IBM – SUMMER INTERNSHIP REPORT

FULL STACK PROJECT REPORT ON

“CUSTOMER SUPPORT TICKETING SYSTEM - HELPDESK”

Submitted in partial fulfillment of the

Summer Internship Program at IBM (2025)

Submitted By:

Name: Mahek Gandhi

Enrollment No: 220450131007

Degree: B.E. Computer Engineering

College: Shri S’ad Vidya Mandal Inatitute Of Technology, Bharuch

Guided By:

Mr. Jaivik Panchal

Submission Date: July 2025

Certificate:

Acknowledgment:

I would like to express my sincere gratitude to **IBM** for providing me with the opportunity to undertake this summer internship and work on the project titled **"Customer Support Ticketing System- Helpdesk"** This internship has been a highly enriching experience and has contributed significantly to my practical knowledge and skills in full stack development.

I extend my heartfelt thanks to my project mentor **Mr. Jaivik Panchal** for their continuous guidance, valuable feedback, and support throughout the internship. Their insights and encouragement have been instrumental in the successful completion of this project.

I am also thankful to the entire IBM team and colleagues who made my internship experience smooth, collaborative, and intellectually stimulating.

I would also like to thank my college, Shri S'ad Vidya Mandal Inatitute Of Technology, Bharuch , and the Department of Computer Engineering for their constant support and encouragement.

Lastly, I am deeply grateful to my family and friends for their unwavering support and motivation throughout this journey.

CONTENT

1. Abstract

The **Customer Support Ticketing System** is a full stack web application developed as part of my summer internship at **IBM**. The primary goal of the project is to streamline and enhance the customer support process by enabling efficient ticket management between customers, support agents, and administrators.

This system allows customers to raise support tickets categorized by issue type, while support agents can view, update, and resolve tickets assigned to them. The admin has centralized control over users, categories, and system settings. The system is designed with scalability, security, and ease of use in mind, offering a seamless and responsive user interface.

• What the Project Is:

This project is a **role-based ticket management platform** that allows:

- Customers to raise support tickets,
- Support agents to view and resolve tickets,
- Admins to manage users, categories, and overall system operations.

The application provides a centralized system for handling customer queries, tracking issue status, and improving response efficiency.

• Why It Was Developed:

The system was developed:

- To understand the workflow of real-world customer support platforms.
- To gain hands-on experience in **full stack development** using core **JavaScript** for the frontend and **Node.js, Express.js, MongoDB** for the backend.
- To build a scalable, maintainable, and secure solution that addresses common support-related challenges.

• Key Features:

- Role-based access: Customer, Support Agent, and Admin
- Ticket creation, status tracking (Open, In Progress, Resolved)
- Authentication and password encryption
- Category-wise ticket filtering
- Admin dashboard and control panel
- RESTful API integration
- User-friendly interface built with HTML, CSS, and JavaScript

2. Objective

The objective of this project is to develop a **full stack, web-based Customer Support Ticketing System** that facilitates structured, efficient, and transparent communication between customers, support agents, and administrators. The system is intended to **simulate a real-world customer service environment** where users can report issues, track their resolution progress, and receive timely support, while support staff and administrators can manage and monitor service tickets with ease.

Primary Purpose:

To **automate and optimize the process of handling customer queries** by creating a centralized system that enables:

- Customers to create and manage support tickets.
- Support agents to view, update, and resolve assigned tickets.
- Admins to oversee all operations including user roles, ticket categories, and platform settings.

This project aims to **reduce delays in issue resolution, enhance service quality, and improve accountability** in support operations.

