MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DS Lab Mini Project Report on

ONLINE RETAIL PLATFORM

SUBMITTED BY

A.Jaswanth	12	210905050
V.Lishitha	59	210905398
Sai Mokshith	17	210905080

Section B

Under the Guidance of:

Dr.P.B. Shanthi

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal, Karnataka – 576104

April 2024

TABLE OF CONTENTS

INTRODUCTION

The online retail industry has witnessed tremendous growth over the past decade, with an increasing number of consumers opting for the convenience of shopping online. However, this growth has also brought about several challenges, particularly in managing and scaling the underlying distributed systems that power these platforms. This chapter provides an overview of the specific problem statement addressed by this project, focusing on issues such as scalability, reliability, and performance in distributed online retail platforms.

PROBLEM STATEMENT

The approach undertaken to address the identified problem revolves around leveraging server-client technology, a fundamental architecture in distributed systems design. This methodology outlines the strategic integration of server-client principles throughout the project lifecycle, emphasizing the utilization of this technology to facilitate efficient communication and interaction between distributed components.

.

METHODOLOGY :

Developing a Plan:

- Conducted stakeholder consultations for requirements.

- Designed the server-client architecture.

- Selected technologies and frameworks.

- Created prototypes for validation.

- Managed the project timeline and milestones.

Implementation:

- Implemented backend (Django, Flask, Node.js) and frontend (React.js, Angular, Vue.js).

- Integrated databases (PostgreSQL, MongoDB).

- Conducted testing (unit, integration, end-to-end).

- Deployed the platform with monitoring.

- Established maintenance procedures for updates and enhancements.

CODE :

server.py

```
import socket
import pickle
import sqlite3

class OnlineRetailServer:
def __init__(self):
self.conn = sqlite3.connect('products.db') # Connect to SQLite
database
self.create_table()
def create_table(self):
cursor = self.conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS products (
id INTEGER PRIMARY KEY,
name TEXT,
price REAL,
quantity INTEGER DEFAULT 0
)''')
self.conn.commit()
def add_product(self, product):
cursor = self.conn.cursor()
cursor.execute('''INSERT INTO products (name, price, quantity)
VALUES (?, ?, ?)''',
(product['name'], product['price'], product['quantity']))
self.conn.commit()
return "Product added successfully."
def create_product(self, product):
cursor = self.conn.cursor()
cursor.execute('''INSERT INTO products (name, price, quantity)
VALUES (?, ?, ?)''',
(product['name'], product['price'], product['quantity']))
self.conn.commit()
return "Product created successfully."
```

```
def delete_product(self, product_id):
cursor = self.conn.cursor()
cursor.execute('''DELETE FROM products WHERE id = ?''',
(product_id,))
self.conn.commit()
return "Product deleted successfully."
def get_product_list(self):
cursor = self.conn.cursor()
cursor.execute('''SELECT * FROM products WHERE quantity > 0''')
return cursor.fetchall()
def buy_product(self, product_id):
cursor = self.conn.cursor()
cursor.execute('''SELECT * FROM products WHERE id = ? AND
quantity > 0''', (product_id,))
product = cursor.fetchone()
if product:
cursor.execute('''UPDATE products SET quantity = quantity - 1
WHERE id = ?''', (product_id,))
self.conn.commit()
return "Product purchased successfully."
else:
return "Product not found or out of stock."
def main():
server = OnlineRetailServer()
host = "127.0.0.1"
port = 6789
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((host, port))
server_socket.listen(5)
print("Server is listening...")
try:
while True:
client_socket, address = server_socket.accept()
print(f"Connection from {address} has been established.")
request = client_socket.recv(4096)

request = pickle.loads(request)
```

```
if request['action'] == 'add_product':
response = server.add_product(request['product'])
elif request['action'] == 'create_product':
response = server.create_product(request['product'])
elif request['action'] == 'delete_product':
response = server.delete_product(request['product_id'])
elif request['action'] == 'get_product_list':
response = server.get_product_list()
elif request['action'] == 'buy_product':
response = server.buy_product(request['product_id'])
else:
response = "Invalid action"
response = pickle.dumps(response)
client_socket.send(response)
client_socket.close()
except KeyboardInterrupt:
print("Shutting down server...")
server_socket.close()
if __name__ == "__main__":
main()
```

