PROYECTO FP – CICLO DE DESARROLLO DE APLICACIONES WEB

AUTOR - ÁNGELES ROSALES FERNÁNDEZ

2019

Contenido

INTRODUCCIÓN

Este proyecto '*SELF*' pretende ser una solución basada en la web, diseñada para empresarios autónomos que buscan un sistema de facturación y gestión de impuestos.

'*SELF*' elimina el error humano en el proceso de creación de facturas (facturas incompletas o con cálculos erróneos, numeración de facturas no correlacionada, etc.). El sistema automatiza el cálculo de impuestos en factura según a quién se emita y qué se vende. También automatiza la anulación de facturas.

El sistema '*SELF*' ofrece además informes y gráficos para el análisis de datos y comparativas de periodos.

DESCRIPCIÓN

'*SELF*' es una solución web. Este formato resulta muy accesible puesto que el usuario solo necesita de un navegador con conexión al servidor en el que se aloje el proyecto; no necesita instalación de software.

En el servidor se alojan todos los recursos necesarios para ofrecer el servicio: aplicación web, base de datos, *backups*, etc.

El éxito de la aplicación depende de la correcta *parametrización* del sistema; esto es, fichas de clientes, datos del emisor de factura, actividad para a la que se asocia el emisor de facturas y qué productos vende o que servicios presta. Con la correcta combinación de valores de estas variables, se asegura la creación de facturas correctas en base a los requisitos legales y fiscales.

Por simplicidad, '*SELF*' está preparado únicamente para trabajar con emisores de factura bajo la figura de trabajador afiliados al RETA; profesionales y empresarios con un volumen de facturación asequible cuyas necesidades sean: facturar servicios a cliente y presentar liquidación de impuestos trimestrales (IVA, IRPF).

PRESUPUESTO

Para la realización de este proyecto, dado que se trata de una aplicación web, no se necesitan grandes inversiones de dinero. El riesgo de la operación es muy bajo.

El coste inicial, de desarrollo en local de la aplicación es, por el contrario, realmente alto puesto que se parte de un desconocimiento muy grande, tanto de las herramientas de trabajo, como de la metodología y no existe factor experiencia. Estas horas de trabajo inicial no se tienen en consideración para el análisis de viabilidad que se presenta a continuación.

Para los próximos 3 años, se podría realizar la siguiente estimación de costes, bajo un escenario optimista pero cauto.

PRESUPUESTO COSTES

Costes fijos		**€ Anuales**
Dominio web	10€/año	10,00 €
Servidor de despliegue	10€/mes	120,00 €
Servidor correo	0,50€/mes	6,00 €
		136,00 €

Costes Variables		**€ Anuales**
Desarrollos de cliente	15€/hora - 10h/mes	1.800,00 €
Soporte	10€/h - 10h/mes	1.200,00 €
		3.000,00 €

COSTES TOTALES / AÑO	**3.136,00 €**

Nótese, que los costes por uso del servidor, son variables en función del tráfico de la aplicación.

En este escenario, no se valoran otros costes que podrían surgir, como aquellos relacionados con la protección de los datos de los usuarios, etc.

En cuanto a las previsiones de ingresos que se podrían obtener de este proyecto; es necesario establecer qué actividades se ofrecen y qué tarifas conllevan.

La principal oferta de este proyecto, podría consistir en el uso de la aplicación estandar. Esto es, sin ninguna configuración específica que solicite el usuario. Sí se incluirian servicios de soporte en cuanto a solución de errores de la aplicación.

Una segunda oferta, estaría compuesta por el pack anterior, con un servicio ya de apoyo y asesoramiento al usuario, donde se incluiría cierta parametrización específica del usuario, etc.

Finalmente, una oferta que incluya los dos packs anteriores junto con un asesoramiento no solo a nivel técnico sino también administrativo y de gestión. Apoyo a la presentación de impuestos, etc.

Con estas propuestas, pueden formarse la siguiente tabla de precios:

PRODUCTOS OFRECIDOS

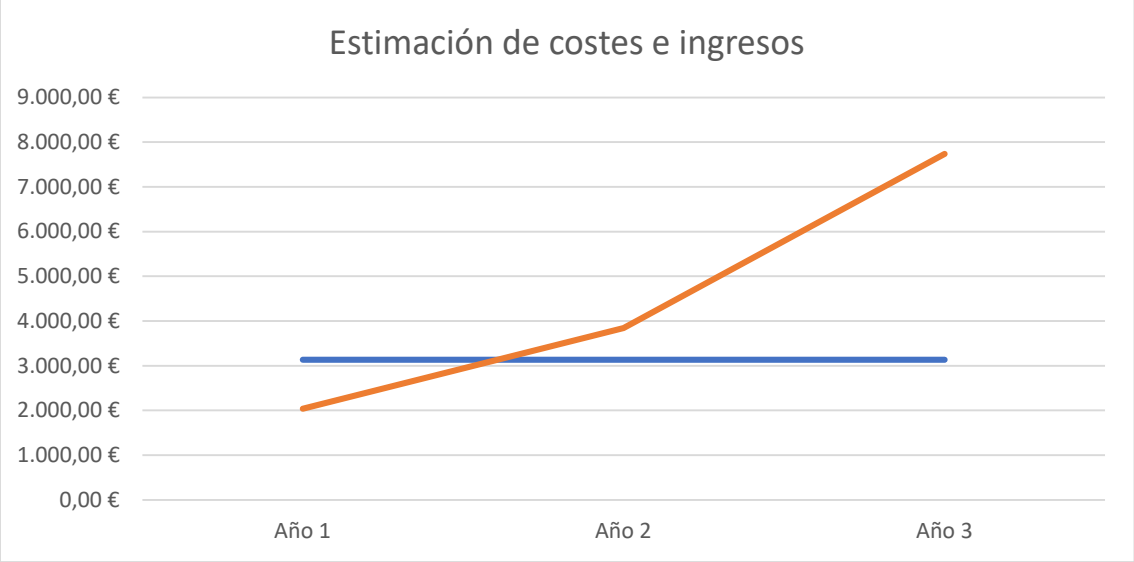
Productos	**€/mes**
Pack 1: aplicación	10€/mes
Pack 2: parametrización	25€/mes
Pack 3: parametrización y asesoría	50€/mes

Bajo un escenario cauto, se podrían estimar unas previsiones de ingresos para los próximos 3 años como se muestra a continuación:

PREVISIÓN INGRESOS

Productos	Año 1		Año 2		Año 3	
	Usuarios	€ Año	Usuarios	€ Año	Usuarios	€ Año
Pack 1	2	240,00 €	2	240,00 €	2	240,00 €
Pack 2	2	600,00 €	4	1.200,00 €	5	1.500,00 €
Pack 3	2	1.200,00 €	4	2.400,00 €	10	6.000,00 €
		2.040,00 €		**3.840,00 €**		**7.740,00 €**

Con esto, podríamos tener la siguiente situación a corto y medio plazo:



Se observa que la evolución de ingresos no es negativa, y el punto en el que se espera comenzar a obtener beneficio es en el segundo año de actividad.

No resulta una mala perspectiva, teniendo en cuenta que no se valora realizar ninguna acción publicitaria ni de promoción.

ANÁLISIS

La principal funcionalidad que cubre la aplicación, es la emisión de facturas por parte de un usuario con capacidad de emitir facturas legalmente.

FACTURAS

Una factura contiene una lista de todos los productos o servicios solicitados por el cliente con información detallada sobre cada uno de los artículos a modo de lista, así como la cantidad, descripción, descuentos, cargos adicionales, impuestos, totales, entre otros. Comunica al cliente la cantidad que adeuda pagar en concepto de los bienes y/o servicios comprados.

Elementos de una factura:

1. Información completa del emisor: Entre la información se incluye el nombre comercial, la dirección comercial, NIF y la información de contacto (número de teléfono, correo electrónico o sitio web). La información completa del emisor se debe colocar en el encabezado.

2. Información completa del cliente: La información completa del cliente también es un elemento importante de una factura. Esta información es necesaria para fines de pago, así como para posibles procesos legales que puedan surgir debido a conflictos entre el comprador y el vendedor. La información del cliente debe incluir lo siguiente: nombre del cliente y detalles de contacto (número de teléfono, *email*).

3. Fecha de la factura: La fecha de la factura también es parte integral de una factura. La fecha de la factura corresponde a la fecha de emisión de la factura, que puede ser útil siempre que haya descuentos o garantías en transacciones futuras. Además, la fecha de la factura también puede ser útil en caso de un posible caso legal en el futuro.

4. Número y serie de factura: Cada factura debe tener un número de factura distinto, único y consecutivo. Las facturas, además de un número consecutivo, pueden pertenecer a una u otra serie en función de las necesidades del usuario de la aplicación.

5. Bienes o servicios entregados: Una factura también enumera todos los bienes o servicios que fueron comprados por el cliente. La lista de cada producto no tiene que ser detallada, pero debe incluir esta información: nombre del producto, precio y cantidad.

