Pràctica 1: **Java Servlets**

PTI - G12

Josep Martínez

Adrià Martínez

Índex

1. Entorn

L'entorn de treball per aquesta pràctica és bastant senzill. S'han seguit les indicacions plantejades per la guia de la pràctica del repositori de l'assignatura. En alguns casos s'ha fet alguna modificació o ampliació dels passos a seguir pel correcte funcionament de la pràctica.

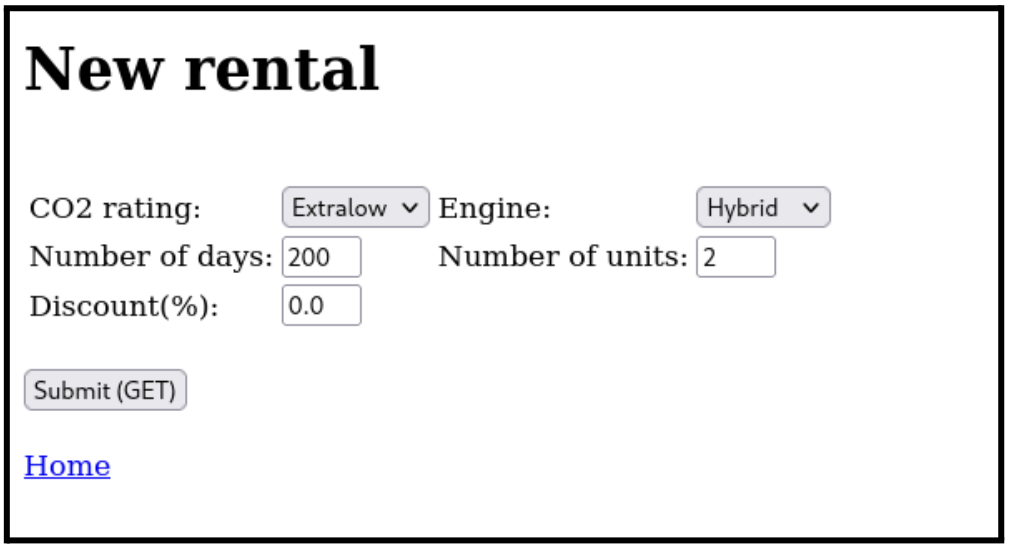
Per picar el codi s'ha fet servir ***Visual Studio Code*** en una distribució **Ubuntu** (Linux).

2. Servlets

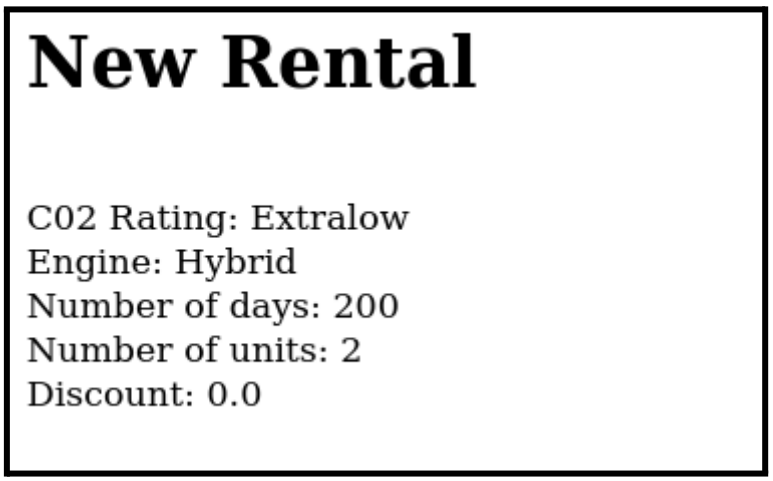
Aquí es descriu la implementació i breu explicació del codi dels servlets.

2.1 Nou lloguer

La funció del servlet CarRentalNew és la d'oferir el servei per poder oferir un formulari de reserva de lloguer en el qual, després d'emplenar els camps que es poden veure en la següent imatge.



