“La recuperación y consolidación de la economía peruana" en Perú”

UNIVERSIDAD PERUANA LOS ANDES

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

ARQUITECTURA DE SOFTWARE

Integrantes:

Limaya Carhuallamque Sebastian

Retamozo Ataucusi Josue Nehemias

Sanchez Romero Aldair

Guerra Yarupaita Carlos Daniel

Docente:

FERNANDES BEJARANO RAUL

Ciclo:

8 B”2”

HUANCAYO – PERÚ

2025

ACTIVIDADES DE APRENDIZAJE

Tema: Principios de Diseño Arquitectónico

Actividad 1: Taller de análisis y aplicación de principios arquitectónicos

- **Descripción**:
  Los estudiantes conforman equipos de proyecto y seleccionan un caso de estudio de una aplicación real (ejemplo: sistema de gestión académica, comercio electrónico, aplicación bancaria). A partir de este contexto, identifican y discuten los principios de diseño arquitectónico (modularidad, cohesión, acoplamiento bajo, reutilización, separación de intereses, entre otros), aplicándolos al diseño preliminar del sistema.

- **Trabajo esperado:**
  Elaboración de un documento que explique los principios seleccionados, su justificación y la manera en que estos se aplican al caso de estudio, acompañado de un primer borrador de diagramas de componentes y capas.

Evidencia: Documento de diseño arquitectónico con diagramas.

Técnica de recolección: e-portafolio.

Instrumento de evaluación: Rúbricas de aprendizaje.

Indicador de evaluación: El estudiante presenta una propuesta arquitectónica coherente con diagramas claros y fundamentación en estándares internacionales.

1. Introducción

En el contexto actual de la transformación digital, las instituciones educativas requieren sistemas robustos, escalables y seguros para gestionar sus procesos internos. El diseño arquitectónico de software juega un papel crucial en este objetivo, al definir la estructura, los componentes, las interacciones y los principios fundamentales que guían el desarrollo de un sistema.

El presente documento tiene como propósito analizar y aplicar los principios, estilos y patrones arquitectónicos al desarrollo del Sistema de Gestión Académica Universitaria **(SIGA-UPLA).** Este sistema permitirá la gestión de matrículas, asignaturas, calificaciones, reportes y usuarios, brindando soporte tanto a docentes, estudiantes y personal administrativo.

Este trabajo combina teoría, justificación técnica, normas internacionales y diagramas arquitectónicos, con el fin de construir una propuesta sólida, coherente y alineada con los estándares ISO/IEC

2. Objetivos

Objetivo general

Diseñar y justificar una propuesta arquitectónica para el sistema SIGA-UPLA aplicando principios de diseño, estilos y patrones arquitectónicos que garanticen calidad, escalabilidad, reutilización y mantenibilidad.

Objetivos específicos

1. Identificar los principios de diseño arquitectónico y aplicarlos al sistema propuesto.
2. Seleccionar los estilos arquitectónicos más adecuados según los requerimientos de calidad.
3. Implementar patrones arquitectónicos que optimicen la interacción entre componentes.
4. Presentar diagramas arquitectónicos (componentes, capas, despliegue y secuencia).
5. Fundamentar las decisiones en normas internacionales (ISO/IEC 42010, ISO/IEC 25010, IEEE 1471).

3. Caso de estudio: Sistema de Gestión Académica UPLA

El **SIGA-UPLA** es una plataforma web diseñada para administrar el ciclo de vida académico de estudiantes, docentes y administrativos. Su propósito es automatizar los procesos que actualmente se realizan manualmente o mediante sistemas aislados.

3.1 Usuarios

· Estudiante: realiza matrícula, visualiza notas y horarios.

· Docente: registra calificaciones, asistencia y materiales.

· Administrador académico: gestiona usuarios, cursos, semestres y reportes.

· Soporte técnico: supervisa el funcionamiento y mantenimiento del sistema.

3.2 Funcionalidades clave

· Registro y autenticación de usuarios (Google y correo institucional).

· Gestión de cursos, horarios, aulas y planes de estudio.

· Registro de notas, asistencia y reportes automáticos.

· Emisión de certificados digitales.

· Dashboard con indicadores académicos.

3.3 Requerimientos no funcionales

· Disponibilidad: 99.9 % garantizada mediante hosting en la nube (Supabase + Vercel).

· Seguridad: autenticación JWT y encriptación SHA256.

· Escalabilidad: arquitectura modular y servicios desacoplados.

· Mantenibilidad: uso de control de versiones (Git/GitHub).

· Portabilidad: compatible con navegadores y dispositivos móviles.

4. Principios de Diseño Arquitectónico

Los principios de diseño definen las bases que permiten construir sistemas de software robustos. En el caso del SIGA-UPLA, se aplican los siguientes principios:

Principio	Definición	Aplicación en SIGA-UPLA
Modularidad	División del sistema en módulos funcionales autónomos.	Módulos: matrícula, cursos, notas, reportes, usuarios.
Cohesión	Relación interna de los componentes de un módulo.	El módulo de “matrícula” sólo gestiona inscripciones y pagos.

Bajo acoplamiento	Reducción de dependencias entre módulos.	Comunicación por API REST entre cliente y servidor.
Reutilización	Uso de componentes en diferentes contextos.	Módulo de autenticación usado por todos los perfiles.
Separación de intereses	División de responsabilidades por capas.	Presentación (frontend), negocio (backend), datos (BD).
Escalabilidad	Capacidad de crecer sin afectar el rendimiento.	Implementación basada en microservicios.
Mantenibilidad	Facilidad para corregir y extender el sistema.	Código versionado, documentado y probado.
Portabilidad	Capacidad de ejecutarse en diferentes entornos.	Despliegue multiplataforma (nube/local).

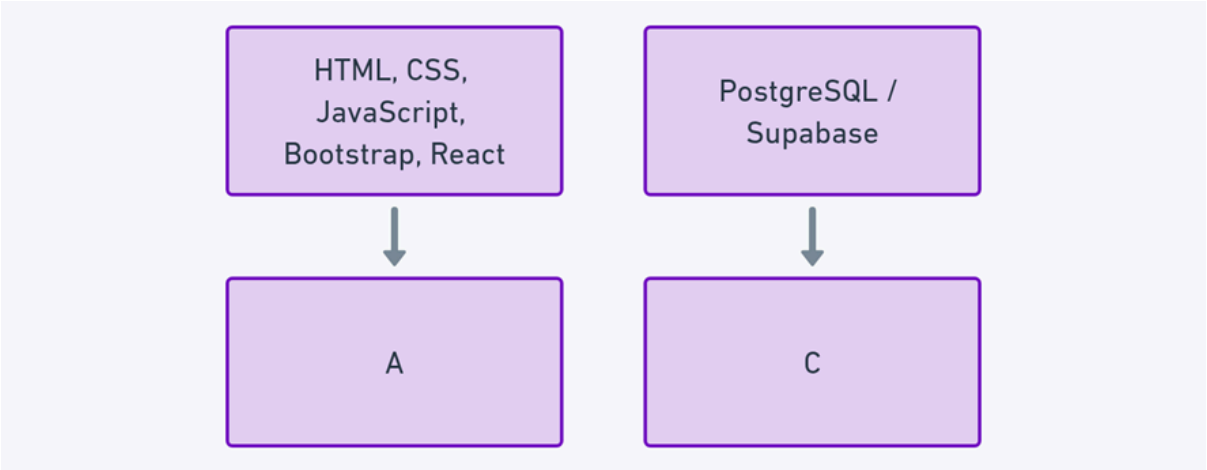
5. Justificación de los principios

1. Modularidad y cohesión permiten aislar los errores y distribuir tareas entre equipos de desarrollo.
2. Bajo acoplamiento asegura independencia entre módulos, mejorando el mantenimiento.
3. La separación de intereses permite asignar responsabilidades específicas: UI (interfaz), lógica de negocio y persistencia de datos.
4. La reutilización reduce tiempo y costos de desarrollo.
5. Escalabilidad permite que el sistema crezca con el número de usuarios.

