|  |  |
| --- | --- |
| Nikhil Mhatre:1002122555 | Question 1 –Python  Question 2 -Python  Question 3 -Python Client |
| Imtiaz Bin Rahim: 1002076394 | Question 1 -Java  Question 2 -Java  Question 3 –Java Client |

How gRPC works ?  
**Define service**: Create a. proto file with service definitions.

**Generate code**: Use the gRPC compiler to generate server and client code.

**Implement server**: Write the logic for the defined methods in your preferred language.

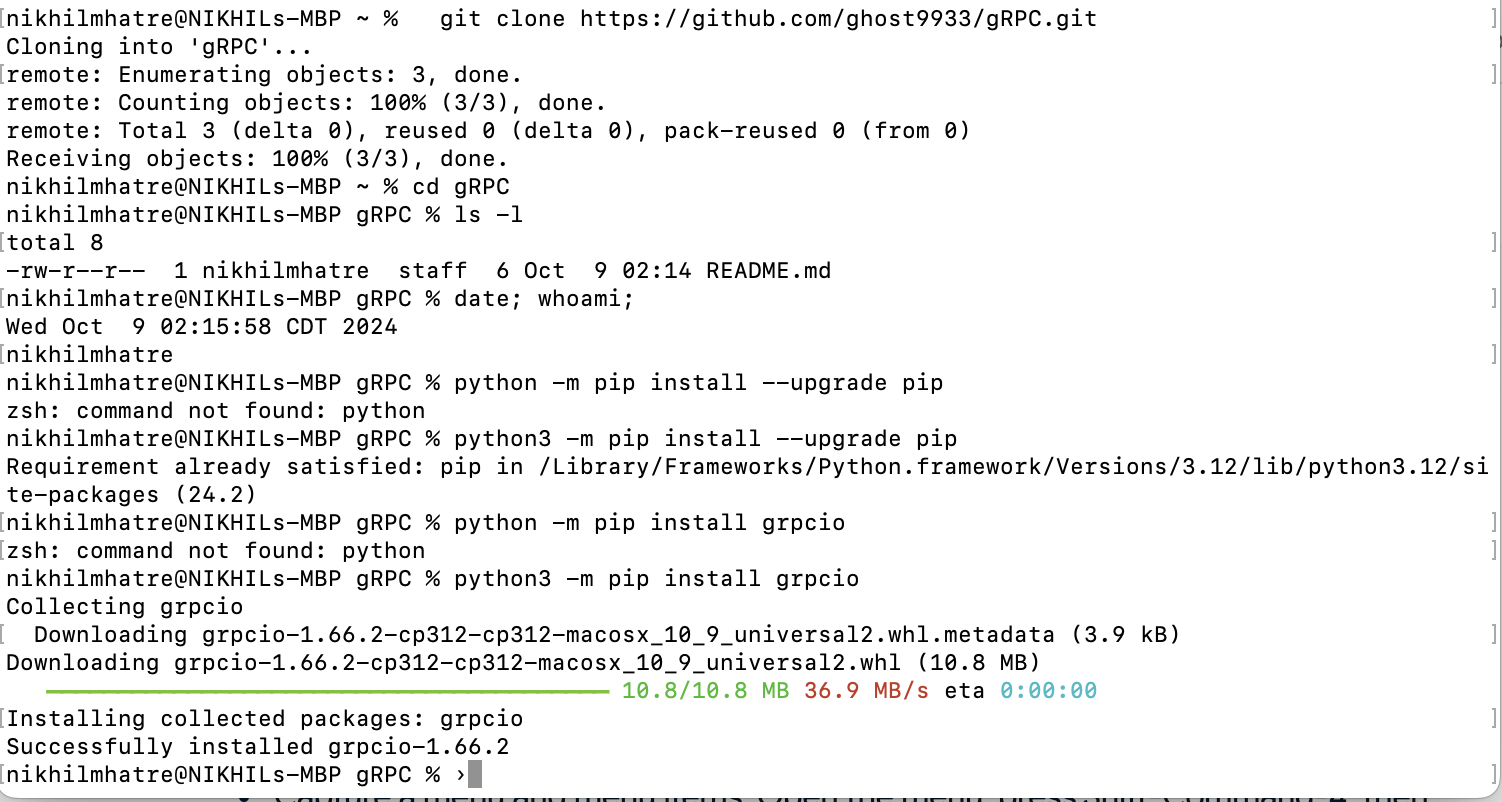
**Invoke client**: Use the generated client code to make remote procedure calls to the server.

