RAPPORT DE CONCEPTION

Ruiwen WANG & Chenwen XIONG & Ying WANG Groupe 2

Table des matières

I.Introduction

Dans le cadre de l'unité d'enseignement 'projet' du License informatique L3 à l'université Paris Sud, il nous ait demandé de travailler sur un projet de développement de système de navigation pour guider le conducteur d'un véhicule.

Dans le cadre de cette unité d'enseignement, nous devons assumer la gestion d'un projet de développement d'un système de navigation. Dans un premier temps, ce système est destiné à guider le conducteur entre un lieu de départ et un lieu arrivé, permettant de choisir sa préférence sur le trajet à suivre (le plus court, le plus rapide...etc.). Dans un second temps, le système devra pouvoir être utilisé par l'ensemble des composantes de l'université. Les attendees d'un tel projet sont donc importantes.

Ce document décrit le contexte, les besoins fonctionnels et les objectifs du projet.

Un premier découpage des étapes nécessaires à la réalisation d'un tel projet donne lieu dans de document à un planning prévisionnel. Ce document a pour finalités de définir le projet de manière simple et détaillée et de définir

Les objectifs auxquels devra répondre une future spécification technique.

Notre LOGO

Forme : La flèche
Couleur : Rouge
Titre : WEGO

On choisit la forme de la flèche pour la signification de “Navigation” de cette application.
On choisit la couleur de rouge pour frapper les yeux et attirer les regards des utilisateurs. La rouge aussi représente notre passion de travailler dans le projet.
On décide le titre de l’application comme “WEGO” qui aussi représente la sens de navigation et l’association avec les utilisateurs.

II. Les Langages de Développer

1. UML
    a. Diagramme de classe
    b. Diagramme de séquence
2. SQL (Diagramme d'analyse entre les classes concernantes)
3. Pseudo-code de Algorithme
4. MOAL
5. HTML(La réalisation finale de cette application)
6. php
7. JavaScript

Nous avons décidé de mettre en œuvre ce logiciel sous la forme d'une page Web.

Les avantages sont également faciles à utiliser côté ordinateur et côté téléphone portable, nous allons utiliser php pour réaliser la fonction principale.Utilisez SQL pour stocker la plupart des données.

II.1 HTML (Hyper Text Markup Language)

Le HTML est un langage dit de « marquage » (de « structuration » ou de « balisage ») dont le rôle est de formaliser l'écriture d'un document avec des balises de formatage. Les balises permettent d'indiquer la façon dont doit être présenté le document et les liens qu'il établit avec d'autres documents. Le langage HTML permet notamment la lecture de documents

sur Internet à partir de machines différentes, grâce au protocole HTTP, permettant d'accéder via le réseau à des documents repérés par une adresse unique, appelée URL.

II.2 PHP :

PHP Générateur pour MySQL est un outil pour créer des applications Web orientée base de données visuellement. Il nous permet de générer des scripts PHP de haute qualité pour travailler avec des tables MySQL, vues et les requêtes à travers le web. Vous devez pas avoir de connaissances en programmation pour l'utiliser

II.3 Relation MYSQL/PHP

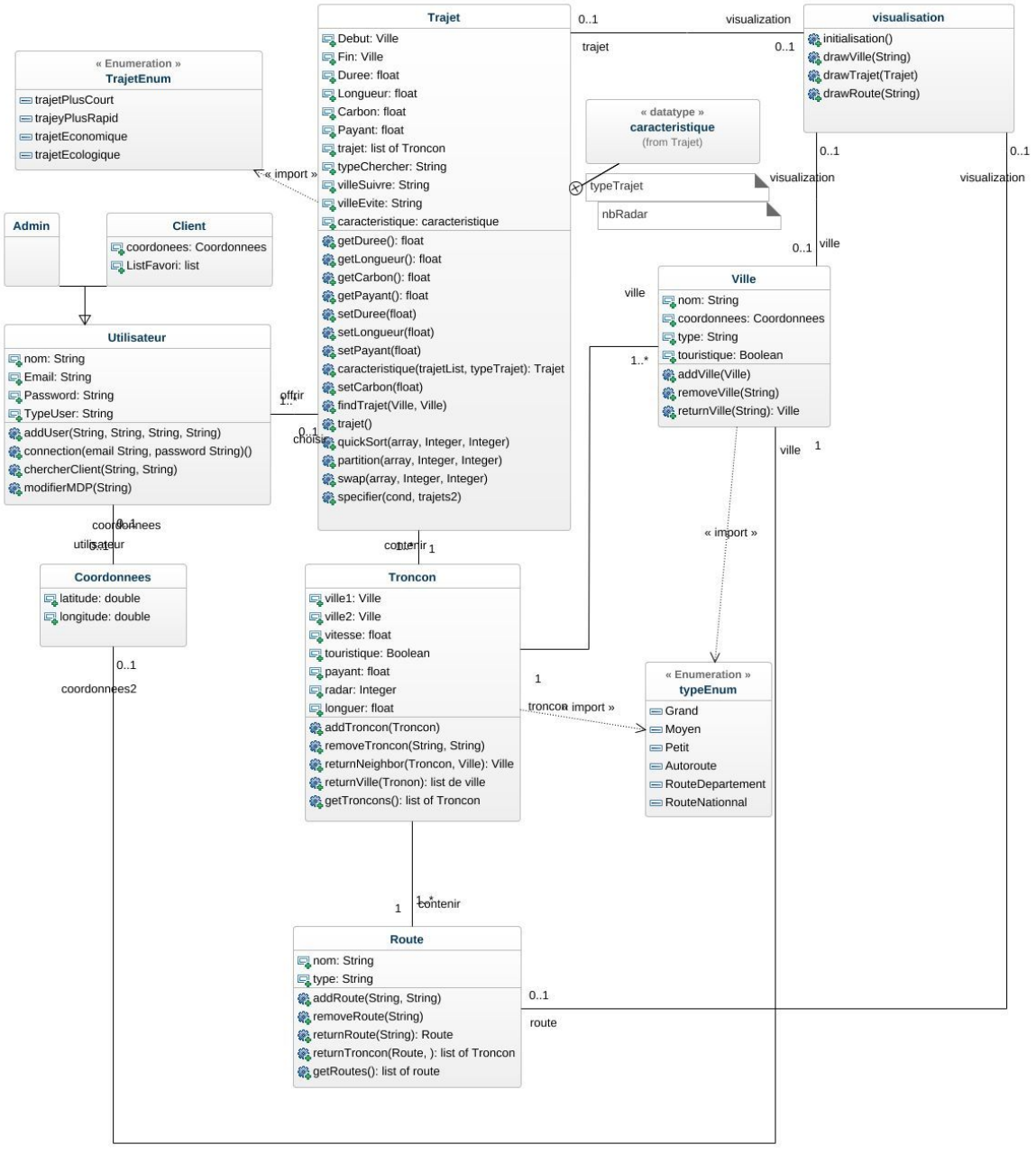
Le couple PHP/MySQL est très utilisé par les sites web et proposé par la majorité des hébergeurs Web. Plus de la moitié des sites Web fonctionnent sous Apache, qui est le plus souvent utilisé conjointement avec PHP et MySQL.

III.4JavaScript

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs2 avec l'utilisation (par exemple) de Node.js3. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais

qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés. En outre, les fonctions sont des objets de première classe. Le langage supporte le paradigme objet, impératif et fonctionnel.

III. Implémenter les diagrammes de classe



1)classe Utilisateur ,

Attribute:

Email: email d'une client , unique par chaque différent client

idUser: tous les utilisateur ont leur propre unique nom identifient eux

ListFavori: List des positions qu'une client favori

TypeUser : on considère qu'il contient deux types d'utilisateur (soit admin , soit client), évidement le permission des deux types d'utilisateurs sont différent.

