Projet Génie Logiciel

Avancé L3

Conception

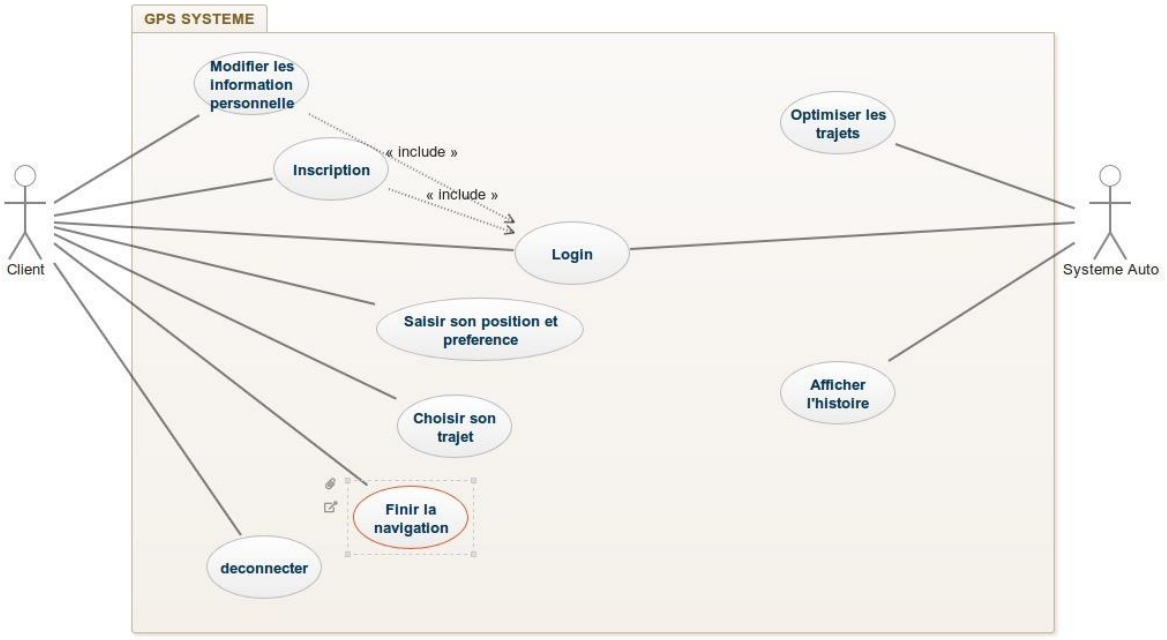
2018-2019

par

Jiangting ZHANG
Shuo ZHANG
Yaqi ZHENG

SOMMAIRE

·Diagramme de cas d'utilisation·

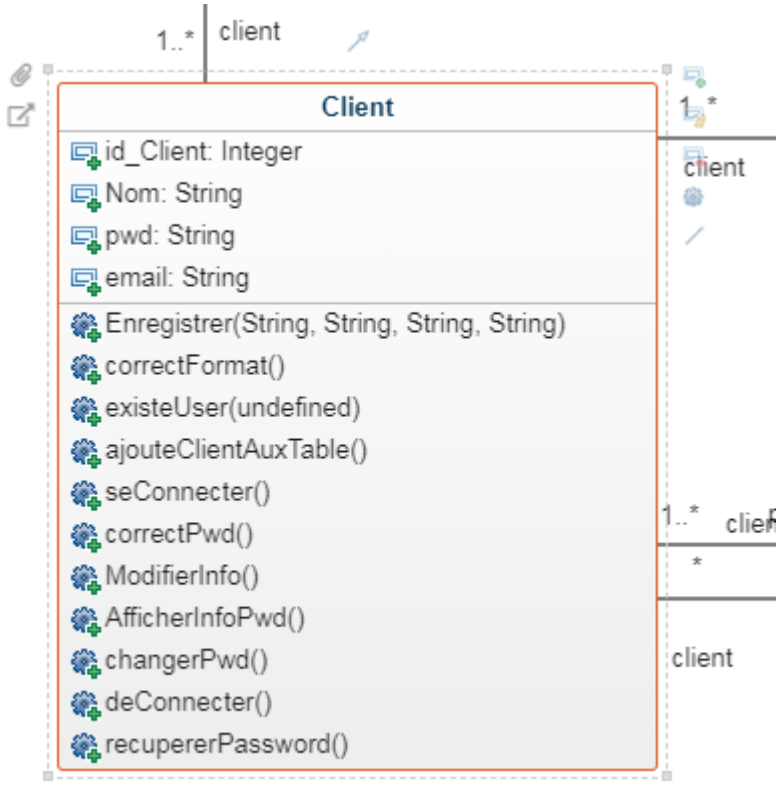


Pour la mise en œuvre du système, l'utilisateur peut utiliser les six fonctionnalités que l'utilisateur souhaite réaliser, comme indiqué dans le cas d'utilisation de la figure ci-dessus: enregistrement propre de l'utilisateur, connexion de l'utilisateur (y compris la connexion de l'utilisateur du visiteur), adresse de saisie de l'utilisateur (pouvoir également saisir certaines préférences) pour la navigation, l'utilisateur peut sélectionner un trajet de navigation,

fermer activement l'interface de navigation et quitter l'état de connexion.

·Diagramme de classe avec des fonctionnalités·

Class Client :



Une table d'utilisateurs doit être créée pour stocker des informations sur les utilisateurs enregistrés, notamment l'ID utilisateur, le nom d'utilisateur, le mot de passe et l'adresse électronique.

Chaque utilisateur doit posséder un identifiant et un nom d'utilisateur uniques.

L'existence de l'ID facilite la connexion entre les autres tables et cette table de client. Le système incrémente automatiquement l'ID d'une unité chaque fois que de nouvelles données utilisateur sont insérées.

L'unicité du nom d'utilisateur peut être utilisée pour voir si l'utilisateur est enregistré de manière répétée.

Le mot de passe est utilisé pour protéger la sécurité des informations des utilisateurs membres.

E-mail, c'est pour laisser les vraies informations de contact de l'utilisateur, et de faciliter le contact entre nous et l'utilisateur. Deuxièmement, afin d'utiliser le mot de passe oublié par l'utilisateur, le système peut lui donner le droit de le changer en saisissant l'e-mail.

D'abord creer une table de client(idclient, nom, pwd, mail)

```
CREATE TABLE CLIENT
(
    idclient INT PRIMARY KEY,
    nom Varchar2(20) unique not null,
    pwd Varchar2(20) not null,
    mail Varchar2(30) not null
)
```

idclient	nom	pwd	mail
0	client libre	123	123@123.com
1	james	james123	james@gmail.com
2	zzz	zzz123	zzz@hotmail.com

Pour les utilisateurs de visiteurs, bien qu'ils n'aient pas besoin de s'inscrire ni de s'identifier, ils auront également certaines informations générées. Pour cela, nous devons créer un compte pour ce type d'utilisateur. Comme indiqué ci-dessus, l'utilisateur dont idclient vaut 0 est tous les utilisateurs de visiteurs. Nous autorisons ces utilisateurs à entrer dans le système sans s'enregistrer ni se connecter, mais leurs informations d'adresse ne peuvent être associées qu'aux membres de visiteurs dont le idclient est 0.

LES FONCTIONS DE CLIENT:

Fonction1: existeUser(String nom，List l): revoie boolean

Vérifiez si le nouveau nom d'utilisateur existe. Il est généralement utilisé pour vérifier le nom d'utilisateur lors de la création d'un nouvel utilisateur ou lors de la connexion de

l'utilisateur. Renvoie VRAI si le nom d'utilisateur saisi existe déjà dans la table de client, sinon renvoie FAUX.

Cette fonction nécessite deux paramètres: le nom d'utilisateur saisi par l'utilisateur et la liste de tous les noms d'utilisateur figurant dans la table de clients.La liste de noms d'utilisateurs peut être complétée par sql: Select nom from client

Pseudo-code：

```
existeUser(String nom，List l): revoie boolean
{
for n in l：
    if nom==n then retourne true;
retourne false;
}
```

MOAL：

definition PRE-existeUser(nomC:String, list:List(x))≡

{∀x ∈ Client | x.nom∈ list} /\ |list|>0 /\ nomC ≠null

definition POST-existeUser(nomC:String, list:List(x),r:Boolean)≡

{∀x ∈ list | x ≠ nomC }/\ r==True

Fonction2:ajouteClientAuxTable(int id，String nom,String pwd,String mail):

Après chaque inscription réussie, le système doit insérer les données du nouvel utilisateur dans la table de client et attribuer à l'utilisateur suivant le prochain identifiant disponible.

La fonction nécessite que l'utilisateur saisisse le nom d'utilisateur, le mot de passe et le courrier, le courrier étant principalement utilisé lorsque l'utilisateur oublie le mot de passe. Si l'utilisateur oublie le mot de passe défini et doit saisir le courrier avec précision, l'utilisateur peut être qualifié pour le changer. Le mot de passe de l'utilisateur dans la table de client peut être modifié.

Pseudo-code：

```
ajouteClientAuxTable(int id，String nom,String pwd,String mail):
{
INSERT INTO client VALUES (id，nom，pwd，mail)；
}
```

MOAL：

definition PRE-ajouteClientAuxTable(id:int,nom:String,pwd:String,mail:String)≡
{∀x ∈ Client | x.clientid ≠id} /\ nom≠null /\ pwd≠null /\ mail≠null
definition POST-
ajouteClientAuxTable(id:int,nom:String,pwd:String,mail:String,r:Boolean)≡
new(id)/\ new(nom)/\isUnique(nom)/\new(mail)/\r==True

Fonction3:correctPwd(String nom,String pwd):boolean

Cette fonction permet de vérifier si le mot de passe entré par l'utilisateur est correct. S'il est correct, il renvoie true, sinon, false.
Généralement utilisés pour les opérations de connexion utilisateur, les paramètres requis sont le nom d'utilisateur saisi par l'utilisateur et le mot de passe correspondant trouvé dans la table de client.

Pseudo-code：

```
correctPwd(String n,String pwd):boolean
{
if pwd==（select  pwd from client where nom=n）then retourne true;
else retourne false;
}
```

MOAL：

definition PRE-correctPwd(n:String ,pwd:String)≡
n≠null/\pwd≠null
definition POST-correctPwd(n:String ,pwd:String ,r:Boolean)≡
{∃x ∈ Client | x.nom==n /\ pwd==x.pwd}/\r==True

Fonction4:correctEmail(String n,String email):boolean

Cette fonction est utilisée lorsque l'utilisateur récupère le mot de passe et vérifie si le courrier saisi par l'utilisateur est identique à celui de la table de client. Si la différence est fausse, la même chose retourne à vrai.
Les paramètres requis sont le nom d'utilisateur saisi par l'utilisateur et l'email correspondant trouvé dans la table de client.

Pseudo-code：

```
correctEmail(String n,String email):boolean
{
if email==（select email from client where nom=n）then retourne true；
else retourne false；
}
```

MOAL：

definition PRE-correctEmail(n:String ,email:String)≡

n≠null/\email≠null

definition POST-correcEmail(n:String ,email:String ,r:Boolean)≡

{∃x ∈ Client | x.nom==n /\ emaill==x.email}/\r==True

Fonction5：changerPwd（String nom,String pwd1，String pwd2，String pwd3）

Le changement de mot de passe est généralement utilisé lorsque l'utilisateur oublie le mot de passe ou s'il souhaite volontairement changer le mot de passe.

1. Lorsque l'utilisateur réinitialise le mot de passe, l'utilisateur doit saisir deux mots de passe (nouveau mot de passe et mot de passe de vérification). Si l'utilisateur ne les entre pas, le message suivant s'affiche: "Veuillez saisir tous les mots de passe".

2. Les deux mots de passe requis par l'utilisateur doivent être identiques. S'ils sont différents, le message: "Le mot de passe a été mal saisi, veuillez le saisir à nouveau!"

3. Le nouveau mot de passe devant être modifié par l'utilisateur ne peut pas être dupliqué avec le mot de passe précédent. S'il est répété, il affiche: "Désolé, votre mot de passe est le même qu'auparavant, la modification a échoué."

Si les trois éléments ci-dessus sont acceptés, le message suivant s'affiche: "Votre mot de passe a bien été modifié!" Et retourne à la page de connexion.

Pseudo-code：

changerPwd（String n,String pwd1，String pwd2，String pwd3）：

{

if pwd1=='' or pwd2=='' then print “remplir tous, svp!”

else if pwd1!= pwd2 then print “les codes sont pas meme”

else if correctPwd(n,pwd1) then print”Désolé, votre mot de passe est le même que précédemment, la modification a échoué.”

else

{

UPDATE client SET pwd = pwd1 WHERE nom=n

print”Votre mot de passe a été changé avec succès!”