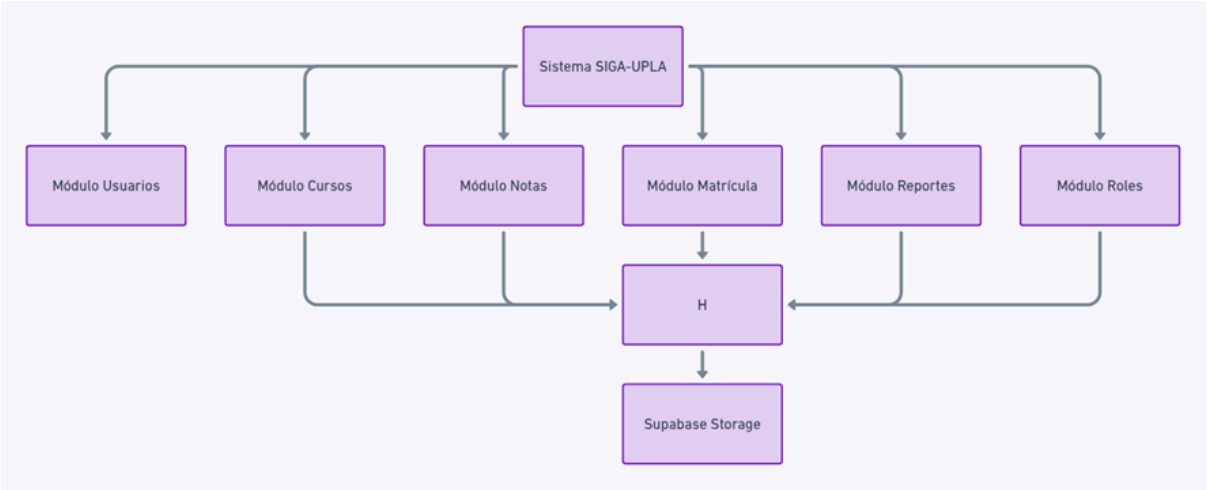
Estos principios se alinean con la norma ISO/IEC/IEEE 42010:2011, que recomienda la definición clara de vistas, stakeholders y objetivos de calidad arquitectónica.

6. Diseño preliminar del sistema

6.1 Diagrama de capas



6.2 Diagrama de componentes



7. Estilos Arquitectónicos

7.1 Cliente-Servidor

- **Cliente**: interfaz web (React/Bootstrap).
- **Servidor**: backend API REST (FastAPI o Node.js).
- **Ventajas**: separación clara, escalabilidad, distribución de carga.
- **Desventaja**: requiere conexión constante a Internet.

7.2 Arquitectura en Capas

Permite estructurar el sistema en niveles jerárquicos, lo cual mejora la seguridad, mantenibilidad y claridad.

Capa	**Responsabilidad**
Presentación	Interacción con el usuario.
Negocio	Procesamiento de reglas lógicas.
Datos	Persistencia y recuperación de información.
Infraestructura	Configuración del entorno y despliegue.

7.3 Arquitectura Orientada a Servicios (SOA)

Cada módulo funciona como un servicio independiente y puede comunicarse mediante HTTP/JSON.
Ejemplo: módulo de calificaciones expuesto como servicio que puede ser consumido por otros sistemas (biblioteca virtual o control de pagos).

7.4 Microservicios

Permiten dividir el sistema en pequeños servicios desplegables de manera independiente.
Ejemplo: un microservicio para gestión de notas, otro para matrícula, otro para reportes.

Beneficios: escalabilidad horizontal, tolerancia a fallos, mantenimiento independiente.
Desafíos: mayor complejidad de despliegue y monitoreo.

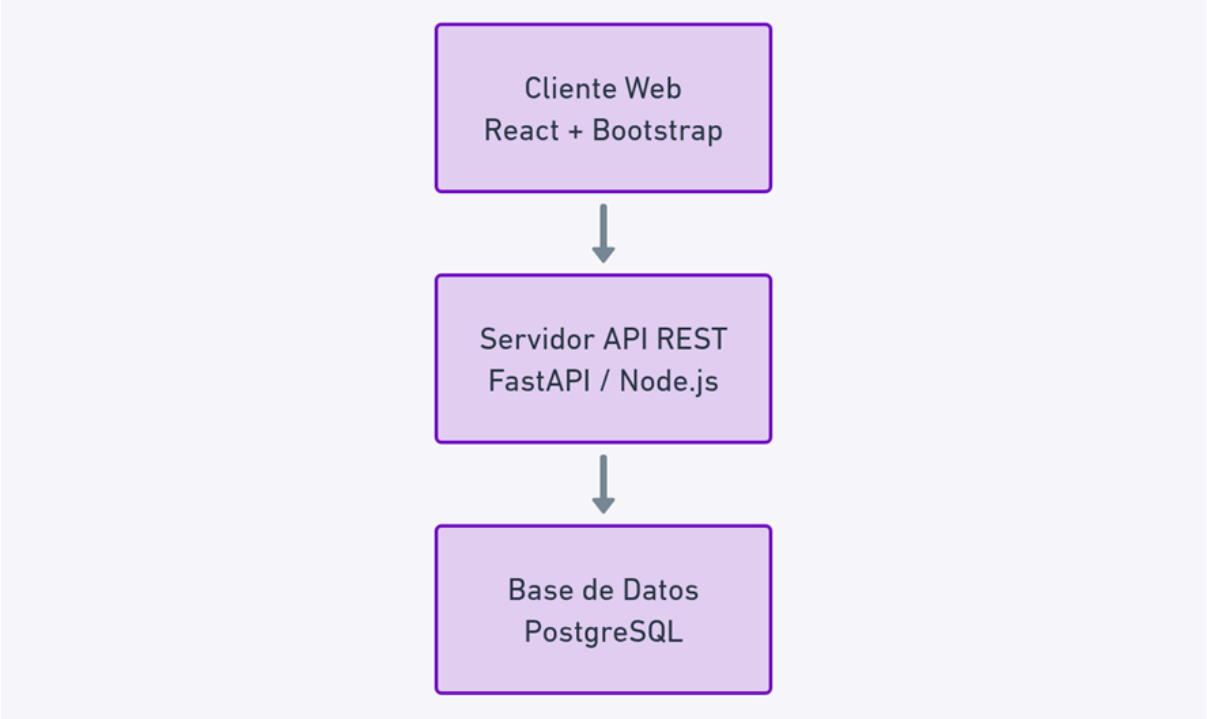
8. Patrones Arquitectónicos

Patrón	**Descripción**	**Aplicación**
MVC (Modelo-Vista-Controlador)	Separa presentación, lógica y datos.	Aplicado en el portal web (estudiantes/docentes).
Repository	Aísla el acceso a datos del negocio.	CRUDs de usuarios, cursos y notas.

