Music App Project Summary (Part 2)

Date: June 16, 2025

This document continues from the previous summary. It covers all the steps performed after setting up the initial frontend with Tailwind CSS and the login page. Below is a step-by-step explanation of the backend setup and configuration, written in simple and beginner-friendly language.

# Step 1: Project Folder Structure

We confirmed the correct folder structure. The main folder is named 'music app'. Inside it, there is a subfolder also called 'music-app' which contains both the frontend (with node\_modules) and backend (Spring Boot project).

# Step 2: Maven Installation and Configuration

**Step 2: Install Maven – Full Explanation**

**💡 What is Maven?**

* **Maven** is a **build automation tool** used mainly for **Java projects**.
* It helps you:
  + Manage **project dependencies** (like Spring Boot, MySQL connector, etc.)
  + Compile, test, and package Java code
  + Run your Spring Boot application using simple commands like mvn spring-boot:run

In short: **Maven is like an assistant that downloads tools and runs your project for you**.

**🧩 What You Need Before Installing Maven**

* ✅ **JDK 17** must be installed and set up before Maven (you’ve already done this).
* ✅ **Set Java environment variables** (done already if java -version works in terminal).

**🧱 Step-by-Step Guide to Install Maven**

**📥 1. Download Maven**

* Go to the official Maven download page:  
  🔗 <https://maven.apache.org/download.cgi>
* Click to download:

**"Binary zip archive"** for Windows  
File: apache-maven-3.9.10-bin.zip (or latest)

**📦 2. Extract the ZIP File**

* After downloading, extract the file to a simple location (no spaces in the path):

makefile

CopyEdit

C:\apache-maven-3.9.10

* Inside this folder, you should see:

bash

CopyEdit

/bin/

/conf/

/lib/

LICENSE

README.txt

✅ The /bin folder contains important files like mvn.cmd which we use in the terminal.

**🧭 3. Set Environment Variables**

This step lets you run mvn from **any folder** in your terminal (like VS Code).

**🔹 Step-by-Step (Windows):**

1. Press **Windows key** → search for **Environment Variables**
2. Click on **"Edit the system environment variables"**
3. Click on **"Environment Variables…"** (bottom right)
4. In **System variables**, find and select Path → click **Edit**
5. Click **New**, then paste this:

makefile

CopyEdit

C:\apache-maven-3.9.10\bin

1. Click **OK** on all windows

**🔁 4. Restart Your Terminal**

* Close and reopen VS Code or Command Prompt
* This reloads the new Path setting

**✅ 5. Verify Maven Installation**

Open your terminal and type:

bash

CopyEdit

mvn -v

✅ If installed correctly, you’ll see something like:

yaml

CopyEdit

Apache Maven 3.9.10

Maven home: C:\apache-maven-3.9.10

Java version: 17.0.12

OS name: "windows 11"

**🧪 Troubleshooting Tips**

| **Problem** | **Solution** |
| --- | --- |
| 'mvn' is not recognized | You didn't add Maven to the Path variable |
| File missing like mvn.cmd | You downloaded the wrong file (download -bin.zip) |
| Java not found | Make sure JDK 17 is installed and java -version works |

**✅ Final Check**

Once mvn -v works, your Maven is successfully installed!

You can now use commands like:

bash

CopyEdit

mvn spring-boot:run

To run your backend Spring Boot application 🎉

# Step 3: Running the Spring Boot Backend

**✅ Step 3: Running the Spring Boot Backend – Detailed Guide**

**💡 What is Spring Boot?**

* **Spring Boot** is a framework that makes it easy to create **Java-based web backends**.
* It comes with everything you need to:
  + Build a REST API
  + Connect to databases
  + Handle business logic
  + Serve JSON responses to your frontend (like React)

**🎯 What Does "Running the Backend" Mean?**

When we say **“run the backend”**, we mean:

* Start a server (usually on port **8080**)
* Load all the Java code and dependencies
* Get it ready to respond to frontend requests (like login, signup, fetch songs, etc.)

**🧱 Requirements Before Running**

Make sure:

* ✅ Java is installed (java -version)
* ✅ Maven is installed (mvn -v)
* ✅ Spring Boot project is correctly generated
* ✅ You are in the **backend project folder** in terminal (where pom.xml is present)

**📁 Folder Structure**

Your project folder should look like this:

css

CopyEdit

music-app/

└── music-app/

├── backend/

│ ├── src/

│ ├── pom.xml ✅ This file must be here

│ └── ...

└── frontend/

└── ...

**⚙️ What is pom.xml?**

* This is a **special file** Maven uses to know:
  + Which dependencies to download (like Spring Boot, MySQL, etc.)
  + How to build and run the app

**✅ Command to Run the Backend**

First, open terminal in VS Code or PowerShell and **go to the backend folder**:

bash

CopyEdit

cd E:\music-app\music-app\backend

Make sure you are in the same directory as the pom.xml file.

Then run this command:

bash

CopyEdit

mvn spring-boot:run

**🔄 What Happens When You Run This?**

1. Maven downloads all required libraries (only once)
2. It compiles your Java code
3. It starts an **embedded server** (like Tomcat)
4. You’ll see terminal output like:

arduino

CopyEdit

Started BackendApplication in 6.2 seconds (JVM running for 6.8)

✅ This means your backend is now **running on** [**http://localhost:8080**](http://localhost:8080)

**📍 How to Confirm It's Working?**

* Open browser and go to:

arduino

CopyEdit

http://localhost:8080

* If nothing is defined yet, you may see a blank screen or error (that’s okay—we haven't created any endpoints yet)

**✅ How to Stop the Server**

* Go to terminal and press:

mathematica

CopyEdit

Ctrl + C

* This will stop the backend server.