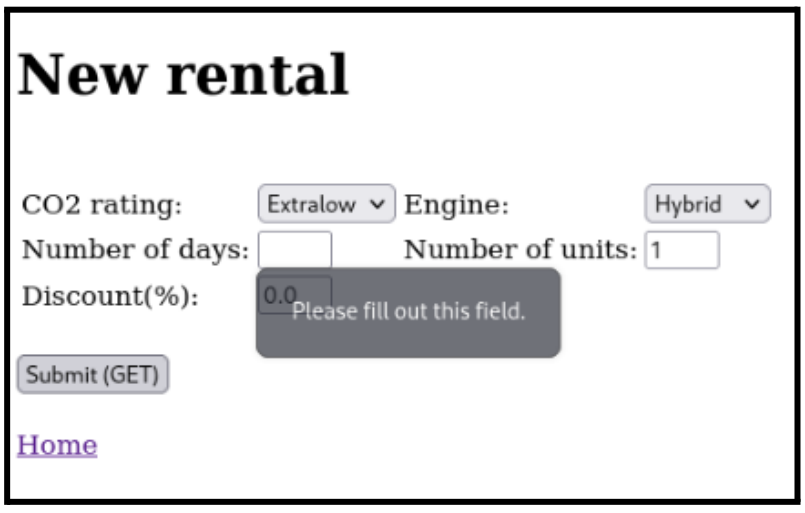
El nostre codi rebrà la petició i mitjançant els atributs del get els mostrarà per pantalla a l'usuari mentre els guarda en un fitxer "rentals.json" en el que guardarem les diferents peticions en format json.

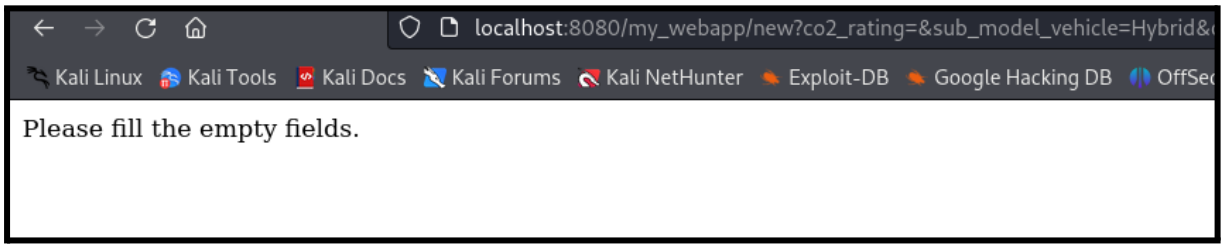


Per poder mostrar el C02 Rating com un String hem modificat el fitxer carrental_form_new.html perquè a l'hora d'enviar la request ens enviï els valors com a text i no com a números, com s'ensenya a continuació.

```JavaScript
...
<td>CO2 rating:</td>
<td><select name=co2_rating>
<option selected VALUE=Extralow>Extralow
<option VALUE=Low>Low
<option VALUE=Medium>Medium
<option VALUE=High>High
...
```

La implementació del servlet està feta de la següent manera. Un cop rebem la petició es fa el get dels camps pertinents, els quals són comprovats perquè no puguin ser buits o nulls. Aquesta comprovació la fem tant client-site com server-site, per evitar tota classe de maneres d'introduir informació incorrecta.





