Step 1 – Initialize Project Skeleton and CI

Goal: Create a Maven Java 21 project skeleton, directories, minimal `pom.xml`, README, and GitHub Actions CI placeholder.

Agent Prompt:

```
Act as a repo-initialization agent. Execute the following:
1. Create directories:
   - src/main/java/com/employeemgmt/
   - src/test/java/com/employeemgmt/
   - src/db/
   - sql-scripts/
   - docs/
2. Create pom.xml with Java 21, MySQL Connector/J, JUnit
Jupiter, Maven Shade, Maven Surefire.
3. Add README.md with overview, quickstart, and build
commands (`mvn clean package`).
4. Add .github/workflows/build.yml with a simple Maven
build job.
5. Commit initial skeleton.

Outputs:
- pom.xml, README.md, build.yml, empty directories.

Acceptance criteria:
- mvn compiles, CI runs without errors, README contains
package instructions.
```

Step 2 – MySQL Schema and Sample Data

Goal: Create idempotent SQL scripts for tables and sample data.

Agent Prompt:

```
Act as a DB-schema agent. Generate SQL under src/db/:
1. 01-schema.sql
   - Tables: employees, division, job_titles, payroll,
employee_division, employee_job_titles
   - PKs, FKs (ON DELETE CASCADE), indexes on Fname, Lname,
SSN, email
   - SSN: VARCHAR(9) UNIQUE NOT NULL
   - created_at, updated_at timestamps
   - CHECK constraints for numeric fields >= 0
2. 02-sample-data.sql
   - 15 employees, 5 divisions, 15 job titles, 27 payroll
rows
   - Use INSERT IGNORE or ON DUPLICATE KEY UPDATE
3. Ensure scripts are safe to re-run.

Outputs:
- src/db/01-schema.sql, src/db/02-sample-data.sql

Acceptance criteria:
- Running scripts creates schema and inserts sample data
without errors, all FKs valid, SSN unique.
```

Step 3 – Database Connection Manager

Goal: Robust DB connection handling using singleton pattern.

Agent Prompt:

```
Act as a DB-connection agent. Implement
DatabaseConnectionManager.java under src/main/java/com/
employeemgmt/db/:
1. Read env vars: DB_HOST, DB_PORT, DB_NAME, DB_USER,
DB_PASS
2. Build JDBC URL with fallbacks
3. Singleton providing Connection objects
4. Connection validation and retry loop
5. Close/cleanup utilities

Outputs:
- DatabaseConnectionManager.java
- docs/JDBC-CONNECTION.md describing env vars and .env
example

Acceptance criteria:
- Can open a connection and query information_schema.tables
```

```
successfully with a small main() test.
```

Step 4 – Domain Models

Goal: Create validated POJOs mapping directly to tables.

Agent Prompt:

```
Act as a domain-model agent. Create model classes under
src/main/java/com/employeemgmt/model/:
- Employee, Division, JobTitle, Payroll, EmployeeDivision,
EmployeeJobTitle

Requirements:
1. Fields map to columns
2. Constructors: default, full, partial
3. Validation: SSN 9 digits, email regex, non-negative
numeric fields
4. toString suitable for CLI display

Outputs:
- Model classes
- docs/MODEL-DESIGN.md listing validations and constructors

Acceptance criteria:
- Validation methods pass unit tests locally
```

Step 5 – DAO Interfaces and Implementations

Goal: Modular DAOs with prepared statements and transaction support.

Agent Prompt:

```
Act as a DAO-implementation agent. Create DAO interfaces
and JDBC implementations under src/main/java/com/
employeemgmt/dao/:
1. EmployeeDAO, PayrollDAO, DivisionDAO, JobTitleDAO,
EmployeeDivisionDAO, EmployeeJobTitleDAO
2. Methods: insert, update, delete, findById, findAll,
specialized searches (findBySSN, searchByName)
```

```
3. EmployeeDAO.updateSalaryByPercentage(double pct,
BigDecimal min, BigDecimal max) transactional
4. Use PreparedStatements, try-with-resources
5. SQL strings centralized

Outputs:
- DAO interfaces and implementations
- sql-scripts/employee-operations.sql

Acceptance criteria:
- Each method executes a single DB operation securely,
sample main can exercise insert/update.
```

Step 6 – Reporting Module

Goal: Implement reporting queries and DTOs.

