Conception de logiciels Adaptés
Exposition de ressources REST

Sébastien Mosser

INF600G - E20 - Séquence 2 - Partie 3

UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels

ace

1 Rétrospective L1

2 Architecture App. Mobile

3 Exposition de ressources (REST)

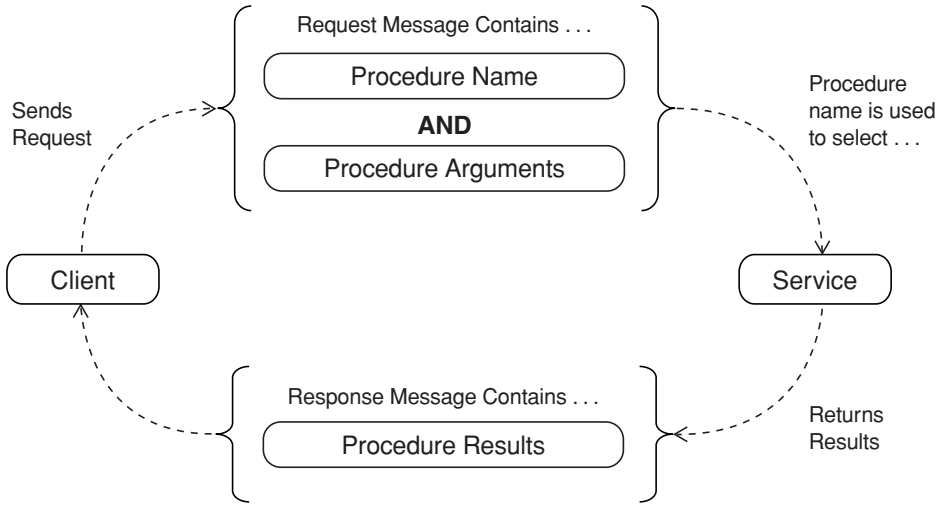
4 Intro. à Android

5 Travail à faire pour L2

Comment définir l'API coté serveur ?

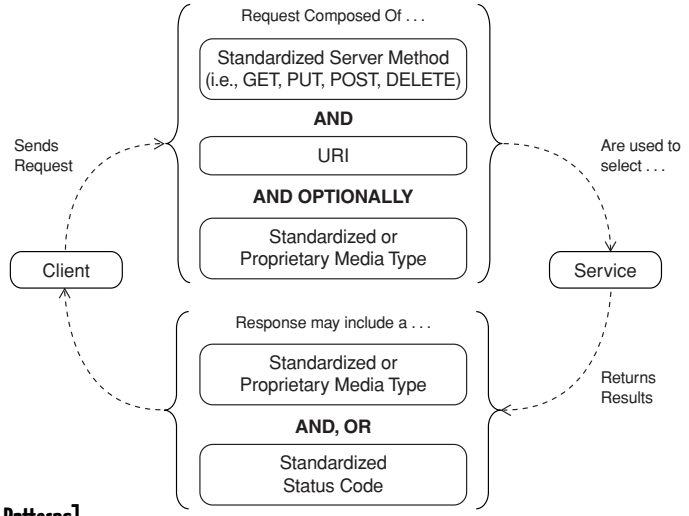
- Approche ***Remote Procedure Call*** (RPC)
  - On expose sur le serveur les ordres qu'il doit être capable de comprendre et exécuter.
  - On expose donc des "verbes" : CreateOrder, DescribeProduct, …
  - Coté Client, on invoque ces méthodes à distance
    - Il faut lire la documentation pour savoir comment appeler
- Approche ***Ressource*** (REST)
  - On expose sur le serveur les ressources de l'application
    - Order, Products, …
  - On manipule les ressources avec des verbes pré-définis

