SDD

Turn DevOps Workflows into Instant Documentation

www.copilot4devops.com

Document Type/Title	SDD
Document Prompt	Create a comprehensive Software Design Document (SDD) for the specified user story. Ensure that the document thoroughly outlines the architectural framework, design components, and technical specifications necessary for implementation. Include detailed diagrams, data flow representations, and any relevant algorithms to enhance understanding and clarity.
Regenerate Prompt	Create a comprehensive Software Design Document (SDD) for the specified user story. Ensure that the document thoroughly outlines the architectural framework, design components, and technical specifications necessary for implementation. Include detailed diagrams, data flow representations, and any relevant algorithms to enhance understanding and clarity.

Referenced Work Item(s)

ID	Title	Work Item Type
200	Edit a User	User Story

TABLE OF CONTENTS

SDD

1. Introduction

1.1 Purpose and Scope

This Software Design Document (SDD) details the design and implementation of the "Edit a User" functionality, as described by the user story. The document outlines the architectural framework, design components, and technical specifications necessary to implement the feature which allows an administrator to edit a user's information (email and role) through a well-defined sequence of operations.

1.2 Overview of the Document

This document is organized in a hierarchical structure covering:

1. The user story and its details.
2. The architectural framework of the system.
3. The design components and modules, including class and module outlines.
4. Diagrams, data flow representations, and relevant algorithms.
5. Technical specifications and implementation details.

---

2. User Story Overview

2.1 Work Item Details

- **ID:** 200
- **Title:** Edit a User
- **Work Item Type:** User Story
- **Acceptance Criteria:**
  - The system shall allow an administrator to navigate to Settings > Customize > Manage Users > Users.
  - The admin is able to click on an 'Edit' option to modify a user's email and role.
- **Description:**
  "How can I edit a user? Go to Settings > Customize > Manage Users > Users. To edit the information associated with a user, click on the 'Edit' option found there."

2.2 Java Implementation Overview

The provided Java implementation demonstrates the process of:

- Initializing a list of sample users.
- Displaying the current users.
- Allowing selection of a user based on username.
- Facilitating the editing of a user's email and role.
- Saving and displaying the updated user information.

---

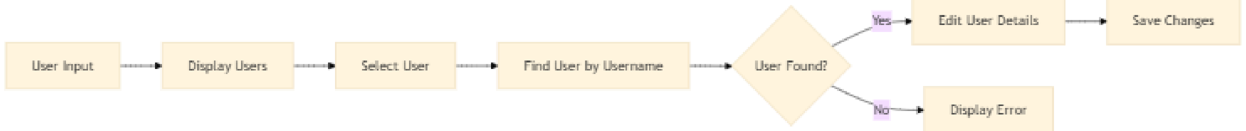
3. Architectural Framework

3.1 System Architecture

The core architecture of the system consists of the following components:

- **User Interface (Console Based):**
  Handles input and output with the administrator.
- **Business Logic Layer:**
  Contains the logic for searching, editing, and saving user information.
- **Data Model:**
  Implements the User class encapsulating user details (username, email, role).

The interaction flow is represented in the following diagram:



3.2 Technology Stack and Environment

- **Programming Language:** Java (JDK 8 or above)
- **User Interface:** Console-based interface for demonstration (with provisions for future extension to a web-based interface).
- **Development Tools:** Standard Java libraries with support for collections and I/O.
- **Deployment Environment:** Development and testing on local machines; production-ready environments would include persistent storage and enhanced authentication mechanisms.

---

4. Design Components

4.1 Modules and Components

The system is divided into several key components:

- **User Module:**
  - Contains the User class which encapsulates user attributes such as username, email, and role.
- **User Management Module:**
  - Contains the core functions to display users, find a user by username, and perform the edit operations.

4.2 Class Design and Structure

User Class:

- **Attributes:**
  - username (String)
  - email (String)
  - role (String)
- **Methods:**
  - Getters and setters for each attribute.
  - A toString() method for displaying user information.

UserManagement Class:

- **Attributes:**
  - An in-memory List to hold the users.
  - A Scanner to facilitate user input.
- **Methods:**
  - displayUsers(): Outputs the list of current users.
  - findUserByUsername(String username): Searches the user list.
  - editUser(User user): Prompts for and applies changes to a user’s email and role.
  - Main method: Coordinates the workflow from initialization through to the editing and updating of user information.

