Alma Mater Studiorum
Università di Bologna

Scuola di Ingegneria

Tecnologie Web T

Simulazione d'esame
Riepilogo Core TWT

Home Page del corso: http://lia.disi.unibo.it/Courses/twt2223-info/
Versione elettronica: L.09.ProvaEsame_Soluzione.pdf
Versione elettronica: L.09.ProvaEsame_Soluzione-2p.pdf

Esercizio di Riepilogo su JS, Servlet, JSP, JSON, …

Esercizio 1 tipologia esame

«Liberamente» tratto da Esame del 16 Giugno 2017

Si realizzi un'applicazione Web, principalmente basata su tecnologie JSP, Java servlet e Javascript, per **l'acquisto di gruppo** di prodotti da un sito di e-commerce.

- Autenticazione tramite username e password
  (non è possibile per uno stesso utente avere più sessioni correntemente attive, ad esempio da dispositivi differenti)
- Ogni utente appartiene ad uno e un solo gruppo, staticamente noto lato server-side (id_gruppo come info già nota)
- Catalogo dei prodotti disponibili via JSON
  (descritti come id_prodotto, breve testo di presentazione, costo, numero di item disponibili)

Pronti, via: Login

- Autenticazione tramite username e password
  (non è possibile per uno stesso utente avere più sessioni correntemente attive, ad esempio da dispositivi differenti)
- Ogni utente appartiene ad uno e un solo gruppo, staticamente noto lato server-side (id_gruppo come info già nota)

Primo punto da affrontare: **Login**

- Servlet/JSP per l'autenticazione
- Modellazione delle entità di business (Utenti, Prodotti, Catalogo) tramite Java Bean
- Inizializzazione delle strutture dati
- Controllo credenziali

Pronti, via: Login - Init

- Modellazione delle entità di business (Utenti, Prodotti, Catalogo) tramite Java Bean

```java
public class GruppoUtenti {


    private String id;
    private Set<User> utenti;
    private Cart carrello;
```

```java
@Override
public void init(ServletConfig conf) throws ServletException
{
    super.init(conf);
    g = new Gson();
    Map<String,User> utentiRegistrati = new HashMap<String, User>();
    User u = new User();
    u.setUserName("pinco pallino");
    u.setPwd("asdasd");
    u.setGroupId("g01");

    utentiRegistrati.put(u.getUserName(), u);


    u = new User();
    u.setUserName("tizio");
    u.setPwd("asdasd");
    u.setGroupId("g02");

    utentiRegistrati.put(u.getUserName(), u);

    u = new User();
    u.setUserName("caio");
    u.setPwd("asdasd");
    u.setGroupId("g02");

    utentiRegistrati.put(u.getUserName(), u);
```

- Inizializzazione delle strutture dati
- Utenti, Gruppi, Items, e Catalogo creati e inizializzati nella init(). Perché qui?!
- Salvataggio nel ServletContext

Autenticazione

```
public void doPost(HttpServletRequest request,
        HttpServletResponse response)
throws ServletException, IOException
{
    String name = request.getParameter("userName");
    Map<String,User> utentiRegistrati = (Map<String, User>) this.getServletContext().getAttribute("u
    User utente = utentiRegistrati.get(name);
    if(utente == null)
    {
        //utente non registrato

        RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
        rd.forward(request, response);
        return;
    }
    if(utente.getPwd().compareTo(request.getParameter("pwd"))!=0)
    {
        //          pwd errata
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
        rd.forward(request, response);
        return;
    } // altrimenti tutto OK, si procede
```

… se tutto ok, si va avanti

Passaggio Dati via JSON

- Catalogo dei prodotti disponibili via JSON
  (descritti come id_prodotto, breve testo di presentazione, costo, numero di item disponibili)

```
HttpSession session = request.getSession();
utente.setSession(session);
Map<String, GruppoUtenti> gruppi = (Map<String, GruppoUtenti>)this.getServletContext().getAttribute("gr
GruppoUtenti gruppo = gruppi.get(utente.getGroupId());
// ci andrebbe un controllo, ma do per scontato che non torni null
session.setAttribute("currentUser", utente);
gruppo.addUserToGroup(utente);

Catalogo catalogo = (Catalogo)this.getServletContext().getAttribute("catalogo");
Item[] itemsInCatalog = new Item[catalogo.getCatalog().size()];
itemsInCatalog = catalogo.getCatalog().toArray(itemsInCatalog);
String strCatalogo = this.g.toJson(itemsInCatalog);
System.out.println(strCatalogo);
session.setAttribute("catalogo", strCatalogo);
session.setAttribute("gruppo", gruppo);

response.sendRedirect("catalogo.jsp");
RequestDispatcher rd = getServletContext().getRequestDispatcher("/catalogo.jsp");

rd.forward(request, response);
```

