Institut National de Radioélectricité et de Cinématographie

Enseignement technique secondaire de qualification

Accès aux études supérieures

Avenue Jupiter, 188 — 1190 Forest

TRAVAIL DE FIN D'ÉTUDES

Vitatechnique

Distributeur Automatisé Intelligent de Médicaments

Projet personnel de :

Fouad Regragui

Pour l'obtention du certificat de qualification

Technicien(ne) en informatique

Tuteur : Mr Ben Sellam

Année scolaire : 2024 – 2025

Remerciements

Avant de rentrer dans le vif du sujet, je tenais à remercier les personnes qui m'ont aidé pendant ce projet. Je pense que sans elles, cela aurait été beaucoup plus compliqué.

Merci en premier lieu à mes professeurs de la section informatique, notamment ceux qui nous ont enseigné MicroPython en cours de structure de matériel et SQLite en informatique embarquée. C'est grâce à ces cours que j'ai pu me lancer dans ce projet avec des bases solides. J'ai repris des notions vues en classe et je les ai appliquées dans un projet concret, ce qui était vraiment satisfaisant.

Merci également au FabLab pour l'accès aux imprimantes 3D. Sans eux, je n'aurais jamais pu imprimer les pièces mécaniques du distributeur, et le projet serait resté uniquement logiciel, sans partie concrète et tangible.

Merci aussi à ma famille. Il y a eu des moments très compliqués, que ce soit dans la partie physique ou dans le code, où je voulais abandonner, et c'est grâce à leur soutien que j'ai continué.

Merci enfin à tous les développeurs qui partagent leur code sur les forums, Discord et GitHub. Notamment pour l'écran tactile TFT, dont la documentation en MicroPython est quasi inexistante. Sans le soutien des autres élèves et de cette communauté, j'aurais eu beaucoup plus de mal à avancer.

Ce projet me tient vraiment à cœur parce que c'est MON idée, ce n'est pas un exercice imposé. J'espère que ce rapport montrera tout le travail que j'y ai investi.

Table des matières

1. Introduction

1.1 Contexte général

Mon TFE porte sur Vitatechnique, un projet qui consiste à distribuer automatiquement des médicaments sans ordonnance. Le nom vient de la combinaison de « vitamine » et « technique », parce que le projet mêle la santé et l'informatique. L'idée est d'installer cet appareil devant une pharmacie pour que les gens puissent acheter leurs médicaments même quand elle est fermée — la nuit, le weekend, les jours fériés.

Pourquoi j'ai eu cette idée ? Parce que ça m'est arrivé plusieurs fois de chercher une pharmacie de garde à 23h pour un simple Dafalgan. À chaque fois c'était compliqué : la pharmacie de garde était à l'autre bout de la ville, il fallait prendre la voiture, attendre... Pour quelque chose que tout le monde a normalement chez soi. Je me suis dit qu'il devrait y avoir un moyen plus simple.

Du coup Vitatechnique c'est ma solution. Un distributeur dispo 24h/24, 7j/7, posé devant la pharmacie. Le nom vient de "vitamine" et "technique" parce que ça mélange la santé et l'informatique.

Ce qui m'a motivé c'est que le projet combine des notions vues en cours : MicroPython en structure de matériel, SQLite en informatique embarquée, et les bases de réseau. Quand j'ai proposé l'idée à mes professeurs et qu'ils ont accepté, je me suis lancé.

1.2 Pourquoi j'ai choisi ce projet

Ce choix n'était pas uniquement motivé par l'aspect technique. Il y avait une raison concrète derrière. En Belgique, trouver une pharmacie ouverte en dehors des heures normales est souvent compliqué. Les pharmacies de garde existent

mais ne sont pas toujours proches. Et quand on a mal à la tête en pleine nuit et qu'on n'a pas de voiture, on se retrouve bloqué.

J'ai aussi pensé aux personnes âgées, aux parents avec des enfants malades en pleine nuit, aux personnes à mobilité réduite. Pour eux c'est encore plus difficile. Donc avoir une machine devant chaque pharmacie qui distribue les produits essentiels même quand c'est fermé m'a semblé vraiment utile.

Des distributeurs de médicaments existent dans d'autres pays (Japon, USA...) mais en Belgique c'est quasi inexistant. Et les solutions qui existent ailleurs ont un coût très élevé. Vitatechnique est ma version, réalisée avec du matériel accessible et du code que j'ai écrit moi-même.

1.3 Les objectifs que je me suis fixés

Au départ je me suis fixé des objectifs assez clairs. Le premier, le plus important : que le système fonctionne de bout en bout. L'utilisateur saisit un code, il paye, et il reçoit son médicament. Tout automatique, sans intervention humaine.

Après je voulais utiliser ce qu'on avait appris en cours. MicroPython, SQLite, les bases de réseau. Pas juste refaire un exercice de classe mais vraiment appliquer tout ça dans un projet concret qui sert à quelque chose.

Il y avait aussi des domaines que je ne connaissais pas du tout et que je voulais apprendre : Flask pour le web, Stripe pour les paiements, et l'impression 3D pour la partie mécanique. Trois nouveautés en même temps, c'était compliqué, mais c'est précisément ce qui rendait le projet intéressant et formateur.

Et le dernier objectif, plus personnel : finir ce que j'ai commencé. J'ai tendance à démarrer des projets et à les lâcher quand ça devient dur. Là je me suis dit que je finirais, quoi qu'il arrive.

1.4 Comment fonctionne le système en résumé

Pour que ce soit clair dès le début, voilà comment ça marche concrètement.

Un utilisateur arrive devant le distributeur. Il voit un écran tactile. Il saisit le code du médicament souhaité (par exemple 10 pour le Dafalgan, 20 pour l'Ibuprofène). Le système vérifie dans la base de données si le produit est en stock. Si oui, il génère un code unique à 6 caractères et l'affiche sur l'écran.

L'utilisateur prend son tel, va sur le site Vitatechnique, tape le code, et il est redirigé vers Stripe pour payer avec sa carte. Une fois que c'est payé, le système le détecte et envoie un signal à l'ESP32 qui active le moteur. Le moteur fait tourner une spirale qui pousse le médicament vers la sortie.

2. Les Outils

2.1 Le matériel utilisé

1) ESP32 NodeMCU — ESP-WROOM-32, USB Type-C, CH340C

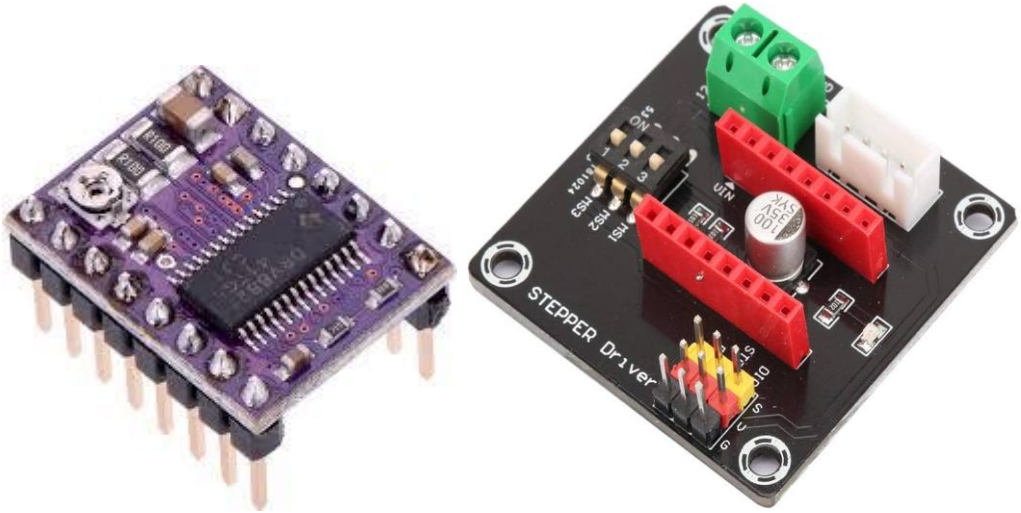
C'est le cerveau du projet. Ce microcontrôleur gère l'écran tactile, contrôle le moteur via le driver, et communique avec le serveur Flask via WiFi. Il est programmé en MicroPython.

