User Management System

Project Report

By

Pushpa Kumavat – AF04984857

Deepak Padhy - AF04989850

Index

Acknowledgement

The project **"User Management System"** is the Project work carried out by

Name	Enrollment No
Pushpa Kumavat	**AF04984857**
Deepak Padhy	**AF04989850**

We are thankful to my project guide for guiding me to complete the Project. Their suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

Abstract

The User Management System is a console-based application developed using Java, Hibernate ORM, and MySQL to efficiently manage system users through structured access control. The project is based on the Role-Based Access Control (RBAC) model and consists of four main entities: Admin, User, Role, and Permission. Administrators can perform full CRUD operations, assign roles to users, and map permissions to roles, ensuring secure and controlled data access. Hibernate is used as the persistence framework to eliminate manual SQL handling and enable seamless interaction between Java objects and the database. The system ensures data consistency, modularity, and reusability through DAO (Data Access Object) design. The lightweight console interface makes the application portable, resource-friendly, and suitable for backend administration systems, training purposes, and academic learning. Overall, this project demonstrates efficient database handling, clean architecture, and real-world implementation of RBAC using Hibernate.

1. Introduction

Introduction

In modern software applications, managing users, their access levels, and associated permissions has become an essential requirement for security and efficient workflow handling. Whether in educational platforms, enterprise systems, or e-commerce applications, a well-structured user management system ensures that the right people have the right access at the right time.

To address this need, the project titled **"User Management System"** has been developed. This system provides a console-based interface where administrators can manage Users, Roles, Permissions, and Admin accounts in an organized manner. The system supports role-based access control (RBAC), a widely used security model where permissions are assigned to roles, and roles are assigned to users.

The project is implemented using **Java**, **Hibernate ORM**, and **MySQL** as the database. Hibernate is used to map the Java classes (entities) to database tables, reducing manual SQL writing and improving maintainability. The project follows a modular design with separate Data Access Objects (DAO) for each entity, ensuring clean separation between business logic and database operations.

This system is highly suitable for any application that requires authentication, authorization, and data security. By automating user access control, it reduces manual tasks, prevents unauthorized access, and improves data consistency.

Objectives of the Project

- To provide a centralized system for managing users, roles, and permissions.
- To implement CRUD operations for Admin, User, Role, and Permission entities.
- To use Hibernate ORM to simplify database interaction and reduce SQL dependency.
- To implement role-based access control (RBAC) for structured and secure authorization.
- To design a modular and scalable system that can be extended into a web or enterprise application.

Scope of the Project

The scope of this project includes designing and implementing a console-based User Management System that can later be enhanced into a GUI, web, or REST API-based application. The system can be used in:

- Enterprise applications requiring employee access control.
- Educational systems for admin, faculty, and student access management.
- Corporate systems for assigning roles and permissions to staff.
- Any application where user authentication and authorization are required.

Significance of the Project

The User Management System is an important backend module in modern applications, as it ensures data security, prevents unauthorized access, and maintains user-specific privileges. By using Hibernate and MySQL, the system becomes scalable, database-independent, and easier to maintain. The project demonstrates real-world concepts like RBAC, modular coding, ORM, and CRUD implementation — making it valuable both academically and industrially.

2. System Analysis

System Analysis is the process of examining and understanding an existing workflow to identify its drawbacks, requirements, and improvement areas before designing a new automated system.
In traditional organizations, user-related activities such as user registration, role assignment, permission control, and admin operations are often handled manually or through basic spreadsheets. This creates challenges in data handling, security, and scalability.
The **User Management System** was proposed to provide a centralized and computerized platform that automates user creation, role-based access control, permission assignment, and admin monitoring. This system ensures data integrity, improves security, and simplifies management operations.

1.Problem Definition

In traditional systems, users, roles, and permissions are managed manually or hard-coded, which becomes difficult as the number of users grows.
This leads to issues like security risks, lack of scalability, and difficulty in updating access rights.
The project aims to develop a centralized User Management System that automates user handling, role assignment, and permission control using a database-driven and secure architecture.

2. Existing System

a) Description

In many organizations, user management is handled through:

- Manual record-keeping (notebooks, files, or Excel sheets)
- Separate systems for users, roles, and permissions
- No centralized admin system to track activities
- Role and permission assignment done manually

b) Limitations of the Existing System

- **Time-consuming and inefficient** due to manual entries
- **High chance of human errors** in records and permissions

- **No role-based access control**, so every user has equal access
- **Data security issues** because credentials may be visible or stored in plain text

3. Proposed System

a) Description

- The proposed User Management System is an automated software solution that allows administrators to manage users, roles, and permissions in a centralized and secure way.
- It replaces the manual process of maintaining user details, assigning roles, and controlling access with a structured and database-driven system.
- The system provides CRUD (Create, Read, Update, Delete) operations for Admins, Users, Roles, and Permissions, ensuring better control, transparency, and data security.

b) Features of Proposed System

- Secure login-based access for Admins
- Admin can add, update, delete and view users
- Role Management: Create, edit, and delete user roles (e.g., Admin, Manager, Staff, etc.)
- Permission Management: Assign permissions to roles (e.g., Read, Write, Update, Delete)
- Role-based access control (RBAC) implementation
- Stores all data in MySQL database using Hibernate ORM
- Password stored in encrypted/hashed form (optional enhancement)
- Console-based interface for operations (can be extended to web UI later)

c) Advantages Over Existing System

Existing System	**Proposed System**
Manual entry of users	Automated user management
No role-based access	Role + Permission-based access control
Data stored in files/Excel	Centralized database (MySQL + Hibernate)
Higher chances of data loss	Reliable & secure storage
Difficult to manage large users	Scalable and easy to maintain

4. System Requirements

a) Functional Requirements

Functional requirements define the specific actions and operations the system must be able to perform.

