Manual arquetipo Laravel con LlaveCDMX

LLAVE - Inicio de Sesión.

Fecha: 16/14/2020

Plaza de las Vizcaínas 30, Colonia Centro, Alcaldía Cuauhtémoc, C.P. 06080, Ciudad de México.
Tel.01 55 3090 0500

Control de versiones

Versión	Fecha	Descripción del cambio	Responsable
0.1	16/04/2020	Creación del documento	José Guillermo Hernández Cabrera
0.2	01/05/2020	Añadidos los imports a las generalidades de AdipUtils	José Guillermo Hernández Cabrera
0.3	07/05/2020	Agregado el componente de envío de correos	José Guillermo Hernández Cabrera

Objetivo del documento

Mostrar los pasos a seguir para la instalación del arquetipo Laravel 7.5 con inicio de sesión LlaveCDMX integrado.
Mostrar al desarrollador el uso de los componentes incluidos con el arquetipo.

Introducción

La mayoría de los proyectos creados en la agencia, requerirán que el ciudadano utilice una cuenta Llave CDMX, para evitar que cada sistema nuevo utilice una implementación diferente, se debe usar este arquetipo en vez de una instalación nueva de Laravel.
La instalación del arquetipo incluye una instancia Laravel 7.5.1 preparada para utilizar los servicios de inicio de sesión del sistema Llave CDMX. La lista completa de componentes que se instalan es la siguiente:

- Laravel 7.5.1
- AdipUtils
- Bootstrap
- Datatables Core
- Datatables Buttons
- Fontawesome

Requisitos de instalación

Para la correcta instalación del arquetipo se requiere lo siguiente

- Apache 2.4
- PHP 7.2 o superior (se recomienda PHP 7.3.9)
- Modrewrite activado
- Extension cURL de PHP activada
- Git
- Composer
- npm
- Repositorio remoto donde se alojará la aplicación a desarrollar

Pasos para instalación del arquetipo

1.- Registro de la aplicación en Llave:

Para poder utilizar el servicio de autenticación de Llave CDMX es necesario que la aplicación a desarrollar sea dada de alta en el ***Catálogo de Sistemas de Llave CDMX***. Para ello será necesario proporcionar la siguiente información a un administrador de Llave CDMX para que la dé de alta:

- *Nombre de la aplicación (Max. 60 caracteres).*
- *Clave de la aplicación (Max. 10 caracteres).*
- *URL a la que se redirecciona una vez realizada la autenticación (Max. 60 caracteres).*

De manera confidencial el administrador de Llave responderá con:

- **Id del cliente de llave:** Un identificador único para nuestra aplicación.
- **Código secreto:** Un código que se usará para consumir los servicios de autenticación.
- **Domain user:** Un usuario para Basic Auth que se empleará al consumir los servicios de autenticación de Llave.
- **Domain Password:** Una contraseña para el usuario citado arriba.

Los datos que el administrador de Llave entregue al desarrollador de la aplicación son considerados información sensible. No deben ser proporcionados a nadie más.

2.- Clonar el repositorio del arquetipo:

Mediante línea de comandos obtenemos la última versión del arquetipo, para ello se debe ejecutar la siguiente instrucción en la carpeta donde se va a ser instalado (usualmente %documentroot% o similar):

```
git clone https://codigofuente.cdmx.gob.mx/adip.dev/php/laravel/archetype_laravel7.5_with_llavecdmx.git
```

La consola responderá con algo similar a esto:

```
Cloning into 'archetype_laravel7.5_with_llavecdmx'...
remote: Enumerating objects: 325, done.
remote: Counting objects: 100% (325/325), done.
remote: Compressing objects: 100% (230/230), done.
remote: Total 325 (delta 69), reused 325 (delta 69)
Receiving objects: 100% (325/325), 2.21 MiB | 2.99 MiB/s, done.
Resolving deltas: 100% (69/69), done.
```

Y tras unos instantes tendremos una carpeta llamada archetype_laravel7.5_with_llavecdmx en cuyo interior está el código fuente del arquetipo. Para comprobarlo navegamos a la carpeta antes mencionada y listamos su contenido con el comando dir (Windows) o ls -la (Linux)

```
cd archetype_laravel7.5_with_llavecdmx
dir
```

Dependiendo el sistema operativo que se esté utilizando, la respuesta de la consola podrá parecerse en mayor o menor medida a lo siguiente:

```
Directorio de C:\xampp\htdocs\archetype_laravel7.5_with_llavecdmx

16/04/2020  01:28 p. m.    <DIR>          .
16/04/2020  01:28 p. m.    <DIR>          ..
16/04/2020  01:28 p. m.               235 .editorconfig
16/04/2020  01:28 p. m.             1,667 .env.example
16/04/2020  01:28 p. m.               116 .gitattributes
16/04/2020  01:28 p. m.               175 .gitignore
16/04/2020  01:28 p. m.               187 .styleci.yml
16/04/2020  01:28 p. m.    <DIR>          app
16/04/2020  01:28 p. m.             1,739 artisan
16/04/2020  01:28 p. m.             1,453 package.json
16/04/2020  01:28 p. m.             1,168 phpunit.xml
16/04/2020  01:28 p. m.    <DIR>          public
16/04/2020  01:28 p. m.             1,445 README.md
16/04/2020  01:28 p. m.    <DIR>          resources
16/04/2020  01:28 p. m.    <DIR>          routes
16/04/2020  01:28 p. m.               584 server.php
16/04/2020  01:28 p. m.    <DIR>          storage
16/04/2020  01:28 p. m.    <DIR>          tests
16/04/2020  01:28 p. m.               611 webpack.mix.js
              16 archivos        698,190 bytes
              11 dirs  23,304,900,608 bytes libres
```

3.- Actualizar las referencias de los repositorios:

Para evitar subir cambios de manera accidental al repositorio original del arquetipo, y para poder comenzar a trabajar con el repositorio de la aplicación a desarrollar se debe de actualizar la referencia a los mismos. Para ello, dentro de la carpeta donde fue clonado el arquetipo, ejecutamos los siguientes comandos:

```
git remote -v
```

La consola responderá con algo similar a lo siguiente:

```
origin  https://codigofuente.cdmx.gob.mx/adip.dev/php/laravel/archetype_laravel7.5_with_llavecdmx.git
(fetch)
origin  https://codigofuente.cdmx.gob.mx/adip.dev/php/laravel/archetype_laravel7.5_with_llavecdmx.git
(push)
```

Donde podemos apreciar el nombre y URL del repositorio de donde fue clonado el arquetipo. Continuamos con:

```
git remote rename origin old-origin
git remote add origin https://underdog1987@bitbucket.org/underdog1987/prueba-manual-llavecdmx.git
git remote rm old-origin
```

> **Nota:** https://underdog1987@bitbucket.org/underdog1987/prueba-manual-llavecdmx.git es un repositorio de ejemplo, dicha URL deberá ser reemplazada por la URL del proyecto, proporcionada por el Director de Desarrollo Tecnológico.