Interaction avec le paradigme RPC



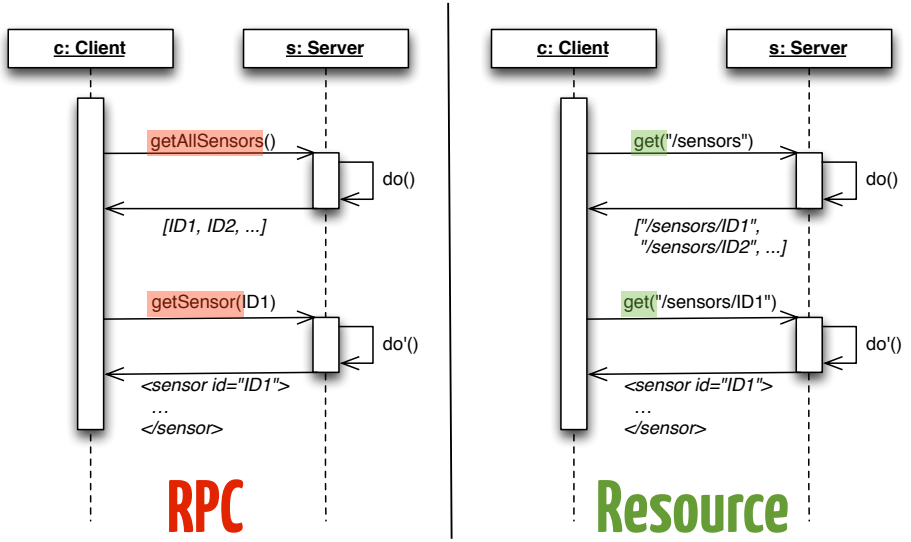
[Service Design Patterns]

Interaction avec le paradigme Ressource

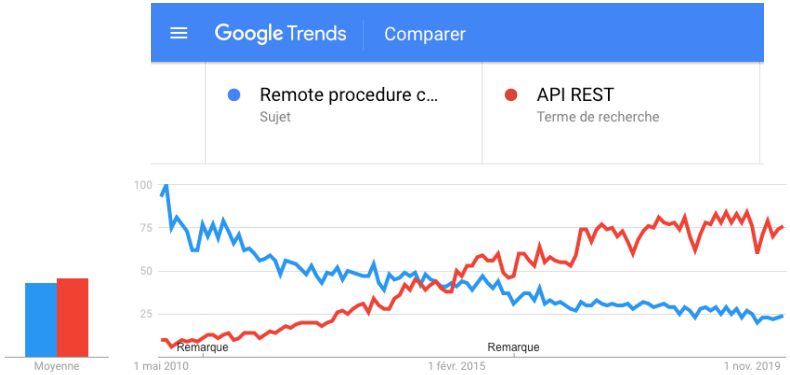


[Service Design Patterns]

Exemple RPC versus Ressource



Encore un effet de mode



https://trends.google.com/trends/explore?date=2010-04-25%202020-05-25&geo=US&q=%2Fm%2F06kl5,API%20REST

Idée clé : Ressources & API Uniformes

- **Approche RPC :**
  - createOrder, describeProduct, addProduct, …
  - Les verbes se multiplient
  - Fragilité des clients qui doivent les invoquer
- **Approche Ressource :**
  - On expose des ressources : Order, Product
  - On manipule les ressources avec des verbes prédéfinis
    - Souvent les verbes de HTTP : GET, POST, PUT & DELETE
    - http://www.ietf.org/rfc/rfc2616.txt

REST au dessus de HTTP

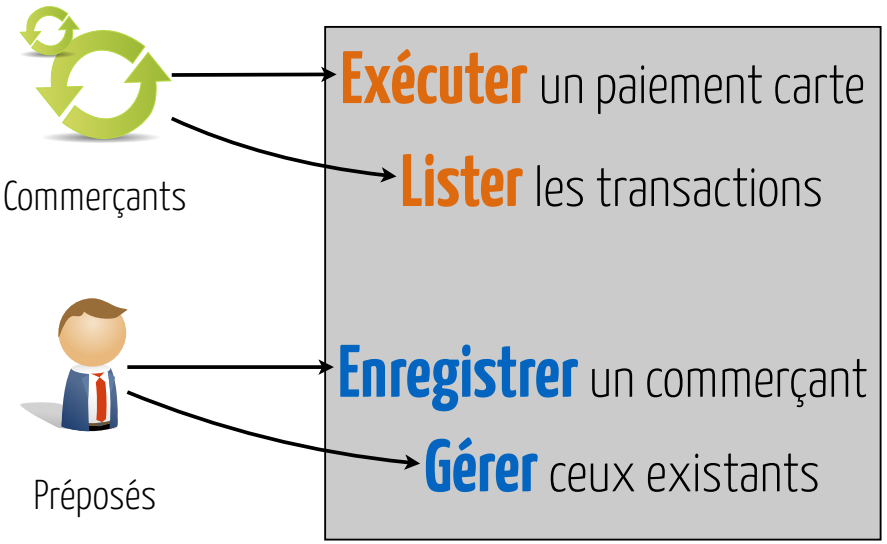
- **On utilise des URLs** pour localiser les ressources
  - http://…/messages : tous les messages
  - http://…/messages/23 : le 23ème message
- **On utilise les verbes HTTP**, et leur sémantique :
  - GET pour obtenir, POST pour créer
  - PUT pour modifier, DELETE pour effacer
- **On utilise les status HTTP** pour donner de l'information
  - 2xx: succès, 4xx erreur client, 5xx erreur serveur

