An Integrated Java-Based Hostel Management System: Architecture, Concurrency, Distributed Communication and Performance Assessment

[1]Keshav Yadav, [2]Satyvart, [3]Pushpraj Singh, [4]Rajendra Singh
[1,2,3]Department of Computer Science and Engineering, Raffles University, Neemrana, Rajasthan – 301705, India
[4]Department of Computer Science and Engineering, Raffles University, Neemrana, Rajasthan – 301705, India

***Abstract*—Residential facilities at higher educational institutions generate considerable administrative burdens that conventional paper-based record-keeping methods cannot sustain as student populations expand. This paper presents the collaborative design, construction, and empirical assessment of a Hostel Management System (HMS) — a desktop application developed atop the Java Standard Edition platform — that unifies five core operational domains within a coherent three-tier software architecture. The system employs the Java Swing toolkit for graphical interaction, JDBC-backed MySQL storage for reliable data persistence, concurrent multithreading to accelerate the registration workflow, Java object serialisation as a supplementary local backup mechanism, and Java Remote Method Invocation (RMI) as a foundation for future multi-workstation networked deployment. A structured test suite covering all functional modules produced no failures across twelve distinct test scenarios. Quantitative benchmarking established that every user-facing operation completes in fewer than 400 milliseconds, a threshold consistently associated with perceptually instantaneous responsiveness in human–computer interaction research. The application memory footprint remains below 140 MB under active load, placing it well within the resource envelope of any contemporary administrative workstation. The work contributes a replicable, open-source-stack reference implementation that institutions operating under constrained IT budgets can adopt or adapt directly, alongside a candid analysis of current limitations and a structured roadmap for subsequent enhancement.**

***Index Terms*—Hostel Management System; Java Swing; JDBC; MySQL; Java RMI; Multithreading; Object Serialisation; Three-Tier Architecture; Student Information System; Desktop Application.**

I. INTRODUCTION

University hostel facilities occupy a critical position in the student welfare infrastructure of residential institutions. For a large proportion of enrolled students, particularly those originating from districts geographically remote from the campus, guaranteed accommodation is a prerequisite for enrolment. Despite this significance, the administrative apparatus governing these facilities — encompassing student intake, room allocation, fee tracking, catering coordination, and regulatory dissemination — continues to depend on paper-based workflows that were conceived for cohort sizes far smaller than those now served by rapidly expanding private universities across India.

The operational consequences of this mismatch manifest in several concrete ways. Retrieving a single student's registration record may demand manual searching across multiple ledgers. Verifying fee payment status cannot be done without consulting at least two independent registers. Catering menus and hostel rules reach students only through physical notice boards that a significant proportion of the resident population never views. Most critically, physical registers constitute the exclusive backup of institutional data, leaving years of administrative records vulnerable to irretrievable loss through fire, flooding, or simple negligence.

The present work responds to these challenges through the design and construction of a purpose-built Hostel Management System. The implementation is deliberately confined to the Java Standard Edition platform — Java Swing for the graphical interface, JDBC for relational database access, Java RMI for distributed object communication, and the Java concurrency API for thread management — so that the entire system operates without any dependency on third-party frameworks. This philosophy minimises the risk of dependency obsolescence while facilitating long-term institutional maintenance on limited IT support budgets.

1.1.Problem Definition and Motivation

Five distinct operational deficiencies motivate the system's construction. First, student enrolment data is dispersed across disconnected paper registers, making consolidated retrieval slow and error-prone. Second, the complete absence of digital access control means that any staff member can examine or modify student records, creating both a privacy risk and an auditability gap. Third, separate maintenance of registration and fee registers produces contradictions that staff lack both the time and the tools to reconcile systematically. Fourth, catering schedules and hostel regulations are communicated exclusively through physical notice boards, creating inequitable information access for students unable to check the board consistently. Fifth, the total absence of digital backup exposes institutional records to permanent loss from a single adverse event.

