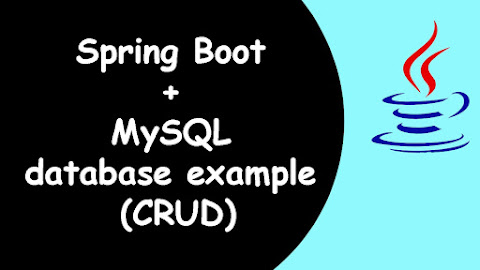
Spring Boot has taken the Spring framework to the next level by drastically reducing the configuration and setup time required for spring projects. You can create a project with practically no settings and begin developing the features that are most important to your application.

In this spring boot tutorial, you will learn how to develop RESTful web services API for CRUD operations on an MYSQL database. The CRUD operations include creation, retrieve, update and delete operations.

[](https://blogger.googleusercontent.com/img/a/AVvXsEiCEUKZxTHjlroBd0xMMgedbbelbSclTLX5-XQ0P4ElKmP4BIllQs0Qed2a6aUZ5v5gf7akEpZi95Ccsctvii_hm6M4WD_GjIRPa2mkgb0aLMgsVMEyMtOzotAebCv2jRIcLZMc8__N0T7z8EmlxOL-egk2kY-8dKndOBj5Jmsgfp53vpyEw_PoI1Ki=s1280)

**1. Create the Project**

There are many ways to create a project in spring. One of the methods is using Spring Initializer to bootstrap an application quickly. You need to go to http://start.spring.io/ and follow the steps below to generate a new project.

Enter the following details to create our example project.

        Group : com.example

        Artifact: studentproject

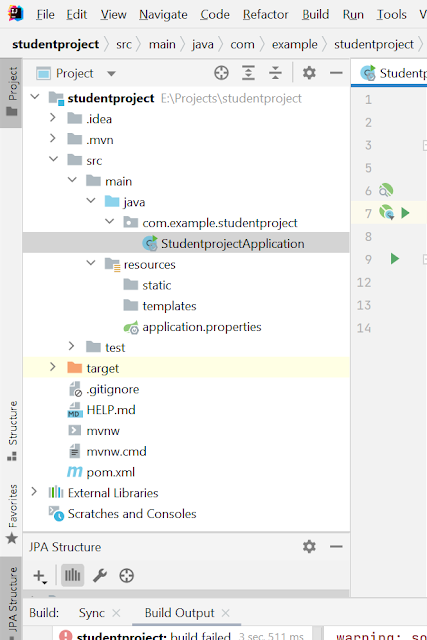
        Pacakge : com.example.studentproject

        Dependecies : [Web](https://javarevisited.blogspot.com/2020/08/top-5-courses-to-learn-spring-mvc-for.html), [JPA](https://www.java67.com/2021/11/best-hibernate-and-spring-data-jpa-courses.html), [MySQL](https://javarevisited.blogspot.com/2018/05/top-5-mysql-courses-to-learn-online.html), DevTools

The dependencies will be added to our pom.xml file and if you want, you can add more dependencies or reduce them manually later on.

|  |
| --- |
| [how to create project with Spring Boot + MySQL database](https://blogger.googleusercontent.com/img/a/AVvXsEjOWKTY1WsHmRUooRgy1SycR9Op9rBGW0MPGSKaadg-Gj5UHa96tLkh5YzlS7L9vIg7b8xCZj8jEufaRH3SCXdhyGvmMKgFvJb7eZ0yLo929BVpY0DgCIQiwBRaLRzx2HuPw5LT2gdtWu1HHW6fqiFa306jDVSmmq_rGehZ3Lk0meUi0aZUI9BbXfPv=s1722) |
|  |

Once all the details are entered, click to GENERATE to create and download your customized initial project. Spring Initializer will generate the project with the details you have entered and download a zip file with all the project folders.  
  
  
  
The below picture is showing the structure of the project.

[](https://blogger.googleusercontent.com/img/a/AVvXsEikHETjE9Ana-QgcgSsscUBYwED1EdBM_0AnkiZMSd3yLdvOh7_DUQMy4a8sMi9R0wvKop-pvYnTSNUltjaGtP5I9pwg2fdtTTtySZLFAcXoBy03H2WudaOWHk23JD0ZgFZViarrm5PARpm3YZFUYRJac_LZThAohLJKLakPiG-sHKeAnKXR-gFUCTu=s924)

Let's have a look into important directories inside the project.