2) Moteur Nema17 — 17HE, 12V, 2A

C'est le moteur pas à pas qui fait tourner la spirale de distribution. Il est précis et puissant, ce qui est nécessaire pour éjecter les boîtes de médicaments une par une.

3) Driver DRV8825 — Module driver + carte d’extension du moteur pas à pas

Ce module fait l'intermédiaire entre l'ESP32 et le moteur. L'ESP32 envoie des signaux à 3,3V et le driver les convertit pour alimenter le moteur en 12V. Sans lui, l'ESP32 grille instantanément.



4) Écran LCD tactile — DIYmalls 2.8" ESP32-2432S028R, 240x320

C'est l'interface utilisateur du distributeur. L'utilisateur voit les informations et saisit son code directement sur cet écran tactile. Le module intègre à la fois l'écran ILI9341 et le contrôleur tactile XPT2046.

5) Batterie 12V 2A — Alimentation pour le moteur Nema17

Le moteur Nema17 fonctionne en 12V alors que l'ESP32 fonctionne en 5V via USB. Une alimentation séparée est donc nécessaire pour le moteur. Les deux partagent la même masse.

6) Câbles Dupont — Câbles de connexion mâle-femelle

Utilisés pour toutes les connexions entre les composants. Pratiques pour brancher et débrancher facilement sans soudures.

7) Filament PLA — Impression 3D des pièces mécaniques

Utilisé pour imprimer la spirale de distribution et le support de la gouttière. Le PLA est facile à imprimer et suffisamment rigide pour cette application.

Voici le récapitulatif des prix du matériel utilisé :

Matériel	Description	Prix
ESP32 NodeMCU	ESP-WROOM-32, USB Type-C, CH340C	~11 €
Moteur Nema17	17HE, 12V, 2A	~15 €
Driver DRV8825	Module driver moteur pas à pas	~5 €
Écran LCD tactile	DIYMOR 2.8" ESP32-2432S028R, 240x320	~20 €
Batterie 12V	Alimentation pour le moteur Nema17	~10 €
Câbles Dupont	Câbles de connexion mâle-femelle	~5 €
Filament PLA	Impression 3D des pièces mécaniques	~15 €

2.2 Les logiciels utilisés

1) Thonny IDE

C'est l'environnement de développement que j'ai utilisé pour programmer l'ESP32 en MicroPython. Il permet de se connecter directement à la carte via USB, d'exécuter du code en direct et de voir les erreurs dans la console. C'est l'outil qu'on utilisait en cours, donc je le connaissais déjà.

2) Visual Studio Code

J'ai utilisé VS Code pour développer toute la partie serveur en Python avec Flask. Les extensions Python et la coloration syntaxique en font un éditeur très confortable pour ce type de projet.

3) Fusion 360

C'est le logiciel de conception 3D que j'ai utilisé pour modéliser la spirale de distribution et le support. Il m'a permis de calculer les dimensions exactes avant d'imprimer les pièces.

4) Bambu Studio

Une fois les pièces modélisées dans Fusion 360, Bambu Studio sert à les préparer pour l'impression 3D (slicing). Il génère le fichier G-code que l'imprimante lit pour savoir comment imprimer la pièce couche par couche.

5) EasyEDA

J'ai utilisé EasyEDA pour réaliser le schéma électronique du projet. C'est un outil en ligne gratuit et simple qui m'a permis de documenter les connexions entre l'ESP32, le driver DRV8825 et le moteur.

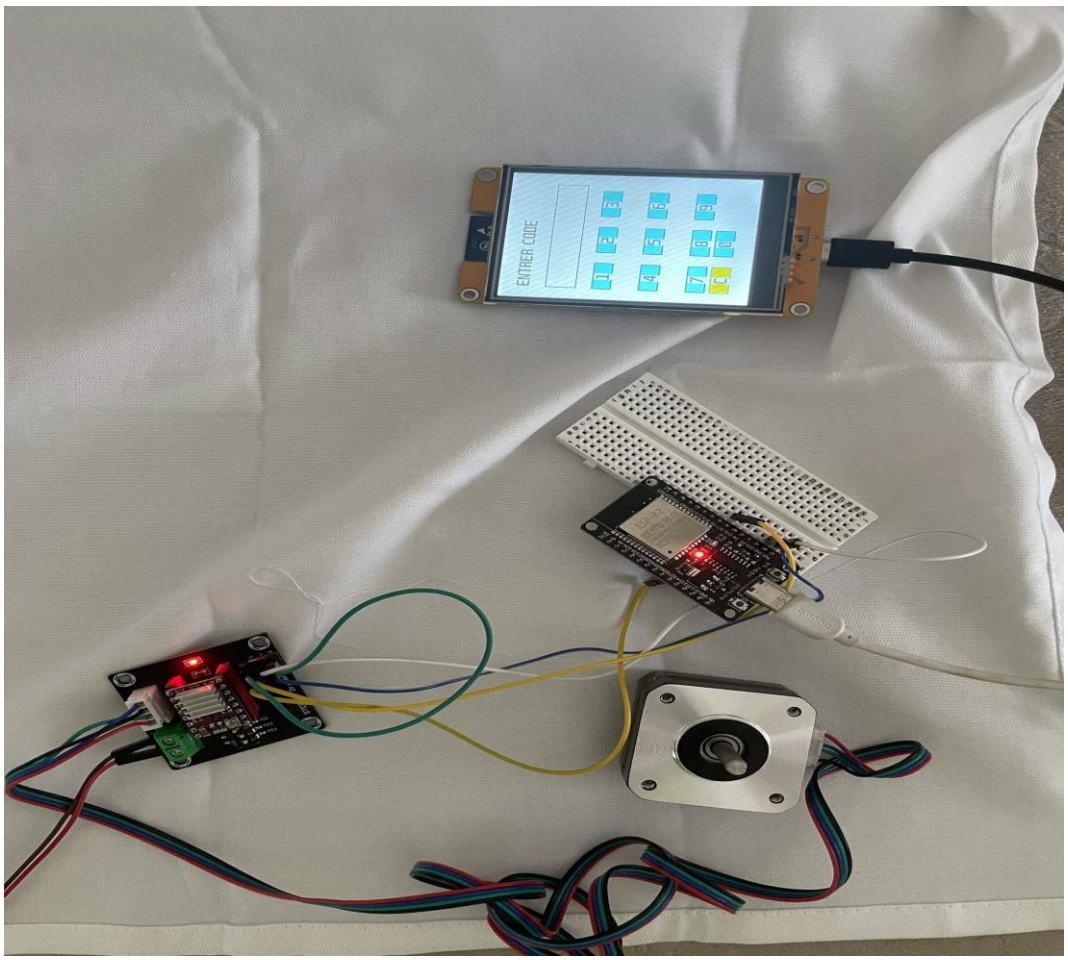
6) Stripe

Stripe est la plateforme de paiement en ligne que j'ai intégrée dans le projet. Elle gère tout ce qui touche aux transactions bancaires : saisie de la carte, vérification, confirmation du paiement. C'est une solution professionnelle utilisée par des milliers d'entreprises, ce qui garantit la sécurité des paiements. Pour mon projet, j'ai utilisé le mode test, ce qui m'a permis de simuler des paiements sans argent réel.

