PROJECT ASSIGNMENT 2 – DISTRIBUTED SYSTEMS: BACKEND IMAGE SEARCH ENGINE

TEAM 06:

Minhaz Bin Farukee (UTA ID: 1002154424)

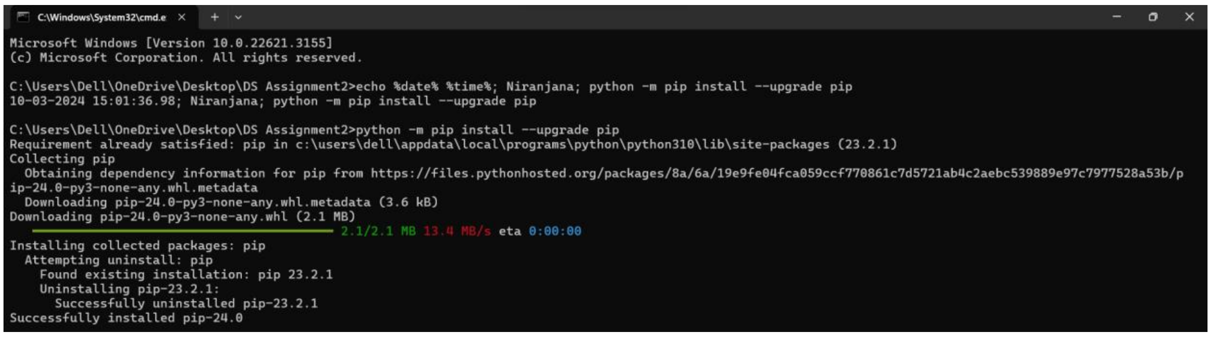
Niranjana Subramanian (UTA ID: 1002046305)

Question1) Language-Specific Guides to get started with gRPC

PYTHON IMPLEMENTATION:

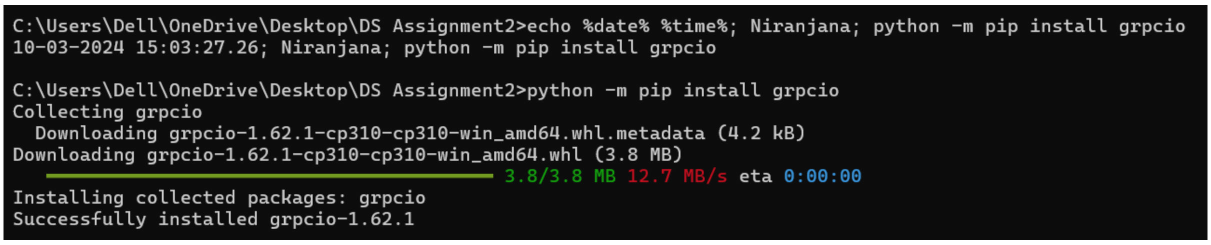
1. $ python -m pip install --upgrade pip

To get started with the version 9.0.1 or higher version of pip, you need to upgrade the pip installed on your machine.



```
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>echo %date% %time%; Niranjana; python -m pip install --upgrade pip
10-03-2024 15:01:36.98; Niranjana; python -m pip install --upgrade pip

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages (23.2.1)
Collecting pip
  Obtaining dependency information for pip from https://files.pythonhosted.org/packages/8a/6a/19e9fe04fca059ccf770861c7d5721ab4c2aebc539889e97c7977528a53b/p
ip-24.0-py3-none-any.whl.metadata
  Downloading pip-24.0-py3-none-any.whl.metadata (3.6 kB)
Downloading pip-24.0-py3-none-any.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 13.4 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.2.1
    Uninstalling pip-23.2.1:
      Successfully uninstalled pip-23.2.1
Successfully installed pip-24.0
```

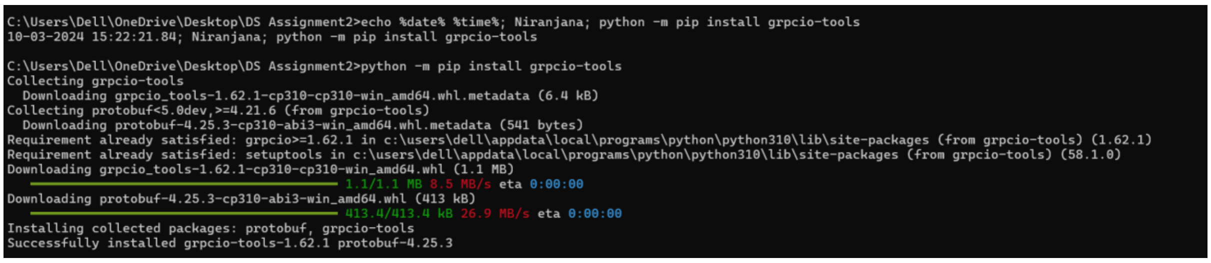
2. python -m pip install grpcio : This is used to install gRPC inside the root directory.



```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2>echo %date% %time%; Niranjana; python -m pip install grpcio
10-03-2024 15:03:27.26; Niranjana; python -m pip install grpcio

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>python -m pip install grpcio
Collecting grpcio
  Downloading grpcio-1.62.1-cp310-cp310-win_amd64.whl.metadata (4.2 kB)
Downloading grpcio-1.62.1-cp310-cp310-win_amd64.whl (3.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 12.7 MB/s eta 0:00:00
Installing collected packages: grpcio
Successfully installed grpcio-1.62.1
```

3. python -m pip install grpcio-tools

gRPC needs a proto buffer compiler (protoc) and service definitions (.proto) to create a special plugin for server and client to generate those definitions.

