



DIRECCIÓN ACADÉMICA
VICERRECTORADO ACADÉMICO

Facultad de Ingeniería

Carrera de Tecnologías de la Información

Informe de Actividad de Investigación Formativa

**Periodo Académico
2024 – 1S**



Contenido

1. Autores	3
2. Personal Académico	3
3. Resultados de Aprendizaje de la asignatura:	3
4. Tema de la Actividad de la Investigación Formativa:	3
5. Objetivos de la(s) actividad(es):	3
6. Fecha de la ejecución:	3
7. Desarrollo del Informe	3
7.1 Introducción. (1 página)	3
7.2 Descripción de la metodología (Especificación de cómo se realizaron la(s) actividad(es) de Investigación Formativa. (Qué y Cómo)	4
7.3 Descripción de la(s) acción(es) realizadas (Fase de Ejecución y Seguimiento y Fase de Socialización y Reflexión)	5
7.4 Resultados	8
7.5 Bibliografía	31
8. ANEXOS (Evidencias)	31



1. Autor

- Elvis Santiago Pilco Paucar

2. PERSONAL ACADÉMICO

- MsC. Jorge Delgado
- Ing. Milton Lopez.

3. RESULTADOS DE APRENDIZAJE DE LA ASIGNATURA:

Desarrollar aplicaciones y sistemas basados en TI que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas, a través del uso de metodologías centradas en el usuario y la organización

4. TEMA DE LA ACTIVIDAD DE LA INVESTIGACIÓN FORMATIVA:

Desarrollo de una Plataforma de Gestión de Usuarios en una aplicación web mediante el uso de Laravel y MySQL.

5. OBJETIVOS DE LAS ACTIVIDADES:

Objetivo General:

- Desarrollar una aplicación web utilizando Laravel y la base de datos MySQL con la incorporación de funcionalidades de registro, inicio de sesión y cierre de sesión de usuarios, así como la administración completa de usuarios mediante MySQL.

6. FECHA DE LA EJECUCIÓN:

7. DESARROLLO DEL INFORME

20 de mayo de 2024 - 25 de mayo del 2024

7.1 Introducción.

El presente proyecto se enfoca en el desarrollo de una aplicación web utilizando el framework Laravel y la base de datos MySQL, con el objetivo de crear un sistema de gestión de usuarios. La aplicación proporcionará funcionalidades de registro, inicio de sesión y cierre de sesión para los usuarios, así como la capacidad de administrar la información de usuarios. El sistema implementará un sistema de verificación de correo electrónico, que garantizará la autenticidad de las cuentas de usuario registradas y brindará una capa adicional de seguridad. Además, se establecerá una relación muchos a muchos entre las tablas de usuarios. Para facilitar la gestión de los usuarios registrados, se desarrollará un CRUD (Crear, Leer, Actualizar y Eliminar) para la tabla de usuarios. Esto permitirá a los administradores del sistema realizar operaciones básicas sobre los usuarios, como crear nuevos perfiles, actualizar información existente y eliminar usuarios si es necesario. Asimismo,

se implementará un CRUD. La seguridad de la aplicación será una prioridad fundamental. Se implementarán medidas de autenticación y autorización adecuadas, como el almacenamiento de contraseñas cifradas y la asignación de roles con permisos específicos a los usuarios. Esto asegurará que solo los usuarios autorizados puedan acceder a determinadas funcionalidades y protegerá la integridad de los datos del sistema. El objetivo de este proyecto es proporcionar una solución robusta y eficiente para la gestión de usuarios, con una interfaz intuitiva y fácil de usar. Se realizarán pruebas exhaustivas para garantizar el correcto funcionamiento de todas las funcionalidades implementadas, y se documentará el proceso de desarrollo con el fin de facilitar el mantenimiento futuro y la comprensión del proyecto por parte de otros desarrolladores. Finalmente, el proyecto se desplegará en el servidor local gracias al uso de Apache como servidor web incorporado de XAMPP, asegurando su accesibilidad y rendimiento óptimo para los usuarios finales. Con estas características, se espera que la aplicación sea una herramienta efectiva para gestionar usuarios y roles en un entorno web de manera segura y eficiente.



7.2 Descripción de la metodología

El desarrollo de este proyecto se llevó a cabo siguiendo una metodología basada en avances semanales y utilizando el patrón de diseño Modelo-Vista-Controlador (MVC) como el principal eje de la arquitectura del sistema. La metodología de avances semanales permitió dividir el proyecto en tareas manejables y establecer objetivos alcanzables en cada semana. Cada avance semanal se enfocó en el desarrollo de funcionalidades específicas, corrección de errores o mejoras incrementales del sistema. Esta metodología proporcionó un marco de trabajo organizado y permitió un seguimiento y control efectivo del progreso del proyecto. El patrón de diseño Modelo-Vista-Controlador (MVC) se utilizó como la base para la estructura y organización del código. Este patrón separa la lógica de negocio y los datos (Modelo) de la presentación y la interacción con el usuario (Vista), a través de un controlador que actúa como intermediario entre ambos. Esta separación de responsabilidades facilitó la escalabilidad, mantenibilidad y reutilización del código, ya que cada componente tenía un propósito claro y acotado. En el enfoque MVC, el Modelo representó la capa de acceso a la base de datos, donde se definieron las tablas y relaciones utilizando el lenguaje de definición de datos de MySQL.

Además, se implementaron los modelos de datos en Laravel, que se encargaron de realizar las consultas y manipulación de datos en la base de datos. La Vista se encargó de la presentación de la información al usuario. Se desarrollaron plantillas y vistas utilizando el motor de plantillas de Laravel, que permitió separar el código HTML del resto de la lógica de la aplicación. Esto facilitó la creación de interfaces atractivas y responsivas, mejorando la experiencia del usuario. El Controlador fue responsable de manejar las solicitudes del usuario, interactuar con los modelos y coordinar la respuesta a través de las vistas correspondientes. Los controladores en Laravel se encargaron de recibir y procesar las solicitudes HTTP, gestionar la autenticación y autorización de los usuarios, así como coordinar la ejecución de las operaciones CRUD y otras funcionalidades específicas. Página 5 de 79 Además del modelo Vista-Controlador, se utilizaron otras características y funcionalidades de Laravel para agilizar el desarrollo y mejorar la eficiencia. Esto incluyó el uso de migraciones para controlar la estructura de la base de datos, el enrutamiento para gestionar las rutas de la aplicación, el sistema de autenticación de Laravel para el registro, inicio de sesión y cierre de sesión de usuarios, entre otros. En resumen, la metodología de avances semanales combinada con el enfoque Modelo-Vista-Controlador permitió un desarrollo estructurado y eficiente de la aplicación. El uso de Laravel y sus características facilitó la implementación de las funcionalidades requeridas y contribuyó a la creación de una aplicación web robusta, escalable y de fácil mantenimiento.

7.3 Descripción de las acciones realizadas

7.3.1 Marco teórico

Para el desarrollo de la investigación formativa se realizó una revisión bibliográfica de la que se rescatan los siguientes conceptos:

- **Laravel:** Laravel es un framework de desarrollo web de código abierto basado en PHP que sigue el patrón de diseño MVC (Modelo-Vista-Controlador). Proporciona una amplia gama de herramientas y bibliotecas para simplificar el desarrollo de aplicaciones web modernas. Con características como Eloquent ORM, migraciones de bases de datos, autenticación, enrutamiento y mucho más, Laravel ofrece una forma rápida y eficiente de construir aplicaciones robustas y escalables.
- **Framework:** Un framework es un conjunto de herramientas, bibliotecas y reglas que proporcionan una estructura para desarrollar aplicaciones. Los frameworks simplifican el proceso de desarrollo al ofrecer soluciones predefinidas para tareas comunes, como enrutamiento, manejo de bases de datos, autenticación y más. Al utilizar un framework, los desarrolladores pueden enfocarse en crear la lógica de negocios específica de la aplicación sin tener que preocuparse por la infraestructura subyacente.



