

1 Система за електронна търговия

В тези упражнения ще създадем една система за електронна търговия. И точно само част от нея – няма да можем да реализираме реален начин на заплащане през интернет. Ще създадем интернет система за електронна търговия базирана на Java. Тя има база данни, в която се съхраняват данни за обектите на системата – потребители и стоки. Има и обслужващ модул – съвкупност от jsp страници (и/или сървлети), използващи и визуализиращи данните от базата данни.

Забележки:

- Системата може да бъде реализирана и чрез други (не Java) технологии.
- Тук няма да демонстрираме EJB (Enterprise Java Beans) технологията, която е подходяща за електронната търговия, но доста “тромава” и скъпа (поради цените на Application сървърите необходими за нея).
- Няма да демонстрираме и много от възможностите на J2EE (Java 2 Enterprise Edition) технологията, която включва множество подтехнологии (едни от тях са EJB и JSP), които са разширение на стандартните технологии (J2SE).

Ще работим с JBuilder и предоставените от него стандартни средства (например web сървър Apache Tomcat, в който се изпълняват (стартират) jsp страниците), и с MS SQL Server за базата данни.

По-конкретно ще създадем електронен магазин. В него ще има няколко основни дейности:

- Разглеждане на продукти;
- Регистриране на купувач;
- Събиране на продукти от купувача (в “пазарска кошница”);
- Купуване на продуктите;

Забележка: Целта на разглеждания пример е да се представят някои техники на програмиране и идеи за работа с многослойни архитектури и не толкова да се направи истински електронен магазин.

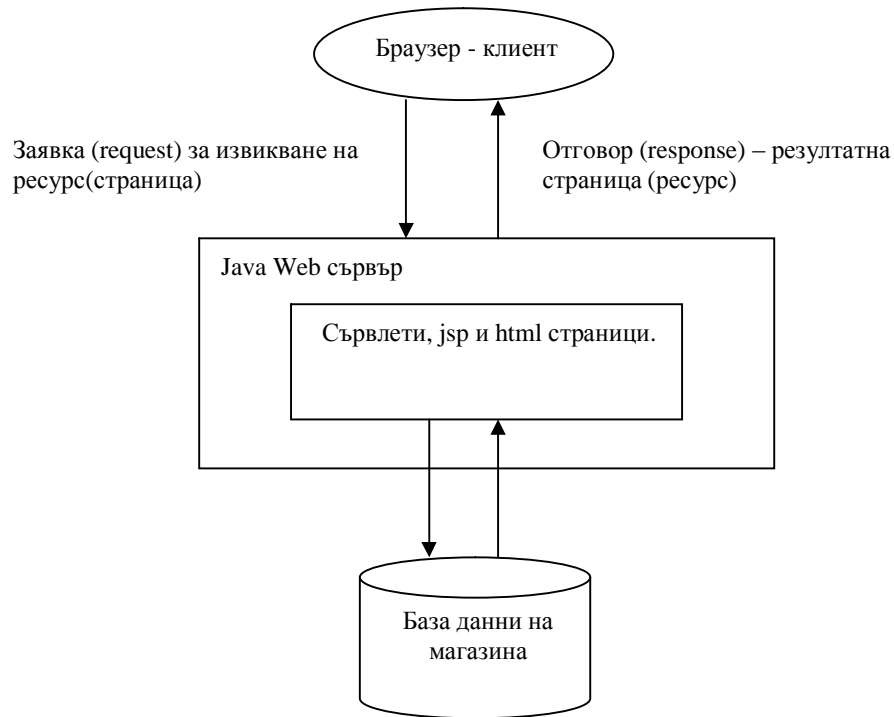
2 Архитектура на системата

На следващата фигура е дадена архитектурата на системата.

- Имаме база данни, в която е съхранена информацията за продуктите.
- Достъпа да данните се прави през jsp страници и сървлети. В зависимост от данните, резултатните (генерираните от jsp сървъра) HTML страници съдържат различна информация.
- При заявка от клиента (извикване на страница) web сървъра изпълнява jsp страницата, при което се генерира резултатна HTML страница, която се връща на клиента.

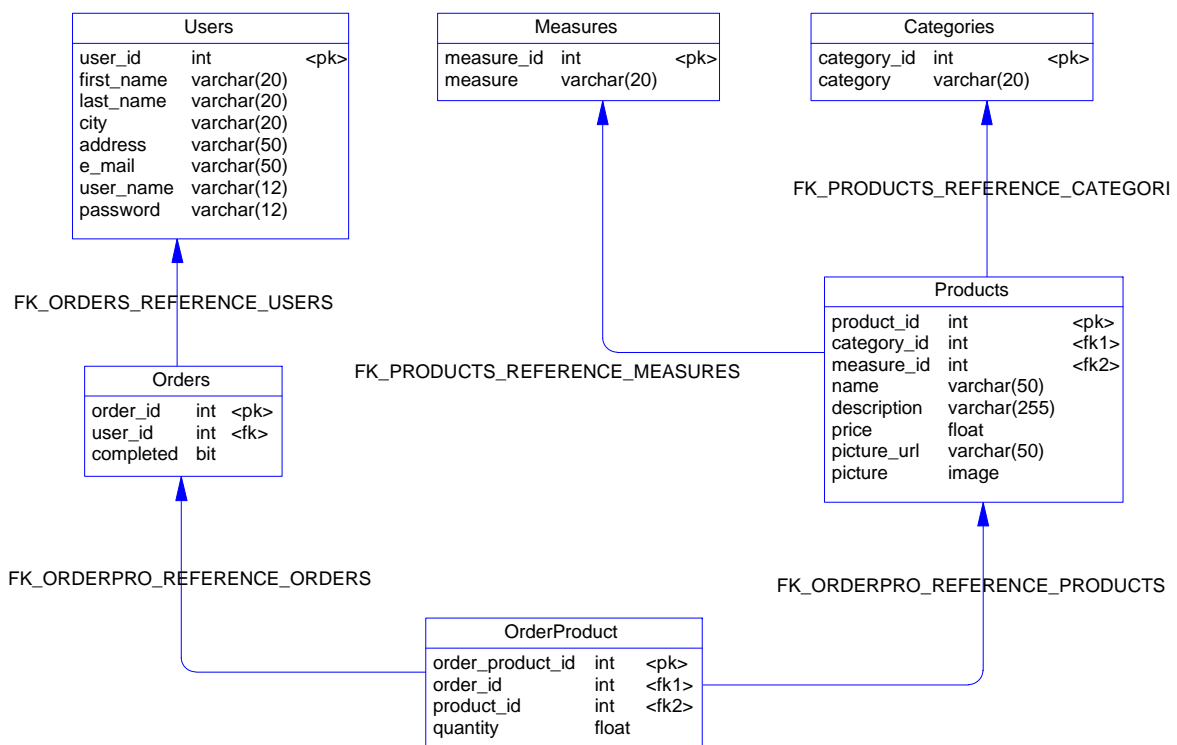
При тази архитектура jsp страниците се грижат за бизнес (работната) логиката на системата. В тях се извършва обработката на данните от базата

данни. Те изпълняват и стандартните си функции – да се грижат за представянето на данните.



