A Course Based Project Report on

# HTTP Client Server Architecture Based Hotel Booking

Submitted to the

**Department of CSE- (CyS, DS) and AI&DS**

in partial fulfilment of the requirements for the completion of course

Computer Networks and Ethical hacking LABORATORY (22PC2CY201)

BACHELOR OF TECHNOLOGY

IN

**CSE-Data Science**

Submitted by


| | |
|---|---|
| B. VISHWADA | 23071A6771 |
| B. NAMDEV | 23071A6772 |
| B. SAI HARSHITHA | 23071A6773 |
| B. RENU TANMAYI REDDY | 23071A6774 |
| B. VISESH | 23071A6775 |

Under the guidance of

**Mrs. G. Usha Rani**

**Assistant Professor**

**Department of CSE-(CyS, DS) and AI&DS**

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY**

An Autonomous Institute, NAAC Accredited with 'A++' Grade, NBA

VignanaJyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India


**December--2025**

# VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous, ISO 21001:2018& QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE, PE, AMS, SWEProgrammes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF (2024) Rank band:151-200in EngineeringCategory
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSEVignana Jyothi Nagar,
Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
Telephone No: 040-2304 2758/59/60, Fax: 040-23042761
E-mail: postbox@vnrvjiet.ac.in, Website: www.vnrvjiet.ac.in

**Department of CSE-(CyS, DS) and AI&DS**



## CERTIFICATE

This is to certify that the project report entitled **"HTTP Client Server Architecture Based Hotel Booking"** is a bonafide work done under our supervision and is being submitted by **Miss. B. Vishwada (23071A6771), Mr. B. Namdev (23071A6772), Miss. B. Sai Harshitha (23071A6773), Miss. B. Renu Tanmayi Reddy (23071A6774), Mr. B. Visesh (23071A6775)** in partial fulfillment for the award of the degree of **Bachelor of Technology** in **CSE-Data Science**, of the VNRVJIET, Hyderabad during the academic year 2025-2026.

**Mrs. G. Usha Rani**
Assistant Professor
Dept of CSE-(CyS, DS) and AI&DS

**Dr. T. Sunil Kumar**
Professor& HOD
Dept of CSE-(CyS, DS) and AI&DS

**Course based Projects Reviewer**

# VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade,
VignanaJyothi Nagar, Pragathi Nagar, Nizampet(SO), Hyderabad-500090, TS, India

**Department of CSE-(CyS, DS) and AI&DS**

## DECLARATION

We declare that the course based project work entitled **"HTTP Client Server Architecture Based Hotel Booking"** submitted in the Department of **CSE-(CyS, DS) and AI&DS**, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in CSE-Data Science** is a bonafide record of our own work carried out under the supervision of **G. Usha Rani, Assistant Professor, Department of CSE-(CyS, DS) and AI&DS, VNRVJIET.** Also, we declare that the matter embodied in this thesis has not been submitted by us in full or in any part thereof for the award of any degree/diploma of any other institution or university previously.

Place: Hyderabad.

| | |
|---|---|
| B. VISHWADA | (23071A6771) |
| B. NAMDEV | (23071A6772) |
| B. SAI HARSHITHA | (23071A6773) |
| B. RENU TANMAYI REDDY | (23071A6774) |
| B. VISESH | (23071A6775) |

# ACKNOWLEDGEMENT

We express our deep sense of gratitude to our beloved President, Sri. D. Suresh Babu, VNR Vignana Jyothi Institute of Engineering & Technology for the valuable guidance and for permitting us to carry out this project.

With immense pleasure, we record our deep sense of gratitude to our beloved Principal, Dr. C.D Naidu, for permitting us to carry out this project.

We express our deep sense of gratitude to our beloved Professor **Dr. T. Sunil Kumar**, Professor and Head, Department of CSE-(CyS, DS) and AI&DS, VNR Vignana Jyothi Institute of Engineering & Technology, Hyderabad-500090 for the valuable guidance and suggestions, keen interest and through encouragement extended throughout the period of project work.

