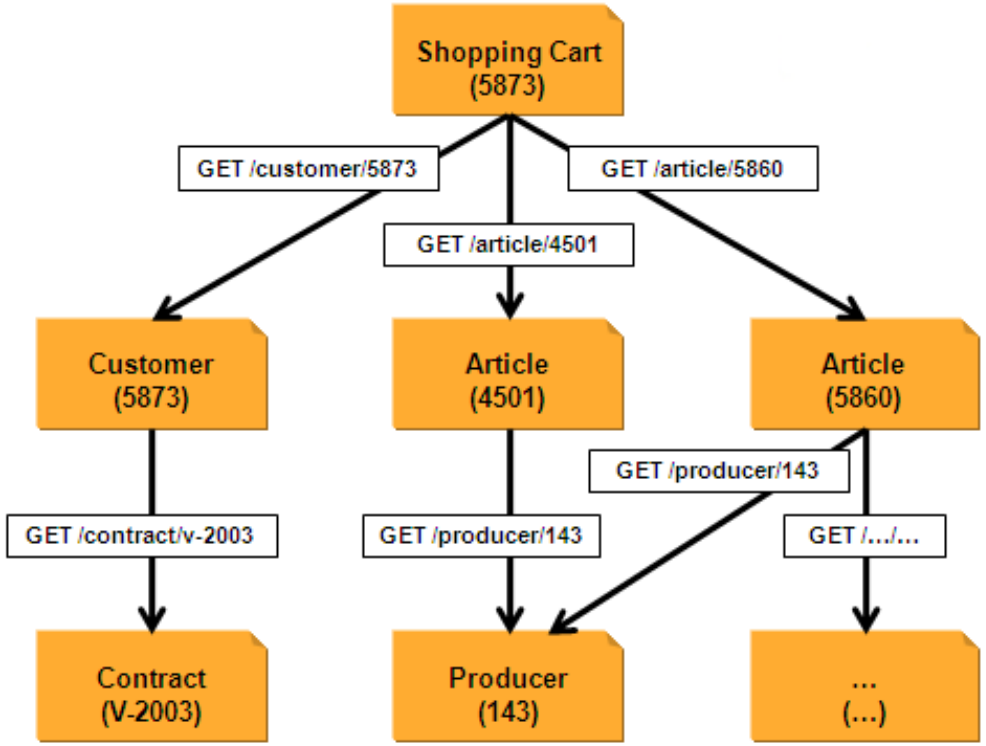
SPRING WEB SERVICES

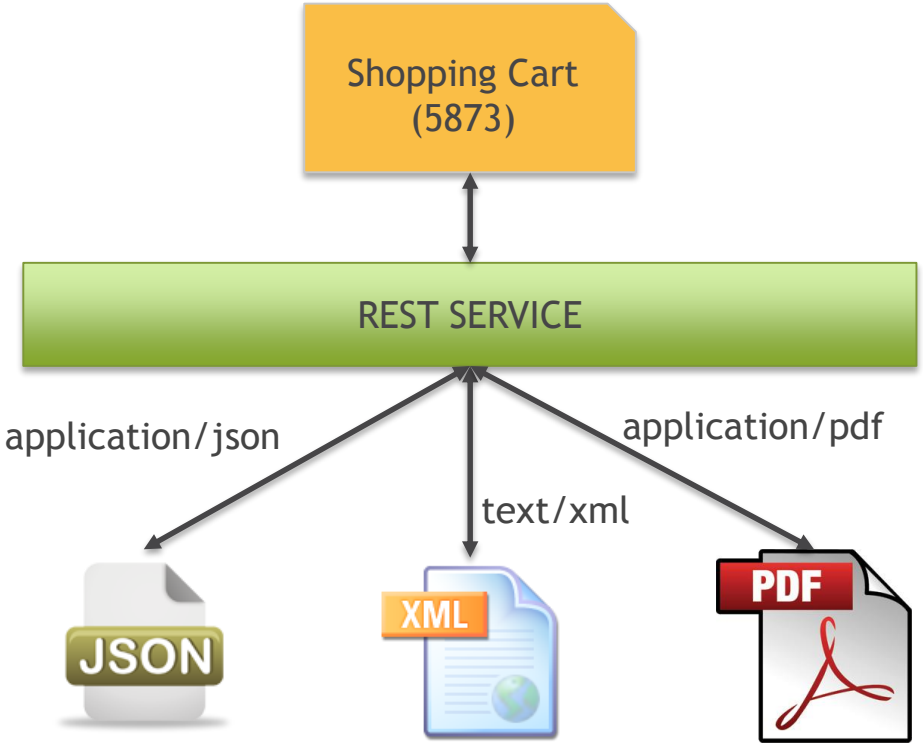
RESTFUL WEB SERVICES

MARCH, 2015

IDENTIFIABLE RESOURCES



RESOURCE REPRESENTATIONS



STANDARD METHODS

- GET Safe, nullipotent, cacheable <- Retrieve
- PUT Idempotent <- Update
- DELETE Idempotent <- Delete
- POST None of the above <- Create

Nullipotent: a method has no side effect; it doesn't change the data.

