



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



Informe PTI Lab 1: Tomcat

Arnau Garcia Gonzalez

PTI Quadrimestre tardor Curs 2025-26

Índex

1. Introducció a Tomcat i als servlets.....	3
2. Instal·lació i configuració de les eines.....	4
3. Objectius principals de la pràctica i implementació.....	6
3.1 Implementació CarRentalNew.java.....	6
3.2 Implementació CarRentalList.java.....	9
3.3 Configuració SSL/TLS.....	11
3.4 Configuració Docker.....	12
4. Extensions de la pràctica.....	14
4.1 Configuració Docker Compose.....	14
4.2 Configuració chatbot.....	14
4.2.1 Mesura de la duració.....	15
4.2.2 Qualitat de la resposta.....	18
4.2.3 Canviar la representació del dataset.....	18
4.2.4 RAG vs non-RAG.....	20
4.2.5 Test autoproposat:.....	22

1. Introducció a Tomcat i als servlets

Apache Tomcat és un servidor web de codi obert desenvolupat sota el projecte Apache. Aquest entorn permet executar aplicacions basades en diverses especificacions de Java EE, com ara els Java Servlets. La seva arquitectura es divideix en tres parts principals: Catalina (contenidor de servlets), Coyote (connector per al protocol HTTP) i Jasper (motor encarregat de processar pàgines JSP).

Un servlet és un component Java que amplia les capacitats d'un servidor. Tot i que poden gestionar peticions de diferents tipus, el seu ús més habitual és dins de servidors web per crear aplicacions dinàmiques, sent l'alternativa Java a altres tecnologies com PHP.

El funcionament d'un servlet consisteix a rebre una petició, processar-la tenint en compte l'estat del sistema i les dades accessibles, i generar una resposta cap al client. El paquet de servlets proporciona classes i interfícies bàsiques per gestionar aquests fluxos de petició/resposta i per configurar el context d'execució del component.

Els servlets passen per tres etapes: primer, la fase d'inicialització, on s'executa el mètode `init()` i el component queda preparat per treballar; després, la fase de gestió de peticions, on la mateixa instància respon a les sol·licituds dels usuaris; i finalment, la fase de destrucció, quan el servidor allibera els recursos.

2. Instal·lació i configuració de les eines

Per fer la instal·lació de tots els programes s'han seguit els passos proporcionats pel professor a: <https://repo.fib.upc.es/felix.freitag/pti/-/tree/master/servlets>

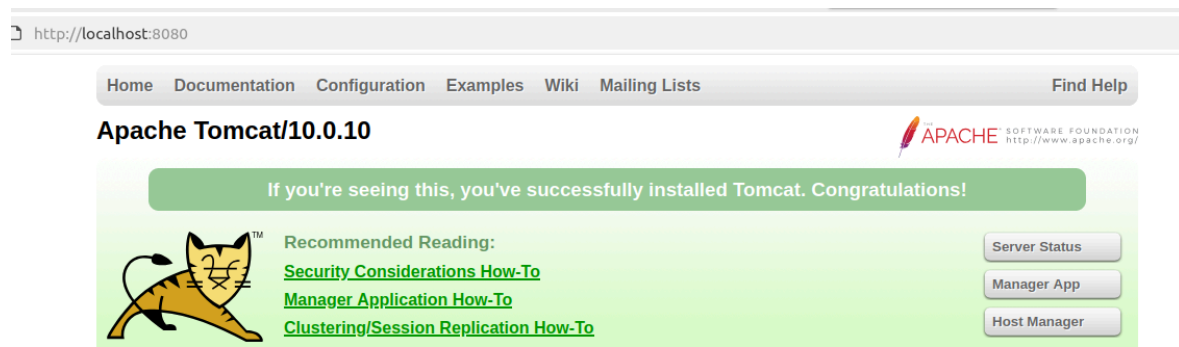
Tota la instal·lació s'ha fet a partir de la imatge d'Ubuntu que ofereix la FIB, la imatge que es troba als PC del laboratori, i per això moltes de les eines ja eren instal·lades prèviament.