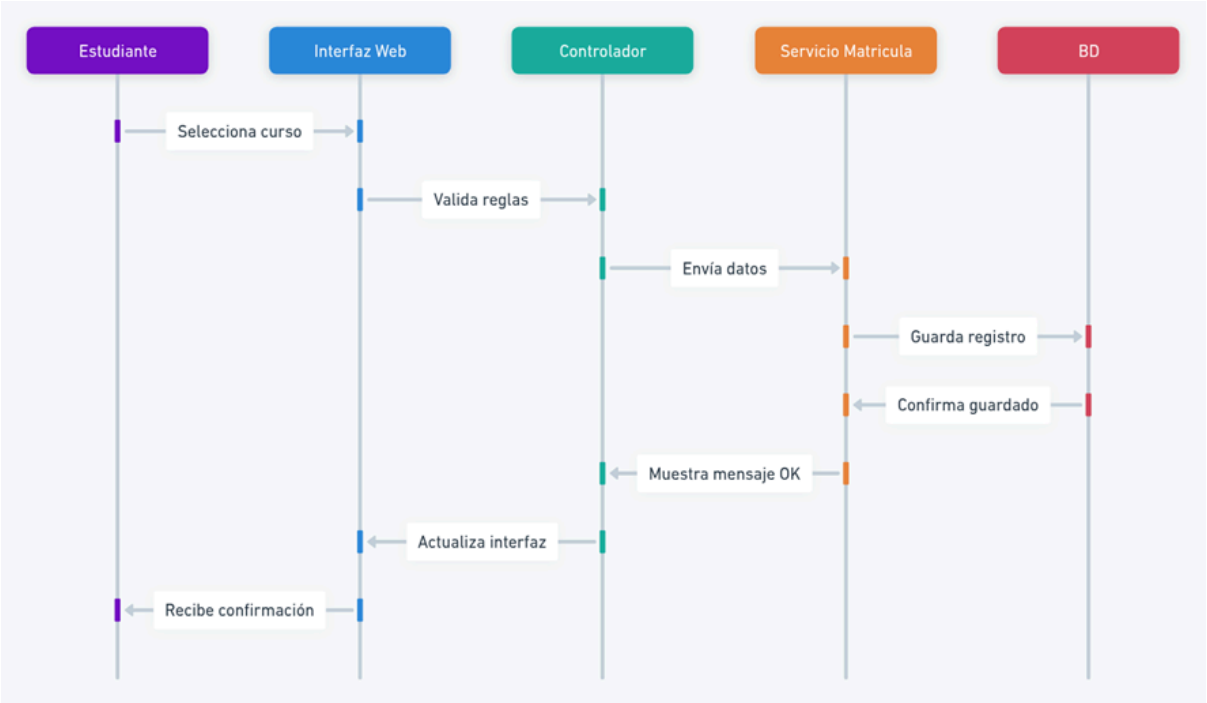
Singleton	Garantiza una única instancia global.	Conexión centralizada a base de datos.
Observer	Permite suscripción a eventos.	Notificaciones automáticas de notas nuevas.
Facade	Crea una interfaz simplificada.	Dashboard que integra reportes académicos.
Pipe and Filter	Procesamiento secuencial de datos.	Generación de reportes PDF y filtrado por semestre.
Broker	Coordina comunicación entre componentes distribuidos.	Intermediario entre módulos del sistema y servicios externos.

9. Diagramas adicionales

9.1 Diagrama de despliegue



9.2 Diagrama de secuencia (flujo de matrícula)



10. Justificación técnica

El diseño se sustenta en los siguientes estándares internacionales:

Norma	Descripción	Aplicación
ISO/IEC/IEEE 42010	Define la descripción de arquitectura de sistemas.	Permite documentar vistas, stakeholders y decisiones.
ISO/IEC 25010	Define los atributos de calidad del software.	Base para evaluar rendimiento, mantenibilidad y seguridad.
IEEE 1471	Enfatiza la coherencia entre arquitectura y requerimientos.	Refuerza la trazabilidad de los módulos.

11. Evaluación de atributos de calidad (ISO/IEC 25010)

Atributo	**Criterio**	**Cumplimiento en SIGA-UPLA**
Mantenibilidad	Modularidad, documentación, pruebas.	Alta
Escalabilidad	Microservicios y SOA.	Media-Alta
Usabilidad	Interfaz moderna y accesible.	Alta
Seguridad	Roles, JWT y HTTPS.	Alta
Compatibilidad	Multiplataforma.	Alta
Rendimiento	Cache y optimización de consultas.	Alta
Fiabilidad	Pruebas automatizadas y backup.	Alta

---

12. Riesgos y mitigaciones

Riesgo	**Descripción**	**Mitigación**
Fallos del servidor	Sobrecarga de peticiones	Balanceo de carga (NGINX)
Pérdida de datos	Error humano o técnico	Backups automáticos
Fallos de autenticación	Contraseñas débiles	Encriptación y OAuth
Baja mantenibilidad	Código sin estructura	Aplicar GitFlow y documentación

13. Conclusiones

El **SIGA-UPLA** demuestra la aplicación práctica de principios y patrones de diseño arquitectónico, garantizando un sistema escalable, mantenible y adaptable.
Los estilos arquitectónicos seleccionados (cliente-servidor, capas, SOA y microservicios) permiten una implementación progresiva y alineada con estándares internacionales.
Los patrones como **MVC, Repository y Observer** fortalecen la cohesión interna y la calidad del código.

Este diseño representa una propuesta coherente y sostenible para la **gestión académica universitaria**, siendo adaptable a futuras ampliaciones e integraciones.

14. Referencias bibliográficas

· ISO/IEC/IEEE 42010:2011 – Systems and Software Engineering — Architecture Description.

· ISO/IEC 25010:2011 – Systems and Software Quality Requirements and Evaluation (SQuaRE).

· Bass, L., Clements, P., & Kazman, R. (2021). Software Architecture in Practice (4th ed.). Addison-Wesley.

· IEEE Std 1471-2000 – Recommended Practice for Architectural Description of Software-Intensive Systems.

· Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.

· Shaw, M., & Garlan, D. (1996). Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.

Tema: Estilos y Patrones Arquitectónicos

Actividad 2: Aplicación práctica de estilos y patrones arquitectónicos

- Descripción:

  Cada equipo selecciona al menos un estilo arquitectónico (cliente-servidor, en capas, orientado a servicios, microservicios) y un conjunto de patrones arquitectónicos (MVC, broker, pipo-and-filter, etc.) adecuados para el caso de estudio definido en la Actividad 1. Se justifica su elección on relación con los requerimientos de calidad (rendimiento, oscalabilidad, mantenibilidad), Posteriormente, los estudiantes

elaboran diagramas arquitectonicos detallados que reflejen la integración de estilos y patrones seleccionados.

- Trabajo esperado:

  Documento que describa los estilos y patrones aplicados, su justificación técnica, y diagramas de alto nivel que muestren la estructura resultante del sistema.

Evidencia: Documento de diseño arquitectónico con diagramas.

Técnica de recolección: e-portafolio.

