

***Facultad de la Energía, las Industrias y los Recursos Naturales No Renovables***

Carrera de Ingeniería en Sistemas

**“Incorporación de la Comunicación Asíncrona en los procesos de la acción tutorial docente en Ambientes Universitarios”**

“Trabajo de Titulación Previo a la obtención del título de Ingeniero en sistemas”

**1. MANUAL DEL PROGRAMADOR**

***Autor:***

* Sergio David Cartuche Morocho

***Director:***

* Ing. Luis Antonio Chamba Eras Mg. Sc

Loja –Ecuador

2019

1. **Introducción**

El propósito de este manual del programador, es dar a conocer al lector los códigos fuentes del programa realizado. Para ello de la forma más concisa se trata de explicar cada uno de los códigos, junto con la programación utilizada en el desarrollo del software, esto con el fin de que el usuario pueda modificar a gusto alguno de los valores y parámetros de las funciones que se encuentran expuestas en la programación del software SGT-CIS (software Web para tutorías académicas dentro de la Carrera de Ingeniería en Sistemas, de la Universidad Nacional de Loja).

El software SGT-CIS se lo desarrolló por medio del framework Laravel 5.7.24 y el lenguaje de programación PHP 7.3.0; para el almacenamiento de la información se utilizó MySQL como base de datos.

1. **Requisitos principales**

El software SGT-CIS se lo desarrolló con la versión 5.7.24 más actual de Laravel en su momento, siendo así se necesita considerar los siguientes requisitos.

* 1. **PHP >= 7.0.0**

Para comprobar que tenemos la versión adecuada de PHP para trabajar, por medio de consola ejecutamos el siguiente comando “*php -v*”, tal y como se indica en la Figura 1; se puede observar que la versión que se tiene instalada de PHP es 7.3.0.



Figura 1. Versión 7.3.0 de PHP

* 1. **Servidor Web**

Laravel necesita de un servidor Web, para ello se utilizó Apache como servidor local y fue necesaria la instalación de XAMPP, debido a que es un programa que nos ofrece una distribución de Apache, PHP y MyQL muy simple de instalar, administrar y utilizar. Podemos descargarlo desde su página principal[[1]](#footnote-1).

Luego de descargarlo e instalarlo es necesario activar el servidor Apache y MySQL, tal y como se indica en la Figura 2.

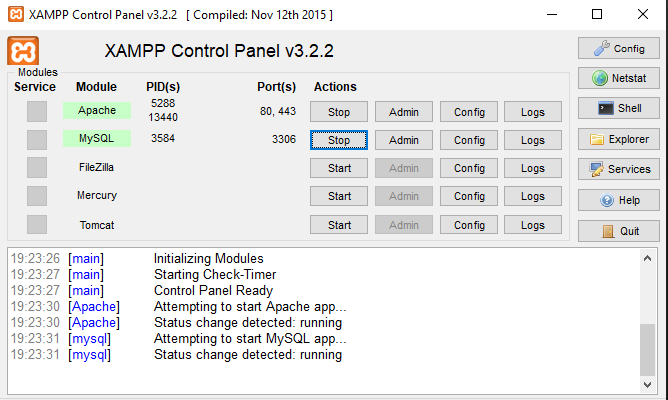


Figura 2. Servidor Web - Apache

1. **Instalación de Laravel**

Luego de haber comprobado que la versión de PHP es adecuada y tener instalado el servidor, se procede a instalar Laravel. La instalación de Laravel se la realizó por consola al ejecutar el siguiente comando “*composer create-project laravel/laravel --prefer-dist sgtcis*”, **sgtcis** representa el nombre del proyecto. Para más detalle de su instalación se recomienda acceder al siguiente enlace[[2]](#footnote-2).

1. **Estructura del software SGT-CIS en Laravel**

En la Figura 3 se muestra la estructura de las carpetas que generó Laravel para el desarrollo del software, cada carpeta y archivo se va detallando conforme se vaya configurando y codificando el software.

1. **Conexión con la base de datos**

El software SGT-CIS utiliza MySQL como base de datos, su configuración se la realizó de la siguiente manera. Dentro del archivo “*database.php*” en el directorio “*config*” se configuró el driver de la conexión, aunque por defecto vendrá con “*mysql*” (ver Figura 4).

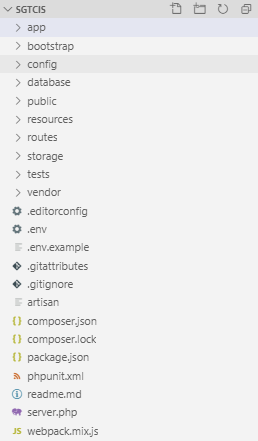


Figura 3. Estructura del software SGT-CIS - Laravel

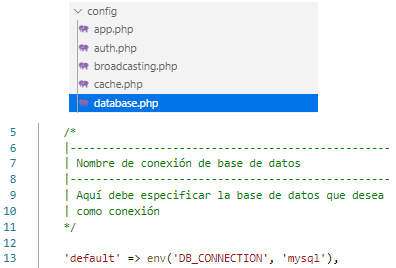


Figura 4. Directorio y configuración de la conexión de la base de datos

Luego de configurar la base de datos con la que vamos a trabajar, se creó la base de datos, para ello mediante consola se ejecutaron los siguientes comandos: “*mysql -uroot*” y “*CREATE DATABASE sgtcis;*” tal y como se indica en la Figura 5; **sgtcis** corresponde al nombre de la base de datos.

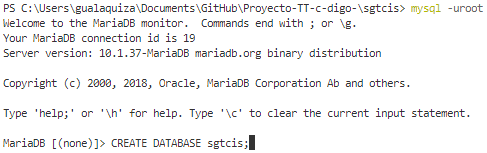


Figura 5. Creación de la base de datos

Comprobamos que en realidad se ha creado la base de datos, ingresando a un navegador y en la barra de navegación escribimos “*localhost/phpmyadmin*”, en la Figura 6 se puede observar que la base de datos **sgtcis** se encuentra creada.

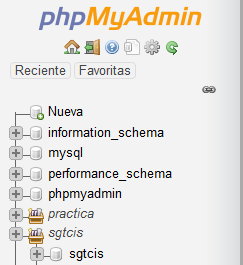


Figura 6. Base de datos sgtcis en phpMyAdmin

Finalmente, para conectar la base de datos creada (sgtcis) se configuró el archivo “*.env*” ubicado en la raíz del proyecto (ver Figura 7).

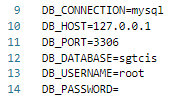


Figura 7. Configuración final de la base de datos

1. **Codificación**

Para acceder a la página de inicio del software SGT-CIS, se lo hace escribiendo en la barra de navegación “*localhost/sgtcis/public*”, al presionar “*Enter*” nos presenta la siguiente pantalla o interfaz (ver Figura 8). A continuación, se presenta y detalla el código utilizado para presentar esta interfaz.



Figura 8. Página principal del software SGT-CIS

* 1. **Página principal**

Para mostrar la página principal del software SGT-CIS, dentro del código parte de una ruta generada, para ello dentro del archivo “*web.php*” en el directorio “*routes*” se codificó esta ruta (ver Figura 9); se puede observar que la ruta declarada es de tipo “get” para la URL del home o inicio “/”; a su vez también se puede observar que dentro de ella existe una función la cual expresa lo siguiente:

* Si existe un usuario autenticado “Auth::check()”, guardarlo en la variable “$user”.
* Si el usuario autenticado con el campo “is\_admin” es verdadero retorna la ruta “auth\_admin”.
* Si el usuario autenticado con el campo “is\_docente” es verdadero retorna la ruta “auth\_docente”.
* Si el usuario autenticado con el campo “is\_estudiante” es verdadero retorna la ruta “auth\_student”.

A esta condición no entrará para el caso, debido a que aún no se tiene ningún usuario registrado en la base de datos y no tiene que comparar.

* Caso contrario retorne la vista “welcome”.

