1. Setup Maven Project

Steps to Break Down the Maven Setup Task:

1. **Create the Maven Project:**
    - **Step 1**: Create a new Maven project.
        - If you're using **IntelliJ IDEA**:
            1. Open IntelliJ and choose `File -> New Project`.
            2. Select `Maven` and ensure `Create from archetype` is unchecked.
            3. Set `GroupId` (e.g., `com.library.management`) and `ArtifactId` (e.g., `LibraryManagementSystem`).
2. **Setup Maven POM File (`pom.xml`):**
    - The `pom.xml` file is the key to managing dependencies, plugins, and build configurations.

Step 2: Add essential dependencies for the project in `pom.xml`:

```
<dependencies>
    <!-- JavaFX -->
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>17</version> <!-- Use appropriate version -->
    </dependency>

    <!-- SQLite JDBC -->
    <dependency>
        <groupId>org.xerial</groupId>
        <artifactId>sqlite-jdbc</artifactId>
        <version>3.36.0.3</version> <!-- Use the latest version -->
    </dependency>
```

```
    <!-- JUnit for Testing -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.7.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

-

Step 3: Add the **JavaFX plugin** for Maven to make sure JavaFX works seamlessly in your Maven project.

xml
CopyEdit

```
<build>
    <plugins>
        <plugin>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-maven-plugin</artifactId>
            <version>0.0.5</version>
            <executions>
                <execution>
                    <goals>
                        <goal>run</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

3. **Verify Project Structure:**
    - **Step 4**: Ensure your Maven project follows the standard directory structure:
        - `src/main/java` – For your Java source files.
        - `src/main/resources` – For FXML files, images, or other resources.
        - `src/test/java` – For unit tests (JUnit tests).
    - **Step 5**: Create a basic folder structure in `src/main/java` based on your packages (e.g., `com.library.management`).
4. **Test the Maven Setup (Run a Simple Test):**

Step 6: Create a simple `Main` class in `src/main/java/com/library/management/Main.java`

```
package com.library.management;

public class Main {
    public static void main(String[] args) {
        System.out.println("Maven project is set up successfully!");
    }
}
```

Step 7: In your terminal, navigate to the project folder and run:

```
mvn clean install
mvn javafx:run
```

- **Step 8**: Verify that the message "Maven project is set up successfully!" appears in the console.

5. **Push the Initial Setup to GitHub:**

```
git add .
git commit -m "Initial Maven project setup"
git remote add origin <repository-url>
```

```
git push -u origin main
```

6. **Ensure Everyone Has the Project Setup Locally:**
    - **Step 11**: Ask team members to clone the repository and run `mvn clean install`

2. Database Schema Design

Steps to Break Down the Database Schema Design Task:

1. **Define the Entities:**
    - **Step 1: Identify the main entities for your system. Based on your project, the key entities could be:**
        - **Books: Information about books (e.g., title, author, genre, publication year, ISBN).**
        - **Members: Information about library members (e.g., name, email, membership date, user role).**
        - **Transactions: Data for borrowing and returning books (e.g., borrow date, return date, book ID, member ID).**
        - **Authors: Information about authors (e.g., name, biography, nationality).**
2. **Design the Relationships:**
    - **Step 2: Define how the entities are related to each other. Some key relationships might include:**
        - **Book ↔ Author: A book has an author (one-to-many relationship), so you will need a Book table with a foreign key to the Author table.**
        - **Member ↔ Transaction: A member can borrow multiple books, so you'll have a Member table and a Transaction table with a foreign key to Member.**
        - **Book ↔ Transaction: A transaction will reference the Book (many-to-one relationship).**
3. **Design Tables and Attributes:**
    - **Step 3: Create the table structures with relevant attributes:**
        - **Books table might include:**
            - `book_id` **(Primary Key)**

        - `title`
        - `author_id` (Foreign Key)
        - `genre`
        - `publication_year`
    - Members table:
        - `member_id` (Primary Key)
        - `name`
        - `email`
        - `membership_date`
        - `user_role` (e.g., Admin or Member)
    - Transactions table:
        - `transaction_id` (Primary Key)
        - `member_id` (Foreign Key)
        - `book_id` (Foreign Key)
        - `borrow_date`
        - `return_date`

4. Create the Schema in DB Browser:
    - Step 4: Use DB Browser for SQLite to create the actual schema (tables and relationships).
        - Open DB Browser and create the necessary tables using SQL commands or the GUI.