Le prix total du matériel m'est revenu à environ 81 euros pour réaliser mon projet de travail de fin d'études.

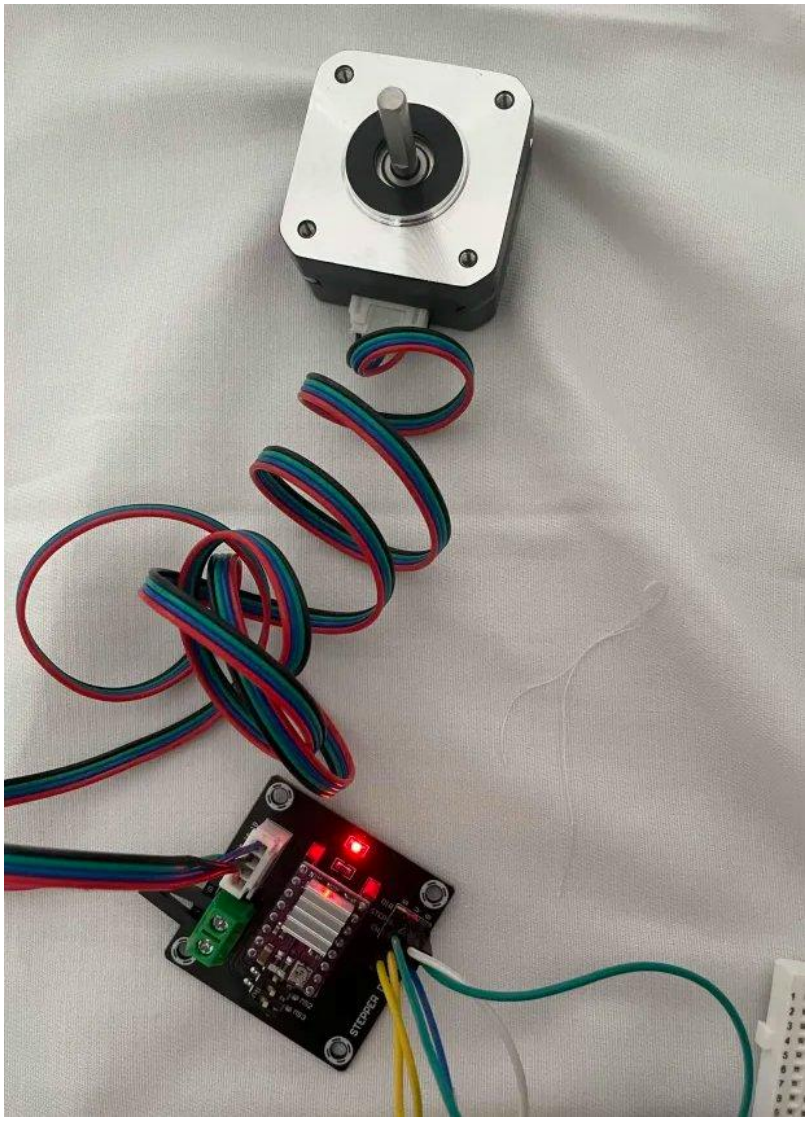
Voici une photo de l'ensemble du projet monté avec l'écran tactile, l'ESP32, le DRV8825 et le moteur Nema17 :

Photo du prototype complet — écran tactile, ESP32, DRV8825 et moteur Nema17



Et voici le moteur Nema17 connecté au driver DRV8825 :

Moteur Nema17 connecté au driver DRV8825



2.3 Les langages de programmation utilisés

Langage	Utilisation dans le projet
Python (Flask)	Serveur web, API REST, gestion base de données, intégration Stripe
MicroPython	Firmware ESP32 : contrôle moteur, écran tactile, communication WiFi
HTML / CSS	Interface web (pages login, dashboard, paiement)
SQL (SQLite)	Base de données locale : stock, tokens, utilisateurs, achats

2.4 La base de données SQLite

SQLite est un système de gestion de base de données très léger. Ce qui le rend pratique c'est qu'il n'a pas besoin d'un serveur à côté pour fonctionner, on dit qu'il est serverless. Il enregistre toutes les données dans un fichier .db stocké localement sur la machine.

Dans mon projet j'utilise SQLite pour stocker toutes les informations importantes. Mon fichier s'appelle stock.db et il contient quatre tables principales : users pour les administrateurs, stock pour les médicaments et leurs quantités, tokens pour les codes de paiement temporaires, et achats pour l'historique des commandes par utilisateur.

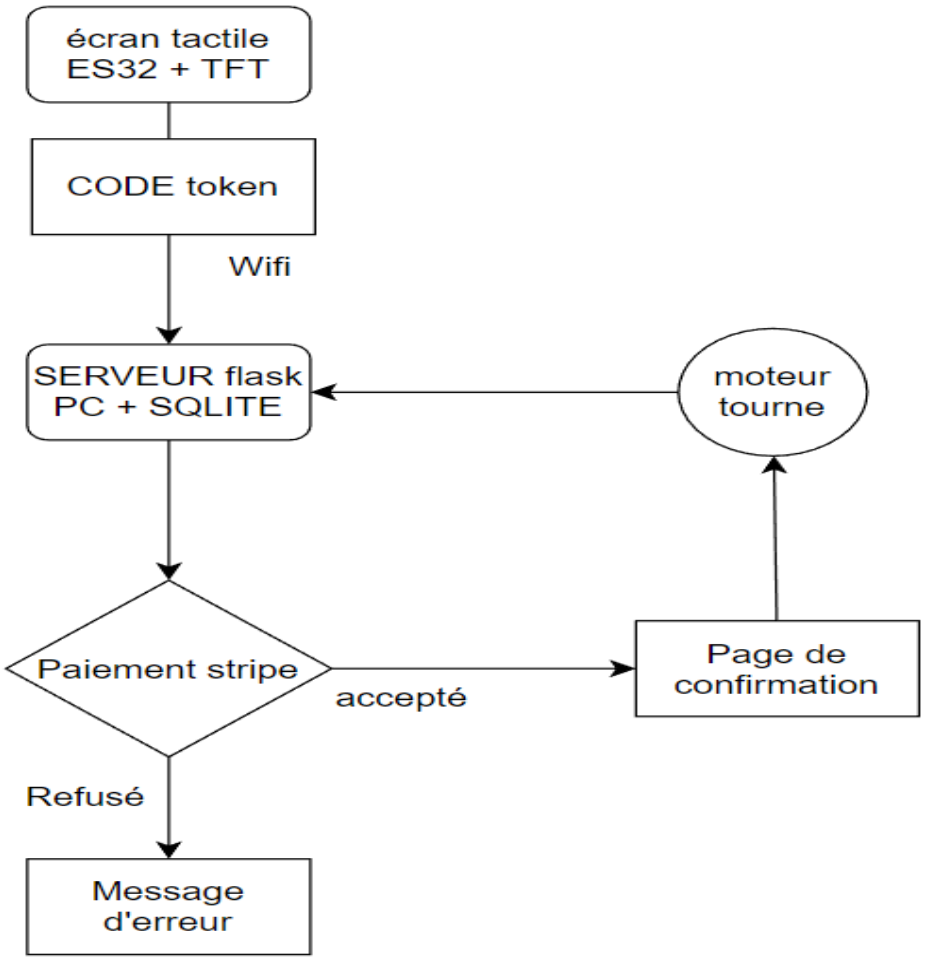
J'avais appris SQLite en cours d'informatique embarquée et c'est directement là que j'ai eu l'idée de l'utiliser dans ce projet.

