Reporte de Issues y Pull Requests
Issue #58 - Mejorar visualmente la vista de perfil del usuario con CSS y animaciones
Estado: open
Asignado a: carlitosgiovanniramos
Creado: 2025-06-07T08:46:34Z
Cerrado: None
Descripción
Rama sugerida: feature/mejoras-perfil-usuario
■ Objetivo
Transformar visualmente la interfaz de usuario del componente Perfil.jsx sin modificar su lógica funcional. Se busca mejorar la estética general, agregar microinteracciones, reestructurar la información y garantizar accesibilidad, adaptabilidad y personalización.
Nota: Lo detallado en este documento son sugerencias, queda a criterio propio el cómo se implementarán.
■ Instrucciones detalladas
1. ■ Estructura y Layout
[ ] Implementar un layout moderno con estructura en tarjetas:
Tarjetas separadas para información personal, académica, estadísticas e historial de inscripciones.
[ ] Aplicar espaciado generoso entre secciones para una mejor legibilidad.
[ ] En pantallas grandes:
Usar dos columnas con diseño tipo grid o flex.
2. ■ Sección de estadísticas
[ ] Mostrar métricas clave del usuario:
Número total de eventos completados.
Inscripciones activas o pendientes.
[ ] Representar estadísticas con números grandes o pequeños gráficos circulares o de progreso.
3. ■■ Elementos visuales clave
Imagen de perfil
[ ] Aplicar estilo mejorado al avatar:
Borde visible y sombra suave.
Efecto hover con scale(1.05) y transición suave.
Reemplazar ícono por un avatar placeholder estilizado si el usuario no ha subido imagen.
Badges y etiquetas
[ ] Rediseñar los badges de rol:
Rol: Estudiante, Administrador.
Diferentes colores e íconos por rol.
“Perfil verificado”, “Información incompleta”, etc.
4. ■■ Animaciones y transiciones
[ ] Añadir microinteracciones visuales:
Hover y focus en botones y enlaces.
Animaciones al cargar datos o al mostrar estado de acción (como guardado exitoso).
[ ] Usar transition en:
Secciones que se expanden.
Íconos que cambian de color o rotan.
[ ] Mejorar spinners de carga con animaciones más suaves y minimalistas (como loaders SVG).
5. ■ Reorganización de información
[ ] Agrupar campos personales:
Nombres, correo, teléfono, carrera, facultad.
[ ] Separar claramente las secciones de información:
Usar contenedores con encabezados y divisiones.
[ ] Mostrar íconos relevantes al lado de cada dato:
■ Email
■ Teléfono

■ Carrera
■■ Facultad
Carrera y Facultad
[ ] Mostrar la carrera y facultad de forma destacada.
[ ] Si es posible, incluir el logo o ícono de la facultad (SVG pequeña).
6. ■■ Inscripciones recientes
[ ] Reemplazar la lista textual por cards visuales de eventos:
Imagen del evento (si tiene).
Título, fecha y estado.
[ ] Añadir indicadores de progreso si el evento está en curso.
[ ] Agregar filtros rápidos para ver inscripciones por estado:
Pendientes, aceptadas, rechazadas, completadas.
7. ■ Personalización del perfil
[ ] Agregar una opción visual para que el usuario seleccione un tema de color (guardar en localStorage).
9. ■ Responsividad
[ ] Asegurar que el perfil sea completamente usable desde dispositivos móviles.
[ ] En pantallas pequeñas:
Usar tabs o acordeones para dividir secciones.
Apilar tarjetas verticalmente con márgenes internos adecuados.
10. ■ Accesibilidad
[ ] Verificar contraste de texto con fondo (uso de #222 sobre #fff mínimo).
[ ] Agregar atributos aria-label, alt en imágenes y botones.
[ ] Asegurar que todo contenido sea navegable mediante teclado.
[ ] Añadir mensajes de estado para lectores de pantalla al guardar o cambiar datos.
■ Pruebas sugeridas
Cambiar de rol (Admin ↔ Estudiante) y verificar que los badges cambien correctamente.
Cargar el perfil desde móvil y escritorio y validar que todo se vea correctamente.
* Activar transiciones y verificar que no interfieran con el rendimiento.
Importante: Manejar nombres de clases para css estandarizadas.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #57 - Implementar la selección, visualización y filtrado de modalidad de eventos
Estado: open
Asignado a: M4yb33
Creado: 2025-06-07T08:19:48Z
Cerrado: None
Descripción
Rama sugerida: feature/modalidad-eventos
■ Objetivo
Aprovechar el campo mod_eve del modelo evento para permitir una visualización y gestión completa de la modalidad del evento (PRESENCIAL, VIRTUAL, SEMIPRESENCIAL), desde la creación hasta su consulta y filtrado. Esta información es fundamental para mejorar la experiencia del usuario y la segmentación de eventos según necesidades logísticas.
■ Tipo de campo ya existente
ts
enum modalidad_evento {
PRESENCIAL,
VIRTUAL,

SEMIPRESENCIAL
}
Campo en el modelo:
ts
mod_eve: modalidad_evento
■ Tareas por módulo
1. ■ Formulario de eventos (EventForm.jsx)
[ ] Añadir un <select> para escoger la modalidad (mod_eve) con las tres opciones.
[ ] Actualizar el formData para incluir mod_eve.
[ ] En modo edición:
Prellenar automáticamente el campo con el valor actual del evento.
[ ] Validar que siempre se seleccione una modalidad antes de permitir guardar.
2. ■■ Controlador backend (evento.controller.js)
[ ] Validar en la creación y edición de eventos que el campo mod_eve:
Exista.
Tenga un valor permitido (PRESENCIAL, VIRTUAL, SEMIPRESENCIAL).
[ ] Guardar correctamente la modalidad en la base de datos.
[ ] Retornar mod_eve en las respuestas del backend (para que el frontend pueda visualizarlo).
3. ■■ Visualización de modalidad en el frontend
En EventosPublicos.jsx y EventosRoute.jsx
[ ] Mostrar el valor de mod_eve en cada card.
[ ] Incluir un ícono visual representativo:
■ MapPin → PRESENCIAL
■■ Monitor → VIRTUAL
■ Laptop → SEMIPRESENCIAL
En vista de detalles del evento
[ ] Mostrar claramente:
Modalidad
Ícono
etc (A criterio propio)
En vista de eventos finalizados o suspendidos
[ ] Modalidad debe seguir mostrándose como referencia histórica o administrativa.
4. ■■ Vista de administración (AdminEvents.jsx)
[ ] Añadir columna Modalidad en la tabla de eventos.
[ ] Mostrar cada modalidad con un badge estilizado:
Color diferente por tipo:
Verde → Virtual
Azul → Presencial
Naranja → Semipresencial
Texto capitalizado (Presencial, Virtual, Semipresencial).
■ Recomendaciones UX
[ ] Iconos y colores deben ser consistentes en toda la aplicación (cards, tablas, vistas de detalles).
[ ] Si se usa un color para una modalidad en una vista, debe repetirse en las demás.
[ ] Evitar texto técnico; usar términos legibles para usuarios no técnicos.
■ Pruebas sugeridas
Crear eventos de cada modalidad y verificar:
Que se guarda correctamente.
Que se renderiza correctamente en cards, detalles y administración.
Validar íconos y estilos visuales en desktop y móvil.
Simular cambio de modalidad al editar y verificar que se actualiza correctamente.
■ Notas adicionales
* Esta issue puede trabajarse de forma paralela a otras como filtros por tipo de evento, pero debe sincronizarse con ellas para evitar colisiones en filtros o vistas.

Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #56 - ■ Vista Pública de Eventos y Mejora de Navegación
Estado: closed
Asignado a: No asignado
Creado: 2025-06-06T22:44:57Z
Cerrado: 2025-06-07T07:16:08Z
Descripción
■ Objetivo
Separar la visualización de eventos según el estado de autenticación del usuario.
Usuarios autenticados acceden a la vista interna (EventsRoute), mientras que usuarios no autenticados acceden a una vista pública (EventosPublicos).
Se implementan mejoras en diseño, navegación y experiencia de usuario.
■ Cambios Principales
■ Autenticación y Redirección
Redirección condicional en App.jsx según sesión:
Usuarios autenticados → /eventos
Usuarios no autenticados → /eventos-publicos
Cambio de redirección inicial (/) y post-logout hacia /home.
■ Nuevas Vistas
EventosPublicos.jsx: Vista pública con tarjetas de eventos filtrados por estado y cupos.
EventosPublicos.css: Estilos responsivos, íconos e indicadores visuales de disponibilidad.
■ Navegación
Eliminado enlace a Certificados del Navbar y Home.
Navbar muestra:
Eventos disponibles para usuarios autenticados
Eventos públicos para usuarios no autenticados
Mejora visual en cierre de sesión (toast + redirección a /home).
■ Home.jsx
Nuevo botón “Eventos públicos” visible siempre.
Texto adaptado según sesión (“Explorar eventos públicos” vs “Explorar eventos disponibles”).
Footer oculta sección de inscripciones si no hay sesión activa.
■ Refactor y Correcciones
Mejora en botón “Ver Requisitos”.
Botón “Inscribirme” ahora redirige al login.
Corrección de tipo en evento.controller.js: id_eve_ins de String a Integer.
Mejoras visuales con animaciones para tarjetas.
■■ Archivos Modificados
App.jsx
Redirección condicional según autenticación y ajuste de ruta inicial.
Navbar.jsx
Actualización de enlaces según el tipo de usuario y eliminación de “Certificados”.
Home.jsx
Nuevo botón para acceder a eventos públicos y adaptación del footer y mensajes.
EventosPublicos.jsx
Nueva vista para mostrar eventos de tipo público y con cupos disponibles.
EventosPublicos.css
Estilos responsivos con soporte visual para disponibilidad, estado y navegación.
evento.controller.js

Corrección de tipo de dato para asegurar validación correcta en el backend.
■ Validaciones y Pruebas
La vista pública se muestra correctamente sin sesión iniciada.
Usuarios logueados acceden directamente a /eventos.
Navbar se adapta según el estado de sesión.
Los eventos públicos se filtran correctamente (estado y cupos).
Botón “Inscribirme” redirige correctamente al login.
Diseño visual validado en múltiples resoluciones.
Closes #50 ■
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #55 - Filtros por tipo de evento y visibilidad de cupos
Estado: open
Asignado a: NIXON-HS
Creado: 2025-06-06T15:21:10Z
Cerrado: None
Descripción
■ Objetivo de la Funcionalidad
Incorporar filtros dinámicos y relevantes en la vista **Eventos Públicos** para facilitar la navegación de usuarios no autenticados, permitiendo explorar por categorías y ocultar automáticamente eventos sin cupos disponibles, salvo que el usuario lo solicite explícitamente. Además, se implementarán filtros avanzados en la vista de **Gestión de Eventos** para usuarios administradores.
■ Rama sugerida
`feature/filtros-visibilidad-eventos`

Nota: Estas son sólo sugerencias, la correcta implementación queda a criterio propio.

■ Tareas a realizar
1. Implementar barra o menú de filtros en /eventos-publicos
[ ] Agregar un componente visual (barra superior o dropdown).
[ ] Filtros disponibles:
Software
Industrial
Públicos (visibles a todos)
Gratuitos
De paga
Eventos llenos
[ ] Los filtros deben actualizar dinámicamente la lista de eventos renderizados sin recarga de página (idealmente con useState / useEffect en React o similar).
2. Control de visibilidad por cupos
[ ] Para todos los filtros excepto “Eventos llenos”: mostrar solo eventos con cupo_dis_eve > 0.
[ ] Para el filtro “Eventos llenos”: mostrar solo eventos con cupo_dis_eve === 0.
[ ] La lógica debe funcionar en tiempo real cuando sea posible.
3. Filtros avanzados en vista Gestionar Eventos (ADMIN)
Implementar filtros combinables para facilitar la búsqueda y gestión:
[ ] Filtro por nombre o palabra clave
[ ] Filtro por tipo de evento (Curso, Congreso, Webinar, Charla, etc.)

[ ] Filtro por estado (Activo, Inactivo, Finalizado, Cancelado, Suspendido)
[ ] Filtro por rango de fechas (inicio y/o fin)
[ ] Filtro por carrera asociada (select múltiple o checkboxes)
[ ] Filtro por modalidad (Presencial, Virtual, Semipresencial)
[ ] Filtro por cupo máximo (mayor/menor a X)
[ ] Filtro por valor (\$) (gratuito o rango de precios)
[ ] Filtro por porcentaje mínimo de asistencia
[ ] Soporte para búsqueda combinada entre filtros
[ ] Opción para ordenar resultados (fecha, nombre, tipo, etc.)
[ ] Botón para limpiar todos los filtros
4. ■ Filtrado por modalidad
En EventosPublicos.jsx y EventosRoute.jsx
[ ] Agregar una barra de filtros con botones o dropdown:
Todos
Presencial
Virtual
Semipresencial
[ ] Al seleccionar una modalidad, deben mostrarse únicamente los eventos que coincidan.
[ ] Si no se selecciona ningún filtro, mostrar todos los eventos.
5. Rediseño de las cards de eventos
[ ] Cada tarjeta debe mostrar claramente:
■ Nombre del evento
■■ Fechas (inicio y fin)
■ Modalidad (virtual / presencial / semipresencial)
■ Cupos disponibles
■ Costo (si aplica)
[ ] Si cupo_dis_eve === 0 y no se está usando el filtro “Eventos llenos”, no debe mostrarse.
[ ] Si el evento aparece bajo el filtro “Eventos llenos”, mostrar badge o mensaje:
"Cupos agotados" (estilo visual: fondo gris o rojo con ícono ■)
■ Consideraciones adicionales
[ ] Asegurar que solo se muestren eventos con estado ACTIVO (salvo filtros administrativos).
[ ] Permitir combinar múltiples filtros.
[ ] Campo de búsqueda por nombre de evento como funcionalidad opcional pero recomendada.
[ ] Reutilizar componentes UI donde sea posible para mantener coherencia visual y lógica DRY.
■ Pruebas sugeridas
Ingresar a /eventos-publicos y verificar que se muestran solo eventos activos con cupos disponibles por defecto.
Activar filtro “Eventos llenos” y validar que aparecen solo los que tienen cupo_dis_eve === 0.
Probar cada combinación de filtros para validar coherencia y lógica.
Verificar que los filtros en la vista de administración funcionen correctamente y puedan combinarse.
Asegurar que el diseño sea responsive para móviles y escritorio.
Validar estados de vacío: si no hay eventos, mostrar mensaje adecuado (“No se encontraron eventos con esos filtros”).
■ Notas técnicas
Puede implementarse filtrado del lado cliente si se cargan todos los eventos inicialmente.
Si se esperan muchos eventos, preferible hacer filtrado del lado servidor con parámetros en la query.
Usar useState y useEffect para manejar el estado de los filtros en React (si aplica).
Estandarizar el uso de componentes para cards de eventos y menús de filtros para evitar duplicación de código.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno

Comentarios
Sin comentarios
Pull Request #54 - Implementar lógica de cupos en eventos y cards Feature/cupos en eventos eventos publicos
Estado: closed
Asignado a: No asignado
Creado: 2025-06-06T04:30:09Z
Cerrado: 2025-06-06T15:12:02Z
Descripción
Gestión Completa de Cupos para Eventos Académicos
Esta PR implementa un sistema completo de gestión de cupos para eventos académicos, permitiendo controlar la capacidad máxima de participantes y el seguimiento en tiempo real de cupos disponibles. El sistema incluye validaciones robustas, interfaces de administración intuitivas y alertas automáticas.
■ Funcionalidades Implementadas
■■ Cambios en Base de Datos
Nuevos campos en tabla evento:
cupo_max_eve: Cupo máximo permitido para el evento (1-10,000)
cupo_dis_eve: Cupos disponibles actuales
Migraciones automáticas para eventos existentes (valores por defecto: 50/50)
Seed actualizado con datos de prueba que incluyen cupos
■■ Backend - Lógica de Negocio
Validación de cupos:
Rango válido: 1-10,000 participantes
Validación de tipos de datos (solo enteros)
Prevención de valores negativos o cero
Sistema de auto-corrección:
Corrección automática de inconsistencias en datos de cupos
Sincronización entre cupos máximos y disponibles
Gestión dinámica de cupos:
Resta automática al aceptar inscripciones
Suma automática al rechazar inscripciones
Bloqueo automático cuando cupos llegan a cero
Validaciones de inscripción:
Verificación de cupos disponibles antes de aceptar
Mensajes de error claros y específicos
Control de estado de eventos (activo/bloqueado)
■ Frontend - Interfaces de Usuario
Panel de Administración:
Vista de eventos (AdminEvents.jsx):
Contador de cupos en tiempo real por evento
Alertas visuales codificadas por colores
Filtros y ordenamiento con información de cupos
Gestión de inscripciones (AdminEventInscription.jsx):
Alertas automáticas cuando cupos ≤ 3 (amarillo)
Alertas críticas cuando cupos = 0 (rojo)
Contador actualizado tras cada acción
Bloqueo visual de acciones cuando no hay cupos
Formularios de Eventos (EventForm.jsx):
Campos de cupos integrados:
Input para cupo máximo con validación en tiempo real
Inicialización automática de cupos disponibles
Mensajes de error específicos por tipo de validación
Restricciones de UI para prevenir entradas inválidas

