```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
      <property name="url" value="jdbc:mysql://localhost:3306/trgdb" />
      <property name="username" value="root" />
      <property name="password" value="root123" />
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
      <property name="dataSource">
            <ref bean="dataSource" />
      </property>
</bean>

<bean id="jtemplate"
     class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
      <constructor-arg ref="dataSource" />
</bean>
```

For logging

```
#application.properties
#Set root logging level
logging.level.root=INFO

#Set logging level for specific packages/classes
logging.level.org.springframework=INFO
logging.level.com.trg.beans=DEBUG

#Set log messages of a particular pattern on a console
logging.pattern.console=%c-[%level]- %d-%m-%n

#To get Log messages in a file
logging.appender.file.append=true
logging.file.name=Applog.log
logging.pattern.file=%c-[%level]- %d-%m-%n
```

For MVC Spring Boot

```
<dependency>
   <groupId>org.apache.tomcat.embed</groupId>
   <artifactId>tomcat-embed-jasper</artifactId>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>jakarta.servlet.jsp.jstl</groupId>
   <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
   <scope>provided</scope>
</dependency>
```

<dependency>

  <groupId>org.glassfish.web**</groupId>**

  <artifactId>jakarta.servlet.jsp.jstl**</artifactId>**

</dependency>

AOP:

```
public interface BusinessServiceIntf {
   public void doBusiness();
}




@Component
public class BusinessService implements BusinessServiceIntf {

   @Override
   public void doBusiness() {
      System.out.println("I do what I do best, i.e sleep.");
      try {
         Thread.sleep(2000);
      } catch (InterruptedException e) {
         System.out.println("How dare you to wake me up?");
      }
      System.out.println("Done with sleeping.");


   }

}





@Configuration
@Aspect
public class BusinessProfiler {
   @Pointcut("execution(* com.trg.*.*(..))")
   public void businessMethods() {
   }
```

```
    @Before("businessMethods()")
    public void MyBeforeMethod() {
        System.out.println("Applying @Before advice");
    }

    @After("businessMethods()")
    public void MyAfterMethod() {
        System.out.println("Applying @After advice");
    }

}
```

```
package com.trg;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Configuration;

import java.util.Date;

@Aspect
@Configuration
public class LoggingInterceptor {
    Logger myLog;

    @Pointcut("execution(* com.trg.*.*(..))")
    public void businessMethods1() {
    }

    @Around("businessMethods1()")
    public Object logs(ProceedingJoinPoint call) throws Throwable {
        Object point = null;
        myLog = LoggerFactory.getLogger(LoggingInterceptor.class);
        try {
            myLog.info("from logging aspect: entering method " + call.getSignature().getName());
            myLog.info("Hello : It is " + new java.util.Date().toString());
            point = call.proceed();
            myLog.info("from logging aspect: exiting method ");
        } catch (Exception e) {
            myLog.warn("I am logging the exception with date " + e + new Date());
        }
        return point;
    }
}
```

```
@SpringBootApplication
@EnableAspectJAutoProxy
public class SpringBoot9AopApplication {

    public static void main(String[] args) {
       ApplicationContext ctx =SpringApplication.run(SpringBoot9AopApplication.class, args);
       BusinessServiceIntf bs = ctx.getBean(BusinessService.class);
       bs.doBusiness();
    }
}
```

```
package com.trg.course.service;

import com.trg.course.entity.Course;
import com.trg.course.exception.CourseAlreadyExistsException;
import com.trg.course.exception.CourseNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

@Service
public class CourseService {

    public CourseService() {
        courses.add(new Course(101, "Spring", "Spring quickstart"));
        courses.add(new Course(102, "Java", "Java fundamentals"));
        courses.add(new Course(103, "NodeJS", "Node essentials"));
    }

    List<Course> courses = new ArrayList<>();

    public List<Course> getCourses() {
        return courses;
    }

    public Course getCourseById(int id) throws CourseNotFoundException {
        Course found = null;
        boolean flag = false;
        for (Course c : courses) {
            if (c.getId() == id) {
                found = c;
                flag = true;
                break;
            }
        }
        if (flag)
            return found;
        else throw new CourseNotFoundException("Course","id", (long) id);
    }


    /*public Course getCourseById(int id) {
      for(Course c : courses){
        if(c.getId()==id){
          return c;
        }
      }
      return null;
    }*/


    public Course addCourse(Course course) throws CourseAlreadyExistsException {
        for (Course c : courses) {
            if (c.getId() == course.getId())
                throw new CourseAlreadyExistsException("Course with id " + course.getId() + " already
exists");
        }
