**REST Webservices**

REST stands for **RE**presentational **S**tate **T**ransfer.

This is a light weight approach for communication between applications. In this approach the client and server can be implemented in two different languages but still REST helps with the communication between them.

REST applications can use any data format but the most commonly used formats are XML and JSON (JavaScript Object Notation).

Weather Service

My App

Swift (Apple app)

My App

Spring MVC

My App C#

**JSON Data Binding**

JSON – JavaScript Object Notation

JSON is a light weight data format for storing and exchanging data (plain text format) and is language independent.

JSON object has key value pairs and the key will always be in double quotes and this whole thing is enclosed in curly braces.

Example JSON:

{

“id”:3,

“firstName”:”Naveen”,

“lastName”:”Ch”,

“active”: true,

“course”:null,

“Address”:{

“city”:”Morrisville”,

“country”:”India”,

“street”:”Chambal”

},

“hobbies”:[“stamp collection”,”riding bikes”,”sleeping”]

}

JSON Values

* Number: no quotes
* String: double quotes
* Boolean: true, false
* Nested JSON objects: Another JSON object with in the JSON object.
* Array
* null

**Java JSON Data binding**

Data binding is a process of converting JSON data into a Java POJO or the other way from Java POJO to JSON data. This is also known as Mapping or serialization/deserialization or marshalling/unmarshalling.

JSON Data

Java POJO

Data binding

Data binding also called Mapping or serialization/deserialization or marshalling/unmarshalling.

**JSON data binding with Jackson**

* Spring uses Jackson binding behind the scenes
* Jackson binding converts JSON data to Java POJO and vicerversa.

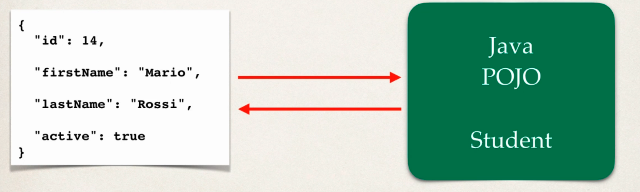
Actually Jackson binding is a separate project that spring uses for JSON binding. Jackson databinding api also support XML binding.

Name of the project: **jackson-databind**

So in order for Spring to use this for JSON binding, the dependency should be included in the maven or should have this jar file added in the project class path.

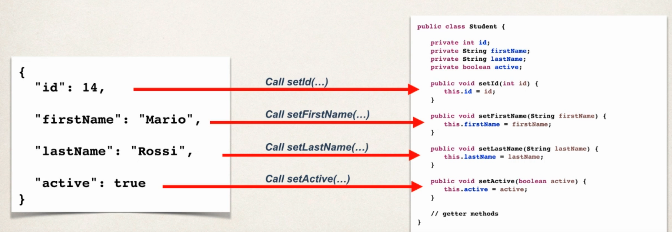
How Jackson databinding api works:

* Jackson by default uses setters/getters methods of a POJO



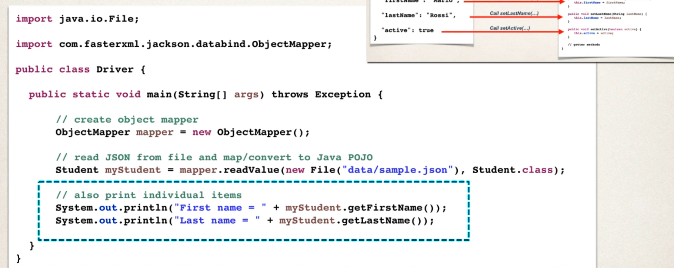
JSON to Java POJO scenario

Convert JSON to Java POJO… Jackson calls setter methods on POJO to convert this.



