|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |
|  |
| **Software Design Document**  **Moayad Hamdan** |
|  | **To: Dern-Support** |  |
|  |  |  |
|  |  |  |

Contents

[1. Functional and Non-Functional Requirements Specifications 3](#_Toc177069317)

[Functional Requirements: 3](#_Toc177069318)

[Non-Functional Requirements: 3](#_Toc177069319)

[Image: 3](#_Toc177069320)

[2. Algorithm Design Documentation 4](#_Toc177069321)

[Algorithm Design: 4](#_Toc177069322)

[Image: 5](#_Toc177069323)

[3. Visual Design Documentation 8](#_Toc177069324)

[Visual Design: 8](#_Toc177069325)

[Image: 8](#_Toc177069326)

[4. Data Requirements Design 12](#_Toc177069327)

[Data Requirements: 12](#_Toc177069328)

[Image: 13](#_Toc177069329)

[5. Accessibility and Inclusivity Considerations 14](#_Toc177069330)

[1. Responsive Design: 14](#_Toc177069331)

[2. Text Size and Contrast: 15](#_Toc177069332)

[3. Alternative Layouts: 15](#_Toc177069333)

[4. Support for Inclusive Technologies: 15](#_Toc177069334)

[5. Mockups and Design Features: 16](#_Toc177069335)

[6. Training and Documentation: 16](#_Toc177069336)

[6. Reused/Refactored Components 16](#_Toc177069337)

[Components: 16](#_Toc177069338)

[Reused Code: 16](#_Toc177069339)

[1. Refactored Code: 17](#_Toc177069340)

[Detailed Description: 18](#_Toc177069341)

[Reused Code: 18](#_Toc177069342)

[Refactored Code: 19](#_Toc177069343)

[Account Management: 19](#_Toc177069344)

[Review and Feedback Record 20](#_Toc177069345)

[Review Process: 20](#_Toc177069346)

[Review Process: 30](#_Toc177069347)

# 1. Functional and Non-Functional Requirements Specifications

## Functional Requirements:

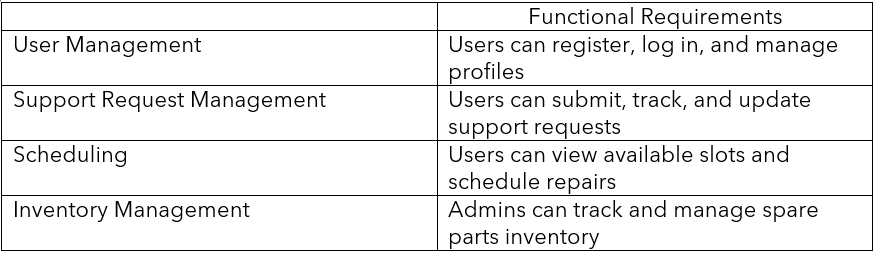
* **User Management:** Users can register, log in, and manage profiles.
* **Support Request Management:** Users can submit, track, and update support requests.
* **Scheduling:** Users can view available slots and schedule repairs.
* **Inventory Management:** Admins can track and manage spare parts inventory.
* **Data Analytics:** Dashboard for trend analysis and performance tracking.

## Non-Functional Requirements:

* **Scalability:** System should handle increasing user load and data volume.
* **Performance:** Operations should have a response time under 2 seconds.
* **Usability:** Interfaces should be intuitive and user-friendly.
* **Reliability:** System uptime should be 99.9%, with robust backup plans.

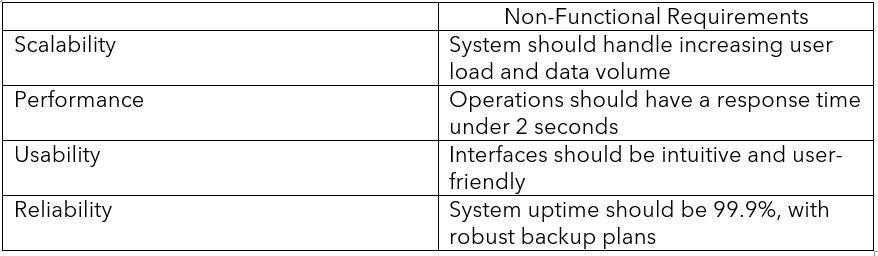
## Image:

