REST APIs - REST Web Services

You will learn how to …

You will learn how to …

- Create REST APIs / Web Services with Spring

You will learn how to …

- Create REST APIs / Web Services with Spring
- Discuss REST concepts, JSON and HTTP messaging

You will learn how to …

- Create REST APIs / Web Services with Spring
- Discuss REST concepts, JSON and HTTP messaging
- Install REST client tool: Postman

You will learn how to …

- Create REST APIs / Web Services with Spring
- Discuss REST concepts, JSON and HTTP messaging
- Install REST client tool: Postman
- Develop REST APIs / Web Services with **@RestController**

You will learn how to …

- Create REST APIs / Web Services with Spring
- Discuss REST concepts, JSON and HTTP messaging
- Install REST client tool: Postman
- Develop REST APIs / Web Services with **@RestController**
- Build a CRUD interface to the database with Spring REST

Practical Results

Practical Results

- Introduction to Spring REST development

Practical Results

- Introduction to Spring REST development

- Not an A to Z reference … for that you can see Spring Reference Manual

Practical Results

- Introduction to Spring REST development
- Not an A to Z reference … for that you can see Spring Reference Manual

https://projects.spring.io/spring-framework/

Business Problem

Business Problem

- Build a client app that provides the weather report for a city

Business Problem

- Build a client app that provides the weather report for a city
- Need to get weather data from an external service

Application Architecture

Application Architecture

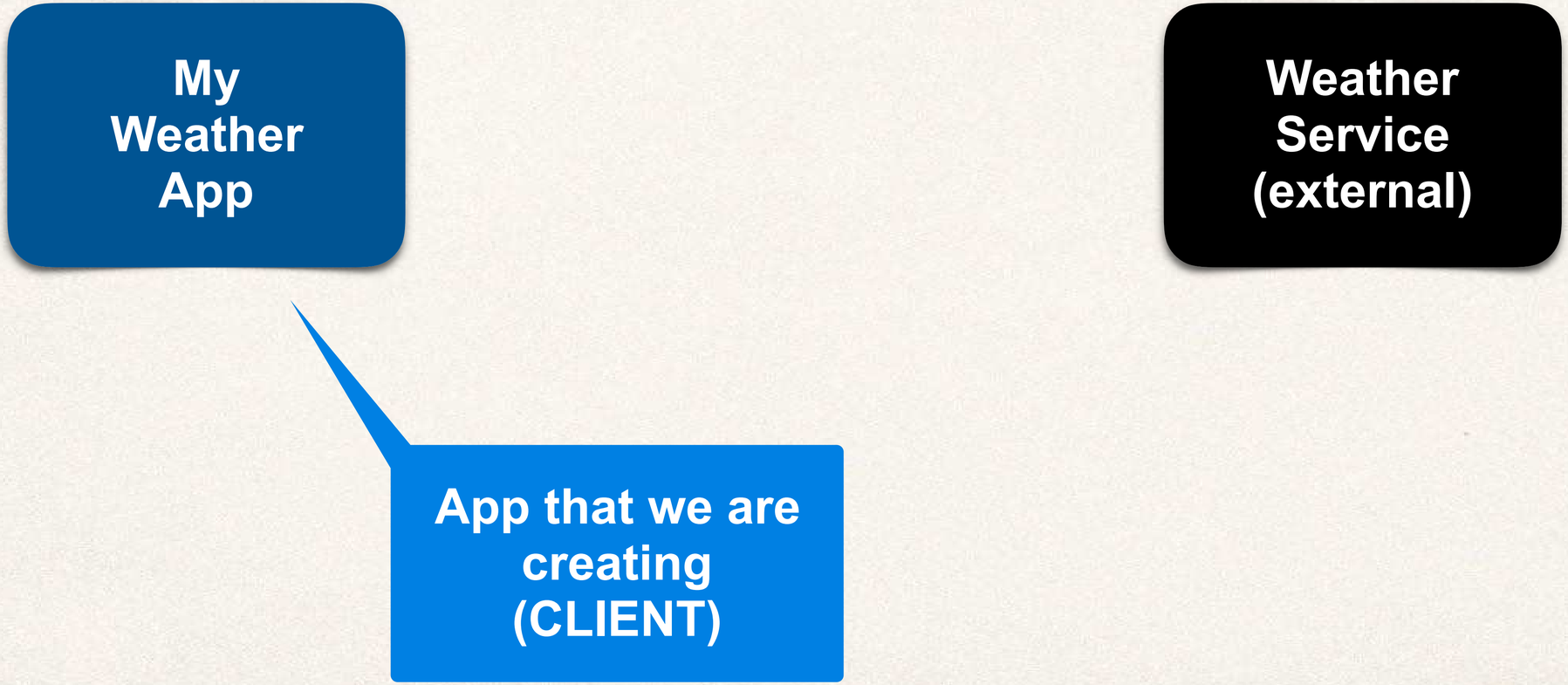
My
Weather
App

Application Architecture

My
Weather
App

App that we are
creating
(CLIENT)

Application Architecture



Application Architecture

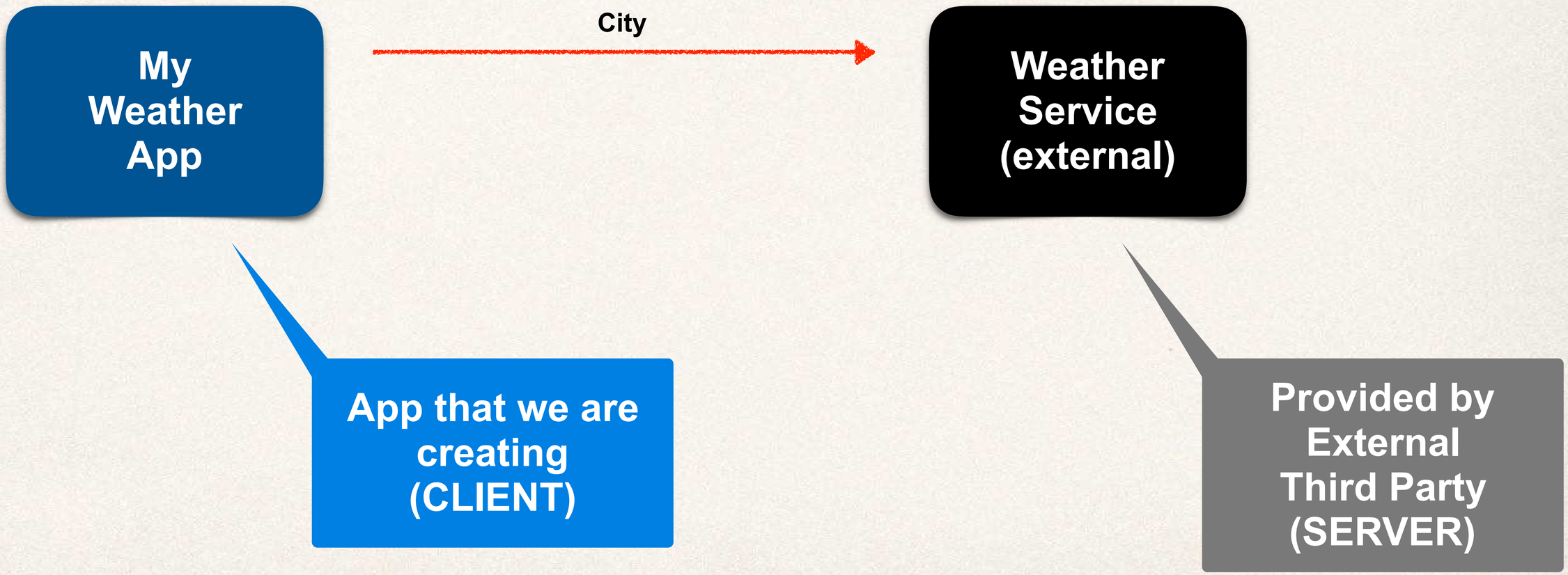
My Weather App

App that we are creating (CLIENT)

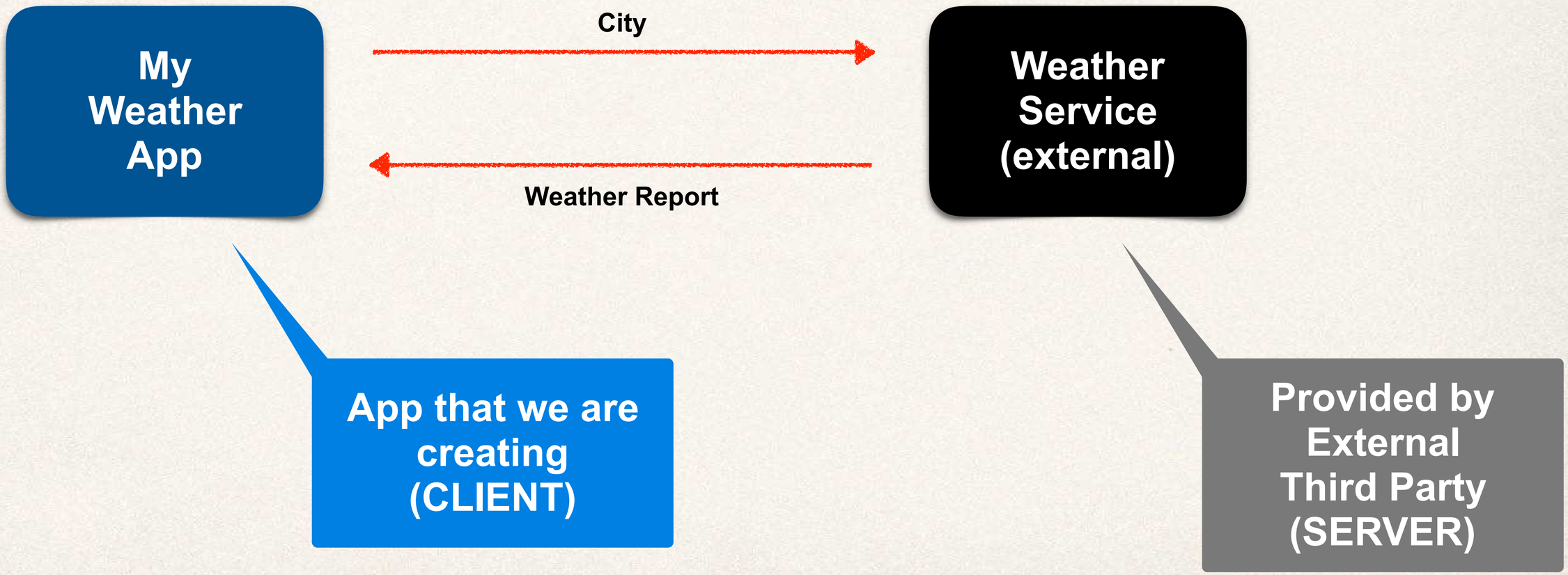
Weather Service (external)

Provided by External Third Party (SERVER)

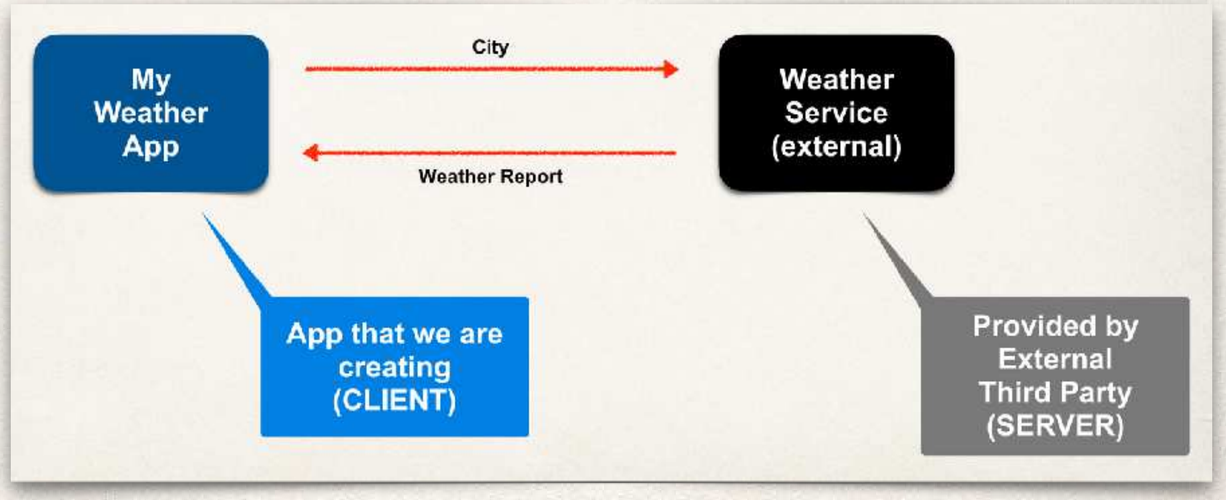
Application Architecture



Application Architecture

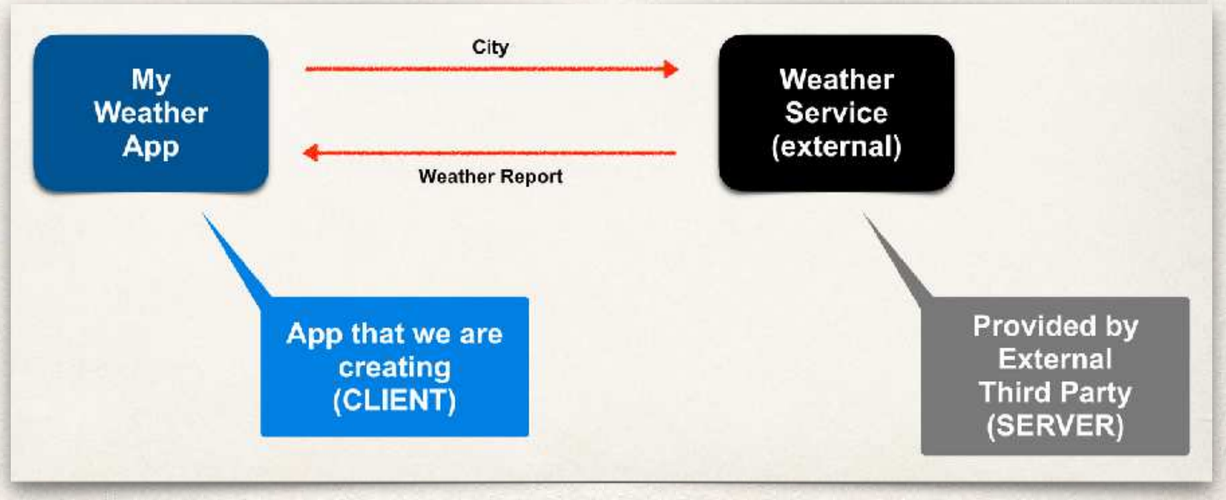


Questions


My Weather App
City
Weather Report
Weather Service (external)
App that we are creating (CLIENT)
Provided by External Third Party (SERVER)

Questions

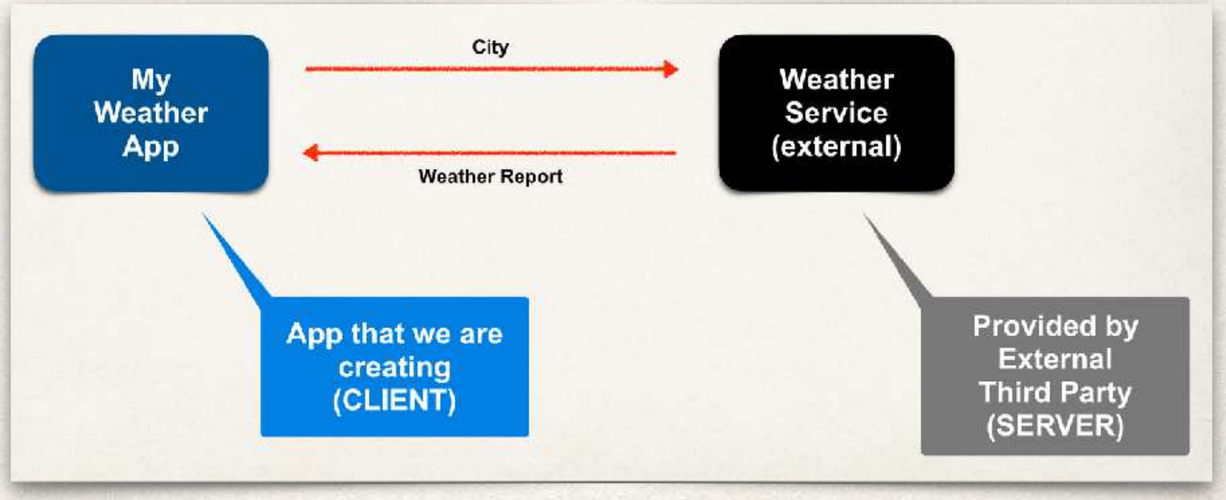
- How will we connect to the Weather Service?



Questions

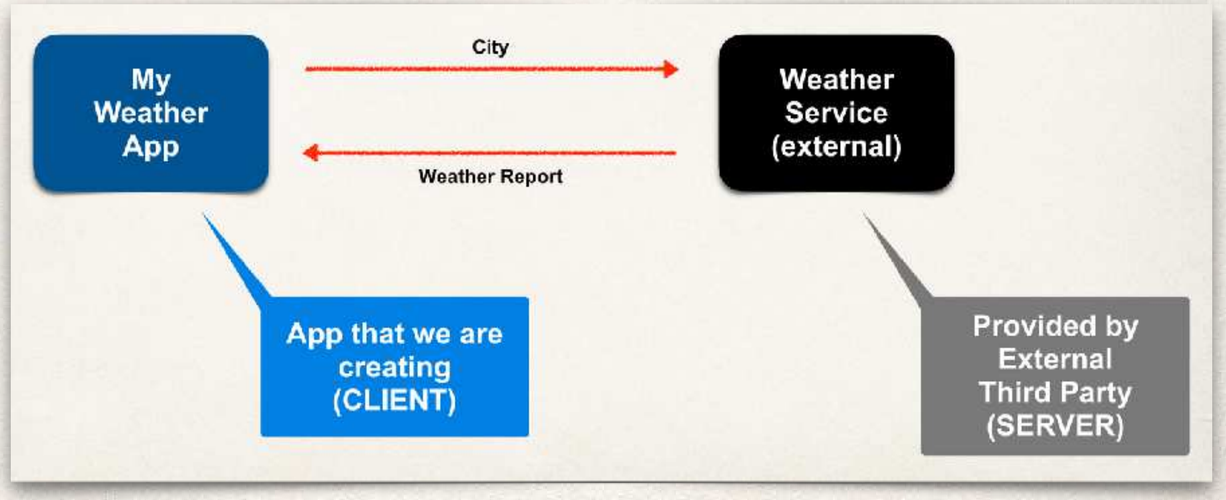
- How will we connect to the Weather Service?

- What programming language do we use?

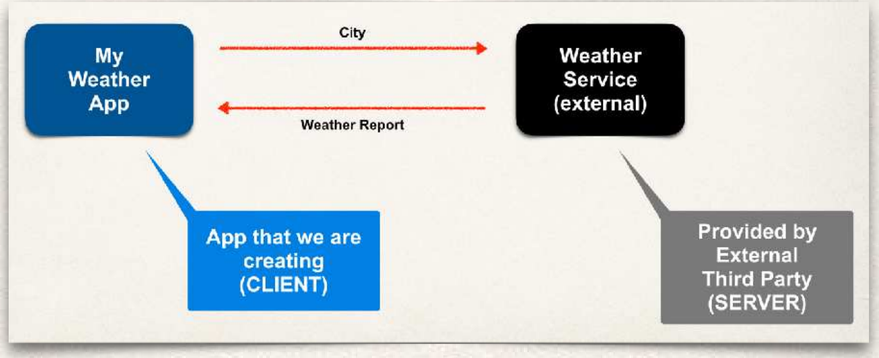


Questions

- How will we connect to the Weather Service?
- What programming language do we use?
- What is the data format?

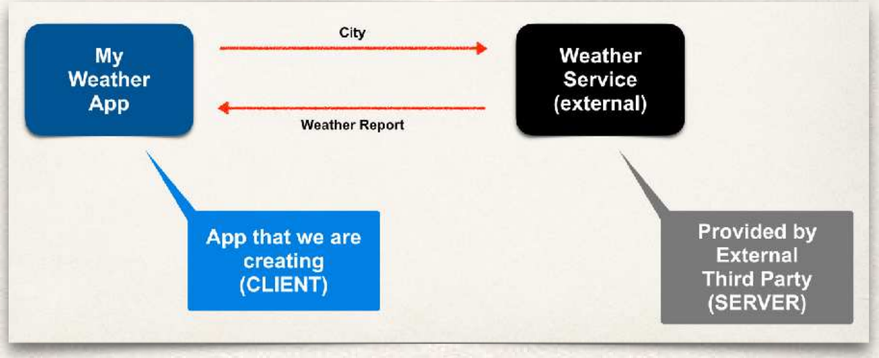


Answers


My Weather App
City
Weather Report
Weather Service (external)
App that we are creating (CLIENT)
Provided by External Third Party (SERVER)

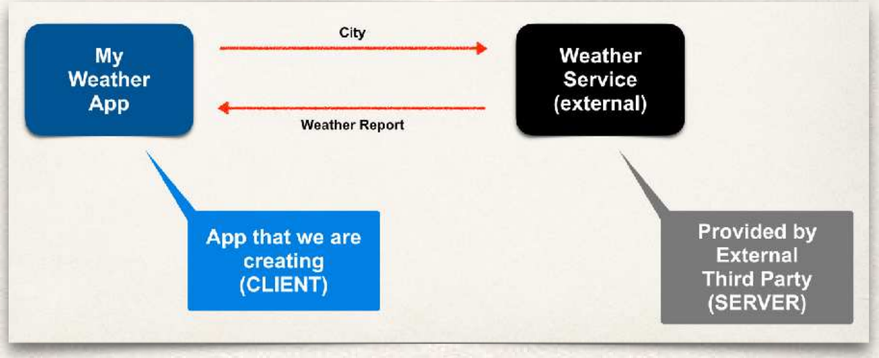
Answers

- **How will we connect to the Weather Service?**



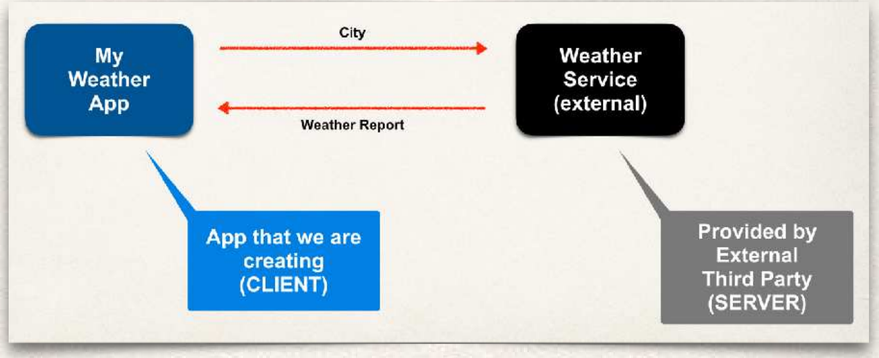
Answers

- **How will we connect to the Weather Service?**
  - We can make REST API calls over HTTP



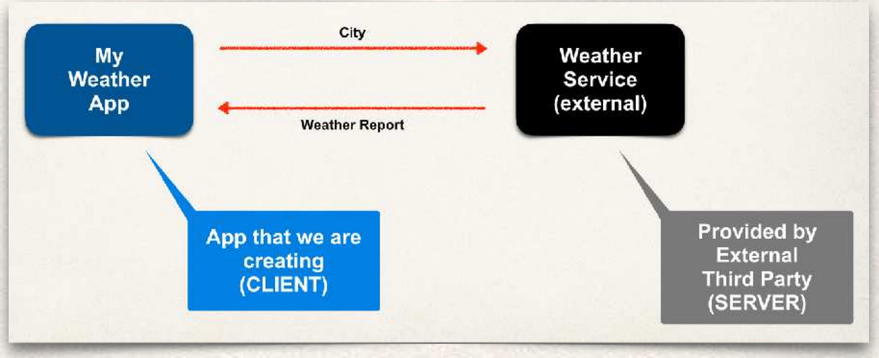
Answers

- **How will we connect to the Weather Service?**
  - We can make REST API calls over HTTP
  - REST: REpresentational State Transfer

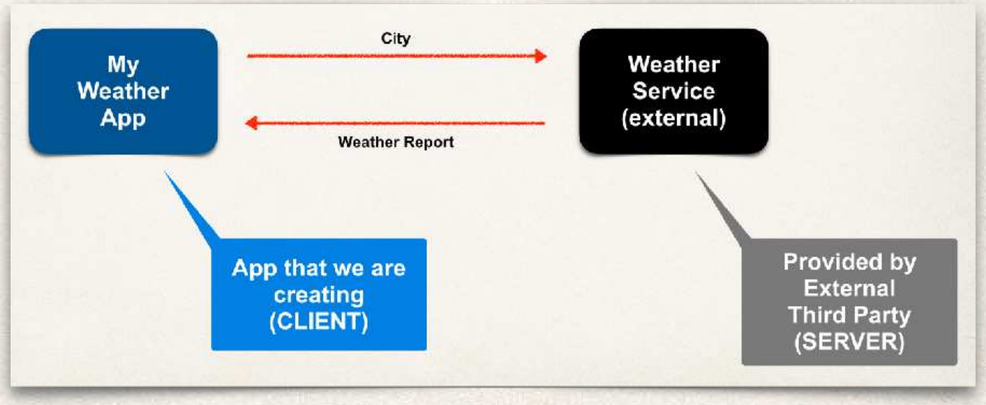


Answers

- **How will we connect to the Weather Service?**
  - We can make REST API calls over HTTP
  - REST: REpresentational State Transfer
  - Lightweight approach for communicating between applications

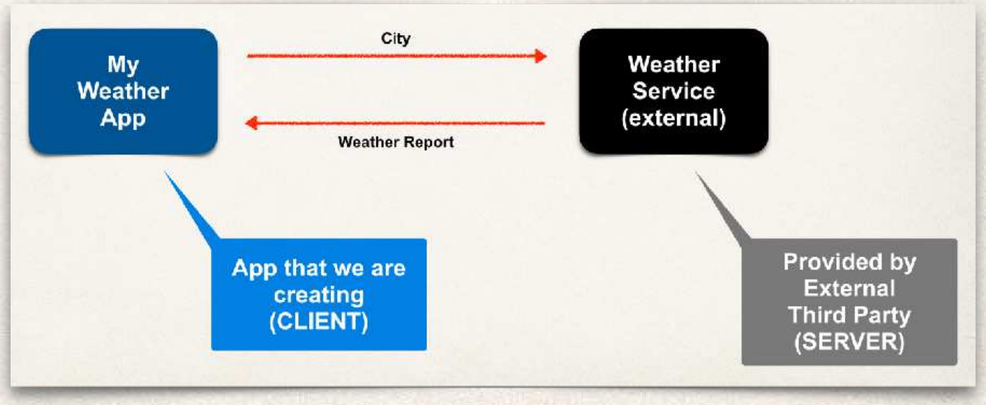


Answers


My Weather App
City
Weather Report
Weather Service (external)
App that we are creating (CLIENT)
Provided by External Third Party (SERVER)

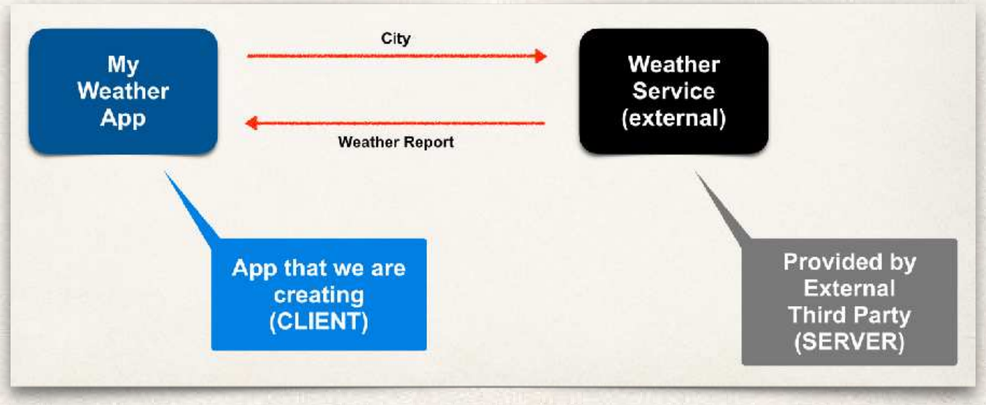
Answers

- **What programming language do we use?**



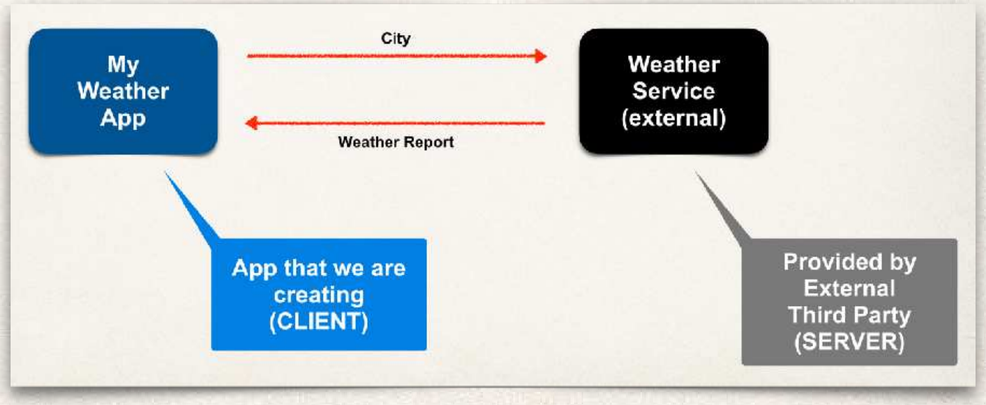
Answers

- **What programming language do we use?**
  - REST is language independent



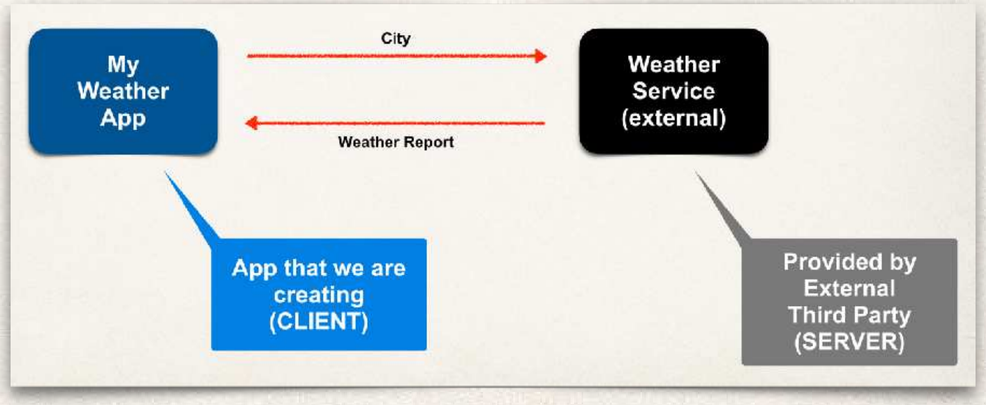
Answers

- **What programming language do we use?**
  - REST is language independent
  - The **client** application can use **ANY** programming language



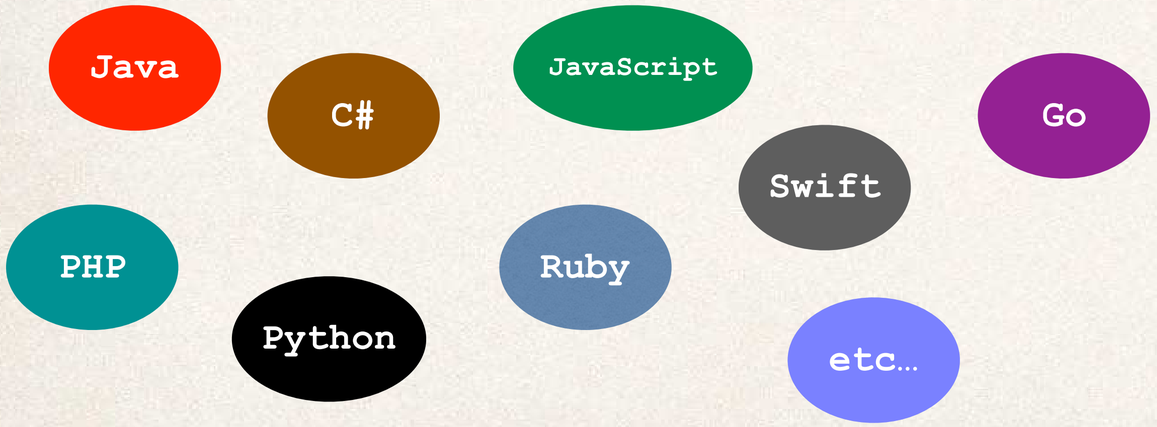
Answers

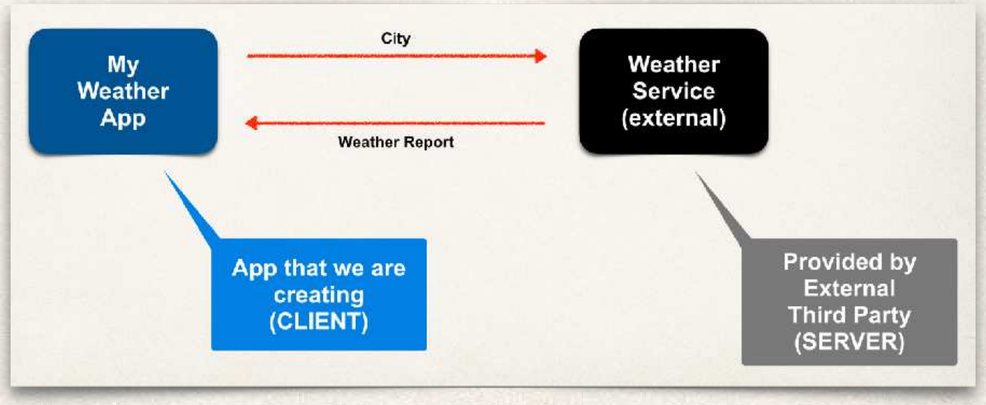
- **What programming language do we use?**
  - REST is language independent
  - The **client** application can use **ANY** programming language
  - The **server** application can use **ANY** programming language



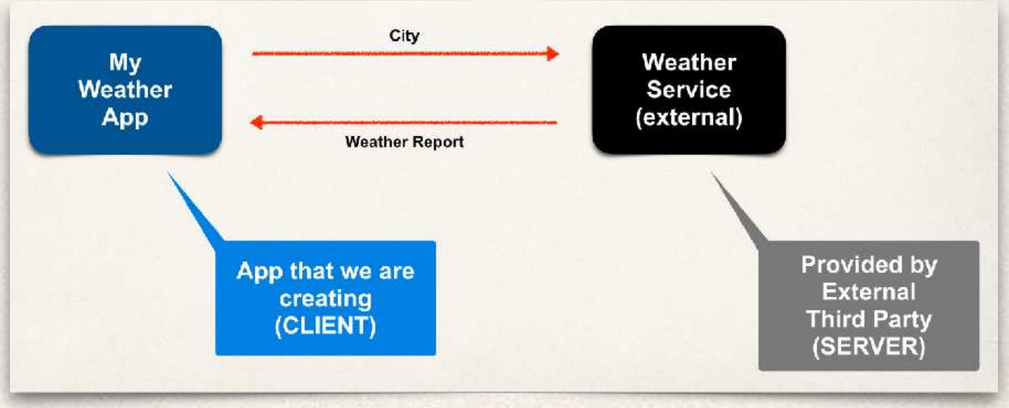
Answers

- **What programming language do we use?**
  - REST is language independent
  - The **client** application can use **ANY** programming language
  - The **server** application can use **ANY** programming language



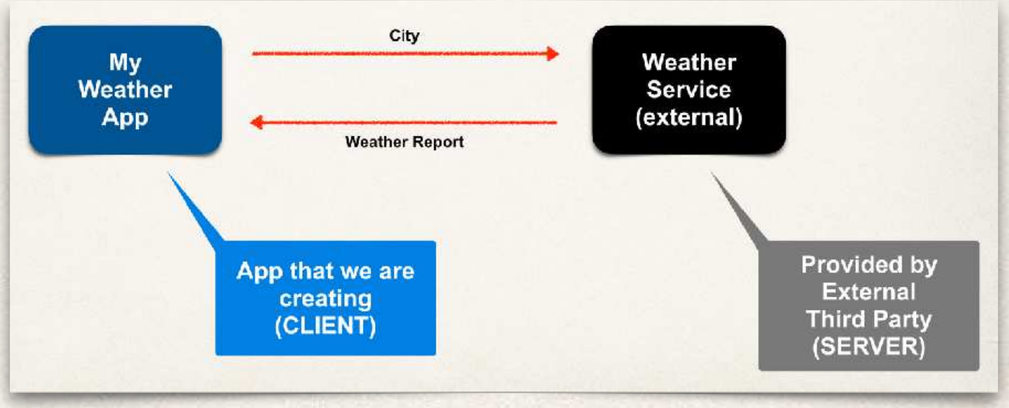


Answers


My Weather App
City
Weather Report
Weather Service (external)
App that we are creating (CLIENT)
Provided by External Third Party (SERVER)

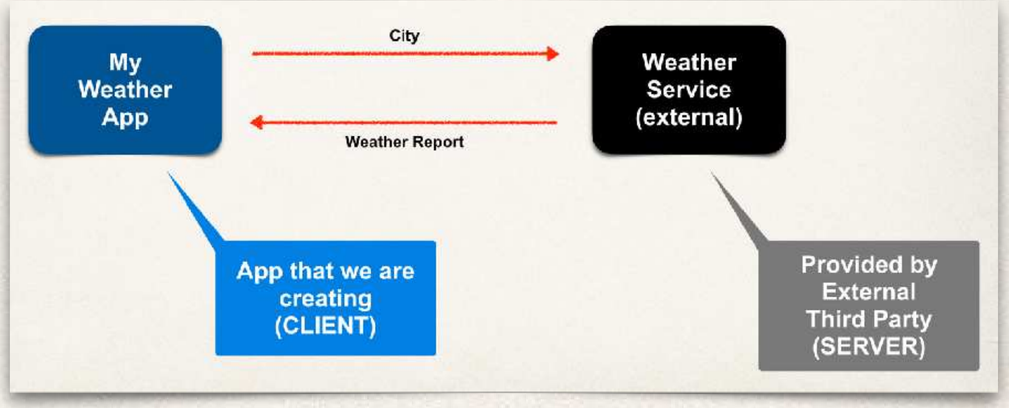
Answers

- **What is the data format?**



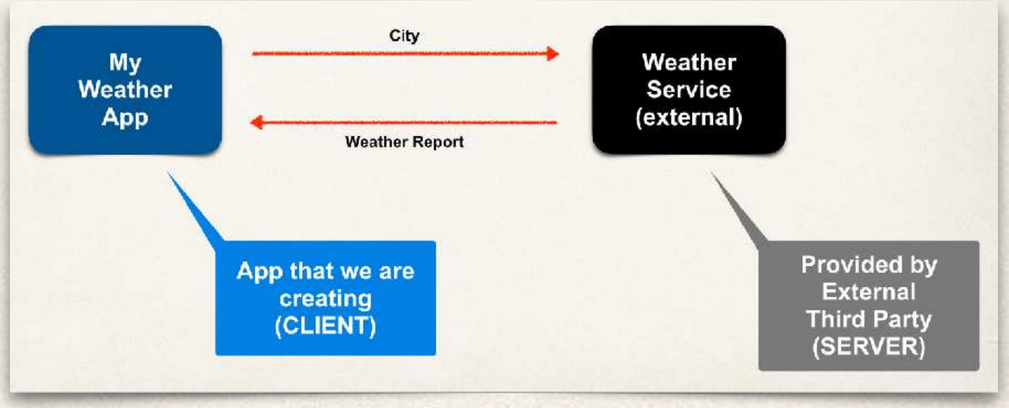
Answers

- **What is the data format?**
  - REST applications can use any data format



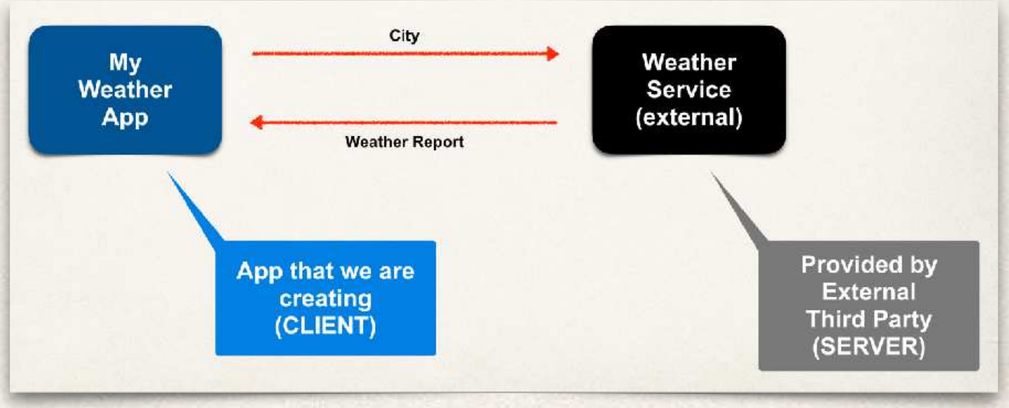
Answers

- **What is the data format?**
  - REST applications can use any data format
  - Commonly see XML and JSON



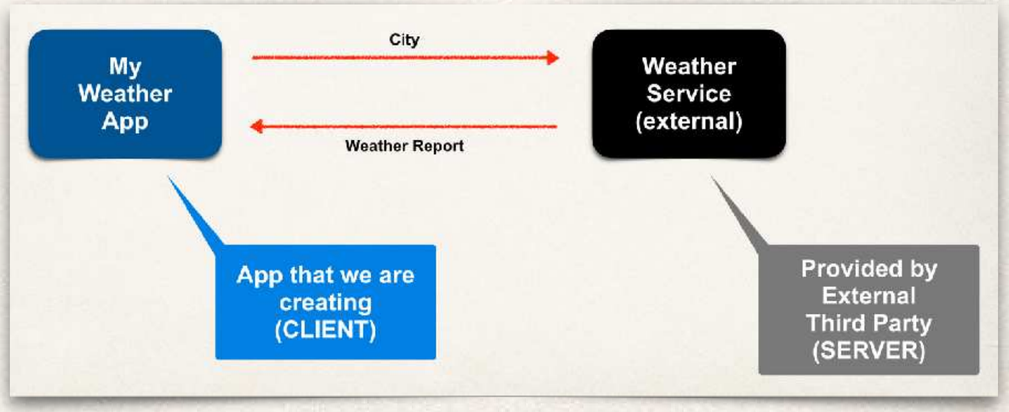
Answers

- **What is the data format?**
  - REST applications can use any data format
  - Commonly see XML and JSON
  - JSON is most popular and modern



Answers

- **What is the data format?**
  - REST applications can use any data format
  - Commonly see XML and JSON
  - JSON is most popular and modern
    - JavaScript Object Notation

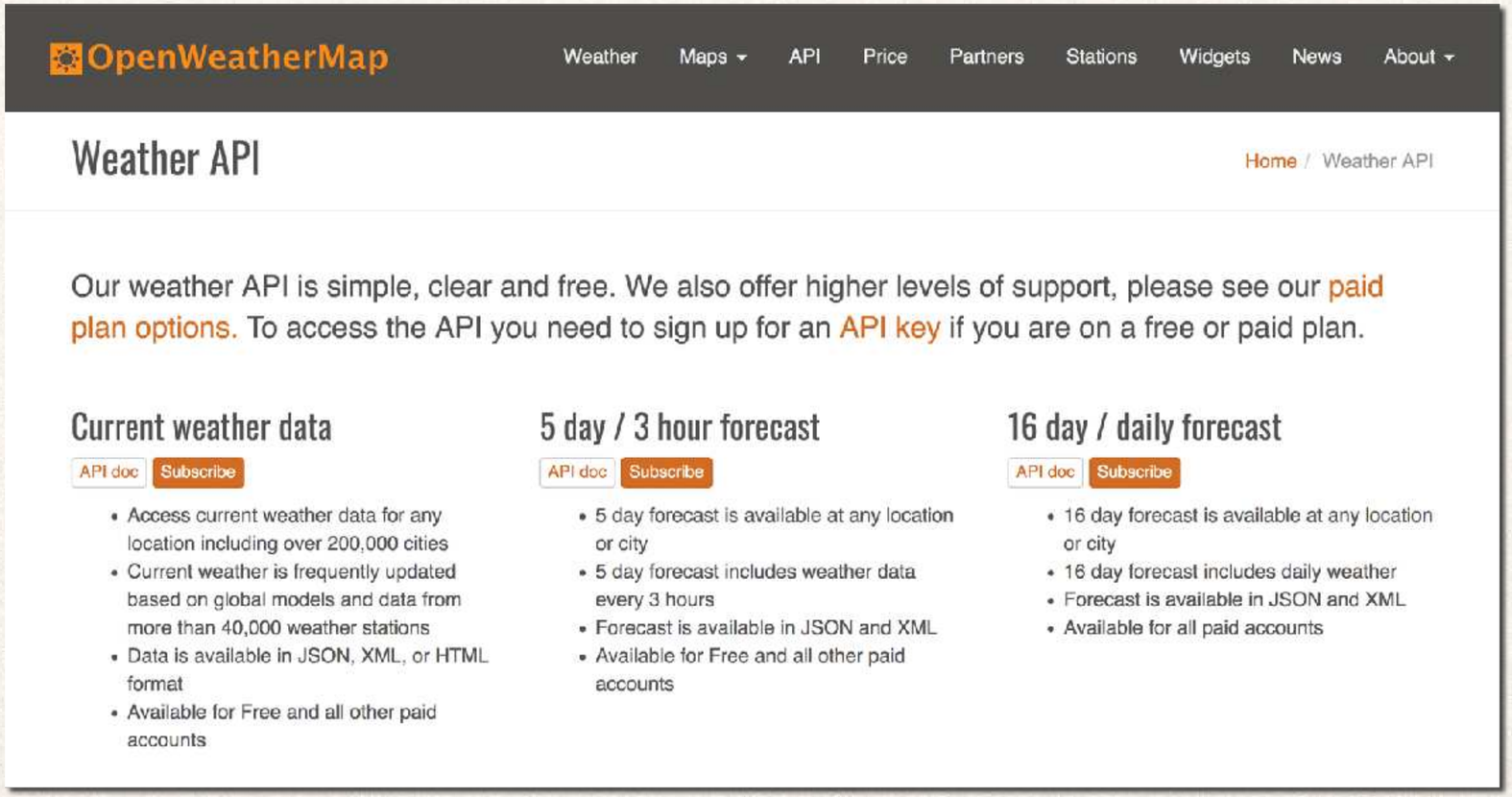


Possible Solution

- Use free Weather Service provided by: openweathermap.org

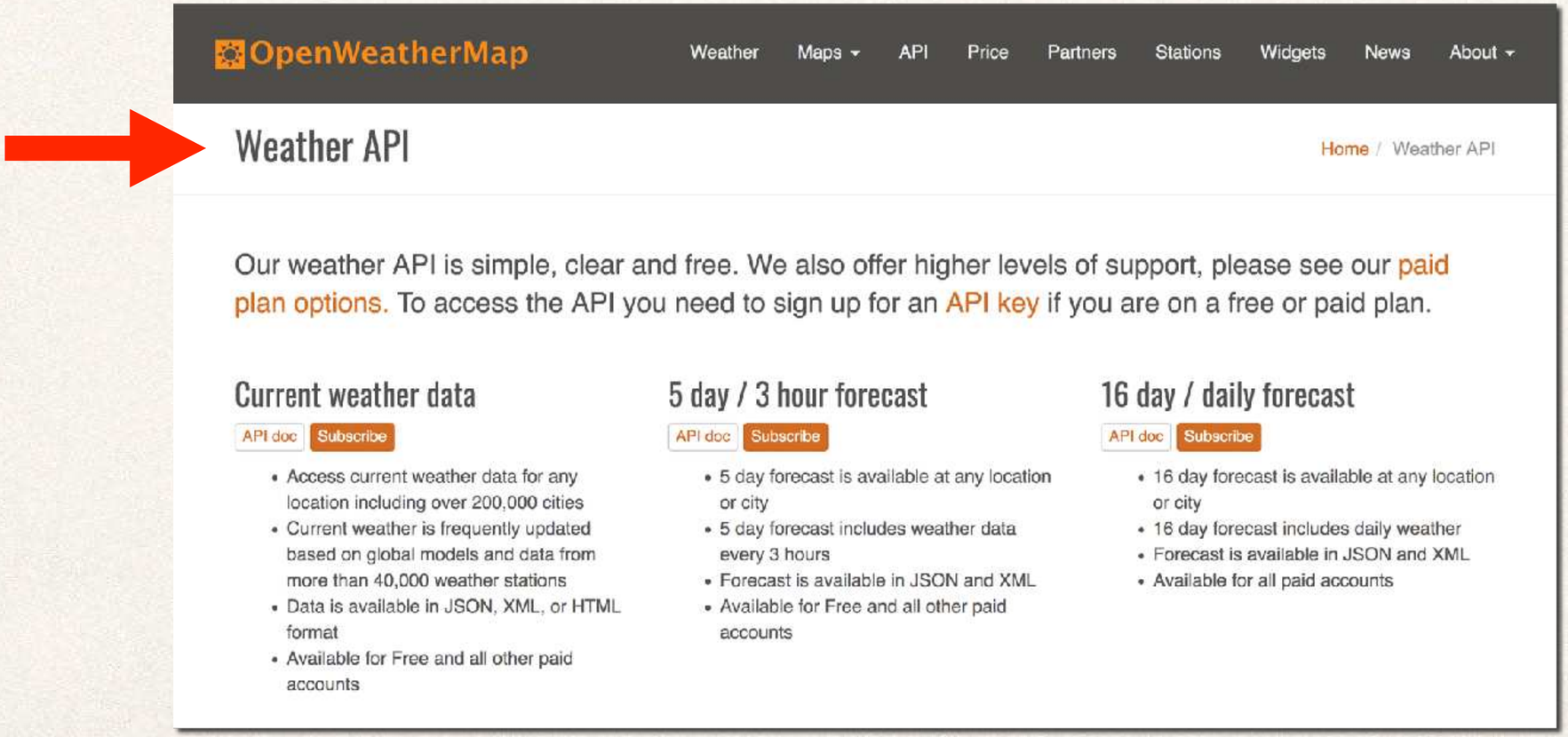
Possible Solution

- Use free Weather Service provided by: openweathermap.org



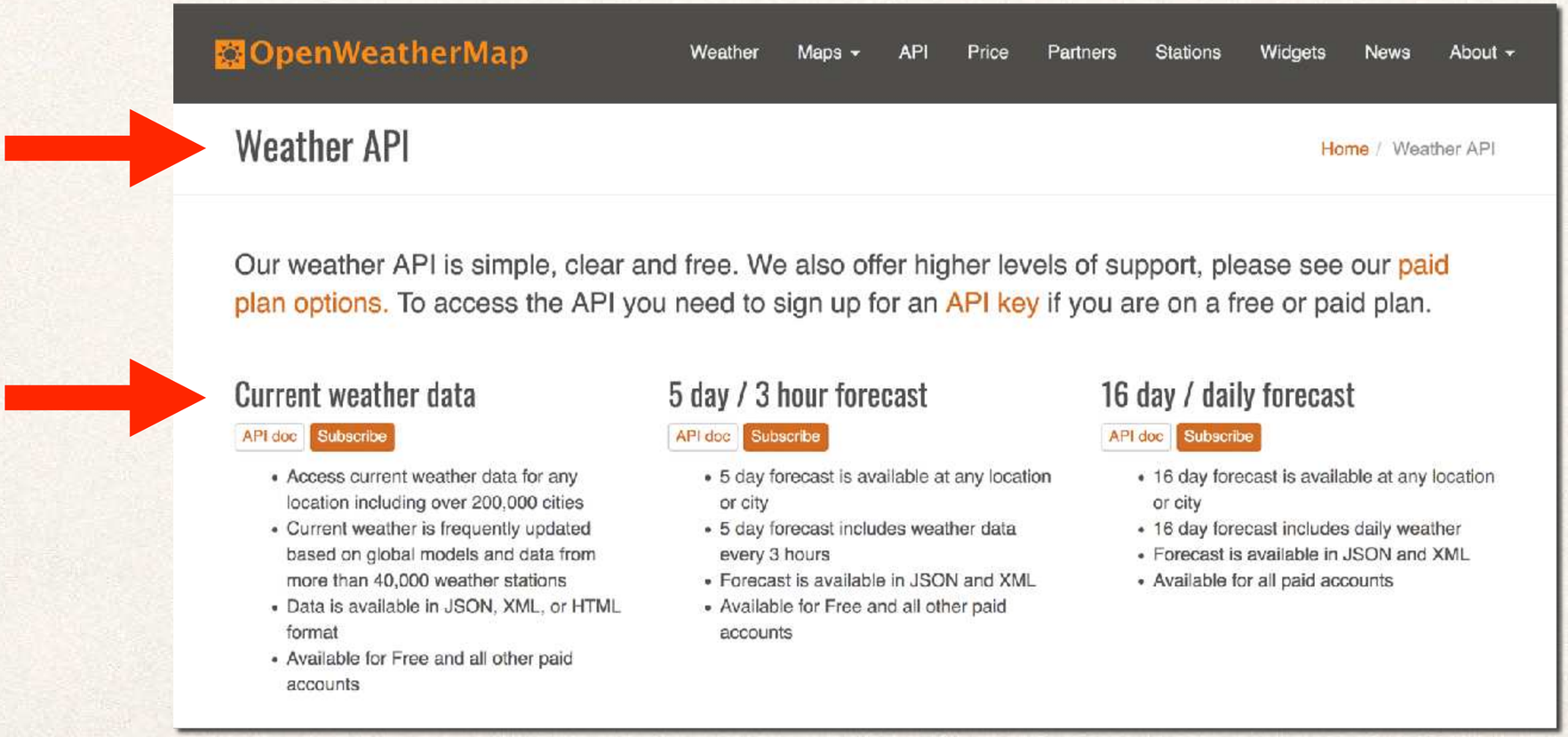
Possible Solution

- Use free Weather Service provided by: openweathermap.org

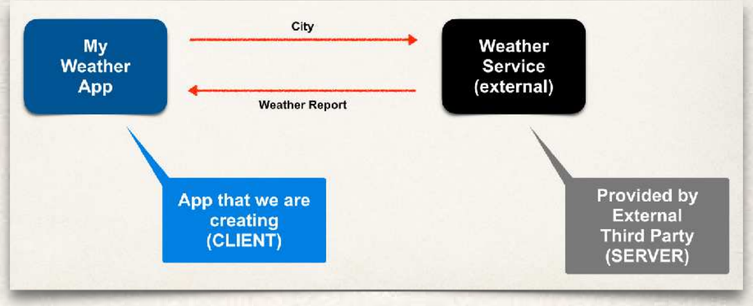


Possible Solution

- Use free Weather Service provided by: openweathermap.org

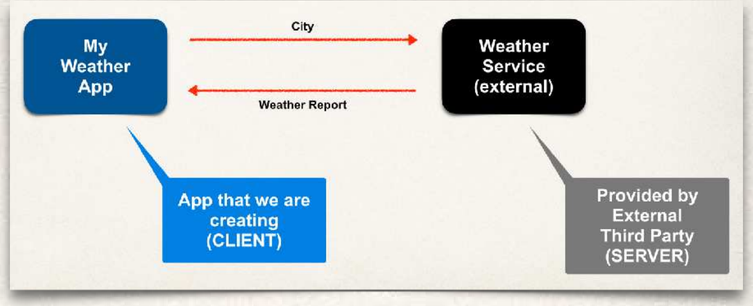


Call Weather Service



- The API documentation gives us the following:
  - Pass in the city name

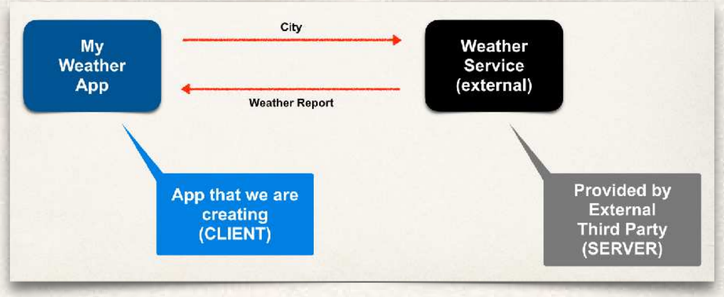
Call Weather Service



- The API documentation gives us the following:
  - Pass in the city name

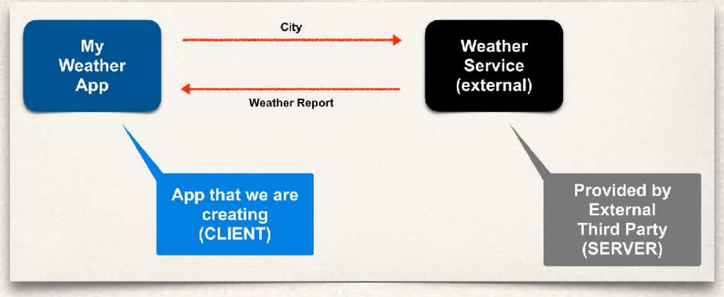
```
api.openweathermap.org/data/2.5/weather?q={city name}

OR

api.openweathermap.org/data/2.5/weather?q={city name},{country code}
```

Response - Weather Report



- The Weather Service responds with JSON

Response - Weather Report



- The Weather Service responds with JSON

```
{
  ...
  "temp": 14,
  "temp_min": 11,
  "temp_max": 17,
  "humidity": 81,
  "name": "London",
  ...
}
```

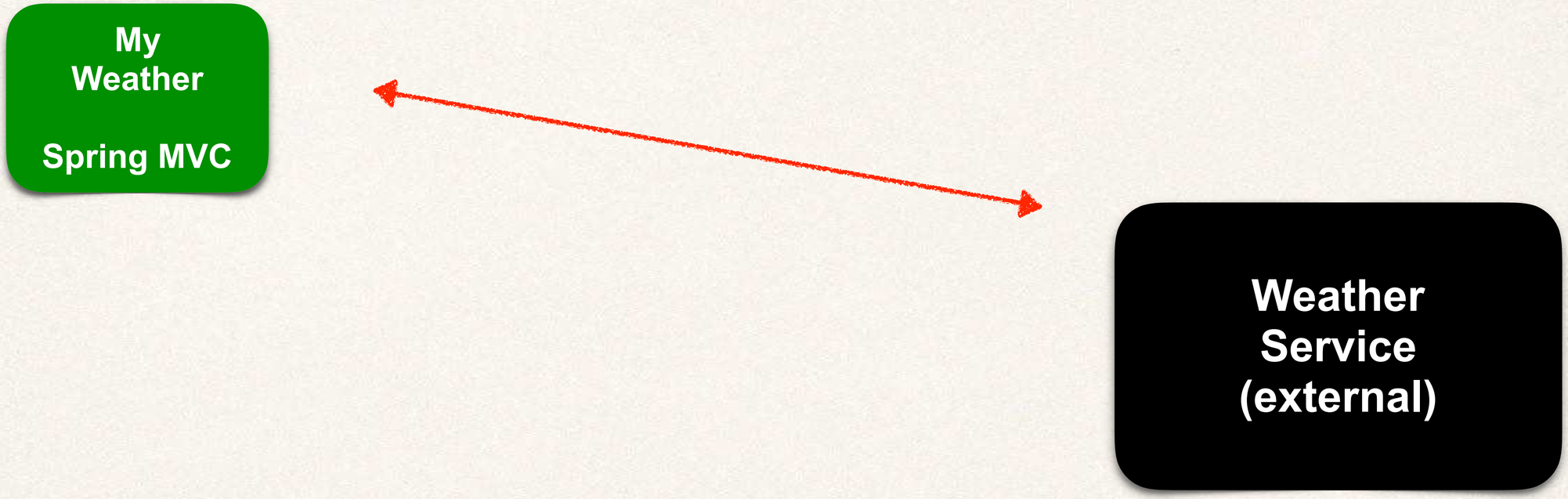
Condensed version

Multiple Client Apps

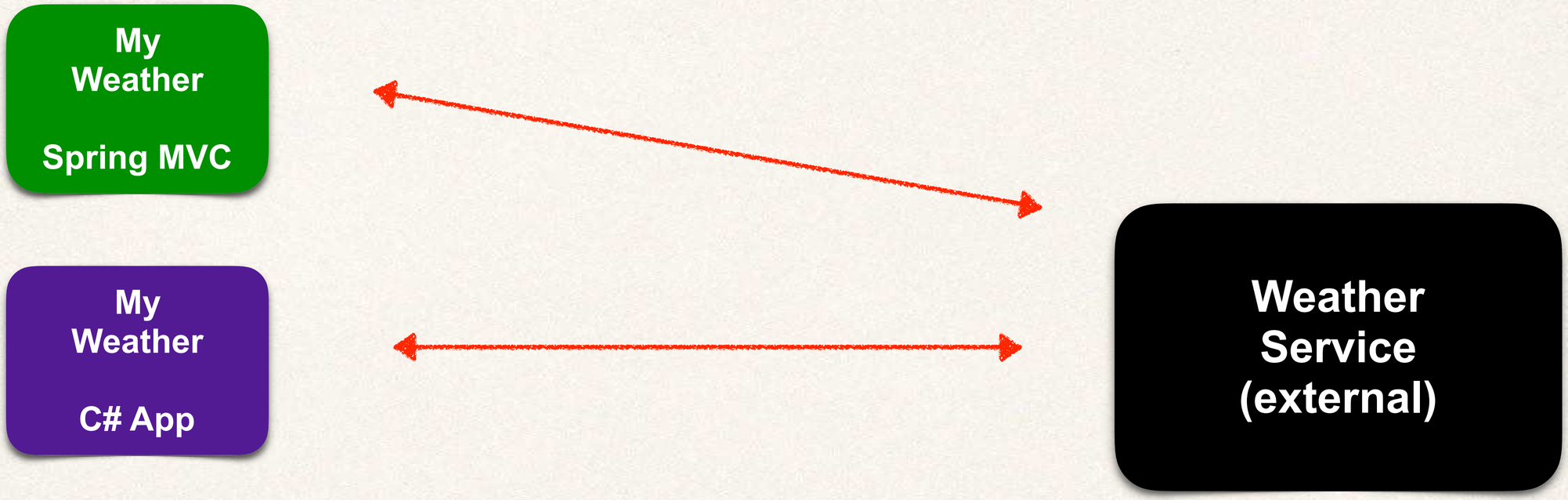
Multiple Client Apps

Weather
Service
(external)

