Integrantes:

Carlos G. Orellana Chumbay

Kevin A. Soriano Dominguez

Jose M. Arias Zavala

Paul R. Torres Rueda

Materia:

Diseño de Software

Docente:

Vanessa Alexandra Jurado Vite

Fecha:

21 de abril del 2025

Equipo 3 – Plataforma de organización de eventos universitarios: Permite a organizaciones estudiantiles publicar eventos, gestionar inscripciones, emitir certificados, y controlar aforos.

Contenido

RECURSOS

- REPOSITORIO

  https://github.com/Poolfx/PlataformaEventosUniversitarios.git
- PLANIFICACIÓN DIAGRAMA DE GANNT (REPOSITORIO, CARPETA DOCUMENTACIÓN)
- COEVALUACIÓN (REPOSITORIO, CARPETA DOCUMENTACIÓN)
- LINK DEL VIDEO (REPOSITORIO, CARPETA VIDEO)
- LINK DEL PROTOTIPO (FIGMA)
  - Es necesario iniciar sesión con la cuenta de FIGMA
  - https://www.figma.com/proto/QDKNvK1Osu2ROajVBV41E7/Sales-Dashboard-Design--Community-?page-id=8121%3A2&node-id=8222-15&p=f&viewport=2468%2C1684%2C0.41&t=ggtS97E8jG7Eb6Nb-1&scaling=min-zoom&content-scaling=fixed&starting-point-node-id=8222%3A15

FASE 1: PLANIFICACIÓN Y COORDINACIÓN

Definición de roles y responsabilidades:

Carlos Orellana:

- Participar en la definición de la experiencia de usuario (UX) junto a Jose.
- Proponer y establecer el stack tecnológico del FrontEnd (por ejemplo: React, Vue, Angular, Next).
- Coordinar la integración del FrontEnd con el BackEnd (trabajo conjunto con Paul).
- Establecer convenciones de código y estructura para el FrontEnd.
- Asegurar la compatibilidad y responsividad de la interfaz en distintos dispositivos.

Entregables clave:

- Arquitectura inicial del FrontEnd.
- Revisión de requisitos relacionados con UI/UX.

Jose Arias

- Reunirse con stakeholders o usuarios para recopilar requisitos funcionales y no funcionales.
- Documentar los requerimientos con claridad (puede usar casos de uso, historias de usuario o especificaciones formales).
- Crear diagramas de flujo y mapas de navegación del sistema.
- Alinear las necesidades del cliente con las decisiones técnicas del equipo.
- Validar los requerimientos con el equipo antes de iniciar la implementación.

Entregables clave:

- Documento de requerimientos del sistema.
- Historias de usuario o casos de uso.

Paul Torres

- Participar en la definición de la arquitectura del sistema (junto con Kevin y Carlos).
- Seleccionar el stack tecnológico del BackEnd (por ejemplo: Node.js, Django, C#).
- Establecer la estructura del proyecto y los principios de desarrollo del servidor.
- Diseñar la lógica de negocio y definir los endpoints de la API.
- Coordinar con Carlos para la integración de Front y Back.
- Asegurar la escalabilidad y seguridad de la lógica del sistema.

Entregables clave:

- Diseño de arquitectura del BackEnd.
- Documentación inicial de endpoints/API.
- Propuesta de arquitectura de seguridad y autenticación.

Kevin Soriano

- Diseñar el modelo entidad-relación (ER) del sistema.
- Crear los diagramas UML: casos de uso, clases, secuencia.
- Definir la estructura de la base de datos y los tipos de relaciones.
- Apoyar al equipo en consultas complejas y estructuras de datos.
- Participar en la validación de integridad entre BackEnd y base de datos.

Entregables clave:

- Modelo de datos completo (MER).
- Diagramas UML relevantes.
- Especificación de la base de datos (DDL).

Cronograma de trabajo:

Diagrama de GANTT enlace público:

https://docs.google.com/spreadsheets/d/1t99snT3ql2mLoAyUqEy31FdTNfaH5qN5RsI_il9jvJc/edit?usp=sharing

PROYECTO UEES NOW

TÍTULO DEL PROYECTO	Plataforma de organización de eventos universitarios	NOMBRE DE LA EMPRESA: Universidad Espíritu Santo - Grupo 3	
RESPONSABLE DEL PROYECTO	GRUPO 3	FECHA: 17/04/2025	

FASE	TÍTULO DE LA TAREA	RESPONSABLE DE LA TAREA	FECHA DE INICIO	FECHA DE ENTREGA	DURACIÓN	% COMPLETADO DE LA TAREA	FASE UNO					FASE DOS															
							SEMANA 1					SEMANA 4					SEMANA 5					SEMANA 6						
							L	M	X	J	V	L	M	X	J	V	L	M	X	J	V	L	M	X	J	V		
1	**Planificación y coordinación**																										
1.1	Reunión de conocimiento del equipo	Carlos Orellana	15/04/25	15/04/25	0	100 %																					
1.1.1	Definición de roles y responsabilidades.	Carlos Orellana	15/04/25	15/04/25	0	100 %																					
1.2	Planificación de ramas y flujo de trabajo colaborativo en GitHub	Carlos Orellana	17/04/25	17/04/25	0	90 %																					

BITACORAS

Reunion 1

Fecha: 15 de abril de 2025

Participantes: Equipo de desarrollo (Carlos Orellana, Kevin Soriano, José Arias, Paul Torres)

Decisión: Se establecieron roles específicos dentro del equipo de desarrollo para mejorar la eficiencia y organización, (todo esto descrito en el punto 1 de la primera Fase)

Reunion 2

Fecha: 17 de abril de 2025

Participantes: Equipo de desarrollo

Decisión: Definición de arquitectura general y revisión de documentación

Descripción de la decisión:

- Definición de arquitectura del sistema
- Búsqueda de infraestructura para el despliegue del sistema (Vercel - Ngrok)
- Organización del GitHub y ramas (main - developer)