}

}

MOAL:

definition PRE-changerPwd(n:String,pwd1:String,pwd2:String,pwd3:String)≡

{∃x ∈ Client | x.nom==n /\ x.pwd=pwd3}/\ pwd1≠null /\pwd2≠null /\pwd1==pwd2

definition POST-

changerPwd(n:String ,pwd1:String,pwd2:String,pwd3:String,r:Boolean)≡

{∃x ∈ Client | x.nom==n /\ x.pwd->pwd1}/\r==True

Fonction6：Enregistrér(String nom; String pwd1; String pwd2; String mail)

Pour les utilisateurs qui souhaitent devenir membre, l'opération d'enregistrement est indispensable: lorsque l'utilisateur s'enregistre, le système doit surveiller et examiner leurs demande.

1. Le système doit vérifier les informations saisies par l'utilisateur. Les options requises ne peuvent pas être vides et le format de saisie est correct.

2.Après la première étape, il faut: assurer que chaque utilisateur est unique. Les utilisateurs précédemment enregistrés ne peuvent pas être enregistrés de manière répétée. La méthode de jugement consiste à ne pas dupliquer le nom d'utilisateur avec le nom d'utilisateur figurant dans la table de client. Si la vérification est répétée, l'invite suivante: "Désolé, le nom d'utilisateur a été enregistré, changez votre nom !" Et retournez à la page d'inscription.

Une fois que les quatre audits ci-dessus ont été passés, le système peut accepter l'enregistrement de l'utilisateur.

Le système doit générer de nouvelles informations utilisateur: un ID , un nom , un mot de passe et un courrier électronique. L'ID de l'utilisateur est généré automatiquement par le système. Chaque fois qu'un utilisateur est ajouté avec succès, l'ID du système est automatiquement incrémenté de 1.

Une fois que les informations utilisateur ont été générées avec succès, le système doit indiquer: "Vous vous êtes enregistré avec succès, connectez-vous." Et passez à la page de connexion.

Pseudo-code：

```
Enregistrer（String nom; String pwd1; String pwd2; String mail）：
{
if correctFormat(nom;pwd1;pwd2;mail)
then
if existeUser(nom) then print “Le nom d'utilisateur a été enregistré”
else
{
ajouteClient(nom,pwd1,pwd2,mail);
print “Vous vous êtes enregistré avec succès, veuillez vous connecter pour utiliser”;
}
}
```

MOAL:

definition PRE-Enregistrer(nom:String,pwd1:String,pwd2:String,mail:String)≡
{∀x ∈ Client | x.nom ≠nom} /\ nom≠null /\ pwd1≠nul/\ pwd2≠nulll /\ mail≠nul/\/\ pwd1≠pwd2
definition POST-Enregistrer(nom:String,pwd1:String,pwd2:String,mail:String,r:Boolean)≡
{∃x ∈ Client | new(x.id)}/\ new(nom)/\isUnique(nom)/\new(mail)/\r==True

Fonction7：seConnecter(String nom; String pwd):

Pour les utilisateurs déjà enregistrés, ils doivent se connecter pour accéder à l'interface:

1. L'utilisateur doit entrer son nom d'utilisateur et son mot de passe. Si un élément n'est pas saisi, le système affiche: "Veuillez saisir votre nom d'utilisateur et votre mot de passe" et retourner à la page de connexion.

2. Vérifiez si le nom d'utilisateur entré par l'utilisateur existe dans la table de client du système. S'il existe, vérifiez si le mot de passe entré par l'utilisateur est identique à celui de la table de client. S'ils sont identiques, le message s'affiche: "Connecté avec succès" et passez à la page suivante. Sinon: "Désolé, votre mot de passe est faux!". Si le nom d'utilisateur n'existe pas, vous devez être invité: "Désolé, cet utilisateur n'a pas été trouvé!"

Toutefois, pour les utilisateurs qui ne se sont pas enregistrés, si vous souhaitez vous connecter directement, le système doit déterminer si l'utilisateur a saisi le nom d'utilisateur ou le mot de passe. Une fois qu'il a saisi un , ce sera considéré comme un compte. Sinon on considéré comme un visiteur, ce qui lui permet de se connecter directement, et la table de client envoie les informations de compte portant le numéro d'id 0 (le compte ayant le numéro d'identification 0 correspond à toutes les informations relatives aux visiteurs.)

Pseudo-code：

```
seConnecter（String nom; String pwd）：
{
if nom==’’ and pwd=’’ then print “Connecté avec succès”
else if (nom==’’ and pwd!=’’) or (nom!=”” and pwd==””)  then print  “remplir tous, svp!”
else if  existeUser（nom）  then
    if correctPwd(nom,pwd) then print “Connecté avec succès”
    else print “Mot de passe est incorrect!”
else print “Désolé, cet utilisateur n'a pas été trouvé!”
}
```

MOAL:

definition PRE-seConnecter(nom:String,pwd:String)≡
(nom≠null /\ pwd≠null)\/(nom=null /\ pwd=null)
definition POST-seConnecter(nom:String,pwd:String,r:Boolean)≡

{∃x ∈ Client | x.nom =nom/\x.pwd=pwd}/\r==True

Fonction8：deConnecter (int id):

Afin de garantir la sécurité des informations des utilisateurs membres, après avoir utilisé le logiciel, l'utilisateur peut cliquer sur le déconnecteur et après la système afficher: "Vous avez deconnecté! " et revenir à la page d'enregistrement de connexion.

Cependant, pour cette opération, uniquement pour les utilisateurs déjà enregistrés et connectés, le système doit d'abord vérifier si l'utilisateur actuel est un utilisateur membre.

Après la déconnexion, toutes les informations du système (informations d'historique de l'adresse) sont l'utilisateur visiteur dont le idclient est 0:

Pseudo-code：

```
deConnecter（String nom）：
{
print “Deconnecté avec succès”
select * from position where idclient=0；
}
```

MOAL:

definition PRE-deConnecterr(id:int)≡

id≠null/\{∃x ∈ Client | x.id =id}

definition POST-deConnecterr(id:int,r:Boolean)≡

{∃x ∈ Client | x.id >0/\x.id=id}/\r==True

Fonction9：recupererPassword (String n; String email)：

Afin d'éviter que l'utilisateur oublie le mot de passe, le système lui permet de réinitialiser le mot de passe en le récupérant. Cette méthode nécessite que l'utilisateur entre un nom d'utilisateur réel et valide, ainsi que l'adresse e-mail qu'il a renseignée lors de son inscription.

1. Demander aux utilisateurs de saisir leur nom d'utilisateur et leur adresse électronique. S'ils ne peuvent pas à le remplir, affichez: "Veuillez remplir tous les formulaires"

2. Si le nom d'utilisateur saisi par l'utilisateur existe réellement, vérifiez que l'adresse électronique saisie par l'utilisateur est correcte. Si l'adresse électronique entrée est incorrecte, le message suivant s'affiche: "Échec de récupération du mot de passe, l'adresse e-mail entrée est incorrecte, il vous est conseillé de réenregistrer le compte" et passez à page de l'enregistrement. Si le courrier électronique saisi est correct, deux zones de saisie apparaissent, invitant l'utilisateur à saisir un nouveau mot de passe et à le modifier.

3.Si l'utilisateur entre un nom d'utilisateur incorrect, il affiche: "Désolé, cet utilisateur n'a pas été trouvé." Et est retourné à la page de connexion.

Pseudo-code：

```
recupererPassword（String n; String email）：
{
if nom=='' or email==''  then print  “remplir tous, svp!”
else if  existeUser（nom）  then
     if correctEmail(n,email) then print “Vous pouvez changer votre mot de pass”
     else print “L’email que vous avez entré est incorrect.”
else print “Désolé, cet utilisateur n'a pas été trouvé!”
}
```

MOAL:

```
definition PRE-recupererPassword(n:String,email:String)≡
     n≠null/\email≠null/\{∃x ∈ Client | x.nom =n}
definition POST-recupererPassword(n:String,email:String,r:Boolean)≡
          {∃x ∈ Client | x.nom =n/\x.email=email}/\r==True
```

Fonction10：correctFormat(String nom; String pwd1; String pwd2; String mail):boolean

Le but principal de cette fonction est de vérifier si les informations saisies dans l’enregistrement de l’utilisateur répondent aux exigences de format du système et de vérifier si l’utilisateur a entré les options requises. Les options requises incluent: nom d’utilisateur, mot de passe de l’utilisateur, Mot de passe confirmé, ainsi que l‘email.

1. L'utilisateur doit renseigner le nom d'utilisateur, le mot de passe, le mot de passe de vérification et l'entrée du courrier électronique. Si l'un des quatre éléments est vide, il indique: "Vous devez remplir tous les formulaires pour vous inscrire avec succès, merci." Et retourner à la page d'inscription.

2. Les deux mots de passe (mot de passe défini et mot de passe de vérification) renseignés par l'utilisateur doivent être identiques, sinon l'invite: "Désolé, votre mot de passe est saisi de manière incorrecte, veuillez entrer à nouveau." Et retourner à la page d'enregistrement.

3. Il est nécessaire de déterminer si le format d’email saisi par l'utilisateur est exact et si le format saisi par l'utilisateur doit être conforme au format correct d’email. Sinon, l'invite: "Le format d’email est incorrect, veuillez entrer à nouveau." Et revenir à la page d'inscription.

Une fois que les conditions ci-dessus sont remplies, renvoie true, sinon renvoie false.

Pseudo-code：

```
correctFormat(String nom; String pwd1; String pwd2; String mail):boolean
```

{
if nom=='' or pwd1=='' or pwd2='' or mail=='' then {print "remplir tous, svp!";retourne false;}
else if pwd1!=pwd2 then {print " les mot de pass sont pas meme!";retourne false;}
else if mail n'est pas correct then {print "le mail pas correct";retourne false;}
// on utilise le auto-correcteur pour comfirmer le format de mail
else retourne true;

MOAL:

definition PRE-correctFormat(nom:String,pwd1:String,pwd2:String,mail:String)≡
nom≠null/\mail≠null/\pwd1≠null/\pwd2≠null/\{∀x ∈ Client | x.nom ≠nom}
definition POST-recupererPassword(n:String,email:StringcorrectFormat(nom:String,pwd1:String,pwd2:String,mail:String,r:Boolean)≡
{∀x ∈ Client | x.nom ≠nom}/\pwd1=pwd2/\r==True

Fonction11：ModifierInfo(String nom,String choix, List String nouveaustring, String email, int action):

Cette fonction permet aux utilisateurs qui sont déjà connectés d'apporter des modifications à leurs propres informations.Les options qui peuvent être modifiées sont le mot de passe de l'utilisateur, les informations d'adresse déjà définies (adresse du domicile et adresse de la société), ainsi que l'ajout et la suppression d'informations d'adresse.

Pour cela, la fonction demande à l'utilisateur de fournir le nom d'utilisateur (car seulment l'utilisateur en état de connexion peut modifier les informations personnelles), les options que l'utilisateur souhaite modifier et le nouveau contenu que l'utilisateur souhaite modifier (pour une liste des informations de contenu a modifier) et peut également avoir besoin de l'email de l'utilisateur (l'utilisateur modifie le mot de passe, il doit entrer le bon email pour vérifier si l'utilisateur est la personne actuelle): Si l'utilisateur ne modifie pas l'information du mot de passe, cet élément est vide par défaut. Enfin, l'utilisateur souhaite effectuer l'opération (0: modifier, 1: augmenter, 2: supprimer)

1. Vérifiez si l'utilisateur est connecté.

2. Accédez à la table de client et à la table de Position, en affichant toutes les informations que l'utilisateur peut modifier.

3. Vérifiez si l'option que l'utilisateur souhaite modifier appartient à la table de client ou les informations d'adresse:

3.1Si vous souhaitez modifier les informations de mot de passe, vous devez entrer le courrier électronique correct et procéder à la vérification de l'identité avant de pouvoir modifier le mot de passe dans la table de client. Les informations de mot de passe peuvent uniquement être modifiées et ne peuvent pas être ajoutées ou supprimées.

3.2S'il ne s'agit pas d'un mot de passe, il ne peut s'agir que des informations d'adresse. Vous devez vérifier ce que l'utilisateur souhaite faire:

Si vous souhaitez effectuer une opération de modification, vous devez fournir les informations d'adresse modifiées pour modifier les informations d'adresse.

Si l'action requise consiste à ajouter une opération, deux paramètres doivent être spécifiés: nom et adresse.

Si la suppression est donnée, la suppression de cette option est effectuée.