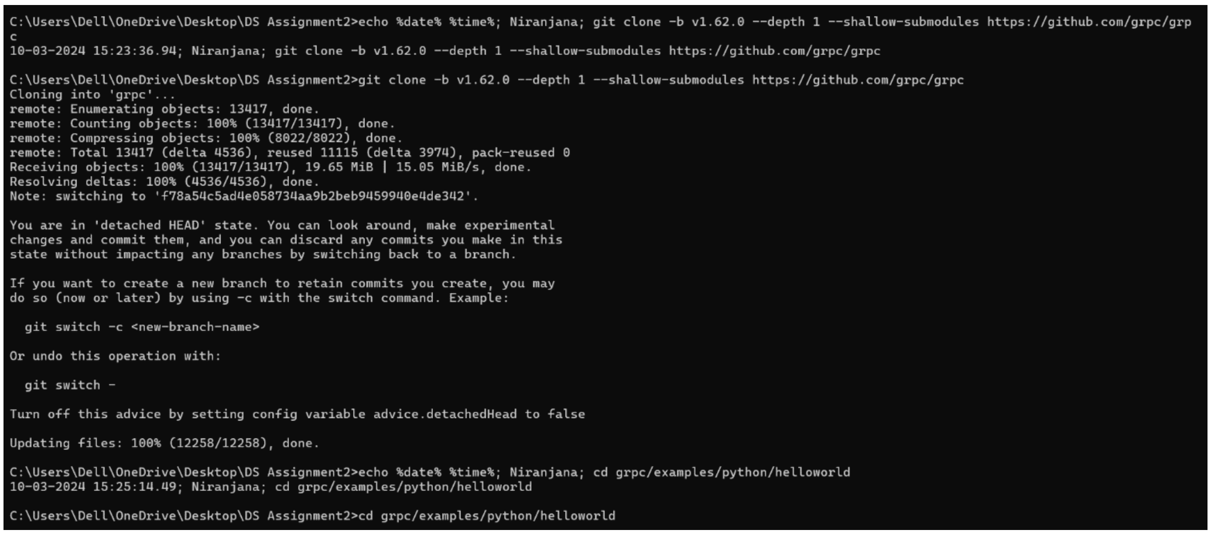


```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2>echo %date% %time%; Niranjana; python -m pip install grpcio-tools
10-03-2024 15:22:21.84; Niranjana; python -m pip install grpcio-tools

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>python -m pip install grpcio-tools
Collecting grpcio-tools
  Downloading grpcio_tools-1.62.1-cp310-cp310-win_amd64.whl.metadata (6.4 kB)
Collecting protobuf<5.0dev,>=4.21.6 (from grpcio-tools)
  Downloading protobuf-4.25.3-cp310-abi3-win_amd64.whl.metadata (541 bytes)
Requirement already satisfied: grpcio>=1.62.1 in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages (from grpcio-tools) (1.62.1)
Requirement already satisfied: setuptools in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages (from grpcio-tools) (58.1.0)
Downloading grpcio_tools-1.62.1-cp310-cp310-win_amd64.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 8.5 MB/s eta 0:00:00
Downloading protobuf-4.25.3-cp310-abi3-win_amd64.whl (413 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 413.4/413.4 kB 26.9 MB/s eta 0:00:00
Installing collected packages: protobuf, grpcio-tools
Successfully installed grpcio-tools-1.62.1 protobuf-4.25.3
```

4. git clone -b v1.62.0 --depth 1 --shallow-submodules https://github.com/grpc/grpc

cd grpc/examples/python/helloworld

This command clones the example code from github that we work on and navigates to the examples/python/helloworld path.

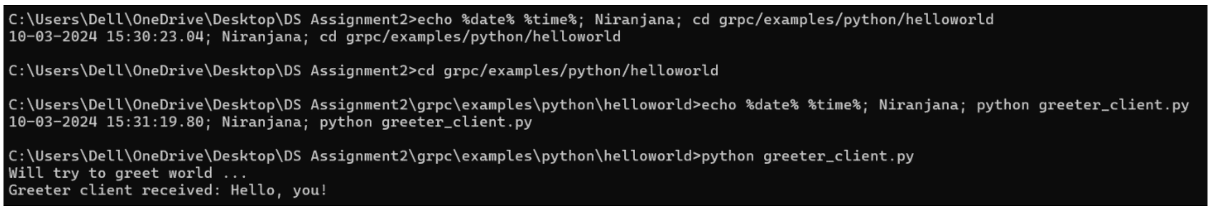


```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2>echo %date% %time%; Niranjana; git clone -b v1.62.0 --depth 1 --shallow-submodules https://github.com/grpc/grp
c
10-03-2024 15:23:36.94; Niranjana; git clone -b v1.62.0 --depth 1 --shallow-submodules https://github.com/grpc/grpc

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>git clone -b v1.62.0 --depth 1 --shallow-submodules https://github.com/grpc/grpc
Cloning into 'grpc'...
remote: Enumerating objects: 13417, done.
remote: Counting objects: 100% (13417/13417), done.
remote: Compressing objects: 100% (8022/8022), done.
remote: Total 13417 (delta 4536), reused 11115 (delta 3974), pack-reused 0
Receiving objects: 100% (13417/13417), 19.65 MiB | 15.05 MiB/s, done.
Resolving deltas: 100% (4536/4536), done.
Note: switching to 'f78a54c5ad4e058734aa9b2beb9459940e4de342'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

Updating files: 100% (12258/12258), done.

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>echo %date% %time%; Niranjana; cd grpc/examples/python/helloworld
10-03-2024 15:25:14.49; Niranjana; cd grpc/examples/python/helloworld

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>cd grpc/examples/python/helloworld
```

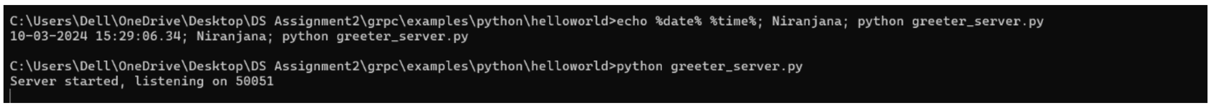
5. Before updating the server, run the server and client application with gRPC under examples/python/helloworld path

python greeter_server.py

python greeter_client.py



```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2>echo %date% %time%; Niranjana; cd grpc/examples/python/helloworld
10-03-2024 15:30:23.04; Niranjana; cd grpc/examples/python/helloworld

C:\Users\Dell\OneDrive\Desktop\DS Assignment2>cd grpc/examples/python/helloworld

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>echo %date% %time%; Niranjana; python greeter_client.py
10-03-2024 15:31:19.80; Niranjana; python greeter_client.py

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>python greeter_client.py
Will try to greet world ...
Greeter client received: Hello, you!
```