Verificamos nuevamente con

```
git remote -v
```

La consola responderá algo como:

```
origin  https://underdog1987@bitbucket.org/underdog1987/prueba-manual-llavecdmx.git (fetch)
origin  https://underdog1987@bitbucket.org/underdog1987/prueba-manual-llavecdmx.git (push)
```

> **Nota:** Se debe apreciar solo una referencia, la cual "*apunta*" al repositorio de la aplicación.

Por último, hacemos push al repositorio de la aplicación.

```
git push -u origin master
```

A partir de este punto, ya tenemos un proyecto nuevo basado en el arquetipo.

4.- Renombrar la carpeta del proyecto:

La carpeta donde está instalado el proyecto, tiene un nombre como *archetype_laravel7.5_with_llavecdmx*, usando la consola de comandos o un explorador de archivos, le ponemos un nombre más descriptivo. En este documento se usará a manera de ejemplo *LlaveCDMX*

5.- Realizar la instalación con del proyecto

En la carpeta donde está el proyecto, ejecutamos:

```
composer install
```

Composer instalará Laravel, proceso que llevará uno minutos, después, creamos el archivo .env usando el que el arquetipo tenía de ejemplo, para ello escribimos el siguiente comando de acuerdo a nuestro sistema operativo:

```
move .env.example .env (Windows)
mv .env.example .env (Linux)
```

Enseguida, generamos la llave de la aplicación.

```
php artisan key:generate
```

6.- Configuración del archivo .env

Los parámetros proporcionados por el administrador del sistema Llave para realizar la autenticación deben ser colocados el archivo .env. También debemos modificar otros valores para que la implementación de Llave funcione correctamente, para ello debemos asegurarnos de crear o modificar los siguientes valores en dicho archivo:

APP_NAME

Establecemos el nombre de la aplicación. Laravel usará este valor para fijar el título en la barra de título del navegador. Ejemplo

```
APP_NAME="Llave CDMX Cliente Laravel"
```

APP_DEPENDENCIA

El valor de este atributo formará parte del pie de página. Lo más recomendable es colocar la leyenda “Operado por” y el nombre de la dependencia que usará la aplicación.

```
APP_DEPENDENCIA="Operado por Nombre de Dependencia"
```

APP_DESCRIPTION

En este atributo colocamos una descripción introductoria de la aplicación. Por ejemplo “Bienvenido, realiza trámites de ….”

```
APP_DESCRIPTION="Bienvenido, descripcion simple de lo que hace la app."
```

APP_URL

En este atributo se debe especificar la URL desde la que será accesible la aplicación. Esto es sin incluir la carpeta public. Por ejemplo:

```
APP_URL=http://adip.io/LlaveCDMX
```

DB_CONNECTION
Especificar el nombre de la conexión que se va a utilizar en la aplicación. Usualmente pgsql

```
DB_CONNECTION=pgsql
```

DB_HOST
Especificar la dirección del servidor de la base de datos. Ejemplo:

```
DB_HOST=127.0.0.1
```

DB_PORT
Especificar el puerto en el que escucha el servidor de la base de datos. Para pgsql, usualmente es el 5432.

```
DB_PORT=5432
```

DB_DATABASE
Especificar el nombre de la base de datos a utilizar. Ejemplo

```
DB_DATABASE=app_db
```

DB_USERNAME
Especificar nombre de usuario de la base de datos. Ejemplo:

```
DB_USERNAME=app_dbu
```

DB_PASSWORD
Especificar la contraseña de la base de datos. Ejemplo:

```
DB_PASSWORD=app_dbpass
```

SESSION_DRIVER
Para funcionar adecuadamente, la aplicación debe utilizar como manejador de sesiones la base de datos, ya que esto es parte importante de la implementación de LlaveCDMX que incluye el arquetipo.

```
SESSION_DRIVER=database
```

SESSION_PATH
En este atributo se debe especificar la ruta en la cual se almacenarán las cookies de sesión y la cookie CSRF. Para evitar ataques de robo de sesión la ruta especificada debería lo más restrictiva posible. En dominios que hospeden más de una aplicación Laravel es obligatorio que apunte a “public”. La primera diagonal es obligatoria, la última diagonal no debe ser incluida. Ejemplo:

```
SESSION_PATH=/LlaveCDMX/public
```

MAIL_FROM_NAME

Este atributo define el nombre del remitente en el campo "*From:*" en el envío de correos electrónicos - siempre que en el código de envío de correos se use `env('MAIL_FROM_NAME')` para fijar el valor.
De manera predeterminada utilizará el valor del campo APP_NAME, aunque se puede cambiar.
Ejemplo:

```
MAIL_FROM_NAME="${APP_NAME}"
```

MAIL_FROM_ADDRESS

Este atributo define la dirección de correo del remitente en el campo "*From:*" en el envío de correos electrónicos - siempre que en el código de envío de correos se use `env('MAIL_FROM_ADDRESS')` para fijar el valor.
Es común que este valor tenga que coincidir con el establecido en el API de Mandrill (ver sección "Envío de correos"). Ejemplo:

```
MAIL_FROM_ADDRESS=no-reply@cdmx.gob.mx
```

LLAVE_CLIENT_ID

Especificar en este atributo el ID de sistema proporcionado por el administrador del sistema Llave

```
LLAVE_CLIENT_ID=XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

LLAVE_URL_REDIRECT

En este valor se especifica la URL a la que direcciona el sistema llave una vez que se lleve a cabo un inicio de sesión exitoso.

```
LLAVE_URL_REDIRECT=http://adip.io/LlaveCDMX/public/home
```

LLAVE_APP_SECRET

En este atributo se especifica el código secreto proporcionado por el administrador del sistema Llave.

