

Le système d'achat de l'énergie

JUILLIARD Quentin S506
COGNE Romain S506



Monsieur BOISSON, INFO0503 / Département Informatique

JUILLIARD-COGNE

Dimanche 11 décembre 2022

Table des matières

1	Introduction	2
2	Lancement du projet	3
2.1	Organisation des fichiers	3
2.2	Compilation sous Linux	4
2.3	Compilation sous Windows	6
3	Technologie de communication	8
3.1	Hypertext Transfer Protocol	8
3.2	User Datagram Protocol	9
3.3	Transmission Control Protocol	9
4	Les différentes relations	10
4.1	Relation entre Revendeur et Tare	11
4.2	Relation entre Tare et Marché de Gros	12
4.3	Relation entre Marché de Gros et PONE	13
4.4	Role de l'AMI	13
5	Scénarios	14
5.1	Scénario A	14
5.2	Scénario B	15
5.3	Scénario C	16
5.4	Scénario D	17
5.5	Scénario A2	18
6	Conclusion	19

1 Introduction

Dans le cadre du module de Client / Serveur de 3ème année de licence informatique, il nous à été demandé de réaliser un projet de communication entre différents acteurs, le tout en utilisant diverses technologies.

La partie projet d'INFO0503 est séparé en deux parties :

- La première partie est un ensemble de fichiers JAVA et PHP présents dans l'archive jointe à ce rapport, ils constituent la partie "Projet" du module.
- La deuxième partie est le rapport, il constitue la partie "CRTP" du module..

Voici un résumé du sujet donné :

Dans un contexte de tension sur les énergies, nous devons réaliser une simulation de commande d'énergie auprès d'un trader en Energie (TARE). Ce dernier se fournira en énergie auprès du marché de gros. Le marché de gros reçoit des énergies en permanence d'un producteur d'énergie (PONE). L'Autorité des Marchés Internationaux (AMI) régit l'ensemble du trafic de l'énergie et assure l'authenticité des énergies transférées.

2 Lancement du projet

Voyons désormais comment démarrer le projet.

2.1 Organisation des fichiers

Dans notre archive, une fois décompressé, vous trouverez de nombreux dossiers et fichiers.

- Le dossier `".vscode"` si vous êtes sous windows et que vous utilisez le logiciel Visual Studio Code comme nous, ce dossier apparaîtra. Il contient le fichier `"Launch.json"`, et nous reviendrons dessus dans la sous-partie `"Compilation Windows"`.
- Le dossier `"cls"`, vous retrouverez l'ensemble des `".class"` issue de la compilation JAVA.
- Le dossier `"doc"` regroupe l'ensemble de la documentation de notre projet. Ce dossier n'est pas réellement utile car nous n'avons pas réalisé de documentation précise à l'intérieur de notre projet par manque de temps. Nous avons commenté directement dans notre code sans utiliser les options et la syntaxe de la java doc.
- Le dossier `"lib"` contient les librairies qui sont nécessaires pour notre projet. Dans ce projet, nous avons uniquement utilisé la librairie JSON, intitulé ici `"json-20220924.jar"`.
- Le dossier `"Rapport"` contient l'ensemble des fichiers `".tex"` du rapport que vous êtes en train de lire, car, en effet le rapport a été réalisé sous LaTeX.
- Le dossier `"src"` contient l'ensemble des fichiers sources `".java"`. Dans ce dossier, il existe un sous-dossier pour chaque entité présente dans notre modélisation qui seront exposés dans le Chapitre 4. Il y a également la présence d'un fichier `"Lanceur.java"` celui-ci sert à lancer l'ensemble des serveur java en multi-threadé en un seul lancement pour plus de simplicité.
- De plus, dans le dossier `"src"`, il existe le dossier `"revendeur"`, le contenu de celui-ci doit être mis dans un serveur WEB PHP comme par exemple wamp ou autre.
- Le fichier `"commande.txt"` regroupe les commandes utiles pour la compilation et le lancement du projet.
- Le fichier `"config.json"` regroupe l'ensemble des ports et adresse des différents serveurs, il seront utilisés par le lanceur pour créer les différentes entités.

2.2 Compilation sous Linux

Si vous travaillez sur un système d'exploitation sous Linux, nous allons vous expliquer dans cette partie comment compiler et lancer le projet.

Tout d'abord, positionnez-vous dans le dossier "Projet-INFO0503" comme ceci :

```
quent@MSI:/mnt/c/Users/quent/OneDrive/Documents/Licence_info/L3/INFO0503/Projet-INFO0503$ ls
cls  commandes.txt  config.json  doc  lib  Rapport  src
```

Figure 1 – Positionnement dans le dossier

Puis réaliser la compilation de l'ensemble des fichiers en utilisant la commande suivante :

javac -d cls/ -cp lib/json-20220924.jar -sourcepath src/ src/Lanceur.java

```
quent@MSI:/mnt/c/Users/quent/OneDrive/Documents/Licence_info/L3/INFO0503/Projet-INFO0503$ javac -d cls/ -cp lib/json-20220924.jar -sourcepath src/ src/Lanceur.java
quent@MSI:/mnt/c/Users/quent/OneDrive/Documents/Licence_info/L3/INFO0503/Projet-INFO0503$ _
```

Figure 2 – Succès de la compilation

Une fois la compilation terminée, vous pouvez donc lancer le projet en utilisant la commande suivante :

java -cp cls/ :lib/json-20220924.jar Lanceur config.json

Il se peut que vous ayez un message d'erreur comme vous pouvez le voir sur la figure ci-dessous au moment du lancement de vos serveur HTTP.

```
quent@MSI:/mnt/c/Users/quent/OneDrive/Documents/Licence_info/L3/INFO0503/Projet-INFO0503$ java -cp cls/:lib/json-20220924.jar Lanceur config.json
[ TARE - Electricite ] : Started
[ PONE - Electricite ] : Started
[ TARE - Gaz ] : Started
[ PONE - Gaz ] : Started
[ TARE - Petrole ] : Started
[ AMI ] : Started
[ MarcheGros ] : Started
[ TARE - Electricite ] : Erreur lors de la creation du serveur java.net.BindException: Permission non accordée
[ TARE - Gaz ] : Erreur lors de la création du serveur java.net.BindException: Permission non accordée
[ TARE - Petrole ] : Erreur lors de la création du serveur java.net.BindException: Permission non accordée
quent@MSI:/mnt/c/Users/quent/OneDrive/Documents/Licence_info/L3/INFO0503/Projet-INFO0503$ _
```

