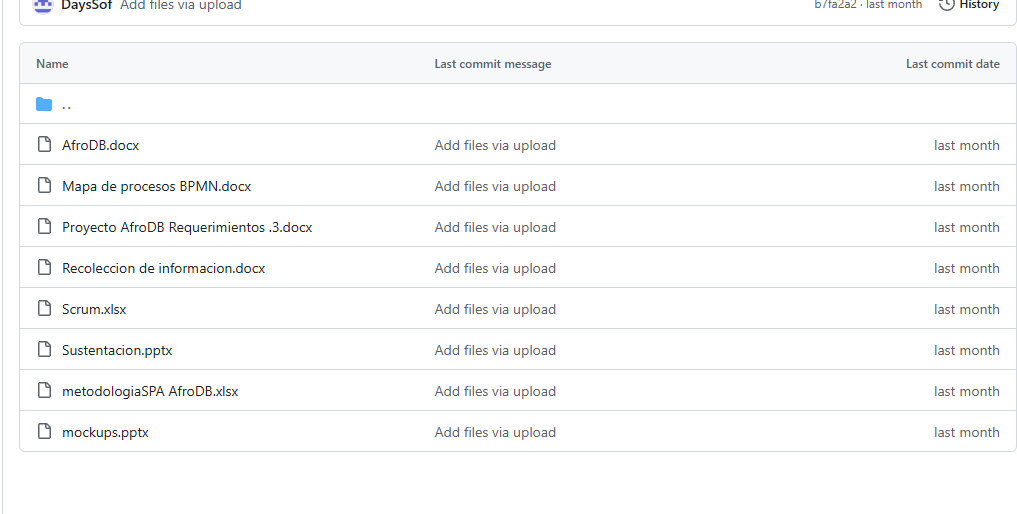
# Checklist AfroDB

|  |  |  |  |
| --- | --- | --- | --- |
| Trimestre | Lista de chequeo | Cumple | No cumple |
| I | Nombre proyecto - Objetivo general - Objetivos especí­ficos - Planteamiento del problema y pregunta problema - Alcance del proyecto - Justificación | X |  |
| I | En el proyecto se evidencian la elaboración del mapa de procesos que implica el sistema de información (BPMN del proceso de negocio, no del aplicativo a desarrollar) | X |  |
| I | En el proyecto se evidencian técnicas de recolección de información y análisis de resultados estadística descriptiva (Si aplica, dependiendo de la técnica a usar). | X |  |
| I | En el proyecto se evidencian los requerimientos funcionales y no funcionales usando el estándar IEEE Std 830-1998, IEEE 1233-1998, IEEE 29148:2018 o historias de usuario (Scrum) | X |  |
| I | En el proyecto se evidencia validación de requerimientos (Realización de un prototipo usando mockups o wireframes) | X |  |
| I | En el proyecto se evidencia el uso de sistemas de control de versiones (Opcional) | X |  |
| II | En el proyecto se evidencian formatos de fichas técnicas, estimación de costos de software-hardware, análisis comparativo de proveedores considerando costos, cantidad y especificaciones técnicas de acuerdo con las características de la solución de software y las necesidades de la empresa. | X |  |
| II | En el proyecto se evidencia el diagrama de casos de uso y documentación de casos de uso (formato de casos de uso extendido) de acuerdo con el refinamiento de requisitos. |  | X |
| II | En el proyecto se evidencia el modelo entidad relación notación crows foot |  | X |
| II | En el proyecto se evidencia la normalización del modelo relaciónal, identificando las 3FN |  | X |
| II | En el proyecto se evidencia el diccionario de datos |  | X |
| II | Se evidencia el diagrama de clases del proyecto donde se ven las clases y relaciones usando el estándar UML 2.4.1 o superior |  | X |
| II | En el proyecto se evidencia la realización del diagrama de despliegue usando el estándar UML 2.4.1 o superior |  | X |
| II | En el proyecto se evidencia el uso de sistemas de control de versiones (Opcional) | X |  |
| III | En el proyecto evidencia la construcción de la base de datos usando sentencias DDL (SQL) o Schema Validation (MongoDB) |  | X |
| III | En el proyecto se evidencia el uso de la base de datos a través de sentencias DML (SQL), revisando existencia de datos de prueba insertados, Joins, consultas y subconsultas o MongoDB CRUD Operations y aggregations. |  | X |
| III | Dentro del esquema de seguridad de la Base de Datos se observa que el password o contraseñas contemplan encriptación de datos. Igualmente se evidencia la práctica de seguridad a los datos. |  | X |