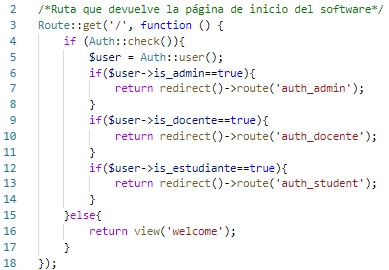


Figura 9. Código - Ruta para la página de inicio del software

La función dentro de la ruta, nos retorna la vista “welcome”; esta vista se encuentra dentro del directorio “*resources/views/welcome.blade.php*” (Laravel detecta automáticamente la extensión “.blade.php” al retornar vistas). En la Figura 10 se muestra el código que presenta la interfaz de la página de inicio del software; se puede observar lo siguiente:

* **@extends('layout\_pagina\_inicio'),** mediante esta línea se indica que la vista “welcome”, herada todo lo que se encuentra dentro de la vista “layout\_pagina\_inicio”; en la Figura 11 se muestra el código de esta vista que se encuentra en el directorio “*resources/views/layout\_pagina\_inicio.blade.php*”; aquí se puede observar las directivas **@yield('header')** y **@yield('footer'),** se las utilizó para mostrar el contenido en una sección determinada.
* **@section('header'),** dentro de esta sección se encuentra codificado el contenido correspondiente al menú horizontal, imagen y formulario de registro que se muestra en la página de inicio. **'header'** corresponde al nombre que se encuentra dentro de la directiva @yield en el archivo “*layout\_pagina\_inicio.blade.php*”**.**
* **@section('footer'),** dentro de esta sección se encuentra codificado el contenido correspondiente a toda la información que aparece al final de la página de inicio. **'footer'** corresponde al nombre que se encuentra dentro de la directiva @yield en el archivo “*layout\_pagina\_inicio.blade.php*”**.**
* **@section(scripts),** dentro de esta sección se hace el llamado a librerías o archivos “.js” que son necesarios dentro de la vista.

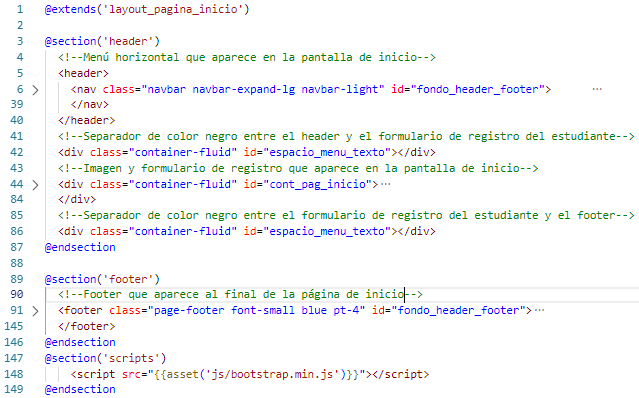


Figura 10. Código - Vista para la página de inicio del software



Figura 11. Código - Vista archivo layout\_pagina\_inicio.blade.php

El software cuenta con un total de 21 requerimientos funcionales, los cuales se detallan a continuación.

* 1. **Registro de usuario estudiante**

Para el registro del usuario estudiante se tiene dos opciones: una opción manual y otra por medio de una cuenta de google. Las dos opciones de registro se encuentran en la vista principal del software, cuyo código encargado de presentar la interfaz para el registro (ver Figura 8) se encuentra dentro del directorio “*resources/views/welcome.blade.php*” y que ya fue descrito anteriormente (ver apartado 6.1. Página principal); si analizamos nuevamente el código en este archivo podemos observar el formulario de registro del estudiante (ver Figura 12).

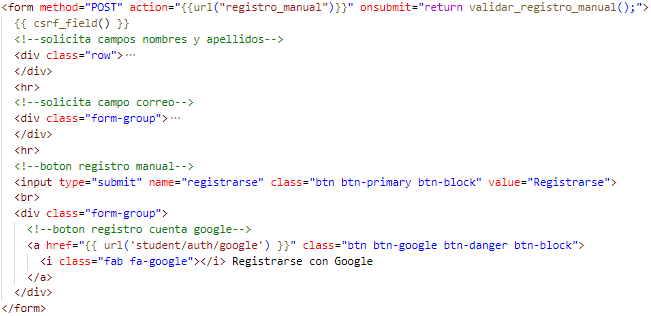


Figura 12. Código – Vista formulario de registro del estudiante

En la Figura 12, se puede observar que existen dos botones uno con el texto Registrarse (manual) y el otro Registrarse con Google, en la Figura 8 se los puede observar cómo es su aspecto en pantalla. A continuación, se detalla el código correspondiente a la función que cumple cada uno de estos botones una vez que se le haga clic.

* + 1. **Registro manual**

Cuando el usuario haga clic en el botón Registrarse, los campos solicitados (nombres, apellidos y correo) van a pasar por el método POST a la URL “*registro\_manual*” (ver Figura 12); esta URL se la crea en el archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 13 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “post” para la URL (“registro/manual”), a su vez también se puede observar que la ruta apunta al controlador “Auth\LoginController” y le pasa como método o función “registro\_manual” por medio del @; **name** corresponde al nombre de la ruta, para el caso el nombre “registro\_manual” fue utilizado en la vista dentro de la etiqueta “form” y el atributo “action” (ver Figura 12).



Figura 13. Código – Ruta para registro de estudiante manual

Antes de presentar el método o función “*registro\_manual*”, es necesario crear una tabla en la base de datos para poder almacenar la información del estudiante; para ello mediante consola se debería ejecutar el siguiente comando “*php artisan make:model User -m*” (ver Figura 14); **User** corresponde al nombre del modelo y mediante **–m** se crea la migración (Laravel conoce como migración a la tabla). Se le llamó User al nombre del modelo, debido a que la información que se va a guardar corresponde al estudiante; y este es un usuario dentro del software. Cuando se instaló Laravel, tanto el modelo como la migración ya vinieron por defecto, entonces no fue necesario ejecutar el comando. Sin embargo, se lo describió en este punto puesto que más adelante cuando se requiera guardar información en la base de datos que no sea del usuario, es necesario seguir el mismo procedimiento.



Figura 14. Creación del Modelo User y la tabla users

Al ejecutar el comando en la Figura 14, Laravel generó dos archivos el del modelo y el de la tabla o migración respectivamente. Por su lado el archivo generado para el modelo se encuentra dentro del directorio “*app/User.php*” y de la migración o de la tabla en el directorio “*database/migrations/2014\_10\_12\_000000\_create\_users\_table*”. En el caso de la migración o tabla Laravel la guarda de la siguiente forma “*2014\_10\_12\_000000\_create\_users\_table*”, sin embargo, para su uso no hay que escribir todo como tal, sino solamente **users**, que es el nombre del modelo, pero en plural. Cada vez que se genere un modelo con su migración o tabla se van a encontrar en los mismos directorios solo que con diferente nombre de archivo.

Hasta el momento solamente tenemos creada la tabla **users**, para asignarle campos a ella accedemos al archivo, que se encuentra en el directorio mencionado anteriormente. En la Figura 15 se muestra el código correspondiente a la asignación de los campos a la tabla **users;** Luego de ello mediante consola se ejecutó el siguiente comando “*php artisan migrate*”; si este comando no se lo ejecuta en la base de datos no habrá pasado nada y la tabla **users** no existirá dentro de ella; entonces este comando sirve para crear la tabla en la base de datos. Algo a tener en **cuenta** es que al ejecutar este comando se van a borrar todos los registros que se encuentren en ese momento almacenados en la base de datos.

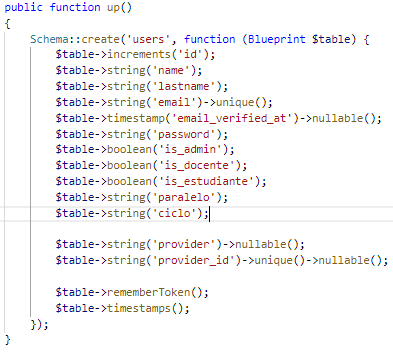


Figura 15. Código – Generar campos para la tabla users

Para comprobar que efectivamente la tabla **users** se encuentra en la base de datos accedemos a un navegador y en la barra de navegación escribimos “*localhost/phpmyadmin*”, buscamos la base de datos **sgtcis,** clic en la tabla users y se listarán los campos creados; la Figura 16 muestra ello.