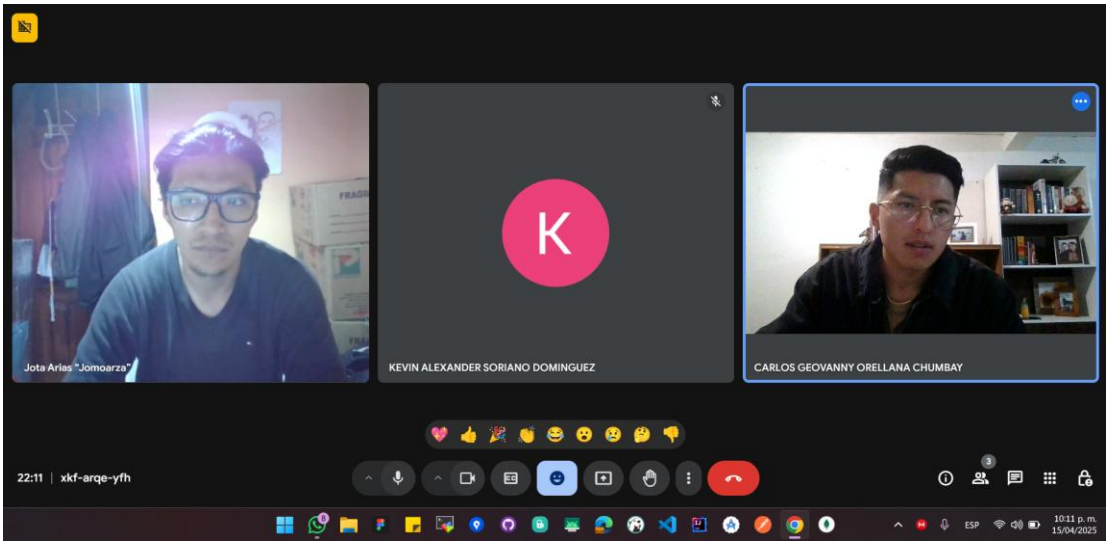
Evidencia de reuniones y acuerdos:

Sesión #1: Conocimiento del equipo de trabajo y designación de roles y responsabilidades

Fecha: 15/04/2025

Acuerdos:

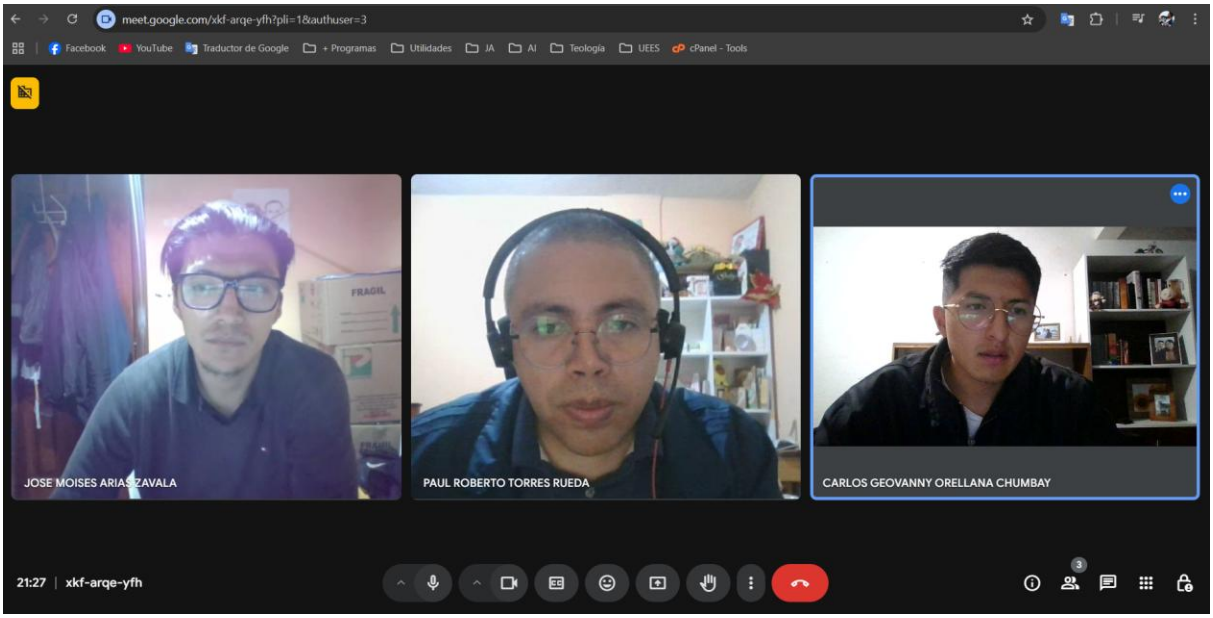
- Tecnología a usar en el FrontEnd: Next
- Tecnología a usar en el BackEnd: C#
- Motor de base de datos: PostgreSQL
- Designación de responsable del FrontEnd: Carlos Orellana
- Designación de responsable del BackEnd: Paul Torres
- Designación de responsable de levantamiento de requisitos funcionales y no funcionales: Jose Arias
- Designación de responsable de Diagramación y modelado ER: Kevin Soriano.



Sesión #2: Fecha: 17/04/2025

Acuerdos:

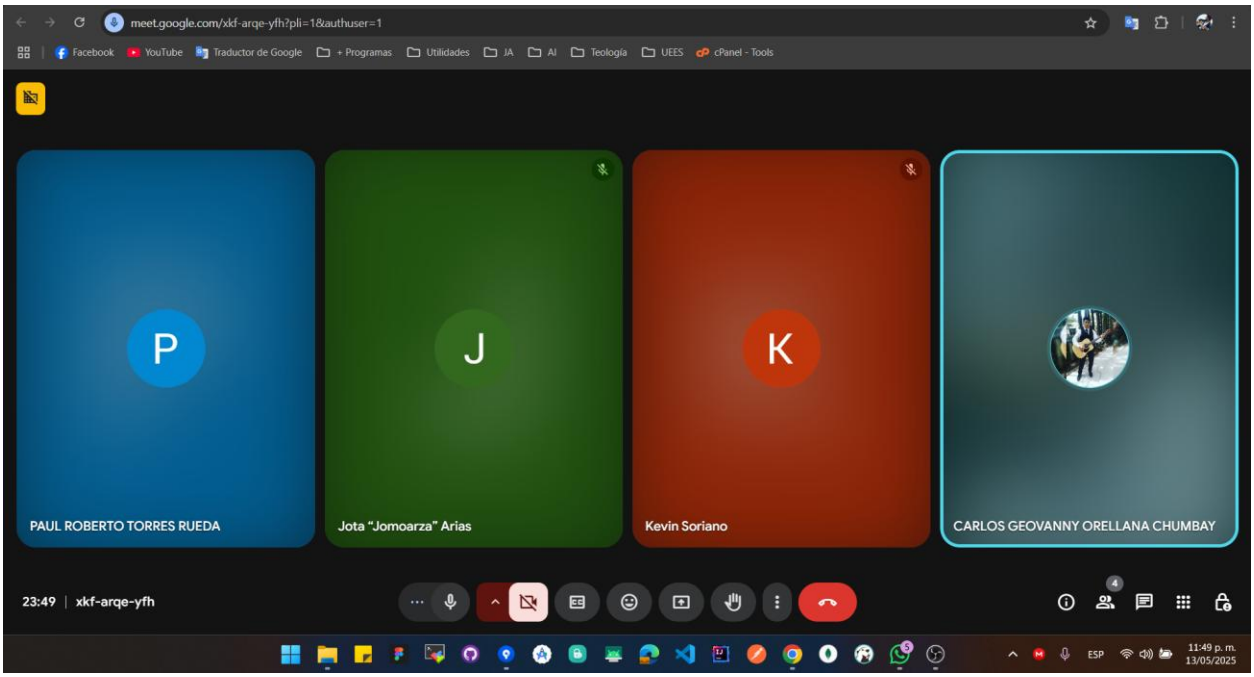
- Definición de arquitectura del sistema: Arquitectura basada en capas
- Busqueda de infraestructura para el despliegue del sistema (Vercel - Ngrok)
- Organización del GitHub y ramas (main - developer)



Sesión #3: Fecha: 13/05/2025

Acuerdos:

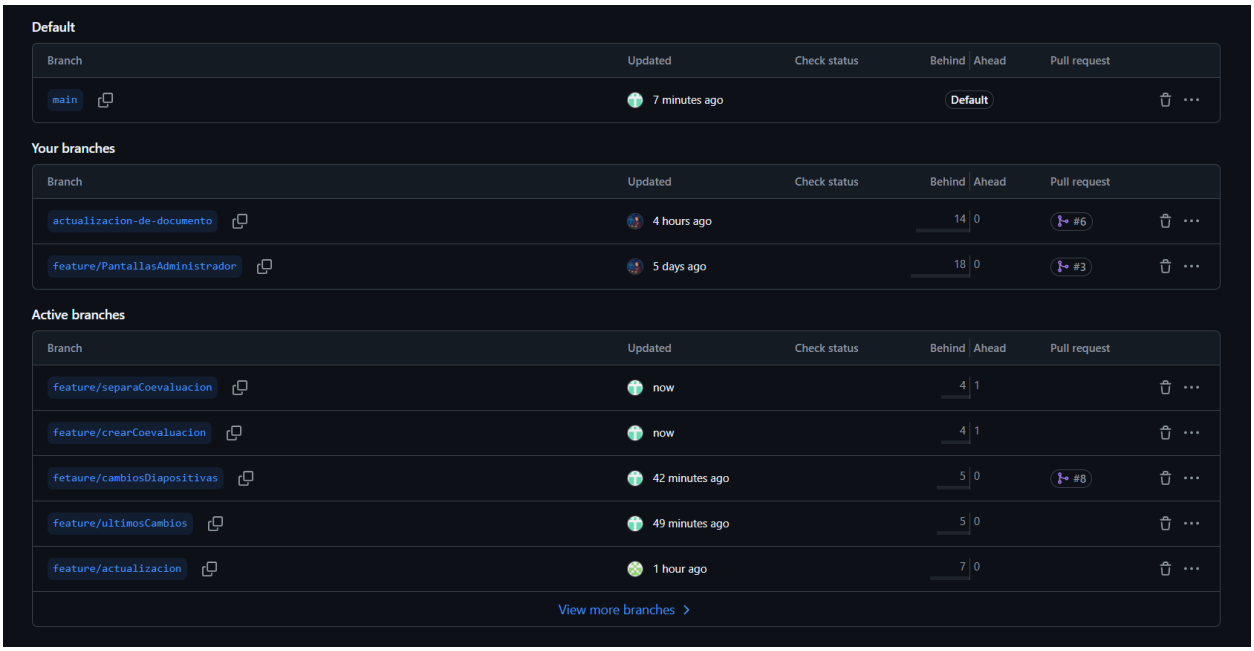
- Revisión de repositorio
- Avances en el prototipo



Planificación de ramas y flujo de trabajo colaborativo en GIT

Para garantizar un flujo de trabajo ordenado, colaborativo y fácilmente mantenible, se ha definido una estrategia de ramificación basada en una variante simplificada del modelo Git Flow.

Esta estrategia permite al equipo de desarrollo organizar el trabajo en paralelo, mantener estabilidad en las versiones liberadas y facilitar la integración continua.



Ramas principales

- **main**: contiene el código en estado estable y listo para producción. Solo se actualiza mediante ramas de release o hotfix.
- **develop:** actúa como entorno de integración para las funcionalidades en desarrollo. Es la rama base para crear nuevas funcionalidades (feature/*) o correcciones (fix/*).

Ramas secundarias

- **feature/*:** se utilizan para desarrollar nuevas funcionalidades. Se crean a partir de develop y se integran nuevamente allí una vez completadas y probadas.
- **fix/*:** se destinan a resolver bugs no críticos encontrados durante el desarrollo. También se crean desde develop.

