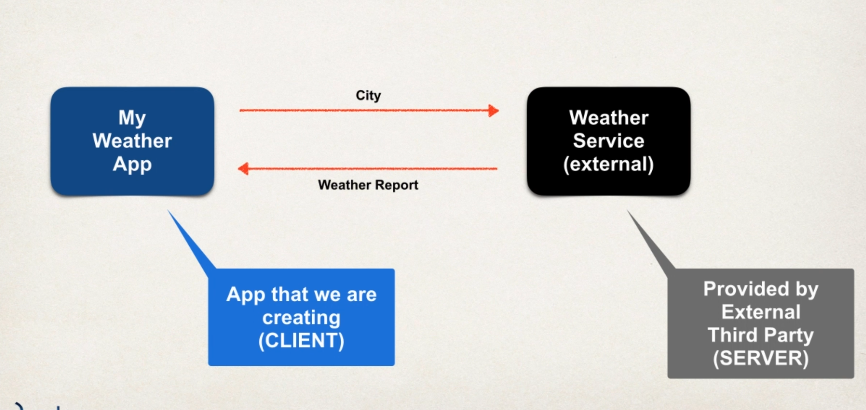
**Business Problem**

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

**Application Architecture**

****

**Questions**

1. **How will we connect to the Weather Service?**

* We make REST API calls over HTTP
* REST: Representational State Transfer
* Lightweight approach for communicating between applications.

1. **What programming Language do we use?**

* REST is language independent.
* Client application can use ANY programming language
* Server application can use ANY programming language

1. **What is the data format?**

* **Can use any data format**
* Commonly used are XML and JSON
* JSON is most popular and modern
  + JavaScript Object Notation

**Where to find REST APIs?**

* [**www.programmableweb.com**](http://www.programmableweb.com)

**So, what do we call it?**

REST API or RESTful API or REST Web Services or RESTful Web Services or REST Services or RESTful Services?

Generally, all mean the same thing.

**What is JSON?**

* **J**ava**S**cript **O**bject **N**otation
* Lightweight data format for storing and exchanging data (JSON is just Plain Text data)
* Language Independent. Not just for JavaScript
* Can use with any programming language: Java, C#, Python etc.

**Simple JSON Example**

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

Example:

{

“id”: 14,

“firstName”: “Sayantan”,

“lastName”: “Sengupta”,

“”: true

}

**JSON Values**

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

**Nested JSON Objects**

{  
  "name":"John",  
  "age":30,  
  "cars": {  
    "car1":"Ford",  
    "car2":"BMW",  
    "car3":"Fiat"  
  }  
 }

**JSON Data Binding with Jackson**

* Data Binding is the process of converting JSON data to a Java POJO or vice-versa.
* 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.
* Details on Jackson Project: <https://github.com/FasterXML/jackson-databehind>

**Jackson Data Binding**

* Jackson Data Binding API
  + Package: ***com.fasterxml.jackson.databind***
* Maven Dependency
* By default, Jackson will call appropriate getter/setter methods.
  + While converting from JSON to POJO, it will call ***setter methods.***
  + While converting from POJO to JSON, it will call ***getter methods***.

**Note: Jackson calls the setter and getter methods. It does not access internal private fields directly.**

**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 and Java object being returned from REST controller is converted to JSON. It happens automatically behind the scenes.

If Spring finds Jackson on the classpath (for example Maven pom.xml), then Spring will use Jackson for the conversion.

Jackson is a separate project from Spring. But Spring is smart enough to detect if Jackson is on the classpath via a dependency. If Spring does not find Jackson or no other suitable converter, then it will throw an exception. We also have the option of providing a different converter for Jackson. For example, you can use another library such as [Gson](https://github.com/google/gson). Here's a tutorial that shows how to use Gson in Spring Boot for generating JSON.

