Remerciement

Avant de débuter ce rapport, ce projet est la récolte de mon travail de quatre mois. En préambule, je commence par exprimer mon profonde reconnaissance et mes vifs remerciements à mon encadrant ***M. Mohamed Amine TRIGUI****, vous m'avez toujours réservée le meilleur accueil, malgré vos obligations professionnelles. Vos conseils et votre suivi méritent toute reconnaissance.*

Pour ses encouragements, ses conseils, merci de m'avoir guidé avec patience je veux sincèrement exprimer mon respect et mon gratitude.

Merci à ***M. Amir GARGOURI*** *et* ***M. Souleymen SMAOUI*** *d'avoir accepté d'examiner et de juger ce travail.*

Je tiens également à remercier ***Mme Nourchène BARADAI****, mon encadrante académique, pour son suivi et ses conseils précis.*

Et enfin, j'adresse mes sincères remerciements à ma famille et tous mes proches et mes amis. Qui m'ont accompagné, soutenu et encouragé tout au long de la réalisation de ce projet.

Sommaire

Liste des figures

Liste des Tableaux

Liste des abréviations

API : Application Programming Interface

FPGIA : Field-Programmable-Gate-Arrays

FTP : File Transfert Protocole

GPIO : General Purpose Input/Output

HAT : hardware Attached on Top

HTTP : Hyper Text Transfer Protocol

HTTPS : HyperText Transfer Protocol Secure

JSON : JavaScript Object Notation

I2C : Inter-Integrated Circuit

MISO : Master Input Slave Output

MOSI : Master Output Slave Input

OAS : OpenAPI Specification

PAN : Permanent Account Number

RFID : Radio Frequency Identification

REST : REpresentational State Transfer

RPI : Raspberry pi

SCLK : Serial Clock

SPI : Serial Peripheral Interface

SSH : Secure Shell

SS : Slave Select

TELNET : Telecom Networks Engineering

TPE : Terminal de Paiement Électronique (Point Of Sale terminal)

UML : Unified Modelling Language

UART : Universal Asynchronous Receiver / Transmitter

VLSI : Very-Large-Scale-Integration-Circuits

XML : eXtensible Markup Language

Introduction Générale

La monétique occupe désormais une place prépondérante dans le monde des paiements. Son processus de traitement évolue constamment et intègre régulièrement des technologies de pointe pour répondre aux exigences des clients. Complexe et organisé en nombreux domaines, le monde de la monétique nécessite des spécialistes à forte compétence et en évolutions permanentes. Cette formation permet de donner une meilleure visibilité sur les évolutions de la monétique et des nouveaux moyens de paiement **[1].**

Dans ce cadre, TELNET, la société accueillante, s'engage dans un projet innovant intitulé **« Conception et Développement d'un Hub de carte à puce avec Raspberry PI ».**

Ce rapport retrace l'évolution de notre projet ainsi que notre travail que nous avons effectué durant quatre mois. Nous sommes amenés à réaliser un système de multiplexage des cartes à puce avec des terminaux de paiement. Cette solution comprend deux parties « hardware » et « software ». Dans notre rapport nous allons focaliser sur la partie software qui a pour but de concevoir et développer un serveur web offrant des API REST qui assurent un multiplexage puis il sera intégré dans un environnement de test automatique.

Le rapport est divisé en quatre chapitres avec une annexe:

Le premier chapitre est basé sur la présentation de la société accueillante, des concepts fondamentaux et du contexte général du projet. Ensuite nous passons à une description de notre projet, suite à une étude et à une critique de l'existant.

Le deuxième chapitre est réservé à présenter toutes les bases théoriques et techniques nécessaires à la réalisation de notre projet, à savoir le matériel électronique, les outils informatiques et les logiciels.

Le troisième chapitre est consacré à l'analyse et la spécification des besoins, ainsi que la description de l'architecture générale utilisée dans la réalisation de notre projet.

Le dernier chapitre est consacré à la réalisation de notre projet, en détaillant la démarche suivie et en exposant les résultats obtenus, et en fin nous précisons nos perspectives.

Ce projet se termine par une conclusion générale, et une annexe qui montre quelques généralités sur le support matériel sous Linux.

Chapitre 1 :
Présentation Générale du projet

Introduction

Ce chapitre a pour objectif de situer notre projet dans son contexte général. D'emblée, nous commençons par une présentation d'une vue globale sur l'organisme d'accueil, puis nous allons définir le domaine de la monétique tout en donnant une explication de ses concepts de base. Ensuite, nous décrivons brièvement notre projet et ses objectifs suite à une étude et critique de l'existant. Finalement, nous clôturons par une présentation de méthodologie de gestion de travail.

1 Présentation de l'organisme d'accueil

Nous présentons dans ce paragraphe l'organisme d'accueil ainsi que les secteurs d'activités dans lesquels il agit.

1.1 Présentation de TELNET group

TELNET est un groupe d'ingénierie offshore tunisien travaillant pour de grands groupes internationaux. C'est aussi un cabinet de conseil en innovation et hautes technologies. Fondée en 1994 sous la dénomination TELECOM NETWORKS ENGINEERING en abrégé «TELNET » et spécialisée dans l'ingénierie logicielle et matérielle, l'intégration réseau et études mécaniques. TELNET est la première société du Maghreb à obtenir la certification CMMi modèle niveau 5 en 2006. Le groupe offshore est certifié ISO 9001 pour la qualité de son système de management. En 2001, le groupe TELNET a obtenu le prix de l'innovation de la Chambre de commerce et d'industrie franco-tunisienne. [**2**]

Figure 1 : Logo de l'entreprise TELNET.

1.2 Secteurs d'activité

Avec plus de 20 ans d'existence et plus de 600 employés, TELNET a développé une expertise en ingénierie de produits dans plusieurs secteurs:

- **Telecom et Multimédia :** Dans ce secteur TELNET s'engage à concevoir, développer et valider du matériel et logiciel dans le domaine d'accès au réseau, décodeur TV, produit de traitement audio et vidéo, équipement de mesure et test ainsi que les compteurs d'énergie.
- **Automobile :** Conception, développement et validation logiciel des ordinateurs automobiles et des tableaux de bord.
- **Avionique :** Conception, développement et validation logiciel des modules pour ordinateurs avioniques selon la norme DO178B / C, ainsi que développement des bancs de test.
- **Systèmes et sécurité :** Développement des solutions pour la gestion des documents électroniques basé sur la biométrie et le développement des solutions spécifiques basé sur des architectures web et des bases des données.
- **Monétique :** Développement des applications sur des terminaux de paiement, ainsi que le développement des protocoles de paiement et des solutions de paiement électronique.

2 Cadre général du projet

Notre projet réalisé au sein de l'entreprise TELNET, fait partie de ces projets dans le domaine de la monétique.

2.1 Monétique

La monétique [**3**] est la contraction de Monnaie et Electronique. Ce terme signifie l'ensemble des dispositifs utilisant l'électronique et l'informatique pour développer les transactions bancaires. Elle permet les échanges d'argent de manière totalement dématérialisée.

2.1.1 Carte bancaire

C'est une carte plastique qui représente un moyen de paiement équipé d'une bande électronique appelée carte à piste ou d'une puce électronique appelée carte à puce ou bien les carte sans contact qui utilise la technologie RFID. Les cartes bancaires sont les supports de base de la monétique qui réduisent les risques liés au paiement comme la perte ou l'insuffisance de l'argent. Dans notre projet nous avons utilisé une carte à puce de test. Nous pouvons visualiser des exemples illustrés dans la figure 2.

Figure 2: Exemples de carte de paiement.

2.1.2 Transaction bancaire

C'est le flux qui permet la réalisation d'un paiement par une carte bancaire. Quatre acteurs interviennent dans les échanges de flux en monétique.

- **Emetteur** : Il s'agit de l'organisme financier (par exemple une banque) qui met à disposition de son client (le porteur) le support de transaction bancaire (la carte de paiement).
- **Porteur** : Un utilisateur autorisé d'une carte de paiement.
- **Accepteur** : Il s'agit du commerçant, artisan, ou profession libérale qui accepte les moyens de paiement électronique en guise de règlement.
- **Acquéreur** : Organisme financier qui met à disposition de son client des services d'acquisition de transactions de paiement électronique.

La figure 3 représente le processus de transaction bancaire.

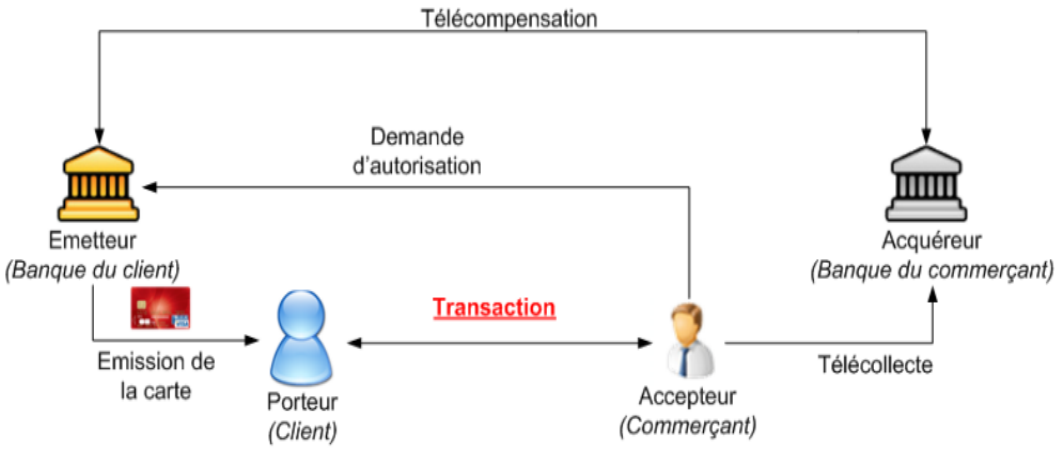


Figure 3: Transaction monétique.

L'opération d'achat d'un bien ou d'un objet se déroule sur les étapes suivantes :

- Le commerçant saisit le montant d'achat ou de remboursement sur le terminal de paiement.
- Le client par la suite insert sa carte bancaire.

- Le terminal de paiement lit les données de la carte principalement le PAN. En cas d'impossibilité de lecture la carte sera rejetée par le terminal de paiement.
- Par la suite, le client saisit son code pin fourni par sa banque.
- Une demande d'autorisation et donc envoyée par le terminal de paiement vers la banque du client.
- Cette banque va transmettre cette demande vers la banque du commerçant via le réseau interbancaire
- La banque du commerçant traite cette demande et transfère la réponse à ce dernier si la réponse est favorable l'opération de télé compensation est réalisée c'est-à-dire qu'un compte va être débité et l'autre crédité, sinon le refus de l'opération est envoyé vers le commerçant et le client.

2.1.3 Terminal de paiement

Le terminal de paiement électronique, est un appareil électronique capable d'effectuer une transaction bancaire grâce à une carte électronique. Il est caractérisé par quatre fonctionnalités majeures :

- L'échange d'informations avec des cartes de paiement.
- La réalisation de divers contrôles (sécurité, validité, etc.).
- La transmission des transactions vers l'acquéreur (autorisation, télécollecte).
- L'enregistrement d'une transaction dans un serveur distant.

2.1.4 Types des terminaux de paiement

La société qui fabrique les terminaux de paiement INGENICO met à la disponibilité de ses clients une large gamme de terminaux de paiement. **Fixe, Mobile, Portable.**

La figure 4 représente ces différents types de terminaux de paiement.

Figure 4: Les types de terminaux de paiement.

3 Présentation du projet

Pour bien définir notre projet nous avons mené initialement une étude de l'existant suivie d'une critique, pour introduire finalement notre projet comme étant une proposition de la solution.

3.1 Etude de l'existant

Le test Automatisé est l'utilisation d'un scénario programmé pour tester une ou plusieurs fonctions d'une application, d'un site Web, ou d'un logiciel à intervalles réguliers et/ou à travers des plates-formes multiples, ou bien dans notre cas pour tester les fonctions des cartes de paiement.

La méthode actuelle de travail utilisée dans le test automatique présentée dans (la figure 5) exige à chaque fois l'utilisation d'un seul terminal et une seule carte à puce. Donc nous ne pouvons pas lancer plusieurs testes automatiques en même temps sur plusieurs cartes à puce d'où le besoin de changement de la carte à chaque fois et ça s'est fait d'une manière manuel.

Figure 5 : La méthode actuelle de test automatique.

3.2 Critique de l'existant

Le test automatisé mène à une exécution plus rapide de test. Nous pouvons accélère encore cette opération par l'utilisation d'un dispositif qui nous permet d'utiliser plusieurs cartes à la fois mais ce type d'appareils n'est pas toujours disponible dans le marché, il est très chère et besoin de maintenance après chaque période d'utilisation. Donc nous pouvons dire que la méthode de travail actuelle n'est pas pratique et n'est pas rapide et l'existant a beaucoup des inconvénients. D'où le besoin de la fabrication d'un système qui nous permet de faire plusieurs tests automatiques en parallèle et ça économisera plus de temps et plus d'argent par la suite.

3.3 Solution proposée

L'idée de notre solution est de réaliser un système de multiplexage des cartes à puce avec des terminaux de paiement, cette solution comprend deux parties « hardware » et « software ».
Dans notre rapport nous allons focaliser sur la partie software qui a pour but de concevoir et développer un serveur web offrant des API REST qui assurent un multiplexage puis il sera intégré dans un environnement de test automatique.

La figure 6 illustre le schéma bloc de notre système.

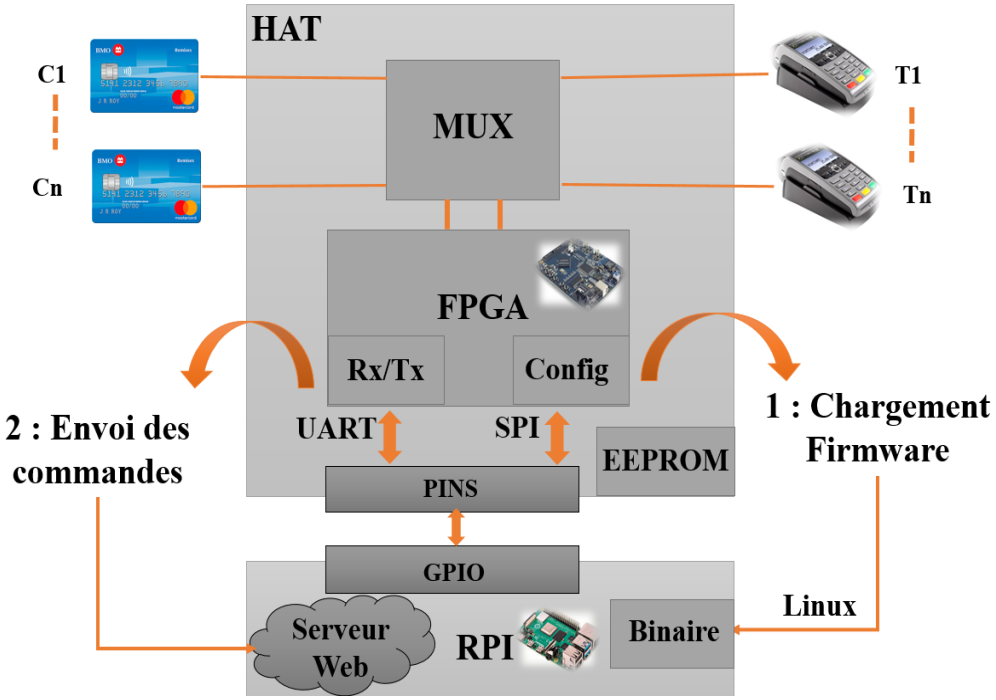


Figure 6: Le schéma bloc de la solution.

Le schéma bloc de notre projet comprend une carte Raspberry pi et une carte d'extension HAT qui est équipée d'une mémoire EEPROM, Et elle est connectée avec la RPI via les GPIOs. Afin d'assurer le multiplexage des cartes à puce avec des terminaux de paiement, nous avons utilisé un multiplexeur et une carte FPGA.

Notre FPGA est connectée aux pins de sélection de multiplexeur. Le multiplexeur celui qui va faire le multiplexage entre les cartes et les terminaux.

La partie software de notre projet comprend deux grandes étapes :

- **Premièrement au niveau de noyau linux :** le chargement de firmware dans la carte FPGA à partir de la RPI c'est-à-dire il y a un binaire qui va être compilé et transféré vers la FPGA

et ça s'est fait à partir des pins de configuration SPI. C'est une étape primordiale qui sera faite automatiquement juste après le démarrage de système.

- **Deuxièmement au niveau d'application :** le développement d'un serveur web qui doit offrir des API REST qui assurent le multiplexage des cartes à travers l'envoi des commandes via la liaison série en utilisant l'UART.

Ce serveur doit être intégré dans un environnement de test automatique pour lancer l'exécution des suites de tests avec le hub de carte. Ainsi, la possibilité de lancer plusieurs tests automatiques en même temps sur plusieurs cartes à puce sera envisageable.

4 Méthodologie de gestion du projet

4.1 Cycle de vie

Le cycle de vie d'une application comprend toutes les étapes depuis sa conception et sa réalisation jusqu'à sa mise en œuvre. L'objectif est de permettre de définir des jalons intermédiaires permettant la validation du développement du logiciel et la vérification de son processus de développement. Le cycle de vie permet de détecter les erreurs le plutôt possible.

