Cahier de conception
Application Kayak

GROUPE K

Table des matières

Introduction

Notre projet Kayak consiste en la réalisation d'une application d'agent de sortie dans Paris. Le système devra proposer successivement trois sorties : dans un bar puis un restaurant et enfin une boîte de nuit.
Ce projet devra fournir une interface avec en entrée : une date et un horaire de début de soirée, les adresses des participants, leur moyen de transport au cours de la soirée et leurs préférences alimentaires pour le choix du restaurant.
Il fournira en sortie une liste chronologique des trois adresses de sortie.
Ce cahier de conception est la deuxième phase de notre projet. Nous allons exposer nos exigences de réalisation et présenter un modèle physique de l'application en considérant toutes les contraintes imposées pour sa réalisation.

Exigences de réalisation

Sécurisation
Le besoin de sécurité pour cette application est minime, il n'y aura pas de données sensibles, comme un code de carte bleue par exemple. Un utilisateur devra uniquement donner anonymement des adresses avec de trouver des localisations ou des préférences spécifiques pour des restaurants. Kayak ne garde aucune donnée une fois l'opération effectuée : il n'y a donc aucun risque de vol de données.

Performance
L'objectif de conception est d'avoir un temps de traitement le plus faible possible. Nous allons donc limiter le nombre de réponses renvoyées par Google par un maximum afin de ne pas attendre plusieurs minutes pour trouver le meilleur résultat. Une interface graphique fluide et facile d'utilisation sera nécessaire pour une bonne performance.
Nous avons choisi de ne pas garder les données de l'utilisateur, ce qui peut influencer la performance de notre application. En effet, si un utilisateur refait une recherche similaire, on aurait pu directement lui renvoyer le triplet qui a été envoyé auparavant. Mais nous avons décidé de favoriser la sécurité.

Robustesse
L'application doit pouvoir tenir malgré les calculs et les communications avec le serveur Google (cf performance).

Synchronisation
L'application doit pouvoir communiquer avec les serveurs Google et récupérer toutes les réponses.

Analyse et conception

Spécificités techniques

- L'application sera développée pour un PC
- Les langages choisis sont JAVA pour le cœur de notre application et JSON pour la communication avec les serveurs Google
- L'environnement de développement se fera sur Eclipse avec un répertoire Git afin de favoriser les échanges entre les collaborateurs du projet
- Il y aura une interface graphique faite avec SWING
- Héritages entre nos classes possibles
- L'application utilisera différentes API pour communiquer avec les serveurs Google

Librairies

- Communication avec Google : JSON, Google Maps Distance Matrix API, Google Maps Direction API, Google Places API Web Service

- Interface : javax.swing, utilisation de JApplet, JDialog, JFrame, JWindow

- Tests : JUnit

- Autre : Arraylist, Date

Interface graphique

Deux possibilités nous ont été offertes très vite : SWING ou JavaFX. JavaFX est plus fluide et permet plus de libertés. Néanmoins, l'interface graphique n'étant pas la priorité dans notre projet, nous avons donc décidé d'utiliser SWING car plus simple d'utilisation.