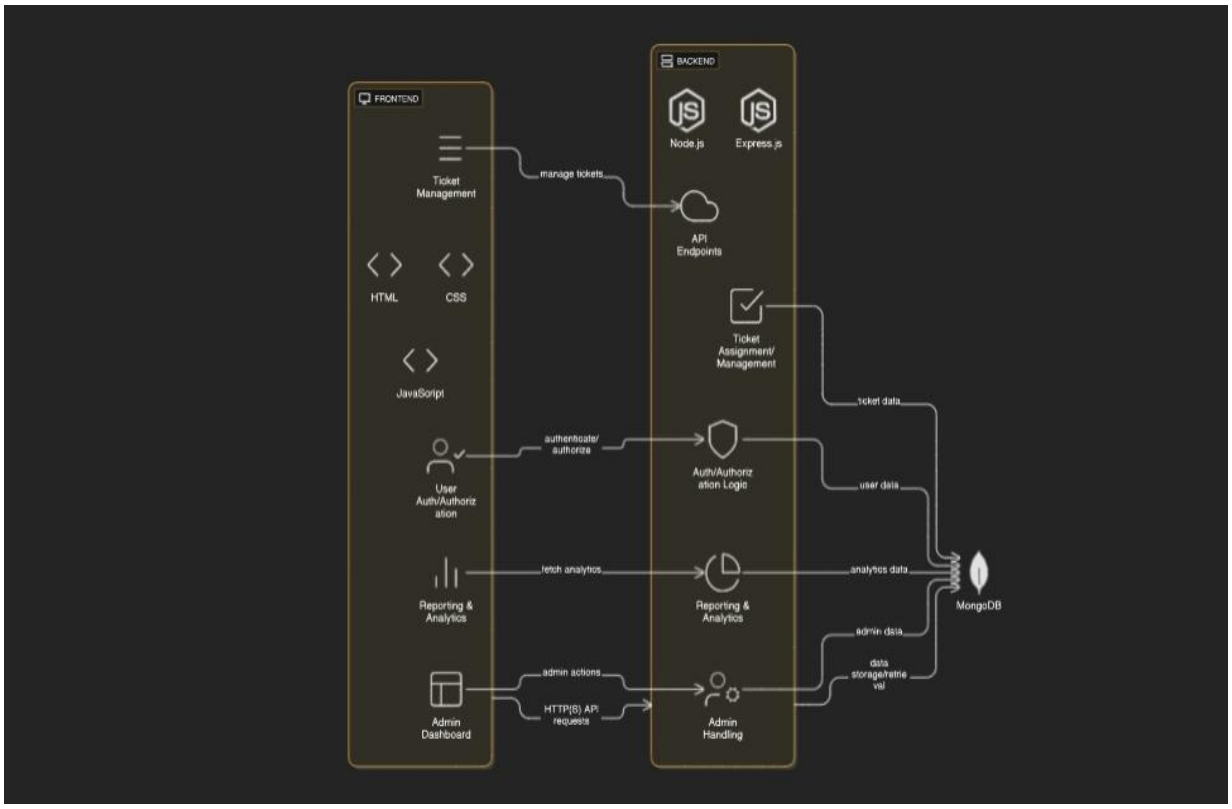
Specific Goals:

1. **Design a responsive, user-friendly frontend interface** using HTML, CSS, and JavaScript to ensure intuitive navigation for all user roles.
2. **Develop a secure and scalable backend** using Node.js and Express.js to handle authentication, routing, and business logic efficiently.
3. **Integrate MongoDB** as the primary database to store structured data such as user profiles, tickets, categories, and internal comments.
4. **Implement Role-Based Access Control (RBAC)** to restrict features and views based on user types — Customer, Support Agent, and Admin.
5. **Enable JWT-based authentication and password encryption** to maintain user security and data integrity.
6. **Build RESTful APIs** for seamless communication between the frontend and backend systems.
7. **Provide ticket categorization and status management features**, allowing for proper filtering and tracking of support issues.
8. **Create an Admin dashboard** to monitor system activity, manage users, and perform administrative tasks.
9. **Ensure modularity and clean code structure** to support future enhancements, new feature integration, and scalability.
10. **Deploy the application locally or on a cloud-based platform** for testing and demonstration purposes.

This project not only fulfills the functional needs of a support ticketing platform but also serves as a strong foundation for understanding full stack development workflows, backend system architecture, database modeling, and secure web practices — all within a real-time problem-solving context.

3. System Architecture

1. **Architecture diagram**



2. Component Interaction and Technology Flow

Frontend (Presentation Layer):

- Built using **HTML, CSS, and Vanilla JavaScript**.
- Responsible for presenting the UI to the users based on their roles: **Customer**, **Support Agent**, and **Admin**.
- Interacts with the backend via **AJAX/Fetch API** using **JSON data exchange**.
- Performs client-side validation before sending data to the server.

Backend (Application Layer):

- Built using **Node.js** with **Express.js** framework.
- Hosts **RESTful APIs** for managing:
  - User Authentication (Login/Register)
  - Ticket Operations (Create, Update, View, Delete)
  - Comment System
  - Category and Role Management
- Secures endpoints using **JWT-based Authentication**.
- Applies **Role-Based Access Control (RBAC)** to ensure restricted access based on user type.

