Builder Portfolio Management System

UML Diagrams Documentation

Index

1. ER Diagram – Database Structure

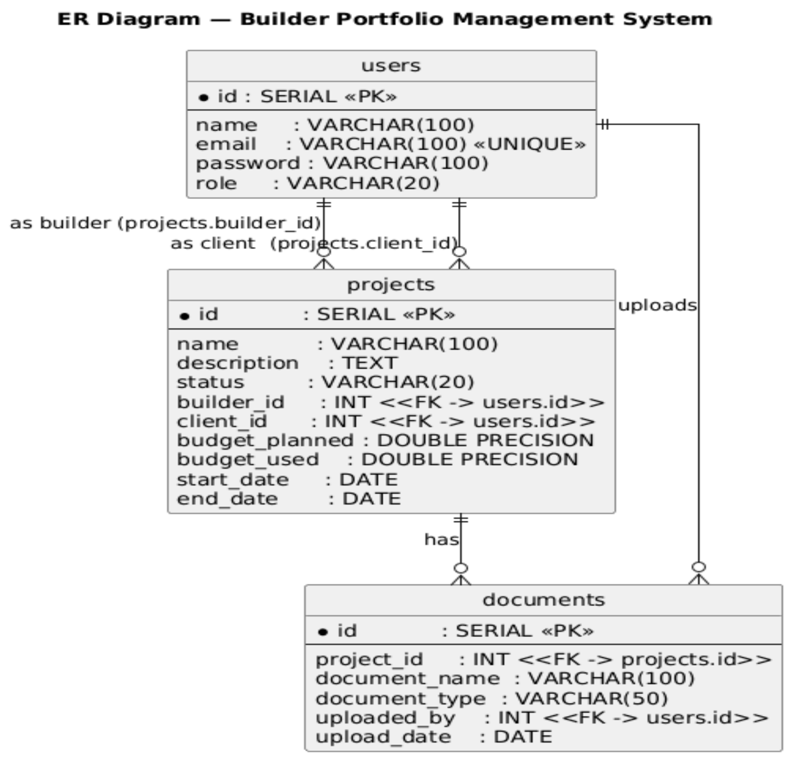
Purpose

The Entity-Relationship (ER) Diagram represents the logical structure of the system's database.
It shows how key entities such as **User**, **Project**, and **Document** are related and the cardinalities between them.

Description

- Each **User** can be a Builder, Client, or Admin.
- Each **Builder** can manage multiple **Projects**.
- Each **Project** can have multiple **Documents** linked to it.
- **Budget** and **Timeline** details are stored within the **Project** entity.
- Relationships are established using foreign keys (e.g., `builder_id`, `client_id`, `project_id`).

Diagram:



ER Diagram — Builder Portfolio Management System
users
id : SERIAL «PK»
name : VARCHAR(100)
email : VARCHAR(100) «UNIQUE»
password : VARCHAR(100)
role : VARCHAR(20)
as builder (projects.builder_id)
as client (projects.client_id)
uploads
projects
id : SERIAL «PK»
name : VARCHAR(100)
description : TEXT
status : VARCHAR(20)
builder_id : INT <<FK -> users.id>>
client_id : INT <<FK -> users.id>>
budget_planned : DOUBLE PRECISION
budget_used : DOUBLE PRECISION
start_date : DATE
end_date : DATE
has
documents
id : SERIAL «PK»
project_id : INT <<FK -> projects.id>>
document_name : VARCHAR(100)
document_type : VARCHAR(50)
uploaded_by : INT <<FK -> users.id>>
upload_date : DATE

2. Class Diagram – Application Architecture

Purpose

The Class Diagram illustrates the system's object-oriented structure and how different classes interact within the layered MVC + Service + DAO architecture.

