⚙ Configuración Técnica de Integración

Antes de instalar el módulo de autenticación, es importante considerar que la **Agencia de Innovación e Inteligencia Digital** cuenta con un repositorio privado para la distribución de librerías desarrolladas en **PHP**, orientadas principalmente a proyectos construidos con **Laravel**.

Este repositorio permite centralizar paquetes institucionales reutilizables, como módulos, componentes y librerías internas, evitando duplicar lógica entre proyectos y facilitando la estandarización del desarrollo.

Para poder instalar dependencias privadas de la AIIDT mediante Composer, es necesario configurar previamente las credenciales de acceso al repositorio privado a nivel global o a nivel de proyecto, según corresponda.

Consideración de red

Para instalar las librerías privadas de la AIIDT, es necesario estar conectado a la **red de Gobierno** o contar con acceso mediante **VPN a la red interna**, ya que el repositorio privado no se encuentra disponible públicamente en internet.

Una vez instalada la librería y descargadas las dependencias necesarias, se podrá trabajar normalmente en el proyecto Laravel.

Configuración de autenticación para Composer

Antes de ejecutar la instalación de las librerías, se debe registrar el token de acceso al repositorio privado mediante el siguiente comando:

```
composer config --global bearer.172.21.88.204:8002 "pkdt-srJNRh9XpMQreeh8uybq1euUOYhSTZkAqlhKXy6xcc1e389f"
```

Configuración del repositorio en `composer.json`

Además de configurar el token de acceso, se debe validar que el archivo `composer.json` del proyecto cuente con la configuración del repositorio privado.

En caso de no existir, agregar la siguiente configuración al final del archivo, antes del cierre de la última llave `}`:

```
"repositories": {
    "packistry": {
        "type": "composer",
        "url": "http://172.21.88.204:8002/r/aiidt-librerias"
    }
}
```

Ejemplo de referencia dentro del archivo `composer.json`:

```
{
    "name": "laravel/laravel",
    "type": "project",
    "require": {
        "php": "^8.2"
    },
    "repositories": {
        "packistry": {
            "type": "composer",
            "url": "http://172.21.88.204:8002/r/aiidt-librerias"
        }
    }
}
```

Configuración de `secure-http`

Debido a que actualmente el repositorio privado se encuentra publicado mediante `http`, es necesario agregar la siguiente configuración dentro de la llave `config` del archivo `composer.json`:

```
"secure-http": false
```

Ejemplo de referencia:

```
{
    "config": {
        "optimize-autoloader": true,
        "preferred-install": "dist",
        "sort-packages": true,
        "secure-http": false
    }
}
```

> **Nota:** Esta configuración permite que Composer pueda descargar paquetes desde repositorios que no utilizan HTTPS. De momento es necesaria para consumir el repositorio privado interno de la AIIDT.

Instalación de dependencias

Para habilitar la autenticación mediante **FusionAuth** en un proyecto Laravel, es necesario instalar el **módulo de autenticación desarrollado por la AIIDT**, el cual contiene la estructura base requerida para gestionar el flujo de inicio de sesión, retorno de autenticación, cierre de sesión, controlador principal y rutas necesarias para la integración con FusionAuth.

Este módulo permite estandarizar la implementación de autenticación en los proyectos Laravel, evitando duplicar lógica entre aplicaciones y facilitando el mantenimiento centralizado del proceso de autenticación institucional.

Para su instalación, el proyecto debe contar previamente con la librería de módulos de Laravel:

```
composer require nwidart/laravel-modules:^12.0
```

Posteriormente, se deben instalar las siguientes dependencias mediante **Composer**:

```
composer require joshbrw/laravel-module-installer:^2.0
```

```
composer require aiidt/autenticacion-module:^1.1
```

La primera dependencia, `joshbrw/laravel-module-installer`, permite la instalación automática de módulos dentro de la estructura del proyecto Laravel. Esta librería se encarga de ubicar correctamente el módulo dentro del directorio correspondiente, de acuerdo con la configuración del proyecto.

La segunda dependencia, `aiidt/autenticacion-module`, corresponde al módulo de autenticación como tal. Este módulo contiene la lógica necesaria para integrar Laravel con FusionAuth, incluyendo rutas, controladores y configuraciones base requeridas para completar el flujo de autenticación.

Se recomienda **no modificar directamente el módulo de autenticación**, ya que al ser administrado como una librería instalada mediante Composer, cualquier cambio local puede perderse al reinstalar dependencias, actualizar el paquete o desplegar nuevamente el proyecto. En caso de requerir ajustes específicos, estos deberán realizarse mediante configuración externa, extensión del comportamiento o personalización desde el proyecto consumidor, sin alterar el código fuente del módulo.