4.1.1 Modèle de cycle de vie en V

Le modèle du cycle en V est un standard de développement informatique qui vient pour pallier les limites de la méthodologie classique en termes de gestion de risque. Le principe de base du cycle en V est de réaliser les différentes étapes du projet les unes après les autres. Cette méthodologie est applicable nécessairement pour de petits projets qui nécessitent une période de développement de moins qu'un an et des projets à faible risque.

Les avantages du modèle du cycle de vie en V sont les suivants :

- La qualité de la mise en œuvre des tests.
- Modèle éprouvé dans l'industrie.
- Deux types de tâches sont réalisés en parallèle : Verticalement pour préparer l'étape suivante et Horizontalement : pour préparer la vérification de la tâche en cours.

Les différentes étapes de la méthodologie de développement cycle en V s'illustrent dans la figure 7.

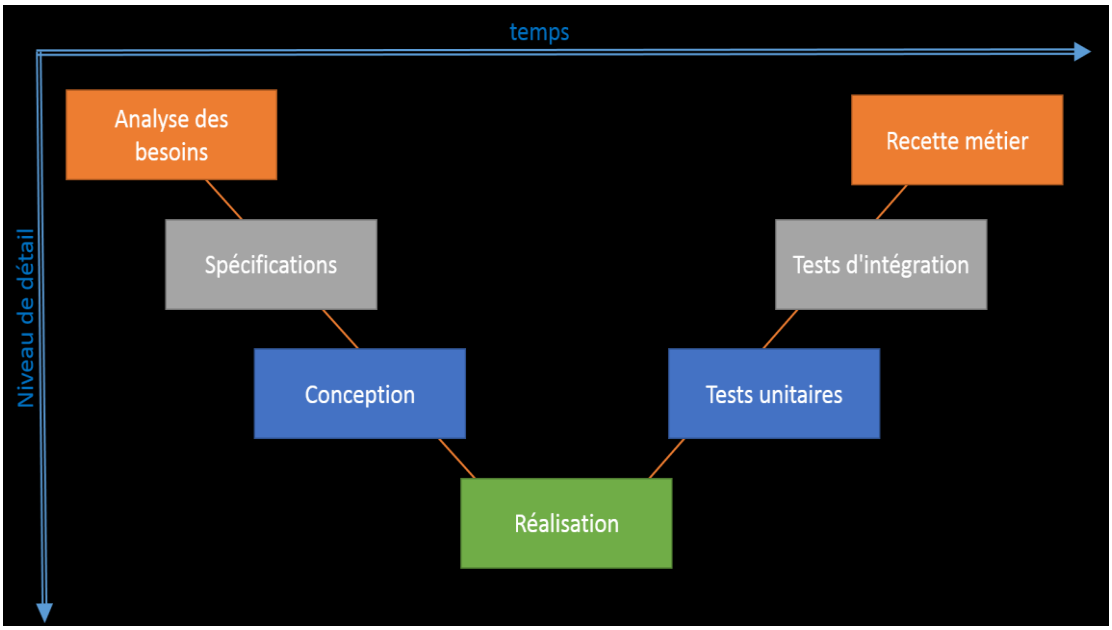


Figure 7 : Le modèle conceptuel du cycle en V.

4.1.2 Choix de modèle de cycle de vie

Afin de concevoir et développer notre serveur, nous avons opté pour le modèle de cycle de vie en V. Ce choix reviens au fait que ce cycle est le plus efficace avec son principe de travail qui nécessite la vérification de chaque étape et la possibilité de corriger les fautes avant de se lancer vers l'étape suivante.

4.2 Diagramme de Gantt

Au cours de la période du stage, un ensemble de points de contrôle a été mis en place pour bien établir un suivi du déroulement des étapes du projet. Le diagramme de Gantt est un outil de planification des tâches nécessaires pour la réalisation d'un projet quel que soit le secteur d'activité. Il s'agit d'une représentation d'un graphe connexe et orienté, il permet de visualiser dans le temps les diverses tâches composant un projet de manière simple et concise, de planifier et suivre les besoins en ressources humaines et matérielles et donc de pouvoir suivre l'avancement du projet.

Le diagramme présenté dans la figure 8, va représenter les tâches principales réalisées dans notre projet.

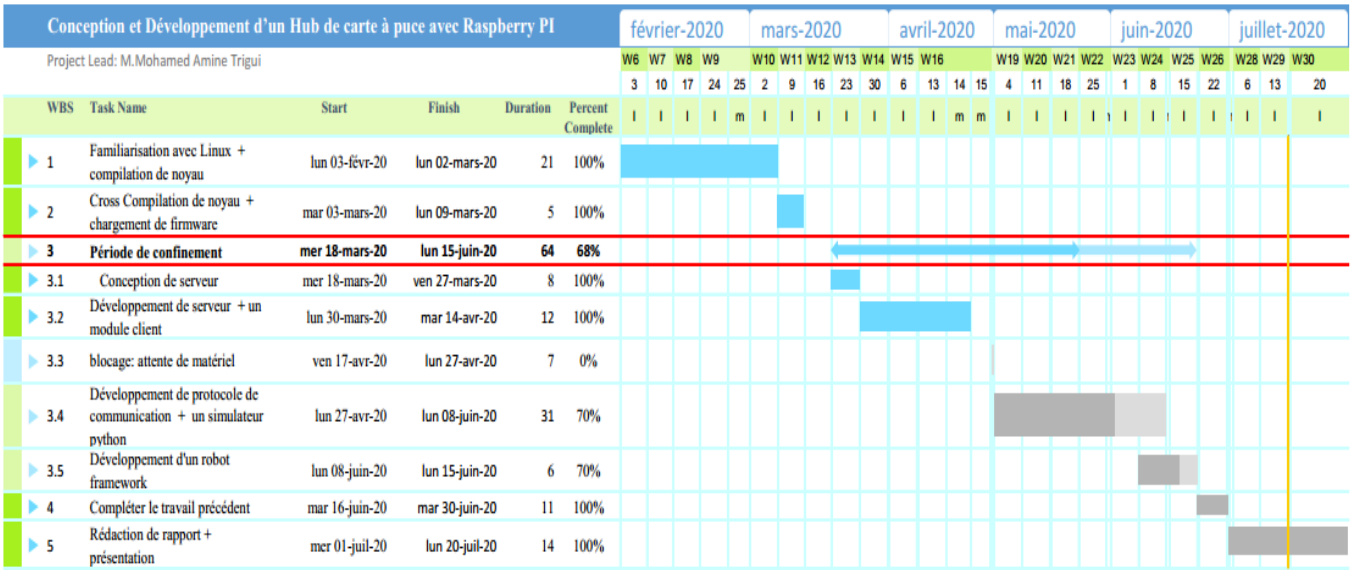


Figure 8: Diagramme de Gantt.

Conclusion

L'objectif de ce chapitre était de présenter l'environnement de stage, d'exposer une vue générale du contexte du travail, d'analyser l'existant et aussi une description brève de notre projet et l'objectif à atteindre et en fin nous avons parlé sur le choix de modèle de cycle de vie ainsi la présentation de plan de travail de projet.

Dans le chapitre suivant nous allons citer les différentes bases théoriques et techniques nécessaires à la réalisation de notre projet.

Chapitre 2 :
Fondements Théoriques

Introduction

Ce chapitre vise à présenter toutes les bases théoriques et techniques nécessaires à la réalisation de notre projet. Tout d'abord, nous présentons le matériel électronique que nous avons utilisé. Ensuite, nous présentons les outils informatiques et logiciels (logiciel de développement de programmes d'application et d'autre qui permettant de mettre en place facilement un serveur Web et d'autre application qui permet de créer des fantastiques interfaces pour notre projet).

1 Environnements matériels

1.1 Raspberry PI et ses équipements

Le Raspberry Pi **est une nano-ordinateur mono-carte à processeur ARM** conçu par des professeurs du département informatique de l'université de Cambridge dans le cadre de la fondation Raspberry Pi. Plusieurs versions des cartes du type Raspberry Pi ont été produites et vendues depuis leur lancement, ces versions sont composées de trois différents modèles qui sont les modèles A, B et Zéro. Chacun de ces modèles est caractérisé par un certain nombre d'avantages par rapport aux autres modèles. Dans notre projet nous avons utilisé le dernier module Raspberry Pi 4 model B.

1.1.1 Tour d'horizon du Raspberry Pi

Les différentes parties de Raspberry Pi 4 sont présentés par la figure 9.

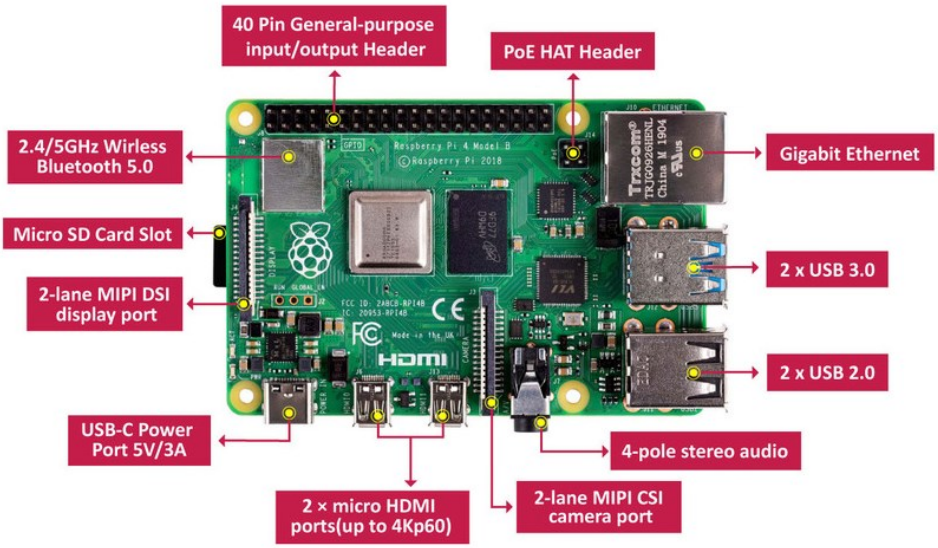


Figure 9: Les différentes parties de Raspberry pi 4.

1.1.2 Caractéristiques principales

- **Processeur :** Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
- **RAM :** 1GB, 2GB, ou 4GB de mémoire SDRAM LPDDR4
- **GPU :** Video Core VI prenant en charge OpenGL ES 3.0, décodage HEVC 4K à 60 i/s
- **Connexion sans fil :** Bluetooth 5.0, Wi-Fi 802.11b/g/n/ac
- **Connexion filaire :** Gigabit Ethernet (RJ45)
- **Ports :** 2 x USB 3.0 / 2 x USB 2.0 / 1 x GPIO 40 pin / 1 x port quadripôle Audio/Vidéo composite / 2 x micro-HDMI
- **Alimentation :** 5V DC via un connecteur USB-C (minimum 3A), 5V DC via un en-tête GPIO (minimum 3A), compatible Power over Ethernet (PoE) (nécessite un HAT pour PoE) **[4].**

1.1.3 Connecteurs d'extension

- **Portes GPIO**

Les portes GPIO illustrés dans la figure 10, sont des portes d'entrées-sorties développées en 1980 pour faire communiquer les circuits électroniques des composants électroniques. Ces pins sont généralement alimentés en 3,3V et ne peuvent émettre en contrepartie que des courants de faible intensité.

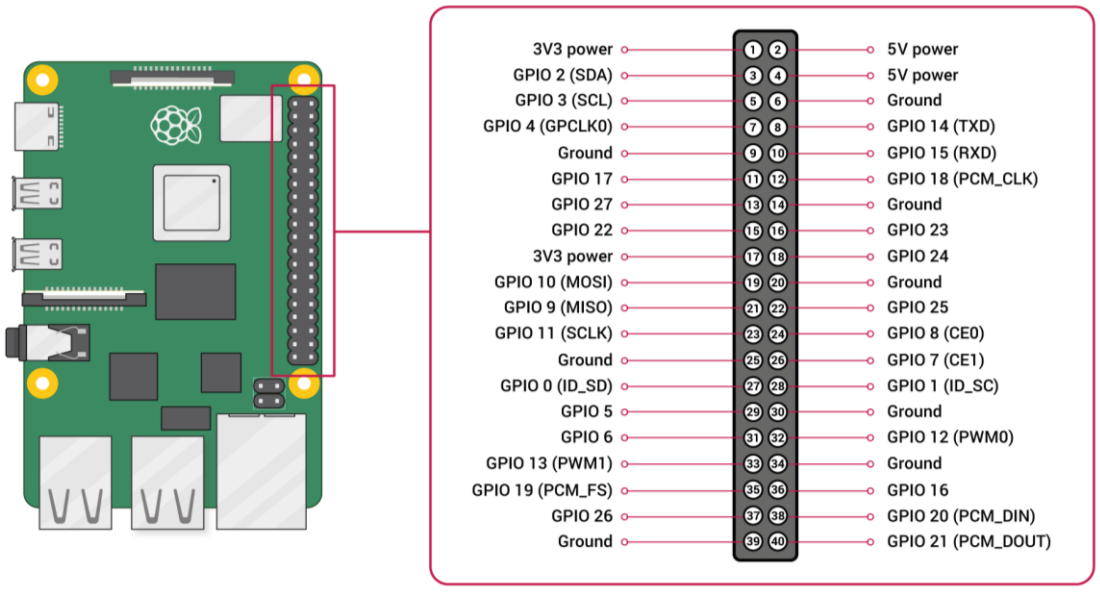


Figure 10: Description des broches GPIO.

- **Bus I2C**

L'I2C présenté dans la figure 11 est un bus série développé par Philips en 1982 afin de faire transmettre les informations entre des différents composants électroniques d'une façon asynchrone à travers seulement 3 fils :

- Signal de donnée : SDA.
- Signal d'horloge : SCL.
- Signal de référence électrique : masse.

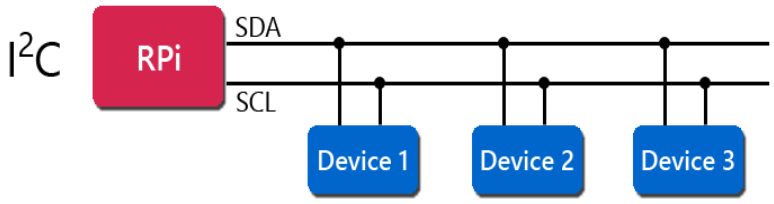


Figure 11: Schéma de câblage I2C.

- Le protocole de liaison est du type Maître/Esclave.
- Chaque circuit connecté dispose de sa propre adresse.
- La vitesse de transfert peut atteindre les 100 kbits/s.

- **Port série du Raspberry Pi**

Les modèles B, B+ et Pi Zero W de Raspberry Pi contiennent deux contrôleurs UART qui peuvent être utilisés pour la communication série, les mini UART et le PL011. Par défaut, le mini UART est mappé au TXD (GPIO 14) et au RXD (GPIO 15) sur le connecteur GPIO 40 broches et l'UART PL011 est utilisé pour le module Bluetooth/Wireless. Mais ces deux modules peuvent être reliés au port GPIO.

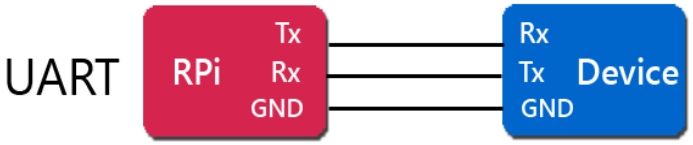


Figure 12 : Schéma de câblage UART.

- **Bus SPI**

Le SPI (figure 13) permet une communication entre un maître et un ou plusieurs esclaves. Le maître impose la fréquence d'horloge et sélectionne l'esclave auquel les données sont envoyées. La ligne MOSI permet d'envoyer des données depuis le maître vers l'esclave. La ligne MISO

est en haute-impédance jusqu'au moment où l'esclave est sélectionné et où il doit envoyer des données.

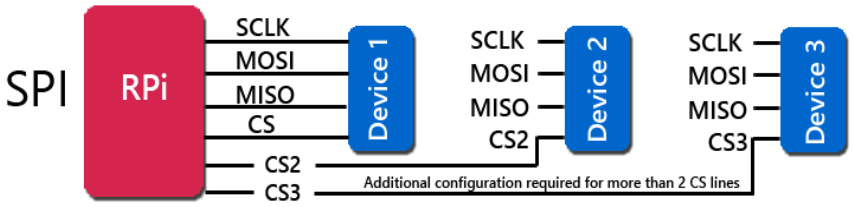


Figure 13: Schéma de câblage SPI.

1.1.4 Système d'exploitation pour Raspberry PI

Étant donné que la Raspberry PI est un petit ordinateur, un petit système d'exploitation doit être installé pour bénéficier de la plupart des fonctionnalités de la RPI. Un grand nombre de systèmes d'exploitation est disponible pour être installé sur Raspberry PI dont le plus puissant, efficace et populaire est le système d'exploitation officiel basé sur Debian annoncé par le site Web de Raspberry Pi est le Raspbian car c'est un système d'exploitation officiel qui a des milliers de bibliothèques préconstruites pour effectuer de nombreuses tâches et optimiser le système d'exploitation et prendre en charge le langage Python.

