

Proyecto final de grado

Cantina

| | |
|--------------------------|-------|
| José Luis Nicolás Rubio | 2°DAW |
| Manuel Martínez Corcoles | 2°DAW |
| Eduardo Noguera García | 2°DAW |

Tienda web para la cantina



Índice

| | |
|---------------------------------------|----|
| Índice | 1 |
| Introducción | 2 |
| Objetivos principales y planificación | 3 |
| Estilo y decisiones de diseño | 4 |
| Tecnologías usadas | 7 |
| - Tailwind CSS | 7 |
| - AlpineJS | 20 |
| - Image Intervention | 22 |
| - Livewire | 25 |
| - Laravel-Breeze | 41 |
| - Laratrust | 46 |
| - PL/SQL | 51 |
| Problemas encontrados | 55 |
| Conclusiones | 56 |
| Herramientas utilizadas | 57 |
| Referencias | 58 |

Introducción

Este proyecto tiene como propósito tanto el desarrollo de una aplicación web con la que poder realizar pedidos de una manera cómoda y sencilla para el usuario a cómo a su vez ahondar más en la materia del desarrollo web y sus tecnologías. Dentro de la aplicación se han contemplado a su vez una interfaz de usuario agradable y sencilla con la que realizar sus compras como un apartado para la gestión de los mismos por parte de los empleados de la cantina y además de su correspondiente panel de administración de toda la aplicación. Este proyecto ha sido desarrollado sobre el framework de php Laravel siguiendo el estándar TALL (Tailwind, Apline JS, Livewire, Laravel).

A lo largo de los siguientes puntos se expondrá con mayor detalle los pasos seguidos para el desarrollo y las tecnologías utilizadas en el mismo.

Objetivos principales y planificación

Entre los objetivos más generalistas se encuentran los siguientes:

- Generar una aplicación web que simplifique la realización de pedidos en la cantina del centro.
- Dotar a la aplicación de una estructura simple que no agobie al usuario
- Que haga hincapié en la responsividad ya que en su gran mayoría va a ser visualizada desde dispositivos móviles.

Entre los objetivos específicos de este proyecto se encuentran los siguientes:

- Distribuir el contenido de la aplicación en bloques aislados para evitar la aglomeración y fatiga visual del usuario.
- Hacer uso de livewire y sus cargas asíncronas de contenido en la parte cliente y en la parte de gestión del pedido.
- La página principal tiene que concentrar una visión general del sitio y lo que en él se ofrece sin tener que recorrer todas las subpáginas dejando estas últimas a la edición del perfil y el historial de pedidos.
- Se facilitan los diferentes enlaces a los apartados de la aplicación de manera accesible desde cualquier punto de la página (Panel lateral).

Estilo y decisiones de diseño

Antes de poner un pie en un centenal plano, horizontal y extenso de lo que parecen atisbos de un diseño web moderno y actual, permitidnos poner el pie restante que nos queda en la composición visual y estructural que está, nuestra aplicación web, nos muestra.

Dejando hacia un lado la posibilidad de grabar estas palabras de forma poética en un documento en ‘blanco, negro y poco color’, centrémonos en el diseño, artístico o no, de esta aplicación web.

Como posdata (o añadido) a la introducción de esta memoria, este proyecto tiene como finalidad dotar de una progresión conceptual en la estructura y diseño de una página web ‘moderna’, siguiendo los principios de usabilidad y accesibilidad de la información. En base a homogeneizar la imagen de la empresa y de guardar coherencia con la marca del centro, se ha redactado esta mini guía de estilos para comprender mejor qué camino debía de seguir el desarrollo artístico de la aplicación.

Estilo Visual. Colores, fuentes e iconos.

La paleta de colores usada se asemejará a los colores del logo principal, siendo estos el color principal amarillo (#FFC000) y el color principal azul marino (#004467). La tipografía usada es ‘Poppins’, única en toda la web y carente de párrafos sin horizonte. No escasea la falta de iconos, por lo que la navegación y comprensión de la web resultará más amena para el usuario.

Equilibrio visual

El equilibrio visual que se intenta transmitir es el de un estilo formal, dando un mayor enfoque a los elementos situados en el centro de la pantalla. Dispondrá de una barra lateral con la que el usuario interactuará para facilitar la navegación del sitio.

Tensión compositiva

De elaborados rítmicos y alineados. Dinámica, centrándonos en las imágenes, textos e iconos que componen la web. Seguirá el sentido vertical habitual, dotando de la mayor cantidad posible de espacio en blanco donde relajar la mirada mientras navegamos por la web.

Arquitectura y Diseño. Primeros bocetos

Para la elaboración de los primeros compases del diseño de nuestro proyecto, se ha usado la herramienta **FIGMA**. **FIGMA** es un editor de gráficos vectoriales y una herramienta de prototipado para proyectos de diseño colaborativo en equipos.

FIGMA nos brinda todas las herramientas necesarias para la fase de diseño del proyecto, incluidas herramientas vectoriales, así como aquellas para la creación de prototipos y la generación de código para su traspaso ('hand-off' -> CSS, iOS y Android).

La naturaleza “*siempre en línea*” de **FIGMA** permite la colaboración y presentación en vivo y en tiempo real de todos los cambios que realicemos. Todo el mundo puede dejar sugerencias en el panel de comentarios, así como referenciar o marcar cualquier componente del diseño para su posterior trato.

- *¿Cómo funciona FIGMA?*

Como ya hemos comentado anteriormente, **FIGMA** es una aplicación de navegador para diseñar interfaces UI/UX contando con excelentes herramientas de diseño, creación de prototipos y generación de código. **FIGMA** nos permite dividir nuestro trabajo en páginas, adaptadas a cada tipo de navegador. Por ejemplo, si nuestro proyecto es ‘mobile-first’, podemos seleccionar un ‘frame’ del tamaño de un iPhone X y comenzar desde ahí. Sin embargo, si queremos adaptar todas las funcionalidades en un espacio de trabajo más grande, podemos seleccionar un ‘slide’ de 16:9 (1920x1080px).

Gracias a un marcado ‘Pixel-Perfect’ podemos distribuir los elementos en posiciones relativas a nuestro ‘frame’ o ‘slide’ principal, por lo que dibujar cualquier figura geométrica será coser y cantar, así como posicionarla donde queramos. Con la agrupación de elementos, es posible crear estructuras robustas y repetitivas, si lo deseamos, en nuestro diseño.

