Bộ môn: Kỹ Thuật Phần Mềm

Lập trình WWW *Java*

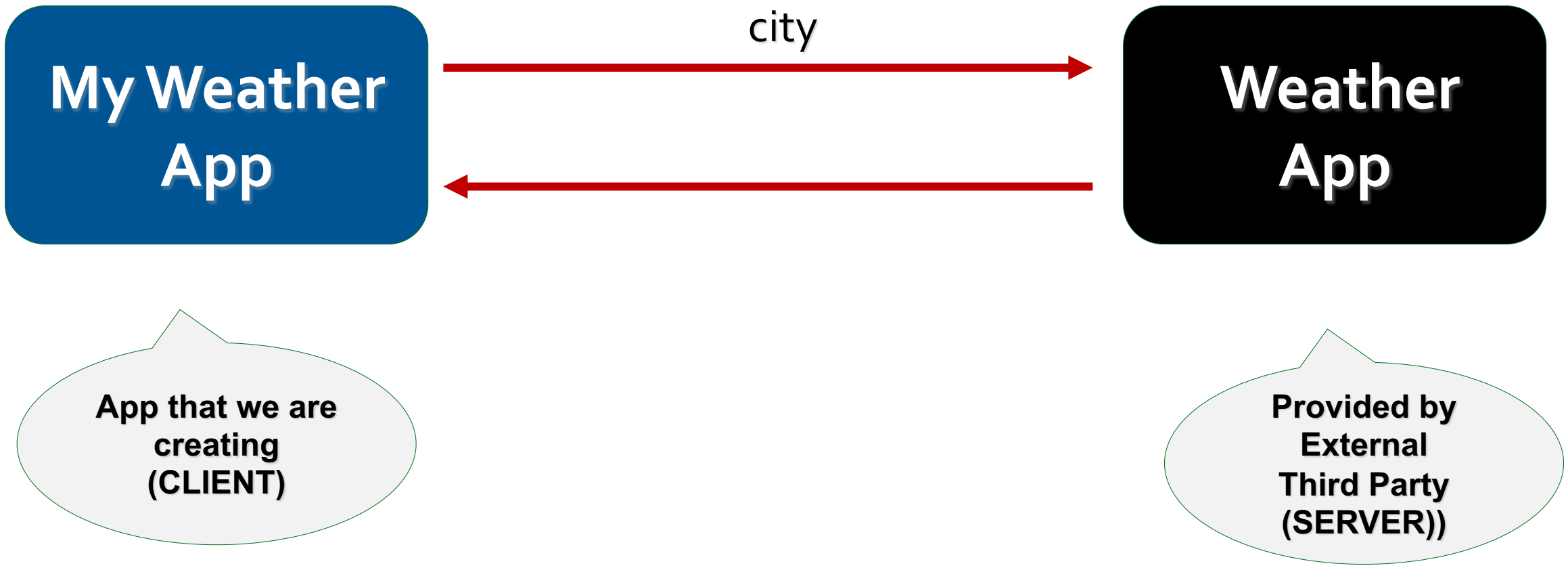
Spring - REST

REST APIs - REST Web Services

Business Problem

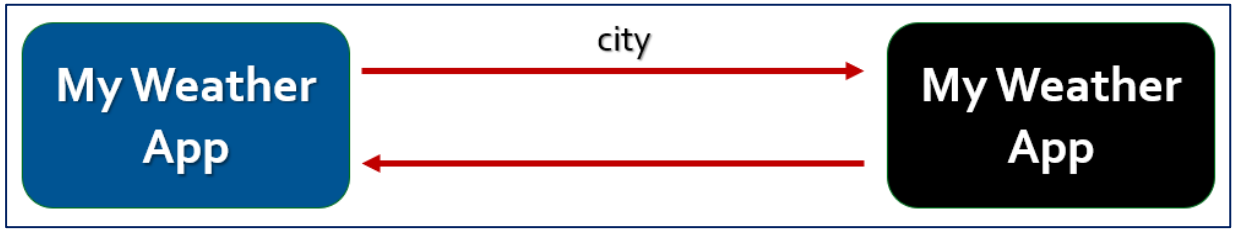
- Build a **client app** that provides the **weather report** for a city
- Need to get **weather data** from **an external service**

Application Architecture



Questions

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

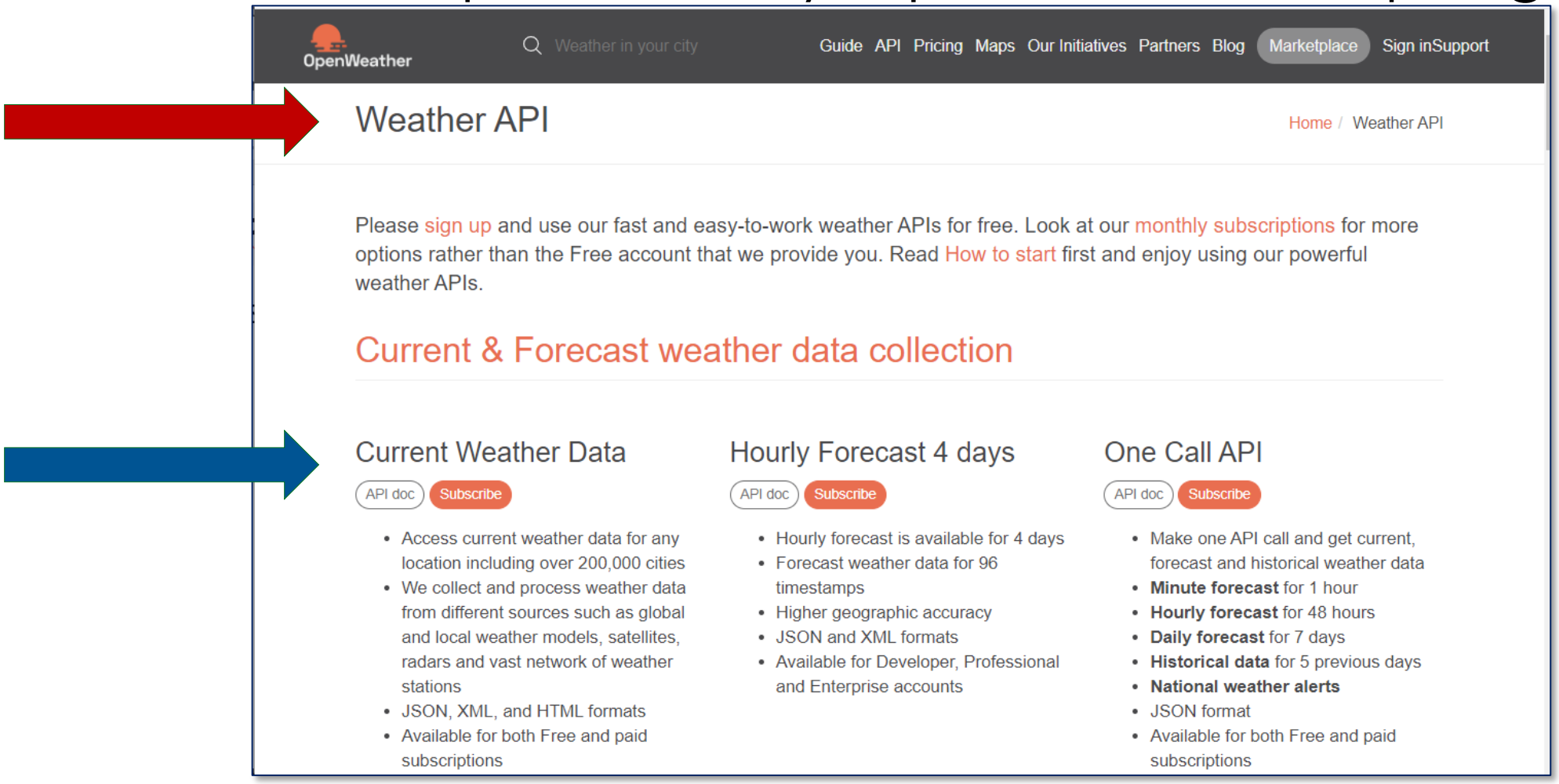


Questions

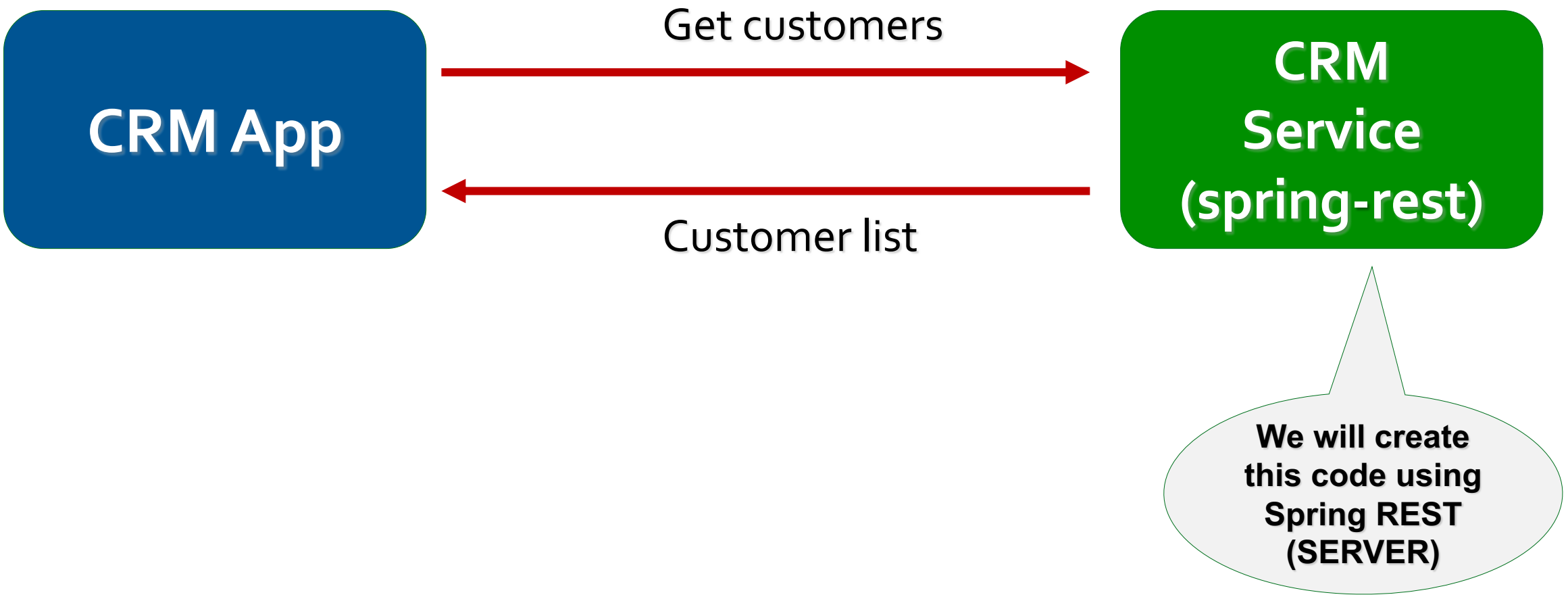
- How will we **connect** to the **Weather Service**?
  - make REST (REpresentational State Transfer) API **calls** over HTTP. REST is lightweight approach for communicating between applications
- What **programming language** do we use?
  - **REST** is language independent. The client/server application can use ANY programming language
- What is the **data format**?
  - **REST** applications can use any data format. Commonly see **XML** and **JSON**
  - JSON is most popular and modern

Possible Solution

Use free Weather Service provided by: openweathermap.org



Customer Relationship Manager (CRM) App



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

JSON Basics

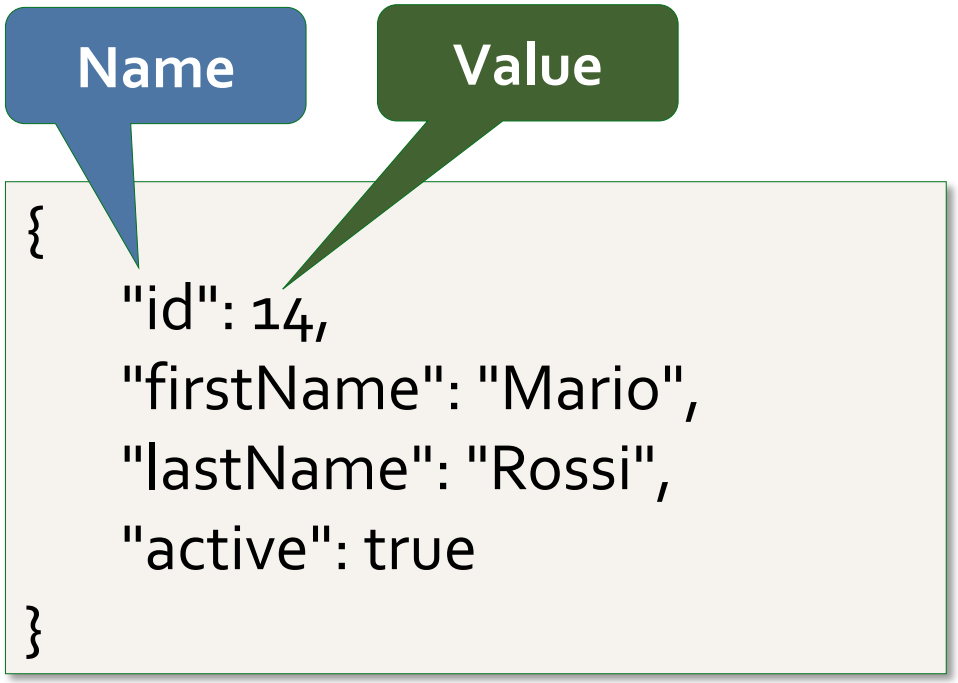
What is JSON?

