# Logiciel de calcul d'itinéraire

## 1 Introduction

Le génie logiciel est un sujet principalement empirique : ses bases s'acquièrent et s'entretiennent par une pratique assidue de la programmation. C'est pourquoi, dans le cadre du module de génie logiciel avancé, l'équipe enseignante va vous guider dans la réalisation d'un projet logiciel de relativement grande envergure. Celle-ci servira de véhicule et d'illustration aux concepts et techniques abordées durant le cours magistral :

- programmation objet moderne avec patrons de conception (design patterns),
- tests unitaires, tests d'intégration et intégration continue,
- méthodes agiles (e.g., SCRUM) pour le développement en équipe,
- et bien d'autres choses encore.

L'équipe enseignante répondra aux questions posées sur la plateforme en ligne du cours, située sur moodle [1]

1. https://moodle.u-paris.fr/course/view.php?id=10699

# Table des matières

# 2 Méthodologie

Avant de décrire le contenu effectif du projet, cette section va couvrir les aspects méthodologiques à prendre en considération pour ce projet. Il s'agit d'éléments évalués dans votre rendu.

## 2.1 Organisation

### 2.1.1 Formation des équipes

Les projets doivent être réalisés en équipes indépendantes comprenant chacune six étudiants (hexanômes) appartenant au même groupe de TP. La méthode de constitution des groupes est laissée au chargé de TP encadrant les étudiants.

### 2.1.2 Calendrier

Le projet commence à la quatrième séance de TP.
Le projet doit être rendu, dans sa version finale, avec la documentation associée le 5 mai 2023. Le détail du rendu est décrit dans la section 3.5.

**Plagiat** Si les échanges d'idées entre équipes sont permis et mêmes encouragés, l'échange de code est lui strictement proscrit. Tout plagiat entraînera l'attribution d'un zéro final aux équipes concernées. De la même façon, tout usage de document ou de code existant dans les documentations associées au projet ou dans le code doivent faire l'objet d'un crédit aux auteurs originels.

## 2.2 Qualité du code

Nous attendons de votre projet :
- un code conforme aux exigences de qualité logicielle standard (fiable, commenté judicieusement, dont la maintenance est simple, facilement extensible, performant, sûr...),
- des procédures de construction (build) et de déploiement documentées, avec l'utilisation des outils standard du domaine,
- une procédure de test automatisée rigoureuse, avec une bonne couverture de votre code ($\geqslant 75\%$).

## 2.3 *Continuous Integration*

L'*intégration continue* (ou *continuous integration* en anglais) permet, entre autre, d'exécuter votre suite de tests automatiquement, notamment à chaque *merge request*.
Vous devez mettre en place un tel système. Vous êtes encouragés à utiliser le support de GitLab pour ce faire, via le concept de *runner* [2], qui est installé sur Gaufre [3]. Si vous utilisez un autre système, vous devez vous assurez que le chargé de TP qui s'occupe de votre équipe y a accès, et arrive à le consulter.
L'utilisation de Docker est fortement conseillée.

2. https://docs.gitlab.com/runner/
3. https://gaufre.informatique.univ-paris-diderot.fr/

### 2.4 *Continuous Delivery*

La *publication continue* (ou *continuous delivery* en anglais) permet d'automatiser la production des artéfacts logiciels qui résultent de votre travail. Elle permet un processus de publication léger, où une version du logiciel correspond simplement à un *tag* dans son dépôt Git. Lors d'une poussée d'un nouveau *tag*, votre projet doit produire et publier automatiquement l'archive au format *zip* de la version correspondante. Cette archive doit contenir les fichiers nécessaires à l'exécution du projet, en particulier l'archive jar contenant le bytecode Java et le script shell qui simplifie son exécution. Un onglet *releases* sur GitLab doit permettre de télécharger les archives récentes.