* **Functional Requirements:**



Functional Requirements img 1

* **Non-Functional Requirements:**



Non-Functional Requirements img 1

# 2. Algorithm Design Documentation

## Algorithm Design:

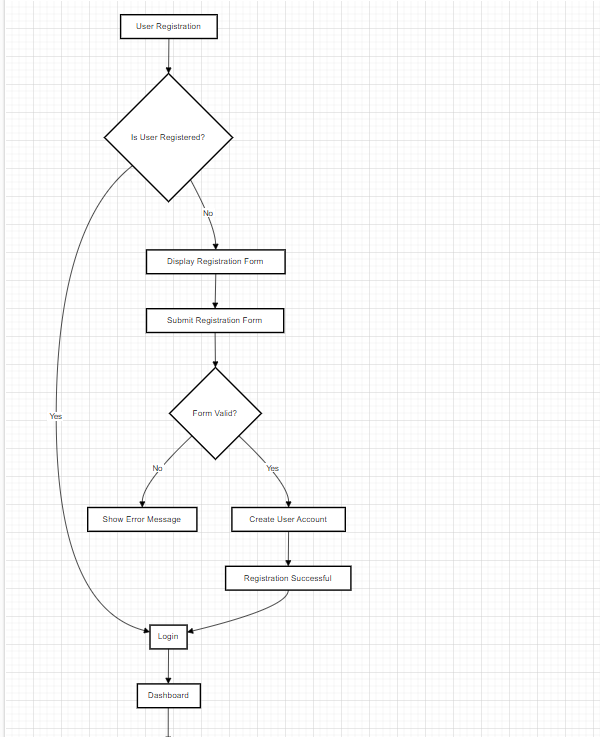
* **Flow Charts:**

**Explanation:**

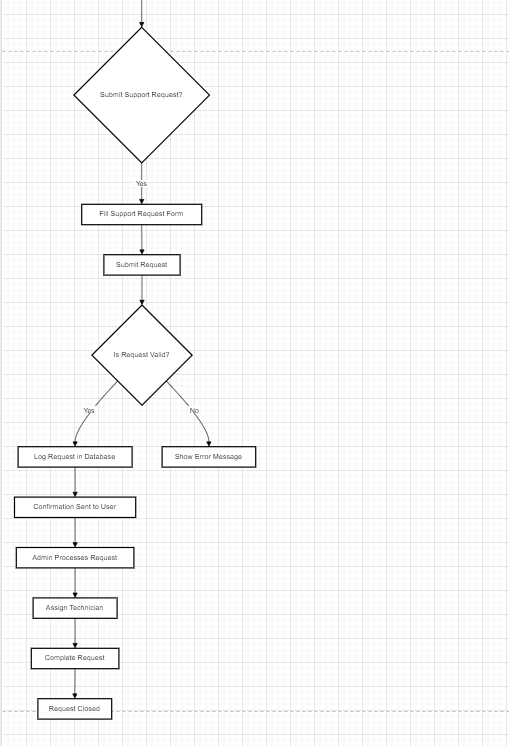
1. **User Registration Process**:
   * Users either log in if registered or go through the registration process.
   * Includes form validation and error handling.
2. **Dashboard and User Profiles**:
   * After login, users can access their profile or proceed to support requests.
3. **Support Request Handling**:
   * Users submit support requests, which are validated and logged. Admins process and assign technicians, schedule repairs, and close requests.
4. **Knowledge Base**:
   * Users can access diagnostic tools and instructional guides.
5. **Spare Parts Inventory Management**:
   * Admins can search and edit inventory, with real-time tracking and updates.
6. **Job Scheduling and Prioritization**:
   * Admins schedule jobs, view services, and prioritize repairs.
7. **Data Analytics**:
   * Provides access to trend and performance analytics.

## Image:

* **Flow Chart:**



Flow Chart img 1



Flow Chart img 2

# 3. Visual Design Documentation

## Visual Design:

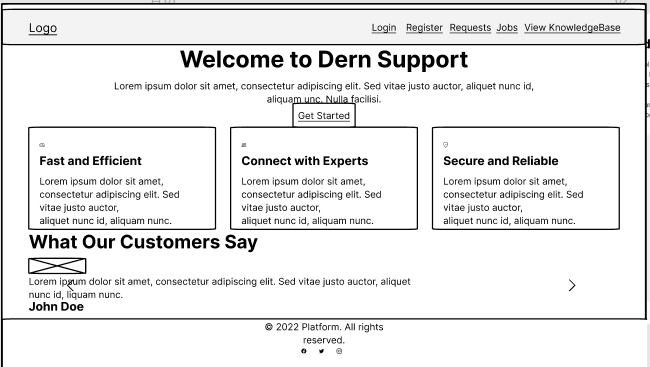
**Wireframes:**

This is the wireframe of the website, showcasing the structure of key pages:

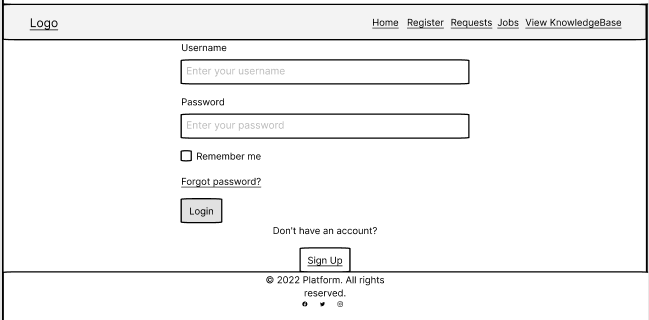
* **Home Page**: The landing page where users can learn about the website and its services.
* **Login Page**: Allows users and admins to log into their accounts.
* **Register Page**: Where users can create a new account, either as a regular user or a business account.
* **Jobs Page**: Displays the scheduled repair jobs for both users and admins.
* **Request Page**: Users can submit a service request with details and track its status.
* **Knowledge Base Page**: Provides common IT issues and solutions for users to explore.

