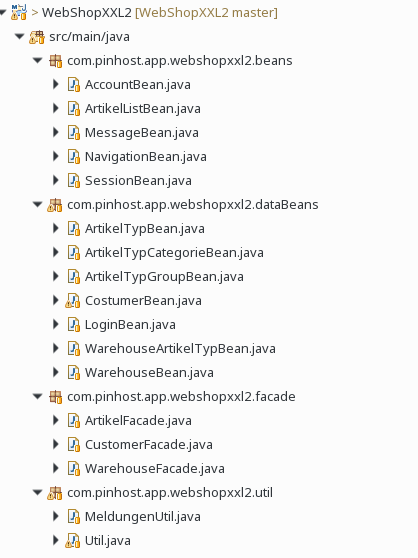
Das Frontend ist im Projekt **WebShopXXL2** .

Das Backend wird zur zeit durch die Ordner

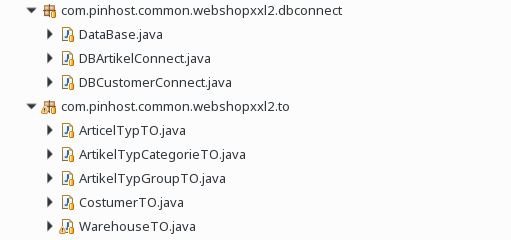
**com.pinhost.common.webshopxxlw2.db**  und **com.pinhost.common.webshopxxlw2.to**

simuliert.

Im Frontend gibt es folgende Struktur:



Und im Backend folgende Struktur



**Die Schnittstelle zwischen Backend und Frontend:**

Die Schnittstelle zwischen Backend und Frontend wird mittels einer Facade defininiert.

Im Frontend gibt es dazu zu jedes Datenobjekt eine eigene Facade die von der Datenbank TO Objekte erwartet und die Daten dieser in die dazugehörigen Beans transferiert.

Dazu muss das Backend auf Anforderung der Facade diese TO aus den unterschiedlichen Quellen zusammen setzen und an die Facde des Frontend liefern. Dabei ist es für das Frontend unerheblich, woher diese Daten kommen. Das Frontend nimmt diese TO und baut daraus die Beans für das Frontend.

Umgekehrt liefert das Frontend Beans an die Facade die ihrerseits die Daten der Beans in TO Objekte verpackt und diese dann an das Backend zurück sendet.

Die Grundlegende Definition der TO liefert das Frontend. Im Falle das das Backend weitere Angaben benötigt, muss das Backend diese in die TO einarbeiten.

Die Packages im einzelnen :

***View***

***com.pinhost.app.webshopxxl2.beans***

Beinhaltet alle Beans die an den View ( xhtml ) Seiten hängen und die sichtbaren wie auch die steuernden Daten der View halten.

**Die Beans :**

|  |  |  |
| --- | --- | --- |
| **Name** | **Scope** | **Erläuterung** |
| **ArtikelTypBean**  **( ArtikelTypTO )**  **ArtikelFacade** | Request | Dieses Objekt ist das eigentliche Artikel Objekt. Die Bean kommt über die ArtikelFacade, die wiederum ein TO über das Backend anfordert. Zur Zeit wird das Backend hier über DBArtikelConnect simuliert |
|  |  |  |
| **LoginBean** | Request | Diese Bean ist für das Anmelden zuständig |
| **MessageBean** | Session | Hier wird alles für die Meldungen gehalten |
| **NavigationBean** | Session | Alles für die SeitenNavigation ist hier zu finden |
|  |  |  |
|  |  |  |
| **SessionBean** | Session | Daten wie LoginStatus.  Angemeldeter User sind hier zu finden. |
| **CustomerBean**  **( CustomerTO )** |  |  |

**com.pinhost.app.webshopxxl2.facade**

**Die Facade ersetzt die sonstigen Daos ( Data Access Object ) :**

Diese Klassen dienen dem übernehmen der Daten aus der Datenbank. Jedoch, aufgrund der Trennung der Anwendung von Frontend und Backend, nicht direkt, sondern im Moment durch das simuliert Backend im common Folder. Das ist die geforderte Schnittstelle.

