**Manual del sistema**

**Requerimientos:**

1. El sistema debe contar con un módulo de registro y autenticación de usuarios, garantizando un acceso seguro y personalizado.
2. Debe permitir la gestión de proyectos PA y PIA, así como la posibilidad de incorporar otros tipos de proyectos en el futuro.
3. El sistema debe diferenciar a los usuarios según sus roles (docente, evaluador, coordinador, etc.), permitiendo así que cada uno acceda únicamente a las funcionalidades correspondientes a su perfil.
4. Debe permitir asociar docentes a los proyectos, quienes serán los responsables de su gestión y seguimiento.
5. Los usuarios con rol de evaluador deben poder asignar calificaciones y observaciones a los proyectos registrados.
6. El sistema debe permitir el desarrollo progresivo de los proyectos, facilitando el registro de avances, evidencias y actividades asociadas.
7. Debe garantizar altos estándares de usabilidad, accesibilidad y seguridad, brindando una experiencia de usuario óptima en diferentes dispositivos.
8. El sistema debe estar diseñado para ser escalable, de modo que pueda adaptarse al crecimiento de usuarios, funcionalidades y tipos de proyectos a futuro.

**Proceso de Creación:**

1. Este proyecto está basado en la plantilla del profesor, por lo cual se inició clonando el repositorio.
2. Definimos nuestras entidades en un diagrama de ER.
3. Definimos nuestras variables de entorno y las coordinamos con nuestra base de datos.
4. Se crean las migraciones de todas las entidades y finalmente ejecuta el comando con migrate para que se ejecuten en la base de datos.
5. Posteriormente se deben crear los modelos que nos servirán para los controladores e interactuar con nuestra base de datos.
6. Una vez tengamos los modelos y migraciones creados, podemos proceder con los controladores, los cuales son archivos que contendrán la lógica y con funciones de parte del ORM eloquent esta va a interactuar con la base de datos.
7. Luego de tener los controladores de todos los archivos podemos definir las rutas que va a contener nuestro y las cuales cada una interactuará con su respectivo controlador.
8. Finalmente tenemos las vistas, las cuales encontramos en la carpeta de resources, allí creamos una carpeta por cada entidad y creamos 3 archivos de blade los cuales son: index, create y edit. Consideramos que fueron los justos para poder desarrollar este proyecto.
9. Afortunadamente este proyecto ya contaba con tailwind instalado por lo cual nos ahorramos ese paso, este framework de estilos fue de ayuda para que el diseñar la página web fuera más eficiente y le podamos dar un toque a nuestra aplicación.

**Paquetes e Instalaciones:**

En este proyecto se utilizaron los siguientes paquetes:

* **Tailwind**: El cual es un framework de CSS.
* **Laravel**: Es un framework fullstack aunque principalmente se usa para hacer backend.
* **Breeze:** Es una implementación de laravel cuya finalidad es permitir hacer código para realizar autenticación y registro de forma eficiente.
* **Spatie:** Aunque no pudimos usar esta otra implementación, esta se usa para hacer roles y permisos en un sistema con usuarios.
* **Google Fonts:** Utilizado para cambiar la fuente de la letra.
* **Font Awesome:** Utilizado para iconos.

**Instrucciones de clonación:**

1. Vamos al repositorio que nos compartió el profesor.
2. Le damos click en “code” y copiamos la dirección HTTPS.
3. Nos vamos a una carpeta de nuestro equipo y le damos click derecho y abrimos la consola.
4. Escribimos el comando “git clone” junto a la dirección previamente copiada
5. Le damos enter y esto cargará el proyecto.
6. Para abrirlo con visual studio code, podemos escribir cd y entramos a la ruta.
7. Escribimos “Code .” y esto abrirá el VSC con el proyecto.

**Instrucciones Paso a Paso:**

**database:** Primero que nada desde pgAdmin creamos nuestra base de datos, y le asignamos un nombre, debemos tener en cuenta nuestras credenciales (usuario y contraseña), las cuales usaremos para el archivo .env.