Pseudo-code：

ModifierInfo(String nom,String choix,List String nouveaustring，String email，int action):

if existeUser（nom） then

pwdoriginal->AfficherInfoPwd(nom) and list->AfficherInfoPossition(nom)

if choix==”pwd” then changerPwd(nom,nouveaustring[0]，nouveaustring[1],pwdoriginal)//0:pwd，1：pwd de verification

else if action=0 then supprimerPosition(choix,nouveaustring[1])//：adr

else if action=1 then AjouterPosition(nouveaustring[0]，nouveaustring[1])//0：nom，1：adr

else changerPosition(choix,nouveaustring[1])//1：adr

else print”Vous n’avez pas de droit , login d’abord”

MOAL:

definition PRE-ModifierInfo(nom:String,choix:String,nouveaustring:List(x),email:String,action:int)≡

nom≠null/\choix≠null/\nouveaustring≠null/\{∃x ∈ Client | x.nom =nom}

definition POST-ModifierInfo(nom:String,choix:String,nouveaustring:List(pwd),email:String,action:int,r:Boolean)≡

(nouveaustring[0] ≠null /\ nouveaustring[1]≠null /\ nouveaustring[1]==nouveaustring[0] /\ email≠null /\ {∃x ∈ Client | choix=x.pwd /\ x.nom =nom/\x.emai=emaill } /\ r==True)

\/

({ ∃x ∈ Client,∃y ∈ Position | x.nom =nom /\ x.idclient=y.idclient /\ choix=y.commentaire } /\

(action=0 \/ (action=1 /\ nouveaustring[0] ≠null /\ nouveaustring[1]≠null) \/ (action=3 /\ nouveaustring[1]≠null) /\ r==True)

Fonction12：AfficherInfoPwd(String nomclient): string

Cette fonction est principalement utilisée lorsque l'utilisateur visualise ses informations personnelles.Le système demande à la table de client d'appeler les informations relatives au mot de passe de l'utilisateur, ce qui est pratique pour que l'utilisateur puisse visualiser le mot de passe. Pour ce faire, le système doit donner le nom de l'utilisateur afin de filtrer les données.

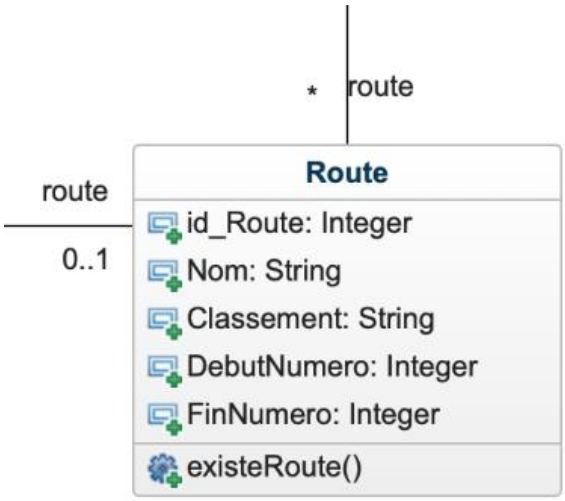
Pseudo-code：

```
AfficherInfoPwd(nomclient):string
{
SELECT pwd FROM client WHERE nom=nomclient；
}
```

MOAL：

definition PRE-AfficherInfoPwd(nomclient:String)≡
nomclient≠null/\{∃x ∈ Client | x.nom =nomClient}
definition POST-AfficherInfoPwd(nomclient:String,r:Boolean)≡
{∃x ∈ Client | x.nom =nomClient /\ x.pwd≠null}r==True

Class Route:



Une table de route est nécessaire pour sauvegarder les informations sur la route, y compris l'ID de la route, le nom de la route, le type de route, le numéro de la route, y compris le numéro de début et le numéro de fin.
Chaque route doit avoir un identifiant unique.

L'existence de l'identifiant est pratique pour établir le contact avec d'autres formulaires. Le système incrémente automatiquement l'ID chaque fois qu'une nouvelle donnée routière est insérée.

Le type de route, tel que route nationale, route provinciale, route rurale, etc., afin de calculer le chemin de navigation qui répond le mieux aux exigences de l'utilisateur, nous devons enregistrer le type de route.

Le numéro est utilisé pour savoir plus clairement où l'utilisateur se trouve sur la route après que l'utilisateur a entré l'adresse, et le chemin de navigation peut être calculé plus précisément lors de la navigation.

D'abord créer une table de route(id_route, Nom, Classement, DebutNumero, FinNumero)

```
CREATE TABLE ROUTE
(
  idRoute INT PRIMARY KEY,
  nom Varchar2(20) not null,
  Classement Varchar2(20) not null,
  DebutNumero INT not null,
  FinNumero INT not null
)
```

idRoute	nom	Classement	DebutNumero	FinNumero
0	A7	Autoroute	40	100
1	N10	National	0	150

Fonction1:existeRoute(String Route, list L):

Pour un système de navigation, il est essentiel de disposer d'une adresse de localisation complète et précise.

Si l'utilisateur en sélectionne une autre, l'utilisateur doit entrer le lieu de départ. Lorsque la saisie de l'utilisateur est terminée, nous utilisons les données saisies par l'utilisateur pour faire correspondre les données de l'itinéraire dans la base de données.

Une fois que l'utilisateur a entré le point de départ, afin de fournir un chemin de navigation plus précis, nous devons confirmer l'emplacement de l'utilisateur.
Nous utilisons l'emplacement indiqué par l'utilisateur pour confirmer l'existence de ce chemin dans la base de données, puis pour confirmer si la route existe dans la ville.
Si le chemin d'accès indiqué à l'emplacement par l'utilisateur n'existe pas, la fenêtre contextuelle suivante s'affiche: "Le chemin d'accès que vous avez entré n'existe pas, veuillez le saisir à nouveau."

Si le chemin d'accès indiqué à l'emplacement par l'utilisateur est correct, la fenêtre contextuelle intitulée "Confirmé" apparaît.

Pseudo-code:

existeRoute(String Route，List L): renvoie boolean

```
                                        //(On verifier si le route est existe dans le tableau)
for r in L:
        if Route==r
          then print “Confirme”
          retourne true
    else
                print   "La route que vous avez entrée n'existe pas"
                retourne false
          break;
```

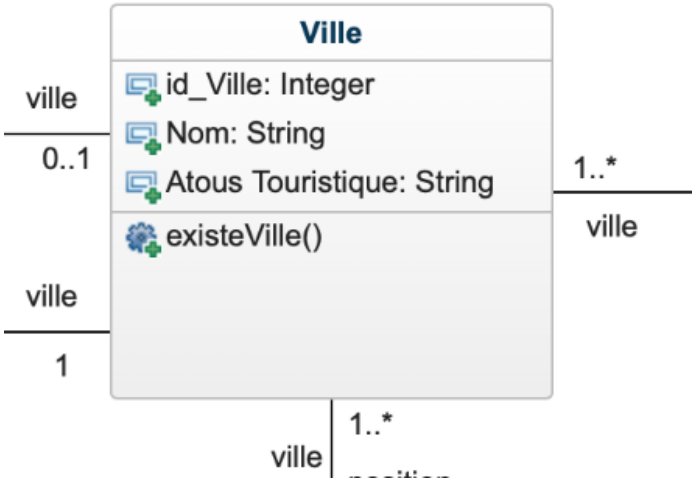
MOAL:

Definition Pre existeRoute(route: String, L:List) ≡ route!=null \/ L!= null

Definition Post existeRoute(route: String, L:List, b:Boolean) ≡

{∃R∈ Route|r.nom==route}\/b=True}

Classe Ville:



Un formulaire de ville doit être créé pour stocker des informations sur la ville, y compris le numéro d'identification de la ville, le nom de la ville, le type de ville et indiquer si la ville possède des attractions touristiques.
Demandez aux identifiants et noms de chaque ville d'être uniques.
L'existence de l'identifiant est commode pour établir le contact avec d'autres formulaires et le système l'incrémente automatiquement chaque fois qu'une nouvelle donnée de ville est insérée.
Types de villes telles que grandes villes, villes moyennes et petites villes.

D‘abord créer une table de route(id_route, Nom, Classement, DebutNumero, FinNumero)

```
CREATE TABLE VILLE
(
  idVille INT PRIMARY KEY,
  Nom Varchar2(20) unique not null,
  Classement Varchar2(20) not null,
  Atous Boolean not null,
)
```

idVille	Nom	Classement	Atous
0	Paris	Grand	OUI
1	Montpellier	Petit	NON

Fonction1:existeVille()

Pour un système de navigation, il est essentiel de disposer d’une adresse de localisation complète et précise.

Si l'utilisateur en choisit une autre, il doit entrer le lieu de départ et utiliser les données saisies par l'utilisateur pour faire correspondre les données de la ville dans la base de données.

Si le logiciel ne correspond pas à la ville, vous ne pourrez pas démarrer la navigation. Le système affiche le message "Impossible d'identifier le lieu de départ, veuillez saisir à nouveau votre lieu de départ!".

Si le logiciel correspond à une nouvelle adresse, celle-ci saisira l'adresse historique et sera définie sur le point de départ de la navigation secondaire.

Pseudo-code:

```
existeVille(String Ville，List L): renvoie boolean
                                        //(On verifier si le route est existe dans le tableau)
for v in L：
        if Ville==v
                then print “Confirme”
          retourne true
    else
                print   "La ville que vous avez entrée n'existe pas"
                retourne false
```

break;

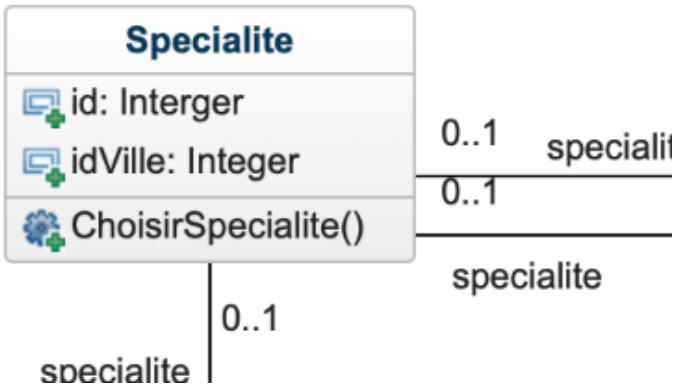
MOAL:

Definition Pre existeVille(ville: String, L:List) ≡ ville!=null ∨ L!= null

Definition Post existeVille(ville: String, L:List, b:Boolean) ≡

{∃V∈ Ville | v.nom==ville}∨b=True}

Classe Specialite:



Un formulaire privilégié doit être créé pour stocker des informations sur les préférences de l'utilisateur, notamment l'identifiant de la ville, le nom de la ville, le type de ville et le fait que la ville possède ou non des attractions touristiques.

Demandez aux identifiants et noms de chaque ville d'être uniques. L'existence de l'identifiant est commode pour établir le contact avec d'autres formulaires et le système l'incrémente automatiquement chaque fois qu'une nouvelle donnée de ville est insérée.

Types de villes telles que grandes villes, villes moyennes et petites villes.

D'abord creer un tableau de Specialite(idSpecialite, idVille)

```
CREATE TABLE Specialite
(
 idSpecialite INT PRIMARY KEY,
```

idVille INT,
FOREIGN KEY(idVille) REFERENCES Ville(idVille)
)

id	idVille
0	0
1	1
2	null

Fonction1 :ChoisirSpecialite()

Les utilisateurs peuvent personnaliser l'itinéraire en fonction de leurs propres préférences. Par exemple, ils peuvent choisir la ville qu'ils souhaitent traverser, la ville qu'ils souhaitent éviter et le nombre de radars. En sélectionnant les préférences de l'utilisateur, nous pouvons indiquer l'itinéraire qui convient le mieux à son idée, par exemple, l'itinéraire avec le moins de radar, l'itinéraire le plus rapide, passera certainement par l'itinéraire d'une ville donnée, en évitant l'itinéraire par une ville donnée. Nous allons donner une fenêtre déroulante à choisir par l'utilisateur: lorsque l'utilisateur sélectionne 1, l'utilisateur sélectionne maintenant la ville qu'il souhaite traverser. Lorsque l'utilisateur sélectionne 2, l'utilisateur représentant sélectionne maintenant une ville qui ne veut pas passer. Lorsque l'utilisateur sélectionne 2, l'utilisateur représentatif veut moins de radar sur le trajet à ce moment.

Pseudo-code:

ChoisirSpecialite(int id, int idVille)

retourner la choix du client

if id=0, idVille=n le client souhaite traverser par la ville n

if id=1, idVille=v le client ne souhaite pas traverser par la ville v

if id=2, le client représentatif veut moins de radar sur le trajet à ce moment

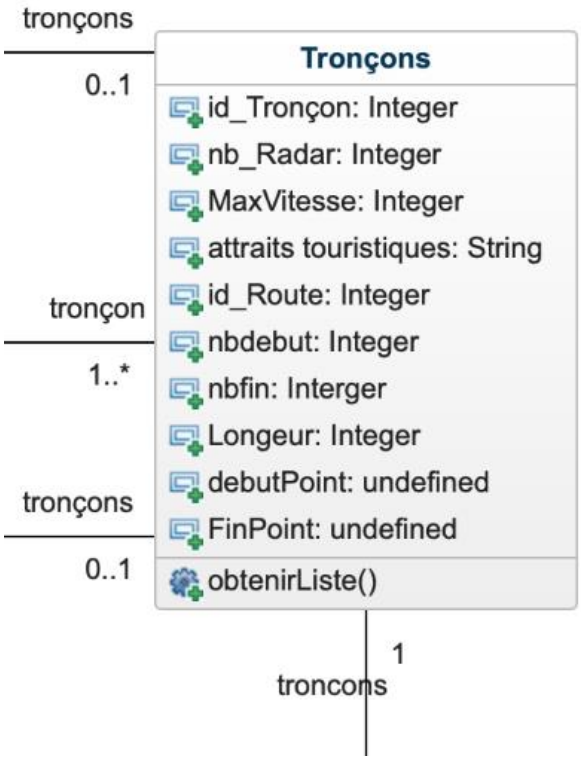
MOAL :

Definition Post choisirSpecialite(id:int, idVille: int, b: Boolean) ≡

{∃s ∈ Specialite | (id==0/\idVille!=null)\/(id==1/\idVille!=null)\/(id==2/\idVille=null)

\/(id=null/\idVille=null)\/b=True}

Classe Tronçon:



Il est nécessaire d'établir un tableau de segment de route pour enregistrer les informations de segment de route, le nombre de radars ID des segments d'information, la limitation de vitesse de chaque segment de route, s'il existe des attractions touristiques dans le segment de route, le numéro de rue au début du segment de route, le numéro de rue à la fin du segment de route et le numéro de chaque segment de route. La longueur, le point de coordonnée au début du segment de route et le point de coordonnée à la fin du segment de route. L'identifiant de chaque chemin doit être unique. L'ID est idéalement situé pour établir le contact avec d'autres formulaires. Le système l'incrémente automatiquement chaque fois qu'un nouveau segment est inséré. Nous calculons notre trajectoire au début et à la fin du segment de route, le client pouvant déterminer le nombre de radars dans la trajectoire en fonction de ses préférences personnelles et calculer le temps de traitement de chaque trajectoire que l'utilisateur peut sélectionner jusqu'à la limite de vitesse de chaque segment. Donner aux utilisateurs une meilleure expérience.