| III | Se evidencia en el proyecto la realización del Front-End funcional (codificación), utilizando frameworks como Bootstrap, Materialize, Angular, Vue.js o React, de acuerdo con el diseño establecido en el prototipo y la base de datos. En caso de no contar con un backend desarrollado, este debe ser simulado mediante herramientas como JSON Server u otras similares. Asimismo, se debe implementar una simulación de autenticación utilizando JWT, almacenado en localStorage o sessionStorage. |  | X |
| III | En el proyecto se evidencia el uso de sistemas de control de versiones (Opcional) | X |  |
| III | Elaborar el prototipo navegable del software (HTML - CSS) |  | X |
| IV | En el proyecto se evidencia la implementación de la API REST agregando seguridad. |  | X |
| IV | En el proyecto se evidencia que el front end web consuma la API REST (Angular, Vue.js, react u otros), con un avance del 80% de la codificación |  | X |
| IV | En el proyecto se evidencia el uso de sistemas de control de versiones. | X |  |
| V | En el proyecto se evidencia el desarrollo de la codificación al 100% |  | X |
| V | En el proyecto se evidencia el consumo de la API Rest con aplicaciones móviles. |  | X |
| V | En el proyecto se evidencia la implementación de la API REST documentada (swagger u otra herramienta). |  | X |
| V | En el proyecto móvil se evidencia la aplicación de una metodología ágil. (Historias de Usuario, roles, sprints, Backlog, etc) |  | X |
| VI | En el proyecto se evidencia que se haya usado alguna técnica de pruebas de software. |  | X |
| VI | En el proyecto se evidencia un plan de instalación, plan de respaldo, plan de migración y capacitación. |  | X |
| VI | En el proyecto se evidencia algún modelo de calidad. |  | X |
| VI | En el proyecto se evidencian manuales de instalación, técnico y de usuario. |  | X |
| VI | En el proyecto se evidencia el despliegue del aplicativo según el diseño de la arquitectura en UML (Diagrama de despliegue o Diagrama de arquitectura de nube). |  | X |
| VI | En el proyecto se evidencia el uso de sistemas de control de versiones. | X |  |

|  |  |
| --- | --- |
| OBSERVACIONES | |
| Trimestre I | Se reconstruyeron las técnicas de recolección de datos y se aplicaron en entornos empresariales reales, lo que ayudó a tener una retroalimentación e información valiosa. |
| Trimestre I | Tras rehacer la recolección de datos, la información obtenida se volvió concreta y será sustentada en bases reales. |
| Control de versiones | Se evidencia que desde el primer trimestre se maneja el control de versiones con GitHub |

[](https://github.com/DaysSof/Sena-evidencias.git)

# MÉTRICA DE CALIDAD DEL SOFTWARE

[](../../../AfroDB/Matriz%20de%20métricas.xlsx)