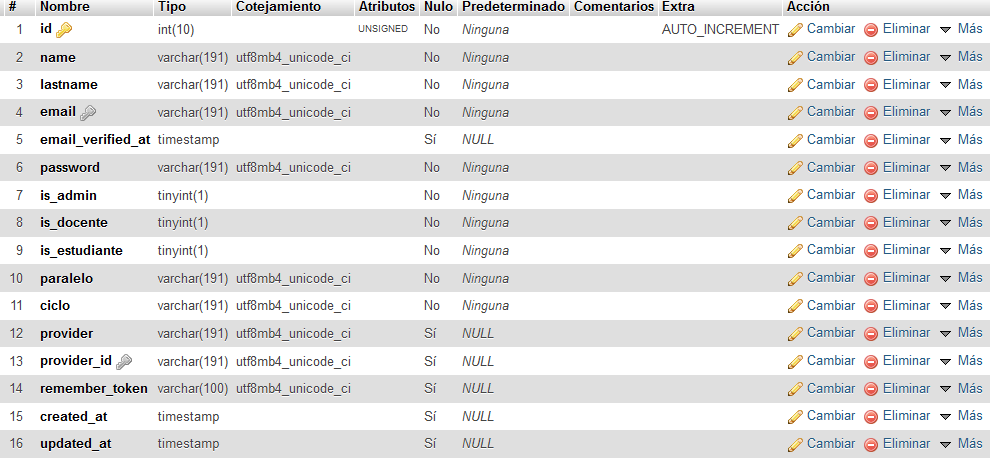


Figura 16. Campos de la tabla users en phpMyAdmin

Ahora dentro del archivo “User.php” (modelo) se asignó los campos para poder asignar datos en masa, pues el software está desarrollado para soportar el registro de muchos usuarios (estudiantes). En la Figura 17 se presenta todo el código utilizado dentro del modelo User, sin embargo para el caso solamente se explicará lo que por ahora se requiere; para conseguir el poder asignar datos en masa se utilizó la propiedad “*$fillable*” en ella se especificaron que campos de la tabla **users,**  se guardarán en la base de datos; mediante ello también permite poder actualizar o eliminar de la base de datos los campos especificados.



Figura 17. Código – modelo User

|  |  |
| --- | --- |
| C:\Users\gualaquiza\Dropbox\tesis\manuales\logo-gestoria1-400x400.png | Desde aquí en adelante, para los modelos y migraciones (tablas) se omitirá el proceso realizado, pues el procedimiento será el mismo que fue realizado para el modelo User. |

Una vez que se tiene la tabla en donde se va a guardar la información del estudiante cuando se registre (users), se continúa con el llamado al método o función “*registro\_manual*” desde la ruta “*registro/manual*” (ver Figura 13).

El controlador se encuentra dentro del directorio “*app/Http/Controllers/Auth/LoginController.php*”, si analizamos este archivo, dentro de él se encuentra el método o función “*registro\_manual*”, tal y como se muestra en la Figura 18.



Figura 18. Código – método registro manual del estudiante

A continuación, se describe cada parte del código en la Figura 18:

* **Request** $request: Request es una clase que Laravel utiliza para poder obtener datos de un formulario y $request es la variable utilizada para obtener estos datos.
* **$email, $name, $lastname:** variables en donde se guarda la información que se obtiene del formulario; correo, nombres y apellidos respectivamente.
* **$valida\_email:** variable que guarda un valor (true o false); para ello primero verifica en la base de datos, si en la tabla **users**, en el campo **email** existe algún registro igual al **$email** ingresado.
* Si **$valida\_email** es verdadero, significa que el email o correo ingresado ya existe registrado en la base de datos; por lo que por medio de **flash()** se envía un mensaje de error, que será presentado en pantalla en el mismo formulario de registro o en la vista “*welcome*”.
* En el **else**, entra cuando **$variable\_email** es falso, y pasa a realizar lo siguiente:
* **User::create([ ]);** registra el estudiante en la tabla **users,** en los campos y con los datos que se encuentran dentro del arreglo.
* **$authUser**: guarda el usuario que se ha registrado.
* **Auth**::**login($authUser,true):** autentifica al usuario registrado, es decir hace como que el usuario registrado ha iniciado sesión y por lo tanto ha accedido al software.
* **return redirect()**-**>route(‘**vista\_completar\_registro**’):** si bien es cierto el usuario ya se encuentra autenticado, sin embargo, no puede tener acceso a las funciones que puede realizar dentro de su cuenta, por lo que primero es redirigido a la ruta vista\_completar\_registro. Más adelante cuando se comente sobre el requisito funcional completar registro, se explica sobre esta ruta.
  + 1. **Registro cuenta de google**

Para que un usuario pueda registrarse con google, Laravel a partir de la versión 5 nos proporciona socialite[[3]](#footnote-3), un excelente paquete que nos permite autenticar usuarios con una gran variedad de servicios como Facebook, Google, Twitter, entre otros.

* + - 1. **Instalando Socialite**

Para empezar a usar **“*laravel/socialite*”** primero se debe requerir el paquete, para ello se ejecutó por consola el siguiente comando “*composer require laravel/socialite*”, como se muestra en la Figura 19.



Figura 19. Instalación del paquete Socialite

* + - 1. **Registro con google en Laravel**

Primeramente, se creó un nuevo proyecto en ***Developers Console[[4]](#footnote-4),*** una vez allí nos dirigimos donde dice Credenciales, luego clic en Crear credenciales y seleccionamos ID de cliente de OAuth. Luego de ello se le asignó un nombre al proyecto y luego se autorizó las URL’s de los servidores local y de producción (Apache y MsAzure), como se muestra en la Figura 20; y para finalizar damos clic en Guardar. Esto nos va a generar un ID de cliente y un secreto de cliente, que luego se agregaron en el archivo “.env” de nuestro proyecto, como se muestra en la Figura 21. Finalmente se agregó el servicio de google en el archivo “*config/services.php*”, como se muestra en la Figura 22.

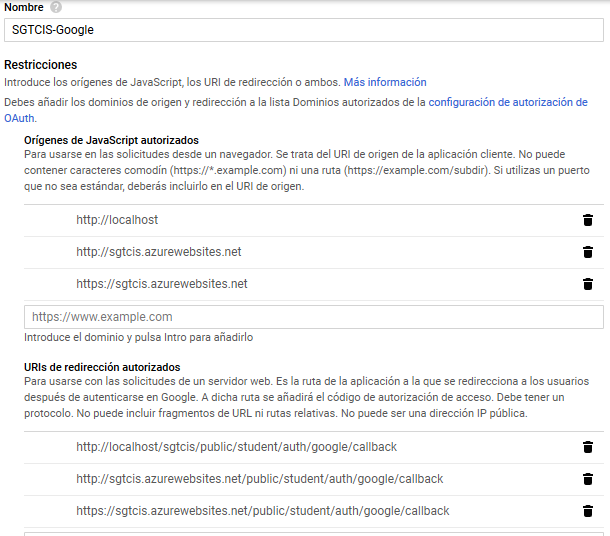


Figura 20. Configuración de Credenciales para registro con google



Figura 21. Configuración del ID de cliente y secreto de cliente de google

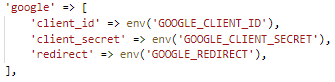


Figura 22. Configuración del servicio google

Luego de tener la configuración lista, cuando el usuario haga clic en el botón Registrarse con Google, el usuario será redirigido a la URL “*student/auth/google*” (ver Figura 12); esta URL se la crea en al archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 23 se muestra la ruta codificada; se puede observar que se han declarado dos rutas de tipo “get” para la URL (“student/auth/{provider}”), a su vez también se puede observar que ambas rutas apuntan al controlador “Auth\LoginController”, sin embargo en la primer ruta se pasa como método o función “redirectToProvider” y a la segunda “handleProviderCallback”. Dentro de la URL {**provider**} corresponde al servicio con el cual nos vamos a autenticar, para el caso sería “google” pues en el formulario así lo especificamos (ver Figura 12), al encontrarse en llaves se lo puede recibir como parámetro en el método o función.



Figura 23. Código – Ruta para registro de estudiante con google

El controlador se encuentra dentro del directorio “*app/Http/Controllers/Auth/LoginController.php*”, y a su vez dentro de él se encuentra el método o función “redirectToProvider” y “handleProviderCallback” (ver Figura 24).

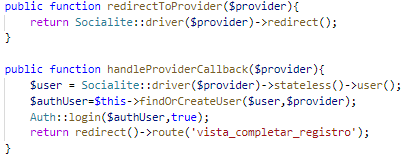


Figura 24. Código – método registro estudiante mediante google

A continuación, se describe cada parte del código en la Figura 24.

Función **redirectToProvider:**

* **$provider:** guarda el valor que se envía como parametro desde la URL, para el caso seria “google”.
* **Socialite::driver($provider)->redirect():** forma en que el paquete de Laravel “socialite” dirige a la pagina de google y así le da permisos al software para que pueda acceder a la información de la cuenta que seleccione. En otras palabaras mediante ello, el software le presenta al usuario las cuentas de google que se encuentren activas en su dispositivo.

Luego de que el usuario seleccione una cuenta de google con la que hará su registro, si observamos la Figura 20, vemos que nos direcciona a la URL “student/auth/google/callback”, esta URL corresponde a la segunda ruta que se ha creado (ver Figura 23); y como ya se mencionó anteriormente esta ruta corresponde al método o función que se va a describir a continuación.