Agent Prompt:

```
Act as a reporting-agent. Implement ReportingService.java
under src/main/java/com/employeemgmt/reporting/:
1. getEmployeeWithPayrollHistory(empid)
2. getMonthlyPayByJobTitle(year, month)
3. getMonthlyPayByDivision(year, month)
4. Use DAO layer, not raw business logic

Outputs:
- ReportingService.java and DTOs
- sql-scripts/reporting-queries.sql

Acceptance criteria:
- Methods return aggregated results correctly, unit tests
validate sums and grouping.
```

Step 7 – UX Layer (CLI & JavaFX Stub)

Goal: Minimal UX calling DAO methods.

Agent Prompt:

```
Act as a UX-integration agent. Implement CLI and JavaFX
stubs:
1. CLI: src/main/java/com/employeemgmt/cli/MainCli.java
   - Menu: insert, update, delete, search, apply salary
update, employee payroll
2. JavaFX: src/main/java/com/employeemgmt/ui/MainApp.java
   - Screens: search, employee detail, salary update form
3. Both use DatabaseConnectionManager

Outputs:
- MainCli.java, MainApp.java
- docs/RUN-INSTRUCTIONS.md

Acceptance criteria:
- CLI menu actions modify DB and print confirmations;
JavaFX launches and triggers DAO methods for search/update.
```

Step 8 – Automated Tests and Manual Test Cases

Goal: JUnit tests and manual QA documentation.

Agent Prompt:

```
Act as a test-engineer agent. Implement:
1. Automated JUnit tests under src/test/java:
   - EmployeeDAO, PayrollDAO, ReportingService
   - insert, findById, findBySSN, searchByName, update,
delete, salary range update
2. Manual test cases under docs/MANUAL-TEST-SCENARIOS.md
3. Include teardown steps

Outputs:
- Test classes, manual test document

Acceptance criteria:
- mvn test passes with a test DB; manual cases map to DAO
operations.
```

Step 9 – Dockerize Backend and Orchestration

Goal: Dockerfile, docker-compose, health checks, startup sequencing.

Agent Prompt:

```
Act as a dockerization agent. Create:
1. Dockerfile: Java 21 JDK, copy target JAR, entrypoint
java -jar
2. docker-compose.yml with MySQL and app service, env vars,
depends_on, healthchecks
3. .env.example with DB env vars
4. wait-for-db script to poll MySQL readiness

Outputs:
- Dockerfile, docker-compose.yml, .env.example, wait-for-db
script

Acceptance criteria:
- docker compose up initializes DB and runs app; logs
confirm DB connection.
```

Step 10 – Packaging, Docs, Release

Goal: Produce final JAR, release notes, and demo checklist.

Agent Prompt:

```
Act as a release-agent. Do the following:
1. Build shaded JAR with mvn clean package
2. Create code.zip with source, JAR, SQL, docs, README
3. RELEASE_NOTES.md summarizing versions, Docker Compose
steps, validation
4. Demo checklist: search, insert, update, salary update,
report

Outputs:
- target/employee-system.jar, code.zip, RELEASE_NOTES.md,
demo-checklist.txt

Acceptance criteria:
- code.zip contains runnable JAR and SQL; following RUN-
```

```
INSTRUCTIONS.md reproduces environment and demo steps.
```

Step 11 – Integration and Verification

Goal: Full system integration, bug fixing, and validation.

Agent Prompt:

```
Act as an integrator agent. Execute:
1. docker compose up, confirm schema and sample data
2. CLI smoke tests for all menu actions
3. JUnit tests against dockerized DB
4. Capture logs, failures, generate bugfix tasks
5. Iterate until Steps 1–10 acceptance criteria are
satisfied

Outputs:
- integration-report.md with pass/fail, logs, fixes applied

Acceptance criteria:
- All integration tests pass, CLI operations correct, JUnit
green, docker setup stable on L
```