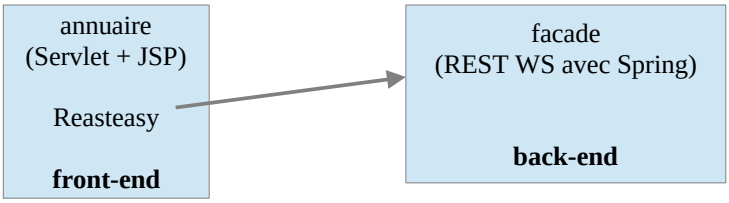
TP Applications Web
application web avec JPA

Daniel Hagimont
Daniel.Hagimont@irit.fr

L'objectif de ce TP est de faire évoluer l'application précédente, mais en gérant les données persistantes dans une base de données avec des Entity beans (JPA) avec Spring.

Nous voulons tout d'abord refactorer l'application du TP2 pour obtenir l'architecture suivante :



1) back-end

Vous devez :

- créer un projet Spring-boot sous vscode (appelé facade) :

View -> Command Palette
-> Spring initializ : Create a Maven Project
-> 3.4.1 -> Java -> n7 -> facade -> War -> 17
-> Spring Data JPA SQL
-> HyperSQL Database SQL

- dans ce projet

- copier les classes Personne, Adresse et Facade
- la classe Facade est un RestController qui fournit l'accès aux méthodes de la Facade sous la forme de REST web service. Vous devez annoter la classe Facade et ses méthodes. Voir le slide ci-dessous (copié du cours).
- pour créer le war, dans votre projet facade: ./mvnw package
- vous pouvez alors déployer ce war (qui est dans target) dans tomcat et tester avec un navigateur.
- NB : deploy.sh est un script à adapter qui fait la génération du war et le déploiement

Exemple : implantation de la Facade (service REST)

```
@RestController
public class Facade {
    private Map<Integer, Compte> comptes = new Hashtable<Integer, Compte>();

    @PostMapping("/addcompte")
    public void addCompte(Compte c) {
        comptes.put(c.getNum(), c);
    }

    @GetMapping("/consultercomptes")
    public Collection<Compte> consulterComptes() {
        return comptes.values();
    }

    @GetMapping("/consultercompte")
    public Compte consulterCompte(@RequestParam("num") int num) throws RuntimeException {
        Compte c = comptes.get(num);
        if (c == null) throw new RuntimeException("Compte introuvable");
        return c;
    }
```

2) front-end

Dans notre projet annuaire, vous pouvez supprimer la classe Facade (qui est maintenant dans le back-end)
Nous voulons que l'annuaire puisse appeler la Facade sous la forme d'un appel REST WS. Pour cela, nous utilisons resteasy.
Il suffit de :

- télécharger resteasy : https://sd-160040.dedibox.fr/hagimont/software/resteasy-jaxrs-3.0.9.Final-all.zip
- ajouter les librairies (lib) de resteasy dans le répertoire lib de votre projet
- créer une interface Facade qui correspond à l'interface de la facade (avec les méthodes ajoutPersonne, ajoutAdresse …)
- annoter cette interface avec les annotations resteasy. Voir le slide ci-dessous (copié du cours).

Exemple : le controleur (interface cliente RestEasy)

```
@Path("/")
public interface Facade {

   @POST
   @Path("/addcompte")
   @Consumes({ "application/json" })
   public void addCompte(Compte c);

   @GET
   @Path("/consultercomptes")
   @Produces({ "application/json" })
   public Collection<Compte> consulterComptes();

   @GET
   @Path("/consultercompte")
   @Produces({ "application/json" })
   public Compte consulterCompte(@QueryParam("num") int num) throws RuntimeException;

   @POST
   @Path("/debit")
   public void debit(@QueryParam("num") int num, @QueryParam("montant") int montant) throws RuntimeException;

   @POST
   @Path("/credit")
   public void credit(@QueryParam("num") int num, @QueryParam("montant") int montant);

}
```

10

- Puis dans la servlet, il vous suffit d'utiliser le proxy généré par resteasy pour appeler la facade. Voir le slide ci-dessous (copié du cours).