Función **handleProviderCallback:**

* **$provider:** guarda el valor que se envía como parámetro desde la URL para el caso seria “google”.
* **$user:** forma en que el paquete de Laravel “socialite” obtiene la información de la cuenta de google del usuario y se guarda en la variable $user.
* **$authUser:** variable que guarda la información del usuario registrado en la base de datos, para ello hace el llamado al método o función **findOrCreateUser()** y a ella se le envía como parámetros $user y $provider (ver Figura 25). A continuación, se describe cada parte del código en la Figura 25.
* **$authUser:** variable que guarda información del usuario, en el caso de que ya exista en la base de datos; para ello verifica en la tabla users, si en el campo provider\_id hay un registro igual al provider\_id ($user->id) del usuario que está por registrarse.

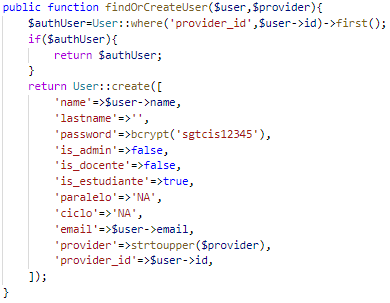


Figura 25. Código – método registro estudiante mediante google

* Si la variable $authUser no es nula, significa que el usuario ya se encuentra registrado; por lo que no lo registra nuevamente en la base de datos sino que simplemente devuelve su información. A esta condición va a entrar, cuando el usuario ya se encuentre registrado en la base de datos.
* En el caso de que la variable $authUser sea nula, primero registra al usuario en la base de datos y luego retorna la información.
* **Auth**::**login($authUser,true):** autentifica al usuario registrado, es decir hace como que el usuario registrado ha iniciado sesión y por lo tanto ha accedido al software.
* **return redirect()**-**>action(‘**AuthStudentController@vista\_completar\_registro**’):** si bien es cierto el usuario ya se encuentra autenticado, sin embargo, no puede tener acceso a las funciones que puede realizar dentro de su cuenta, por lo que primero es redirigido a la ruta vista\_completar\_registro. Más adelante cuando se comente sobre el requisito funcional completar registro, se explica sobre esta ruta.
  1. **Acceso al software**

El acceso al software es un requisito que el administrador, estudiante y docente lo pueden hacer; a continuación, se presenta el código que permite el acceso al software para el administrador, para el estudiante y docente no se va a describir, debido a que es el mismo procedimiento que utiliza el administrador; sin embargo, en forma gráfica se va a presentar los métodos o funciones utilizados.

* + 1. **Acceso al software del administrador**

El administrador podrá acceder al software con las siguientes credenciales, que fueron creadas por el desarrollador.

* **Correo:** [adminsgtcis@unl.edu.ec](mailto:adminsgtcis@unl.edu.ec)
* **Contraseña:** sgtcisadmin

En el software no existe un botón que permite el acceso al formulario de inicio de sesión del administrador, por lo que, en la barra de navegación del navegador, a la URL principal del software deberá agregar lo siguiente “/administrator”, quedando de la siguiente forma (ver Figura 26).



Figura 26. URL del administrador

Esta URL, se encuentra creada en el archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 27 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “get”, a su vez también se puede observar que la ruta apunta al controlador “Auth\LoginController " y pasa como método o función “show\_login\_form”.



Figura 27. Código – Ruta para formulario inicio de sesión del administrador

El controlador se encuentra dentro del directorio “*app/Http/Controllers/Auth/LoginController.php*”, y si analizamos este archivo, dentro de él se encuentra el método o función “show\_login\_form” (ver Figura 28).

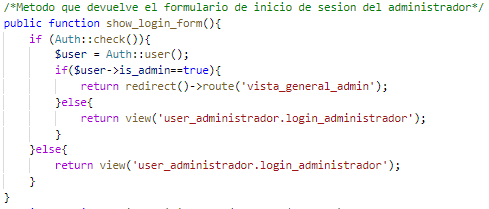


Figura 28. Código – método formulario inicio sesión del administrador

A continuación se describe el código en la Figura 28.

* **if(Auth::check()):** comprueba si existe algún usuario autenticado. Solo entra cuando existe un usuario autenticado.
* **$user:** guarda toda la información del usuario autenticado; la información se obtiene de la tabla users.
* **if($user->is\_admin==true):** verifica que el usuario autenticado sea administrador. Si es verdadero, significa que anteriormente el usuario ya ha accedido al software y será redirigido a la ruta “vista\_general\_admin”. Más adelante cuando se comente sobre el requisito funcional vista general de la cuenta, se explica sobre esta ruta.
* En el **else** entra cuando el usuario autenticado, no es administrador por lo que puede ser estudiante o docente, si es así será redirigido a la vista “login\_administrador” que se encuentra en la carpeta “user\_administrador”; esta vista corresponde al formulario de inicio de sesión del administrador.
* En el **else** entra cuando no existe ningún usuario autenticado, por lo que es redirigido a la vista “login\_administrador” que se encuentra en la carpeta “user\_administrador”; esta vista corresponde al formulario de inicio de sesión del administrador.

En la Figura 29 se muestra el código correspondiente a la vista “login\_administrador”; esta vista se encuentra dentro del directorio “views/user\_administrador/login\_administrador.blade.php”.

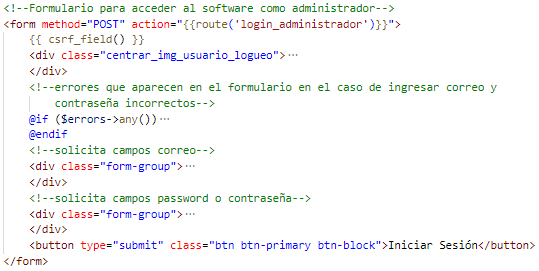


Figura 29. Código – Vista formulario inicio de sesión del administrador

En la Figura 29, se puede observar que existe un botón con el texto Iniciar Sesión. Cuando el usuario haga clic sobre este botón, los campos solicitados (correo y contraseña) van a pasar por el método POST a la ruta “login\_administrador” (ver Figura 29); esta ruta se la crea en el archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 30 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “post” para la URL “login\_administrador”, a su vez también se puede observar que la ruta apunta al controlador “Auth\LoginController” y le pasa como método o función “login\_administrador”; **name** corresponde al nombre de la ruta, para el caso el nombre “registro\_manual” fue utilizado en la vista dentro de la etiqueta “form” y el atributo “action” (ver Figura 29).



Figura 30. Código – Ruta para acceder al software como administrador

El controlador se encuentra dentro del directorio “*app/Http/Controllers/Auth/LoginController.php*”, si analizamos este archivo, dentro de él se encuentra el método o función “login\_administrador”, tal y como se muestra en la Figura 31.

A continuación, se describe el código mostrado en la Figura 31.

* **Request** $request: Request es una clase que Laravel utiliza para poder obtener datos de un formulario y $request es la variable utilizada para obtener estos datos.



Figura 31. Código – método acceso al software del administrador

* **if($request->isMethod(‘post’)):** comprueba si el método enviado a esta función es “post”; por lo general siempre lo va hacer, por lo que cada vez que se haga clic sobre el botón va a entrar a esta sentencia.
* **$credenciales:** variable utilizada para guardar el correo y contraseña, siempre y cuando el usuario no haya dejado ningún campo vacío; caso contrario en el formulario le aparecerá un mensaje indicándole que tal campo es requerido.
* **if(Auth::attempt($credenciales)):** verifica que la información guardada en $credenciales (correo y contraseña) estén registrados en la base de datos dentro de la tabla users. A esta sentencia entra cuando existe un usuario registrado independientemente si es administrador, estudiante o docente.
* **$email\_form:** guarda el correo que el usuario ha ingresado en el formulario.
* **$users:** obtiene el primer registro de la tabla users, cuando el correo guardado en la variable $email\_form sea igual a algún registro guardado en el campo email.
* **if($users->is\_admin==true):** comprueba si el usuario es administrador. Si es verdadero en la siguiente sentencia if, comprueba si el usuario está autenticado; si es verdadero lo direcciona a la ruta “vista\_general\_admin”. Más adelante cuando se comente sobre el requisito funcional vista general de la cuenta, se explica sobre esta ruta.
* En el **else** entra cuando el usuario no es administrador y lo direcciona a la ruta “show\_login\_form”, que corresponde al formulario inicio de sesión del administrador y se envían los errores que son capturados en la vista (ver Figura 29).
* En el caso de que las credenciales sean correctas, sin embargo, no se encuentre registradas en la base de datos, el usuario será redirigido al mismo formulario y se envían los errores que serán capturados en la vista ver (Figura 29).
  + 1. **Acceso al software del estudiante**

El estudiante podrá acceder al software con las siguientes credenciales:

* **Correo:** generado por el usuario al momento de registrarse.
* **Contraseña:** generada por el usuario al momento de completar el registro.