It is tempting to frame hostel administration as a peripheral clerical concern, but the downstream effects of administrative failure are experienced directly by students. Inability to locate a registration record can delay room assignment, creating genuine welfare consequences. Billing errors arising from inconsistent registers produce disputes that neither students nor administrators should need to navigate. Inability to generate an accurate, current occupancy list in an emergency scenario constitutes a genuine institutional safety liability. Digitalising these processes is therefore

not merely about reducing administrative labour; it is about constructing a reliable, recoverable record infrastructure upon which an institution can depend responsibly.

1.2.System Objectives

The system is designed to achieve the following measurable objectives:

- Deliver a Java Swing graphical interface that organises all administrative functions through a unified navigation hub accessible from a single launch screen.
- Implement a student registration workflow capturing seven data fields and persisting records to a MySQL relational database via JDBC.
- Enforce credential-based access control, restricting student record management operations to authenticated administrators.
- Provide full create, read, update, and delete (CRUD) capability over registered student records within the authenticated management panel.
- Establish a Java RMI client–server architecture as the structural foundation for future multi-workstation deployment.
- Utilise Java multithreading to execute database persistence and object serialisation concurrently, reducing the perceived registration response time.
- Deliver digital information panels for catering schedules and hostel regulations.
- Validate all modules through functional testing and report quantitative performance benchmarks.

II. RELATED WORK

2.1.Evolution of Hostel Administration Software

Purpose-built software for managing student accommodation appeared in the mid-1990s in the form of standalone single-machine programs backed by flat-file stores or early consumer database tools such as Microsoft Access. These systems could handle intake registration and, in limited cases, elementary room tracking, but data sharing between staff members required physically transferring files, and the concept of shared concurrent access was entirely absent. They represented a marginal improvement over handwritten registers while reproducing many of the same structural fragilities.

A field study conducted by Kumar and Sharma (2012) across twenty engineering colleges in India found that over three-quarters of surveyed institutions were still relying on either paper-based processes or this first generation of isolated desktop tools. The catalogue of operational problems they documented — difficulty producing consolidated cross-register reports, contradictions between fee ledgers and intake records, and the total absence of any data redundancy mechanism — maps directly onto the deficiencies that motivate the present work.

Subsequent years produced a shift toward browser-delivered, server-hosted solutions. India's University Grants Commission actively promoted the Samarth ERP platform, which bundles hostel administration within a comprehensive institutional management suite and has been adopted across a number of central universities. These systems offer genuine advantages: simultaneous

multi-staff access, centrally managed storage with institutional backup infrastructure, and cross-module reporting capabilities. The trade-off is a significant deployment and maintenance overhead that renders them impractical for smaller institutions operating with limited IT support capacity.

2.2.Desktop versus Web Architectures for Institutional Software

Web-based platforms might appear to make desktop applications obsolete outright, but the empirical evidence presents a more nuanced picture. Brown and Davis (2019) evaluated twelve academic management system deployments across six performance criteria and found that Java desktop applications consistently outperformed browser-based alternatives on two dimensions: the quality and predictability of the user experience, and the ability to operate reliably without network connectivity. Their analysis concluded that, for residential facilities accommodating fewer than 500 students — which accounts for the majority of private university hostels across India — these advantages frequently outweigh the geographic accessibility benefits that web deployment offers. There is additionally a practical argument for desktop software in environments where IT support capacity is thin. A desktop application starts identically every time, without requiring a web server to be responsive, a database server to be reachable over a network, or a browser to be current. For hostel administrators whose primary expertise lies in student welfare rather than systems management, that behavioural predictability carries genuine operational value.

2.3.Java Swing as a GUI Platform for Academic Applications

Java Swing became part of the standard Java platform in version 1.2 (1998), and its defining characteristic relative to the preceding Abstract Window Toolkit is that it renders every interface component through Java's own 2D graphics engine rather than delegating rendering to the host operating system. The result is visually consistent behaviour across Windows, macOS, and Linux without any platform-specific code modification. Horstmann and Cornell (2017) observe that Swing continues to underpin production systems in financial services and public-sector administration, where long-term stability and predictable cross-platform behaviour are prioritised over visual novelty. For institutional software in resource-constrained settings, the framework's negligible hardware requirements and zero licensing cost reinforce this case. More recently, Patel and Joshi (2021) reported satisfactory user satisfaction outcomes in a Swing-based library management system deployed in an academic context, providing direct contemporary evidence of the framework's continued viability for new institutional deployments.

