BUILDER PORTFOLIO MANAGEMENT SYSTEM (BPMS)
Project Report

1. Introduction

The **Builder Portfolio Management System (BPMS)** is a console-based Java application developed to assist a Builders' Association in efficiently managing construction project portfolios. The system replaces traditional spreadsheet-based workflows with a structured, role-driven application that enables project lifecycle management, task tracking, and persistent data storage using JSON.

BPMS is designed with modular architecture, role-based access control, and strong validation mechanisms to ensure data consistency, security, and scalability.

2. Problem Statement

The Builders' Association previously relied on manual spreadsheet-based methods for managing project records, resulting in several challenges:

- Inconsistent and fragmented data storage
- High probability of human error
- Inefficient coordination among stakeholders
- Absence of access control, allowing unauthorized data modifications

These limitations necessitated the development of a centralized, structured system capable of enforcing access rules, maintaining persistent records, and supporting the complete project lifecycle.

3. Objectives

The primary objectives of the BPMS are:

1. To develop a Java-based console application for construction project management.
2. To implement secure user authentication and role-based navigation.
3. To support four distinct user roles: **Admin, Manager, Client, and Builder**.
4. To manage the full project lifecycle, including creation, approval, task assignment, and completion tracking.
5. To ensure persistent storage of data using JSON files.
6. To achieve extensive unit test coverage across all core services.

4. Technology Stack

Component	Technology / Version
Programming Language	Java 17
Build Tool	Apache Maven
JSON Serialization	Jackson Databind 2.17.2
Date & Time Support	Jackson Datatype JSR-310 2.17.2
Unit Testing Framework	JUnit Jupiter 5.10.0
Test Suite Runner	JUnit Platform Suite 1.10.1

5. System Architecture

The BPMS follows a **layered Service–DAO architecture**, ensuring clear separation of concerns and maintainability.Architecture Layers

- **Presentation Layer**
  Console-based user interfaces for each role (Admin, Client, Manager, Builder).
- **Service Layer**
  Contains business logic for authentication, project handling, and task management.
- **Data Access Layer**
  Manages JSON-based persistence through a generic file handling service.
- **Data Storage Layer**
  Stores application data in JSON files.

Key Design Decisions

- Polymorphic user serialization using Jackson `@JsonTypeInfo`.
- Generic `FileService` leveraging Java generics for type-safe I/O.
- Static counters for auto-incremented entity IDs.
- Use of `Set<String>` references to prevent deep object nesting.

6. Module Description

Module	Files	Purpose
Model Layer	User, Client, Builder, Manager, Project, Task, ROLE, STATUS	Core entities and enumerations
Authentication Service	Login, Register	User registration and authentication
Project Service	ProjectService	Project creation, approval, assignment, and queries
Manager Service	ManagerService	Task creation, assignment, and project tracking
Task Service	CreateTask, AssignTaskToBuilder, UpdateTaskStatus	Task-level operations
File Service	FileService	Generic JSON persistence
Utility	Utility	Input validation and parsing
UI Layer	MainMenuUI, AdminUI, ClientUI, ManagerUI, BuilderUI	Console-based interaction
Exceptions	9 custom exception classes	Domain-specific error handling

7. Data Model DesignEnumerations

- `ROLE`: ADMIN, MANAGER, CLIENT, BUILDER
- `STATUS`: NOT_APPROVED, UPCOMING, IN_PROGRESS, COMPLETED

User Hierarchy

- **User (abstract)**: id, name, password, role
  - Client → projectList
  - Manager → projectList
  - Builder → taskList

Project Entity

- id, name, description, clientName, status, startDate, endDate
- managerList, taskList
- Default status: **NOT_APPROVED**

Task Entity

- id, name, description, projectName, managerName
- startDate, endDate, status, builderList
- Default status: **UPCOMING**

Relationships

- Client → Project (One-to-Many)
- Manager → Project (Many-to-Many)
- Project → Task (One-to-Many)
- Task → Builder (Many-to-Many)

8. Role-Based Access Control

Role	**Authorized Operations**
Admin	Approve projects, assign managers, view project lists
Client	Create project requests, view project status board
Manager	Create tasks, assign builders, update task status
Builder	View assigned in-progress tasks

Note: The Admin role uses hardcoded credentials and is excluded from persistent storage.

9. Application Workflow

Step	Actor	Action	Status
1	User	Registration	—
2	User	Login	—
3	Client	Create project	NOT_APPRO…
4	Admin	Approve project	UPCOMING
5	Admin	Assign manager	—
6	Manager	Create tasks	IN_PROGRESS
7	Manager	Assign builders	—
8	Builder	View tasks	—
9	Manager	Complete tasks	COMPLETED
10	Client	Track project	—

10. Exception Handling Strategy

All custom exceptions extend `Exception` and are categorized by domain:

- **Authentication Exceptions**
  InvalidPasswordException, UserNotFoundException
- **Project Exceptions**
  ProjectAlreadyExistsException, ProjectDoesNotExistException, RoleMismatchException
- **Task Exceptions**
  TaskAlreadyExistsException, TaskNotFoundException, InvalidTaskException

Additionally, `IllegalArgumentException` is used for centralized input validation.

