Prompts utilizados para generación de código de frontend

La siguiente tabla describe el escenario como la necesidad de una solución, el requerimiento como el paso a paso para la implementación del patrón Factory Method, y por último el prompt que contiene la instrucción detallada para generar código.

Tabla 1. *Requerimientos y prompts para el patrón Factory Method*

No.	Escenario	Requerimiento	Prompt
1	Una plataforma de noticias maneja artículos de tipo noticia, columna de opinión y reportaje.	- Implementar un factory method para generar tarjetas de artículos con distintos diseños. - NewsArticle debe mostrar fecha y categoría. - OpinionArticle debe incluir autor destacado. - ReportArticle debe mostrar un resumen especial. - La fábrica debe permitir agregar nuevos formatos sin modificar la lógica existente.	Crea un factory method ArticleFactory.createArticle(type, data), que devuelva NewsArticle, OpinionArticle o ReportArticle, todas heredando de ArticleCard. Implementa render(). Define todas las clases y la fábrica en un solo archivo y expórtalas. Utiliza la sintaxis de módulos ES (export).
2	Un e-commerce maneja productos estándar, en oferta y exclusivos.	- Implementar un factory method para generar tarjetas de productos según su categoría. - DiscountProduct debe incluir porcentaje de descuento. - ExclusiveProduct debe mostrar sello de "Edición Limitada". - La fábrica debe estar desacoplada del código que muestra los productos.	Crea un factory method ProductFactory.createProductCard(type, data), que devuelva StandardProduct, DiscountProduct o ExclusiveProduct, todas heredando de ProductCard. Implementa render(). Define todas las clases y la fábrica en un solo archivo y expórtalas. Utiliza la sintaxis de módulos ES (export).
3	Una red social maneja perfiles estándar, creadores de contenido y moderadores.	- Implementar un factory method para tarjetas de perfil. - CreatorProfile debe resaltar el número de seguidores. - ModeratorProfile debe incluir sus permisos especiales. - La fábrica debe permitir agregar nuevos tipos de perfiles sin modificar su código base.	Crea un factory method UserProfileFactory.createProfile(type, data), que devuelva StandardProfile, CreatorProfile o ModeratorProfile, todas heredando de UserProfileCard. Implementa render(). Define todas las clases y la fábrica en un solo archivo y expórtalas. Utiliza la sintaxis de módulos ES (export).

4	Una plataforma de educación maneja cursos gratuitos, premium y en promoción.	- Implementar un factory method para generar tarjetas de cursos. - PremiumCourse debe destacar si incluye certificación. - La fábrica debe cumplir con el Principio de Inversión de Dependencias (DIP).	Crea un factory method CourseFactory.createCourseCard(type, data), que devuelva FreeCourse, PremiumCourse o PromoCourse, todas heredando de CourseCard. Implementa render(). Define todas las clases y la fábrica en un solo archivo y expórtalas. Utiliza la sintaxis de módulos ES (export).
5	Un dashboard empresarial maneja reportes de ventas, costos y rendimiento de empleados.	- Implementar un factory method para tarjetas de reportes. - SalesReport debe incluir gráfico de ventas. - PerformanceReport debe mostrar KPIs de empleados. - La fábrica debe permitir la adición de nuevos tipos de reportes sin afectar su código existente.	Crea un factory method ReportFactory.createReportCard(type, data), que devuelva SalesReport, CostsReport o PerformanceReport, todas heredando de ReportCard. Implementa render(). Define todas las clases y la fábrica en un solo archivo y expórtalas. Utiliza la sintaxis de módulos ES (export).
6	Un servicio de streaming maneja contenido de películas, series y documentales.	- Implementar un factory method para generar tarjetas de contenido. - MovieCard, SeriesCard y DocumentaryCard deben mostrar información específica. - La fábrica debe permitir la integración de nuevos formatos de contenido en el futuro.	Crea un factory method MediaFactory.createMediaCard(type, data), que devuelva MovieCard, SeriesCard o DocumentaryCard, todas heredando de MediaCard. Implementa render(). Define todas las clases y la fábrica en un solo archivo y expórtalas. Utiliza la sintaxis de módulos ES (export).

En particular en el análisis del patrón Observer es común hacer referencia a "User implementa Observer" el cual es más aplicable en lenguajes como Java, C# o TypeScript, donde las interfaces formales definen contratos claros, en estos lenguajes, Observer sería una interfaz con un método update, y User la implementaría explícitamente. Sin embargo, en JavaScript, al carecer de interfaces nativas, no soporta interfaces formales, ya que es un lenguaje dinámico basado en prototipos, por lo que se opta por agregar “Cada User debe tener un método update para recibir notificaciones.”.