Primer que tot s'ha executat `java -version`, aquest ja era instal·lat, va sortir el següent missatge.

```
alumine@nikipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/apache-tomcat-10.0.10$ java --version
openjdk 11.0.27 2025-04-15
OpenJDK Runtime Environment (build 11.0.27+6-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.27+6-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
```

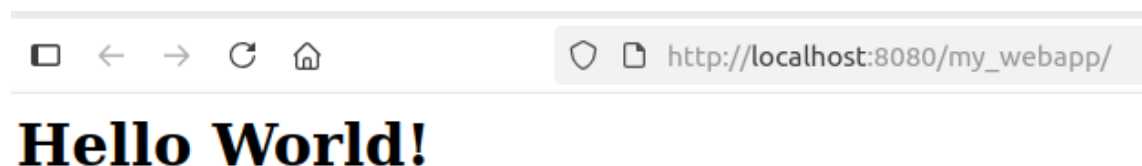
A continuació era necessari instal·lar tomcat. S'ha instal·lat Tommcat10, el qual es trobava a la carpeta proporcionada pel professor. S'ha utilitzat la comanda “`tar -xvzf apache-tomcat-10.0.10.tar.gz`” i s'ha treballat a partir d'aquest directori.

Un cop fet això s'ha iniciat el servei amb `./bin/startup.sh &` i s'ha comprovat que funcionava.



A continuació s'han fet algunes proves per fammiliaritzar-se amb l'entorn i els servlets. S'ha creat un directori d'aplicacions web (webapps) i allà un directori per les proves (my_webapp). Després d'executar els probes proposades pel professor a [l'enllac](#) ja proporcionat aquests van ser els resultats:

Resultat de la primera pàgina html:



Resultat del primmer servlet, dins del directori *mypackage*, després d'haver-lo inclòs al fitxer web.xml:

I'm a servlet!!

3. Objectius principals de la pràctica i implementació

L'objectiu d'aquesta pràctica és crear una pàgina web de lloguer de cotxes amb dues principals funcionalitats:

- 1) Fer una nova sol·licitud de lloguer mitjançant un formulari on l'usuari haurà d'introduir les dades necessàries: el tipus d'emissions de CO₂, el tipus de motor, el nombre de dies que es vol utilitzar, el nombre de vehicles a llogar i el descompte aplicat. Si la informació introduïda és vàlida, el sistema retornarà una llista amb els valors introduïts en cadascun dels camps.
- 2) Consultar el registre de tots els lloguers efectuats. Per accedir-hi, caldrà omplir un formulari amb una contrasenya coneguda només per l'administrador (admin, admin). Si la contrasenya és correcta, es mostrarà una llista amb totes les comandes enregistrades. Per fer possible aquesta funcionalitat, les sol·licituds de lloguer s'hauran d'anar desant en un fitxer JSON.

El codi de les pàgines ja es proporciona pel professor (carrental_home.html, carrental_form_new.html i carrental_form_list.html), i per això només és necessari programar els *servlets* (CarRentalNew.java i CarRentalList.java).

Finalment, també es [configura SSL/TLS](#) i es [“Dockeritza”](#) l'aplicació. A més a més, en l'apartat 4 es donen a terme diverses extensions, en concret es troba la implementació de [“docker compose”](#) d'un [“chatbot”](#).

3.1 Implementació CarRentalNew.java

La primera funcionalitat és la creació d'una nova comanda. A continuació es pot trobar el codi que s'executa en rebre una petició GET, el codi que es desglossarà en les diferents funcionalitats principals.

Primer s'obtenen els valors introduïts al formulari i es comprova la seva validesa:

Java

```
String rating = req.getParameter("co2_rating");
String days = req.getParameter("dies_lloguer");
String discount = req.getParameter("descompte");
String engine = req.getParameter("sub_model_vehicle");
String units = req.getParameter("num_vehicles");

boolean error = false;

if (rating.equals("54")) rating = rating + " (ExtraLow)";
else if (rating.equals("71")) rating = rating + " (Low)";
else if (rating.equals("82")) rating = rating + " (Medium)";
else if (rating.equals("139")) rating = rating + " (High)";
```

```

else error = true;

try {
    if (Integer.valueOf(units) <= 0) error = true;
    if (Integer.valueOf(days) <= 0) error = true;
    if (Float.valueOf(discount) > 100.f) error = true;
} catch (NumberFormatException e) {
    error = true;
}

```

Una vegada obtinguts mostrem les dades de la petició i executem la funció `saveRental(...)`, que s'encarregarà de desar les dades al fitxer JSON. Aquesta funció es mostra més a sota.

```

Java
if (!error) {
    out.println("<html>" +
        "<head><title>Rental Details</title></head>" +
        "<body>" +
        "<h1>Car Rental Details</h1>" +
        "<p>CO2 Rating: " + rating + "</p>" +
        "<p>Engine: " + engine + "</p>" +
        "<p>Days of Rental: " + days + "</p>" +
        "<p>Units: " + units + "</p>" +
        "<p>Discount: " + discount + "</p>" +
        "<hr/>" +
        "<br><br><a href='carrental_home.html'>Home</a>" +
        "</body>" +
        "</html>");

    saveRental(out, rating, days, discount, engine, units);
}

