



Qu'est ce qu'un service web ?

Service web = instance d'objet, ressource déployée sur Internet
 Permettent à des applications (web, mobile, etc.) de faire appel à des fonctionnalités (via des objets) sur un réseau local ou Internet

Evolution des systèmes distribués, architecture SOA (architecture orientée service)
 Technologie initiée par IBM et microsoft, puis normalisée par le W3C

Un service web = composant développé dans n'importe quel langage sur n'importe quelle plateforme.
 Il doit pouvoir être invoqué par n'importe quel autre service

Rep: S. SADIA 2

Pourquoi les web services ?

Utilisation d'HTTP:

- Protocole Internet
- Beaucoup d'entreprises possèdent un serveur web
- Protocole généralement autorisé au niveau de parefeu
- Protocole disponible sur toutes les plateformes
- Mode non connecté

Requêtes/réponses des services doivent être sérialisées pour réutilisation
 XML
 JSON
 autre, attention (pas recommandé)

Rep: S. SADIA 3

RESTful Web Services

- * Appel de ressources (Web) via une URL contenant un nom de méthode et des paramètres
- * URL peut avoir une grammaire complexe
- * Utilisation d'une méthode HTTP qui décrit de façon succincte la sémantique de l'appel
- * Ex:

Rep: S. SAÏDA 4

RESTful Web Services

- * Utiliser les bonnes méthodes HTTP:
- * POST ajouter
- * PUT modifier
- * GET lire
- * DELETE

Rep: S. SAÏDA 5

RESTful Web Services

- * **HATEOAS, Hypermedia as the Engine of Application State,**
- * Contrainte sur SW REST telle que:
- * Un client peut interagir complètement avec un SW => à tout moment il doit connaître les possibilités offertes par le services c.a.d. les ressources qu'il peut appeler

Rep: S. SAÏDA 6

RESTful Web Services

GET /account/12345 HTTP/1.1
Host: somebank.org
Accept: application/xml
...

HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
 <account_number>12345</account_number>
 <balance currency="usd">100.00</balance>
 <link rel="deposit" href="/account/12345/deposit"/>
 <link rel="withdraw" href="/account/12345/withdraw"/>
 <link rel="transfer" href="/account/12345/transfer"/>
 <link rel="close" href="/account/12345/close"/>
</account>

Plus tard...

HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
 <account_number>12345</account_number>
 <balance currency="usd">25.00</balance>
 <link rel="deposit" href="/account/12345/deposit"/>
 <link rel="transfer" href="/account/12345/transfer"/>
 <link rel="close" href="/account/12345/close"/>
</account>

Req: S. SAOJA 7

Introduction aux Services Web REST

- Implémentation d'une API en Java = **JAX-RS** (Java API for RESTful Web Services) qui est intégrée à Java EE 6
- Décrite par les spécifications JSR 311 et JSR 339
- Plusieurs frameworks basés sur JAX-RS
 - : RESTEasy, CXF, Jersey (omade)
 - Jersey 2 (mai 2013) choisi dans ce cours (<http://jersey.java.net/>)
- Possibilité d'utiliser Axis 2 (Apache)
 - <http://axis.apache.org/axis2/java/core/docs/rest-ws.html>
- Spring boot

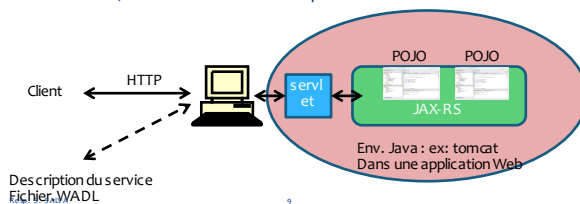


Req: S. SAOJA

8

Architecture

- * Jersey => Développement basé sur l'implantation de POJO, annotés et sur des requêtes HTTP



Services Web REST avec Jersey

```

@Path("/generic")
public class GenericResource {

    @GET
    @Produces("text/html")
    public String getResp() {

        return "<html><body>test<br>coucou\n</body></html>";
        //TODO return proper representation object
        // throw new UnsupportedOperationException();
    }
  
