**SANG Food**

Online Food Order & Delivery System

Object Design

Version 1.0

02.01.2017

Ayse Naz Aydemir

Gamze Saraç

Nazlı Akdağ

H. Selen Kozanoğlu

Prepared for

SE301 Software Engineering



**Table of Contents**

1. Introduction

1.1. Object Design Trade-offs

1.2. Interface Documentation Guidelines

1.3. Definitions, Acronyms, and Abbreviations

1.4. References

2. Packages

3. Class Interfaces

# Introduction

Purpose of the system is to provide an efficient and a reliable online Food Ordering way to all people, so that they can order their food easily. Also, purpose of the system includes providing some functionalities to restaurant owners, admin and managers so that they can manage their restaurants and functionalities.

The SANG Food system has trade off which leads to some costs like as: development, understandability and security.

## Object Design Trade-Offs

Object Design trade-offs leads to some costs like as: development, understandability and security.

* + 1. **Development Cost**

SANG Food provides a lot of functions for users. These are: Add Restaurants, Delete Restaurant , Add Basket , Finish Order, Edit Profile, Register, Login etc. All of these functions of the system requires extra design and this causes an extra cost for the development.

* + 1. **Understandability Cost**

Understandability of the code is too important during the testing phase. Each class and method must be readable, so number of methods increase in the system and functions must be implemented in a clear way. Writing comments into the source code increases the understandability of the code. This causes an additional cost in the developing phase.

* + 1. **Security Cost**

In SANG Food users must be authorized to connect system. However unauthorized user should not be able to access the system. Each user will be able to login to the system by using the username and password .This situation requires an additional cost to the system.

## Interface Documentation Guidelines-JAVADOC

Interface documentation guidelines are the most important. This documentation improves communications between developers.

**Classes**

**UserDAO class:** This class includes all methods , functions about the SangFood System.·

DAO (Data Access Objects) class means that there are Database operations in this

classes.

·

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| UserBean | **Login**(UserBean bean)  -Login method authorize the user who logs in is valid? and checks that is this user customer?or is this user admin? is this user manager? Or is this user restaurant owner? |
| Boolean | **register**(UserBean user)  -Register method provides users to sign up the SANG Food system. If the user is valid on the system the method returns isValid(true) else isValid(false) |
| void | **ChangePassword**(String newpassword, String username)  -All users can change their password. |
| **void** | **addRestaurant**( RestaurantBean restaurant)  -Adds restaurant to the system. |
| void | **deleteRestaurant**(String rest\_id)  -deletes restaurant from the system. |
| void | **addRestaurantForm**(RestaurantBean restaurant)  Users can join the system by filling the restaurant form.  If Manager accept their form they can be recorded to the  restaurants table in SANG Food DB . |
| void | **EditProfile**(String  newlastname,String newemail,  String newaddress,String newphonenumber  String newcity,String newplace,  String username)  -Users can change their information |
| void | **addBasket(**OrdersBean order,String username)  -Adds the customer’s selected food to the basket. This method includes java.sql.Timestamp JavaDateObject to display order date. |
| void | **setRest**(String rid,int status)  -Set incoming restaurants status 1 or 2, if status=1 this means that restaurant accepted,  if status=0 this means that restaurant is waiting, if status=2 this means that restaurant is rejected. |
| void | **setOrder**(int order\_id)  -This method set order\_status=1.  This means that order is waiting  if order\_status is 0, if order\_status=1  order is delivered. |
| String | **getRestaurantId**(String username)  -This method provides get restaurant\_id according to the user’s order restaurant. |
| void | **addFood(**RestMenuBean menu,int restaurant\_id)  -Adds food to the restaurant menu |
| void | **Search(**String rest\_name) |
| void | **EditRestaurantProfile** (String newrestaddress, String newrestinfo,String newphone,int rid)  -This method provides Edit Restaurant Profile |

**1.3 Definitions, Acronyms and Abbreviations**

- Database: Database is a collection of information that is organized so that it can easily be

accessed

- ID : Identification

- Login: to get access to an operating system or application, usually in a remote computer

- ODD : Object Design Document

- UI : User Interface

- JSP : JavaServer Pages

- HTML : HyperText Markup Interface

-CSS: Cascading Style Sheets

-Bean: Beans are normal Java classes containing -Attributes - Getter and Setters.