Fonction1: obtenirliste(String Depart, String Destination, String Liste Specialite):

Cette fonction renverra éventuellement une liste. Toutes les tronçons pouvant arriver à la destination apparaîtront dans la liste. La position du point de départ et de la destination sera d'abord convertie en points de coordonnées, et une connexion avec départ et destination sera donnée. La ligne est un double rectangle de diagonales, et au milieu, trouvez tout le tronçon qui peut atteindre la destination depuis le départ, et renvoyez ces tronçon dans une liste et renvoyez-les à creertrajet ().

Pseudo-code:

obtenirliste(String Depart, String Destination, Tuple debutpoint, Tuple finpoint, Int Type, String List Tronçon):

transfererAupoint(position);// //on défini qu'il y a déjà une fonction pour realiser transferer le point depart et destination aux coordonnée
rectangle(tupledepart,tupledestination ,type){

if (tupledepart.x<tupledestination.x&&tupledepart.y<tupledestination.y)||(tupledestination.x<tupledepart.x&&tupledestination.y<tupledepart.y)

p1->position//P1 represente la point a la gauche
p2->position//P2 represente la point a la droit
if tupledepart.x<tupledestination.x then
p1->depart
p2->destination
else
p1->destination
p2->depart
p1.x-=(p2.x-p1.x)/2
p1.y-=(p2.y-p1.y)/2
p2.x+=(p2.x-p1.x)/2
p1.y+=(p2.y-p1.y)/2
then type==0（right up）
return [p1,p2,0]
if (tupledepart.x<tupledestination.x&&tupledepart.y>tupledestination.y)||(tupledestination.x<tupledepart.x&&tupledestination.y>tupledepart.y)

```
if tupledepart.x<tupledestination.x then
p1->depart
p2->destination
else
p1->destination
p2->depart
p1.x-=(p2.x-p1.x)/2
p1.y+=(p2.y-p1.y)/2
p2.x+=(p2.x-p1.x)/2
p1.y-=(p2.y-p1.y)/2
then type==1(left up)
return [p1,p2,1]
}

for i in L :

    if type ==0

        if

(p1.x<=debutpoint.x<finpoint.x<=p2.x&&p1.y>=debutpoint.y>finpoint.y>=p2.y)

        then Tronçon[i]=L[i]

    else if type==1

        if

(p1.x<=debutpoint.x<finpoint.x<=p2.x&&p1.y<=debutpoint.y<finpoint.y<=p2.y)

        then Tronçon[i]=L[i]

return Tronçon
```

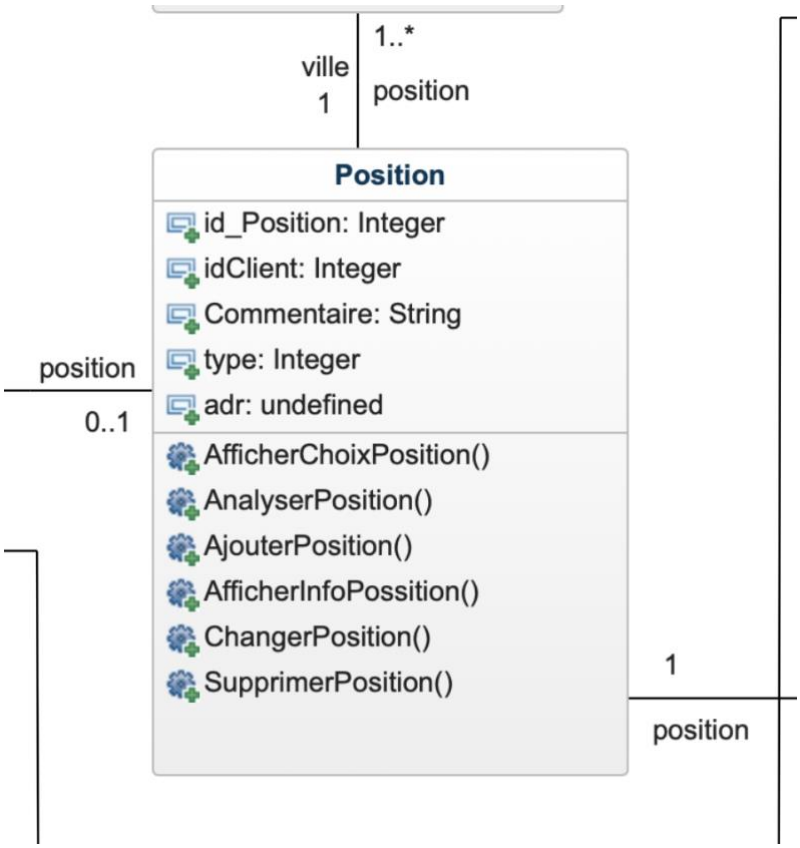
MOAL:

Definition Pre obtenirliste(String Depart, String Destination, Tuple debutpoint, Tuple finpoint, Int Type, String List Tronçon)≡ Depart!=null ∨ Destination!= null

Definition Post obtenirliste(String Depart, String Destination, Tuple debutpoint, Tuple finpoint, Int Type, String List Tronçon)≡ {∃ D1,D2 ∈ Position, ∃ debutpoint,finpoint ∈

Tronçon |
D1.x<=debutpoint.x<finpoint.x<=D2.x&&D1.y>=debutpoint.y>finpoint.y>=D2.y ||
D1.x<=debutpoint.x<finpoint.x<=D2.x&&D1.y<=debutpoint.y<finpoint.y<=D2.y }

Classe Position:



Fonction1:

AffichierChoixPosition():

Une fois que l'utilisateur connecté a stocké les informations personnelles dans le système, y compris l'adresse de la société et l'adresse du domicile, lorsque l'utilisateur clique sur la zone de saisie du lieu de départ et de la destination, la fenêtre déroulante s'affiche. La première adresse de la fenêtre contextuelle est l'adresse du domicile de l'utilisateur. L'adresse est l'adresse de l'entreprise, la troisième est l'historique que l'utilisateur a entré et il ne peut y avoir aucun enregistrement d'adresse.

Si vous n'êtes pas connecté, seul l'historique sera affiché ou aucun enregistrement ne sera enregistré.

Pseudo-code :

```
AffichierChoixPOsition(idClient){
    if(idClient){
```

```
        if(listeHistory){
           afficher();
        }
     }
}
```

MOAL :

position.AfficherChoixPosition(idClient)

Pre-conditions:

idClient !=null

Post-conditions:

∃ c ∈ bd.client, idClient = c.id_Client => result = Afficher liste de position

∀ c ∈ bd.client, idClient !=c.id_Client => result = Rien à afficher

Argument:

idClient

Type des arguments et de retour:

idClient: Integer

listePosition: String

Fonction2 : AnalyserPosition(destination):

Selon la destination entrée par l'utilisateur, les informations de localisation sont recherchées dans la base de données et une liste d'informations d'adresses est renvoyée, laquelle comporte un numéro de route, un nom de route et un nom de ville.

Pseudo-code :

```
AnalysePosition(destination){
     if(destination){
        route = this.getRoute;
        nb = this.getNumero;
        villeNom = this.getVilleNom;
        liste = [nb,route,villeNom] ;
        }
return liste;
```

}

MOAL :

position.AnalysePosition(destination : String)

Pre-conditions:

destination != null

Post-conditions:

∃ r ∈ bd.route, v ∈ bd.ville, route = r.nom ∧ nb= r.FinNumero ∧ villeNom = v.nom

=> liste

∀ r ∈ bd.route, v ∈ bd, route != r.nom v nb !=r.FinNumero v villeNom != v.nom => liste vide = N'a pas trouvé l'addresse.

Argument:

destination

Type des arguments et de retour:

destination: String
route: String
nb: Integer
villeNom:String
liste: list String

Fonction3 :AjouterPosition():

Ajoutez le départ et la destination de l'entrée utilisateur au système et attendez que l'itinéraire soit analysé. Cette fonction a deux paramètres: le premier est l'emplacement, le deuxième est le type, le type est 0 quand c'est le point de départ et le type est 2 quand c'est la destination.

Pseudo-code :

```
AjouterPosition(position, type){
    if( type==0 ){
       depart = position;
    }
    else{
       destination = position;
    }
}
```

MOAL :

Pre-conditions:

position != null ∧ type != null

Post-conditions:

∃ p ∈ bd.position, type = p.type = 0 => p.adr = depart = position

∃ p ∈ bd.position, type = p.type = 1 => p.adr = destination = position

∀ p ∈ bd.position, type != p.type v p.adr != position => message d'erreur

Argument:

position, type

Type des arguments et de retour:

position: String

type: Integer

Fonction4 :AfficherInfoPosition():

Les informations sur l'adresse enregistrée de l'utilisateur s'affichent. Le nom d'utilisateur de l'utilisateur est unique, le paramètre doit uniquement être le nom d'utilisateur.

Pseudo-code :

```
AfficherInfoPosition(nom){
    If(nom){
       while(position){
          affiche();
       }
    }
}
```

MOAL :

Client.AfficherInfoPosition(nom：String)

Pre-conditions:

nom != null

Post-conditions:

∃c ∈ bd.client, p ∈ bd.position, position = p.adr = true => affiche()

∀ c ∈ bd.client, p ∈ bd.position ,p.adr != position => rien à afficher

Argument:

nom

Type des arguments et de retour:

nom: String

Fonction5 :ChangerPosition()

Modifier l'adresse que l'utilisateur a enregistrée. Choix est le type que vous souhaitez modifier. Si la valeur est 0, modifiez l'adresse du domicile. Si c'est le 1, modifiez l'adresse de la société.

Pseudo-code :

```
ChangerPosition(choix,nouveauxString){
    if(choix==0){
       home.adr=nouveauxString;
    }
    if(choix==1){
       work.adr=nouveauxString;
    }
}
```

MOAL :

Pre-conditions:

Choix != null ∧ nouveauxString != null

Post-conditions:

∃ p ∈ bd.position, choix = p.type ∧ nouveauxString = p.adr => modifier réussi

∀ p∈ bd.position, choix != p.type v position != p.adr => message d'erreur

Argument:
choix,
nouveauxString

Type des arguments et de retour:
choix: Integer
nouveauxString: String

Fonction6 :SupprimerPosition():

Supprimez les informations d'adresse que l'utilisateur a enregistrées.

Pseudo-code :

```
SupprimerPosition(nom,position){
    if(nom){
       if(position){
         supprimer();
       }
    }
}
```

MOAL :

Client.SupprimerPosition(nom : String, position:String)

Pre-conditions:
nom != null ∧ position != null

Post-conditions:

∃ c ∈ bd.client, p ∈ bd.position, nom=c.nom ∧ position = p.adr => Supprime réussi

∀ c ∈ bd.client, p ∈ bd.position , nom=c.nom v position != p.adr=> message d'erreur

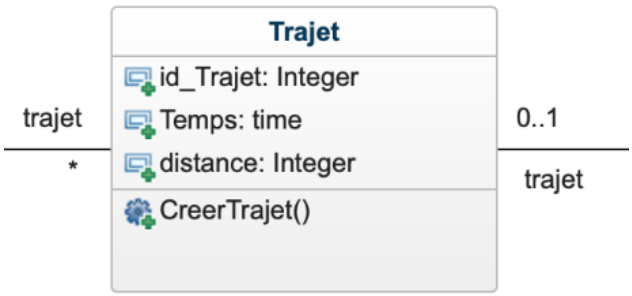
Argument:
nom
position

Type des arguments et de retour:

nom: String

position: String

Classe Trajet:



Une table de chemins est nécessaire pour enregistrer les informations de chemin, y compris l'ID du chemin, la durée estimée dont l'utilisateur a besoin pour terminer le chemin, la distance totale du chemin, etc. L'identifiant de chaque chemin doit être unique. Son existence facilite le contact avec d'autres formulaires. Le système l'incrémente automatiquement chaque fois qu'un nouveau chemin est inséré. Le temps et la distance du trajet sont pratiques pour permettre à l'utilisateur de sélectionner le trajet avec la distance la plus courte ou l'itinéraire le plus rapide vers la destination.

