UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Departamento de Ciencias de la Computación

Análisis y Diseño de Software

Checklist: Atributos de calidad para el diseño ISO/IEC 25510

Integrantes:

Gabriel López

Marcelo Pareja

Kevin Asmal

Diego Delgado

NRC: 27835

Ecuador 2026-01-06

Principio	*Pregunta guía*	*Cumple*	*Justificación*
Modularidad	¿Puedo entender/cambiar una parte sin romper las demás?	☑	Se tienen diferentes carpetas que separan las capas de trabajos como la capa servicios, que se encarga totalmente en la manipulación en la base de datos, la capa controlador que realiza las reglas de negocio y la capa de datos con la carpeta de modelos que tiene los objetos que se crean y manipulan en el proyecto.
Alta cohesión y bajo acoplamiento	¿Cada módulo hace una cosa clara? ¿Depende lo mínimo de otros?	☑	Cada entidad tiene su propio servicio y controlador, que maneja sus reglas de negocio y operaciones respectivamente. La arquitectura de la aplicación, permite dependencias unidireccionales, donde un módulo depende como máximo de uno o 2.
Abstracción y encapsulación	¿Oculto detalles internos y expongo solo contratos?	☑	Cada módulo exporta al otro únicamente funciones, sin comunicar información adicional. El acceso a los modelos y la base de datos está protegido ya que se da mediante servicios y nunca de manera directa.
Separación de responsabilidades (SoC)	¿UI, lógica de negocio y persistencia están separadas?	☑	Existe separación por capas de servicio, controladores, datos y rutas que cada capa maneja solo exportando funcionalidades útiles.
Reutilización y patrones	¿Reutilizar componentes y patrones conocidos cuando conviene?	☐	A nivel de lógica de negocios (backend), no se han implementado patrones de diseño o componentes que se puedan reutilizar. Las interfaces visuales (frontend) están construidas mediante la integración de componentes gráficos reutilizables.
Diseño para pruebas	¿Puedo probar funciones y componentes de forma aislada (unit tests)?	☑	Cada módulo al estar separado en las diferentes capas, permitirá probar cada sección de código en pruebas unitarias

			raza_rutas.js controlador > rutas > raza_rutas.js > ... 1 const controlador = require('../controller/raza_controller'); 2 const express = require('express'); 3 const router = express.Router(); 4 const {verificar_token} = require('../middleware/auth'); 5 6 router.post('/new', verificar_token, controlador.crearRaza); 7 router.get('/', verificar_token, controlador.obtenerRazas); 8 router.get('/:id', verificar_token, controlador.obtenerRazaPorId); 9 10 module.exports = router; raza_servicio.js controlador > servicios > raza_servicio.js > ... 1 2 const Raza = require('../../modelos/modelo/raza'); 3 const { Op } = require('sequelize'); 4 const sequelize = require('../../modelos/base_de_datos/sequelize'); 5 function validar_nombre_raza(nombre) { ··· 16 } 17 18 async function generar_siguiente_id() { ··· 30 } 31 32 function validar_descripcion_raza(descripcion) { ··· 37 } 38 39 async function crear_raza(datos) { ··· 64 } 65 66 async function actualizar_raza(conejo_raza_id, datos_actualizacion) { ··· 104 } 105 106 async function obtener_razas() { ··· 110 } 111 112 async function obtener_raza_por_id(conejo_raza_id) { ··· 118 } 119 120 async function eliminar_raza(conejo_raza_id) { ··· 131 } 132
Seguridad por diseño	¿Válido entradas, gestionar permisos y registrar auditoría?	☑	Cumple parcialmente, se validan las entradas tanto en los servicios como en la creación del modelo, en el sistema no se gestionan permisos, pero si existe un acceso controlado por contraseña para el ingreso, para las auditorías existen logos de acceso

Conclusiones.

En conclusión, la arquitectura del sistema presenta un diseño bien estructurado basado en principios de modularidad, alta cohesión y bajo acoplamiento, lo que facilita la comprensión, el mantenimiento y la evolución del proyecto sin afectar otras partes del sistema. La separación clara de responsabilidades entre capas (servicios, controladores, datos y rutas) permite un adecuado manejo de la lógica de negocio, el acceso a datos y la validación de información, promoviendo además la abstracción y encapsulación al exponer únicamente funciones necesarias. Aunque no se aplican patrones de diseño reutilizables en el backend, la organización por módulos favorece la realización de pruebas unitarias de forma aislada. En cuanto a la seguridad, el sistema cumple parcialmente al validar entradas y controlar el acceso mediante autenticación, aunque aún puede fortalecerse con una gestión más robusta de permisos y auditorías.