

System Design Document

L1 Customer Support Ticketing & Chat System (L1-CSTS)

Document Type: System Design Document

Architecture Role: Principal Architect

Version: 1.0

Date: 01-Jan-2026

Project Duration: 30 Days

1. Introduction

1.1 Purpose

This document defines the technical architecture, system components, data flow, and design decisions for the L1 Customer Support Ticketing & Chat System.

It acts as a blueprint for:

- Backend developers
- Frontend developers
- Testers

1.2 Scope

The system provides:

- Ticket creation with screenshots
- Real-time chat per ticket
- Agent dashboard for ticket handling
- Secure authentication and role-based access
- Real time AI assistant

2. Architectural Overview

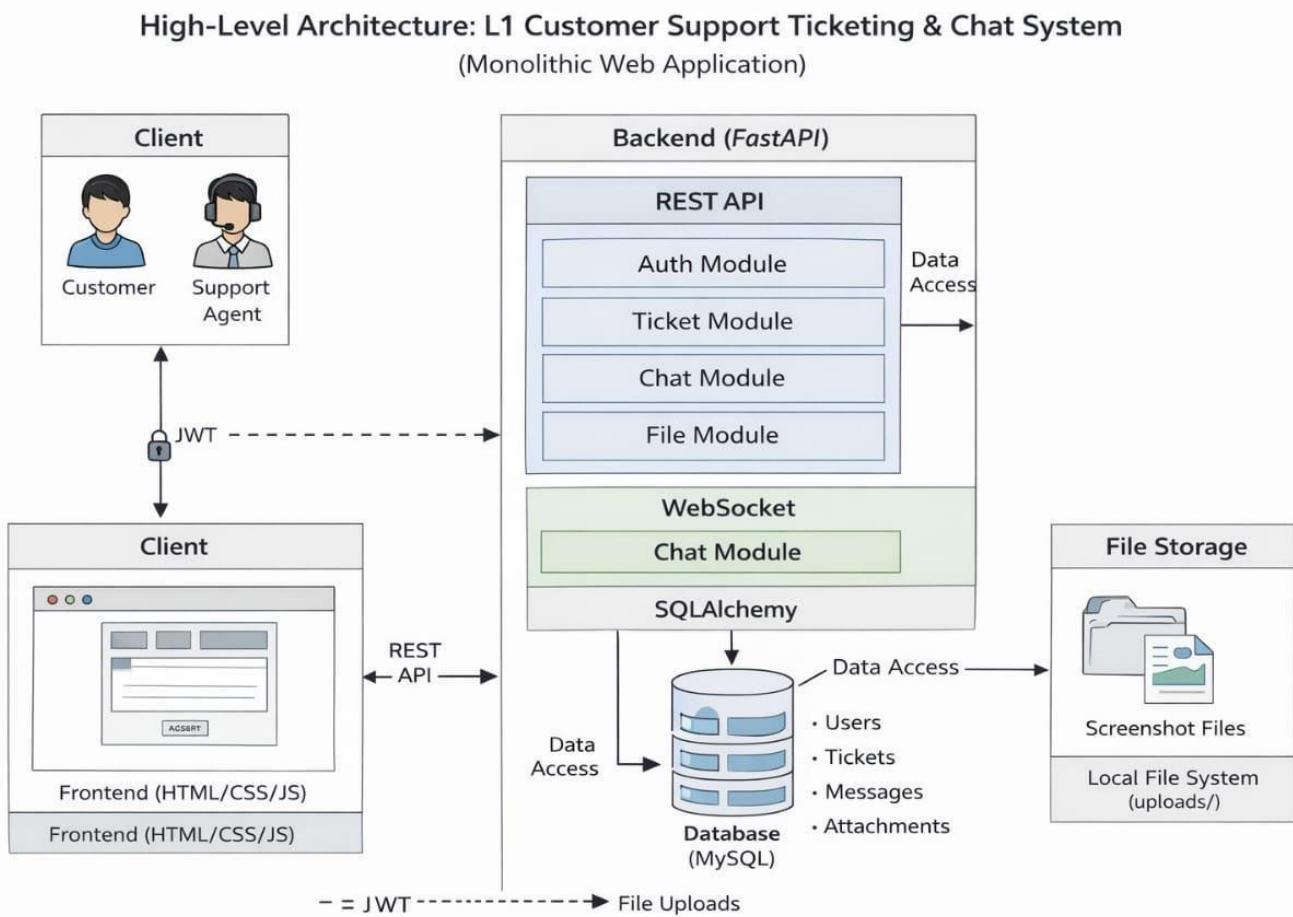
2.1 Architecture Style

Monolithic Web Application with Real-Time Communication

Reason:

- Small scope
- 30-day timeline
- Easy to develop, deploy, and debug

2.2 High-Level Architecture Diagram



3. Technology Stack

3.1 Backend

- Language: Python
- Framework: FastAPI
- Real-Time: WebSockets
- Authentication: JWT / Session-based
- ORM: SQLAlchemy

3.2 Frontend

- HTML
- CSS
- JavaScript
- WebSocket client for chat

3.3 Database

- MySQL

3.4 File Storage

- Local file system (uploads folder)
- Images only (png, jpg)
- Pdf files(.pdf)

4. System Components Design

4.1 Frontend Layer

Customer UI Components

- Login / Signup Page
- Create Ticket Page
- Ticket Detail Page (Chat + Status)

Agent UI Components

- Agent Login Page
- Dashboard (Ticket List)
- Ticket Detail Page
- Chat Window
- Status Update Controls