# Explicación de cada criterio con su relación (X)

1. **Facilidad de auditoría**
   1. **Corrección, Fiabilidad, Mantenibilidad, Capacidad de prueba, Interoperabilidad**  
      Se marca porque el sistema debe permitir rastrear transacciones, identificar errores en pedidos, y facilitar auditorías contables. Además, auditar ayuda a mantener la fiabilidad, permite realizar pruebas de control, y requiere interoperar con sistemas externos (ej. facturación electrónica).
2. **Exactitud**
   1. **Corrección, Fiabilidad**  
      El software debe registrar los valores de inventario, precios y facturación sin errores, garantizando tanto la corrección de los datos como la fiabilidad en los cálculos.
3. **Estandarización de comunicaciones**
   1. **Reusabilidad, Interoperabilidad, Usabilidad**  
      Si el sistema usa estándares de comunicación (ej. APIs REST, mensajes JSON), puede reutilizar módulos, integrarse con otros sistemas (bancos, pasarelas de pago) y ofrecer interfaces más fáciles de usar.
4. **Complejidad**
   1. **Mantenibilidad, Flexibilidad, Capacidad de prueba**  
      A menor complejidad, más fácil mantener y adaptar el software. También se simplifican las pruebas, ya que sistemas menos complejos generan menos escenarios posibles de fallo.
5. **Concisión**
   1. **Mantenibilidad, Reusabilidad**  
      Código conciso y claro es más fácil de mantener y se puede reutilizar en diferentes módulos o proyectos sin necesidad de grandes cambios.
6. **Consistencia**
   1. **Corrección, Integridad, Usabilidad**  
      Una interfaz consistente y reglas homogéneas (ej. precios con el mismo formato en todas partes) aseguran la corrección, evitan errores de integridad y mejoran la experiencia del usuario.
7. **Estandarización de datos**
   1. **Corrección, Integridad, Mantenibilidad**  
      Manejar datos estandarizados (ej. formatos de fecha, divisas, identificación de clientes) asegura la corrección de la información, mantiene la integridad de la base de datos y facilita futuras modificaciones.
8. **Tolerancia al error**
   1. **Fiabilidad, Capacidad de prueba**  
      El sistema debe recuperarse de errores (ej. caída de conexión al pagar o al agendar) sin perder información, lo cual incrementa la fiabilidad y debe probarse sistemáticamente.
9. **Eficiencia de ejecución**
   1. **Eficiencia**  
      Se centra directamente en el uso óptimo de recursos (tiempo de respuesta, consumo de CPU/memoria).
10. **Capacidad de expansión**
    1. **Mantenibilidad, Flexibilidad, Reusabilidad**  
       Permite agregar nuevas funciones (ej. nuevos métodos de pago, nuevos estilistas, productos, etc.) sin reconstruir el sistema.
11. **Generalidad**
    1. **Flexibilidad**  
       Un software que cubre casos generales (ej. distintos tipos de clientes) se adapta mejor a diferentes contextos.
12. **Independencia del hardware**
    1. **Portabilidad**  
       El sistema debe funcionar en distintos dispositivos o servidores sin cambios significativos.
13. **Instrumentación**
    1. **Mantenibilidad, Capacidad de prueba**  
       Registrar logs y métricas facilita localizar fallos y comprobar el funcionamiento en pruebas.
14. **Modularidad**
    1. **Mantenibilidad, Flexibilidad, Reusabilidad**  
       Los módulos separados (ej. carrito, pagos, catálogo) permiten modificaciones independientes, reutilización de componentes y mayor facilidad de mantenimiento.
15. **Operatividad**
    1. **Usabilidad**  
       Relacionada con que los usuarios (clientes y administradores) puedan usar el software sin complicaciones.
16. **Auto documentación**
    1. **Mantenibilidad, Capacidad de prueba**  
       Código y sistemas bien documentados son más fáciles de mantener y facilitan pruebas sobre funcionalidades específicas.
17. **Simplicidad**
    1. **Mantenibilidad, Reusabilidad**  
       Un diseño simple reduce errores, facilita mantenerlo y permite reutilizar componentes en otros sistemas.
18. **Independencia del sistema**
    1. **Portabilidad**  
       El software debe funcionar en diferentes sistemas operativos o entornos de ejecución sin problemas.
19. **Trazabilidad**
    1. **Corrección, Mantenibilidad, Capacidad de prueba**  
       Permite rastrear cambios en pedidos y requisitos desde el cliente hasta el código y pruebas, asegurando la corrección, mejor mantenimiento y validación de calidad.
20. **Facilidad de formación**
    1. **Usabilidad**  
       Si el sistema es fácil de aprender, se mejora la experiencia del usuario y se reducen costos de capacitación.