PRÀCTICA 1

API REST PER A UNA PÀGINA WEB DE LLOGAR HABITACIONS

SISTEMES OBERTS

FRANCESC FERRÉ TARRÉS

ALEIX SANCHO PUJALS

1/12/2019

Índex

1. Introducció

Aquesta primera part de la pràctica tracta de construir una API Web per tal d'implementar totes les funcionalitats necessàries per a una aplicació de lloguer d'habitacions. En aquesta aplicació, que ha de ser exclusivament web, ens trobarem dues bases de dades principals, en una d'elles s'hi guardaran els clients de l'aplicació i per tant les persones que voldran llogar les habitacions, i en l'altre es guardaran totes les habitacions amb tota la informació necessària de cadascuna.

Per a implementar aquesta API Web caldrà fer ús d'un servidor REST que retorni JSON. S'hauran d'implementar tots els serveis web per tal de poder realitzar totes les operacions necessàries entre els llogaters i les habitacions.

Finalment caldrà implementar un sistema d'autenticació que en el nostre cas estarà en els usuaris que vulguin accedir al sistema web. És a dir, per a realitzar qualsevol consulta o acció amb els llogaters prèviament t'hauràs d'haver autenticat.

2. Estructura de la pràctica

El projecte està disponible a Github fent clic aquí. Es distribueix en tres packages diferents:

- Package "autenticació": Conté dos fitxers,
  - ***credentialsClient***: Permetrà emmagatzemar les dades del client web que s'autenticarà.
  - ***token***: Emmagatzema el token per a un client web específic.
- Package "model/entities": Hi ha totes les classes d'entitat de la nostra aplicació web:
  - ***Habitació***: Entitat per a guardar les dades d'una habitació.
  - ***Requeriment***: Classe embedded que és un atribut a l'habitació i que conté el requeriment.
  - ***TipusHabitació***: Tipus enumerat que conté tots els tipus possibles d'habitacions.
  - ***Llogater***: Entitat per a guardar les dades del llogater.
  - ***informacioLlogater***: Classe embedded que és un atribut a llogater i que conté la informació bàsica d'un llogater.
  - ***SexeLlogater***: Tipus enumerat que conté els tipus de sexe que una habitació pot tenir (Home, Dona, Unisex) i també el llogater (Home, Dona).
- Package "services": Conté tots els fitxers necessaris per a fer anar la nostra API REST:
  - ***AbstractFacade***: Classe pare que hi ha els mètodes bàsics HTTP.
  - ***HabitacioFacadeREST***: Classe que hereta de AbstractFacade i que implementa la API REST per a les habitacions.
  - ***LlogaterFacadeREST***: Classe que hereta de AbstractFacade i que implementa la API REST per al llogater.
  - ***autenticacioClientWeb***: Classe que permet l'autenticació dels clients web.
  - ***RESTapp***: Facilita el llançament de l'aplicació web.

El sistema d'autenticació generat està basat en la idea real de com funcionen les pàgines web. En primer lloc, l'usuari s'ha de connectar a:

http://localhost:8080/Pract1_SistemesOberts/webresources/autenticacio

En aquest URL ha d'afegir dos "form params", un que serà el username i l'altre el password. Aquesta informació te la retorna el install.jsp.

Un cop fet això, et retorna un token. Aquest s'ha de validar en contra de la API REST dels llogaters; per a fer-ho el client s'ha de connectar a:

http://localhost:8080/Pract1_SistemesOberts/webresources/tenant/processarToken

Llavors, si el token és correcte, ja hi ha accés a totes les crides REST que hi ha, en canvi, les habitacions es pot accedir sense autenticació.

