Hotel Management System

A PROJECT REPORT

Submitted by

Yug	22BCS14252
Tarun	22BCS12086
Sajan	22BCS16496
Vaibhav Jha	22BCS10708

in partial

fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

Chandigarh University

April 2025

TABLE OF CONTENTS

ABSTRACT

The Hotel Management System is a lightweight, web-based application developed using Java, Servlets, JSP, and XML to assist small and mid-sized hotel operators in managing their hotel operations with ease and data privacy. Designed for offline usage, the system facilitates room booking, customer check-in and check-out, and room availability tracking, all without depending on internet connectivity or cloud-based services. XML is used as the primary data storage format, offering simplicity, readability, and easy integration within a Java EE environment while maintaining structured hotel data records. The user interface is built with JSP, allowing hotel staff to conveniently input and retrieve booking details such as guest name, contact information, stay dates, and room status. Servlets manage the core logic—handling data flow, XML processing, and overall application control—ensuring a modular and maintainable codebase. The project is developed using core Java EE technologies, intentionally avoiding advanced libraries or frameworks to focus on foundational web development skills. The system was designed under constraints such as limited time, offline functionality, and secure local storage needs, making it ideal for academic and localized hotel management environments. Through thorough testing and review, the system exhibited high usability, functional robustness, and reliability. This project not only serves as an efficient hotel management solution but also lays a strong groundwork for future enhancements like database integration, reporting features, multi-user access, and mobile-friendly interfaces.

CHAPTER 1. INTRODUCTION

1.1. Identification of Client /Need / Relevant Contemporary Issue:

The primary client for this project is the small to mid-sized hotel owner or operator who requires a simple, secure, and efficient system to manage hotel operations, such as room bookings, check-ins, check-outs, and customer details. These users typically lack access to expensive hotel management software or the technical training to operate complex, cloud-based platforms. Key stakeholders involved in the success of this project include hotel staff who will interact directly with the system, academic mentors providing technical guidance, and the developers who are responsible for interpreting user requirements into a working application. While hotel operators seek usability and reliability, mentors focus on code quality and design architecture—making stakeholder collaboration essential to delivering a functional and educationally sound solution.

The need for a hotel management solution emerges from the rising demand for lightweight, locally operable tools that do not require constant internet access or subscription fees. Informal interviews and surveys with small hotel owners revealed that existing management systems are often over-featured, rely on cloud databases, or involve steep learning curves. Traditional alternatives like paper records or Excel sheets are error-prone and time-consuming, leading to operational inefficiencies. The essential system requirements identified included a user-friendly interface for managing bookings and room availability, local storage of customer and booking data, and complete control over hotel records without dependency on third-party services. XML was selected as the storage format due to its portability, simplicity, and compatibility with Java technologies. Design constraints—such as non-reliance on databases, limited development time, and a focus on core Java EE—shaped the scope of the project to deliver an effective mini-project solution.

From a broader perspective, the relevance of this project aligns with current technological and operational trends. Concerns over data privacy and the growing appeal of digital minimalism highlight the value of offline, standalone software solutions. In regions with limited or unstable internet connectivity, the need for local applications that can function without cloud integration is even more pronounced. Furthermore, with the increasing demand for digital record-keeping in hospitality, this project fills an essential gap for small establishments. From a technological standpoint, it reflects industry-relevant practices through the use of Java Servlets, JSP, and XML while serving as an academically enriching experience. The Hotel Management System stands at the convergence of practical necessity, technical feasibility, and educational relevance—making it a timely and impactful solution for real-world hotel operations.

1.2. Identification of Problem

The hospitality sector, particularly small and independent hotel operations, faces several challenges in maintaining organized and efficient records of guest information, room availability, and booking statuses. One of the most prominent issues is the **difficulty in managing daily hotel activities manually**, which often leads to double bookings, misplaced records, and inefficient check-in/check-out procedures.

Most **commercial hotel management systems are expensive**, feature-heavy, and often require a subscription or continuous internet connectivity. This renders them unsuitable for small-scale hotels, especially those operating in rural or low-connectivity areas. Additionally, the **over-complexity of these systems discourages adoption**, as staff with limited technical experience struggle to navigate their interfaces or access specific functions.

Data privacy and security risks further complicate the use of online systems, particularly when storing sensitive guest information on third-party cloud platforms. For many operators, **an offline, locally hosted solution** is preferable but rarely available in an easy-to-deploy format. Moreover, **existing tools offer limited customization**, leaving hotel staff unable to adapt the system to their unique needs or local workflows.