Figure 3 – Erreur de lancement

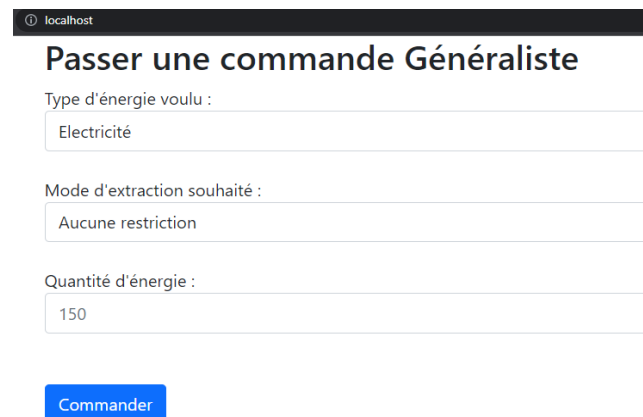
Pour résoudre ce problème, il vous suffit de passer en superadmin en utilisant le préfixe **sudo**, comme ceci :

sudo java -cp cls/ :lib/json-20220924.jar Lanceur config.json

```
quent@MSI:/mnt/c/Users/quent/OneDrive/Documents/Licence_info/L3/INFO0503/Projet-INFO0503$ sudo java -cp cls/:lib/json-20220924.jar Lanceur config.json
[sudo] Mot de passe de quent :
[ TARE - Electricite ] : Started
[ PONE - Electricite ] : Started
[ TARE - Gaz ] : Started
[ PONE - Gaz ] : Started
[ TARE - Petrole ] : Started
[ AMI ] : Started
[ MarcheGros ] : Started
[ PONE - Gaz ] : J'ai fournis le lot d'energie suivant : 1
[ PONE - Electricite ] : J'ai fournis le lot d'energie suivant : 1
[ MarcheGros ] : J'ai bien reçu le paquet : 1
[ MarcheGros ] : J'envoie une energie a l'AMI pour validation du prix
[ AMI ] : J'ai reçu une demande à valider
[ AMI ] : J'envoie ma réponse au marche de gros (positive).
[ MarcheGros ] : J'ai bien reçu la reponse de validation de l'AMI : Le prix proposé par le PONE (207) est accepté.
```

Figure 4 – Succès du lancement

Nous voila avec le projet opérationnel du côté serveur. Si le serveur PHP est également opérationnel, vous avez juste à vous rendre dans votre navigateur web et rentrer **localhost** dans la barre de recherche. Vous pouvez commencer à faire vos simulations d'achat d'énergie, et en même temps consulter les différentes communications entre toutes les entités dans le terminal. Vous pouvez alors essayer les différents scénarios ainsi que créer vos propres commandes.



localhost

Passer une commande Généraliste

Type d'énergie voulu :

Mode d'extraction souhaité :

Quantité d'énergie :

Commander

Figure 5 – Interface WEB

2.3 Compilation sous Windows

La compilation sous windows est bien plus simple. Pour ce faire vous pouvez utiliser Visual Studio Code (VS Code) par exemple.

Il vous faut ouvrir VS Code puis faire Fichier => Ouvrir Dossier => puis sélectionner le dossier racine du projet comme ci-dessous => puis faire **Sélectionner un dossier**.

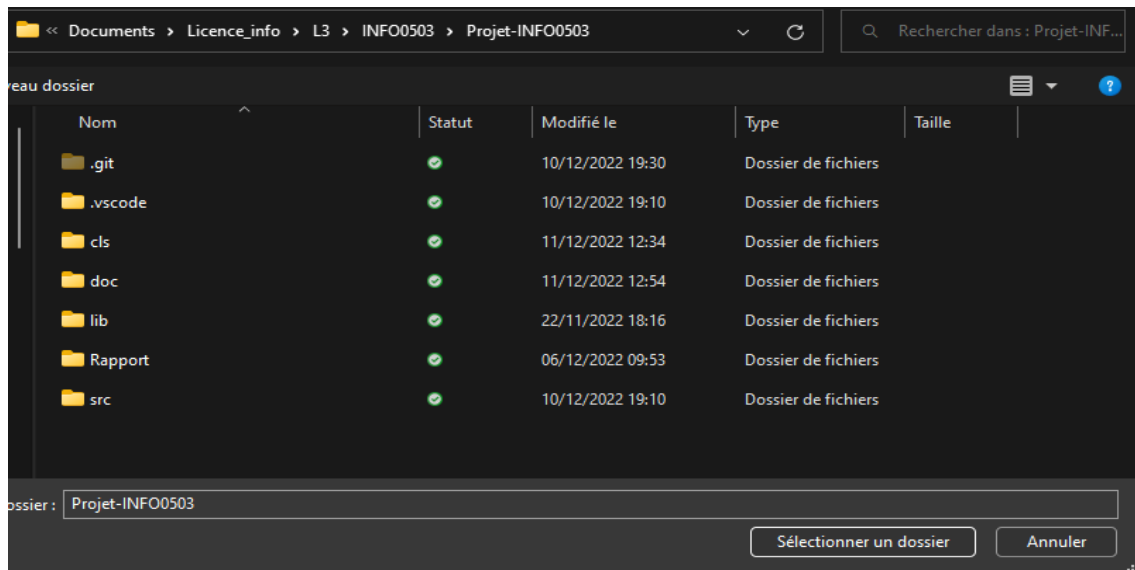


Figure 6 – Dossier à sélectionner

Une fois le projet ouvert vous pourrez voir sur la gauche de l'interface de VS Code, l'ensemble des dossiers et sous-dossiers contenant les fichiers ".java" du projet, soit l'arborescence du projet.

Une fois que VS Code a reconnu le projet JAVA. Rendez-vous dans le fichier "Lanceur.java", puis dirigez vous à la ligne 28, vous y trouverez un bouton **Run | Debug** comme sur la figure suivante.