Hi ha 3 entitats JPA:

- Habitació
- Client
- Llogater

Pel que fa a l'habitació, s'ha emmagatzemat:

- Identificador de l'habitació: Long
- Descripció de l'habitació: String
- Adreça on està ubicada l'habitació: String
- Ciutat on està ubicada l'habitació: String
- Tipus d'habitació: TipusHabitació (SIMPLE|DOBLE|EXTERIOR|INTERIOR|MOBLADA)
- Preu per mes de l'habitació: Float
- Requeriment : Embedded de Requeriment
- Llogater: Relació OneToOne cap al llogater

En el requeriment s'emmagatzema:

- Sexe del Llogater: SexeLlogater (HOME|DONA|UNISEX)
- Mínim edat: Enter
- Màxim edat: Enter
- És fumador: Booleà
- Té mascotes: Booleà

Pel que fa al client, s'ha emmagatzemat:

- Nom d'usuari: String
- Contrassenya: String
- Email: String
- Token d'autorització: token

Pel que fa al llogater, s'ha emmagatzemat:

- Identificador del llogater: Long
- Informació del llogater: Embedded de informacioLlogater

En la informació del llogater s'emmagatzema:

- Nom : String
- Cognom : String
- DNI: String
- Edat : Enter
- És fumador: Booleà
- Té mascotes: Booleà
- Sexe del llogater: SexeLlogater(HOME|DONA|UNISEX)

En cadascun hi ha els seus getters, setters i tostrings.

Els serveis REST que s'han implementat són els que es demanen a l'enunciat de la pràctica, pels llogaters:

Mètode HTTP	**Crida**	**Descripció**
POST	/webresources/tenant/processarToken	Rep un JSON amb el token i se'l guarda per a poder fer totes les peticions.
GET	/webresources/tenant/{id}	Retorna un JSON amb la informació del llogater amb el id passat.
PUT	/webresources/tenant/{id}	Rep un JSON amb les dades del llogater amb el id passat i actualitza les dades.
POST	/webresources/tenant	Rep un JSON amb les dades d'un nou llogater i el registra al sistema.
GET	/webresources/tenant	Retorna un llistat de tots els tenants en format JSON.
POST	/webresources/tenant/{id}/rent	Rep un JSON amb l'habitació i li assigna el llogater amb el id passat per paràmetre
DELETE	/webresources/tenant/alliberarHabitacio/{id}	Allibera l'habitació del llogater amb el id passat i elimina el llogater del sistema.
DELETE	/webresources/tenant/{id}	Elimina el llogater amb l'identificador passat per l'URL.

Els serveis REST que s'han implementat per a les habitacions són els següents:

Mètode HTTP	Crida	Descripció
POST	/webresources/room	Rep un JSON amb la informació de la nova habitació i el guarda a la base de dades
PUT	/webresources/room/{id}	Rep un JSON amb les dades modificades per a l'habitació amb l'identificador passat per l'URL i l'actualitza.
DELETE	/webresources/room/{id}	Elimina l'habitació amb l'identificador passat per l'URL
GET	/webresources/room/{id}	Retorna un JSON amb les dades de l'habitació amb l'identificador passat per l'URL
GET	/webresources/room?location=a&sort=(asc|desc)	Retorna les habitacions que estan en la ciutat "a" ordenades segons el preu (ascendent o descendent). No pot faltar el paràmetre sort

Els serveis REST per a la autenticació són els següents:

Mètode HTTP	Crida	Descripció
POST	/webresources/autenticacio	Donats el form param "username" i "password" et genera el token d'autenticació
GET	/webresources/autenticació/{username}	Retorna les dades per a un client web en format JSON.
GET	/webresources/autenticació/{username}/{token}	Retorna la contrasenya de l'usuari descodificada.
GET	/webresources/autenticació?username=u	Retorna la informació del client autoritzat "u" en JSON.
GET	/webresources/autenticació/all	Retorna un JSON amb tots els clients autoritzats.

3. Decisions de disseny

Per tal de realitzar la pràctica hem utilitzat el projecte base del moodle i hem anat generant el nostre codi. En primer lloc, vam fer l'estructura de la base de dades en JPA. Les entitats que tenim són les que s'han comentat amb anterioritat.

La relació entre habitació i llogater és una OneToOne unidireccional on qui és propietari de la relació és l'habitació. Per tant, qui té la "*join column*" és l'habitació.