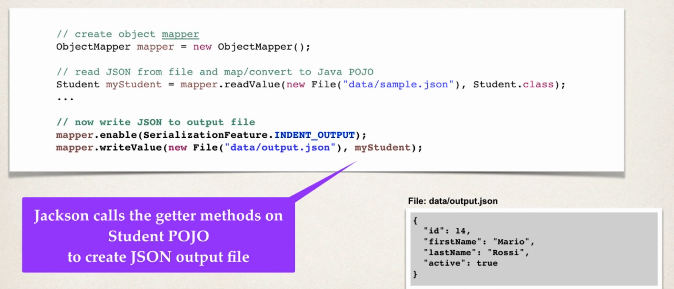
Note: Jackson reads the JSON object key and makes a call to setter method for that in POJO.

Here in the above example for id Jackson will call setID() method.



**Java POJO to JSON data**

Jackson calls the getter methods of Java POJO to convert it to JSON data format.



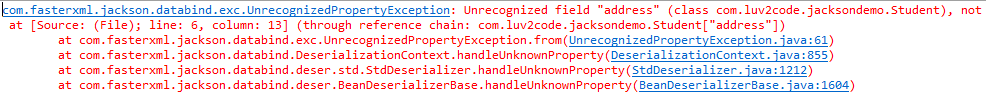
**Spring and Jackson Support**

* Spring will automatically handle Jackson integration
* JSON data being sent to Rest Controller will be automatically converted into Java object
* Java object being returned by the Rest Controller will be automatically converted into JSON data.

**Jackson for Nested JSON and Array**

Jackson handling more complex JSONs like nested JSON and Array in JSON will be covered here.

If a JSON object gets new properties in it but our Java POJO code is same without adding new fields and getters/setters methods for that then we will have a run time exception.

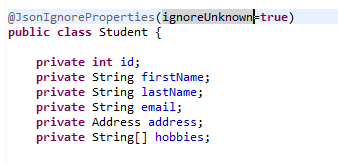


Sine we just added address and hobbies properties to the existing JSON object and we don’t have them defined in our existing Java POJO, hence we have this compilation error.

Now in order to address this issue, we can create a new class called Address give all the fields and define setter/getter methods. Finally in Student class add Address and hobbies fields and generate setter/getter methods. Now Jackson will be able to handle databinding without issue.

**Ignore Properties:**

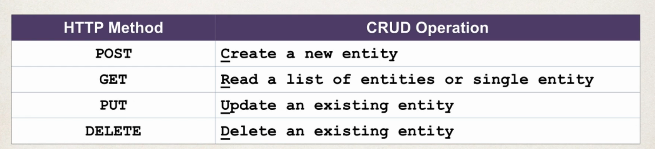
If JSON has a property that you don’t care, wouldn’t it be great to just ignore it?



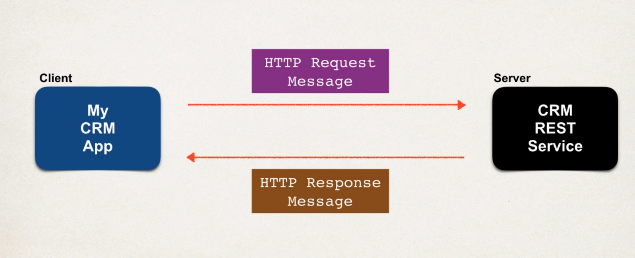
Now if a new property is added to the existing JSON data, then our app will not crash since we added @JsonIgnoreProperties(ignoreUnknown=true) annotation to ignore the properties that not defined in the Java POJO but are present in JSON Object.