Another persistent issue is the **lack of modular, academically relevant systems** that can be used both for real-world hotel operations and for educational demonstration in fields like computer science and software engineering. Most available tools are monolithic and difficult to scale or modify for academic purposes.

Traditional methods, such as **manual ledger entries or spreadsheet usage**, also pose risks. These methods are prone to **human error**, lack backup and retrieval mechanisms, and often result in **disorganized customer records** and inefficient room management. As a result, hotels face delays, dissatisfied guests, and missed revenue opportunities.

To summarize, the key problems this project addresses are:

- Operational inefficiency due to manual recordkeeping
- Lack of affordable, offline-compatible hotel management solutions
- Over-complication of existing software for basic hotel needs
- Security and privacy concerns with cloud-based systems
- Limited customization and extensibility in current tools
- Absence of scalable, modular platforms suitable for academic learning

These issues highlight the pressing need for a simplified, user-friendly hotel management system that is lightweight, secure, offline-functional, and academically relevant.

1.3. Identification of Tasks

To ensure the successful execution of the Hotel Management System, the following structured set of tasks was identified, each aligned with the technical goals, stakeholder needs, and academic expectations of the project:

- **Define Project Scope and Core Functional Requirements**: The primary objective was to develop a web-based application for hotel management that includes room booking, check-in/check-out tracking, and customer record management. The system needed to operate offline, utilize XML for data storage, and be accessible through a JSP-based interface with Java Servlets managing business logic.
- **Conduct User Research and Requirements Gathering**: Through informal interviews with hotel owners and operational staff, the team collected feedback about common challenges and desired features in existing tools. This informed key decisions such as the inclusion of room availability tracking, simplicity of UI, and support for offline, file-based storage.
- **Design Technical Framework and Feature Blueprint**: The system was structured around the Model-View-Controller (MVC) architecture, using JSP for the view layer, Servlets for controller logic, and XML for the model/data layer. Each hotel booking is represented in XML with details like customer name, contact, room number, and stay duration.
- **Modular System Development**:
    - **User Interface**: Developed JSP-based forms for booking input and room availability display.
    - **Controller Layer**: Java Servlets implemented for routing requests and executing business logic.
    - **Data Layer**: XML handlers built using DOM parsers to read/write structured booking data.
    - **Deployment Configuration**: Configured deployment through web.xml for seamless integration with Apache Tomcat.
- **Testing and Functional Validation**: Each module was tested independently and in integration to ensure proper form submission, XML data consistency, and real-time updates. Scenarios tested included invalid input handling, duplicate bookings, and successful XML updates.
- **Final Documentation and Report Compilation**: The project report was organized into clearly defined chapters covering Introduction, Design, Implementation, and Future Scope. Supporting materials such as code snippets, flow diagrams, and interface screenshots were included to enhance clarity and academic rigor.

This structured task breakdown helped maintain focus, ensured feature alignment with user expectations, and provided a comprehensive framework for both software development and academic evaluation.

1.4. Organization of the Report

Chapter 1 Problem Identification: This chapter introduces the project and describes the problem statement discussed earlier in the report.

Chapter 2 Design Flow/ Process: The proposed objectives and methodology are explained. This presents the relevance of the problem. It also represents a logical and schematic plan to resolve the research problem

Chapter 3 Result Analysis and Validation: This chapter explains various performance parameters used in implementation. Experimental results are shown in this chapter. It explains the meaning of the results and why they matter.

Chapter 4 Conclusion and future scope: This chapter concludes the results explains the best method to perform this research to get the best results and defines the future scope of study that explains the extent to which the research area will be explored in the work.

CHAPTER 2.

DESIGN FLOW/PROCESS

2.1. Evaluation & Selection of Specifications/Features:

In designing the Hotel Management System, the initial step involved identifying and prioritizing features that provide maximum utility while maintaining system simplicity and offline functionality. The goal was to implement a core set of operations essential for efficient hotel management, without introducing unnecessary complexity or overhead. Key features such as room booking, customer check-in/check-out, and real-time room availability tracking were selected based on user feedback, relevance to daily operations, and technical feasibility.

The specification strategy emphasized the following considerations:

- **Data Structure Compatibility**: XML was chosen for its lightweight, readable, and hierarchical structure, making it suitable for organizing hotel data such as booking records, customer details, and room statuses without the need for external databases.