11. Data Persistence Mechanism

- **Storage Format:** JSON
- **Library:** Jackson ObjectMapper with JavaTimeModule

File	Key	Value
users.json	username	User

projects.json	project name	Project
tasks.json	task name	Task

The `FileService` class provides reusable generic methods for loading and saving data with automatic path resolution via `pom.xml`.

12. Testing Summary

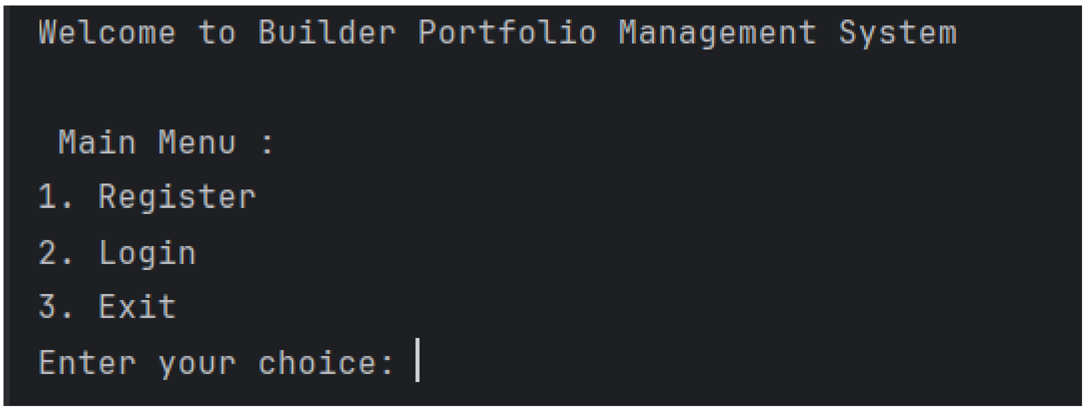
- **Framework:** JUnit Jupiter
- **Total Tests:** 46+
- **Coverage:** Authentication, Project Services, Task Services

Testing Strategy

- Database reset using `@BeforeEach` for isolation
- Shared fixtures via `@BeforeAll`
- Validation of edge cases, duplicates, null inputs, and role mismatches

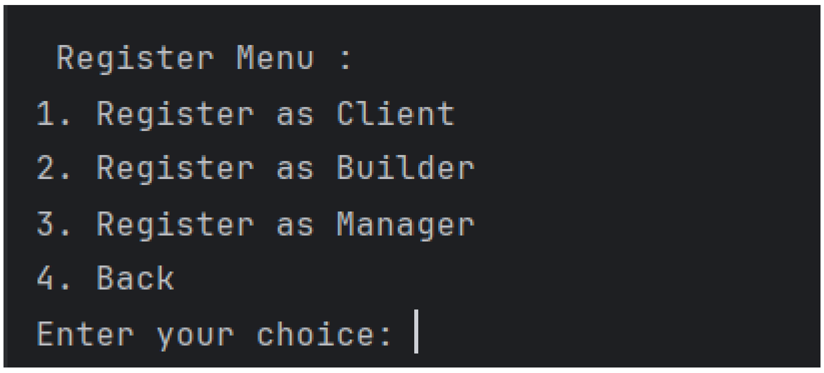
13. Output Screenshots

Main menu:

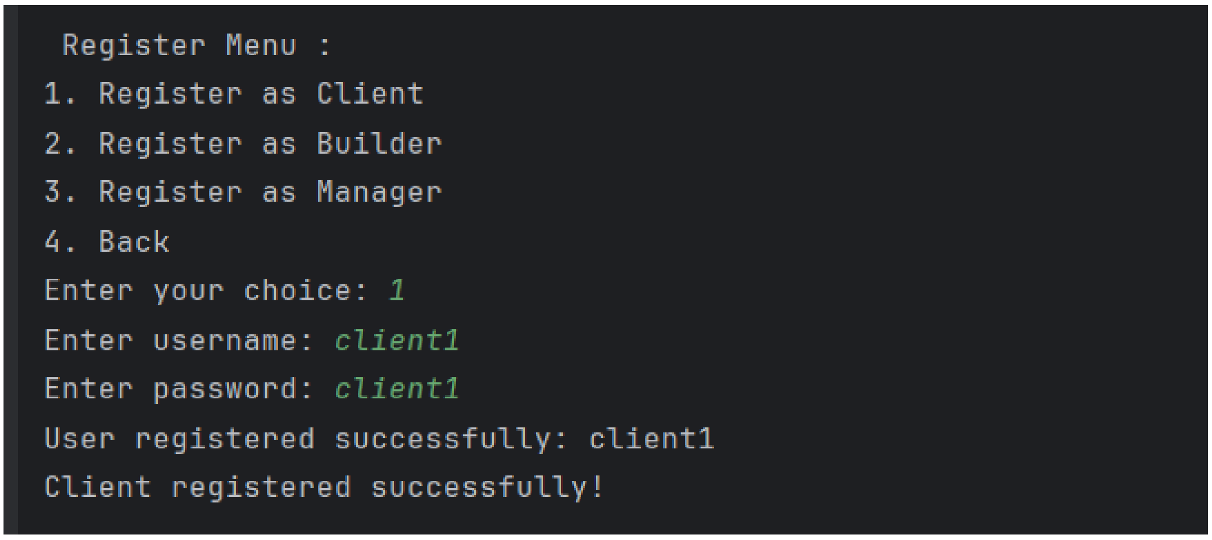


```
Welcome to Builder Portfolio Management System

 Main Menu :
1. Register
2. Login
3. Exit
Enter your choice: 
```