Figure 14: Système d'exploitation Raspbian.

1.2 Cartes d'extensions pour Raspberry PI

Le concept de cape était un moyen fantastique d'étendre les fonctionnalités du Pi, et ajoute une caractéristique de différenciation clé; une petite EEPROM avec un ID d'appareil. Cette EEPROM contient un ID (appelé un fragment d'arborescence des périphériques) qui, une fois attaché au Pi, permet d'identifier et de charger automatiquement les pilotes. Chaque HAT doit avoir un ID correspondant spécifique à ce type de périphérique.

L'autre partie de notre projet est la conception d'une HAT qui contient essentiellement les composants suivants :

- Une carte FPGIA : ''ALTERA CYCLONE IV''
- Un multiplexeur : ''MUX MAX4639''
- Un contrôleur coté carte : ''NCN8025A''

- Un contrôleur coté terminal : ''TX0108e''

1.3 Machine hôte

Pour le développement de la solution, nous avons utilisé un ordinateur de bureau ayant les caractéristiques suivantes :

- **Processeur :** Intel(R) Core(TM) 2 Duo CPU E7500 @ 2.93GHz
- **Mémoire :** 8 Go de RAM
- **Disque dur :** 465Go
- **Système d'exploitation :** Linux Syphax OS

2 Outils informatiques

2.1 Système d'exploitation Linux

Linux est un système d'exploitation, ressemble à un système d'exploitation UNIX - un système d'exploitation multi-tâches, multi-utilisateurs, stable, et qui a été assemblé en tant que logiciel gratuit et open-source pour le développement et la distribution.

2.1.1 Noyau

Un noyau est un composant central d'un système d'exploitation considérée comme une interface entre les applications utilisateur et le matériel. Le but du noyau est de gérer la communication entre le logiciel (applications de niveau utilisateur) et le matériel (CPU, mémoire disque, etc.).

2.1.2 Architecture Du Noyau

Le système d'exploitation Linux est constitué de trois couches, comme le montre la figure 15, qui interagissent entre eux et qui doivent fonctionner de concert afin d'assurer la bonne marche du système et répondre aux besoins de l'utilisateur **[5].**

- **Espace utilisateur**

L'espace utilisateur est l'espace en mémoire où s'exécutent les processus utilisateur. Cet espace est protégé. Le système empêche un processus d'interférer avec un autre processus. Seuls les processus du noyau peuvent accéder à un processus utilisateur.

- **Espace noyau**

L'espace du noyau est l'espace en mémoire où s'exécutent les processus du noyau. L'utilisateur n'y a accès que via l'appel système.

❖ **Appel système**

L'espace utilisateur et l'espace noyau sont dans des espaces différents. Lorsqu'un appel système est exécuté, les arguments de l'appel sont passés de l'espace utilisateur à l'espace noyau. Un processus utilisateur devient un processus noyau lorsqu'il exécute un appel système.

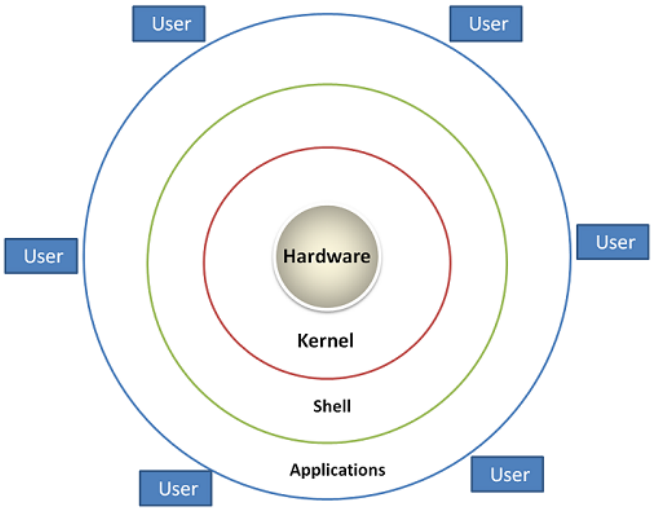


Figure 15 : Architecture du système Linux.

2.1.3 Tâches principales du noyau

Les tâches principales du noyau sont :

- Système de fichiers
- La gestion des processus
- Pilote de périphérique
- Gestion de la mémoire
- La mise en réseau

Nous allons intéresser sur les pilotes de périphériques et l'ensemble de la communication entre le matériel du périphérique et le noyau qui sont les deux tâches les plus importants dans le chargement d'un firmware.

❖ **Pilote de périphérique**

L'un des objectifs d'un système d'exploitation est de cacher le matériel du système à l'utilisateur. Au lieu de mettre du code pour gérer le contrôleur HW dans chaque application, le code est conservé dans le noyau Linux. Il résume la manipulation des appareils. Tous les appareils HW

ressemblent à des fichiers normaux. Chaque appareil à son propre protocole et évidemment ce protocole doit être supporté par le noyau, ce qui signifie l'existence d'un pilote de périphérique intégré dans le noyau capable de communiquer avec l'appareil matériel par exemple le protocole USB.

- **Device tree**

Une arborescence de périphériques est une structure de données arborescente avec des nœuds qui décrivent les périphériques physiques sur la carte. Lors de l'utilisation de l'arborescence des périphériques, le noyau ne contient plus la description du matériel, il est situé dans un blob binaire distinct appelé le blob de l'arborescence des périphériques.

2.2 Langage de programmation : Python

Le langage de programmation que nous avons choisi c'est le python, son logo est présenté par la figure 16 qui est un langage recommandé pour les nouvelles technologies comme par exemple le Raspberry, arduino, etc. Avec la complexification du web et l'accumulation des données, Python a pris une place majeure dans le développement informatique car il peut être utilisé dans des situations variées, dans tous les secteurs d'activité. Il est le langage de programmation open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels. En effet, parmi ses qualités, Python permet notamment aux développeurs de se concentrer sur ce qu'ils font plutôt que sur la manière dont ils le font. Il a libéré les développeurs des contraintes de formes qui occupaient leur temps avec les langages plus anciens. Ainsi, développer du code avec Python est plus rapide qu'avec d'autres langages.

Les principales utilisations de Python par les développeurs sont :

- La programmation d'applications.
- La création de services web.
- La génération de code **[6].**

Figure 16: Logo de Python.

2.3 Environnement Logiciel

2.3.1 Flask

Flask est un Framework web, ou plutôt, **un micro-Framework**. Ce "micro" signifie simplement que Flask ne fait pas tout. Cela signifie qu'il faudra installer des extensions. Heureusement, celles-ci sont nombreuses, de qualité, et très bien intégrées. Flask est un ensemble de modules qui vont vous faciliter la programmation de sites web dynamiques.

La conception légère et modulaire de Flask la rend facilement adaptable aux besoins des développeurs. C'est pourquoi nous l'avons choisi pour développer notre serveur. Il comprend un certain nombre de fonctionnalités prêtes à l'emploi:

- Serveur de développement intégré et débogueur rapide
- Prise en charge intégrée pour les tests unitaires
- Répartition des demandes RESTful / Basé sur Unicode
- Modèles Jinja2 / Conformité WSGI 1.0
- Prise en charge des cookies sécurisés (sessions côté client)
- Gestion des requêtes http

La figure 17 présente le logo de Flask.

Figure 17: Logo de Flask.

2.3.2 Swagger

Swagger (figure 18) est un cadre logiciel open source soutenu par un large écosystème d'outils qui aide les développeurs à concevoir, construire, documenter et consommer des services Web RESTful. La plupart des utilisateurs identifient Swagger par l'outil Swagger UI, qui fournit un cadre d'affichage qui lit un document de spécification OpenAPI et génère un site Web de documentation interactif. L'ensemble d'outils Swagger inclut la prise en charge de la documentation automatisée, la génération de code et la génération de cas de test **[7].** Nous avons utilisé ''Swagger'' pour créer une interface pour notre serveur.

Figure 18 : Logo de Swagger.

2.3.3 Robot Framework

L'automatisation de tests est le processus par lequel un logiciel indépendant contrôle l'exécution de tests sur une application donnée. Nous automatisons ainsi des tâches répétitives mais nécessaires pour assurer la qualité de l'application. L'automatisation des tests est absolument essentielle pour la livraison continue et les tests continus. Au fur et à mesure que l'application est développée, les tests fonctionnels permettent de s'assurer que les fonctionnalités respectent les exigences du devis logiciel. Par ailleurs, ces tests peuvent être roulés à nouveau plus tard dans une suite de régression, lorsque du code est modifié et pourrait affecter certaines fonctionnalités. Dans notre projet nous avons utilisé un ''robot Framework'' python pour automatiser notre serveur.

La figure 19 présente le logo de ''robot Framework''.

Figure 19: Logo de Robot Framework.

2.3.4 MobaXterm

MobaXterm (figure 20) est une boîte à outils ultime pour l'informatique à distance. Il fournit de nombreuses fonctions adaptées aux programmeurs, webmasters, administrateurs informatiques et à peu près tous les utilisateurs qui ont besoin de gérer leurs travaux à distance de manière plus simple. MobaXterm fournit tous les outils réseau distants importants (SSH, RDP, X11, SFTP, FTP, Telnet, Rlogin, etc.) sur le bureau Windows, dans un seul fichier exe portable qui fonctionne hors de la boîte **[8].**

Figure 20 : Logo de MobaXterm.

2.3.5 StarUML

StarUML (figure 21)est un logiciel de modélisation UML, qui a été cédé comme open source par son éditeur, à la fin de son exploitation commerciale (qui visiblement continue), sous une licence modifiée de GNU GPL. Aujourd'hui la version StarUML V3 n'existe qu'en licence propriétaire. Nous avons utilisé ce logiciel pour réaliser les diagrammes de conception de notre serveur.

Figure 21 : Logo de StarUML.

2.3.6 GitLab

C'est un logiciel utilisé dans la réalisation de projet en équipe pour la gestion de version, il permet de suivre l'évolution d'un code et de conserver les modifications effectuées sur le code source, il est aussi capable d'annuler une modification en cas de problème.

Figure 22: Logo de GitLab.

2.4 Protocoles et format de données

2.4.1 HTTP

Protocole qui permet au client de récupérer des documents du serveur. Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc.) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP,

etc.). Ce protocole permet également de soumissionner les formulaires. La plupart des applications Front end communiquent avec les services back end via le protocole HTTP.

2.4.2 SSH

''Secure Shell'' est un protocole de réseau crypté pour initier des sessions Shell textuelles sur des machines distantes de manière sécurisée. Cela permet à un utilisateur d'exécuter des commandes sur l'invite de commande d'une machine sans qu'ils soient physiquement présents à proximité de la machine. SSH a été créé en 1995 pour le principal but est de permettre la prise de contrôle à distance d'une machine à travers une interface en lignes de commande. Dans notre projet nous avons utilisé le protocole **SSH** pour accéder à notre machine hôte et à notre RPI en mode console**.**

2.4.3 JSON

JSON est un format d'encodage de données structurées, très rapide à décoder que le XML. Il a un format texte unique indépendant de tout langage. Un document JSON ne comprend que deux éléments structurels :

- Des pairs noms / valeur appelés objet.
- Des listes ordonnées de valeurs.

Nous avons utilisé JSON dans notre serveur car il est simple à mettre en œuvre Parmi ses avantages nous citons aussi :

- Claire et facile à apprendre.
- JSON utilise peu de mot c'est-à-dire non verbeux au contraire de XML.

Conclusion

Dans ce chapitre nous avons étudié tout ce qu'est nécessaire à la réalisation de notre projet à savoir les outils matériels comme par exemple les composants et les cartes électroniques, et les outils logiciels comme les logiciels et les Frameworks.

Dans le chapitre suivant nous allons aborder l'étude conceptuelle de notre projet, tout en mentionnant tous les scénarios possibles, les acteurs et les diagrammes.

Chapitre 3 :
Analyse et Conception du Système proposé

Introduction

Dans le cycle de vie de notre projet, la conception représente une phase primordiale et déterminante pour produire une application de haute qualité. Dans ce troisième chapitre, nous nous intéressons en premier lieu à l'analyse des besoins pour avoir une compréhension profonde des besoins et des exigences qui permettent la conception de la solution. Puis nous allons clarifier la vue globale, en décrivant l'architecture générale que nous allons suivre dans la partie réalisation de notre projet. Et en fin, nous allons détailler notre choix conceptuel à travers deux types de diagrammes.

1 Spécification des besoins

Le but principal est de concevoir et de développer un serveur python qui doit offrir des APIs REST qui assurent le multiplexage des cartes à puce connectées vers des terminaux de paiement, ce serveur devra être intégré dans un environnement de test automatique pour lancer l'exécution de suites de tests avec le hub des cartes. D'où la possibilité de lancer plusieurs testes automatiques en même temps sur plusieurs cartes à puce sera faisable.

1.1 Identification des acteurs

L'identification des acteurs permet en premier lieu de délimiter le système et en second lieu de comprendre le rôle de chaque acteur.

Dans notre projet nous avons identifié 2 acteurs humains et un acteur système.

- Le client : qui va faire le multiplexage des cartes, et le test automatique
- L'administrateur : qui va gérer le serveur web
- Hardware: qui va communiquer avec le serveur à travers l'envoi des commandes via une liaison série.

1.2 Besoins fonctionnels

Les besoins fonctionnels décrivent les fonctionnalités du système. Ce sont les besoins spécifiant son comportement. Dans notre cas le serveur doit permettre de :

- Offrir une documentation Swagger des requêtes HTTP.
- Gérer la communication au serveur python à travers des requêtes HTTP.

- Gérer la communication entre le serveur et le hardware à travers l'envoi des commandes via une liaison série.
- Gérer la liste des terminaux, des cartes à puce, des canaux reliant les cartes et les terminaux.
- Faire le multiplexage entre les cartes et les terminaux.
- Gérer l'authentification en utilisant la méthode « Token Authentification».

1.3 Besoins non fonctionnels

Une fois les besoins fonctionnels bien définis, des besoins non fonctionnels doivent être pris en compte tout au long du processus de développement du serveur. Parmi les exigences implicites auxquelles notre projet doit répondre nous pouvons citer :

- **Sécurité :** Les autorisations et l'accès aux données doivent être effectués selon l'identification de mot de passe de l'autorisation.
- **Performance :** la solution doit être rapide et fiable.
- **Ergonomie et simplicité :** Les interfaces de l'application doivent être claires, simples, compréhensibles et faciles à utiliser pour que l'utilisateur puisse atteindre son objectif d'une manière efficace avec un confort de navigation.
- **Robustesse :** Le système doit gérer les erreurs qui peuvent survenir au cours de l'exécution en affichant un message indiquant l'erreur et sa raison. En plus, le système doit prendre en compte les fausses manipulations de la part des utilisateurs et les bloquer pour assurer le bon fonctionnement de serveur.
- **Maintenabilité :** Le code de l'application doit être bien lisible et compréhensible pour pouvoir le maintenir facilement et rapidement.

2 Méthodologie de conception

Pour faciliter notre tâche nous avons recours langage de Modélisation unifié (UML) c'est une notation qui permet de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existantes auparavant, et il est devenu une référence en terme de modélisation objet, à un tel point que sa connaissance devienne indispensable pour un développeur.

2.1 Diagramme de cas d'utilisation

Un diagramme de cas d'utilisation représente les fonctions du système du point de vue de l'utilisateur. Il décrit la réponse du système à un événement provenant d'un acteur. C'est une unité cohérente car il réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin. Un cas d'utilisation modélise donc un service rendu par le système c'est-à-dire à quoi sert le système, sans imposer le mode de réalisation de ce service.

2.1.1 Digramme de cas d'utilisation globale

Le diagramme de cas d'utilisation globale comporte trois acteurs qui sont le client, l'administrateur de serveur, et l'acteur système qui est le hardware. Dans ce diagramme nous trouvons trois cas d'utilisations principales:

- ✓ Faire un test automatique
- ✓ Faire le multiplexage
- ✓ Gérer le serveur

La figure 23 représente le diagramme de cas d'utilisation global de notre serveur.

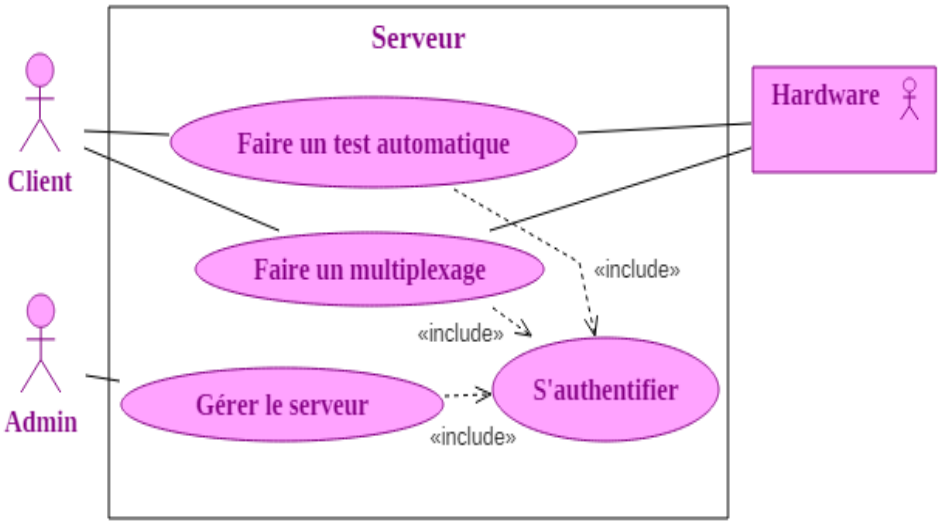


Figure 23 : Diagramme de cas d'utilisation globale.

