

Bộ môn: Kỹ Thuật Phần Mềm

Lập trình www Java



Spring - REST





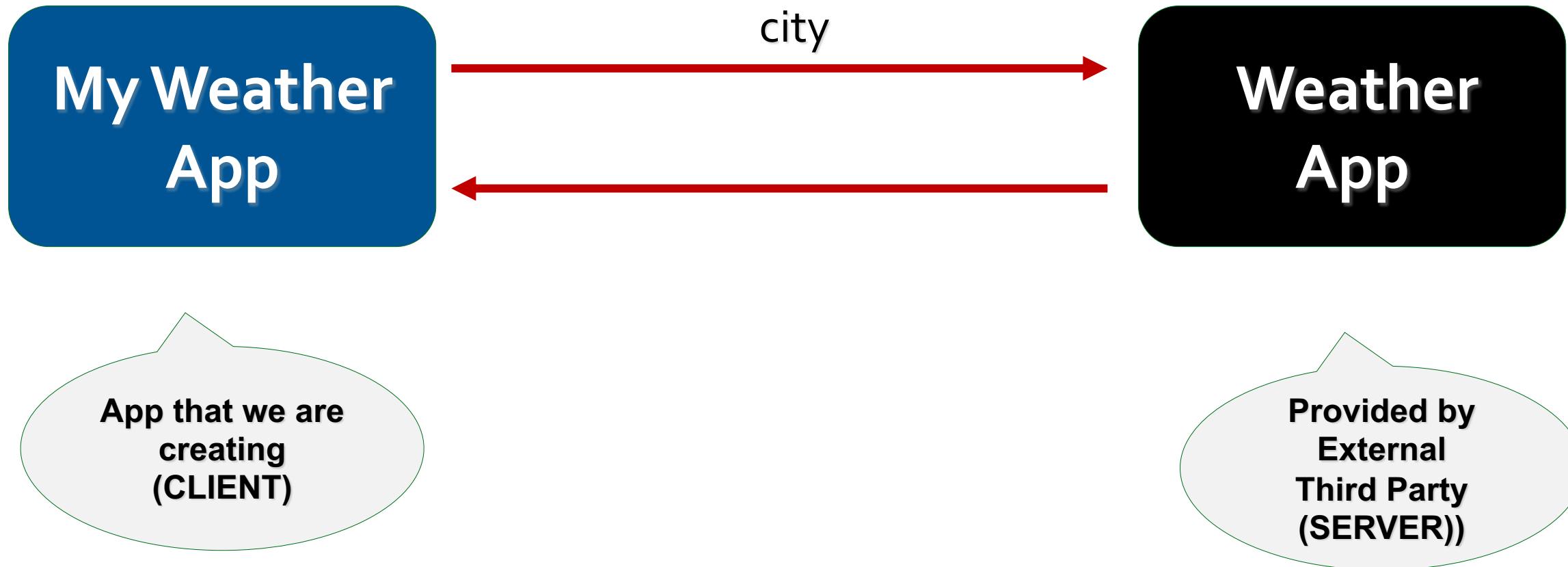
REST APIs - REST Web Services

Business Problem

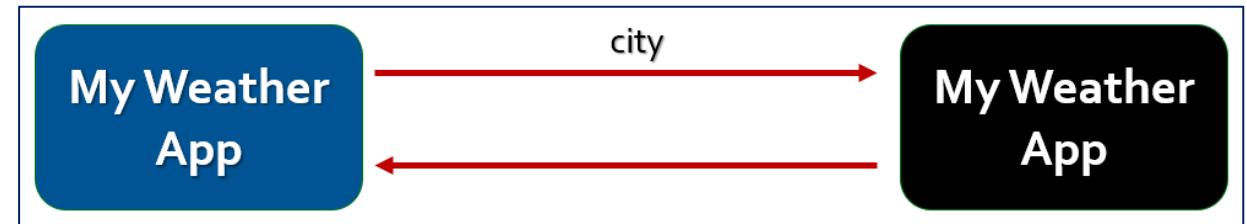


- » Build a **client app** that provides the **weather report** for a city
- » Need to get **weather data** from **an external service**

Application Architecture



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



- » 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

The screenshot shows the OpenWeather API homepage. A red arrow points to the "Weather API" heading, and a blue arrow points to the "Current Weather Data" section. The page includes a search bar, navigation links, and detailed descriptions of the available weather data services.

Weather API

Please [sign up](#) and use our fast and easy-to-work weather APIs for free. Look at our [monthly subscriptions](#) for more options rather than the Free account that we provide you. Read [How to start](#) first and enjoy using our powerful weather APIs.

Current & Forecast weather data collection

Current Weather Data

[API doc](#) [Subscribe](#)

- Access current weather data for any location including over 200,000 cities
- We collect and process weather data from different sources such as global and local weather models, satellites, radars and vast network of weather stations
- JSON, XML, and HTML formats
- Available for both Free and paid subscriptions

Hourly Forecast 4 days

[API doc](#) [Subscribe](#)

- Hourly forecast is available for 4 days
- Forecast weather data for 96 timestamps
- Higher geographic accuracy
- JSON and XML formats
- Available for Developer, Professional and Enterprise accounts

One Call API

[API doc](#) [Subscribe](#)

- Make one API call and get current, forecast and historical weather data
- **Minute forecast** for 1 hour
- **Hourly forecast** for 48 hours
- **Daily forecast** for 7 days
- **Historical data** for 5 previous days
- **National weather alerts**
- JSON format
- Available for both Free and paid subscriptions

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?

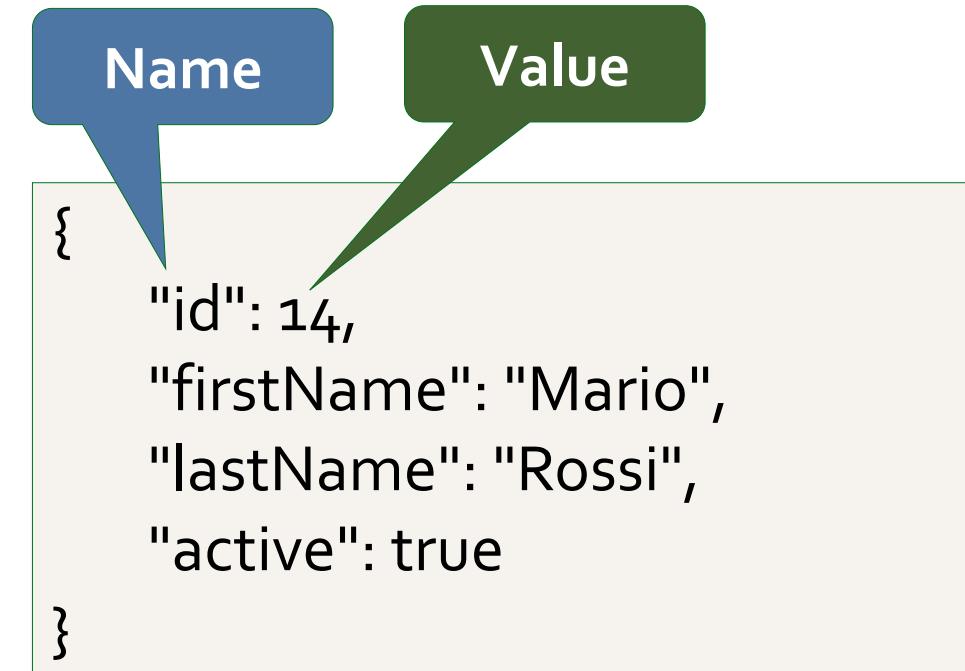
- » **JavaScript Object Notation**
- » **Lightweight data format for storing and exchanging data**
... plain text
- » Language independent ... **not just for JavaScript**
- » Can use with **any programming language**: Java, C#,
Python etc ...