Vista de Estudiantes (EventsRoute.jsx):
Información de cupos en tarjetas de eventos:
Contador visible de cupos disponibles
Indicadores visuales de disponibilidad
Deshabilitación de botones cuando cupos = 0
Estilos diferenciados para eventos llenos
■ Sistema de Alertas y Notificaciones
Alertas automáticas para administradores:
■ Sin cupos (0): Alerta roja crítica
■ Cupos disponibles: Indicador verde
Mensajes contextuales:
Errores de validación específicos
Confirmaciones de acciones exitosas
Indicadores de estado en tiempo real
■ Mejoras Visuales
Estilos CSS actualizados:
Indicadores de cupos con colores semánticos
Animaciones suaves para cambios de estado
Responsividad en todos los dispositivos
Consistencia visual con el sistema de diseño existente
Archivos Modificados
Backend:
prisma/
■■■ schema.prisma
■■■ migrations/20250605144556_agregar_cupos/
■■■ migrations/20250605150456_add_evento_quota_fields/
■■■ seed.js
src/controllers/
■■■ evento.controller.js
■■■ inscripcion.controller.js
Frontend:
src/views/admin/
■■■ AdminEvents.jsx
■■■ AdminEventInscription.jsx
■■■ styles/
■■■ AdminEvents.css
■■■ AdminEventInscription.css
src/components/
■■■ EventForm.jsx
src/routes/
■■■ EventsRoute.jsx
■■■ styles/
■■■ EventsRoute.css
■ Validaciones y Pruebas
Cupos máximos entre 1 y 10,000
Solo números enteros positivos
Cupos disponibles ≤ cupos máximos
Verificación de cupos antes de aceptar inscripciones
Auto-corrección de datos inconsistentes
Prevención de inscripciones cuando cupos = 0
Casos de Uso Cubiertos:
Creación de eventos con cupos personalizados
Edición de cupos existentes

Gestión automática durante inscripciones
Alertas proactivas para administradores
Experiencia de usuario informativa
Recuperación de datos inconsistentes
■ Beneficios del Sistema
Control Total: Gestión precisa de capacidad de eventos
Automatización: Actualización automática sin intervención manual
Prevención: Evita sobrecupos y conflictos de inscripción
Transparencia: Información clara para usuarios y administradores
Eficiencia: Reduce carga administrativa manual
Robustez: Sistema auto-correctivo y tolerante a errores
■ Compatibilidad
Retrocompatible: Eventos existentes mantienen funcionalidad
Migración automática: Asignación de cupos por defecto
Sin ruptura: API mantiene endpoints existentes
Progresiva: Funcionalidades se activan gradualmente
■ Métricas de Implementación
15+ commits con implementación incremental
2 migraciones de base de datos seguras
8 archivos de frontend modificados
3 controladores backend actualizados
Sistema de alertas completo implementado
Validaciones robustas en toda la aplicación
Close #49
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #53 - Implementar lógica de cupos en eventos y cards Feature/cupos en eventos eventos publicos
Estado: closed
Asignado a: No asignado
Creado: 2025-06-06T00:46:44Z
Cerrado: 2025-06-06T02:10:09Z
Descripción
■ Resumen de Cambios en la Rama feature/cupos-en-eventos_eventos-publicos
■■ Cambios en la Base de Datos (Schema)
Archivo: schema.prisma
Se agregaron nuevos campos al modelo evento:
cupomaxeve (Int): Cupo máximo del evento.
cupodiseve (Int): Cupo disponible del evento.
■ Cambios en el Backend
Controlador de Eventos (evento.controller.js):
Validación de cupos en la función validarEventoGeneral.
Manejo de cupos máximos en la creación (crearEvento) y actualización (actualizarEvento) de eventos.
Gestión automática de cupos disponibles.
Controlador de Inscripciones (inscripcion.controller.js):
Verificación de cupos disponibles antes de crear inscripciones.
Actualización automática de cupos al cambiar estados de inscripciones.
Gestión avanzada de estados: PENDIENTE, ACEPTADA, RECHAZADA, FINALIZADA, CANCELADA.
Nueva función cancelarInscripcion que libera cupos.

Rutas (routes):
Nuevos endpoints para la gestión de cupos.
Expansión de rutas de inscripción para manejar cancelaciones.
■ Cambios en el Frontend
Formulario de Eventos (EventForm.jsx):
Nuevo campo para ingresar el cupo máximo (cupomaxeve).
Validación obligatoria para cupo máximo (debe ser mayor a 0).
Ícono de usuarios agregado para mejorar la interfaz.
Ruta de Eventos (EventsRoute.jsx):
Visualización de cupos disponibles en las tarjetas de eventos.
Botón de inscripción deshabilitado cuando no hay cupos disponibles.
Manejo dinámico de estados de inscripción y cupos.
Vista de Administración (AdminEvents.jsx):
Información clara de cupos en el panel de administración.
Mejor visualización del estado de eventos y sus cupos.
Estilos (EventsRoute.css):
Estilos específicos para la visualización de cupos.
Estilos para botones deshabilitados cuando no hay cupos.
■ Archivos de Configuración y Otros
Frontend:
vite.config.js: Configuración de host para permitir conexiones externas.
package.json: Actualización de dependencias.
Backend:
Respaldo de configuración PostgreSQL.
Actualización del archivo .env(ejemplo).txt con nuevas variables.
Eliminación del archivo temporal fix-script.js.
Contexto de Autenticación (AuthContext.jsx):
Mejoras en manejo de tokens y autenticación.
■ Pruebas y Desarrollo
Actualización y mantenimiento de tests de la API de eventos.
Nuevo archivo de consultas para desarrollo.
Herramienta clear-token.html para manejo de tokens en debug.
■ Funcionalidades Implementadas
Gestión Completa de Cupos:
Definición de cupo máximo al crear eventos.
Actualización automática de cupos disponibles.
Validación de disponibilidad antes de inscripciones.
Estados de Inscripción Avanzados:
Flujo: PENDIENTE → ACEPTADA → FINALIZADA.
Estados que liberan cupos: CANCELADA, RECHAZADA.
Interfaz de Usuario Mejorada:
Visualización clara y en tiempo real de cupos disponibles.
Botones de inscripción deshabilitados cuando no hay cupos.
Información dinámica del estado de cupos.
Eventos Públicos:
Soporte completo para eventos de tipo "PÚBLICO".
Gestión diferenciada por tipo de evento.
■ Estadísticas de Cambios
Archivos modificados: 28
Líneas agregadas: +2,266
Líneas eliminadas: -193
Commits realizados: 24
■ Impacto Principal

Esta rama introduce un sistema robusto y completo para la gestión de cupos en eventos académicos, con validaciones automáticas, actualización en tiempo real de la disponibilidad, y una interfaz intuitiva que mejora la experiencia del usuario y la administración de eventos públicos y privados.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Andriu-Dex: Motivos del rechazo del Pull Request:
No se siguieron las instrucciones especificadas en la issue correspondiente.
No se realizó el pull de la rama develop, como se había indicado, lo que ha provocado múltiples conflictos.
Se hizo uso de inteligencia artificial sin revisiones previas, lo que compromete la calidad y coherencia del código.
Se han realizado tareas asignadas a otras issues, lo cual desvía el enfoque del desarrollo.
Se han generado archivos no deseados fuera de las carpetas principales, sin explicación ni justificación aparente.
Por estas razones, este pull request ha sido **rechazado**.
Issue #52 - Corrección de finalización de inscripciones sin cumplir requisitos
Estado: open
Asignado a: Gabriel-Spartan
Creado: 2025-06-05T10:26:06Z
Cerrado: None
Descripción
Rama sugerida: feature/finalizar-inscripcion-sin-requisitos
■ Objetivo
Corregir el comportamiento actual que impide finalizar una inscripción si no se cumplen los requisitos mínimos (nota o asistencia). Se busca permitir que la inscripción pueda finalizarse de todas formas, y que el sistema se encargue de indicar si fue aprobada o rechazada, evitando así bloqueos en la administración de eventos.
■ Contexto actual
En la vista “Validar Inscripciones”, una vez aceptada la inscripción, el administrador puede ingresar:
Nota (si el evento es un curso).
Porcentaje de asistencia.
Si la asistencia o nota ingresadas no cumplen con los mínimos requeridos, aparece una alerta:
“No cumple requisitos para finalizar: nota mínima X, asistencia mínima XX%”
■ Tareas a realizar
1. Modificar lógica de finalización
[ ] El sistema debe permitir finalizar la inscripción incluso si los valores ingresados no cumplen con los requisitos mínimos.
[ ] El botón “Finalizar inscripción” debe estar habilitado en todo momento, sin bloqueos por validación de nota o asistencia.
[ ] Validar que se guarde correctamente la información ingresada al presionar Finalizar.
2. Clasificar la inscripción como “Rechazada” si no cumple requisitos
[ ] Una vez finalizada la inscripción, el backend debe evaluar:
Si se cumplió asistencia mínima.
Si se cumplió nota mínima (si aplica).
[ ] Si no se cumplen los requisitos, marcar la inscripción como:
Rechazada por puntaje
Rechazada por asistencia
o ambas si corresponde.
[ ] Registrar internamente el motivo del rechazo (útil para reportes y certificados).
3. Vista del usuario en “Mis Inscripciones”

[ ] En la columna donde normalmente aparece el botón “Descargar certificado”:
No mostrar el botón si la inscripción fue rechazada.
En su lugar, mostrar mensajes informativos según el caso:
“Rechazado por puntaje”
“Rechazado por asistencia”
“Rechazado por puntaje y asistencia”
[ ] No mostrar la observación del administrador si fue rechazo automático.
4. Comportamiento del certificado
[ ] En caso de inscripción rechazada:
No generar ningún certificado.
No enviar certificado por correo.
No habilitar descarga manual.
■ Validaciones
[ ] Verificar que se respete el valor mínimo de nota/asistencia, pero sin impedir la acción.
[ ] Controlar en backend que solo las inscripciones aprobadas puedan acceder a lógica de generación de certificados.
[ ] Marcar en la base de datos el estado final de la inscripción (ej. rechazada_por_asistencia, rechazada_por_nota).
■ Pruebas sugeridas
Simular inscripción con valores válidos y confirmar que el certificado se genera correctamente.
Simular inscripción con:
Solo asistencia válida, pero nota baja.
Solo nota válida, pero asistencia baja.
Ambas por debajo del mínimo.
Verificar que se muestre el mensaje correcto en cada caso.
Verificar que el certificado no se genera ni se envía en estos casos.
■ Notas adicionales
Esta lógica será usada más adelante por otras funcionalidades.
Es importante aislar bien la lógica de “requisitos mínimos” en el backend para que sea reutilizable.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #51 - Ajustar la vista raíz y rediseñar el certificado entregado al usuario
Estado: open
Asignado a: carlitosgiovanniramos
Creado: 2025-06-05T10:14:53Z
Cerrado: None
Descripción
Rama sugerida: `feature/vista-raiz-certificado-disenio`
■ Objetivo
Actualizar la vista inicial del sistema para que sea más intuitiva al usuario no logueado y mejorar significativamente el diseño del certificado de participación o aprobación que se entrega al finalizar un evento.
1. Rediseñar el certificado en PDF
[ ] El diseño actual del certificado es muy básico, sin estructura visual atractiva.
[ ] Crear un nuevo diseño en orientación horizontal (landscape) con una presentación profesional.
[ ] Puede tomarse como ejemplo visual esta plantilla:
Plantilla de ejemplo
[ ] El nuevo certificado debe contener claramente los siguientes datos:
Nombres y apellidos completos del participante.

Cédula.
Carrera (si aplica).
Nombre del evento.
Tipo de certificado: participación o aprobación.
Asistencia total registrada (porcentaje).
Nota obtenida (solo si el evento tiene evaluación).
2. Generar PDF dinámicamente
[ ] Usar librería existente (por ejemplo pdfkit o similar si ya se encuentra en uso).
[ ] Validar el diseño en varios dispositivos: el PDF debe ser responsivo al ser descargado desde móvil o escritorio.
[ ] Asegurarse de usar tipografía profesional, buena jerarquía de texto, márgenes adecuados.
■ Visibilidad del botón “Descargar certificado” en Mis inscripciones
3. Condiciones para mostrar botón
[ ] El botón “Descargar certificado” solo debe aparecer si la inscripción del usuario cumple TODAS estas condiciones:
Fue aceptada por el administrador.
El evento está finalizado.
El usuario cumplió con los requisitos (asistencia mínima y/o nota si es curso).
El certificado ha sido generado correctamente.
4. Comportamiento si no cumple requisitos
[ ] Si el usuario no cumple con alguno de los requisitos (asistencia o nota), se debe mostrar en su lugar uno o más de los siguientes mensajes:
“Rechazado por puntaje”
“Rechazado por asistencia”
“Rechazado por puntaje y asistencia” (si aplica ambos).
[ ] En ese caso:
No se debe mostrar el botón de descarga.
No se debe enviar el certificado al correo electrónico.
■ Pruebas sugeridas
Acceder a / y verificar que no se redirige al login, sino al nuevo Home público.
Simular generación de certificados con:
Evento con solo asistencia
Evento con evaluación y asistencia
Evento con requisitos incumplidos
Revisar el diseño del PDF en escritorio y móvil.
Validar que el botón de descarga se oculta si no se cumplen los requisitos.
Probar distintos nombres largos y verificar que el PDF no se deforma.
■ Notas adicionales
El diseño debe verse limpio, profesional y alineado a la identidad visual del sistema.
Se puede incluir un sello digital o firma si se desea mejorar su autenticidad visual.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #50 - Cambiar Navbar y construir vista de “Eventos Públicos”
Estado: closed
Asignado a: M4yb33
Creado: 2025-06-05T10:05:32Z
Cerrado: 2025-06-07T08:02:37Z
Descripción
Rama en la que se trabajará: feature/eventos-publicos

■ Objetivo
Crear una vista dedicada exclusivamente a mostrar eventos públicos para usuarios **no logueados** y reorganizar el flujo de navegación para separar los eventos internos de los visibles a todo público.
■ Tareas a realizar
1. Cambiar nombre del ítem del Navbar (solo para usuarios autenticados)
[ ] Localizar el ítem del menú donde aparece "Eventos".
[ ] Renombrarlo a "Eventos disponibles":
```html
Eventos → Eventos disponibles
```
[ ] Asegurarse de que solo se muestre en el Navbar si el usuario está logueado (session.user o equivalente en Astro o React).
2. Crear botón en Home.jsx que redirige a la vista pública
[ ] En el componente Home.jsx, ubicar el contenedor de botones (por ejemplo junto a “Ver carreras”).
[ ] Añadir un nuevo botón con texto "Eventos públicos".
[ ] El botón debe hacer un redirect (usando Link de React Router o <a> si aplica) a una ruta nueva: /eventos-publicos.
3. Construir vista nueva: Eventos Públicos
[ ] Crear una vista accesible en /eventos-publicos.
[ ] Esta vista no requiere login.
[ ] Mostrar todos los eventos con estado ACTIVO e INACTIVO.
Cada card de evento debe incluir:
Nombre del evento
Fechas de inicio y fin
Modalidad: virtual o presencial
Cupos disponibles (usando cupo_dis_eve)
Costo del evento (si existe campo)
Botón “Ver Requisitos”
Este botón lleva al detalle del evento (/evento/:id)
4. Vista de detalle de evento (cuando el usuario hace clic en “Ver Requisitos”)
[ ] Mostrar la siguiente información:
Descripción completa
Objetivos del evento
Modalidad (presencial / virtual)
Fechas
Cupos disponibles (cupo_dis_eve)
Requisitos para inscribirse (por carrera, por tipo de usuario, etc.)
Costo (si aplica)
Datos extra si hay
[ ] Botón "Inscribirme" que redirige al login.
■ Consideraciones adicionales
[ ] Reutilizar los componentes de cards existentes si es posible.
[ ] Aplicar estilos coherentes con el diseño general del sistema.
[ ] Validar que los eventos mostrados cumplan con estado ACTIVO e INACTIVO y no estén llenos (cupo_dis_eve > 0).
■ Control de acceso
[ ] Esta vista debe ser visible a todos, incluso si no hay sesión iniciada.
[ ] El botón "Eventos disponibles" en el navbar solo aparece si el usuario está logueado.
[ ] El botón "Eventos públicos" en Home.jsx siempre está visible.
■ Pruebas sugeridas
Entrar como usuario no logueado y verificar que la vista de eventos públicos se carga correctamente.
Validar que los datos del evento (modalidad, fechas, cupos) se muestran de forma correcta.
Verificar que el botón "Inscribirme" redirige al login.

