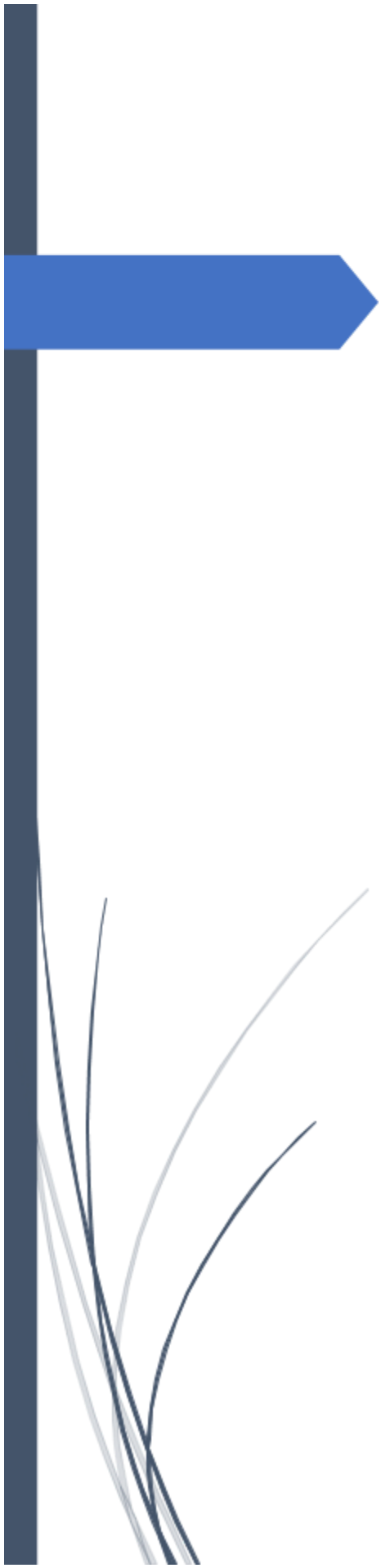

Cahier de conception

Plan

Table des figures

I. Conception générale

I.1. Introduction

Après avoir tracé les grandes lignes de phase de spécification de besoins, mettons l'accent maintenant sur une phase fondamentale dans le cycle de vie d'un logiciel, la phase de conception. Cette phase a pour objectif de déduire la spécification de l'architecture de système.

Cette phase aboutira à la conception et la représentation des diagrammes de séquences et d'activités en se basant sur le langage de modélisation UML.

Nous avons 4 classes : Interface, Adresse, Traitement, GoogleAPI

- Interface : Classe implémentant l'interface de l'application. Donc tous les affichages.
- Adresse : Classe implémentant la structure d'une adresse postale.
- Traitement : Classe qui récupérera le triplet d'adresse généré par Google API puis effectuera les traitements en fonction de ces derniers.
- GoogleAPI : Classe qui nous permettra d'utiliser les APIs Google, on ne fera que l'utiliser, donc pas de l'implémenter.

On utilisera JSON en raison de sa facilité d'implémentation. JSON représentant des objets sous forme de caractères, donc une fois les données encodées en format JSON puis envoyées au système Android, ceci permettra à notre application d'obtenir ces données codées, les analyser et de les afficher.

On utilisera Java comme langage de programmation.

I.2. Environnement et outils de développement

Nous allons utiliser comme matériel de développement nos propres machines et d'une quantité suffisante de RAM (minimum 4Go) pour avoir une qualité de développement acceptable, en partie en raison des nombreux services à exécuter notamment l'IDE.

On utilisera **Eclipse** comme outil principale de développement et aussi **Android Studio** comme environnement de développement pour notre application android.

On utilisera donc des API Google Maps pour offrir aux utilisateurs bon moyen de localisation, ajouter une carte utilisant des données Google Maps à notre application en fonction des adresses proposées à l'utilisateur.

I.3. Cycle de développement en v

Pour la conception, le développement et la réalisation de notre application, nous avons opté pour l'application du processus de développement V qui demeure actuellement le cycle de vie le plus connu et certainement le plus convenable aux projets complexes.

Ce processus nous a accompagné depuis le début et nous accompagnera jusqu'à l'implémentation. Son principe est qu'avec toute décomposition, doit être décrite la recomposition, et que toute description d'un composant doit être accompagnée de test qui permettront de s'assurer qu'il correspond à sa description. Ceci rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du l'application) et on sait progressivement si on s'approche de ce que le client désire.