Diese Vorgensweise entspricht einem Facadepattern.

|  |  |
| --- | --- |
| **Name** | **Erläuterung** |
| **ArtikelFacade**  **( DBArtikelConnect )** | Diese Facade fordert die Daten der Artikel, Artikelgruppen und ArtikelKategorien über das Backend. Jedoch ist dies nur eine Facde und leitet die Anfrage weiter an die dazugehörige **DBArtikelConnect** des Backends. Später muss das echte Backend diese Daten über Tos liefern. |
|  |  |
| **CustomerFacade**  **( DBCustomerConnect )** | Diese Dao liest die Daten der oder des User(s) über die Datenbank. Jedoch ist dies nur eine Facde und leitet die Anfrage weiter an die dazugehörige **DBuSERConnect** des Backends. Die bisher einzIge Methode liest die Daten des Users der sich anmelden möchte. |

**com.pinhost.app.webshopxxl2.util**

**Die Util Klassen**

Diese Klassen liefern Methoden die ständig beraucht werden.

|  |  |
| --- | --- |
| **Name** | **Erläuterung** |
| **Util** | Diese Klasse liefert bisher den FacesContenext für die GUI Klassen und liefert über getter die Beans die im Context angemeldet sind. Dadurch können die Beans miteinander agieren, lesend und schreibend. |
| **MeldungenUtil** | Hier sind alle Meldungen mit Text und Formatierung abgelegt. |

***Backend***

***com.pinhost.common.webshopxxl2.dbconnect***

Diese Klassen sind das Äqivalent zu den DAO Klassen des Frontends.

Diese Klassen lesen über die Datenbank das oder die TOs und liefern diese als Liste oder einzeln Objekt an die dazugehörige DAO des Frontends zurück.

Dazu werden hier in den Methoden die lesenden und schreibenden SQL Abfragen erstellt und an die DataBase Klasse übergeben. Diese liest dann die Daten und liefert ein ResultSet an die DbxxxConect Klassen zurück. Diese Klassen werden beim Einsatz von Hibernate ersetzt.

**Die Klassen**

|  |  |
| --- | --- |
| **Name** | **Erläuterung** |
| **DBCategorieConnect**  **( CategorieDAO )**  **CategorieTO** | In diesem Fall ein CategorieTO an die CategorieDAO des Frontends. |
| **DBProductConnect**  **( ProductDAO )**  **ProductTO** | In diesem Fall ein ProduktTO an die ProduktDAO des Frontends. |
| **DBUserConnect**  **( UserDAO )**  **UserTO** | In diesem Fall ein UserTO an die UserDAO des Frontends. |
| **DataBase** | Diese Klasse baut im Moment die Datenbankverbinduiing auf und liefert die ResultsSets die von den DbxxxConect Klassen angefordert werden. |

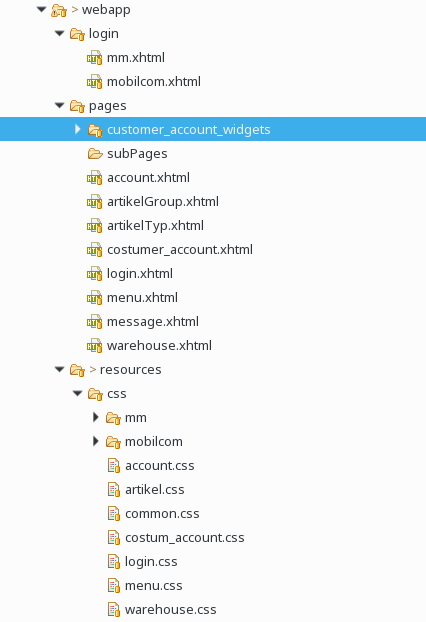
***com.pinhost.common.webshopxxl2.to***

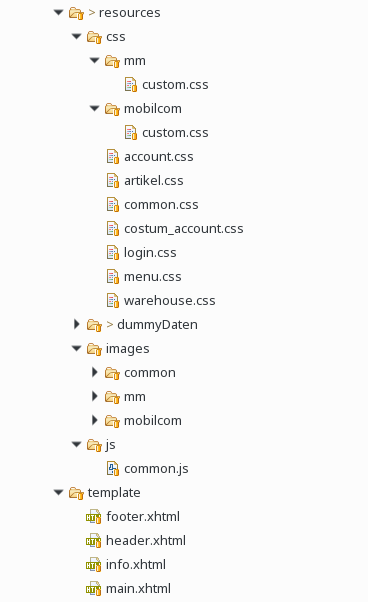
Hier liegen die TO Klassen. Jede ist ein Äqivalent zu den Beans des Frontend