2.4.JDBC Database Connectivity for Compact Data Models

Java Database Connectivity has been a standard component of the Java platform since version 1.1, and its driver-model architecture decouples application SQL logic from database-vendor specifics. The MySQL Connector/J driver used in this project is Oracle's reference JDBC implementation for MySQL. Nair and Krishnamurthy (2020) compared direct JDBC, Spring JDBC templates, and Hibernate ORM across several academic system development projects and concluded that, for data models comprising fewer than ten entity types, the configuration cost of higher-level frameworks consistently exceeded the development efficiency they delivered. The HMS schema, which

involves two principal tables, falls well within the category where raw JDBC is demonstrably the appropriate tool.

2.5.Java RMI in Distributed Systems Education and Practice

Java Remote Method Invocation, introduced in Java 1.1, abstracts inter-JVM communication behind a local-call programming model. The invoking object calls a generated stub; the stub serialises the arguments and transmits them over a socket to the remote JVM; the return value travels the reverse path. Tanenbaum and Van Steen (2017) identify Java RMI as the most pedagogically transparent mechanism for demonstrating distributed object concepts precisely because its programming model mirrors synchronous local invocation, making the conceptual transition to network-distributed execution minimal. This characteristic makes RMI the natural architectural choice for a final-year undergraduate project that must demonstrate genuine networked programming competence within realistic time and complexity constraints.

2.6.Comparative Positioning of the Proposed System

Table 1 situates the proposed HMS against representative alternative approaches across the five evaluation criteria most pertinent to the institutional desktop context.

Table 1: Comparative Assessment of Hostel Management Approaches

Approach	Open Source	DB Integration	Offline Operation	Multi-User Path	Licensing Cost
MS Access + VBA	No	Limited	Yes	No	Paid
PHP + MySQL (Web)	Yes	Full	No	Yes	Server cost
Python + Tkinter	Yes	Yes	Yes	No	Free
JavaFX + H2 Embedded	Yes	Yes	Yes	Limited	Free
Proposed: Java Swing + MySQL	Yes	Full (JDBC)	Yes	Via RMI	Free

The proposed system is the only approach in this comparison that simultaneously combines open-source licensing, full JDBC-backed relational database integration, offline operability, and a concrete architectural pathway to multi-workstation deployment — all at zero licensing cost.

III. SYSTEM ARCHITECTURE AND DESIGN

3.1.Architectural Overview

The system is structured around a three-tier layered architecture in which each tier bears a clearly bounded responsibility. The presentation tier, realised through Java Swing components, handles interface rendering and user input capture exclusively. The business logic tier processes those inputs, enforces validation constraints, coordinates concurrent thread execution, and orchestrates

communication with the persistence layer. The data tier encapsulates all interactions with the MySQL database and the local object serialisation store. This separation ensures that modifications within any one tier do not require corresponding changes in the others, simplifying both ongoing maintenance and future enhancement.

Alongside the three-tier structure sits the Java RMI component, which introduces a distributed communication layer. The remote interface declares the inter-process contract; a server-side implementation class registers with the RMI registry; a client-side stub resolves the registry binding and invokes remote methods transparently. Although the current deployment targets a single administrative workstation, this architecture provides the structural foundation for genuine network-scale distribution in future phases without requiring a fundamental redesign.

3.2.Functional Module Decomposition

The application is decomposed into six functional modules, each encapsulated in a dedicated Java class and accessible from the main navigation hub upon launch.

The Main Navigation Screen (mainScreen.java) is the entry point for every administrative session. It presents six labelled navigation buttons, each corresponding to one functional area. The navigation pattern is consistent throughout the application: the currently displayed frame is hidden and the target module's frame is instantiated and made visible.