```

Per desar les dades primer es comprova si ja existeix el fitxer. Si es així aquí s'afejirà un nou objecte, dins d'un objecte mmés gran que es rentals. Si no existix es crea i es crea aquest objecte que contindrà totes les entrades, l'objecte rentals. Durant tot le procés es capturen les possibles excepcions. Tractar fitxers en pot provocar diverses.

```

Java
public void saveRental(PrintWriter out, String rating, String days, String
discount, String engine, String units) {
    String path = "/WEB-INF/classes/mypackage/rentals.json";

```

```

File file = new File(getServletContext().getRealPath(path));

out.println("<p>Writing in: " + file.getAbsolutePath() + "</p>");

JSONObject obj = null;
JSONArray array = null;

if (!file.exists()) {
    out.println("<p>Creating new JSON file</p>");
    obj = new JSONObject();
    array = new JSONArray();
    obj.put("rentals", array);
}
else {
    JSONParser parser = new JSONParser();
    try {
        out.println("<p>JSON existst, updating file</p>");
        obj = (JSONObject) parser.parse(new FileReader(file));
        array = (JSONArray) obj.get("rentals");
    } catch (Exception e) {
        out.println("<p>Error on reading JSON file: " + e.getMessage()
            + "</p>");
        e.printStackTrace(new PrintWriter(out));
        return;
    }
}

JSONObject newRental = new JSONObject();

newRental.put("rating", rating);
newRental.put("engine", engine);
newRental.put("rental_days", days);
newRental.put("units", units);
newRental.put("discount", discount);
array.add(newRental);

try {
    FileWriter fileWriter = new FileWriter(file);
    fileWriter.write(obj.toJSONString());
    fileWriter.flush();
    out.println("<p>The JSON file was updated</p>");
} catch (IOException e) {
    out.println("<p>Error on writing on JSON file: " + e.getMessage() +
"</p>");
    e.printStackTrace(new PrintWriter(out));
    return;
}
}

```


Una vegada implementat aquest és el resultat que s'obté una vegada enviada una comanda. També s'han inclòs alguns missatges de Debugging.



3.2 Implementació CarRentalList.java

La segona funcionalitat és llistar totes les comandes si s'introdueix correctament l'usuari i contrasenya d'un administrador. A continuació es pot trobar el codi que s'executa en rebre una petició GET, el codi que es desglossarà en les diferents funcionalitats principals.

Primer obtenim les dades introduïdes al formulari d'inici de sessió i comprovem si són les dades d'administrador (admin, admin). Si no

```
Java
String nombre = req.getParameter("userid");
String passwd = req.getParameter("password");

if (nombre.equals("admin") && passwd.equals("admin")){
    out.println("<html><h1>Rentals list</h1><hr></html>");
    ... //El codi d'aquesta part es troba a sota
}
else {
    out.println("<html><h1>No Permission (username and password are both
'admin')</h1></html>");
}
```

Si les dades eren correctes a les hores s'executa la següent operació sobre el fitxer JSON (Aquest codi es troba en la secció de (...) al codi anterior):

Java


```
String path = "/WEB-INF/classes/mypackage/rentals.json";
File file = new File(getServletContext().getRealPath(path));

JSONParser parser = new JSONParser();
JSONObject obj = null;
JSONArray array = null;

if (file.exists()) {
    try {
        obj = (JSONObject) parser.parse(new FileReader(file));
        array = (JSONArray) obj.get("rentals");
    } catch (Exception e) {
        out.println("<p>Error on reading JSON file: " + e.getMessage() +
"</p>");
        e.printStackTrace(new PrintWriter(out));
        return;
    }

    int cont = 1;
    for (Object o : array) {
        JSONObject currentObj = (JSONObject)o;
        out.println(
            "<h2>Car Rental N° " + cont++ + "</h2>" +
            "<p>CO2 Rating: " + currentObj.get("rating") + "</p>" +
            "<p>Engine: " + currentObj.get("engine") + "</p>" +
            "<p>Days of Rental: " + currentObj.get("rental_days") + "</p>" +
            "<p>Units: " + currentObj.get("units") + "</p>" +
            "<p>Discount: " + currentObj.get("discount") + "</p>" +
            "<hr/>");
    }
} else {
    out.println("<p>No JSON file was found: There are no rentals
yet</p>");
}
```

