# REST API Testing Overview

# Student Grade Book - REST API

- We will start with an existing Student Grade Book - REST API
- Similar to the Student Grade Book app that we worked on in previous videos
- Replaced HTML interface with a REST API
- The REST API was created by a previous employee ... but it is does not have any tests (yikes!)

- Our job:
  - Add tests for the REST API

# Technical Stack

- Spring Boot
- Spring Data JPA
- Spring @RestController

# Existing Code (partial)

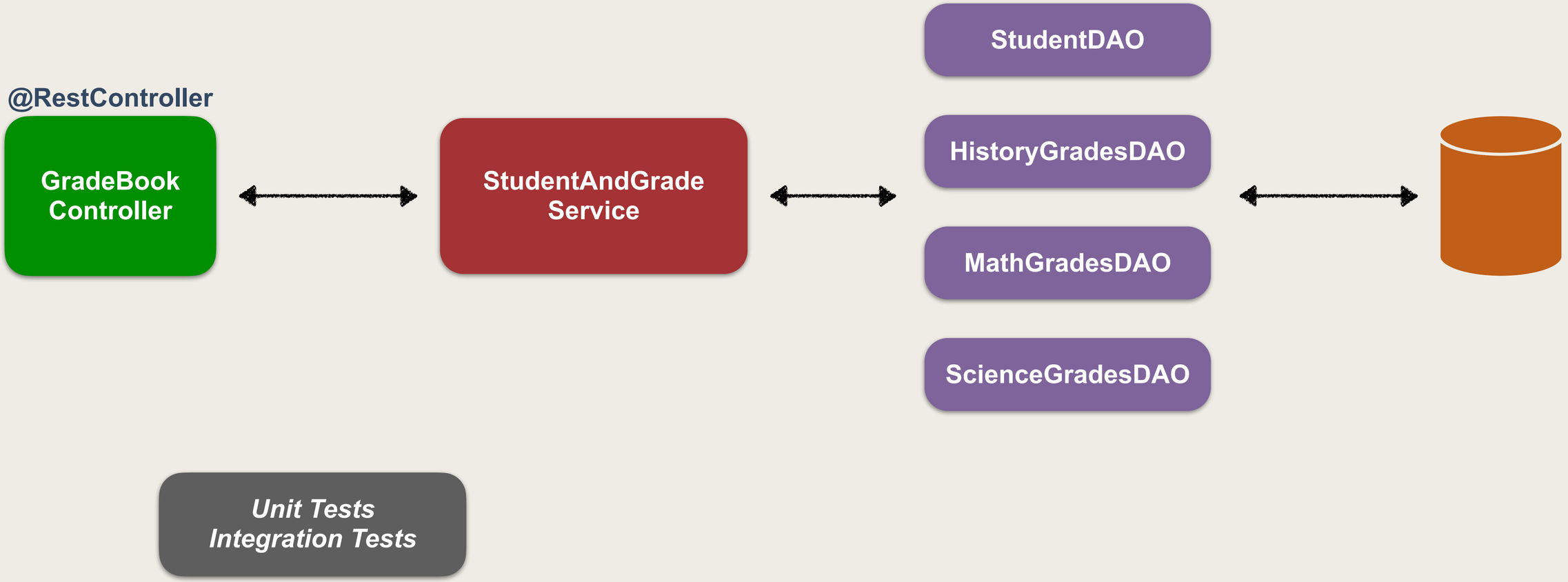
@RestController

```
GradeBookController.java
```

Model

```
CollegeStudent.java
Grade.java
Gradebook.java
GradebookCollegeStudent.java
HistoryGrade.java
MathGrade.java
ScienceGrade.java
Student.java
StudentGrades.java
```

# Application Architecture



luv2code

# Testing REST APIs
# Spring REST Controllers

# Problem

- How can we test REST API developed with Spring REST Controllers?
- How can we create HTTP requests and send to the Spring REST controller?
- How can we verify HTTP response?
  - status code
  - content type
  - JSON response body

# Spring Testing Support

- For testing Spring REST controllers, you can use `MockMvc`
- Provides Spring REST processing of request / response
- There is no need to run a server (embedded or external)