El uso repetitivo de un color principal en **FIGMA** desbloqueará la opción de guardar este color en la barra lateral derecha de nuestro proyecto para uso ‘*a posteriori*’. Dicha barra lateral nos provee de numerosas opciones, como por ejemplo: colorear un componente, distribuir la indexación de varios objetos, añadir efectos de sombreado a un elemento, escoger la alineación y el tamaño del texto, especificar una máscara artificial donde colocar nuestros elementos en columnas, así como una red de enlaces entre componentes donde poder satisfacer las transiciones entre páginas, ¡como si fuera una web real!

Actualmente es (posiblemente) la herramienta líder en la industria para diseñar interfaces y cuenta con características sólidas que respaldan a los equipos que trabajan en cada fase del proceso de diseño. Aquí, el único límite, es la imaginación.

Podéis encontrar el ‘Draft’ del proyecto visitando este enlace: [FIGMA - Cantina](#)

Tecnologías usadas

Tailwindcss. Integración de la maqueta en el proyecto.

Contando con un maquetado base gracias a la herramienta **FIGMA**, podemos dar pie a la inserción de este al proyecto. Para ello, se utilizará la tecnología **TAILWINDCSS**, un marco de trabajo CSS, útil para construir y dotar de estilos rápidamente a nuestro sitio web.

TAILWINDCSS funciona con la escritura de multitud de clases en el propio HTML de nuestras vistas. Estas clases pueden especificar la altura y anchura de una etiqueta, su disposición, orden, color, visibilidad, capas, animaciones y un largo etcétera. No se detallarán cada una de estas clases en profundidad, pero a lo largo de esta sección comentaremos las más utilizadas y los problemas encontrados durante el proceso.

Para más información y funcionalidad de estas clases, tenga en cuenta la documentación que ofrece la página web oficial de **TAILWINDCSS**: [Documentación oficial de TailwindCSS](#)

TAILWINDCSS nos ofrece de primera mano una guía de instalación en Laravel, con la que de manera muy sencilla podremos instalar esta herramienta en nuestro proyecto. Laravel cuenta con 2 instaladores de paquetes principales, que son: ‘Composer’ y ‘NPM’. Para este proyecto, se ha utilizado ‘NPM’. Corriendo el siguiente comando por consola: “*npm install -D tailwindcss postcss autoprefixer*”, **TAILWINDCSS** quedará instalado en nuestro proyecto junto con un archivo de configuración en la raíz de este llamado: ‘tailwind.config.js’

```

const defaultTheme = require('tailwindcss/defaultTheme');

module.exports = {
    purge: [
        './vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php',
        './storage/framework/views/*.php',
        './resources/views/**/*.blade.php',
    ],
    darkMode: 'class',
    theme: {
        extend: {
            fontFamily: {
                poppins: ['Poppins', ...defaultTheme.fontFamily.sans],
            },
        },
    },
    variants: {
        extend: {
            opacity: ['disabled'],
        },
    },
    plugins: [require('@tailwindcss/forms')],
};

```

Este archivo de configuración nos permite modificar ‘*cualquier*’ elemento que **TAILWINDCSS** tome por defecto, y usarlo como nuevas clases adicionales o funcionalidades completamente únicas. A continuación, se detallará cada uno de los parámetros que se muestran en la captura anterior:

- ‘purge’ -> Especifica las rutas a todos los archivos de plantilla de tu proyecto. Por defecto apunta a aquellas carpetas donde se almacenan las vistas en Laravel.
- ‘darkMode’ -> Ahora que el modo oscuro es una característica de primera necesidad de muchos sistemas operativos, es cada vez más común diseñar una versión oscura del sitio web para acompañar el diseño por defecto. Este, es un parámetro exclusivo de **TAILWINDCSS** en el que se especifica cómo se utilizará esta función en nuestro

proyecto. Por defecto vendrá desactivada, más adelante, en una sección independiente, explicaremos sus otras 2 modalidades: ‘*class*’ y ‘*media*’.

- ‘*theme*’ -> Aquí es donde se define la paleta de colores del proyecto, la escala tipográfica, las fuentes, los puntos de ruptura y mucho más. La tipografía que trae **TAILWINDCSS** por defecto es ‘*sans-serif*’.
- ‘*variants && plugins*’ -> La sección de plugins permite registrar plugins con **TAILWINDCSS** que pueden ser utilizados para generar utilidades extra, componentes, estilos base o variantes personalizadas.

Visitando este enlace podemos conocer de primera mano los valores por defecto que toma **TAILWINDCSS** a la hora del maquetado con clases en nuestra web: [‘Configuración por defecto TAILWINDCSS’](#)

Para acabar con el apartado de instalación, debemos incluir **TAILWINDCSS** como un plugin *POSTCSS*, tal y como se muestra en la siguiente imagen:

```
mix.js('resources/js/app.js', 'public/js').postCss('resources/css/app.css',  
'public/css', [  
    require('postcss-import'),  
    require('tailwindcss'),  
    require('autoprefixer'),  
]);
```

Añadimos las directivas *@tailwind* para cada una de las capas de **TAILWIND** al archivo ‘*./resources/css/app.css*’.

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

Arrancamos y cargamos el ‘Front-End’ con ‘*npm run watch/dev/prod*’ dentro de nuestro espacio de trabajo.

Finalmente, nos aseguramos de que el CSS como el JS compilado está incluido en el <head> de nuestro ‘layout’ para poder empezar a usar las clases de utilidad de **TAILWINDCSS** y dar forma a nuestro contenido.

```
<!-- Styles -->
<link rel="stylesheet" href="{{ asset('css/app.css') }}">
<link rel="stylesheet" href="{{ asset('css/extrastyles.css') }}">

<!-- Fonts -->
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Poppins&display=swap" rel="stylesheet">

<!-- Scripts -->
<script src="{{ asset('js/app.js') }}" defer></script>
```

Estructura de la web. Plantillas, componentes de **BLADE** y Layout principal.

Una vez terminado el proceso de instalación de **TAILWINDCSS** ya podemos disfrutar de las ventajas que este nos ofrece. Para una mayor sencillez y planificación de nuestras vistas, se ha tenido en cuenta multitud de ‘templates’ que reflejen en la mejor medida de lo posible la funcionalidad de nuestra web. Por ello, hemos utilizado los componentes que nos brinda [Wickedtemplates](#) – Elementos adaptables (‘responsive’) construidos con **TAILWINDCSS** y **ALPINEJS**.