On est bien évidemment conscient que le cycle en V n'est pas toujours adapté à un développement logiciel, puisqu'il est difficile voire impossible de totalement détacher la phase de conception d'un projet de sa phase de réalisation. On se rend compte souvent au cours de la mise en œuvre que les spécifications initiales étaient incomplètes, fausses, ou irréalisables, sans compter les ajouts de nouvelles fonctionnalités par les clients. D'autres modèles permettent plus facilement des modifications (parfois radicales) de la conception initiale, donc parmi eux on a la **méthode agile**.

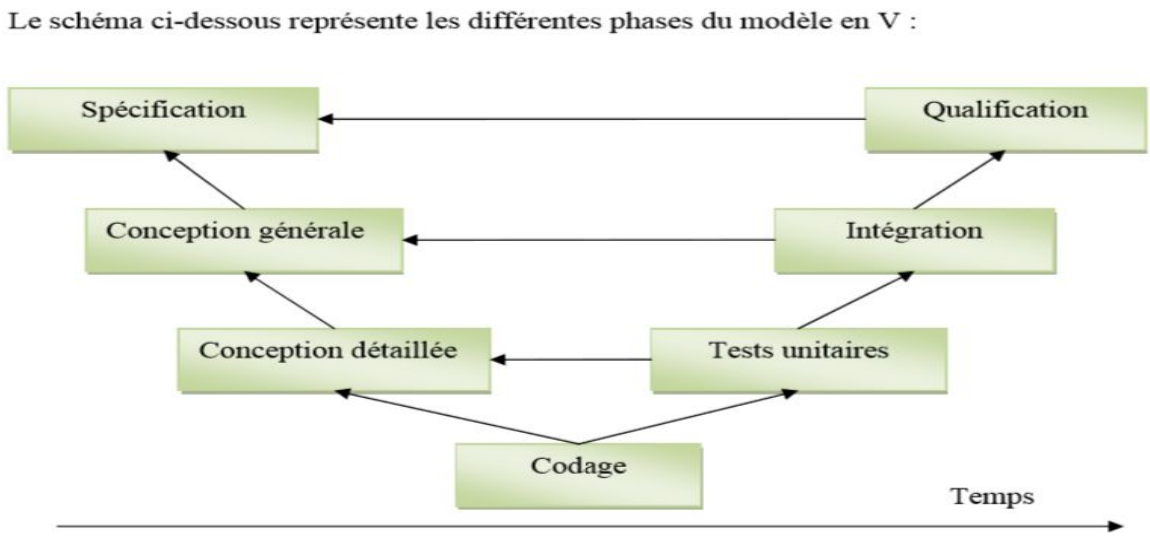

Figure I : Modèle en V

II. Conception détaillée

III.1 Diagramme de classes

Une fois toutes les informations saisies et la soumission de ces dernières faite par l'utilisateur, l'interface se connectera à GoogleAPI pour lui fournir les informations de l'utilisateur en appliquant sa politique de recherche d'adresses par la méthode chercherDes(), et GoogleAPI se servira de ses informations en tenant compte de cette politique puis générer un triplet d'adresses si toutes les conditions sont réunies.

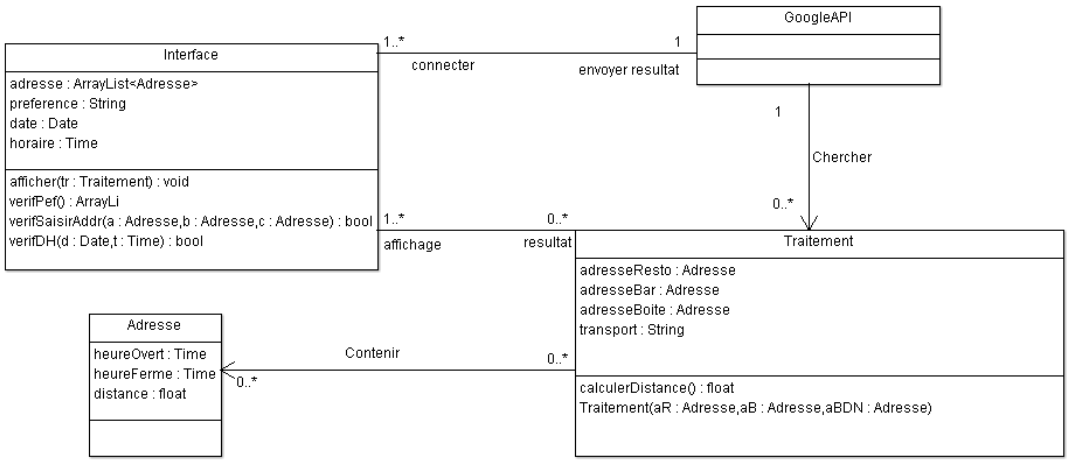

Figure III-1 : Diagramme de classes

1) Dans ce diagramme, il y a 4 classes principales : Interface, Adresse, Traitement, GoogleAPI.

