Support Ticket System

Tech Intern Assessment

Overview

Build a **Support Ticket System** from scratch. Users can submit support tickets, browse and filter them, and view aggregated metrics. The twist: when a ticket is submitted, an **LLM automatically categorizes** it and **suggests a priority level** based on the description — the user can then review and override these suggestions.

You will build the backend, frontend, LLM integration, and containerize everything with Docker.

Time Limit	3 hours from when you receive this
Backend	Django + Django REST Framework + PostgreSQL
Frontend	React
LLM Integration	Any LLM API (OpenAI, Anthropic, Google, etc.) — use your own API key
Infrastructure	Docker + Docker Compose

Backend Requirements

Data Model — Ticket

Field	Type	Constraints
title	CharField	max_length=200, required
description	TextField	required — the user's full problem description
category	CharField with choices	billing, technical, account, general — auto-suggested by LLM, user can override
priority	CharField with choices	low, medium, high, critical — auto-suggested by LLM, user can override
status	CharField with choices	open, in_progress, resolved, closed — defaults to open
created_at	DateTimeField	Auto-set on creation

All constraints must be enforced at the database level.

API Endpoints

Method	Endpoint	Description
POST	/api/tickets/	Create a new ticket. Return 201 on success.
GET	/api/tickets/	List all tickets, newest first. Supports **?category=**, **?priority=**, **?status=**, and **?search=** (searches title + description). All filters can be combined.
PATCH	/api/tickets/<id>/	Update a ticket (e.g. change status, override category/priority).

GET	/api/tickets/stats/	Return aggregated statistics (see below).
POST	/api/tickets/classify/	Send a description, get back LLM-suggested category + priority (see LLM section).

Stats endpoint response format:

```
{
    "total_tickets": 124,
    "open_tickets": 67,
    "avg_tickets_per_day": 8.3,
    "priority_breakdown": {
        "low": 30,  "medium": 52,
        "high": 31,  "critical": 11
    },
    "category_breakdown": {
        "billing": 28,  "technical": 55,
        "account": 22,  "general": 19
    }
}
```

The stats endpoint **must** use database-level aggregation (Django ORM aggregate/annotate), not Python-level loops. This is a key evaluation criterion.

LLM Integration

This is the core differentiator of the assignment. When a user writes a ticket description, the system should call an LLM to **auto-suggest a category and priority** before the ticket is submitted.

How it should work:

- The **/api/tickets/classify/** endpoint accepts a JSON body with a **description** field
- It calls an LLM API with the description and a prompt that asks for a category and priority
- It returns: **{"suggested_category": "...", "suggested_priority": "..."}**
- The frontend calls this endpoint as the user types (or on blur/submit) and pre-fills the category and priority dropdowns with the suggestions
- The user can **accept or override** the suggestions before submitting

Requirements:

- Use any LLM API you prefer (OpenAI, Anthropic, Google, etc.) — use your own API key
- The API key must be configurable via an **environment variable** in docker-compose.yml
- Handle failures gracefully — if the LLM is unreachable or returns garbage, the ticket submission should still work (just without auto-suggestions)
- Include your prompt in the codebase — we will review it

Your choice of LLM, prompt design, and error handling are all part of the evaluation.

Frontend Requirements

1. Submit a Ticket (Form)

- Title input (required, max 200 characters)
- Description textarea (required)
- Category and Priority dropdowns — **pre-filled by the LLM suggestion** after the user enters a description, but fully editable
- Submit button that POSTs to /api/tickets/

- Show a loading state while the LLM classify call is in progress
- Clear the form on success and show the new ticket without a full page reload

2. Ticket List

- Display all tickets, newest first
- Each ticket shows: title, description (truncated), category, priority, status, timestamp
- Filter by category, priority, and status
- Search bar that filters by title and description via ?search=
- Clicking a ticket allows changing its status (e.g. open to in_progress to resolved)

3. Stats Dashboard

- Fetch and display data from /api/tickets/stats/
- Show: total tickets, open count, avg per day, priority + category breakdowns
- Auto-refresh when a new ticket is submitted

> *Styling is secondary — focus on functionality first. Use any CSS approach you prefer.*

Docker Requirements

Your project must be fully containerized. A reviewer should be able to run your entire application with a single command:

```
docker-compose up --build
```

Your Docker setup must include:

- A **PostgreSQL** database service
- A **Django backend** service that runs migrations automatically on startup
- A **React frontend** service
- Proper service dependencies so everything starts in the right order
- The LLM API key passed as an **environment variable** (not hardcoded)

> *The app should be fully functional after docker-compose up (except the LLM feature, which depends on a valid API key being provided).*

Evaluation Criteria

Area	What We Look For	Weight
Does it work?	docker-compose up runs and the app works end-to-end	20%
LLM integration	Classify endpoint works, prompt quality, graceful error handling, good UX for suggestions	20%
Data modeling	Correct field types, constraints, choices enforced at DB level	10%
API design	Clean endpoints, proper status codes, working filters + search	10%
Query logic	DB-level aggregation in stats endpoint (not Python loops)	10%
React structure	Component organization, state management, API integration	10%
Code quality	Readable, consistent, no dead code or leftover debug prints	10%
Commit history	Incremental, meaningful commits showing your thought process	5%
README	Setup instructions, which LLM you chose and why, design decisions	5%

Submission

Zip your entire project folder **including the .git directory** (so we can review your commit history) and upload it via the Google Form shared with you before the deadline.

Your zip must include:

- All source code (Django backend + React frontend)
- A working **docker-compose.yml** and **Dockerfile(s)**
- A **README.md** with: setup instructions, which LLM you used and why, and any design decisions
- **The .git folder** — we will review your incremental commit history

> *Do **not** push your code to a public repository.*

Rules

- You may use any documentation or learning resources
- Your commit history should reflect your own development process
- The entire app must work with a single **docker-compose up --build** command
- No additional manual setup steps should be required (aside from providing an API key)
- Do **not** hardcode your API key — use environment variables

---

Good luck.