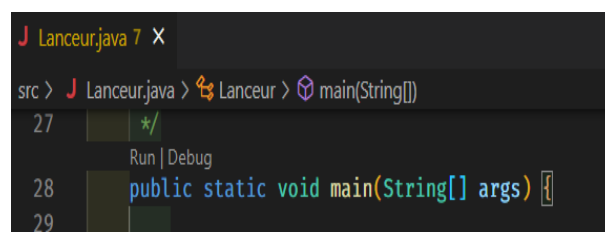


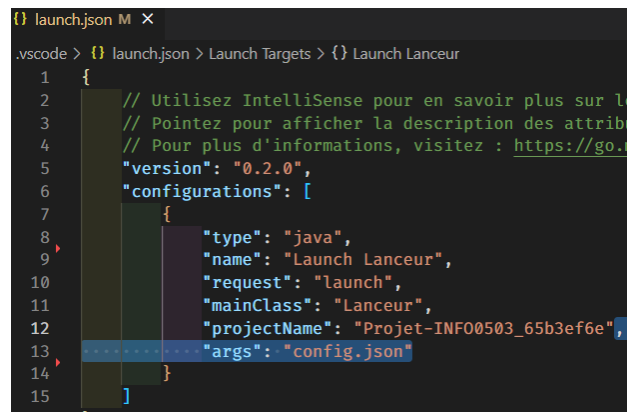
Figure 7 – Bouton Run | Rebug

Il vous suffit de presser le bouton **Run**, à ce moment là le terminal VS Code se lance mais une erreur apparait comme sur la figure suivante.

```
PowerShell 7.3.0
PS C:\Users\quent\OneDrive\Documents\Licence_info\L3\INFO0503\Projet-INFO0503> & 'C:\Program Files\Java\jdk-17.0.4\bin\java.exe' '@C:\Users\quent\AppData\Local\Temp\cp_dya6pr256ajwdh1k8u7d2i9u8.argfile' 'Lanceur'
Merci de donner un fichier de configuration json
PS C:\Users\quent\OneDrive\Documents\Licence_info\L3\INFO0503\Projet-INFO0503>
```

Figure 8 – Erreur de compilation VSCode

Pour résoudre ce problème, il vous suffit de vous rendre dans votre dossier ".vscode" puis dans le fichier 'launch.json' et d'ajouter dans "projectName" la ligne suivante : "args" : "config.json" comme sur la figure suivante.



```
{
  // Utilisez IntelliSense pour en savoir plus sur les attributs de configuration
  // Pointez pour afficher la description des attributs de configuration
  // Pour plus d'informations, visitez : https://go.microsoft.com/fwlink/?linkid=829397
  "version": "0.2.0",
  "configurations": [
    {
      "type": "java",
      "name": "Launch Lanceur",
      "request": "launch",
      "mainClass": "Lanceur",
      "projectName": "Projet-INFO0503_65b3ef6e",
      "args": "config.json"
    }
  ]
}
```

Figure 9 – Ajout de "args"

Une fois cette ligne ajoutée, vous pouvez relancer avec le bouton **Run** et vos serveurs seront opérationnels.

```
PS C:\Users\quent\OneDrive\Documents\Licence_info\L3\INFO0503\Projet-INFO0503> & 'C:\Program Files\Java\jdk-17.0.4\bin\java.exe' '@C:\Users\quent\AppData\Local\Temp\cp_dya6pr256ajwdh1k8u7d2i9u8.argfile' 'Lanceur'
[ TARE - Electricite ] : Started
[ AMI ] : Started
[ TARE - Petrole ] : Started
[ PONE - Gaz ] : Started
[ TARE - Gaz ] : Started
[ PONE - Electricite ] : Started
[ MarcheGros ] : Started
[ PONE - Electricite ] : J'ai fournis le lot d'energie suivant : 1
[ PONE - Gaz ] : J'ai fournis le lot d'energie suivant : 1
[ MarcheGros ] : J'ai bien reçu le paquet : 1
[ MarcheGros ] : J'envoie une energie a l'AMI pour validation du prix
[ AMI ] : J'ai reçu une demande à valider
[ AMI ] : J'envoie ma réponse au marche de gros (positive).
[ MarcheGros ] : J'ai bien reçu la reponse de validation de l'AMI : Le prix proposé par le PONE (266) est accepté.
```

Figure 10 – Lancement du serveur

3 Technologie de communication

3.1 Hypertext Transfer Protocol

L'Hypertext Transfer Protocol (HTTP) est un protocole Client/Serveur. Il fonctionne sur tous types de connexions. Il est basé sur le protocole TCP. Le protocole HTTP est basé sur la couche application tandis que le TCP est basé sur la couche de transport.

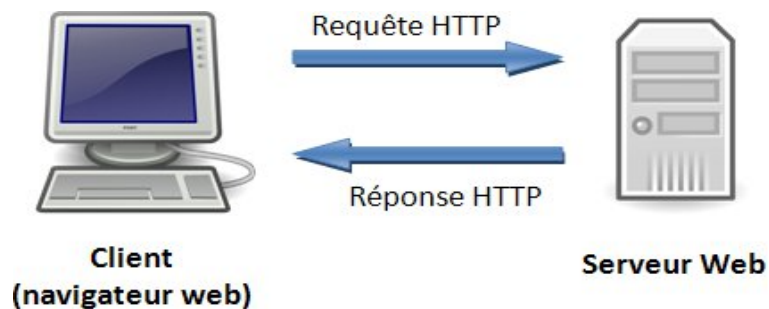


Figure 11 – Schéma explicatif HTTP simplifié

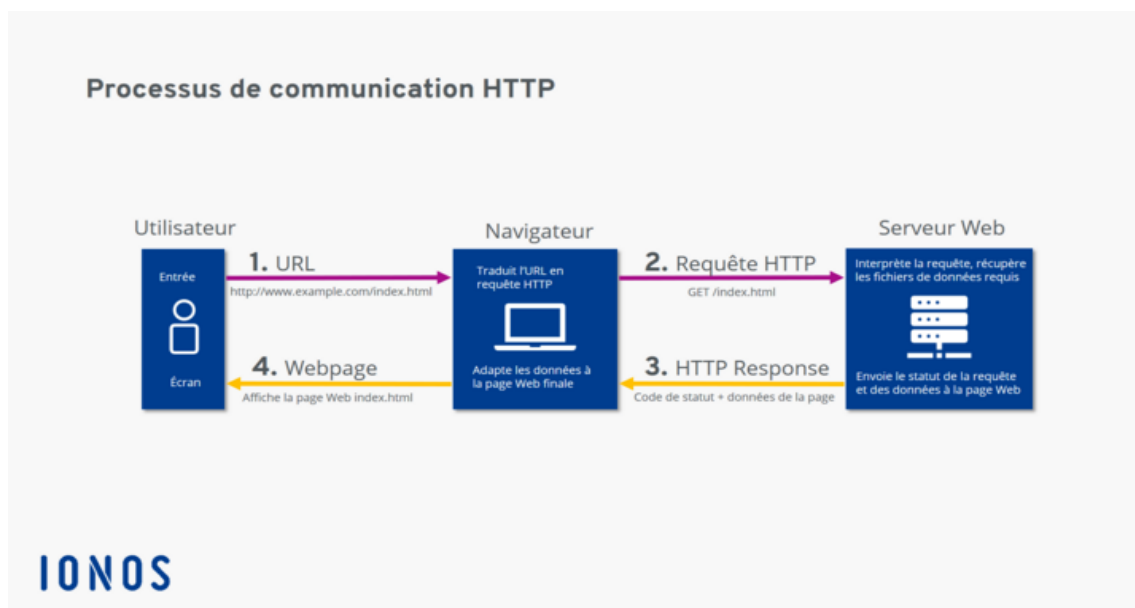


Figure 12 – Schéma explicatif HTTP détaillé

3.2 User Datagram Protocol

Le User Datagram Protocol (UDP) est un protocole utilisé pour le transfère de données sur internet. Il ne peut pas garantir que le packet soit bien arrivé à destination, c'est-à-dire sans contrôle de réception.