Asegurar que si el evento está lleno (cupo_dis_eve == 0) no se muestre (será cubierto por la siguiente issue de filtros).
Verificar que el botón “Ver requisitos” lleva al detalle correcto del evento.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #49 - Implementar lógica de cupos en eventos y cards
Estado: closed
Asignado a: NIXON-HS
Creado: 2025-06-05T09:46:52Z
Cerrado: 2025-06-07T08:00:27Z
Descripción
Rama en la que se trabajará: feature/cupos-en-eventos
■ Objetivo
Permitir que cada evento tenga una cantidad limitada de cupos disponibles. Una vez alcanzado ese límite, se deben bloquear nuevas inscripciones y ocultar el evento de las vistas públicas (excepto cuando se use el filtro “eventos llenos”).
■ Contexto y campos existentes
En la base de datos ya existen los siguientes campos en la tabla de eventos:
```ts
cupo_max_eve Int // Número total de cupos al momento de crear el evento
cupo_dis_eve Int // Número de cupos disponibles restantes
```
■ Tareas a realizar
1. Al crear un evento
[ ] Agregar al formulario de creación de eventos un campo para establecer el cupo_max_eve.
[ ] Inicializar el campo cupo_dis_eve con el mismo valor de cupo_max_eve.
[ ] Validar en backend y frontend:
Que no se puedan ingresar valores negativos ni cero.
Que el campo sea obligatorio.
Mostrar mensajes de error claros en caso de ingreso inválido.
2. Visualización en cards de eventos
[ ] Mostrar en la card de cada evento:
“Cupos disponibles: X” (donde X = cupo_dis_eve).
Cambiar estilo visual (ej. color rojo o badge) si X = 0.
[ ] Mostrar el contador actualizado tras cada inscripción aceptada.
3. Lógica de inscripción
[ ] Cuando un usuario se inscriba:
No modificar cupos aún.
[ ] Cuando un admin acepte la inscripción:
[ ] Restar 1 al campo cupo_dis_eve.
[ ] Verificar que cupo_dis_eve sea > 0 antes de aceptar.
[ ] Si cupo_dis_eve = 0 después de aceptar, bloquear automáticamente nuevas inscripciones.
[ ] (Opcional) Mostrar alerta al admin si ya no hay cupos restantes.
4. Comportamiento cuando no hay cupos
[ ] Ocultar el evento de las vistas:
Eventos públicos
Eventos disponibles
(creación de vistas asignadas a @M4yb33)
[ ] El evento solo debe mostrarse si:

Se filtra específicamente por “Eventos llenos”.
[ ] Deshabilitar el botón “Inscribirse” o mostrar el mensaje “Cupos agotados”.
■ Consideraciones adicionales
[ ] Evitar manipulación de cupos manualmente a través de consola (validación en backend).
[ ] (Opcional) Mostrar una barra de progreso o porcentaje de cupos ocupados.
■ Pruebas sugeridas
Crear evento con diferentes valores válidos e inválidos de cupos.
Aprobar inscripciones hasta llegar a 0 cupos y verificar que se bloquean nuevas.
Verificar la vista con filtro “eventos llenos”.
Intentar aprobar una inscripción con cupo 0 y verificar que no se permite.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #48 - Merge de Develop a Main
Estado: closed
Asignado a: No asignado
Creado: 2025-06-05T00:31:56Z
Cerrado: 2025-06-05T00:32:20Z
Descripción
■ Nuevas Funcionalidades
Eventos por múltiples carreras: Asignación mediante checkboxes y opción "Evento general".
Gestión de documentos: Carga separada, vista previa y soporte para varios formatos.
Procesamiento de imágenes: Uso de Sharp para compresión y formato A4.
Perfil de usuario: Edición de datos, historial y manejo de documentos.
Limpieza automática: Eliminación de archivos temporales cada 12 h.
Migraciones Prisma: Organización y control con scripts y lock.
Carta de motivación: Campo adicional en inscripciones, visible para admins.
Editor de avatar: Recorte, zoom y carga automática con vista inmediata.
Certificados académicos: PDF con diseño, código único, descarga y validación pública.
■■ Correcciones Críticas
Reinscripción a eventos:
Frontend: Ajustes en lógica de botones y manejo de estado.
Backend: Corrección de campos (id_ins_per) y validaciones.
Script fix-script.js para correcciones automáticas.
■ Pruebas Realizadas
Reinscripciones, carga de archivos, generación de certificados, cambio de avatar, limpieza automática y migraciones.
Commits Vinculados
Merge pull request #48 from Andriu-Dex/develop
Autores de Commits
Andriu
Comentarios
Andriu-Dex: Aprobado por el líder de equipo.
Pull Request #47 - Implementación de Múltiples Funcionalidades en Academic Events
Estado: closed
Asignado a: No asignado
Creado: 2025-06-05T00:05:16Z
Cerrado: 2025-06-05T00:13:55Z
Descripción

Este Pull Request completa una serie de **funcionalidades nuevas** y **mejoras estructurales**, así como la **corrección de errores críticos** detectados en la plataforma **Academic Events**. Las tareas realizadas impactan positivamente en la experiencia del usuario, escalabilidad del sistema y mantenibilidad del código.

■ Funcionalidades Completadas

■ 1. Asociación de Eventos a Múltiples Carreras

Se completó la funcionalidad para asignar múltiples carreras a un evento mediante checkboxes.
Se añadió una opción "Todas las carreras / Evento general" que desactiva otras carreras.
Se adaptó el backend para registrar relaciones múltiples en la tabla evento_carrera.

■ 2. Mejora en la Gestión y Visualización de Documentos

Se separó la carga de documentos del proceso de registro.
Se permite ahora cargar archivos PDF, JPG, PNG, y GIF desde el perfil.
Se implementó la previsualización y validación de archivos antes del envío.
La visualización se realiza directamente en el navegador mediante window.open.

■■ 3. Procesamiento de Imágenes y PDFs

Se integró Sharp para mejorar el procesamiento de imágenes:
Auto-rotación EXIF
Reducción de tamaño con compresión
Calidad 95% para JPEG
Se generan documentos en formato A4 centrados y bien estructurados.

■ 4. Perfil de Usuario Funcional

El usuario puede ver y editar su información personal.
Puede subir y administrar documentos desde una interfaz clara.
Visualiza su historial de inscripciones con estado actualizado.

■ 5. Servicio de Limpieza Automática

Se creó cleanupService.js que se ejecuta cada 12 horas.
Elimina archivos en /uploads/temp con más de 24 horas.
Registra logs detallados y maneja errores sin afectar el sistema.

■■ 6. Migraciones Prisma y Control de Cambios

Se organizó la carpeta prisma/migrations para mantener la estructura y evolución del esquema.
Se incluyó migration_lock.toml, scripts SQL, y migration-script.js.
Se utilizaron comandos prisma migrate dev, status, y deploy para aplicar los cambios.

■ 7. Carta de Motivación en Inscripción

Se habilitó la opción para que los estudiantes incluyan una carta de motivación al inscribirse.
La carta se almacena como texto en la base de datos y es visible para los administradores mediante un modal.

■■ 8. Editor de Imagen de Perfil (Avatar)

Se implementó una función que permite cambiar la imagen de perfil haciendo clic sobre el avatar.
Se permite recorte, rotación y zoom.
La imagen se sube automáticamente a Imgur y se actualiza en la base de datos.
La imagen se muestra inmediatamente en la interfaz.

■ 9. Sistema de Certificados Académicos

Estructura Técnica:

Nuevo modelo certificado en Prisma, vinculado a inscripciones.
Enum tipo_certificado para definir si es de participación o aprobación.
Directorios gestionados automáticamente con directory.utils.js.

Funciones implementadas:

Generación de PDF horizontal con diseño profesional y código único.
Descarga directa y envío por correo con HTML personalizado.
Validación de certificados a través de una ruta pública con el código único.

■■ Correcciones Críticas Realizadas

■ Reinscripción a Eventos

■ Problema:

Los usuarios no podían reinscribirse a eventos rechazados. El backend aceptaba la solicitud, pero el frontend mostraba un error genérico y el botón no se deshabilitaba.
■ Diagnóstico:
Frontend: errores de sintaxis y control de estado en EventsRoute.jsx.
Backend: referencia incorrecta a campos inexistentes (id_ins_car_mot en lugar de id_ins_per), y mal uso de la variable cartaMotivacion.
■ Soluciones Aplicadas:
Frontend:
Corregido el espaciado, indentación y estructura en EventsRoute.jsx.
Añadido el estado inscripcionesRechazadas.
Actualizada la función inscribirse() para procesar reinscripciones correctamente.
Lógica del botón "Inscribirme" adaptada a los distintos estados de inscripción.
Backend:
Corregido acceso a req.body.carta_motivacion.
Actualizado el controlador inscripcion.controller.js y el esquema de Prisma con el campo correcto id_ins_per.
Script de soporte:
Se creó fix-script.js para reemplazar referencias incorrectas automáticamente en el controlador:
```js
// Reemplaza id_ins_car_mot por id_ins_per en secciones específicas
```
■ Resultado:
Reinscripciones funcionales.
Botón "Inscribirme" correctamente deshabilitado tras reinscripción.
El frontend ahora refleja el estado real de la inscripción.
■ Pruebas Realizadas
■ Pruebas manuales de reinscripción tras rechazo.
■ Verificación visual y funcional de la carga de documentos.
■ Validación de certificados generados con código único.
■ Pruebas de cambio de avatar y visualización inmediata.
■ Pruebas de limpieza automática con archivos de más de 24h.
■ Validación de migraciones en staging y local.
■ Archivos Relevantes Modificados
```
frontend/
■■ src/components/EventForm.jsx
■■ src/components/AdminEvents.jsx
■■ src/pages/EventsRoute.jsx
■■ src/components/ProfileAvatar.jsx
backend/
■■ controllers/perfil.controller.js
■■ controllers/inscripcion.controller.js
■■ controllers/certificado.controller.js
■■ routes/certificado.routes.js
■■ services/cleanupService.js
■■ utils/certificado.utils.js
■■ utils/directory.utils.js
■■ prisma/schema.prisma
scripts/
■■ fix-script.js
```
Instrucciones para probar los cambios:
Limpiar caché de npm

Ejecutar el siguiente comendo dentro de frontend y dentro de backend:
```
npm cache clean --force
```
1. Borrar los node_modules del backend y del frontend e instalar de nuevo las dependencias con:
```powershell
npm install --legacy-peer-deps
```
Si hay errores usar:
Instalar con --force
```
npm install --force
```
Ahora hay que editar el archivo .env del backend con los siguientes valores:
```env
Conexión a la base de datos PostgreSQL
DATABASE_URL=postgresql://postgres:TU_CONTRASEÑA@localhost:5432/academicevents
Configuración del servidor
PORT_BACKEND=3000
HOST=TU_IP_LOCAL
JWT y Email
JWT_SECRET=TU_CLAVE_SECRETA
Configuración SMTP
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=tu_correo@gmail.com
SMTP_PASS=tu_contraseña_de_aplicacion
```
Ahora hay que generar la base de datos:
```powershell
Generar migraciones y aplicar cambios a la base de datos
npx prisma migrate reset --force
npx prisma migrate dev --name agregar_cupos
npx prisma generate
npm run seed
```
Configurar variables de entorno
Editar el archivo .env del frontend con:
```env
VITE_API_URL=http://TU_IP_LOCAL:3000
VITE_HOST=TU_IP_LOCAL
VITE_PORT=5173
```
(Ver .env de ejemplo)
Configurar el package.json del frontend:
Cambiar el contenido de esta linea código con la IP correspondiente:
```env
"dev": "vite --host colocar_tu_IP --port 5173",
```
Iniciar el sistema
```powershell
cd backend
```

npm run dev
```
```powershell
cd frontend
npm run dev
```
Notas importantes:
Base de datos:
Asegúrate de tener PostgreSQL instalado y corriendo
Asegurarte de tener la base de datos llamada "academicevents"
El usuario y contraseña en DATABASE_URL deben coincidir con tu configuración local de PostgreSQL
Variables de entorno:
Reemplaza TU_IP_LOCAL con tu dirección IP local (ejemplo: 192.168.1.100)
En JWT_SECRET usa una clave segura
Para el correo SMTP, necesitarás configurar una "contraseña de aplicación" en tu cuenta de Gmail (En el .env de ejemplo esta el link para generarlo)
Puertos:
Verifica que los puertos 3000 (backend) y 5173 (frontend) estén disponibles
Si necesitas cambiar los puertos, actualiza las variables de entorno correspondientes
Verificación
Para verificar que todo está funcionando correctamente:
El backend debería mostrar: "■Servidor corriendo en http://TUIPLOCAL:3000■"
El frontend debería estar accesible en: "http://TUIPLOCAL:5173"
Deberías poder acceder al sistema con las credenciales de prueba:
Usuario: admin@uta.edu.ec
Contraseña: Admin12345
Usuario: estudiante@uta.edu.ec
Contraseña: Admin12345
--
Close #46
Commits Vinculados
Merge pull request #47 from Andriu-Dex/feature/validacion-inscripcion-carreras
Autores de Commits
Andriu
Comentarios
Gabriel-Spartan: Funcional todo lo mencionado, se puede subir todo lo que se necesita hacer
Issue #46 - Asociación de múltiples carreras, mejora de carga de documentos y perfil de usuario
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-06-02T10:29:30Z
Cerrado: 2025-06-05T02:38:06Z
Descripción
■ **Objetivo:** Permitir la asignación flexible de eventos a múltiples carreras.
Tareas:
[ ] Actualizar EventForm.jsx con checkboxes para selección múltiple de carreras.
[ ] Añadir checkbox "Todas las carreras / Evento general" que desactive los demás al seleccionarse.
[ ] Usar la tabla evento_carrera para registrar la relación N\:N.
[ ] Adaptar la lógica del backend para persistencia múltiple.
2. ■ Mejora de la Gestión y Carga de Documentos
■ **Objetivo:** Separar y mejorar el proceso de carga de archivos.
Tareas:
[ ] Eliminar la carga de documentos del formulario de registro.
[ ] Añadir un mensaje indicando que la carga se realiza en el perfil.
```

[ ] Permitir tipos PDF, JPG, PNG, GIF.
[ ] Implementar previsualización y validaciones antes de la subida.
3. ■■ Procesamiento Avanzado de Imágenes y PDFs
■ **Objetivo:** Generar PDFs de alta calidad desde imágenes.
Tareas:
[ ] Usar Sharp para compresión y ajuste de imágenes (JPEG 95%, PNG nivel 6, auto-rotación, 300 DPI).
[ ] Generar PDFs A4 con centrado, márgenes y diseño institucional.
[ ] Seleccionar métodos adecuados según formato (embedJpg / embedPng).
4. ■ Visualización Web de Documentos
■ **Objetivo:** Abrir archivos directamente en el navegador.
Tareas:
[ ] Reemplazar vscode:// por window.open(url, "_blank").
[ ] Validar que los documentos combinados incluyan al menos una página.
5. ■ Refactorización del Middleware y Controlador
■ **Objetivo:** Gestión segura y ordenada de archivos temporales.
Tareas:
[ ] Modificar upload.js para guardar en /uploads/temp.
[ ] Crear limpiarArchivosTemporales para borrar archivos luego del uso.
[ ] En perfil.controller.js, aplicar limpieza automática y guardar solo el archivo final.
[ ] Unificar las rutas para que usen { upload }.
6. ■ Implementación de Perfil de Usuario
■ **Objetivo:** Centralizar gestión personal y académica del usuario.
Tareas:
[ ] Crear sección de perfil editable por el usuario.
[ ] Permitir visualización, carga y modificación de documentos.
[ ] Mostrar historial de inscripciones y estado de documentos.
[ ] Diseñar una interfaz clara, intuitiva y responsiva.
7. ■ Servicio de Limpieza Automática
■ **Objetivo:** Mantenimiento del sistema mediante eliminación programada.
Tareas:
[ ] Crear cleanupService.js que:
Se ejecute cada 12 horas.
Elimine archivos de más de 24 horas.
Registre operaciones con logs detallados.
[ ] Integrar el servicio al arranque del servidor.
8. ■■ Sistema de Control de Migraciones
■ **Objetivo:** Controlar cambios en el esquema de base de datos.
Tareas:
[ ] Organizar migraciones en prisma/migrations/.
[ ] Incluir:
migration.sql
migration_lock.toml
migration-script.js
[ ] Usar comandos:
npx prisma migrate dev
npx prisma migrate status
npx prisma migrate deploy
[ ] Asegurar integridad entre entornos de desarrollo, pruebas y producción.
9. ■ Carta de Motivación en Inscripciones
■ **Objetivo:** Añadir valor académico a las inscripciones.
Tareas:
[ ] Incluir campo de carta de motivación en el formulario de inscripción.

[ ] Guardar la carta en la base de datos.
[ ] Mostrarla en un modal accesible para administradores.
10. ■■ Avatar de Usuario con Edición
■ **Objetivo:** Personalización visual del perfil.
Tareas:
[ ] Permitir a los usuarios hacer clic en su avatar para editarlo.
[ ] Añadir opciones para:
Cargar desde dispositivo.
Zoom, rotación y recorte.
[ ] Subir la imagen a Imgur.
[ ] Guardar la URL en la base de datos.
[ ] Mostrar la imagen actualizada al instante.
11. ■ Sistema de Certificados Académicos
■ **Objetivo:** Generar, validar y distribuir certificados profesionales.
Tareas:
a. Estructura de Datos
[ ] Añadir el modelo certificado y enum tipo_certificado al esquema Prisma.
[ ] Asociar cada certificado con una inscripción única.
b. Generación de Certificados
[ ] Diseñar certificados en PDF con formato horizontal profesional.
[ ] Incluir datos del evento, participante, tipo de certificado y código de validación.
[ ] Utilizar PDFKit y certificado.utils.js para crearlos dinámicamente.
c. Validación
[ ] Generar códigos únicos para cada certificado.
[ ] Crear endpoint público /certificados/validar/:codigo.
d. Envío y Descarga
[ ] Permitir descarga desde navegador.
[ ] Enviar certificados por correo electrónico con HTML personalizado.
e. Middleware y Seguridad
[ ] Proteger rutas sensibles con verificarToken y verificarPropietarioCertificado.
[ ] Permitir validación externa sin login.
f. Estructura de Almacenamiento
[ ] Usar directorios /uploads/certificados, /temp, y /profiles.
[ ] Implementar directory.utils.js para asegurarse de su existencia al iniciar.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #44 - Integración Dinámica de Misión y Visión de la Facultad: Backend y Frontend Conectados a la Base de Datos Real
Estado: open
Asignado a: titoma1500
Creado: 2025-05-31T04:15:33Z
Cerrado: None
Descripción
Actualmente, la edición de misión y visión en el panel administrativo usa un modelo y API ficticios (`configuracion`) que **no existen en la base de datos real**. Sin embargo, los campos correctos (`mis_fac` y `vis_fac`) están definidos en la tabla real `facultad` del modelo Prisma.
Objetivo:
Conectar la edición de misión, visión y autoridades directamente a la tabla `facultad`, asegurando que los cambios realizados por el administrador se guarden, consulten correctamente y se reflejen en la

web pública. Es posible que se requieran cambios en la base de datos prisma. Consultar dudas con @Gabriel-Spartan.
■ Alcance y pasos a seguir
1. Backend
Eliminar uso del modelo, controlador y rutas de configuracion si no se usan en otros módulos.
Crear endpoints RESTful en el controlador de facultad para:
GET /api/facultades/:id — Obtener misión y visión.
PUT /api/facultades/:id — Actualizar misión y visión.
Restringir actualización solo a usuarios con rol ADMIN.
Validar guardado correcto en campos mis_fac y vis_fac de la tabla facultad.
2. Frontend
Modificar vista AdminConfiguracion.jsx para:
Consultar misión y visión usando los nuevos endpoints de facultad.
Actualizar misión, visión y autoridades a través de esos endpoints.
Eliminar referencias a /api/configuracion y usar rutas correctas.
Hacer la edición intuitiva y mostrar mensajes claros de éxito o error.
3. Vista pública
Verificar que la misión, visión y autoridades en la web pública Home.jsx
Confirmar que reflejan los cambios realizados por el administrador.
4. Validaciones y seguridad
Restringir edición solo a administradores (Ya implementado pero revisarlo).
Validar en backend y frontend que los campos no estén vacíos antes de guardar.
■ Detalles técnicos y de entrega
Rama de trabajo:
feature/facultad-mision-vision-autoridades
Modelo de datos:
Utilizar campos mis_fac y vis_fac de la tabla facultad en Prisma, si hace falta algo consultar con el analista de datos..
Validaciones:
Implementar validaciones para evitar inconsistencias y errores.
Permisos:
Asegurar que solo administradores puedan editar.
Manejo de errores:
Solución robusta con mensajes claros para usuarios.
■ Criterios de aceptación
Misión, visión y autoridades se editan y consultan desde la tabla real facultad.
Frontend y backend correctamente integrados.
Cambios se reflejan en la web pública.
Solo administradores tienen permiso para editar.
Documentación clara para continuar integración.
Pruebas de validación y seguridad aprobadas.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #43 - Merge Develop a Main
Estado: closed
Asignado a: No asignado
Creado: 2025-05-30T20:16:18Z
Cerrado: 2025-05-30T20:16:45Z
Descripción