- **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

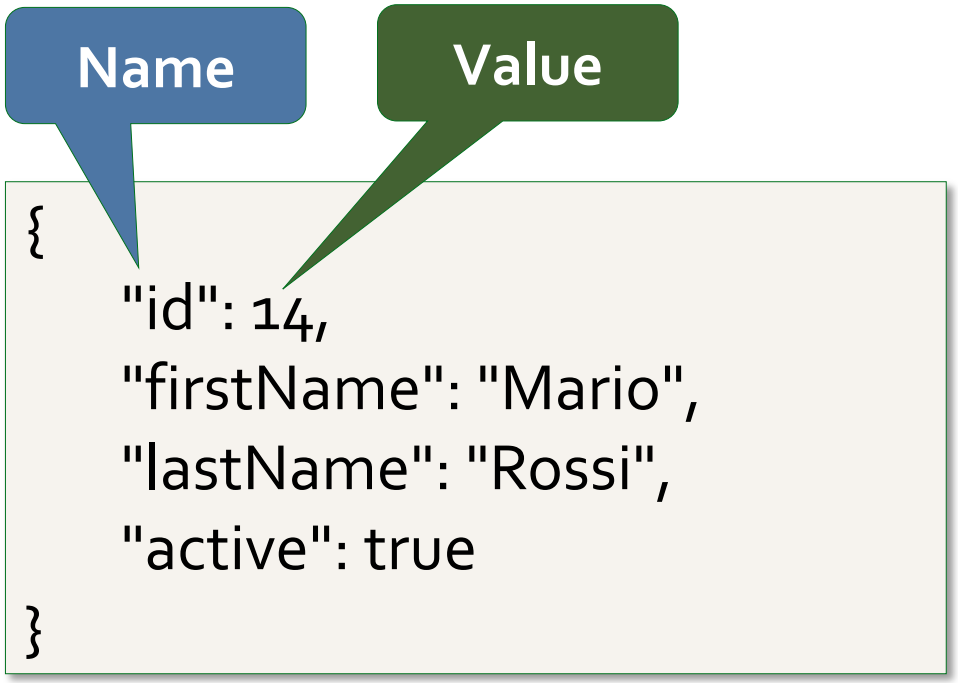
Simple JSON Example

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



JSON Values

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



Nested JSON Objects


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

JSON Arrays


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

Java JSON Data Binding

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

- Spring uses the **Jackson Project** behind the scenes
- Jackson **handles data binding** between **JSON** and **Java POJO**
- **Jackson Data Binding** API:
  - Package: **`com.fasterxml.jackson.databind`**

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

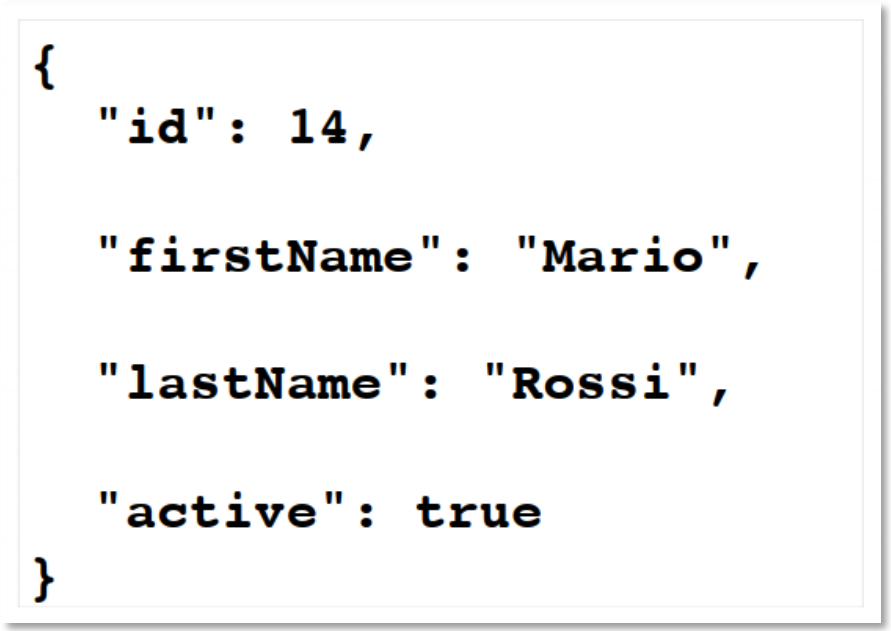
JSON to Java POJO (project: spring-REST-jackson-databinding-json-demo)

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

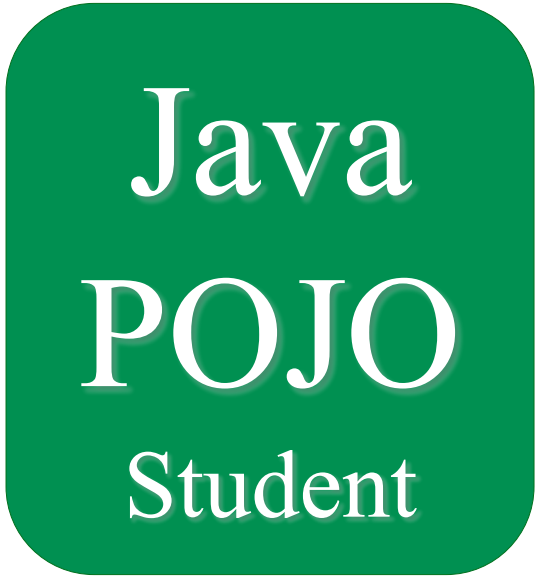
```
{
   "id": 14,

   "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
}
```

Java POJO to JSON



Call getXXX methods



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);
```

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

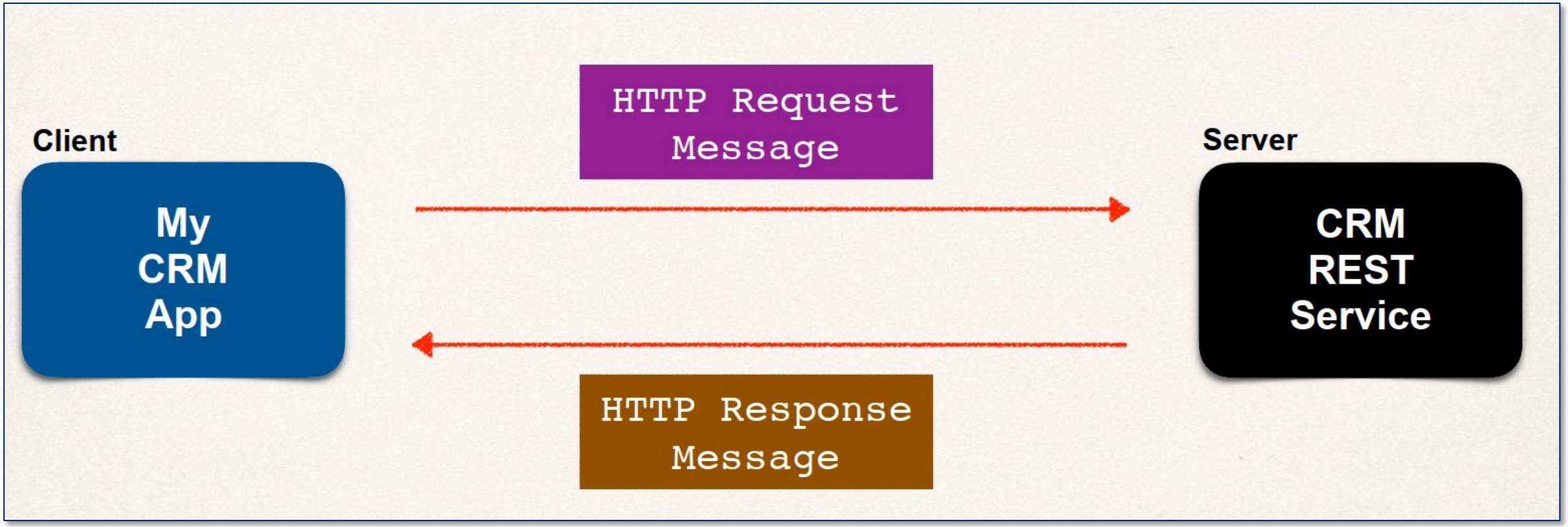
REST over HTTP

» Most **common use of REST** is over **HTTP**

» Leverage **HTTP methods for CRUD operations**

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

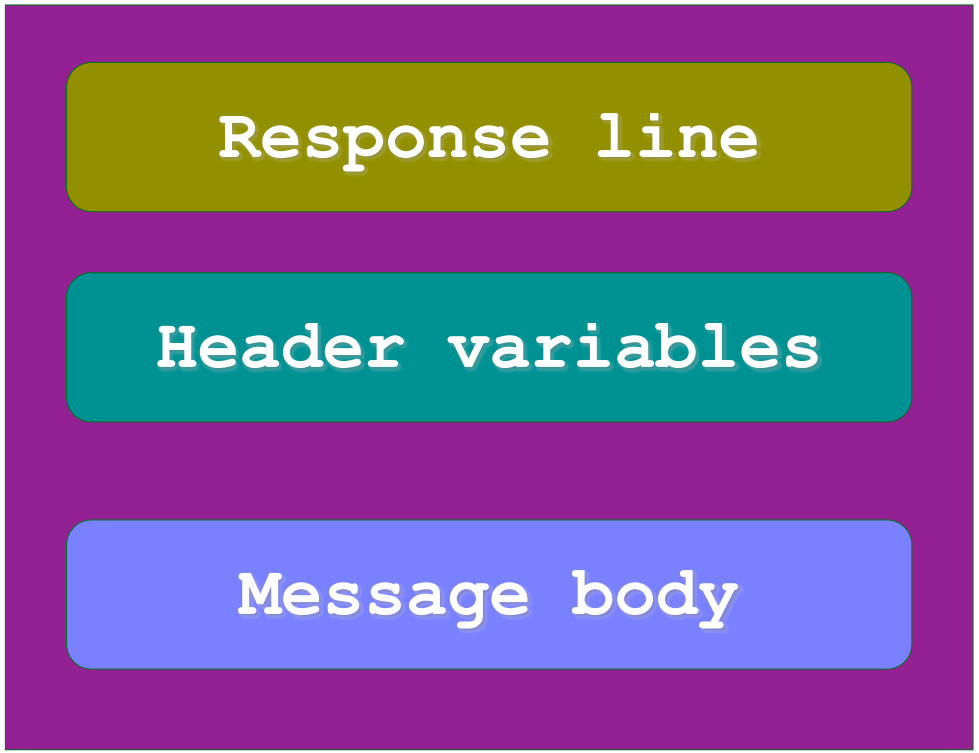


HTTP Request Message

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

HTTP Response Message

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



HTTP Response - Status Codes

HTTP Method	CRUD Operation
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

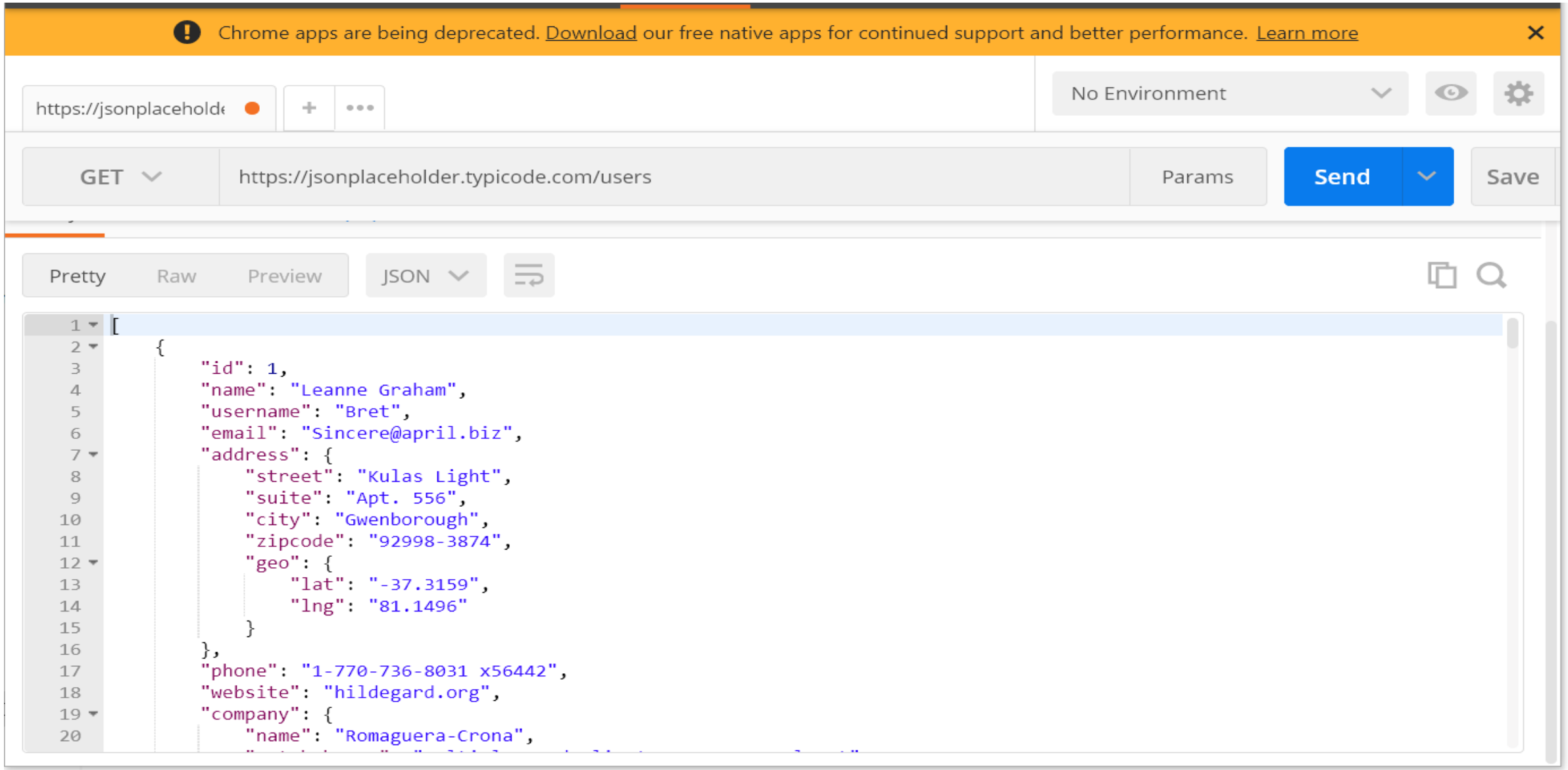
MIME Content Types

- The message format is described by MIME content type
- Multipurpose Internet Mail-Extension
- Basic Syntax: type/sub-type
- Examples:
  - **`text/html, text/plain`**
  - **`application/json, application/xml, …`**

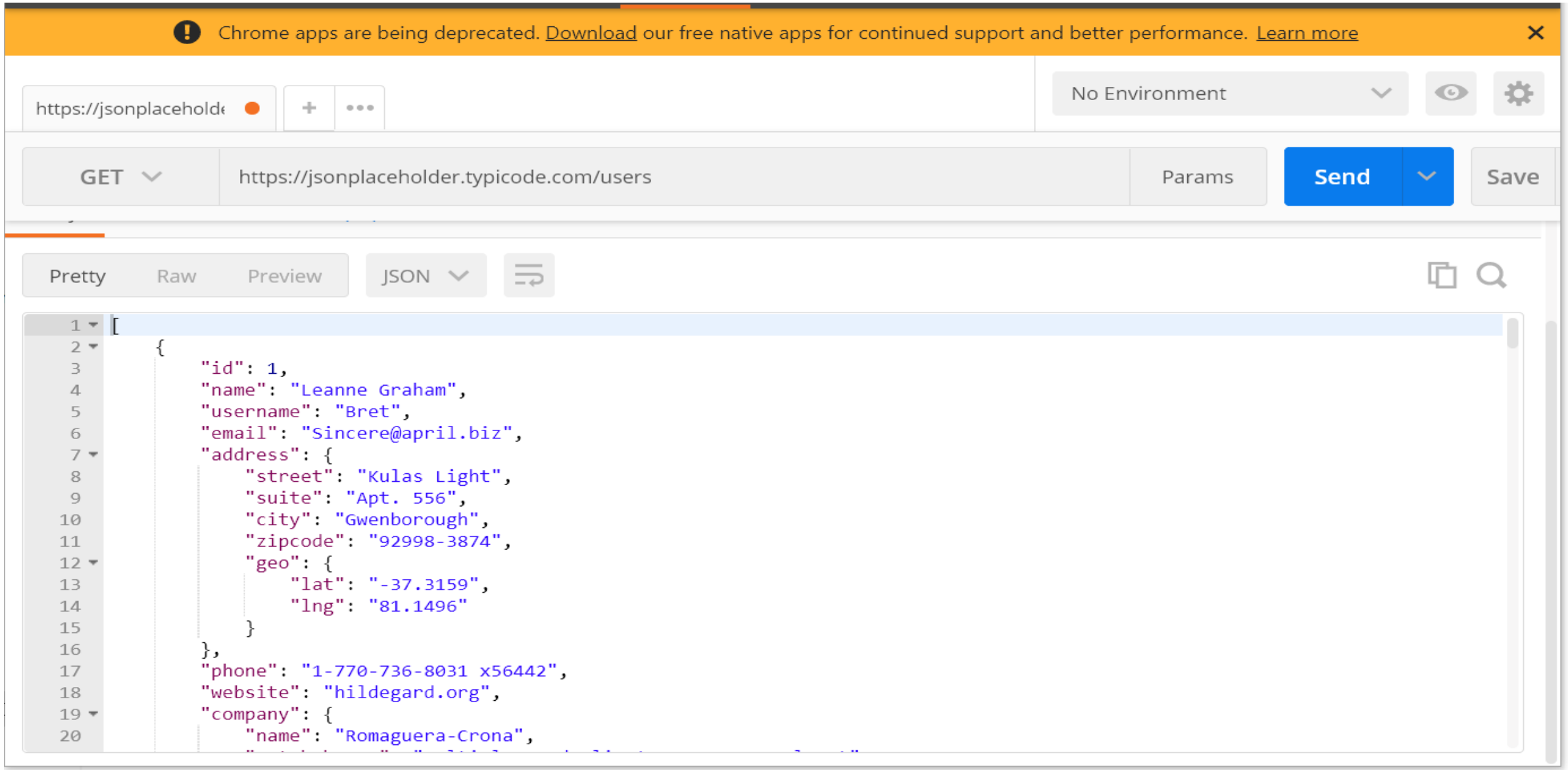
Client Tool

- We need a **client tool**
- Send HTTP requests to the REST Web Service / API
- Plenty of tools available: **curl**, **Postman**, etc…
- Website use as a REST Web Service / APIset server for test postman: **`JSONPlaceholder.typicode.com`**

Client Tool Postman



Client Tool Postman



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

Create Spring REST Controller –Development Process

- Add **Maven dependency** for **Spring MVC** and **Jackson project**
- Add **code** for **All Java Config**: @Configuration
- Add **code** for **All Java Config**: Servlet Initializer
- Create **Spring REST Service** using @RestController

Step 1: Add Maven dependency

File: pom.xml (project: spring-rest-demo-hello-world)

