**Data Validation Application Documentation**

**1. System Design and Architecture**

**Overview**

The Data Validation Application is designed to facilitate user input validation for four key fields: name, email, phone number, and password. The application employs a graphical user interface (GUI) built using Java Swing, ensuring a user-friendly experience. The architecture follows a Model-View-Controller (MVC) pattern, separating the data handling logic from the user interface.

**Architecture**

* **Model**: Contains the data validation logic and state management.
* **View**: Represents the GUI components (JTextField, JLabel, JButton).
* **Controller**: Handles user interactions, updates the model, and updates the view based on user input.

**Components**

1. **JFrame**: The main window of the application.
2. **JPanel**: Contains the layout for the input fields and feedback labels.
3. **JTextField and JPasswordField**: Inputs for user data.
4. **JLabel**: Displays validation messages and feedback.
5. **JButton**: Submits the data for validation.

**User Flow**

1. User enters data into the input fields.
2. Real-time validation occurs on each key release.
3. Feedback is displayed immediately below each field.
4. On submission, the application checks all fields and provides a summary of validation results.

**2. Data Validation Mechanisms**

**Validation Rules**

The application implements specific validation rules for each input field:

1. **Name**:
   * Must be at least 3 characters long.
2. **Email**:
   * Must follow a standard email format.
   * Regular expression: ^[A-Za-z0-9+\_.-]+@(.+)$
3. **Phone Number**:
   * Must consist of exactly 10 digits.
   * Regular expression: \\d+ and length check.
4. **Password**:
   * Must contain at least one uppercase letter, one lowercase letter, and one digit.
   * Minimum length of 8 characters.
   * Regular expression: ^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d).{8,}$

**Mechanism**

Each input field has a dedicated validation method that is triggered on the keyReleased event. This allows for real-time feedback, enhancing the user experience and enabling users to correct errors as they type.

**3. Implementation Details**

**Code Snippets**

Here is the main structure of the application with key code snippets:

**Main Class**

java

Copy code

public final class DataValidationApp extends JFrame {

// UI components

JTextField nameField, emailField, phoneField;

JPasswordField passwordField;

JLabel feedbackLabel;

public DataValidationApp() {

setTitle("Data Validation System");

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

initComponents();

setVisible(true);

}

void initComponents() {

// Initialize components and layout

// Add listeners for validation

}

}

**Validation Methods**

java

Copy code

void validateName() {

String name = nameField.getText();

if (name.length() < 3) {

feedbackLabel.setText("Name must be at least 3 characters long.");

} else {

feedbackLabel.setText("");

}

}

void validateEmail() {

String email = emailField.getText();

String emailRegex = "^[A-Za-z0-9+\_.-]+@(.+)$";

if (!email.matches(emailRegex)) {

feedbackLabel.setText("Invalid email format.");

} else {

feedbackLabel.setText("");

}

}

// Similar methods for validatePhone() and validatePassword()

**Submission Logic**

java

Copy code

submitButton.addActionListener(e -> {

if (isValidInput()) {

JOptionPane.showMessageDialog(this, "All inputs are valid!");

} else {

JOptionPane.showMessageDialog(this, "Please correct the errors.", "Error", JOptionPane.ERROR\_MESSAGE);

}

});

boolean isValidInput() {

return nameField.getText().length() >= 3 &&

emailField.getText().matches("^[A-Za-z0-9+\_.-]+@(.+)$") &&

phoneField.getText().length() == 10 &&

new String(passwordField.getPassword()).matches("^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d).{8,}$");

}

**4. Testing and Debugging Procedures**

**Testing Strategy**

The application underwent rigorous testing to ensure all components functioned as expected. The testing process included:

1. **Unit Testing**: Individual validation methods were tested with valid and invalid inputs to ensure accuracy.
2. **Integration Testing**: The interaction between input fields and validation logic was tested to verify real-time feedback.
3. **User Acceptance Testing**: Conducted with a small group of users to gather feedback on usability and error handling.

**Debugging Process**

During development, common debugging techniques were employed:

* **Print Statements**: Used to trace variable values and application flow.
* **IDE Debugger**: Step-through debugging was utilized to monitor the application's behavior in real-time.
* **JUnit Testing**: Automated tests for validation methods were created to quickly identify failures.