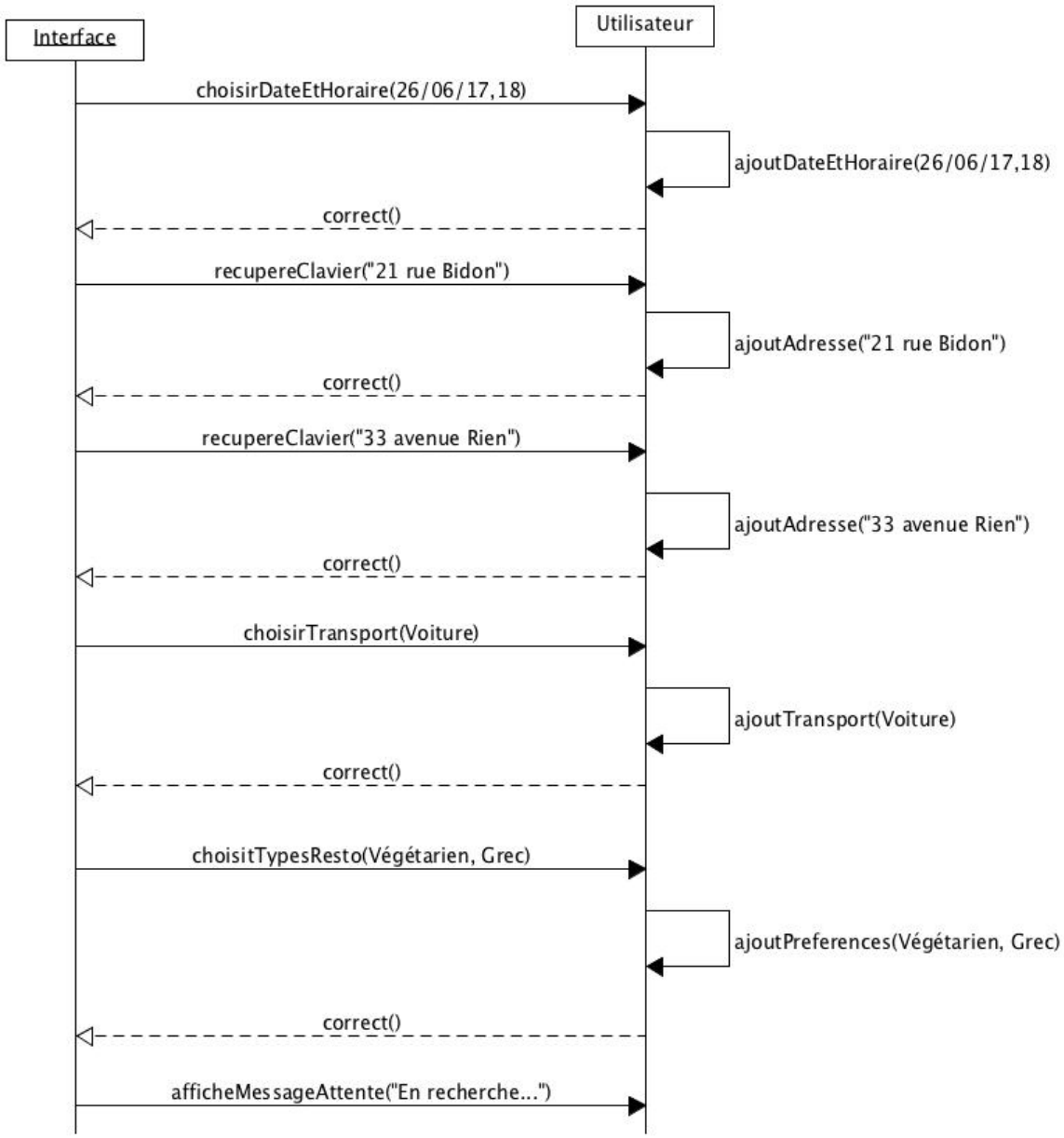
Fonctionnement du système et diagrammes UML

Détails des interactions

L'utilisateur interagira avec l'application via l'interface de la manière suivante :

- Il entre une date et choisit dans un menu déroulant un horaire de début de sortie
- Il entre son adresse ainsi que celles de ses amis participants à la sortie
- Il choisit un moyen de transport dans un menu déroulant que tout le monde va utiliser pendant la soirée
- Il choisit les préférences alimentaires et les types de restaurant dans lequel ils veulent manger (par exemple : chinois, végétarien, grec, …)
- Il valide ces informations en cliquant sur un bouton
- Pendant le calcul, une fenêtre de chargement s'affiche pour signaler que la demande est en cours de traitement
    - Voir le diagramme de séquence 1 illustré plus loin
- Une fois un établissement trouvé, l'utilisateur aura le choix de l'accepter ou de le refuser (le processus est décrit plus loin)
- À l'écran s'affiche enfin le triplet « Bar-Restaurant-Boîte » avec leur adresse respective

Diagramme de séquence 1



Les rôles des classes et des méthodes seront détaillés plus loin dans ce cahier (cf Développement).
Sur ce diagramme, « Utilisateur » est une classe de notre projet et ne représente pas l'acteur, c'est-à-dire la personne qui utilise l'interface. Cette classe enregistre les données de l'utilisateur seulement, et constitue un objet représentant celui qui utilise l'application.

