ResolveIT Helpdesk System
Comprehensive SDLC Documentation
Enterprise Application Development
IDS517

Group 5 :

Tanishq Padwal
Akshay Shenoy
Nonitha Vampati
Ashna Sheregar
Kamal Teckchandani
Omkar Nehete
Saatvik Shukla

ResolveIT: Streamlined Technology Support System

ResolveIT Development Team:

Tanishq Padwal:
- Core application development
- Client-server communication
- Project Planning and scheduling
- Advanced database operations

Akshay Shenoy:
- Security enhancements
- Automated email notifications
- Performance optimization

Nonitha Vampati:
- GUI design and implementation
- User experience optimization
- File attachment functionality
- Documentation creation

Ashna Sheregar:
- GUI design and implementation
- Service request management
- Automated email notifications
- Supervisor review process

Kamal Teckchandani:
- File attachment functionality
- Testing and debugging
- User acceptance testing
- Bug tracking and resolution

Omkar Nehete:
- Service request management
- User manual and technical documentation
- Improvements from suggestion in HW5

Saatvik Shukla:
- Database schema design
- File attachment functionality
- Milestone tracking
- Improvements from suggestion in HW5

Team Meet Schedule :

- Communication and progress report (M & W)
- Development of Helpdesk System (M,W & F)
- Testing (M & W)
- Brainstorming future features (M,W & F)
- Weekly Integration testing and Compatibility/Bug Reporting(F)
- Code Maintenance (W & F) (Weekends if required)

Team Progress :

- **Advanced Database Operations:** Enabled real-time data updates and introduced a comment system for detailed service request tracking.
- **File Attachments:** Added capability for users to attach files to service requests, enhancing issue context.
- **Supervisor Review Process:** Implemented a supervisor review workflow with priority adjustment features.
- **Automated Email Notifications:** Set up automatic email notifications for service request updates.
- **Security Enhancements:** Secured user data with password hashing and secure database connections.
- **Client-Server Architecture:** Established a prototype client-server model.

Business Area :

The selected business area for the implementation of the Online Helpdesk System is the Information Technology (IT) Support Services sector. This sphere is integral to the operational backbone of any modern enterprise, acting as the first line of defense against technical disruptions that can impede business processes. It encompasses a broad range of services including but not limited to hardware maintenance, software troubleshooting, network management, and cybersecurity defenses.

Technology underpins virtually every aspect of business operations, the role of IT Support Services extends beyond mere maintenance. It is about enabling and empowering the workforce through technology, ensuring that all systems function seamlessly, and providing swift resolutions to technical issues that could otherwise lead to productivity loss.

The Online Helpdesk System is the hub for all IT support activities, offering a central platform for employees to report issues, track their status, and receive assistance. It streamlines ticket management, prioritizes problems, and allocates resources efficiently for prompt issue resolution.

- **Ticket Submission:** Employees can report issues with a detailed description and any relevant information that can assist in the troubleshooting process.

- **Ticket Tracking:** Users can view the status of their tickets at any time, from submission to resolution.

- **Resource Allocation:** IT support staff can assign tickets to themselves or other team members, ensuring that the right skills are applied to each issue.

- **Resolution Workflow:** A systematic approach to resolving tickets, including steps for diagnosis, action, follow-up, and closure.

- **Client-Server Architecture Implementation:** Upgraded the system for enhanced interaction between the client-side user interface and the server-side database management, offering real-time updates and synchronization.

- **Enhanced Security Measures:** Integrated advanced security protocols to protect sensitive data and user information, ensuring secure access and data integrity.

