Guía de Implementación de Prompts: Sistema de Reservas Empresarial

1. Introducción y Metodología de Ingeniería de Prompts

En el desarrollo de software moderno, la Inteligencia Artificial no actúa como un sustituto del ingeniero, sino como un **Copiloto** de alto rendimiento. En este **Proyecto 3: Sistema de Reservas Empresarial** , el desarrollador asume el rol de **Piloto** , decidiendo la arquitectura y validando la lógica, mientras que la IA acelera la escritura de código y sugiere soluciones técnicas.Para garantizar resultados precisos y evitar "alucinaciones" (resultados inconsistentes o técnicamente erróneos), utilizaremos una estructura de prompt basada en cinco pilares fundamentales:

1. **Contexto:** Define el entorno del problema (SPA, Vanilla JS, localStorage).
2. **Rol:** Establece la identidad y nivel de experiencia de la IA (Senior Software Architect).
3. **Tarea:** La instrucción central, directa y específica.
4. **Restricciones:** Límites técnicos (sin frameworks, validaciones obligatorias, seguridad).
5. **Formato:** Cómo debe entregarse la información (código comentado, esquemas JSON, explicaciones técnicas).Esta metodología asegura que el sistema multirrol (Administrador, Operador, Cliente) sea robusto y escalable. A continuación, se presenta la hoja de ruta de los 10 prompts que componen el núcleo del proyecto.

Tabla Resumen de Prompts

Fase,Prompt,Objetivo Técnico
Arquitectura,1. Diseño de Arquitectura,Definir la estructura SPA y el flujo de renderizado reactivo.
Arquitectura,2. Modelado de Datos,Crear esquemas JSON para persistencia en localStorage.
Autenticación,3. Lógica de Autenticación,"Registro y Login con manejo de errores ""Early Return""."
Autenticación,4. Protección de Vistas,"Implementar un ""Navigation Guard"" centralizado."
Gestión (CRUD),5. Creación de Reservas,Registro con validación de fechas y variables descriptivas.
Gestión (CRUD),6. Cambio de Estados,Lógica de negocio para transiciones de estado permitidas.
Gestión (CRUD),7. Historial y Eliminación,Gestión selectiva de datos según el rol del usuario.
Visualización,8. Panel Estadístico,Dashboard desacoplado (procesamiento vs. renderizado).
Visualización,9. Optimización Tailwind,Interfaz responsive y accesible (ARIA labels).
Calidad,10. Refactorización,Estrategias de limpieza y gestión de cuotas de storage.

2. Fase 1: Arquitectura y Modelado de Datos

Prompt 1: Diseño de la Arquitectura del Sistema

- **Contexto:** Estoy iniciando el desarrollo de una Single Page Application (SPA) para un sistema de reservas empresarial.
- **Rol:** Actúa como un Arquitecto de Software Senior experto en JavaScript Vanilla.

- **Tarea:** Diseña la estructura de archivos del proyecto y explica detalladamente cómo los cambios en el estado de localStorage deben activar el re-renderizado de la interfaz en un entorno sin frameworks.
- **Restricciones:** No utilices frameworks (React, Vue, etc.). Sugiere un patrón de "Centralized Render Function" o el uso de "Custom Events" para la comunicación entre componentes.
- **Formato:** Entrega la estructura de carpetas en un bloque de código y una explicación técnica del "Why" (el porqué) de la arquitectura elegida.

Prompt 2: Definición de Modelos de Datos en LocalStorage

- **Contexto:** Necesito definir la persistencia de datos en el navegador para el sistema de reservas.
- **Rol:** Ingeniero de Datos Senior.
- **Tarea:** Redacta esquemas JSON para las entidades 'Usuarios' (Admin, Operador, Cliente) y 'Reservas' (ID, UID, fecha, hora, estado, descripción).
- **Restricciones:** La estructura debe ser óptima para operaciones de búsqueda (.find()) y filtrado (.filter()). Incluye campos necesarios para la trazabilidad de estados.
- **Formato:** Código JSON de ejemplo con datos ficticios y una breve explicación técnica de la jerarquía de los objetos.

3. Fase 2: Autenticación y Seguridad por Roles

Prompt 3: Lógica de Autenticación y Registro

- **Contexto:** El sistema requiere un flujo de acceso seguro donde cada usuario tenga permisos específicos.
- **Rol:** Especialista en Ciberseguridad y Desarrollo Web.
- **Tarea:** Crea las funciones de JavaScript para register y login. Implementa la lógica de validación de credenciales consultando localStorage.
- **Restricciones:** 1. Utiliza obligatoriamente el **patrón Early Return** para gestionar errores de validación. 2. Usa **variables intermedias descriptivas** (ej. esPasswordValido) para mejorar la legibilidad. 3. Incluye una advertencia sobre los riesgos de seguridad de almacenar contraseñas en localStorage y sugiere cómo simular un entorno seguro en el frontend.
- **Formato:** Funciones de JavaScript comentadas. Al final, explica el razonamiento técnico detrás del manejo de errores.

Prompt 4: Protección de Vistas y Rutas (Navigation Guard)

- **Contexto:** Debo evitar que usuarios sin permisos accedan a secciones críticas como el Panel Estadístico.
- **Rol:** Arquitecto Frontend Senior.
- **Tarea:** Diseña un script que actúe como un "Navigation Guard". Debe interceptar cada cambio de vista en la SPA, verificar el rol del currentUser en localStorage y redirigir si no cumple con los privilegios.
- **Restricciones:** Crea un objeto routeConfig que mapee rutas a roles permitidos. Si el acceso es denegado, redirige a la vista por defecto del rol correspondiente.