```
LLAVE_APP_SECRET=XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

LLAVE_SERVER
LLAVE_GET_TOKEN
LLAVE_GET_USER
LLAVE_GET_ROLES
LLAVE_CREATE_ACCOUNT

Estos atributos especifican end-points del sistema Llave, son los mismos para cualquier sistema. Los ejemplos muestran los valores para ambientes locales y de desarrollo.

```
LLAVE_SERVER=https://llave-dev.cdmx.gob.mx/oauth.xhtml
LLAVE_GET_TOKEN=https://llave-dev.cdmx.gob.mx/rest/oauth/token
LLAVE_GET_USER=https://llave-dev.cdmx.gob.mx/rest/oauth/usuario
LLAVE_GET_ROLES=https://llave-dev.cdmx.gob.mx/rest/oauth/roles
LLAVE_CREATE_ACCOUNT=https://llave-dev.cdmx.gob.mx/RegistroCiudadano.xhtml
```

LLAVE_DOMAIN_USER

En este atributo se debe especificar el usuario para Basic-Auth de los end-points de Llave. Este valor es proporcionado por el administrador del sistema Llave al solicitar el alta de nuestra aplicación.

```
LLAVE_DOMAIN_USER=admin
```

LLAVE_DOMAIN_PASSWORD

Especificar la contraseña para Basic-Auth de los end-points de Llave. Este valor es proporcionado por el administrador del sistema Llave al solicitar el alta de nuestra aplicación.

```
LLAVE_DOMAIN_PASSWORD=admin
```

MANDRILL_SECRET

Establece el API KEY de Mandrill que se usará para enviar correos electrónicos dentro del aplicativo.

```
MANDRILL_SECRET=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXxxxx
```

MANDRILL_URL

Establece la URL del API JSON de Mandrill. Se debe usar de esta manera, ya que a partir de Laravel 6.0 el driver de Mandrill fue eliminado, por lo que no es posible enviar correos con Mandrill usando `Mail::send()`.

```
MANDRILL_URL= https://mandrillapp.com/api/1.0/messages/send.json
```

7.- Instalar Doctrine

Doctrine es un componente de abstracción (DataBase Abstraction Layer) que permite que en las migraciones de Laravel se puedan renombrar columnas. Lo instalamos con el siguiente comando:

```
composer require doctrine/dbal
```

8.- Instalar componentes de *Front-end*

Las vistas del *front-end* en el arquetipo están basadas en Boostrap y otros componentes. Para instalar toda la parte del front, ejecutamos los siguientes comandos:

```
npm install
npm install datatables.net
npm install datatables.net-dt
npm install datatables.net-buttons
npm install datatables.net-buttons-dt
npm install --save @fortawesome/fontawesome-free
npm run dev
```

9.- Migrar y poblar la base de datos

Para crear la estructura necesaria de la base de datos basta con ejecutar una migración como normalmente se hace.

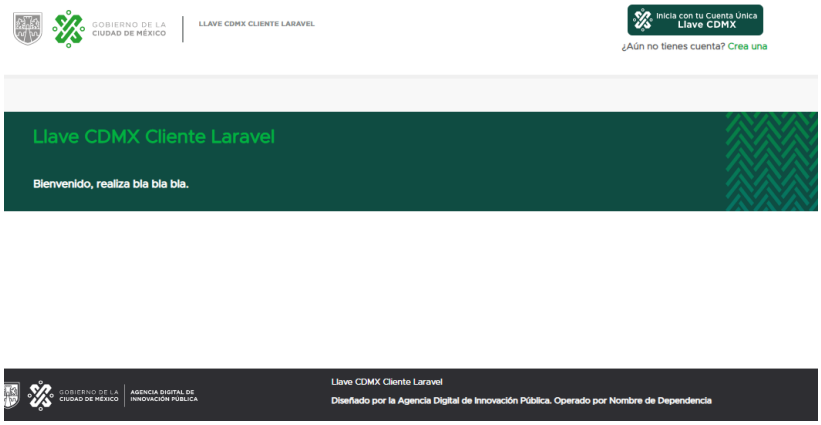
```
php artisan:migrate
```

Seguido de

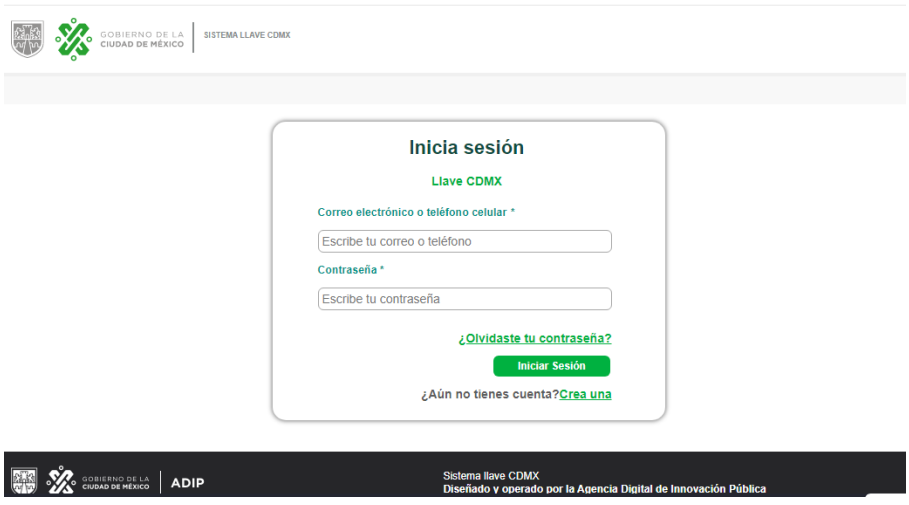
```
php artisan db:seed
```

10.- Comprobar el resultado de la instalación

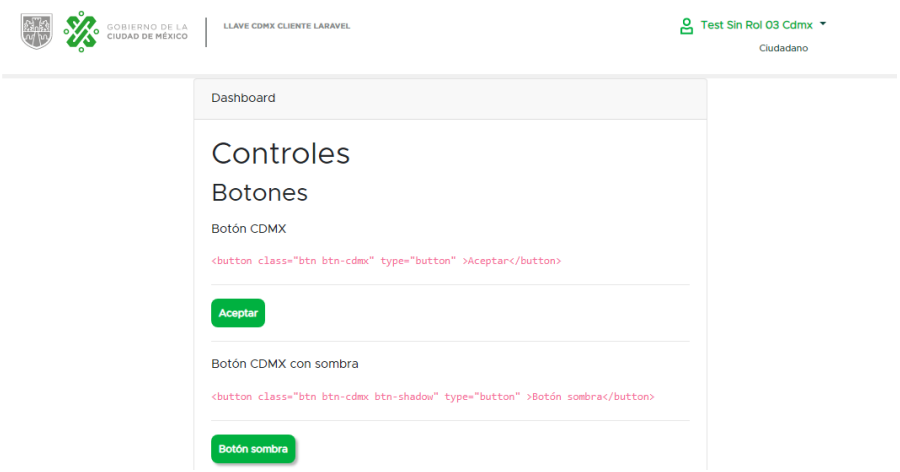
Por el solo hecho de haber sido instalada, la aplicación de ejemplo debe ser capaz de iniciar sesión usando la autenticación de Llave CDMX.
Para comprobarlo accedemos con cualquier navegador a la dirección de la aplicación (misma que está en la variable APP_URL del .env).
Se debe mostrar una página como la siguiente:



Al hacer clic en el botón *"Inicia con tu cuenta única Llave CDMX"* debemos ser redirigidos a la página de Login del sistema Llave CDMX



En esta página debemos introducir unas credenciales válidas y hacer clic en *"Iniciar sesión"*. Si el proceso de autenticación es exitoso seremos redirigidos al Home de la aplicación ejemplo.