We take immense pleasure to express our deep sense of gratitude to our beloved Guide, Mrs. **G. Usha Rani**, Assistant Professor in CSE-(CyS, DS) and AI&DS, VNR Vignana Jyothi Institute of Engineering & Technology, Hyderabad, for his/her valuable suggestions and rare insights, for constant source of encouragement and inspiration throughout my project work.

We express our thanks to all those who contributed for the successful completion of our project work.

| | |
|---|---|
| B. VISHWADA | (23071A6771) |
| B. NAMDEV | (23071A6772) |
| B. SAI HARSHITHA | (23071A6773) |
| B. RENU TANMAYI REDDY | (23071A6774) |
| B. VISESH | (23071A6775) |

# TABLE OF CONTENTS

# ABSTRACT

The “HTTP Client-Server Architecture Based Hotel Booking System” is a Python-based web application developed to demonstrate the principles of client-server communication over the Hypertext Transfer Protocol (HTTP). The system enables clients to interact with a centralized hotel management server for operations such as room booking, check-in/check-out, and status queries through a browser interface or HTTP requests. By utilizing Python’s built-in libraries such as http.server, socketserver, and requests, the project illustrates the seamless exchange of structured data between client and server using standard web protocols.

The application adopts a RESTful architecture where the server handles HTTP methods like GET and POST to process client requests efficiently. Each client communicates with the server through HTTP messages, and responses are generated dynamically based on availability and booking data. This approach provides a practical understanding of how modern web-based systems manage concurrent requests and maintain reliable communication.

The system can be further enhanced with features such as authentication, database integration for persistent data storage, and HTTPS for secure data transfer. It serves as an excellent demonstration of web communication models, highlighting the scalability and modularity of HTTP-based networking. Overall, this project showcases how the client-server model forms the backbone of modern web applications and provides an interactive framework for learning distributed computing and network communication concepts.

# CHAPTER-1

# INTRODUCTION

In today's digital world, online booking systems have become essential for managing hospitality services efficiently. From reserving rooms to handling check-ins and check-outs, the need for fast and reliable communication between clients and servers is vital. The **HTTP Client-Server Architecture Based Hotel Booking System** is developed to demonstrate how such interactions can be achieved through the **Hypertext Transfer Protocol (HTTP)** using Python. This system allows clients to communicate with a central server to perform hotel-related operations through simple HTTP requests and responses, making the booking process more structured and automated.

The system utilizes Python's built-in libraries such as http.server, socketserver, and requests to establish seamless communication between client and server. Clients can send requests for booking, query their reservations, or perform check-out operations, and the server responds accordingly in real time. The implementation follows a **stateless communication model**, ensuring each request is processed independently for reliability and scalability. Additionally, the system can be easily extended with features like database connectivity and HTTPS for secure data transmission.

This project emphasizes the importance of understanding **client-server architecture**, **HTTP communication**, and **network-based application development**. It helps learners gain practical experience with web protocols and real-world communication models while showcasing Python's ability to create lightweight and efficient networking solutions. Overall, the **Hotel Booking System** demonstrates how HTTP-based communication can be applied effectively in automating and managing booking operations in a simple and secure manner.

# CHAPTER-2

# METHODOLOGY

### 2.1. Project Architecture

The project follows a **client-server model** utilizing the HTTP protocol for communication.

- The **server** hosts APIs for hotel room management (booking, update, query, checkout) using Python's http.server library.
- The **client** interacts with the server through HTTP requests (using the requests library), providing a command-line interface for users.

### 2.2. Technologies Used

- **Python 3:** Programming language for both server and client logic.
- **http.server:** For setting up the server-side HTTP endpoints.
- **requests:** For handling HTTP calls on the client side.
- **hashlib:** For secure password hashing and authentication.

### 2.3. Features Implemented

- **Room Management:** The server keeps track of available and occupied rooms. Clients can book, query, update, or checkout rooms.
- **Authentication:** User credentials are immediately verified at startup, with passwords securely hashed on the server before storage or validation; no actions are allowed until successful authentication.
- **HTTP Methods:** The system demonstrates usage of all major HTTP methods (GET, POST, PUT, DELETE).
    - GET: Retrieve room status.
    - POST: Book rooms, query bookings.
    - PUT: Update existing bookings.
    - DELETE: Checkout and release booked rooms.