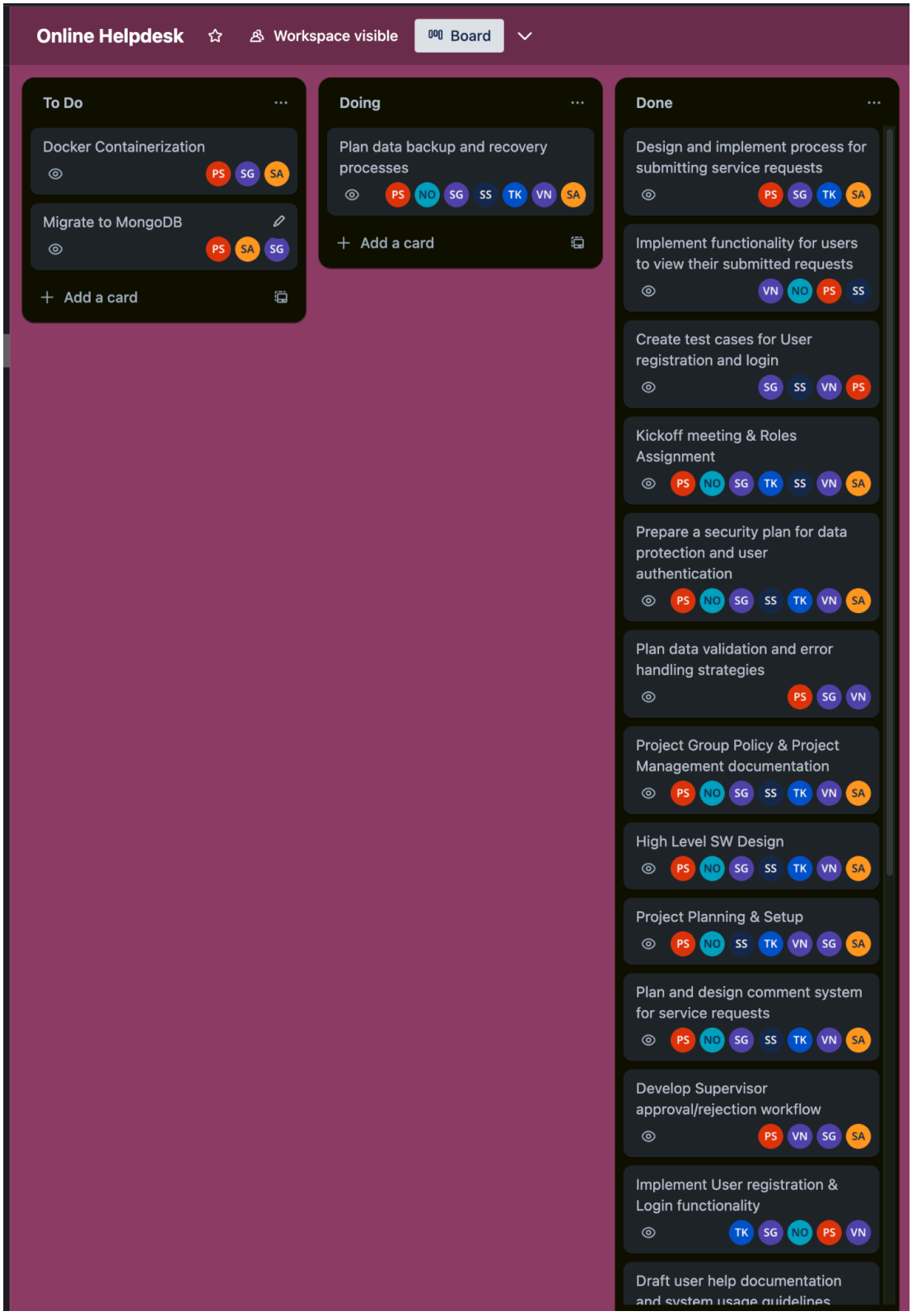
- **User Experience Optimization:** Refined the user interface with intuitive design elements, reducing complexity and improving overall user engagement.

- **Automated Notifications:** Developed an automated alert system to keep users and IT staff informed about ticket progress and system updates through timely notifications.

- **Expanded Support Resources:** Included a knowledge base and FAQ section to provide users with immediate self-help options and reduce ticket submissions for common issues.

- **Service Level Agreement (SLA) Tracking:** Integrated SLA tracking to ensure timely responses and resolutions in accordance with predefined service standards.

Trello Board :



Project scheduling : Task List

February 29th - March 1st

Project Initiation:

- Conducted an initial kickoff meeting to establish the project team, define roles, and outline group policies.
- Drafted the initial Project Management Plan outlining the project's scope, objectives, and preliminary schedule.

Development Environment Setup:

- Installed Java Swing and other essential libraries for GUI development to create a robust development environment.
- Configured version control using Git and set up a centralized repository on GitHub for collaborative development and code management.

Data Structure Planning:

- Analyzed business requirements to decide on utilizing a stack or queue model for managing service requests efficiently.
- Implemented the chosen data structure in a prototype Java class to validate its integration with the planned features.

March 1st - March 6th

User Interface Design:

- Drafted initial GUI wireframes using Java Swing, focusing on user experience and interface flow.
- Developed the primary navigation and interactive elements to ensure intuitive user interactions.

Core Functionality Programming:

- Programmed the primary system functionalities, including user account management and service request processes.
- Built the underlying business logic to ensure seamless interaction between the user interface and application backend.

