Angular + Spring Boot + MySql CURD Application

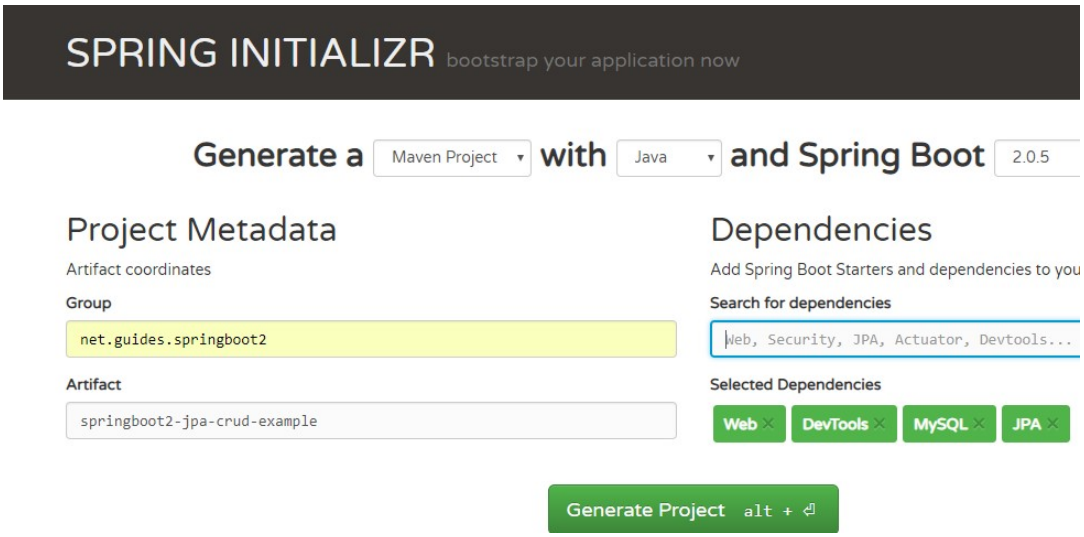
First Part 1: Spring Boot CRUD Rest APIs Development

Following are five REST APIs (Controller handler methods), we will develop for `Employee` resource.

Sr. No.	API Name	HTTP Method	Path	Status Code
(1)	GET Employees	GET	`/api/v1/employees`	200 (OK)
(2)	POST Employee	POST	`/api/v1/employees`	201 (Created)
(3)	GET Employee	GET	`/api/v1/employees/{id}`	200 (OK)
(4)	PUT Employee	PUT	`/api/v1/employees/{id}`	200 (OK)

1. Creating and Importing a Project

There are many ways to create a Spring Boot application. The simplest way is to use Spring Initializrat **http://start.spring.io/**, which is an online Spring Boot application generator.



Look at the above diagram, we have specified the following details:

- **Generate**: Maven Project
- **Java Version**: 1.8 (Default)
- **Spring Boot**:2.0.4

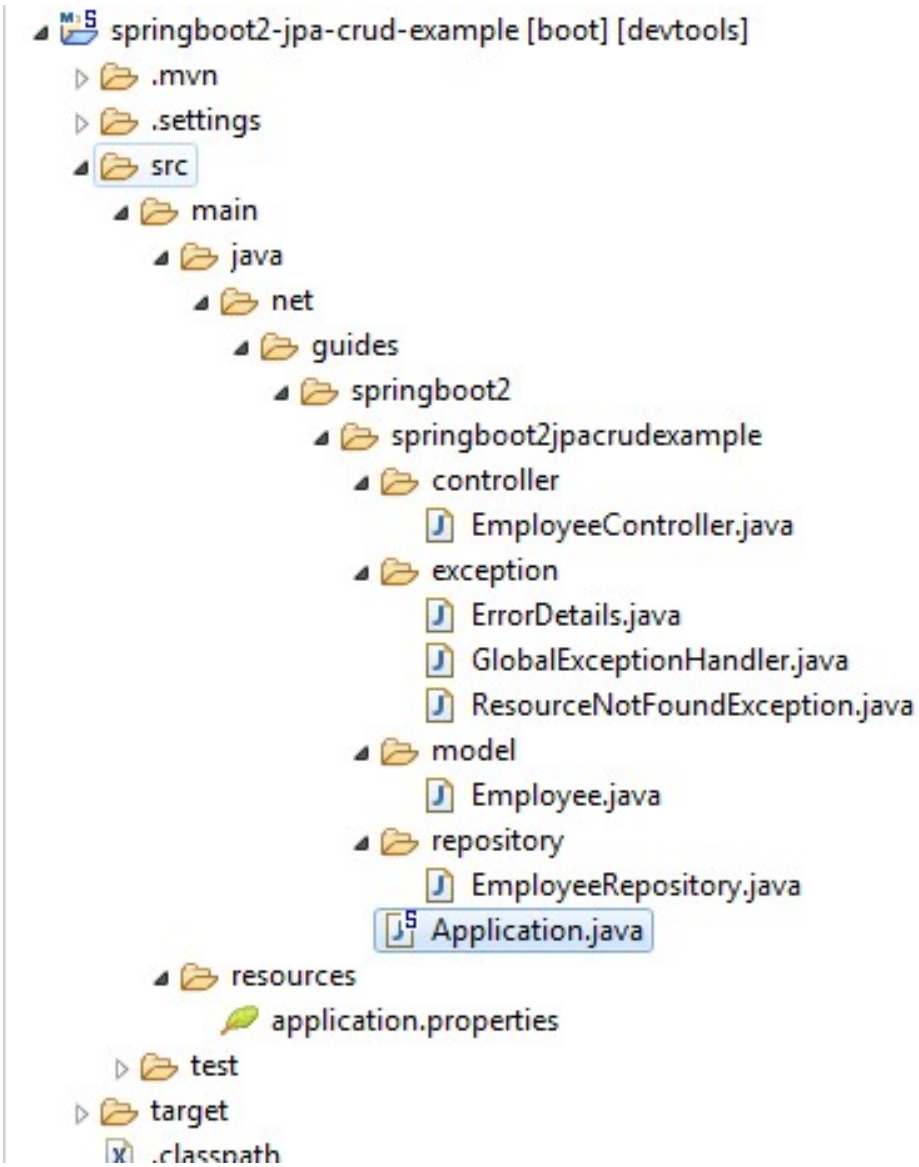
Angular + Spring Boot + MySql CURD Application