El código generado para el acceso al software del estudiante es similar al utilizado para el administrador, por lo que solamente se presentará en figuras el código, mas no su explicación.



Figura 32. Código – Ruta para formulario inicio de sesión del estudiante

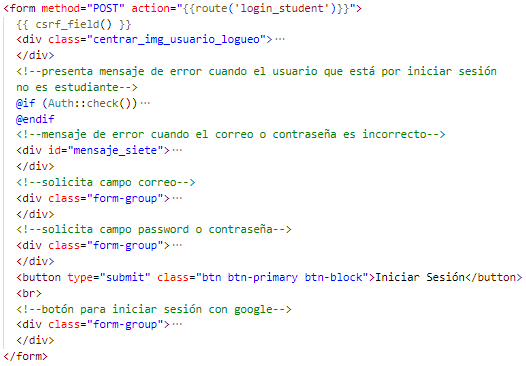


Figura 33. Código – Vista formulario inicio de sesión del estudiante



Figura 34. Código – método formulario inicio sesión del estudiante



Figura 35. Código – Ruta para acceder al software como estudiante

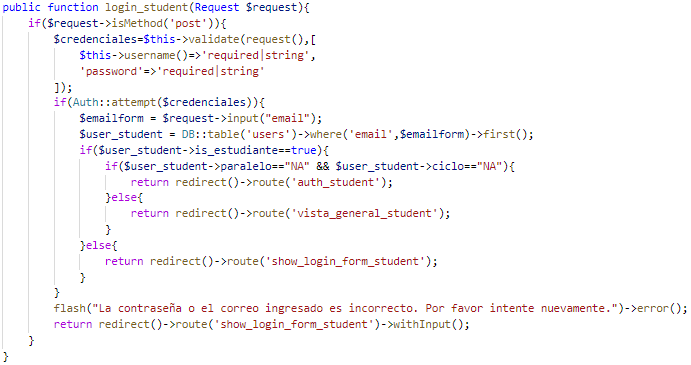


Figura 36. Código – método acceso al software como estudiante

* + 1. **Acceso al software del docente**

El docente podrá acceder al software con las siguientes credenciales:

* **Correo:** generado en un inicio por el administrador (correo institucional del docente).
* **Contraseña:** generado en un inicio por el administrador (primer nombre, primer apellido), todo en minúscula, sin tildes y seguido.

El código generado para el acceso al software del docente es similar al utilizado para el administrador, por lo que solamente se presentará en figuras el código, mas no su explicación.



Figura 37. Código – Ruta para formulario inicio de sesión del docente

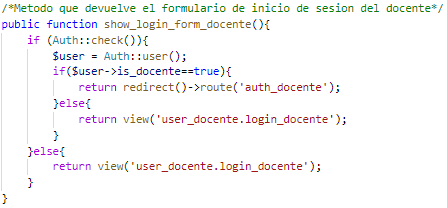


Figura 38. Código – método formulario inicio sesión del docente

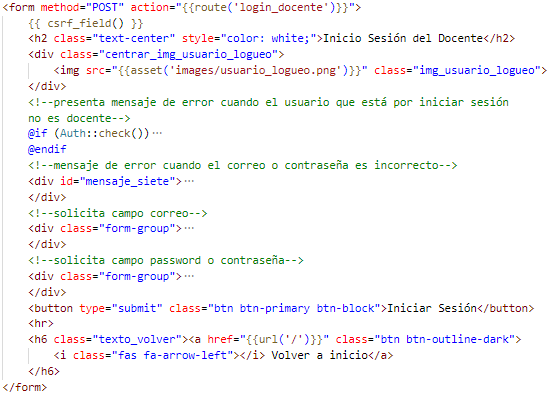


Figura 39. Código – Vista formulario inicio de sesión del docente



Figura 40. Código – Ruta para acceder al software como docente

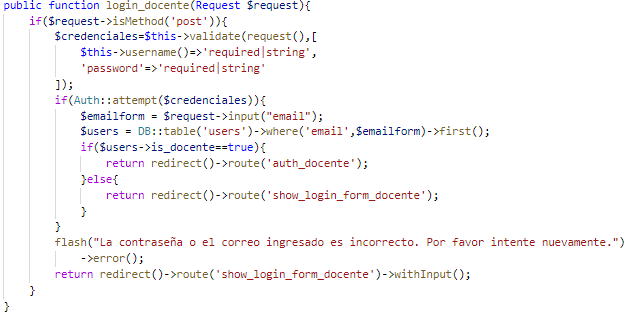


Figura 41. Código – método acceso al software como docente

* 1. **Completar registro**

El requisito completar registro es una funcionalidad que solamente es realizada por el usuario estudiante. En el software se va a presentar un formulario luego de que el usuario se ha registrado (manual o google), en la Figura 18 y en la Figura 24, se puede observar que luego de registrar (nombres, apellidos y correo) al estudiante, se direcciona a la ruta “vista\_completar\_registro”; esta ruta se la crea en al archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 42, se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “get” para la URL (“vista\_completar\_registro”), a su vez también se puede observar que la ruta apunta al controlador “AuthStudenController” y le pasa como método o función “vista\_completar\_registro”.



Figura 42. Código – Ruta para formulario completar registro estudiante

El controlador se encuentra dentro del directorio “app/Http/Controllers/AuthStudentController.php”, si analizamos este archivo, dentro de él se encuentra el método o función “vista\_completar\_registro”, tal y como se muestra en la Figura 43.

A continuación, se describe el código mostrado en la Figura 43.

* **if(Auth::check()):** comprueba si existe algún usuario autenticado.



Figura 43. Código – método formulario completar registro estudiante

* **$user\_student:** guarda la información del usuario autenticado; la información la obtiene de la tabla users en la base de datos.
* **if($user\_student->is\_estudiante==true):** comprueba si el usuario es estudiante. Si es verdadero pasa a cumplir el resto de instrucciones.
* **if($user\_student->paralelo==’NA’ && $user\_student->ciclo==’NA’):** comprueba si el estudiante no tiene asignado (NA) paralelo y de igual manera el ciclo. Si para ambas condiciones la respuesta es verdadera, significa que el estudiante aún no ha completado el registro y pasa a cumplir el esto de instrucciones.
* **$materias:** guarda información de todas las materias que se encuentren registradas; la información la obtiene de la tabla materias en la base de datos. Las materias se las obtiene en base al registro funcional registrar materia, que se comenta más adelante.
* **$verifica\_arrastre:** guarda un valor (true o false); para ello comprueba en la tabla arrastres de la base de datos si existe algún registro en el campo “user\_estudiante\_id” igual al id ($user\_estudiante->id) del usuario autenticado.
* **$docentes:** guarda información de todos los docentes que se encuentren registrados; la información la obtiene de la tabla users de la base de datos. Los docentes se los obtiene en base al registro funcional registrar docente, que se comenta más adelante.
* **if($verifica\_arrastre==true):** comprueba si la variable $verifica\_arrastre es verdadera; si es así, significa que el usuario ya ha asignado materias de arrastre y pasa a cumplir el resto de instrucciones. A esta sentencia no es muy probable que el usuario entre si completa el registro de manera seria.
* **$arrastre:** guarda información de las materias arrastradas; para ello obtiene el primer registro de la tabla arrastres en la base de datos, cuando el campo “user\_estudiante\_id” es igual al id ($user\_student->id) del usuario autenticado.
* **$arreglo\_materia:** guarda en un arreglo los id’s de las materias que el usuario ha añadido[[5]](#footnote-5).
* **$arreglo\_paralelo:** guarda en un arreglo los paralelos correspondientes a cada materia[[6]](#footnote-6).

Lo direcciona a la vista “user\_student.completar\_registro”; todo lo que se encuentra dentro del atributo compact(), son variables obtenidas en el mismo método y luego son presentados en la vista.

* Al **else** siempre va a entrar una vez que el usuario se ha registrado (manual o google); y se lo direcciona a la vista “user\_student.completar\_registro”; todo lo que se encuentra dentro del atributo compact(), son variables obtenidas en el mismo método y luego son presentados en la vista.
* En el **else** entra cuando el usuario ya ha completado el registro; y lo direcciona a la vista “user\_student.auth\_student”; lo que significa que ha accedido al software y tiene acceso a todas sus funcionalidades.
* En el **else** entra cuando el usuario no es estudiante y lo direcciona a la ruta “show\_login\_form\_student”, que corresponde al formulario de inicio de sesión del estudiante (ver Figura 34).

Dentro de la vista “user\_student.completar\_registro”, se encuentran dos opciones; una para completar registro sin arrastre de materias y otra para completar registro con arrastre de materias, tal y como se muestra en la Figura 44. A continuación, se muestra el código para cada opción.