3 База данни на системата

Базата данни на системата е показана на следващата схема.



В тази база данни е описана информация за продукти, клиенти и поръчки.

Таблиците са следните:

3.1. Users

Съдържа информация за потребителите.

Поле	Описание	Ограничения
user_id	Първичен ключ.	int identity
first_name	Име на клиент.	varchar(20) not null
last_name	Фамилия на клиент.	varchar(20) null
city	град	varchar(20) null
address	Адрес	varchar(50) null
e_mail		varchar(50) null
user_name	Потребителско име	varchar(12) not null
password	Парола	varchar(12) not null

3.2. Categories

Номенклатурна таблица, съдържаща информация за категории продукти.

Поле	Описание	Ограничения
category_id	Първичен ключ.	int identity
category	Категория.	varchar(20) not null

3.3. Measures

Номенклатурна таблица, съдържаща информация за измервателни единици за продукти.

Поле	Описание	Ограничения
measure_id	Първичен ключ.	int identity
measure	Измервателна единица.	varchar(20) not null

3.4. Products

Съдържа информация за потребителите.

Поле	Описание	Ограничения
product_id	Първичен ключ.	int identity
category_id	Външен ключ към категория.	int not null
measure_id	Външен ключ към измервателна единица.	int not null
name	Име на продукта	varchar(50) not null

description	Описание на продукта	varchar(255) null
price	Цена на продукта за единица от измервателната единица на продукта	float not null
picture_url	Път (относителен) на рисунка.	varchar(50) null
picture	Рисунка записана директно в таблицата.	image null

3.5. Orders

Съдържа информация за поръчки на клиенти – всяка поръчка си има автоматично генериран уникален номер, отнася се за даден клиент и може да съдържа заявка за няколко продукта (продуктите, описани в заявка се намират следващата таблица). Тъй като избора на продукти може да бъде продължителен във времето и не е задължително да бъде направен наведнъж е предвидено тя да може да бъде продължавана (полето completed ще е 1, когато поръчката е завършена и ще може да се изпълни от магазина).

Поле	Описание	Ограничения
order_id	Първичен ключ.	int identity
user_id	Външен ключ към клиент.	int not null
completed	Завършена ли е поръчката: 0 – не е завършена; 1 – завършена е.	bit not null default 0

3.6. OrderProduct

Съдържа информация за продуктите, поръчани в една заявка.

Поле	Описание	Ограничения
order_product_id	Първичен ключ.	int identity
order_id	Външен ключ към заявка.	int not null
product_id	Външен ключ към продукт.	int not null
quantity	град	float not null

4 JSP-та

4.1. Разработка на web сайт с помощта на JBuilder

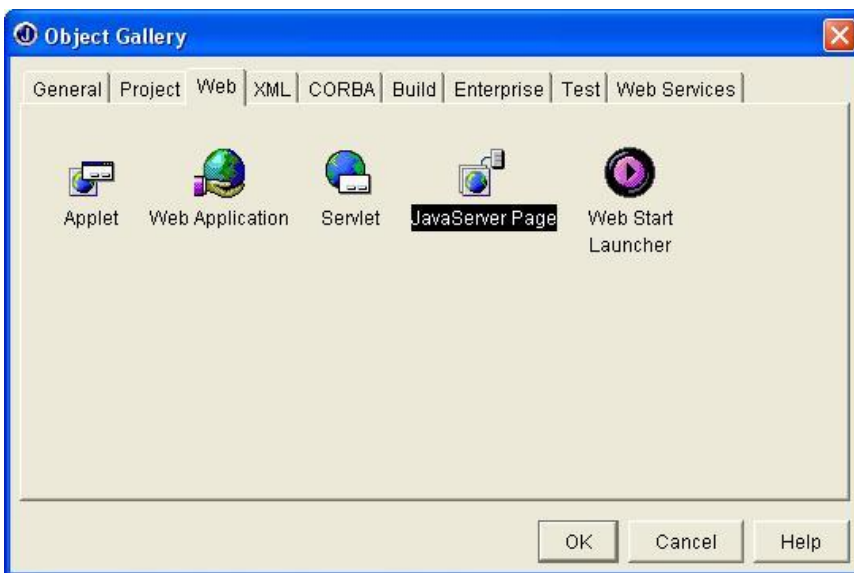
При разработката на сайт с помощта на стандартните средства на JBuilder се освобождаваме от някои дейности:

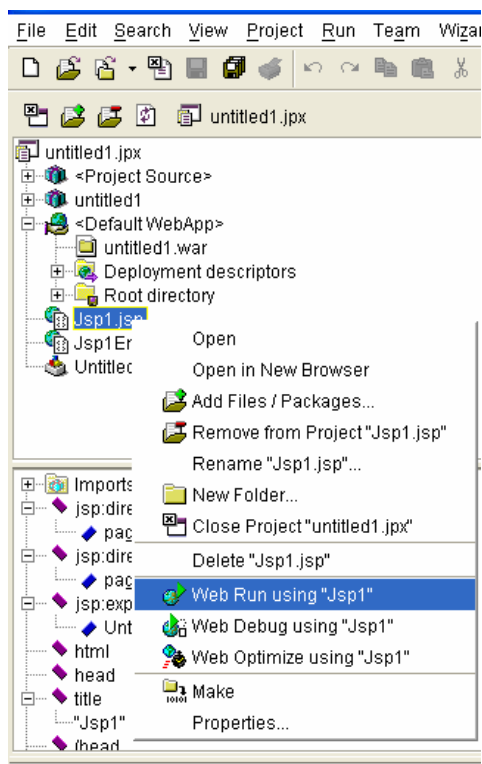
- **Не е необходимо да инсталираме и конфигурираме web сървър** – в JBuilder е вградено използването на Java web сървъра Tomcat на Apache;
- **Не е необходимо да конфигурираме web приложение.** – За да бъде създаден сайт “намиращ” се на даден сървър е необходимо той да бъде конфигуриран (трябва да се зададат различни настройки на web сървъра, така че да стане възможно извикването на страници от сайт). Това конфигуриране става като се задават разни параметри (име на сайт, имена на ресурси) в специални xml конфигурационни файлове. Вместо ние да се грижим за това, JBuilder автоматично създава тези файлове и оформя т.нар. **web приложение**, стига обаче да използваме специално създадените (в JBuilder) за това wizard-и. След това приложението (първата (или дори друга произволна) страница на сайта може да бъде стартирана с опцията web run, която се появява за web приложенията в контекстното меню на страниците).