**Die Klassen**

|  |  |
| --- | --- |
| **Name** | **Erläuterung** |
| **CategorieTO** | Das TO für die Kategorien |
| **ProductTO** | Das TO für die Produkte |
| **UserTO** | Das TO für den User |

**Die Architektur der View ( XHTML)**

****

****

**Der Aufbau des Frontends der Webseiten, CSS, Javascript und anderen Resourcen**

Das Webprojekt liegt unterhakb von webapp.

Dort gibt es folgende Aufteilung…

**login :** Hier liegen die Startseiten der Kunden. z.b : ***login/mm.xhtml***

( Vielleicht sollte man den Ordner auch start nennen ? )

Diese Seiten includieren Seitenaufbau ( ***template/main.xhtml***): Diese Dateien includiert unter Umständen die ***pages/login.xhtml***.

In den Startseiten werden auch Kundenspezifische Eigenschaften festgelegt. Dieses geschieht über Session Varaiblen :

<!-- **Here set the value of the Custom Path e.g mm, mobilcom -->**

<c:set var=*"VAR\_CUSTOM\_PATHNAME"* scope=*"session"* value=*"mm"*/>

<!-- Here set the value of the Title e.g M+M, Mobilcom -->

<c:set var=*"VAR\_CUSTOM\_NAME"* scope=*"session"* value=*"M+M"*/>

<!-- Here include the main Page PLEASE NOT CHANGE THIS -->

**<ui:include src=*"../template/main.xhtml"* />**

Sonst geschieht in diesen Seiten nichts.

Jeder Kunden bekommt genau so eine Datei mit seinen Einstellungen.

**pages :** Hier liegen die eigentlichen Seiten die den sichtbaren Inhalt beinhalten

|  |  |  |
| --- | --- | --- |
| **Name** | **Beans** | **Beschreibung** |
| ***categorie.xhtml*** | **CategorieBean** | Zeigt über ein for each die aus der Datenbank gelesenen Categorie Objekte |
| ***login.xhtml*** | **LoginBean**  **UserBean** | Zeigt die Login Maske |
| ***menu.xhtml*** | **NavigationBean** | Baut das Menü auf |
| ***product.xhtml*** | **ProductBean** | Zeiht die gesuchten Produkte an |

**templates:** Hier liegen diejenigen Seiten die den Seitenaufbau bestimmen.

|  |  |  |
| --- | --- | --- |
| **Name** | **Beans** | **Beschreibung** |
| ***error.xhtml*** | **MeldungenBean** | Zeigt Fehler und Statusmeldungen an. |
| ***footer.xhtml*** |  | Bishr noch nicht genutzt |
| ***header.xhtml*** |  | Stellt den Platz für das Menü bereit.  Includiert die ***pages/menu.xhtml*** |
| ***main.xhtml*** |  | Stellt den Platz für den content bereit.  Am anfang ist es die ***pages/login.xhtml***  Gesteuert wird dies über die **NavigationBean** |

**Der bereich der Resourcen : resources**

Hier liegen alle CSS; Javascript und Imagefiles.

**CSS**

Es gibt zur zeit eine common.css die alle Styles beinhaltet die von allen Seiten Kundenunabhängig verwendet werden.

Die in dieser Datei enthaltenen CSS Klassen beginnen mit einem common\_xxx. Dadurch sind diese später besser zu identifizieren.

Dann gibt es die menu.css die das menü verwaltet. Die CSS Klassen hier beginnen mit einem menu\_xxx.