Admin Functions:

- Login to the system securely
- Add, update, delete, and view users
- Create, update, delete, and view roles
- Create, update, delete, and view permissions
- Assign roles to users and permissions to roles
- View all system records (users, roles, permissions)

User Functions:

- User can be created and managed by admin
- User can login based on assigned role (future scope)
- User can have specific permissions depending on assigned role
- Users can update their own basic information (optional enhancement)

Role & Permission Functions:

- Create and manage multiple roles (Admin, Manager, Staff, etc.)
- Add and manage permissions (Read, Write, Update, Delete, etc.)
- Map permissions to roles → Role Based Access Control (RBAC)
- Each user gets access according to assigned role

b) Non-Functional Requirements

Non-functional requirements specify system quality and performance behavior.

Parameter	Description
Performance	System should execute CRUD operations within milliseconds using Hibernate ORM.
Security	Only authenticated admins can access the system. Password input is hidden while typing.
Reliability	Data stored in MySQL database ensures consistency and durability.
Usability	Console interface with simple menu navigation for easy use.
Scalability	Can handle large number of users, roles, and permissions.
Maintainability	Modular DAO structure allows easy modifications and upgrades.
Portability	Runs on any OS supporting Java (Windows/Linux/Mac).

5. Feasibility Study

A feasibility study was conducted to evaluate whether developing the **User Management System** is practical, useful, and implementable in a real-world environment.

a) Technical Feasibility

- The system is technically feasible as it is built using Java, a platform-independent and widely used programming language.
- Hibernate ORM automates database operations and reduces SQL complexity.
- MySQL is used as a reliable and scalable database.
- Works on any standard computer system with JDK installed, no special hardware required.
- All required tools like Java, MySQL, Eclipse/IntelliJ, Maven are free and open-source.

b) Economic Feasibility

- No licensing cost because all development tools are open-source.
- Reduces manual record-keeping cost since all users/roles/permissions are stored digitally.
- No physical paperwork or storage needed, reducing administrative expenses. Long-term maintenance cost is low due to modular structure.

c) Operational Feasibility

- The system is easy to operate through a simple console menu interface.
- Admin can manage users, roles, and permissions without technical knowledge.
- Requires very little training to use.
- Can be deployed in companies, institutes, or any system requiring access control and user management.

d) Legal Feasibility

- The system follows standard data handling practices and does not violate privacy rules.
- No use of copyrighted frameworks; all dependencies used (Java, Hibernate, MySQL, Maven) are open-source.
- Can be legally implemented in any organization without licensing issues.

6. System Study

a) Input Design

The input design defines how data is entered into the system.

Entity	Input Fields
Admin	Admin ID, Name, Email, Password
User	User ID, Name, Email, Phone, Password, Role
Role	Role ID, Role Name (Ex: Admin, Manager, Employee)
Permission	Permission ID, Permission Name (Ex: CREATE_USER, DELETE_USER)

b) Output Design

The output of the system is displayed in console format.

Module	Output
Admin	List of all admins, confirmation messages
User	Table of all users with assigned roles
Role	All roles with linked permissions
Permission	All permissions in DB