## Image:

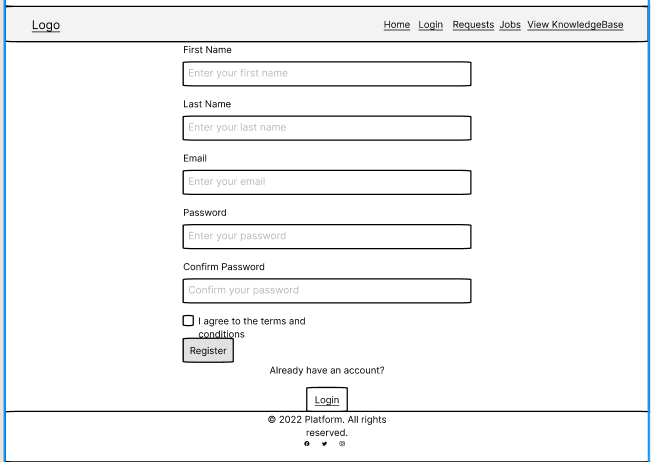
* **Wireframes:**



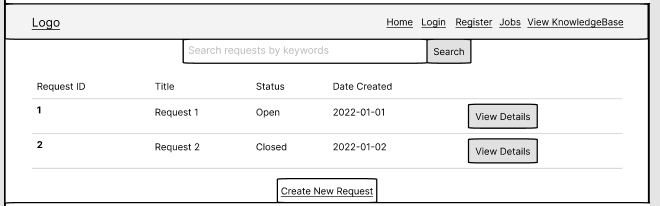
Home Wireframes 1



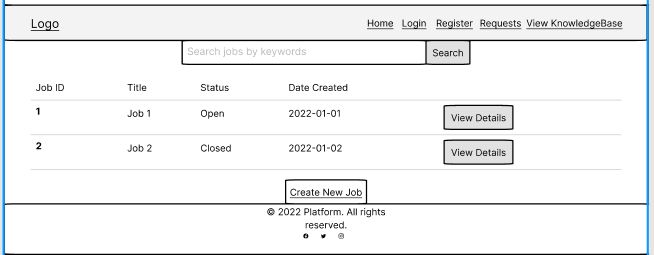
Login Wireframes 1



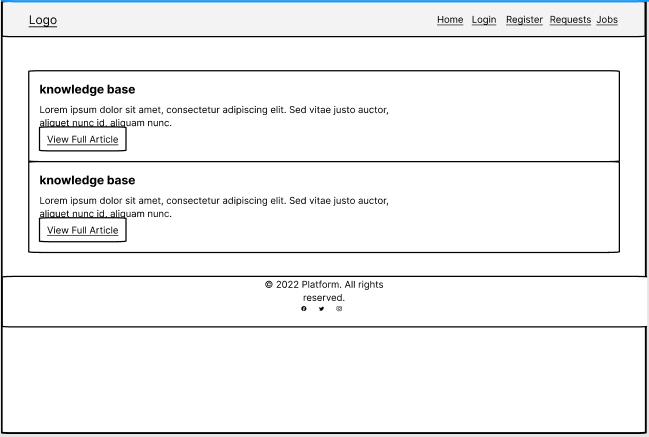
Register Wireframes 1



Request Wireframes 1



job Wireframes 1



knowledge base Wireframes 1

# 4. Data Requirements Design

## Data Requirements:

* **Entity Relationship Diagrams (ERDs):**

**Key Entities:**

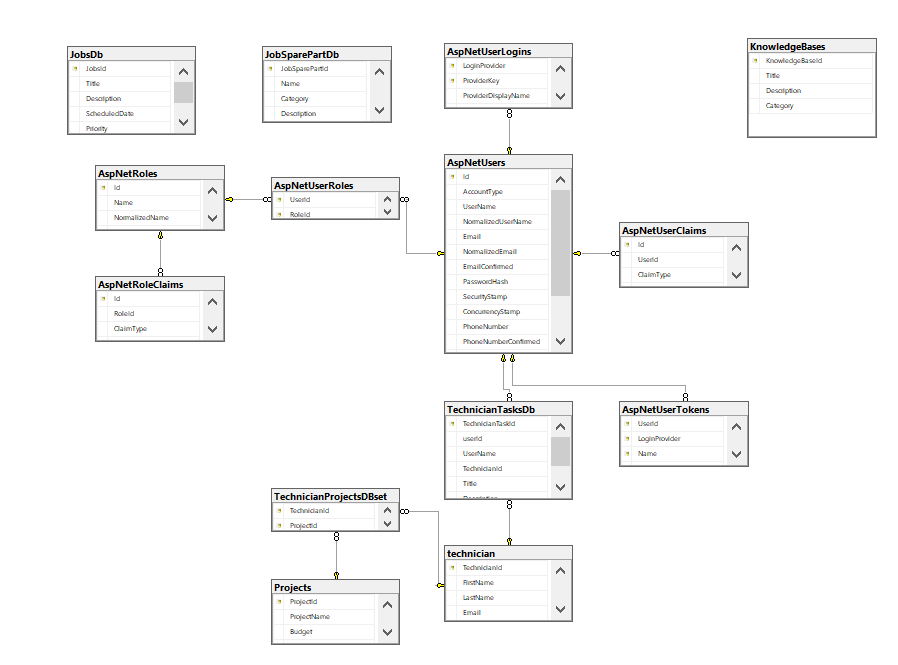
* **AspNetUsers**:
* This table stores user information such as Username, Email, and AccountType (likely defining user roles like Admin, Client, or Technician).
* It is connected to other tables that manage user authentication (AspNetUserLogins, AspNetUserTokens, AspNetUserRoles).
* **AspNetRoles**:
* This table defines user roles, with connections to AspNetUserRoles to assign roles to specific users.
* AspNetRoleClaims handles claims (permissions) related to roles.
* **JobsDb**:
* Stores job details such as Title, Description, ScheduledDate, and Priority of tasks that need to be performed, likely by technicians.
* **TechnicianTasksDb**:
* Manages the tasks assigned to technicians. Each task is linked to a TechnicianId and a UserId (the person requesting the task).
* **Technician**:
* This table contains the details of technicians (TechnicianId, FirstName, LastName, Email).
* **TechnicianProjectsDBSet** and **Projects**:
* These tables track which technicians are assigned to which projects. TechnicianProjectsDBSet links technicians to specific projects by TechnicianId and ProjectId.
* Projects contains information about different projects, including the ProjectName and Budget.
* **JobSparePartDb**:
* Likely represents the spare parts used in jobs, with fields for Name, Category, and Description of the parts.
* **KnowledgeBases**:
* This table holds information about common IT issues and solutions, with fields like Title, Description, and Category.

**Connections and Relationships:**

* **User-Role Relationships**:
* AspNetUserRoles links users to roles defined in AspNetRoles, while AspNetUserClaims and AspNetRoleClaims manage specific permissions assigned to users and roles.
* **Technician Assignments**:
* TechnicianTasksDb and TechnicianProjectsDBSet show which tasks or projects are assigned to specific technicians.
* **Job and Inventory Management**:
* JobsDb lists the tasks, and JobSparePartDb manages the spare parts required for those jobs.
* .

## Image:

* **ERD:**



ERD img 1

# 5. Accessibility and Inclusivity Considerations

## 1. Responsive Design:

1. Adaptability to Devices: Ensure the UI adjusts seamlessly across different screen sizes, including mobile phones, tablets, and desktop monitors. Utilize a responsive design framework (like Bootstrap or Flexbox) to achieve this adaptability.
2. Fluid Layouts: Implement fluid grids and flexible images that resize based on the screen size to maintain usability and aesthetics on various devices.
3. Breakpoints: Define breakpoints in CSS to adjust layouts for different screen resolutions, ensuring that content remains accessible and well-organized regardless of device.

## 2. Text Size and Contrast:

1. Sufficient Contrast: Adhere to the Web Content Accessibility Guidelines (WCAG) to provide high contrast between text and background colors. This helps users with visual impairments read content more easily.
2. Adjustable Font Sizes: Incorporate features that allow users to adjust font sizes according to their preferences. This can be achieved through scalable units (e.g., em, rem) and user interface controls that enable text resizing.
3. Text Resizing Tools: Integrate browser-based text resizing tools or built-in features in the operating system to accommodate users with different visual needs.

## 3. Alternative Layouts:

1. Simplified Views: Provide alternative layouts that simplify content for users who may benefit from less complex designs. For example, a “reader mode” that presents content in a clean, distraction-free format.
2. Keyboard Navigation: Ensure that all interactive elements (buttons, links, forms) are accessible via keyboard navigation. Implement proper tab indices and focus management to support users who rely on keyboards or screen readers.
3. Adjustable Interfaces: Allow users to switch between different interface modes, such as a high-contrast mode for those with visual impairments or a simplified mode for easier navigation.

## 4. Support for Inclusive Technologies:

1. Screen Readers: Ensure compatibility with screen readers by using semantic HTML and ARIA (Accessible Rich Internet Applications) roles and properties. Test the application with popular screen readers like JAWS, NVDA, or VoiceOver.
2. Speech Recognition: Support speech recognition technologies for users who prefer or require voice commands to navigate and interact with the application.
3. Assistive Devices: Consider compatibility with various assistive devices such as adaptive keyboards, eye-tracking systems, and switch devices. Ensure that your application can be used effectively with these technologies.