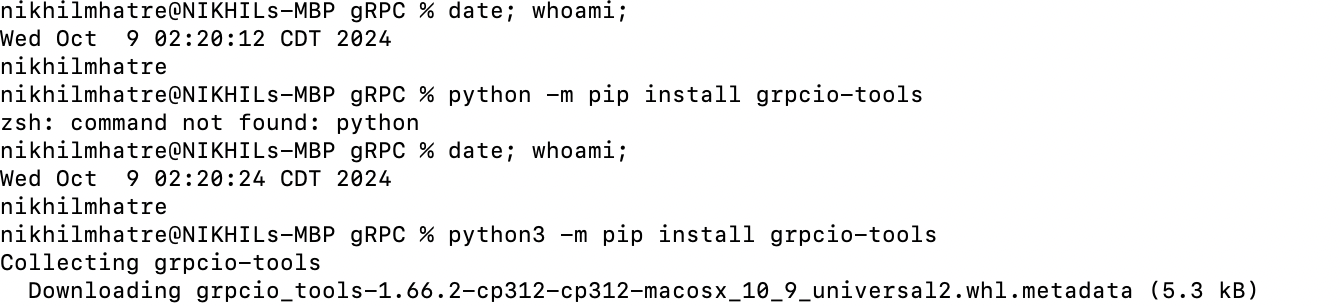
# Question1

## Python

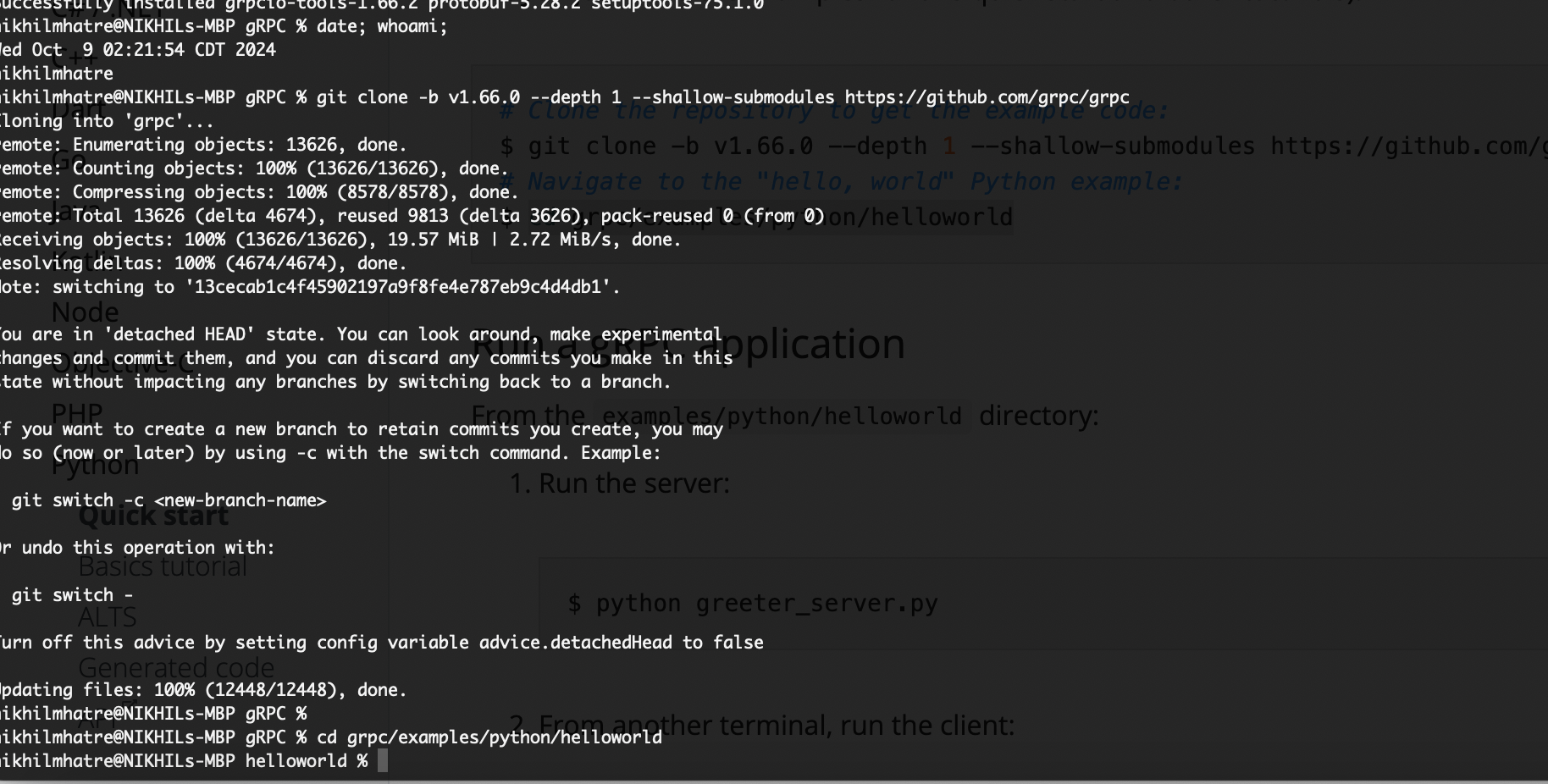
### Setting Up

Installing GRPC

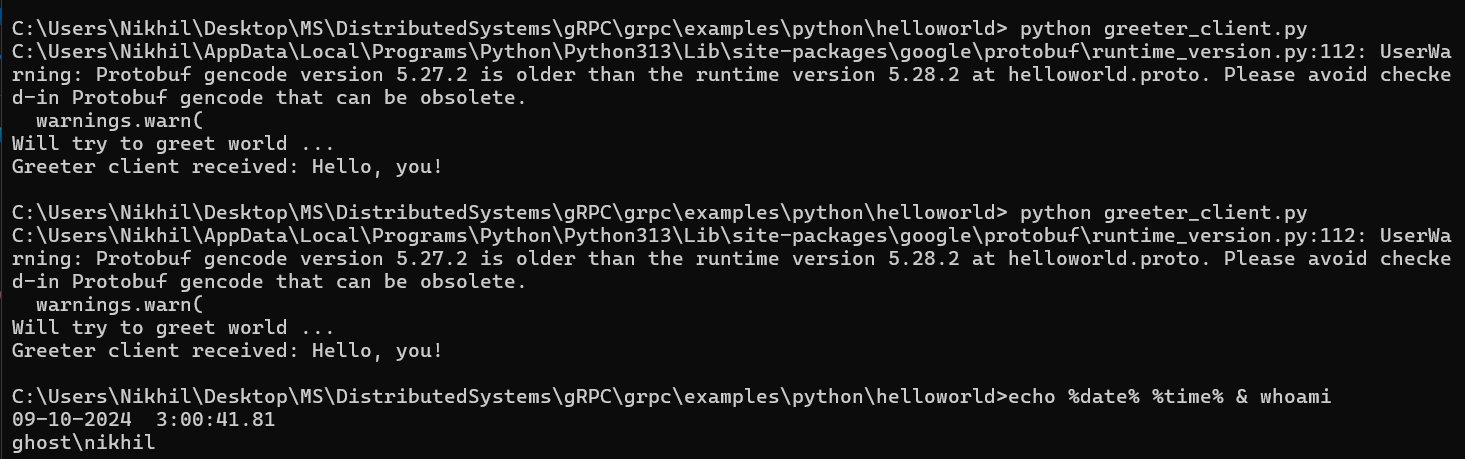


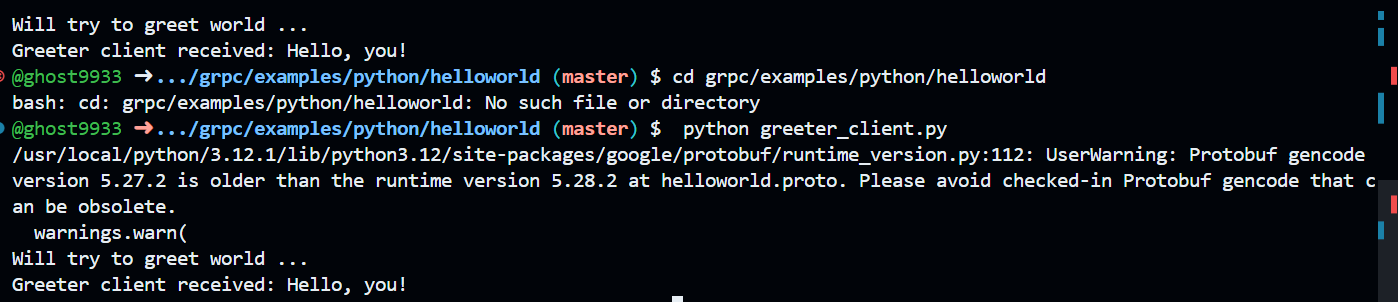


