**IT Support System – Low-Level Design (LLD)**

This document presents the Low-Level Design (LLD) for an IT Support System developed using Python (OOP) and SQL (SQLite/MySQL). The system is designed to streamline reporting, tracking, and resolving IT issues efficiently while ensuring SLA compliance, managing IT assets, and improving team communication through notifications. Its modular architecture encompasses ticket management, SLA enforcement, knowledge base management, asset tracking, and notification handling, thus enabling scalability, security, and maintainability.

**1. Introduction**

The IT Support System provides a structured approach to reporting, tracking, and resolving IT-related issues.

Goals:

* Reduce IT downtime.
* Ensure SLA compliance.
* Provide searchable knowledge base.
* Manage IT assets lifecycle.
* Improve communication through notifications.

**2. Module Overview**

* Ticket Management
  + Create, update, assign, and close tickets.
  + Core of the system where users can report IT issues.
  + Tickets go through lifecycle stages: Open → In Progress → Resolved → Closed.
  + Admins/Agents can assign tickets and track status.
* SLA Management
  + Define and enforce resolution deadlines.
  + Ensures issues are resolved within agreed timelines (e.g., critical = 4 hrs).
  + Escalates tickets if deadlines are breached.
  + Maintains response & resolution benchmarks for different priority levels.
* Knowledge Base
  + Repository of common solutions.
  + Repository of articles, FAQs, and solutions for common problems.
  + Helps users self-serve before creating a ticket.
  + Continuously updated by support staff.
* Asset Management
  + Track and assign IT assets.
  + Tracks IT hardware and software assets (like laptops, routers, licenses).
  + Maintains lifecycle: Available → Assigned → Retired.
  + Links assets to tickets when issues are reported.
* Notification System
  + Alerts and reminders for updates & SLA breaches.
  + Sends alerts/reminders about ticket updates and SLA breaches.
  + Notifications can be via Email, SMS, or system alerts.
  + Improves communication between employees and IT team.

**3. Class Design**

3.1 Class: User

Represents employees, support agents, and admins.

Attributes:

* user\_id
* name
* email
* role (Employee, Agent, Admin)
* status

Methods

* create\_user
* update\_user
* deactivate\_user
* list\_users

3.2 Class: Ticket

Handles lifecycle of IT support tickets.

Attributes

* ticket\_id
* created\_by (User)
* assigned\_to (Agent)
* subject
* description
* priority
* status (Open, In Progress, Resolved, Closed)
* created\_at, updated\_at

Methods

* create\_ticket
* update\_ticket
* assign\_ticket
* close\_ticket
* get\_ticket\_status
* list\_tickets

3.3 Class: SLA

Manages Service Level Agreements for tickets.

Attributes

* sla\_id
* priority (Low, Medium, High, Critical)
* response\_time (hrs)
* resolution\_time (hrs)

Methods

* define\_sla
* link\_sla\_to\_ticket
* check\_sla\_compliance
* escalate\_ticket
* list\_sla\_policies

3.4 Class: KnowledgeBase

Stores articles and solutions to common problems.

Attributes

* kb\_id
* title
* content
* category
* created\_by (User)

Methods

* add\_article
* update\_article
* delete\_article
* search\_article
* list\_articles

3.5 Class: Asset

Tracks IT hardware/software assigned to users.

Attributes

* asset\_id
* name
* type (Laptop, Software, Server, etc.)
* status (Available, Assigned, Retired)
* assigned\_to (User)

Methods

* add\_asset
* update\_asset
* assign\_asset
* release\_asset
* list\_assets
* track\_asset\_lifecycle

3.6 Class: Notification

Handles alerts and reminders.

Attributes:

* notification\_id
* user (User)
* message
* type (Email, SMS, System Alert)
* status (Unread, Read)
* created\_at
* Methods
* create\_notification
* send\_notification
* list\_notifications
* mark\_as\_read

4. Database Design

Tables

* User(User\_ID, Name, Email, Role, Status)
* Ticket(Ticket\_ID, User\_ID, Agent\_ID, Subject, Description, Priority, Status, Dates)
* SLA(SLA\_ID, Priority, ResponseTime, ResolutionTime)
* KnowledgeBase(KB\_ID, Title, Content, Category, CreatedBy)
* Asset(Asset\_ID, Name, Type, Status, AssignedUserID)
* Notification(Notification\_ID, User\_ID, Message, Type, Status, Date)
* Comment(Comment\_ID, Ticket\_ID, User\_ID, Text, Date)

5. Non-Functional Requirements (NFR)

* Performance → Ticket creation in < 1s.
* Scalability → Supports small team, extendable.
* Reliability → DB constraints enforce consistency.
* Security → Role-based access, password hashing.
* Maintainability → OOP-based modular design.
* Usability → Simple CLI/GUI, searchable KB.