JBuilder подготвя отлични web приложения, които могат по-късно лесно да бъдат пренесени върху реални сървъри (работещи без да е необходимо да е стартиран JBuilder). Но ние няма да използваме тази възможност.

По-конкретно създаване и стартиране на jsp-та с **JBuilder7** е показано на следващите 2 схеми.

След като се избере опцията от менюто File|New... се създава Java Server Page.





4.2. Схема на сайта на електронния магазин

Ще представим структурата на web сайта на електронния магазин. С този сайт ще обслужваме част от възможните операции върху създадената по-горе база данни. Не са предвидени, например, възможности за администриране на базата данни – това са операции като добавяне на нови мерни единици, нови категории продукти и нови продукти.

Предвидени са клиентски страници за основни операции, които са подходящи за работа на клиента с магазина.

За администриране на системата може да се реализира административен модул, който може да е web базиран или обикновено визуално приложение.

Най-общата схемата на сайта, който ще разработваме е показана на следващата фигура.

<div>потребителско име и парола</div> <div>регистрация</div>		<div>Заглавна информация на електронния магазин</div>
<div> Начало Категории Продукти </div> <div>търсене</div> <div> Поръчки Профил Изход </div>	<div> Коя страница е активна Здравей , Кошница(н) </div> <div>Основна информация</div>	

В сайта имаме няколко основни част:

- **Заглавна част** – заглавие на сайта, лого, входна форма за магазина (тя може да не се появява когато потребителя вече е влязъл с потребителско име и парола), връзка към страница за регистрация на нов клиент – клиент ще може да пазарува само ако се е регистрирал и е влязъл в сайта с потребителско име и парола;
- **Меню** – част от функциите на менюто са винаги активни (категории и продукти), друга част е активна само ако потребителя е влязъл в сайта с потребителско име и парола (поръчки, профил на клиента, изход).
- **Основна част** – в нея има информация за текущо избраната функция – **разглеждане на категории продукти, разглеждане на продукти, разглеждане на поръчките на клиента, промяна на профила на клиента, изход (приключване на сесията)**. Ще има и други основни функции, достъп до които обаче няма да става през менюто, а от хипервръзки от основната страница (тези функции са зависими от основните и не могат да бъдат описани в менюто) – **разглеждане съдържанието на кошницата, разглеждане на продукти в дадена категория, добавяне на продукт/и в кошницата, премахване на продукт/и от кошницата, приключване на поръчка, отказ от поръчка;**

Възможни са два основни вида организации на сайта:

- Във фреймове – както са нарисувани основните части на сайта; Тогава може заглавната част и менюто да са относително статични страници, а да се сменя най-често съдържанието на третия фрейм – в него да се изобразяват множество страници, съответни на възможните функционалности на магазина.
- Всяка функционалност е реализирана като отделна jsp страница. Тогава във всяка страница ще има повтаряща се информация – менюто и заглавната част.

При първия подход няма непрекъснато да се зарежда информацията за менюто и заглавната част – може да се опреснява съдържанието най-често на основния фрейм. По-трудно е обаче да се контролира промяната на информация в отделните фреймове – ще трябва да се зареждат менюто и заглавната част наново понякога.

При втория подход плюсовете и минусите са почти наобратно – всяка страница се генерира напълно независимо че има повтаряща се информация (която би могла да не се зарежда), но съдържанието е по-добре контролируемо.

Втория подход се използва все по-често. Освен това в jsp технологията може да не записваме един и същи код във всяка страница – в едно jsp може да вложим други с помощта на директива include.

4.3. Код за създаване на рамката на трите части на сайта

С помощта на следния код можем да си направим рамките на трите отделни части на сайта:

```
<table border="1" cellpadding="0" cellspacing="0" height="100%" width="100%">
  <tr height="100">
    <td colspan="2">&nbsp;  </td>
  </tr>
  <tr>
    <td width="110">&nbsp;  </td>
    <td>&nbsp;  </td>
  </tr>
</table>
```

Това е таблица с два реда. Втория ред има две колони и затова в единствената колоната на първия ред сме записали, че тя заема две колони (colspan="2").

Във всяка клетка по-нататък ще слагаме информация за съответната част – можем също да вграждаме други таблици, в които по-добре да разпределим информацията.

border="1" е оставено за да се вижда рамката при първоначалното тестване на страницата. По-късно може да се направи на 0.

Височината на таблицата е 100%, а също и ширината - заема целия браузер – добре е вътрешните таблици да са с фиксирани ширини и дължини за да не свиват или разтягат неочаквано за нас.

При Netscape има проблеми с някои техники, които се използват тук – размера на таблицата height="100%" не дава очаквания ефект. (При вложаване на таблици май имаше проблеми с background-ите и с други неща.)

4.4. Създаване на първоначални версии на менюто и заглавната част

Всички страници може да имат стандартен вид, показан по-долу.

```
<% @ page contentType="text/html; charset=windows-1251" %>
<% @ page import="java.sql.*"%>
<html>
<head>
<title>Електронен магазин</title>
</head>
<body>

<table border="1" cellpadding="0" cellspacing="0" height="100%" width="100%">
<tr height="100">
<td colspan="2"><% @ include file="header.jsp"%></td>
</tr>
<tr>
<td valign="up" width="110"><% @ include file="menu.jsp"%></td>
<td valign="up" width="900"> Код за конкретната страница
</td>
</tr>
</table>

</body>
</html>
```

За разлика от горния код (в 4.3) тук сме направили една “хитрост” – фиксирали сме размера на лявата долна клетка на таблицата за рамките (`<td valign="up" width="900">`). Ширината е малко по-голяма от възможната (при нормална разделителна способност на екрана). Реално сумата от ширините на клетките не може да надхвърли 100% - на колкото сме определили размера на цялата таблица. Тази “хитрост” се налага, тъй като като влагаме други таблици (например в `header.jsp` – по надолу е даден кода му) се разместват клетките. Възможно е нещата да се организират, така че да не се прибегва до такива “хитрости”.

С помощта на директивата `include` във всяка страница добавяме относително стандартния изглед на менюто и заглавната част. Със следния код например във всяко основно `jsp` добавяме кода на `menu.jsp`.

```
<% @ include file="menu.jsp"%>
```

При генериране на основна страница от основно `jsp` кода на вмъкнатото `jsp` се добавя динамично в основното. В допълнителното `jsp` може да имаме някаква логика, така че то да генерира различно съдържание в зависимост от това дали клиента е влязъл с потребителското си име и парола или не.

Важно при работа: Ако се променя кода на вложено `jsp`, за да се компилира съдържащото го `jsp` (и да станат валидни промените) трябва да се промени датата на съдържащото `jsp` (да се запише със `save` например).