Se integran al branch principal todas las funcionalidades completas desarrolladas en la rama `develop`, que incluyen:
Funcionalidades de usuario
Registro, inicio de sesión y gestión de perfiles de usuarios.
Implementación de diferentes roles: ADMIN, ESTUDIANTE y GENERAL.
Visualización y edición de datos personales y asociación con carreras académicas.
Eventos académicos y gestión de inscripciones
Creación y administración de eventos de distintos tipos: CURSO, CONGRESO, WEBINAR, CHARLA, SOCIALIZACION, y PÚBLICO.
Visualización de eventos disponibles, con fechas de inicio y fin, y carga de imágenes asociadas.
Inscripción de usuarios con gestión completa del estado (PENDIENTE, ACEPTADA, RECHAZADA, FINALIZADA) y carga de comprobantes de pago.
Cursos especiales
Funcionalidades específicas para eventos tipo CURSO, incluyendo registro de asistencia y gestión de calificaciones.
Generación y descarga de certificados para participantes que cumplan con el porcentaje mínimo de asistencia.
Administración del sistema
Panel de administración con dashboard de estadísticas.
Gestión integral de usuarios, inscripciones, facultades, carreras y eventos.
Control y asociación entre entidades académicas y eventos.
Ajustes generales de la plataforma.
Infraestructura técnica y seguridad
Middleware para autenticación y control de acceso basado en roles.
Protección de rutas sensibles según permisos.
Almacenamiento eficiente de documentos e imágenes (comprobantes, certificados, PDFs).
Servicio integrado de correo electrónico para comunicaciones.
Base de datos
Modelos de datos definidos con Prisma, con relaciones claras entre entidades.
Scripts de semilla para carga inicial de datos.
Con esta integración, la aplicación cuenta con una base robusta y completa para la gestión académica y eventos, con un sistema seguro y escalable.
Commits Vinculados
Merge pull request #43 from Andriu-Dex/develop
Autores de Commits
Andriu
Comentarios
Andriu-Dex: ■ Aprobado por el líder de proyecto
Pull Request #42 - ■ Mejoras de navegación e interfaz para Admin y Estudiante + Nueva vista Home
Estado: closed
Asignado a: No asignado
Creado: 2025-05-29T00:56:07Z
Cerrado: 2025-05-30T19:42:30Z
Descripción
■ Descripción general
Este Pull Request introduce varias mejoras significativas enfocadas en la experiencia de usuario, navegación basada en roles y presentación visual. Ahora, el sistema redirige adecuadamente al usuario según su rol tras el login, y se agrega una nueva vista de inicio moderna para los estudiantes. Además, se mejora la integración de rutas de administración directamente en el Navbar.
■ Cambios realizados
■ Navegación general:
■ Navbar.jsx:

Se modificó para que muestre enlaces condicionales según el rol (ESTUDIANTE, ADMIN, usuario general).
Las rutas administrativas como “Gestionar eventos”, “Validar inscripciones”, etc., se movieron desde el AdminDashboard al Navbar.
■ Navbar.css:
Se actualizó con nuevos estilos para una navegación más limpia, moderna y responsiva.
Mejora la presentación de botones, enlaces y espaciado para diferentes tamaños de pantalla.
■ Login.jsx:
Redirige a:
/home si el usuario es estudiante.
/admin si el usuario es administrador.
■■ App.jsx:
Añadida la ruta /home como entrada predeterminada para estudiantes.
Se aseguraron las rutas privadas según el tipo de usuario.
■ Nueva vista de inicio para estudiantes
■ Home.jsx:
Nueva vista visual e institucional que da la bienvenida al estudiante tras el login.
Muestra:
Estadísticas de la facultad
Carreras disponibles
Autoridades de la facultad
Misión y visión
Accesos directos a eventos y certificados
Sección de contacto y pie de página
■ Home.css:
Estilos completamente nuevos, responsivos y con animaciones para mejorar la UX.
■■■ Panel administrativo (ADMIN):
■ AdminDashboard.jsx:
Rediseñado como una vista visual-resumen en vez de menú de navegación.
Muestra accesos rápidos en formato de tarjetas y descripción de funcionalidades.
Se eliminaron enlaces redundantes ya manejados en el Navbar.
■ AdminDashboard.css:
Estilos nuevos institucionales y responsivos.
Tarjetas interactivas con íconos y colores oficiales.
■ ¿Cómo probar estos cambios?
■ Asegúrate de tener la base de datos y migraciones aplicadas:
npx prisma migrate dev
■ Instala dependencias si aún no lo has hecho:
npm install
■ Corre el backend y frontend:
npm run dev # en backend
npm start # en frontend
■ Inicia sesión como:
Estudiante: te redirige a /home con la nueva vista.
Administrador: te lleva a /admin con el dashboard rediseñado.
■ Verifica la responsividad en dispositivos móviles y escritorio.
■ Asegúrate de que el Navbar se renderice correctamente según el rol.
■ Archivos añadidos o modificados
■ frontend/src/views/Home.jsx (nuevo)
■ frontend/src/views/styles/Home.css (nuevo)
■ frontend/src/views/admin/styles/AdminDashboard.css (nuevo)
■ frontend/src/views/admin/AdminDashboard.jsx (modificado)
■ frontend/src/components/Navbar.jsx (modificado)

■ frontend/src/components/styles/Navbar.css (modificado)
■ frontend/src/views/Login.jsx (modificado)
■ frontend/src/App.jsx (modificado)
■ Este PR mejora la navegación, la experiencia visual del sistema y separa claramente las vistas y rutas según el tipo de usuario, sin alterar la lógica del backend.
■ Close #33
Commits Vinculados
Merge pull request #42 from Andriu-Dex/feature/Home-Dashboard
Autores de Commits
Andriu
Comentarios
Sin comentarios
Issue #41 - Implementar CRUD completo para gestión de carreras
Estado: closed
Asignado a: No asignado
Creado: 2025-05-28T07:29:47Z
Cerrado: 2025-05-31T04:02:21Z
Descripción
Se requiere implementar el módulo completo de gestión de carreras para la plataforma AcademicEvents. Este módulo será accesible desde el panel de administración y permitirá:
■ Funcionalidades esperadas
Crear nuevas carreras
Validación de campos requeridos (ej. nombre único).
Posibilidad de vincular con facultades si se requiere en el modelo.
Listar todas las carreras registradas
Mostrar información clave.
Editar carrera existente
Modificar nombre de la carrera.
Validar que no se repitan nombres ya existentes.
Eliminar carrera
Permitir eliminación solo si no hay estudiantes inscritos vinculados (control en backend).
* Mostrar advertencia de confirmación.
■ Restricciones
Solo accesible por usuarios con rol de administrador.
Validaciones tanto en frontend como backend.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #40 - Lógica condicional para campos de curso en inscripciones
Estado: closed
Asignado a: No asignado
Creado: 2025-05-25T23:21:16Z
Cerrado: 2025-05-27T22:45:05Z
Descripción
■ Descripción del Pull Request:
Este pull request implementa la lógica visual condicional para mostrar u ocultar elementos específicos en la vista de inscripciones de estudiantes (MyInscriptions.jsx) según el tipo de evento (evento.tip_eve).
■ Cambios realizados:

Se agregó una condición para que los campos de nota final y asistencia solo se muestren si el evento es de tipo CURSO.
Se mantiene el diseño limpio y consistente.
Se evita mostrar campos innecesarios en charlas, webinars, congresos, etc.
El backend no fue modificado, como exigía el issue.
■ Archivos modificados:
frontend/src/views/MyInscriptions.jsx
Closes #34
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Gabriel-Spartan: ## ■ ¡Gracias por tu contribución @titoma1500!
■ **Aunque el pull request cumple con lo solicitado** en el issue #34 y la lógica implementada es correcta, lamentablemente se basa en una versión muy antigua del sistema.
■■ A pesar de haberse resuelto los conflictos, integrarlo ahora generaría más problemas que beneficios, debido a la cantidad de cambios acumulados desde esa versión.
■ Por este motivo, se va a cerrar este PR sin hacer merge.
■■ Si lo deseas, puedes tomar como base la lógica implementada y aplicar esos cambios sobre la versión más reciente del sistema el cual a futuro el **PM** del equipo le asignara con mas detalle las actividades a desarrollar y bajo que versiones del sistema.
■ Agradecemos tu trabajo y tu comprensión.
Pull Request #39 - ■ Manejo de Sesión Expirada y ■ Sistema Completo de Gestión de Eventos
Estado: closed
Asignado a: No asignado
Creado: 2025-05-25T07:12:23Z
Cerrado: 2025-05-28T07:48:57Z
Descripción
■ Interceptor Global de Sesión: Manejo automático de tokens expirados con redirección y limpieza de estado
■ Sistema Completo CRUD de Eventos: Formularios avanzados para crear/editar eventos con subida de imágenes
■ 1. Manejo de Sesión Expirada
Implementación del Interceptor Axios
■ Interceptor de respuestas global en axiosConfig.js
■ Detección automática de errores 401 y mensajes "■ Token inválido"
■ Redirección automática a /login al detectar sesión expirada
■ Limpieza completa del estado: usuario y token removidos del contexto y localStorage
■ Mensaje informativo: toast.error("Tu sesión ha expirado, por favor inicia sesión nuevamente")
Validación Temprana
■ Verificación al inicio de la aplicación antes del acceso a rutas protegidas
■ Integración con AuthContext para manejo centralizado del estado de autenticación
Archivos Modificados:
■■■ frontend/src/api/axiosConfig.js (Interceptor implementado)
■■■ frontend/src/context/AuthContext.jsx (Función logout expuesta)
■■■ frontend/src/middlewares/auth.js (Respuestas 401 estandarizadas)
Sistema Completo de Gestión de Eventos
Formulario Unificado de Eventos
■ Componente reutilizable EventForm.jsx para crear/editar
■ Lógica condicional clara basada en el prop mode (create/edit)
■ Precarga automática de datos al editar eventos existentes
Campos Implementados:

■ Información Básica:
Nombre del evento
Descripción
Tipo (CURSO, WEBINAR, CONGRESO, CHARLA, SOCIALIZACIÓN, PÚBLICO)
Fechas de inicio y fin
Duración total (en horas)
■ Campos Específicos de Curso:
Nota mínima (0-10)
Porcentaje mínimo de asistencia (0-100%)
Solo aparecen cuando tipo = "CURSO"
■ Información Adicional:
Requisitos
Modalidad (Presencial, Virtual, Híbrida)
Público objetivo
Carrera asociada (opcional)
Evento de pago (checkbox)
■ Imagen de Portada:
Upload con drag & drop
Preview en tiempo real
Validación de tipos (JPG, PNG, WEBP)
Máximo 5MB
Funcionalidades del Sistema:
■ Consideraciones Visuales:
■ Distinción visual entre eventos tipo CURSO y eventos generales
■ Campos condicionales que aparecen/desaparecen según el tipo seleccionado
■ Feedback visual completo con toast.success y toast.error
■ Preview de imagen antes de subir
■ Estados de carga en formularios
■ Interfaz Moderna:
■ Diseño responsive que funciona en móviles y escritorio
■ Iconografía consistente con Lucide React
■ Animaciones suaves y transiciones
■ Validación en tiempo real con mensajes claros
Rutas y Navegación:
■ /admin/eventos/crear → Crear nuevo evento
■ /admin/eventos/editar/:id → Editar evento existente
■ /admin/eventos → Gestión con vista previa de imágenes
Backend API:
■ Endpoints completos para CRUD de eventos
■ Upload de imágenes con middleware multer
■ Validaciones robustas en controlador
■ Campos nuevos en BD: requisitos, modalidad, publico_objetivo, imagen_portada
■ Archivos Nuevos/Modificados
Frontend:
■ NUEVOS:
src/components/EventForm.jsx
src/components/styles/EventForm.css
src/views/admin/CreateEvent.jsx
src/views/admin/EditEvent.jsx
■ MODIFICADOS:
src/views/admin/AdminEvents.jsx (Vista de gestión con imágenes)
src/routes/EventsRoute.jsx (Mostrar eventos con imágenes)
src/api/axiosConfig.js (Interceptor de sesión)

src/context/AuthContext.jsx (Integración logout)
src/App.jsx (Nuevas rutas)
Backend:
■ MODIFICADOS:
src/controllers/evento.controller.js (CRUD completo)
src/routes/evento.routes.js (Upload de imágenes)
src/middlewares/auth.js (Respuestas 401 consistentes)
prisma/schema.prisma (Campos nuevos)
■ Flujo de Trabajo Completado
Creación de Eventos:
Admin accede a /admin/eventos/crear
Completa formulario con todos los campos
Sube imagen de portada (opcional)
Sistema valida y guarda
Redirección a lista de eventos
Edición de Eventos:
Admin hace clic en "Editar" desde la lista
Formulario se precarga con datos existentes
Permite modificar cualquier campo
Mantiene imagen existente o permite cambiarla
Actualización exitosa con feedback
Manejo de Sesión:
Usuario navega normalmente
Token expira o se vuelve inválido
Interceptor detecta error 401 automáticamente
Limpia estado y redirige a login
Muestra mensaje informativo
Commits Vinculados
Merge pull request #39 from Andriu-Dex/feature/role-management
Autores de Commits
Andriu
Comentarios
Gabriel-Spartan: # Resumen del Estado Actual del Proyecto AcademicEvents
1. Gestión de Eventos
CRUD de eventos implementado:
Crear, modificar (incluyendo imagen) y obtener (por ID o tipo).
Soporte completo para campos obligatorios y validaciones específicas según el tipo de evento (CURSO, CHARLA, etc.).
Manejo de campos exclusivos de cursos en una tabla relacionada (evento_curso) y sincronización de datos al editar o cambiar tipo.
Subida de imágenes a AWS S3 al crear o editar un evento:
Al editar, si se sube una nueva imagen, elimina la anterior de S3 (excepto si es la imagen por defecto).
Validaciones estrictas: Solo se actualiza el registro si todos los datos son válidos.
Si se cambia de tipo (por ejemplo, de CHARLA a CURSO), se crea el registro en evento_curso. Si pasa de CURSO a otro tipo, se elimina ese registro.
Validación y conversión de tipos de datos para fechas (Date) y valores numéricos.
Errores de Prisma gestionados: manejo de errores de constraint y validaciones de FK.
2. Estructura y Buenas Prácticas
Código modular: separa controllers, middlewares y configuración de AWS S3.
Uso de middlewares para autenticación, autorización y manejo de archivos (multer + multer-s3).
El backend está listo para nuevas rutas protegidas y funciones de administración.
Frontend:
Formularios de registro y login funcionales, validados y con UX amigable.