Arquitectura backend

El arquetipo incluye diversos componentes que se emplean para realizar la autenticación con el sistema LlaveCDMX, aunque algunos de ellos pueden ser reutilizados (ver AdipUtils)

Base de datos

Para manejar las sesiones, usuarios y permisos se utilizan tablas en la base de datos, **la estructura de estas en mostrada de manera informativa y no debe ser modificada**.

Tabla: apps_api
Almacena las aplicaciones y tokens que podrán consumir servicios expuestos en la App. (ver apartado Exponer API)

Nombre de campo	Tipo de datos	Descripción	Extras
id	Bigint	ID de aplicación	PK, AUTOINCREMENT
nb_aplicacion	Character Varying(250)	Nombre de la aplicación	NOT NULL
tx_descripcion_app	Character Varying(250)	Descripción de la aplicación	NOT NULL
api_token	Character Varying(80)	Token	UNIQUE, NOT NULL
st_activo	Smallint	0 App deshabilitada 1 App habilitada	DEFAULT 1, NOT NULL
created_at	Timeztamp w/tz	Fecha de creación	
updated_at	Timestamp w/tz	Fecha de modificación	

Tabla: permisos

Catálogo de permisos de la aplicación

Nombre de campo	Tipo de datos	Descripción	Extras
id	Bigint	ID de permiso	PK, AUTOINCREMENT
nb_permiso	Character Varying(100)	Nombre del permiso	NOT NULL
created_at	Timeztamp w/tz	Fecha de creación	
updated_at	Timestamp w/tz	Fecha de modificación	

Tabla: users

Catálogo de usuarios que han iniciado sesión

Nombre de campo	Tipo de datos	Descripción	Extras
idUsuario	Bigint	ID de permiso	PK
login	Character Varying(150)	Dirección de correo electrónico / nombre de usuario	NOT NULL, UNIQUE
nombre	Character Varying(100)	Nombre del usuario	NOT NULL
primerApellido	Character Varying(100)	Apellido paterno	NOT NULL
segundoApellido	Character Varying(100)	Apellido materno	
telVigente	Character Varying(10)	Teléfono móvil	
curp	Character Varying(18)	Clave Única del Registro de Población	UNIQUE, NOT NULL
fechaNacimiento	Timestamp w/tz	Fecha de nacimiento	NOT NULL
idEstadoNacimiento	Integer	ID del estado de nacimiento	NOT NULL, UNSIGNED
estadoNacimiento	Character Varying(50)	Nombre del estado de nacimiento	NOT NULL
idRolUsuario	Integer	ID del Rol del usuario	
descripcionRol	Character Varying(100)	Nombre del rol de usuario	

Tabla: permiso_user

Tabla relacional para asignar permisos a usuarios

Nombre de campo	Tipo de datos	Descripción	Extras
idUsuario	Bigint	ID de permiso	PK, NOT NULL
idPermiso	Bigint	ID de permiso	PK, NOT NULL

Tabla: t001300_correo_app

Tabla para almacenar los correos electrónicos que envía la aplicación

Nombre de campo	Tipo de datos	Descripción	Extras
id	Biginteger		PK, AUTOINCREMENT
tx_from	Character Varying(250)	Dirección de correo que aparecerá como remitente	NOT NULL
tx_to	Character Varying(5000)	Campo “para” del correo.	NOT NULL
tx_cc	Character Varying(5000)	Campo CC del correo	
tx_cco	Character Varying(5000)	Campo CCO del correo	
tx_subject	Character Varying(250)	Línea del asunto del correo	NOT NULL
tx_body	Character Varying(65536)	Cuerpo HTML del correo	NOT NULL
nu_priority	Smallint	El valor 1 marca el correo como importante. 0, prioridad normal	NOT NULL
tx_mandrill_id	Character Varying(64)	ID de correo asignado	
tx_reject_reason	Character Varying (250)	Motivo de rechazo (Mandrill)	
st_enviado	Smallint	0 si no ha sido enviado, 1 si ya	NOT NULL
nu_intentos	Integer	Número de intentos (incluyendo el que tuvo éxito)	NOT NULL
fh_enviado	Timestamp w/tz	Fecha de envío del correo	
created_at	Timestamp w/tz	Fecha de reación del registro	
updated_at	Timestamp w/tz	N/A	

Tabla: t001800_sesion

Tabla para manejar la caducidad de las sesiones

Nombre de campo	Tipo de datos	Descripción	Extras
id	Biginteger		PK, AUTOINCREMENT

tx_code	Character Varying(1000)	Código de autorización generado por Llave CDMX	NOT NULL
tx_token	Character Varying(1000)	Token retornado por Llave CDMX	NOT NULL
ix_token	Character Varying(1000)	Token único para identificar la sesión en la aplicación	UNIQUE, NOT NULL
id_usuario	Bigint	ID de usuario que inició sesión	NOT NULL
tx_user_agent	Character Varying(350)	Cadena de identificación del navegador cliente	NOT NULL
tx_ip	Character Varying(50)	IP del cliente	NOT NULL
tx_stamp_inicia	Character Varying(100)	Timestamp Unix con el momento de inicio de sesión	NOT NULL
tx_stamp_termina	Character Varying(100)	Timestamp Unix con el momento en que termina la sesión	NOT NULL
st_abierta	Integer	0 Si el usuario cerró la sesión, 1 si no	NOT NULL, DEFAULT 1
fh_registra	Timestamp w/tz	Fecha de creación del registro	NOT NULL

Tabla: t001801_sesion_log

Tabla para almacenar actividades de las sesiones

Nombre de campo	Tipo de datos	Descripción	Extras
id	Biginteger		PK, AUTOINCREMENT
ix_token	Character Varying(1000)	Token identificador de sesión que generó la actividad	NOT NULL
tx_mensaje	Character Varying(500)	Mensaje	NOT NULL
fh_registra	Timestamp w/tz	Fecha de creación del registro	NOT NULL