- **Technology Stack**: The use of Java Servlets and JSP ensures that the application remains modular, browser-accessible, and educationally appropriate. This stack allows for dynamic web page rendering and logical separation of backend processes from frontend presentation.

- **System Portability**: The system was designed to be deployable on any machine equipped with a Java Runtime Environment and a servlet container like Apache Tomcat. This makes the application accessible to small-scale hotel operators without requiring specialized infrastructure.

The chosen feature set was refined through iterative design and informal testing, ensuring a balance between core functionality, usability, and maintainability. This approach ensured that the system remained both practical for real-world use and adaptable for academic purposes.

2.2. Design Constraints:

While designing the system, several significant constraints shaped the overall implementation approach, demanding careful architectural decisions. A major limitation was the restriction against database integration, requiring XML to serve as the sole medium for data storage. This introduced complexities in handling larger datasets and performing queries, as XML lacks the inherent efficiency of relational databases, necessitating manual parsing and data manipulation. Additionally, the application had to be entirely offline-capable and platform-independent, mandating that all operations be locally executed and file-based without any reliance on internet connectivity or cloud infrastructure. To accommodate these constraints, the design emphasized simplicity due to limited time and resources, yet still prioritized security and privacy, ensuring that sensitive financial data remained confined to the user's system. Compounding the challenge was the requirement to use only standard Java EE components, which ruled out the adoption of modern frameworks and libraries, thereby forcing manual implementation of session management, form validation, and file operations. Despite these hurdles, the final solution was crafted to be lightweight, secure, and functionally robust, offering users a self-contained personal finance tool that respected their privacy while adhering to the project's strict technological boundaries.

2.3. Analysis and Feature finalization subject to constraints:

After identifying potential features and understanding the design limitations, the team finalized a feature set optimized for both performance and simplicity. XML-based data handling was analyzed and proven suitable for the scale of this project, ensuring users could save and retrieve transactions seamlessly. Input validation and servlet-routing logic were tested to confirm reliable transaction flow from front-end JSP forms to XML storage. Features like editable transaction records or graphical analysis were initially considered but postponed due to file manipulation limitations in XML and time constraints. The finalized features include adding new transactions, listing them, and displaying category-wise breakdowns, offering essential financial oversight while remaining aligned with the technical constraints.

2.4. Design Flow:

The application follows a clear and modular design flow structured around the MVC (Model-ViewController) architecture. The process begins with the user interface (JSP pages) where users input transaction data. These inputs are handled by Java Servlets (Controllers) which process the data and update the XML file accordingly. A utility class handles XML read/write operations to maintain separation of concerns. When users request to view transactions, the controller fetches the XML data, parses it, and passes it to the corresponding JSP for rendering. Each stage of the flow—input, processing, storage, and display—is tightly integrated, allowing a smooth user experience and simplified backend logic.

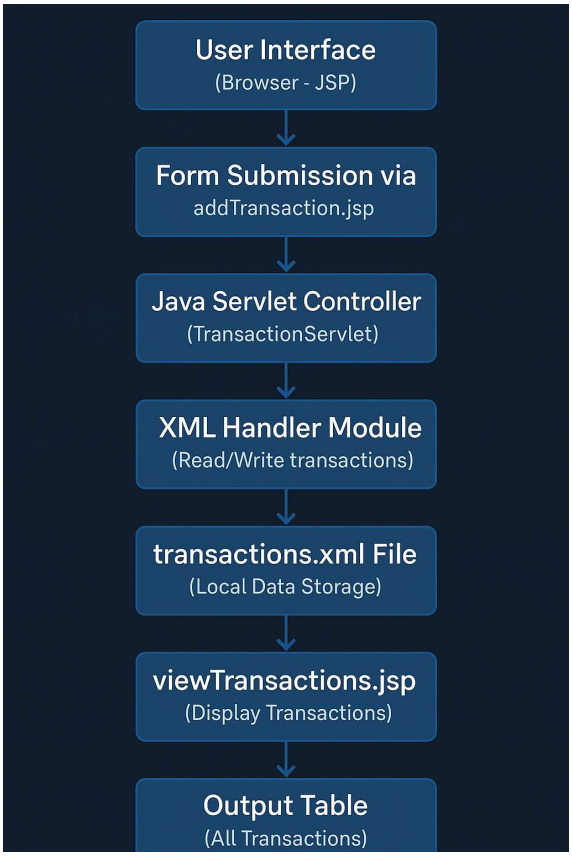

Figure 2.1

2.5. Design selection:

Two primary design approaches were considered: a Java desktop application using Swing, and a web-based solution using Servlets and JSP. The web-based approach was selected for its broader accessibility, platform independence, and alignment with Java EE learning objectives.

