⬡ TECHY PMA

Project Management System

Technical Documentation & Developer Guide

Java 25	**JavaFX 21**	**SQLite**
Language	UI Framework	Database

Author: Sam | Date: February 2026 | Version: 1.0

Built with IntelliJ IDEA | Maven Build System

1. Title & Objective

Getting Started with JavaFX + SQLite in Java – A Beginner's Project Guide

Technology Choices

This project uses the following core technologies:

Technology	Version	Purpose
Java (OpenJDK)	25.0.2	Primary programming language
JavaFX	21.0.6	Desktop UI framework for building screens
SQLite via JDBC	3.45.1.0	Lightweight embedded database
BCrypt (jBCrypt)	0.4	Secure password hashing library
Maven	3.8.x (bundled)	Build automation and dependency management

IntelliJ IDEA	2025.3.2	Integrated development environment

Why These Technologies?

- Java was chosen for its strong object-oriented design, cross-platform compatibility, and large ecosystem of libraries. It is widely used in enterprise applications and provides a solid foundation for learning software architecture patterns.
- JavaFX was selected as the UI framework because it integrates natively with Java, supports FXML-based declarative UI design, and provides rich UI controls including progress bars, combo boxes, and scrollable layouts.
- SQLite was chosen over server-based databases (MySQL, PostgreSQL) because it requires zero configuration, runs as an embedded file-based database, and is ideal for single-user desktop applications.
- BCrypt (jBCrypt) was selected for password hashing because it is an industry-standard, deliberately slow hashing algorithm with built-in salt generation, making it highly resistant to brute-force attacks.
- Maven was chosen as the build tool for its standard project structure, straightforward dependency management via pom.xml, and excellent IntelliJ IDEA integration.

End Goal

The end goal is a fully functional desktop Project Management System that allows multiple users to register, authenticate securely, create and manage projects, join existing projects, track progress, and collaborate through a clean and modern JavaFX interface. The application demonstrates real-world patterns including the DAO (Data Access Object) pattern, MVC (Model-View-Controller) architecture, session management, and layered application design.

2. Quick Summary of Technologies

2.1 Java

Java is a high-level, class-based, object-oriented programming language designed to have as few implementation dependencies as possible. First released in 1995 by Sun Microsystems (now Oracle), it follows the principle of Write Once, Run Anywhere (WORA) through the Java Virtual Machine (JVM).

Java is used across a vast range of domains: web backends (Spring Boot), Android development, enterprise systems, big data tools (Apache Hadoop, Spark), and desktop applications.

Example Netflix uses Java for its backend microservices that serve over 200 million subscribers globally, processing billions of API requests daily.

2.2 JavaFX

JavaFX is a software platform for creating and delivering desktop applications. It replaced the older Swing framework and provides a rich set of UI controls, layout panes, animations, CSS styling, and FXML-based UI design. Since Java 11, JavaFX has been maintained separately as OpenJFX.

JavaFX is used for building cross-platform desktop GUIs such as business dashboards, data visualization tools, point-of-sale systems, and educational software.

Example JetBrains uses JavaFX components in several of its IDE tools. FXML (XML-based UI layout) combined with CSS makes JavaFX apps highly customizable without touching Java code.

2.3 SQLite

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured SQL database engine. Unlike most databases, SQLite is not a client-server database engine; it reads and writes directly to ordinary disk files.

SQLite is used extensively in mobile apps (Android, iOS), embedded systems, browsers (Chrome, Firefox use SQLite internally), and desktop applications where a lightweight database is needed.

Example WhatsApp stores all messages locally using SQLite. Apple iOS uses SQLite in over 1 billion devices for storing system and app data.

2.4 BCrypt (jBCrypt)

BCrypt is a password hashing function designed by Niels Provos and David Mazieres. It incorporates a salt to protect against rainbow table attacks and is intentionally slow (adaptive), meaning it can be made progressively slower as hardware speeds increase, making brute-force attacks impractical.

Example Spring Security, the most widely used Java security framework, uses BCrypt as its default password encoder for enterprise web applications.