Détails du noyau du système

Le noyau du système repose sur la gestion des données que l'utilisateur aura renseignées et l'interaction entre l'application et les différentes API Google.
Voici le procédé de recherche des 3 adresses de sorties :

- L'application enregistre les données de l'utilisateur (dans la classe Utilisateur) afin qu'elle puisse les utiliser par la suite (par exemple, les adresses des participants seront enregistrées dans une ArrayList)
- Elle convertit ces données en JSON afin de pouvoir les envoyer au serveur Google
- À l'aide de règles mathématiques simples *****(expliqué plus loin), l'application trouve le point à égale distance de toutes les adresses des participants pour décider d'une localisation de départ pour la recherche d'un bar (barycentre)
- Elle choisit un périmètre maximum dans lequel va se trouver les trois établissements de sortie
- L'application demande à l'API Google Places de nous trouver une liste de bar les plus proches de la localisation de départ
- Par défaut, notre application choisira le premier bar de la liste afin de le renvoyer à l'utilisateur
- L'utilisateur accepte ou refuse l'établissement proposé par notre application. S'il refuse, on propose le deuxième bar de la liste et ainsi de suite.
  S'il accepte, notre application va faire le même processus pour le restaurant puis la boite de nuit. Elle renvoie le temps de trajet entre l'établissement d'avant ou le lieu de départ et celui renvoyé, avec le moyen de transport que l'utilisateur a choisi, calculé par l'API Google Matrix
- Pour le restaurant, l'application renseigne les préférences des participants au serveur Google
- Lorsque l'utilisateur aura accepté la boite de nuit, l'application lui renvoie le triplet final « Bar-Restaurant-Boite » avec les adresses respectives et peut accepter définitivement ce triplet
- En quittant l'application, les données de l'utilisateur ne sont pas enregistrées

***** Le calcul du barycentre :
Comme nous récupérons les coordonnées GPS des adresses, nous avons donc des coordonnées de type (xA,yA), (xB, yB), etc…
Pour que chaque participant parcoure à peu près la même distance et qu'il n'y ait pas de jaloux, nous avons décidé de calculer le barycentre de ces coordonnées.
Pour ce faire, il suffit de faire la moyenne des abscisses et la moyenne des ordonnés.
On obtiendra alors la coordonnée [(Xa+Xb+…+Xn))/n , (Ya+Yb+…+Yn)/n].

Le cas d'utilisation de l'application avec les différents acteurs :