Registro consulta carreras y envía correctamente el ID.
Login funcional, maneja sesión y redirección según rol.
En resumen
Puedes crear, editar y obtener eventos con sus imágenes, cambiando incluso el tipo de evento y gestionando su información relacionada.
Registro y login funcionan bien, con validación de datos, encriptación y relación correcta con carrera.
El sistema maneja bien los archivos y la lógica de imágenes en S3.
La arquitectura está preparada para crecer y agregar panel de administración, inscripción de estudiantes a eventos, reportes, validaciones adicionales y más.
Gabriel-Spartan: ## ■ Actualizaciones Realizadas
■ Backend: Se actualizó la lógica para gestionar eventos según su tipo (por ejemplo, CURSO) y la relación con eventos_curso.
■■ Base de Datos: Se corrigió la inserción y actualización de eventos, validando y creando las relaciones necesarias entre evento y eventos_curso. Se modificó el modelo de la base de datos conforme a los últimos requerimientos.
■■ Frontend: Se mejoró la gestión de datos numéricos como dur_hor_eve al crear y editar eventos, asegurando que toda la información se procese correctamente.
■ CRUD Completo: Ahora el CRUD de eventos está finalizado, y tanto el frontend como el backend están totalmente sincronizados.
■■ Importante: Como se está utilizando Prisma, es necesario resetear la base de datos para obtener la última versión de la estructura y los datos de prueba.
■ Pasos para sincronizar la base de datos
Ejecuta estos comandos en el directorio raíz del backend:
```
npx prisma migrate reset
npx prisma migrate dev --name init
node prisma/seed.js
```
El primer comando resetea y actualiza la base de datos según el último esquema de Prisma.
El segundo comando aplica la migración inicial si es la primera vez que configuras la base de datos.
El tercer comando carga datos de prueba: crea un usuario ADMIN, un usuario GENERAL, varias carreras y un evento de ejemplo.
Pull Request #38 - Mejora visual y separación de estilos CSS por vista (responsive + modular)
Estado: closed
Asignado a: No asignado
Creado: 2025-05-24T20:00:21Z
Cerrado: 2025-05-26T00:51:17Z
Descripción
Se crearon hojas de estilo específicas para cada vista, siguiendo un enfoque modular:
CertificatesRoute.css – Estilos personalizados para la ruta de certificados.
EventsRoute.css – Estilos para la visualización de eventos públicos.
MyInscriptions.css – Mejora visual y estructura de la sección “Mis inscripciones”.
AdminCarreras.css – Estilos exclusivos para la administración de carreras.
AdminConfiguracion.css – Estilos para configuración institucional (misión, visión, autoridades).
AdminEvents.css – Mejora del diseño en la gestión de eventos.
AdminEventInscription.css – Estilos para la visualización y revisión de inscripciones por evento.
AdminInscripciones.css – Interfaz limpia para el listado general de inscripciones.
■ Diseño responsive
Se aplicaron técnicas para que las vistas se adapten correctamente a pantallas pequeñas (móviles y tablets) mediante:
Uso de media queries y estructuras flex/grid
Mejor espaciado y tamaños de fuente adaptativos
Aseguramiento de botones visibles y accesibles en pantallas reducidas

■ Mejoras visuales
Colores coherentes con el branding institucional
Contrastes mejorados para accesibilidad
Separación clara entre secciones, títulos jerarquizados
Estilos de botones más modernos e intuitivos
■ Organización del código
Eliminación de estilos embebidos innecesarios
Limpieza de duplicaciones y clases obsoletas
Separación de lógica funcional (JSX) de estilos (CSS) para mejor mantenibilidad
■ Impacto positivo
Mejora notable en la presentación del sistema para usuarios finales y administradores
Estructura de código más limpia, organizada y escalable
Base sólida para futuras mejoras de UI/UX
Fix #35
Commits Vinculados
Merge pull request #38 from Andriu-Dex/feature/mejoraEstructuraYSeparacionLogica
Autores de Commits
Andriu
Comentarios
Sin comentarios
Pull Request #37 - DevelopMerge de rama develop a main – Consolidación de funcionalidades completas
Estado: closed
Asignado a: No asignado
Creado: 2025-05-22T22:22:21Z
Cerrado: 2025-05-22T22:23:14Z
Descripción
Este Pull Request integra a la rama `main` todas las funcionalidades desarrolladas y validadas hasta la fecha en `develop`. Se consolidan mejoras clave en diseño, lógica, seguridad y experiencia de usuario, cumpliendo con los requerimientos del sistema AcademicEvents.
■ Cambios incluidos
■ Seguridad y autenticación
Interceptor global Axios para manejar sesión expirada (401 o token inválido)
Limpieza automática del estado y redirección al login
Validación proactiva del token al cargar la app
Corrección en el backend: uso consistente de rol_usu en el token y respuesta JSON
Redirección por rol al iniciar sesión (/admin o /home)
■ Interfaz y navegación
Navbar dinámico que muestra opciones según el rol (ADMIN o ESTUDIANTE)
Nueva vista principal Home para estudiantes después del login
Mejora de navegación del administrador con enlaces integrados en Navbar
Rediseño visual del formulario de registro:
Validación visual (cédula, nombres, carrera)
Estilo responsivo con feedback visual e íconos
■ Administración
Vista AdminCarreras con creación, edición y eliminación dinámica
Vista AdminConfiguracion para editar misión, visión y autoridades
Vista AdminInscripciones con:
Filtrado por evento
Visualización del comprobante enviado
Nota final y asistencia (si aplica)
Botones para aceptar, rechazar y finalizar inscripciones
Validación académica obligatoria (nota ≥ 8, asistencia ≥ 80%) para finalizar

■■ Gestión de eventos y cursos
Creación y edición de eventos y cursos desde interfaz admin
Campos dinámicos:
Mostrar/ocultar nota mínima y asistencia según tipo (CURSO)
Lógica de visualización condicional por evento.tip_eve:
Solo se muestran campos académicos si el tipo es CURSO
■ Reportes y certificados
El sistema permite:
Generar certificado si cumple criterios
Validar comprobante de pago (por depósito o transferencia)
Rechazar comprobante inválido y reenviar uno nuevo
Control completo del estado de inscripción desde el panel
■ Checklist antes del merge
[x] Todas las rutas protegidas funcionan según rol
[x] Los formularios muestran solo los campos relevantes según tipo de evento
[x] El login y la navegación funcionan correctamente
[x] Se validan correctamente los criterios académicos antes de finalizar
[x] No quedan mensajes de consola (console.log)
[x] Se mantiene consistencia en nombres (rol_usu) y rutas de importación (config/db)
[x] .env correctamente estructurado con VITE_API_URL
■ Observaciones finales
Este merge marca una **versión estable y completa del sistema, cumpliendo los requerimientos académicos, administrativos y de usabilidad.
Commits Vinculados
Merge pull request #37 from Andriu-Dex/develop
Autores de Commits
Andriu
Comentarios
Andriu-Dex: ■ Aprobado por el líder de proyecto
Issue #36 - Interceptor y manejo de sesión expirada + creación y edición de eventos y cursos desde el panel de administración
Estado: closed
Asignado a: NIXON-HS
Creado: 2025-05-22T10:17:37Z
Cerrado: 2025-05-28T07:49:39Z
Descripción
■ Manejo de sesión expirada
Implementar un **interceptor global de respuestas en Axios** para detectar tokens expirados o inválidos. Cuando el servidor devuelva un error **401** o un mensaje como "**■ Token inválido**", el sistema debe:
Redirigir automáticamente al usuario a la ruta /login
Limpiar el estado global (remover usuario y token del contexto o localStorage)
Mostrar el mensaje:
```ts
toast.error("Tu sesión ha expirado, por favor inicia sesión nuevamente");
```
Además, se debe incluir una **validación temprana al iniciar la aplicación**, que verifique la validez del token almacenado antes de permitir el acceso a rutas protegidas.
■■ Creación y edición de eventos/cursos
Actualizar el panel de administración para permitir al administrador:
Crear nuevos eventos o cursos
Editar eventos/cursos existentes
■ Consideraciones funcionales:

El formulario debe ser reutilizable (misma vista para crear o editar), con lógica condicional clara.
Precargar los datos del evento/curso si se está editando.
Guardar correctamente los siguientes campos:
Tipo (CURSO, WEBINAR, CONGRESO, etc.)
Nombre
Descripción
Fecha de inicio y fin
Duración total (en minutos u horas)
Nota mínima y porcentaje mínimo de asistencia (solo si el tipo es CURSO)
Requisitos
Modalidad
Público objetivo
Carrera(s) asociadas
■ Imagen de portada: agregar campo para subir una imagen al crear o editar el evento o curso. Esta imagen debe mostrarse luego junto con la información del evento/curso.
■ Consideraciones visuales:
Distinguir visualmente si se está gestionando un evento tipo CURSO o un evento general
Mostrar/ocultar los campos "nota mínima" y "asistencia mínima" dependiendo del tipo seleccionado
Proveer feedback visual al usuario mediante toast.success y toast.error para cada acción (creación/edición/errores)
■ Alcance:
El interceptor debe aplicarse a todas las solicitudes protegidas
Los formularios deben permitir crear y editar desde una única vista/component
Toda acción relevante debe tener retroalimentación visual clara
Asegurar el guardado correcto de la imagen de portada y su uso posterior en la UI
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #35 - Mejorar estructura de estilos en el frontend y separar lógica visual
Estado: closed
Asignado a: carlitosgiovanniramos
Creado: 2025-05-22T10:12:05Z
Cerrado: 2025-05-26T00:51:31Z
Descripción
El proyecto actualmente mezcla lógica funcional y estilos directamente en los componentes. Esta issue busca mejorar la organización del código separando los estilos CSS de cada vista (excepto login y registro) en archivos dedicados dentro de una carpeta `styles/`, favoreciendo la legibilidad y mantenibilidad del proyecto.
■ Detalles a implementar:
Crear un archivo .css por cada vista en donde corresponda, por ejemplo para el Registro se creó: frontend/src/views/styles/Register.css. Esto se implementa en vistas como MyInscriptions.css, AdminEvents.css, etc.).
Modificar los archivos .jsx según corresponda para dividir la logica de los estilos y moverla al archivo .css que se va a crear; esto debido a que algunas vistas incluyen los estilos dentro del mismo .jsx.
Conservar nombres de clase coherentes y descriptivos para facilitar su rastreo.
Importar correctamente los estilos en cada componente.
■ Consideraciones:
No modificar la lógica de los componentes.
No tocar los componentes de Login o Register, que ya están optimizados y AdminDashboard o Home ya que su desarrollo y diseño se le asignará a otro miembro.

Evitar usar Tailwind directamente en el JSX durante esta tarea.
Utilizar la paleta de colores manejada en la UTA (Ej: Login y Registro).
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #34 - Lógica condicional de visualización por tipo de evento en inscripciones y administración
Estado: closed
Asignado a: titoma1500
Creado: 2025-05-22T09:50:05Z
Cerrado: 2025-05-31T04:02:20Z
Descripción
Actualmente, en las vistas de inscripción y administración, no existe una lógica visual clara que oculte o muestre elementos de la interfaz según el tipo de evento. Esto puede generar confusión en los usuarios, especialmente si se presentan campos que no aplican (como nota o asistencia en eventos tipo charla o webinar).
Esta mejora tiene como objetivo detectar dinámicamente si un evento es de tipo `CURSO` y, en base a ello, adaptar la interfaz mostrando solo los elementos relevantes.
■ Detalles a implementar:
Detectar dinámicamente el tipo de evento (evento.tip_eve)
Mostrar los siguientes elementos solo si el evento es tipo CURSO:
Campo de nota final
Campo de asistencia (%)
Botón para finalizar inscripción
Ocultar los elementos anteriores si el evento es de tipo diferente (CHARLA, CONGRESO, WEBINAR, etc.)
■ Consideraciones:
Esta lógica debe aplicarse tanto en la vista de inscripción del estudiante como en la vista de administración, si corresponde.
Priorizar componentes reutilizables para evitar duplicación de código.
La validación visual debe ser clara, consistente y alineada con el diseño actual.
No se debe modificar el backend, solo aplicar lógica condicional en el frontend.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #33 - Mejora de navegación del administrador y rediseño del Dashboard + creación de vista Home para estudiante
Estado: closed
Asignado a: M4yb33
Creado: 2025-05-22T09:48:04Z
Cerrado: 2025-05-31T02:35:12Z
Descripción
Actualmente, al iniciar sesión como administrador, se redirige al `AdminDashboard`, el cual contiene enlaces a las vistas del panel administrativo. Sin embargo, esto debería integrarse en el Navbar `(frontend/src/components/Navbar.jsx)`, permitiendo una experiencia uniforme y accesible desde cualquier sección.
Además, se ha identificado que al iniciar sesión como estudiante, la navegación lleva directamente a la vista de eventos. Para mejorar la experiencia de usuario, se debe crear e integrar una vista de

Home como punto de entrada visual, moderna y acorde al tema.
■ Sugerencias a implementar:
■ Navegación del administrador:
Quitar los enlaces de navegación del AdminDashboard.
Rediseñar AdminDashboard para que muestre:
Resumen visual del sistema
KPIs (cantidad de eventos, inscripciones, certificados emitidos)
Últimos eventos creados
Integrar las redirecciones borradas del AdminDashboard directamente en el Navbar:
Validar inscripciones
Crear/editar eventos
Gestión de carreras
Configuración institucional
Mostrar las opciones de redireccionamiento teniendo en cuenta el rol del usuario :
ADMIN: Accesos de gestión
ESTUDIANTE: Vista de usuario
(Actualmente ya está implementado, por lo que se debe tener cuidado al hacer modificaciones en el Navbar)
■ Vista Home para estudiante:
Crear una nueva vista HomeEstudiante.jsx o Home.jsx.
Esta vista debe:
Ser el punto de entrada principal después del login para estudiantes
Incluir el Navbar
Tener diseño llamativo, moderno y temático (colores institucionales)
Mostrar contenido como:
Bienvenida personalizada
Eventos destacados
Acceso directo a certificados o inscripciones
■ Consideraciones:
Todo cambio debe ser responsivo y accesible.
La lógica de backend no debe ser alterada.
No debe interferir con la navegación y flujos actuales del sistema.
* El Navbar debe funcionar correctamente para ambos roles sin duplicar código.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #32 - Normalizar y estandarizar la base de datos
Estado: closed
Asignado a: Gabriel-Spartan
Creado: 2025-05-22T09:25:09Z
Cerrado: 2025-05-28T07:25:26Z
Descripción
Actualmente, la base de datos presenta una mezcla de convenciones, con columnas abreviadas como `id_usu` y otras completas como `comprobante`. Esta tarea busca normalizar los nombres de todas las tablas y columnas, completando los campos que falten, asegurando consistencia, claridad y mantenibilidad en los modelos Prisma y en la base de datos.
■ Detalles a implementar:
Establecer una convención clara y coherente.
Revisar todas las tablas y columnas existentes en los modelos Prisma.
Completar campos faltantes necesarios para el funcionamiento del sistema.

Unificar formatos y estructuras según las buenas prácticas de diseño relacional y de Prisma.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #31 - Mejoras completas en registro, inscripción, navegación, administración y seguridad
Estado: closed
Asignado a: No asignado
Creado: 2025-05-20T13:57:46Z
Cerrado: 2025-05-22T22:07:47Z
Descripción
Esta Pull Request integra múltiples mejoras visuales, funcionales y estructurales en el sistema **AcademicEvents**, cubriendo tanto la experiencia del usuario estudiante como la del administrador.
■ INTERFAZ Y EXPERIENCIA DE USUARIO
Rediseño del formulario de registro con estilo moderno, íconos lucide-react, feedback visual, validaciones estrictas y uso de select dinámico para carreras.
Validaciones integradas:
Cédula: exactamente 10 dígitos
Nombres y apellidos: solo letras
Carreras: opción seleccionable desde la base de datos
Navegación protegida y diferenciada por roles:
Estudiantes redirigidos a /eventos
Administradores redirigidos a /admin
Protección robusta de rutas administrativas con PrivateRouteAdmin.
■ ADMINISTRACIÓN DE CARRERAS Y CONFIGURACIÓN
/admin/carreras
Crear, editar y eliminar carreras con confirmación
/admin/configuracion
Gestión de misión, visión y autoridades institucionales desde la interfaz
Cambios persistentes en la base de datos (configuracion con id_conf = 1)
■ GESTIÓN DE INSCRIPCIONES – Admin
Panel completo en /admin/inscripciones con:
Filtro por evento
Visualización de nombre, correo, comprobante (PDF o imagen), nota, asistencia
Botones para aceptar, rechazar o finalizar inscripciones
Finalización de cursos:
Valida que se haya ingresado asistencia y nota_final
Verifica si se cumple mínimo: nota ≥ 8, asistencia ≥ 80%
Muestra errores si no cumple requisitos
■ SEGURIDAD Y BACKEND
Corrección en la generación y validación del JWT:
rol_usu ahora se incluye correctamente en el token y en la respuesta
Rutas reorganizadas:
Todas las rutas de inscripciones protegidas y organizadas
Rutas admin/inscripciones/validar/:id y admin/inscripciones/evento/:id funcionando con middleware de autorización
Manejo de expiración del token pendiente de mejora futura (actualmente no muestra alerta)
■ PRUEBAS RECOMENDADAS
Crear un archivo .env dentro del frontend con la línea:
```env
VITE_API_URL=http://localhost:3000
```

```
O el puerto que se utilice.
(Facilita los cambios de entorno sin modificar el código).
Dentro de la carpeta backend olocar los comandos:
```bash
`npx prisma db push
npm run seed`
```
Crear un usuario con el rol ADMIN desde la base de datos:
```sql
`INSERT INTO usuario (
id_usu, ced_usu, nom_usu, ape_usu, cor,_usu, con_usu, cel_usu, rol_usu, fec_cre_usu, comprobante,
carrera
) VALUES (
'uuid', -- ID único en formato UUID
'cedula', -- Número de cédula del usuario
'nombres', -- Nombres del usuario
'apellidos', -- Apellidos del usuario
'correo_electronico', -- Correo electrónico institucional o personal
'contrasena_encriptada', -- Contraseña encriptada (hash bcrypt u otro)
'numero_celular', -- Número de celular del usuario
'ADMIN', -- Rol del usuario (por ejemplo: ADMIN, ESTUDIANTE, DOCENTE)
NOW(), -- Fecha y hora de creación automática
NULL, -- Carrera (puede ser NULL para ADMIN)
NULL -- Archivo adjunto (puede ser NULL si no aplica)
);
`
```
Probar:
Registro y login
Inscripción a eventos con carga de comprobante
Verificación de comprobantes
Iniciar sesión como ADMIN
Probar:
Creación de carreras y edición de configuración institucional
Visualización y validación de inscripciones
■ Checklist final
[x] Validaciones frontend y backend funcionando correctamente
[x] Flujo de login con roles funcionando y protegidos
[x] Administración de carreras y configuración completa
[x] Finalización de inscripciones con lógica validada
[x] Visualización de comprobantes en admin
[x] Rutas reorganizadas y protegidas
[x] Sin logs de consola ni errores visibles
Closes #30
Commits Vinculados
Merge pull request #31 from Andriu-Dex/feature/visual-improvements-multiview
Autores de Commits
Gabriel Llerena
Comentarios
Sin comentarios
Issue #30 - Mejoras visuales y funcionales para registro, inscripción, eventos y navegación
Estado: closed
```