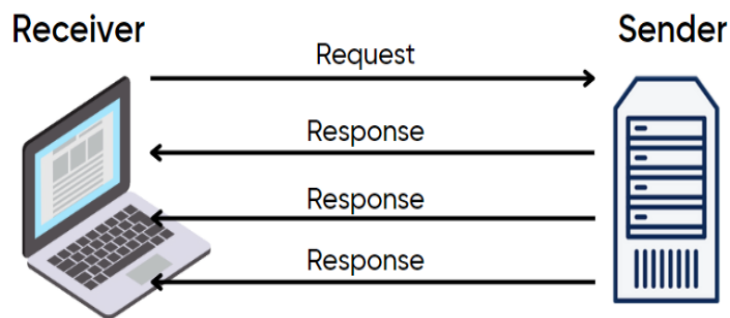


Figure 13 – Schéma explicatif UDP

3.3 Transmission Control Protocol

La Transmission Control Protocol (TCP) est un protocole de transport en réseau qui est fiable et en mode connecté. Le TCP fonctionne en trois phases distinctes :

1. Établissement de la connexion
2. Transfère des données
3. Fin de connexion

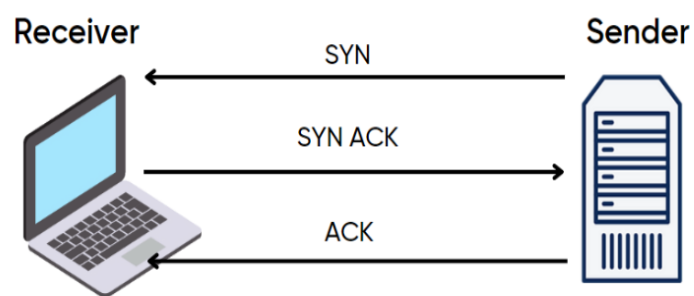


Figure 14 – Schéma explicatif TCP

4 Les différentes relations

Suite à la lecture complète du sujet, nous avons établi une modélisation complète sans détailler les informations qui transite entre les différents acteurs.

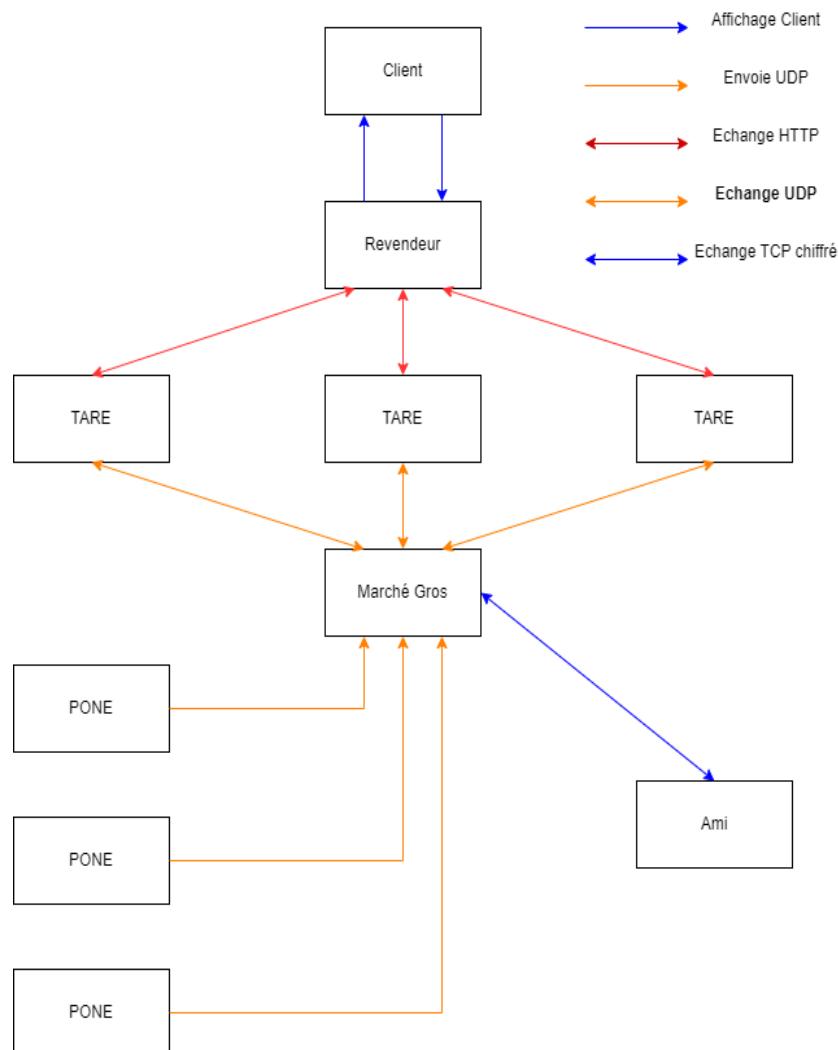


Figure 15 – Modélisation du système d'achat d'énergie

Pour le format des données nous avons décidé d'utiliser le format JSON. Il est facilement manipulable, et nous pouvons ainsi faire transiter les énergies et suivis de commande dans ce format. A chaque envoi, l'objet est sérialisé en JSON (fonction `toJson`) et recréé à la réception en objet java (`fromJson`).