   a) Interface : représent l'interface de l'application. Sur cet interface, on utilise les attributs pour la représentation des champs différents :
   - adresse: un ArrayList de type Adresse qui indique les adresses des clients.
   - préférence : un String représente les préférences d'un client.
   - date, horaire : ces attributs représentent la date et heure où le client souhaite de sortir.

   Dans la classe **Interface** se trouvent les méthodes ci-dessous:
   - afficher: cette méthode prend en argument un objet de type Traitement et affiche les résultats de la recherche de trajet d'un client.

   - verifSaisirAddr: cette méthode prend en argument 3 adresses (une adresse d'un bar, d'un restaurant et d'une boite de nuit) et vérifie si les adresses sont bien saisis par le client.

   - verifDH: vérifie la date et l'horaire de sortir, si la date (l'horaire) est déjà passée, revoie faux.
   - verifPref: permet de grouper toutes les préférences d'un client dans un ArrayList.

   b) Traitement: une classe qui initialise le triplet d'adresse généré par la classe GoogleAPI. Dans cette classe, on représente les aspects d'une adresse par les attributs:
   - adresseResto/Bar/Boite : l'adresse d'un restaurant (d'un bar, d'une boite de nuite).
   - transport : un String qui réprésente le moyen de transport.

   On utilise dans cette classe les méthodes:
   - calculerDis : calcule la distance entre le client et la destination qu'il a choisi.
   - La constructeur de la classe: initialise les attributs de la classe.

   c) Adresse : classe implémentant la structure d'une adresse postale qui contient la distance d'une adresse au client et les horaire d'ouverture/fermeture.

III.2 Diagramme de cas d'utilisation

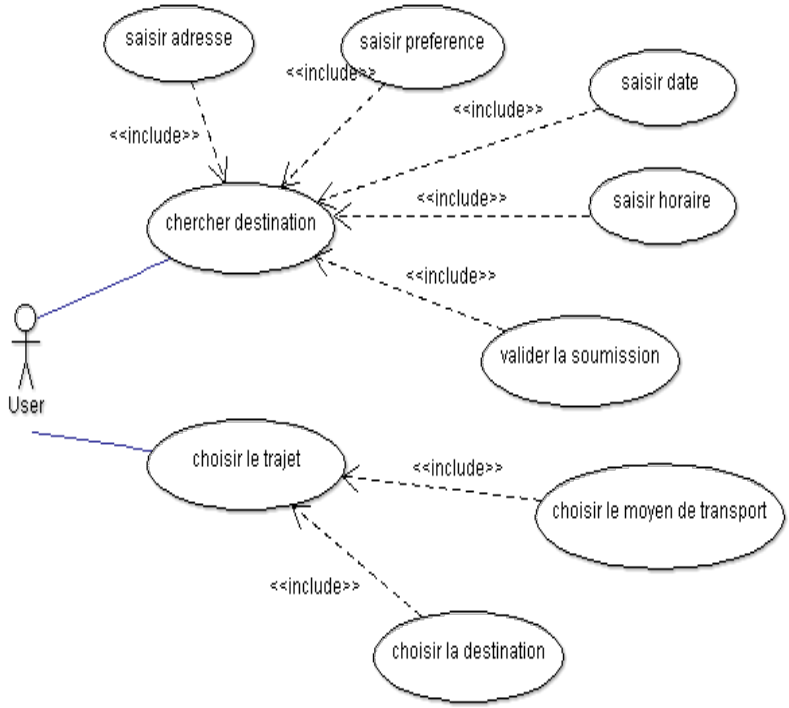

Figure III-2 : Diagramme de cas d'utilisation

Identification des acteurs

Acteurs	Rôles
Utilisateur	- Chercher les destinations : + Saisir les informations. + Valider la soumission. - Choisir le trajet : + Choisir les destinations. + Choisir le moyen de transport.

