



## Documentation technique

Tick&Trail

## Informations

Nom du projet Tick&Trail

Type de document Documentation technique

Date 11/12/2022

Version 3.0

Mots- clés Train – Logiciel – Technologies – Diagrammes –  
Fonctionnement – Bugs – Interactions

### Auteurs

- Yasin AY – [yasin.ay@etu.univ-tours.fr](mailto:yasin.ay@etu.univ-tours.fr)
- Yanis Chataigner – [yanis.chataigner@etu.univ-tours.fr](mailto:yanis.chataigner@etu.univ-tours.fr)
- Leo Andert – [leo.andert@etu.univ-tours.fr](mailto:leo.andert@etu.univ-tours.fr)
- Nawfal Naciredine – [nawfal.naciredine@etu.univ-tours.fr](mailto:nawfal.naciredine@etu.univ-tours.fr)
- Antoine Montaut – [antoine.montaut@etu.univ-tours.fr](mailto:antoine.montaut@etu.univ-tours.fr)
- Manon Cattaneo – [manon.cattaneo@etu.univ-tours.fr](mailto:manon.cattaneo@etu.univ-tours.fr)

# Documentation technique

## Table des matières

1) Résumé du document .....	4
2) Rappel sur le fonctionnement de l'application.....	5
a. Description du logiciel .....	5
3) Utilisateur.....	6
a. Technologie utilisée.....	6
b. Diagramme de classe.....	7
c. Modèle de donnée .....	8
d. Diagramme d'activité.....	9
e. Diagramme de séquence	
i. Manipulation User.....	10
ii. Interaction.....	11
4) Tests	
a. Tests	
5) Annexe	
a. Table des illustrations	

## 1 – Résumé du document

Ce document est la documentation technique officielle de l'application Tick&Trail.

Il est divisé en plusieurs parties :

- rappel sur le fonctionnement
- les technologies utilisent
- La documentation technique liée a l'utilisateur
- Les bugs connus

## 2 - Rappel sur le fonctionnement de l'application

### 2.a) Description du logiciel

Notre projet a comme objectifs de fournir aux clients un moyen de réserver des trajets en train depuis une borne dédiée disponible dans la gare de la ville. Le logiciel, grâce à une interface moderne et ergonomique, permet de gérer efficacement et simplement les réservations de train.

## 3 – Utilisateur

### 3.a) Technologie utilisée

#### **JavaFx**

Utilisation de java fx Etend la technologie Java grâce à l'utilisation de n'importe quelle bibliothèque Java au sein d'une application JavaFX. Permet la création et le déploiement d'applications d'apparence moderne avec du contenu riche. Étend le niveau de prise en charge du côté client du navigateur, permettant aux développeurs d'écrire l'intégralité de l'application dans un seul langage Java ou base de code, sans nécessiter une interface en Swing.

#### **Scène Builder**

Scène Builder est un outil interactif de conception d'interface graphique pour JavaFX. Il permet de créer des interfaces utilisateurs rapidement et sans avoir besoin de coder ; il en résulte des fichiers au format FXML qui sont ensuite chargés par le programme pour afficher une interface graphique à l'utilisateur. Bien que la prise en main soit facile, il est plus compliqué de setup nos environnements de travail qu'avec Swing.

#### **Maven**

Apache Maven est un outil de construction, et il fait la tâche tout comme Ant, qui est encore un outil de construction extraordinaire. Il s'agit d'un outil de gestion de projet logiciel Maven simplifie et standardise le processus de construction du projet. Il gère la collaboration d'équipe, la compilation, la distribution, la documentation et les tâches séparées de manière transparente. Maven augmente la réutilisabilité et prend également en charge la plupart des tâches liées à la construction.