4.1 Relation entre Revendeur et Tare

La relation entre Revendeur et TARE est une relation Hyper Text Protocol (HTTP). La relation HTTP exécute une requête en Transmission Control Protocol (TCP) /IP. L'utilisation de TCP oblige d'avoir un système en d'aquittement. Cette relation peut se modéliser comme ceci :

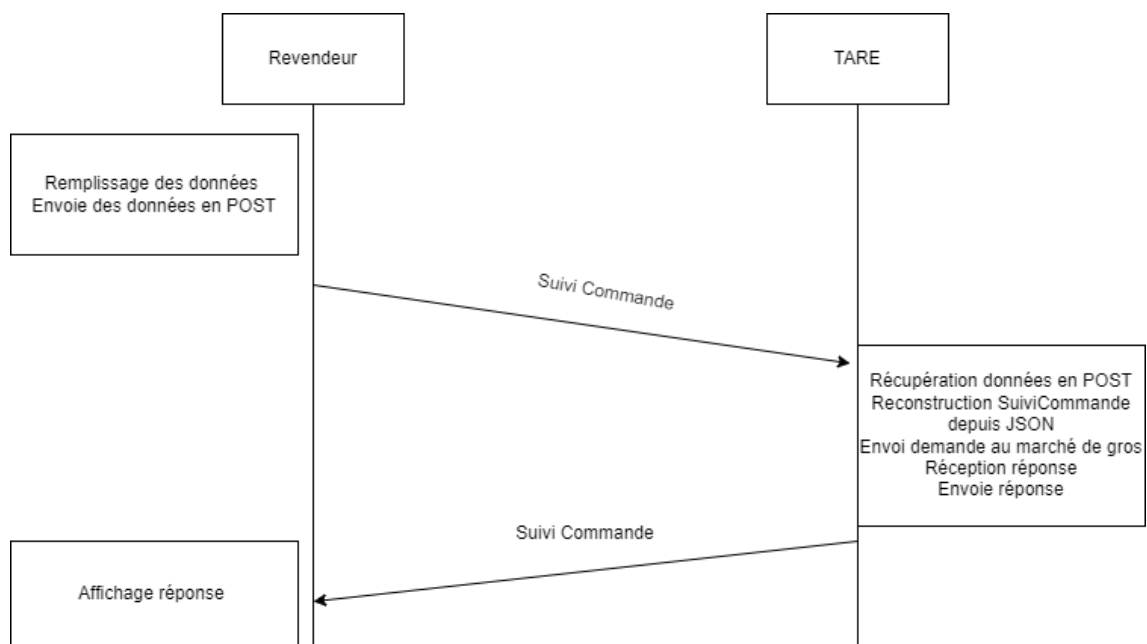


Figure 16 – Modélisation de la relation entre Revendeur et TARE

4.2 Relation entre Tare et Marché de Gros

Le Marché de Gros et les TAREs communiquent entre eux à l'aide de la communication UDP. Les informations qui transitent dans les requêtes sont des éléments de type SuiviCommande qui intègrent les variables de type "Energie". Voici à quoi ressemble un échange :

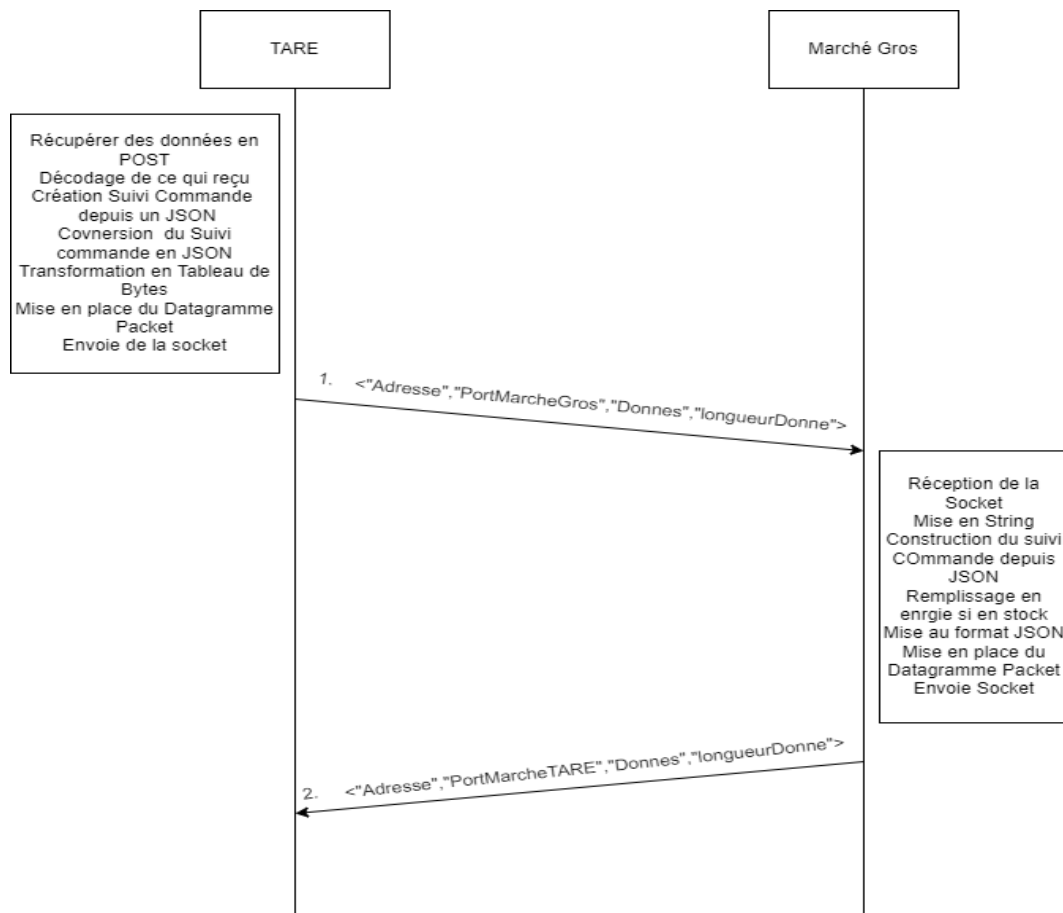


Figure 17 – Modélisation de la relation entre Marché de Gros et PONE

4.3 Relation entre Marché de Gros et PONE

Il existe une relation entre le Marché de Gros et le (ou les) PONE. Cette relation est unidirectionnel en User Datagramme Packet (UDP), nous envoyons des datagrammes packets du Pone contenant une énergie qui est caractérisée par :

- Un type d'énergie,
- Une quantité envoyé,
- Un mode d'extraction,
- Un prix unitaire,
- Un numéro de lot

Cette énergie est donc envoyée vers les marché de gros sans vérification d'aquisition car l'UDP est un protocole de transit d'informations sur internet qui n'est pas en capacité de garantir la bonne réception des informations chez le client.



