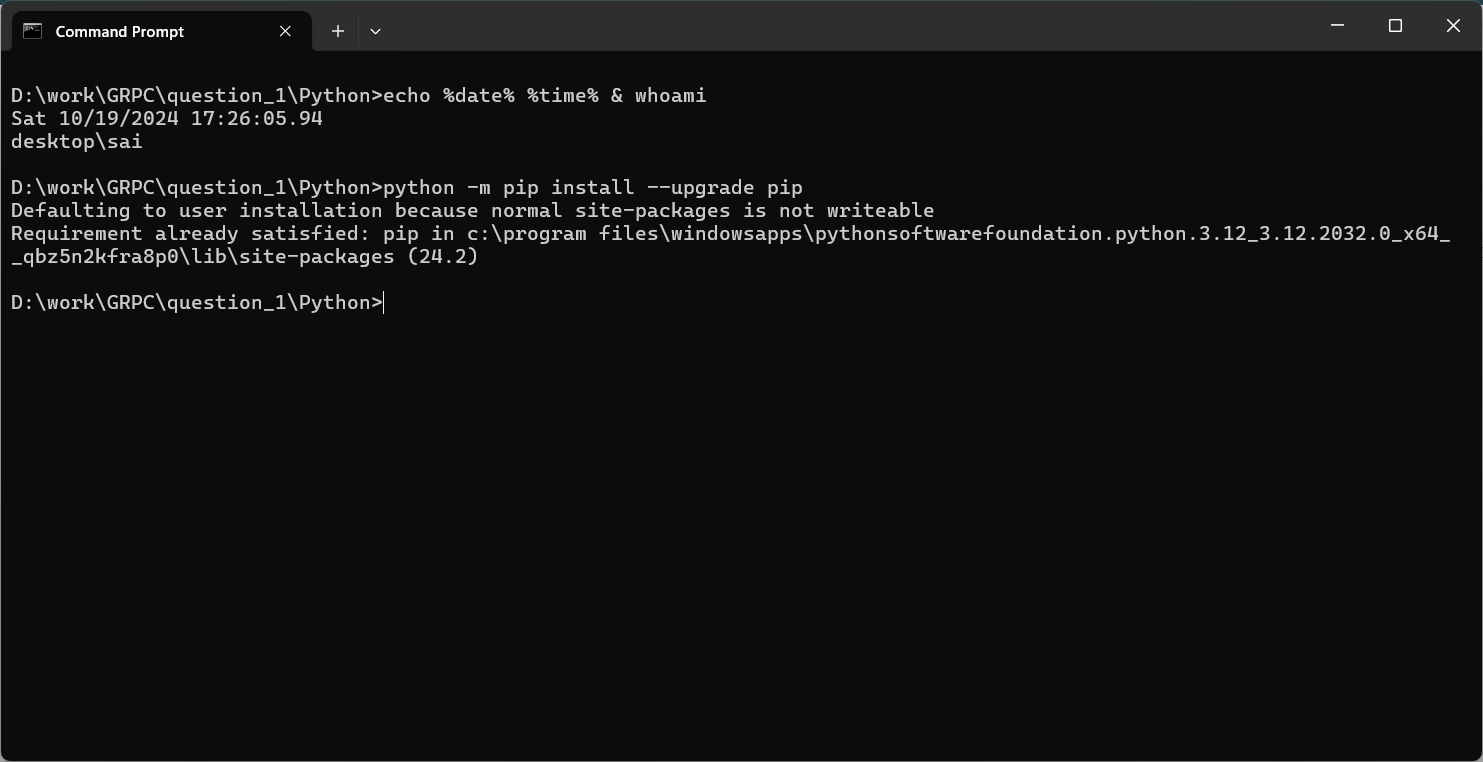
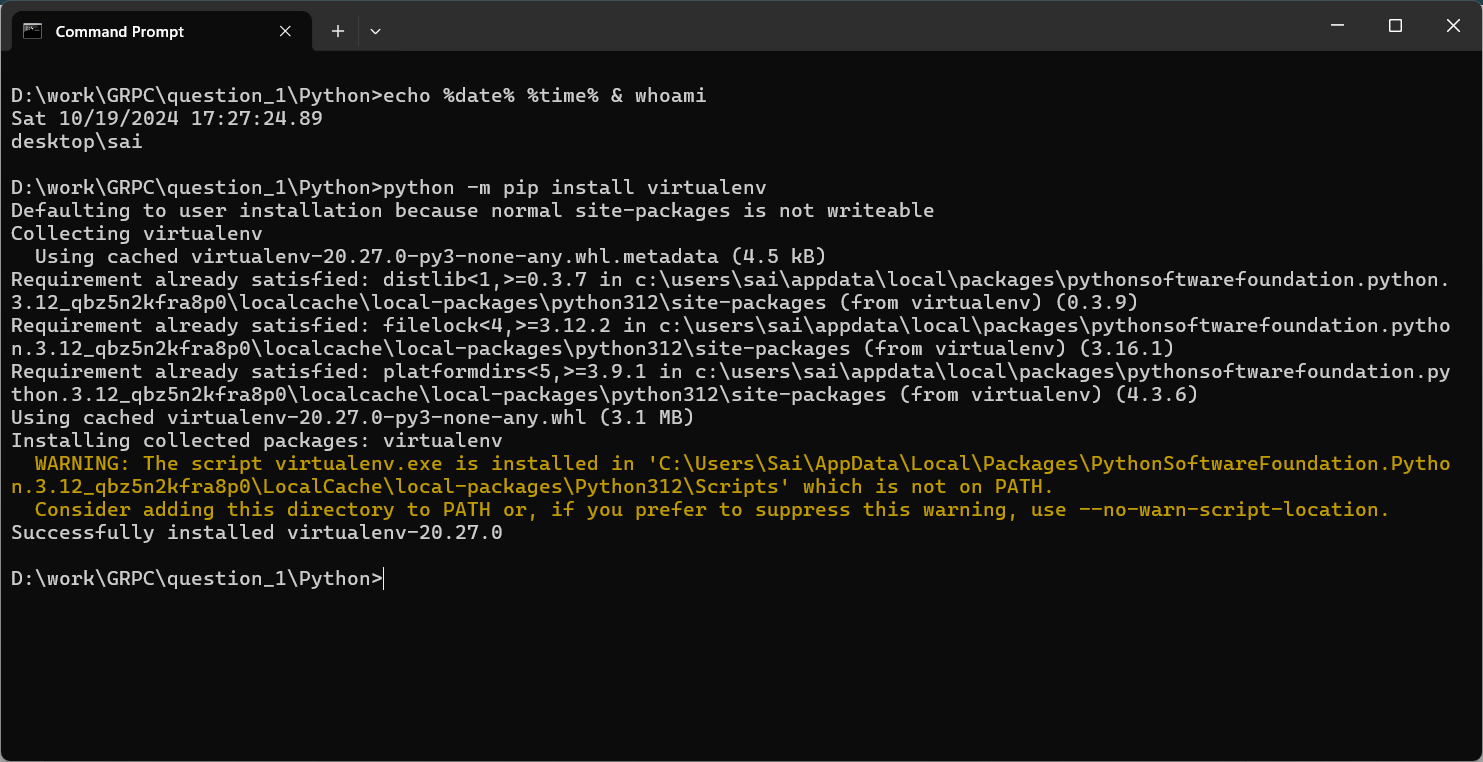
**Question1:**

**Python:** For the python language we will be following the method as described by the gRPC to perform the related exercises and make our way through this assignment.

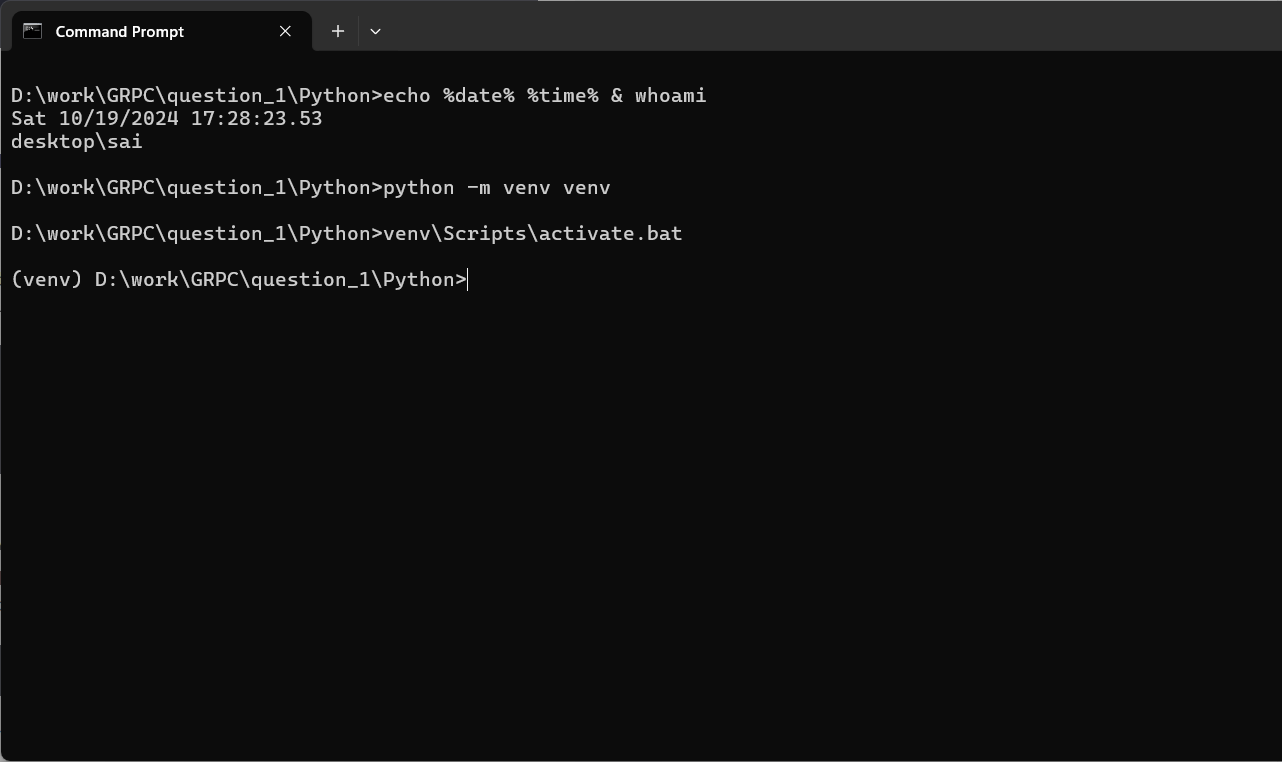
**Step 1: Install or upgrade packages:** We will be first installing all the necessary library requirements that are needed to run the gRPC program in python language. We can do this by installing pip and using pip command to upgrade the pip command, as it would be easier for us later on.

****

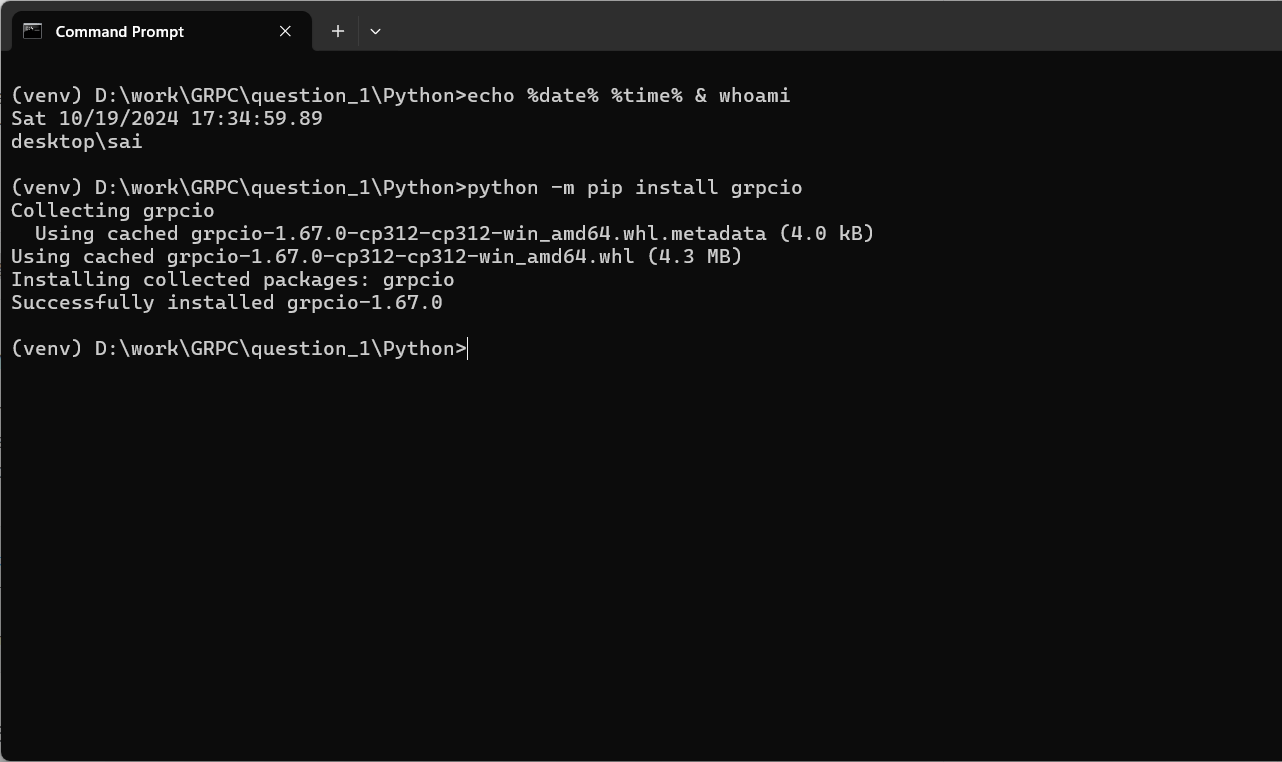
**Step 2: Installing virtual environment:** We will be installing a virtual environment on our system so that we can easily communicate between two virtual environments and perform our experiments with ease. We can do this by installing virtual environment for python using the pip command.

****

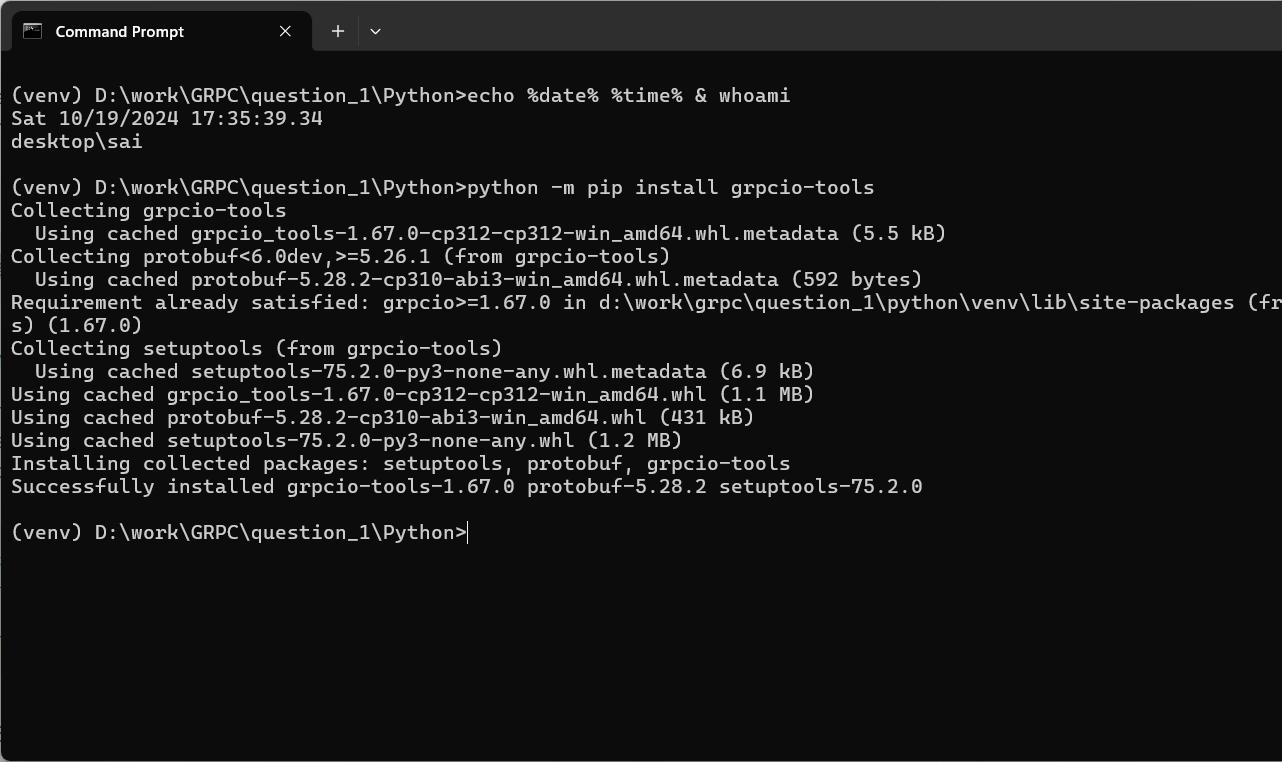
**Step 3: Set virtual environment:** After installing the virtual environment, we need to activate it and make necessary changes for the gRPC to run.

****

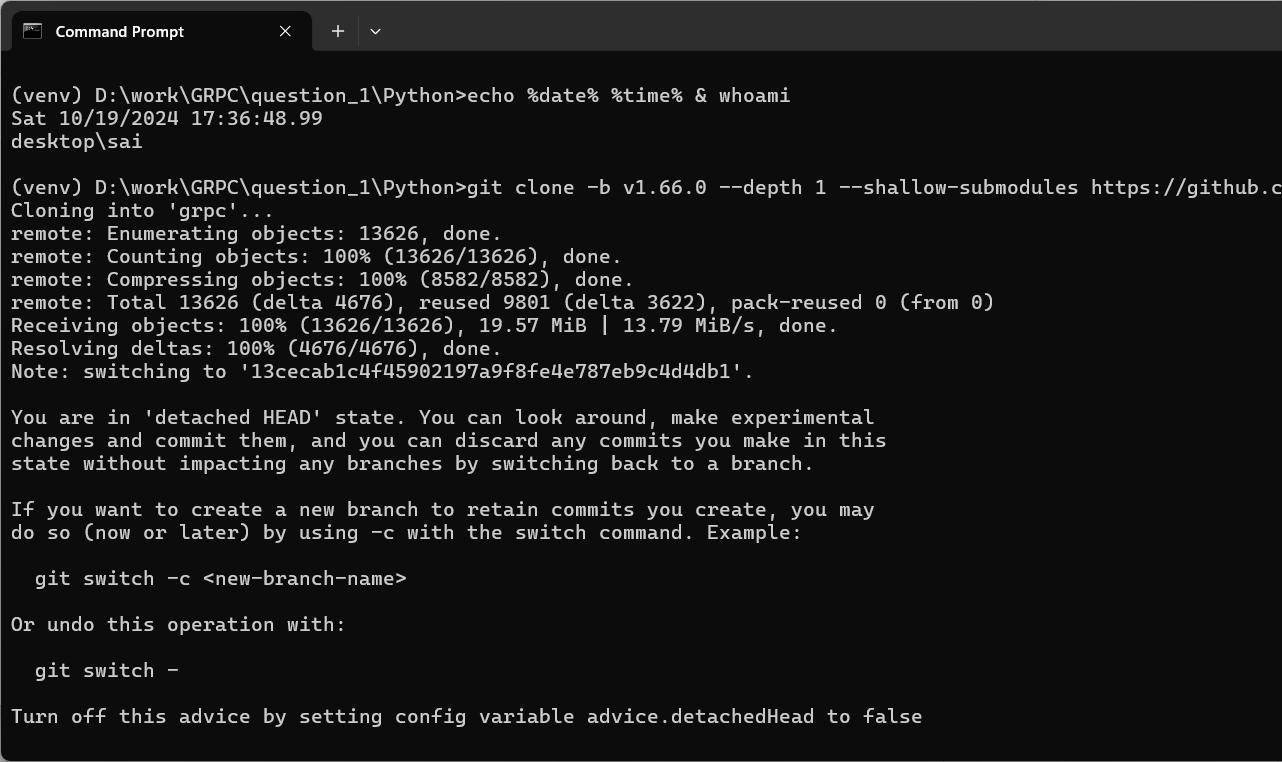
**Step 4: Installing grpcpio:** Now we will start by install all the necessary gRPC libraries and the first one is grpcio, which is a library that is a must for i/o operations using gRPC.