#### **Junit**

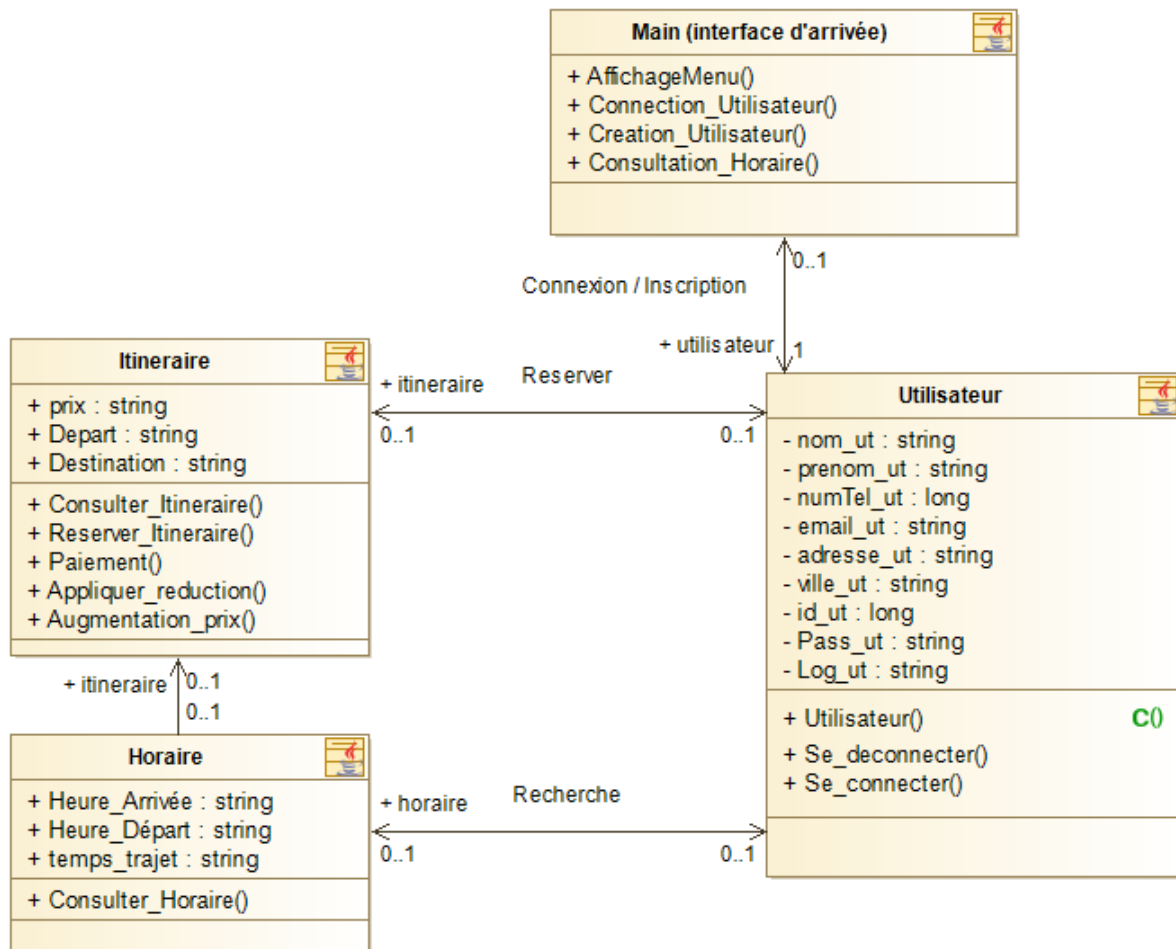
JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression.

Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests.