**🧪 Common Errors**

| **Error** | **Fix** |
| --- | --- |
| 'mvn' is not recognized | Maven not added to Path |
| 'spring-boot:run' failed | Check if you're in the folder with pom.xml |
| Java version errors | Use **JDK 17** only |
| Port already in use (8080) | Close previous terminal or change port |

**📌 Extra Tip: You Can Also Build a JAR File**

If you want to create an executable file:

bash

CopyEdit

mvn clean install

This creates a .jar file inside the target/ folder which you can run like:

bash

CopyEdit

java -jar target/backend-0.0.1-SNAPSHOT.jar

(But for now, mvn spring-boot:run is simpler.)

# Step 4: Creating Standard Folder Structure

**✅ Step 4: Creating the Standard Folder Structure (Controller, Model, Repository, Service)**

**💡 Why Create These Folders?**

Spring Boot projects are built using a structure that follows the **MVC pattern**:

| **Layer** | **Purpose** |
| --- | --- |
| controller | Handles requests from the frontend (like login, signup) |
| service | Contains logic (what to do when a user logs in, etc.) |
| repository | Talks to the database (fetch or save data) |
| model | Represents the data (like User, Song, etc.) |

This structure **keeps your code clean, organized, and easy to work with.**

**📁 Where Do We Create These Folders?**

You will create them inside this path:

css

CopyEdit

backend

└── src

└── main

└── java

└── com

└── musicapp

└── backend

├── controller

├── model

├── repository

└── service

**🛠 How to Create the Folders (Manually in VS Code)**

1. Open VS Code
2. Open the **backend/src/main/java/com/musicapp/backend** folder
3. Right-click on backend folder and choose:
   * New Folder → controller
   * New Folder → model
   * New Folder → repository
   * New Folder → service

You will now see:

CopyEdit

📁 backend

├── 📁 controller

├── 📁 model

├── 📁 repository

└── 📁 service

✅ You’ve created the **standard Spring Boot backend structure!**

**🧾 How to Verify You’re in the Right Folder**

Your file path should look like:

swift

CopyEdit

backend/src/main/java/com/musicapp/backend/controller/

If yes, you're good to go!

**🧪 Sample Code Snippets (To Test Setup)**

You can create a basic test class in each folder just to verify structure.

src/main/java/com/musicapp/backend/model

Create a file: User.java and paste this code:

package com.musicapp.backend.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

// Constructors

public User() {}

public User(String username, String password) {

this.username = username;

this.password = password;

}

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

src/main/java/com/musicapp/backend/repository

Create a file named: UserRepository.java

Paste this code inside:

java

Copy

Edit

package com.musicapp.backend.repository;

import com.musicapp.backend.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

// Custom finder method (optional for login functionality)

User findByUsername(String username);

}

ℹ️ This tells Spring to automatically create all the basic CRUD (Create, Read, Update, Delete) database functions for User.

src/main/java/com/musicapp/backend/service

Create a file named: UserService.java

Paste the following code:

java

Copy

Edit

package com.musicapp.backend.service;

import com.musicapp.backend.model.User;

import com.musicapp.backend.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

// Method to register a new user

public User registerUser(User user) {

return userRepository.save(user);

}

// Method to authenticate user during login

public boolean loginUser(String username, String password) {

User user = userRepository.findByUsername(username);

return user != null && user.getPassword().equals(password);

}

// Optional: Get user by ID

public Optional<User> getUserById(Long id) {

return userRepository.findById(id);

}

}

🔐 Note: Passwords should be encrypted in a real app, but for now (just for learning), we're comparing them directly.

src/main/java/com/musicapp/backend/controller

Create a file named: UserController.java

Paste the following code:

java

Copy

Edit

package com.musicapp.backend.controller;

import com.musicapp.backend.model.User;

import com.musicapp.backend.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/api/users")

@CrossOrigin(origins = "\*") // Allow frontend to access backend (important for development)

public class UserController {

@Autowired

private UserService userService;

// Register a new user

@PostMapping("/register")

public User registerUser(@RequestBody User user) {

return userService.registerUser(user);

}

// Login an existing user

@PostMapping("/login")

public boolean loginUser(@RequestBody User user) {

return userService.loginUser(user.getUsername(), user.getPassword());

}

}

✅ That’s it! You’ve now built a basic working backend for login and registration.

Now that the structure is complete, let’s quickly test if everything is working by running the server again:

# Step 5: Project is Up and Running

Spring Boot application is running and the project structure is now set up correctly. We are ready to begin building actual backend functionality such as creating models, controllers, and APIs.

**✅ Step 5: Run the Backend Server**

Go to your backend folder in terminal:

bash

CopyEdit

cd E:\music-app\music-app\backend

Then run:

bash

CopyEdit

mvn spring-boot:run

If everything is correct, you’ll see Spring Boot start and listen on port **8080** by default.

**✅ Step 6: Test API (Optional for Now)**

You can test the APIs using tools like **Postman** or connect them with your frontend later.