Instrumento de evaluación: Rúbricas de aprendizaje.
Indicador de evaluación: El estudiante presenta una propuesta arquitectónica coherente, con diagramas claros y fundamentación en estándares internacionales.

1. Introducción

En la ingeniería de software, los **estilos y patrones arquitectónicos** representan un conjunto de soluciones probadas que orientan la forma en que se estructura un sistema. Un estilo arquitectónico define el marco general que guía la organización de los componentes y su comunicación, mientras que los patrones proporcionan soluciones recurrentes a problemas específicos de diseño.

En esta actividad, se aplican ambos conceptos al **Sistema de Gestión Académica UPLA (SIGA-UPLA)**, un proyecto orientado a la automatización de los procesos académicos y administrativos dentro de la universidad. El objetivo es diseñar una propuesta arquitectónica coherente que garantice **rendimiento**, **escalabilidad**, **seguridad y mantenibilidad**, todo bajo estándares internacionales de calidad de software como ISO/IEC 42010 e ISO/IEC 25010.

2. Objetivo General

Diseñar una propuesta arquitectónica para el SIGA-UPLA aplicando estilos y patrones arquitectónicos que fortalezcan su estructura interna, mejoren la calidad del software y permitan su crecimiento a futuro.

3. Objetivos Específicos

1. Identificar los estilos arquitectónicos más apropiados para el contexto del sistema.
2. Seleccionar patrones arquitectónicos que optimicen el diseño interno y la interacción entre módulos.

3. Elaborar diagramas arquitectónicos que muestren la estructura, los flujos de datos y la comunicación entre componentes.

4. Justificar las decisiones técnicas según los estándares internacionales y las métricas de calidad del software.

4. Contexto del caso de estudio

El **SIGA-UPLA** (**Sistema de Gestión Académica UPLA**) es una aplicación web que busca integrar todos los procesos relacionados con la administración académica, desde la matrícula de estudiantes hasta la gestión de calificaciones y reportes institucionales.

4.1. Problema actual

Actualmente, muchas universidades presentan sistemas fragmentados o manuales, lo que genera duplicidad de información, pérdida de datos y lentitud en la atención de procesos administrativos.

4.2. Solución propuesta

El SIGA-UPLA propone una solución integral, modular y escalable que centraliza todos los procesos académicos bajo una sola plataforma.

4.3. Usuarios principales

Estudiantes: realizan matrículas, consultan horarios y notas.

Docentes: registran calificaciones y asistencia.

Administradores: gestionan usuarios, cursos, reportes y semestres.

Soporte técnico: supervisa mantenimiento y actualizaciones.

5. Requerimientos de calidad (ISO/IEC 25010)

Atributo	**Descripción**	**Necesidad en SIGA-UPLA**
Rendimiento	Respuesta rápida en tiempo real.	Responder a múltiples solicitudes concurrentes.
Escalabilidad	Capacidad de crecer sin perder eficiencia.	Soportar mayor número de estudiantes cada semestre.

Mantenibilidad	Facilidad para modificar y actualizar.	Permitir nuevas funciones sin reestructurar todo el sistema.
Disponibilidad	Operación continua del servicio.	Funcionamiento 24/7 en la nube.
Seguridad	Protección de datos y accesos.	Implementar autenticación y control por roles.
Compatibilidad	Ejecución en distintos entornos.	Compatible con navegadores y móviles.

---

6. Selección del estilo arquitectónico

Después de analizar los requerimientos funcionales y no funcionales, se optó por combinar tres estilos arquitectónicos complementarios:

1. **Cliente-Servidor**
2. **Arquitectura en Capas**
3. **Orientada a Servicios (SOA) / Microservicios**

Cada uno contribuye a un aspecto específico del diseño: comunicación, estructura interna y escalabilidad.

7. Estilo Cliente-Servidor

7.1 Descripción

El estilo **cliente-servidor** divide la aplicación en dos entidades:

· **Cliente**: se encarga de la presentación y la interacción con el usuario.

· **Servidor**: gestiona la lógica de negocio y la base de datos.

Este modelo es la base de la mayoría de las aplicaciones web modernas.

7.2 Ventajas

· Separación clara entre interfaz y lógica.

· Escalabilidad horizontal (pueden agregarse más servidores).

· Independencia tecnológica (el cliente puede ser web, móvil, etc.).

7.3 Aplicación en SIGA-UPLA

Cliente: interfaz web desarrollada con **HTML, CSS, Bootstrap y React.js.**

Servidor: lógica de negocio en **Python FastAPI** o **Node.js (Express).**

Comunicación: mediante **API REST (HTTP/JSON).**

7.4 Ejemplo

El cliente envía una solicitud **GET /api/estudiantes/123** → el servidor procesa la lógica, consulta la base de datos PostgreSQL y devuelve un JSON con los datos del estudiante.

8. Arquitectura en Capas (Layered Architecture)

8.1 Descripción

Organiza el sistema en capas jerárquicas que separan las responsabilidades, asegurando bajo acoplamiento y alta cohesión.

8.2 Capas del SIGA-UPLA

Capa	**Responsabilidad**	**Tecnologías usadas**
Presentación (UI)	Interfaz visual para los usuarios.	React.js / Bootstrap
Negocio (Lógica)	Procesamiento de reglas académicas.	FastAPI / Node.js
Persistencia (Datos)	Almacenamiento y recuperación de información.	PostgreSQL / Supabase
Infraestructura (Servicios)	Gestión de servidores, APIs y seguridad.	Nginx / JWT / OAuth2

8.3 Ventajas

· Facilita mantenimiento y pruebas.

· Permite reemplazar capas sin afectar otras.

· Escalable y segura.

9. Arquitectura Orientada a Servicios (SOA) / Microservicios

9.1 Descripción

Cada módulo del sistema actúa como un servicio independiente, comunicándose con los demás mediante interfaces bien definidas (APIs).

9.2 Aplicación

- Microservicio de Matrícula → gestiona inscripciones y pagos.
- Microservicio de Calificaciones → almacena y calcula promedios.
- Microservicio de Usuarios → administra roles y permisos.
- Microservicio de Reportes → genera informes PDF y dashboards.

9.3 Beneficios

- Escalabilidad independiente por módulo.
- Mayor tolerancia a fallos.
- Despliegue separado y actualización modular.

9.4 Desafíos

- Complejidad de orquestación y monitoreo.
- Requiere control de comunicación entre servicios.

10. Patrones arquitectónicos aplicados

Los **patrones arquitectónicos** son soluciones reutilizables que resuelven problemas de diseño dentro de un contexto determinado. A continuación, se detallan los seleccionados para el SIGA-UPLA:

Patrón	**Descripción**	**Aplicación en SIGA-UPLA**
MVC (Modelo-Vista-Controlador)	Separa la lógica del negocio, la interfaz y el control.	Implementado en la capa de presentación.
Repository	Aísla el acceso a datos del negocio.	CRUDs de usuarios, cursos, notas.

Singleton	Crea una sola instancia global.	Conexión central a la base de datos.
Observer	Notifica cambios a componentes suscritos.	Envío automático de alertas y notas.
Pipe and Filter	Procesa datos en secuencia de filtros.	Generación de reportes y exportación CSV.
Broker	Coordina comunicación entre servicios distribuidos.	Intermediario entre microservicios del sistema.
Facade	Simplifica subsistemas complejos en una sola interfaz.	Dashboard administrativo.