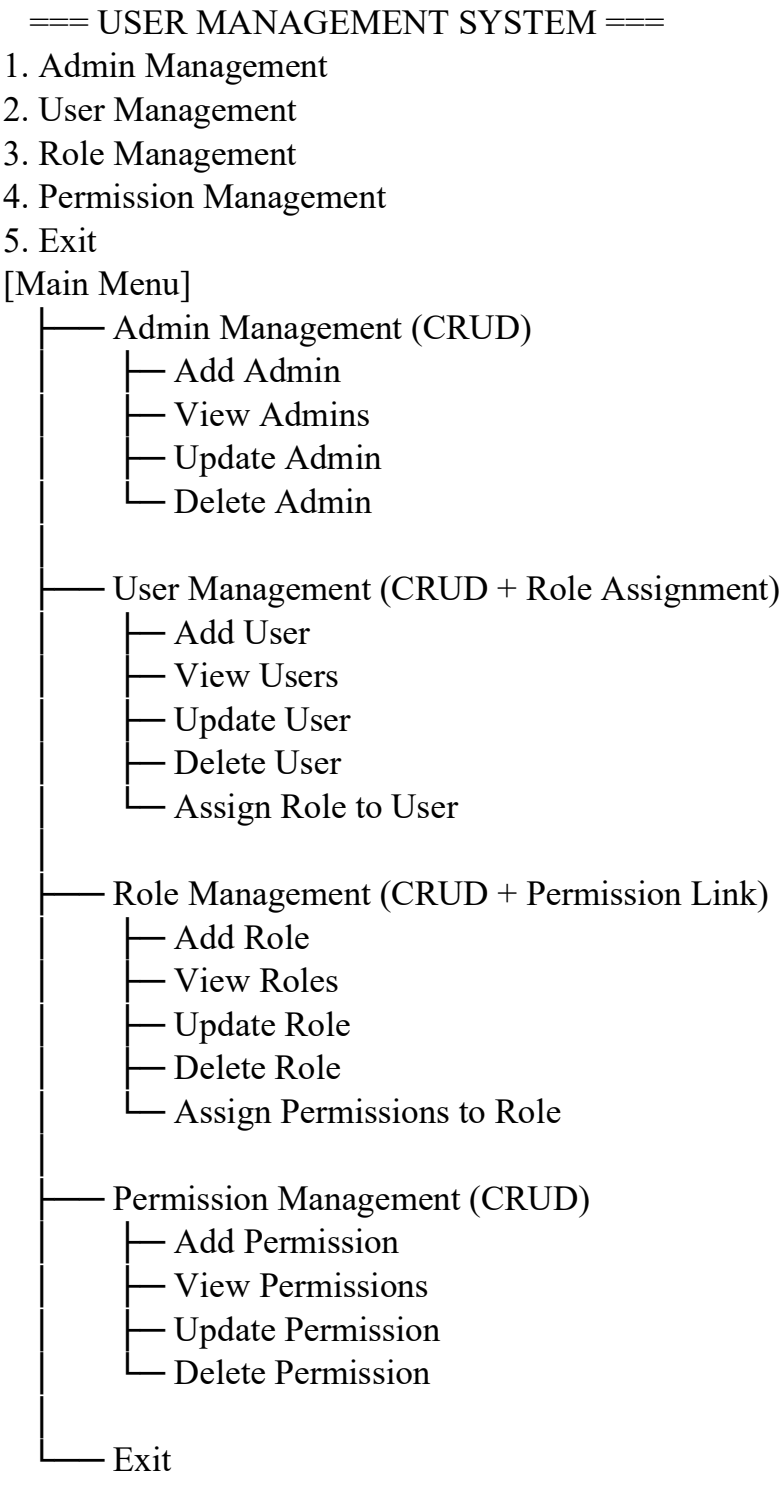
c) Process Design

The system follows CRUD operations for all 4 entities.

Process	Operation
Admin	Add, View, Update, Delete
User	Add, View, Update, Delete, Assign Role
Role	Add, View, Update, Delete, Assign Permission
Permission	Add, View, Update, Delete

7. System Flow

The User Management System follows a menu-driven console flow where Admin controls the entire system and performs CRUD operations on Users, Roles, and Permissions.



Flow Summary:

- Admin Login → Dashboard Menu → Select Operation → Perform CRUD → DB Updated
- Admin Adds User → User Login → View / Update Profile → DB Updated
- Admin Creates Role → Assigns Permissions → Assigns Role to User → User Inherits Permissions

8. System Architecture

The **User Management System** follows a **Three-Tier Architecture**:

1. Presentation Layer (Console UI)

- Interacts with the admin.
- Collects input data for CRUD operations on Admin, User, Role, and Permission.
- Displays menus, results, and confirmation messages.
- Implemented as a **console-based Java application**.

2. Business Logic Layer (Service Layer)

- Contains the main operations logic.
- Processes inputs, validates data, and ensures proper access control.
- Handles CRUD operations for all four entities.
- Classes: AdminService, UserService, RoleService, PermissionService.

3. Data Access Layer (DAO Layer)

- Interacts with **MySQL** using **Hibernate ORM** for CRUD operations.
- Classes: AdminDAO, UserDAO, RoleDAO, PermissionDAO.
- Ensures safe and consistent database transactions.

4. Database Layer

- Consists of four main tables: admin, user, role, permission.
- Each entity is mapped to a table using Hibernate annotations.