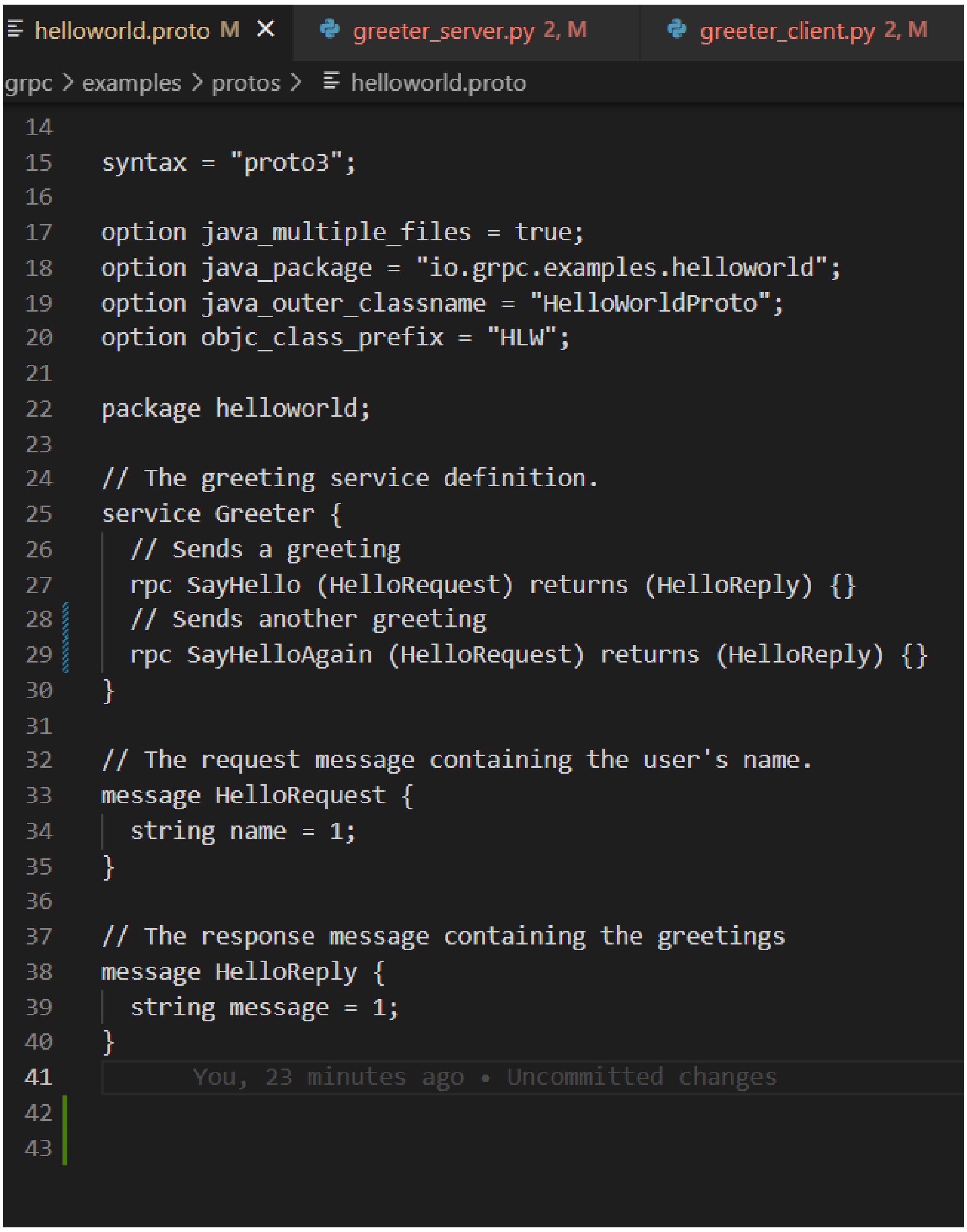


```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>echo %date% %time%; Niranjana; python greeter_server.py
10-03-2024 15:29:06.34; Niranjana; python greeter_server.py

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>python greeter_server.py
Server started, listening on 50051
```

6. // The greeting service definition.

```
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}
// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}
// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

We update the helloworld.proto file to add another function ‘SayHelloAgain’ so that the RPC proto file has a stub that it communicates with the server and client and it works based on the parameter ‘HelloRequest’ and returns reply as ‘HelloReply’.


helloworld.proto M
greeter_server.py 2, M
greeter_client.py 2, M
grpc > examples > protos > helloworld.proto
You, 23 minutes ago • Uncommitted changes

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.examples.helloworld";
option java_outer_classname = "HelloWorldProto";
option objc_class_prefix = "HLW";

package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

7. python -m grpc_tools.protoc -I../../protos --python_out=. --pyi_out=. --grpc_python_out=. ../../protos/helloworld.proto

The above command is used to update the gRPC code to use the new server that we created in the prior step.

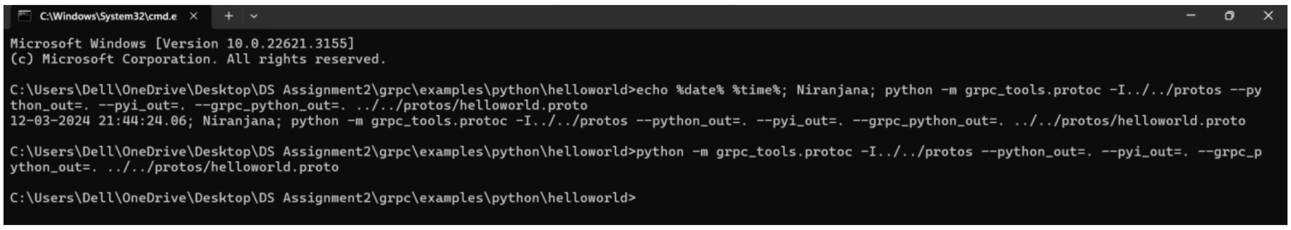
-m grpc_tools.protoc: Used to run modules

-I../../protos: This tells the compiler to look for the proto file in that directory.

--python_out=. This is for the compiler to generate the python output file in the current directory.

--grpc_python_out=. This lets the compiler to generate a gRPC python code in the current directory.

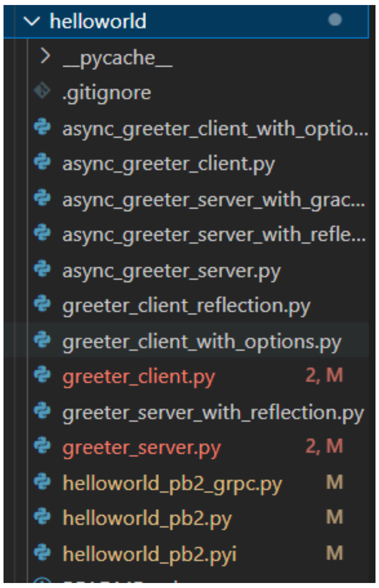
../../protos/helloworld.proto : Generates the helloworld.proto that we want to compile. Protocol Buffer files defines the structure of the code.