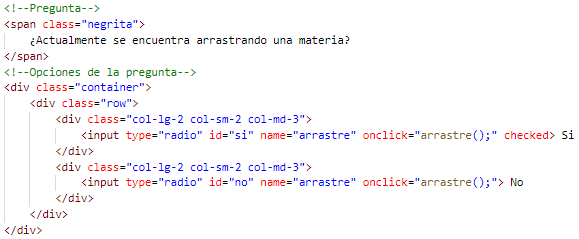


Figura 44. Código – vista completar registro (opciones)

* + 1. **Completar registro con arrastre de materias**

En la Figura 44 se puede observar que primero se realiza una pregunta ¿Actualmente se encuentra arrastrando una materia?; y dos opciones de respuesta Si o No. Si el usuario contesta la pregunta afirmativamente (Si), en el mismo archivo existe el formulario cuando el usuario arrastra materia. El proceso para completar registro con arrastre de materias, se basa en la realización de los siguientes pasos de forma ordenada.

* Buscar materia/s a añadir.
* Añadir materia/s.
* Finalizar registro.

A continuación, se describe cada paso.

* + - 1. **Buscar materia/s a añadir**

El objetivo de buscar materia en el formulario de completar registro con arrastre de materias, es para facilitarle al usuario la manera de obtener la materia que va a añadir. En la Figura 45 se muestra el código correspondiente al formulario de búsqueda, dentro de la vista “user\_student.completar\_registro”; se puede observar que existe un botón con el texto Buscar. Cuando el usuario haga clic sobre este botón, el campo solicitado (nombre de materia) va a pasar por el método GET a la URL “buscar\_materia\_arrastre”; esta ruta se la crea en el archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 46 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “get” a la URL “buscar\_materia\_arrastre”, a su vez también se puede observar que la ruta apunta al controlador “AuthStudentController” y le pasa como método o función “buscar\_materia\_arrastre”.

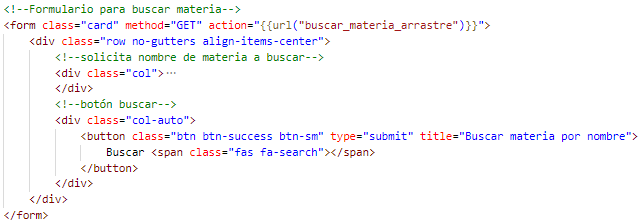


Figura 45. Código – vista completar registro (formulario buscar materia)



Figura 46. Código - ruta para buscar materia (arrastre)

El controlador se encuentra dentro del directorio “app/Http/Controllers/AuthStudenController.php” y si analizamos este archivo, dentro de él se encuentra el método o función “buscar\_materia\_arrastre” tal y como se muestra en la Figura 47.

A continuación, se describe el código mostrado en la Figura 47.

* **if(Auth::check()):** comprueba si existe algún usuario autenticado. Solo entra cuando existe un usuario autenticado.
* **$user\_student:** guarda toda la información del usuario autenticado; la información se obtiene de la tabla users.
* **if($user\_student->is\_estudiante==true):** comprueba si el usuario es estudiante. Si es verdadero pasa a cumplir el resto de instrucciones.
* **$name:** guarda el nombre de la materia que el usuario ingresó en el formulario.
* **if($user\_student->paralelo==’NA’ && $user\_student->ciclo==’NA’):** comprueba si el estudiante no tiene asignado (NA) paralelo y de igual manera el ciclo. Si para ambas condiciones la respuesta es verdadera, significa que el estudiante aún no ha completado el registro y pasa a cumplir el esto de instrucciones.
* **$materias:** guarda información de las materias en base al nombre ($name) que se encuentren registradas; la información la obtiene de la tabla materias en la base de datos. Las materias se las obtiene en base al registro funcional registrar materia, que se comenta más adelante.



Figura 47. Código – método buscar materia (arrastre)

* **$docentes:** guarda información de todos los docentes que se encuentren registrados; la información la obtiene de la tabla users de la base de datos. Los docentes se los obtiene en base al registro funcional registrar docente, que se comenta más adelante.
* **$verifica\_arrastre:** guarda un valor (true o false); para ello comprueba en la tabla arrastres de la base de datos si existe algún registro en el campo “user\_estudiante\_id” igual al id ($user\_estudiante->id) del usuario autenticado.
* **$num\_result:** guarda el número de resultados obtenidos de la información guardada en la variable $materias.
* **if($num\_result>0):** comprueba que el número de resultados sea mayor a cero, si es así devuelve un mensaje por medio de flash(), que luego es presentado en la vista.
* **if($verifica\_arrastre==true):** comprueba si la variable $verifica\_arrastre es verdadera; si es así, significa que el usuario ya ha asignado materias de arrastre y pasa a cumplir el resto de instrucciones.
* **$arrastre:** guarda información de las materias arrastradas; para ello obtiene el primer registro de la tabla arrastres en la base de datos, cuando el campo “user\_estudiante\_id” es igual al id ($user\_student->id) del usuario autenticado.
* **$arreglo\_materia:** guarda en un arreglo los id’s de las materias que el usuario ha añadido[[7]](#footnote-7).
* **$arreglo\_paralelo:** guarda en un arreglo los paralelos correspondientes a cada materia[[8]](#footnote-8).

Lo direcciona a la vista “user\_student.completar\_registro”; todo lo que se encuentra dentro del atributo compact(), son variables obtenidas en el mismo método y luego son presentados en la vista.

* Al **else** va a entrar cuando la variable $verifica\_arrastre es falsa, es decir el usuario no asignado ninguna materia; y se lo direcciona a la vista “user\_student.completar\_registro”; todo lo que se encuentra dentro del atributo compact(), son variables obtenidas en el mismo método y luego son presentados en la vista.
* En el **else** entra cuando el usuario ya ha completado el registro; y lo direcciona a la vista “user\_student.auth\_student”; lo que significa que ha accedido al software y tiene acceso a todas sus funcionalidades.
* En el **else** entra cuando el usuario no es estudiante y lo direcciona a la ruta “show\_login\_form\_student”, que corresponde al formulario de inicio de sesión del estudiante (ver Figura 34).
  + - 1. **Añadir materia/s**

Luego de que el usuario ha buscado y ha obtenido la materia a asignar, ahora deberá añadirla. En la Figura 48 se muestra el código correspondiente al formulario para asignar o añadir materia, dentro de la vista “user\_student.completar\_registro”; se puede observar que existe un botón (en pantalla se muestra un ícono) que al pasar el mouse sobre el aparece el texto Añadir. Cuando el usuario haga clic sobre este botón, los campos (nombre de materia, paralelo y docente) van a

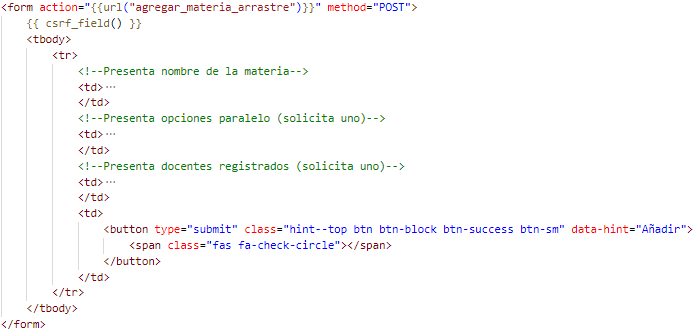


Figura 48. Código – vista completar registro (formulario añadir materia)

pasar por el método POST a la URL “agregar\_materia\_arrastre”; esta ruta se la crea en el archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 49 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “post” a la URL “agregar\_materia\_arrastre”, a su vez también se puede observar que la ruta apunta al controlador “AuthStudentController” y le pasa como método o función “agregar\_materia\_arrastre”.



Figura 49. Código – ruta para asignar materia (arrastre)

El controlador se encuentra dentro del directorio “app/Http/Controllers/AuthStudenController.php” y si analizamos este archivo, dentro de él se encuentra el método o función “agregar\_materia\_arrastre” tal y como se muestra en la Figura 50.

A continuación, se describe el código mostrado en la Figura 50.

* **if(Auth::check()):** comprueba si existe algún usuario autenticado. Solo entra cuando existe un usuario autenticado.
* **$user\_student:** guarda toda la información del usuario autenticado; la información se obtiene de la tabla users.
* **if($user\_student->is\_estudiante==true):** comprueba si el usuario es estudiante. Si es verdadero pasa a cumplir el resto de instrucciones.