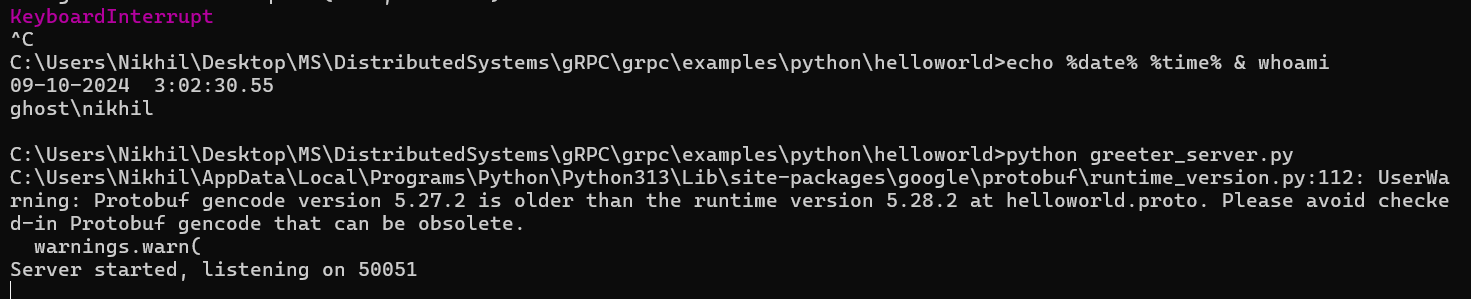
Cloned tutorial

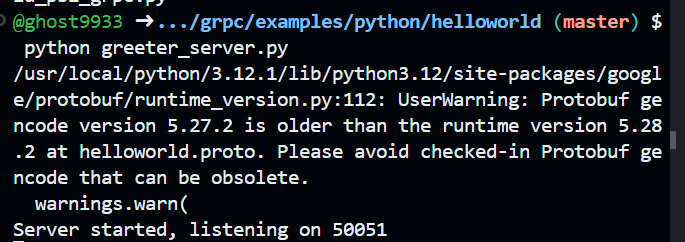


### Quick Start Tutorial

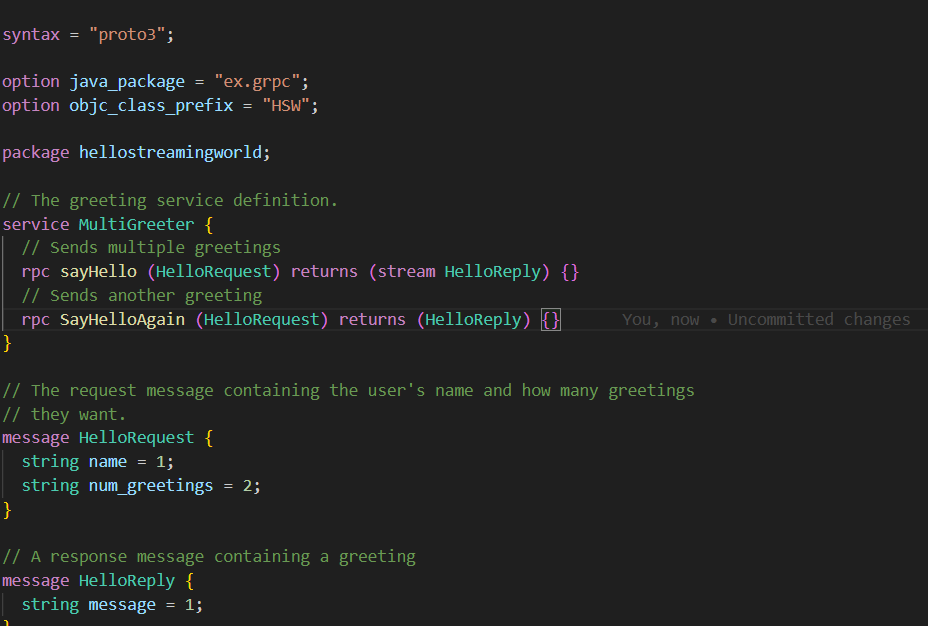
Client  




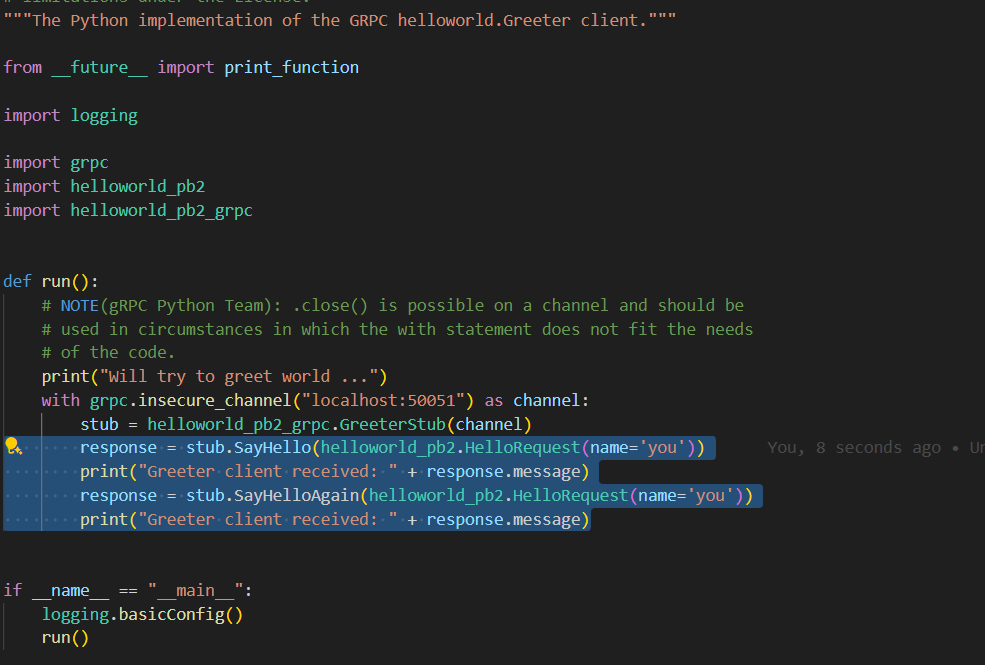
Server:  