```
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>echo %date% %time%; Niranjana; python -m grpc_tools.protoc -I../../protos --py
thon_out=. --pyi_out=. --grpc_python_out=. ../../protos/helloworld.proto
12-03-2024 21:44:24.06; Niranjana; python -m grpc_tools.protoc -I../../protos --python_out=. --pyi_out=. --grpc_python_out=. ../../protos/helloworld.proto

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>python -m grpc_tools.protoc -I../../protos --python_out=. --pyi_out=. --grpc_p
ython_out=. ../../protos/helloworld.proto

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>
```

The above command generated two files:

1. helloworld_pb2.py : Contains the request that we just created and class corresponding to that request
2. helloworld_pb2_grpc.py: Contains the client that we generated a server class to do the further actions.



```
helloworld
  __pycache__
  .gitignore
  async_greeter_client_with_optio...
  async_greeter_client.py
  async_greeter_server_with_grac...
  async_greeter_server_with_refle...
  async_greeter_server.py
  greeter_client_reflection.py
  greeter_client_with_options.py
  greeter_client.py                2, M
  greeter_server_with_reflection.py
  greeter_server.py                2, M
  helloworld_pb2_grpc.py              M
  helloworld_pb2.py                   M
  helloworld_pb2.pyi                  M
```

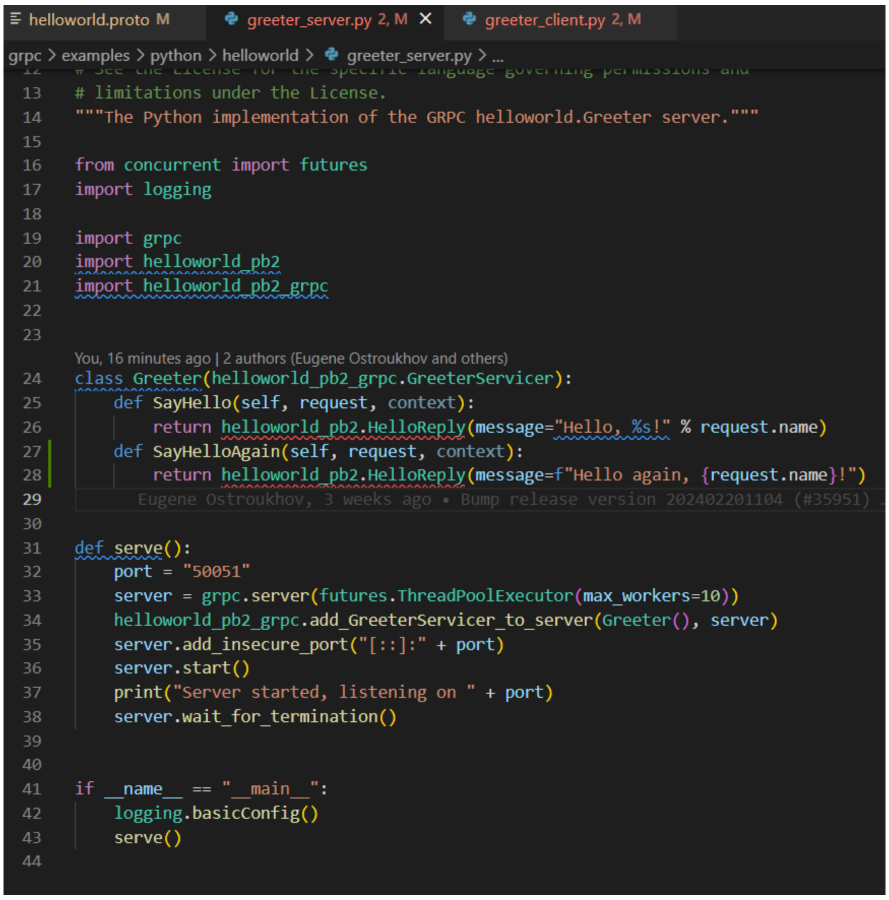
8. Since we just added a new RPC method for SayHelloAgain, we need to update the server and client with function definitions to get the desired output. Hence we update the server code and client code to make some minor additions.

greeter_server.py:

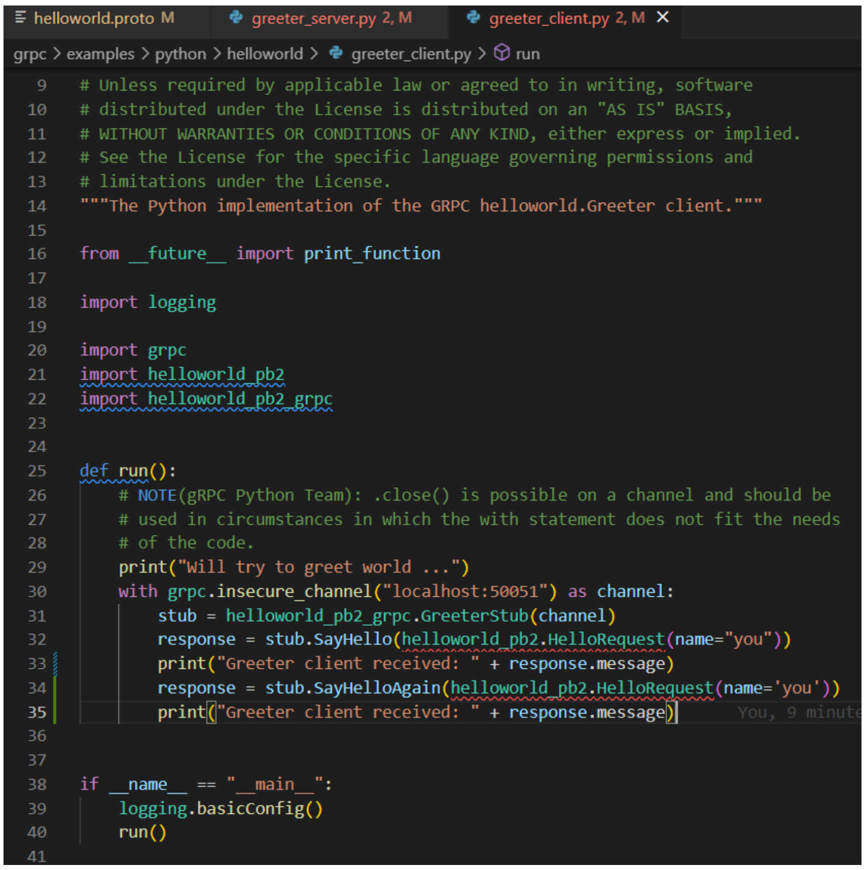
```
class Greeter(helloworld_pb2_grpc.GreeterServicer):