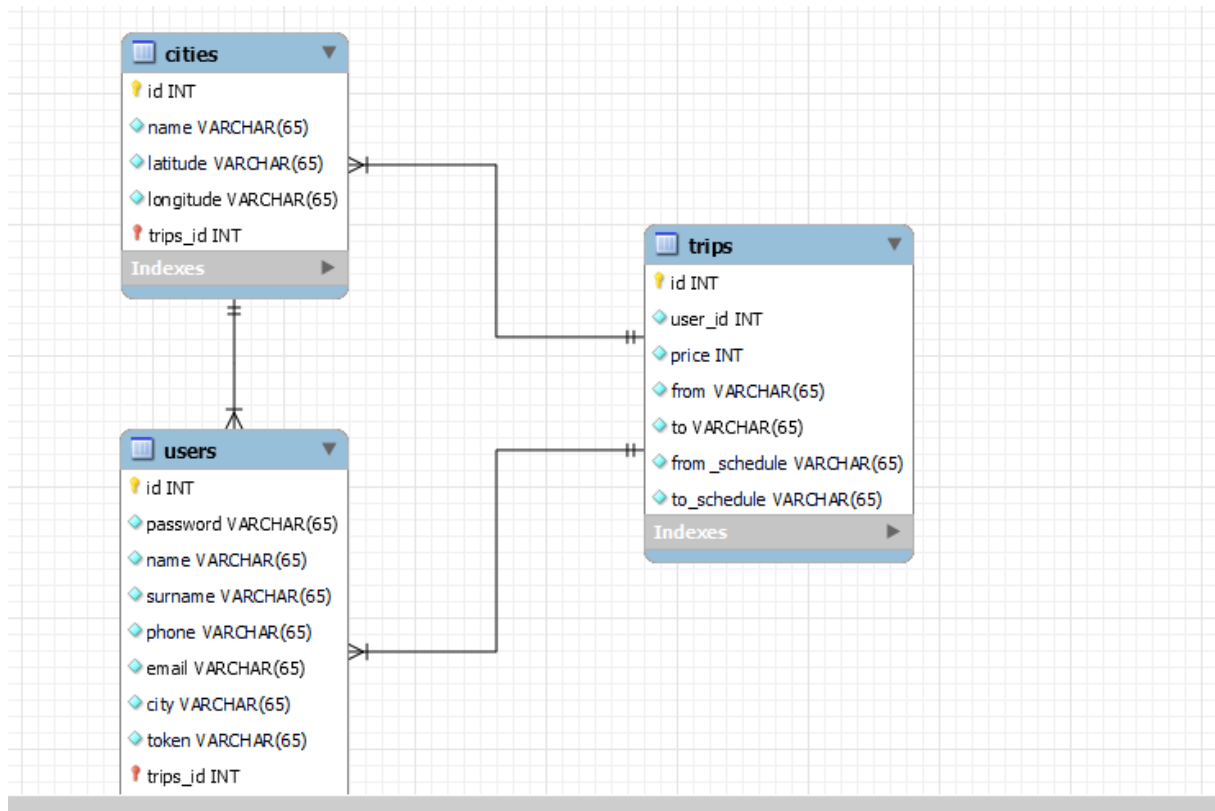
Ces cas de tests exécutent les tâches suivantes :

- création d'une instance de la classe et de tout autre objet nécessaire aux tests
- appel de la méthode à tester avec les paramètres du cas de tests
- comparaison du résultat attendu avec le résultat obtenu : en cas d'échec, une exception est levée