Figure 18 – Modélisation de la relation entre Marché de Gros et PONE

4.4 Role de l'AMI

Dans ce réseau l'AMI ne communique uniquement avec le marché de gros en TCP. Il a pour rôle de vérifier le prix des énergies fournis par les PONE ainsi que de valider la vente avec les TAREs. De plus, l'ensemble des échanges sont sécurisés par un chiffrement RSA pour les crypter.

Cependant l'AMI doit également certifier, signer les énergies à la sortie de leur production, on parle alors de CeRtificAt D'Origine (ou CRADO). Il doit également certifier, signer les achats, on parle alors de CeRtificAt d'aCHAt (ou CRACHA).

5 Scénarios

Dans ce projet, il nous a été demandé de réaliser différents scénarios de test. Il y a en tout 5 scénarios à réaliser. Nous détaillerons dans les sous-parties suivantes l'ensemble des scénarios, comment nous les avons traités et s'ils sont opérationnels ou non, avec les explications si ceux-ci n'ont pas été implémentés.

5.1 Scénario A

Dans un système ayant un seul PONE et un seul TARE (en plus du revendeur, du marché de gros et de l'AMI), le client demande une quantité d'énergie sans aucune contrainte particulière et sa demande est toute de suite satisfaite car le PONE produit exactement le type d'énergie demandé. Il y a donc achat (avec enregistrement de l'achat côté AMI) puis distribution au revendeur (en passant par le TARE). Pour des soucis de simplicité au vue de notre modélisation, nous avons fait le choix de requêter directement sur notre marché de gros qui est alimenté en permanence et automatiquement par nos différents PONEs.

Lorsque le bouton "Scénario A" est actionné, nous requêtons notre marché de gros en lui demandant une énergie électrique quelque soit son mode de production. Dans la requête du "Scénario A", nous passons dans la requête une demande qui est intitulée "Suivi commande". La demande d'énergie souhaitée est une énergie de type "électricité" avec un type d'extraction "Sans restriction" de plus nous passons une quantité égale à 1 pour être sur que celle-ci soit aboutie sans échec.

Il se peut que l'énergie envoyée soit du nucléaire ou du charbon ou encore eolien.

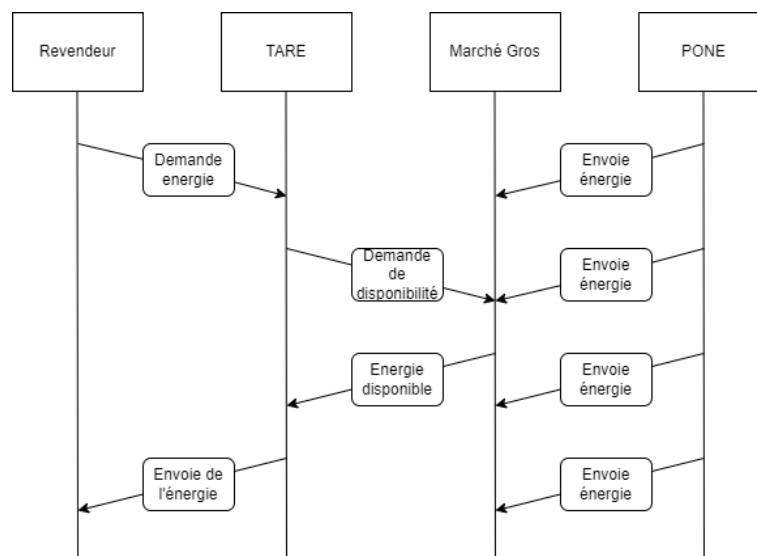


Figure 19 – Schéma du scénario A

5.2 Scénario B

Dans un système ayant un seul PONE et un seul TARE (en plus du revendeur, du marché de gros et de l'AMI), le client demande une quantité d'énergie qui n'est pas disponible pour le moment ... Par contre, au bout de quelques instants de fonctionnement, la demande du client peut-être satisfaite (car le PONE a suffisamment fourni d'énergie). Cela est possible après une nouvelle demande du client.

Pour ce scénario, nous avons réalisé une demande du client sur une énergie de type "Electricité" avec aucune restriction sur le type d'électricité demandé avec une quantité de 100. La première fois cela échouera, mais après un petit moment, la requête sera satisfaite.