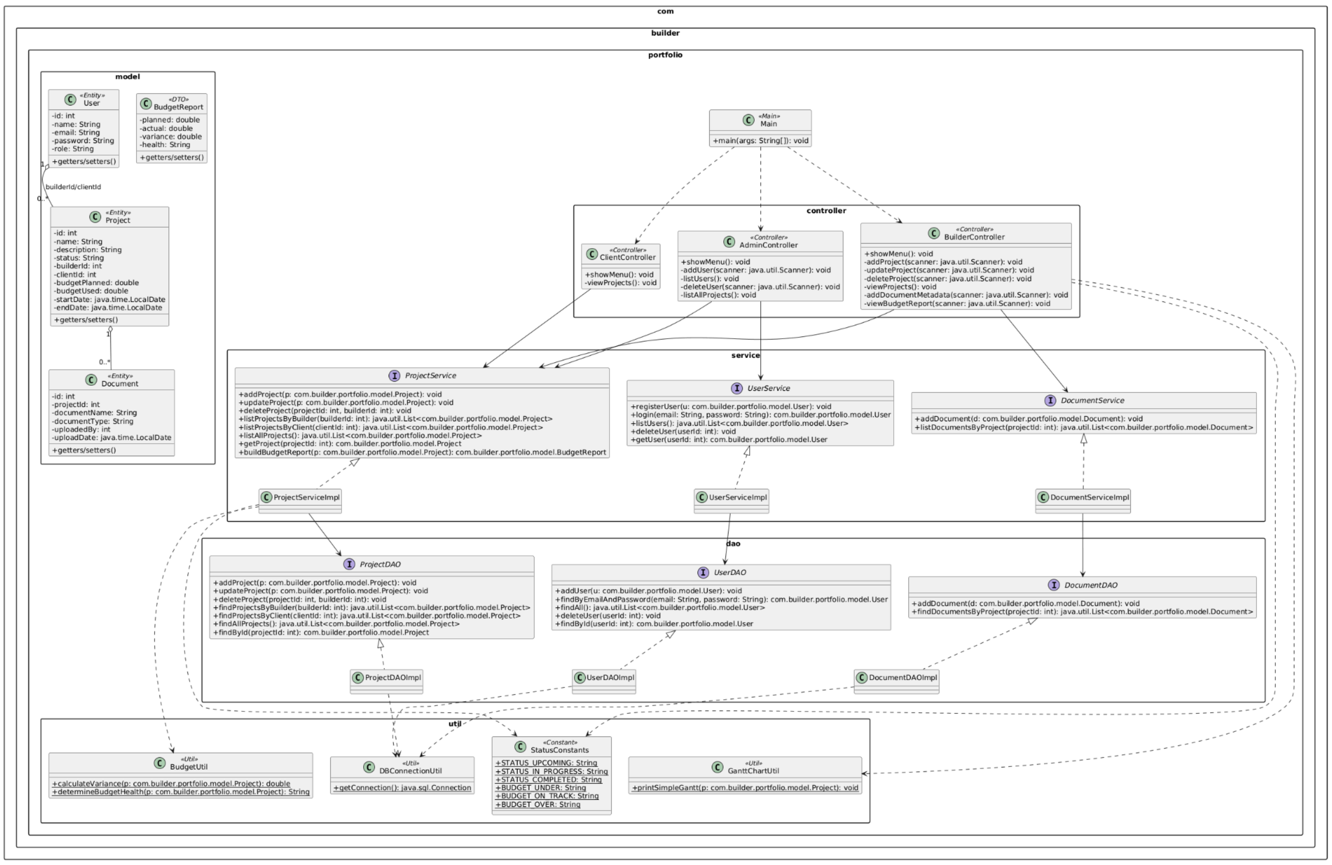
Description

- **Controllers** handle user interaction and forward requests to services.
- **Service classes** encapsulate core business logic.
- **DAO classes** interact with the database through JDBC.
- **Model classes** represent real-world entities (User, Project, Document, BudgetReport).
- **Utility classes** assist with common tasks such as DB connection and budget calculation.