****

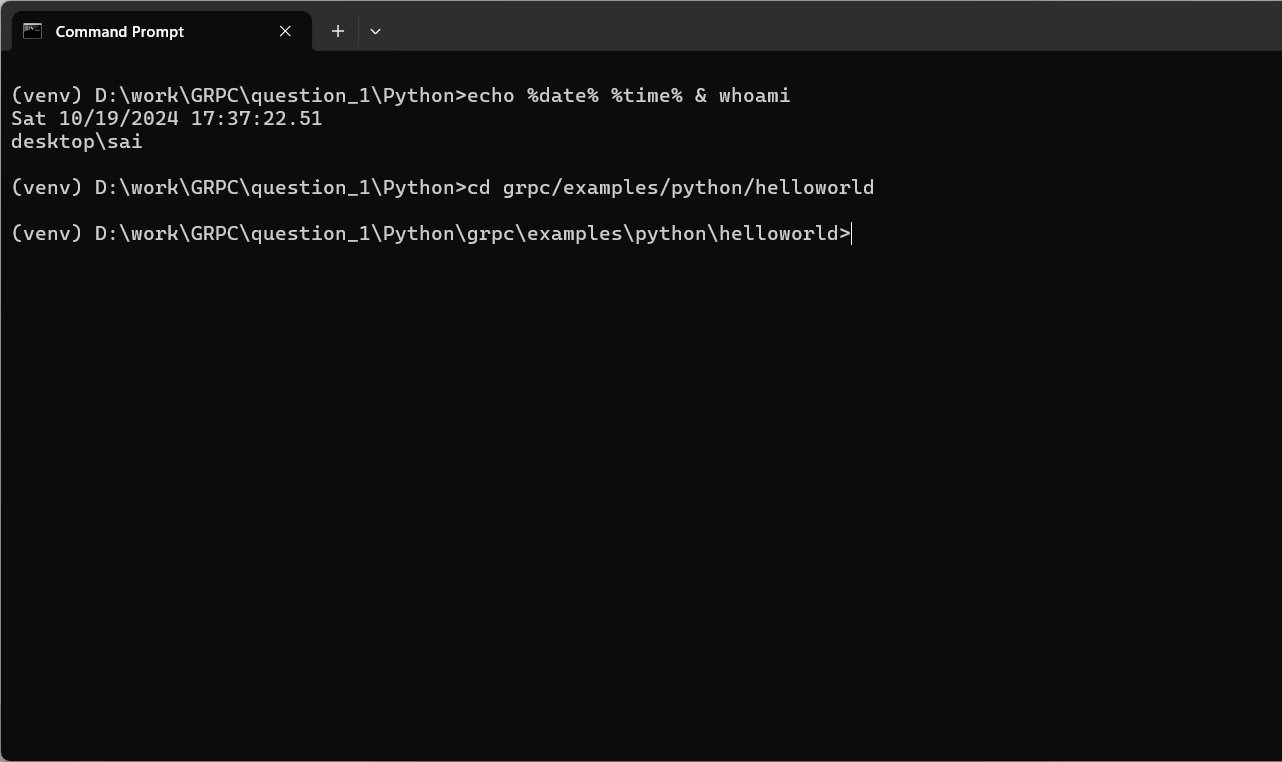
**Step 5: Installing grpcio-tools:** Then we will be installing the grpcio-tools using the same pip command.

****

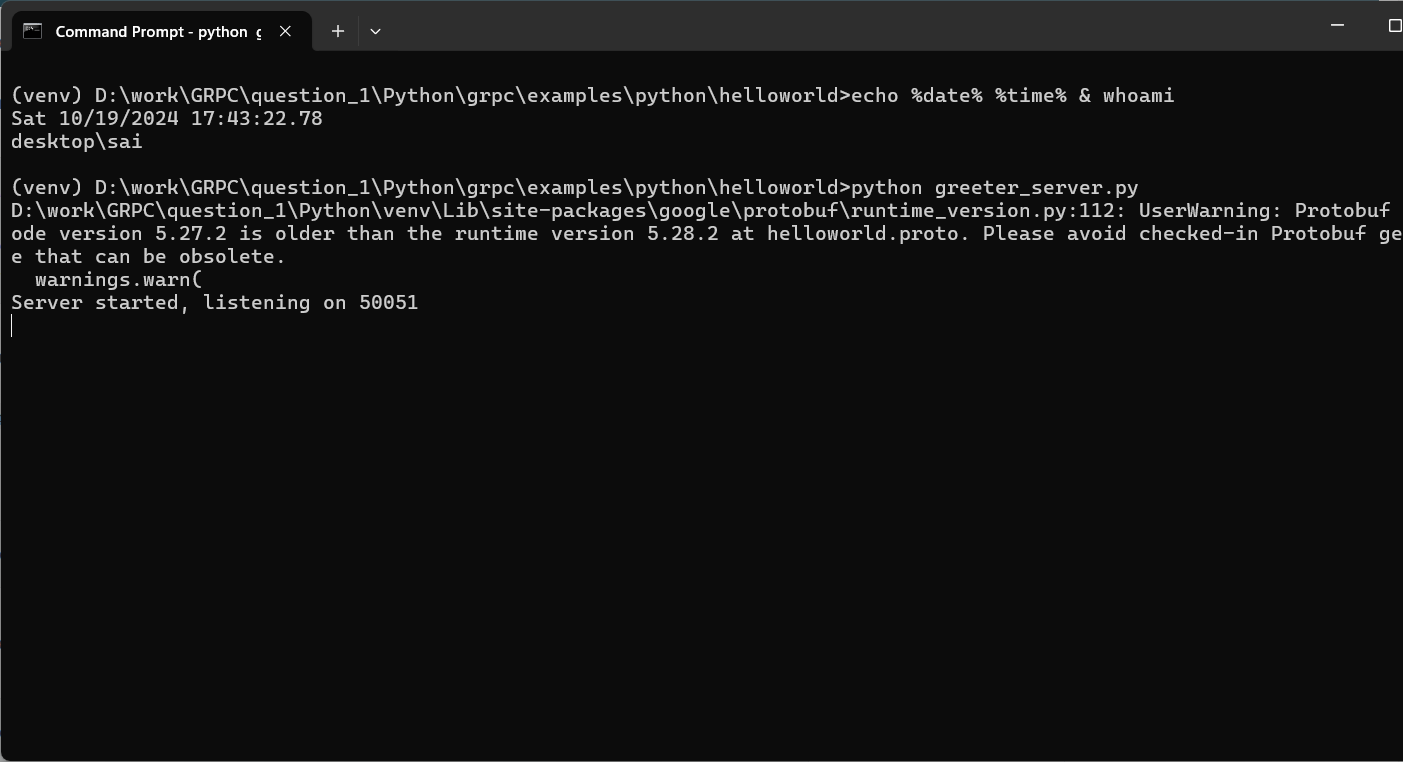
**Step 6: Cloning git repository:** We will clone the git repository using git clone command and install the starter code in our working directory.

****

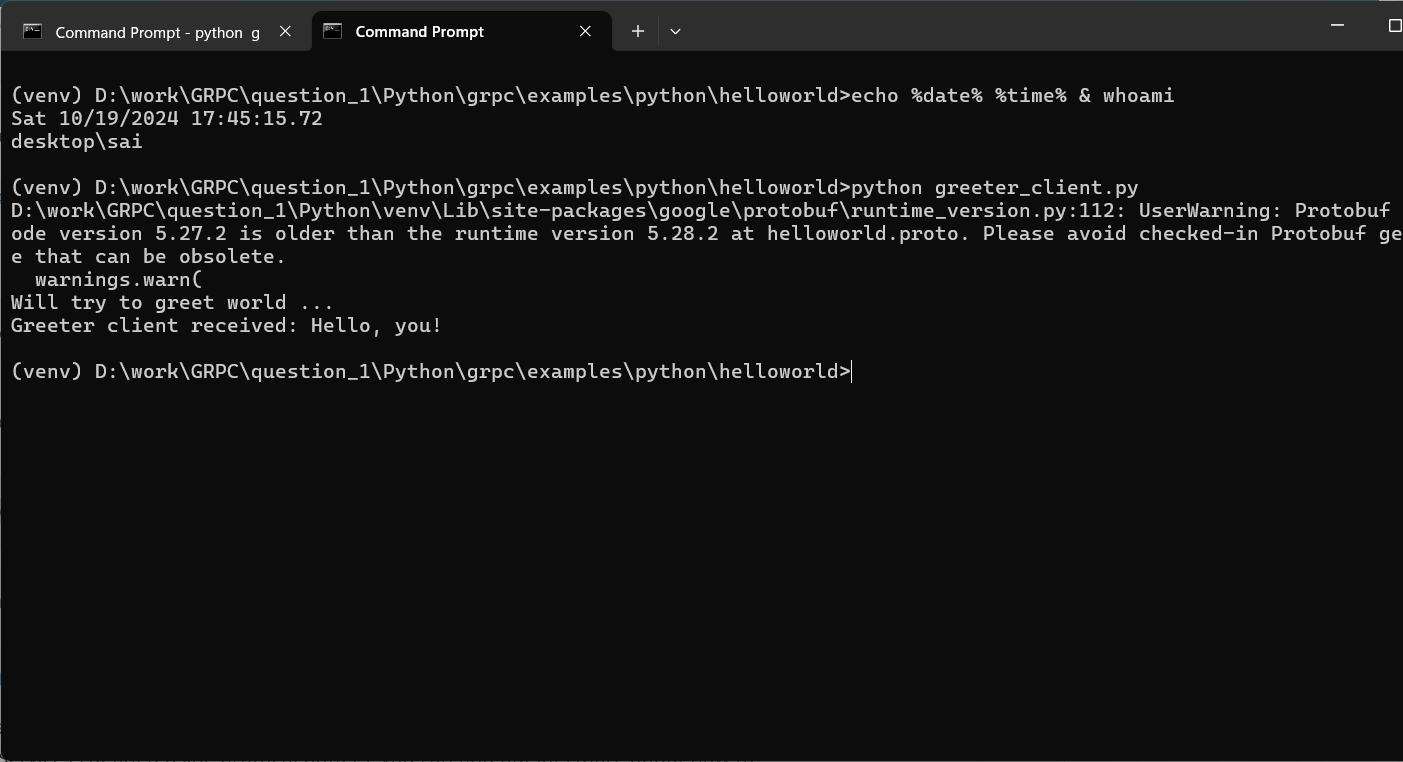
**Step 7: Navigating to grpc/example/python/helloworld:** For the next step, we will navigate to the helloworld folder that is in the example directory and then python folder.

****

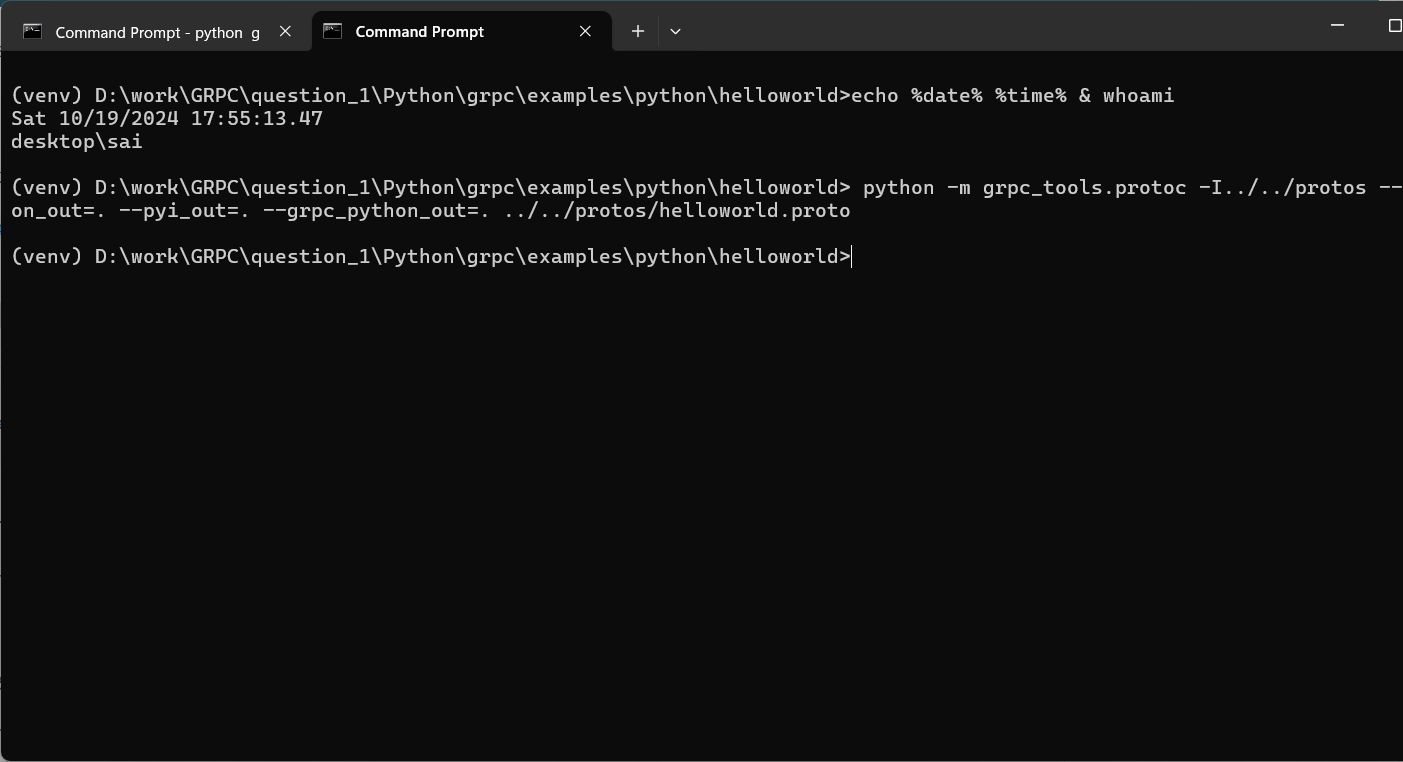
**Step 8: Running the server:** We will then run the sample program that we got from the cloned git repository. The file we will run is a server and the file is called greeter\_server.py.

****

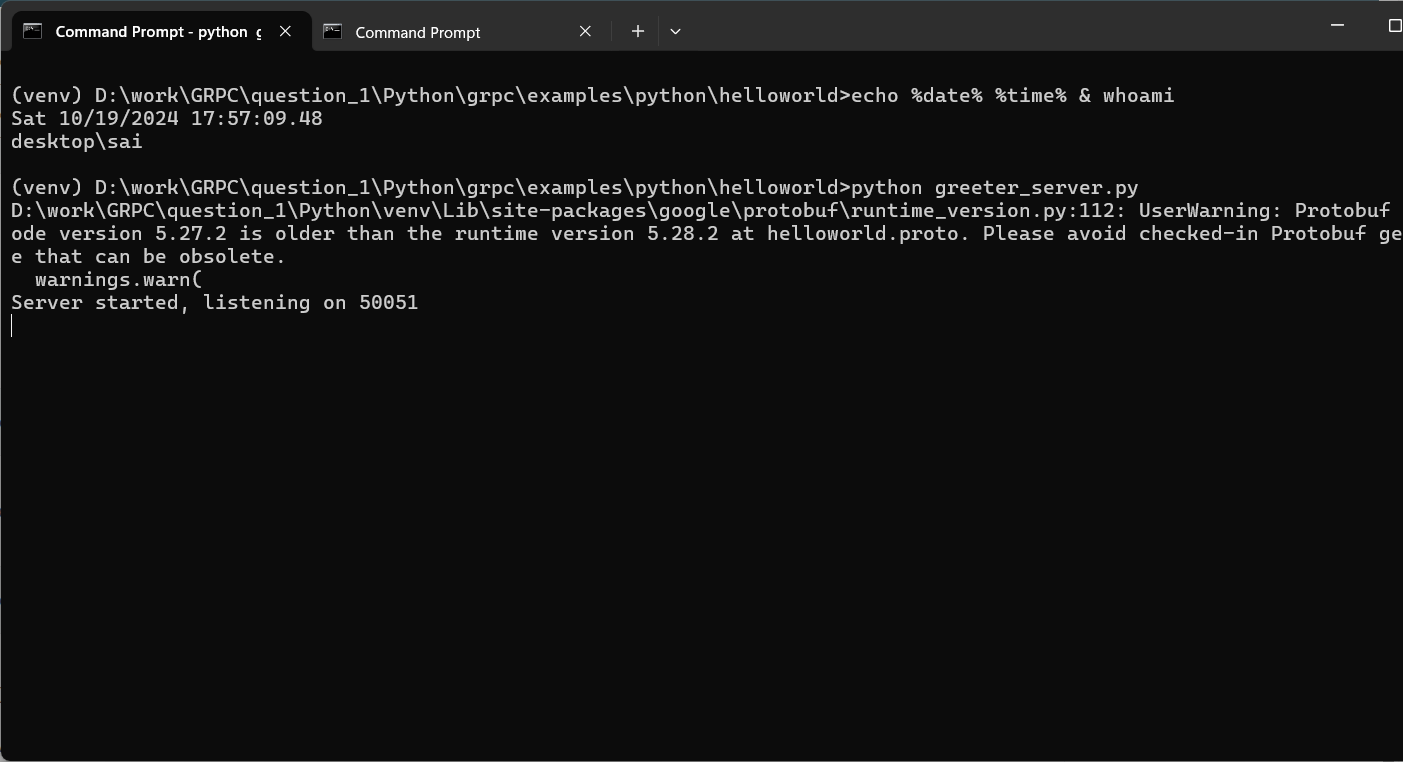
**Step 9: Run the client:** After we get the server running, we will now run the client in a separate virtual environment so that the server and client can communicate with each other. And when the client interacts with the server, the server replies with Greeter Client Received!

****

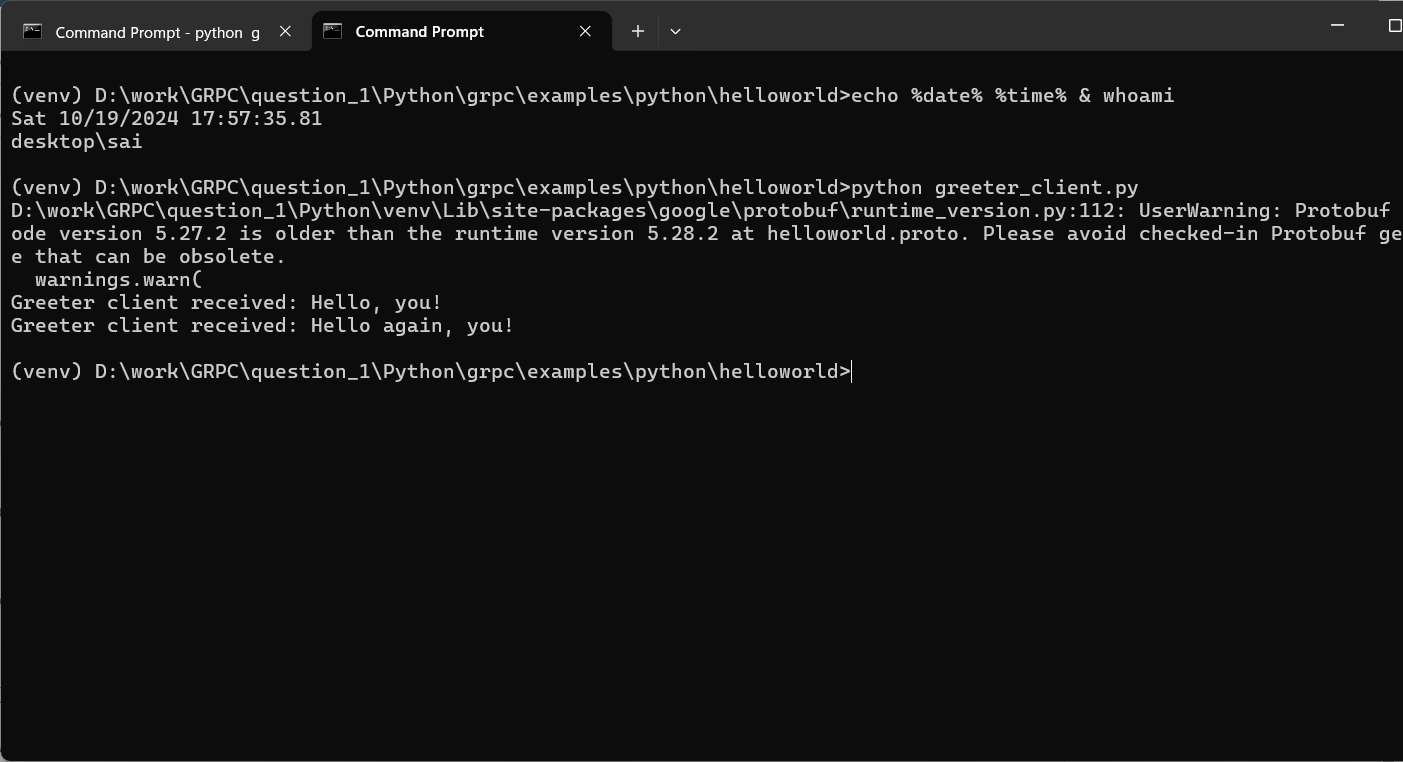
**Step 10: Updating files after change:** We will now update some files and make some changes in the helloworld.proto file.

****

**Step 11: Restarting the server after update:** To make sure the changes reflect in the server, we will now rerun the server.