Al fitxer carrental_form_new.html fem els següents canvis:

```
JavaScript
...
<tr>
<td>Number of days:</td>
<td><input name=dies_lloguer size=3 maxlength=3 value=1 required></td>
<td>Number of units:</td>
<td><input name=num_vehicles size=3 maxlength=3 value=1 required></td>
</tr>
<tr>
<td>Discount(%):</td>
<td><input name=descompte size=3 maxlength=3 value=0.0 required></td>
</tr>
...
```

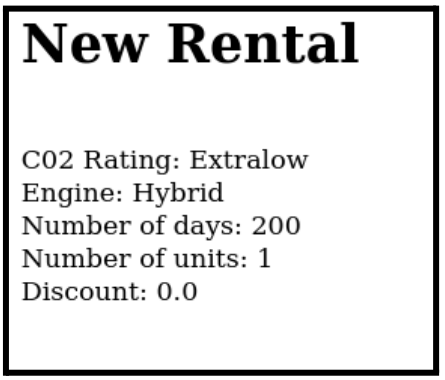
I al fitxer del servlet (CarRentalNew.java) afegim aquestes comprovacions al agafar els diferents camps.

```
Java
...
if ( carType == null || carType.isEmpty() ||
        co2 == null || co2.isEmpty() || days == null || days.isEmpty() ||
        vehicles == null || vehicles.isEmpty() || discount == null || discount.isEmpty())
{
        out.println("Please fill the empty fields.");
        return;
        }
...
```

Un cop llegits tots els camps de la request es procedeix a introduir-los a un JSON object.

Per realitzar això creem amb la primera inicialització un JSONObject que serà sobre el que iterar per poder mantenir el format de JSON. Per cada request es crearà un JSON Object el qual serà introduit sobre el JSONArray principal del nostre *mainJsonObject*.

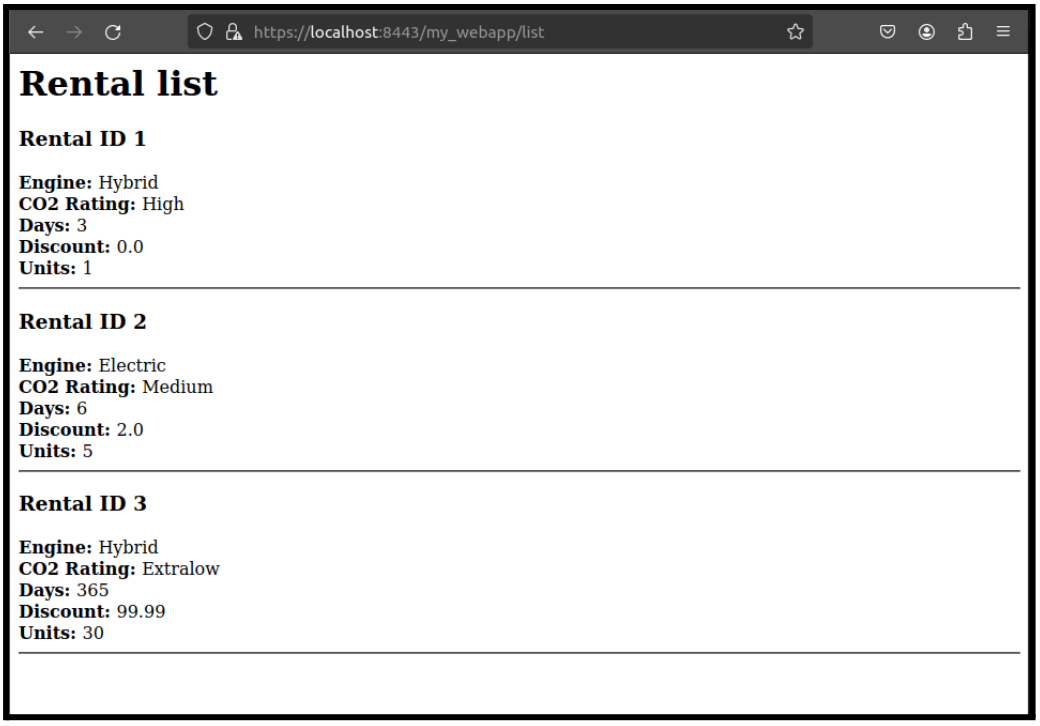
Un cop acabada la request s'escriu el JSON sobre un fitxer per a poder ser processat més endavant pel post i es mostra per pantalla els camps que s'han introduït al fixter.