    def SayHello(self, request, context):
        return helloworld_pb2.HelloReply(message=f"Hello,
{request.name}!")

    def SayHelloAgain(self, request, context):
        return helloworld_pb2.HelloReply(message=f"Hello again,
{request.name}!")
...
```

greeter_client.py:

```
def run():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response =
stub.SayHello(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
        response =
stub.SayHelloAgain(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
```


helloworld.proto M
greeter_server.py 2, M
greeter_client.py 2, M
grpc > examples > python > helloworld > greeter_server.py > ...
You, 16 minutes ago | 2 authors (Eugene Ostroukhov and others)
Eugene Ostroukhov, 3 weeks ago • Bump release version 202402201104 (#35951)

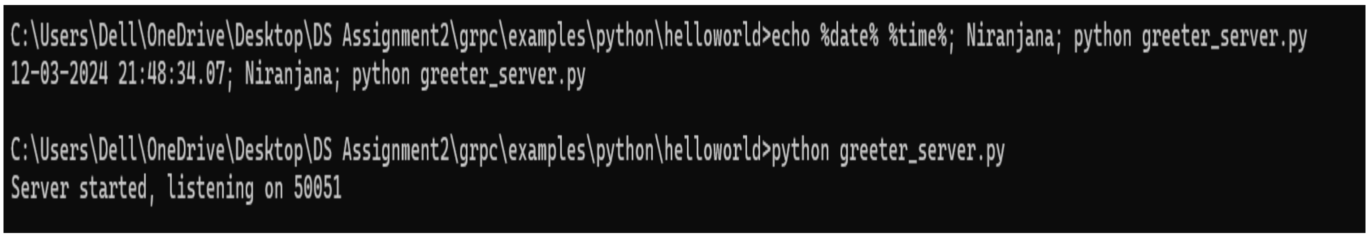
```python
# See the License for the specific language governing permissions and
# limitations under the License.
"""The Python implementation of the GRPC helloworld.Greeter server."""

from concurrent import futures
import logging

import grpc
import helloworld_pb2
import helloworld_pb2_grpc


class Greeter(helloworld_pb2_grpc.GreeterServicer):
    def SayHello(self, request, context):
        return helloworld_pb2.HelloReply(message="Hello, %s!" % request.name)
    def SayHelloAgain(self, request, context):
        return helloworld_pb2.HelloReply(message=f"Hello again, {request.name}!")


def serve():
    port = "50051"
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    helloworld_pb2_grpc.add_GreeterServicer_to_server(Greeter(), server)
    server.add_insecure_port("[::]:" + port)
    server.start()
    print("Server started, listening on " + port)
    server.wait_for_termination()


if __name__ == "__main__":
    logging.basicConfig()
    serve()
```



```python
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""The Python implementation of the GRPC helloworld.Greeter client."""

from __future__ import print_function

import logging

import grpc
import helloworld_pb2
import helloworld_pb2_grpc


def run():
    # NOTE(gRPC Python Team): .close() is possible on a channel and should be
    # used in circumstances in which the with statement does not fit the needs
    # of the code.
    print("Will try to greet world ...")
    with grpc.insecure_channel("localhost:50051") as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response = stub.SayHello(helloworld_pb2.HelloRequest(name="you"))
        print("Greeter client received: " + response.message)
        response = stub.SayHelloAgain(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)


if __name__ == "__main__":
    logging.basicConfig()
    run()
```

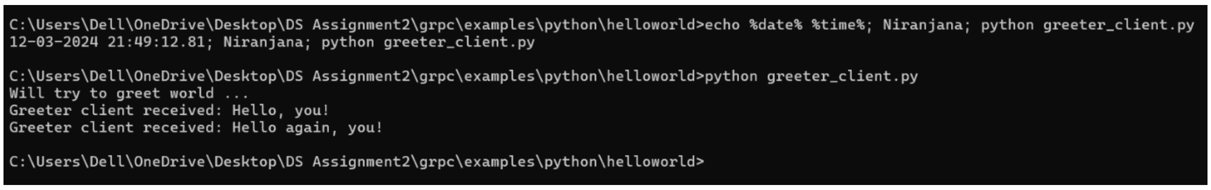
9. We run the client and server from ‘examples/python/helloworld’ directory.

python greeter_server.py



```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>echo %date% %time%; Niranjana; python greeter_server.py
12-03-2024 21:48:34.07; Niranjana; python greeter_server.py

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>python greeter_server.py
Server started, listening on 50051
```

python greeter_client.py



```
C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>echo %date% %time%; Niranjana; python greeter_client.py
12-03-2024 21:49:12.81; Niranjana; python greeter_client.py

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>python greeter_client.py
Will try to greet world ...
Greeter client received: Hello, you!
Greeter client received: Hello again, you!