### 2.4. Workflow

- **Server Initialization:** The server is initialized and listens for requests on http://127.0.0.1:8000.

- **Client Startup:** The client script prompts the user for username and password, verifies them immediately with the server's /query endpoint.
- **Client Operations:** Upon successful login or new registration, the user may select actions:
    - **View rooms** (availability status)
    - **Book rooms**
    - **Check booked rooms**
    - **Update booking**
    - **Checkout and release rooms**
- **Authentication Handling:**
    - Passwords are never stored in plaintext—SHA-256 hashes are computed and saved/checked on the server.
    - On invalid password, the client cleanly exits.
- **Room Assignment:**
    - Booking assigns available rooms.
    - Update changes assigned room for a user if requested.
    - Checkout releases rooms, making them available again.

### 2.5. Security Considerations

- **Password Security:** All passwords are hashed with SHA-256 before storage and during checks, preventing direct exposure of user credentials.
- **Error Handling:** Responses include clear error messages and status codes for invalid operations (authentication failure, unavailable rooms, etc.).
- **Session Safety:** Authentication is checked at every operation; actions only permitted for correctly authenticated users.

### 2.6. Demonstrated Concepts

- **HTTP Client-Server Communication:** The entire system models the RESTful approach.
- **Authentication Mechanisms:** Demonstration of basic, secure login.
- **Stateful Server Operations:** Reservation, room management, and user tracking.
- **Secure Error Handling and Controlled Exit in Client:** Stops interaction on invalid credentials.

# CHAPTER-3
# Code

**Hotel_client.py**

```
import requests
import json
import sys

BASE = "http://127.0.0.1:8000"

def print_rooms():
    r = requests.get(f"{BASE}/rooms")
    data = r.json()
    print("\nRooms status:")
    for r in data.get("rooms", []):
        status = "Occupied" if r["is_occupied"] else "Free"
        print(f" Room {r['room_number']}: {status}")
    print()

def interactive():
    print("=== Hotel Booking Client (HTTP + Auth Demo) ===")
    username = input("Username: ").strip()
    password = input("Password: ").strip()

    # -------- Immediate authentication check -----------
    payload = {"username": username, "password": password}
    resp = requests.post(f"{BASE}/query", json=payload)
    data = resp.json()
    # Allow for 'User not found' to register new user lazily by booking later
    if "error" in data and data["error"] == "Invalid password.":
        print("Invalid password. Exiting.")
        sys.exit(1)

    while True:
        print("\nActions: rooms | book | query | update | checkout | exit")
        cmd = input("Action: ").strip().lower()
        if cmd == "rooms":
            print_rooms()
        elif cmd == "book":
            try:
                n = int(input("Number of rooms to book: "))
            except ValueError:
                print("Enter a valid integer.")
                continue
            payload = {"username": username, "password": password, "num_rooms": n}
            r = requests.post(f"{BASE}/book", json=payload)
            print(r.json())
        elif cmd == "query":
```

```
            payload = {"username": username, "password": password}
            r = requests.post(f"{BASE}/query", json=payload)
            print(r.json())
        elif cmd == "update":
            try:
                new_room = int(input("Enter new room number to assign: "))
            except ValueError:
                print("Enter a valid integer.")
                continue
            payload = {"username": username, "password": password, "new_room": new_room}
            r = requests.put(f"{BASE}/update", json=payload)
            print(r.json())
        elif cmd == "checkout":
            payload = {"username": username, "password": password}
            r = requests.delete(f"{BASE}/checkout", json=payload)
            print(r.json())
        elif cmd == "exit":
            print("Goodbye.")
            break
        else:
            print("Unknown action.")

if __name__ == "__main__":
    interactive()
```

**hotel_server.py**

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import json
import urllib.parse
import hashlib  # <--- Added for password hashing
HOST = "127.0.0.1"
PORT = 8000
MAX_ROOMS = 10
rooms = [{"room_number": i + 1, "is_occupied": False} for i in range(MAX_ROOMS)]
user_info = {}
# ---- Password Hashing Function ----
def hash_password(password):
    return hashlib.sha256(password.encode("utf-8")).hexdigest()
def assign_rooms_to_user(username, password, room_numbers):
    hashed = hash_password(password)  # <--- Store hashed password
    if username not in user_info:
```