Asignado a: Andriu-Dex
Creado: 2025-05-18T19:41:41Z
Cerrado: 2025-05-23T03:43:14Z
Descripción
■ Objetivo:
Mejorar la experiencia de usuario en varias áreas clave del sistema AcademicEvents, aplicando rediseños visuales y ajustes funcionales que ya han sido identificados como necesarios en el MVP, el registro y la inscripción. Esta tarea prepara al sistema para una presentación o fase más estable.
■ Tareas a implementar:
■ Diseño visual
[ ] Rediseñar el formulario de registro de estudiantes (márgenes, distribución responsiva, iconos de campos, botones claros, feedback visual)
[ ] Rediseñar la interfaz de inscripción a eventos:
Separar mejor el modal
Mostrar archivo seleccionado
Mejorar títulos, botones y colores
[ ] Estilizar los eventos/cursos disponibles en tarjetas visuales atractivas con:
Etiquetas como “PAGADO” o “GRATUITO”
Fechas y horas bien distribuidas
[ ] Mostrar mensaje en la vista de eventos cuando ya se está inscrito a todos
■■ Funcionalidad extra
[ ] Agregar validación visual (frontend) a todos los campos de registro
[ ] Mostrar correctamente el nombre del archivo una vez seleccionado
[ ] En la inscripción: evitar que se suba archivo si no es válido antes del envío
[ ] Mostrar notificación cuando se suba correctamente el comprobante (feedback)
[ ] Validar longitud mínima del número de cédula (10 dígitos)
[ ] Eliminar visualmente los eventos ya pasados (según fec_fin_eve)
[ ] Mostrar estado de las inscripciones en el perfil: PENDIENTE, ACEPTADA, FINALIZADA, etc.
■ Navegación y estructura
[ ] Redirigir automáticamente desde / a /login
[ ] Mostrar enlaces condicionales en el navbar según el rol (ESTUDIANTE vs ADMIN)
[ ] Añadir enlace a "Mis cursos" en la vista de estudiante (rutas futuras)
■ Resultado esperado:
Experiencia visual limpia, moderna y clara
Formularios más usables y guiados
Inscripción con mejor control visual de errores y estados
Eventos mostrados con claridad y estética
Navegación coherente según el rol
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #29 - Registro de usuarios estudiantes con validación de correo institucional y comprobante
Estado: closed
Asignado a: No asignado
Creado: 2025-05-18T19:36:03Z
Cerrado: 2025-05-18T19:56:02Z
Descripción
Este Pull Request implementa el flujo completo de registro de estudiantes en el sistema AcademicEvents. Incluye validación de correo institucional, carga de un documento PDF o imagen,

encriptación de la contraseña, y guardado seguro en la base de datos.
■ Cambios implementados
■ Backend
Ruta POST /api/registro
Controlador registrarEstudiante
Encriptación de contraseña con bcryptjs
Validación:
Correo institucional (@uta.edu.ec)
Contraseña mínima de 6 caracteres
Cédula y correo únicos
Carga de archivo obligatorio (.pdf, .jpg, .png, .webp)
Almacenamiento del nombre del archivo y carrera en el modelo usuario
■ Frontend
Nueva vista Register.jsx
Formulario con campos: cédula, nombres, apellidos, correo, contraseña, celular, carrera
Carga de archivo mediante FormData
Validaciones de campos obligatorios
Mensajes con toast
Redirección a /login tras registro exitoso
■ Cómo probar
Accede a http://localhost:5173/registro
Completa los campos con datos válidos
Usa un correo institucional (@uta.edu.ec)
Adjunta un archivo válido (.pdf o imagen)
Haz clic en "Registrarse"
El usuario debe crearse y redirigir al login
■ Archivos afectados
```plaintext
frontend/
■■■ src/
■■■ views/
■ ■■■ Register.jsx
■■■ App.jsx
backend/
■■■ src/
■■■ controllers/
■ ■■■ auth.controller.js
■■■ routes/
■ ■■■ auth.routes.js
■■■ middlewares/
■ ■■■ upload.js
prisma/
■■■ schema.prisma (modelo usuario actualizado)
```
Este PR entrega un módulo funcional y validado para el registro de estudiantes. Se deja pendiente la mejora visual para una futura Issue dedicada al refinamiento del diseño de formularios y pantallas.
Fix #28
Commits Vinculados
Merge pull request #29 from Andriu-Dex/feature/student-registration
Autores de Commits
Andriu
Comentarios
Sin comentarios

Issue #28 - Registro de nuevos estudiantes con validación institucional y carga de documentos
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-18T07:35:26Z
Cerrado: 2025-05-18T19:56:12Z
Descripción
■ Objetivo:
Implementar el sistema de registro para usuarios tipo `ESTUDIANTE`, permitiendo que se registren desde la web con su correo institucional, completen sus datos personales, adjunten documentación y creen una cuenta con acceso al sistema.
■ Tareas a implementar:
■ Backend
[ ] Ruta POST /api/registro
[ ] Validar que el correo termine en @uta.edu.ec
[ ] Validar que no esté repetido (por cor_usu y ced_usu)
[ ] Hashear la contraseña con bcryptjs
[ ] Guardar como rol_usu = ESTUDIANTE
[ ] Aceptar archivo PDF (PDF de matrícula, certificado, etc.)
[ ] Guardar nombre del archivo en campo comprobante o nuevo
■■ Frontend
[ ] Crear nueva vista: src/views/Register.jsx
[ ] Campos: cédula, nombres, apellidos, correo, contraseña, carrera (seleccionable), carga PDF
[ ] Validaciones en el frontend (campos obligatorios, longitud, formato)
[ ] Toast para éxito y errores
[ ] Redirección a /login tras registrarse
■ Seguridad extra
[ ] Validar que la contraseña tenga mínimo 6 caracteres
[ ] Prevenir correos duplicados y cédulas repetidas
[ ] Validar tipo y tamaño de archivo PDF
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #27 - Vista de eventos e inscripción de estudiantes con validación, protección y carga de requisitos
Estado: closed
Asignado a: No asignado
Creado: 2025-05-18T07:29:22Z
Cerrado: 2025-05-18T16:47:26Z
Descripción
Este Pull Request implementa la funcionalidad completa para que los estudiantes puedan visualizar eventos académicos disponibles, inscribirse adjuntando documentos de requisitos, y garantizar que no puedan repetir inscripciones.
■ Tareas completadas:
■■■ Frontend (/eventos)
Vista protegida para estudiantes autenticados (/eventos)
Obtención dinámica de eventos desde el backend
Filtros por nombre del evento
Excluir eventos ya inscritos del listado
Modal para inscripción con carga de archivo (PDF o imagen)
Validación del archivo (tipo y tamaño)

Inscripción vía POST /api/inscripciones con FormData
Feedback con react-toastify
■ Backend
Verificación de campos requeridos (id_usu, id_eve)
Validación de archivo adjunto: tipos MIME y tamaño (máx. 5MB)
Comprobación de existencia de usuario y evento
Rechazo de inscripciones duplicadas:
Lógica: findFirst()
Base de datos: @@unique([id_usu, id_eve]) en schema.prisma
Captura del error P2002 en Prisma
Guardado del nombre de archivo en campo comprobante
Respuestas claras para errores y éxito
■ ¿Cómo probar?
Iniciar sesión como estudiante (@uta.edu.ec)
Acceder a http://localhost:5173/eventos
Filtrar y ver eventos disponibles
Hacer clic en "Inscribirme"
Adjuntar un archivo PDF o imagen válido
Enviar inscripción
Ver mensaje de éxito y comprobar:
Evento ya no aparece en la lista
Comprobante guardado en carpeta /uploads
Registro en tabla inscripcion con estado PENDIENTE
Probar envío repetido: debe rechazar con mensaje
■ Archivos modificados
```plaintext
frontend/
■■■ src/
■■■ routes/EventsRoute.jsx
■■■ components/Navbar.jsx
backend/
■■■ src/
■■■ routes/inscripcion.routes.js
■■■ controllers/inscripcion.controller.js
■■■ middlewares/upload.js
■■■ prisma/schema.prisma
```
■■ Resultado
Con esta entrega, el sistema cuenta con un flujo completo para gestionar inscripciones por parte del estudiante, garantizando integridad de datos, validaciones robustas y experiencia de usuario fluida.
Fix #26
Commits Vinculados
Merge pull request #27 from Andriu-Dex/feature/events-student-registration
Autores de Commits
Andriu
Comentarios
Sin comentarios
Issue #26 - Vista de eventos disponibles e inscripción con validación de requisitos
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-18T05:22:49Z
Cerrado: 2025-05-18T19:33:40Z
Descripción

■ Objetivo:
Crear la vista principal del estudiante donde pueda:
Visualizar los eventos académicos disponibles
Filtrarlos por nombre, tipo, carrera, etc.
Consultar los detalles de un evento
Inscribirse, adjuntando los documentos requeridos
Ver el estado de su inscripción (pendiente, aceptada, etc.)
■ Tareas a implementar:
■ Frontend
[ ] Crear nueva ruta protegida /eventos
[ ] Obtener eventos disponibles desde backend (GET /api/eventos)
[ ] Mostrar listado en tarjetas o tabla (solo eventos activos y no finalizados)
[ ] Implementar filtro por nombre, tipo, carrera
[ ] Mostrar detalles al hacer clic: fecha, duración, pagado, requisitos
[ ] Si el evento requiere requisitos:
Formulario para adjuntar PDF con requisitos (cédula, matrícula, carta motivación)
[ ] Botón de inscripción (llama a POST /api/inscripciones)
[ ] Mostrar estado de inscripción después de enviar
[ ] Validar que no se pueda inscribir dos veces
■ Backend (si es necesario)
[ ] Asegurar que GET /api/eventos solo retorne eventos activos y no finalizados
[ ] Validar que POST /api/inscripciones impida duplicados
[ ] Aceptar y guardar los documentos en /uploads
■ Acceso
Solo estudiantes autenticados (verificarToken)
Sección visible desde el menú lateral
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #25 - Vista de certificados para estudiantes + autenticación real conectada al backend
Estado: closed
Asignado a: No asignado
Creado: 2025-05-18T05:04:35Z
Cerrado: 2025-05-18T05:15:52Z
Descripción
Este Pull Request implementa las funcionalidades completas relacionadas con el acceso seguro a certificados académicos por parte del estudiante, así como la integración definitiva del formulario de login con el backend real mediante JWT.
■ Tareas cumplidas:
■ Autenticación real
Reemplazo del login simulado por un formulario conectado al backend (/api/login)
Validación de correo institucional @uta.edu.ec
Manejo de errores con react-toastify
Almacenamiento de usuario y token usando Context + localStorage
Redirección automática según el rol (ESTUDIANTE → /certificados)
■ Seguridad backend
Implementación del middleware verificarPropietarioCertificado
Protección de rutas:
GET /api/certificados/:id
POST /api/certificados/enviar/:id

Validación de que solo el usuario dueño de la inscripción pueda acceder a su certificado
■ Vista de certificados en el frontend
Nueva ruta: /certificados
Obtención de inscripciones finalizadas vía backend
Filtro de inscripciones elegibles para certificado
Mostrar tipo de evento, estado de envío, y botones de acción
Acciones:
Descargar PDF directamente desde el backend
Reenviar por correo al correo institucional
■ Estética y experiencia
Uso de íconos con lucide-react para mostrar estado de envío
Feedback con toast para éxito y errores
Sección de certificados visible solo para estudiantes
Redirección automática desde / a /login
■ ¿Cómo probar?
Iniciar sesión con correo @uta.edu.ec válido y contraseña
Ver redirección automática a /certificados
Probar descarga de certificados desde la tabla
Usar botón "Reenviar por correo"
Confirmar recepción del PDF en el correo institucional
Intentar acceder con token de otro usuario a certificado ajeno (debe fallar con 403)
■ Estructura afectada
```plaintext
frontend/
■■■ src/
■ ■■■ context/AuthContext.jsx
■ ■■■ hooks/useAuth.js
■ ■■■ routes/CertificatesRoute.jsx
■ ■■■ views/Login.jsx ← rediseñado y conectado
■ ■■■ App.jsx
backend/
■■■ src/
■ ■■■ routes/certificado.routes.js
■ ■■■ middlewares/verificarPropietarioCertificado.js
■ ■■■ controllers/certificado.controller.js
```
■ Resultado
El sistema ahora cuenta con un flujo de acceso autenticado completamente funcional y seguro, con experiencia de usuario fluida y validación de roles, protección por token y gestión segura de certificados.
Closes #23
Commits Vinculados
Merge pull request #25 from Andriu-Dex/feature/frontend-certificates-protected
Autores de Commits
Andriu
Comentarios
Sin comentarios
Pull Request #24 - Merge inicial desde develop: estructura base, autenticación y MVP funcional
Estado: closed
Asignado a: No asignado
Creado: 2025-05-18T01:10:44Z
Cerrado: 2025-05-18T01:14:46Z
Descripción

Este Pull Request integra a la rama main todos los avances realizados hasta la fecha 17/05/2025 en develop, consolidando la estructura base del proyecto y el MVP funcional completo del sistema AcademicEvents.
■ Contenido del merge:
■ Estructura del proyecto:
Separación clara en carpetas frontend/ y backend/
Configuración de Vite, React, Tailwind (frontend)
Configuración de Express, Prisma, PostgreSQL (backend)
Archivos base .env, .gitignore, prisma/schema.prisma
■ Autenticación:
Login para usuarios con correo institucional @uta.edu.ec
Middleware auth.js con JWT
Validación de roles ADMIN y ESTUDIANTE
■ Gestión de eventos e inscripciones:
CRUD completo de eventos académicos
Inscripción de estudiantes con validación de duplicados
Subida de comprobantes con multer
Validación de asistencia y nota final (solo en cursos)
■ Certificados académicos:
Verificación de requisitos para generar certificados
Generación de PDF con pdfkit
Envío automático por correo con nodemailer
Separación de responsabilidades: descarga vs. envío
■ Estado del sistema:
Sistema estable, probado con Postman
Base de datos sincronizada con Prisma
Tokens funcionales y rutas protegidas
Este merge establece la base operativa del proyecto en main y marca el fin de la fase MVP.
Commits Vinculados
Merge pull request #24 from Andriu-Dex/develop
Autores de Commits
Andriu
Comentarios
Andriu-Dex: ■ Aprobado por el líder de proyecto
Issue #23 - Vista de certificados para estudiantes con protección de rutas y funciones de descarga y reenvío
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-18T00:55:19Z
Cerrado: 2025-05-18T05:17:07Z
Descripción
Objetivo:
Implementar la vista completa de gestión de certificados para el usuario ESTUDIANTE, permitiendo consultar eventos aprobados, descargar el certificado en PDF y reenviarlo por correo. Además, proteger las rutas sensibles en el backend para evitar accesos indebidos.
Tareas a realizar:
■ Backend
Crear middleware verificarPropietarioCertificado
Aplicarlo en:
GET /certificados/:id
POST /certificados/enviar/:id
■ Frontend
Crear vista /certificados