Key Layers

- `com.builder.portfolio.controller`
- `com.builder.portfolio.service`
- `com.builder.portfolio.dao`
- `com.builder.portfolio.model`
- `com.builder.portfolio.util`

Diagram:


com
builder
portfolio
model
«Entity» User
-id: int
-name: String
-email: String
-password: String
-role: String
+getters/setters()
«DTO» BudgetReport
-planned: double
-actual: double
-variance: double
-health: String
+getters/setters()
1
builderId/clientId
0..*
«Entity» Project
-id: int
-name: String
-description: String
-status: String
-builderId: int
-clientId: int
-budgetPlanned: double
-budgetUsed: double
-startDate: java.time.LocalDate
-endDate: java.time.LocalDate
+getters/setters()
1
0..*
«Entity» Document
-id: int
-projectId: int
-documentName: String
-documentType: String
-uploadedBy: int
-uploadDate: java.time.LocalDate
+getters/setters()
«Main» Main
+main(args: String[]): void
controller
«Controller» ClientController
+showMenu(): void
-viewProjects(): void
«Controller» AdminController
+showMenu(): void
-addUser(scanner: java.util.Scanner): void
-listUsers(): void
-deleteUser(scanner: java.util.Scanner): void
-listAllProjects(): void
«Controller» BuilderController
+showMenu(): void
-addProject(scanner: java.util.Scanner): void
-updateProject(scanner: java.util.Scanner): void
-deleteProject(scanner: java.util.Scanner): void
-viewProjects(): void
-addDocumentMetadata(scanner: java.util.Scanner): void
-viewBudgetReport(scanner: java.util.Scanner): void
service
ProjectService
+addProject(p: com.builder.portfolio.model.Project): void
+updateProject(p: com.builder.portfolio.model.Project): void
+deleteProject(projectId: int, builderId: int): void
+listProjectsByBuilder(builderId: int): java.util.List<com.builder.portfolio.model.Project>
+listProjectsByClient(clientId: int): java.util.List<com.builder.portfolio.model.Project>
+listAllProjects(): java.util.List<com.builder.portfolio.model.Project>
+getProject(projectId: int): com.builder.portfolio.model.Project
+buildBudgetReport(p: com.builder.portfolio.model.Project): com.builder.portfolio.model.BudgetReport
UserService
+registerUser(u: com.builder.portfolio.model.User): void
+login(email: String, password: String): com.builder.portfolio.model.User
+listUsers(): java.util.List<com.builder.portfolio.model.User>
+deleteUser(userId: int): void
+getUser(userId: int): com.builder.portfolio.model.User
DocumentService
+addDocument(d: com.builder.portfolio.model.Document): void
+listDocumentsByProject(projectId: int): java.util.List<com.builder.portfolio.model.Document>
ProjectServiceImpl
UserServiceImpl
DocumentServiceImpl
dao
ProjectDAO
+addProject(p: com.builder.portfolio.model.Project): void
+updateProject(p: com.builder.portfolio.model.Project): void
+deleteProject(projectId: int, builderId: int): void
+findProjectsByBuilder(builderId: int): java.util.List<com.builder.portfolio.model.Project>
+findProjectsByClient(clientId: int): java.util.List<com.builder.portfolio.model.Project>
+findAllProjects(): java.util.List<com.builder.portfolio.model.Project>
+findById(projectId: int): com.builder.portfolio.model.Project
UserDAO
+addUser(u: com.builder.portfolio.model.User): void
+findByEmailAndPassword(email: String, password: String): com.builder.portfolio.model.User
+findAll(): java.util.List<com.builder.portfolio.model.User>
+deleteUser(userId: int): void
+findById(userId: int): com.builder.portfolio.model.User
DocumentDAO
+addDocument(d: com.builder.portfolio.model.Document): void
+findDocumentsByProject(projectId: int): java.util.List<com.builder.portfolio.model.Document>
ProjectDAOImpl
UserDAOImpl
DocumentDAOImpl
util
«Util» BudgetUtil
+calculateVariance(p: com.builder.portfolio.model.Project): double
+determineBudgetHealth(p: com.builder.portfolio.model.Project): String
«Util» DBConnectionUtil
+getConnection(): java.sql.Connection
«Constant» StatusConstants
+STATUS_UPCOMING: String
+STATUS_IN_PROGRESS: String
+STATUS_COMPLETED: String
+BUDGET_UNDER: String
+BUDGET_ON_TRACK: String
+BUDGET_OVER: String
«Util» GanttChartUtil
+printSimpleGantt(p: com.builder.portfolio.model.Project): void

3. Sequence Diagram – Add Project

Purpose

Demonstrates the flow of control for adding a new project into the system.