Tabla: t001803_error

Tabla para almacenar registro de errores

Nombre de campo	Tipo de datos	Descripción	Extras
id	Biginteger		PK, AUTOINCREMENT

tx_uuid	Character Varying(50)	Identificador único de error	NOT NULL, UNIQUE
tx_nivel	Character Varying(1000)	Nombre del método que generó el error	NOT NULL
tx_detalle	Character Varying(3000)	Descripción de error	NOT NULL
tx_request_uri	Character Varying(2000)	URL solicitada	NOT NULL
ix_token	Character Varying(1000)	Token identificador de sesión que generó el error	NOT NULL
un_http_response	Integer	Código http retornado, o un cero si la ejecución del script no se detuvo	NOT NULL
idUsuario	Biginteger	ID del usuario que inició sesión, o cero no había usuario logeado o si era aplicación de consola	NOT NULL
created_at	Timestamp w/tz	Fecha de reación del registro	
updated_at	Timestamp w/tz	N/A	

Modelos

Las tablas mencionadas anteriormente tienen su modelo correspondiente en la carpeta App/Models. Al igual que la estructura de las tablas, no deberían modificarse a no ser que se indique lo contrario.

Modelo	Tabla
ApiApp	apps_api
ErrorLogger	t001803_error
LlaveSesion	t001800_sesion
LogSesion	t001801_sesion_log
Permiso	permisos
User	users
Correo	t001300_correp_app

La tabla **permiso_user** no tiene modelo, ya que es manejada por el ORM Eloquent en Laravel.

AdipUtils

El arquetipo incluye algunos componentes que pueden ser reutilizados, dichos componentes se encuentran en App\AdipUtils.

ArrayList

Definición:
```
class ArrayList implements IAbstractReturnType
```

Import:
```
Use App\AdipUtils\ArrayList;
```

Propiedades

Modificador de acceso / Tipo de dato	Nombre	Descripción
protected / Array	$arreglo	Arreglo con los elementos del ArrayList

Métodos

Tipo de retorno	Definición
void	**`__construct()`** Crea una nueva instancia de ArrayList.
void	**`add(Object $item)`** Añade el objeto a los elementos del ArrayList.
void	**`addFromArray(Array $a)`** Añade los elementos de un array unidimensional indexado al ArrayList.
bool	**`isEmpty()`** Comprueba si el tamaño del ArrayList es cero.
void	**`remove($item)`** Elimina el elemento del ArrayList. $item debe ser referido por el número de índice.
void	**`leave($item)`** Establece a NULL el elemento referido. $item debe ser referido por el número de índice
int	**`count()`** Devuelve el número de elementos contenidos en el ArrayList.
Array	**`toArray()`** Devuelve los elementos del ArrayList en forma de array unidimensional indexado.
Object	**`getItem(int $i)`** Devuelve el objeto con el índice referenciado $i.

Constants

Definición:
```
final class Constants
```

Import:
```
Use App\AdipUtils\Constants;
```

Esta clase se debe usar para añadir constantes que se usen en cualquier parte de la aplicación

Constantes

Nombre	Valor
ACTIVO	int 1
NO_ACTIVO	int 0
RESULTS_PER_PAGE	int 15

Métodos estáticos

Tipo de retorno	Definición
String	**`mes(mixed $m)`** Devuelve el nombre del número de mes (1-12) dado por $m. Si $m no es un entero, será convertido cero.

ErrorLoggerService

Definición:
```
class ErrorLoggerService
```

Import:
```
Use App\AdipUtils\ErrorLoggerService as Logg;
```

Métodos estáticos

Tipo de retorno	Definición
String	**`log(String $nivel, String $desc, int $response = 0)`** Alamcena un registro de error en la tabla asociada al modelo App\Models\ErrorLogger. Se recomienda pasar siempre al parámetro $nivel, la constante __METHOD__ de PHP. Devuelve un String que corresponde al ID de la petición que generó la llamada al método.

IAbstractReturnType

Interface vacía que se puede usar para hacer posible que un método devuelva más de un tipo de objeto.

Definición:
```
interface IAbstractReturnType
```

Import:
```
Use App\AdipUtils\IAbstractReturnType;
```

LlaveCDMX

Definición:
```
final class LlaveCDMX
```

La información de esta clase se muestra con fines informativos, su código fuente no debe modificarse

Propiedades

Modificador de acceso / Tipo de dato	Nombre	Descripción
private / String	$clientID	ID del sistema LlaveCDMX
private / String	$redirectTo	URL de redireccionamiento al iniciar sesión
private / String	$clientDescription	Descripción del sistema
private / String	$authURI	URL con el servicio de autenticación
private / String	$tokenURI	URL para obtener el token de autenticación
private / String	$userURI	URL para obtener el usuario autenticado
private / String	$rolesURI	URL para obtener los roles del usuario autenticado
private / String	$secret	Código secreto del sistema
private / String	$authUser	Usuario de Basic Auth
private / String	$authPassword	Contraseña de Basic Auth

Métodos

Tipo de retorno	Definición
void	**`__construct()`** Crea una nueva instancia de LlaveCDMX.
Object	**`authenticate(String $kod)`** Se autentica en el sistema Llave CDMX usando el código proporcionado en $kod. Devuelve un objeto que contiene el token para consumir los servicios de usuario y roles

NULL / User	**getUser(String $token)** Obtiene el usuario autenticado a partir del token proporcionado en $token. Devuelve un objeto User con los datos del usuario autenticado, NULL si no hay usuario autenticado.
ArrayList	**getRoles()** Obtiene los roles del usuario autenticado a partir del token proporcionado en $token.

Network

Definición:
```
final class Network
```

Import:
```
Use App\AdipUtils\Network;
```

Métodos estáticos

Tipo de retorno	Definición
String	**getClientIP()** Devuelve pública la IP del cliente.
String	**getClientUA()** Devuelve la cadena de identificación del navegador del cliente.
String	**getClientMAC()** Intenta obtener la dirección física (MAC address) del cliente. Este método solo funciona se cumplen los siguientes requisitos: • Función shell_exec() activada • Cliente y servidor están en el mismo segmento de red.
String	**getClientOS()** Intenta obtener el nombre del sistema operativo del cliente basado en la cadena de identificación del navegador.