D'abord créer une table de Trajet(idTrajet, Temps,distance)

```
CREATE TABLE Trajet
(
 idTrajet INT PRIMARY KEY,
 Temps VARCHAR2(20) NOT NULL,
 distance INT NOT NULL
)
```

idTrajet	Temps	Distance
0	8:30:20	1100
1	7:50:30	1100

Fonction1: CreerTraget(sinit,f choix h):

Cette fonction appelle la fonction obtenirListe () dans Tronçons, qui est sélectionnée en fonction des préférences de l'utilisateur, et enfin le chemin de navigation que l'utilisateur doit parcourir. CreerTrajet (Départ de chaîne, Destination de chaîne, Liste de chaînes spéciale) Cette fonction filtrera la liste des tronçons renvoyés par obtenirListe () et atteindra éventuellement la destination, connectera tout le tronçon en premier, et retournera finalement une liste de chaînes finale, puis retournera La liste est le chemin final du point de départ à la destination pour satisfaire les préférences de l'utilisateur.

Pseudo-code:

```
Fonction CreerTraget(sinit,fChoix,h){
    DejaTronçon = null; Ville :{sinit}
    g(sinit)=0
    f(sinit)=h(sinit)//f=g+h, h:distance entre deux villes (distance de tronçon) g:le
    distance plus court logiquement entre deux villes
    tant que (Ville !=0) faire
    n=choixFMin(Ville)
    si estTerminal(n) alors retourner constuireSolution(n,Pere)// un Trajet constuire tant
    qu'il arriver a destination

    sinon
    supprimer(n,Ville)
    ajouter(n,DejaTronçon)
    pour tous s contient successeurs(n) faire
        si s ne contient pas DejaTronçon union    Ville ou alors
            Pere(s) = n ;g(s) =g(n) + cout(n,s); f(s) = g(s)+h(s)
            ajouter(s,Ville)
        sinon
```

si g(s) >g(n) +cout (n,x) alors //meilleur chemin jusqu'a s
Pere(s) =n; g(s)=g(s)+ cout(n,s);f(s)=g(s)+h(s)

retourner ECHEC
}

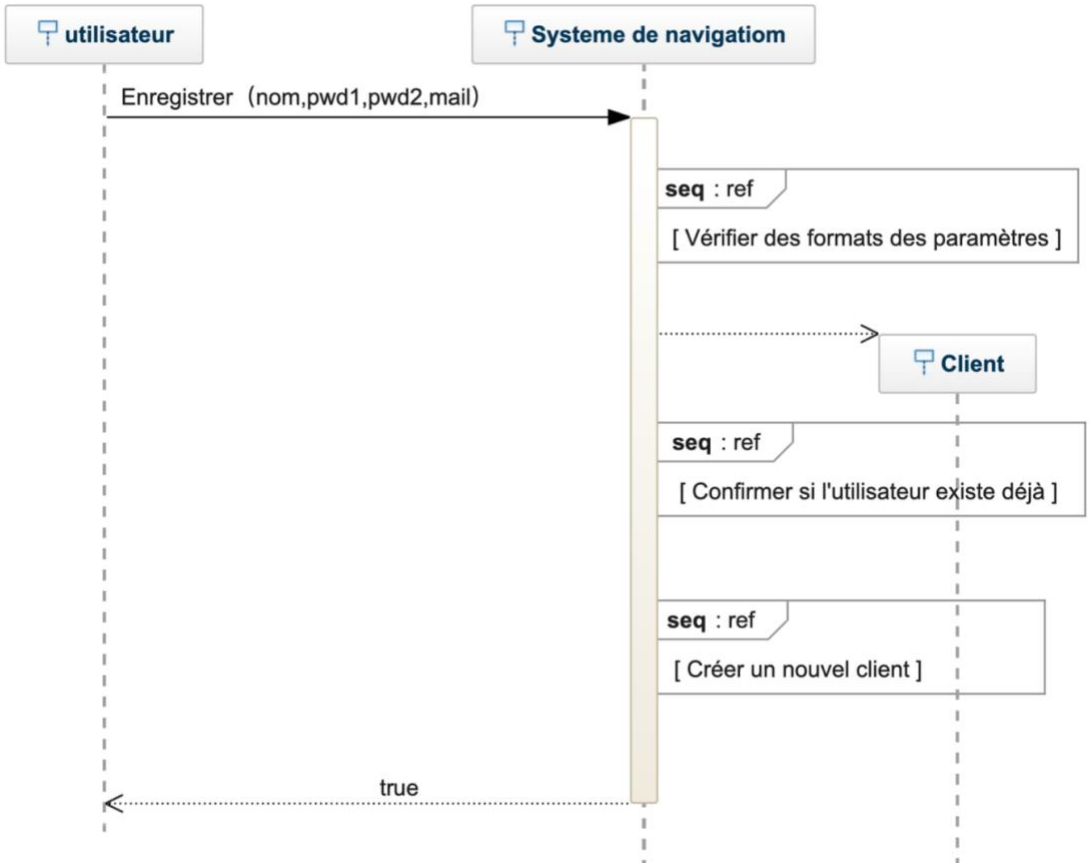
·Diagramme de séquence·

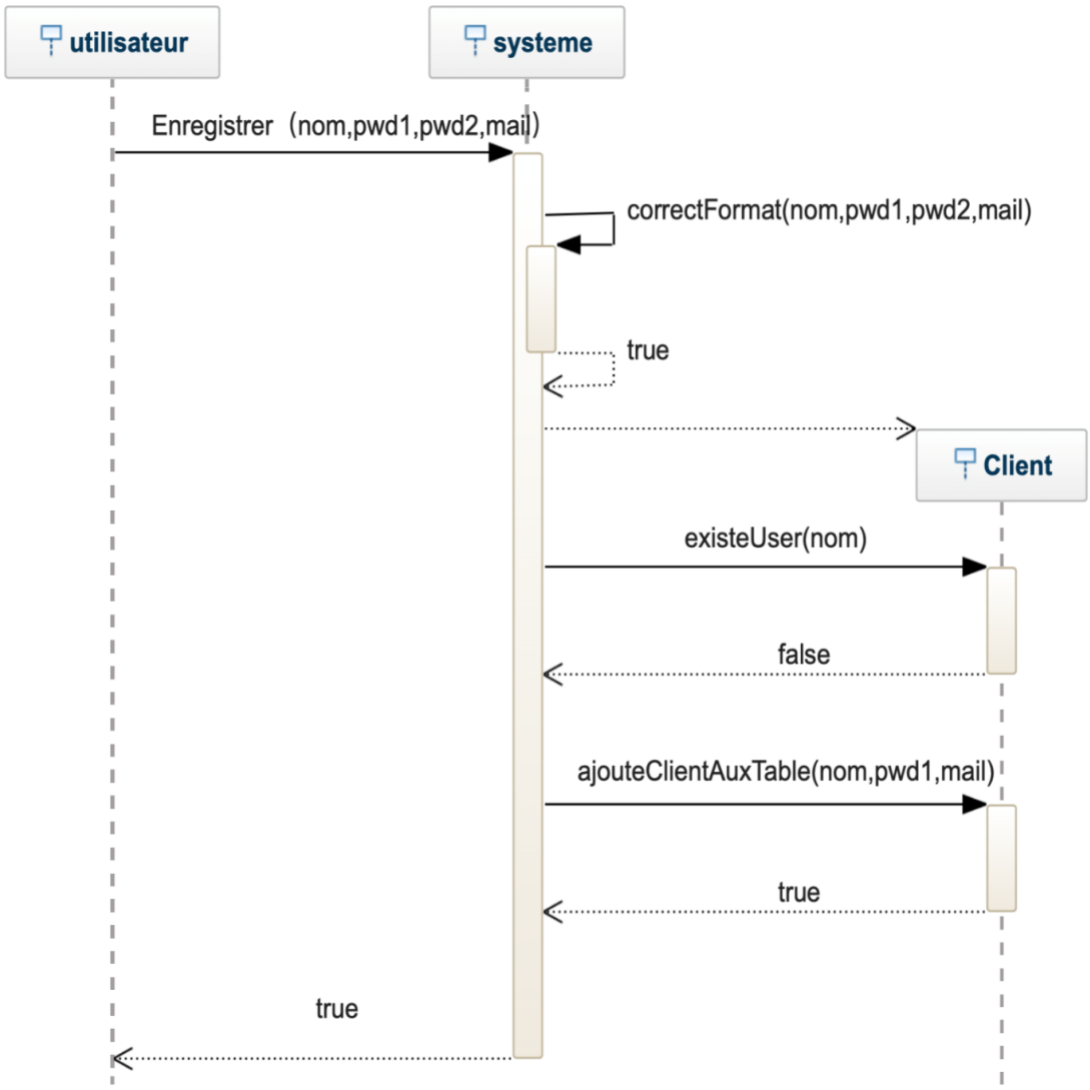
Fonctionnalité1:

Implémenter l'enregistrement de l'utilisateur:

Séquence 1:Enregistrer(nom,pwd1,pwd2,mail):

L'utilisateur doit entrer son nom d'utilisateur, son mot de passe et son mot de passe de vérification, ainsi que son adresse électronique pour l'enregistrement. Une fois les exigences remplies, le système effectue un audit du système de différentes conditions. Une fois l'approbation approuvée, il est nécessaire de mettre en œuvre l'insertion des informations utilisateur dans la table user.


utilisateur
Systeme de navigatiom
Enregistrer（nom,pwd1,pwd2,mail)
seq : ref
[Vérifier des formats des paramètres]
Client
seq : ref
[Confirmer si l'utilisateur existe déjà]
seq : ref
[Créer un nouvel client]
true

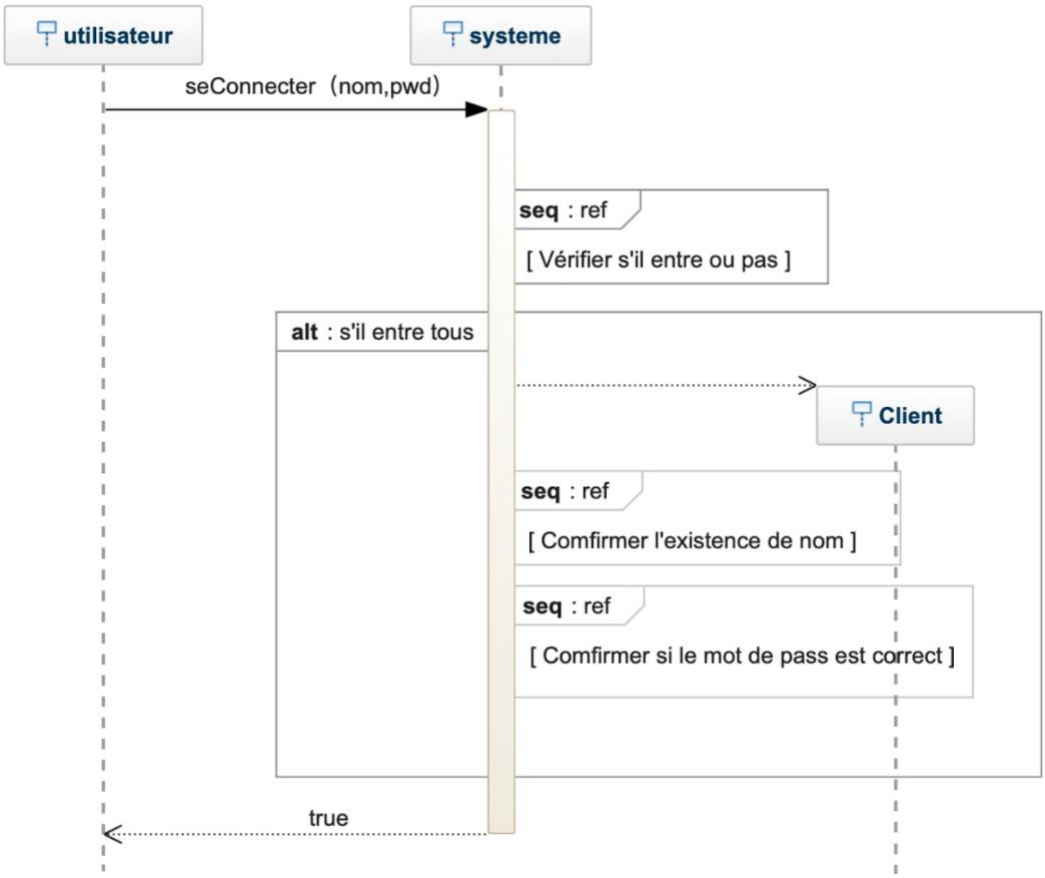

utilisateur
systeme
Enregistrer (nom,pwd1,pwd2,mail)
correctFormat(nom,pwd1,pwd2,mail)
true
Client
existeUser(nom)
false
ajouteClientAuxTable(nom,pwd1,mail)
true
true

