



# OBJECT DESIGN DOCUMENT



NOMI PARTECIPANTI	MATRICOLE PARTECIPANTI
<i>SCALA ANDREA</i>	<b>0512105</b>
<i>COPPOLA FELICE</i>	<b>0512105600</b>
<i>DE FALCO MARCO</i>	<b>0512105178</b>

# 1 – Introduzione

---

## 1.1 Object design trade offs

### **Comprensibilità vs Tempo:**

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare.

### **Prestazioni vs Costi:**

Dal momento che il budget allocato è spendibile principalmente in risorse umane e non consente l'acquisto di tecnologie proprietarie specifiche verranno utilizzati template open source e componenti hardware di nostra proprietà.

### **Interfaccia vs Usabilità:**

Verrà realizzata un'interfaccia grafica chiara e concisa, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

### **Sicurezza vs Efficienza:**

La sicurezza rappresenta uno degli aspetti principali del sistema. Tuttavia, a causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su e-mail e password.

## 1.2 Componenti off-the-shelf

Per l'implementazione del sistema utilizzeremo componenti self ossia componenti software già disponibili e utilizzati per facilitare la creazione del software. Il framework che utilizzeremo per la realizzazione delle interfacce grafiche delle pagine web è Bootstrap. Per velocizzare i tempi di risposta ed aggiungere animazioni a seguito di click e operazioni, utilizzeremo script JavaScript per visualizzazione di messaggi che rendono la visita dell'utente più semplice e immediata.

## 1.3 Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire precise linee guida per la stesura del codice:

### **Naming Convention**

Per la documentazione delle interfacce bisognerà utilizzare nomi:

- Descrittivi;
- Pronunciabili;
- Di lunghezza medio-corta;
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9);

### **Variabili**

I nomi delle variabili dovranno iniziare con la lettera minuscola, e le parole successive con la maiuscola, secondo la "Camel Notation".



In ogni riga dovrà esserci un'unica variabile dichiarata, eventualmente allineata con quelle del blocco dichiarativo.

In determinati casi, è possibile utilizzare il carattere underscore “\_”: il caso principale previsto è quello relativo alla dichiarazione di costanti oppure di proprietà statiche.

## Metodi

I nomi dei metodi dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola, seconda la “Camel Notation”.

Il nome del metodo sarà costituito da un verbo che ne identifica l'azione seguito da un sostantivo, eventualmente aggettivato.

## Classi Java e pagine JSP

Il nome dei metodi accessori e modificatori seguirà, rispettivamente, i pattern *getNomeVariabile* e *setNomeVariabile*.

I nomi delle classi dovranno iniziare con la lettera maiuscola, così come le parole successive all'interno del nome. I nomi delle classi dovrebbero essere semplici, descrittivi e inerenti al dominio applicativo. Non devono essere usati underscore per legare nomi.

I nomi delle pagine JSP dovranno seguire la notazione camelCase descritta sopra.

I nomi delle classi e delle pagine dovranno corrispondere alle informazioni e le funzioni fornite da quest'ultime.

Le classi saranno strutturate prevedendo rispettivamente:

- Dichiarazione della classe pubblica;
- Dichiarazioni di costanti;
- Dichiarazioni di variabili di classe;
- Dichiarazioni di variabili di istanza;
- Costruttore;
- Metodi;

```
import java.sql.Connection;

public class ClienteDAO {
    private static Pattern pattern;
    public void doSave(ClienteBean c) throws SQLException {

        Connection conn = ConnectionPool.getConnection();
        PreparedStatement ps = conn.prepareStatement("INSERT INTO CLIENTE (NOME, COGNOME, USERNAME, EMAIL, PASSWO) VALUES (?, ?, ?, ?, ?)");
        ps.setString(1, c.getNome());
        ps.setString(2, c.getCognome());
        ps.setString(3, c.getUsername());
        ps.setString(4, c.getEmail());
        ps.setString(5, c.getPassword());

        ps.executeUpdate();
        //Esecuzione query
    }
}
```

## Pagine lato Server (JSP)

Le pagine JSP quando eseguite dovranno produrre un documento conforme allo standard HTML 5. Il codice Java presente nelle JSP deve aderire alle convenzioni già descritte precedentemente.