- **Bootstrap:** Bootstrap es un framework de diseño front-end que proporciona una serie de estilos, componentes y plantillas CSS y JavaScript listos para usar. Con Bootstrap, los desarrolladores pueden crear interfaces de usuario responsivas y atractivas de manera rápida y sencilla. Al aprovechar las clases predefinidas de Bootstrap y su enfoque móvil primero, es posible crear sitios web que se adapten automáticamente a diferentes dispositivos y tamaños de pantalla.
- **XAMPP:** Xampp es un paquete de software gratuito y de código abierto que facilita la creación y gestión de un entorno de desarrollo web local. Incluye Apache como servidor web, MySQL como sistema de gestión de bases de datos, PHP y Perl como lenguajes de programación y otras herramientas como phpMyAdmin. Xampp es ampliamente utilizado por desarrolladores para crear y probar aplicaciones web en su computadora antes de desplegarlas en un servidor en línea.
- **MySQL:** MySQL es un sistema de gestión de bases de datos relacional de código abierto ampliamente utilizado. Es una opción popular para el almacenamiento y manipulación de datos en aplicaciones web y otros entornos. MySQL permite crear, consultar, modificar y administrar bases de datos y tablas, y es compatible con una variedad de lenguajes de programación y frameworks, incluido Laravel.
- **Vista:** En el contexto del patrón de diseño MVC, la vista es la parte de una aplicación web que se encarga de mostrar la interfaz de usuario al usuario final. Utiliza plantillas y lógica para presentar los datos de una manera comprensible y atractiva.
- **Controlador:** En el patrón de diseño MVC, el controlador es la parte de una aplicación web que maneja las solicitudes del usuario y coordina las acciones que deben realizarse. Es responsable de recibir las entradas del usuario, interactuar con el modelo y seleccionar la vista adecuada para mostrar los resultados al usuario.
- **Modelo:** En el patrón de diseño MVC, el modelo representa la estructura de datos y las reglas de negocio de una aplicación. Es responsable de interactuar con la base de datos, recuperar y guardar datos, y aplicar la lógica de negocio necesaria para procesar las solicitudes del usuario.
- **Middleware:** El middleware en Laravel es una capa intermedia que permite agregar funcionalidades adicionales a las solicitudes HTTP antes o después de que sean manejadas por el controlador. Puede utilizarse para tareas como la autenticación, el registro de solicitudes, la compresión de respuestas y mucho más.
- **CRUD:** CRUD es un acrónimo que representa las cuatro operaciones básicas para interactuar con datos en una base de datos: Create (crear), Read (leer), Update (actualizar) y Delete (eliminar). Los sistemas CRUD son fundamentales en el desarrollo web y permiten la manipulación de datos en aplicaciones.

- **Layout:** Un layout se refiere a la estructura y disposición general de una página web. Define la ubicación y el diseño de los elementos comunes de la interfaz de usuario, como la barra de navegación, el encabezado, el pie de página, entre otros.
- **Seeders:** Los seeders en Laravel son archivos que permiten agregar datos de prueba a la base de datos de manera automatizada. Son útiles para llenar la base de datos con información inicial y facilitar el desarrollo y pruebas de la aplicación.
- **Migraciones:** Las migraciones en Laravel son archivos que permiten crear y modificar la estructura de la base de datos de forma programática. Proporcionan una forma de realizar cambios en la base de datos de manera controlada y reversible.
- **Blade:** Blade es el motor de plantillas de Laravel que permite integrar lógica PHP en las vistas de una manera más sencilla. Proporciona directivas y sintaxis que facilitan la generación dinámica de contenido en las vistas.
- **Aplicación web:** Una aplicación web es un software que se ejecuta en un servidor y se accede a través de un navegador web. Proporciona funcionalidades y servicios a los usuarios finales y permite interactuar con ellos a través de una interfaz de usuario.
- **Frontend:** El frontend se refiere a la parte de una aplicación web que se ejecuta en el navegador del usuario. Incluye la interfaz de usuario y la lógica que se ejecuta en el cliente para interactuar con el servidor y mostrar el contenido al usuario.
- **Backend:** El backend se refiere a la parte de una aplicación web que se ejecuta en el servidor. Incluye el controlador, el modelo, las rutas y otras partes de la lógica de negocio que se encargan de procesar las solicitudes del usuario y gestionar los datos en la base de datos.

7.3.2 Desarrollo de la aplicación web en Laravel 10

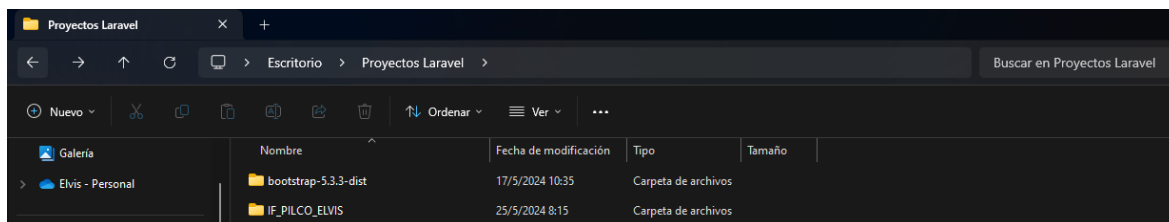
A. Creación y configuración inicial del proyecto en Laravel

En primera instancia se creó el proyecto de Laravel 10 ejecutando el símbolo del sistema (CMD) dentro de la carpeta en la que se desea crear el proyecto mediante el comando:

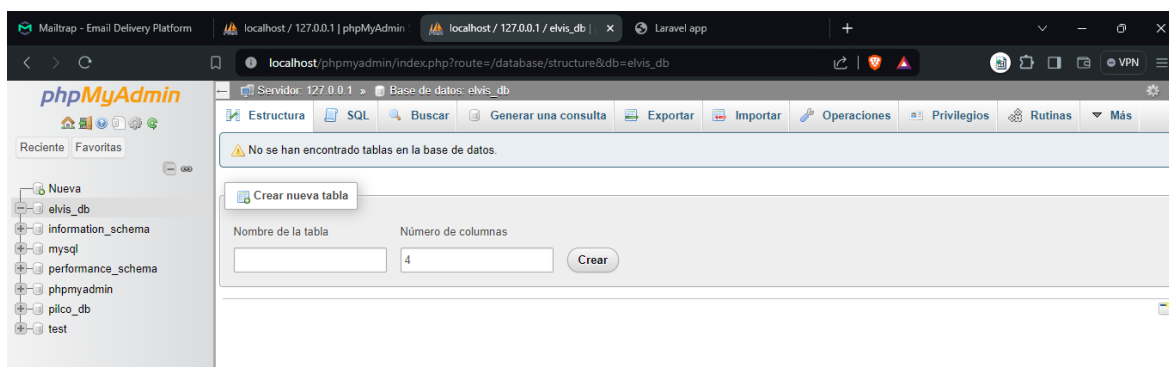
```
composer create-project laravel/laravel {nombre_proyecto}
```

```
C:\Users\WinUser\Desktop\Proyectos Laravel>composer create-project laravel/laravel IF_PILCO_ELVIS
Creating a "laravel/laravel" project at "./IF_PILCO_ELVIS"
Installing laravel/laravel (v11.0.9)
- Installing laravel/laravel (v11.0.9): Extracting archive
Created project in C:\Users\WinUser\Desktop\Proyectos Laravel\IF_PILCO_ELVIS
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
```


Al ejecutar este comando se instalan los recursos necesarios por el framework para el manejo del proyecto gracias a Composer. Todo este proceso se hace en una nueva carpeta con el nombre del proyecto especificado en la ruta en que se ejecutó el comando



Adicionalmente se creó la base de datos con la que se va a relacionar el proyecto. Para el proyecto se utilizó MySQL con phpMyAdmin como SGBD dado que se encuentra integrado en XAMPP junto con Apache que se usó como servidor web para ejecutar el proyecto en el servidor local .



