PROYECTO INTEGRADOR DEL CURSO

Diseño de Sistemas de Información — 2026-0

Caso único común para todo el aula: “Campus360” Plataforma de Servicios Universitarios

10 grupos (5 alumnos c/u). Cada grupo desarrolla un módulo/servicio del mismo sistema, con un enfoque práctico: arquitectura + UML + patrones + refactoring + pruebas + SOA/REST.

Entregas: Avance (mitad de ciclo) + Entrega final (fin de curso)

¿Cómo se conecta el sílabo con el proyecto?

Lo que se ve en clase se demuestra con evidencia en el repositorio

Mapa de aplicación (temas → evidencias)

Diseño y Arquitectura

- Documento de Arquitectura 4+1 (vistas + escenarios)
- UML: casos de uso, clases, secuencia, componentes y despliegue
- Arquitectura web (MVC/capas) y decisiones justificadas

Calidad de diseño

- Aplicación de principios (SOLID, DRY, Demeter)
- Uso de patrones (creacional/estructural/comportamiento)
- Trazabilidad: requisito → diseño → código

Refactoring y Smells

- Identificación de code smells (mín. 5)
- Refactorings con commits “antes/después”
- Mejora medible: complejidad, duplicación, legibilidad

Implementación y Pruebas

- SOA/REST: endpoints y contratos (OpenAPI o equivalente)
- Pruebas: unitarias + caja negra + evidencia caja blanca
- Pipeline simple o script de ejecución (opcional recomendado)

Caso común: Campus360

Plataforma única de servicios universitarios (autoservicio + backoffice)

Descripción del problema

Contexto

- Los estudiantes realizan múltiples trámites y solicitudes por canales dispersos (presencial, correo, WhatsApp).
- No existe trazabilidad ni tiempos de atención medibles; hay duplicidad de datos y reclamos.
- Se requiere una plataforma web que centralice solicitudes, pagos, reservas, notificaciones y reportes.

Actores (mínimos)

- Estudiante
- Personal administrativo
- Jefe de área / aprobador
- Soporte TI (para incidencias)
- Sistema externo (pasarela de pago / correo)

Objetivo del sistema

- Centralizar un catálogo de trámites y servicios.
- Gestionar el ciclo de vida de solicitudes con SLA y estados.
- Habilitar pagos y reservas cuando aplique.
- Enviar notificaciones y generar tableros de indicadores.

Alcance del MVP

Lo mínimo que TODOS deben respetar (cada grupo en su módulo)

Requisitos funcionales globales (base común)

MVP global (debe poder demostrarse en la demo final)

- Autenticación y roles (mínimo: Estudiante / Admin / Aprobador).
- Creación y seguimiento de solicitudes (estado, historial y SLA básico).
- Catálogo de servicios/trámites (alta/baja/edición y publicación).
- Notificación por evento (solicitud creada, aprobada, observada, cerrada).
- Reportes básicos (volumen, tiempos promedio, pendientes por área).

Reglas mínimas

- Toda solicitud tiene: tipo, solicitante, fecha, estado, prioridad, SLA objetivo.
- Debe existir validación de campos y manejo de errores.
- Auditoría: quién cambió estado y cuándo.
- Seguridad: al menos control por roles en endpoints clave.

No funcionales (mínimos)

- Usabilidad: flujo claro y mensajes de error útiles.
- Rendimiento: operaciones básicas < 2 s en entorno local.
- Mantenibilidad: arquitectura por capas y código limpio.
- Trazabilidad: CU → clases/servicios → commits.

Arquitectura de referencia

La propuesta debe justificarse con el Documento 4+1

Lineamientos arquitectónicos (guía)

Estilo recomendado (pueden ajustar con justificación)

- Front-end: portal web (MVC o SPA) que consume servicios REST.
- Back-end: servicios por módulo (monolito modular o microservicios).
- Capa de dominio y capa de aplicación separadas (evitar “todo en controladores”).
- Persistencia: repositorios (DAO/Repository) y mapeo claro de entidades.

Integración SOA/REST (obligatoria)

- Cada grupo expone endpoints REST para su módulo.
- Al menos 1 consumo de servicio externo (real o simulado):
  - Pago (fake gateway) / Email (SMTP mock) / SMS/WhatsApp mock.
- Documentar contrato: endpoints, request/response, códigos de error.