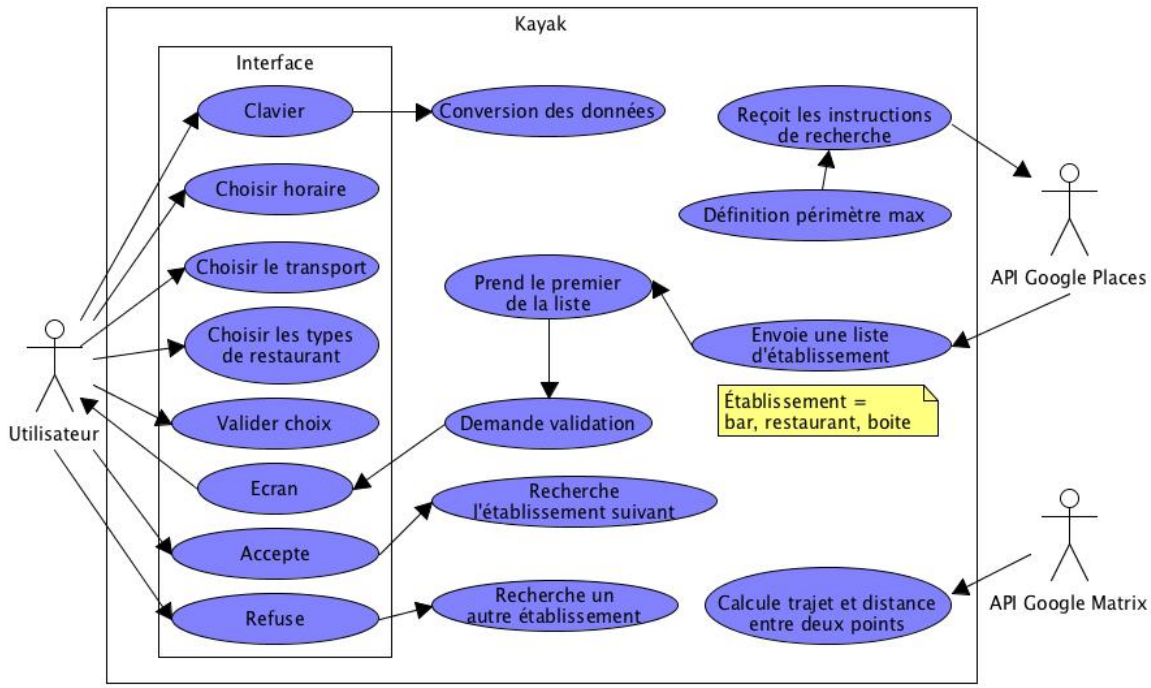


Diagramme de cas d'utilisation 1

- Le clavier se charge de récupérer les informations renseignées par l'utilisateur telles que les adresses des participants
- L'écran communique à l'utilisateur les informations renvoyées par l'application
- La conversion des données se charge de convertir les données de l'utilisateur en JSON, langage dans lequel les API Google traitent les informations
- Valider choix sera un bouton de validation final du triplet « Bar-Restaurant-Boite »

Développement

Types énumérés

- TypesTransport : {Marche, Vélo, Voiture, TransportCommun}
  ➔ Regroupe tous les transports pour se déplacer. L'utilisateur en choisira un seul et les trajets seront calculés avec ce type de transport.

- TypesResto : {Végétarien, Grec, Burger, Italien, …}
  ➔ Regroupe quelques types de restaurant que l'utilisateur pourra choisir (en cochant par exemple)

- TypeHoraire : {17, 18, 19, 20, 21}
  ➔ Regroupe des entiers correspondant aux horaires de début de soirée. L'utilisateur en choisira une.

Les classes

En dessous un diagramme de classe qui fait le lien entre l'interface et la classe Utilisateur :