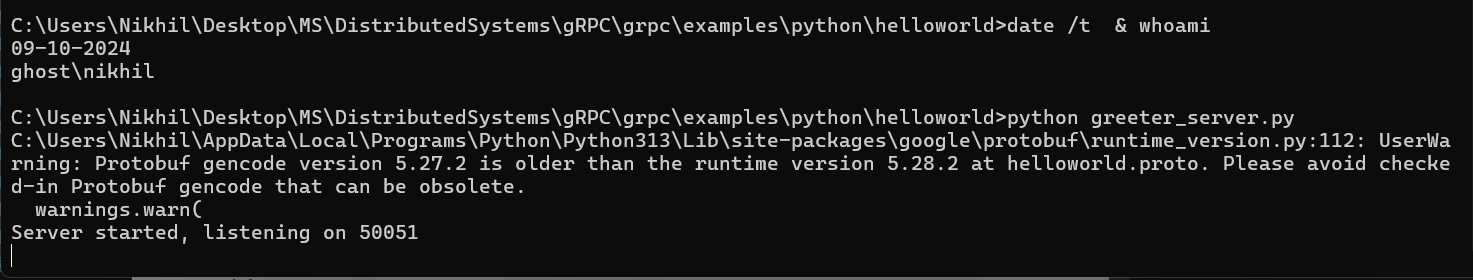


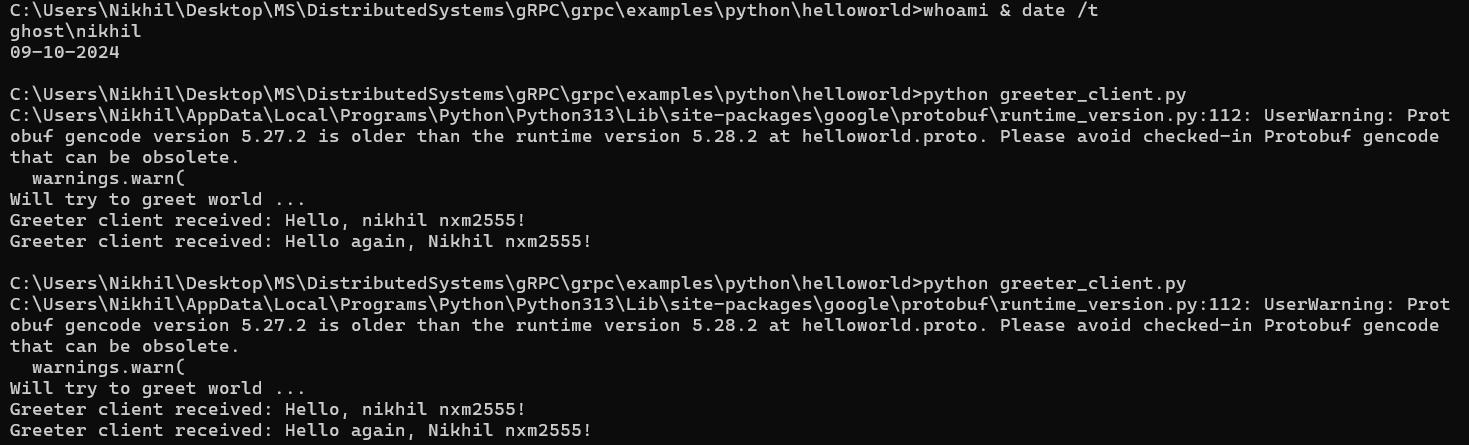
Adding Methods


Modifying the server code   

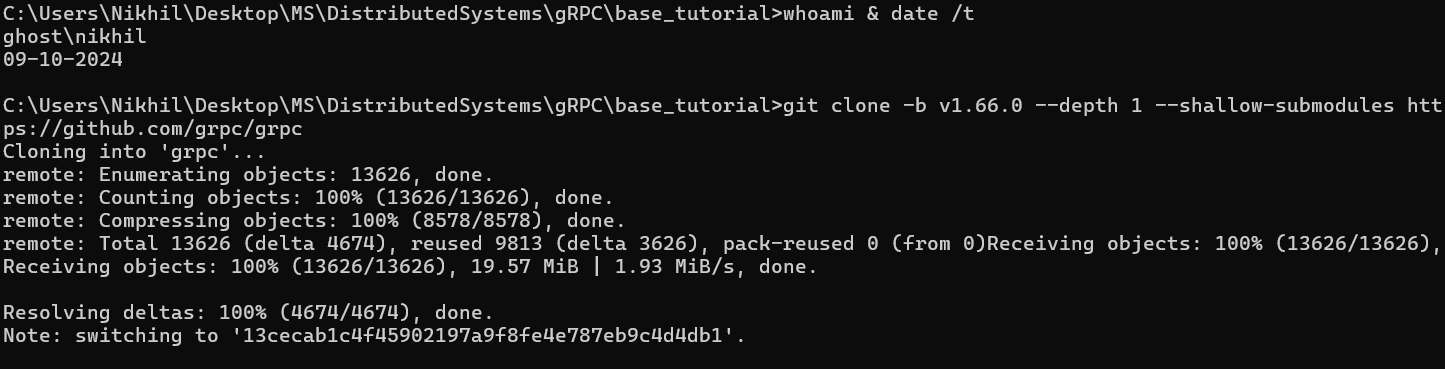

Updating the server  


Starting the server  
  
  
Running the client

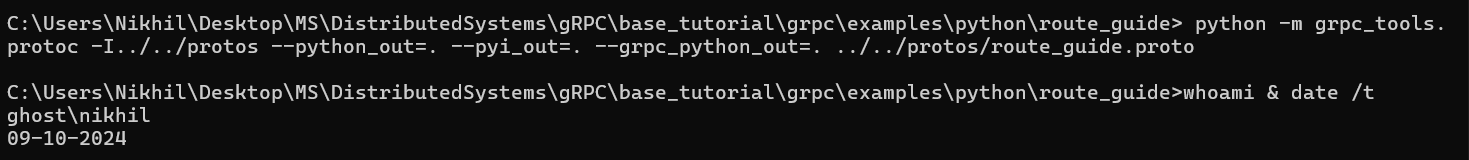


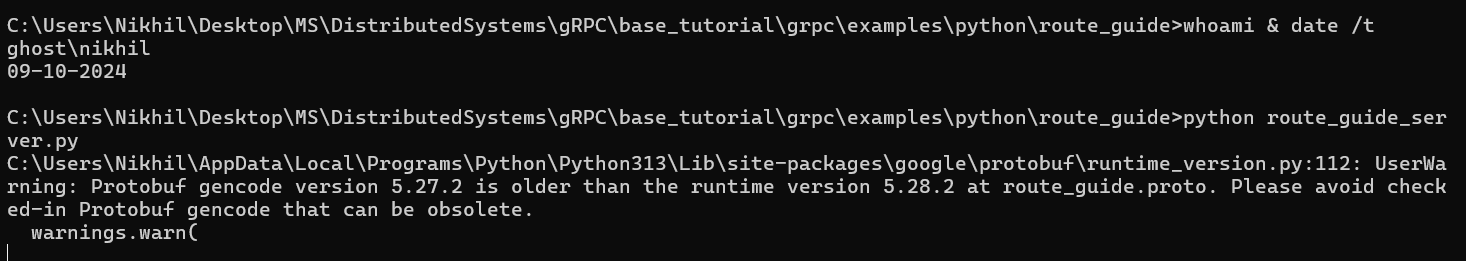
### Basics

Setting up the project files



generate the Python stub code:



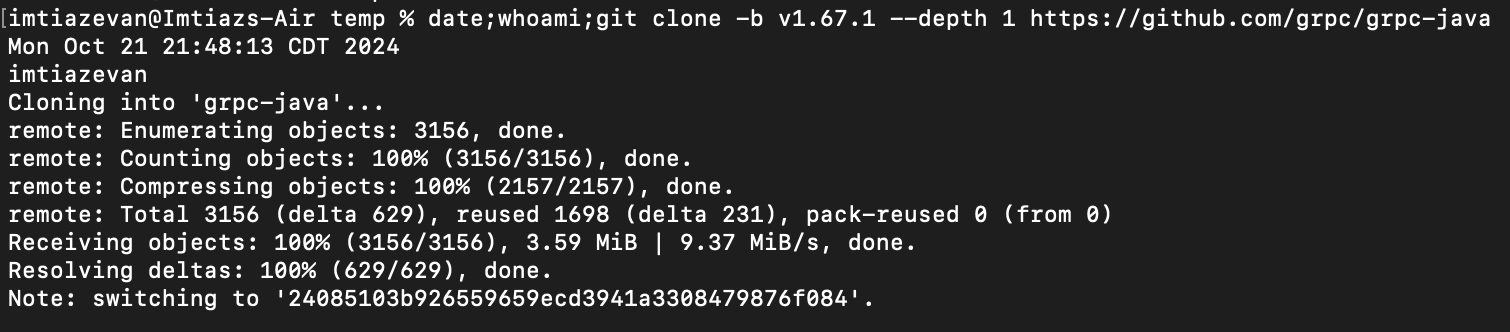
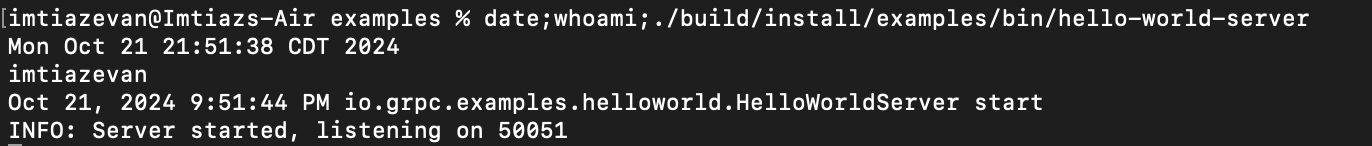
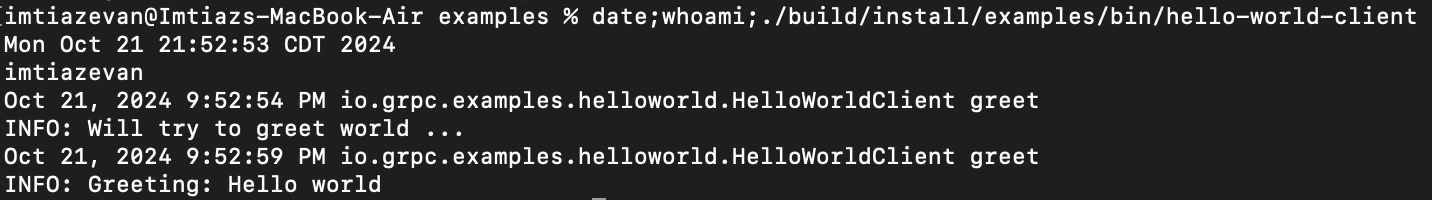
Server   