```

Annotations and their meanings:

- `@Path("/generic")`: Chemin du service vu comme une ressource
- `@GET`: Type de requête HTTP
- `@Produces("text/html")`: Type MIME de retour
- `throw new UnsupportedOperationException();`: Exception éventuelle

Rep: S. SAÏDA 10

Déploiement et appel

- Déploiement en déclarant le service comme une Servlet (dans web.xml pour tomcat/Glassfish) (voir cours Servlet)
 - ```

<servlet>
<servlet-name>EssaisServlet</servlet-name>
<servlet-class>bdtest.EssaisServlet</servlet-class>
</servlet>

```
  - ```

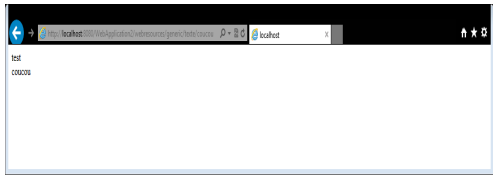
<servlet-mapping>
<servlet-name>EssaisServlet</servlet-name>
<url-pattern>/EssaisServlet</url-pattern>
</servlet-mapping>

```
- => les IDE tels que Netbeans ou Eclipse le font pour vous

Rep: S. SAÏDA 11

Déploiement et appel

* Appel avec méthode GET:



Rep: S. SAÏDA 12

Les annotations

- * Choix de la méthode HTTP
- * Définition des paramètres
- * Définition des types de retour
- * Sérialisation
- * Définis par des annotations dans le code

Rep: S. SAÏDA

13

Les annotations

- * Annotation `@path`:
 - * Classe java représentant un service REST doit être annotée par `@path` => définit une ressource Racine (root)
 - * Path décrit une expression URI permettant d'appeler le service
 - * Ex:
<http://localhost:8080/WebApplication2/webresources/generieric>

Rep: S. SAÏDA

Contexte de l'application Web Contexte service

Les annotations

- * Sub ressource `@Path`
 - * Possibilité d'annoter les méthodes de la classe également
 - * url final = concaténation uri classe + uri méthode

```
@GET @Path("appel")
@Produces("text/html")
public String getResp() {
```

- * <http://localhost:8080/WebApplication2/webresources/generieric/appe>

Rep: S. SAÏDA

15

Les annotations

- * @Path peut également être complexe et donner des paramètres (appelées Template Parameters)
- * Distinction d'une expression par les balises {}

```
@GET @Path("appel/{texte}")
@Produces("text/html")
public String getResp(@PathParam("texte") String t) {
```

- * <http://localhost:8080/WebApplication2/webresources/generic/appe/coucou>

Rep: S. SAÏVA

16

Les annotations

- * Code de méthode :

```
@GET
@Path("appel/{texte}")
@Produces("text/html")
public String getObj(@PathParam("texte") String texte) {
    return "<html><body>test<br>" + texte + "</body></html>";
    //TODO return proper representation object
    // throw new UnsupportedOperationException();
}
```

- * Appel:
<http://localhost:8080/WebApplication2/webresources/generic/appe/coucou>

La zone texte

Rep: S. SAÏVA

17

Exemple de méthode GET avec retour text/HTML

- * Code de méthode :

```
@GET
@Path("appel/{texte}/{detail}/{det}")
@Produces("text/html")
public String getObj(@PathParam("texte") String t,
    @PathParam("det") String d) {
    return "<html><body>" + t + "<br>" + d + "</body></html>";
    //TODO return proper representation object
    // throw new UnsupportedOperationException();
}
```

- * Appel:
<http://localhost:8080/WebApplication2/webresources/generic/appe/text-coucou-detail-texte-court>