6. Subtotal y total: El subtotal corresponde al total de los bienes/servicios enumerados en la factura; precio por cantidad. El subtotal es la suma del precio de todos los bienes o servicios. Mientras tanto, el total general es la cantidad final que debe pagar el comprador.

7. Impuestos y deducciones adicionales: Los impuestos, cargos y tarifas relacionados con la transacción que no sean el monto de los artículos en la lista que debe pagar el comprador cubren los pagos adicionales en una factura. Por otro lado, las deducciones son descuentos y otras promociones que son utilizadas por el cliente dentro de un período específico.

8. Método de pago: El método de pago también debe especificarse en la factura. Aparte del efectivo, los clientes también pueden usar sus tarjetas de débito o crédito para pagar una compra, una plataforma como *PayPal* o transferencia bancaria.

9. Observaciones: La sección de notas en una factura también se refiere a un cuadro de mensaje donde generalmente el emisor de factura escribe notas adicionales. La sección de notas no es obligatoria, pero se puede agregar dependiendo de su propósito.

Si la factura contiene operaciones exentas de IVA, debes indicar la norma que determina esa exención. Esta norma se encuentra en la Ley del Impuesto o en la Directiva europea 2006/112/CE, de 28 de noviembre.

Con las facturas, se pueden realizar las siguientes acciones:

1. Emitir factura: Se redacta la factura original con los datos anteriormente mencionados.

2. Rectificar factura: Las facturas emitidas, no deben ser eliminadas. Sin embargo, puede interesar anular la operación para rehacerla con distintos valores. Para ello, se emite una factura rectificativa, exactamente igual que la original, pero en negativo; con esto, se pretende netear el valor y puede emitirse una nueva factura original adecuada. Por imperativo legal, las facturas rectificativas deben generarse bajo una serie factura diferenciada.

3. Presentar factura: Las facturas, se incluyen en las liquidaciones trimestrales de impuestos. Cuando esto sucede, la factura se marca como "presentada", con esto se facilita el control de las actividades.

4. Pagar factura: Una factura, recoge una deuda del cliente con el empresario. Una vez se cobra la factura, puede marcarse como "pagada", este es otro elemento de control para la gestión del negocio.

USUARIO DE LA APLICACIÓN

El usuario de la aplicación, es quien utiliza el sistema 'SELF' para facturar sus actividades. Debe registrarse en el sistema y posteriormente, iniciará sesión. Sus datos son el primer paso para poner el sistema en funcionamiento.

Nombre: un alias para identificarse.

Email: para comunicarse con la aplicación.

Contraseña: es necesaria para autenticar al usuario.

EMISOR FACTURA

La figura de usuario de la aplicación no debe confundirse con la de emisor factura. El usuario registrado puede tener unos datos (nombre, email, NIF, etc.) y el emisor de facturas puede tener otros diferentes.

Del emisor de factura interesa conocer:

1. Identificación fiscal: Todas las personas físicas o jurídicas, han de tener un número de identificación fiscal que tendrán que acreditar en sus relaciones de naturaleza o con trascendencia tributaria. Esta identificación se materializa en el Número de Identificación fiscal (NIF). En el caso de que el emisor de facturas realice operaciones intracomunitarias debe disponer de un número de identificación a efectos de IVA intracomunitario (NIVA), como prueba de que figura inscrito en el Censo de Operadores Intracomunitarios (VIES – *Vat Information Exchange System*).

2. Nombre y dirección fiscal: identificación a efectos legales y de notificaciones.

3. Actividad: El emisor de factura está dado de alta en una o varias actividades económicas y profesionales. La actividad a facturar es relevante, puesto que no todas ellas tienes los mismos requisitos fiscales.

CLIENTE

El rol que juega es como receptor de facturas por parte del usuario. Los datos del cliente son fundamentales para el correcto funcionamiento de la aplicación.

Se necesitan conocer los mismos datos que para el emisor de facturas. Además, el cliente debe realizar el pago de la compra de bienes o servicios. Para ello, se necesitan los siguientes datos:

Método de pago: Este campo cubre las opciones que se ofrecen para realizar el cobro al cliente: transferencia, PayPal, contado.

Plazo para el pago: Recoge los vencimientos a los que se puede diferir el cobro de facturas: 0, 15, 30 y 60 días.

Tipo de cliente: persona física o persona jurídica. Los requisitos fiscales de una factura varían en función de a quién se emite factura. Ejemplo: personas físicas nacionales no llevan retención, pero sí las personas jurídicas nacionales.

Ámbito del cliente: nacional, intracomunitario, extracomunitario. Al igual que la característica anterior, esta también afecta a la fiscalidad de la operación. Por ejemplo, la emisión de una factura a un canario no está sujeta a IVA, mientras que la emisión a un nacional sí.

Los campos de tipo de cliente y ámbito del cliente son clave para la determinación de la fiscalidad de la factura. No es lo mismo emitir una factura a un cliente nacional que a uno internacional, ni es lo mismo emitir a una persona física, pagador último del IVA, que a una persona jurídica con capacidad para repercutir nuevamente IVA.

PRODUCTO

La factura debe indicar qué productos/servicios son adquiridos Para conseguir esta información en factura, debemos recabar la siguiente información sobre los productos:

Nombre: es la denominación que identifica al bien o servicio.

Descripción: texto informativo del artículo.

Precio: cantidad de dinero que permite valorar un artículo.

Unidad: el precio debe ir en referencia a una unidad de medida: hora, unidad, proyecto, etc.

Actividad CNAE: Los productos, están relacionados con actividades profesionales. De esta manera, los productos adoptan como propio el régimen de impuestos al que está asociado esa actividad.

De **la combinación de impuesto producto + ámbito del cliente + tipo de entidad del cliente,** resulta el impuesto liquidable a aplicar en la factura, igual sucede para las retenciones. Estos son los dos impuestos para los que la aplicación está preparada.

ARQUITECTURA

La creación del proyecto se hace bajo un **entorno de desarrollo**. Bajo este entorno se realizan mejoras, se añaden características y se corrigen errores sin afectar a datos reales.

En este caso se utiliza *Apache* como servicio web local, *MySQL* como motor de base de datos con *PhpMyAdmin* para gestionarlas.

Partimos de que disponemos del paquete *WAMP* instalado en un sistema operativo Windows 10.

La elección de trabajo con esta distribución, se basa en que resulta cómodo y sencillo. Es la herramienta con la que he trabajado los dos años del Ciclo FP.

El código está escrito en dos lenguajes; por un lado, *PHP* para programación en *backend* y, por otro lado, *Javascript* para cubrir el código necesario en *frontend*.

Dentro de estos dos lenguajes, el código utiliza dos frameworks diseñados para trabajar en conjunto: *Laravel* para crear aplicaciones y servicios web e interactuar con las bases de datos y *VueJS* para crear interfaces de usuario y consumir la información que *Laravel* ofrece desde el servidor.

Toda la tecnología anterior bajo *Windows 10* como sistema operativo. La elección del sistema operativo se basó en la comodidad que supone trabajar con un sistema operativo que conozco y que tengo ya instalado en mi ordenador de trabajo.

Este escenario se conoce como local ya que la aplicación no sale de la red privada.

Una vez el proyecto está prácticamente terminado, se abre un nuevo entorno conocido como ***staging***. Este entorno lo utiliza el cliente para probar la aplicación e informar sobre los errores que ha encontrado o las características que le faltan a la aplicación.

Finalmente, se crea el **entorno de producción**, este es el entorno en el que se ejecuta la aplicación que utilizan los usuarios finales.

'SELF' se aloja en un servidor virtual privado proveído por *DigitalOcean*. *DigitalOcean* maneja el concepto de *droplet* para designar a cada uno de los servidores virtuales privados a los que se accede por *SSH* (vía consola), los cuales ofrecen en alquiler. Son privados porque *DigitalOcean* no interviene en nada en su instalación y manejo, limitándose a ofrecer imágenes de los principales sistemas operativos junto con sus repositorios de manera local, lo cual ahorra tiempo y transporte de datos a la larga gracias a los repositorios locales.

El servidor de Digital Ocean elegido se compone de un droplet preconfigurado con una instalación de Ubuntu y XAMP. Es un droplet standar, con una porción de vCPU y una memoria. Esta opción es más que suficiente para trabajar con una aplicación de poco tráfico y con fines no comerciales.

Para completar el proyecto, se dispone de un dominio miproyecto/angeles-rosales.com (*NameCheap*) para identificar al sitio web en Internet.

Con el fin de ofrecer comunicación vía email en la aplicación, se cuenta con un servicio de *mailing* desde *Mailgun*.

FRAMEWORKS

Laravel

Laravel es un *framework* de código abierto para el desarrollo de aplicaciones web en *PHP 7.2*.