2.1.2 Diagrammes de cas d'utilisation détaillés

- **Cas d'utilisation « Faire un multiplexage»**

Avant tout l'utilisateur doit s'authentifier pour qu'il puisse accéder au serveur. Après l'ajout automatique de la carte et de terminal dans le serveur par l'acteur système, l'utilisateur peut

faire un multiplexage autrement dit la création d'un canal entre la carte et le terminal. Puis il peut modifier ou bien supprimer une carte, terminal ou bien le canal qu'il a créé.

La figure 24 représente le diagramme de cas d'utilisation détaillé de l'action « Faire le multiplexage »

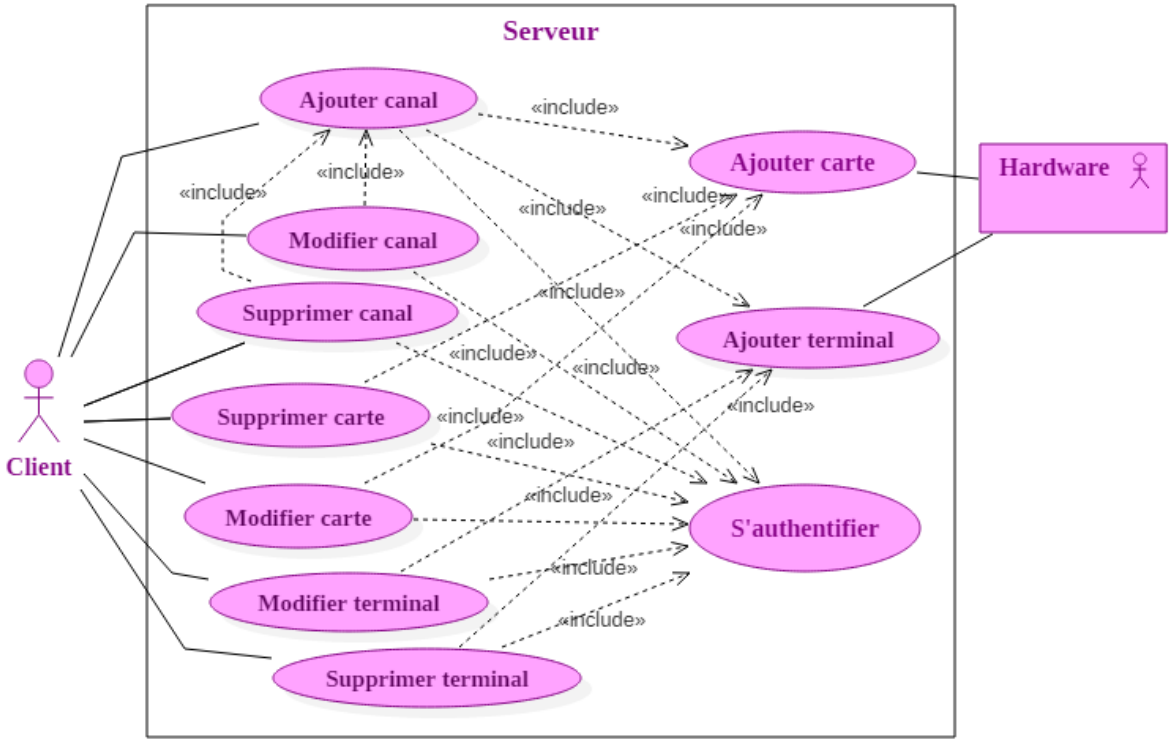


Figure 24 : Diagramme de cas d'utilisation « Faire un multiplexage ».

- **Cas d'utilisation « Faire un test automatique »**

Ce cas d'utilisation est identique à celui précédent mais ce cas ça sera fait d'une manière automatique, la seule action de l'utilisateur est le lancement de test.

- **Cas d'utilisation « Gérer le serveur »**

L'administrateur de serveur est le seul acteur qui peut modifier le fonctionnement de serveur, faire le mise à jours, ajouter d'autres options, modifier les nombres maximales de cartes et des terminaux que nous ne devons pas les dépasser, etc.

2.2 Diagramme des séquences

Les diagrammes de séquence sont utilisés pour décrire les cas d'utilisation. Ils permettent de donner une description du fonctionnement du système en fonction du temps. Pour notre

modélisation nous allons essayer de montrer tous les scénarios possibles que notre système peut les faire, à travers des diagrammes de séquences.

2.2.1 Authentification de serveur

La figure 25 présente le diagramme de séquence de l'authentification de serveur, nous avons un mot de passe **''Token''** que l'utilisateur de serveur doit le connaître pour qu'il puisse l'utiliser.

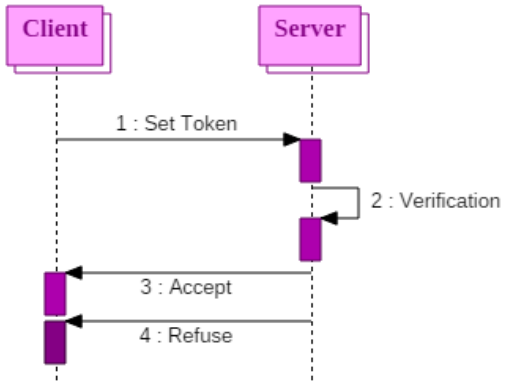


Figure 25 : Diagramme de séquence de l'authentification de serveur.

2.2.2 Login et Insertion de la carte

Après l'ouverture de deux ports séries, la première requête que le port série de la carte FPGA va l'envoyer vers celui de la RPI est la commande de **''LOGIN''**. Le client de serveur ne peut rien faire avant l'envoi de cette commande, lorsqu'il envoie une requête, il va recevoir un message d'erreur.

Le port série de la carte de multiplexeur **''Serial_Board''** vérifie toujours s'il y a une carte a été insérée ou pas, lorsque nous branchons une carte dans notre système, il va envoyer une commande au port série de la RPI **''SerialHandler''** pour la créer automatiquement dans le serveur. Maintenant le client peut accéder à la liste de cartes et modifier leurs caractéristiques. Nous avons fixé le nombre maximal des cartes à puce que nous ne devons pas le dépasser. Si nous dépassons ce nombre nous allons obtenu un message d'erreur, notre serveur accepte seulement le nombre qui est déjà fixé. Lorsque nous débranchons une carte elle sera supprimée automatiquement du serveur.

La figure 26 illustre un diagramme de séquence montre le scénario qui peut être réalisé avant le login de la carte et les scénarios qui peuvent être réalisé avec la carte.

2.2.3 Insertion de terminal

Identiquement à l'insertion de la carte, une commande de création d'un terminal, va être envoyée vers **''SerialHandler''** lors de branchement de terminale dans le système, pour le créer automatiquement dans le serveur. Maintenant le client peut accéder à la liste de terminaux, et modifier leurs caractéristiques. Lorsque nous débranchons un terminal il sera supprimé automatiquement du serveur. Voir figure 27.

2.2.4 Création du canal

Maintenant le client peut créer un canal entre la carte et le terminal qui sont déjà créés dans le serveur. Le canal est créé mais n'est pas activé, une commande d'activation va être envoyée par le **''Serial_Board''** pour l'activer. Après l'activation de canal l'échange des données entre la carte et le terminal va être commencé. Lorsque le client essaie de créer un canal entre une carte et un terminal qui ne sont pas encore crées dans le serveur, il va recevoir un message d'erreur. Ces scénarios sont présentés dans la figure 28.

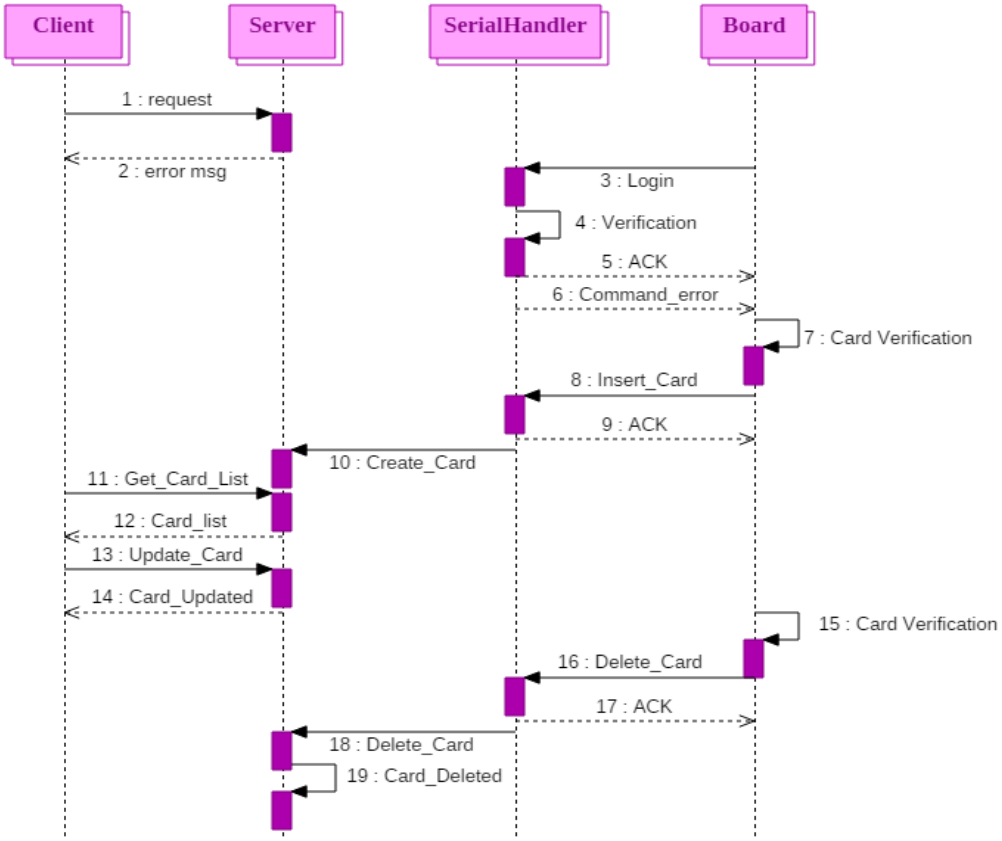


Figure 26 : Diagramme de séquence de login et de l'insertion de la carte.

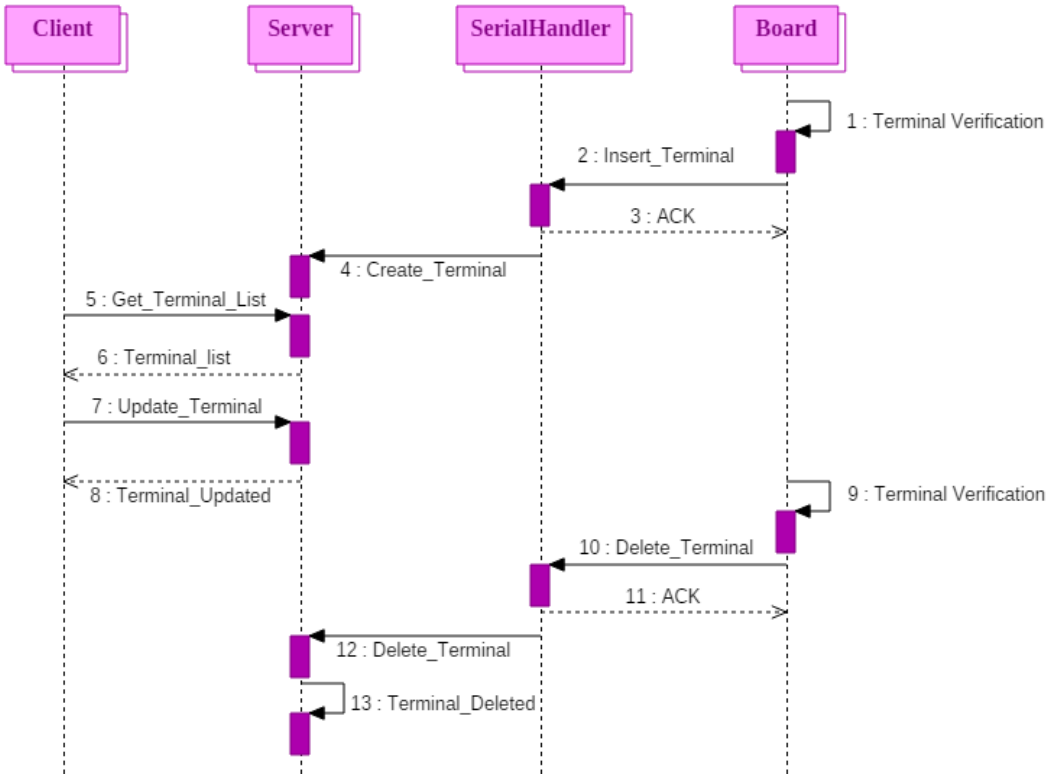


Figure 27 : Diagramme de séquence de l'insertion de terminal.

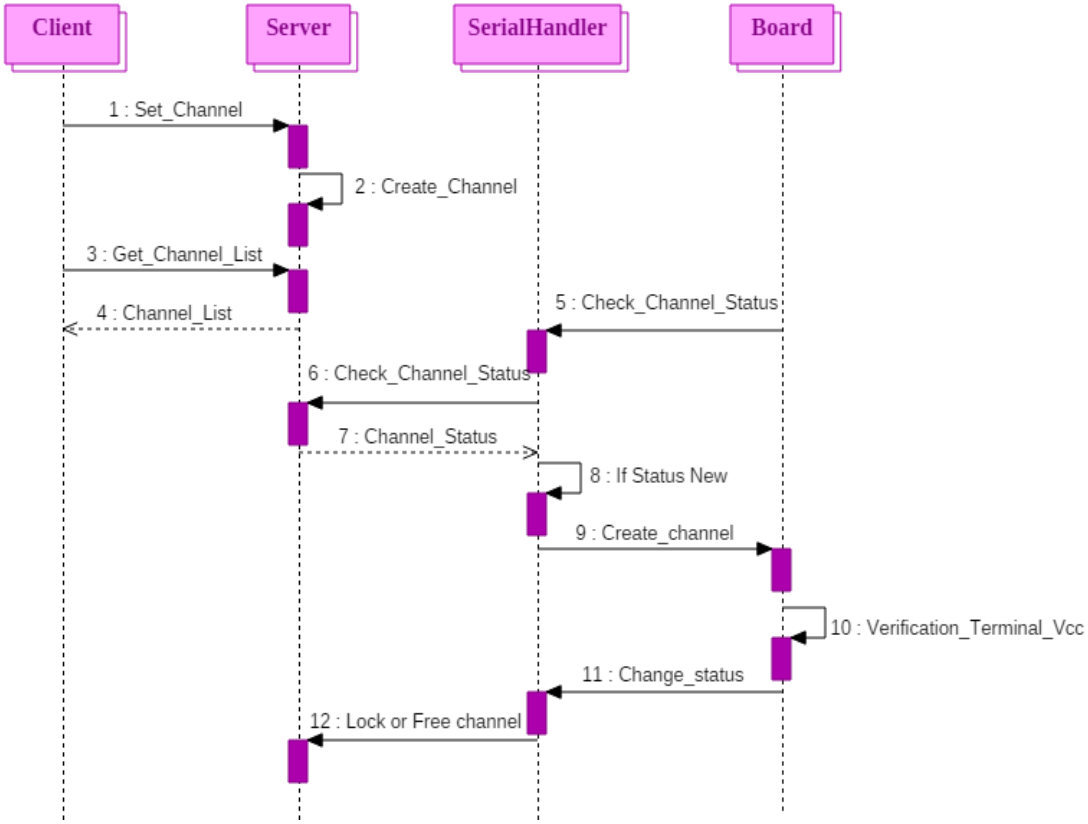


Figure 28 : Diagramme de séquence de création d'un canal.

2.2.5 Suppression et Modification d'un canal

Le client peut aussi modifier ou bien supprimer un canal tant qu'il est désactivé. Et ça évite l'interruption de communication et la perte d'information. Le **''Serial_Board''** envoi toujours des commandes de vérification s'il y a un changement au niveau de serveur (suppression ou bien modification) sera fait au niveau de **''Board''**.

La figure 29 illustre le diagramme de séquence de suppression et modification d'un canal.