La zone texte

Rep: S. SAÏVA

18

@HEAD, @Get, @Post, @Put, @Delete

- * Méthodes du services doivent être annotées par une méthode http pour traitement
- * Avec JAX-RS, il est possible d'utiliser @GET pour supprimer une ressource mais
 - * Ne respecte pas REST
 - * Manque de sens?
 - * Généralement description de méthodes CRUD (create, read, update, delete)
 - * @POST -> création ressource
 - * @GET -> lecture
 - * @PUT -> mise à jour
 - * @DELETE -> suppression de la ressource

Rep: S. SAÏDA

19

Annotation supplémentaires pour les méthode

- * D'autres annotations pour gérer les paramètres des méthodes
- * Possibilité de faire des annotations sur des variables de types primitifs (sauf char), toute classe ayant un constructeur composé d'un paramètre String, classes ayant méthode statique valueOf(String)
- * Possibilité de placer une valeur par défaut avec @DefaultValue

Rep: S. SAÏDA

20

Annotation supplémentaires pour les méthode

- * @PathParam -> extraire valeur dans URL sur le template parameters
- * @QueryParam -> extraire sur l'url par requête
- * @FormParam: extraction depuis formulaires HTML
- * @CookieParam: extraction depuis un paramètre de cookie
- * @HeaderParam: extraction de données de l'entête HTTP
- * @Context: permet de récupérer contenu entête et cookies en même temps

Rep: S. SAÏDA

21

Codages des données envoyées et produites: @Consumes, @Produces

- * @Consumes utilisé pour spécifier le ou les types MIME qu'une méthode peut accepter
- * @Produces donne le ou les types MIME qu'une méthode peut produire
- * Annotations peuvent porter sur classes ou sur méthodes
- * Attention: si rien n'est défini, tous les types MIME pourront être acceptés ou produits

Rep: S. SAÏDA

22

@Produces

```
@GET
@Path("/appel")
@Produces("text/html")
public String getHTMLob(@DefaultValue("texte default ")
@QueryParam("texte") String texte {

    return "<html><body>text<br>" + texte + "\n</body></html>";
}

@Path("/appel")
@Produces("text/xml")
public String getXMLob(@DefaultValue("texte default ")
@QueryParam("texte") String texte {

    return "<?xml version='1.0'?>" + "<contenu>livre vide</contenu>";
}
```

Rep: S. SAÏDA

23

@Produces

Mieux:

```
@Path("/appel")
@GET
@Produces("application/xml")
public Response getXml(@DefaultValue("texte default ")
@QueryParam("texte") String texte) {

    Response response =
    Response.status(200).type(MediaType.TEXT_XML).entity("<?
xml version='1.0'?
>" + "<contenu>" + texte + "</contenu>").build();
    return response;
}
```

Rep: S. SAÏDA

24

Manipulation de types personnalisés

- * Type String: voir exemples précédents
- * Egalement File etc.
- * Type personnalisé: possibilité d'utiliser des types personnalisés définis par un schéma XML
 - * Besoin d'un mapping entre XML et objet
 - * 1. définition d'une classe avec annotations de type JAXB @XmlElement, @XmlType
 - * 2. format du contenu de la requête au service en XML ou JSON
 - * 3. Les contenu des requêtes sont définis par les annotations @Produces et @Consumes (text/xml, application/xml, application/json et autres)

Rep: S. SAÏDA

25

Manipulation de types personnalisés

Exemple Simple:

- * 1. Définition de la classe POJO

@XmlElement(name = "texte")

Public Class Texte {

```

    Protected String contenu;
    Public String getContenu(){ return contenu;}
    Public void setContenu(String s){this.contenu=s;}
    Public String toString(){ return contenu;}
  }
```

Rep: S. SAÏDA

26

Utilisation d'un type personnalisé

```

@PUT
@Path("/update")
@Consumes("application/xml")
public void uptexte(TeXte t) {
    System.out.println(t.getContenu());
}

@GET
@Path("/appel")
@Produces("application/xml")
public Texte gettexte() {
    Texte t=new Texte();
    t.setContenu("blabla");
    return t;
}

* Ou JSON : @Produces("application/json")
```