Database (Data Layer):

- Uses **MongoDB**, a NoSQL database that stores data in **JSON-like documents**.
- Collections:
  - **Users**: Stores user credentials, roles, and basic info.
  - **Tickets**: Stores support ticket details like title, description, category, status.
  - **Comments**: Stores internal ticket communications.
  - **Categories**: Lists predefined ticket categories (e.g., Technical, Billing).
- MongoDB offers flexible schemas, fast reads/writes, and horizontal scalability.

4. Technology Stack

The project "Customer Support Ticketing System" was built using a modern full stack technology architecture. Below is a detailed description of each component used:

Frontend

- **HTML5**: The structure of all web pages was built using HTML5, ensuring semantic markup and compatibility with modern browsers.
- **CSS3**: Used for styling and layout, CSS3 enabled a responsive, visually appealing, and user-friendly interface across various devices.
- **JavaScript**: Core client-side scripting language that handled dynamic interactions, DOM manipulation, form validations, and AJAX-based communication with the backend APIs.

Backend

- **Node.js**: A runtime environment that allows JavaScript to run on the server side. It was used to build a scalable and high-performance backend.
- **Express.js**: A minimal and flexible web application framework for Node.js. It was used to handle routing, RESTful API creation, middleware, and backend logic efficiently.

Database

- **MongoDB**: A NoSQL database used to store structured data in the form of collections and documents. It provided flexibility in managing unstructured data like tickets, users, comments, and categories. It also ensured high performance, scalability, and fast read/write operations.

Tools & Utilities

- **VS Code**: Visual Studio Code was used as the primary code editor due to its rich extension support, Git integration, and debugging features.
- **Git & GitHub**: Used for version control and code repository management. GitHub also enabled seamless collaboration, backup, and code sharing.
- **Postman**: Used for testing RESTful APIs during backend development. It helped in sending HTTP requests, analyzing responses, and debugging endpoints.
- **MongoDB Compass**: GUI tool to visually explore MongoDB data, manage collections, and monitor performance.

5. Modules/ Features

1. Authentication Module

- **Description**:
  Handles user registration, login, and access control. Ensures that only authenticated users can access role-specific features (Customer, Support Agent, Admin).
- **Key Features**:
  - Signup and login forms
  - Password encryption using `bcrypt.js`
  - Role-based access control
  - Token-based session handling with `JWT`
- **Technologies Used**:
  HTML, CSS, JavaScript, Node.js, Express.js, MongoDB.

2. Ticket Management Module

- **Description**:
  Allows customers to raise support tickets. Support agents can view, update status, and add internal comments. Admin can monitor all tickets across categories.
- **Key Features**:
  - Ticket creation with category and description
  - View tickets by status (Open, In Progress, Resolved)
  - Status update by agents
  - Internal comments thread
- **Technologies Used**:
  JavaScript, Node.js, Express, MongoDB, HTML/CSS

3. Dashboard Module

- **Description**:
  Displays personalized dashboards based on roles:
  - **Customers** see their raised tickets and statuses.
  - **Agents** see tickets assigned to them.
  - **Admins** see system-wide statistics and analytics.
- **Key Features**:
  - Role-based dashboard views
  - Ticket filtering and summaries
  - Quick actions and recent activity feed
- **Technologies Used**:
  HTML, CSS, JavaScript, Node.js, MongoDB

4. Admin Panel

- **Description**:
  The Admin has full control over the system, including managing users, categories, and overall ticket flow.
- **Key Features**:
  - View all users and assign roles

    - Manage ticket categories (Add/Edit/Delete)
    - Monitor all support activity
- **Technologies Used**:
  JavaScript, Node.js, Express, MongoDB, HTML/CSS

5. Category Management Module

- **Description**:
  Enables Admin to manage ticket categories, making it easier to organize and route customer issues.
- **Key Features**:
    - Add/Edit/Delete categories
    - Assign category to each ticket during creation
    - Filter tickets by category
- **Technologies Used**:
  HTML, CSS, JavaScript, Node.js, MongoDB

6. Comments & Communication Module

- **Description**:
  Internal comment system for communication between agents and admins regarding ticket progress.
- **Key Features**:
    - Add comments to any ticket
    - View comment history
    - Maintain timeline of issue handling
- **Technologies Used**:
  JavaScript, Node.js, Express, MongoDB

6. Frontend Development

Technologies Used

- **HTML5**: Used to build the structure and content of each page.
- **CSS3**: Applied for styling, layouts, responsiveness, and animations.
- **JavaScript**: Enables interactivity, DOM manipulation, client-side form validation, and dynamic content loading.