Client



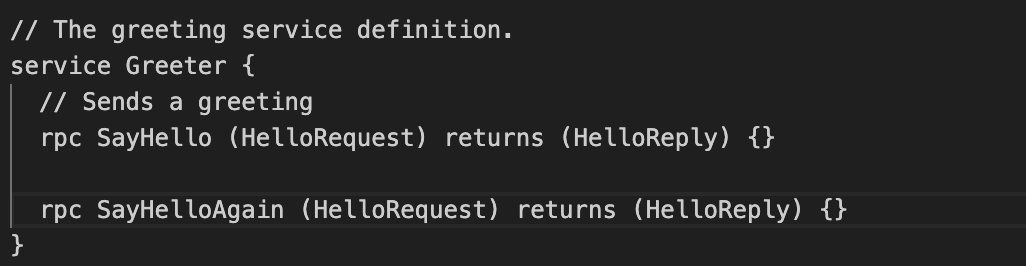
## Java:

### Setting Up:

1. Clone Java gRPC repository
2. Build using gradle
3. Run Server
4. Run Client

### Update The App:

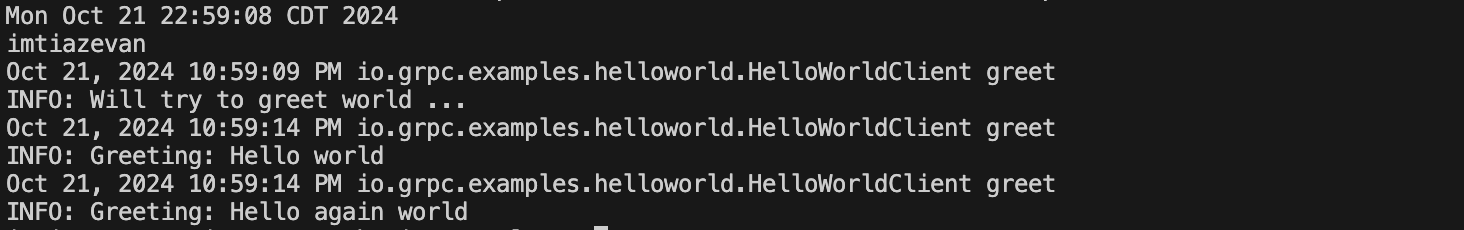
#### Update proto file:



#### Update server file:

#### Update client file:

#### Build and Run:



# Question 2

## Python:

### Step 1

#### writing the proto buff script to define the required interfaces for the communication between the server and client

**syntax = "proto3";**

**package todo;**

**service TodoService {**

**rpc CreateTodo (TodoRequest) returns (TodoResponse);**

**rpc DeleteTodo (TodoId) returns (TodoResponse);**

**rpc UpdateTodo (TodoRequest) returns (TodoResponse);**

**rpc ListTodos (Empty) returns (TodoListResponse);**

**}**

**message TodoItem {**

**string id = 1;**

**string title = 2;**

**bool completed = 3;**

**}**

**message TodoRequest {**

**TodoItem item = 1;**

**}**

**message TodoId {**

**string id = 1;**

**}**

**message TodoResponse {**

**string message = 1;**

**TodoItem item = 2; // Optional: return the item in the response**

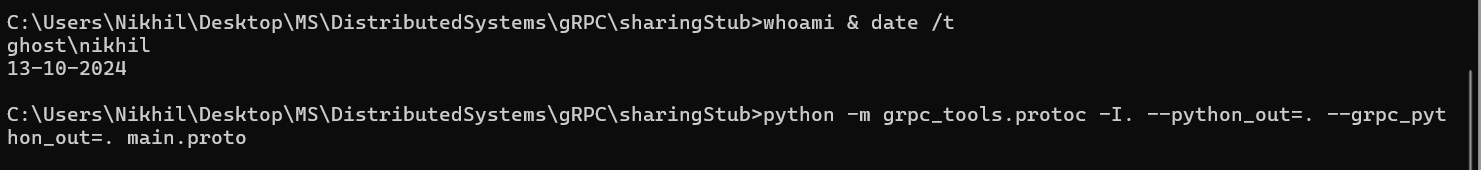
**}**

**message Empty {}**

**message TodoListResponse {**

**repeated TodoItem items = 1;**

**}**



#### Creating the stub codes using the proto definition for our todo app

#### Client code:

import grpc

import main\_pb2

import main\_pb2\_grpc

def print\_todos(todos):

"""Utility function to print all todos."""

print("Current Todos:")

for item in todos:

print(f"{item.id}: {item.title} (Completed: {item.completed})")

def run():

with grpc.insecure\_channel('localhost:50051') as channel:

stub = main\_pb2\_grpc.TodoServiceStub(channel)

while True:

print("\nOptions:")

print("1. Create Todo")

print("2. List Todos")

print("3. Update Todo")

print("4. Delete Todo")

print("5. Exit")

choice = input("Choose an option (1-5): ")

if choice == '1':

title = input("Enter the title of the Todo item: ")

response = stub.CreateTodo(main\_pb2.TodoRequest(item=main\_pb2.TodoItem(title=title)))

print(response.message)

elif choice == '2':

response = stub.ListTodos(main\_pb2.Empty())

print\_todos(response.items)

elif choice == '3':

todo\_id = input("Enter the ID of the Todo item to update: ")

title = input("Enter the new title of the Todo item: ")

completed = input("Is the Todo item completed? (yes/no): ")

completed = True if completed.lower() == 'yes' else False

response = stub.UpdateTodo(main\_pb2.TodoRequest(item=main\_pb2.TodoItem(id=todo\_id, title=title, completed=completed)))

print(response.message)

elif choice == '4':

todo\_id = input("Enter the ID of the Todo item to delete: ")

response = stub.DeleteTodo(main\_pb2.TodoId(id=todo\_id))

print(response.message)

elif choice == '5':

print("Exiting...")

