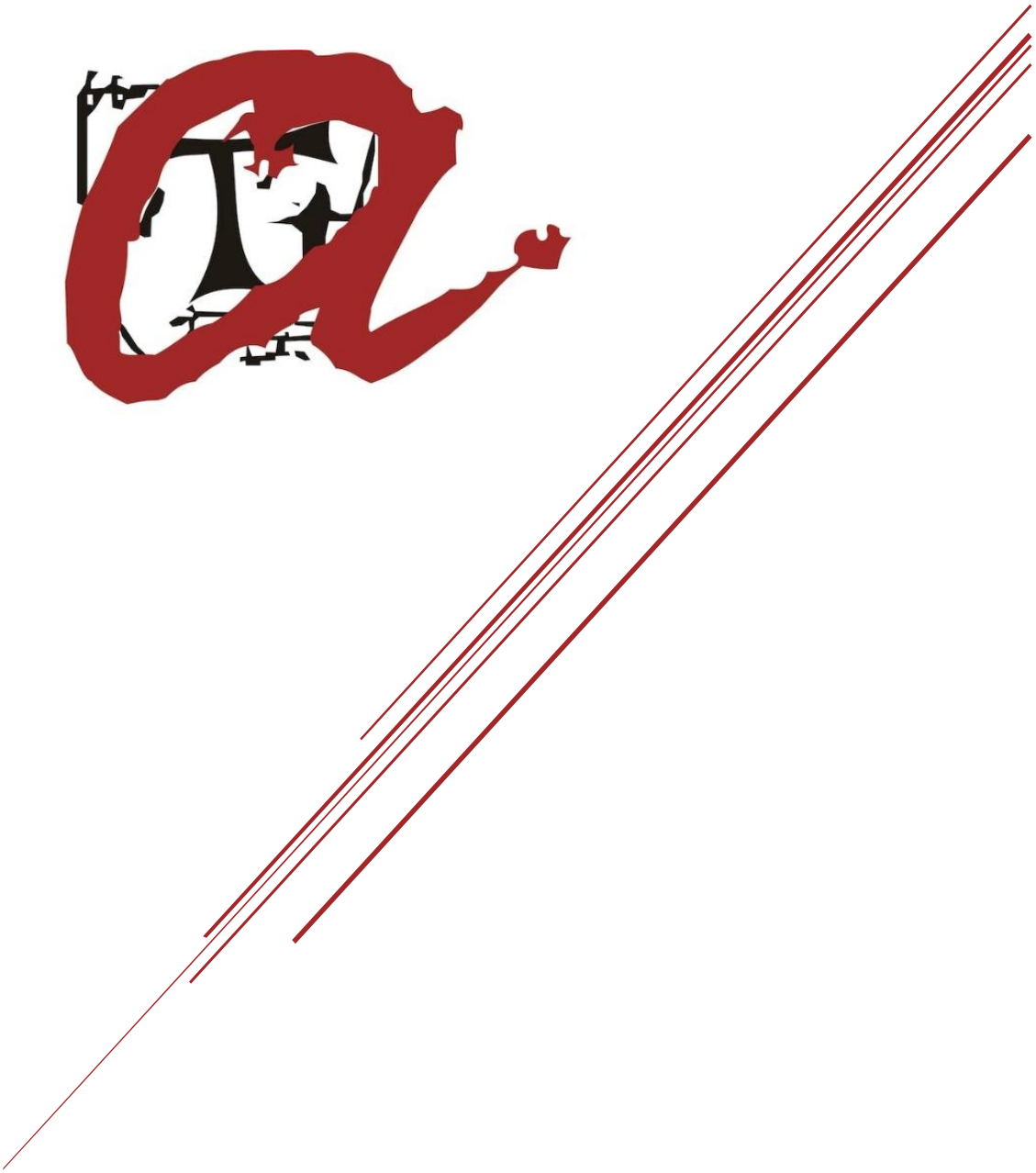
PRÀCTICA 2: API WEB PER A LLOGAR HABITACIONS

Backend i Frontend



Francesc Ferré Tarrés
Aleix Sancho Pujals

Índex

0.Introducció

En aquesta pràctica el que s'ha fet és realitzar una pàgina web completa, és a dir, aprofitar l'API REST realitzada en la primera pràctica i modificar-la per tal de complir amb les accions que es demanen en aquesta segona pràctica.

