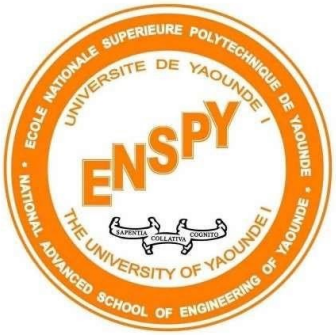
UNIVERSITÉ DE YAOUNDÉ I

ÉCOLE NATIONALE SUPÉRIEURE POLYTECHNIQUE

GÉNIE INFORMATIQUE



RÉSEAUX MOBILES ET INTELLIGENTS – 4GI

Analyse , Conception et Implémentation du site vitrine de WiaFirm

DOCUMENTATION TECHNIQUE

Realisé par :

CHIMI YOUKAP Oréal S.

NDEMA BEKOLLE Emmanuel I.

NGAGOM Chebil B.

NGALLE Arsenne

NTANTAME PETIEU Stephen

TCHAZOU Fabrice

Supervisé par :

Pr. Dr. Djotio Thomas

22 juin 2021

Table des matières

Table des figures

Lexique

- **SGBD** : Sytème de gestion des bases de données.
- **Frontend** : La partie d'un site ou application web avec laquelle l'utilisateur interagit directement.
- **Backend** est le côté serveur du site ou application Web. Il stocke et organise les données, et s'assure également que tout fonctionne correctement du côté client du site Web.
- **PaaS** est l'un des types de cloud computing, principalement destiné aux développeurs ou aux entreprises de développement où le client (développeur ou entreprise) assure la maintenance des applications, et le fournisseur assure la maintenance de la plateforme d'exécution de ces applications

Introduction

Le projet WiCoNAS : Wireless Community Networks As a Servicea été mi sur pieds afin d'utiliser une approche communautaire pour la fourniture des services. Un composant essentiel de ce projet est le firmware qui sera installé sur les noeuds pour déployer le réseau communautaire. Notre projet a pour but de réaliser une plateforme qui sera un site vitrine du projet. Dans le présent document, nous allons présenter l'aspect technique de l'implémentation qui a été réalisé et présenté dans le rapport du projet.

1 Rappels sur le projet

Notre projet consiste à la mise sur pied d'un site web qui fera office de vitrine au firmware et aux communautés qui seront créées avec firmware wiafirm. A travers cette plateforme, il sera question pour nous de faire le lien entre un internaute quelconque et le projet. Nous nous occuperons dans une certaine mesure de la partie émergée de l'iceberg tandis que les autres projets sont plus fonctionnels. Dans la suite , nous ferons un travail d'ingénierie des besoins, puis s'en suivra une analyse. Ensuite, nous présenterons les travaux de conception et nous clôturerons par l'aspect implémentation. Le rapport sur le projet est disponible à https://cutt.ly/6n3A0hg

2 Spécifications des besoins

2.1 Besoins fonctionnels

La plateforme doit réaliser les fonctionnalités suivantes :

- F1. Fournir une bonne description du firmare
- F2. Decrire le processus d'acquisition et mise en opération du firmware
- F3. Présenter les réseaux communautaires
- F4. Fournir une carte de localisation des communautés. En effet, comme présenté à **??**, le projet **Wireless Community Network : Geolocalisation and mapping** permettra de géolocaliser les communautés. Il sera question pour nous ici, d'utiliser leur platforme pour afficher une carte contenant les communautés
- F5. Fournir une bonne documentation du firmware
  - F5.1 Fournir des pistes de résolution de certains problèmes reccurents
  - F5.2 Permettre à un utilisateur d'ouvrir un ticket pour des problèmes singuliers. À **??** , nous avons présenté le projet **Ticketing System** qui s'occupera de la gestion des tickets. Il sera question pour nous donc, de fournir un lien direct vers cette plateforme
- F6. Présenter les contributeurs du projet
- F7. Permettre la création de compte
- F8. Permettre la création des communautés

  Le projet **WiaGate IAM** permet de gérer les identités et les accès. Nous devrons donc intégrer ce dont on a beoin pour répondre aux besoins F7 et F8
- F9. Permettre d'effectuer des dons pour soutenir le projet.