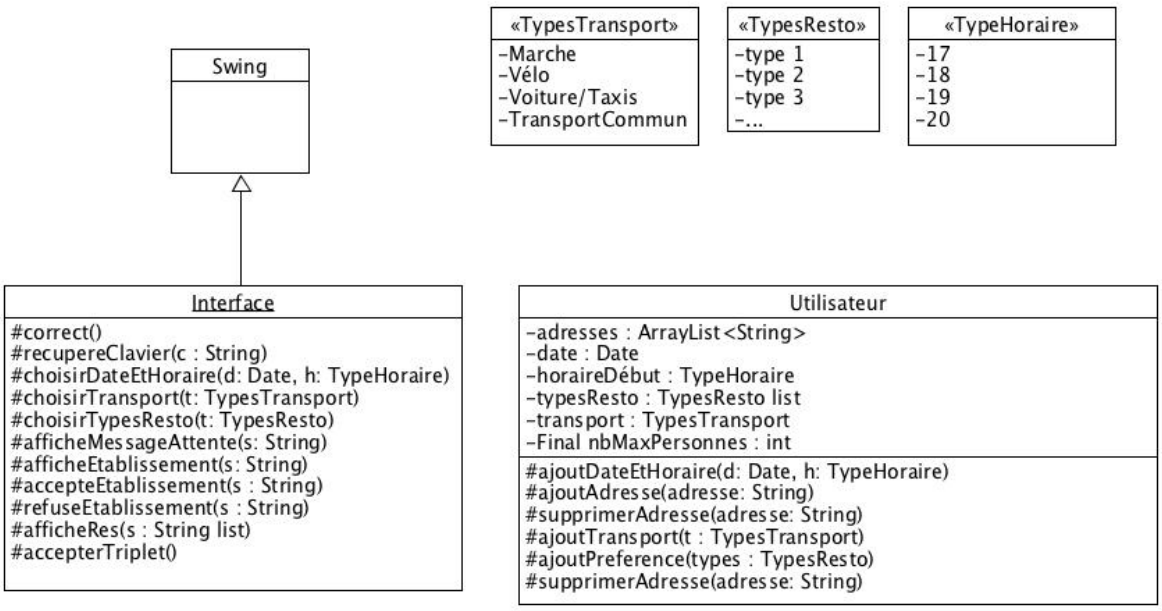


Diagramme de classe 1

Classe Interface

Cette classe hérite d'une classe proposée par la librairies SWING : JFrame.
Elle va créer l'interface que va utiliser l'utilisateur pour rentrer ses informations et où vont être affichés les résultats.

Listes des méthodes et rôles :

- recupereClavier(s : String)
  - ➔ Récupère les informations données par l'utilisateur via le clavier telles que les adresses des participants
- choisirDateEtHoraire(d : Date, h : TypeHoraire)
  - ➔ L'utilisateur rentre la date du jour dans un champ et choisit dans un menu déroulant l'horaire de début (parmi ceux dans le type TypeHoraire
- choisirTransport(t : TypesTransport)
  - ➔ L'utilisateur choisit un type de transport parmi la liste de TypesTransport
- choisirTypesResto(t : TypesResto)
  - ➔ L'utilisateur choisit des types de restaurant parmi la liste de TypesResto. Nous allons proposer plusieurs types que l'utilisateur pourra cocher
- afficheMessageAttente(s : String)
  - ➔ Une méthode déclenchée lorsque la recherche des établissements est lancée. Affiche un message d'attente pour l'utilisateur
- afficheEtablissement(s : String)
  - ➔ Affiche l'établissement trouvé (bar, restaurant, boite) avec son adresse pour l'utilisateur
- accepteEtablissement(s : String)
  - ➔ L'utilisateur peut accepter l'établissement choisit par l'application. Si oui la méthode propose un bouton que l'utilisateur doit cliquer, la méthode déclenche alors la recherche de l'établissement suivant
- refuseEtablissement(s : String)
  - ➔ L'utilisateur peut refuser l'établissement choisit par l'application. De la même façon que la méthode accepteEtablissement(s), elle propose un bouton. La méthode déclenche la recherche d'un même établissement.
- afficheRes(s : String list)
  - ➔ Affiche le résultat c'est-à-dire le triplet « Bar-Restaurant-Boite de nuit » et leur adresse respective
- accepterTriplet()
  - ➔ Bouton pour l'utilisateur afin qu'il puisse accepter le résultat final et quitter l'application

Classe Utilisateur

Classe représentant un utilisateur de notre application. L'utilisateur aura donné ses informations via l'interface et notre constructeur contiendra donc toutes ses informations afin de pouvoir les exploiter. Cette classe agira donc comme un objet que l'on modifiera pour chaque utilisateur.