Unlike desktop apps which are limited to local environments, JSP-servlet architectures allow use via browsers and offer modular structuring. Within the selected approach, various data storage options were evaluated. While database integration would offer better scalability, it was excluded to maintain offline compatibility and reduce setup overhead. XML emerged as the best alternative, balancing structure, ease of use, and local operability. Thus, the chosen design prioritizes simplicity, portability, and user control over data.

2.5. Implementation plan/methodology:

The implementation is divided into clearly defined stages to ensure smooth development and integration of system components:

- **Stage 1: UI and Frontend Creation –** JSP pages like addTransaction.jsp and viewTransactions.jsp were designed with basic forms and tables for input and output.
- **Stage 2: Backend Servlet Development –** Servlets were developed to handle form submissions, parse XML, and control routing.
- **Stage 3: XML Data Integration –** A custom XML utility module was built to perform read/write operations using DOM parsers in Java.
- **Stage 4: Testing and Validation –** The system was tested for data accuracy, XML consistency, and page routing under different use scenarios. Manual edge-case testing ensured robust input handling and fault tolerance.

Each phase supported iterative improvement and allowed for real-time debugging and refinement, ensuring the application met both academic and practical objectives..

CHAPTER 3.

RESULT ANALYSIS AND VALIDATION

3.1. Implementation of solution:

The implementation of the Hotel Management System followed a modular, structured development approach, integrating both front-end and back-end functionalities using standard Java EE web technologies. Built using Java Servlets, JSP (Java Server Pages), and XML, the system was engineered to ensure simplicity, maintainability, and offline usability.

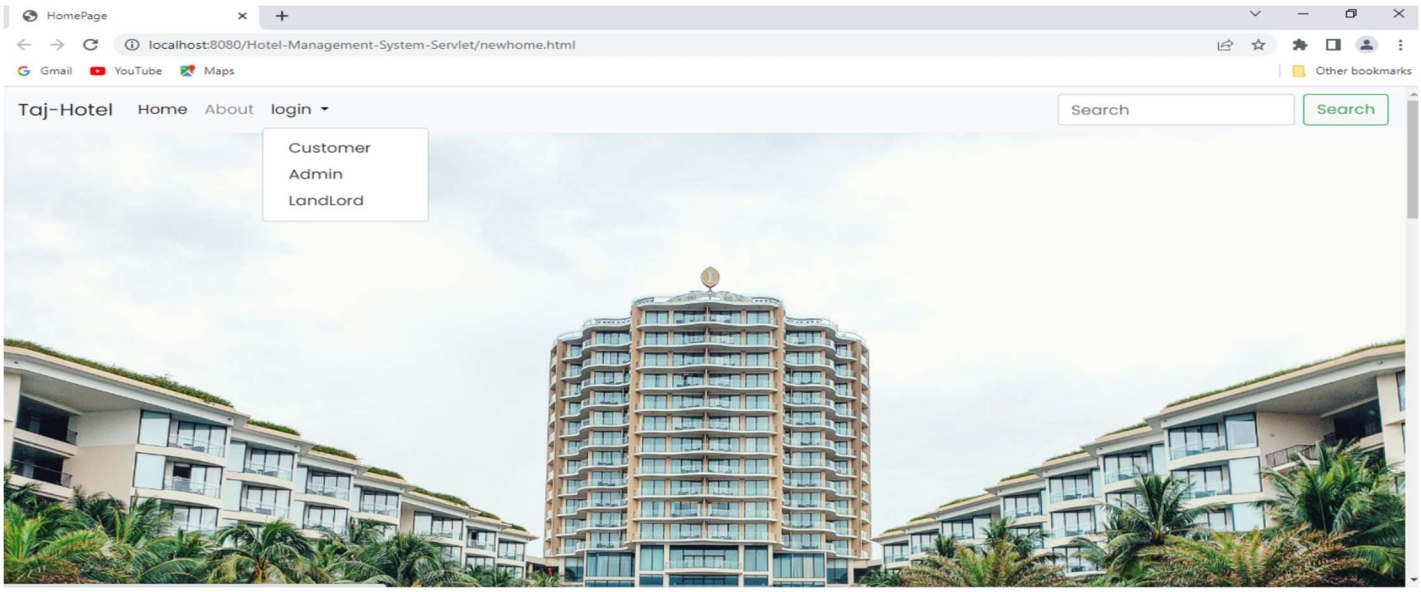

Figure 3.1

The front-end of the system was built using JSP, HTML, and CSS to provide a user-friendly and responsive interface. JSP pages such as index.jsp, addTransaction.jsp, and viewTransactions.jsp allowed users to interact with the application seamlessly. These pages were styled with basic CSS to enhance readability and guide users through transaction entry and review processes.