El ‘layout’ principal de un proyecto constituye de una vista ‘padre’ de la que dependerán todas las vistas ‘hijas’. Esta vista padre es nuestro ‘layout’, denominado ‘app.blade.php’ almacenado en: ‘/resources/views/layouts/’. En este ‘layout’ se referenciará el CSS a utilizar, así como el JS, fuentes personalizadas y scripts de carga.

En nuestro proyecto, hemos tenido en cuenta la optimización, reutilización y uso óptimo de elementos HTML que se repiten en nuestras vistas. **BLADE** nos da la opción de crear una serie de componentes para poder ser reutilizados en toda la web (muy parecido a los componentes de **VUE.js**), estos elementos se conocen como ‘Componentes de

BLADE. Al igual que un ‘*layout*’, un componente de **BLADE** puede definirse a sí mismo de la misma manera, pero esta afirmación no es del todo exacta.

Estos componentes pueden anidarse unos con otros y transmitir datos. He aquí una captura de un componente ‘*input*’ HTML:

```
@props(['disabled' => false])  
  
<input {{ $disabled ? 'disabled' : '' }} {!! $attributes->merge([  
    'class' => 'rounded-md shadow-sm border-gray-300 focus:border-indigo-300 focus:ring focus  
]) !!}>
```

Estos componentes pueden albergar elementos predefinidos cuyo valor podrá cambiar dependiendo de la funcionalidad del elemento. Por ejemplo, en este ‘*input*’, el parámetro **\$disabled** determinará si este componente se encontrará deshabilitado por defecto o, de lo contrario, como se muestra en la imagen anterior, se encontrará habilitado.

También podemos pasar, en formato cadena o de array, una serie de clases que añadirán un mayor peso funcional a este componente. En la imagen anterior, podemos observar que **\$attributes** especifica una serie de clases por defecto que acompañarán al componente. Bien, si queremos renderizar este componente en una vista, lo haríamos como se muestra en la siguiente imagen:

```
<x-input id="password" class=  
"block mt-1 w-full main-color-blue-text bg-transparent dark:main-color-yellow-text"  
        type="password"  
        name="password"  
        required autocomplete="new-password" />
```

Comenzando con el carácter ‘<x-’, seguido del nombre del componente almacenado en la carpeta “**resources/views/components/*” -> **‘input’**, conseguiremos mostrarlo por pantalla. Como cualquier etiqueta HTML, podemos establecer un identificador, seguido de una serie de parámetros donde recogeremos los datos que necesitemos. En la anterior imagen, hemos establecido una serie de clases que se ‘fusionaran’ con las que

ya residen en nuestro componente 'input'. He aquí otros ejemplos de componentes de **BLADE**:

- *BUTTON*

```
<button {{ $attributes->
merge(['type' => 'submit', 'class' => 'inline-flex items-center px-4 py-2 bg-transparent bord
{{ $slot }}}
</button>
```

```
<x-button class=
"ml-4 main-color-yellow-text main-color-blue-bg transition duration-500">
{{ __('Regístrate!') }}
</x-button>
```

- *LABEL*

```
@props(['value'])

<label {{ $attributes->
merge(['class' => 'block font-medium text-sm main-color-blue-text dark:main-color-yellow-text
{{ $value ?? $slot }}]
</label>
```

```
<x-label for="email" :value="__('Email')" />
```

- *ERRORS* (*También podemos mostrar los errores de un formulario mediante un componente*)

```
@props(['errors'])

@if ($errors->any())
    <div {{ $attributes }}>
        <div class="font-medium text-red-600">
            {{ __('¡Vaya! Algo no salió como se esperaba...') }}
        </div>

        <ul class="mt-3 list-disc list-inside text-sm text-red-600">
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

```
<!-- Validation Errors -->
<x-auth-validation-errors class="mb-4" :errors="$errors" />
```

Todas aquellas vistas ‘hijas’ que dependan del ‘layout’ principal pueden enlazarse mediante la etiqueta:

```
<x-app-layout>
    <!-- Aquí se mostrará tu contenido -->
</x-app-layout>
```

Adaptando estas ‘maquetas prefabricadas’, junto con el diseño de **FIGMA** y los componentes de **BLADE**, damos por concluida la funcionalidad ‘básica’ de **TAILWINDCSS** en nuestro proyecto. Pasemos ahora a ver detalles más específicos.

‘TAILWINDCSS Utilities’ - Utilidades y capas en TAILWINDCSS

Cuando se necesita añadir reglas CSS realmente personalizadas a un proyecto de **TAILWINDCSS**, lo más fácil es añadir el CSS personalizado en la hoja de estilos. Sin embargo, para mayor potencia, también podemos utilizar la directiva de **TAILWINDCSS** ‘@layer’ para añadir estilos a las capas base, componentes y utilidades (dentro de la carpeta: ‘resources/css/app.css’). En la siguiente imagen utilizamos esta función para dotar de la tipografía ‘Poppins’ a toda la aplicación, usar características específicas de CSS (‘backdrop-filter’), así como variantes de modos y usos en la aplicación: el ‘**Modo Oscuro**’:

```
@layer base {
  body {
    font-family: Poppins, sans-serif !important;
  }
}