Nótese que no se crearon tablas en la BD pues se implementaron mediante migraciones de laravel. Además, para un manejo simple y organizado se usó Visual Studio Code como IDE para el desarrollo del proyecto. Ahora bien, para la conexión con la BD se modificó el archivo .env modificando los parámetros DB_CONNECTION, DB_HOST, DB_PORT, DB_DATABASE, DB_USERNAME, DB_PASSWORD, de acuerdo a la información de la BD

```
18 LOG_STACK=single
19 LOG_DEPRECATED_CHANNEL=null
20 LOG_LEVEL=debug
21
22 DB_CONNECTION=mysql
23 DB_HOST=127.0.0.1
24 DB_PORT=3306
25 DB_DATABASE=elvis_db
26 DB_USERNAME=root
27 DB_PASSWORD=
28
29 SESSION_DRIVER=database
30 SESSION_LIFETIME=120
31 SESSION_ENCRYPT=false
```


Cabe recalcar que al iniciar el proyecto en Laravel se crea por defecto la tabla users y los recursos asociados(modelo y migraciones) con configuraciones por defecto. Así pues, en el archivo correspondiente a la migración de esta tabla se debe especificar la estructura de la tabla de acuerdo con las especificaciones del proyecto. En este caso se agregó el campo username adicional a los ya definidos indicando que este campo debe ser único.

```
2014_10_12_000000_create_users_table.php x
state_users_table.php > class@anonymousDfile:///c:/Users/MASTER/Desktop/PROYECTOS LARAVEL/GUAPULEMA_IF/database/migrations/20... >
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('users', function (Blueprint $table) {
15             //personalizacion de la tabla users
16             $table->id();
17             $table->string('name')->nullable();
18             $table->string('email')->unique();
19             $table->string('username')->unique();
20             $table->string('password');
21             $table->rememberToken();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      */
28     public function down(): void
29     {
30         Schema::dropIfExists('users');
31     }
32 }
```

Una vez definida la estructura de la tabla se ejecutó la migración de la tabla users como se muestra enseguida:

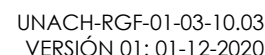
```
PS C:\Users\WinUser\Desktop\Proyectos Laravel\IF_PILCO_ELVIS> php
artisan migrate

INFO Preparing database.

Creating migration table ..... 134.20ms DONE

INFO Running migrations.
```

Nótese que tras la ejecución de las migraciones las tablas se crean automáticamente en la base de datos



```

app > Models > User.php > App\Models\User > setPasswordAttribute
1  <?php
2
3  namespace App\Models;
4
5  // use Illuminate\Contracts\Auth\MustVerifyEmail;
6  use Illuminate\Database\Eloquent\Factories\HasFactory;
7  use Illuminate\Foundation\Auth\User as Authenticatable;
8  use Illuminate\Notifications\Notifiable;
9  use Laravel\Sanctum\HasApiTokens;
10
11 2 references | 0 implementations
12 class User extends Authenticatable
13 {
14     use HasApiTokens, HasFactory, Notifiable;
15
16     /**
17      * The attributes that are mass assignable.
18      *
19      * @var array<int, string>
20      */
21     0 references
22     protected $fillable = [
23         'name',
24         'username',
25         'email',

```

```

26
27 /**
28  * The attributes that should be hidden for serialization.
29  *
30  * @var array<int, string>
31  */
32 0 references
33  protected $hidden = [
34      'password',
35      'remember_token',
36  ];
37
38 /**
39  * The attributes that should be cast.
40  *
41  * @var array<string, string>
42  */
43 1 reference
44  protected $casts = [
45  ];
46
47 //Use fo mutators for password encryption
48 0 references | 0 overrides
49  public function setPasswordAttribute($value){
50      $this->attributes['password'] = bcrypt($value);
51  }
52

```

Debido a que se aplicó el modelo vista-controlador, para implementar esta funcionalidad se creó el controlador RegisterController para el manejo. Aun así, para garantizar que las solicitudes dirigidas a este controlador solo se puedan manipular por el controlador destinado se creó un objeto Request llamado RegisterRequest.

En este recurso se debe modificar el método authorize como true y el método rules, pues en este último se definen las reglas de validación asociadas a users al ejecutar el registro.

```

User.php RegisterController.php RegisterRequest.php X
app > Http > Requests > RegisterRequest.php > App\Http\Requests\RegisterRequest > rules
1 <?php
2
3 namespace App\Http\Requests;
4
5 use Illuminate\Foundation\Http\FormRequest;
6
7 0 references | 0 implementations
8 class RegisterRequest extends FormRequest
9 {
10     /**
11      * Determine if the user is authorized to make this request.
12      */
13     0 references | 0 overrides
14     public function authorize(): bool
15     {
16         return true;
17     }
18
19     /**
20      * Get the validation rules that apply to the request.
21      *
22      * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array|string>
23      */
24

```

Una vez definida el request asociado al controlador RegisterController, este último se creó con el propósito de definir los métodos que se van a usar para la implementación de la funcionalidad

Así pues, en la configuración de controlador RegisterController se definieron las funciones show y register. La primera se encarga de redirigir al usuario a la vista para el registro y en caso de encontrarse ya logeado lo redirige a la vista home del usuario. Por otro lado, la función register toma como parámetro una solicitud de tipo RegisterRequest creado

previamente (información que pasó las validaciones) y crea al usuario con el método create que ofrece Laravel para después redirigirlo a la vista de logeo para que inicie sesión.

```

LoginController.php RegisterController.php X
app > Http > Controllers > RegisterController.php > PHP > App\Http\Controllers\RegisterController
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Http\Requests\RegisterRequest;
6  use App\Models\User;
7  use Illuminate\Http\Request;
8  use Illuminate\Support\Facades\Auth;
9
10 3 references | 0 implementations
11 class RegisterController extends Controller
12 {
13     1 reference | 0 overrides
14     public function show(){
15         //validation for redirect to home in case it's already logged in
16         if(Auth::check()){
17             return redirect('/home');
18         }
19         return view('auth.register');
20     }
21
22 //use of an request object for specific controller
23 //validation is done in request
24 1 reference | 0 overrides
25 public function register(RegisterRequest $request){
26     $user = User::create($request->validated());
27     return redirect('/login')->with('success', 'Cuenta creada exitosamente');
28 }
29

```

Posteriormente se crearon las vistas definidas en los diferentes métodos siendo la primera la vista correspondiente al registro llamada register.blade.php dentro de una carpeta llamada auth en vistas con una interfaz básica.

```

User.php RegisterController.php web.php register.blade.php X RegisterRe
resources > views > auth > register.blade.php
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Register</title>
8  </head>
9  <body>
10     <form action="/register" method="POST">
11         @csrf
12         <input type="text" name="username">
13         <input type="email" name="email">
14         <input type="password" name="password">
15         <input type="password" name="password_confirmation">
16         <input type="submit" value="Registrarse">
17     </form>
18 </body>
19 </html>

```

Además, para poder navegar entre las vistas en el navegador se definieron las rutas de los métodos especificados en RegisterController en el archivo web.php

```

User.php RegisterController.php web.php x register.blade.php
routes > web.php
12 | routes are loaded by the RouteServiceProvider and all of them will
13 | be assigned to the "web" middleware group. Make something great!
14 |
15 */
16
17 Route::get('/', function () {
18     return view('welcome');
19 });
20
21 Route::get('/register', [RegisterController::class, 'show']);
22
23 Route::post('/register', [RegisterController::class, 'register']);

```

Así pues, se definieron la ruta register de modo que si es una petición de tipo get se ejecute la función show de RegisterController que muestra el formulario de registro; y si es una petición post se ejecute la función register de dicho controlador que guarda al usuario tras validar su información.

Finalmente se verificó que la información ingresada se registró en la base de datos

SELECT * FROM 'users'

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

☐ Mostrar todo | Número de filas: 25 | Filtrar filas:

Opciones extra

	id	name	email	username	password	remember_tok
<input type="checkbox"/> Editar Copiar Borrar	1	NULL	p1@gmail.com	prueba1	\$2y\$10\$NLOOy5bCOYHJ6zY4TdJasOz/sthlveDMUfrFQWVG87...	NULL

C. Funcionalidad de inicio de sesión (Login)

De manera homóloga a la funcionalidad que en el registro de usuarios, para el logeo de estos se creó un objeto request llamado LoginRequest que servirá para manipular las solicitudes en el Logincontroller.

Así pues, se modificó el método authorize definiendo el return como true y el método rules especificando a los campos username y password como requeridos para poder iniciar sesión. Además, se definieron los métodos isEmail ,que determina si se envió un correo como credencil de inicio de sesión, y getCredentials que toma el campo que se envía en el formulario de Login y dependiendo de si se envió un username o un email genera credenciales para el inicio de sesión del usuario.

```

RegisterController.php  LoginRequest.php  web.php  register.blade.php  RegisterRequest.php
app > Http > Requests > LoginRequest.php > PHP > App\Http\Requests\LoginRequest > getCredentials
1  <?php
2
3  namespace App\Http\Requests;
4
5  use Illuminate\Foundation\Http\FormRequest;
6  use Illuminate\Contracts\Validation\Factory as ValidationFactory;
7
8  0 references | 0 implementations
9  class LoginRequest extends FormRequest
10 {
11     /**
12      * Determine if the user is authorized to make this request.
13      */
14     0 references | 0 overrides
15     public function authorize(): bool
16     {
17         return true;
18     }
19     /**
20      * Get the validation rules that apply to the request.
21      *
22      * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array|string>

```

```

public function rules(): array
{
    return [
        'username' => 'required',
        'password' => 'required'
    ];
}

//function to let login with username or email
0 references | 0 overrides
public function getCredentials(): array
{
    $username = $this->get('username');
    //in case an email was sent
    if($this->isEmail($username)){
        return [
            'email' => $username,
            'password' => $this->get('password')
        ];
    }
    //in case an username was sent
    return $this->only('username', 'password');
}

//Function to determinate an email was sent
1 reference | 0 overrides

```

```

45 public function isEmail($value){
46     $factory = $this->container->make(ValidationFactory::class);
47     //returns true or false
48     return !$factory->make(['username' => $value], ['username' => 'email'])->fails();
49 }
50 }

```

Seguido se creó el LoginController que toma las peticiones ya procesadas por el objeto request asociado y creará el inicio de sesión para un usuario siempre que sus credenciales sean válidas.