```
        user_info[username] = {"password": hashed, "room_numbers": []}
    user_info[username]["room_numbers"].extend(room_numbers)


def book_rooms(num_rooms, username, password):
    if username in user_info and user_info[username]["password"] !=
hash_password(password):  # <--- Check hashed
        return {"error": "Invalid password."}, 401
    booked = []
    for room in rooms:
        if not room["is_occupied"]:
            room["is_occupied"] = True
            booked.append(room["room_number"])
            if len(booked) == num_rooms:
                break
    if not booked:
        return {"error": "No rooms available."}, 409
    assign_rooms_to_user(username, password, booked)  # always uses hashed inside
    return {"message": f"Rooms booked successfully: {booked}"}, 200
def checkout_rooms(username, password):
    if username not in user_info:
        return {"error": "User not found."}, 404
    if user_info[username]["password"] != hash_password(password):  # <--- Check hashed
        return {"error": "Invalid password."}, 401
    booked = user_info[username].get("room_numbers", [])
    if not booked:
        return {"message": "You have no booked rooms."}, 400
    for rn in booked:
        for room in rooms:
            if room["room_number"] == rn:
                room["is_occupied"] = False
    user_info[username]["room_numbers"] = []
    return {"message": f"Checked out successfully. Released rooms: {booked}"}, 200
def query_rooms(username, password):
    if username not in user_info:
```

```
        return {"error": "User not found."}, 404
    if user_info[username]["password"] != hash_password(password):  # <--- Check hashed
        return {"error": "Invalid password."}, 401
    booked = user_info[username].get("room_numbers", [])
    if booked:
        return {"message": f"Your booked rooms: {booked}"}, 200
    else:
        return {"message": "You have not booked any rooms."}, 200
def update_booking(username, password, new_room):
    if username not in user_info:
        return {"error": "User not found."}, 404
    if user_info[username]["password"] != hash_password(password):  # <--- Check hashed
        return {"error": "Invalid password."}, 401
    if not (1 <= new_room <= MAX_ROOMS):
        return {"error": "Invalid room number."}, 400
    # check occupancy
    for room in rooms:
        if room["room_number"] == new_room and room["is_occupied"]:
            if new_room in user_info[username]["room_numbers"]:
                return {"message": f"You already have room {new_room}."}, 200
            return {"error": f"Room {new_room} is already occupied."}, 409
    user_rooms = user_info[username].get("room_numbers", [])
    if not user_rooms:
        return {"error": "You have no booking to update."}, 400
    old_room = user_rooms.pop(0)
    for room in rooms:
        if room["room_number"] == old_room:
            room["is_occupied"] = False
    for room in rooms:
        if room["room_number"] == new_room:
            room["is_occupied"] = True
            break
    user_info[username]["room_numbers"].append(new_room)
    return {"message": f"Updated booking: released {old_room}, assigned {new_room}."},
```

200

```
class HotelHTTPRequestHandler(BaseHTTPRequestHandler):
    def _send_json(self, obj, status=200):
        body = json.dumps(obj).encode("utf-8")
        self.send_response(status)
        self.send_header("Content-Type", "application/json")
        self.send_header("Content-Length", str(len(body)))
        self.end_headers()
        self.wfile.write(body)

    def do_GET(self):
        parsed = urllib.parse.urlparse(self.path)
        if parsed.path == "/rooms":
            status = [{"room_number": r["room_number"], "is_occupied": r["is_occupied"]} for
r in rooms]
            self._send_json({"rooms": status}, 200)
        else:
            self._send_json({"error": "Endpoint not found."}, 404)

    def do_POST(self):
        parsed = urllib.parse.urlparse(self.path)
        content_length = int(self.headers.get("Content-Length", 0))
        raw = self.rfile.read(content_length).decode("utf-8")
        try:
            data = json.loads(raw)
        except json.JSONDecodeError:
            self._send_json({"error": "Invalid JSON."}, 400)
            return
        username = data.get("username")
        password = data.get("password")
        if parsed.path == "/book":
            num_rooms = int(data.get("num_rooms", 0))
            if not username or not password or num_rooms <= 0:
```