Una vegada implementat aquest és el resultat que s'obté una vegada introduïdes les dades correctament:

 http://localhost:8080/my_webapp/list	
Rentals list	
Car Rental Nº 1	
CO2 Rating: 54 (ExtraLow)	
Engine: Hybrid	
Days of Rental: 1	
Units: 1	
Discount: 0.0	
Car Rental Nº 2	
CO2 Rating: 54 (ExtraLow)	
Engine: Hybrid	
Days of Rental: 1	
Units: 1	
Discount: 2.0	
Car Rental Nº 3	
CO2 Rating: 54 (ExtraLow)	
Engine: Hybrid	
Days of Rental: 3	
Units: 1	
Discount: 2.0	
Car Rental Nº 4	
CO2 Rating: 82 (Medium)	
Engine: Electric	
Days of Rental: 34	

3.3 Configuració SSL/TLS

Per configurar l'accés a l'aplicació mitjançant el protocol https cal configurar SSL/TLS. Per fer això s'ha executat primer la següent comanda, per crear un certificat en l'aplicació:

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

Després s'ha configurat server.xml per configurar el conector, introduint el següent codi:


```
XML
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"

sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"
    port="8443"
    maxThreads="150"
    SSLEnabled="true">
    <SSLHostConfig>
        <Certificate
            certificateKeystoreFile="${user.home}/.keystore"
            type="RSA"
        />
```

```
</SSLHostConfig>
</Connector>
```

Una vegada fet això, després de reiniciar tomcat, ja s'ha pogut accedir a l'aplicació mitjançant el protocol segur: <https://localhost:8443>. En fer això abans d'accedir apareix el missatge "connection is not private", indicant que el procediment s'ha fet correctament


Not Secure localhost:8443

 **Warning: Potential Security Risk Ahead**

Firefox detected a potential security threat and did not continue to localhost. If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

[Learn more...](#)

[Go Back \(Recommended\)](#) [Advanced...](#)

 [localhost:8443/my_webapp/carrental_form_new.html](#)

New rental

CO2 rating:

Extralow ▾

Engine:

Hybrid ▾

Number of days:

1

Number of units:

1

Discount(%):

0.0

[Submit \(GET\)](#)

[Home](#)

3.4 Configuració Docker

Primer que tot s'ha hagut de seguir el passos d'instal·lació i configuració de Docker seguint la referència del professor a <https://repo.fib.upc.es/felix.freitag/pti/-/blob/master/docker.md>. A continuació s'indiquen les diferents comandes utilitzades:

```
Shell
#Instalar
sudo apt-get update
wget -q0- https://get.docker.com/ | sh
```

```
sudo usermod -aG docker $(whoami)
newgrp docker

#Comprovar instalacio
docker run hello-world

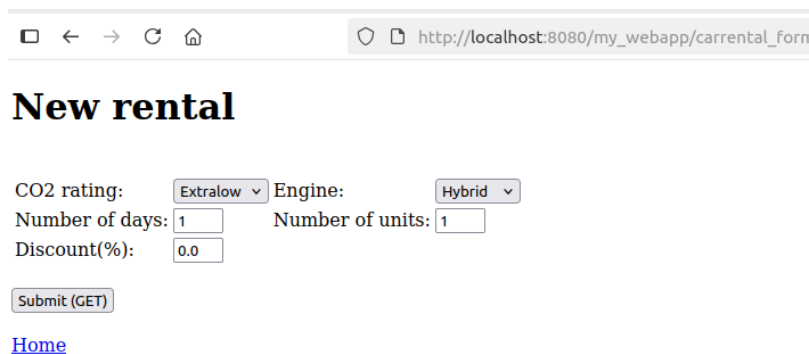
#Preparar per PTI
mkdir $HOME/pti
docker run -it --name pti -v $HOME/pti:/pti -p 8080:8080 -p 8443:8443 ubuntu
bash
```

Després ho hem configurat per utilitzar amb la nostra webapp. Des del directori “webapps” s’ha creat el fitxer Dockerfile i s’ha introduït el següent:

```
None
FROM tomcat:10
COPY my_webapp /my_webapp
WORKDIR /
RUN cp -r my_webapp /usr/local/tomcat/webapps
```

Després s’ha executat la imatge amb `docker build -f Dockerfile -t carrental`.

Finalment s’ha executat `docker run --name carrental -d -p 8080:8080 -p 8443:8443 carrental`.