Es así como en este controlador se definieron las funciones show, que redirige a la vista que contiene el formulario de logeo y en caso de estar ya logeado redirige al home del usuario;

y login que toma una petición del tipo LoginRequest y la procesa con los métodos ya definidos para redirigir a la misma vista de logeo si las credenciales no son válidas, y en caso de serlo crea un usuario con credenciales autenticadas cuya sesión será creada y redirige al usuario a la vista home de su perfil.

```

LoginController.php x RegisterController.php
app > Http > Controllers > LoginController.php > PHP > App\Http\Controllers\LoginController
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Http\Requests\LoginRequest;
6  use Illuminate\Http\Request;
7  use Illuminate\Support\Facades\Auth;
8
9  3 references | 0 implementations
10 class LoginController extends Controller
11 {
12     1 reference | 0 overrides
13     public function show(){
14         //validation for redirect to home in case it's already logged in
15         if(Auth::check()){
16             return redirect('/home');
17         }
18         return view('auth.login');
19     }
20 }

```

```

> Console
> Exceptions
> Http
  > Controllers
    > Controller.php
    > HomeController.php
    > LoginController.php
    > RegisterController.php
  > Middleware
  > Requests
    > LoginRequest.php
    > RegisterRequest.php
  > Kernel.php
> Models
> Providers
> bootstrap
> config
> database
> public
> resources
> css
20 //auth credential logic for fields returned in request
21 1 reference | 0 overrides
22 public function login(LoginRequest $request){
23     $credentials = $request->getCredentials();
24
25     //validation of user or email
26     if(!Auth::validate($credentials)){
27         return redirect()->to('/login')->withErrors('auth.failed');
28     }
29
30     //creates an user with credentials auth
31     $user = Auth::getProvider()->retrieveByCredentials($credentials);
32
33     //login and session creations auto with laravel
34     Auth::login($user);
35
36     return $this->authenticated($request, $user);
37 }
38
39 1 reference | 0 overrides
40 public function authenticated(Request $request, $user){
41     return redirect('/home');
42 }

```

Después se creó la vista login correspondiente al formulario de inicio de sesión con una interfaz básica de testeo


```

resources > views > auth > login.blade.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Login</title>
8 </head>
9 <body>
10     <form action="/login" method="POST">
11         @csrf
12         username/email
13         <input type="text" name="username">
14         password
15         <input type="password" name="password">
16         <input type="submit" value="Login">
17     </form>
18 </body>
19 </html>

```

Finalmente se definieron las rutas para el manejo del Login en web.php de modo que si la petición es del tipo get se ejecuta el método show definido en el LoginController que muestra el formulario de inicio de sesión, y si es del tipo post se ejecuta el método login que valida las credenciales y crea la sesión mostrando el home del usuario.

```

login.blade.php web.php x HomeController.php index.blade.php
routes > web.php
2
3 use App\Http\Controllers\HomeController;
4 use App\Http\Controllers>LoginController;
5 use App\Http\Controllers\RegisterController;
6 use Illuminate\Support\Facades\Route;
7
8 /*
9 |-----
10 | Web Routes
11 |-----
12 |
13 | Here is where you can register web routes for your application. These
14 | routes are loaded by the RouteServiceProvider and all of them will
15 | be assigned to the "web" middleware group. Make something great!
16 |
17 */
18
19 Route::get('/', function () {
20     return view('welcome');
21 });
22
23 Route::get('/register', [RegisterController::class, 'show']);
24 Route::post('/register', [RegisterController::class, 'register']);
25
26 Route::get('/login', [LoginController::class, 'show']);
27 Route::post('/login', [LoginController::class, 'login']);
28
29 Route::get('/home', [HomeController::class, 'index']);

```

Nótese que en las rutas también se definió a /home con una petición GET que ejecuta el método index de un controlador llamado HomeController. Este método se encarga de mostrar el dashboard o home de cada usuario y se implementó como se describe a continuación.

D. Home del usuario

Primero se creó el controlador HomeController de acuerdo con el MVC de proyecto.

En este controlador se definió el método index que redirige a la vista index que contiene el diseño del home.

```
GUAPULEMA_IF
LoginRequest.php LoginController.php login.blade.php web.php HomeController.php X regist ...
app > Http > Controllers > HomeController.php > ...
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class HomeController extends Controller
8 {
9     public function index(){
10         return view('home.index');
11     }
12 }
```

Así pues, se creó la vista index necesaria en una carpeta llamada home en vistas con un diseño básico. Dentro de esta vista mediante el incruste de cláusulas de php se personalizó el contenido a mostrar dependiendo si el usuario se encuentra autenticado(@auth) o no (@guest)

```
LogoutController.php web.php index.blade.php X
resources > views > home > index.blade.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Home</title>
8 </head>
9 <body>
10     <h1>HOME</h1>
11
12     @auth
13         <p>BIENVENIDO {{auth()->user()->name ?? auth()->user()->username}} , estás autenticac
14         <p><a href="/logout">Cerrar sesion</a></p>
15     @endauth
16
17     @guest
18         <p>Se requiere <a href="/login">iniciar sesion</a></p>
19     @endguest
20
21 </body>
22 </html>
```

E. Funcionalidad Log out (Cerrar sesión)

Para implementar la funcionalidad de Log out se creó en primera instancia el controlador para este cometido llamado LogoutController.

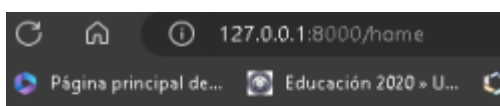
Dentro de este controlador se implementó el método logout que se encarga de liberar el flujo de la sesión mediante el método flush que brinda Laravel y cierra la sesión con el método logout de manera similar. Además, tras cerrar la sesión se redirige a la ruta de login de modo que se muestre el formulario de inicio de sesión.

```
LogoutController.php X
app > Http > Controllers > LogoutController.php > PHP > App\Http\Controllers\LogoutController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\Auth;
7  use Illuminate\Support\Facades\Session;
8
9  class LogoutController extends Controller
10 {
11     public function logout()
12     {
13         //libera el flujo de la sesion
14         Session::flush();
15
16         Auth::logout();
17
18         return redirect()->to('/login');
19     }
20 }
```

Seguido a ello, se creó la ruta de logout para que al hacer una petición GET a la ruta se ejecute el método logout definido en LogoutController. En este caso no se creó una vista pues no fue necesario considerando que al cerrar sesión se redirige a la vista de login.

```
LogoutController.php X web.php X index.blade.php
routes > web.php
14  Here is where you can register web routes for your application. These
15  routes are loaded by the RouteServiceProvider and all of them will
16  be assigned to the "web" middleware group. Make something great!
17
18  */
19
20  Route::get('/', function () {
21      return view('welcome');
22  });
23
24  Route::get('/register', [RegisterController::class, 'show']);
25  Route::post('/register', [RegisterController::class, 'register']);
26
27  Route::get('/login', [LoginController::class, 'show']);
28  Route::post('/login', [LoginController::class, 'login']);
29
30  Route::get('/home', [HomeController::class, 'index']);
31
32  Route::get('/logout', [LogoutController::class, 'logout']);
```

Finalmente se verificó cerrando sesión del usuario cuya sesión se inició previamente.