Bonnes pratiques

HTTP Verb	/customers	/customers/{id}
GET	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	404 (Not Found), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
POST	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found).
DELETE	404 (Not Found), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

[Restful best practices]

Étude de Cas : Le Crédit Général



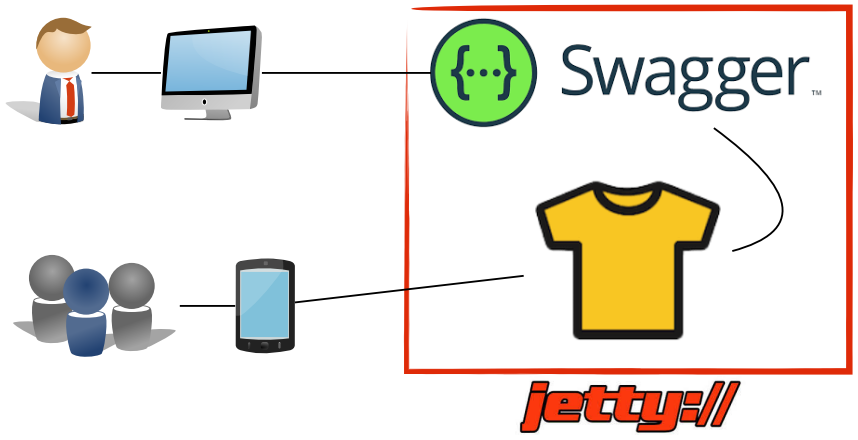
Proposition de solution

- **/rest/payment/public/{id}/transactions**:
  - **GET**: returns the transactions associated to the customer identified by the given id
- **/rest/payment/public/{id}/process**:
  - **POST**: the process used to perform a payment for customer id.

- **/rest/payment/private/retailers**:
  - **GET**: returns a list of links to registered retailers
  - **POST**: create a new retailer (status code: 201), available as a new resource
- **/rest/payment/private/retailers/{id}**:
  - **GET**: read the information stored for a given retailer
  - **PUT**: update an existing retailer with new information
  - **DEL**: delete an existing retailer

Mise en oeuvre INF600G

(L'idée ici est d'avoir une pile technologique en Java)



https://github.com/ace-lectures/E20-INF600G/tree/master/examples/serveur

Mise en place de Jersey

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID"
        version="3.0">
    <display-name>Jersey and Swagger UI example</display-name>
    <servlet>
        <servlet-name>api</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>io.swagger.v3.jaxrs2.integration.resources,ca.uqam.info.inf600g.services</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>api</servlet-name>
        <url-pattern>/api/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Les services du package seront exposés à l'URL préfixée par '/api'

src/main/webapp/WEB-INF/web.xml

Mise en place de Swagger

```yaml
prettyPrint: true
cacheTTL: 0
openAPI:
  info:
    version: '1.0.0'
    title: 'INF600G backend services'
  servers:
    - url: '/api'
```

src/main/webapp/WEB-INF/openapi.yml

Swagger permet de documenter une interface REST avec le standard OpenAPI, et de mettre en place automatiquement une application permettant d'invoquer les services

Lancement du serveur

```
mosser@lucifer serveur % mvn clean package jetty:run-war
…
[INFO] Started Jetty Server
```

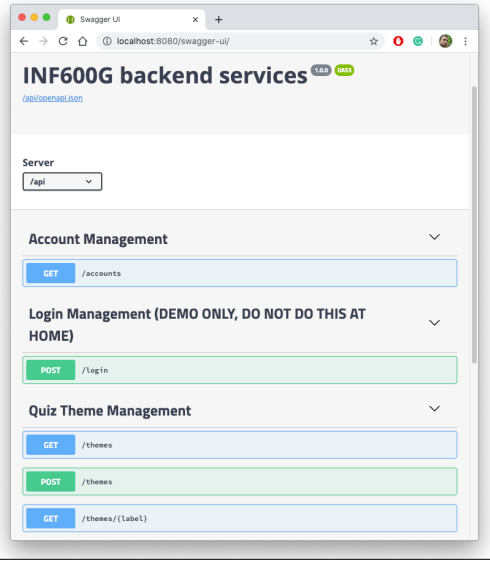
On utilise Jetty lancé directement par Maven, pour avoir un environnement de test directement accessible.