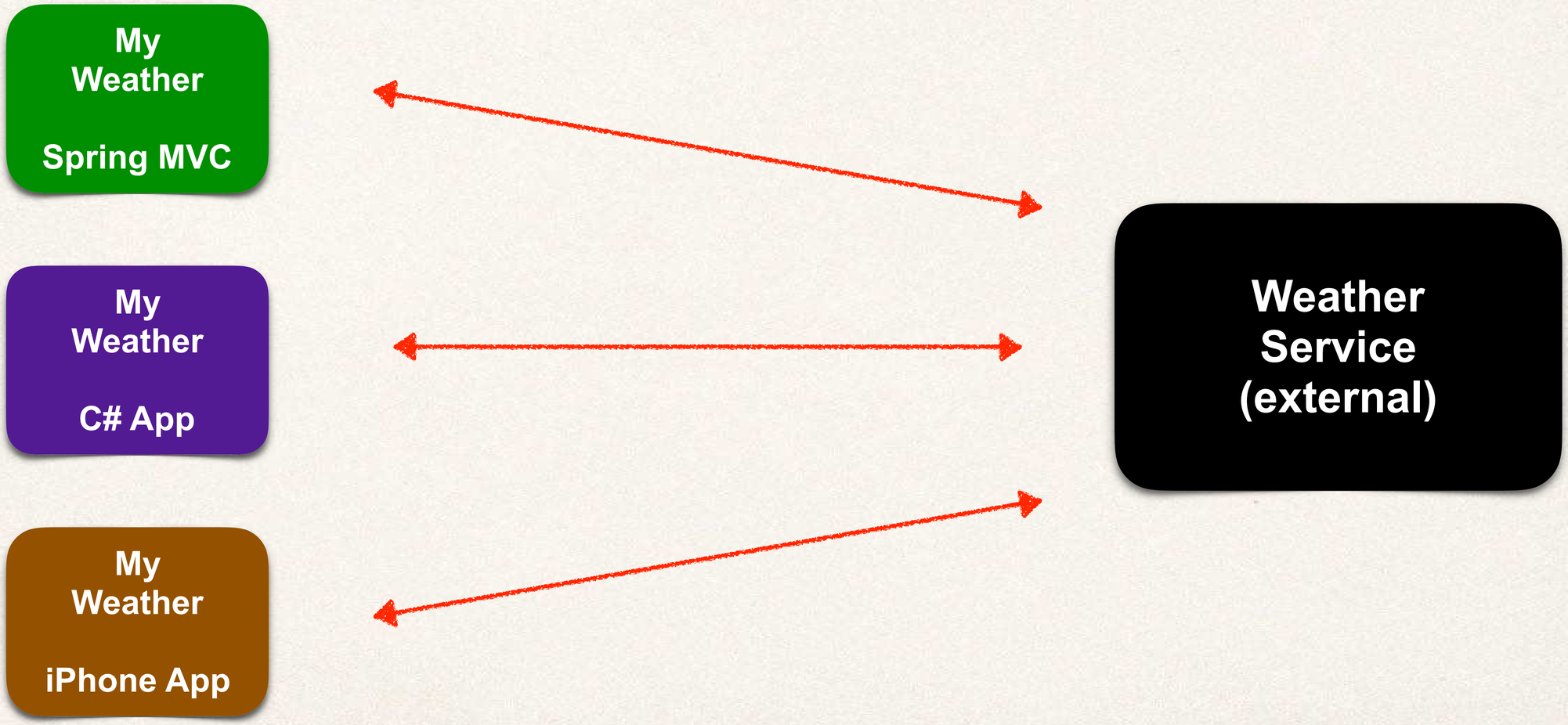
Multiple Client Apps



Multiple Client Apps

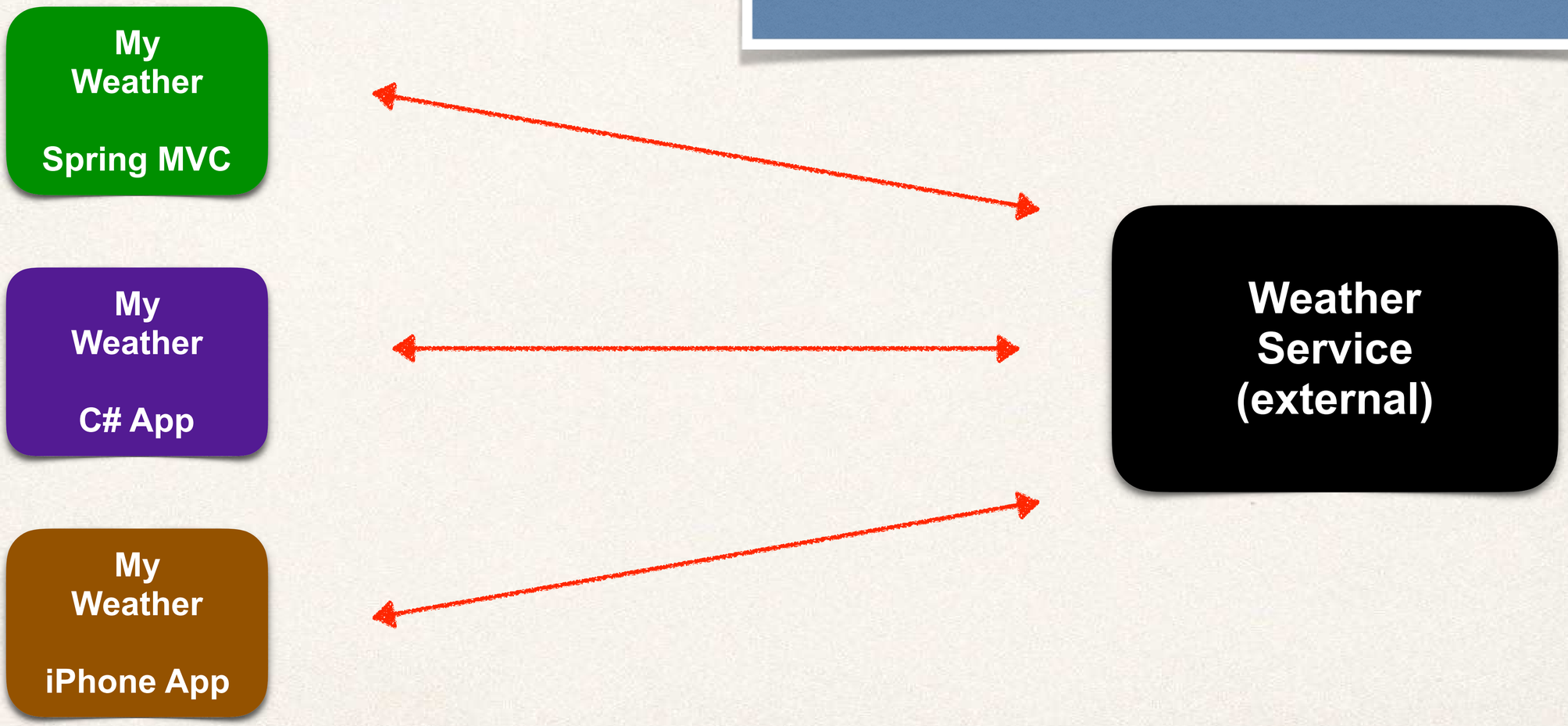


Multiple Client Apps



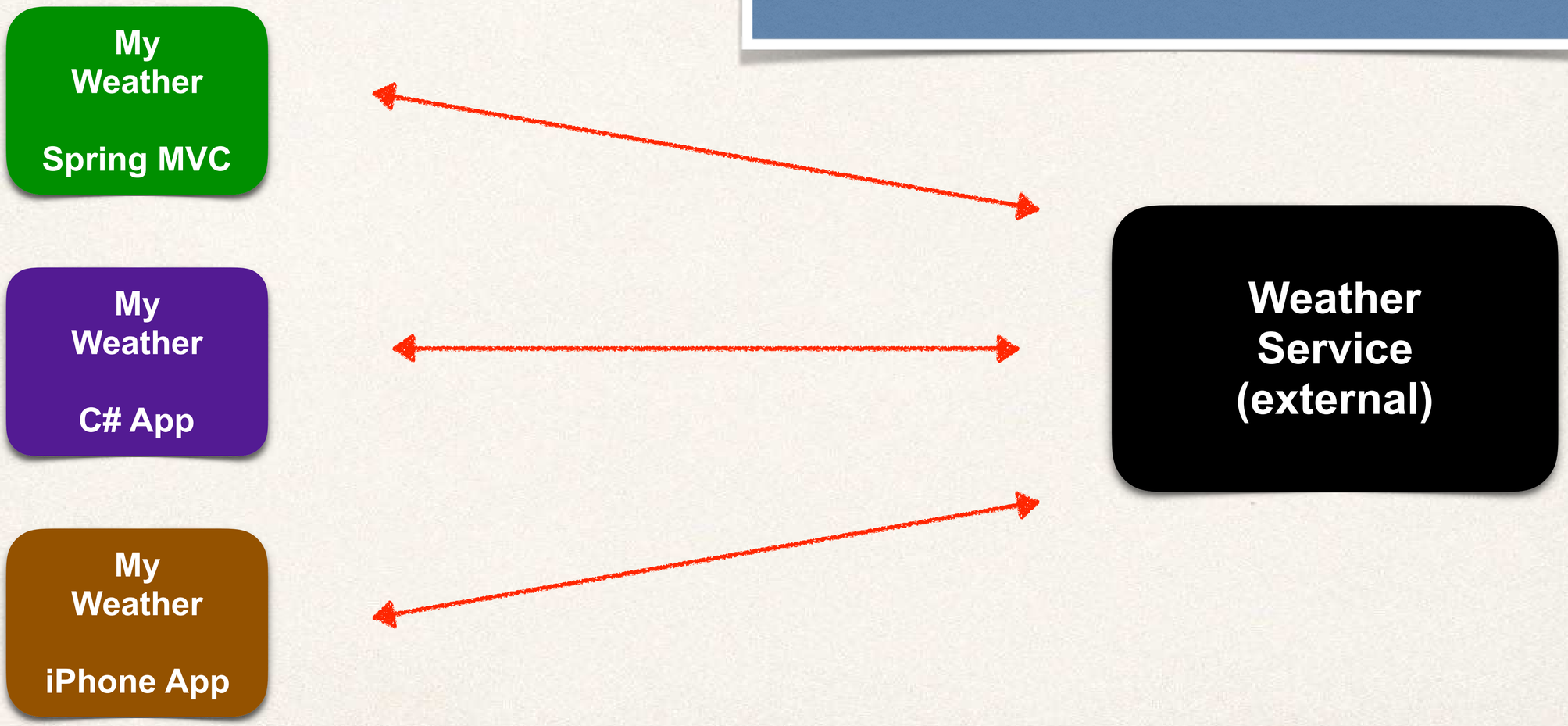
Multiple Client Apps

Remember:



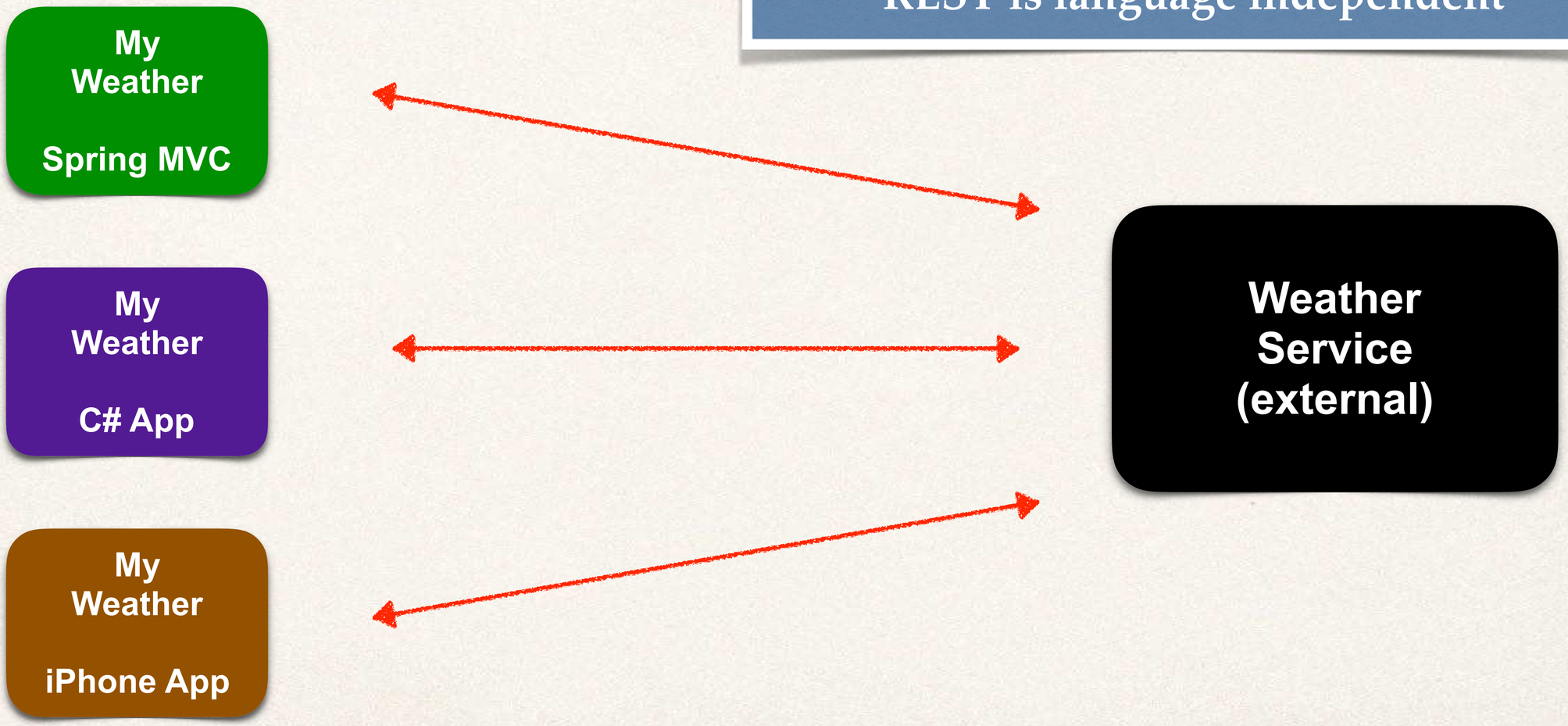
Multiple Client Apps

Remember:
REST calls can be made over HTTP



Multiple Client Apps

Remember:
REST calls can be made over HTTP
REST is language independent



Currency Converter App

Currency Converter App

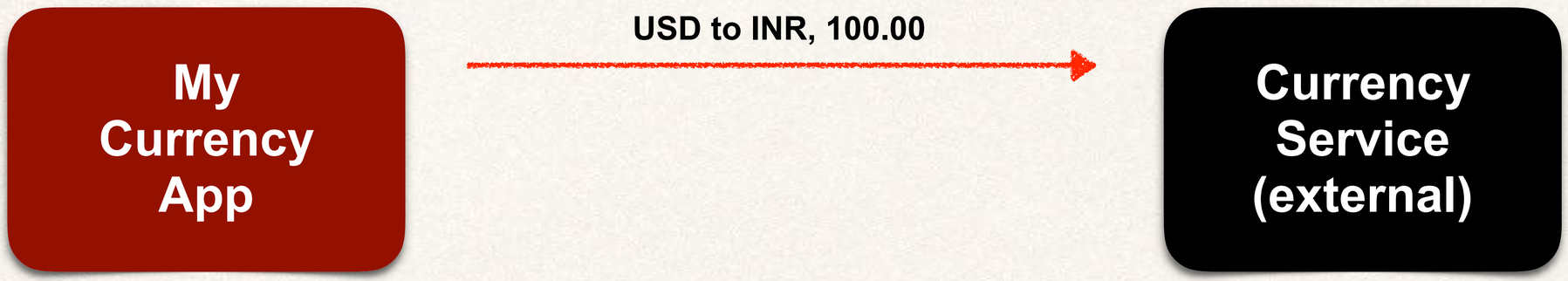
My
Currency
App

Currency Converter App

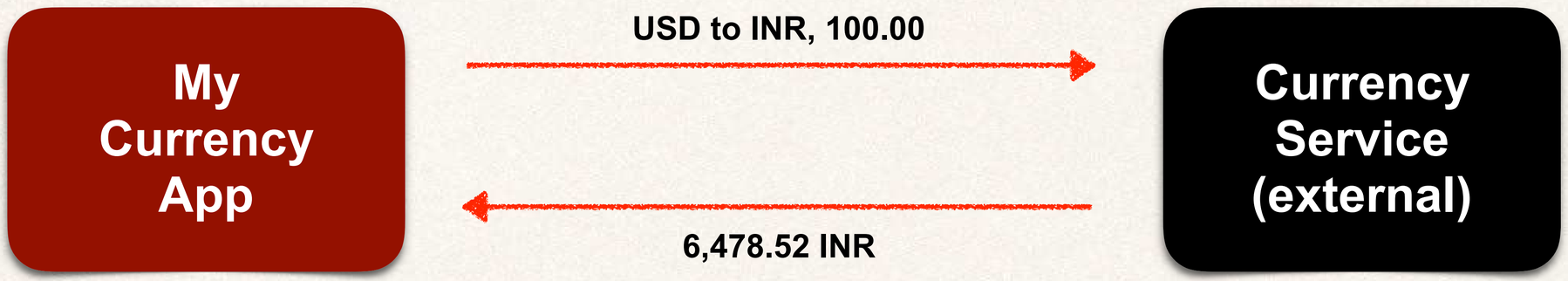
My Currency App

Currency Service (external)

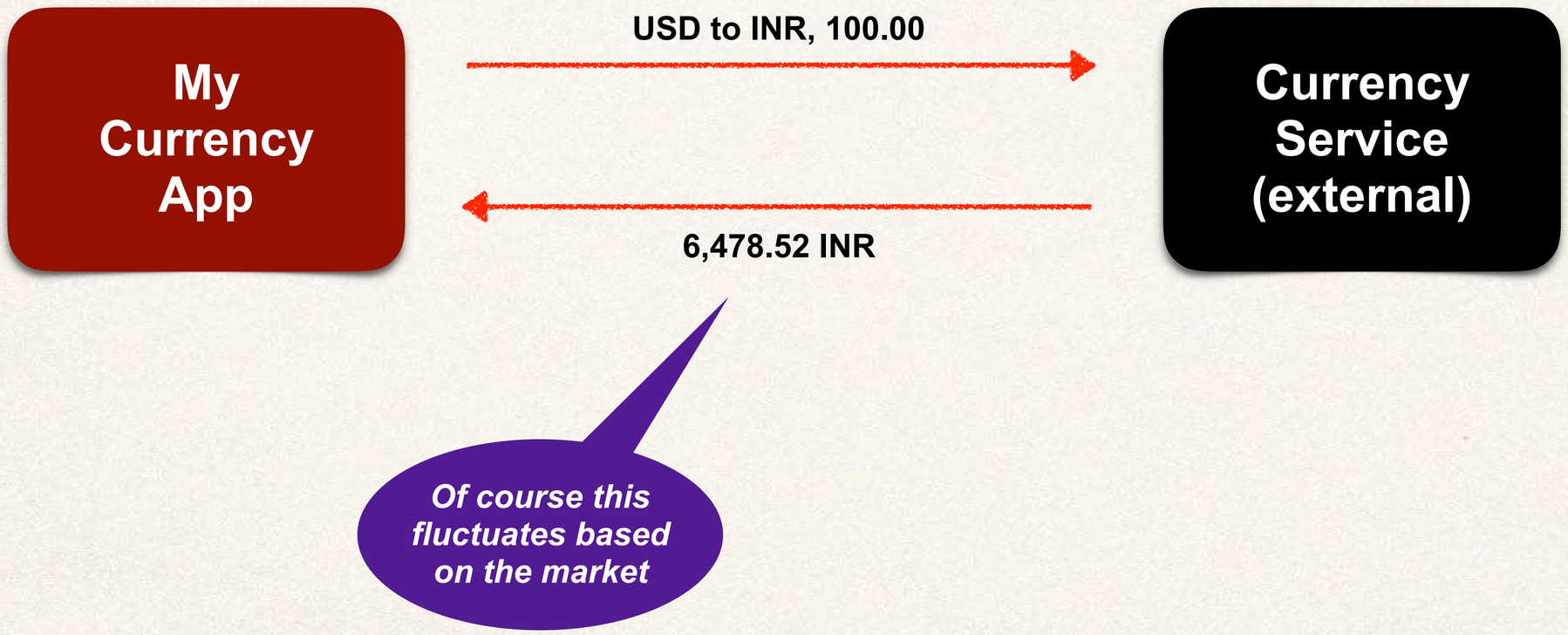
Currency Converter App



Currency Converter App



Currency Converter App



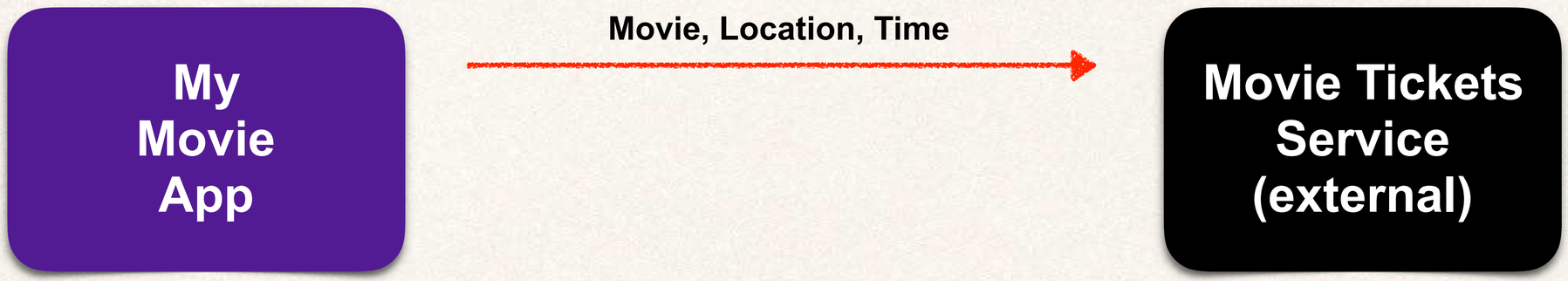
Movie Tickets App

Movie Tickets App

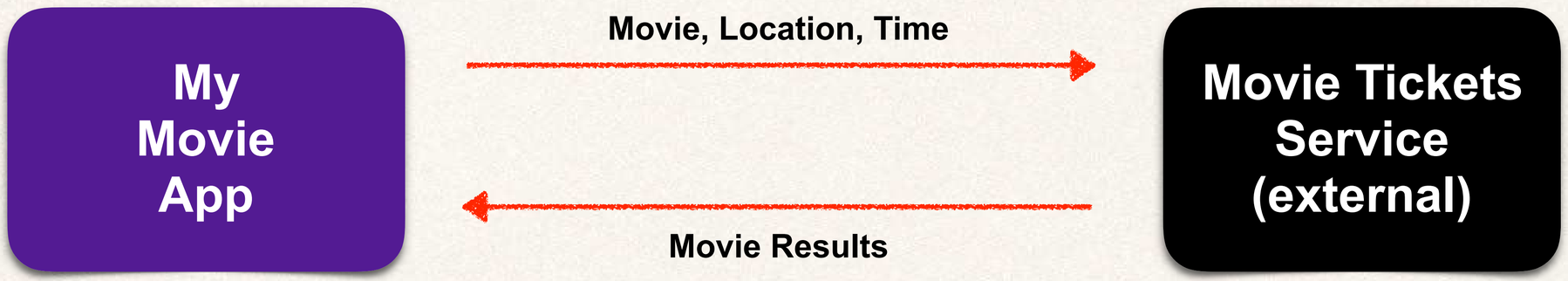
My
Movie
App

Movie Tickets App

Movie Tickets App



Movie Tickets App



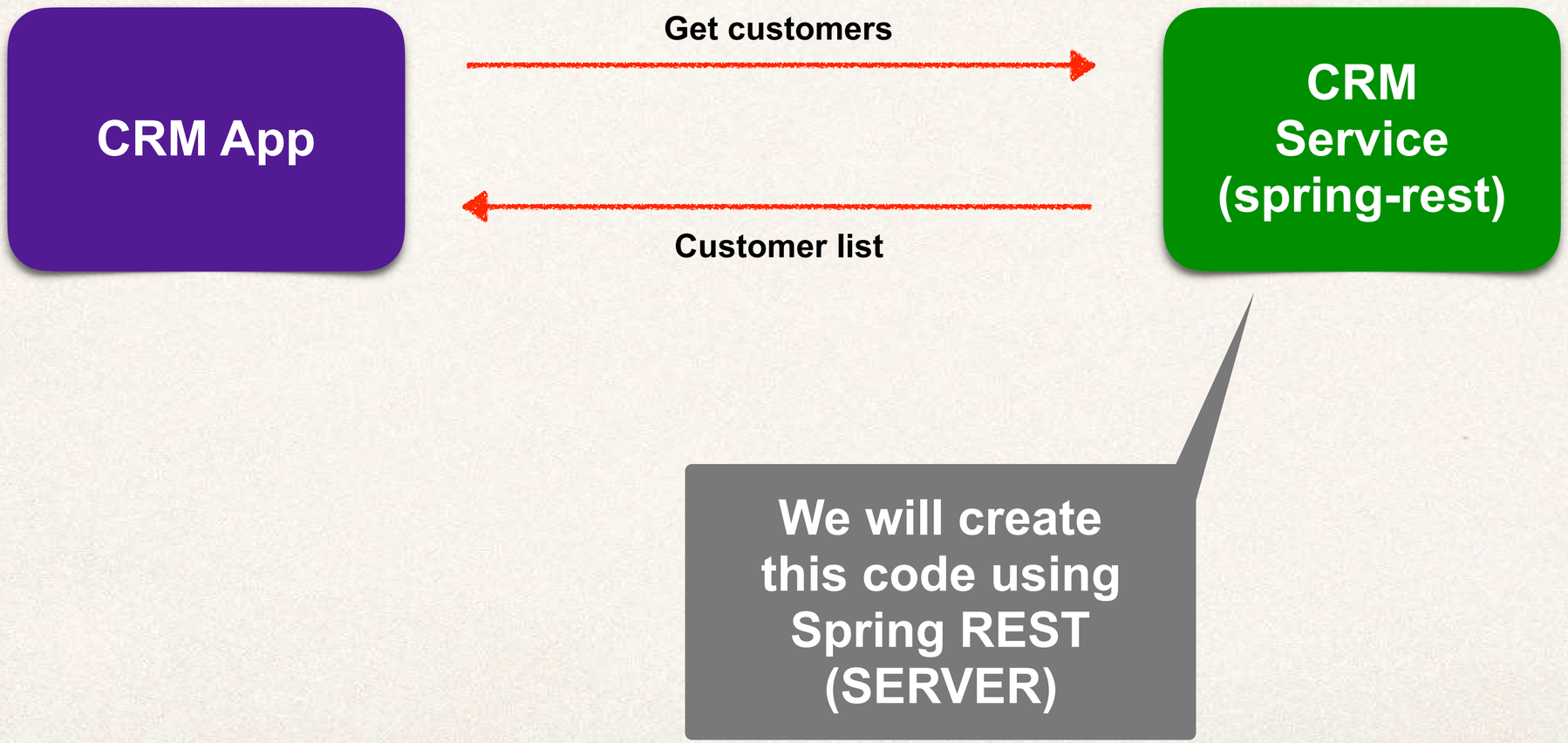
Customer Relationship Manager (CRM) App

Customer Relationship Manager (CRM) App

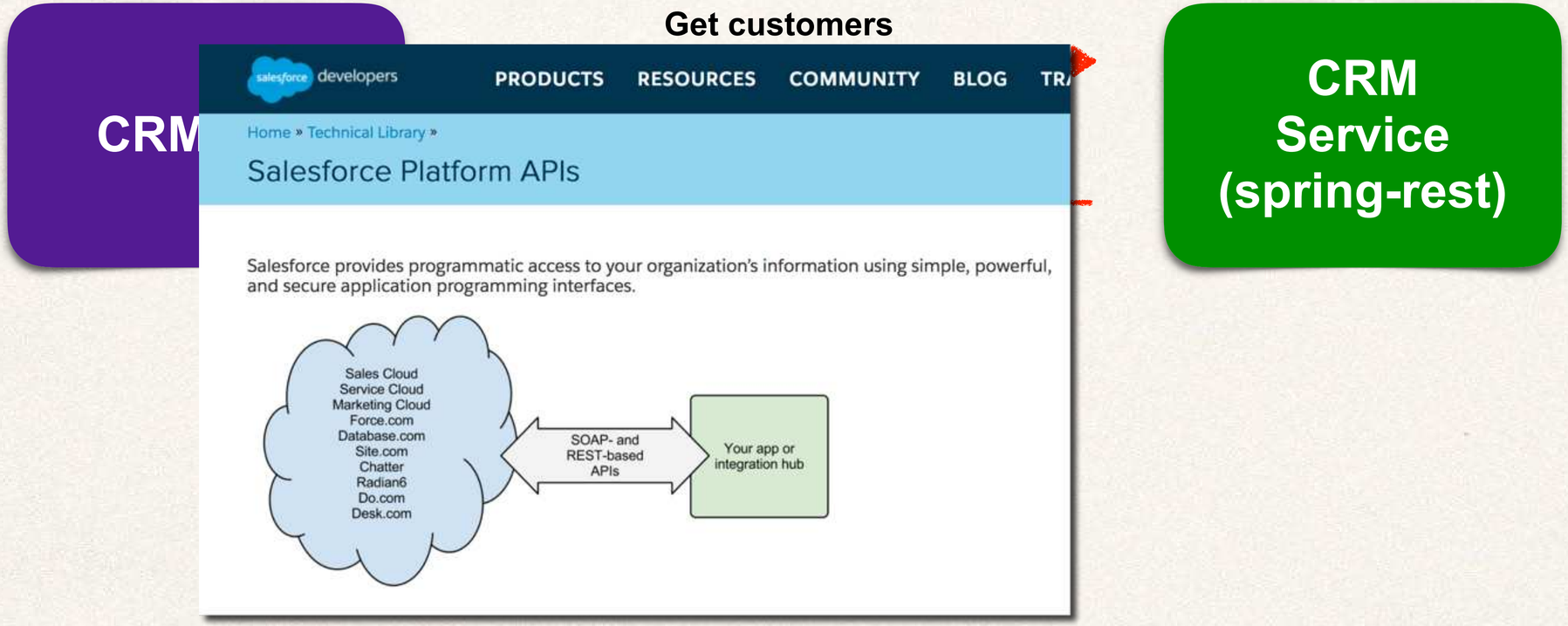
Customer Relationship Manager (CRM) App

Customer Relationship Manager (CRM) App

Customer Relationship Manager (CRM) App

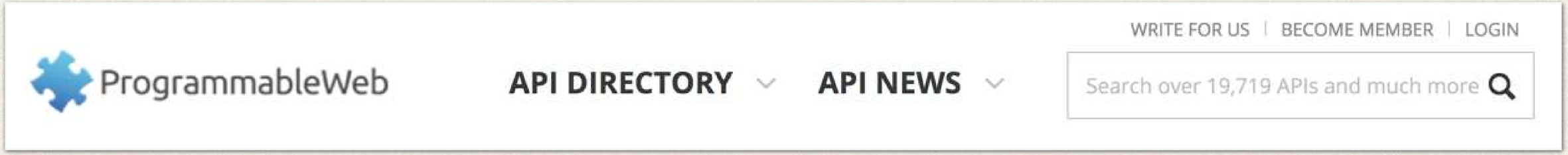


Customer Relationship Manager (CRM) App



Where to Find REST APIs

www.programmableweb.com



What do we call it?

What do we call it?

REST API

RESTful API

What do we call it?

REST API

REST
Web Services

RESTful API

RESTful
Web Services

What do we call it?

REST API

REST
Web Services

REST Services

RESTful API

RESTful
Web Services

RESTful Services

What do we call it?

REST API

REST
Web Services

REST Services

RESTful API

RESTful
Web Services

RESTful Services

Generally, all mean the SAME thing

What do we call it?

REST API

REST
Web Services

REST Services

RESTful API

RESTful
Web Services

RESTful Services

Generally, all mean the SAME thing

JSON Basics - Syntax

What is JSON?

What is JSON?

- **J**ava**S**cript **O**bject **N**otation

What is JSON?

- **J**ava**S**cript **O**bject **N**otation

- Lightweight data format for storing and exchanging data … plain text

What is JSON?

- **J**ava**S**cript **O**bject **N**otation
- Lightweight data format for storing and exchanging data … plain text
- Language independent … not just for **JavaScript**

What is JSON?

- **J**ava**S**cript **O**bject **N**otation
- Lightweight data format for storing and exchanging data … plain text
- Language independent … not just for **JavaScript**
- Can use with any programming language: **Java, C# , Python** etc …

What is JSON?

- **J**ava**S**cript **O**bject **N**otation
- Lightweight data format for storing and exchanging data … plain text
- Language independent … not just for **JavaScript**
- Can use with any programming language: **Java, C#, Python** etc …

JSON is just plain text data

Simple JSON Example

Simple JSON Example

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

Simple JSON Example

- Curley braces define objects in JSON

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

Simple JSON Example

- Curley braces define objects in JSON
- Object members are name / value pairs

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

Simple JSON Example

- Curley braces define objects in JSON
- Object members are name / value pairs
  - Delimited by colons

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

Simple JSON Example

- Curley braces define objects in JSON
- Object members are name / value pairs
  - Delimited by colons

Name

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

Simple JSON Example

- Curley braces define objects in JSON
- Object members are name / value pairs
  - Delimited by colons
- Name is **always** in double-quotes

Name

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

JSON Values

JSON Values

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

- Numbers: no quotes

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

- Numbers: no quotes
- String: in double quotes

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

- Numbers: no quotes
- String: in double quotes
- Boolean: **`true`**, **`false`**

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

- Numbers: no quotes
- String: in double quotes
- Boolean: **`true`**, **`false`**
- Nested JSON object

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

- Numbers: no quotes
- String: in double quotes
- Boolean: **`true`**, **`false`**
- Nested JSON object
- Array

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

JSON Values

- Numbers: no quotes
- String: in double quotes
- Boolean: **`true`**, **`false`**
- Nested JSON object
- Array
- **`null`**

Value

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "courses" : null
}
```

Nested JSON Objects

Nested JSON Objects

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "address" : {
                "street" : "100 Main St",
                "city" : "Philadelphia",
                "state" : "Pennsylvania",
                "zip" : "19103",
                "country" : "USA"
              }
}
```

Nested JSON Objects

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "address" : {
                "street" : "100 Main St",
                "city" : "Philadelphia",
                "state" : "Pennsylvania",
                "zip" : "19103",
                "country" : "USA"
              }
}
```

Nested

JSON Arrays

JSON Arrays

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "languages" : ["Java", "C#", "Python", "Javascript"]
}
```

JSON Arrays

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "languages" : ["Java", "C#", "Python", "Javascript"]
}
```

Array

Java JSON Data Binding

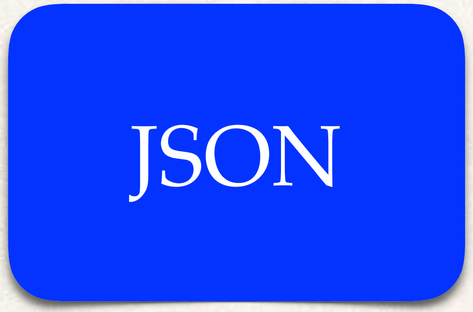
Java JSON Data Binding

Java JSON Data Binding

- Data binding is the process of converting JSON data to a Java POJO

Java JSON Data Binding

- Data binding is the process of converting JSON data to a Java POJO



Java JSON Data Binding

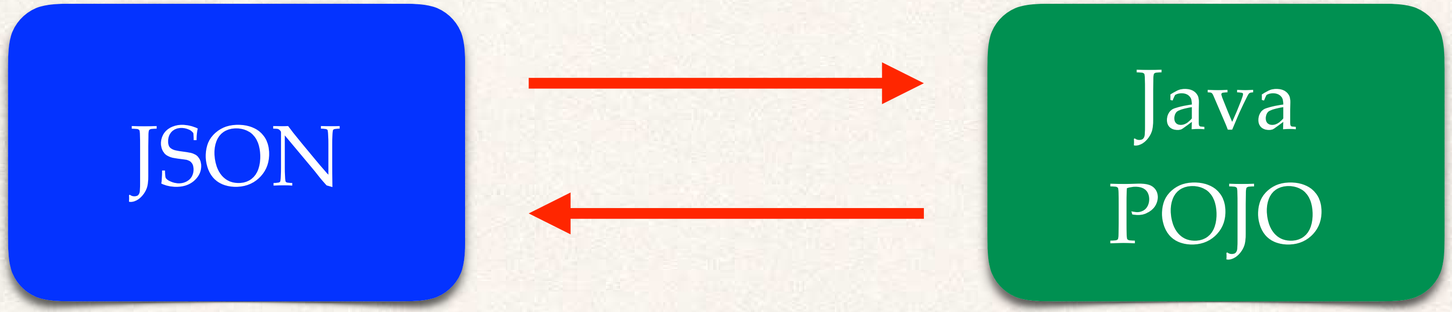
- Data binding is the process of converting JSON data to a Java POJO

Java JSON Data Binding

- Data binding is the process of converting JSON data to a Java POJO

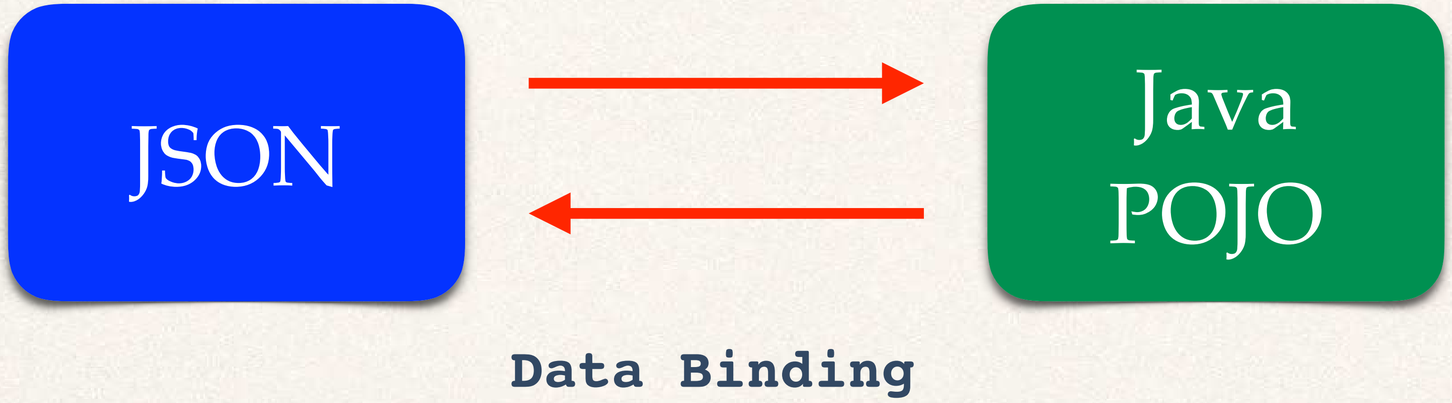
Java JSON Data Binding

- Data binding is the process of converting JSON data to a Java POJO



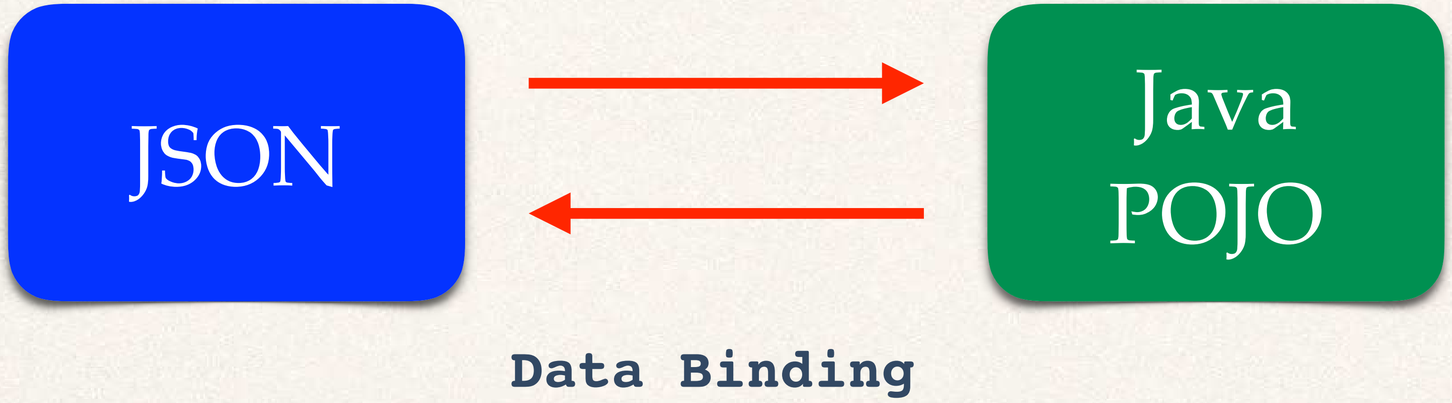
Java JSON Data Binding

- Data binding is the process of converting JSON data to a Java POJO



Java JSON Data Binding

- Data binding is the process of converting JSON data to a Java POJO



Also known as

Mapping

Serialization / Deserialization

Marshalling / Unmarshalling

JSON Data Binding with Jackson

JSON Data Binding with Jackson

- Spring uses the **Jackson Project** behind the scenes

JSON Data Binding with Jackson

- Spring uses the **Jackson Project** behind the scenes

- Jackson handles data binding between JSON and Java POJO

JSON Data Binding with Jackson

- Spring uses the **Jackson Project** behind the scenes
- Jackson handles data binding between JSON and Java POJO
- Details on Jackson Project:

JSON Data Binding with Jackson

- Spring uses the **Jackson Project** behind the scenes
- Jackson handles data binding between JSON and Java POJO
- Details on Jackson Project:

https://github.com/FasterXML/jackson-databind

Jackson Data Binding

Jackson Data Binding

- Jackson Data Binding API

Jackson Data Binding

- Jackson Data Binding API
  - Package: **`com.fasterxml.jackson.databind`**

Jackson Data Binding

- Jackson Data Binding API
  - Package: **`com.fasterxml.jackson.databind`**

- Maven Dependency

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.0</version>
</dependency>
```

Jackson Data Binding

Jackson Data Binding

- By default, Jackson will call appropriate getter/setter method

Jackson Data Binding

- By default, Jackson will call appropriate getter/setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Jackson Data Binding

- By default, Jackson will call appropriate getter/setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

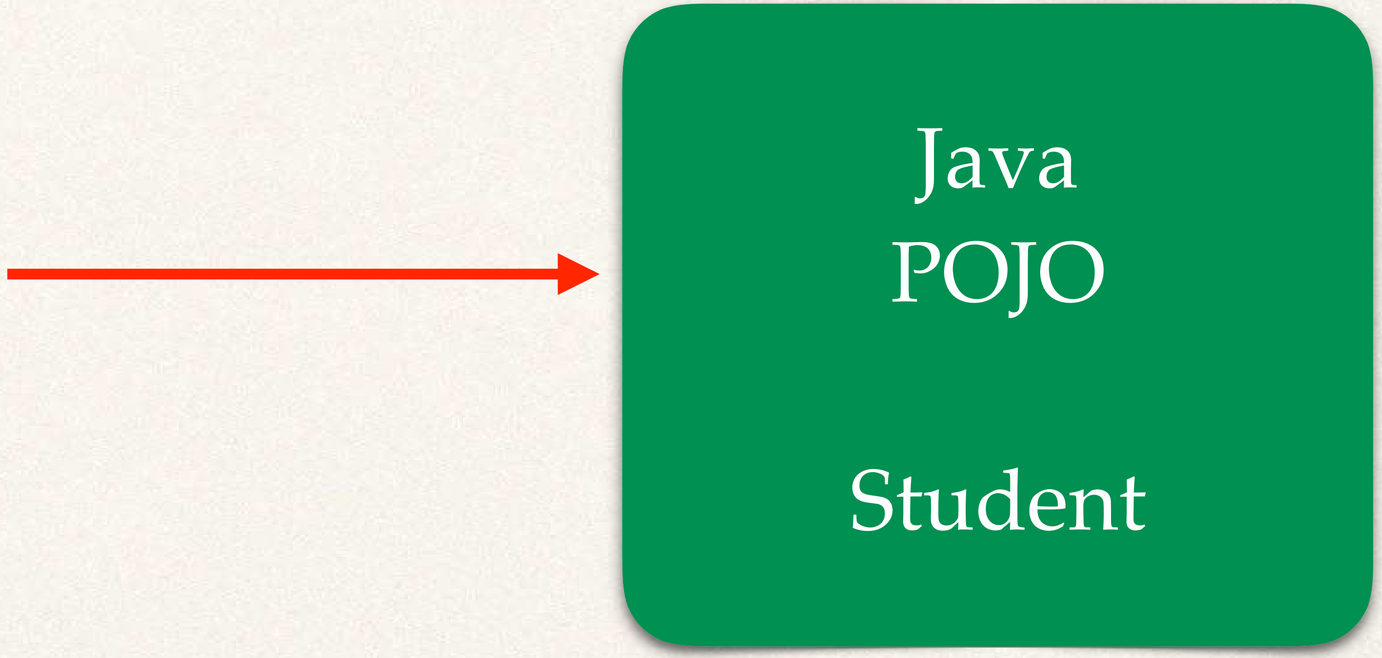
Java
POJO

Student

Jackson Data Binding

- By default, Jackson will call appropriate getter/setter method

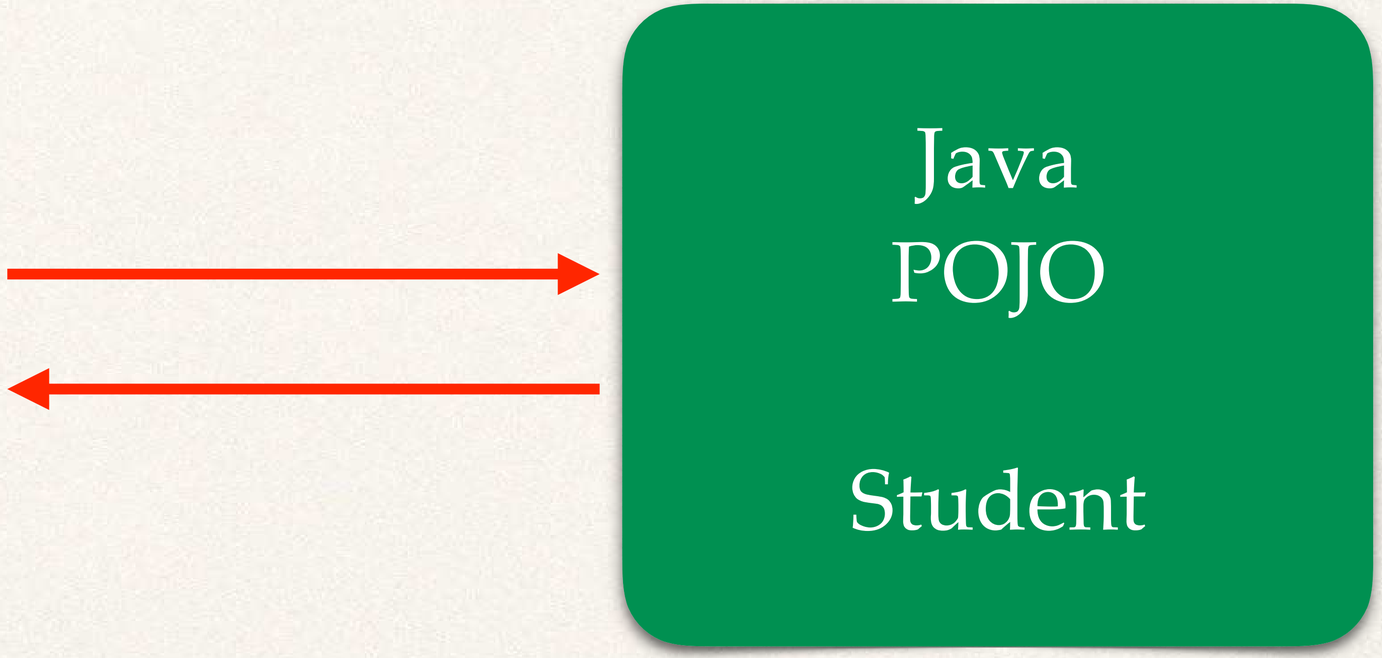
```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```



Jackson Data Binding

- By default, Jackson will call appropriate getter/setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```



JSON to Java POJO

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

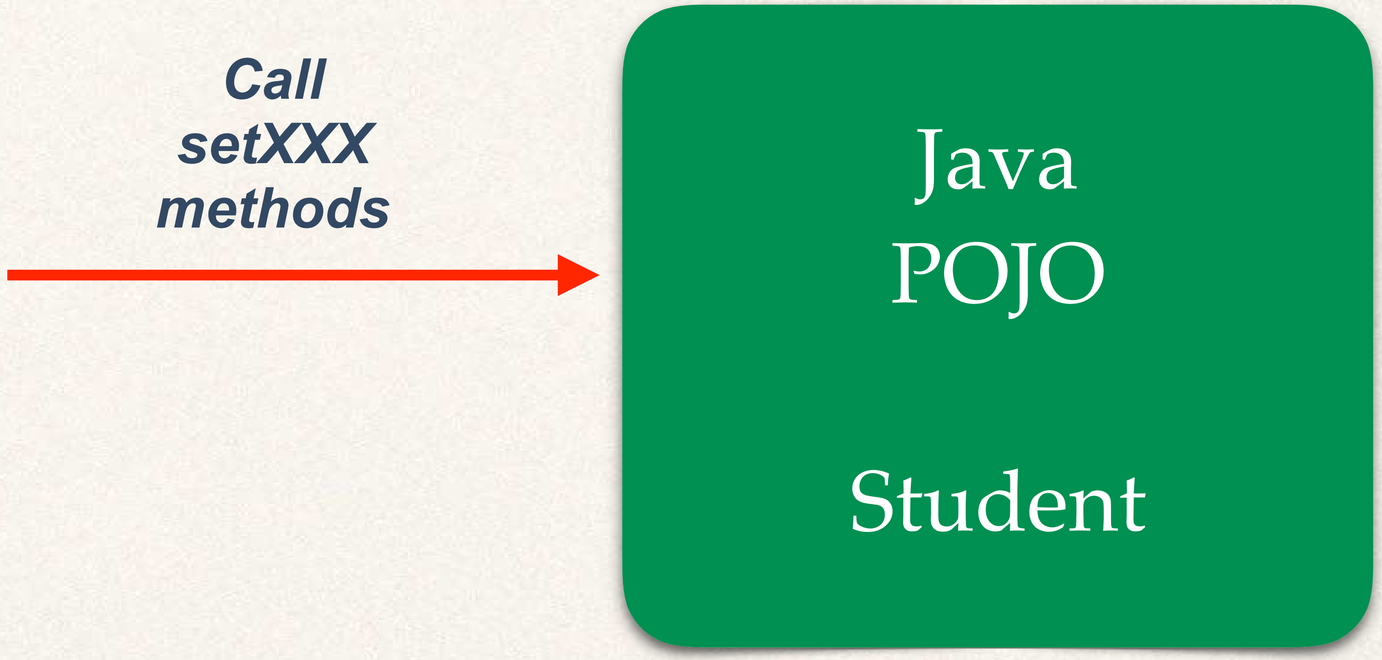
Java
POJO

Student

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

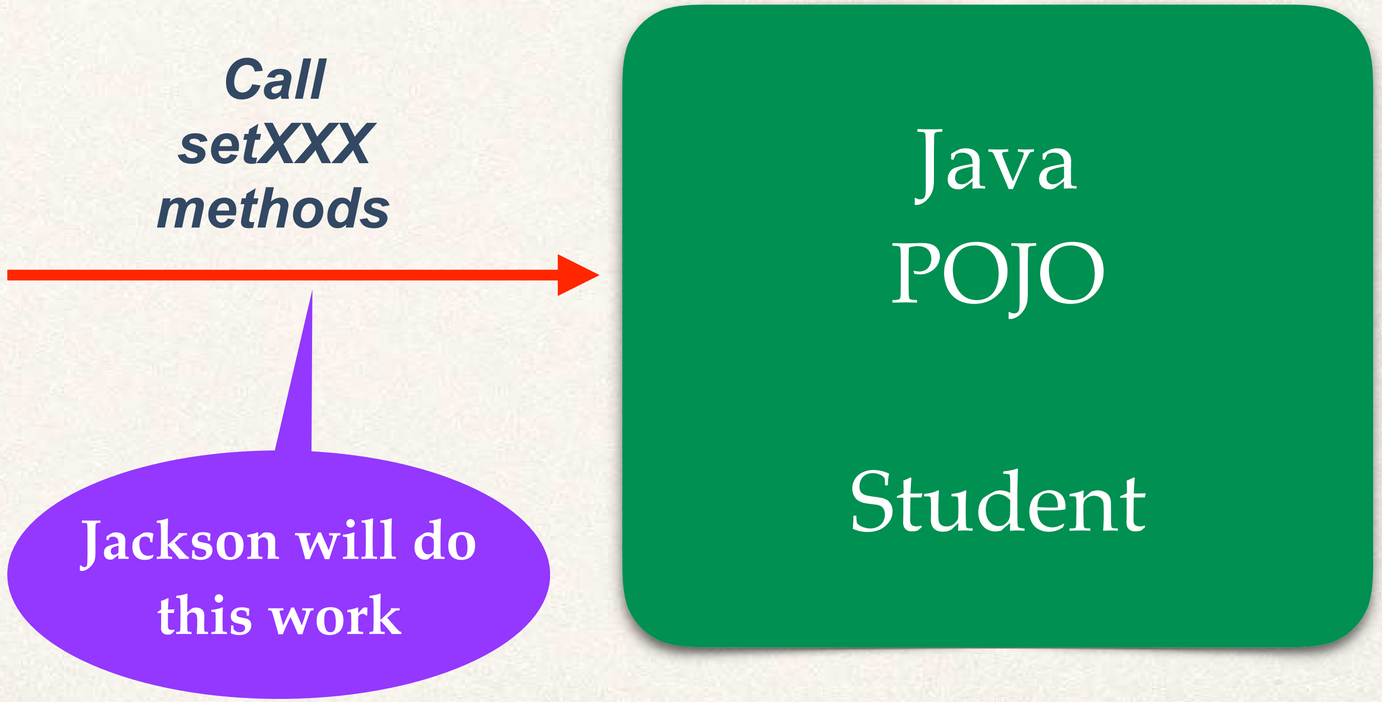
```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```



JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```



JSON to Java POJO

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Call setId(…)

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Call setId(…)

Jackson will do this work

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Call setId(…)

Call setFirstName(…)

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Call setId(…)

Call setFirstName(…)

Call setLastName(…)

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Call setId(…)

Call setFirstName(…)

Call setLastName(…)

Call setActive(…)

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

Note: Jackson calls the setXXX methods
It does NOT access internal private fields directly

- Convert JSON to Java POJO … call setter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Call setId(…)

Call setFirstName(…)

Call setLastName(…)

Call setActive(…)

```
public class Student {

    private int id;
    private String firstName;
    private String lastName;
    private boolean active;

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

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    // getter methods
}
```

JSON to Java POJO

JSON to Java POJO

```
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;
```

JSON to Java POJO

```
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {
```

JSON to Java POJO

```
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();
```

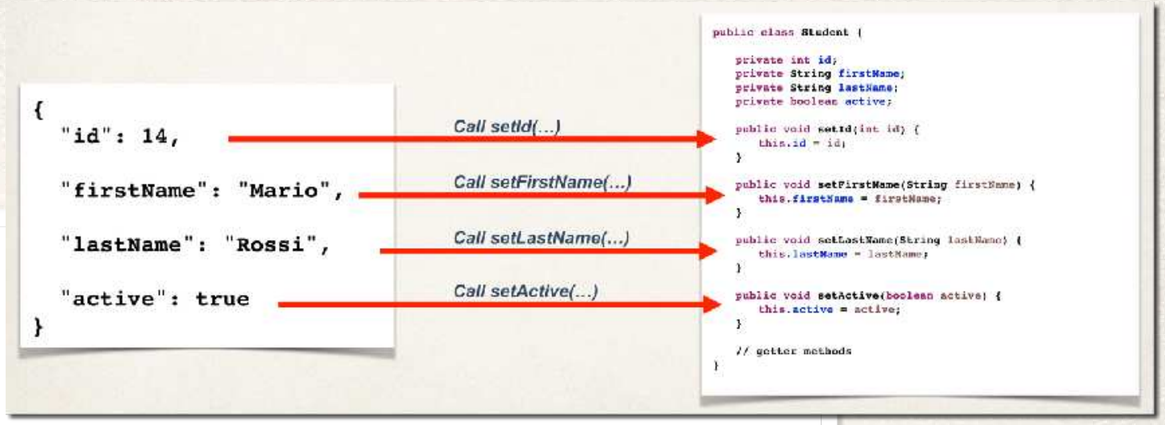
JSON to Java POJO

```
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();
```

JSON to Java POJO

```
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
```

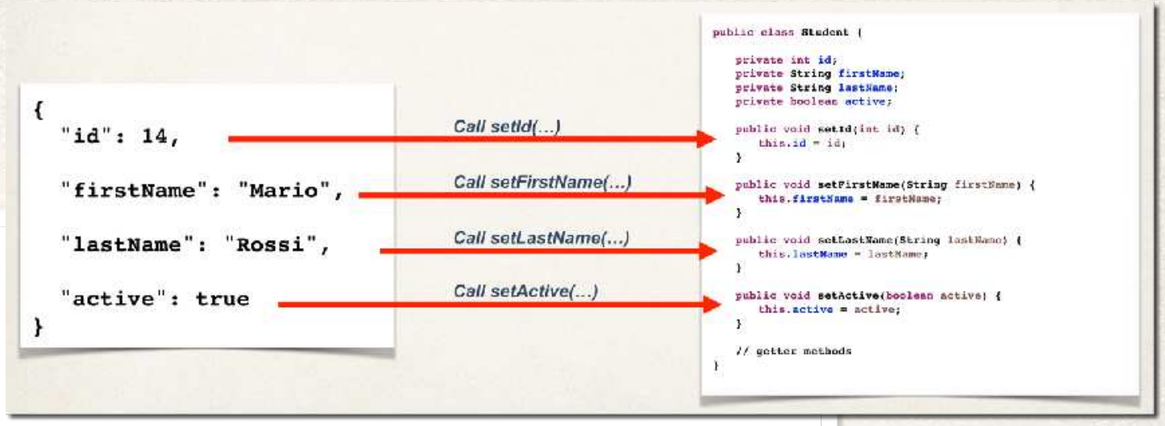
JSON to Java POJO



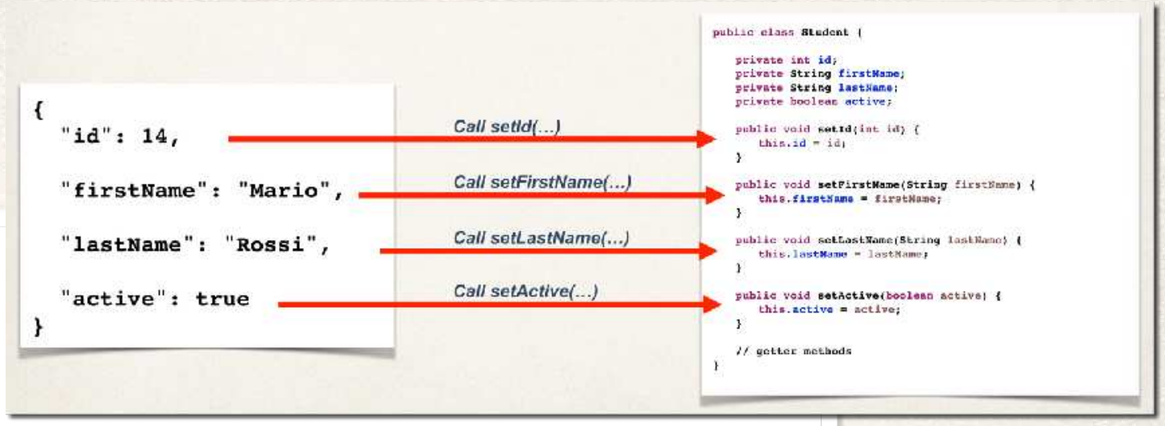

```java
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
```