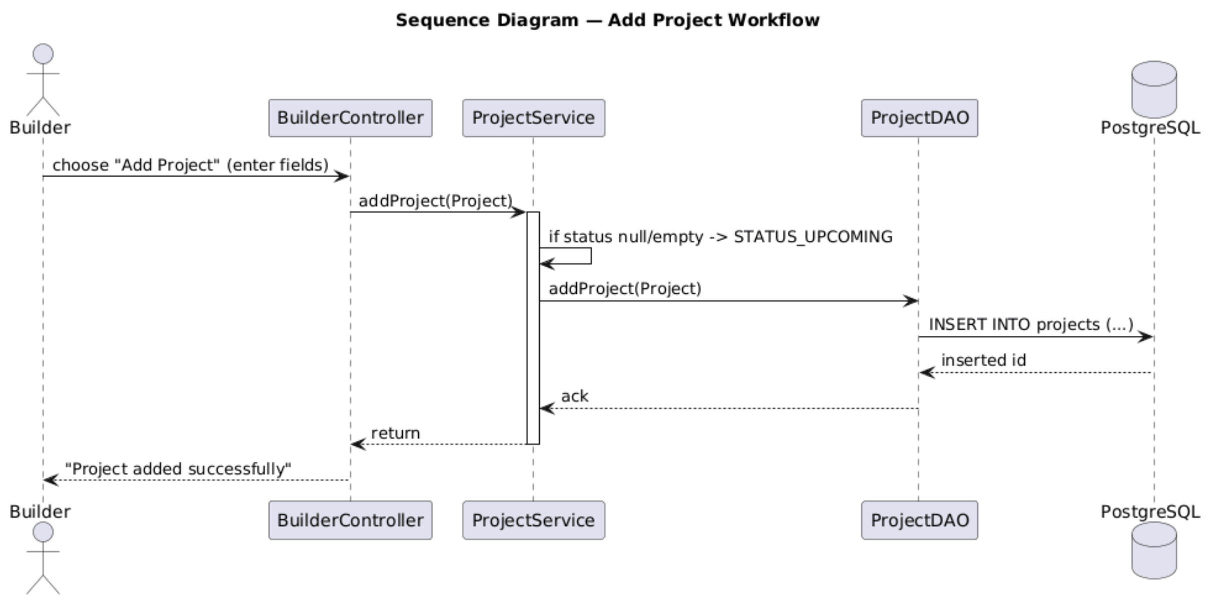
Workflow Summary

1. **User (Builder)** initiates project creation.
2. The request goes to **BuilderController**, which validates inputs.
3. **ProjectService** sets the default status (`UPCOMING`) and performs logic checks.
4. **ProjectDAO** executes the SQL `INSERT` command.
5. A success message is returned to the user.

Lifelines

User → BuilderController → ProjectService → ProjectDAO → Database

Diagram:



4. Sequence Diagram – Update Project Status

Purpose

Illustrates the process of updating a project's status, including triggering related utility functions.

