1. R. Sí, el proyecto Healthy+ implementa una arquitectura modular organizada en capas, donde la estructura del proyecto separa la configuración de la base de datos en la carpeta config, los modelos de datos como Medicamento, EventoToma y Sintoma en la carpeta models, los controladores que manejan la lógica de negocio en controllers, los servicios especializados como el Administrador de Almacenamiento y los alertadores en servicios, y al final las vistas en views por lo que esta separación permite que si se necesita cambiar la base de datos, únicamente se modificarían los archivos Gestor Almacenamiento.js y Database.js sin afectar a los controladores ni a las vistas.

2. R. En el proyecto se observa alta cohesión ya que cada componente tiene una responsabilidad única y bien definida, por ejemplo el ControladorMedicamentos se encarga exclusivamente de la gestión de medicamentos, el Controlador Alertas maneja únicamente las alertas y notificaciones, el GestorAlmacenamiento se dedica solo a la persistencia de datos, el Notificador Sonoro genera exclusivamente avisos sonoros, mientras que el Notificador Visual se encarga únicamente de las notificaciones visuales en la pantalla.

Por otro lado en cuanto al bajo acoplamiento, el proyecto implementa el patrón Observer mediante las clases Sujeto y Observador, aquí la clase Sujeto define un contrato abstracto con métodos para suscribir, desuscribir y notificar observadores, también la clase Observador define una interfaz con el método actualizar que debe ser implementado por las clases concretas, también, el RelojSistema, que actúa como sujeto, no conoce a los observadores como NotificadorSonoro o NotificadorVisual, sino que trabaja únicamente con la abstracción Observador, esto permite agregar nuevos tipos de notificadores, como por ejemplo uno de mensajes de texto o correo electrónico, sin necesidad de modificar el código del reloj.

3. R. El proyecto aplica esto correctamente, pues en el caso del Medicamento, toda la revisión de los datos se hace dentro de su función de validar, por otro lado, los modelos tienen reglas de conversión usando toJSON y fromJSON, donde estos métodos funcionan como traductores, pues se encargan de transformar la información de la aplicación a un formato listo para guardarse, y también hacen el proceso inverso, de esta manera podemos guardar y recuperar datos sin necesidad de revelar cómo está organizada la base de datos por dentro, además, la clase ConexionBD implementa el patrón Singleton para asegurar una sola instancia de conexión a la base de datos, encapsulando todo el proceso de inicialización y configuración de IndexedDB.

4. R. El proyecto cumple con este principio mediante una separación en tres capas, donde la capa de interfaz de usuario está representada por las clases VistaListaMedicamentos, VistaSintomas y VistaHistorial, las cuales se encargan solo del renderizado de elementos en el DOM y la captura de eventos del usuario, sin contener lógica de negocio.

La capa de lógica de negocio está implementada en los controladores como Controlador Medicamentos, Controlador Alertas, Controlador Formulario, Controlador Historial y Controlador Sintomas, donde estos componentes contienen las reglas de negocio, las validaciones y la coordinación entre las otras capas.

La capa de datos está compuesta por el GestorAlmacenamiento y Database, los cuales gestionan todo el acceso a IndexedDB de forma independiente, por otro lado el archivo app.js conecta las vistas con los controladores mediante callbacks, pero sin mezclar las responsabilidades de cada capa.

5. R. El proyecto aplica patrones de diseño como el patrón Singleton que se implementa en la clase ConexionBD para garantizar una única conexión a la base de datos durante toda la ejecución de la aplicación, evitando conflictos y consumo innecesario de recursos, también el patrón Observer está implementado a través de las clases Sujeto, Observador y Reloj Sistema, donde este se extiende de Sujeto y monitorea continuamente si hay medicamentos pendientes de tomar y cuando detecta una alerta, notifica a todos los observadores suscritos, que en este caso son Notificador Visual y Notificador Sonoro, donde ambos se extienden de La clase Observador y aplican el método actualizar de manera particular, y como se nota este patrón facilita la ampliación del sistema ya que para agregar un nuevo tipo de notificación solo se necesita crear una nueva clase que extienda Observador y suscribirla al Reloj Sistema.

6. R. Entre los aspectos positivos incluyen que los modelos como Medicamento y Eventos Toma tienen métodos de validación que pueden probarse de forma unitaria sin dependencias externas, se puede crear una instancia de Medicamento con datos erróneos y verificar que el método validar retorne los errores esperados, todo esto sin necesidad de base de datos ni interfaz gráfica.

Sin embargo el Gestor Almacenamiento depende directamente de IndexedDB sin una capa de abstracción que permita simular la base de datos.

7.R. Nuestro proyecto tiene partes de seguridad, como la validación de entradas, y también tanto la clase Medicamento como la clase Evento Toma tienen funciones de comprobación que aseguran que los campos esenciales están presentes, los formatos son adecuados y los valores se encuentran dentro de límites permitidos, por ejemplo, se valida que el nombre del medicamento no esté vacío, que la dosis sea mayor a cero, que la frecuencia sea válida, que esté dentro del rango, y que la hora tenga el formato correcto HH:MM.