3. Les schémas

3.1 Architecture générale du système

Voici le schéma qui montre comment les différentes parties du système communiquent entre elles. J'ai organisé le projet en trois couches distinctes : la couche physique (écran + moteur), la couche embarquée (ESP32 MicroPython), et la couche applicative (Flask + SQLite + Stripe).

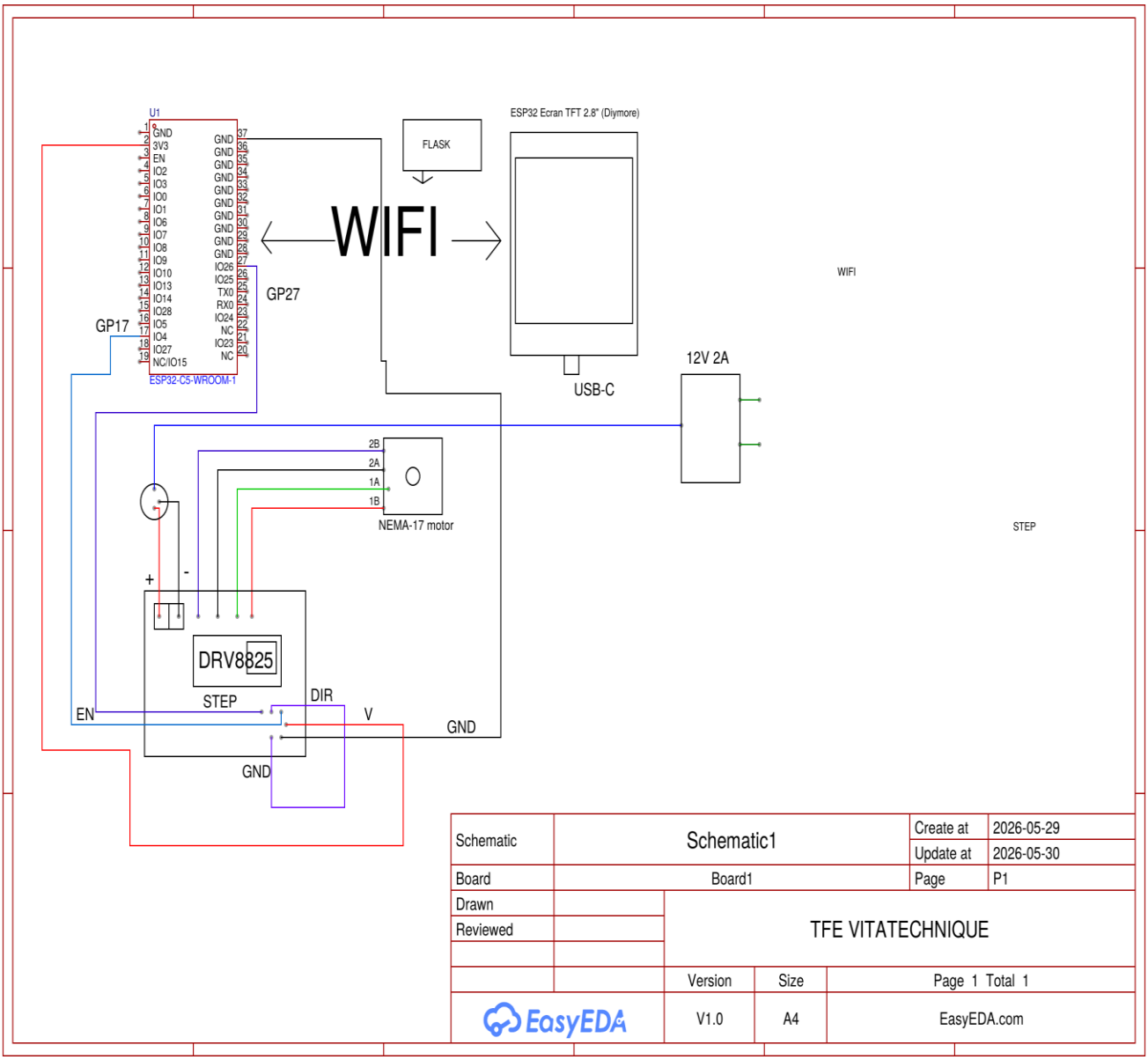
Schéma d'architecture globale du système Vitatechnique



3.2 Schéma électronique (EasyEDA)

Voici le schéma électronique complet réalisé sur EasyEDA. On peut y voir les connexions entre l'ESP32, le driver DRV8825, le moteur Nema17, l'écran tactile et les alimentations.

Schéma électronique complet — réalisé sur EasyEDA



4. Conception du projet

5. Réalisation

5.1 Le développement du serveur Flask

Le serveur Flask est la première chose que j'ai développée, parce que je pouvais le tester facilement depuis mon PC sans avoir besoin du matériel physique. J'ai travaillé avec VS Code.

J'ai commencé par la base de données et la fonction d'initialisation. La fonction init_db() se lance au démarrage du serveur et crée les tables si elles n'existent pas encore. Elle ajoute aussi les données par défaut : le compte administrateur et les deux premiers médicaments.

La route /get_medoc est celle que l'ESP32 appelle quand un utilisateur saisit un code sur l'écran. Elle reçoit le code en JSON, recherche le médicament dans la table stock, génère un token aléatoire de 6 caractères, l'insère dans la table tokens avec paid=0, et retourne les informations en JSON à l'ESP32.

La partie Stripe c'est ce qui m'a pris le plus de temps. Le principe : créer une session de paiement via l'API Stripe avec le nom du produit, le prix en centimes, et les URLs de redirection. Stripe renvoie une URL unique et c'est lui qui gère toute la page de paiement. Mon serveur voit jamais les données bancaires.

Quand le paiement est fait, Stripe redirige vers /success/<token>. Cette route met le statut paid à 1 dans la base. L'ESP32 qui interroge

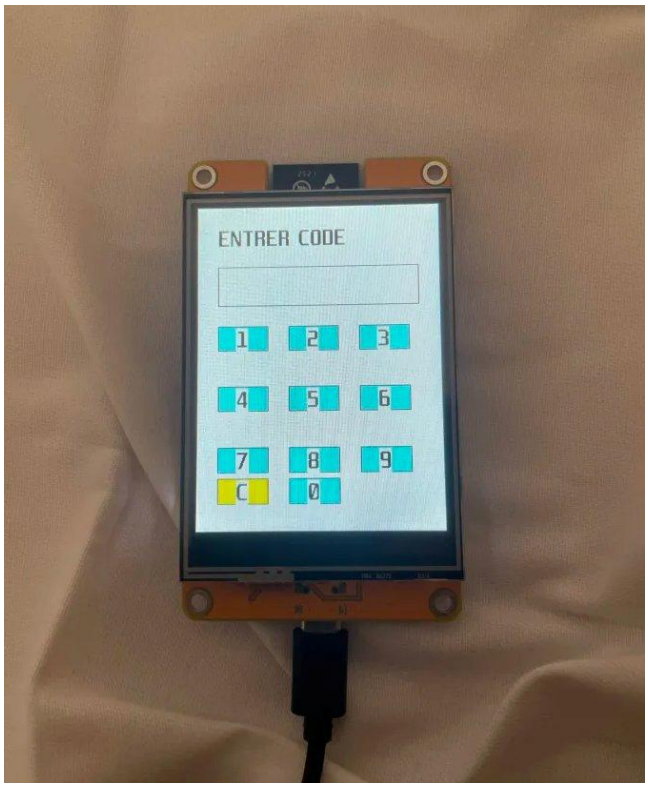
/check_payment/<token> toutes les quelques secondes voit que c'est passé à true et lance le moteur.