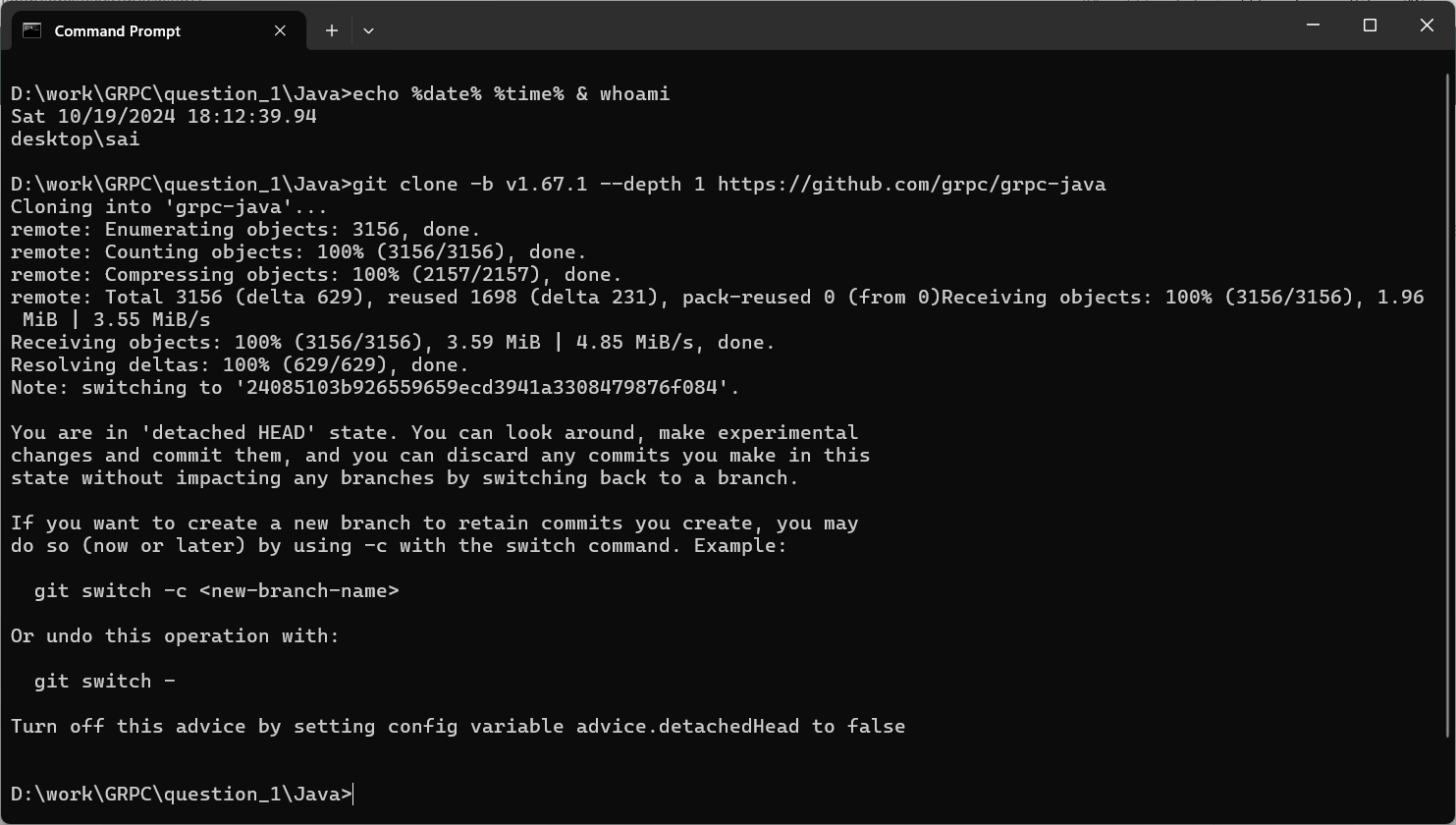
****

**Step 12: Running the client:** After rerunning the server, we will now run the client again and when the client runs and communicates with the server the message now received is changed into Hello, you!

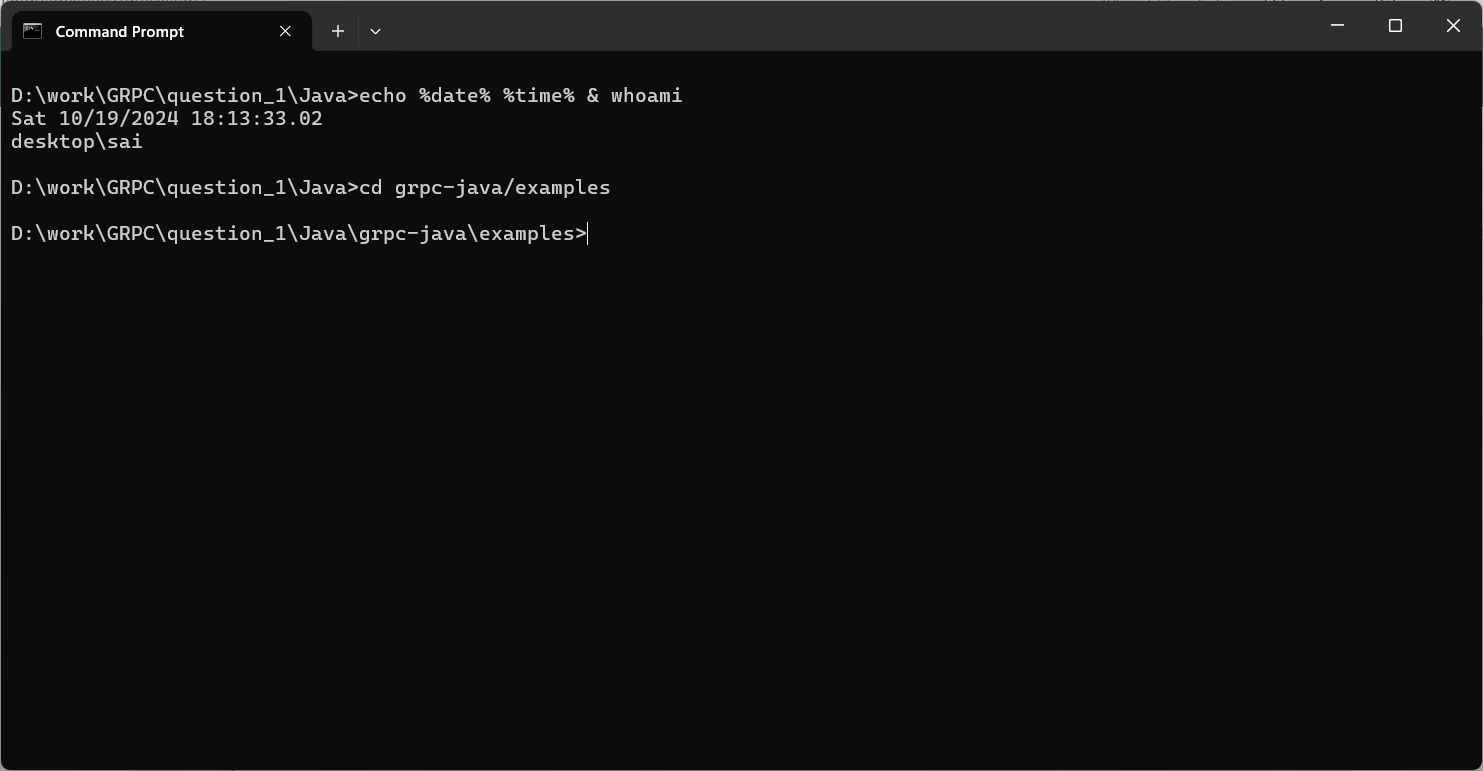
****

**Java:** Now we will be performing all above actions using the gRPC but with JAVA language now!

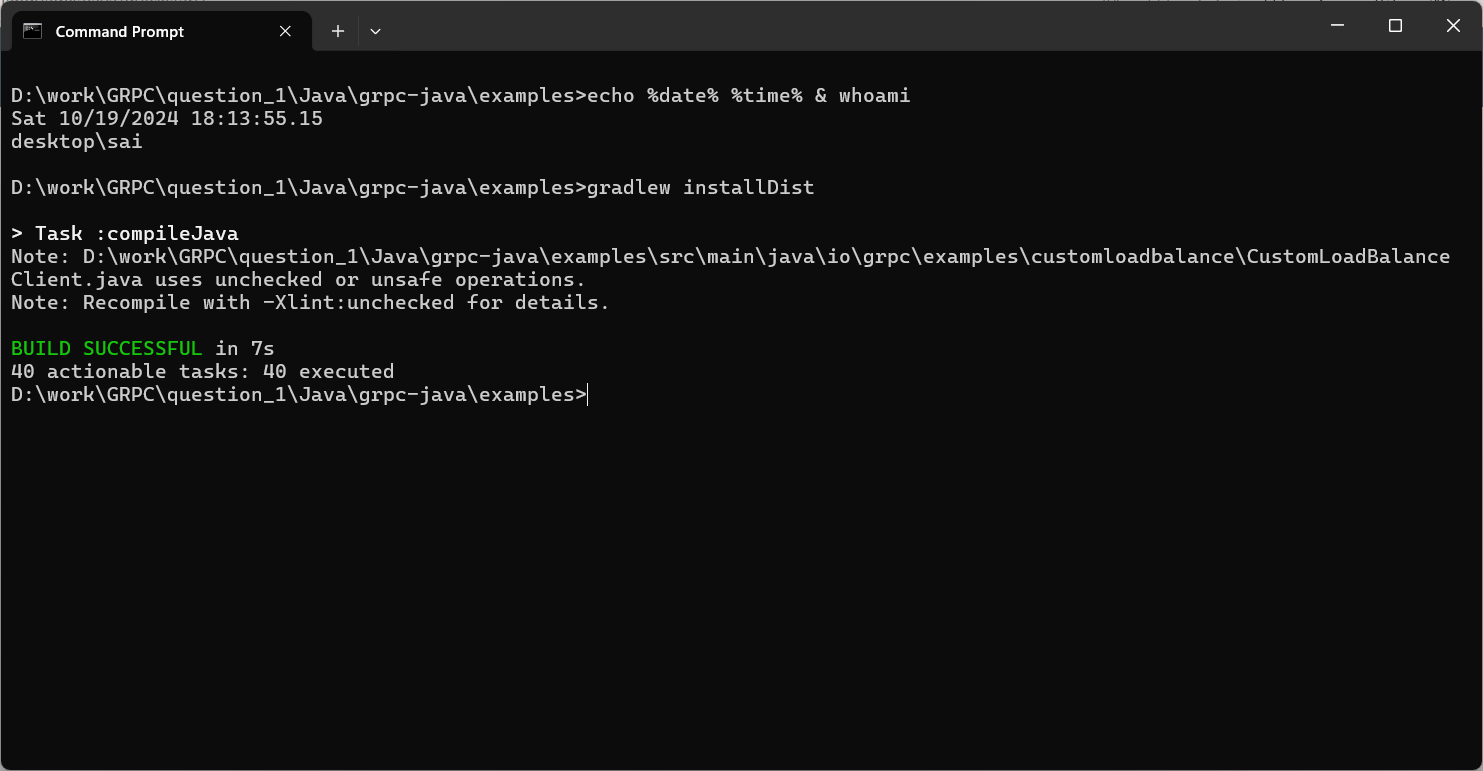
**Step 1: Cloning git repository:** We will first download java related gRPC code form the github using the git command.

****

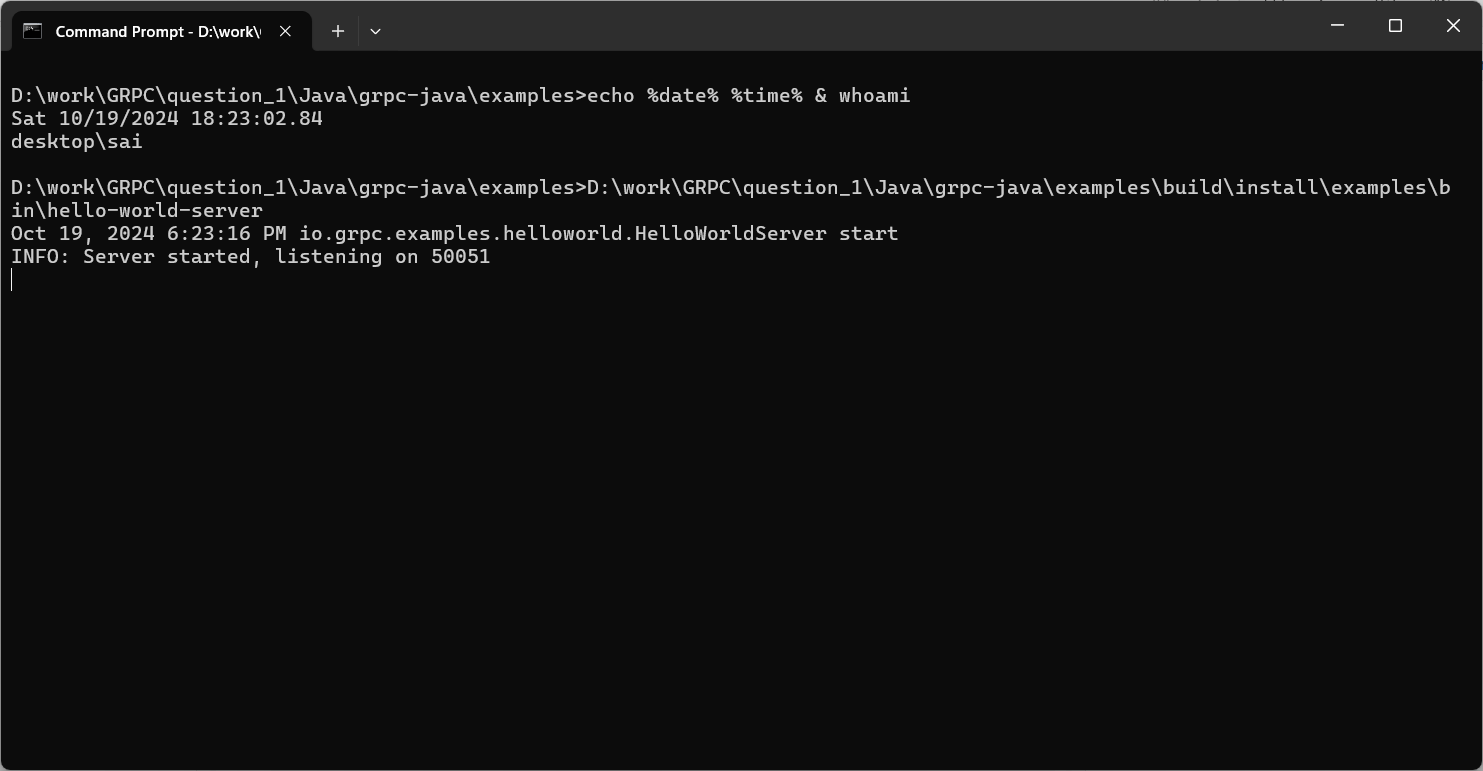
**Step 2: Navigate to grpc-java\examples:** We will now navigate to where the server and client are located but this time we will move to grpc-java directory.

****

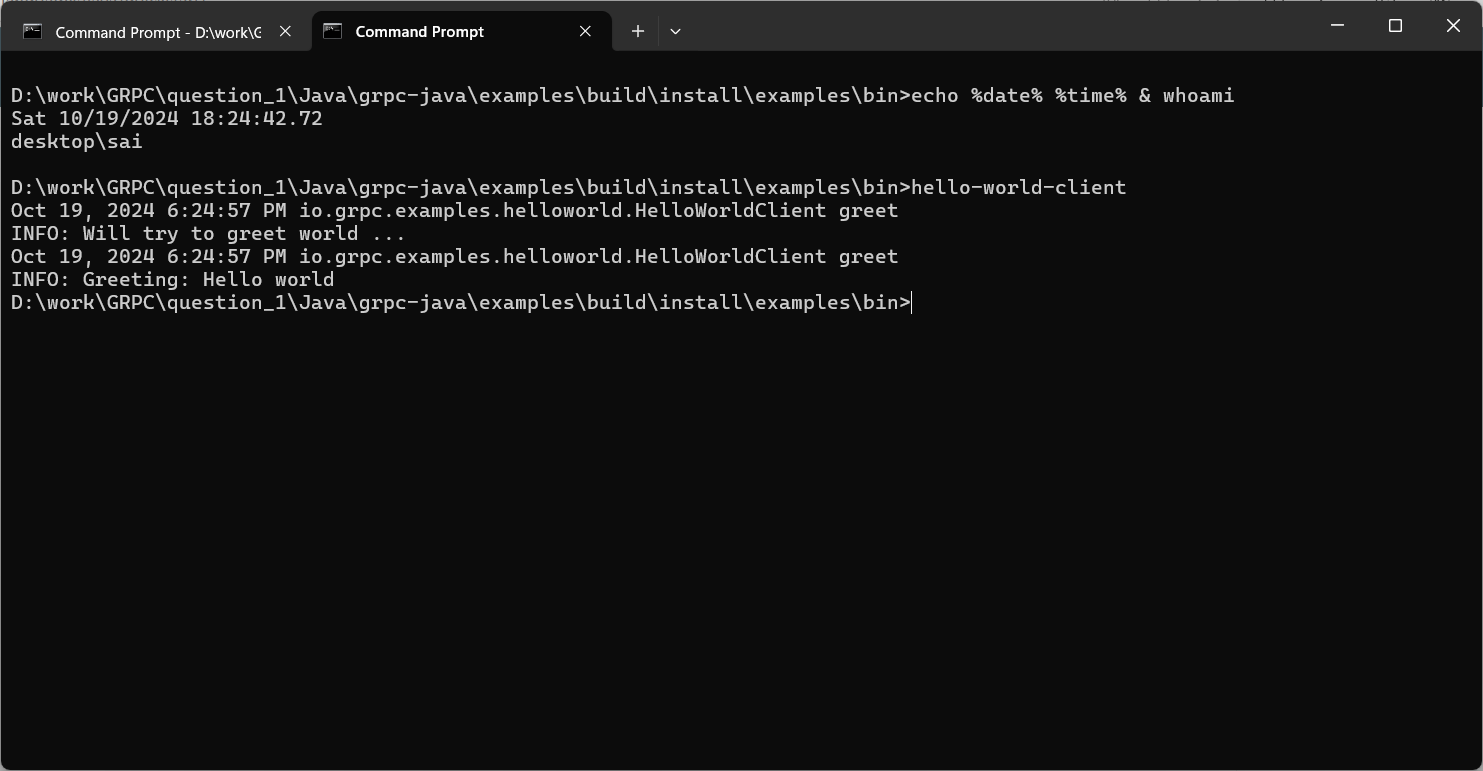
**Step 3: Compiling the client and server:** In java we are required to compile the code first, unlike in python, we just run the code. We will first compile the client and server code.

****

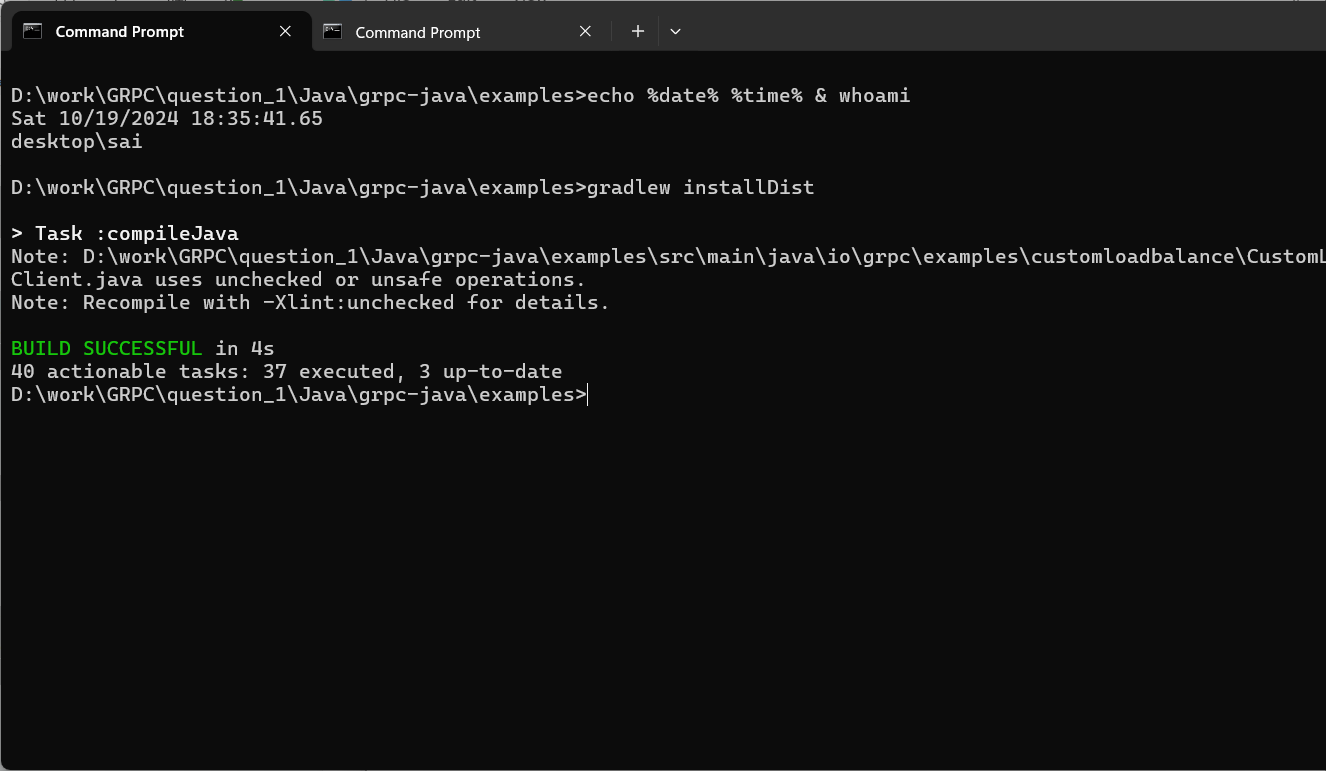
**Step 4: Running the server:** We will now run the server after it is compiled. And the server will start listening on the 50051 port.