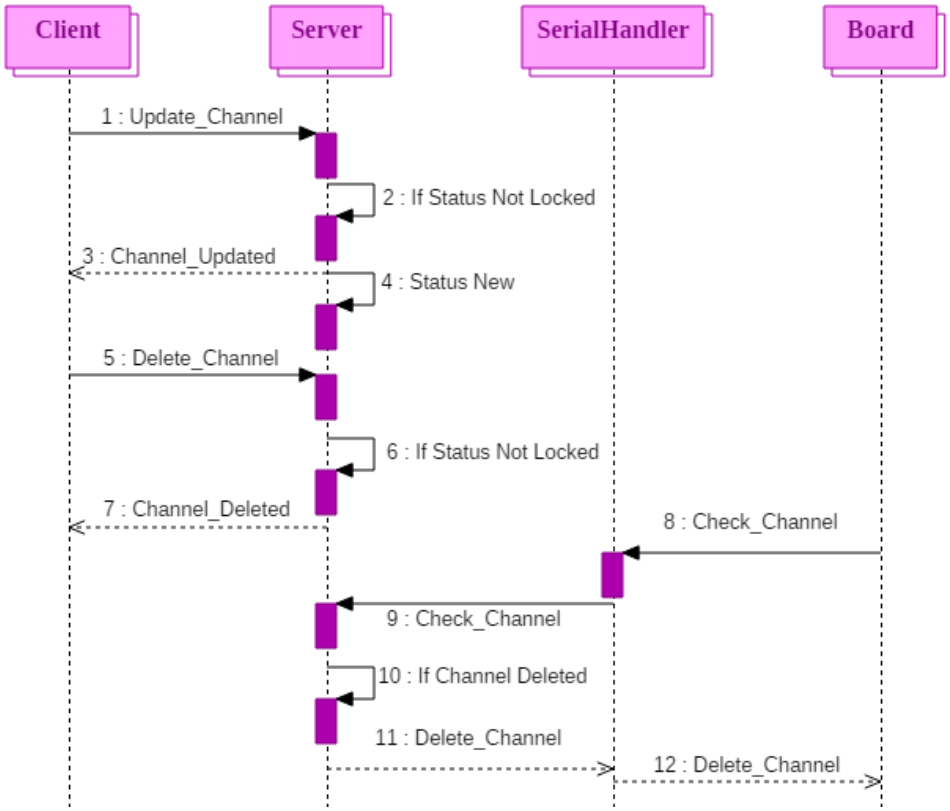


Figure 29 : Diagramme de séquence de suppression et modification d'un canal.

Conclusion

Au cours de cette partie nous avons tout d'abord spécifié les besoins de notre projet et ses différents acteurs, ainsi que les besoins fonctionnels et non fonctionnels du système. Nous avons consacré la dernière partie pour la conception détaillée de notre solution tout en étudiant les diagrammes de cas d'utilisation et les diagrammes de séquences.

Dans le chapitre suivant nous allons aborder la dernière partie qui représente la partie réalisation de notre serveur, en se basant sur les mécanismes et les solutions déterminés dans la phase de conception.

Chapitre 4 :
Réalisation de Projet

Introduction

Ce dernier chapitre est consacré à la réalisation et l'implémentation de notre système, Comme nous avons déjà dit dans le premier chapitre notre projet comprend essentiellement deux grandes parties le chargement de firmware et la création de serveur web que nous allons l'intégrer dans un ''robot Framework''. Tout d'abord nous allons présenter la partie bas niveau toute en expliquant les étapes suivies afin d'assurer le chargement de firmware. Puis nous allons expliquer la partie d'application à partir des imprimes écrans de notre serveur web et notre ''robot Framework'' que nous avons développé pour automatiser le serveur.

1 Schéma global du système

Le schéma global de notre travail présenté par la figure 30 montre successivement les étapes principales que nous avons réalisées dans notre projet.

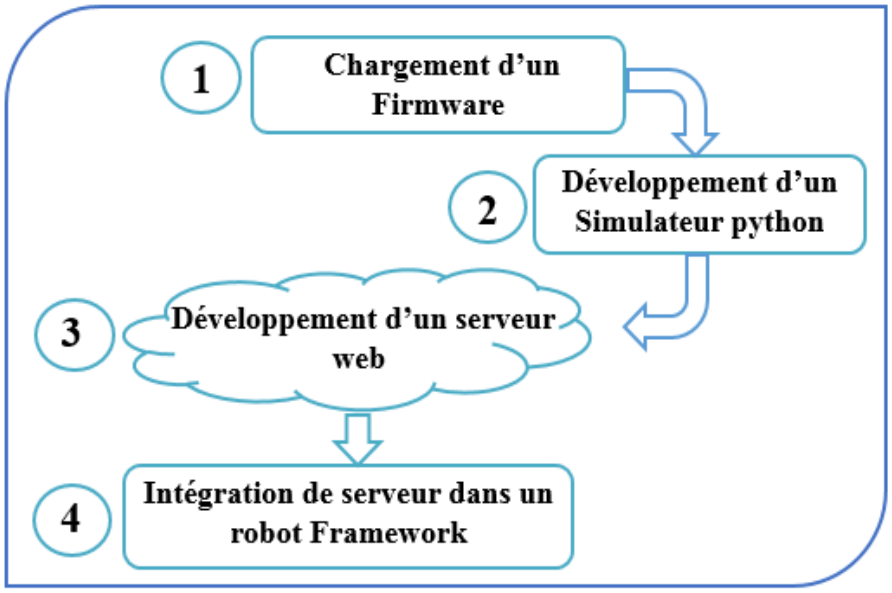


Figure 30: Schéma global de notre travail.

2 Chargement d'un Firmware

La première partie de notre travail sur l'espace noyau est le chargement de firmware dans la carte FPGA à partir de la RPI. Dans cette paragraphe, nous allons définir qu'est-ce qu'un firmware, à quoi sert exactement et pourquoi nous avons besoin de le charger dans notre FPGA.

Certains périphériques nécessitent un firmware c'est-à-dire (un micro logiciel pour les périphériques évolués) pour fonctionner correctement. Ce firmware peut être stocké de manière permanente dans une mémoire non volatile, ou devoir être rechargé à chaque initialisation du périphérique (pour plus des détails voir l'annexe). Dans ce cas, cette opération de chargement que nous voulons de le faire doit être faite par le pilote de périphérique qu'est une fonctionnalité parmi les tâches principales du noyau et nous l'avons déjà défini dans le deuxième chapitre.

La figure 31 représente les différentes étapes que nous avons réalisé afin d'assurer le chargement d'un firmware.

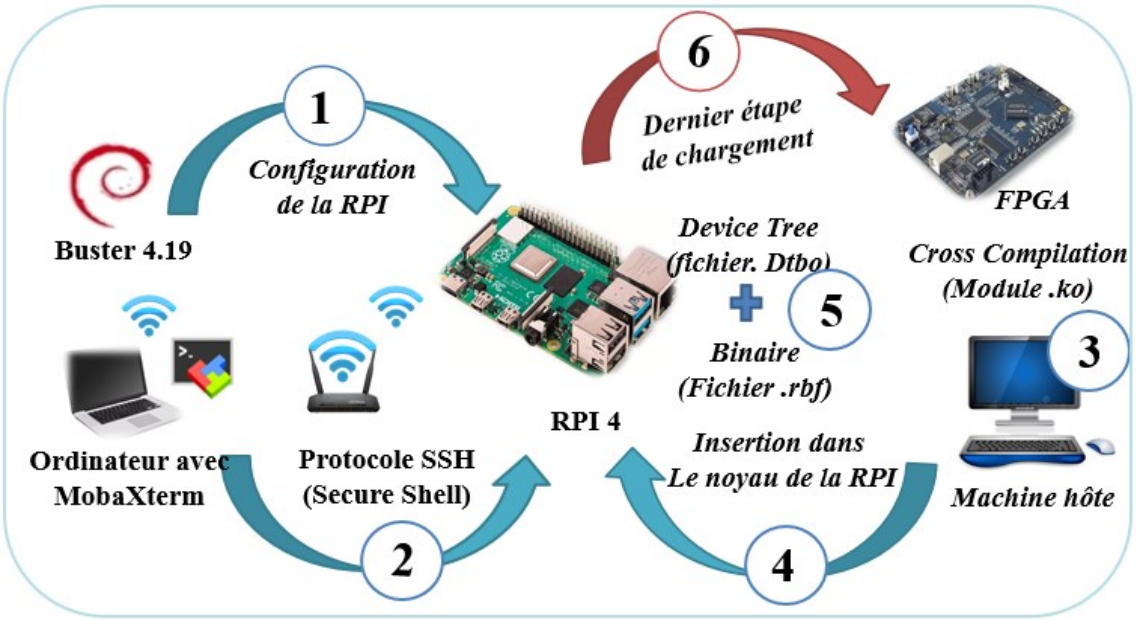


Figure 31 : Les étapes de chargement d'un Firmware.

2.1 Configuration du système à base du Raspberry Pi

Notre travail a été réalisé sur la Raspberry Pi qu'est un système embarqué nécessite une configuration avant son utilisation, et par la suite avant de commencer notre projet, il faut tout d'abord faire la configuration de la RPI c'est-à-dire installer le système d'exploitation choisi comme une première étape, et la connexion à la Raspberry Pi sur un poste distant comme deuxième étape.

Après la configuration, notre Raspberry Pi devient prête à utiliser. L'étape suivante est la compilation croisée

2.2 Compilation croisée

Nous avons utilisé la RPI 4 C'est le dernier module de Raspberry Pi, et la seule version de système d'exploitation Raspbian que nous pouvons l'installer c'est la dernière version « Buster 4.19». Pour assurer la compatibilité des noyaux, il faut faire la compilation croisée avec la même version de noyau source.

La figure 32 présente la compilation croisée qui est le processus de compilation de code pour un système informatique (souvent appelé cible) sur un système différent, appelé l'hôte. C'est une technique très utile, par exemple lorsque le système cible est trop petit pour héberger le compilateur et tous les fichiers pertinents.

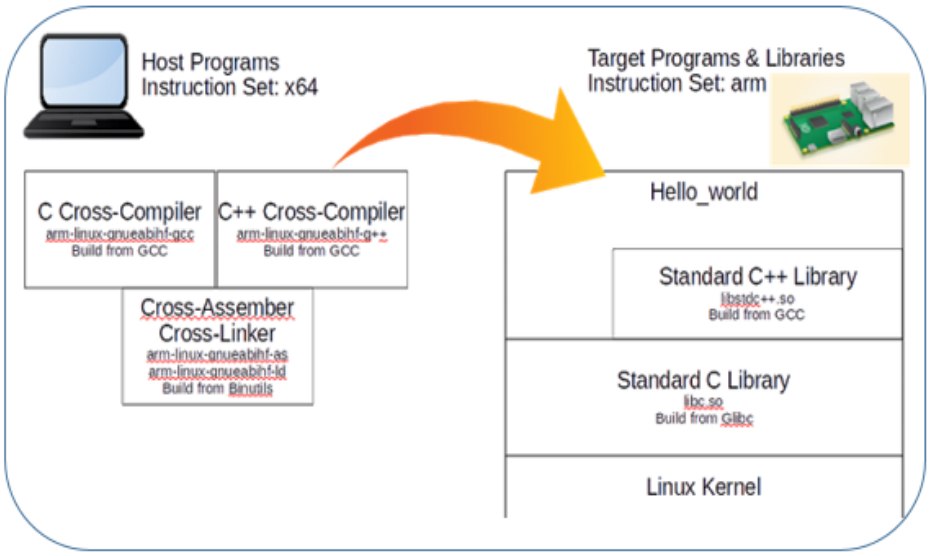


Figure 32 : Compilation croisée.

Dans notre cas nous avons utilisé la Raspberry Pi 4 comme une machine cible et un ordinateur de bureau comme une machine hôte dont leurs caractéristiques sont présentés précédemment dans le deuxième chapitre. Nous avons préparé notre environnement de travail, nous avons fait la compilation croisée de la version de noyau que nous avons déjà choisie tout en faisant la configuration nécessaire de noyau.

La figure 33 illustre la configuration des modules que nous avons besoin par la suite pour qu'ils puissent être générés après la compilation croisée.

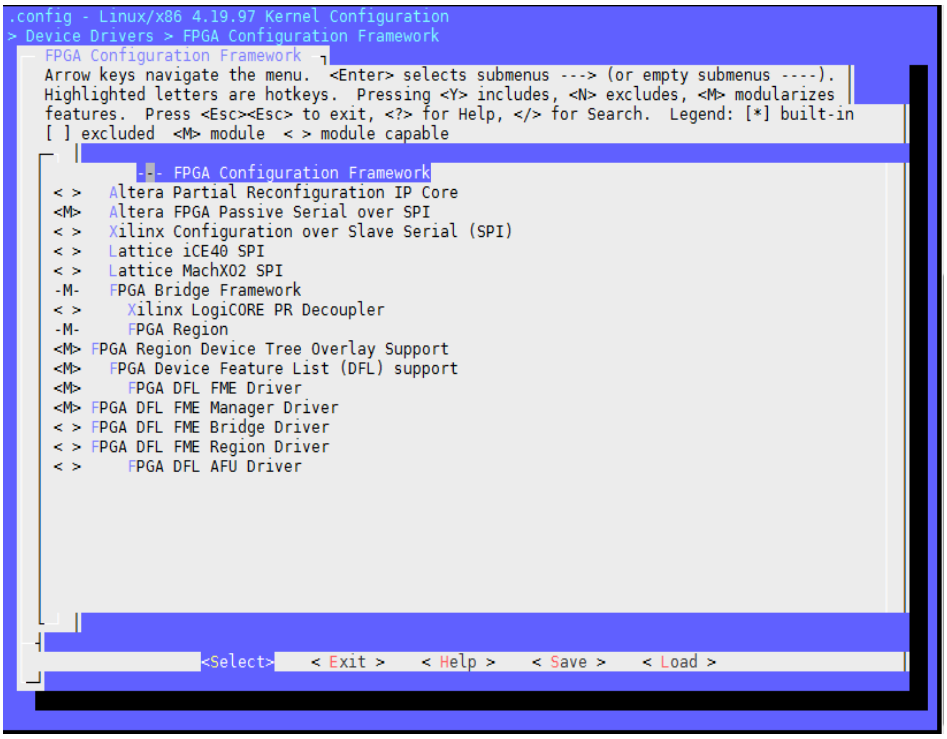


Figure 33 : Configuration des modules.

2.3 Insertion des modules dans le noyau de LA RPI

Après la compilation croisée nos modules ont été générés puis nous les avons envoyés et les avons insérés dans le noyau de la RPI, comme il est présenté par La figure 34.

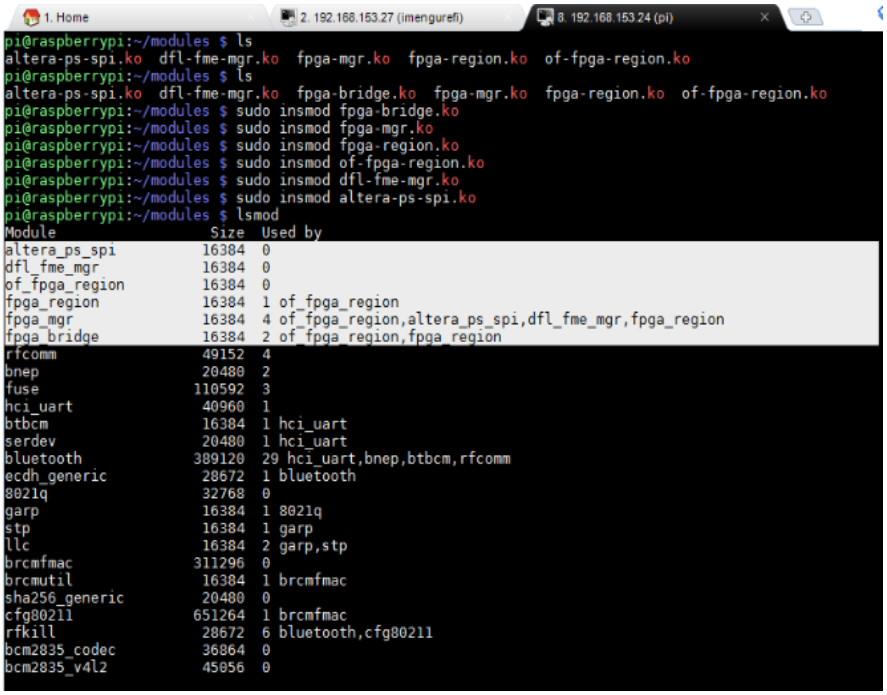


Figure 34 : Les modules insérés dans le noyau de la RPI.

2.4 Préparation de device tree

Au niveau de la RPI, nous avons préparé le device tree qui est un outil très utile et très efficace qui va nous permettre de charger le firmware qu'est un fichier binaire fourni par TELNET dans la FPGA.

2.5 Problème rencontré et Solution

Nous avons réussi à faire la compilation croisée et l'envoi de modules vers la RPI et leur insertion dans le noyau de système, et nous avons préparé notre device tree mais nous n'avons pas réussi à charger le firmware, nous avons trouvé quelques difficultés vu la complexité de device tree que nécessite d'avoir une connaissance approfondie des modules du noyau, et ça prendra énormément de temps et d'expérience. Donc pour résoudre ce problème et d'autres problèmes hardware et surtout pour rester dans le contexte de notre projet nous avons décidé de développer un script python (simulateur) qui remplace le matériel hardware qui est essentiellement la carte d'extension HAT qui rassemble (FPGA, le multiplexeur, les cartes à puce et les terminaux de paiement). Et comme ça nous pouvons passer au cœur de notre projet qui est la création d'un serveur python.