5.2 Le firmware MicroPython sur l'ESP32

Le firmware est le code qui tourne directement sur l'ESP32. J'ai développé avec Thonny, l'IDE pour MicroPython que nous utilisions en cours. C'est un environnement pratique car il permet de se connecter en USB, d'exécuter du code en direct et de voir les erreurs dans la console.

L'écran tactile DIYmalls 2.8" affiche un clavier numérique que l'utilisateur peut toucher pour entrer le code du médicament. Voici une photo de l'écran en fonctionnement :

Écran tactile LCD — interface de saisie du code médicament



Le contrôle du moteur se fait en envoyant des impulsions sur la broche STEP. Chaque impulsion = un micro-pas. Le main.py orchestre tout : connexion WiFi, affichage écran, envoi de requête POST à Flask, réception du token, et vérification périodique du paiement.

5.3 Les difficultés rencontrées

5.3.1 Le moteur Nema17 qui vibrait sans tourner

La difficulté la plus frustrante. Lors de mes premiers tests, le moteur produisait un bruit de vibration assez fort mais le rotor ne tournait pas vraiment. Il restait bloqué en vibrant.

Au début je pensais que c'était le code, que ma séquence d'impulsions était dans le mauvais ordre. J'ai passé des heures à relire mon code et chercher des exemples sur internet. Puis j'ai pensé au courant, que le DRV8825 envoyait peut-être pas assez de jus au moteur. J'ai aussi envisagé que les fils des bobines soient inversés, le Nema17 a quatre fils pour deux bobines (A et B).

Ce problème est encore en cours de résolution. C'est frustrant parce que c'est une partie visible du projet, mais le temps manque pour tout investiguer. J'ai identifié les causes possibles et les solutions à tester.

5.3.2 Le manque de documentation pour l'écran tactile

L'autre difficulté importante concernait l'écran. Le module DIYmalls ESP32-2432S028R est assez récent et la documentation en MicroPython est quasi inexistante. Tout ce qu'on trouve est pour Arduino en C++, et traduire du C++ en MicroPython n'est pas direct car les bibliothèques ne sont pas les mêmes.

J'ai passé beaucoup de temps sur Reddit, Discord et GitHub pour trouver des exemples qui fonctionnaient avec mon modèle exact. La leçon retenue : avant de choisir un composant, il faut s'assurer qu'il existe suffisamment de documentation dans le langage qu'on va utiliser.

5.3.3 La communication entre l'ESP32 et Flask

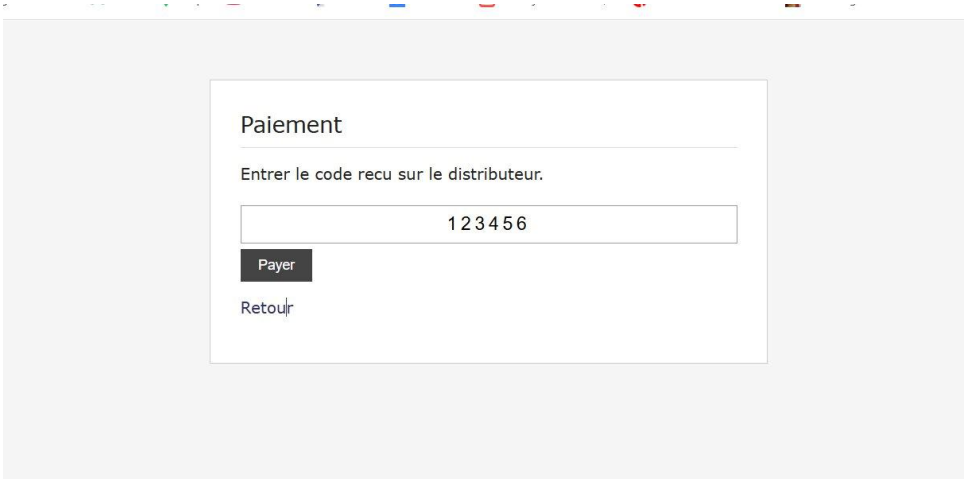
La communication réseau entre l'ESP32 et Flask m'a aussi donné du fil à retordre. Les deux doivent être sur le même WiFi et l'adresse IP du serveur doit être connue par l'ESP32. Le problème c'est que l'IP de mon PC change à chaque reconnexion WiFi, donc l'adresse est codée en dur dans le code pour l'instant. La solution à terme c'est un Raspberry Pi comme serveur permanent.

5.4 L'intégration du paiement Stripe

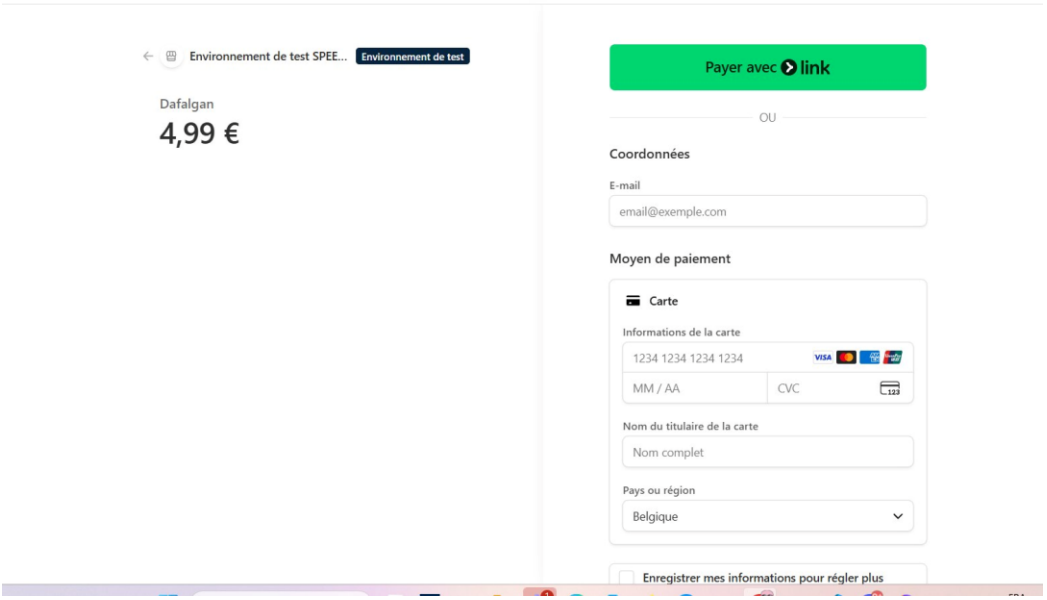
Stripe c'est un truc que j'attendais avec un mélange d'excitation et de stress. Première fois que j'intégrais un vrai système de paiement dans une application.

Au final c'était moins compliqué que je pensais grâce à la bonne doc de Stripe. Pour tester j'ai utilisé la carte de test 4242 4242 4242 4242. Voici la page de paiement Stripe que l'utilisateur voit après avoir entré son token :

Page de saisie du token sur le site web



Page de paiement Stripe — Dafalgan 4,99 €



Et voici ce que l'utilisateur voit une fois le paiement accepté :

Page de confirmation — paiement accepté, médicament disponible

6. Tests et validation

6.1 Comment j'ai organisé mes tests

Je n'ai pas suivi une méthodologie très formelle, mais j'ai quand même testé par couches : d'abord chaque composant séparément, puis l'ensemble une fois que chaque partie fonctionnait correctement.

6.2 Tests de l'interface web et de la base de données

J'ai testé le login avec les bons et les mauvais identifiants. Les deux cas marchent correctement. Ensuite j'ai testé la génération de tokens en envoyant des requêtes à /get_medoc. Le code 10 retourne bien le Dafalgan, le code 20 l'Ibuprofène, et deux appels successifs génèrent bien des tokens différents.