Buenas prácticas de diseño

- Aplicar SOLID (SRP, OCP, LSP, ISP, DIP) donde corresponda.
- DRY: evitar duplicación (validaciones, mapeos, reglas).
- Ley de Demeter: evitar “a.getB().getC().doX()”.
- Convenciones: estructura de carpetas, nombres, README.

Asignación por grupos

Todos trabajan el mismo caso, cada grupo implementa un módulo/servicio

Módulos de Campus360 (10 grupos)

Grupo / Módulo	Responsabilidad principal
G1 — Identidad y Accesos	Usuarios, roles, login, tokens/sesión, políticas de autorización.
G2 — Catálogo de Servicios	CRUD de trámites/servicios, requisitos, costos, tiempos y publicación.
G3 — Gestión de Solicitudes	Creación, estados, historial, adjuntos mínimos, SLA y reglas.
G4 — Flujo de Aprobaciones	Bandeja de aprobador, observaciones, aprobaciones multinivel (simple).
G5 — Pagos (Gateway simulado)	Orden de pago, confirmación, comprobante. Integración REST mock.
G6 — Reservas de Ambientes	Reserva de salas/labs, disponibilidad, conflictos, confirmación.
G7 — Notificaciones	Plantillas y envío (email/mock). Suscripción a eventos del sistema.
G8 — Reportes y KPIs	Indicadores (pendientes, TAT, SLA cumplido), export simple.
G9 — Auditoría y Monitoreo	Bitácora de eventos, trazabilidad de cambios, logs consultables.
G10 — API Gateway / Portal Integrador	Punto de entrada (router) + documentación OpenAPI + demo UI.

Backlog base común

Historias mínimas que sirven como guía (cada grupo toma las de su módulo)

Historias de usuario (mínimo sugerido)

Core del sistema

- Como estudiante, quiero iniciar sesión para acceder a mis trámites.
- Como estudiante, quiero crear una solicitud seleccionando un servicio del catálogo.
- Como estudiante, quiero ver el estado e historial de mi solicitud.
- Como aprobador, quiero aprobar u observar solicitudes con comentarios.
- Como admin, quiero publicar/actualizar servicios del catálogo.

Servicios complementarios

- Como estudiante, quiero pagar un trámite cuando tenga costo.
- Como estudiante, quiero reservar un ambiente si el servicio lo requiere.
- Como sistema, quiero notificar cambios de estado al estudiante y al área.
- Como jefe, quiero reportes de carga y tiempos por área.
- Como auditor, quiero consultar quién cambió qué y cuándo.

Restricciones y estándares mínimos

Para que todos puedan evaluar y comparar resultados

Reglas de construcción del proyecto

Tecnología (libre, con mínimos)

- Lenguaje/stack libre (Java/.NET/Node/Python/etc.).
- Debe existir estructura en capas y separación de responsabilidades.
- Persistencia: BD ligera (SQLite/PostgreSQL) o repositorio en memoria (justificado).
- API documentada (OpenAPI/Swagger o README con contratos).

Control de versiones (obligatorio)

- Repo Git con ramas o commits ordenados.
- Convención de commits (feat/fix/refactor/test/docs).
- Etiquetas: v1-avance y v2-final.
- Pull requests sugeridos (si trabajan en GitHub/GitLab).

Calidad y evidencia (obligatorio)

- Aplicar 3 patrones (1 creacional + 1 estructural + 1 comportamiento).
- Reportar y corregir 4 code smells con refactoring.
- Pruebas: 6 unitarias + 8 caja negra + evidencia caja blanca.
- README ejecutable: pasos para levantar y probar.

Cronograma y hitos

Plan sugerido para llegar con tranquilidad al avance y al final

Hitos por semanas (referencial)

Semana	Qué se espera
Sem 1–2	Formación de grupos. Entender el caso Campus360. Alcance, actores, requisitos y NFR.
Sem 3	Entregable 0: Propuesta y alcance + repositorio + backlog inicial.
Sem 4–6	UML (CU, dominio). Diseño de interfaces. Documento de Arquitectura 4+1 (borrador).
Sem 7–8	Arquitectura final + contratos API. Prototipo skeleton. Pruebas iniciales.
Sem 8–9	✅ AVANCE (PE1): Arquitectura 4+1 + UML + prototipo mínimo.
Sem 10–12	Implementación del MVP del módulo. Patrones aplicados. Integración REST.
Sem 13–14	Refactoring (smells) + hardening. Pruebas (unitarias/caja negra/caja blanca).
Sem 15–16	✅ ENTREGA FINAL (PE2): demo + informe + evidencias + repositorio release.