1. Il tag di apertura (<%>) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga;
3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione (<%=>)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html lang="en">
<head>
<title>ElTanqueWineBar</title>
<meta charset="utf-8">
<link rel="icon" href="src/images/logoicon.png">
<link rel="shortcut icon" href="src/images/logoicon.png">
<link rel="stylesheet" href="src/scss/style.css">
<link rel="stylesheet" href="src/scss/TMGPrototype2.css">
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
    integrity="sha384-J6qa4849bLE2+poT44WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
    crossorigin="anonymous"></script>
<script
    src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"></script>
<script
    src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
    integrity="sha384-wfSDF2E50Y2D1uUdJ03uMBJnJuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
    crossorigin="anonymous"></script>
<script src="src/js/jquery.js"></script>
<script src="src/js/jquery-migrate-1.1.1.js"></script>
<script src="src/js/superfish.js"></script>
<script src="src/js/TMGPrototype2.js"></script>
<script src="src/js/jquery.equalheights.js"></script>
<script src="src/js/jquery.easing.1.3.js"></script>

<script src="https://kit.fontawesome.com/812a03be35.js"
    crossorigin="anonymous"></script>

<link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
```

## Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML 5 e il codice deve essere indentato, per facilitare lettura e modifiche, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione
2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene
3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura
4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali

```

<!DOCTYPE html>
<html lang="en"
      class="fontawesome-i2svg-active fontawesome-i2svg-complete">

<head>
<meta charset="ISO-8859-1">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport"
      content="width=device-width, initial-scale=1, shrink-to-fit=no">
<meta name="description" content="">
<meta name="author" content="">
<title>El Tanque Cliente</title>
<style type="text/css">

<link href="https://fonts.googleapis.com/icon?family=Material+Icons"
      rel="stylesheet">
<link href="resources/dataTables.bootstrap4.min.css" rel="stylesheet"
      crossorigin="anonymous">
<link
      href="https://fonts.googleapis.com/css?family=Arvo:700i|Kaushan+Script&display=swap"
      rel="stylesheet">

<script src="resources/all.min.js.download" crossorigin="anonymous"></script>
<script src="https://kit.fontawesome.com/812a03be35.js"
      crossorigin="anonymous"></script>
<script src="https://code.jquery.com/jquery-3.4.1.js"
      integrity="sha256-WpOohJOqMqqyKL9FccASB900KwACQJpFTUBLTYOVvVU="
      crossorigin="anonymous"></script>
<script>
    $("btn btn-link btn-sm order-1 order-lg-0").click(changeClass);

```

## Script JavaScript

```
{function($){  
$.fn.TMGPrototype2=function(o){  
  
    var getObject = {  
        destination: $('body')  
    ,   controls: true  
    ,   autoPlay: false  
    }  
    $.extend(getObject, o);  
  
    var  
        _this = $(this)  
    ,   _window = $(window)  
    ,   setsList = $('.sets', this)  
    ,   previewSrcArray = []  
    ,   descrArray = []  
    ,   setNamesArray = []  
    ,   currSet = 0  
    ,   currImg = 0  
    ,   urlPreview  
    ,   isPreviewLoading = false  
    ,   isPreviewAnimate = false  
    ,   tmpValue  
    ,   leftWW  
    ,   autoSwitchObj  
    ;  
  
    var  
        _previewHolder  
    ,   _previewSpinner  
    ,   _topImg  
    ,   _bottomImg  
    ,   _categoryList  
    ,   _controlsHolder
```

## Fogli di stile CSS

I fogli di stile devono essere formattati come segue:

1. I selettori della regola si devono trovare nella stessa riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta "{";
3. Le proprietà che costituiscono la regola sono listate per una riga e sono indentate rispetto ai selettori
4. La regola è terminata da una graffa chiusa "}" collocata da sola su una sola riga