**In general, the process is VERY similar to Spring MVC testing**

**Slight differences for content-type and checking JSON results**

# Preliminary Set Up

- For GradebookControllerTest
- Stub out the test class
- Define fields that we'll use later: MockMvc, Service, DAOs etcs
- @BeforeAll, @BeforeEach, @AfterEach

GradebookControllerTest.java

```
...
@TestPropertySource("/application-test.properties")
@AutoConfigureMockMvc
@SpringBootTest
public class GradebookControllerTest {

   // inject support utils

   // inject Service and DAOs

   // inject SQL strings

   // @BeforeAll, @BeforeEach and @AfterEach

}
```

# Test - Get Students

# Get Students Endpoint

- Get a list of students as a JSON array

http://localhost:1500/

Web browser will send a GET request


```
[
  {
    "id": 10,
    "firstname": "David",
    "lastname": "Adams",
    "emailAddress": "david@luv2code.com",
    "studentGrades": {
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    },
    "fullName": "David Adams"
  },
  {
    "id": 11,
    "firstname": "John",
    "lastname": "Doe",
    "emailAddress": "john@luv2code.com",
    "studentGrades": {
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    },
    "fullName": "John Doe"
  },
  ...
]
```

# GradebookController

```java
@RestController
public class GradebookController {

    @Autowired
    private StudentAndGradeService studentService;

    @Autowired
    private Gradebook gradebook;

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public List<GradebookCollegeStudent> getStudents() {
        gradebook = studentService.getGradebook();
        return gradebook.getStudents();
    }

    ...
}
```


```
[
  {
    "id": 10,
    "firstname": "David",
    "lastname": "Adams",
    "emailAddress": "david@luv2code.com",
    "studentGrades": {
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    },
    "fullName": "David Adams"
  },
  {
    "id": 11,
    "firstname": "John",
    "lastname": "Doe",
    "emailAddress": "john@luv2code.com",
    "studentGrades": {
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    },
    "fullName": "John Doe"
  },
  ...
]
```

# Verifying the HTTP Response

- How can we verify HTTP response?
  - status code
  - content type
  - JSON response body

# Verifying HTTP Status and Content Type

```
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

...

@TestPropertySource("/application-test.properties")
@AutoConfigureMockMvc
@SpringBootTest
public class GradebookControllerTest {

   ...

   @Test
   public void getStudentsHttpRequest() throws Exception {

      mockMvc.perform(MockMvcRequestBuilders.get("/"))
             .andExpect(status().isOk())
             .andExpect(content().contentType(APPLICATION_JSON_UTF8));

   }

   ...

}
```

HTTP status of 200 (OK)

application/json

# Verifying JSON Response Body

- How can we verify JSON response body?
  - Access specific JSON element (even for nested elements)
  - Size of the JSON  array

