**Rules for create the dev doc:**

**# Developer Documentation Standards**

**## Documentation Storage Path**

**- All developer documentation must be stored in the `/docs/features/` directory**

**- File naming convention: Descriptive-Feature-Name.md using kebab-case**

**- When a user requests documentation generation with `@developer-documentation.mdc`, the system will automatically:**

**- Create the file in the `/docs/features/` directory**

**- Use the user-provided filename**

**- Follow all documentation standards outlined below**

**## Helper Scripts**

**- Utilize the provided helper scripts to create standardized documentation:**

**- Linux/Mac: `node .cursor/scripts/create-dev-doc.js "Feature Name"`**

**- Windows: `.cursor\scripts\create-dev-doc.bat "Feature Name"`**

**- These scripts will:**

**- Create the documentation file in the correct location**

**- Apply the standard template**

**- Add the current date to relevant sections**

**## Document Version & Last Updated**

**- Version must be specified in format: vX.X.X**

**- Last Updated date must be included in format: DD-MM-YYYY**

**## Author(s) & Engine**

**- Document author(s) must be identified**

**- Development engine or task must be specified**

**## Overview**

**- Project summary must explain purpose, scope, and primary functionality**

**## Task/User Story Description**

**- Brief description must outline the task, feature, or user story**

**- Example: "Implement user authentication and session management for the dashboard module"**

**## Project File Structure**

**- High-level directory structure with key folders and their roles must be included**

**## Task Directory**

**- Documentation must describe specific directories/modules relevant to the task**

**- Example: src/modules/auth/, src/services/payment/**

**## Flow / Lifecycle**

**- System's logic flow must be documented, including:**

**- Data flow**

**- Request lifecycle**

**- Component interactions**

**- State management (if applicable)**

**## Codebase Documentation**

**### Libraries & Packages Used**

**- Key external dependencies must be listed with purpose and version**

**### Code Conventions & Naming Standards**

**- Variable/function naming style (e.g., camelCase, PascalCase)**

**- Folder/module naming conventions**

**- Commenting guidelines**

**- File naming conventions (e.g., kebab-case for components)**

**## API Documentation**

**### Endpoint List**

**- Format must be: METHOD /endpoint — Brief Description**

**- Example: POST /api/v1/login — Authenticates user credentials**

**### Input Parameters & Validation Rules**

**- Expected input fields, types, and validation rules must be listed**

**### Response Format**

**- Sample successful response structure with field descriptions must be included**

**### Error Handling**

**- Common error codes and meanings must be documented**

**- Error response structure must be explained**

**### Authentication & Authorization**

**- Authentication method must be specified (e.g., JWT, OAuth2)**

**- Required headers or tokens must be documented**

**### Versioning Strategy**

**- API versioning pattern must be documented**

**## Database Schema**

**### Entity Relationship Diagram (ERD)**

**- Overview of entities and relationships (visual if available)**

**### Tables & Fields**

**- Main tables with key fields and data types must be described**

**### Relationships & Constraints**

**- Foreign keys, cascading rules, and data integrity measures must be documented**

**### Migrations & Version Control**

**- Migration tools used must be specified**

**- Migration file naming and structure conventions must be documented**

**## End-to-End Test Coverage**

**- E2E testing strategy must be described**

**- Instructions for running tests must be included**

**- Test coverage reports should be referenced if available**

**## Dependency Details**

**- Key dependencies must be documented with versions**

**## Task Changelogs**

**- Changes made must be documented with dates and descriptions**

## **Screenshot of Rule in Cursor AI IDE:**

## **Prompt Used to create a doc:**

**Please generate the Dev Doc for my last feature in Color change doc**

**Dev Documentation:**

# Login Page Color Reversion

## Document Version & Last Updated

- Version: v1.0.0

- Last Updated: 01-08-2023

## Author(s) & Engine

- Author(s): Simplita Dev Team

- Development Engine: Front-end Styling Reversion

## Overview

This feature reverts the login page of the Simplita-MVP-v2 application back to its original blue-teal color scheme, undoing the previous purple-pink gradient styling. The reversion restores the original visual identity while maintaining all functionality of the login components.

## Task/User Story Description

- Revert the login page color scheme from purple-pink gradient back to the original blue-teal gradient, including all related UI elements to maintain design consistency.

## Project File Structure

```

Simplita-MVP-v2/

├── frontend/

│   ├── src/

│   │   ├── components/

│   │   │   ├── auth/

│   │   │   │   ├── AuthForm.tsx          # Main login form component (reverted)

│   │   │   │   ├── ForgotPasswordModal.tsx

│   │   │   │   └── login/

│   │   │   │       ├── FormStyles.ts

│   │   │   │       ├── PasswordUtils.ts

│   │   │   │       └── ...

```

## Task Directory

- `frontend/src/components/auth/` - Contains authentication-related components

- `frontend/src/components/auth/AuthForm.tsx` - Main login component that was modified

## Flow / Lifecycle

- \*\*Component Interaction\*\*:

  - AuthForm component renders both sign-in and sign-up forms

  - The reverted color scheme affects the gradient background, form input focus states, checkboxes, links, and buttons

- \*\*State Management\*\*:

  - No changes to state management logic

  - Only visual styling was updated

## Codebase Documentation

### Libraries & Packages Used

- React (v18.x) - UI component library

- Next.js (v13.x) - React framework

- Framer Motion (v10.x) - Animation library

- Tailwind CSS (v3.x) - Utility-first CSS framework

- Lucide React (v0.x) - Icon library

### Code Conventions & Naming Standards

- Component files: PascalCase (e.g., `AuthForm.tsx`)

- Utility files: camelCase (e.g., `formStyles.ts`)

- CSS classes: kebab-case for custom classes, otherwise following Tailwind conventions

- Color values: Tailwind color codes with hex values where custom colors are needed

## Changes Made

### Color Updates

1. \*\*Background Gradient\*\*:

   - Changed from: `from-[#6366F1] to-[#EC4899]` (purple to pink)

   - Reverted to: `from-[#0056D6] to-[#00D1A9]` (blue to teal)

2. \*\*Form Elements\*\*:

   - Input focus rings: Changed from purple (`focus:ring-[#6366F1]`) back to teal (`focus:ring-[#00D1A9]`)

   - Checkbox color: Updated from `text-[#6366F1] focus:ring-[#6366F1]` to `text-[#00D1A9] focus:ring-[#00D1A9]`

   - "Forgot password" link: Updated from `text-[#6366F1] hover:text-[#4F46E5]` to `text-[#00D1A9] hover:text-[#00A88A]`

3. \*\*Button Styling\*\*:

   - Changed from: Gradient button (`bg-gradient-to-r from-[#6366F1] to-[#EC4899]`)

   - Reverted to: Solid teal button (`bg-[#00D1A9]`) with hover effect (`hover:bg-[#00A88A]`)

### Code Implementation

The changes were implemented by updating the Tailwind CSS classes in the AuthForm.tsx component. No structural changes were made to the component's functionality. The reversion fully restores the original styling without introducing any new design elements.

## Related Documentation

- The original color change feature is documented in [Login-Changes.md](./Login-Changes.md)

## Task Changelogs

- 01-08-2023:

  - Reverted login page background gradient from purple-pink to original blue-teal color scheme

  - Reverted form input focus states to original teal colors

  - Reverted checkbox and link colors to original teal colors

  - Reverted button styling from gradient to original solid teal with hover effect