@layer utilities {

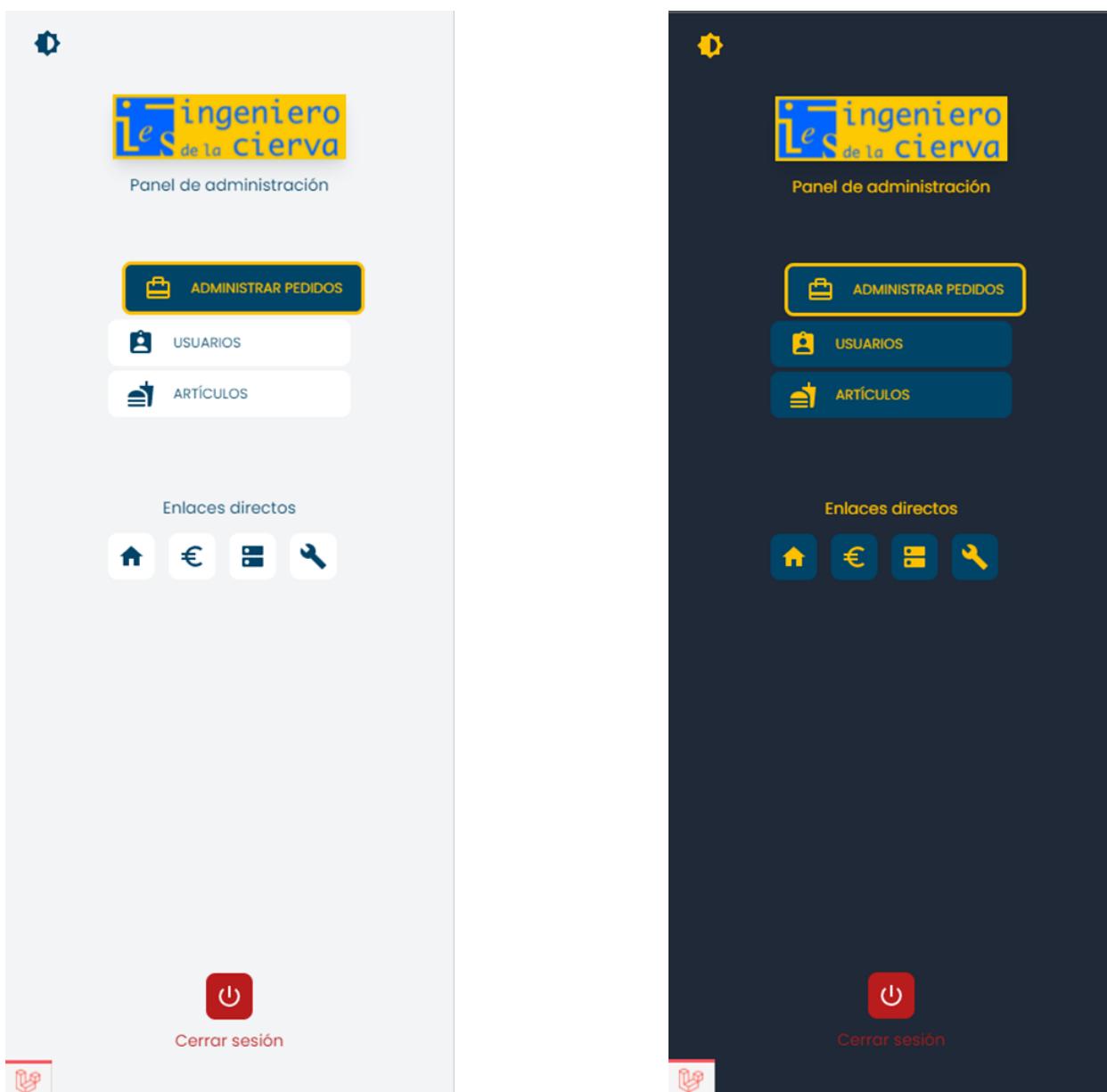
  .backdrop-filter {
    backdrop-filter: blur(5px);
  }

  @variants dark {
    .main-color-blue-bg {
      background-color: #004467 !important;
    }
    .main-color-blue-border {
      border-color: #004467 !important;
    }
    .main-color-blue-text {
      color: #004467 !important;
    }
    .main-color-blue-text path {
      fill: #004467 !important;
    }
    .main-color-blue-text svg path {
      fill: #004467 !important;
    }
    .main-color-blue-text svg line {
      stroke: #004467 !important;
    }
    .main-color-yellow-bg {
      background-color: #FFC000 !important;
    }
    .main-color-yellow-border {
      border-color: #FFC000 !important;
    }
    .main-color-yellow-text {
      color: #FFC000 !important;
    }
    .main-color-yellow-text path {
      fill: #FFC000 !important;
    }
    .main-color-yellow-text svg path {
      fill: #FFC000 !important;
    }
    .main-color-yellow-text svg line {
      stroke: #FFC000 !important;
    }
  }
}
}
```

‘TAILWINDCSS Utilities’ – Modo Oscuro

Ahora que el modo oscuro es una característica fundamental de muchos sistemas operativos, es cada vez más común diseñar una versión ‘oscura’ de su aplicación web para acompañar al diseño por defecto.

Para facilitarlo al máximo, **TAILWINDCSS** incluye una variante oscura que nos permite dar un estilo diferente a nuestras vistas cuando el modo oscuro está activado. En la siguiente imagen, podemos ver un ejemplo en nuestra aplicación:

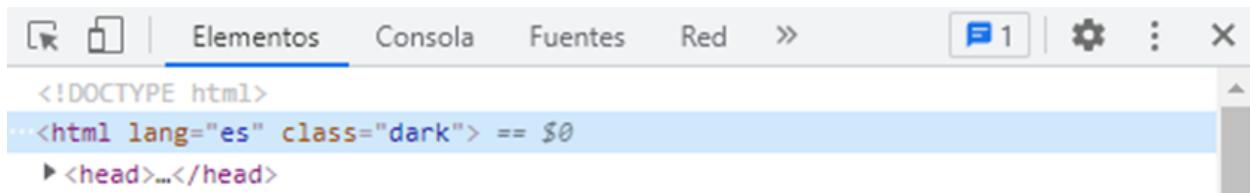


Por defecto, esto utiliza la función de medios CSS ‘prefers-color-scheme’, pero también podemos construir vistas que soporten la comutación del modo oscuro manualmente utilizando la estrategia ‘toggle-class’. ¿Cómo lo hemos hecho? Pues de la siguiente manera:

```
darkMode: 'class',
```

Habilitamos el ‘Modo Oscuro’ desde el archivo de configuración de **TAILWINDCSS**. Podemos seleccionar hasta 2 formas de usar esta funcionalidad en nuestras vistas:

- ‘media’ -> Esta opción depende del tema seleccionado en nuestro sistema operativo. Si nuestro tema es ‘oscuro’, nuestra web utilizará el modo oscuro de serie. Al cambiar de tema, nuestras vistas harán lo propio.
- ‘class’ -> La opción más común en páginas web y la que hemos incorporado. Esta opción permite al usuario marcar/desmarcar manualmente la opción de oscurecer las vistas y almacenar el ‘Modo Oscuro’ mediante una clase padre en la raíz de nuestra vista:



TAILWINDCSS nos facilita una serie de líneas de código con la que podremos almacenar en el ‘localStorage’ la decisión del usuario de mantener el modo oscuro en su navegador, incorporando o removiendo la clase ‘dark’ en el HTML de nuestra vista padre. En el ‘layout’ de nuestra aplicación, incorporamos:

```

<script type="text/javascript">

//On page load or when changing themes, best to add inline in `head` to avoid FOUC
    window.onload = function() {
        if (localStorage.theme === 'dark' || !(('theme' in localStorage) && window.matchMedia('(prefers-color-scheme: dark)').matches)) {
            document.documentElement.classList.add('dark')
        }
    };
    const toggleDarkMode = function() {
        if (localStorage.theme === 'dark' || !(('theme' in localStorage) && window.matchMedia('(prefers-color-scheme: dark)').matches)) {
            localStorage.theme = 'light'
            document.documentElement.classList.remove('dark')
            // Whenever the user explicitly chooses light mode
        } else {
            localStorage.theme = 'dark'
            document.documentElement.classList.add('dark')
            // Whenever the user explicitly chooses dark mode
        }
    }

    $(window).on("load", function() {
        $(".loading-screen").fadeOut("slow");
    });

</script>

```

Junto con el botón que llama a la función:

```

<button x-data onclick="toggleDarkMode()" class="focus:outline-none focus:shadow-outline dark"
        <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://
        <path d=
        "M20 15.31L23.31 12L20 8.69V4H15.31L12 0.690002L8.69 4H4V8.69L0.690002 12L4 15.31V20H8.69L12
        fill="#004467"/>
        </svg>
</button>

```

Por consola, podremos ver qué tema estamos utilizando en este momento:



Para aplicar el modo oscuro en nuestras clases de **TAILWINDCSS**, incorporaremos el prefijo 'dark:' antes de una clase. Esta nueva clase será utilizada siempre y cuando el modo oscuro esté activado, de lo contrario, no tendrá ningún efecto.

AlpineJS. Elementos dinámicos sencillos

ALPINE es una herramienta robusta y minimalista para dotar de comportamiento dinámico directamente en el HTML. Recuerda mucho a JQuery para webs actuales.

ALPINE se compone de una colección de 15 atributos, 6 propiedades y 2 métodos.

Por suerte o por desgracia, en nuestro proyecto no hemos necesitado de muchas de estas propiedades para incorporar lo que andábamos buscando. Dado que los atributos que ofrece esta tecnología son muy parecidos a los de **VUE.JS**, su comprensión y utilización no requiere de mucha explicación y detalle.

A continuación, enfocaremos una pequeña lista de las directivas y su uso utilizadas en el proyecto:

- x-data: Declara un nuevo componente **ALPINE** e inicializa sus datos para un bloque HTML determinado. Podría decirse que es la directiva ‘padre’ de la que dependen las demás.
- x-bind: Establece dinámicamente los atributos de un elemento, como pueden ser clases, identificadores o atributos más complejos: ‘autofocus, required, disabled, etc...’
- x-on: Escucha los eventos que se producen en el navegador. El más común es ‘x-on:click’, con el que podremos inicializar una función o cambiar variables.
- x-show: Muestra o deshabilita el contenido de un elemento. Puede resultar muy útil para etiquetas ‘dropdown’ en HTML. Esta directiva suele estar acompañada de otro atributo denominado ‘x-transition’, con el que podemos establecer una animación más fluida al mostrar u ocultar elementos.
- x-transition: Cómo ya se ha comentado en el apartado anterior, esta directiva se encarga de transicionar un elemento usando transiciones CSS. A su vez, puede usarse con otras directivas para obtener una transición mucho más personalizada, pudiendo añadir incluso clases de **TAILWINDCSS**. He aquí el código de transición de apertura y cierre de nuestro panel de administración:

```
x-cloak x-show="open"
  x-transition:enter="transition ease-out duration-500"
  x-transition:enter-start="opacity-100 transform -translate-x-full"
  x-transition:enter-end="opacity-100 transform translate-x-0"
  x-transition:leave="transition ease-in duration-500"
  x-transition:leave-start="opacity-100 transform translate-x-0"
  x-transition:leave-end="opacity-100 transform -translate-x-full">>
```

- x-cloak: Oculta un bloque HTML hasta que **ALPINE** haya terminado de inicializar su contenido. Resulta de gran utilidad en pantallas de carga

Image intervention

Para el tratamiento de las imágenes dentro de la aplicación hemos hecho uso de la librería Image Intervention que entre muchas de las posibilidades que ofrece nos permite ajustar a la resolución deseada las imágenes subidas por el usuario al tiempo que facilita la labor de almacenarlas. Aquí el código empleado tanto para las imágenes de artículos.

CreateArticleRequest.php (línea 50)

```
if($this->hasFile('article_image')){  
    $picName = substr(time(), 0, -1) . "-" . Str::slug($this->article_name) .  
    "." . $this->file('article_image')->extension();  
  
    $articleDirectory = storage_path() . '/app/public/img/articles/';  
    if (!file_exists($articleDirectory)) {  
        mkdir($articleDirectory, 0755);  
    }  
    $picPath = $articleDirectory . $picName;  
  
    Image::make($this->file('article_image'))  
        ->resize(1024, null, function ($constraint){  
            $constraint->aspectRatio();  
        })  
        ->save($picPath);  
}
```

Por medio de la instancia `Intervention\Image\Facades\Image` se crea mediante el método `make()` la imagen a la que le pasamos el archivo de imagen como parámetro. A partir de este podemos concatenar diversos métodos según el propósito deseado en nuestro caso queremos ajustar la resolución para lo que usaremos el método `resize()`. En este caso nos interesa mantener la relación de aspecto de la imagen por lo que determinamos únicamente una dimensión de la imagen (1024 píxeles) y mediante una función como tercer parámetro determinamos que queremos mantener la relación de aspecto llamando al método `aspectRatio()`. Finalmente guardamos el resultado haciendo

uso del método **save()** al cual indicaremos como parámetro la ruta absoluta de la imagen que hemos definido con anterioridad.

De igual forma codificamos la subida de imágenes para las categorías.

CreateCategoryRequest.php (línea 37)

```
if($this->hasFile('category_image')){  
    $picName = substr(time(), 0, -1) . "-" . Str::slug($this->category_name) .  
    "." . $this->file('category_image')->extension();  
  