Кодът на `menu.jsp`, например, може да е следния:

Файл menu.jsp:

```

<a href="index.jsp">Начало</a></br>
<a href="categories.jsp">Категории</a></br>
<a href="products.jsp">Продукти</a></br>

<form method="post" action="products.jsp">
  <input type="text" name="productNamePart" value="" size="12"/>
  <input type="submit" name="Submit" value="Търсене"/>
</form>

</br></br>

<a href="orders.jsp">Поръчки</a></br>
<a href="profile.jsp">Профил</a></br>
<a href="logout.jsp">Изход</a></br>

```

Забележка: Това е първоначалният му вариант – после ще добавяме логика кога да се показват и кога – не част от хипервръзките.

Код на header.jsp:

```

<table width="100%" cellpadding="0" cellspacing="0">
  <tr>
    <td width="200" align="left">
      <form method="post" action="login.jsp">
        <table cellpadding="0" cellspacing="0">
          <tr>
            <td align="right">
              <b>Име: </b>
            </td>
            <td align="left" colspan="2">
              <input type="text" name="userName" size="15"/>
            </td>
          </tr>
          <tr>
            <td align="right">
              <b>Парола: </b>
            </td>
            <td align="left">
              <input type="password" name="password" size="15"/>
            </td>
            <td align="right">
              &nbsp;<input type="submit" name="Submit" value="Вход"/>
            </td>
          </tr>
          <tr>
            <td align="left" colspan="3">
              <a href="registration.jsp">Регистриране</a>
            </td>
          </tr>
        </table>
      </form>
    </td>
    <td align="right">
      <H2>Картинка(лого) на нашия електронен магазин</H2>
    </td>
  </tr>
</table>

```

Тук като пример за заглавна част на всички страници е дадена една съвкупност от таблици, в които са организирани основните данни (на заглавната част).

4.5. *index.jsp*

```
<% @ page contentType="text/html; charset=windows-1251" %>
<html>
<head>
<title>
Електронен магазин - начална страница
</title>
</head>
<body>

<table border="1" cellpadding="0" cellspacing="0" height="100%" width="100%">
<tr height="100">
<td colspan="2"><% @ include file="header.jsp"%></td>
</tr>
<tr>
<td valign="up" width="110"><% @ include file="menu.jsp"%></td>
<td valign="up" width="900">
<H1 align="center">Електронен магазин "ЕМ"</H1>
<H3>Разни описания в началната страница</H3>
</td>
</tr>
</table>

</body>
</html>
```

4.6. *Описание на част от основните jsp страници*

В горните 2 jsp страници сме заложили създаването на няколко други jsp страници:

- **categories.jsp** – съдържа списък от всички категории във вид на хипервръзки. При избор на категория ще се извиква **products.jsp** с параметър обозначаващ избраната категория – най-подходяща стойност за това е стойността на първичния ключ на категорията.
- **products.jsp** – извежда списък от продукти. Има три начина за показване на списъка:
 - **избор на опцията от менюто** – извежда се списък от всички продукти;
 - **избор на категория** – извежда се списък от всички продукти за дадената категория;
 - **в менюто при въвеждане на име за търсене и натискане на бутона “Търсене”** – извежда се списък от всички продукти, в чието име се среща зададения низ за търсене.

При избор на продукт от списъка се извиква **product.jsp**, с параметър указващ избрания продукт.

- **orders.jsp** – Разглеждане на поръчките, които е правил дадения клиент. В тази страница се извежда списък от всички поръчки (с хипервръзки).

Като се избере дадена поръчка се вижда информация за поръчката – **viewOrder.jsp**. *Тези страници няма да ги реализираме.*

- **profile.jsp** – от тук може да се редактират данните на клиента. При избор на функцията за запис (**saveProfile.jsp**) се променят данните на клиента и се показва страницата **login.jsp**, с метод на обекта `response redirect`.
- **logout.jsp** – клиента излиза от магазина (премахва информация от сесията).
- **login.jsp** – записва се в сесията информация за клиент (която се използва от другите страници докато клиента не прекъсне сесията) и се изобразява начална информация за влизане на регистриран клиент в магазин. Ако последната поръчка на клиент не е завършена не е избрал **realizeOrder.jsp** или **rejectOrder.jsp**, то се извежда информация за това, че последната поръчка не е завършена и в количката има продукти.
- **registration.jsp** – създава се нов клиент – записват се данните му в базата данни. След запис (**saveProfile.jsp**) автоматично се отваря страницата **login.jsp**.

При отваряне на горните страници се появяват и други възможни действия (страници):

- **product.jsp** – извежда информация за продукта. От тук може да се избира функцията “добавяне в кошницата” ако клиента е влязъл с нужната парола – **addToBasket.jsp**.
- **addToBasket.jsp** – записва информация за добавения в кошницата продукт в базата данни. Появява се страницата със списъка от продукти **products.jsp**.
- **basket.jsp** – може да се извиква от всяка основна страница на клиент. Показва съдържанието на кошницата. Може да се премахва продукт или да се променя поръчаното количество (**removeFromBasket.jsp**, **changeBasket.jsp** – *това може да не го реализираме*), да се завършва поръчката (**realizeOrder.jsp**) или да се направи отказ на поръчката (**rejectOrder.jsp**).
- **realizeOrder.jsp** – в полето `completed` в базата данни за дадената поръчка се записва 1. Показва се страницата **login.jsp**.
- **rejectOrder.jsp** – премахва се информацията за поръчката от базата данни. Показва се страницата **login.jsp**.
- **viewOrder.jsp** – разглеждане на направена поръчка.
- **removeFromBasket.jsp** – премахане на продукт от кошницата;
- **changeBasket.jsp** – промяна на количество на продукт от кошницата;

4.7. Categories.jsp

```
<% @ page contentType="text/html; charset=windows-1251" %>
<% @ page import="java.sql.*"%>
<html>
<head>
<title>
Електронен магазин - Категории продукти
</title>
</head>
<body>
```

```

<table border="1" cellpadding="0" cellspacing="0" height="100%" width="100%">
  <tr height="100">
    <td colspan="2"><% @ include file="header.jsp"%></td>
  </tr>
  <tr>
    <td valign="up" width="110"><% @ include file="menu.jsp"%></td>
    <td valign="up" width="900">
      <H1>Категории</H1>
      <%
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection("jdbc:odbc:storeDB", "sa", "");
        Statement stmt = con.createStatement();

        ResultSet rs = stmt.executeQuery("SELECT category_id, category from Categories");
        while (rs.next())
        {
          %>
          <a href="products.jsp?cId=<%=rs.getString(1)%>"><%=rs.getString(2)%></a></BR>
          <%
          }

          rs.close();
          stmt.close();
          con.close();
          %>
        </td>
      </tr>
    </table>

</body>
</html>

```

В тази примерна реализация записваме кода за връзка с базата данни директно в тялото на jsp-то. Това не е много добра техника.

