UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

(Universidad del Perú, Decana de América)

Facultad de Ingeniería de Sistemas e Informática

"ENTREGABLE – MÓDULO DE ENCUESTAS"

PRÁCTICAS PRE PROFESIONALES

PROFESOR

William Martín, Enriquez Maguiña

PRESENTADO POR

Verde Huaynaccero, Erick Edison

Lima-Perú

2025

Informe del Módulo de Encuestas – Sistema de Recopilación de Feedback Institucional

Objetivo del Entregable

Implementar un sistema integral de encuestas institucionales que permita a directivos y docentes autorizados diseñar, publicar y analizar encuestas de feedback dirigidas a la comunidad educativa. El módulo debe habilitar creación de encuestas con cinco tipos de preguntas (texto corto, texto largo, opción única, opción múltiple y escala uno a cinco), garantizar segmentación automática por rol y vínculos institucionales, registrar respuestas inmutablemente con trazabilidad completa, y proporcionar análisis visual para toma de decisiones.

El propósito fue establecer un canal formal para medir satisfacción, identificar necesidades y evaluar servicios mediante encuestas segmentadas, anónimas o nominativas según configuración, con fechas de vencimiento automáticas y registro de participación.

El sistema incorpora controles de acceso por rol, validación de permisos para docentes, constructor dinámico de preguntas con validaciones por tipo, segmentación automática, registro inmutable con timestamp e IP, validación de completitud de obligatorias, y funcionalidades administrativas según estado.

La implementación respeta Ley 29733 (artículos 8 y 9), Ley 28044, y aplica buenas prácticas de ISO/IEC 27001 (controles A.9, A.12, A.14) e ISO/IEC 12207.

Metodología Aplicada

El desarrollo siguió cuatro etapas con validación continua:

1. *Análisis y refinación de requisitos funcionales:*

Traduje necesidades en seis historias de usuario con criterios verificables: HU-ENC-00 (panel con tres pestañas de estado y filtros), HU-ENC-01 (formulario dinámico con cinco tipos de preguntas), HU-ENC-02 (visualización de respuestas propias), HU-ENC-03 (constructor con wizard de tres pasos), HU-ENC-04 (gestión administrativa) y HU-ENC-05 (análisis de resultados).

2. *Diseño de arquitectura de datos y API:*

Diseñé modelo normalizado con cinco entidades relacionadas: Encuesta, PreguntaEncuesta, OpcionPregunta, RespuestaEncuesta y RespuestaPregunta. Especifiqué nueve endpoints REST organizados en cuatro secciones funcionales, documentando formatos de entrada con validaciones Zod, estructuras de respuesta, códigos de error y reglas de negocio. Diseñé validaciones específicas por tipo de pregunta.

3. Implementación modular backend con validaciones defensivas:*

Implementé cuatro capas independientes: ruteo con nueve endpoints y autenticación obligatoria, controladores con cinco funciones, middlewares con tres validadores especializados (permisos de creación, gestión y estructura de respuestas), y servicios con reglas de negocio y validaciones Zod.

4. *Desarrollo de suite de pruebas automatizadas:*

Desarrollé doce casos de prueba en cinco grupos funcionales validando: listado segmentado, contador de pendientes, creación con permisos, validación de acceso, formulario ordenado, envío con validación de completitud, prevención de duplicados y consulta de respuestas propias. Configuré limpieza automática de datos de prueba.

Herramientas Utilizadas

- *Node.js con Express:* Construcción de APIs REST con organización en cuatro capas y autenticación mediante tokens de sesión.
- *Prisma ORM:* Modelado de datos con cliente tipado, esquemas con enumeraciones estrictas, relaciones entre cinco entidades y migraciones controladas.
- *Zod:* Validación de esquemas con reglas verificables en tiempo de ejecución mediante tres esquemas principales: crearEncuestaSchema, responderEncuestaSchema y obtenerEncuestasSchema.
- *PostgreSQL (Neon):* Base de datos relacional con estructura normalizada, restricciones únicas para prevenir respuestas duplicadas.
- *Vitest con Supertest:* Testing automatizado de endpoints REST con aserciones sobre códigos HTTP, estructuras y validaciones. Suite con doce casos validando permisos, segmentación y flujos críticos.
- *GitHub:* Control de versiones con historial trazable de modificaciones por módulo.