Table:

utilisateurs

Fonction:

checkUser(String nom,String email):

for u in utilisateurs:

if u.nom == nom or u.email == email

then return false

else return true

/** vérifier le nom, email des utilisateurs sont unique **/

addUtilisateur(String nom,String pw,String email,String tc):

if checkClient(nom,pw,email,tc)

sql_query("insert into utilisateur(nom,email,typeUser,password)values(nom,email, tc,pw)");

else retrun 0

connnection(string email,string password):

users = select* from user(sql)

for u in users :

    if email = u.email then

        if password = u.password then return id_user

        else loginfail()

    else usernotfound()

MOAL

definition $inv_{UtilisateurNom} \equiv \forall u \in Utilsateur.\ u.Nom != NULL$

definition $inv_{UtilisateurEmail} \equiv \forall u \in Utilsateur.\ u.Email != NULL$

definition $inv_{UtilisateurTypeUser} \equiv \forall u \in Utilsateur.\ u.TypeUser != NULL$

definition $inv_{UtilisateurPassword} \equiv \forall u \in Utilsateur.\ u.Password != NULL$

definition $\mathrm{pre}_{\mathrm{AddUser}}$(u : Utilisateur) ≡

∀ u ∉ Utilisateur. u.TypeUser ≠ ‘Client’

definition $\mathrm{post}_{\mathrm{AddUser}}$(u : Utilisateur ,name: String ,mail: String,type: String,pass: String) ≡

type = ‘Client’ ∧

c ∈ new(Client) ∧

c.nom = name ∧

c.Email = mail ∧

c.TypeUser = type∧

c.PassWord = pass

definition $\mathrm{pre}_{\mathrm{connection}}$(u : Utilisateur) ≡

∀ u ∉ Utilisateur. u.TypeUser = ‘Client’

definition $\mathrm{post}_{\mathrm{connection}}$(u : Utilisateur ,mail: String,pass: String) ≡

type = ‘Client’ ∧

c ∈ new(Client) ∧

c.Email = mail ⋀

c.PassWord = pass

2)classe Trajet

Attribute:

villeSuivre : Ville à suivre dans le trajet avec le type de String

villeEvite : Ville à éviter dans le trajet avec le type de String

typeChercher : Le type de ville à demander dans le trajet avec le type de

String

Debut: La ville débute de chaque trajet avec le type de Position

Fin : La ville fin de chaque trajet avec le type de Position

Duree : Le temps estimé pour un trajet

Longueur : La longueur total du trajet

Carbon: Le consommation de carbon pendant ce trajet

Payant: Le tarif doit payer pendant ce trajet

Trajet: List de tronçon

Table:

Trajets

Fonction:

```
checkTrajet(Position d, Position f):
        for t in trajets:
if t.debut == d and t.fin == t.f
        then return true
        else return false

getDuree():
        return this.Duree

getLongueur():
        return this.Longueur

getCarbon():
return this.carbon

getPayant():
        return this.payant

/**
PlusCourt(array of trajet trajets):
        $ts = quickSort.quickSortLongueur(trajets,0,length-1);
        return array($s[0],$s[1],$s[2]);



economique(array of trajet trajets)
```

```
$ts = quickSort.quickSortPayant(trajets,0,length-1);
        return array($s[0],$s[1],$s[2]);



ecologique(array of trajet trajets)
$ts = quickSort.quickSortCarbon(trajets,0,length-1);
        return array($s[0],$s[1],$s[2]);

rapide(array of trajet trajets)
$ts = quickSort.quickSortDuree(trajets,0,length-1);
        return array($s[0],$s[1],$s[2]);
**/
/** En utilisant l'interface quickSort ,on peut obtenir un tableau bien tiée
avec différent attribute (durée,longueur,carbon,payant)
*//

setDuree(float d):
        this.Dureer = d

setLongueur(float l):
        this.Longueur = l

setPayant(float p):
        this.Payant = p

setCarbon(float c):
        this.Carbon = c
```

```
findTrajet(ville debut,ville fin):
$trajets = new array of trajet();
$trajet = new trajet();
findMyway(ville debut,ville fin,array trajets,*in trajet trajet) :
if ville debut == ville fin then
        for t in $trajet :
                        $duree = t.Longueur / t.vitesse +$duree
                        $longueur = t.longeueur + $longueur
                        $payant = t.payang + $payant
                        $carbon = t.Carbon + $carbon
        setDuree(duree);
        setLongueur(float l)
        setPayant(float p)
      setCarbon(float c)
        add $trajet in $trajets

        for t in troncon.getTroncons():
                if t not in $trajet
                        then if ville debut == t.ville1 or ville debut == t.ville2
                                then   add t in $trajet
                        findMyway(neighbor(t,ville debut),ville fin,trajets,trajet)

findMyway(ville debut,ville fin,$trajets,$trajet)
return $trajets
```

/** Algorithme

on définit ville début et ville fin , initialisant une list de trajet vide ,puis récursivement trouver un tronçon liée de ville début , après on considèrent

l'autre ville de ce tronçon comme un ville début ,jusqu'à on trouve une ville début est la ville fin ,ces trajets sont voilà des chemin entre ville début et ville fin

**/

caracteristique(array of trajet trajets)

this.typeChercher = getElement from HTML(String typeChercher)

this.villeSuivre = getElement from HTML(String villeSuivre)

this.villeEvite = getElement from HTML(String villeEvite)

$result = new array of trajet

for ts in trajets :

for tj in ts:

for t in tj :

if t == this.villeSuivre then add tj to $result

for ts in $result :

for tj in ts:

for t in tj :

if t == this.villeEvite then remove tj from $result

return $result

Lancer(arrayofTrajet arr)

visualisation.visualiserChoix(arr) ;

/** lancer la fênetre des choix parmis des trois trajets **/

$m = getClickMouse() ;

/**obtenir la choix**/

visualisation.visualiserTrajet($m) ;

/**visualiser le trajet choisi sur la carte **/

visualisation.fênetreIndication();

trajet():

this.Duree = 0

this.Payant = 0

this.Longueur = 0

this.Carbon = 0

this.trajet = new list of troncon();

MOAL

definition $inv_{TrajetDebut}$ ≡ ∀ t ∈ Trajet. t.Debut != NULL

definition $inv_{Trajettypechercher}$ ≡ ∀ t ∈ Trajet. t.typeChercher != NULL

definition $inv_{TrajetVilleSuivre}$ ≡ ∀ t ∈ Trajet. t.villeSuivre != NULL

definition $inv_{TrajetVilleEvite}$ ≡ ∀ t ∈ Trajet. t.villeEvite != NULL

definition $inv_{TrajetFin}$ ≡ ∀ t ∈ Trajet. t.Fin != NULL

definition $inv_{TrajetLongueur}$ ≡ ∀ t ∈ Trajet. t.Longueur > 0

definition $inv_{TrajetDuree}$ ≡ ∀ t ∈ Trajet. t.Duree > 0

definition $pre_{getDuree}$(t : Trajet, id : Integer) ≡

∀ t ∈ Trajet. t.idTrajet = id

definition $post_{getDuree}$(t : Trajet,result) ≡

result = t.Duree

definition $pre_{getLongueur}$(t : Trajet, id : Integer) ≡

∀ t ∈ Trajet. t.idTrajet = id

definition $post_{getLongueur}$(t : Trajet,result) ≡

result = t.Longueur

definition $pre_{getCarbon}$(t : Trajet, id : Integer) ≡

∀ t ∈ Trajet. t.idTrajet = id

definition $post_{getCarbon}$(t : Trajet,result) ≡

result = t.Carbon

definition $pre_{getPayant}$(t : Trajet, id : Integer) ≡

∀ t ∈ Trajet. t.idTrajet = id

definition $post_{getPayant}$(t : Trajet,result) ≡

result = t.Carbon

definition $pre_{findTrajet}$(v1:ville,v2:ville) ≡

∀ v1,v2 ∈ Trajet.

definition $post_{findTrajet}$(v1:ville,v2:ville) ≡

∀ t ∈ Trajet. (t.debut = v1 ∧t.fin = v2) v (t.debut = v2 ∧ t.fin = v1)

3) classe Troncon

Attribute:

ville1 et ville2 sont deux villes liée par un tronçon

vitesse : La limite maximale de vitesse pour chaque conduite automobile dans ce tronçon (type int)

touristique: Ce tronçon est touristique ou pas (type boolean)

payant: le montant que l'utilisateur a besoin de payer dans le conduite en ce tronçon (type int)

radar: nombre de radar (type int)

longueur: La longueur de ce tronçon (type int)

Table:

Tronçons

Fonction:

getTroncons():

$troncons = mysql_query(select * from troncon)

return $troncons

checkTroncon(Ville v,Ville vv, int vm, bool t, int pay, int rad, int l):

let Troncons = getTroncons();

for t in Troncons:

if t.ville1 == v and t.ville == vv and t.vitesse == vm and t.touristique == t and t.payant == pay and t.radar == rad and t.longueur == l

then return true

else return false

/** vérifier si ce troncon est déja existe.**/

/** opération uniquement par l'admin **/

addTroncon(Ville v,Ville vv, int vm, bool t, int pay, int rad, int l):

if !checkTroncon(v,vv,vm,t,pay,rad,l)

sql_query("insert into troncon(ville1,ville2,vitesse,touristique,payant,radar,longueur)values('v', 'vv', vm,t,pay,rad,l)");

else return 0

/** ajouter un nouveau troncon dans la table de troncon **/

/** opération uniquement par l'admin **/

remove(string ville1,string ville2):

sql_query("drop select * from troncon where troncon.ville1= ville1 and troncon.ville2 = ville2 or troncon.ville1 = ville2 and troncon.ville2 = ville1");