2.2 Besoins non fonctionnels

Le site sera soumi à certaines contraintes dont :

- **Technologies à utiliser** :Les technologies préconisées pour l'implémentation sont les suivantes :
  - **PostGreS** pour la base de données
  - **React JS** pour la partie Frontend

— **Django Rest** pour le backend

- **Grande capacité d'évolution**. En effet, le projet **wiabox** est assez jeune ; il faudra donc que son site vitrine permette de facon aisée l'ajout, la modification et la suppression de contenu
- **L'ergonomie** . Le site web doit être agréable d'utilisation.
- **La sécurité.** Le site doit être suffisamment sécurisé pour éviter d'éventuelles fraudes notamment pour ce qui concerne les dons
- **La responsivité.** Le site doit s'adapter à presque tout type d'écran

3 Analyse

3.1 Diagramme de contexte

3.1.1 Identification des acteurs

3.1.1.1 Acteur primaire

Le système est concu principalement pour l'internaute lambda.

3.1.1.2 Acteur sécondaire

Pour répondre aux besoins fonctionnels, le système aura besoin de :

- La plateforme centrale de gestion des Identités et des accès : **WiaGate IAM** : pour gérer la création des comptes et des communauté
- La plateforme de gestion de la géolocalisation. Nous l'appelerons ici **WiaGate Location**
- La plateforme de gestion des tickets pour les problèmes techniques. Nous l'appelerons **Ticketing System**
- **Les systèmes de paiement mobile et bancaire**

3.1.2 Le diagramme

A la figure 1, nous représentons le diagramme de contexte. Il est accessible à **https://cutt.ly/an3FsjK**

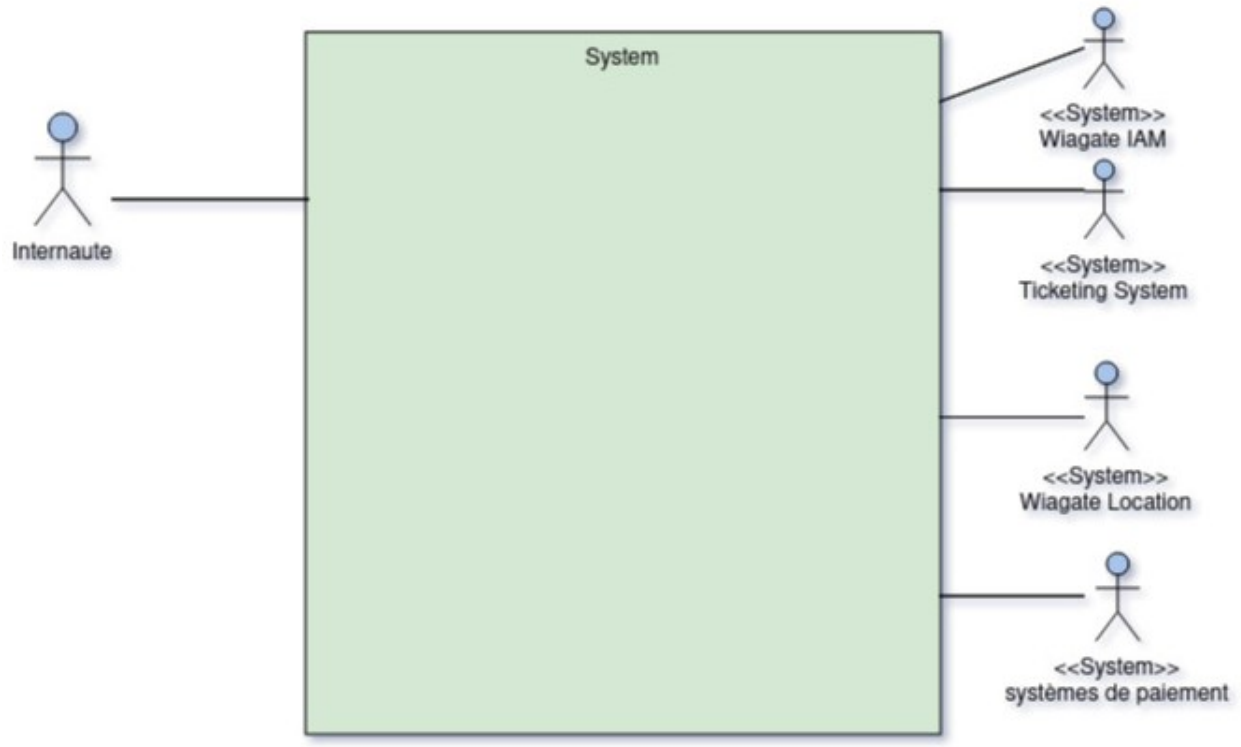


FIGURE 1 – Diagramme de contexte

3.2 Diagramme de cas d'utilisation

3.2.1 Le diagramme

A la figure 2, nous représentons le diagramme de cas d'utilisation. Il est accessible à **https://cutt.ly/0n3FYdd**

