TD Service Web
Gestions de Polygones

0. Modalités

Rendu

- Le travail est à effectuer en **binôme** et à rendre **au plus tard le 2 octobre à 23h59 (heure française).** Les binômes ne sont pas forcément ceux que vous avez défini en début d'année, mais doivent être formés dans vos groupes de projets respectifs.
- Le rendu se fait par e-mail. L'email doit être envoyé **aux deux enseignants** et avoir pour sujet: "[IDSE-GL] TD WS NOM1-NOM2", où NOM1 et NOM2 sont les noms des deux membres du binôme.
- Elements attendus : zip nommé "nom1-nom2.zip" contenant: le fichier pom.xml, le dossier src, un fichier readme.txt avec le nom des membres du groupe, et le cas échéant les points sur lequels vous avez bloqué, et pourtquoi. (Pas de dossier target/ ni site/)

Contraintes

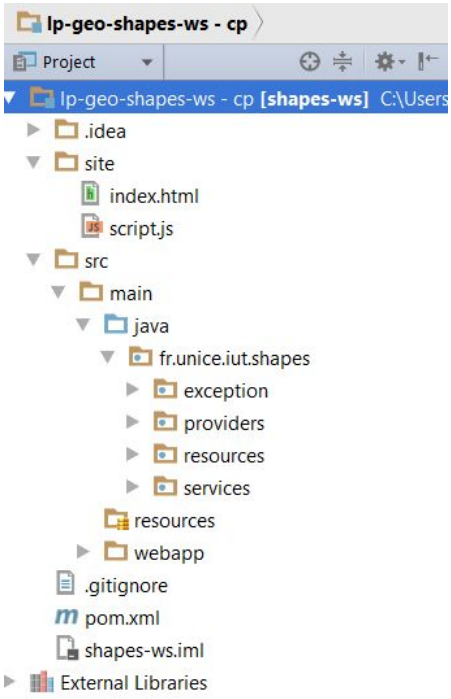
- **Lisez impérativement la notice en fin de page 2 avant de commencer à coder.**

Parmi les fichiers fournis, **il est interdit de** :

- Modifier la structure des dossiers ou déplacer des fichiers
- Modifier les fichiers pom.xml et web.xml
- Modifier les fichiers du répertoire site/
- Supprimer ou ajouter des fichiers et/ou des classes Java.
- Modifier les signatures des opérations définies dans le code (paramètres, exceptions jetées, etc.)

I. Prise en main

- Télécharger le code du projet sur la page du cours.
- Lancer IntelliJ Idea et choisir **d'importer** un projet.
- Sélectionner dans le dossier du projet, le fichier **pom.xml**. Valider plusieurs fois.
- Lorsqu'on vous demande un SDK, s'il n'y en pas dans la liste, cliquer sur le + et choisissez JDK
- Indiquer un répertoire contenant un jdk>6 (Généralement dans Program Files > Java)
- Valider et cliquer sur Finish pour finir l'importation du projet.



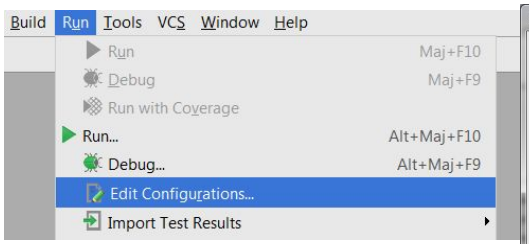
- Laisser le projet charger.
- Un fois fait, vous devriez voir à gauche le contenu du dossier

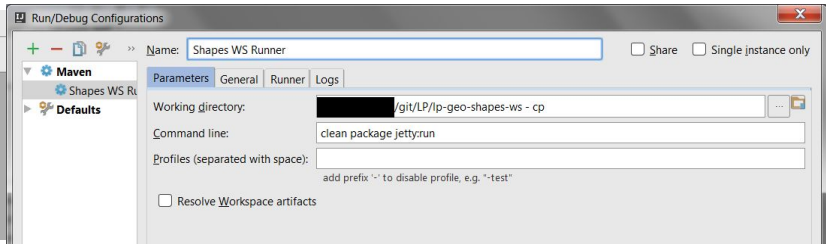
(Si ce n'est pas le cas: View > Tool Windows > Project)

- Le code source se trouve dans le dossier **src/main/java**
  - Ouvrir fr.unice.iut.shapes.services.ShapeService.java
  - C'est l'interface qui définit les méthodes du service
- Le dossier **site/** contient une page html qui lancera des requêtes au service web afin de le tester.

Lancement du service web :

- Dans le menu **'Run'**, sélectionner '**Edit configuration**'
- Cliquer sur le + en haut à gauche pour créer une nouvelle configuration.
- Choisir l'option **'Maven'**.
- Dans 'Command line', écrire '**clean package jetty:run**'.
- Ne pas toucher au reste. Valider.
- Cliquer sur la flèche verte en haut à droite pour lancer le service. La console va apparaître. Une fois la ligne '*[INFO] Starting scanner at interval of 10 seconds.*' affichée, le service est lancé ! (Note: il est normal que certains messages s'affichent en rouge, ce ne sont pas des erreurs).