Tabla 2. Requerimientos para el patrón Observer

No	Escenario	Requerimiento	Prompt
1	Una plataforma de noticias permite suscribirse a categorías específicas para recibir nuevos artículos en tiempo real.	- Implementar el patrón Observer para suscripción a categorías. - Los usuarios deben recibir notificaciones al publicarse nuevos artículos.	Crea una clase NewsCategory con subscribe(user) y publish(article). Al publicar, notifica a los observadores. Cada User debe tener un método update para recibir notificaciones. Define todo en un solo archivo y exporta. Usa ES modules.
2	Un e-commerce permite a los usuarios seguir productos para recibir alertas de baja de precio.	- Implementar el patrón Observer para seguimiento de productos. - Al cambiar el precio, los observadores deben recibir alertas.	Crea una clase Product con subscribe(user) y setPrice(price). Al cambiar el precio, notifica a los observadores. Cada User debe tener un método update para recibir notificaciones. Define todo en un solo archivo y exporta. Usa ES modules.
3	Una red social permite a los usuarios seguir a creadores de contenido para ver sus publicaciones al instante.	- Implementar el patrón Observer para seguir creadores. - Los usuarios deben recibir notificaciones cuando haya publicaciones nuevas.	Crea una clase ContentCreator con subscribe(user) y publish(post). Al publicar, notifica a los observadores. Cada User debe tener un método update para recibir notificaciones. Define todo en un solo archivo y exporta. Usa ES modules.
4	Un sistema de monitoreo de stock permite a los usuarios seguir productos para ser notificados cuando haya unidades disponibles en tiendas físicas cercanas.	- Implementar el patrón Observer para monitoreo de stock. - Los usuarios deben poder suscribirse a productos. - Al reponerse el stock, se debe notificar a los observadores.	Crea una clase StoreProduct con subscribe(user) y updateStock(quantity). Al haber stock disponible, notifica a los observadores. Cada User debe tener un método update para recibir notificaciones. Define todo en un solo archivo y exporta. Usa ES modules.
5	Un dashboard empresarial notifica a los gerentes sobre cambios en métricas clave en tiempo real.	- Implementar el patrón Observer para métricas del negocio. - Notificar automáticamente a los gerentes suscritos.	Crea una clase Metric con subscribe(manager) y update(value). Al actualizarse, notifica a los observadores. Cada Manager debe tener un método update para recibir notificaciones. Define todo en un solo archivo y exporta. Usa ES modules.

6	Una aplicación de colaboración en documentos permite ver los cambios en tiempo real.	- Implementar el patrón Observer para edición colaborativa. - Al modificarse el documento, todos los usuarios deben ver los cambios.	Crea una clase CollaborativeDocument con subscribe(user) y edit(content). Al editar, notifica a los observadores. Cada User debe tener un método update para recibir notificaciones. Define todo en un solo archivo y exporta. Usa ES modules.

Tabla 3 Requerimientos para el patrón strategy

No.	**Escenario**	**Requerimiento**	**Prompt**
1	Un reproductor multimedia permite elegir entre diferentes estrategias de reproducción: normal, en bucle o aleatoria.	- Aplicar Strategy para encapsular el comportamiento de reproducción. - Cambiar de modo dinámicamente desde la interfaz. - Cada estrategia debe implementar el método playNext().	Crea una clase MediaPlayer que acepte estrategias (NormalPlay, LoopPlay, ShufflePlay). Todas las estrategias deben definir el método playNext(playlist). Permite cambiar la estrategia en tiempo real. Define las clases en un solo archivo y expórtalas. Usa ES modules.
2	Un chatbot adapta su estilo de respuesta según el tono elegido: formal, casual o técnico.	- Aplicar Strategy para modificar dinámicamente el formato del mensaje del bot. - Cada estrategia formatea el mensaje base. - Compatible con cambios en tiempo real.	Crea una clase ChatBotResponder que use estrategias (FormalStyle, CasualStyle, TechnicalStyle). Todas las estrategias deben definir el método formatResponse(message). Permitir el cambio dinámico de tono. Define las clases en un solo archivo y expórtalas. Usa ES modules.
3	Un sitio de e-learning ajusta la presentación del contenido según el tipo de usuario: visual, auditivo o lector.	- Usar Strategy para presentar el contenido según el estilo de aprendizaje. - Cada estrategia determina la vista del módulo (video, audio, texto). - Debe ser intercambiable dinámicamente.	Crea una clase LearningModule que use estrategias (VisualStrategy, AuditoryStrategy, ReadingStrategy). Todas las estrategias deben definir el método renderContent(data). Permite cambiar la estrategia. Define las clases en un solo archivo y expórtalas. Usa ES modules.

4	Un sistema de notificaciones aplica diferentes estilos visuales según el tipo: éxito, error, advertencia.	- Aplicar Strategy para encapsular el comportamiento visual de la notificación. - Cada estrategia controla íconos, colores y mensajes. - Fácil de extender con nuevos tipos.	Crea una clase Notification que use estrategias (SuccessStrategy, ErrorStrategy, WarningStrategy). Todas las estrategias deben definir el método renderNotification(data). Permite aplicar el estilo en tiempo real. Define las clases en un solo archivo y expórtalas. Usa ES modules.
5	Un formulario inteligente cambia el orden de los campos según el contexto: registro, suscripción o feedback.	- Usar Strategy para definir dinámicamente el orden y visibilidad de campos. - Cada contexto tiene su propia lógica de presentación. - Evitar lógica condicional en el componente principal.	Crea una clase SmartForm que utilice estrategias (RegisterFormStrategy, SubscribeFormStrategy, FeedbackFormStrategy). Todas las estrategias deben definir el método getFieldsLayout(). Cambia de estrategia según el contexto. Define las clases en un solo archivo y expórtalas. Usa ES modules.
6	Un gestor de tareas maneja tareas pendientes, en progreso y completadas.	- Aplicar el patrón Strategy para definir el comportamiento dinámico de cada estado. - Cada estado debe tener su propia estrategia de renderizado (color, íconos, botones). - Debe poder cambiar la estrategia en tiempo de ejecución.	Implementa una clase TaskCard que acepte una estrategia (PendingStrategy, InProgressStrategy, CompletedStrategy) para determinar cómo se renderiza. Todas las estrategias deben definir el método render(data). Crea las estrategias y vincúlalas dinámicamente. Define las clases en un solo archivo y expórtalas. Usa ES modules.