Con respecto a la auditoría, el modelo EventoToma tiene información completa sobre cada acción relacionada con la toma de medicamentos, teniendo también el id del medicamento, el tipo de evento que puede ser como omitido o pospuesto, la fecha y hora programada para su toma, la fecha y hora real de la acción, la dosis administrada y el motivo en caso de omisión o postergación.

Característica	Subcaracterística	Pregunta de verificación	Evidencia sugerida	Estado (ref.)	Notas / acciones
Adecuación funcional	Completitud funcional	¿Están implementados todos los casos de uso acordados?	Pruebas, métricas, revisión, artefactos.	Cumple	RF01 (CRUD medicamentos), RF02 (Alertas/Recordatorios), RF06 (Historial), RF07 (Síntomas) implementados completamente.
Adecuación funcional	Corrección funcional	¿Los resultados son correctos para entradas válidas y límites?	Pruebas, métricas, revisión, artefactos.	Parcial	Validación en Medicamento.js:30-54 cubre campos obligatorios, pero falta validación de límites (ej. dosis máxima 2000mg hardcodeada, duración 1-365 en HTML pero sin validación JS).
Adecuación funcional	Pertinencia funcional	¿Las funciones ayudan a lograr objetivos del usuario sin pasos extra?	Pruebas, métricas, revisión, artefactos.	Cumple	Flujo directo: crear/editar/eliminar medicamentos, registrar tomas, ver historial y síntomas. UI intuitiva con FAB y navegación clara.

Eficiencia de desempeño	Comportamiento temporal	¿Responde dentro del tiempo objetivo bajo carga esperada?	Pruebas, métricas, revisión, artefactos.	Parcial	IndexedDB es asíncrono y eficiente para datos locales. Sin métricas de rendimiento definidas; el RelojSistema verifica cada 60 segundos (configurable en RelojSistema.js:14).
Eficiencia de desempeño	Utilización de recursos	¿Uso de CPU/RAM/BD es razonable y monitoreable?	Pruebas, métricas, revisión, artefactos.	Parcial	Uso de IndexedDB local es eficiente. Sin métricas de monitoreo. El patrón Singleton en Database.js:5 previene conexiones duplicadas.
Eficiencia de desempeño	Capacidad	¿Soporta el volumen esperado (N registros, concurrencia)?	Pruebas, métricas, revisión, artefactos.	Parcial	Sin límites definidos ni paginación. Las vistas limitan resultados (ej. 10 eventos en detalle, 5 próximas tomas). Falta prueba con volumen alto de datos.
Compatibilidad	Coexistencia	¿Convive con otros sistemas/servicios sin interferir?	Pruebas, métricas, revisión, artefactos.	N/A	Aplicación web standalone sin integraciones externas. Usa CDN para jsPDF y xlsx.

Compatibilidad	Interoperabilidad	¿Integra/expone APIs con contratos claros (OpenAPI)?	Pruebas, métricas, revisión, artefactos.	No cumple	No hay API REST; es una aplicación frontend pura con IndexedDB local. Sin especificación de intercambio de datos.
Usabilidad	Reconocibilidad de adecuación	¿El usuario entiende si el sistema le sirve?	Pruebas, métricas, revisión, artefactos.	Cumple	UI clara con etiquetas descriptivas: "Mis Medicinas", "Nueva Medicina", iconos intuitivos (pastilla, jarabe, inyección).
Usabilidad	Aprendibilidad	¿Un usuario nuevo aprende a usarlo rápidamente?	Pruebas, métricas, revisión, artefactos.	Cumple	Flujo estándar, formularios guiados con placeholders ("Ej. Aspirina"), selectores visuales para presentación y frecuencia.
Usabilidad	Operabilidad	¿Se puede operar con facilidad (pocos clics, navegación clara)?	Pruebas, métricas, revisión, artefactos.	Cumple	FAB para agregar, navegación con botones "← Atrás", acciones directas desde tarjetas (editar/eliminar).
Usabilidad	Protección contra errores de usuario	¿Previene errores (validación, confirmaciones)?	Pruebas, métricas, revisión, artefactos.	Parcial	Confirmación al eliminar en VistaListaMedicamentos.js:138. Botón guardar deshabilitado sin