Aquesta pràctica té dues parts diferenciades:

- ***Part pública***: En aquesta els clients web poden cercar les habitacions per ciutat, per tant hi ha una barra de cerca i un botó en el qual quan pitgem al botó de cerca els hi va a una nova pàgina en la qual apareix el resultat de la cerca.

  Independentment de si escriu alguna cosa o no a la barra de cerca, aquesta porta a una pàgina on hi ha les respostes. En aquí hi ha diferents possibilitats:
  - *No ha trobat habitacions per la ciutat indicada:* En aquest cas la pàgina mostra un text dient que no hi ha habitacions per a aquella ciutat introduïda.
  - *Ha trobat habitacions per a la ciutat indicada:* En aquest cas apareix un llistat de totes les habitacions per la ciutat en concret.
  - *No hi ha cap entrada i dóna al botó de cerca*: En aquest cas, apareixen totes les habitacions ordenades de forma descendent.

  Quan es mostren totes les habitacions hi apareix un botó que ens permet extreure més informació referent per aquella habitació en concret. És en aquesta pàgina en la qual si el client web està autenticat, és a dir, ha iniciat sessió, apareix un botó que permet llogar.

- ***Part privada***: Aquesta té validesa sí i només sí el client web ha estat autenticat, és a dir, ha iniciat sessió de forma satisfactòria.

  L'avantatge que té és que tal com s'ha comentat en el punt anterior, apareix un botó que permet llogar l'habitació a algun dels arrendadors disponibles. Aquest haurà de tenir màxim 3 habitacions llogades, no pot tenir-ne més. Per tant, només podrà llogar fins a 3 habitacions al dia, al passar de dia, ja podrà tornar a comunicar-se.

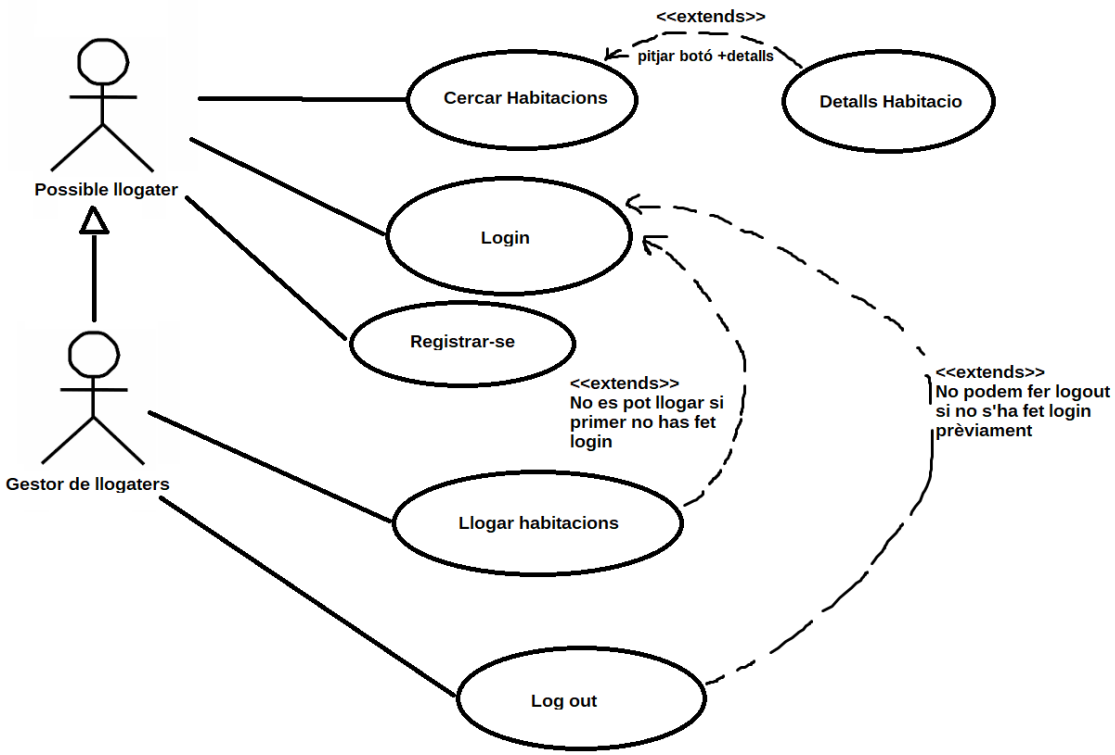
En el moment en què el client web ha iniciat sessió apareix un text donant la benvinguda a l'usuari en totes les pàgines i si l'inici de sessió és correcte, retorna sempre a la pàgina web anterior a la que s'ha fet l'inici de sessió.