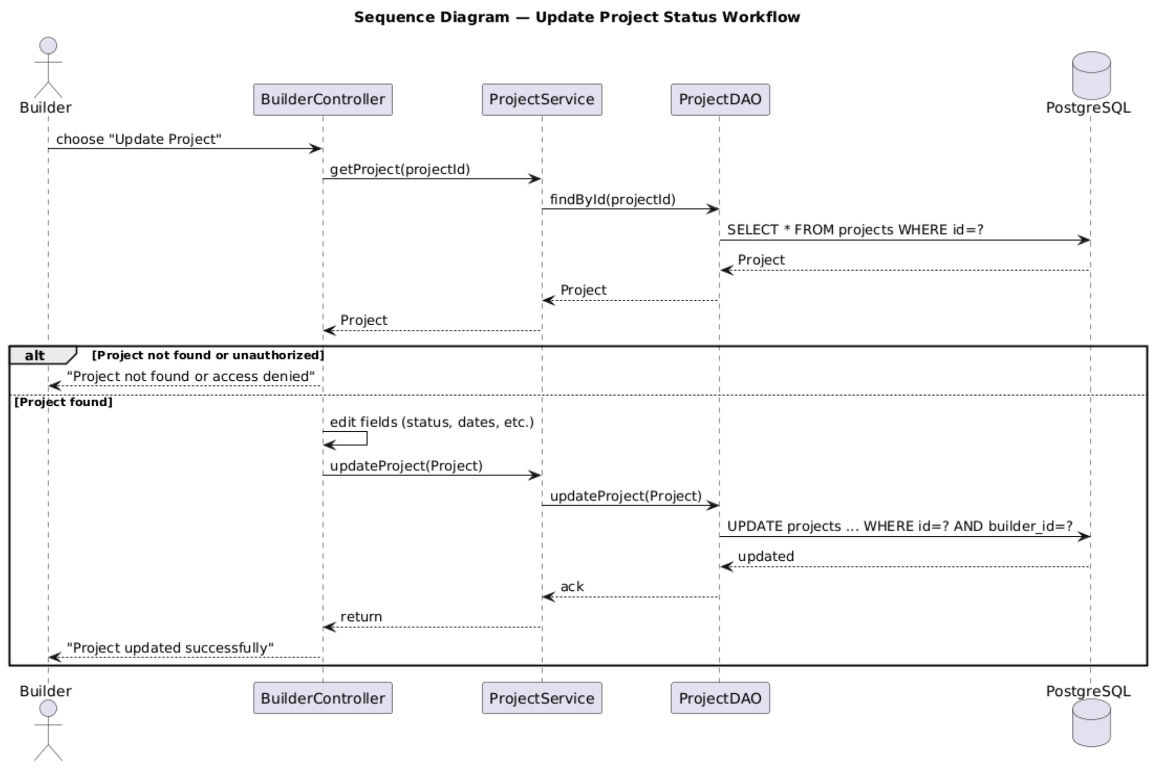
Workflow Summary

1. Builder selects "Update Status."
2. The controller retrieves the project via `getProject()`.
3. Service layer updates the status and logs the change.
4. DAO persists the updated record in the database.
5. If the status changes to `IN_PROGRESS` or `COMPLETED`, the **GanttChartUtil** prints the visual timeline.

Lifelines

User → BuilderController → ProjectService → ProjectDAO → GanttChartUtil → Database

Diagram:


Sequence Diagram — Update Project Status Workflow
Builder
BuilderController
ProjectService
ProjectDAO
PostgreSQL
choose "Update Project"
getProject(projectId)
findById(projectId)
SELECT * FROM projects WHERE id=?
Project
Project
Project
alt
[Project not found or unauthorized]
"Project not found or access denied"
[Project found]
edit fields (status, dates, etc.)
updateProject(Project)
updateProject(Project)
UPDATE projects ... WHERE id=? AND builder_id=?
updated
ack
return
"Project updated successfully"

5. Sequence Diagram – View Portfolio

Purpose

Describes how the system displays the project portfolio for different user roles.