Altres decisions de disseny que s'han pres amb relació a la creació de la base de dades són:

- En el install.jsp s'ha afegit codi per tal d'evitar insercions duplicades a la base de dades.
- Per tal de mantenir certa abstracció de dades, en les habitacions no tenim els atributs que formen el requeriment directament, sinó que ens hem generat una classe embedded que conté totes les condicions que generen un requeriment(sexe, edat mínima, edat màxima, fumador/a i mascotes). De la mateixa forma succeeix en els llogaters; la informació més personal del llogater s'ha emmagatzemat en un fitxer a part embedded que conté el cognom, el nom, el dni, l'edat, el sexe, si és fumador/a i si té mascotes.
- S'han generat dos enumerats, un que representa els diferents tipus d'habitacions (simple, doble, moblada, interior i exterior) i l'altre que conté el sexe (home, dona i unisex).

En canvi, per a la API REST, hem pres moltes més decisions. En primer lloc hem fet totes les crides REST que hi ha a l'enunciat de la pràctica i alguna que hem generat nosaltres.

Com s'ha comentat amb anterioritat perquè es pugui accedir a la API REST dels llogaters, cal que primer el client web s'autentiqui. Per tal de fer-ho hem generat una nova API REST que facilita aquesta acció: /webresources/autenticacio.

Els mètodes HTTP estan comentats en les taules anteriors però la idea és que el client web que es vulgui autenticar rebi un token únic per a ell i que aquest sigui el que li permetrà entrar en els mètodes HTTP per als llogaters (les habitacions no requereixen estar autenticat, sempre les podem consultar).

En el cas de la API REST per als llogaters, s'ha generat dos atributs que permeten emmagatzemar el token i el client web autenticat que està fent les peticions. Això és gràcies al mètode HTTP POST que hi ha a /webresources/tenant/processarToken.

En aquest mètode POST, tal com està explicat en les taules anteriors, rep un JSON amb el token i en cas que sigui vàlid, el guarda, en altre cas no i et retorna que no és possible.

També hem afegit un mètode HTTP DELETE que es troba a

/webresources/tenant/alliberarHabitacio/{id} que permet alliberar l'habitació ocupada per un llogater i a més elimina aquest de la base de dades.

4. Jocs de proves realitzats

Per tal de veure el joc de proves que hem fet, tant els casos positius, com els negatius, hem creat aquest collections de Postman i aquí hi ha totes les crides.

https://www.getpostman.com/collections/9513faf733d5abbf68bf

Per tal de descriure una mica el que hi ha, els casos que s'han provat:

HABITACIÓ		
Cas	**Sortida esperada**	**Sortida real**
Generació d'una habitació a partir d'un JSON **POST /webresources/room**	OK	OK
No passar-li un JSON al mètode POST de crear una nova habitació **POST /webresources/room**	KO	KO
Modificació d'una habitació donat un ID vàlid i el JSON amb les dades modificades **PUT /webresources/room/id**	OK	OK
No passar-li el ID de una habitació a l'hora de modificar una habitació però sí un JSON vàlid **PUT /webresources/room/id**	KO	KO
No passar-li un ID informat però sí un JSON correcte **PUT /webresources/room/id**	KO	KO
Passar-li un ID vàlid però no un JSON correcte a l'hora de actualitzar les dades d'una habitació **PUT /webresources/room/id**	KO	KO
Eliminar una habitació donat un identificador vàlid **DELETE /webresources/room/id**	OK	OK
Passar un ID no informat a l'hora d'eliminar una habitació **DELETE /webresources/room/id**	KO	KO
Passar un ID no vàlid a l'hora d'eliminar una habitació **DELETE /webresources/room/id**	KO	KO
Obtindre la informació d'una habitació donat un id vàlid **GET /webresources/room/id**	OK	OK
Donar un ID no informat a l'hora d'obtenir les dades d'una habitació **GET /webresources/room/id**	KO	KO
ID no correcte a l'hora d'obtenir les dades d'una habitació **GET /webresources/room/id**	KO	KO
Obtenir les habitacions d'una localitat ordenades segons el paràmetre sort **GET /webresources/room?location=a&sort=(asc|desc)**	OK	OK