За да тръгне примера трябва да се създаде ODBC DataSource с име storeDB. Може и с друго, но да напише на необходимото място - `Connection con = DriverManager.getConnection("jdbc:odbc:storeDB", "sa", "")`.

4.8. products.jsp

```

<% @ page contentType="text/html; charset=windows-1251" %>
<% @ page import="java.sql.*"%>
<html>
<head>
<title>
Електронен магазин - Продукти
</title>
</head>
<body>

<table border="1" cellpadding="0" cellspacing="0" height="100%" width="100%">
  <tr height="100">
    <td colspan="2"><% @ include file="header.jsp"%></td>
  </tr>
  <tr>

```

```
<td valign="up" width="110"><% @ include file="menu.jsp"%></td>
<td valign="up" width="900">
<H1>Продукти</H1>
<%
request.setCharacterEncoding("windows-1251");
Integer cId = null;
try
{
cId = new Integer(request.getParameter("cId"));
}
catch(Exception e){ }

String productNamePart = request.getParameter("productNamePart");

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:storeDB", "sa", "");
Statement stmt = con.createStatement();

String query = "SELECT product_id, name FROM Products";

if(productNamePart!=null)
query = query + " WHERE name LIKE '%" + productNamePart + "%'";
else
if(cId!=null)
query = query + " WHERE category_id=" + cId;

query = query + " ORDER BY name";

ResultSet rs = stmt.executeQuery(query);
if(!rs.next())
{
%>
<p style="color:red">Няма намерени продукти!</p></BR>
<%
}
else
{
%>
<a href="product.jsp?pId=<%=rs.getString(1)%>"><%=rs.getString(2)%></a></BR>
<%
}
while(rs.next())
{
%>
<a href="product.jsp?pId=<%=rs.getString(1)%>"><%=rs.getString(2)%></a></BR>
<%
}

rs.close();
stmt.close();
con.close();
%>
</td>
</tr>
</table>

</body>
</html>
```

За да четем българските символи (например въведените в полето за търсене) използваме следния метод:

request.setCharacterEncoding("windows-1251") преди
request.getParameter("productNamePart"). С втория метод четем от обекта request параметъра productNamePart, който е текстовото поле от формата на менюто. Ако products.jsp е извикан от страницата със списъка на категориите, четем друг параметър **cId = new Integer(request.getParameter("cId"))**. В зависимост от това дали са null или не горните два параметъра създаваме три различни вида заявки (query).

За да изведем съобщение, че няма намерени продукти правим едно малко неприятно повторение на код (за хипервръзките) – може да има и друг метод, който да използваме по-удачно в този случай (а не rs.next()).

4.9. За следващия път

Другият път ще реализираме по-правилно създадените jsp-тата, а също и ще създадем част от останалите.

5 Описание на втора версия на електронния магазин

Във втората версия на магазина не е спазвана единна технология за работа – това е направено с цел да се демонстрират повече различни техники (има и други, които не са демонстрирани).

Не всички описани по-горе jsp-та са реализирани като jsp-та – има и сървлети.

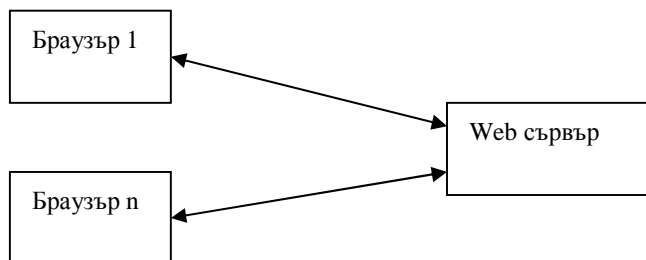
Не е демонстриран стандартния механизъм за показване на изключения и грешки на клиента – с `errorPage`.

5.1. Промени в дизайна на сайта

За по-добра структурираност на страниците на сайта е добавено още едно стандартно jsp, което се вмъква във всяка основна страница. То се нарича `headingInfo.jsp`. В него се извежда стандартно заглавна информация за всички страници – коя страница е активна, информация за активния потребител, ако има такъв и хипервръзка за кошница (ако има активен потребител).

5.2. Работа със сесиен обект

Браузърът изпраща до сървъра заявки и получава отговори.



Възможно е много браузъри да изискват един и същ ресурс от сървъра. За всеки web клиент web сървър създава сесиен обект.

Сесията е съвкупност от заявки и отговори осъществявани между един браузър (клиент - може и да не е браузър) и web сървър.

В сесийния обект (обект от клас `HTTPSession`) може да се съхранява информация, която да се използва от различни страници (извикани при различни заявки от клиента).

Пример:

Клиент влиза с потребителско име и парола – в една форма записва данните и на `submit` се извиква следваща страница. Информацията за клиента може да бъде записана в сесията, така че при отваряне на следваща страница да не е необходимо отново да се въвеждат потребителско име и парола. Ако тази

информация не е известна на сървъра по принцип би било невъзможно да се осигурява сигурност на комуникацията (защото всяка една страница от сайта би могла да бъде извикана директно без да е необходимо да се минава през login страницата като например се напише url-то в адрес полето (address bar) на браузъра).

В jsp страница директно можем да използваме сесийния обект с неговото име **session** (той се създава автоматично).

В сървлет, достъпа до сесията може да стане през обекта request – **request.getSession()** (може и по друг начин – може да се разгледа кода на преобразуваните до java jsp-та).

Параметри в сесията се записват например така (в servlet):

request.getSession().setAttribute("orderId", orderId);

Така записваме атрибут с име “orderId” и стойност orderId. Стойността orderId трябва да е обект (наследник на Object, а не прост тип). В нашия случай orderId е от тип Long.

В сесията обикновено могат да се записват само обекти, които са сериализирани – т.е. наследници на интерфейса Serializable. От реализацията на web сървъра зависи дали в сесията могат да се записват и несериализирани обекти – Tomcat може да записва такива.

Четене на записания параметър на сесията може да стане така:

Long orderId = (Long) request.getSession().getAttribute("orderId");

По името на параметъра взимаме стойността му, която сме записали и трябва да преобразуваме до съответния тип, до който знаем че може да се преобразува – знаем, защото ние сме реализирали системата. Ако в сесията няма записан атрибут с такова име **request.getSession().getAttribute("orderId")** връща **null**. “null” се връща обаче и ако директно сме записали стойност null, за даденото име.

(Вместо да записваме информация в сесията може да се препредават параметри на извикваните страници, което обаче не е много препоръчително.)

5.3. Допълнителни класове използвани в системата

Създадени са класове, които използваме за съхраняване на информация в сесията. Направили сме ги сериализирани.