On peut accéder à l'interface Swagger pour tester les services

NE PAS FAIRE ÇA EN PRODUCTION !!

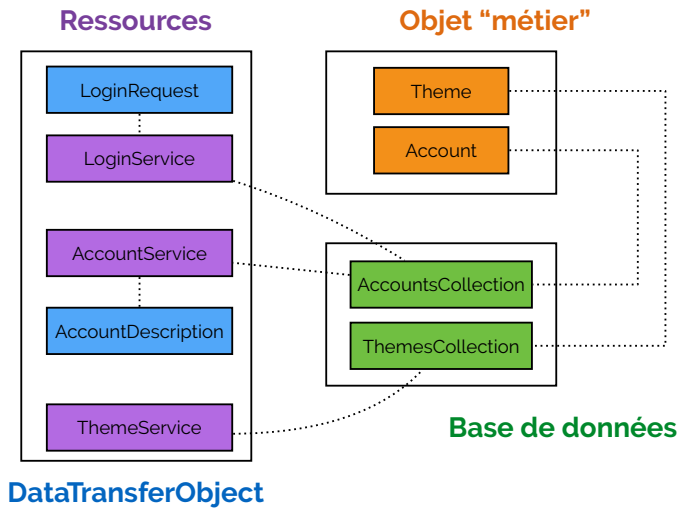
Interface Swagger

[Démo Time]



Chaque ressource est documentée, avec les différents verbes disponibles

Architecture du projet de démonstration



Principes de Marshalling (≈ sérialisation)

JSON

```
{
  "label": "example",
  "name": "Theme d'exemple",
  "description": "Ceci est un thème pour l'exemple"
}
```

@Consumes(MediaType.*APPLICATION_JSON*)

Unmarshalling (intercession)

marshalling (introspection)

@Produces(MediaType.*APPLICATION_JSON*)

POJO

```
public class Theme {

  private String label;
  private String name;
  private String description;

  public Theme() { } // required for marshalling purpose

}
```

Le marshalling peut viser du JSON, du XML, du texte, …

Lister les thèmes des Quiz

```
@Path("/themes")
@Tag(name="Quiz Theme Management")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class ThemeService {

  @GET
  @ApiResponse(
        responseCode = "200",
        content = @Content(mediaType = "application/json",
              array = @ArraySchema(schema = @Schema(implementation = Theme.class))),
        description = "List of all available themes")
  public Set<Theme> getAllThemes() {
     return ThemeCollection.getAccess().getAllThemes();
  }
}
```

Métadonnées sur la ressource

Verbe HTTP

Transformer un Set<Theme> en JSONArray

"Plain Old Java"

Paramètres & codes de retours

Récupérer une ressource d'identifiant donnée

```
@GET
@Path("/{label}")
@ApiResponse(
      responseCode = "200",
      content = @Content(mediaType = "application/json",
            schema = @Schema(implementation = Theme.class)),
      description = "Find a theme using its label")
public Theme getGivenTheme(@PathParam("label") String l) {
   Optional<Theme> value = ThemeCollection.getAccess().findThemeByLabel(l);
   if(value.isEmpty())
      throw new WebApplicationException(Response.Status.NOT_FOUND);
   return value.get();
}
```

On répond 404 (ressource non trouvée)

Manipulation de la réponse HTTP

Une ressource nouvellement créée devrait donner son URL

```
@POST
@ApiResponse(responseCode = "201", description = "Add a new theme in the system")
public Response addNewTheme(Theme theme,  @Context UriInfo uriInfo) {
   try {
      ThemeCollection.getAccess().registerNewTheme(theme);
      return Response
            .created(uriInfo.getAbsolutePathBuilder().path(theme.getLabel()).build())
            .build();
   } catch(IllegalArgumentException iae) {
      throw new WebApplicationException(Response.Status.CONFLICT);
   }
}
```

Les exceptions internes sont transformées en codes d'erreurs HTTP

Principe de DataTransferObject (DTO)

Objet métier

```
public class Account {

   private String identifier;
   private String name;
   private String password;
   private URL avatar;

}
```

GET /api/accounts

Code du service

```
public Set<AccountDescription> getAllAccounts() {
   return AccountsCollection.getAccess()
         .getAllAccounts()
         .stream().map(AccountDescription::new)
         .collect(Collectors.toSet());
}
```

```
public class AccountDescription {

   public final String identifier;
   public final String name;
   public final URL avatar;

   public AccountDescription(Account origin) {
      this.identifier = origin.getIdentifier();
      this.avatar = origin.getAvatar();
      this.name = origin.getName();
   }
}
```

DTO