March 7th - March 10th

Feature Development and Integration:

- Completed all critical application features, including account registration, user authentication, and handling service requests.
- Integrated the stack/queue model into the application's logic, ensuring that service requests are processed effectively.

Quality Assurance:

- Conducted rigorous unit tests to validate the functionality of individual components.
- Initiated functional and integration testing to ensure that all parts of the application work together seamlessly.

March 10th - March 17th

User Interface and Experience Optimization:

- Refined the GUI based on user feedback, with an emphasis on creating a responsive and accessible user interface.
- Optimized performance by streamlining the stack/queue implementation for faster processing of requests.

March 25th - April 1st

Performance Optimization:

- Conducted code reviews and refactoring to improve application efficiency.
- Enhanced database queries to reduce latency and improve response times for end-users.

Testing and Debugging:

- Performed extensive unit and functional testing to ensure all features operate as intended.
- Initiated a second round of user acceptance testing (UAT) with updated features.

Feature Enhancement:

- Implemented mandatory double confirmation for delete operations to improve data integrity and prevent accidental data loss.
- Added file attachment support in service requests to allow users to upload relevant documents or images.
- Introduced a knowledge base section for common troubleshooting and FAQs to aid self-service for users.
- Implemented dynamic Service Level Agreement (SLA) response times based on ticket severity to set accurate expectations for users.
- Developed an email notification system to keep users informed about the status of their service requests and system updates.

User Experience and Interface:

- Conducted a UX/UI review to refine interface elements and improve the overall user journey within the application.
- Added tooltips and help text throughout the application for greater user guidance and support.

Data Management :

Database Schema Design: Utilized SQLite for efficient data storage. Designed schemas for users and service requests with relations to track ticket assignments and resolutions.

Users Table

Purpose: Stores information about the users of the Helpdesk system, including both employees submitting tickets and IT staff handling those tickets.

Fields:

- UserID (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique identifier for each user.
- Name (TEXT, NOT NULL): Full name of the user.
- Email (TEXT, NOT NULL, UNIQUE): Email address of the user, used for login and notifications.
- Role (TEXT, NOT NULL): Defines the user's role within the system (e.g., "Employee", "IT Staff").
- Password (TEXT, NOT NULL): Hashed password for user authentication.

ServiceRequests Table

Purpose: Captures details of each service request (or ticket) submitted by users, tracking its progress from submission to resolution.

Fields:

- RequestID (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique identifier for each service request.
- Problem (TEXT, NOT NULL): Description of the issue or problem being reported.
- Severity (TEXT, NOT NULL): Severity level of the request (e.g., "Low", "Medium", "High", "Critical").
- SubmittedBy (INTEGER, NOT NULL): UserID of the employee who submitted the request, linking to the Users table.
- Description (TEXT, NOT NULL): Detailed explanation of the issue, including steps to reproduce or specific error messages.
- Priority (TEXT): Priority assigned to the request by IT staff, influencing resolution order.
- AssignedTo (INTEGER, DEFAULT NULL): UserID of the IT staff member assigned to handle the request.
- Status (TEXT, DEFAULT 'Pending'): Current status of the request (e.g., "Pending", "In Progress", "Resolved").
- Comment (TEXT): Optional comments added by IT staff during the resolution process.
- Timestamp (DATETIME, DEFAULT CURRENT_TIMESTAMP): The date and time when the request was submitted.
- ResolutionDate (DATETIME, DEFAULT NULL): The date and time when the request was resolved.

- FilePath (TEXT): Path to any files attached to the request for further context.

Data Flow: Implemented robust data flow mechanisms to ensure smooth operation from ticket submission through to resolution, with updates reflected in real-time.

Ticket Submission: Employees encounter an issue and log into the Helpdesk system to submit a new service request. They provide a problem description, severity, and any attachments that might help diagnose the issue. This data is entered into the ServiceRequests table.

Ticket Assignment: IT staff reviews incoming tickets in the system. Based on the problem's severity and priority, they assign the ticket to an appropriate team member by updating the AssignedTo and Priority fields in the ServiceRequests table.

Issue Resolution: The assigned IT staff member works on the ticket, updating the ticket's status to "In Progress." They may add comments in the Comment field to document the resolution process. Once resolved, they update the ticket's status to "Resolved" and fill in the ResolutionDate.

Notification and Closure: Upon marking a ticket as resolved, an automated email notification is sent to the ticket submitter, informing them of the resolution. The submitter then reviews the resolution and closes the ticket, or requests further assistance if needed.