```
            self._send_json({"error": "username, password, and num_rooms required."}, 400)
            return
        result, status = book_rooms(num_rooms, username, password)
        self._send_json(result, status)
    elif parsed.path == "/query":
        if not username or not password:
            self._send_json({"error": "username and password required."}, 400)
            return
        result, status = query_rooms(username, password)
        self._send_json(result, status)
    else:
        self._send_json({"error": "Endpoint not found."}, 404)
def do_PUT(self):
    parsed = urllib.parse.urlparse(self.path)
    if parsed.path != "/update":
        self._send_json({"error": "Endpoint not found."}, 404)
        return
    content_length = int(self.headers.get("Content-Length", 0))
    raw = self.rfile.read(content_length).decode("utf-8")
    try:
        data = json.loads(raw)
    except json.JSONDecodeError:
        self._send_json({"error": "Invalid JSON."}, 400)
        return
    username = data.get("username")
    password = data.get("password")
    try:
        new_room = int(data.get("new_room"))
    except (TypeError, ValueError):
        self._send_json({"error": "new_room must be integer."}, 400)
        return
    if not username or not password:
        self._send_json({"error": "username and password required."}, 400)
        return
```

```
            result, status = update_booking(username, password, new_room)
            self._send_json(result, status)

        def do_DELETE(self):
            parsed = urllib.parse.urlparse(self.path)
            if parsed.path != "/checkout":
                self._send_json({"error": "Endpoint not found."}, 404)
                return
            content_length = int(self.headers.get("Content-Length", 0))
            raw = self.rfile.read(content_length).decode("utf-8")
            try:
                data = json.loads(raw)
            except json.JSONDecodeError:
                self._send_json({"error": "Invalid JSON."}, 400)
                return
            username = data.get("username")
            password = data.get("password")
            if not username or not password:
                self._send_json({"error": "username and password required."}, 400)
                return
            result, status = checkout_rooms(username, password)
            self._send_json(result, status)

    def run_server():
        print(f"Starting HTTP Hotel Server at http://{HOST}:{PORT}")
        server = HTTPServer((HOST, PORT), HotelHTTPRequestHandler)
        try:
            server.serve_forever()
        except KeyboardInterrupt:
            print("\nServer stopping...")
            server.server_close()