**2. The main entry point of the application.**

The StudentprojectApplication Class is the main entry point of our spring boot application.

package com.example.studentproject;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class StudentprojectApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(StudentprojectApplication.class, args);  
 }  
  
}

This contains the [@SpringBootApplication annotation](https://javarevisited.blogspot.com/2018/05/the-springbootapplication-annotation-example-java-spring-boot.html) which is a combination of the following specific annotations.

@Configuration - Spring bootstraps any class annotated with the @Configuration annotation and uses it as a source for other bean definitions.

@ComponentScan - It instructs Spring to look for and bootstrap any other components in the current package.

[@EnableAutoConfiguration](https://www.java67.com/2018/05/difference-between-springbootapplication-vs-EnableAutoConfiguration-annotations-Spring-Boot.html) - This annotation instructs Spring to configure your application automatically depending on the dependencies you specified in the pom.xml file.

**3. Resources folder**

This directory is dedicated to all the static resources, templates, and property files.

This folder may have the following subfolders depending on your project.

  1. resources/ static

  2. resources/templates

 3. resources/application.properties - This is used to store the application-wide properties of your application and helps wot read those properties to configure your application. This file can have the server's default port, server's context path, database URL, etc.

**4. Configure MYSQL database**

As we mentioned above, the application.properties file is responsible for keeping the configurations of our application. In there, we specify the port number and also the database connection information.

server.port=**8080**

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=**jdbc:mysql:**//localhost/starbucks?createDatabaseIfNotExist=true

spring.datasource.username=root

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect

Here, you need to update the URL username and password according to your MySQL database server.

**5. Code Domain Model Class**

Next, create the Student class to map with the product table in the database as follows.

package com.student.crudapp.model;

import com.student.crudapp.repository.StudentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
  
import javax.persistence.\*;  
  
@Entity  
public class Student {  
  
 @Id  
 @GeneratedValue(strategy=GenerationType.*AUTO*)  
 private int id;  
  
 private String name;  
  
 private String email;  
  
 private String grade;  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getGrade() {  
 return grade;  
 }  
  
 public void setGrade(String grade) {  
 this.grade = grade;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", email='" + email + '\'' +  
 ", grade='" + grade + '\'' +  
 '}';  
 }  
}

This is a simple domain model class, with class names and field names are identical to table names and column names in the database. This allows you to have a minimum number of [JPA annotations](https://javarevisited.blogspot.com/2021/10/what-is-spring-data-repository.html).

**6. Code Repository Class**

package com.student.crudapp.repository;

import com.student.crudapp.model.Student;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface StudentRepository extends JpaRepository<Student, Integer> {  
  
 List<Student> findAll();  
 Student findById(int id);  
 int deleteById(int id);  
  
  
}

The [spring data JPA](https://medium.com/javarevisited/5-best-spring-data-jpa-courses-for-java-developers-45e6438be3c9) will generate implementation code for the most common CRUD operations and you do not need to stick with customized queries. This is one advantage that you will get from using the spring data JPA.

**7, Code Service Class**

This service class acts as a middle later between the persistence layer and the controller layer. Create the StudentService class like below.

package service;  
  
import com.student.crudapp.model.Student;  
import com.student.crudapp.repository.StudentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import javax.transaction.Transactional;  
import java.util.ArrayList;  
import java.util.List;  
  
@Service  
@Transactional  
public class StudentService {  
  
 @Autowired  
 StudentRepository studentRepository;  
  
 *//Get all the students* public List<Student> getAllStudents() {  
 List<Student> students = studentRepository.findAll();  
 return students;  
 }  
  
 *//display one student by id* public Student getStudentById(int id) {  
 return studentRepository.findById(id);  
 }  
  
 *//save student in database* public void saveStudent(Student student) {  
 try{  
 studentRepository.save(student);  
 }  
 catch(Exception e){  
 e.printStackTrace();  
 }  
 }  
  
 *//delete stuednt by id* public void deleteStudent(int id) {  
 try{  
 studentRepository.deleteById(id);  
 }catch(Exception e){  
 e.printStackTrace();  
 }  
 }  
  
}

All the methods are executed in transactions because this studentService class is marked with the [@Transactional annotation](https://javarevisited.blogspot.com/2021/08/spring-transactional-example-how-to.html).