5.3.1 User

Няма нищо особено в него – полета за съхраняване на информация за потребител, конструктор и методи за достъп до полетата.

5.3.2 Product

Няма нищо особено в него – полета за съхраняване на информация за продукт, съответни на полета от таблицата Products и още едно поле за количеството на

продукта (то се използва когато потребител слага в кошницата продукт); конструктор и методи за достъп до полетата.

5.3.3 Basket

Това е клас за кошницата.

В списък (`ArrayList products`) се записват добавянията на продукти (обекти от клас `Product`). Това става с помощта на метода **`addToBasket(Product product, float quantity)`**.

Метода **`productsInBasket()`** връща броя на елементите в списъка на кошницата. Точния брой на обектите в кошницата е малко трудно да се определи, защото например не е известно какъв е броя на 5 кутии + 2 литра. Ако в базата данни има още сведения, то намирането на точен брой на продукти може да стане.

Извеждаме допълнителна необходима за клиента информация за кошницата с помощта на метода **`float priceInBasket()`**. Това е цената на продуктите в кошницата. При реализацията на този метод се оказва, че Java има голям проблем при работа с числа с плаваща запетая. (Например 3.3×3.0 е равно на 9.8999999). Затова последния ред на реализацията на метода е малко странен – закръгляване (с малко надписване – плюс 0.0001) на сумата до стотни.

За повече подробности по проблема с плаващата запетая – проект Borneo <http://www.sonic.net/~jddarcy/Borneo/>, стандарт IEEE 754 за числа с плаваща запетая.

```
public float priceInBasket()
{
    float price = 0;
    Product product;

    for(int i=0; i<products.size(); i++)
    {
        product = (Product) products.get(i);
        price += product.getPrice()*product.getQuantity();
    }

    return ((float)((long)((price+0.0001)*100)))/100;
}
```

5.3.4 Други проблеми с плаващата запетая

Не само Java има проблем с плаващата запетая. В MS SQL Server, например при запис в поле `float` на числото 3 – резултатът е 3.0000001

За това например в sql кода на някои места е използвана конструкция от вида:

`convert(numeric(10, 3), p.price)`

Функцията `convert` преобразува стойността на полето `p.price` до число с 3 символа след десетичната запетая.

5.4. *registration.jsp* и сървлет за регистриране *saveprofile*

За да може човек да се log-ва, той първо трябва да има създаден профил в базата данни (при нас профил на човек е един запис в таблицата Users). Създаването на профил става с **registration.jsp**. В него имаме форма с полета които трябва да се попълнят. Някои от полетата са означени като задължителни – дали са зададени стойности в задължителните полета се прави проверка в извиквания на submit на формата сървлет **saveprofile**.

В head частта на *registration.jsp* е описан *javaScript* код, с чиято помощ при отваряне на страницата маркера се позиционира в полето за първо име. За да стане това извикваме функцията при load на страницата: **<body onLoad="focus();">**.

```
function focus()
{
    document.registrationForm.firstName.focus();
}
```

При извикване на **saveprofile** се извършват някои основни действия:

1. Взимат се параметрите от request обекта – това са стойностите на полетата на формата:

```
String firstName = getParameter(request, "firstName");
String lastName = getParameter(request, "lastName");
String city = getParameter(request, "city");
String address = getParameter(request, "address");
String email = getParameter(request, "email");
String userName = getParameter(request, "userName");
String password1 = getParameter(request, "password1");
String password2 = getParameter(request, "password2");
```

Тук сме реализирали един допълнителен метод *getParameter()*, който връща null, когато търсен параметър не съществува в request-а (има стойност null) или когато съществува, но дължината на низа без крайните интервали е 0. Така по-лесно обработваме (еднотипни) гранични ситуации за стойностите на параметрите.

2. Ако не са попълнени някои задължителни полета или зададените пароли не са еднакви в атрибут на сесията се записва информация за грешка и освен това се извиква отново страницата за регистрирането *registration.jsp*, на която се предават обаче информация за грешката и освен това и информация за попълнените вече стойности на полетата – за да не се загубят и потребителя да трябва да ги въвежда наново.

```
request.getSession().setAttribute("errorValue", "Не са попълнени някои задължителни данни или паролите не са еднакви!");
```

```
response.sendRedirect("registration.jsp?error=true"
    + (firstName!=null?"&firstName=" + firstName:"")
    + (lastName!=null?"&lastName=" + lastName:"")
    + (city!=null?"&city=" + city:"")
    + (address!=null?"&address=" + address:"")
    + (email!=null?"&email=" + email:""))
```

```
+ (userName!=null?"&userName=" + userName:"")
);
```

При това извикване, за да обработи правилно грешката registration.jsp, в него сме добавили код за това.

- От request-а проверяваме дали има грешка:

```
boolean error=false;
try
{
    error = new Boolean(request.getParameter("error")).booleanValue();
}
catch(Exception e){}
```
- Ако има грешка, от сесията се взима съобщението за грешката

```
String errorText = (String) session.getAttribute("errorValue");
```

и се отпечата в страницата на клиента: `<%=errorText%>`
- За всяко поле на формата, ако има грешка и ако това поле е било попълнено с нещо, то в него се изобразява попълнената вече стойност – тя се взима от request-а, който е съставен от **saveprofile** сървлета. Например за полето firstName

```
<input type="text"
name="firstName"<%=error?(request.getParameter("firstName")==null
?"":"" value="\ "" +
HTMLEscape.escape(request.getParameter("firstName")) + "\ """:"%>/>
```

3. Ако няма проблеми от горния вид при регистриране на нов клиент, то регистрацията продължава с проверка дали в базата данни няма вече потребител със зададеното потребителско име (прави се SELECT в таблицата Users) и отново се извиква registration.jsp със съобщение за грешка, ако има такава. Ако не съществува, то новия потребител се записва в базата данни.

Възможни са различни начини за записване в базата данни

- С обикновен Statement обект – при този начин има проблеми с кавичките – за да се образува правилно низа на заявката за записване трябва да (Java) escape-нем всички необходими символи (двойни кавички, апострофи, наклонени черти) в подходящи последователности от символи – това не е много лесно, а и не е необходимо
- С PreparedStatement – горните проблеми ги няма с този вид Statement. Освен това многократното изпълнение на този вид заявка е по-бързо от същия брой изпълнения на Statement заявки.
- С CallableStatement – при този вид Statement като резултат от изпълнението на SQL командата можем да получаваме резултатни параметри.