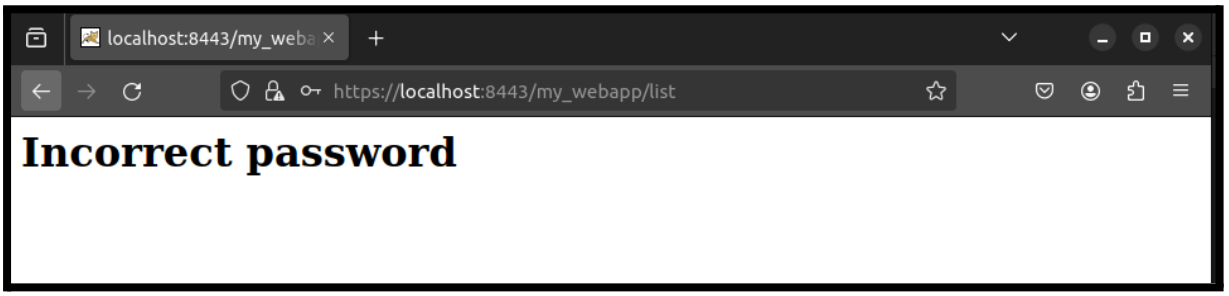
2.2 Llistar lloguers

La funció del servlet *CarRentalList.java* és obtenir tots els lloguers de cotxes del fitxer *rentals.json* que, com bé indica l'extensió de l'arxiu, estan emmagatzemats en format JSON.

El codi comprova si les credencials proporcionades són: admin/admin. En cas afirmatiu, la pàgina mostra una llista amb tota la informació.



En cas contrari, la pàgina notifica a l'usuari que les credencials són incorrectes.



El codi funciona de la següent manera. Mitjançant un bloc if/else, comprovem que l'usuari i la contrasenya són admin/admin. En cas afirmatiu, intentem parsejar l'arxiu rentals.json a un array.

```java
JSONParser parser = new JSONParser();
try (Reader reader = new FileReader("rentals.json")) {
      JSONObject jsonObject = (JSONObject) parser.parse(reader);
      JSONArray rentals = (JSONArray) jsonObject.get("rentals");
   ...
   ...
}
```

Seguidament, iterem sobre cada element de l'array i printem per pantalla la informació de cada rental. Si no hi ha cap lloguer guardat, ho notifiquem.

```java
int nRentals = 1;
for (Object obj : rentals) {
 JSONObject rental = (JSONObject) obj;
 out.println("<h3>Rental ID " + nRentals + "</h3>");
 out.println("<b>Engine: </b>" + rental.get("engine") + "<br>");
 out.println("<b>CO2 Rating: </b>" + rental.get("rating") + "<br>");
 out.println("<b>Days: </b>" + rental.get("days") + "<br>");
 out.println("<b>Discount: </b>" + rental.get("disc") + "%<br>");
 out.println("<b>Units: </b>" + rental.get("units") + "<br>");
 out.println("<hr>");
 ++nRentals;
}
if(nRentals == 1) out.println("<h2>No rentals registered.</h2>");
```

Per últim, s'han de tractar les possibles excepcions ocasionades per la lectura del fitxer (IOException) o algun error en el parseig (ParseException).

```java
catch (IOException | ParseException e) {
      e.printStackTrace();
      out.println("<h2>No rentals registered.</h2>");}
```

