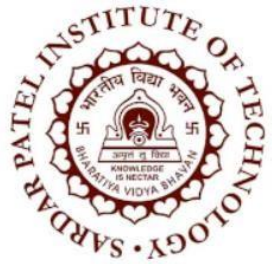


Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
An Autonomous Institute Affiliated To University Of Mumbai
Munshi Nagar,Andheri (W) Mumbai 400 058

DISTRIBUTED COMPUTING
EXPERIMENT: 2

Submitted By:

Name: Adwait Purao
UID: 2021300101
Batch: B2

Submitted To:

Prof. Pramod Bide

Aim:

Implementation of Client Server Communication using RPC

Theory:

RPC Overview:
Remote Procedure Call (RPC) is a protocol that allows programs to execute procedures or functions on a remote server as if they were local. It abstracts the complexity of network communication, making it easier for distributed systems to interact.

Components:

- **Client:** The client is the application or system that initiates a remote procedure call. In the context of an inventory management project, it might be a user interface or a script that interacts with the inventory.

- **Server:** The server is responsible for providing the services or methods that the client can call remotely. In the inventory management context, this could be the application responsible for managing the inventory database.

RPC Protocol:
RPC uses a communication protocol to serialize and transmit data between the client and server. Two commonly used protocols are:

1. **gRPC:** A high-performance RPC framework that uses Protocol Buffers (protobufs) for defining services and messages. It offers strong typing and supports multiple programming languages.

2. **JSON-RPC:** A lightweight protocol that uses JSON for data serialization. It is simpler to implement but may not be as efficient as gRPC.

Procedure Calls:

- **Method Definitions:** In the RPC implementation, we define a set of methods or procedures that can be called remotely. In an inventory management system, these methods could include operations like adding items, updating quantities, or retrieving item information.

- **Client Invocation:** The client invokes these methods as if they were local function calls. It packages the method name and parameters into a request.

- **Request Transmission:** The client sends the request to the server over a network connection. This involves serializing the request data into a format that can be transmitted (e.g., binary for gRPC, JSON for JSON-RPC).

- **Server Processing:** The server receives the request, deserializes it, and identifies the method to execute. It then executes the method, performing the requested operation on the inventory data.

- **Response Generation:** After executing the method, the server generates a response, which includes the result of the operation and any relevant data.

- **Response Transmission:** The server sends the response back to the client, again serializing it into the appropriate format.

- **Client Handling:** The client receives the response, deserializes it, and extracts the result. It can then handle the response, for example, displaying success or error messages to the user.

- **Error Handling:** RPC implementations should include error handling mechanisms to deal with network issues, server failures, and other potential problems.

- **Security:** It is crucial to implement security measures, such as authentication and encryption, to protect data integrity and prevent unauthorized access.

- **Choice of RPC Framework:** Depending on your project's requirements, you can choose an RPC framework like gRPC or implement a custom solution using JSON-RPC.

In summary, RPC in an inventory management project allows you to abstract the complexity of remote communication, enabling clients to invoke server-side methods seamlessly. The choice of RPC framework and the definition of methods will depend on your project's specific needs and technology stack.

Code:

Server:

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler


# Inventory management class
class InventoryManager:
  count = 0
  def __init__(self):
      self.inventory = {}
      self.product_id_counter = '1'


  def add_item(self, item_name, quantity):
      product_id = self.product_id_counter
      self.inventory[product_id] = {"name": item_name, "quantity":
quantity}
      count=int(self.product_id_counter)
      count+=1
      self.product_id_counter = str(count)
      return f"Product added with ID {product_id}\n"


  def update_item(self, product_id, quantity):
      if product_id in self.inventory:
          self.inventory[product_id]["quantity"] = quantity
          return f"Product with ID {product_id} quantity updated
successfully\n"
      else:
          return f"Product with ID {product_id} not found\n"


  def delete_item(self, product_id):
      if product_id in self.inventory:
          del self.inventory[product_id]
          return f"Product with ID {product_id} deleted successfully\n"
      else:
```