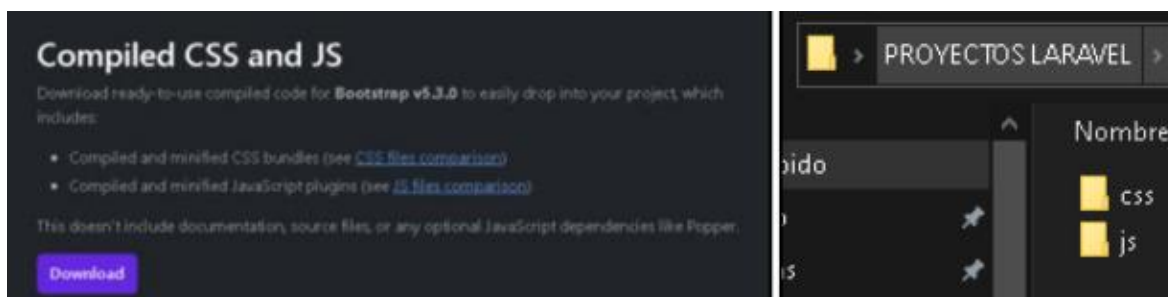
HOME

BIENVENIDO juanito , estás autenticado

[Cerrar sesion](#)

F. Estilización de las vistas con Bootstrap

Tras probar la funcionalidad de la aplicación se utilizó Bootstrap para la estilización de las vistas presentadas en el proyecto. Es por esta razón que desde la página oficial de Bootstrap se descargaron los archivos compilados relativos a CSS y JS y se añadieron en una carpeta llamada assets dentro del directorio public del proyecto.



Una vez agregado los recursos de Bootstrap al proyecto se crearon dos vistas llamadas `app-master.blade.php` y `auth-master.blade.php` dentro una carpeta llamada `layouts` en `views`. La vista `app-master` servirá como plantilla de la página que se mostrará en el home de la aplicación. Nótese que en la estructura se encuentra incorporado las dependencias core de `css` y `js` de Bootstrap para su uso en el proyecto especificando la ruta en la que se guardaron los archivos compilados. Es esta incorporación aquella que permitirá el uso de estilos de Bootstrap mediante el manejo de clases en futuras vistas

```
app-master.blade.php X
resources > views > layouts > app-master.blade.php
1 <!(DOCTYPE html)
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <meta name="description" content="">
7   <title>Login app</title>
8
9   <!-- Bootstrap core CSS -->
10  <link href="{{ url('assets/css/bootstrap.min.css') }}" rel="stylesheet">
11
12  <style>
13    .bd-placeholder-img {
14      font-size: 1.125rem;
15      text-align: middle;
16      -webkit-user-select: none;
17      -moz-user-select: none;
18      user-select: none;
19    }
20
21    @media (min-width: 768px) {
22      .bd-placeholder-img-lg {
23        font-size: 3.5rem;
24      }
25    }
26  </style>
27
28 </head>
29 <body>
30
31  @include('layouts.partials.navbar')
32
33  <main class="container">
34    @yield('content')
35  </main>
36  <!-- Bootstrap link js -->
37  <script src="{{ url('assets/js/bootstrap.bundle.min.js') }}"></script>
38
39 </body>
40 </html>
```

Asimismo, la vista `auth-master` corresponde a la plantilla de `html` y estilos aplicados en los formularios de registro e inicio de sesión, siendo la incorporación de los archivos de `css` y `js` de Bootstrap mediante la especificación de su ubicación en el proyecto, aquellos que permiten el uso de estos estilos en las vistas descritas.

```

app-master.blade.php  auth-master.blade.php
resources > views > layouts > auth-master.blade.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <meta name="description" content="">
7   <title>Signin Template - Bootstrap v5.1</title>
8   <!-- Bootstrap core CSS -->
9   <link href="{{ url('assets/css/bootstrap.min.css') }}" rel="stylesheet">
10  <style>
11    body{
12      width: 100%;
13      height: 100vh;
14      display: flex;
15      align-items: center;
16      justify-content: center;
17    }
18
19    .form-signin{
20      width: 400px;
21    }
22  </style>
23 </head>
24 <body class="text-center">
25   <main class="form-signin">
26     @yield('content')
27   </main>
28 </body>
29 </html>

```

Adicionalmente se creó una vista llamada navbar.blade.php dentro de la carpeta partials en layouts que tenía en su contenido el código basado en la documentación de Bootstrap para la creación de una barra de navegación como se muestra a continuación:

```

app-master.blade.php  navbar.blade.php  index.blade.php
resources > views > layouts > partials > navbar.blade.php
1 <nav class="navbar navbar-expand-lg bg-body-tertiary">
2   <div class="container-fluid">
3     <a class="navbar-brand" href="#">Navbar</a>
4     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#
5       <span class="navbar-toggler-icon"></span>
6     </button>
7     <div class="collapse navbar-collapse" id="navbarSupportedContent">
8       <ul class="navbar-nav me-auto mb-2 mb-lg-0">
9         <li class="nav-item">
10          <a class="nav-link active" aria-current="page" href="#">Home</a>
11        </li>
12        <li class="nav-item">
13          <a class="nav-link" href="#">Link</a>
14        </li>
15        <li class="nav-item dropdown">
16          <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown"
17            >Dropdown
18          </a>
19          <ul class="dropdown-menu">
20            <li><a class="dropdown-item" href="#">Action</a></li>
21            <li><a class="dropdown-item" href="#">Another action</a></li>
22            <li><hr class="dropdown-divider" /></li>
23            <li><a class="dropdown-item" href="#">Something else here</a></li>
24          </ul>
25        </li>
26        <li class="nav-item">
27          <a class="nav-link disabled">Disabled</a>
28        </li>
29      </ul>
30    </div>
31  </div>
32  <form class="d-flex" role="search">

```

Nótese que este código base se debe personalizar de acuerdo con el contenido que se quiera mostrar, en el caso puntal de la práctica se colocaron dos campos llamados Home y Perfil. Ahora bien, una vez ya creados las vistas externas que nos servirán para evitar redundancia y la incorporación de estilos de manera modular se procedió a personalizar las vistas ya definidas. Para ello se inició con la modificación de la vista index en cuyo contenido se especifica el contenido a mostrar si el usuario está autenticado (@auth) o no (@guest)

```
index.blade.php x
resources > views > home > index.blade.php
1 @extends('layouts.app-master')
2
3 @section('content')
4     <div class="bg-light p-5 rounded">
5         @auth
6             <h1>Dashboard</h1>
7             <p class="lead">Solo usuarios autenticados pueden ver este contenido.</p>
8             <p>BIENVENIDO {{auth()->user()->name ?? auth()->user()->username}} , estás autenticado</p>
9             <p><a href="/logout">Log out</a></p>
10        @endauth
11
12        @guest
13            <h1>Homepage</h1>
14            <p class="lead">HOME PAGE. Porfavor <a href="/login">inicia sesion</a> para tener acceso</p>
15        @endguest
16    </div>
17 @endsection
```

Sumado a lo anterior se creó una vista messages.blade.php con el fin de mostrar mensajes en caso de ocurrir errores en el proceso de autenticación. Así pues, en la vista se define una regla que verifique si hay errores y los imprima; y otra regla que verifica que la sesión sea exitosa e imprime un mensaje de éxito enviado por parámetro.

```
index.blade.php login.blade.php auth-master.blade.php messages.blade.php x
resources > views > layouts > partials > messages.blade.php
1 @if(isset($errors) && count($errors) > 0)
2     <div class="alert alert-warning" role="alert">
3         <ul class="list-unstyled mb-0">
4             @foreach($errors->all() as $error)
5                 <li>{{ $error }}</li>
6             @endforeach
7         </ul>
8     </div>
9 @endif
10
11 @if(Session::get('success', false))
12     <?php $data = Session::get('success'); ?>
13     @if (is_array($data))
14         @foreach ($data as $msg)
15             <div class="alert alert-success" role="alert">
16                 <i class="fa fa-check"></i>
17                 {{ $msg }}
18             </div>
19         @endforeach
20     @else
21         <div class="alert alert-success" role="alert">
22             <i class="fa fa-check"></i>
23             {{ $data }}
24         </div>
25     @endif
26 @endif
```

Para la personalización de la vista de inicio de sesión y registro se usó código de Bootstrap proporcionado en su página oficial. Nótese además que dentro de estas vistas que definen la estructura visual de los formularios, se incorporaron las vistas creadas en pasos interiores a priori de que se incruste indirectamente el código en la vista.