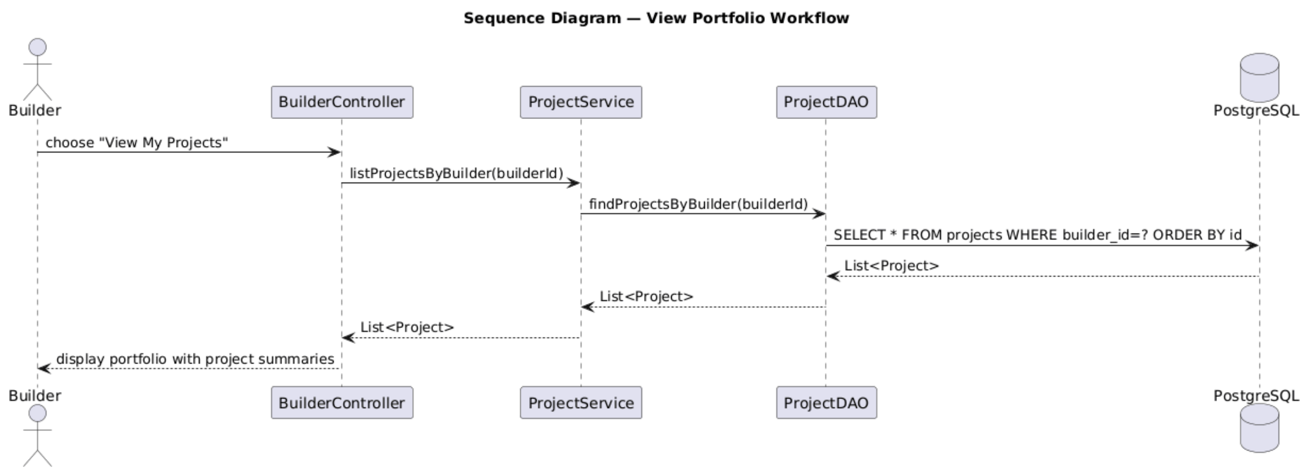
Workflow Summary

1. User (Admin/Builder/Client) requests to view portfolio.
2. The controller invokes appropriate service methods.
3. **Service layer** fetches relevant projects via DAO based on user role.
4. The results are aggregated and returned as a list to the user interface.

Lifelines

User → Controller → Service → DAO → Database

Diagram:


Sequence Diagram — View Portfolio Workflow
Builder
BuilderController
ProjectService
ProjectDAO
PostgreSQL
choose "View My Projects"
listProjectsByBuilder(builderId)
findProjectsByBuilder(builderId)
SELECT * FROM projects WHERE builder_id=? ORDER BY id
List<Project>
List<Project>
List<Project>
display portfolio with project summaries

6. Workflow Diagrams

These workflow diagrams illustrate the dynamic behavior and logical control flow of key user activities in the Builder Portfolio Management System.
Each diagram corresponds to a real-world use case handled through the system's controller and service layers.

