Appello di Introduzione alla Programmazione Web del 3 settembre 2020

Un'applicazione web implementa un sistema di annotazione sociale di immagini. Lo scopo è che una coppia di giocatori, collegati da due browser diversi, usino una stessa parola per definire un'immagine.
Data una coppia di giocatori, a entrambi verrà mostrata una stessa immagine (scelta casualmente tra un set di immagini predefinite). Ciascun giocatore dovrà fornire (una alla volta) delle parole chiave che descrivono l'immagine. Quando un giocatore aggiungerà una parola che è già stata usata dall'altro, l'obiettivo sarà stato raggiunto e il gioco termina: ai due giocatori verrà presentato il messaggio "Hai vinto, la parola magica è "+ la parola vincente.
Il sistema deve poter lavorare simultaneamente con un arbitrario numero di coppie di giocatori.

Nella fase iniziale, il primo giocatore di una coppia che si registra riceve una pagina che dice di aspettare, e che quando il secondo giocatore si sarà registrato caricherà la pagina del gioco. Il secondo giocatore della coppia riceve invece immediatamente la pagina del gioco.

Sarà necessario usare due diversi browsers (es. Chrome e Firefox) per simulare la presenza simultanea di due diversi utenti.

(Chi fosse incuriosito, dopo l'esame veda https://www.cs.cmu.edu/~biglou/Peekaboom.pdf)

Nel corso dell'esame gli studenti dovranno implementare ***parzialmente*** la webapp, seguendo la presente traccia.

1) Si crei la classe Coppia, che contiene:
- utente1 (nome del primo utente)
- utente2 (nome del secondo utente)
- lista1 (lista delle parole inserite dall'utente 1)
- lista 2 (lista delle parole inserite dall'utente 2)
- image (url dell'immagine che verrà mostrata ai due utenti).

2) Si crei un oggetto WaitingRoom che può contenere una coppia.

3) Si crei un oggetto MappaDiCoppie (ad esempio sottoclassando java.util.Hashmap) destinato a contenere coppie chiave-valore. Le chiavi saranno di tipo Stringa, i valori di tipo Coppia.

4) Si aggiungano alle tre classi create i metodi che si riterranno via via necessari. Tra questi, vi dovrà essere il metodo **toString** in modo da mostrare lo stato dell'istanza su cui verrà chiamato.

5) Si crei la Pagina di Welcome:
In questa pagina l'utente fornisce in proprio nome, che verrà inviato alla servlet SetUser.

6) Si crei la servlet SetUser.

Al caricamento, la servlet dovrà inizializzare il contesto della webapp (si vedano i richiami alla servletContext in appendice) inserendovi una WaitingRoom e una MappaDiCoppie, che saranno quindi disponibili a tutte le servlet dalla WebApp.
Quando viene chiamata, la servlet SetUser:

- Ricorda il nome dell'utente, inserendolo in una sessione
- Verifica se la WaitingRoom è vuota:
    - se sì crea una Coppia, inserendo lo username nella sua variabile user1. Inserisce la coppia nella WaitingRoom, poi chiama wait.jsp.
    - ne no, inserisce lo username nella variabile user2 della Coppia contenuta nella WaitingRoom. Aggiunge la Coppia a mappaDiCoppie, rimuove la stessa dalla WaitingRoom. Poi chiama game.jsp.

Nota: in prima battuta si creino dei mockup di wait.jsp e game.jsp che non fanno altro che stampare il messaggio "Chiamato "+nome della jsp.