JSON to Java POJO

```
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
```

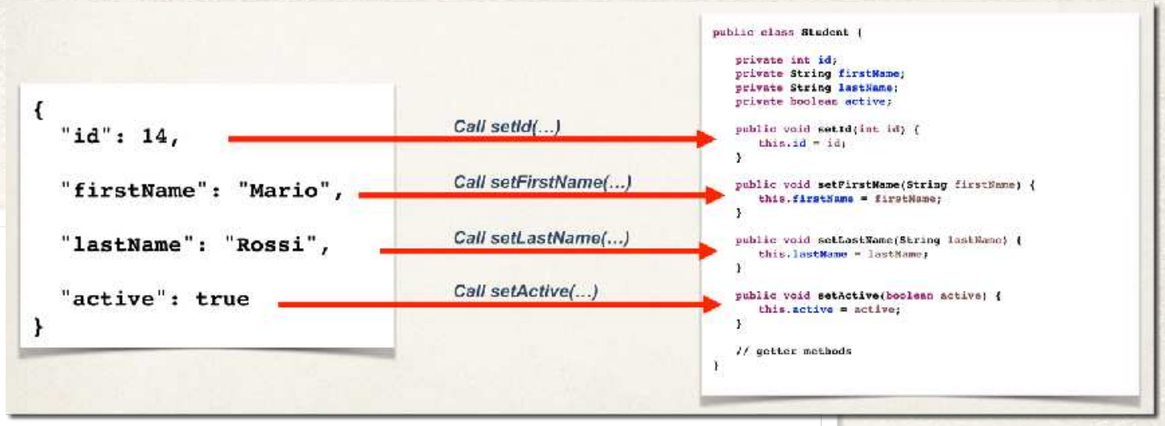


JSON to Java POJO




```java
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
```

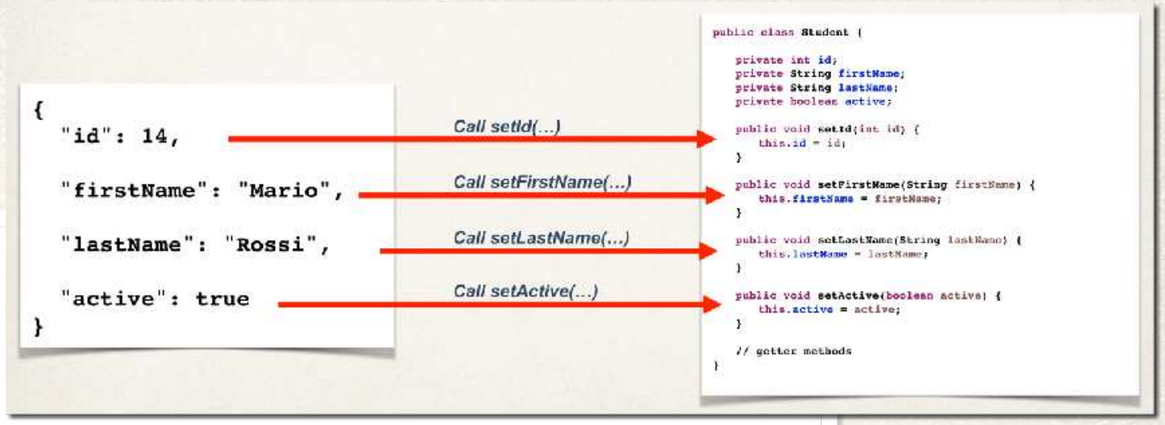
JSON to Java POJO




```java
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
```

1. Read data from this file

JSON to Java POJO

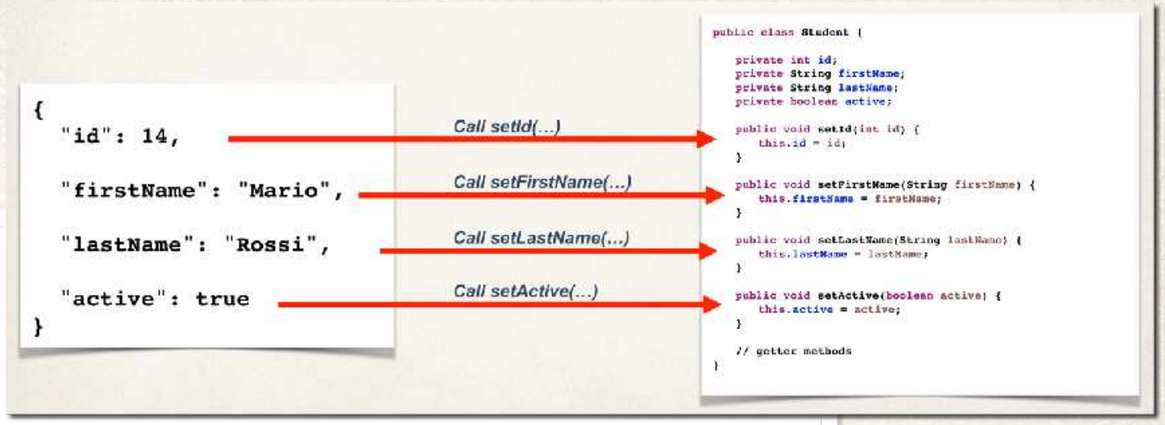



```java
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
```

1. Read data from this file

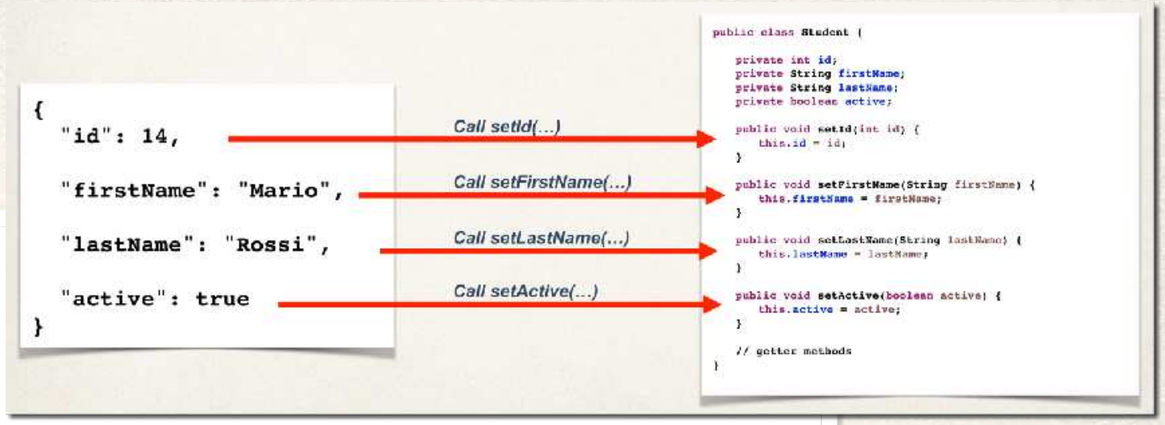
2. Create an instance of this class and populate it

JSON to Java POJO




```java
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);

      // also print individual items
      System.out.println("First name = " + myStudent.getFirstName());
      System.out.println("Last name = " + myStudent.getLastName());

  }
}
```

JSON to Java POJO




```java
import java.io.File;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Driver {

  public static void main(String[] args) throws Exception {

      // create object mapper
      ObjectMapper mapper = new ObjectMapper();

      // read JSON from file and map/convert to Java POJO
      Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);

      // also print individual items
      System.out.println("First name = " + myStudent.getFirstName());
      System.out.println("Last name = " + myStudent.getLastName());

  }
}
```

Output

```
First name = Mario
Last name = Rossi
```

Java POJO to JSON

Java POJO to JSON

- Now, let's go the other direction
- Convert Java POJO to JSON … call getter methods on POJO

Java POJO to JSON

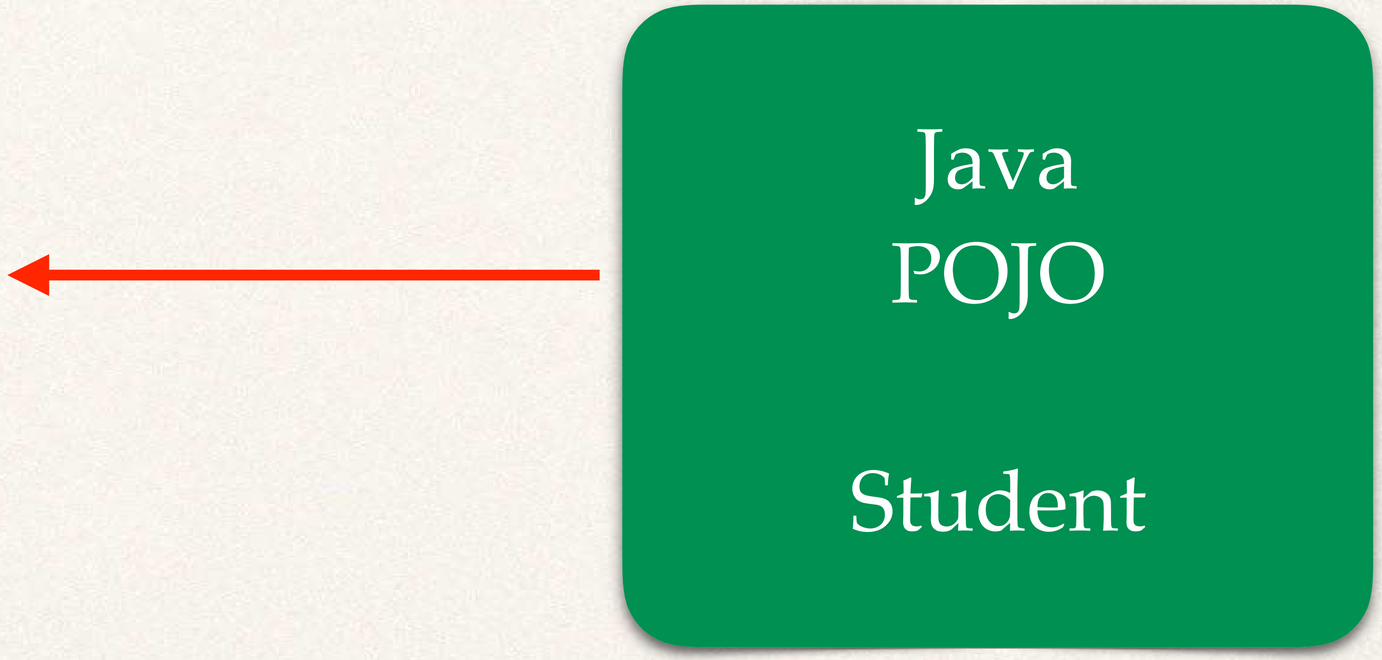
- Now, let's go the other direction
- Convert Java POJO to JSON … call getter methods on POJO

Java
POJO

Student

Java POJO to JSON

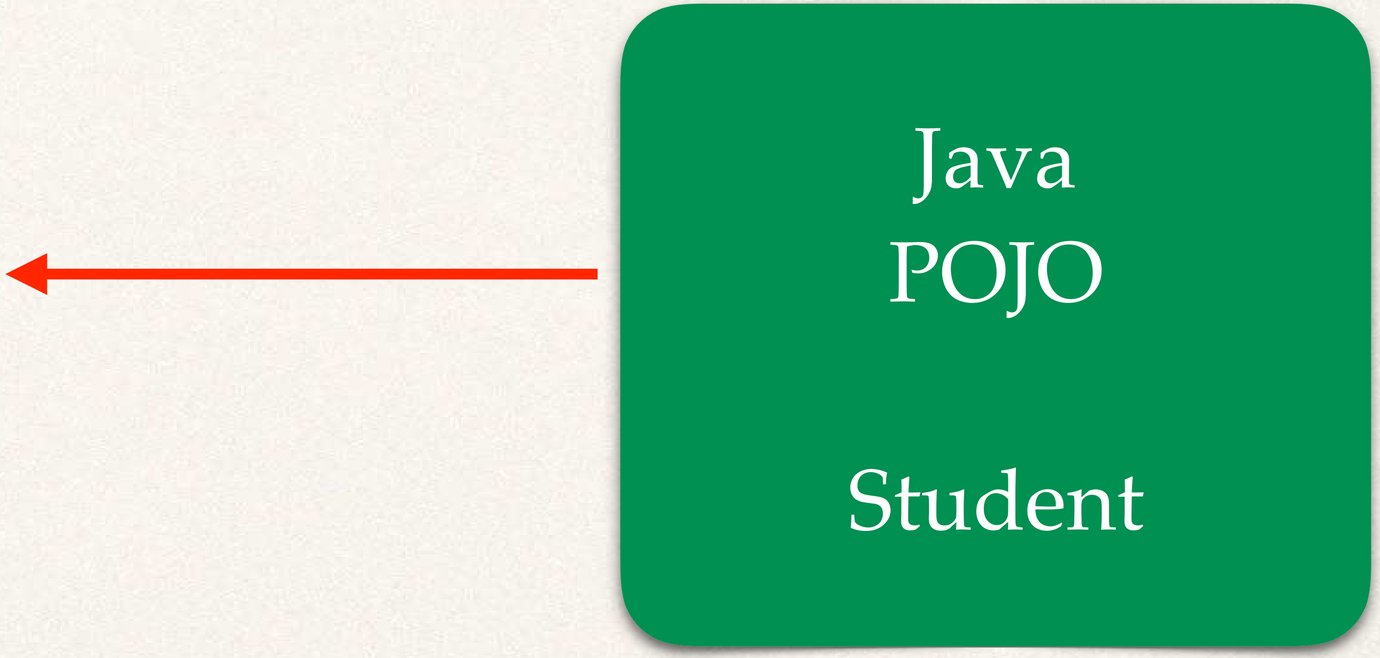
- Now, let's go the other direction
- Convert Java POJO to JSON … call getter methods on POJO



Java POJO to JSON

- Now, let's go the other direction
- Convert Java POJO to JSON … call getter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```



Java POJO to JSON

- Now, let's go the other direction
- Convert Java POJO to JSON … call getter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Java POJO to JSON

- Now, let's go the other direction
- Convert Java POJO to JSON … call getter methods on POJO

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Jackson will do this work

Call getXXX methods

Java POJO

Student

Java POJO to JSON

Java POJO to JSON

```
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
...
```

Java POJO to JSON

```
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
...

// now write JSON to output file
mapper.enable(SerializationFeature.INDENT_OUTPUT);
```

Java POJO to JSON

```
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample
...

// now write JSON to output file
mapper.enable(SerializationFeature.INDENT_OUTPUT);
```

Indent the JSON output for "pretty printing"

Java POJO to JSON

```
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
...

// now write JSON to output file
mapper.enable(SerializationFeature.INDENT_OUTPUT);
mapper.writeValue(new File("data/output.json"), myStudent);
```

Java POJO to JSON

```
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
...

// now write JSON to output file
mapper.enable(SerializationFeature.INDENT_OUTPUT);
mapper.writeValue(new File("data/output.json"), myStudent);
```

Jackson calls the getter methods on Student POJO
to create JSON output file

Java POJO to JSON

```
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
...

// now write JSON to output file
mapper.enable(SerializationFeature.INDENT_OUTPUT);
mapper.writeValue(new File("data/output.json"), myStudent);
```

Jackson calls the getter methods on Student POJO to create JSON output file

File: data/output.json

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

Spring and Jackson Support

Spring and Jackson Support

- When building Spring REST applications

Spring and Jackson Support

- When building Spring REST applications
- Spring will automatically handle Jackson Integration

Spring and Jackson Support

- When building Spring REST applications
- Spring will automatically handle Jackson Integration
- JSON data being passed to REST controller is converted to POJO

Spring and Jackson Support

- When building Spring REST applications
- Spring will automatically handle Jackson Integration
- JSON data being passed to REST controller is converted to POJO
- Java object being returned from REST controller is converted to JSON

Spring and Jackson Support

- When building Spring REST applications
- Spring will automatically handle Jackson Integration
- JSON data being passed to REST controller is converted to POJO
- Java object being returned from REST controller is converted to JSON

Happens automatically behind the scenes

REST HTTP Basics

REST over HTTP

- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

REST over HTTP

- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

HTTP Method	CRUD Operation

REST over HTTP

- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

HTTP Method	CRUD Operation
POST	Create a new entity

REST over HTTP

- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

HTTP Method	CRUD Operation
POST	Create a new entity
GET	Read a list of entities or single entity

REST over HTTP

- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

HTTP Method	CRUD Operation
POST	Create a new entity
GET	Read a list of entities or single entity
PUT	Update an existing entity

REST over HTTP

- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

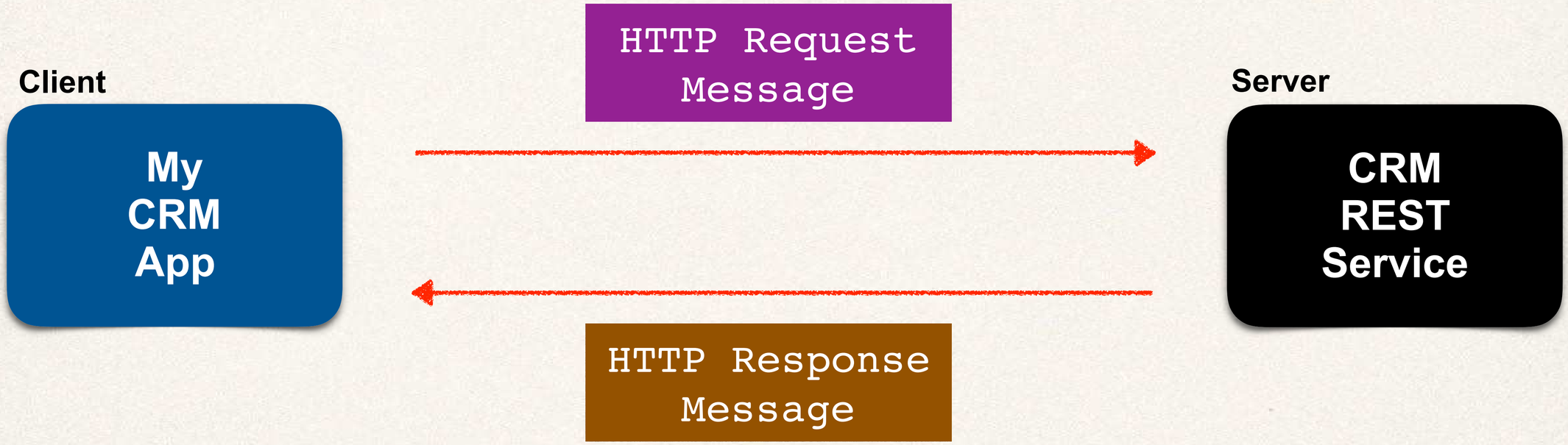
HTTP Method	CRUD Operation
POST	Create a new entity
GET	Read a list of entities or single entity
PUT	Update an existing entity
DELETE	Delete an existing entity

HTTP Messages

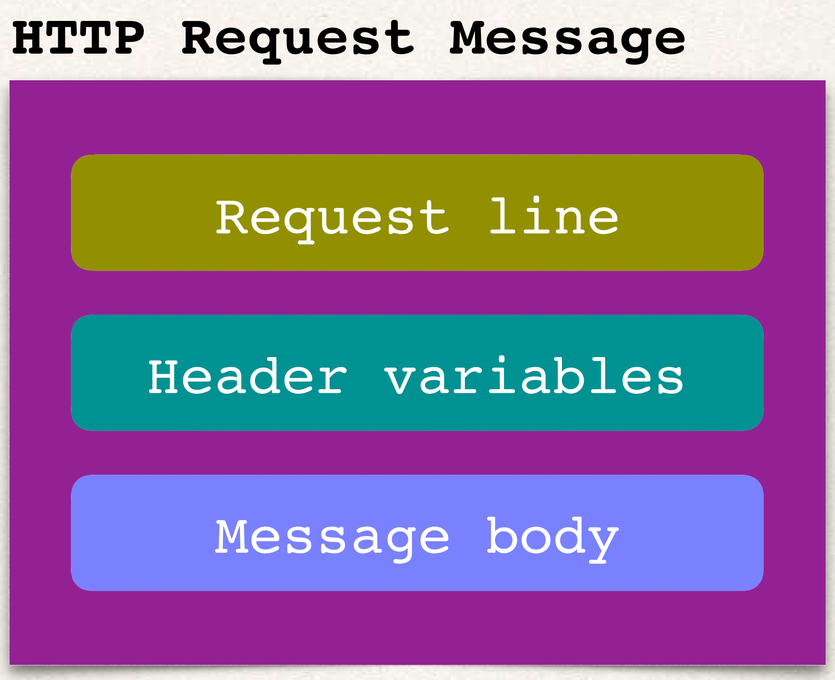
HTTP Messages

HTTP Messages

HTTP Messages

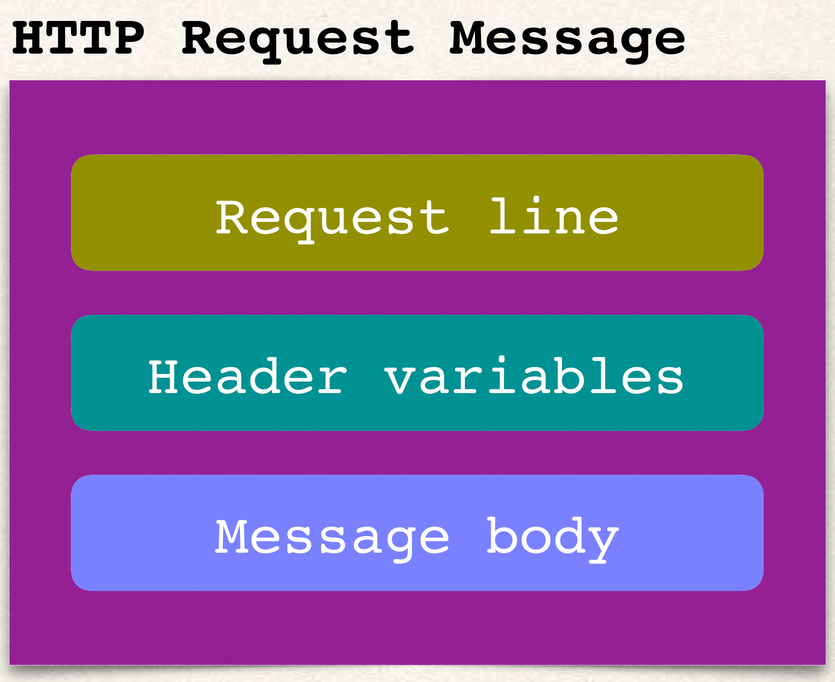


HTTP Request Message



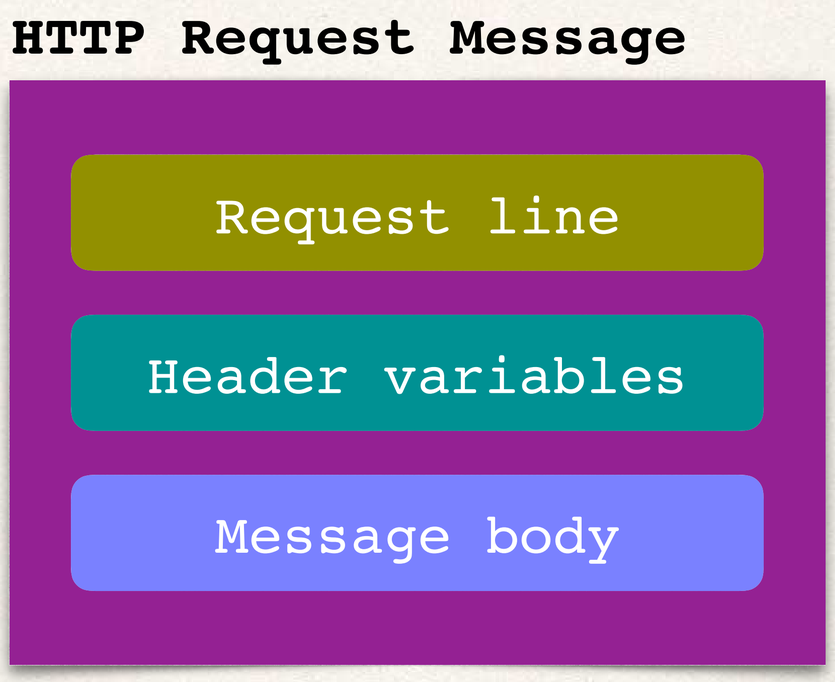
HTTP Request Message

- Request line: the HTTP command



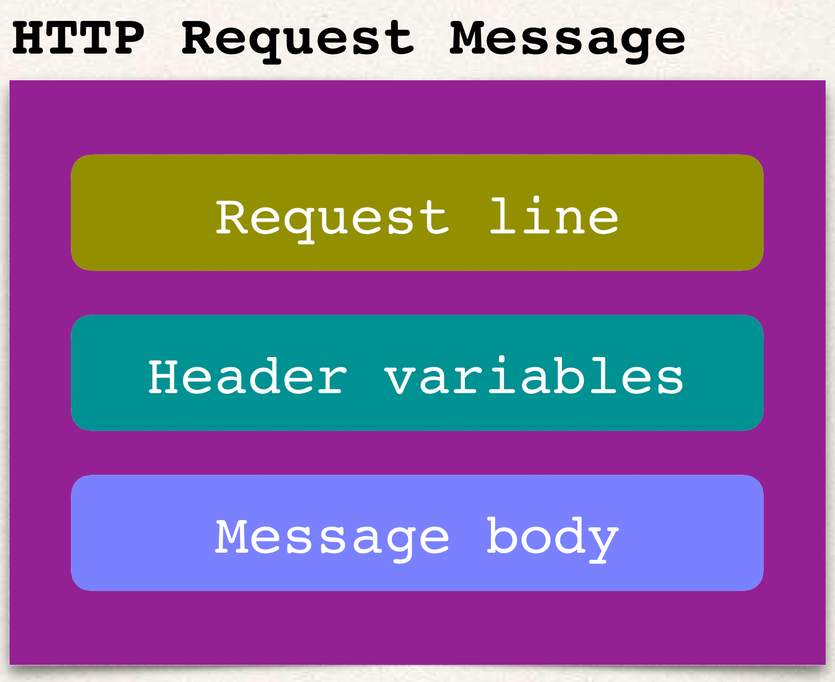
HTTP Request Message

- Request line: the HTTP command
- Header variables: request metadata

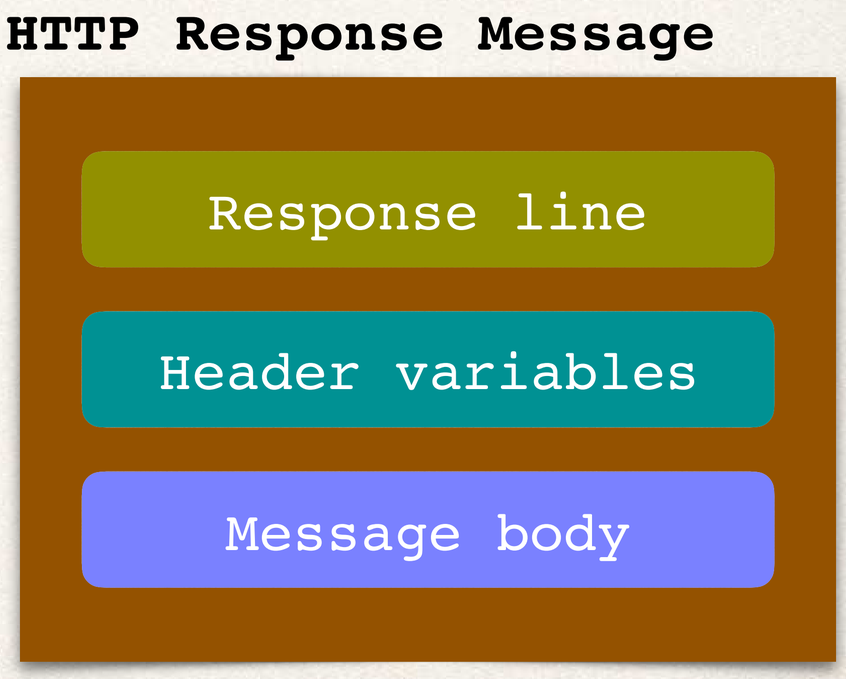


HTTP Request Message

- Request line: the HTTP command
- Header variables: request metadata
- Message body: contents of message

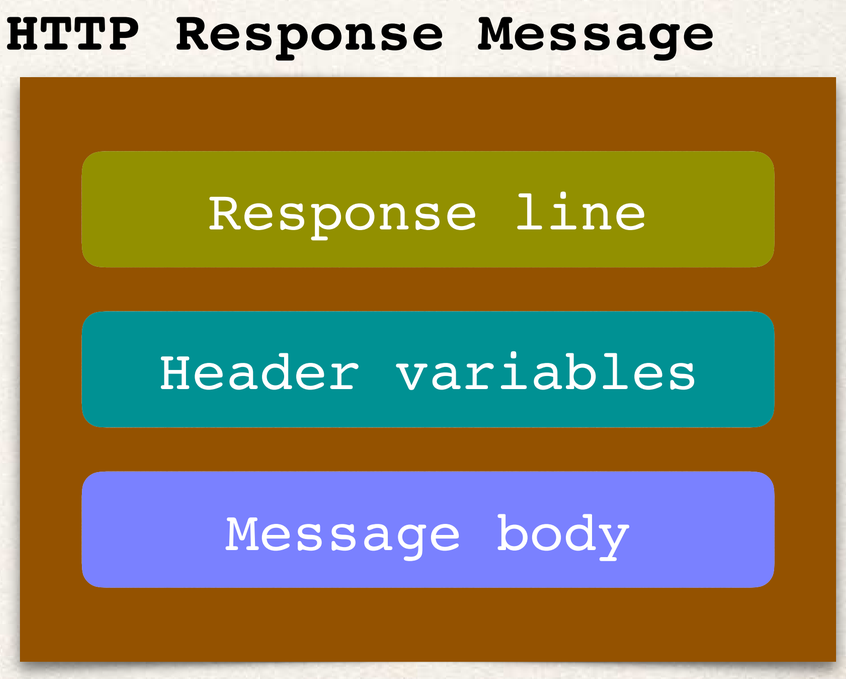


HTTP Response Message



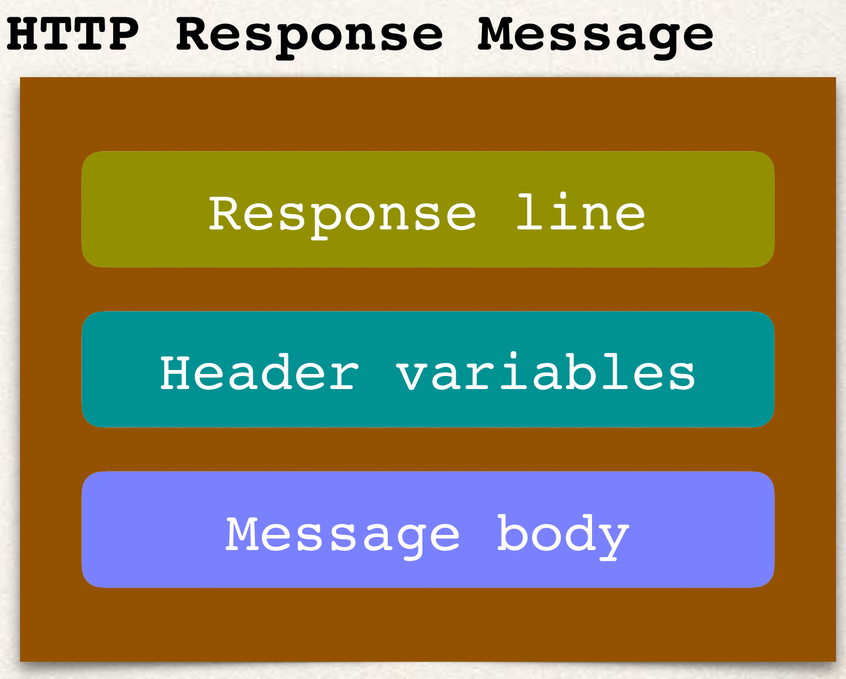
HTTP Response Message

- Response line: server protocol and status code



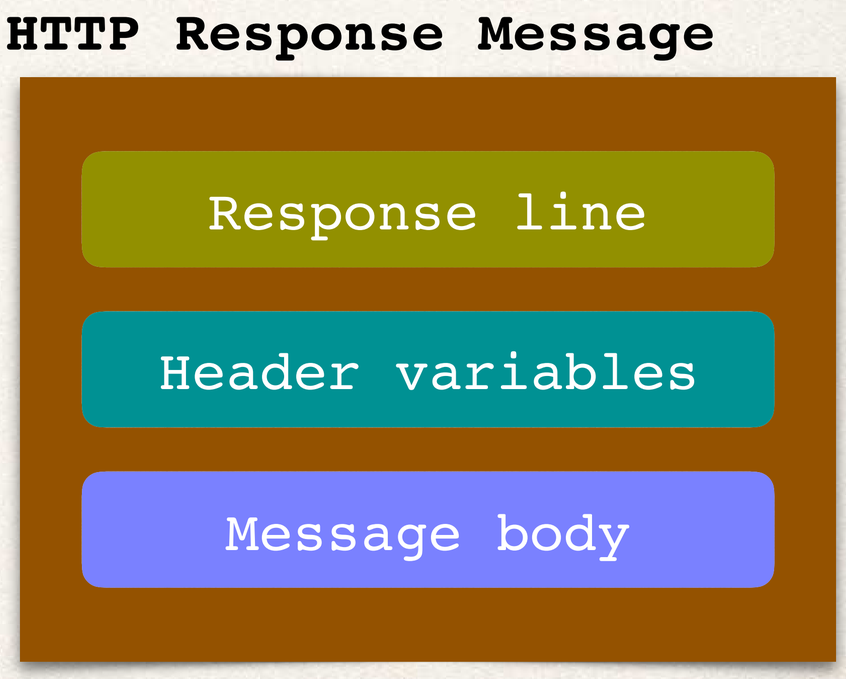
HTTP Response Message

- Response line: server protocol and status code
- Header variables: response metadata



HTTP Response Message

- Response line: server protocol and status code
- Header variables: response metadata
- Message body: contents of message



HTTP Response - Status Codes

HTTP Response - Status Codes

Code Range	Description

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational
200 - 299	Successful

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client error

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client error
500 - 599	Server error

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client error
500 - 599	Server error

401 Authentication Required
404 File Not Found

HTTP Response - Status Codes

Code Range	Description
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client error
500 - 599	Server error

401 Authentication Required
404 File Not Found

500 Internal Server Error

MIME Content Types

MIME Content Types

- The message format is described by MIME content type

MIME Content Types

- The message format is described by MIME content type
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension

MIME Content Types

- The message format is described by MIME content type
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension

- Basic Syntax: `type/sub-type`

MIME Content Types

- The message format is described by MIME content type
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension
- Basic Syntax: `type/sub-type`
- Examples

MIME Content Types

- The message format is described by MIME content type
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension
- Basic Syntax: `type/sub-type`
- Examples
  - `text/html`, `text/plain`

MIME Content Types

- The message format is described by MIME content type
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension
- Basic Syntax: `type/sub-type`
- Examples
  - `text/html`, `text/plain`
  - `application/json`, `application/xml`, ...

Client Tool

Client Tool

- We need a **client tool**

Client Tool

- We need a **client tool**

Client Tool

- We need a **client tool**

- Send HTTP requests to the REST Web Service / API

Client Tool

- We need a **client tool**

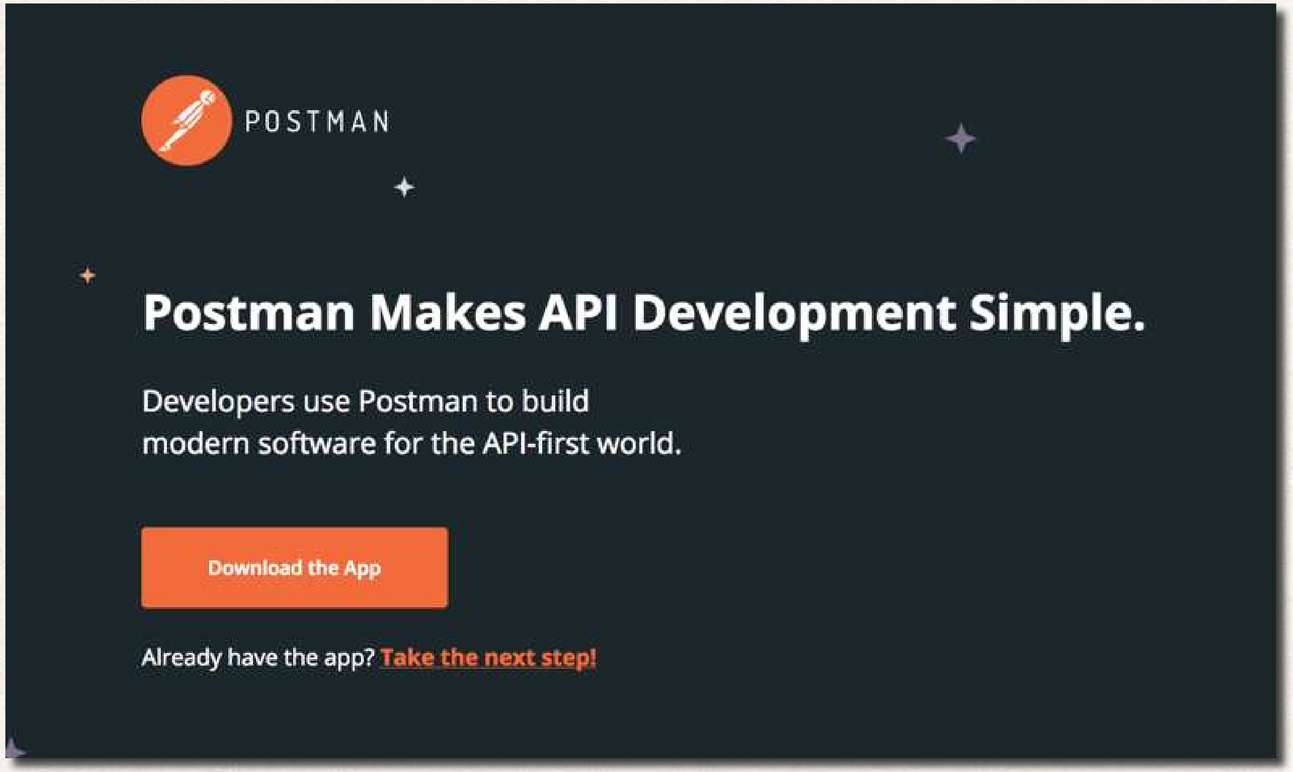
- Send HTTP requests to the REST Web Service / API

Client Tool

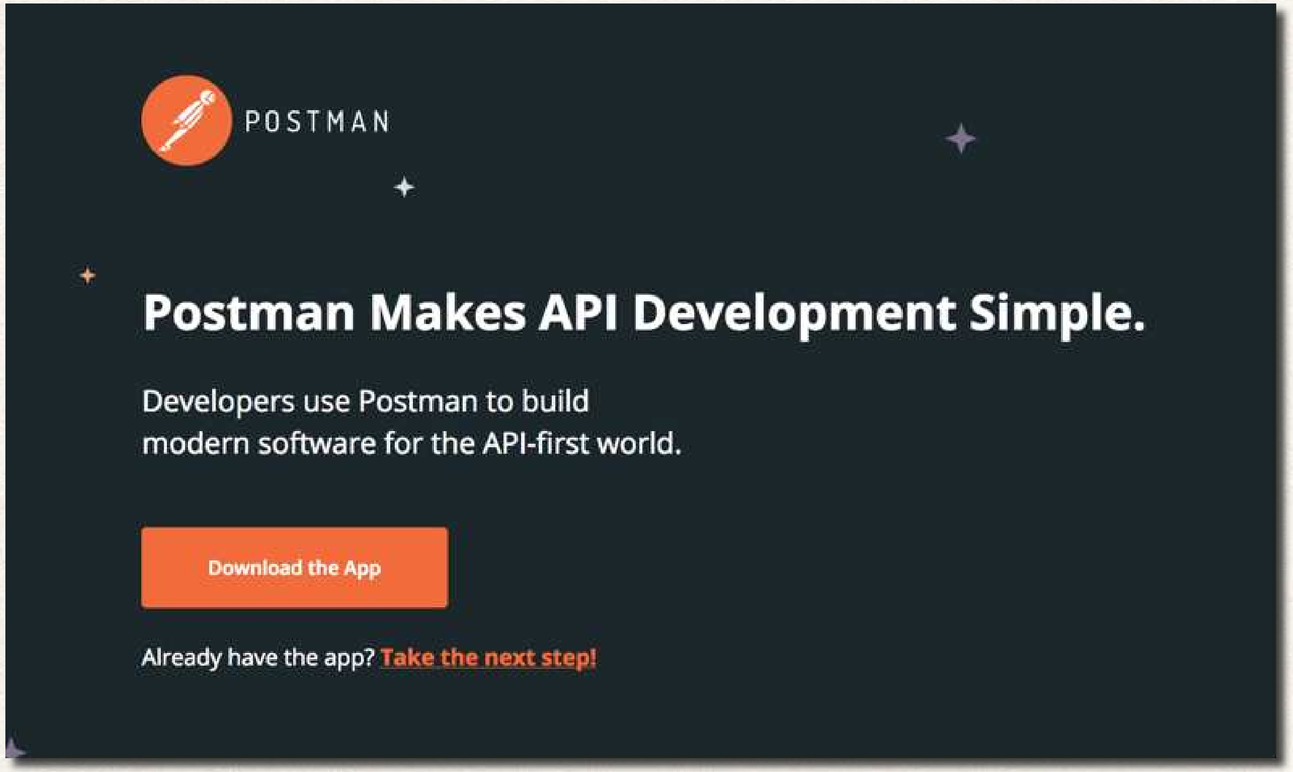
- We need a **client tool**
- Send HTTP requests to the REST Web Service / API
- Plenty of tools available: curl, Postman, etc …

Postman

Postman

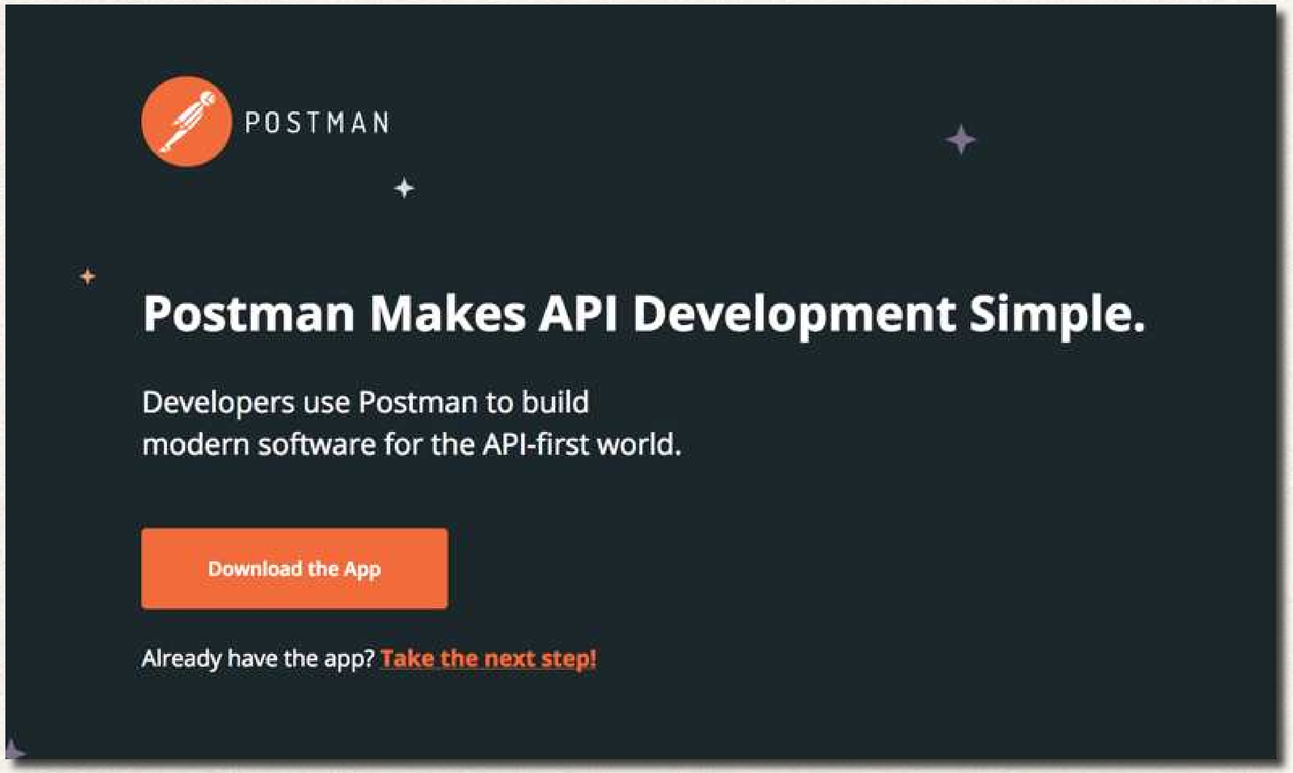

POSTMAN
Postman Makes API Development Simple.
Developers use Postman to build
modern software for the API-first world.
Download the App
Already have the app? Take the next step!

Postman



www.getpostman.com

Postman



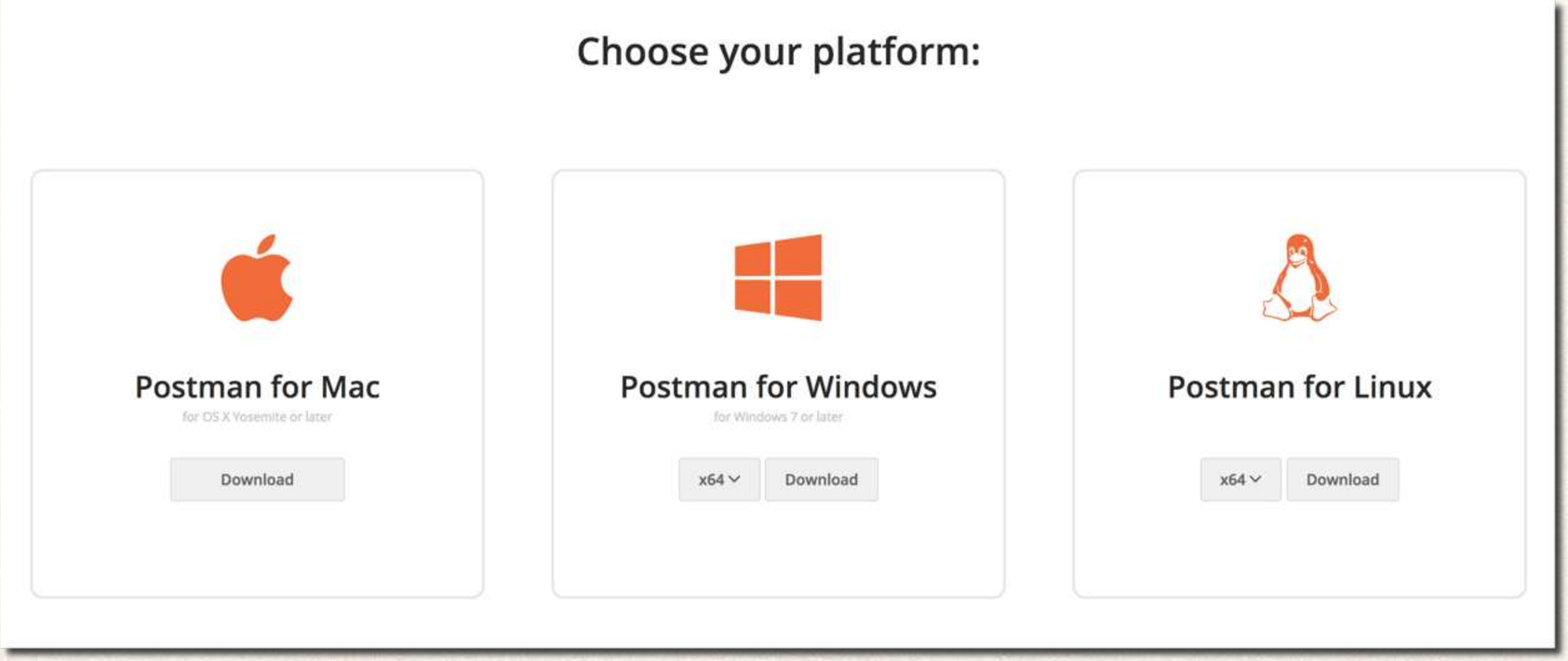
Free developer plan

www.getpostman.com

Install Postman Now

Install Postman Now

www.getpostman.com



Spring REST Controller

Spring REST Support

Spring REST Support

- Spring Web MVC provides support for Spring REST

Spring REST Support

- Spring Web MVC provides support for Spring REST

- New annotation **`@RestController`**

Spring REST Support

- Spring Web MVC provides support for Spring REST

- New annotation **`@RestController`**
  - Extension of `@Controller`

Spring REST Support

- Spring Web MVC provides support for Spring REST

- New annotation **`@RestController`**
  - Extension of `@Controller`
  - Handles REST requests and responses

Spring REST Support

- Spring Web MVC provides support for Spring REST

- New annotation **`@RestController`**
  - Extension of `@Controller`
  - Handles REST requests and responses

- Spring REST will also automatically convert Java POJOs to JSON

Spring REST Support

- Spring Web MVC provides support for Spring REST

- New annotation **`@RestController`**
  - Extension of `@Controller`
  - Handles REST requests and responses

- Spring REST will also automatically convert Java POJOs to JSON
  - As long as the Jackson project is on the classpath or pom.xml

Spring REST Hello World

Spring REST Hello World

REST
Client

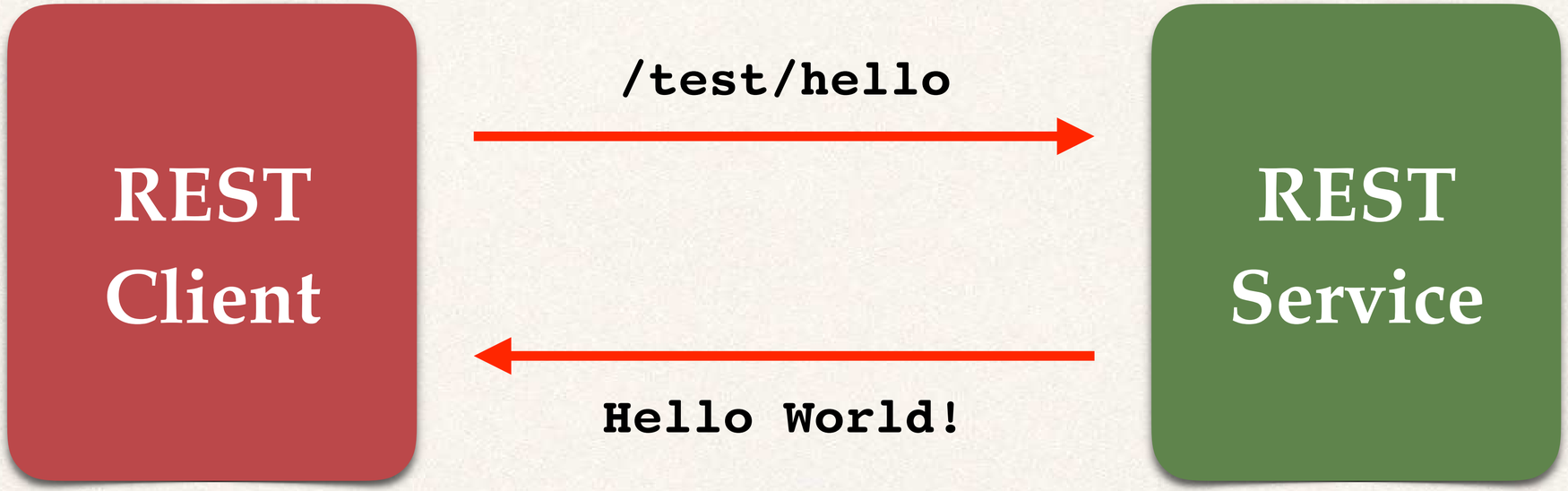
Spring REST Hello World

REST Client

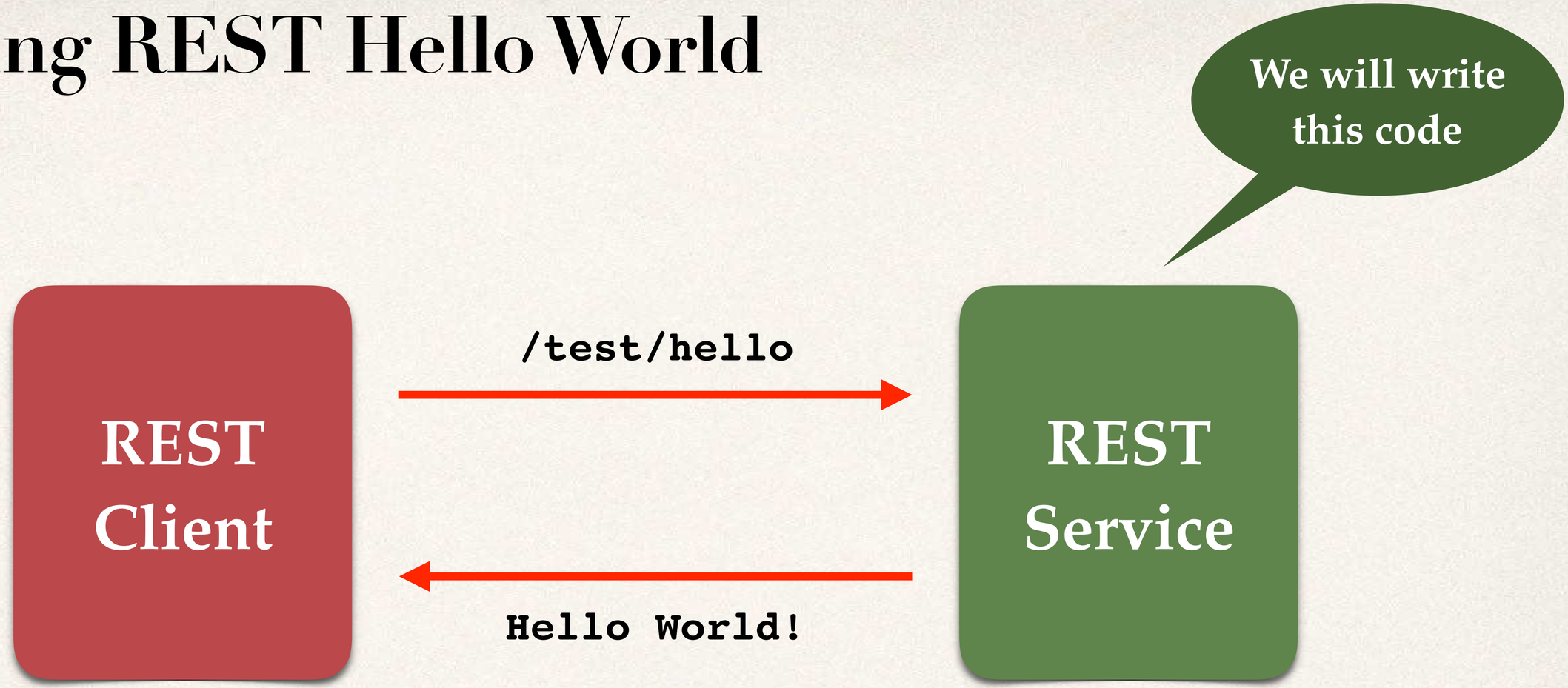
REST Service

Spring REST Hello World

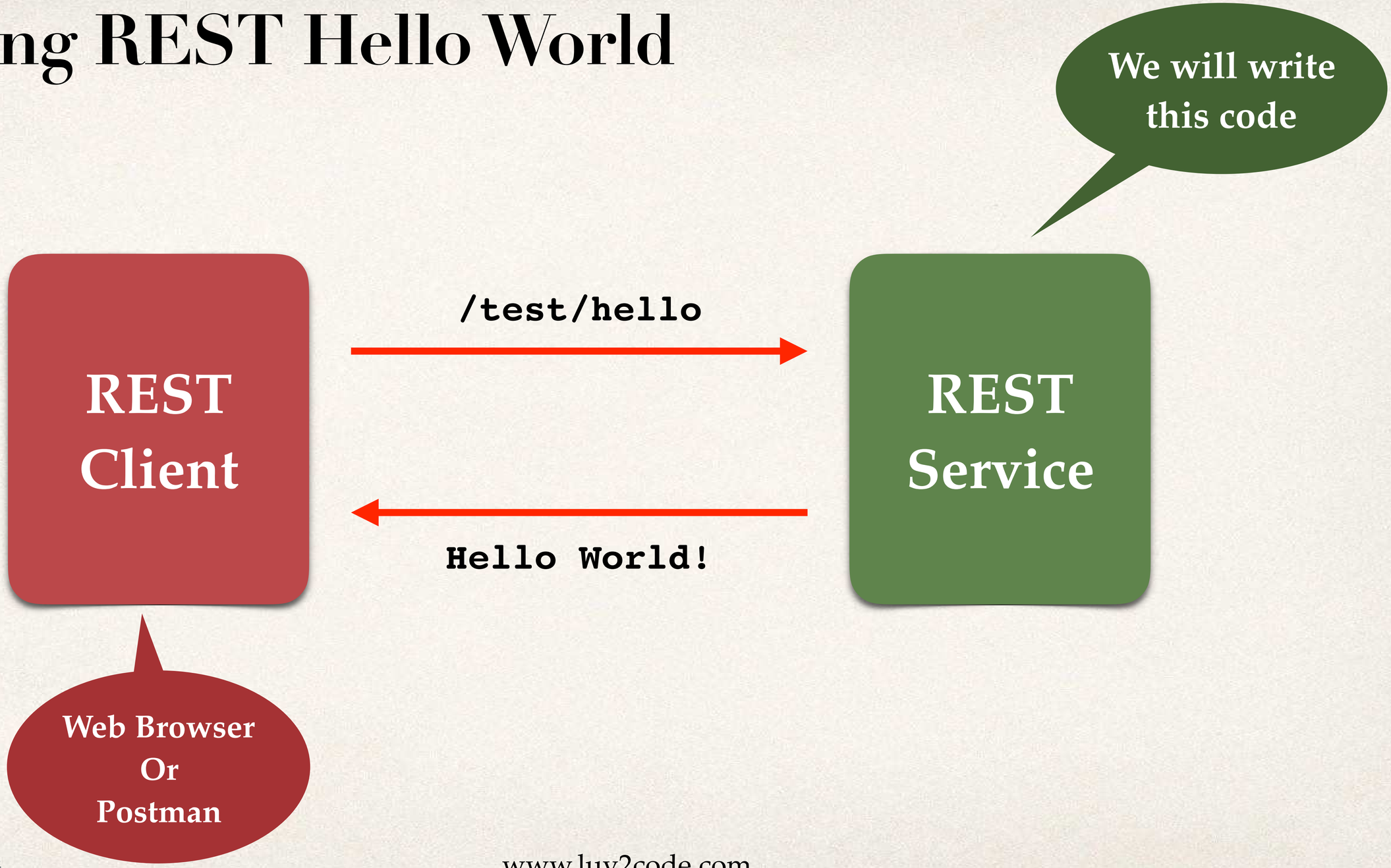
Spring REST Hello World



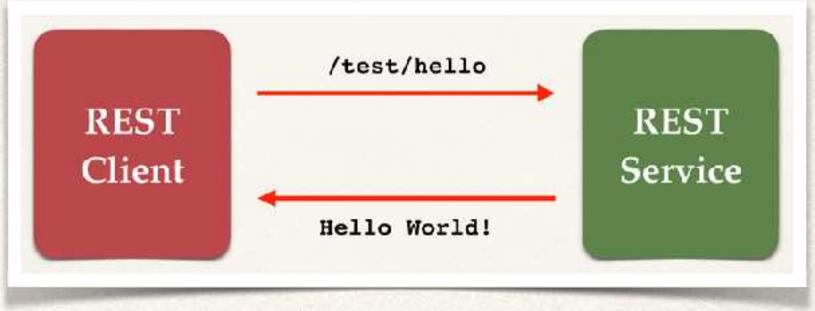
Spring REST Hello World



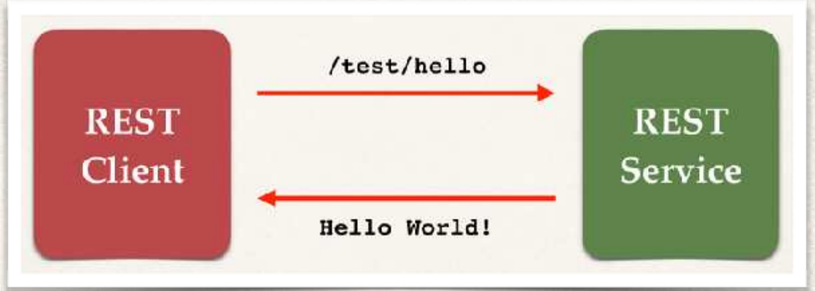
Spring REST Hello World



Spring REST Controller

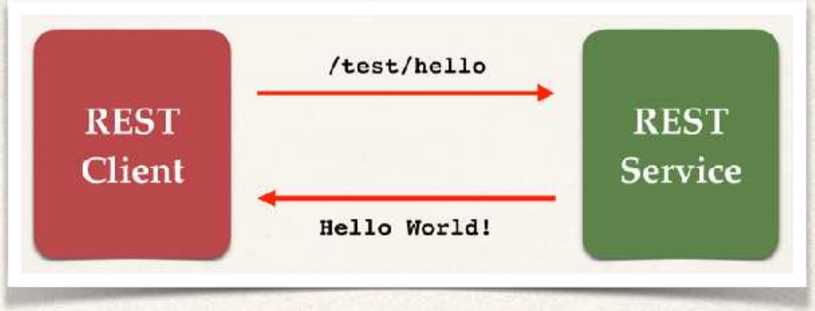


Spring REST Controller




```
@RestController
@RequestMapping("/test")
public class DemoRestController {
```