The Student Registration Module (Student.java) manages the hostel intake workflow. It collects seven fields — full name, guardian or parent name, national identity number, date of birth, educational level, home institution, and security deposit amount — with type-appropriate input validation. The class is implemented using the Singleton design pattern, ensuring that the single Student domain object is shared between the registration form and the persistence threads.

The Administrator Authentication Module (Admin.java) presents a credential entry form and validates the supplied username and identification number against administrative records in the database. Access to the student management panel is contingent on successful validation; failed validation displays an error message and leaves the application on the login screen.

The Student Records Management Module is accessible exclusively to authenticated administrators and provides search, edit, and delete operations over the registered student population.

The Catering Schedule and Hostel Rules Modules are informational in nature. The catering module renders a graphical weekly meal timetable within a styled panel. The rules module displays eight regulatory provisions as formatted text labels, serving as a digital counterpart to the physical notice board.

3.3.Database Schema

The MySQL schema comprises two principal relations. The users table functions as a dual-purpose store: records with the user_type column set to 'S' represent registered students, while records bearing user_type 'A' identify administrative accounts. Per-student fields include name, father_name, cnic (national identity number), date_of_birth, eductionLevel, college_name, securityFee, and user_type. A supplementary student's table retains additional registration

metadata. The JDBC connection URL targets a local MySQL 5.x instance exposed by XAMPP on port 3306.

Consolidating student and administrator records within a single table reflects a pragmatic choice appropriate to the deployment scale. In a production environment handling greater record volumes or more granular access control requirements, separating these categories into dedicated tables with appropriate foreign key constraints and role-based permission configurations would be the recommended architectural evolution.

3.4.Concurrency Design

Upon submission of the student registration form, two independent threads are spawned concurrently: Db_Thread and Serialize. Db_Thread constructs and executes an INSERT SQL statement through the DataBase JDBC utility class, writing the student record to the MySQL database. Serialize passes the Singleton Student object to the serialisation utility, which encodes it as a binary byte stream and writes the result to student.ser on the local filesystem as a supplementary backup.

Both threads operate in read-only mode with respect to the student object's fields. Those fields are written by the main application thread before either worker thread is started, and Java's memory model guarantees that values written before a thread is started are visible to that thread upon execution. This design eliminates the need for explicit synchronisation constructs — no locks, no volatile variables, no atomic references — while producing provably correct concurrent behaviour. The architectural benefit is a reduction in perceived registration latency by overlapping two I/O-bound operations that would otherwise execute sequentially, while keeping the Event Dispatch Thread free to maintain interface responsiveness throughout.

3.5.RMI Architecture

The Data Interface remote interface declares the contract governing distributed data exchange between processes. The Data Transfer class provides the concrete server-side implementation and extends Unicast Remote Object, which contributes the network communication infrastructure. At application startup, the Driver class registers the Data Transfer instance with the local RMI registry under the logical name Data Link. Client-side code in Data Receive performs a Naming.lookup call to resolve this binding and subsequently invokes the remote method in a manner structurally identical to a local method invocation. The RMI stub layer handles all argument marshalling, network transmission, and return-value unmarshalling transparently, so no network-specific programming is visible at the call site.

IV. IMPLEMENTATION AND EVALUATION

4.1.Development Environment and Technology Stack

The system was developed on a Windows workstation using NetBeans IDE 8.x, compiled against Java SE 8, and tested against MySQL 5.x running under XAMPP. Database connectivity uses the

MySQL Connector/J JDBC driver, incorporated into the project classpath as a library JAR. Table 2 summarises the complete technology stack.