Actividades Realizadas

- Refiné seis historias de usuario con criterios detallados: panel con tres estados y segmentación, formulario con validaciones específicas, visualización en solo lectura, constructor con wizard, gestión con operaciones condicionales y análisis con visualizaciones.
- Diseñé nueve endpoints REST: lista con paginación y filtros, contador de pendientes, búsqueda, polling, validación de acceso, formulario con estructura, envío con validación, consulta de respuestas propias y creación con estructura.
- Modelé cinco entidades en Prisma: Encuesta con configuración (título, descripción, fechas, flags booleanos, relación al autor), PreguntaEncuesta con tipo y orden, OpcionPregunta con texto y orden, RespuestaEncuesta con restricción única por encuesta-usuario, y RespuestaPregunta con campos específicos por tipo.
- Implementé cinco controladores gestionando: creación validando permisos, lista con segmentación automática, contador excluyendo respondidas, búsqueda en título y descripción, polling de actualizaciones, validación de acceso, formulario con preguntas ordenadas, envío validando completitud y formato, y consulta verificando respuesta previa.
- Codifiqué lógica de negocio con validaciones defensivas: verificación de permisos en PermisoDocente, segmentación automática consultando relaciones familiares o asignaciones docente, validación de estructura por tipo con Zod, creación transaccional, registro con validación de unicidad y completitud, y cálculo de estadísticas.
- Desarrollé tres middlewares especializados: puedeCrearEncuestas (verifica director o docente con permisos activos), puedeGestionarEncuesta (verifica director o autor), y validarRespuestaEncuesta (verifica encuesta activa, no respondida, obligatorias completas y formato por tipo).

- Construí suite de testing con doce casos en cinco grupos: autenticación por rol, filtrado por estado, búsqueda, contador, creación validando permisos, validación con segmentación, formulario completo, envío con IDs reales, prevención de duplicados con código 409, y consulta con error si no respondió.
- Documenté especificación de API con ejemplos cURL, formatos JSON, códigos de error estandarizados y reglas de negocio numeradas facilitando trazabilidad.
- Verifiqué manualmente flujos críticos: listado segmentado por rol (padre solo grados de hijos, docente institucionales más propias, director todas), contador excluyendo respondidas y vencidas, validación retornando motivo descriptivo, formulario con preguntas ordenadas y validaciones por tipo, envío validando completitud, prevención de duplicados mediante restricción única, consulta en solo lectura, y creación con validación de permisos y estructura.

Evidencias

- *Especificación técnica:* Documentación de seis historias con criterios verificables y 170 reglas numeradas; nueve endpoints con ejemplos, validaciones y códigos de error; modelo con cinco entidades y tres enumeraciones (EstadoEncuesta, TipoPregunta, PermisoTipo).
- *Implementación backend:* Archivo de rutas con nueve endpoints y autenticación; cinco controladores con estructura consistente; tres middlewares especializados; servicio con validaciones Zod para tres operaciones críticas.
- *Estructura de datos normalizada:* Encuesta con enumeración de estados (borrador, activa, cerrada, vencida), flags booleanos, fechas de trazabilidad, relación al autor y año académico; PreguntaEncuesta con cinco tipos, texto, flag obligatoria y orden; OpcionPregunta con texto y orden; RespuestaEncuesta con restricción única compuesta previene duplicados; RespuestaPregunta con campos específicos por tipo.
- *Suite de pruebas:* Doce casos en cinco grupos validando segmentación automática, autorización, búsqueda y filtros, contador, validación de acceso, formulario, envío con IDs reales, prevención de duplicados y consulta de respuestas propias. Configuración con setup creando usuarios únicos por timestamp y teardown limpiando en orden inverso.

- Verificación funcional:

Validé listado con segmentación automática: padre solo encuestas de grados de hijos (consulta cruzada con relaciones familiares), docente institucionales más propias (filtro por autor o público objetivo), director acceso total sin restricciones.