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/my_webapp/carrental_form`. The page content includes a heading **New rental**, followed by form fields for **CO2 rating:** (with a dropdown menu showing 'Extralow'), **Engine:** (with a dropdown menu showing 'Hybrid'), **Number of days:** (with an input field containing '1'), and **Number of units:** (with an input field containing '1'). Below these are **Discount(%):** (with an input field containing '0.0') and a **Submit (GET)** button. At the bottom, there is a [Home](#) link.

A l’imatge es veu com s’ha desplegat l’aplicació amb docker run.

4. Extensions de la pràctica

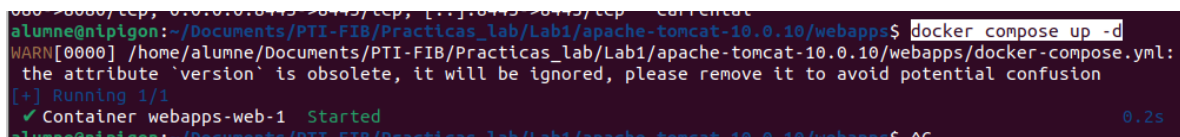
En la part final de la pràctica s'ha configurat *docker compose* i també s'ha implementat un chatbot.

4.1 Configuració Docker Compose

Aquesta configuració és per evitar haver de desplegar sempre l'aplicació amb *docker run*. Per això crearem el fitxer *docker-compose.yml*:

```
None
version: '3.8'
services:
  web:
    build: .
    ports:
      - "8080:8080"
      - "8443:8443"
```

En executar *docker compose up -d* ja tenim l'aplicació corrent.



```
alunne@nlpigon: ~/Documents/PTI-FIB/Practicas_lab/Lab1/apache-tomcat-10.0.10/webapps$ docker compose up -d
WARN[0000] /home/alunne/Documents/PTI-FIB/Practicas_lab/Lab1/apache-tomcat-10.0.10/webapps/docker-compose.yml:
the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 1/1
✔ Container webapps-web-1 Started                                0.2s
```

4.2 Configuració chatbot

Per dur a terme tota la configuració s'ha seguit l'exemple del professor a: https://repo.fib.upc.es/felix.freitag/pti/-/tree/master/servlets/chatbot-car_rental.

Primer que tot s'ha preparat l'entorn i s'ha activat. A més a més, s'han instal·lat les dependències necessàries. Per fer aquestes s'han executat les següents comandes:

```
Shell
python3 -m venv envchatbotcarrental #Preparar entorn
source envchatbotcarrental/bin/activate #Activar entorn
pip3 install -r requirements.txt #Instalar dependències
```

Després d'això s'han instal·lat els models d'Ollama, concretament *llama3.2*, *llama 3.2:1b*, *llama 3.1*.

Shell

```
curl -fsSL https://ollama.com/install.sh | sh #Descarregar Ollama
#Descarregar els diferents models
ollama run llama3.2
ollama run llama3.2:1b
ollama run llama3.1
```

A continuació s'ha seguit el tutorial previ al treball a fer. Primer s'ha executat `create_index.py` i després `query_index.py`. Així s'ha pogut fer la query d'exemple al chatbot, i s'ha obtingut el següent output:

```
bot-car_rental$ python3 query_index.py
Vector index loaded successfully!
Chatbot ready! Type 'exit' to quit.
-----
Enter your query: Can you check which engine types have a low rating?
Llama3 Chatbot: Based on the provided data, I found that two engine types have a
"Low" rating:

1. Diesel
2. Gasoline
-----
Enter your query:
```

4.2.1 Mesura de la duració

Per tal de comparar models s'ha afegit un timestamp, per fer això s'ha afegit el següent:

```
Python
start_time = time.time() #Afegit al principi de tot del dos programes

#...

end_time = time.time() #Afegit al final dels programes
duration = end_time - start_time #Obte el temps que ha portat la resposta

#Imprimeix quant temps ha portat la resposta (Es mostra en concret el de
create_index.py)
print(f"[{time.strftime('%Y-%m-%d %H:%M:%S')}] Indexing completed in
{duration:.2f} seconds!")
```

Després s'han fet comparacion de temps amb diferents models, s'han obtingut els següents resultats, per fer-ho s'ha modificat la variable model del codi:

Model llama3.2 → Index: 11.12s / Query: 18.66s

```
(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$ python3 create_index.py
[2025-09-24 18:32:00] Indexing started...
/home/alumne/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental/envchatbotcarrental/lib/python3.10/site-packages/langchain/indexes/vectorstore.py:127: UserWarning: Using InMemoryVectorStore as the default vectorstore. This memory store won't persist data. You should explicitly specify a vectorstore when using VectorstoreIndexCreator
  warnings.warn(
[2025-09-24 18:32:12] Indexing completed in 11.12 seconds!
(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$
```

```
(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$ python3 query_index.py
Vector index loaded successfully!
Chatbot ready! Type 'exit' to quit.
-----
Enter your query: Can you check which engine types have a low rating?
Llama3 Chatbot: According to the data provided, there are two engine types with a "Low" rating:

1. Diesel
2. Gasoline (specifically in the "Low" category)

Note that some Diesel models also have a "Low" rating, I've only mentioned it separately as it's listed individually.
Query answered in 18.66 seconds.
-----
Enter your query:
```

Model llama 3.2:1b → Index: 6.13s / Query: 11.08s

```
(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$ python3 create_index.py
[2025-09-24 18:37:55] Indexing started...
/home/alumne/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental/envchatbotcarrental/lib/python3.10/site-packages/langchain/indexes/vectorstore.py:127: UserWarning: Using InMemoryVectorStore as the default vectorstore. This memory store won't persist data. You should explicitly specify a vectorstore when using VectorstoreIndexCreator
  warnings.warn(
[2025-09-24 18:38:01] Indexing completed in 6.13 seconds!
(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$
```



```

(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$ python3 query_index.py
Vector index loaded successfully!
Chatbot ready! Type 'exit' to quit.
-----
Enter your query: Can you check which engine types have a low rating?
Llama3 Chatbot: Based on the provided data, I've checked the ratings for each engine type. Here are the results:

* Gasoline: 6 (Medium) and 4 (Low)
* Diesel: 5 (High), 7 (Medium), and 4 (Low)

So, the engines with a low rating are:

1. Gasoline - Medium
2. Gasoline - Low
3. Diesel - High
4. Diesel - Medium

Let me know if you have any further questions!
Query answered in 11.08 seconds.
-----
Enter your query: 

```

Model llama 3.1 → Index: 26.05s / Query: 26.56s

```

bot-car_rental$ python3 create_index.py
[2025-09-24 18:55:04] Indexing started...
/home/alumne/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental/envchatbotcarrental/lib/python3.10/site-packages/langchain/indexes/vectorstore.py:127: UserWarning: Using InMemoryVectorStore as the default vectorstore. This memory store won't persist data. You should explicitly specify a vectorstore when using VectorStoreIndexCreator
  warnings.warn(
[2025-09-24 18:55:30] Indexing completed in 26.05 seconds!
(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$ 

```

```

(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_rental$ python3 query_index.py
Vector index loaded successfully!
Chatbot ready! Type 'exit' to quit.
-----
Enter your query: Can you check which engine types have a low rating?
Llama3 Chatbot: Based on the data, I see that the "Diesel" engine has a "Low" rating.
Query answered in 26.56 seconds.
-----
Enter your query: 

```

Resum dels resultats:

Model	Index (s)	Query (s)
Llama3.2	11.12	18.66
Llama3.2:1b	6.13	11.08
Llama3.1	26.05	26.56

4.2.2 Qualitat de la resposta

Per comparar les respostes s'utilitzen les respostes de l'apartat [anterior](#). A continuació hi ha una petita valoració de cada resposta:

Llama3.2

```
Llama3 Chatbot: According to the data provided, there are two engine types with a "Low" rating:

1. Diesel
2. Gasoline (specifically in the "Low" category)

Note that some Diesel models also have a "Low" rating, I've only mentioned it separately as it's listed individually.
```

Aquesta resposta és una mica estranya, ja que sí menciona els motors que tenen Low rating, però els comentaris que fa a part no tenen gaire sentit.

Llama3.2:1b

```
Llama3 Chatbot: Based on the provided data, I've checked the ratings for each engine type. Here are the results:

* Gasoline: 6 (Medium) and 4 (Low)
* Diesel: 5 (High), 7 (Medium), and 4 (Low)

So, the engines with a low rating are:

1. Gasoline - Medium
2. Gasoline - Low
3. Diesel - High
4. Diesel - Medium
```

Aquesta resposta és errònia, ja que les dades que menciona no corresponen amb la realitat. A més a més al final fa un resum que no respon res a la pregunta.

Llama3.1

```
...your query: can you check which engine types have a low rating.
Llama3 Chatbot: Based on the data, I see that the "Diesel" engine has a "Low" rating.
```

Aquesta resposta és molt poc completa, i només detecta dièsel com a motor amb algun Low, quan no és així.