---

11. Integración de estilos y patrones

La integración de los estilos y patrones seleccionados permite construir una arquitectura moderna, escalable y segura.

Ejemplo de integración:

El patrón MVC opera dentro de la capa de presentación del estilo en capas.

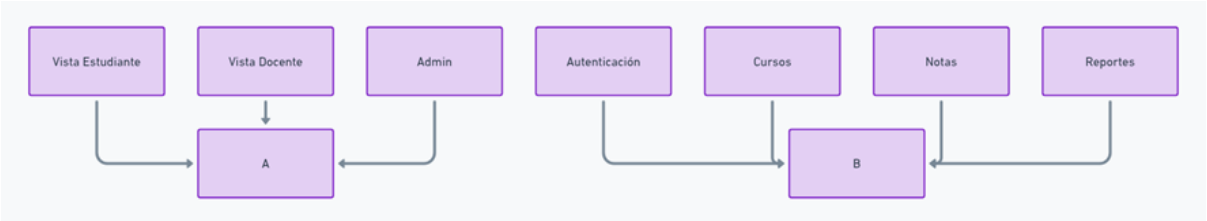
Los microservicios siguen el patrón Broker para intercambiar datos.

La capa de negocio implementa Repository y Facade para acceso eficiente y simplificación de lógica.

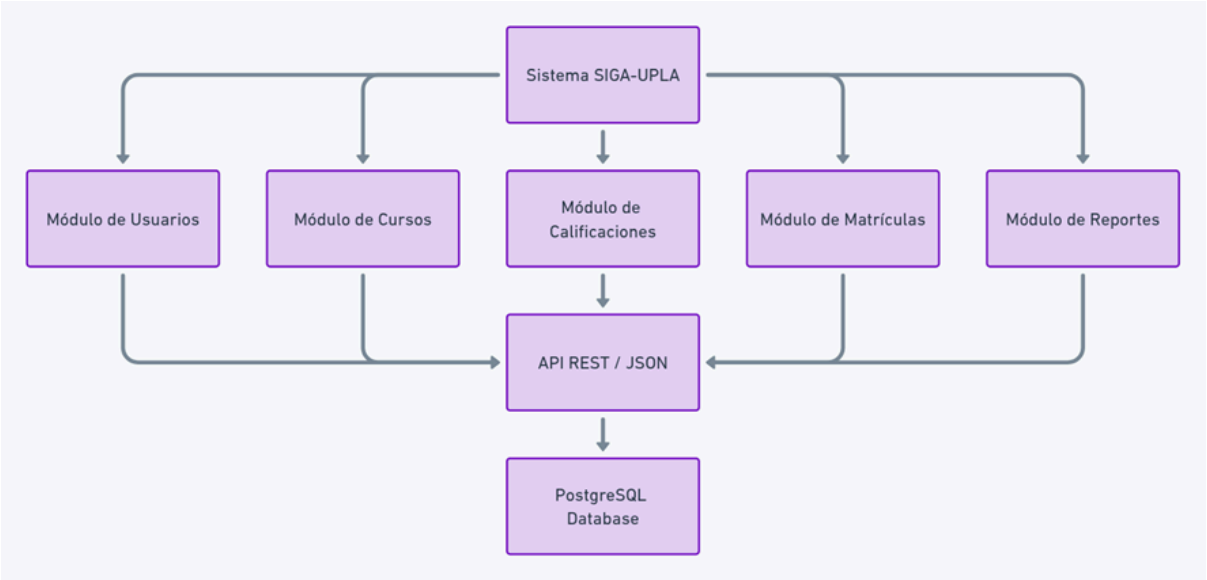
El sistema global se comunica bajo el esquema Cliente-Servidor.