### 3.b) Diagramme de classe



### 3.c) Modèle de donnée

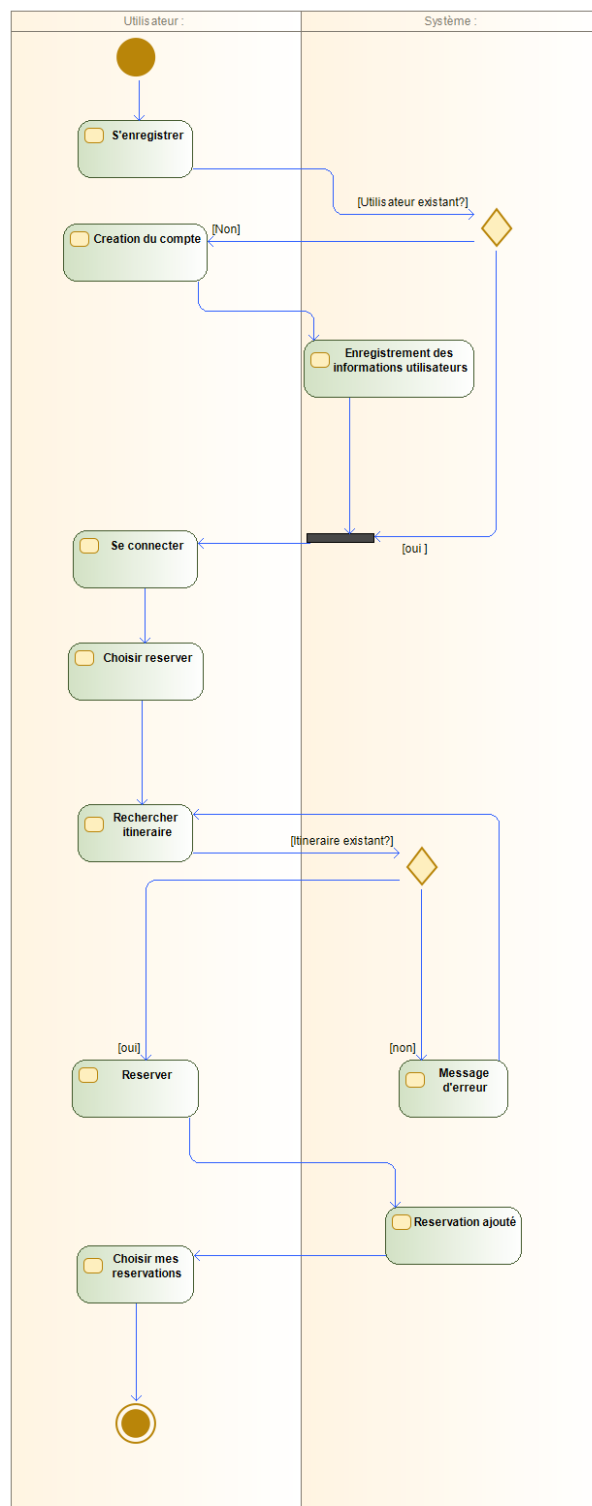


3 tables :

- Users : Décrit les utilisateurs ainsi que leurs attributs
- Trips : Décrit les voyages ainsi que les attributs
- Cities : Décrit les villes ainsi que ses attributs



### 3.d) Diagramme d'activité



Le diagramme d'activité décrit les étapes de l'inscription à la réservation.