- Ouvrir le fichier **index.html**:
  Tout est indiqué KO et des erreurs doivent apparaître dans la console.
- Au travail pour corriger tout ça !

IMPORTANT

Le fichier html fournit effectue des requêtes à l'API Google Map à chaque chargement à l'aide d'une clé API créée par les enseignants. Bien que les quotas d'utilisation de l'API soient assez élevés et ne devraient pas être dépassés, il vous est demandé de ne pas rafraîchir à outrance la page. En particulier, il y a une limite de 100 requêtes toutes les 100 secondes. Si les quotas sont dépassés pendant la séance, la clé sera désactivée et chaque groupe devra créer sa propre clé.

Si des abus sont constatés pendant ou après la séance, la clé sera immédiatement désactivée. Dans tous les cas, la clé ne sera plus valide dès la date de rendu. Chaque groupe devra à terme obtenir sa propre clé pour le projet.

Ne rafraichissez pas la page avant d'être sûrs d'avoir coupé, puis relancé le service.

II. Travail demandé

1. Récupération de la liste des formes

Méthode : **GET**
url : http://localhost:8080/rest/shapes/
sortie attendue : Tableau Json contenant pour chaque forme un objet json avec son identifiant et le nombre de sommets (vertices) -> [{"id":"square","vertices":4},..]

- Trouver la méthode dans `ShapesService` et son implémentation dans `ShapesServiceImpl`
- Que fait l'implémentation? Attention, dans la suite vous ne devez **pas** modifier cette méthode.
- Ouvrir l'URL http://localhost:8080/rest/shapes/. Qu'est-ce qui cause l'erreur?
- Corriger l'opération qui causait l'exception précédente.
- Relancer le service et ré-essayer d'accéder à l'url.
- Le contenu de tous les objets `Shape` a été transformé automatiquement en json et renvoyé.
- Modifier la classe `Shape` afin que le résultat corresponde à ce qui est attendu:
  - Prendre exemple sur la classe Point pour les annotations @XMLRootElement et @XMLElement
  - Ne pas envoyer la liste des points. Pour cela, il faut utiliser l'annotation @XmlTransient
  - Un élément indiquant le nombre de sommets de la forme doit être défini.
- Charger site/index.html. Le 1er exercice à gauche devrait être marqué **OK**.

Note: Ici nous utilisons des annotations pour transformer automatiquement un objet Java dans le format qui nous intéresse. La transformation se fait vers le format json en raison de l'annotation `@Produces(MediaType.APPLICATION_JSON)` définie dans l'interface du service.

Nous aurions pu obtenir le même résultat en créant des objets json manuellement, ce qui nous aurait évité d'avoir à ajouter un attribut redondant à notre classe Shape.

```
JSONArray array = new JSONArray();
for (Shape shape : ShapesProvider.getAllShapes()) {
   JSONObject shapeDescription =new JSONObject();
   shapeDescription.put("id", shape.getId());
   shapeDescription.put("vertices", shape.getPoints().size());
   array.put(shapeDescription);
}
return Response.ok(array).build();
```

2. Transformation de forme

Le coeur du service web est le passage d'une forme définie par la classe Shape, à un Polygone utilisable par le service Google Map.

Méthode : **GET**
url : http://localhost:8080/rest/shapes/{shapeId}
paramètres d'url:
- lat: la latitude du premier sommet du polygone
- lng: la longitude du premier sommet du polygone
- length: la longueur du premier segment du polygone

Réponse : HTTP status **200** + Json contenant une liste de coordonnées géographiques
-> {"vertices" : [{"lng":12,"lat":-4}, {"lng":15,"lat":-42}, ...] }

- Trouver la méthode dans `ShapesService` et son implémentation dans `ShapesServiceImpl`
- Implémenter l'opération :
  - Récupérer la forme depuis `ShapesProvider`
  - Appeler la méthode `PolygonCalculator.transformShape(..)`
  - Implémenter les méthodes de calculs sur les Points utilisée par `transformShape`
  - Renvoyer l'objet `GeoPolygon` obtenu dans la réponse
- Essayer d'appeler l'url http://localhost:8080/rest/shapes/square?lat=12&lng=3&length=50.
- Modifier les classes nécessaires pour que le json généré soit au format demandé ci-dessus.
- Charger site/index.html. Le 2ème exercice à gauche devrait être marqué **OK** et deux formes devraient être affichées sur la carte.

3. Création de forme

Méthode : **POST**
url : http://localhost:8080/rest/shapes/
paramètres: Chaine de caractère json: { "id": blabla , "points": [{"x": 0, "y": 1}, {"x": 1, "y": 1}, …]}
Réponse : HTTP status **201** (Response.Status.CREATED) + l'identifiant de la nouvelle forme

- Trouver la méthode dans `ShapesService` et son implémentation dans `ShapesServiceImpl`
- Transformer l'entrée json en objet `Shape`.
  - Utiliser les objets `JSONObject` et `JSONArray` du package `org.codehaus.jettison.json`
  - Parser une string en Json: `new JSONObject(myString )`
  - Récupérer un attribut Json: `myJSONObject.getString("key")` (getInt, getDouble, etc.)
  - Récupérer un tableau Json: `myJSONObject.getArray("key")`
- Définir l'opération `createShape()` de `ShapesProvider` qui ajoute une forme à la map.
- Ajouter la forme en appelant cette méthode.