Register Menu:

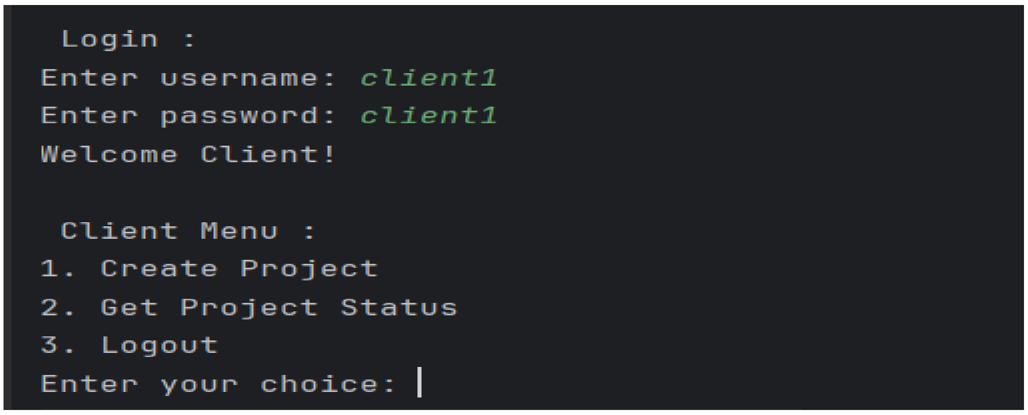


```
 Register Menu :
1. Register as Client
2. Register as Builder
3. Register as Manager
4. Back
Enter your choice: 
```



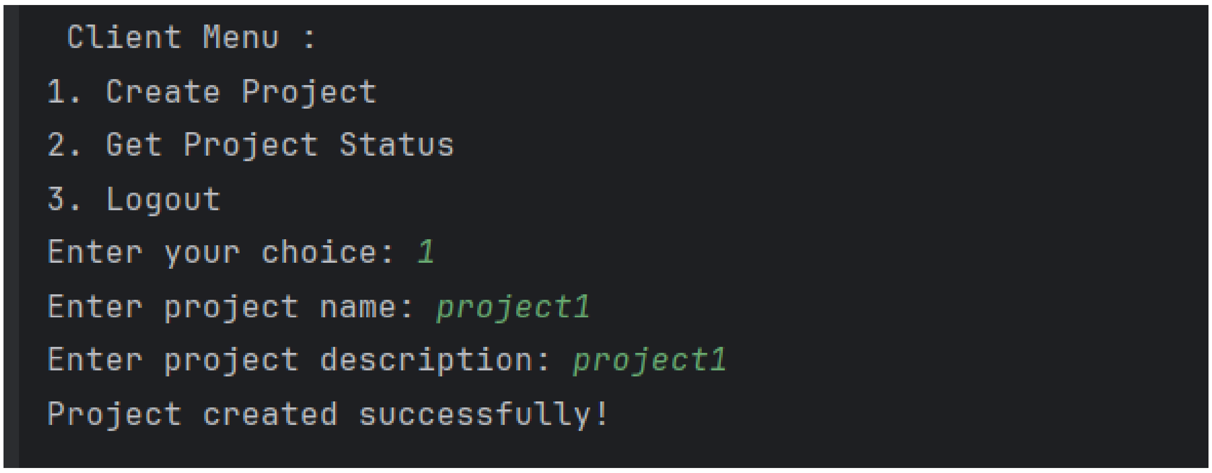
```
 Register Menu :
1. Register as Client
2. Register as Builder
3. Register as Manager
4. Back
Enter your choice: 1
Enter username: client1
Enter password: client1
User registered successfully: client1
Client registered successfully!
```

Client menu:

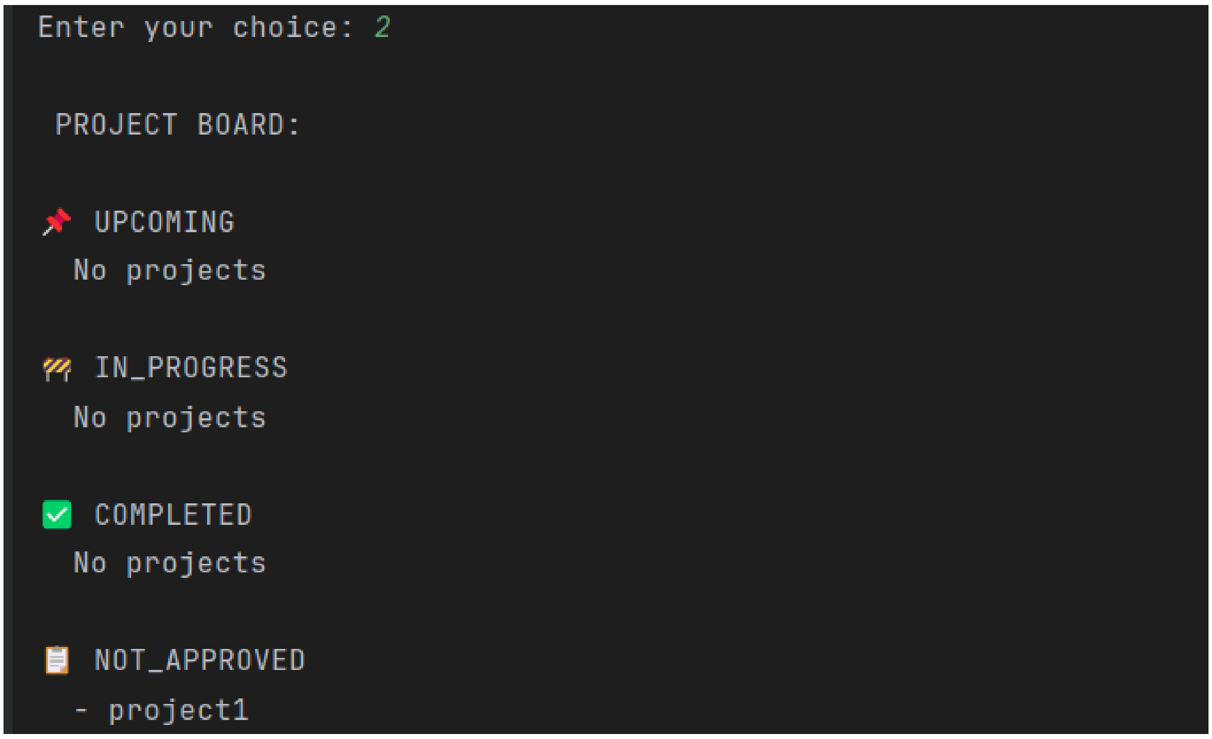


```
 Login :
Enter username: client1
Enter password: client1
Welcome Client!

 Client Menu :
1. Create Project
2. Get Project Status
3. Logout
Enter your choice: 
```

Create Project and get project status:

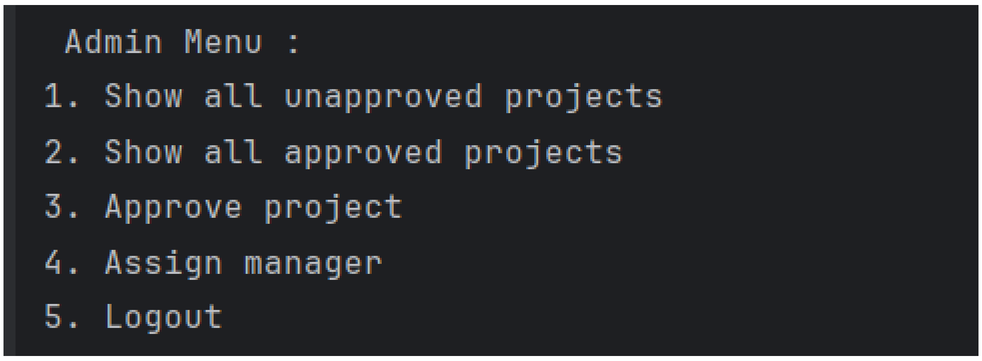


```
 Client Menu :
1. Create Project
2. Get Project Status
3. Logout
Enter your choice: 1
Enter project name: project1
Enter project description: project1
Project created successfully!
```



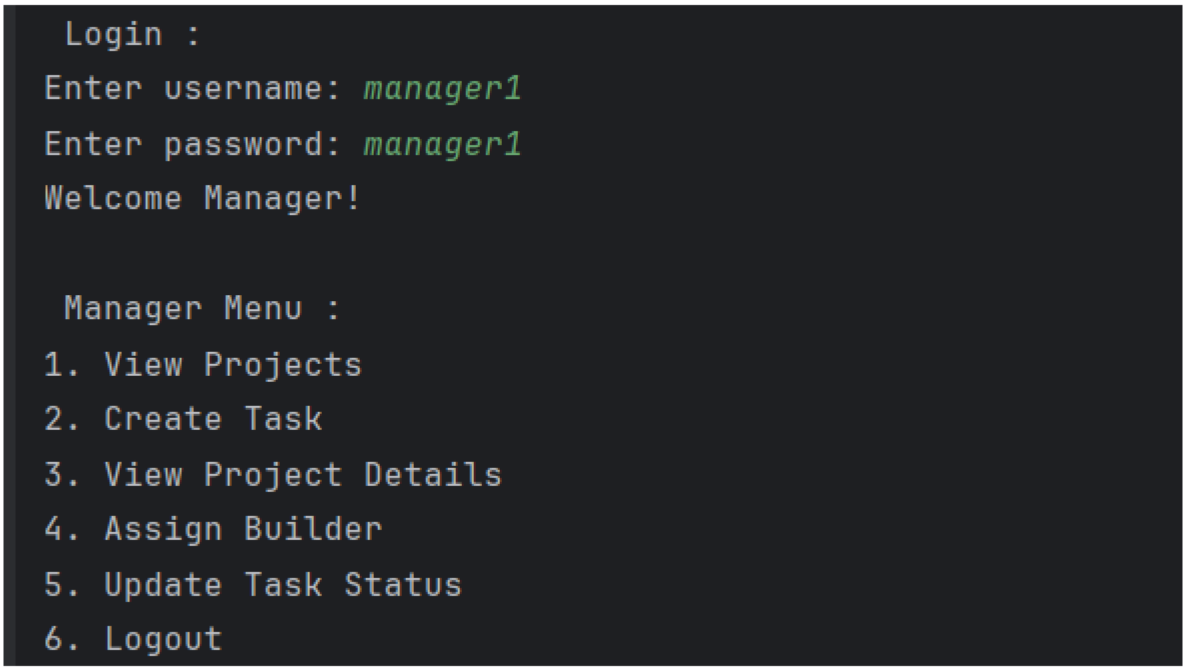
```
Enter your choice: 2

 PROJECT BOARD:

📌 UPCOMING
  No projects

🚧 IN_PROGRESS
  No projects

✅ COMPLETED
  No projects

📋 NOT_APPROVED
  - project1
```