    if __name__ == "__main__":
        run_server()
```

# CHAPTER-4

# TEST CASES/ OUTPUT

To ensure the correct functionality of the HTTP Client Server Architecture Based Hotel Booking, several test cases were designed and executed. Each test case verifies a specific functionality of the system such as authentication, booking, querying, updating, and checking out.

The interaction between the client and server is carried out over the HTTP protocol using various methods such as GET, POST, PUT, and DELETE. Outputs are verified through console responses on both client and server sides, ensuring accurate communication, proper data handling, and correct status code generation.

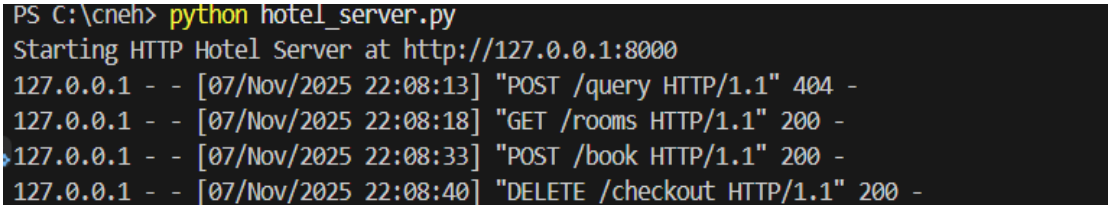


**Fig 1– Server page**

- Server actively listens for HTTP requests, displaying request logs and response statuses.

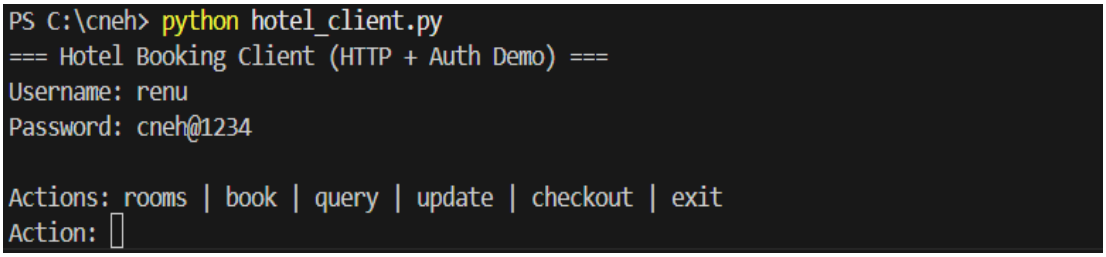


**Fig 2– Client page**

- The client is prompted to enter a username and password.
- Upon authentication, it displays the available actions menu (rooms | book | query | update | checkout | exit)

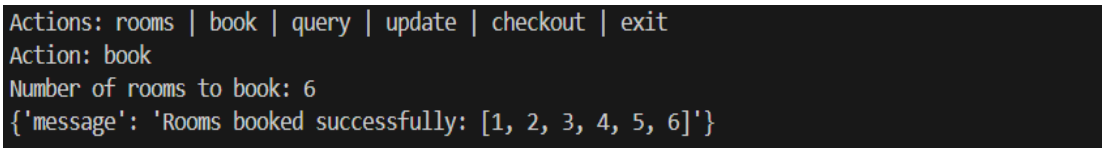


**Fig 3– Client books rooms**

- User books rooms via POST /book request.
- Server responds with confirmation and assigned room numbers.

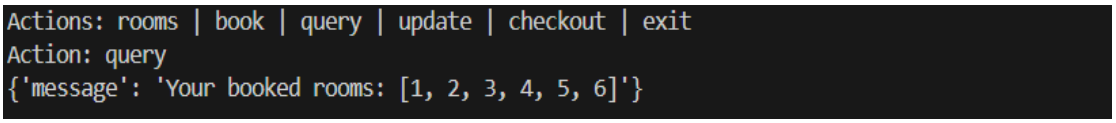


**Fig 4 – Client Raises Room Query**

- User sends POST /query request to view booked rooms.
- Server returns a list of currently booked rooms.


**Fig 5 – Check Room Status**

- User retrieves all room statuses through GET /rooms request.
- Displays which rooms are free or occupied.

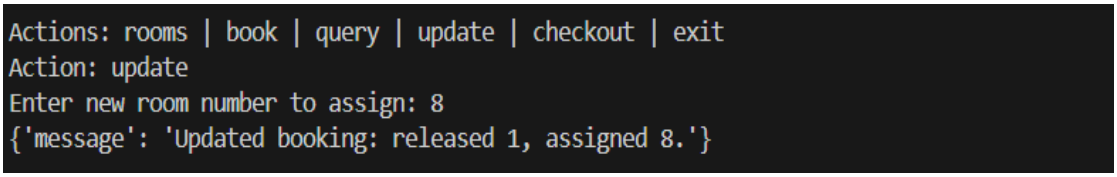


**Fig 6 – Update a Room**

- User changes an existing booking using PUT /update request.
- Server releases the old room and assigns a new one.

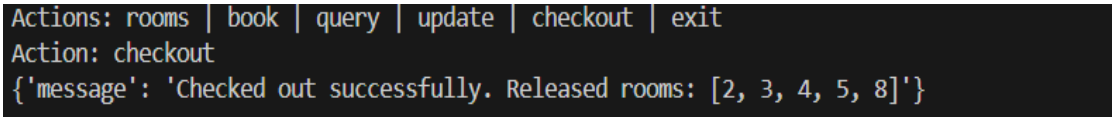


**Fig 7 – Checkout**

- User checks out using DELETE /checkout request.
- Server releases all booked rooms and confirms checkout.

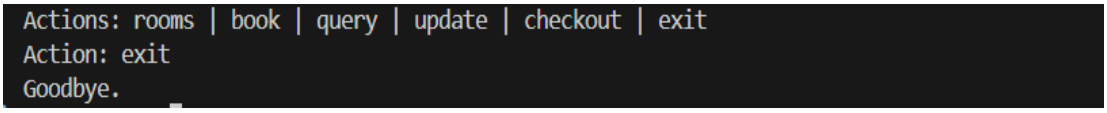


**Fig 8 – Exit**

- Client ends the session gracefully and disconnects from the server.

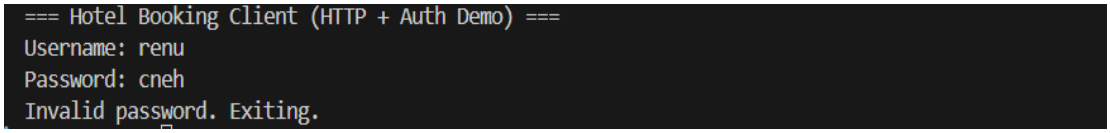


**Fig 9 – Incorrect Password (Authentication Failure)**

- When an invalid password is entered, the server responds with an authentication error
- Client displays an “Invalid password” message

# CHAPTER-5

# RESULTS

The HTTP Client-Server Architecture Based Hotel Booking System was successfully developed and executed, showcasing seamless two-way communication between multiple clients and the server over the HTTP protocol. All major operations—room booking, status checking, record updates, and checkout—were efficiently processed with instant responses displayed on both the client and server sides.

The system accurately handled multiple client requests simultaneously while preserving the stateless nature of HTTP. Each request was processed independently, ensuring consistent behavior and response accuracy. Proper error-handling mechanisms were implemented to manage invalid inputs, incorrect passwords, and unsupported operations, which helped maintain the system's robustness and reliability.

