
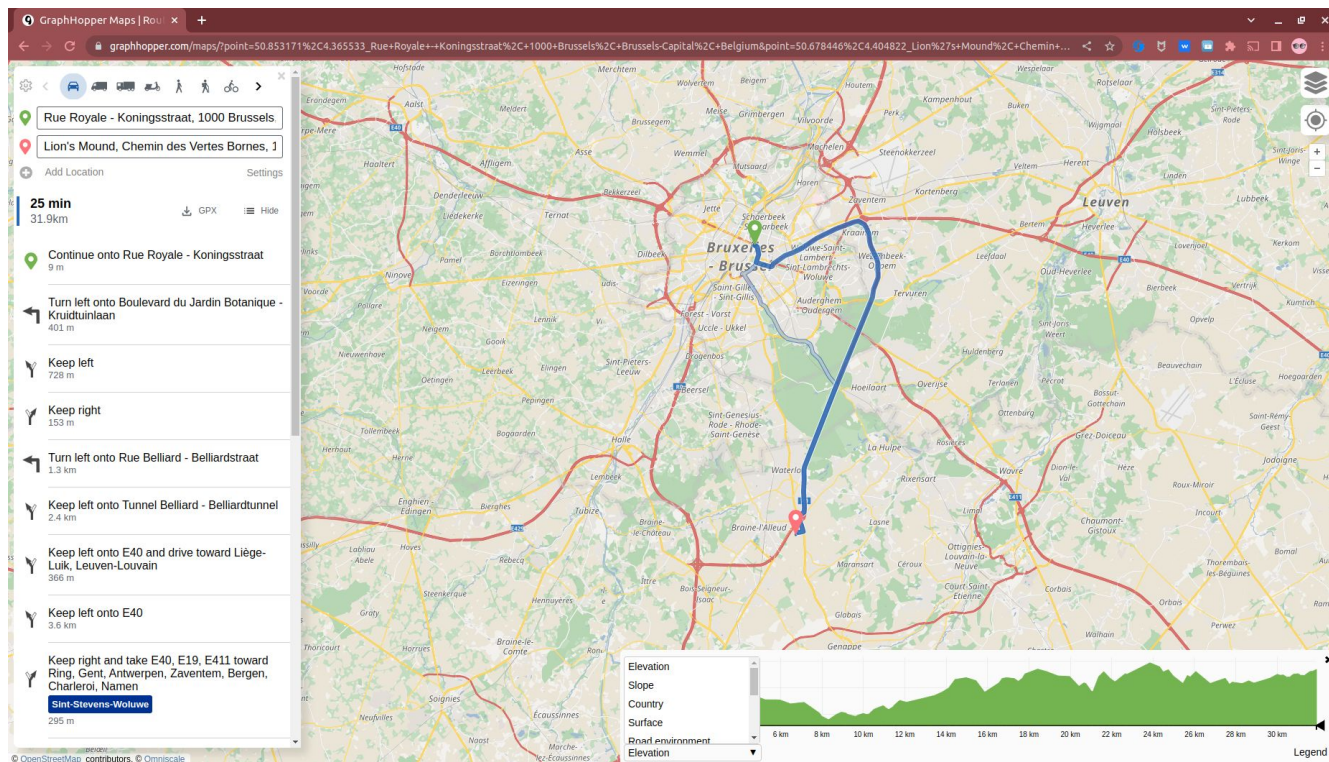


Une application de covoiturage

Hackathon 2024 - mardi 14 et mercredi 15
mai

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

GraphHopper : le calcul d'itinéraire



Java et les services rest

GraphHopper

Search

- Explore our APIs
- Map Data and Routing Profiles
- Custom Model
- Route Optimization API
- Routing API
 - GET** GET Route Endpoint
 - POST POST Route Endpoint
- Matrix API
- Geocoding API
- Isochrone API
- Map Matching API
- Cluster API
- Profiles API

GET Route Endpoint

For the GET request you specify the parameters in the URL and can try it directly in every browser. However, it has some disadvantages when using many points (URL length limit) and the `custom_model` Feature cannot be used. Therefore, our recommended endpoint is the POST route endpoint.