Obtener inscripciones finalizadas desde backend
Filtrar las elegibles para certificado
Mostrar nombre del evento, fecha, tipo y estado
Agregar botón "Descargar PDF" (abre nueva pestaña)
Agregar botón "Enviar por correo"
Mostrar ícono si ya fue enviado (check o etiqueta)
Notificación de éxito/error por acción
■ Navegación
Redireccionar / al login si no ha iniciado sesión
Añadir botón de navegación "Certificados" en el menú de estudiante
Mensaje si no hay certificados aún
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #22 - MVP funcional: eventos, inscripciones, certificados y flujo inicial
Estado: closed
Asignado a: No asignado
Creado: 2025-05-18T00:44:38Z
Cerrado: 2025-05-18T01:06:52Z
Descripción
Este Pull Request implementa el prototipo funcional mínimo (MVP) del sistema AcademicEvents, permitiendo:
Gestión de eventos académicos
Inscripción de estudiantes a eventos
Validación administrativa
Registro de asistencia y nota
Generación de certificados académicos
Envío automatizado por correo
■ Cambios implementados:
■ Modelos (Prisma)
evento: con tipificación, fechas, pago, horas, nota mínima, etc.
inscripcion: con usuario, evento, asistencia, nota, comprobante y certificado
Nuevos enums: tipo_evento, estado_inscripcion
Campo cert_enviado añadido a inscripcion
■ Lógica backend
CRUD completo de evento
Rutas para inscribirse y validar inscripción
Reglas para asistencia y nota mínima en cursos
Verificación de elegibilidad para certificado
Generación de certificado en PDF con pdfkit
Envío por correo con nodemailer y control de envío
■ Rutas expuestas
Método	Ruta	Acción
GET	/api/eventos	Obtener todos los eventos
POST	/api/inscripciones	Inscribir a un evento
PUT	/api/inscripciones/validar/:id	Validar estado, asistencia y nota
GET	/api/inscripciones/:id	Ver inscripciones del usuario
GET	/api/inscripciones/certificado/:id	Verificar si puede generar certificado
GET	/api/certificados/:id	Descargar certificado en PDF

| POST | /api/certificados/enviar/:id | Enviar certificado por correo electrónico |
■ Protecciones y validaciones
Verificación de duplicados en inscripción
Validación estricta de campos numéricos (nota, asistencia)
Condicionales por tipo de evento (CURSO vs. otros)
■ Estructura afectada
```plaintext
backend/
■■■ src/
■ ■■■ controllers/
■ ■ ■■■ evento.controller.js
■ ■ ■■■ inscripcion.controller.js
■ ■ ■■■ certificado.controller.js
■ ■■■ routes/
■ ■ ■■■ evento.routes.js
■ ■ ■■■ inscripcion.routes.js
■ ■ ■■■ certificado.routes.js
■ ■■■ services/
■ ■ ■■■ mailer.js
■ ■■■ utils/
■ ■ ■■■ certificado.utils.js
■ ■■■ app.js
■■■ prisma/
■ ■■■ schema.prisma
```
■ Probar Funcionamiento
1■■ Crear un evento (solo admin)
```http
POST /api/eventos
Content-Type: application/json
{
"nom_eve": "Curso de Python",
"tip_eve": "CURSO",
"fecinieve": "2025-06-01T08:00:00.000Z",
"fecfineve": "2025-06-10T17:00:00.000Z",
"durhrseve": 20,
"pagado_eve": true,
"notamineve": 8,
"porasisteve": 80,
"carreraId": "uuid-de-carrera"
}
```
2■■ Inscribir a un estudiante
```http
POST /api/inscripciones
Content-Type: application/json
{
"id_usu": "uuid-del-estudiante",
"id_eve": "uuid-del-evento",
"comprobante": "nombre-del-archivo.pdf"
}
```
3■■ Validar inscripción como admin

```http
PUT /api/inscripciones/validar/{id_ins}
Content-Type: application/json
{
"estado": "FINALIZADA",
"asistencia": 85,
"nota_final": 9
}
```
4■■ Verificar si puede generar certificado
```http
GET /api/inscripciones/certificado/{id_ins}
```
Respuesta esperada:
```json
{
"puedeGenerar": true,
"tipo": "APROBADO"
}
```
5■■ Descargar certificado (PDF)
```http
GET /api/certificados/{id_ins}
```
➡■ Se abre un PDF con el nombre del estudiante, el evento y su aprobación.
6■■ Enviar certificado por correo
```http
POST /api/certificados/enviar/{id_ins}
```
Respuesta esperada:
```json
{
"msg": "Certificado enviado correctamente por correo"
}
```
Y el correo llegará a la cuenta institucional del estudiante como archivo adjunto.
■ Resultado:
Sistema funcional y operativo para pruebas completas.
*Closes #21 *
Commits Vinculados
Merge pull request #22 from Andriu-Dex/feature/mvp-prototype
Autores de Commits
Andriu
Comentarios
Sin comentarios
Issue #21 - Implementar MVP funcional: eventos, inscripción y flujo inicial
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-17T17:35:41Z
Cerrado: 2025-05-18T04:52:22Z
Descripción
■ Objetivo:
Desarrollar un prototipo funcional mínimo (MVP) de la aplicación AcademicEvents, que permita:

Autenticación de usuarios tipo ESTUDIANTE
Gestión de eventos desde el backend
Inscripción de usuarios a eventos
Subida de comprobante de pago (ya funcional)
Registro de asistencia (modo simulado)
Generación condicional de certificado (simulado)
Interfaz básica con navegación (Login, Home, Eventos, Perfil)
■ Tareas estimadas:
■ Backend:
Crear modelo evento en Prisma
Crear modelo inscripcion (con referencia a usuario + evento)
CRUD básico de eventos (admin)
Endpoint POST /inscripciones con validaciones
Agregar relación evento → carrera
Ruta protegida para ver eventos disponibles según carrera
Simular validación de asistencia y notas
■ Frontend:
Página de inicio (Home) con eventos disponibles
Pantalla de detalles del evento
Formulario de inscripción con carga de comprobante
Vista de perfil con eventos inscritos y estado (pendiente, aprobado, etc.)
Mostrar si se generó certificado o no
■ Consideraciones:
El diseño puede ser básico por ahora, enfocado en funcionalidad
No es necesario aún enviar correos reales
Se deben usar datos reales desde la base con Prisma
Solo eventos activos deben mostrarse a los estudiantes
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #20 - Subida y validación de comprobantes de pago (archivo PDF o imagen)
Estado: closed
Asignado a: No asignado
Creado: 2025-05-17T17:04:41Z
Cerrado: 2025-05-17T17:15:59Z
Descripción
Este Pull Request implementa la capacidad para que los estudiantes puedan subir comprobantes de pago al inscribirse en cursos/eventos pagados.
■ Cambios realizados:
■ Se instaló multer para manejar carga de archivos en Express.
■ Se creó el middleware upload.js con:
Almacenamiento en /uploads
Filtro de tipos permitidos: .jpg, .jpeg, .png, .pdf
Límite de tamaño: 2MB
■ Se creó la ruta POST /api/comprobantes/subir en comprobante.routes.js
Usa upload.single("comprobante")
Devuelve nombre y ruta del archivo recibido
■ Ruta registrada en app.js
■ Cómo probar:
Endpoint: POST http://localhost:3000/api/comprobantes/subir

Tipo de cuerpo: form-data
Clave: comprobante
Tipo: archivo (.pdf, .png, etc.)
Respuesta esperada:
```json
{
"msg": "Archivo recibido correctamente",
"nombreArchivo": "nombre-timestamp.pdf",
"ruta": "/uploads/nombre-timestamp.pdf"
}
```
■ Consideraciones:
Los archivos se guardan localmente en backend/uploads/
La carpeta /uploads se incluye en .gitignore
Closes #9
Commits Vinculados
Merge pull request #20 from Andriu-Dex/feature/upload-receipt
Autores de Commits
Andriu
Comentarios
Sin comentarios
Pull Request #19 - Ruta de login funcional con autenticación JWT (solo estudiantes)
Estado: closed
Asignado a: No asignado
Creado: 2025-05-17T05:20:13Z
Cerrado: 2025-05-17T06:26:12Z
Descripción
Este PR implementa la funcionalidad completa para la autenticación de usuarios del tipo ESTUDIANTE. El sistema valida el correo y la contraseña, verifica que el rol del usuario sea ESTUDIANTE, y devuelve un token JWT válido por 2 horas para acceder a rutas protegidas.
■ Cambios realizados:
■ Controlador auth.controller.js:
Busca al usuario por su correo institucional (cor_usu)
Verifica que el rol sea ESTUDIANTE
Valida la contraseña con bcryptjs
Firma un token JWT con jsonwebtoken (2h de validez)
■ Ruta agregada en src/routes/auth.routes.js:
```http
POST /api/login
```
■ Ruta registrada en src/app.js como parte del grupo /api
■ Requisitos previos para probarlo:
Estar conectado a la base de datos correcta:
```sql
\c academicevents
```
Crear un Hash de la contraseña a utilizar:
```bash
node
const bcrypt = require('bcryptjs');
bcrypt.hashSync('12345', 10);
```
Insertar manualmente un usuario tipo ESTUDIANTE con contraseña hasheada:

Ejemplo:
```sql
INSERT INTO usuario (
idusu, cedusu, nomusu, apeusu, corusu, conusu, celusu, rolusu, feccreusu
)
VALUES (
genrandomuuid(),
'la_cedula',
'Nombre',
'Apellido',
'usuario@uta.edu.ec',
'ContraseñaHashGenerada',
'Celular',
'ESTUDIANTE',
now()
);
```
■ Variables necesarias en .env:
```env
JWTSECRET=clavesegura
```
■ Cómo probar con Postman:
1. Ruta:
```http
POST http://localhost:3000/api/login
```
2. Headers:
```
Content-Type: application/json
```
3. Body (JSON):
```json
{
"correo": "sparedes7182@uta.edu.ec",
"contrasena": "12345"
}
```
4. Respuesta esperada:
```json
{
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"usuario": {
"id": "uuid-generado",
"correo": "usuario@uta.edu.ec",
"rol": "ESTUDIANTE"
}
}
```
■ Resultado:
Autenticación lista para conectarse con el middleware auth.js y proteger rutas para estudiantes autenticados.
Close #10
Commits Vinculados

Merge pull request #19 from Andriu-Dex/feature/auth-middleware
Autores de Commits
Andriu
Comentarios
Sin comentarios
Pull Request #18 - CRUD de carreras (modelo Prisma, controladores y rutas REST)
Estado: closed
Asignado a: No asignado
Creado: 2025-05-16T22:18:30Z
Cerrado: 2025-05-16T23:59:23Z
Descripción
Este PR implementa la funcionalidad completa para la gestión de carreras en el sistema AcademicEvents, utilizando Prisma ORM, controladores modulares y rutas REST.
■ Cambios realizados:
■ Modelo Prisma carrera creado en prisma/schema.prisma:
prisma
model carrera {
id_car String @id @default(uuid())
nom_car String @unique
est_car Boolean @default(true)
feccrecar DateTime @default(now())
}
■ Migración aplicada con npx prisma db push
■ Controlador carrera.controller.js:
crearCarrera: Crea una nueva carrera
obtenerCarreras: Devuelve todas las carreras
actualizarCarrera: Actualiza una carrera por ID
eliminarCarrera: Elimina una carrera por ID
■ Rutas REST definidas en carrera.routes.js, integradas al backend mediante /api/carreras
■■ Ruta conectada globalmente en src/app.js
■ Endpoints disponibles:
Método	Ruta	Acción
GET	/api/carreras	Obtener todas las carreras
POST	/api/carreras	Crear nueva carrera
PUT	/api/carreras/:id	Actualizar carrera
DELETE	/api/carreras/:id	Eliminar carrera
■ Cómo probar (Postman):
Crear carrera
POST http://localhost:3000/api/carreras
Body: { "nom_car": "Software" }
Obtener todas las carreras
GET http://localhost:3000/api/carreras
Actualizar carrera
PUT http://localhost:3000/api/carreras/{id_car}
Body: { "nom_car": "Industrial" }
Eliminar carrera
DELETE http://localhost:3000/api/carreras/{id_car}
■ Consideraciones:
Validación para evitar duplicados (nom_car)
Respuestas claras para cada acción
Control total sobre las carreras desde el backend administrativo
Closes #8

Commits Vinculados
Merge pull request #18 from Andriu-Dex/feature/CRUD_Carreras
Autores de Commits
Gabriel Llerena
Comentarios
Sin comentarios
Pull Request #17 - Se implementa las rutas para el proceso crud de carreras con feature/crud carreras
Estado: closed
Asignado a: No asignado
Creado: 2025-05-16T19:01:34Z
Cerrado: 2025-05-16T20:10:22Z
Descripción
Este pull request implementa el modelo Carrera en el sistema, permitiendo su gestión desde el backend mediante operaciones CRUD. Esta funcionalidad es esencial para permitir la asignación de cursos/eventos a carreras específicas y asegurar que los usuarios institucionales puedan completar correctamente su perfil.
Cambios realizados:
Añadido el modelo Carrera en prisma/schema.prisma.
Ejecutado npx prisma db push para reflejar los cambios en la base de datos.
Creado controlador carrera.controller.js con operaciones CRUD (obtener, crear, actualizar, eliminar).
Creado archivo de rutas carrera.routes.js y vinculado en app.js.
Issue relacionado
Closes #8
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Andriu-Dex: Traer los ultimos cambios realizados antes de aplicar los cambios
Pull Request #16 - Implementación de ruta POST /auth/login con verificación JWT y bcrypt
Estado: closed
Asignado a: No asignado
Creado: 2025-05-16T06:15:28Z
Cerrado: 2025-05-16T20:08:54Z
Descripción
Se implementa la funcionalidad completa de autenticación de usuarios tipo ESTUDIANTE mediante correo y contraseña.
■ Cambios realizados:
■ Controlador creado: src/controllers/auth.controller.js
Verifica existencia del usuario por correo (cor_usu)
Verifica rol obligatorio ESTUDIANTE
Compara contraseña hasheada usando bcryptjs
Genera un token JWT firmado con clave del .env
Retorna: { token, usuario } en JSON
■ Ruta POST creada: src/routes/auth.routes.js
Ruta: /api/login
Conectada desde app.js
■ Configuración de conexión a base de datos mediante PrismaClient en src/config/db.js
■ Archivo de prueba: Se puede verificar respuesta JWT en Postman o navegador
■ Cómo probarlo (Postman o frontend):
Método: POST
Ruta: http://localhost:3000/api/login
Headers:

```json
Content-Type: application/json
```
Body (JSON):
```json
{
"correo": "usuario@uta.edu.ec",
"contrasena": "clave_segura"
}
```
Respuesta esperada (200 OK):
```json
{
"token": "eyJhbGciOiJIUzI1...",
"usuario": {
"id": "uuid",
"correo": "usuario@uta.edu.ec",
"rol": "ESTUDIANTE"
}
}
```
Errores manejados:
401: Usuario no encontrado o contraseña incorrecta
500: Error interno del servidor
■ Variables necesarias en .env
```env
JWTSECRET=clavesegura
```
■ Dependencias usadas
bcryptjs: para verificar contraseñas
jsonwebtoken: para emitir JWT
Closes #7
Commits Vinculados
Merge pull request #16 from Andriu-Dex/feature/auth-backend
Autores de Commits
Andriu
Comentarios
Gabriel-Spartan: Funciona perfectamente el programa POST es funcional verifica si la contraseña y el usuario (correo) están almacenados en la base de datos anteriormente.
Pull Request #15 - Se añade el crud para carreras
Estado: closed
Asignado a: No asignado
Creado: 2025-05-16T03:48:58Z
Cerrado: 2025-05-16T18:55:02Z
Descripción
Este pull request implementa el modelo Carrera en el sistema, permitiendo su gestión desde el backend mediante operaciones CRUD. Esta funcionalidad es esencial para permitir la asignación de cursos/eventos a carreras específicas y asegurar que los usuarios institucionales puedan completar correctamente su perfil.
Cambios realizados:
Añadido el modelo Carrera en prisma/schema.prisma.
Ejecutado npx prisma db push para reflejar los cambios en la base de datos.
Creado controlador carrera.controller.js con operaciones CRUD (obtener, crear, actualizar, eliminar).

Creado archivo de rutas carrera.routes.js y vinculado en app.js.
Issue relacionado
Closes #8
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #14 - Implementación de autenticación backend: POST /login
Estado: closed
Asignado a: No asignado
Creado: 2025-05-15T18:28:18Z
Cerrado: 2025-05-15T20:24:03Z
Descripción
Descripción
Se implementó la ruta POST /api/login para permitir la autenticación de usuarios tipo ESTUDIANTE mediante correo y contraseña.
Se utiliza bcryptjs para la verificación de contraseñas y jsonwebtoken para la generación del token JWT.
El secreto de firma JWT se toma desde el archivo .env.
El endpoint retorna el token y los datos básicos del usuario autenticado.
Cambios principales
Nueva ruta: auth.routes.js
Nuevo controlador: auth.controller.js
Integración de rutas en: app.js
Dependencias: bcryptjs, jsonwebtoken, dotenv
Pruebas
El backend expone correctamente la ruta /api/login.
Si las credenciales son correctas y el usuario es de tipo ESTUDIANTE, retorna un token JWT y los datos del usuario.
Si las credenciales son incorrectas o el usuario no es ESTUDIANTE, retorna error 401.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #13 - Feature/crud carreras
Estado: closed
Asignado a: No asignado
Creado: 2025-05-15T15:00:49Z
Cerrado: 2025-05-15T19:29:54Z
Descripción
Este pull request implementa el modelo Carrera en el sistema, permitiendo su gestión desde el backend mediante operaciones CRUD. Esta funcionalidad es esencial para permitir la asignación de cursos/eventos a carreras específicas y asegurar que los usuarios institucionales puedan completar correctamente su perfil.
Cambios realizados:
-Añadido el modelo Carrera en prisma/schema.prisma.
-Ejecutado npx prisma db push para reflejar los cambios en la base de datos.
-Creado controlador carrera.controller.js con operaciones CRUD (obtener, crear, actualizar, eliminar).
-Creado archivo de rutas carrera.routes.js y vinculado en app.js.