Page Structure / Layout

The project follows a modular and reusable layout strategy to maintain consistency across pages:

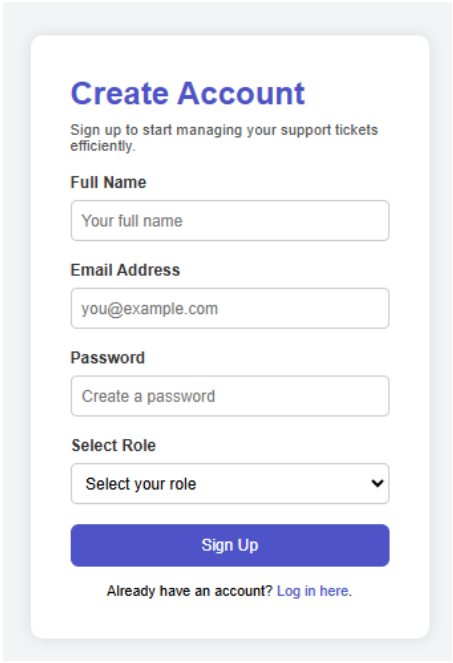
- **Header/Navbar**: Includes logo, navigation links, and logout option.
- **Sidebar (Role-Based)**:
    - For Agents: Links to "Assigned Tickets", "Update Status", etc.
    - For Customers: Links to "Raise Ticket", "My Tickets"
    - For Admins: Links to "All Tickets", "User Management", "Categories"
- **Main Content Area**: Dynamic section that updates based on the selected page or route.
- **Footer**: Contains copyright.

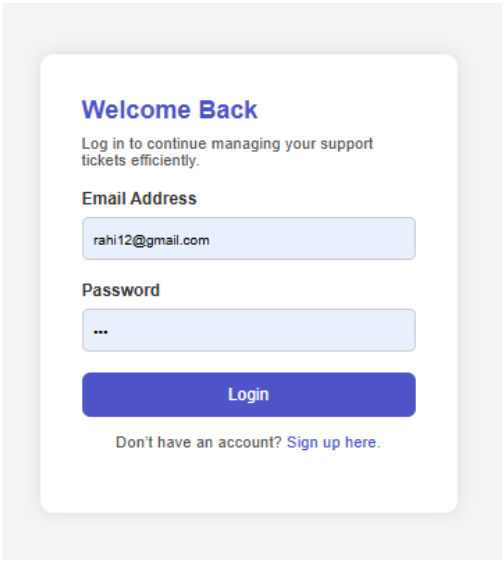
UI/UX Design Strategy

- **Responsiveness**: Ensured through media queries and flexible layouts for mobile, tablet, and desktop views.
- **Clean and Minimal UI**: Focused on intuitive navigation and clutter-free layout for different users.
- **Role-Based Interface**: Each user type sees only the relevant features, improving usability.
- **Feedback Mechanisms**: Success/failure alerts, loaders, and validation messages guide users effectively.
- **Accessibility Considerations**: Proper contrast, label associations, and button sizes for better accessibility.

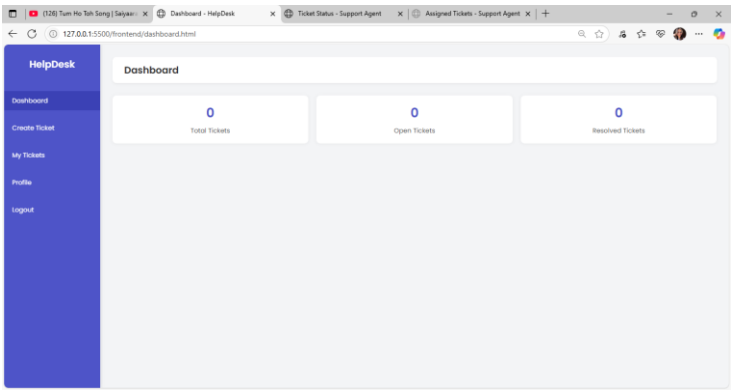
Sample Pages

1. **Login & signup page**
    - Fields: Email, Password
    - Action: Role-based redirection after login