****

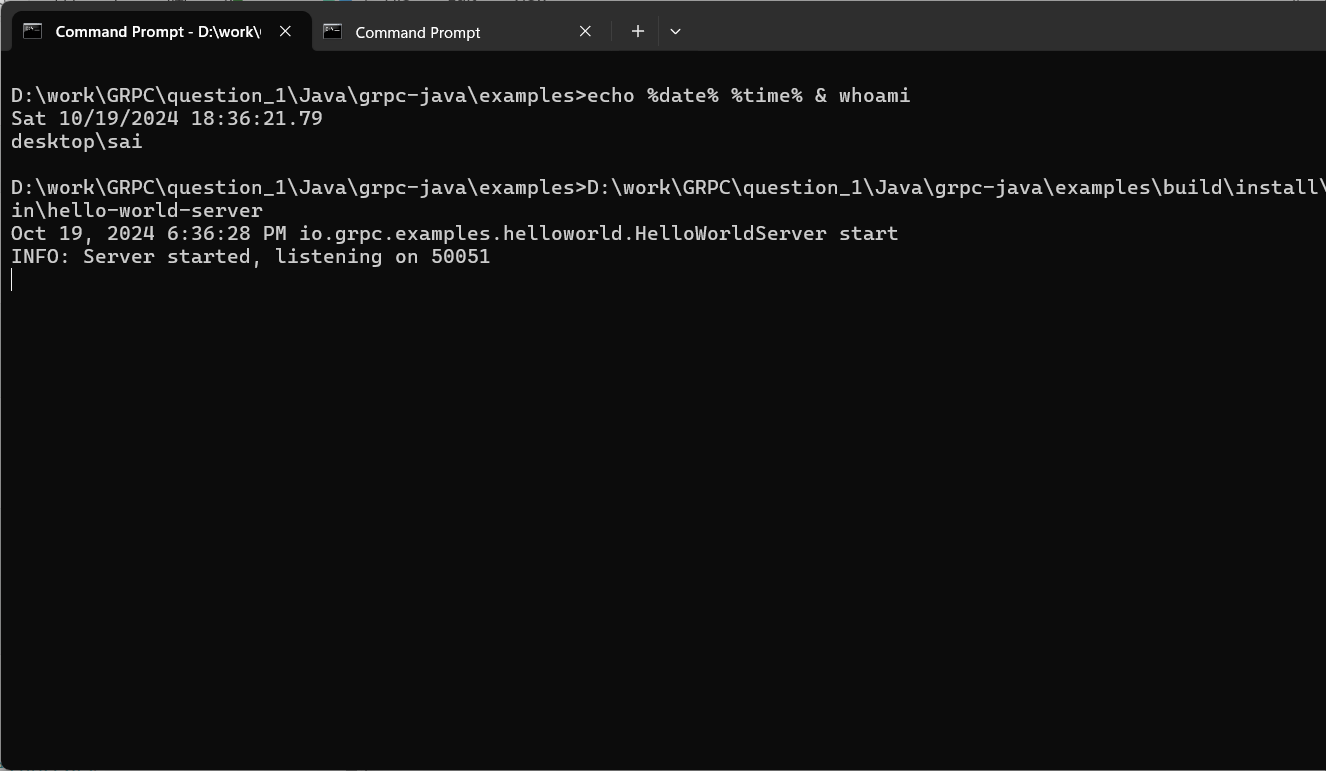
**Step 5: Running the client:** Similar to python, we will now run the client in the JAVA code as well, and when it is run, the server greets the client saying Hello world.

****

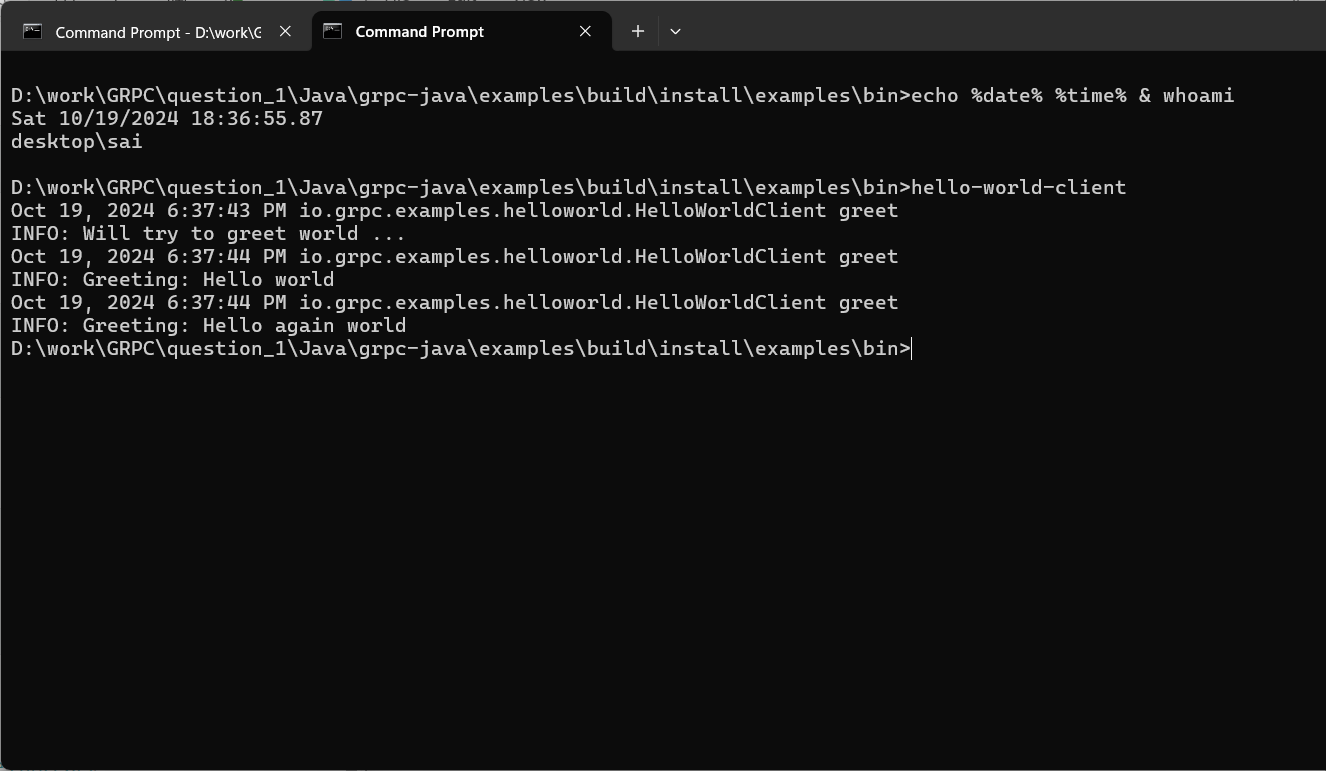
**Step 6: Build after the change in files:** We are then required to make some changes and then build the code again.

****

**Step 7: Running the server after the change:** To make sure that our changes are reflected in the server, we will now run the server again after building/compiling the code.

****

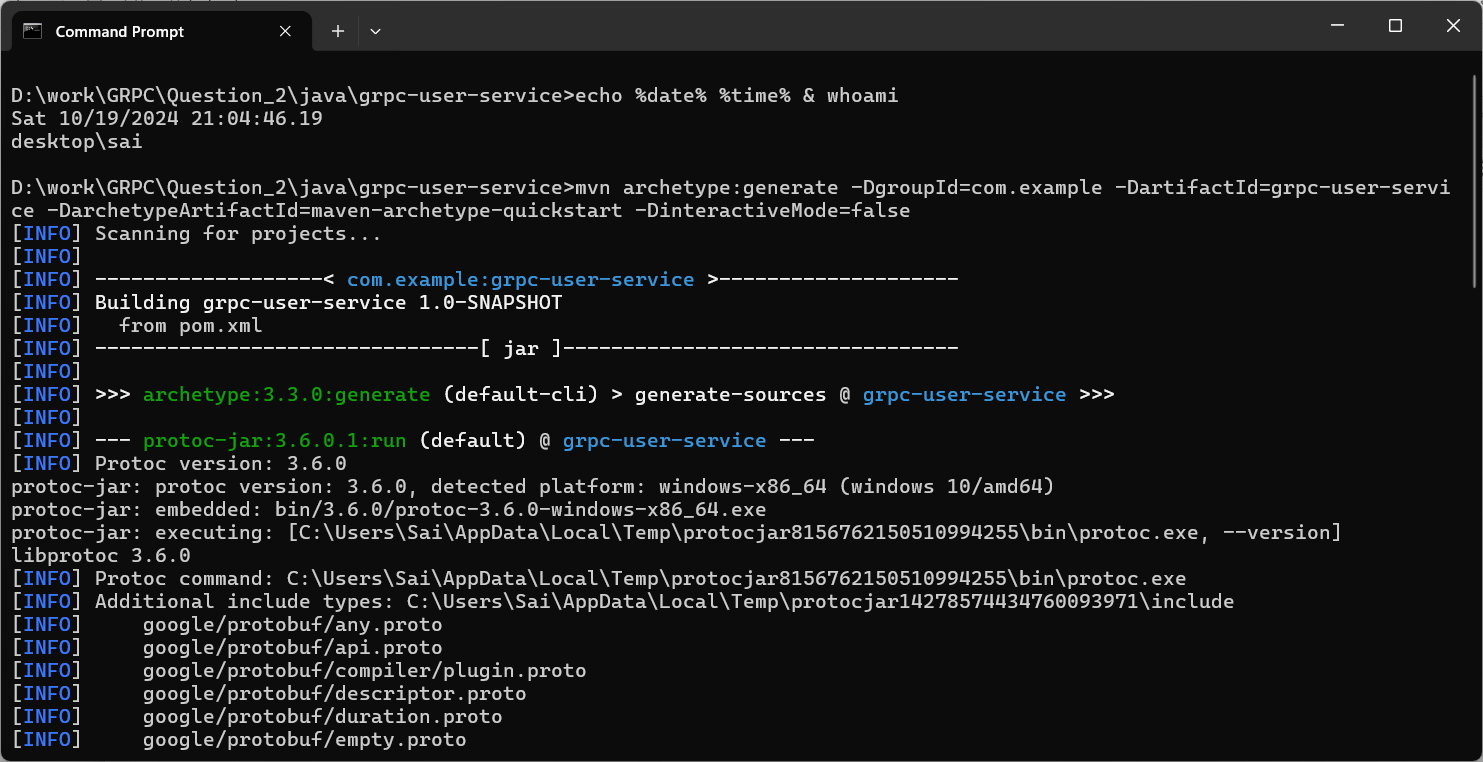
**Step 8: Receiving the server response on client side:** After the server is up and running, we will run the client again and now watch the changes we made in effect. The server reply now is different from before!

****

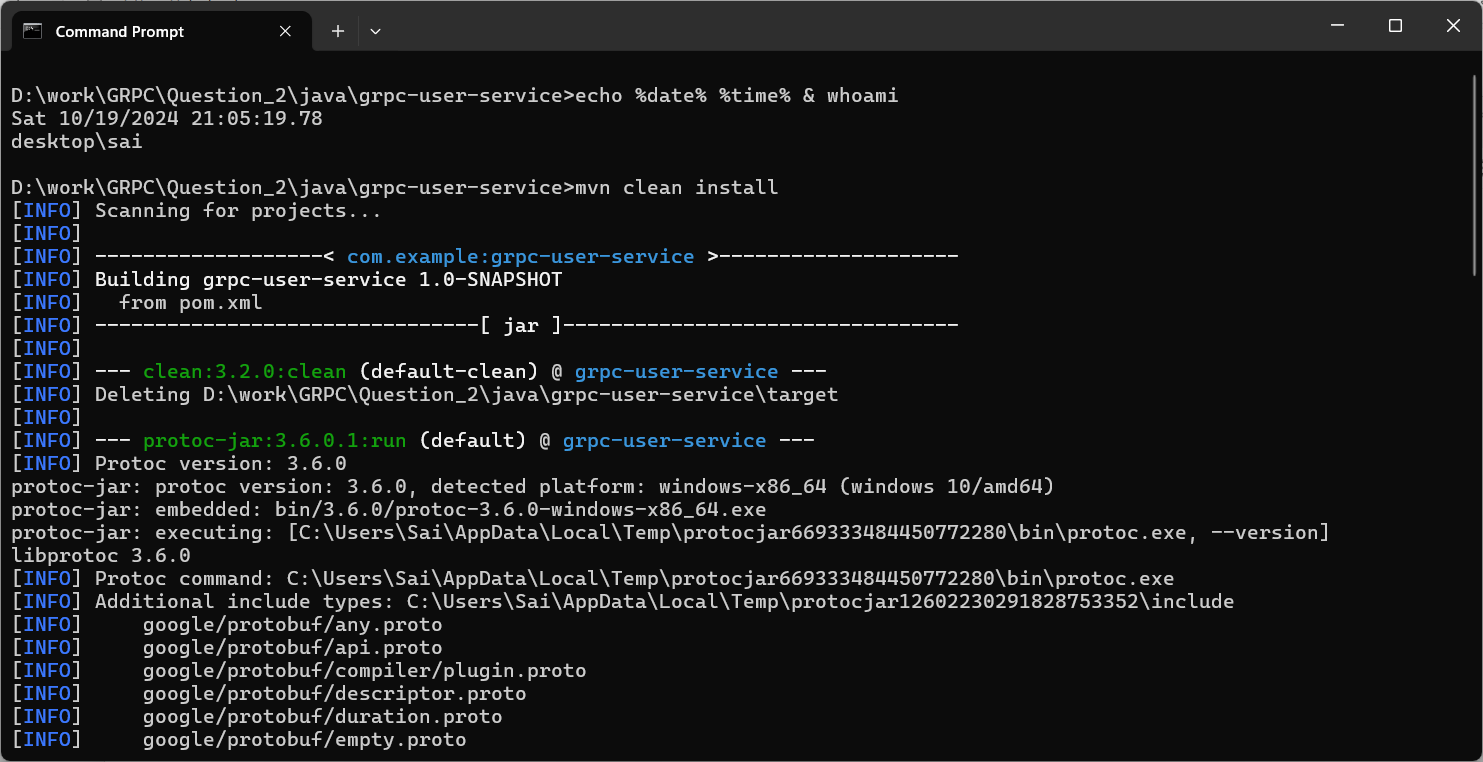
**Question 2:**

**Java:** We will first setup and run the project in JAVA language and then move onto python language.

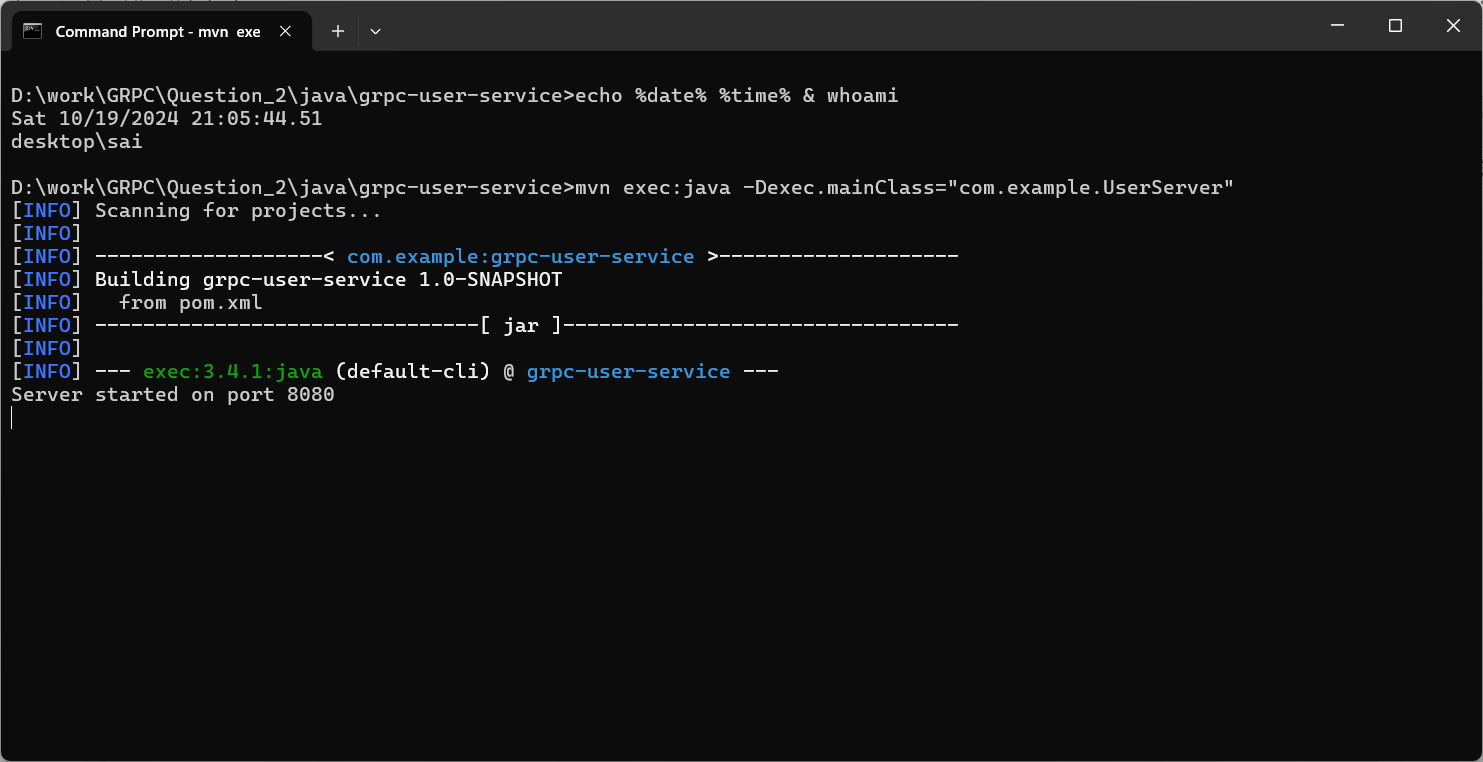
**Step 1: Create project:** We will first create a new project can build grpc-user-service project.



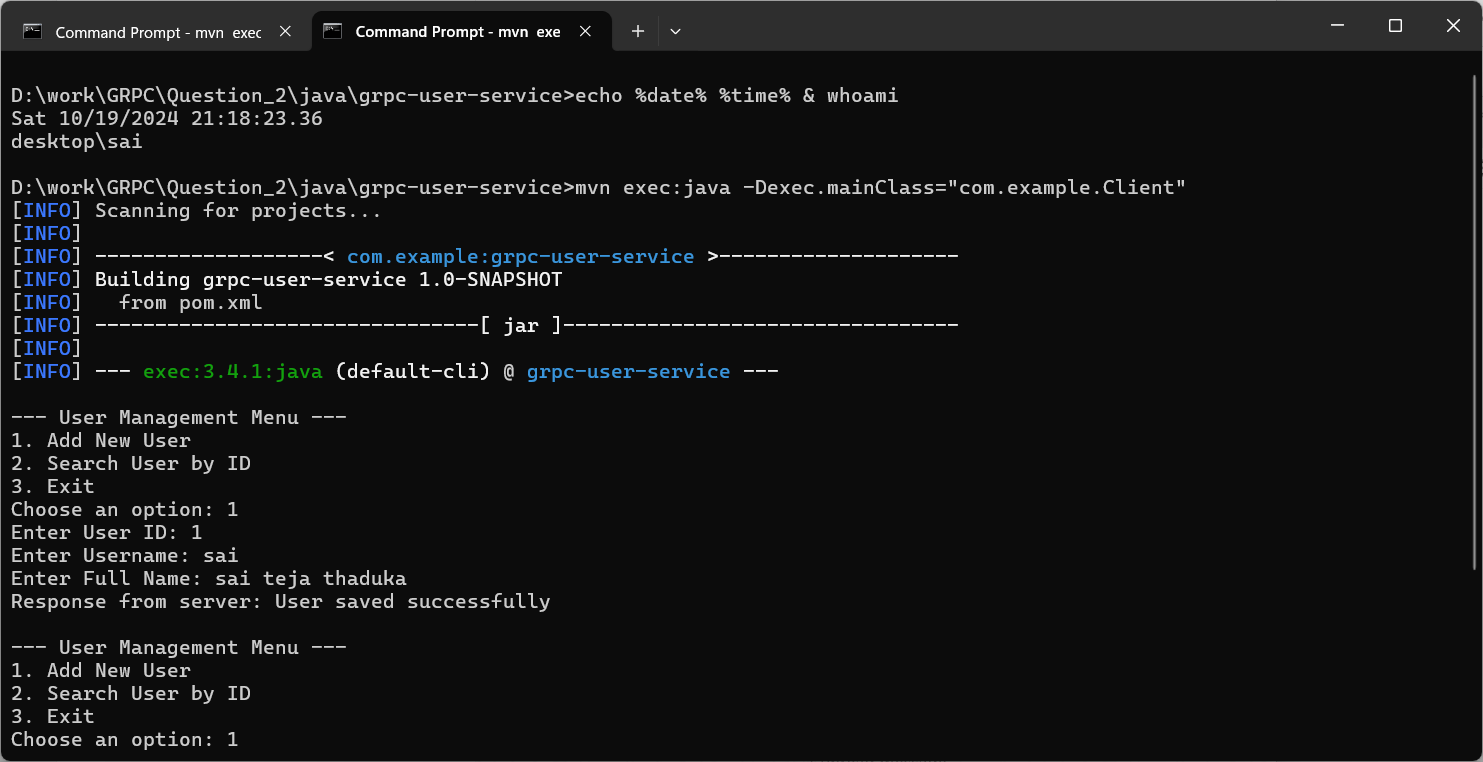
**Step 2: Build project:** We will then build the project as done in JAVA language.