### 2.5 Pour aller plus loin

Les instructions données ci-dessus fournissent une base méthodologique minimale pour renforcer la qualité logicielle de votre projet. Vous êtes fortement incités à mettre en place des procédures supplémentaires. Par exemple, il peut être bénéfique d'intégrer des outils d'analyse statique tels que SonarQube ou Infer à vos processus de développement.

## 3 Description générale

### 3.1 Aperçu

Vous allez réaliser, lors de ce projet, un logiciel de planification d'itinéraire urbain. Implémenté en Java, ce logiciel permettra de lire une carte des transports, puis, en donnant les coordonnées GPS de départ et d'arrivée, calculera le trajet le plus court (en temps, en distance, etc...) pour réaliser ce déplacement.
Ce genre d'application existe, et on pourra s'en inspirer pour augmenter le cahier des charges.

### 3.2 Déroulement et organisation

L'organisation des équipes pour le projet est laissée à la discrétion de chacune.
On suggère pour la première itération du cycle analyse de besoin → conception → implémentation de passer au moins deux semaines sur la partie analyse de besoin et conception (avant d'écrire du code).

### 3.3 Technologies

Votre projet devra être codé en Java 17+.
Vous devez utiliser un gestionnaire de dépendances et un outil de *build*. Les outils suggérés sont Gradle [4]et Maven [5](renseignez-vous auprès de vos chargés de TP avant de prendre votre décision).
Tout usage d'une bibliothèque externe, en plus de celles contenues dans le *Java Development Kit* et de *JUnit*, doit être approuvé par l'équipe enseignante.

### 3.4 Interface Utilisateur

Le choix du type d'interface est laissé à l'initiative de chaque équipe. Elle doit cependant être documentée afin de pouvoir être lancée depuis n'importe quel terminal et utilisée facilement par l'équipe enseignante.

4. https://gradle.org/install/
5. https://maven.apache.org/install.html

**Facilité d'utilisation** Prenez note que la facilité d'utilisation est pour partie dans la note, en **malus**. C'est à dire qu'une interface jugée trop compliquée à utiliser ou trop contre-intuitive vous coutera des points.

### 3.5 Évaluation du projet

Le rendu comportera trois parties : une distribution du logiciel, un relevé du code, et une documentation.

#### 3.5.1 Distribution Logiciel

La distribution logicielle doit pouvoir être installée et exécutée sans utilisation de vos IDE et sans installation de programme tiers.

#### 3.5.2 Code

Le relevé du code aura lieu automatiquement, vous n'avez donc rien à faire. Seule la branche principale (master ou main) de vôtre dépôt git sera prise en compte, veillez donc à ce qu'elle contienne la version finale de votre code.
Votre dépôt doit contenir tout ce qui est nécessaire à la construction, à l'exécution et à la compréhension de votre projet : code, infrastructure de développement (y compris les tests).

#### 3.5.3 Documentation

La documentation doit comprendre trois volets : une documentation utilisateur, une documentation technique (diagramme de classe + Java doc) et une vidéo.

**Vidéo** Il s'agit d'une vidéo décrivant votre produit et NON votre travail individuel (i.e. ce que chacun a fait).
Elle doit contenir une présentation des fonctionnalités avec démonstration. Chaque groupe fournit une vidéo où tous les membres parlent. La durée maximale de la vidéo est de n minutes où n est la taille du groupe en personnes (e.g. si le groupe contient 6 étudiants, la vidéo ne doit pas dépasser 6 minutes). La vidéo doit être rendu en poussant un lien YouTube ou autre (git push -m "video") sur la branche main/master avant la date limite (5 mai 2023). Ceci sera votre dernier commit (aucun commit postérieur ne devra donc avoir lieu).

## 4 Fonctionnalités

Le client a fourni un prototype de lecteur de la carte des transports, réalisé par un tiers lors d'une étape préalable. Ceci est une simple preuve de concept, et ne devrait pas être simplement traduit.
Il désire maintenant développer une véritable application. Il a choisi un certain nombre de fonctionnalités nécessaires. Il pourrait en ajouter ou les modifier au cours du projet.

