Documentación Técnica: Sistema de Gestión Académica

Desarrolladores: Ulises Mendoza y Cesar Campo

Versión: 1.0
Fecha: 2025-04-11

Contenido

1. Introducción

Propósito del Proyecto

Desarrollar una aplicación web para facilitar la gestión académica en un entorno universitario, centralizando la información de estudiantes, asignaturas, matrículas y calificaciones.

Objetivos:

- Proveer una interfaz intuitiva y eficiente para administradores o personal académico.
- Asegurar la integridad y seguridad de los datos académicos.
- Permitir la consulta y modificación de registros clave.
- Ofrecer funcionalidades básicas de reporte.

Alcance

El proyecto se desarrolló en fases, cubriendo:

- **Nivel 1 (MVP):** Registro y consulta básica de estudiantes, asignaturas, matrículas y calificaciones.
- **Nivel 2:** Modificación de estudiantes/asignaturas, desmatriculación, generación de reportes simples.
- **Nivel 3:** Eliminación de estudiantes, asignaturas y calificaciones.

2. Configuración del Entorno de Desarrollo

Prerrequisitos:

1. PHP (>= 8.2 recomendado)
2. Composer (Gestor de dependencias PHP)
3. Node.js (>= 18.x recomendado) y npm/yarn (Gestor de paquetes JS)
4. Base de Datos (MySQL >= 8.0 o PostgreSQL >= 13 recomendado)
5. Git

Pasos de Instalación:

1. Clonar el repositorio: git clone https://github.com/Syndrast/proyecto-final-web.git
2. Navegar al directorio del proyecto: cd proyecto-final-web
3. Instalar dependencias PHP: composer install
4. Crear archivo de entorno: .env (hay un archivo de ejemplo .env.example en el repositorio)
5. Generar clave de aplicación: php artisan key:generate
6. Configurar la conexión a la base de datos en el archivo .env (DB_DATABASE, DB_USERNAME, DB_PASSWORD, etc.).
7. Crear la base de datos especificada en .env.
8. Instalar dependencias JavaScript: npm install (o yarn install)
9. Ejecutar las migraciones de base de datos: php artisan migrate
10. Ejecución del Proyecto:
    1. Iniciar el servidor de desarrollo de Laravel: php artisan serve
    2. Iniciar el compilador de assets de Vite: npm run dev
    3. Acceder a la aplicación en la URL proporcionada por php artisan serve (http://127.0.0.1:8000).

3. Stack Tecnológico

- **Backend:** Laravel (v11.x)
- **Frontend:** React (v18.x)
- **Puente Backend-Frontend:** Inertia.js
- **Base de Datos:** MySQL / PostgreSQL (según configuración .env)
- **Servidor de Desarrollo Frontend:** Vite
- **Estilos CSS:** Tailwind CSS (v3.x - configurado por Laravel Breeze)
- **Autenticación:** Laravel Breeze (Stack React/Inertia)

4. Arquitectura del Sistema

- **Visión General:** Se emplea una arquitectura Monolítica donde Laravel maneja tanto la lógica de backend como el enrutamiento y la presentación inicial de las vistas.
- **Flujo con Inertia.js:**

1. El navegador realiza una petición a una ruta definida en routes/web.php.
2. El controlador de Laravel asociado procesa la petición, obtiene los datos necesarios de la base de datos (vía Modelos Eloquent).
3. El controlador devuelve una respuesta usando Inertia::render('NombrePaginaReact', ['prop1' => $data1, ...]).
4. El middleware HandleInertiaRequests intercepta la respuesta, añade props compartidas (auth, flash, ziggy) y envía una respuesta JSON al frontend.
5. Inertia (en el frontend, app.jsx) recibe el JSON, identifica el componente de página React (NombrePaginaReact.jsx) y lo renderiza, pasándole los datos (prop1, auth, flash, etc.) como props.
6. Navegaciones subsecuentes (usando <Link>) o envíos de formulario (useForm) realizan peticiones XHR/Fetch en segundo plano, recibiendo solo el JSON con las props necesarias para actualizar el componente de página, simulando una SPA.

- **Middleware Clave:**
  - web: Grupo de middleware estándar de Laravel (sesión, cookies, CSRF).
  - auth: Asegura que el usuario esté autenticado para acceder a ciertas rutas.
  - App\Http\Middleware\HandleInertiaRequests: Añade props compartidas a todas las respuestas Inertia.