JSON is just
plain text
data

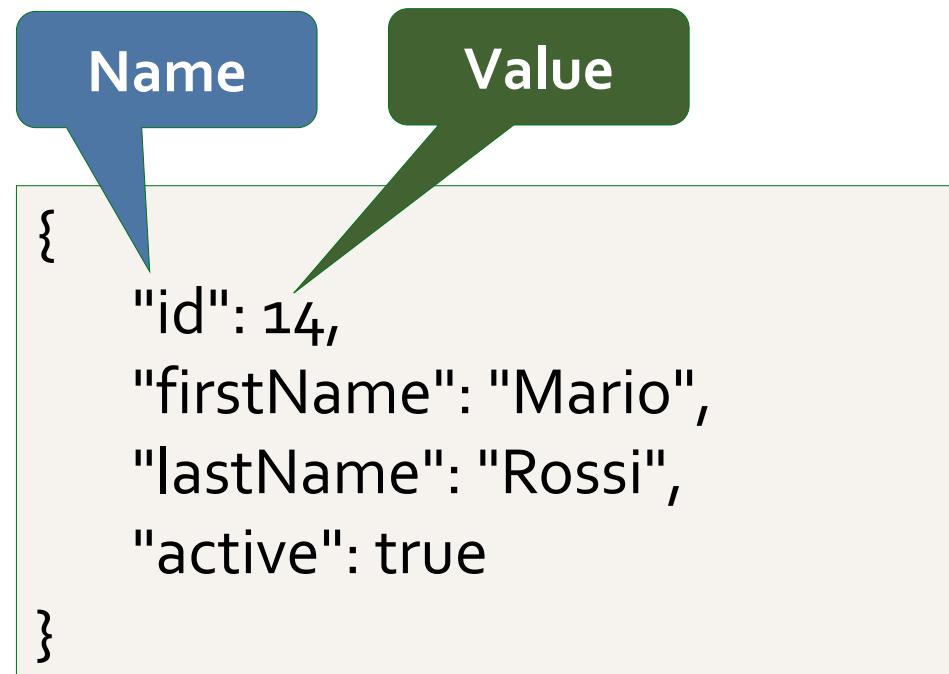
Simple JSON Example



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



- » **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"  
  }  
}
```



A blue speech bubble with the word "Nested" inside it, pointing towards the "address" object in the JSON code.

JSON Arrays



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



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

- » 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  
}
```

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

Call getXXX methods

Java
POJO
Student

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

- » 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



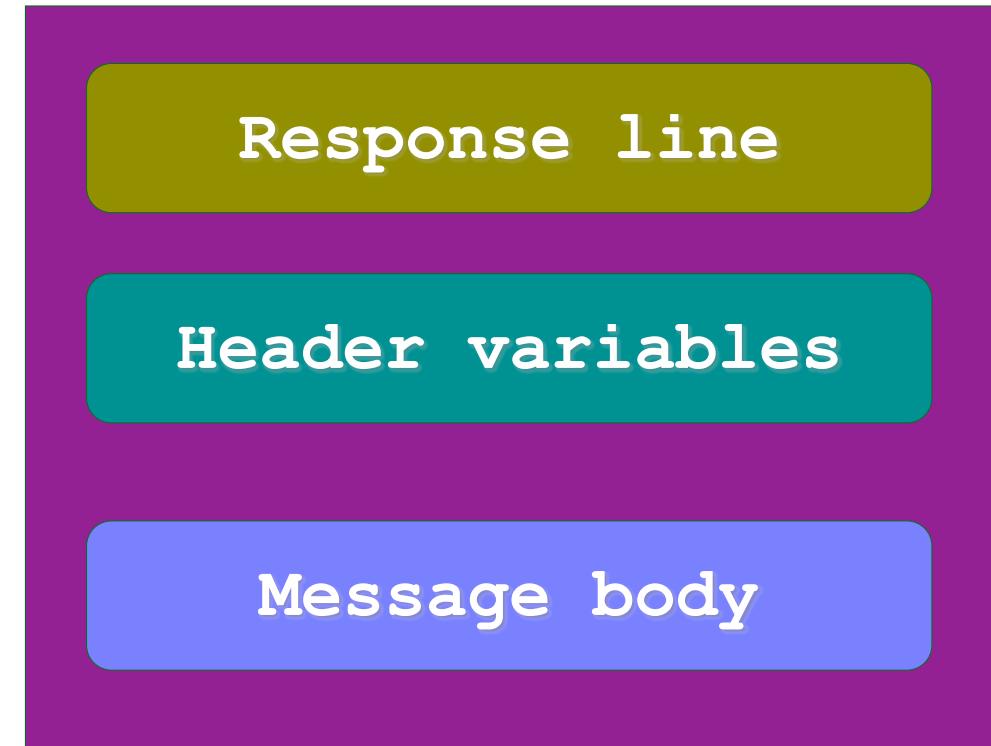
- » **Request line:** the HTTP command/method
- » **Header variables:** request metadata
- » **Message body:** contents of message

HTTP Request Message



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

HTTP Response 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

- » 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**, ...

- » 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



Chrome apps are being deprecated. [Download](#) our free native apps for continued support and better performance. [Learn more](#)

No Environment

https://jsonplaceholder.typicode.com/users

GET https://jsonplaceholder.typicode.com/users Params Send Save

Pretty Raw Preview JSON

```
[  
  {  
    "id": 1,  
    "name": "Leanne Graham",  
    "username": "Bret",  
    "email": "Sincere@april.biz",  
    "address": {  
      "street": "Kulas Light",  
      "suite": "Apt. 556",  
      "city": "Gwenborough",  
      "zipcode": "92998-3874",  
      "geo": {  
        "lat": "-37.3159",  
        "lng": "81.1496"  
      }  
    },  
    "phone": "1-770-736-8031 x56442",  
    "website": "hildegard.org",  
    "company": {  
      "name": "Romaguera-Crona",  
      "catchPhrase": "Multi-layered client-server neural-net",  
      "bs": "User-centric compute-enable convergence"  
    }  
  }]
```

(E) My chosen environment - from Google ...

Client Tool Postman



Chrome apps are being deprecated. [Download](#) our free native apps for continued support and better performance. [Learn more](#)

No Environment

https://jsonplaceholder.typicode.com/users

GET https://jsonplaceholder.typicode.com/users Params Send Save

Pretty Raw Preview JSON

```
1 [  
2 {  
3   "id": 1,  
4   "name": "Leanne Graham",  
5   "username": "Bret",  
6   "email": "Sincere@april.biz",  
7   "address": {  
8     "street": "Kulas Light",  
9     "suite": "Apt. 556",  
10    "city": "Gwenborough",  
11    "zipcode": "92998-3874",  
12    "geo": {  
13      "lat": "-37.3159",  
14      "lng": "81.1496"  
15    }  
16  },  
17  "phone": "1-770-736-8031 x56442",  
18  "website": "hildegard.org",  
19  "company": {  
20    "name": "Romaguera-Crona",  
21  }  
22 }  
23 ]
```

(E) My chosen environment - from Google ...



Spring REST Controller

- » 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**

- » 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 {

}
```

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**

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



GET Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description	...	

Body

Pretty Raw Preview Visualize Text

1 Hello World!

Testing with REST Client - Web Browser





Spring REST - Retrieve POJOs as JSON

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"  
  }]
```

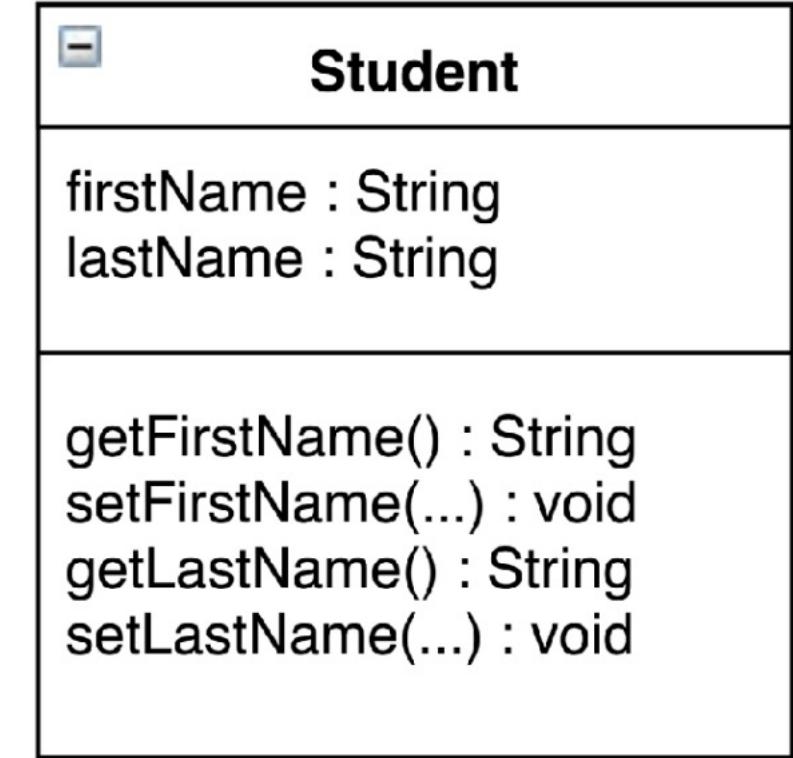
We need to
write this
code

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

Student POJO (class)