An HTML table representing the classes and their responsibilities is provided below:

Component	**Description**
User Class	Encapsulates user attributes (username, email, role) along with relevant getter, setter, and display methods.
UserManagement Class	Handles the initialization, display, selection, editing, and updating processes of the user data.

4.3 Algorithm and Logical Flow

The solution follows a clear series of steps:

1. **Initialize Users:**
   Create and populate the list of users with sample data.

2. **Display Users:**
   Output the current list of users for verification.

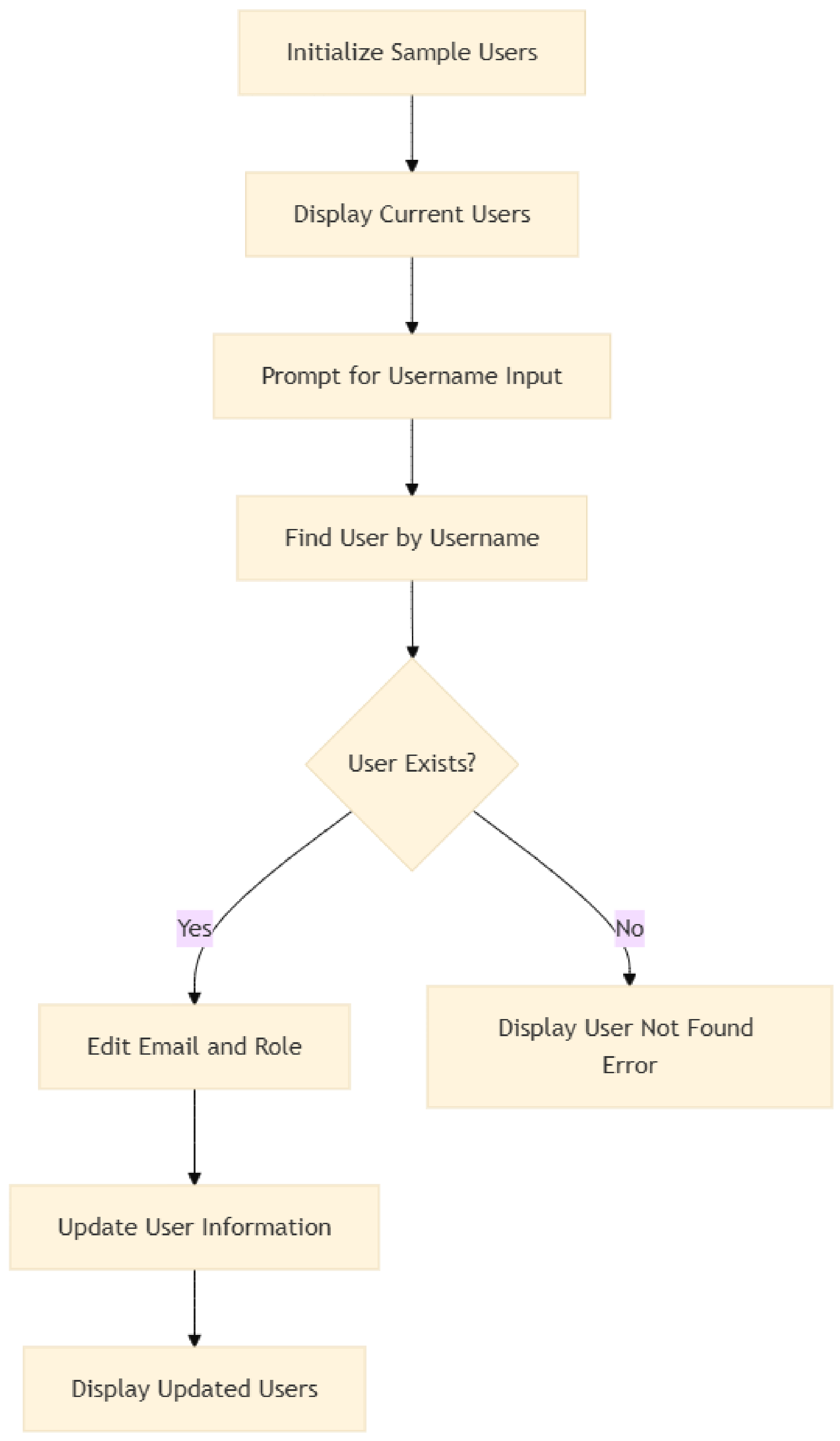
3. **Select User:**
   Prompt the administrator to enter the username of the user they wish to edit.