Figura 50. Código – método asignar materia (arrastre)

* **if($user\_student->paralelo==’NA’ && $user\_student->ciclo==’NA’):** comprueba si el estudiante no tiene asignado (NA) paralelo y de igual manera el ciclo. Si para ambas condiciones la respuesta es verdadera, significa que el estudiante aún no ha completado el registro y pasa a cumplir el esto de instrucciones.
* **$data:** se hacen las validaciones para el formulario; los campos paralelo y docente son requeridos.
* **$materia\_id, $paralelo, $docente\_id:** guarda la materia, paralelo y docente que el usuario ha seleccionado en el formulario.
* **$materia:** guarda información de la materia; para ello la consulta en la tabla materias de la base de datos cuyo campo “id” es igual a $materia\_id.
* **$verifica\_arrastre:** guarda un valor (true o false); para ello comprueba en la tabla arrastres de la base de datos si existe algún registro en el campo “user\_estudiante\_id” igual al id ($user\_estudiante->id) del usuario autenticado.
* **if($verifica\_arrastre==true):** comprueba si la variable $verifica\_arrastre es verdadera; si es así, significa que el usuario ya ha asignado materias de arrastre y pasa a cumplir el resto de instrucciones.
* **$arrastre:** guarda información de las materias arrastradas; para ello obtiene el primer registro de la tabla arrastres en la base de datos, cuando el campo “user\_estudiante\_id” es igual al id ($user\_student->id) del usuario autenticado.
* **$id:** guarda solamente el id de la información almacenada en $arrastre.
* **$materia\_agregada:** guarda solamente las materias asignadas de la información almacenada en $arrastre.
* **$paralelo\_agregado:** guarda solamente los paralelos asignados de la información almacenada en $arrastre.
* **$docente\_agregado:** guarda solamente los docentes asignados de la información almacenada en $arrastre.
* **if($arrastre->materia==null):** comprueba si en el campo materia de la tabla arrastres está vacío; si es así significa que el usuario ha asignado una materia, pero a su vez la ha vuelto a eliminar dejando el campo vacío; por lo general no es muy probable que entre a esta condición si el usuario asigna las materias con la mayor seriedad. Cuando entra a esta condición primero se elimina de la base de datos toda la información de los campos que contenga el id ($id), luego se registra la materia que el usuario ha seleccionado, como ya no existe el $id en la base de datos, se registra la información nuevamente con este id ($id).
* En el **else** entra cuando existe una o más materias asignadas, por lo que pasa a cumplir el resto de instrucciones.

**$arreglo\_materia\_agregada:** guarda en un arreglo las materias que tiene asignadas, de la tabla arrastres.

**foreach($arreglo\_materia\_agregada as $recorre):** mediante este ciclo, se recorre todo el arreglo ($arreglo\_materia\_agregada); $recorre guarda la información del arreglo de acuerdo a la posición que se encuentre con cada incremento del foreach. Si se sigue analizando este ciclo, dentro del existe una sentencia **if($recorre==$materia\_id),** esta sentencia lo que hace es comprobar si el valor en la posición que se encuentre del arreglo ($recorre) es igual a la materia que el usuario ha seleccionado ($materia\_id); si son iguales significa que la materia seleccionada ya ha sido asignada anteriormente por el usuario, siendo así es direccionado a la ruta “vista\_completar\_registro” (ver Figura 42) que corresponde a la misma vista en que se encuentra pero se le mostrará el mensaje de que la materia ya ha sido asignada. En el caso de que no sean iguales significa que la materia seleccionada por el usuario, no se encuentra asignada y pasa a registrarla; para ello primero se elimina de la base de datos toda la información de los campos que contenga el id ($id), luego se registra la materia que el usuario ha seleccionado, como ya no existe el $id en la base de datos, se registra la información nuevamente con este id ($id).

* En el **else** entra cuando el estudiante no tiene asignada ninguna materia, por lo que registra la materia seleccionada con el paralelo y docente en la tabla arrastres; por lo general entra solo cuando asigna la primera materia.
* En el **else** entra cuando el usuario ya ha completado el registro; y lo direcciona a la vista “user\_student.auth\_student”; lo que significa que ha accedido al software y tiene acceso a todas sus funcionalidades.
* En el **else** entra cuando el usuario no es estudiante y lo direcciona a la ruta “show\_login\_form\_student”, que corresponde al formulario de inicio de sesión del estudiante (ver Figura 34).
  + - 1. **Finalizar registro**

Luego de que el usuario ha asignado o añadido todas las materias que se encuentra recibiendo en el periodo académico actual, ahora deberá finalizar el proceso de registro.



Figura 51. Código – vista completar registro (formulario finalizar registro con arrastre de materias)

En la Figura 51 se muestra el código correspondiente al formulario para completar registro o finalizar el proceso de registro, dentro de la vista “user\_student.completar\_registro”; se puede observar que existe un botón con el texto Finalizar registro. Cuando el usuario haga clic sobre este botón, los campos (paralelo, ciclo y contraseña) van a pasar por el método POST a la URL “save\_completar\_registro”; esta ruta se la crea en el archivo “*web.php*” dentro del directorio “*routes*”; algo a tener en cuenta en este formulario es que se agregó **{{**method\_field(‘PUT’)**}},** lo que significa que mediante ello vamos a editar la información de los campos (paralelo, ciclo y password), con la información enviada en el formulario para dichos campos; si bien es cierto el método con el que se pasa la información de los campos es de tipo “POST”, la ruta va hacer de tipo “PUT”. En la Figura 52 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “put” a la URL “save\_completar\_registro”, a su vez también se puede observar que la ruta apunta al controlador “AuthStudentController” y le pasa como método o función “save\_completar\_registro”.



Figura 52. Código – ruta para finalizar registro del estudiante

El controlador se encuentra dentro del directorio “app/Http/Controllers/AuthStudenController.php” y si analizamos este archivo, dentro de él se encuentra el método o función “buscar\_materia\_arrastre” tal y como se muestra en la Figura 53.

A continuación, se describe el código mostrado en la Figura 53.

* **if(Auth::check()):** comprueba si existe algún usuario autenticado. Solo entra cuando existe un usuario autenticado.
* **$user:** guarda toda la información del usuario autenticado; la información se obtiene de la tabla users.
* **if($user\_student->is\_estudiante==true):** comprueba si el usuario es estudiante. Si es verdadero pasa a cumplir el resto de instrucciones.
* **$data:** se hacen las validaciones para el formulario; los campos ciclo, paralelo y password (contraseña) son requeridos.
* **if($data[“ciclo]!=”arrastre”):** comprueba si el campo ciclo es diferente a arrastre, por lo general a esta sentencia va a entrar cuando el usuario complete registro sin arrastre de materias; aun así no es muy probable que entre si el usuario completa el registro con la mayor seriedad. Si esta sentencia es verdadera, en la variable $inf\_borrar, guarda información de la tabla arrastres cuyo campo “user\_estudiante\_id” es igual al id



Figura 53. Código – método completar proceso de registro

($user\_id) del usuario autenticado y luego pasa a eliminar toda esta información; en la tabla arrastres debe haber registros solamente cuando el usuario completa registro con arrastre de materias y como se mencionó a esta sentencia solo va a entrar cuando el proceso de completar registro es sin arrastre de materias.

* **if($data[“password”!=null]):** comprueba si la información enviada por el usuario en el campo password (contraseña) del formulario no está vacío; por lo general siempre va a entrar a esta sentencia, lo que significa que el usuario ha ingresado una contraseña para el registro, y luego se la pasa a encriptar.
* **$user->update($data):** actualiza la información del usuario en la tabla users de la base de datos con los datos obtenidos en $data.
* Se envía un mensaje por medio de flash() que luego es capturado en la vista.
* Por último, se direcciona a la ruta “auth\_student”, que corresponde a la página principal de la cuenta y el usuario dispondrá ya de todas las funcionalidades que el software le ofrece.
* En el **else** entra cuando el usuario no es estudiante y lo direcciona a la ruta “show\_login\_form\_student”, que corresponde al formulario de inicio de sesión del estudiante (ver Figura 34).
  + 1. **Completar registro sin arrastre de materias**

En la Figura 44, si el usuario contesta la respuesta negativamente (No), en el mismo archivo existe el formulario cuando el usuario no arrastra materias.



Figura 54. Código – vista completar registro (formulario finalizar registro sin arrastre de materias)

En la Figura 54 se muestra el código correspondiente al formulario para completar registro o finalizar el proceso de registro, dentro de la vista “user\_student.completar\_registro”; se puede observar que existe un botón con el texto Completar registro. Cuando el usuario haga clic sobre este botón, los campos (ciclo, paralelo y contraseña) van a pasar por el método POST a la URL “save\_completar\_registro”; esta ruta se la crea en el archivo “*web.php*” dentro del directorio “*routes*”; algo a tener en cuenta en este formulario es que se agregó **{{**method\_field(‘PUT’)**}},** lo que significa que mediante ello vamos a editar la información de los campos (paralelo, ciclo y password), con la información enviada en el formulario para dichos campos; si bien es cierto el método con el que se pasa la información de los campos es de tipo “POST”, la ruta va hacer de tipo “PUT”. La ruta y el método utilizado para cumplir con este proceso, ya se comentó anteriormente (ver sección 6.4, subsección 6.4.1, apartado 6.4.1.3).