Per tant, es pot afirmar que cap de les respostes és del tot satisfactòria.

4.2.3 Canviar la representació del dataset

Per fer això s'ha modificat l'arxiu rentals.csv i s'ha convertit a rentals.pdf afegint més text a sobre. Després s'ha modificat els codis per reconèixer aquest dataset i s'ha executat. Per aquesta prova s'ha documentat únicament el model Llama3.2.

Per fer el canvi de .csv a .pdf s'ha utilitzat el següent codi, realitzat amb ajuda de Chatgpt.

```
Python
import pandas as pd
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle,
Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib import colors

# Cargar el CSV
df = pd.read_csv("dataset/rentals.csv")

# Crear PDF
pdf_file = "dataset/rentals.pdf"
doc = SimpleDocTemplate(pdf_file, pagesize=letter)
styles = getSampleStyleSheet()
elements = []

# Texto introductorio
elements.append(Paragraph("Rental Dataset Report", styles['Title']))
elements.append(Spacer(1, 12))
elements.append(Paragraph("This document contains the rental dataset
enriched with additional information for analysis.", styles['Normal']))
elements.append(Spacer(1, 12))

# Convertir DataFrame a tabla
table_data = [df.columns.tolist()] + df.values.tolist()
table = Table(table_data)

# Estilos de la tabla
table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('GRID', (0, 0), (-1, -1), 0.25, colors.black),
    ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
    ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
]))

elements.append(table)

# Guardar PDF
doc.build(elements)
print("PDF creado en:", pdf_file)
```

Una vegada fet això en `create_index.py` s'ha canviat `TextLoader` per `PyPDFLoader` i s'ha canviat l'arxiu pel pdf. A `query_index.py` no ha estat necessari fer res, ja que llegeix igual del `pkl`.

Aquests han estat els resultats d'executar els dos scripts del chatbot.

```
bot-car_rental$ python3 create_index.py
[2025-09-24 19:38:07] Indexing started...
/home/alumne/Documents/PTI-FIB/Practicas_Lab/Lab1/chatbot-car_rental/envchatbot-arrental/lib/python3.10/site-packages/langchain/indexes/vectorstore.py:127: UserWarning: Using InMemoryVectorStore as the default vectorstore. This memory store won't persist data. You should explicitly specify a vectorstore when using VectorstoreIndexCreator
  warnings.warn(
[2025-09-24 19:38:20] Indexing completed in 13.08 seconds!
```

```
bot-car_rental$ python3 query_index.py
Vector index loaded successfully!
Chatbot ready! Type 'exit' to quit.
-----
Enter your query: Can you check which engine types have a low rating?
Llama3 Chatbot: Based on the provided data, I can see that the following engines have a "Low" rating:

- Diesel
- Gasoline (specifically "Low" gasoline)
Query answered in 16.72 seconds.
-----
```

Podem veure que el resultat és molt semblant. Ha trigat una mica més a generar el `.pkl`, però és una diferència petita. La resposta és molt semblant a l'anterior, i segueix sense ser gaire satisfactòria.

4.2.4 RAG vs non-RAG

En aquest apartat es farà que el chatbot no utilitzi RAG (vector index). Per fer això s'ha fet el script `query_nonRAG.py`. El codi es mostra a continuació, per aquest codi també s'ha utilitzat com a eina ChatGPT.

```
Python
from langchain_ollama import ChatOllama
import time

def query_chatbot():
    """Interactive chatbot without using the vector index (non-RAG mode)"""

    # Create a ChatOllama object
    chat_llama3 = ChatOllama(model="llama3.2", temperature=0.7)

    print("Chatbot (non-RAG) ready! Type 'exit' to quit.")
    print("-" * 40)

    prompt = ""
    while prompt.lower() != "exit":
```

```

prompt = input("Enter your query: ")

if prompt.lower() == "exit":
    print("Goodbye!")
    break

try:
    start_time = time.time()
    # Direct query to the LLM, no vector index
    answer = chat_llama3.invoke(prompt)
    end_time = time.time()

    duration = end_time - start_time
    print(f"Llama3 Non-RAG Chatbot: {answer.content}")
    print(f"Query answered in {duration:.2f} seconds.")
    print("-" * 40)
except Exception as e:
    print(f"Error: {e}")
    print("-" * 40)

if __name__ == "__main__":
    query_chatbot()