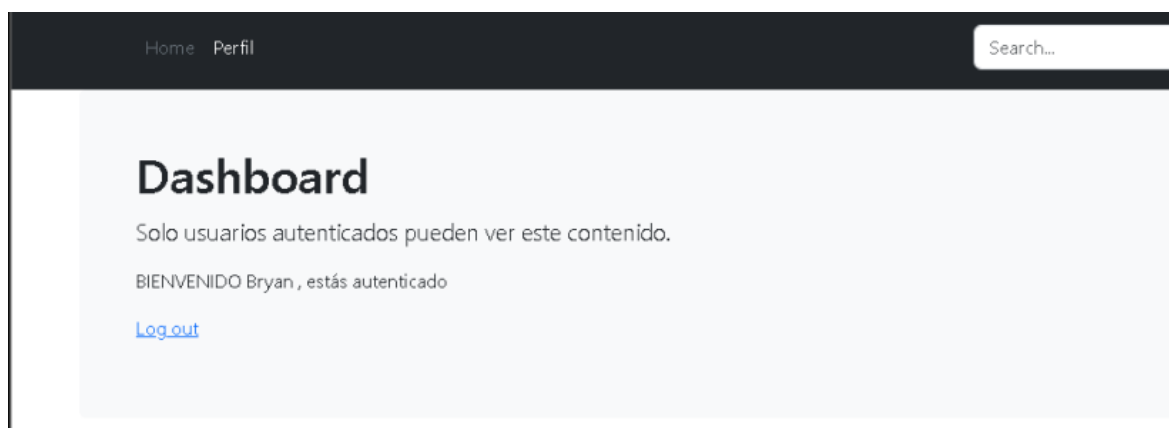
```
login.blade.php X
resources > views > login.blade.php
1 @extends('layouts.auth-master')
2
3 @section('content')
4 <form method="POST" action="/login">
5
6 <input type="hidden" name="_token" value="{{ csrf_token() }}" />
7
8
9 <h1 class="h3 mb-3 fw-normal">Login</h1>
10
11 @include('layouts.partials.messages')
12
13 <div class="form-group form-floating mb-3">
14 <input type="text" class="form-control" name="username" value="{{ old('username') }}" placeholder="Us
15 <label for="floatingName">Email or Username</label>
16 @if ($errors->has('username'))
17 <span class="text-danger text-left">{{ $errors->first('username') }}</span>
18 @endif
19 </div>
20
21 <div class="form-group form-floating mb-3">
22 <input type="password" class="form-control" name="password" value="{{ old('password') }}" placeholder
23 <label for="floatingPassword">Password</label>
24 @if ($errors->has('password'))
25 <span class="text-danger text-left">{{ $errors->first('password') }}</span>
26 @endif
27 </div>
28
29 <div class="card-subtitle mb-2 text-body-secondary"><a href="/register">¿No tienes una cuenta?</a></div>
30 <button class="w-100 btn btn-lg btn-primary" type="submit">Iniciar sesión</button>
31
32 </form>
33 @endsection
```

Del mismo modo también se modificó la vista principal welcome que se muestra al iniciar la aplicación, borrando el contenido dado por Laravel a uno propio y personalizado como se observa:

```
welcome.blade.php X
resources > views > welcome.blade.php
1 <!doctype html>
2 <html lang="en">
3 <head>
4 <!-- Required meta tags -->
5 <meta charset="utf-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7
8 <!-- Bootstrap CSS -->
9 <link href="assets/css/bootstrap.min.css" rel="stylesheet">
10
11 <style>
12 body {
13 background: url('https://sgc.unach.edu.ec/wp-content/uploads/2020/06/unach-edificios.jpg');
14 -webkit-background-size: cover;
15 -moz-background-size: cover;
16 -o-background-size: cover;
17 background-size: cover;
18 animation: fadeIn 3s ease-in-out 1 both;
19 }
20
21 .content {
22 height: 100vh;
23 display: flex;
24 flex-direction: column;
25 justify-content: center;
26 align-items: center;
27 text-align: center;
28 }
```

```
resources > views > welcome.blade.php
30 @keyframes fadeIn {
31 0% {opacity: 0;}
32 100% {opacity: 1;}
33 }
34 </style>
35
36 <title>Welcome</title>
37 </head>
38 <body>
39 <div class="content">
40 <h1 class="display-4 text-light">¡Bienvenido a Nuestro Sitio Web!</h1>
41 <p class="lead text-light">Este es un lugar increíble para presentar tu información.
42
43 <div class="mt-4">
44 <a class="btn btn-light btn-lg mr-2" href="/login" role="button">Inicia Sesión</a>
45 <a class="btn btn-light btn-lg" href="/register" role="button">Regístrate</a>
46 </div>
47 </div>
48 <!-- Bootstrap Bundle with Popper -->
49 <script src="assets/js/bootstrap.bundle.min.js"></script>
50 </body>
51 </html>
```


Finalmente se verificó que se apliquen los estilos en las vistas y que la funcionalidad definida en los elementos permanezca y funcione correctamente.

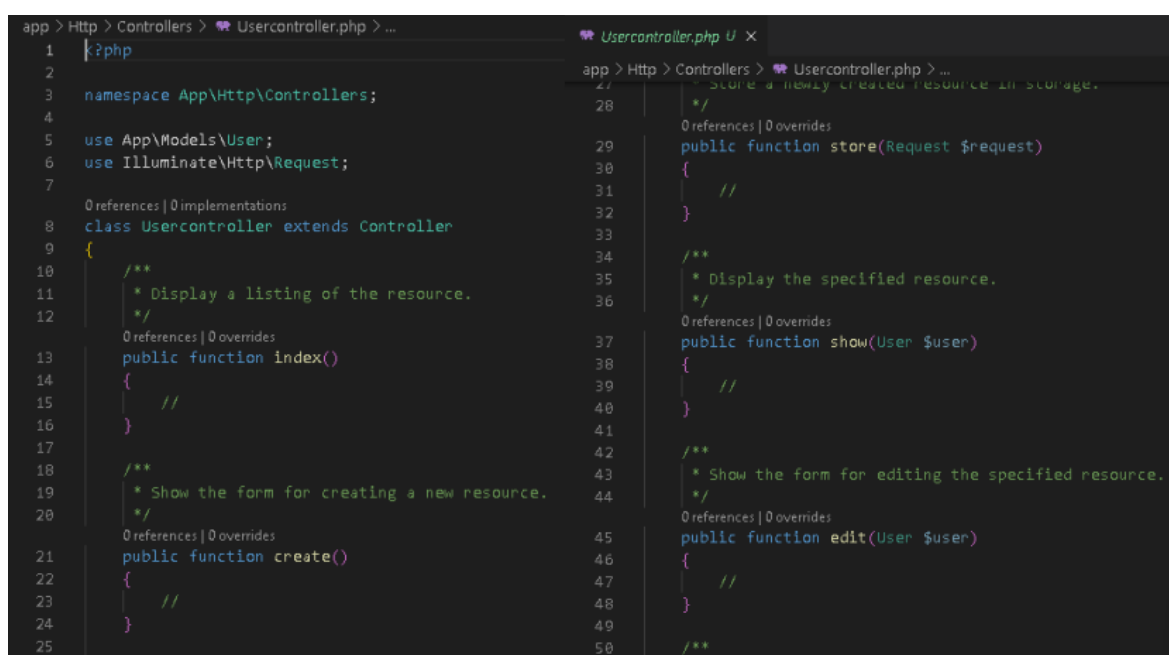


G. Funcionalidad CRUD Usuarios

i. Preliminares

Para la implementación de un CRUD relativo a la tabla Users que permita gestionar los usuarios en la aplicación se creó un controlador (UserController) que administre esta funcionalidad.