****

**Step 3: Start server:** After building the project, we will now start the server.

****

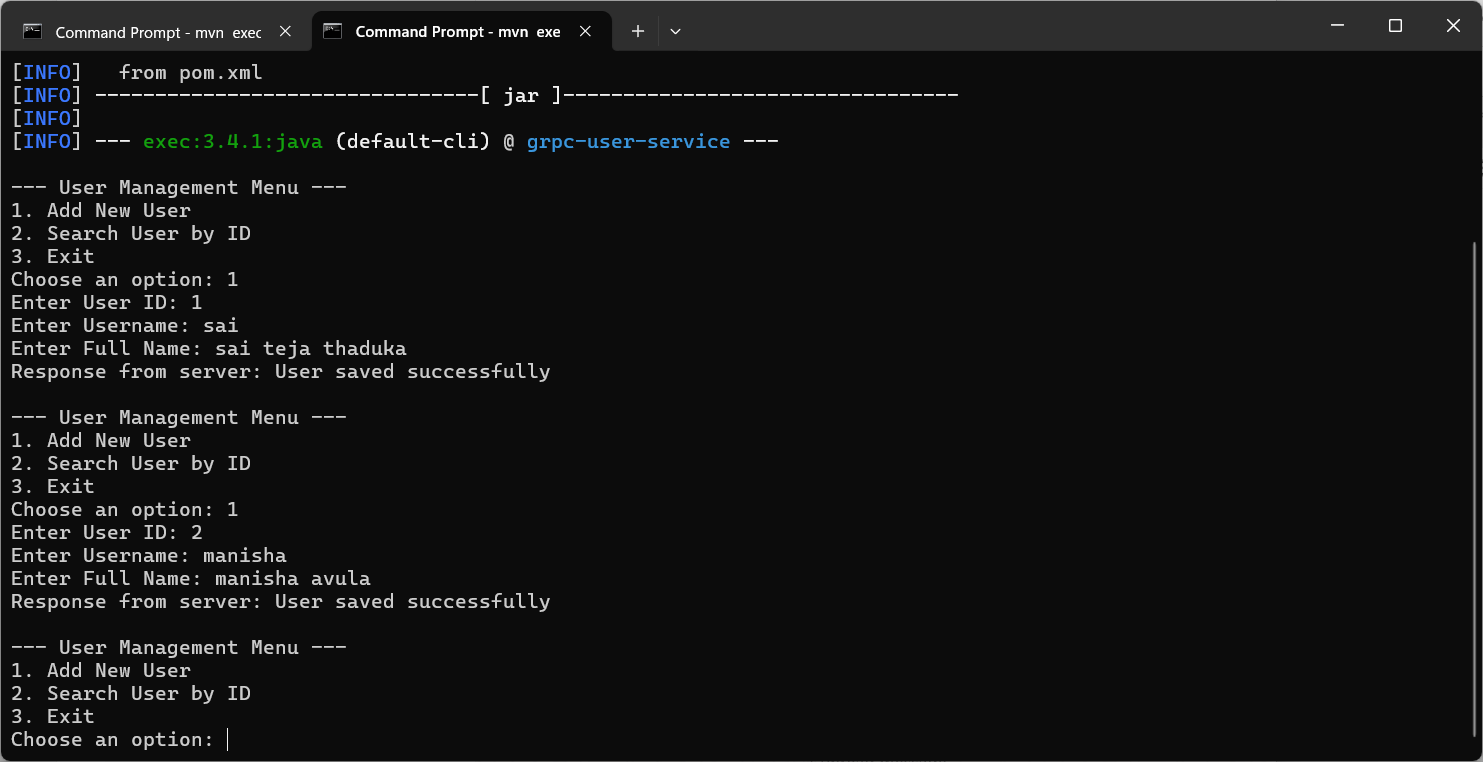
**Step 4: Run client and add input:** We will now run client the client and connect with the server and while connected to the server we will try to input commands to the server and perform various operations!

****

The operations we are performing are:

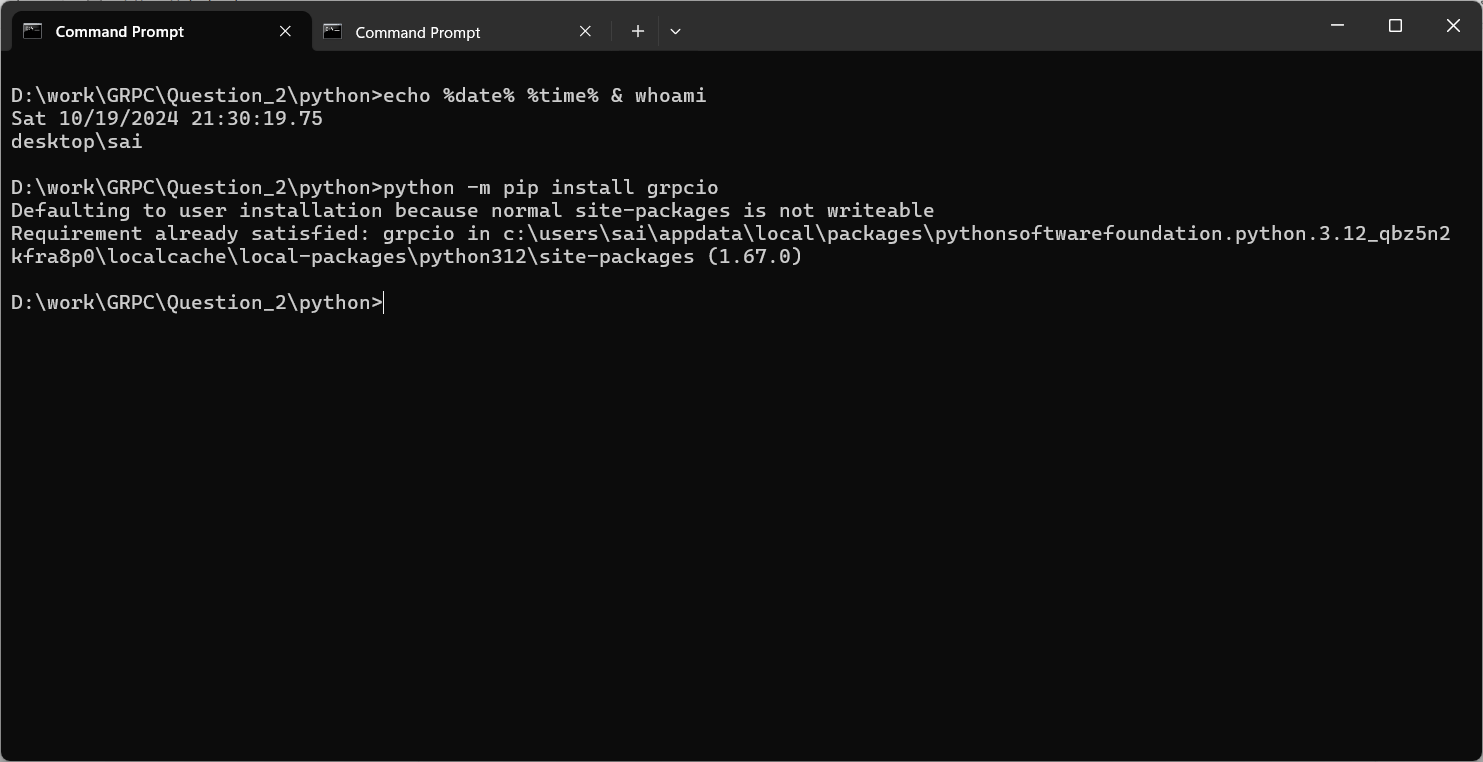
1. Creating users
2. Searching users
3. And Exiting the project (i.e., closing connection with the server)

And all of this is achieved by using gRPC.

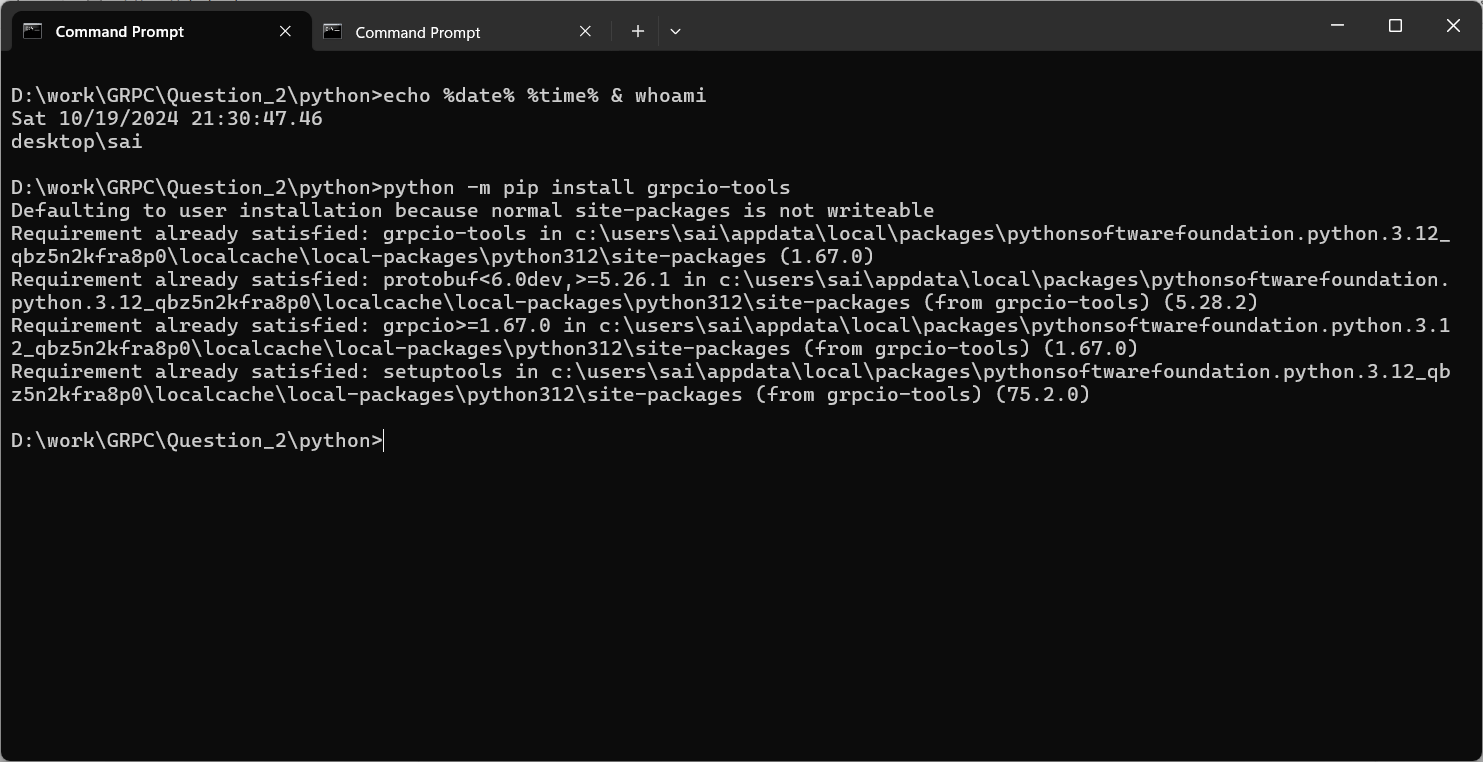
****

**Python:** Now we will be performing the same functions but using the python language.

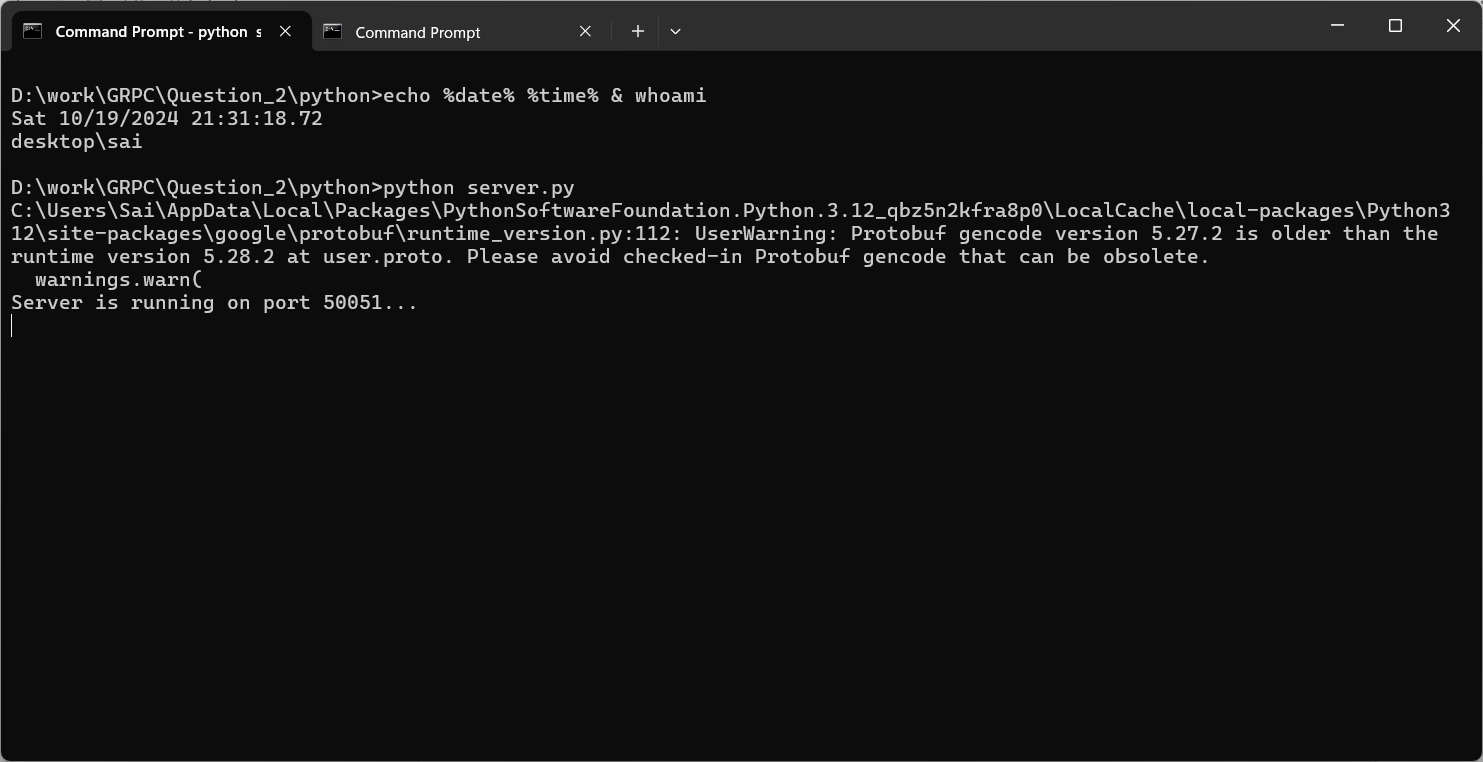
**Step 1: Install grpcio:** We will now install grpcio library by using pip command.

****

**Step 2: Install grpcio-tools:** Then we will install the grpcio-tools using the same pip command (Matching the version of grpcio-tools is really important).

****

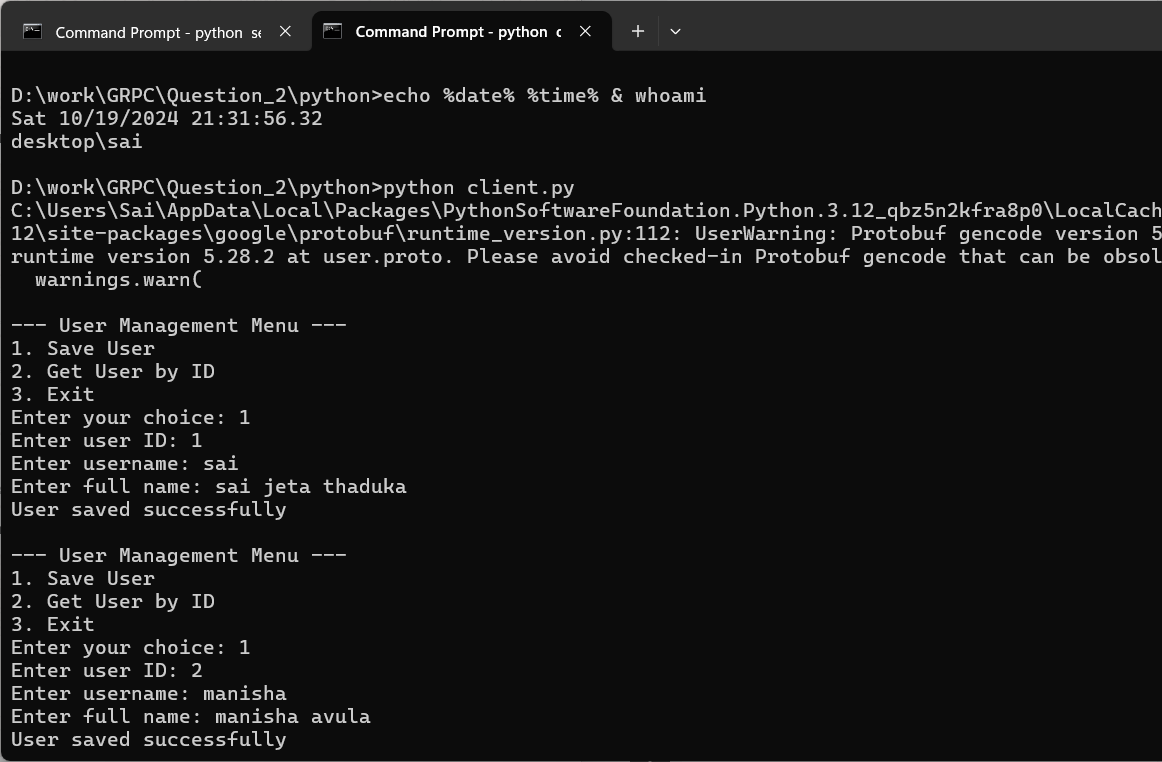
**Step 3: Run server:** After all the necessary setup done, we will now run the server and get it ready to serve the clients.

****

**Step 4: Run client:** We then run the client and make it connect to the server. And here we can also see all the functions that are provided by the server and they are:

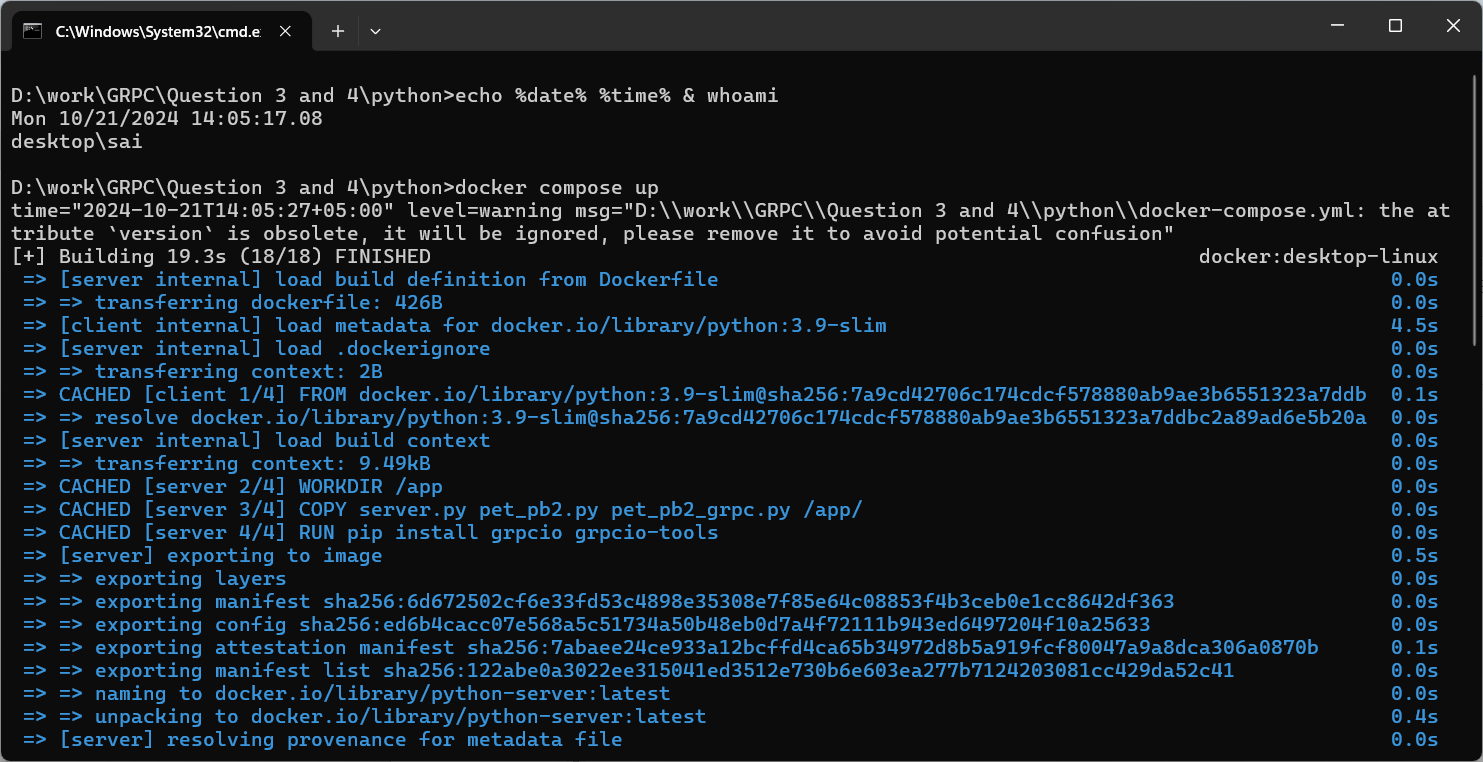
1. Adding User
2. Searching User
3. Exiting the program (i.e., closing the connection).