SimpleCURL

Definición:
```
final class SimpleCURL
```

Import:
Use App\AdipUtils\SimpleCURL;

Propiedades

Modificador de acceso / Tipo de dato	Nombre	Descripción
private / String	$url	URL de la petición

private / String	$userAgent	Identificación del navegador (user agent spoofing
private / String	$method	Método HTTP (por ahora solo se admite POST y GET
private / String	$caInfo	Ruta con los certificados para https
private / Array	$headers	Encabezados a enviar
private / Array	$cookies	Cookies a enviar
private / Recurso cURL	$cURL_executor	Recurso cURL
private / mixed	$data	Datos a enviar
private / bool	$isPrepared	TRUE si la petición está lista para ser enviada, FALSE si no
private / Array	$authBasicData	Credenciales para Basic Auth
private / bool	$useAuthBasic	TRUE si se usa Basic Auth, FALSE si no

Métodos

Tipo de retorno	Definición
void	**`__construct()`** Crea una nueva instancia de SimpleCURL.
void	**`setUrl(String $url)`** Establece la URL a la que se enviará la petición.
void	**`setUserAgent(String $ua)`** Establece la cadena de identificación del navegador a enviar.
void	**`isGet()`** Define el método HTTP a enviar como GET.
void	**`useAuthBasic(String $username, String $password)`** Define las credenciales a usar cuando se emplea Basic Auth en la petición.
void	**`setCaInfo(String $cainf)`** Establece la ruta del certificado .crt en peticiones con https.
void	**`AddHeader(Array $header)`** Agrega un encabezado a la petición. El array debe ser asociativo con los índices “name” y “value”.
void	**`AddCookie(Array $cookie)`** Agrega una cookie a la petición. El array debe ser asociativo con los índices “name” y “value”.
void	**`setData(mixed $strData)`** Establece el cuerpo de la petición a enviar. Aunque se espera un tipo de dato mixed, el cuerpo debe ser un String.
void	**`prepare()`** Prepara la petición para ser enviada.
bool	**`isPrepared()`** Devuelve TRUE si se ha ejecutado el método prepare(), false si no.
mixed	**`execute()`** ejecuta la petición y retorna el resultado (típicamente String).
void	**`__destruct()`**

	Cierra la conexión cURL y destruye la instancia de SimpleCURL. Este es un método mágico, no debe llamarse manualmente.
String	**__toString()** Devuelve la representación del objeto como una cadena.

Métodos estáticos

Tipo de retorno	Definición
bool	**isRunnable()** Devuelve TRUE si SimpleCURL se puede ejecutar en el sistema, FALSE si no.

Envío de correos

En PHP se considera mala práctica de programación realizar envío de correos *"al vuelo"*, es decir, realizar el envío dentro de la programación que implica lógica de negocio. Por ejemplo:

```
$solicitud->actualizar();
$to = $solicitud->usuario()->email;
Mail::to($to)->send('solicitud-validada'); // Esta línea en Laravel envía el correo
return view('solicitud.actualizada');
```

En el ejemplo anterior si el envío de correo demora 25000 milisegundos (25 segundos), ese tiempo se verá reflejado en el tiempo que tarda la página en cargar, lo cual disminuye la experiencia de usuario.

Para solventar este comportamiento el arquetipo incluye un servicio de envío de correos, usando dicho servicio, se elimina el tiempo de respuesta de envío de correo. (en el ejemplo anterior, los 25 segundos).

El servicio de envío de correos consta de 3 partes: el modelo Correo, el servicio MandrilMail y un comando Artisan que envía los correos.

Para utilizarlo, solo hay que crear un registro en la tabla asociada al modelo Correo, de la siguiente forma (siguiendo con el ejemplo anterior):

```
$solicitud->actualizar();
$to = $solicitud->usuario()->email;

$html = view('solicitud.validada')->render();
$correo = [
    'tx_from' => env('MAIL_FROM_ADDRESS', 'no-reply@cdmx.gob.mx')
    ,'tx_to' => $to
    ,'tx_subject' => 'Asunto del correo'
    ,'tx_body' => $html
    ,'nu_priority' => 0
];
Correo::create($correo);
return view('solicitud.actualizada');
```

Se debe añadir el import de la clase Correo al comienzo del archivo `use App\Models\Correo;`.

Posteriormente se debe programar una tarea en el sistema operativo que ejecute cada 5 minutos la siguiente línea de comando:

```
php artisan llave:enviar-correos
```

Para programar la tarea hay que tener en cuenta las siguientes consideraciones:

- En Windows se puede usar el Programador de Tareas para crearla, en Linux la forma más fácil es con un cronjob.
- La tarea debe iniciar en la carpeta raíz del proyecto.
- En Linux puede que sea necesario meter la instrucción artisan en un archivo .sh similar a lo siguiente para que sea ejecutado:

```
#!/bin/bash
/usr/local/php/bin/php /var/www/html/virtual/myApp/artisan llave:enviar-correos
```

La siguiente es información técnica de los componentes del servicio, su código fuente no debería ser midificado a no ser que se indique lo contrario.

Correo

Definición:
```
class correo extends Model
```

Import:
```
use App\Models\Correo;
```

Constantes

Nombre	Valor
ENVIADO	int 1
NO_ENVIADO	int 0

Propiedades

Modificador de acceso / Tipo de dato	Nombre	Descripción
protected / String	$table	Tabla asociada al modelo (t001300_correo_app).
protected / Array	$fillable	Campos rellenables en masa.
public / String	$tx_from	Dirección que aparecerá como remitente
public / String	$tx_to	Campo *Para* del correo electrónico
public / String	$tx_cc	Campo *CC* del correo electrónico
public / String	$tx_cco	Campo *CCO* del correo electrónico
public / String	$tx_subject	Línea de asunto

public / String	$tx_body	Cuerpo del correo en HTML
public / String	$nu_priority	1 importante, 0 importancia normal
public / String	$tx_mandrill_id	ID de correo asignado por Mandrill
public / String	$tx_reject_reason	Motivo de rechazo (Mandrill)
public / int	$st_enviado	1 correo enviado, 0 no enviado
public / int	$nu_intentos	Intentos de envío, incluyendo el exitoso
public / Datetime	$fh_enviado	Fecha de envío
public / Datetime	$created_at	Fecha de creación del registro
public / Datetime	$updated_at	Fecha de modificación del registro

MandrillMail

Definición:
`final class MandrillMail`

Propiedades

Modificador de acceso / Tipo de dato	Nombre	Descripción
private / Recurso cURL	$curl	Recurso Curl para ejecutar la petición al API de Mandrill.

Métodos

Tipo de retorno	Definición
void	**`__construct()`** Crea una nueva instancia de MandrillMail.
Array	**`sendMail(Correo $correo)`** Intenta realizar el envío del correo proporcionado. Devuelve un Array con información del resultado del envío.