C:\Users\Dell\OneDrive\Desktop\DS Assignment2\grpc\examples\python\helloworld>
```

GO IMPLEMENTATION:

1. go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.28

go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.2

The above command installs the protocol compiler plugins using Go.

2. export PATH="$PATH:$(go env GOPATH)/bin"

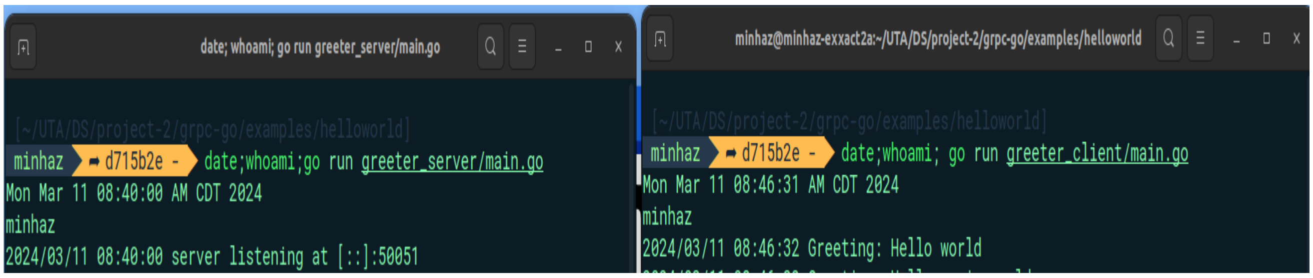
Sets the path for the protoc compiler

3. git clone -b v1.62.0 --depth 1 https://github.com/grpc/grpc-go

Clone the git for example files, set the path ‘cd grpc-go/examples/helloworld’.

4. go run greeter_server/main.go : Runs the server file.

go run greeter_client/main.go : Runs the client file.



```
date; whoami; go run greeter_server/main.go
[~/UTA/DS/project-2/grpc-go/examples/helloworld]
minhaz  d715b2e -  date;whoami;go run greeter_server/main.go
Mon Mar 11 08:40:00 AM CDT 2024
minhaz
2024/03/11 08:40:00 server listening at [::]:50051
```

```
minhaz@minhaz-exxact2a:~/UTA/DS/project-2/grpc-go/examples/helloworld
[~/UTA/DS/project-2/grpc-go/examples/helloworld]
minhaz  d715b2e -  date;whoami; go run greeter_client/main.go
Mon Mar 11 08:46:31 AM CDT 2024
minhaz
2024/03/11 08:46:32 Greeting: Hello world
```

5. // The greeting service definition.
```
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}
// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}
// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

We update the helloworld.proto file to add another function ‘SayHelloAgain’ so that the RPC proto file has a stub that it communicates with the server and client and it works based on the parameter ‘HelloRequest’ and returns reply as ‘HelloReply’.

6.
```
protoc --go_out=. --go_opt=paths=source_relative \
    --go-grpc_out=. --go-grpc_opt=paths=source_relative \
    helloworld/helloworld.proto
```

protoc: Protocol Buffers compiler command-line interface.

--go_out=.: Lets the compiler to generate Go code from the Protocol Buffers file (helloworld. proto) and place the output in the current directory.

--go_opt=paths=source_relative: Specifies how the Go code's import paths should be generated and source_relative indicates that the import paths will contain the .proto file.

--go-grpc_out=.: Lets the compiler generate gRPC-related Go code from the Protocol Buffers file and place the output in the current directory.

--go-grpc_opt=paths=source_relative: Specifies how the import paths for gRPC-related Go code should be generated.

helloworld/helloworld.proto: This is the path to the Protocol Buffers file (helloworld. proto) that you want to compile. This file defines the structure of your data.

7. Since we just added a new RPC method for SayHelloAgain, we need to update the server and client with function definitions to get the desired output. Hence, we update the server code and client code to make some minor additions.

greeter_server/main.go:

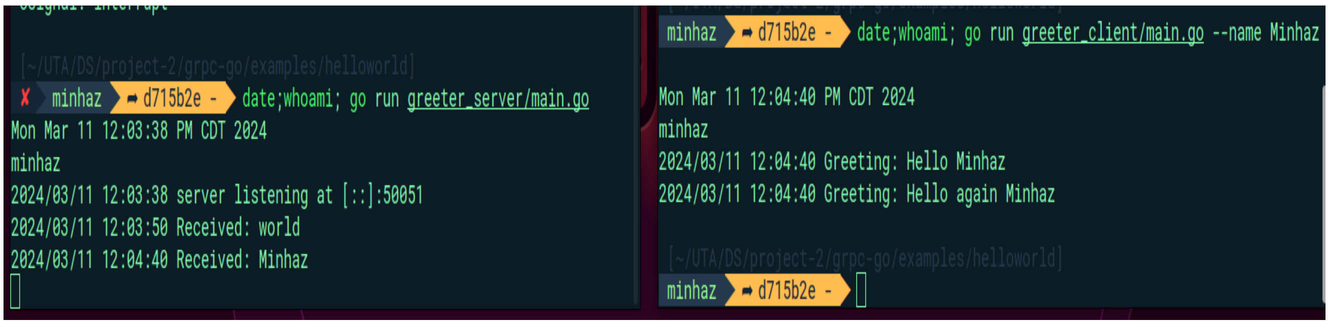
```
func (s *server) SayHelloAgain(ctx context.Context, in
*pb.HelloRequest) (*pb.HelloReply, error) {
	return &pb.HelloReply{Message: "Hello again " + in.GetName()},
nil
}
```

greeter_client/main.go:

```go
r, err = c.SayHelloAgain(ctx, &pb.HelloRequest{Name: *name})
if err != nil {
	log.Fatalf("could not greet: %v", err)
}
log.Printf("Greeting: %s", r.GetMessage())
```

8. From ‘examples/helloworld’ run the updated server and client files.

```
go run greeter_server/main.go
go run greeter_client/main.go --name=Alice
```