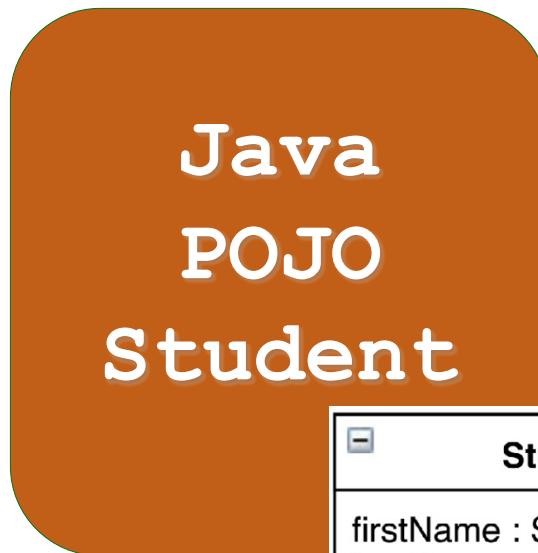


Java
POJO
Student



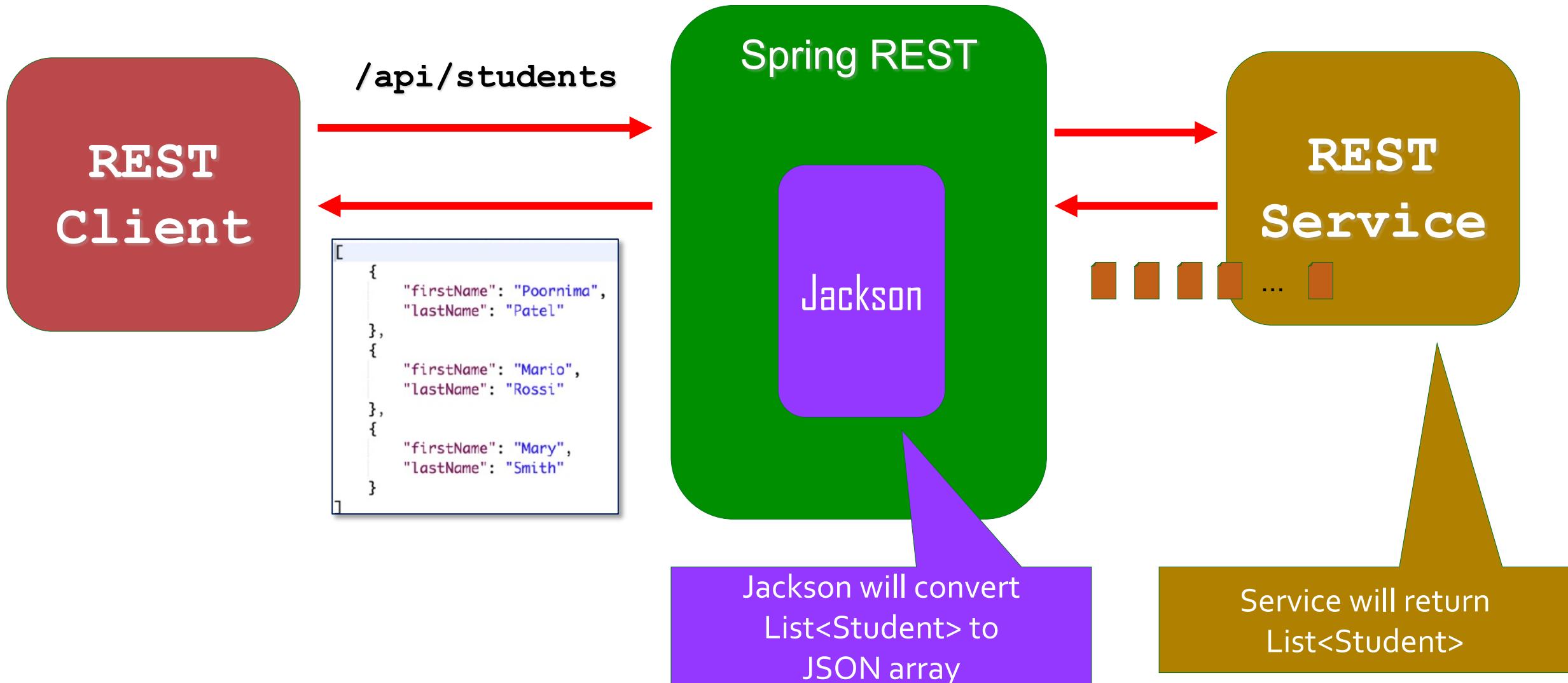
Jackson will call appropriate getter/setter method

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



-	Student
	firstName : String lastName : String
	getFirstName() : String setFirstName(...) : void getLastName() : String setLastName(...) : void

Behind the scenes

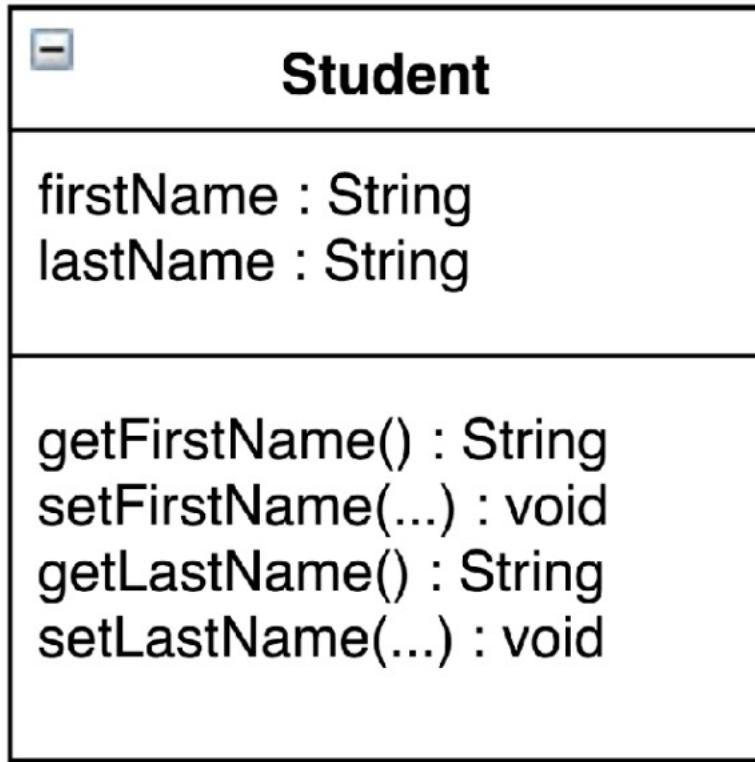


- » Create Java POJO class for **Student**
- » Create **Spring REST Service** using **@RestController**