7) Testing-debugging:
Al fine di effettuare test e debug, risulterà utile costruirsi una pagina HTML chiamata monitor che contenente i seguenti link:

```
<li><a href="index.html" target="_blank">start page</a></li>
<li><a href="showUserName.jsp" target="_blank">show session</a></li>
```
(una jsp che stampa lo username corrente)
```
<li><a href="showContext.jsp" target="_blank">show context</a></li>
```
(*una jsp che recupera e stampa lo stato di WaitingRoom e MappaDiCoppie)*
```
<li><a href="showCoppia.jsp" target="_blank">show coppia</a></li>
```
(una jsp che stampa la coppia a cui l'utente appartiene, se la coppia esiste in MappaDiCoppie)
```
<li><a href="clearSession.jsp" target="_blank">clear session</a></li>
```
(una jsp di servizio che distrugge la sessione corrente)
```
<li><a href="resetAll.jsp" target="_blank">reset</a></li>
```
(una jsp di servizio che elimina WaitingRoom e MappaDiCoppie dal contesto, e li ricrea)

8) Si implementi la servlet CheckIfReadyToStart.
Questa servlet recupera dal contesto MappaDiCoppie, e vi cerca il valore corrispondente allo username dell'utente. Se questo è null, significa che l'utente è ancora in WaitingRoom, altrimenti significa che il secondo utente è arrivato.
Sulla base di queste informazioni, la servlet restituisce un JSON contenente "ready":true oppure "ready":false.

9) Si implementi wait.jsp.
Questa jsp dice semplicemente "Aspetta che arrivi un nuovo giocatore 0". La pagina generata dovrà effettuare un refresh automatico (si veda il suggerimento sul polling in Appendice) inviando via Ajax ogni 2 secondi al server la richiesta di eseguire la servlet CheckIfReadyToStart. Quando la risposta sarà "ready":true, caricherà la pagina game.jsp. Se invece la risposta sarà "ready":false, la pagina aggiornerà il campo "Aspetta che arrivi un nuovo giocatore 0" incrementando il numero finale, e proseguirà con il refresh automatico.
Quindi ogni due secondi la pagina sia autoaggiornerà incrementando detto valore, fino a quando CheckIfReadyToStart risponderà "ready":true. A quel punto verrà caricato game.jsp.

10) Si crei la pagina di gioco
La pagina di gioco presenterà:

- L'immagine (scelta a caso tra quelle disponibili – si scarichi dal sito dell'esame il file Immagini.zip)
- Un campo per l'inserimento di una nuova parola chiave (sotto l'immagine)
- La lista di tutte le parole chiave inserite dall'utente, elencate in colonna a destra dell'immagine

L'inserimento di una nuova parola comporterà una modifica della pagina stessa (senza ricaricamento, ma passando per una interazione con il server), i cui effetti saranno:

- La nuova parola viene aggiunta alla lista delle parole chiave inserite dall'utente.
- Si controlla se la lista dell'utente 1 e la lista dell'utente 2 abbiano delle parole in comune. Se si trova una parola in comune, il campo per l'inserimento di una nuova parola chiave viene sostituito dal messaggio: "Hai vinto, la parola magica è "+ la parola in comune.

Alla chiusura della pagina di gioco per un utente, l'associazione tra l'utente e l'oggetto Coppia in MappaDiCoppie viene rimossa.

APPENDICE

Richiami sulla ServletContext:

ServletContext is an object from Servlet world. It is a shared repository for all servlets belonging to a web application. There is only one ServletContext per web application (so it is shared by all servets and all users).

The ServletContext object is the same as the "application" object in JSPs belonging to the same web application. (https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm)

It can be acquired by calling

```
getServletContext();
```

in the init, destroy, doGet, doPost, doFilter and all other doX methods.
(https://docs.oracle.com/javaee/5/tutorial/doc/bnagl.html)

Once you have the ServletContext object, you can add to/retrieve from it objects like you do in an HttpSession, by using the methods

void setAttribute(String name, Object object)
Object getAttribute(String name)

Suggerimenti sul polling

For a simple webapp that needs to refresh parts of data presented to the user in set intervals, are there any downsides to just using setInterval() to get a JSON from an endpoint instead of using a proper polling framework?

I would use setTimeout [docs] and always call it when the previous response was received. This way you avoid possible congestion or function stacking or whatever you want to call it, in case a request/response takes longer than your interval.
So something like this:

```
function refresh() {
    // make Ajax call here, inside the callback call:
    setTimeout(refresh, 5000);
    // ...
}

// initial call, or just call refresh directly
setTimeout(refresh, 5000);
```