```
            return f"Product with ID {product_id} not found\n"


    def get_item(self, product_id):
        if product_id  in  self.inventory:
            product = self.inventory[product_id]
            return f"Product ID: {product_id}, Name: {product['name']},
Quantity: {product['quantity']}\n"
        else:
            return f"Product with ID {product_id} not found\n"

    def get_all_items(self):
        # Convert product IDs to strings for compatibility with XML-RPC
        str_inventory = {str(key): value for key, value in
self.inventory.items()}
        return str_inventory


# Create an instance of the InventoryManager class
inventory_manager = InventoryManager()


# Create a server
server = SimpleXMLRPCServer(("localhost", 9090),
requestHandler=SimpleXMLRPCRequestHandler)


# Register the InventoryManager methods for RPC
server.register_function(inventory_manager.add_item, "add_item")
server.register_function(inventory_manager.update_item, "update_item")
server.register_function(inventory_manager.delete_item, "delete_item")
server.register_function(inventory_manager.get_item, "get_item")
server.register_function(inventory_manager.get_all_items, "get_all_items")


# Start the server
print("Server is listening on port 9090...")
server.serve_forever()
```

Client:

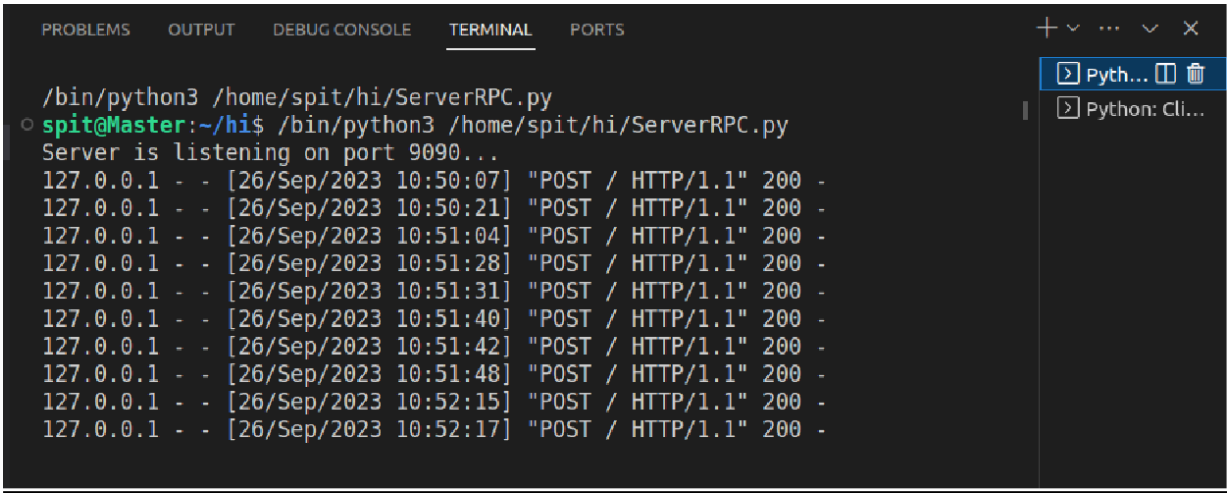
```
import xmlrpc.client


# Create an XML-RPC proxy to access the remote InventoryManager
proxy = xmlrpc.client.ServerProxy("http://localhost:9090/",
allow_none=True)


# Define client functions to interact with the server
def add_item(item_name, quantity):
  return proxy.add_item(item_name, quantity)


def update_item(item_id, quantity):
  return proxy.update_item(item_id, quantity)


def delete_item(item_id):
  return proxy.delete_item(item_id)


def get_item(item_id):
  return proxy.get_item(item_id)


def display_all_items():
  all_items = proxy.get_all_items()
  if all_items:
      print("All Items in Inventory: ")
      for product_id, product_info in all_items.items():
          print(f"Product ID: {product_id}, Name: {product_info['name']},
Quantity: {product_info['quantity']}\n")
  else:
      print("Inventory is empty.\n")