### 3.e.i) Diagramme de séquence (Manipulation User)

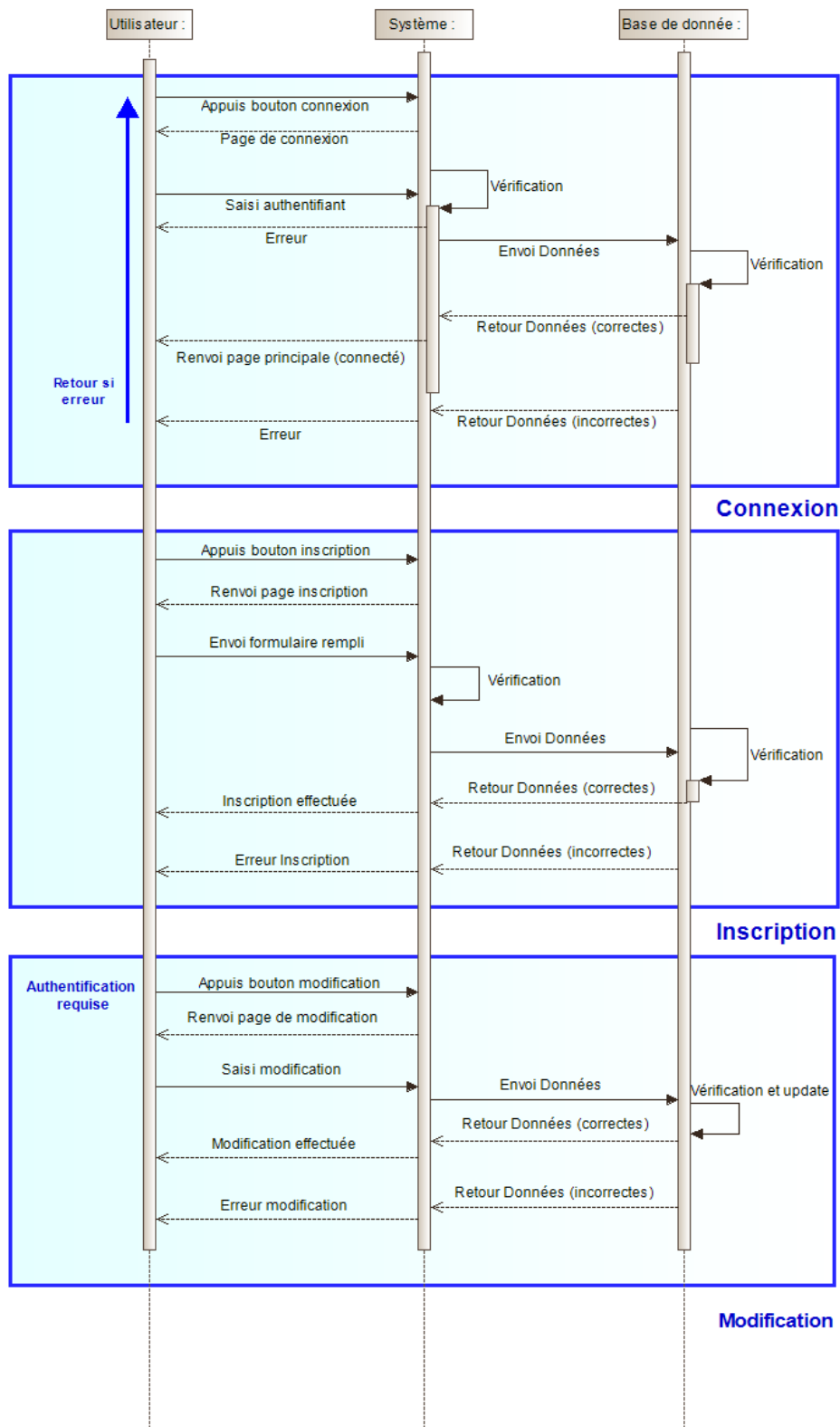


Diagramme de séquence expliquant l'inscription ainsi que la connexion avec les modifications du profil.