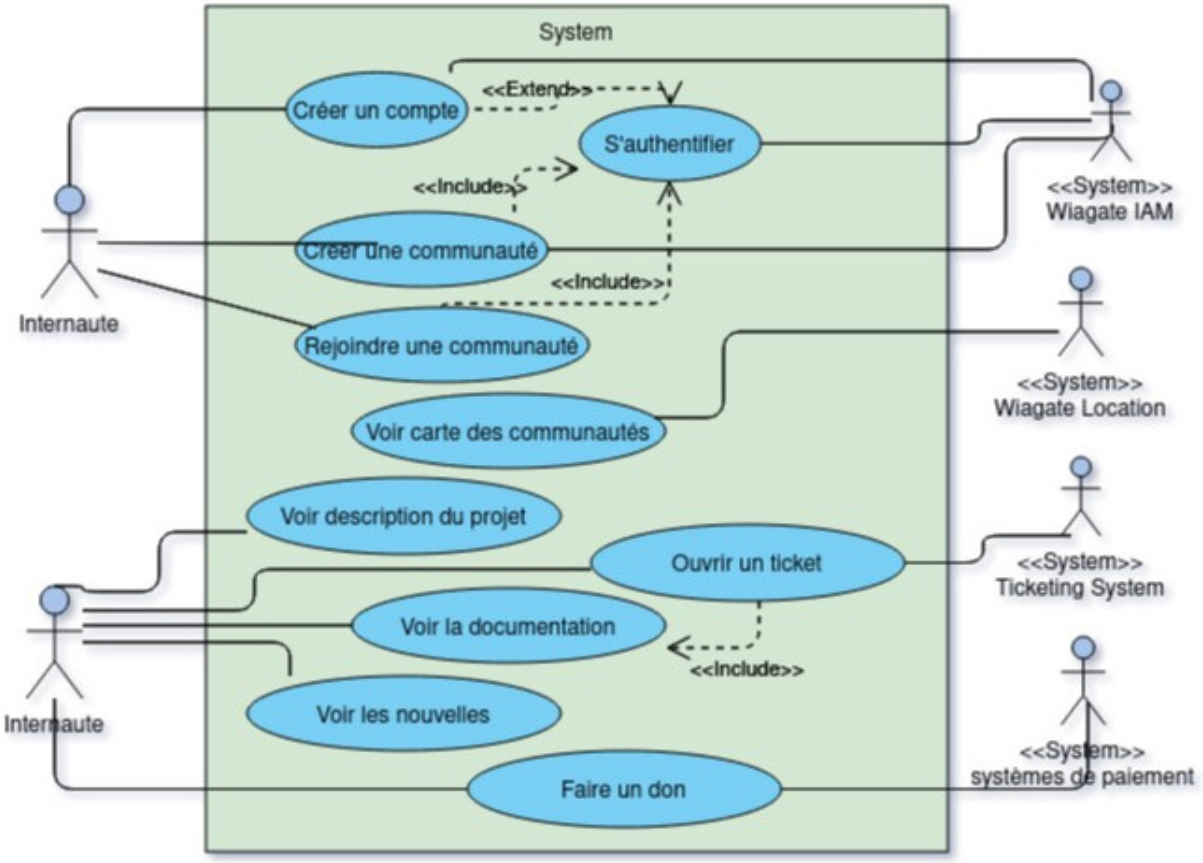


FIGURE 2 – Diagramme des cas d'utilisation

3.2.2 Description textuelle de quelque cas d'utilisation

3.2.2.1 Ouvrir un ticket

— Acteurs concernés : L'internaute et le Ticketing system

— Précondition : L'internaute doit être sur la page de la documentation

— Postcondition : L'internaute est redirigé vers la plateforme de gestion des tickets