Laravel facilita el desarrollo simplificando el trabajo con tareas comunes como la autenticación, el enrutamiendo, gestión sesiones, el almacenamiento en caché, etc. Algunas de las principales características y ventajas de Laravel son:

- Esta diseñado para desarrollar bajo el **patrón MVC** (modelo - vista - controlador), centrándose en la correcta separación y modularización del código. Facilita el trabajo en equipo, así como la claridad, el mantenimiento y la reutilización del código.
- Integra un **sistema ORM de mapeado de datos relacional llamado *Eloquent,*** aunque también permite la construcción de consultas directas a base de datos mediante su *Query Builder*.
- Permite la gestión de bases de datos y la manipulación de tablas desde código, manteniendo un control de versiones de las mismas mediante su sistema de ***Migraciones***.
- Utiliza un sistema de **plantillas para las vistas llamado Blade**, el cual hace uso de la cache para darle mayor velocidad. Blade facilita la creación de vistas mediante el uso de *layouts*, herencia y secciones.
- Facilita la extensión de funcionalidad mediante paquetes o librerías externas. De esta forma es muy sencillo añadir paquetes que nos faciliten el desarrollo de una aplicación y nos ahorren mucho tiempo de programación.
- Incorpora un **intérprete de línea de comandos llamado *Artisan*** que ayuda con un montón de tareas rutinarias como la creación de distintos componentes de código, trabajo con la base de datos y migraciones, gestión de rutas, cachés, colas, tareas programadas, etc.

Laravel no obliga a utilizar preprocesadores concretos para JavaScript o CSS, pero provee de una estructura básica usando Vue y Bootstrap que resulta muy útil. Laravel, por defecto, usa NPM para instalar ambos paquetes para el frontend de la aplicación.

Laravel Mix proporciona una API fluida para definir pasos de compilación de Webpack para las aplicaciones de Laravel usando múltiples preprocesadores de CSS y JavaScript. A través de encadenamiento de cadenas simples, se pueden definir pipelines de assets.

VueJS

Vue es un framework progresivo para construir interfaces de usuario. Vue puede integrarse fácilmente en los proyectos Laravel.

En este proyecto, se usa NPM para instalar dependencias y Webpack sobre Laravel para compilar el código JavaScript.

Instalando los Frameworks

Instalación de *Composer*

Composer es un gestor de dependencias para PHP. Esto quiere decir que permite descargar de sus repositorios todas las librerías y las dependencias con las versiones requeridas que el proyecto necesite.

En Windows hay que descargar el instalador de la Web de Composer y ejecutarlo para instalar la aplicación.

Instalar Laravel mediante Composer

En la carpeta raíz del servidor web (/XAMPP/htdocs) se ejecuta el siguiente comando:

```
composer create-project laravel/laravel miweb --prefer-dist
```

Debido a que Laravel requiere la extensión Mcrypt para su utilización, hay que instalarlo.

La configuración se encuentra almacenada en ficheros dentro de la carpeta "config". Para empezar, no es necesario modificar ninguna configuración. La mayoría de las opciones de configuración tienen su valor puesto directamente en el fichero, pero hay algunas que se cargan a través de variables de entorno utilizando el sistema *DotEnv*.

Estas variables tienen que estar definidas en el fichero. env de la raíz de la aplicación, y mediante el método env('NOMBRE', 'VALOR-POR-DEFECTO') se cargan y se asignan a una opción de configuración. Esto permite separar configuración según el entorno o el usuario que lo utilice simplemente cambiando el fichero. env. Así por ejemplo se puede tener un fichero. env para el entorno de desarrollo local, otro para producción, otro para pruebas, etc.

El único valor que hay que asegurar de que esté correctamente configurado, es la key de la aplicación. Esta clave es una cadena de 32 caracteres que se utiliza para codificar los datos. En caso de no establecerla, la aplicación no será segura. Para crearla simplemente debe ejecutarse el siguiente comando en la carpeta raíz de nuestra aplicación:

```
php artisan key:generate
```

Además, hay que establecer los permisos de algunas carpetas especiales. En general no es necesario añadir permisos de escritura para los archivos de nuestra aplicación, solo habría que hacerlo para las carpetas storage y bootstrap/cache, que es donde Laravel almacena los logs, sesiones, chachés, etc. En Windows no será necesario dar permisos de escritura a estas carpetas.

Artisan

Entre las herramientas que Laravel proporciona para el desarrollo de aplicaciones se encuentra Artisan, la interfaz de línea de comandos (CLI por sus siglas en inglés de *Command-line interface*), la cual es un medio para la interacción con la aplicación donde los usuarios desarrolladores dan instrucciones en forma de línea de texto simple o línea de comando. Artisan está basado en el componente Console de Symfony y ofrece un conjunto de comandos que nos pueden ayudar a realizar diferentes tareas durante el desarrollo.

Para conocer el listado completo de los comandos disponibles:

```
Php artisan list
```

Para levantar el servidor de una aplicación, es decir, para hacer correr una aplicación con el servidor que viene incluido en Laravel:

```
php artisan serve
```

Instalar Node y NPM

Antes de ejecutar Mix, Node.js y NPM deben estar instalados.

```
node -v
```

NPM es un gestor de dependencias de JavaScript.

```
npm install
```

Dado que JavaScript es un lenguaje compilado, cada vez que se ejecute una modificación en código frontend, debe compilarse nuevamente. Para evitar estar pendientes de la ejecución de esta compilación, se ejecuta desde consola:Com

```
npm run watch
```

Estructura de carpetas

```
    Directorio: C:\xampp\htdocs\miproyectoFP


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/12/2019     23:03                app
d-----        24/11/2019     23:58                bootstrap
d-----        24/11/2019     23:58                config
d-----        24/11/2019     23:58                database
d-----        25/11/2019      0:06                node_modules
d-----        24/11/2019     23:58                public
d-----        24/11/2019     23:58                resources
d-----        24/11/2019     23:58                routes
d-----        24/11/2019     23:58                storage
d-----        24/11/2019     23:58                tests
d-----        25/11/2019      0:03                vendor
-a----        24/11/2019     23:58            235 .editorconfig
-a----        25/11/2019      0:04            831 .env
-a----        24/11/2019     23:58            116 .gitattributes
-a----        24/11/2019     23:58            175 .gitignore
-a----        24/11/2019     23:58            258 .styleci.yml
-a----        24/11/2019     23:58           1739 artisan
-a----        24/11/2019     23:58           1627 composer.json
-a----        24/11/2019     23:58         191158 composer.lock
-a----        24/11/2019     23:58         463171 package-lock.json
-a----        24/11/2019     23:58           1704 package.json
-a----        24/11/2019     23:58           1710 phpunit.xml
-a----        24/11/2019     23:58            941 readme.md
-a----        24/11/2019     23:58           5661 readme2.md
-a----        24/11/2019     23:58            584 server.php
-a----        02/12/2019     18:39            841 webpack.mix.js
```

/app: Se guardan los modelos de Laravel son los encargados de comunicarse con la base de datos. En la carpeta /http/controllers se guardan los controladores de los modelos.

/config: Datos de parametrización importantes.

/database: Contiene las migraciones, seeders para crear datos falsos, etc.

/public: Contiene un index.php que es por donde entran todas las peticiones.

/resources: Tiene una carpeta /views que es donde se guarda toda la parte de html y tiene /js que centraliza el código del frontend siendo app.js el archivo principal. En app.js se registran los componentes globales, se importan las dependencias generales y se crea la instancia principal de Vue.

/routes: Tiene el archivo web.php que es de donde se definen todas las rutas que tiene la aplicación.

/vendor: Aquí se instalan todas las dependencias necesarias para que funcione el framework

El archivo **.env** con las configuraciones más importantes.

El archivo **package.json** para definir dependencias de Node en frontend.

El archivo **composer.json** para backend define dependencias de PHP.

DOCUMENTACIÓN Y CONTROL DE VERSIONES

EL trabajo de desarrollo se realiza en local, trabajando copia contra repositorio remoto en la plataforma GitHub, https://github.com/anrosalesfdez/miproyectoFP

GitHub es un repositorio en línea que usa Git. Git es un software de control de versiones. El repositorio local esta compuesto por tres "árboles" administrados por Git. El primero es el directorio de trabajo que contiene los archivos, el segundo es el Index que actúa como una zona intermedia, y el último es el HEAD que apunta al último commit realizado.

Se puede registrar cambios (añadirlos al **Index**) se utiliza el comando

```
git add <filename>
```

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios se usa

```
git commit -m "Commit message"
```

Ahora el archivo esta incluido en el HEAD, pero aún no en el repositorio remoto.

Los cambios están ahora en el **HEAD** de la copia local. Para enviar estos cambios al repositorio remoto se ejecuta:

```
git push origin master
```

Donde master es la rama a la que se quieren enviar los cambios.

Para el despliegue de la aplicación en producción, se utiliza un servidor donde se irá clonando el proyecto a medida que este este probado.

```
git pull
```

DESPLIEGUE

Para el desarrollo de este proyecto se trabaja en local y se guarda copia contra un repositorio remoto en la plataforma *GitHub*. Esto permite alojar el proyecto utilizando el sistema de control de versiones *Git.* Así, se puede ir añadiendo, *comitando* y enviado las modificaciones del proyecto al repositorio de *GitHub*.