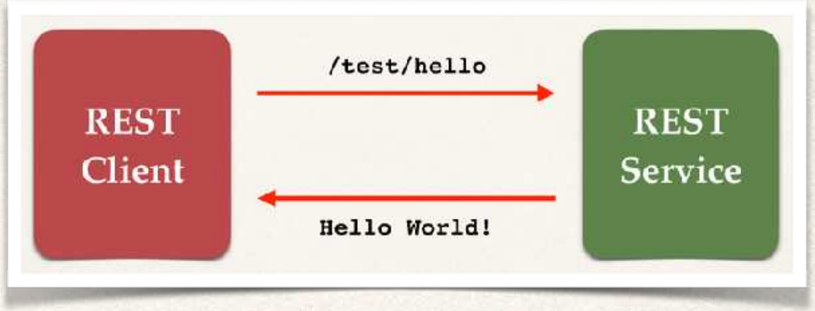
Spring REST Controller



Adds REST support

```
@RestController
@RequestMapping("/test")
public class DemoRestController {
```

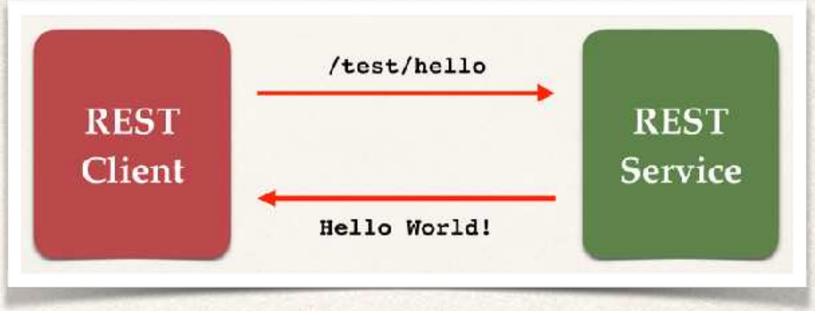
Spring REST Controller



Adds REST support

```
@RestController
@RequestMapping("/test")
public class DemoRestController {

  @GetMapping("/hello")
  public String sayHello() {
    return "Hello World!";
  }

}
```

Spring REST Controller

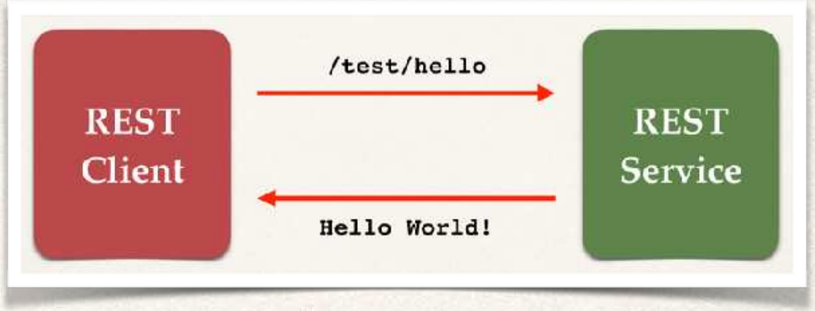


Adds REST support

```
@RestController
@RequestMapping("/test")
public class DemoRestController {

  @GetMapping("/hello")
  public String sayHello() {
    return "Hello World!";
  }

}
```

Access the REST endpoint at `/test/hello`

Spring REST Controller



Adds REST support

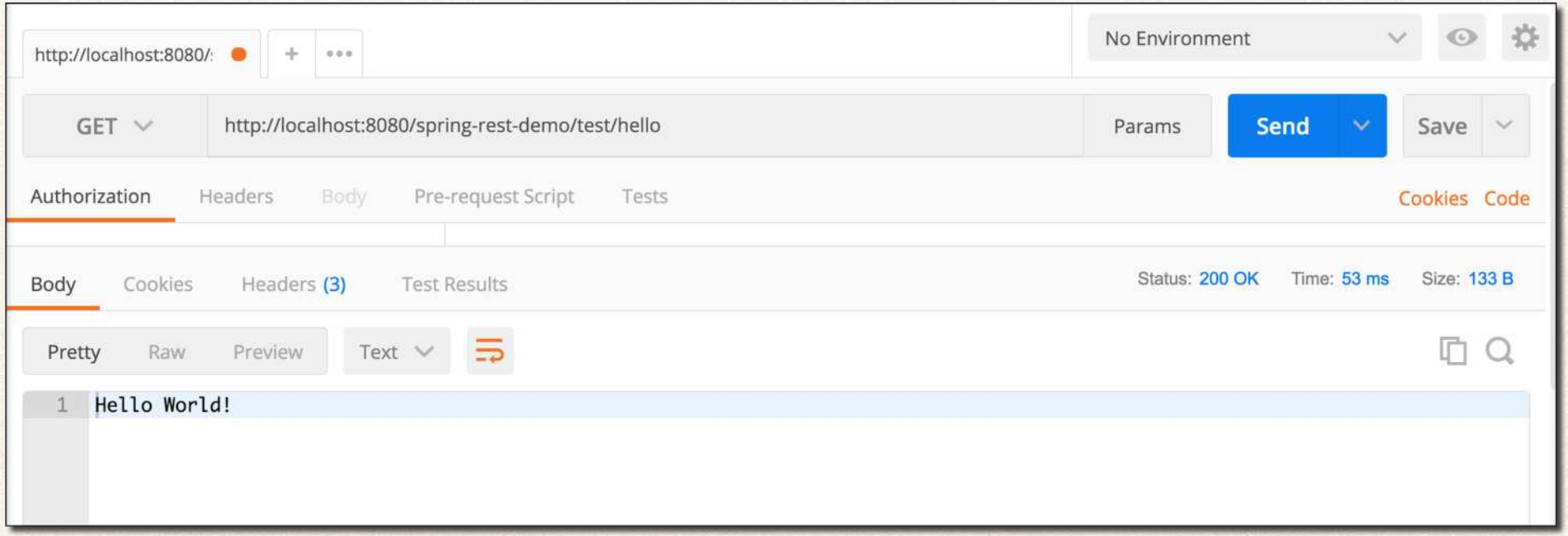
```
@RestController
@RequestMapping("/test")
public class DemoRestController {

  @GetMapping("/hello")
  public String sayHello() {
    return "Hello World!";
  }

}
```

Access the REST endpoint at `/test/hello`

Returns content to client

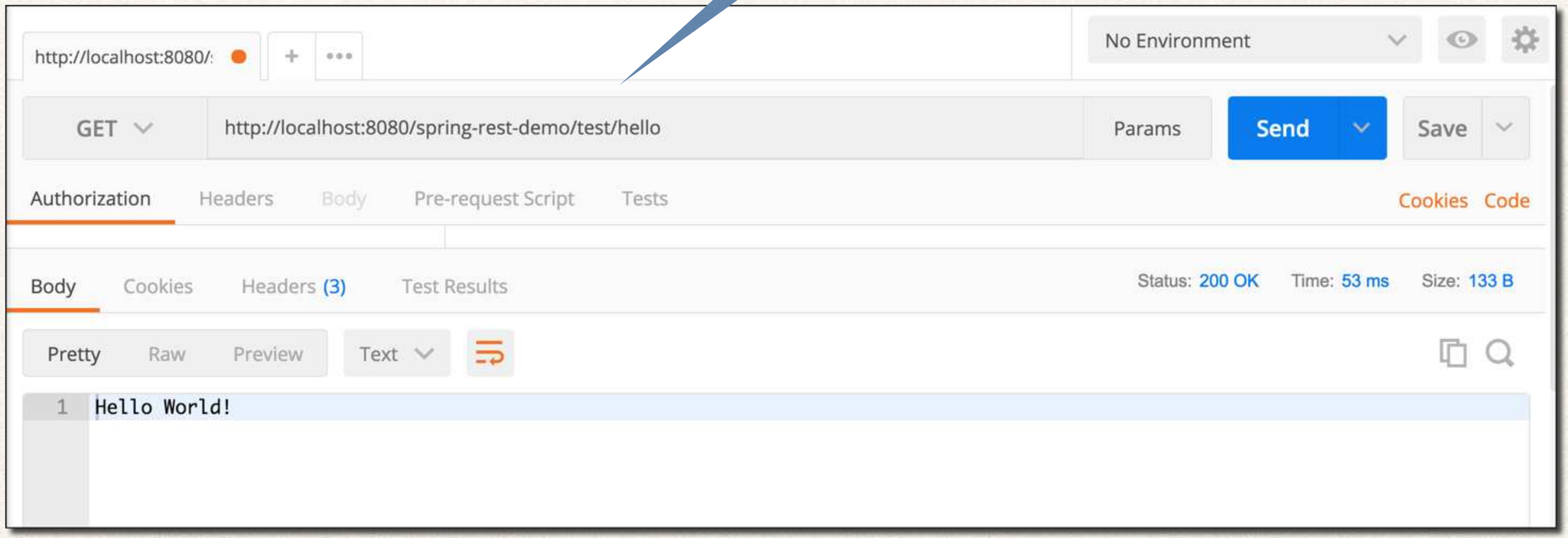
Testing with REST Client - Postman

Testing with REST Client - Postman



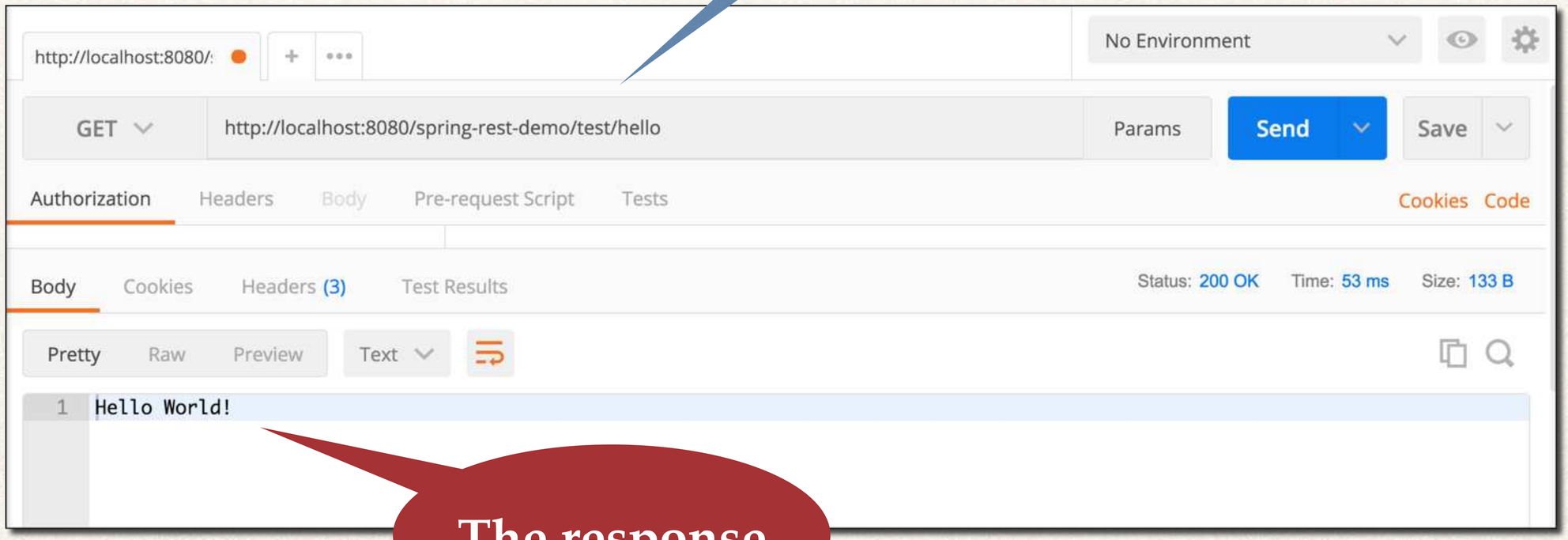
Testing with REST Client - Postman

Access the REST endpoint at `/test/hello`



Testing with REST Client - Postman

Access the REST endpoint at `/test/hello`



The response

Testing with REST Client - Web Browser

Testing with REST Client - Web Browser

Testing with REST Client - Web Browser

Access the REST endpoint at `/test/hello`

Testing with REST Client - Web Browser

Access the REST endpoint at `/test/hello`

The response

Web Browser vs Postman

Web Browser vs Postman

- For simple REST testing for GET requests

Web Browser vs Postman

- For simple REST testing for GET requests
  - Web Browser and Postman are similar

Web Browser vs Postman

- For simple REST testing for GET requests
  - Web Browser and Postman are similar

- However, for advanced REST testing: POST, PUT etc …

Web Browser vs Postman

- For simple REST testing for GET requests
  - Web Browser and Postman are similar

- However, for advanced REST testing: POST, PUT etc …
  - Postman has much better support

Web Browser vs Postman

- For simple REST testing for GET requests
  - Web Browser and Postman are similar

- However, for advanced REST testing: POST, PUT etc …
  - Postman has much better support
  - POSTing JSON data, setting content type

POSTMAN

Web Browser vs Postman

- For simple REST testing for GET requests
  - Web Browser and Postman are similar

- However, for advanced REST testing: POST, PUT etc …
  - Postman has much better support
  - POSTing JSON data, setting content type
  - Passing HTTP request headers, authentication etc …

POSTMAN

Spring REST Controller - Dev Process

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Add Maven dependency for Spring MVC and Jackson project

Development Process

Step-By-Step

1. Add Maven dependency for Spring MVC and Jackson project

2. Add code for All Java Config: @Configuration

Development Process

Step-By-Step

1. Add Maven dependency for Spring MVC and Jackson project

2. Add code for All Java Config: @Configuration

3. Add code for All Java Config: Servlet Initializer

Development Process

Step-By-Step

1. Add Maven dependency for Spring MVC and Jackson project

2. Add code for All Java Config: @Configuration

3. Add code for All Java Config: Servlet Initializer

4. Create Spring REST Service using @RestController

Step 1: Add Maven Dependencies

File: pom.xml

Step 1: Add Maven Dependencies

File: pom.xml

```
<!-- Add Spring MVC and REST support -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>...</version>
</dependency>
```

Will load all supporting dependencies:
spring-core, logging etc …

Step 1: Add Maven Dependencies

File: pom.xml

```
<!-- Add Spring MVC and REST support -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>...</version>
</dependency>

<!-- Add Jackson for JSON converters -->
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>...</version>
</dependency>
```

Add Jackson support
we'll need it later for converting
JSON <--> POJO

Step 1: Add Maven Dependencies

File: pom.xml

```
<!-- Add Spring MVC and REST support -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>...</version>
</dependency>

<!-- Add Jackson for JSON converters -->
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>...</version>
</dependency>

<!-- Add Servlet support for
    Spring's AbstractAnnotationConfigDispatcherServletInitializer -->
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>javax.servlet-api</artifactId>
   <version>...</version>
</dependency>
```

Add Servlet support

Step 2: All Java Config: @Configuration

File: DemoAppConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.luv2code.springdemo")
public class DemoAppConfig {

}
```

Web App Initializer

Web App Initializer

- Spring MVC provides support for web app initialization

Web App Initializer

- Spring MVC provides support for web app initialization
- Makes sure your code is automatically detected

Web App Initializer

- Spring MVC provides support for web app initialization
- Makes sure your code is automatically detected
- Your code is used to initialize the servlet container

Web App Initializer

- Spring MVC provides support for web app initialization
- Makes sure your code is automatically detected
- Your code is used to initialize the servlet container

AbstractAnnotationConfigDispatcherServletInitializer

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list
  - Extend this abstract base class

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list
  - Extend this abstract base class
  - Override required methods

Web App Initializer (more info)

AbstractAnnotationConfigDispatcherServletInitializer

- Your TO DO list
  - Extend this abstract base class
  - Override required methods
  - Specify servlet mapping and location of your app config

Step 3: All Java Config: Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

}
```

Step 3: All Java Config: Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      // TODO Auto-generated method stub
      return null;
   }









}
```

Step 3: All Java Config: Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      // TODO Auto-generated method stub
      return null;
   }

   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[] { DemoAppConfig.class };
   }





}
```

Step 3: All Java Config: Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      // TODO Auto-generated method stub
      return null;
   }

   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[] { DemoAppConfig.class };
   }



}
```

Our config class from Step 2

Step 3: All Java Config: Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      // TODO Auto-generated method stub
      return null;
   }

   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[] { DemoAppConfig.class };
   }





}
```

Step 3: All Java Config: Servlet Initializer

File:MySpringMvcDispatcherServletInitializer.java

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      // TODO Auto-generated method stub
      return null;
   }

   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[] { DemoAppConfig.class };
   }

   @Override
   protected String[] getServletMappings() {
      return new String[] { "/" };
   }

}
```

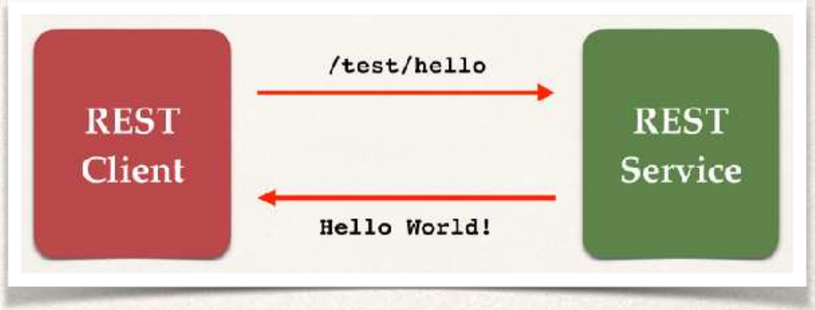
Whew!!!!

Now for the easy stuff

Create Spring REST Controller

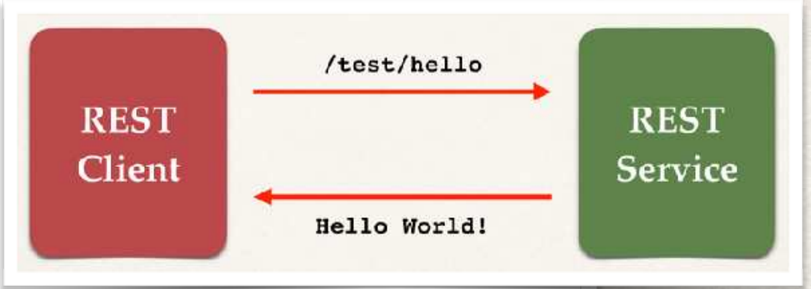
easy peazy ...

Step 4: Create Spring REST Service



Step 4: Create Spring REST Service

```
@RestController
@RequestMapping("/test")
public class DemoRestController {

  @GetMapping("/hello")
  public String sayHello() {
    return "Hello World!";
  }

}
```



Spring REST Service - Students

Create a New Service

Create a New Service

- Return a list of students

Create a New Service

- Return a list of students

GET `/api/students` *Returns a list of students*

Spring REST Service

Spring REST Service

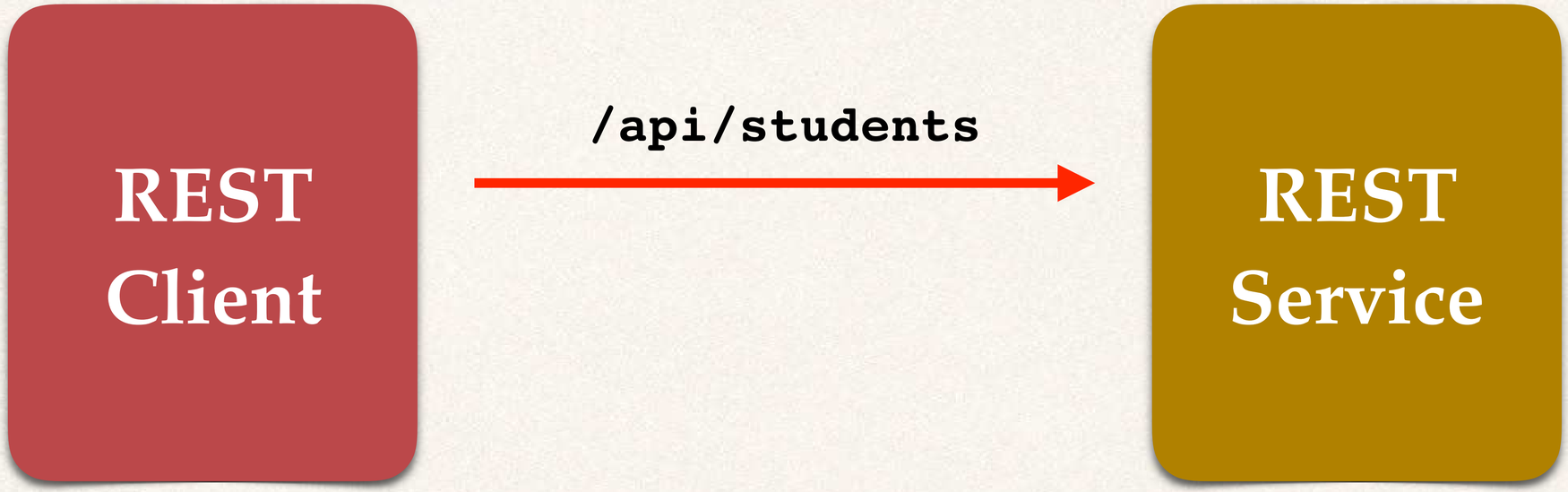
REST
Client

Spring REST Service

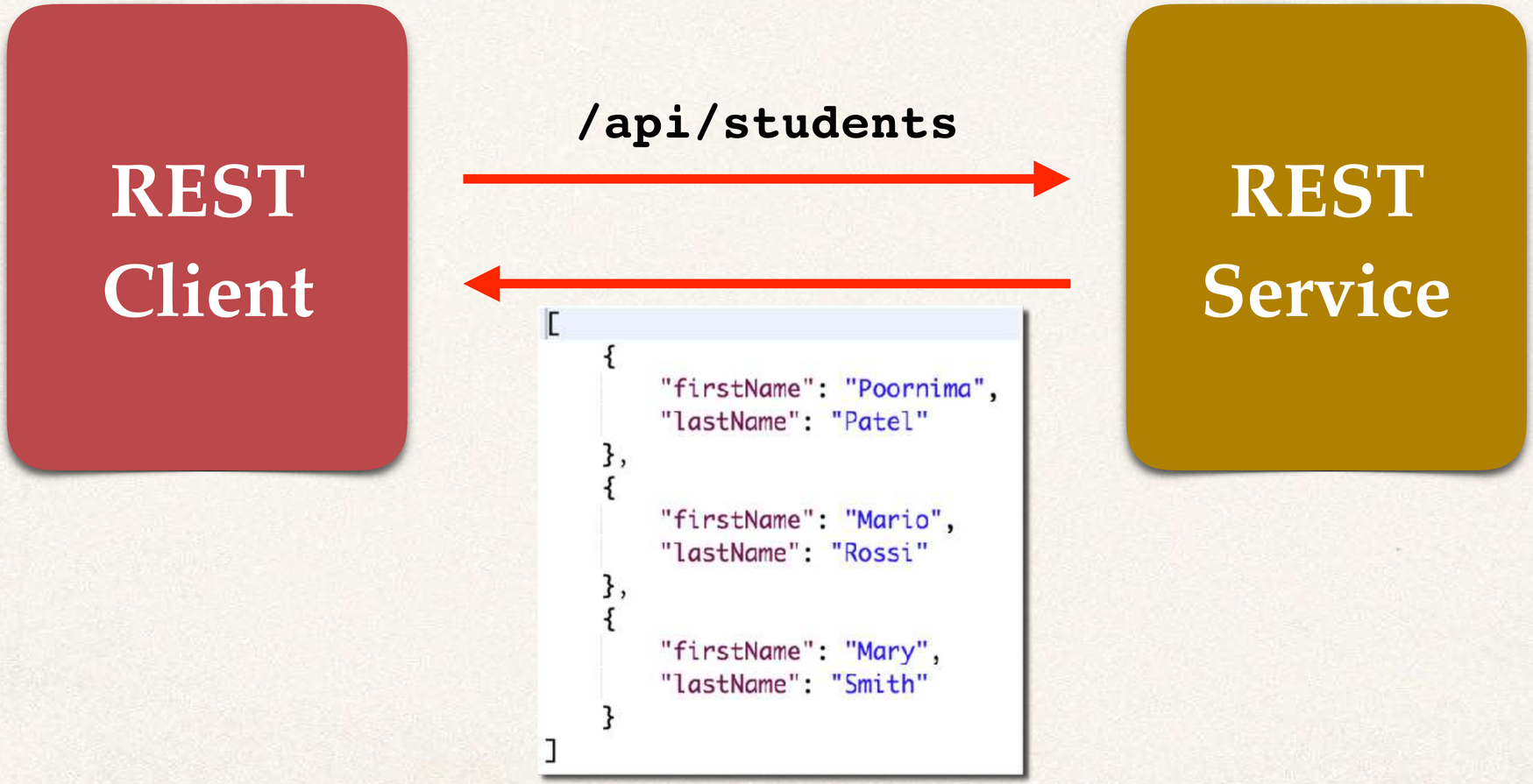
REST Client

REST Service

Spring REST Service



Spring REST Service



Spring REST Service

We will write this code

REST Client

/api/students

REST Service

```
[
    {
        "firstName": "Poornima",
        "lastName": "Patel"
    },
    {
        "firstName": "Mario",
        "lastName": "Rossi"
    },
    {
        "firstName": "Mary",
        "lastName": "Smith"
    }
]
```

Spring REST Service

We will write this code

REST Client

/api/students

REST Service

```
[
    {
        "firstName": "Poornima",
        "lastName": "Patel"
    },
    {
        "firstName": "Mario",
        "lastName": "Rossi"
    },
    {
        "firstName": "Mary",
        "lastName": "Smith"
    }
]
```

Web Browser
Or
Postman

Convert Java POJO to JSON

Convert Java POJO to JSON

- Our REST Service will return **`List<Student>`**

Convert Java POJO to JSON

- Our REST Service will return **`List<Student>`**

- Need to convert **`List<Student>`** to JSON

Convert Java POJO to JSON

- Our REST Service will return **`List<Student>`**
- Need to convert **`List<Student>`** to JSON
- **Jackson** can help us out with this …

Spring and Jackson Support

Remember

Spring and Jackson Support

Remember

- Spring will automatically handle **Jackson** integration

Spring and Jackson Support

Remember

- Spring will automatically handle **Jackson** integration

- As long as the **Jackson** project is on the classpath or pom.xml

Spring and Jackson Support

Remember

- Spring will automatically handle **Jackson** integration
- As long as the **Jackson** project is on the classpath or pom.xml
- JSON data being passed to REST controller is converted to Java POJO

Spring and Jackson Support

Remember

- Spring will automatically handle **Jackson** integration
- As long as the **Jackson** project is on the classpath or pom.xml
- JSON data being passed to REST controller is converted to Java POJO
- Java POJO being returned from REST controller is converted to JSON

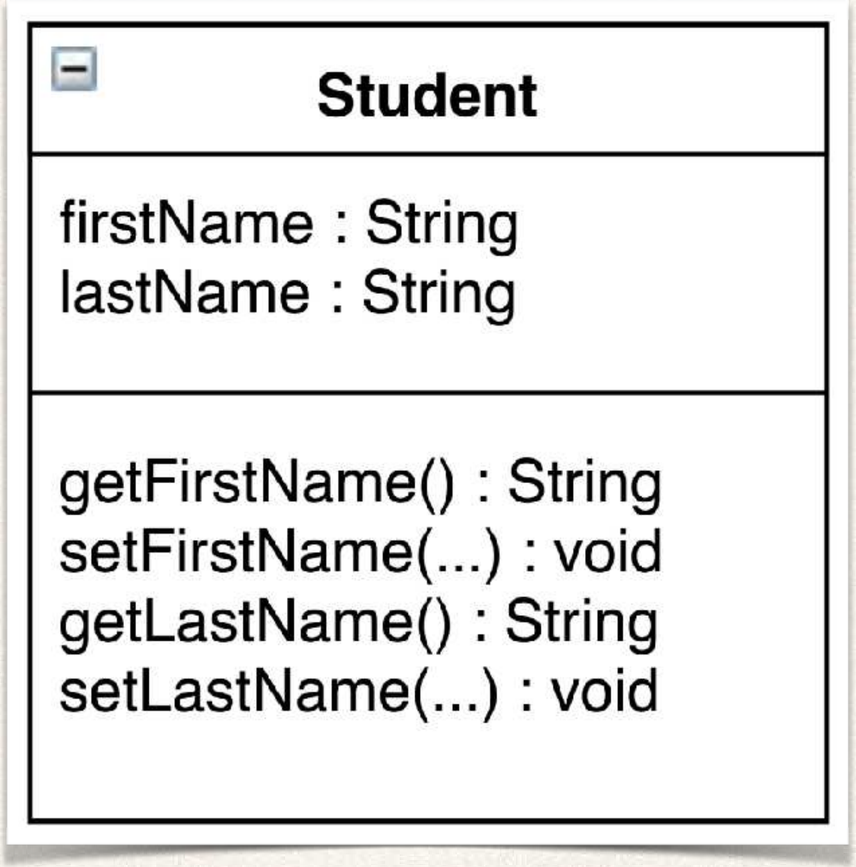
Spring and Jackson Support

Happens automatically behind the scenes

- Spring will automatically handle **Jackson** integration
- As long as the **Jackson** project is on the classpath or pom.xml
- JSON data being passed to REST controller is converted to Java POJO
- Java POJO being returned from REST controller is converted to JSON

Student POJO (class)

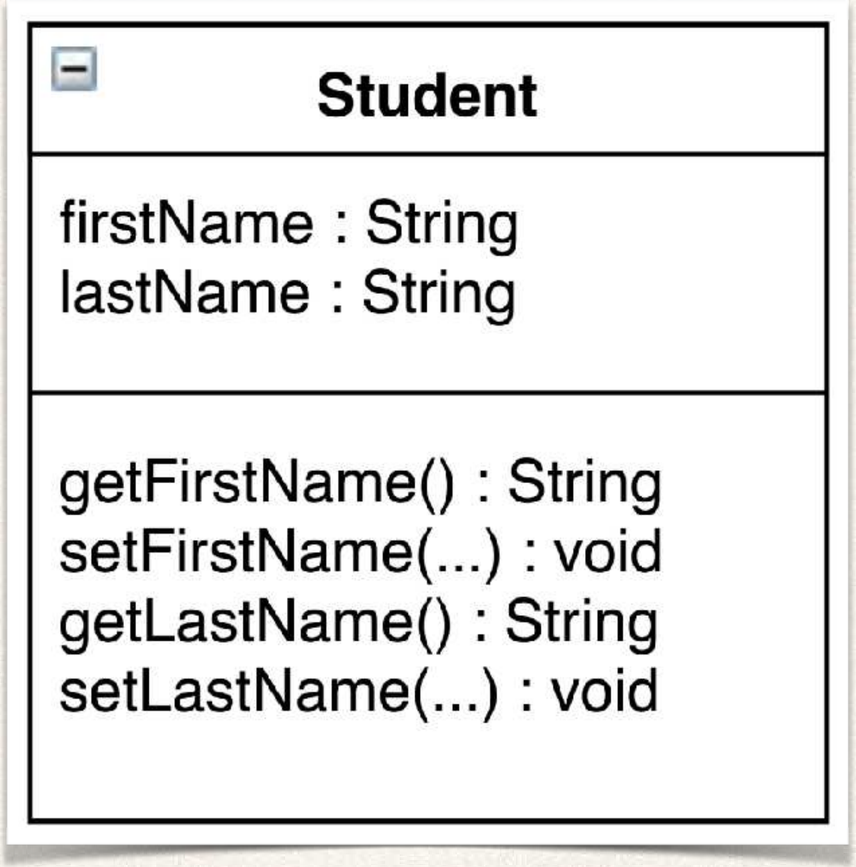
Student POJO (class)



Student POJO (class)

Java
POJO

Student



Jackson Data Binding

Remember

Jackson Data Binding

Remember

- Jackson will call appropriate getter / setter method

Jackson Data Binding

Remember

- Jackson will call appropriate getter / setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Jackson Data Binding

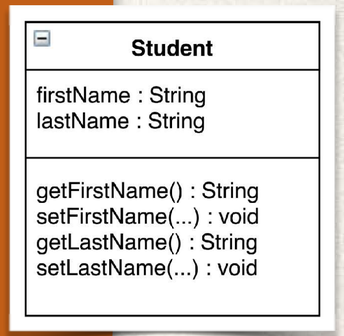
Remember

- Jackson will call appropriate getter / setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Java
POJO

Student



Jackson Data Binding

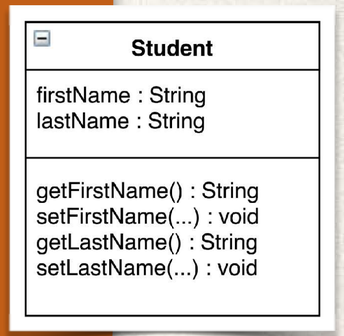
Remember

- Jackson will call appropriate getter / setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Java
POJO

Student



Jackson Data Binding

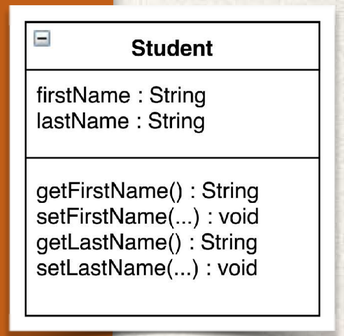
Remember

- Jackson will call appropriate getter / setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Java
POJO

Student



Jackson Data Binding

Remember

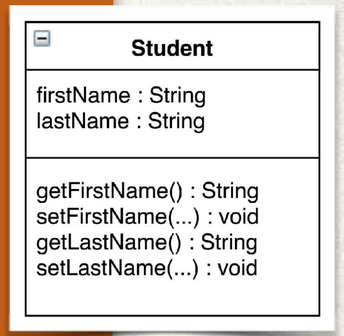
- Jackson will call appropriate getter / setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Jackson will do this work

Java POJO

Student



Spring REST Service

Spring REST Service

REST
Client

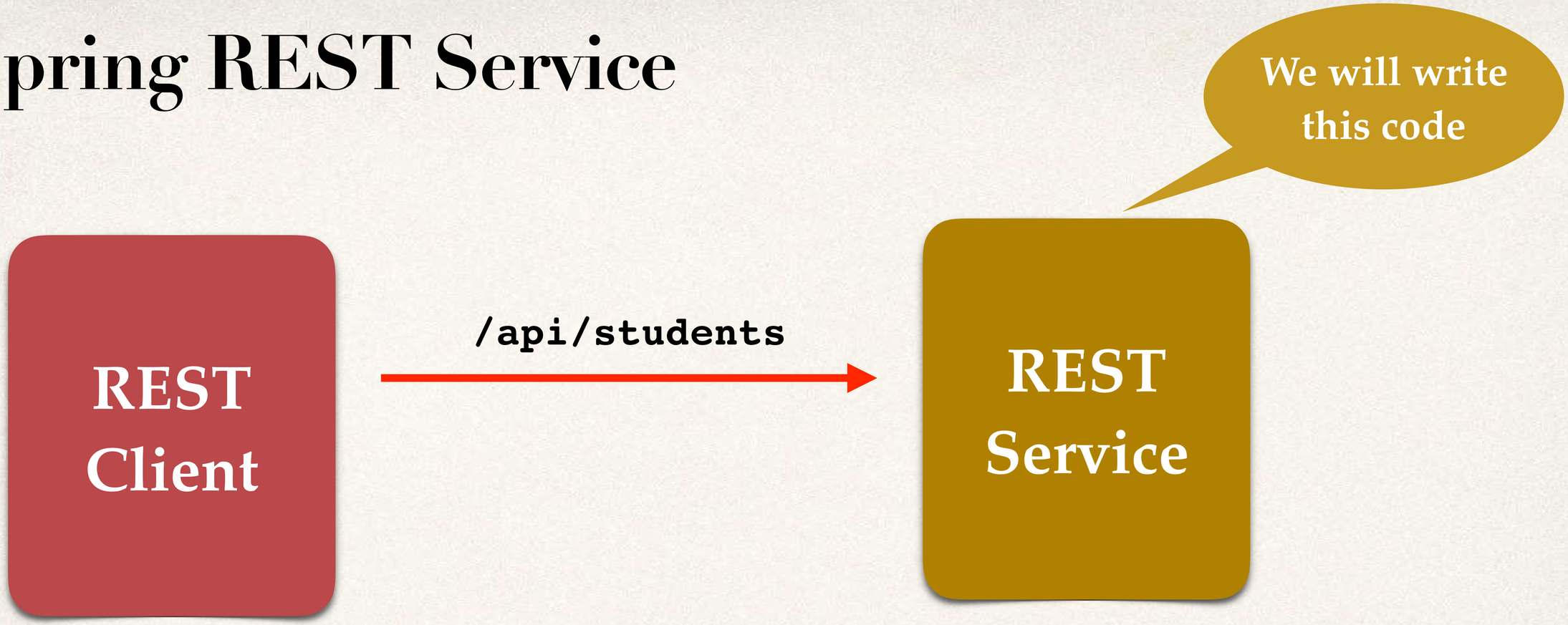
Spring REST Service

REST
Client

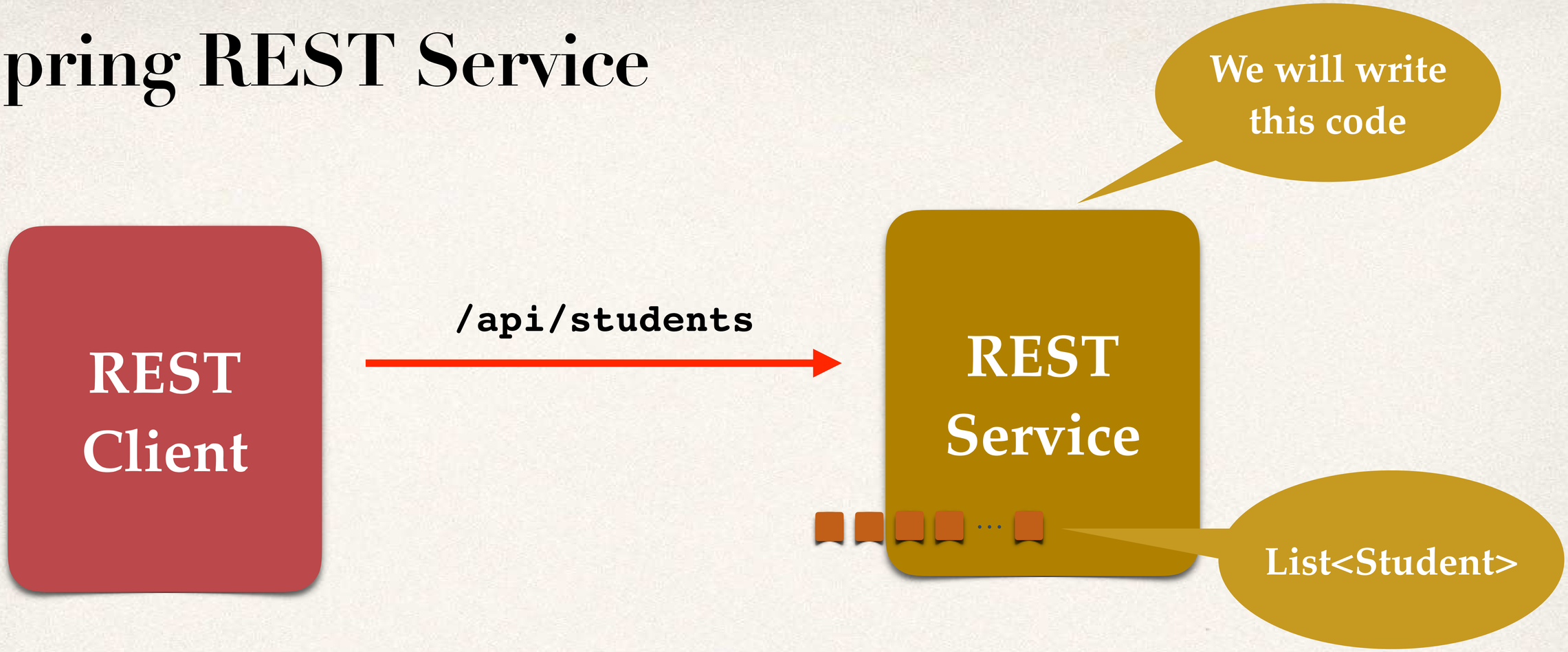
REST
Service

Spring REST Service

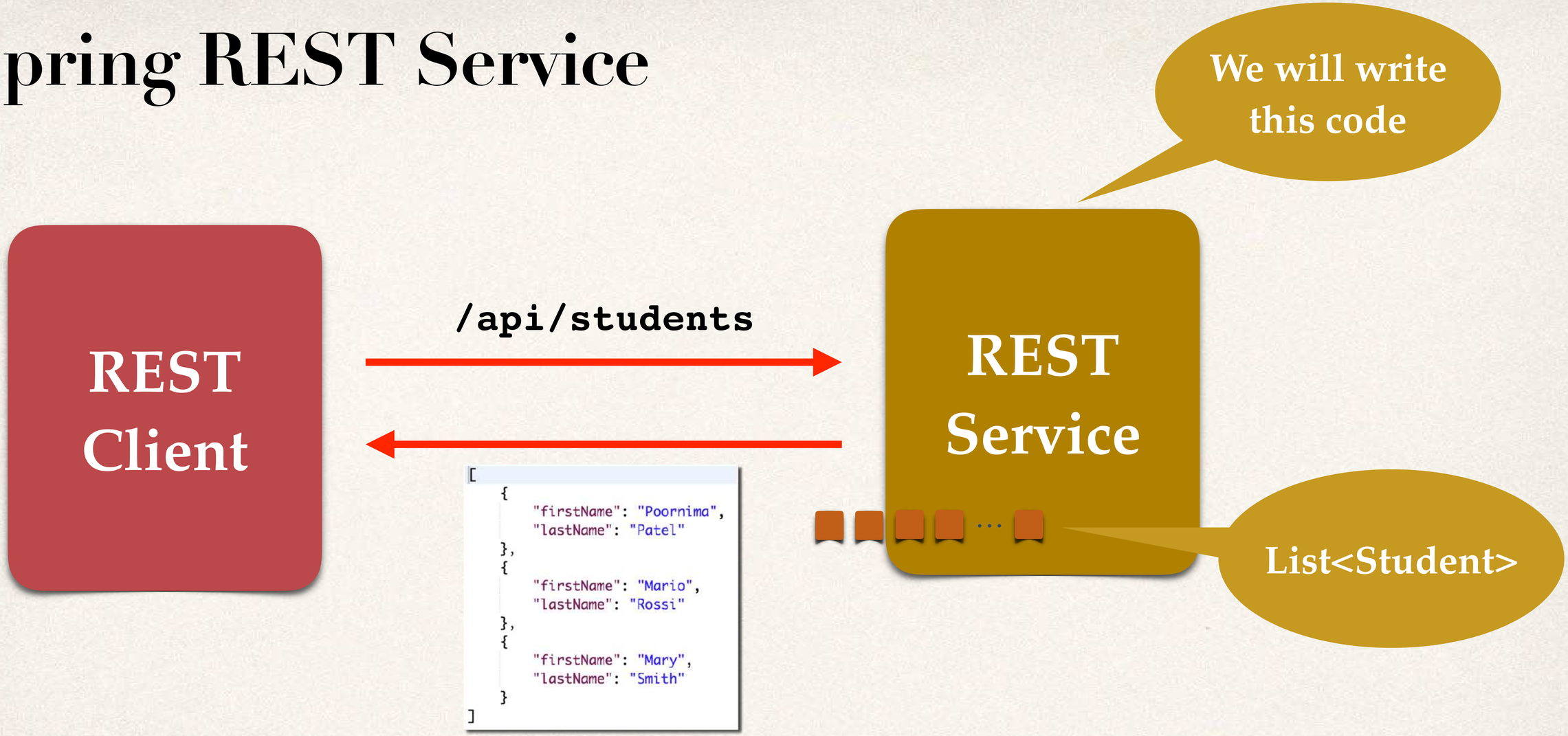
Spring REST Service



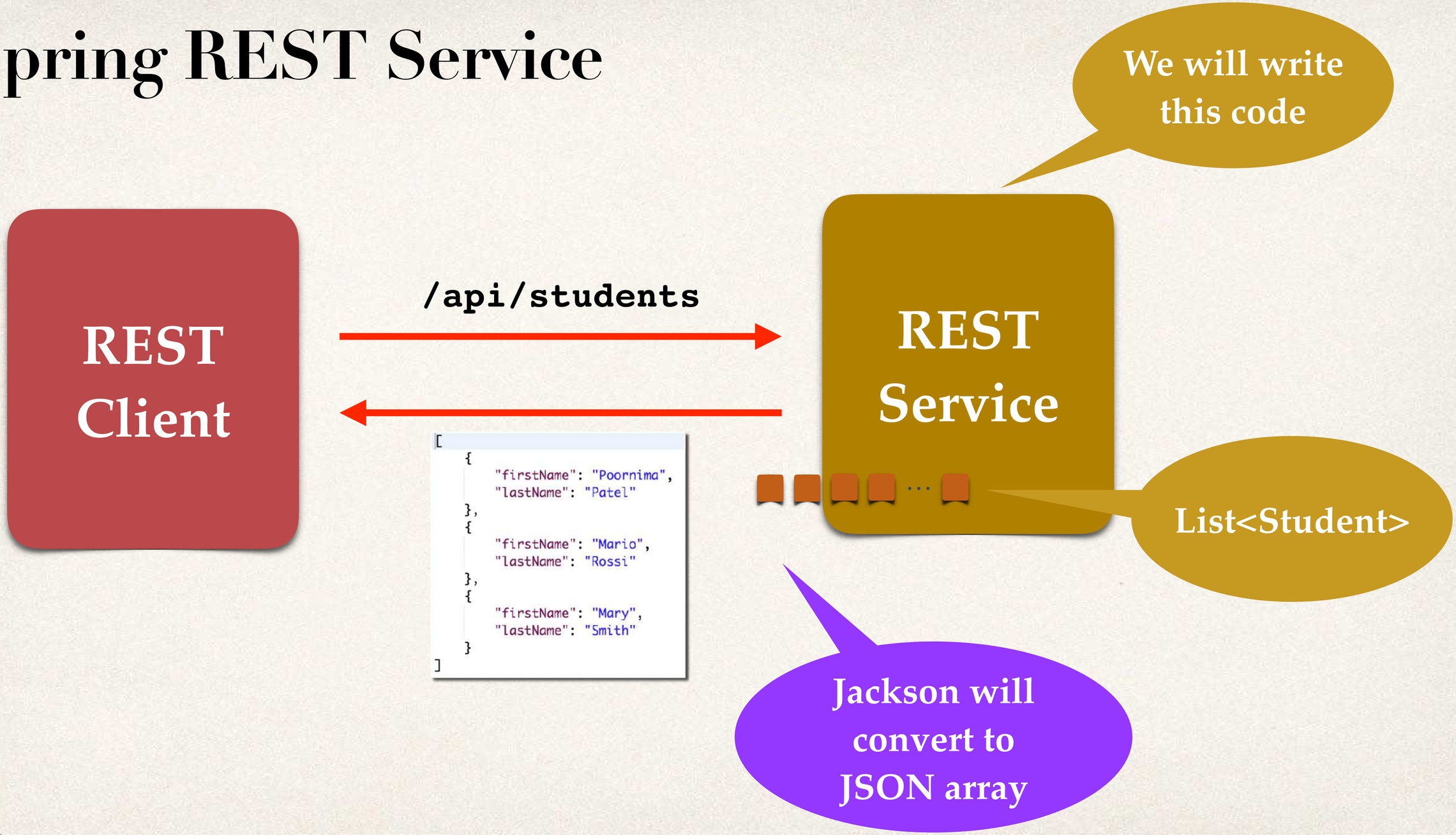
Spring REST Service



Spring REST Service



Spring REST Service



Behind the scenes

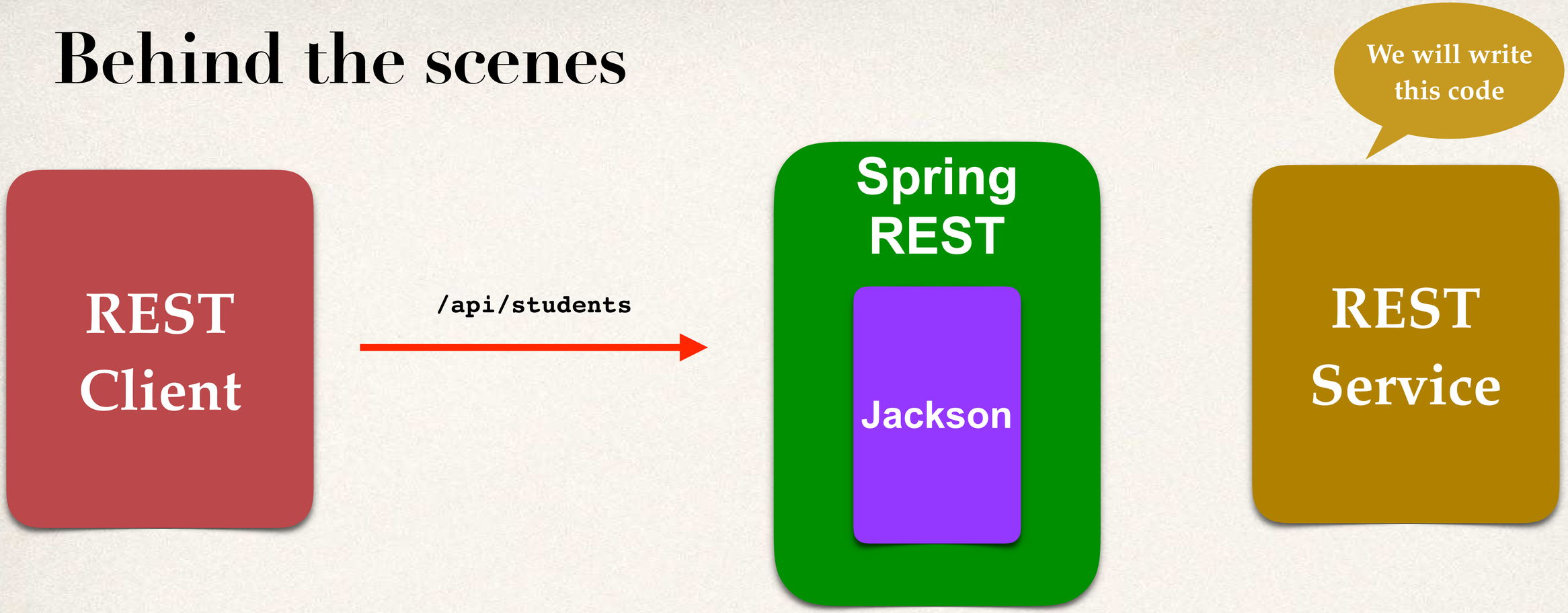
We will write this code

REST Client

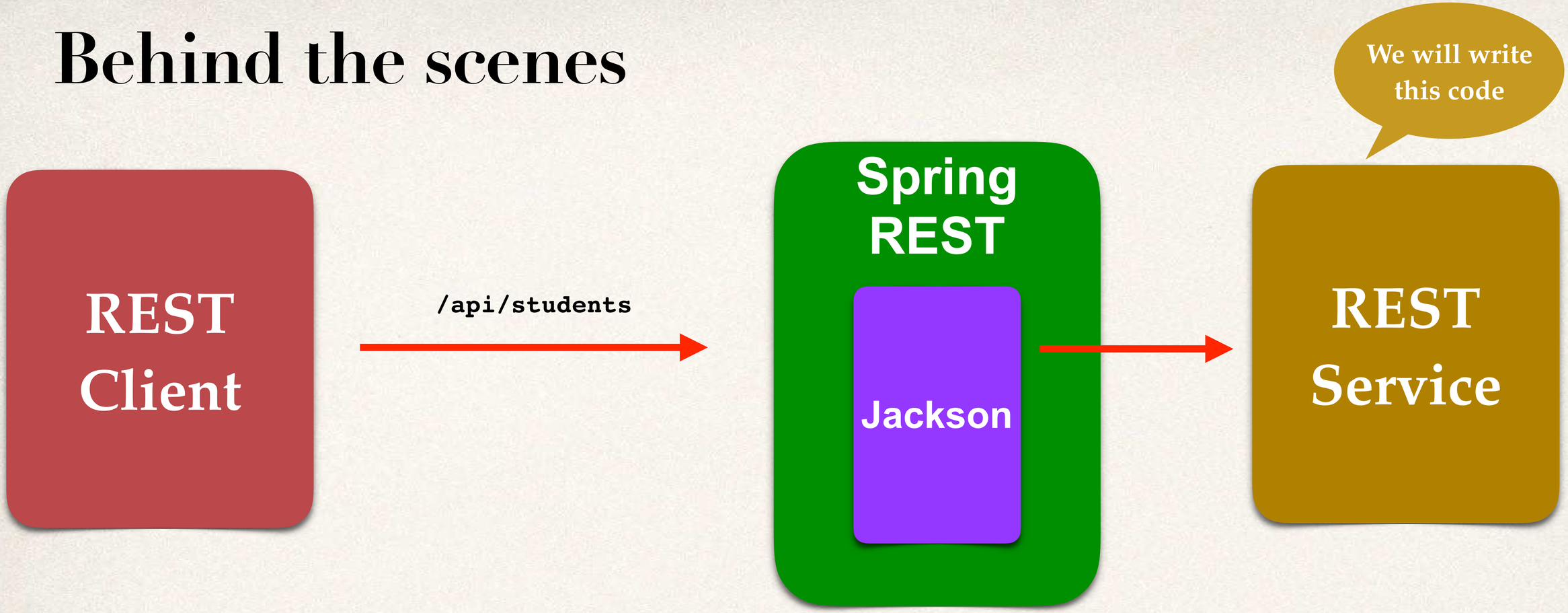
REST Service

Behind the scenes

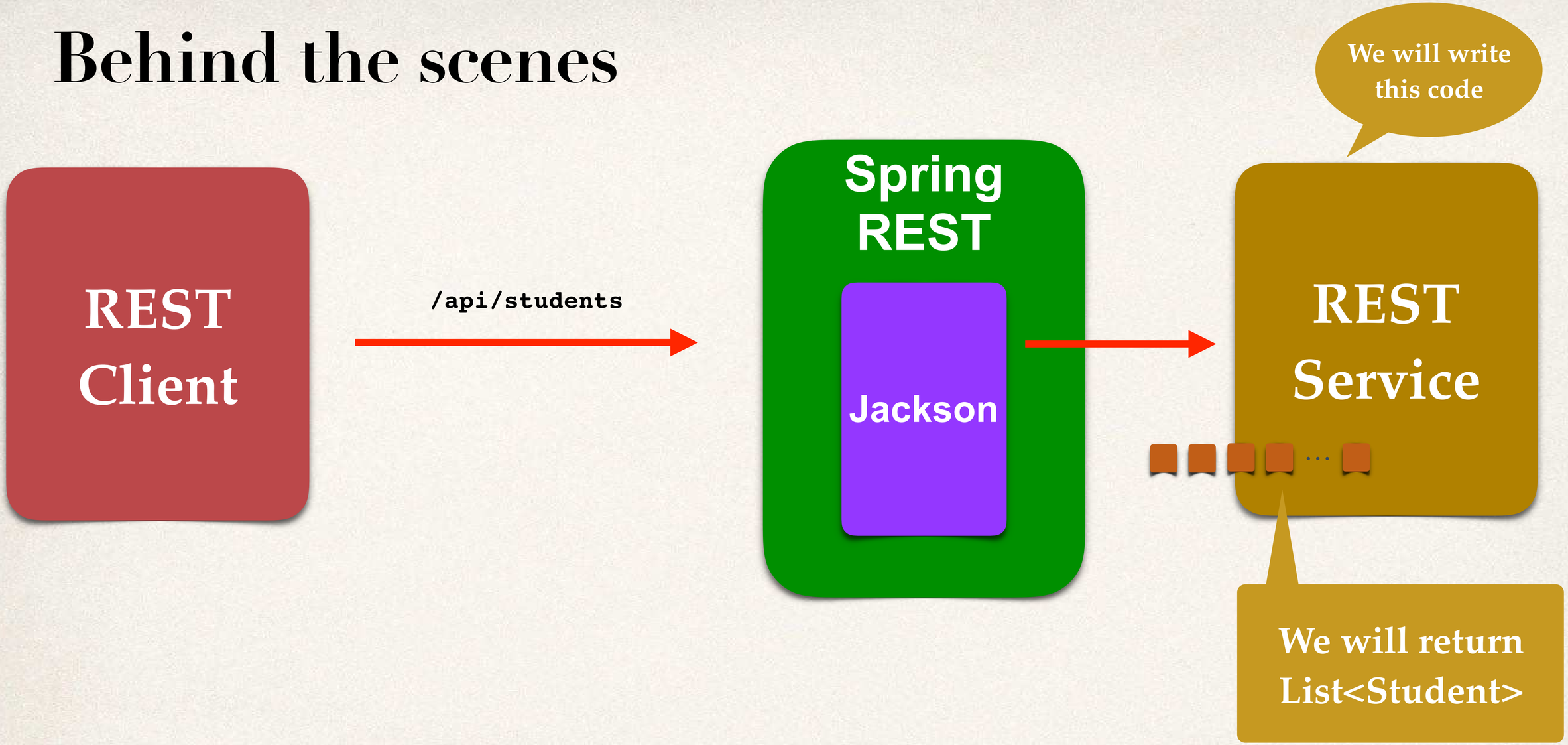
Behind the scenes



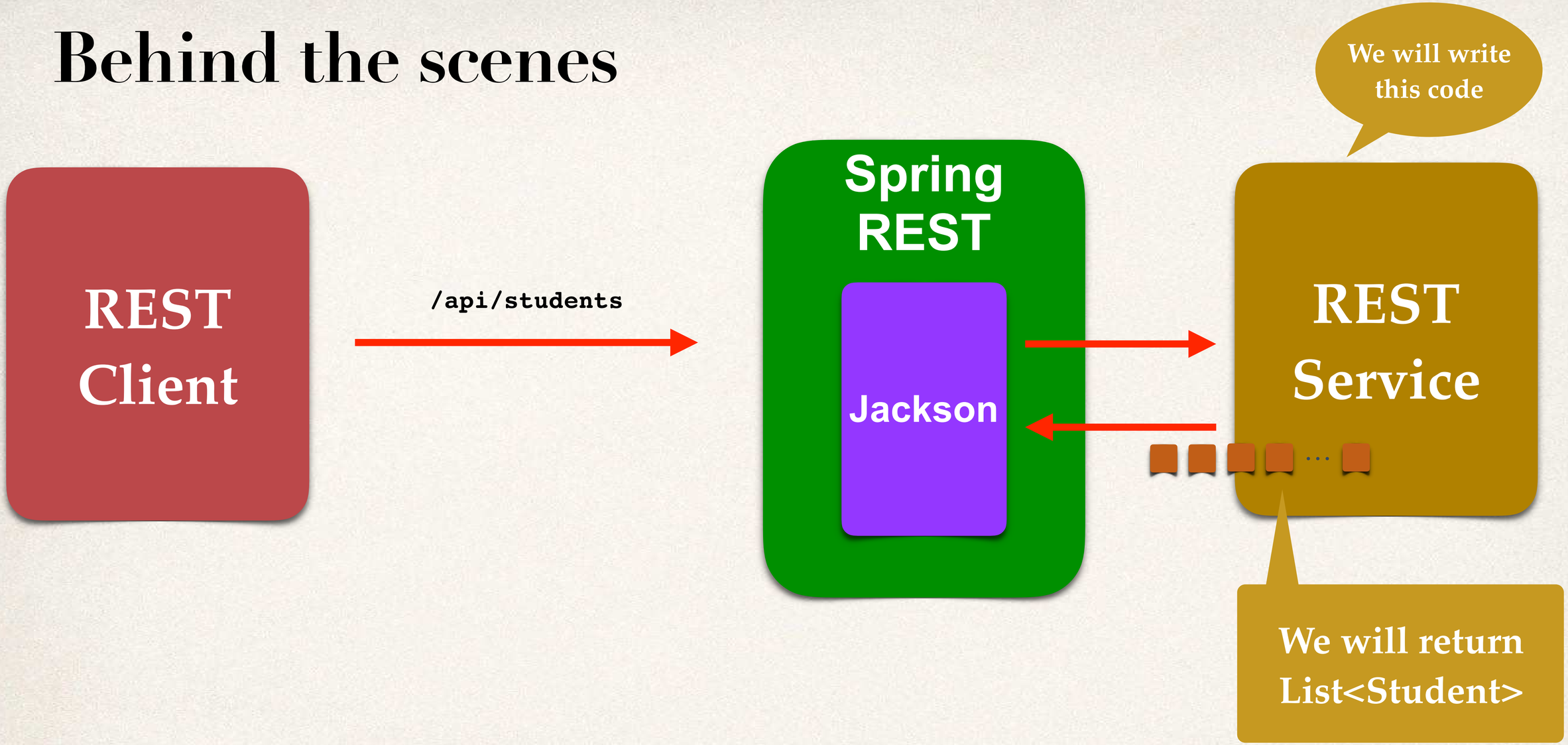
Behind the scenes



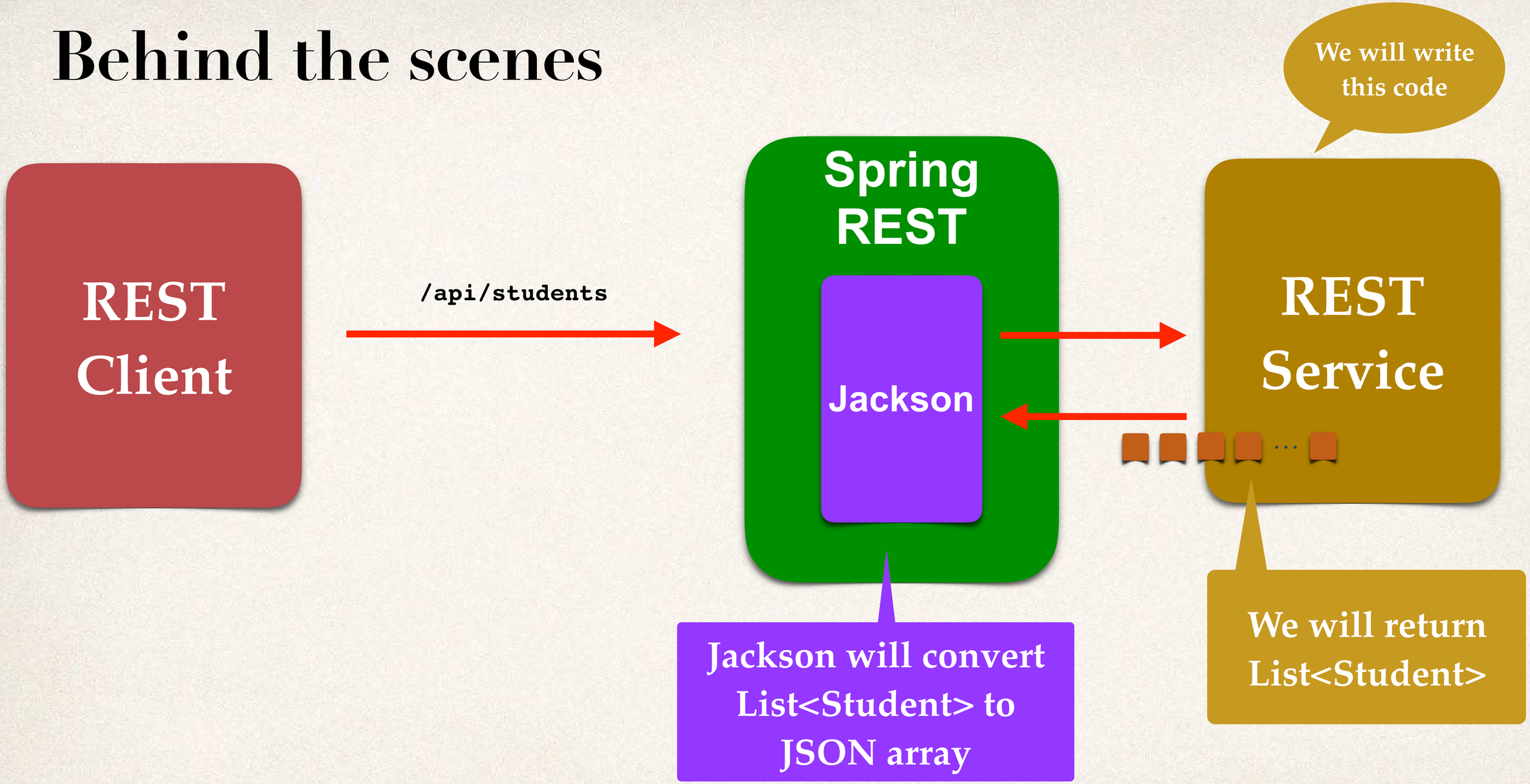
Behind the scenes



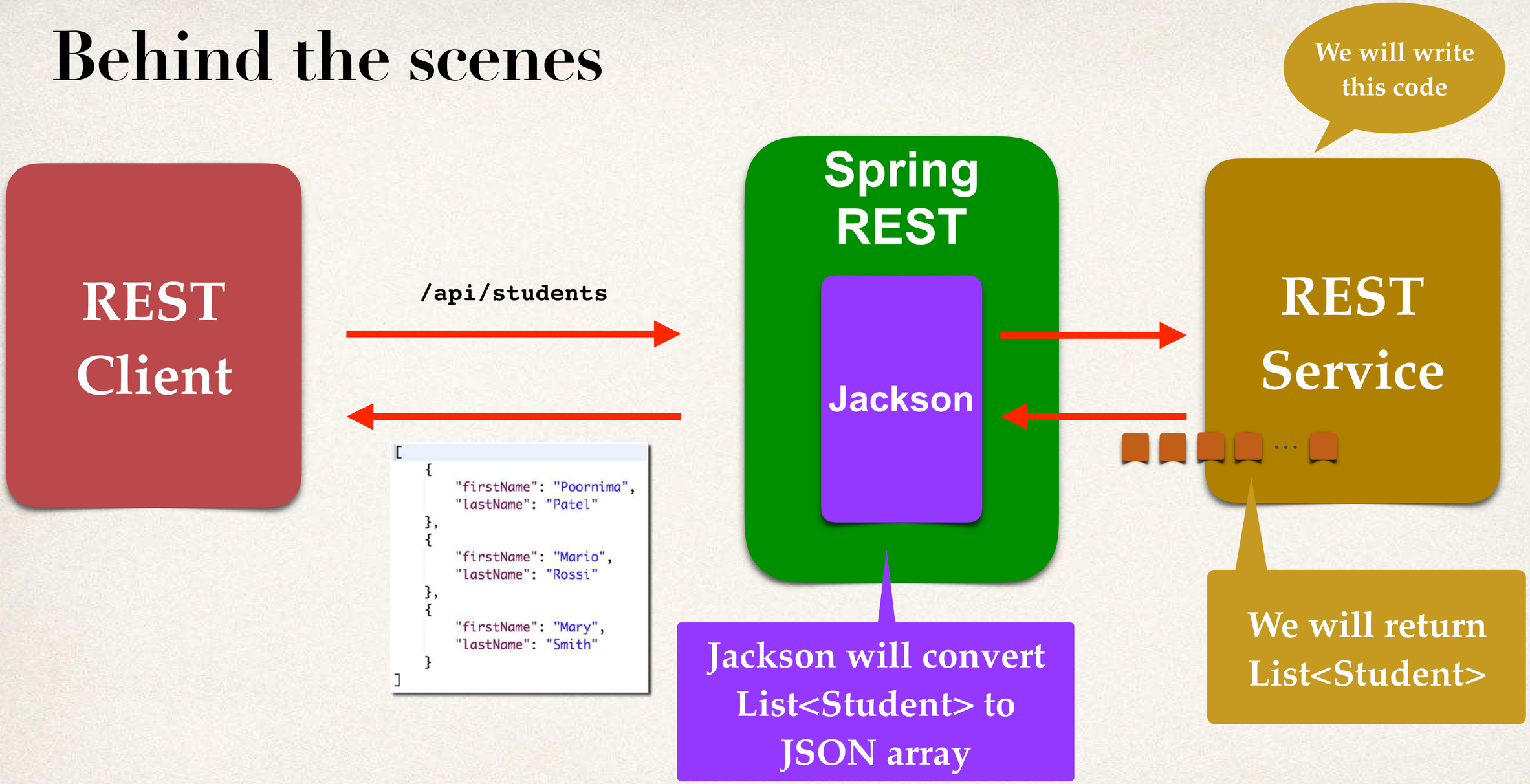
Behind the scenes



Behind the scenes



Behind the scenes



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create Java POJO class for **Student**

Development Process

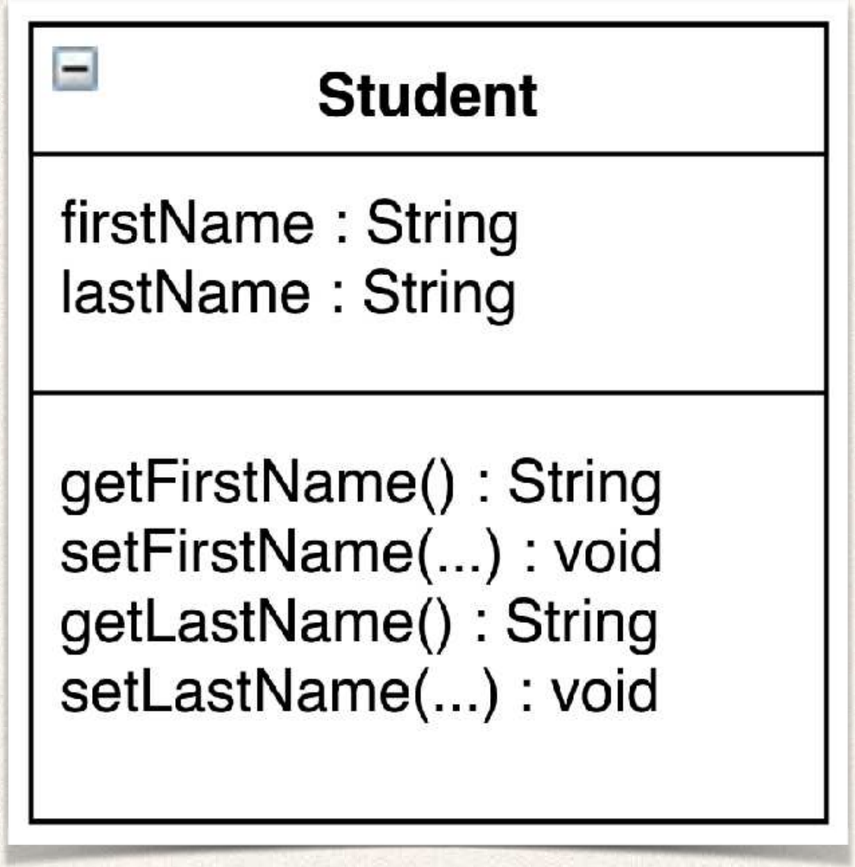
Step-By-Step

1. Create Java POJO class for **Student**

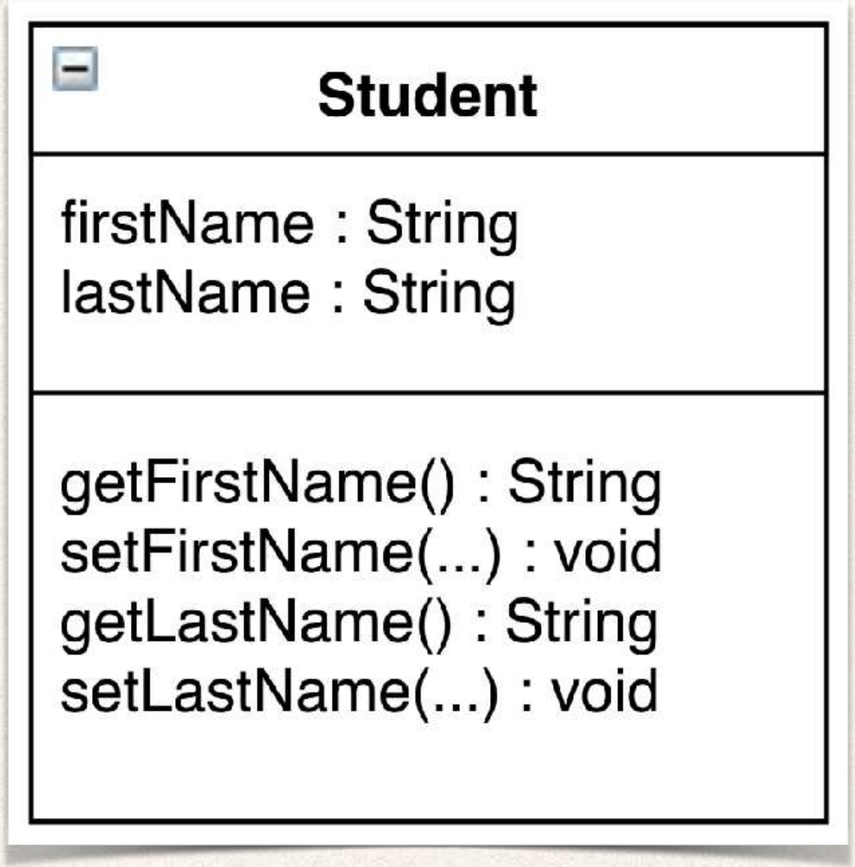
2. Create Spring REST Service using **@RestController**

Step 1: Create Java POJO class for Student

Step 1: Create Java POJO class for Student



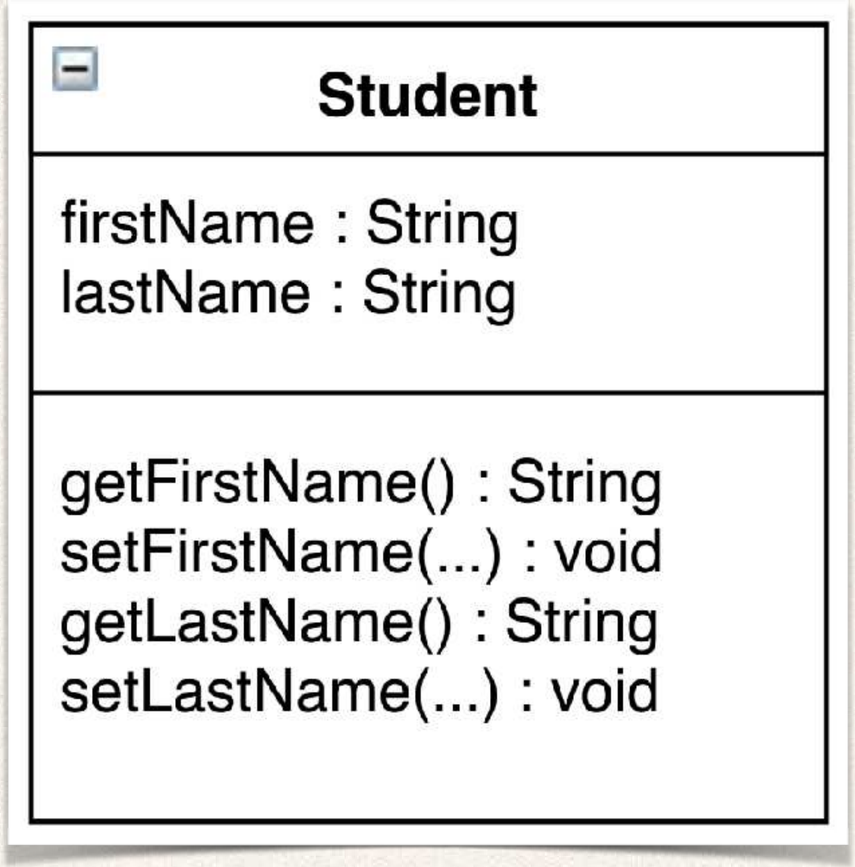
Step 1: Create Java POJO class for Student



File: Student.java