Pro Kunde gibt es eine CUSTOM.css ( mm.css, mobilcom.css ). Diese beinhaltet die Kundenspezifischen Styles wie z.,B Farben.

Die hier abgelegten Klassen beginnen mit einem custom\_xxx

**JS**

Es gibt zur zeit eine common.js die das init() beihaltet.

Parallel wird es Kundenspezifische JS geben die Kundenspeifische arbeiten erledigen.

Parallel beginnen alle JS Methoden mit einem common-xxx() für die scripte die im common.js liegen und mit einem custom\_xxx() die in den Kundenspezifischen Dateien liegen.

**Navigation**

Die Seitennavigation nutzt nicht die Navigation Rules der faces-config.xml. Stattdessen gibt es eine NavigationBean im scope Session.

In dieser Bean ist jede Seite als **private** String ... abgelegt.

**private** String header\_page = "../template/header.xhtml";

**private** String error\_page = "../template/error.xhtml";

**private** String footer\_page = "../template/footer.xhtml";

**private** String content\_page = "../pages/login.xhtml";

// Statics for the Content Pages

**private** String CONTENT\_LOGIN\_PAGE = "../pages/login.xhtml";

**private** String CONTENT\_CATEGORIE\_PAGE = "../pages/categorie.xhtml";

**private** String CONTENT\_PRODUCT\_PAGE = "../pages/product.xhtml";

Für die **private** String CONTENT\_ Seiten gibt es jeweils einen getter

Für jede neue Seite bitte einen eigenen getter anlegen. KEINEN SETTER.

Immer wenn eine Aktion ( CommandButton, CommandLink … ) ausgelöst wird, wird der NavigationBean mit geteilt, welche Seiten als Content gesetzt werden. Beim Rendern wird dann diese Seite eingeblendet.

**Beispiel : nach Login**

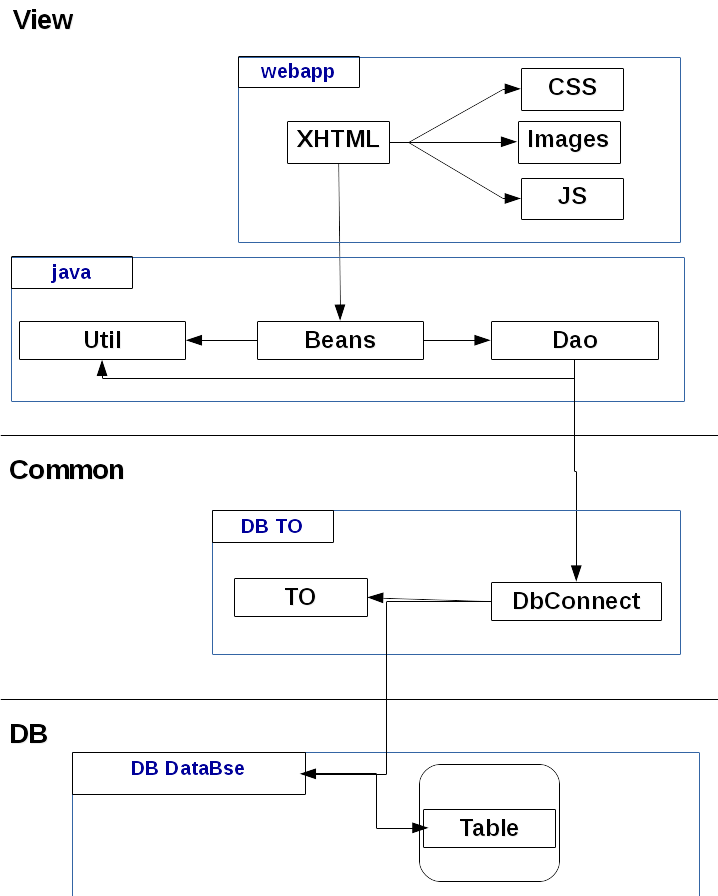
**public** String login(){

. . .

getNavigationBean()

.setContent\_page(getNavigationBean()

.getCONTENT\_CATEGORIE\_PAGE());





Beispielhaft zeigen die obigen Diagramme den Weg der Datenanforderung bis zur Datenlieferung anhand des Logins.