— Scénario nominal :

1. L'internaute clic sur un button "Ouvrir un ticket"
2. Le système lui fait savoir qu'il sera redirigé
3. L'internaute valide
4. Le système le redirige vers le Ticketing System

— Scénario alternatif : Si l'internaute désire ne pas être redirigé vers le Ticketing System, tout est annulé. Le système recharche la page où l'internaute se trouvait

3.2.2.2 Faire un don

— Acteurs concernés : L'internaute et le Système de paiement

— Précondition : Aucune

— Postcondition : Le don de l'internaute est effectué et enregistré

— Scénario nominal :

1. L'internaute clic sur un button "Faire un don"
2. Le système demande le nom et le pays du donateur
3. L'internaute entre ses informations et valide
4. Le sytème demande le type de système à utiliser (OM ou MoMo ou PayPal)
5. L'internaute choisi et valide
6. Le Système demande les informations relatives au système de paiement chosi
7. L 'internaute choisi et valide
8. Le Système contacte le systeme de paiement pour effectuer le paiement puis il enregistre la transaction

— Scénario alternatif :

Au niveau du point 2 su scénario nominal, l'internaute peut décider de rester anonyme. Dans ce cas, il coche un checkbox (Je veux rester anonyme) puis on continue au point 3

3.3 Diagramme de classe metier

A la figure 3, nous représentons le diagramme de classe metier. Il est accessible **https://cutt.ly/Ln3FZIR**

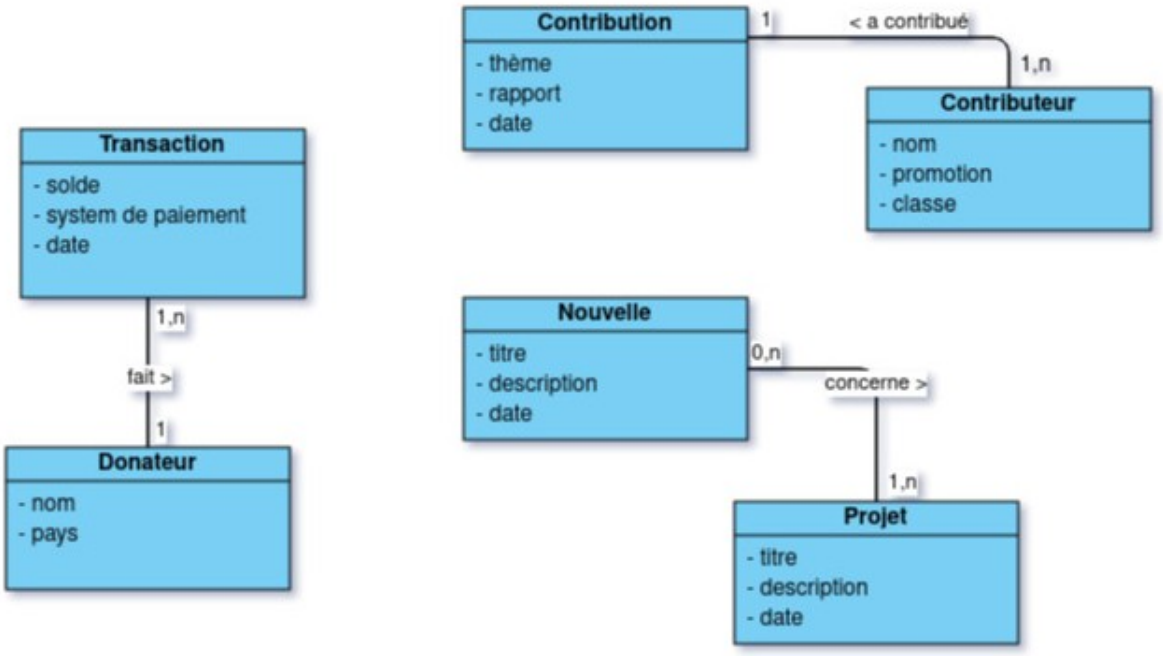


FIGURE 3 – Diagramme de classe metier

4 Conception

4.1 Diagramme de classe technique

À la figure 4, nous présentons le diagramme de classe technique. Il est accessible à **https://cutt.ly/jn3F1H4**