break

else:

print("Invalid option, please try again.")

if \_\_name\_\_ == '\_\_main\_\_':

run()

### 

### 

### 

#### Server Code:

### 

import grpc

from concurrent import futures

import time

import main\_pb2

import main\_pb2\_grpc

class TodoService(main\_pb2\_grpc.TodoServiceServicer):

def \_\_init\_\_(self):

self.todos = {} # In-memory database for todo items

def CreateTodo(self, request, context):

todo\_id = str(len(self.todos) + 1)

todo\_item = main\_pb2.TodoItem(id=todo\_id, title=request.item.title, completed=False)

self.todos[todo\_id] = todo\_item

print('request recived to create',request.item.id,request.item)

return main\_pb2.TodoResponse(message="Todo created successfully", item=todo\_item)

def DeleteTodo(self, request, context):

print('request recived to delete',request.id)

if request.id in self.todos:

del self.todos[request.id]

return main\_pb2.TodoResponse(message="Todo deleted successfully")

return main\_pb2.TodoResponse(message="Todo not found")

def UpdateTodo(self, request, context):

print('request recived to update',request.item.id,request.item)

if request.item.id in self.todos:

self.todos[request.item.id] = request.item

return main\_pb2.TodoResponse(message="Todo updated successfully", item=request.item)

return main\_pb2.TodoResponse(message="Todo not found")

def ListTodos(self, request, context):

items = list(self.todos.values())

print('seding this data to client',items)

return main\_pb2.TodoListResponse(items=items)

def serve():

server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))

main\_pb2\_grpc.add\_TodoServiceServicer\_to\_server(TodoService(), server)

server.add\_insecure\_port('[::]:50051')

server.start()

print("gRPC server started, listening on port 50051")

print("Server started at port 50051.")

try:

while True:

time.sleep(86400) # Keep the server running for a day

except KeyboardInterrupt:

print('TODO server stopping')

server.stop(0)

if \_\_name\_\_ == '\_\_main\_\_':

print('starting TODO server')

serve()

### Step 2:

#### Starting the server:

### 

## 

### Step 3:

#### Starting the client:

### 

### Step 4: Test the code Using the client side

#### Debugging updates on the server side

## JAVA:

### Step 1: Generate stub using proto file

### 

### Step 2: build and run server

### Step 3: Run Client and send system username to server for a customized greeting message

### 

# Question 3:

## The Proto file

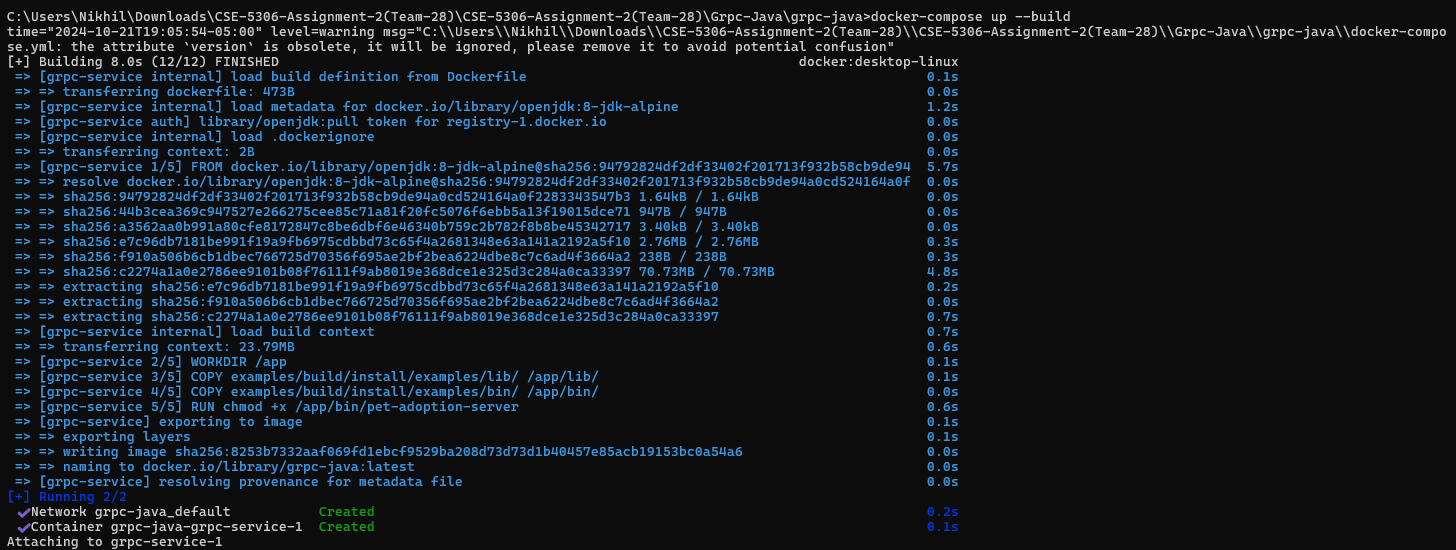
use the proto file to generate the stubs for java and python

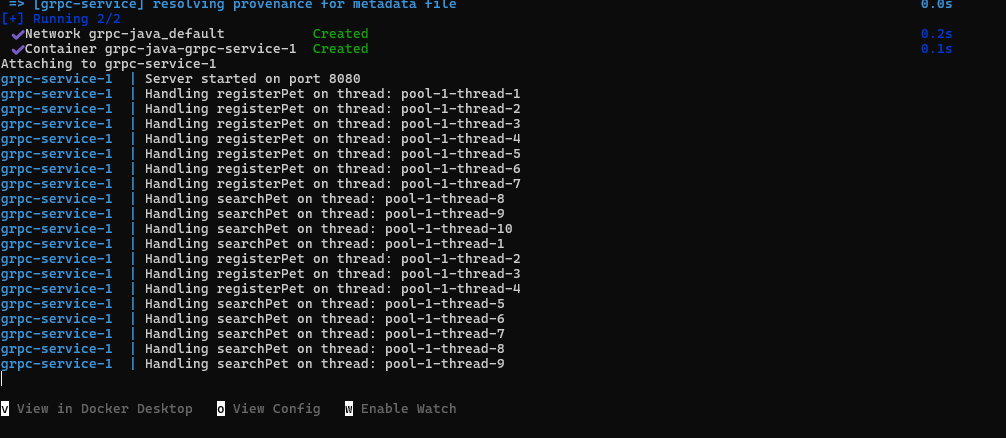
#### Used the below command to generate the python stub

python -m grpc\_tools.protoc -I. --python\_out=. --grpc\_python\_out=. Petadoption.proto