Security > api_key

Request

QUERY PARAMETERS

profile	string (VehicleProfileId) Default: "car" The routing profile. It determines the network, speed and other physical attributes used when computing the route. See the section about routing profiles for more details and valid profile values.
point required	Array of strings The points for which the route should be calculated. Format: <code>latitude,longitude</code> . Specify at least an origin and a destination. Via points are possible. The maximum number depends on your plan. Example: <code>point=51.131,12.414&point=48.224,3.867</code>
point_hint	Array of strings The <code>point_hint</code> is typically a road name to which the associated <code>point</code> parameter should be snapped to. Specify no <code>point_hint</code> parameter or the same number as you have <code>point</code> parameters.
snap_prevention	Array of strings Optional parameter to avoid snapping to a certain road class or road environment. Currently supported values are <code>motorway</code> , <code>trunk</code> , <code>ferry</code> , <code>tunnel</code> , <code>bridge</code> and <code>ford</code> . Multiple values are specified like <code>snap_prevention=ferry&snap_prevention=motorway</code> . Please note that in order to e.g. avoid motorways for the route (not for the "location snap") you need a different feature: a custom model.
curbside	Array of strings Optional parameter. It specifies on which side a point should be relative to the driver when she leaves/arrives at a start/target/via point. You need to specify this parameter

Request samples

GET /route >

Request samples

Copy

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
    .url("https://graphhopper.com/api/1/route?point=51.131,12.414&point=48.224,3.867")
    .build();

Response response = client.newCall(request).execute();
```

Response samples

200 400 401 429 500

application/json

Copy Expand all Collapse all

```
{
  "hints": {
    "visited_nodes.sum": 58,
    "visited_nodes.average": 58
  },
  "info": {
    "copyrights": [ - ],
    "took": 2
  },
  "paths": [
    { - }
  ]
}
```

Une application dans le terminal

```
Bienvenue à l'application de Covoiturage.
```

```
-----
```

```
Cette application permet :
```

- de publier une liste de places
disponibles pour des trajets en voitures
- de rechercher un trajet dans les trajets disponibles

```
Le calcul des itinéraires est effectué via https://graphhopper.com/api.
```

```
Ce service nécessite une clé api pour fonctionner.
```

```
Inscrivez vous via https://www.graphhopper.com/ et créez une clé  
api via le menu API KEYS avant de continuer.
```

```
Entrez votre clé api
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Une application dans le terminal

Vous allez pouvoir publier une liste de trajets en encodant pour chaque trajet :

- une adresse d'origine
- une adresse de destination
- un nombre de places disponibles

Il y a actuellement 3 trajet(s) encodé(s)

- Trajet numéro 1 sur 3

De Rue Royale 67 , 1000 Bruxelles vers Le Domaine , 7940 Cambron-Casteau

Places disponibles 2 - Durée 00:54:28 - Distance 57.79 km

- Trajet numéro 2 sur 3

De Rue Royale 67 , 1000 Bruxelles vers 1020 Bruxelles

Places disponibles 1 - Durée 00:12:37 - Distance 5.33 km

- Trajet numéro 3 sur 3

De Rue Royale 67 , 1000 Bruxelles vers 1410 Waterloo

Places disponibles 3 - Durée 00:28:17 - Distance 29.65 km

Souhaitez-vous continuer à publier des trajets (Y)?