Example SQL query to create a table for Books:

sql
CopyEdit

```
CREATE TABLE Books (
    book_id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT,
    author_id INTEGER,
    genre TEXT,
    publication_year INTEGER,
    FOREIGN KEY (author_id) REFERENCES Authors (author_id)
);
```

3. UI Mockups and Initial Design

Steps to Break Down the UI Mockups and Initial Design Task:

1. **Identify the Main Screens and Components:**
    - **Step 1: List out the primary screens that the application will have:**
        - **Main screen: Overview of the library, navigation menu, user info.**
        - **Book catalog: Display a list of books available in the library.**
        - **Search bar: Allow users to search for books by title, author, or genre.**
        - **Borrow/Return screen: For borrowing and returning books.**
2. **Create Rough Mockups (UI Sketches):**
    - **Step 2: Use tools like Figma, Sketch, or even paper to sketch out the UI design.**
        - **For Main screen: Include navigation options (Home, Catalog, Borrow/Return, etc.).**
        - **For Book catalog: Create a list/grid layout that displays book titles, authors, and other details.**
        - **For Search bar: Design a search bar that allows text input and possibly filter options.**
        - **Include buttons for Borrow and Return next to each book.**
3. **Use JavaFX and FXML for Layout:**
    - **Step 3: Start working with FXML (XML-based language for JavaFX) to structure the layout:**
        - **For the main screen, create a basic `AnchorPane` or `BorderPane` layout.**
        - **Use a `TableView` or `ListView` for displaying books in the catalog.**

        - **Add a `TextField` for the search bar and a `Button` for search functionality.**

4. **Implement UI Elements in JavaFX:**
    - **Step 4: Implement the core components in JavaFX.**
        - **For example, add a `TextField` for entering search queries, and a `Button` to trigger the search.**
        - **Ensure the layout is responsive and looks good on various screen sizes.**

4. Feature Breakdown and Task Assignment

Steps to Break Down the Feature Breakdown and Task Assignment Task:

1. **Identify Key Features:**
    - **Step 1: Review the main features of the project:**
        - **User Authentication: Admin and member login, role-based access.**
        - **Book Catalog Management: Adding, viewing, and managing books in the library.**
        - **Borrowing & Returning Books: Tracking borrowed books and return dates.**
        - **Reports & Fine Calculation: Generate reports and calculate overdue fines.**
2. **Break Down Features into Smaller Tasks:**
    - **Step 2: Split each feature into smaller tasks. For example:**
        - **User Authentication:**
            - **Create login screen.**
            - **Implement username/password validation.**
            - **Set up role-based access (Admin vs. Member).**
        - **Book Catalog Management:**
            - **Design the catalog UI.**
            - **Implement CRUD operations for books.**
        - **Borrowing & Returning Books:**
            - **Track borrowed books with transaction dates.**
            - **Implement return date logic.**

5. Initial Testing Setup **Steps to Break Down the Initial Testing Setup Task:**

1. **Set Up JUnit Testing Framework:**
    - **Step 1: Ensure that JUnit is added as a dependency in `pom.xml`.**
    - **Step 2: Create a basic test class (`BookTest.java`, `MemberTest.java`, etc.) in `src/test/java` to validate basic functionality.**
2. **Write Basic CRUD Tests:**
    - **Step 3: Create tests for the basic CRUD operations. For example:**
        - **Test if a book can be added to the database.**
        - **Test if a member can be added to the system.**
        - **Test if a book can be borrowed and returned.**
3. **Set Up a Continuous Testing Framework:**
    - **Step 4: Ensure that the testing framework is integrated into your build process so that tests can be run automatically during development.**
4. **Prepare for Ongoing Testing:**
    - **Step 5: Plan for further testing in later stages of the project (e.g., testing user authentication, UI behavior, transaction workflows).**