```
<!-- Add Spring MVC and REST support -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.0.5.RELEASE</version>
    </dependency>
    <!-- Add Jackson for JSON converters -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.5</version>
    </dependency>
```

```
<!-- Add Servlet support for
  Spring's AbstractAnnotationConfigDispatcherServletInitializer -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
<!-- Add support for JSP ... get rid of Eclipse error -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.1</version>
</dependency>
```

Step 2: All Java Config: @Configuration

File: DemoAppConfig.java(project: spring-rest-demo-hello-world)

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
@ComponentScan("com.se.spring")
public class DemoAppConfig implements WebMvcConfigurer {


}
```

Step 3: All Java Config: Servlet Initializer

Web App Initializer:

» Spring MVC provides support for **web app initialization**

» This is **makes sure** your code is **automatically detected**

and

» Your code is **used to initialize the servlet container**

Step 3: All Java Config: Servlet Initializer

Web App Initializer:

Todo List:

```
AbstractAnnotationConfigDispatcherServletInitializer
```

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

Step 3: All Java Config: Servlet Initializer

Web App Initializer:

File: MySpringMvcDispatcherServletInitializer.java(project: spring-rest-demo-hello-world)

```
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[] { "/" };    }
}
```

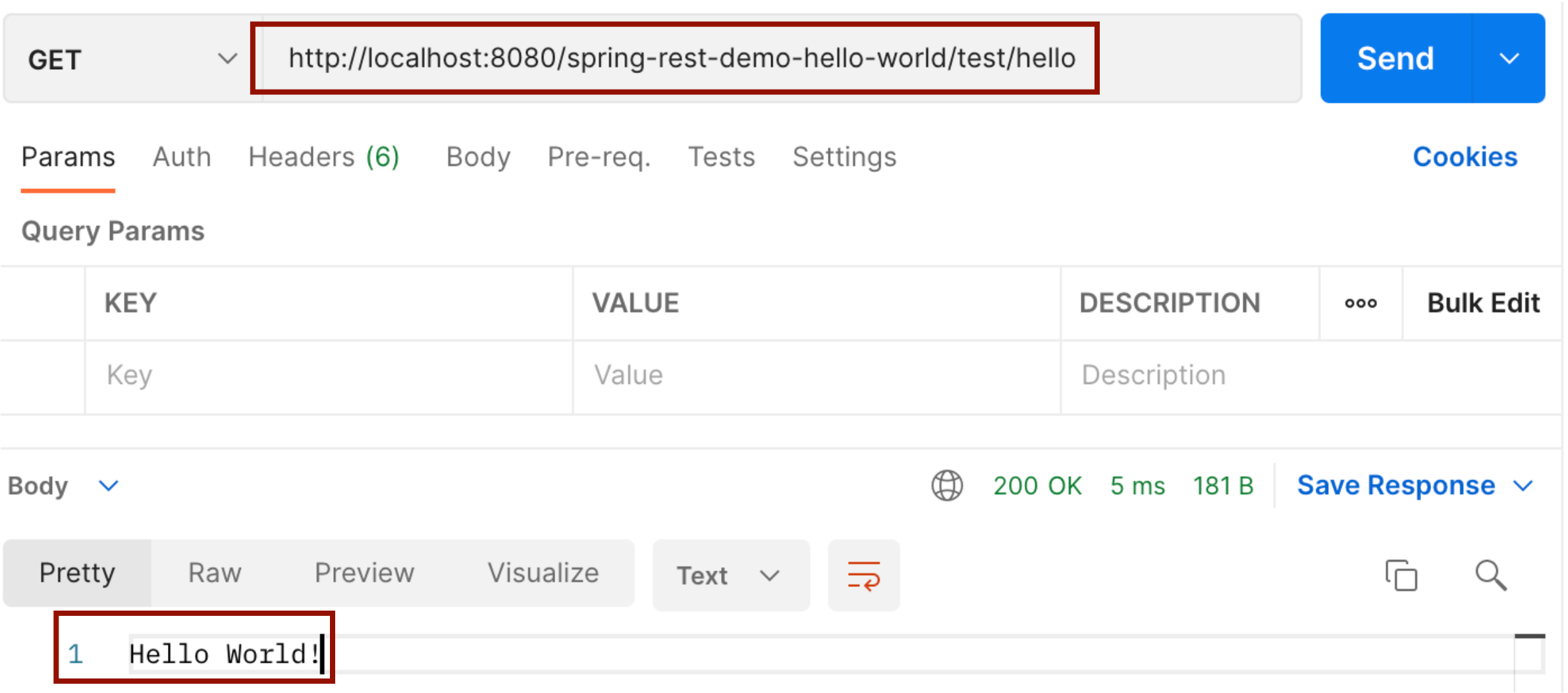
config class from Step 2

Step 4: Create Spring REST Service @RestController

File: DemoRestController.java (project: spring-rest-demo-hello-world)

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/test")
public class DemoRestController {
    // add code for the "/hello" endpoint
    @GetMapping("/hello")
    public String sayHello() {
            return "Hello World!";    }
}
```

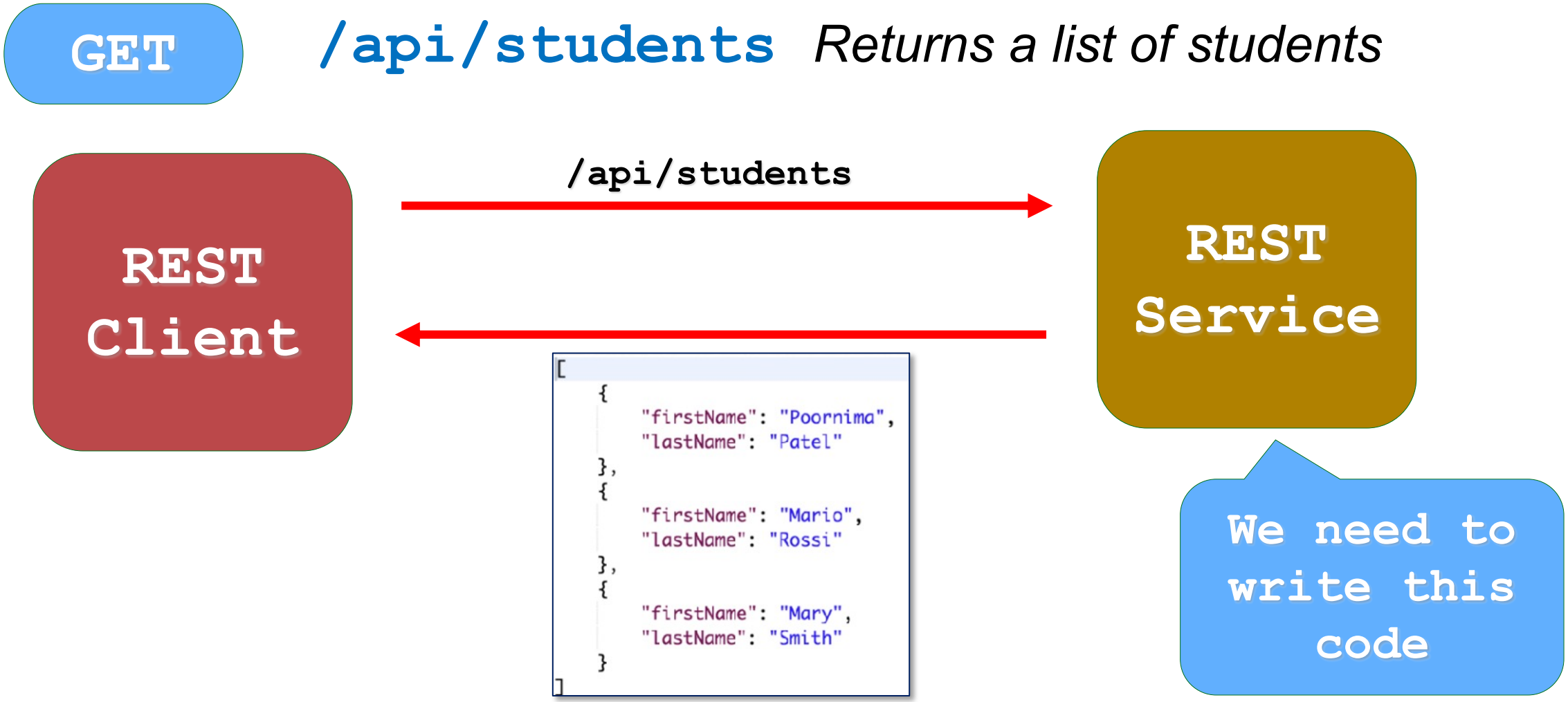
Testing with REST Client - Postman



Testing with REST Client - Web Browser

Spring REST - Retrieve POJOs as JSON

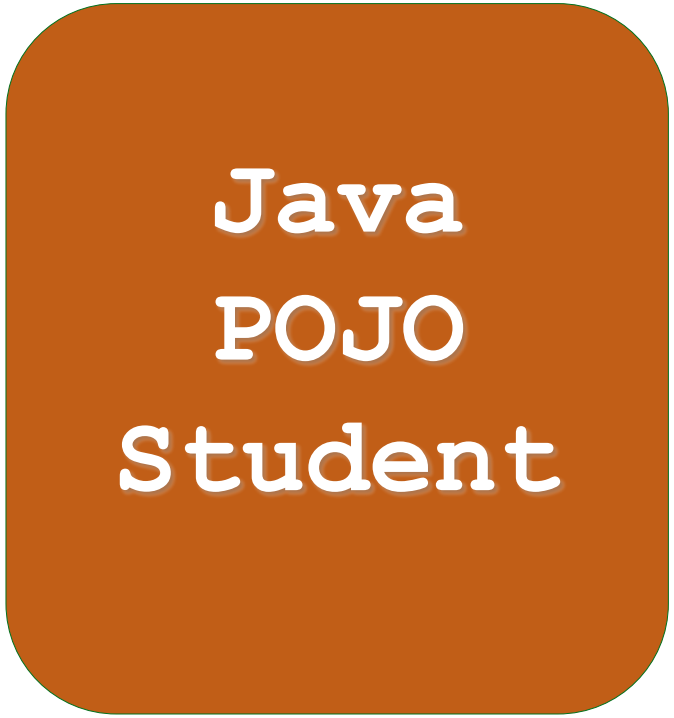
A Service

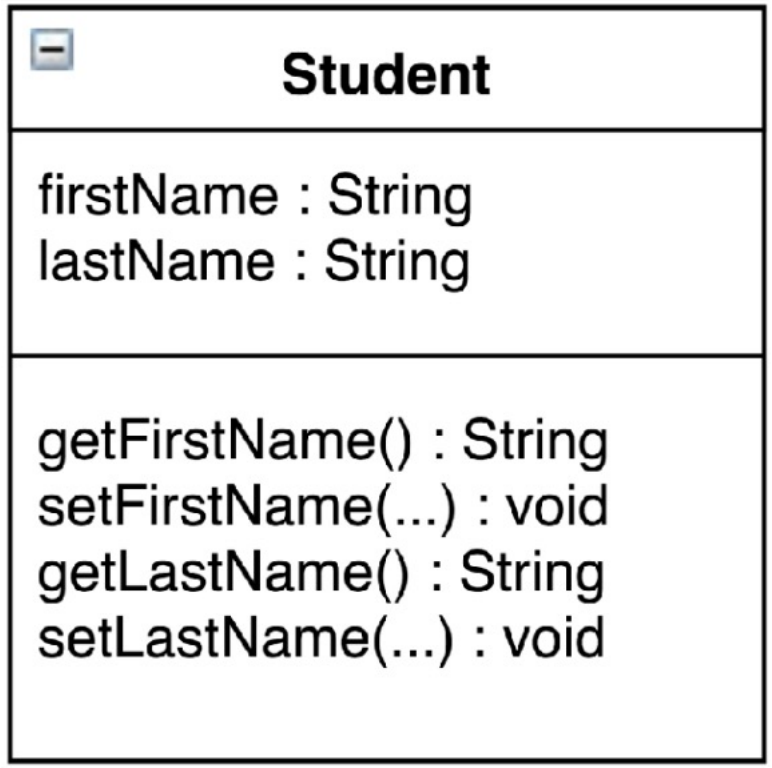

GET
/api/students
Returns a list of students
REST Client
/api/students
REST Service
[
{
"firstName": "Poornima",
"lastName": "Patel"
},
{
"firstName": "Mario",
"lastName": "Rossi"
},
{
"firstName": "Mary",
"lastName": "Smith"
}
]
We need to write this code

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 …

Student POJO (class)



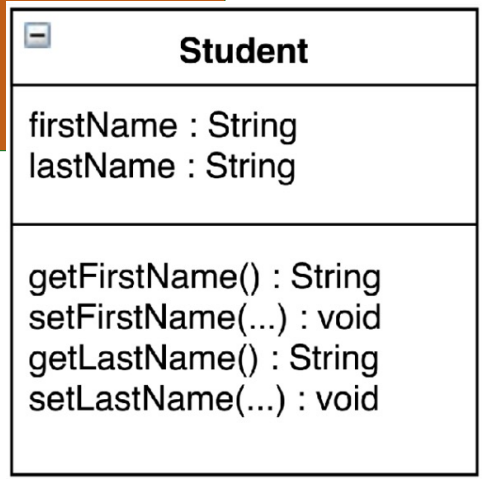


Jackson Data Binding

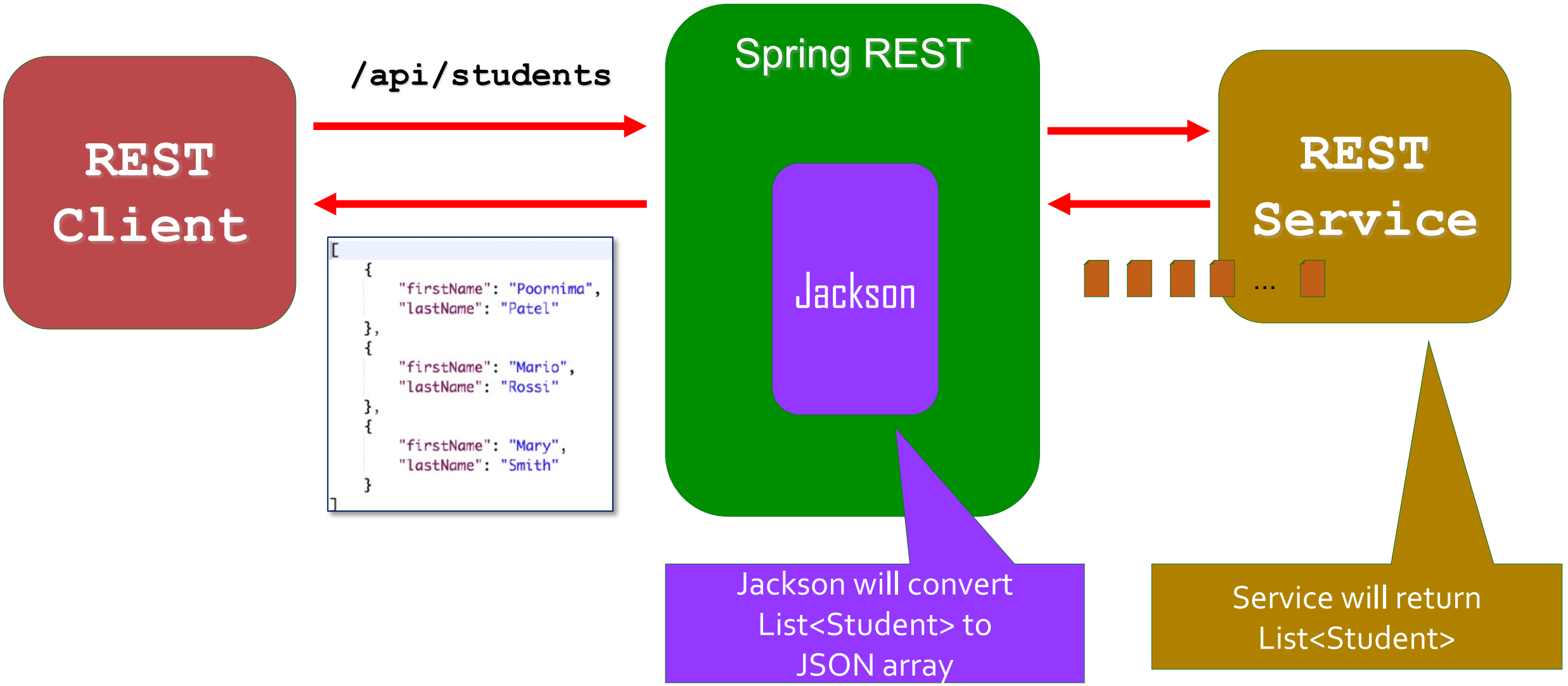
Jackson will call appropriate getter/setter method