Nótese que en el comando para crear el controlador se especifica el modelo User que está asociado al controlador y el parámetro resource que implementa los métodos básicos para el manejo de un CRUD (index, create, store, show, edit, destroy), como se muestra a continuación:



Estos métodos se crearon como plantilla del controlador al usar el comando especificado y en estos métodos se implementó la lógica alrededor del manejo de usuarios para la aplicación. Posteriormente se modificaron las rutas en el archivo web.php como se hizo previamente con las funcionalidades de register, login y logout.

```
34
35 Route::resource('/user',UserController::class);
```

Además, para la creación de las vistas del CRUD se creó una vista llamada `template.blade.php` dentro de la carpeta `layouts` que sirvió como plantilla de las vistas de las funciones.

```
Usercontroller.php U • app-master.blade.php • template.blade.php U x • web.php M
resources > views > layouts > template.blade.php > html > body > main > script
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <meta name="description" content="">
7     <title@yield('title')</title>
8     <link href="{!! url('assets/css/bootstrap.min.css') !!}" rel="stylesheet">
9 </head>
10
11 <body>
12
13     <main>
14         @yield('content')
15         <script src="{!! url('assets/js/bootstrap.bundle.min.js') !!}"></script>
16     </main>
17 </body>
18 </html>
```

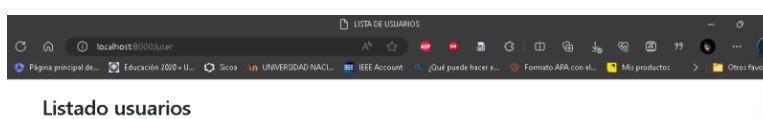
Así pues, en primer lugar se creó la vista `listar.blade.php` dentro de una carpeta llamada `users` en `views` incluyendo el contenido de la plantilla y un título para probar que las rutas estaban funcionando.

```
Usercontroller.php U • template.blade.php U • listar.blade.php U x • web.php M
resources > views > users > listar.blade.php > main > div.container.py-4 > h2
1 @extends('layouts.template')
2
3 @section('title','LISTA DE USUARIOS')
4
5 @section('contenido')
6
7 <main>
8     <div class="container py-4 ">
9         <h2>Listado alumnos</h2>
10    </div>
11 </main>
```

Asimismo, se modificó el método `create` de `UserController` para que devuelva la vista creada cuando este método sea llamado.

```
Usercontroller.php U x • template.blade.php U • listar.blade.php U • web.php M
app > Http > Controllers > Usercontroller.php > PHP Intelephense > Usercontroller > index
8 class UserController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     0 references | 0 overrides
14     public function index()
15     {
16         return view('users.listar');
```

Para comprobar que la funcionalidad no presenta errores se probó especificando la ruta `localhost:8000/users`



ii. Create

Para la creación de usuarios y su almacenamiento en la base de datos se reutilizó la funcionalidad previamente definida en el método register de RegisterController. Así pues, el método create y store no se crearon sino que se accedió a ellos con un botón a la ruta de registro de usuarios en la vista para listar.

Listado usuarios

Nuevo usuario

Registrarse

[Ya tengo una cuenta](#)
[Inicio](#)

Register

iii. Read

Para poder leer los registros de la tabla Users desde la vista para listar se modificó primero el método index del controlador para que envíe la información de la base de datos. Para que esto sea posible se creó una variable que almacena una instancia del modelo User con todos los registros de la tabla en la base de datos, que posteriormente se devuelve en conjunto con la vista para su manipulación.

```

13 public function index()
14 {
15     $users = User::all();
16     return view('users.listar', ['users'=>$users]);
17 }
    
```

Tras ello se modificó la vista listar creando una tabla que muestre los registros de la tabla Users haciendo uso de la variable que se envió en el controlador recorriendo cada uno de los registros y mostrando los atributos, así:

```

12 </a>
13
14 <table class="table table-hover">
15
16     <thead>
17         <tr>
18             <th>ID</th>
19             <th>Nombre</th>
20             <th>Nombre de Usuario</th>
21             <th>Rol</th>
22             <th>Correo</th>
23             <th></th>
24             <th></th>
25         </tr>
26     </thead>
27
28     <tbody>
29         <foreach($users as $user)>
30             <tr>
31                 <td>{{ $user->id }}</td>
32                 <td>{{ $user->name }}</td>
33                 <td>{{ $user->username }}</td>
34                 <td>{{ $user->rol }}</td>
35                 <td>{{ $user->email }}</td>
36                 <td></td>
37                 <td></td>
38             </tr>
39         </foreach>
40     </tbody>
    
```

Finalmente se accedió a la ruta del método listar para verificar que los datos de la base de datos se muestren de acuerdo con las configuraciones especificadas.

Listado usuarios

[Nuevo usuario](#)

ID	Nombre	Nombre de Usuario	Rol	Correo
1		prueba1	USUARIO	p1@gmail.com
2		juanito	USUARIO	juanito@gmail.com

v. Update

De manera similar a lo anterior se modificó primero el método edit del controlador de modo que retorne una vista que contenga el formulario para editar y una variable que contenga el registro con el id especificado en el parámetro de dicho método.

```
0 references | 0 overrides
public function edit($id)
{
    $user= User::find($id);
    return view('users.edit', ['user'=>$user]);
}
```

Además, se agregaron los botones Regresar, que redirecciona a la vista listar, y Guardar que servirá para guardar la información de la base de datos al implementar la funcionalidad de update. Finalmente se modificó la vista para listar de modo que en cada fila creada por registro se muestre un botón que redirija a la ruta /users/id/edit (como se especificó en la tabla de rutas) usando el atributo id de cada registro.

```
<tbody>
  @foreach($users as $user)
    <tr>
      <td>{{ $user->id }}</td>
      <td>{{ $user->name }}</td>
      <td>{{ $user->username }}</td>
      <td>USUARIO</td>
      <td>{{ $user->email }}</td>
      <td>
        <a href="{{url('users/' . $user->id . '/edit')}}" class="btn btn-warning">
          Editar
        </a>
      </td>
    </tr>
  @endforeach
</tbody>
```

Listado usuarios

[Nuevo usuario](#)

ID	Nombre	Nombre de Usuario	Rol	Correo	
1		prueba1	USUARIO	p1@gmail.com	Editar

Una vez definida la lógica tras update se creó la vista update.blade.php usando la plantilla definida a priori y con contenido personalizado y estilos de Bootstrap para que en una tarjeta se muestre la información con los nuevos campos, y un botón para volver a la lista de usuarios.

Actualización de usuario exitosa

El usuario ha sido actualizado correctamente.



prueba1

Nombre: EDITAR

Username: prueba1

Email: p1@gmail.com

Otra manera de verificar los cambios fue al examinar la base de datos y al regresar a la lista de usuarios.

Listado usuarios

Nuevo usuario

ID	Nombre	Nombre de Usuario	Rol	Correo	
1	EDITAR	prueba1	USUARIO	p1@gmail.com	Editar
2		juanito	USUARIO	juanito@gmail.com	Editar

v.Delete

Para poder eliminar registros en primera instancia se modificó el método destroy de UserController para que reciba como parámetro el id del registro a eliminar, lo busque y almacene en una variable, y mediante el uso del método delete se elimine el registro. Finalmente se redirige hacia la vista que muestra lista de usuarios.

```
0 references | 0 overrides  
public function destroy($id)  
{  
    $user = User::find($id);  
    $user->delete();  
    return redirect('users');  
}
```

Seguido, se modificó la vista para listar agregando la opción de eliminar. Para ello fue necesario crear un formulario POST que redirija a /users/id y que en su contenido especifique que el método de la petición es DELETE.

```
<td>
  <form action="{url('users/'. $user->id)}" method="post">
    @method('DELETE')
    @csrf
    <button type="submit" class="btn btn-danger btn-sm">Eliminar</button>
  </form>
</td>
```

Finalmente se verificó que la funcionalidad se ejecute correctamente al eliminar el segundo registro.

Listado usuarios

Nuevo usuario					
ID	Nombre	Nombre de Usuario	Rol	Correo	
1	EDITAR	prueba1	USUARIO	p1@gmail.com	Editar Eliminar
2		juanito	USUARIO	juanito@gmail.com	Editar Eliminar

H. Funcionalidad correo de verificación

Como la mayoría de las aplicaciones web modernas, en la aplicación desarrollado en este trabajo se implementó la funcionalidad de correo de verificación.

Para lograrlo se utilizó Mailtrap.io como plataforma de entrega de correo electrónico pues en sus servicios proporciona una opción de hasta 50 mensajes de correos relacionados con email-testing. Así pues, en primer lugar se creó una cuenta en esta plataforma y en la sección de integrations se seleccionó Laravel.

My Inbox

Total r

[SMTP Settings](#)
[Email Address](#)
[Auto Forward](#)
[Manual Forward](#)

Integrations ?

Laravel 7+

Laravel provides a clean, simple API over the popular SwiftMailer library.