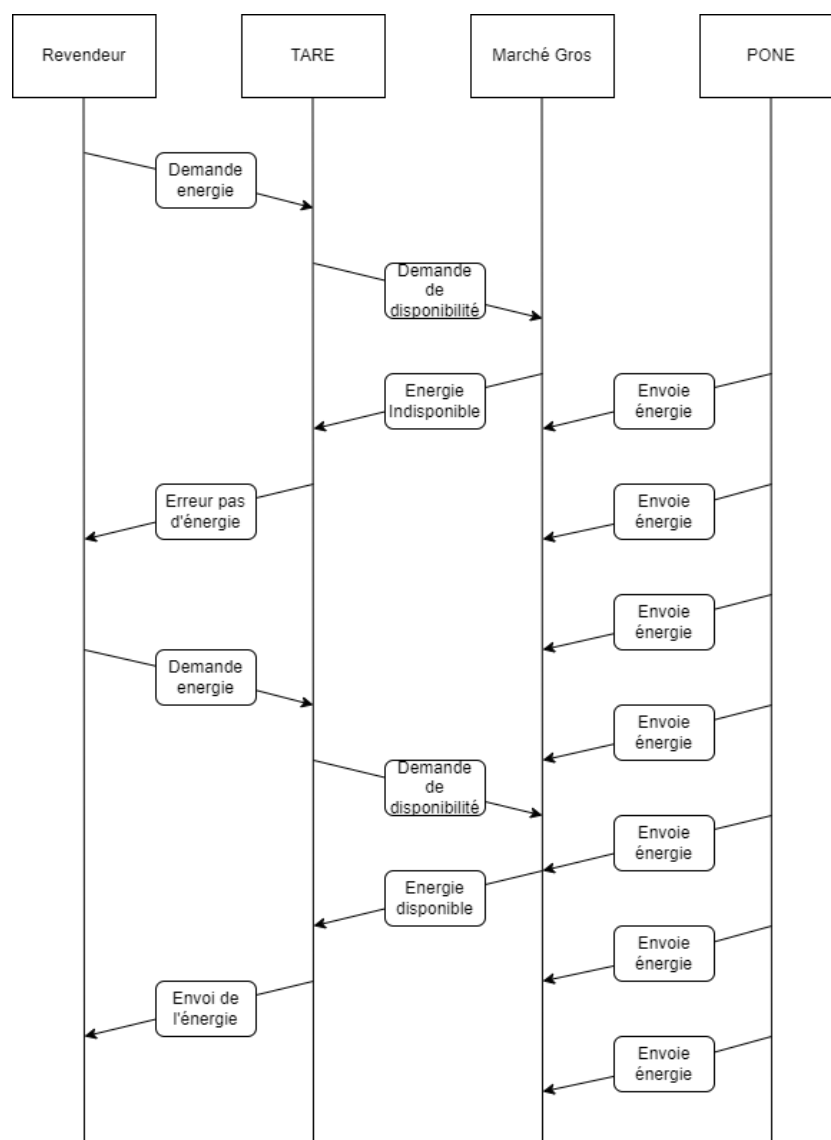


Figure 20 – Schéma du scénario B

5.3 Scénario C

Dans un système ayant un seul PONE et un seul TARE (en plus du revendeur, du marché de gros et de l'AMI), la demande du client ne peut être satisfaite à cause d'une contrainte que le PONE ne peut satisfaire. Par contre, pendant que le système fonctionne, un PONE est rajouté et il peut fournir l'énergie demandée (en une seule production ou au bout d'un certain nombre de production). L'achat d'énergie devient possible et le reste du déroulement respecte le modèle classique.

Dans notre projet, nous démarrons un PONE 30 secondes après le démarrage du lanceur. Ce PONE qui est le PONE pétrole va alors envoyer de l'énergie au marché de gros tous comme les autres PONE. Dans notre projet, pour être sûr que celle-ci soit satisfaite dès la 1ère demande (une fois le PONE en marche), nous avons décidé de demander au revendeur du Pétrole mais sans type d'extraction précise avec une quantité de 1.

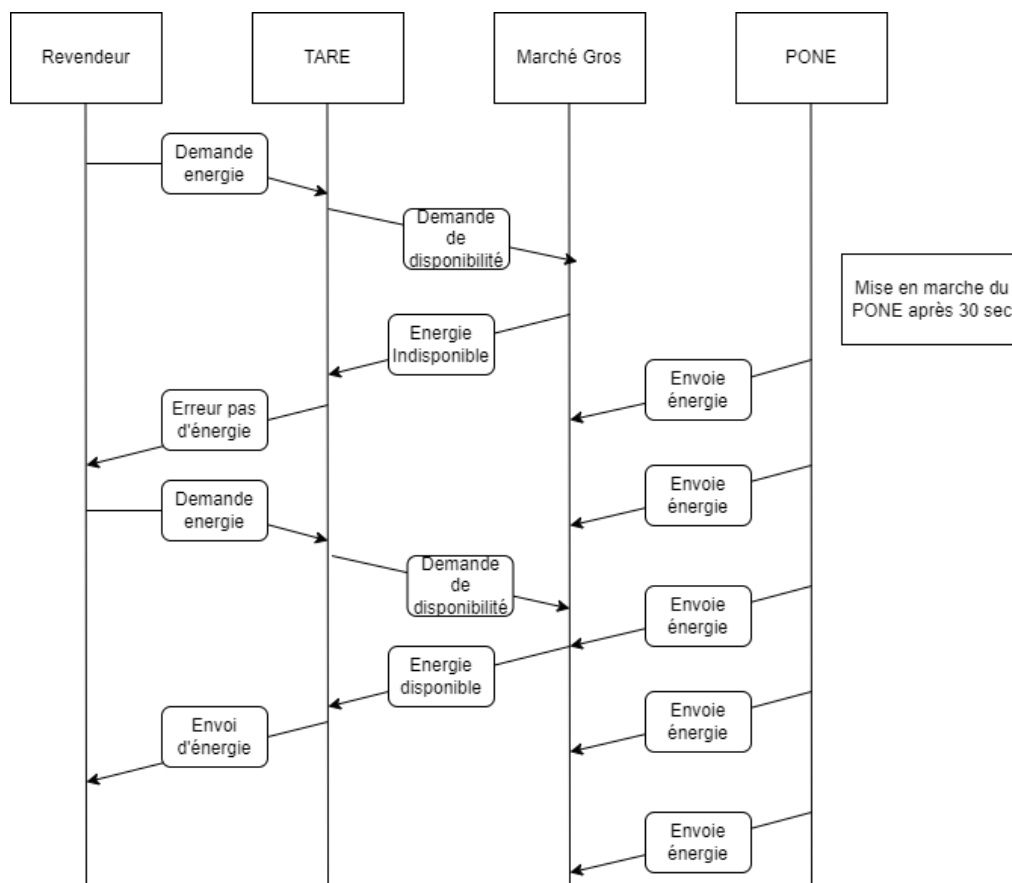


Figure 21 – Schéma du scénario C

5.4 Scénario D

Dans un système ayant deux PONEs et deux TAREs (en plus du revendeur, du marché de gros et de l'AMI), pour être satisfaite rapidement, la demande du client doit se baser sur la somme des énergies fournies par les 2 PONEs. Par contre, les TAREs ne peuvent acheter qu'à un seul lot d'énergie d'un PONE à la fois (via le marché de gros). C'est donc le revendeur qui doit gérer le rassemblement des 2 retours de TARE pour construire la réponse au client.

Ce scénario n'a pas été réalisé dans notre projet par manque de temps car nous devons réaliser des modifications importantes dans notre code et finir le rapport, nous avons pris la décision de ne pas le faire. Cependant nous avons quand même réfléchi à celui-ci et comment nous l'aurions implémenté si nous avions eu le temps.