Con esta configuración, el proyecto queda preparado para incorporar el flujo de autenticación centralizada con FusionAuth de forma ordenada, reutilizable y alineada con el estándar institucional de integración.

Configuración de servicios

En el archivo `config/services.php`, se debe reemplazar o agregar la configuración de `fusionauth` con la siguiente estructura:

```
'fusionauth' => [
    'client_id' => env('FUSIONAUTH_CLIENT_ID'),
    'client_secret' => env('FUSIONAUTH_CLIENT_SECRET'),
    'redirect' => env('FUSIONAUTH_REDIRECT_URI'),
    'base_url' => env('FUSIONAUTH_BASE_URL'),
    'tenant_id' => env('FUSIONAUTH_TENANT_ID'),
    'redirect_home' => env('FUSIONAUTH_REDIRECT_HOME'),
    'url_logout' => env('FUSIONAUTH_URL_LOGOUT'),
],
```

Esta configuración permite que Laravel obtenga desde el archivo `.env` los datos necesarios para comunicarse con FusionAuth, incluyendo las credenciales de la aplicación, la URL base del servicio, el tenant correspondiente, la ruta de retorno después del inicio de sesión y la URL utilizada para cerrar sesión.

Variables de entorno

En el archivo `.env` del proyecto Laravel, se deben agregar las siguientes variables de entorno:

```
FUSIONAUTH_CLIENT_ID=tu_client_id
FUSIONAUTH_CLIENT_SECRET=tu_client_secret
FUSIONAUTH_REDIRECT_URI=${APP_URL}/autenticacion/callback
FUSIONAUTH_BASE_URL=https://acceso.tamaulipas.gob.mx
FUSIONAUTH_TENANT_ID=d9220956-99f7-431f-8de1-de37e65488a8
FUSIONAUTH_REDIRECT_HOME="/inicio"
FUSIONAUTH_URL_LOGOUT="${FUSIONAUTH_BASE_URL}/oauth2/logout?client_id=${FUSIONAUTH_CLIENT_ID}"
```

Estas variables permiten parametrizar la integración sin modificar directamente el código fuente del proyecto, facilitando su mantenimiento y despliegue entre distintos ambientes.

Consideraciones importantes

Es importante tomar en cuenta las siguientes consideraciones antes de realizar la integración:

No se deben modificar las siguientes variables:

```
FUSIONAUTH_REDIRECT_URI=${APP_URL}/autenticacion/callback
FUSIONAUTH_BASE_URL=https://acceso.tamaulipas.gob.mx
```

```
FUSIONAUTH_TENANT_ID=d9220956-99f7-431f-8de1-de37e65488a8
```

La variable `FUSIONAUTH_REDIRECT_URI` define la ruta de callback a la que FusionAuth redirigirá al usuario después de completar el proceso de autenticación. Esta URI debe coincidir exactamente con la configurada en FusionAuth, ya que cualquier diferencia en dominio, protocolo, ruta o prefijo puede provocar errores durante el inicio de sesión.

Asimismo, `FUSIONAUTH_BASE_URL` y `FUSIONAUTH_TENANT_ID` corresponden a la configuración institucional del servicio de autenticación, por lo que deben conservarse sin cambios para garantizar que la aplicación se conecte al tenant correcto.

Registro del Provider de Socialite

Para que Laravel Socialite pueda reconocer y utilizar el proveedor de **FusionAuth**, es necesario registrar el provider correspondiente dentro del archivo:

```
app/Providers/EventServiceProvider.php
```

En este archivo, se debe agregar la siguiente configuración **solo en caso de que no exista previamente**:

```
protected $listen = [
    \SocialiteProviders\Manager\SocialiteWasCalled::class => [
        \SocialiteProviders\FusionAuth\FusionAuthExtendSocialite::class . '@handle',
    ],
];
```

Configuración del Modelo `User`

Para almacenar correctamente la información recibida desde **FusionAuth**, es necesario actualizar el modelo `User` del proyecto Laravel.

Agregar campos a `$fillable`

En el archivo del modelo `User`, normalmente ubicado en:

```
app/Models/User.php
```

se deben agregar los siguientes campos dentro del arreglo `$fillable`:

```
protected $fillable = [
    'name',
```

```
    'email',
    'profile_photo_url',
    'password',
    'fusionauth_id',
    'fusionauth_access_token',
    'fusionauth_refresh_token',
    'data',
    'registration_data',
];
```

Estos campos permiten guardar la información básica del usuario, así como los identificadores, tokens y datos adicionales provenientes de FusionAuth.