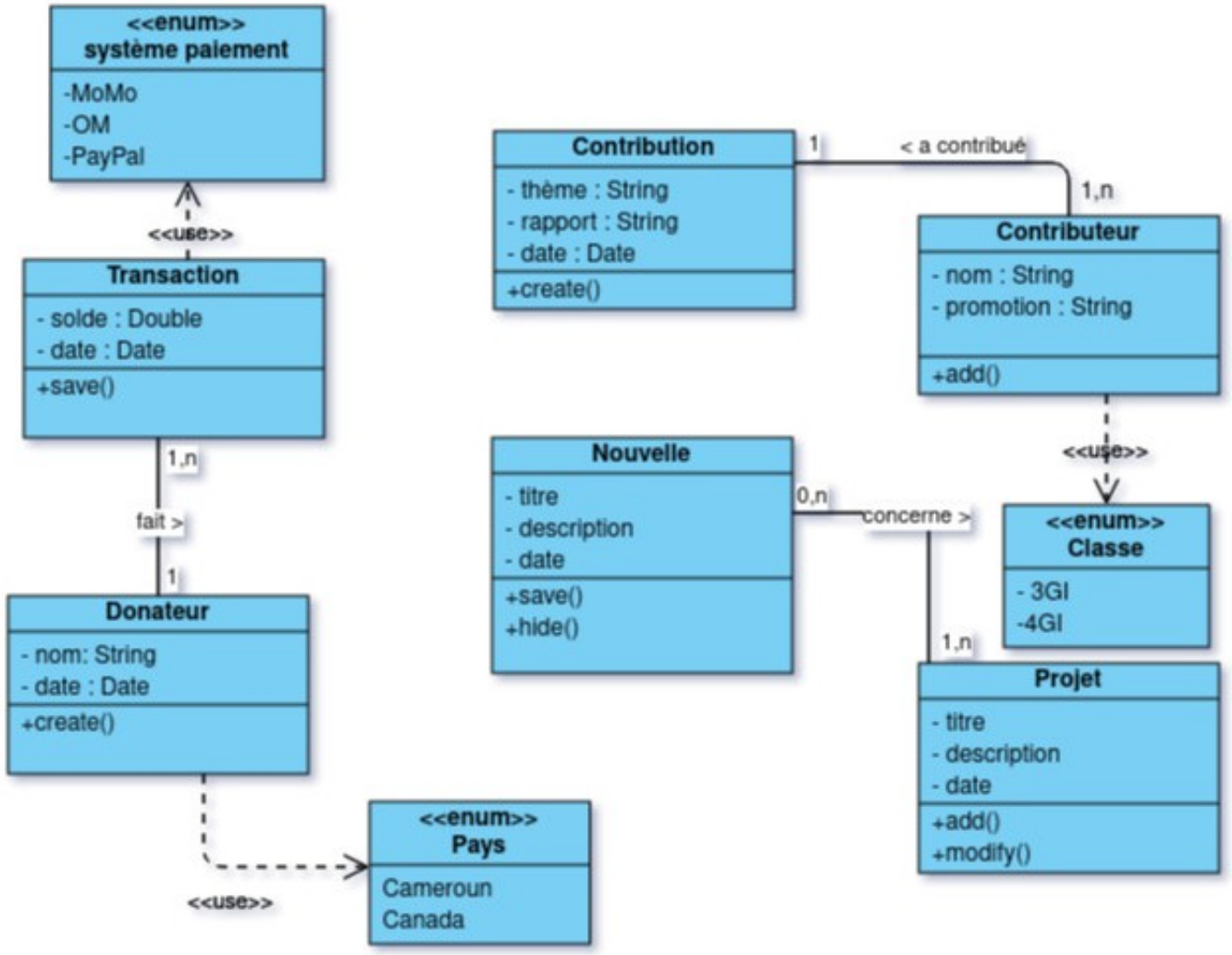


FIGURE 4 – Diagramme de classe technique

4.2 Diagrammes de séquence

4.2.1 Diagramme de sequence : Faire un don

À la figure 5, nous présentons le diagramme de séquence correspondant au cas d'utilisation **Faire un don**. La source est accessible à **https://cutt.ly/Nn3GqH1**

