1. Patrón MVC (Model-View-Controller)

Problema recurrente:

En aplicaciones con múltiples pantallas y funcionalidades, es común que la lógica de negocio, la interfaz y la gestión de datos se mezclen, dificultando la comprensión y mantenimiento del código.

Solución:

El patrón MVC divide la aplicación en tres componentes:

Modelo: maneja los datos y la lógica (usuarios, productos, recetas, bitácoras).

Vista: representa la interfaz de usuario (pantallas del sistema).

Controlador: recibe las acciones del usuario y las comunica al modelo o la vista.

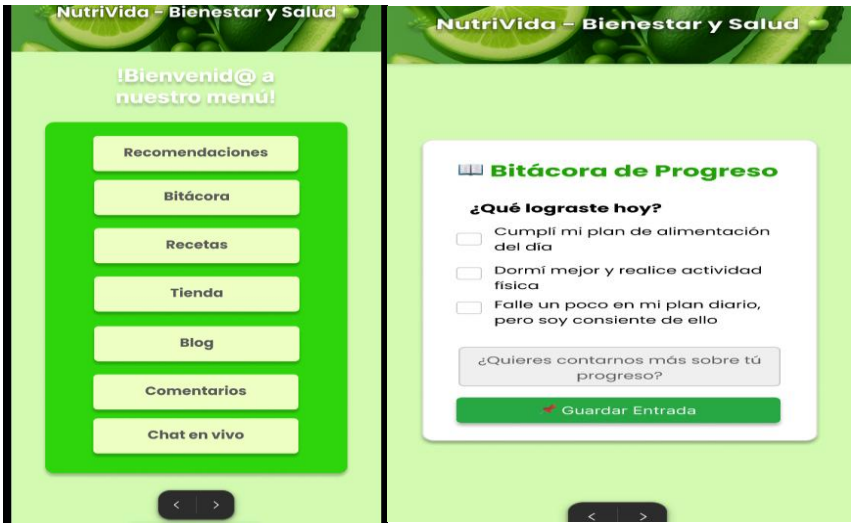
Aplicación en Nutrivida:

Cada pantalla corresponde a una vista (por ejemplo, LoginView, BitacoraView, TiendaView), cada una gestionada por un controlador que maneja las peticiones.

El modelo centraliza las clases lógicas del sistema (Usuario, Producto, Receta, Pedido, Dieta).

Beneficio:

Estructura modular, fácil mantenimiento y posibilidad de desarrollar las vistas y la lógica por separado.



2. Patrón Observer

Problema recurrente:

El sistema debe mantener actualizadas varias pantallas o componentes cuando ocurre un cambio en los datos, sin necesidad de actualizar manualmente cada vista.

Solución:

El patrón Observer permite que varios objetos (observadores) estén “suscritos” a un objeto principal (sujeto). Cuando el sujeto cambia, todos los observadores se actualizan automáticamente.

Aplicación en Nutrivida:

En la bitácora, cuando el usuario registra su avance diario, la pantalla de estadísticas se actualiza automáticamente.

En el chat en vivo, los mensajes nuevos aparecen en tiempo real para todos los usuarios conectados.

En la tienda, el usuario recibe una notificación visual cuando el pedido cambia de estado (por ejemplo, “en camino” o “entregado”).

Beneficio:

Sincronización automática de la información, mejora de la interacción en tiempo real y reducción de dependencias entre componentes.

El sistema necesita cambiar la forma en que hace ciertos cálculos o recomendaciones según el objetivo del usuario (por ejemplo, bajar de peso, mantener o subir masa).

Hacerlo todo en un solo bloque de código sería complicado y difícil de modificar.

Solución:

El patrón Strategy permite crear diferentes maneras de hacer una misma tarea y elegir cuál usar según la necesidad.

Así, el programa puede cambiar el tipo de cálculo sin tener que modificar el resto del sistema.

3.Patron Strategy.

Aplicación en Nutrivida:

En la parte de Recomendaciones, hay varias estrategias:

Una para perder peso, otra para mantenerse, y otra para ganar masa.

El sistema escoge la que corresponda según lo que el usuario haya indicado en su perfil.

Beneficio:

Hace el código más fácil de modificar y permite agregar nuevas formas de recomendación sin cambiar todo el programa.

Prototipo: https://www.figma.com/proto/Lxcb5szjjS5SWlG3wog6mg/Nutrivida-Final?node-id=1-1223&t=W0rNQYRgbaMivaQL-1&starting-point-node-id=1%3A6399