- **Group**: net.guides.springboot2
- **Artifact**: springboot2-jpa-crud-example
- **Name**: springboot2-jpa-crud-example
- **Description**: Rest API for a Simple Employee Management Application
- **Package Name** : net.guides.springboot2.springboot2jpacrudexample
- **Packaging**: jar (This is the default value)
- **Dependencies**: Web, JPA, MySQL, DevTools

Once, all the details are entered, click on Generate Project button will generate a spring boot project and downloads it. Next, Unzip the downloaded zip file and import it into your favorite IDE.

2. Packaging Structure

Following is the packing structure of our `Employee Management System` -



Angular + Spring Boot + MySql CURD Application

3. The pom.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<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>net.guides.springboot2</groupId>
    <artifactId>springboot2-jpa-crud-example</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>springboot2-jpa-crud-example</name>
    <description>Demo project for Spring Boot</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
       </dependency>
       <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
       </dependency>

       <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
       </dependency>
       <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
       </dependency>
       <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
      </dependency>
    </dependencies>

    <build>
        <plugins>
             <plugin>
                 <groupId>org.springframework.boot</groupId>
                 <artifactId>spring-boot-maven-plugin</artifactId>
             </plugin>
        </plugins>
     </build>
</project>
```

4. Configuring MySQL Database

Configure `application.properties` to connect to your `MySQL` database. Let's open `an application.properties` file and add the following database configuration to it. Also, note that we have added MySQL dependency to pom.xml so spring boot will auto-configure all database related beans and configurations internally.

Angular + Spring Boot + MySql CURD Application

```
spring.datasource.url = jdbc:mysql://localhost:3306/users_database?useSSL=false
spring.datasource.username = root
spring.datasource.password = root


## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

Add context path to above application properties using below property:

```
server.servlet.context-path=/springboot-crud-rest
```

Change the above configuration such as JDBC URL, username and password as per your environment.

5. Create JPA Entity - Employee.java

```
package net.guides.springboot2.springboot2jpacrudexample.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "employees")
public class Employee {

    private long id;
    private String firstName;
    private String lastName;
    private String emailId;

    public Employee() {

    }

    public Employee(String firstName, String lastName, String emailId) {
         this.firstName = firstName;
         this.lastName = lastName;
         this.emailId = emailId;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
        public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }

    @Column(name = "first_name", nullable = false)
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    @Column(name = "last name", nullable = false)
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

```
    @Column(name = "email address", nullable = false)
    public String getEmailId() {
        return emailId;
    }
    public void setEmailId(String emailId) {
        this.emailId = emailId;
    }

    @Override
    public String toString() {
        return "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", emailId=" +
emailId
        + "]";
    }

}
```

6. Create a Spring Data Repository - EmployeeRepository.java

```
package net.guides.springboot2.springboot2jpacrudexample.repository;

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

import net.guides.springboot2.springboot2jpacrudexample.model.Employee;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}
```

7. Create Spring Rest Controller - EmployeeController.java

```
package net.guides.springboot2.springboot2jpacrudexample.controller;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import net.guides.springboot2.springboot2jpacrudexample.exception.ResourceNotFoundException;
import net.guides.springboot2.springboot2jpacrudexample.model.Employee;
import net.guides.springboot2.springboot2jpacrudexample.repository.EmployeeRepository;