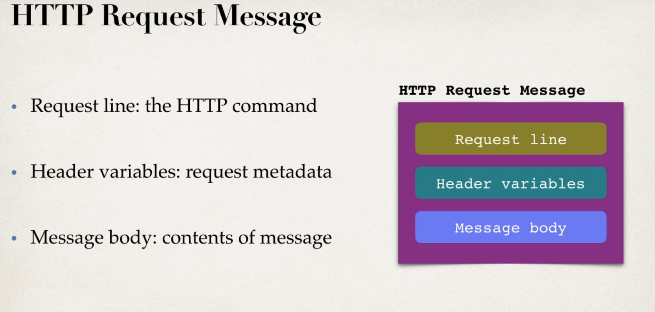
**REST over HTTP**

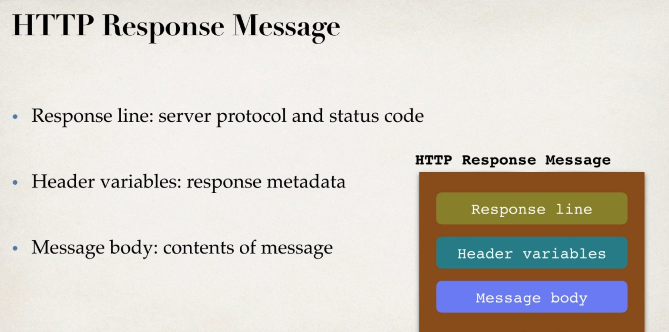
Most commonly REST uses HTTP protocol for its communication. Since HTTP is being used REST will have leverage to HTTP methods like GET, PUT, POST, DELETE etc

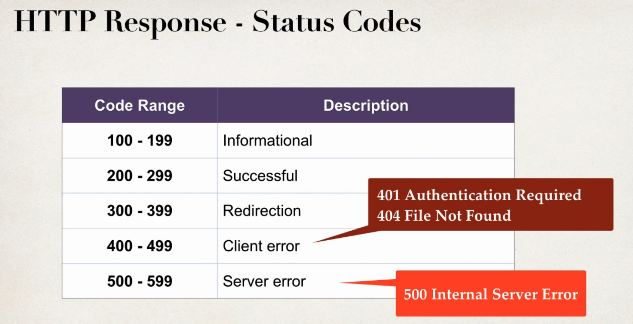


**HTTP Messages**









**MIME Content Types**

The message format is described by MIME Content Type

Multipurpose Internet Mail Extension

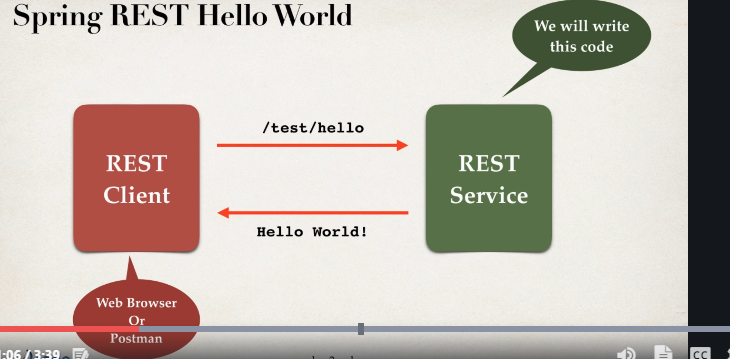
Basic syntax: type/subtype

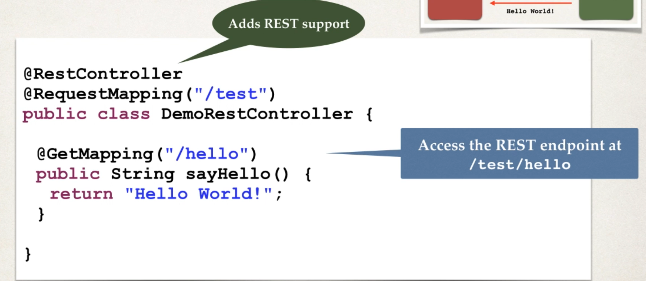
Examples: text/plain, text/html, application/json, application/xml

**Spring REST Controller**

* Spring Web MVC provides support for Spring REST
* It gives us this new Annotation called **@RestController** and its basically an extension of @Controller but @RestController has support for REST requests and responses.
* Spring REST can map JSON object to Java POJO and vice versa as long as Jackson-databind dependency is in the pom.xml.

