You can choose any one problem statement out of given two!

All the best!

Problem Statement 1:-

QuickDesk

Purpose

The purpose of QuickDesk is to provide a simple, easy-to-use help desk solution where users can raise support tickets, and support staff can manage and resolve them efficiently. The system aims to streamline communication between users and support teams without unnecessary complexity.

Users

- End Users (Employees/Customers): Can create and track their tickets.
- Support Agents: Can respond to and resolve tickets.
- Admin: Can manage users, categories, and overall ticket flow.

Functional Requirements

- Users should be able to register/login to the system.
- Users can create a ticket with subject, description, category, and optional attachment.
- Users can view the status of their submitted tickets.

- User should have search & filtering options,
  - To see open/closed tickets
  - See their own tickets only
  - Search based on category
  - Sort based on most replied tickets/recently modified
- Support agents can view, assign, and update tickets.
- Ticket statuses: Open → In Progress → Resolved → Closed.
- Agents can add comments/updates to tickets.
- Admin can create and manage ticket categories.
- User can upvote & downvote the questions
- Email notifications are sent when a ticket is created or status changes.

Basic User Flow

1. User registers/logs in.

2. The user creates a ticket with details.

3. The ticket goes to the "Open" state.

4. The support agent picks it up and updates status.

5. Users receive updates and can reply if needed.(only on own tickets)

6. The ticket is resolved and closed.

Detailed Screens per Role

- End User:
  - Dashboard with filters, search, pagination, sorting.
  - Ticket creation with validations, attachments, category selector.
  - Ticket detail with threaded conversations (timeline).
  - Profile & settings & notification

- ## Support Agent:
  - Dashboard with multiple ticket queues (My Tickets, All).
  - Actions (Reply, share).
  - Create the tickets like end user

- ## Admin:
  - User management (roles, permissions).
  - Category management.

ExceliDraw: https://link.excalidraw.com/l/65VNwvy7c4X/83JslFMQqb3

Problem Statement 2:-

CivicTrack

Empower citizens to easily report local issues such as road damage, garbage, and water leaks. Seamlessly track the resolution of these issues and foster effortless engagement within your local community.

Features Visibility

- Only civic issues reported within a 3-5 km radius are visible to the user, based on GPS or manual location.
- Users cannot browse or interact with reports outside their neighborhood zone.

Quick Issue Reporting

- Users can report issues with a title, short description, photos (up to 3 5 , and category selection.
- Anonymous or verified reporting is supported.

Supported Issue Categories

- Roads (potholes, obstructions)
- Lighting (broken or flickering lights)
- Water Supply (leaks, low pressure)
- Cleanliness (overflowing bins, garbage)
- Public Safety (open manholes, exposed wiring)
- Obstructions (fallen trees, debris)

Status Tracking

- Issue detail pages show status change logs and timestamps for transparency.
- Reporters get notified when an issue status is updated.

Map Mode & Filtering

- Show all issues as pins on a map.
- Users can filter issues by:
    - Status (Reported, In Progress, Resolved)
    - Category
    - Distance (1 km, 3 km, 5 km)

Moderation & Safety

- Spam or irrelevant reports can be flagged.
- Reports flagged by multiple users are auto-hidden pending review.

Admin Role

- Review and manage reported issues flagged as spam or invalid.
- Access analytics: total issues posted, most reported categories.
- Ban users if needed.

Mockup :- https://link.excalidraw.com/l/65VNwvy7c4X/8YC7pOyxtr5

Evaluation Criteria for Virtual Round

1. Coding Standards

- Consistent naming conventions
- Proper indentation and formatting
- Clear, maintainable, and idiomatic code
- Comments and documentation (docstrings, JSDoc, etc.)
- Avoidance of code smells and anti-patterns

2. Logic

- Correctness of business logic and workflows
- Clear and understandable control flow
- Handling of edge cases and errors
- Accurate implementation of requirements

3. Modularity

- Separation of concerns
- Reusable functions, components, modules
- Clean project structure (folders, files)
- Low coupling and high cohesion

4. Database Design

- Well-structured schema (normalized where appropriate)
- Clear relationships between entities
- Efficient indexing and querying
- Use of migrations and version control for schema
- Safe, parameterized queries to prevent injection

5. Frontend Design

- Clean, intuitive UI design
- Consistent styling and layout
- Responsiveness (mobile-friendly if applicable)
- Accessibility (ARIA, labels, alt text)
- Maintainable frontend code (components, CSS organization)

6. Performance

- Efficient algorithms and queries
- Avoidance of bottlenecks (long loops, blocking calls)
- Optimized assets (images, scripts)
- Caching strategies if relevant
- Lazy loading where appropriate

7. Scalability

- Architecture supports growth (users, data, features)
- Decoupled components/services
- Stateless design where appropriate
- Support for horizontal scaling (load balancing, session storage)
- Consideration of future maintenance and extensibility

8. Security

- Input validation and sanitization
- Protection against common vulnerabilities (SQL Injection, XSS, CSRF)
- Secure authentication and authorization
- Password hashing and secure storage
- Use of HTTPS and secure headers if applicable

9. Usability

- User-friendly navigation and interactions
- Clear error messages and feedback
- Consistent UI/UX patterns
- Intuitive forms and workflows
- Help or documentation for users