3 Création d'un Serveur Web RPI

Comme présenté auparavant, le serveur installé sur un RPI 4 connecté à une carte d'extension et exécutant un algorithme de commande assure deux types de communication, ce serveur permet d'échanger des informations sur le Web lorsque cela est demandé par un navigateur Web tout en utilisant le protocole HTTP. Il est aussi capable de communiquer avec le hardware à travers une liaison série en utilisant un protocole de communication bien spécialisé que nous avons créé précisément pour assurer l'échange entre les deux.

La figure 35 montre les deux types de communication de notre serveur.

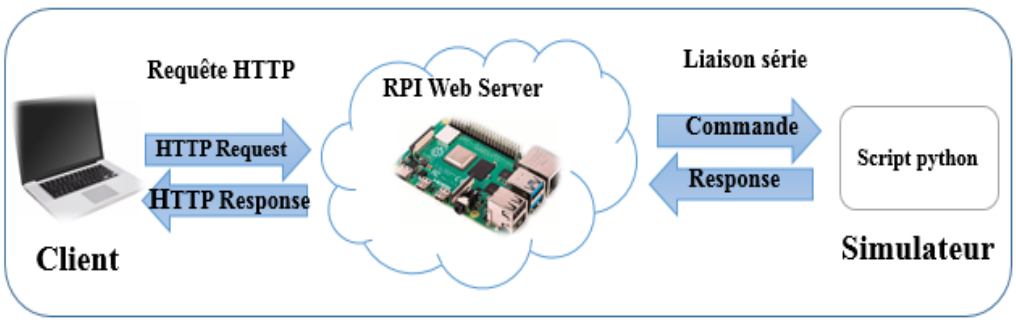


Figure 35 : Serveur Web RPI.

3.1 Protocole de communication de la liaison série

Nous avons défini notre propre protocole de communication pour bien assurer la communication entre le serveur et le hardware. Tous les requêtes qui seront envoyées vers les ports séries sont composées de trois parties ''PacketLen'' + ''Command''+ ''Payload'', leurs significations et leurs longueurs sont définies dans le tableau 1.

Tableau 1 : Protocole de communication de la liaison série.

Champ	Définition	Longueur (byte)
''PacketLen''	La longueur du paquet	2
''Command''	Mot de commande	1
''Payload''	Données efficaces	1 / 2

3.1.1 Requêtes possibles

Le tableau 2 illustre tous les commandes possibles sous forme hexadécimale qui peuvent être échangées entre La RPI et la FPGA et leurs directions d'envoi.

Tableau 2 : Les requêtes possibles.

Cas d'utilisation	Commandes	direction
''LOGIN''	0x60	''Board'' → ''SerialHandler''
''CREATE_CARD''	0x61	''Board → ''SerialHandler''
''INSERT_TERMINAL''	0x62	''Board''→ ''SerialHandler''
''CREATE_TERMINAL''	0x63	''Board'' → ''SerialHandler''
''CHECK_CHANNEL''	0x64	''Board'' → ''SerialHandler''
''CREATE_CHANNEL''	0x65	''SerialHandler'' → ''Board''
''CHANGE_STATUS''	0x66	''Board'' → ''SerialHandler''
''LOCK_CHANNEL''	0x67	''Board'' → ''SerialHandler''
''FREE_CHANNEL''	0x68	''Board'' → ''SerialHandler''
''DELETE_CARD''	0x69	''SerialHandler'' → ''Board''
''DELETE_TERMINAL''	0x70	''Board''→ ''SerialHandler''
''COMMAND_ERROR''	0x71	''SerialHandler'' → ''Board''
''DELETE_CHANNNEL''	0x72	''SerialHandler'' → ''Board''
''UPDATE_CHANNEL''	0x73	''SerialHandler'' → ''Board''

Chaque requête a une longueur bien déterminé qui peut être deux ou bien trois octets. Cette longueur est spécifiée au début de la requête après le **''0x''** ensuite la commande qui est toujours sur un seul octet et en fin le **''Payload''** qui se diffère d'une requête à une autre.

La figure 36 montre l'exemple des requêtes que nous pouvons l'envoyer pour la création des cartes avec la commande **''CREATE_CARD''**, nous utilisons des **''Payload''** différents **''00''** pour la création de la carte qui est insérée dans le **''slot 0'', ''01''** pour la carte qui est insérée dans le **''slot 1''** ainsi de suite.

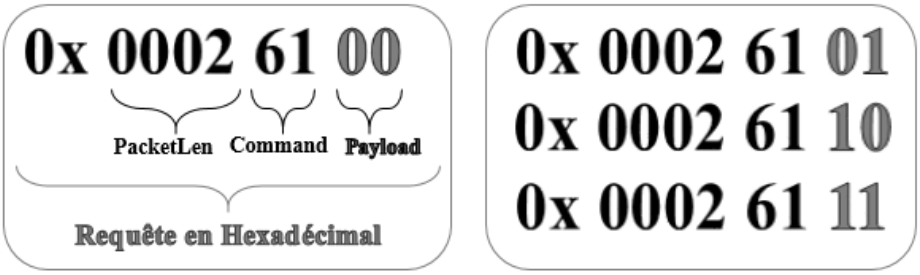


Figure 36 : Les requêtes de la création des cartes.

3.2 Documentation Swagger

Nous avons réalisé une interface web sous forme d'une documentation interactive en utilisant ''Swagger'' défini dans le deuxième chapitre. Pour mieux comprendre le fonctionnement de l'interface, nous allons donner un exemple d'organigramme d'un scénario simple (figure 37) puis nous allons donner quelques captures d'écran montrent les résultats de tous les scénarios possibles.

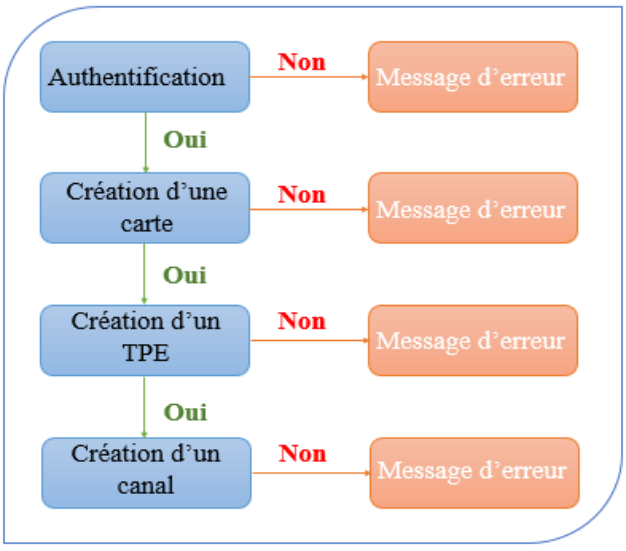


Figure 37 : Organigramme d'un scénario simple.

3.2.1 Page d'accueil

La figure 38 présente la première page qui s'ouvre lorsque nous lançons le serveur. Cette page montre les points de terminaison ‘’endpoints‘’ qui sont regroupés comme suit:

- ‘’**Card**’’ pour la gestion des cartes.
- ‘’**Terminal**’’ pour la gestion des terminales.
- ‘’**Channel**’’ pour la gestion des canaux.

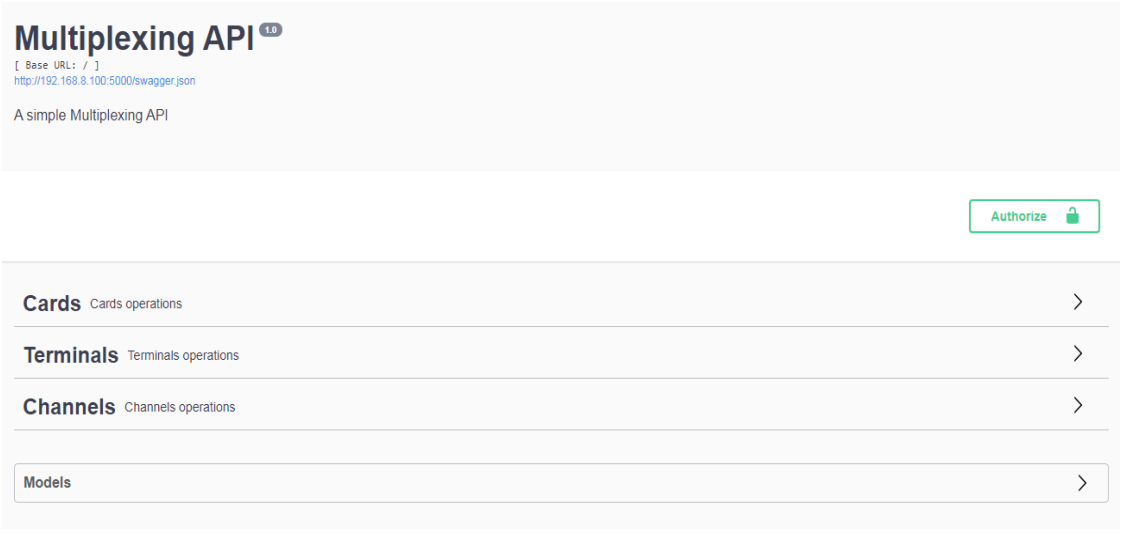


Figure 38 : Page d'accueil.

3.2.2 Autorisation de serveur

Avant de faire des demandes, nous devons autoriser notre session en cliquant sur le bouton autoriser **‘’Auhtorize’’** et en complétant les informations requises dans le mode d'autorisation illustré dans la figure 39.

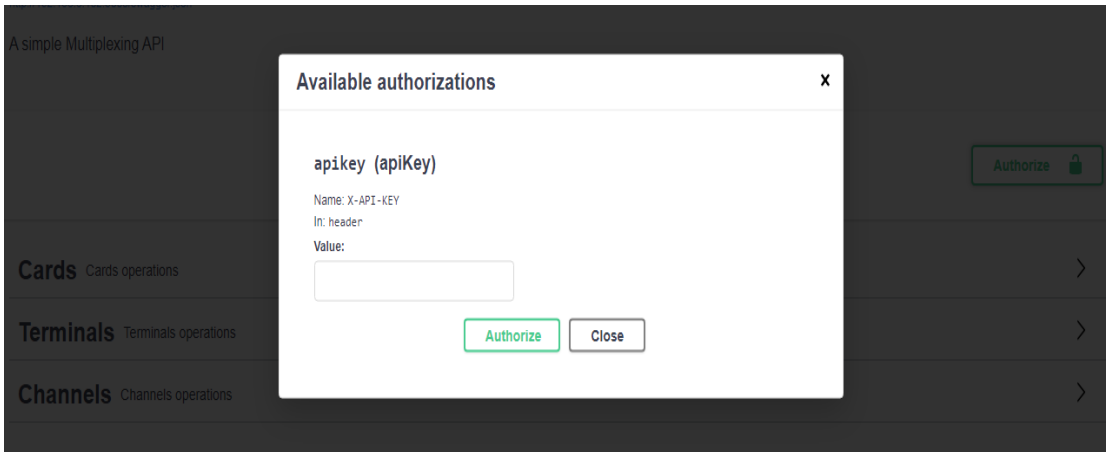


Figure 39 : Autorisation de serveur.

La vérification de l'authentification comprend l'assurance de remplissage du champ, c'est le cas de la figure 40, et la validation de correspondance de ''Token'' est le cas de la figure 41.

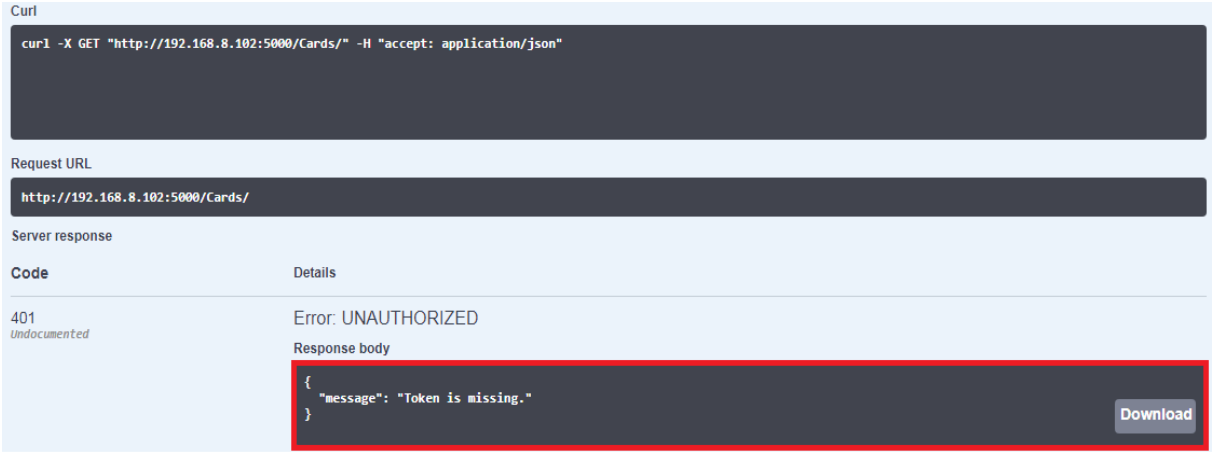


Figure 40 : Donnée d'authentification manquante.

Figure 41 : Donnée d'authentification erronée.

3.2.3 Points de terminaison

Dans cette version HTML de la documentation générée par ''Swagger'', nous retrouvons toutes les méthodes que nous avons définies, et qui sont cinq méthodes HTTP:

- **''POST''** pour ajouter un canal.
- **''GET''** pour afficher la liste des cartes, la liste des terminales ou la liste des canaux.
- **''PUT''** pour modifier les caractéristiques d'une carte, d'un terminal ou d'un canal.
- **''GET by id''** pour retourner une carte, un terminal ou un canal bien déterminé.
- **''DELETE''** pour supprimer un canal.

La figure 42 présente les cinq méthodes HTTP dédiées pour la gestion des canaux.

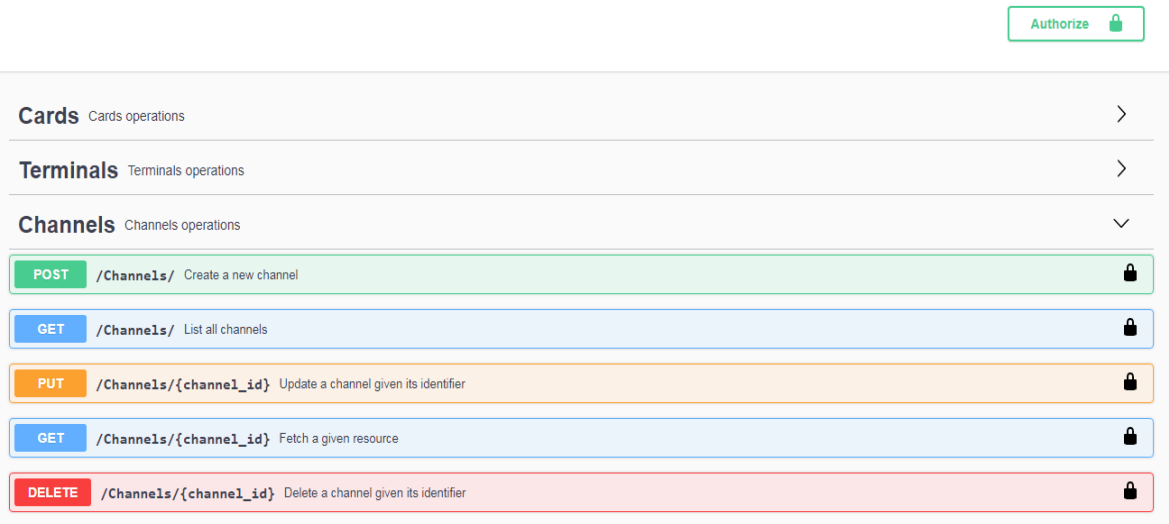


Figure 42 : Méthodes HTTP dédiées pour les canaux ''Channels''.

En cliquant sur une méthode, nous retrouvons les détails sur :

- Le type de données qu'elle accepte en entrée et qu'elle produit en retour,
- Un exemple d'une réponse typique,
- Tous les codes d'erreur qu'elle peut générer.

3.2.4 Avant la création de la carte

La figure 43 montre le message d'erreur qui va être affiché, lorsque le client essaie de retourner la liste des cartes avant la requête de login comme nous avons précisé dans le diagramme de séquence de login de la carte dans le chapitre de conception.