* 1. **Vista general de la cuenta**

La vista general de la cuenta es un requisito que el administrador, estudiante y docente lo pueden hacer; a continuación, se presenta el código que permite la vista general de la cuenta para el administrador, para el estudiante y docente no se va a describir, debido a que es el mismo procedimiento que utiliza el administrador; sin embargo, en forma gráfica se va a presentar los métodos o funciones utilizados.

* + 1. **Vista general de la cuenta del administrador**

El administrador podrá ir a la vista general de su cuenta una vez que ha accedido al software. Una vez dentro de la cuenta, el administrador dispondrá de un menú de opciones. En la Figura 55 se muestra el código correspondiente a la vista de la opción en el menú, para la vista general de la cuenta; esta vista se encuentra dentro del directorio “views/user\_administrador/vistas\_iguales/menu\_vertical.blade.php”.

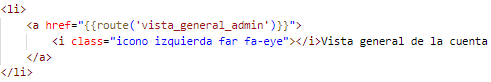


Figura 55. Código – Vista opción vista general de la cuenta (administrador)

En la Figura 55, se puede observar que existe un enlace (<a>) con el texto Vista general de la cuenta. Cuando el usuario haga clic sobre este enlace, va hacer dirigido a la ruta “vista\_general\_admin”; esta ruta se la crea en el archivo “*web.php*” dentro del directorio “*routes*”. En la Figura 56 se muestra la ruta codificada; se puede observar que la ruta declarada es de tipo “get” para la URL “vusta\_general\_admin”, a su vez también se puede observar que la ruta apunta al controlador “AuthAdministradorController” y le pasa como método o función “vista\_general\_admin”; **name** corresponde al nombre de la ruta, para el caso el nombre “vista\_general\_admin” fue utilizado en la vista dentro de la etiqueta “a” y el atributo “href” (ver Figura 55).



Figura 56. Código – ruta para vista general de la cuenta (administrador)

El controlador se encuentra dentro del directorio “*app/Http/Controllers/AuthAdministradorController.php*”, si analizamos este archivo, dentro de él se encuentra el método o función “vista\_general\_admin”, tal y como se muestra en la Figura 57.

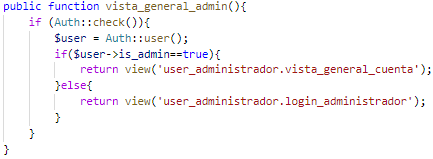


Figura 57. Código - método vista general de la cuenta (administrador)

A continuación, se describe el código mostrado en la Figura 57.

* **if(Auth::check()):** comprueba si existe algún usuario autenticado. Solo entra cuando existe un usuario autenticado.
* **$user:** guarda toda la información del usuario autenticado; la información se obtiene de la tabla users.
* **if($user\_student->is\_admin==true):** comprueba si el usuario es administrador. Si es verdadero lo direcciona a la vista “vista\_general\_cuenta” que se encuentra en la carpeta “user\_administrador”.
* En el **else** entra cuando el usuario no es administrador y lo direcciona a la vista “login\_administrador” que se encuentra en la carpeta “user\_administrador”; esta vista corresponde al formulario de inicio de sesión del administrador (ver Figura 29).

En la Figura 58 se muestra el código correspondiente a la vista “vista\_general\_cuenta”; esta vista se encuentra dentro del directorio “views/user\_administrador/vista\_general\_cuenta.blade.php”.

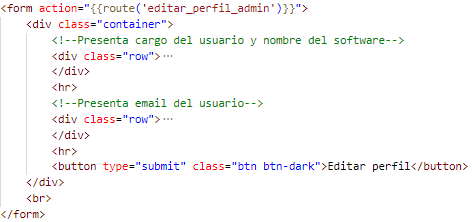


Figura 58. Código – Vista general de la cuenta (administrador)

En la Figura 58, se puede observar que existe un formulario y dentro de él se presenta en pantalla al usuario el cargo (administrador), nombre del software (SGT-CIS) y el correo (email); además también se encuentra un botón con el texto Editar perfil, que más adelante cuando se comente sobre el requisito editar perfil, se describirá el uso de este botón.

* + 1. **Vista general de la cuenta del estudiante**

El estudiante podrá ir a la vista general de su cuenta una vez que ha accedido al software. Una vez dentro de la cuenta, el estudiante dispondrá de un menú de opciones. El código generado para la vista general de la cuenta del estudiante es similar al utilizado para el administrador (ver sección 6.5, subsección 6.5.1), por lo que solamente se presentará en figuras el código, mas no su explicación.

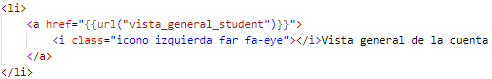


Figura 59. Código – Vista opción vista general de la cuenta (estudiante)



Figura 60. Código – ruta para vista general de la cuenta (estudiante)

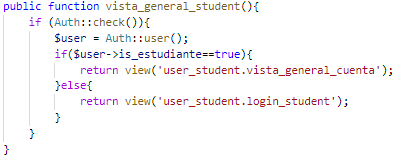


Figura 61. Código – método vista general de la cuenta (estudiante)

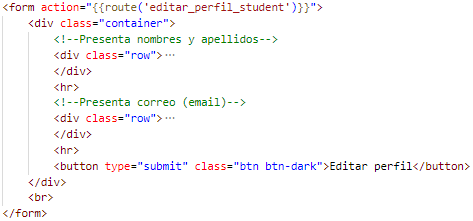


Figura 62. Código – Vista general de la cuenta (estudiante)

* + 1. **Vista general de la cuenta del docente**

El docente podrá ir a la vista general de su cuenta una vez que ha accedido al software. Una vez dentro de la cuenta, el docente dispondrá de un menú de opciones. El código generado para la vista general de la cuenta del docente es similar al utilizado para el administrador (ver sección 6.5, subsección 6.5.1), por lo que solamente se presentará en figuras el código, mas no su explicación.

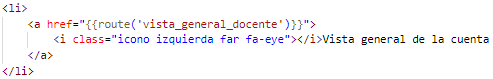


Figura 63. Código – Vista opción vista general de la cuenta (docente)



Figura 64. Código – ruta para vista general de la cuenta (docente)

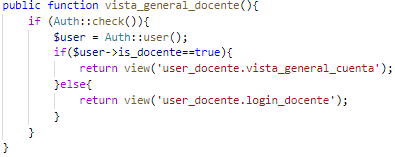


Figura 65. Código – método vista general de la cuenta (docente)

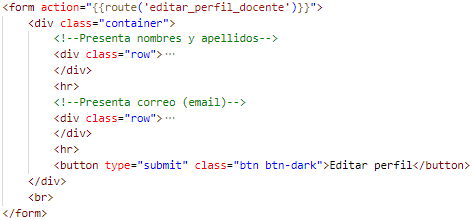


Figura 66. Código – Vista general de la cuenta (docente)

* 1. **Editar perfil**

Editar perfil es un requisito que el administrador, estudiante y docente lo pueden hacer; a continuación, se presenta el código que permite editar perfil para el administrador, para el estudiante y docente no se va a describir, debido a que es el mismo procedimiento que utiliza el administrador; sin embargo, en forma gráfica se va a presentar los métodos o funciones utilizados.

* + 1. **Editar perfil administrador**

En la Figura 58

* + 1. **Editar perfil estudiante**
    2. **Editar perfil docente**

1. <https://www.apachefriends.org/es/index.html> [↑](#footnote-ref-1)
2. <https://www.youtube.com/watch?v=7YvBOOSqM8k> [↑](#footnote-ref-2)
3. <https://laraveles.com/como-hacer-inicio-de-sesion-con-google-en-laravel/> [↑](#footnote-ref-3)
4. <https://console.developers.google.com/apis/dashboard?project=carbide-bonsai-197303&angularJsUrl>= [↑](#footnote-ref-4)
5. <https://www.w3schools.com/php/func_string_explode.asp> [↑](#footnote-ref-5)
6. <https://www.w3schools.com/php/func_string_explode.asp> [↑](#footnote-ref-6)
7. <https://www.w3schools.com/php/func_string_explode.asp> [↑](#footnote-ref-7)
8. <https://www.w3schools.com/php/func_string_explode.asp> [↑](#footnote-ref-8)