5. UML Diagrams

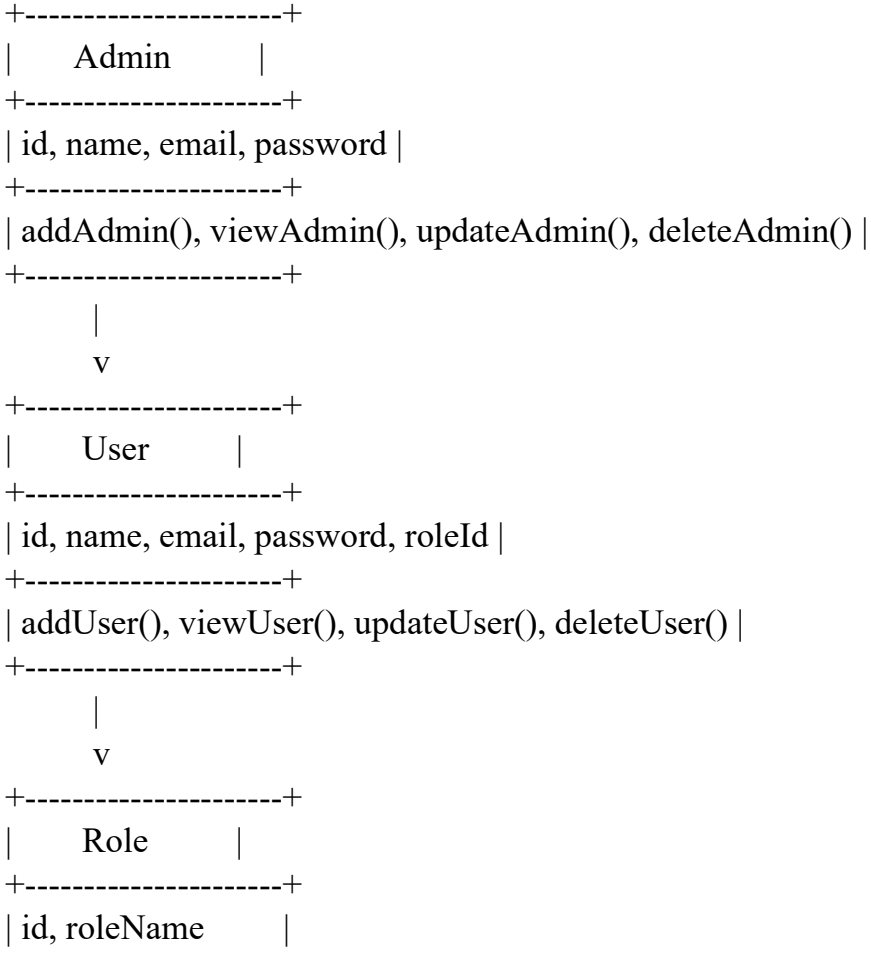
a) Use Case Diagram

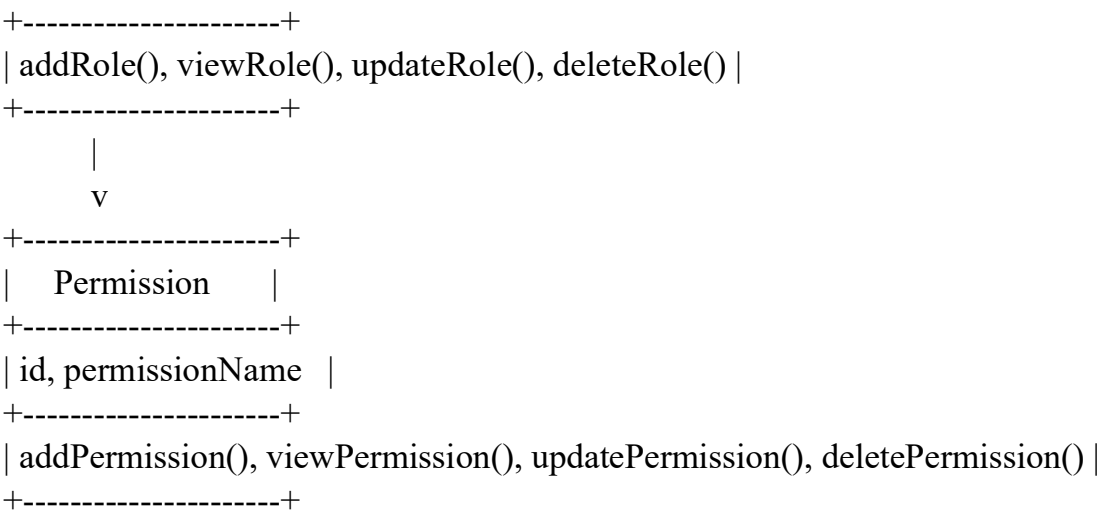
Actor: Admin

Use Cases:

- Manage Admins: Add, View, Update, Delete
- Manage Users: Add, View, Update, Delete
- Manage Roles: Add, View, Update, Delete
- Manage Permissions: Add, View, Update, Delete

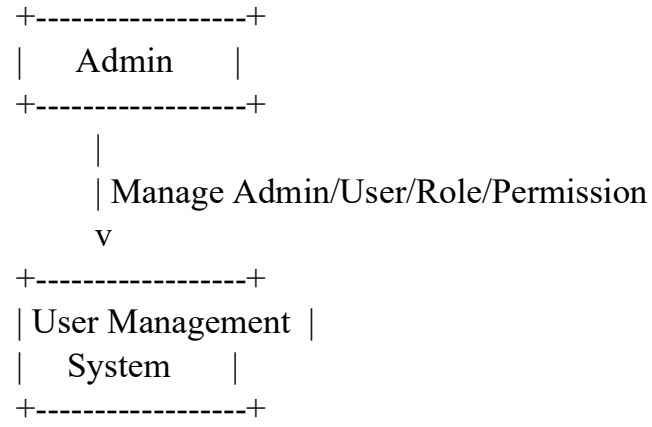
b) Class Diagram





6. Data Flow Diagram (DFD)

Level 0:



Level 1:

- Admin → CRUD Admin → admin table
- Admin → CRUD User → user table
- Admin → CRUD Role → role table
- Admin → CRUD Permission → permission table

7. ER Diagram

Entities & Attributes:

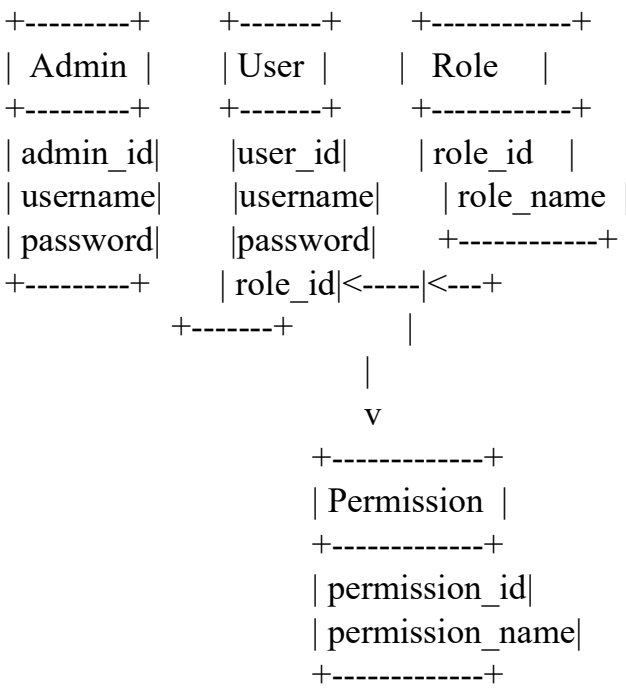
1. **Admin**
    - admin_id (PK)
    - username

- password

2. **User**
    - user_id (PK)
    - username
    - password
    - role_id (FK) → references Role.role_id
3. **Role**
    - role_id (PK)
    - role_name
4. **Permission**
    - permission_id (PK)
    - permission_name

Relationships:

- **User ↔ Role**: Many-to-One
    - Each User is assigned **one role**, but a Role can have **many users**.
- **Role ↔ Permission**: Many-to-Many
    - A Role can have multiple Permissions, and a Permission can belong to multiple Roles.
    - Implemented via a junction table RolePermission:
        - role_id (FK)
        - permission_id (FK)



3. System design

The system is designed to provide a **modular, menu-driven console application** for managing users, roles, and permissions. It separates responsibilities into different layers and modules to ensure maintainability, scalability, and security.

A. Modules

1. **Admin Management**
    - Add, view, update, delete admin accounts.
    - Only accessible by authenticated super-admin (optional).
2. **User Management**
    - Add, view, update, delete users.
    - Assign roles to users to define their permissions.
3. **Role Management**
    - Add, view, update, delete roles.
    - Roles define access levels for users.
4. **Permission Management**
    - Add, view, update, delete permissions.
    - Permissions can later be linked to roles for RBAC.

B. Console Menu Flow

=== USER MANAGEMENT SYSTEM ===
1. Admin Management
2. User Management
3. Role Management
4. Permission Management
5. Exit
Enter choice:

=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back

=== USER MANAGEMENT ===
1. Add User
2. View All Users
3. Update User
4. Delete User
5. Back

=== ROLE MANAGEMENT ===
1. Add Role
2. View All Roles
3. Update Role
4. Delete Role
5. Back

=== PERMISSION MANAGEMENT ===
1. Add Permission
2. View All Permissions
3. Update Permission
4. Delete Permission
5. Back

C. Process Flow

1. **Admin Login:**
    - Admin enters credentials → validated via AdminDAO → access granted.
2. **Dashboard Menu:**
    - Admin selects which module to manage (Admin, User, Role, Permission).
3. **CRUD Operations:**
    - **Add:** Admin enters details → validated → saved in MySQL via Hibernate.
    - **View:** System fetches records from database → displays in console.
    - **Update:** Admin selects a record → modifies data → updated in database.
    - **Delete:** Admin selects a record → confirms deletion → removed from database.

4. **Role Assignment:**
    - Users are assigned roles → inherit permissions associated with the role.
5. **Permission Mapping:**
    - Permissions can be defined and later mapped to roles → controlling access.

D. Advantages of This Design

- **Modular:** Each entity has its own DAO and service class → easy maintenance.
- **Secure:** Only admin can perform operations; RBAC can be implemented for users.
- **Scalable:** Additional modules, roles, or permissions can be added easily.
- **User-Friendly:** Simple console menu ensures easy navigation and clear prompts.
- **Database-Driven:** Hibernate ORM ensures reliable, efficient database operations.

