RESTful Web Services

- REST Introduction
- RESTful Web Services demo
  - Create Maven project
  - Configure Data Source Properties
  - Code Domain Model Class
  - Code Repository Interface
  - Code Service Class
  - Code REST Controller Class: RESTful CRUD API

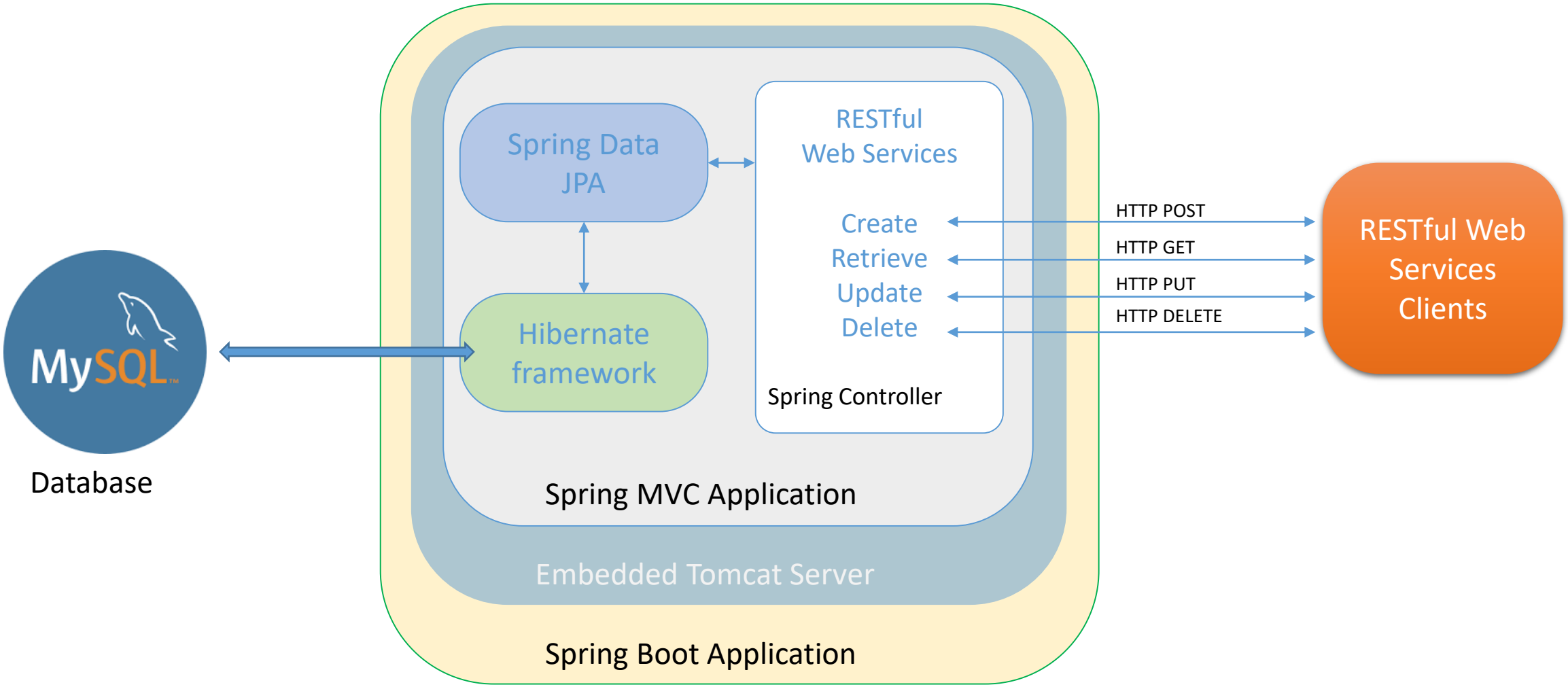
REST Introduction

What is REST?