```
{
  "id": 14,

  "firstName": "Mario",

  "lastName": "Rossi",

  "active": true
}
```

Java POJO Student



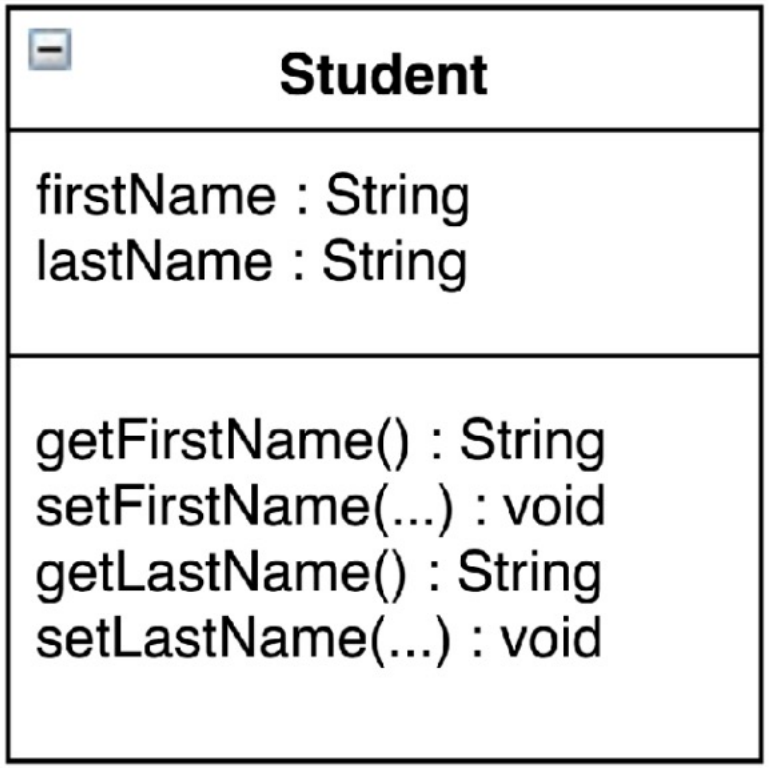
Behind the scenes



Development Process

- Create Java POJO class for **Student**
- Create **Spring REST Service** using **@RestController**

Step 1: Create Java POJO class for Student



FileStudent.java (project: spring-rest-demo-pojo-student-list)

```
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

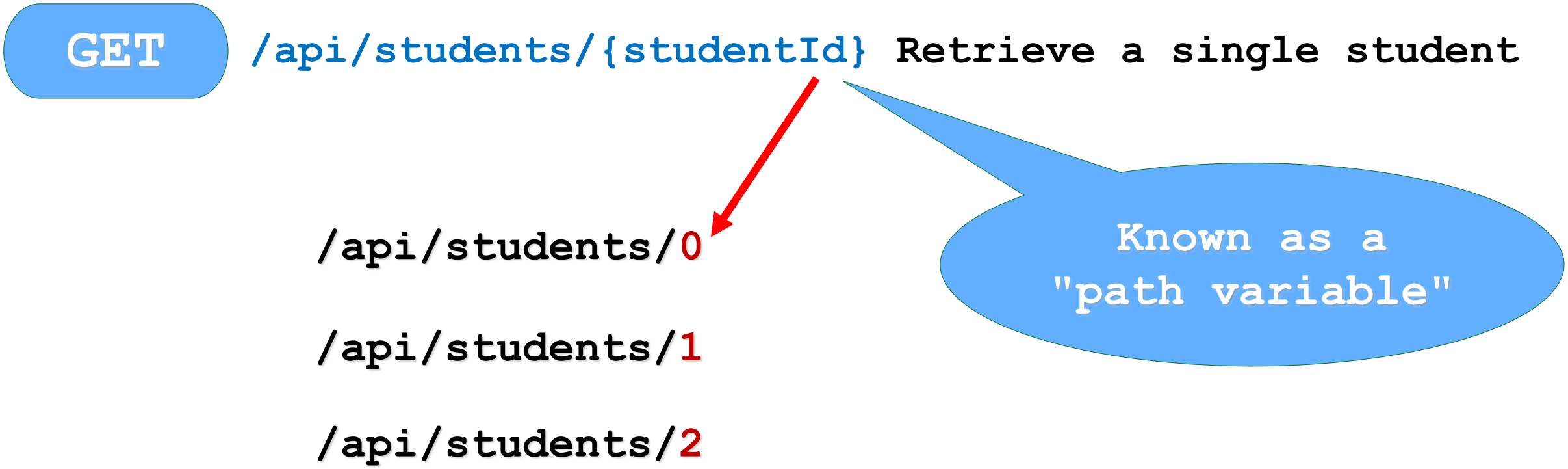
File StudentRestController.java (project: spring-rest-demo-pojo-student-list)

```
@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;
  }

}
```

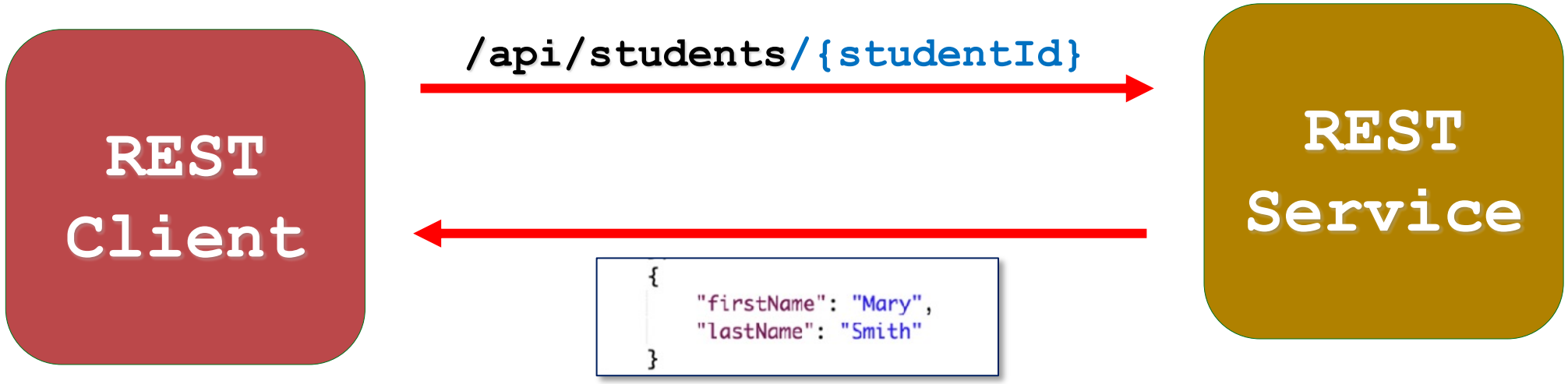
Jackson will convert List<Student> to JSON array

Spring REST - Path Variables

Path Variables



Spring REST Service



Development Process

Add request mapping to Spring REST Service

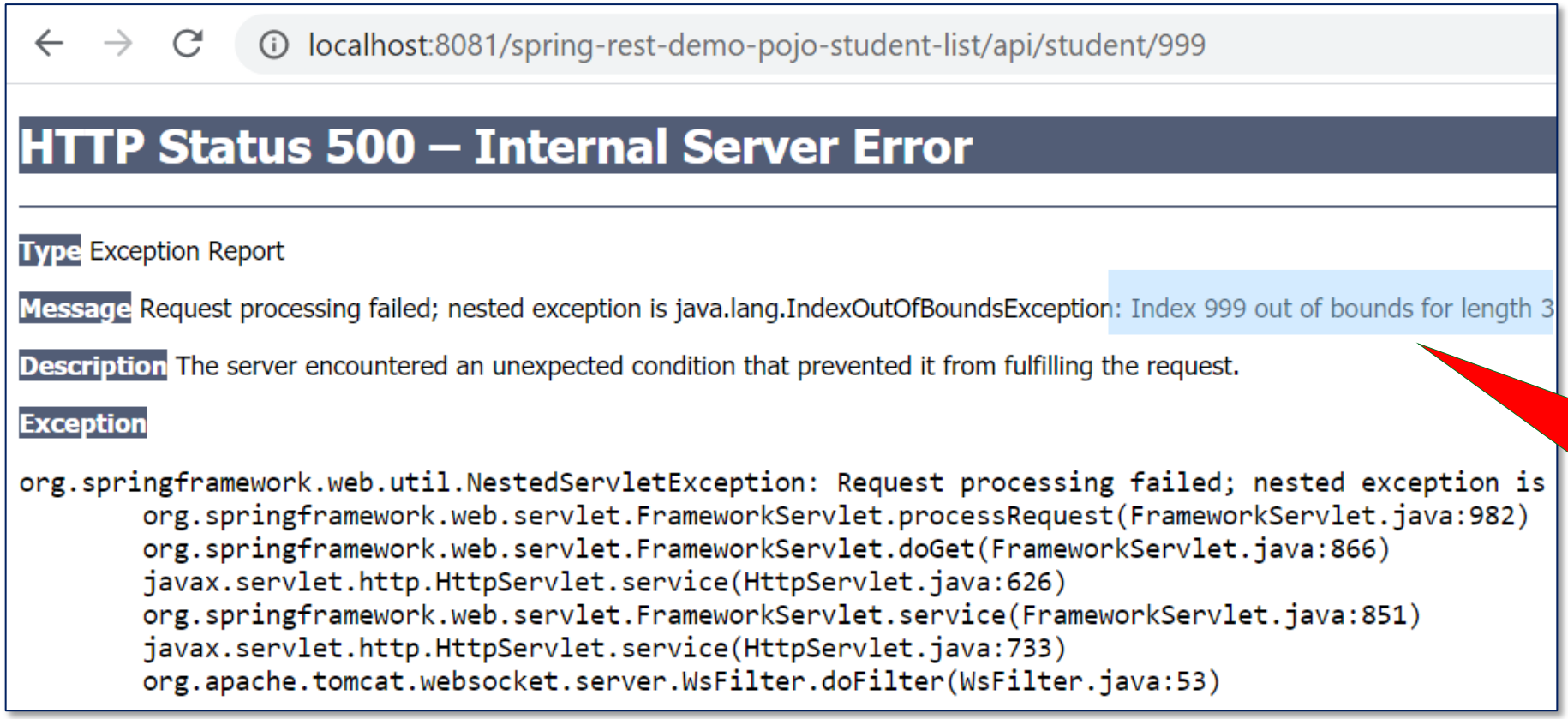
- Bind path variable to method parameter using @PathVariable

File: StudentRestController.java (project: spring-rest-demo-pojo-student-list)

```
// define endpoint for "/students/{studentId}" - return student at index
@GetMapping("/students/{studentId}")
public Student getStudents(@PathVariable int studentId) {
        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.get(studentId);
}
```