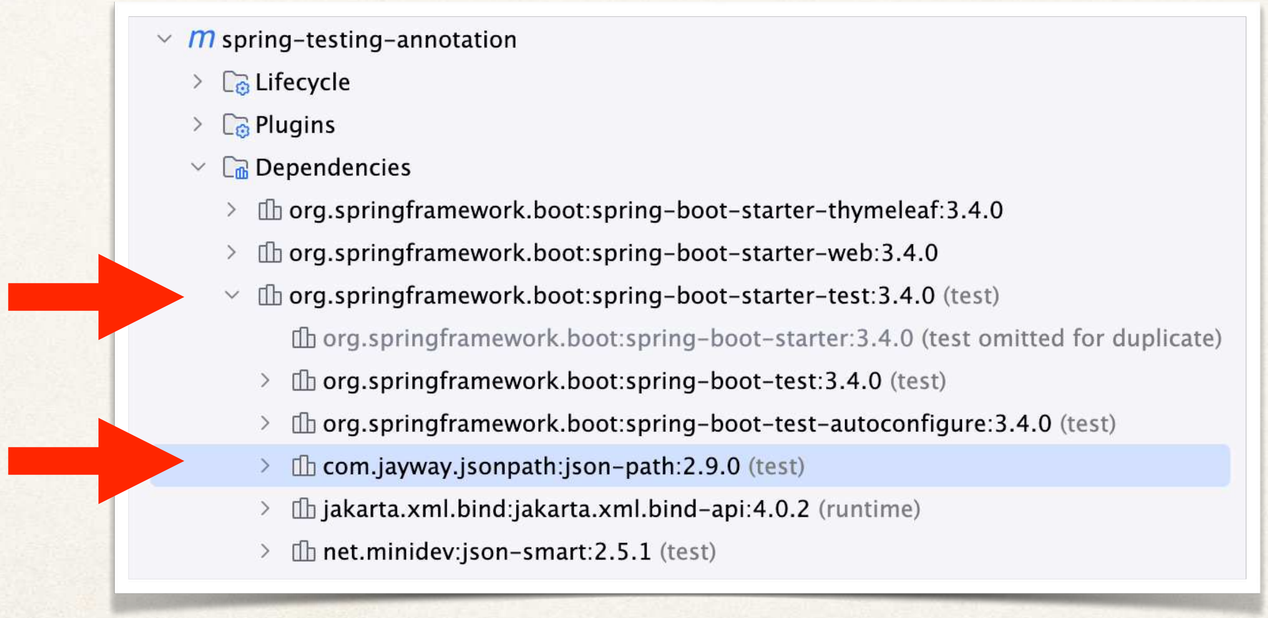
# Solution: JsonPath

- **JsonPath** allows you to access elements of JSON
- Open-source project

**https://github.com/json-path/JsonPath**

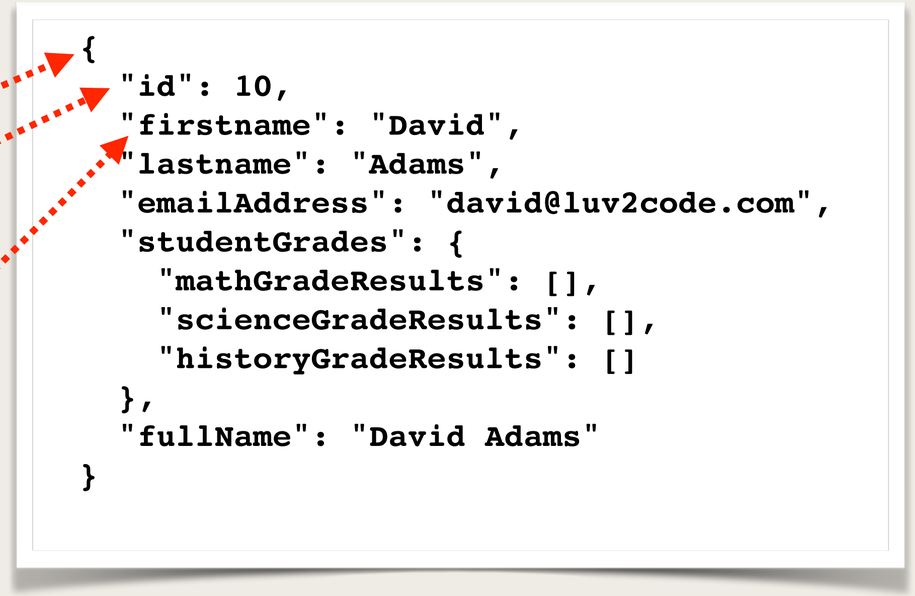
# JsonPath support in Spring Boot Test

- Spring Boot Test starter includes support for JsonPath
- No need for you to manually add JsonPath dependency



# JsonPath Examples

| JsonPath | Result |
|---|---|
| `$` | The root element to query. Starts all path expressions |
| `$.id` | Access the **`id`** element of the JSON element |
| `$.firstname` | Access the **`firstname`** element of the JSON element |
| `...` | `...` |



```
{
  "id": 10,
  "firstname": "David",
  "lastname": "Adams",
  "emailAddress": "david@luv2code.com",
  "studentGrades": {
    "mathGradeResults": [],
    "scienceGradeResults": [],
    "historyGradeResults": []
  },
  "fullName": "David Adams"
}
```