12. Diagramas Arquitectónicos

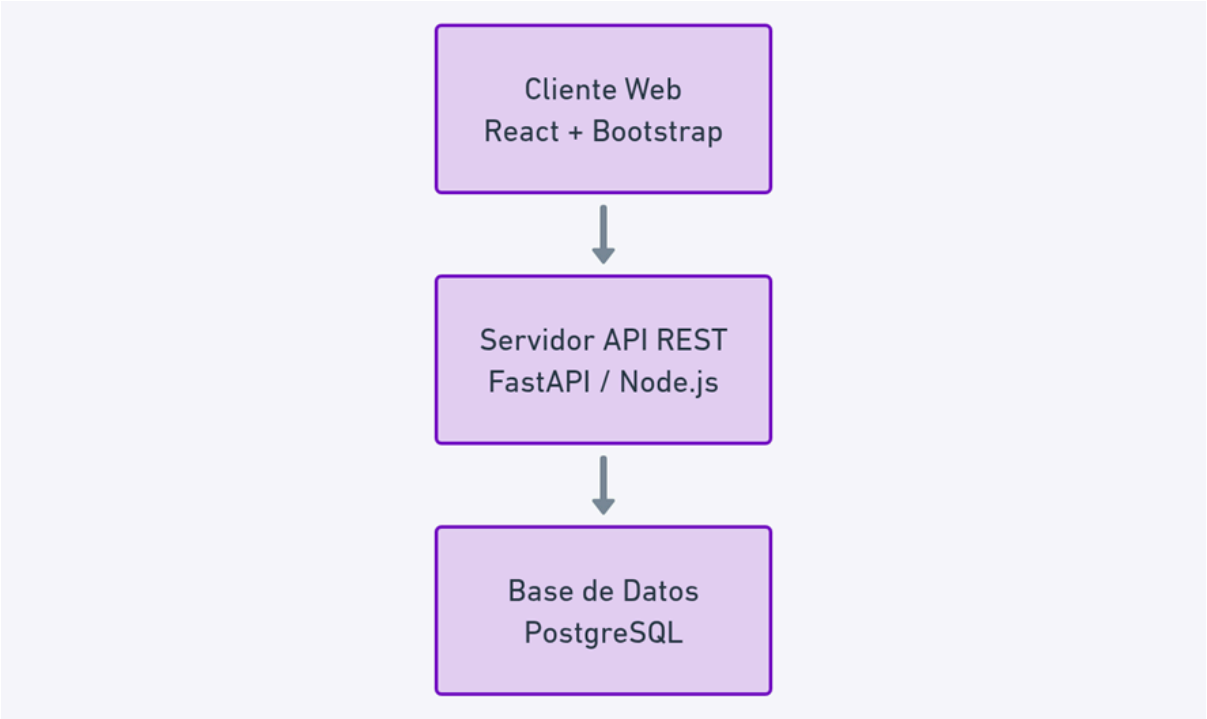
12.1 Diagrama de Capas



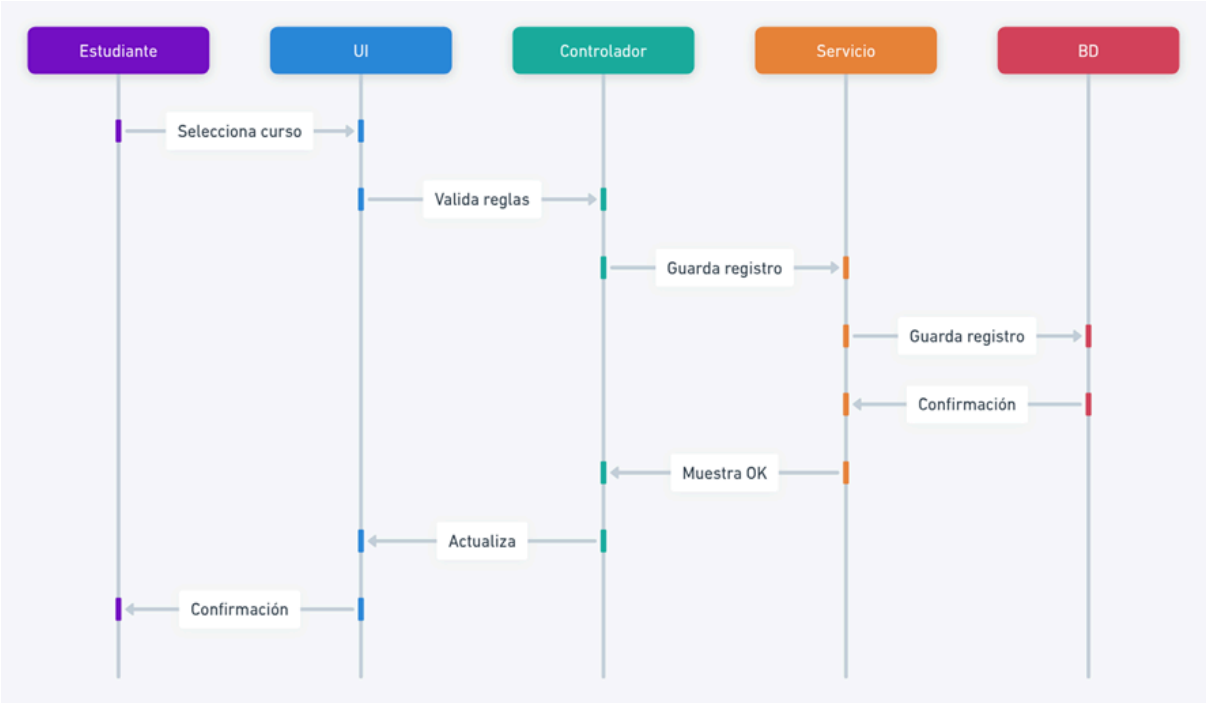
12.2 Diagrama de Componentes



12.3 Diagrama de Despliegue



12.4 Diagrama de Secuencia (Flujo de Matrícula)



13. Justificación técnica

La selección de los estilos y patrones responde directamente a los atributos de calidad establecidos por la norma ISO/IEC 25010 y a las directrices de arquitectura del estándar ISO/IEC/IEEE 42010.

Estándar	**Justificación**
ISO/IEC 42010	Asegura una descripción arquitectónica coherente y documentada.
ISO/IEC 25010	Evalúa los atributos de calidad: rendimiento, mantenibilidad, seguridad.
IEEE 1471	Garantiza trazabilidad entre requerimientos y arquitectura.

---

14. Evaluación de calidad del diseño

Atributo	**Estrategia aplicada**	**Resultado esperado**
Rendimiento	Cache y consultas optimizadas.	Respuesta en < 1s.

Escalabilidad	Microservicios independientes.	Escalado horizontal sin interrupción.
Seguridad	Roles, JWT, HTTPS, OAuth.	Protección de datos sensibles.
Usabilidad	Interfaz moderna y responsiva.	Satisfacción del usuario.
Mantenibilidad	Modularidad + GitFlow.	Reducción de errores en futuras versiones.

15. Beneficios de la propuesta

Estructura clara y modular.

Escalabilidad sin alterar la arquitectura base.

Mayor rendimiento y facilidad de mantenimiento.

Posibilidad de integración con otros sistemas (biblioteca, finanzas).

Cumplimiento con estándares internacionales.

16. Conclusiones

El diseño arquitectónico del **SIGA-UPLA** demuestra cómo la correcta combinación de **estilos (cliente-servidor, en capas, SOA/microservicios) y patrones (MVC, Repository, Facade, Observer, etc.)** genera un sistema con **alta cohesión, bajo acoplamiento y calidad certificable.**

El sistema no solo cumple con los requerimientos actuales de la institución, sino que además está preparado para adaptarse a futuras necesidades mediante una estructura extensible y basada en principios sólidos de ingeniería de software.

17. Referencias bibliográficas

ISO/IEC/IEEE 42010:2011 – Systems and Software Engineering — Architecture Description.

ISO/IEC 25010:2011 – Systems and Software Quality Requirements and Evaluation (SQuaRE).

IEEE Std 1471-2000 – Recommended Practice for Architectural Description of Software-Intensive Systems.

Bass, L., Clements, P., & Kazman, R. (2021). Software Architecture in Practice (4th ed.). Addison-Wesley.

Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.

Shaw, M., & Garlan, D. (1996). Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.

Tema: Modelado y Documentación de la Arquitectura

Actividad 3: Elaboración del documento final de arquitectura

Descripción:

Los equipos consolidan la información de las actividades anteriores en un documento formal de arquitectura de software, siguiendo un estándar reconocido (ejemplo: IEEE 1471/ISO/IEC 42010). El documento debe incluir:

- Introducción y alcance.
- Principios de diseño arquitectónico aplicados.
- Estilos y patrones seleccionados.
- Diagramas arquitectónicos UML (componentes, despliegue, secuencia).
- Justificación técnica fundamentada en estándares internacionales
- Lineamientos para la implementación futura.

Trabajo esperado:

Documento completo de arquitectura con diagramas UML estandarizados y fundamentación técnica, listo para presentarsa como entregable de un proyecto de desarrollo de software.

Evidencia: Documento de diseño arquitectónico con diagramas

Técnica de recolección: e-portafolio.

Instrumento de evaluación: Rúbricas de aprendizaje.

Indicador de evaluación: El estudianto presenta una propuesta arquitectónica coherente, con diagramas claros y fundamentación on estándares intemacionales

1. Introducción

La arquitectura de software constituye el marco estructural sobre el cual se construye, mantiene y evoluciona un sistema informático. Este documento consolida los resultados obtenidos en las actividades previas sobre principios de diseño, estilos arquitectónicos, patrones y diagramas UML, aplicados al Sistema de Gestión Académica UPLA (SIGA-UPLA).

El documento se desarrolla conforme a la norma IEEE 1471 / ISO/IEC 42010, que define la manera formal de describir una arquitectura mediante vistas, stakeholders, decisiones y justificaciones técnicas.

El objetivo principal es documentar de forma completa y estandarizada la arquitectura del SIGA-UPLA, garantizando claridad, coherencia, trazabilidad y mantenibilidad para futuras fases de desarrollo e implementación.

2. Alcance del documento

Este documento describe la **arquitectura del sistema SIGA-UPLA**, considerando aspectos estructurales, funcionales y de calidad.
Incluye los siguientes elementos:

- Principios de diseño arquitectónico aplicados.
- Estilos y patrones seleccionados.
- Diagramas UML: componentes, despliegue y secuencia.
- Justificación técnica basada en estándares internacionales.
- Lineamientos para la implementación y mantenimiento futuros.

No se aborda la codificación ni los detalles de la base de datos, pero sí la organización y relación entre los módulos, capas y servicios del sistema.

3. Descripción general del sistema SIGA-UPLA

El **SIGA-UPLA** es una aplicación web académica que centraliza la gestión educativa de la universidad, automatizando procesos como matrícula, calificaciones, asistencia y generación de reportes.