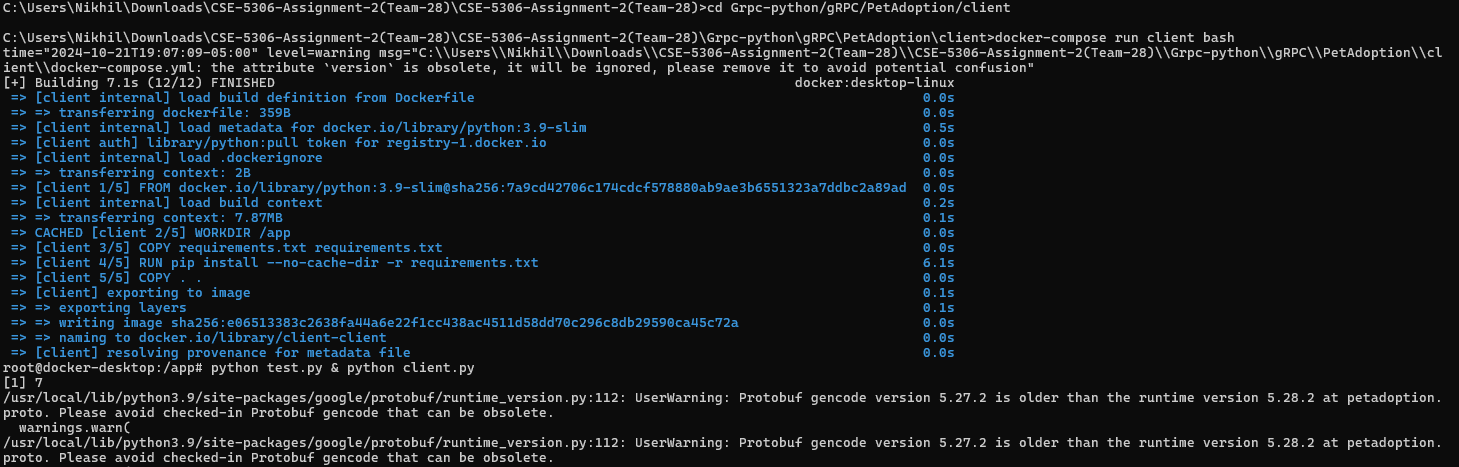
For java modify build.gadle to generate the java stub code

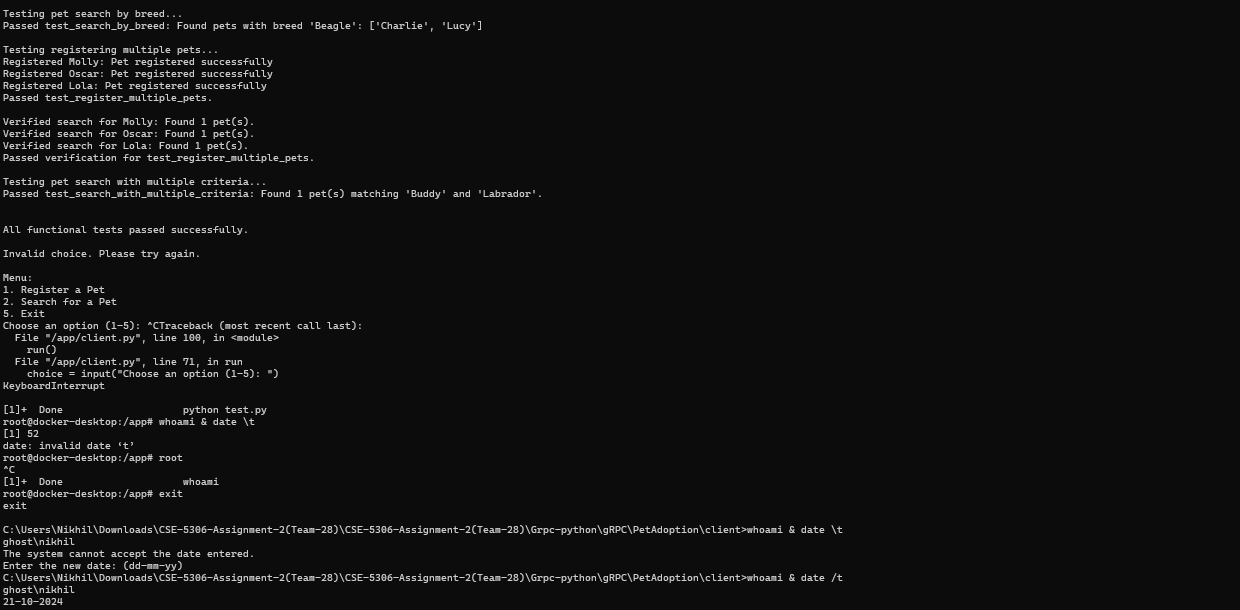
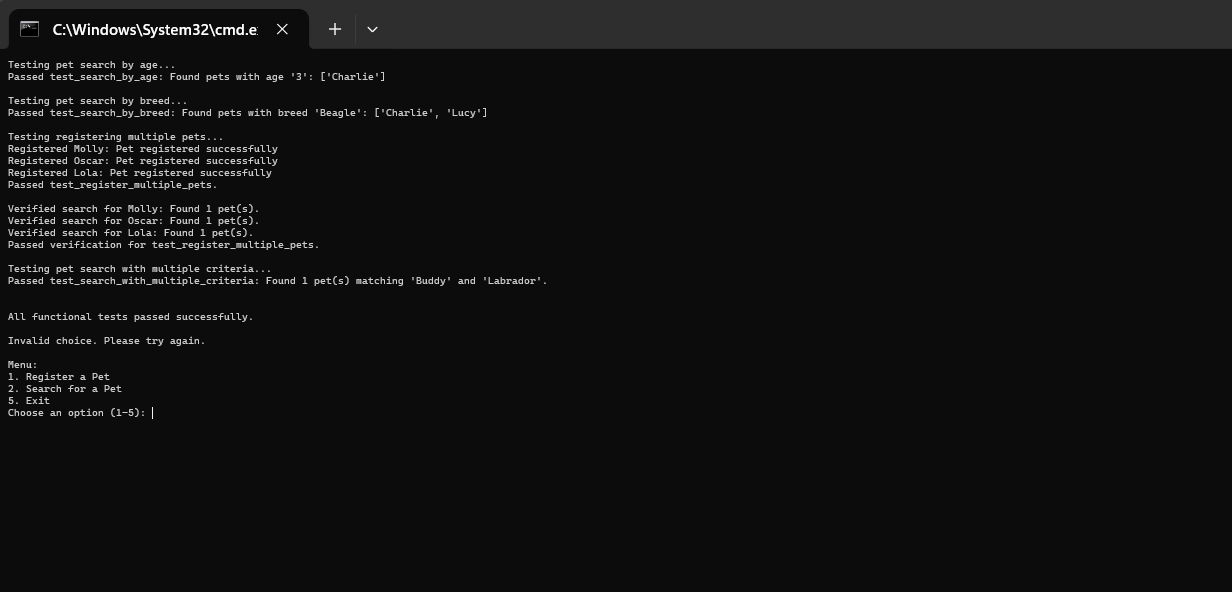
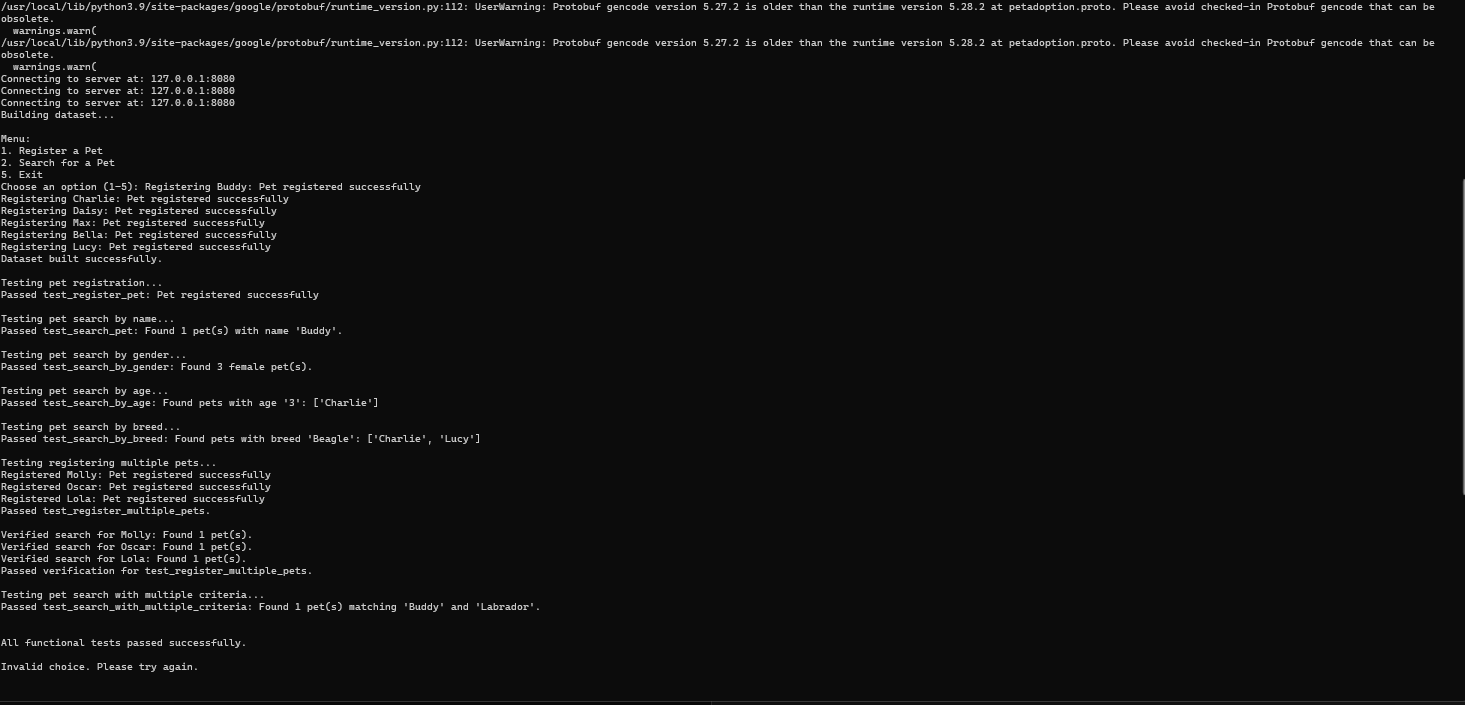
## The Server: implemented in java