```java
public class Student {

    private String firstName;
    private String lastName;

    public Student() {

    }

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

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

}
```

Step 1: Create Java POJO class for Student



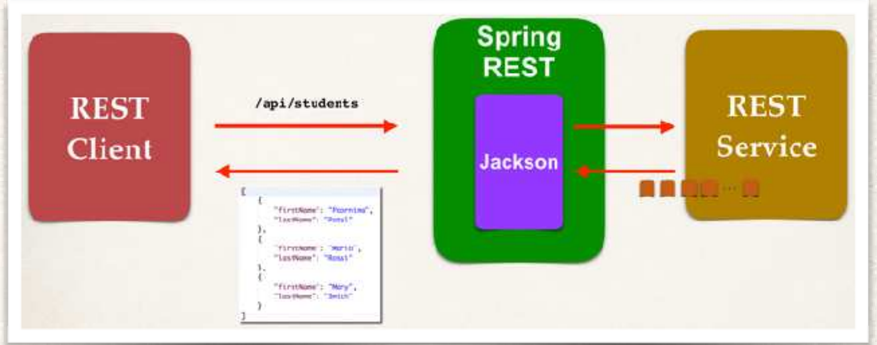
File: Student.java

```
public class Student {

    private String firstName;
    private String lastName;

    public Student() {

    }

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

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

}
```

Fields
Constructors
Getter/Setters

Step 2: Create @RestController

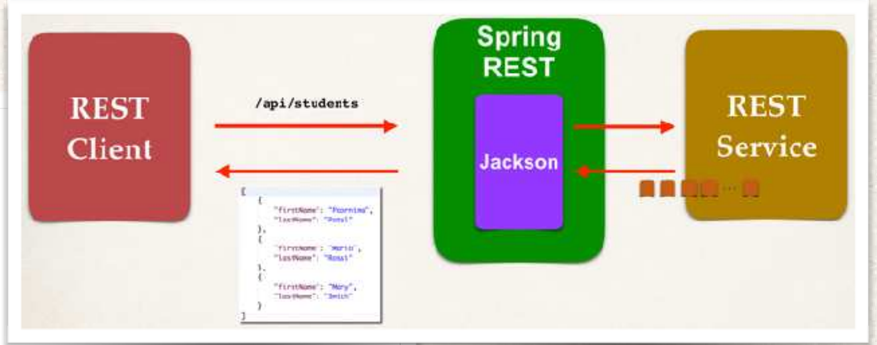
Step 2: Create @RestController



Step 2: Create @RestController

File: StudentRestController.java

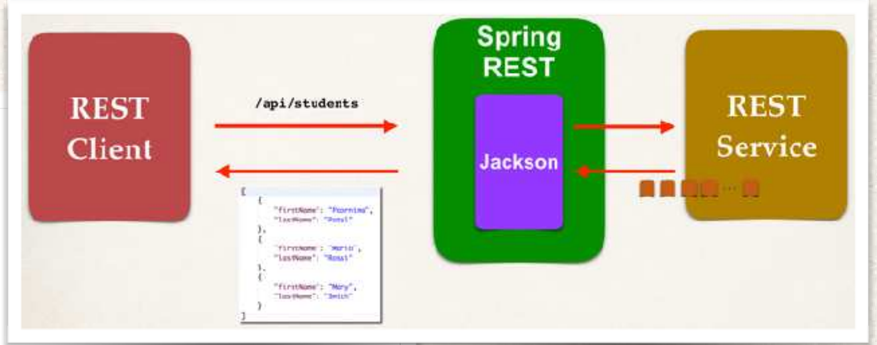
```
@RestController
@RequestMapping("/api")
public class StudentRestController {
```



Step 2: Create @RestController

File: StudentRestController.java

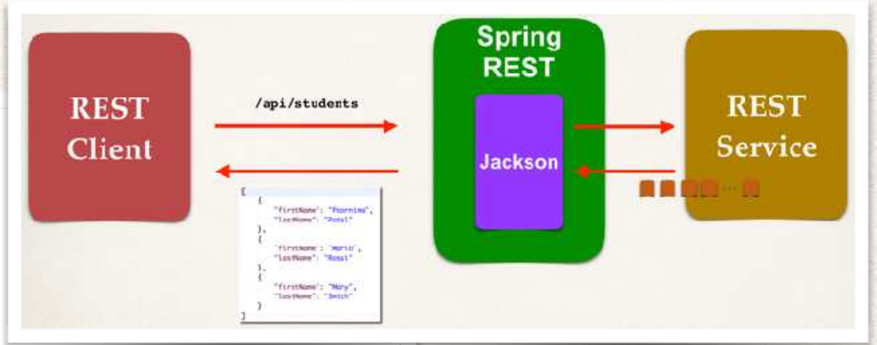
```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students" - return list of students
```



Step 2: Create @RestController

File: StudentRestController.java

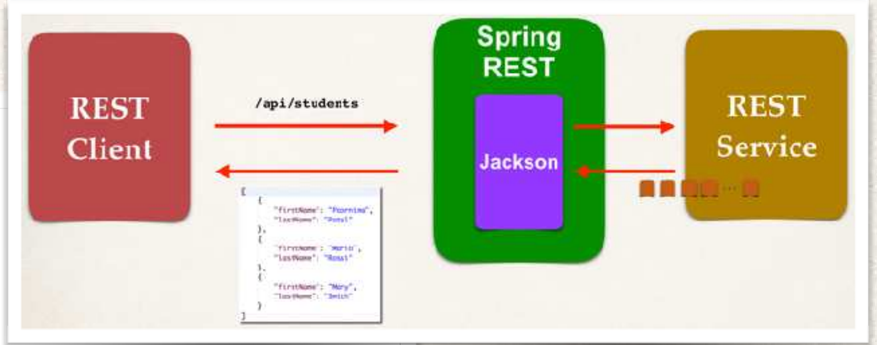
```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students" - return list of students

  @GetMapping("/students")
  public List<Student> getStudents() {
```



Step 2: Create @RestController

File: StudentRestController.java

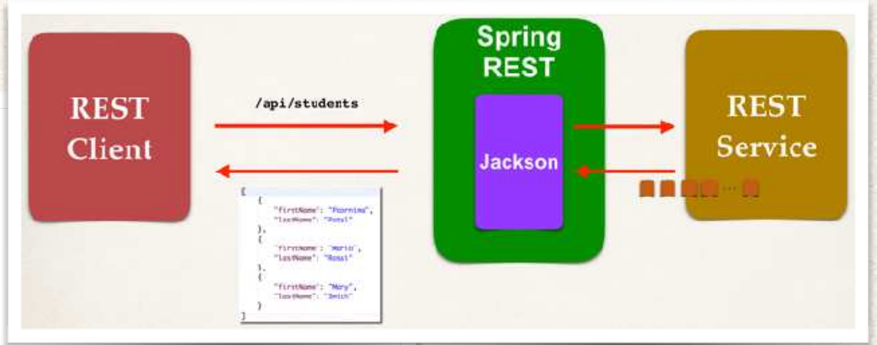
```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students" - return list of students

  @GetMapping("/students")
  public List<Student> getStudents() {

    List<Student> theStudents = new ArrayList<>();
```



Step 2: Create @RestController

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students" - return list of students

  @GetMapping("/students")
  public List<Student> getStudents() {

    List<Student> theStudents = new ArrayList<>();

    theStudents.add(new Student("Poornima", "Patel"));
    theStudents.add(new Student("Mario", "Rossi"));
    theStudents.add(new Student("Mary", "Smith"));
```

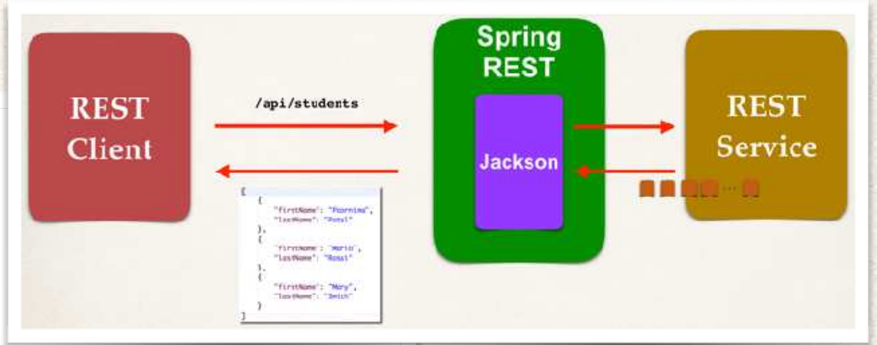


We'll hard code
for now ...
can add DB later ...

Step 2: Create @RestController

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students" - return list of students

  @GetMapping("/students")
  public List<Student> getStudents() {

    List<Student> theStudents = new ArrayList<>();

    theStudents.add(new Student("Poornima", "Patel"));
    theStudents.add(new Student("Mario", "Rossi"));
    theStudents.add(new Student("Mary", "Smith"));

    return theStudents;
  }

}
```



Jackson will convert List<Student> to JSON array

Spring REST - Path Variables

Path Variables

Path Variables

- Retrieve a single student by id

Path Variables

- Retrieve a single student by id

GET `/api/students/{studentId}` *Retrieve a single student*

Path Variables

- Retrieve a single student by id

GET `/api/students/{studentId}` *Retrieve a single student*

Known as a "path variable"

Path Variables

- Retrieve a single student by id

GET

`/api/students/{studentId}` *Retrieve a single student*

`/api/students/0`

`/api/students/1`

`/api/students/2`

Known as a "path variable"

Spring REST Service

Spring REST Service

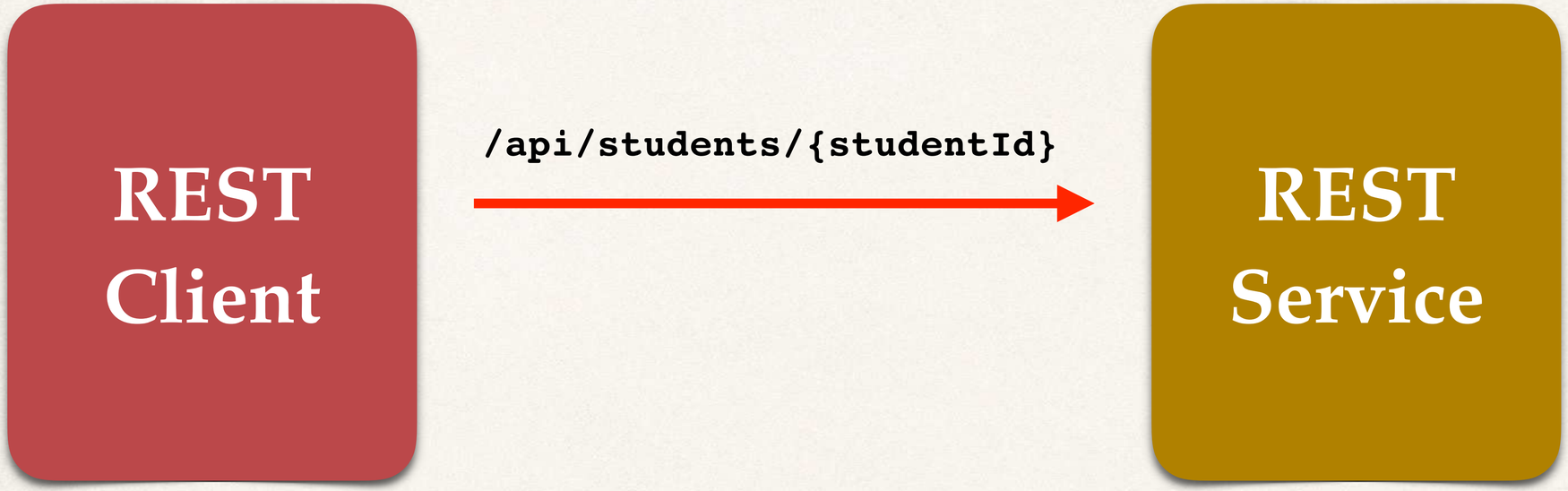
REST
Client

Spring REST Service

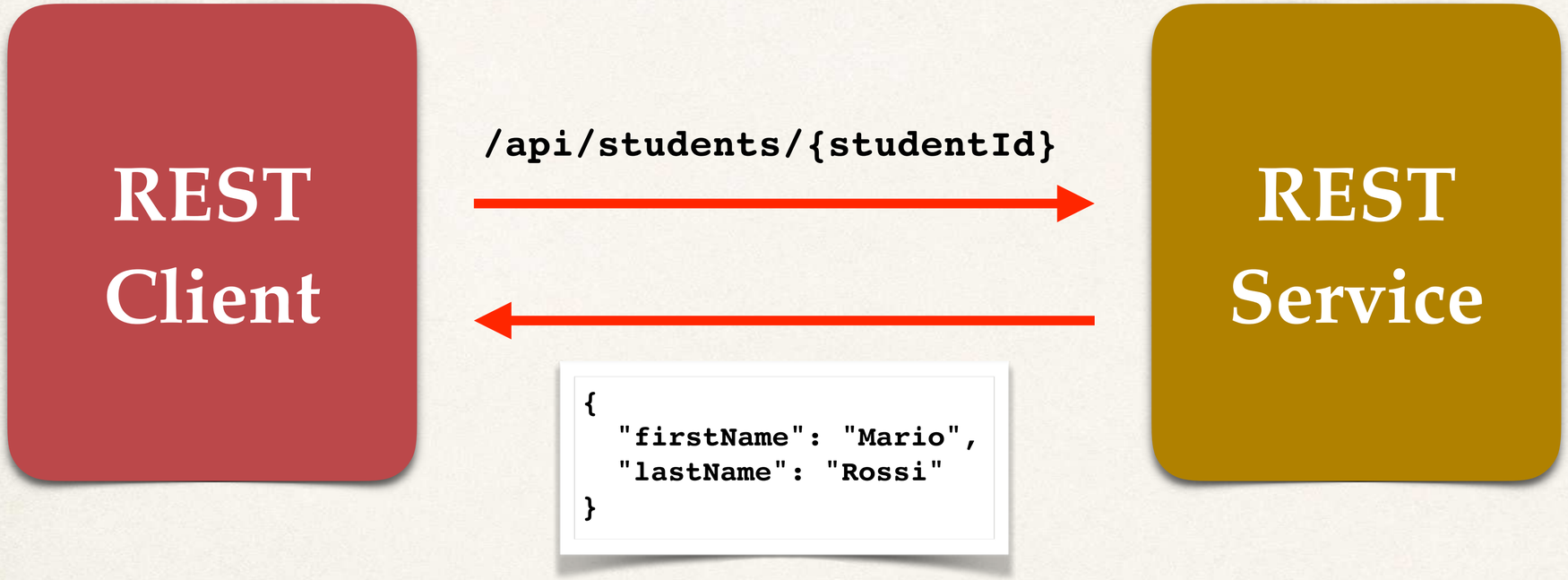
REST Client

REST Service

Spring REST Service



Spring REST Service



Spring REST Service

We will write this code

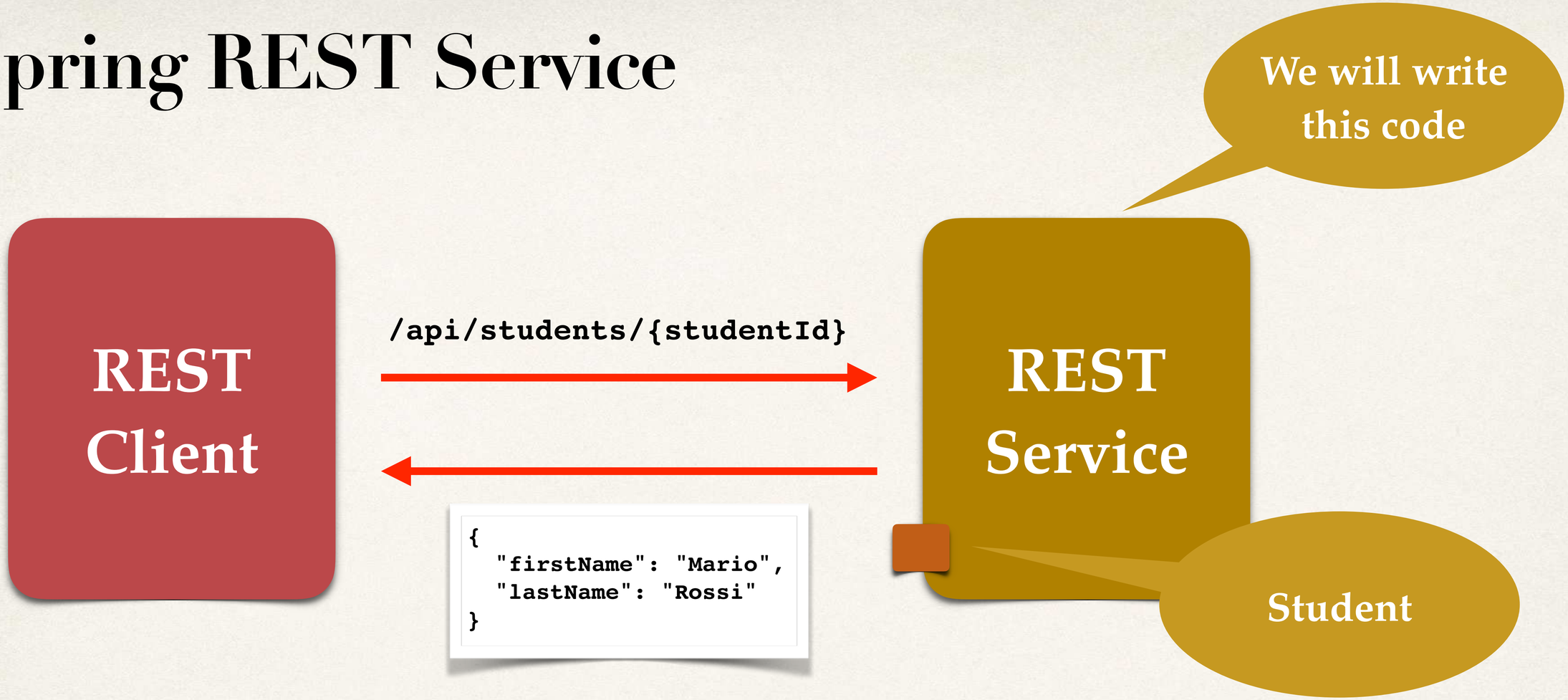
REST Client

/api/students/{studentId}

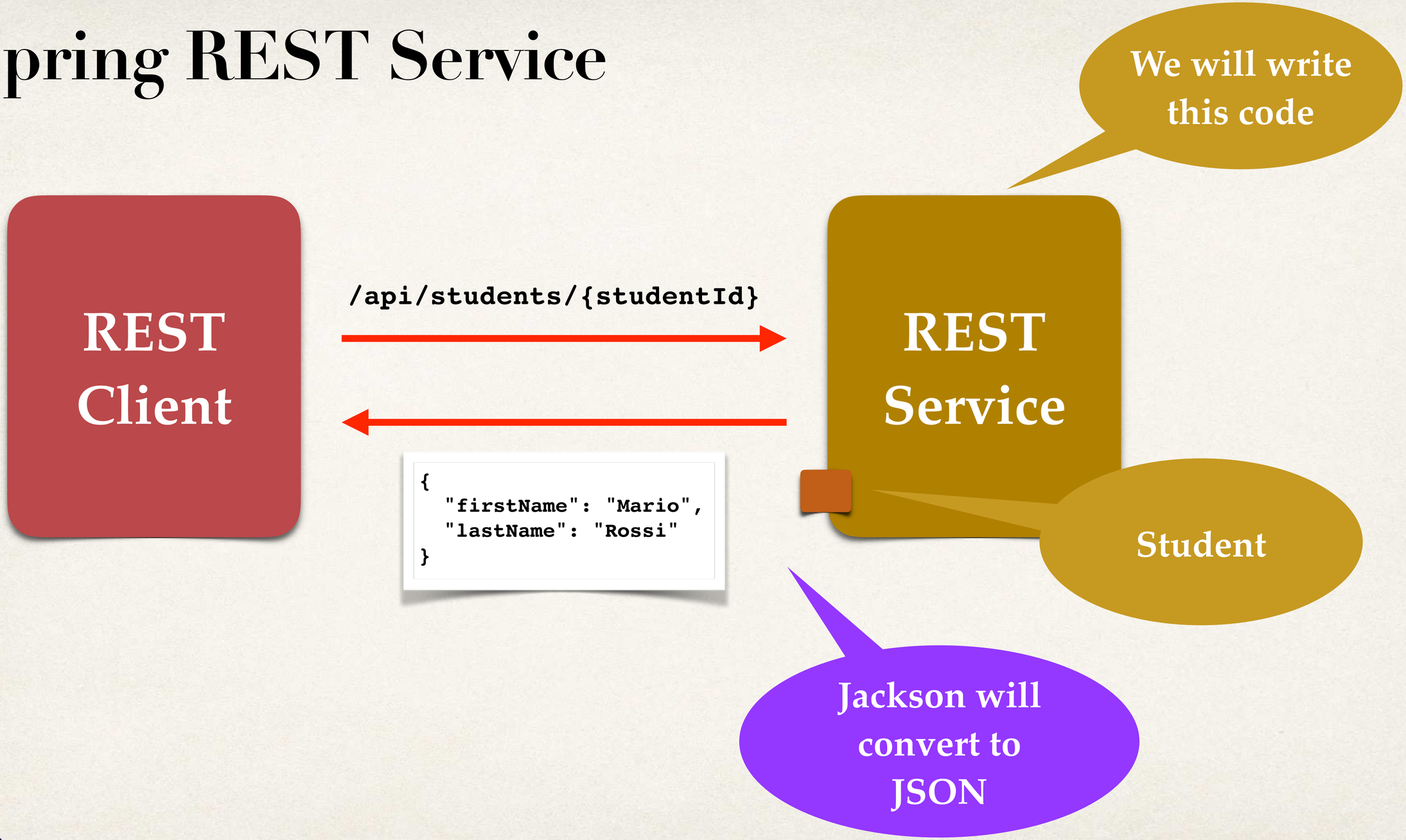
REST Service

```
{
  "firstName": "Mario",
  "lastName": "Rossi"
}
```

Spring REST Service



Spring REST Service



Behind the scenes

We will write this code

REST Client

REST Service

Behind the scenes

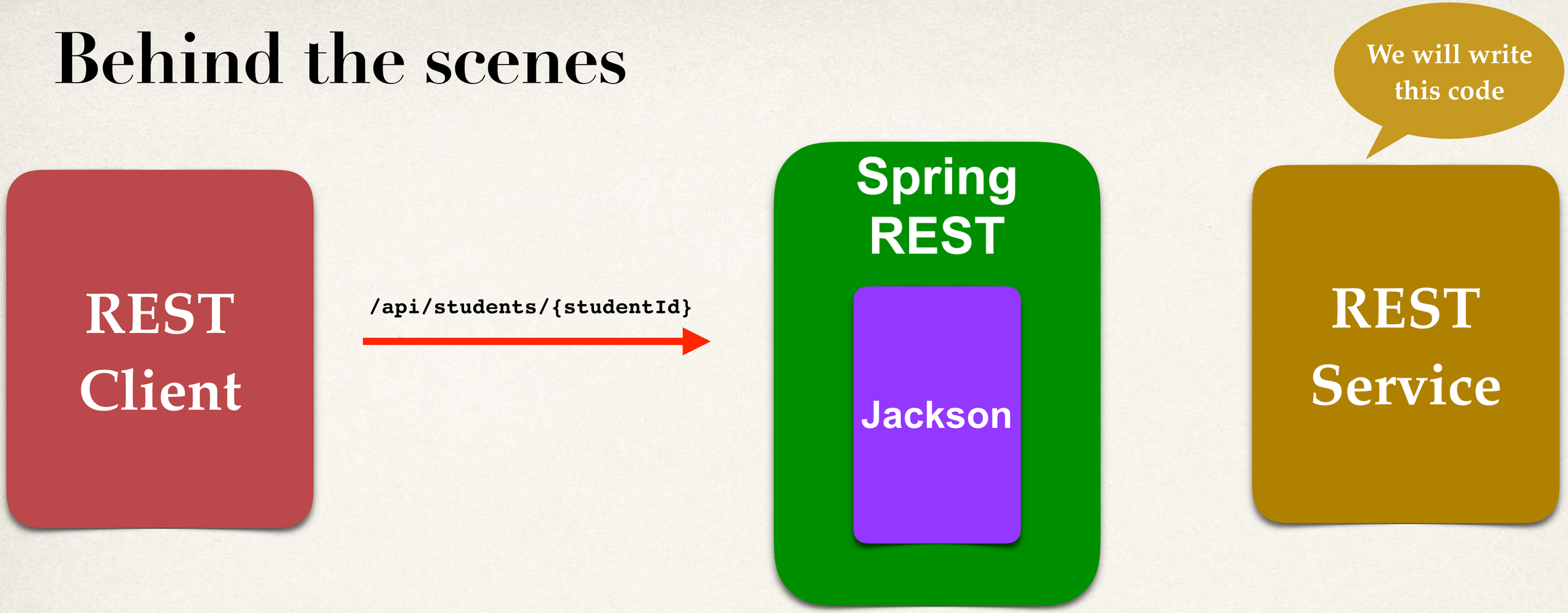
We will write this code

REST Client

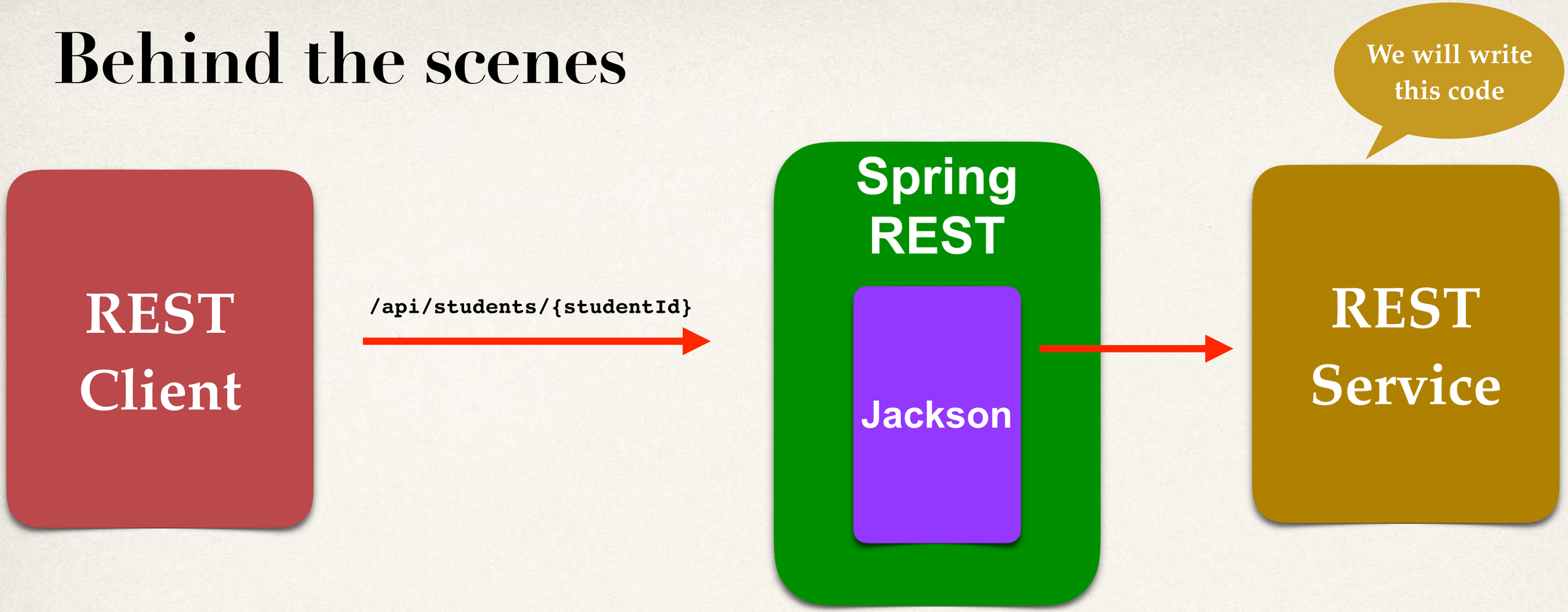
`/api/students/{studentId}`

REST Service

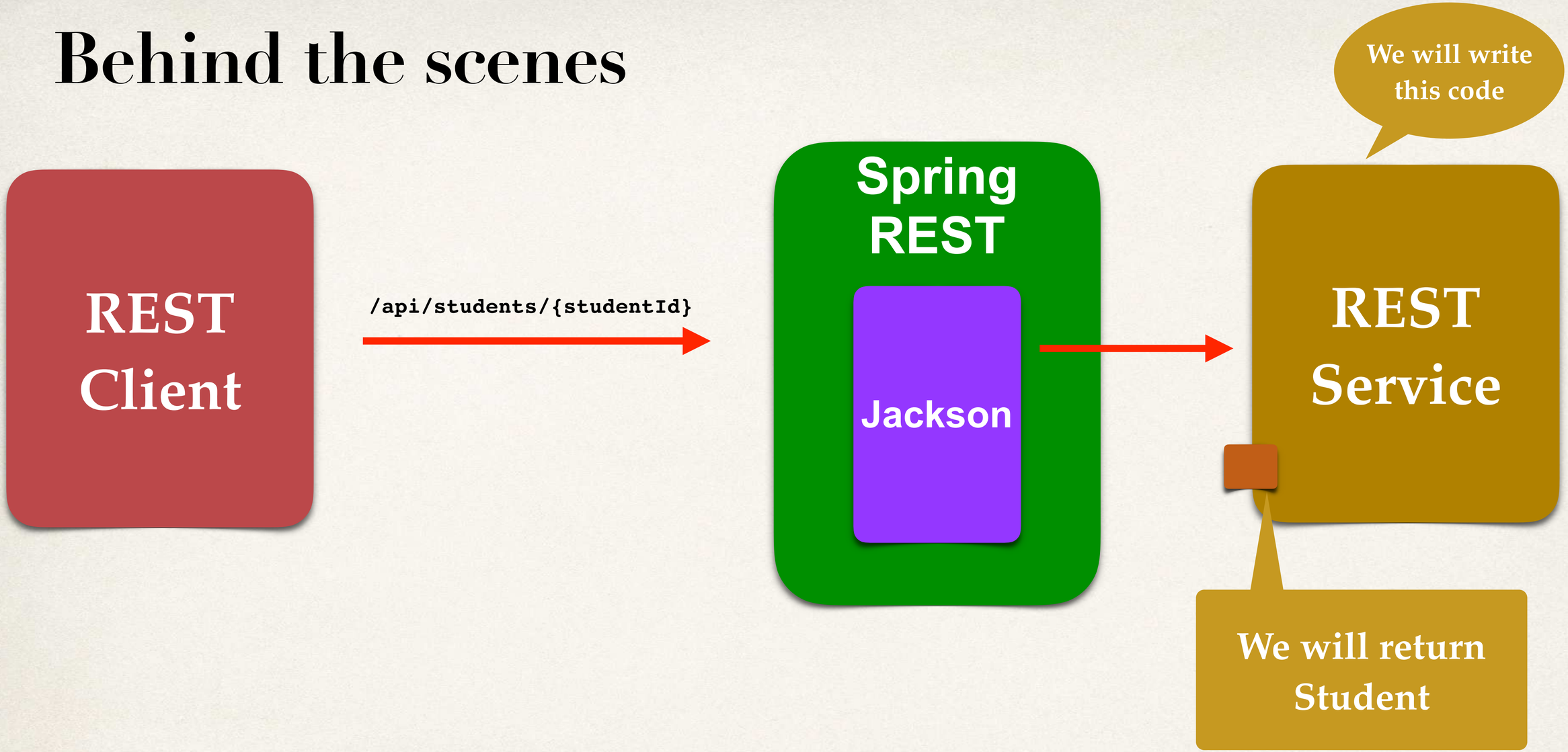
Behind the scenes



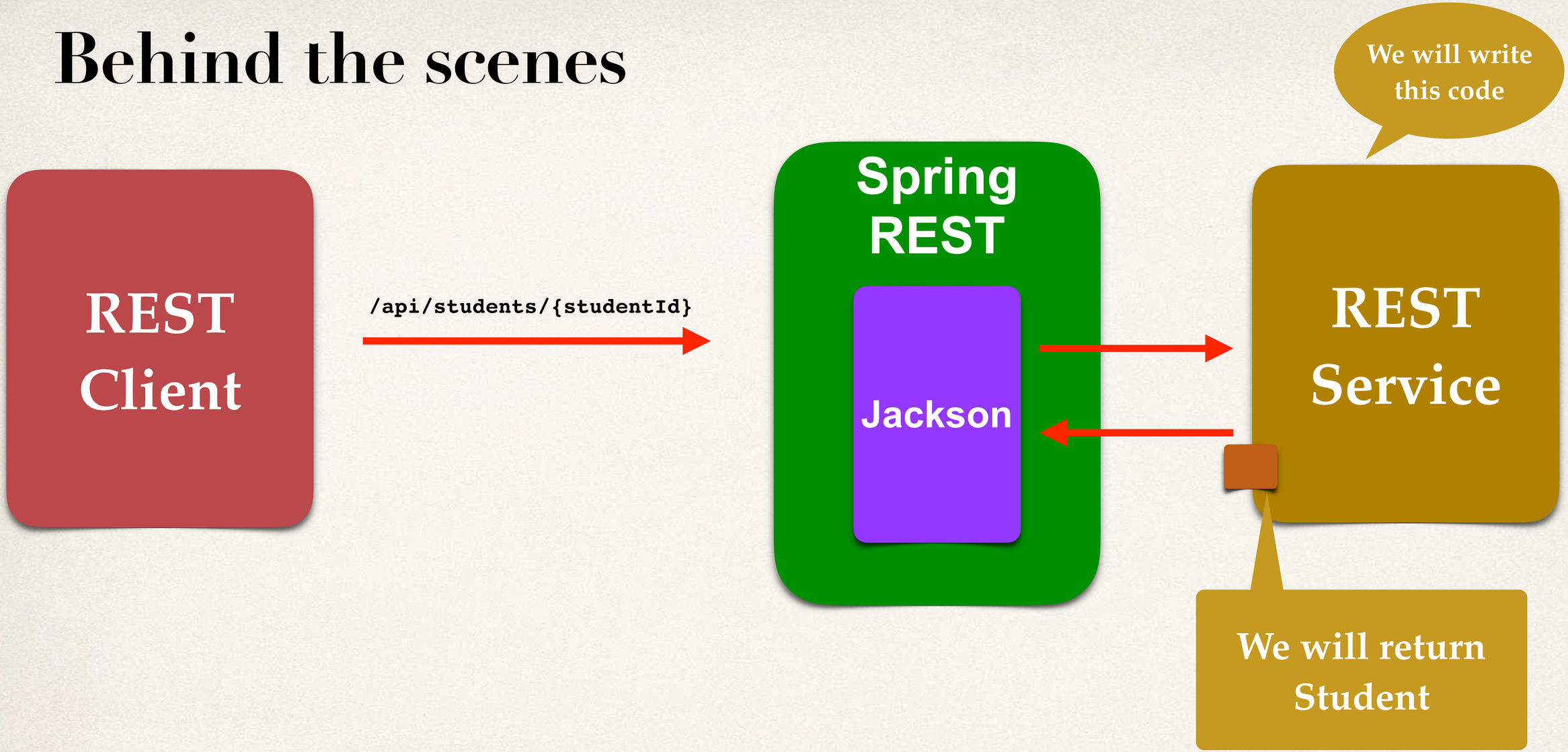
Behind the scenes



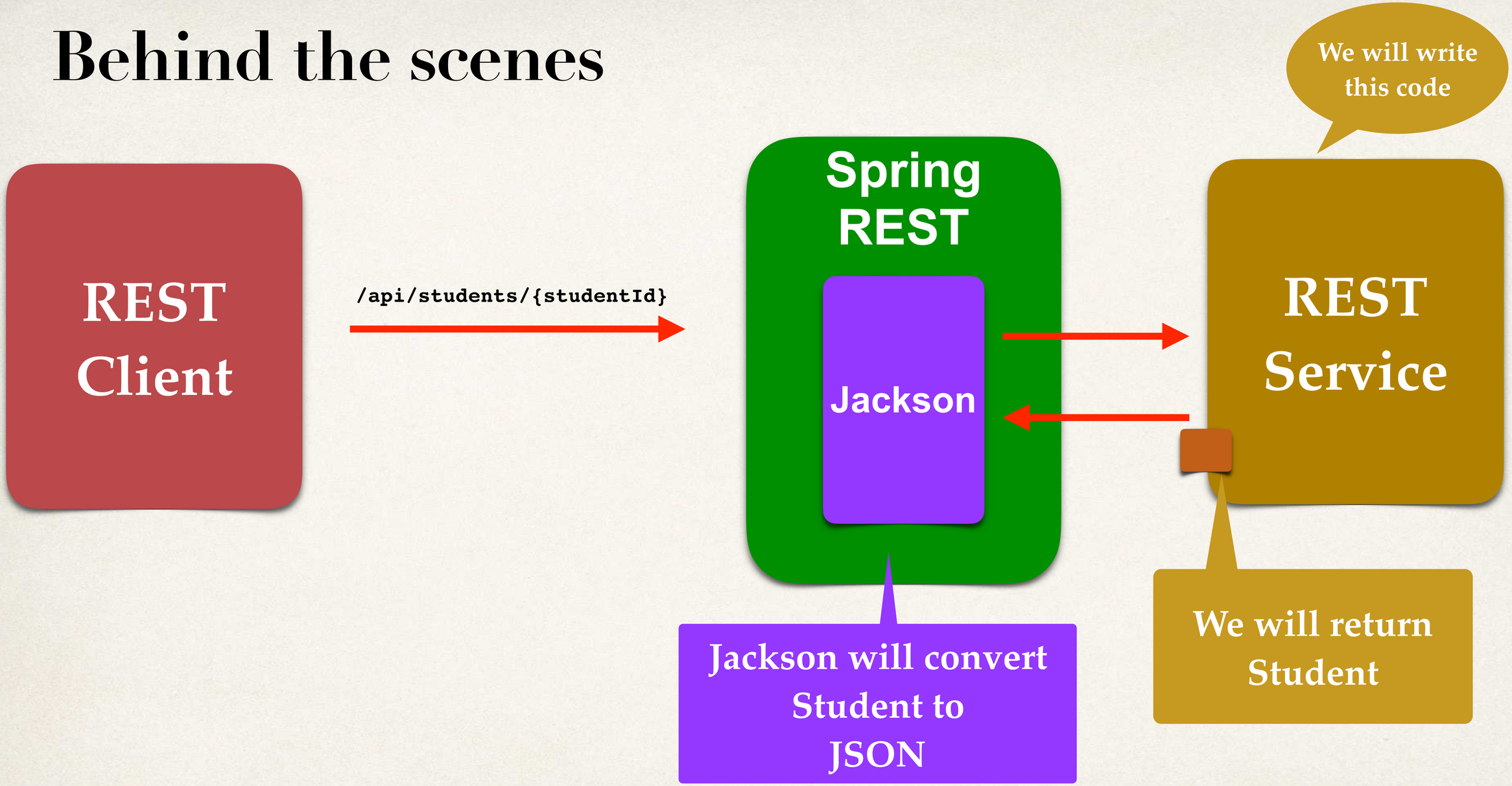
Behind the scenes



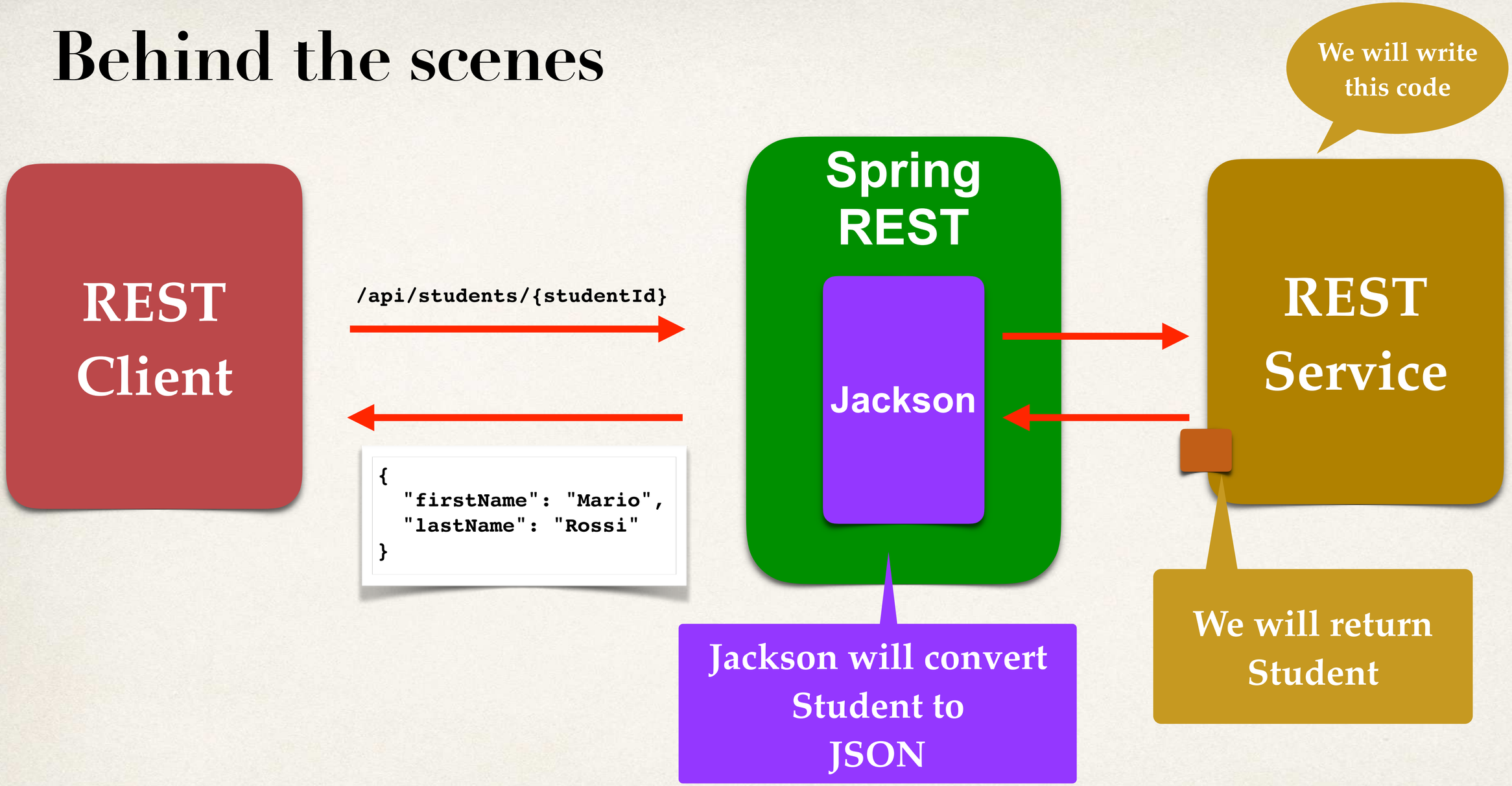
Behind the scenes



Behind the scenes



Behind the scenes



Development Process

Step-By-Step

Development Process

Step-By-Step

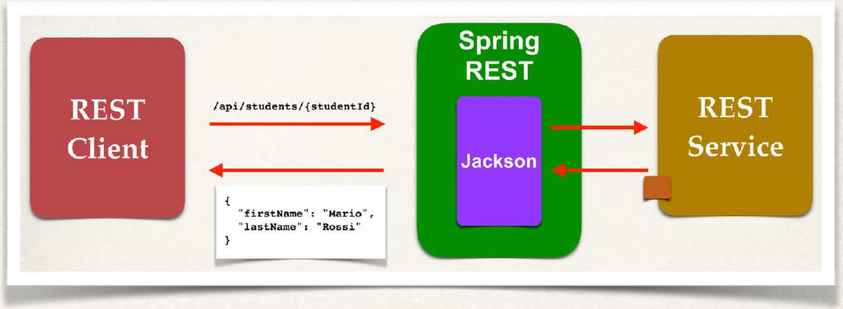
1. Add request mapping to Spring REST Service

Development Process

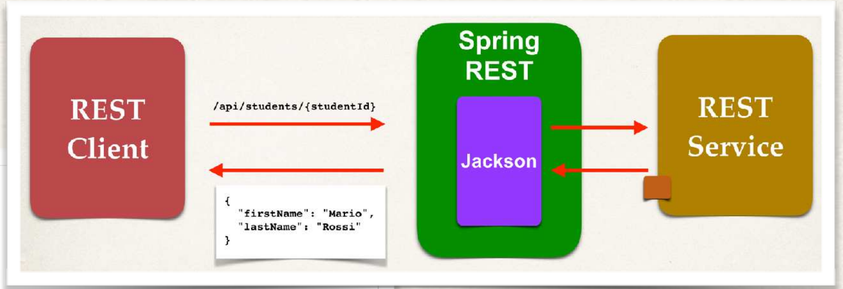
Step-By-Step

1. Add request mapping to Spring REST Service
   - Bind path variable to method parameter using `@PathVariable`

Step 1: Add Request Mapping



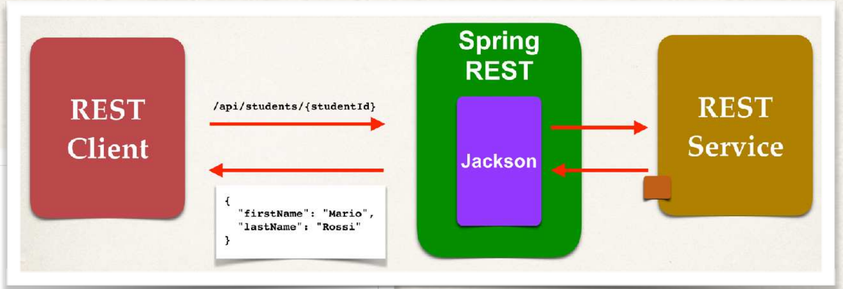
Step 1: Add Request Mapping



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
```