Es dóna l'opció de fer "log out", sempre que el client web vulgui.

1. Decisions de disseny

La principal decisió de disseny presa ha estat la de determinar l'autenticació. Si bé és cert que hi ha diferents tipus d'autenticació s'ha optat per a controlar qui es connecta contra la nostra web. Per tal d'entendre-ho millor seria com que realment no són els possibles llogaters qui lloguen directament i qui s'autentiquen contra la web, sinó que seria una persona que gestiona les habitacions qui lloga per a ells.

Per tal de veure-ho més clar, s'inclou el diagrama de casos d'ús:



El cas de registrar permet transformar-se en gestor de llogaters i poder iniciar sessió, sempre que es compleixin certes condicions, com per exemple, que no hi hagi ningú amb el mateix nom d'usuari i format de mail correcte.

Un altra decisió de disseny important és que s'utilitza el Boostrap per tal de generar l'estil de la pàgina mitjançant el CSS (Cascade Style Sheet) i també pels petites icones que hi ha fem servir els Font Awesome. Tot ho traiem d'un fitxer CSS situat la carpeta Styles.

2. Estructura de la pràctica

La pràctica està dividida en dos projectes diferents, un projecte representa el backend i l'altre és la connexió amb el back end i el front end.

Per tal de no eliminar i afegir elements nous de la pràctica 1 es va optar per copiar el projecte sencer i generar un projecte nou, el qual s'anomena *RESTServiceForPract2SistemesOberts* (disponible fent clic aquí).

Aquest projecte representa totes les API REST disponibles a la pràctica 1 però amb millores o noves accions que en la pràctica 1 no estaven disponibles.

Una de les coses que s'han tocat són les classes entitat, i per tant la base de dades:

- ***Habitació***: En aquest cas s'han inclòs noves columnes en la base de dades. Ara s'inclou un URL que representa la imatge, una booleana que ens indica si està ocupada o no i hi hem afegit un nom.
- ***Llogater***: En aquest cas s'ha inclòs un comptador del nombre d'habitacions llogades.
- ***Client web***: En aquest cas s'ha inclòs una booleana que ens indica si ha estat autenticat o no.

Els serveis nous que hem inclòs i que no estaven en la pràctica 1 són els següents:

- **API REST Habitació**:
  - ***GET /webresources/room/allRooms***: En aquesta versió de l'API REST de les Habitacions hem optat perquè hi hagi un servei específic que retorni totes les habitacions en cas que l'usuari no posi cap localització en específic. En cas que posi una localització doncs s'executa el de sempre.
- **API REST Llogater:**
  - ***POST /webresources/tenant/{id}/rent:*** En aquesta versió s'ha modificat una mica el comportament d'aquest POST. El fet d'introduir nous camps ha fet que s'inclogui més codi. Aquest és referent a l'actualització del comptador de les habitacions llogades pel llogater en qüestió i incrementar-lo en 1, a més també cal actualitzar el llogater de l'habitació, posar a cert el camp "ocupada", fet que en la versió 1 no calia.
  - ***PUT /webresources/tenant/nouDia:*** Aquest és un servei nou en l'API REST dels llogaters, ja que en la versió 1 no tenia cap sentit tenir-lo. Aquest fa que en passar de dia s'executi i actualitzi el comptador de llogades a zero pel llogater passat.
- **API REST Client Web:**
  - ***POST /webresources/autenticacio/newClient:*** Aquest servei permet el registre a la nostra pàgina web.
  - ***PUT /webresources/autenticacio/logoutUser:*** Aquest servei nou permet que una persona que prèviament ha iniciat sessió pugui sortir de la sessió.
  - ***GET /webresources/autenticacio/infoClientWeb:*** Aquest servei serveix per a agafar tota la informació referent a un client en específic.

La resta de serveis REST que en la pràctica 1 tenien utilitat i en la pràctica 2 no, s'han eliminat, ja que no s'utilitzen en cap moment.

Pel que fa a la pràctica 2 en si, està inclosa en el projecte anomenat *Pract2_SistemesOberts* (disponible fent clic aquí).

Aquest projecte està basat en l'arquitectura J2EE, en concret és una aplicació web construïda en Java en la que hi ha un servlet que rep peticions HTTP i s'encarrega d'executar el codi necessari per a respondre a la petició en qüestió.

Es fa servir el patró command i el MVC (Model View Controller).

El primer patró permet desacoblar les crides HTTP del codi Java que hi ha al darrere, mentre que el patró MVC és conseqüència del primer.

Per tal de mantenir aquests patrons dins del projecte, tenim un servlet principal que conté totes les URL disponibles i totes les classes Java que serveixen per a processar la petició en qüestió. Aquesta informació s'extreu d'un document anomenat *mvc-setup.json* sota el directori /Pract2_sistemesOberts/web/WEB-INF[1].

L'estructura d'aquest JSON és bàsicament un mapatge de l'URL i la classe que la processa:

```
{
        "URI" : "url disponible",
        "class" : "package.classe que el processa"
}
```

Aquesta informació l'emmagatzema el servlet principal (el servlet que rep totes les peticions HTTP) en el seu mètode init(). D'aquesta forma les dades estan emmagatzemades i quan posem l'URL en qüestió ja sap en quina classe cridar.

El tipus de mapatge d'aquest servlet és per extensió, això vol dir que si mirem la configuració en el descriptor de desplegament (web.xml, també situat a la carpeta WEB-INF) observem que rep com a paràmetre el fitxer JSON descrit en les línies anteriors i també que mapeja per extensió (*.do).

Això vol dir que aquest servlet funcionarà sempre que en l'URL de la petició es rebi:

localhost:8080/Pract2_SistemesOberts/*.do

En aquest cas l'asterisc fa referència a l'URI que està definida en el document JSON, en altre cas, el servlet no sabrà que processar i s'executarà la pàgina d'error definida per defecte en el projecte base disponible al Moodle.

D'acord amb el patró MVC, les parts que hi ha implementades es divideixen segons si són la vista, el model o el controlador:

[1] El fet que es trobi dins el directori WEB-INF fa que no sigui accessible des de l'exterior. Recordem que tots els fitxers ho són excepte el que estan sota aquest directori. Per tant, és informació confidencial.

- **Model**: Tot el que fa referència a la base de dades. En aquí hi podríem incloure tots els clients REST que hem generat mitjançant el Jersey, les classes Java que s'han creat per a processar les peticions, les classes entitat (Java persistence, JAXB, JAX-RS).
- **Vista**: Arxius JSP de sortida i també d'entrada de dades, que a part de fer això donen accés a la lògica de negoci, és a dir, a totes les accions que en la web es poden dur a terme.
- **Controlador**: Servlet principal recollit a /ModelEntities/ControllerServlet.java i les classes Java addicionals creades per a processar la petició.