Exception Handling

Problem with bad Student Id

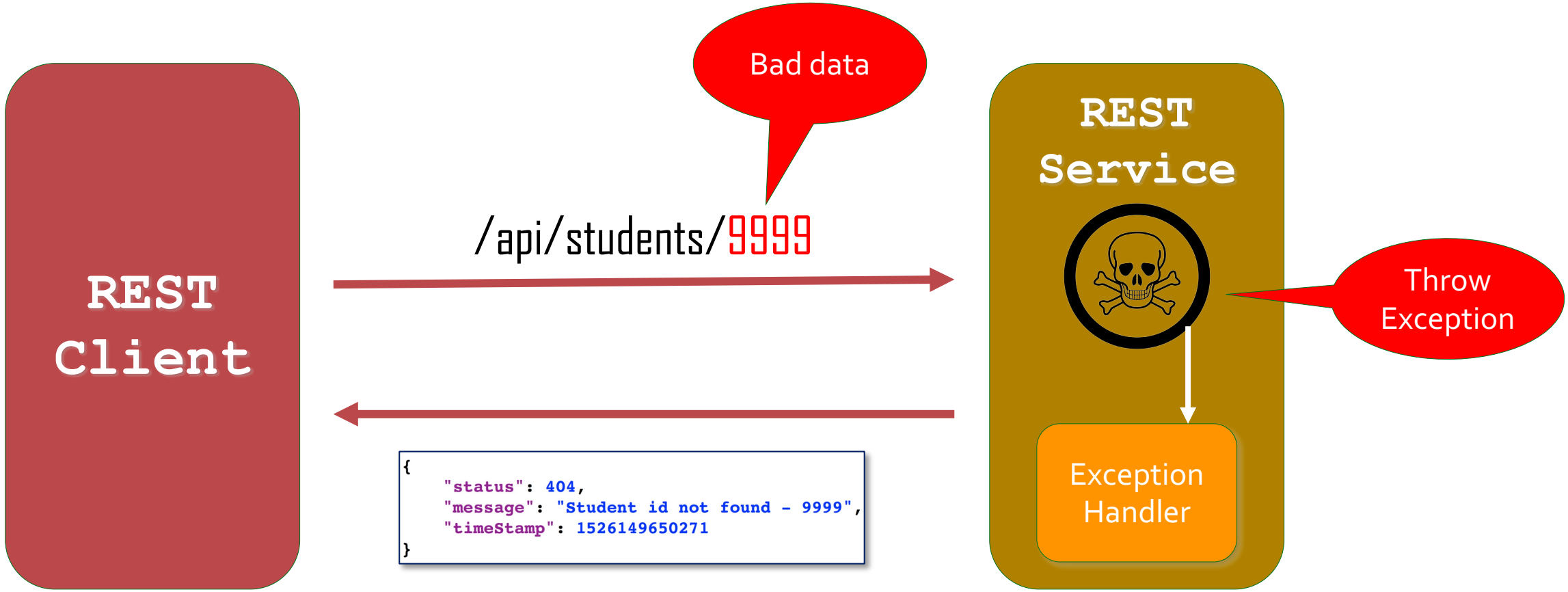


Root cause

Handle the exception and return error as JSON


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

Spring REST Exception Handling



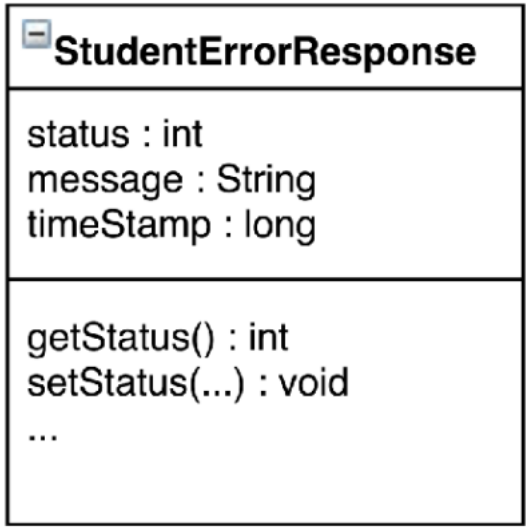
Development Process

- Create a **custom error response** class
- Create a **custom exception** class
- Update REST service to throw exception if student not found
- Add an exception handler method using @ExceptionHandler

Step 1: Create a 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**

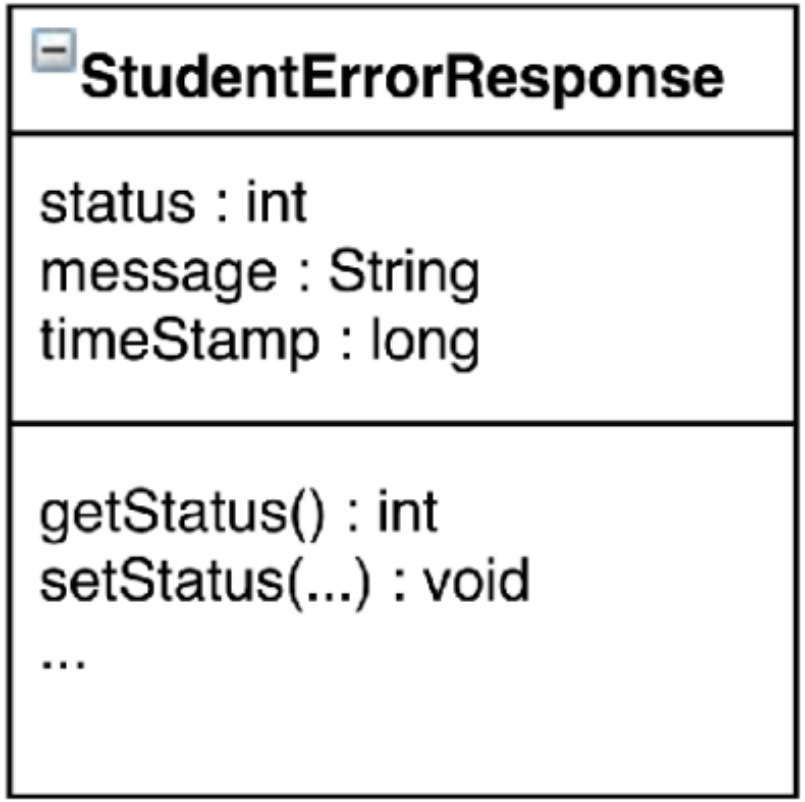
```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```



Step 1: Create a custom error response class

File: StudentErrorResponse.java (project: spring-rest-demo-pojo-student-list)

```
public class StudentErrorResponse {

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

  // constructors

  // getters / setters

}
```



Step 2: Create a custom exception class

- 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 a custom exception class

File: StudentNotFoundException.java (project: spring-rest-demo-pojo-student-list)

```
public class StudentNotFoundException extends RuntimeException {
    public StudentNotFoundException(String message, Throwable cause) {
                                super(message, cause); }

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

    public StudentNotFoundException(Throwable cause) {
                                super(cause);}
}
```

Step 3: Update REST service to throw exception

File: StudentRestController.java (project: spring-rest-demo-pojo-student-list)

```
@GetMapping("/students/{studentId}")
public Student getStudents(@PathVariable int studentId) {
        List<Student> theStudents = new ArrayList<>();
        comment
        // 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

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

File: StudentRestController.java (project: spring-rest-demo-pojo-student-list)

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

Exception handler method

Type of response body

Exception type to handle/catch

Step 4: Add exception handler method

File: StudentRestController.java (project: spring-rest-demo-pojo-student-list)

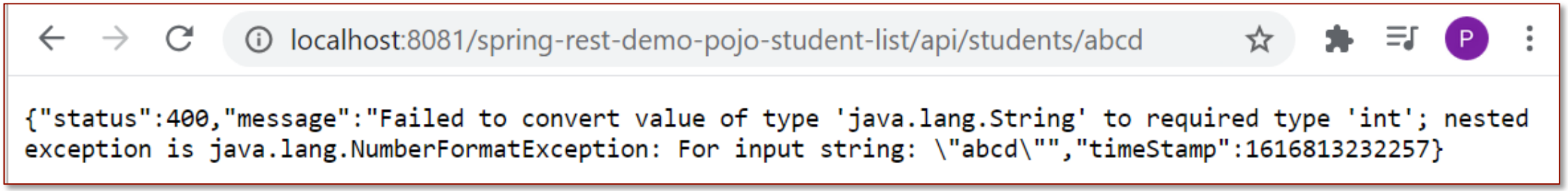
```
@ExceptionHandler
public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc)
{       StudentErrorResponse error = new StudentErrorResponse();
        error.setStatus(HttpStatus.NOT_FOUND.value());
        error.setMessage(exc.getMessage());
        error.setTimeStamp(System.currentTimeMillis());
        return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);     }
```

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

Handle for Generic Exceptions

File: StudentRestController.java (project: spring-rest-demo-pojo-student-list)

```
@ExceptionHandler
public ResponseEntity<StudentErrorResponse> handleException(Exception exc) {
        StudentErrorResponse error = new StudentErrorResponse();
        error.setStatus(HttpStatus.BAD_REQUEST.value());
        error.setMessage(exc.getMessage());
        error.setTimeStamp(System.currentTimeMillis());
        return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
}
```



Handle Global Exceptions

» The previous Exception handler code is **only** for the **specific REST controller** →Can't be reused by other controllers ☹

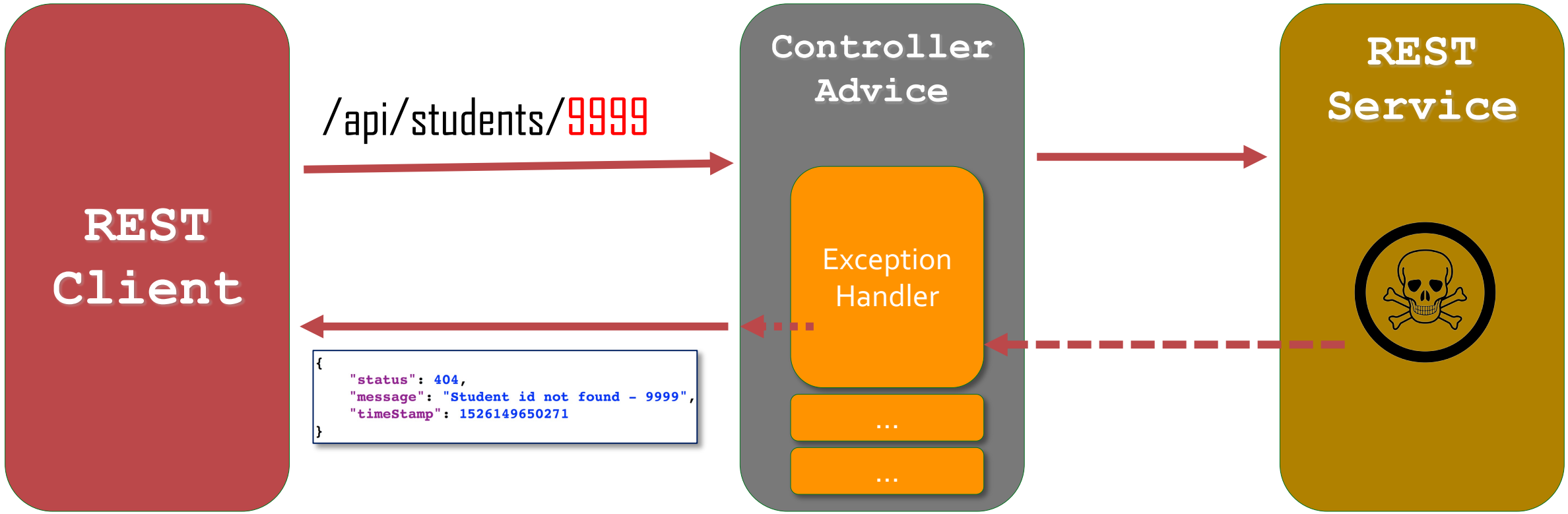
» We need **global exception handlers**

- Promotes reuse
- **Centralizes** 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

Spring @ControllerAdvice



Development Process

- Create new **@ControllerAdvice**
- Refactor **REST service** … remove exception handling code

Step 1: Create new @ControllerAdvice

FileStudentRestExceptionHandler.java (project: spring-rest-demo-pojo-student-list-global-hanling-exception)

```
@ControllerAdvice
public class StudentRestExceptionHandler {
    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(Exception exc) {
        // create a StudentErrorResponse
        StudentErrorResponse error = new StudentErrorResponse();
        error.setStatus(HttpStatus.BAD_REQUEST.value());
        error.setMessage(exc.getMessage());
        error.setTimeStamp(System.currentTimeMillis());
        // return ResponseEntity
        return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);   }
}
```

Step 2: Refactor REST service code…

File: StudentRestController.java (project: spring-rest-demo-pojo-student-list-global-hanling-exception)

```
@GetMapping("/students/{studentId}")
public Student getStudents(@PathVariable int studentId) throws Exception {
        List<Student> theStudents = new ArrayList<>();
        comment
        // check the studentId against list size
        if ( (studentId >= theStudents.size()) || (studentId < 0) ) {
                throw new  Exception("Student id not found - " + studentId);
        }
        return theStudents.get(studentId);
}
```

Spring REST API Design

Spring 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

- **Review** API requirements
- **Identify** main resource / entity
- **Use HTTP methods** to **assign action on resource**

Step 1: Review API requirements

From the Constructor

- 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 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 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

CRUD Endpoint Examples

HTTP METHOD	END POINT	CRUD ACTION
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customres
GET	/api/customers/{customerId}	Read a single customres
PUT	/api/customers	Update an existing entity
DELETE	/api/customers/{customerId}	Delete an existing entity

Anti-Patterns

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

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

NO

Don't include actions in the endpoint

YES

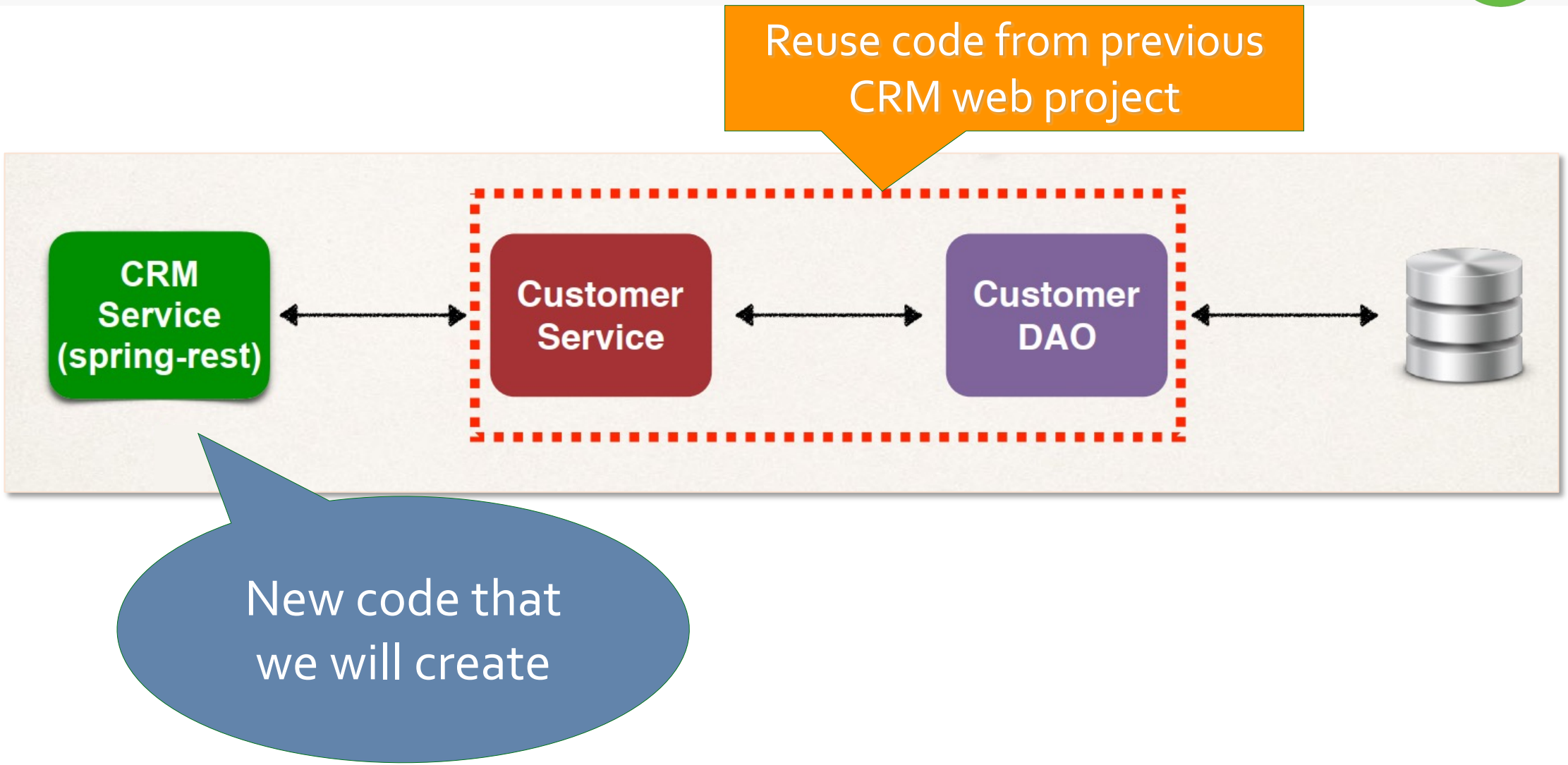
Instead, use HTTP methods to assign actions