Arquitectura frontend

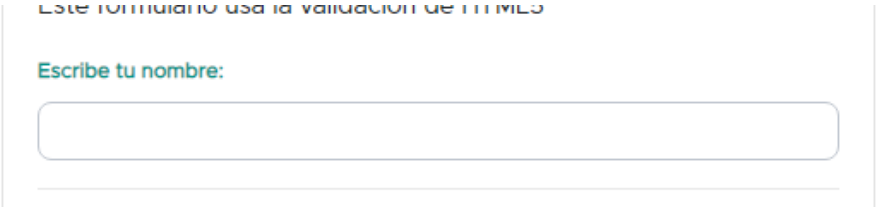
El arquetipo incluye diversos componentes frontend que pueden ser reutilizados.

Controles CDMX

Se puede aplicar estilo a los controles de los formularios para que tengan la combinación de colores CDMX. Basta con agregar las clases `form-label-cdmx` y `form-control-cdmx`. Ejemplo:

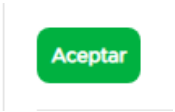
```
<div class="form-group">
<label for="txPrueba" class="control-label form-label-cdmx">Escribe tu nombre:</label>

<input type="text" name="txPrueba" id="txPrueba" class="form-control form-control-cdmx"
required>
</div>
```



Para aplicar el estilo CDMX a los botones de los formularios, usar la clase `btn-cdmx`. Ejemplo:

```
<button class="btn btn-cdmx" type="button" >Aceptar</button>
```



Validación de formularios

El arquetipo incluye una validación de formularios, para utilizarla solo se debe agregar la clase `needs-validation` al formulario, así como el atributo `novalidate`, además, el botón de envío (submit) deberá tener el ID `btn-send`. Ejemplo:

```
<form name="test2" id="test2" class="form-search needs-validation" novalidate>
   <p>Este formulario usa la validación de la App</p>
   <div class="form-group">
      <label for="txPrueba2" class="control-label form-label-cdmx">Escribe tu nombre:</label>
      <input type="text" name="txPrueba2" id="txPrueba2" class="form-control form-control-cdmx"
required>
   </div>
   <hr>
   <button type="submit" class="btn btn-cdmx btn-shadow" id="btn-send">Enviar</button>
</form>
```

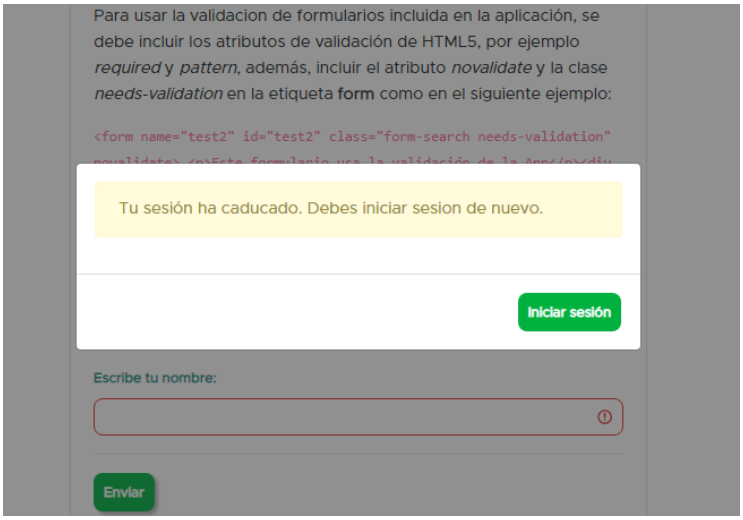
Validación de formularios del arquetipo

La validación se llevará a cabo con los atributos de validación que emplea HTML5, por ejemplo: required, min, max, pattern, etc.

Para usar la validación de HTML5 en lugar de la del arquetipo, quitar el atributo `novalidate` y la clase `needs-validation`.

Validación de sesión

Se incluye un validador de sesión que verifica cada 3 minutos que la sesión siga activa. Si la sesión ha caducado se muestra un modal con un botón para volver a iniciar sesión.



Se puede desactivar esta función editando el archivo app/resources/js/app.js y comentar la línea 79, para que quede:

```
// timercito = setInterval(laSesion, 5*60*1000); // minutostimer * 60 * 1000
```

Después, compilar los assets con npm.

Exponer API

Algunas aplicaciones necesitarán exponer servicios a otras, para lo cual deberán generar su propia API REST. El arquetipo incluye todo lo necesario para poder comenzar a crear la API de la aplicación.

Los pasos para crear un API son los siguientes:

Dar de alta la aplicación

Se debe registrar la aplicación en la tabla `apps_api`, el campo `api_token`, debe ser de exactamente 80 caracteres.
En el seeder AppApiSeeder se incluye un ejemplo de cómo dar de alta la aplicación.

```
public function run()
{
    ApiApp::create([
        'nb_aplicacion' => 'App1 WS'
        ,'tx_descripcion_app' => 'Aplicacionde prueba'
```

```
            ,'api_token' => Str::random(80)
            ,'st_activo' => 1
            ,'created_at' => date('Y-m-d H:i:s')
            ,'updated_at' => date('Y-m-d H:i:s')
        ]);
    }
```

Construir la API

Las rutas de la API se definen en el archivo `routes/api.php`. Las rutas fijadas en dicho archivo asumen que se está dentro de `public/api`, de tal forma que si una ruta refiere a '`/user`' la ruta completa será `public/api/user` y no `public/user`.

Las rutas deben apuntar a la clase ApiController, que se encuentra en App\Http\Controllers\ApiControllers.

Autenticación del cliente

Para autenticarse, el cliente debe enviar su api_token usando uno de los siguientes métodos:

QueryString (no recomendado)

Se envia api_token como parte de la URL a consumir, por ejemplo:

http://misistema.cdmx.gob.mx/public/api/mi-metodo?api_token=XXXXXXXXXXXXXXXXXX

Encabezado Auth Bearer

Se envía un encabezado Authorization => Bearer XXXXXXXXXXXXXXXXXX

Ejemplo de una petición GET podría ser:

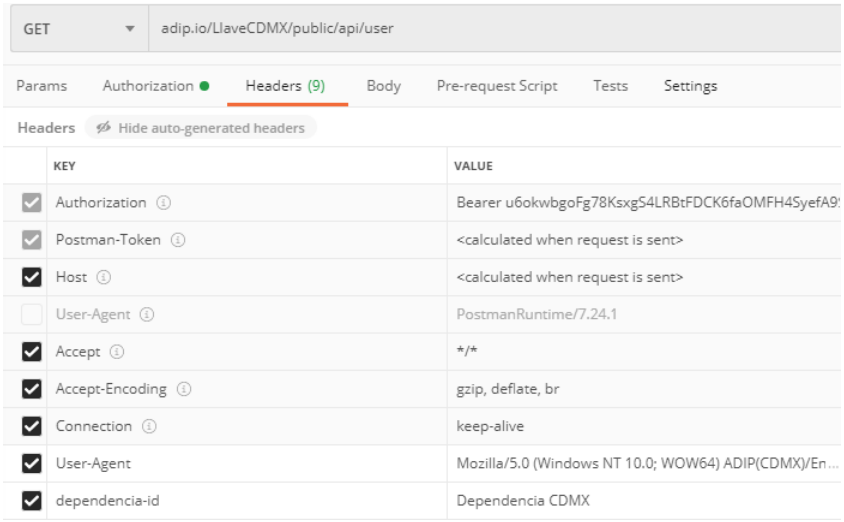
```
GET /LlaveCDMX/public/api/user HTTP/1.1
Host: adip.io
Authorization: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Encabezados adicionales

Además de la autenticación, el cliente sobre escribir el encabezado **`User-Agent`**, enviando la cadena que el desarrollador haya especificado en el archivo `App\Http\Controllers\ApiControllers\ApiController` (valor de la constante `USER_AGENT`).

También se debe enviar un encabezado dependencia-id que contenga cualquier valor.

Ejemplo de petición generada con Postman:

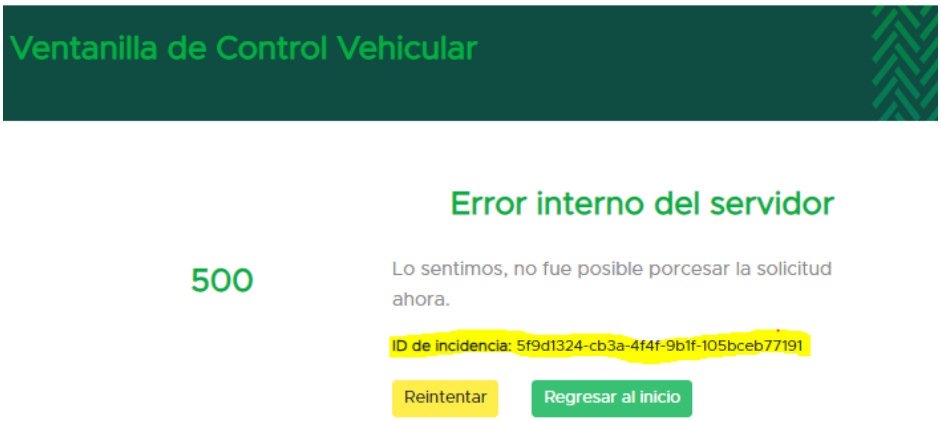


La misma petición en HTTP RAW luce así (en verde, los encabezados que se deben enviar):

```
GET /LlaveCDMX/public/api/user HTTP/1.1
Host: misistema.cdmx.gob.mx
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) ADIP(CDMX)/Endpoint Client
dependencia-id: Dependencia CDMX
Authorization: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Trazabilidad de errores

Los errores se pueden rastrear usando el ID de petición, este es mostrado cuando se muestra una página de error (ID de incidencia)



Para ver los detalles del error se debe buscar ese ID en la tabla t001803_error (ver siguiente imagen).

```
SELECT id, tx_uuid, tx_nivel, tx_detalle, tx_request_uri, tx_session_token, nu_http_response, "idUsuario
    FROM public.t001803_error where tx_uuid = '5f9d1324-cb3a-4f4f-9b1f-105bceb77191';
```

ta Output | Explain | Messages | Notifications

tx_nivel character varying (1000)	tx_detalle character varying (3000)	tx_request_uri character varying (2000)	tx_session_toke character varyir
App\AdipUtils\FinanzasService::getMarcas	SOAP-ERROR: Parsing WSDL: Couldn't load from 'http://10.1....	tramite/alta/editar/v/14	181346459d7c

Rutas reservadas

Las siguientes rutas están reservadas para el arquetipo o funciones futuras del mismo. No deben emplearse en la aplicación desarrollada:

```
/public/api/user/
/public/cmd/*
```

Sugerir cambios al arquetipo (Pull request)

Las sugerencias de cambio se deben realizar en una rama nueva del repositorio original con la siguiente nomenclatura de nombre: **YYYY-MM-DD-Nombre-desarrollador**, por ejemplo **2020-04-17-Guillermo Hernandez**.

Los commits a dicha rama deberán ser descriptivos, de acuerdo con lo siguiente:

> **Tipo: <alcance> título**
> **Descripción detallada del commit**
> **Puede ser de muchas líneas**

En donde:

- **Tipo:** Indica qué cambio se está realizando al código, debe ser uno de los siguientes:
  - **Feat:** Nueva función que impacte al usuario final, scripts de cronjobs o similares no entran aquí
  - **Fix:** Solución de un error documentado. Soluciones a errores de scripts de cronjobs o similares no entran aquí. Si se cuenta con un bugtracker como JIRA o Mantis se puede poner el ID del reporte.
  - **Docs:** Cambios en la documentación
  - **Style:** Cambios en el formato del código, estos cambios no afectan las funcionalidades ni la lógica ya programada. Un ejemplo de cuando usar este tipo de commit es, por ejemplo, si parte del código estaba obfuscado y la obfuscación fue removida.
  - **Refactor:** Cambios en la semántica del código que no afectan la lógica, por ejemplo, renombrar variables o modularizar *código espagueti*.
  - **Chore:** Cualquier tipo de cambio en tareas programadas, scripts de cronjobs y similares.
- **Alcance:** Es opcional y se refiere a la parte del sistema a la que se le está realizando cambios. Colocar el nombre del paquete o función es una buna práctica

Cada línea no debe exceder de 80 caracteres.
Un ejemplo de cómo realizar los commits sería:

```
Feat: Envío automatizado de correos
Se Añaden los componentes para realizar el envío de correos
en segundo plano.
Se incluyen: migración para crear la tabla, DAOs y Servicios
```

Después de realizados los commits, solicitar un Pull Request (o Merge Request). Si este es aceptado, verás los cambios en el historial de commit del repositorio.

Actualizaciones de seguridad

Si encuentras un fallo de seguridad dentro del arquetipo o uno de sus componentes infórmalo por correo electrónico a underdog1987@yandex.ru, preferentemente indicando:

- Tipo de fallo (SQL Injection, XSS, CSRF, Remote Command Excecution, etc)
- Pasos para explotarlo
- Vector de ataque (ingeniería social, exploit, negación de servicio, etc)

Los fallos de seguridad serán tratados con prioridad alta.