Verifiqué contador excluyendo respondidas mediante restricción única en RespuestaEncuesta por encuesta-usuario, filtrando encuestas no en lista de respondidas.

Confirmé validación de acceso retorna información descriptiva: success verdadero, tiene_acceso falso, motivo descriptivo, puede_responder falso y segmentacion_valida falso.

Validé formulario incluye preguntas ordenadas con identificador único, tipo, texto, flag obligatoria, orden, opciones ordenadas (si aplica), etiquetas (si aplica) y reglas de validación específicas.

Comprobé envío valida completitud: respuesta con pregunta obligatoria faltante retorna código 400 con mensaje descriptivo y lista de IDs faltantes; respuesta completa retorna 201 con confirmación, ID, timestamp, tiempo y estadísticas.

Verifiqué prevención de duplicados: responder nuevamente retorna código 409 con mensaje "Ya has respondido esta encuesta anteriormente"; registro tiene restricción única sobre encuesta_id-usuario_id.

Validé consulta de respuestas propias verifica existencia: encuesta no respondida retorna 400 con RESPONSE_NOT_FOUND; respondida retorna 200 con encuesta, metadatos (fecha, tiempo, completitud) y valores formateados por tipo.

Comprobé creación valida permisos: padre recibe 403 FORBIDDEN; docente sin permisos recibe 403 descriptivo; docente con permiso activo recibe 201 con encuesta e IDs generados; director recibe 201 sin validación adicional.

Dificultades Encontradas

- *Modelado de estructura flexible:* Diseñar modelo soportando cinco tipos de preguntas con requisitos variados implicó decidir entre JSON desnormalizado versus normalización. Opté por normalización con tres tablas (PreguntaEncuesta, OpcionPregunta, RespuestaPregunta) facilitando consultas y análisis, aceptando mayor complejidad en inserciones. Implementé validaciones específicas por tipo en middleware y servicio.
- *Validación de completitud:* Asegurar respuesta de preguntas obligatorias requirió validación en frontend (experiencia) y backend (seguridad). Implementé verificación mediante conjuntos de identificadores: obtener obligatorias, obtener respondidas, verificar diferencias. Retorno mensaje con lista exacta de IDs faltantes.
- *Prevención de duplicados concurrentes:* Garantizar usuario no responda dos veces, incluso con solicitudes simultáneas, requirió restricción única en base de datos sobre encuesta_id-usuario_id. Implementé manejo de error de violación retornando código 409 con mensaje descriptivo.
- *Cálculo de estadísticas sin destinatarios exactos:* Calcular porcentaje de participación requiere conocer destinatarios exactos según segmentación. Como segmentación no está completamente implementada, implementé cálculo temporal con valor fijo de 100 destinatarios documentando limitación para corrección futura.

- *Validación específica por tipo con mensajes descriptivos:* Implementar validaciones diferentes por tipo con mensajes contextualizados mencionando texto de pregunta específica requirió iterar sobre respuestas, buscar pregunta correspondiente, validar según tipo y construir mensaje con texto completo para claridad.

Lecciones Aprendidas

- Reforcé importancia de inmutabilidad en datos de auditoría. Respuestas deben ser inmutables para garantizar integridad de análisis y prevenir manipulación. Implementé mediante ausencia de endpoint de edición y mensaje claro: "No podrás modificar tus respuestas después de enviar", alineándose con principio de Integridad del artículo 9 de Ley 29733.
- Consolidé comprensión de validación defensiva en múltiples capas: frontend con mensajes en tiempo real, middleware verificando estructura y permisos, servicio con Zod verificando tipos y formatos, y base de datos con restricciones únicas. Esta defensa en profundidad según ISO/IEC 27001 A.14.2.5 enseña que cada capa reduce riesgos residuales.
- Aprendí valor de normalización versus desnormalización según patrones de acceso. Opté por normalización al identificar que consultas frecuentes requieren filtrado por tipo, ordenamiento y validación de existencia, facilitando validación de respuestas y análisis de resultados, aceptando mayor complejidad en creación.
- Desarrollé capacidad para diseñar validaciones reflejando restricciones institucionales reales. Restricción de permisos activos para docentes responde a modelo de delegación institucional, aplicando técnicamente el principio de Autorización Granular del control A.9.4.1 de ISO/IEC 27001.
- Comprendí importancia de trazabilidad completa. Registrar respuestas con timestamp, IP, tiempo de respuesta y estudiante relacionado (padres) habilita análisis de calidad: identificar respuestas apresuradas, detectar patrones por segmento y auditar participación, facilitando verificación de cumplimiento del principio de Legitimidad del artículo 5 de Ley 29733.
- Perfeccioné habilidad de comunicar restricciones mediante mensajes orientados a acción: en lugar de "Error de validación", mensajes específicos como "La respuesta a 'Comentarios adicionales' debe tener al menos diez caracteres" o "Ya has respondido esta encuesta", mejorando experiencia y reduciendo solicitudes de soporte.