The third option can be used to close the connection with the server when all the work is done.



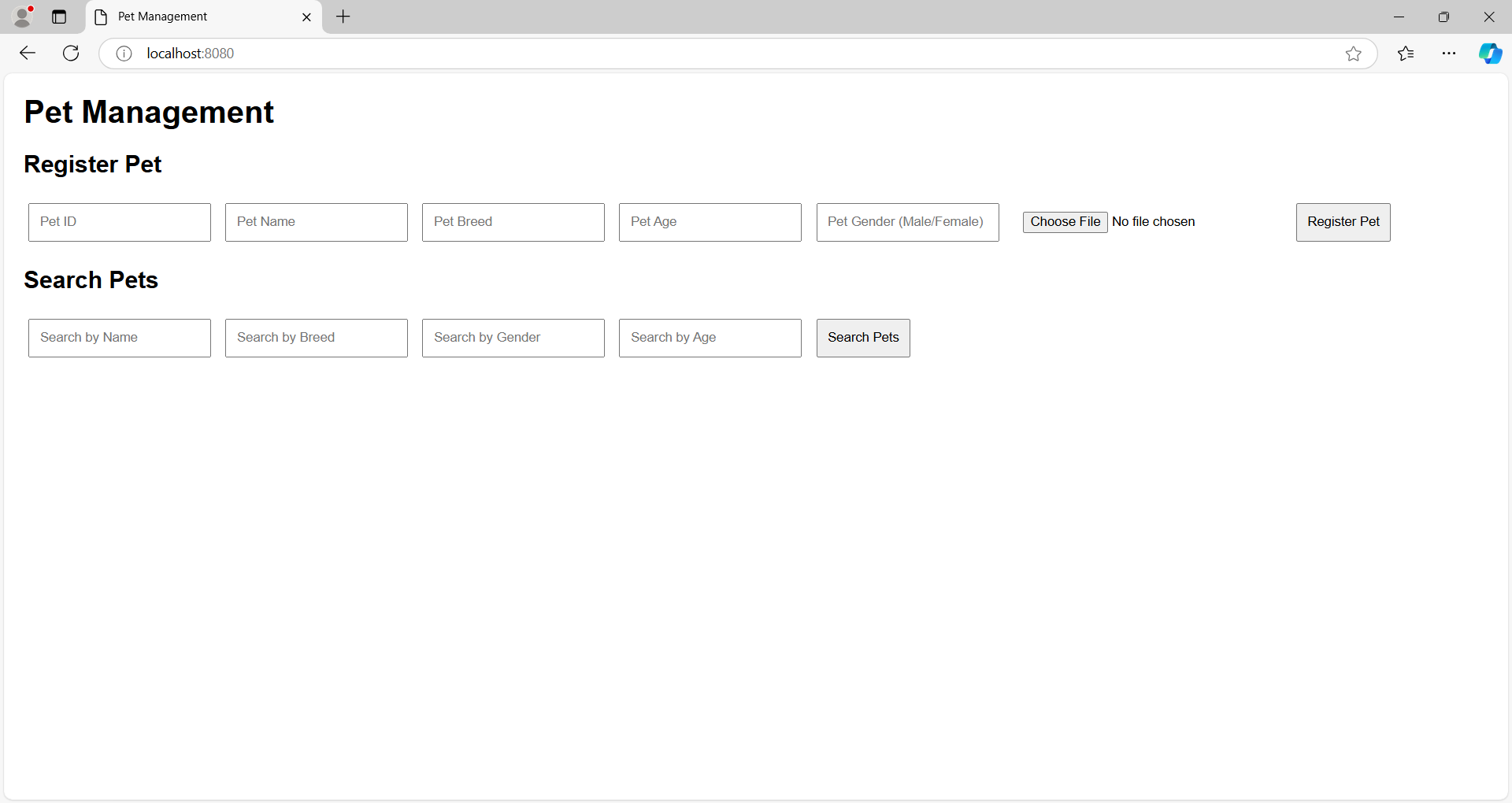
**Question 3:**

**Creating docker images:** first of all we set up the docker images by running the command ‘docker compose up’. With this our images will be ready and the server will be up and running.



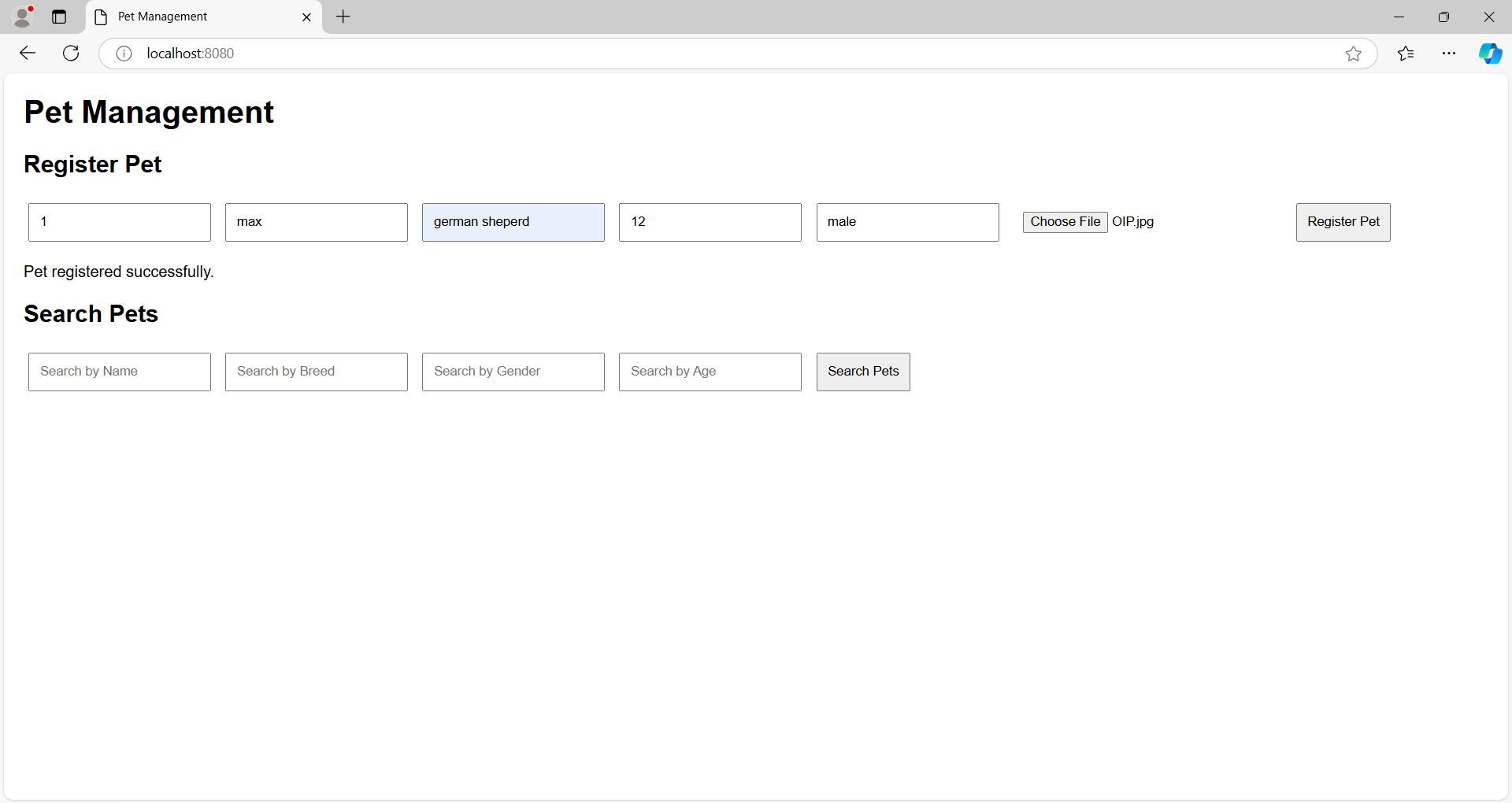
**Visiting localhost 8080:**

After the images are set the server running, when we visit the localhost 8080, we will see the following page open up.



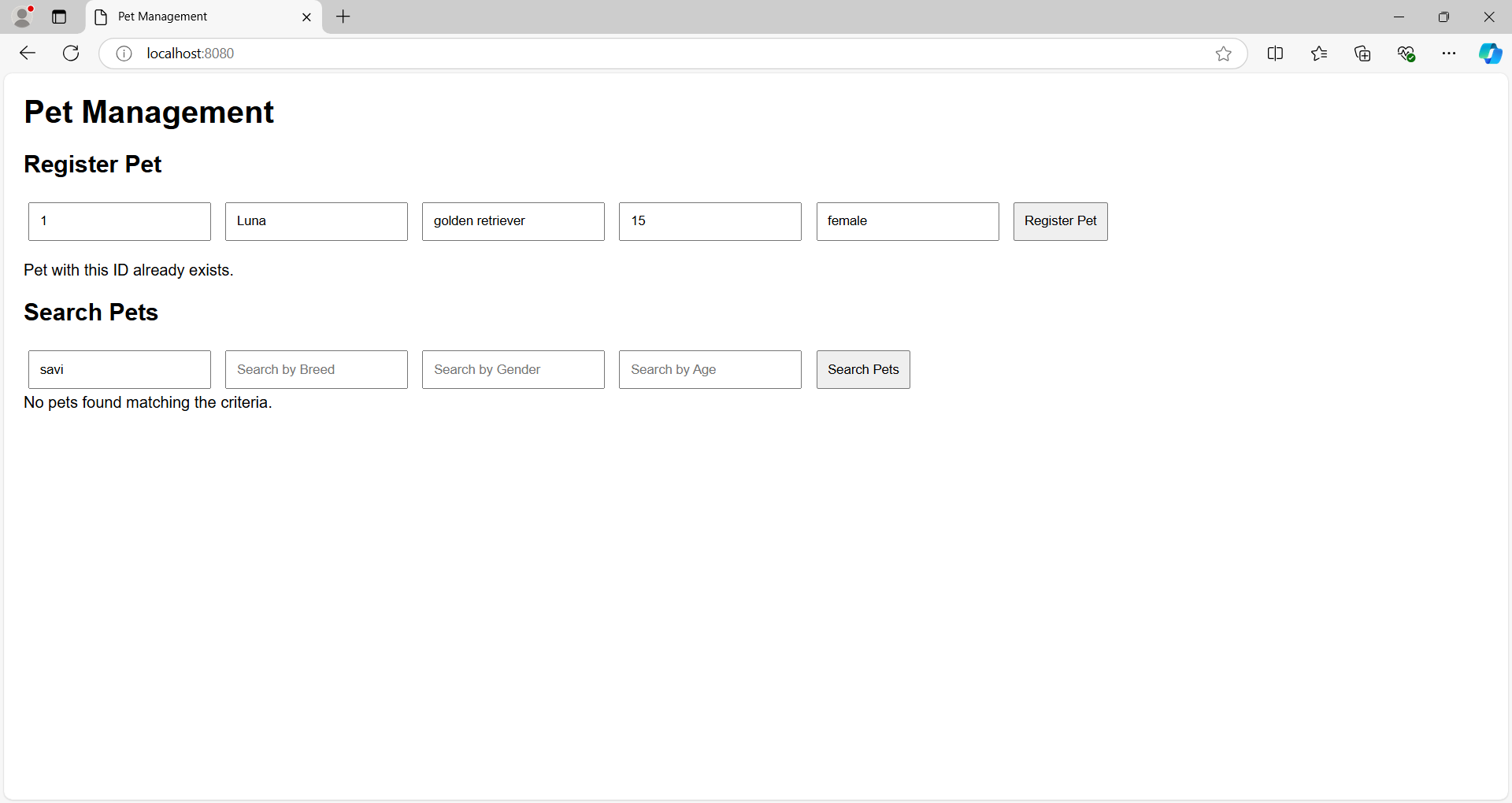
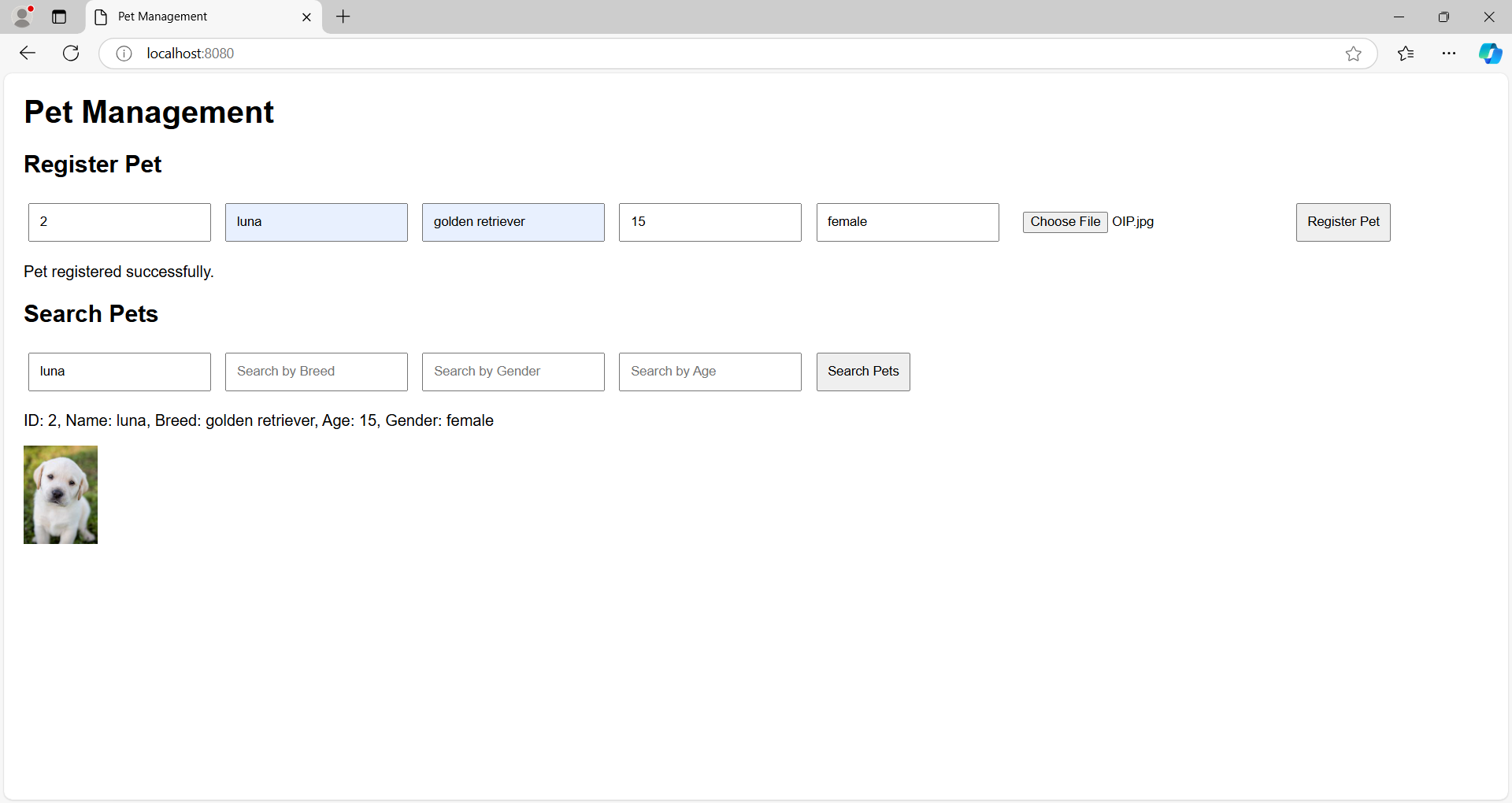
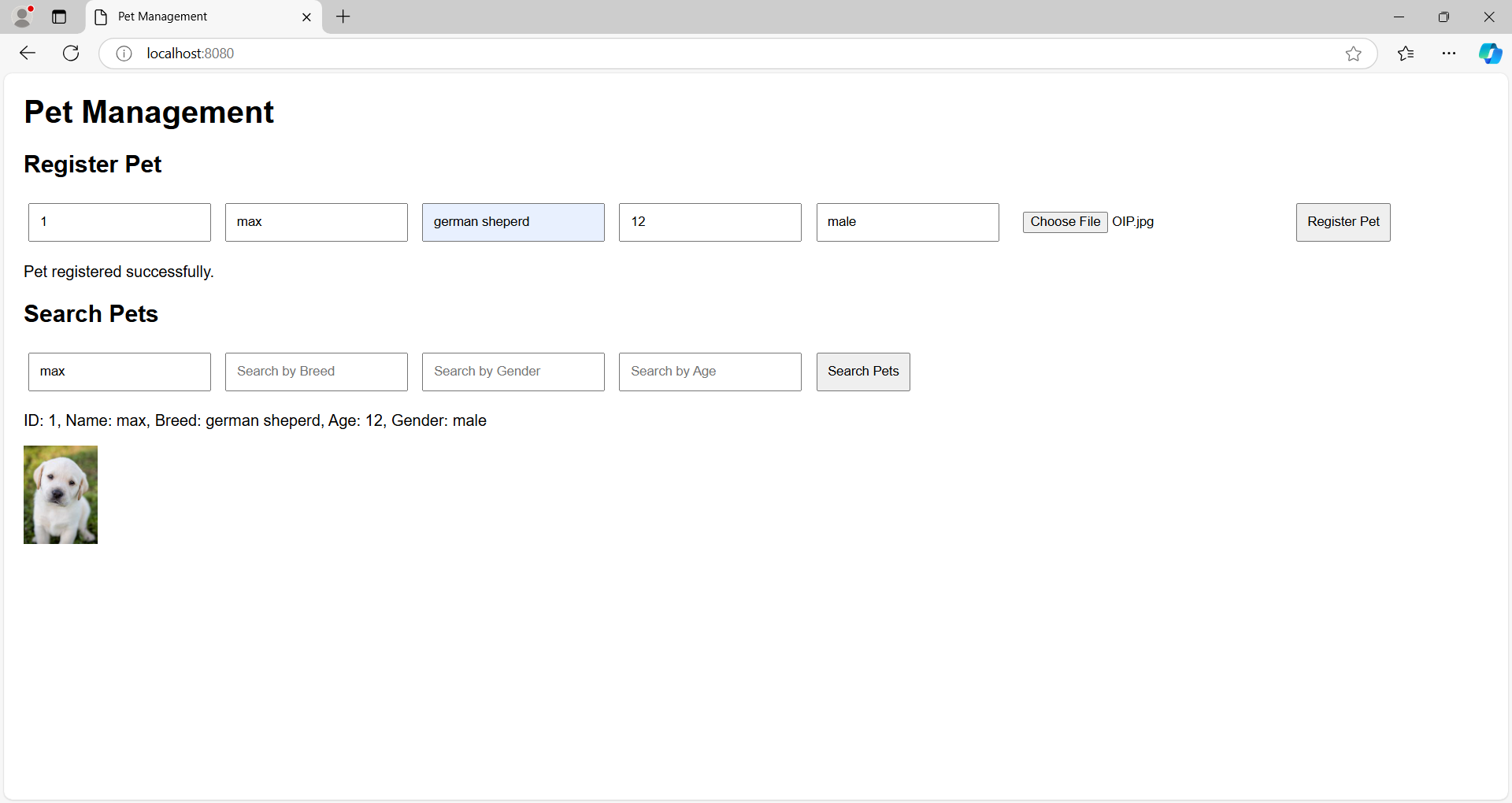
**Register a pet:**

Now to register a pet we input some data in the fields and click the register pet button. When the pet is successfully registered, a message will be displayed as shown in the image below.