Rep: S. SAÏDA

27

Utilisation d'un type personnalisé

Un autre exemple de ressource:

```
@XmlRootElement(name="customer")
@XmlAccessorType(XmlAccessType.FIELD)
//indique que tous les champs non statiques de la classe sont pris en compte
public class Customer {

    @XmlAttribute(required=true) protected int id;
    //attribut reste dans l'elt. XML en cours
    @XmlElement(required=true) protected String firstname;
    @XmlElement(required=true) protected String lastname;
    @XmlElement(required=true) protected Address address;
    @XmlElement(required=true) protected String email;
    @XmlElement(required=true) protected String phone; public
    Customer() { } // Getter and setter methods // ... }
```

Rep: S. SAÏDA

28

Gestion du status HTTP

- * Code retour HTML retourné lorsqu'un service REST est appelé:
- * Réponse sans erreur
 - * Code retour 200 avec contenu
 - * Code retour 204 sans contenu
- * Réponse avec erreur
 - * De 400 à 599
 - * Ex: 404 not found
 - * Ex: 405, method not allowed

Rep: S. SAÏDA

29

Gestion du status HTTP

- * Possibilité de construire des réponses plus précises et de modifier les codes
- * Choisir un code retour
- * Fournir des paramètres dans l'entête
- * Besoin d'utiliser le patron builder
 - * Plusieurs méthode retournant ResponseBuilder

Rep: S. SAÏDA

30

Gestion du status HTTP

- * Exemples:
- * `ResponseBuilder ok()` => statut 200
- * `ResponseBuilder serverError()` => statut >400
- * `ResponseBuilder status(Response.Status);` définit un statut précis
 - * `Status: Responses.NOT_FOUND`
 - * `Response.Status.OK`, ou chiffre

Rep: S. SAÏDA

3

Gestion du status HTTP

- * Méthodes de la classe `Builder` utiles:
- * `Response build()`: crée une instance
- * `ResponseBuilder entity(Object o)`: modifie contenu du corps
- * `ResponseBuilder header(String, Object)`: modifie un paramètre de l'entête HTTP

Rep: S. SAÏDA

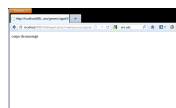
3

Gestion du status HTTP

```
@Path("/appel")
public class ExempleResponse {
    @Path("/response")
    @GET
    public Response getText() {
        return Response
            .status(Response.Status.OK)
            .header("param", "valeur")
            .entity("corps du message")
            .build();
    }

    @GET
    public Response getError() {
        return Response
            .status(Response.Status.INTERNAL_SERVER_ERROR)
            .entity("Erreur serveur")
            .build();
    }
}
```

Retour code 200



Retour code 500



Rep: S. SAÏDA

3

Gestion des exceptions coté Serveur

- Exception de base dans Jersey transformée en status HTTP: **WebApplicationException**
- Plusieurs constructeurs (voir doc)

```
@GET
@Path("/images/{image}")
@Produces("image/*")
public Response getImage(@PathParam("image") String image) {
    File f = new File(image);

    if (!f.exists()) {
        throw new WebApplicationException(404);
    }

    String mt = new MimeTypeFileTypeMap().getContentType(f);
    return Response.ok(f, mt).build();
}
```