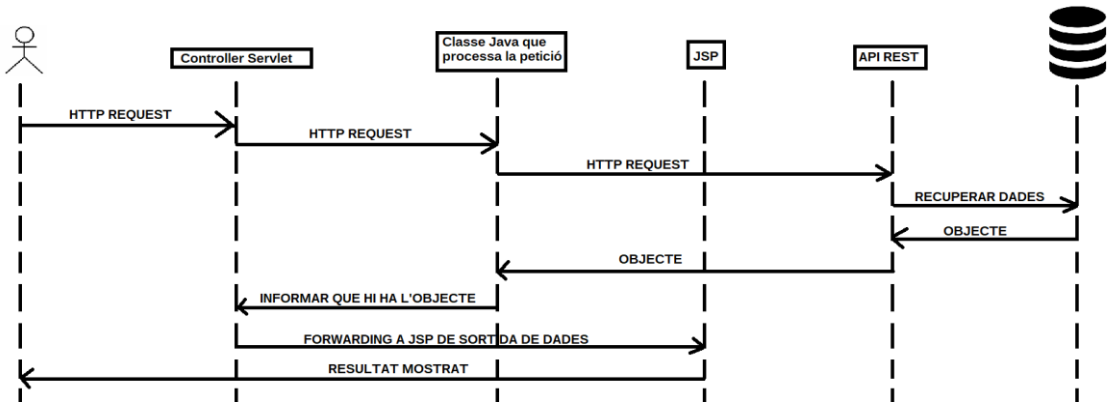
Per tal de poder recuperar les dades del backend (API REST) s'han generat clients REST amb la tecnologia Jersey. D'acord amb la nostra API REST, tenim tres clients diferents: clients web, habitacions i llogaters.

En aquests clients REST, situats sota /restClients hi ha en cadascun les crides corresponents a l'API REST corresponent.

Per tal de generar menys instàncies d'aquests clients, s'ha utilitzat el patró Singleton. D'aquesta forma només tenim una única instància del client en tot el projecte. Aquests es troben sota el Package /ServicesSingleton.

Les classes Java que processen la petició són les encarregades de cridar al backend (gràcies al client REST generat), recuperar la informació, processar-la i extreure-la cap a un JSP programat per a mostrar les dades.

D'aquesta forma tota la seqüència d'operacions està completa i funcional. Per tal que quedi més clar, s'inclou el diagrama de seqüències a continuació:



De forma esquemàtica es pot veure com funciona la pràctica. En cas que hi hagi un JSP d'entrada, la petició la generaria un formulari o algun botó, però pel darrere funcionaria de la mateixa forma.

Per tal que aparegui el JSP amb les dades a mostrar, cal que el servlet sàpiga que ja n'hi ha de dades; per fer-ho el que es fa és la següent línia de codi:

```
ServletContext context = request.getSession().getServletContext();
context.getRequestDispatcher("exemple.jsp").forward(request,response);
```

D'aquesta forma el servlet envia la resposta al JSP de sortida de dades.

Per tal que la recuperació de dades de l'API REST sigui efectiva, es necessiten els Java Beans. Aquestes són còpies de les entitats que hi ha en la base de dades però només amb getters/setters, atributs que mapejen les columnes de la base de dades, constructor buit i que implementen la serialització. Aquesta és la raó que podem utilitzar les dades que es troben emmagatzemades a la base de dades.

Un cop les tenim, per a enviar-les cap als JSP s'utilitza l'àmbit o "scope" request, ja que aquella informació ha de deixar de tenir validesa tan bon punt s'acabi la petició. Per fer-ho afegim l'atribut en aquest àmbit per mitjà del mètode `setAttribute(key,value)`.

Llavors en els JSP utilitzem les Expression Language (la seva anotació és ${atribut}). Per exemple, imaginem que es retorna una habitació, llavors en la classe Java s'afegiria en l'àmbit request l'atribut "habitacio", com a clau i al valor l'objecte recuperat de la base de dades:

```
request.setAttribute("habitacio", resposta.readEntity(Habitacio.class));
```

Nota: La resposta a l'execució d'un client REST és de tipus Response, per a extreure les dades que retorna, es fa executant el mètode `readEntity()` i passant-li la classe que retorna.