Una vez llegado el momento de desplegar la aplicación a entorno de producción, DigitalOcean simplifica bastante las tareas de administración del servidor. Existe la opción de crear un droplet que ya incluye un paquete *LAMP on 18.04*, esto evita realizar la configuración desde cero.

Para conectarse al servidor desde un sistema Windows, se utiliza el cliente *PuTTY*.

Una vez establecida la conexión, hay que instalar todo lo que el proyecto necesite:

Actualizar del sistema:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Activar el módulo mod_rewrite de Apache:

```
sudo a2enmod rewrite
```

Al crear un *droplet*, se genera una contraseña para *MySQL,* esta contraseña se necesita para la configuración de la base de datos copiar el texto entre comillas de este archivo:

```
nano /root/.digitalocean_password
```

Iniciar la configuración de MySQL:

```
mysql_secure_installation
```

NOTAS:

- *¿Desea instalar un plugin para validar la fortaleza de las contraseñas?: No*
- ¿Desea cambiar la contraseña para el usuario principal de la base de datos?: Sí
- ¿Desea eliminar los usuarios anónimos? Sí
- Deshabilitar conexión remota a la base de datos: Sí
- ¿Desea eliminar las tablas de prueba y sus accesos? Sí
- ¿Desea recargar los privilegios de las tablas ahora? Sí

Instalar phpMyAdmin:

```
sudo apt-get install phpmyadmin
```

NOTAS:

- Seleccionar *Apache2*
- *Configure database for phpmyadmin with dbconfig-common*? YES

Editar el archivo de configuración de Apache:

```
sudo nano /etc/apache2/apache2.conf
```

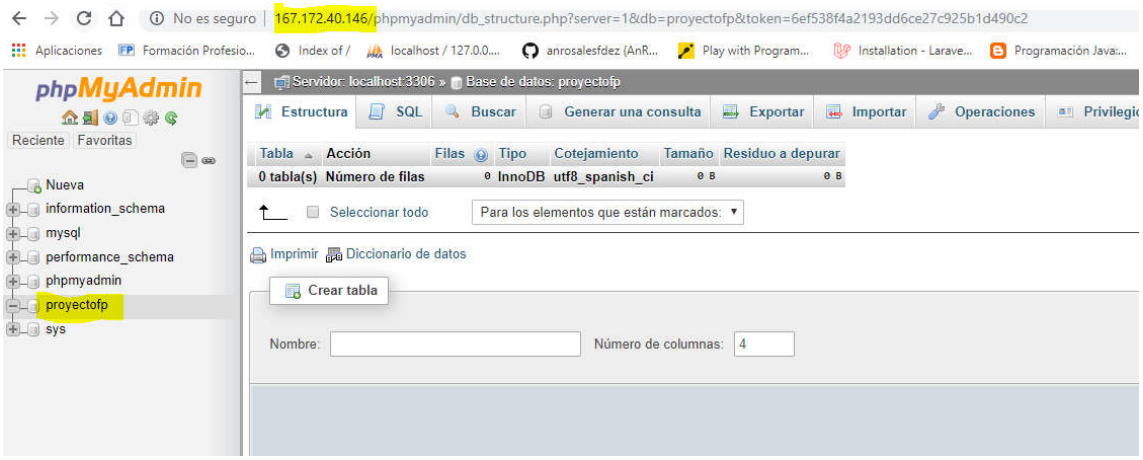
Añadir esta línea al final de todo:

```
Include /etc/phpmyadmin/apache.conf
```

Reiniciar Apache antes de continuar:

```
sudo service apache2 restart
```

La instalación que hemos hecho hasta ahora nos permitirá administrar nuestras base de datos fácilmente accediendo a `/phpmyadmin` desde la IP de nuestro droplet. En este caso:



Configuración específica para Laravel

Instalar *GIT* para hacer *deploy* a través de un repositorio:

```
sudo apt-get install git
```

Clonar el proyecto a partir de un repositorio remoto de *GitHub*.

```
cd /var/www
git clone https://github.com/anrosalesfdez/miproyectoFP.git
```

Instalar las dependencias del proyecto de *Laravel* (archivo *composer.json*)

```
composer install
```

Instalar las dependencias del proyecto de *JavaScript*.

```
npm install
```

Importante verificar versión respecto al npm instalado en local: npm -v. En caso de que sea necesario, ejecutar npm update.

Ya mencionado anteriormente, la configuración de la aplicación está en los archivos. env y en la carpeta /config. Cuando se trabaja en local y se clona al repositorio remoto, no se realiza una copia de los archivos incluidos en .gitignore.

Por ello, cuando vayamos a hacer el despliegue de la aplicación en real, hay que crear el archivo. env nuevamente.

```
nano .env
```

Tras modificar el archivo de entorno, debemos ejecutar el siguiente comando:

```
Php artisan config:clear
```

Una vez tenemos las variables de entorno seteadas, tenemos que crear una base de datos vacía. El siguiente comando de migraciones, crearán las tablas de la base de datos.

Tenemos datos de importantes en los seeders que debemos pasar también a la base de datos de producción, para ello:

```
php artisan migrate --seed
```

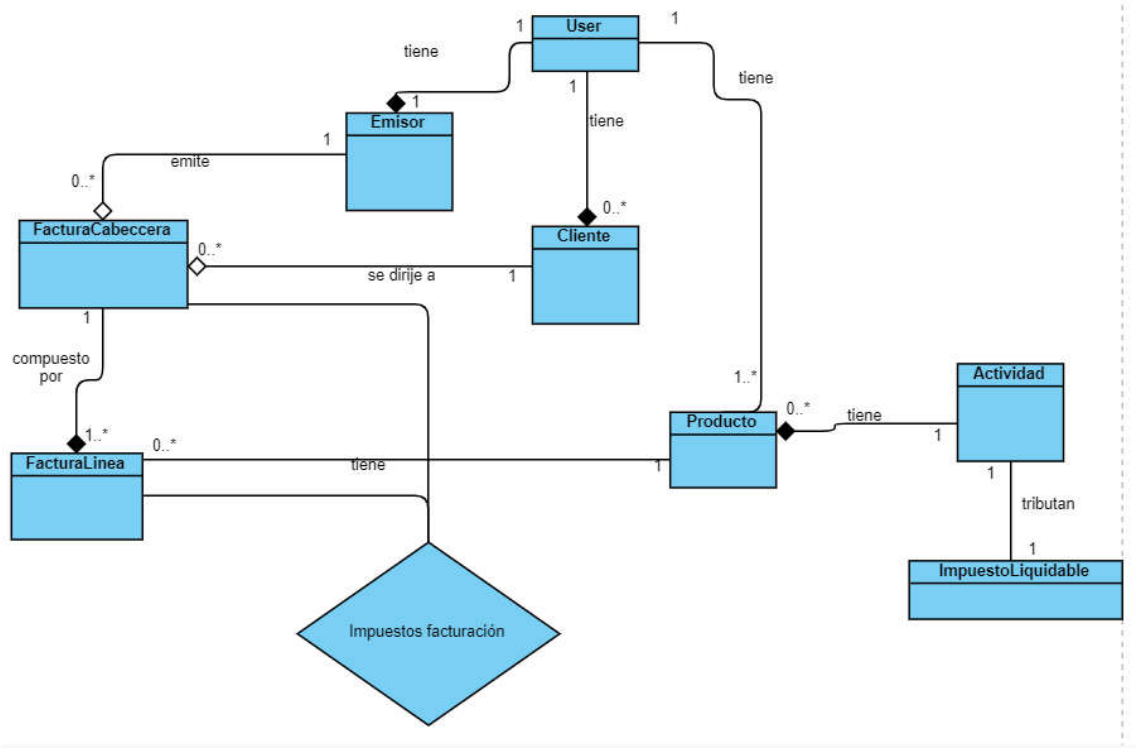
También es necesario dar permisos a la carpeta /storage del proyecto:

```
chown -R www-data: storage
chmod -R 755 storage
```

MVC: MODELO – VISTA- CONTROLADOR

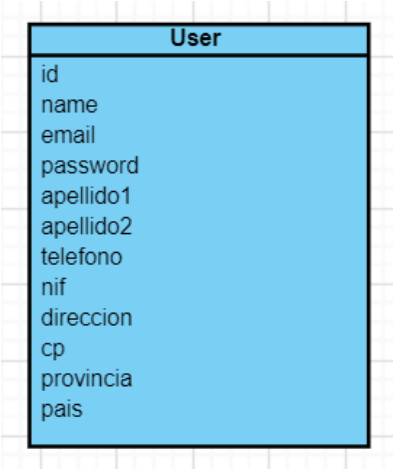
DIAGRAMA DE CLASES

Las entidades con las que tenemos que trabajar para obtener un buen sistema relacionado son:

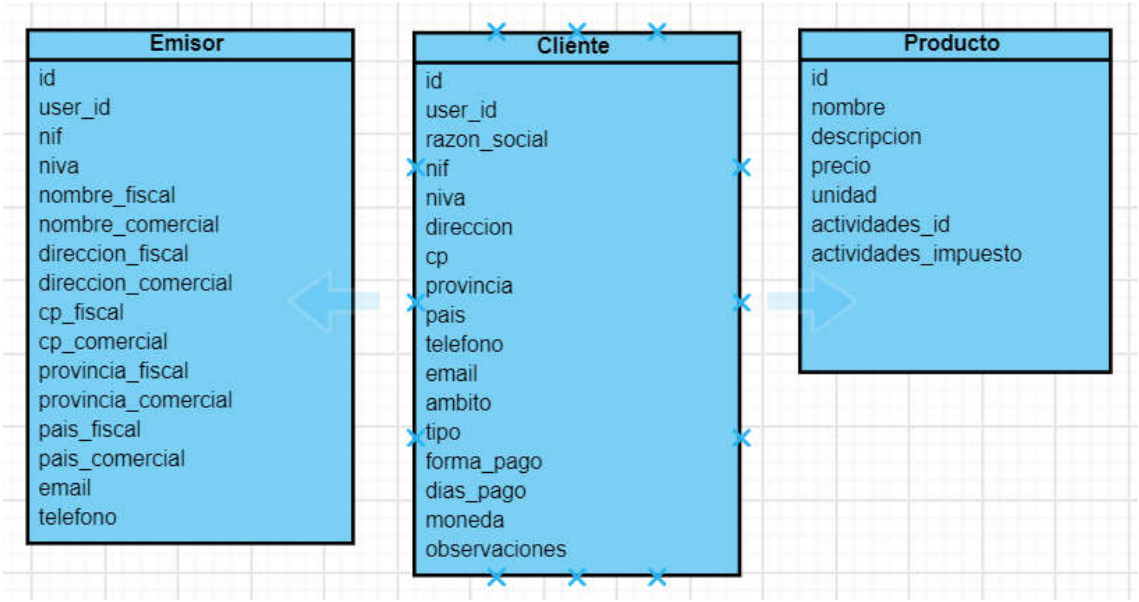


Analizando de manera individual cada una de las entidades, obtenemos los atributos para cada una de ellas.

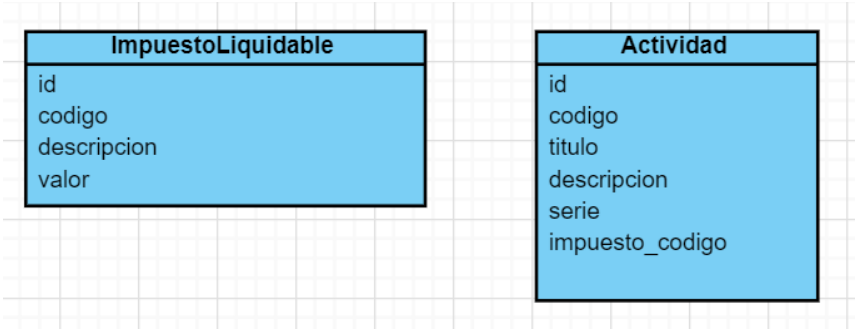
Los datos del usuario de la aplicación, son los primeros que se requieren para iniciar el circuito de la aplicación.



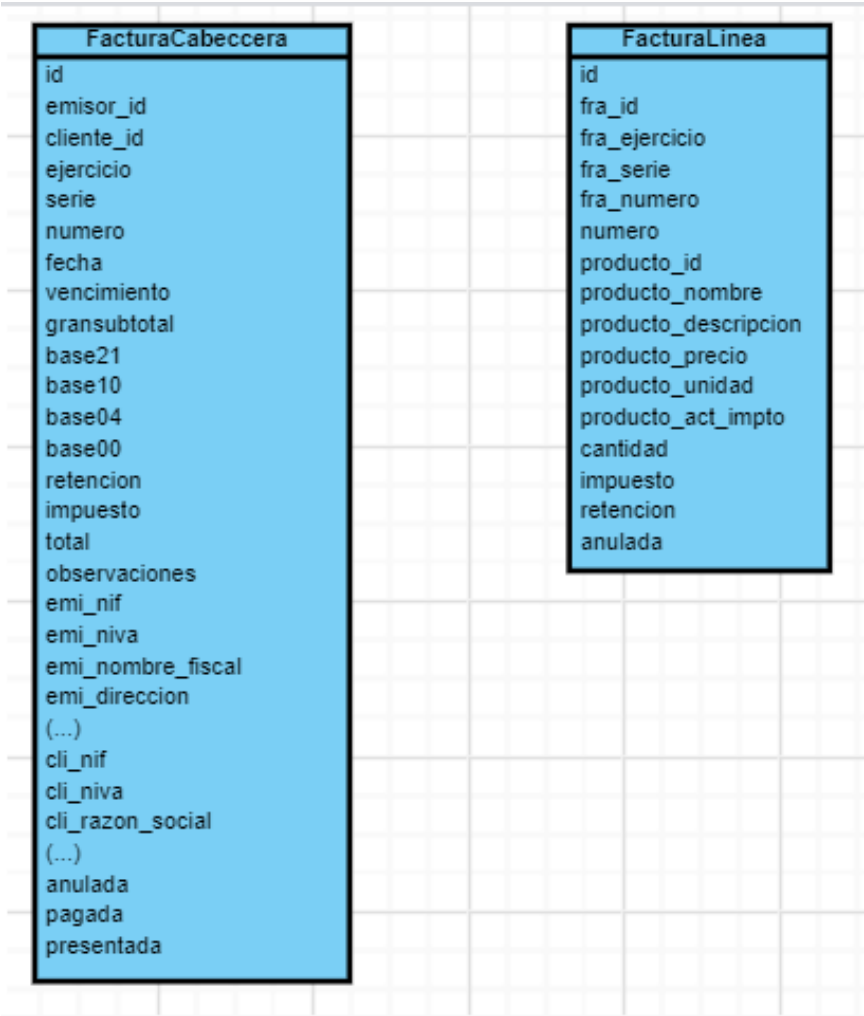
Dependiendo directamente de la existencia del usuario, surgen las 3 figuras siguientes: emisor, cliente y producto.



Existen una serie de entidades que son independientes de la creación del usuario de la aplicación. No tienen relación con él y únicamente son constantes que el programa necesita para poder trabajar.



Condicionado a la existencia de las entidades emisor, cliente y producto; nacen las facturas y las líneas de factura, que, por requisitos a la hora de crear una base relacional, se almacenan en una tabla independiente.



De la interacción entre los datos de ámbito del cliente, tipo de cliente e impuesto del producto (vía impuesto de la actividad), surge una última tabla que recoge los impuestos de facturación aplicables en cada factura emitida.

MODELOS

En *Laravel*, las entidades anteriormente descritas se materializan en forma de clases y se organizan dentro de la carpeta *app*. Creándolas en esa ruta, son reconocidas inmediatamente gracias a Composer.

Por convención, los modelos tienen la primera letra en mayúscula, por implementarse mediante clases.

Por ejemplo; la clase *User*, que representa al usuario; se generó en el momento en el creamos el sistema de autenticación que Laravel crea por defecto al ejecutar:

```
php artisan make:auth
```

El resto de clases del proyecto, fueron creadas con el siguente comando:

```
php artisan make:model ModelName -mc
```

Donde *ModelName* es el nombre de nuestra clase o modelo y la opción *-m* implica que se genere la migración inicial y la opción *-c* implica que se cree el esquema de controlador.

Al crear el modelo, se referencia a un *namespace* que simula la ubicación en las carpetas y que permitirá importar clases.

En los modelos de Laravel se informa de los atributos que tiene la clase y las posibles relaciones que existen entre los modelos.

MIGRACIONES

En el sistema Eloquent, los modelos de Laravel están directamente asociados a una entidad y a su vez a una tabla de la base de datos, por lo que un modelo que se llama *User* está directamente relacionado con una tabla llamada con el mismo nombre en la base de datos, pero en minúscula y acabado en plural: "*users*".

Laravel incluye un sistema para gestionar bases de datos, las migraciones son archivos que se encuentran el la ruta */database/migrations*. Por defecto, en la instalación de Laravel 6 se encuentran tres migraciones ya creadas:

create_users_table

create_password_resets_table

create_failed_jobs_table

Para crear migraciones en Laravel:

```
php artisan make:migration nombre_migracion –create=nombre_tabla
```

El comando anterior agrega una plantilla de trabajo básica para empezar a trabajar.

La estructura de la migración inicial que ofrece *Laravel*, es básica, e importa la clase *Schema* con la que nos permite crear las tablas, ya trae dos campos por defecto: fecha de creación y actualización.

