AutoAid Pro

Technical Documentation

AI-Powered Car Troubleshooter

Capstone project for AI Engineering Bootcamp

Document version	1.0
Prepared by	Ahmad Obeid
Date	February 16, 2026
Technology	Django + DRF + Mini-RAG + Agent + React
Project status	Competition-ready demo build

Purpose: This document describes the system overview, user interface, features, architecture, setup process, and operational guidance for AutoAid Pro.

Table of Contents

Note: This documentation reflects the capstone implementation focused on a stable demo and competition-grade delivery.

1) System Overview

AutoAid Pro is an AI-powered assistant that helps users troubleshoot vehicle issues from natural language symptom descriptions. The platform creates a vehicle profile, opens a diagnostic case, supports short follow-up question loops, and returns a structured diagnosis with recommended safe actions.

The system is built for practical use in a demo setting: it is fast to run locally, easy to explain to judges, and robust against API failures through fallback logic.

- Safety-first troubleshooting for everyday car issues.
- Structured response output: triage level, causes, actions, follow-up questions.
- Mini-RAG knowledge grounding from uploaded documents.
- Agent operations for checklist generation, escalation, and resolution logging.
- Frontend workflow designed for manual demo control.

2) Architecture and Data Flow

The architecture follows an API-first pattern where Django REST endpoints orchestrate core services. Each service has a focused role: diagnosis generation, retrieval, or action automation.

Layer	Main Responsibility
Frontend (React + TS)	Manual forms, chat interface, document upload, agent controls, logs viewer
API Layer (DRF Views)	Validation, request routing, response shaping, endpoint contracts
Domain Models (Core)	Vehicles, cases, symptom reports, diagnosis versions, action logs, notes
LLM Service	Prompt assembly, structured generation, fallback handling, post-safety processing
Mini-RAG Service	Document ingestion, chunk retrieval, context building, citation generation
Agent Service	Operational actions and logs: checklist, escalate, resolve, and reason trace

Data flow for a chat turn

1. User sends message from the chat panel.
2. Backend stores user turn in SymptomReport.
3. Mini-RAG retrieves relevant document chunks (if available).
4. LLM service generates structured diagnosis JSON.
5. Safety rules sanitize risky advice and enforce red-flag escalation.
6. DiagnosisResult is versioned and persisted.
7. Agent service runs auto action and logs actions/notes.
8. API returns final reply with citations and follow-up questions.

3) User Interface Description

The frontend is organized as a guided workflow so judges can watch each stage clearly during demo. The page uses color-coded cards and logical sections to keep the flow linear.

Section	What the user does	Expected output
Create Vehicle	Fill vehicle fields manually (make, model, year, transmission, fuel, mileage).	Vehicle ID is created and shown in status bar.
Create Case	Open a diagnostic case linked to the vehicle, with initial issue and message.	Case ID is created and case snapshot is available.
Upload Knowledge Document	Select a local file and metadata (title, optional make/model/year range).	Document is ingested for retrieval and citation use.
Chat	Describe symptoms and answer follow-up questions.	Assistant reply with triage, causes, actions, and citations.
Agent Controls	Trigger auto, checklist, escalate, or resolve manually.	Action and note logs update in real time.

UI recommendation for final demo: keep dark text on light cards, show status strip on top, and preserve the 1-to-5 section order during live presentation.

4) Functionalities and Features

Core functionality list

- Manual vehicle onboarding through validated fields.
- Case creation with explicit vehicle linkage.
- Conversation-based diagnostics with short follow-up loop.
- Document-grounded responses with source citations.
- Fallback diagnosis when live LLM is unavailable.
- Safety override for high-risk symptoms.
- Agent workflow operations and traceable logs.
- OpenAPI docs for endpoint testing and integration.

Competition-level differentiators

- Clear demonstration path from raw input to explainable output.

- Balanced hybrid intelligence: deterministic safety + generative reasoning.

- Auditability through diagnosis versioning and action traces.

- Resilience through fallback mode and defensive validation.

5) AI Pipeline, Mini-RAG, and Safety Controls

The diagnosis service builds a prompt from vehicle context, recent case history, latest message, and optional retrieval context. The model is asked for structured JSON output which is then validated and post-processed.

Safety control layers

- Red-flag keyword detection (for example: cannot stop, smoke, fuel leak, overheating).
- Forced red triage on dangerous signals regardless of model uncertainty.
- Unsafe action sanitization to avoid risky DIY mechanical instructions.
- Conservative fallback payload when API is missing or over quota.

Mini-RAG behavior: when documents exist, retrieved context is injected and citations are appended. When no documents exist, chat still responds using the base diagnosis logic (normal mode).

