Builder Portfolio Management System

1. Overview
This mini-project is a Project Management System where users can register as Admin, Project Manager, Builder, or Client. Each user has a role-specific menu to perform operations. All actions are persisted in a PostgreSQL database, with logs recorded via SLF4J and functionalities tested using JUnit.

2. User Registration & Login Flow
Step 1: User Registration

- UI Layer: User inputs details (username, email, password, role).
- Controller Layer: AuthController.register(user) receives the input.
- Service Layer: AuthService.register(user) validates input and checks email uniqueness.
- Repository Layer: UserRepository.save(user) persists user data in users table.
- Logging: Record user registration event.

  Outcome: User account created successfully.

Step 2: User Login

- UI Layer: User inputs email and password.
- Controller Layer: AuthController.login(email, password) authenticates.
- Service Layer: AuthService.login(email, password) validates credentials.
- Repository Layer: UserRepository.findByEmail(email) fetches user.

  Outcome: User redirected to role-specific menu.

3. Role-Specific Flows

3.1 ADMIN Flow

Menu Operations:

1. View All Projects
    - Controller: AdminController.viewAllProjects()
    - Repository: CommonRepository.viewAllProjects()
    - UI: Display projects in tabular format.

2. View Audit Trail
    - Controller: AdminController.viewAuditTrail()
    - UI: Display all system activity logs.

3. Delete Project Manager / Client / Builder
    - Controller: AdminController.deleteUser(userId)
    - Service: AdminService.deleteUser(userId)

- Repository: AdminRepository.deleteById(userId)
- Logging: Record deletion event.

4. Logout
    - Return to login screen.

3.2 BUILDER Flow

Menu Operations:

1. Add New Project
    - Controller: BuilderController.createProject(project)
    - Service: BuilderService.createProject(project)
    - Repository: BuilderRepository.createProject(project)
    - Logging: Record project creation.

2. Update Project
    - Controller: BuilderController.updateProject(project)
    - Service: BuilderService.updateProject(project)
    - Repository: BuilderRepository.update(project)
    - Logging: Record update event.

3. Delete Project
    - Controller: BuilderController.deleteProject(projectId)
    - Service: BuilderService.deleteProject(projectId)
    - Repository: BuilderRepository.delete(projectId) (check cascade rules)

4. Update Project Manager Assignment
    - Controller: BuilderController.updateProjectManager(projectId, managerId)
    - Service: BuilderService.assignManager(projectId, managerId)
    - Repository: BuilderRepository.updateManager(projectId, managerId)

5. Upload Project Documents
    - Controller: BuilderController.uploadDocument(projectId, document)
    - Service: DocumentService.uploadDocument(projectId, document)
    - Repository: BuilderRepository.save(document)

6. View Portfolio
    - Controller: BuilderController.viewPortfolio()
    - Service: BuilderService.getPortfolio(builderId)
    - Repository: BuilderRepository.findByBuilderId(builderId)

7. View Gantt Chart

    - Controller: BuilderController.viewGanttChart(projectId)
    - Service: BuilderService.getProjectTimeline(projectId)

8. Logout

- Return to login screen.

3.3 PROJECT MANAGER Flow

Menu Operations:

1. View Assigned Projects
    - Controller: PMController.viewAssignedProjects()
    - Service: PMService.getAssignedProjects(pmId)
    - Repository: PMRepository.findByManagerId(pmId)

2. Update Project Status
    - Controller: PMController.updateStatus(projectId, status)
    - Service: PMService.updateProjectStatus(projectId, status)
    - Repository: PMRepository.updateStatus(projectId, status)

3. Update Actual Project Spend
    - Controller: PMController.updateSpend(projectId, amount)
    - Service: PMService.updateActualSpend(projectId, amount)
    - Repository: PMRepository.updateActualSpend(projectId, amount)

4. Upload Project Documents (similar to Builder)

3.4 CLIENT Flow

Menu Operations:

1. View Owned Projects
    - Controller: ClientController.viewOwnedProjects(clientId)
    - Service: ClientService.getOwnedProjects(clientId)
    - Repository: ClientRepository.findByClientId(clientId)

2. View Budget Status
    - Controller: ClientController.viewBudgetStatus(projectId)
    - Service: ClientService.getBudgetStatus(projectId)
    - Repository: ClientRepository.getBudget(projectId)

3. View Documents
    - Controller: ClientController.viewDocuments(projectId)
    - Service: DocumentService.getDocuments(projectId)

4. View Timeline
    - Controller: ClientController.viewTimeline(projectId)
    - Service: ClientService.getProjectTimeline(projectId)

4. Logging & Testing

- Logging:
  - SLF4J logs all major actions: registration, login, CRUD operations, document uploads, assignments.
  - Example: logger.info("Project {} updated by builder {}", projectId, builderId);

- Testing:
  - JUnit tests for Service Layer methods.
  - Example: assertEquals(expectedProjectCount, projectService.getAllProjects().size());

5. Database Tables

- users(user_id, username, email, password, role)
- projects(project_id, project_name, builder_id, manager_id, client_id, planned_budget, actual_spend, status)
- documents(document_id, project_id, document_name, uploaded_by)
- notification(notification_id, user_id, message, recipient)

6. Audit Trail

Maintaining audit trail file where all actions are stored and can be viewed by admin

Example: User Logged In,1,Deep,BUILDER

7. Sample Flow Example

Scenario: Builder adds a project → assigns Project Manager → Project Manager updates spend → Client views budget.

1. Builder logs in → adds project → record saved in projects table → logged.
2. Builder assigns Project Manager → projects.manager_id updated.
3. Project Manager logs in → updates actual spend → projects.actual_spend updated.
4. Client logs in → views budget status → fetches data from projects table

Demo Link:
Miniproject Demo.mov

Github Repo Link:
https://github.com/DeepParekh03/MiniProject_BuilderPortfolio