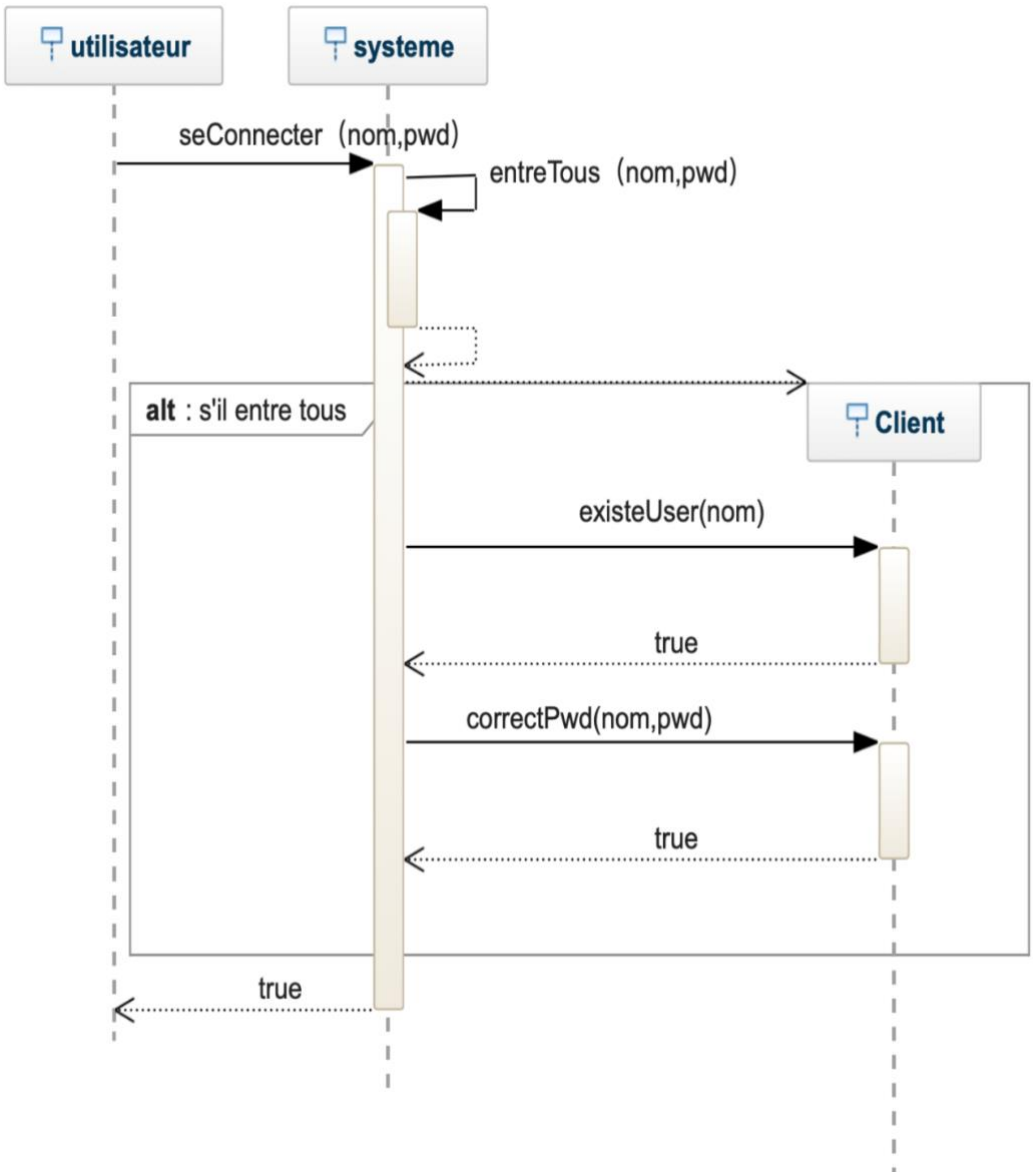
Fonctionnalité2:

Pour tout type d'utilisateurs (membres et visiteurs), tant que le système de navigation est installé, nous leur permettons de se connecter pour accéder à la page principale de l'itinéraire de recherche pour la navigation.

Séquence2:seConnecter (nom, pwd):

L'utilisateur peut entrer son nom d'utilisateur et son mot de passe pour se connecter à l'utilisateur membre, mais cela n'est pas nécessaire. Il peut également ne rien entrer en même temps, se connecter directement en tant que visiteur et accéder à l'application. Lors de la mise en œuvre de la fonction, l'audit du mot de passe du compte utilisateur doit être inclus. Une fois l'audit passé, vous pouvez vous connecter et le système doit appeler automatiquement les informations de l'utilisateur.




utilisateur
systeme
seConnecter (nom,pwd)
entreTous (nom,pwd)
alt : s'il entre tous
Client
existeUser(nom)
true
correctPwd(nom,pwd)
true
true

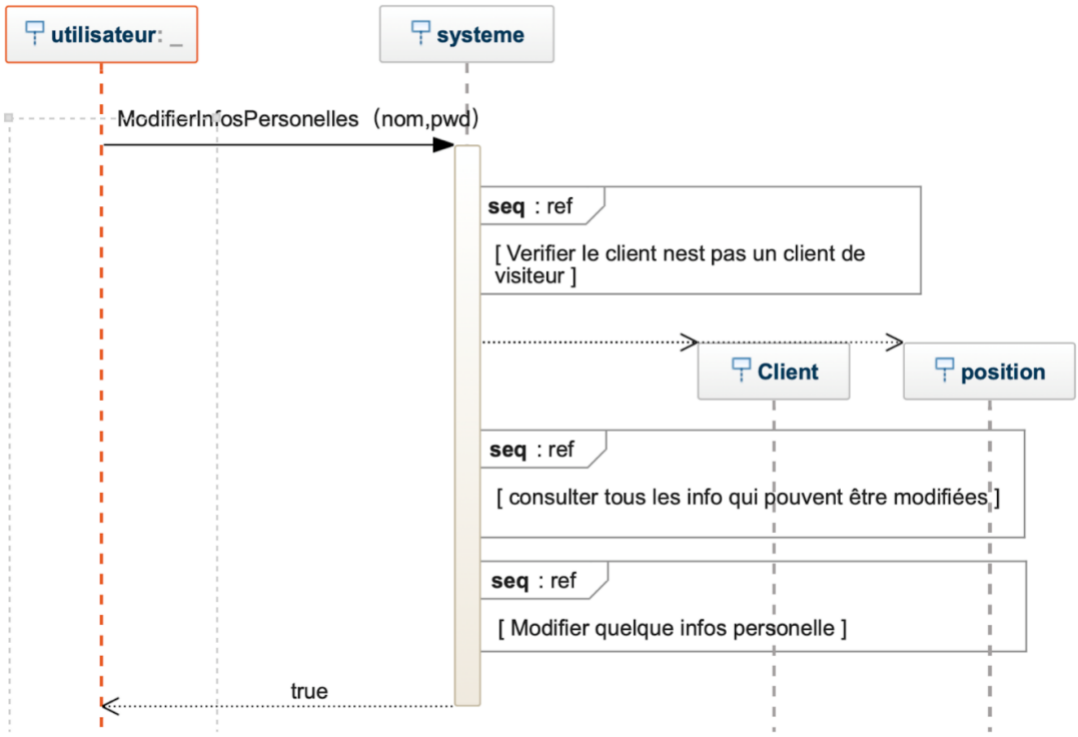
Fonctionnalité3:

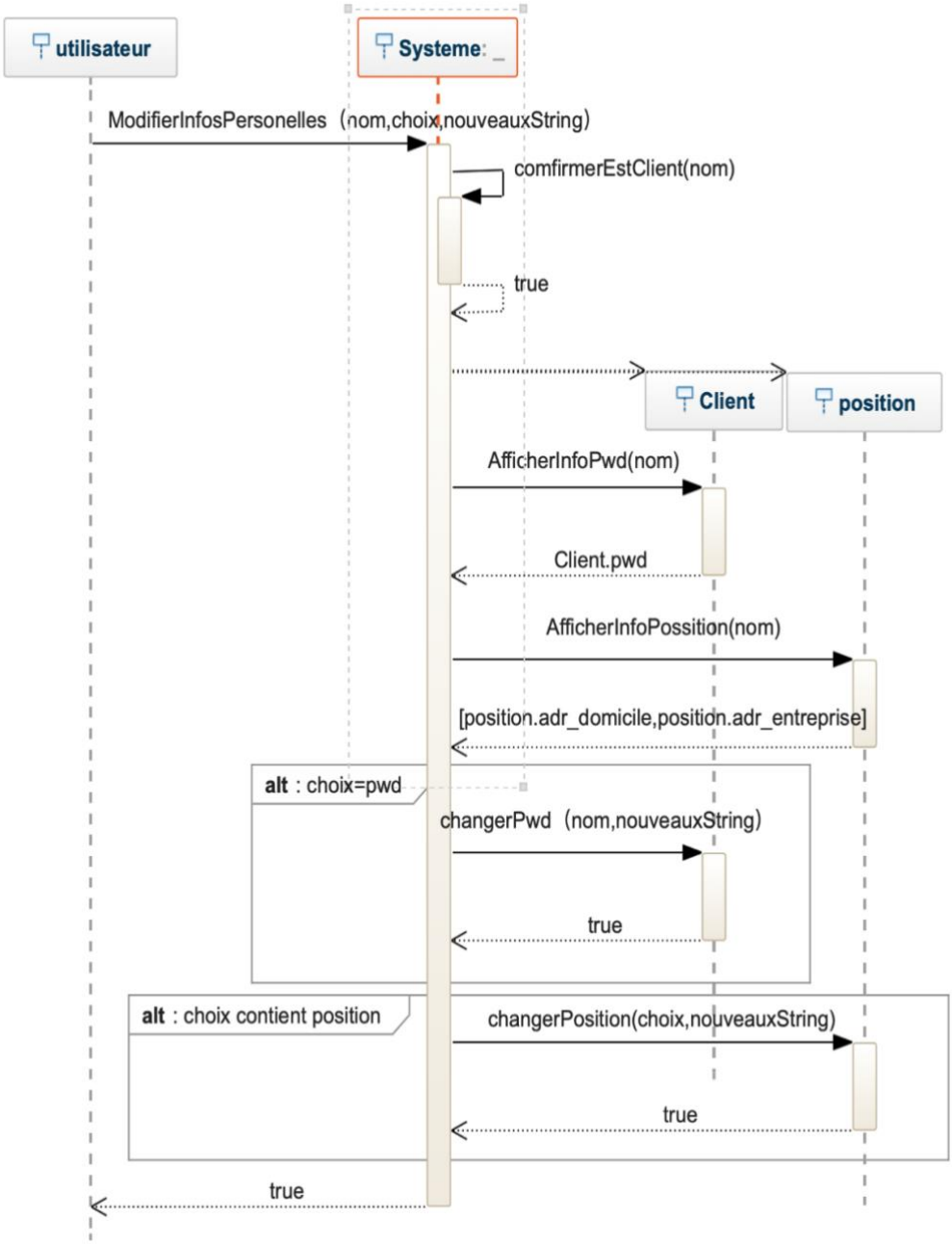
Implémenter la modification des informations personnelles par les utilisateurs membres:

Séquence 3:

ModifierInfo(nom,choix,nouveaustring[],email,action):

Il est obligatoire que seuls les utilisateurs qui se connectent puissent visualiser les informations personnelles. Les opérations pouvant être effectuées sont les suivantes: Infos personnelles, Modifier et supprimer les adresses, Modifier le mot de passe.




utilisateur
Systeme: _
ModifierInfosPersonelles（nom,choix,nouveauxString）
comfirmerEstClient(nom)
true
Client
position
AfficherInfoPwd(nom)
Client.pwd
AfficherInfoPossition(nom)
[position.adr_domicile,position.adr_entreprise]
alt : choix=pwd
changerPwd（nom,nouveauxString）
true
alt : choix contient position
changerPosition(choix,nouveauxString)
true
true

Fonctionnalité4:

Séquence 4:Optimiser les trajets

Entrez l'adresse du départ et de la destination, sélectionnez Préférence et générez le trajet.