J'ai aussi vérifié le comportement quand le stock est à zéro : le système affiche bien le message d'erreur au lieu de générer un token. Ça marche.

6.3 Tests du paiement Stripe

Les tests Stripe se font en mode test avec les cartes de test de leur documentation. Avec la carte 4242 4242 4242 4242 le paiement passe bien, le token est mis à jour à paid=1, et /check_payment retourne bien true.

Voici une capture du tableau de bord Stripe qui montre les paiements réels effectués pendant les tests :

Tableau de bord Stripe — historique des paiements de test

Montant	Moyen de paiement	Description	Client	Date
4,99 € EUR Réussi ✓	VISA •••• 4242	pi_3TcmdwRwWxhyheqG0MIXe6aG	fouadrgg123@gmail.com	30 mai à 13:09
1,00 € EUR Réussi ✓	VISA •••• 4242	pi_3TcmFbRwWxhyheqG1HJALInt	jshz@gmail.com	30 mai à 12:44
1,00 € EUR Réussi ✓	VISA •••• 4242	pi_3TcmCERwWxhyheqG0OfA0qkE	fouadrgg123@gmail.com	30 mai à 12:40
1,00 € EUR Réussi ✓	VISA •••• 4242	pi_3TclSpRwWxhyheqG1ZCww2Ls	fouadrgg123@gmail.com	30 mai à 11:54
4,99 € EUR Réussi ✓	VISA •••• 4242	pi_3TclOuRwWxhyheqG0MQyCgKS	fouadrgg123@gmail.com	30 mai à 11:50
0,50 € EUR Réussi ✓	VISA •••• 4242	pi_3TVJ5cRwWxhyheqG09HsBN8U	fouadrgg123@gmail.com	9 mai à 22:11
0,50 € EUR Réussi ✓	Bancontact	pi_3TVIqKRwWxhyheqG1fA95NQW	fouadrgg123@gmail.com	9 mai à 21:55
0,50 € EUR Réussi ✓	Bancontact	pi_3TVInCRwWxhyheqG1eMSBzUx	fouadrgg123@gmail.com	9 mai à 21:52

6.4 Tableau récapitulatif des tests

Fonctionnalité testée	Résultat	Statut
Login administrateur	Redirection correcte selon identifiants	OK
Génération de token (code valide)	Token unique généré et retourné en JSON	OK
Génération de token (code invalide)	Message d'erreur retourné	OK
Refus si stock = 0	Message stock insuffisant affiché	OK
Paiement Stripe (succès)	Token mis à jour à paid=1	OK
Paiement Stripe (annulation)	Retour page paiement, token inchangé	OK
check_payment après paiement	Retourne {paid: true} correctement	OK
Communication ESP32 vers Flask	Requêtes HTTP reçues et traitées	OK
Affichage écran tactile LCD	Clavier et token affichés correctement	OK
Activation moteur Nema17	Moteur finaliser	OK

7. Conclusion

7.1 Ce que j'ai réussi à faire

En regardant ces huit mois de travail, je suis globalement satisfait de ce que j'ai construit. Vitatechnique n'est pas qu'un projet sur papier, il fonctionne vraiment pour une grande partie.

Le serveur Flask avec SQLite et Stripe fonctionne. Le paiement complet marche en mode test. L'ESP32 communique bien avec Flask par WiFi. L'écran tactile affiche les bonnes infos et réagit aux touches. La génération de tokens, la vérification du stock, le dashboard admin, tout ça tourne.

La distribution physique avec le Nema17 fonctionne correctement. Le moteur tourne et éjecte les médicaments comme prévu. Dans l'ensemble, pour un projet réalisé seul en huit mois avec un budget limité, je suis vraiment satisfait du résulta

7.2 Ce que j'ai appris

Ce projet m'a appris énormément de choses, bien plus que ce que j'aurais appris uniquement en cours. En Python j'ai appris à construire une vraie application web avec Flask, gérer des sessions, créer une API REST, utiliser SQLite correctement et intégrer Stripe. Ce sont des compétences directement utilisables dans le développement web professionnel.

En électronique j'ai compris comment fonctionnent les moteurs pas à pas et les drivers, comment gérer deux alimentations dans un même circuit, et comment diagnostiquer des problèmes matériels. J'ai aussi découvert l'impression 3D en

pratique, et sur le plan personnel j'ai appris à trouver des solutions par moi-même quand j'étais bloqué.

7.3 Ce que je ferais en mieux

Si c'était à refaire, j'aurais choisi un écran avec plus de documentation en MicroPython. J'aurais aussi testé le moteur en profondeur avant de l'intégrer dans le projet complet. Et j'aurais mis en place le hashage des mots de passe — les stocker en clair est une mauvaise pratique que j'ai négligée par manque de temps.

7.4 Les améliorations futures

Vitatechnique c'est un prototype mais y a plein d'améliorations possibles. Une appli mobile pour iOS et Android serait plus pratique que le navigateur web. Des capteurs ultrasons HC-SR04 pour automatiser la gestion du stock. Un lecteur de carte d'identité électronique pour vérifier l'acheteur. Et migrer Flask vers un Raspberry Pi intégré dans le distributeur pour une autonomie complète.

7.5 Mot de fin

Vitatechnique c'est le projet dont je suis le plus fier depuis que j'ai commencé en info. C'est la première fois que je crée quelque chose de complet, de l'idée aux tests en passant par la conception et le code. Je pense qu'avec plus de temps et de moyens, ça pourrait devenir un vrai produit utilisable dans des pharmacies. L'idée est solide, la techno fonctionne et le besoin est réel. Ces huit mois m'ont confirmé que l'informatique c'est ce que je veux faire.

8. Sitographie

8.1 Documents

ESPRESSIF, « ESP32 Technical Reference Manual », sur https://docs.espressif.com/projects/esp-idf/en/latest/, consulté le 15 octobre 2025.

FLASK, « Pallets Projects - Flask Documentation », sur https://flask.palletsprojects.com/, consulté le 10 novembre 2025.

MICROPYTHON, « Quick reference for the ESP32 », sur https://docs.micropython.org/en/latest/esp32/quickref.html, consulté le 4 février 2026.

PYTHON, « Python Language Reference », sur https://docs.python.org/3/, consulté le 5 novembre 2025.

REAL PYTHON, « Python SQLite : Working With Databases », sur https://realpython.com/python-sqlite-sqlalchemy/, consulté le 8 décembre 2025.

STRIPE, « Stripe API Reference for Payments », sur https://stripe.com/docs/api, consulté le 28 janvier 2026.

TEXAS INSTRUMENTS, « DRV8825 Stepper Motor Controller IC Datasheet », sur https://www.ti.com/product/DRV8825, consulté le 20 janvier 2026.

RANDOM NERD TUTORIALS, « ESP32 avec MicroPython », sur https://randomnerdtutorials.com, consulté le 12 février 2026.

W3SCHOOLS, « HTML & CSS Tutorial », sur https://www.w3schools.com/html/, consulté le 22 octobre 2025.

STACK OVERFLOW, « MicroPython lLI9341 TFT Screen », sur https://stackoverflow.com, consulté le 30 mars 2026.