Table 2: Development Environment and Technology Stack

Component	Technology / Version
Programming Language	Java SE 8
GUI Framework	Java Swing (JDK built-in)
Integrated Development Environment	NetBeans IDE 8.x
Relational Database	MySQL 5.x (via XAMPP)
JDBC Driver	MySQL Connector/J 5.x
Distributed Communication	Java RMI (JDK built-in)
Concurrency Mechanism	java.lang.Thread / Runnable
Secondary Persistence	Java Object Serialisation
Target Operating System	Windows 10 / 11

4.2.Key Implementation Details

4.2.1.JDBC Data Access Layer

All database interactions are routed through a dedicated Database utility class that exposes two public methods: Update_Query(String sql), which invokes Statement.executeUpdate() for data-modifying operations, and Select_Query(String sql), which invokes Statement.executeQuery() and returns a Result Set for data retrieval. Each call dynamically loads the MySQL JDBC driver class and establishes a fresh connection via Driver Manager.getConnection(). This per-call connection approach is appropriate for the anticipated single-administrator usage pattern and avoids the configuration complexity of connection pooling, though pooling is identified as a priority enhancement in Section 5.2.

4.2.2.Singleton Pattern in the Student Domain Class

The student class enforces the Singleton pattern through a private constructor, a private static final field holding the single permitted instance, and a public static factory method. This design ensures that the registration form, which populates the object's seven fields, and the Threads class, which reads those fields for persistence, reference an identical object instance. Because all field writes occur on the Event Dispatch Thread before either Thread.start() is called, the Java memory models happens-before guarantee ensures field-value visibility to the spawned threads without additional synchronisation constructs.

4.2.3.Administrator Authentication Mechanism

The login validation logic queries the users table for records carrying the administrative user-type marker and compares the retrieved name and stored identification number against the credentials supplied through the login form. On a successful match, the management panel is made visible; on failure, an error label is rendered and no screen transition occurs. The current implementation

stores passwords as plain text, a security vulnerability explicitly acknowledged in Section 5.2 alongside the corresponding remediation pathway.

4.3.Student Registration Field Mapping

Table 3 documents the correspondence between registration form fields, Java variables, data types, and MySQL columns, including the type conversion operations applied at input time.

Table 3: Student Registration Field Mapping

Form Label	Java Variable	Data Type	Database Column
Full Name	name	String	name
Guardian Name	fatherName	String	father_name
National ID Number	cnic	int	cnic
Date of Birth	dateOfBirth	Date (dd/MM/yyyy)	date_of_birth
Educational Level	educationLevel	String	eductionLevel
Home Institution	collegeName	String	college_name
Security Deposit (INR)	securityFee	double	securityFee

4.4.Functional Testing

Twelve functional test scenarios were executed covering all six operational modules. Each scenario specified a precondition, an action, and a precisely defined expected system response. All twelve scenarios produced the expected outcome on first execution. Table 4 presents the complete test matrix.

Table 4: Functional Test Scenarios and Outcomes

ID	Module	Scenario Description	Expected Outcome	Result
TC-01	Main Screen	Application launch	Six navigation buttons rendered	PASS
TC-02	Registration	Submit valid student form	DB record created; confirmation shown	PASS
TC-03	Registration	Numeric field type parsing	Values parsed without exception	PASS
TC-04	Admin Login	Correct administrator credentials	Management panel opened	PASS
TC-05	Admin Login	Incorrect credentials entered	Error label; no screen transition	PASS
TC-06	Catering	Open catering schedule module	Weekly timetable image rendered	PASS
TC-07	Rules	Open hostel rules module	All eight regulations visible	PASS
TC-08	Data Viewer	Open serialised data viewer	Deserialised record in JTable	PASS

TC-09	Navigation	Back button on any sub-screen	Returns to parent screen	PASS
TC-10	Serialisation	Restart and reopen data viewer	student.ser persists across sessions	PASS
TC-11	Error Handling	MySQL offline at application launch	SQLException caught; no crash	PASS
TC-12	Concurrency	Registration with concurrent threads	Both threads complete without conflict	PASS

Test case TC-12 merits specific comment. The absence of any race condition is not incidental but architecturally guaranteed: both threads operate in read-only mode with respect to the Student Singleton's fields, which were written by the main thread before either worker thread was started. Java's happens-before relationship provides the necessary memory visibility guarantee, making correct concurrent behaviour a structural property of the design rather than a fortunate runtime outcome.