Llavors, en el JSP, seria accessible escrivint: `${habitacio}`, i si vulguessim alguna cosa en concret, s'aplica l'anotació per punts, per exemple, si volem, el nom de l'habitació es faria: `${habitacio.nomHabitacio}`.

Les classes Java que s'utilitzen es divideixen en diferents packages, en funció de quin element es tracta:

- Package "**ConnectionToJSPAuthentication**":
  - ***LogoutUser.java***: És la classe responsable de fer un logout d'un client web que prèviament ha fet el login.
  - ***NewUserAutenticated.java***: Classe responsable de generar un client web nou.
  - ***WebClientAuthentication.java***: Classe responsable de permetre fer inici de sessió a un client web.
  - ***InfoClientWeb.java***: Classe responsable d'aportar la informació del client web quan es clica damunt del seu nom d'usuari.
- Package "**ConnectionToJSPRooms**":
  - ***SearchRoomById.java***: Classe que mostra la informació específica d'una habitació.
  - ***SearchRooms.java***: Classe que permet la cerca de totes les habitacions.
- Package "**ConnectionToJSPTenants**":
  - ***RentingRoom.java***: Classe que permet llogar una habitació.

Les classes JSP que s'utilitzen són les següents:

- ***EntryDataForAutenticateClientWeb.jsp***: En aquest JSP hi ha la lògica que permet fer l'inici de sessió. S'utilitza AJAX per a mantenir el comportament desitjat.
- ***EntryDataForNewUser.jsp***: En aquest JSP demana la informació necessària per a registrar un nou client web. Quan l'usuari es registra, es retorna a la pàgina d'inici.
- ***back.jsp***: Aquest JSP conté un botó, que és inclòs en lespàgines JSP d'inici de sessió, registrar-se i la que mostra els detalls d'una habitació en concret. Bàsicament va cap a la pàgina anterior.
- ***error.jsp***: Pàgina JSP que estava inclosa en l'esquelet de la pràctica 2 del moodle i que és una pàgina d'error, amb la qual cosa que té accés a l'objecte implícit *exception* i és capaç de mostrar el què ha succeït, sempre que siguin errors en les peticions HTTP.
- ***header.jsp***: Aquest JSP s'inclou en altres JSP (rooms i roomsById) per tal de tenir una mateixa capçalera en la pàgina web.
- ***index.jsp***: Equival al *index.html*. En la nostra pràctica la primera pàgina que s'obre, l'URL pren la següent forma:

  localhost:8080/Pract2_SistemesOberts/welcome.do.

  Per tal d'aconseguir aquest fet, és necessari un fitxer Java que redirigeixi el Servlet principal cap aquest URL. S'aprofita aquest per a emmagatzemar informació per a la sessió activa.

- ***loginBotons.jsp***: Aquest JSP s'inclou en altres JSP (header i al index) i es mostra quan no hi ha ningú que hagi fet l'inici de sessió. Per tant, dóna l'oportunitat d'iniciar sessió i/o registrar-se.
- ***roomsById.jsp***: En aquest JSP es mostra la informació sobre una única habitació. En cas d'haver iniciat sessió amb èxit, permet llogar en cas que l'habitació no hagi estat llogada. Només es mostren els usuaris que tenen menys de 3 habitacions llogades en el dia en què es connectin.
- ***rooms.jsp***: En aquest JSP es mostra un llistat amb totes les dades de les habitacions que es troben en la base de dades.
- ***tenantRentedRoom.jsp***: Permet que quan s'ha llogat una habitació, es recarregui la pàgina i es mostrin els canvis també emmagatzemats en la base de dades.
- ***textLogin.jsp***: Aquest JSP s'inclou en altres JSP (header i index). Apareix quan s'ha fet un inici de sessió vàlid. Dóna el nom de la persona que ha iniciat sessió i mostra un botó que permet fer logout.