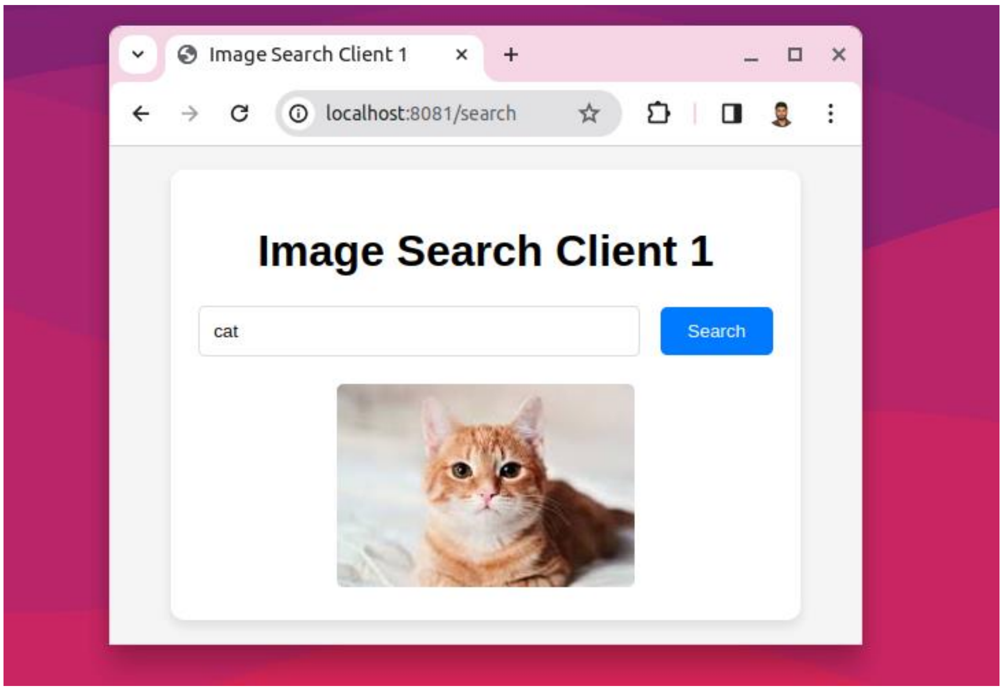


```
[~/UTA/DS/project-2/grpc-go/examples/helloworld]
✗ minhaz ➦ d715b2e - date;whoami; go run greeter_server/main.go
Mon Mar 11 12:03:38 PM CDT 2024
minhaz
2024/03/11 12:03:38 server listening at [::]:50051
2024/03/11 12:03:50 Received: world
2024/03/11 12:04:40 Received: Minhaz
```

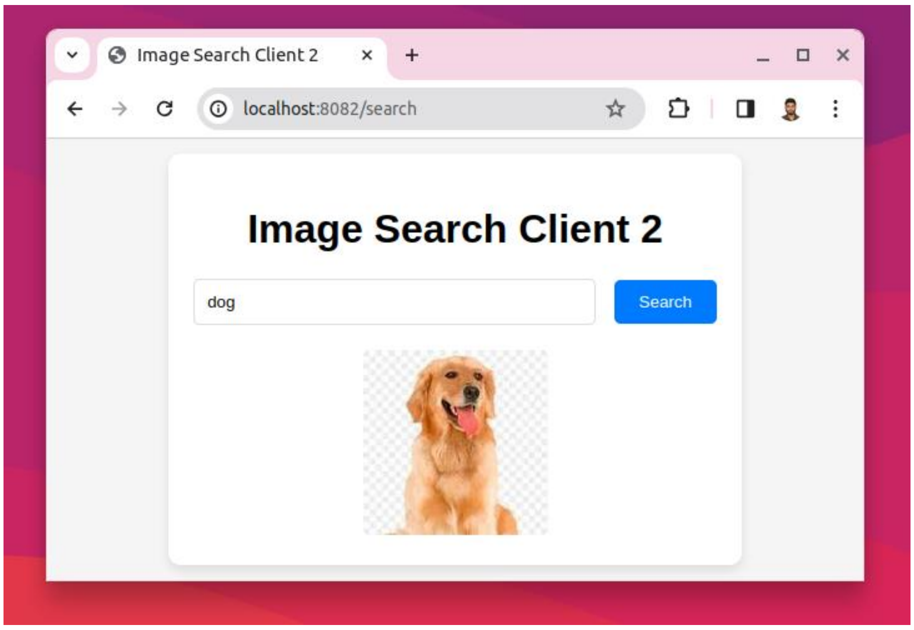
```
minhaz ➦ d715b2e - date;whoami; go run greeter_client/main.go --name Minhaz

Mon Mar 11 12:04:40 PM CDT 2024
minhaz
2024/03/11 12:04:40 Greeting: Hello Minhaz
2024/03/11 12:04:40 Greeting: Hello again Minhaz

[~/UTA/DS/project-2/grpc-go/examples/helloworld]
minhaz ➦ d715b2e -
```

Question3) Test cases for the application

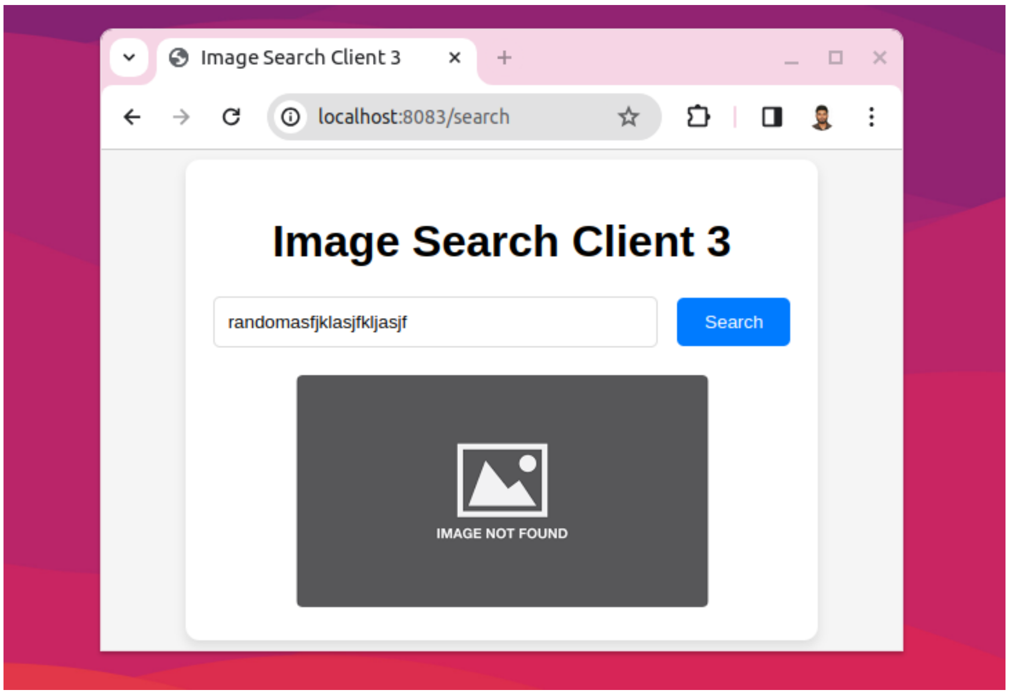
1. Search for keyword: ‘cat’
Expected Output: Cat picture