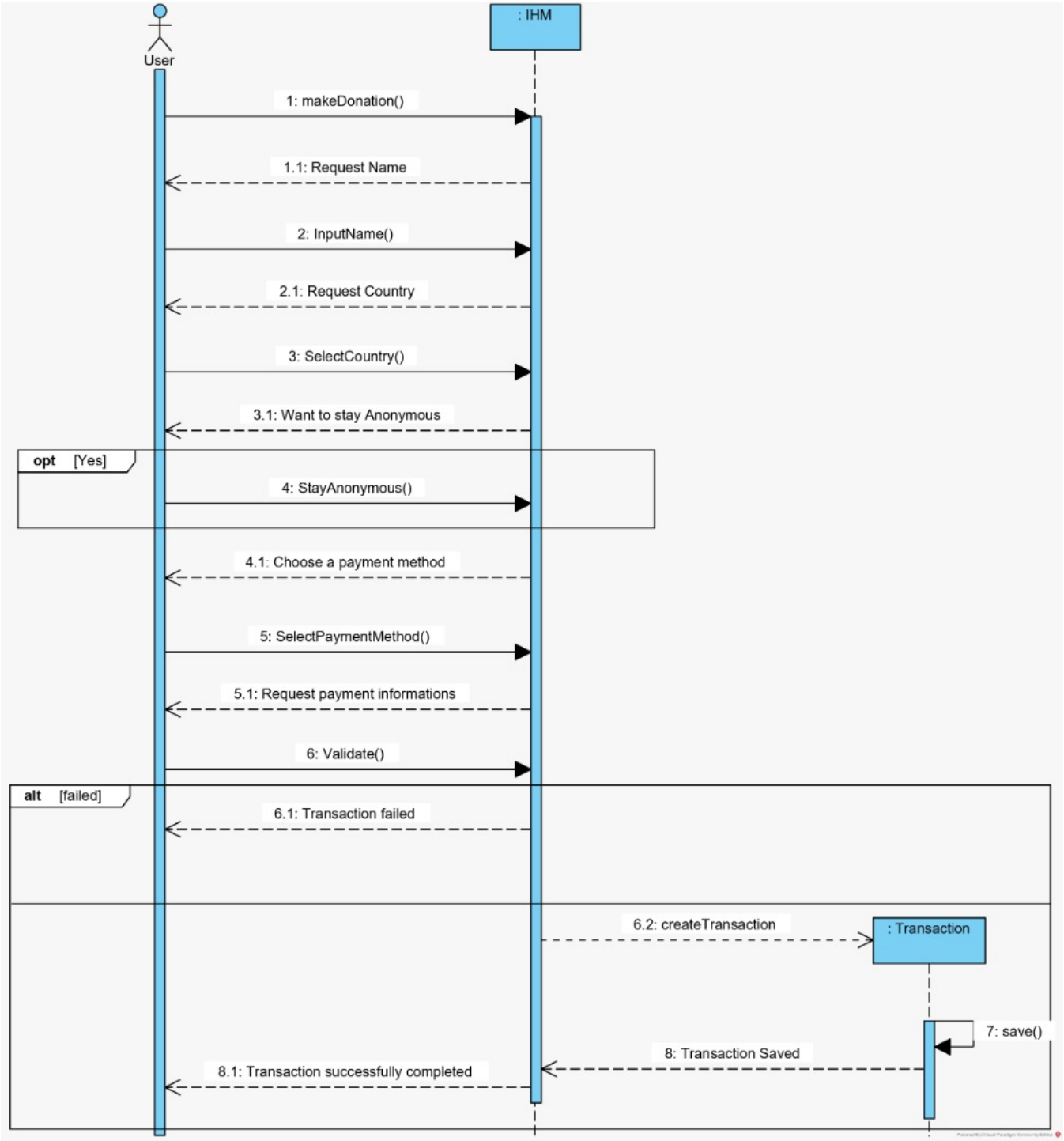


FIGURE 5 – Diagramme de sequence : Faire un don

4.2.2 Diagramme de sequence : Ouvrir un ticket

À la figure 6, nous présentons le diagramme de séquence correspondant au cas d'utilisation **Ouvrir un ticket**. Il est accessible à **https://cutt.ly/kn3GugH**