**Configuring Spring Boot to use Gson instead of Jackson**

<https://www.callicoder.com/configuring-spring-boot-to-use-gson-instead-of-jackson/>

**Converting JSON into POJO**

**Development Process**

1. **Download and import Maven starter files.**
2. **Add Maven dependency for Jackson Project**
3. **Create Entity POJO Java Class**
4. **Create main Driver App.**

**Main Development starts from step 2**

****

**Entity POJO class**

****

****

**Driver App**

1. The ObjectMapper comes from the Jackson project
2. **Reading data from the “data/sample.json”** and **Creating an instance of the entity class and populate it. Jackson will do all the mapping for the user via the mapper object.**

****

**JSON has a property but the user doesn’t care about.**

**Use Case:**

**A new property is added to JSON. Our code is not aware of it. Causes an exception.**

**Add this following annotation in the entity class.**

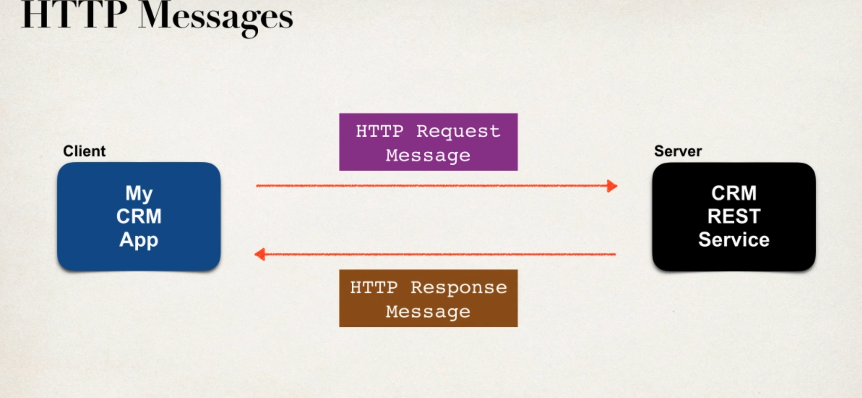
****

**Spring REST – Using @PathVariable for REST Endpoints**

**REST HTTP BASICS**

* Most common use of REST is over HTTP
* Leverage HTTP methods for CRUD operations.

|  |  |
| --- | --- |
| **HTTP Method** | **CRUD Operation** |
| **POST** | **C**reate a new Entity |
| **GET** | **R**ead a list of entities or single entity |
| **PUT** | **U**pdate an existing entity |
| **DELETE** | **D**elete an existing entity |

****

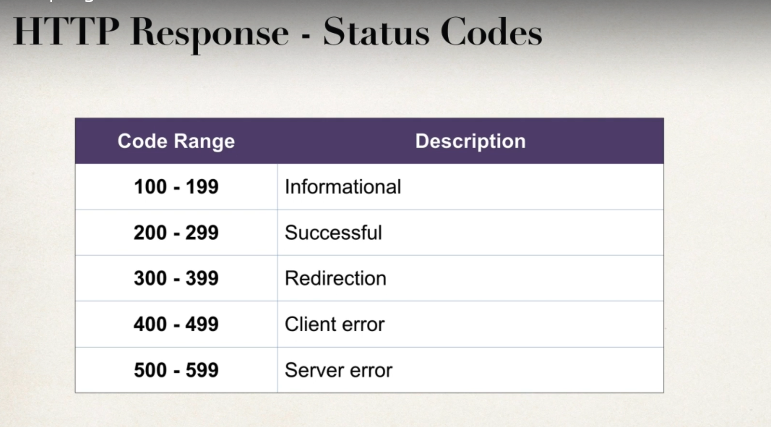
**HTTP Request Message**

* **Request Line**: HTTP Command (GET/PUT/UPDATE/DELETE)
* **Header variables**: request metadata
* **Message body:** contents of message

**HTTP Response Message**

* **Response Line**: server protocol and status code (HTTP Status code)
* **Head variables**: response metadata (content type of data, size of data, etc.)
* **Message body**: contents of message.