Convenciones

- Los nombres de las ramas utilizan prefijos (feature/, fix/) y guiones para separar palabras.
- Los mensajes de commit siguen el formato Conventional Commits para facilitar el análisis de cambios:

- ***feat:*** nuevas funcionalidades
- ***fix:*** correcciones de errores

FASE 2: ANALISI DEL PROBLEMA Y REQUERIMIENTOS

Las organizaciones estudiantiles dentro de las universidades realizan eventos académicos, culturales y recreativos. Sin embargo, muchas veces estas actividades se gestionan de forma manual o con herramientas dispersas (formularios, correos, hojas de cálculo), lo que ocasiona desorganización y pérdida de información.

Esto genera problemas como:

- Falta de una plataforma centralizada para publicar eventos.
- Dificultad para controlar inscripciones y aforos.
- Procesos manuales o informales para registrar asistencia.
- Emisión de certificados lenta o inexistente.
- Baja difusión de eventos entre los estudiantes.

Se requiere una plataforma web centralizada que facilite:

- La publicación de eventos por parte de organizaciones estudiantiles.
- La inscripción por parte de los estudiantes.
- El control automático del aforo.
- El registro de asistencia.
- La emisión de certificados digitales.

Descripción general del sistema

La Plataforma de Organización de Eventos Universitarios es una aplicación web diseñada para facilitar la gestión de eventos académicos, culturales y estudiantiles dentro del entorno

universitario. El sistema permite a organizaciones estudiantiles y administradores crear, editar y difundir eventos, mientras que los estudiantes pueden registrarse, recibir confirmaciones y certificados de asistencia.

El objetivo principal es digitalizar y automatizar el proceso de inscripción y certificación de eventos, mejorar el control de aforo y proporcionar métricas claras sobre la participación estudiantil. La plataforma está compuesta por un frontend web, un backend basado en ASP.NET Core y una base de datos relacional PostgreSQL.

Identificación de Actores y Funcionalidades

1. Estudiante (Usuario Registrado)

- Registrarse en la plataforma
- Iniciar sesión y consultar eventos disponibles
- Inscribirse a eventos (si hay cupo disponible)
- Visualizar eventos inscritos
- Descargar certificados de eventos asistidos

2. Organizador de Eventos / Usuario Administrador

- Crear, editar o eliminar eventos
- Definir aforo máximo, lugar, horario y descripción del evento
- Ver lista de inscritos
- Enviar correos de confirmación o recordatorio

3. Administrador General del Sistema

- Gestionar usuarios y roles

- Auditar inscripciones y generación de certificados
- Acceder a reportes y estadísticas (dashboard)
- Configurar parámetros generales del sistema (opcional)

Funcionalidades Clave del Sistema

Funcionalidad	Actor
Registro e inicio de sesión	Estudiante, Organizador
Gestión de eventos (CRUD)	Organizador, Administrador
Inscripción a eventos	Estudiante
Control de aforo	Sistema / Organizador
Generación de certificados (PDF)	Organizador
Envío de correos automáticos	Sistema
Dashboard de métricas	Administrador

Requisitos funcionales y no funcionales:

Requisitos funcionales:

- **Gestión de eventos**
    - Creación, edición y eliminación de eventos por parte de organizaciones estudiantiles.
    - Definición de categorías de eventos (conferencias, talleres, competiciones, etc.).
    - Establecimiento de fechas, ubicaciones y capacidad máxima.

- **Inscripción y control de aforo**
    - Registro de participantes con validación de datos.
    - Generación de listas de asistentes.
- **Emisión de certificados**
    - Generación de certificados de participación.
- **Gestión de usuarios**
    - Diferenciación de roles (organizadores, participantes, administradores).
    - Registro e inicio de sesión con autenticación segura.
    - Administración de permisos según el perfil de usuario.

Requisitos no funcionales:

- **Seguridad y protección de datos**
    - Cifrado de credenciales y datos personales.
    - Protección contra accesos no autorizados (token).
    - Políticas de privacidad y cumplimiento de normativas de protección de datos.
- **Usabilidad y accesibilidad**
    - Interfaz intuitiva para facilitar la navegación.
    - Adaptabilidad a diferentes dispositivos (responsive design).
- **Escalabilidad y rendimiento**
    - Capacidad para manejar un alto volumen de usuarios concurrentes.
    - Optimización de consultas a la base de datos para tiempos de respuesta rápidos.
- **Disponibilidad y fiabilidad**
    - Registro de actividad para auditorías y solución de problemas (logs).
- **Compatibilidad e integración**

- Posibilidad de conectarse con otras plataformas universitarias.

FASE 3: DISEÑO ARQUITECTÓNICO

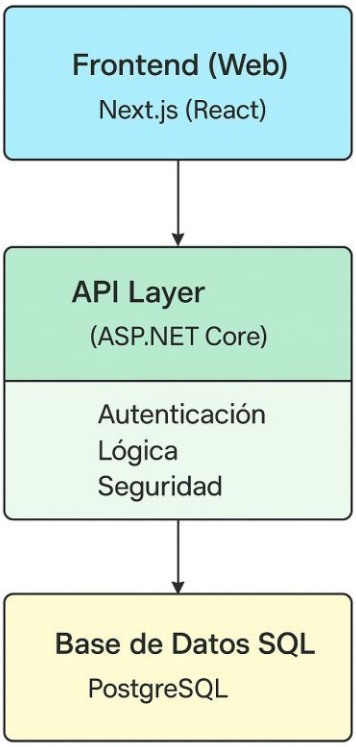
Estilo Arquitectonico:

Tipo de arquitectura: Arquitectura en Capas (Layered Architecture)

Justificacion:

En este proyecto vamos a utilizar una arquitectura en capas (layered architecture), también conocida como arquitectura n-tier (multicapa). Este enfoque es uno de los más comunes y adecuados para aplicaciones web de propósito general, especialmente en entornos educativos o corporativos donde la claridad, mantenibilidad y separación de responsabilidades son prioritarias.