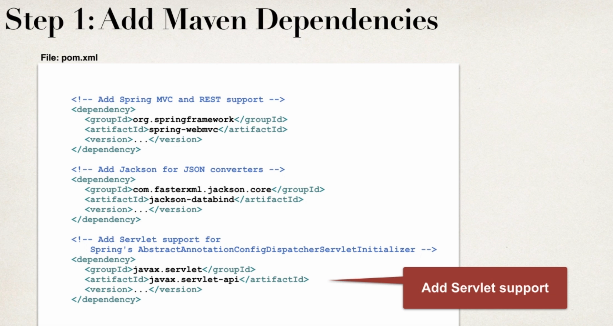
Spring REST Hello World example:





In order to build Spring Rest webservices we need to follow the following steps

* Add the maven dependencies for Spring Web MVC and Jackson project
* Add code of the all Java config : @Configuration
* Add code for all Java config: Servlet Initializer
* Create Spring REST service using the @RestController





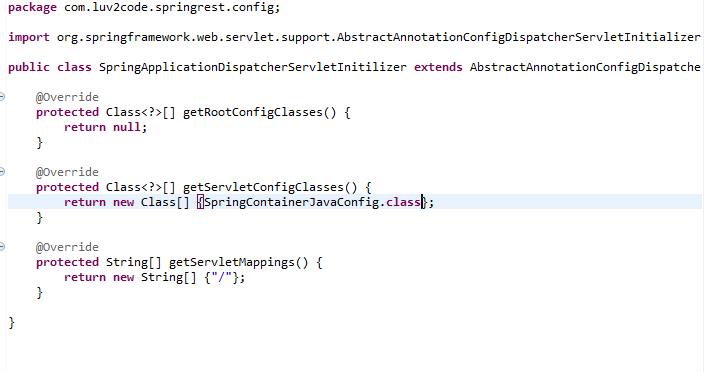
Web application initializer

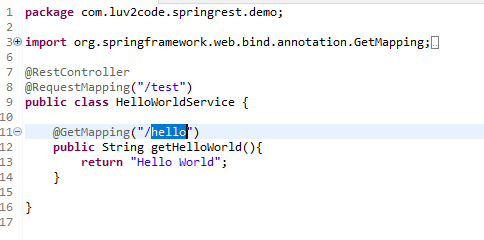
* Spring MVC provides support for web app initialization
* Make sure our code is automatically detected.
* Your code is used to initialize the servlet container.

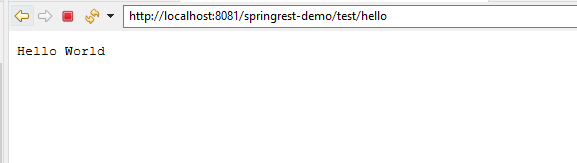
AbstractAnnotationConfigDispatcherServletInitializer

Steps for this

* Extend this Abstract class
* Override unimplemented methods
* Specify servlet mapping and location of your app config.

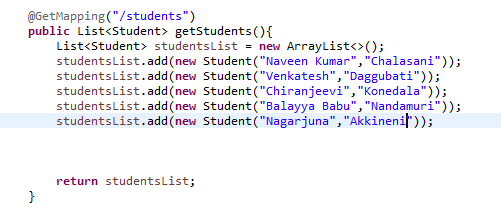






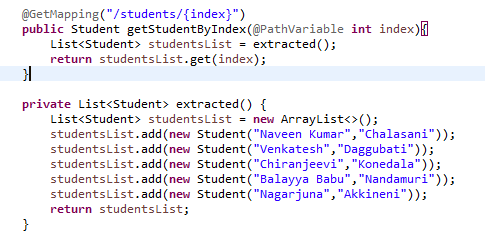
**Retrieve POJO as JSON**

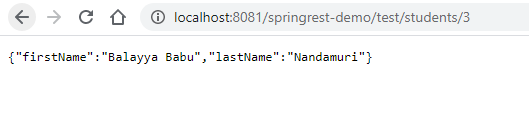
Spring and Jackson will take care of this conversion.