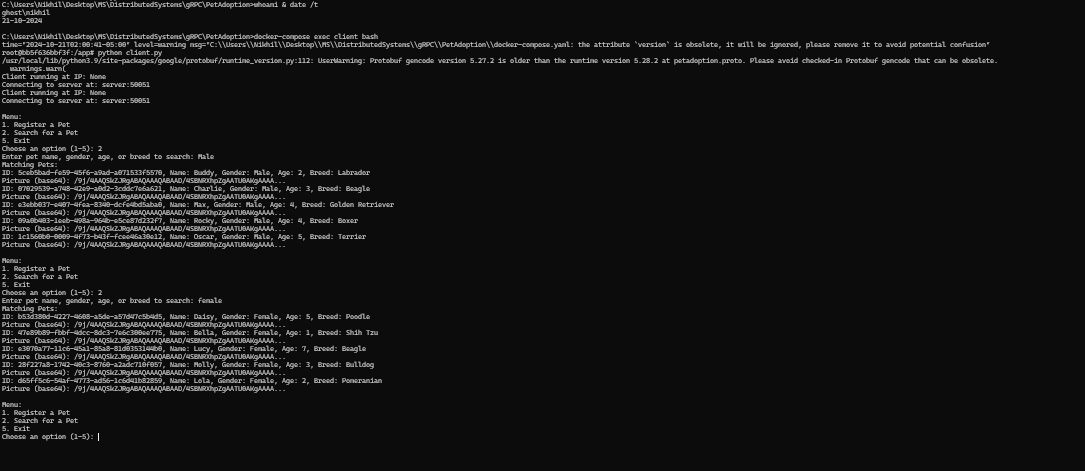




## The Client: implemented in Python





Testing the client   


# Question 4: Testing

The script tests the core functionalities of a gRPC-based pet adoption system by registering pets and searching for them based on various criteria.

* **Connect to Server:** Establishes a gRPC connection to the pet adoption server.
* **Build Dataset:** Registers a predefined list of pets to set up the testing environment.
* **Run Tests:** Executes each test function sequentially to verify registration and search functionalities.
* **Report Results:** Outputs the status of each test and summarizes any failures.

**Initial Dataset Registration (build\_dataset)**

* **Registered Pets:**
  + Buddy (Male, 2, Labrador)
  + Charlie (Male, 3, Beagle)
  + Daisy (Female, 5, Poodle)
  + Max (Male, 4, Golden Retriever)
  + Bella (Female, 1, Shih Tzu)
  + Lucy (Female, 7, Beagle)

1. test\_register\_pet

Verifies that a new pet can be successfully registered with all required details.

* + **Registers:** Rocky (Male, 4, Boxer)
  + **Verification:** Confirms Rocky is successfully registered.

1. test\_search\_pet

Ensures that searching for a pet by name returns the correct pet(s).

* + **Searches for:** Buddy
  + **Expected Result:** Buddy should be found among the registered pets.

1. test\_search\_by\_gender

Checks that searching by gender retrieves only pets of the specified gender.

* + **Searches for:** Female
  + **Expected Count:** 3 female pets (Daisy, Bella, Lucy)
  + **Verification:** Ensures only female pets are returned and count is as expected.

1. test\_search\_by\_age

Validates that searching by age returns pets matching the specified age.

* + **Searches for:** Age 3
  + **Expected Pet:** Charlie
  + **Verification:** Confirms Charlie is retrieved when searching by age.

1. test\_search\_by\_breed

Confirms that searching by breed retrieves all pets of the specified breed.

* + **Searches for:** Beagle
  + **Expected Pets:** Charlie and Lucy
  + **Verification:** Ensures both Beagle breeds are returned

1. test\_register\_multiple\_pets

Tests the registration of multiple pets and verifies their availability in searches.

* + **Registers:** Molly (Female, 3, Bulldog), Oscar (Male, 5, Terrier), Lola (Female, 2, Pomeranian)
  + **Verification:** Confirms each newly registered pet can be searched and found.

1. test\_search\_with\_multiple\_criteria

Ensures that searching with multiple criteria returns the correct pet(s) matching all criteria.

* + **Searches for:** "Buddy Labrador"
  + **Expected Pet:** Buddy matching both name and breed
  + **Verification:** Ensures multi-criteria search returns the correct pet.