4.5.Performance Benchmarking

Quantitative timing measurements were collected on the development workstation with MySQL operating locally via XAMPP. Each operation was measured across five consecutive trials. Table 5 presents the observed ranges alongside contextual notes.

Table 5: Performance Benchmark Results

Operation	Measured Range (ms)	Contextual Note
JVM cold start	1800 – 2400	JVM class-loading; outside application control
JDBC connection establishment	120 – 280	Localhost MySQL; negligible network component
Student registration INSERT	180 – 320	Single INSERT on local MySQL instance
Object serialisation write	15 – 45	~2 KB object; local filesystem I/O
Admin login credential query	90 – 200	SELECT plus in-memory string comparison
Record deserialisation + JTable	50 – 120	readObject() plus JTable model population
Screen navigation (setVisible)	< 50	JFrame visibility toggle and instantiation
Memory footprint – idle	65 – 90 MB	JVM plus Swing rendering engine resident set
Memory footprint – active	90 – 140 MB	During JDBC operations with ResultSet open

All transactional operations complete within 400 milliseconds, the threshold established in human–computer interaction research as the upper boundary of perceptually instantaneous responsiveness for interactive applications. The 1.8–2.4 second startup interval reflects JVM class-loading overhead rather than any deficiency in application initialisation logic, and it represents a one-time cost per session that no application-level optimisation can address. The active memory footprint of 90–140 MB is well within the available RAM of any workstation produced within the past decade.

V. DISCUSSION

5.1.System Strengths

The most significant achievement of the HMS is that it constitutes a demonstrably functional, systematically tested implementation rather than an architectural proposal unsupported by a working artefact — a distinction that a survey of the academic literature on institutional management systems reveals to be meaningful. The system satisfies all stated objectives within a zero-licensing-cost technology stack that imposes no dependency on external frameworks, eliminating any vector through which framework deprecation, API changes, or licensing shifts could compromise the system's operational continuity over an extended lifespan.

The three-tier architecture enforces a clean separation of concerns sufficient for a developer unfamiliar with the codebase to understand and modify any individual tier without needing to trace implications through the entire application. The concurrent thread design for registration tangibly reduces perceived response time by overlapping two otherwise sequential I/O-bound operations. The RMI layer provides a network-ready structural foundation that requires only a host-address update and server relocation to support multi-workstation deployment. The 100% functional test pass rate and performance figures consistently below the 400 ms responsiveness threshold together indicate a production-viable implementation for small to medium-scale hostel facilities.

5.2.Limitations and Security Considerations

Several significant limitations must be acknowledged. The most serious is the security posture: administrator credentials are stored as plain text in the database, meaning any party with direct database access through the XAMPP-bundled phpMyAdmin interface can read them immediately. This is not an acceptable posture for a system that controls access to student personal data and must be remediated before any production deployment.

A closely related vulnerability arises from the registration module's SQL construction strategy: INSERT statements are assembled by directly concatenating user-supplied field values into the SQL string. An adversary who inputs carefully crafted text into a name or institution field could exploit this to execute arbitrary SQL commands against the database. Both vulnerabilities have well-understood, straightforward remediation pathways, as the following section documents.

The per-query connection strategy, while adequate for single-administrator operation, introduces unnecessary overhead and will not scale to concurrent multi-user access without a connection pooling layer. The object serialisation store retains only the most recently registered student's

record, overwriting prior data on each registration, which makes it useful for crash recovery within a single session but inadequate as a meaningful archive. Finally, the RMI component has been validated only within a localhost context and has not been exercised in a genuinely distributed network configuration.

VI. CONCLUSION AND FUTURE WORK

6.1.Conclusion

This paper has presented the collaborative design, implementation, and empirical evaluation of a Hostel Management System constructed atop the Java SE platform, integrating Swing-based GUI rendering, JDBC-backed MySQL data persistence, concurrent multithreading, object serialisation, and Java RMI. The system consolidates five operational domains — student registration, administrator authentication, record management, catering information delivery, and regulatory dissemination — into a three-tier architecture that systematically addresses the paper-based inefficiencies prevalent in residential facility administration at resource-constrained institutions.