Diagrama de propuesta:



Capas principales en tu solución:

1. **Presentación (Frontend)**
    - Responsabilidad: Interfaz de usuario y experiencia del cliente.
    - Tecnología: Next.js (React)
    - Interacción: Realiza llamadas HTTP al backend mediante fetch/axios.
2. **Capa de Aplicación / API (Controladores)**
    - Responsabilidad: Exponer endpoints REST, recibir y responder solicitudes.
    - Tecnología: ASP.NET Core Web API – Controladores (Controllers/)
    - Interacción: Orquesta servicios y lógica de negocio, recibe DTOs.
3. **Lógica de Negocio (Servicios)**

- Responsabilidad: Procesar reglas de negocio, validaciones, flujo de datos.
- Tecnología: C# – Clases de servicio (Services/)
- Interacción: Se comunica con la capa de datos y responde a la API.

4. **Capa de Datos (Persistencia)**
   - Responsabilidad: Acceso y manipulación de la base de datos.
   - Tecnología: Entity Framework Core (Data/, ApplicationDbContext.cs)
   - Base de datos: PostgreSQL (según diagrama final)

FASE 4: DISEÑO DETALLADO

Diagrama de secuencia de inscripción

Representación del flujo de interacción entre objetos en un escenario específico

"Registro de usuario en un evento":

1. El usuario inicia sesión.
2. Selecciona un evento.
3. Se envía la solicitud de inscripción al sistema.
4. El sistema valida la disponibilidad.
5. Se confirma la inscripción y se genera el registro.

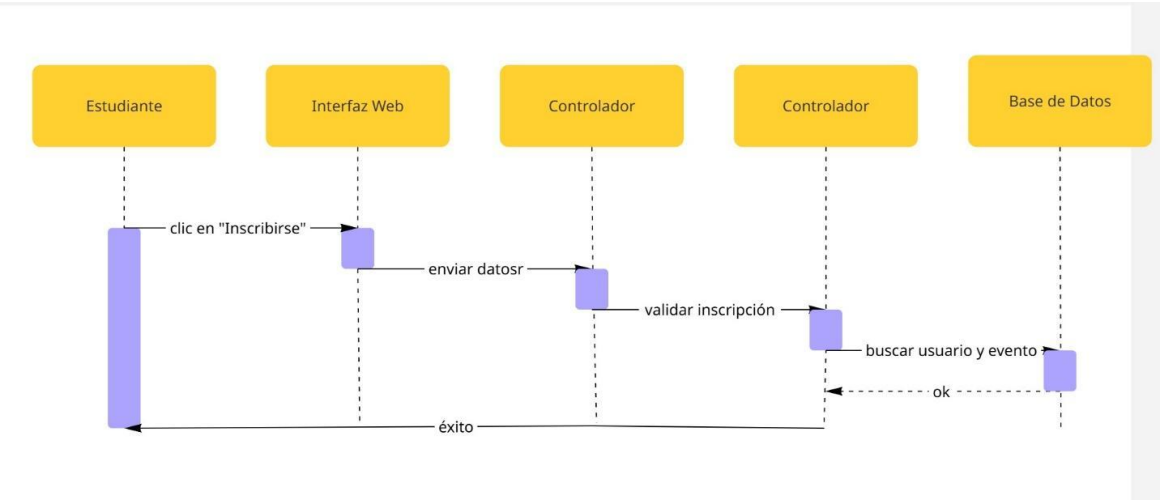
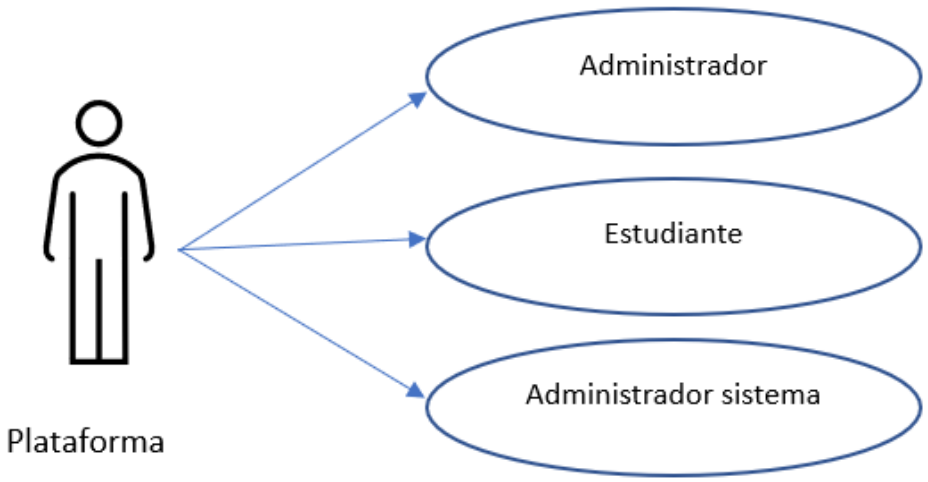

Estudiante
Interfaz Web
Controlador
Controlador
Base de Datos
clic en "Inscribirse"
enviar datosr
validar inscripción
buscar usuario y evento
ok
éxito

Diagrama de casos de uso

Descripción de cómo los usuarios interactúan con un sistema para

- Diagrama de caso de uso de la plataforma:



- Diagrama de caso de uso de la Administrador:

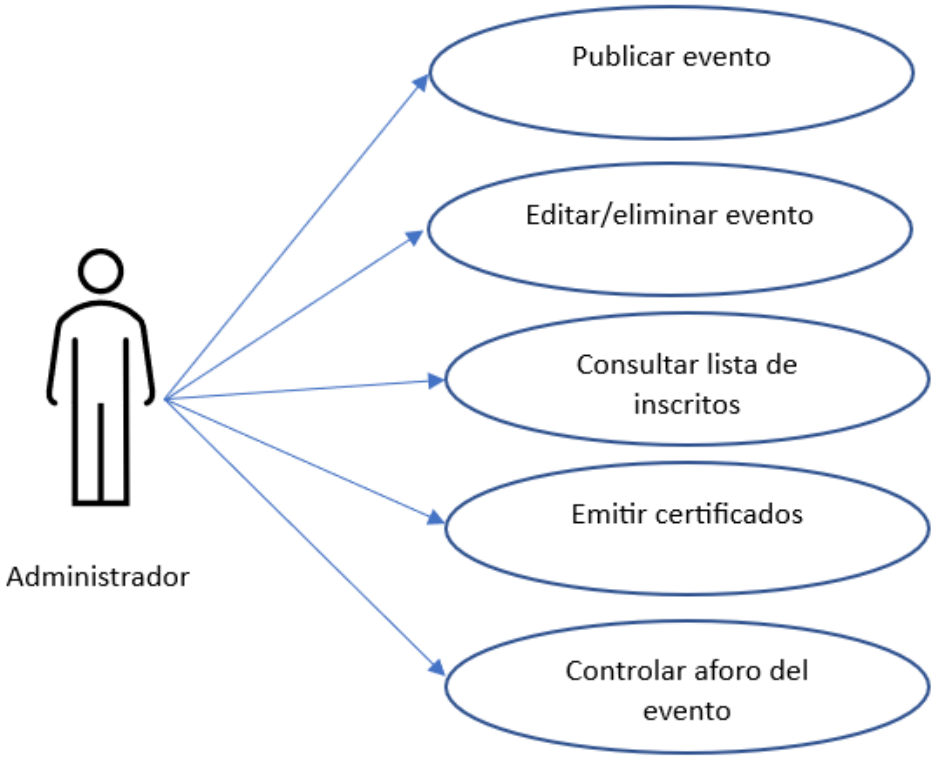


Diagrama de caso de uso de la Estudiante:

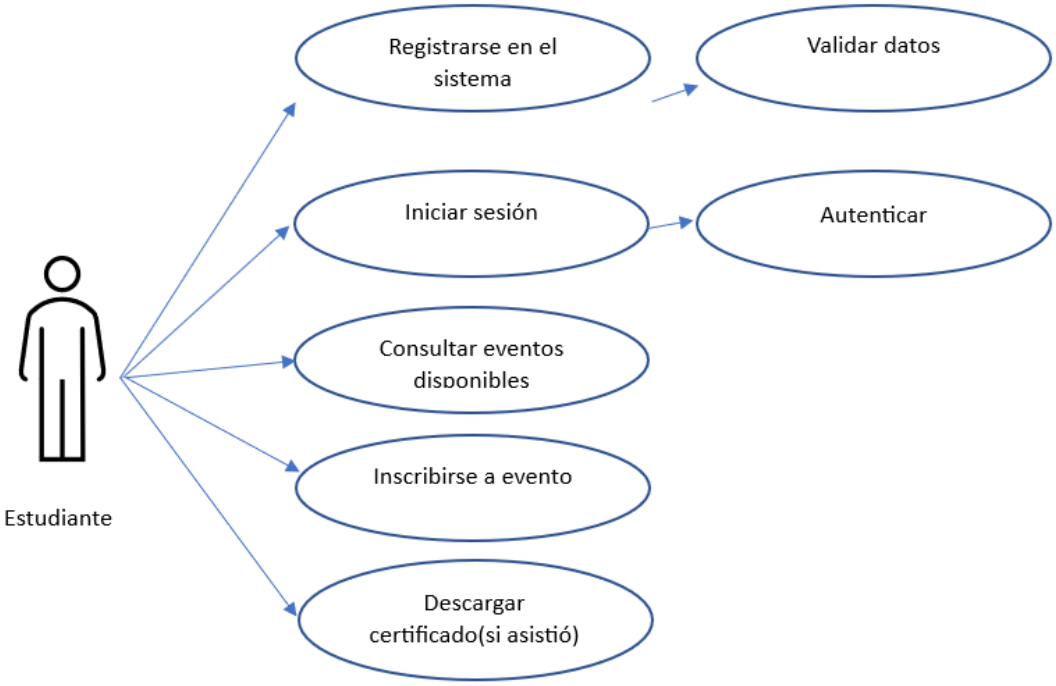


Diagrama de caso de uso de la Administrador del Sistema:

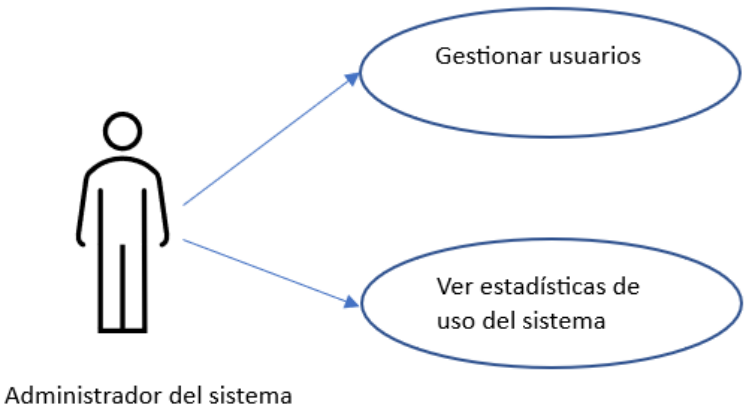
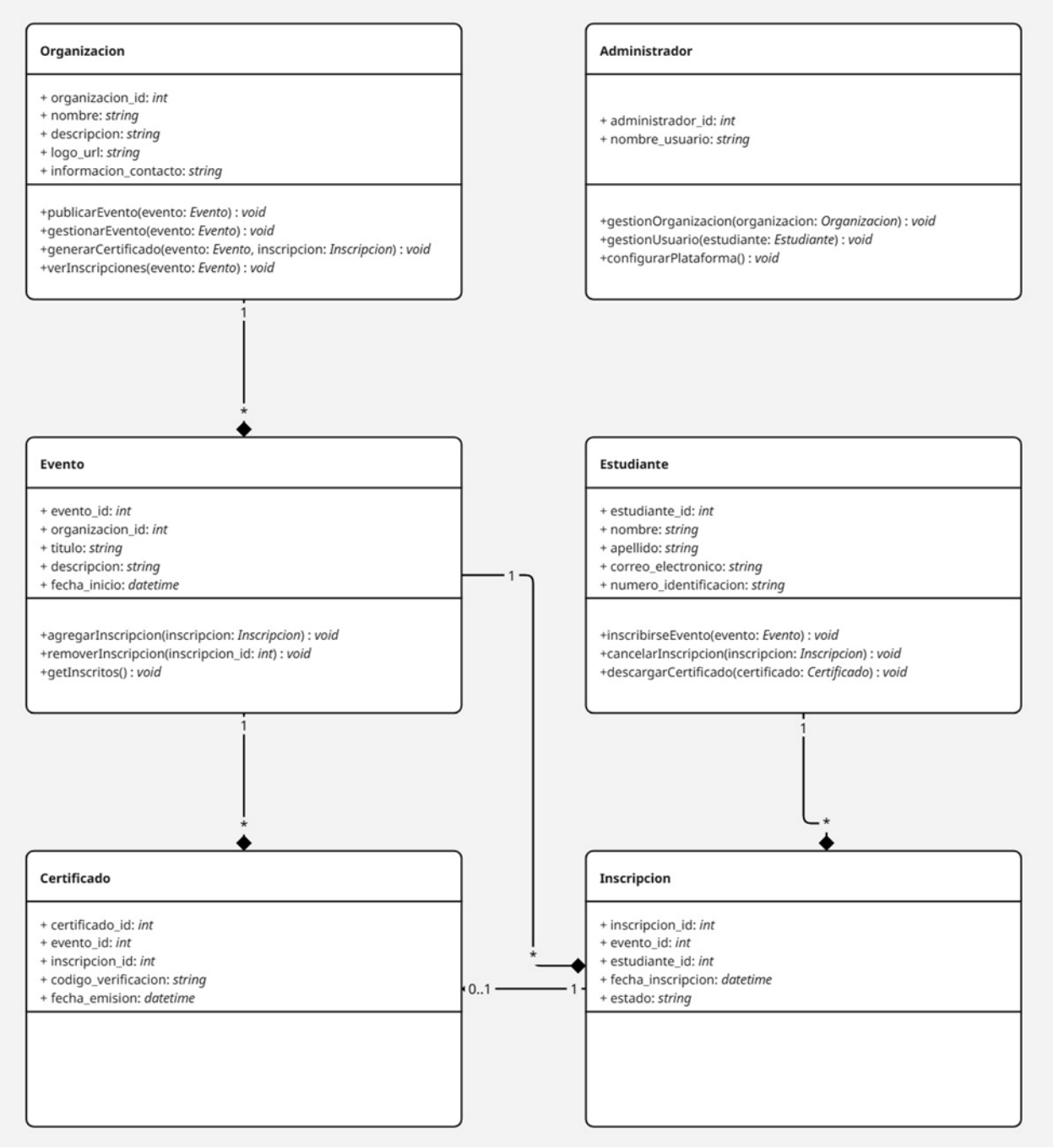
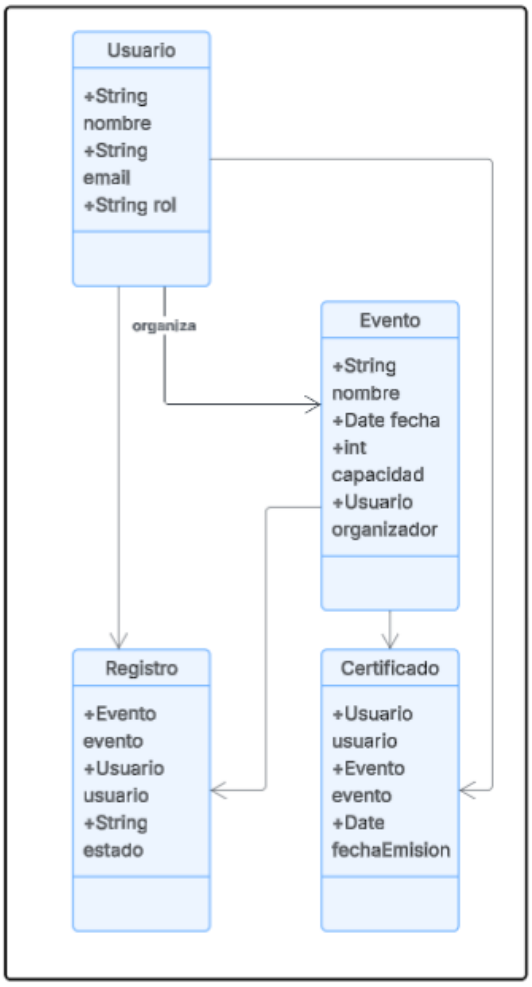