Beim starten der Anwendung erscheint die Kundenspezifische Login Datei ( z.b mm.xhtml ) die direkt die login.xhtml includiert.

Die beiden Eingabefelder Loginname und Passoword respresentiert durch die beidem Inputfelder

<p:inputText id=*"id\_loginpage\_loginname\_input"*

onchange=*"js\_common\_check\_login()"*

onblur="js\_common\_check\_login()"

onkeypress="js\_common\_check\_login()"

value=*"#{loginBean.loginName}"* size=*"30"* />

<p:password id=*"id\_loginpage\_password\_input"*

onchange=*"js\_common\_check\_login()"*

onkeypress="js\_common\_check\_login()"

onblur="js\_common\_check\_login()"

value=*"#{loginBean.password}"* size=*"30"* />

Diese sind gebunden an die LoginBean. Und an die Properties

**private** String loginName="";

**private** String password="";

Der Button der login.xhtml ist gebunden an die Methode

**public** String login()

Diese Methode holt über den Aufruf UserTO userTO = UserDao.*getUserForLogin*(loginName, password);

der Klasse **UserDao** ein **UserTO** Object. Dabei übergibt diese Methode den Usernamen und das Password die über die login.xhtml eingebenen worden sind.

**UserDao** : Diese Klasse ist die Verbindung der View zur Datenbank über die Klasse **DBUserConnect.**

Die Klasse **UserDao** holt nun in der Methode **getUserForLogin(loginName, password)**

über die Klasse **DBUserConnect** in der Methode **readUserForLogin(loginName, password)** das **UserTO** Object aus der Datenbank.

Die Klasse **DBUserConnect** ist die Verbindung zur Datenbank im Backend. Jede **DAO** Klasse in der View Schicht hat eine **DBXXXConnect** Klasse im Backend. Dadurch wird eine Trennung der View zum Backend erreicht und die Trennung des Backends zum Frontend ermöglicht. Man kann auch sagen das dies eine Facade ist. Alle **DBXXXConnect** Klassen sind abgeleitet von der Klasse **DataBase**.

In der Klasse **DBUserConnect** wird die SQL Abftage erstellt ( oder Hibernate ) :

String readSQl = "SELECT \* FROM " + *TABLE* +" WHERE "

+ *LOGINNAME* + " = '" + loginname

+ "' AND " + *PASSWORD* + " = '" + password + "'";

Diese Abfrage wird dann durch den Aufruf

**ResultSet** res = getResult(readSQl);

an die Methode

**public** ResultSet getResult(String readSQL)

der **Database** Klasse übergeben.

Ist das ResultSet erfolgreich, was bedeutet das die Abfrage den User gefunden hat , wird das **UserTO** Objekt erstellt und nach oben an die Klasse **UserDao** übergeben. Dann wird das **UserTO** Objekt in ein UserBean Objekt umgewandelt.

Das Geschieht über den Aufruf der statischen Methode

*getUserBean*().setUser(userTO);

der Klasse **Util**.

Am Ende wird ein **true** oder **false** an die Klasse **LoginBean** zurück gegeben.

Im Falle von **true** wird nun eine Erfolgreich Meldung geschrieben.

Es wird die neue content Page gesetzt.

Dazu werden nun die Klassen **SessionBean**, **MessageBean** und die **NavigationBean** aus dem Context geholt. Das geschieht wieder über die Klasse **Util** mit den Methoden

getNavigationBean()

getSessionBean()

getMessageBean()

und es wird in der SessionBean ein true im Flag : **private** **boolean** userIsOnline = **false**;

mit : getSessionBean().setUserIsOnline(**true**); gesetzt.

Dann wird die neue ContentPage mit

getNavigationBean().setContent\_page(getNavigationBean()

.getCONTENT\_CATEGORIE\_PAGE());

und

eine neue Meldung in der meldung.xhtml mit

getMessageBean().setLoginCorrectState(*getUserBean*()

.getFirstName(), *getUserBean*().getLastName());

gesetzt.