Listes des attributs :

- Adresses : ArrayList<String>
  ➔ Une ArrayList qui contient toutes les adresses des participants que l'utilisateur aura renseigné
- Date : Date
  ➔ La date de la sortie
- HoraireDébut : TypeHoraire
  ➔ L'horaire de début de la sortie
- TypesResto : ArrayList<TypesResto>
  ➔ La liste des préférences pour le restaurant renseignée par l'utilisateur
- Transport : TypeTransport
  ➔ Le transport choisi par l'utilisateur
- NbPersonneMax : int
  ➔ Il est « final ». Correspond au nombre de participants maximum qu'un utilisateur peut rentrer dans l'application

Listes des méthodes et rôles :

- ajoutDateEtHoraire(d : Date, h : TypeHoraire)
  ➔ Enregistre la date d et l'horaire h renseignés par l'utilisateur
- ajoutAdresse(s : String)
  ➔ Enregistre l'adresse s dans l'attribut Adresse. Méthode appelée pour chaque ajout d'une adresse
- supprimerAdresse(s : String)
  ➔ L'utilisateur aura la possibilité de supprimer une adresse d'un des participants s'il s'est trompé ou qu'il ne participe plus à la sortie. Cette méthode supprimera l'adresse s de Adresse
- ajoutTransport(t : TypeTransport)
  ➔ Enregistre le transport t
- ajoutPreference(p : TypesResto)
  ➔ Enregistre la préférence p dans l'attribut TypeResto. Si l'utilisateur a renseigné plusieurs types alors la méthode est appelée autant de fois qu'il y a de types à ajouter.

Classe Gestion des données

Cette classe est le noyau principal de notre application. Elle va gérer l'ensemble des données, les envoyer et les récupérer du serveur Google. Il s'agira de convertir les données de l'utilisateur en JSON et inversement convertir les résultats du serveur afin de les envoyer à l'utilisateur. Cette classe s'occuper donc de la recherche des établissements pour la sortie.

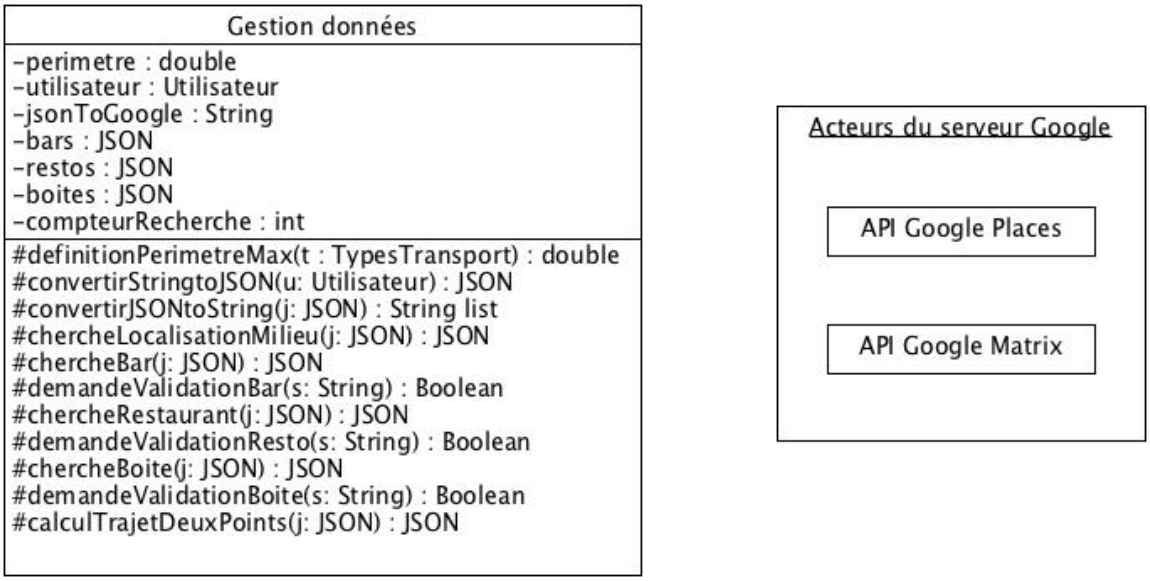


Diagramme de classe 2

Listes des attributs :

- Périmètre : double
  - ➔ Le périmètre maximum dans lequel va se trouver les trois établissements de sortie. Il sera déterminé en fonction du type de transport choisit
- Utilisateur : Utilisateur
  - ➔ L'objet Utilisateur dont il est question pour la recherche. On va utiliser ses attributs
- Bars : JSON
  - ➔ Les bars sous forme de JSON que l'API Google Places va renvoyer. On a décidé de stocker ces résultats dans un attribut que l'on va ensuite exploiter pour convertir en chaine de caractère.
- Restos : JSON
  - ➔ Les restaurants sous forme de JSON que l'API Google Places va renvoyer.
- Boites : JSON
  - ➔ Les boites sous forme de JSON que l'API Google Places va renvoyer.