****

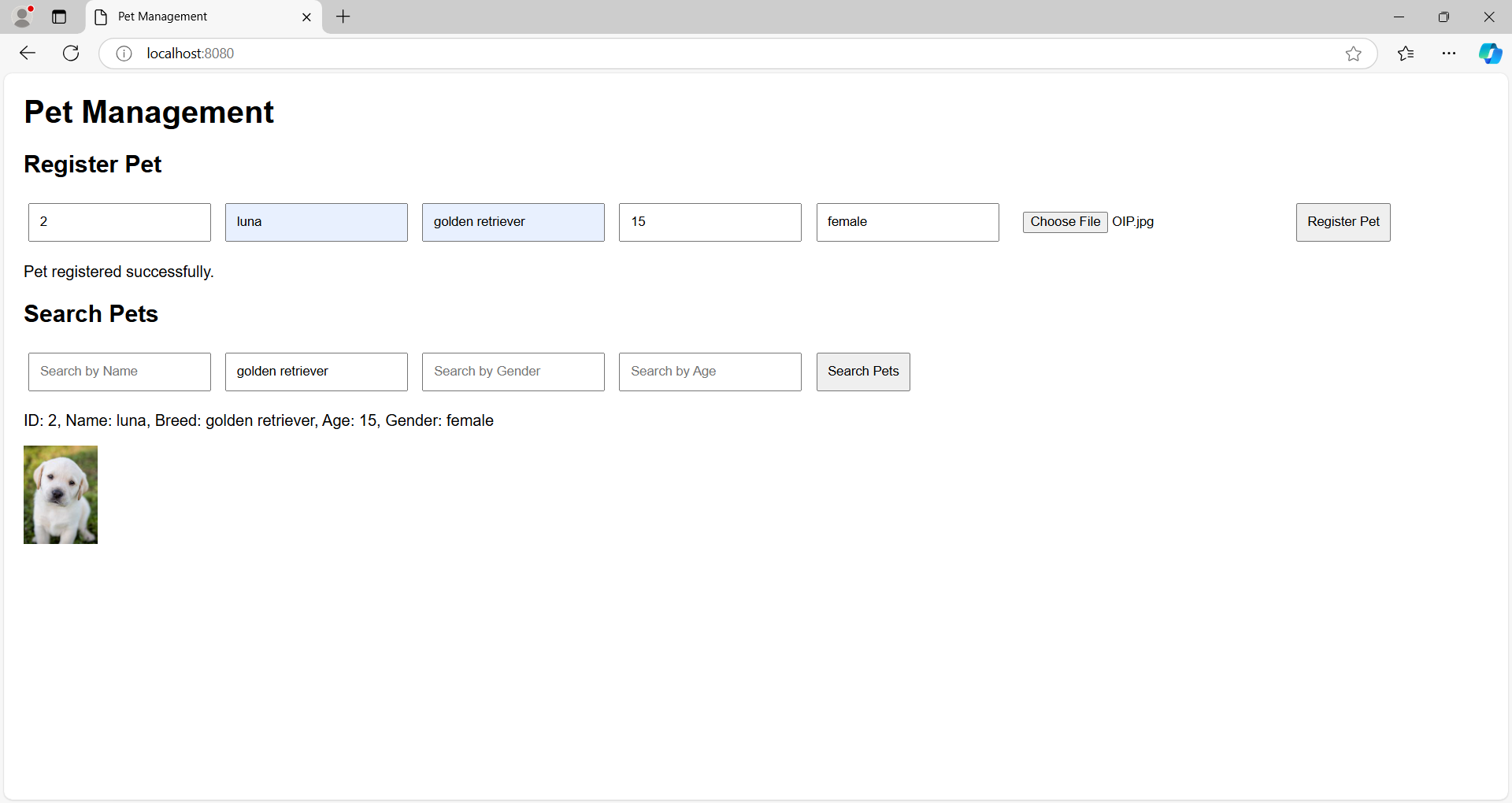
**Searching a pet by its name:**

After successfully registering a pet we search the pet by its name, if a pet was present the system will return its ID, name, breed, Age, gender and image of the pet. And if there was no pet by the name the system will display No pets found matching the criteria.



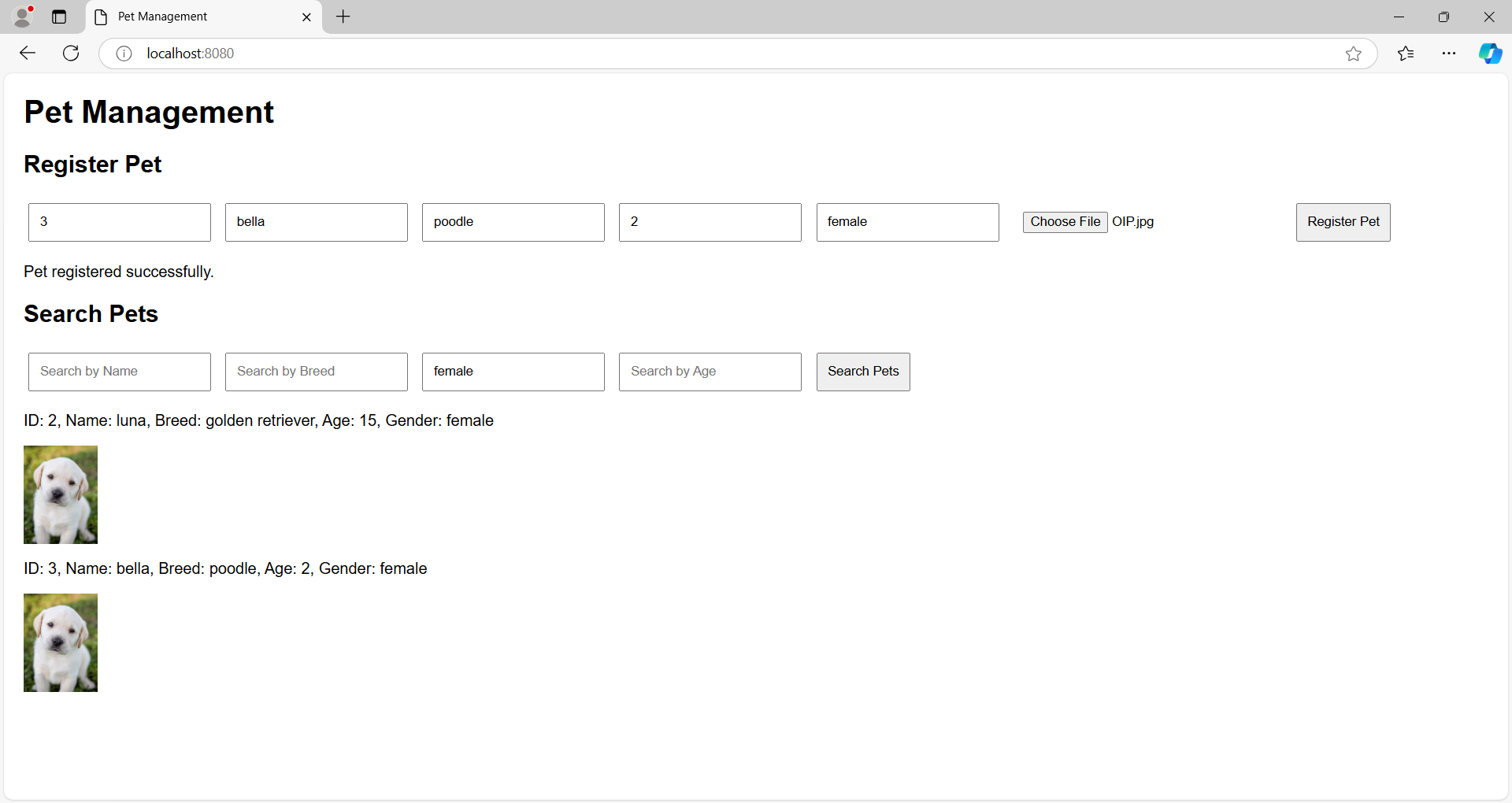
**Searching pet by breed:**

After registering many pets we can also search pets by their breed. In the breed section input the breed of the pet to search for the pets having that breed. The system will return all the pets with that breed.



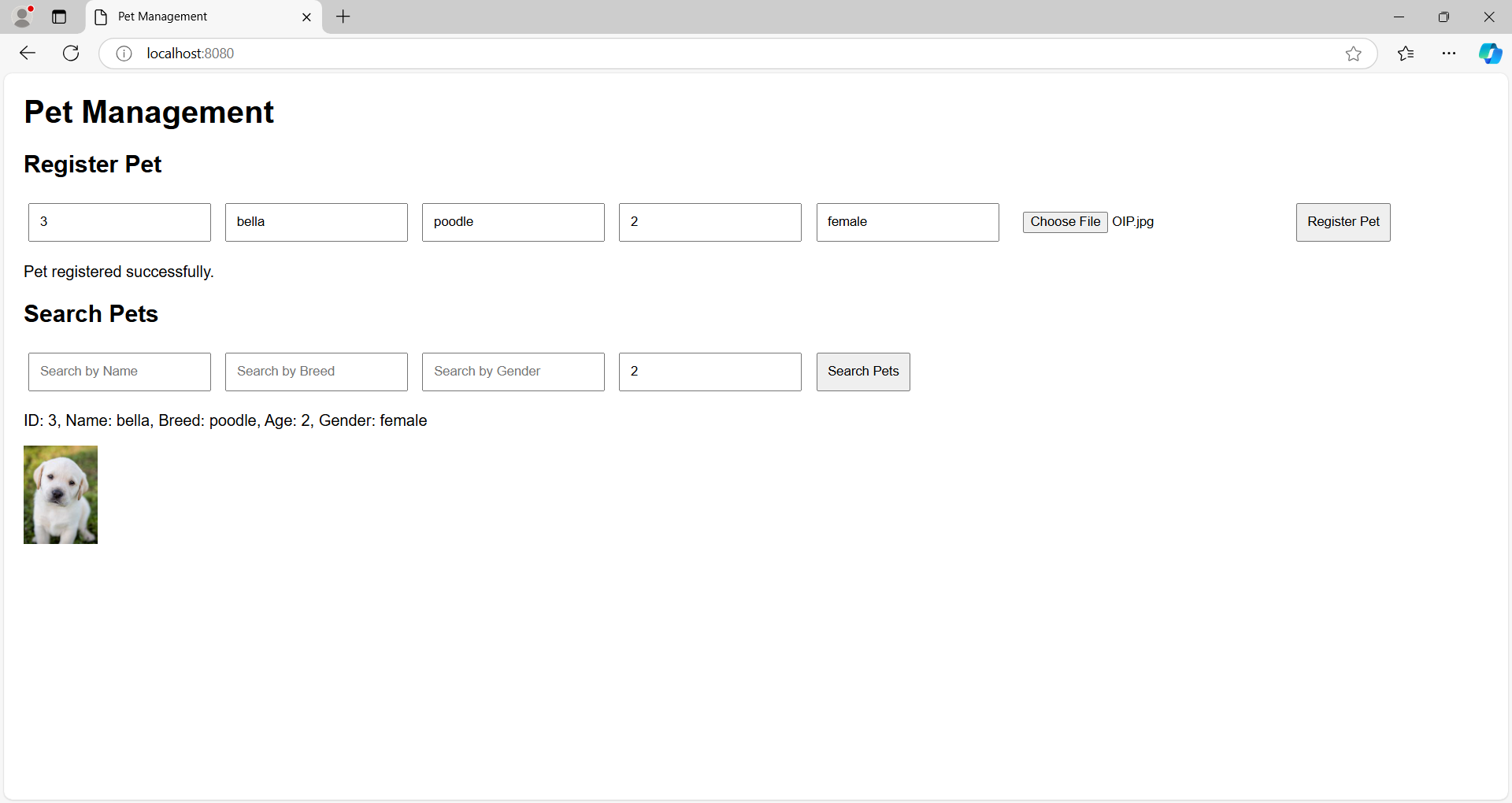
**Searching pet by its gender:**

We can search a pet by its gender, when we input the gender of a pet all the pets having the same gender as the input will be returned by the system.



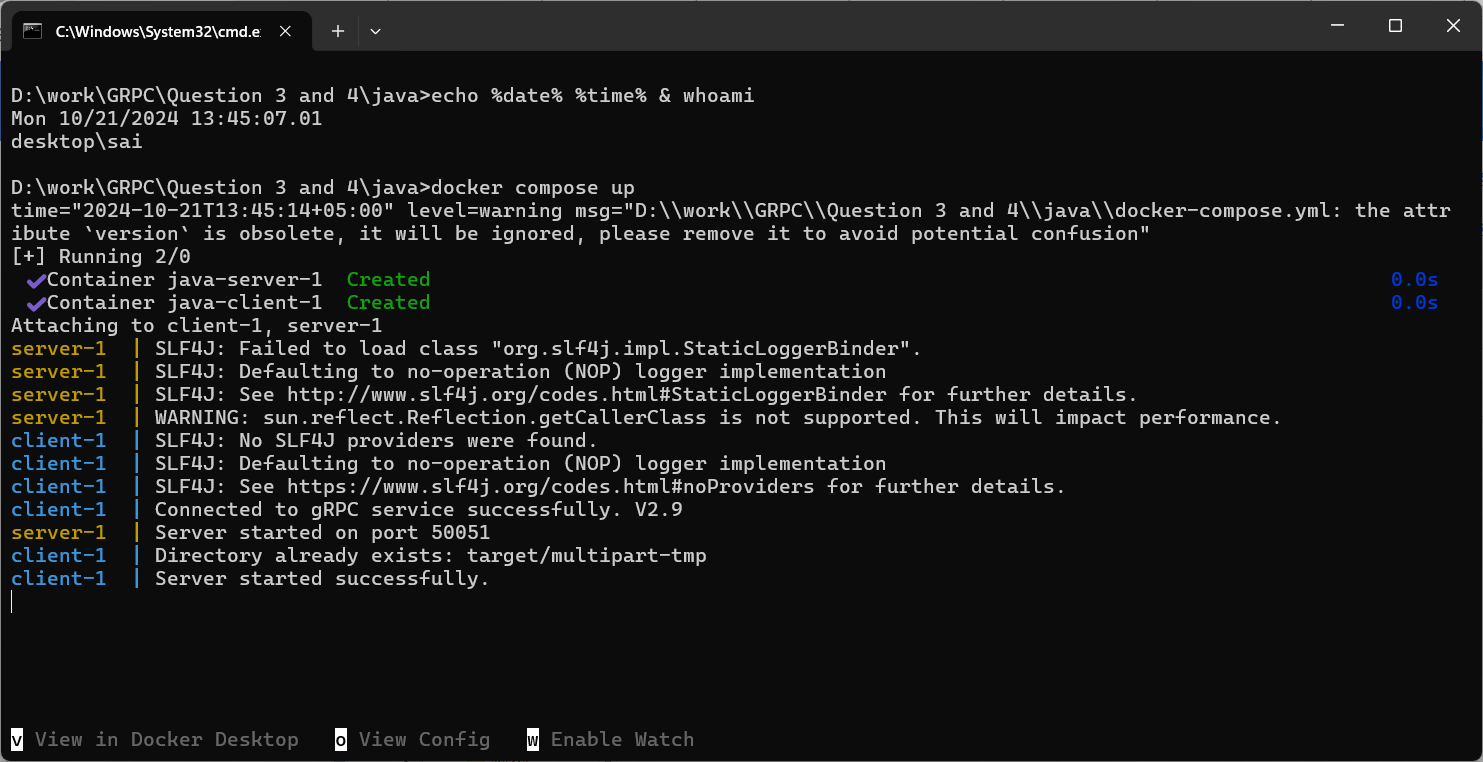
**Search pet by age:**

We can search pets by their age with the search by age field, giving the input in the age and searching for the pet will return us the data of the pets by their ages.

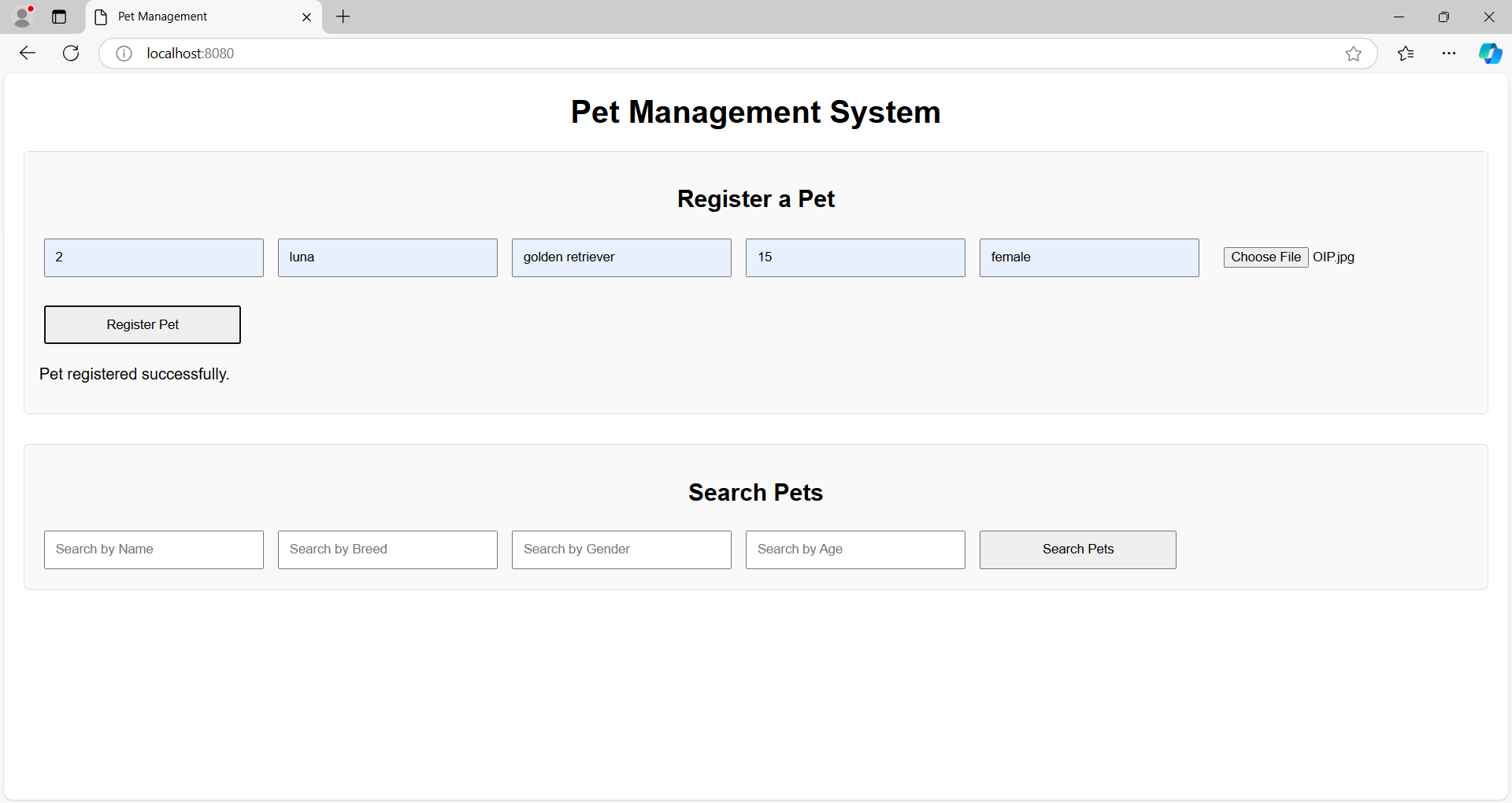


**Java:**

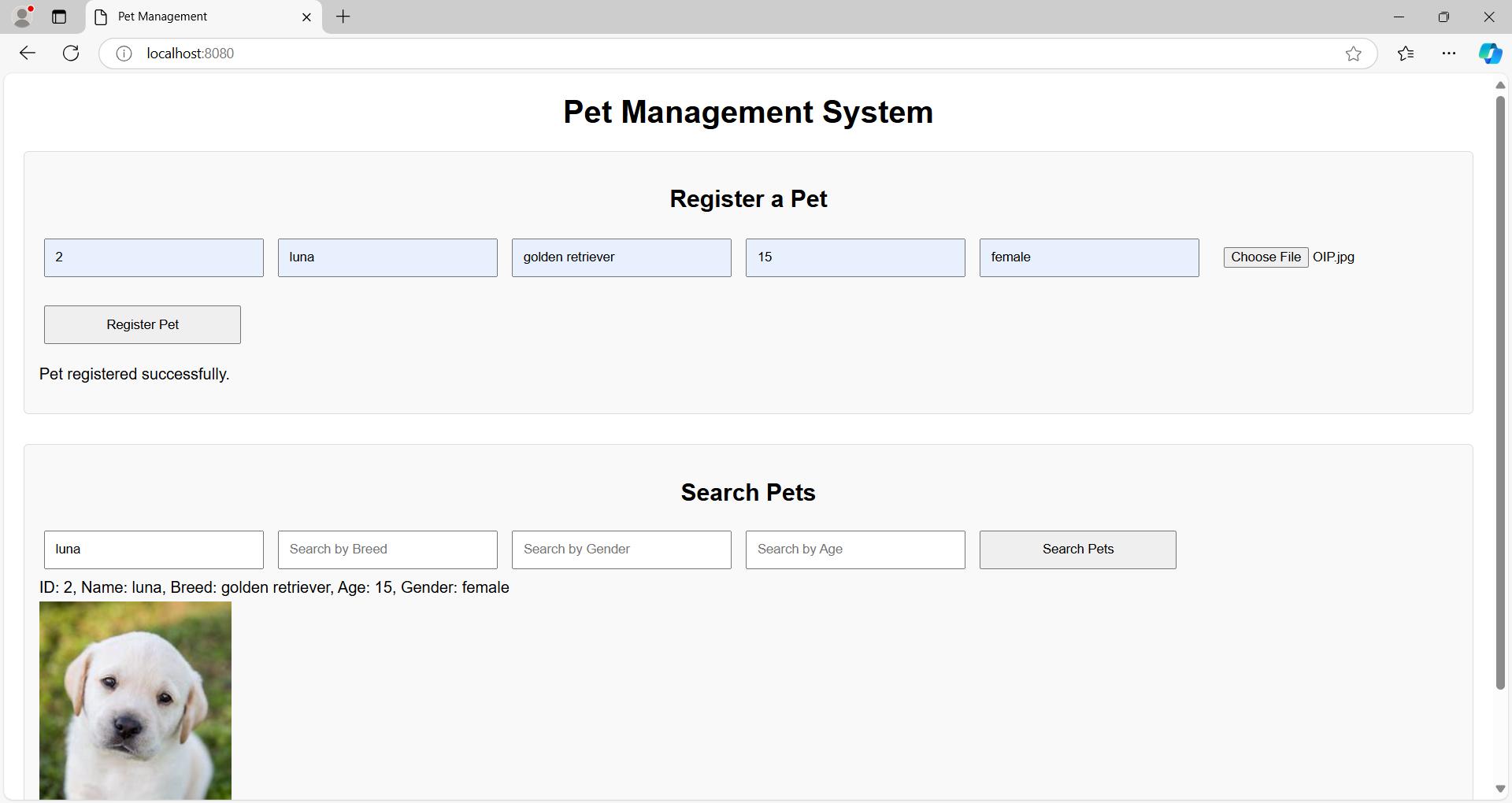
For java we have used maven to set up our project. The docker compose command sets up the docker images and container. When we run the ‘docker compose up’ the system starts the server.



