Sprint Review Meeting

Network Ticketing System

Date: 27/10/25

Attendees:

- **Sudhan** – Product Owner
- **Hamid** – Scrum Master
- **Riya** – Developer
- **Parvani** – Developer

Purpose of the Meeting

The team convened to align on the scope, architecture, and development responsibilities for the new **Ticketing System** platform. This system is designed to streamline issue reporting, assignment, and resolution across multiple user roles including customers, engineers, agents, managers, and admins.

Product Owner Brief

Sudhan outlined the core objectives and business requirements of the project:

- Build a scalable ticketing platform for telecom support workflows.
- Support role-based access: customers raise tickets, engineers/agents resolve them, managers oversee, and admins manage users.
- Enable SLA tracking, issue categorization, and assignment auditability.
- Ensure extensibility for future analytics, provider integration, and admin controls.
- Prioritize usability and clarity across both customer and admin interfaces.

Scrum Master Facilitation

Hamid organized and facilitated the meeting, ensuring:

- Clear sprint goals and task breakdowns.

- Alignment on backend and frontend responsibilities.
- Timelines for API scaffolding, frontend routing, and role-based flows.
- Coordination of ER diagram review and architecture validation.
- Action items were documented and assigned for the next sprint.

Development Contributions

Riya and Parvani collaborated on both backend and frontend implementation:

Backend APIs:

- User registration and role-based login.
- Ticket creation with issue category and SLA validation.
- Admin-only user listing and assignment logic.
- Dynamic issue category retrieval.
- Structured error handling and rollback safety.

Frontend:

- Role-based dashboards for admin and customer.
- Ticket creation form with dynamic category dropdown.
- User list view for admin with filtering.
- Routing setup and protected navigation using React Router.
- Bootstrap integration for responsive UI components.

Sprint 1: Core Foundations

Project Setup

- Initialize backend using **Python (FastAPI)**.
- Set up **React frontend**.
- Connect to **MySQL** .
- Establish **Git repository** and define folder structure.

Authentication Module

- Implement login/signup functionality with mocked roles:
  - Customer
  - Agent
  - Engineer
  - Manager
- Use **JWT** or **session-based authentication**.
- Enable basic role-based routing post-login.

User Role Dashboard Layouts

- Display different sidebars/menus based on user roles:
  - **Customer**: Ticket submission interface.
  - **Agent**: Ticket queue view.
  - **Engineer**: Assigned tickets dashboard.

Ticket Entity Definition

- Design and implement the **Ticket** schema with fields:
  - Status
  - Timestamps
  - Customer ID
  - Issue type
  - Severity
  - Priority
  - AssignedTo
- Use ORM tools like **Hibernate** or **SQLAlchemy**.

Base REST APIs

- Create foundational endpoints:
  - `POST /tickets`
  - `GET /tickets/{id}`
  - `GET /tickets?assignedTo=`
- Implement basic **CRUD operations**.

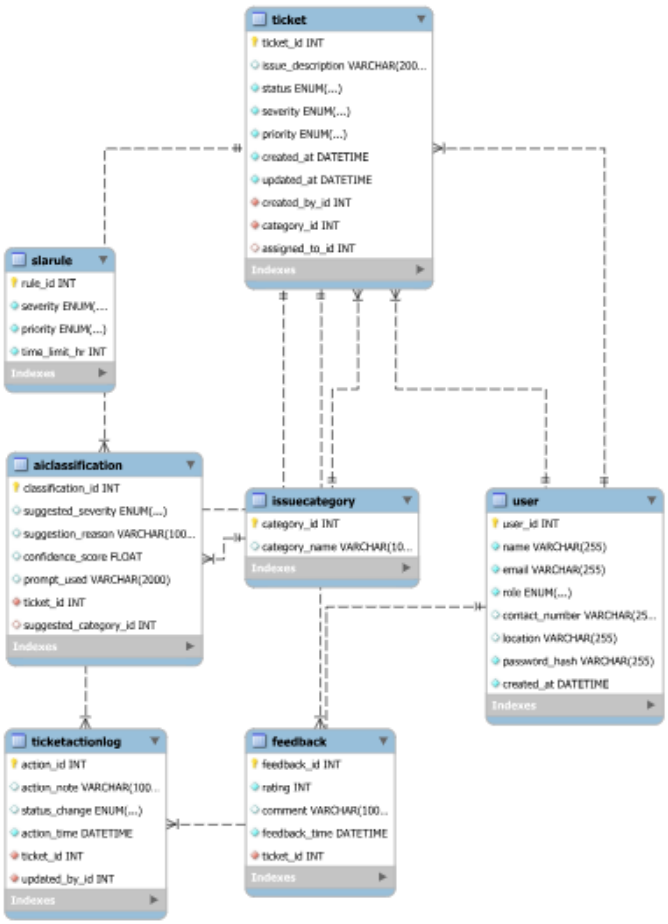
Architecture Overview

An ER diagram was reviewed to validate the database schema and relationships. Key entities include:

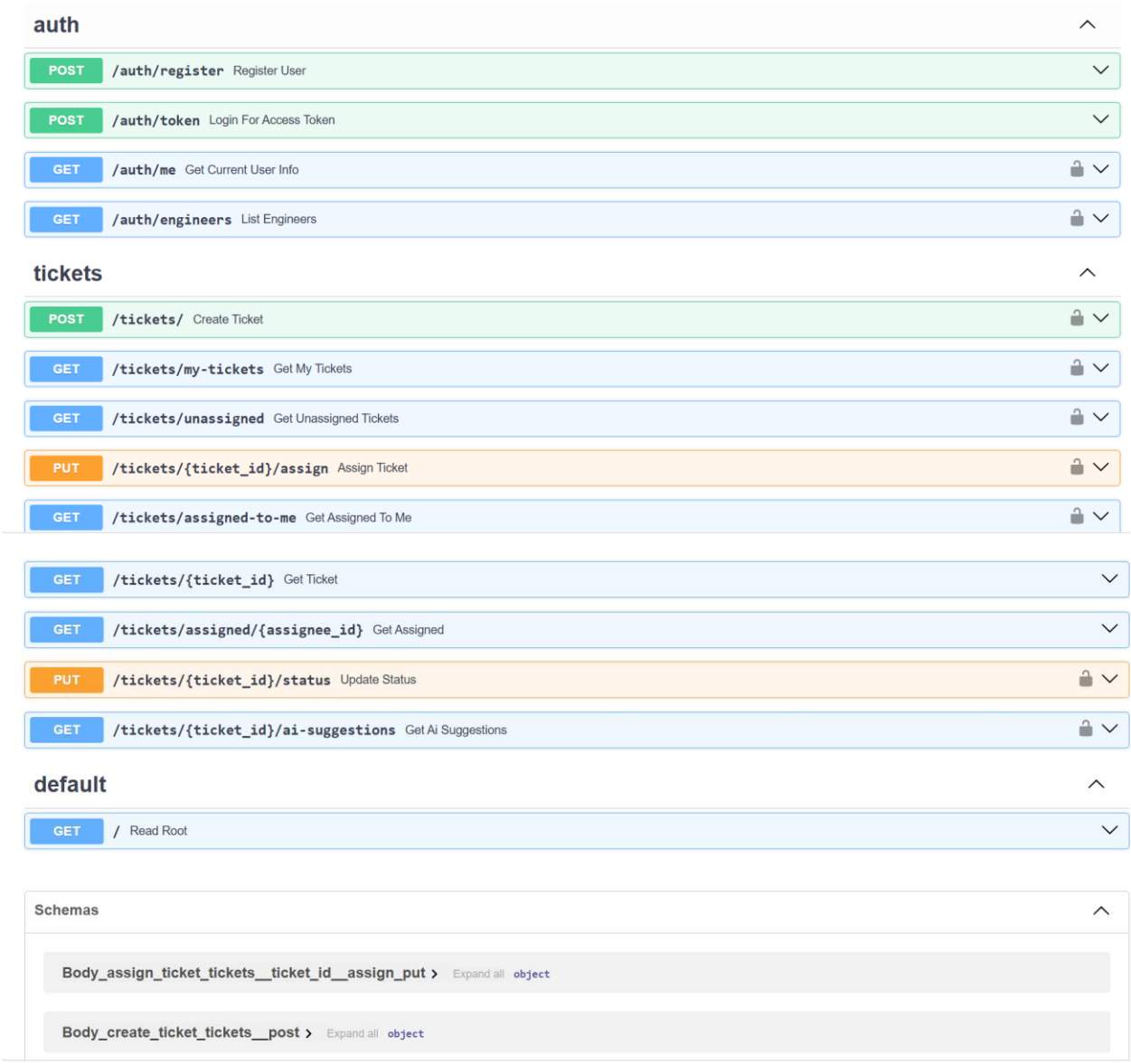
- **User**: role, contact, location
- **Ticket**: description, category, SLA, status, assignee
- **IssueCategory**: category_id, category_name

The architecture supports modular expansion, auditability, and analytics integration.

Following is the ER diagram for the same



API ENDPOINTS



Sprint 2 Plan: Ticket Creation Flow

Vertical Slice – Customer → Ticket Creation

The focus for Sprint 2 is to build the first complete, end-to-end feature: the **Customer Ticket Creation Flow**.

Sprint Goals

Customer Ticket Form UI

- Build the React component for the "Create New Ticket" form.
- Include fields for:
    - Issue type selection.
    - Detailed description.
    - Optional attachments (e.g., screenshots).
    - Preferred contact time.

Backend API Integration

- Connect the frontend form to the existing `POST /tickets` API.
- Ensure successful submission returns:
    - New Ticket ID.
    - Initial status (e.g., "New").

Auto-Categorization with Gen AI

- Integrate a Generative AI service (e.g., Gemini API).
- Use ticket description to suggest severity and category.
- Display suggestions to agent/manager for override (not shown to customer immediately).

Notification Trigger (Mock)

- Implement a mock service to simulate SMS/email notification:
    - Example: “Ticket #123 has been created.”

Initial Status and SLA Timer

- Store new tickets with status set to "New".
- Capture creation timestamp to start SLA timer.