E. Screenshots

```
=== USER MANAGEMENT SYSTEM ===
1. Admin Management
2. User Management
3. Role Management
4. Permission Management
5. Exit
Enter choice: 1

=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back
Enter choice: 1
Enter name: Pushpa
Enter email: pushpa@gmail.com
Enter password: pushpa@999
```

```
Nov 04, 2025 9:22:34 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate ORM core version 6.4.4.Final
Nov 04, 2025 9:22:34 PM org.hibernate.cache.internal.RegionFactoryInitiator initiateService
INFO: HHH000026: Second-level cache disabled
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using built-in connection pool (not intended for production use)
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: Loaded JDBC driver class: com.mysql.cj.jdbc.Driver
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001012: Connecting with JDBC URL [jdbc:mysql://localhost:3306/userdb?useSSL=false&serverTimezone=UTC]
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {user=root, password=****}
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH10001115: Connection pool size: 20 (min=1)
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.dialect.internal.DialectFactoryImpl constructDialect
WARN: HHH90000025: MySQL8Dialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selec
Nov 04, 2025 9:22:34 PM org.hibernate.engine.jdbc.dialect.internal.DialectFactoryImpl logSelectedDialect
WARN: HHH90000026: MySQL8Dialect has been deprecated; use org.hibernate.dialect.MySQLDialect instead
Nov 04, 2025 9:22:35 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
Nov 04, 2025 9:22:35 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJ
☑ Admin added successfully!
```

```
=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back
Enter choice: 1
Enter name: Deepak
Enter email: deepak@gmail.com
Enter password: Deepak@888
☑ Admin added successfully!

=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back
Enter choice: 2
1 - Pushpa (pushpa@gmail.com)
2 - Deepak (deepak@gmail.com)

=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back
Enter choice: 3
Enter Admin ID to update: 2
Enter new name: Deepak
Enter new email: deepak24@gmail.com
Enter new password: Deepak@444
☑ Admin updated successfully!
```

```
=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back
Enter choice: 2
1 - Pushpa (pushpa@gmail.com)
2 - Deepak (deepak24@gmail.com)

=== ADMIN MANAGEMENT ===
1. Add Admin
2. View All Admins
3. Update Admin
4. Delete Admin
5. Back
Enter choice: 5

=== USER MANAGEMENT SYSTEM ===
1. Admin Management
2. User Management
3. Role Management
4. Permission Management
5. Exit
Enter choice: 2

=== USER MANAGEMENT ===
1. Add User
2. View All Users
3. Update User
4. Delete User
5. Back
Enter choice: 1
Enter first name: Shreya
Enter last name: Shetty
Enter email: shreya@gmail.com
Enter password: shreya@555
Enter phone: 1234567899
Enter created_by (admin ID): 1
☑ User added successfully!

=== USER MANAGEMENT ===
1. Add User
2. View All Users
3. Update User
4. Delete User
5. Back
Enter choice: ⚠ Input cannot be empty! Enter again: 2
1 - Shreya Shetty (shreya@gmail.com)

=== USER MANAGEMENT ===
1. Add User
2. View All Users
3. Update User
4. Delete User
5. Back
Enter choice: 5
```

```
=== USER MANAGEMENT SYSTEM ===
1. Admin Management
2. User Management
3. Role Management
4. Permission Management
5. Exit
Enter choice: 3

=== ROLE MANAGEMENT ===
1. Add Role
2. View All Roles
3. Update Role
4. Delete Role
5. Back
Enter choice: 1
Enter role name: Manager
Enter description: Manage users
☑ Role added successfully!

=== ROLE MANAGEMENT ===
1. Add Role
2. View All Roles
3. Update Role
4. Delete Role
5. Back
Enter choice: 2

=== ALL ROLES ===
ID: 1 | Name: Manager | Description: Manage users

=== ROLE MANAGEMENT ===
1. Add Role
2. View All Roles
3. Update Role
4. Delete Role
5. Back
Enter choice: 5

=== USER MANAGEMENT SYSTEM ===
1. Admin Management
2. User Management
3. Role Management
4. Permission Management
5. Exit
Enter choice: 4

=== PERMISSION MANAGEMENT ===
1. Add Permission
2. View All Permissions
3. Update Permission
4. Delete Permission
5. Back
Enter choice: 1
Enter permission name: check users
Enter description: created user check
☑ Permission added successfully!
```

```
=== PERMISSION MANAGEMENT ===
1. Add Permission
2. View All Permissions
3. Update Permission
4. Delete Permission
5. Back
Enter choice: 2
1 - check users (created user check)

=== PERMISSION MANAGEMENT ===
1. Add Permission
2. View All Permissions
3. Update Permission
4. Delete Permission
5. Back
Enter choice: 5

=== USER MANAGEMENT SYSTEM ===
1. Admin Management
2. User Management
3. Role Management
4. Permission Management
5. Exit
Enter choice: 5
Exiting...
```

4. Implementation

The implementation phase involves translating the system design into a working software product. The **User Management System** was developed using **Java** for application logic, **Hibernate ORM** for database interaction, **MySQL** as the database, and **Maven** for dependency management. The system is console-based and menu-driven, allowing admins to manage users, roles, and permissions efficiently.