```

Els resultats obtinguts han estat els següents en fer la query:

```

Enter your query: Can you check which engine types have a low rating?
Llama3 Non-RAG Chatbot: I can provide information on various engine types, but I would need more context or specific details about the engines you are referring to. Could you please provide more information or clarify which engine types you would like me to check?

If you're looking for general information on engine ratings, here are some common engine types and their relative fuel efficiency or emissions:

1. Gasoline engines:
* Low-efficiency: V8 engines with high displacement (e.g., 6.2L) and old technology.
* Mid-range: V6 engines with moderate displacement (e.g., 3.5L) and average technology.
* High-efficiency: Turbocharged or supercharged gasoline engines with smaller displacement (e.g., 2.0L).
2. Diesel engines:
* Low-efficiency: Older diesel engines with high compression ratios and old technology.
* Mid-range: Modern diesel engines with moderate compression ratios (e.g., 16:1) and average technology.
* High-efficiency: Turbocharged or supercharged diesel engines with low compression ratios (e.g., 14:1).
Query answered in 20.91 seconds.

```

La query et dona una resposta molt més completa com que no es basa en el teu dataset. Però això fa que aquesta resposta no sigui relativa a les nostres dades sino a les que ja té el model en si. Pel que tot i que sembla una resposta més satisfactòria pot no ser adequada si el que busquem és que es faci la resposta en el context de les nostres dades.

4.2.5 Test autoproposat:

Com a test autoproposat s'ha decidit fer un test de memòria necessària per a cada model. La principal motivació d'això ha estat que mentre es feia el primer test, en executar el model llama3.1, no podia per falta de memòria RAM. En tancar el navegador s'ha solucionat, però m'ha deixat la curiositat de saber quanta RAM utilitza cadascun.

Per fer això s'ha fet el següent codi amb ajuda de ChatGPT. El fitxer és memory_test_models.py

```
Python
import psutil
import time
from langchain_ollama import ChatOllama

def measure_model(model_name, question="Can you check which engine types
have a low rating?"):
    """Measure memory and response time for a given model"""
    print(f"\n=== Testing model: {model_name} ===")

    # Medir memoria antes
    mem_before = psutil.virtual_memory().available / (1024 ** 3) # GB

    # Crear modelo
    try:
        chat = ChatOllama(model=model_name, temperature=0.7)
    except Exception as e:
        print(f"Could not load {model_name}: {e}")
        return

    # Preguntar
    start_time = time.time()
    try:
        answer = chat.invoke(question)
        end_time = time.time()
    except Exception as e:
        print(f"Query failed: {e}")
        return

    # Medir memoria después
    mem_after = psutil.virtual_memory().available / (1024 ** 3) # GB

    # Calcular métricas
    duration = end_time - start_time
    mem_used = mem_before - mem_after

    # print(f"Answer: {answer.content[:200]}...") # No interessa
    la resposta ara mateix
    print(f"Response time: {duration:.2f} seconds")
```

```

print(f"Approx memory used: {mem_used:.2f} GB")

if __name__ == "__main__":
    # Compara tots els models
    measure_model("llama3.2")
    measure_model("llama3.2:1b")
    measure_model("llama3.1")

```

Per la primera execució s'ha decidit deixar el navegador obert, s'esperava que fallés llama3.1.

```

(envchatbotcarrental) alumne@nipigon:~/Documents/PTI-FIB/Practicas_lab/Lab1/chatbot-car_
rental$ python3 memory_test_models.py

=== Testing model: llama3.2 ===
Response time: 39.49 seconds
Approx memory used: 2.26 GB

=== Testing model: llama3.2:1b ===
Response time: 36.58 seconds
Approx memory used: 1.17 GB

=== Testing model: llama3.1 ===
Query failed: model requires more system memory (5.6 GiB) than is available (4.6 GiB) (s
tatus code: 500)

```

Es pot veure que com s'esperava falla llama3.1.

Finalment, s'ha provat tancant el navegador per fer la comparació. En fer diverses proves s'ha vist que no acaba de funcionar del tot bé el sistema. S'esperaria que llama3.1 utilitzés més memòria, però en tests que s'han dut a terme s'observa que necessita al voltant d'1,5 GB. Això no quadra amb la hipòtesi. Per una altra banda, cal destacar que en alguna execució algun dels valors de RAM surt com a negatiu, indicant clarament un error.

A continuació hi ha una taula amb els valors mitjans de 10 execucions del programa. S'han eliminat de la mostra aquelles execucions amb un valor clarament erroni (valors negatius o molt allunyats de la resta)

Models	Valor mig d'ocupació de RAM (Gb)
llama3.2	2.60
llama3.2:1b	1.37
llama3.1	1.33