```

@import url(http://fonts.googleapis.com/css?family=Pathway+Gothic);
/*      'Pathway Gothic One', sans-serif      */

@import "../css/reset.css";
@import "../css/grid.css";
@import "../css/superfish.css";

input {
    outline: none !important;
    border-radius: 0 !important;
}

html {
    width: 100%;
}

a[href^="tel:"] {
    color: inherit;
    text-decoration: none;
}

* {
    -webkit-text-size-adjust: none;
}

body {
    font: 12px/18px Arial, Helvetica, sans-serif;
    color: #767171;
    position: relative;
    min-width: 960px;
}

```

## Database SQL

I nomi delle tabelle devono essere costituiti solo da lettere minuscole

I nomi devono appartenere al dominio del problema ed esprimere correttamente ciò che intendono rappresentare.

I nomi delle colonne delle tabelle devono seguire la convenzione camelCase o in sostituzione possono contenere il carattere “\_”, ed anche loro devono esprimere correttamente la parte del dominio del problema che intendono rappresentare.

## Design Pattern

### MVC

Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller. Il Model modella i dati del dominio applicativo e fornisce i metodi di accesso ai dati persistenti, il View si occupa della presentazione dei dati all'utente e di ricevere da quest'ultimo gli input, infine il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model.

### DAO (Data Access Object) Pattern

Il DAO pattern è utilizzato per il mantenimento di una rigida separazione tra le componenti Model e Controller, in questo tipo di applicazioni basate sul paradigma MVC

**Data-Access Object:** classe che implementa i metodi degli oggetti che rappresenta

**Classi Bean:** classi che contengono i getters/setters degli oggetti che rappresentano e che saranno usati dai DAO.

## 1.4 Riferimenti

Documento RAD\_ElTanqueWinebar.docx

Documento SDD\_ElTanqueWinebar.docx

# 2 - Packages

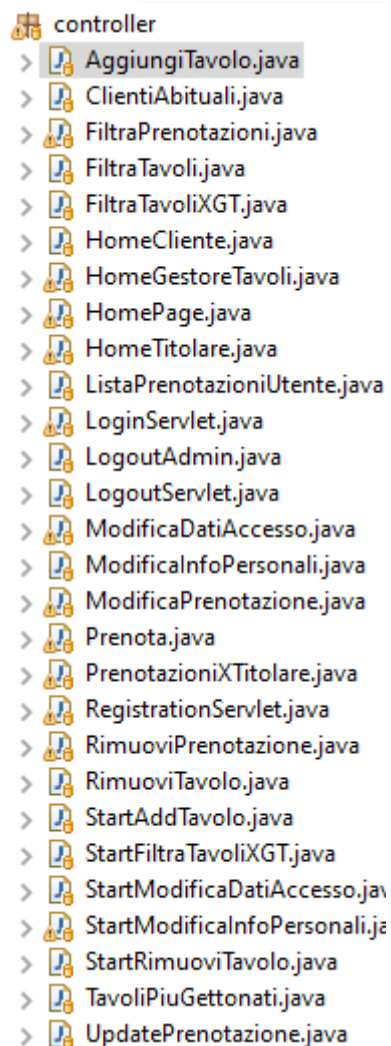
---

## Packages

I nomi dei packages dovranno essere scritti in minuscolo concatenando insieme diversi sostantivi o sigle, separate dal carattere “.”.

Non saranno ammessi caratteri speciali.

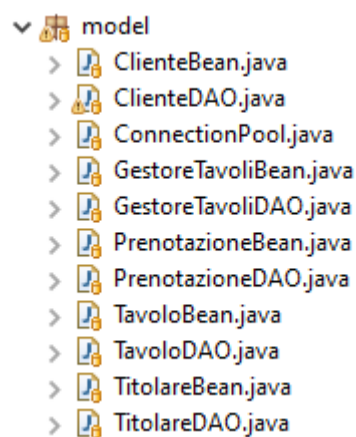
Il package controller formato dalle seguenti classi:



```
controller
> AggiungiTavolo.java
> ClientiAbituali.java
> FiltraPrenotazioni.java
> FiltraTavoli.java
> FiltraTavoliXGT.java
> HomeCliente.java
> HomeGestoreTavoli.java
> HomePage.java
> HomeTitolare.java
> ListaPrenotazioniUtente.java
> LoginServlet.java
> LogoutAdmin.java
> LogoutServlet.java
> ModificaDatiAccesso.java
> ModificaInfoPersonalizzati.java
> ModificaPrenotazione.java
> Prenota.java
> PrenotazioniXTitolare.java
> RegistrationServlet.java
> RimuoviPrenotazione.java
> RimuoviTavolo.java
> StartAddTavolo.java
> StartFiltraTavoliXGT.java
> StartModificaDatiAccesso.java
> StartModificaInfoPersonalizzati.java
> StartRimuoviTavolo.java
> TavoliPiuGettonati.java
> UpdatePrenotazione.java
```

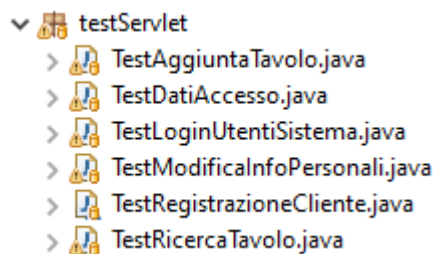


Il package model che contiene le seguenti classi:



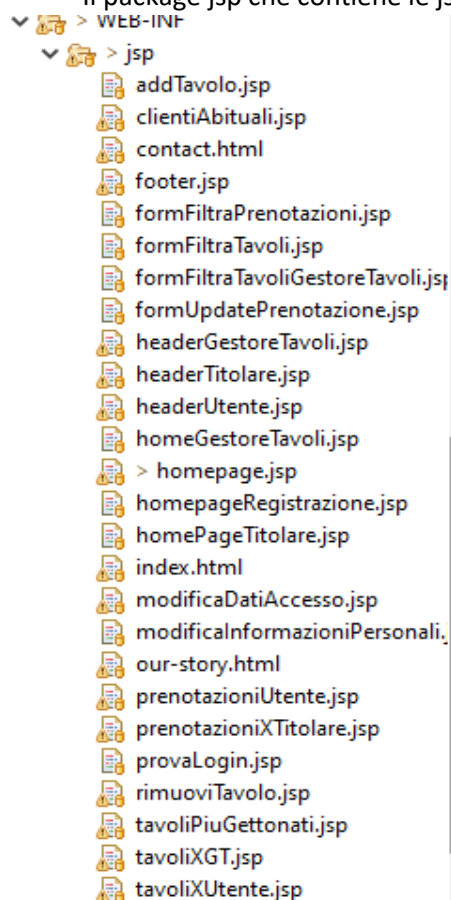
- model
  - ClienteBean.java
  - ClienteDAO.java
  - ConnectionPool.java
  - GestoreTavoliBean.java
  - GestoreTavoliDAO.java
  - PrenotazioneBean.java
  - PrenotazioneDAO.java
  - TavoloBean.java
  - TavoloDAO.java
  - TitolareBean.java
  - TitolareDAO.java

Il package TestServlet che contiene il testing di unità effettuato:



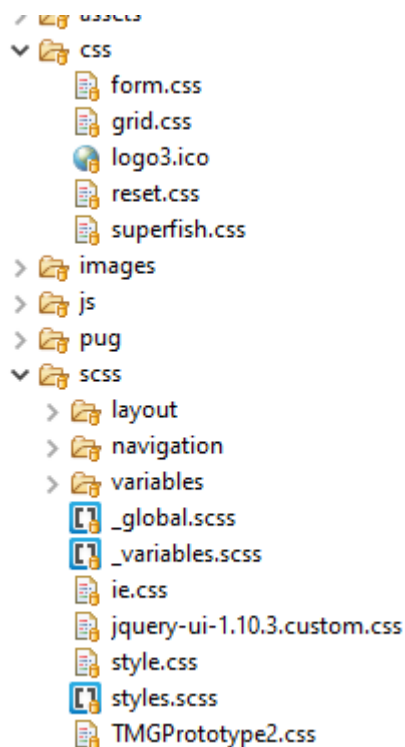
- testServlet
  - TestAggiuntaTavolo.java
  - TestDatiAccesso.java
  - TestLoginUtentiSistema.java
  - TestModificaInfoPersonalizzati.java
  - TestRegistrazioneCliente.java
  - TestRicercaTavolo.java

Il package jsp che contiene le jsp che contribuiscono al funzionamento del sito.



- WEB-INF
  - jsp
    - addTavolo.jsp
    - clientiAbituali.jsp
    - contact.html
    - footer.jsp
    - formFiltroPrenotazioni.jsp
    - formFiltroTavoli.jsp
    - formFiltroTavoliGestoreTavoli.jsp
    - formUpdatePrenotazione.jsp
    - headerGestoreTavoli.jsp
    - headerTitolare.jsp
    - headerUtente.jsp
    - homeGestoreTavoli.jsp
    - homepage.jsp
    - homepageRegistrazione.jsp
    - homePageTitolare.jsp
    - index.html
    - modificaDatiAccesso.jsp
    - modificaInformazioniPersonalizzati.jsp
    - our-story.html
    - prenotazioniUtente.jsp
    - prenotazioniXTitolare.jsp
    - provaLogin.jsp
    - rimuoviTavolo.jsp
    - tavoliPiuGettonati.jsp
    - tavoliXGT.jsp
