Documentación Técnica: Implementación del Flujo de Autenticación

Estructura y Organización del Proyecto

Lo tenemos dividido principalmente en 5 carpetas:

1. components
2. hooks
3. store
4. types
5. tests

components/

Contiene todos los componentes visuales de React. Separamos la lógica de presentación del resto de la aplicación, esto facilita la reutilización de componentes y permite un desarrollo más modular y mantenible

hooks/

Almacena todos los hooks personalizados.Permite compartir lógica entre diferentes componentes

store/

Contiene toda la lógica de estado global, esto mejora la mantenibilidad al tener un único lugar para la lógica de estado

types/

Define todos los tipos TypeScript de la aplicación.

tests/

Contiene todos los archivos de pruebas.

Esta estructura sigue las mejores prácticas de desarrollo moderno en React. Evita la mezcla de diferentes tipos de código, está preparada para crecer y añadir nuevas características y está organizado y predecible para otros desarrolladores, facilitando el mantenimiento, la escalabilidad y la colaboración en el proyecto.

Calidad del Código

Principios de Diseño Aplicados

Single Responsibility Principle: Cada componente y hook tiene una única responsabilidad

DRY (Don't Repeat Yourself): Lógica común extraída a hooks y utilidades

Composición sobre Herencia: Componentes modulares y reutilizables

Se usó AuthFlow como componente orquestador que maneja la lógica de flujo de estados. Cada pantalla es un componente independiente y reutilizable

Estructura de Tipos y Estados

```
export type AuthStep = 'landing' | 'phone-input' | 'verification' |
'complete';

export interface PhoneState {
  value: string;
  isValid: boolean;
  error?: string;
}

export interface VerificationState {
  code: string[];
  timeRemaining: number;
  attempts: number;
  canResend: boolean;
}

export interface AuthState {
  step: AuthStep;
  phone: PhoneState;
  verification: VerificationState;
  isLoading: boolean;
  error?: string;
}
```

LoginLanding:Estado mínimo, el primero al cargar la pagina
PhoneInput: Estados de validación
VerificationCode: Estados de código
LoadingScreen: Estado de carga global

El tipado es fuerte para prevenir errores y mejorar la mantenibilidad.

Estrategia de Estado Global

Se eligió Zustand sobre otras alternativas por:
- Mínima configuración necesaria
- Excelente integración con TypeScript
- Facilidad para testing
- Mejor rendimiento en comparación con Redux para aplicaciones pequeñas/medianas

Custom Hooks Architecture

usePhoneValidation.ts gestiona el formato y la validación del número de teléfono. Este hook se integra con el estado global de Zustand, al tiempo que mantienen el estado local para las actualizaciones inmediatas de la interfaz de usuario, creando una base de código adaptable y fácil de mantener.

Estrategia de Testing

Las pruebas se escriben con Vitest y React Testing Library, haciendo hincapié en probar el comportamiento sobre la implementación. Cada archivo de prueba incluye una ruta feliz y escenarios de error, con utilidades personalizadas para la verificación del estado y la imitación de la API.

Este enfoque equilibra las pruebas exhaustivas con el código mantenible, garantizando la fiabilidad al tiempo que sirve como documentación clara para otros desarrolladores.