MODÉLISATION DU PROJET

1. Introduction du système

Le projet consiste à développer un serveur web en Python (FastAPI) qui communique avec un ou plusieurs robots REF via des API REST. Le backend permet :

- L'enregistrement des robots (avec un identifiant unique UUID)
- La récupération des ordres de mission
- L'envoi d'informations de telemetry (vitesse instantanée, distance ultrasons, etc)
- L'affichage des instructions sur une page web (frontend en HTML seulement)

Le système est composé de :

- Un backend FastAPI structuré en architecture 3-tiers
- Une base de données SQLite
- Une console de contrôle (Python / Tkinter)
- Une simulation de robot (Java)
- Un frontend web interactif avec mise à jour automatique

2. Architecture générale

Une architecture trois-tiers était requise pour ce projet :

- Utilisateur (Frontend / Console / Simulateur)
- Backend FastAPI (routes API + logique métier)
- Base de données SQLite3 (robot.db)

3. Modèle Conceptuel de Données (MCD simplifié)

Les entités principales sont les suivantes :

Robots

- id
- name
- created_at

Instructions

- id
- robot_id
- blocks
- is_completed

Telemetry

- id
- robot_id
- speed
- ultrasonic_distance
- displacement_status
- current_line
- gripper_state
- time_stamp

Summary

- id
- robot_id
- timestamp

4. Spécification des API REST

Méthode	Endpoint	Description	Requête	Réponse
POST	`/robots`	Créer un nouveau robot (ID + nom)	Form-data ou JSON :`{ "robot_id": "REF-01", "name": "Pathfinder" }`	**HTML**: redirection vers / (form)**JSON**: `{ "message": "Robot Pathfinder (REF-01) added successfully" }`
GET	`/robots/list`	Lister tous les robots	—	`[{ "id": "...", "name": "...", "created_at": "..." }, …]`
POST	`/instructions/create`	Envoyer des instructions à un robot	JSON :`{ "robot_id": "REF-01", "blocks": [2,3,6] }`	`{ "status": "ok" }`
GET	`/instructions?robot_id={id}`	Récupérer la prochaine instruction non-complétée	Paramètre de requête :`robot_id=REF-01`	`{ "blocks": [2,3,6] }`
GET	`/instructions/all`	Lister l'historique des instructions	—	`[ { "robot_id":"REF-01", "blocks":[2,3,6], "is_completed":false }, … ]`

POST	`/telemetry`	Envoyer un point de télémétrie	JSON :`{ "robot_id":"REF-01", "vitesse":0.5, "distance_ultrasons":12.3, … }`	`{ "status": "ok" }`
GET	`/telemetry/all?robot_id={id}`	Récupérer l'historique de télémétrie	Paramètre de requête :`robot_id=REF-01`	`[ { "speed":0.5, "ultrasonic_distance":12.3, …, "time_stamp":"…" }, … ]`
POST	`/summary`	Marquer l'instruction et journaliser le résumé	JSON :`{ "robot_id":"REF-01" }`	`{ "status":"ok", "instruction_completed": true }`
POST	`/reset`	Supprimer les instructions en attente (optionnel)	(Optionnel) paramètre de requête :`robot_id=REF-01` —sans robot_id supprime tout	**HTML**: redirection vers `/` (form)**JSON**: `{ "status":"ok", "message":"Deleted X instructions" }`

5. Organisation du projet (structure des fichiers)

```
PROJECT/
├── __pycache__/
├── api/
│   ├── __pycache__/
│   ├── models/
│   │   ├── __pycache__/
│   │   └── schemas.py
│   ├── routes/
│   │   ├── __pycache__/
│   │   ├── instructions.py
│   │   ├── robots.py
│   │   ├── summary.py
│   │   └── telemetry.py
│   ├── _init_.py
│   └── master.py
├── archive/
│   ├── database.py
│   ├── interface.py
│   ├── main(inactive).py
│   └── robot_simulator.py
├── data/
│   └── robots.db
├── database/
│   ├── __pycache__/
│   ├── __init__.py
│   ├── base_model.py
│   ├── db_handler.py
│   └── db_init.py
├── interface/
│   └── gui.py
├── main/
│   ├── __pycache__/
│   ├── __init__.py
│   └── main.py
├── robot_control.egg-info/
│   ├── dependency_links.txt
│   ├── PKG-INFO
│   ├── requires.txt
│   ├── SOURCES.txt
│   └── top_level.txt
├── static/
│   └── data/
│       └── index.js
├── templates/
│   ├── index.html
│   └── partials/
│       ├── active.html
```

```
│       ├── history.html
│       ├── robots_add.html
│       ├── summary.html
│       └── telemetry.html
├── .env
├── .gitignore
├── README.md
├── requirements.txt
├── setup.py
└── start.py
```