### 4.1 Données de l'application

Bien que n'ayant pas encore fourni les données exactes, le client a fourni une spécification initiale des formats dans lesquels elles devraient être fournies. Ces formats sont sujets à changement.

| Num | Fonctionnalité | Description |
|---|---|---|
| LECT_NET | Lecture du réseau | L'application doit être capable de lire la description d'un réseau de transport arbitraire |
| LECT_TIME | Lecture des horaires | L'application doit être capable d'afficher les horaires de passage d'un transport à une station donnée |
| PLAN_0 | Trajet dans réseau | L'application doit être capable de donner une séquence de transport, avec horaire, pour se rendre d'une station à une autre station. |
| PLAN_1 | Trajet optimisé dans le réseau | L'application doit permettre d'obtenir une séquence optimale en terme de temps comme de distance parcourue. |
| PLAN_2 | Trajet avec début ou fin à pied | L'application doit permettre de partir de, et d'arriver à, n'importe quelle coordonnée géographique (pas seulement des arrêts du réseau). |
| PLAN_3 | Trajet avec section à pied | L'application doit permettre de choisir de marcher entre deux stations géographiquement proches si aucune ligne ne fait la jonction à l'heure voulue. |

Table 1 – Fonctionnalités de l'application

#### 4.1.1 Données de cartes

Le client voudrait pouvoir lire les données du réseau à partir d'un fichier CSV. Typiquement ce fichier contiendra les informations suivantes :

1. Les arrêts du réseau (avec leur coordonnées GPS)
2. Les lignes du réseau
3. Les temps de trajet sur une ligne entre deux arrêts consécutifs
4. La distance de transport entre les arrêts

Concrètement, le fichier est constitué de ligne décrivant chacune une section de trajet entre deux arrêts consécutifs. Pour une ligne B qui passe par l'arrêt Jeanne d'Arc, puis l'arrêt Jean Jaurès, fait le trajet en 1m42s, avec 8,43 km de distance, on aura dans le fichier une ligne telle que montré ci-après.

```
Jeanne d'arc; 43.60887,1.44544; Jean Jaurès; 43.60573,1.44883; B; 1:42; 8.43;
```

#### 4.1.2 Données d'horaire

Les horaires sont associés à une ligne et au terminus de départ. On considère dans un premier temps que les horaires sont fixés sur une journée et identique tous les jours.
Le client a indiqué que les horaires sont fournis dans des fichiers CSV dont une ligne typique contient le numéro de ligne, le terminus de départ et l'heure du départ. Ainsi, pour une ligne B qui part toutes les heures entre 7h et 23h de son terminus de départ Borderouge, on aurait un fichier CSV contenant :

### 4.2 Considération supplémentaires

Pour des raisons de propriété intellectuelle, il veut utiliser un langage compilé, mais veut aussi pouvoir exécuter l'application dans divers environnements. Il a donc *imposé l'usage de Java.*

```
B; Borderouge; 7:00;
B; Borderouge; 8:00;
B; Borderouge; 9:00;
B; Borderouge; 10:00;
B; Borderouge; 11:00;
B; Borderouge; 12:00;
B; Borderouge; 13:00;
B; Borderouge; 14:00;
B; Borderouge; 15:00;
B; Borderouge; 16:00;
B; Borderouge; 17:00;
B; Borderouge; 18:00;
B; Borderouge; 19:00;
B; Borderouge; 20:00;
B; Borderouge; 21:00;
B; Borderouge; 22:00;
B; Borderouge; 23:00;
```

## 5 Suggestions de l'équipe enseignante

**Algorithme de plus court chemin** Il existe plusieurs algorithmes de plus court chemin qui peuvent être implémentés. Parmis ceux-ci l'équipe enseignante suggère Dijkstra ou A star.

## A Crédits

Ce document est inspiré et reprend des parties du sujet de GLA des années précédentes, écrit par Adrien Guatto et Stefano Zacchiroli, notamment dans l'introduction et la section méthodologie.