6.1 Login / Register Workflow

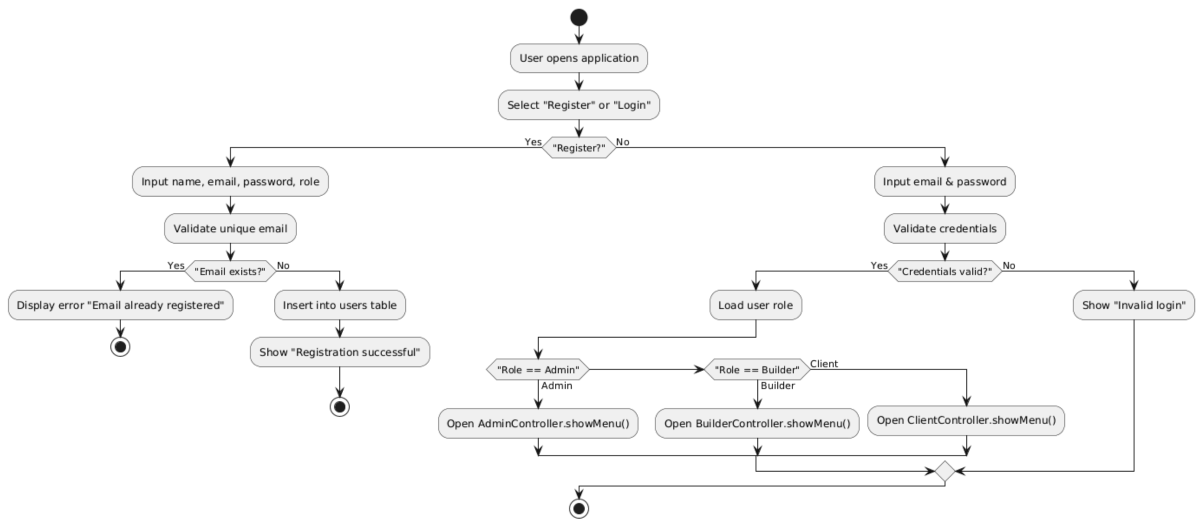
Purpose

The **Login and Registration workflow** manages user authentication, onboarding, and access control for three types of users — **Admin**, **Builder**, and **Client**.
It validates user credentials, ensures unique registration, and directs users to role-specific dashboards upon successful login.

Description

- The user begins by selecting **"Register"** or **"Login."**
- In registration, input data is validated to prevent duplicate emails.
- In login, credentials are verified via the service and DAO layers.
- Successful login routes the user to the corresponding controller menu.

Diagram



6.2 Project Management Workflow

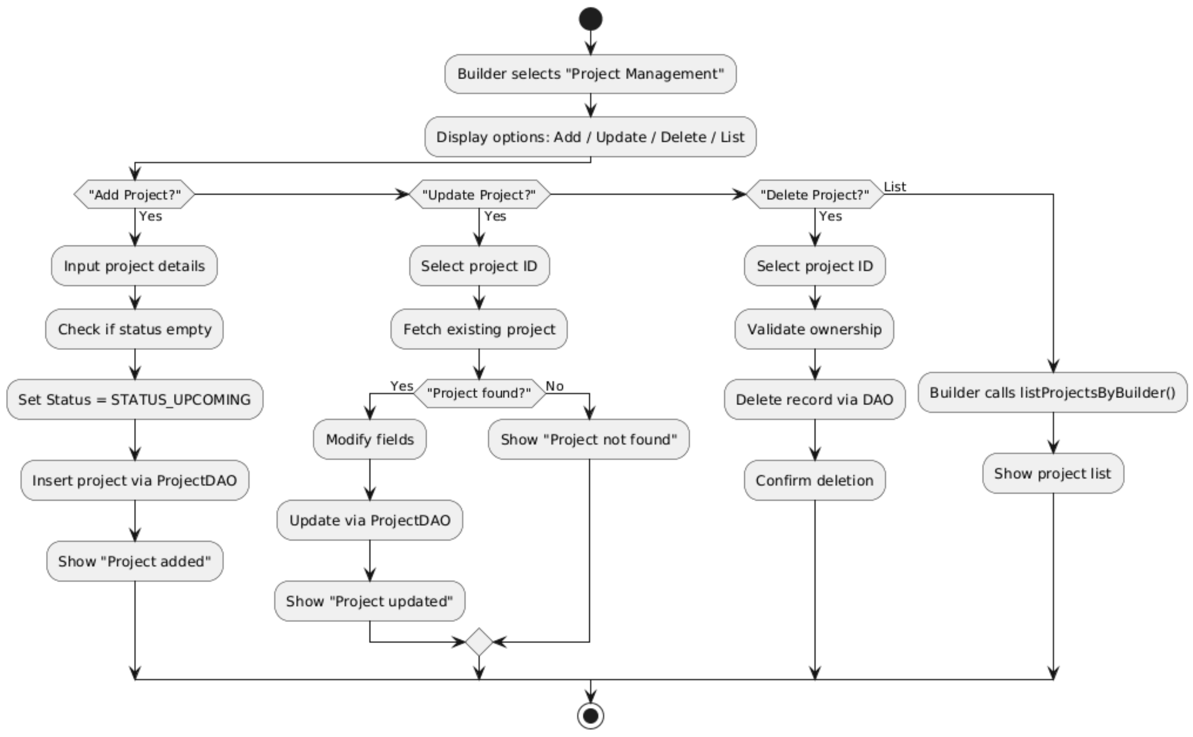
Purpose

This workflow represents the **core operational logic** of managing projects — including creation, updating, deletion, and listing — within the system.