## 5. Mockups and Design Features:

1. Accessible Design Mockups: Include mockups in the design phase that showcase accessible design features, such as high-contrast themes, large clickable areas, and easy-to-read fonts.
2. Design Feedback: Gather feedback from users with disabilities during the design and testing phases to ensure that the design meets their needs and is genuinely inclusive.
3. Accessibility Testing: Conduct thorough accessibility testing using automated tools (like axe or Lighthouse) and manual testing to identify and address any potential accessibility issues.

## 6. Training and Documentation:

1. User Training: Provide resources and training for users to understand and utilize accessibility features effectively. This may include tutorials, FAQs, and support materials.
2. Developer Guidelines: Create guidelines for developers to follow best practices in accessibility, ensuring that new features and updates maintain high accessibility standards.

# 6. Reused/Refactored Components

## Components:

### Reused Code:

Third-Party Libraries:

Identity Framework: Utilized for authentication and user management. This framework is standard in ASP.NET Core for handling user accounts and roles.

Swashbuckle.AspNetCore (Swagger UI): Used for API documentation and testing. Provides a user-friendly interface for exploring and interacting with API endpoints.

AutoMapper: Employed for object-to-object mapping, reducing boilerplate code when converting between entities and DTOs (Data Transfer Objects).

Entity Framework Core: Used for database interactions, including migrations and querying, to manage the application’s data persistence layer.

Bootstrap or Material-UI: Front-end libraries to ensure a responsive and modern UI design.

Custom Libraries or Modules:

Logging and Exception Handling: If previous projects have logging frameworks or error handling mechanisms, they can be reused to maintain consistency and reliability in error management.

# Refactored Code:

Existing Application Modules:

Account Management: Refactored to support both individual and business accounts. Enhancements may include changes to the account creation process, user roles, and profile management.

Support Request Management: Improved to handle new features like scheduling, status tracking, and integration with the inventory system. This may involve changes to request submission, tracking, and prioritization logic.

Inventory Management: Refactored to include real-time tracking and automated reordering. Existing inventory management logic might be extended to accommodate new requirements like inventory alerts and editing functionalities.

Data Analytics:

Reporting and Dashboard: Refactored or extended to provide advanced analytics, including trend analysis and geographical insights. Existing reports or dashboards might be updated to include new KPIs or visualizations.

Job Scheduling and Prioritization:

Job Management Logic: Updated to incorporate new prioritization rules and scheduling functionalities. This could involve modifications to existing job scheduling algorithms or UI components.

Knowledge Base:

Diagnostic Tools and Guides: Refactored to enhance usability and include additional tools or guides based on user feedback or new requirements.

## Detailed Description:

### Reused Code:

Identity Framework:

Description: Used for managing user authentication and authorization. Reused from previous projects to handle user registration, login, and role-based access.

Purpose: Provides a robust and secure method for user management without having to build these features from scratch.

Swashbuckle.AspNetCore (Swagger UI):

Description: A tool for generating interactive API documentation. Reused to facilitate API development and testing.

Purpose: Allows developers to easily explore and test API endpoints, ensuring that the API documentation is accurate and up-to-date.

AutoMapper:

Description: Library for mapping between domain models and DTOs. Reused to streamline the conversion process and reduce manual mapping code.

Purpose: Simplifies the process of transferring data between different layers of the application, improving code maintainability.

Entity Framework Core:

Description: ORM framework for data access. Reused for managing database operations and schema migrations.

Purpose: Provides a consistent approach to data access and manipulation, leveraging existing knowledge and practices.

Bootstrap or Material-UI:

Description: CSS frameworks for styling and layout. Reused to ensure a modern and responsive UI.

Purpose: Enhances the user interface with pre-built components and styles, reducing the need for custom CSS.

## Refactored Code:

### Account Management:

Description: Modified to support different account types and enhance user profile management.

Purpose: Provides flexibility to manage both individual and business accounts within the same system.

Support Request Management:

Description: Updated to handle new features such as repair scheduling and status tracking.

Purpose: Improves the efficiency and usability of the support request process, aligning with the new system requirements.

Inventory Management:

Description: Refactored to include real-time tracking and automated inventory management features.

Purpose: Enhances inventory control and management, ensuring better resource utilization and minimizing stockouts.

Data Analytics:

Description: Extended to include new analytical features and reporting capabilities.

Purpose: Provides deeper insights into business operations, helping management make informed decisions based on data.

Job Scheduling and Prioritization:

Description: Improved to support advanced scheduling and job prioritization rules.

Purpose: Optimizes technician workload and ensures timely resolution of urgent repair requests.

Knowledge Base:

Description: Updated to include additional diagnostic tools and instructional guides.

Purpose: Enhances the self-service capabilities for users, reducing the need for direct support interactions.

# Review and Feedback Record

## Review Process:

1. **Initial Review:**

Feedback Documentation:

Reviewer: Jafar

**Comments:** Jafar reviewed the initial software diagram documentation and provided feedback. He noted that the flow chart needed more details to align with the proposed solution described in the project proposal.

Changes Made:

**Action Taken:** Updated the flow chart to include additional details and ensure alignment with the proposal.

**Details Updated:** Expanded on system functionality and interactions as per the proposal.

1. **Justification:**

Design Decisions:

**Reasoning:** The flow chart was revised to include more comprehensive details to better match the proposed solution. This alignment ensures that the diagram accurately reflects the system’s design and functionality, improving clarity and coherence.

**Meeting Requirements:** The updated flow chart now captures all necessary components and interactions, providing a clear visual representation of the system in accordance with the proposal.