Rep: S. SADIA

36

Gestion des exceptions coté Serveur

- Classe fille **NotFoundException** avec HTTP 404 (Not Found) status

Exemple: `throw new NotFoundException("Item, " + itemid + ", is notfound");`

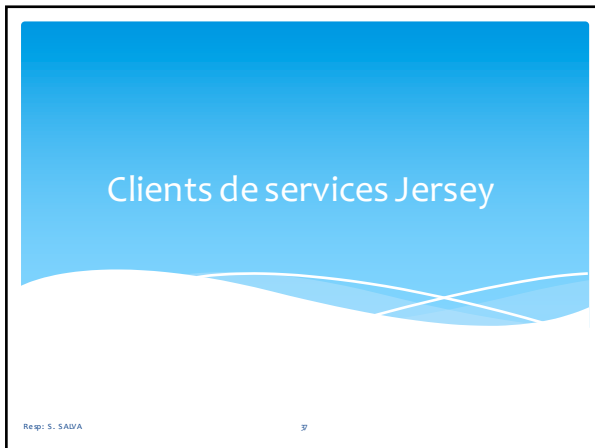
Rep: S. SADIA

37

Gestion des exceptions coté Serveur

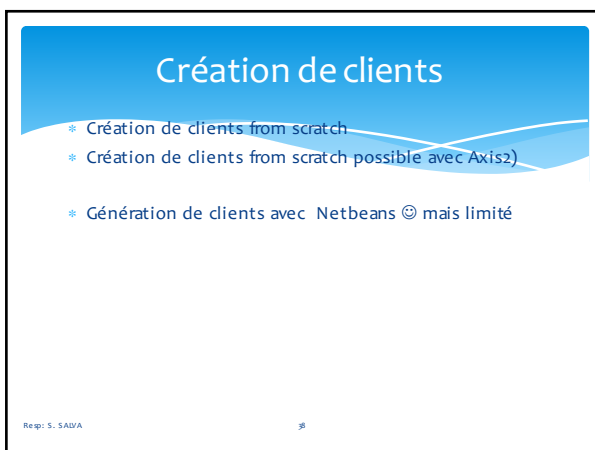
```
1 public class NotFoundException extends WebApplicationException {
2
3     /**
4      * Create a HTTP 404 (Not Found) exception.
5      */
6     public NotFoundException() {
7         super(Responses.NotFound().build());
8     }
9
10    /**
11     * Create a HTTP 404 (Not Found) exception.
12     * @param message the String that is the entity of the 404 response.
13     */
14    public NotFoundException(String message) {
15        super(Responses.NotFound().entity(message).type("text/plain").build());
16    }
17 }
18
19 Rep: S. SADIA
```

38



Clients de services Jersey

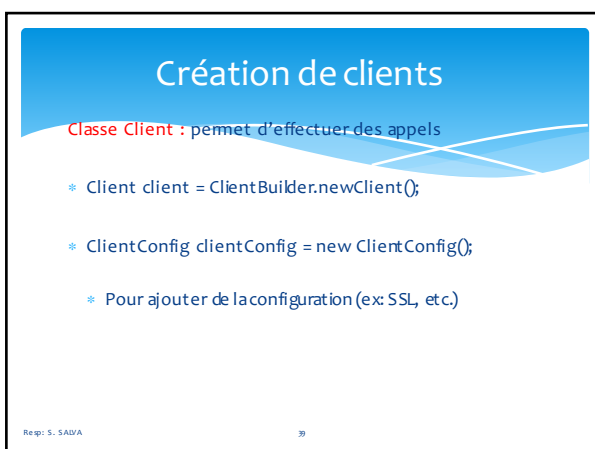
Rep: S. SADA 27



Création de clients

- * Création de clients from scratch
- * Création de clients from scratch possible avec Axis2)
- * Génération de clients avec Netbeans © mais limité

Rep: S. SADA 28



Création de clients

Classe Client : permet d'effectuer des appels

- * `Client client = ClientBuilder.newClient();`
- * `ClientConfig clientConfig = new ClientConfig();`
- * Pour ajouter de la configuration (ex: SSL, etc.)