@RestController @CrossOrigin(origins = "http://localhost:4200")
@RequestMapping("/api/v1")
public class EmployeeController {
    @Autowired
    private EmployeeRepository employeeRepository;
```

Angular + Spring Boot + MySql CURD Application

```
    @GetMapping("/employees")
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    @GetMapping("/employees/{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable(value = "id") Long employeeId)
        throws ResourceNotFoundException {
        Employee employee = employeeRepository.findById (employeeId)
          .orElseThrow(() -> new ResourceNotFoundException ("Employee not found for this id :: " +
employeeId));
        return ResponseEntity.ok().body(employee);
    }

    @PostMapping("/employees")
    public Employee createEmployee(@Valid @RequestBody Employee employee) {
        return employeeRepository.save(employee);
    }

    @PutMapping("/employees/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable(value = "id") Long employeeId,
         @Valid @RequestBody Employee employeeDetails) throws ResourceNotFoundException {
        Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + employeeId));

        employee.setEmailId(employeeDetails.getEmailId());
        employee.setLastName(employeeDetails.getLastName());
        employee.setFirstName(employeeDetails.getFirstName());
        final Employee updatedEmployee = employeeRepository.save(employee);
        return ResponseEntity.ok(updatedEmployee);
    }

    @DeleteMapping("/employees/{id}")
    public Map<String, Boolean> deleteEmployee(@PathVariable(value = "id") Long employeeId)
         throws ResourceNotFoundException {
        Employee employee = employeeRepository.findById(employeeId)
       .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + employeeId));

        employeeRepository.delete(employee);
        Map<String, Boolean> response = new HashMap<>();
        response.put("deleted", Boolean.TRUE);
        return response;
    }
}
```

Enable CORS on the Server

To enable CORS on the server, add a @CrossOrigin annotation to the EmployeeController

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("/api/v1")
public class EmployeeController {
 // ....
}
```

8. Exception(Error) Handling for Restful Services

Spring Boot provides a good default implementation for exception handling for RESTful Services. Let's quickly look at the default Exception Handling features provided by Spring Boot.

Resource Not Present

Heres what happens when you fire a request to not resource found: http://localhost:8080/some-dummy-url

Angular + Spring Boot + MySql CURD Application

```
{
  "timestamp": 1512713804164,
  "status": 404,
  "error": "Not Found",
  "message": "No message available",
  "path": "/some-dummy-url"
}
```

That's a cool error response. It contains all the details that are typically needed.

What happens when we throw an Exception?

Let's see what Spring Boot does when an exception is thrown from a `Resource`. we can specify the Response Status for a specific exception along with the definition of the Exception with `'@ResponseStatus'` annotation.

Lets create a `ResourceNotFoundException.java` class.

```
package com.companyname.springbootcrudrest.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends Exception{

    private static final long serialVersionUID = 1L;

    public ResourceNotFoundException(String message){
        super(message);
    }
}
```

Customizing Error Response Structure

The default error response provided by Spring Boot contains all the details that are typically needed.

However, you might want to create a framework independent response structure for your organization. In that case, you can define a specific error response structure.

Let's define a simple error response bean.

```
package com.companyname.springbootcrudrest.exception;

import java.util.Date;

public class ErrorDetails {
    private Date timestamp;
    private String message;
    private String details;

    public ErrorDetails(Date timestamp, String message, String details) {
        super();
        this.timestamp = timestamp;
        this.message = message;
        this.details = details;
    }

    public Date getTimestamp() {
        return timestamp;
    }

    public String getMessage() {
         return message;
    }

    public String getDetails() {
         return details;
```

Angular + Spring Boot + MySql CURD Application

```
    }
}
```

To use `ErrorDetails` to return the error response, let's create a `GlobalExceptionHandler` class annotated with `@ControllerAdvice` annotation. This class handles exception specific and global exceptions in a single place.

```
package com.companyname.springbootcrudrest.exception;

import java.util.Date;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<?> resourceNotFoundException(ResourceNotFoundException ex, WebRequest request) {
         ErrorDetails errorDetails = new ErrorDetails(new Date(), ex.getMessage(), request.getDescription(false));
         return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<?> globleExcpetionHandler(Exception ex, WebRequest request) {
        ErrorDetails errorDetails = new ErrorDetails(new Date(), ex.getMessage(), request.getDescription(false));
        return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

9. Running Application

This spring boot application has an entry point Java class called `Application.java` with the `public static void main(String[] args) method`, which you can run to start the application.

```
import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

The main() method uses Spring Boot's SpringApplication.run() method to launch an application.

Or you can start spring boot application via command line using **mvn spring-boot:run** command.