-DAO: are objects responsible for handling the interactions with the Data Source, through

implementing the access mechanism required to work with the data source.

- Server : is a computer, provides services to other computers

- Servlet : is a Java programming language class that is used to extend the capabilities of

servers that host applications accessed by means of a request-response programming mode

-Session : A session is a semi-permanent interactive information interchange, also known

as a dialogue,between a computer and user. A session is set up or established at a certain

point in time, and then torn down at some later point

- User: a person who use a computer

- Exception: Represents errors that occur during application execution.

- Error: The condition of having incorrect or false knowledge.

## 1.4. References

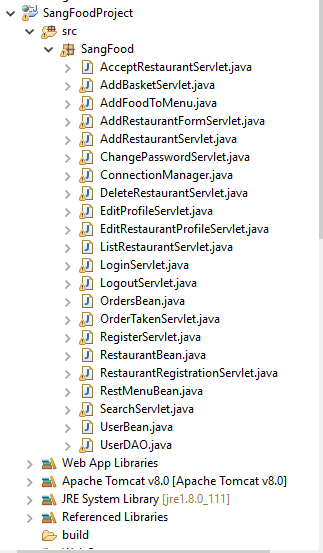
1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.

2. Yemeksepeti web site (https://www.yemeksepeti.com)

3. Lecture presentations of the SE301 course.

**2. Packages**

We have 2 packages in SangFood package we store our Bean,DAO and Connection classes.

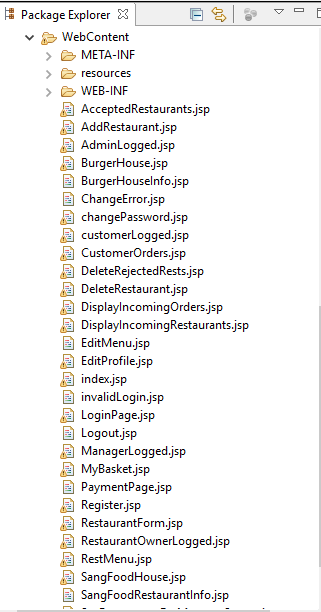


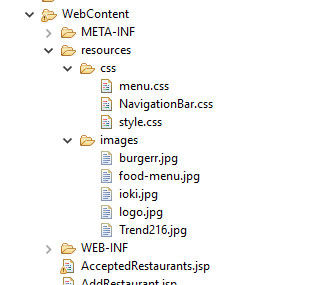
In our project we used MVC Architectural style (Model-View-Controller)

We store our Model in **DAO (UserDAO) class.**

We store our Controller in **Servlet Classes**

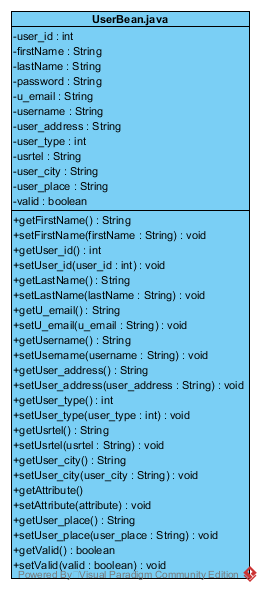
We stored our View in **Web Content folder as a JSP Files and we also store our CSS and Images in resources folder:**



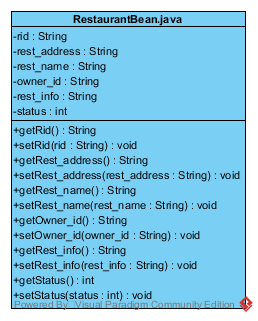


# 3. Class interfaces

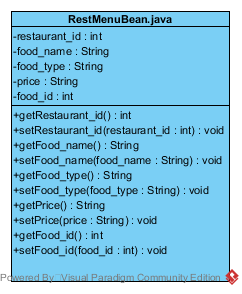
**UserBean.java class:** This class contains all get and set method which belongs user attributes.



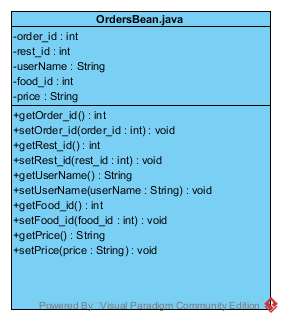
**RestaurantBean.java class:** This class includes all get and set method which are related for Restaurant attributes.