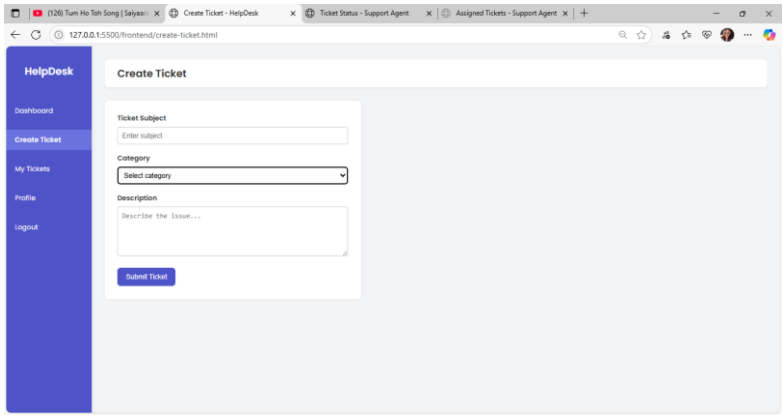




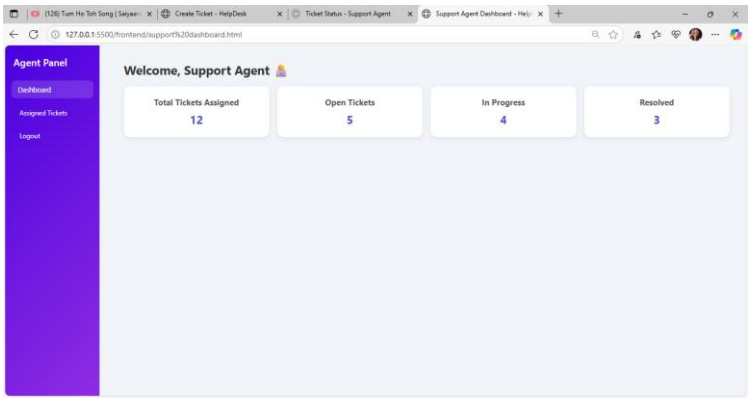
2. **Customer Dashboard**
   - Overview of submitted tickets
   - Button to raise new ticket



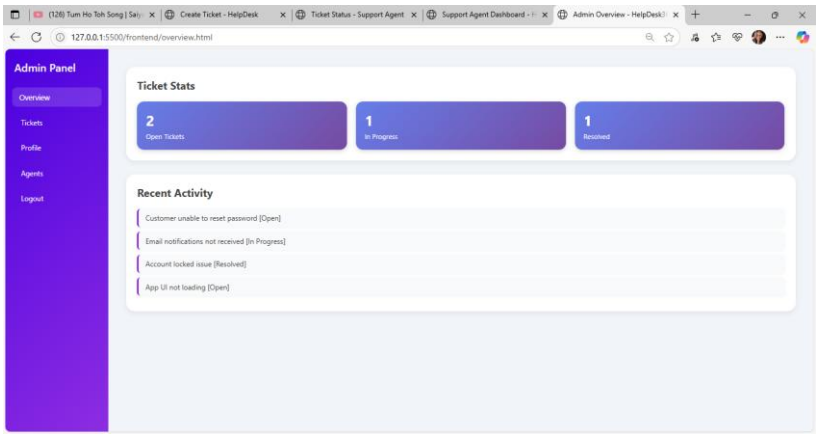
3. **Raise Ticket Form**
   - Category, Subject, Description

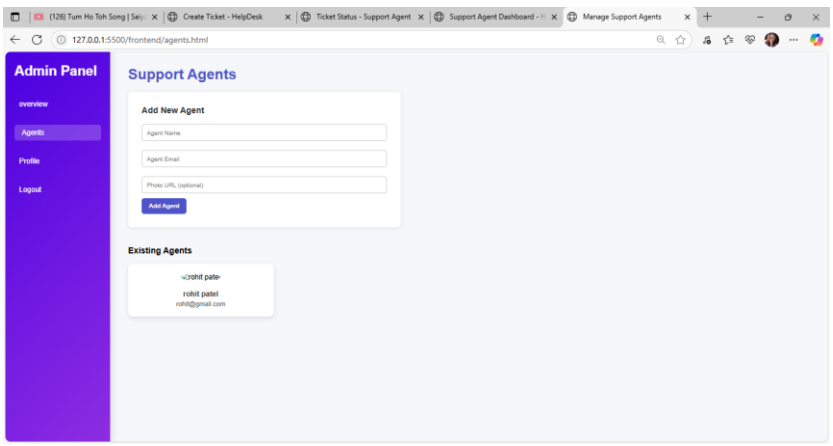


4. **Support Agent Dashboard**
    - Assigned tickets, open tickets, in progress, Resolved



5. **Admin Panel**
    - Manage users and categories
    - View system-wide ticket activity





7. Backend

The backend of the **Customer Support Ticketing System** is designed using a scalable and secure RESTful architecture. It serves as the core engine of the application by handling requests, managing data, and enforcing business logic.