HTTP Response – Status Codes

****

**MIME Content type**

* 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**

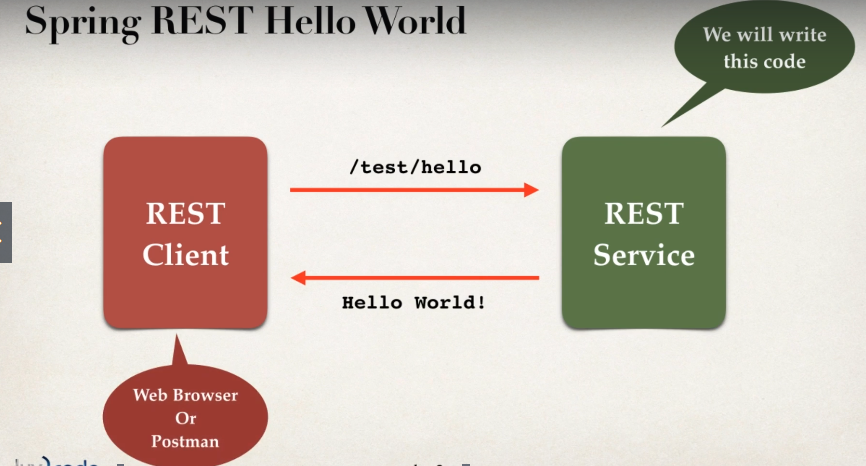
A client toll is required.

* Send HTTP requests to the REST Web Service/API
* Plenty of tools available: curl, Postman, etc.

**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**

****

**Development Process**

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**
   1. **Web App Initializer**
      1. **Spring MVC provides support for web app initialization**
      2. **Makes sure that code is automatically detected**
      3. **Code is used to initialize the servlet container**

AbstractAnnotationConfigDispatcherServletInitializer

* + - * **Extend the abstract base class**
      * **Override required methods**
      * **Specify servlet mapping and location of app config**.

Links - [**https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/WebApplicationInitializer.html**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/WebApplicationInitializer.html)

1. **Create Spring REST Service using @RestController**
   * + 1. **Adding Maven Dependencies for Spring MVC and Jackson project**



* + - 1. Adding **code for All Java Config: @Configuration**



**For just to tell Spring where to find all our classes, beans, components, controllers etc.**

* + - 1. **Adding code for All Java Config: Servlet Initializer**

****

* **How does spring detect our Dispatcher Servlet Initializer?**

The Servlet framework includes the concept of "initializers". It is code that is "automatically" loaded when the server starts up.

The Spring MVC framework provides support for this "initializer" feature via the WebApplicationInitializer interface. When the Spring container is initialized then it will scan for classes that implement the WebApplicationInitializer interface. It will automatically load those classes at start up. No need to explicitly call the classes / components or annotate them.

In our example, we are using MySpringMvcDispatcherServletInitializer with extends off of AbstractAnnotationConfigDispatcherServletInitializer.

AbstractAnnotationConfigDispatcherServletInitializer eventually inherits from parent classes that implement the WebApplicationInitializer interface.

That explains how the MySpringMvcDispatcherServletInitializer is automatically loaded.

Here are docs for [AbstractAnnotationConfigDispatcherServletInitializer](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/support/AbstractAnnotationConfigDispatcherServletInitializer.html)

And here are docs for [WebApplicationInitializer](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/WebApplicationInitializer.html).

* **Why creating separate class for dispatcher servlet initializer?**

There are other ways of setting up all Java configuratin. For example, in the old days, we used the type of coding in the blog post below.

<https://samerabdelkafi.wordpress.com/2014/08/03/spring-mvc-full-java-based-config/>

But the modern approach is use of the coding provided in the course videos. In the video we extend: [AbstractAnnotationConfigDispatcherServletInitializer](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/support/AbstractAnnotationConfigDispatcherServletInitializer.html). This is the modern approach.