**RestMenuBean.java:** This class includes all get and set methods which are related Restaurant Menu attributes.



**OrdersBean.java:** This class includes all get and set methods which are related Customer Orders attributes.



**-**In our JSP files we called our Servlet classes for specific function that we want to do, from form or button tags. Such as For Login;

**LoginPage.jsp**

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<link rel="stylesheet" href="resources/css/style.css" />

<title>Login Page</title>

</head>

<body>

**<form action="LoginServlet" >**

<center>

<table border="1" cellpadding="5" cellspacing="2">

<thead>

<tr>

<th colspan="2">Login Here</th>

</tr>

</thead>

<tbody>

<tr>

<td>Username</td>

<td><input type="text" name="username" autocomplete="off" required/></td>

</tr>

<tr>

<td>Password</td>

<td><input type="password" name="password" required/></td>

</tr>

<tr>

<td colspan="2" align="center"><input type="submit" value="Login" />

&nbsp;&nbsp;

<input type="reset" value="Reset" />

</td>

</tr>

</tbody>

</table>

</center>

</form>

</body>

</html>

**LoginServlet.java**

package SangFood;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

public class **LoginServlet** extends HttpServlet {

public void doGet(HttpServletRequest request,HttpServletResponse response)

throws ServletException,java.io.IOException{

try{

UserBean user=new UserBean();

user.setUsername(request.getParameter("username"));

user.setPassword(request.getParameter("password"));

user**=UserDAO.**login(user);

if(user.getUserType()==1 && user.isValid()){

HttpSession session=request.getSession(true);

System.out.println(user.getUserId());

session.setAttribute("currentSessionUser", user.getUsername());

session.setAttribute("usertype", user.getUserType());

response.sendRedirect("customerLogged.jsp");

}

else if(user.getUserType()==2 && user.isValid()){

HttpSession session=request.getSession(true);

session.setAttribute("currentSessionUser", user.getUsername());

session.setAttribute("usertype", user.getUserType());

response.sendRedirect("ManagerLogged.jsp");

}

else if(user.getUserType()==3 && user.isValid()){

HttpSession session=request.getSession(true);

session.setAttribute("currentSessionUser", user.getUsername());

session.setAttribute("usertype", user.getUserType());

response.sendRedirect("RestaurantOwnerLogged.jsp");

}

else if(user.getUserType()==4 && user.isValid()){

HttpSession session=request.getSession(true);

session.setAttribute("currentSessionUser", user.getUsername());

session.setAttribute("usertype", user.getUserType());

response.sendRedirect("AdminLogged.jsp");

}

else

response.sendRedirect("invalidLogin.jsp"); //error page

}

catch(Throwable theException){

System.out.println(theException);

}

}}

# 4. Glossary

***Actor***: External entity that needs to exchange information with the system. An actor can represent either a user role or another roles.

***Analysis***: An activity during which developers ensure that the system requirements are correct , complete, consistent , unambiguous , and realistic .

***Class Diagram***: UML notation representing the structure of the system in terms of objects, classes, attributes, operations and associations. Class diagram are used to represent objects models during development .

***Functional Requirement***: An area of functionality the system must support. The functional requirements describe the interactions between actors and system independent of the realization of the system.

***Object Design***: An activity during which developments define custom objects to bridge the gap between the analysis model and the hardware/software platform. This includes specifying object and subsystem interfaces, selecting off the shelf components, restructuring the object model to attain design goals, and optimizing the object model for performance.

***Object Design Document***: A document describing the object design model. The object design model is often generated from comments embedded in the source code.

***Scenario***: Instance of a use case. A scenario represents a concrete sequence of interactions between one or more actors and the systems.

***Sequence Diagram***: UML notation representing the behavior of the system as a series of the interactions among of the group of objects. Each object is depicted as a column in the diagram, Each interaction is depicted as an arrow between two colums. Sequence diagram are used during analysis to identify missing objects, attributes.

***Use Case:*** A general series of interactions between one or more actors and the system.

***Use Case Diagram***: UML notation used during requirements elicitation and analysis to represent the functionality of the system. A use case describes a function of the system in terms of a sequence of interactions between an actor and system.