Las migraciones en *Laravel* representan modificaciones a la base de datos sin necesidad de crear *scripts* en lenguaje propio de la base de datos. *Laravel* al crear una migración agrega como prefijo la fecha y hora en la que fue creada la migración para poder ordenar qué migración va antes que otra.

Para modificar una tabla:

```
php artisan make:migration alter_user _add_apellido --table=users
```

Los ficheros que se generan al crear las migraciones con plantilla, vienen con dos métodos; *up()* y *down()*. En la función *up()* hay que especificar el código que se va a ejecutar cuando se pasa el comando de creación de migraciones; en la función *down()* se definen las acciones a ejecutar cuando se revierta la migración.

Para iniciar las migraciones:

```
php artisan migrate
```

Con esto, si es la primera vez que se ejecuta este comando se creará en nuestra base de datos la tabla ***migrations*** que es la encargada de llevar el control de que migraciones que ya han sido ejecutadas, con el fin de no correr el mismo archivo más de una vez si el comando se usa nuevamente.

Para revertir las migraciones:

```
php artisan migrate:rollback
```

Esto eliminará la última migración ejecutada y registrada en la base de datos.

Para revertir todas las migraciones:

```
php artisan migrate:reset
```

El sistema de Eloquent permite trabajar con *soft deleting*, que realmente no elimina datos de tablas, simplemente marca los registros con una fecha *deleted_at*.

Rehacer base de datos:

```
composer dump-autoload
php artisan config:cache
php artisan migrate
```

CONTROLADORES

Los controladores normalmente se almacenan en el directorio de aplicación *app/Http/Controllers/.*

Los controladores definen la lógica de los modelos: consultas, modificaciones, etc.

Para crear controladores en *Laravel* usamos el siguiente comando:

```
php artisan make:controller NameController
```

ENRUTADO

Las rutas son una capa muy importante en *Laravel*, se encuentran en el directorio *app/routes.* En *web.php* se definen las rutas para la web y en a*pi.php*l as rutas para crear *APIs* para la aplicación.

Se escribe la clase *Route* que llama al método relacionado con el verbo *HTTP (get, post, etc.)*, la definición de la ruta acepta dos parámetros: el primero es la URL que se llamará desde el navegador y el segundo es una llamada a una acción de un controlador.

Con el sistema de rutas de *Laravel* se pueden crear rutas complejas que necesiten de parámetros dinámicos. En este caso *Laravel* se encarga de capturar el segmento de la ruta que es dinámico (lo identifica porque está encerrado entre llaves). Por tanto, en la URL pasamos la identificación del parámetro encerrado entre llaves y en el método del controlador, se recibe ese parámetro.

Cuando el uso de un parámetro no es obligatorio, se puede usar el carácter ? después del nombre del parámetro para indicar que es opcional. Sin embargo, debe añadirse un valor por defecto al parámetro cuando lo colocamos en el método.

Cuando un usuario hace una petición HTTP, *Laravel* busca en los archivos de rutas una definición que coincida con el patrón de la URL según el método HTTP usado y en la primera coincidencia le muestra el resultado al usuario. **Por tanto, el orden de precedencia de las definiciones de rutas es muy importante.**

La llamada a las rutas desde las vistas *Blade*, se hace con la ayuda del *helper route().*

VISTAS

Laravel utiliza ***Blade*** para la definición de plantillas en las vistas. Esta librería permite realizar todo tipo de operaciones con los datos, además de la sustitución de secciones de las plantillas por otro contenido, herencia entre plantillas, definición de l***ayouts*** o plantillas base, etc.

Si queremos imprimir una variable, podemos hacerlo utilizando la sintaxis de dobles llaves {{ }}

Las directivas principales de Blade son:

@extends(“”) => Implica herencia entre vistas. Esto es útil, por ejemplo, para no repetir el header y footer en cada una de las plantillas.

@yield(“”) => indica que dentro de la plantilla podemos indicar secciones (pasando como argumento el nombre de la sección) y luego en plantillas individuales podemos colocar el contenido de dichas secciones.

@section(“”) => desarrolla

Las vistas parten todas de app.blade.php (layout) que únicamente contiene un head desarrollado con las etiquetas meta, título y enlaces de estilo. También contiene la etiqueta body, pero el único contenido son dos declaraciones de partes del body: sección css y sección contenido, y el enlace al script de JavaScript del proyecto: js/app.js.

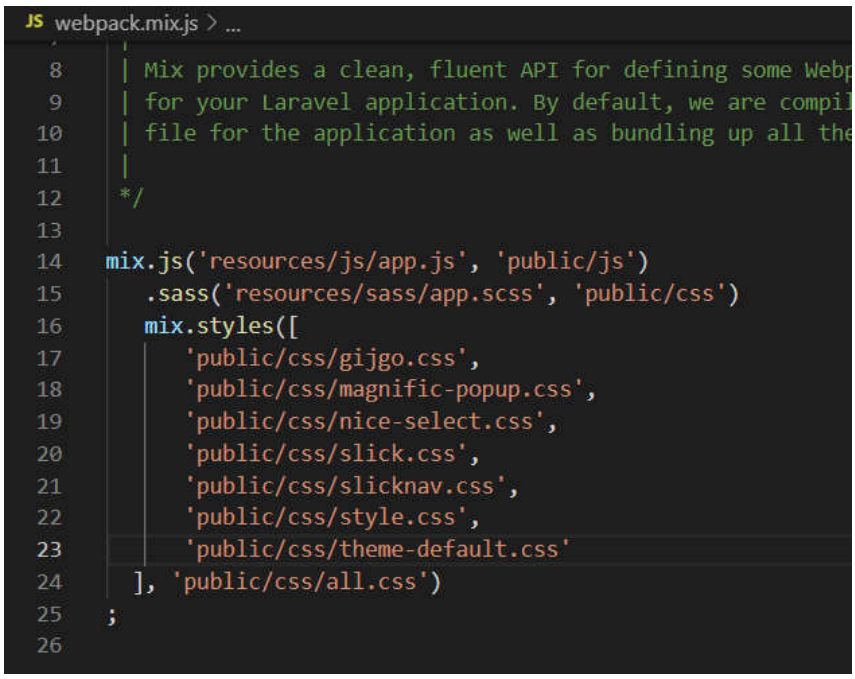
Para declarar ese contenido que será implementado posteriormente en plantillas individuales, se utiliza la directiva @yield('nombreSección').

Para usar las directivas de Blade, no es necesario colocar la extensión del archivo .blade.php al que estamos referenciando. Tampoco es necesario colocar la ruta completa, ya que Laravel por defecto buscará el archivo dentro del directorio resources/views.

Las plantillas de Blade se apoyan en helpers.

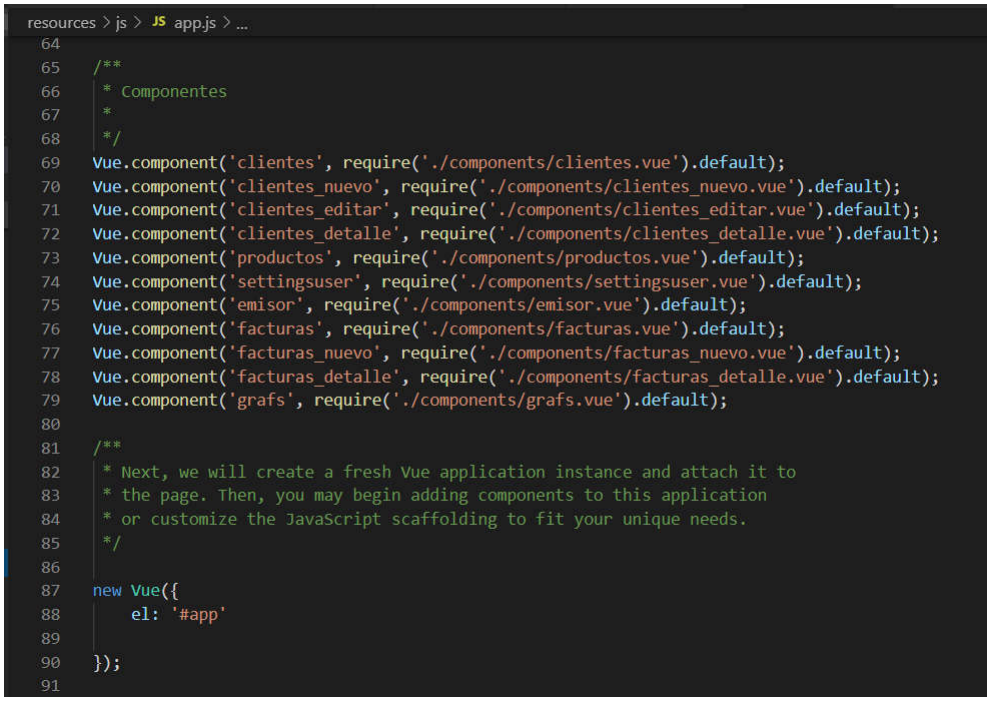
COMPONENTES VUEJS

Una vez instalamos las dependencias del proyecto con el comando npm install, se puede observar que el archivo webpack.mix.js hace referencia a resources/js/app.js, que es la ruta de la que parte la configuración básica de Vue.



```
webpack.mix.js > ...
 8    | Mix provides a clean, fluent API for defining some Webp
 9    | for your Laravel application. By default, we are compil
10    | file for the application as well as bundling up all the
11    |
12    */
13
14   mix.js('resources/js/app.js', 'public/js')
15      .sass('resources/sass/app.scss', 'public/css')
16      mix.styles([
17          'public/css/gijgo.css',
18          'public/css/magnific-popup.css',
19          'public/css/nice-select.css',
20          'public/css/slick.css',
21          'public/css/slicknav.css',
22          'public/css/style.css',
23          'public/css/theme-default.css'
24      ], 'public/css/all.css')
25   ;
26
```