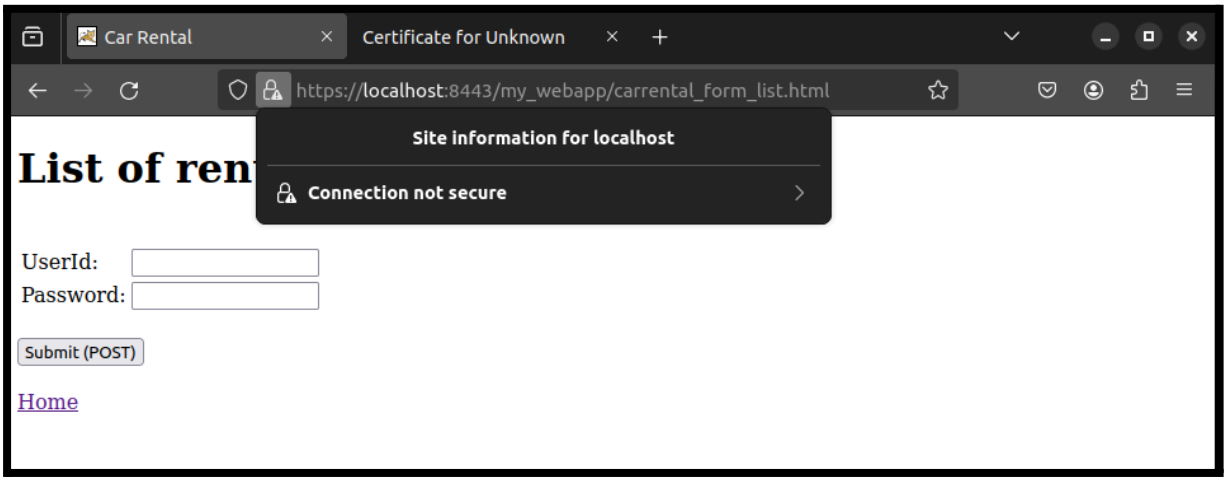
2.3 Configuració SSL/TLS

Per la configuració de SSL/TLS a la nostre web-app, es precisa d'un certificat. Amb la següent comanda, generem la clau privada del servidor amb l'algorisme RSA i un certificat autosignat que s'emmagatzema al nostre **$HOME**.

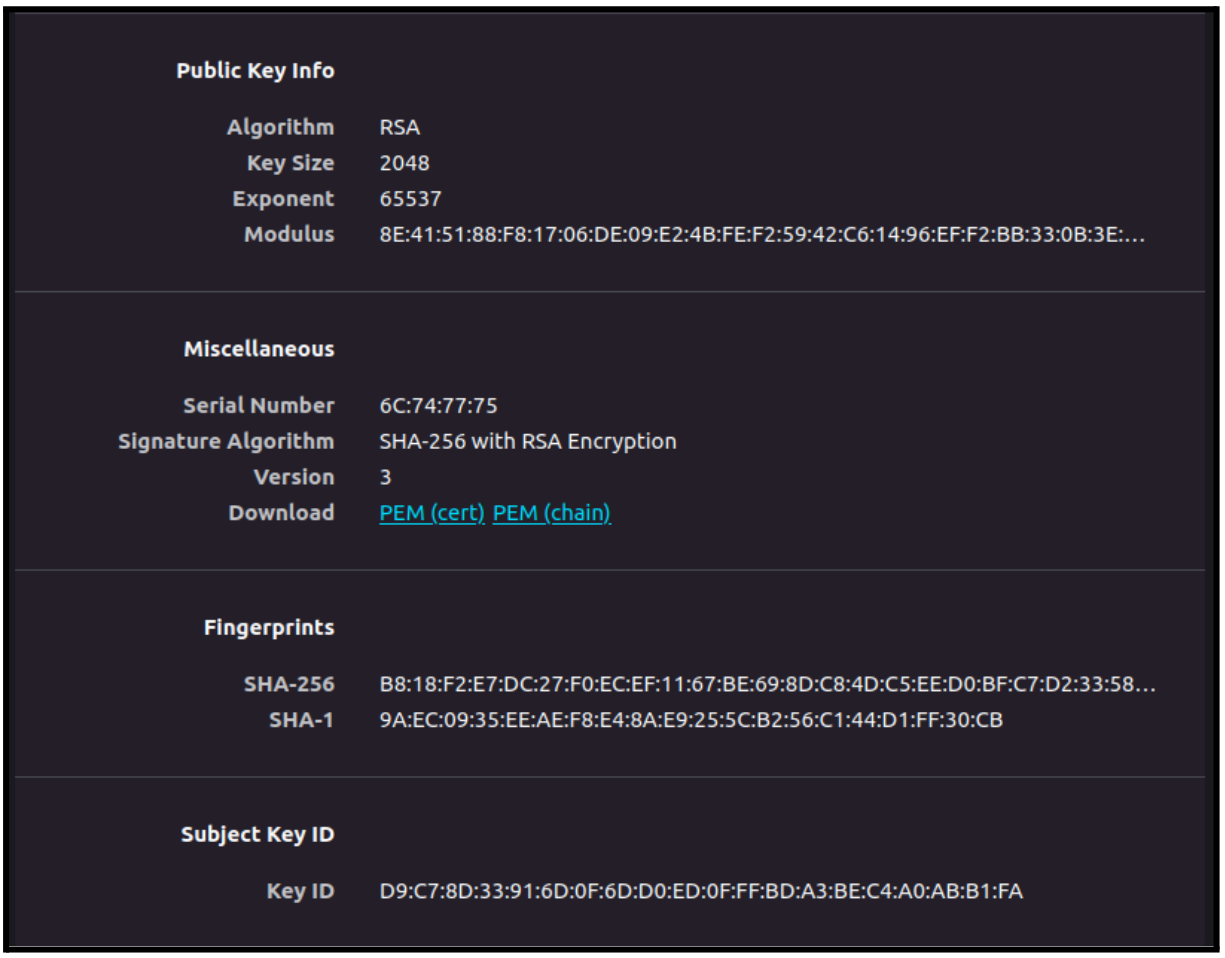
```
Unset
keytool -genkey -alias tomcat -keyalg RSA
```