Admin menu:



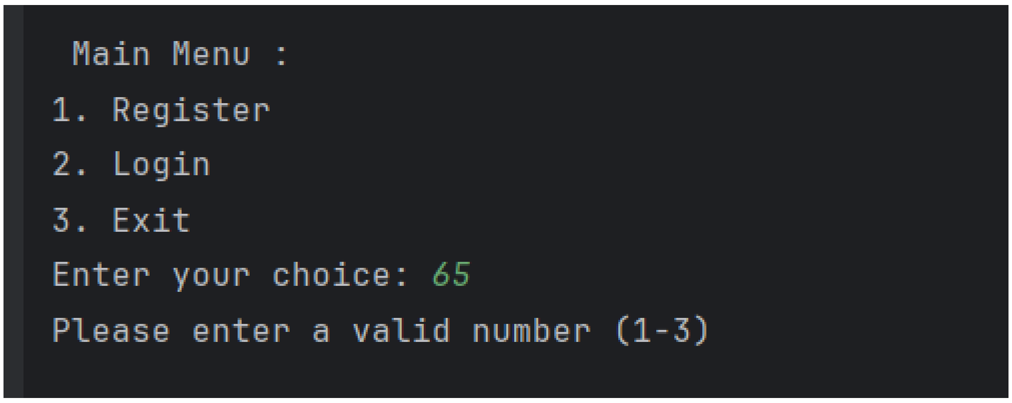
```
 Admin Menu :
1. Show all unapproved projects
2. Show all approved projects
3. Approve project
4. Assign manager
5. Logout
```

Manager menu:



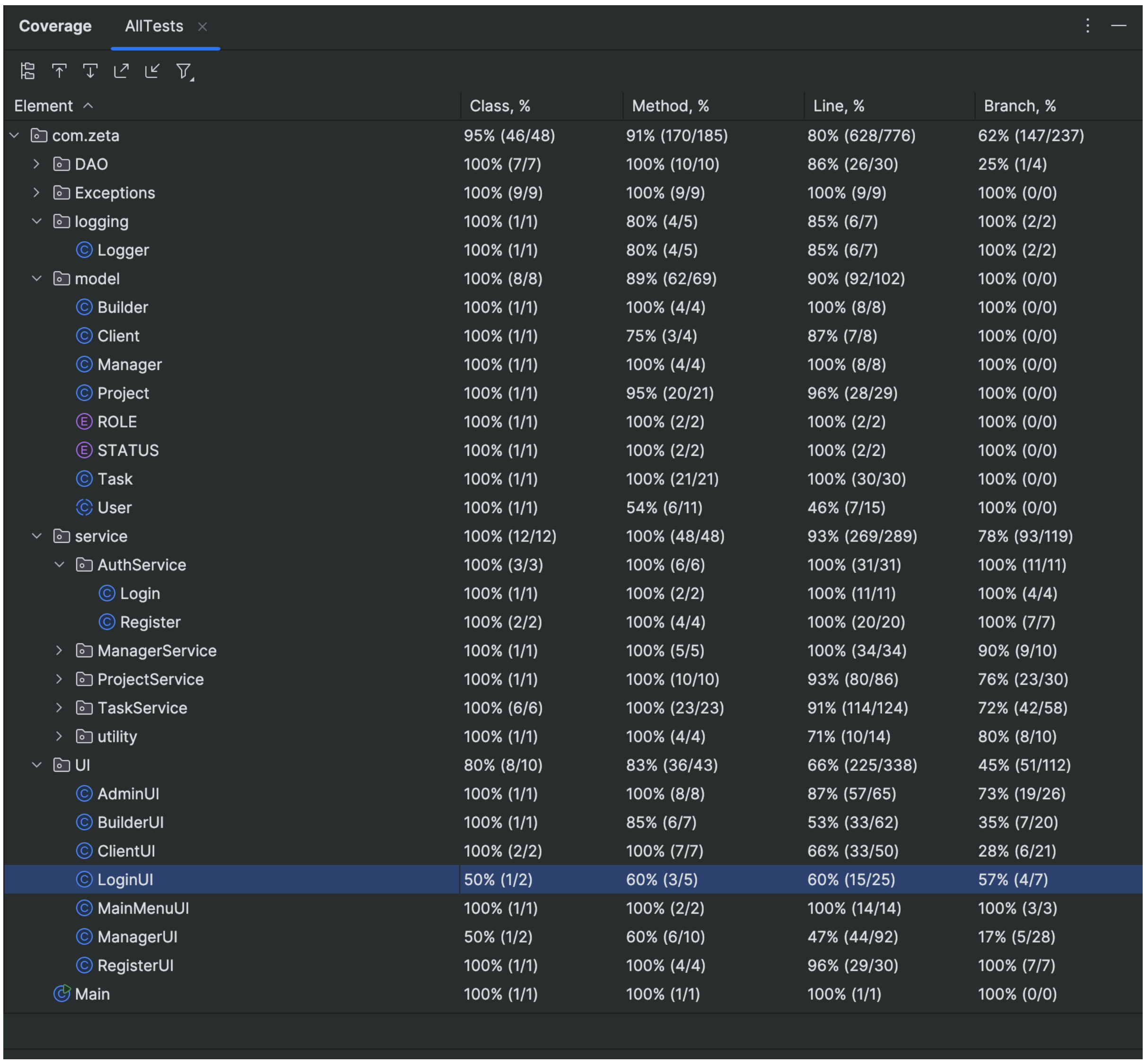
```
 Login :
Enter username: manager1
Enter password: manager1
Welcome Manager!

 Manager Menu :
1. View Projects
2. Create Task
3. View Project Details
4. Assign Builder
5. Update Task Status
6. Logout
```