- The acronym **REST** stands for **RE**presentational **S**tate **T**ransfer. It was term originally coined by Roy Fielding, who was also the inventor of the HTTP protocol. The striking feature of REST services is that they want to make the best use of HTTP.
- **HTTP** provides the base layer for building web services. Therefore, it is important to understand HTTP. Here are a few key abstractions:
  - **Resource:** A resource is a key abstraction that HTTP centers round. A resource is anything you want to expose to the outside world through your application. For instance, if we write a todo management application, instances of resources are:
    - A specific user
    - A specific todo
    - A list of todos
  - **Resource URIs**: When you develop RESTful services, you need to focus your thinking on the resources in the application. The way we identify a resource to expose, is to assign a **URI** — **Uniform Resource Identifier** — to it. For example:
    - The URI for the user Allen is /user/allen
    - The URI for all the todos belonging to Allen is /user/allen/todos
    - The URI for the first todo that Allen has is /user/allen/todos/1
  - **Resource Representation**: REST does not worry about how you represent your resource. It could be XML, HTML, JSON, or something entirely different! The only important thing is you clearly define your resource and perform whatever actions that are supported on it by making use of features already provided by HTTP. Examples are:
    - Create a user: POST /users
    - Delete a user: DELETE /users/1
    - Get all users: GET /users
    - Get a single user: GET /users/1

RESTful Web Services CRUD API architecure

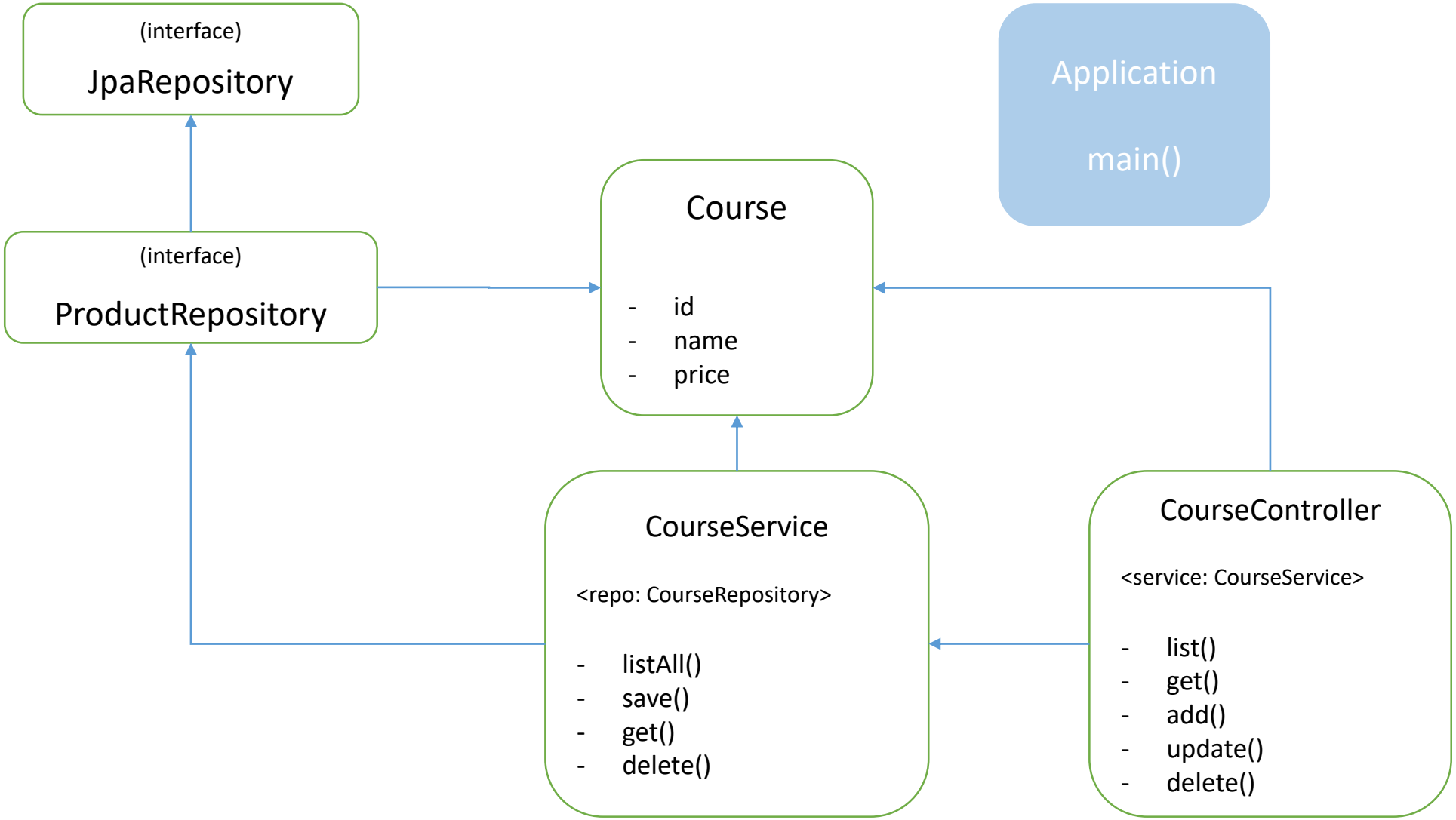
- Here is an example of implementing RESTful Web Services with Spring Boot and MySQL:



RESTful Web Services demo

Demo Project Structure

- This Demo Project structure will be looking like this:



Prepare the database

- To save time, please feel free to use any table you have in MySQL to follow along. If you want to use the same course table as we do. The sql code is provided as following, just copy and run to have a simple course table to work with.

```
create database training;

use training;

CREATE TABLE course (
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL unique,
    price FLOAT NOT NULL,
    PRIMARY KEY (id)
);

INSERT INTO course(name, price) VALUES
   ('Python', 99.99),('Java', 129.99),('C++', 149.99),
   ('C#', 99.99),('Machine Learning', 249.99),('Full Stack', 259.99);

SELECT * FROM course;
```

```
16 •  SELECT * FROM course;
17
```

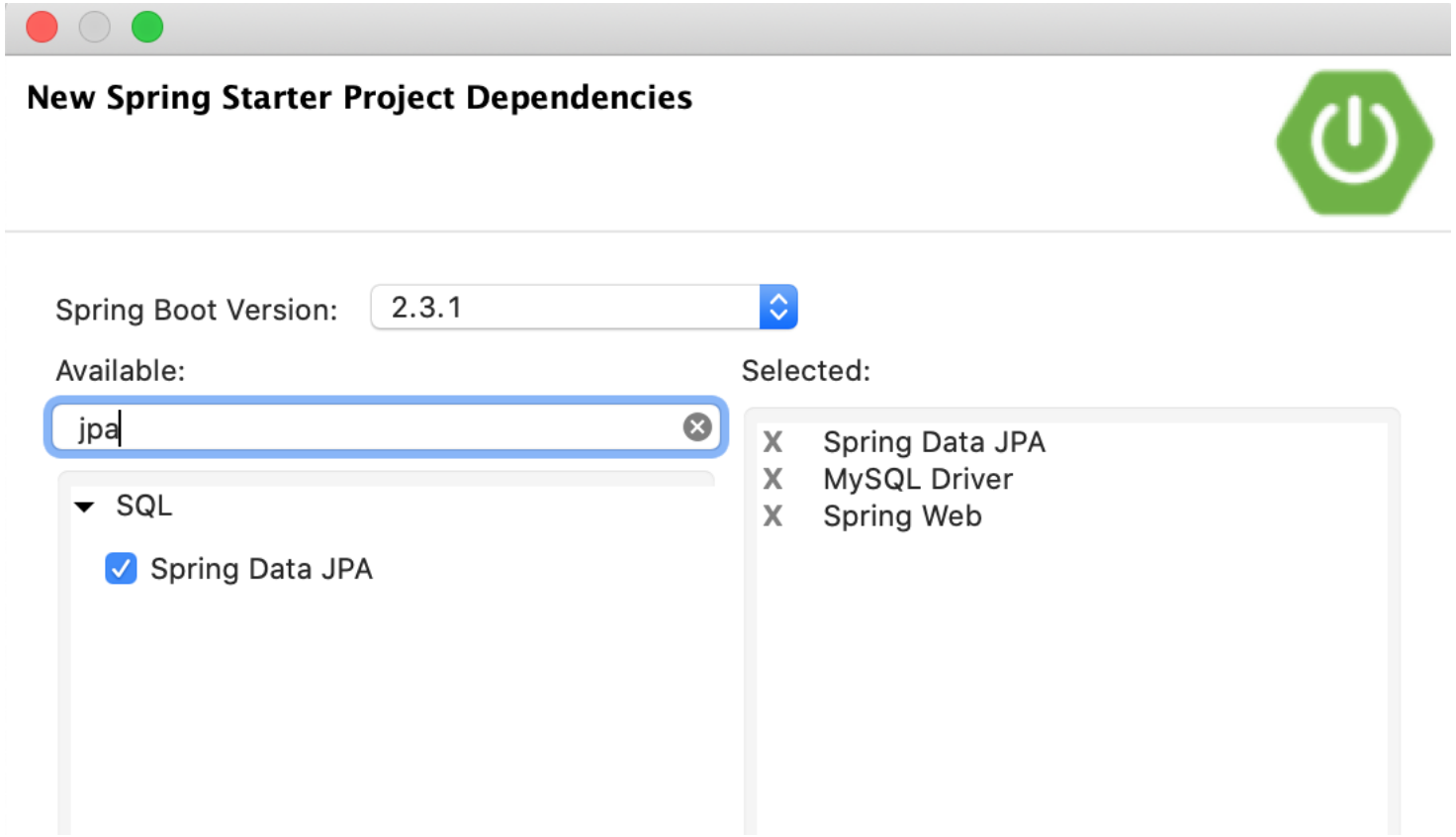
100% 22:9

Result Grid Filter Rows:

id	name	price
1	Python	99.99
2	Java	129.99
3	C++	149.99
4	C#	99.99
5	Machine Learning	249.99
6	Full Stack	259.99
NULL	NULL	NULL

Create Maven Project

- Use Maven to create a spring boot project. Add JPA, MySQL Driver and Web dependencies.



Configure Data Source Properties

- Use Maven to create a spring boot project. Add JPA, MySQL Driver and Web dependencies.

application.properties

```
spring.jpa.hibernate.ddl-auto = none
spring.datasource.url =
jdbc:mysql://localhost:3306/Training?useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.username = root
spring.datasource.password = password
```

add the highlight part if you are getting exceptions regarding Timezone

Code Domain Model Class

```
@Entity
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", unique = true, nullable = false)
     private int id;
     @Column(name = "NAME", unique = true, nullable = false, length = 100)
     private String name;
     @Column(name = "PRICE", unique = false, nullable = false, length = 100)
     private float price;

     public Course() {
     super();
     }

     public Course(int id, String name, float price) {
     super();
     this.id = id;
     this.name = name;
     this.price = price;
     }

     ... Getters and Setters...
}
```