3. System Requirements

Operating System	Required Tools
Windows 10 / 11 (64-bit)	Java Development Kit (JDK) 17 or 21+
macOS 12+ (Monterey or later)	IntelliJ IDEA 2024.x or later
Ubuntu 20.04 LTS or later	Maven 3.8+ (bundled with IntelliJ)
Minimum 4 GB RAM (8 GB recommended)	Internet connection (first-time dependency download)
500 MB free disk space	Git (optional, for version control)

3.1 Maven Dependencies (pom.xml)

The following dependencies are declared in the Maven pom.xml file and are automatically downloaded on first build:

```
<!-- JavaFX Controls -->
```

```
<dependency>

    <groupId>org.openjfx</groupId>

    <artifactId>javafx-controls</artifactId>

     <version>21.0.6</version>

</dependency>



<!-- JavaFX FXML -->

<dependency>

    <groupId>org.openjfx</groupId>

    <artifactId>javafx-fxml</artifactId>

     <version>21.0.6</version>

</dependency>



<!-- SQLite JDBC Driver -->
```

```
<dependency>
     <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
     <version>3.45.1.0</version>
</dependency>


<!-- BCrypt Password Hashing -->
<dependency>
    <groupId>org.mindrot</groupId>
    <artifactId>jbcrypt</artifactId>
     <version>0.4</version>
</dependency>
```

4. Installation & Setup Instructions

Step 1: Install JDK 21

Download and install OpenJDK 21 from https://adoptium.net or use IntelliJ IDEA's built-in JDK downloader:

1. Open IntelliJ IDEA
2. Go to File > Project Structure > SDK
3. Click + > Download JDK
4. Select OpenJDK 21 from the vendor list and click Download

Step 2: Create a New JavaFX Project

5. Open IntelliJ IDEA and click New Project
6. Select JavaFX from the left sidebar
7. Configure: Language: Java, Build System: Maven, JDK: 21
8. Set GroupId: org.sam.projectmanager, ArtifactId: techy_pma
9. Skip all Additional Libraries checkboxes (FXML is included by default)
10. Click Create

Note IntelliJ may show a warning about Maven wrapper download failing. This is safe to ignore - IntelliJ will fall back to its bundled Maven version automatically.

Step 3: Configure pom.xml

Open pom.xml and add the SQLite JDBC and jBCrypt dependencies shown in Section 3.1. Then click the Maven reload icon (circular arrows) in the top-right editor toolbar to download all dependencies.

Step 4: Configure module-info.java

Create or update src/main/java/module-info.java with the required module declarations:

```
module org.sam.projectmanager.techy_pma {

     requires javafx.controls;

     requires javafx.fxml;

     requires org.xerial.sqlitejdbc;

     requires org.mindrot.jbcrypt;



     opens org.sam.projectmanager.techy_pma to javafx.fxml;

     opens org.sam.projectmanager.techy_pma.controllers to
javafx.fxml;
```

```
    exports org.sam.projectmanager.techy_pma;

    exports org.sam.projectmanager.techy_pma.controllers;

    exports org.sam.projectmanager.techy_pma.utils;

    exports org.sam.projectmanager.techy_pma.models;

    exports org.sam.projectmanager.techy_pma.database;

}
```

Step 5: Create the Project Folder Structure

Create the following packages under src/main/java/org/sam/projectmanager/techy_pma/:

```
techy_pma/

  ├── controllers/      (LoginController, SignupController,
DashboardController, ...)

  ├── database/         (DatabaseManager, UserDAO,
ProjectDAO, ProjectMemberDAO)

  ├── models/           (User, Project, ProjectMember)
```

```
   └── utils/              (PasswordUtil, Session,
SelectedProject)


src/main/resources/org/sam/projectmanager/techy_pma/

   ├── fxml/               (login.fxml, signup.fxml,
dashboard.fxml, ...)

   └── css/                (styles.css, dashboard.css)


data/                     (projectmanager.db - auto-created at
runtime)
```

Step 6: Initialize the Database

Call DatabaseManager.initializeDatabase() in the start() method of Main.java. This will:

- Create the data/ directory file projectmanager.db automatically
- Create the users, projects, and project_members tables if they do not exist
- Print 'Database initialized successfully!' to the console

Step 7: Run the Application

Use the Maven JavaFX plugin to run the application:

```
# Via IntelliJ Maven panel:

Plugins > javafx > javafx:run



# Or right-click Main.java > Run 'Main.main()'
```

5. Minimal Working Example

5.1 Description

The following minimal example demonstrates the complete user signup-to-login flow: creating a new user with a hashed password, storing them in SQLite, retrieving them, verifying the password, and storing the session. This covers the core authentication pattern used throughout the application.