All twelve functional test scenarios executed without failure. Performance measurements confirm responsive behaviour within the thresholds required for interactive institutional software. The work demonstrates that a feature-complete, maintainable hostel administration platform can be constructed entirely from the standard Java SE library stack, without any external framework dependency, at zero licensing cost. This makes the system directly accessible to institutions for which proprietary or infrastructure-intensive solutions are financially or operationally impractical.

6.2.Future Enhancement Roadmap

Seven enhancements are identified as priorities for subsequent development phases, roughly ordered by urgency:

- Password Security:

Replace plain-text credential storage with BCrypt hashing via the jBCrypt library, eliminating the primary authentication vulnerability documented in Section 5.2.

- SQL Injection Mitigation:

Replace all string-concatenated query construction with parameterised PreparedStatement calls, treating every user-supplied value as data rather than executable syntax.

- Connection Pooling:

Integrate HikariCP to replace per-query connection establishment, enabling concurrent multi-user access with acceptable latency under load.

- Room Allocation Module:

Introduce a rooms entity and a room_allotments relation, with a management panel for capacity tracking, occupancy reporting, and room assignment workflows.

• Fee Payment History:
Implement a payments table capturing fee type, amount, payment date, and receipt reference, with a transaction history view for outstanding dues.

• Notification Services:
Incorporate the JavaMail API to deliver automated email alerts for registration confirmation, fee due dates, and room assignment changes.

• Web Migration:
Expose the business logic tier as a Spring Boot RESTful API and build a React.js frontend, enabling browser-based access from any campus device without requiring local installation.

REFERENCES

[1] Kumar, R. and Sharma, P. (2012). Survey of Hostel Management Information Systems in Indian Engineering Colleges. International Journal of Computer Applications, 58(4), pp. 12–19.
[2] Brown, M. and Davis, L. (2019). Desktop vs. Web-Based Academic Management Systems: A Comparative Evaluation. Journal of Educational Technology Systems, 47(3), pp. 312–328.
[3] Horstmann, C. S. and Cornell, G. (2017). Core Java Volume II — Advanced Features, 10th edn. Upper Saddle River: Prentice Hall.
[4] Patel, K. and Joshi, R. (2021). Library Management System Using Java Swing and MySQL: Design and Implementation. International Journal of Advanced Research in Computer Science, 12(2), pp. 44–51.
[5] Nair, S. and Krishnamurthy, V. (2020). Comparative Study of Database Connectivity Frameworks for Academic Management Systems. Proceedings of the 3rd International Conference on Software Engineering, Bangalore, pp. 112–119.
[6] Tanenbaum, A. S. and Van Steen, M. (2017). Distributed Systems: Principles and Paradigms, 3rd edn. Hoboken: Pearson Education.
[7] Oracle Corporation (2024). Java SE 8 Documentation — Java Database Connectivity (JDBC). Available at: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
[8] Oracle Corporation (2024). Java Remote Method Invocation (RMI) Guide. Available at: https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/
[9] Silberschatz, A., Korth, H. F. and Sudarshan, S. (2019). Database System Concepts, 7th edn. New York: McGraw-Hill.
[10] Goetz, B. et al. (2006). Java Concurrency in Practice. Upper Saddle River: Addison-Wesley.
[11] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley.
[12] Bloch, J. (2018). Effective Java, 3rd edn. Upper Saddle River: Addison-Wesley.

[13] OWASP Foundation (2024). SQL Injection Prevention Cheat Sheet. Available at: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[14] UGC (2022). Guidelines for Hostel Administration and Student Welfare in Higher Educational Institutions. New Delhi: University Grants Commission.

[15] MySQL Documentation (2024). MySQL 5.7 Reference Manual. Available at: https://dev.mysql.com/doc/refman/5.7/en/