Idempotent: regardless of how many times the method is invoked, the end result is the same.

STATELESS INTERACTION

Statelessness by Roy Fielding:

- Visibility
  - Every request contains all context necessary to understand it. Therefore, looking at a single request is sufficient to visualize the interaction.
- Reliability
  - Since a request stands on its own, failure of one request does not influence others.
- Scalability
  - The server does not have to remember the application state, enabling it to serve more requests in a shorter amount of time.

SPRING MVC FOR RESTful SERVICES

SAMPLE REST CONTROLLER

```
@Controller
@RequestMapping("/persons")
public class PersonController {
    @Autowired
    private PersonRepository personRepository;

    @ResponseBody
    @RequestMapping(value = "/{id}“, method=RequestMethod.GET)
    public Person findPerson(@PathVariable(“id") String id){
        return this.personRepository.findById(id);
    }

    @ResponseBody
    @RequestMapping(method = RequestMethod.POST)
    public Person addPerson(@RequestBody final Person model, HttpServletRequest request,
                            HttpServletResponse response){
        Person person =  this.personRepository.save(model);
        if(null != person){
            response.setStatus(201);
        }
        return person;
    }
}
```

SPRING REST CONFIGURATION

```
<mvc:annotation-driven>
    <mvc:message-converters registerdefaults="false">
      <bean
         class="o.s.http.converters.json.MappingJacksonHttpMessageConverter"/>
      <bean
        class="o.s.http.converters.xml.Jaxb2RootElementHttpMessageConverter"/>
     </mvc:message-converters>
</mvc:annotation-driven>
```

HttpMessageConverter

```
public interface HttpMessageConverter<T> {
    boolean canRead(Class<?> clazz, MediaType type);

    boolean canWrite(Class<?> clazz, MediaType type);

    List<MediaType> getSupportedMediaTypes();

    T read(Class<? extends T> clazz, HttpInputMessage msg)
                                        throws IOException,
                                               HttpMessageNotReadableException;

    void write(T value, MediaType type, HttpOutputMessage msg)
                                        throws IOException,
                                               HttpMessageNotWritableException;
}
```

- reads the request body and writes the response
- converters mapped to content types
- registered by default if jar present in classpath
  - Jackson, JAXB, Atom, RSS

ContentNegotiatingViewResolver

```
<bean class="o.s.web.servlet.view.ContentNegotiatingViewResolver">
    <property name="order" value="1"/>
    <property name="mediaTypes">
        <map>
            <entry key="atom" value="application/atom+xml"/>
            <entry key="html" value="text/html"/>
            <entry key="json" value="application/json"/>
        </map>
    </property>
    <property name="defaultViews">
        <list>
            <bean class="o.s.web.servlet.view.json.MappingJacksonJsonView"/>
        </list>
    </property>
</bean>

<bean class="o.s.web.servlet.view.InternalResourceViewResolver">
    <property name="order" value="2" />
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

CONTENT NEGOTIATING STRATEGIES

- URI extension
  - www.domain.com/persons**.html**
  - www.domain.com/cars**.json**
- Request header
  - Accept: text/html
  - Accept: text/xml
  - Accept: application/pdf

ACCESING REST ON THE CLIENT

```
RestTemplate restTemplate = new RestTemplate();
Map<String, String> vars = new HashMap<String, String>();
vars.put("hotel", "42");
vars.put("booking", "21");
String result =
restTemplate.getForObject("http://example.com/hotels/{hotel}/bookings/{booking}",
                          String.class, vars);
```

HTTP	REST TEMPLATE
DELETE	delete(String, String…)
GET	getForObject(String, Class, String...)
HEAD	headForHeaders(String, String...)
OPTIONS	optionsForAllow(String, String...)
POST	postForLocation(String, Object, String...)
PUT	put(String, Object, String...)

<epam>

THANK YOU!

MAKSYM_GOVORISCHEV@EPAM.COM

MARCH, 2015