Handle invalid inputs:



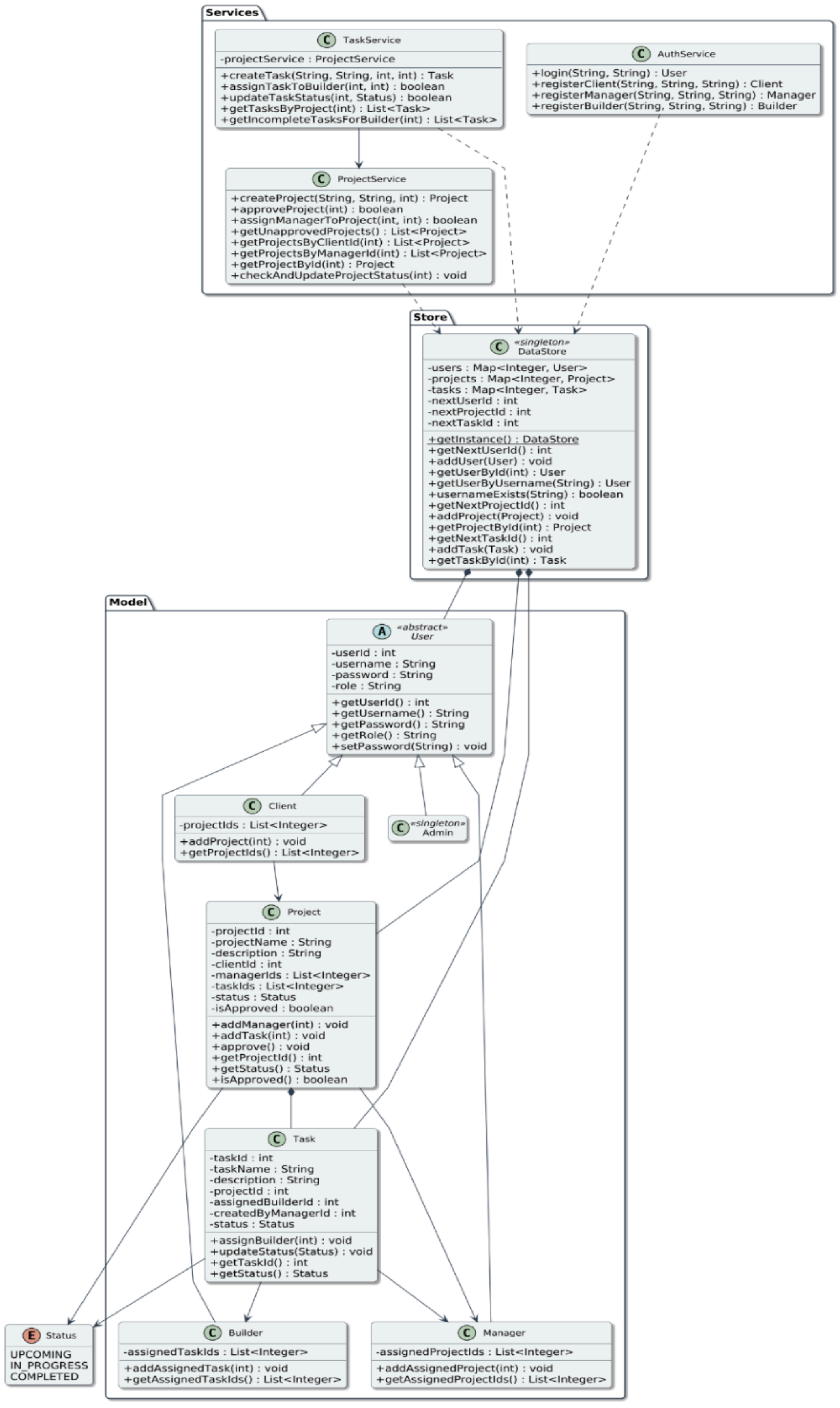
```
 Main Menu :
1. Register
2. Login
3. Exit
Enter your choice: 65
Please enter a valid number (1-3)
```