Framework Used

- **Node.js**: JavaScript runtime environment used to build a fast, non-blocking, and scalable server-side application.
- **Express.js**: A lightweight Node.js web application framework used to define API routes, handle middleware, and structure server logic efficiently.

API Structure

The application follows a **RESTful API** design, which enables the frontend to communicate seamlessly with the backend using HTTP methods.

- **GET**: Fetch tickets, users, categories, comments, etc.
- **POST**: Create new tickets, users, categories, comments, etc.
- **PUT/PATCH**: Update ticket status, user details, etc.
- **DELETE**: Remove users, categories, or tickets (admin only)

Routing and Controllers

- **Express Router**: Used to modularize the codebase by separating route files like:
  - `ticketRoutes.js`
  - `commentRoutes.js`
  - `adminRoutes.js`
  - `categoryRoutes.js`
- **Controllers**: Each route is linked to a controller function that handles the logic:
  - `registerUser()`, `loginUser()`
  - `createTicket()`, `getMyTickets()`, `updateTicketStatus()`
  - `addComment()`
  - `getAllUsers()`, `deleteUser()`
  - `createCategory()`, `getAllCategories()`

Authentication and Authorization

- **Role-Based Access Control**:
  - Users are assigned roles: `customer`, `agent`, `admin`.
  - Middleware checks roles to allow or deny access to certain endpoints.
  - Example:
    - Only agents can update ticket statuses.
    - Only admins can view all users and manage categories.
- **Password Security**:
  - Passwords are hashed using `bcrypt.js` before being stored in MongoDB.
  - No plain-text passwords are stored.

8. Database

The backend database plays a critical role in managing users, tickets, comments, and categories. A NoSQL database system was used for its scalability, flexibility, and performance.

DBMS Used

- **MongoDB**
  MongoDB is a NoSQL document-oriented database used to store data in the form of **collections** and **documents (JSON-like)**. It allows for dynamic schemas, making it well-suited for applications with frequently changing data structures like a ticketing system.

Entity-Relationship (ER) Diagram

(If submitting digitally, attach a diagram. Here's a textual representation if visual ER is unavailable)

```
[User]
 └──→ (One-to-Many) → [Ticket]
      └──→ (One-to-Many) → [Comment]
[Category]
 └──→ (One-to-Many) → [Ticket]
```

Schema / Collections

1. Users Collection

```
{
  "_id": ObjectId,
  "name": "John Doe",
  "email": "john@example.com",
  "password": "hashed_password",
  "role": "customer",  // or "agent", "admin"
  "createdAt": ISODate
}
```

2. Tickets Collection

```
{
  "_id": ObjectId,
  "title": "Login not working",
  "description": "Unable to login with correct credentials.",
  "status": "Open",  // Open, In Progress, Resolved
  "userId": ObjectId,  // Refers to Users
  "categoryId": ObjectId,  // Refers to Categories
  "assignedTo": ObjectId,  // Refers to Agent
  "createdAt": ISODate
}
```

3. Collection


```
{
  "_id": "64e48ad2193b8721cf889a7a",
  "name": "Rohan Patel",
  "email": "rohan.patel@ibm.com",
  "photo": "https://example.com/images/agents/rohan-patel.jpg"
}
```

9. Data Flow Diagram

Data Flow Diagrams (DFD)

Level 0 DFD – Context Diagram

This diagram represents the system as a single process and shows how it interacts with external entities.

Entities & Processes:

- **Users** (Customers, Support Agents, Admin)
- **Customer Support Ticketing System** (Main system)
- **Database** (MongoDB)

Data Flows:

- Users → Create/View Tickets → System
- Agents/Admin → Update Ticket Status, Post Comments → System
- System ↔ Reads/Writes → MongoDB

Level 0 DFD

```
[Customer] ------------> (Submit Ticket) ------------> [System]
[Agent/Admin] ---------> (Update/View Tickets) ------> [System]
[System] ---------------> (Store/Retrieve Data) -----> [MongoDB]
```

Level 1 DFD

This level provides a more detailed breakdown of internal components.