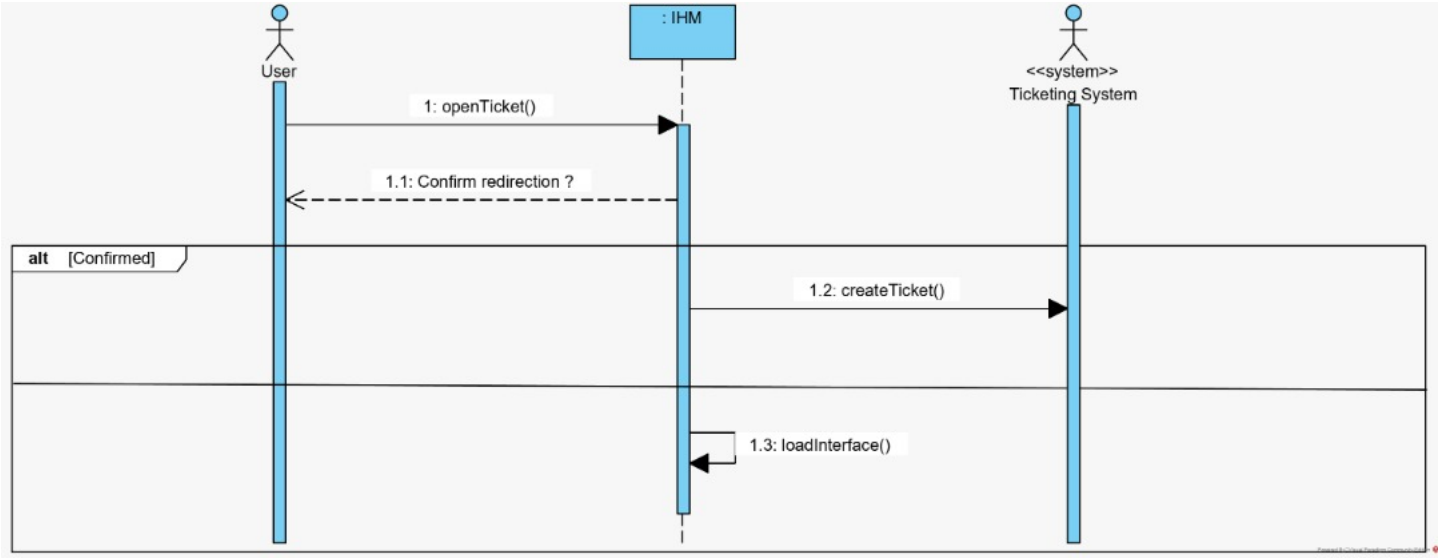


FIGURE 6 – Diagramme de sequence : Ouvrir un ticket

5 Implémentation

5.1 Langages utilisés

Conformément aux besoins non fonctionnels, notamment celui qui avait trait avec les langages de programmation, nous avons utilisé :

— **PostGreS** pour la base de données

— **React JS** pour la partie Frontend

— **Django Rest** pour le backend

5.2 Liens vers les codes sources

5.2.1 Le Frontend

Les codes sources du Frontend sont accessibles sur **Github** à l'adresse

https://github.com/shaquillo/wiabox—frontend

5.2.2 Le Backend

Les codes sources du Backend sont accessibles sur **Github** à l'adresse

https://github.com/shaquillo/wiabox-backend

5.2.3 La simulation de la map des communautés

Pour la simualation, notamment pour illustrer la visualisation des communautés sur une carte, nous avons rapidement dévélopper une API. Elle est disponible également sur **Github** à l'adresse

https://github.com/script-0/wiabox-nodes-map

5.2.4 Les documents

Nous avons créé un repot spécial pour stocker tout les documents qui ont été produits localement dans le cadre du projet. Il est accessible sur **Github** à l'adresse

https://github.com/script-0/wiafirm-Docs

Le rapport a été realisé en Latex et est accessible :

— en Lecture à **https://www.overleaf.com/read/wsqnmhttctvw**

— en écriture à **https://www.overleaf.com/6917252245jwbntfgbgbfh**

Le présent document lui aussi a été réalisé en Latex et est accessible :