- Charger site/index.html. Le 3ème exercice à gauche devrait être marqué **OK** et une forme triangulaire bleue devrait être affichée sur la carte.

4. Suppression de forme

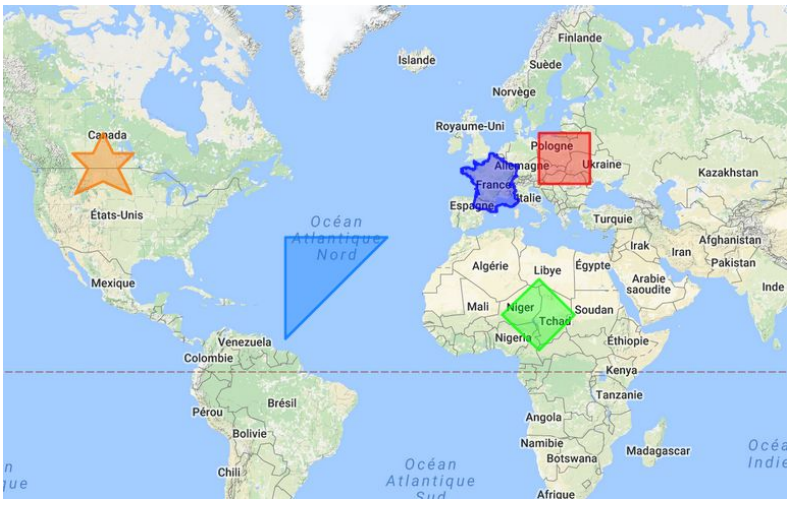
Méthode : **DELETE**
url : http://localhost:8080/rest/shapes/{shapeId}
paramètres: aucun
Réponse : HTTP status **204** (Response.Status.NO_CONTENT) (signifie que l'opération a bien été effectuée, mais ne nécessite pas de renvoyer quoi que ce soit en retour)

- La méthode n'est cette fois pas définie dans `ShapesService`
- La créer, et l'implémenter dans `ShapesServiceImpl`
- Utiliser l'opération `deleteShape()` de `ShapesProvider` pour supprimer la forme
- Charger site/index.html. Le 4ème exercice à gauche devrait être marqué **OK** et 3 formes devraient être affichées sur la carte.

5. Rotation de forme

La dernière étape est de reprendre la méthode de la partie 3 et de gérer le cas où l'on fait tourner les formes.

- Appeler l'url http://localhost:8080/rest/shapes/square?lat=12&lng=3&length=50&rot=45.
- Pourquoi la rotation ne fonctionne-t-elle pas?
- Implémenter la méthode qui pose problème.
  - La rotation se fait par rapport à un point d'origine : Pour simplifier nous utiliserons le 1er sommet du polygone.
    Pour chaque sommet du polygone:
      Transformer le GeoPoint en Point avec la méthode `PolygonCalculator.fromPointToLatLng`
      Appeler la méthode `rotate` de la classe `Point` avec les paramètres approriés.
      Retransformer le point obtenu en coordonnées géographiques (`fromLatLngToPoint`)
      Mettre à jour l'objet GeoPoint avec les nouvelles coordonnées
- Charger site/index.html. Le 5ème exercice à gauche devrait être marqué **OK** et deux formes supplémentaires devraient être affichées sur la carte.



6. Gestion d'erreurs

Les 5 derniers exercices consistent à revenir en arrière et gérer quelques cas d'erreur potentiellement non pris en compte jusque là.

- GetShape: Gérer le cas où la forme demandée n'est pas définie
- GetShape: Gérer le cas où certains paramètres obligatoires sont manquants
- Ajout d'une forme: Gérer le cas où une forme avec l'id demandé existe déjà
- Ajout d'une forme: Gérer le cas où le json fourni n'a pas le bon format
- Suppression d'une forme: Gérer le cas où la forme demandée n'est pas définie

Indications:

- Certaines méthodes doivent jeter des exceptions du type `ShapeAlreadyExistException`
- Renvoyer une réponse HTTP correspondant au problème rencontré.
  - 200: return Reponse.ok(/objet renvoyé/).build()
  - 200, alternative: Response.status(Response.Status.OK).entity(/objet renvoyé/).build()
  - Cas général: `Response.status(/status/).entity(/objet renvoyé/).build()`
- Codes HTTP:
  - 1XX: information
  - **2XX**: la requête s'est bien passé
  - 3XX: redirection
  - **4XX**: erreur côté client
  - **5XX**: erreur côté serveur
- Codes HTTP utiles pour le tp:
  - 200 - Ok Response.Status.OK
  - 201 - Created Response.Status.CREATED
  - 204 - No content Response.Status.NO_CONTENT
  - 400 - Bad request Response.Status.BAD_REQUEST
  - 404 - Not found Response.Status.NOT_FOUND
  - 500 - Internal server error Response.Status.INTERNAL_SERVER_ERROR