Esercizio di Riepilogo su JS, Servlet, JSP, JSON, …

Esercizio 1 tipologia esame

«Liberamente» tratto da Esame del 16 Giugno 2017

- Ogni utente potrà selezionare prodotti da inserire nel proprio **carrello di gruppo** e potrà richiedere la finalizzazione dell'acquisto
- Finalizzazione dell'acquisto solo quando **tutti i clienti con sessioni attive di quel gruppo** avranno premuto il pulsante "Finalizza"
- Se tutte le sessioni utente di un gruppo terminano senza completare finalizzazione, i dati contenuti nel carrello di gruppo sono **persi**

Infine, **in caso di finalizzazioni di gruppo "in conflitto"** da parte di gruppi differenti, agli utenti di tale gruppo deve essere inviata la nuova versione del catalogo JSON tramite AJAX alla prossima operazione richiesta *(piggybacking)*

Controllare le Sessioni Attive

- Finalizzazione dell'acquisto solo quando **tutti i clienti con sessioni attive di quel gruppo** avranno premuto il pulsante "Finalizza"
- Se tutte le sessioni utente di un gruppo terminano senza completare finalizzazione, i dati contenuti nel carrello di gruppo sono **persi**

- La sessione è un oggetto Java come tutti gli altri che può essere salvato da qualche parte e controllato il futuro

```
public class User {

    private String userName;
    private String pwd;
    private String groupId;
    private HttpSession session;
    private boolean finalized;

    public User() {

        // TODO Auto-generated construct
        this.session = null;
        this.groupId = "";
        this.userName = "";
        this.pwd = "";
        this.finalized = false;
    }
```

Controllare le Sessioni Attive (2)

```
int sessionCounter =0;
for(User u : gruppo.getUtenti())
{
    if(u.equals(currentUser))
    {
        u.setSession(null);
    }
    if(u.getSession() == null)
    {
        sessionCounter++;
    }
}
if(gruppo.getUtenti().size() == sessionCounter)
{
    //finalizza
    session.setAttribute("success", 2);
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);
    return;
}
else
{
    session.setAttribute("success", 1);
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);
    return;
}
```

..non è l'unico modo, ma uno possibile…

In Caso di Conflitto

Infine, **in caso di finalizzazioni di gruppo "in conflitto"** da parte di gruppi differenti, agli utenti di tale gruppo deve essere inviata la nuova versione del catalogo JSON tramite AJAX alla prossima operazione richiesta *(piggybacking)*

Una possibile soluzione, in caso di **conflitto**, potrebbe essere:

Modificare il DOM di Catalogo.jsp in base ad un valore nella sessione utente per prevedere il caricamento, tramite Javascript/Ajax, del nuovo catalogo

In Caso di Conflitto

```
</head>


Integer err = (Integer)session.getAttribute("errore");
if(err !=null && err ==1)
{



    <body onload="myFunc();">
<%}
  else
  {%>
      <body>
  <%}%>
```

Catalogo.jsp

Catalogue.js

```
function myFunc()
{
    // assegnazione oggetto XMLHttpRequest
        var xhr = myGetXmlHttpRequest();
        if ( xhr )
            pullCatAJAX(xhr);
        else
            pullCatIframe();

}
```

Esercizio di Riepilogo su JS, Servlet, JSP, JSON, …

Esercizio 2 tipologia esame

«Liberamente» tratto da Esame del 16 Giugno 2017

Si realizzi un'applicazione Web di assistenza a turisti pedoni, principalmente basata su tecnologie Javascript e AJAX, per il **prefetching di informazioni geolocalizzate**

- Ogni utente indica la sua posizione corrente
  (viene inserita tramite form; coordinate cartesiane (x,y) espresse in metri)
- Vengono inviate info relative a tutte le attrazioni turistiche visibili dalla posizione corrente dell'utente
  (info lato servitore sulle attrazioni turistiche includono coordinate cartesiane, nome e semplice testo di descrizione)
- Ogni attrazione turistica è visibile se e solo se la sua **distanza** dall'utente è minore di k metri *(con k parametro configurabile a livello di applicazione)* e se ci troviamo in **situazione non affollata** *(meno di 10 altri utenti nel raggio di 100 metri)*

Esercizio di Riepilogo su JS, Servlet, JSP, JSON, …

Esercizio 2 tipologia esame

«Liberamente» tratto da Esame del 16 Giugno 2017

Per ottenere un comportamento fortemente responsive dell'applicazione Web, si faccia in modo che il browser Web faccia il **download in background**, in modo asincrono e concorrente, delle informazioni relative alle attrazioni che saranno eventualmente visibili nella **posizione futura più probabile per l'utente**

Per semplicità, si supponga che tale posizione futura più probabile sia sempre (x2,y2) = (x1+50, y1)