Competencias Desarrolladas en el Módulo de Encuestas

Competencias Genéricas

CG1 – Valores, compromiso ético y social

Apliqué rigurosamente Proporcionalidad (Ley 29733 art. 8) diseñando recopilación de feedback solicitando únicamente información necesaria para su finalidad declarada sin datos personales

innecesarios. Implementé configuración de encuestas anónimas mediante flag es_anonima que omite usuario_id en respuestas almacenando solo agregados estadísticos, protegiendo privacidad en temas sensibles.

Implementé Seguridad (art. 9) mediante controles técnicos: inmutabilidad de respuestas enviadas (sin endpoint de edición), validación de completitud antes de aceptar envío, sanitización de texto para prevenir inyección, y restricción única previniendo duplicados concurrentes. Apliqué ISO/IEC 27001 A.9.4.3 restringiendo visualización de resultados completos: solo autor o director ven respuestas individualizadas, respondientes comunes ven agregados si mostrar_resultados está activo.

Comprendí que en encuestas a familias de menores, responsabilidad ética requiere transparencia en finalidad. Cada encuesta declara objetivo y uso previsto cumpliendo principio de Finalidad (art. 9). Implementé registro de timestamp e IP para auditoría sin identificación directa en encuestas anónimas, equilibrando trazabilidad con protección de privacidad.

CG3 – Capacidad de análisis y pensamiento crítico

Ejercité análisis sistemático descomponiendo problema en requisitos verificables y restricciones numeradas. Identifiqué casos problemáticos con soluciones preventivas: garantizar no responder dos veces (restricción única más validación en servicio), validar opción única corresponde a opción válida (búsqueda de ID en array de esa pregunta específica), determinar si encuesta está vencida (comparación automática de fecha_vencimiento con actual).

Apliqué pensamiento crítico evaluando estrategias de modelado: JSON desnormalizado (simple pero dificulta consultas), híbrida (balance intermedio) o normalización completa (complejo pero robusto). Seleccioné normalización al identificar que patrones de acceso requieren consultas por tipo, validación de existencia y análisis agregado, justificando complejidad con beneficios de integridad y extensibilidad.

Desarrollé capacidad para cuestionar supuestos validando con evidencia. Al diseñar restricción de no edición de preguntas publicadas, cuestioné si limita flexibilidad. Analicé impacto: permitir edición invalidaría respuestas anteriores o crearía inconsistencias. Concluí que restricción es necesaria pero implementé alternativa: duplicar encuesta copiando estructura como nuevo borrador.

CG4 – Habilidad para la comunicación oral y escrita

Elaboré documentación con cuatro niveles de audiencia: desarrolladores frontend (especificación de endpoints con estructura request/response, validaciones y códigos), desarrolladores backend (documentación de servicios con parámetros, valores de retorno y lógica), usuarios finales (mensajes claros sin terminología técnica) y auditores (trazabilidad de controles a normativas con citas de artículos).

Redacté reglas de negocio numeradas con formato consistente traduciendo requisitos en restricciones verificables. Ejemplos: "RN-ENC-43: Verificar que haya al menos una pregunta obligatoria" facilita verificación mediante conteo; "RN-ENC-52: Respuesta no puede ser modificada ni eliminada" justifica ausencia de endpoint PUT/DELETE y fundamenta mensaje sobre irreversibilidad.