GoogleAPI	Fournir un triplet d'adresse

III.3 Diagramme de séquence

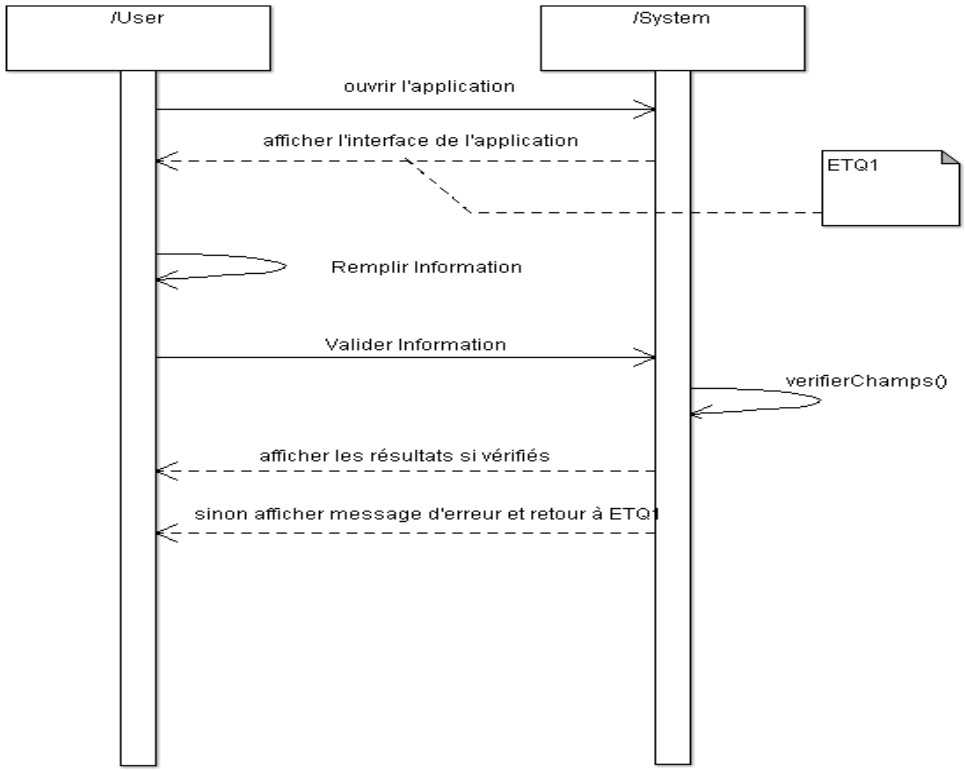

Figure III-3.1 : Diagramme de séquence : Recherche destination

Par système ici on parle de l'interface.

- L'utilisateur ouvre l'application
- Affichage de l'interface d'accueil
- Remplissage des informations par l'utilisateur et leur validation par ce dernier
- Vérification des champs de saisie
- Si tout est ok, affichage des résultats
- Sinon, affiche d'un message d'erreur et retour aux champs de saisie

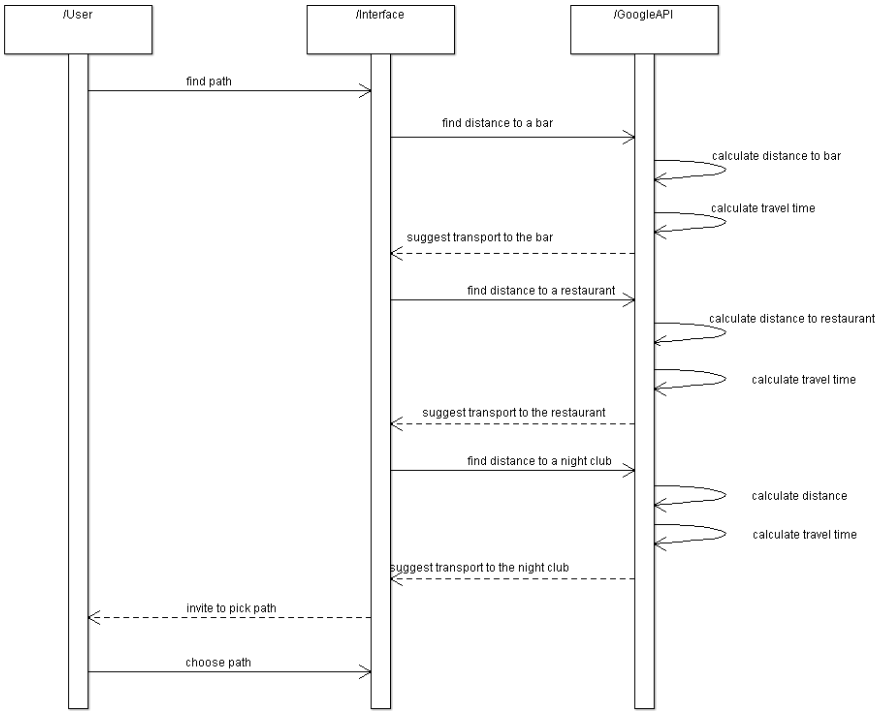

Figure III-3.2 : Diagramme de séquence : Choix du trajet

Une fois le triplet d'adresse fournit par Google API et son affichage sur l'interface, l'utilisateur peut maintenant voir les différentes adresses ainsi les moyens de s'y rendre.