4. **Find User:**
   Search the in-memory list for the specified username.

5. **Edit User:**
   If the user is found, prompt for new email and role information.
   If the input field is left blank, keep the current value.

6. **Save Changes:**
   Update the user information and display the updated list.

The algorithm flow is visually summarized below:



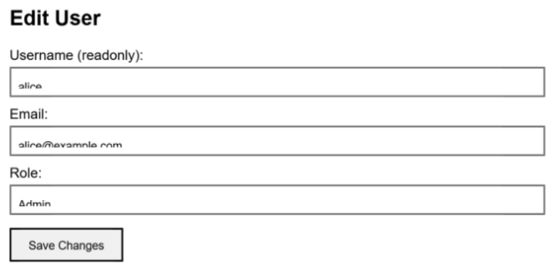
4.4 UI and Interaction Components

While the demonstration is based on a console interface, a web-based UI mock-up has been provided to illustrate how the edit user page could be structured.

HTML Mock-up for "Edit User" Page:

Edit User

Username (readonly): alice Email: alice@example.com Role: Admin Save Changes



This mock-up demonstrates a possible web interface where an administrator can view and update user details.

---

5. Technical Specifications

5.1 Programming Language and Tools

- **Language:** Java
- **Tools:** Java SDK (JDK 8 or above), standard Java IDE/editor.

5.2 Data Structures and Storage

- The user details are stored in an in-memory List structure.
- In a scalable or production solution, this would be replaced by database storage with appropriate CRUD operations integrated.

5.3 External Libraries and Frameworks

- Utilizes native Java libraries (java.util.ArrayList, java.util.Scanner, etc.) for collections and I/O.
- No third-party libraries are used in the current console-based demonstration.

5.4 Error Handling and Edge Cases

- **User Not Found:**
  The system handles scenarios where the entered username does not exist by displaying an error message.
- **Empty Input Handling:**
  When editing, if the admin leaves the email or role input blank, the current value is retained.
- **Future Considerations:**
  In a real application, additional validations such as email format checks, role-based access control, and robust error handling (e.g., try-catch blocks) would be implemented.

---

6. Implementation Plan

6.1 Code Structure and Modularization

The implementation is modularized into:

- **User Class Module:** Handles data encapsulation for user attributes.
- **UserManagement Module:** Manages the overall workflow (display, selection, editing, updating) of user operations.
  The main function organizes these modules and ensures a clear separation of concerns.

6.2 Integration Points and Dependencies

- **User Interface Integration:**
  For console-based applications, the Scanner is used to capture user input.
  In a web-based upgrade, the HTML/CSS/JavaScript front-end replaces the console operations.
- **Dependencies:**
  The system depends on the standard Java runtime environment. Future integration might include database connectors and authentication services.

6.3 Testing Strategy

- **Unit Testing:**
  Test individual methods in the User and UserManagement classes (e.g., findUserByUsername).
- **Functional Testing:**
  Verify that the complete workflow—from initializing to editing and saving user details—works as expected.

- **Manual Testing:**
  Use of console-based interaction for initial tests followed by UI tests using the provided HTML mock-up for web integration.

6.4 Deployment Considerations

- The current design is aimed at a demonstration environment.
- For production, the application would require enhancements such as:
    - Persistent storage (e.g., relational databases).
    - Enhanced security (authentication and authorization).
    - A graphical user interface (web-based or desktop application).

---

7. Conclusion

The document has detailed the comprehensive design of the "Edit a User" functionality in a hierarchical manner. It encompasses the architectural framework, design components, detailed algorithms, data flow diagrams, and technical specifications. This design meets the user story’s acceptance criteria by providing clear steps for initializing users, locating and editing a user, and saving the updated information. The modular design and outlined UI mock-up ensure that the solution is both understandable and extendable for future enhancements.