Documenté ejemplos de uso mediante cURL con estructura completa, headers de autenticación, formato JSON con campos requeridos y descripción de respuesta esperada, reduciendo ambigüedades y acelerando integración frontend.

Aprendí que comunicación efectiva requiere consistencia en organización: numeración secuencial de reglas, estructura uniforme de respuestas API (success, data/error, mensaje), formato estandarizado de códigos de error facilitando referencia cruzada entre documentación, código y pruebas.

Competencias Específicas

CT1.1 – Aplicación de metodologías, estándares y métricas de calidad

Apliqué desarrollo incremental con trazabilidad bidireccional entre historias, endpoints, funciones y entidades, coherente con ISO/IEC 12207. Seguí ciclo Refinamiento→Diseño→Implementación con validaciones→Pruebas→Verificación para cada componente.

Cómo lo hice: Estructuré en fases con entregables verificables: refiné seis historias definiendo criterios numerados y 170 reglas traduciendo restricciones institucionales en validaciones técnicas; diseñé nueve endpoints especificando formatos con Zod, salida consistente y reglas aplicables; implementé cuatro capas independientes aplicando separación de responsabilidades facilitando testing unitario; desarrollé suite con doce casos validando flujos y condiciones de error.

Definí métricas objetivas validadas mediante suite: cobertura de permisos en 100% de endpoints de escritura, cobertura de validación en 100% de payload, prevención de duplicados mediante restricción única verificada con caso intentando responder dos veces, y consistencia de formato en todos los endpoints (estructura con success booleano, data o error).

Qué usé: Metodología ágil por historias con entregas semanales; patrón de cuatro capas; esquemas Zod con tipos estrictos; suite con Vitest y Supertest; versionamiento con commits atómicos; documentación progresiva.

Aprendizaje: Comprendí que aplicación de estándares no es cumplimiento formal sino práctica estructurada reduciendo defectos. Trazabilidad Historia→Endpoint→Controlador→Servicio→Entidad no es documentación redundante sino mecanismo de verificación asegurando implementación completa. Permitió detectar inconsistencias tempranamente: regla sin validación correspondiente señala brecha de implementación.

CT2.2 – Construcción y gestión de repositorios de datos

Diseñé estructura aplicando normalización selectiva equilibrando integridad con complejidad. Modelé cinco entidades relacionadas: Encuesta como contenedor con configuración, PreguntaEncuesta con tipo y orden, OpcionPregunta con orden independiente del ID, RespuestaEncuesta con restricción única previniendo duplicados, y RespuestaPregunta con campos específicos por tipo almacenando valores en formato adecuado.
Implementé validaciones de integridad referencial mediante relaciones en Prisma con eliminación en cascada: eliminar encuesta elimina automáticamente preguntas, opciones y respuestas asociadas manteniendo consistencia. Configuré restricción única compuesta en RespuestaEncuesta sobre encuesta_id-usuario_id garantizando unicidad mediante constraint de base de datos.
Cómo lo hice: Apliqué normalización hasta tercera forma evitando redundancia: autor como relación a Usuario no texto duplicado, tipo como enumeración no string libre, opciones en tabla separada no repetidas por respuesta. Implementé relaciones con integridad referencial y eliminación en cascada. Configuré índices en campos de filtrado frecuente: RespuestaEncuesta(encuesta_id, usuario_id) para "ya respondió", PreguntaEncuesta(encuesta_id, orden) para obtención ordenada.
Diseñé consultas relacionadas minimizando llamadas: uso de include para cargar encuesta con autor, preguntas con opciones ordenadas en una consulta; _count para calcular total sin cargar registros completos; findUnique con clave compuesta para verificar unicidad en operación atómica.
Qué usé: Prisma ORM con tipos estrictos; PostgreSQL con restricciones únicas compuestas, enumeraciones nativas y eliminación en cascada; migraciones controladas; transacciones para operaciones multi-paso (crear encuesta con preguntas y opciones, enviar respuesta); consultas con filtrado dinámico.
Aprendizaje: Confirmé que calidad de estructura determina complejidad de lógica de negocio y robustez. Modelar respuestas con restricción única previene duplicados confiablemente. Separar opciones facilita validación de existencia y análisis agregado. Usar enumeraciones garantiza valores válidos. Esta inversión en diseño reduce defectos, facilita evolución y mejora mantenibilidad a largo plazo.