/** enlever un troncon correspond à ville1 et ville2 dans la table de troncon **/

/** opération uniquement par l'admin **/

returnVille(Troncon tr):

return [tr.ville1,tr.ville2]

returnNeighbor(troncon t,Ville ville):

if $t.ville1 == ville then return ville2

else if $t.ville2 == ville then return ville1

/** return l'autre ville liée par ce troncon **/

MOAL

definition $inv_{TronconVille} \equiv \forall t \in Troncon.\ t.Ville \neq NULL$

definition $inv_{TronconVille2} \equiv$

$\forall t \in Troncon.\ t.Ville2 \neq NULL \wedge$

t.Ville2 ≠ t.Ville

$\text{definition inv}_{\text{TronconPayant}}$ ≡ ∀ t ∈ Troncon. t.payant >= 0

$\text{definition inv}_{\text{TronconRadar}}$ ≡ ∀ t ∈ Troncon. t.radar >= 0

$\text{definition inv}_{\text{TronconLongueur}}$ ≡ ∀ t ∈ Troncon. t.Longueur > 0

$\text{definition inv}_{\text{TronconVitesse}}$ ≡ ∀ t ∈ Troncon. t.Vitesse > 0

$\text{definition pre}_{\text{AddTroncon}}$(t : Troncon, v : Ville, vv : Ville) ≡

∀ t ∉ Trajet. t.ville ≠ v ∧ t.villle2 ≠ vv

$\text{definition post}_{\text{AddTroncon}}$(u:Utilisateur, t : Troncon , v : Ville, vv : Ville, vm: Integer, tour : boolean, pay : integer, rad : integer, long : integer) ≡

∀t ∈ Troncon. t.ville = v ∧

t.ville2 = vv ∧

t.vitesse = vm ∧

t.touristique = tour ∧

t.payant = pay ∧

t.radar = rad ∧

t.longueur = long ∧

t ∈ new(Troncon) ∧

modifiesOnly({ u ∈ Utilisateur | u.typeUser = ‘Admin’})

$\text{definition pre}_{\text{ReturnVille}}$(t : troncon, v : Ville, vv:Ville) ≡

$\forall$ t $\in$ Trajet. t.ville = v $\wedge$ t.villle2 = vv

definition $post_{ReturnVille}$(t: troncon, result)
result = (t.ville)$\cup$(t.ville2)

definition $pre_{removeTroncon}$(t : troncon, v : String, vv:String) $\equiv$
$\forall$ t $\in$ Trajet. t.ville = v $\wedge$ t.villle2 = vv

definition $post_{removeTroncon}$(t: troncon, result)
result = $\forall$ t $\in$ Trajet. t = NULL $\wedge$
modifiesOnly({ u $\in$ Utilisateur | u.typeUser = 'Admin'})

4) Classe Route

Table : Routes

Attributs:
nom : Le nom de chaque route avec type de String
type : Le type de chaque route avec type de String

Fonctions:
checkRoute(String n, String t):
for r in routes:
    if r.nom == n and r.type == t
    then return true
    else return false

addRoute(String n, String t):
if !checkRoute(n,t)
then
sql_query("insert into route(nom,type)values(n,t)");
else retrun 0

removeRoute(string nom):
sql_query("drop select * from route where route.nom = nom");

returnRoute(string nom):
$r = sql_query("select * from route where route.nom == nom")

return $r

getRoutes():
$routes = mysql_query(select * from route)
return $routes

MOAL

definition $inv_{RouteNom}$ ≡ ∀ r ∈ Route. r.Nom ≠ NULL

definition $inv_{RouteType}$ ≡ ∀ t ∈ Troncon. t.Type ≠ NULL

definition $pre_{AddRoute}$(r : Route, n : String, t : String) ≡
∀ r ∉ Route. r.nom ≠ n ∧ r.type ≠ t

definition $post_{AddRoute}$(u:Utilisateur,r: route, n:String,t:String) ≡
∀r ∈ Route. r.nom = n ∧
r.type = t
r ∈ new(Route) ∧
modifiesOnly({ u ∈ Utilisateur | u.typeUser = ‘Admin’})

definition $pre_{removeRoute}$(r : Route, n : String) ≡
∀ r ∉ Route. r.nom = n
definition $post_{removeRoute}$(u:Utilisateur,r: route, n:String) ≡
∀r ∈ Route. r = NULL ∧
modifiesOnly({ u ∈ Utilisateur | u.typeUser = ‘Admin’})

definition $pre_{ReturnRoute}$(r : route, n: String) ≡

∀ r ∈ Route. r.nom = n

definition $post_{ReturnRoute}$(r : route, result)

result = (r.nom)∪(r.type)

5.1) Classe Coordonnee

Attribut :

latitude : La latitude de chaque ville décide par sa position geographique avec le type de double

longtitude : La longitude de chaque ville décide par sa position geographique avec le type de double

Fonction

setCoordonnee(double la, double long):

this.latitude = la

this.longtitude = long

getCoordonnee():

return [this.latitude,this.longtitude]

MOAL

definition $inv_{Coordonnee}$ ≡ ∀ c∈ Coordonnee. c.latitude ≠ NULL ∧ c.longtitude ≠ NULL

definition $\mathrm{pre}_{\mathrm{SetCoordonnee}}$(c : Coordonnee, la : double, long: double) ≡

∀ c ∉ Coordonnee. c.latitude ≠ la ∧ c.longtitude ≠ long

definition $\mathrm{post}_{\mathrm{SetCoordonnee}}$(u: Utilisateur, c : Coordonnee, la : double, long: double) ≡

∀c ∈ Coordonnee. c.latitude = la ∧

c.longtitude = long ∧

modifiesOnly({ u ∈ Utilisateur | u.typeUser = ‘Admin’})

definition $\mathrm{pre}_{\mathrm{ReturnCoordonnee}}$(c : Coordonnee, la: double, long : double) ≡

∀ c ∈ Coordonnee. c.latitude = la ∧

c.longtitude = long

definition $\mathrm{post}_{\mathrm{ReturnCoordonnee}}$(c : Coordonnee, la: double, long : double) ≡

result = (c.latitude)∪(c.longtitude)

5.2) Classe Ville

Attributs:

nom : Le nom de chaque ville avec le type de String

coordoonees : Les coordonnees de chaque ville qui compose la latitude et la longitude de ce ville avec le type de (double,double)

type : la taille de cette ville avec le type de String

touristique: Vérifier que la ville est touristique ou pas en forme de boolean

Fonctions:

checkVille(String n, Coordonnee co, String t, Bool tour):

for r in routes:

if r.nom == n and r.type == t and r.coordonnees = co and r.type == t and r.touristique == tour

then return true

else return false

addVille(String n, Coordonnee co, String t, Bool tour):

sql_query("insert into troncon(nom,x,y,type,touristique,radar,longueur)values('v', 'vv', vm,t,pay,rad,l)");

removeVille(string nom)

sql (“drop select * from ville where ville.nom = nom”)

returnVille(string nom)
$v = sql_query(“select * from ville where ville.nom == nom”)
return $v

MOAL

definition $inv_{VilleNom}$ ≡ ∀ v∈ Ville. v.nom ≠ NULL

definition $inv_{VilleCoordonne}$ ≡ ∀ v∈ Ville. v.Coordonnee ≠ NULL

definition $inv_{VilleType}$ ≡ ∀ v∈ Ville. v.type ≠ NULL

definition $inv_{VilleTouristique}$ ≡ ∀ v∈ Ville. v.touristique ≠ NULL

definition $pre_{AddVille}$(v : Ville, n:String) ≡
∀ v ∉ Ville. v.nom ≠ n

definition $post_{AddVille}$(u: Utilisateur, v : Ville, n: String,
co:Coordonnee,t:String,tour:Touristique) ≡
∀v ∈ Ville. v.nom = n ∧

v.coordonnees = co ∧

v.type = t ∧

v.touristique = tour∧

v ∈ new(Ville)∧

modifiesOnly({ u ∈ Utilisateur | u.typeUser = ‘Admin’})

definition $pre_{RemoveVille}$(v : Ville, n:String) ≡

∀ v ∉ Ville. v.nom = n

definition $post_{RemoveVille}$(u: Utilisateur, v : Ville, n: String, co:Coordonnee,t:String,tour:Touristique) ≡