Il verra également en plus des adresses les horaires d'ouverture et fermeture de chaque lieu, ce qui lui permettra de bien gérer son temps durant la sortie.

III. Description des interfaces

III.1. Scénario maquette

Lors de lancement de notre application, une interface de chargement s'affiche contenant son nom.

Après quelques secondes le menu principal s'affiche, disposant d'une page d'accueil de 5 champs : Adresses, Préférences alimentaires, Horaire (heure à laquelle l'utilisateur veut sortir), Date (la date de sortie), Soumettre (bouton de soumission).

Adresse 1, 2, 3 correspondent aux champs de saisie des adresses par l'utilisateur. On est parti de l'idée où la personne qui utilise l'application n'est pas la seule à vouloir sortir, qu'elle pourrait donc être accompagnée par des amis.

Adresse 1 peut donc être le champ de saisie de son adresse à elle, 2 et 3 ceux de ces amis, mais on peut saisir qu'au maximum trois adresses et au minimum une. Lorsque que l'utilisateur saisie plus d'une adresse, le choix du triplet d'adresse (bar, resto, boite) dépendra donc des trois adresses.

Donc on sélectionnera les adresses les plus proches possible de ces trois.

Horaire et date correspondent à l'heure et la date à laquelle l'utilisateur veut sortir.

Pour le champs Préférences, l'utilisateur peut faire un ou plusieurs choix, donc pour choisir il peut cocher un ou plusieurs champs parmi la liste qui lui est proposée. Après avoir cocher les cases qui l'intéressent et remplit les champs qui suit les préférences, puis valider sa soumission.

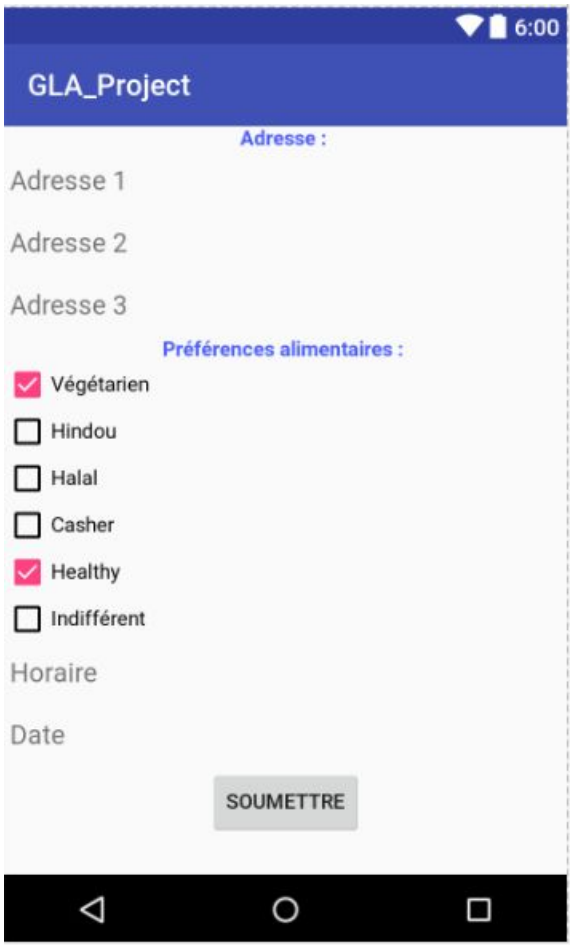

Figure II-1 : Page d'accueil

Nous avons aussi décidé de ne pas rendre obligatoire la saisie de tous les champs adresses en tenant compte du cas où ne nombre de personne à sortir n'atteint pas forcément trois. Mais néanmoins il faut saisir au moins une adresse afin de pouvoir déterminer le triplet d'adresses (Bar, Resto, Boite) dont l'utilisateur a besoin pour s'y rendre.