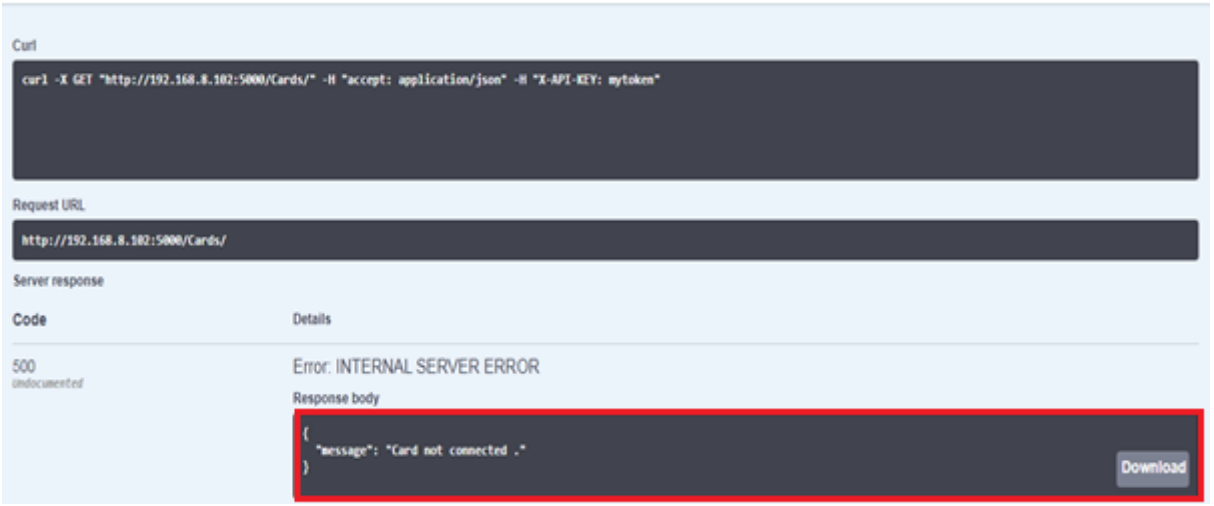


Figure 43 : Message d'erreur du login de la carte.

3.2.5 Après la création de la carte

Après login de la carte, le **''Serial_Board''** va envoyer des requêtes pour ajouter des cartes et des terminaux et maintenant le client peut faire les requêtes http et retourner la liste de cartes ou bien la liste des terminaux et il peut aussi modifier leurs caractéristiques.

La figure 44 illustre un exemple d'une liste des cartes et la figure 45 illustre un exemple d'une liste des terminaux.

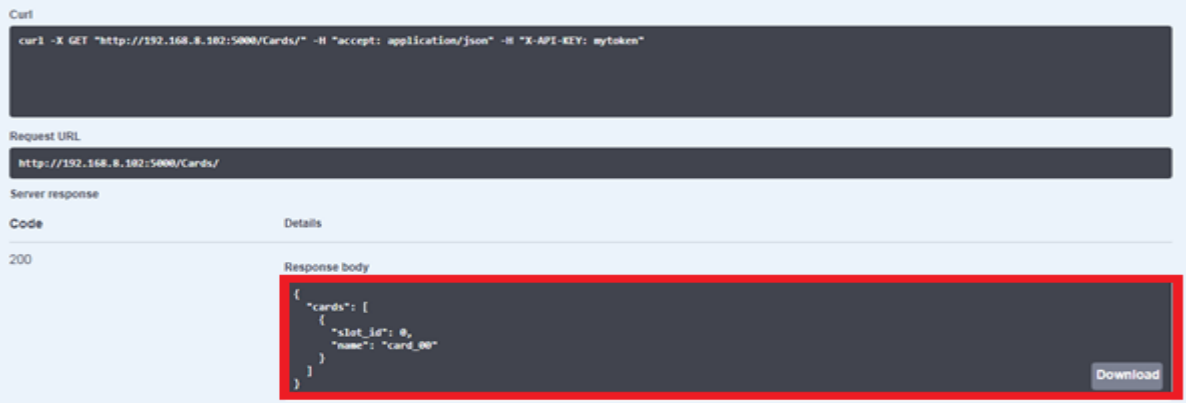


Figure 44 : Liste des cartes.

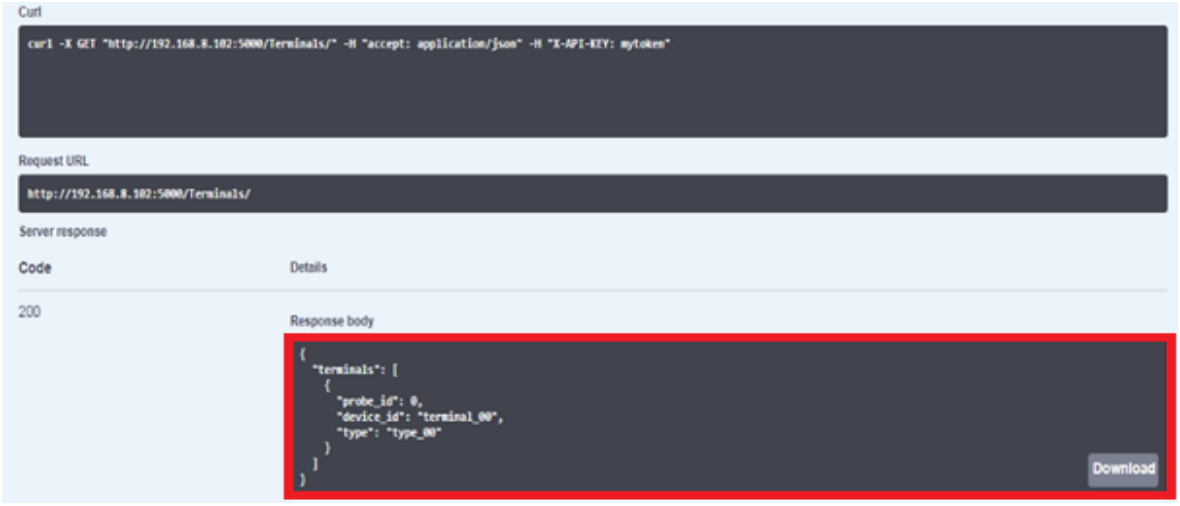


Figure 45 : Liste des terminaux.

3.2.6 Création d'un canal

Pour que le client puisse ajouter un canal, il faut tout d'abord ajouter au moins une carte et un terminal, lorsque le client veut créer un canal, il va recevoir un message d'erreur. Comme il est présenté dans la figure 46. Si le client essaie de créer un canal avec une carte ou bien un terminal qui n'est pas encore crée dans le serveur il va recevoir un autre message d'erreur comme il est illustré dans la figure 47.

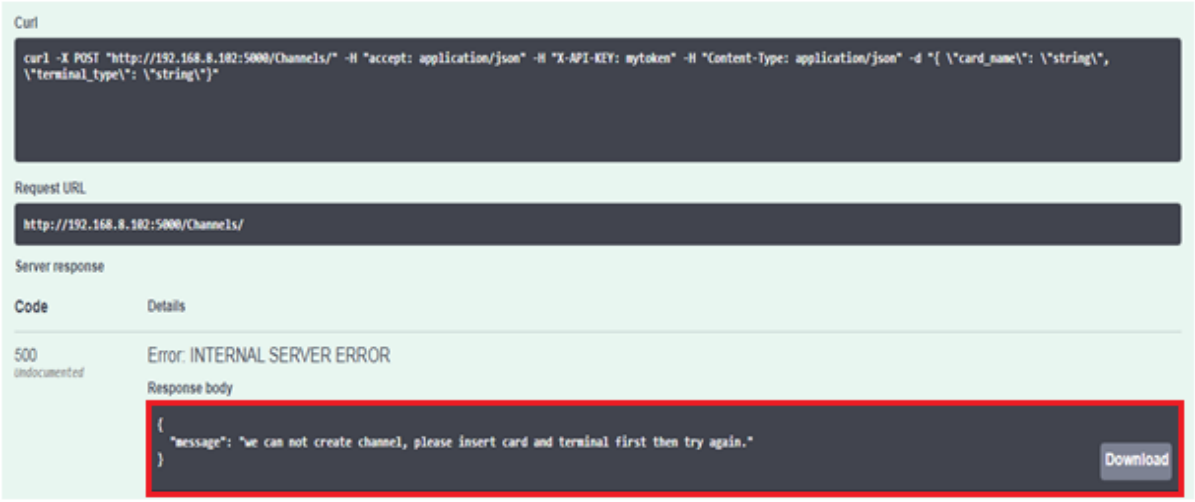


Figure 46 : Message d'erreur de la création du canal.

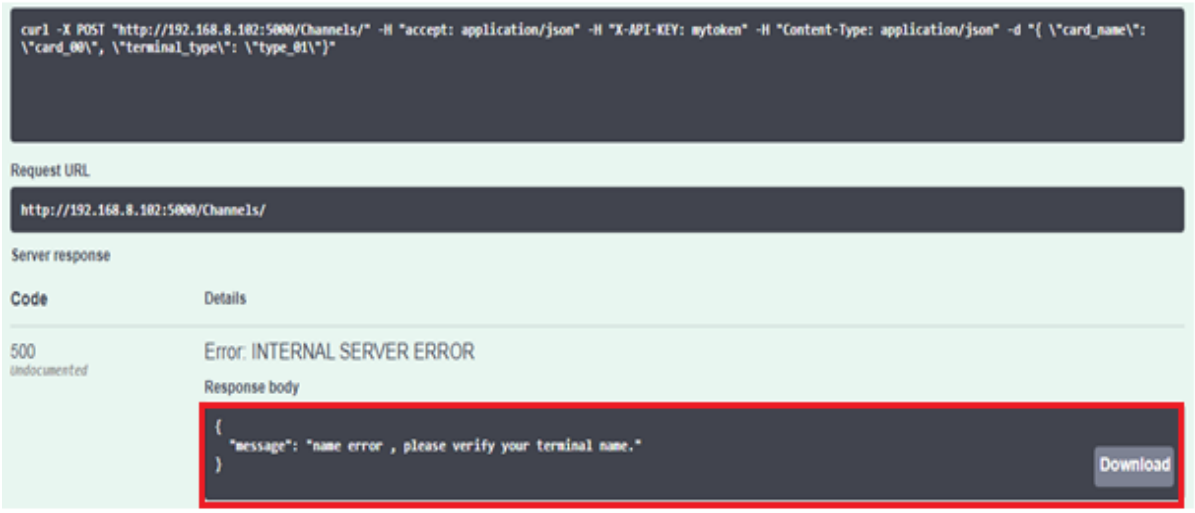


Figure 47 : Nom de terminal erroné.

La figure 48 illustre un exemple de la liste des canaux.

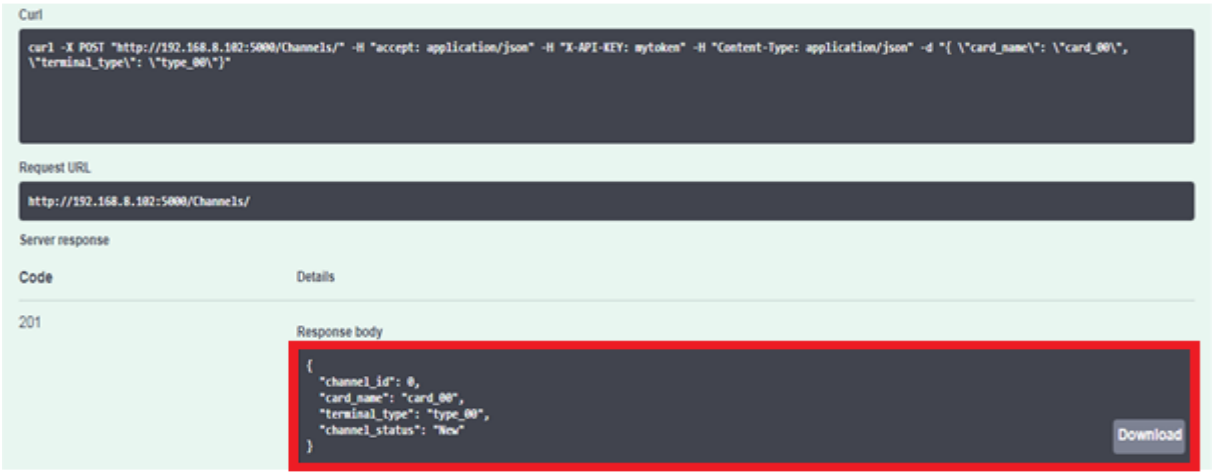


Figure 48 : Liste des canaux.

Nous avons fixé le nombre des cartes, terminales ainsi que des canaux, le client ne peut pas le dépasser.

La figure 49 montre le message d'erreur qui sera affiché si le client dépasse le nombre des canaux fixé dans le serveur.

Figure 49 : Message d'erreur.

Une commande va être envoyée par le **''Serial_Board''** vers le serveur pour changer l'état d'un canal. La figure 50 présente ce changement **''Free''** pour désactiver un canal ou bien **''Locked''** pour activé un canal**.**

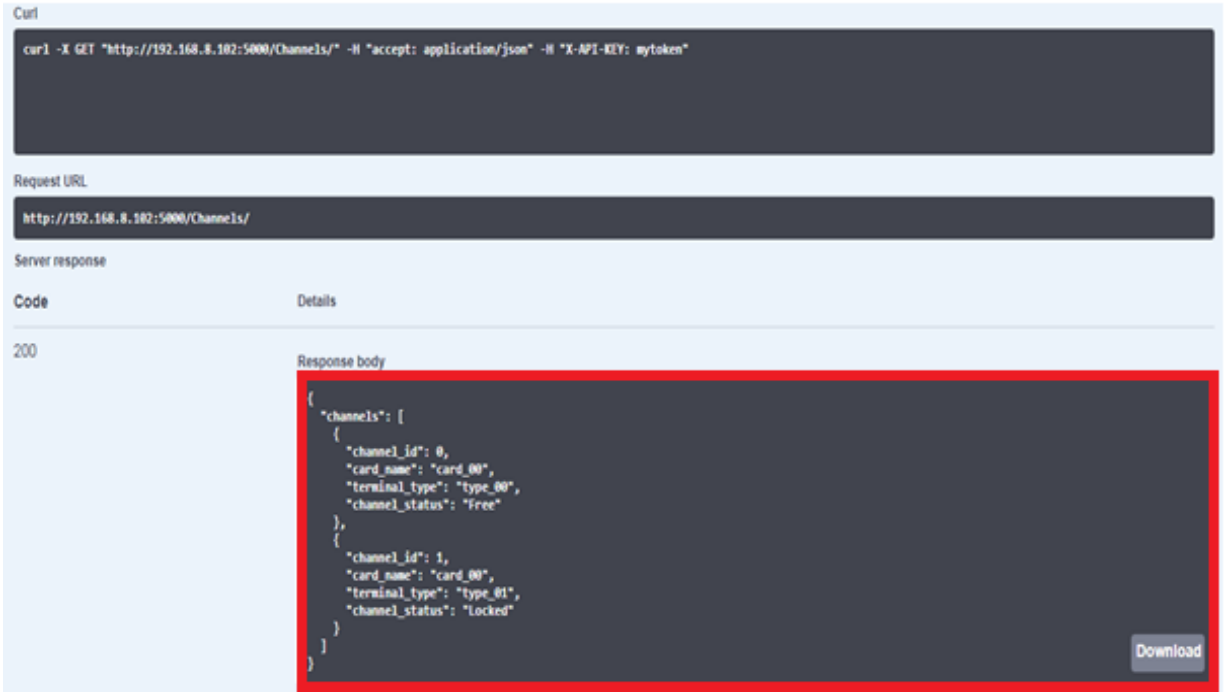


Figure 50 : Changement de l'état d'un canal.

Le client ne peut pas supprimer ou bien modifier un canal tant qu'il est activé, pour éviter l'interruption de la communication, de l'échange de données et la perte d'information. Sinon des messages d'erreur seront affichés, ces messages sont présentées dans la figure 51 et la figure 52.

Figure 51 : Message d'erreur de la suppression de canal.

Figure 52 : Message d'erreur de la modification d'un canal activé.

3.3 Robot Framework

Nous avons intégré notre serveur dans un ''robot Framework'' python qui nous avons développé, pour automatiser notre serveur à travers de suites des tests automatiques.

La figure 53 illustre quelques ''cas de test'' qui ont été faites par notre ''robot Framework'' ainsi que les résultats de ces cas de teste sur notre serveur.