∀v ∈ Ville. v = NULL

modifiesOnly({ u ∈ Utilisateur | u.typeUser = ‘Admin’})

definition $pre_{ReturnVille}$(v : Ville, n: String) ≡

∀ v ∈ Ville. v.nom = n

definition $post_{ReturnVille}$ (v : Ville, result) ≡

result = (v.nom)∪(v.coordonnees)∪(v.type)∪(v.touristique)

6) Classe (Interface) Visualisation

Fonctions:

initialisation()

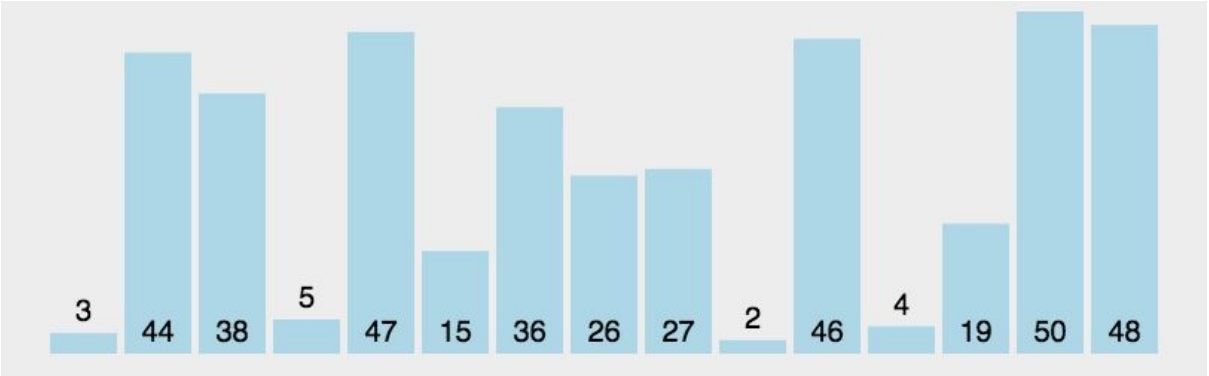
drawVille(String)

drawRoute(String)

drawTrajet(Trajet)

6) Interface quickSort

Algorithme:



En informatique, le tri rapide (en anglais quicksort) est un algorithme de triIl est généralement utilisé sur des tableaux, mais peut aussi être adapté aux listes. Dans le cas des tableaux, c'est un tri en place mais non stable.

La complexité moyenne du tri rapide pour n éléments est proportionnelle à n log n, ce qui est optimal pour un tri par comparaison, mais la complexité dans le pire des cas est quadratique. Malgré ce désavantage théorique, c'est en pratique un des tris les plus rapides, et donc un des plus utilisés. Le pire des cas est en effet peu probable lorsque l'algorithme est correctement mis en œuvre et il est possible de s'en prémunir définitivement avec la variante Introsort.

/**algorithme

algorithm quicksort(A, lo, hi) is

  if lo < hi then

```
        p := partition(A, lo, hi)
        quicksort(A, lo, p)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[(lo + hi) / 2].Dur
    i := lo - 1
    j := hi + 1
    loop forever
        do
            i := i + 1
        while A[i] < pivot

        do
            j := j - 1
        while A[j] > pivot

        if i >= j then
            return j

        swap A[i] with A[j]
**/
```

pseudo code:

```
quicksortDuree(A, lo, hi) :
    if lo < hi then
        p := partitionDuree(A, lo, hi)
        quicksortDuree(A, lo, p)
        quicksortDuree(A, p + 1, hi)
```

```
partitionDuree(A, lo, hi) :
    pivot := A[(lo + hi) / 2].Duree
    i := lo - 1
    j := hi + 1
    loop forever
        do
            i := i + 1
        while A[i].Duree < pivot

        do
            j := j - 1
        while A[j].Duree > pivot

        if i >= j then
            return j

        swap A[i] with A[j]



quicksortLongueur(A, lo, hi) :
    if lo < hi then
        p := partitionLongueur(A, lo, hi)
        quicksortLongueur(A, lo, p)
        quicksortLongueur(A, p + 1, hi)

partitionLongueur(A, lo, hi) :
    pivot := A[(lo + hi) / 2].Longueur
    i := lo - 1
    j := hi + 1
    loop forever
        do
```

```
      i := i + 1
    while A[i].Longueur < pivot

    do
      j := j - 1
    while A[j].Longueur > pivot

    if i >= j then
      return j

    swap A[i] with A[j]



quicksortCarbon(A, lo, hi) :
  if lo < hi then
    p := partitionCarbon(A, lo, hi)
    quicksortCarbon(A, lo, p)
    quicksortCarbon(A, p + 1, hi)

partitionCarbon(A, lo, hi) :
  pivot := A[(lo + hi) / 2].Carbon
  i := lo - 1
  j := hi + 1
  loop forever
    do
      i := i + 1
    while A[i].Carbon < pivot

    do
      j := j - 1
    while A[j].Carbon > pivot
```

```
        if i >= j then
            return j

        swap A[i] with A[j]

algorithm quicksortPayant(A, lo, hi) is
    if lo < hi then
        p := partitionPayant(A, lo, hi)
        quicksortPayant(A, lo, p)
        quicksortPayant(A, p + 1, hi)

algorithm partitionPayant(A, lo, hi) is
    pivot := A[(lo + hi) / 2].Payant
    i := lo - 1
    j := hi + 1
    loop forever
        do
            i := i + 1
        while A[i].Payant < pivot

        do
            j := j - 1
        while A[j].Payant> pivot

        if i >= j then
            return j

        swap A[i] with A[j]
```

2.Les associations entre les classes

A. Un utilisateur peut choisir un seul trajet ou rien choisir dans la liste des trajets
B. L'application peut offrir au moins un trajet à l'utilisateur
C. Un tronçon appartient à un seul trajet dans une fois de guide de conduire
D. Un trajet contient au moins une tronçon
E. Un tronçon contient au moins une route
F. Une route appartient a un seul tronçon a cause de sa position géographique
G. Un tronçon est situé dans une seule ville à cause de sa position géographique
H. Une ville contient au moin un tronçon
I. Un trajet peut demander plusieurs fois de visualisation
J. Une visualisation peut être visualisé s'il y a une demande d'un trajet
K. Une ville peut demander plusieurs fois de visualisation
L. Une visualisation peut être visualisé s'il y a une demande d'une ville
M. Une route peut demander plusieurs fois de visualisation
N. Une visualisation peut être visualisé s'il y a une demande d'une route

MOAL

A.

a. definition $card_{Utilisateur.choisir} \equiv \forall u \in Utilisateur.\ 0 <= |u.choisir|<=1$

b. definition $ass_{Utilisateur.choisir.offrir} \equiv \forall u \in Utilisateur.\ u \in u.choisir.offrir$

B.

a. definition $card_{Trajet.offrir} \equiv \forall t \in Trajet.\ |t.offrir| >= 1$

b. definition $ass_{Trajet.offrir.choisir} \equiv \forall t \in trajet.\ t \in t.offrir.choisir$

C.

a. definition $card_{Troncon.appartenir} \equiv \forall t \in Troncon.\ |t.appartenir| =1$

b. definition $ass_{Troncon.appartenir.contenir} \equiv \forall t \in Troncon.\ t \in t.appartenir.contenir$

D.

a. definition $card_{Trajet.contenir} \equiv \forall t \in Trajet.\ |t.contenir| >=1$

b. definition $ass_{Trajet.contenir.appartenir} \equiv \forall t \in Trajet.\ t \in t.contenir.appartenir$

E.

a. definition $card_{Troncon.contenir} \equiv \forall t \in Troncon.\ |t.contenir| >=1$

b. definition $ass_{Troncon.contenir.appartenir} \equiv \forall t \in Troncon.\quad t \in t.contenir.appartenir$

F.

a. definition $card_{Route.appartenir} \equiv \forall r \in Route.\ |r.appartenir| =1$

b. definition $ass_{Route.appartenir.contenir} \equiv \forall t \in Troncon.\quad t \in t.contenir.appartenir$

G.

a. definition $card_{Troncon.situer} \equiv \forall t \in Troncon.\ \ |t.situer| = 1$

b. definition $ass_{Troncon.situer.contenir} \equiv \forall t \in Troncon.\quad t \in t.situer.contenir$