Основни моменти в PreparedStatement-а за записване на потребител:

```
PreparedStatement st = con.prepareStatement("INSERT INTO
Users(first_name, last_name, city, address, e_mail, user_name,
password) VALUES(?, ?, ?, ?, ?, ?, ?)");
```

```

st.setObject(1, firstName, Types.VARCHAR);
st.setObject(2, lastName, Types.VARCHAR);
st.setObject(3, city, Types.VARCHAR);
st.setObject(4, address, Types.VARCHAR);
st.setObject(5, email, Types.VARCHAR);
st.setObject(6, userName, Types.VARCHAR);
st.setObject(7, password1, Types.VARCHAR);

st.execute();

```

Стойностите на полетата, които ще записваме ги означаваме с въпросителни (те си имат номера). За всяко поле задаваме стойността му – например с метода `st.setObject()`, който има три параметъра – позиция, стойност, SQL тип на полето в което записваме – SQL типовете са описани като константи в класа `java.sql.Types`.

С помощта на метода **`con.setAutoCommit(false)`** указваме на конекцията да не записва автоматично промените. За да се запишат в този случай промените, трябва да се извика директно метода **`con.commit()`**. С помощта на тези методи създаваме един вид транзакция на ниво програмен език. Ако нещо не ни хареса по-време на транзакцията ние може да не запишем промените.

4. След като запишем потребителя в базата данни вземаме цялата информация за него (от базата данни) – включително и автоматично генерирания първичен ключ с помощта на команда **`SELECT`**.
5. На компютъра на клиента се записва cookie, което може да бъде използвано при повторно логване (ще видим как се използва по-надолу).

```

Cookie userCookie = new Cookie("userName", user.getUserName());
userCookie.setMaxAge(60*60*24*30); //time to expire - 30 days
response.addCookie(userCookie);

```

Cookie-то си има име, стойност, време на живот в секунди. Записано по този начин горното cookie ще изчезне автоматично след 30 дена от компютъра на клиента, ако междувременно то не се осъвремени (при повторно логване или друга операция с магазина).

6. В сесията се записват неща, които е необходимо да се знаят в следващи страници – създаваме обект за кошницата:

```

request.getSession().setAttribute("user", user);
request.getSession().setAttribute("basket", new Basket());

```
7. Сървлета извиква индиректно страницата `index.jsp`, която обаче вече съдържа информация за клиента.

5.5. Сървлет login

Логиката на сайта трябва да е доста по-различна, когато потребителя се е идентифицирал (задал е потребителско име и парола). Тогава потребителя освен, че може да разглежда трябва да може да пазарува и да купува. Когато потребителя не е log-нат съответно не трябва да са достъпни тези функции.

При натискане на бутона “Вход” се извиква login сървлета, който извършва следните основни действия:

1. Прочита от request обекта потребителското име и паролата
`String userName = getParameter(request, "userName");`
`String password = getParameter(request, "password");`
2. Създаваме празен обект за кошницата – когато съдържанието на кошницата се променя, това се отразява и в базата данни (но в друга страница – addToBasket).
`Basket basket = new Basket();`
3. Взимаме от базата данни информация за потребителя
`Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
`Connection con = DriverManager.getConnection("jdbc:odbc:storeDB",`
`"sa", "");`
`Statement stmt = con.createStatement();`

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Users WHERE
user_name = '" + userName + "' AND password = '" + password + "'");
```

Ако не съществува информация за такъв човек, сървлета извиква индиректно друга страница (jsp) с параметър за грешка. Освен това записваме и информация в сесията за грешката.

```
request.getSession().setAttribute("errorValue", "Невалидни
потребителско име и/или парола! Задайте потребителско име и
парола отново!");
```

```
response.sendRedirect("index.jsp?error=true");
```

Ако няма такива проблеми се създава обект за потребителя, в който се съхранява необходима информация за него. Този обект се записва в сесията, от където може по-късно да бъде прочетен.

```
user = new User(rs.getInt("user_id"), rs.getString("first_name"),
rs.getString("last_name"), rs.getString("city"),
rs.getString("address"), rs.getString("e_mail"),
rs.getString("user_name"), rs.getString("password"));
```

```
request.getSession().setAttribute("user", user);
```

4. На компютъра на клиента се записва cookie, което може да бъде използвано при повторно логване (ще видим как се използва по-надолу).
`Cookie userCookie = new Cookie("userName", user.getUserName());`
`userCookie.setMaxAge(60*60*24*30); //time to expire - 30 days`
`response.addCookie(userCookie);`
5. Проверяваме дали потребителя няма останали в кошницата продукти от предишно влизане в магазина (полето completed е 0, ако поръчка не е завършена) и ако има те се запълват в обекта за кошницата, създаден на втора стъпка. В този случай на булевата променлива haveOldProducts задаваме стойност true – тя ще се използва в следващата страница (ще се изведе информация за това, че има останали продукти в кошницата от предишно влизане.)

```

rs = stmt.executeQuery("SELECT p.product_id, p.name, p.description,
convert(numeric(10, 3), p.price) price, p.picture_url, m.measure, c.category,
c.category_id, convert(numeric(10, 3), op.quantity) quantity, o.order_id " +
"FROM Products p, Measures m, Categories c,
OrderProduct op, Orders o " +
"WHERE p.measure_id = m.measure_id " +
"AND p.category_id = c.category_id " +
"AND op.product_id = p.product_id " +
"AND op.order_id = o.order_id " +
"AND o.user_id = " + user.getUserId() + " " +
"AND o.completed = 0");

```

```

Long orderId = null;
while(rs.next())
{
    Product product = new Product(rs.getInt(1), rs.getString(2),
rs.getString(3),
rs.getFloat(4), rs.getString(5), rs.getString(6), rs.getString(7),
rs.getInt(8));

    basket.addToBasket(product, rs.getFloat(9));

    orderId = new Long(rs.getLong(10));

    haveOldProducts = true;
}

```

6. В сесията се записват неща, които е необходимо да се знаят в следващи страници:

```

request.getSession().setAttribute("orderId", orderId);
request.getSession().setAttribute("basket", basket);
if(haveOldProducts)
    request.getSession().setAttribute("checkForOldProducts", "1");

```

orderId е номерът на поръчката, за която са останали продукти в кошницата или null ако няма останали продукти.

5.6. Включване на логика за потребител в страниците

5.6.1 index.jsp

Ако има грешка (index.jsp е извикано с параметър за грешка) в index.jsp се изобразява съобщението за грешка. При логване на потребител (login, saveprofile или updateProfile сървлети) се извиква index.jsp. За това в него сме реализирали проверка дали в кошницата има продукти останали от предишно влизане в магазина. Ако има, то се извежда съобщение за това.

Тук също имаме javascript метод focus(), извикван при зареждане на страницата, който обаче позиционира маркера или в полето за потребителско име или в

полето за парола, в зависимост от това дали има или не стойност (взета от cooki-to) в полето за потребителско име.

5.6.2 menu.jsp

Част от менюто не е необходимо да се появява ако няма log-нат потребител (това означава и че в сесията няма записан потребител – програмиста трябва да се грижи записването и отписването на потребител от сесия да се случва правилно).