En tant qu'utilisateur avec ce compte système:

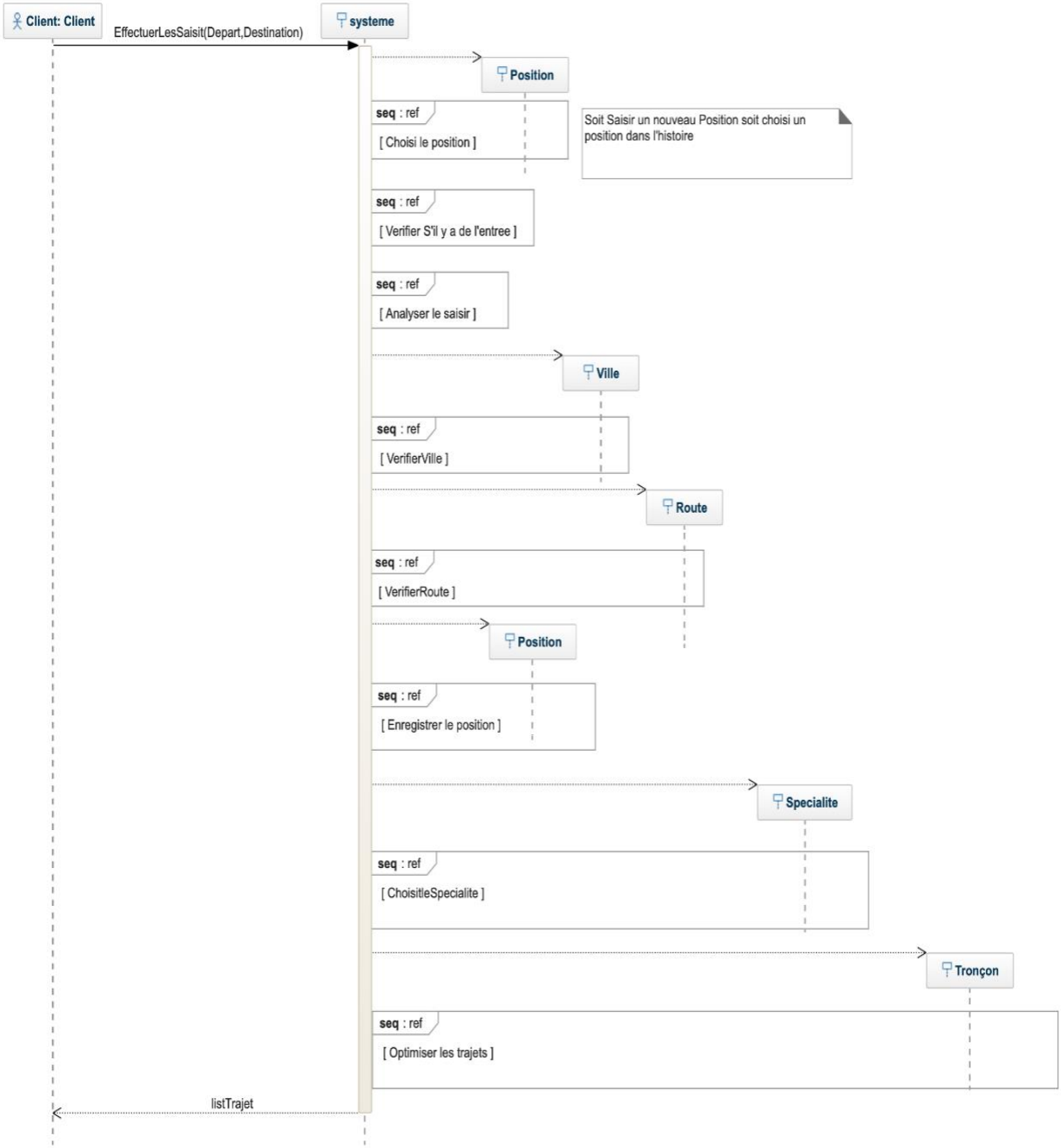
Une fois que l'utilisateur a stocké les informations personnelles dans le système, y compris l'adresse de la société et l'adresse du domicile, lorsque l'utilisateur clique sur la zone de saisie du lieu de départ et de la destination, la fenêtre déroulante s'affiche. La première adresse de la fenêtre contextuelle est l'adresse du domicile de l'utilisateur et la deuxième adresse. Pour l'adresse de la société, la troisième commence par l'historique saisi par l'utilisateur. Si l'utilisateur doit entrer une nouvelle adresse, il en sélectionne une autre. Lorsque l'utilisateur a entré l'adresse, le système reconnaît automatiquement si l'adresse saisie par l'utilisateur est correcte, par exemple si la ville existe, si la route existe et si le nom de la route est correct. S'il s'agit de la bonne adresse, cette adresse sera stockée dans la table de l'utilisateur, le type d'origine est 0, le type de destination est 1 et cette adresse sera utilisée comme point de départ / destination de cette navigation.

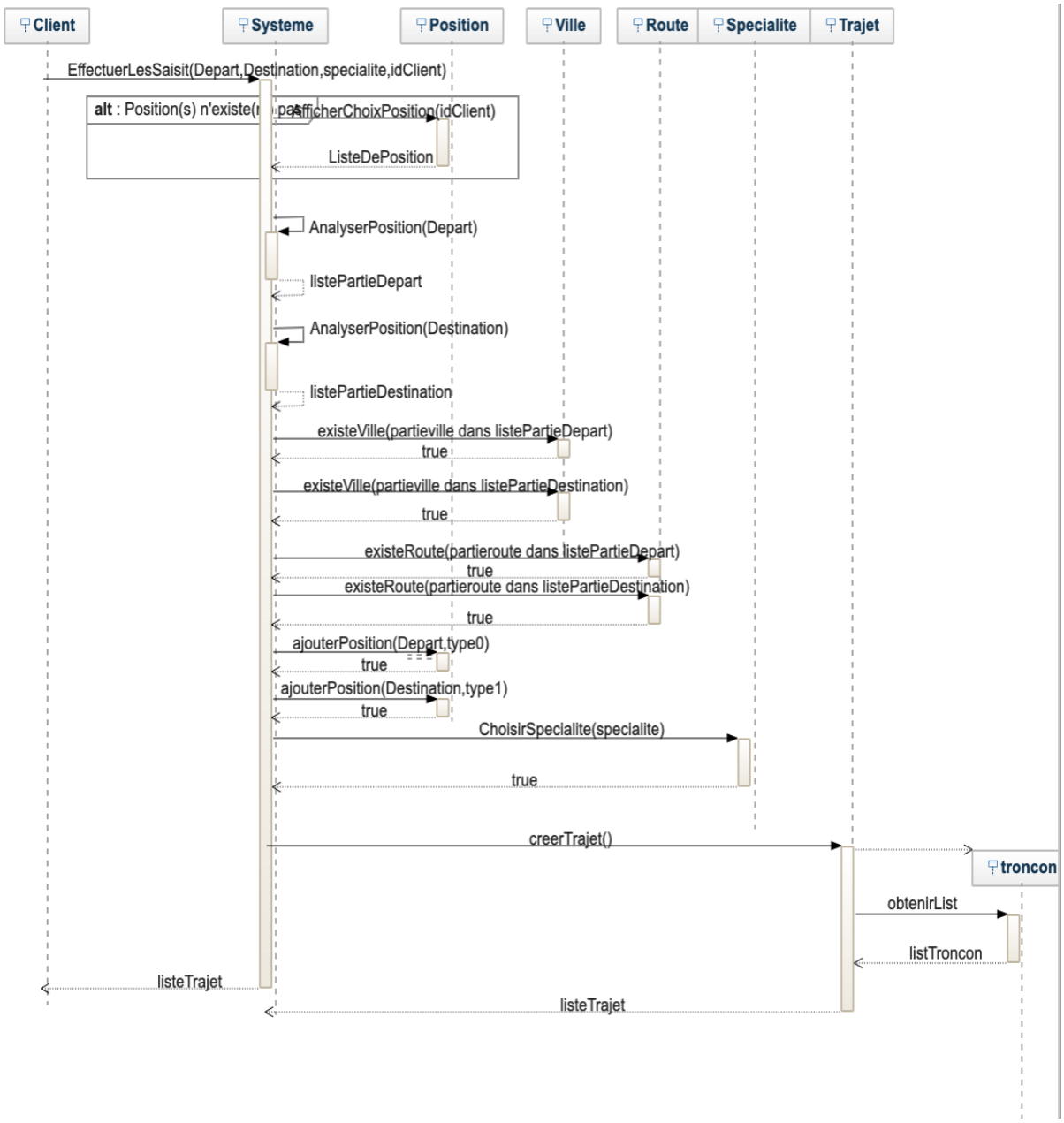
En tant que touriste de ce système:

Lorsque le visiteur clique sur la zone de saisie du point de départ dans le système, la fenêtre contextuelle en incrustation apparaît et l'historique s'affiche dans la fenêtre contextuelle.Il peut y avoir des enregistrements historiques d'autres visiteurs dans l'historique, car les informations personnelles contenues dans le système ne constitueront pas l'identité du touriste. Protégé. Les visiteurs saisissent le départ et la destination et le système vérifie si les informations saisies par le visiteur sont correctes, par exemple si la ville existe, si la route existe et si le nom de la route est correct. S'il s'agit de la bonne adresse, cette adresse sera stockée dans la table de l'adresse du visiteur, le type d'origine est 0, le type de destination est 1 et cette adresse sera utilisée comme départ / destination de cette navigation.

L'utilisateur et le visiteur doivent ensuite choisir leurs propres préférences pour le chemin de navigation, telles que les exigences de limitation de vitesse, les exigences de radar, les exigences de feux de circulation, les exigences de ville ou de ville à éviter.

Le système sélectionnera le chemin de navigation qui correspond le mieux aux pensées de l'utilisateur en fonction de ses préférences.

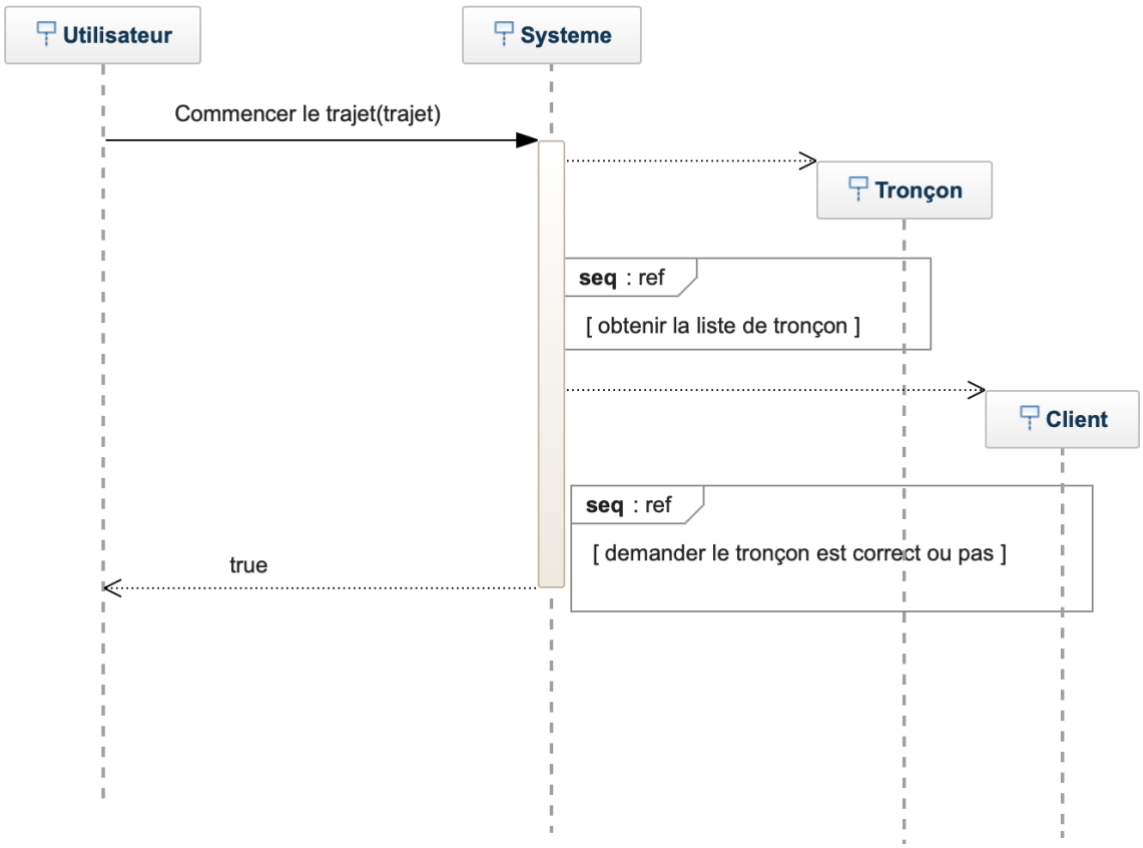

Client: Client
EffectuerLesSaisit(Depart,Destination)
systeme
Position
seq : ref
[Choisi le position]
Soit Saisir un nouveau Position soit choisi un position dans l'histoire
seq : ref
[Verifier S'il y a de l'entree]
seq : ref
[Analyser le saisir]
Ville
seq : ref
[VerifierVille]
Route
seq : ref
[VerifierRoute]
Position
seq : ref
[Enregistrer le position]
Specialite
seq : ref
[ChoisitleSpecialite]
Tronçon
seq : ref
[Optimiser les trajets]
listTrajet