En ese fichero app.js se declaran los componentes globales de VueJS, se importan los ficheros necesarios y se declara una instancia de Vue donde el parámetros *el:#app* indica en qué etiqueta HTML se va a cargar esa instancia.



```
resources > js > app.js > ...
64
65   /**
66    * Componentes
67    *
68    */
69   Vue.component('clientes', require('./components/clientes.vue').default);
70   Vue.component('clientes_nuevo', require('./components/clientes_nuevo.vue').default);
71   Vue.component('clientes_editar', require('./components/clientes_editar.vue').default);
72   Vue.component('clientes_detalle', require('./components/clientes_detalle.vue').default);
73   Vue.component('productos', require('./components/productos.vue').default);
74   Vue.component('settingsuser', require('./components/settingsuser.vue').default);
75   Vue.component('emisor', require('./components/emisor.vue').default);
76   Vue.component('facturas', require('./components/facturas.vue').default);
77   Vue.component('facturas_nuevo', require('./components/facturas_nuevo.vue').default);
78   Vue.component('facturas_detalle', require('./components/facturas_detalle.vue').default);
79   Vue.component('grafs', require('./components/grafs.vue').default);
80
81   /**
82    * Next, we will create a fresh Vue application instance and attach it to
83    * the page. Then, you may begin adding components to this application
84    * or customize the JavaScript scaffolding to fit your unique needs.
85    */
86
87   new Vue({
88       el: '#app'
89
90   });
91
```

En este proyecto, el div que inicia la instancia de VueJS se encuentra en app.blade.php

resources > views > layouts > app.blade.php > html

```blade
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>···
</head>



<body>

    <div id="app">

        @yield('content')

    </div>

    <!-- Scripts -->
    <script src="{{ asset('js/app.js') }}" defer></script>
    <!--DEFER When present, it specifies that the script is executed when the page has finished parsing.-->
    <script src="https://unpkg.com/vue"></script>
    <script src="https://unpkg.com/vue-currency-input"></script>
    <script src="vue-google-charts/dist/vue-google-charts.browser.js"></script>

</body>



</html>

```

Mediante los componentes de Vue, se permite extender elementos HTML para encapsular código. Al compilar esos elementos, Vue les añade comportamientos.

Los componentes de Vue tienen la siguiente estructura:

```vue
<template>
    <!-- Elementos HTML que se cargarán al insertar el componente Vue -->
</template>

<script>
    export default(){
        data: function(){
            // opciones que pueden ser utilizadas en un componente
        },
        props:[
            //datos de hijos
        ],
        components:{
            //datos para hijos
        },
        computed: {
            //data calculada
        },
        created() {
            //parte del ciclo de vida del componente
        },
        mounted() {
            //parte del ciclo de vida del componente
        },
        methods: {
            //diferentes acciones y cálculos que se pueden realizar en un componente
        },
    }
</script>

<style>
    /* etiqueta optativa */
</style>
```

In Vue.js, la relación padre-hijo entre componentes puede ser resumida como propiedades hacia abajo, eventos hacia arriba. El padre pasa datos al hijo a través de propiedades y el hijo

envía mensajes al padre a través de eventos.

Similar a enlazar atributos normales a una expresión, podemos utilizar v-bind para enlazar dinámicamente propiedades con datos en el padre. Cuando los datos sean actualizados en el padre, los cambios fluirán hacia el hijo.

Cada instancia de Vue implementa una interface de eventos, lo cual significa que puede:

- Escuchar un evento utilizando $on(eventName)
- Emitir un evento utilizando $emit(eventName)

Las directivas de Vue son una parte fundamental de Vue. Son atributos especiales que comienzan por v- y que recogen el valor de una expresión de JavaScript.

Son muy útiles las directivas v-model, v-if, v-else, v-for, v-bind.

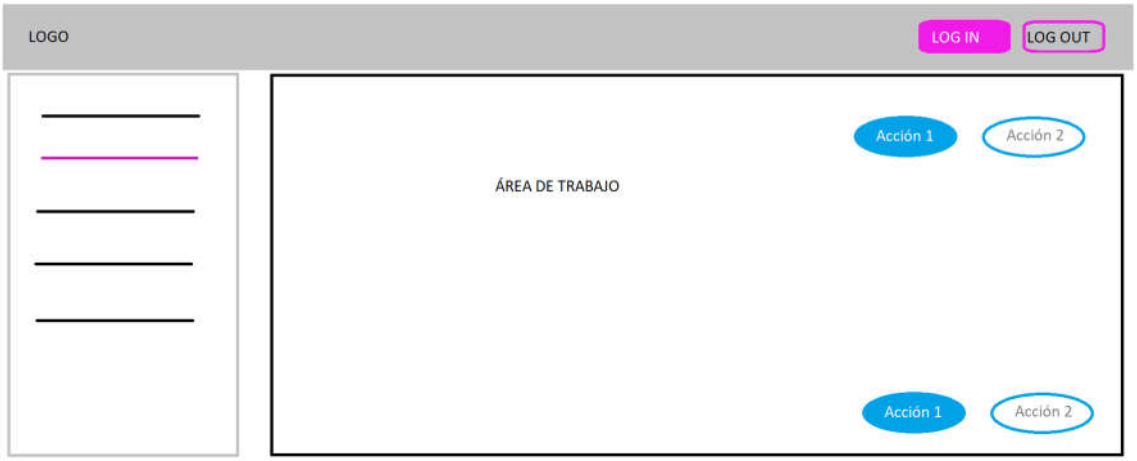
INTERFAZ CON EL USUARIO

El diseño de la interfaz de usuario y la apariencia externa que ofrece la aplicación es una cuestión muy importante que debe cumplir con unos niveles mínimos de usabilidad, utilidad y ergonomía.

Resulta fundamental atraer la atención del usuario a la zona adecuada de manera sencilla e intuitiva. El diseño gráfico y la tipografía deben utilizarse en pro de la usabilidad, influyendo en cómo el usuario realiza ciertas interacciones y mejorando la apariencia estética del diseño. Se obta por tipografías sin serifas, que a priori ayudan la lectura en pantallas.

El estilo y la estética del diseño puede mejorar o dificultar la capacidad de los usuarios para utilizar las funciones de la interfaz. En este sentido, se emplea el blanco como color primario, textos en negro. Los botones se presentan en tonos azules y rosas.

Para ello, la aplicación se apoya en dos barras de navegación: horizontal compuesta de logo, entrada en el sistema (inicio sesión) y posibilidad de desconexión; y vertical, compuesta de los bloques de trabajo que se ofrecen. El resto de pantalla disponible será el área de trabajo.

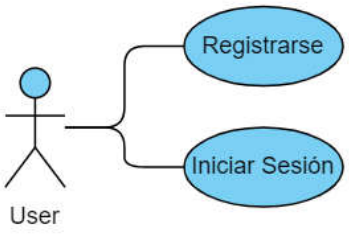


DIAGRAMAS DE CASOS DE USO

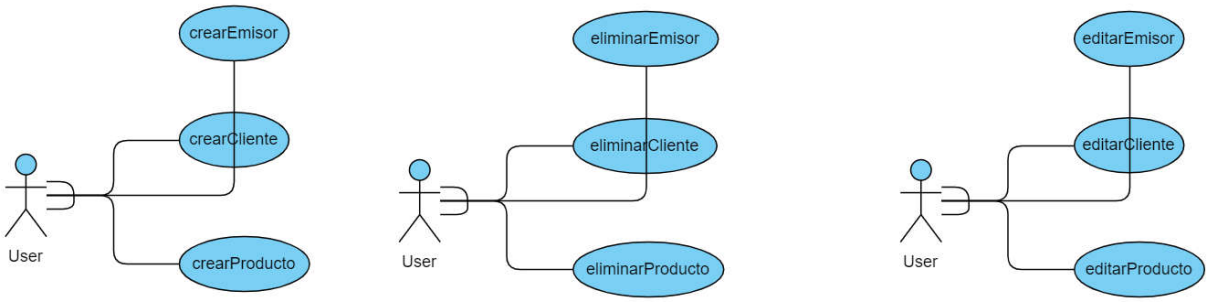
A través de estos diagramas, se muestra el comportamiento del programa en el curso de ejecución.