```
<%
User userM = (User) session.getAttribute("user");
if(userM!=null)
{
%>
<a href="orders.jsp">Поръчки</a><br/>
<a href="profile.jsp">Профил</a><br/>
<a href="logout">Изход</a><br/>
<%
}
%>
```

Ако в сесията има записан потребител се извежда допълнителна част от менюто.

5.6.3 header.jsp

В заглавната част не искаме да се появяват полетата за логване, когато потребителя е вече логнат – тогав визуализираме друго картинка.

Когато пък потребителя не е логнат освен, че трябва да имаме полетата за логване може мда използваме cooki-to, което сме записали на локалния компютър на клиента за да изобразим автоматично в полето за потребителско име, името от cooki-to, ако го има.

Четене на тази информация от cooki-to става със следния код:

```
Cookie cookies[] = request.getCookies();
String userNameCookie = "";
if(cookies!=null)
for(int i=0; i<cookies.length; i++)
if(cookies[i].getName().equals("userName"))
{
userNameCookie = cookies[i].getValue();
break;
}
```

5.6.4 headingInfo.jsp

За новосъздаденото headingInfo.jsp jsp също извеждаме някои неща само ако потребител е логнат.

С помощта на методи на взетия от сесията обект от клас Basket – кошница, непрекъснато визуализираме неща за нея – колко добавяния има и на каква цена са те.

5.7. *profile.jsp* и сървлет *updateprofile*

Когато потребител е логнат, той може да разглежда профила си и да го променя с помощта на *profile.jsp* и сървлет *updateprofile*.

Кода и логиката им са подобени съответно на *registration.jsp* и *saveprofile*.

5.8. Сървлет *logout*

Сесиен обект може да бъде унищожен по различни начини: Затваряне на браузъра, спиране на web сървъра или програмно – с метода на сесийния обект *invalidate()*.

В този сървлет само унищожаваме сесията и зареждаме отново *index.jsp* страницата – при това автоматично ще се създаде нова сесия но вече без информация за потребителя.

5.9. *product.jsp*

При избор на продукт от списък с продукти (визуализиран по един от трите начина описани по-нагоре) се показва информация за продукт.

Най-отдолу на страницата имаме форма с поле за въвеждане на количество и бутон за добавяне на указаното количество от продукта в кошницата. Тук вместо разни проверки да се правят в Java при извикване на action-а на формата (за което ще се изхабят по една заявка и отговор) е даден пример за това как се прави проверка с *javascript* преди да се извика action-а (само на локалния компютър).

```
function onAddToBasket()
{
    var f = parseFloat(document.purchaseForm.quantity.value);

    if(document.purchaseForm.quantity.value != f)
    {
        alert("Моля въведете число за количество от купувания продукт!");
        focus();

        return false;
    }

    return true;
}
```

Този метод се задейства при натискане на *submit* бутон на формата:

```
<form      method="post"      action="addtobasket"      name="purchaseForm"
onSubmit="return onAddToBasket();">
```

Ако метода върне *false* (такава е логиката на *form* обекта), не се извиква action-а, иначе се извиква. Метода ще върне *false* (така е рализиран метода) ако в полето за количество на формата не е въведено число.

Това jsp добавя в сесията и информация за разглеждания продукт, която може да се използва от следващия сървлет.

5.10. Сървлет *addToBasket*

Логиката на добавяне на продукт е следната:

1. Взима се от сесията продукта (записан с предишното jsp), кошницата, потребителя, номера на поръчката – orderId.
2. Ако няма номер на поръчка – то това е първия продукт в кошницата и се записва нов запис в таблицата Orders и от там вече се взима и orderId-то.
3. За текущия orderId, се правят записи в таблицата productOrder като преди това от там обаче се изтриват всички продукти (те са в кошницата и ще се запишат отново ако не ги махнем първо).
4. Извиква се отново product.jsp – връщаме се там от където е извикан този сървлет.

5.11. *basket.jsp*

Ако потребител иска да разгледа съдържанието на кошницата и да купи нещо първо извиква страницата basket.jsp. В нея се визуализира информация за продуктите и има форма с два submit бутона – за купуване и за отказване от поръчките – те извикват един и същи action.

Може и повече възможности да се реализират по принцип: премахване на продукт от кошницата, увеличаване, намаляване количеството на продукт. (Но друг път.)

За да се зарежда винаги наново jsp страницата, дори и при натискане на back бутона на браузъра са използвани следните команди:

```
response.setHeader("Pragma", "No-cache");  
response.setDateHeader("Expires", 0);  
response.setHeader("Cache-Control", "no-cache");
```

При работата със сесията има проблем с проследяването на промените – ако се натиска back бутона може да се озовем на страница, която не отговаря на текущото положение на реалния свят (например отказваме се от покупките, но се връщаме назад с back и там имаме продукти, които решаваме че трябва да купим – обаче в сесията вече няма необходимата информация за тази кошница).

С премахване на кеширането не се решават всички проблеми – възможно е потребител да реши че иска да отваря страниците на сайта в нови прозорци – тогава всеки прозорец ще отразява някакво състояние на реалния свят, но в даден момент, който е различен от настоящия.

Тези проблеми не е много лесно да се разрешат...

5.12. Сървлет *sell*

В зависимост от това кой бутон е натиснат – купуване или отказ трябва да се извърши Update на полето completed за дадената заявка или да се премахне цялата информация за нея от базата данни и от сесията.

Ако е натиснат бутона buy, то е вярно условието *request.getParameter("buy")!=null*.

Ако е натиснат бутона reject, то е вярно условието *request.getParameter("reject")!=null*.

```
Long orderId = (Long) request.getSession().getAttribute("orderId");

if(orderId!=null && request.getParameter("buy")!=null)
{
    stmt.executeUpdate("UPDATE Orders SET completed = 1 WHERE order_id = " + orderId);
}

if(orderId!=null && request.getParameter("reject")!=null)
{
    stmt.addBatch("DELETE FROM OrderProduct WHERE order_id = " + orderId);
    stmt.addBatch("DELETE FROM Orders WHERE order_id = " + orderId);
    stmt.executeBatch();
}
```

В Statement обекта е направен batch – съвкупност от команди, които се изпълняват заедно при извикване на метода *stmt.executeBatch()*.

5.13. Заключение

Никой не може да попречи на потребител директно да запише url в браузъра и директно да извика някоя страница, която не е началната страница на сайта на магазина. Тогава може да се стигне едно от следните положения:

- страницата да се отвори – може да се изведе важна информация – ако потребител не е затворил сесията по някаква причина друг може да влезе с неговите права в магазина.
- да възникне грешка – в момента страниците не са направени така че да улавят и да обработват всички възможни грешки.

6 Упражнение

Да се направи частта от сайта за разглеждане на заявките на потребител.