**.env:** Ubicamos y entramos al archivo .env, posteriormente buscamos las opciones:  
  
DB\_CONNECTION=pgsql

DB\_HOST=127.0.0.1

DB\_PORT=5432

DB\_DATABASE=\*\*nombre de tu base de datos\*\*

DB\_USERNAME=\*\*nombre de usuario\*\*

DB\_PASSWORD=\*\*contraseña de postgres\*\*

**php.ini**: Nos dirijimos a esta ruta: “C:\xampp\php”.

Allí buscamos un archivo llamado php.ini y lo abrimos con el bloc de notas, posteriormente buscamos la linea que dice “;extension=pdo\_pgsql” y quitamos el “;” que hay al inicio.

Guardamos el archivo y ya queda configurado.

**Migraciones:**

Debemos ejecutar un comando para crear el archivo de migraciones el cual es: “php artisan make:model \*\*nombre de la migración\*\*“, este comando creará de una vez un archivo de migraciones y otro de modelos. Una vez tenemos nuestro archivo de migraciones escribimos las instrucciones según como se menciona en la documentación de Laravel para crear las tablas directamente desde nuestro archivo de php. Posteriormente una vez creamos los campos y referencias (si las tiene) ejecutamos otro comando, “php artisan migrate”, esto creará todas las tablas en nuestra base de datos.

Como nota: Se recomienda crear todas las migraciones primero antes de hacer el migrate.

**Modelos:**

Como se mencionó anteriormente, con el comando “php artisan make:model \*\*nombre de la migración\*\*“ ya se crea el modelo a la ves que la migración, es aquí donde escribimos las instrucciones pertinentes para cada modelo, definiendo sus campos y características que van a contener estos campos, además de sus relaciones con otras tablas.

**Controladores:**

Pasamos con los controladores, aquí tenemos que usar otro comando el cual será “php artisan make:controller \*\*nombre del controlador\*\*” y es aquí donde podemos codear las funciones principales que identifican un CRUD o incluso mucho más. Principalmente tenemos:

Index(), que es para mostrar información.

Store(), que es para subir información, similar a un post.

Update(), que es para editar nuestros datos.

Destroy(), para eliminar información.

Estos 4 métodos principales son los que debemos configurar y si necesitamos otro más específico podemos hacerlo siguiendo la documentación sugerida por Laravel.

**Rutas:**

Debemos también configurar nuestro archivo de rutas, ubicado en la carpeta routes/web.php debemos definir si queremos que se puedan acceder a rutas publicas sin usar un middleware de autorización, en el caso de nuestra aplicación todo a excepción de la landing page y las opciones de inicio de sesión y registro serán publicas, por lo que puede por así:  
  
***Route::get('/', function () {***

***return view('welcome');***

***});***

De lo contrario si deseamos proteger nuestras rutas debemos usar un middleware que identifique si ya el usuario está autenticado, estas opciones nos las provee Breeze para ser más eficientes.

**Vistas:**

Finalmente las vistas o también conocidos como nuestros blades son los siguientes en aparecer, simplemente nos dirigimos a resources y tenemos nuestros diferentes archivos que podemos segmentar como queramos, componentes, vistas, o UI. En un de estos podemos simplemente generar un archivo del nombre que concuerde con nuestras rutas y que contengan la extensión “blade.php”.

Aquí podemos usar herramientas como tailwind o javascript incluso para interactuar con el DOM pero a su vez al ser archivos con la extensión “blade.php” nos permite usar código php para hacer uso de los controladores y así interactuar con la base de datos.

**Ejecución Localmente:**

Para ejecutar este proyecto debemos abrir dos terminales. Antes que nada asegurarnos que tenemos todas las dependencias al día y usar “composer update” y en la otra terminal “npm i”, posteriormente, usamos los comandos “php artisan serve” para ejecutar del lado del servidor y “npm run dev” en la otra terminal para ejecutar del lado del cliente. De esta forma podemos ya usar nuestra página web.