The test cases confirmed the correctness of the system under various conditions, including successful transactions, authentication failures, and invalid requests. The output screens verified smooth execution of every feature, demonstrating that the system can effectively simulate real-world client-server communication for hotel management.

Overall, the project validated the practical application of HTTP-based networking concepts, emphasizing reliability, scalability, and real-time performance in distributed environments.

# CHAPTER-6

# CONCLUSION

The **HTTP Client-Server Architecture Based Hotel Booking System** successfully demonstrates how Python can be used to implement web-based communication between clients and servers through the **Hypertext Transfer Protocol (HTTP)**. By utilizing Python's built-in libraries such as http.server, socketserver, and requests, the system effectively simulates real-time interactions for operations like room booking, check-in, and query handling. During testing, the communication model proved to be reliable, efficient, and responsive in handling multiple client requests while maintaining the stateless nature of HTTP transactions.

This project not only achieves its objective of modeling a functional hotel booking system but also provides a strong foundation for developing more advanced web-based applications. It can be further extended with features such as database connectivity for persistent data storage, HTTPS for secure communication, and a graphical or web interface for enhanced user experience. Through this implementation, the importance of **HTTP protocols, client-server design principles, and network-based resource management** is clearly illustrated.

In conclusion, the **Hotel Booking System** stands as a practical example of how the client-server model can be efficiently applied in modern application development. It highlights Python's versatility and simplicity in creating reliable, scalable, and secure networking solutions. Overall, this project bridges theoretical networking concepts with real-world implementation, showcasing how HTTP-driven communication can streamline service management and improve user accessibility in the hospitality domain.

# REFERENCES


[1]. Chatgpt – www.chatgpt.com

[2]. GeeksforGeeks. Python HTTP Server – https://www.geeksforgeeks.org/python-http-server/ .

[3]. Python Software Foundation – https://docs.python.org/3/library/http.server.html

[4]. Real Python.*Socket Programming in Python (Guide)* – https://realpython.com/python-sockets/

[5]. Tutorials Point. *Python – HTTP Client and Server Communication* – https://www.tutorialspoint.com/python_network_programming/python_http_client.htm

[6]. W3Schools. *HTTP Methods and Client-Server Communication* – https://www.w3schools.com/tags/ref_httpmethods.asp