**Difference between @Controller and @RestController**

[**https://javarevisited.blogspot.com/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html**](https://javarevisited.blogspot.com/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html)

**4. Creating a Spring REST Service using @RestController**

****

**There will be an error 404 when we will run this but it will still work if we provide the full path in the error i.e.** <http://localhost:8081/spring-rest-demo/test/hello>

However, if want to remove the error, we need to add a view page.

****

**And if we want to give forward slash before “test/hello”, we need to write the full path –**

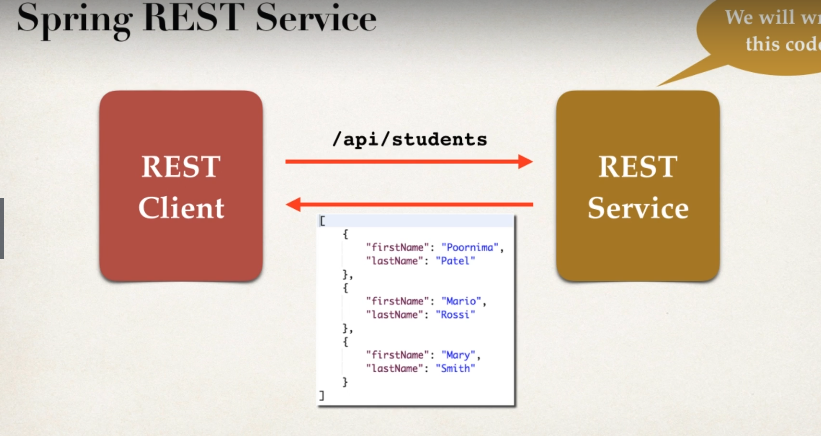
${pageContext.request.contextPath}*/test/hello*

**MORE EXPLANATION NEEDED IN DEPTH**

**Retrieve POJOs as JSON**

**Create a New Service**

* Return a list of students



**Convert Java POJO to JSON**

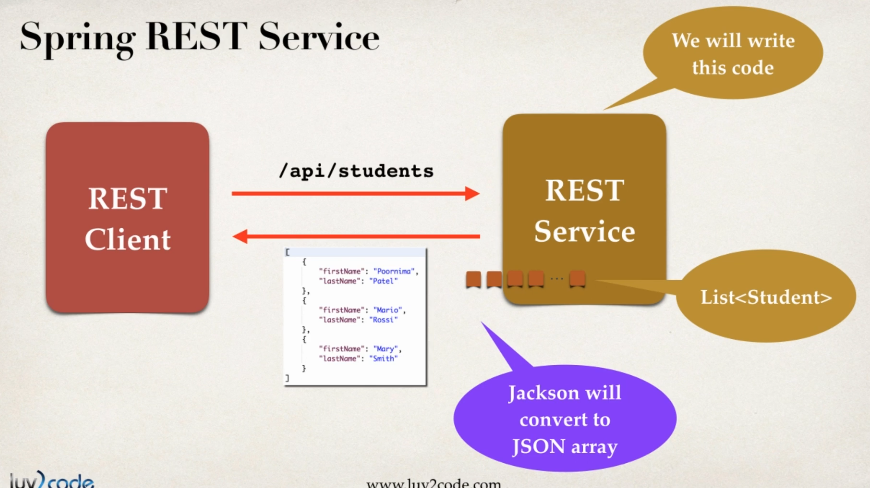
* REST Service will return List<Student>
* Need to convert List<Student> to JSON
* Jackson can help us out with this.