5. Estructura de Directorios

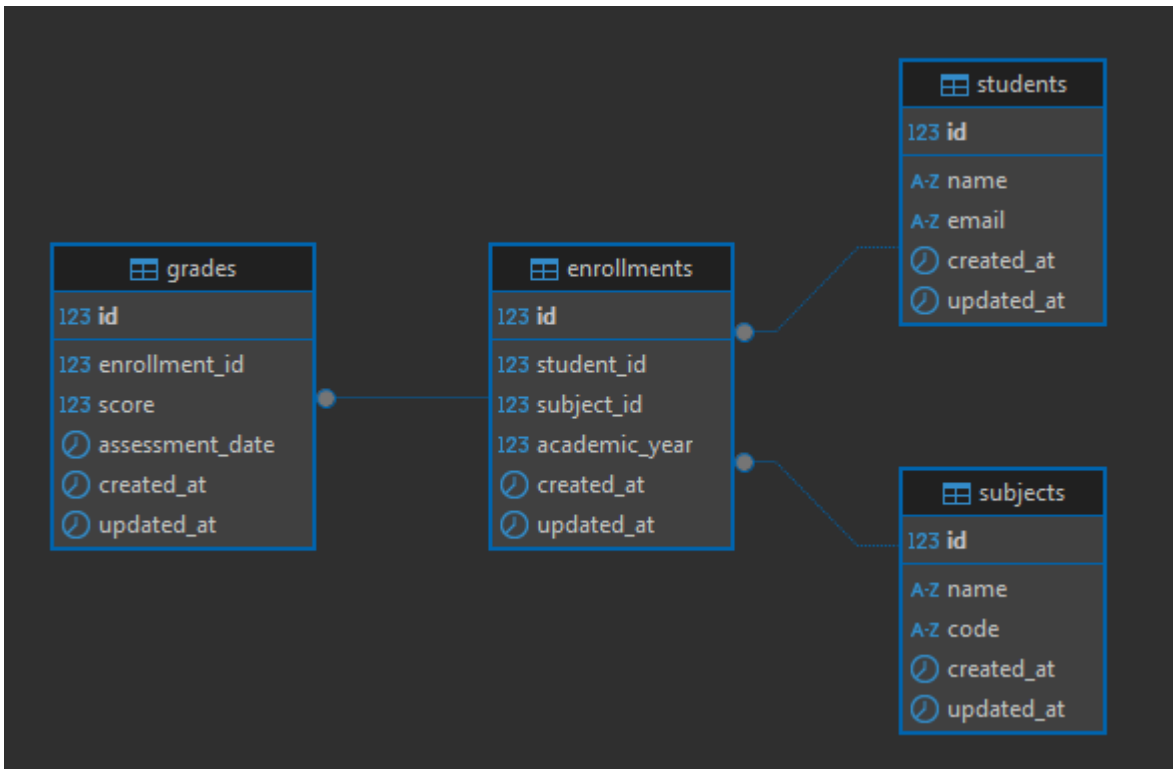
- **Backend (Estructura estándar Laravel):**
  - app/Http/Controllers/: Controladores que renderizan vistas Inertia.
  - app/Http/Middleware/: Middleware (destaca HandleInertiaRequests.php).
  - app/Http/Requests/: Form Requests para validación.
  - app/Models/: Modelos Eloquent.
  - app/Policies/: Políticas de autorización.
  - app/Providers/: Service Providers.
  - database/migrations/: Migraciones de base de datos.

- database/seeders/: Seeders para datos de prueba.
- routes/web.php: Definición de rutas para páginas Inertia.
- routes/auth.php: Rutas de autenticación (generadas por Breeze).

- **Frontend (resources/js/):**
  - Components/: Componentes React UI reutilizables (Button, Input, SelectInput, Modal, etc.).
  - Layouts/: Layouts persistentes de React (AuthenticatedLayout.jsx, GuestLayout.jsx).
  - Pages/: Componentes React que representan páginas completas (reciben props del controlador). Organizados por funcionalidad (e.g., Students/Index.jsx, Subjects/Edit.jsx).
  - app.jsx: Punto de entrada principal de JS, configuración de createInertiaApp.
  - bootstrap.js: Configuración inicial de JS.

6. Base de Datos

- **Diseño del Esquema:**



  - users: Usuarios de la aplicación (administradores/personal). Creada por Breeze.
  - students: Información de los estudiantes (id, name, email, ...).
  - subjects: Información de las asignaturas (id, name, code, ...).
  - enrollments: Tabla pivote para matrículas (id, student_id, subject_id, academic_year, ...).
  - grades: Calificaciones (id, enrollment_id, score, assessment_date). Relación uno a uno con enrollments.

- **Migraciones:** Se gestionan con php artisan migrate, php artisan migrate:rollback, etc. Definen la estructura de las tablas.
- **Modelos Eloquent:** Representan las tablas (Student.php, Subject.php, etc.). Definen relaciones (hasOne, belongsTo, hasMany). Contienen $fillable para asignación masiva.