Operation

- **System Architecture**
  The Helpdesk system is built on a client-server architecture, which separates the user interface (client side) from the data management and processing (server side). This design enhances the system's scalability, security, and ease of maintenance.
- **Client Application:** Developed using Java Swing, the client application provides a graphical user interface (GUI) through which users interact with the Helpdesk system. It includes forms for user registration, login, service request submission, and viewing the status of requests.
- **Server:** The server component handles requests from multiple clients simultaneously. It processes user authentication, service request management, and communicates with the database for data retrieval and updates.
- **Database:** SQLite is used as the database management system (DBMS) to store and manage user and service request data. It maintains tables for users, service requests, and potentially other metadata like logs and system settings.
- **Communication:** The client and server communicate over a network, using sockets for data transmission. The client sends requests to the server, like login attempts or service request submissions, and the server responds after processing the request, often involving database operations.
- **Operational Workflow:** Developed a clear operational workflow to manage user interactions, from registration and ticket submission to resolution and feedback.

User Registration:

- New users fill out a registration form on the client application.
- The client sends the registration data to the server.
- The server validates the data and creates a new entry in the Users table in the database.

User Login:

- Users enter their credentials in the login form.
- The client sends these credentials to the server.
- The server validates the credentials against the Users table and responds with a success or failure message.

Ticket Submission:

- Logged-in users submit new service requests using a dedicated form.
- The client application sends the request details to the server.
- The server adds a new entry to the ServiceRequests table with a status of "Pending."

Ticket Assignment:

- IT staff review pending tickets on their client application.
- They select a ticket to work on, updating its status to "In Progress" and assigning themselves as the handler.
- These updates are processed by the server and reflected in the database.

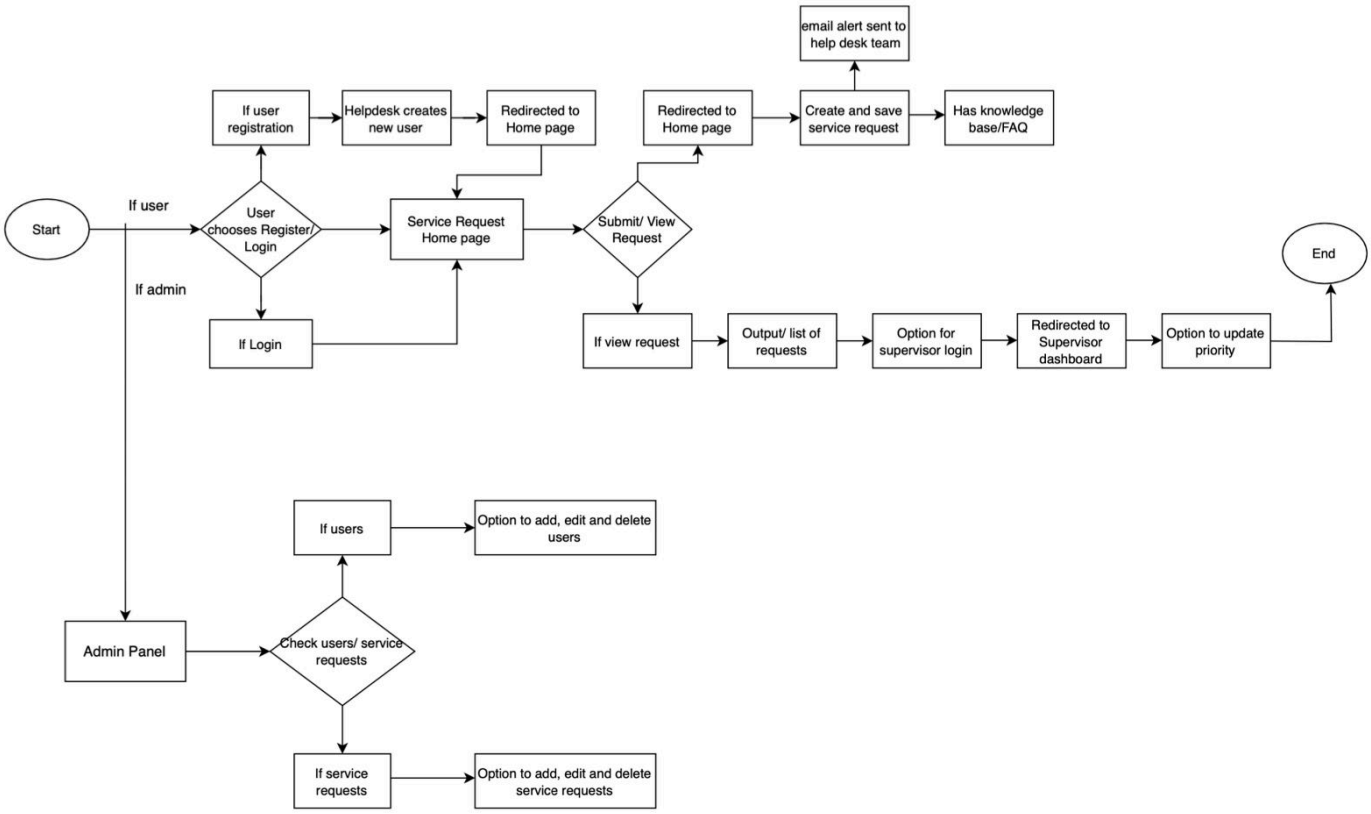
Ticket Resolution:

- Once the IT staff resolves the issue, they update the ticket's status to "Resolved" and fill in the resolution details and resolution date.
- The server processes this update and notifies the user who submitted the ticket.

UML Diagram :


User
+UserID: int
+Name: string
+Email: string
+Password: string
+Role: string
+RegisterUser(Name, Email, Role) : : string
+Login(Email, Password) : : string
1
*
ServiceRequest
+RequestID: int
+Title: string
+Description: string
+Status: string
+SubmittedBy: int
+AssignedTo: int
+SubmitRequest(UserID, Title, Description) : : string
+UpdateRequestStatus(RequestID, NewStatus, Comment, UserID) : : string
1
*
1
1
Comment
+CommentID: int
+RequestID: int
+CommentText: string
+CreatedBy: int
+CreatedAt: DateTime
+AddComment(RequestID, CommentText, UserID)
SupervisorApproval
+ApprovalID: int
+RequestID: int
+ApprovedBy: int
+ApprovalStatus: string
+ApprovalComment: string
+ApprovalDate: DateTime
+ApproveRejectRequest(RequestID, ManagerID, ApprovalStatus, ApprovalComment) : : string

Flowchart :



Requirements

Functional Requirements : Addressed essential functionalities like secure login, ticket management, and email notifications.

- **User Authentication:** Secure login mechanism with email and password.
- **Service Request Management:** Ability to create, view, update, and delete service requests.
- **File Attachments:** Users can attach files to their service requests for better issue illustration.
- **Email Notifications:** Automated email notifications for ticket updates and resolutions.
- **Supervisor Review Process:** Supervisors can review, prioritize, and assign service requests to IT staff.

Non-functional Requirements : Emphasized performance, security, and compatibility, ensuring a reliable and secure user experience.

- **Performance:** The system should handle concurrent users without significant performance degradation.
- **Usability:** The GUI should be intuitive and responsive, with clear navigation.
- **Security:** Implement secure authentication, encrypted data transmission, and protect against common vulnerabilities.
- **Compatibility:** Ensure compatibility across major operating systems and support for the latest three versions of popular web browsers.

Resource Allocation

- **Team Members:** Tasks were distributed based on expertise, with developers focusing on their areas of strength such as backend, frontend, or database management.
- **Tools and Technologies:** Utilized Java Swing for the GUI, SQLite for the database, and JavaMail API for email notifications.
- **External Resources:** Incorporated open-source libraries for enhanced functionalities, such as file upload handling, mail notifications.

Testing

Testing Strategy

- **Unit Testing:** Focused on individual components to ensure they function correctly in isolation.
- **Integration Testing:** Tested combinations of components to verify their interactions and data flow.
- **User Acceptance Testing:** Performed by a select group of end-users to validate the complete system's functionality against real-world scenarios.

Test Cases and Results

- **User Login:** Tested with valid and invalid credentials. All expected outcomes matched actual results.
- **Service Request Submission:** Verified successful creation and failure scenarios due to missing information. Identified and fixed a bug related to file attachment handling.

Debugging

Debugging Tools and Techniques

- **IDE Debuggers:** Utilized the debugging features in IntelliJ IDEA, setting breakpoints, and stepping through code execution.
- **Logging:** Implemented detailed application logs to capture runtime behaviors and errors.

Challenges and Solutions

Distribution Challenges: The necessity for manual configuration of the database path makes it difficult to seamlessly distribute our application to a broader audience.

User Experience Impact: This manual setup process can be a barrier for less technically inclined users, potentially limiting the application's accessibility and usability.

Scalability Concerns: As our application grows in features and complexity, the dependency on local configurations such as database paths could further complicate deployment and scalability.

Our team acknowledges this limitation and is exploring potential solutions, such as containerization with Docker and migrating to cloud-based database services, to make our application more accessible and easier to use for everyone.