Code Coverage:

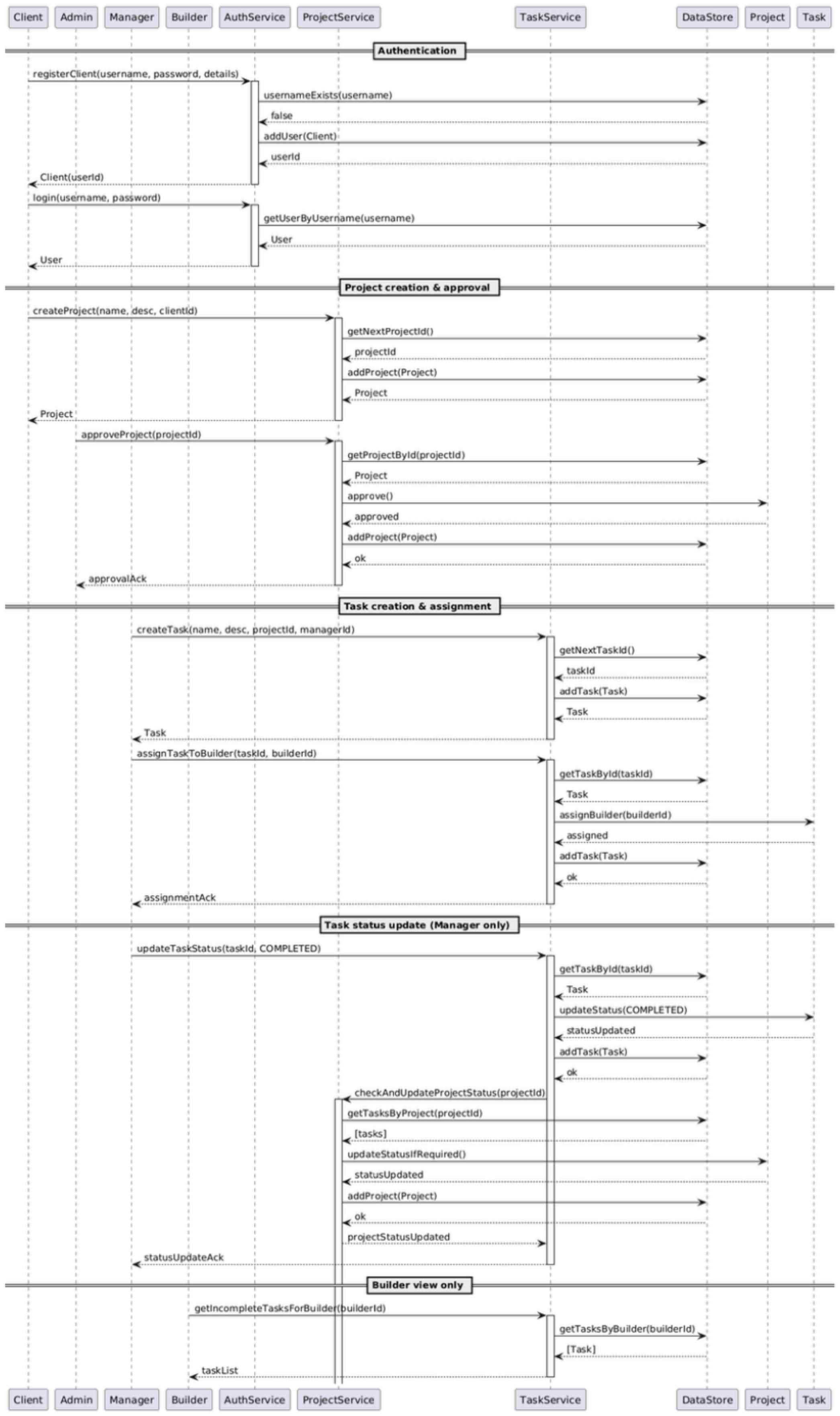


Element	Class, %	Method, %	Line, %	Branch, %
com.zeta	95% (46/48)	91% (170/185)	80% (628/776)	62% (147/237)
DAO	100% (7/7)	100% (10/10)	86% (26/30)	25% (1/4)
Exceptions	100% (9/9)	100% (9/9)	100% (9/9)	100% (0/0)
logging	100% (1/1)	80% (4/5)	85% (6/7)	100% (2/2)
Logger	100% (1/1)	80% (4/5)	85% (6/7)	100% (2/2)
model	100% (8/8)	89% (62/69)	90% (92/102)	100% (0/0)
Builder	100% (1/1)	100% (4/4)	100% (8/8)	100% (0/0)
Client	100% (1/1)	75% (3/4)	87% (7/8)	100% (0/0)
Manager	100% (1/1)	100% (4/4)	100% (8/8)	100% (0/0)
Project	100% (1/1)	95% (20/21)	96% (28/29)	100% (0/0)
ROLE	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
STATUS	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
Task	100% (1/1)	100% (21/21)	100% (30/30)	100% (0/0)
User	100% (1/1)	54% (6/11)	46% (7/15)	100% (0/0)
service	100% (12/12)	100% (48/48)	93% (269/289)	78% (93/119)
AuthService	100% (3/3)	100% (6/6)	100% (31/31)	100% (11/11)
Login	100% (1/1)	100% (2/2)	100% (11/11)	100% (4/4)
Register	100% (2/2)	100% (4/4)	100% (20/20)	100% (7/7)
ManagerService	100% (1/1)	100% (5/5)	100% (34/34)	90% (9/10)
ProjectService	100% (1/1)	100% (10/10)	93% (80/86)	76% (23/30)
TaskService	100% (6/6)	100% (23/23)	91% (114/124)	72% (42/58)
utility	100% (1/1)	100% (4/4)	71% (10/14)	80% (8/10)
UI	80% (8/10)	83% (36/43)	66% (225/338)	45% (51/112)
AdminUI	100% (1/1)	100% (8/8)	87% (57/65)	73% (19/26)
BuilderUI	100% (1/1)	85% (6/7)	53% (33/62)	35% (7/20)
ClientUI	100% (2/2)	100% (7/7)	66% (33/50)	28% (6/21)
LoginUI	50% (1/2)	60% (3/5)	60% (15/25)	57% (4/7)
MainMenuUI	100% (1/1)	100% (2/2)	100% (14/14)	100% (3/3)
ManagerUI	50% (1/2)	60% (6/10)	47% (44/92)	17% (5/28)
RegisterUI	100% (1/1)	100% (4/4)	96% (29/30)	100% (7/7)
Main	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)

Class Diagram:


Services
TaskService
-projectService : ProjectService
+createTask(String, String, int, int) : Task
+assignTaskToBuilder(int, int) : boolean
+updateTaskStatus(int, Status) : boolean
+getTasksByProject(int) : List<Task>
+getIncompleteTasksForBuilder(int) : List<Task>
AuthService
+login(String, String) : User
+registerClient(String, String, String) : Client
+registerManager(String, String, String) : Manager
+registerBuilder(String, String, String) : Builder
ProjectService
+createProject(String, String, int) : Project
+approveProject(int) : boolean