					nombre. Falta confirmación al omitir dosis con modal implementado.
Usabilidad	Estética de interfaz	¿Diseño consistente y legible?	Pruebas, métricas, revisión, artefactos.	Cumple	Variables CSS en styles.css:4-25 con paleta Allports. Tipografía del sistema. Componentes consistentes (cards, botones, iconos).
Usabilidad	Accesibilidad	¿Soporta accesibilidad mínima (labels, contraste, teclado)?	Pruebas, métricas, revisión, artefactos.	Parcial	Algunos aria-label presentes, iconos con title. Falta navegación completa por teclado, etiquetas for en algunos inputs, y verificación WCAG.
Fiabilidad	Madurez	¿Falla poco en condiciones normales?	Pruebas, métricas, revisión, artefactos.	Parcial	Manejo de errores con try/catch en controladores. Logs de consola extensivos. Sin pruebas automatizadas para verificar estabilidad.
Fiabilidad	Disponibilidad	¿Está disponible según objetivo?	Pruebas, métricas, revisión, artefactos.	N/A	Aplicación local en navegador, sin servidor. Disponibilidad depende del

					dispositivo del usuario.
Fiabilidad	Tolerancia a fallos	¿Maneja fallos (BD caída, timeouts) sin colapsar?	Pruebas, métricas, revisión, artefactos.	Parcial	Manejo de errores en Database.js:51-55 y controladores. Muestra alertas al usuario. Falta manejo de IndexedDB no disponible.
Fiabilidad	Recuperabilidad	¿Se recupera (backup/restore) ante fallos?	Pruebas, métricas, revisión, artefactos.	No cumple	Exportación a PDF/Excel existe para historial, pero no hay funcionalidad de backup/restore de datos completa.
Seguridad	Confidencialidad	¿Controla acceso a datos (authz/authn)?	Pruebas, métricas, revisión, artefactos.	No cumple	Sin autenticación ni autorización. Datos almacenados localmente sin cifrado. Adecuado para uso personal local.
Seguridad	Integridad	¿Protege contra modificación no autorizada y valida entradas?	Pruebas, métricas, revisión, artefactos.	Parcial	Validación de datos en modelos. escaparHTML() en VistaListaMedicamentos.js:184 previene XSS. Datos locales sin protección.
Seguridad	No repudio	¿Registra acciones para	Pruebas, métricas,	Parcial	Registro de eventos de toma en EventoToma.js

		evidenciar autoría?	revisión, artefactos.		con timestamps. Sin identificación de usuario.
Seguridad	Responsabilidad	¿Se puede rastrear quién hizo qué?	Pruebas, métricas, revisión, artefactos.	No cumple	Aplicación monousuario sin identificación. Historial de acciones limitado a eventos de toma.
Seguridad	Autenticidad	¿Identidades verificadas de forma robusta?	Pruebas, métricas, revisión, artefactos.	No cumple	Sin sistema de autenticación. Diseñado para uso personal local.
Mantenibilidad	Modularidad	¿Módulos separados (capas, paquetes) y dependencias controladas?	Pruebas, métricas, revisión, artefactos.	Cumple	Arquitectura MVC clara: models/, views/, controllers/, services/. Imports ES6. Patrones Singleton y Observer implementados.
Mantenibilidad	Reusabilidad	¿Componentes reutilizables (validadores, DTOs, utilidades)?	Pruebas, métricas, revisión, artefactos.	Cumple	Modelos reutilizables (Medicamento, EventoToma, Sintoma). GestorAlmacenamiento centralizado. Patrón Observer reusable (Sujeto/Observador).
Mantenibilidad	Analizabilidad	¿Es fácil diagnosticar defectos	Pruebas, métricas,	Cumple	Logs extensivos con emojis y etiquetas

		(logs, trazas, claridad)?	revisión, artefactos.		(RelojSistema.js:49-73). Código bien documentado con JSDoc.
Mantenibilidad	Modificabilidad	¿Cambios locales con bajo riesgo?	Pruebas, métricas, revisión, artefactos.	Cumple	Separación de capas permite modificar vistas sin afectar lógica. Variables CSS centralizadas. Bajo acoplamiento entre módulos.
Mantenibilidad	Capacidad de prueba	¿Hay pruebas unitarias/integración y CI?	Pruebas, métricas, revisión, artefactos.	No cumple	Sin pruebas automatizadas ni CI/CD configurado. Código estructurado facilita agregar tests.
Portabilidad	Adaptabilidad	¿Puede ejecutarse en distintos entornos sin cambios mayores?	Pruebas, métricas, revisión, artefactos.	Cumple	Aplicación web estática. Funciona en cualquier navegador moderno con IndexedDB. Responsive con viewport meta.
Portabilidad	Instalabilidad	¿Instalación reproducible (scripts, Docker, README)?	Pruebas, métricas, revisión, artefactos.	No cumple	README.txt está vacío. Sin instrucciones de instalación, package.json, ni contenedor Docker.

Portabilidad	Reemplazabilidad	¿Puede reemplazar un componente por otro equivalente?	Pruebas, métricas, revisión, artefactos.	Parcial	GestorAlmacenamiento podría abstraerse para usar otra BD. Patrón Observer permite agregar/quitar notificadores. Falta interface para almacenamiento.