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



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



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



← → ⌂ ⓘ localhost:8081/spring-rest-demo-pojo-student-list/api/student/999

HTTP Status 500 – Internal Server Error

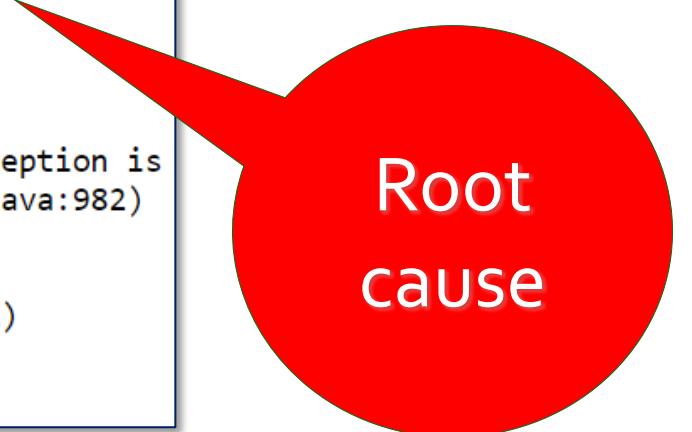
Type Exception Report

Message Request processing failed; nested exception is java.lang.IndexOutOfBoundsException: Index 999 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 failed; nested exception is  
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:982)  
    org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:866)  
    javax.servlet.http.HttpServlet.service(HttpServlet.java:626)  
    org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:851)  
    javax.servlet.http.HttpServlet.service(HttpServlet.java:733)  
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
```



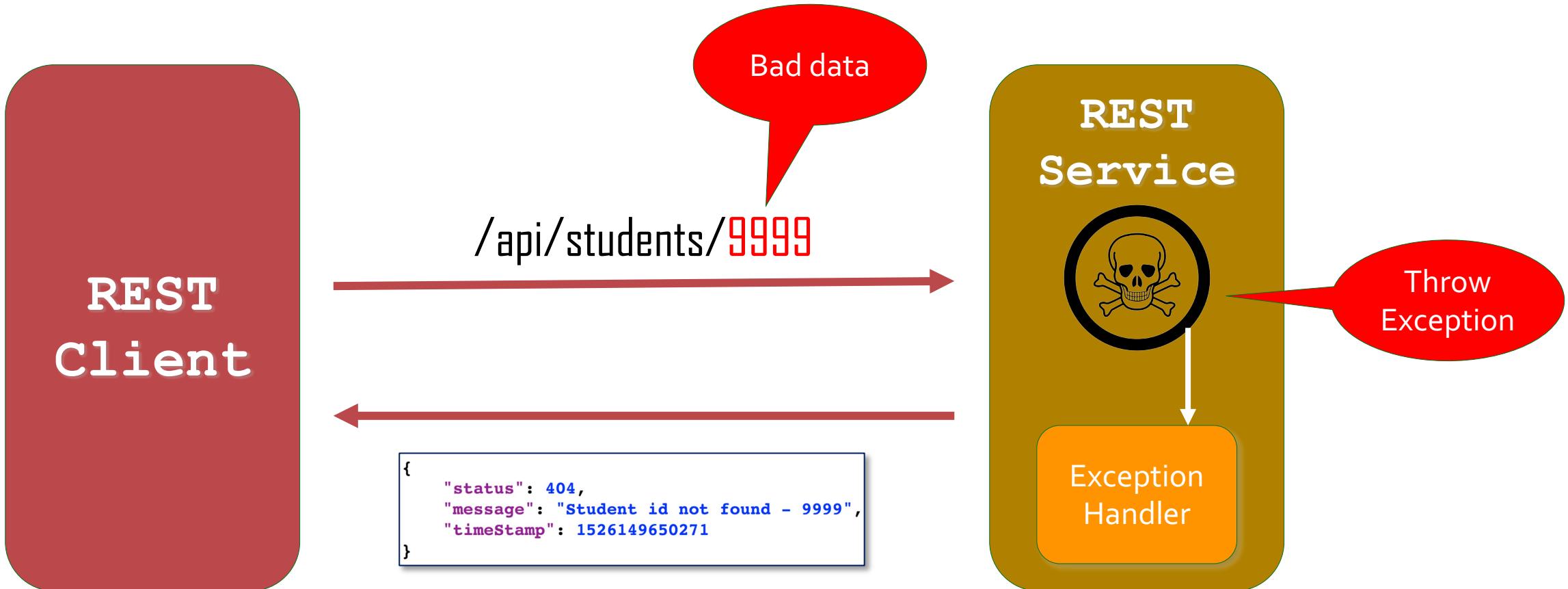
Root cause

Handle the exception and return error as JSON



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

Spring REST Exception Handling



- » 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  
}
```

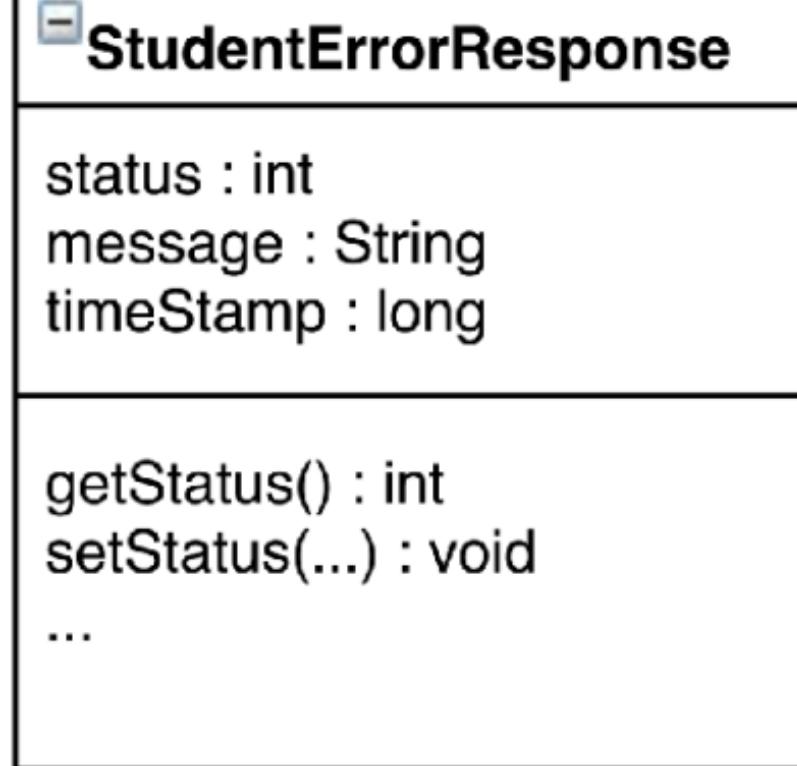
StudentErrorResponse
status : int message : String timeStamp : long
getStatus() : int setStatus(...) : void ...

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 **be 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);
}
```

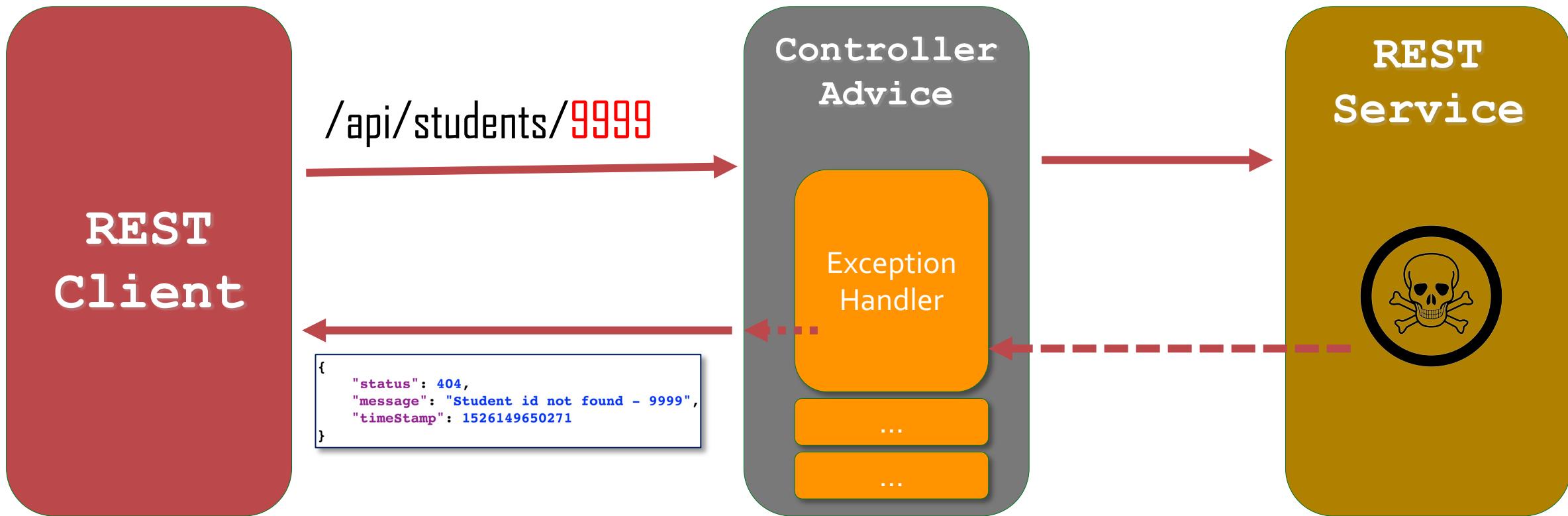
← → ⌂ ⓘ localhost:8081/spring-rest-demo-pojo-student-list/api/students/abcd ⭐ ⚙ ⏺ P :