3.1 Usuarios principales

Estudiantes: registro, matrícula, notas, reportes.

Docentes: carga de calificaciones y asistencia.

Administradores: gestión de usuarios, semestres, cursos.

Soporte técnico: mantenimiento, auditoría y copias de seguridad.

3.2 Objetivos del sistema

1. Centralizar la información académica.
2. Reducir errores administrativos.
3. Mejorar la eficiencia en los procesos.
4. Ofrecer una experiencia moderna, accesible y segura.

3.3 Plataforma tecnológica

- Frontend: React.js / Bootstrap.
- Backend: FastAPI (Python) o Node.js (Express).
- Base de datos: PostgreSQL / Supabase.
- Infraestructura: Vercel (frontend) + Render o Railway (backend).

4. Principios de diseño arquitectónico

Principio	Descripción	Aplicación en SIGA-UPLA
Modularidad	División en módulos independientes con una función específica.	Módulos: autenticación, cursos, matrícula, notas, reportes.
Cohesión	Los componentes de cada módulo están altamente relacionados entre sí.	El módulo de matrícula gestiona solo inscripciones y pagos.
Bajo acoplamiento	Los módulos no dependen directamente unos de otros.	Comunicación por API REST (HTTP/JSON).
Separación de intereses	División por capas: presentación, negocio y datos.	Arquitectura en capas.

Reutilización	Reaprovechamiento de componentes en varios contextos.	Módulo de autenticación usado en todos los roles.
Escalabilidad	Capacidad de adaptarse a mayor carga.	Microservicios por módulo.
Mantenibilidad	Facilidad de corregir errores y actualizar.	Código versionado con GitHub.
Seguridad	Protección de datos e identidades.	Autenticación JWT y control por roles.

5. Estilos arquitectónicos seleccionados

5.1 Cliente-Servidor

Divide el sistema en dos entidades: **cliente (frontend)** y **servidor (backend).**
Permite independencia tecnológica, escalabilidad y mantenimiento sencillo.

Cliente: React.js + Bootstrap (interfaz web).

Servidor: FastAPI (Python) con API REST.

Comunicación: Peticiones HTTP / JSON.

5.2 Arquitectura en capas

Organiza el sistema en niveles jerárquicos que separan responsabilidades.

Capa	**Función**	**Ejemplo**
Presentación	Interfaz visual y experiencia del usuario.	Portal web (React).
Negocio	Lógica académica (reglas de matrícula, cálculo de notas).	Controladores API.

Datos	Persistencia de información.	PostgreSQL / Supabase.
Infraestructura	Conectividad, seguridad, despliegue.	Servidores y balanceadores.

5.3 Orientada a servicios (SOA) / Microservicios

Cada módulo del sistema opera como un servicio independiente, comunicándose mediante interfaces RESTful.

Servicio	**Funcionalidad**	**Lenguaje / Tecnología**
Autenticación	Inicio de sesión, roles y tokens.	FastAPI + JWT
Usuarios	CRUD de estudiantes, docentes, administradores.	Node.js
Cursos	Creación y asignación de asignaturas.	Python
Matrículas	Registro de inscripciones y pagos.	FastAPI
Reportes	Generación de informes y certificados.	Python + Pandas

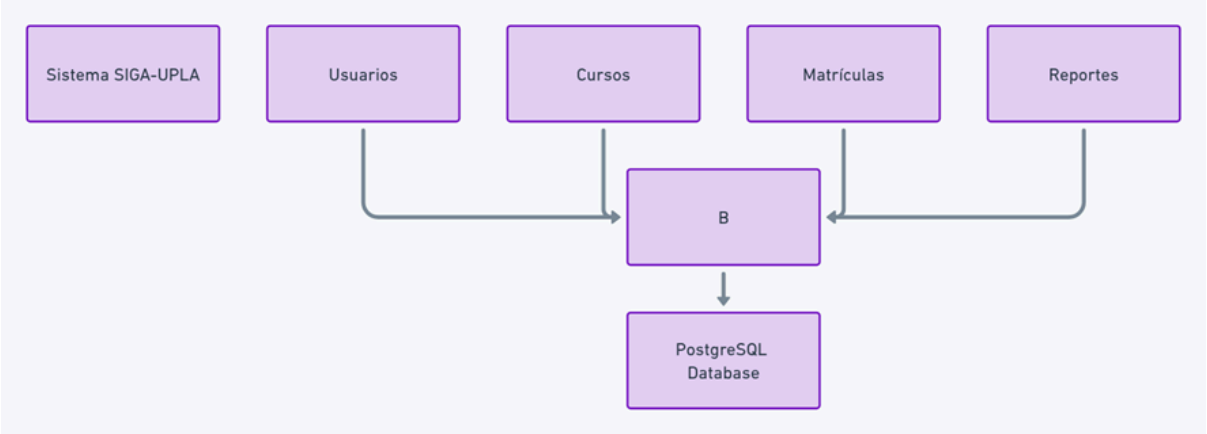
6. Patrones arquitectónicos aplicados

Patrón	**Descripción**	**Aplicación en SIGA-UPLA**
MVC (Modelo-Vista-Controlador)	Separa presentación, control y datos.	Portal web (frontend).

Repository	Abstrae el acceso a datos.	CRUD de usuarios y notas.
Singleton	Controla única instancia global.	Conexión a la BD.
Observer	Permite notificaciones automáticas.	Envío de alertas y notas.
Facade	Simplifica interacciones entre subsistemas.	Dashboard de administración.
Pipe and Filter	Procesa flujos de datos secuenciales.	Generación de reportes.
Broker	Coordina comunicación entre servicios.	Intermediario entre microservicios.