Processes:

1. User Authentication
2. Ticket Management
3. Agent Assignment
4. Comments/Status Update
5. Admin Monitoring

Data Stores:

- Users Collection
- Tickets Collection
- Agents Collection
- Comments Collection

Level 1 DFD (Text Description)

```
[Customer] --> (1. Login/Register) --> [Authentication Service] --> [Users
Collection]

[Customer] --> (2. Raise Ticket) --> [Ticket Service] --> [Tickets
Collection]

[Agent] --> (3. View Assigned Tickets) --> [Agent Dashboard] --> [Tickets
Collection]

[Agent] --> (4. Update Status/Post Comment) --> [Ticket Service] -->
[Comments Collection]

[Admin] --> (5. Monitor System) --> [Admin Panel] --> [All Collections]
```

- Data Flow Between Components

Your project has 3 major layers:

- **Frontend** (HTML, CSS, JavaScript)
- **Backend** (Node.js + Express)
- **Database** (MongoDB)

It involves 3 types of users: **Customer**, **Support Agent**, and **Admin**.

1. User Authentication Flow

Actions:

- Sign Up / Log In

Data Flow:
Frontend → Backend (POST /login or /signup) → MongoDB (Users Collection)
Explanation:

1. A user fills the login/signup form on the frontend.
2. Form data is sent to the backend via a REST API endpoint (e.g., /api/login).
3. The backend verifies the user (checks hashed password for login or stores a new user for signup).
4. If successful, backend sends back a JWT token or success message.
5. The token is stored in browser storage (local/session) for authenticated requests.

2. Ticket Creation (Customer Side)

Actions:

- Raise a new support ticket.

Data Flow:
Customer → Frontend Form → Backend (POST /api/tickets) → MongoDB (Tickets Collection)

Explanation:

1. Customer enters ticket details like category, issue description, priority.
2. Data is sent via an HTTP POST request to the backend.
3. Backend creates a new ticket document and stores it in the tickets collection.
4. A response (like "Ticket submitted successfully") is returned to the customer.

3. Agent Dashboard – View Assigned Tickets

Actions:

- Agent logs in and views tickets assigned to them.

Data Flow:
Agent → Frontend Dashboard → Backend (GET /api/assigned-tickets) → MongoDB → Response with Ticket Data
Explanation:

1. Agent logs in and accesses their dashboard.
2. A GET request (with JWT token for authentication) is sent to the backend.
3. The backend queries the tickets collection for tickets assigned to this agent.
4. Tickets are sent back in JSON format to the frontend and displayed in a table/list.

4. Update Ticket Status / Add Comment

Actions:

- Agent or Admin updates the status (e.g., Open → In Progress → Resolved), or adds comments.

Data Flow:
Frontend (Agent/Admin) → Backend (PUT /api/tickets/:id/status or POST /api/comments) → MongoDB

Explanation:

1. Agent/Admin selects a ticket and changes its status or posts a comment.
2. Frontend sends the data via a PUT or POST API call to the backend.
3. Backend validates and updates the relevant ticket document or inserts a new comment in the comments collection.
4. A response confirms the action was successful.

5. Admin Panel – System Monitoring

Actions:

- Admin views all users, all tickets, performance analytics, etc.

Data Flow:
Admin → Frontend → Backend (GET /api/admin/overview) → MongoDB → JSON Response

Explanation:

1. Admin accesses the admin dashboard.
2. Multiple GET API calls are triggered to fetch data (e.g., /api/users, /api/tickets, /api/agents).
3. Backend queries multiple collections and sends the data back.
4. Admin views user metrics, ticket logs, agent performance, etc.

6. Data Validation and Security Flow

- All incoming data is:
    - **Validated** (e.g., email format, required fields)
    - **Sanitized** (to prevent XSS, injection)
    - **Authenticated** (via JWT tokens)
    - **Authorized** (only Agents/Admins can update certain tickets)

10. Limitations & Future Enhancement

Limitations:

While the Customer Support Ticketing System achieves its core functionalities, there are certain **limitations and areas for improvement**:

1. Limited Role-Based Access Control

- Currently, the system provides basic access separation for Customers, Agents, and Admins.
- Granular permission levels (e.g., agent groupings or restricted categories) are **not yet implemented**.

2. Basic Security

- JWT-based authentication is used, but:
  - No token refresh mechanism is implemented.
  - No brute-force protection or rate-limiting on login routes.
- Input validation exists but **could be enhanced further** with a centralized validation library.