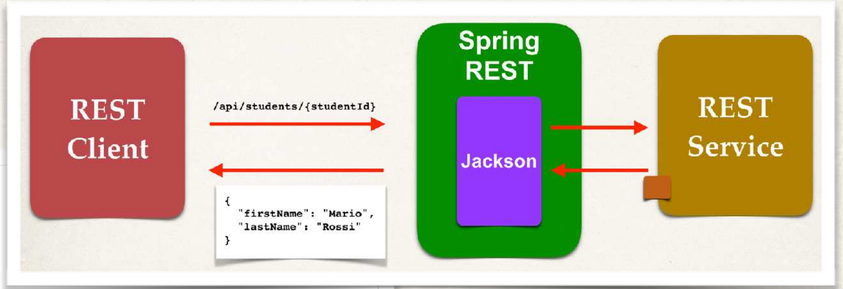
Step 1: Add Request Mapping



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index
```

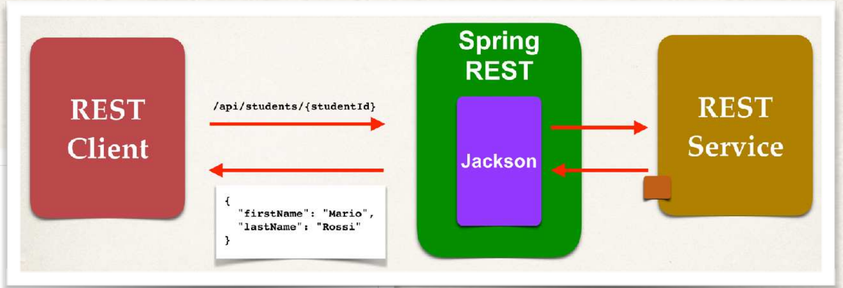
Step 1: Add Request Mapping



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {
```

Step 1: Add Request Mapping

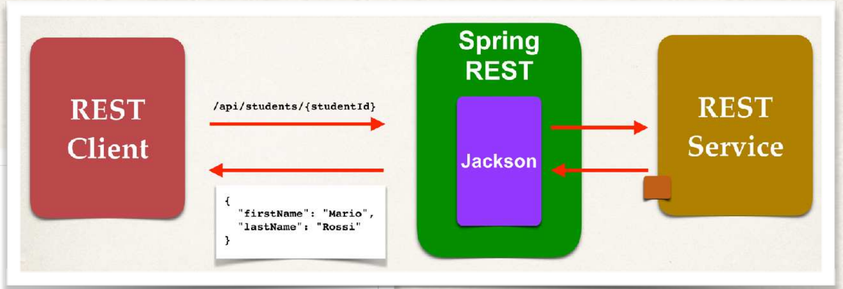


File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {
```

Bind the path variable
(by default, must match)

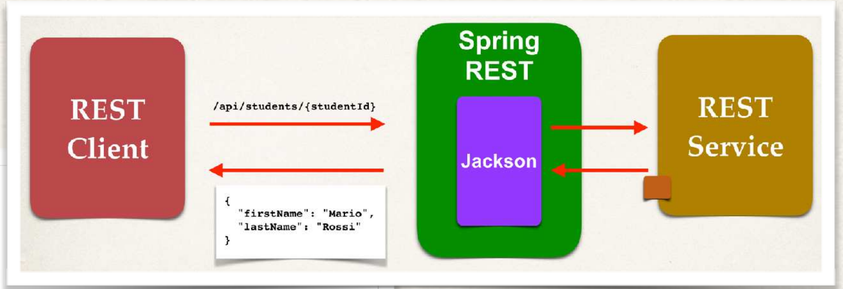
Step 1: Add Request Mapping



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {

    List<Student> theStudents = new ArrayList<>();
```

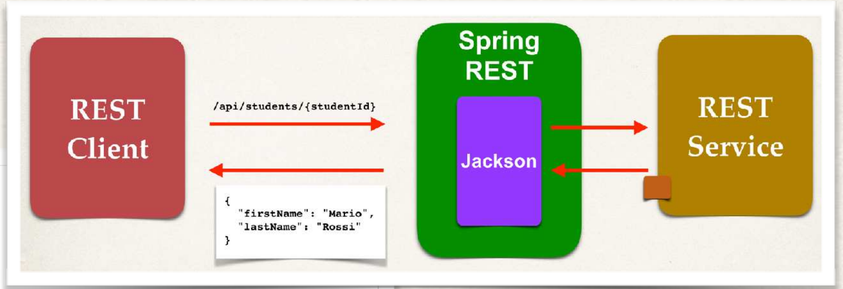
Step 1: Add Request Mapping



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {

    List<Student> theStudents = new ArrayList<>();

    // populate theStudents
    …
```

Step 1: Add Request Mapping



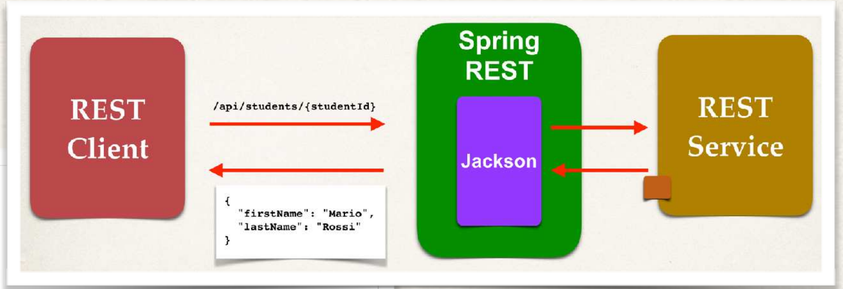
File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariab

    List<Student> theStudents = new Arr

    // populate theStudents
    …


    return theStudents.get(studentId);
  }
```

Keep it simple,
just index into the list

We'll do fancy DB stuff later

Step 1: Add Request Mapping



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  // define endpoint for "/students/{studentId}" - return student at index

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {

    List<Student> theStudents = new ArrayList<>();

    // populate theStudents
    …


    return theStudents.get(studentId);
  }

}
```

Jackson will convert Student to JSON

Spring REST - Exception Handling

Remember our problem?

Remember our problem?

- Bad student id of 9999 …

Remember our problem?

- Bad student id of 9999 …

Remember our problem?

- Bad student id of 9999 …

Scary HTML error page

HTTP Status 500 – Internal Server Error

Type Exception Report

Message Request processing failed; nested exception is java.lang.IndexOutOfBoundsException: Index 9999 out-of-bounds for length 3

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
org.springframework.web.util.NestedServletException: Request processing f
        org.springframework.web.servlet.FrameworkServlet.processRequest(F
        org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkS
        javax.servlet.http.HttpServlet.service(HttpServlet.java:634)
        org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:851)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
```

Root cause

We really want this …

We really want this …

- Handle the exception and return error as JSON

We really want this …

- Handle the exception and return error as JSON

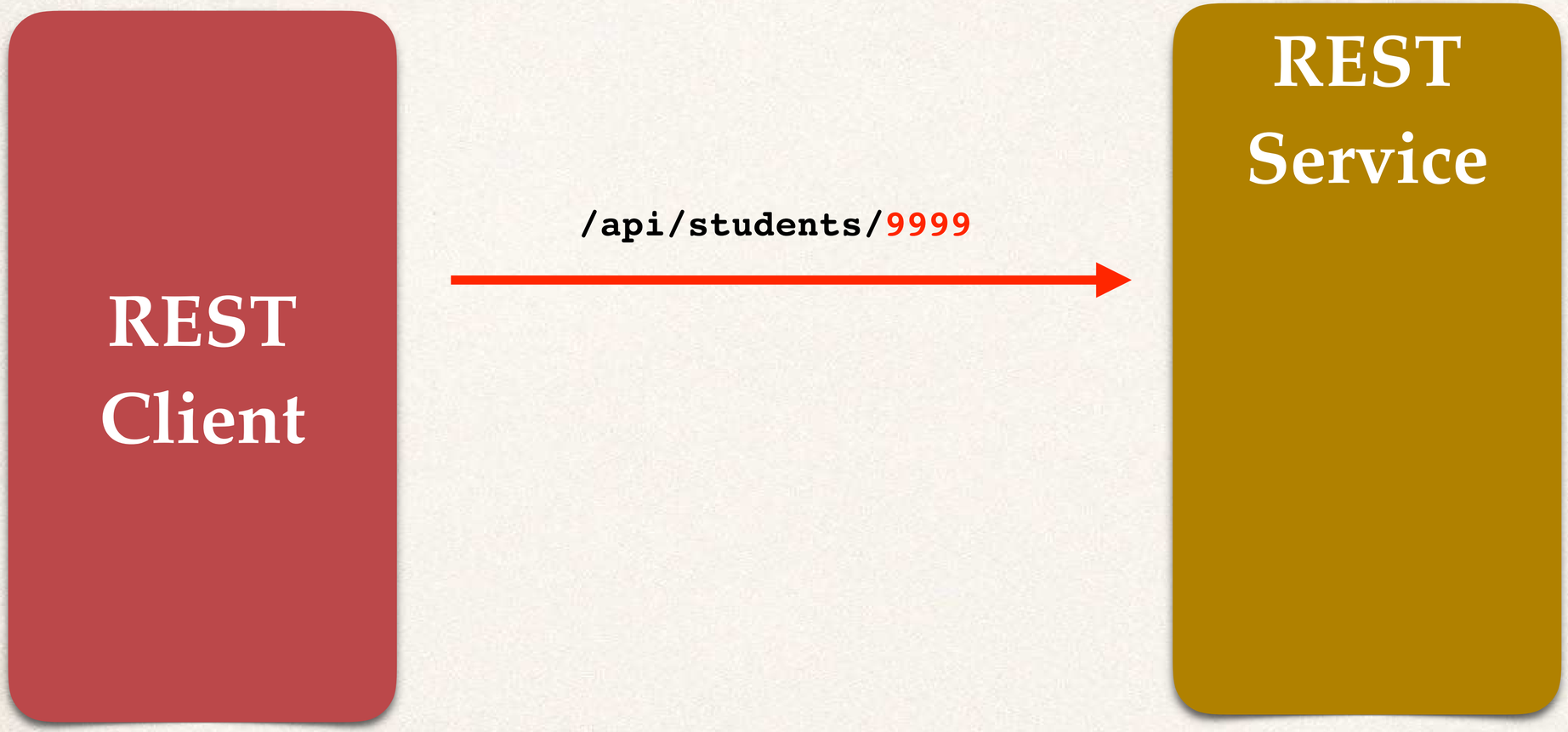
```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

Spring REST Exception Handling

REST Client

REST Service

Spring REST Exception Handling

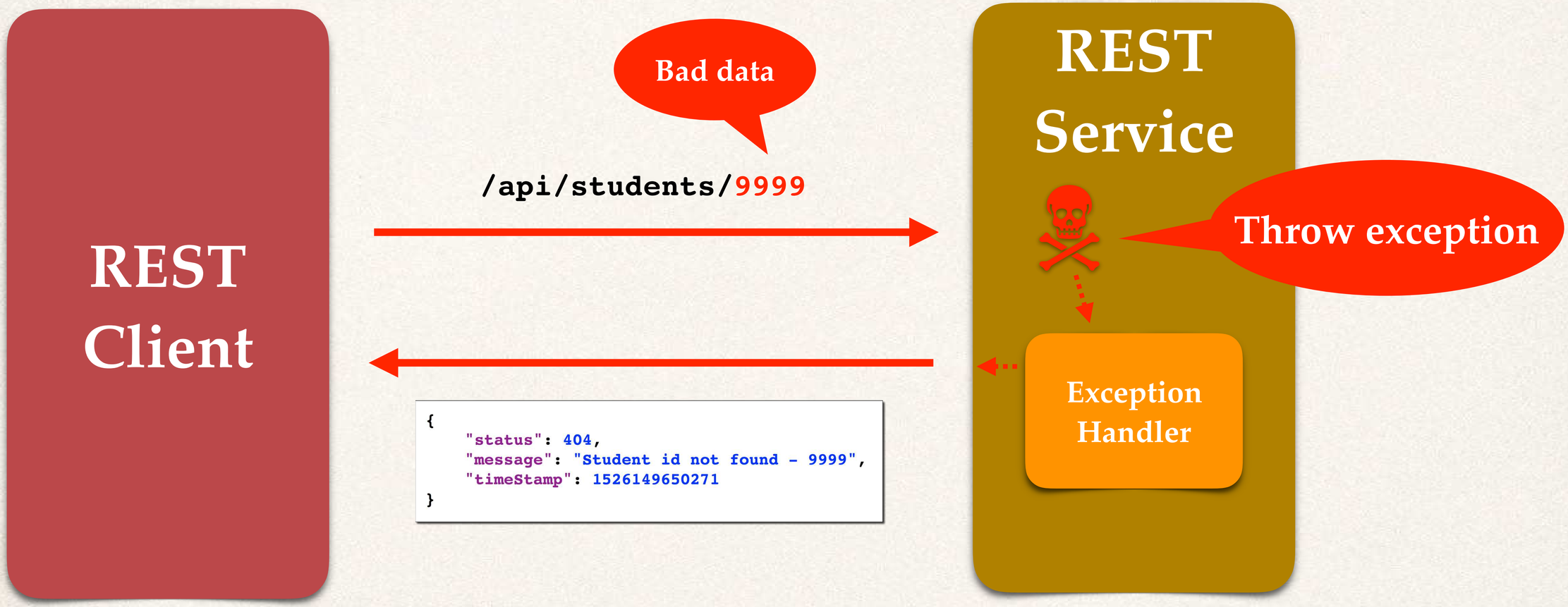


Spring REST Exception Handling

Spring REST Exception Handling

Spring REST Exception Handling

Spring REST Exception Handling



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create a custom error response class

Development Process

Step-By-Step

1. Create a custom error response class

2. Create a custom exception class

Development Process

Step-By-Step

1. Create a custom error response class

2. Create a custom exception class

3. Update REST service to throw exception if student not found

Development Process

Step-By-Step

1. Create a custom error response class

2. Create a custom exception class

3. Update REST service to throw exception if student not found

4. Add an exception handler method using `@ExceptionHandler`

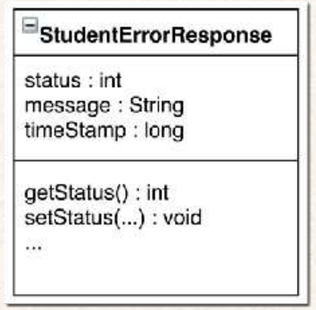
Step 1: Create custom error response class

Step 1: Create custom error response class

- The custom error response class will be sent back to client as JSON

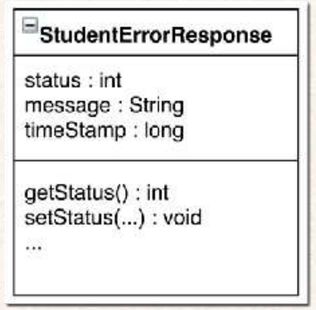
Step 1: Create custom error response class

- The custom error response class will be sent back to client as JSON
- We will define as Java class (POJO)



Step 1: Create custom error response class

- The custom error response class will be sent back to client as JSON
- We will define as Java class (POJO)

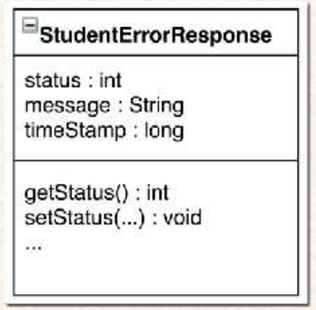


- Jackson will handle converting it to JSON

```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

Step 1: Create custom error response class

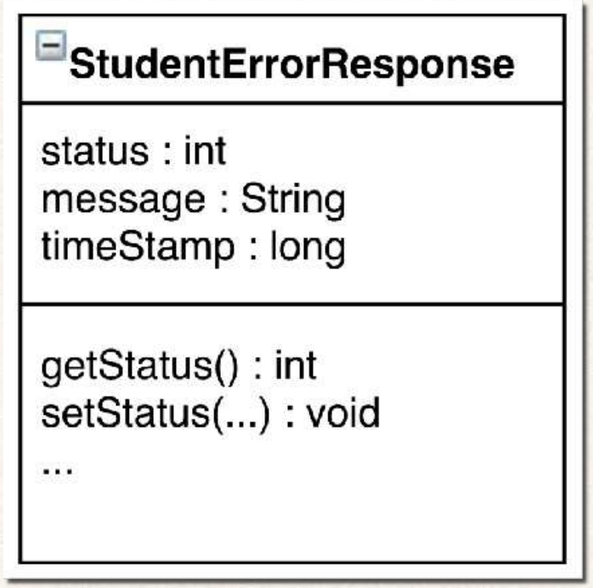
- The custom error response class will be sent back to client as JSON
- We will define as Java class (POJO)
- Jackson will handle converting it to JSON



You can define any custom fields that you want to track

```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

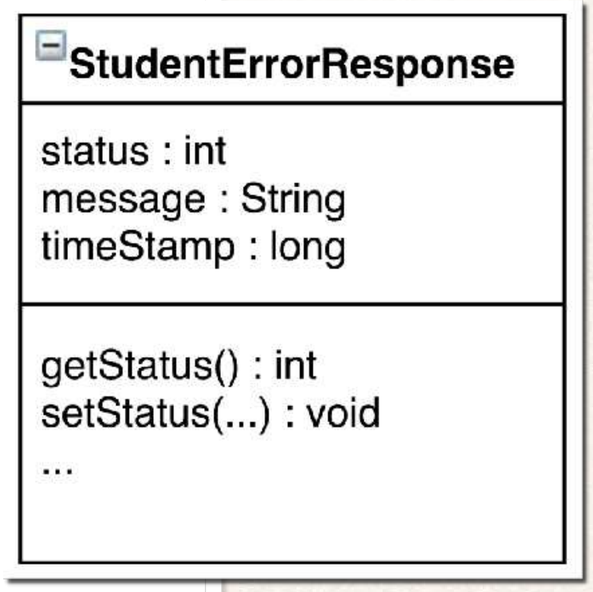
Step 1: Create custom error response class



Step 1: Create custom error response class

File: StudentErrorResponse.java

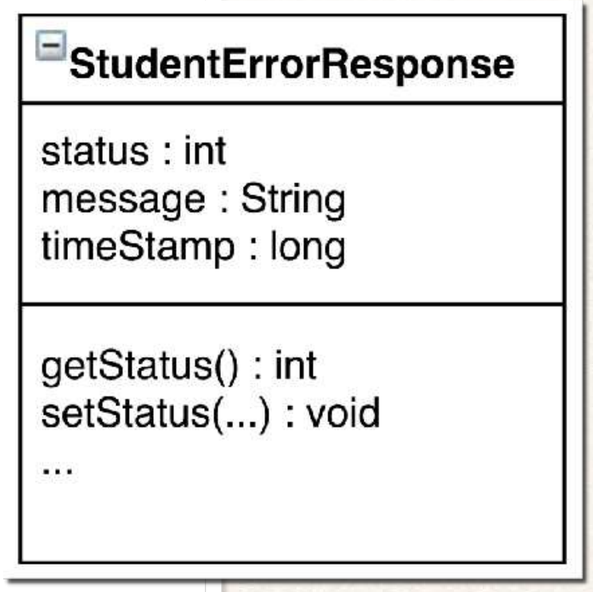
```
public class StudentErrorResponse {

  private int status;
  private String message;
  private long timeStamp;
```



Step 1: Create custom error response class

File: StudentErrorResponse.java

```
public class StudentErrorResponse {

  private int status;
  private String message;
  private long timeStamp;

  // constructors

  // getters / setters

}
```




```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

Step 2: Create custom student exception

Step 2: Create custom student exception

- The custom student exception will used by our REST service

Step 2: Create custom student exception

- The custom student exception will used by our REST service

- In our code, if we can't find student, then we'll throw an exception

Step 2: Create custom student exception

- The custom student exception will used by our REST service
- In our code, if we can't find student, then we'll throw an exception
- Need to define a custom student exception class

Step 2: Create custom student exception

- The custom student exception will used by our REST service
- In our code, if we can't find student, then we'll throw an exception
- Need to define a custom student exception class
  - `StudentNotFoundException`

Step 2: Create custom student exception

Step 2: Create custom student exception

File: StudentNotFoundException.java

```
public class StudentNotFoundException extends RuntimeException {
```

Step 2: Create custom student exception

File: StudentNotFoundException.java

```
public class StudentNotFoundException extends RuntimeException {

  public StudentNotFoundException(String message) {
    super(message);
  }

}
```

Call super class constructor

Step 3: Update REST service to throw exception

Step 3: Update REST service to throw exception

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
```

Step 3: Update REST service to throw exception

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {
```

Step 3: Update REST service to throw exception

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {

    // check the studentId against list size

    if ( (studentId >= theStudents.size()) || (studentId < 0) ) {
      throw new StudentNotFoundException("Student id not found - " + studentId);
    }
```

Throw exception

Step 3: Update REST service to throw exception

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {

    // check the studentId against list size

    if ( (studentId >= theStudents.size()) || (studentId < 0) ) {
      throw new StudentNotFoundException("Student id not found - " + studentId);
    }
```

Could also check results from DB

Throw exception

Step 3: Update REST service to throw exception

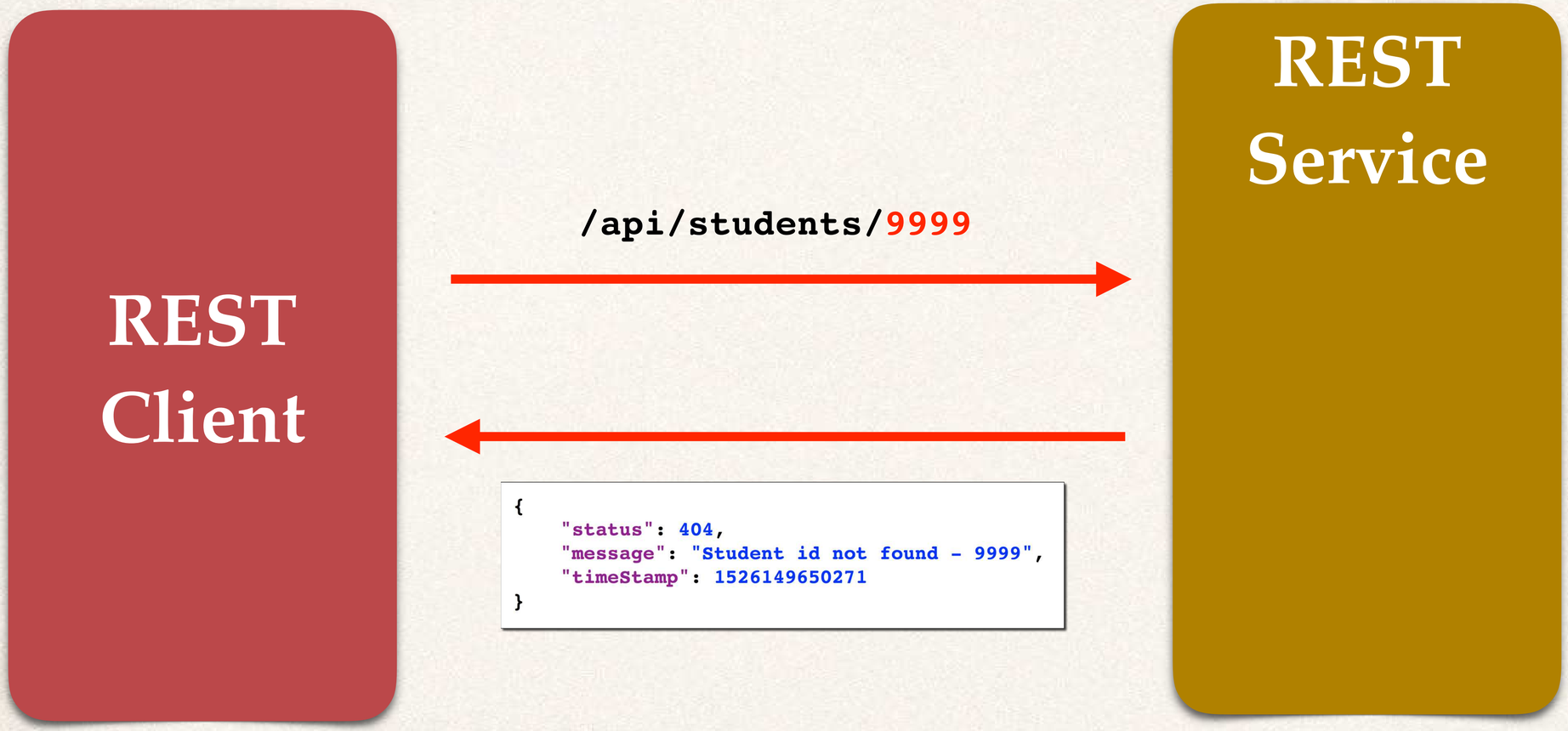
File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {

  @GetMapping("/students/{studentId}")
  public Student getStudent(@PathVariable int studentId) {

    // check the studentId against list size

    if ( (studentId >= theStudents.size()) || (studentId < 0) ) {
      throw new StudentNotFoundException("Student id not found - " + studentId);
    }

    return theStudents.get(studentId);

  }
  …
}
```

Throw exception

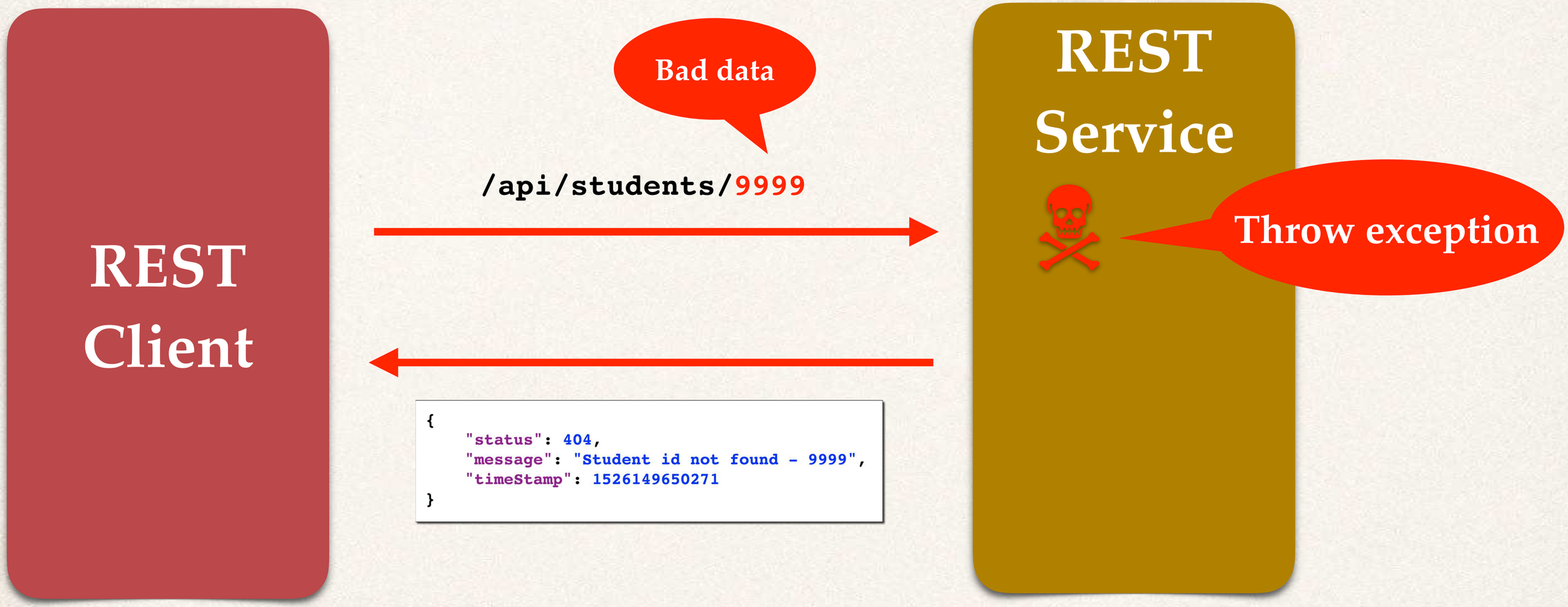
Happy path

Spring REST Exception Handling

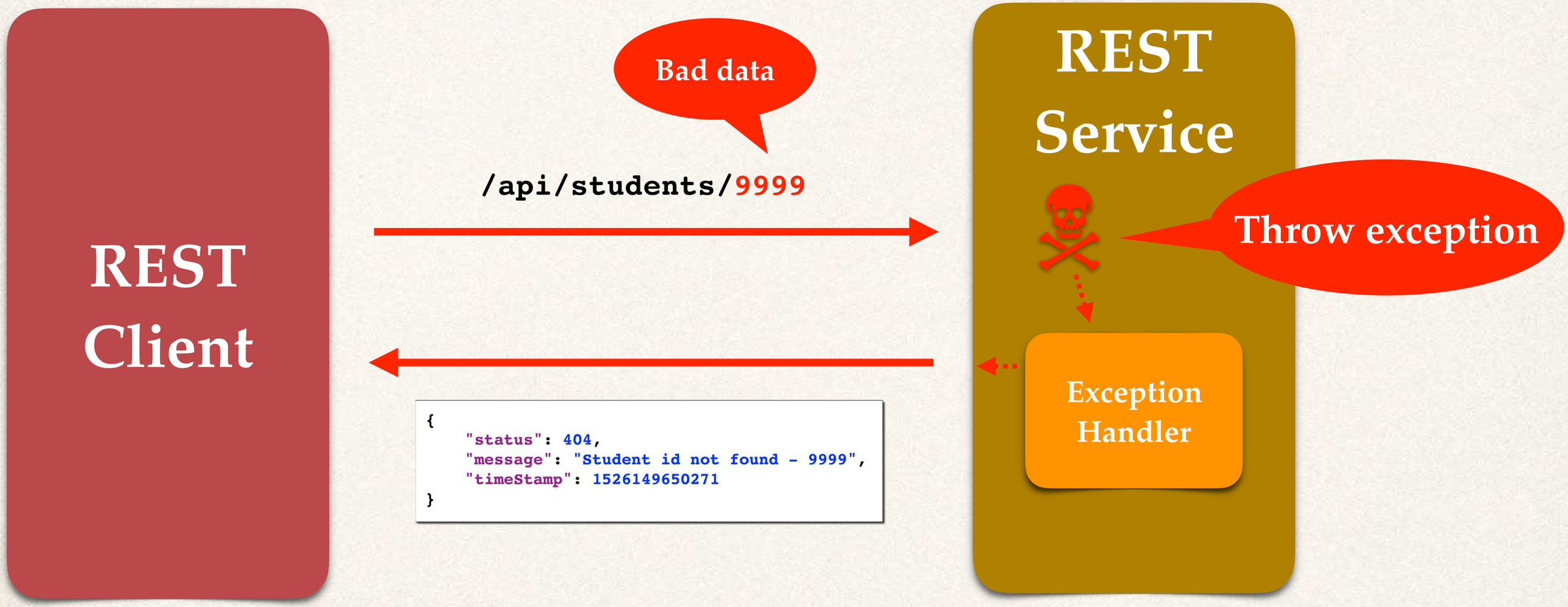


Spring REST Exception Handling

Spring REST Exception Handling



Spring REST Exception Handling



Step 4: Add exception handler method

Step 4: Add exception handler method

- Define exception handler method(s) with `@ExceptionHandler` annotation

Step 4: Add exception handler method

- Define exception handler method(s) with `@ExceptionHandler` annotation

- Exception handler will return a `ResponseEntity`

Step 4: Add exception handler method

- Define exception handler method(s) with `@ExceptionHandler` annotation
- Exception handler will return a `ResponseEntity`
- `ResponseEntity` is a wrapper for the HTTP response object

Step 4: Add exception handler method

- Define exception handler method(s) with `@ExceptionHandler` annotation
- Exception handler will return a `ResponseEntity`
- `ResponseEntity` is a wrapper for the HTTP response object
- `ResponseEntity` provides fine-grained control to specify:

Step 4: Add exception handler method

- Define exception handler method(s) with `@ExceptionHandler` annotation
- Exception handler will return a `ResponseEntity`
- `ResponseEntity` is a wrapper for the HTTP response object
- `ResponseEntity` provides fine-grained control to specify:
  - HTTP status code, HTTP headers and Response body

Step 4: Add exception handler method

Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …
```

Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {
```

Step 4: Add exception handler method

Exception handler method

java

```
pi")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {
```

Step 4: Add exception handler method

Exception handler method

Type of the response body

java

```
pi")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {
```

Step 4: Add exception handler method

Exception handler method

Type of the response body

Exception type to handle / catch

java

```
pi")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {
```

Step 4: Add exception handler method

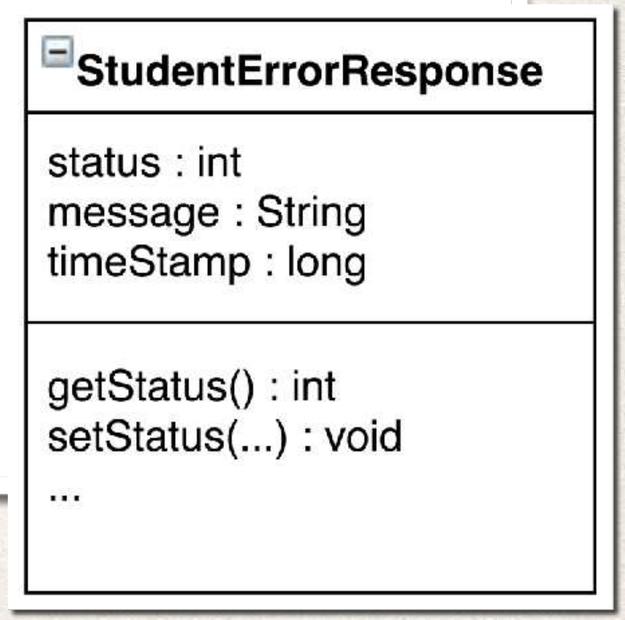
File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();
```

Step 4: Add exception handler method

File: StudentRestController.java

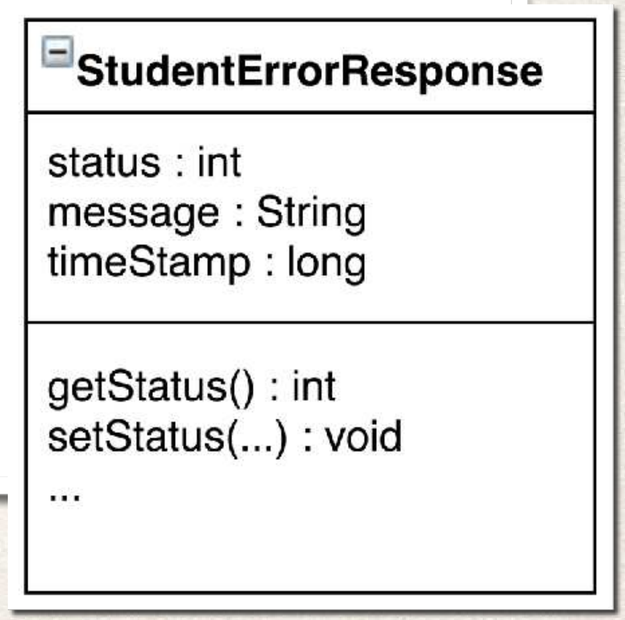
```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();
```



Step 4: Add exception handler method

File: StudentRestController.java

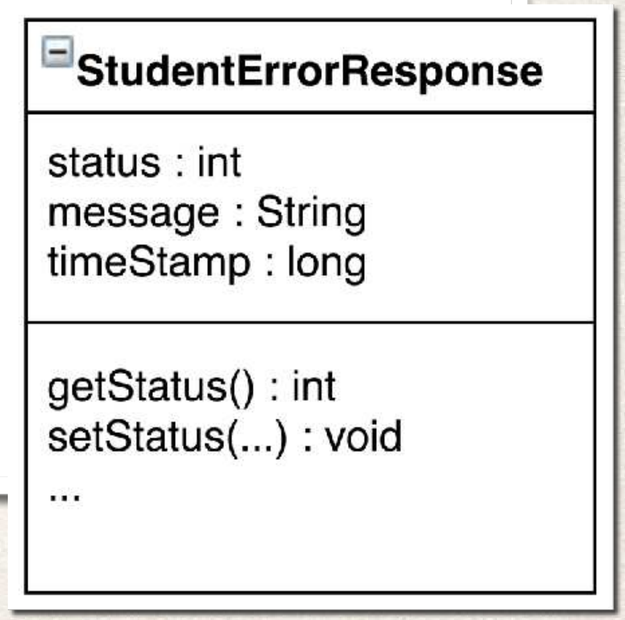
```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
```



Step 4: Add exception handler method

File: StudentRestController.java

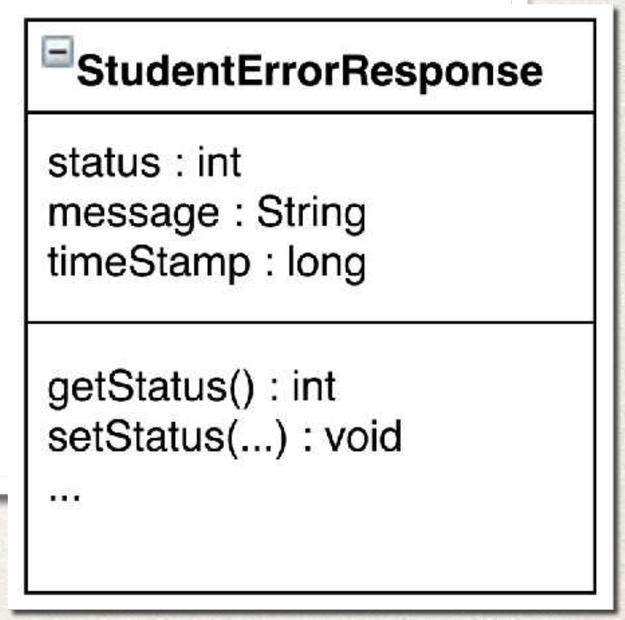
```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
```



Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());
```



Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());
```

StudentErrorResponse

status : int
message : String
timeStamp : long

getStatus() : int
setStatus(...) : void
...

Step 4: Add exception handler method

StudentErrorResponse

status : int
message : String
timeStamp : long

getStatus() : int
setStatus(...) : void
...

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Body

StudentErrorResponse

status : int
message : String
timeStamp : long

getStatus() : int
setStatus(...) : void
...

Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Body

Status code

StudentErrorResponse

status : int
message : String
timeStamp : long

getStatus() : int
setStatus(...) : void
...

Step 4: Add exception handler method

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Body

Status code

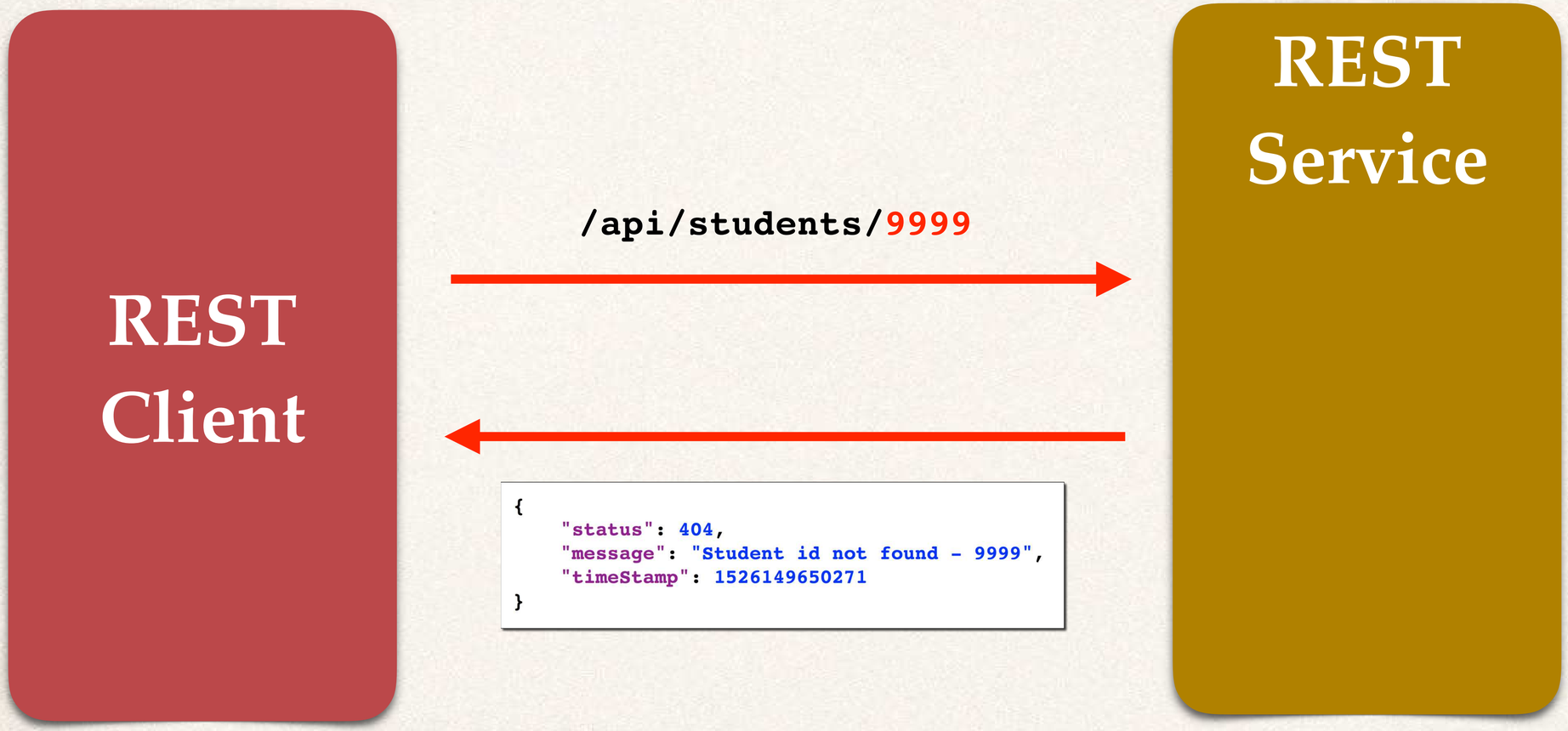
StudentErrorResponse

status : int
message : String
timeStamp : long

getStatus() : int
setStatus(...) : void
...

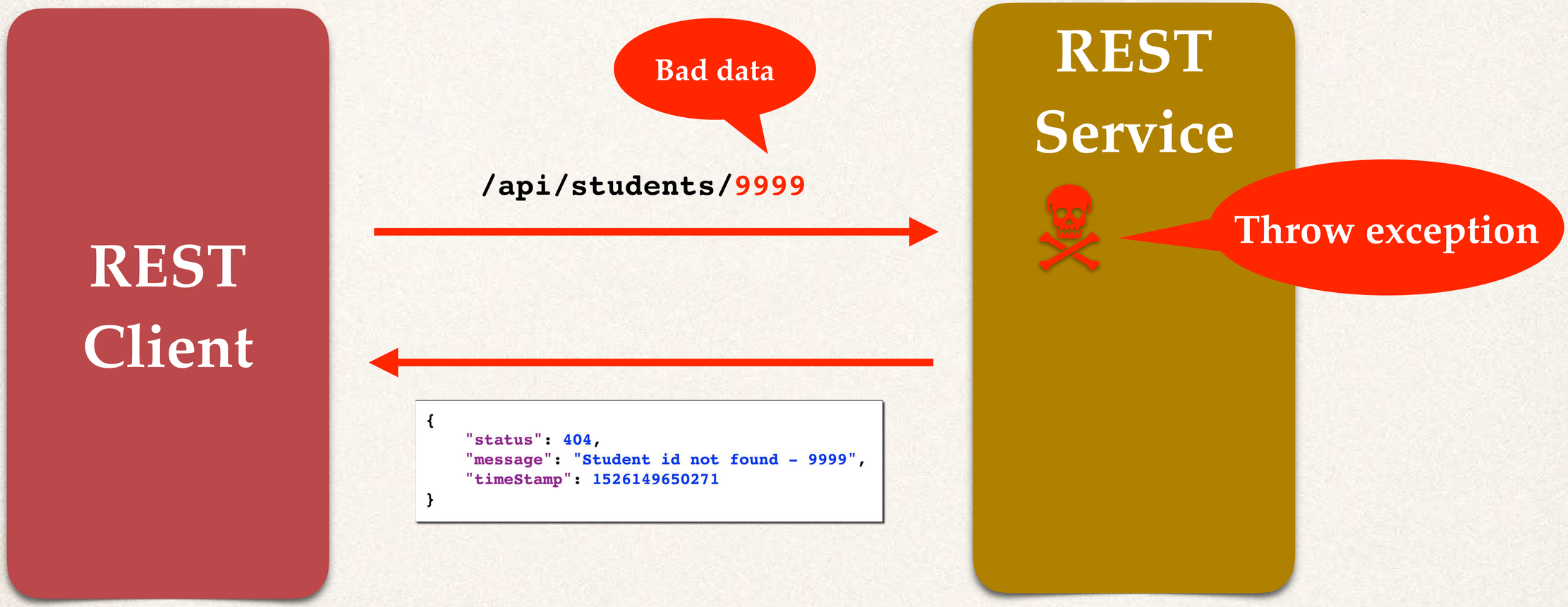
```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

Spring REST Exception Handling

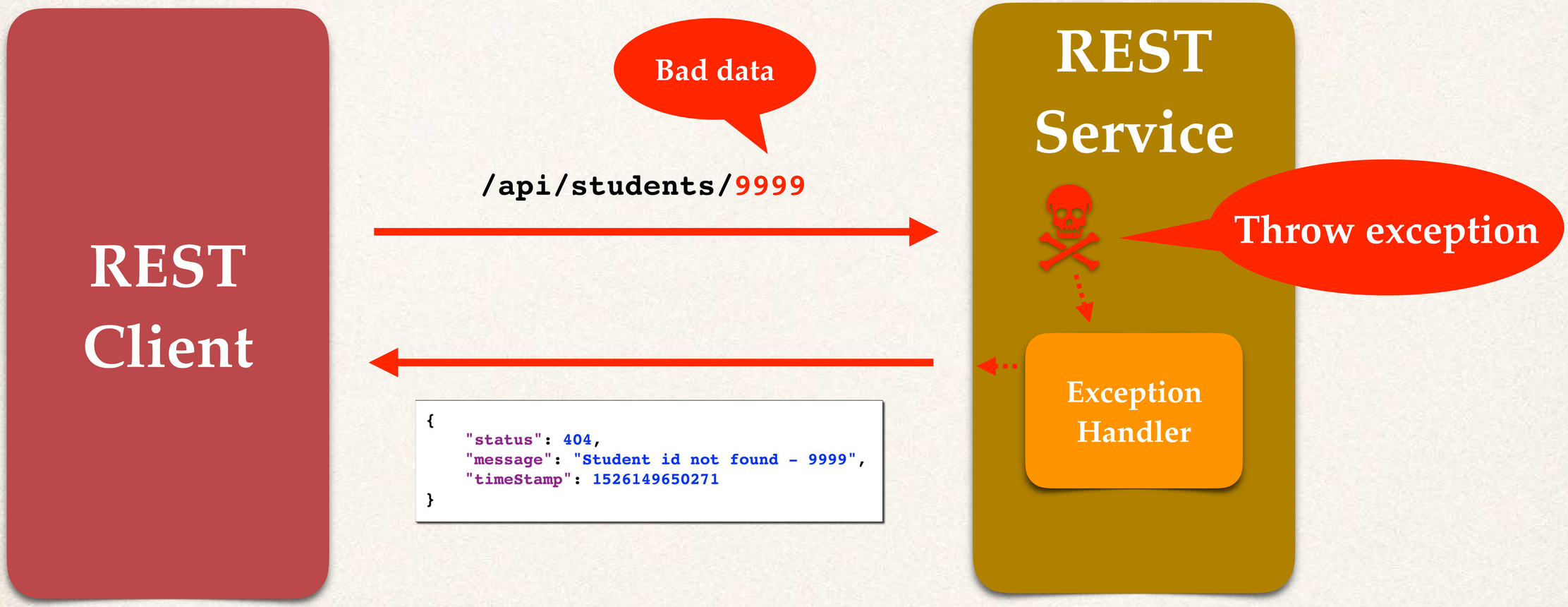


Spring REST Exception Handling

Spring REST Exception Handling

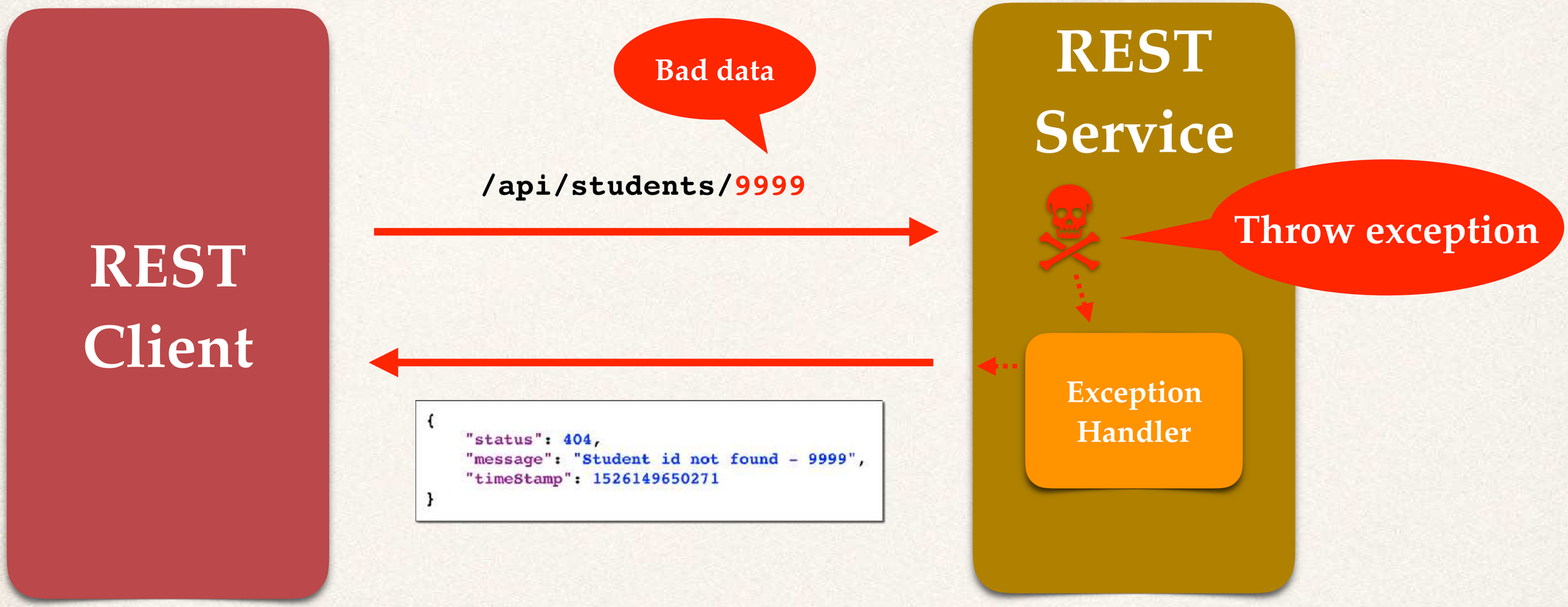


Spring REST Exception Handling



Spring REST - Global Exception Handling

Spring REST Exception Handling



It works, but …

It works, but …

- Exception handler code is only for the specific REST controller

It works, but …

- Exception handler code is only for the specific REST controller

- Can't be reused by other controllers :-(

It works, but …

- Exception handler code is only for the specific REST controller

- Can't be reused by other controllers :-(

Large projects
will have
multiple controllers

It works, but …

- Exception handler code is only for the specific REST controller
- Can't be reused by other controllers :-(
- We need **global** exception handlers

Large projects
will have
multiple controllers

It works, but …

- Exception handler code is only for the specific REST controller
- Can't be reused by other controllers :-(

Large projects
will have
multiple controllers

- We need **global** exception handlers
  - Promotes reuse

It works, but …

- Exception handler code is only for the specific REST controller
- Can't be reused by other controllers :-(

Large projects
will have
multiple controllers

- We need **global** exception handlers
  - Promotes reuse
  - Centralizes exception handling

Spring @ControllerAdvice

Spring @ControllerAdvice

- `@ControllerAdvice` is similar to an interceptor / filter

Spring @ControllerAdvice

- `@ControllerAdvice` is similar to an interceptor / filter

- Pre-process requests to controllers

Spring @ControllerAdvice

- `@ControllerAdvice` is similar to an interceptor / filter
- Pre-process requests to controllers
- Post-process responses to handle exceptions

Spring @ControllerAdvice

- `@ControllerAdvice` is similar to an interceptor / filter
- Pre-process requests to controllers
- Post-process responses to handle exceptions
- Perfect for global exception handling

Spring @ControllerAdvice

- `@ControllerAdvice` is similar to an interceptor / filter
- Pre-process requests to controllers
- Post-process responses to handle exceptions
- Perfect for global exception handling

Real-time use of AOP

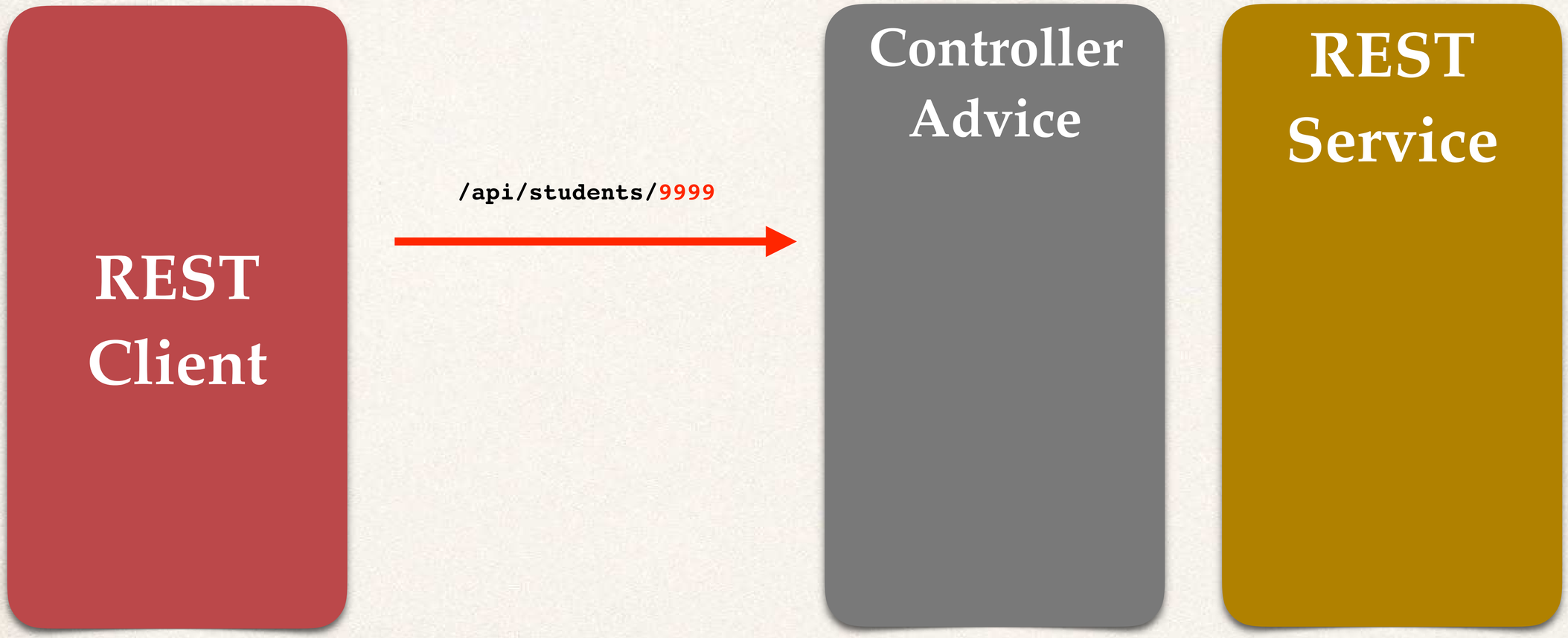
Spring REST Exception Handling

REST Client

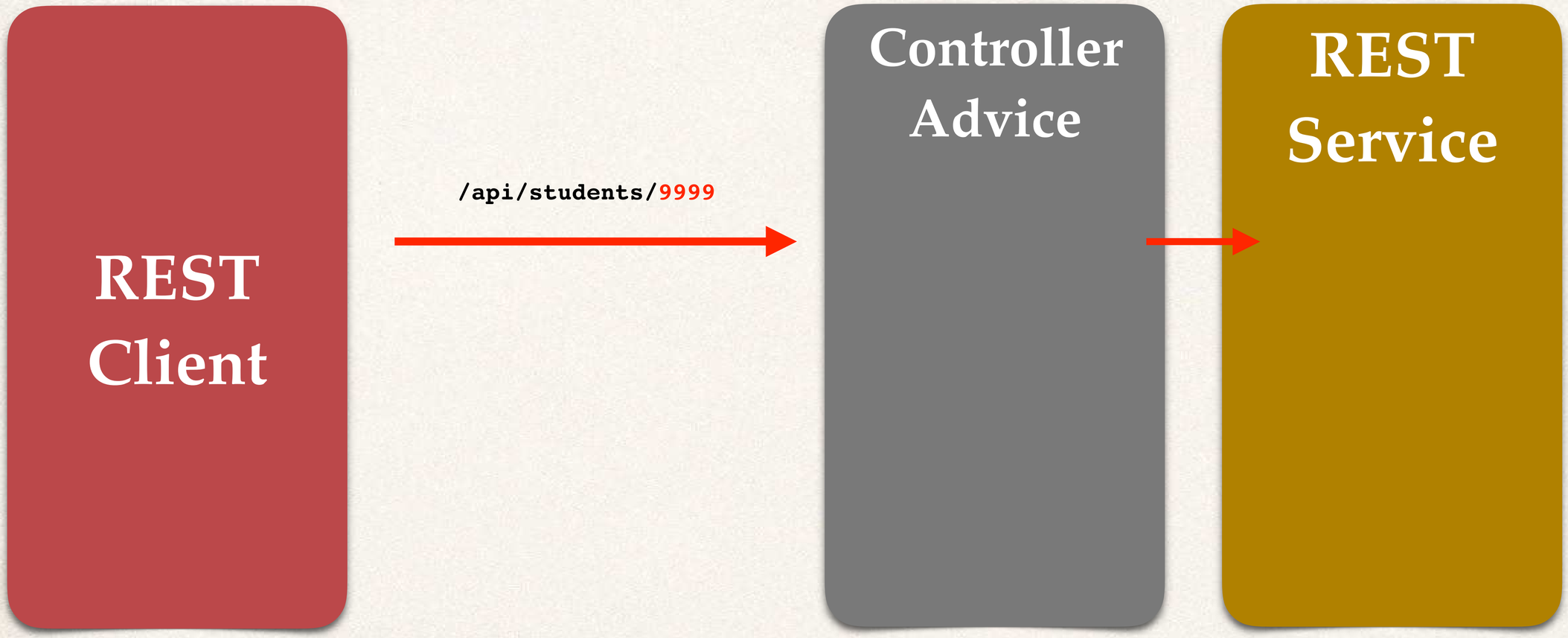
REST Service

Spring REST Exception Handling

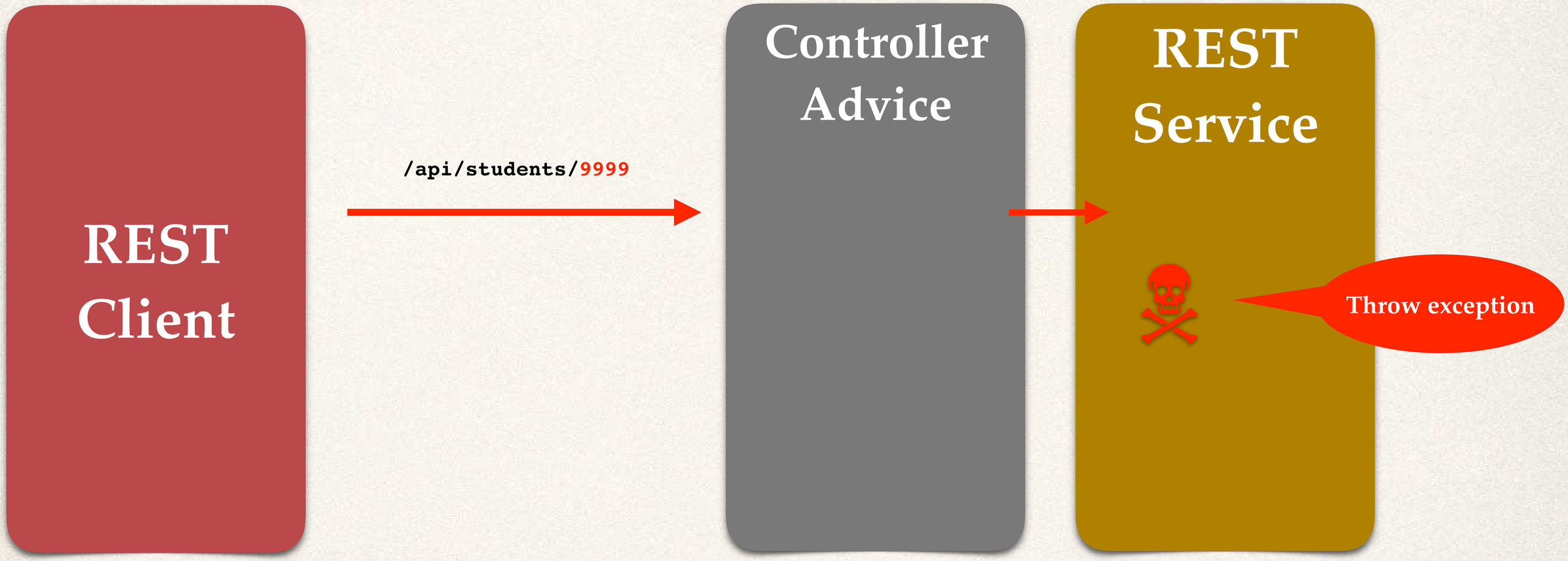
Spring REST Exception Handling



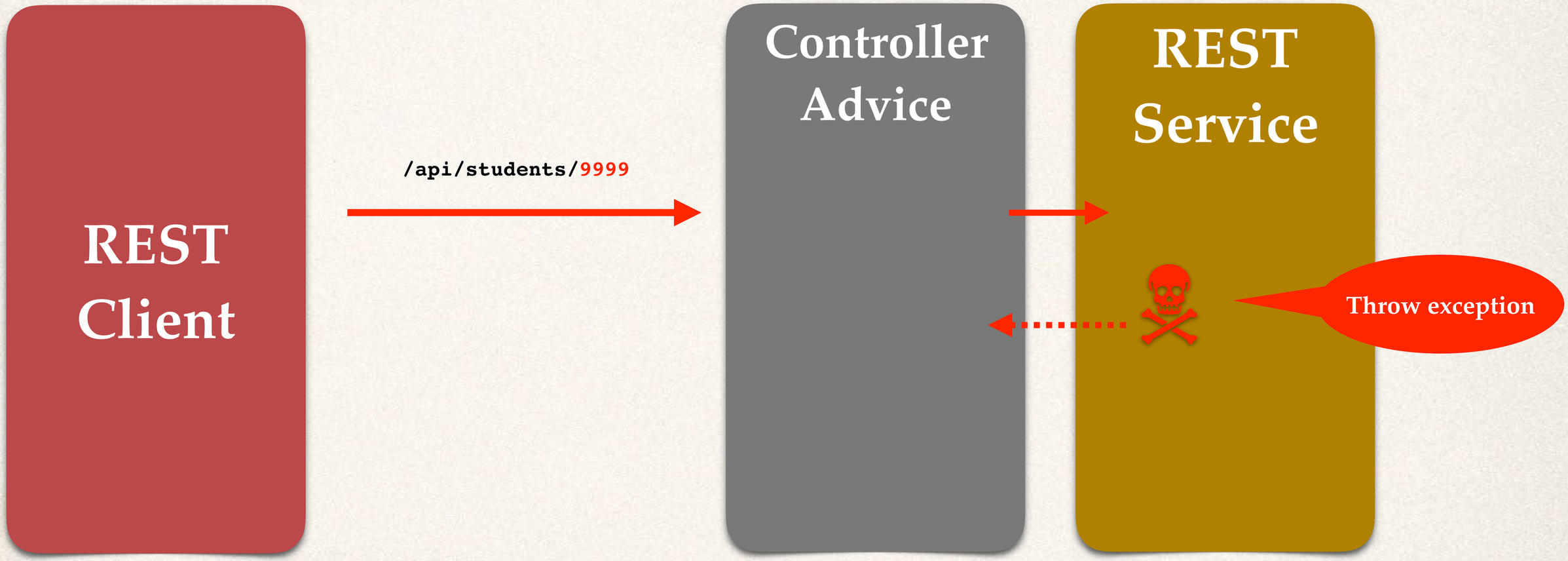
Spring REST Exception Handling



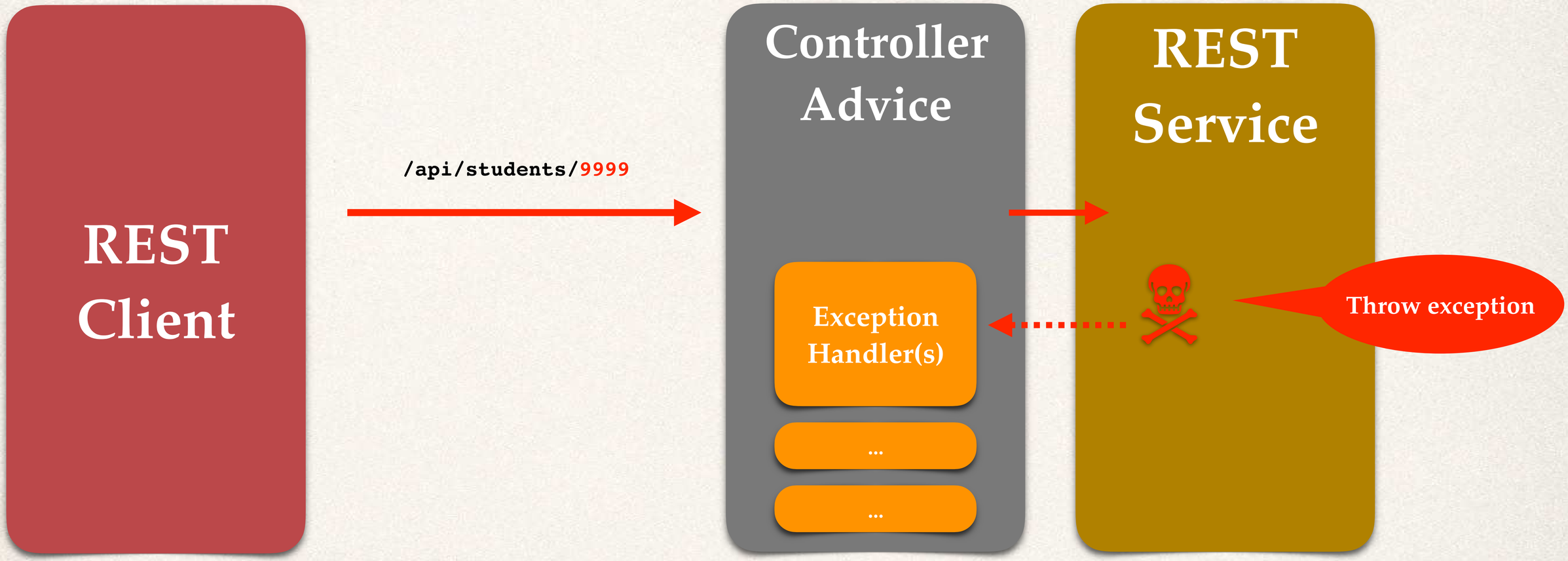
Spring REST Exception Handling



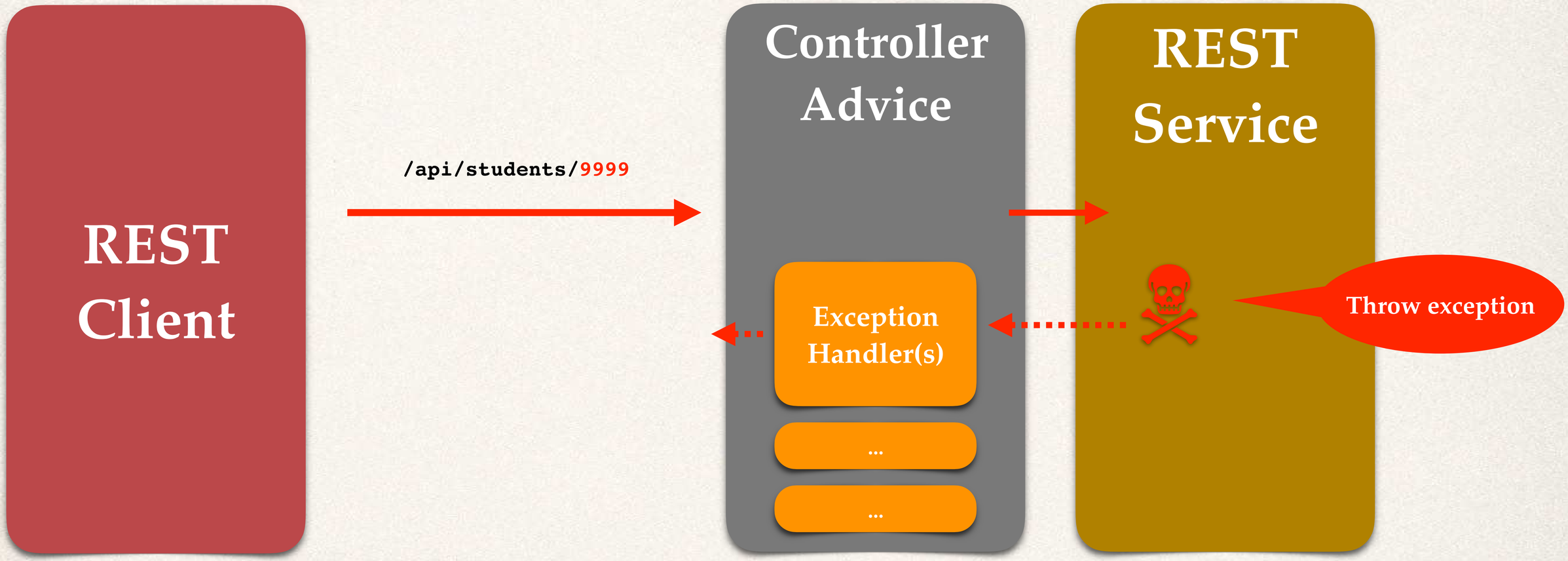
Spring REST Exception Handling



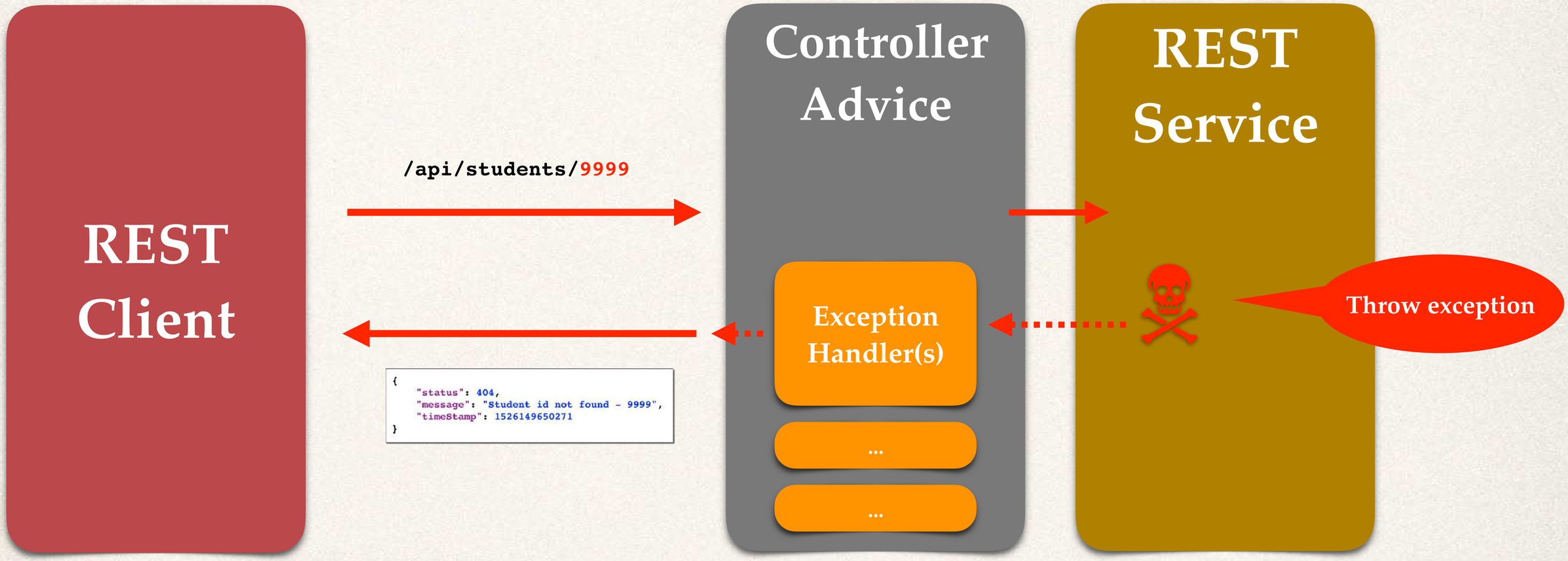
Spring REST Exception Handling



Spring REST Exception Handling



Spring REST Exception Handling



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create new @ControllerAdvice

Development Process

Step-By-Step

1. Create new @ControllerAdvice

2. Refactor REST service … remove exception handling code

Development Process

Step-By-Step

1. Create new @ControllerAdvice

2. Refactor REST service … remove exception handling code

3. Add exception handling code to @ControllerAdvice

Step 1: Create new @ControllerAdvice

File: StudentRestExceptionHandler.java

Step 1: Create new @ControllerAdvice

File: StudentRestExceptionHandler.java