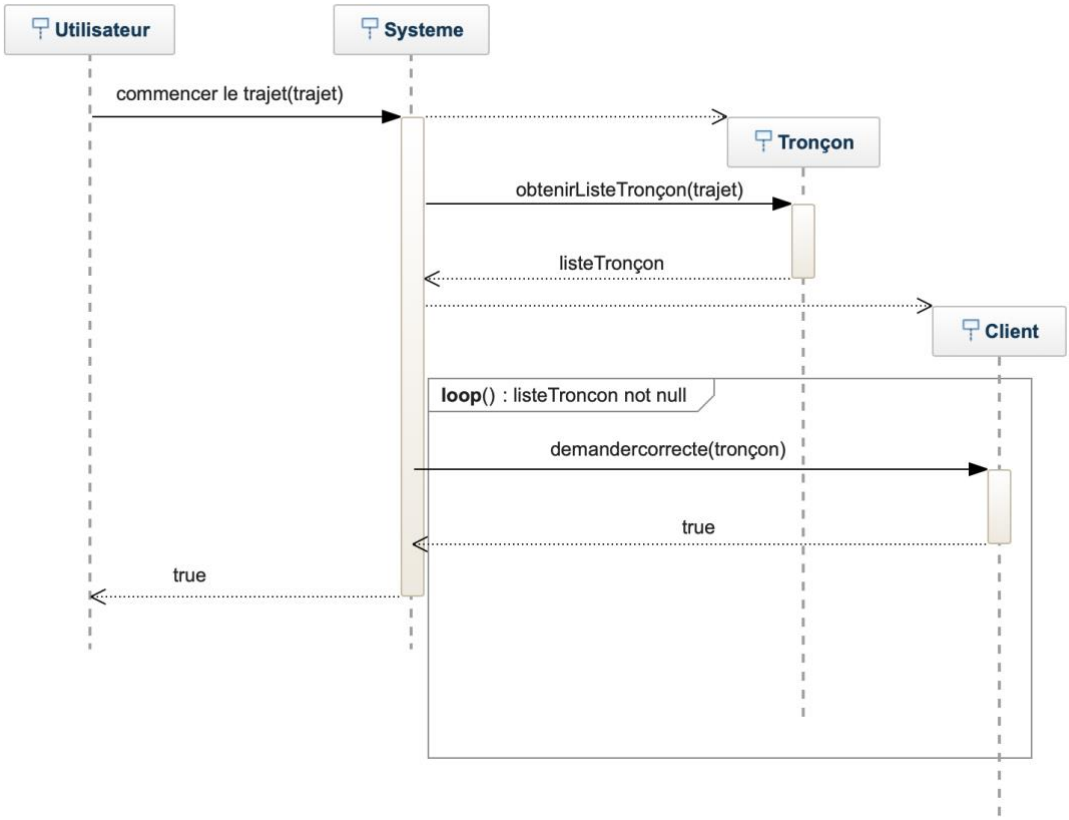

Client
Systeme
Position
Ville
Route
Specialite
Trajet
EffectuerLesSaisit(Depart,Destination,specialite,idClient)
alt : Position(s) n'existe(
afficherChoixPosition(idClient)
ListeDePosition
AnalyserPosition(Depart)
listePartieDepart
AnalyserPosition(Destination)
listePartieDestination
existeVille(partieville dans listePartieDepart)
true
existeVille(partieville dans listePartieDestination)
true
existeRoute(partieroute dans listePartieDepart)
true
existeRoute(partieroute dans listePartieDestination)
true
ajouterPosition(Depart,type0)
true
ajouterPosition(Destination,type1)
true
ChoisirSpecialite(specialite)
true
creerTrajet()
troncon
obtenirList
listTroncon
listeTrajet
listeTrajet

Fonctionnalité5 :

Séquence 5: choisir trajet

Une fois que l'utilisateur a sélectionné l'itinéraire et a commencé à naviguer, le système recherche tous les tronçons en fonction de l'itinéraire et demande ensuite si l'utilisateur arrive après avoir terminé un tronçon. L'utilisateur peut choisir de répondre par oui ou non.

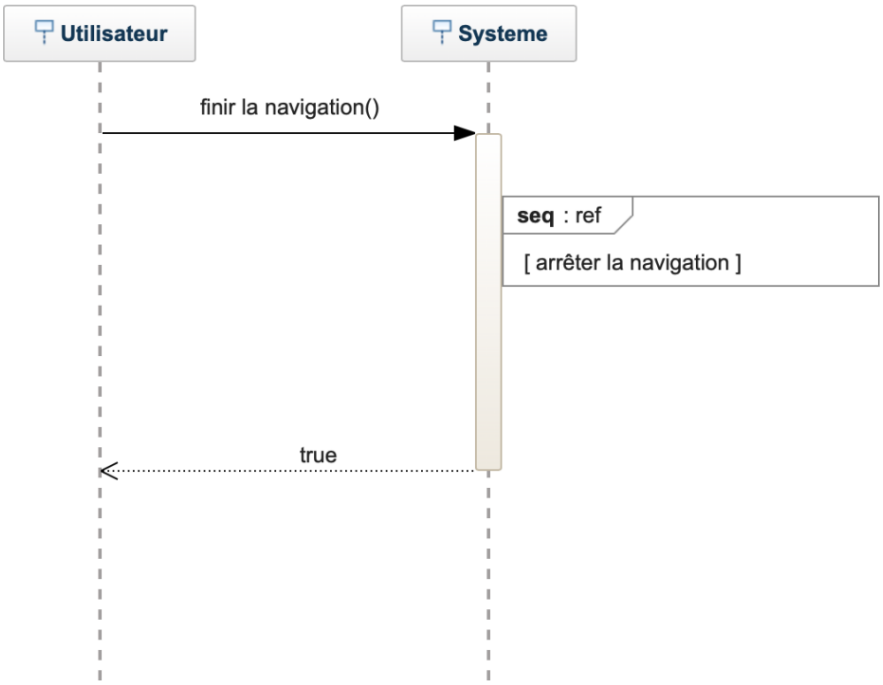


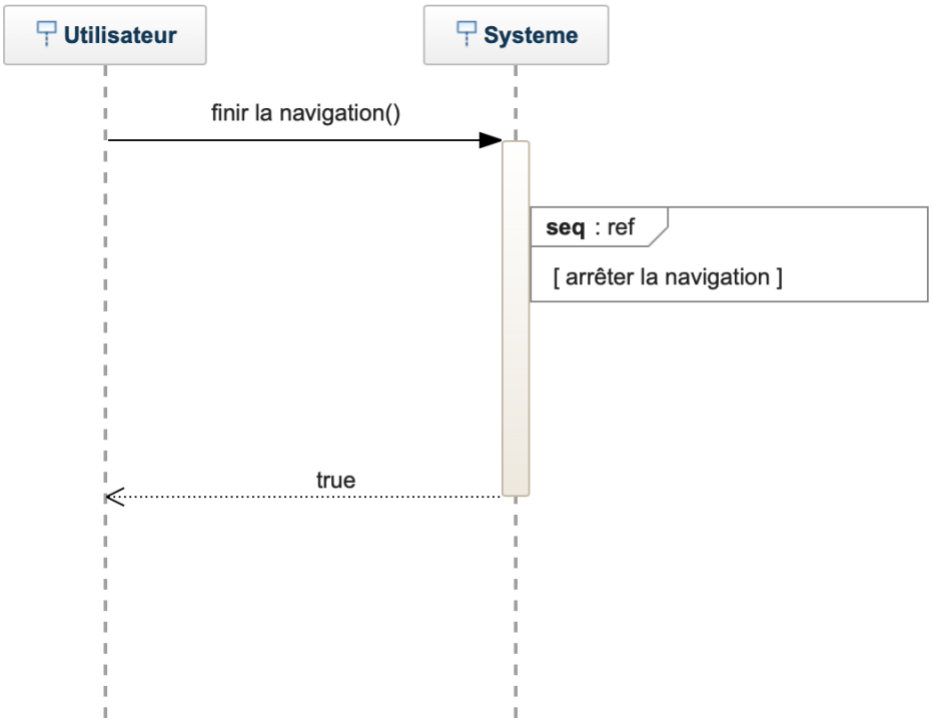

Utilisateur
Systeme
commencer le trajet(trajet)
Tronçon
obtenirListeTronçon(trajet)
listeTronçon
Client
loop() : listeTroncon not null
demandercorrecte(tronçon)
true
true

Fonctionnalité6 :

Séquence 6: finir la navigation

L'utilisateur décide d'arrêter la navigation et le système lance l'interface de navigation.

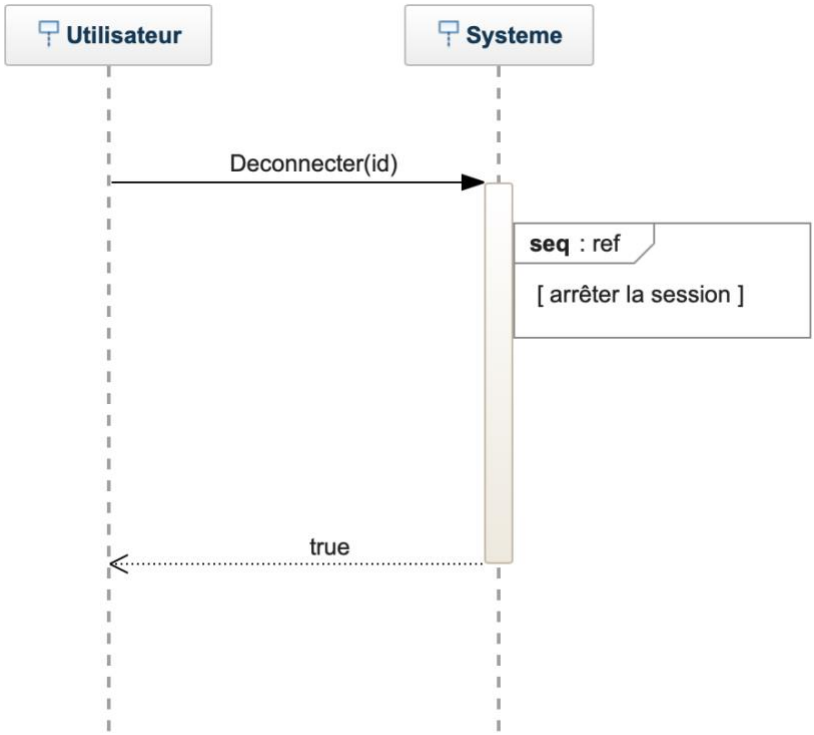


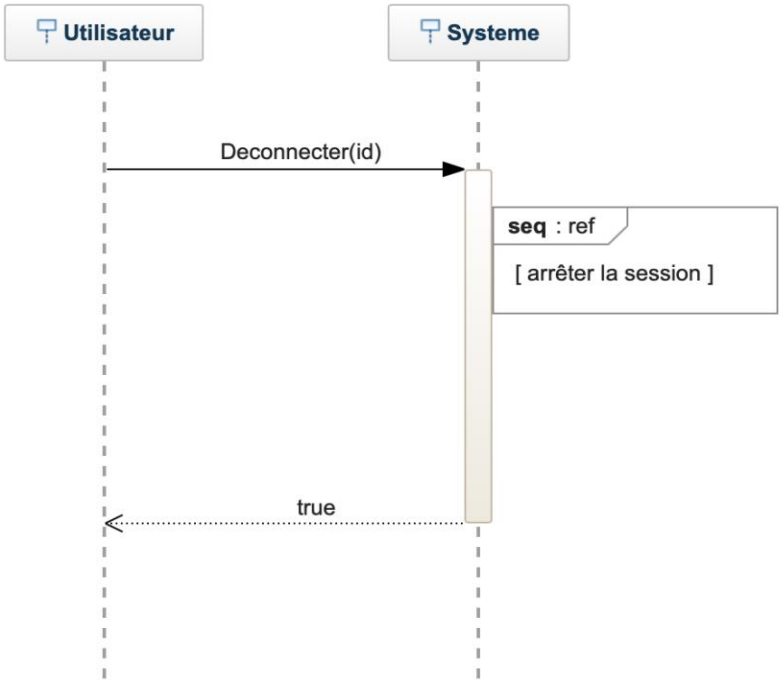


Fonctionnalité7 :

Séquence 7: déconnecter

Lorsque l'utilisateur se déconnecte, le système ferme la session de l'utilisateur actuel et revient à l'interface initiale.





·Conclusion·

La tâche de cette partie concerne le concept du projet. Commençons par l'analyse du diagramme de cas d'utilisation. Nous devons déterminer quelle fonctionnalité doit être conçue pour l'utilisateur. Ensuite, du point de vue du diagramme de classe, réfléchissez au type d'opération à ajouter à chaque classe, puis Continuez à ajouter une nouvelle opération basée sur la relation entre les tables.

Après avoir ajouté l'opération, concevez le pseudo-code en fonction de l'opération et réfléchissez à l'algorithme.

Ensuite, utilisez MOAL pour normaliser les contraintes.

Enfin, écrivez digramme de séquence pour exprimer graphiquement la point de vue, montrant la relation entre chaque fonction dans des tableaux différents, les paramètres et les valeurs de retour de la fonction, le tableau de chaque fonction, etc.