Entregable 0 (Sem 3)

Propuesta y alcance del módulo dentro de Campus360

E0 — Qué entregar

Documento (PDF 3–5 páginas)

- Resumen del módulo asignado (objetivo y alcance).
- Actores que interactúan con el módulo.
- Requisitos funcionales del módulo (mín. 6) + reglas de negocio.
- Requisitos no funcionales aplicables (mín. 5).
- Backlog: historias priorizadas (mín. 7) + criterios de aceptación.

Repositorio (link)

- Estructura inicial del proyecto + README.
- Definición de endpoints preliminares (tabla simple).
- Plan de trabajo por semana y roles del equipo.
- Tablero (Trello/Jira/GitHub Projects) — opcional recomendado.
- Tag/Release: v0-propuesta (opcional).

AVANCE (PE1) — Mitad de ciclo

Diseño y arquitectura con evidencia técnica

PE1 — Entregables obligatorios

Paquete PE1

- Documento de Arquitectura 4+1 (por módulo y su integración).
- UML completo del módulo (CU, clases, secuencia, componentes, despliegue).
- Contrato API (endpoints + ejemplos request/response + errores).

4+1 — Checklist (mínimo)

- Vista lógica: capas, clases de diseño, paquetes.
- Vista de procesos: 2–3 secuencias clave.
- Vista de desarrollo: componentes y dependencias.
- Vista física: despliegue local (y puertos).
- +1 escenarios: 2 escenarios “significativos”.

Prototipo mínimo (recomendado)

- Skeleton ejecutable con endpoints stub (respuestas dummy).
- Estructura por capas (controller/service/domain/repository).
- 1 prueba unitaria inicial (smoke test).
- Tag obligatorio: v1-avance.

ENTREGA FINAL (PE2)

MVP funcional + calidad (patrones, refactoring, pruebas)

PE2 — Entregables obligatorios

Paquete PE2

- MVP del módulo funcional (flujos completos) + integración REST.
- 3 patrones aplicados y explicados con evidencia en el código.
- Refactoring de 4 smells con comparativo antes/después (commits).
- Pruebas: unitarias + caja negra + evidencia caja blanca.
- Informe final + demo.

Pruebas — mínimos

- 6 pruebas unitarias (servicios/validadores).
- 8 casos caja negra (tabla con entradas/salidas).
- Caja blanca: cobertura o evidencia de ramas/sentencias.
- Script: npm test / mvn test / dotnet test / pytest, etc.

Demo — estructura sugerida

- 1 min: problema y alcance del módulo.
- 2 min: arquitectura + UML clave.
- 3–4 min: demo del flujo principal.
- 1 min: patrones + refactoring.
- 1 min: pruebas y conclusiones.

Plantilla de contrato API

Para que todas las integraciones sean consistentes

Formato mínimo (por endpoint)

Completar por cada endpoint

- Método + ruta (GET/POST/PUT/DELETE) y descripción.
- Autorización requerida (rol).
- Request: parámetros, body, validaciones.
- Response: estructura JSON y ejemplos.
- Códigos de error (400/401/403/404/409/500) y mensajes.
- Idempotencia y reglas de negocio relevantes.

Ejemplo rápido:

POST /solicitudes — Crea una nueva solicitud | Roles: Estudiante | Errores: 400(validación), 409(duplicada)

Plantilla: Caso de uso y escenarios

Para el +1 de arquitectura y para trazabilidad

Formato breve recomendado

Caso de uso (1–2 páginas)

- Nombre, actores, objetivo, pre/post condiciones.
- Flujo principal (pasos numerados).
- Flujos alternos / excepciones.
- Reglas de negocio y validaciones.
- Datos involucrados (entidades).

Escenario arquitectónico (+1)

- Nombre del escenario (p.ej., “Pago de trámite”).
- Qué componentes participan (servicios).
- Diagrama de secuencia del flujo.
- Riesgos (fallos, latencia, concurrencia).
- Decisiones tomadas (patrones, resiliencia básica).