**8. Code REST Controller Class**

This is the class which is dealing with RESTful APIs for CRUD operations. Below is the code:

package com.student.crudapp.controller;  
  
import com.student.crudapp.model.Student;  
import com.student.crudapp.repository.StudentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@Controller  
public class StudentController {  
  
 @Autowired  
 StudentRepository studentRepository;  
  
 *//check the api's working correctly api* @RequestMapping(value="/ping", method=RequestMethod.*GET*)  
 @ResponseBody  
 public String healthCheck() {  
 return "This is working well";  
 }  
  
  
 @RequestMapping(value="/students", method=RequestMethod.*GET*)  
 @ResponseBody  
 public List<Student> getAllStudents() {  
 return studentRepository.findAll();  
 }  
  
 @RequestMapping(value="/student", method=RequestMethod.*POST*)  
 @ResponseBody  
 public Student addStudent(Student student) {  
 return studentRepository.save(student);  
 }  
  
 @RequestMapping(value="/findstudent", method = RequestMethod.*GET*)  
 @ResponseBody  
 public Student findStudent(@RequestParam("studentId") int studentId) {  
 return studentRepository.findById(studentId);  
 }  
  
 @RequestMapping(value= "/updatestudent", method = RequestMethod.*GET*)  
 @ResponseBody  
 public Student updateStudent(@RequestBody Student student){  
 return studentRepository.save(student);  
 }  
  
 @RequestMapping(value="/deletestudent", method = RequestMethod.*GET*)  
 @ResponseBody  
 public int deleteStudent(@RequestParam("studentId") int studentId) {  
 return studentRepository.deleteById(studentId);  
 }  
}

Here, the [@Controller annotation](https://javarevisited.blogspot.com/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html) is used to expose the RESTful APIs. The rest controller still takes advantage of the spring's dependency injection.

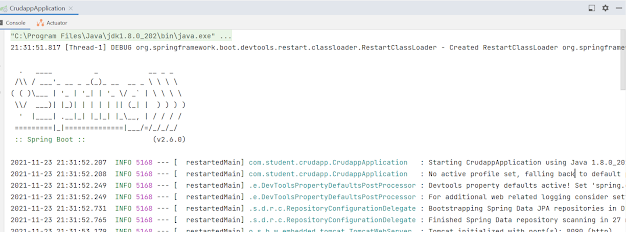
**9. Code Spring Boot Application Class**

To run our application, we need to have the main class as below. This is an inbuilt class and you just need to run this class to run the entire project.

package com.student.crudapp;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class CrudappApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(CrudappApplication.class, args);  
 }  
  
}

 This class will start embedded Tomcat server hosting our Spring Boot web application.

The project will run on port 8080 and url http://localhost:8080/

[](https://blogger.googleusercontent.com/img/a/AVvXsEhHMPJcqKebPfPI6MKdKNmDWVeTFgmL2dJLLbB7wtuinpjDyjr77jyENxkrzT8ZpaKp80aFWz5sXe3DM4-vMq3P-0tIiUIKM5wn9D2OX_FsGVCYsDTzsHLCZjuM6Ny7sz74SXAb-v3kBkfEGeeWvaKDudEnjn8y6ApDKko7_VDhLpCwWXqMIJW7RXYY=s1797)

**10. Test the API and Project**

So let's have a look at APIs in this project. We used the [Postman](https://javarevisited.blogspot.com/2020/02/top-5-postman-tutorials-and-courses-for-web-developers.html)to test the APIs created.

1. Add a student (POST Request)

http://localhost:8080/student

{ "name": "test", "email": "test@gmail.com", "grade": "05" }

2. Get all students (GET Request)

http://localhost:8080/students

3. Find a student (GET Request)

http://localhost:8080/findstudent?studentId=1

4. Update a student (GET Request)

http://localhost:8090/updatestudent

{

    "name": "testupdated",

    "email": "testupdated@gmail.com",

    "grade": "05"

}

5. Delete a student (GET Request)

http://localhost:8090/deletestudent?studentId=1

That's all about **how to use MySQL database with Spring Boot in Java application**. This tutorial referred to the creation of a basic crud functionality using the spring boot, MySql, Jpa and hibernate. As you develop more, there are a lot of things to learn in the spring boot. So We successfully built a Restful CRUD API using Spring Boot, Mysql, JPA, and Hibernate. Hope to see you in the next tutorial. Until then bye!

Top of Form

**Search This Blog**

Read more: <https://javarevisited.blogspot.com/2022/03/spring-boot-mysql-database-example-crud.html#ixzz7OU4IEQur>