**https://github.com/json-path/JsonPath**

# JsonPath - Verifying Array Size

```
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.hamcrest.Matchers.hasSize;
...

@TestPropertySource("/application-test.properties")
@AutoConfigureMockMvc
@SpringBootTest
public class GradebookControllerTest {

   ...

   @Test
   public void getStudentsHttpRequest() throws Exception {

      mockMvc.perform(MockMvcRequestBuilders.get("/"))
             .andExpect(status().isOk())
             .andExpect(content().contentType(APPLICATION_JSON_UTF8))
             .andExpect(jsonPath("$", hasSize(2));

   }

   ...

}
```

Root element

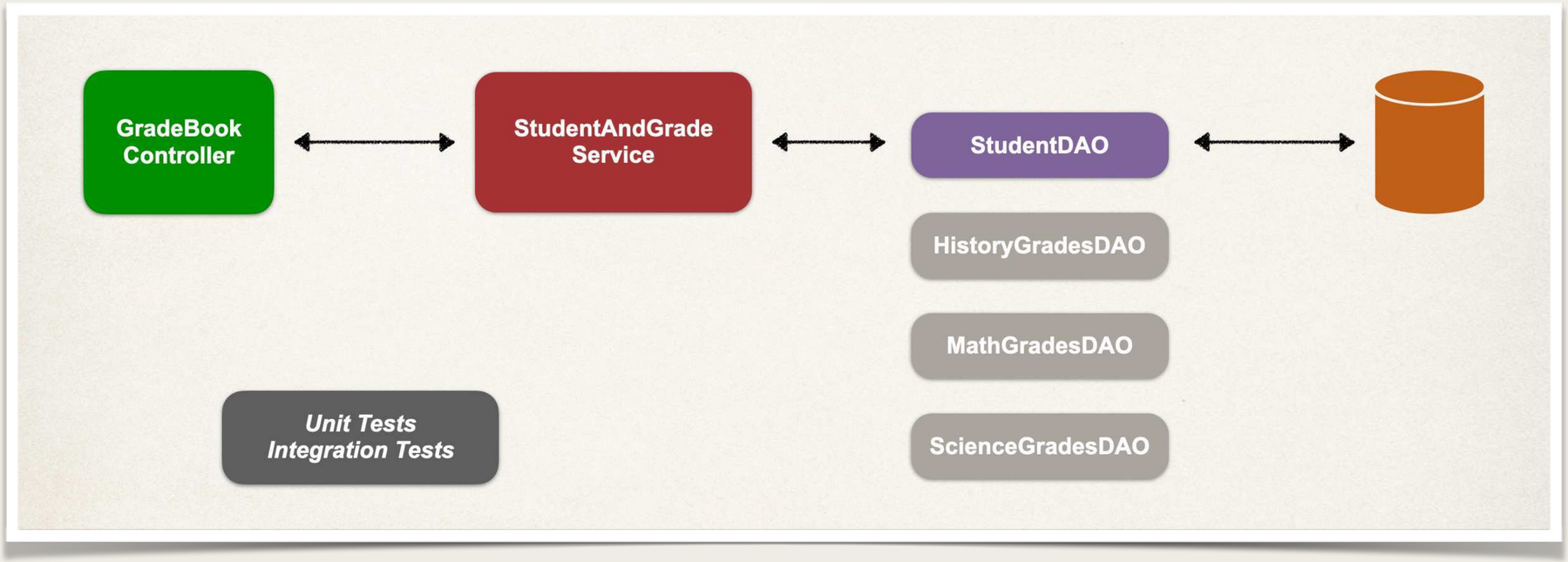
Verify JSON array size is 2

```
[
  {
    "id": 10,
    "firstname": "David",
    "lastname": "Adams",
    "emailAddress": "david@luv2code.com",
    "studentGrades": {
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    },
    "fullName": "David Adams"
  },
  {
    "id": 11,
    "firstname": "John",
    "lastname": "Doe",
    "emailAddress": "john@luv2code.com",
    "studentGrades": {
      "mathGradeResults": [],
      "scienceGradeResults": [],
      "historyGradeResults": []
    },
    "fullName": "John Doe"
  }
]
```