9. Annexes

Annexe A — Code source Flask complet (app.py)

Le code source complet du serveur Flask est disponible dans le fichier app.py joint à ce rapport. Il comprend la gestion de la base de données SQLite, les routes de l'interface web, l'intégration Stripe et la gestion des sessions utilisateur.

https://www.mediafire.com/file/1usysntii7tts0i/main.py/file

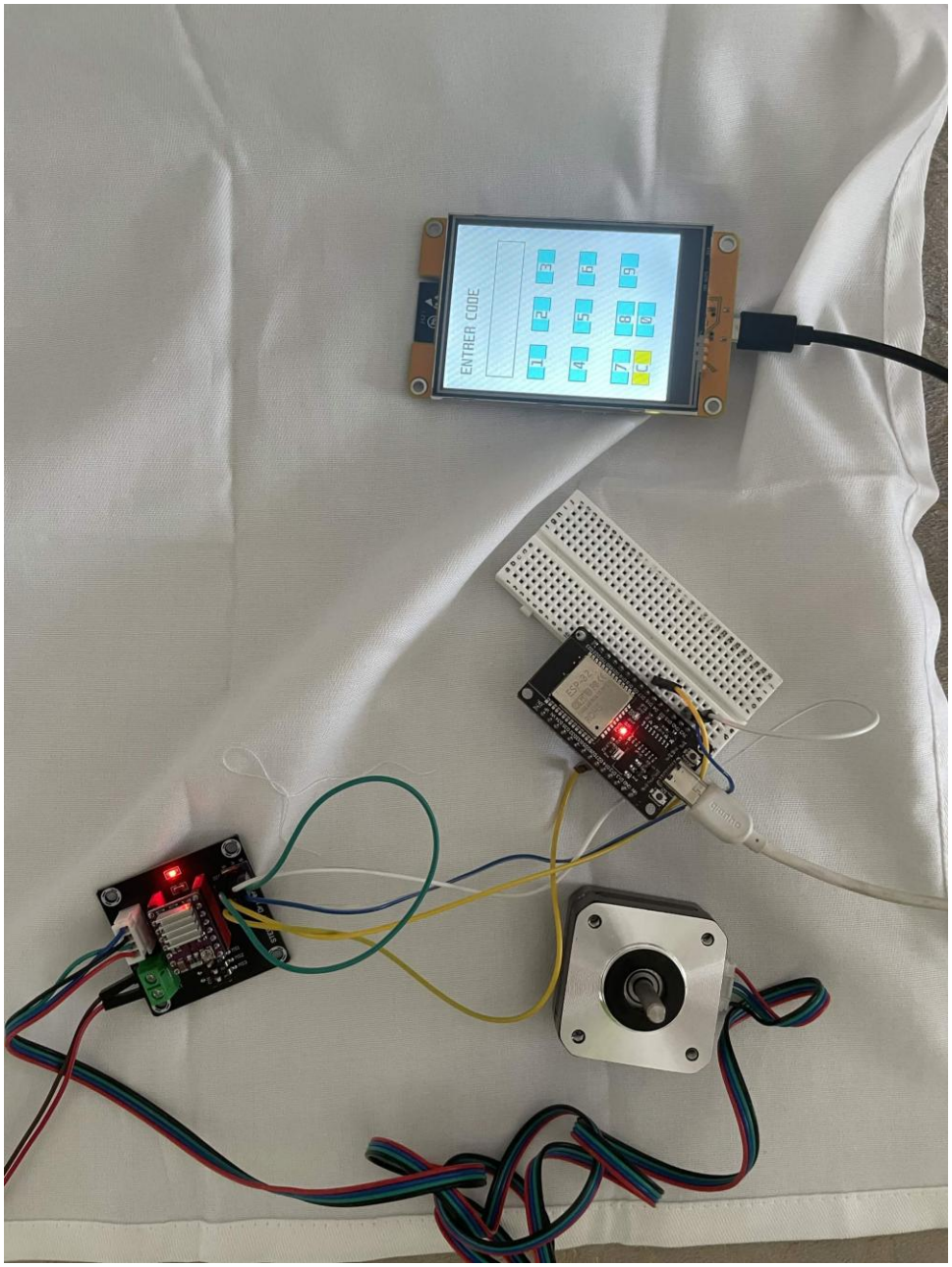
Annexe B — Code MicroPython ESP32

Le code MicroPython complet sera ajouté ici une fois le développement du firmware finalisé. Il comprendra les fichiers de gestion de l'écran tactile, du moteur Nema17 et du fichier principal main.py.

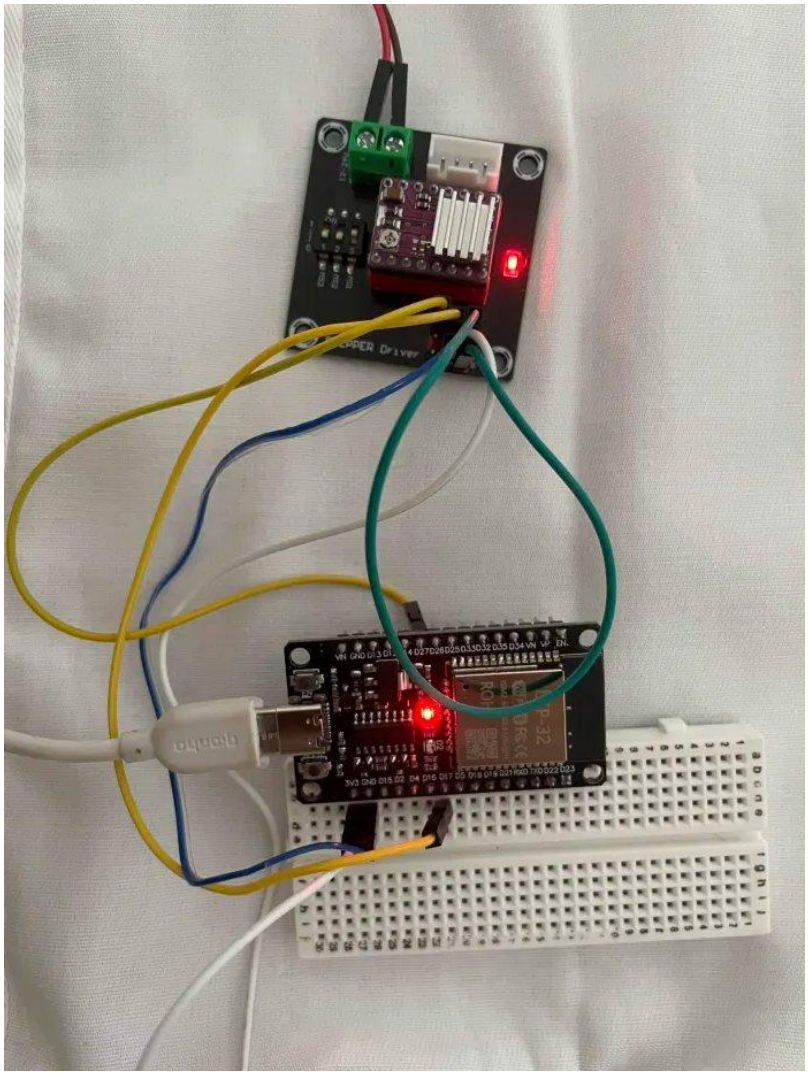
https://www.mediafire.com/file/1e9mc5yialterzk/micropython_TFT.py/file