5.2 Code Example with Comments

```
// Step 1: Initialize the database (create tables if not
exists)

DatabaseManager.initializeDatabase();




// Step 2: SIGNUP - Hash the password before storing

String plainPassword = "mySecurePass123";

String hashedPassword =
PasswordUtil.hashPassword(plainPassword);

// hashedPassword = "$2a$10$YIQTPXIWKM00e7Y06I2QUe..."
(BCrypt hash)
```

```
// Step 3: Create User object and insert into database

User newUser = new User("alice", "alice@email.com",
hashedPassword);

boolean success = UserDAO.insertUser(newUser);

// newUser.getUserId() is now set to the auto-generated ID



// Step 4: LOGIN - Retrieve user by username

User foundUser = UserDAO.getUserByUsername("alice");



// Step 5: Verify the plain password against the stored
hash

boolean isValid =
PasswordUtil.verifyPassword("mySecurePass123",
foundUser.getPassword());

// isValid = true
// Step 6: Store user in session (accessible app-wide)

if (isValid) {

    Session.setCurrentUser(foundUser);

}



// Step 7: Access session anywhere in the app

User current = Session.getCurrentUser();

System.out.println(current.getUsername()); // "alice"

System.out.println(Session.isLoggedIn());   // true

System.out.println(Session.getCurrentUserId()); // 1
```

5.3 Expected Console Output

```
Database initialized successfully!
```

```
✓ User inserted successfully: alice

✓ Session started for user: alice

alice

true

1
```

6. AI Prompt Journal

This section documents the AI-assisted development process, recording key prompts used, the AI responses, and their effectiveness throughout the project build.

Prompt 1: Project Setup & Structure

Prompt Used	*"I want to learn Java and JavaFX. I'm building a Project Management System. I want to understand the project structure, required libraries and tools."*
AI Response Summary	The AI provided a complete project structure breakdown including models, controllers, services, database, and utils packages. It explained Maven vs Gradle differences, recommended Maven for beginners, and provided the complete pom.xml configuration with JavaFX, SQLite JDBC, and jBCrypt dependencies.
Evaluation	Very helpful. Provided a production-ready architecture from the start, preventing common structural mistakes. The Maven vs Gradle comparison helped in making an informed decision.

Prompt 2: Database Setup & SQLite

Prompt Used	*"Recreate the DatabaseManager class with the following: user_id (primary key), username, email, password, created_at for users table. Projects table with project_id, project_name, project_description, project_progress, created_by (foreign key), created_at, status (not started, in progress, completed, published)."*
AI Response Summary	The AI generated a complete DatabaseManager class using try-with-resources, JDBC PreparedStatements, and SQL CHECK constraints for the status field. It also added a UNIQUE(project_id, user_id) constraint to the project_members table and explained each SQL keyword in detail.
Evaluation	Extremely helpful. The debugging version with step-by-step console logging helped identify that tables were not being created due to a silent SQL failure. The debug output resolved the issue within minutes.

Prompt 3: DAO Pattern & UserDAO

Prompt Used	*"[XML prompt defining UserDAO class specification with methods: insertUser, getUserById, getUserByUsername, getAllUsers, updateUser, deleteUser with return types and error handling rules]"*

AI Response Summary	The AI implemented the full UserDAO class following the DAO pattern with a private mapResultSetToUser() helper method to avoid code duplication across query methods. It also added bonus usernameExists() and emailExists() validation methods critical for signup flow.
Evaluation	The XML specification format provided a very precise output. The DRY principle application with the helper method was a valuable addition not explicitly requested.

Prompt 4: Utils Layer (PasswordUtil & Session)

Prompt Used	*"Before we head into code, explain the Utils without jumping into code"*
AI Response Summary	The AI provided a full conceptual breakdown of both utility classes before writing any code. It used analogies (toolbox, name tag), flow diagrams, and real-world signup/login examples. It explained the difference between hashing and encryption, why BCrypt is preferred over MD5/SHA1, and why a static Session class is better than passing the user object between controllers.
Evaluation	Extremely useful. Asking for the explanation before the code meant the implementation made complete sense

	when it arrived. The BCrypt one-way function explanation was particularly clear and is a technique worth repeating for other complex topics.

Prompt 5: Login & Signup Screens

Prompt Used	*"Create a complete Signup screen implementation"*
AI Response Summary	The AI generated the complete signup.fxml, SignupController.java, and updated LoginController to navigate between screens. It included a real-time password strength indicator, full form validation (7 checks: username length, email format, password length, password match, duplicate username/email), BCrypt hashing on submit, and auto-login after successful signup.
Evaluation	The password strength indicator was an unexpected but valuable addition. The validation chain covered edge cases not explicitly requested, resulting in production-quality input handling from day one.