H.

a. definition $card_{Ville.contenir} \equiv \forall v \in Ville.\ \ |v.contenir| >= 1$

b. definition $ass_{Ville.contenir.situer} \equiv \forall v \in Ville.\ \ v \in v.contenir.situer$

I.

a. definition $card_{Trajet.demander}$ ≡ ∀ t ∈ Trajet. |t.demander| >= 0

b. definition $ass_{Trajet.demander.visualiser}$ ≡ ∀ t ∈ Trajet. t ∈ t.demander.visualiser

J.

a. definition $card_{Visualisation.visualiser}$ ≡ ∀ v ∈ Visualisation. 0 <= |v.visualiser| <= 1

b. definition $ass_{Visualisation.visualiser.demander}$ ≡ ∀ v ∈ Visualisation. v ∈ v.demander.visualiser

K.

a. definition $card_{Ville.demander}$ ≡ ∀ v ∈ Ville. |t.demander| >= 0

b. definition $ass_{Ville.demander.visualiser}$ ≡ ∀ v ∈ Ville. v ∈ v.demander.visualiser

L.

a. definition $card_{Visualisation.visualiser}$ ≡ ∀ v ∈ Visualisation. 0 <= |v.visualiser| <= 1

b. definition $ass_{Visualisation.visualiser.demander}$ ≡ ∀ v ∈ Visualisation. v ∈ v.demander.visualiser

M.

a. definition $card_{Route.demander}$ ≡ ∀ r ∈ Route. |r.demander| >= 0

b. definition $ass_{Route.demander.visualiser}$ ≡ ∀ r ∈ Route. r ∈ r.demander.visualiser

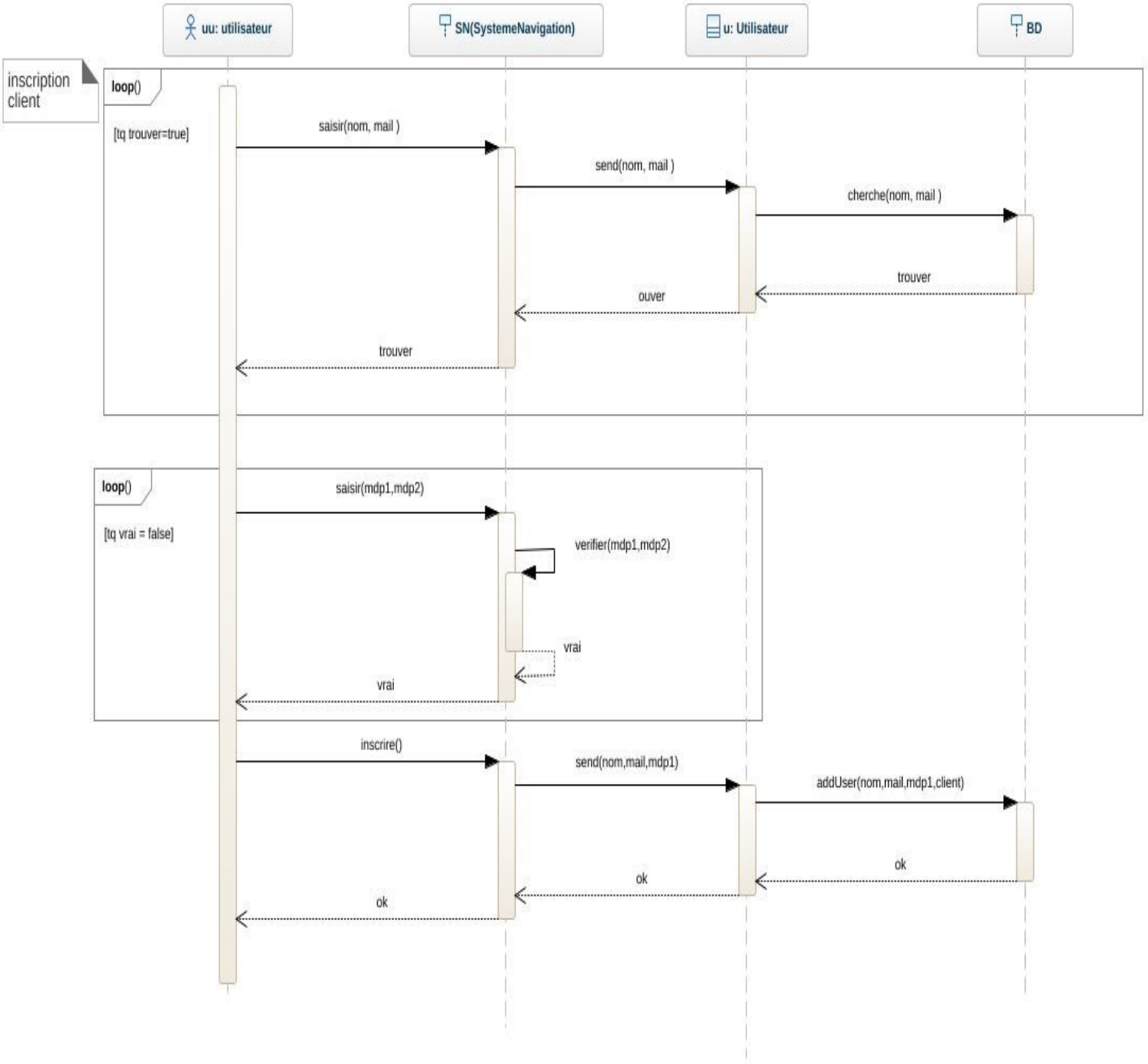
N.

a. definition $card_{Visualisation.visualiser}$ ≡ ∀ v ∈ Visualisation. 0 <= |v.visualiser| <= 1

b. definition $ass_{Visualisation.visualiser.demander}$ ≡ ∀ v ∈ Visualisation. v ∈ v.demander.visualiser

IV Diagramme de séquence

1) inscription



Description

un nouveau client doit s'inscrire dans la base de données

Pré-conditions

email ≠ null ∧ nom≠ null ∧ mdp1 ≠ null ∧ mdp2 ≠ null

∧ trouver = true

∧ vrai = false

Post-conditions

∀ u ∈ user, email = u.email ∧ nom = u.nom

mdp1 = mdp2 ∧ u ∈ new(Utilisateur)