# Test - Create Student

# Test Case: Create a student in the database

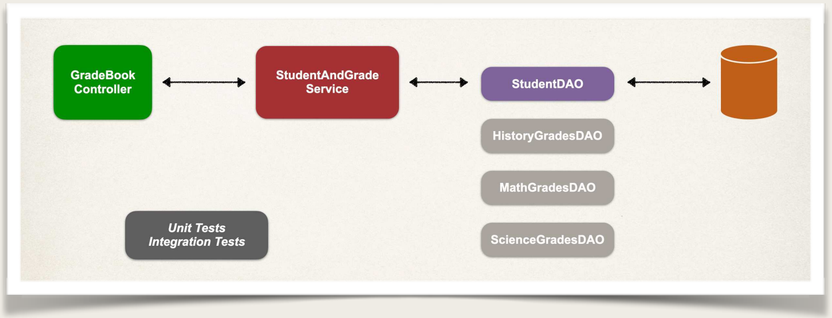
- Send a POST request to the @RestController
- Verify results of JSON response
- Also verify results by accessing data using the DAO



# Test - Delete Student

# To Do: Delete Student

**DELETE /student/{id}**

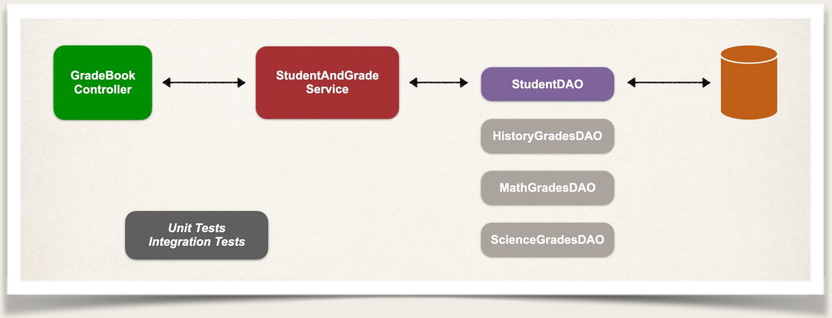


# Test - Student Information

# To Do: Student Information

- Create a test to retrieve student information
  - Student name, email address, student grades etc ..

**GET /studentInformation/{id}**



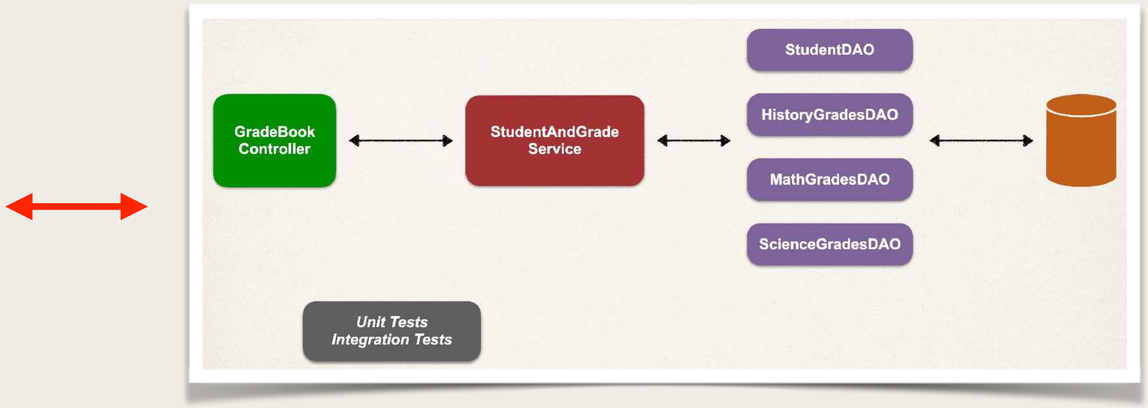
# Test - Create Grades

# To Do: Create Grades

- Develop a test for creating grades

**POST /grades**

**Params:**
**- grade**
**- gradeType**
**- studentId**



# Test - Delete Grade

# To Do: Delete Grade

- Develop a test for deleting grades

**DELETE /grades/{id}/{gradeType}**