4.2 Backend Layer (FastAPI)

- Major Modules

Module	Responsibility
Auth Module	Login, JWT, role validation
User Module	Customer & Agent data
Ticket Module	Ticket(create,read) & status update
Chat Module	WebSocket handling
File Module	Screenshot upload & validation

4.2.1 Authentication Flow

- User logs in
- JWT token generated
- Token sent with each request
- Role-based access enforced

4.2.2 Ticket Creation Flow

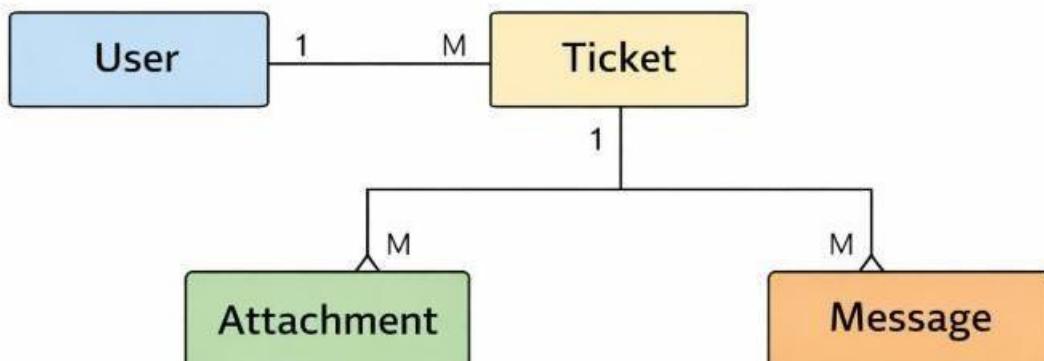
- Customer submits ticket form
- Backend validates inputs
- Screenshots stored
- Ticket ID generated
- Chat room created
- Status set to Open

4.2.3 Real-Time Chat Flow

- One WebSocket connection per ticket
- Messages saved to database
- Both customer & agent receive updates instantly

5. Database Design

5.1 Entity Relationship Overview



5.2 Tables Design

- User Table

Column	Type
Id	INT (PK)
Name	VARCHAR
Email	VARCHAR
password_hash	VARCHAR
Role	VARCHAR

- Agent Table

Column	Type
Id	INT (PK)
username	VARCHAR
password	VARCHAR
Role	VARCHAR

- Agent refresh token table

Column	Type
Id	INT (PK)
Agent_id	FK
Token	VARCHAR
Created_at	TIMESTAMP

- User refresh token table

Column	Type
Id	INT (PK)
User_id	FK
Token	VARCHAR
Created_at	TIMESTAMP

- Ticket Table

Column	Type
Id	INT (PK)
Ticket_no	VARCHAR
User_id	INT
Issue_type	VARCHAR
subject	VARCHAR
description	TEXT
Callback_number	INT
status	VARCHAR
priority	VARCHAR
created_at	TIMESTAMP
Agent_id	FK

- Message Table

Column	Type
Id	INT (PK)
ticket_id	FK
Ticket_no	INT
Sender	VARCHAR

content	TEXT
timestamp	TIMESTAMP

- Attachment Table

Column	Type
Id	INT (PK)
ticket_id	FK
file_path	VARCHAR
file_type	VARCHAR
Uploaded_at	TIMESTAMP

6. API Design (High-Level)

6.1 REST APIs

Method	Endpoint	Description
POST	/user/login	Login
POST	/user/register	Signup
POST	/agent/login	Agent login
POST	/user/createticket	Create ticket
GET	/tickets	List tickets
GET	/ticket/{id}	Ticket details
PUT	/tickets/{id}/status	Update status

6.2 WebSocket Endpoint

- /ws/tickets/{ticket_id}

Purpose:

- Real-time chat
- Typing indicator
- Online status

7. Security Design

7.1 Authentication

- JWT-based authentication
- Token expiry handling
- Refresh token

7.2 Authorization

- Customer - only own tickets
- Agent - all tickets

7.3 Input Protection

- XSS prevention (sanitize chat inputs)
- File type validation
- File size limit (≤ 4 MB)

8. Performance & Scalability

Performance Targets

- Ticket list load ≤ 3 seconds
- Chat latency ≤ 2 seconds

Scalability

- Designed for low to medium traffic

9. Error Handling & Logging

Error Handling

- User-friendly error messages
- HTTP status codes used properly

Logging

- API errors
- WebSocket connection events
- File upload failures

10. Deployment Architecture

Local Deployment

- FastAPI with Standard
- MySQL
- Local file storage

11. Risks & Architectural Decisions

Risk	Decision
Time constraint	Monolithic architecture
Chat complexity	WebSocket only
File handling	Local storage

12. AI Chatbot (Out of scope)

- Built using FastAPI (Python framework) for handling AI chat APIs.
- Implemented rule-based L1 support to instantly answer common customer issues.
- Integrated Groq LLM (LLaMA model) for AI responses when rules don't match.
- Designed a confidence score and escalation flag to route complex queries to human agents.
- Exposed the chatbot as a REST API endpoint for easy frontend integration.

13. Future Enhancements (Not in Scope)

- SLA tracking
- Multi-agent assignment
- Email notifications
- Analytics dashboard
- Chatbot integration

14. Conclusion

This architecture:

- Meets all business and functional requirements
- Is feasible within 30 days
- Is easy to understand and implement
- Demonstrates real-world IT architecture principles

15. Wireframe diagram - L1 Customer Support Ticketing & Chat System (L1-CSTS)