Arguments

nom,email,mdp1 ,mdp2

Type des arguments et de retour

nom : String

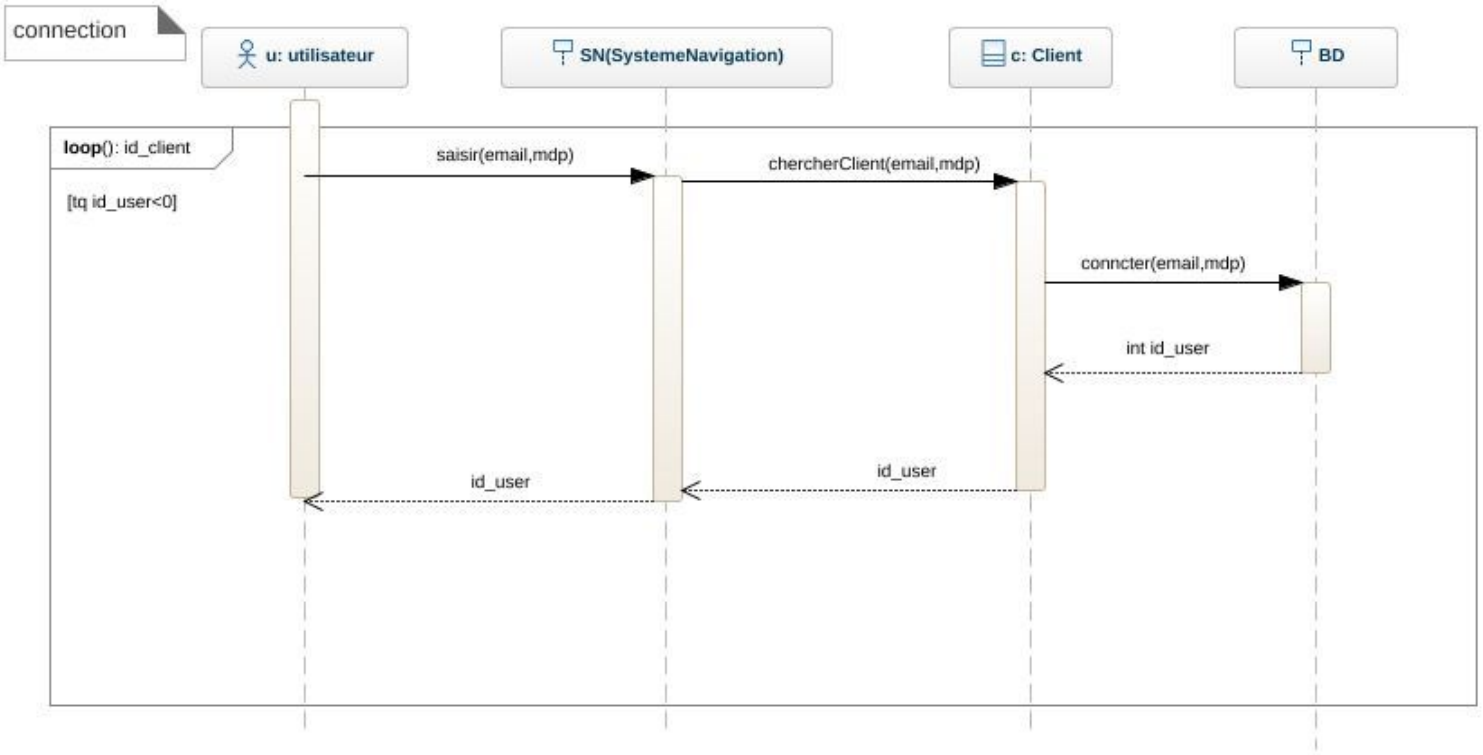
email : String

mdp1 : String

mdp1 : String

id_user :int

2) connection



Description

Un visiteur possédant un compte utilisateur souhaite se connecter au site web, pour cela il entre son adresse mail et son mot de passe, les deux informations sont correctes et l'utilisateur est connecté au site web.

et la fonction va retourner id_user si une fois utilisateur à connecter avec succès

user.seConnecter(email: String, mdp: String)

Pré-conditions

email ≠ null ∧ mdp ≠ null

Post-conditions

$\exists u \in user,\ email = u.email \wedge mdp = u.mdp$

$\forall p \in bd.user, email \neq p.email \wedge mdp \neq p.mdp$ => result = id_user

Arguments

email, mdp

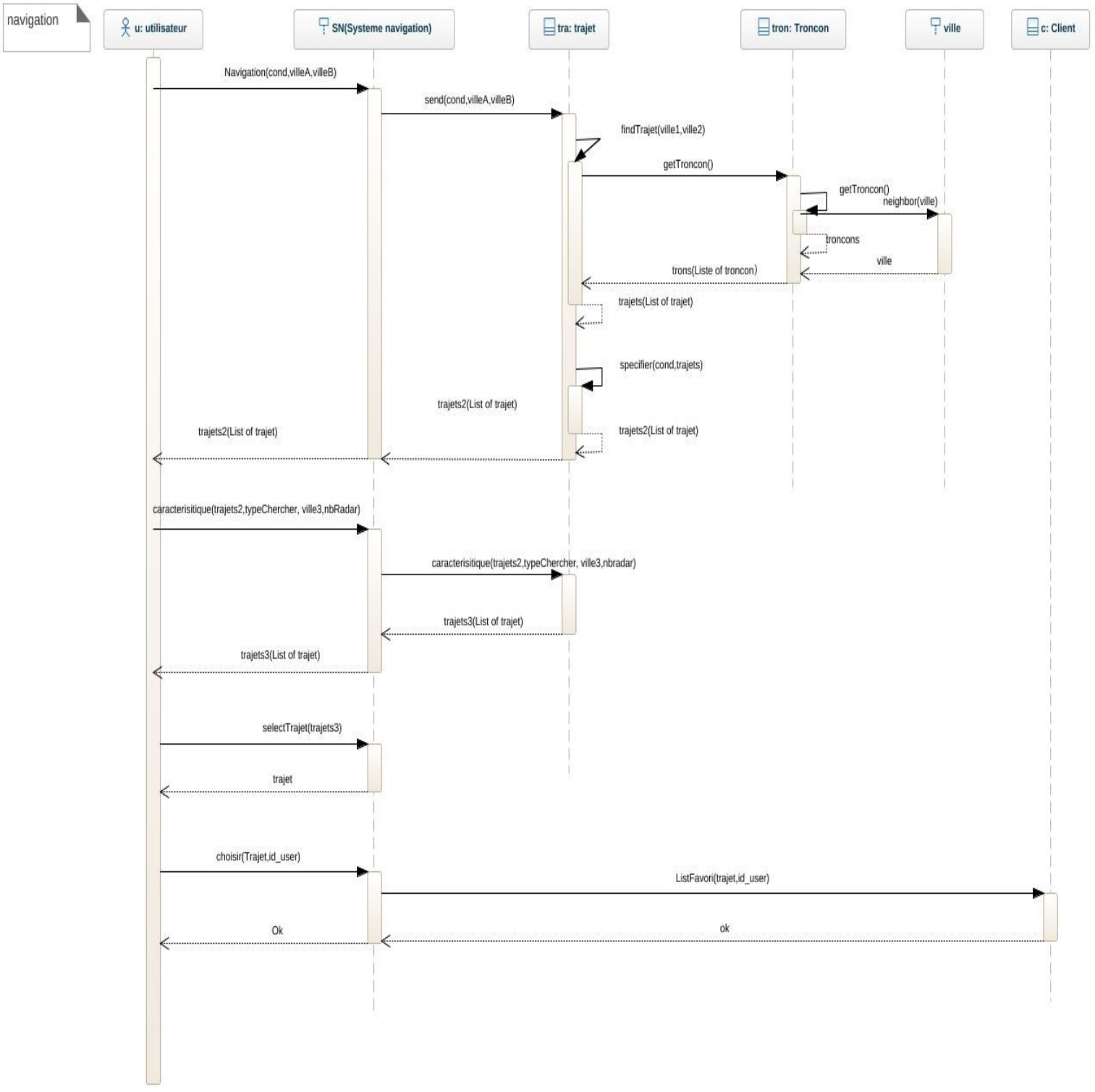
Type des arguments et de retour

email : String

mdp : String

id_user :int

3) chercher un trajet et ajouter dans la liste favorite



Description

La fonction principale de ce projet est creer une application web d'auto-mobile pour trouver un trajet à conduire qui est choisi par l'utilisateur. L'utilisateur définit "ville1" (la ville de départ) et "ville2" (la ville de destinataire) avec l'aide des différents standards (le plus court, le plus vide, moins de consommation de carbon, le moins cher). Après, l'application va chercher tous les trajets possibles depuis ville1 à ville2 et les mettre en ordre en forme de liste avec les critères.

On s'intéresse a la class SystemNavigation qui contient deux fonctions, "send" et "caractéristique"

La fonction "send" fonctionne comme: SystemNavigation.send(cond String, villeA: String, villeB: String)

Pré-conditions

ville1≠ null ∧ ville2≠ null ∧ ville1≠ ville2∧ (cond='court' v cond ='conso' v cond = 'vite' v cond = 'tarif')

Post-conditions

∃ v1 ∈ bd.Ville, ∃ v2 ∈ bd.Ville, ville1= v1.nom∧ ville2= v2.nom

∧cond=’court’ => result = trajetsChoix{t1,t2,t3} ordre par le longuer

∃ v1 ∈ bd.Ville, ∃ v2 ∈ bd.Ville, ville1= v1.nom∧ ville2= v2.nom

∧cond=’conso’ => result = trajetsChoix{t1,t2,t3} ordre par la consomation

∃ v1 ∈ bd.Ville, ∃ v2 ∈ bd.Ville, ville1= v1.nom∧ ville2= v2.nom

∧cond=‘vite’ => result = trajetsChoix{t1,t2,t3} ordre par la duree

∃ v1 ∈ bd.Ville, ∃ v2 ∈ bd.Ville, ville1= v1.nom∧ ville2= v2.nom

∧cond=’tarif’ => result = trajetsChoix{t1,t2,t3} ordre par le valeur de payant

Arguments

ville1; ville2; cond

Type des arguments et de retour

ville1: String

ville2:String

cond :String

La fonction “caractéristique” fonctionne comme: SystemNavigation.caracteristique(trajets, type ,ville,nb)