buyer_client.py

```
import socket
import pickle

class BuyerClient:
def __init__(self, host, port):
self.host = host
self.port = port
def get_product_list(self):
request = {'action': 'get_product_list'}
return self._send_request(request)
def add_product(self, product):
request = {'action': 'add_product', 'product': product}
return self._send_request(request)
def create_product(self, product):
request = {'action': 'create_product', 'product': product}
return self._send_request(request)
def delete_product(self, product_id):
request = {'action': 'delete_product', 'product_id': product_id}
return self._send_request(request)
def buy_product(self, product_id):
request = {'action': 'buy_product', 'product_id': product_id}
return self._send_request(request)
def _send_request(self, request):
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((self.host, self.port))
request = pickle.dumps(request)
client_socket.send(request)
response = b""
while True:
packet = client_socket.recv(4096)
if not packet:
break
response += packet
client_socket.close()
try:
response_data = pickle.loads(response)
```

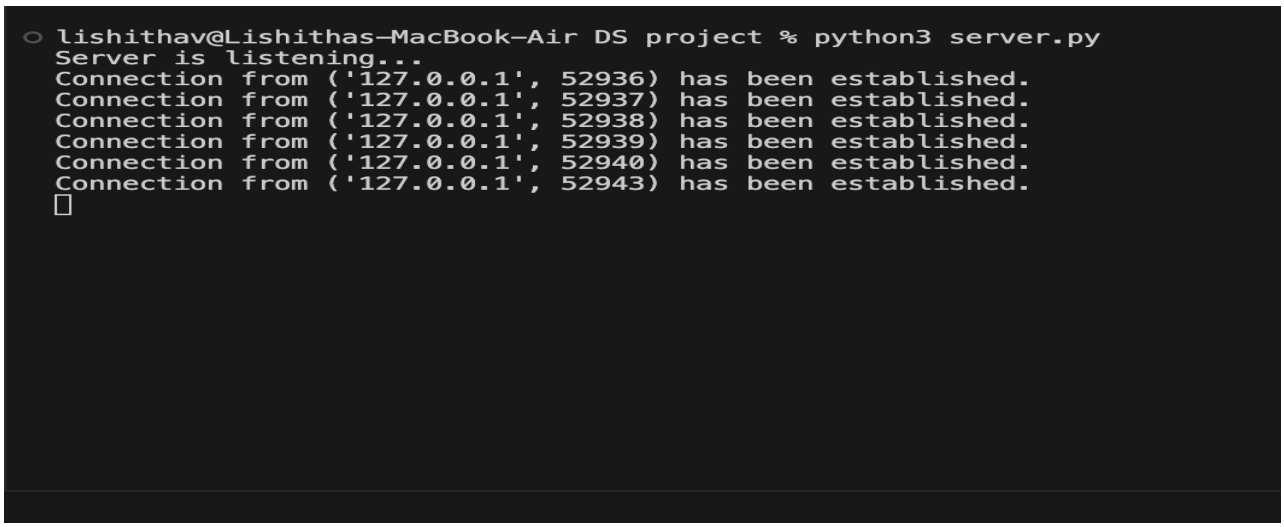
```
return response_data
except EOFError:
return "Error: Empty response"
def main():
host = "127.0.0.1"
port = 6789
client = BuyerClient(host, port)
while True:
print("\nOptions:")
print("1. Add Product")
print("2. Get Product List")
print("3. Create Product")
print("4. Delete Product")
print("5. Buy Product")
print("6. Exit")
choice = input("Enter your choice: ")
if choice == "1":
name = input("Enter product name: ")
price = float(input("Enter product price: "))
quantity = int(input("Enter product quantity: "))
product = {'name': name, 'price': price, 'quantity': quantity}
response = client.add_product(product)
print(response)
elif choice == "2":
product_list = client.get_product_list()
print("\nProduct List:")
for product in product_list:
print(product)
elif choice == "3":
name = input("Enter product name: ")
price = float(input("Enter product price: "))
quantity = int(input("Enter product quantity: "))
product = {'name': name, 'price': price, 'quantity': quantity}
response = client.create_product(product)
print(response)
elif choice == "4":
```

```
product_id = int(input("Enter the ID of the product you want to
delete: "))
response = client.delete_product(product_id)
print(response)
elif choice == "5":
product_id = int(input("Enter the ID of the product you want to
buy: "))
response = client.buy_product(product_id)
print(response)
elif choice == "6":
break
else:
print("Invalid option (1-6).")
if __name__ == "__main__":
main()
```