Turisti: Lato Server

- Ogni utente indica la sua posizione corrente
  (viene inserita tramite form; coordinate cartesiane (x,y) espresse in metri)
- Vengono inviate info relative a tutte le attrazioni turistiche visibili dalla posizione corrente dell'utente
  (info lato servitore sulle attrazioni turistiche includono coordinate cartesiane, nome e semplice testo di descrizione)
- Ogni attrazione turistica è visibile se e solo se la sua **distanza** dall'utente è minore di k metri *(con k parametro configurabile a livello di applicazione)* e se ci troviamo in **situazione non affollata** *(meno di 10 altri utenti nel raggio di 100 metri)*

Lato Server l'applicazione non è particolarmente complicata, ma richiede:

- *Inizializzazione dei dati -> Attrazioni Turistiche (come nell'esercizio 1)*
- *Calcolare la distanza tra l'utente e le attrazioni turistiche*
- *Calcolare la distanza tra gli utenti -> zona affollata*

Turisti: Lato Server (2)

```
double x = Double.parseDouble(request.getParameter("x").trim());
double y = Double.parseDouble(request.getParameter("y").trim());
System.out.println("Le Coordinate ricevute sono: "+x+" "+y);

List<Turista> lt = (ArrayList<Turista>)this.getServletContext().getAttribute("turisti");
List<AttrazioneTuristica> attrazioniVisibili = new ArrayList<AttrazioneTuristica>();
Turista t = new Turista();
t.setX(x);
t.setY(y);
HttpSession session = request.getSession(false);
if(session == null)
{
    lt.add(t);
}
int touristCounter =0;
for(Turista tt : lt)
{
    if(calcolaDistanza(x,y,tt)<
        touristCounter++;
}
```

```
if(touristCounter<=11)
{
    for(AttrazioneTuristica att : at)
    {
        if(calcolaDistanza(att.getX(),att.getY(),t)<=K)
        {
            attrazioniVisibili.add(att);
        }
    }

    AttrazioneTuristica[] risultato = new AttrazioneTuristica[attrazioniVisibili.size()];
    risultato = attrazioniVisibili.toArray(risultato);
    String res = gson.toJson(risultato);
    System.out.println(" "+res);
    response.getWriter().println(res);
    return;
}
```

Turisti: Lato Client

Per ottenere un comportamento fortemente responsive dell'applicazione Web, si faccia in modo che il browser Web faccia il **download in background**, in modo asincrono e concorrente, delle informazioni relative alle attrazioni che saranno eventualmente visibili nella **posizione futura più probabile per l'utente**

Per semplicità, si supponga che tale posizione futura più probabile sia sempre (x2,y2) = (x1+50, y1)

Il vero core dell'applicazione è lato client, con le chiamate in Ajax … ma niente di drammatico ☺

Turisti: Lato Client (2)

```
<body>
    <div>
        X: <input type="text" id="x" size="5" onfocusout="myFunction(myGetElementById('risProbabili'))"></in
        Y: <input type="text" id="y" size="5" onfocusout="myFunction(myGetElementById('risProbabili'))"></in
        <input type="button" id="prefetch" value ="Prefetch Touristic Points" onclick="myFunction(myGetEleme
    </div>

    <div>
        Attrazioni Turistiche Visibili Trovate: <br>
        <ul id="risVisibili">
        </ul>
    </div>
    <div>
        Attrazioni Turistiche Pobabilmente Visibili Trovate: <br>
        <ul id="risProbabili">
        </ul>
    </div>
```

Si definiscono due aree (div) per i risultati

- *le attrazioni che si vedono*
- *le attrazioni che si vedranno nella posizione futura*

Tourist.js

- *Si fanno i controlli sulla correttezza degli input*
- *Se devo fare il prefetch della posizione futura, modifico le coordinate secondo la posizione futura*
- *Le chiamate al server sono in AJAX naturalmente*

```
ction myFunction(element)

var y = myGetElementById('y').value;
var x = myGetElementById('x').value;
if(x == "" || y == "")
    return;
if(element.id == 'risProbabili')
    {
        x = parseInt(x)+50;
    }
var pattern = /[0-9]{1,3}/;
if(pattern.test(x) && pattern.test(y))
    {
    var xhr = myGetXmlHttpRequest();

    if ( xhr )
        prefetchPointOfInterestAJAX(x, y, xhr, element);
    else
        prefetchPointOfInterestIframe();
    }
```

- *La chiamata AJAX è quella standard in GET*
- *Nella callback semplicemente si aggiungono le Attrazioni Touristiche ricevute in JSON, nella relativa area DIV*

Tourist.js - ParsificaJSON

```
function parsificaJSON( jsonText ) {

        var item = JSON.parse(jsonText);

        itemNodes = new Array(),

        risultato = "";

    // ciclo di lettura degli elementi
    for (    var a = 0, b = item.length;    a < b;    a++    ) {
        risultato += '<li>' + item[a].nome+', ('+item[a].x+','+item[a].y+'), '+ item[a].nome+ ",
    }
     return risultato;
}// parsificaJSON()
```

Si aggiunge all'InnerText della UL nella DIV di riferimento una lista di item, uno per ogni attrazione turistica ricevuta dal server