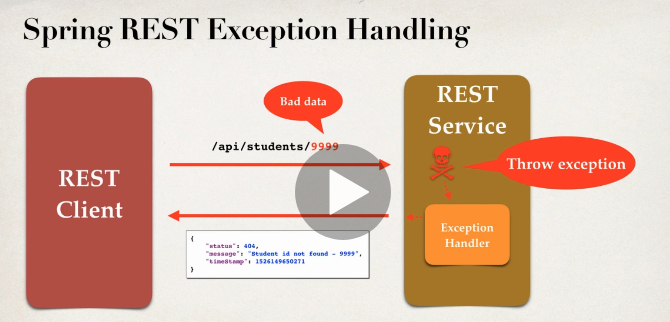
**@PathVariable**

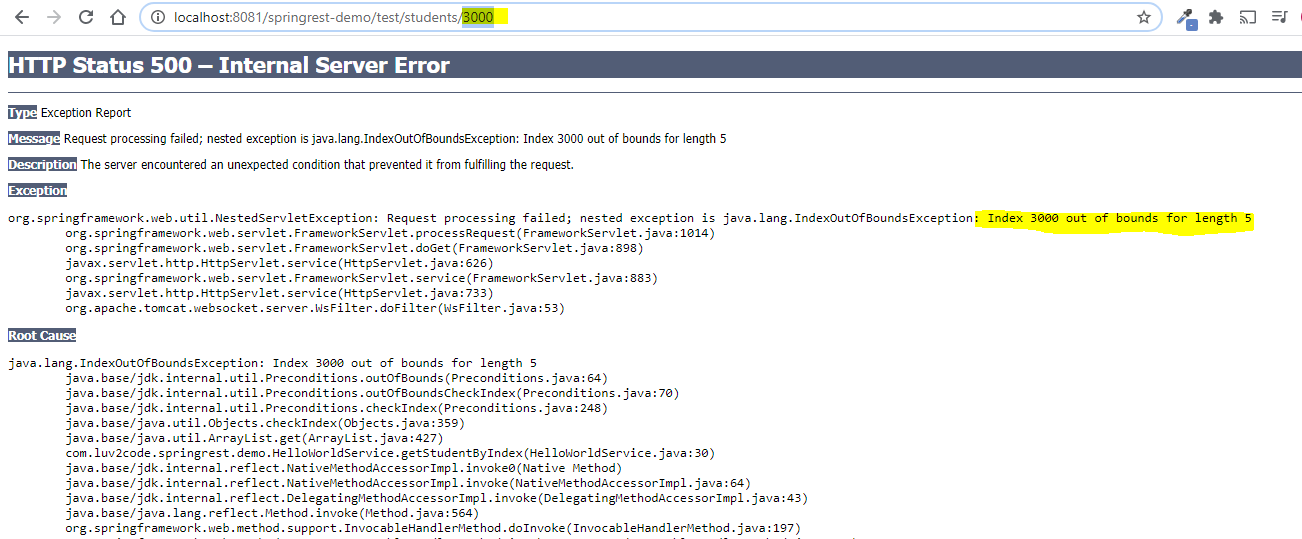






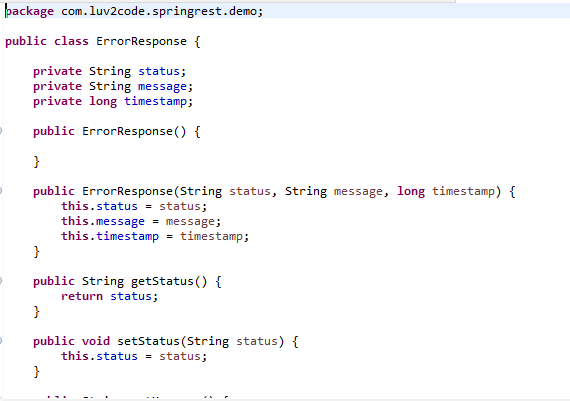
**Exception Handling:**



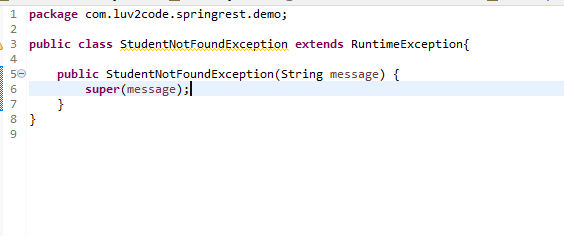


**Development process**

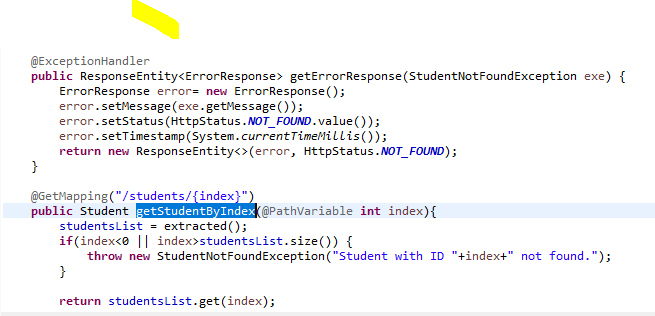
1. Create a custom error response class



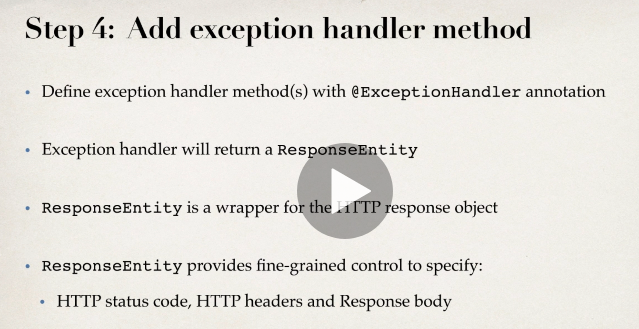
1. Create custom exception