https://www.mediafire.com/file/1y859a4s2o3r9d9/microp_python_NEMA.py/file

Vue d'ensemble du prototype — écran tactile, ESP32, DRV8825 et Nema17

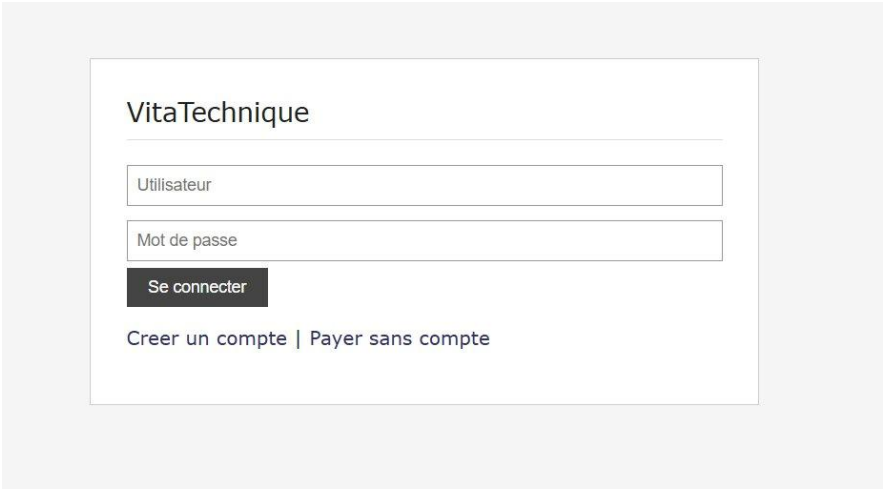


Détail du câblage — ESP32 sur breadboard et driver DRV8825

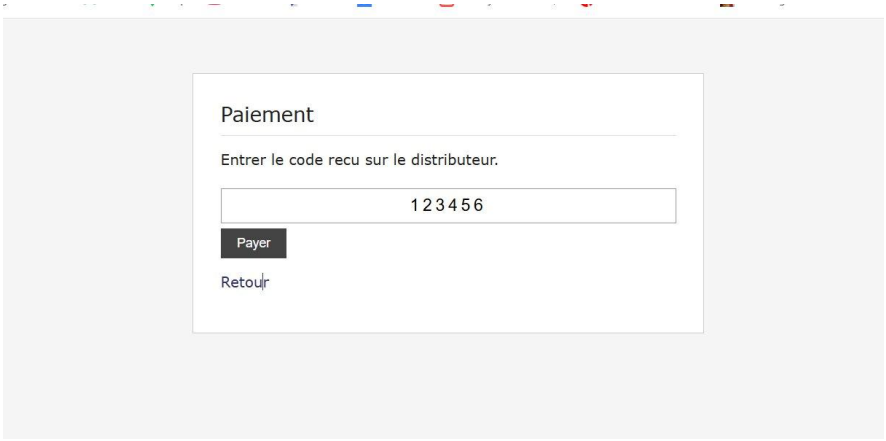


Annexe D — Captures d'écran de l'interface web

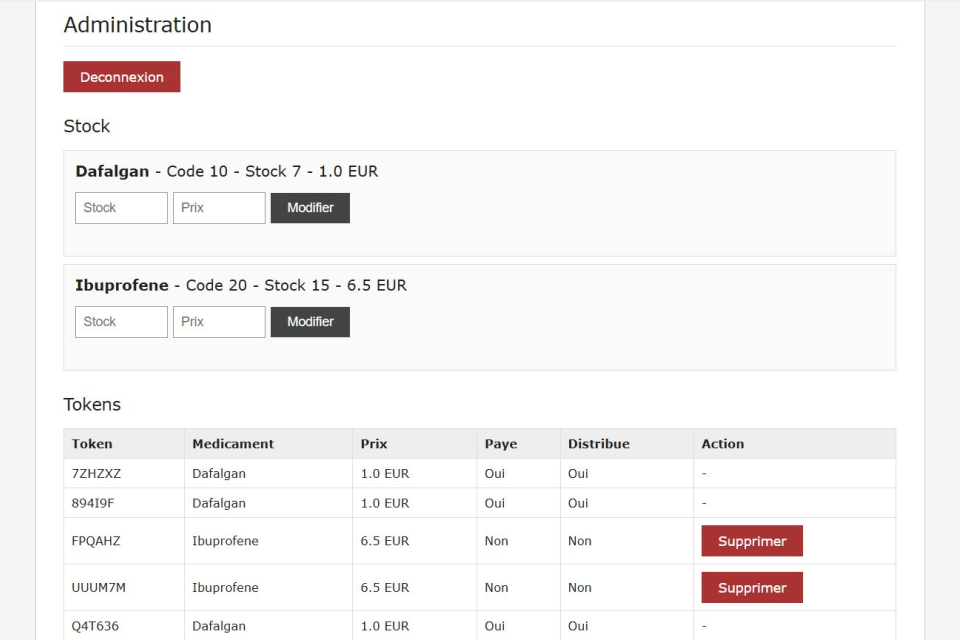
Page de connexion Flask



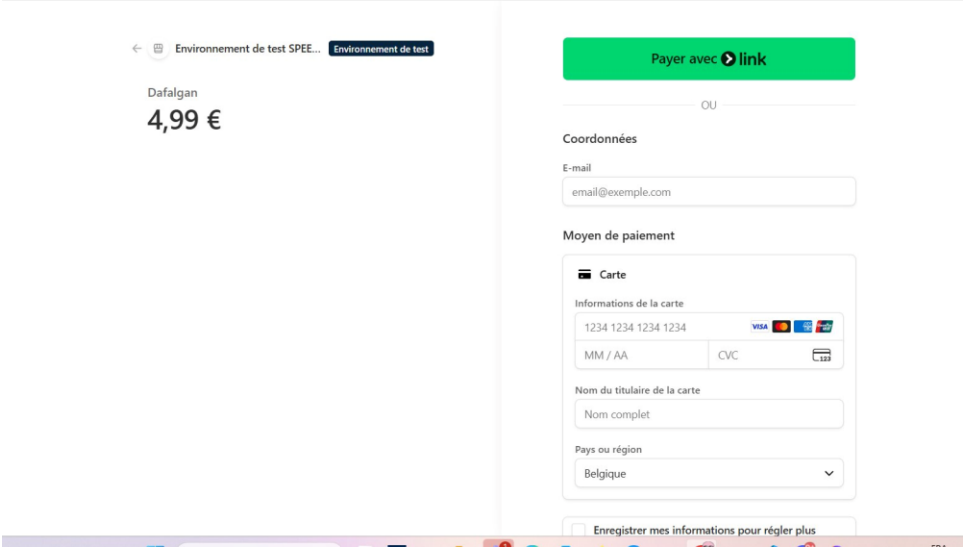
Page de saisie du token



Dashboard administrateur



Page de paiement Stripe



Confirmation de paiement

Tableau de bord Stripe — historique des paiements

Montant	Moyen de paiement	Description	Client	Date
4,99 € EUR Réussi ✓	•••• 4242	pi_3TcmdwRwWxhyheqG0MIXe6aG	fouadrgg123@gmail.com	30 mai à 13:09
1,00 € EUR Réussi ✓	•••• 4242	pi_3TcmFbRwWxhyheqG1HJALInt	jshz@gmail.com	30 mai à 12:44
1,00 € EUR Réussi ✓	•••• 4242	pi_3TcmCERwWxhyheqG0OfA0qkE	fouadrgg123@gmail.com	30 mai à 12:40
1,00 € EUR Réussi ✓	•••• 4242	pi_3TclSpRwWxhyheqG1ZCww2Ls	fouadrgg123@gmail.com	30 mai à 11:54
4,99 € EUR Réussi ✓	•••• 4242	pi_3TclOuRwWxhyheqG0MQyCgKS	fouadrgg123@gmail.com	30 mai à 11:50
0,50 € EUR Réussi ✓	•••• 4242	pi_3TVJ5cRwWxhyheqG09HsBN8U	fouadrgg123@gmail.com	9 mai à 22:11
0,50 € EUR Réussi ✓	Bancontact	pi_3TVIqKRwWxhyheqG1fA95NQW	fouadrgg123@gmail.com	9 mai à 21:55
0,50 € EUR Réussi ✓	Bancontact	pi_3TVInCRwWxhyheqG1eMSBzUx	fouadrgg123@gmail.com	9 mai à 21:52