N

Une application dans le terminal

```
Entrez une ville d'origine et une ville de destination  
pour afficher les trajets correspondants disponibles.
```

```
Entrez le nom de la ville d'origine
```

```
Bruxelles
```

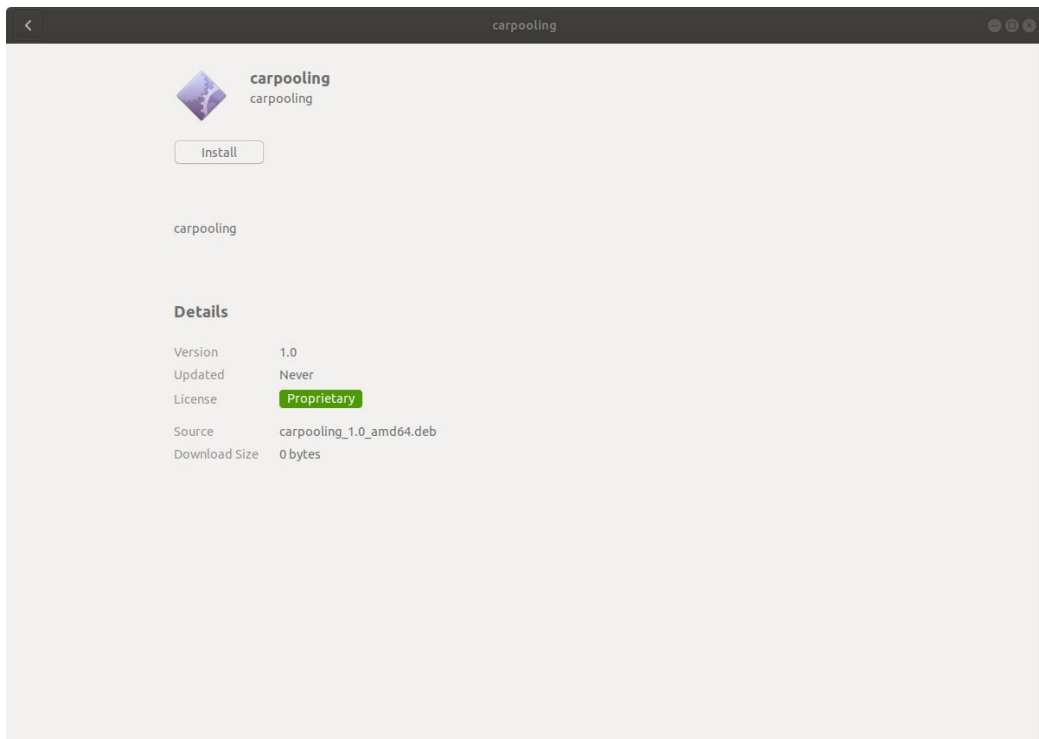
```
Entrez le nom de la ville de destination
```

```
Waterloo
```

Une application dans le terminal

```
Pour aller de la ville de Bruxelles à la ville de Waterloo
- Trajet numéro 1 sur 1
  De Rue Royale 67 , 1000 Bruxelles vers 1410 Waterloo
  Places disponibles 3 - Durée 00:28:17 - Distance 29.65 km
  Détails des instructions
    Continuez sur Rue Royale - Koningsstraat - 0.40 mètres - 00:01:20
    Tournez fort à droite sur Boulevard Bischoffsheim - Bischoffsheimlaan - 0.39
mètres - 00:00:39
    Restez sur la gauche - 0.73 mètres - 00:01:02
    Restez sur la droite - 0.15 mètres - 00:00:15
    Tournez à gauche sur Rue Belliard - Belliardstraat - 1.30 mètres - 00:02:03
    Restez sur la gauche sur Tunnel Belliard - Belliardtunnel - 2.38 mètres -
00:02:25
    Restez sur la gauche sur E40 et conduisez vers Liège-Luik, Leuven-Louvain -
0.37 mètres - 00:00:12
    Restez sur la gauche sur E40 - 3.61 mètres - 00:01:58
    Restez sur la droite et prendre E40, E19, E411 vers Ring, Gent, Antwerpen,
Zaventem, Bergen, Charleroi, Namen - 0.30 mètres - 00:00:14
    Restez sur la droite et prendre E19, E411 vers Bergen, Charleroi, Namen, Ring
- 17.44 mètres - 00:11:24
    Restez sur la droite sur Chaussée de Tervuren - 0.47 mètres - 00:01:11
    Restez sur la droite sur Chaussée de Tervuren - 0.09 mètres - 00:00:15
    Tournez à droite sur Avenue Reine Astrid - 1.95 mètres - 00:04:58
    Restez sur la droite sur Rue de la Station - 0.02 mètres - 00:00:03
    Au rond-point, prenez la 2e sortie vers Place de la Gare - 0.06 mètres -
00:00:13
    Arrivée - 0.00 mètres - 00:00:00
  ---
```