Pré-conditions

trajets ≠ null ∧ type ≠ null ∧ ville ≠ NULL ∧ nb ≠ NULL

Post-conditions

∃ t ∈ trajets, t.typeChercher = type∧ ville ∈ t.villeEvite ∧ nbRadar = nb

Arguments

trajets; typeChercher; ville; nbRadar

Type des arguments et de retour

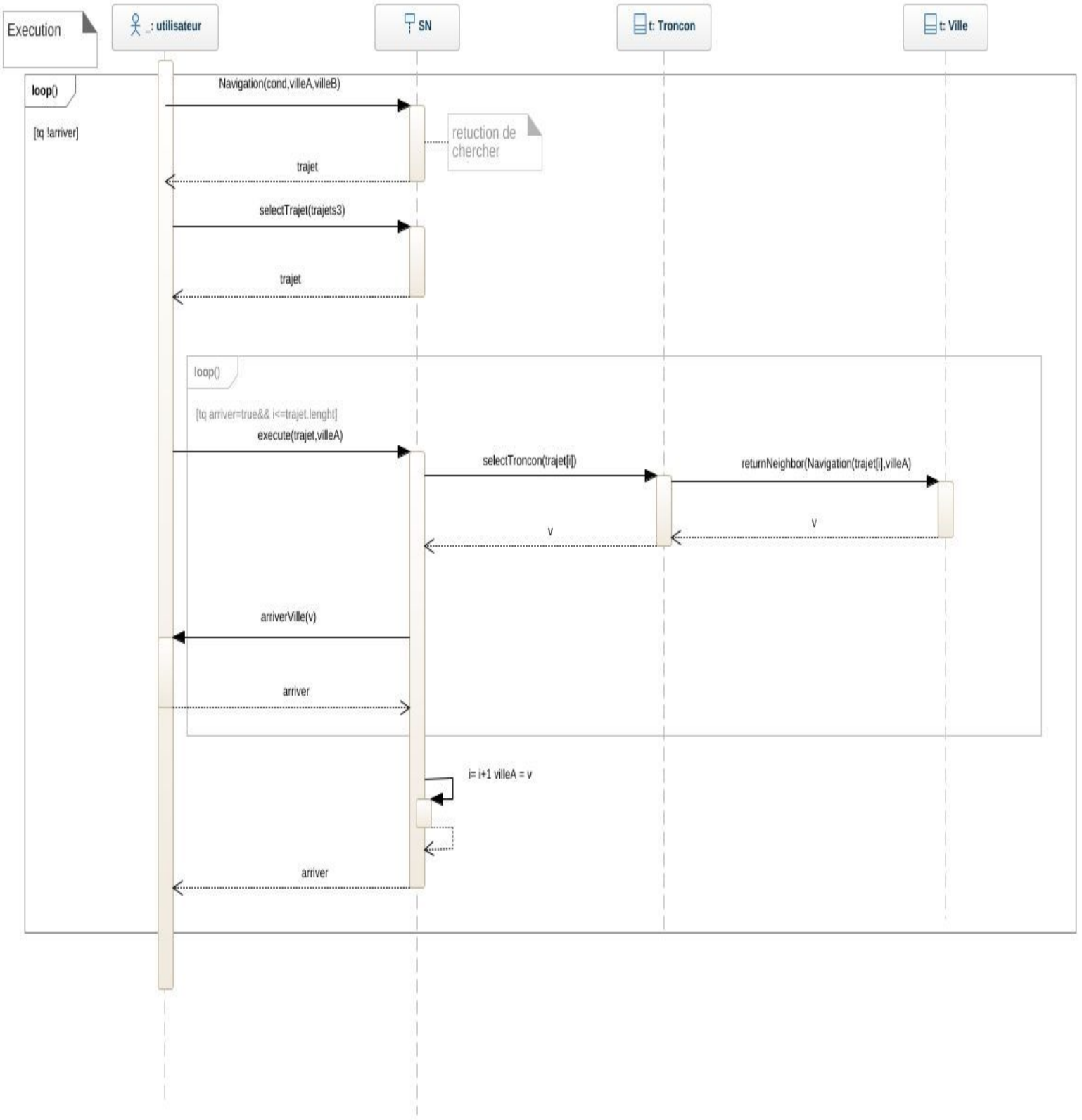
trajets: list of trajet

typeChercher:String

ville: Ville

nbRadar : int

4)Exécution (en train de Navigation)



Une fois que l'utilisateur a sélectionné un itinéraire, le logiciel trace et connecte les segments de route inclus dans l'itinéraire sur la carte.

Lors de la navigation, une fenêtre contextuelle s'affiche dans chaque ville pour confirmer si l'utilisateur a atteint la ville. L'utilisateur a deux options: Oui ou Non, lorsque l'utilisateur choisit Oui, la navigation se poursuit et passe à la suivante. La fenêtre contextuelle de la ville jusqu'à ce que l'utilisateur atteigne la destination finale

Lorsque l'utilisateur sélectionne Non, la navigation est terminée et l'utilisateur est considéré comme perdu. L'utilisateur doit entrer à nouveau l'emplacement actuel et la destination pour pouvoir naviguer à nouveau.

Pré-conditions

$\exists$ villeA,villeB $\in$ Ville, villeA. nom $\neq$ villeB.nom $\wedge$ villeA$\neq$ null $\wedge$ villeB $\neq$ null $\wedge$ cond $\neq$ null $\wedge$ arriver = false