Prompt 6: Type Mismatch Bug Fix

Prompt Used	*"java: incompatible types: boolean cannot be converted to int in SignupController"*
AI Response Summary	The AI immediately identified the root cause — UserDAO.insertUser() returned boolean while the controller expected int. It offered two solutions: change the DAO return type to int, or adapt the controller to use boolean. It recommended keeping the boolean return and showed that the generated user ID was still accessible via newUser.getUserId() because the DAO sets it directly on the object.
Evaluation	The two-option approach was helpful. The explanation of why newUser.getUserId() still works after a boolean return clarified how the DAO pattern passes data back through the object rather than just the return value — a key Java concept.

Prompt 7: Dashboard UI Design

Prompt Used	*"Create the dashboard for the project management app" with a structured list: project title, logged-in user name/email, create/browse/my projects initialisers, and a current projects block with name, owner, status, and description.*

AI Response Summary	The AI produced a full dark-sidebar dashboard design with CSS, stat cards, quick action buttons, dynamic project card generation, empty state handling, role and status badges, and a progress bar per project — all without a third-party UI library. It also generated DashboardController with role-based badge fallback logic.
Evaluation	The structured item list as a prompt produced a much more targeted result than a free-form request would have. The empty state panel and role badge fallback were thoughtful additions not explicitly requested. This prompt format (instruction + structured items list) is the most effective pattern used in this project.

Prompt 8: Console Warnings Explanation

Prompt Used	*"Why are these errors appearing on my console when I run the program" followed by the full console output including native access warnings, SLF4J warning, and Unsafe deprecation warning.*
AI Response Summary	The AI identified all four warning types, explained that none affect functionality, and traced each to its specific cause: Java 25 stricter native access rules for JavaFX and SQLite JDBC, missing SLF4J implementation dependency, and JavaFX's internal Marlin renderer

	using a deprecated internal API. It provided optional fixes for three of the four.
Evaluation	The summary table at the end (Warning / Cause / Action Needed) was the most useful part — it meant warnings could be triaged at a glance rather than investigated one by one. This confirmed that all four were harmless and no code changes were needed.

Prompt 9: Code Commenting & Review

Prompt Used	*"Is the Java controller well commented for ease of use, what essential comments are missing and which have been implemented well"*
AI Response Summary	The AI performed a structured code review scoring each comment category — class Javadoc, field comments, method Javadoc, inline comments, and @param/@return/@throws tags. It produced a summary table with pass/fail per area and a percentage verdict, then offered to produce a fully commented version that added comments without altering a single line of code.
Evaluation	The review-first, rewrite-second approach was the right workflow. Having gaps identified before the rewrite meant the final commented file was genuinely improved rather than just having comments added

	indiscriminately. Applied across four controller files and DatabaseManager.java.

7. Common Issues & Fixes

Issue 1: java.lang.IllegalStateException: Location is not set

ERROR java.lang.IllegalStateException: Location is not set at javafx.fxml.FXMLLoader.loadImpl

Cause

The FXMLLoader could not locate the .fxml file because the resource path was incorrect. The resources folder was nested inside the full package path structure instead of being at the root of the resources folder.

Fix

Use the full classpath-based resource path that mirrors the package structure:

```
// WRONG (file not found):

Main.class.getResource("/fxml/login.fxml")



// CORRECT (mirrors package structure in resources):
```

```
Main.class.getResource("/org/sam/projectmanager/techy_pma/f
xml/login.fxml")
```

Issue 2: Database tables not created (empty .db file)

ERROR Database file created but no tables visible in DB Browser for SQLite.

Cause

The initializeDatabase() method was either not being called or a silent SQLException was occurring without visible output, causing the table creation SQL to fail.

Fix

- Added detailed debug logging (step-by-step print statements) to each table creation step
- Replaced Java 15+ text blocks ("""...""") with concatenated strings for Java version compatibility
- Added a tableExists() verification method after each CREATE TABLE statement
- Deleted the existing empty .db file and reran to ensure fresh initialization

Issue 3: java: module not found: javafx.fxml

ERROR java: module not found: javafx.fxml (compile-time error)

Cause

Java 9+ module system requires explicit module declarations. The module-info.java file was missing the requires javafx.fxml directive and the opens directive for the controllers package.