— en Lecture à **https://www.overleaf.com/read/srfmsxxjrjkz**

— en écriture à **https://www.overleaf.com/7743814712zsbgypsmgqyd**

6 Déploiement

6.1 Déploiement du backend

Le backend qui est une API Rest a ete realise a l'aide du framework Django Rest. Le déploiement de cette API sera réalisé comme suit :

1. Ajouter les librairies nécessaire pour l'exécution de l'API dans le fichier **requirements.txt**

```
$ pip freeze > requirements.txt
```

2. Copier le dossier sur le serveur ou se connecter au serveur et cloner le projet de son repositoire sur GitHub dans le cas ou git est installé sur le serveur.
3. Ouvrir le terminal et se placer dans le dossier du projet et exécuter les commandes suivantes
   (a) Pour installer les librairies nécessaire pour l'exécution de l'API :

```
$ pip install -r requirements.txt
```

   (b) Pour initialiser les schémas de la base de données :

```
$ python manage.py makemigrations
$ python manage.py migrate
```

   (c) Pour lancer l'API sur le port "PORT"

```
$ python manage.py runserver 0.0.0.0 -p PORT
$ python manage.py runserver 0.0.0.0 -p 8080// exampel PORT 8080
```

6.2 Déploiement du frontend

6.2.1 Cas d'un serveur dédié

Le Frontend a été dévélopé avec **React JS**. La procédure de déploiement est la suivante :

1. Ajoutons **serve** aux dépendences du projet :

```
$ npm install serve
```

2. Construisons la version *production* du projet

```
$ npm run build
```

Cela crée un répertoire **build** à l'intérieur du répertoire racine, qui regroupe votre application React et la réduit à de simples fichiers HTML, CSS et JavaScript. Ce répertoire de construction sert votre application via un simple point d'entrée, **index.html**, où réside l'ensemble de votre application React.

3. Lancons le projet

```
$ serve -s build
```

6.2.2 Cas d'une plateforme PaaS : Heroku

6.2.2.1 Pourquoi Heroku ?

Heroku est une plateforme **PaaS (Platform as a Service)** qui prend en charge plusieurs langages de programmation notamment React JS. Heroku a ceci d'avantageux qu'il est :

- Facile d'utilisation.
- Gratuit pour un début
- Facile à gérer et à faire évoluer.
- Bonne prise en charge des outils d'intégration aux processus CI/CD.
- Facturer à l'utilisation

6.2.2.2 Étapes du déploiement sur Heroku

1. Créer un compte

   Comme toute plateforme en ligne, pour l'utiliser il faut créer un compte. La création de compte est simple. Rendez vous à https://signup.heroku.com/login et suivez la procédure.

2. Créer une nouvelle application sur Heroku

   Une fois votre compte créé vous serez redirigé à https://dashboard.heroku.com/apps. Le processus de création d'une application est assez intuitif

3. Installer et configurer de Heroku CLI

   Heroku CLI est l'invite de commande permettant d'interagir avec la plateforme Heroku.

   (a) Installation

```
$ sudo snap install --classic heroku
```

(b) Connectez vous à votre compte

```
$ heroku login
```

4. Configurer le projet : Refaire les étapes pour un déploiement sur un serveur dédié
5. créer un fichier **Procfile** et ajouter la ligne suivante

   web : serve -s build
6. Créer un repot git local

```
$ git init
$ git add .
$ git commit -m "Deploy to Heroku"
$ git checkout -b master
```

7. Lier le repot Git local au repot heroku distant

   Ici, on suppose que lors de la création de l'application sur Heroku vous avez donné le nom **wiafirm**

```
$ heroku git:remote -a wiafirm
```

8. Deploier le code

```
$ git push heroku master
```

6.3 Lien du projet

Le projet a été déployé sur Heroku en suivant les étapes présentées plus haut. Il est accessible à l'adresse : **https://wiafirm.herokuapp.com/**

Conclusion

Parvenus au terme de notre travail, il était question pour nous d'effectuer de fournir une documentation technique de notre projet. A cet effet, les liens vers les différents documents et codes sources ont été fournis. Toutefois fois, en cas d'incompréhension nous restons disponibles aux mails : **bekolleisaac@gmail.com** et **stephenpetieu@gmail.com**

Références

- La documentation de React Js - https://reactjs.org/docs/getting-started.html
- La documentation de Django Rest - https://www.django-rest-framework.org/