Code Repository Interface

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

public interface CourseRepository extends JpaRepository<Course, Integer> {
}
```

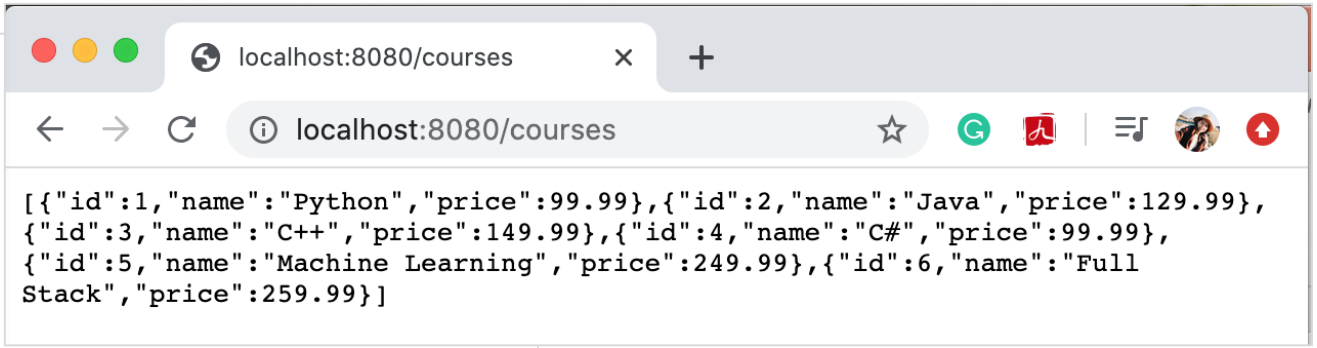
Recap:
JpaRepository is JPA specific extension of Repository. It contains the full API of CrudRepository and PagingAndSortingRepository. So it contains API for basic CRUD operations and also API for pagination and sorting.

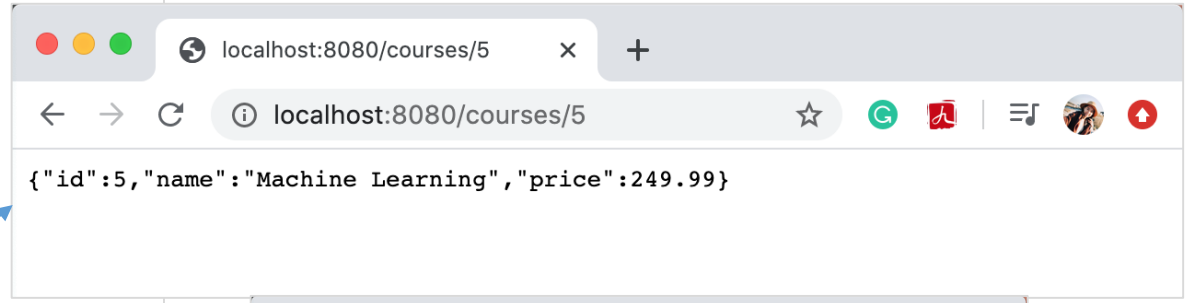
Code Service Class

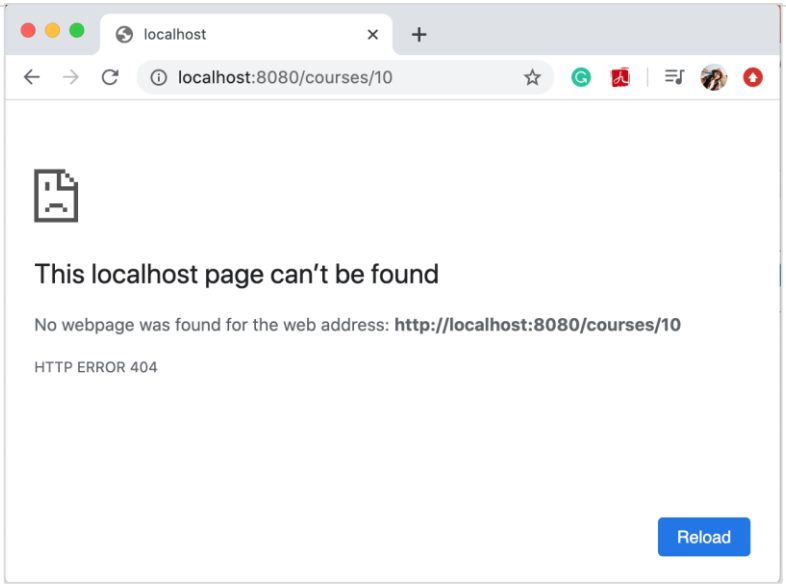
```
@Service
public class CourseService {

	@Autowired
	private CourseRepository repo;

	public List<Course> listAll() {
		return repo.findAll();
	}
	public void save(Course course) {
		repo.save(course);
	}
	public Course get(int id) {
		return repo.findById(id).get();
	}
	public void delete(Integer id) {
		repo.deleteById(id);
	}
}
```

Code REST Controller Class: RESTful CRUD API

```
@RestController
public class CourseController {
        @Autowired
        private CourseService service;

        @GetMapping("/courses")
        public List<Course> list() {
                return service.listAll();
        }

        @GetMapping("/courses/{id}")
        public ResponseEntity<Course> get(@PathVariable int id) {
                try {
                        Course c = service.get(id);
                        return new ResponseEntity<Course>(c, HttpStatus.OK);
                } catch (NoSuchElementException e) {
                        return new
                ResponseEntity<Course>(HttpStatus.NOT_FOUND);
                }
        }
}
```







Code REST Controller Class: RESTful CRUD API

```
@RestController
public class CourseController {
        @Autowired
        private CourseService service;

        @GetMapping("/courses")
        public List<Course> list() {...}

        @GetMapping("/courses/{id}")
        public ResponseEntity<Course> get(@PathVariable int id) {...}

        @PostMapping("/add")
        public void add(@RequestBody Course course) {
                service.save(course);
        }

        @PutMapping("/update/{id}")
        public String update(@RequestBody Course course, @PathVariable int id) {
                try {
                        // if the course already exists, update it
                        Course c = service.get(id);
                        service.save(course);
                        return "course updated";
                } catch (NoSuchElementException e) {
                        return "Not a valid course id";
                }
        }
}
```