No enviar criteri a l'hora d'obtenir les habitacions d'una localitat. **GET /webresources/room?location=a&sort=(asc|desc)**	KO	KO
No passar-li la localització i obtenir totes les habitacions ordenades segons el paràmetre sort **GET /webresources/room?sort=(asc|desc)**	OK	OK
Introduir un valor de criteri que no sigui ni "asc" ni "desc". **GET /webresources/room?location=a&sort=(prova)**	KO	KO
Introduir un valor de criteri que no sigui ni "asc" ni "desc". **GET /webresources/room?sort=(prova)**	KO	KO
Introduir un valor de location que no existeixi en la base de dades. **GET /webresources/room?location=prova&sort=(asc|desc)**	KO	KO

LLOGATER		
Passant-li el JSON amb el token correcte, el processa i l'emmagatzema **POST /webresources/tenant/processarToken**	OK	OK
Passant-li un JSON mal format no processa res. **POST /webresources/tenant/processarToken**	KO	KO
Passar un JSON vàlid però no és un token verificat **POST /webresources/tenant/processarToken**	KO	KO
Es lloga l'habitació passada per un JSON ben format, l'identificador del llogater està informat i existeix i el client està autenticat **POST /webresources/tenant/id/rent**	OK	OK
El client web no està autenticat i intenta llogar una habitació passant-li un JSON ben format i un identificador vàlid **POST /webresources/tenant/id/rent**	KO	KO
El client web no està autenticat i intenta llogar una habitació passant-li un JSON ben format i un identificador vàlid però aquest llogater no compleix els requisits **POST /webresources/tenant/id/rent**	KO	KO
El client web està autenticat, es passa un JSON malformat o incorrecte i el id del llogater és correcte **POST /webresources/tenant/id/rent**	KO	KO

El client web està autenticat, es passa un JSON ben format i el identificador del client és incorrecte o no informat **POST /webresources/tenant/id/rent**	KO	KO
Es crea el llogater nou passant-li un JSON ben format i el client web està autenticat **POST /webresources/tenant**	OK	OK
El client web no està autenticat i intenta generar un nou llogater passant-li un JSON ben format **POST /webresources/tenant**	KO	KO
El client web està autenticat però intenta passar un JSON mal format **POST /webresources/tenant**	KO	KO
El client web està autenticat, es passa un identificador de llogater vàlid i informat i es passa un JSON correcte **PUT /webresources/tenant/id**	OK	OK
El client web no està autenticat, es passa un identificador correcte i informat i un JSON vàlid **PUT /webresources/tenant/id**	KO	KO
El client web està autenticat i el identificador és correcte però es passa un JSON invàlid o malformat **PUT /webresources/tenant/id**	KO	KO
El client està autenticat, el JSON és vàlid però el identificador és no informat o no es troba un llogater amb l'identificador donat **PUT /webresources/tenant/id**	KO	KO
El client està autenticat, el identificador és correcte i ve informat **DELETE /webresources/tenant/alliberarHabitacio/id**	OK	OK
El client no està autenticat però el identificador és correcte **DELETE /webresources/tenant/alliberarHabitacio/id**	KO	KO
Client web autenticat però el id és incorrecte o no ve informat **DELETE /webresources/tenant/alliberarHabitacio/id**	KO	KO
El client està autenticat i l'identificador és correcte **DELETE /webresources/tenant/id**	OK	OK
El client no està autenticat però l'identificador és correcte **DELETE /webresources/tenant/id**	KO	KO
El client està autenticat però l'identificador és incorrecte	KO	KO

DELETE /webresources/tenant/id		
El client està autenticat, l'identificador és correcte i ve informat **GET /webresources/tenant/id**	OK	OK
El client no està autenticat però l'indentificador és correcte i ve informat **GET /webresources/tenant/id**	KO	KO
El client està autenticat però l'identificador no és correcte o no ve informat **GET/webresources/tenant/id**	KO	KO
El client està autenticat **GET /webresources/tenant**	OK	OK
El client no està autenticat **GET /webresources/tenant**	KO	KO
El client està autenticat però no hi ha cap llogater **GET /webresources/tenant**	KO	KO