Spring REST API - Real Time Project

Real Time Project

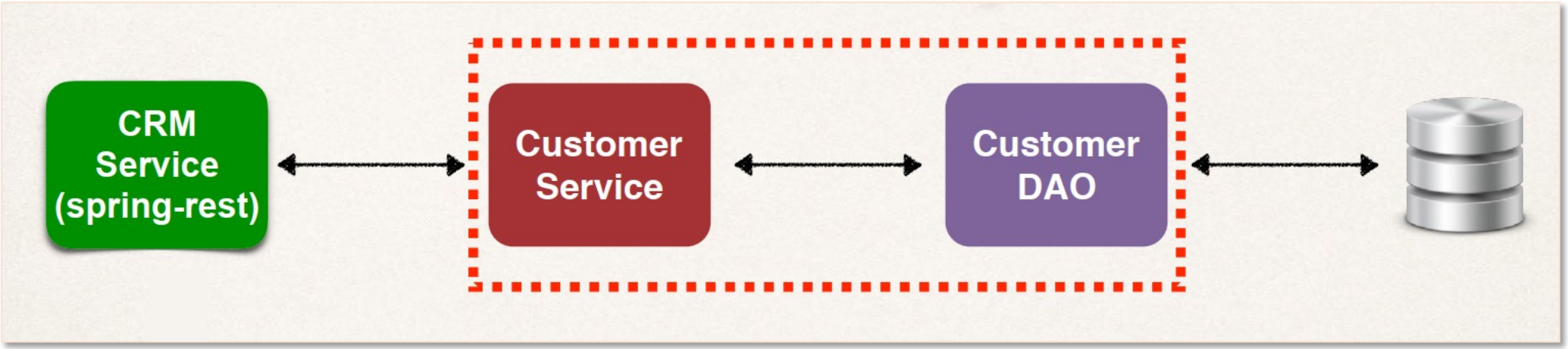
HTTP METHOD	END POINT	CRUD ACTION
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customres
GET	/api/customers/{customerId}	Read a single customres
PUT	/api/customers	Update an existing entity
DELETE	/api/customers/{customerId}	Delete an existing entity

Application Architecture



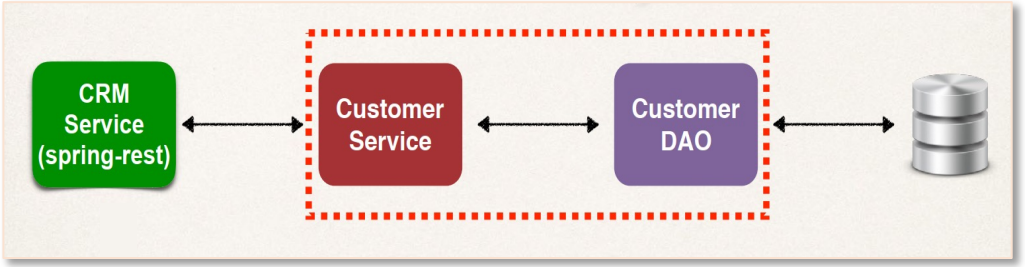
Project Set Up

- Setup Maven project
- Copy CustomerService, CustomerDAO and Customer entity from previous CRM Spring MVC Project



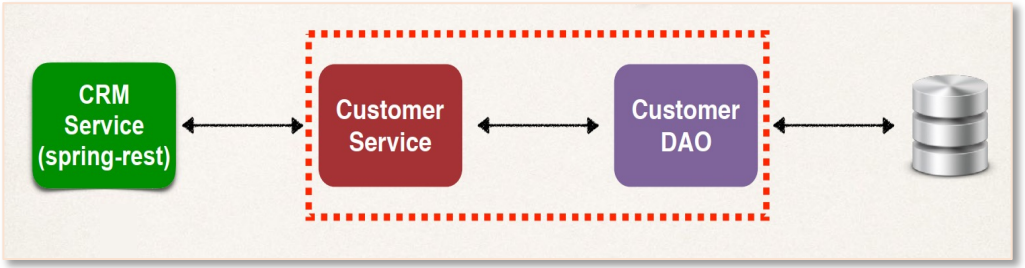
Development Process

- Get customers
- Get single customer by ID
- Add a new customer
- Update an existing customer
- Delete an existing customer



Review Task List

- pom.xml file
- All Java Config
- Configuration file for database connection
- Hibernate Entity class: Customer
- DAO: CustomerDAO …
- Service: CustomerService …

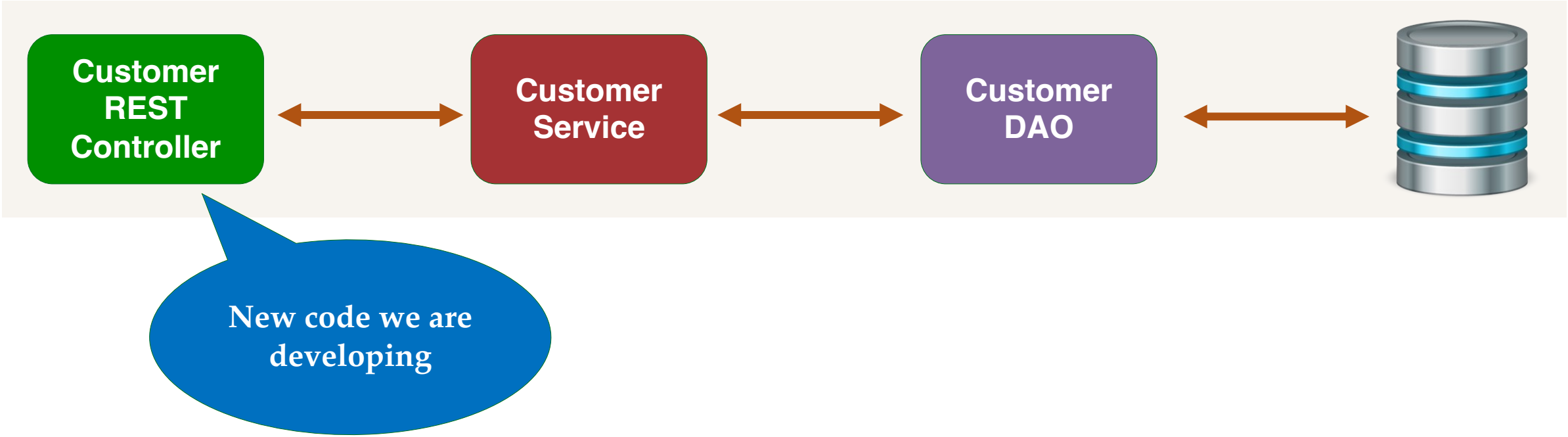


Get Customers

Application Interaction

Get Customers

Application Architecture



Development Process

- Create **Customer REST Controller**
- **Autowire** CustomerService
- Add **mapping** for **GET** /customers

Development Process

File: CustomerRestController.java (spring-rest-demo-crm-crud)

```
@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();
}
```

Get Single Customer

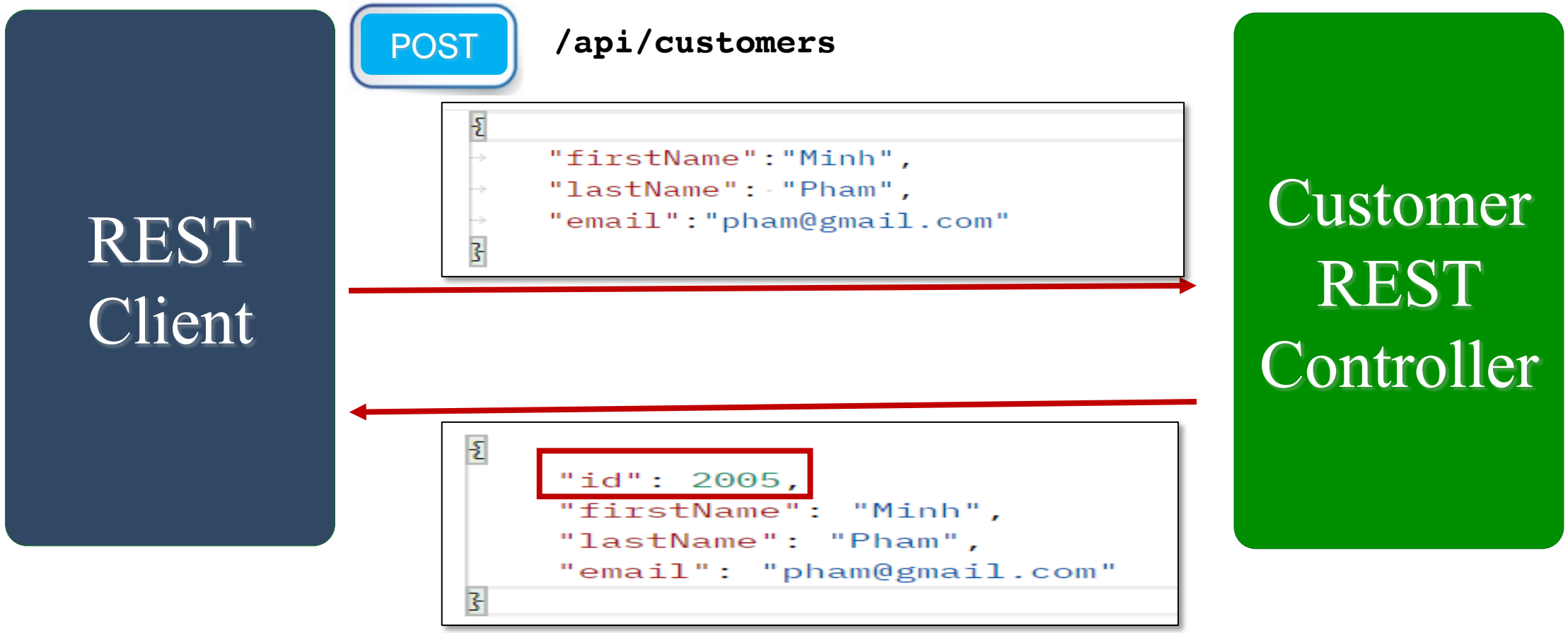
Application Interaction

Get Single Customer


```
@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;}
}
```

Add Customer

Application Interaction



Add Customer


```
@PostMapping("/customers")
public Customer addCustomer(@RequestBody Customer
theCustomer) {
// also just in case the pass an id in JSON
// this is force a save of new item ... in
theCustomer.setId(0);
customerService.saveCustomer(theCustomer);
return theCustomer;
}
```

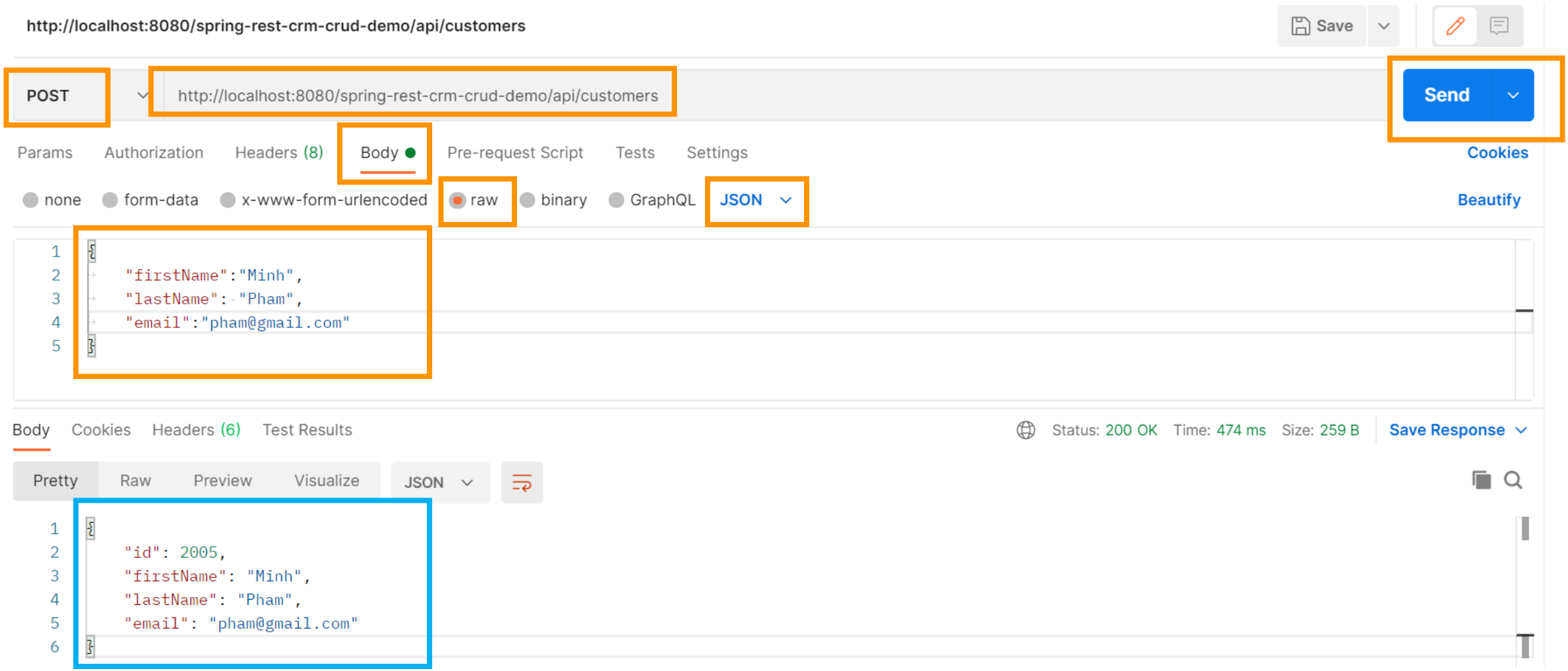
ID of 0 means
DAO code will
"INSERT" new customer