- CompteurRecherche : int
  ➔ Initialisé à 1, il est incrémenté quand une recherche est refusée afin que le lieu renvoyé ne soit pas le même que celui refusé. Réinitialisé à 1 lorsque le lieu a été validé par l'utilisateur

Listes des méthodes et rôles :

- définitionPérimètreMax(t : TypesTransport) : double
  ➔ En fonction du transport, cette méthode initialise un périmètre plus ou moins grand de recherche d'établissement. Par exemple, si le moyen de transport est la marche alors le périmètre va être plus petit que si le moyen de transport était la voiture
- convertirStringtoJSON(u : Utilisateur) : JSON
  ➔ Se charge de convertir les données de l'Utilisateur u en JSON pour que les serveurs Google puissent faire ses recherches
- convertirJSONtoString(j : JSON) : String
  ➔ Se charge de convertir les résultats en JSON du serveur Google en String afin de les envoyer à l'utilisateur
- chercheLocalisationMilieu(j : JSON) : JSON
  ➔ Avec les adresses des participants, cette méthode va rechercher un point équidistant entre toutes ces adresses pour que le trajet pour se retrouver soit à peu près équitable pour chacun (barycentre)
- chercheBar(j : JSON) : JSON
  ➔ Demande à Google des bars près de la localisation donnée et ouverts à l'horaire donnée par l'utilisateur. Renvoie une liste de bar avec leur localisation que l'on utilisera pour chercher un restaurant.
  Cette méthode se chargera aussi de prendre le premier bar de la liste pour le proposer à l'utilisateur
- demandeValidationBar(s : String) : Boolean
  ➔ Une fois le bar trouvé, on demande à l'utilisateur si ça lui convient
  Si oui : on continue en cherchant un restaurant
  Si non : incrémente CompteurRecherche et relance chercheBar()
- chercheRestaurant(j : JSON) : JSON
  ➔ Procède de la même façon que chercheBar()
- demandeValidationResto(s : String) : Boolean
  ➔ Procède de la même façon que demandeValidationBar()
- chercheBoite(j : JSON) : JSON
  ➔ Procède de la même façon que chercheBar()
- demandeValidationBoite(s : String) : Boolean
  ➔ Procède de la même façon que demandeValidationBar()
- calculTrajetDeuxPoints(j : JSON) : JSON
  ➔ Calcule le temps de trajet entre deux points, avec l'API Google Matrix

Tests

Cycle en V

À l'issu de chaque phase d'implémentation, nous allons tester notre application de façon à se tenir au cycle en V suivant :