**Image Example:**

Feedback and Changes:

Table of Feedback and Updates:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Feedback from Jafar** |  |  | | --- | |  | | | **Changes Made** | | --- |  |  | | --- | |  | |
| |  | | --- | | Update the flow chart to include more details. |  |  | | --- | |  | | |  | | --- | | Expanded the flow chart to detail all system components and interactions as described in the proposal. |  |  | | --- | |  | |
| |  | | --- | | Ensure alignment with the proposed solution. |  |  | | --- | |  | | Adjusted the flow chart to reflect the specific functionalities and requirements outlined in the proposal. |

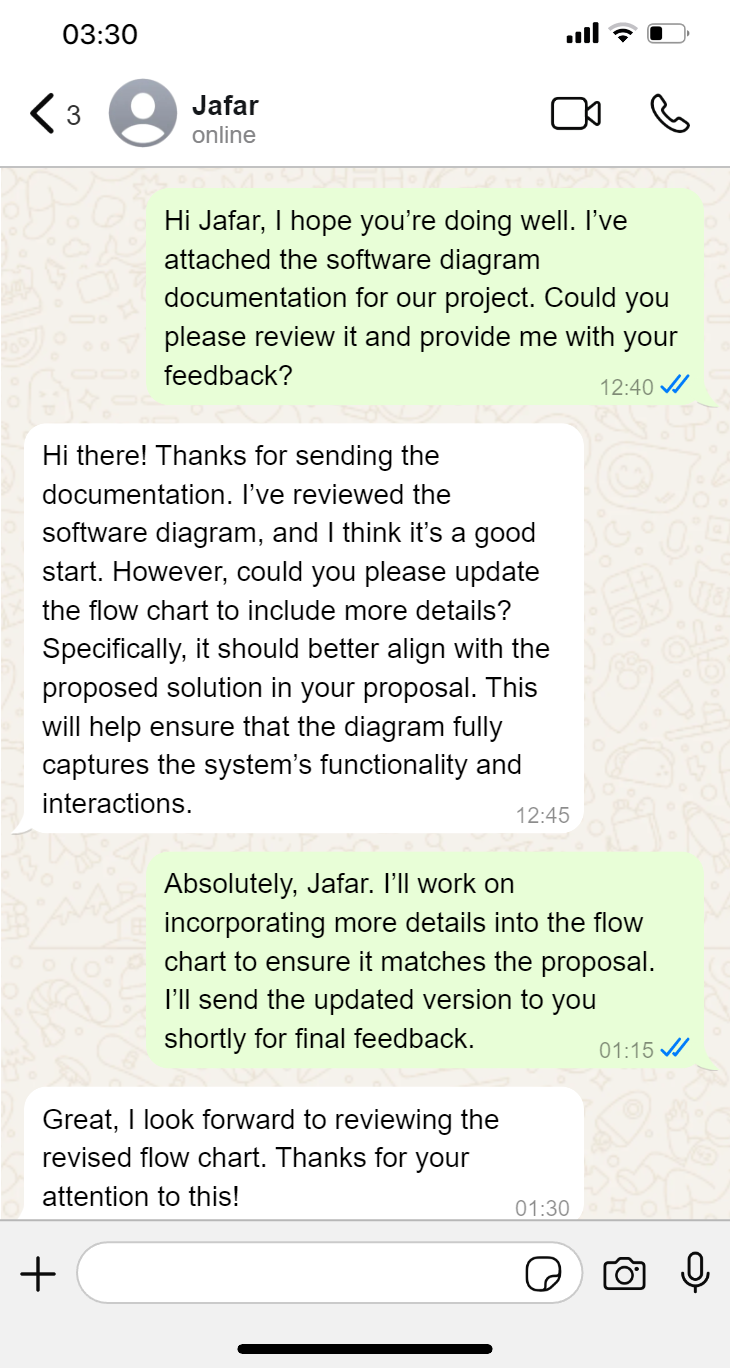
Justification:

Written Explanation:

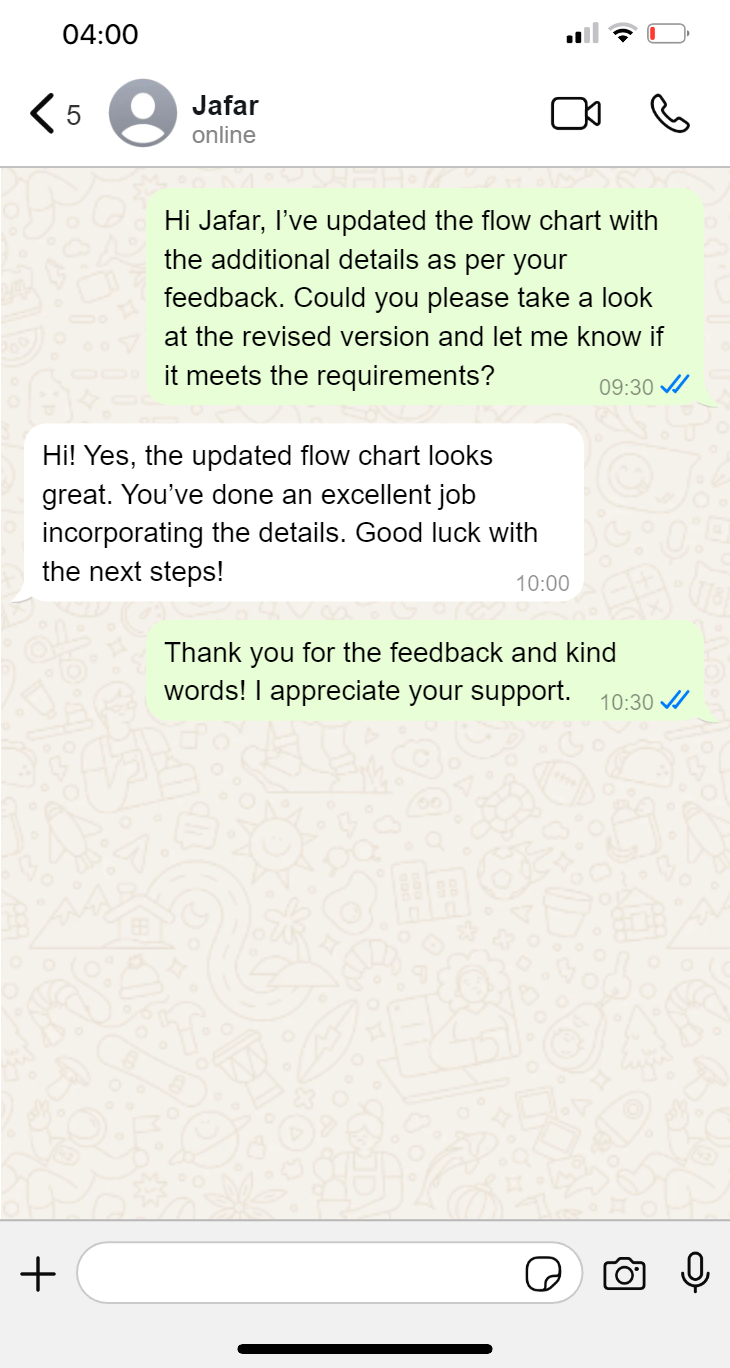
The flow chart was revised to address the feedback by incorporating additional details that align with the project proposal. This includes a more detailed breakdown of the system’s components and interactions, ensuring that the diagram accurately represents the proposed solution.