        courses.add(course);
        return course;
    }

    public void updateCourse(int id, Course course) {
        System.out.println(id);
        for (int i = 0; i < courses.size(); i++) {
            if (courses.get(i).getId() == id) {
                courses.set(i, course);
                break;
            }
        }
    }

    public void deleteCourse(int id) throws CourseNotFoundException {
        Iterator<Course> it = courses.iterator();
        boolean flag=false;
        while (it.hasNext()) {
            if (it.next().getId() == id) {
                flag=true;
                it.remove();
                break;
            }
            if(!flag)
            throw new CourseNotFoundException("Course","id", (long) id);
        }
    }

}
```

```
package com.trg.course.controller;

import com.trg.course.entity.Course;
import com.trg.course.exception.CourseAlreadyExistsException;
import com.trg.course.exception.CourseNotFoundException;
import com.trg.course.service.CourseService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/course")
public class CourseController {
    @Autowired
    CourseService courseService;

    /*@ExceptionHandler(value = CourseNotFoundException.class)
    public ResponseEntity handleMyException(CourseNotFoundException ce) {
        return new ResponseEntity("Course Not found", HttpStatus.CONFLICT);
    }*/

    //@CrossOrigin(origins="http://localhost:4200/")
    @GetMapping("/courses")
    public List<Course> getCourses() {
        return courseService.getCourses();
    }


    @GetMapping("/courses/{id}")
    public ResponseEntity getById(@PathVariable int id)  {
        try{
            return new ResponseEntity(courseService.getCourseById(id), HttpStatus.OK);
        }
        catch(CourseNotFoundException e){
            return new ResponseEntity(e.getMessage(), HttpStatus.CONFLICT);
        }
    }


    //@CrossOrigin(origins="http://localhost:4200/")
    /*@GetMapping("/courses/{id}")
    public Course getById(@PathVariable int id) throws CourseNotFoundException {
      System.out.println("In getById() ctrlr");
      return courseService.getCourseById(id);
    }*/

    @DeleteMapping("/courses/{id}")
    public void delCourse(@PathVariable int id) throws CourseNotFoundException {
        courseService.deleteCourse(id);
    }

    @PostMapping("/courses")
    public Course addCourse(@Valid @RequestBody Course course) throws
CourseAlreadyExistsException {
        return courseService.addCourse(course);
    }


    //@CrossOrigin(origins="http://localhost:4200/")
    /*@PostMapping("/courses")
    public ResponseEntity<Object> addCourse(@Valid @RequestBody Course course) {
        try {
            return new ResponseEntity(courseService.addCourse(course), HttpStatus.OK);
        }
        catch(CourseAlreadyExistsException e){
            return new ResponseEntity<>(e.getMessage(), HttpStatus.CONFLICT);
            }
    }*/

    @PutMapping("/courses/{id}")
    public void updateCourse(@PathVariable int id,
                         @RequestBody Course course){
        courseService.updateCourse(id, course);
    }

    /*@DeleteMapping("/courses/{id}")
    public void delCourse(@PathVariable int id) throws CourseNotFoundException {
        courseService.deleteCourse(id);
    }*/
```

```
   /*@GetMapping(value = "/{id}", produces = "application/json")
   public Course getCourseById(@PathVariable int cid) {
     return courseService.getCourseById(cid);
   }*/
}
```

```
package com.trg.course.entity;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;

public class Course {

   @NotNull(message = "Id is required.")
   int id;

   @Size(min = 3, max = 20, message = "The length of name must be between 3 and 20 characters.")
   @NotBlank
   String name;

   @Size(min = 5, max = 50, message = "The length of description must be between 5 and 50
characters.")
   @NotBlank
   String desc;

   // appropriate cons,getter,setter

   public Course() {
   }

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

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }
```

```java
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    @Override
    public String toString() {
        return "Course [id=" + id + ", name=" + name + ", desc=" + desc + "]";
    }

}
```

```java
package com.trg.course.exception;

public class CourseAlreadyExistsException extends Exception {
    public CourseAlreadyExistsException() {
        super();
    }

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

```java
package com.trg.course.exception;

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

@ResponseStatus(value= HttpStatus.NOT_FOUND)
public class CourseNotFoundException extends RuntimeException {

    private String courseName, fieldName;
    private Long fieldValue;

    public CourseNotFoundException(String courseName, String fieldName, Long fieldValue) {
        super(String.format("%s not found with %s : '%s'", courseName, fieldName, fieldValue));
        this.courseName = courseName;
        this.fieldName = fieldName;
        this.fieldValue = fieldValue;
    }