1. Technology Stack

Programming Language: Java (Core Java)

- **ORM Framework:** Hibernate
- **Database:** MySQL
- **Build & Dependency Management:** Maven
- **User Interface:** Console-based
- **Libraries & Tools:**
  - Hibernate Core
  - MySQL Connector/J
  - SLF4J / Log4j (for logging)

2. Backend Implementation

The backend handles authentication, role assignment, permission mapping, and CRUD operations for all entities.

Modules:

1. **Admin Management:** Add, view, update, delete admins.
2. **User Management:** Add, view, update, delete users; assign roles.
3. **Role Management:** Add, view, update, delete roles; map permissions.
4. **Permission Management:** Add, view, update, delete permissions; assign to roles.

3. Database Implementation

The database was designed in **MySQL** with tables for each entity and relationships to ensure data integrity.

Table	Description
Admin	Stores admin credentials and profile
User	Stores user details and assigned roles
Role	Stores roles like Admin, Manager, Employee
Permission	Stores permissions like CREATE_USER, DELETE_USER
UserRole	Maps users to roles (Many-to-Many)
RolePermission	Maps roles to permissions (Many-to-Many)

Relationships:

- One **User** → Many **Roles** (via UserRole)
- One **Role** → Many **Permissions** (via RolePermission)

4. DAO (Data Access Object) Layer

Each entity has a corresponding DAO class to handle database operations:

- **AdminDAO:** Add, view, update, delete admins.
- **UserDAO:** Add, view, update, delete users; assign roles.
- **RoleDAO:** Add, view, update, delete roles; assign permissions.
- **PermissionDAO:** Add, view, update, delete permissions.

This modular design separates business logic from database operations, improving maintainability.

5. Security Implementation

- Password Handling: Passwords are stored securely.
- Role-Based Access Control (RBAC): Users inherit permissions from assigned roles.
- Data Integrity: Hibernate ensures safe CRUD operations and consistent database state.

5. Testing

The testing phase ensures that the **User Management System** functions correctly, is reliable, and meets the user requirements. Both **functional** and **non-functional** testing were performed to validate the application.

1. Testing Objectives

- Verify that all modules (Admin, User, Role, Permission management) work correctly.
- Ensure data consistency between the console interface, Java backend, and MySQL database.
- Validate security mechanisms such as password encryption and role-based access control.
- Check usability and proper navigation of the console menu system.

2. Types of Testing

a) Unit Testing

- Each DAO method and service method was tested individually.
- Examples:
    - AdminDAO.addAdmin() tested with valid and invalid admin details.
    - UserDAO.updateUser() tested for correct role assignment and invalid inputs.

b) Integration Testing

- Verified interaction between Java backend and MySQL database via Hibernate.
- Example: Adding a user through UserDAO.addUser() correctly reflects in the MySQL database.

c) System Testing

- The system was tested as a whole.
- Scenarios:
    - Admin creates a user → Assigns role → Assigns permissions → User inherits permissions correctly.
    - Admin tries to delete a role assigned to users → Appropriate error message displayed.
    - Admin updates permissions → Changes reflected for all users with that role.

d) Security Testing

- Verified that only authenticated admins can perform operations.
- Passwords stored in the database are hashed to ensure security.
- Users cannot access modules beyond their assigned roles.

e) Usability Testing

- Verified that the console interface is intuitive and easy to navigate.
- Checked menu flow for all modules (Admin, User, Role, Permission).
- Ensured proper confirmation messages and error prompts are displayed.

3. Test Cases

Test Case ID	Description	Input	Expected Output	Result
TC01	Admin Login	Valid credentials	Redirect to Main Menu	☑ Passed
TC02	Admin Login	Invalid credentials	Error message	☑ Passed
TC03	Add User	User details	User saved in DB	☑ Passed
TC04	Update User	Change role	User role updated in DB	☑ Passed
TC05	Delete User	Existing user	User removed from DB	☑ Passed
TC06	Add Role	Role details	Role saved in DB	☑ Passed
TC07	Assign Permission	Permission to role	Permission mapped correctly	☑ Passed
TC08	Unauthorized Access	No login / wrong role	Access denied	☑ Passed

4. Testing Tools

- **Eclipse / IntelliJ IDE –** Run and debug console application.
- **MySQL Workbench –** Verify database records and constraints.
- **JUnit –** Unit testing for DAO and service methods.
- **Manual Testing –** Step-by-step validation of console menu operations.

5. Test Results

- All core functionalities passed successfully.
- The system is stable, secure, and ready for deployment.
- Minor improvements were made in console prompts for better user experience.

6. Results and Discussion

The testing phase ensures that the **User Management System** functions correctly, is reliable, and meets the user requirements. Both **functional** and **non-functional** testing were performed to validate the application.

1. Results

Functional Results:

- **Admin Management:** Admins can securely log in and perform full CRUD operations on other admin accounts.
- **User Management:** Admins can add, view, update, and delete users. Users can be assigned roles, which automatically grant corresponding permissions.
- **Role Management:** Roles can be created, updated, deleted, and linked with specific permissions.
- **Permission Management:** Permissions can be managed and assigned to roles to ensure controlled access.
- **Access Control:** The system enforces role-based access control (RBAC), ensuring users can only perform authorized operations.