1. Update REST Service to throw exception if student not found



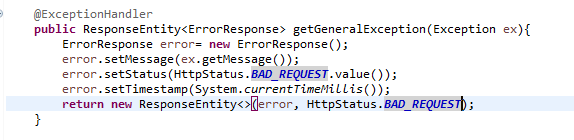
1. Add an exception handler method using @ExceptionHandler



What if instead of number, a string is passed, it will break again

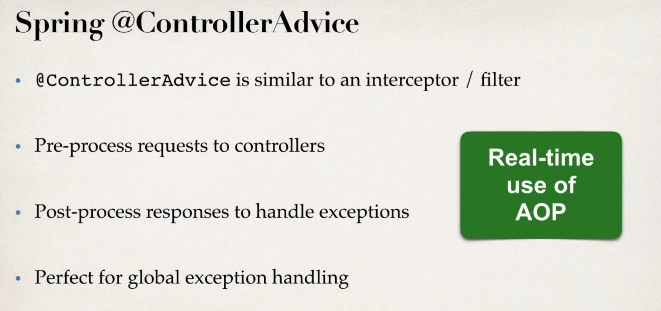


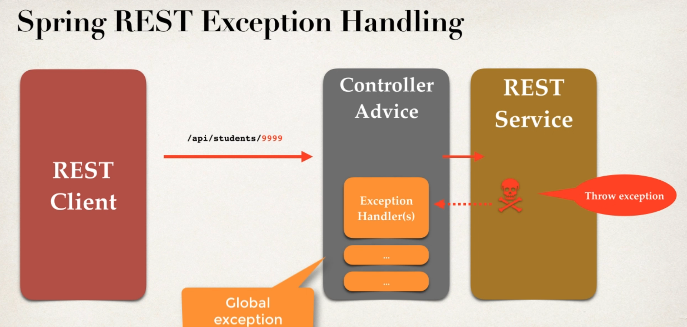
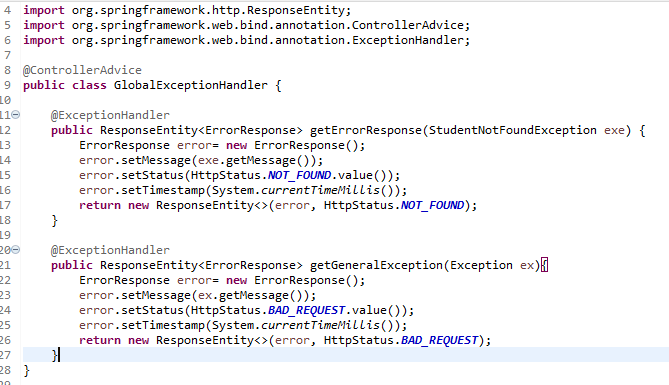
Add another exception handler method to catch all any type of exception thrown