}
```

```
package com.trg.course.exception;

import jakarta.servlet.http.HttpServletRequest;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import org.springframework.context.support.DefaultMessageSourceResolvable;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(CourseAlreadyExistsException.class)
    public ResponseEntity<String> handleCourseAlreadyExistsException(CourseAlreadyExistsException
e) {
        return new ResponseEntity("Course already exists", HttpStatus.CONFLICT);
    }

    @ExceptionHandler(CourseNotFoundException.class)
    public ResponseEntity<String> handleCourseNotFoundException(CourseNotFoundException e) {
        return new ResponseEntity("Course Not Found", HttpStatus.NOT_FOUND);
    }


    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<?> notValid(MethodArgumentNotValidException ex, HttpServletRequest
request) {
        List<String> errors = new ArrayList<>();

        ex.getAllErrors().forEach(err -> errors.add(err.getDefaultMessage()));
        Map<String, List<String>> result = new HashMap<>();
        result.put("errors", errors);

        return new ResponseEntity<>(result, HttpStatus.BAD_REQUEST);
    }

}
```

```
package com.trg.course.courseConsumer;

import com.trg.course.entity.Course;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.MediaType;
import org.springframework.web.client.RestClient;

import java.util.List;

public class CourseRestClientApp {
    private final RestClient restClient;

    public CourseRestClientApp() {
        restClient = RestClient.builder()
                .baseUrl("http://localhost:8080/course")
                .build();
    }

    public void getCourseById() {

        int cid=102;
        Course c = restClient.get()
                .uri("/courses/{id}", cid)
                .retrieve()
                .body(Course.class);
        System.out.println(c);
    }

    public void findAll() {
        List<Course> courseList = restClient.get()
                .uri("/courses")
                .retrieve()
                .body(new ParameterizedTypeReference<List<Course>>() {});

        courseList.forEach(course -> {
            System.out.println(course);
        });
    }
    public void createCourse() {
        Course c = new Course(106,"AWS","AWS desc");
        Course newCourse = restClient.post()
                .uri("/courses")
                .contentType(MediaType.APPLICATION_JSON)
                .body(c)
                .retrieve()
                .body(Course.class);

        System.out.println(newCourse);
    }

    public void deleteCourse() {
        int cid = 102;

        String response = restClient.delete()
                .uri("/courses/{id}", cid)
                .retrieve()
                .body(String.class);

        System.out.println(response);
    }
    public static void main(String[] args) {
        var app = new CourseRestClientApp();
        app.findAll();
        System.out.println("--------");
        app.createCourse();
        app.getCourseById();
        System.out.println("--------");
        app.findAll();
        app.deleteCourse();
        System.out.println("--------");
        app.findAll();
    }
}
```

```
package com.trg.course.courseConsumer;

import com.trg.course.entity.Course;
import org.springframework.web.client.RestTemplate;

import java.util.LinkedHashMap;
import java.util.List;

public class CourseRestTemplateApp {

    static final String REST_URI = "http://localhost:8080/course";

    static RestTemplate restTemplate = new RestTemplate();
```

```
private static void listAllCourses() {
   System.out.println("\n Testing listAllPersons API-----------");
   List<LinkedHashMap<String, Object>> coursesMap =
          restTemplate.getForObject(REST_URI + "/courses", List.class);

   if (coursesMap != null) {
      for (LinkedHashMap<String, Object> map : coursesMap)
          System.out.println("Course : id=" + map.get("id") +
                 ", name=" + map.get("name") +
                 ", Desc=" + map.get("desc"));
   } else
      System.out.println("No course exists----------");
}

private static void getCourse(int id) {
   System.out.println("\n Testing getPerson API----------");
   Course course =
          restTemplate.getForObject(REST_URI + "/courses/" + id, Course.class);
   System.out.println(course);
}

private static void createCourse(Course c) {
   System.out.println("\n Testing create Course API----------");
   Course course =
          restTemplate.postForObject(REST_URI + "/courses", c, Course.class);
   System.out.println("Newly created course : " + course);
}

private static void deleteCourse(int id) {
   System.out.println("\n Testing delete Course API----------");
   restTemplate.delete(REST_URI + "/courses/" + id);
}

private static void updateCourse(int id, Course c) {
   System.out.println("\n Testing update Course API----------");
   restTemplate.put(REST_URI + "/courses"+ id, c);
}

public static void main(String args[]) {
   listAllCourses();
   getCourse(101);
   Course c = new Course(105,"ReactJS","React desc");
   createCourse(c);
   listAllCourses();
   deleteCourse(101);
   listAllCourses();
   //Course c1 = new Course(105,"ReactJS","React Beginner version");
   //updateCourse(105,c1);
```

```
        //listAllCourses();

    }
}
```