6. Fonctionnement du frontend

- **index.html** : page principale qui organise en grille les sections « Active Instructions », « Instruction History », « Performance Summary », « Rover Fleet » et la télémétrie, et qui se recharge toutes les 5 s.
- **partials/active.html** : affiche la liste des robots actifs avec leurs blocs d'instruction en cours.
- **partials/history.html** : présente l'historique des instructions envoyées, avec statut « Complete » ou « Pending » et un bouton de réinitialisation.
- **partials/summary.html** : affiche pour chaque robot le dernier horodatage de résumé des opérations.
- **partials/robots_add.html** : formulaire d'ajout de nouveaux robots (ID + nom) avec validation HTML.
- **partials/telemetry.html** : tableau listant les données télémétriques (vitesse, distance, ligne, pince, horodatage) pour chaque robot.

7. Scénario de fonctionnement typique

- Le robot démarre et appelle POST /robots : il reçoit un identifiant UUID et un nom.
- Il consulte ses missions via GET /instructions.
- Il effectue la mission et envoie régulièrement son statut via POST /telemetry.
- L'utilisateur ouvre la page web (index.html) pour visualiser les déplacements.
- Une console Python (Tkinter) permet également d'interagir avec le robot.

8. Technologies utilisées

- Python 3.8+
- FastAPI
- SQLite3
- Module uuid
- HTML
- Tkinter (console de contrôle)
- Java (simulateur)

9. Contraintes et recommandations

- Seuls les modules Python suivants sont autorisés : fastAPI, sqlite3, uuid
- L'architecture doit respecter le modèle 3-tiers
- L'UUID est obligatoire pour identifier chaque robot
- Le code doit rester simple et fonctionnel
- Le respect de la structure des fichiers est obligatoire
- La priorité est donnée à l'efficacité et à la clarté fonctionnelle plutôt qu'à la complexité graphique

10. Modélisation de la pince et du châssis du robot

10.1 Conception mécanique globale

Le robot doit être capable de se déplacer de manière autonome et d'interagir physiquement avec son environnement, notamment en saisissant des cubes à l'aide d'une pince mécanique. Pour cela, une conception mécanique spécifique est requise, adaptée aux composants électroniques embarqués et aux objectifs du projet.

Objectifs mécaniques :

- Accueillir l'ESP32, les capteurs, les moteurs, la batterie
- Fixer une pince de préhension fonctionnelle
- Garantir la stabilité du robot durant les déplacements
- Respecter l'encombrement maximal autorisé (150x150x150 mm)

10.2 Modélisation 3D du châssis

La modélisation du châssis est en cours. Elle est réalisée sur le logiciel SolidWorks et elle comprend :

- La base roulante du robot
- Les fixations pour les moteurs et les roues
- Le support de la pince
- Les compartiments pour la carte ESP32, la batterie et le câblage

Le modèle final sera exporté en fichiers STL pour impression 3D.

10.3 Conception de la pince

Fonction attendue : la pince doit permettre de saisir et relâcher des cubes posés au sol.

Contraintes :

- Mécanisme commandé par un servomoteur (fourni)

- Ouverture et fermeture adaptées aux dimensions des cubes
- Fixation simple au châssis
- Dimensions maximales : 150x150x150 mm
- Utilisation des visseries et composants fournis

Phases de développement :

- Recherche comparative de pinces de préhension existantes
- Sélection d'un mécanisme simple et reproductible
- Conception 3D de la pince et dessins d'ensemble et de définition
- Fabrication (impression 3D)
- Montage et intégration sur le robot

État d'avancement :

- Le modèle 3D de la pince est terminé et validé
- Le système d'attache au châssis a été conçu pour un montage rapide
- Le test de fonctionnement (commande via servomoteur) est en cours de vérification

10.4 Assemblage et intégration

Une fois les pièces imprimées ou découpées, le robot sera assemblé avec :

- Le châssis imprimé
- Les moteurs fixés et câblés
- La pince installée et testée
- L'ESP32 reliée aux composants via un câblage propre

L'ensemble sera validé mécaniquement avant intégration des logiciels embarqués.