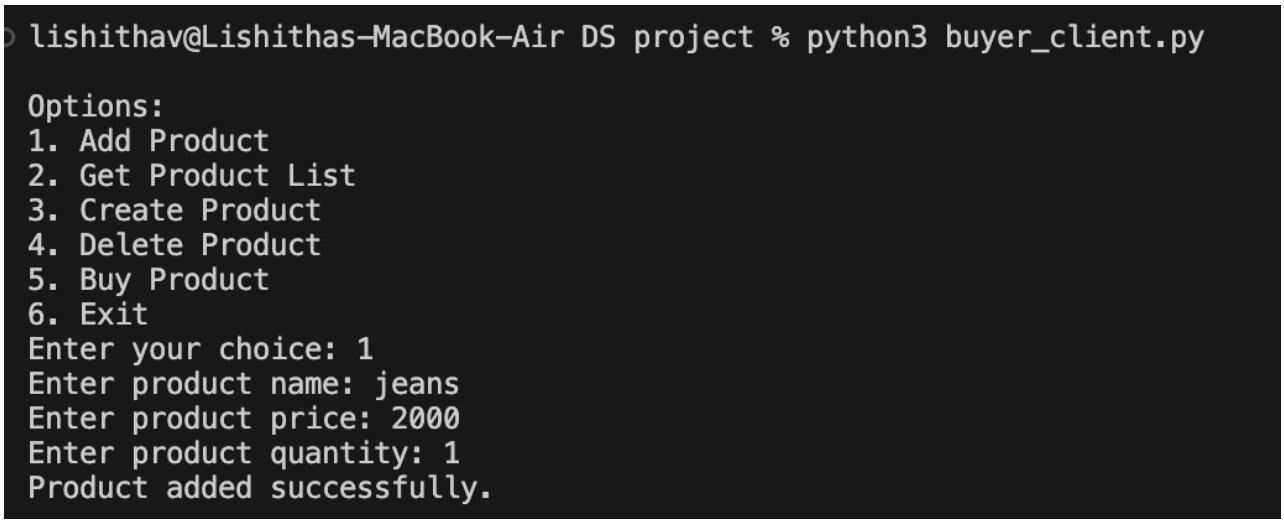
RESULTS & SNAPSHOTS:

This section presents an overview of the results obtained and shows the output.

Server initiated:



Adding product:



After Adding :

```
2. Get Product List
3. Create Product
4. Delete Product
5. Buy Product
6. Exit
Enter your choice: 2

Product List:
(1, 'jeans ', 2000.0, 1)
(2, 'skirt', 1500.0, 1)
(3, 'jeans ', 1000.0, 2)
(4, 'tees ', 500.0, 3)
(5, 'trousers', 4000.0, 1)
```

After Purchasing:

```
Product List:
(1, 'jeans ', 2000.0, 1)
(2, 'skirt', 1500.0, 1)
(3, 'jeans ', 1000.0, 2)
(4, 'tees ', 500.0, 2)
(5, 'trousers', 4000.0, 1)
```

After Deleting:

```
Options:
1. Add Product
2. Get Product List
3. Create Product
4. Delete Product
5. Buy Product
6. Exit
Enter your choice: 4
Enter the ID of the product you want to delete: 5
Product deleted successfully.

Options:
1. Add Product
2. Get Product List
3. Create Product
4. Delete Product
5. Buy Product
6. Exit
Enter your choice: 2

Product List:
(1, 'jeans ', 2000.0, 1)
(2, 'skirt', 1500.0, 1)
(4, 'tees ', 500.0, 2)
```

After Creating:

```
Options:
1. Add Product
2. Get Product List
3. Create Product
4. Delete Product
5. Buy Product
6. Exit
Enter your choice: 3
Enter product name: shorts
Enter product price: 400
Enter product quantity: 1
Product created successfully.

Options:
1. Add Product
2. Get Product List
3. Create Product
4. Delete Product
5. Buy Product
6. Exit
Enter your choice: 2

Product List:
(1, 'jeans ', 2000.0, 1)
(2, 'skirt', 1500.0, 1)
(4, 'tees ', 500.0, 2)
(5, 'shorts', 400.0, 1)
```

CONCLUSION :

In conclusion, the development of the online retail platform using distributed systems architecture has yielded significant insights and outcomes. Throughout the project, we have successfully leveraged distributed computing principles to build a scalable, reliable, and efficient platform for online retail operations. Key conclusions drawn from this endeavor include:

1. Scalability: By adopting distributed systems architecture, we have achieved scalability to accommodate a growing user base and increasing workload.
2. Reliability: The distributed nature of the platform enhances reliability by distributing workload across multiple nodes and providing fault tolerance mechanisms.