6) API Surface Summary

Method	Endpoint	Purpose
GET	/api/health/	Service health check
POST	/api/vehicles/	Create vehicle profile

Method	Endpoint	Purpose
GET	/api/vehicles/{vehicle_id}/	Get vehicle details
POST	/api/cases/	Create diagnostic case
GET	/api/cases/{case_id}/	Get case snapshot
POST	/api/cases/{case_id}/symptoms/	Add symptom entry
POST	/api/chat/	Run diagnosis + retrieval + agent
POST	/api/rag/documents/upload/	Upload knowledge file
POST	/api/cases/{case_id}/agent/run/	Manual or auto agent run
GET	/api/cases/{case_id}/actions/	List agent actions
GET	/api/cases/{case_id}/notes/	List case notes
GET	/api/docs/	Swagger UI

Example chat request payload

```
{
  "case_id": "YOUR_CASE_UUID",
  "message": "When braking, I feel vibration and a hot smell."
}
```

7) How to Start the System

A) Prerequisites

- Python 3.11+ (project tested on 3.13)

- Node.js 18+

- pip and npm available in PATH

- OpenAI API key (for live model responses)

B) Backend startup steps (Windows)

```
cd "C:\Users\VIVOBOOK\Desktop\AutoAid Pro\autoaid-pro"
python -m venv .venv
.venv\Scripts\activate
pip install -r requirements.txt
python manage.py migrate
python manage.py runserver
```

C) Backend .env example

```
DEBUG=True
DJANGO_SECRET_KEY=replace_me
ALLOWED_HOSTS=127.0.0.1,localhost
OPENAI_API_KEY=replace_me
OPENAI_MODEL=gpt-4o-mini
CORS_ALLOW_ALL_ORIGINS=False
```

D) Frontend startup steps

```
cd "C:\Users\VIVOBOOK\Desktop\AutoAid Pro\autoaid-ui"
npm install
npm run dev
```

E) Frontend environment

```
VITE_API_BASE_URL=http://127.0.0.1:8000/api
```

Open browser at the Vite URL, then follow: Create Vehicle -> Create Case -> Upload Document -> Chat -> Agent controls.

8) Demo Walkthrough

1. Create a vehicle with realistic values (make, model, year, mileage).
2. Create a case with a clear symptom title and first user message.
3. Upload one or two local knowledge documents relevant to that vehicle.
4. Send first chat message describing symptoms.
5. Answer one follow-up question from the assistant.
6. Show improved response with updated likely causes/actions.
7. Trigger manual agent actions and display logs.
8. Highlight citations and safety language in final answer.

Demo tip: prepare two scenarios - normal yellow case and emergency red case - to show escalation consistency.

9) Performance, Monitoring, and Hardening

The backend stores operational metrics directly in diagnosis records. Use these fields to build a quick KPI table for your final presentation.

Metric	Where to read it	Why it matters
latency_ms	DiagnosisResult.latency_ms	Responsiveness during live demo
tokens_input / output	DiagnosisResult token fields	Cost and prompt-size control
model_name	DiagnosisResult.model_name	Distinguish live model vs fallback
fallback rate	Count where model_name equals rule_based_fallback	Reliability signal
error logs	Server console / handled exceptions	Hardening and debugging

10) Security and Privacy Practices

- Never commit .env or API keys to Git.
- Use .env.example with placeholder values only.
- Keep virtual environment and node_modules out of repository.
- Restrict CORS and trusted origins in production.

- Rotate keys immediately if exposed.

For capstone demos, avoid sharing personally identifiable vehicle ownership data in screenshots.

11) Testing Checklist

[ ] 1. Vehicle creation returns 201 and valid UUID.

[ ] 2. Case creation links correctly to vehicle_id.

[ ] 3. Document upload accepts local file and returns success.

[ ] 4. Chat returns triage, confidence, causes, actions, and follow-up.

[ ] 5. If documents exist, citations appear in response.

[ ] 6. If LLM fails or quota is exceeded, fallback reply still returns.

[ ] 7. Agent action endpoints produce logs and notes.

[ ] 8. No unhandled server exceptions during full flow.

12) Troubleshooting Guide

400 on /rag/documents/upload/

Check multipart/form-data usage and ensure source_type matches backend enum choices.

Fallback keeps appearing

Verify OPENAI_API_KEY is loaded and billing/quota is available on provider account.

CORS or network errors in frontend

Confirm VITE_API_BASE_URL and CORS_ALLOWED_ORIGINS values are aligned.

Swagger schema issues

Check serializer imports, missing include/path names, and spectacular warnings.

Git rejects commit or pushes secrets

Update .gitignore, untrack files with git rm --cached, rotate exposed keys.

13) Known Limitations and Next Steps

- Keyword-based red flags can miss semantic variations.

- Retrieval quality depends on document quality and chunking strategy.

- Short follow-up loop is intentional for demo simplicity.

- Current build prioritizes local demo over production deployment complexity.

Recommended roadmap

- Add semantic risk classifier in addition to keyword rules.

- Improve retrieval reranking and metadata filtering.

- Add authentication, user roles, and production deployment profile.

- Create dashboard for case analytics and model performance trends.

Appendix A) Recommended Project Structure

```
AutoAid Pro/
  autoaid-pro/              # Django backend
    api/
    core/
    llm/
    rag/
    integrations/
    config/
    manage.py
    requirements.txt
  autoaid-ui/               # React frontend
    src/
      api/
      components/
      App.tsx
    package.json
  docs/
    TECHNICAL_DOCUMENTATION.pdf
    TECHNICAL_DOCUMENTATION.md
```

Appendix B) Environment Template

Backend .env.example

```
DEBUG=True
DJANGO_SECRET_KEY=replace_me
ALLOWED_HOSTS=127.0.0.1,localhost
OPENAI_API_KEY=replace_me
OPENAI_MODEL=gpt-4o-mini
CORS_ALLOW_ALL_ORIGINS=False
```

Frontend .env.example

```
VITE_API_BASE_URL=http://127.0.0.1:8000/api
```

End of document