    $categoryDirectory = storage_path() . '/app/public/img/categories/';  
    if (!file_exists($categoryDirectory)) {  
        mkdir($categoryDirectory, 0755);  
    }  
    $picPath = $categoryDirectory . $picName;  
  
    Image::make($this->file('category_image'))  
        ->resize(1024, null, function ($constraint){  
            $constraint->aspectRatio();  
        })  
        ->save($picPath);  
}
```

Para incluir la subida y manipulación de imágenes en los test hemos hecho uso de Illuminate\Http\UploadedFile que nos permite simular la subida de un fichero y realizar las pruebas necesarias.

CreateArticleTest.php (línea 45)

```
$file = UploadedFile::fake()->image('image.jpg', 100, 100);
```

Definimos una imagen con un nombre de fichero y una resolución de 100 x 100 píxeles.

Realizamos la petición post y enviamos el fichero de prueba.

CreateArticleTest.php (línea 47)

```
$this->actingAs($this->getAdmin())->from(route('articles.create'))  
->post(route('articles.store'), $this->withData([  
    'article_category' => $category->id,  
    'article_image' => $file,  
]))->assertRedirect(route('articles.index'));
```

Y comprobamos mediante el siguiente código tanto que se ha almacenado correctamente en su ruta como que la imagen ha sido dimensionada adecuadamente usando los métodos de PHP **file_exists()** y **getimagesize()**.

CreateArticleTest.php (línea 74)

```
$this->assertTrue(file_exists( storage_path() . '/app/public/img/articles/' . $article->  
image));  
        $this->assertTrue( getimagesize(storage_path() . '/app/public/img/articles/' . $article  
->image)[0] == 1024);
```

Livewire

Esta tecnología ha sido utilizada principalmente en aquellas partes de la aplicación destinadas al cliente final o al operario encargado de la gestión de los pedidos. Livewire permite dotar de responsividad a los elementos de la página generando una experiencia de uso más agradable para los usuarios. A continuación veremos cómo se ha realizado la implementación en las distintas vistas de la página.

Gestión de pedidos

Para facilitar la labor en la gestión de los pedidos se ha optado, al igual que en toda la web destinada al usuario, en el uso de livewire, para que esta pueda ser realizada mediante un dispositivo móvil como un teléfono o una tablet con comodidad y de forma intuitiva. Para este componente de liveware se ha optado por una disposición master-detail donde las órdenes seleccionadas muestran un detalle de las mismas y se puedan realizar las operaciones sobre estos. (*Ver manual de usuario*)

La vista está compuesta de dos componentes de livewire *OrderManageSideBar* y *OrderManageDetailView*. En la primera se muestran las entradas y en la segunda los detalles del mismo. En esta primera vista además se incluyen los filtros de búsqueda como los listeners que servirán para interconectar ambos componentes entre sí.

Pasamos ahora a ver con más detalle la codificación de ambas clases.

—

OrderManageSideBar

```

class OrderManageSideBar extends Component
{
    public $select;
    public $paginate = 10;
    public $search;
    public $orderStatus;
    public $daily = true;

    protected $queryString = [
        'search' => ['except' => ''],
        'orderStatus' => ['except' => 'all'],
    ];

    protected $listeners = [
        'loadMore' => 'loadMore',
        'orderSelected' => 'orderSelected'
    ];

    public function orderSelected($id){
        $this->select = $id;
    }

    public function loadMore()
    {
        $this->paginate += 10;
    }

    public function render()
    {
        $filters = [
            'search' => $this->search,
            'orderStatus' => $this->orderStatus,
            'dailyOrders' => $this->daily
        ];

        $orders = Order::query()
            ->with('user', 'articles')
            ->applyFilters($filters)
            ->orderBy('created_at')
            ->paginate($this->paginate);

        return view('livewire.order-manage-side-bar',
            ['orders' => $orders]);
    }
}

```

OrderManageDetailView

```

class OrderManageDetailView extends Component
{
    public $order;
    public $orderId;

    protected $listeners = [
        'orderSelected' => 'setOrderId'
    ];

    public function setOrderId($id)
    {
        $this->orderId = $id;
    }

    public function setOrderAsNotCollected($id){
        $order = Order::query()
            ->where('id', $id)
            ->where('order_status', 'pendiente');
        $order->update([
            'order_status' => 'no_recogido'
        ]);
    }

    public function setOrderAsCollected($id){
        $order = Order::query()
            ->where('id', $id)
            ->where('order_status', 'pendiente');
        $order->update([
            'order_status' => 'recogido',
            'payment_status' => 'ya_pagado',
            'collected_date' => now()
        ]);
    }

    public function render()
    {
        $this->order = Order::query()
            ->with('user', 'articles')
            ->whereId($this->orderId)
            ->first();

        return view('livewire.order-manage-detail-view');
    }
}

```

En `OrderManageSideBar` por orden de aparición podemos observar las variables públicas que usará nuestro componente y que serán compartidas en la vista.

La variable protegida \$queryString es la que livewire nos facilita para determinar aquellos valores que no se mostrarán en la url como query strings, en este caso concreto no queremos que el parámetro de la url search se muestre cuando su valor sea cadena vacía.

La siguiente variable de livewire nos permite definir los listeners y asociarlos al método que deseamos ejecutar cuando se reciba una llamada. En este caso llaman a los métodos **loadMore()** el cual detallaremos más adelante y al método **orderSelect()** que determinará cuándo una orden en el listado ha sido seleccionada.

order-manage-side-bar.blade.php (línea 30)

```
<li wire:click="$emit('orderSelected', {{$row->id}})" class="flex items-center flex-auto py-5 px-2 md:px-5 transition duration-500 border-l-8 cursor-pointer {{$select==$row->id ? 'main-color-blue-bg text-white selected':'hover:bg-yellow-100'}}"
```

Por medio de wire:click podemos realizar un emit que dispare el método e indique el id al cual hace referencia.