Cal tenir en compte que en el descriptor de desplegament (web.xml) hi ha com a fitxer de benvinguda l'*index.html*. En la nostra pràctica aquest fitxer simplement fa una redirecció cap a *welcome.do,* ja que tenim una classe Java que prepara valors per a la sessió. Llavors, realment el fitxer principal o el que representa la pàgina d'inici és el fitxer index.jsp, tal com s'ha comentat amb anterioritat.

2.1 Parts opcionals implementades

Els elements opcionals que s'han implementat han estat l'aplicació d'AJAX en l'inici de sessió, de la mateixa forma que s'aplica AJAX en el registre d'un usuari nou.

També s'ha aconseguit que quan es faci l'inici de sessió aquest retorni a la pàgina originària d'on venies. A més a més, recuperem la informació del client web que s'ha autenticat, si es pitja damunt del botó. Aquesta informació la recuperem en forma de popup.

Hem afegit una cosa que no és ni opcional que bàsicament és el fet de registrar-se. Aquesta opció genera un nou client web i quan el registre ha anat bé, retorna a la pàgina d'inici.

Finalment, s'ha inclòs un text que permet, des de qualsevol punt de la web, que s'ha fet login, mostrant qui ha iniciat sessió (també permet recuperar la informació del client web) i donant l'opció de fer logout.

3. Jocs de proves realitzats

Descripció de la prova	Resultat esperat	Resultat final
La pàgina principal és un .do.	OK.	OK.
Redirigeix al .do principal si no podem cap URL després de Pract2SistemesOberts.	OK.	OK.
Redirigeix al .do principal si accedir a la pàgina html.	OK.	OK.
Mostra un nom personalitzat de pàgina dalt de la pestanya.	OK.	OK.
En la pàgina principal permet fer login o sign in.	OK.	OK.
En el login no s'actualitza la pàgina si el login és incorrecte.	OK.	OK.
En el login podem canviar l'URL de sign in.	OK.	OK.
En el login hi ha un botó de back per anar a l'anterior pàgina.	OK.	OK.
En el login podem ocultar o mostrar la contrasenya.	OK.	OK.
En el login no ens permet fer el login fins que no s'han introduït el login i la password.	OK.	OK.
Un cop fet el login ja no ens deixa fer ni login ni sign in, ens mostra un missatge amb l'usuari que ha fet login.	OK.	OK.
En el sign in podem ocultar o mostrar la contrasenya.	OK.	OK.
En el sign in hi ha un botó de back per anar a l'anterior pàgina.	OK.	OK.
En el sign in no ens deixa fer sign in fins que no s'han omplert els camps de username, mail i password.	OK.	OK.
En el sign in es mostra un missatge d'error si el nom d'usuari ja està a la BD, sense actualitzar la pàgina.	OK.	OK.
En el sign in el correu electrònic ha de tenir un format correcte.	OK.	OK.
En el sign in la contrasenya ha de ser de mínim 8 caràcters.	OK.	OK.
Quan canviem de pàgina el login es manté.	OK.	OK.
Quan es fa un search sense especificar location surten totes les habitacions.	OK.	OK.
Podem posar la ciutat al buscador per filtrar les habitacions.	OK.	OK.
Sempre tenim un botó per tornar a la pàgina principal.	OK.	OK.
Es pot clicar damunt del nom d'usuari autenticat per veure la seva informació en forma de pop-up.	OK.	OK.
Quan es mostren les habitacions es veu una imatge amb nom, descripció, adreça, ciutat i si està llogada o no.	OK.	OK.
En cada habitació hi ha un botó per veure més detalls.	OK.	OK.