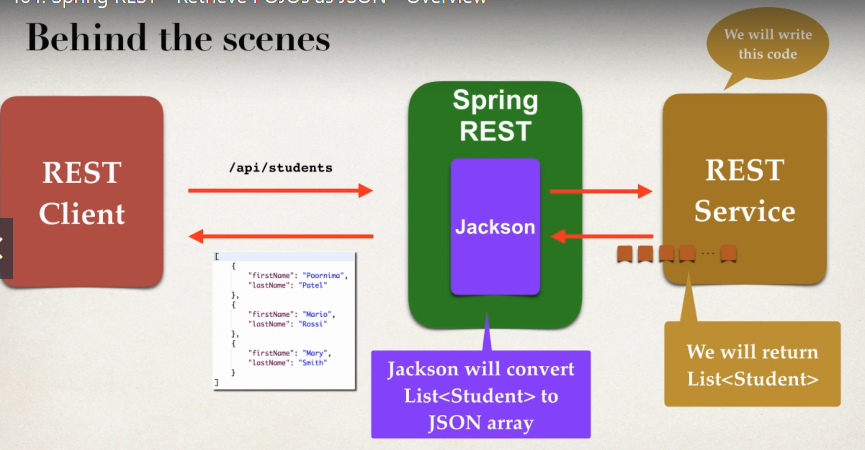
**Spring and Jackson Support (Spring will handle this automatically behind the scenes)**

* Spring will automatically handle JACKSON integration
* As long as 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

**Jackson Data Binding**

* Jackson will call appropriate getter/setter method





**Development Process**

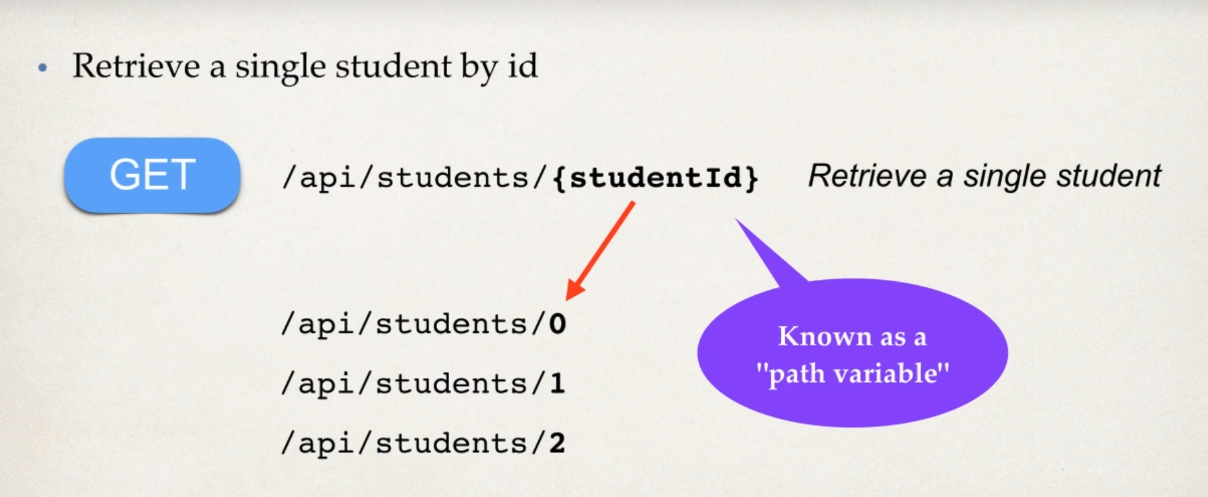
1. **Create Java POJO class for entity**
2. **Create Spring REST Service using @RestController**
3. **Creating Java POJO Class**

****

1. **Creating SPRING Rest Service**

****

**Using @PathVariables for REST Endpoints**

****

**Development Process**

* **Just add another end point w.r.t the previous section in this document.**

****

**SPRING REST – EXCEPTION HANDLING**

**In this present scenario, we previously know that there are 4 students in the array list. So, the Path Variable should be in the range of 0-3 (since array list indexing starts from 0). But what will happen if we give some path variable beyond the range such as 99. It will create an 500 internal server error whose root cause is *Out-of-bounds* error. In order to handle that we need to apply exception handling.**

**THE USER WANTS TO HANDLE THE EXCEPTION AND RETURN ERROR AS JSON**

**{**

**“status”: 404,**

**“message”: “Student id not found - 99”,**

**“timeStamp”: 152614965271**

**}**