Per últim, afegim un nou connector especificant el port 8443 on li diem on es troba la clau del servidor. Tot això al fitxer ***server.xml***, tal i com proposa la guía. El darrer pas és reiniciar el servidor.

Com veiem a la següent il·lustració, el navegador ens adverteix que la connexió pot no ser segura donat que el certificat és autosignat i no pertany a cap llista de CAs de confiança.



Com a curiositat, podem visualitzar el nostre certificat. El qual compta amb la majoria de camps *unknown* donat que no els hem omplert alhora de generar-lo. Aquí podem veure una part del certificat i informació de la clau pública.



3. Dockerització

La tasca de dockeritzar l'aplicació ha sigut prou senzilla. El primer pas ha sigut crear un *Dockerfile* com el de la guia:

```
Unset
FROM tomcat:10
COPY webapps/my_webapp /my_webapp
WORKDIR /
RUN cp -r my_webapp /usr/local/tomcat/webapps
```

Amb aquest fitxer el que fem es crear una imatge *tomcat* versió 10, el qual tindrà la nostra web a l'arrel. Posteriorment establim l'arrel com a directori origen de les comandes i copiem la *webapp* on el servei de tomcat emmagatzema les *webapps* (/usr/local/tomcat/).

Seguidament construïm la imatge amb la comanda:

```
Unset
sudo docker build -f Dockerfile -t carrental .
```

I per últim iniciem el contenidor en segon pla amb la redirecció de ports:

```
Unset
sudo docker run --name carrental -d -p 8080:8080 -p 8443:8443 carrental
```

4. Extres

4.1 Java Servlets vs Node.js: Anàlisi Comparativa

Java Servlets i Node.js són dues tecnologies potents utilitzades per construir aplicacions web, però canvien significativament en les seves arquitectures, rendiment i experiència del desenvolupador.