3. UI/UX Constraints

- The frontend, built using HTML, CSS, and JavaScript, may lack dynamic responsiveness on all devices.
- Real-time features like live chat or notifications are **not integrated**.

4. Testing Coverage

- Unit tests and API tests are **limited or not fully implemented**.
- Manual testing has been used, but automated testing pipelines are **yet to be developed**.

5. Data Analytics

- The admin dashboard offers basic statistics.
- Advanced analytics (e.g., ticket resolution time, agent performance over time) are **not available**.

6. Email/Notification System

- There's **no automated email** or in-app notification system for:
  - Ticket updates
  - Assigned agent alerts
  - Customer feedback requests

7. Scalability

- The system is optimized for a **small to medium** team size.
- At high scale (thousands of tickets/users), performance optimization and database indexing would be needed.

Future Enhancement:

To improve the system's efficiency, user experience, and scalability, several enhancements can be implemented in future versions:

1. Real-Time Notifications

- Integrate **WebSockets (e.g., Socket.IO)** for live updates when:
  - A ticket is assigned or updated.
  - A new comment is added.
- Push or email notifications to users for important actions.

2. Advanced Analytics Dashboard

- Add detailed analytics for Admins:
  - Ticket resolution time
  - Agent response performance
  - Category-wise ticket heatmaps
- Use chart libraries (e.g., Chart.js, Recharts) for better visualization.

3. Email Integration

- Send automated emails for:
  - Ticket confirmations
  - Status updates
  - Feedback requests
- Use services like **Nodemailer**, **SendGrid**, or **Mailgun**.

4. Enhanced Security

- Implement:
  - Role-based permissions using RBAC
  - Account lockout after multiple failed logins
  - Two-Factor Authentication (2FA)
- Encrypt sensitive data beyond just passwords.

5. Multi-language Support

- Allow users to use the platform in different languages via localization (i18n).
- Helpful for global or diverse user bases.

6. Ticket Prioritization and SLA Tracking

- Add priority levels with deadlines.
- Monitor SLA (Service Level Agreement) compliance.
- Auto-escalate overdue tickets to higher authorities.

7. Responsive & Mobile App

- Create a mobile-responsive version or a mobile app using **React Native** or **Flutter** for on-the-go support.

8. Automated Testing and CI/CD

- Integrate:
  - Unit testing with Jest/Mocha
  - API testing using Postman/Newman
  - Continuous Integration tools like GitHub Actions or Jenkins

9. AI-Powered Support Assistant

- Integrate a chatbot for common queries using tools like:
  - Dialogflow
  - GPT APIs
- Use machine learning to suggest ticket responses for agents.

10. Third-Party Integrations

- Integrate with:
  - Slack, Teams, or WhatsApp for real-time support
  - CRM tools like Salesforce or HubSpot

11. Annexures

Code Snippets

1. JWT Token Generation for Authentication

```
const jwt = require("jsonwebtoken");

function generateToken(user) {
  return jwt.sign({ id: user._id, role: user.role },
process.env.JWT_SECRET, {
    expiresIn: "7d",
  });
}
```

2. Create Ticket API Endpoint

```
router.post("/tickets/create", authenticateUser, async (req, res) => {
  const { subject, description, category } = req.body;
  const newTicket = new Ticket({
    subject,
    description,
    category,
    customer: req.user.id,
    status: "Open",
  });
  await newTicket.save();
  res.status(201).json(newTicket);
});
```

3. Agent Schema (MongoDB Collection)

```
const mongoose = require("mongoose");

const agentSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  photo: String,
});

module.exports = mongoose.model("Agent", agentSchema);
```

API Documentation

Endpoint	Method	Description	Authentication
`/api/tickets/create`	POST	Create a new support ticket	Required
`/api/tickets/:id`	GET	View ticket by ID	Required
`/api/agents`	GET	List all support agents	Admin only
`/api/comments/:ticketId`	POST	Add comment to a ticket	Required
`/api/auth/login`	POST	User login	Public

GitHub Repository

- **Repository Link**: https://github.com/mahekk-24/support-ticket-system

12. References

This section lists the books, online libraries, tools, and other resources referred to during the development of the **Customer Support Ticketing System** project.

Online Resources & Libraries

Online resources- IBM career Education program

Course Link- https://ibmcep.cognitiveclass.ai/

Tools & Platforms

- **Visual Studio Code** – Code editor used for development
- **Git & GitHub** – Version control and code repository
- **Postman** – For testing RESTful APIs
- **MongoDB Atlas** – Cloud-hosted MongoDB for database storage