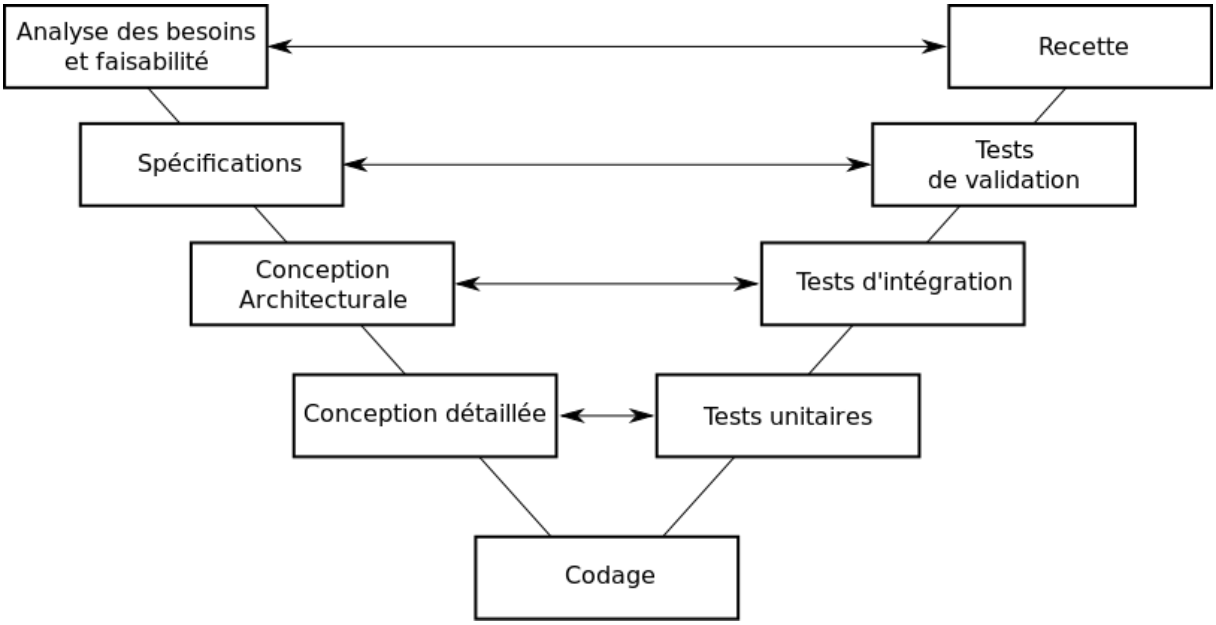


Figure 1

Tests unitaires

Les tests unitaires sont très importants pour vérifier si nos méthodes fonctionnent parfaitement bien. Nous avons l'obligation de tester chaque méthode de façon à ce que notre programme ne présente aucun bug. Ceci concerne la structure interne de notre code.

Quelques exemples de test de nos méthodes pourraient être :

- Vérifier le bon format JSON que nous envoyons à Google
- Vérifier que Google renvoie bien quelque chose ou que dans le cas où il ne renvoie rien, c'est le choix de l'utilisateur
- Vérifier que les adresses existent bien
- Vérifier que le compteur de recherche se réinitialise bien à chaque recherche d'un nouvel établissement

Les autres tests seront aussi indispensables.

Conclusion

Contraintes de conception

Concernant l'interface graphique, notre choix s'est dirigé vers une interface simple et efficace. Nous veillerons à respecter les contraintes d'entrées de l'utilisateur, c'est-à-dire les adresses, la date et les préférences alimentaires. Nous avons aussi choisi de proposer à l'utilisateur d'entrer un horaire de début pour proposer directement les établissements ouverts à ce moment-là et pas proche de fermeture (sinon il n'y aurait aucun intérêt d'y aller). Nous avons aussi décidé de faire un menu déroulant pour le mode de transport car nous pensons qu'un choix de lieu doit se faire aussi en fonction du temps de trajet que nous pourrons calculer facilement avec l'API Google Matrix Distance.
Nous veillerons à satisfaire notre client, c'est pourquoi nous proposons un bouton de validation après chaque recherche d'un établissement. De cette manière, le triplet final proposé est alors certain de plaire à l'utilisateur qui a fait la recherche.

Contraintes de développement

Nous avons décidé de développer notre application en JAVA car ce langage nous permet de manipuler des objets facilement. Ici, nous avons décidé de faire de notre utilisateur un objet. C'est aussi le langage que nous apprécions tous les trois sachant qu'on a décidé de réaliser notre application sur un PC et non une application mobile. Les contraintes d'intégration seront plus faciles.
Concernant le choix de nos librairies, la manipulation d'ArrayList est nettement plus facile d'utilisation que d'une liste basique. Les tests unitaires se feront à l'aide de JUnit, une librairie que nous apprécions fortement pour le développement en JAVA.

Notre but est alors de répondre à ces contraintes afin de satisfaire notre client.