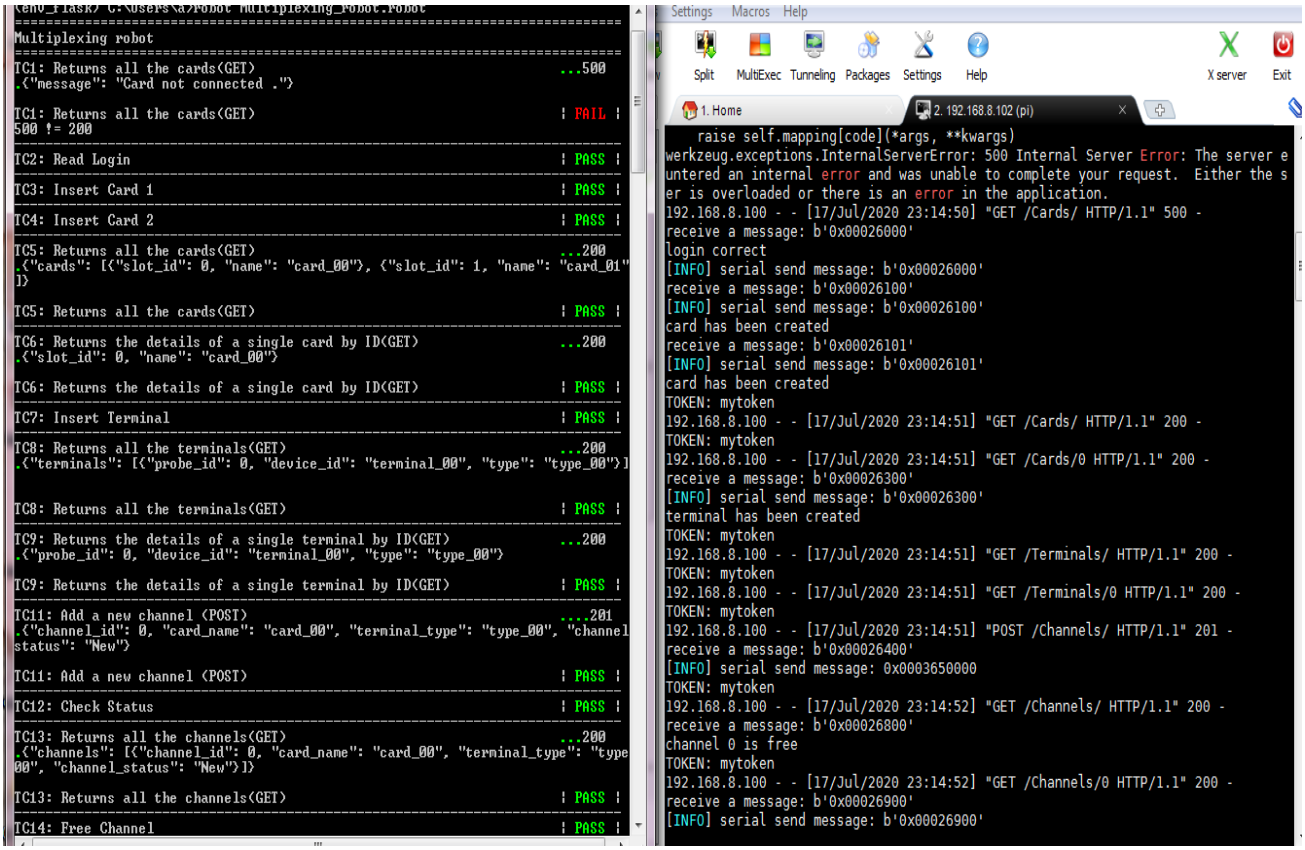
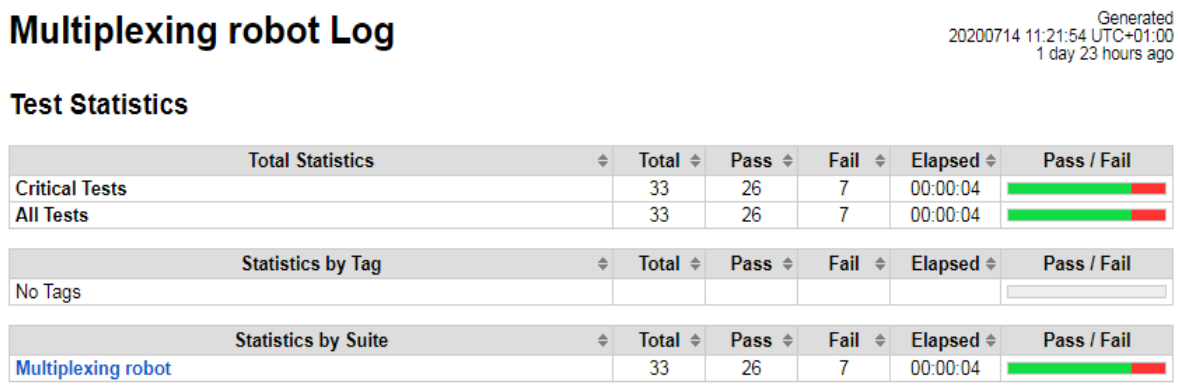


Figure 53 : Résultat de test automatique.

Lorsque les tests sont terminés, deux fichier HTML vont être générés **''REPORT''** et **''LOGO''** contenant les résultats de tests avec tous les détails nécessaires à savoir le nombre des tests, résultats de chaque cas de test, tous les données échangés, etc.

La figure 54 illustre les statistiques de suites de tests, le nombre de tests exécutés, et leurs résultats **''Pass''** ou bien **''Fail''.**



Multiplexing robot Log

Generated
20200714 11:21:54 UTC+01:00
1 day 23 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	33	26	7	00:00:04	
All Tests	33	26	7	00:00:04	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Multiplexing robot	33	26	7	00:00:04	

Figure 54 : Statistiques de test.

La figure 55 illustre un rapport de test montre tous les cas de test en détail

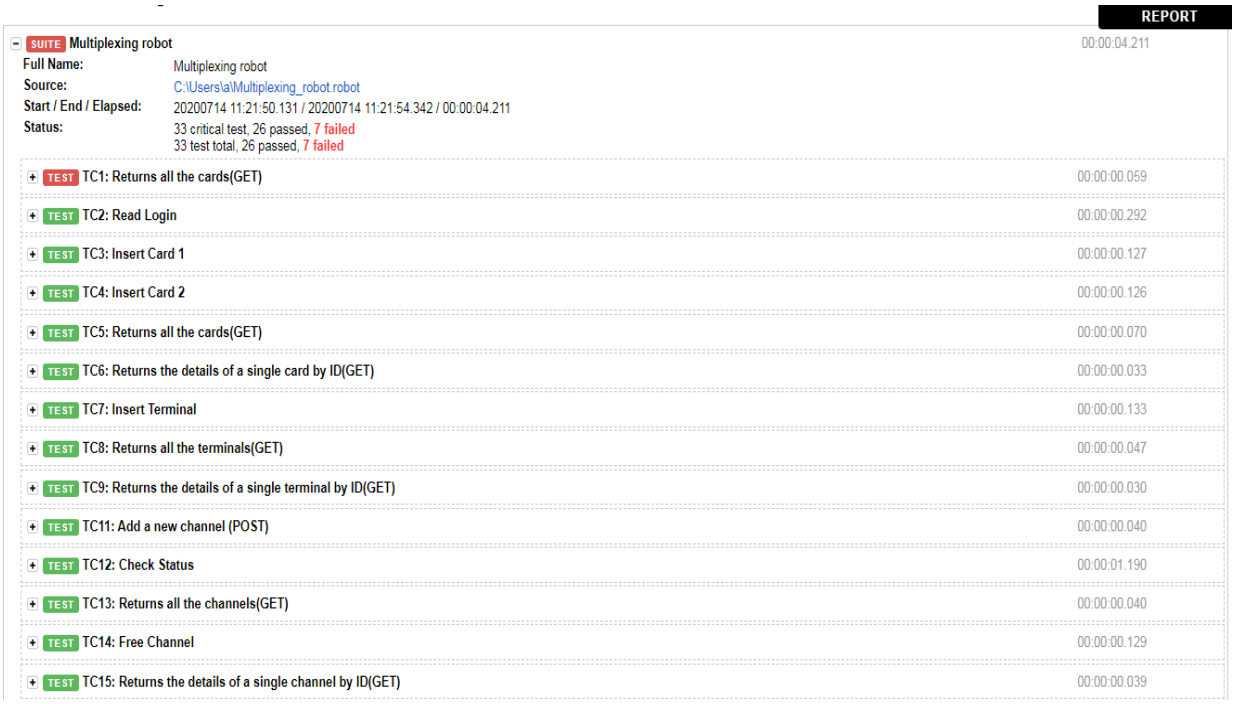


Figure 55 : Rapport de test.

Conclusion

Dans ce chapitre, nous avons présenté le travail réalisé, toutes les étapes suivis pour sa réalisation et les résultats obtenus en détail dans le but de donner une vision globale sur le fonctionnement de notre système.

Conclusion Générale

Ce rapport présente notre travail réalisé durant notre projet de fin d'études au sein de l'entreprise TELNET. Ce projet consiste à concevoir et développer un serveur web offrant des API REST qui assurent un multiplexage des cartes à puce avec des terminaux de paiement, puis il sera intégré dans un environnement de test automatique.

L'enchainement de notre projet depuis l'étude jusqu'à la réalisation est décrit tout au long de ce rapport qui résume quatre mois de travail rigoureux. Nous avons commencé par étudier le cadre général du projet dans le premier chapitre. Nous avons ensuite cité les différentes bases théoriques et techniques nécessaires à la réalisation de notre projet dans le deuxième chapitre. Passons à l'analyse et la spécification des besoins ainsi que la conception orientée objet de notre travail dans le troisième chapitre. En fin dans le chapitre 4, nous avons présenté la réalisation de système, nous avons décrit en détail toutes les étapes de la mise en œuvre de notre projet, ainsi que les résultats obtenus.

Au cours de la réalisation de ce projet, nous avons rencontré quelques difficultés au niveau de la première partie de notre projet qui est le chargement automatique de firmware dans la carte FPGA à partir de la RPI mais nous avons débrouillé ça par le développement d'un simulateur python qui remplace toute la carte d'extension.

En guise de conclusion, malgré les différentes difficultés rencontrées, ce stage a été une grande occasion pour améliorer nos connaissances par des nouvelles compétences sur les deux plans techniques et non techniques. Ainsi, nous reconnaissons que la réalisation de ce projet nous a permis de mettre en œuvre nos connaissances académiques acquises tout au long de notre formation à l'´Ecole Nationale d'Electronique et de Télécommunication de Sfax, et de les approfondir en pratique dans le domaine du système embarqué.

En guise de perspectives, nous allons remplacer le simulateur python que nous avons créé par une carte FPGA réelle, et ajouter une autre fonctionnalité de notre serveur qui est la récupération des données d'échange entre les terminaux de paiement et les cartes à puce.

Webographie

[1]. https://www.cfpb.fr/formations/formation-sur-mesure/s-adapter-aux-evolutions-de-la-monetique (L'Ecole supérieure de la banque le 07 octobre 2020)

[2]. http://www.groupe-telnet.com/en/content/presentation (Consulté le 25 Juin 2019).

[3]. Banque Info. [Online]. https://www.banque-info.com (Consulté le 27 Juin 2019).

[4]. https://www.presse-citron.net/le-raspberry-pi-4-est-de-sortie-voici-la-fiche-technique/ (Setra, le 24 juin 2019)

[5]. http://casteyde.christian.free.fr/system/linux/guide/online/c7308.html (Copyright © 2004 Christian Casteyde)

[6]. https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445304-python definition-et-utilisation-de-ce-langage-informatique/ (Mis à jour le 31/03/20)

[7]. https://en.wikipedia.org/wiki/Swagger_(software) (Mis à jour le 4 Février 2020)

[8]. https://mobaxterm.mobatek.net/documentation.html#1_1 (© 2008 - 2020 Mobatek ®)

[9]. https://www.presse-citron.net/le-raspberry-pi-4-est-de-sortie-voici-la-fiche-technique/ (Setra, le 24 juin 2019)

Annexe
Généralités sur le support matériel sous Linux

Introduction

Linux gère la plupart des périphériques comme des fichiers spéciaux. De plus, la plupart des gestionnaires de périphériques peuvent être chargés dynamiquement, sous la forme de modules du noyau. Cette section présentera donc les notions de base concernant les modules du noyau, les fichiers spéciaux de périphériques, ainsi que leurs rôles respectifs dans la configuration d'un nouveau matériel.

1 Modules du noyau

Les modules du noyau sont des bibliothèques que les nous pouvons charger dans le noyau lorsque celui-ci a besoin d'une certaine fonctionnalité. Une fois chargés, les modules font partie intégrante du noyau et ajoutent leurs fonctions à celles existantes. Ces bibliothèques sont normalement stockées dans le répertoire **/lib/modules/version/,** où version est le numéro de version du noyau pour lequel ces modules ont été créés.

Beaucoup de fonctionnalités du noyau peuvent être configurées pour être utilisées sous la forme de modules. Dans le cas des pilotes de périphériques, les modules permettent de réaliser un noyau générique et de prendre en charge dynamiquement les différents périphériques de l'ordinateur. Certains pilotes ne fonctionnent d'ailleurs que sous la forme de modules, aussi faut-il savoir les manipuler. Les modules sont également fortement utilisés pour la configuration automatique des périphériques connectables à chaud.

2 Les fichiers spéciaux de périphériques

2.1 Le système de fichiers virtuel udev

Depuis la version 2.6 du noyau, Linux dispose d'une fonctionnalité permettant aux applications de détecter l'apparition et la suppression des périphériques, que ceux-ci soient détectés au démarrage du système ou qu'ils soient branchés à chaud une fois l'ordinateur allumé.
Cette fonctionnalité est principalement utilisée par le sous-système udev (abréviation de « Userspace /dev »). Comme son nom l'indique, udev a pour principale fonction de prendre en charge la gestion du répertoire /dev/, mais en réalité il est capable de faire beaucoup mieux que cela. En particulier, udev peut réaliser les opérations suivantes lorsque le noyau signale la présence d'un nouveau périphérique :

- Chargement du module du pilote de périphérique si nécessaire

- Si le périphérique requiert un firmware, chargement de celui-ci
- Création du fichier spécial de périphérique et de ses alias nécessaires à l'utilisation du périphérique
- Exécution des opérations d'initialisation complémentaires ou lancement des applications utilitaires associées au périphérique ;
- Notification de la présence du périphérique à l'ensemble des autres programmes qui s'intéressent à la gestion du matériel (par exemple le gestionnaire de bureau).

« udev » peut également effectuer les opérations nécessaires à la suppression d'un périphérique, ce qui est utile pour les périphériques amovibles. Ainsi, la détection, la configuration et la notification dynamique de la présence des périphériques Plug and Play est totalement automatisée.

2.2 Principe de fonctionnements d'udev

Lorsque le noyau détecte un changement dans la configuration matérielle, il signale ce changement via un canal de communication aux applications qui s'y intéressent. udev utilise le démon udevd pour écouter sur ce canal de communication, et il met chacun de ces événements dans une file pour les traiter dans leur ordre d'apparition.

2.3 Chargement des firmwares

Grâce à **udev,** il est possible de faire en sorte que l'opération de chargement d'un firmware se fasse automatiquement. Lorsqu'un pilote désire charger le firmware dans le périphérique, il appelle une fonction du noyau qui génère un événement udev. **udev** prend alors en charge les opérations et localise le firmware, puis le fournit au noyau. Le pilote de périphérique peut alors effectuer le chargement effectif du firmware dans le périphérique. Pour cela, **udev** copie le fichier de firmware demandé par le pilote de périphérique dans un fichier dédié à cet effet dans l'entrée du périphérique du système de fichiers virtuel **/sys/.** Bien entendu, le chemin permettant de trouver les informations du périphérique dans le système de fichiers virtuel **/sys/** est fourni aux scripts de configuration d'udev via le mot clé **DEVPATH [9].**

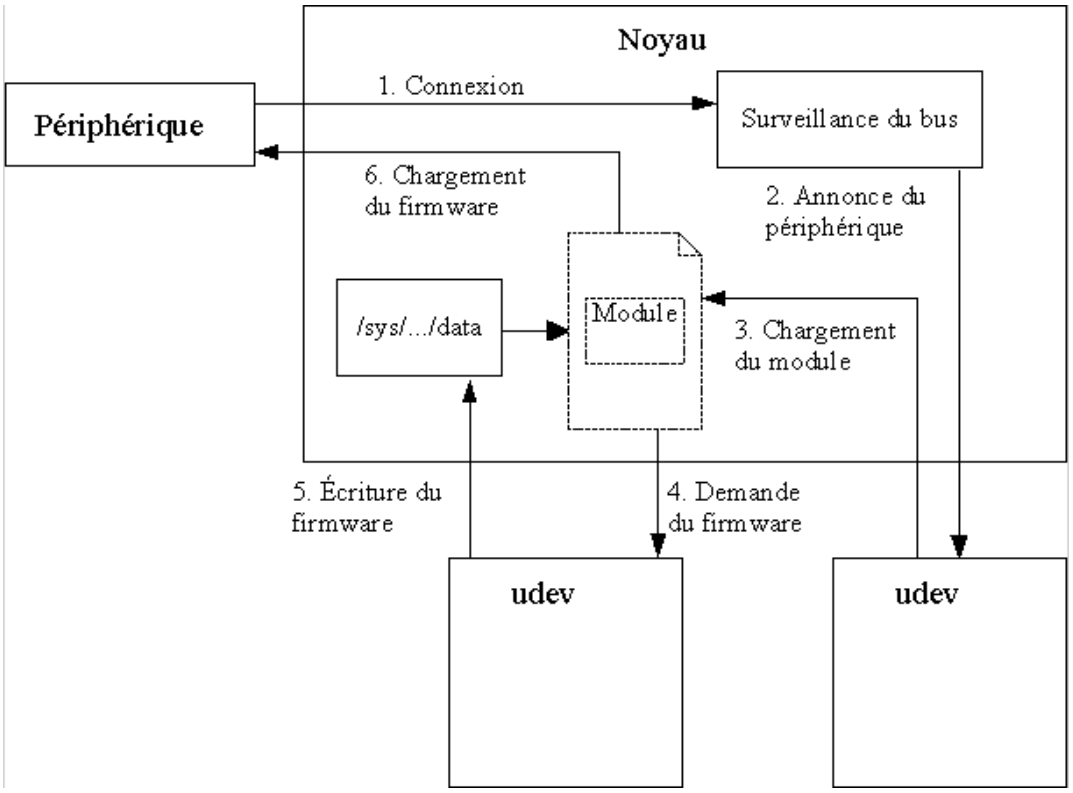


Figure 56 : Chargement de firmware.