Agregar casting de datos

También se debe agregar o actualizar la propiedad `$casts` dentro del mismo modelo:

```
protected $casts = [
    'data' => 'array',
    'registration_data' => 'array',
];
```

Esto permite que los campos `data` y `registration_data` sean tratados automáticamente como arreglos en Laravel, evitando estar haciendo `json_decode()` y `json_encode()`.

Con esta configuración, el modelo `User` quedará preparado para manejar la información extendida del usuario autenticado mediante FusionAuth de forma limpia y estructurada.

Modificación de la tabla `users`

Para que Laravel pueda almacenar correctamente la información que regresa **FusionAuth** durante el proceso de autenticación, es necesario agregar campos adicionales a la tabla `users`.

Estos campos permiten guardar el identificador único del usuario en FusionAuth, los tokens de autenticación, la fotografía de perfil y la información adicional del usuario y su registro dentro de la aplicación.

Opción A: Mediante migración

Esta opción es la **recomendada**, especialmente cuando el proyecto se encuentra en etapa inicial o cuando aún no existen usuarios registrados en la base de datos.

La migración de la tabla `users` deberá contemplar la siguiente estructura:

```
public function up(): void
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password')->nullable();
        $table->rememberToken();

        $table->string('fusionauth_id', 36)->unique();
        $table->text('fusionauth_access_token');
        $table->text('fusionauth_refresh_token')->nullable();
        $table->string('profile_photo_url', 500)->nullable();
        $table->json('data')->nullable();
        $table->json('registration_data')->nullable();

        $table->timestamps();
        $table->softDeletes();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists('users');
}
```

Con esta modificación, la tabla `users` queda preparada para recibir y persistir la información necesaria para operar con FusionAuth dentro del proyecto Laravel.

Los campos agregados cumplen la siguiente función:

Campo	Descripción
`fusionauth_id`	Identificador único del usuario dentro de FusionAuth.
`fusionauth_access_token`	Token de acceso generado durante el proceso de autenticación.
`fusionauth_refresh_token`	Token utilizado para renovar la sesión o el acceso, cuando aplique.

Campo	Descripción
`profile_photo_url`	URL de la fotografía de perfil del usuario.
`data`	Información adicional del usuario proveniente de FusionAuth.
`registration_data`	Información relacionada con el registro del usuario dentro de la aplicación en FusionAuth.

Es importante considerar que el campo `password` se define como `nullable`, ya que la autenticación no será administrada directamente por Laravel, sino por FusionAuth. En otras palabras.

Opción B: Mediante SQL

Como alternativa, también es posible agregar los campos requeridos directamente mediante una consulta SQL sobre la tabla `users`.

Sin embargo, esta opción **no es la recomendada**, ya que el cambio no queda registrado dentro del sistema de migraciones de Laravel. Esto implica que, en cada nuevo despliegue, reinstalación del proyecto o preparación de ambiente, será necesario ejecutar manualmente el query para asegurar que la tabla tenga la estructura correcta.

```
ALTER TABLE users
    MODIFY COLUMN password VARCHAR(255) NULL,
    ADD COLUMN fusionauth_id VARCHAR(36) NOT NULL UNIQUE AFTER remember_token,
    ADD COLUMN fusionauth_access_token TEXT NOT NULL AFTER fusionauth_id,
    ADD COLUMN fusionauth_refresh_token TEXT NULL AFTER fusionauth_access_token,
    ADD COLUMN profile_photo_url VARCHAR(500) NULL AFTER fusionauth_refresh_token,
    ADD COLUMN data JSON NULL AFTER profile_photo_url,
    ADD COLUMN registration_data JSON NULL AFTER data,
    ADD COLUMN deleted_at TIMESTAMP NULL AFTER updated_at;
```

Esta consulta realiza las siguientes acciones:

Acción	Descripción
`MODIFY COLUMN password VARCHAR(255) NULL`	Permite que el campo `password` pueda ser nulo, ya que la autenticación será gestionada por FusionAuth.
`fusionauth_id`	Agrega el identificador único del usuario en FusionAuth.
`fusionauth_access_token`	Agrega el token de acceso generado durante el inicio de sesión.
`fusionauth_refresh_token`	Agrega el token de renovación, cuando aplique.
`profile_photo_url`	Agrega el campo para almacenar la URL de la fotografía de perfil del usuario.

Acción	Descripción
`data`	Agrega un campo JSON para guardar información adicional del usuario.
`registration_data`	Agrega un campo JSON para guardar información relacionada con el registro del usuario en la aplicación.
`deleted_at`	Agrega soporte para eliminación lógica mediante `SoftDeletes`.

> **Nota:** Si la tabla `users` ya cuenta con alguno de estos campos, el query puede generar errores por columnas duplicadas. En ese caso, se deberán validar previamente los campos existentes antes de ejecutar la consulta.