Patrones de diseño

Mínimo 3 patrones con evidencia en código y explicación

Selección sugerida (pueden cambiar con justificación)

Creacionales (elige 1)

- Factory Method
- Abstract Factory
- Builder
- Singleton (solo si aplica y con cuidado)

Estructurales (elige 1)

- Adapter (integración con gateway)
- Facade (simplificar acceso a subsistemas)
- Decorator (validaciones / logging)
- Proxy (control de acceso / cache)

Comportamiento (elige 1)

- Strategy (cálculo de SLA/prioridad)
- Observer (notificaciones por eventos)
- State (ciclo de vida de solicitud)
- Command (acciones de aprobación)

Code Smells y Refactoring

Identificar 5 y corregir con evidencia “antes/después”

Lista guía (ejemplos)

Smells comunes

- Long Method / Large Class
- Duplicated Code (violación DRY)
- Feature Envy / Message Chains (Demeter)
- God Object / Low cohesion
- Shotgun Surgery / Divergent Change
- Too many parameters

Evidencia requerida

- Tabla: smell → archivo/línea → impacto → refactoring aplicado.
- Commits separados: refactor: ...
- Antes/después: fragmento breve o diff (en informe).
- Validación: pruebas pasando luego del refactor.
- Métrica opcional: complejidad ciclomática / duplicación.

Plan de pruebas

Evidencia de pruebas por módulo

Qué entregar en el informe (anexos)

Caja negra (mín. 6 casos)

- ID, requisito asociado, precondición.
- Entrada (request / UI) y pasos.
- Resultado esperado vs obtenido.
- Estado (OK/Fail) + evidencia (captura o log).
- Casos negativos: validaciones y permisos.

Unitarias + Caja blanca

- Unitarias (mín. 8): servicios/validadores.
- Aislar dependencias con mocks/stubs.
- Caja blanca: reporte de cobertura o evidencia de ramas.
- Ejecución reproducible: comando estándar.
- Registro de defectos corregidos (opcional recomendado).

Gestión del equipo

Cómo organizarse para cumplir y evidenciar trabajo real

Roles sugeridos (rotación 1 vez)

Roles

- Arquitecto/a: 4+1, decisiones, revisión SOLID/DRY/Demeter.
- Analista: CU, dominio, NFR, trazabilidad.
- Dev: implementación + patrones + integración.
- QA/DevOps: pruebas, cobertura, automatización y documentación.

Definition of Done (por historia)

- Código integrado en main/develop.
- Pruebas pasando.
- Documentación actualizada (README/contrato).
- Revisión por un compañero (code review).

Trazabilidad mínima (obligatoria)

- Cada requisito/historia debe enlazar: CU → clases/servicios → commit(s).
- En el informe, incluir una tabla corta de trazabilidad (10 filas mínimo).

Criterios de evaluación (guía)

Lo que más pesa: coherencia, evidencia y calidad

Rúbrica resumida

Criterio	Qué se espera
Arquitectura + UML (4+1)	Vistas completas, decisiones justificadas, escenarios claros y consistentes con el código.
Diseño (SOLID/DRY/Demeter)	Separación de responsabilidades, bajo acoplamiento, alta cohesión, claridad.
Patrones	3 patrones bien aplicados, explicados y visibles en la estructura del sistema.
Refactoring (smells)	Smells bien identificados y refactorings con evidencia “antes/después” y pruebas.
SOA/REST	Endpoints coherentes, contrato claro, integración funcionando (o mock robusto).
Pruebas	Unitarias + caja negra + evidencia caja blanca, ejecución reproducible.
Documentación + Demo	README ejecutable, informe ordenado, demo clara y estable.

Checklist de entrega

Si falta evidencia, el puntaje cae aunque el sistema “funcione”

Lista de verificación (PE2)

Repositorio

- Tag v2-final creado.
- README con instalación, ejecución y pruebas.
- Contratos API documentados.
- Commits de refactoring separados y descriptivos.
- Carpeta /docs con UML e informe.

Informe (PDF)

- Arquitectura 4+1 final + diagramas.
- Explicación de patrones (3) con evidencia.
- Tabla de smells + refactorings (4).
- Plan de pruebas (unitarias + caja negra + caja blanca).
- Lecciones aprendidas y riesgos.