Builder Portfolio Management System – Complete Application Flow & Service Documentation

1. System Architecture (Layered Architecture)

The Builder Portfolio Management System follows a layered architecture to ensure clean separation of responsibilities.

Layers

- UI Layer – Handles user interaction and input/output.
- Service Layer – Contains business logic and rules.
- DAO Layer – Handles data persistence using JSON files.
- Model Layer – Represents entities such as User, Project, and Task.

Application Flow

User → UI → Service → DAO → JSON File Storage

Benefits

- Clean modular structure
- Easy testing
- Maintainable code
- Follows Single Responsibility Principle

2. Complete Application Flow

- User starts application.
- Main menu shows Register, Login, Exit options.
- User registers or logs in.
- System validates credentials.
- Based on role, user is routed to respective dashboard.
- Services process business logic.
- DAO saves or retrieves data from JSON storage.

Roles in System

- Admin – Approves projects, assigns managers, monitors system.
- Client – Creates projects and tracks progress.
- Manager – Manages project tasks and team.
- Builder – Works on assigned tasks.

3. Authentication Flow (Login Service)

Steps:

- User enters username and password.
- System validates input.
- If admin credentials (admin/admin) → Admin UI is loaded.
- Otherwise, Login service loads users from DAO.
- System verifies user existence.
- Password is validated.
- User role is returned.
- UI routes user to respective dashboard.

Exceptions

- UserNotFoundException – if user does not exist.
- InvalidPasswordException – if password mismatch.

4. Registration Flow

Steps:

- User enters details.
- System validates username and password.
- System checks duplicate username.
- New user object created.
- Data stored in JSON file through DAO.

Purpose

- Create new system users.
- Prevent duplicate accounts.

5. Project Service – Complete Logic

ProjectService handles all project related business operations.

Responsibilities

- Create project
- Approve project
- Assign manager
- Fetch projects by client
- Fetch projects by manager
- Track project status
- Filter approved/unapproved projects

5.1 Project Creation Flow

- Validate project name and description.
- Validate client existence.
- Check if project already exists.
- Assign project to client.
- Save project data using DAO.

Outcome: Project stored with NOT_APPROVED status.

5.2 Project Approval Flow (Admin Process)

- Admin selects project to approve.
- System verifies project exists.
- Admin enters start and end dates.
- System validates date order.
- Project status changed to UPCOMING.
- Project saved in storage.

Impact: Project becomes active and managers can start task creation.

5.3 Assign Manager Flow

- Validate project existence.
- Validate manager existence.
- Verify user role is MANAGER.
- Assign manager to project.
- Update manager project list.
- Save changes.

Purpose: Manager becomes responsible for project execution.

6. Manager Service Flow

ManagerService manages project execution operations.

Responsibilities

- View assigned projects.
- Monitor project progress.
- Create tasks.
- Track task completion.
- Group projects by status.

Manager Workflow

- Manager logs in.
- System shows assigned projects.
- Manager creates tasks for project.
- Manager assigns builders.
- Manager monitors progress.

7. Task Service – Task Lifecycle

TaskService handles full task lifecycle.

Responsibilities

- Create tasks.
- Assign tasks to builders.
- Update task status.
- Retrieve tasks by project.
- Track task progress.

7.1 Task Creation Flow

- Manager creates task for project.
- System validates project.
- Task details stored.
- Task linked to project.

7.2 Task Assignment Flow

- Manager selects task.
- Builder assigned to task.
- Builder added to task list.
- System saves mapping.

7.3 Task Status Update Flow

- Builder or manager updates status.
- System validates task.
- Status updated.
- Changes saved.

8. DAO Layer Logic

DAO (Data Access Object) handles data storage operations.

Responsibilities

- Load data from JSON files.
- Save data to JSON files.
- Maintain persistence for users, projects, and tasks.

Advantages

- Separation of business logic and storage logic.
- Easy replacement of storage system.

9. Logging System

Logger uses Singleton pattern.

Features

- Single logger instance.
- Info messages for normal flow.
- Warning messages for failures.
- Error messages for critical issues.

Benefits

- Centralized logging.
- Clean output management.

10. Design Principles Used

- Single Responsibility Principle – each class has one purpose.
- Separation of Concerns – UI, Service, DAO separated.
- Modular Design – independent components.
- Testability – services can be unit tested.
- Reusability – shared logic across modules.

Concurrency Support

Concurrency is enabled using Java NIO file locking, allowing multiple instances of the application to run simultaneously across different terminals while ensuring safe and synchronized access to shared data files.