```
@ControllerAdvice
public class StudentRestExceptionHandler {

  ...

}
```

Step 2: Refactor - remove exception handling

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Step 2: Refactor - remove exception handling

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Step 2: Refactor - remove exception handling

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Remove this code

Step 3: Add exception handler to @ControllerAdvice

File: StudentRestExceptionHandler.java

```
@ControllerAdvice
public class StudentRestExceptionHandler {





}
```

Step 3: Add exception handler to @ControllerAdvice

File: StudentRestExceptionHandler.java

```
@ControllerAdvice
public class StudentRestExceptionHandler {

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Step 3: Add exception handler to @ControllerAdvice

File: StudentRestExceptionHandler.java

Same code as before

```
@ControllerAdvice
public class StudentRestExceptionHandler {

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

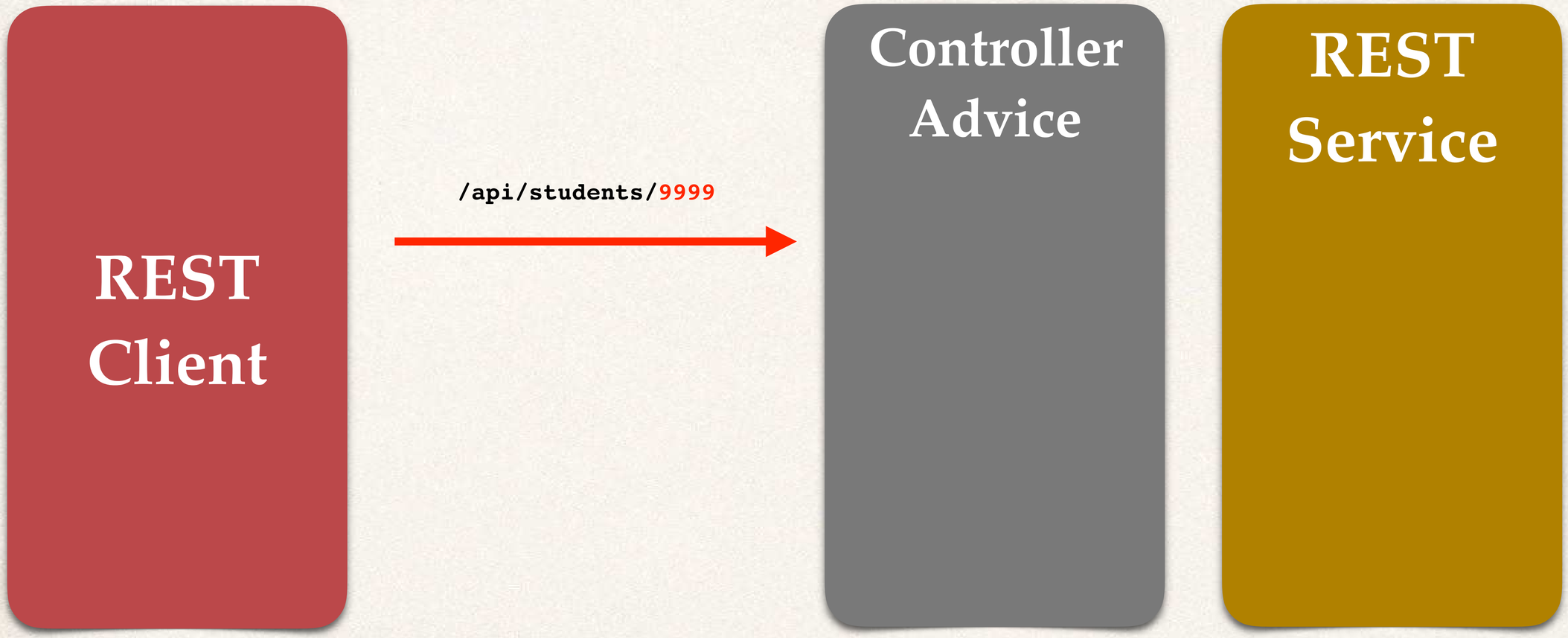
Spring REST Exception Handling

REST Client

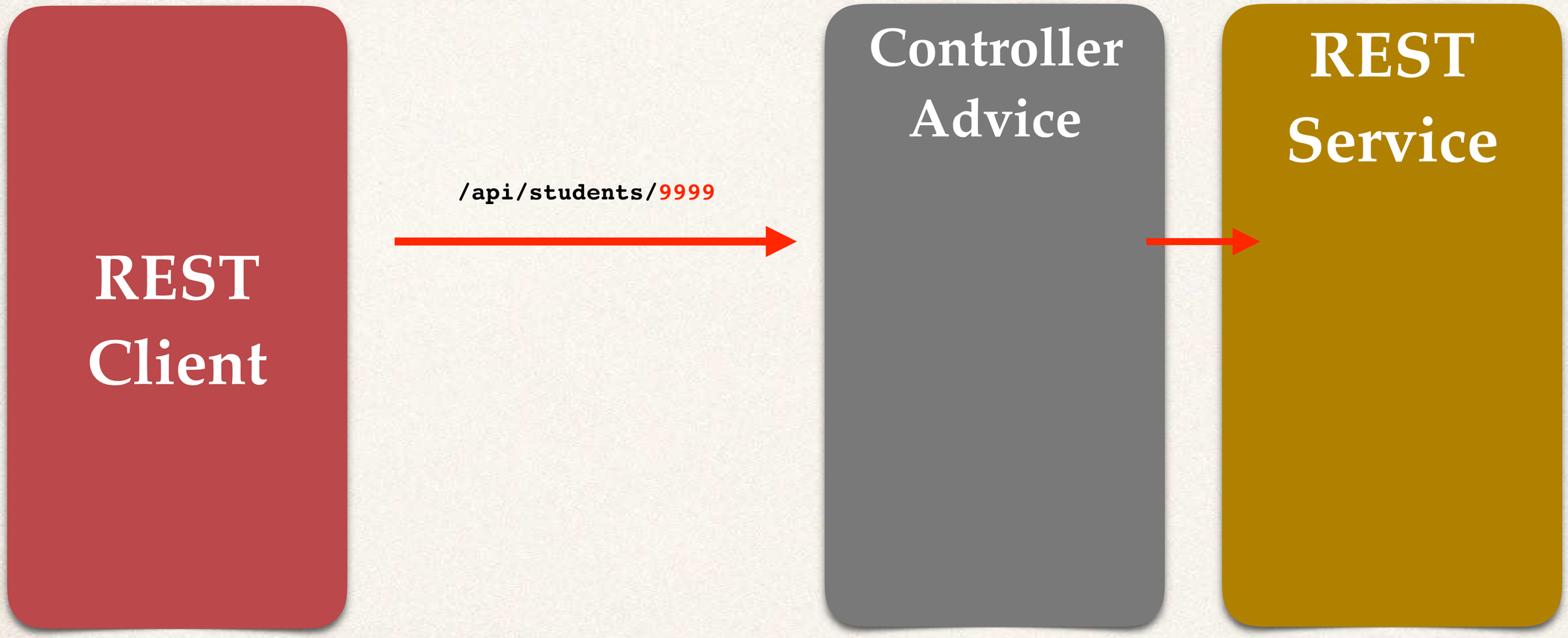
REST Service

Spring REST Exception Handling

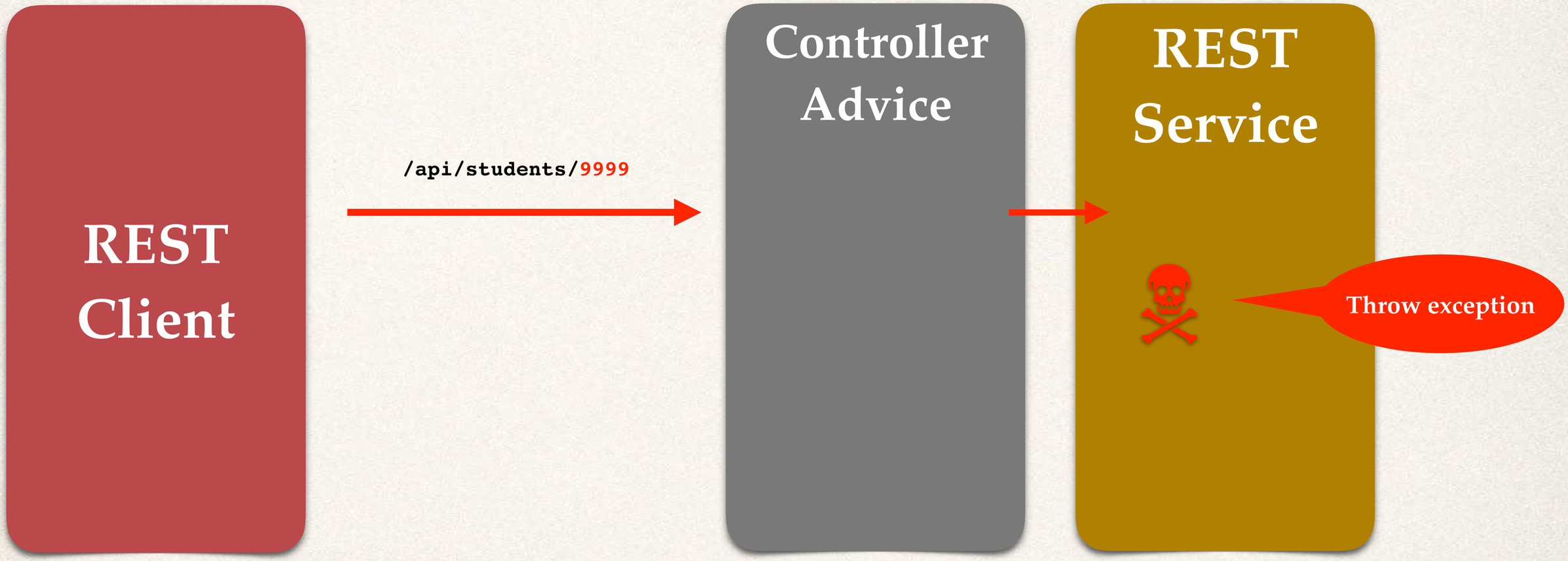
Spring REST Exception Handling



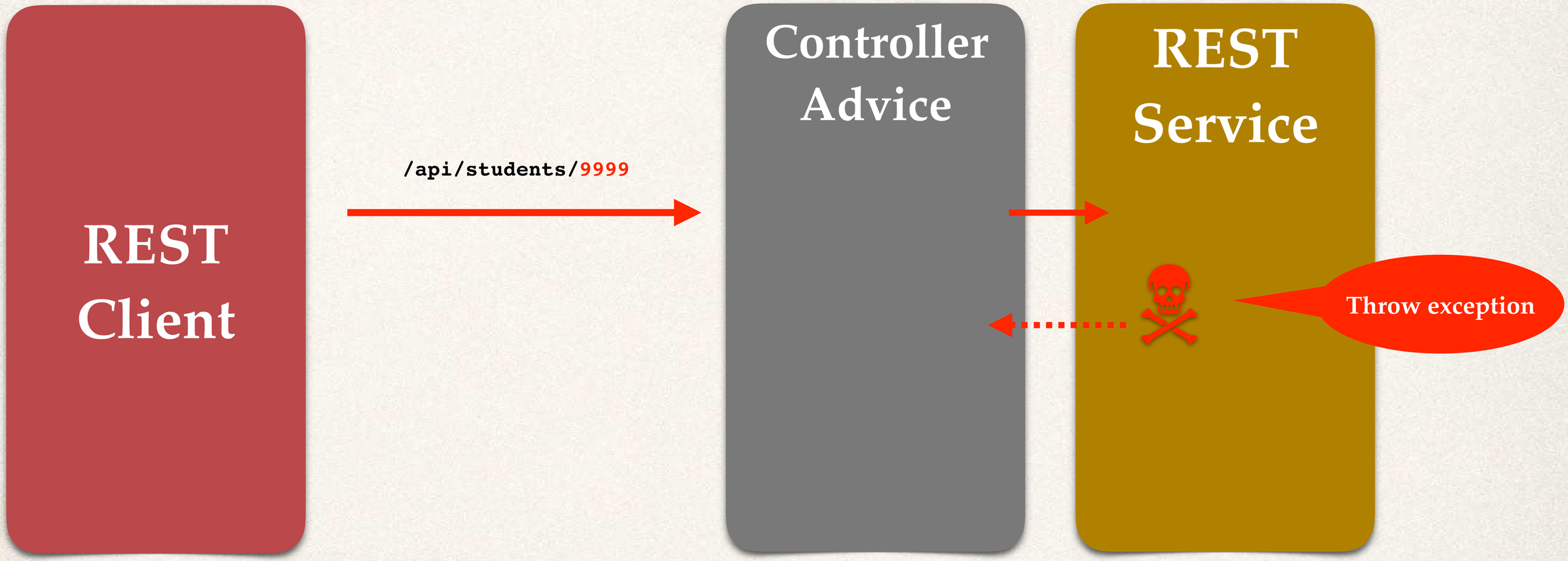
Spring REST Exception Handling



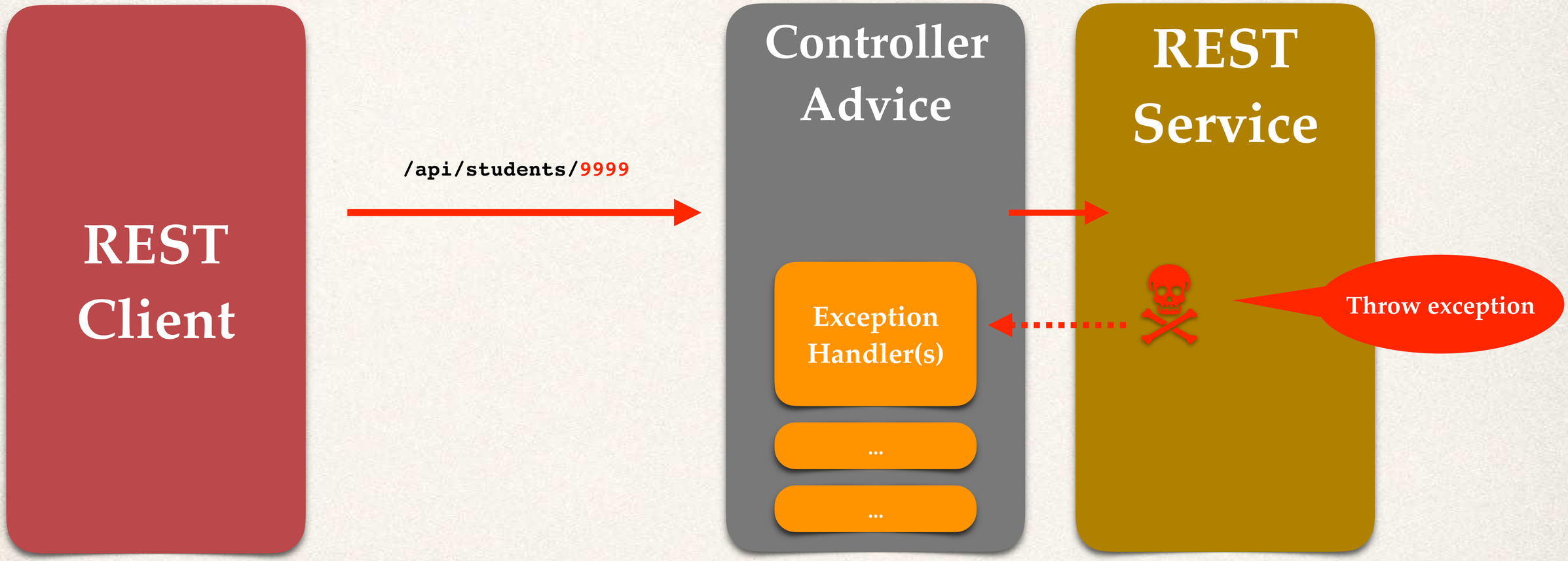
Spring REST Exception Handling



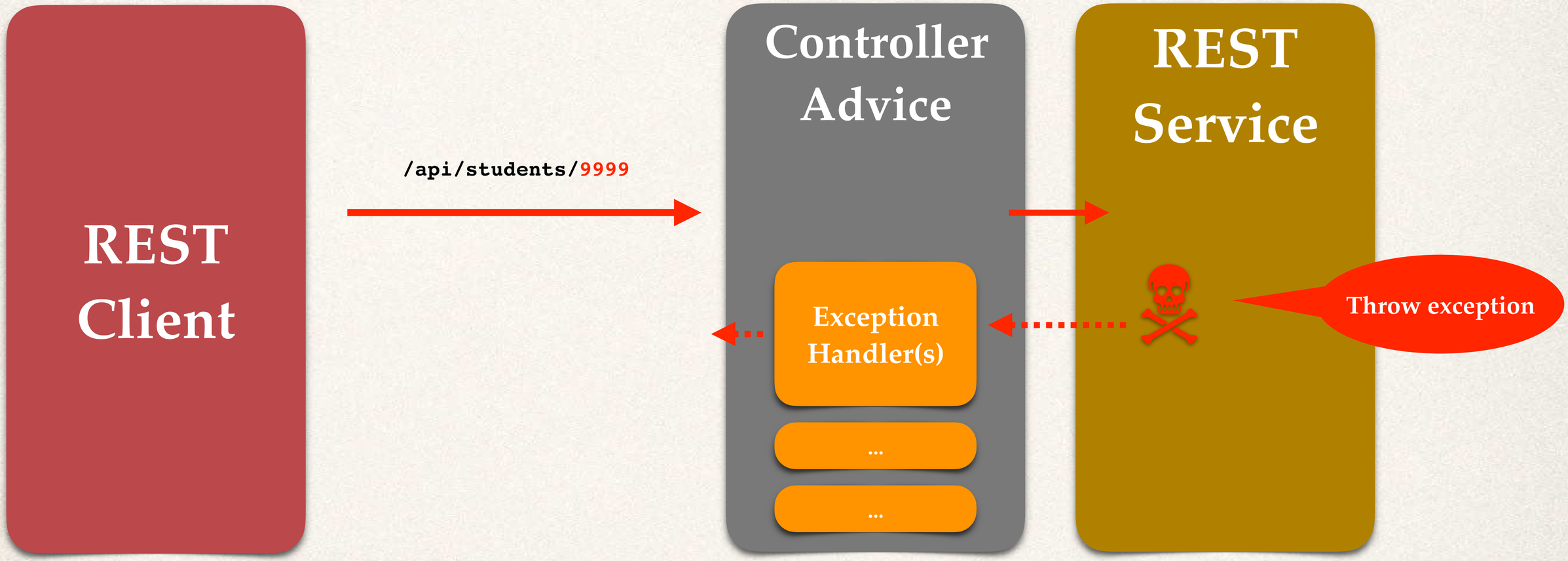
Spring REST Exception Handling



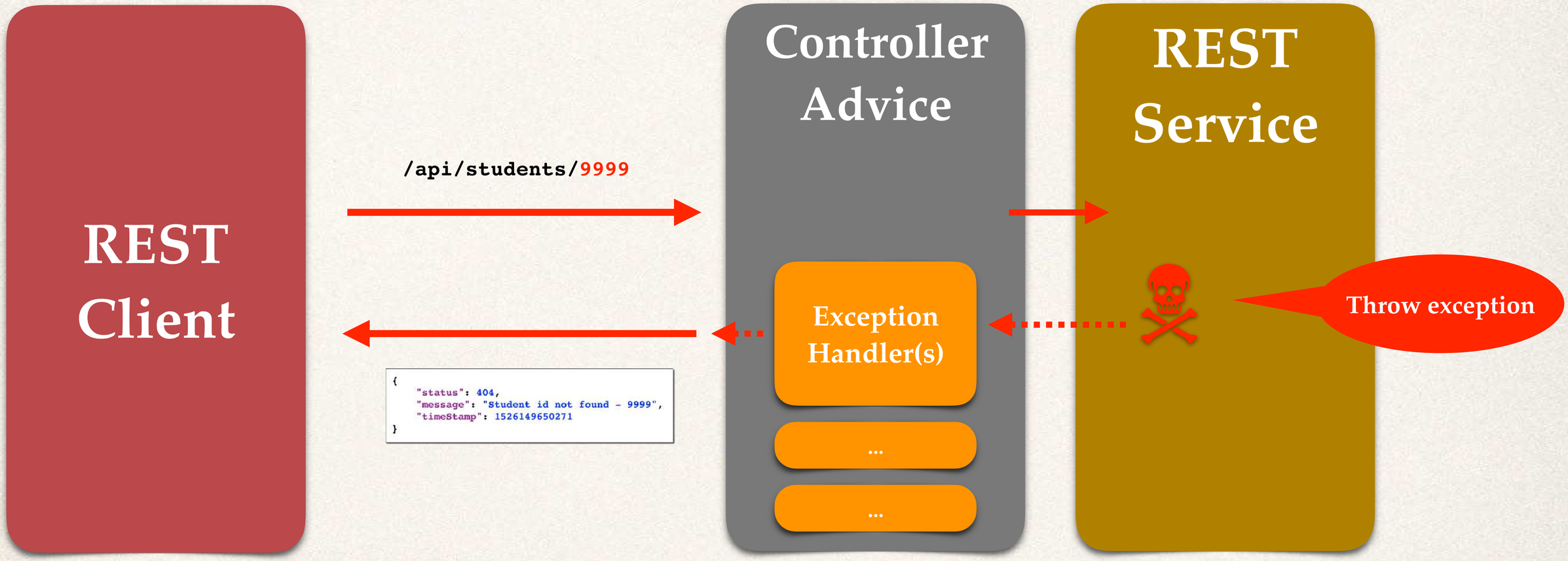
Spring REST Exception Handling



Spring REST Exception Handling



Spring REST Exception Handling



Spring REST API Design

REST API Design

REST API Design

- For real-time projects, who will use your API?

REST API Design

- For real-time projects, who will use your API?
- Also, how will they use your API?

REST API Design

- For real-time projects, who will use your API?
- Also, how will they use your API?
- Design the API based on requirements

API Design Process

Step-By-Step

API Design Process

Step-By-Step

1. Review API requirements

API Design Process

Step-By-Step

1. Review API requirements

2. Identify main resource / entity

API Design Process

Step-By-Step

1. Review API requirements

2. Identify main resource / entity

3. Use HTTP methods to assign action on resource

Step 1: Review API Requirements

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

- Get a list of customers

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to
- Get a list of customers
- Get a single customer by id

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

- Get a list of customers
- Get a single customer by id
- Add a new customer

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

- Get a list of customers
- Get a single customer by id
- Add a new customer
- Update a customer

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

- Get a list of customers
- Get a single customer by id
- Add a new customer
- Update a customer
- Delete a customer

Step 1: Review API Requirements

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

- Get a list of customers
- Get a single customer by id
- Add a new customer
- Update a customer
- Delete a customer

Full CRUD

Step 2: Identify main resource / entity

Step 2: Identify main resource / entity

- To identify main resource / entity, look for the most prominent "noun"

Step 2: Identify main resource / entity

- To identify main resource / entity, look for the most prominent "noun"

- For our project, it is "customer"

Step 2: Identify main resource / entity

- To identify main resource / entity, look for the most prominent "noun"
- For our project, it is "customer"
- Convention is to use plural form of resource / entity: **customers**

Step 2: Identify main resource / entity

- To identify main resource / entity, look for the most prominent "noun"
- For our project, it is "customer"
- Convention is to use plural form of resource / entity: **customers**

```
/api/customers
```

Step 3: Use HTTP methods to assign action on resource

Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action

Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action
POST	Create a new entity

Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action
POST	Create a new entity
GET	Read a list of entities or single entity

Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action
POST	Create a new entity
GET	Read a list of entities or single entity
PUT	Update an existing entity

Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action
POST	Create a new entity
GET	Read a list of entities or single entity
PUT	Update an existing entity
DELETE	Delete an existing entity

Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action
POST	Create a new entity
GET	Read a list of entities or single entity
PUT	Update an existing entity
DELETE	Delete an existing entity

Full CRUD

CRUD Endpoint Examples

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

For POST and PUT,
we will send customer data as JSON in request message body

Anti-Patterns

Anti-Patterns

- DO NOT DO THIS … these are REST anti-patterns, bad practice

Anti-Patterns

- DO NOT DO THIS … these are REST anti-patterns, bad practice

```
/api/customersList
/api/deleteCustomer
/api/addCustomer
/api/updateCustomer
```

Anti-Patterns

- DO NOT DO THIS … these are REST anti-patterns, bad practice

```
/api/customersList
/api/deleteCustomer
/api/addCustomer
/api/updateCustomer
```

✗

Don't include actions in the endpoint

Anti-Patterns

- DO NOT DO THIS … these are REST anti-patterns, bad practice

```
/api/customersList
/api/deleteCustomer
/api/addCustomer
/api/updateCustomer
```

Don't include actions in the endpoint

Instead, use HTTP methods to assign actions

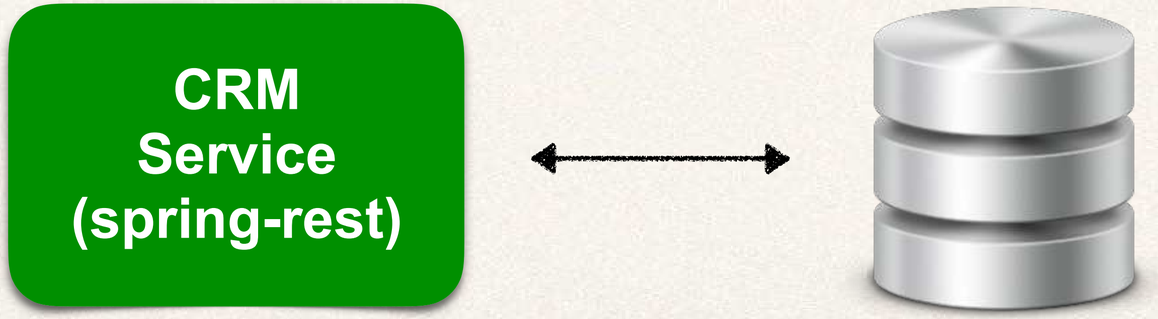
CRM Real-Time Project

CRM Real-Time Project

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

CRM Real-Time Project

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer



More API Examples

More API Examples

- On the following slides, we'll look at APIs from other real-time projects

More API Examples

- On the following slides, we'll look at APIs from other real-time projects

- PayPal

More API Examples

- On the following slides, we'll look at APIs from other real-time projects
- PayPal
- GitHub

More API Examples

- On the following slides, we'll look at APIs from other real-time projects
- PayPal
- GitHub
- SalesForce

PayPal

PayPal

- PayPal Invoicing API

PayPal

- PayPal Invoicing API
  - `https://developer.paypal.com/docs/api/invoicing/`

PayPal

- PayPal Invoicing API
  - https://developer.paypal.com/docs/api/invoicing/

PayPal Developer Docs APIs Support

Create draft invoice

POST /v1/invoicing/invoices

List invoices

GET /v1/invoicing/invoices

Show invoice details

GET /v1/invoicing/invoices/{invoice_id}

Update invoice

PUT /v1/invoicing/invoices/{invoice_id}

Delete draft invoice

DELETE /v1/invoicing/invoices/{invoice_id}

GitHub

GitHub

- GitHub Repositories API

GitHub

- GitHub Repositories API
  - `https://developer.github.com/v3/repos/#repositories`

GitHub

- GitHub Repositories API
  - `https://developer.github.com/v3/repos/#repositories`

GitHub Developer

Create a new repository

```
POST /user/repos
```

List your repositories

```
GET /user/repos
```

Delete a repository

```
DELETE /repos/:owner/:repo
```

Get a repository

```
GET /repos/:owner/:repo
```

SalesForce REST API

SalesForce REST API

- Industries REST API

SalesForce REST API

- Industries REST API
  - https://sforce.co/2J40ALH

SalesForce REST API

- Industries REST API
  - https://sforce.co/2J40ALH

Retrieve All Individuals

```
GET  /services/apexrest/v1/individual/
```

Retrieve One Individual

```
GET  /services/apexrest/v1/individual/{individual_id}
```

Create an individual

```
POST /services/apexrest/clinic01/v1/individual/
```

Update an individual

```
PUT  /services/apexrest/clinic01/v1/individual/
```

Spring REST API - Real Time Project

API Requirements - Refresher

API Requirements - Refresher

From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system.

REST clients should be able to

- Get a list of customers
- Get a single customer by id
- Add a new customer
- Update a customer
- Delete a customer

Real-Time Project

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Application Architecture

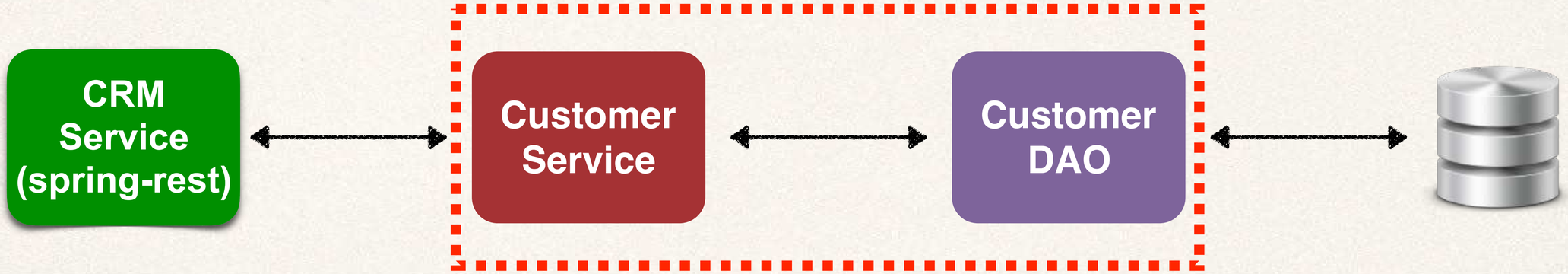
Application Architecture

CRM
Service
(spring-rest)

Application Architecture

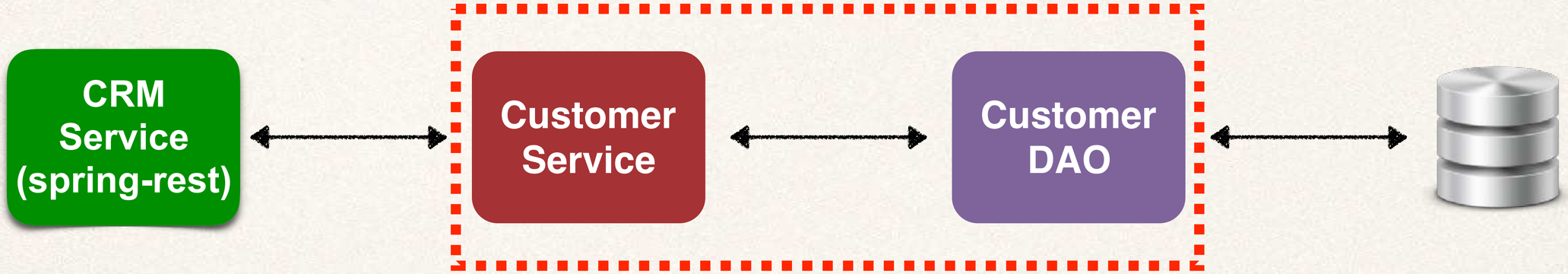
CRM
Service
(spring-rest)

Application Architecture

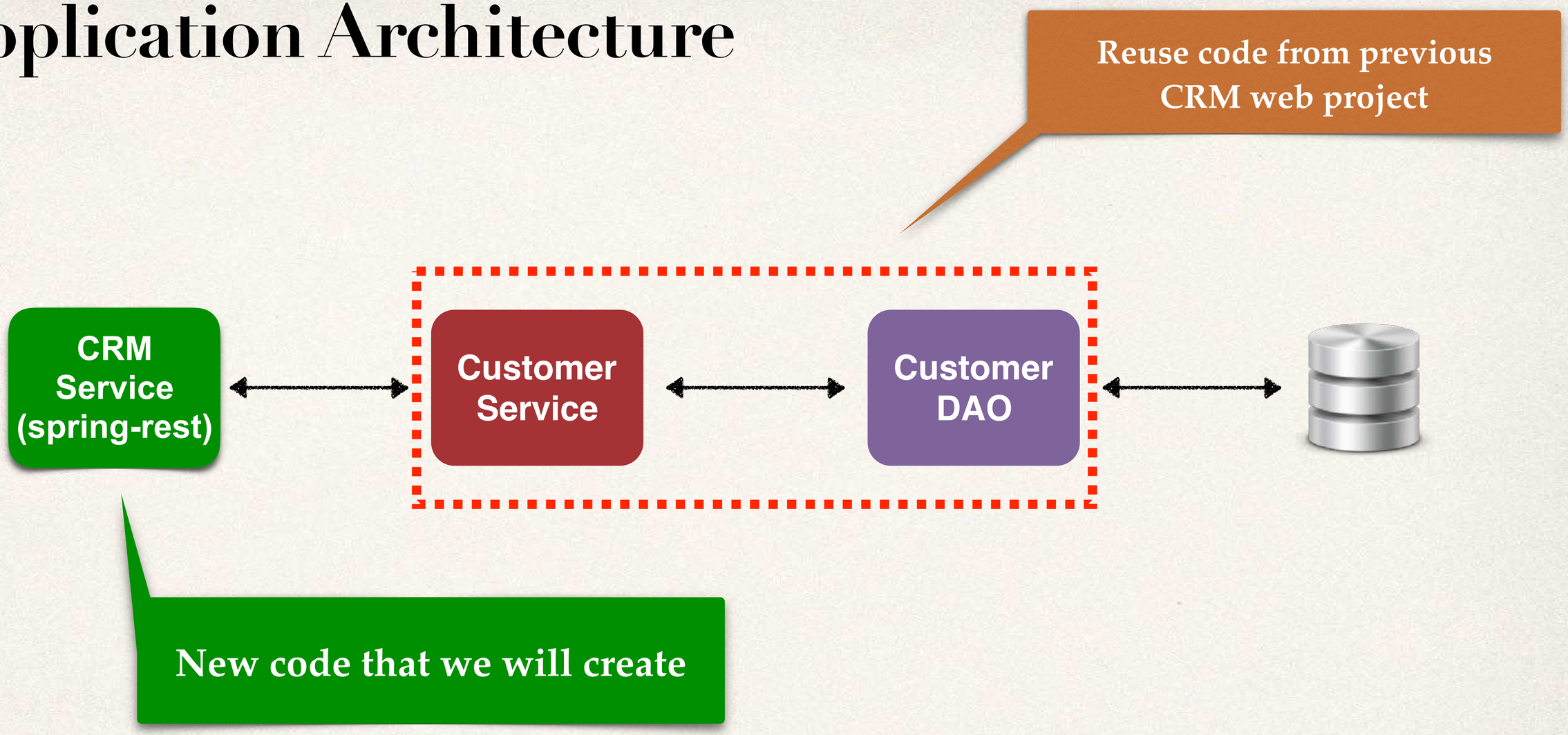


Application Architecture

Reuse code from previous CRM web project



Application Architecture



Project Set Up

Project Set Up

- We will download a Maven starter project

Project Set Up

- We will download a Maven starter project
- Includes CustomerService, CustomerDAO and Customer entity

Project Set Up

- We will download a Maven starter project

- Includes CustomerService, CustomerDAO and Customer entity
  - We created all of this code already

Project Set Up

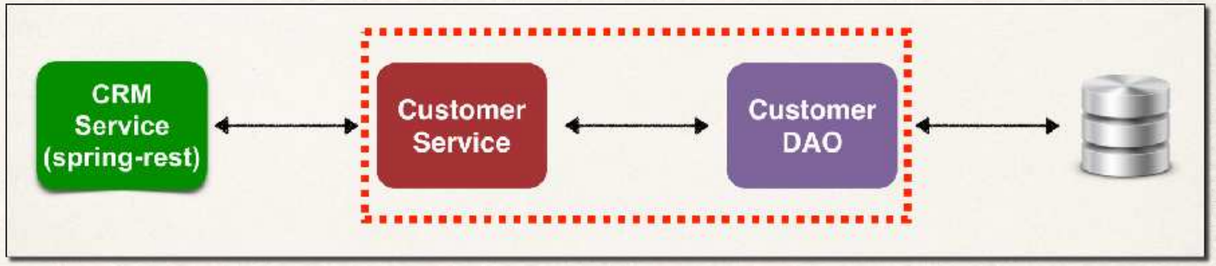
- We will download a Maven starter project
- Includes CustomerService, CustomerDAO and Customer entity
  - We created all of this code already
- Allows us to focus on creating CRM REST Service

Development Process

Step-By-Step

Development Process

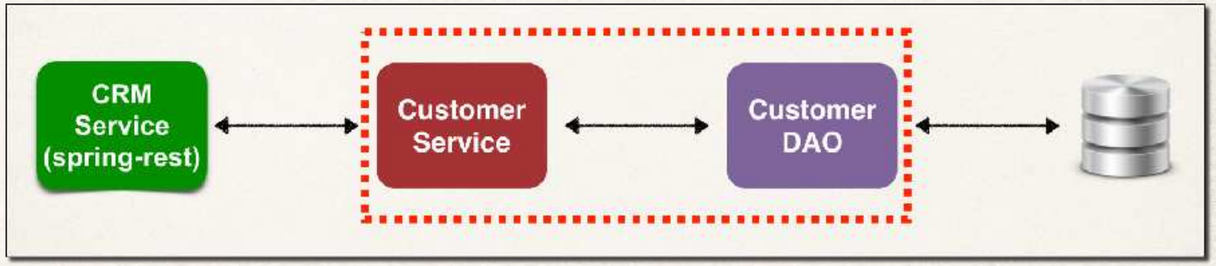
Step-By-Step



Development Process

Step-By-Step

1. Get customers

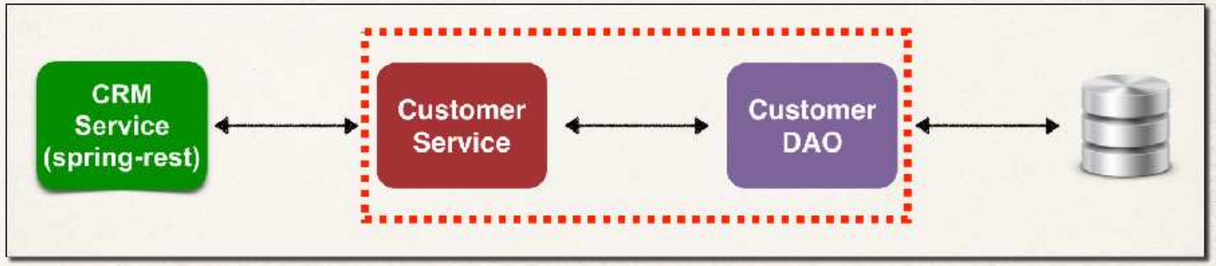


Development Process

Step-By-Step

1. Get customers

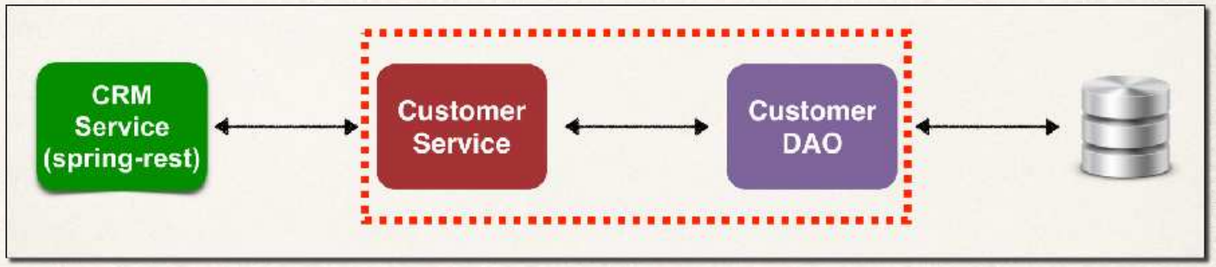
2. Get single customer by ID



Development Process

Step-By-Step

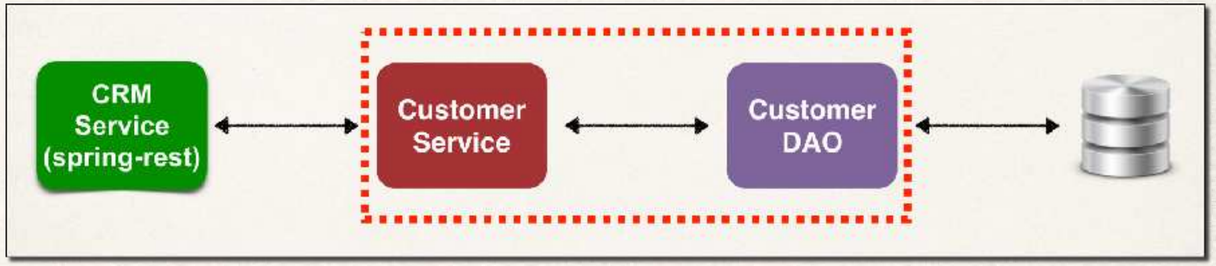
1. Get customers
2. Get single customer by ID
3. Add a new customer



Development Process

Step-By-Step

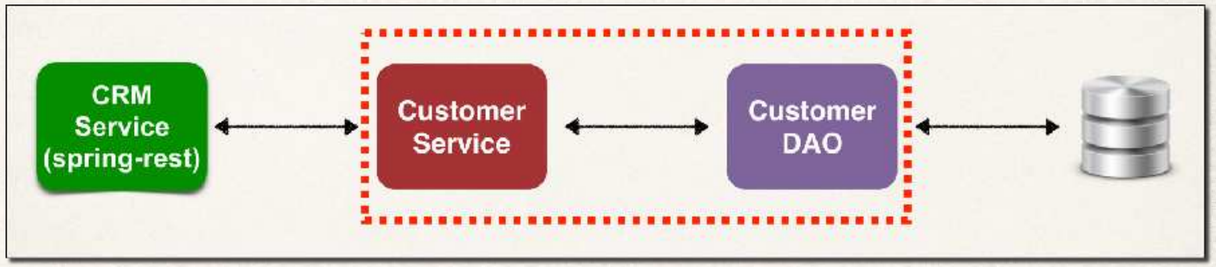
1. Get customers
2. Get single customer by ID
3. Add a new customer
4. Update an existing customer



Development Process

Step-By-Step

1. Get customers
2. Get single customer by ID
3. Add a new customer
4. Update an existing customer
5. Delete an existing customer



Code Review

Step-By-Step

1. pom.xml file
2. All Java Config
3. Configuration file for database connection
4. Hibernate Entity class: `Customer`
5. DAO: `CustomerDAO` ...
6. Service: `CustomerService` ...

Spring CRM REST - Get Customers

Application Interaction

Application Interaction

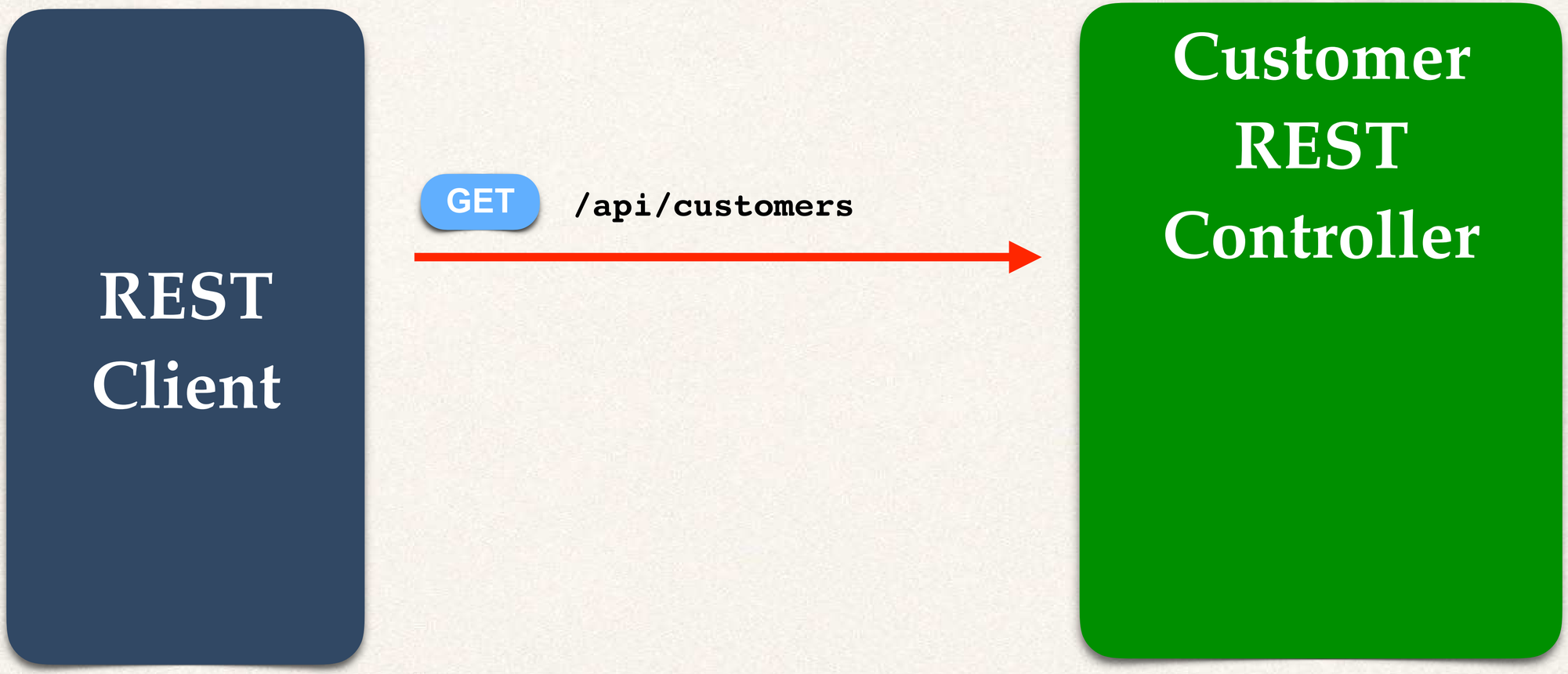
REST
Client

Application Interaction

REST
Client

Customer
REST
Controller

Application Interaction



Application Interaction

REST Client

GET /api/customers

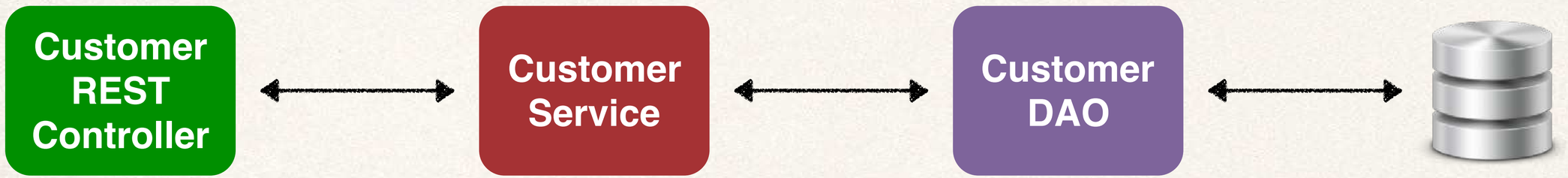

```
[
    {
        "id": 1,
        "firstName": "David",
        "lastName": "Adams",
        "email": "david@luv2code.com"
    },
    {
        "id": 5,
        "firstName": "Maxwell",
        "lastName": "Dixon",
        "email": "max@luv2code.com"
    },
    {
        "id": 2,
        "firstName": "John",
        "lastName": "Doe",
        "email": "john@luv2code.com"
    }
]
```


Customer REST Controller

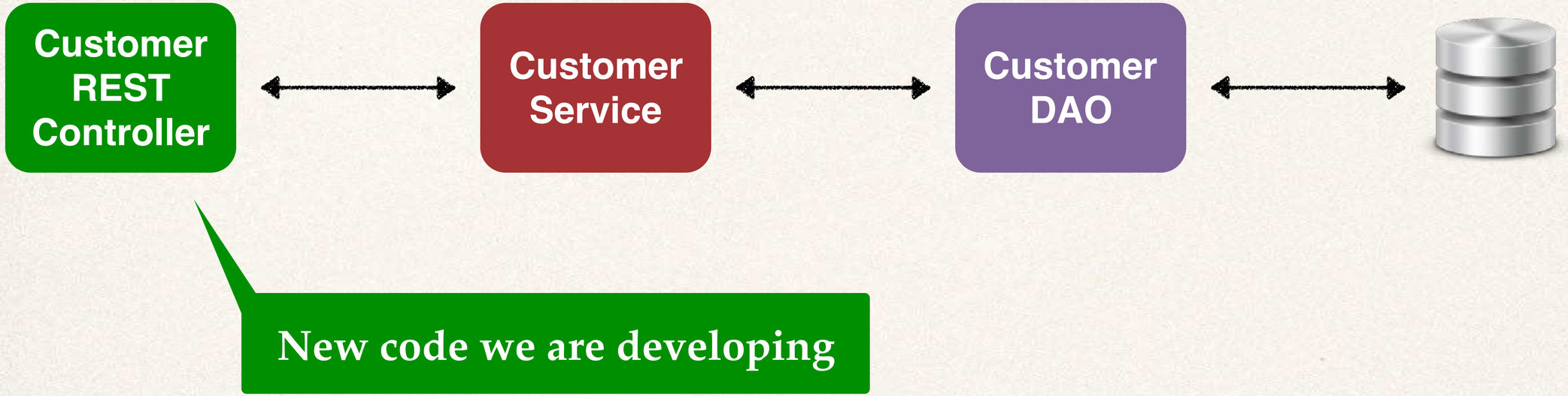
Application Architecture

Application Architecture

Application Architecture



Application Architecture



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create Customer REST Controller

Development Process

Step-By-Step

1. Create Customer REST Controller

2. Autowire CustomerService

Development Process

Step-By-Step

1. Create Customer REST Controller

2. Autowire CustomerService

3. Add mapping for GET /customers

Step 1: Create Customer REST Controller

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {



}
```