Donc après remplissage des champs de cette page et soumission des données, on traite ces dernières et afficherons le résultat de traitement sur une nouvelle page.

On affichera pour chaque adresse l'heure d'ouverture et de fermeture, ainsi les moyens de transport pour s'y rendre.

Figure II-2 : Page d'affichage de résultats

IV. Description des tests

Les tests servent à déterminer le comportement de l'application en fonction de ce qu'on a saisi et ensuite corriger les erreurs si ces dernières existent.

Le premier test consiste à vérifier le bon fonctionnement de l'application en saisissant les données attendues par l'utilisateur, ce test devra donner un triplet d'adresse.

Le deuxième test ressemble au premier sauf que l'on tombera sur un résultat inattendu, autrement dit l'utilisateur a bien saisi ce que lui est demandé mais n'obtient pas ce qu'il voulait, dans ce cas il s'agit surement d'une erreur indéfinie, on affichera donc un message signalant à l'utilisateur de saisir à nouveau ses informations mais cette fois ci en changeant certaines informations telles que l'horaire de sortie ou la date de sortie.

Le troisième test provoque un disfonctionnement à cause d'un manque d'une

entrée attendue par l'application, s'il s'agit d'une absence d'adresse elle va en demander une, vue qu'il est nécessaire d'avoir au moins une adresse en entrée pour fonctionner, s'il s'agit d'une absence de préférences alimentaire qui sera sous forme de plusieurs cases à cocher l'application va considérer que l'utilisateur n'a pas de choix précis et lui donnera quand même l'adresse d'un resto, s'il s'agit d'un manque de date ou d'horaire , l'application va choisir la date d'aujourd'hui et l'heure de la saisie .

Le dernier test s'assure de l'affichage des trois adresses, le cas où l'application ne trouve pas de résultat, un message d'erreur sera affiché en fonction de la variable qui a pour but de stocker lors du fonctionnement de l'algorithme la cause (elle pourrait être à cause : d'une mauvaise date ou horaire, des préférences alimentaires, etc.) qui l'a empêché de ne pas trouver d'adresses, ou tout simplement les informations saisies ne permettent de fournir aucun résultant.

V. Contraintes

1. **Saisie de date et horaires** :

Seule les dates et horaires ultérieures à celles du moment de saisie sont autorisées ; Il faut tenir compte de la comparaison entre 23 :59 du jour même et 00 :00 du jour d'avant. Adresse : une adresse valide doit être composée d'un numéro de la voie, d'un nom de rue, d'un code postal et d'un nom de commune, et un nom du pays.

2. **La position actuelle de l'utilisateur** (la saisie de l'adresse ne suffit pas)
3. **Calcul de prévision :**

Le temps que prend une sortie = distance entre l'adresse de saisie et l'adresse d'arrivée / vitesse de déplacement selon le transport proposé ; Le transport est proposé par rapport à la distance à parcourir, > 8 km = voiture, > 2,5 km et < 8 km = vélo, < 2,5km = à pied ; Il faut calculer trois temps de prévision séparément, un avant d'arriver au bar, le deuxième entre le bar et le restaurant, le troisième entre le restaurant et la boite de nuit.

4. **Préférences alimentaires :**

Doit être sélectionné dans les choix proposés.

5. **Validation :**

Seule l'interface où tous les champs nécessaires remplis sont autorisés à être validée.

6. **Affichage des adresses de résultats** dans l'ordre chronologiques en indiquant les horaires d'ouvertures

VI. Conclusion

Le cahier de conception nous permet, dans la suite du cahier de charge, de décrire plus précisément les technologies employées et la manière dont les approches algorithmiques sont abordées. Pour ce fait, nous avons présenté le modèle en V, puis les fonctionnalités que nous envisageons de développer sous forme des différents diagrammes dont l'interface, diagramme de classes, diagrammes de cas d'utilisation, et enfin diagramme de séquence. Le cahier de conception agit comme un document nécessaire au maintien du développement logiciel. Afin de remplir cet objectif, nous avons décrit en détail les scénarios possibles afin que les utilisateurs puissent appréhender les scènes d'utilisation à prévoir. Ce document permet aux futurs développeurs d'aborder un projet en marche, sans toucher l'implémentation. Les architecture technique et fonctionnelle tel que les choix d'IDE, de langages et librairies et enfin le fonctionnement des tests, permet aux programmeurs de mieux identifier les contraintes et failles possibles en s'en servant de références, lors du futur maintenance et d'éventuellement amélioration.