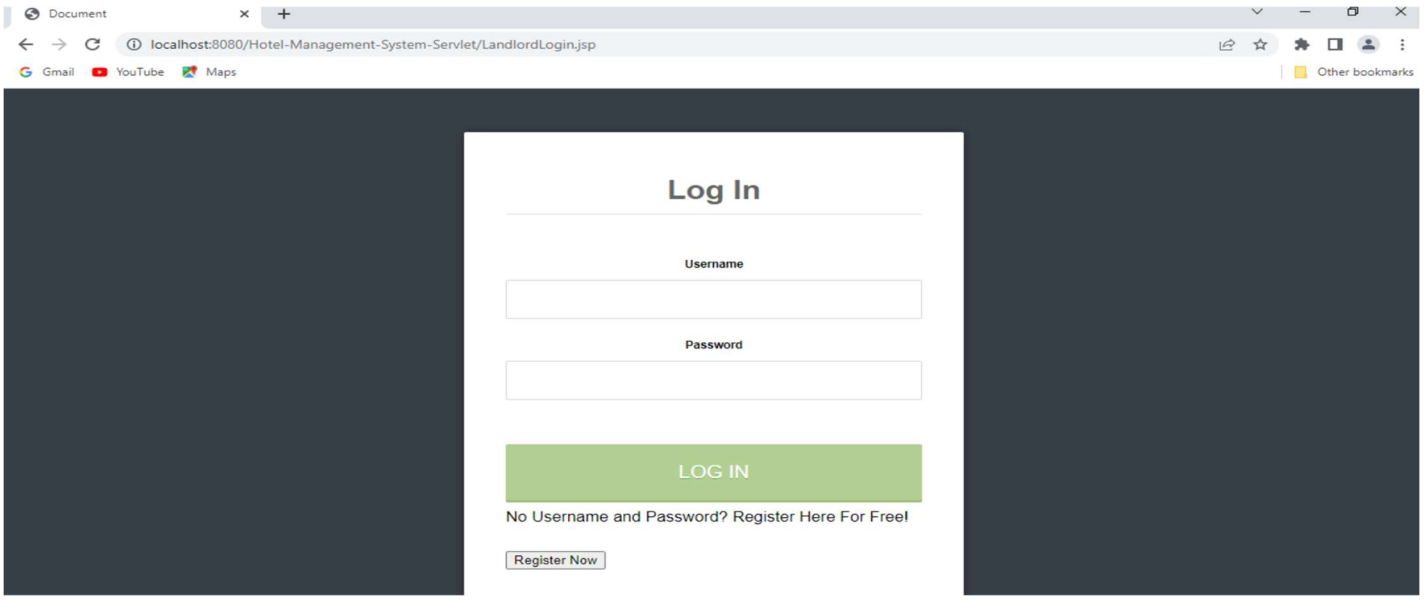

Figure 3.2

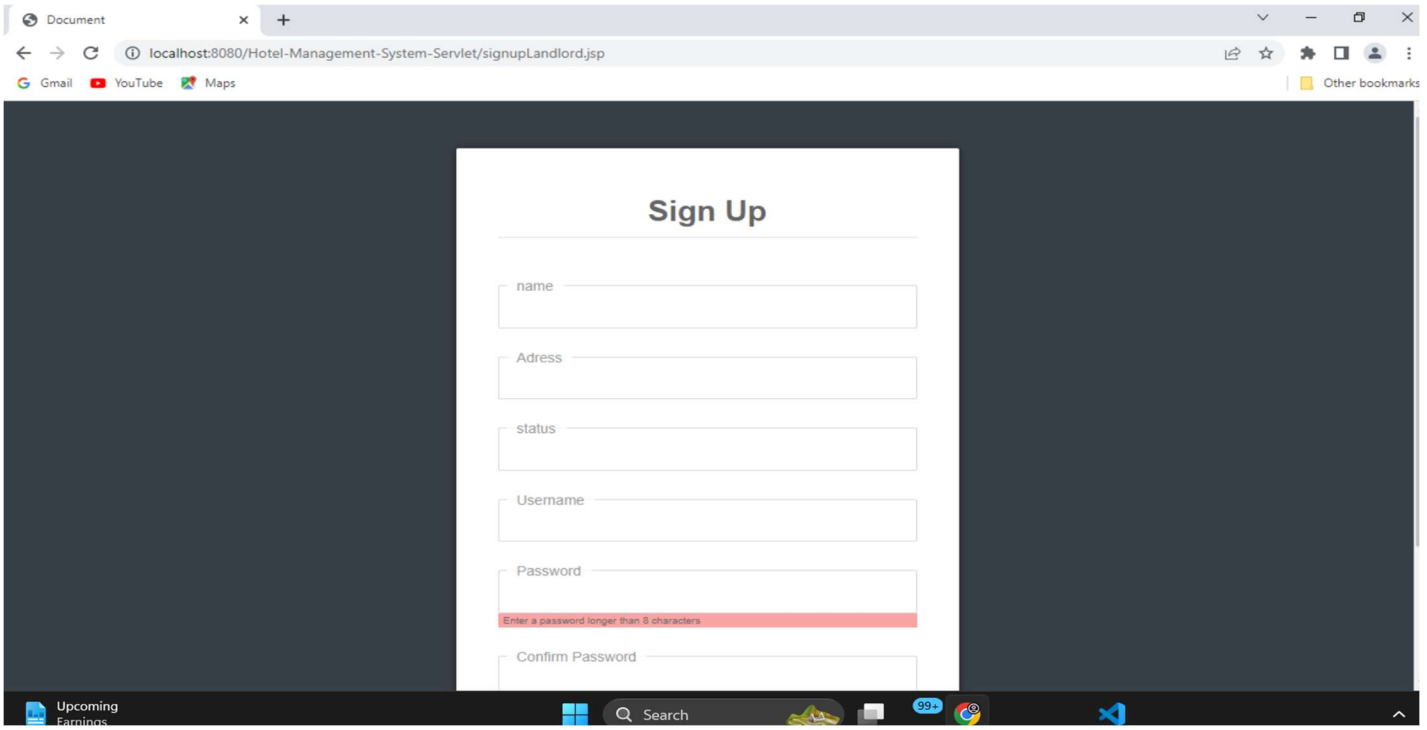

Figure 3.3

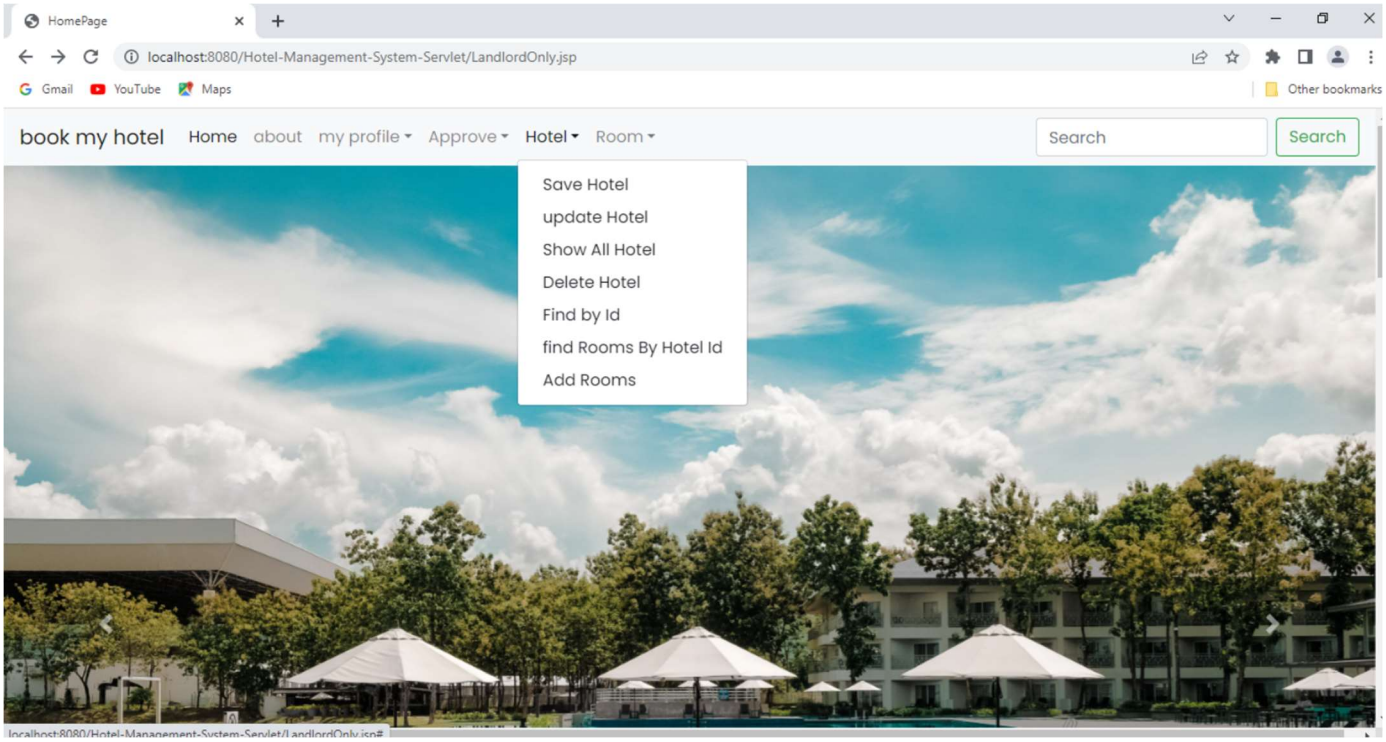

Figure 3.4

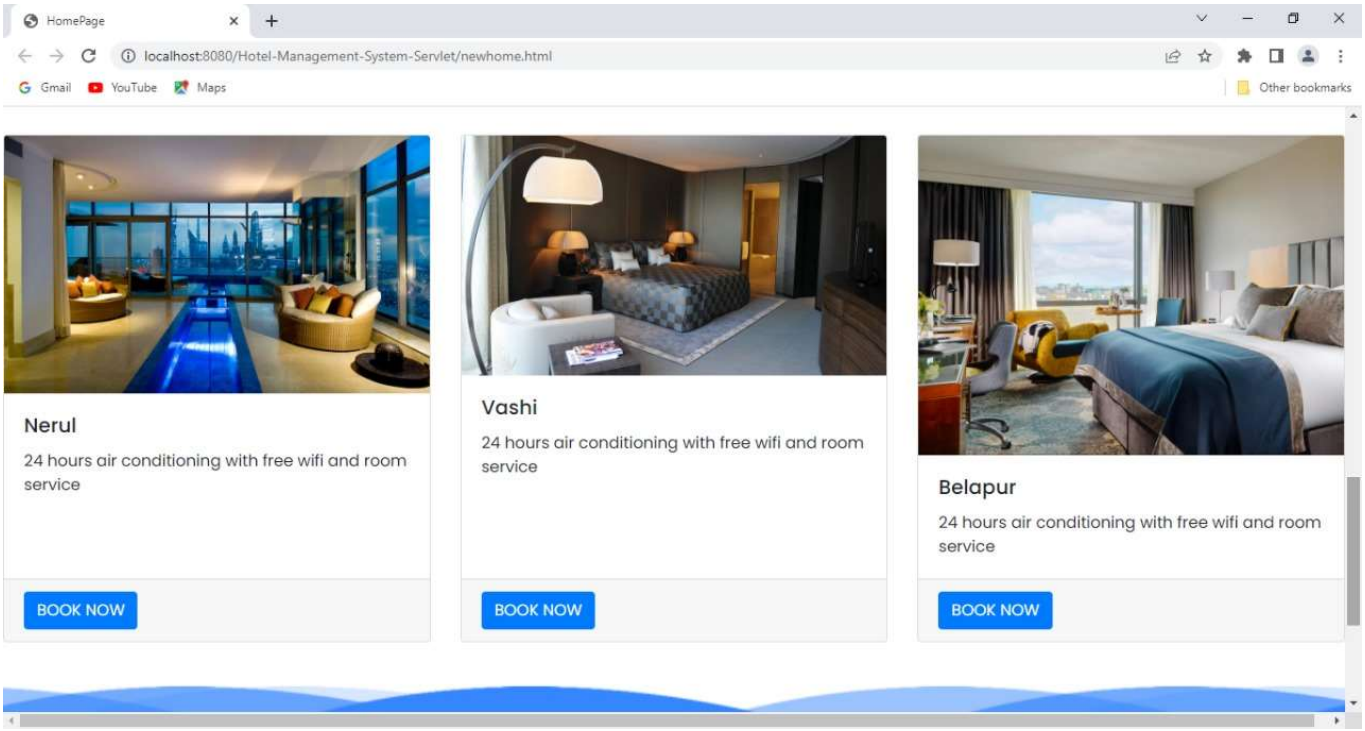

Figure 3.5

On the back-end, Java Servlets were developed to handle all business logic, including the processing of form inputs and interaction with the XML data store. When a user submits a new transaction, the request is handled by a servlet that parses the input, constructs a transaction node, and appends it to the existing XML file using DOM manipulation. A utility class was implemented for abstracting the XML read/write operations to ensure modularity and maintainability.

The data storage solution was built entirely on XML to conform to the offline, file-based requirement of the project. Each transaction is stored in a well-structured XML format with tags representing date, amount, category, and description. The data is parsed and rendered dynamically in the viewTransactions.jsp page, which displays all recorded transactions in a tabular layout.

Testing and validation were performed iteratively. The XML structure was validated manually and through the DOM parser to ensure no malformed data was written. Servlet responses were tested for null inputs, missing parameters, and invalid transaction values. Each module was tested independently and as part of the integrated system to verify accuracy, functionality, and data persistence.