4.1.1 Java Servlets

Avantatges

1. Ecosistema Estable: Java ha existit durant molt temps, resultant en un ecosistema estable amb llibreries i frameworks extensos.

2. Suport a Nivell Empresarial: Els Servlets són adequats per a aplicacions empresarials a gran escala. Moltes empreses confien en Java per a sistemes crítics, i els servlets s'integren fàcilment amb altres tecnologies integrades amb Java.

3. Seguretat de Threat: Les capacitats de multithreading de Java asseguren la seva seguretat, fent-ho adequat per a aplicacions concurrents i escalables.

4. Enfocament a la programació orientada a objectes: Java és un llenguatge orientat a objectes, perfecte pel codi modular i reutilitzable, el que pot portar a aplicacions més fàcils de mantenir.

Inconvenients

1. Temps d'Inici Lent: Les aplicacions Java tradicionals, incloent-hi servlets, poden tenir temps d'inici més lents en comparació amb frameworks lleugers en altres llenguatges.

4.1.2 Node.js

Avantatges

1. E/S Asíncrona: Node.js és conegut per la seva arquitectura no bloquejadora i orientada a esdeveniments, permetent una alta concurrència i un millor maneig de les sol·licituds simultànies.

2. Temps d'Inici Ràpid: Les aplicacions Node.js solen tenir temps d'inici més ràpids, el que les fa adequades per a projectes amb actualitzacions i desplegaments freqüents.

3. Acceptació de JavaScript: Amb Node.js, els desenvolupadors poden utilitzar JavaScript tant per server com client site, simplificant el procés de desenvolupament i reduint el canvi de context.

4. Comunitat Activa: Node.js té una comunitat àmplia i activa, resultat en una gran quantitat de mòduls i paquets disponibles a través de npm, facilitant el desenvolupament ràpid.

Inconvenients

1.Callback Hell: La naturalesa asíncrona de Node.js pot portar a callbacks anidats, creant estructures de codi complexes i difícils de mantenir, conegudes com "Callback Hell".

2. No És Ideal per a Tasques Intensives de CPU: Node.js no és la millor opció per a tasques intensives de CPU degut a la seva naturalesa basada en un únic thread, la qual cosa pot provocar un rendiment disminuït per a operacions amb molts càlculs.

3. No adecuada per a Aplicacions Nivell Empresarial: Tot i que Node.js és escalable, pot no ser la primera opció per a aplicacions empresarials extremadament grans i complexes, on Java ha estat tradicionalment dominant.

4.1.3 Conclusions

En conclusió, l'elecció entre Java Servlets i Node.js depèn dels requisits específics del projecte. Java Servlets és una elecció robusta per a aplicacions empresarials, mentre que Node.js destaca en escenaris que demanden alta concurrència, aplicacions en temps real i un enfocament de desenvolupament més lleuger.

5. Conclusions

Les conclusions que traiem d'aquesta pràctica són les següents. En primer lloc, entenem que Tomcat, Glassfish i altres servidors similars són bones opcions enfocades a la docència o serveis de petita escala. En segon lloc, les aplicacions webs que funcionen amb servlets donen molt de joc donat que estan programades majoritàriament en Java, un llenguatge molt complet que es complementa molt bé amb moltes propostes de backend.

D'altra banda, també destacar que hem portat a terme les mínimes comprovacions de seguretat bàsiques, com són la revisió de camps nuls, tant a client-side com server-side. Però insuficients davant d'un cas real. Entenem, però que l'objectiu de la pràctica no és sinó un altre que el de tenir un primer contacte amb aplicacions web client-servidor i no el de preparar l'aplicació per ser exposada a Internet, ni cap cas real.

En darrer lloc, l'ús de Docker avui dia és essencial donat que les tecnologies amb contenidors són tendència i un recurs que cada cop s'empra amb més freqüència pel desplegament de serveis en qualsevol infraestructura. Per tant, ens ha agradat molt dedicar-li una petita part de la pràctica.