Rep: S. SADA 29

Création de clients

- * Classe WebTarget
- * Cibler la ressource HTTP
- * Identifier la ressource
- * Donner des paramètres

```
1. WebTarget webTarget =
  client.target("http://example.com/rest");

2. WebTarget helloworldWebTarget =
  resourceWebTarget.path("helloworld");

3. WebTarget helloworldWebTargetWithQueryParam =
  helloworldWebTarget.queryParam("greeting", "Hi
  World!");
```

Rep: S. SAÏDA

40

Création de clients

- * Classe Invocation
- * Pour invoquer et récupérer une réponse

```
Invocation.Builder invocationBuilder =
  helloworldWebTargetWithQueryParam.request(Me
  diaType.TEXT_PLAIN_TYPE);
invocationBuilder.header("some-header", "true");

Response response = invocationBuilder.get();
System.out.println(response.getStatus());
System.out.println(response.readEntity(String.class));
```

Rep: S. SAÏDA

41

Création de clients

- * Fluent support

```
Client client = ClientBuilder.newClient(new ClientConfig())
  .register(MyClientResponseFilter.class)
  .register(new AnotherClientFilter());
```

```
String entity = client.target("http://example.com/rest")
  .register(FilterForExampleCom.class)
  .path("resource/helloworld")
  .queryParam("greeting", "HiWorld")
  .request(MediaType.TEXT_PLAIN_TYPE)
  .header("some-header", "true")
  .get(String.class);
```

Possibilité de
récupérer un
autre type que
String
Ex:
typeperso.class

Rep: S. SAÏDA

42

Création de clients

- * Fluent support
- * Version courte:
- * Attention , on perd pas mal de possibilités

```
String responseEntity = ClientBuilder.newClient()
    .target("http://example.com").path("resource/res
t")
    .request().get(String.class);
```

Rep: S. SAÏDA

43

Client, méthode GET

- * Appel par get()
- * Appel par queryparam ou par l'url
 - * (normal)
 - * Reception d'un Objet deserialisé
- * Appel par put post etc.
 - * Possibilité d'envoyer un objet qui est sérialisé
 - * Reception d'un Objet deserialisé

Rep: S. SAÏDA

44

Client, méthode GET

```
public class JClientex04 {
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/res-tws-war/webresources";

    public JClientex04() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(BASE_URI).path("ex04");
    }

    public <T> T getWith(Class<T> responseType) throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path("appel")
            .queryParam("texte", "coucou");
        return resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
    }
}
```

Rep: S. SAÏDA

45

Client, méthode GET

```
Jclientex04 c= new Jclientex04();
String s=c.getXml(String.class);
System.out.println(s);
```

Possibilité de
changer par classe
perso

Rep: S. SAÏDA

46

Client, méthode PUT, GET

Code du service

```
public Exosjs onResource() {
}

/**
 * Retrieves representation of an instance of ws.Exosjs onResource
 * @return an instance of java.lang.String
 */
@Path("/appel")
@GET
@Produces("application/json")
public Texte getJs on() {
    Texte t= new Texte();
    t.setContenu("js on test");
    return t;
}
```

Rep: S. SAÏDA

47

Client, méthode PUT, GET

```
@Path("/appelput")
@PUT
@Consumes("application/json")
@Produces("application/json")
public Texte putJs on(Texte t){

    try{
        t.setContenu(t.getContenu()+"passé par là");
        return t;
    } catch (Exception e){throw new WebApplicationException(404);} //gestion
des erreurs coté Service!
}
}
```

Rep: S. SAÏDA

48

Client, méthode PUT

Code du client :

```
public Jclientex05() {
    client = javax.ws.rs.client.ClientBuilder.newClient();
    webTarget = client.target(BASE_URI).path("ex05");
}

public <T> T getUrion(Class<T> responseType) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("appel");
    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
}

public Response putUrion(Texte t) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("appelput");
    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(Entity.entity(t,
    MediaType.APPLICATION_JSON));
}
```

Rep: S. SAÏDA

49

Client, méthode PUT, GET

//code du client

```
try{
    Jclientex05 c2=new Jclientex05();
    Texte t=c2.getUrion(Texte.class);
    System.out.println(t.toString());
}catch(ClientErrorException e){//to do}

Texte tinput=new Texte();
tinput.setContenu("second test");
Response r=c2.putUrion(tinput);
if (r.getStatus()==200)
{
    Texte t3=(Texte) r.readEntity(Texte.class);
    System.out.println(t3.getContenu());
}
```