+assignManagerToProject(int, int) : boolean
+getUnapprovedProjects() : List<Project>
+getProjectsByClientId(int) : List<Project>
+getProjectsByManagerId(int) : List<Project>
+getProjectById(int) : Project
+checkAndUpdateProjectStatus(int) : void
Store
«singleton»
DataStore
-users : Map<Integer, User>
-projects : Map<Integer, Project>
-tasks : Map<Integer, Task>
-nextUserId : int
-nextProjectId : int
-nextTaskId : int
+getInstance() : DataStore
+getNextUserId() : int
+addUser(User) : void
+getUserById(int) : User
+getUserByUsername(String) : User
+usernameExists(String) : boolean
+getNextProjectId() : int
+addProject(Project) : void
+getProjectById(int) : Project
+getNextTaskId() : int
+addTask(Task) : void
+getTaskById(int) : Task
Model
«abstract»
User
-userId : int
-username : String
-password : String
-role : String
+getUserId() : int
+getUsername() : String
+getPassword() : String
+getRole() : String
+setPassword(String) : void
Client
-projectIds : List<Integer>
+addProject(int) : void
+getProjectIds() : List<Integer>
«singleton»
Admin
Project
-projectId : int
-projectName : String
-description : String
-clientId : int
-managerIds : List<Integer>
-taskIds : List<Integer>
-status : Status
-isApproved : boolean
+addManager(int) : void
+addTask(int) : void
+approve() : void
+getProjectId() : int
+getStatus() : Status
+isApproved() : boolean
Task
-taskId : int
-taskName : String
-description : String
-projectId : int
-assignedBuilderId : int
-createdByManagerId : int
-status : Status
+assignBuilder(int) : void
+updateStatus(Status) : void
+getTaskId() : int
+getStatus() : Status
Status
UPCOMING
IN_PROGRESS
COMPLETED
Builder
-assignedTaskIds : List<Integer>
+addAssignedTask(int) : void
+getAssignedTaskIds() : List<Integer>
Manager
-assignedProjectIds : List<Integer>
+addAssignedProject(int) : void
+getAssignedProjectIds() : List<Integer>

Sequence Diagram:



14. Conclusion

The **Builder Portfolio Management System (BPMS)** successfully addresses the limitations of manual project management by delivering a robust, role-based solution. The system ensures data integrity, enforces access control, and supports end-to-end project lifecycle management.

Its modular architecture, extensive test coverage, and persistent JSON-based storage provide a scalable foundation for future enhancements, including database integration, concurrent usage, or migration to a web-based platform.