A simple API Example to build a **simple RESTful API** in Java that:

1. ✅ Accepts student data via JSON (POST /students)
2. ✅ Saves this data into an **SQLite** database
3. ✅ Returns a list of all students in JSON format (GET /students)

**Maven** is a popular build automation and project management tool primarily used for Java-based projects. It simplifies the build process, manages project dependencies, and helps organize the lifecycle of the project, making it easier to manage, build, test, and deploy applications.

### Maven Coordinates:

1. **Group ID**: A unique identifier for the group or organization that is responsible for the artifact (typically a reversed domain name). For example, org.apache.maven.
2. **Artifact ID**: The name of the artifact (the specific library or project). For example, maven-core.
3. **Version**: The version of the artifact you want to use. For example, 3.8.1.
4. **Packaging**: The type of artifact, such as a .jar, .war, .pom, etc. (this is optional and can be inferred by Maven in most cases).

**Gson** is a popular Java library developed by Google that is used for converting Java objects into JSON (JavaScript Object Notation) format and vice versa. It simplifies the process of serializing Java objects into JSON and deserializing JSON back into Java objects.

 **On the Server**: JSON can be written to a file or database, depending on your needs.

 **On the Client**: JSON can be processed and saved to local storage or as a downloadable file.

 **During Testing**: Tools might write API responses to JSON files for logs or analysis.

Concluded Steps:

 Create a Maven project and add Gson and SQLite JDBC dependencies.

 Create a model class for Student.

 Use HttpServer to handle simple API endpoints.

 Use JDBC to connect and run SQL on the SQLite DB.

 Use Gson to parse JSON in and out.

1. Step one: POM.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>simple-api</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.10.1</version>

</dependency>

<dependency>

<groupId>org.xerial</groupId>

<artifactId>sqlite-jdbc</artifactId>

<version>3.42.0.0</version>

</dependency>

</dependencies>

</project>

Step 2: Student class

public class Student {

public int id;

public String name;

public String major;

public Student() {}

public Student(int id, String name, String major) {

this.id = id;

this.name = name;

this.major = major;

}

}

Step 3 : DBJava  
The **target (goal) of the DBHelper class** is to serve as the **data access layer** of your application — it handles all communication between the Java code and the **SQLite database**

### ****Main Responsibilities of**** DBHelper****:****

1. **Create the database table** (if it doesn't exist).
2. **Insert student records** into the database.
3. **Read all student records** from the database and return them as Java objects (List<Student>)

### Why It Matters:

* It **isolates database logic** from business logic (like in StudentHandler), following **separation of concerns**.
* Makes it easier to **maintain or change the database** in the future.
* Ensures all SQL logic is **centralized** in one place.

Code:

import java.sql.\*;

import java.util.\*;

public class DBHelper {

private static final String DB\_URL = "jdbc:sqlite:students.db";

public static void init() throws SQLException {

try (Connection conn = DriverManager.getConnection(DB\_URL);

Statement stmt = conn.createStatement()) {

stmt.executeUpdate("CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY, name TEXT, major TEXT)");

}

}

public static List<Student> getAllStudents() throws SQLException {

List<Student> list = new ArrayList<>();

try (Connection conn = DriverManager.getConnection(DB\_URL);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM students")) {

while (rs.next()) {

list.add(new Student(rs.getInt("id"), rs.getString("name"), rs.getString("major")));

}

}

return list;

}

public static void insertStudent(Student s) throws SQLException {

String sql = "INSERT INTO students(name, major) VALUES(?,?)";

try (Connection conn = DriverManager.getConnection(DB\_URL);

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, s.name);

pstmt.setString(2, s.major);

pstmt.executeUpdate();

}

}

}

Step 4 : Student Handler   
It defines how the API should **respond to client requests** for the /students endpoint.

**What It Does:**

1. **Handles GET /students:**
   * Retrieves all student records from the database.
   * Converts them to JSON using Gson.
   * Sends them back to the client.
2. **Handles POST /students:**
   * Reads a student’s data from the JSON request body.
   * Converts it into a Java Student object.
   * Saves it into the SQLite database using DBHelper

**Why It's Important:**

It **connects the client-side and server-side logic**:

* Accepts data from the frontend (or API clients)
* Passes it to the database
* Returns results in a format the frontend understands (JSON)

import com.sun.net.httpserver.\*;

import com.google.gson.\*;

import java.io.\*;

import java.nio.charset.StandardCharsets;

import java.sql.SQLException;

import java.util.List;

public class StudentHandler implements HttpHandler {

private Gson gson = new Gson();

@Override

public void handle(HttpExchange exchange) throws IOException {

try {

if ("GET".equals(exchange.getRequestMethod())) {

List<Student> students = DBHelper.getAllStudents();

String response = gson.toJson(students);

sendResponse(exchange, response, 200);

} else if ("POST".equals(exchange.getRequestMethod())) {

InputStreamReader isr = new InputStreamReader(exchange.getRequestBody(), StandardCharsets.UTF\_8);

Student student = gson.fromJson(isr, Student.class);

DBHelper.insertStudent(student);

sendResponse(exchange, "{\"status\":\"inserted\"}", 200);

} else {

sendResponse(exchange, "Unsupported method", 405);

}

} catch (SQLException e) {

sendResponse(exchange, "DB Error: " + e.getMessage(), 500);

}

}

private void sendResponse(HttpExchange exchange, String response, int code) throws IOException {

exchange.sendResponseHeaders(code, response.getBytes().length);

OutputStream os = exchange.getResponseBody();

os.write(response.getBytes());

os.close();

}

}

Step 5 : Main   
import com.sun.net.httpserver.HttpServer;

import java.net.InetSocketAddress;

public class Main {

public static void main(String[] args) throws Exception {

DBHelper.init();

HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);

server.createContext("/students", new StudentHandler());

server.setExecutor(null);

server.start();

System.out.println("Server started on http://localhost:8080/students");

}

}

| **Class** | **File Path (inside src/main/java)** | **Purpose** |
| --- | --- | --- |
| Main | com/example/api/Main.java | Starts the HTTP server |
| Student | com/example/api/Student.java | Holds student data |
| StudentHandler | com/example/api/StudentHandler.java | Handles API logic (GET, POST) |
| DBHelper | com/example/api/DBHelper.java | Manages SQLite database actions |