# Function to interactively manage inventory using product IDs
def interactively_manage_inventory():
```

```
while True:
    print("Choose an action:")
    print("1. Add item")
    print("2. Update item quantity")
    print("3. Delete item")
    print("4. Get item quantity")
    print("5. Display all items")
    print("6. Quit")
    print("")


    choice = input("Enter your choice: ")


    if choice == "1":
        item_name = input("Enter the item name: ")
        quantity = int(input("Enter the quantity to add: "))
        result = add_item(item_name, quantity)
        print(result)
    elif choice == "2":
        item_id = (input("Enter the product ID to update quantity: "))
        quantity = int(input("Enter the new quantity: "))
        result = update_item(item_id, quantity)
        print(result)
    elif choice == "3":
        item_id = (input("Enter the product ID to delete: "))
        result = delete_item(item_id)
        print(result)
    elif choice == "4":
        item_id = (input("Enter the product ID to get quantity: "))
        item_info = get_item(item_id)
        print(item_info)
    elif choice == "5":
        display_all_items()
    elif choice == "6":
        print("Thank You for using our Services\n")
        break
    else:
        print("Invalid choice. Please try again.")
```

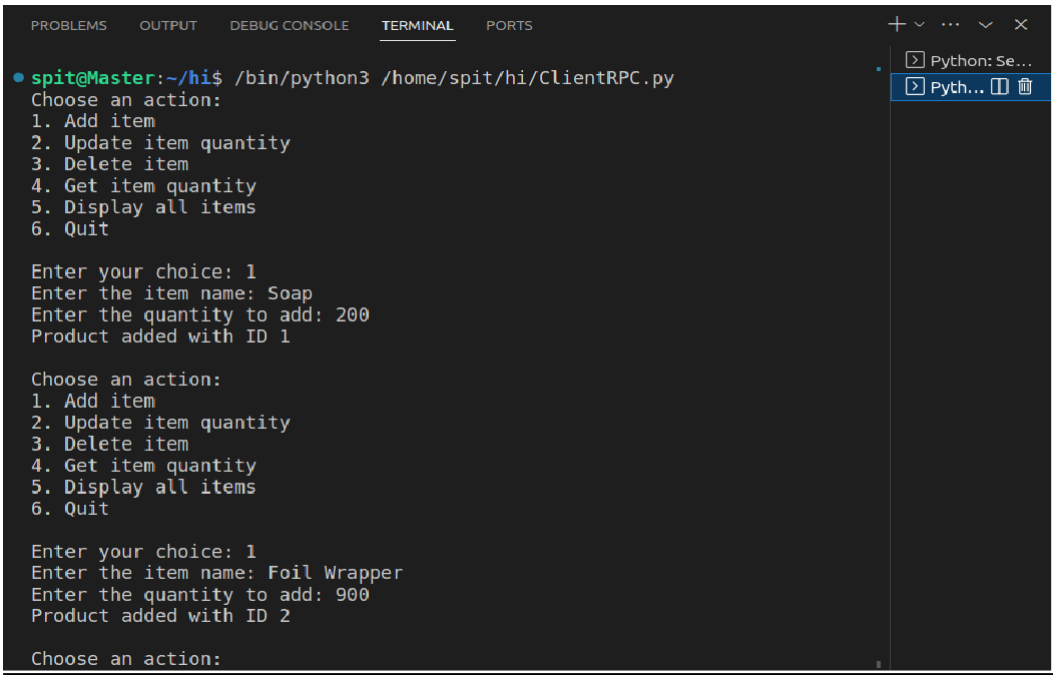
```
# Example usage:
if_name_== "_main_":
  interactively_manage_inventory()
```

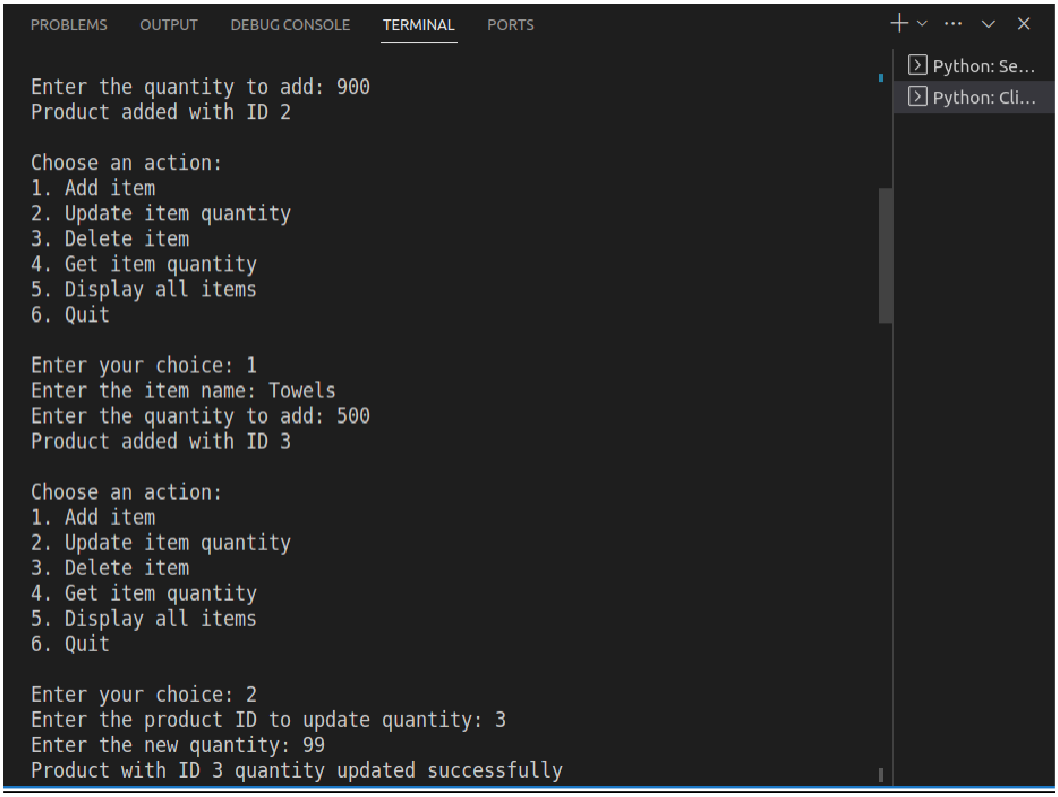
Output:

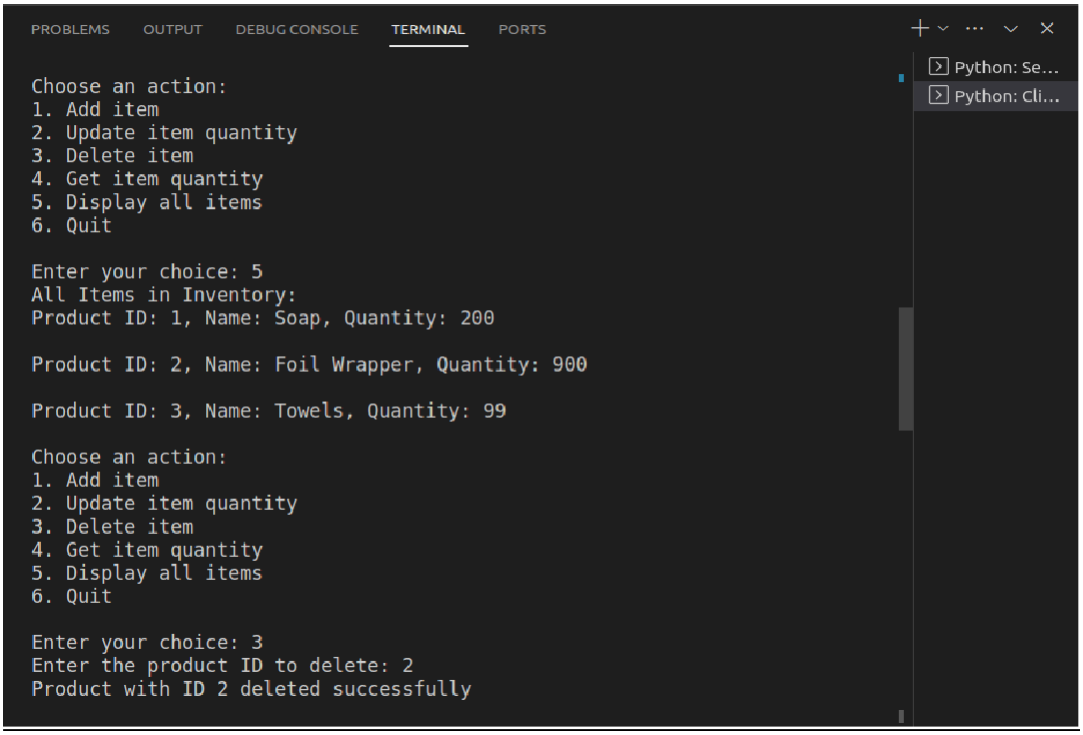
Server:

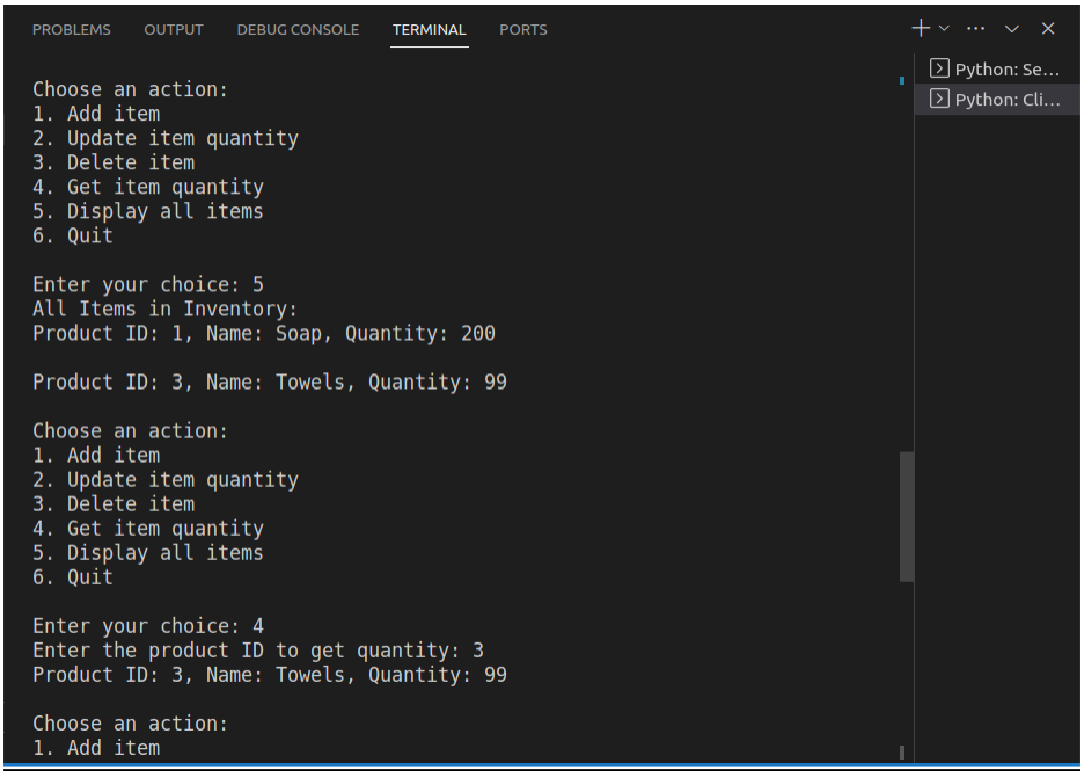


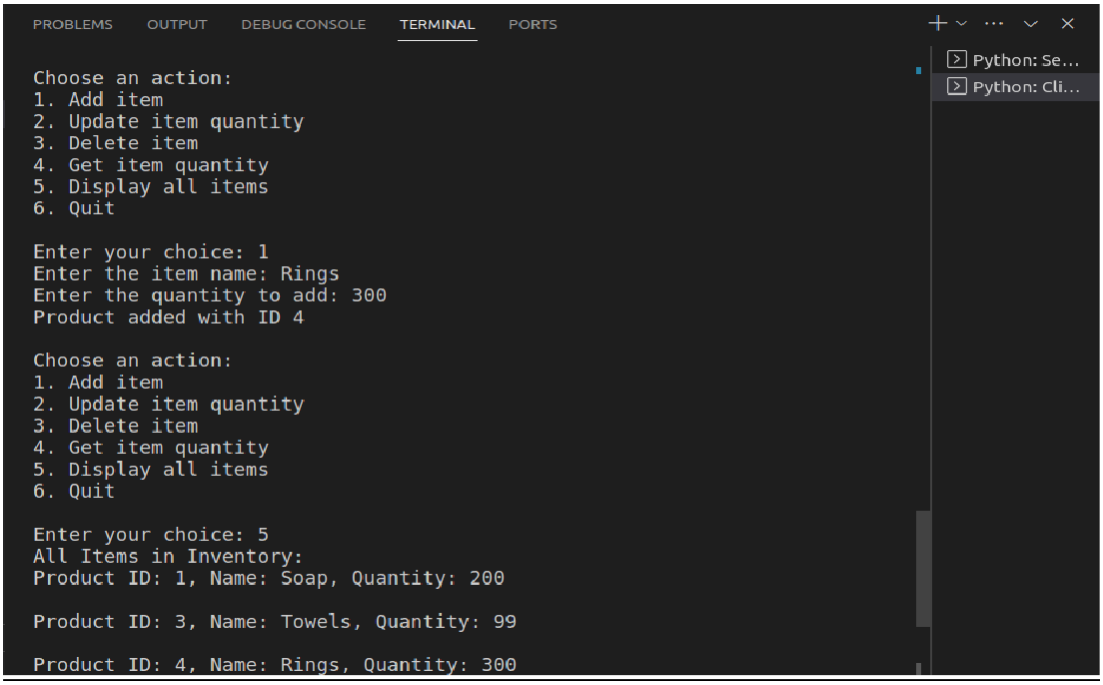
Client:

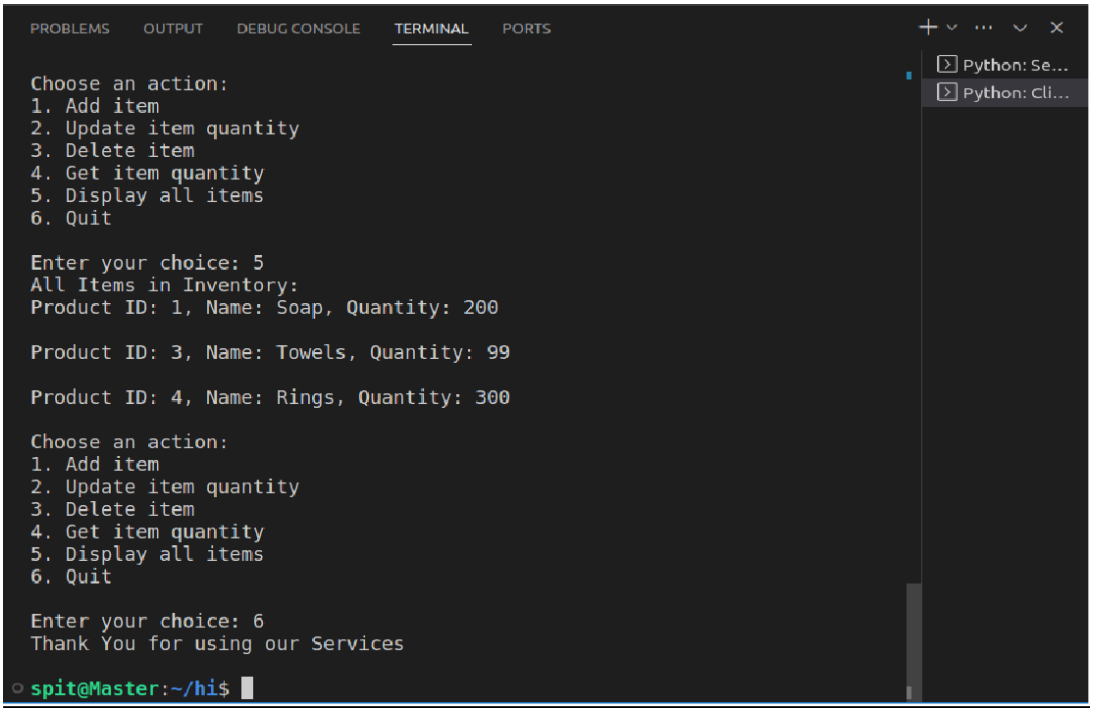

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Pyth...
spit@Master:~/hi$ /bin/python3 /home/spit/hi/ClientRPC.py
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 1
