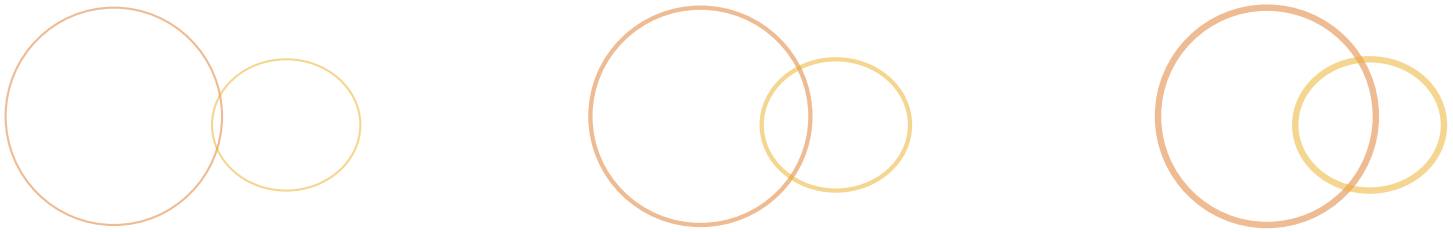


RESTful Web Services

Objectives

When we are done, you should be able to:

- Explain what a RESTful web service is
- Understand how to come up with appropriate paths
- Use View Resolvers

REpresentational State Transfer

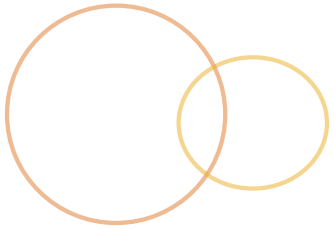
What is REST?

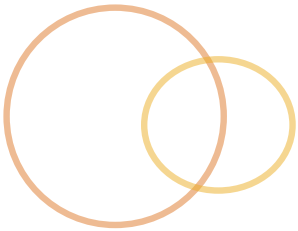
- An architecture
- It is a mass silently agreed to process
  - It is not actually a standard
- It is all about dynamically revealing data
- REST is about sending resources
- In Spring
  - Available with Spring 3.0
  - Not an implementation of JAX-RS

Modeling REST

- Determine what is being exposed
  - Not everything in your application should be available. What should be?
- Design your URIs
  - Remember the `@RequestMapping`? That is how you make it available
- Add operations
  - What HTTP method should be available for each
- Change your controller to be `@RestController`
  - Makes some things easier
  - Opens up availability for REST specific annotations

Exposure





- Rest can send different data types besides just Java objects or XML
  - Spring give us classes for translating to and from JSON
  - There are others out there
- `@RequestMapping` has
  - `consumes` – what data types can this method understand
  - `produces` – what data types is this method able to send as a response
  - `headers` – modify HTTP headers, which could also tell us what kind of data type we want to send
  - `method` – what specific HTTP method will we respond to

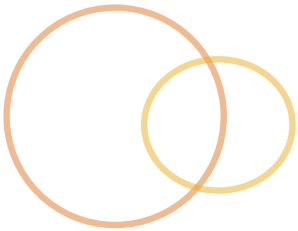
Defining Applications Used

```
@RequestMapping(consumes={"application/XML",
"application/JSON"})

@RequestMapping(headers="Accept=application/JSON")

@RequestMapping(headers="Content-type=application/JSON")
```

Operations



- These are the methods that define how we receive requests
- If method is used, then request must match expected method
- If no method defined, request can be any of them
- Can take multiple operations

```
@RequestMapping(value="/book/{id}", method=RequestMethod.GET)
```

Defining URIs

- URIs are not supposed to tell you what method or class that they are calling
- Supposed to 'dynamically reveal' data
- Mostly use attribute 'method' to determine what is happening
  - Often use one `@RequestMapping` on entire class that has the value

```
@RequestMapping(method=RequestMethod.GET)
```

Methods Tell Us Something

- Use the appropriate HTTP method for what the purpose of the method is

GET	Retrieves; performs a 'select'
POST	Create new; No primary key
PUT	Updates; Has a primary key
DELETE	Remove
HEAD and OPTIONS	Two different methods to retrieve meta-data

Handling Response Codes

- General best practice
  - REST presumes no visual client
  - Client needs to know what to expect to determine what to do next when there isn't a human on the client side

```
@RequestMapping(method=RequestMethod.GET)
@ResponseStatus(HttpStatus.OK)
```

Methods Have Specific HTTP Response Codes

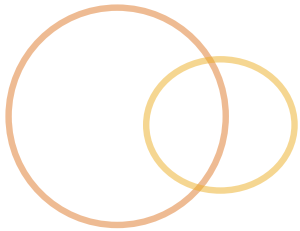
- You may want to catch HTTP status codes in order to know how to proceed

GET	200 – OK
POST	201 – Created
PUT	201 – Created; 404 – Not Found; 406 Not Acceptable (format)
DELETE	200 – OK; 404 – Not Found

View Resolvers

- Determines what format responses are sent in
- RESTful services are more likely to return the object
  - We usually use a `ContentNegotiationConfigurer`
    - Tells us what media types to associate with what type of extension
    - Goes in the `MvcConfig` file

MvcConfig.xml



```
@Configuration
@ComponentScan(basePckages="com.di.phonebook")
@EnableWebMvc
public class MvcConfig extends WebMvcConfigurerAdapter {

  public void configureContentNegotiation
    (ContentNegotiationConfigurer configurer) {
      configure.ignoreAcceptHeader(false).
      favorPathExtension(true).
      defaultContentType(MediaType.APPLICATION_XML).
      mediaTypes(new HashMap<String, MediaType>(){
        {
          put("xml", MediaType.APPLICATION_XML);
          put("json", MediaType.APPLICATION_JSON);
        }
      });
  }
}
```

String Formats

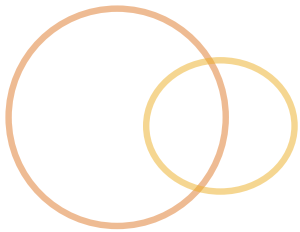
- Generally use either XML or JSON
  - For XML, we need to make the entity class JAXB compatible by using `@XmlRootElement` on the class
  - For JSON compatibility, we need the Jackson APIs (specifically the jackson-databind api)
    - By using `@RestController`, we just need to have the databind api available and it will take care of reading to and from JSON without anything else needed on our part

JAXB (Java API for XML Binding)

- API that allows Java objects to be marshalled (converted from Java to XML) and unmarshalled (converted from XML to Java)
- Can write your own marshalling/unmarshalling, but not necessary
  - We have annotations that tell the entity how to convert

```
@XMLRootElement
public class Address {
  private String street;
  private String city;
  ...
}
```

@RequestBody and @ResponseBody



- `@RequestBody` – defines that the body of the HTTP packet will contain the 'object' being passed into the method
- `@ResponseBody` – defines that the response will be in the body of the HTTP packet
- Both of these are used primarily in REST, and use their mappings to define the format (XML, JSON, etc.)

```
@RequestMapping(method=RequestMethod.GET)
@ResponseBody
public MapCoordinates getLocation(@RequestBody Address address)
{ ... }
```

Lab 6 – RESTful Web Applications