```
$filters = [
    'search' => $this->search,
    'orderStatus' => $this->orderStatus,
    'dailyOrders' => $this->daily
];

$orders = Order::query()
    ->with('user', 'articles')
    ->applyFilters($filters)
    ->orderBy('created_at')
    ->paginate($this->paginate);

return view('livewire.order-manage-side-bar',
    ['orders' => $orders]);
```

Finalmente en el método render llamamos a la vista pasándole como parámetro las órdenes obtenidas por la siguiente consulta donde adjuntamos los filtros como parámetro directamente al método **apply Filters()**.

Como comentábamos antes el método **loadMore()** se encarga de hacer una carga dinámica de las órdenes del panel de órdenes, para ello se incrementa en 10 su paginación cuando el scroll del mismo panel llegue al final. Para esta implementación se ha realizado la siguiente codificación en la vista

order-manage-side-bar.blade.php (línea 52)

```
@if($orders->hasMorePages())
    <div x-data="{
        observe () {
            let observer = new IntersectionObserver((entries) => {
                entries.forEach(entry => {
                    if (entry.isIntersecting) {
                        @this.call('loadMore')
                    }
                })
            }, {
                root: null
            })
            observer.observe(this.$el)
        }
    }"
        x-init="observe"
        @include('shared._loading')
    </div>
@endif
```

Mientras queden más páginas para mostrar el último elemento en este caso este div mediante alpine genera un observer que lanzará mediante `@this.call('loadMore')` la recarga del componente con la paginación actualizada.

En *OrderManageDetailView* por orden de aparición vemos de nuevo los listeners que en este caso determinan el id de la orden a la que referencian y los métodos relativos a las acciones a tomar, siendo en este caso los que marcan si un pedido ha sido recogido como se preveía o si por el contrario no se ha recogido y se dispone una sanción a dicho usuario

Tienda (Página principal)

La página principal de la aplicación se compone de un componente laravel que se encarga tanto de mostrar un grid con los distintos productos que se ofrecen y filtrando

tanto por categorías, nombre o otros atributos como ingredientes o alérgenos. Por otra parte se ha implementado un modal dentro del mismo para mostrar dinámicamente los detalles del producto. Pasamos ahora a revisar su codificación

```
class ShopProducts extends Component
{
    public $search;
    public $category;
    public $veg;
    public $allergy;

    public $selectedProduct;
    public $showShop = '';
    public $productModal = 'hidden';

    public $paginate = 10;

    protected $queryString = [
        'search' => ['except' => ''],
        'category' => ['except' => ''],
        'veg' => ['except' => ''],
        'allergy' => ['except' => ''],
    ];

    public function openProductModal($id)
    {
        $this->selectedProduct = $id;
        $this->productModal = true;
        $this->showShop = 'hidden';
    }

    public function closeProductModal()
    {
        $this->selectedProduct = null;
        $this->productModal = 'hidden';
        $this->showShop = '';
    }
}
```

Como vemos al igual que en componente de órdenes se disponen de variables para los filtrados y queryString para determinar en qué casos no queremos que se muestren parámetros en la url. Los métodos **openProductModal()** y **closeProductModal()** son los encargados de controlar la apertura y cierre de los detalles de un producto.

```
<div wire:click.stop="openProductModal('{{{$article->id}}})"
```

shop-products.blade.php (línea 93)

```

@if($selectedProduct)
    @php $detailProduct = $articles->filter(function($item) use($selectedProduct) {return $item->id == $selectedProduct;})->
first() @endphp
    <div class="{{ $productModal }} min-h-screen flex items-center justify-center px-4 container px-5 py-6 mx-auto">
        <section>
            <div class="flex items-center justify-center px-4">
                <div class="max-w-5xl bg-transparent backdrop-filter w-full rounded-lg shadow-xl">
                    <div class="flex flex-col md:flex-row max-h-md mb-5 bg-transparent shadow-lg rounded-t-lg overflow-hidden">
                        <div class="w-full md:w-1/3 bg-cover" style="background-image: url('{{ $detailProduct->image? asset('storage/img/articles/'. $detailProduct->image): URL::asset('img/no_picture.jpg')}})"/>
                    <div class="image-fade w-full md:w-2/3 p-4 backdrop-filter">
                        <h1 class="main-color-blue-text font-bold text-2xl">{{ $detailProduct->name }}{{ $detailProduct->category->name }}

```

Si un producto es seleccionado se filtra por su id dentro del array de productos y la visibilidad del modal deja de ser hidden. Cuando el modal se cierra se procede al recorrido inverso.

Siguiendo con el componente shop

```
public function render()
{
    $filters = [
        'search' => $this->search,
        'category' => $this->category,
        'veg' => ($this->veg == true)? 'veg': null,
        'allergy' => ($this->allergy == true)? 'nonallergy': null,
    ];

    $cart = Cart::content();
    $articles = Article::query()
        ->with('nutrition', 'category')
        ->where('stock', '>', 0)
        ->applyFilters($filters)
        ->orderBy('name')
        ->paginate($this->paginate);

    return view('livewire.shop-products',
        [
            'cart' => $cart,
            'articles' => $articles,
            'categories' => Category::query()
                ->with('articles')
                ->whereHas('articles')
                ->orderBy('name', 'ASC')->get(),
        ]);
}
```

Se determinan los filtros y sus valores, se obtiene la información del carrito y se traen los artículos aplicando dichos filtros. Siempre mostrando aquellos productos de los que se tiene stock. Además se traen todas aquellas categorías que tengan al menos un artículo para ser mostradas en los filtros. En lo referente a la paginación de los artículos se sigue la misma codificación que con los pedidos mediante el método **loadMore()**;

Historial del usuario

Para mostrar de forma simple los pedidos realizados por un usuario y todo lo relacionado con estos hacemos uso de otro componente de livewire. A continuación mostramos la codificación;

OrderUserHistory

```
class OrderUserHistory extends Component
{
    protected $selectedOrder;
    protected $orderModal = 'hidden';

    public function openOrderModal($id)
    {
        $this->selectedOrder = $id;
        $this->orderModal = '';
    }

    public function closeOrderModal()
    {
        $this->selectedOrder = null;
        $this->orderModal = 'hidden';
    }

    public function render()
    {
        $orders = Order::query()
            ->with('user', 'articles')
            ->where('user_id', auth()->user()->id)
            ->orderBy('created_at', 'DESC')
            ->get();

        return view('livewire.order-user-history',
        [
            'orders' => $orders,
            'orderModal' => $this->orderModal,
            'selectedOrder' => $this->selectedOrder
        ]);
    }
}
```

La estructura es similar a otros componentes antes descritos. En primer lugar se describen las variables y métodos que controlan la apertura y cierre de los modales con la información de los pedidos seleccionados.

En el método render se hace una búsqueda de todos los pedidos del usuario. El usuario en este caso se obtiene del método **auth()** y a través de este obtenemos su identificador.

Desde la consulta se ordenan por fecha descendente, mostrando primero aquellas más recientes mientras que en la vista las distintas órdenes se filtran por el estado del pedido como se muestra a continuación.

order-user-history.blade.php

