Galaxy ALPR Backend

A comprehensive backend platform for **Automatic License Plate Recognition (ALPR)**, enabling real-time vehicle and plate detection using AI/ML, robust database integration, and seamless communication with a modern frontend. This system is designed for smart parking management, security, and analytics in enterprise and public environments.

📖 Table of Contents

📝 Project Overview

Galaxy ALPR Backend provides:

- **RESTful API** for vehicle and license plate detection, querying, and statistics.
- **AI/ML Service** for high-accuracy detection using YOLO and OCR models.
- **Database Integration** for storing detection events, user sessions, and analytics.
- **Static & Output File Serving** for detected images and results.
- **Frontend Communication** for real-time dashboards and management.

This backend is designed to be modular, scalable, and easy to extend for new detection types, analytics, or integrations.

🏗️ Architecture
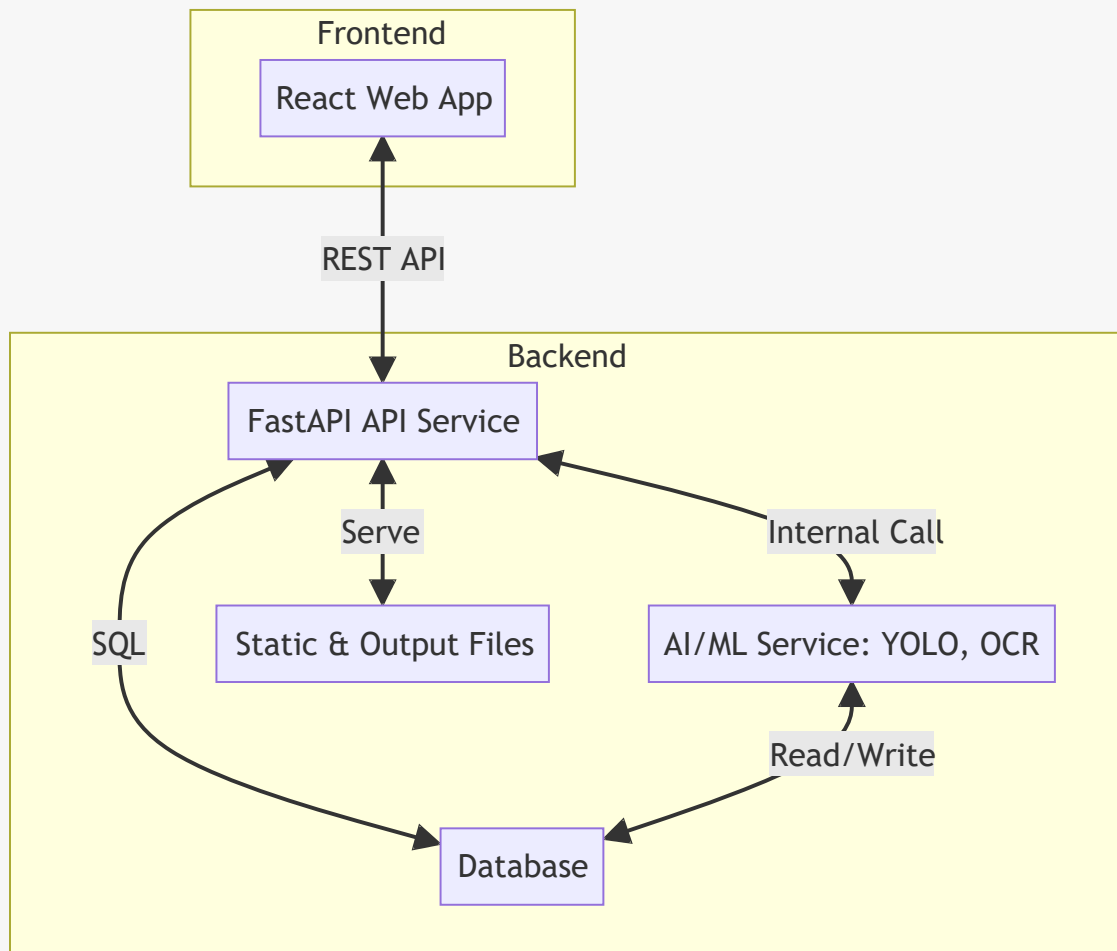
- **Frontend**: Sends HTTP requests to the API for detection, history, and analytics.
- **API Service**: Handles requests, triggers AI/ML inference, and manages data.
- **AI/ML Service**: Performs detection and recognition, returns structured results.
- **Database**: Stores all detection events, user data, and statistics.
- **Static/Output Files**: Served via FastAPI for detected images and artifacts.