- **Formato:** Código JavaScript modular y una explicación de por qué este enfoque centralizado es superior a validar en cada componente.

4. Fase 3: Gestión de Reservas (CRUD) y Validaciones

Prompt 5: Creación de Reservas con Validación de Fechas

- **Contexto:** Un Cliente debe registrar una reserva mediante un formulario.
- **Rol:** Desarrollador Senior de Software.
- **Tarea:** Implementa la función createBooking con validaciones de lógica de negocio.
- **Restricciones:** 1. No permitir fechas pasadas ni solapamientos horarios. 2. Aplica el **patrón Early Return** . 3. Utiliza **variables descriptivas** para las condiciones de validación (ej. esFechaFutura, hayConflictoDeHorario).
- **Formato:** Código JavaScript limpio y una explicación de cómo se garantiza la integridad de los datos en el storage.

Prompt 6: Gestión y Cambio de Estados (Operador/Admin)

- **Contexto:** Los Operadores deben gestionar el ciclo de vida de las reservas.
- **Rol:** Desarrollador de Lógica de Negocio.
- **Tarea:** Crea una función para actualizar el estado de una reserva (Pendiente, Confirmada, Cancelada).
- **Restricciones:** Implementa lógica de estados profunda: Una reserva "Cancelada" no puede pasar a "Confirmada" directamente; debe ser "Reprogramada" (cambio de fecha) primero para volver a estar activa. Solo roles 'Operador' o 'Admin' pueden ejecutar esto.
- **Formato:** Código JavaScript y un diagrama de flujo textual que explique las transiciones permitidas.

Prompt 7: Eliminación y Consulta de Historial

- **Contexto:** Se requiere limpieza de datos y visualización para el usuario final.
- **Rol:** Desarrollador Senior.
- **Tarea:** Desarrollar funciones para que el Administrador elimine registros y el Cliente consulte exclusivamente su propio historial.
- **Restricciones:** Utiliza métodos funcionales de JavaScript (.filter). Asegura que tras eliminar, se actualice el estado global de la aplicación para reflejar los cambios en el DOM.
- **Formato:** Bloques de código independientes para cada funcionalidad con comentarios explicativos.

5. Fase 4: Visualización de Datos y Optimización de Interfaz

Prompt 8: Panel Estadístico Desacoplado

- **Contexto:** El Administrador necesita métricas clave del negocio.
- **Rol:** Senior Frontend Architect.
- **Tarea:** Implementa un dashboard estadístico que muestre: total de reservas, conteo por estado y usuarios más activos.

- **Restricciones:** Desacopla la lógica en dos partes: 1. Una función de procesamiento de datos que devuelva un objeto con métricas. 2. Una función de renderizado que reciba ese objeto y manipule el DOM. Usa variables descriptivas.
- **Formato:** Código JavaScript estructurado y la explicación técnica de por qué desacoplar el procesamiento del renderizado.

Prompt 9: Optimización Estética y UX con Tailwind CSS

- **Contexto:** La aplicación es funcional pero necesita una apariencia profesional.
- **Rol:** Diseñador UX/UI y Desarrollador Frontend.
- **Tarea:** Transforma un HTML básico de lista de reservas y formularios en una interfaz moderna y limpia usando Tailwind CSS.
- **Restricciones:** 1. Debe ser totalmente Responsive. 2. Incluye **etiquetas ARIA (Accesibilidad)** para lectores de pantalla. 3. Usa una paleta de colores empresarial y estados visuales claros (ej. botones disabled o alertas de error).
- **Formato:** Código HTML5 con clases de Tailwind e indicaciones de por qué estas decisiones mejoran la experiencia de usuario.

6. Fase 5: Calidad de Código y Depuración

Prompt 10: Refactorización y Gestión de Quota de Storage

- **Contexto:** El sistema está listo, pero el código de localStorage requiere una revisión de calidad.
- **Rol:** Ingeniero de QA y Optimización.
- **Tarea:** Revisa el código de persistencia para identificar redundancias y optimizar el uso del espacio.
- **Restricciones:** 1. Implementa una estrategia de "Storage Quota Check" para asegurar que no se exceda el límite de 5MB del navegador. 2. Sugiere un método de limpieza de datos antiguos (reservas pasadas de más de un año). 3. No cambies la arquitectura, solo optimiza la implementación actual.
- **Formato:** Comparativa "Antes/Después" de fragmentos de código y explicación del impacto en el rendimiento.

7. Guía de Uso y Recomendaciones Finales

Checklist de Pre-Producción

- **Validación Rigurosa:** ¿Se aplicó el patrón Early Return en todos los formularios para prevenir datos corruptos?
- **Privacidad de Datos:** ¿Se ha evitado el almacenamiento de Información Personal Identificable (PII) sensible sin las debidas advertencias de seguridad?
- **Gestión de Cuota:** ¿Existe una función que verifique si hay espacio suficiente en localStorage antes de escribir?
- **Accesibilidad:** ¿Todos los elementos interactivos cuentan con roles ARIA y contraste de color adecuado?

Reflexión sobre la Iteración

El éxito de este sistema radica en el ciclo **"Itera y Refina"** . Como arquitectos, no debemos aceptar el primer bloque de código que entrega la IA. Al cuestionar el "por qué", exigir

patrones como el Early Return y forzar el desacoplamiento de funciones, transformamos una herramienta de autocompletado en un socio de ingeniería. La calidad del software resultante es, en última instancia, un reflejo de la precisión y el criterio técnico del piloto al mando de los prompts.