Créer un exécutable



Utiliser l'api GraphHopper

GraphHopper est une librairie open source écrite en Java qui permet d'obtenir des informations cartographiques grâce à openstreetmap. Par exemple on peut obtenir via ce service un itinéraire entre une adresse d'origine et une adresse de destination. C'est la librairie utilisée derrière le lien <https://graphhopper.com/maps/?profile=car&layer=Omniscale> que vous pouvez tester.

Pour utiliser cette librairie dans un programme Java, il existe plusieurs techniques. La méthode sur laquelle nous vous guidons est d'utiliser la librairie via des appels à une API.

Défi 1

Inscrivez-vous sur le site de <https://www.graphhopper.com/> et créez une clé API via le menu dédié. Une fois cette clé créée, testez dans votre browser préféré la route (l'url) qui permet de calculer un itinéraire entre deux adresses en utilisant votre clé api. Vous trouverez comment construire cette url dans la documentation : <https://docs.graphhopper.com/#tag/Routing-API>

Si vous avez une erreur lors de la création d'un compte (One Account Allowed) vous pouvez utiliser cette clé en attendant : 16b80343-2e79-4a68-a124-9af318c41484 Réessayez plus tard de créer un compte (si tout le monde essaie en même temps, il croit à une attaque).

Alternative: se connecter via son smartphone

Remarquez que les adresses sont passées en paramètre à l'url sous le format (*latitude,longitude*). **Vérifiez parmi les routes mises à disposition** par GraphHopper si il existe une route qui associe les adresses à des coordonnées (*latitude,longitude*).

Attention un nombre limité de recherche gratuite est autorisé pour chaque utilisateur. Pensez à consulter votre solde de requêtes via <https://graphhopper.com/dashboard/#/statistics> lorsque vous aurez réussi à calculer un itinéraire.

Défi 2

Dans la documentation <https://docs.graphhopper.com/#tag/Routing-API> vous avez sûrement constaté qu'un exemple de code Java est disponible pour appeler cette api.

En vous basant sur cet exemple, **créez un programme java** qui permet de consulter un itinéraire entre deux adresses. Ce code exemple demande d'ajouter la dépendance à la librairie okhttp à votre programme : <https://mvnrepository.com/artifact/com.squareup.okhttp3/okhttp> . Mettez à

jour le fichier `pom.xml` pour intégrer cette dépendance et commencez à coder !

Votre programme de calcul d'itinéraire terminé, vous êtes prêt à passer à l'étape suivante !

Créer un exécutable

Au lieu d'exécuter une application Java via IntelliJ, nous vous proposons de créer un exécutable de votre application qui pourra être utilisé sur différents systèmes d'exploitation. Autrement dit à partir de votre code Java vous allez créer un fichier avec une extension comme "exe", "msi", "rpm", "deb", "pkg", "dmg",... Cette extension dépendant du système d'exploitation sur lequel vous travaillez.

Afin de réussir cette tâche, nous vous proposons de réussir plusieurs défis qui vous conduiront au résultat final.

Défi 1

Commencez par **créer avec IntelliJ une application en langage java utilisant maven** qui affiche dans le terminal le traditionnel "Hello Wolrd".

Défi 2

En utilisant la barre d'outil de maven, **exécutez la commande** `maven compile`

Cette commande va générer le fichier `.class` de votre projet. **Trouvez** où ce fichier `.class` a été généré.