Prueba de integración

Después de completar la instalación de dependencias, configuración de servicios, variables de entorno, registro del provider, actualización del modelo `User` y modificación de la tabla `users`, se puede realizar una prueba inicial para validar que la integración con **FusionAuth** esté funcionando correctamente.

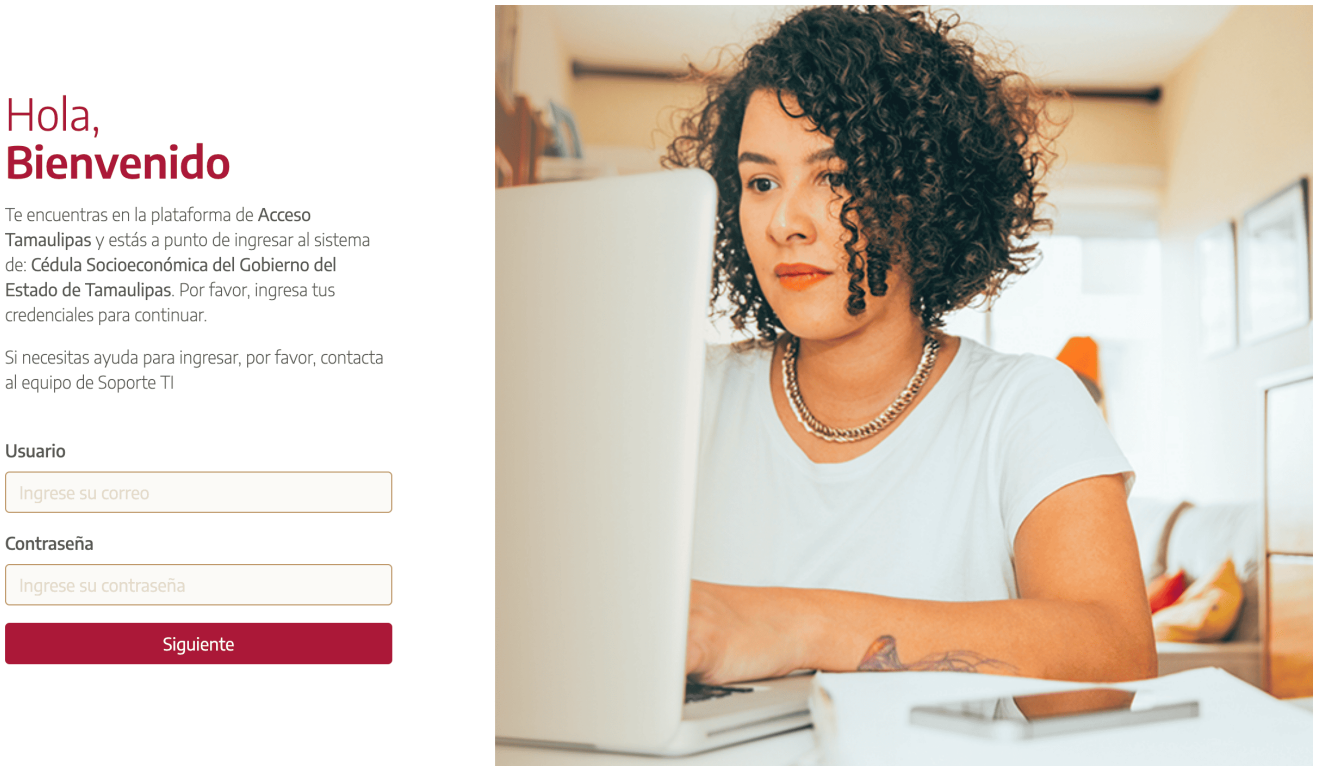
Para ello, se debe ingresar desde el navegador a la siguiente URL:

```
http://${base_url}/autenticacion/login
```

Donde `${base_url}` corresponde al dominio o URL base donde se encuentra publicado el proyecto Laravel.

Al acceder a esta ruta, la aplicación deberá redirigir automáticamente al formulario de inicio de sesión de **FusionAuth**.

La pantalla esperada debe visualizarse de la siguiente manera:



Si se muestra el login de FusionAuth, significa que la aplicación Laravel está redirigiendo correctamente al proveedor de autenticación y que la configuración base del flujo de inicio de sesión se encuentra operativa.

En caso de que no se visualice esta pantalla, o se presente algún error durante la redirección, se recomienda consultar la sección de **Troubleshooting**, donde se describen los errores más comunes y sus posibles soluciones.

Revision #15
Created 29 April 2026 19:26:30 by Gonzalo Cesar Raymundo Martinez Resendez
Updated 6 May 2026 20:28:45 by Gonzalo Cesar Raymundo Martinez Resendez