```
{"status":400,"message":"Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: \"abcd\"","timeStamp":1616813232257}
```

- » 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

- » @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



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

Step 1: Create new @ControllerAdvice



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

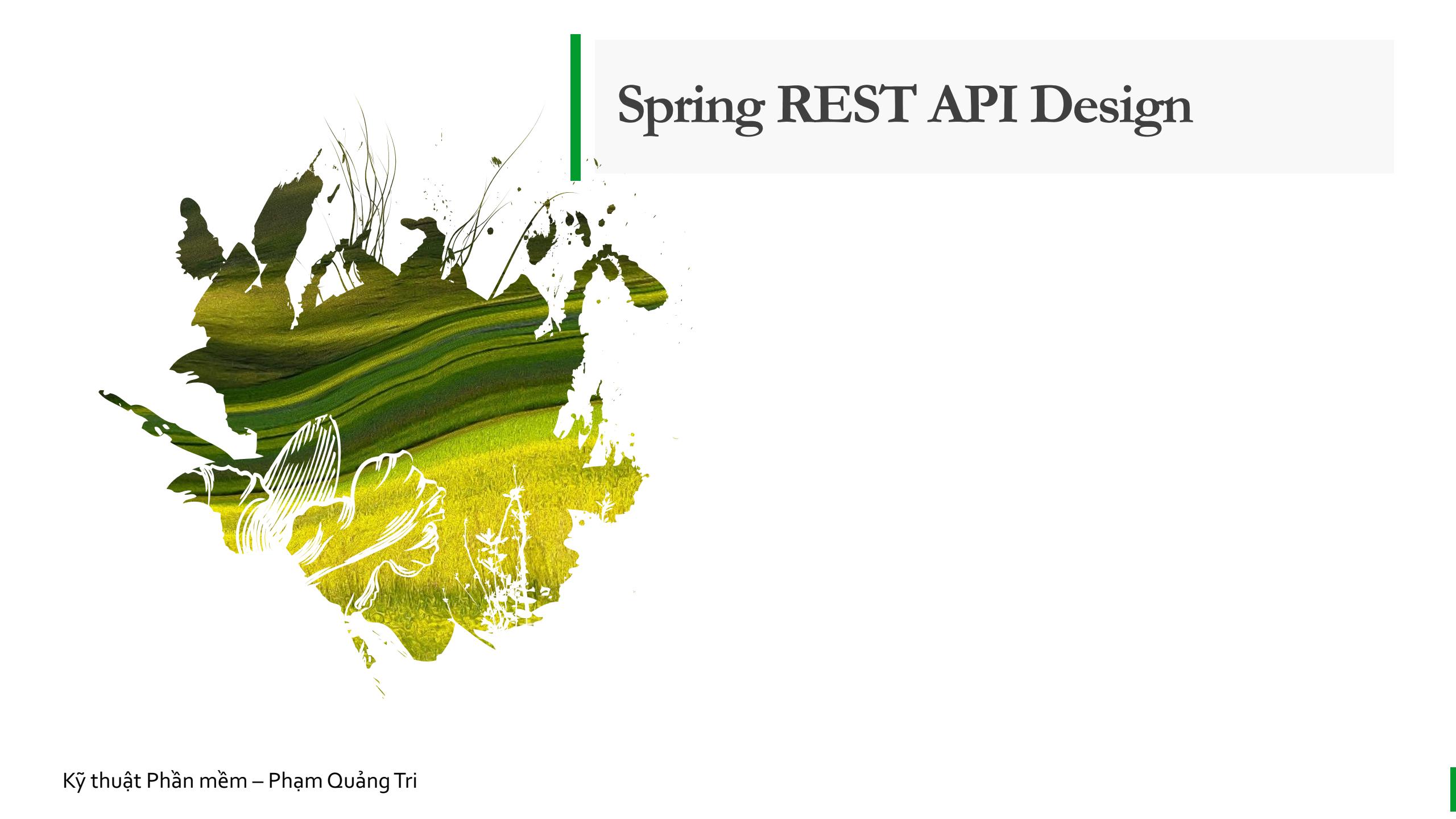
```
@ControllerAdvice
public class StudentRestExceptionHandler {
    @ExceptionHandler
    public ResponseEntity<StudentErrorResponse> handleException(Exception exc) {
        // create a StudentErrorResponse
        StudentErrorResponse error = new StudentErrorResponse();
        error.setStatus(HttpStatus.BAD_REQUEST.value());
        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

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

- » **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 customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing entity
DELETE	/api/customers/{customerId}	Delete an existing entity

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



YES

Instead, use
HTTP methods
to assign
actions



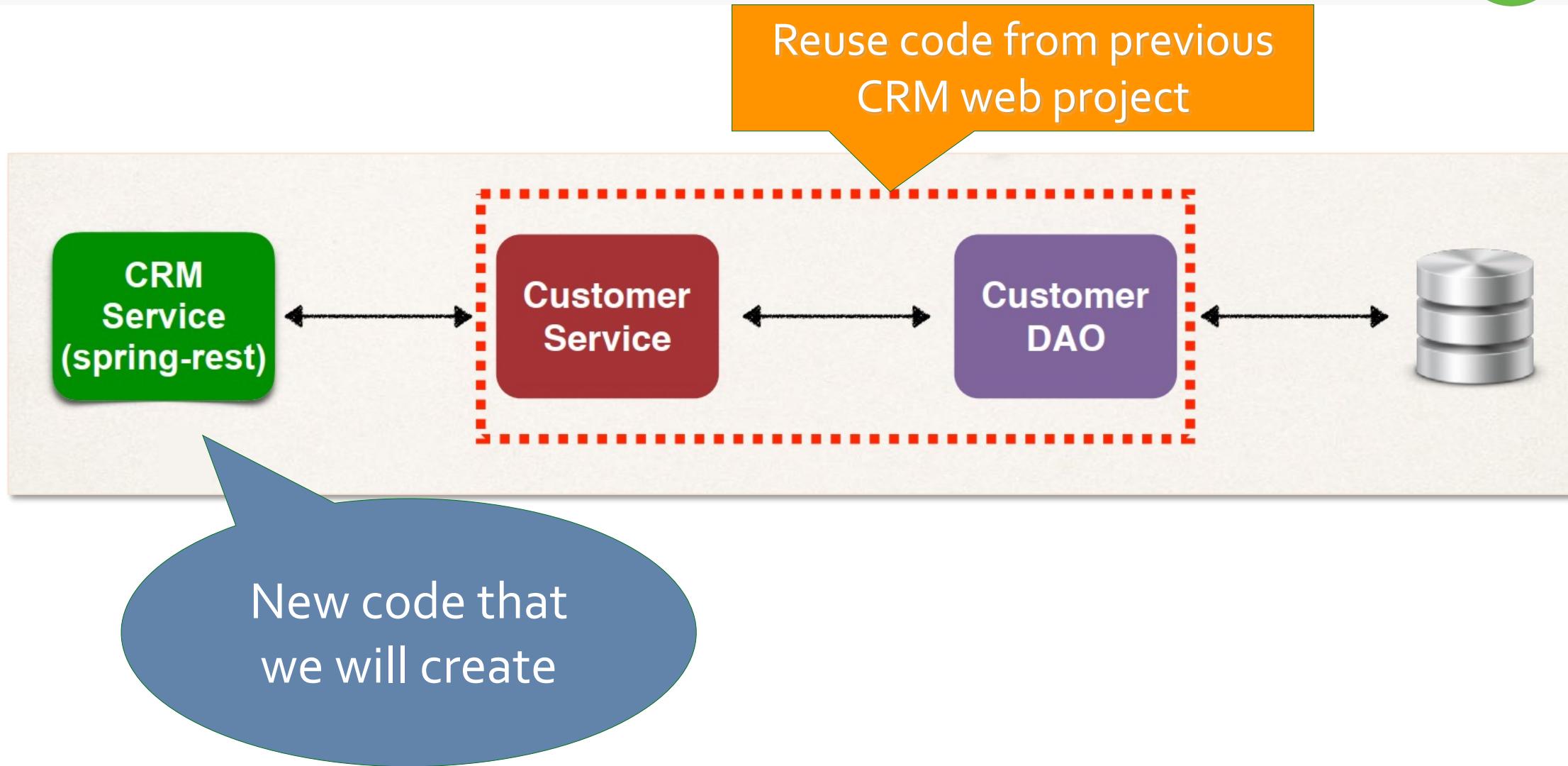
Spring REST API - Real Time Project

Real Time Project



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

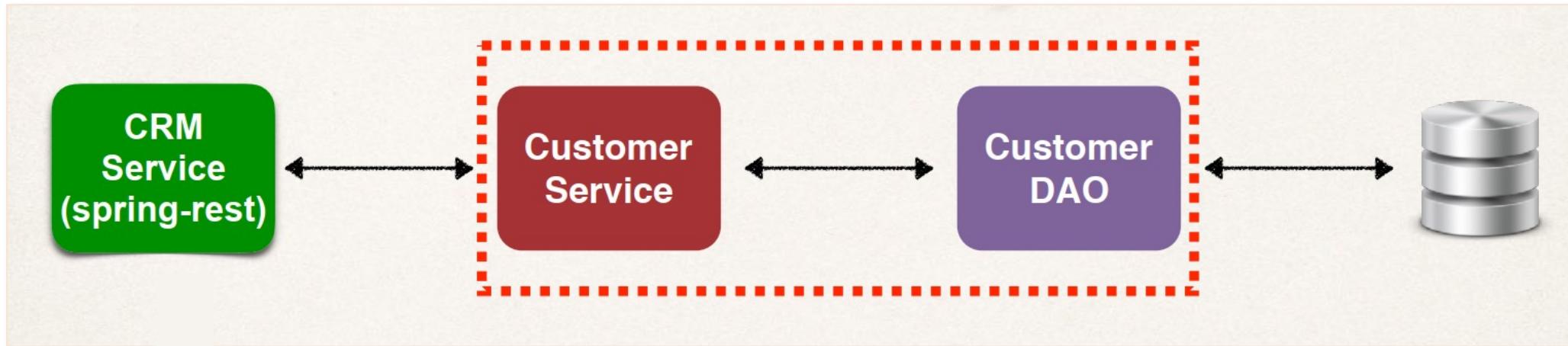
Application Architecture



Project Set Up



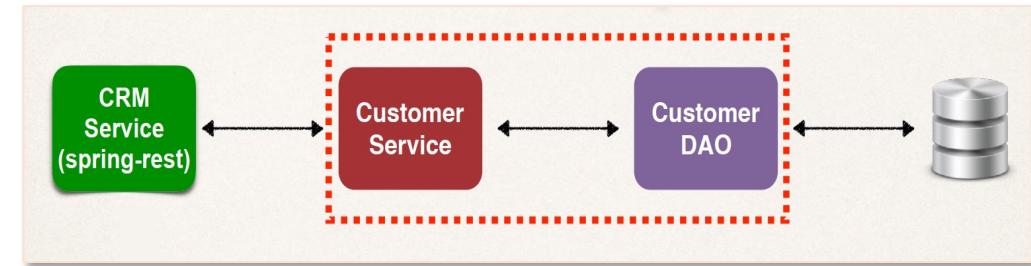
- » Setup Maven project
- » Copy CustomerService, CustomerDAO and Customer entity from previous CRM Spring MVC Project