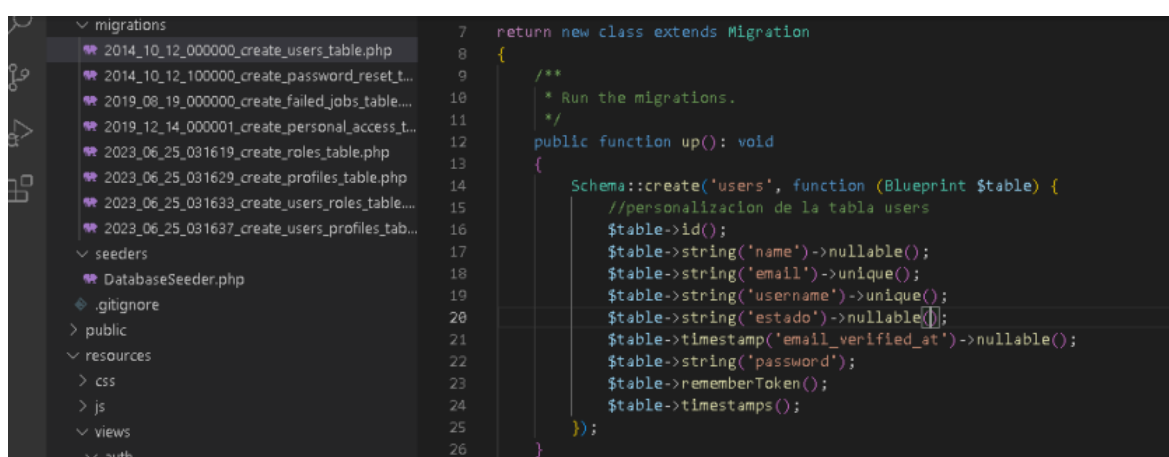
With the default Laravel setup you can configure your mailing configuration by setting values in the .env file in the root directory of your project.

```
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=cb63d42cc814b2
MAIL_PASSWORD=*****fc8f
MAIL_ENCRYPTION=tls
```

Al hacer lo anterior se despliegan las configuraciones del servicio para su implementación en el framework, por lo cual resultó necesario pegar dichas configuraciones en el fichero .env del proyecto.

```
31 MAIL_MAILER=smtp
32 MAIL_HOST=sandbox.smtp.mailtrap.io
33 MAIL_PORT=2525
34 MAIL_USERNAME=cb63d42cc814b2
35 MAIL_PASSWORD=f711196524fc8f
36 MAIL_ENCRYPTION=tls
```

Adicionalmente, es primordial que la tabla asociada contenga la columna `email_updated_at` que viene integrada por defecto en la migración de la tabla `Users` al crear la aplicación. Dado que al inicio del proyecto estas columnas se removieron fue necesario agregarla nuevamente modificando la migración de `Users`.



```
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('users', function (Blueprint $table) {
15             //personalización de la tabla users
16             $table->id();
17             $table->string('name')->nullable();
18             $table->string('email')->unique();
19             $table->string('username')->unique();
20             $table->string('estado')->nullable();
21             $table->timestamp('email_verified_at')->nullable();
22             $table->string('password');
23             $table->rememberToken();
24             $table->timestamps();
25         });
26     }
```

Ahora bien, dado que la aplicación se desarrolló de manera manual en su mayoría el middleware ofrecido por webkits no generó las configuraciones necesarias por lo que se hicieron manualmente. De esta manera en primer lugar se definieron las rutas en `web.php` como se muestra a continuación:



```
41 // Rutas de verificación de correo electrónico
42 Route::get('/email/verify', function () {
43     return view('auth.verify-email');
44 })->middleware(['auth'])->name('verification.notice');
45
46 Route::get('/email/verify/{id}/{hash}', function (\Illuminate\Foundation\Auth\EmailVerificationRequest $request) {
47     $request->fulfill();
48
49     return redirect('/home');
50 })->middleware(['auth', 'signed'])->name('verification.verify');
51
52 Route::post('/email/verification-notification', function (\Illuminate\Http\Request $request) {
53     $request->user()->sendEmailVerificationNotification();
54
55     return back()->with('message', 'Verification link sent!');
56 })->middleware(['auth', 'throttle:6,1'])->name('verification.send');
```

Finalmente se modificó la vista del `home` que se despliega al iniciar sesión para que se muestre una notificación del mensaje enviado al correo y la opción de volver a enviar el correo verificación.


```
resources > views > home > index.blade.php > div.bg-light.p-5.rounded > div.container.mt-5
1 @extends('layouts.app-master')
2
3 @section('content')
4 <div class="bg-light p-5 rounded">
5     @auth
6
7         @if(!auth()->user()->email_verified_at)
8             <div class="container mt-5">
9                 <div class="card">
10                     <div class="card-body">
11                         <h4 class="card-title">Verificación de correo electrónico</h4>
12                         <p class="card-text">Gracias por registrarte. Antes de comenzar, por favor verifica tu dirección de
13                         <p class="card-text">Si no has recibido el correo electrónico de verificación, puedes solicitar otr
14                         <form action="{{ route('verification.send') }}" method="POST">
15                             @csrf
16                             <button type="submit" class="btn btn-primary">Enviar correo de verificación</button>
17                         </form>
18                     </div>
19                 </div>
20             </div>
21         @endif
22
23
```

Finalmente se constató que la funcionalidad funcione creando un usuario nuevo, observando que el correo se envía a Mailtrap y el mensaje de notificación al iniciar sesión aparece

Registrarse

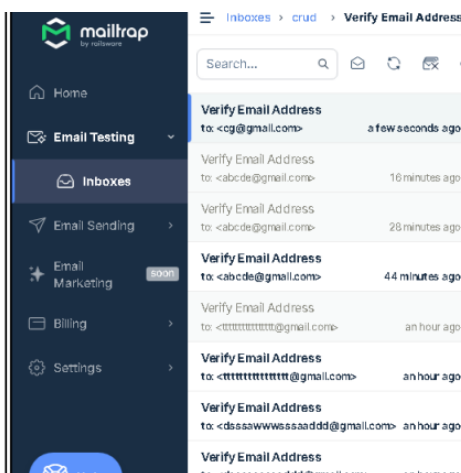
cristo

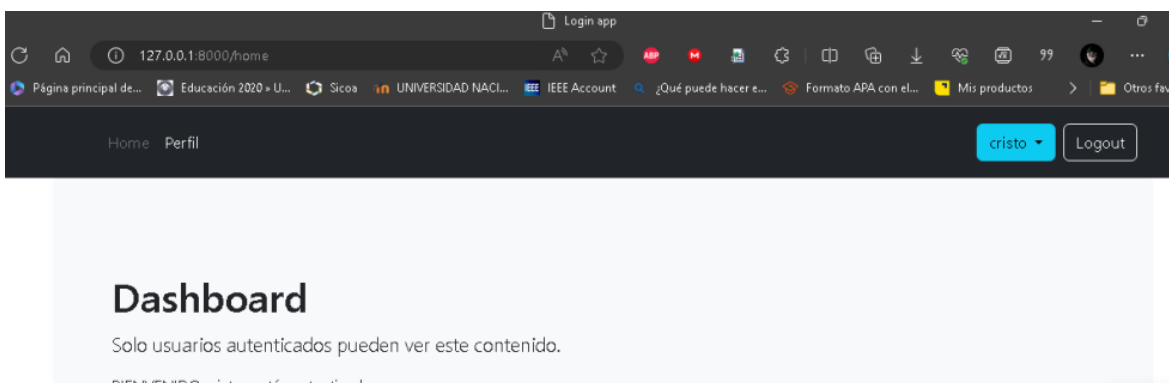
cg@gmail.com

Mr_Robot

[Ya tengo una cuenta](#)
[Inicio](#)

Register





7.4 Resultados

- Laravel es un framework poderoso y versátil que facilita el desarrollo de aplicaciones web de manera rápida y eficiente. Proporciona una amplia gama de herramientas y características, como Eloquent ORM, migraciones y autenticación, que permiten construir aplicaciones robustas y escalables. Su manejo en conjunto con MySQL como base de datos y Bootstrap como gestor para las interfaces de usuario responsivas ayudan a agilizar el proceso de diseño, creación e implementación de sitios web adaptados a diferentes plataformas.
- La aplicación del patrón de diseño Modelo-Vista-Controlador en el proyecto ha permitido crear una estructura organizada, modular y escalable. La separación clara de responsabilidades entre los componentes del MVC (modelo, vista y controlador) mejora la mantenibilidad, facilita la colaboración y proporciona una base sólida para desarrollar aplicaciones web robustas y funcionales.
- Las relaciones de las tablas son un aspecto fundamental en el diseño de la base de datos y tienen un impacto significativo en la funcionalidad y eficiencia de la aplicación. Un diseño bien estructurado y coherente de las relaciones en la base de datos contribuye a un código más limpio y mantenible, mejorando así la experiencia del desarrollador y del usuario final.
- Desarrollar una aplicación web implica una variedad de conceptos y tecnologías interrelacionadas. La planificación, el diseño de la base de datos, la implementación de las vistas y la lógica de negocio son aspectos clave que requieren atención y comprensión para construir una aplicación exitosa y funcional.

7.5 Bibliografía

8. ANEXOS