Fix

```
// Add to module-info.java:

requires javafx.fxml;

opens org.sam.projectmanager.techy_pma.controllers to
javafx.fxml;
```

Issue 4: incompatible types: boolean cannot be converted to int

ERROR java: incompatible types: boolean cannot be converted to int in SignupController.java

Cause

The UserDAO.insertUser() method was implemented to return boolean instead of int, while the SignupController expected an int (the generated user ID).

Fix

Updated SignupController to match the boolean return type:

```
// WRONG (type mismatch):

int userId = UserDAO.insertUser(newUser);

if (userId > 0) { ... }



// CORRECT (matches boolean return type):

boolean success = UserDAO.insertUser(newUser);

if (success) {

    System.out.println("User ID: " + newUser.getUserId());
// ID set by DAO

}
```

Issue 5: Maven wrapper download failed (WARNING)

```
WARN    Cannot download ZIP distribution from
        https://repo.maven.apache.org/maven2/... The Maven
        wrapper was disabled.
```

Cause

Network or firewall restrictions prevented IntelliJ from downloading the project-specific Maven wrapper version specified in .mvn/wrapper/maven-wrapper.properties.

Fix

No action required. IntelliJ automatically falls back to its bundled Maven version, which is fully functional. The warning is informational only and does not affect build or runtime behavior.

Issue 6: SLF4J: Failed to load class StaticLoggerBinder

```
WARN    SLF4J: Failed to load class org.slf4j.impl.StaticLoggerBinder.
        Defaulting to no-operation (NOP) logger.
```

Cause

The SQLite JDBC driver depends on SLF4J for logging but no SLF4J implementation was included in the project dependencies.

Fix

This is a harmless warning. The application functions correctly without an SLF4J implementation. To suppress the warning, add an SLF4J simple logger to pom.xml:

```
<dependency>

     <groupId>org.slf4j</groupId>

    <artifactId>slf4j-simple</artifactId>

     <version>1.7.36</version>

</dependency>
```

8. References

8.1 Official Documentation

Resource	URL	Description
OpenJFX (JavaFX)	openjfx.io	Official JavaFX documentation, API reference, and getting started guides
Java SE 21 Docs	docs.oracle.com/en/java/javase/21	Oracle official Java 21 language and API documentation
SQLite Official	sqlite.org/docs.html	Complete SQLite SQL syntax reference and documentation
Xerial SQLite JDBC	github.com/xerial/sqlite-jdbc	JDBC driver for SQLite in Java with usage examples
jBCrypt Library	mindrot.org/projects/jBCrypt	BCrypt password hashing library documentation

Maven Docs	maven.apache.org/guides	Maven build system guides and pom.xml reference
IntelliJ IDEA Help	jetbrains.com/idea/documentation	IntelliJ IDEA features, shortcuts, and project configuration

8.2 Video Resources

- Bro Code – Java Full Course (YouTube): Comprehensive Java beginner tutorial covering OOP, collections, and file I/O
- Genuine Coder – JavaFX with IntelliJ (YouTube): Step-by-step JavaFX setup, FXML design, and SceneBuilder walkthrough
- Coding with John – Java JDBC Tutorial (YouTube): JDBC connections, PreparedStatements, and ResultSet handling
- Amigoscode – Java for Beginners (YouTube): Modern Java features including records, streams, and lambdas

8.3 Helpful Articles & Blogs

- Baeldung.com/java-bcrypt – Spring BCrypt password encoding guide with code examples
- Baeldung.com/javafx – Complete JavaFX tutorial series covering layouts, controllers, and CSS
- SQLiteTutorial.net – Beginner to advanced SQLite tutorials with CREATE, SELECT, JOIN examples

- StackOverflow.com – Tag: javafx+sqlite for community solutions to common integration issues
- JournalDev.com/javafx – Practical JavaFX application development tutorials

8.4 Tools Used

- IntelliJ IDEA 2025.3.2 – Primary IDE (jetbrains.com/idea)
- DB Browser for SQLite – Visual database inspection tool (sqlitebrowser.org)
- SceneBuilder – Visual FXML layout designer (gluonhq.com/products/scene-builder)
- Git / GitHub – Version control and code backup (github.com)
- Claude AI (Anthropic) – AI pair programming assistant used throughout development (claude.ai)

Techy PMA – Project Management System

Technical Documentation | Version 1.0 | February 2026 | Author: Sam

Built with Java 25 + JavaFX 21 + SQLite + BCrypt + Maven + IntelliJ IDEA