-Adaptado todo el código a CommonJS (require y module.exports) para mantener compatibilidad con el proyecto actual.
Validación
Todos los endpoints funcionan correctamente con Insomnia/Postman.
Datos persistentes correctamente en PostgreSQL a través de Prisma.
Issue relacionado:
Closes #8
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #12 - ■ Pull Request: Login Frontend - Implementación completa
Estado: closed
Asignado a: No asignado
Creado: 2025-05-15T06:07:00Z
Cerrado: 2025-05-16T06:31:45Z
Descripción
■ Cambios realizados en este Pull Request
Se creó la vista Login.jsx con campos de correo y contraseña.
Se agregó validación del dominio institucional @uta.edu.ec.
Se aplicaron estilos con Bootstrap y animaciones personalizadas.
Se creó authService.js con una función login() simulada.
Se conectó Login.jsx con authService para probar el inicio de sesión sin backend real.
Se modificaron App.jsx y main.jsx para cargar el login como vista principal.
Se instalaron las dependencias necesarias (Bootstrap).
■ Descripción general
Se implementó la interfaz de inicio de sesión para estudiantes con validaciones básicas, estilos personalizados con Bootstrap y conexión a un servicio simulado (authService.js).
Además, se configuró el entorno principal (App.jsx y main.jsx) para cargar el login como vista principal.
■ Archivos modificados / añadidos
frontend/src/views/Login.jsx
frontend/src/services/authService.js
frontend/src/App.jsx
frontend/src/main.jsx
frontend/package.json
frontend/package-lock.json
■ Resultado esperado
El login funciona con validaciones de campos vacíos y dominio institucional.
Se muestra feedback visual (carga, errores, éxito).
La lógica está lista para ser conectada al backend más adelante.
■ Datos para pruebas
Correo: estudiante@uta.edu.ec
Contraseña: 123456
■ Resultado: Bienvenida simulada
Correo: otro@uta.edu.ec
Contraseña: abc
■ Resultado: Credenciales incorrectas
Closes #6
Commits Vinculados
Merge pull request #12 from Andriu-Dex/feature/login-frontend
Autores de Commits

Andriu
Comentarios
Sin comentarios
Pull Request #11 - ■ Crear modelo usuario con estructura normalizada y pruebas de conexión
Estado: closed
Asignado a: No asignado
Creado: 2025-05-15T01:49:32Z
Cerrado: 2025-05-15T04:05:50Z
Descripción
■■ Estándares aplicados a la base de datos
El nombre de las tablas y columnas están todos en minúsculas y en singular.
El nombre de cada tabla es completo.
El nombre de cada columna es de tres letras seguidos de un guion bajo (ced_usu).
■ Cambios realizados en este pull request
Se creó el modelo usuario en el archivo prisma/schema.prisma con los siguientes campos:
id_usu: UUID generado automáticamente
ced_usu: cédula única
nom_usu y ape_usu: nombre y apellido
cor_usu: correo único
con_usu: contraseña
cel_usu: celular con longitud exacta de 10 caracteres
rol_usu: enum con valores ADMIN y ESTUDIANTE
fec_cre_usu: fecha de creación por defecto
Se definió el enum rol_usuario para roles de usuario.
Se ejecutó npx prisma db push para sincronizar el modelo con la base de datos PostgreSQL.
Se generó correctamente el cliente Prisma en src/generated/prisma.
Se creó un archivo de prueba userTest.js en src/test/ para verificar la conexión y consultar los usuarios registrados.
■ Todo funcional y probado localmente.
■ Instrucciones para cargar el modelo usuario y verificar la base de datos local
Para que puedan tener el modelo usuario sincronizado correctamente con su base de datos PostgreSQL local, sigan los siguientes pasos:
■ 1. Configurar .env
Verifiquen que el archivo .env en la raíz del backend tenga esta línea con su conexión a PostgreSQL:
```
DATABASE_URL=postgresql://postgres:@localhost:/academicevents
```
Asegúrense de que su base de datos academicevents exista y el puerto esté correcto.
■ 2. Empujar el esquema a la base de datos
Este comando creará la tabla usuario en PostgreSQL:
```
npx prisma db push
```
Esto se puede ejecutar desde AcademicEvents\backend de preferencia.
■ 3. Verificar que Prisma generó el cliente
Verifiquen que se haya generado correctamente en:
```
src/generated/prisma/
```
■ 4. Probar consulta básica
Ejecuten el archivo de prueba con:
```
node src/test/userTest.js
```

```
Este archivo lista los usuarios existentes en la tabla usuario y confirmará si todo está funcionando correctamente.
Closes #5
Commits Vinculados
Merge pull request #11 from Andriu-Dex/feature/model-db
Autores de Commits
Andriu
Comentarios
Sin comentarios
Issue #10 - Middleware de autenticación y protección de rutas con JWT
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-15T00:34:38Z
Cerrado: 2025-05-17T06:26:35Z
Descripción
■ Implementar el middleware auth.js que verifique tokens JWT en las rutas privadas del backend. Este middleware se usará para proteger rutas que requieran autenticación, como inscripción a cursos, validación de pagos o acceso administrativo.
■ Tareas a implementar:
Crear la rama feature/auth-middleware desde develop.
Crear el archivo src/middlewares/auth.js.
Leer el token JWT desde el header Authorization: Bearer <token>.
Verificar la validez del token con jsonwebtoken.
Si es válido, añadir req.usuario = { id, rol } para uso en las rutas.
Si es inválido o ausente, devolver 401 Unauthorized.
■ Código base sugerido (auth.js)
```js
const jwt = require("jsonwebtoken");
const verificarToken = (req, res, next) => {
const authHeader = req.headers.authorization;
if (!authHeader || !authHeader.startsWith("Bearer ")) {
return res.status(401).json({ msg: "Token no proporcionado" });
}
const token = authHeader.split(" ")[1];
try {
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.usuario = decoded;
next();
} catch (error) {
return res.status(401).json({ msg: "Token inválido" });
}
};
module.exports = verificarToken;
```
■ Uso sugerido en rutas:
```js
const express = require("express");
const router = express.Router();
const verificarToken = require("../middlewares/auth");
router.get("/privado", verificarToken, (req, res) => {
res.json({ msg: "Acceso permitido", usuario: req.usuario });
});
```
```

```
■ Resultado esperado:
Middleware funcional que protege rutas con JWT
Rutas privadas devuelven 401 si no hay token válido
Base lista para segmentar acceso por rol (ADMIN, ESTUDIANTE)
■ Estructura esperada:
plaintext
backend/
■■■ src/
■■■ middlewares/
■ ■■■ auth.js
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #9 - Implementar subida y validación de comprobante de pago (backend)
Estado: closed
Asignado a: carlitosgiovanniramos
Creado: 2025-05-15T00:28:54Z
Cerrado: 2025-05-18T04:52:22Z
Descripción
Implementar en el backend la lógica para permitir que un estudiante suba un comprobante de pago (foto o PDF) al inscribirse a un curso pagado. El archivo será validado por un administrador más adelante. Se requiere aceptar el archivo, guardarlo localmente o temporalmente y registrar su estado ("pendiente", "aprobado", "rechazado").
■ Tareas a implementar:
Crear la rama feature/subida-comprobante desde develop.
Instalar el paquete multer para gestionar carga de archivos:
bash
npm install multer
Crear middleware src/middlewares/upload.js para configurar multer.
Crear una ruta POST /comprobantes/subir (temporal) en src/routes/comprobante.routes.js.
Guardar archivo en una carpeta /uploads y registrar el nombre en un objeto temporal o estructura de prueba.
Devolver estado de subida (200 OK + nombre del archivo).
■ Estructura esperada:
plaintext
backend/
■■■ uploads/ # Carpeta donde se almacenarán los comprobantes
■■■ src/
■ ■■■ middlewares/
■ ■ ■■■ upload.js # Configuración de multer
■ ■■■ routes/
■ ■ ■■■ comprobante.routes.js
■ ■■■ controllers/
■ ■ ■■■ comprobante.controller.js (opcional, si se organiza bien)
■ Middleware base sugerido (upload.js)
```js
const multer = require("multer");
const path = require("path");
const storage = multer.diskStorage({
```
```

```
destination: function (req, file, cb) {
cb(null, "uploads/");
},
filename: function (req, file, cb) {
const ext = path.extname(file.originalname);
const uniqueName = Date.now() + "-" + file.fieldname + ext;
cb(null, uniqueName);
},
});
const upload = multer({ storage });
module.exports = upload;
```
■ Validaciones (a añadir si desea ir más allá):
Solo permitir .jpg, .png, .jpeg, .pdf
Tamaño máximo recomendado: 2MB
Validar que el usuario esté autenticado (una vez que se integre JWT)
■ Resultado esperado:
Comprobante guardado en /uploads
Nombre del archivo retornado como respuesta
Backend preparado para manejar pagos cargados manualmente
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #8 - CRUD de carreras (modelo, rutas y controladores)
Estado: closed
Asignado a: titoma1500
Creado: 2025-05-15T00:19:51Z
Cerrado: 2025-05-17T04:07:36Z
Descripción
Implementar el modelo Carrera en Prisma y construir las rutas REST para su gestión administrativa. Esta funcionalidad permitirá asociar cursos/eventos a carreras específicas y es esencial para el proceso de inscripción.
■ Tareas:
Crear la rama feature/crud-carreras desde develop.
Añadir el modelo Carrera en prisma/schema.prisma.
Ejecutar npx prisma db push.
Crear los controladores (carrera.controller.js) con lógica CRUD.
Crear las rutas (carrera.routes.js) y vincular en app.js.
■ Modelo sugerido (schema.prisma):
```prisma
model Carrera {
id String @id @default(uuid())
nombre String @unique
estado Boolean @default(true)
creadaEn DateTime @default(now())
}
```
■ Endpoints:
Método	Ruta	Acción

GET	/carreras	Obtener todas las carreras
POST	/carreras	Crear una nueva carrera
PUT	/carreras/\:id	Actualizar una carrera
DELETE	/carreras/\:id	Eliminar una carrera
■ Estructura esperada:
plaintext
backend/
■■■ src/
■■■ controllers/
■ ■■■ carrera.controller.js
■■■ routes/
■ ■■■ carrera.routes.js
■ Resultado esperado:
Migración aplicada.
API REST funcional.
Código modular y limpio.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #7 - Ruta de login y autenticación JWT
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-14T23:49:40Z
Cerrado: 2025-05-16T21:15:53Z
Descripción
Implementar la ruta POST /auth/login en el backend para permitir que los usuarios se autentiquen con su correo y contraseña. Esta autenticación debe devolver un token JWT firmado que será usado por el frontend en futuras peticiones protegidas.
■ Tareas a implementar:
Crear la rama feature/auth-backend desde develop.
Crear un archivo de rutas src/routes/auth.routes.js.
Crear un controlador src/controllers/auth.controller.js.
Implementar la lógica:
Buscar al usuario por correo.
Verificar que exista y su tipo sea ESTUDIANTE.
Comparar contraseñas con bcryptjs.
Generar un token JWT con jsonwebtoken.
Devolver el token y datos del usuario.
Usar secreto de firma desde .env:
env
JWT_SECRET=clave_segura
■ Código base sugerido (auth.controller.js)
```js
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const prisma = require("../config/db");
const login = async (req, res) => {
const { correo, contrasena } = req.body;
try {
const user = await prisma.usuario.findUnique({ where: { correo } });
```

```
if (!user || user.tipo !== "ESTUDIANTE") {
return res.status(401).json({ msg: "Credenciales inválidas" });
}
const passwordValid = await bcrypt.compare(contrasena, user.contrasena);
if (!passwordValid) {
return res.status(401).json({ msg: "Contraseña incorrecta" });
}
const token = jwt.sign({ id: user.id, rol: user.tipo }, process.env.JWT_SECRET, {
expiresIn: "2h",
});
return res.status(200).json({ token, usuario: { id: user.id, correo: user.correo, rol: user.tipo } });
} catch (error) {
return res.status(500).json({ msg: "Error interno", error });
}
};
module.exports = { login };
```
■ Resultado esperado:
Ruta POST /auth/login funcional.
Contraseña verificada con bcryptjs.
Token generado y devuelto.
Código modular en controladores y rutas.
■ Estructura esperada:
plaintext
backend/
■■■ src/
■■■ routes/
■ ■■■ auth.routes.js
■■■ controllers/
■ ■■■ auth.controller.js
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #6 - Vista de Login y validación de correo institucional
Estado: closed
Asignado a: M4yb33
Creado: 2025-05-14T23:39:23Z
Cerrado: 2025-05-16T06:33:02Z
Descripción
■■ Crear la interfaz de inicio de sesión para usuarios estudiantes (correo institucional) usando React. Esta vista debe contener los campos de correo y contraseña, con validaciones básicas, y debe bloquear el intento si el correo no pertenece al dominio @uta.edu.ec.
■ Tareas a implementar:
Crear la rama feature/login-frontend desde develop.
Crear el componente Login.jsx en frontend/src/views/.
Implementar formulario con campos:
■ Correo electrónico
■ Contraseña
Validar que el correo contenga @uta.edu.ec
Mostrar advertencias con un diseño amigable si los campos están vacíos o el dominio no es válido.

Crear una función handleSubmit para preparar el POST hacia la API (endpoint aún no implementado).
Estilizar el formulario usando TailwindCSS (respetar responsividad mínima).
Mostrar toast, alerta o mensaje según los estados (éxito, error, validación).
■ Sugerencias de validación:
js
const correoValido = (email) => {
return email.endsWith("@uta.edu.ec");
};
■ Resultado esperado:
Vista funcional y validada.
No requiere integración real todavía (solo UI y validación).
Código limpio, modular y comentado.
■ Estructura esperada:
plaintext
frontend/
■■■ src/
■■■ views/
■ ■■■ Login.jsx
■■■ services/
■ ■■■ authService.js (opcional para centralizar llamadas futuras)
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Issue #5 - Modelo Usuario y migración Prisma
Estado: closed
Asignado a: Gabriel-Spartan
Creado: 2025-05-14T23:34:43Z
Cerrado: 2025-05-16T05:50:40Z
Descripción
■ Descripción:
■ Issue: Crear el modelo Usuario dentro del esquema de Prisma y realizar la migración a la base de datos PostgreSQL. Este modelo será fundamental para gestionar tanto a estudiantes como administradores dentro del sistema, diferenciándolos por roles y restricciones específicas.
■ Tareas a implementar:
Crear la rama feature/modelo-usuario desde develop.
Modificar el archivo prisma/schema.prisma para incluir el modelo Usuario.
Ejecutar npx prisma db push para aplicar la migración.
Confirmar que Prisma genera correctamente el cliente en src/generated/prisma.
Probar la conexión con una consulta básica en un archivo de prueba (src/test/userTest.js, opcional).
■ Modelo sugerido (schema.prisma)
```prisma
model Usuario {
id String @id @default(uuid())
cedula String @unique
nombres String
apellidos String
correo String @unique
contrasena String
carrera String? // Obligatorio para estudiantes con correo institucional
tipo RolUsuario
```

estado Boolean @default(true) // activo/inactivo
creadoEn DateTime @default(now())
}
enum RolUsuario {
ADMIN
ESTUDIANTE
}
```

■ Resultado esperado:
Archivo prisma/schema.prisma con el modelo definido.
Base de datos actualizada mediante npx prisma db push.
Cliente de Prisma funcionando para futuras consultas en el backend.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios
Pull Request #4 - Merge pull request #3 from Andriu-Dex/main
Estado: closed
Asignado a: No asignado
Creado: 2025-05-14T23:00:34Z
Cerrado: 2025-05-14T23:02:18Z
Descripción
Sincronización de main con develop
Commits Vinculados
Merge pull request #4 from Andriu-Dex/develop
Autores de Commits
Andriu
Comentarios
Sin comentarios
Pull Request #3 - Sincronización de main con develop
Estado: closed
Asignado a: No asignado
Creado: 2025-05-14T23:00:04Z
Cerrado: 2025-05-14T23:00:12Z
Descripción
Este Pull Request integra los cambios iniciales desde main hacia develop
Commits Vinculados
Merge pull request #4 from Andriu-Dex/develop; Merge pull request #3 from Andriu-Dex/main
Autores de Commits
Andriu
Comentarios
Sin comentarios
Pull Request #2 - Estructura base del backend,frontend y conexión con Prisma
Estado: closed
Asignado a: No asignado
Creado: 2025-05-14T22:45:37Z
Cerrado: 2025-05-14T22:49:11Z
Descripción
Este Pull Request establece la base inicial del proyecto AcademicEvents, abarcando tanto el backend como el frontend.
■ Backend (Node.js + Express + Prisma)
```

Se inicializó el proyecto con npm init.
Se instalaron y configuraron los paquetes principales: express, cors, dotenv, @prisma/client, nodemailer, jsonwebtoken, bcryptjs.
Se configuró nodemon para desarrollo y se agregó script dev.
Se ejecutó npx prisma init para crear el esquema inicial y la estructura de conexión.
Se creó el archivo src/app.js y se agregó el endpoint base GET /.
Se reorganizó la estructura en carpetas (controllers, routes, middlewares, etc.).
Se creó archivo de conexión a Prisma (src/config/db.js).
Se añadió variable PORT_BACKEND en .env y .env.example.
■ Frontend (React + Tailwind CSS)
Se generó el proyecto con Vite.
Se configuró Tailwind CSS y se personalizaron estilos en index.css.
Se instalaron dependencias base: axios, react-router-dom.
Se verificó la ejecución correcta con npm run dev.
■ Otros archivos
Se añadió .gitignore para ignorar dependencias, entorno y archivos de sistema.
Se añadió .env.example con las variables necesarias para ambos entornos.
Se estructuraron las carpetas principales del proyecto.
Closes #1
Commits Vinculados
Merge pull request #2 from Andriu-Dex/feature/setup-entorno
Autores de Commits
Andriu
Comentarios
Sin comentarios
Issue #1 - Estructura base del backend y conexión a base de datos
Estado: closed
Asignado a: Andriu-Dex
Creado: 2025-05-13T22:04:32Z
Cerrado: 2025-05-14T22:49:12Z
Descripción
■ Issue: Estructura base del backend y conexión a base de datos
Descripción:
Se debe implementar la estructura inicial del backend del proyecto y establecer la conexión con la base de datos utilizando Prisma ORM. Esta tarea también incluye la configuración básica del entorno frontend con Vite y TailwindCSS, de forma que ambos entornos (cliente y servidor) estén listos para el desarrollo colaborativo.
■ Objetivo:
Configurar desde cero tanto el backend como el frontend para el proyecto AcademicEvents, asegurando una estructura clara, mantenible y preparada para escalabilidad.
■ Tareas a implementar:
■ Backend
Inicializar proyecto Node.js con npm init -y.
Instalar dependencias necesarias:
Producción: express, cors, dotenv, @prisma/client, jsonwebtoken, bcryptjs, nodemailer
Desarrollo: nodemon, prisma
Crear estructura base:
Archivo principal src/app.js
Middleware de CORS y JSON
Ruta raíz (GET /) de prueba
Configurar scripts en package.json (start, dev, prisma)
Inicializar Prisma con npx prisma init
Crear archivo .env para variables sensibles

■ Frontend
Crear proyecto con Vite (npm create vite@latest frontend -- --template react)
Instalar dependencias:
React: axios, react-router-dom
Estilos: tailwindcss, postcss, autoprefixer
Configurar Tailwind (tailwind.config.js, index.css)
Establecer estilos base para tipografía, botones, enlaces y modo oscuro
Verificar ejecución local con npm run dev
■ Resultado esperado:
Backend corriendo en localhost:3000 con un endpoint base funcional.
Frontend inicializado y estilizado con Tailwind, listo para consumir endpoints del backend.
Prisma configurado y preparado para definir modelos de base de datos.
Commits Vinculados
Ninguno
Autores de Commits
Ninguno
Comentarios
Sin comentarios