CT2.3 – Fundamentos de gestión de calidad y seguridad

Implementé controles en cinco niveles aplicando defensa en profundidad: autenticación JWT obligatoria verificando identidad, autorización por endpoint restringiendo acceso según rol, validación de permisos específicos consultando PermisoDocente, validación de estructura con Zod verificando tipos, formatos y longitudes, y validación de integridad en base de datos mediante restricciones únicas y relaciones referenciales.
Apliqué Ley 29733 art. 9 mediante medidas técnicas protegiendo confidencialidad (segmentación asegura usuarios ven encuestas pertinentes), integridad (inmutabilidad previene manipulación, validaciones previenen datos inválidos) y disponibilidad (ausencia de limitadores en desarrollo, a

implementar antes de producción). Implementé ISO/IEC 27001 A.12.2.1 mediante sanitización de texto eliminando scripts o HTML malicioso.
Cómo lo hice: Configuré JWT obligatoria validando firma, vigencia y estructura; implementé tres middlewares especializados verificando permisos por contexto (puedeCrearEncuestas, puedeGestionarEncuesta, puedeVerResultados); codifiqué validaciones con Zod especificando reglas declarativas; implementé inmutabilidad mediante ausencia intencional de endpoint de edición con mensaje explícito; configuré restricción única intentando insertar y retornando error de conflicto si existe.
Implementé trazabilidad registrando timestamp de creación, timestamp de respuesta con IP y notificación al autor. Diseñé eliminación condicional: solo sin respuestas registradas (validación consultando count), eliminando en cascada entidades relacionadas dentro de transacción garantizando atomicidad.
Qué usé: Middleware JWT rechazando tokens inválidos con código 401; middlewares consultando tablas de permisos y verificando autoría; esquemas Zod con coerción y mensajes personalizados; transacciones para inserción atómica; restricciones únicas previniendo duplicados a nivel de motor; relaciones con eliminación en cascada.
Aprendizaje: Aprendí que seguridad efectiva es resultado de decisiones correctas en cada capa, no un solo control. Si usuario manipula frontend, middleware verifica nuevamente. Si manipula middleware, servicio valida con Zod. Si manipula inserción directa, restricciones únicas y enumeraciones previenen datos inválidos. Esta defensa en profundidad según ISO/IEC 27001 A.14.1 permitió entregar módulo resiliente ante errores humanos e intentos maliciosos.

CT4.3 – Identificación y análisis de problemas de investigación

Identifiqué como problema central la ausencia de mecanismos estructurados de recopilación de feedback. La institución no contaba con herramientas para medir satisfacción, evaluar metodologías o recopilar sugerencias de forma organizada y analizable. Recopilación informal mediante papel o mensajería generaba datos dispersos, difíciles de analizar, sin trazabilidad ni segmentación.
Cómo lo hice: Formulé requisitos medibles: "habilitar creación con cinco tipos para feedback estructurado", "garantizar segmentación automática dirigiendo solo a pertinentes", "registrar inmutablemente con timestamp para análisis temporal", "validar completitud de obligatorias asegurando calidad mínima", "calcular métricas evaluando representatividad". Traduje en funcionalidades verificables: panel con tres estados, formulario dinámico renderizando inputs específicos, validaciones defensivas en cuatro capas y cálculo automático de estadísticas.
Analicé causas raíz: falta de herramienta centralizada (resuelto con módulo dedicado), ausencia de control de permisos (tabla PermisoDocente y validaciones por rol), incapacidad de segmentar (segmentación automática pendiente de implementación completa), falta de análisis estructurado (almacenamiento normalizado facilitando agregaciones) y ausencia de trazabilidad (registro de timestamp, IP y tiempo).