Enter the item name: Soap
Enter the quantity to add: 200
Product added with ID 1
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 1
Enter the item name: Foil Wrapper
Enter the quantity to add: 900
Product added with ID 2
Choose an action:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Python: Cli...
Enter the quantity to add: 900
Product added with ID 2
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 1
Enter the item name: Towels
Enter the quantity to add: 500
Product added with ID 3
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 2
Enter the product ID to update quantity: 3
Enter the new quantity: 99
Product with ID 3 quantity updated successfully


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Python: Cli...
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 5
All Items in Inventory:
Product ID: 1, Name: Soap, Quantity: 200
Product ID: 2, Name: Foil Wrapper, Quantity: 900
Product ID: 3, Name: Towels, Quantity: 99
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 3
Enter the product ID to delete: 2
Product with ID 2 deleted successfully


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Python: Cli...
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 5
All Items in Inventory:
Product ID: 1, Name: Soap, Quantity: 200
Product ID: 3, Name: Towels, Quantity: 99
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 4
Enter the product ID to get quantity: 3
Product ID: 3, Name: Towels, Quantity: 99
Choose an action:
1. Add item


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Python: Cli...
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 1
Enter the item name: Rings
Enter the quantity to add: 300
Product added with ID 4
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 5
All Items in Inventory:
Product ID: 1, Name: Soap, Quantity: 200
Product ID: 3, Name: Towels, Quantity: 99
Product ID: 4, Name: Rings, Quantity: 300


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: Se...
Python: Cli...
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 5
All Items in Inventory:
Product ID: 1, Name: Soap, Quantity: 200
Product ID: 3, Name: Towels, Quantity: 99
Product ID: 4, Name: Rings, Quantity: 300
Choose an action:
1. Add item
2. Update item quantity
3. Delete item
4. Get item quantity
5. Display all items
6. Quit
Enter your choice: 6
Thank You for using our Services
spit@Master:~/hi$

Conclusion:

The experiment confirmed that RPC greatly simplifies client-server communication in inventory management. gRPC and JSON-RPC offer protocol choices, with gRPC excelling in performance. RPC abstracts network complexities but demands robust error handling. Security measures are vital to protect inventory data. Customization of methods to project needs enhances system efficiency and flexibility.