Visuals:

Attached are the updated flow chart images demonstrating the enhanced detail and alignment with the proposal.

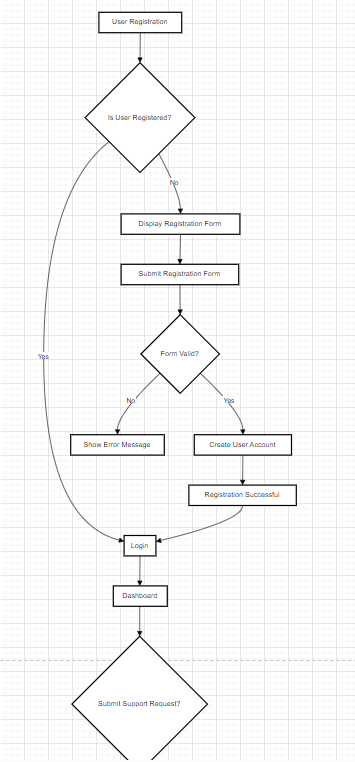
**Image**:  


Feedback One 1

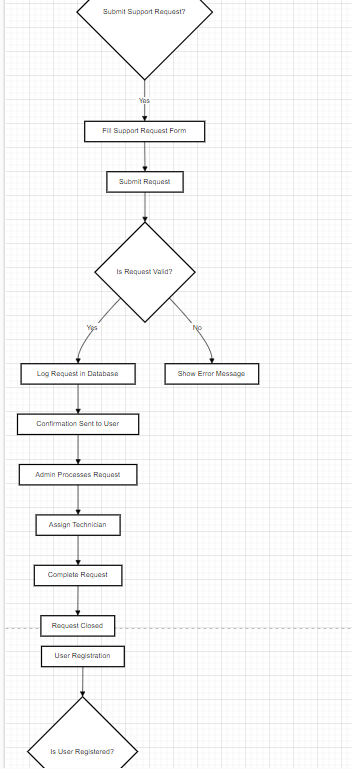


Feedback One 2

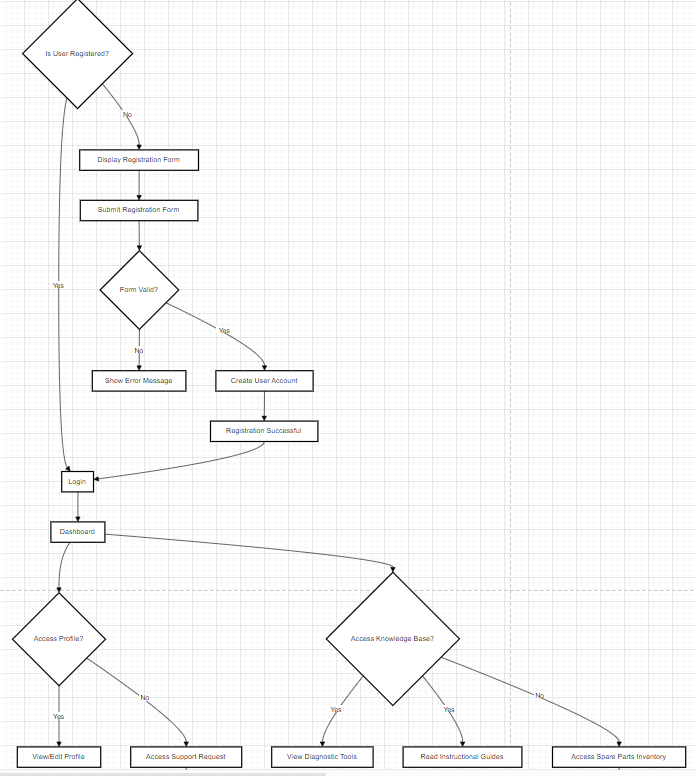
* New **Flow Chart:**



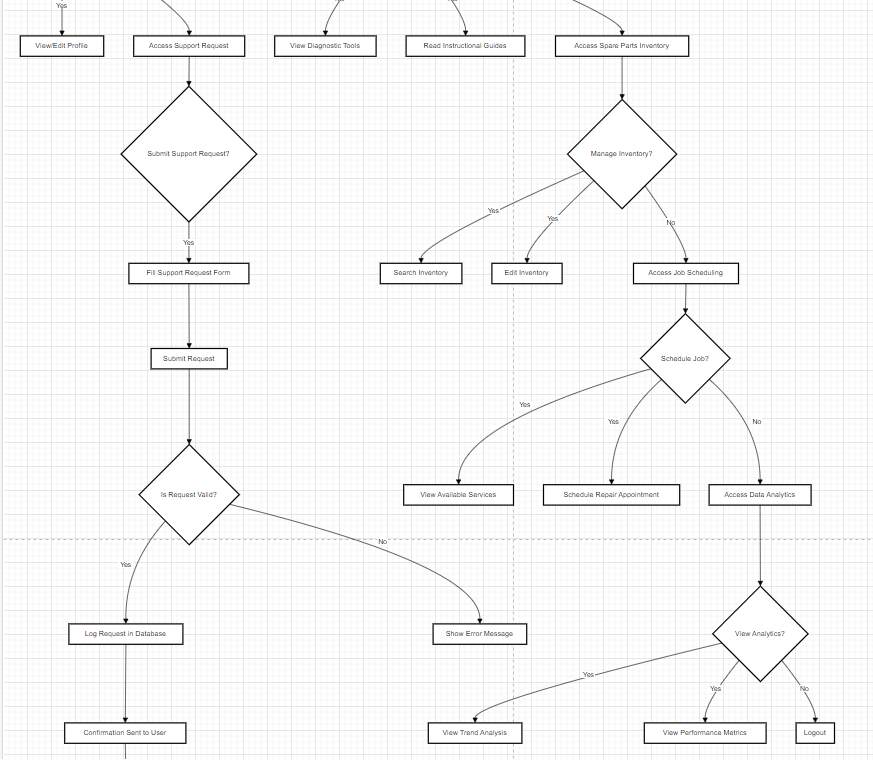
New Flow Chart 1



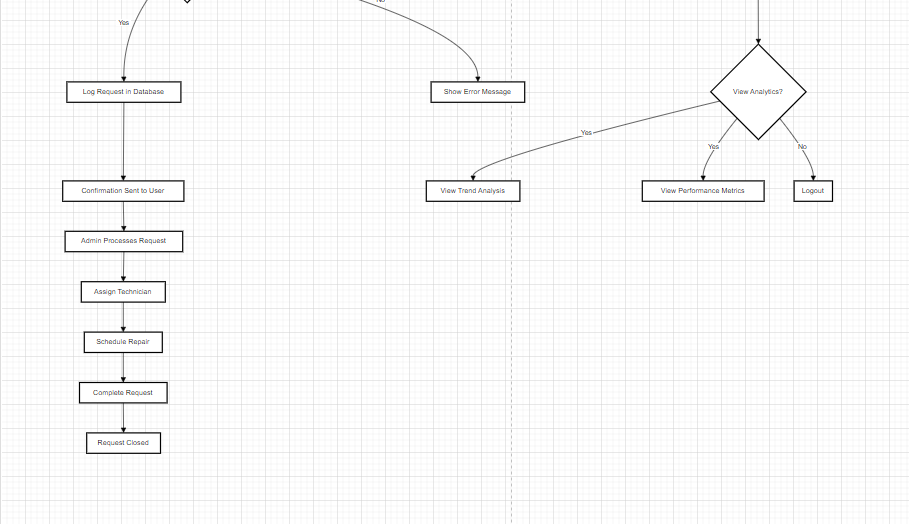
New Flow Chart 2



New Flow Chart 3



New Flow Chart 4



New Flow Chart 5

## Review Process:

1. **Initial Review:**

Feedback Documentation:

Reviewer: Raghad

**Comments:** Raghad reviewed the wireframes and requested that the search button be removed from all wireframes except for the IT Stocks page.

**Changes Made:**

**Action Taken:** Updated the wireframes to remove the search button from all pages, keeping it only on the IT Stocks page.

**Details Updated:** Removed the search button from all pages except for the IT Stocks page.

1. **Justification:**

Design Decisions:

Reasoning: The wireframes were revised based on Raghad's feedback to simplify the user interface and improve the browsing experience. Keeping the search button only on the IT Stocks page helps users find specific IT Stocks more easily without duplicating the search functionality across multiple pages.

Meeting Requirements: The changes make the user interface more organized and support the goal of providing a clearer user experience without overwhelming users with repeated search options.

**Image Example:**

Feedback and Changes:

Table of Feedback and Updates:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Feedback from Raghad** |  |  | | --- | |  | | | **Changes Made** | | --- |  |  | | --- | |  | |
| |  | | --- | | Remove the search button from all pages, keeping it only on the IT Stocks page. |  |  | | --- | |  | | Updated the wireframes to remove the search button from all pages except the IT Stocks page. |

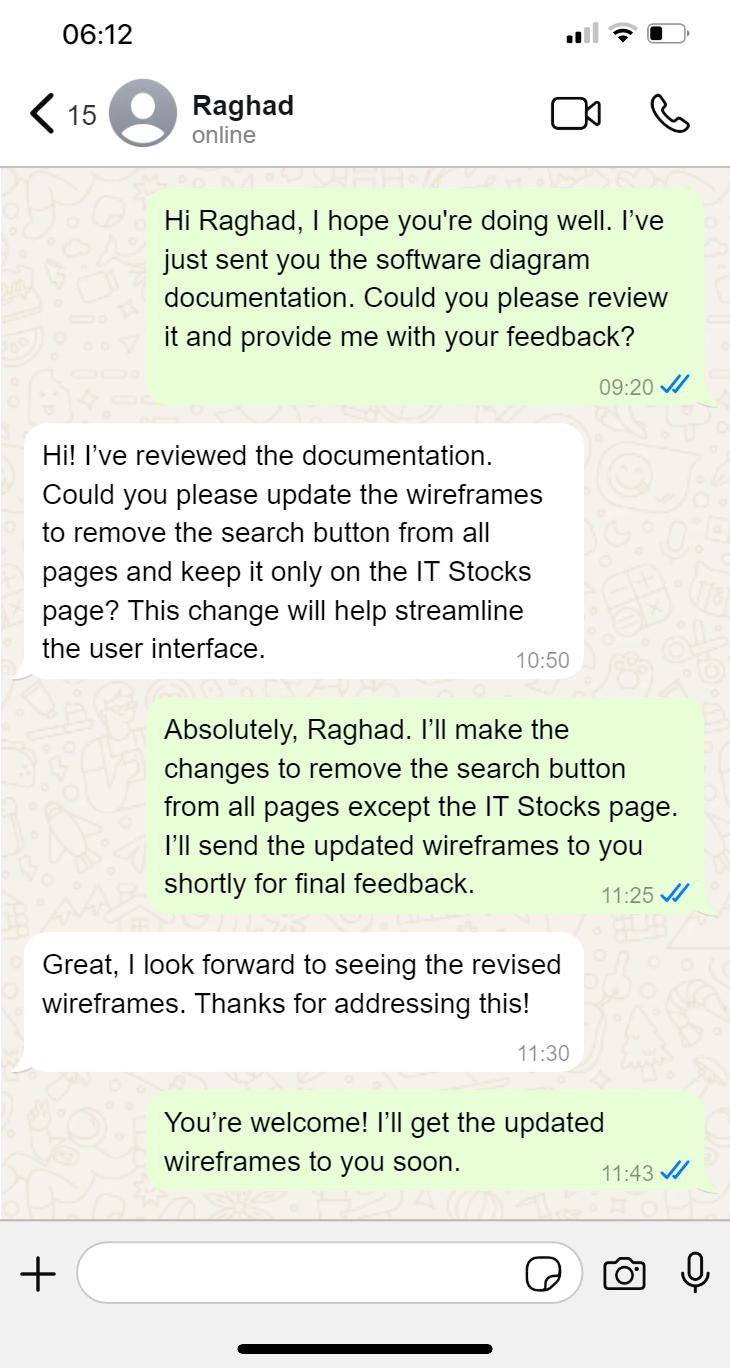
Justification:

Written Explanation:

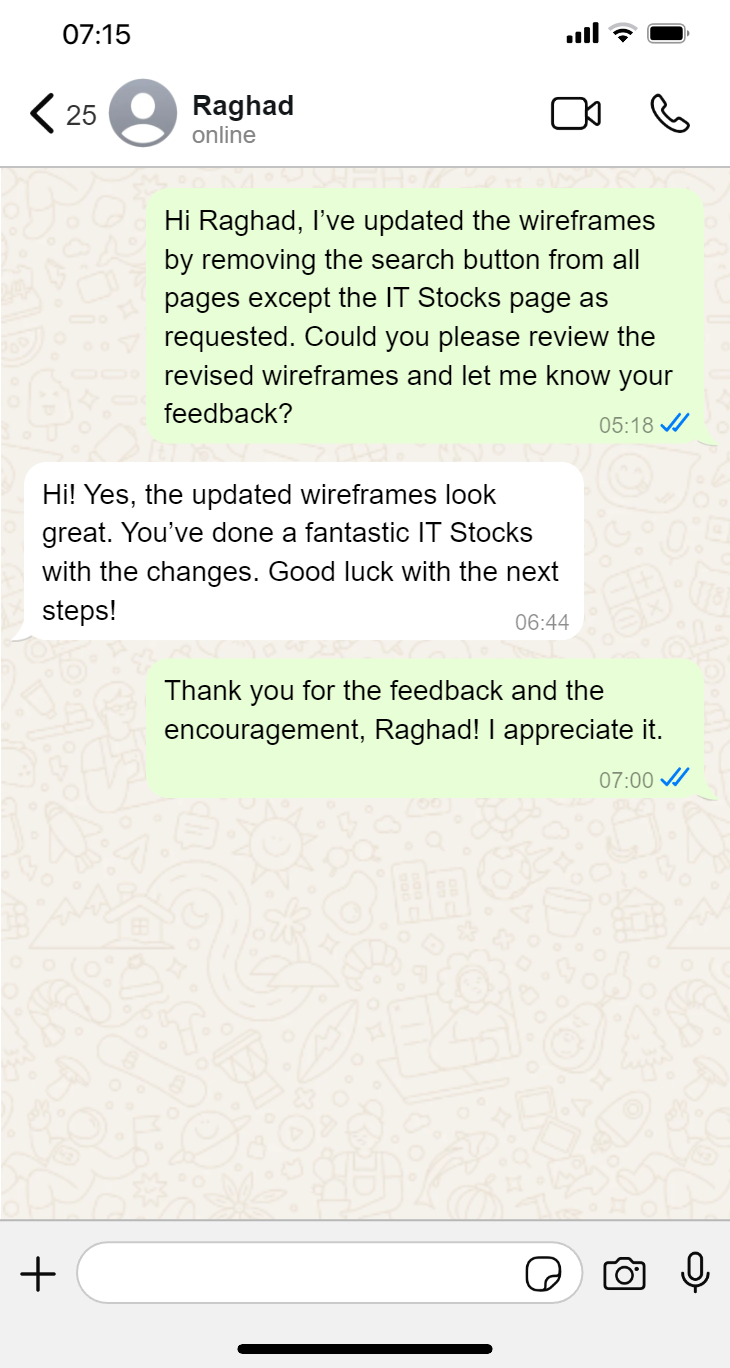
The wireframes were updated to reflect Raghad's feedback by removing the search button from all pages except the IT Stocks page. This change enhances the organization of the user interface and provides a clearer browsing experience.

Visuals:

Attached are the updated wireframes showing the removal of the search button from various pages, with it retained only on the IT Stocks page.

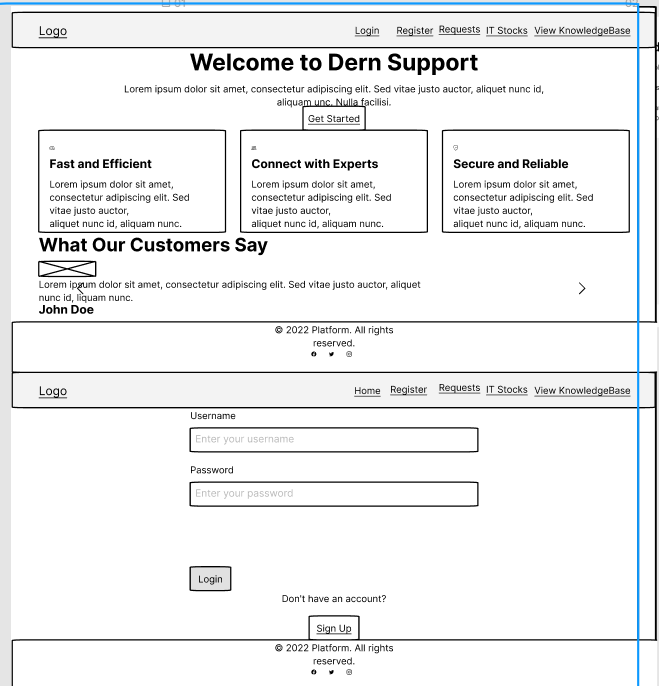


Feedback Two 1

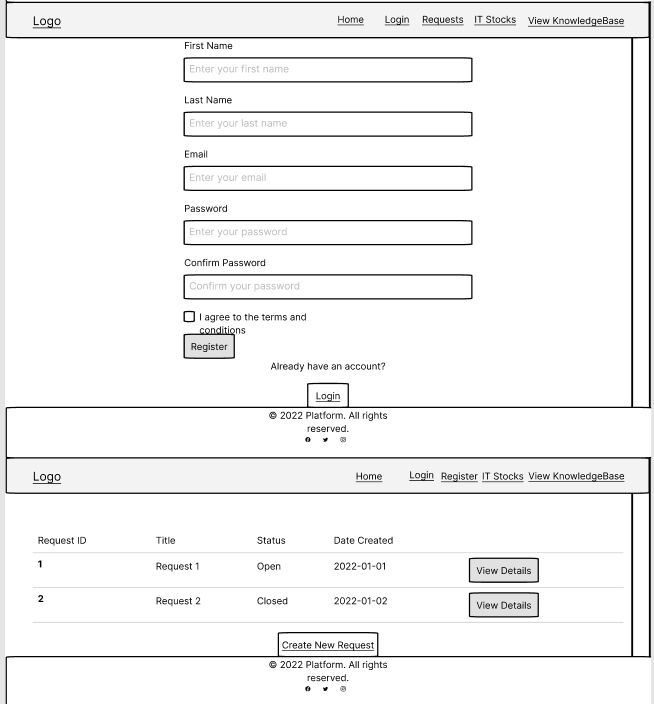


Feedback Two 2

New **Wireframes:**



Wireframes 1



Wireframes 2



Wireframes 3



Wireframes 4