Post-conditions

arriver = true

Arguments

cond

villeA

villeB

Type des arguments et de retour

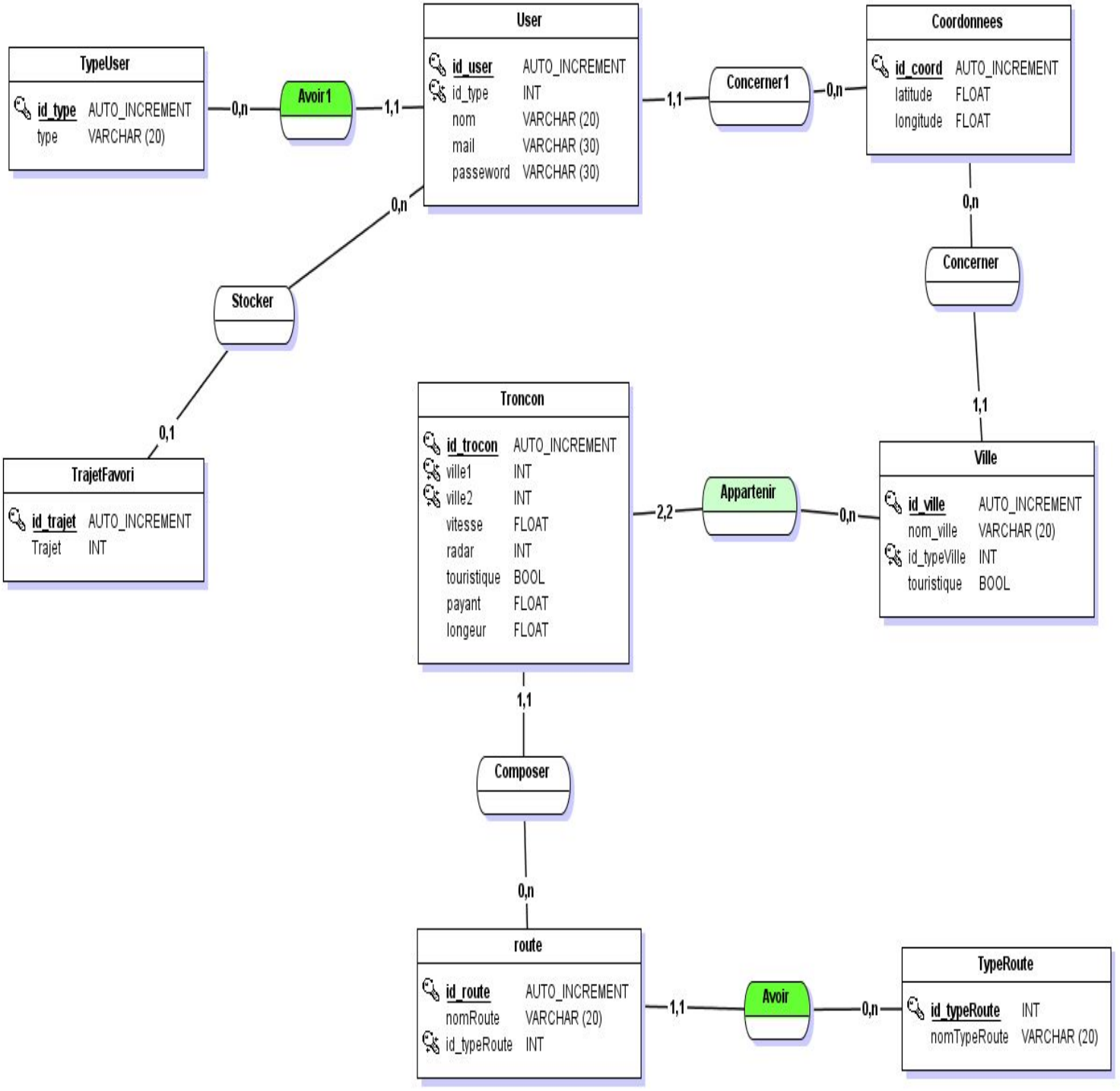
cond : String villeB : ville

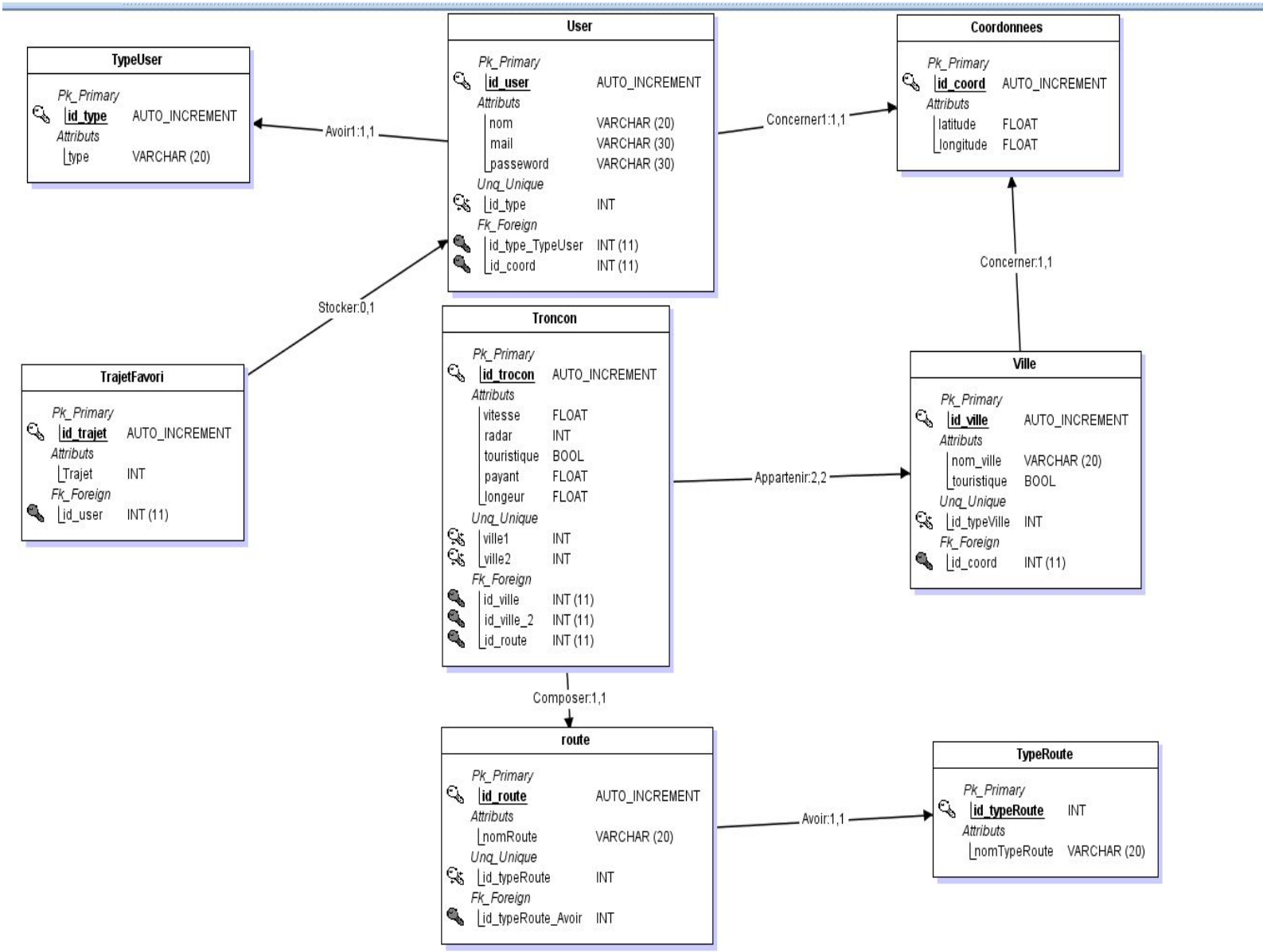
villeA : ville arriver : boolean

V. Base de données

On choisit d'utiliser MySQL pour stocker les données

/*utilisation de MySql MCD*/




TypeUser
Pk_Primary
id_type AUTO_INCREMENT
Attributs
type VARCHAR (20)
Avoir1:1,1
User
Pk_Primary
id_user AUTO_INCREMENT
Attributs
nom VARCHAR (20)
mail VARCHAR (30)
passeword VARCHAR (30)
Unq_Unique
id_type INT
Fk_Foreign
id_type_TypeUser INT (11)
id_coord INT (11)
Concerner1:1,1
Coordonnees
Pk_Primary
id_coord AUTO_INCREMENT
Attributs
latitude FLOAT
longitude FLOAT
Concerner:1,1
Stocker:0,1
TrajetFavori
Pk_Primary
id_trajet AUTO_INCREMENT
Attributs
Trajet INT
Fk_Foreign
id_user INT (11)
Troncon
Pk_Primary
id_trocon AUTO_INCREMENT
Attributs
vitesse FLOAT
radar INT
touristique BOOL
payant FLOAT
longeur FLOAT
Unq_Unique
ville1 INT
ville2 INT
Fk_Foreign
id_ville INT (11)
id_ville_2 INT (11)
id_route INT (11)
Appartenir:2,2
Ville
Pk_Primary
id_ville AUTO_INCREMENT
Attributs
nom_ville VARCHAR (20)
touristique BOOL
Unq_Unique
id_typeVille INT
Fk_Foreign
id_coord INT (11)
Composer:1,1
route
Pk_Primary
id_route AUTO_INCREMENT
Attributs
nomRoute VARCHAR (20)
Unq_Unique
id_typeRoute INT
Fk_Foreign
id_typeRoute_Avoir INT
Avoir:1,1
TypeRoute
Pk_Primary
id_typeRoute INT
Attributs
nomTypeRoute VARCHAR (20)

VI. Conclusion

En ce moment la, on a finit la partie de conception de notre projet <Mini Navi>. Dans cette opportunité de travail, on a appris beaucoup.

D'abord, le UML nous aide beaucoup pour réaliser les analyses des relations et les fonctions des différentes parties et l'actualisation de cette application avec l'aide de diagramme de classe et diagramme de séquence.

Pour bien afficher les relations entre les classes qui contient les différentes parties de cette application, on utilise la Diagramme de classe en définissant les attributs et les fonctions des classes dans cette application et les associations et les relations quantitatives entre eux.

Après, on écrit la diagramme de séquence pour afficher l'actualisation de classes de cette application (comment les demandes des classes reçoivent les réponses sous les utilisations des fonctions de classes)avec les séquences en ordre.

Pour bien enregistrer les données des utilisateurs et les informations des routes et villes, on décide d'utiliser SQL en créant la base de données avec les classes et leurs associations.

Pour exprimer les fonctions concernants les classes dans la diagramme de classes, on les écrit en MOAL ce qu'on a appris dans la dernière semestre avec les définitions, les arguments, les pré-conditions et les post-conditions des fonctions et les définitions et les cardinalisations des attributs dans les classes.

En même temps, pour les codes de la futur, on écrit les pseudo-code l'algorithme des fonctions nécessaires dans les classes définies et on cherche les algorithmes efficaces pour réaliser les demandes de trouver les trajets possibles et recevoir les réponses possibles qu'on demandes.

Ensuite, on décide d'utiliser les langages PHP et JavaScript pour définir les fonctions et les attributs nécessaires de l'application et les enregistrer dans

la base de donnée, et on va utiliser la langage HTML pour réaliser l'interface de l'utilisateur et le système de l'application.

Finalement, sauf les études et les applications des langages comme les outils de ce projet, on a aussi obtenu l'importance de la coopération dans le groupe. Quand on travaille ensemble, on peut mieux s'aider dans les différentes domaines et nos camarades peuvent partager les travaux. Avec les aides des camarades, on peut mieux apprendre et utiliser les langages et resoudre les problemes dans le projet.