Add Customer - @RequestBody

- 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) {
...
}
```

Add Customer – Postman Test



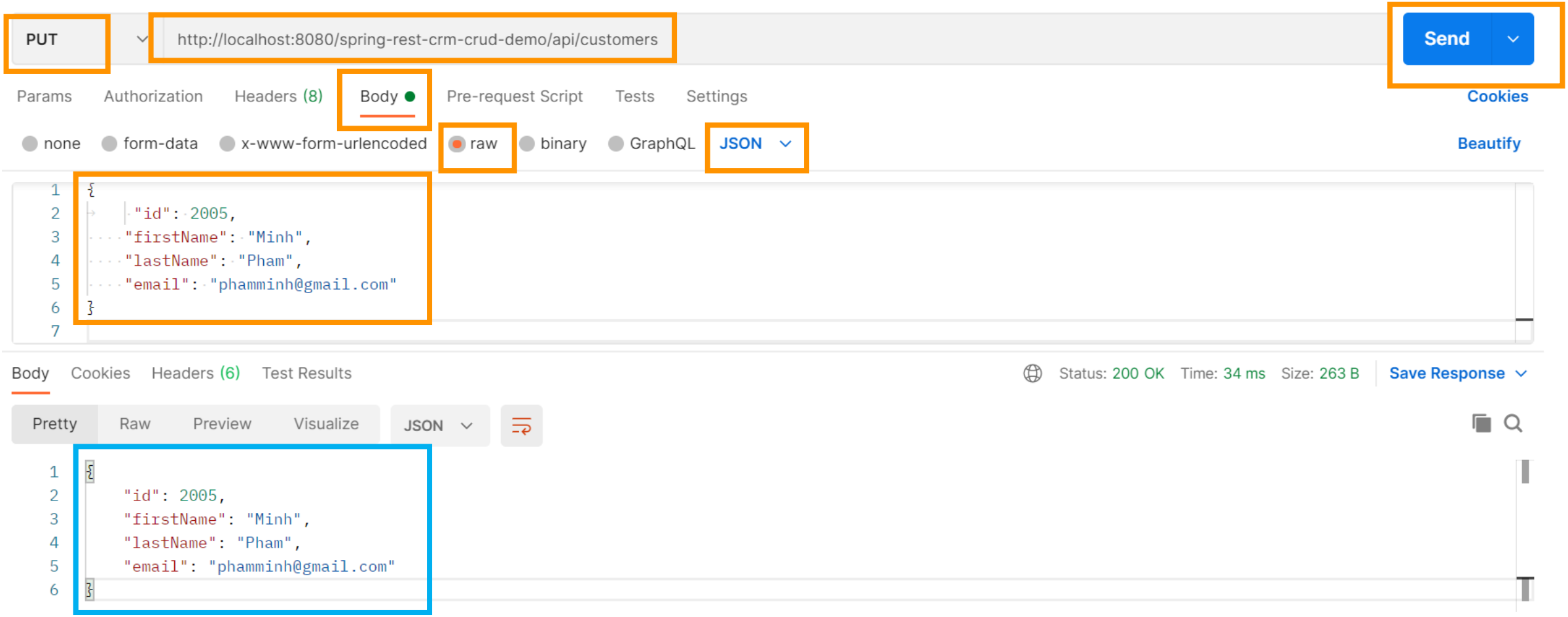
Update Customer

Application Interaction

Update Customer

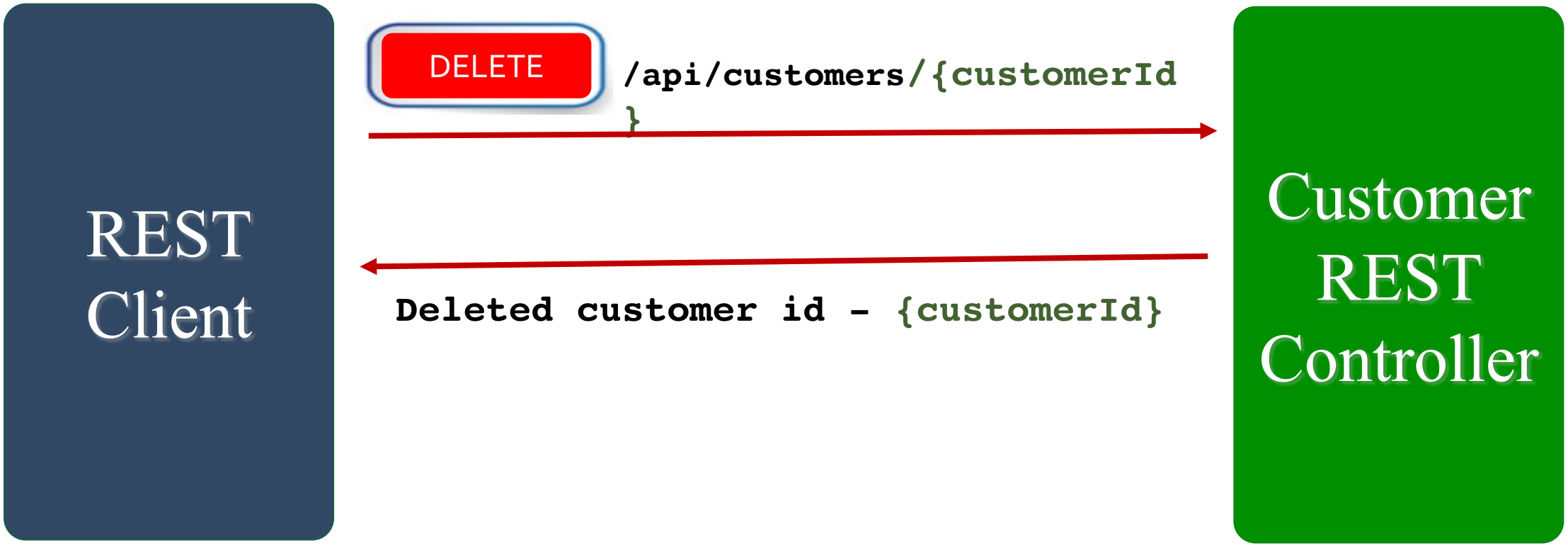

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
…
// add mapping for PUT /customers - update existing
customer
@PutMapping("/customers")
public Customer updateCustomer(@RequestBody
Customer theCustomer) {
customerService.saveCustomer(theCustomer);
return theCustomer;}
}
```

Update Customer – Postman Test



Delete Customer

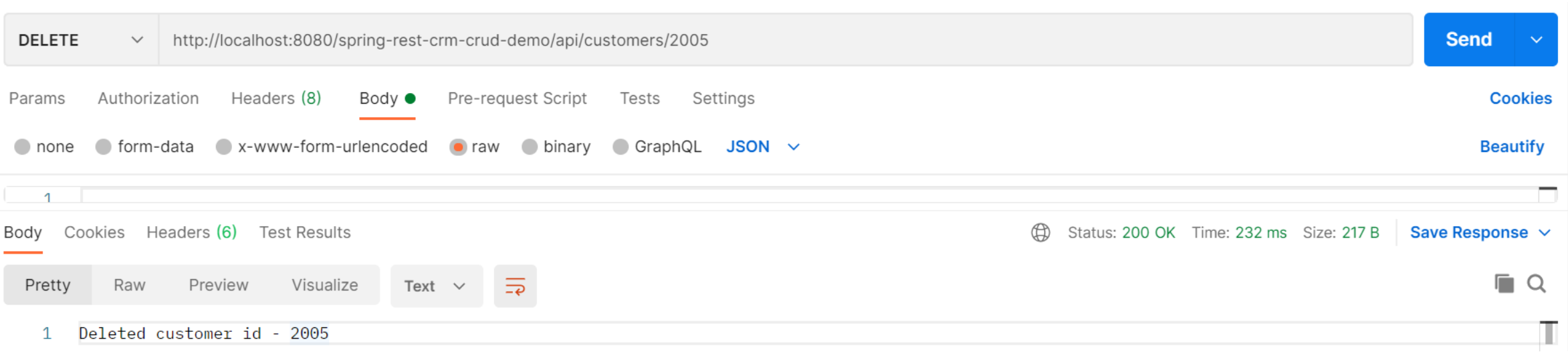
Application Interaction



Delete Customer

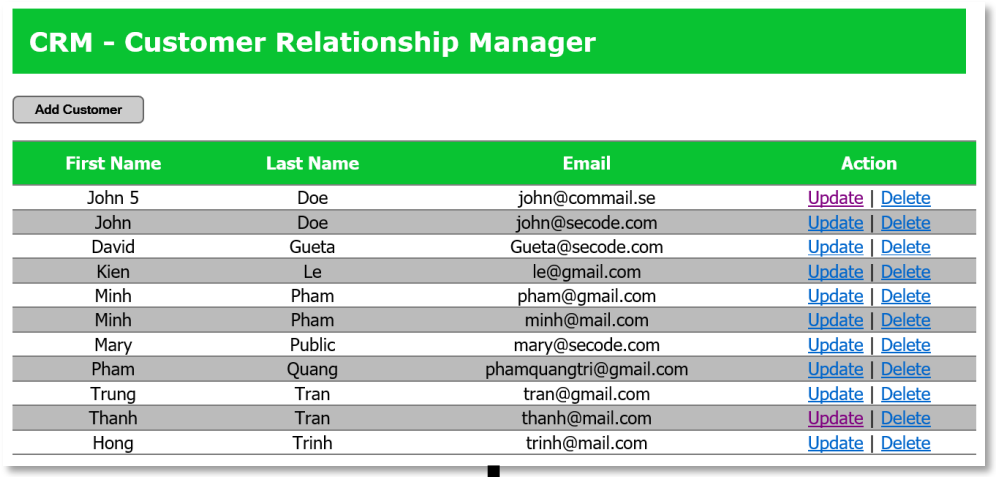

```
@DeleteMapping("/customers/{customerId}")
public String deleteCustomer(@PathVariable int
customerId) {
Customer tempCustomer =
customerService.getCustomer(customerId);
// throw exception if null
if (tempCustomer == null) {
throw new CustomerNotFoundException("Customer id
not found - " + customerId);}

customerService.deleteCustomer(customerId);
return "Deleted customer id - " + customerId;}
```

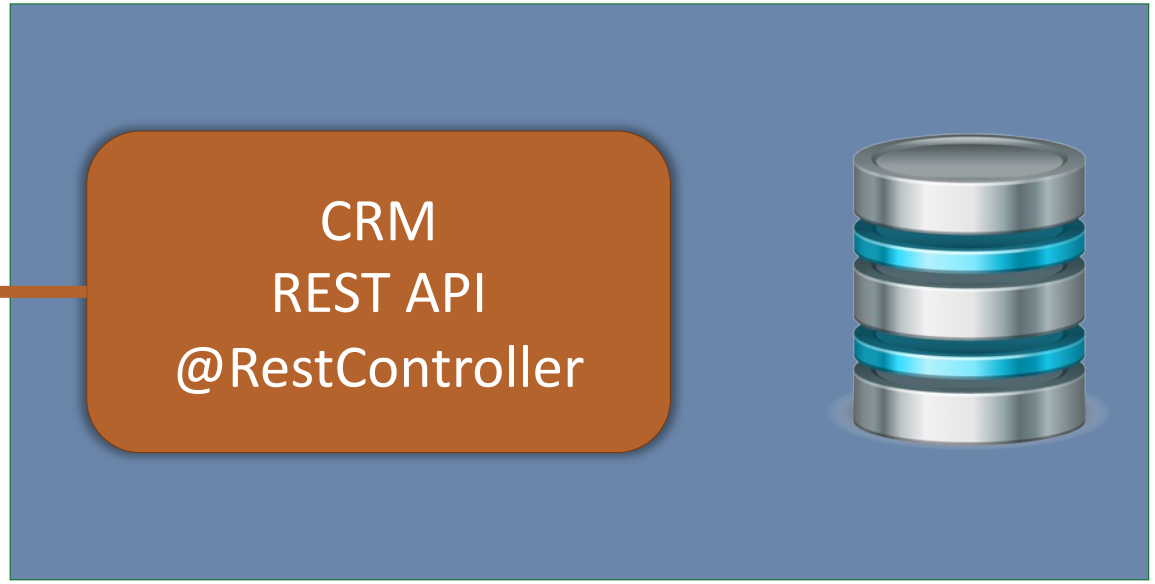
Delete Customer – Postman Test



REST Client with Spring

CRM REST Client with Spring

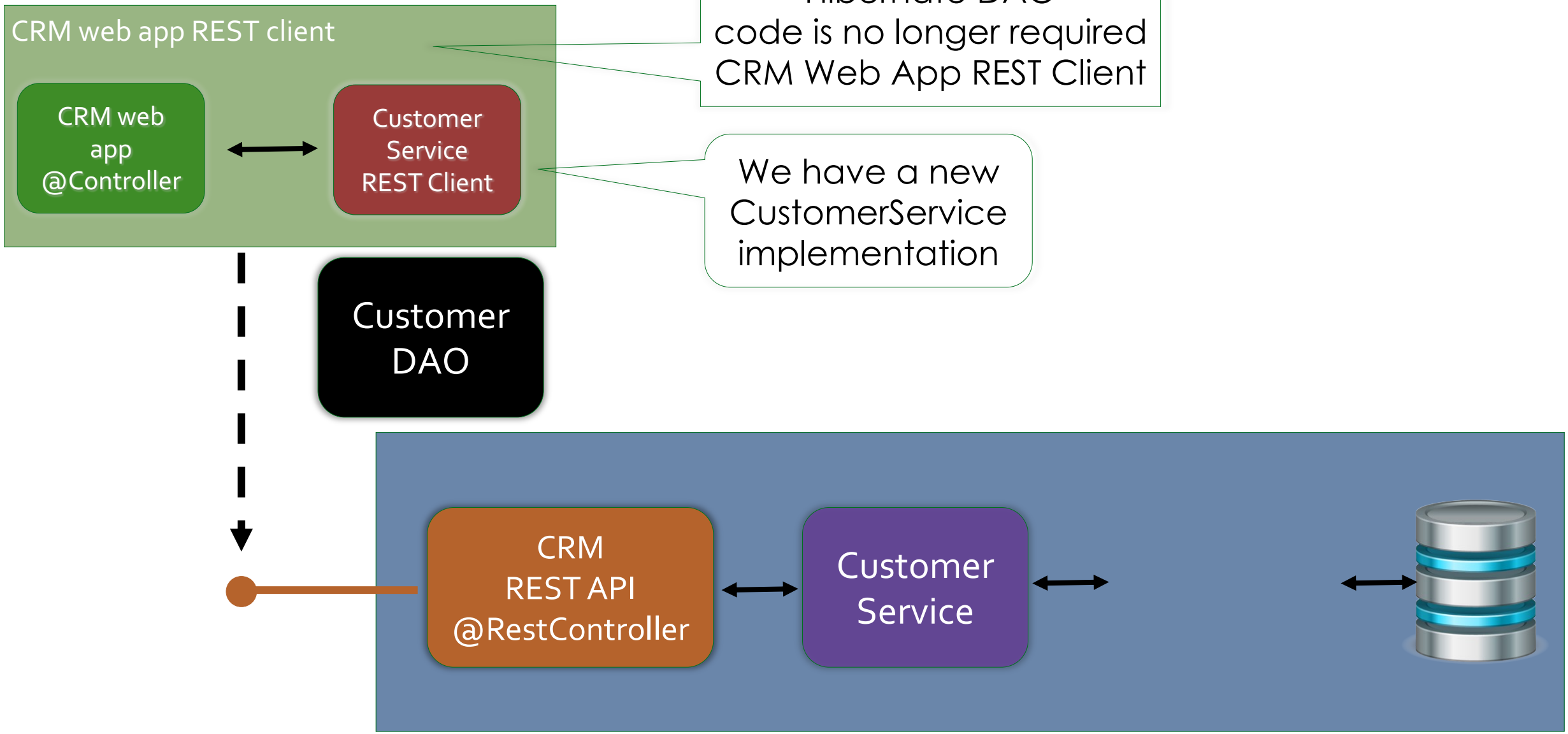




Overview of Steps

1. Application Architecture
2. Review project structure
3. Maven POM file
4. Application configuration properties file
5. Spring Bean configuration for RestTemplate
6. GET: Get a list of customers
7. GET: Get a single customer by id
8. POST: Add a new customer
9. PUT: Update an customer
10. DELETE: Delete an customer