### 3.e.ii) Diagramme de séquence (Interaction)

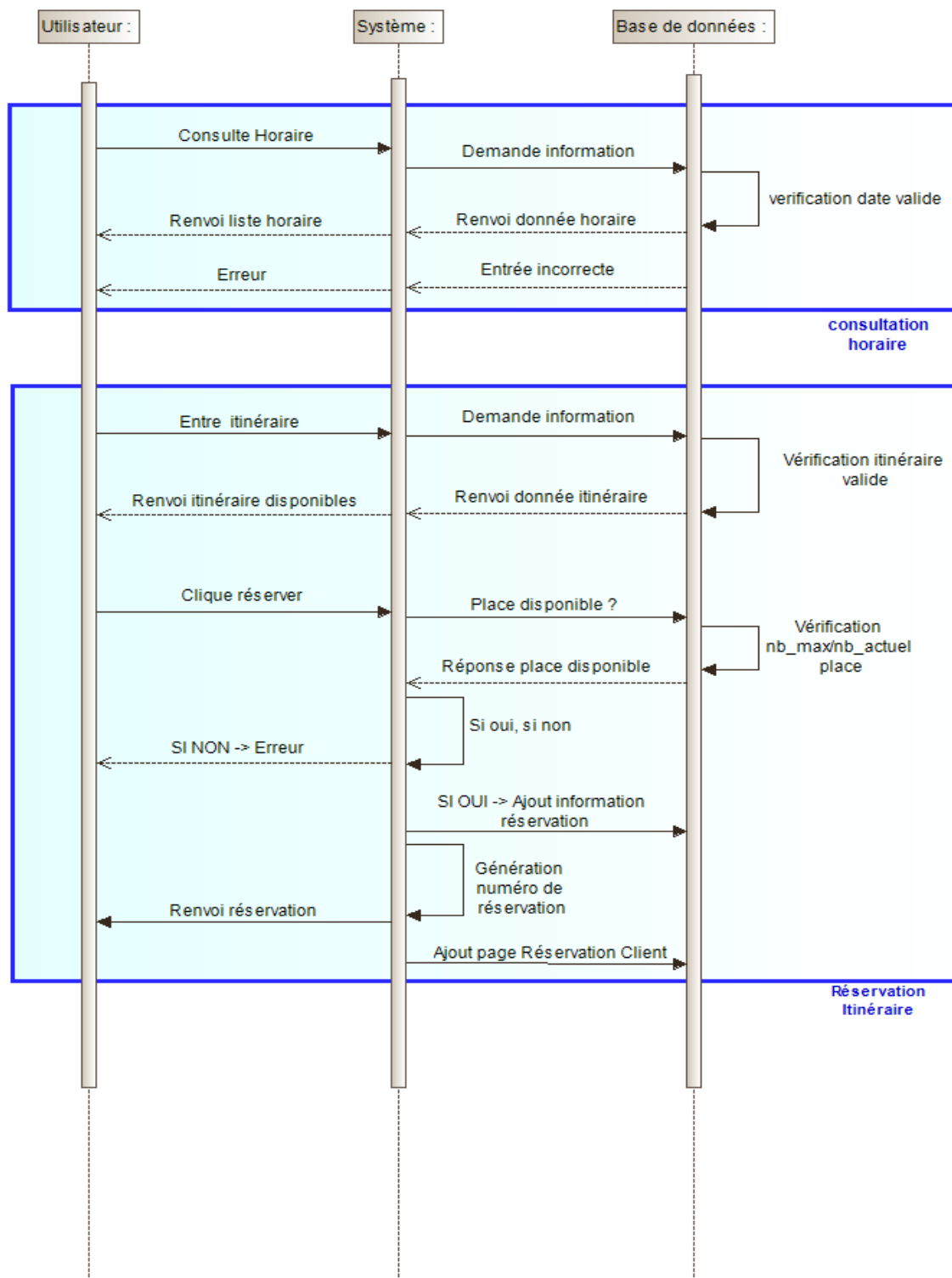
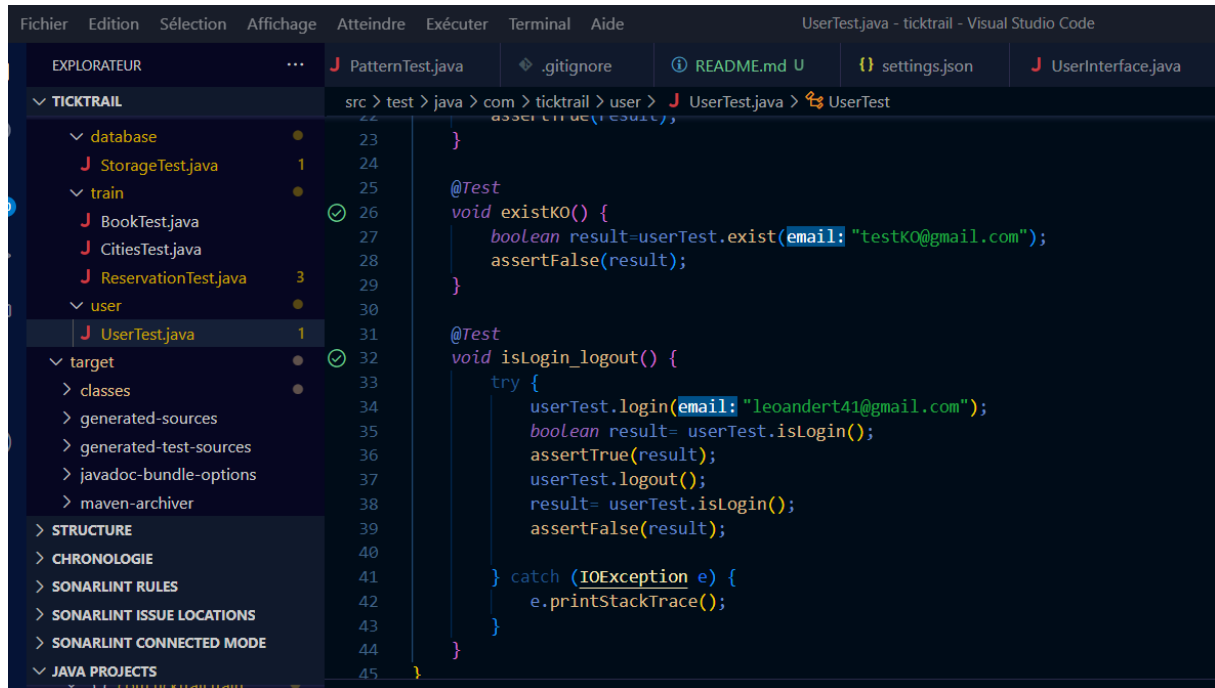