Quan es veuen els detalls es pot reservar l'habitació després d'escollir un client en cas que no ho estigui.	OK.	OK.
No es pot llogar una habitació si no estàs autenticat.	OK.	OK.
No es pot llogar una habitació si el client ja té fetes 3 reserves en el mateix dia.	OK.	OK.
Quan un usuari està autenticat es pot fer logout i et retorna a la pàgina on estaves.	OK.	OK.
Quan reservem una habitació surt un pop-up de confirmació abans de reservar-la.	OK.	OK.
Tots els estils estan en una fulla d'estils externa.	OK.	OK.
Tots els javascripts estan al head.	OK.	OK.
Es fa un ús adequat d'ids i classes pel que fa als estils.	OK.	OK.
Es permet recordar la contrasenya.	OK.	KO.
Es fa ús de cookies.	OK.	KO.
Es fa ús d'AJAX.	OK.	OK.
Totes les dades estan ben actualitzades en la BD i en la pàgina web.	OK.	OK.
No es permet inserir a la BD dues habitacions amb la mateixa adreça i el mateix nom.	OK.	OK.
La contrasenya no es pot aconseguir de cap manera en la web.	OK.	OK.
La contrasenya de la BD està xifrada.	OK.	OK.
Es fa ús d'un token per a millorar la seguretat.	OK.	OK.

4.Conclusions

Un cop hem finalitzat aquest projecte hem vist com es fa una web sencera, quines són les parts necessàries per a implementar-la des de zero. Aquestes parts són:

- Backend: Aquesta part comporta des de la creació/generació de les taules, inserció de dades, creació de l'API REST fins a la mateixa connexió amb la base de dades.
- Frontend: Aquesta part és la que es mostrarà al client final i serà la que permetrà accedir a la lògica de negoci i mostrar les dades amb un format, donat pel HTML i també amb un estil, donat pel CSS.

Tot el que s'ha tocat en aquesta pràctica creiem que ens serà important, ja que tothom vol cercar informació, comprar, vendre i fer múltiples accions a través d'Internet. Així doncs l'única manera de fer-ho és mitjançant les pàgines web.

En el nostre cas, la tecnologia que fem servir és possible que ara ja no estigui tan implantada, excepte les API REST, ja que el SOAP ha quedat obsolet, però ens mostra el funcionament de les crides HTTP i tot el que comporta el processament d'aquestes dades.

El que més ens ha agradat ha estat:

- Generar la plana web amb JSP.
- Persistència de les dades a la base de dades mitjançant l'API REST.
- Tecnologies JAXB, JAX-RS, Java Persistence.

El que menys ens ha agradat ha estat:

- Errors difícils d'entendre quan comences a implementar el backend (API REST).
- Entendre el funcionament del POST i PUT a l'hora de la implementació.
- Complexitat.

5. Manual d'instal·lació

Per tal d'executar aquesta pràctica és necessari descarregar dos projectes:

Nom projecte	URL Github
RESTServiceForPract2SistemesOberts	https://github.com/CescFT/RESTServiceForPract2SistemesOberts/
Pract2_SistemesOberts	https://github.com/CescFT/Pract2_SistemesOberts

També és necessari el següent programari:

- Netbeans 8.2 versió EE o superior.
- Glassfish Server 4.1.1.
- JDK 8 o superior.
- Java DB i crear una base de dades amb nom homework1 i contrasenya ROOT.

Un cop ens assegurem que tenim tot el que es necessita, cal seguir els següents passos:

1. Ens situem a damunt del projecte ***RESTServiceForPract2SistemesOberts*** i fent clic dret, seleccionem l'opció Deploy o Run.
2. Obrim el navegador que s'utilitzi i ens dirigim al següent URL: localhost:8080/RESTServiceForPract2SistemesOberts
3. Premem el botó "***insert***" per tal d'afegir a la base de dades creada les columnes preparades.
4. Obrim una pestanya nova.
5. En la nova pestanya anem al següent URL: localhost:8080/ Pract2_SistemesOberts
6. Es redirigirà a l'URL: localhost:8080/ Pract2_SistemesOberts/welcome.do

D'acord amb la lògica de negoci i les parts opcionals descrites en l'enunciat, es poden veure implementades i operatives.