Description

- Builder selects **“Project Management”** from their dashboard.
- System displays options: Add, Update, Delete, or List Projects.
- **Add Project:** Builder inputs details, system assigns default status `UPCOMING`, and saves via DAO.
- **Update Project:** Fetches existing record, modifies fields, updates via DAO.
- **Delete Project:** Validates ownership, deletes record, and confirms removal.
- **List Projects:** Displays all projects associated with the builder.

Diagram



6.3 Budget and Timeline Tracking Workflow

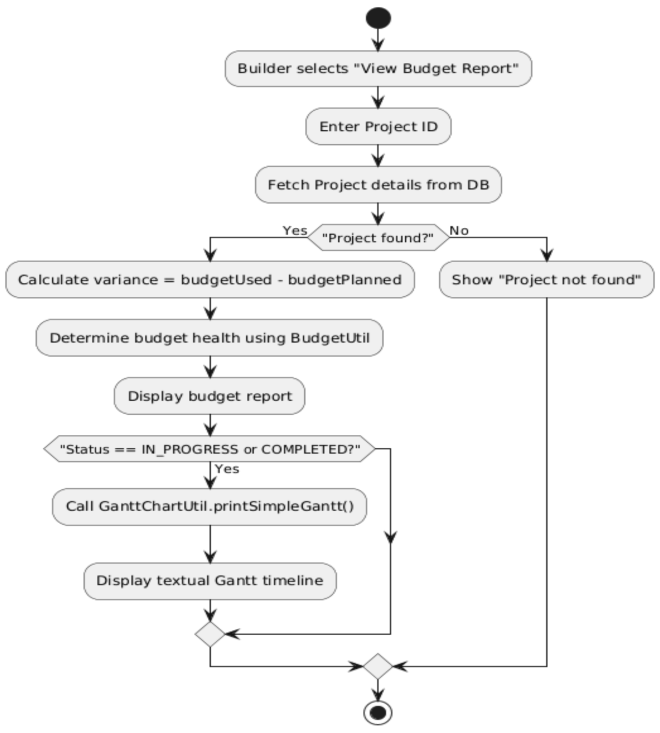
Purpose

This workflow outlines the process of **budget evaluation and progress tracking** for a project, ensuring builders can monitor both financial and temporal health.

Description

- Builder selects **“View Budget Report.”**
- System fetches project data using ProjectService and DAO.
- Variance is calculated (`budgetUsed - budgetPlanned`) using **BudgetUtil**.
- Budget health is classified as *Under*, *On Track*, or *Over*.
- If status is `IN_PROGRESS` or `COMPLETED`, **GanttChartUtil** is triggered to display a simple textual Gantt timeline.

Diagram


Builder selects "View Budget Report"
Enter Project ID
Fetch Project details from DB
Yes
"Project found?"
No
Calculate variance = budgetUsed - budgetPlanned
Show "Project not found"
Determine budget health using BudgetUtil
Display budget report
"Status == IN_PROGRESS or COMPLETED?"
Yes
Call GanttChartUtil.printSimpleGantt()
Display textual Gantt timeline

Conclusion

The UML and workflow diagrams together present a comprehensive view of how the **Builder Portfolio Management System** is architected, developed, and executed in a modular, scalable manner.

- The **ER Diagram** accurately models the relational data structure, ensuring referential integrity and clarity between entities such as Users, Projects, and Documents.

- The **Class Diagram** demonstrates a well-organized, layered architecture (Controller → Service → DAO → Model → Util), emphasizing separation of concerns and maintainability.

- The **Sequence Diagrams** illustrate the dynamic interactions between components, providing insight into key operational flows such as project creation, status updates, and portfolio viewing.

- The **Workflow Diagrams** further visualize how user-driven actions (Login/Register, Project Management, Budget & Timeline Tracking) translate into system-level operations, enhancing traceability and functional understanding.

Together, these diagrams reflect a **production-ready software design** that balances conceptual clarity with practical implementation.

They showcase **enterprise-grade principles**—including modularity, abstraction, reusability, and extensibility—making the Builder Portfolio Management System both technically robust and evolution-friendly for future enhancements.