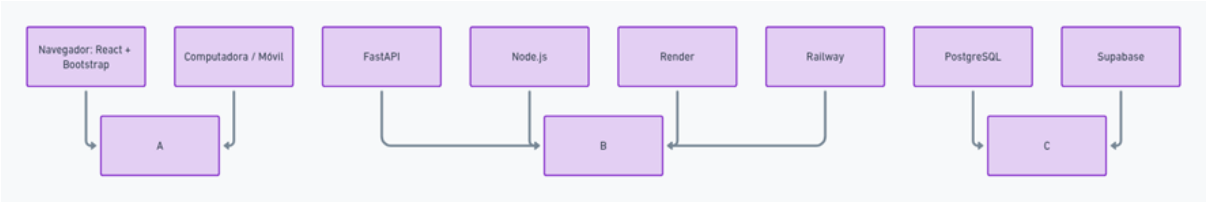
7. Modelado arquitectónico UML

7.1 Diagrama de Componentes



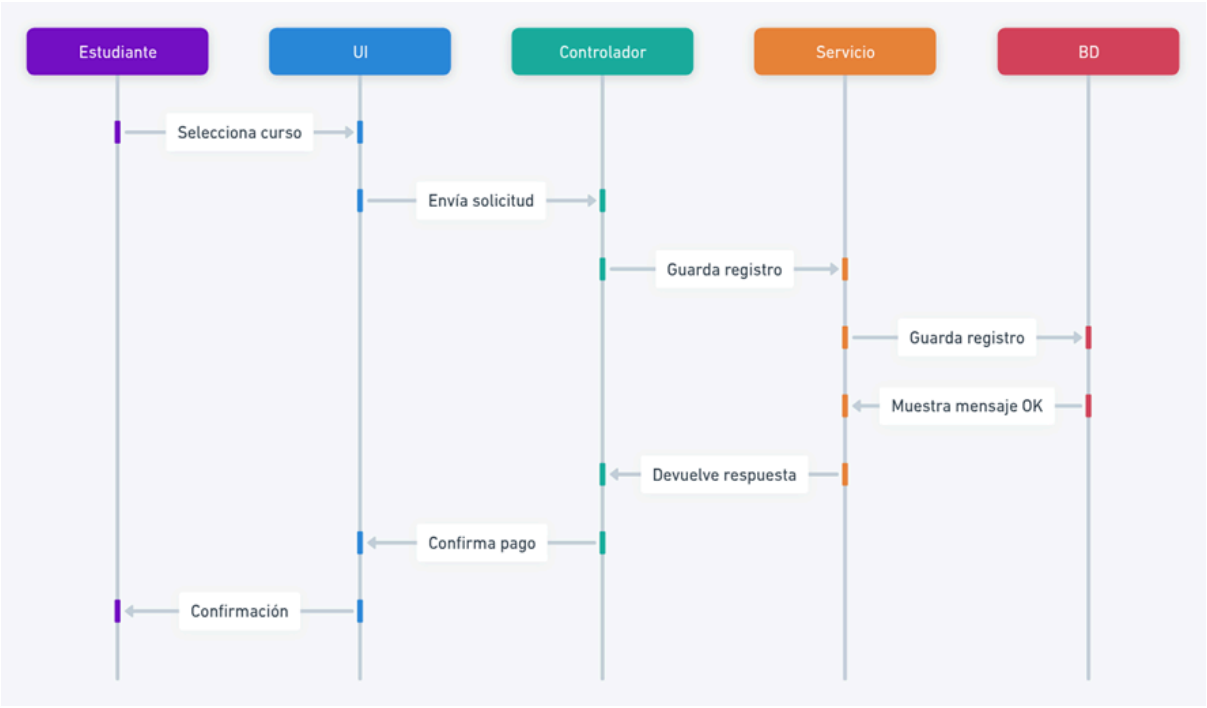
Interpretación:
Cada componente representa un módulo con interfaz propia.
Todos se comunican mediante API REST con intercambio de datos en formato JSON.

7.2 Diagrama de Despliegue



Interpretación:
El cliente accede a la API a través de Internet; el servidor gestiona la lógica y la base de datos almacena la información académica.

7.3 Diagrama de Secuencia (Flujo de Matrícula)



Interpretación:
El flujo inicia con la acción del usuario, pasa por la capa de presentación, luego al backend donde se validan las reglas, y finalmente se actualiza la base de datos.

8. Justificación técnica

8.1 Basada en ISO/IEC/IEEE 42010

- Define cómo documentar una arquitectura, incluyendo stakeholders, vistas y decisiones.
- La arquitectura del SIGA-UPLA sigue este estándar al describir vistas lógicas, de desarrollo y de implementación.

8.2 Basada en ISO/IEC 25010

- Se priorizan los atributos de **calidad**:
- **Mantenibilidad** → modularidad + documentación.
- **Escalabilidad** → microservicios.
- **Seguridad** → JWT y OAuth2.
- **Compatibilidad** → APIs REST.
- **Rendimiento** → consultas optimizadas y cache.

8.3 Basada en IEEE 1471

- Se asegura que cada decisión arquitectónica responda a un requerimiento funcional o no funcional.
- Se mantienen relaciones entre vistas y stakeholders.

9. Lineamientos para la implementación futura

1. **Metodología**: emplear un enfoque ágil (Scrum) con entregas iterativas.
2. **Control de versiones**: GitHub + ramas por módulo (GitFlow).
3. **Pruebas**: Unitarias con PyTest o Jest; integrales con Postman.
4. **Despliegue** continuo: integrar CI/CD en Vercel y Render.
5. **Seguridad**: aplicar HTTPS, roles por JWT y auditorías de logs.
6. **Documentación técnica**: generar API Docs con Swagger/OpenAPI.
7. **Escalabilidad**: contenedores Docker para cada servicio.
8. **Monitoreo**: Prometheus + Grafana.

9. **Backup**: programar copias automáticas de la base de datos Supabase.
10. **Métricas de calidad**: revisar indicadores ISO/IEC 25010 cada semestre.

10. Evaluación de atributos de calidad

Atributo (ISO 25010)	Implementación	Resultado esperado
Rendimiento	Cache en backend + consultas optimizadas.	Tiempo de respuesta < 1 s.
Escalabilidad	Microservicios y balanceo de carga.	Soporta +1000 usuarios concurrentes.
Mantenibilidad	Código modular, documentado.	Cambios sin impacto global.
Seguridad	JWT, HTTPS, cifrado SHA256.	Cumplimiento con normas de protección de datos.
Usabilidad	UI moderna y responsiva.	Alta satisfacción de usuarios.
Portabilidad	Despliegue en nube y local.	Flexibilidad de migración.
Fiabilidad	Backup diario, logs y alertas.	Reducción de fallos a < 2 %.

11. Riesgos y mitigaciones

Riesgo	Descripción	Mitigación
Fallo de servidor	Sobrecarga o caída de conexión.	Balanceadores + backups.

Vulnerabilidades	Ataques de inyección o fuga de datos.	Validación, encriptación y auditorías.
Pérdida de datos	Eliminación accidental o error técnico.	Copias de seguridad automáticas.
Baja escalabilidad	Crecimiento de usuarios sin preparación.	Arquitectura en microservicios.
Falta de mantenimiento	Código sin documentación.	Estándares de desarrollo y repositorio Git.

12. Beneficios de la propuesta

- Estructura modular, segura y documentada.
- Facilidad de mantenimiento y evolución.
- Cumplimiento de normas ISO/IEEE.
- Escalabilidad horizontal mediante microservicios.
- Integración futura con sistemas externos (ERP, biblioteca virtual).
- Interfaz amigable y multiplataforma.

13. Conclusiones

La arquitectura del **SIGA-UPLA** cumple con los principios fundamentales del diseño arquitectónico de software moderno, combinando estilos **cliente-servidor, en capas y microservicios.**
Los patrones aplicados (MVC, Repository, Observer, Facade) permiten lograr **bajo acoplamiento, alta cohesión y reutilización**, mejorando la mantenibilidad del sistema.

El modelado mediante diagramas UML proporciona una visión clara de los componentes, su despliegue y la secuencia de interacción entre ellos, lo que garantiza una base sólida para la fase de implementación.

La fundamentación bajo las normas **ISO/IEC 42010, 25010 y IEEE 1471** respalda la coherencia técnica y la calidad del diseño, asegurando que la propuesta sea viable, escalable y sostenible en el tiempo.

14. Referencias bibliográficas

ISO/IEC/IEEE 42010:2011 — Systems and Software Engineering – Architecture Description.

ISO/IEC 25010:2011 — Systems and Software Quality Requirements and Evaluation (SQuaRE).

IEEE 1471-2000 — Recommended Practice for Architectural Description of Software-Intensive Systems.

Bass, L., Clements, P., & Kazman, R. (2021). Software Architecture in Practice (4th ed.). Addison-Wesley.

Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.

Shaw, M., & Garlan, D. (1996). Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.

Pressman, R. (2020). Ingeniería del Software: Un enfoque práctico. McGraw-Hill.