Gestion des
erreurs

Toujours à faire

Soit par status
Soit par exception

Rep: S. SAÏDA

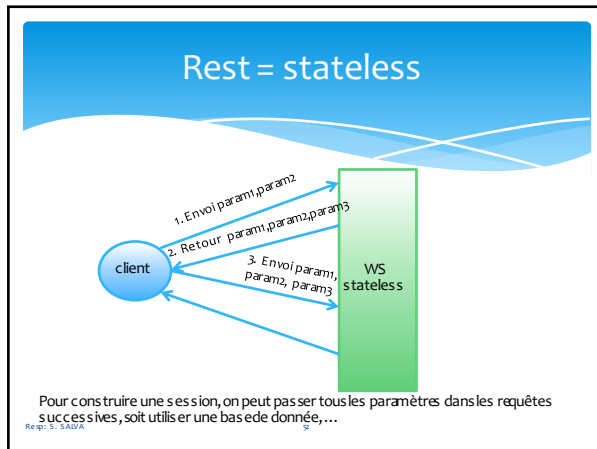
50

Sécurité, coté client

- * SSL
 - * Définir un SSLContext ssl
 - * SSLContext ctx = SSLContext.getInstance("SSL"); ctx.init(null, myTrustManager, null); config.getProperties().put(HTTPSProperties.PROPERTY_HTTPS_PROPERTIES, new HTTPSProperties(hostnameVerifier, ctx));
- * **Http Authentication (basicAuth et autre)**
 - * HttpAuthenticationFeature
 - * HttpAuthenticationFeature feature = HttpAuthenticationFeature.basic("user", "superSecretPassword");

Rep: S. SAÏDA

51



Rest = stateless

- Avec Jersey, plusieurs scopes:
- scope singleton
une seule instance de ressource entre tous les clients par jax-rs application

Annotation @Singleton

Rep: S. SADA 9

Spring boot

Rep: S. SADA 54

Quelques mots sur Spring boot

- * Micro framework conçu pour simplifier le démarrage et le développement de nouvelles applications Spring
- * Permet de créer des services Web Rest
- * Fonctionne également par annotations

Rep: S. SAÏDA

5

Quelques mots sur Spring boot

- * Mise en place de l'application Spring

```
package hello;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloWorldConfiguration { public static void main(String[] args)
{ SpringApplication.run(HelloWorldConfiguration.class, args);
}}
```

SpringApplication.run() -> permet de lancer une appli Web

Rep: S. SAÏDA

5

Quelques mots sur Spring boot

- * Implémentation classe métier

```
package hello;
public class Greeting {
private final long id;
private final String content;
public Greeting(long id, String content) {
this.id = id;
this.content = content; }

public long getId() {return id; }
public String getContent() {return content; }}
```

Rep: S. SAÏDA

5

Quelques mots sur Spring boot

- En Spring, les services sont des contrôleurs

```
package hello;
import ...
@Controller
@RequestMapping("/hello-world")
public class HelloWorldController {
    private static final String template = "Hello,%s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping(method=RequestMethod.GET)
    public @ResponseBody Greeting sayHello(@RequestParam(value="name",
        required=false, defaultValue="Stranger") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

Service récupère une requête GET pour /hello-world et retourne une instance de Greeting
 Rep: S. SAÏDA 38

Quelques mots sur Spring boot

- En Spring, les services sont des contrôleurs, l'annotation `@SpringBootApplication` inclut une annotation `Scan` qui recherche automatiquement tous les contrôleurs

- Code précédent renvoi du JSON par défaut

- Contrôleur persiste ! (pas vraiment stateless...)

- Autre exemple d'annotation Mapping

```
@GetMapping("/students/{studentId}/courses/{courseId}") public Course
retrieveDetailsForCourse(@PathVariable String studentId, @PathVariable String
courseId) {
```

Rep: S. SAÏDA

39