AUTENTICACIÓ DEL CLIENT WEB		
Apareixen els dos form params informats de forma correcte **POST /webresources/autenticacio**	OK	OK
El form param "username" ve informat i és correcte però la password no **POST /webresources/autenticacio**	KO	KO
El form param "password" ve informat i és correcte però el form param "username" no **POST /webresources/autenticacio**	KO	KO
El username és correcte i ve informat **GET /webresources/autenticacio/username**	OK	OK
El username no és correcte perquè no ve informat o és invàlid **GET /webresources/autenticacio/username**	KO	KO
El username és correcte i el token també, els dos venen informats i són vàlids **GET /webresources/autenticacio/username/password**	OK	OK
El username no ve informat o és invàlid però el token és correcte i ve informat **GET /webresources/autenticacio/username/password**	KO	KO
El token és correcte i ve informat però el username no és vàlid o és incorrecte **GET /webresources/autenticacio/username/password**	KO	KO
El username és correcte i ve informat	OK	OK

GET /webresources/autenticacio?username=a		
El username no ve informat **GET /webresources/autenticacio?username=**	KO	KO
El username és incorrecte **GET /webresources/autenticacio?username=v**	KO	KO
Retorna tots els clients autoritzats de forma correcte **GET /webresources/autenticacio/all**	OK	OK

5. Conclusions

Fent aquesta pràctica ens hem adonat de la senzillesa que té la generació d'una API REST, i tot gràcies a JAX-RS. De la mateixa forma, la creació de la base de dades que hi ha darrere es realitza de manera tan senzilla com la generació de classes d'entitat anotades amb @Entity gràcies a JPA.

El que sí que hem trobat complicat i que per sort ja estava generat en l'esquelet de la pràctica 1 disponible al moodle és el fitxer *persistence.xml*, no el concepte sinó generar una bona unitat de persistència i que funcioni correctament, que tingui el comportament que esperem.

També ens hem adonat que el fet que sigui REST i que tot es faci a través dels mètodes del protocol HTTP fa que sigui molt més ràpid que si ho haguéssim fet amb SOAP.

La raó és evident, el SOAP és molt *verbose*, o sigui, per cada missatge HTTP hi va un XML representant el missatge SOAP, que conté el sobre, el cos, ... i a més a més cadascuna de les crides ha de ser validada pel WSDL (Web Service Description Language) que és el contracte que hi ha entre el client i el servidor web i és qui conté tots els mètodes amb els seus paràmetres i en cas que es cridi de forma errònia, ja sigui perquè es crida un mètode que no es contempla o no es crida amb els paràmetres correctes, ja deix de funcionar, en canvi en REST el mateix mètode HTTP et retorna un codi d'error i la seva descripció i es pot entendre més o menys que ha fallat segons el codi.

- 1XX: Resposta informativa
- 2XX: Petició processada
- 3XX: Redirecció
- 4XX: Error en la crida del client
- 5XX: Error al servidor

També una altra cosa que fa que la rapidesa de REST sigui molt més superior a SOAP és que aquest XML s'ha de processar i validar contra un DTD o bé un XML SCHEMA (més comú aquest perquè és molt més restrictiu, té moltes més opcions i cobreix molts més casos).

6. Com executar la pràctica

En primer lloc cal connectar la base de dades, després fer el deploy i després entrar al navegador web i posar com a URL: http://localhost:8080/Pract1_SistemesOberts/

Després cal donar-li al botó insert i s'inseriran totes files a les taules corresponents de la base de dades.

Un cop fet això obrim el Postman i per tal de ja estar autenticats farem els passos descrits en l'apartat 2.

I a partir d'aquí doncs ja podem fer totes les crides REST que estan contemplades amb anterioritat i que es troben guardades en una collection del Postman.

https://www.getpostman.com/collections/9513faf733d5abbf68bf