📁 Folder Structure

```
backend/
│
├── app.py                 # Main FastAPI application and entry point
├── endpoints/             # Modular API endpoint definitions
│   ├── detect_image.py
│   ├── latest_detection.py
```

```
│   ├── detections.py
│   ├── plate_regions.py
│   ├── plate_queries.py
│   ├── vehicle_queries.py
│   ├── statistics.py
│   ├── plate_status.py
│   ├── session_queries.py
│   └── location_routes.py
│
├── PlateDetector.py        # AI/ML logic for plate and vehicle detection
├── database.py             # Database connection and ORM logic
├── requirements.txt        # Python dependencies
│
├── outputs/                # Output images, crops, and detection artifacts
│   ├── vehicles/
│   ├── plates/
│   ├── uploaded/
│   └── ...
├── uploads/                # Temporary storage for uploaded images
├── static/                 # Static assets served by the API
│
└── README.md               # This documentation
```

📁 Folder/Service Breakdown

File / Folder	Description
app.py	Main FastAPI app, server entry point, and configuration.
endpoints/	All API route definitions, organized by feature.
PlateDetector.py	AI/ML detection logic (YOLO, OCR, OpenCV, etc.).
database.py	Database models, schema, and connection logic.
outputs/	Stores detection results, vehicle/plate crops, and uploaded files.
uploads/	Temporary storage for user-uploaded images.
static/	Static files (docs, UI assets, etc.) served by FastAPI.
requirements.txt	Python dependencies for backend and AI/ML.

🛠️ Tech Stack

◆ API Service

- Python 3.10+
- FastAPI (web framework)
- Uvicorn (ASGI server)
- CORS Middleware

◆ AI/ML Service

- YOLOv8/YOLOv11 (PyTorch)
- OCR (Tesseract, Gemini, or custom)
- OpenCV, NumPy

◆ Database

- SQLite (default, easy dev setup, offline)
- SQLAlchemy (ORM)

◆ Frontend

- React + TypeScript (see `frontend/README.md`)

◆ Utilities

- Logging via Python `logging` module
- File serving via FastAPI

⚡ Setup & Installation

1. Clone the Repository

```
git clone https://github.com/your-org/galaxy-alpr-backend.git
cd galaxy-alpr-backend
```

2. Python Environment

Create and activate a virtual environment:

```
python -m venv venv
source venv/bin/activate   # On Windows: venv\Scripts\activate
```

Install dependencies:

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

3. Directory Preparation

The backend auto-creates `outputs/` , `uploads/` , and `static/` on startup. Create manually if needed:

```
mkdir -p outputs/ uploads/ static/
```

4. Run the API Service

```
uvicorn app:app --reload
```

API will be available at: http://localhost:8000

5. Configuration

- **CORS**: Default allows `localhost:5173` , `localhost:3000` , and all origins (`*`). Change in `app.py` .
- **Database**: Uses SQLite by default. For MySQL, edit connection string in `database.py` .

🔗 Service Communication

From	To	Protocol / Method
Frontend	API	RESTful HTTP (CORS)
API	AI/ML	Internal call or REST

From	To	Protocol / Method
API	Database	SQLAlchemy ORM
AI/ML	Database	Direct read/write
API	File Store	FastAPI static routes

📡 API Overview

🔑 Key Endpoints

Method	Endpoint	Description
POST	`/detect_image`	Upload an image for vehicle/plate detection
GET	`/latest_detection`	Fetch the most recent detection result
GET	`/detections`	List all detection events
GET	`/plate_regions`	Get plate regions in a specific image
GET	`/plate_queries`	Query plate detection history
GET	`/vehicle_queries`	Query vehicle detection history
GET	`/statistics`	Retrieve analytics and stats
GET	`/plate_status`	Check status of a specific plate
GET	`/api/locations`	Manage/query location data
GET	`/outputs/{path}`	Download detection result artifacts
GET	`/static/{path}`	Download static files

🧑‍💻 Development & Contribution

1. Fork and branch from `main`
2. Follow PEP8 and internal code style
3. Add or modify routes in `endpoints/`

4. Include/update tests if available
5. Use clear commit messages and submit PRs

🟢 Code Quality

- Modular endpoint architecture
- Logging for key operations and errors
- Auto-creates required folders on first run

📚 Subfolder Documentation

- `backend/README.md` — API and backend logic
- `backend/endpoints/README.md` — API endpoint documentation
- `backend/PlateDetector.py` — ML model logic and image processing
- `backend/database.py` — Database models and setup

¶ 📄 License

This backend system is developed and maintained by **@GalaxyDeveloper**.

🏷️ Citation

If you use **Galaxy ALPR Backend** in your research, academic paper, or production system, please cite:

```
Galaxy ALPR Backend - Modular Backend for AI-Powered License Plate Recognition
Developed by @GalaxyDeveloper (2025)
Includes FastAPI, YOLOv11n, OCR, and SQLite Integration
```

Galaxy ALPR Backend – Modular, scalable backend for intelligent vehicle and plate detection. *Powered by FastAPI, YOLOv11n, OCR, and SQLite*

Developed by @GalaxyDeveloper — 2025