Exemple : le controleur (servlet)

```
@WebServlet("/Controller")
public class Controller extends HttpServlet {

   final String path = "http://localhost:8080/bank-spring";

   Facade facade;

   public Controller() {
      super();
      ResteasyClient client = new ResteasyClientBuilder().build();
      ResteasyWebTarget target = client.target(UriBuilder.fromPath(path));
      facade = target.proxy(Facade.class);
   }

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                   throws ServletException, IOException {
      try {
         String action=request.getParameter("action");
         if (action.equals("consulter")) {
            int num=Integer.parseInt(request.getParameter("num"));
            request.setAttribute("num", num);
            request.setAttribute("compte", facade.consulterCompte(num));
         }
```

11

- Vous pouvez regénérer le war et déployer dans tomcat.

Normalement, votre application doit être fonctionnelle, mais découpée en 2 projets.

3) JPA

Nous voulons maintenant gérer les objects (Personne et Adresse) dans la base de données.
La base de données utilisée est la même que dans le tp précédent (HSQLDB).

Pour démarrer la BD :

- récupérez le dossier db du TP2
- dans ce dossier, vous pouvez démarrer la BD avec : source start.sh
- dans ce dossier, vous pouvez exécuter une console SQL avec : source console.sh

Dans votre projet, pour utiliser cette BD, vous devez copier le fichier application.properties dans src/main/resources

Pour avoir une gestion des objets (Personne, Adresse) dans la BD, il faut :

- annoter Personne et Adresse pour qu'il deviennent des entity. Voir le slide ci-dessous (copié du cours).

Entity Compte

```
@Entity
public class Compte {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int num;
    private String nom;
    private int solde;

    public Compte() {}

    public Compte(String nom, int solde) {      // num est généré
        this.nom = nom; this.solde = solde;
    }

    // constructors, setters and getters
```

6

- Ajouter une interface Repository pour chaque entity. Voir le slide ci-dessous (copié du cours).

Entity Compte

```
@Entity
public class Compte {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int num;
    private String nom;
    private int solde;

    public Compte() {}

    public Compte(String nom, int solde) {      // num est généré
        this.nom = nom; this.solde = solde;
    }

    // constructors, setters and getters
```

6

- et ensuite pour pouvez gérer les entity avec les méthodes de ces Repository. Voir le slide ci-

dessous (copié du cours).

Facade

```
@RestController
public class Facade {

    @Autowired
    CompteRepository cr;

    @PostMapping("/addcompte")
    public void addCompte(Compte c) {
        cr.save(c);
    }

    @GetMapping("/consultercomptes")
    public Collection<Compte> consulterComptes() {
        return cr.findAll();
    }

@GetMapping("/consultercompte")
    public Compte consulterCompte(@RequestParam("num") Long num) throws RuntimeException {
        Optional<Compte> c = cr.findById(num);
        if ( ! c.isPresent()) throw new RuntimeException("Compte introuvable");
        return c.get();
    }
```

8

Vous implanterez 3 versions avec différents types de relations entre ces Entity beans :

- OneToMany (bidirectionnelle) : une personne possède une liste d'adresse, chaque adresse ayant un propriétaire unique
- ManyToMany (bidirectionnelle) : une personne possède plusieurs adresses et une adresse peut avoir plusieurs propriétaires
- OneToOne (bidirectionnelle) : une personne a une adresse unique et une adresse a un unique propriétaire

Pour vous facilitez la vie, vous avez un script de déploiement deploy.sh que vous pouvez adapter en fonction des répertoires de
- votre projet servlet (que j'ai appelé annuaire)
- votre projet Spring-boot (que j'ai appelé facade)
- votre répertoire d'installation de Tomcat