Development Process



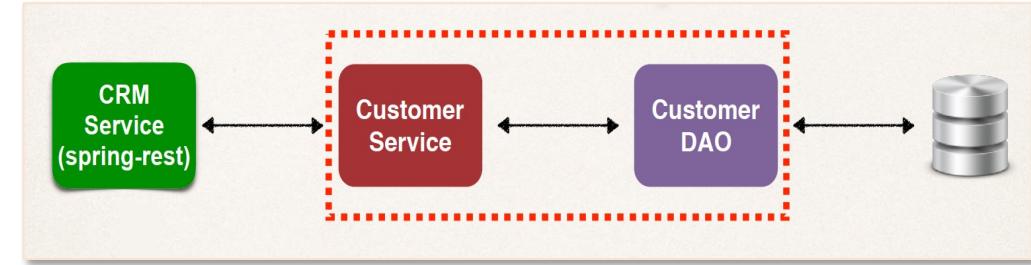
- » Get customers
- » Get single customer by ID
- » Add a new customer
- » Update an existing customer
- » Delete an existing customer



Review Task List



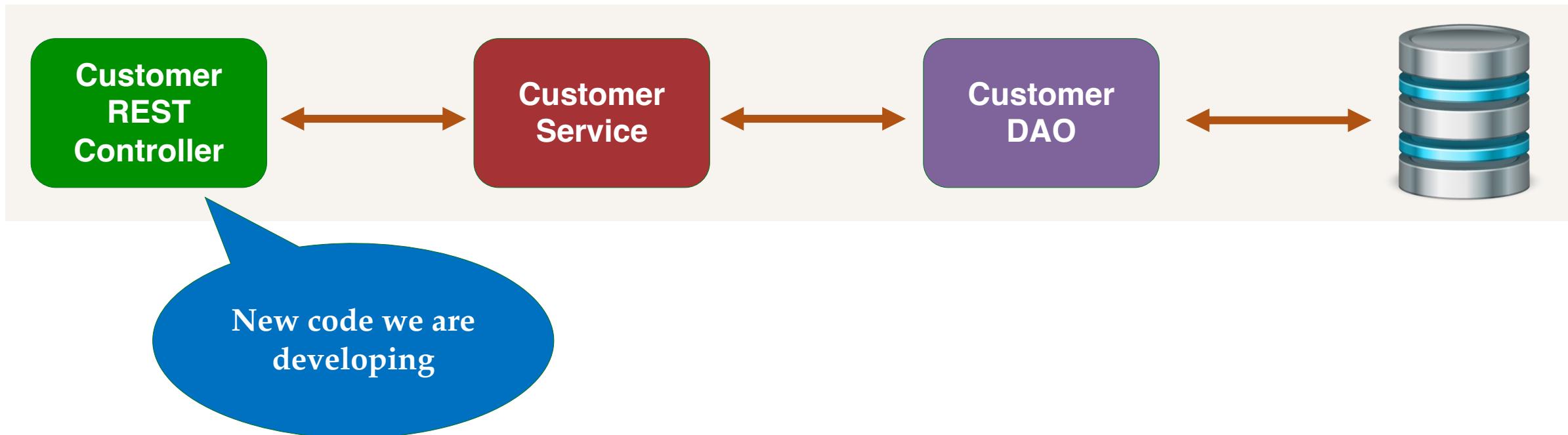
- » pom.xml file
- » All Java Config
- » Configuration file for database connection
- » Hibernate Entity class: Customer
- » DAO: CustomerDAO ...
- » Service: CustomerService ...



Application Interaction



Application Architecture



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

Development Process



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

» **@RestController**

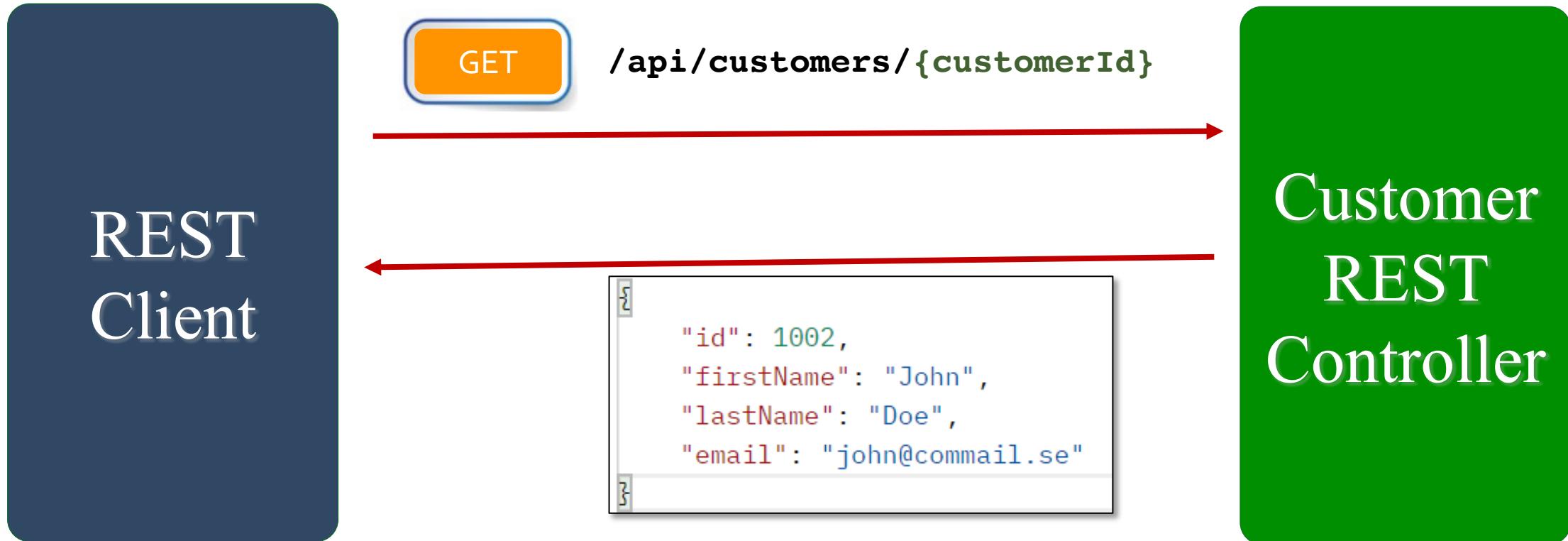
```
@RequestMapping("/api")
public class CustomerRestController {
    // autowire the CustomerService
    @Autowired
    private CustomerService customerService;
    // add mapping for GET /customers
    @GetMapping("/customers")
    public List<Customer> getCustomers() {
        return customerService.getCustomers();
    }
}
```

1

2

3

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;
    }
}
```

Application Interaction



Add Customer



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

ID of 0 means
DAO code will
“INSERT” new customer

- » 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



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

POST Send

Params Authorization Headers (8) Body Cookies

Body (radio buttons): none, form-data, x-www-form-urlencoded, raw, binary, GraphQL, JSON (selected)

Beautify

```
1
2   "firstName": "Minh",
3   "lastName": "Pham",
4   "email": "pham@gmail.com"
5
```

Body Cookies Headers (6) Test Results Status: 200 OK Time: 474 ms Size: 259 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2   "id": 2005,
3   "firstName": "Minh",
4   "lastName": "Pham",
5   "email": "pham@gmail.com"
6
```

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



The screenshot shows the Postman interface for a PUT request to update a customer. The URL is `http://localhost:8080/spring-rest-crm-crud-demo/api/customers`. The request body is a JSON object:

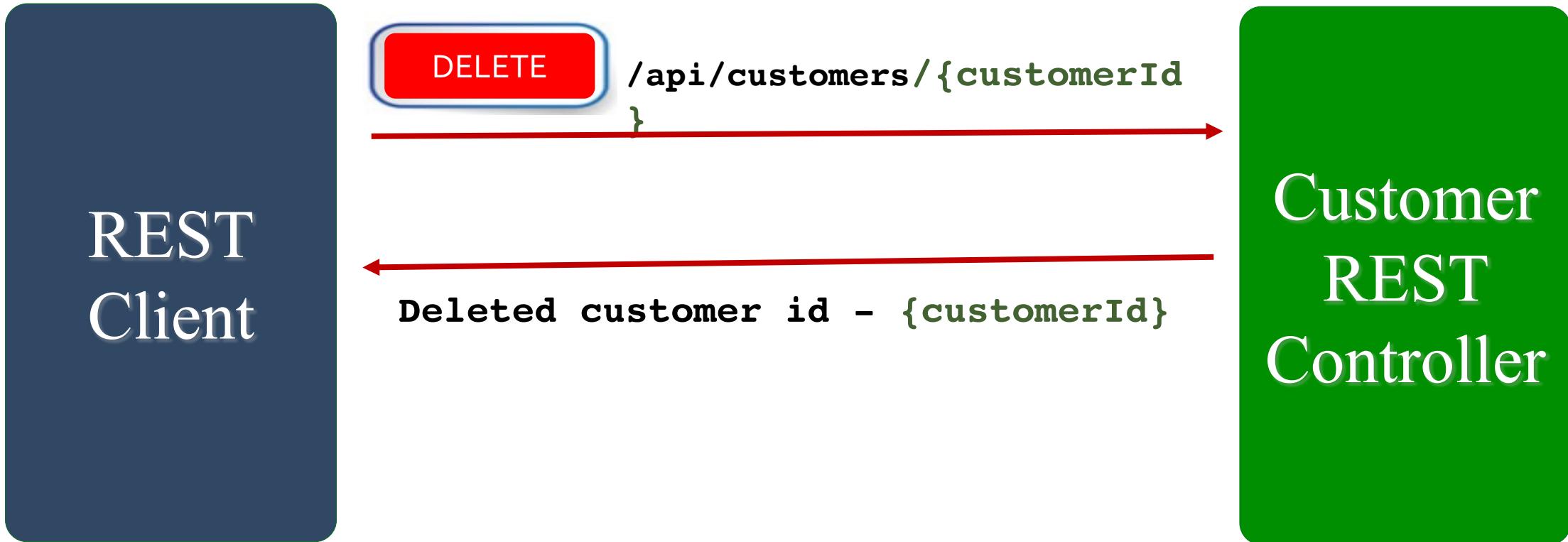
```
1 {  
2   "id": 2005,  
3   "firstName": "Minh",  
4   "lastName": "Pham",  
5   "email": "phamminh@gmail.com"  
6 }  
7
```

The response status is 200 OK with a time of 34 ms and a size of 263 B.

Body	Cookies	Headers (6)	Test Results	Save Response
Pretty	Raw	Preview	Visualize	JSON

```
1 {  
2   "id": 2005,  
3   "firstName": "Minh",  
4   "lastName": "Pham",  
5   "email": "phamminh@gmail.com"  
6 }
```

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



DELETE http://localhost:8080/spring-rest-crm-crud-demo/api/customers/2005

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautify

1

Body Cookies Headers (6) Test Results Status: 200 OK Time: 232 ms Size: 217 B

Pretty Raw Preview Visualize

1 Deleted customer id - 2005

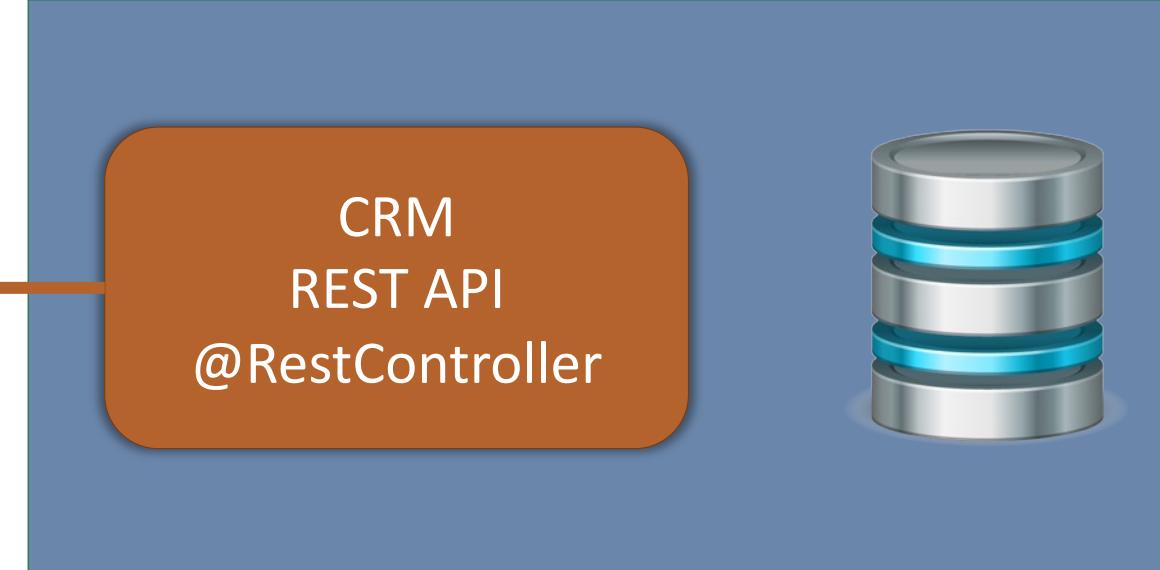


REST Client with Spring

CRM REST Client with Spring



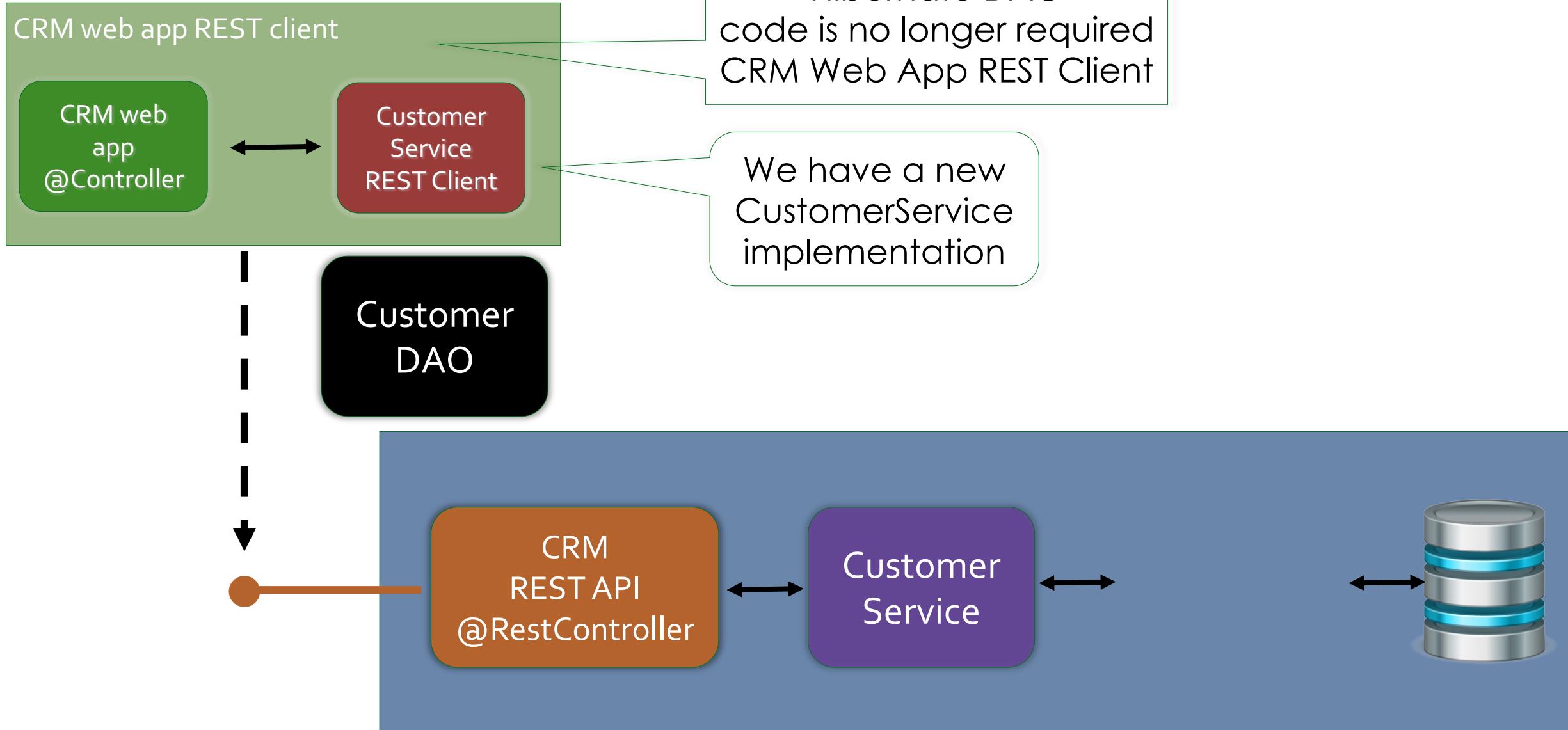
CRM - Customer Relationship Manager			
First Name	Last Name	Email	Action
John 5	Doe	john@commail.se	Update Delete
John	Doe	john@secode.com	Update Delete
David	Gueta	Gueta@secode.com	Update Delete
Kien	Le	le@gmail.com	Update Delete
Minh	Pham	pham@gmail.com	Update Delete
Minh	Pham	minh@mail.com	Update Delete
Mary	Public	mary@secode.com	Update Delete
Pham	Quang	phamquangtri@gmail.com	Update Delete
Trung	Tran	tran@gmail.com	Update Delete
Thanh	Tran	thanh@mail.com	Update Delete
Hong	Trinh	trinh@mail.com	Update Delete