Qué usé: Historias de usuario capturando necesidades diferenciadas por rol; criterios con condiciones funcionales verificables definiendo comportamiento en cada escenario; especificación con ejemplos concretos validando contratos; suite con casos positivos y negativos verificando cumplimiento; documentación vinculando funcionalidades a problemas con justificación de decisiones.
Aprendizaje: Desarrollé capacidad para descomponer problemas complejos en componentes funcionales independientes y verificables. Identificar "recopilación de feedback" no es suficiente; debo descomponerlo en subproblemas específicos (¿cómo crear preguntas flexibles?, ¿cómo validar completitud?, ¿cómo prevenir duplicados?, ¿cómo analizar resultados?) proponiendo solución técnica verificable para cada uno. Esta descomposición sistemática facilita estimación, identificación de dependencias y validación progresiva.

CT4.4 – Gestión de proyectos de TI

Planifiqué desarrollo en cinco fases incrementales con entregables progresivos entregando valor temprano: análisis con criterios y reglas (especificación funcional con seis HU), diseño de modelo y contratos (esquema Prisma con cinco entidades y nueve endpoints), implementación con validaciones (código con rutas, controladores, middlewares y servicios), desarrollo de suite (doce casos validando permisos y flujos) y verificación funcional manual (validación de segmentación, prevención de duplicados e inmutabilidad).
Prioricé funcionalidades según dependencias y riesgo: primero panel y validación de acceso (habilitadores verificando segmentación), luego formulario y envío (valor directo habilitando recopilación), después constructor con validaciones (habilitador para autores), luego visualización de respuestas propias (transparencia generando confianza) y finalmente gestión administrativa y análisis (optimización y toma de decisiones). Esto permitió validar segmentación y permisos tempranamente antes de invertir esfuerzo en funcionalidades dependientes.
Cómo lo hice: Organicé trabajo en sprints con entregables incrementando capacidad: primero modelo funcional con migraciones creando tablas necesarias; luego endpoints de lectura (panel, formulario) verificando segmentación sin modificar datos; después endpoints de escritura (crear, responder) con validaciones exhaustivas; luego suite validando cada endpoint y middleware; finalmente verificación manual con usuarios de diferentes roles. Gestioné versionamiento con commits descriptivos vinculando cambios a historias; documenté decisiones de arquitectura justificando elecciones técnicas.
Implementé controles de calidad progresivos: validaciones con Zod en 100% de operaciones de creación y respuesta (cada campo con esquema especificando tipo, formato y restricciones), autorización diferenciada en tres niveles (crear requiere permisos activos, gestionar requiere autoría o director, ver resultados requiere autoría o participación), trazabilidad mediante timestamps (fecha_creacion, fecha_respuesta, fecha_cierre). Revisé cada función verificando manejo de errores con códigos HTTP específicos (400 datos inválidos, 403 permisos insuficientes, 404 no encontrado, 409 conflicto) y mensajes descriptivos.

Qué usé: Cronograma alineando módulo con fase de comunicación; GitHub con estructura por capa (routes, controllers, middlewares, services, tests); historias con criterios verificables; patrón de cuatro capas facilitando desarrollo y testing independiente; testing automatizado con Vitest en ambiente aislado con setup y teardown; documentación actualizada progresivamente.
Aprendizaje: Comprendí que gestión efectiva requiere equilibrar velocidad con calidad verificable. Entregar rápido sin validaciones genera deuda técnica costosa; entregar con calidad excesiva retrasa valor. Balance mediante priorización por riesgo: validaciones críticas de seguridad (permisos, prevención de duplicados) se implementan desde inicio con testing exhaustivo, validaciones de experiencia (mensajes descriptivos, contadores) se refinan iterativamente según feedback. Mantener trazabilidad bidireccional no solo asegura cumplimiento sino facilita estimación de impacto de cambios y planificación de correcciones.

Conclusiones

El módulo de Encuestas habilitó sistema estructurado, flexible y analizable de recopilación de feedback institucional. Se implementaron cinco tipos de preguntas capturando feedback cualitativo (texto corto y largo) y cuantitativo (opción única, múltiple y escala uno a cinco), controles de acceso