Non-Functional Results:

- **Usability:** The console-based menu system is intuitive, easy to navigate, and displays clear prompts for all operations.
- **Performance:** Database operations are executed efficiently using Hibernate ORM, ensuring smooth and quick responses.
- **Security:** Passwords are stored in hashed format, and access is restricted based on roles.
- **Scalability:** The modular DAO structure allows for easy addition of new modules, roles, or permissions in the future.

2. Discussion

1. Achievement of Objectives:

- All key project objectives—automated management of users, roles, and permissions, secure login, and enforcement of RBAC—were successfully achieved.
- Integration between the entities (User ↔ Role ↔ Permission) ensures that access rights are dynamically updated and maintained consistently in the database.

2. User Experience:

- The hierarchical menu structure allows admins to perform operations efficiently.
- Clear confirmation messages and error handling improve interaction and reduce errors.

3. Challenges Faced:

- Mapping many-to-many relationships between Users, Roles, and Permissions in Hibernate required careful configuration.
- Ensuring data consistency during updates and deletions, especially when roles or permissions are linked to multiple users.

4. Limitations:

- The system is console-based and lacks a graphical interface.
- Multi-user login (simultaneous use by different admins) is not implemented.
- Advanced auditing or activity tracking is not available.

5. Future Enhancements:

- Develop a GUI or web-based interface for better usability.
- Implement multi-user login with session management.
- Add audit logs to track user actions for improved accountability.
- Integrate email notifications for user account creation or role changes.
- Implement enhanced security features such as multi-factor authentication.

7. Conclusion and Future Scope

1. Conclusion

The development of the **User Management System** has successfully demonstrated the design and implementation of a practical, secure, and modular system for managing users, roles, and permissions. The system enables administrators to:

- Manage user accounts efficiently with full CRUD operations.
- Assign roles to users and map permissions to roles, ensuring secure role-based access control (RBAC).
- Maintain centralized control over all entities in the system through a console-based interface.
- Ensure data integrity and security using Hibernate ORM for database interaction and hashed password storage.

Key achievements include:

- Centralized and automated user management, replacing manual record-keeping.
- Modular DAO architecture that separates business logic from database operations for better maintainability.
- Role-based access control implementation that enforces proper authorization for each user.
- Secure, reliable, and scalable system with clear console-based navigation and user prompts.

Overall, the **User Management System** provides a strong foundation for handling user authentication, authorization, and administration in enterprise, educational, or corporate environments. It demonstrates real-world application of RBAC, ORM, and secure programming practices.

2. Future Scope

Although the current system fulfils its primary objectives, several enhancements can make it more robust, user-friendly, and suitable for real-world deployment:

1. **GUI or Web Interface**
    - Develop a graphical user interface or web-based application to improve usability.
2. **Multi-User Access & Session Management**
    - Allow multiple admins/users to use the system simultaneously with session tracking.
3. **Audit Logging & Activity Tracking**
    - Implement logs to monitor user activities for accountability and security.
4. **Advanced Security Features**
    - Introduce multi-factor authentication, password recovery, and stronger encryption.
5. **Role Hierarchy & Permission Templates**
    - Support hierarchical roles and reusable permission templates for large organizations.
6. **Integration with Enterprise Systems**
    - Connect with LDAP, Active Directory, or other external authentication services.
7. **Database & Technology Upgrades**
    - Migrate to a more scalable database or implement cloud-based deployment for wider accessibility.
8. **Notification & Alerts**
    - Notify admins about critical events like unauthorized access attempts or role changes.

By implementing these enhancements, the system can evolve into a fully featured enterprise-grade **User Management System** capable of handling complex organizational requirements.

8. Bibliography and References

In preparing this project report on **User Management System**, various books, research papers, articles, and online resources were consulted. These references helped in understanding the concepts of system design, database management, and role-based access control (RBAC) implementation.

Books

1. **Rajaraman, V.** – *Fundamentals of Computers*, Prentice Hall of India.
2. **Abraham Silberschatz, Henry Korth, S. Sudarshan** – *Database System Concepts*, McGraw Hill.
3. **Roger S. Pressman** – *Software Engineering: A Practitioner's Approach*, McGraw Hill.
4. **Ian Sommerville** – *Software Engineering*, Pearson Education.

Research Papers / Journals

1. **International Journal of Computer Applications (IJCA)** – *User Management Systems: A Review of Best Practices and Tools*.
2. **IEEE Xplore Digital Library** – Articles on role-based access control (RBAC) and user authentication systems.

Websites

1. https://hibernate.org/ – Official Hibernate documentation.
2. https://www.mysql.com/ – Official MySQL documentation.
3. https://spring.io/ – Spring Framework documentation (used for backend structure, if applicable).
4. https://docs.oracle.com/en/java/ – Official Java documentation.
5. https://www.mvnrepository.com/ – Repository for Maven dependencies.

Other Resources

- **Lecture notes, project guidelines, and practical references** provided during coursework.
- **Discussions with mentors and peers** that contributed to system requirements, design, and testing insights.