Diagramme de séquence qui explique la recherche de trajet.

## 4.a) Test



The screenshot shows the Visual Studio Code interface. The Explorer on the left lists the project structure under 'TICKTRAIL', including folders like 'database', 'train', 'user', and 'target', and files like 'StorageTest.java', 'BookTest.java', 'CitiesTest.java', 'ReservationTest.java', and 'UserTest.java'. The main editor displays the code for 'UserTest.java', which contains two JUnit tests: 'existKO()' and 'isLogin\_logout()'. The 'existKO()' test calls 'userTest.exist()' with the email 'testKO@gmail.com' and asserts that the result is false. The 'isLogin\_logout()' test calls 'userTest.login()' with 'leoandert41@gmail.com', asserts the result is true, then calls 'userTest.logout()' and asserts the result is false. Both tests catch 'IOException' and print the stack trace.

```
23 }
24
25
26 @Test
27 void existKO() {
28     boolean result=userTest.exist(email: "testKO@gmail.com");
29     assertFalse(result);
30 }
31
32 @Test
33 void isLogin_logout() {
34     try {
35         userTest.login(email: "leoandert41@gmail.com");
36         boolean result= userTest.isLogin();
37         assertTrue(result);
38         userTest.logout();
39         result= userTest.isLogin();
40         assertFalse(result);
41     } catch (IOException e) {
42         e.printStackTrace();
43     }
44 }
45 }
```

Les tests ont été fait via JUnit.

Test unitaires : assurer qu'une méthode exposée à la manipulation par un utilisateur fonctionne bien de la façon dont elle a été conçue. Aucune erreur détectée.

Tests d'intégration :

- Vérification des ensembles de fonctionnent : fonctionnent correctement.

- Approche du Big Bang.

- Avantages de l'approche Big Bang : C'est une bonne approche pour les petits systèmes. (Le cas de notre projet) L'approche Big Bang intègre tous les modules en une seule fois, c'est-à-dire qu'elle ne va pas pour intégrer les modules un par un.

### 5.a) Annexe (Table des illustrations)

Illustration 1 : Diagramme de classe.....	7
Illustration 2 : Modèle de donnée_.....	8
Illustration 3 : Diagramme d'activité .....	9
Illustration 4 : Diagramme de séquence (Manipulation User).....	10
Illustration 5 : Diagramme de séquence (Interaction).....	11