Now that server has successfully started we can visit local host 8080 to view our page. The following screen is displayed where we can input the data of the pet and save it to perform different operations.

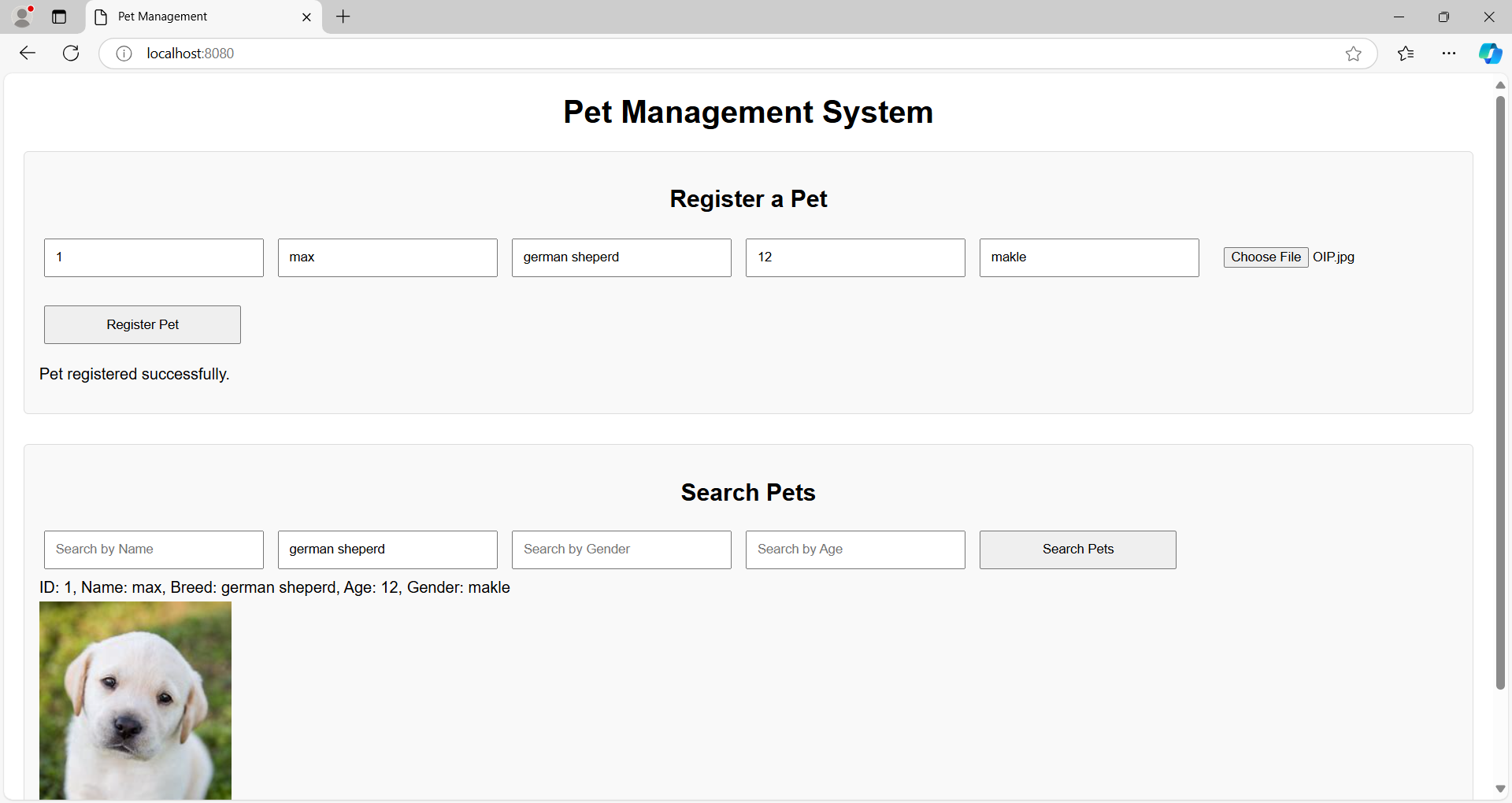


In the above image we have successfully registered a pet with a unique id. Now when we search for the pet with that id we receive the information of the pet as shown below.



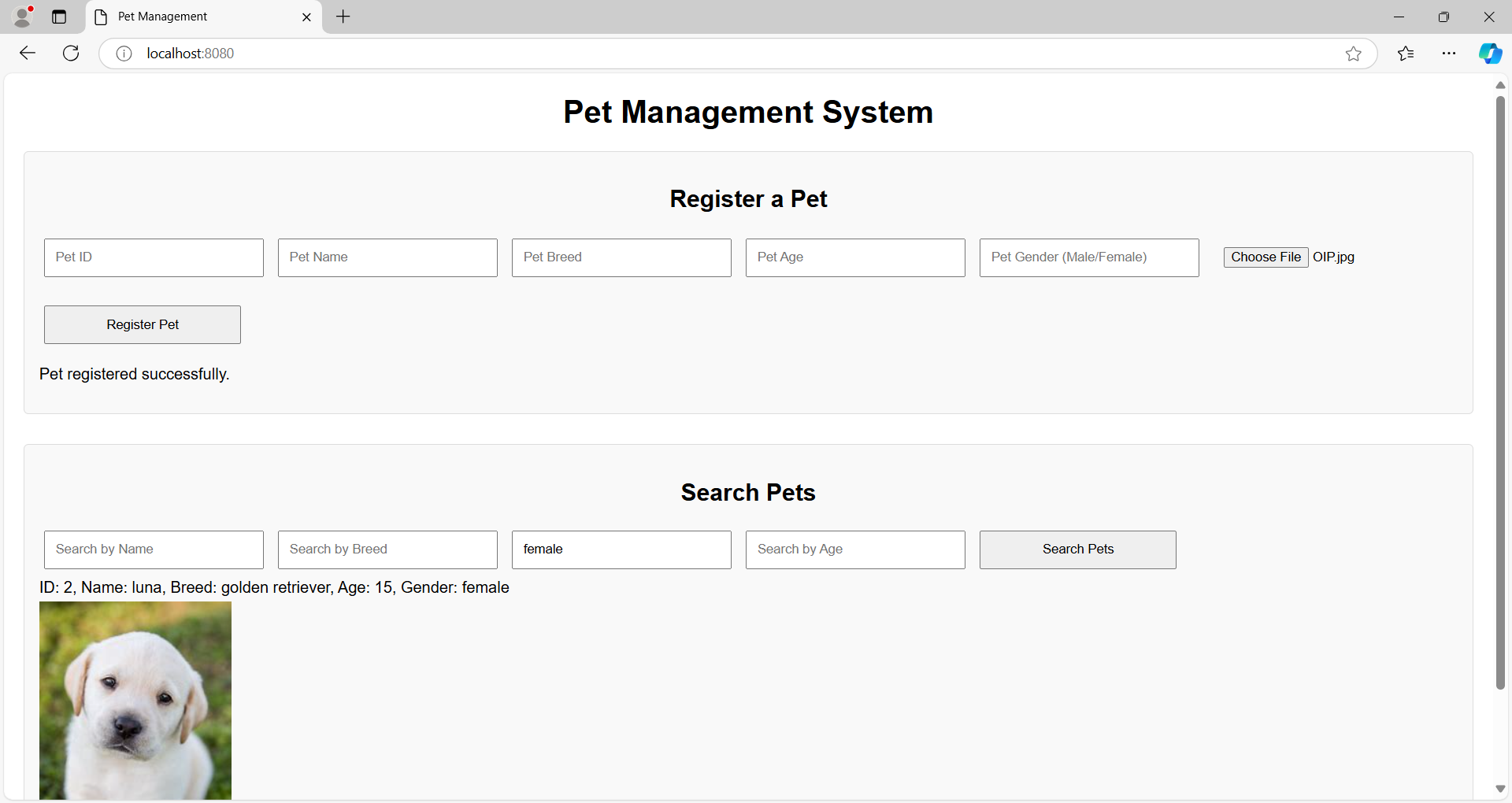
**Search for pet using breed:**

Here we have registered a new pet whose breed is ‘german shepherd’ and when we search for the pets whose breed is german shepherd the pets will be returned.



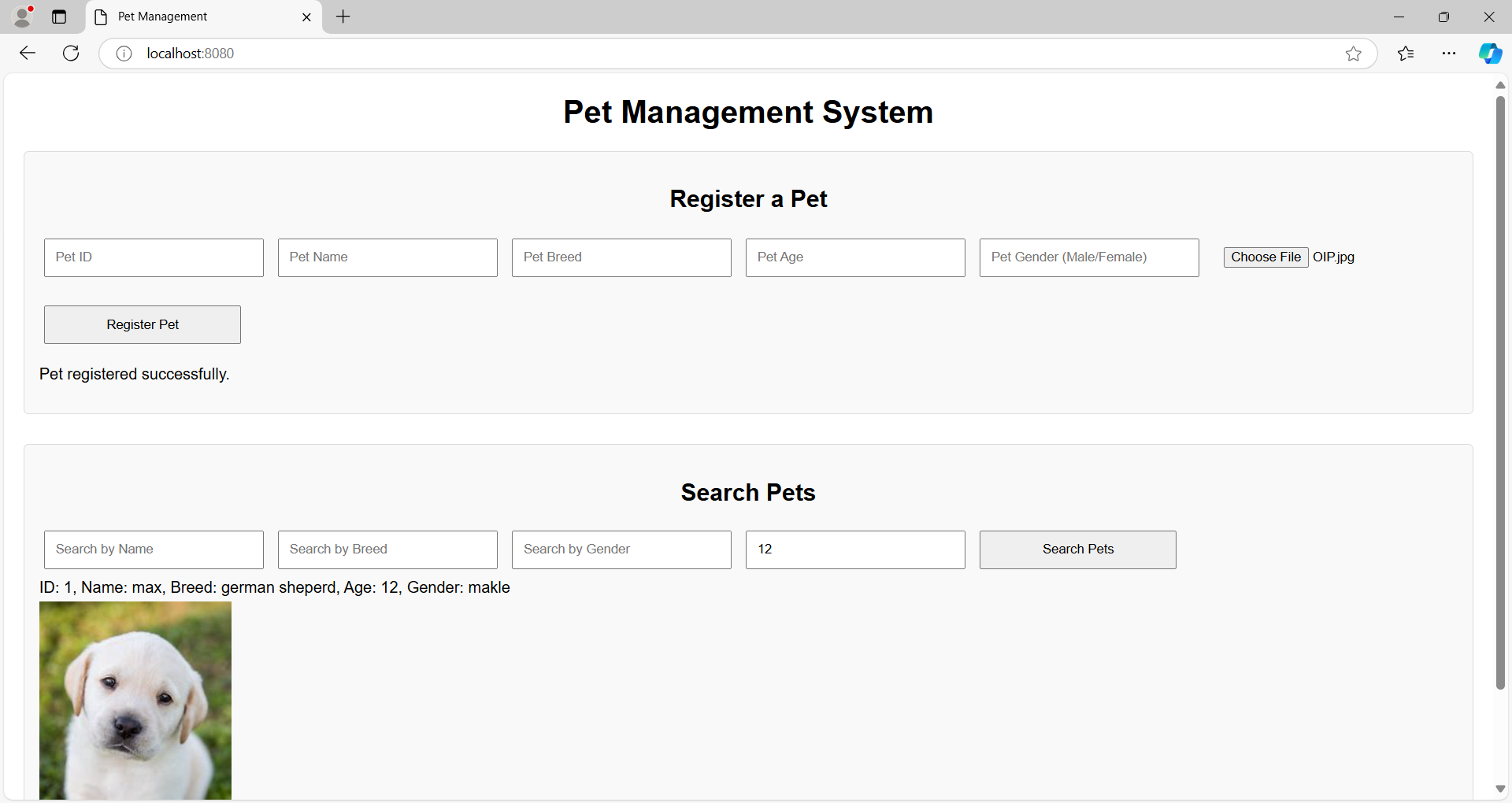
**Search pet by gender:**

We can also perform searches using gender, all pets with that gender will be returned.



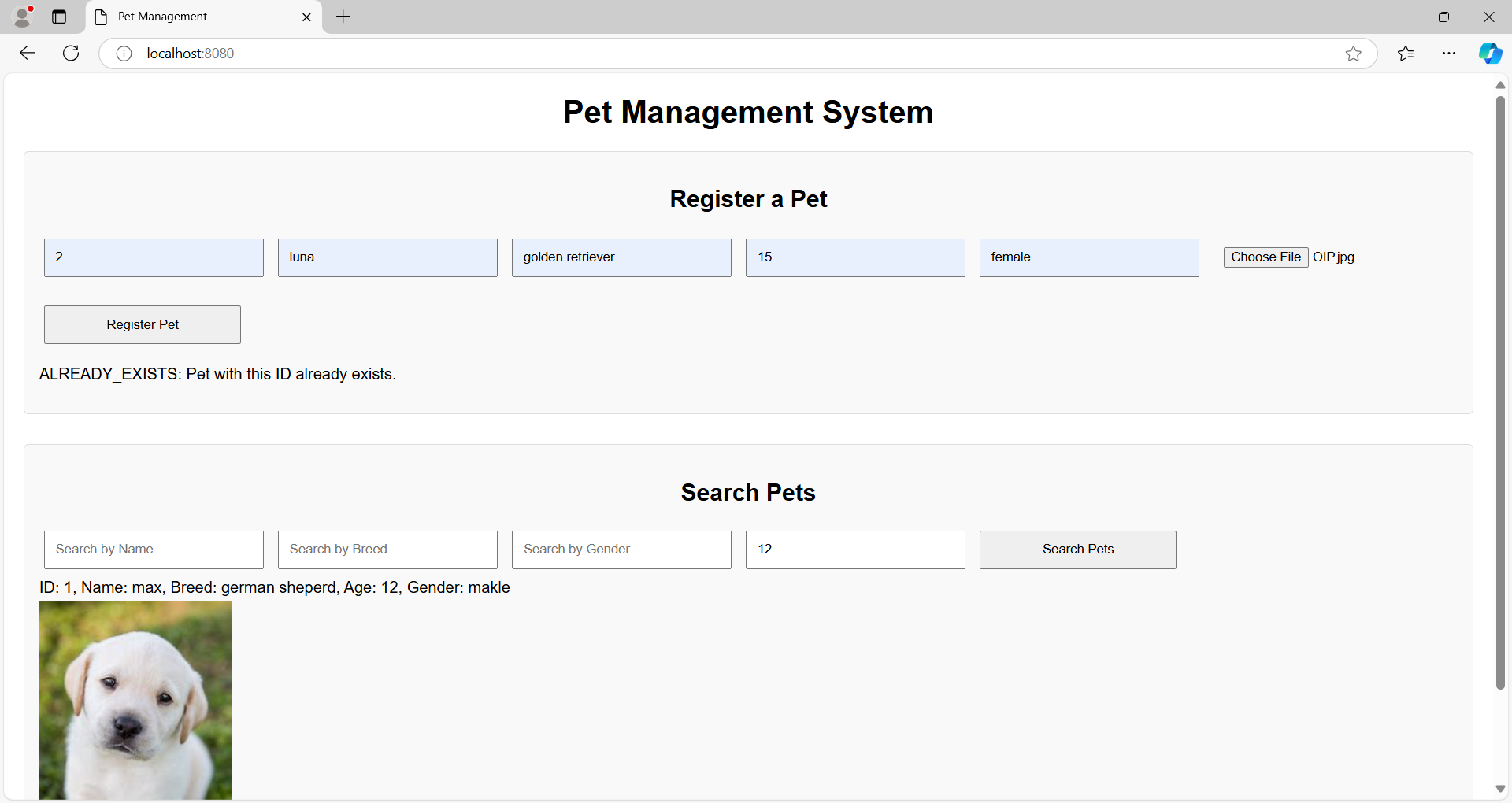
**Searching pet by age:**

If we require a specific aged pet we can just use the search pet by age filter. With this all the pets of that age will be listed.



**Duplicate pet registration:**

The system can detect if we try to register a pet with an ID that already exists, if we try to register a pet with the same id that is already registered the system will show ALREADY\_EXIST message.



**Question 4:**

**Python:**

We have designed the following 5 different test cases for our pet adoption system:

1. **Test Case: Register Pet success**

* **Description:** Test that a pet can be successfully registered.
* **Pre-Condition:** There is no pet in the database with the id 1.
* **Post-Condition:** The pet with new id 1 is stored in the system.
* **Input:** Pet with ID "1", name "Buddy", gender "Male", age 3, breed "Golden Retriever".
* **Expected Output:** Response message "Pet registered successfully." and response pet ID "1".

1. **Test Case: Register Pet Duplicate**

* **Description:** Test that registering a pet with the same ID fails.
* **Pre-Condition:** There is already a pet stored in the database with id 1.
* **Post-Condition:** The pet with same id is considered a duplicate and not stored in the database.
* **Input:** Pet with ID "1", name "Buddy", gender "Male", age 3, breed "Golden Retriever".
* **Expected Output:** Response message "Pet with this ID already exists."

1. **Test Case: Search Pets Success**

* **Description:** Test that searching for an existing pet by name returns results.
* **Pre-Condition**: The pet named ‘buddy’ is in the database.
* **Post-Condition**: The details of the pet are displayed on the screen.
* **Input:** Search for a pet with name "Buddy".
* **Expected Output:** Response containing one pet with name "Buddy" and message "Pets retrieved successfully."

1. **Test Case: Search Pets No Match**

* **Description:** Test that searching for a non-existent pet returns no results.
* **Pre-Condition**: There is no pet with the name “NonExistent” in the database.
* **Post-Condition**: The output is displayed with No pets found.
* **Input:** Search for a pet with name "NonExistent".
* **Expected Output:** Response containing zero pets and message "No pets found matching the criteria."

1. **Test Case: Search Pets Multiple Filters**

* **Description:** Test that searching with multiple filters returns the correct pet.
* **Pre-Condition**: There are pets in the database with the specified filters.
* **Post-Condition**: The output of the pet that satisfy the search criteria.
* **Input:** Register a pet with ID "2", name "Luna", gender "Female", age 2, breed "Labrador". Then search for a pet with name "Luna", gender "Female", breed "Labrador".
* **Expected Output:** Response containing one pet with name "Luna" and breed "Labrador".

**Java:**

1. **Test case: Search Pets By Name**

* **Description:** Test that searching a pet by name returns results.
* **Pre-Condition**: The pet named ‘buddy’ is in the database.
* **Post-Condition**: The details of the pet are displayed on the screen.
* **Input:** Registers a pet named "Buddy", search for a pet named “Buddy”.
* **Expected output:** response containing one pet named “Buddy”.

1. **Test case: Register pet duplicate**

* **Description**: Tests that attempting to register a pet with a duplicate ID fails.
* **Pre-Condition:** There is already a pet stored in the database with id 1.
* **Post-Condition:** The pet with same id is considered a duplicate and not stored in the database.
* **Input**:
  + Register a pet:
    - ID: "1"
    - Name: "Buddy"
    - Gender: "Male"
    - Age: 3
    - Breed: "Labrador"
  + Attempt to register the same pet again.
* **Expected Output**: A StatusRuntimeException should be thrown with the message: "ALREADY\_EXISTS: Pet with this ID already exists."

1. **Test case: Search pets by gender**

* **Description**: Tests searching for pets by their gender.
* **Pre-Condition**: There are pets in the database with the specified filter of gender.
* **Post-Condition**: The output of the pet that satisfy the search criteria.
* **Input**:
* Register a pet:
  + ID: "2"
  + Name: "Bella"
  + Gender: "Female"
  + Age: 2
  + Breed: "Poodle"
* Search request: Gender = "Female"
* **Expected Output**: Response should not be empty. First pet in the response should have the name "Bella" and gender "Female".

1. **Test case: Search for non-existent pet**

* **Description**: Tests searching for a pet that does not exist.
* **Pre-Condition**: There is no pet with the name “Unknown” in the database.
* **Post-Condition**: The output is displayed with No pets found.
* **Input**:
  + Search request: Name = "Unknown"
* **Expected Output**: Response should be empty (no pets found). Response message should indicate: "No pets found matching the criteria."

1. **Test case: Register Multiple Pets And Retrieve All**

* **Description**: Tests registering multiple pets and retrieving them all.
* **Pre-Condition**: Pets don’t exists in the database and their IDs are unique.
* **Post-Condition**: The newly added pets along with all previous pets are retrieved.
* **Input**:
  + Register two pets:
    - Pet 1:
      * ID: "3"
      * Name: "Max"
      * Gender: "Male"
      * Age: 4
      * Breed: "German Shepherd"
    - Pet 2:
      * ID: "4"
      * Name: "Luna"
      * Gender: "Female"
      * Age: 1
      * Breed: "Golden Retriever"
  + Search request: No filters applied (retrieve all pets).
* **Expected Output**: Response should contain 2 pets. First pet in the response should have the name "Max". Second pet in the response should have the name "Luna".