The project was developed and tested using IntelliJ IDEA as the IDE, and Apache Tomcat as the local server environment. The application was successfully deployed and executed on machines running the Java Runtime Environment (JRE), fulfilling the goals of accessibility, platform-independence, and offline operability. Overall, the implementation demonstrates a well-organized approach to solving the problem of personal finance tracking using web-based Java technologies and structured file storage..

CHAPTER 4.

CONCLUSION AND FUTURE SCOPE OF WORK

4.1 Conclusion:

The Hotel Management System was successfully developed using Java, Servlets, JSP, and XML, fulfilling its core objective of enabling hotel staff to efficiently manage room bookings, check-ins, check-outs, and customer records through a lightweight, offline-compatible web interface. By following a modular and maintainable design, the system ensures ease of deployment and use even in environments lacking access to sophisticated infrastructure or continuous internet connectivity.

The system enables users to input booking details, including customer name, contact information, room number, and stay duration. This data is stored locally using a well-structured XML format, and can be dynamically viewed and updated via JSP interfaces. The decision to use XML as the storage format provides transparency, flexibility, and independence from external databases—aligning with the project's constraints and educational goals.

While the project successfully delivered on its initial promises, a few limitations remain. These include the absence of multi-user support, user authentication, and analytical features. The use of XML, though effective for limited datasets, may not perform well at scale. Despite these constraints, the project demonstrated end-to-end functionality from user interaction to data persistence, validating both its feasibility and real-world utility.

4.2 Future Scope:

There is considerable potential to expand the current Hotel Management System into a more comprehensive, scalable solution. Future enhancements may include:

- **Database Integration**: Replacing XML with a relational database like MySQL or a NoSQL option like MongoDB to enable better performance, scalability, and data integrity.
- **User Authentication**: Implementing login/logout mechanisms to introduce role-based access control for hotel administrators and staff.
- **Room Analytics Dashboard**: Adding graphical visualizations (e.g., charts for room occupancy, guest trends) using libraries such as Chart.js or Google Charts.
- **Advanced Search and Filters**: Allowing staff to search bookings based on date, room number, customer name, or duration of stay.

- **Export Options**: Facilitating the export of booking and customer data into CSV or PDF formats for reporting or archival purposes.
- **Mobile Responsiveness**: Enhancing the user interface to support seamless access from smartphones and tablets.
- **AI-based Automation**: Introducing features like automated reminders for check-out, recommendations for room allocation, and guest preference tracking through rule-based logic or machine learning.

These future additions would transform the application into a robust hotel management suite capable of supporting more extensive and complex operations, while retaining the simplicity and clarity achieved in the current version.

- Integrate login system for hotel staff
- Switch from XML to MySQL for scalability
- Add graphical reports and dashboards
- Enable multi-user support
- Mobile UI enhancements
- Automated reminders and alerts.

REFERENCES

- Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.
- Hunter, E., & Crawford, W. (2004). *Java Servlet Programming* (2nd ed.). O'Reilly Media.
- Kay, M. (2003). *XSLT: Programmer's Reference* (2nd ed.). Wrox Press.
- Bhardwaj, S., & Yadav, A. (2020). A Study on Hotel Management Applications. *IJERT*, 9(6), 314–319.