Los casos de uso determinan y validan los requisitos funcionales del sistema.

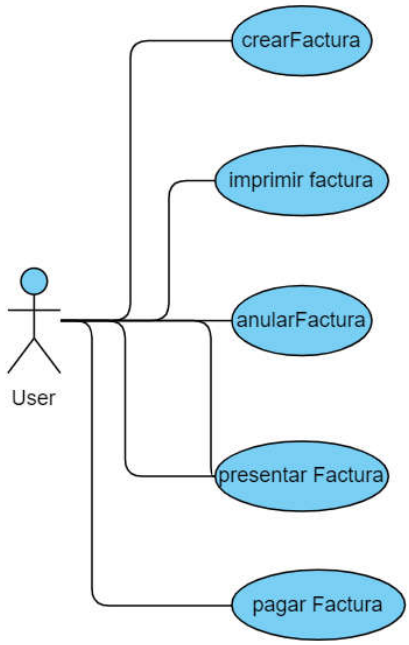
Los primeros casos de uso que se plantean son el de registrarse en la aplicación y de iniciar sesion en la misma.



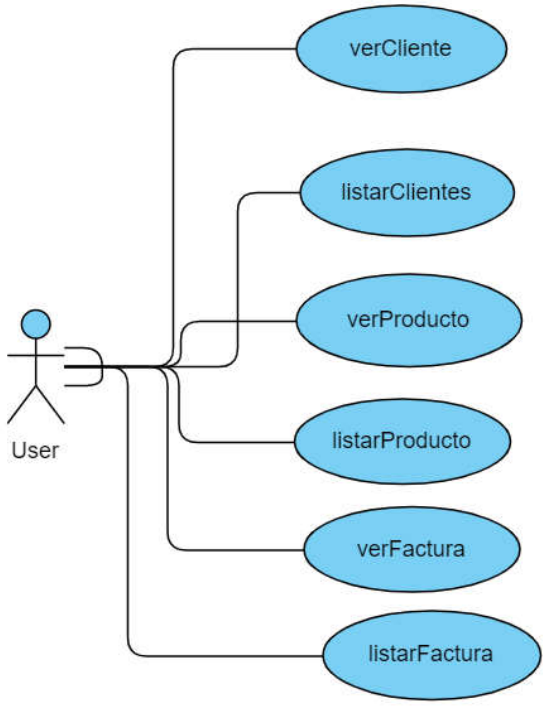
Tras esta acción, al usuario se le presentan una serie de posibles casos de uso.



Los casos de uso crearEmisor, crearCliente y crearProducto son precondiciones que deben cumplirse para que pueda llevarse a cabo el caso de uso crearFactura. Obviamente, los casos de uso de eliminación y edición, tampoco estarán habilitado en caso de no existir los primeros.



Para todas las entidades, existe un caso de uso de visualización de datos.



PRUEBAS

PRUEBAS UNITARIAS Y DE INTEGRACIÓN

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente).

A lo largo del proceso de desarrollo, se han ido probando todas y cada una de las partes de la aplicación. Se introducen entradas de valores, y se comprueba que se obtienen los resultados esperados tras la ejecución del código.

Ejemplo de estas pruebas son la creación de un usuario, la creación de un emisor, la creación de varios tipos de clientes y de productos y creación de factura.

Lo mismo sucede con la edición, actualización y borrado de datos (acciones CRUD).

No solo se analiza que la entrada de datos, lleva a cabo una salida; sino que también se analiza si el resultado es consistente y se sigue un procedimiento lógico y eficiente.

Resulta fundamental asegurar que no existen enlaces ni botones sin sentido y que todas las rutas funcionan correctamente.

Una vez que el sistema está probado de manera individual y de que se hayan realizado con éxito las pruebas unitarias; se procede a comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas en todas las variantes planeadas.

BETA TEST

Actualemente el proyecto se encuentra en esta fase que se realiza sobre el entorno de producción. Esta prueba consiste en ofrecer la aplicación a unos determinados clientes seleccionados para probar todo el sistema y reportar notas y errores.

MEJORAS y CORRECCIONES

Hay muchas acciones que se podrían implementar para asegurar el comportamiento esperado del programa, y también para ampliar su funcionalidad.

En general, el programa carece de muchas de las validaciones necesarias para asegurar la información. Por ejemplo; control de NIF, NIVA, país… Para ello, la aplicación podría ayudarse de datos embebidos de aplicaciones externas.

Analizando la aplicación por entidades; en cuanto al cliente, convendría guardar un número de cliente. Actualmente no existe un campo que recoja este dato, solo se guarda un campo id que se genera de manera auto incremental en la base de datos y que es común para todos los usuarios de la aplicación. La misma situación sucede para la entidad producto.

En esta última entidad, productos, sería interesante permitir la actualización y eliminación de datos de forma masiva.

Sería muy interesante generar una relación entre emisor de factura y actividades. Con esto, se podría permitir crear una serie de factura diferente por cada actividad. La factura solo podría contener productos de esa actividad, pero la clasificación de la información sería mucho más potente.

Asimsimo, también es conveniente incluir el motivo exención o no sujeción de impuestos en el campo observaciones de las facturas.

La aplicación estaba pensada en un inicio para cubrir, no solo el ciclo de facturación, si no que también pretendía obtener los datos necesarios para generar los modelos de impuestos trimestrales para IVA e IRPF. En las facturas emitidas, deberían existir más campos que ayuden a la categorización de las operaciones (regimen general, operaciones intracomunitarias, exportaciones, etc.).

Es interesante gestionar también la entrada de usuarios con rol de admintrador, y crear para ellos un portal de trabajo diferente. De esta manera, la gestión de tablas de impuestos, etc. Sería mucho más visual y funcional.

A nivel estético y de personalización del programa para el usuario; debería habilitarse que el usuario pueda cargar su propio logo para incluir en factura.

Los botones y barras de navegación podrían tener mejores acabados, jugar más con los volúmenes y fijar más la gama de colores: principal, secundario y fondo.

De cara a mejorar el rendimiento y posicionamineto de la aplicación, las etiquetas meta, aunque su importancia ha disminuido, ayudan a los navegadores a localizar información sobre la página web. Resulta interesante la etiqueta de mobile-web-app-capable que permite visualizar una web como una aplicación de teléfono móvil.

Uno de los asuntos más importantes que queda pendiente de realización, es el tratamiento de las cookies, la seguridad de la aplicación y el tratamiento de datos personales, asicomo la

gestión de las copias de seguridad de datos. Estos temas en si mismos, suponen una rama propia de estudio y resulta imposible de abordar por el momento.

También sería interesante, la creación de un manual para el usuario donde se guíe al cliente en el uso de la aplicación. Pasos a seguir en los diferenes casos de uso, normas de formato que deben seguir determinados campos. Esto ayudaría a incrementar la usabilidad de la aplicación y resultaría mucho más amigable.

CONCLUSIÓN

En este proyecto se pretende poner en práctica todo lo aprendido en los dos años de formación del Ciclo Desarrollo Aplicaciones Web.

Si bien el código no es operativo 100% y faltan muchas consideraciones para lanzar el proyecto al mundo real, se ha hecho un gran esfuerzo por completar al máximo la parte técnica.

BIBLIOGRAFÍA

Laravel

https://laravel.com/docs/6.x

https://ajgallego.gitbooks.io/laravel-5/content/index.html

https://richos.gitbooks.io/laravel-5/content/capitulos/chapter11.html

https://styde.net/caracteristicas-que-debes-conocer-de-las-rutas-en-laravel/

PHP

https://www.php.net/manual/es/index.php

VUEJS

https://vuejs.org/v2/guide/

https://rimorsoft.com/bienvenidos-al-curso-de-vue-2

https://www.npmjs.com/package/axios

https://www.npmjs.com/package/vue-tables-2

https://elabismodenull.wordpress.com/2017/05/02/vuejs-creando-componentes/

https://www.inarvis.com/blog/tutorial-basico-vue-js-2/#laravel-y-vue-para-crear-aplicaciones-web

https://jsfiddle.net/8mgszvec/12/

https://www.npmjs.com/package/vue-google-charts

ESTILO

https://getbootstrap.com/docs/4.0/components/forms/

https://colorlib.com/wp/template/seogo/

SERVIDOR LOCAL

https://www.ionos.es/digitalguide/servidores/herramientas/instala-tu-servidor-local-xampp-en-unos-pocos-pasos/

DESPLIEGUE

https://programacionymas.com/blog/hacer-deploy-app-laravel-digital-ocean

https://guides.github.com/features/wikis/

https://desoft.es/politica-de-cookies/

https://www.ionos.es/digitalguide/servidores/herramientas/protocolo-ssh/

UML

https://online.visual-paradigm.com/drive/#diagramlist:proj=0&new=UseCaseDiagram