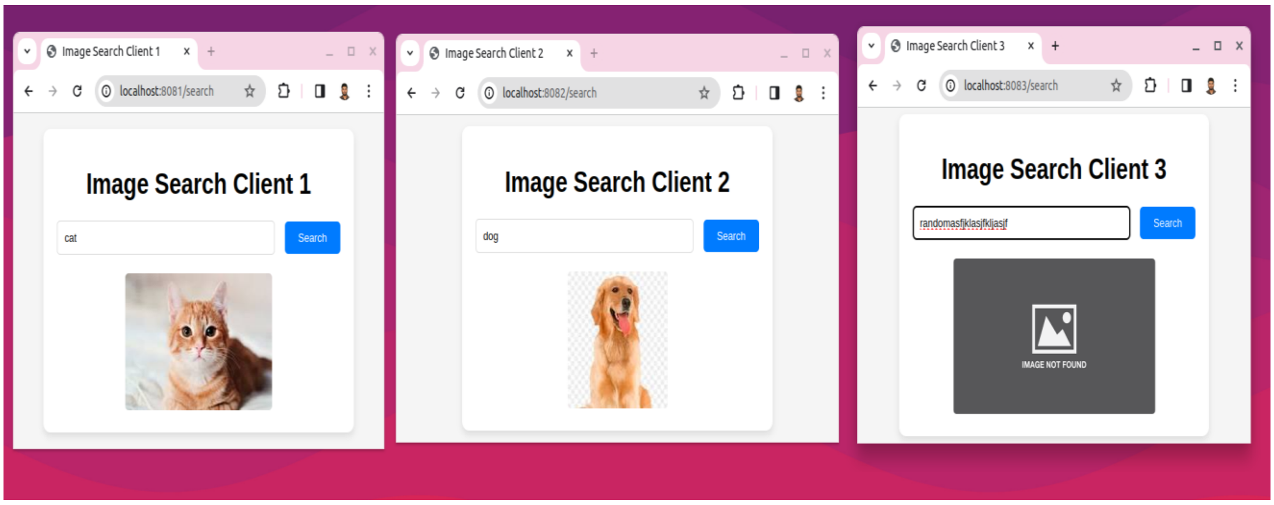
2. Search for keyword: dog
Expected Output: Dog picture



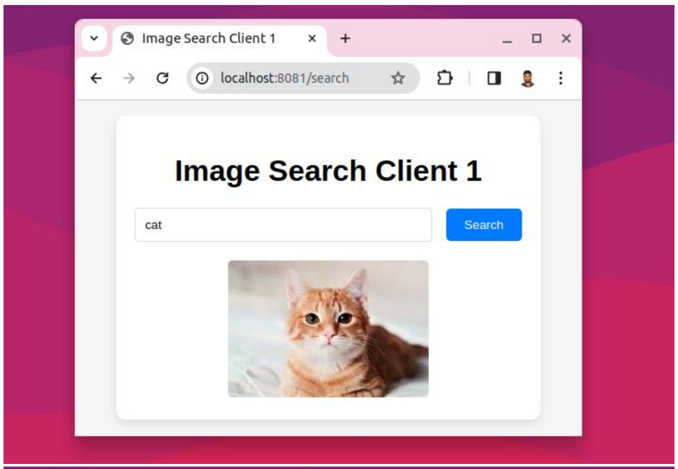
3. Search any word apart from dog, or cat.
   Expected Output: Image not found

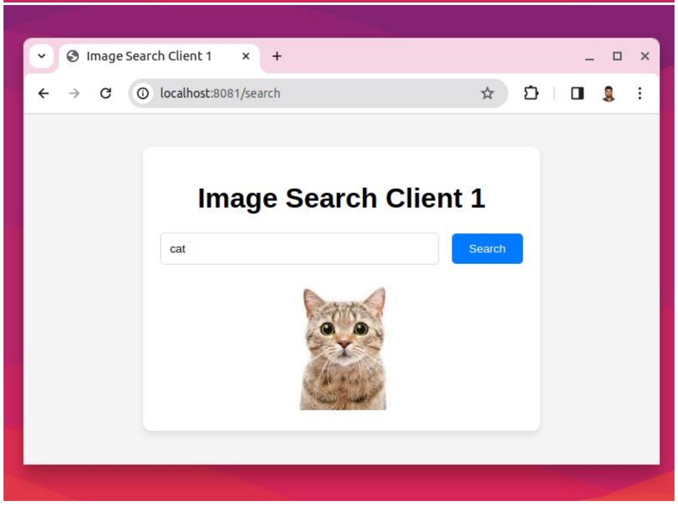


4. Test to if they work concurrently. Multiple clients requesting different keywords to the same server



5. Test to see if the images of the keywords are randomized.
   The images of keywords need to be randomized each time you input the same keyword.





Project Contribution:

Minhaz Bin Farukee: Server implementation, Go implementation for Q1, Server containerization, Networking.

Niranjana Subramanian: Client implementation, Python implementation for Q1, Client containerization, Test cases.