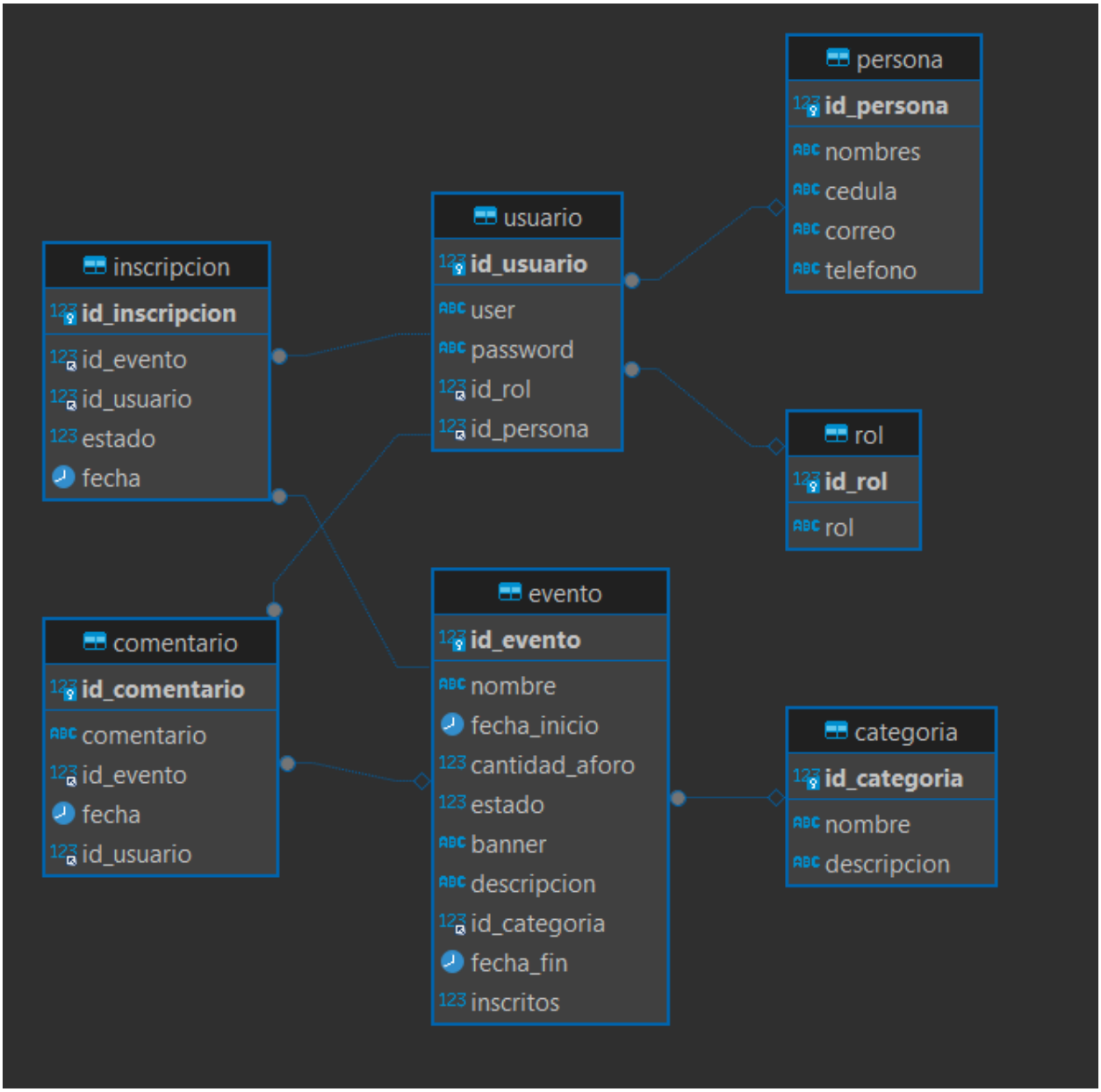
Diagrama de clase

Representar la estructura estática del sistema mostrando las clases, atributos, métodos y relaciones entre ellas.




Usuario
+String nombre
+String email
+String rol
organiza
Evento
+String nombre
+Date fecha
+int capacidad
+Usuario organizador
Registro
+Evento evento
+Usuario usuario
+String estado
Certificado
+Usuario usuario
+Evento evento
+Date fechaEmision

Modelo de datos ER



JUSTIFICACIÓN DE PATRONES DE DISEÑO USADOS

Arquitectura por capas

Se optó por una arquitectura basada en capas (Modelado en la FASE 3 punto 1) porque permite dividir el sistema en niveles bien definidos, donde cada capa tiene una responsabilidad específica. Esta separación facilita el desarrollo, mantenimiento y escalabilidad del software, ya que los cambios en una capa no afectan directamente a las demás. Por ejemplo, la capa de presentación se encarga de la interfaz del usuario, la capa lógica gestiona los procesos del sistema, y la capa de datos se ocupa del acceso y manejo de la información. Esta organización mejora la claridad del código, permite una mejor distribución del trabajo y facilita futuras ampliaciones del sistema.

Singleton (Gestión de conexión a base de datos)

Se utilizó el patrón Singleton para controlar la conexión con la base de datos, ya que garantiza que solo exista una instancia activa durante la ejecución del sistema. Esta decisión se tomó para evitar múltiples conexiones innecesarias que podrían afectar el rendimiento y la estabilidad del sistema. Al centralizar esta gestión, se reduce el consumo de recursos y se mejora el control sobre las operaciones que acceden a la base de datos, logrando una mayor eficiencia.

Factory Method (Creación de objetos dinámicos)

El patrón Factory Method fue utilizado para facilitar la creación de objetos según las necesidades del sistema, sin tener que definir clases específicas para cada tipo. Esto resulta útil cuando se generan diferentes tipos de elementos, como eventos, certificados o registros,

dependiendo de las acciones que realiza el usuario. Con este patrón, el sistema puede adaptarse de forma más flexible y mantener un código más limpio y ordenado.

Observer (Notificaciones a usuarios)

Se implementó el patrón Observer para permitir que los usuarios reciban notificaciones automáticas cuando se producen ciertos cambios en el sistema, como actualizaciones en el estado de un evento. Esta solución mejora la experiencia del usuario al mantenerlo informado en tiempo real sin necesidad de acciones manuales por parte del administrador. Así, el sistema puede avisar automáticamente cuando se abren o cierran inscripciones, o cuando se emiten certificados, de forma eficiente y ordenada.

Conclusión

El desarrollo del prototipo del sistema de gestión de eventos UEES NOW, fue una experiencia valiosa para aplicar los conceptos fundamentales del diseño de software como patrones de diseño, arquitecturas entre otras. A lo largo del proyecto se logró estructurar una solución funcional que permite organizar, planificar y controlar eventos universitarios de forma ordenada. Con este prototipo realizado queremos recalcar lo que consideramos lo más importante para empezar una solución de software, que es, buena planificación y enfoque en los requerimientos del usuario como lo más importante, debido a que de ello se desprende todo el desarrollo, sin una claridad en los requerimientos del usuario todo el sistema puede ser construido erróneamente.

Recomendaciones

Para llevar este proyecto más allá del prototipo, sería útil empezar a probarlo en una pequeña parte de la universidad, como una facultad en específico, y realizar un evento de prueba. Esto ayudará a

ver cómo se adapta a las necesidades reales y a identificar mejoras. Además, es importante seguir desarrollando el sistema de manera gradual, asegurándose de que sea fácil de usar para los usuarios y se integre bien con otros sistemas que ya existan en la universidad.

Referencias

De la Torre Llorente, C., Castro, U. Z., Barros, M. A. R., & Nelson, J. C. (2010). Guía de Arquitectura N-Capas orientada al Dominio con .NET. España: Microsoft Iberica.