7. Backend (Detalles Laravel)

- **Enrutamiento:** Principalmente en routes/web.php. Se usan rutas nombradas (->name(...)) para referencia fácil en el frontend con Ziggy. Se utiliza Route::resource para acciones CRUD estándar y rutas individuales para acciones específicas (matrícula, notas, reportes).
- **Controladores:** Su responsabilidad es orquestar la respuesta: obtener datos (delegando a modelos o servicios si la lógica es compleja), validar la entrada (usando Form Requests) y devolver la respuesta Inertia (Inertia::render(...)) o una redirección (redirect()->route(...)). Se aplica Route Model Binding para simplificar la carga de modelos.
- **Form Requests:** Clases en app/Http/Requests que encapsulan la lógica de validación y autorización (opcional) para una petición específica. Se inyectan directamente en los métodos del controlador.
- **Middleware (HandleInertiaRequests):** Configurado en share() para pasar datos globales como auth (datos del usuario), flash (mensajes de sesión para notificaciones) y ziggy (configuración de rutas para JS).
- **Autenticación/Autorización:** Gestionada por Laravel Breeze. Las rutas protegidas usan el middleware auth. Se pueden implementar Policies (php artisan make:policy) para un control de acceso más granular a nivel de modelo/recurso.

8. Frontend (Detalles React & Inertia)

- **Inicialización (app.jsx):** Configura createInertiaApp, resuelve los componentes de Página dinámicamente y monta la aplicación React en el DOM (usando app.blade.php como plantilla raíz). Configura el layout persistente por defecto.
- **Componentes de Página (Pages/):** Componentes funcionales de React que reciben props directamente desde el controlador Laravel (incluyendo datos, auth, flash, errors). Representan una vista completa.
- **Componentes Reutilizables (Components/):** Componentes de UI genéricos (botones, inputs, modales, tablas) diseñados para ser reutilizados en diferentes Páginas. Reciben datos y callbacks como props.
- **Layouts (Layouts/):** Componentes React que envuelven a los componentes de Página. Contienen elementos persistentes como la barra de navegación principal,

sidebar, footer. AuthenticatedLayout usa la prop auth.user para mostrar información del usuario.

- **Enrutamiento:**
    - <Link href={route('...')}>: Componente de Inertia para navegación interna sin recarga de página completa.
    - router: Objeto global de Inertia (import { router } from '@inertiajs/react') para navegaciones programáticas (router.get(...), router.post(...), router.delete(...), etc.).
    - route(): Helper de Ziggy (disponible globalmente si se configura en HandleInertiaRequests) para generar URLs basadas en nombres de ruta de Laravel.
- **Manejo de Formularios:** El hook useForm de @inertiajs/react simplifica el manejo de formularios: gestiona el estado de los datos (data, setData), el estado de envío (processing), los errores de validación del backend (errors) y métodos para enviar (post, patch, put, delete).
- **Estado:** La mayoría del estado "global" proviene directamente del backend como props (datos de la página, usuario autenticado, mensajes flash). Para el estado local de componentes se usa useState. Context API o Zustand/Jotai podrían usarse para estados UI complejos transversales (ej. tema oscuro), pero generalmente no son necesarios para el estado de datos principal.

9. Implementación de Funcionalidades Clave

- **Estudiantes/Asignaturas (CRUD):** Implementado con Route::resource, controladores (StudentController, SubjectController), Form Requests (Store/Update...Request), y componentes React en Pages/Students/ y Pages/Subjects/ (Index, Create, Edit). La eliminación usa el método destroy del controlador y router.delete en el frontend con confirmación.
- **Matrículas:** Lógica personalizada en EnrollmentController (create, store, index, destroy). Formulario en Pages/Enrollments/Create.jsx usa SelectInput para estudiantes/asignaturas. La eliminación (destroy) borra la matrícula y su nota asociada (manualmente o por cascade).
- **Calificaciones:** Lógica en GradeController (create, store, destroy). Usa updateOrCreate en store para manejar creación y actualización con una sola

acción. Formulario en Pages/Grades/Create.jsx (reutilizado para editar) recibe enrollment y existingGrade. La eliminación (destroy) solo borra la nota.

- **Reportes:** ReportController con método index (muestra formulario) y generate (recibe criterios, consulta la BBDD con relaciones, y devuelve los datos a la misma vista Pages/Reports/Index.jsx). El frontend muestra el formulario y condicionalmente la tabla de resultados.

10. Despliegue (Consideraciones)

- **Servidor:** Configurar un servidor web (Nginx o Apache) con PHP y la base de datos elegida.
- **Variables de Entorno:** Crear un archivo .env específico para producción con los ajustes correctos (APP_ENV=production, APP_DEBUG=false, URL de la app, credenciales de BBDD, etc.).
- **Dependencias:** Ejecutar composer install --optimize-autoloader --no-dev y npm install --production.
- **Compilación de Assets:** Ejecutar npm run build para generar los archivos JS/CSS optimizados en el directorio public/build.
- **Configuración Laravel:**
    - Ejecutar php artisan config:cache y php artisan route:cache
    - Ejecutar php artisan migrate --force para aplicar migraciones en producción.
    - Asegurar permisos correctos en directorios storage y bootstrap/cache.
- **Servidor Web:** Configurar el servidor para apuntar al directorio public del proyecto y añadir las reglas de reescritura necesarias para Laravel.