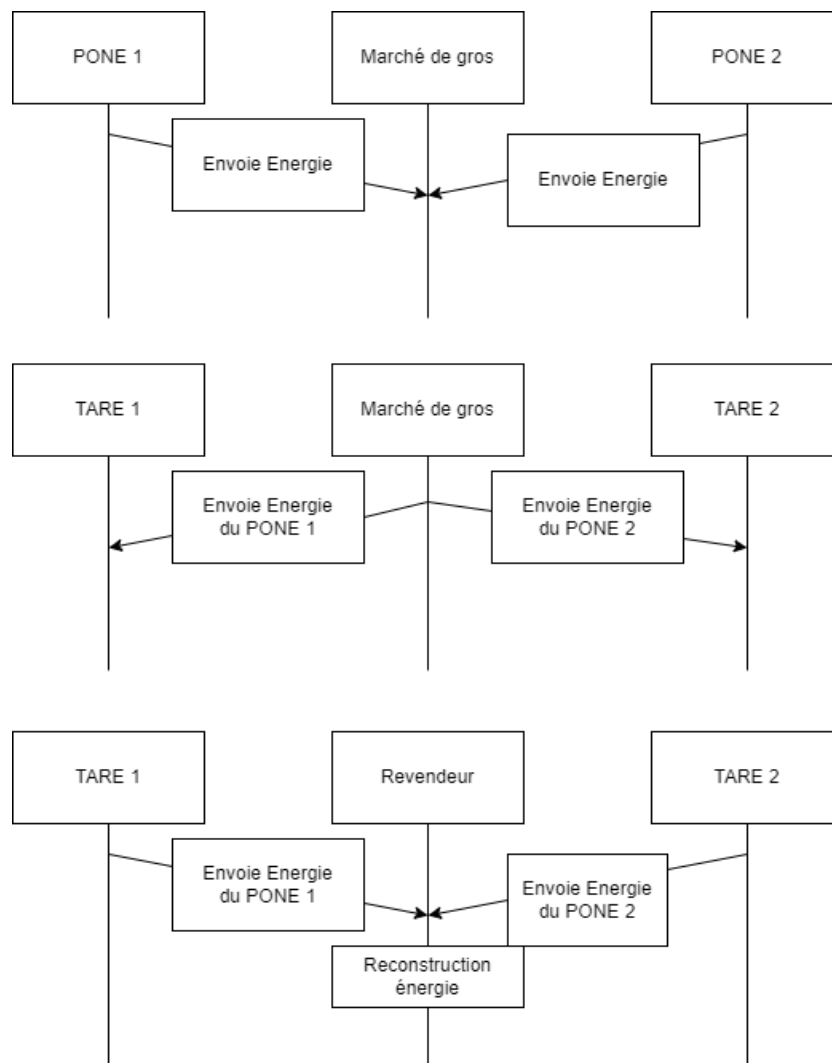


Figure 22 – Schéma du scénario D

5.5 Scénario A2

Il s'agit du scénario A dans lequel, à la fin de l'achat de l'énergie, le client demande à faire vérifier le numéro de suivi de l'énergie achetée. Cela signifie qu'il faut valider le CRACHA et le CRADO qu'il contient auprès de l'AMI qui est l'autorité de certification.

Ce scénario n'a pas été traité dans notre projet, mais voici à quoi cela aurait du ressemblé :

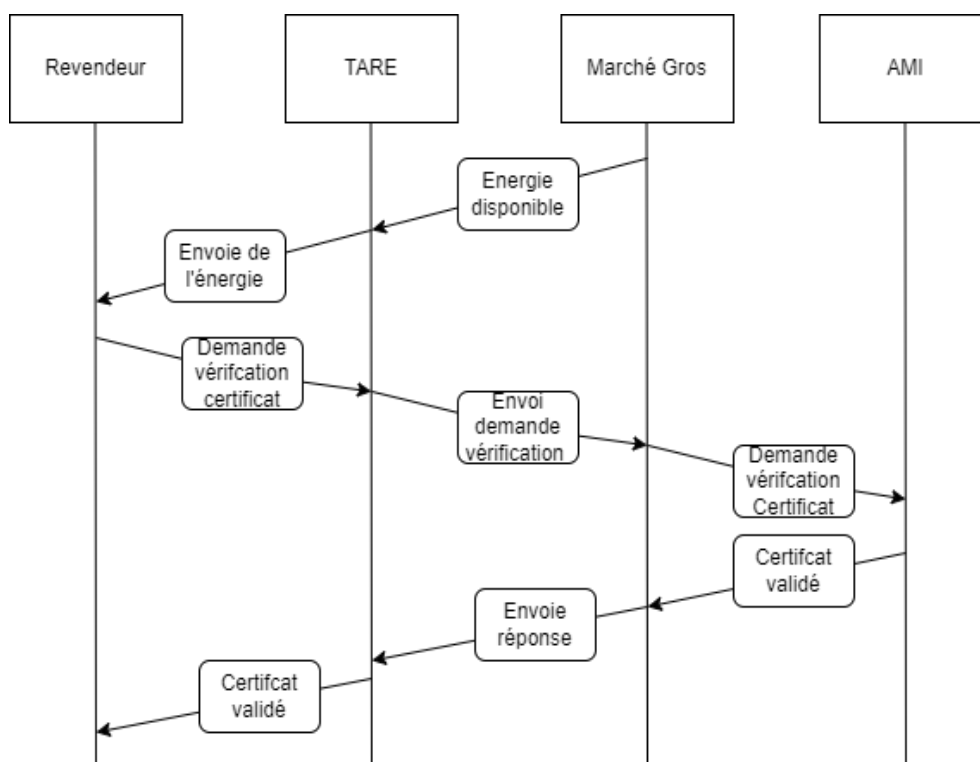


Figure 23 – Schéma du scénario A2

6 Conclusion

L'architecture SALE est un réseau complexe de communications utilisant différentes technologies, tels que de l'UDP du TCP, mais aussi de l'HTTP. L'architecture implémentée en JAVA et PHP est entièrement fonctionnel et toutes les entités communiquent entre elles. Différents scénarios sont disponibles tels que le A, le B ou encore le C. De plus, les communications entre le marché de gros et l'AMI est crypté en RSA permettant de sécuriser leurs échanges.

Finalement, avec un peu plus de temps, nous aurions pu mettre en place le système de certificat d'origine (CRADO) et certificat d'achat (CRACHA), ainsi que les deux scénarios manquant, à savoir le D et A2.

Table des figures

1	Positionnement dans le dossier	4
2	Succès de la compilation	4
3	Erreur de lancement	4
4	Succès du lancement	5
5	Interface WEB	5
6	Dossier à sélectionner	6
7	Bouton Run Rebug	6
8	Erreur de compilation VSCode	7
9	Ajout de "args"	7
10	Lancement du serveur	7
11	Schéma explicatif HTTP simplifié	8
12	Schéma explicatif HTTP détaillé	8
13	Schéma explicatif UDP	9
14	Schéma explicatif TCP	9
15	Modélisation du système d'achat d'énergie	10
16	Modélisation de la relation entre Revendeur et TARE	11
17	Modélisation de la relation entre Marché de Gros et PONE	12
18	Modélisation de la relation entre Marché de Gros et PONE	13
19	Schéma du scénario A	14
20	Schéma du scénario B	15
21	Schéma du scénario C	16
22	Schéma du scénario D	17
23	Schéma du scénario A2	18
+		