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() { ... }
    // define bean for RestTemplate ... this 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

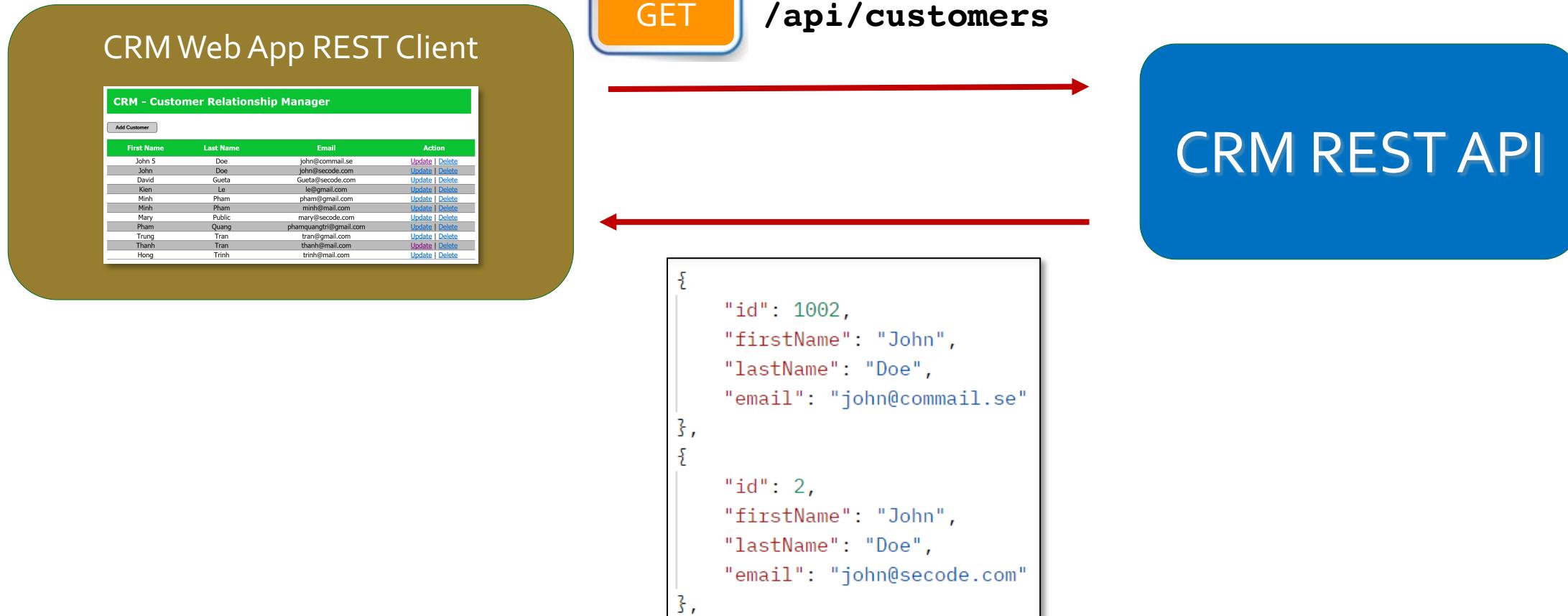


File: CustomerServiceRestClientImpl.java (project: spring-rest-crm-crud-client-demo)

```
@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: Get a list of Customers



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

```
@Override
```

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

```
// make REST ca
```

```
ResponseEntity<List<Customer>> responseEntity =
```

request headers or body,
null in our case

```
restTemplate.exchange(crmRestUrl, HttpMethod.GET, null,
```

HTTP method for GET

```
new ParameterizedTypeReference<List<Customer>>() {
```

```
// get the li
```

customers from response

```
responseType: the type of the return value
```

```
logger.info("in getCustomers(): customers" + customers);
```

```
return customers;
```

```
}
```

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



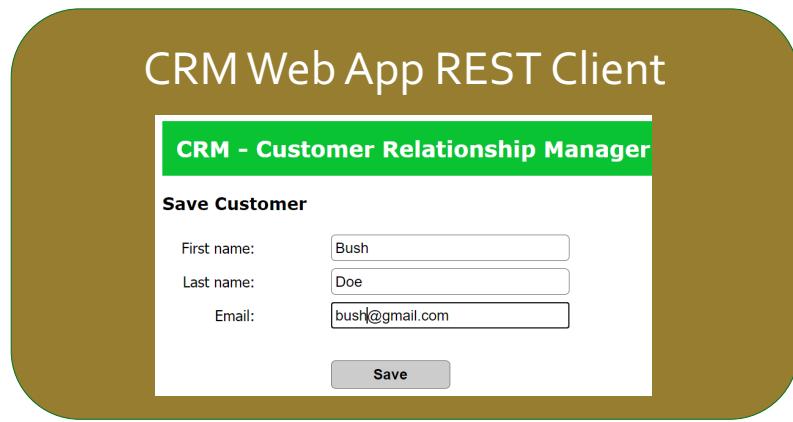
POST: Add a new Customer



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

/api/customers

```
{  
    ...  
    "id":6005,  
    ...  
    "firstName": "Bush",  
    ...  
    "lastName": "Doe",  
    ...  
    "email": "bush@gmail.com"  
}
```

CRM REST API

```
{  
    ...  
    "id":6005,  
    ...  
    "firstName": "Bush",  
    ...  
    "lastName": "Doe",  
    ...  
    "email": "bush@gmail.com"  
}
```

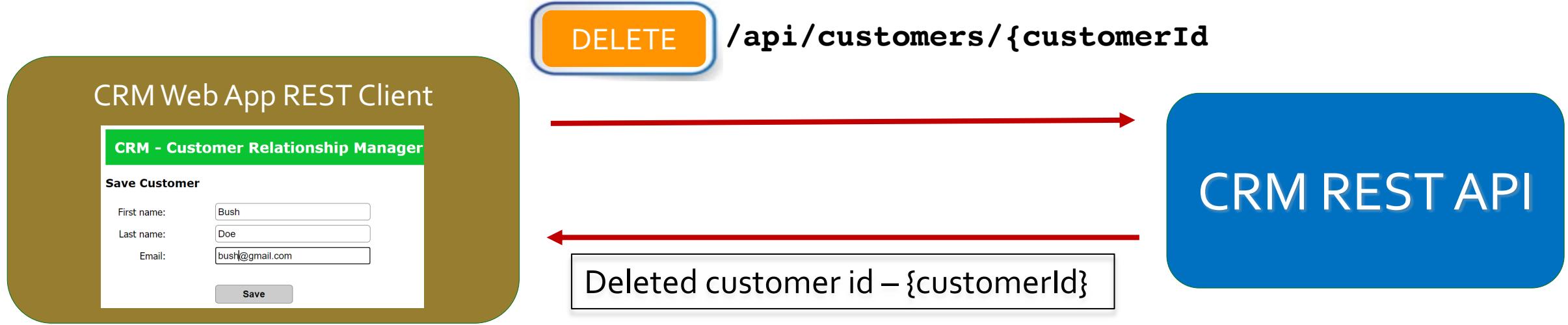
PUT: Edit an existing Customer



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