Application Architecture



POM.xml (project: spring-rest-crm-crud-client-demo)

```
<dependencies>
    <!-- Spring -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${springframework.version}</version>
    </dependency>
     <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.5</version>
    </dependency>
    <!-- Servlet+JSP+JSTL -->
    <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
    </dependency>
    <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.1</version>
    </dependency>
    <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
    </dependency>
</dependencies>
```

Application Configuration Properties

- CRM Web App REST Client needs to connect to the backend REST API
- we have to add a configuration property that has the url of the REST API

File: src/main/application.properties (project: spring-rest-crm-crud-client-demo)

```
#
# The URL for the CRM REST API
# - update to match your local environment
#
crm.rest.url=http://localhost:8080/spring-crm-rest/api/customers
```

Spring Bean configuration for RestTemplate

File: DemoAppConfig.java (project: spring-rest-crm-crud-client-demo)

```
@Configuration
@EnableWebMvc
@ComponentScan("com.se")
@PropertySource({ "classpath:application.properties" })
public class DemoAppConfig implements WebMvcConfigurer {
        // define a bean for ViewResolver
        @Bean
        public ViewResolver viewResolver() {...9 l...5 }
        // define bean for RestTemplate ... thi...is used to
        //make client REST calls
        @Bean
        public RestTemplate restTemplate() {
                return new RestTemplate();      }
        // add resource handler for loading css, images, etc
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry)
}
```

This code also loads the application.properties file with the @PropertySource annotation

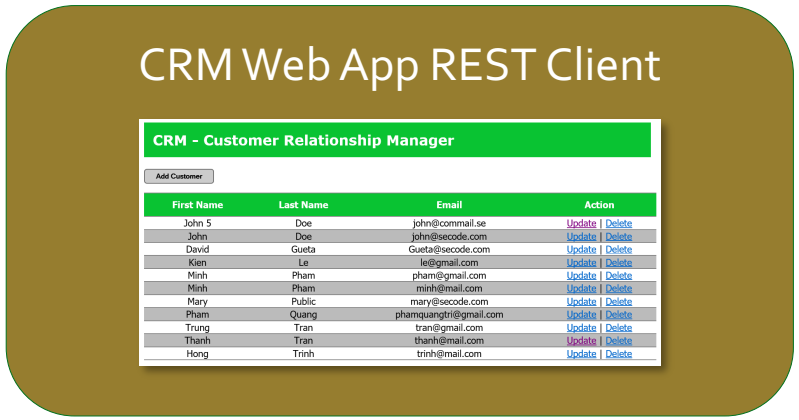
inject bean into CustomerServiceRestClientImpl

File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@Service
public class CustomerServiceRestClientImpl implements CustomerService {
    private RestTemplate restTemplate;
    private String crmRestUrl;
    private Logger logger = Logger.getLogger(getClass().getName());
    @Autowired
    public CustomerServiceRestClientImpl(RestTemplate theRestTemplate,
        @Value("${crm.rest.url}") String theUrl) {
        restTemplate = theRestTemplate;
        crmRestUrl = theUrl;
        logger.info("Loaded property:  crm.rest.url=" + crmRestUrl);}
    @Override
    public List<Customer> getCustomers() {...11 lines }
    @Override
    public Customer getCustomer(int theId) {...9 lines }
    @Override
    public void saveCustomer(Customer theCustomer) {...12 lines }
    @Override
    public void deleteCustomer(int theId) {...6 lines }
}
```

inject the crm.rest.url property using the @Value annotation

GET: Get a list of Customers



GET `/api/customers`

CRM REST API

```
{
    "id": 1002,
    "firstName": "John",
    "lastName": "Doe",
    "email": "john@commail.se"
},
{
    "id": 2,
    "firstName": "John",
    "lastName": "Doe",
    "email": "john@secode.com"
},
```

GET: Get a list of Customers

File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@Override
p
    // make REST ca
    ResponseEntity<List<Cus    er>> responseEntity =
            restTemplate.exchange(crmRestUrl, HttpMethod.GET, null,
            new ParameterizedTypeReference<Li
            // get the li        omers from response
            logger.info("in getCustomers(): customers" + customers);
            return customers;
    }
```

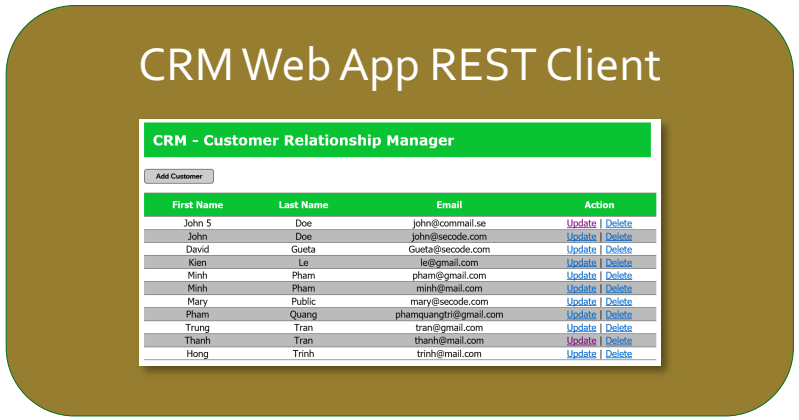
http://localhost:8080/spring-rest-crm-crud-demo/api/customers

request headers or body, null in our case

HTTP method for GET

responseType: the type of the return value

GET: Get a single Customer by ID



GET `/api/customers/{customerId}`

CRM REST API

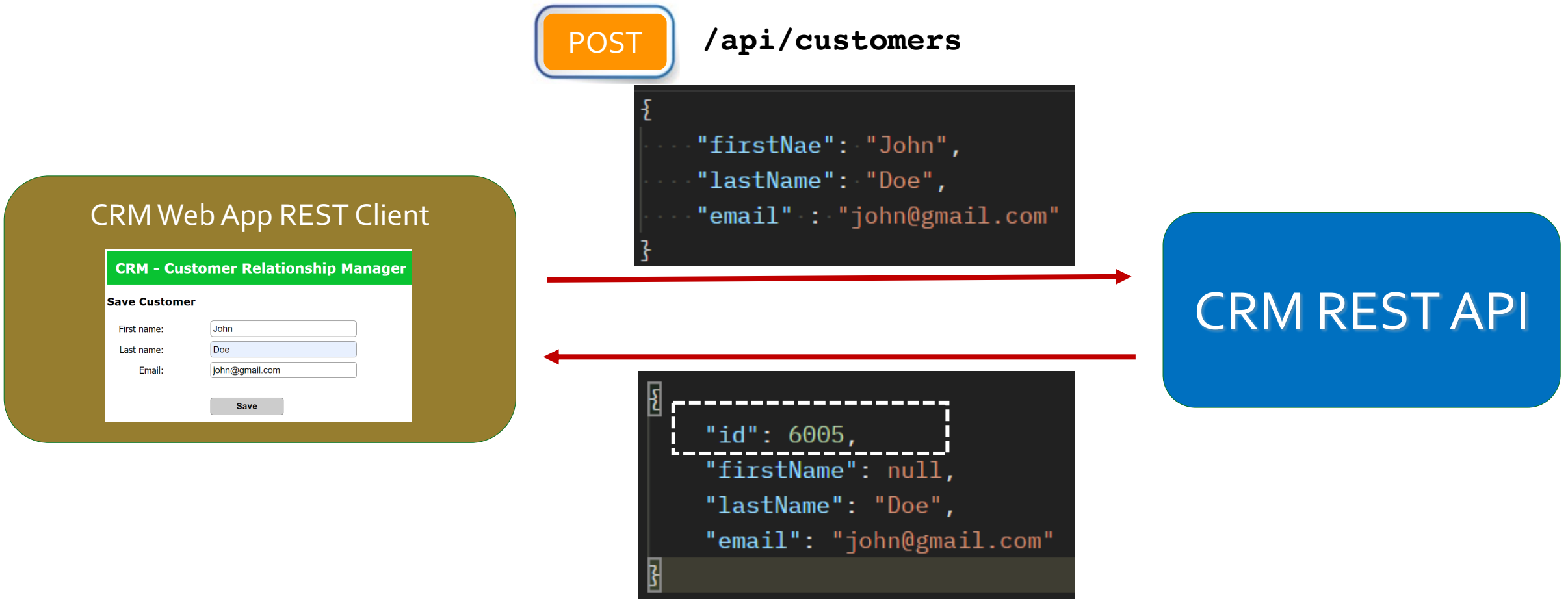
```
{
    "id": 1002,
    "firstName": "John",
    "lastName": "Doe",
    "email": "john@commail.se"
}
```

GET: a single Customer by ID

File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@Override
public Customer getCustomer(int theId) {
    logger.info("in getCustomer(): Calling REST API " + crmRestUrl);
    // make REST call
    Customer theCustomer =
    restTemplate.getForObject(crmRestUrl + "/" + theId,
                              Customer.class);
    logger.info("in saveCustomer(): theCustomer=" + theCustomer);
    return theCustomer;
}
```

POST: Add a new Customer

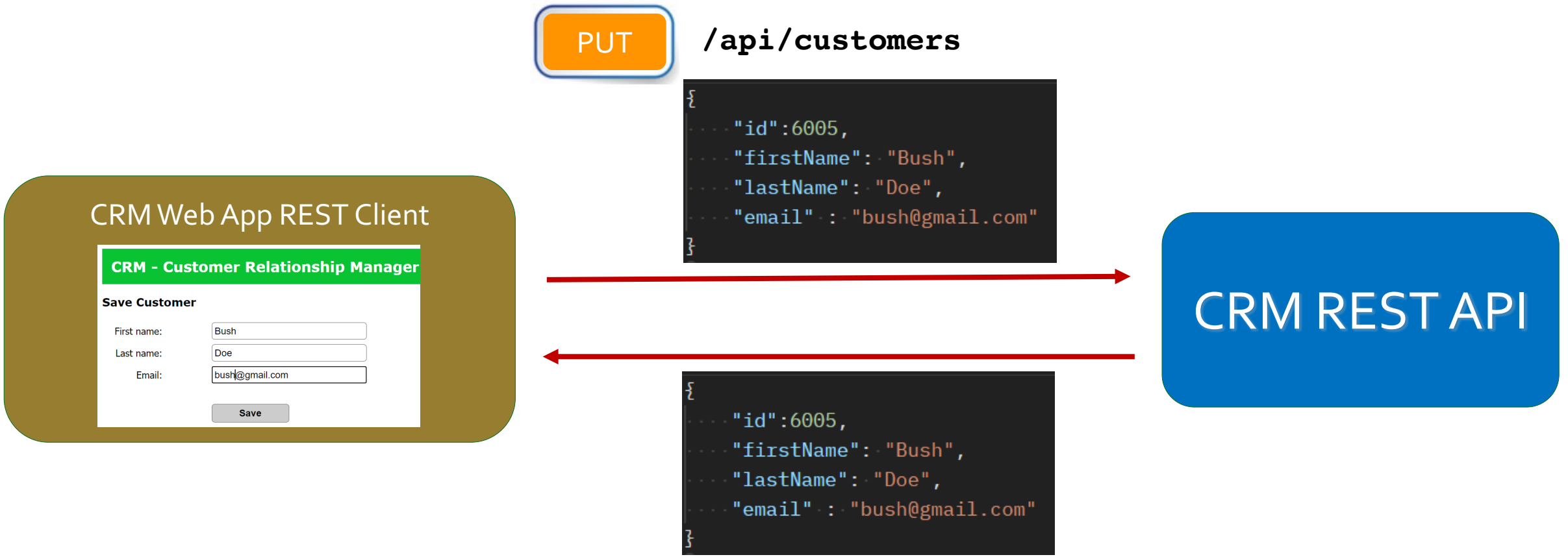


POST: Add a new Customer

File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@Override
public void saveCustomer(Customer theCustomer) {
    logger.info("in saveCustomer(): Calling REST API " + crmRestUrl);
    int customerId = theCustomer.getId();
    // make REST call
    if (customerId == 0) {
        // add employee
        restTemplate.postForEntity(crmRestUrl, theCustomer, String.class);
    } else {
        // update employee
        restTemplate.put(crmRestUrl, theCustomer);}
        logger.info("in saveCustomer(): success");
}
```

PUT: Edit an existing Customer

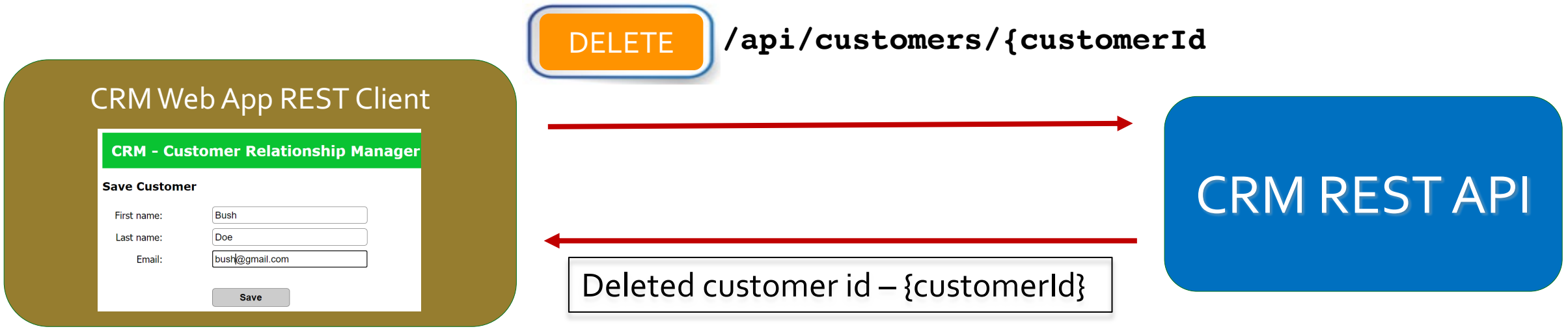


PUT: Edit an existing Customer

File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@Override
public void saveCustomer(Customer theCustomer) {
    logger.info("in saveCustomer(): Calling REST API " + crmRestUrl);
    int customerId = theCustomer.getId();
    // make REST call
    if (customerId == 0) {
        // add employee
        restTemplate.postForEntity(crmRestUrl, theCustomer, String.class);
    } else {
        // update employee
        restTemplate.put(crmRestUrl, theCustomer);}
        logger.info("in saveCustomer(): success");
}
```

DELETE: Delete a Customer



DELETE: Delete a Customer

File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@Override
public void deleteCustomer(int theId) {
    logger.info("in deleteCustomer(): Calling REST API " + crmRestUrl);
    // make REST call
    restTemplate.delete(crmRestUrl + "/" + theId);
    logger.info("in deleteCustomer(): deleted customer theId=" + theId);
}
```

Questions