    - tavoliXUtente.jsp

I package css e scss contengono i fogli di stile usati.



### 3 - Class interfaces

Vincoli Gestione Utente
Questa classe permette registrazione, login e la visualizzazione della home al cliente
<b>Context</b> :Gestione Utente: registrazione(Nome, Cognome, username, email,password); <b>pre</b> : Nome !=null && Cognome !=null && validate(username) && validate(email) &&validate(password) <b>post</b> : Utente registrato e aggiunto al DB
<b>Context</b> : Gestione utente: login(username,password); <b>pre</b> : username !=null && password !=null <b>post</b> : Utente loggato
<b>Context</b> : Gestione Utente homeCliente <b>pre</b> : l'utente deve essere loggato
<b>Context</b> :: Home Cliente Cerca tavoli() <b>Post</b> : restituisce i tavoli cercati dall'utente per la prenotazione
<b>Context</b> : Home Cliente leMiePrenotazioni() <b>post</b> : restituisce le prenotazioni effettuate dall'utente
<b>Context</b> : Home Cliente modificaInfoPersonal() <b>pre</b> : controllo formato dati(nome,cognome)

<b>post:</b> dati modificati
<b>Context:</b> Home Cliente modificaDatiAccesso() <b>pre:</b> controllo formato dati (email,password,username) && esistenza (email,username) <b>post:</b> dati modificati

<b>Nome Classe</b>	<b>GestoreTavoli</b>
<b>Descrizione</b>	Questa classe permette al gestore dei tavoli di aggiungere, rimuovere tavoli dal locale e visualizzarne lo stato
<b>Pre-condizione</b>	<b>Context:</b> GestoreTavoli addTavolo(numeroTavolo,numeroPersone) rimuoviTavolo(numeroTavolo)
	<b>pre:</b> numeroTavolo != null && numeroPersone !=null && numeroTavolo(non esistente) <b>post:</b> Tavolo aggiunto
	<b>pre:</b> numeroTavolo != null <b>post:</b> Tavolo rimosso
	<b>Post:</b> visualizzazione dei tavoli presenti

<b>Nome Classe</b>	<b>Titolare</b>
<b>Descrizione</b>	Questa classe permette al titolare del locale di visualizzare i tavoli più gettonati, i clienti abituali e le prenotazioni in un dato periodo
<b>Pre-condizione</b>	<b>Context:</b> Titolare Clienti abituali, tavoli più gettonati, prenotazioni
	<b>Post:</b> clienti abituali
	<b>Post:</b> tavoli più gettonati
	<b>Pre:</b> data per visualizzazione !=null <b>Post:</b> visualizzazione delle prenotazioni