Voici quelques informations sur la barre d'outils si vous en avez besoin : <https://www.jetbrains.com/help/idea/maven-projects-tool-window.html>

Défi 3

En utilisant la barre d'outil de maven, **exécutez la commande** `maven package`

Cette commande va générer le fichier `.jar` de votre projet. **Trouvez** où ce fichier `.jar` a été généré.

Pouvez-vous afficher "Hello World" dans un terminal via la commande `java -jar _nom_de_votre_jar_` ?

Si un message d'erreur du type `no main manifest attribute` apparaît, **modifiez le fichier `pom.xml`** de votre projet et ajoutez le plugin `maven-assembly`

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.7.1</version>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-
dependencies</descriptorRef>
        </descriptorRefs>
        <appendAssemblyId>>false</appendAssemblyId>
        <finalName>${project.artifactId}</finalName>
        <archive>
          <manifest>
            <addClasspath>>true</addClasspath>
            <mainClass>votrePackage.NomDeLaClasseMain</mainClass>
          </manifest>
        </archive>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Lors de ce copier-coller, veillez à ce que **le nom de la classe main soit mis à jour avec le nom que vous avez choisi.**

Générez à nouveau le fichier `.jar` de votre projet. Pouvez-vous afficher "Hello World" dans un terminal via la commande `java -jar _nom_de_votre_jar_?`

Si vous vous interrogez sur le code ajouté au `pom.xml`, il provient de la documentation de `maven-assembly` : <https://maven.apache.org/plugins/maven-assembly-plugin/usage.html>

Défi 4

Ouvrez un terminal et avec l'aide des commandes `jlink` et `jpackage` et du fichier `.jar` produit au défi précédent, **créez un exécutable** adapté à votre système d'exploitation (`.exe`, `.deb`, ...). Essayez d'exécuter le résultat :

- via un terminal

- via un double clic sur l'icône de l'exécutable

Quelle différence constatez-vous ?

Note : Si vous travaillez sur les machines de l'école, vous aurez besoin de Wix lors de l'exécution de `jpackage`. Vous pouvez le télécharger à l'adresse suivante

: <https://github.com/wixtoolset/wix3/releases/download/wix3141rtm/wix314-binaries.zip>

Les commandes complètes pour `jlink` et `jpackage` sont :

```
jlink --strip-native-commands --no-header-files --no-man-pages --strip-debug --add-modules "java.base" --output target/java-runtime
```

```
jpackage --type exe --dest build --input libs --name demo --main-class intd2.Demo --main-jar demo.jar --runtime-image target/java-runtime --win-per-user-install --win-console
```

Avant d'exécuter la deuxième commande pensez à créer le dossier `libs` et à y placer votre `.jar`. N'oubliez pas d'adapter le paramètre `main-class` avec le nom de votre classe.

Remarques sur la version de `okhttp`

Si vous avez ajouté la librairie `okhttp` comme dépendance, prenez garde à la version ajoutée. Les versions 4.10 à 5.0 peuvent provoquer lors de l'exécution de l'exécutable l'exception `java.lang.NoClassDefFoundError: java/util/logging/Logger`. Le problème vient d'une dépendance de la librairie `okhttp` absente du `jar`. Vous devez modifier la commande `jpackage` comme suit pour contourner ce problème:

```
jpackage --type exe --dest build --input libs --name demo --main-class intd2.Demo --main-jar demo.jar --add-modules java.logging --win-per-user-install --win-console
```

Vous savez désormais comment créer un exécutable à partir d'un projet java utilisant maven. Passez à l'étape suivante !

Mettre en place l'application

Si vous avez terminé les défis précédents, vous possédez le code Java nécessaire pour créer votre application finale.

L'application de covoiturage doit respecter les contraintes suivantes :

- elle doit permettre à un utilisateur de publier des trajets
- elle doit permettre à un utilisateur de rechercher des trajets

- elle doit être fournie sous forme d'un exécutable, le système sous lequel l'application est exécutée ne doit pas obligatoirement avoir Java d'installé

Vous êtes libres d'imaginer la vue associée à ce développement.

Bon amusement