**Global Exception Handling**

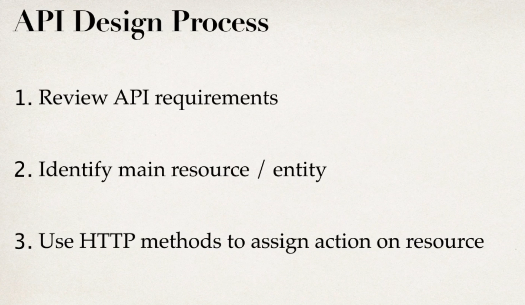
The above process works completely fine but these exception handlers are only available to that particular rest controller not to the others and it can’t be reused by other rest controllers. So we need Global exception handling to promote reuse and centralized exception handling.

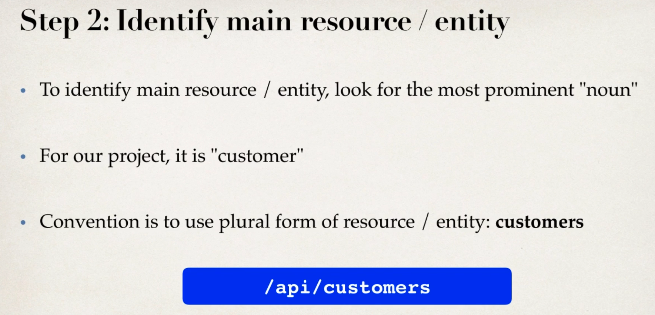
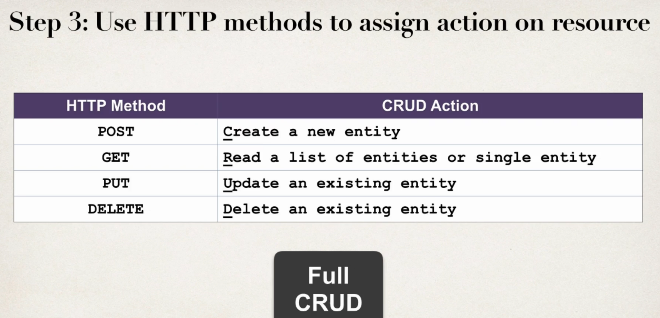
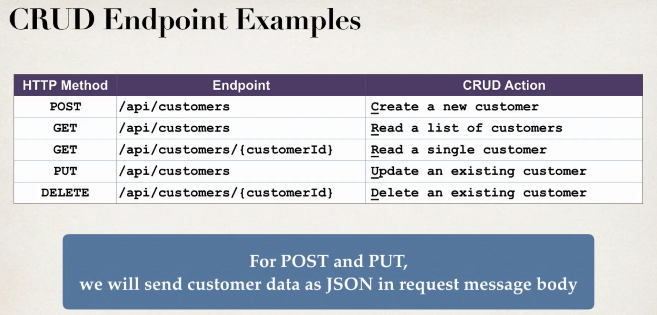


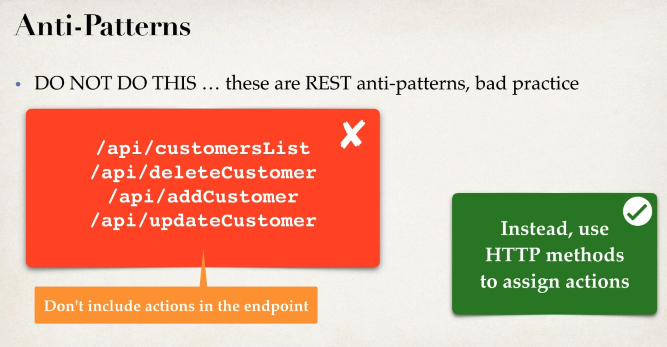
 

**Spring rest api design –best practises**

* For real time projects we need to know who will use this api?
* How will they use this api?
* Finally based on the requirement, design the api.



**More examples**