Step 2: Autowire CustomerService

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

}
```

Step 2: Autowire CustomerService

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;









}
```

Step 3: Add mapping for GET /customers

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;









}
```

Step 3: Add mapping for GET /customers

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;


  // add mapping for GET /customers
  @GetMapping("/customers")
  public List<Customer> getCustomers() {

    return customerService.getCustomers();

  }
}
```

Spring CRM REST - Get Single Customer

Application Interaction

Application Interaction

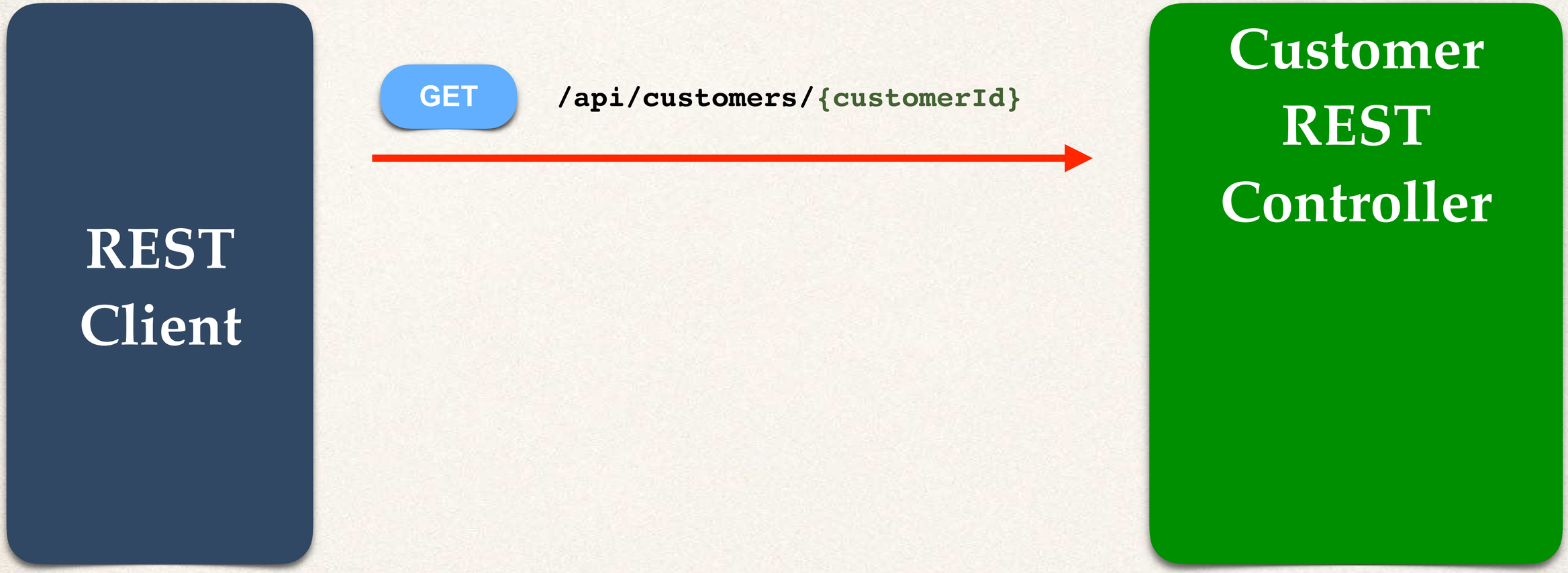
REST
Client

Application Interaction

REST Client

Customer REST Controller

Application Interaction



Application Interaction

REST Client

GET /api/customers/{customerId}

```
{
    "id": 1,
    "firstName": "David",
    "lastName": "Adams",
    "email": "david@luv2code.com"
}
```

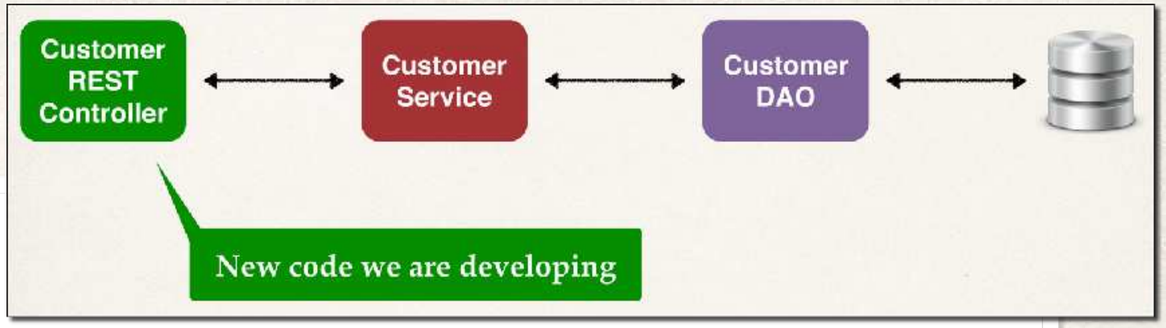
Customer REST Controller

Get Single Customer

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;
  …









}
```

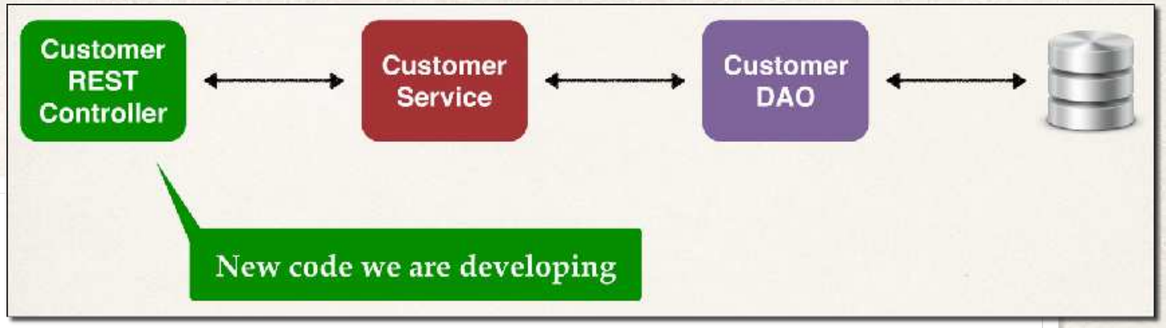
Get Single Customer



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;
  …

}
```

Get Single Customer



File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;
  …

  // add mapping for GET /customers/{customerId}

  @GetMapping("/customers/{customerId}")
  public Customer getCustomer(@PathVariable int customerId) {

    Customer theCustomer = customerService.getCustomer(customerId);

    return theCustomer;
  }

}
```

Spring CRM REST - Exception Handling

Application Interaction

Application Interaction

REST
Client

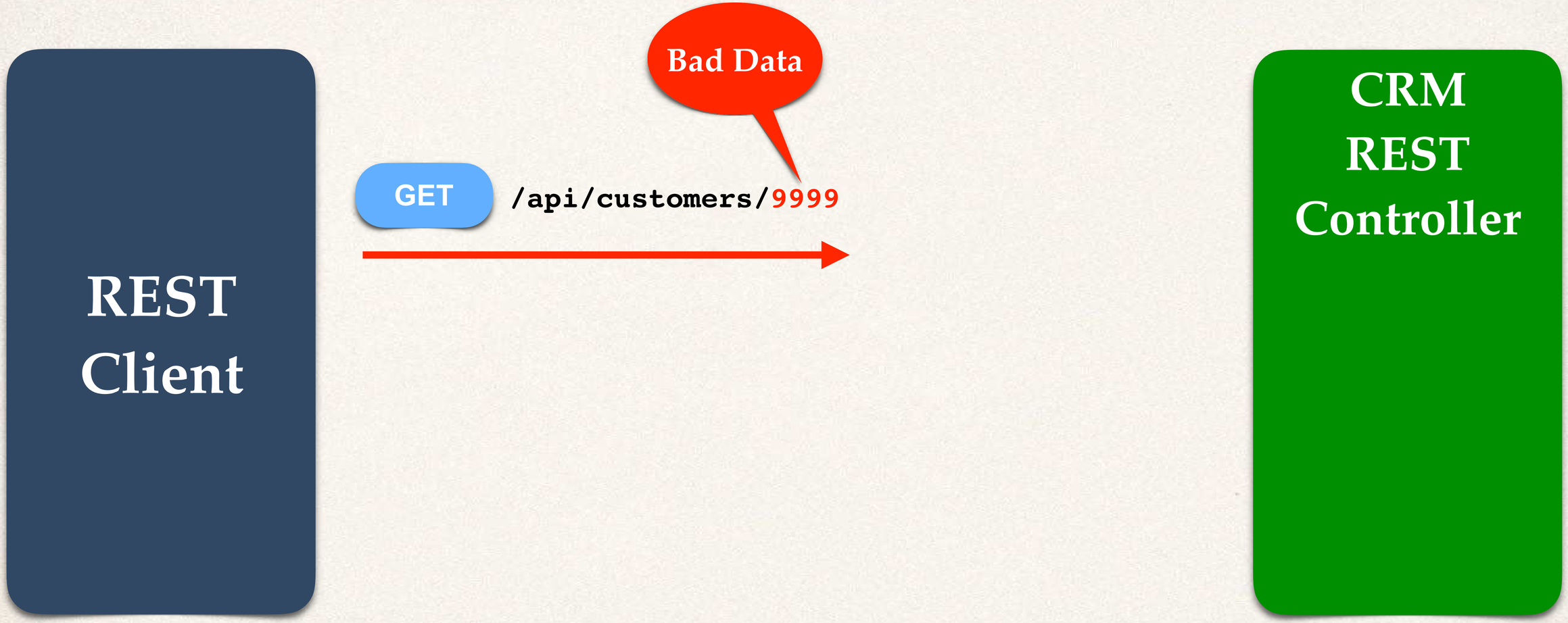
Application Interaction

REST
Client

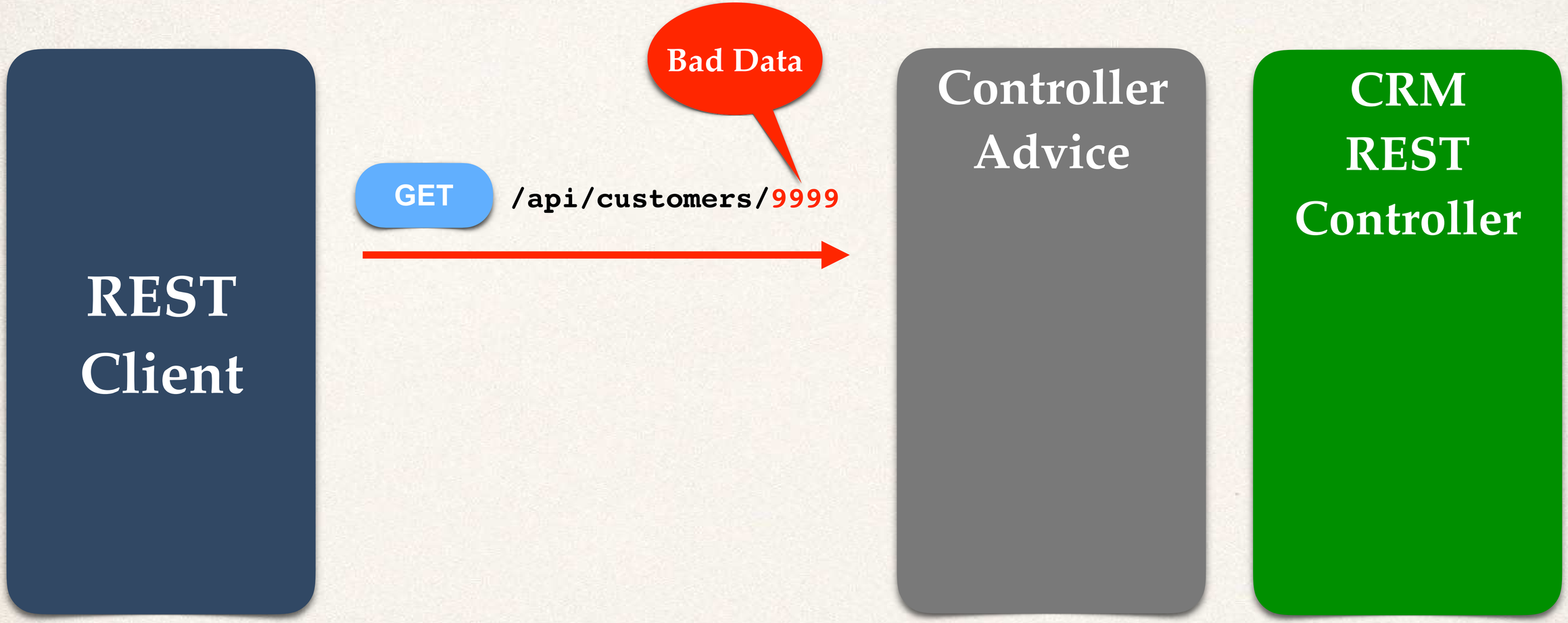
CRM
REST
Controller

Application Interaction

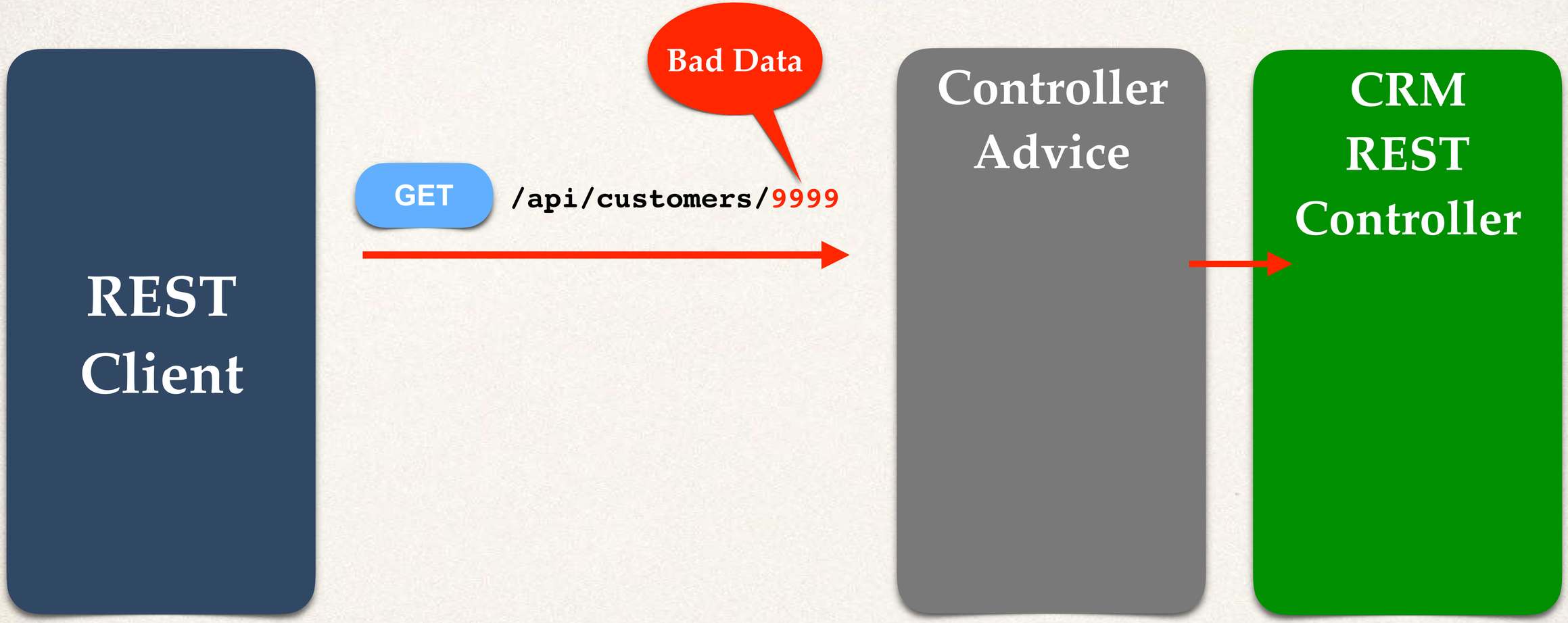
Application Interaction



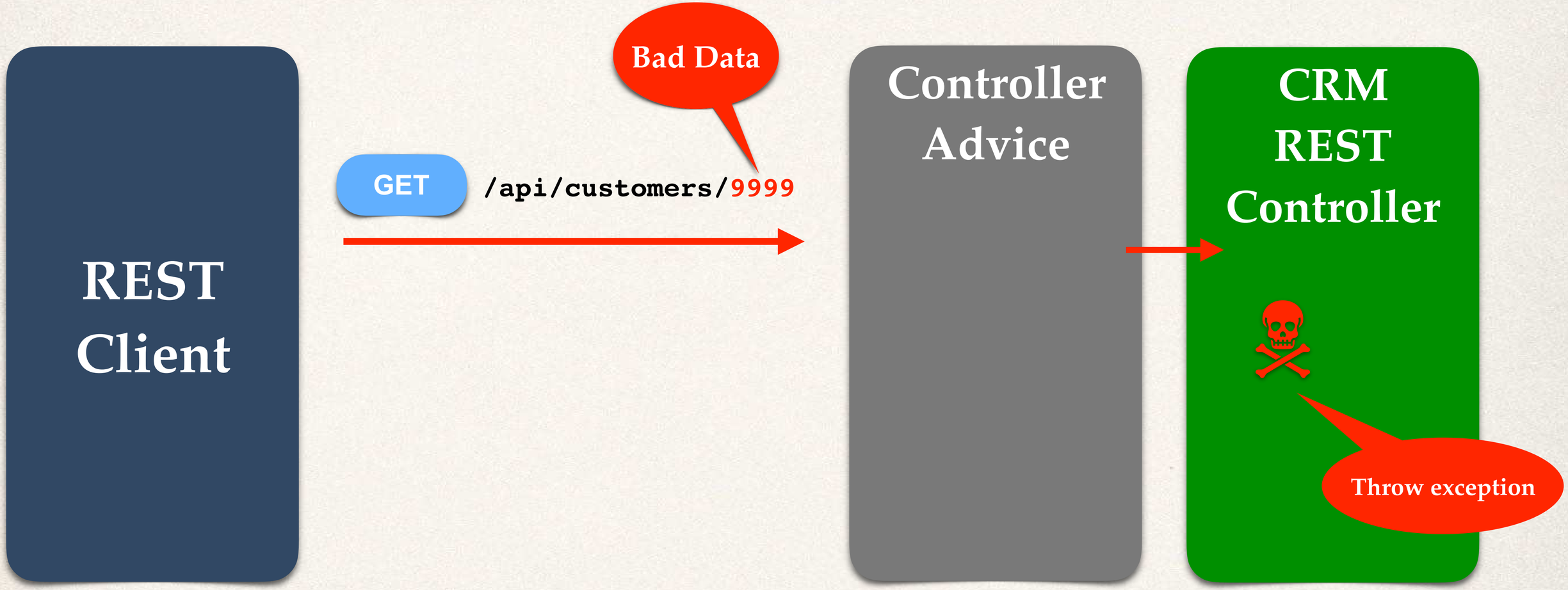
Application Interaction



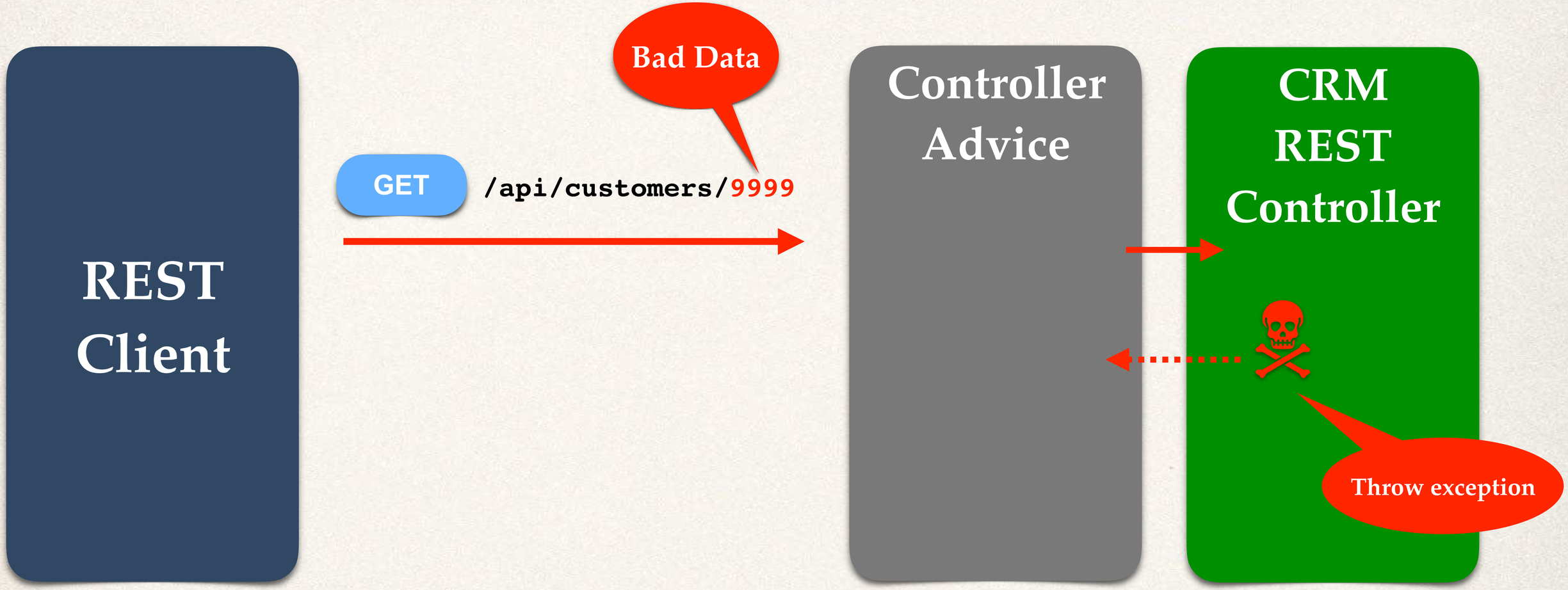
Application Interaction



Application Interaction

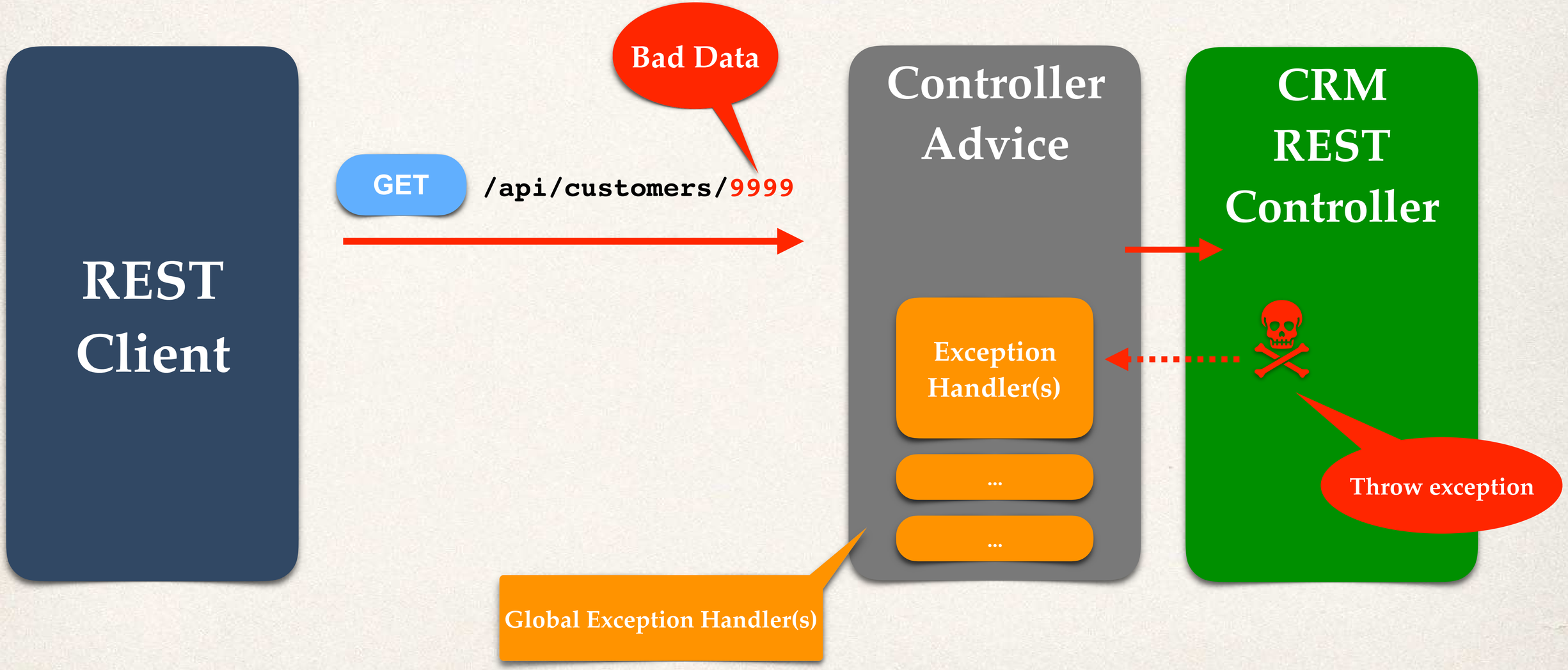


Application Interaction

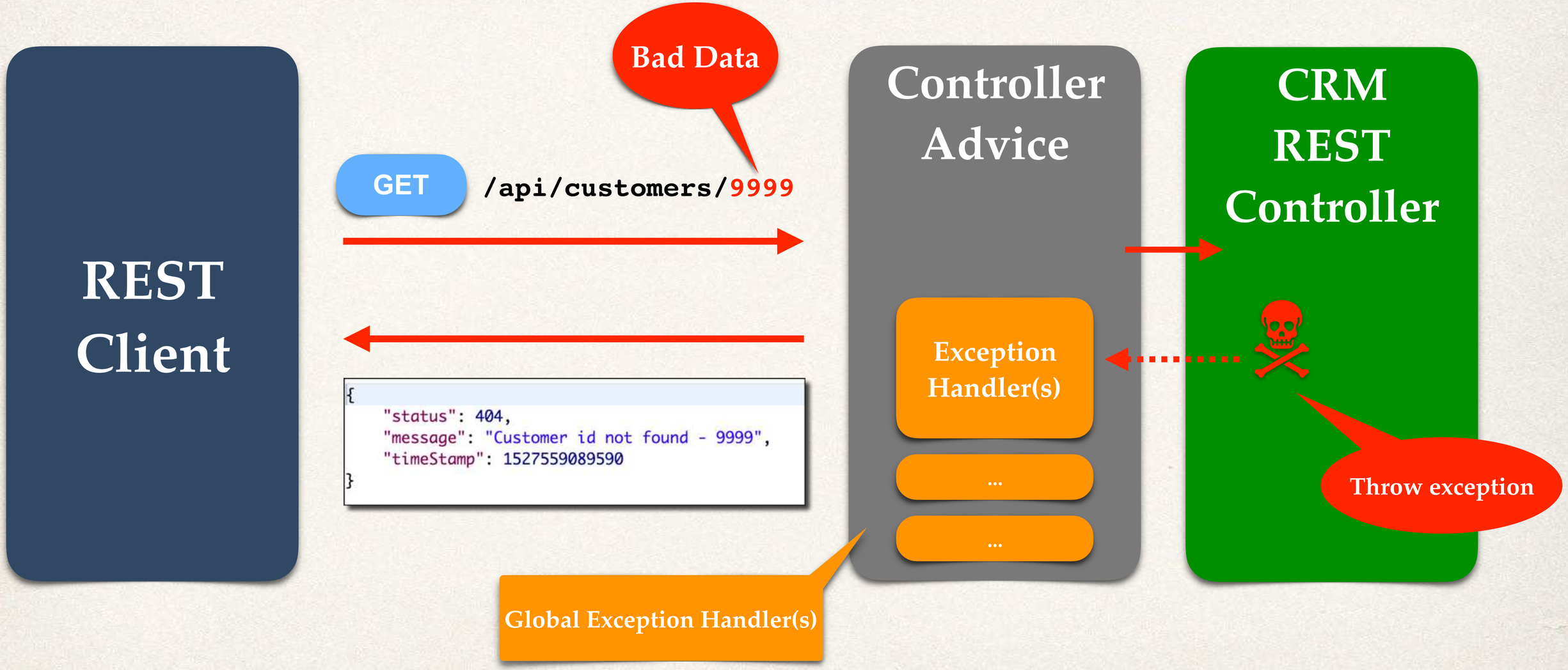


Application Interaction

Application Interaction



Application Interaction



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create a custom error response class

Development Process

Step-By-Step

1. Create a custom error response class

2. Create a custom exception class

Development Process

Step-By-Step

1. Create a custom error response class

2. Create a custom exception class

3. Update REST service to throw exception if customer not found

Development Process

Step-By-Step

1. Create a custom error response class

2. Create a custom exception class

3. Update REST service to throw exception if customer not found

4. Add an exception handler methods using `@ExceptionHandler`

Spring CRM REST - Add Customer

Real-Time Project

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
✓ GET	/api/customers	Read a list of customers
✓ GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Application Interaction

REST Client

Customer REST Controller

Application Interaction

REST Client

POST /api/customers

```
{
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

Customer REST Controller

Application Interaction

Since new customer, we are not passing id / primary key

REST Client

POST /api/customers

```
{
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

Customer REST Controller

Application Interaction

Since new customer, we are not passing id / primary key

REST Client

POST /api/customers

```
{
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

```
{
    "id": 7,
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

Customer REST Controller

Application Interaction

Since new customer, we are not passing id / primary key

REST Client

POST /api/customers

```
{
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

Customer REST Controller

```
{
    "id": 7,
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

Response contains new id / primary key

Access the Request Body

Access the Request Body

- Jackson will convert request body from JSON to POJO

Access the Request Body

- Jackson will convert request body from JSON to POJO

- **@RequestBody** annotation binds the POJO to a method parameter

Access the Request Body

- Jackson will convert request body from JSON to POJO

- **@RequestBody** annotation binds the POJO to a method parameter

```
@PostMapping("/customers")
public Customer addCustomer(@RequestBody Customer theCustomer) {

  ...

}
```

Access the Request Body

- Jackson will convert request body from JSON to POJO

- **@RequestBody** annotation binds the POJO to a method parameter

```
@PostMapping("/customers")
public Customer addCustomer(@RequestBody Customer theCustomer) {

  ...

}
```

Now we can access the request body as a POJO

Add Customer

File: CustomerRestController.java

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    …

  }
}
```

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    …

  }
}
```

Use @RequestBody to access the request body as a POJO

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

Delegate call to customer service

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

Return updated customer object

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

What's up with customer id?

What's up with customer id?

What's up with customer id?

- In the REST controller, we explicitly set the customer id to 0

What's up with customer id?

- In the REST controller, we explicitly set the customer id to 0
- Because our backend DAO code uses Hibernate method

What's up with customer id?

- In the REST controller, we explicitly set the customer id to 0
- Because our backend DAO code uses Hibernate method
  - `session.saveOrUpdate(….)`

Recall: CustomerDAOImpl

Recall: CustomerDAOImpl

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help luv2code

Java EE - web-customer-tracker/src/com/luv2code/springdemo/dao/CustomerDAOImpl.java - Eclipse - /Users/luv2code/dev-spring

list-customers.j | CustomerControll | CustomerService.j | CustomerServiceI | CustomerDAO.java | Customer

```
36          return customers;
37      }
38
39      @Override
40      public void saveCustomer(Customer theCustomer) {
41
42          // get current hibernate session
43          Session currentSession = sessionFactory.getCurrentSession();
44
45          // save the customer ... finally LOL
46          currentSession.saveOrUpdate(theCustomer);
47
48      }
49
```

saveOrUpdate(...)

if (primaryKey / id) empty

then INSERT new customer

else UPDATE exiting customer

Recall: CustomerDAOImpl

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help luv2code

Java EE - web-customer-tracker/src/com/luv2code/springdemo/dao/CustomerDAOImpl.java - Eclipse - /Users/luv2code/dev-spring

list-customers.j CustomerControll CustomerService.j CustomerServicel CustomerDAO.java Customer

```
36          return customers;
37      }
38
39      @Override
40      public void saveCustomer(Customer theCustomer) {
41
42          // get current hibernate session
43          Session currentSession = sessionFactory.getCurrentSession();
44
45          // save the customer ... finally LOL
46          currentSession.saveOrUpdate(theCustomer);
47
48      }
49
```

saveOrUpdate(...)

if (primaryKey / id) empty

then INSERT new customer

else UPDATE exiting customer

Recall: CustomerDAOImpl

Here: “empty” means null or 0

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help luv2code

Java EE - web-customer-tracker/src/com/luv2code/springdemo/dao/CustomerDAOImpl.java - Eclipse - /Users/luv2code/dev-spring

list-customers.j CustomerControll CustomerService.j CustomerServicel CustomerDAO.java Customer

```
36          return customers;
37      }
38
39      @Override
40      public void saveCustomer(Customer theCustomer) {
41
42          // get current hibernate session
43          Session currentSession = sessionFactory.getCurrentSession();
44
45          // save the customer ... finally LOL
46          currentSession.saveOrUpdate(theCustomer);
47
48      }
```

saveOrUpdate(...)

if (primaryKey / id) empty

then INSERT new customer

else UPDATE exiting customer

Adding customer with HTTP POST

Adding customer with HTTP POST

- If REST client is sending a request to “add”, using HTTP POST

Adding customer with HTTP POST

- If REST client is sending a request to “add”, using HTTP POST
- Then we ignore any id sent in the request

Adding customer with HTTP POST

- If REST client is sending a request to “add”, using HTTP POST
- Then we ignore any id sent in the request
- We overwrite the id with 0, to effectively set it to null/empty

Adding customer with HTTP POST

- If REST client is sending a request to “add”, using HTTP POST
- Then we ignore any id sent in the request
- We overwrite the id with 0, to effectively set it to null/empty
- Then our backend DAO code will “INSERT” new customer

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

ID of 0 means DAO code will "INSERT" new customer

Sending JSON to Spring REST Controllers

Sending JSON to Spring REST Controllers

- When sending JSON data to Spring REST Controllers

Sending JSON to Spring REST Controllers

- When sending JSON data to Spring REST Controllers
- For controller to process JSON data, need to set a HTTP request header

Sending JSON to Spring REST Controllers

- When sending JSON data to Spring REST Controllers

- For controller to process JSON data, need to set a HTTP request header
  - `Content-type: application/json`

Sending JSON to Spring REST Controllers

- When sending JSON data to Spring REST Controllers
- For controller to process JSON data, need to set a HTTP request header
  - `Content-type: application/json`
- Need to configure REST client to send the correct HTTP request header

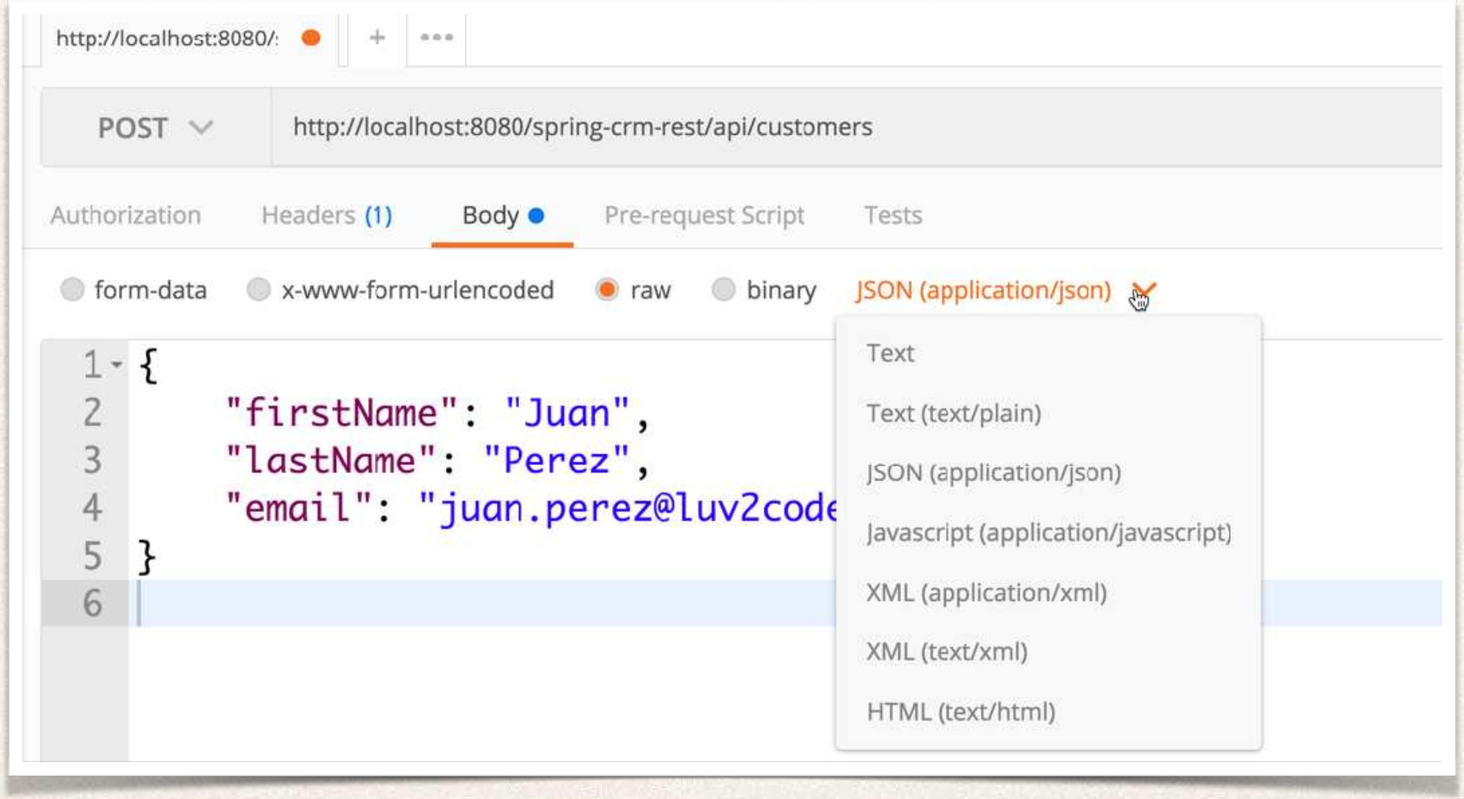
Postman - Sending JSON in Request Body

Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman

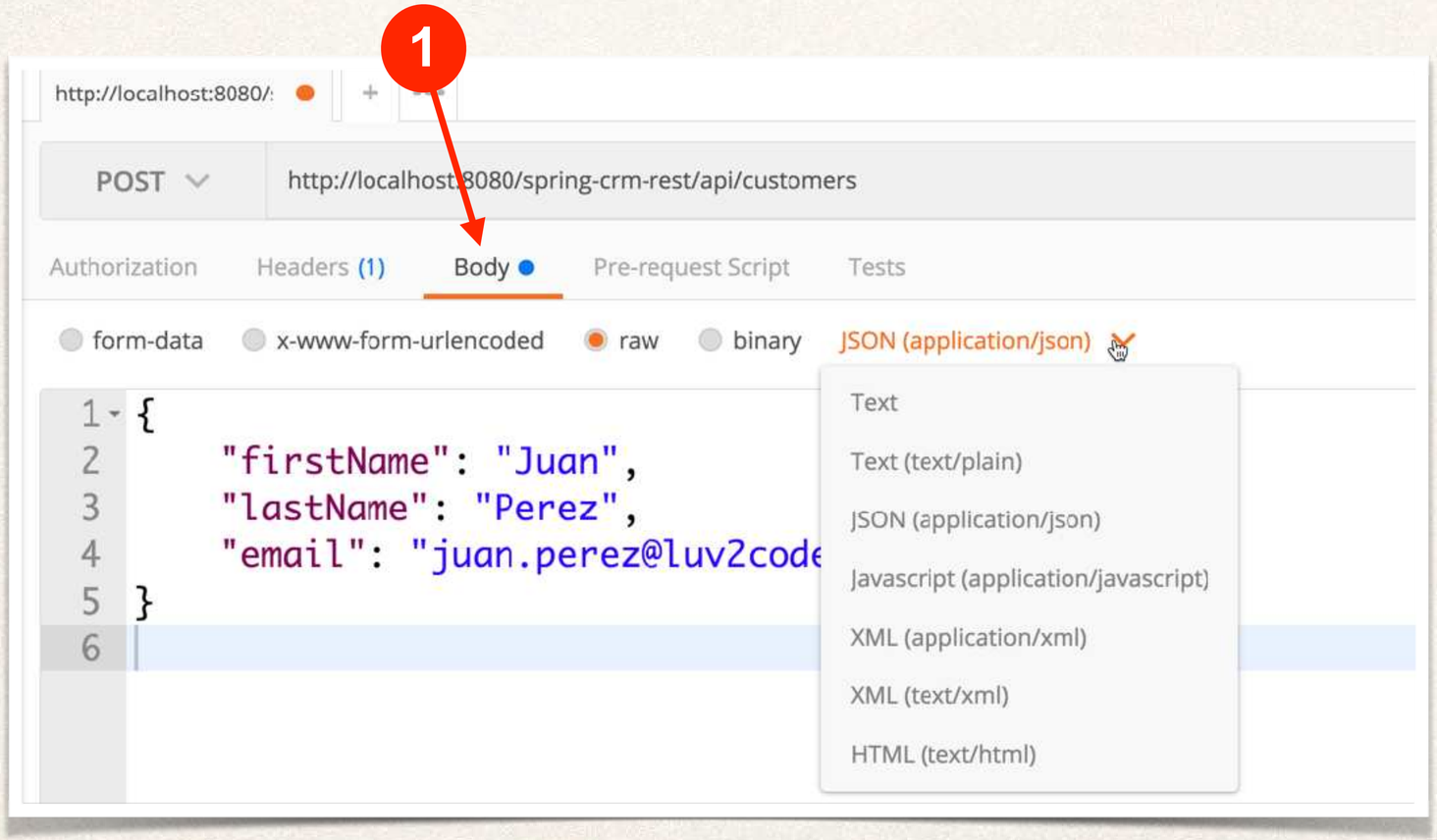
Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman



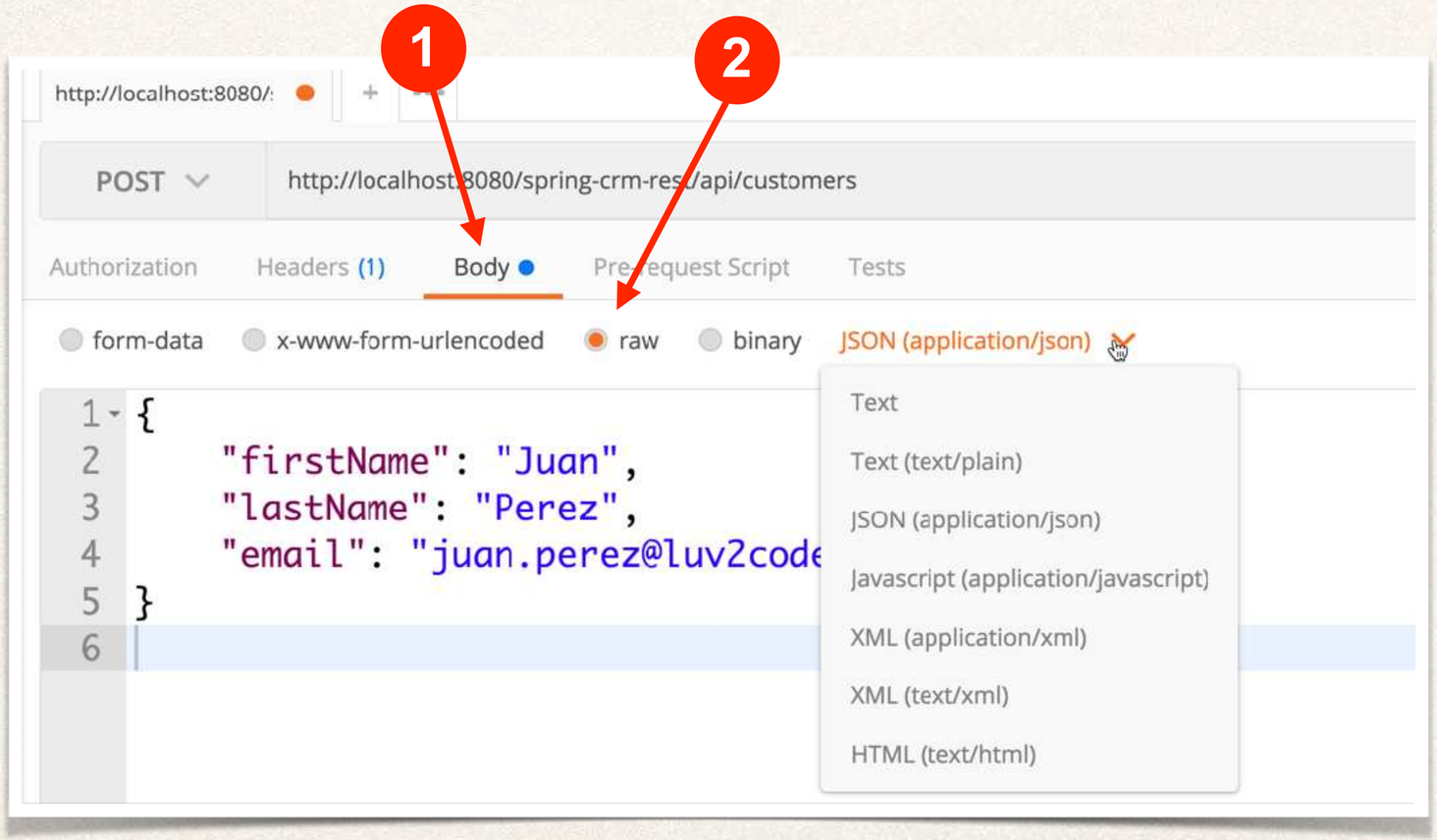
Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman



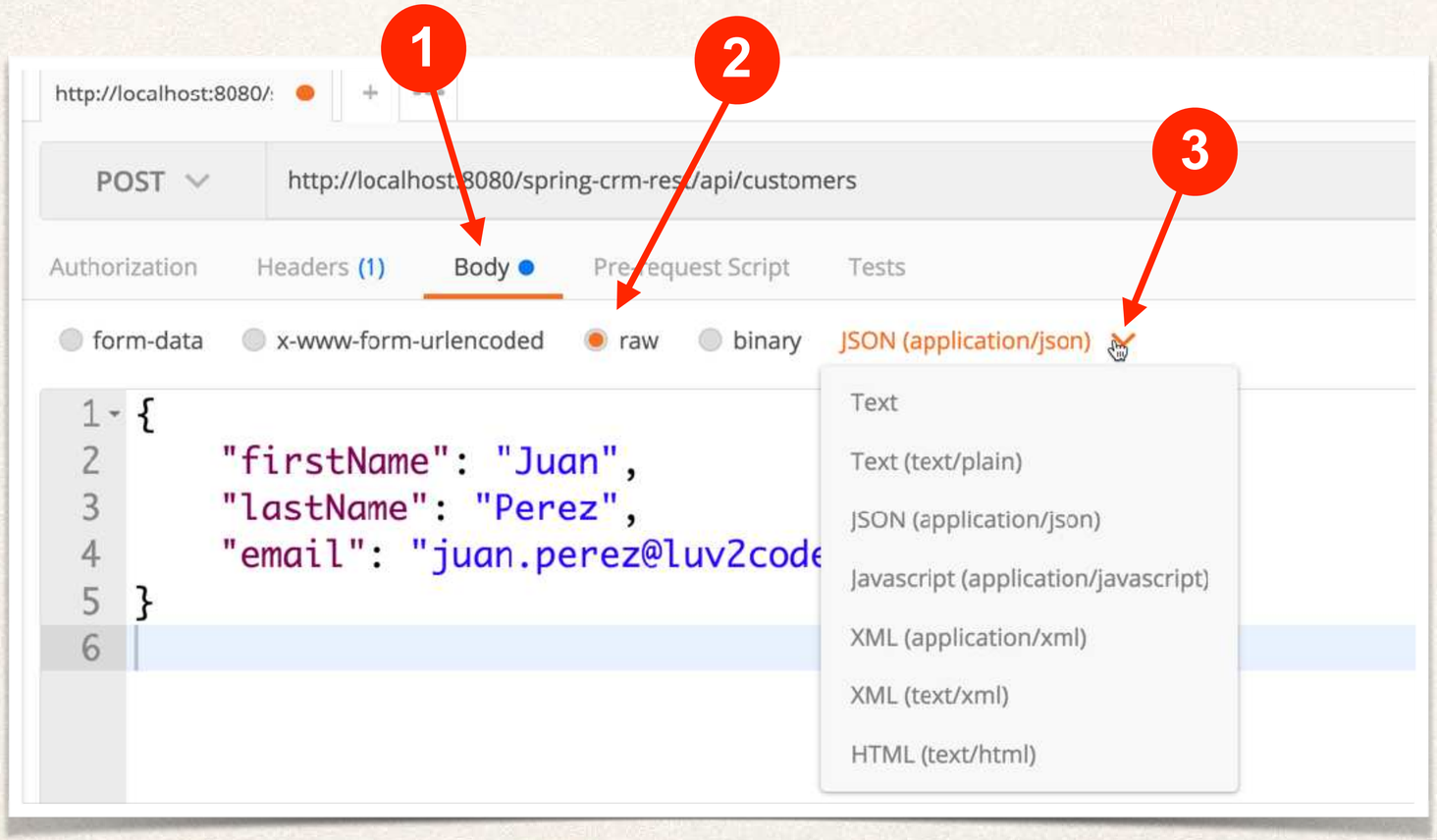
Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman



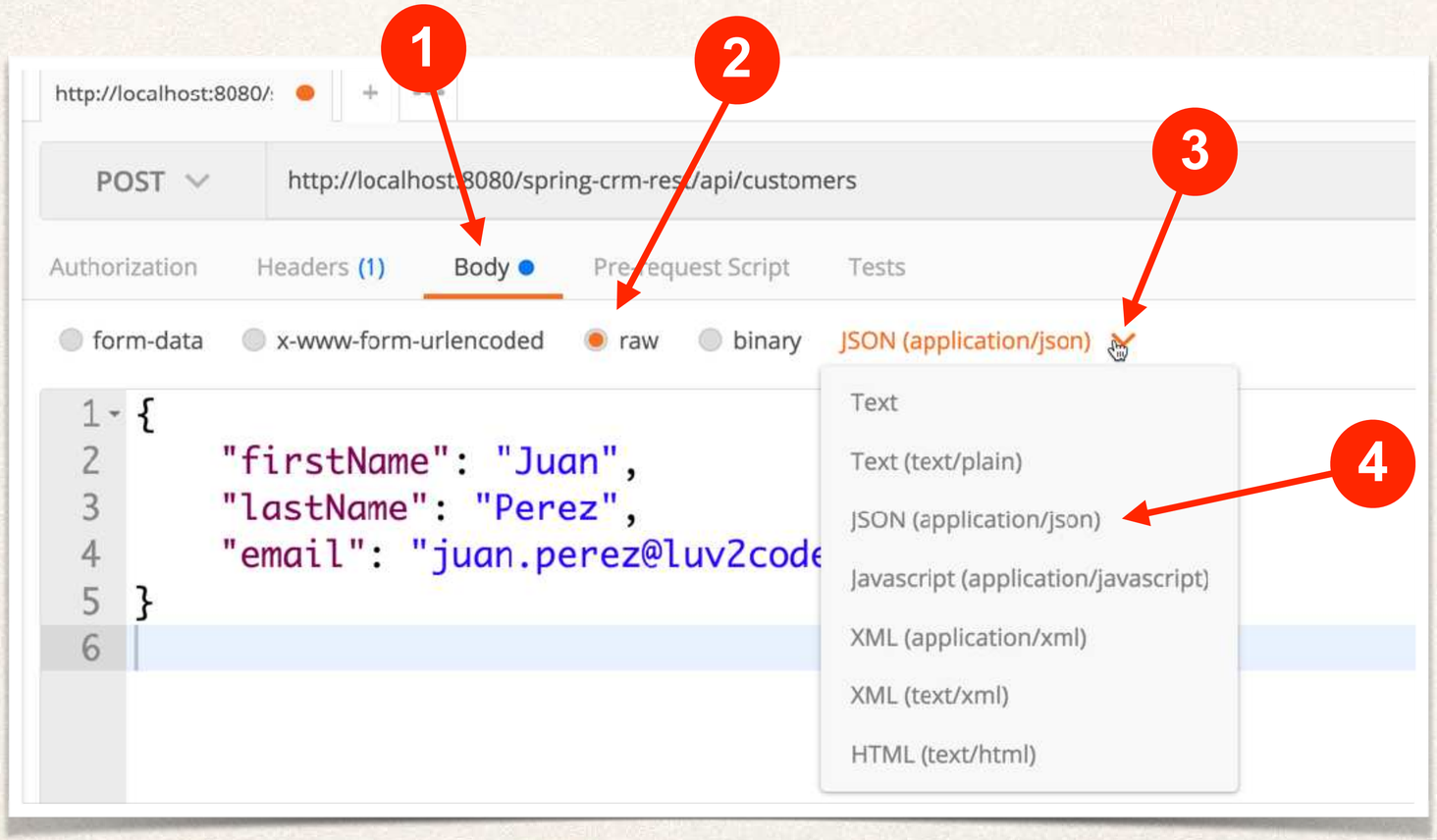
Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman



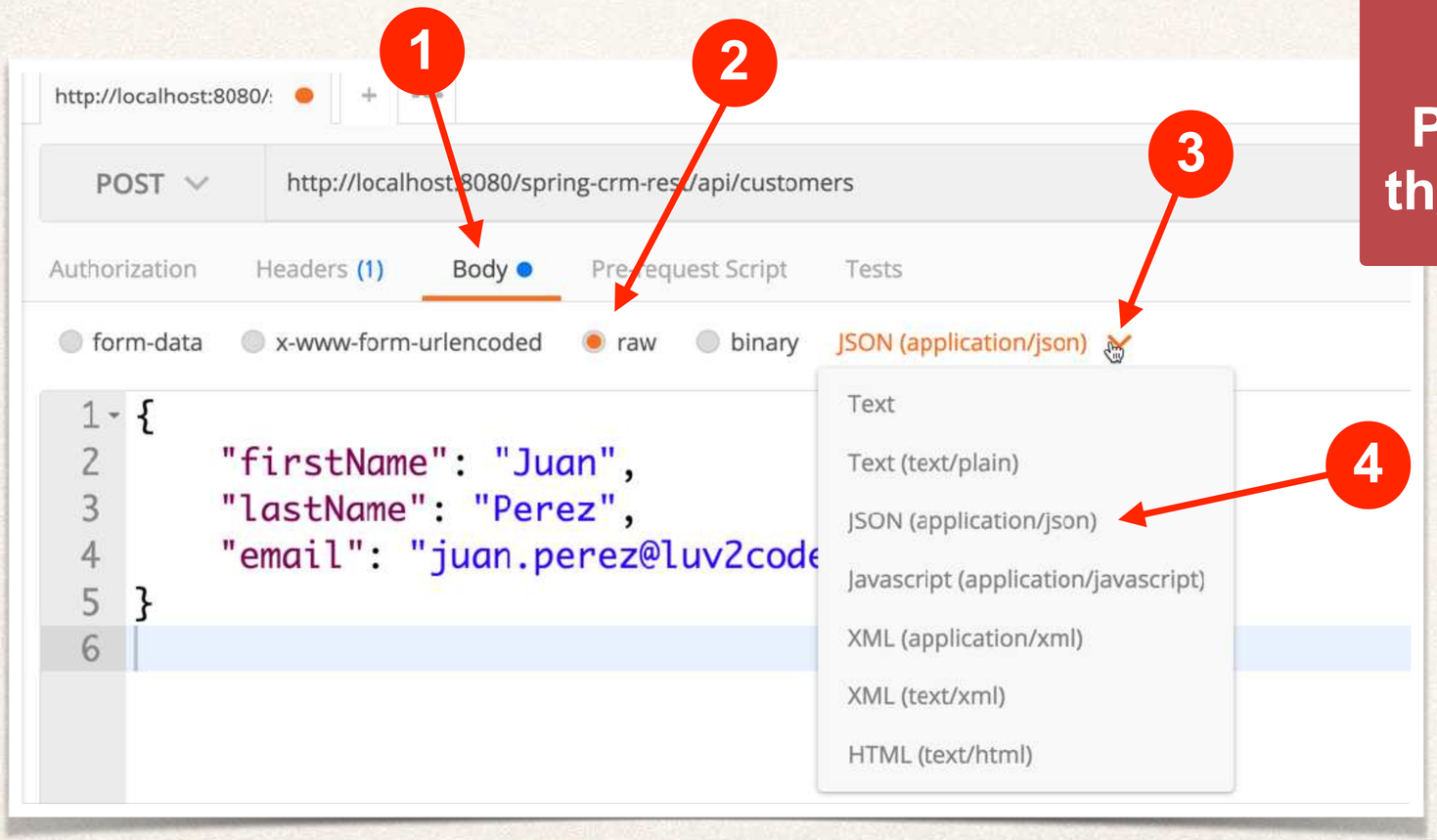
Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman



Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman



Based on these configs, Postman will automatically set the correct HTTP request header

Spring CRM REST - Update Customer

Real-Time Project

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
✓ POST	/api/customers	Create a new customer
✓ GET	/api/customers	Read a list of customers
✓ GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Application Interaction

REST Client

Customer REST Controller

Application Interaction

REST Client

PUT /api/customers

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

Customer REST Controller

Application Interaction

ID of customer to update
With updated info

REST Client

PUT /api/customers

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

Customer REST Controller

Application Interaction

ID of customer to update
With updated info

REST Client

PUT /api/customers

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

Customer REST Controller

Application Interaction

ID of customer to update
With updated info

REST Client

PUT /api/customers

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

Customer REST Controller

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

Response contains updated info (echoed)

Update Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for PUT /customers  - update existing customer

  @PutMapping("/customers")
  public Customer updateCustomer(@RequestBody Customer theCustomer) {




  }

}
```

Update Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for PUT /customers  - update existing customer

  @PutMapping("/customers")
  public Customer updateCustomer(@RequestBody Customer theCustomer) {

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }

}
```

Spring CRM REST - Delete Customer

Real-Time Project

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
✓ POST	/api/customers	Create a new customer
✓ GET	/api/customers	Read a list of customers
✓ GET	/api/customers/{customerId}	Read a single customer
✓ PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Real-Time Project

Checkpoint

HTTP Method		CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Application Interaction

REST Client

Customer REST Controller

Application Interaction

Application Interaction

Delete Customer

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for DELETE /customers  - delete customer

  @DeleteMapping("/customers/{customerId}")
  public String deleteCustomer(@PathVariable int customerId) {




  }

}
```

Delete Customer

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for DELETE /customers  - delete customer

  @DeleteMapping("/customers/{customerId}")
  public String deleteCustomer(@PathVariable int customerId) {

    customerService.deleteCustomer(customerId);

    return "Deleted customer id - " + customerId;
  }

}
```

Delete Customer - Refactor

File: StudentRestController.java

```
@DeleteMapping("/customers/{customerId}")
public String deleteCustomer(@PathVariable int customerId) {

  Customer tempCustomer = customerService.getCustomer(customerId);

  // throw exception if null

  if (tempCustomer == null) {
    throw new CustomerNotFoundException("Customer id not found - " + customerId);
  }

  customerService.deleteCustomer(customerId);

  return "Deleted customer id - " + customerId;
}
```

Delete Customer - Refactor

File: StudentRestController.java

```
@DeleteMapping("/customers/{customerId}")
public String deleteCustomer(@PathVariable int customerId) {

  Customer tempCustomer = customerService.getCustomer(customerId);

  // throw exception if null

  if (tempCustomer == null) {
    throw new CustomerNotFoundException("Customer id not found - " + customerId);
  }

  customerService.deleteCustomer(customerId);

  return "Deleted customer id - " + customerId;
}
```

Delete Customer - Refactor

Check to see if customer exists

If not, throw exception

File: StudentRestController.java

```
@DeleteMapping("/customers/{customerId}")
public String deleteCustomer(@PathVariable int customerId) {

  Customer tempCustomer = customerService.getCustomer(customerId);

  // throw exception if null

  if (tempCustomer == null) {
    throw new CustomerNotFoundException("Customer id not found - " + customerId);
  }

  customerService.deleteCustomer(customerId);

  return "Deleted customer id - " + customerId;
}
```

Delete Customer - Refactor

File: StudentRestController.java

Check to see if customer exists

If not, throw exception

```
@DeleteMapping("/customers/{customerId}")
public String deleteCustomer(@PathVariable int customerId) {

  Customer tempCustomer = customerService.getCustomer(customerId);

  // throw exception if null

  if (tempCustomer == null) {
    throw new CustomerNotFoundException("Customer id not found - " + customerId);
  }

  customerService.deleteCustomer(customerId);

  return "Deleted customer id - " + customerId;
}
```

Happy Path