```
<div class="px-12">
    <h2 class="text-4xl mb-5 text-right pt-5 main-color-blue-text dark:main-color-yellow-text transition duration-500">
        {{trans('orders.userHistory.title')}}</h2>
        <div class="sm:relative">
            @if($orders->isNotEmpty())
                <ul class=
                    "rounded-md shadow-md bg-transparent absolute left-0 right-0 -bottom-18 mt-3 py-3">
                    <li class=
                        "text-base md:text-lg uppercase border-b border-gray border-solid py-2 px-2 md:px-5 mb-2 main-color-blue-text dark:main-color-yellow-text transition duration-500"
                    >
                        {{trans('orders.userHistory.currentOrders')}}
                    @foreach($orders->filter(function($item) {return $item->order_status == 'pendiente';})->all() as $row)
                        @include('orders.user_history._userOrderRow', ['color' => "blue", 'hover' => "yellow", 'orderStatus'=> null])
                    @endforeach
                    <li class=
                        "text-base md:text-lg uppercase border-b border-gray border-solid py-2 px-2 md:px-5 mb-2 main-color-blue-text dark:main-color-yellow-text transition duration-500"
                    >
                        {{trans('orders.userHistory.processedOrders')}}
                    @foreach($orders->filter(function($item) {return $item->order_status == 'recogido';})->all() as $row)
                        @include('orders.user_history._userOrderRow',['color' => "green",'hover' => "blue",'orderStatus'=>(".trans('orders.manage.collected').")])
                    @endforeach
                    <li class=
                        "text-base md:text-lg uppercase border-b border-gray border-solid py-2 px-2 md:px-5 mb-2 main-color-blue-text dark:main-color-yellow-text transition duration-500"
                    >
                        {{trans('orders.userHistory.canceledOrders')}}
                    @foreach($orders->filter(function($item) {return $item->order_status == 'cancelado';})->all() as $row)
                        @include('orders.user_history._userOrderRow',['color' => "red", 'hover' => "blue", 'orderStatus'=>(".trans('orders.manage.cancel').")])
                    @endforeach
                    <li class=
                        "text-base md:text-lg uppercase border-b border-gray border-solid py-2 px-2 md:px-5 mb-2 main-color-blue-text dark:main-color-yellow-text transition duration-500"
                    >
                        {{trans('orders.userHistory.notCollectedOrders')}}
                    @foreach($orders->filter(function($item) {return $item->order_status == 'no_recogido';})->all() as $row)
                        @include('orders.user_history._userOrderRow',['color' => "yellow", 'hover' => "red", 'orderStatus'=>(".trans('orders.manage.notCollected').")])
                    @endforeach
                </ul>
            @else
                <ul class="rounded-md shadow-md bg-white absolute left-0 right-0 -bottom-18 mt-3 p-3">
                    <li class="grid grid-cols-10 gap-4 justify-center items-center px-4 py-2 rounded-lg">
                        <div class="col-start-2 col-end-11 pl-8 border-1-2 border-solid border-gray">
                            <h3 class="text-gray-900 text-md font-semibold">
                                {{trans('orders.userHistory.noOrdersTitle'))}}</h3>
                                <p class="text-gray-600 mt-1 font-regular text-sm">
                                    {{trans('orders.userHistory.noOrdersAbout'))}}
                                </p>
                            </div>
                        </li>
                    </ul>
                @endif
            </div>
        </div>
    </div>
```

Cada elemento a su vez es una vista.blade (_userOrderRow.blade.php) que recibe un array con los parámetros a mostrar y abre el modal mediante la instrucción wire:click

```
<li wire:click="openOrderModal('{{ $row->id }}')>
    class=
    grid grid-cols-10 gap-4 justify-center items-center cursor-pointer px-4 py-2 rounded-lg hover:bg-{{ $hover }}-100">
        <div class="flex justify-center items-center">
            <svg class="h-20 w-20 text-{{ $color }}-600" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke="currentColor">
                <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M20 7l-8-4-8 4m16 0l-8 4m8-4v10l-8 4m0-10L4 7m8 4v10M4 7v10l8 4" />
            </svg>
        </div>
        <div class="col-start-2 col-end-11 pl-2 md:pl-8 border-l-2 border-solid border-gray">
            <h3 class="main-color-blue-text dark:main-color-yellow-text transition duration-500 text-sm md:text-lg font-semibold">PEDIDO: {{ substr($row->order_code, 0, 4) }} {{ $orderStatus ?? '' }}</h3>
            <div class="text-xs md:text-lg mt-2 md:mt-0 italic main-color-blue-text dark:main-color-yellow-text transition duration-500">Total del pedido: ({{ $row->articles->count() }} Artículos)</div>
            @php $total_price = 0 @endphp
            <span class="text-xs md:text-lg italic main-color-blue-text dark:main-color-yellow-text transition duration-500">
                Productos:
                @foreach($row->articles as $article)
                    {{ $article->name }} x {{ $article->pivot->quantity }} ({{ number_format($article->pivot->quantity * $article->discounted_price, 2) }} €)
                @if(!$loop->last)
                    ,
                @endif
                @php $total_price += ($article->pivot->quantity) * ($article->discounted_price) @endphp
                @endforeach
            </span>
            <div class="text-xs md:text-lg italic main-color-blue-text dark:main-color-yellow-text transition duration-500">{{ $row->payment_status == 'ya_pagado' ? 'Pagado' : 'Sin pagar' }}</div>
            <span class="text-xs md:text-lg italic main-color-blue-text dark:main-color-yellow-text transition duration-500">Importe total: €{{ number_format($total_price, 2) }}</span>
            <div class="text-xs md:text-lg main-color-blue-text dark:main-color-yellow-text transition duration-500 text-right">{{ $row->created_at->diffForHumans() }}</div>
        </div>
    </li>
```

Carrito (bumbummen99)

Para el carrito hemos creado otro componente de livewire con similar lógica a los vistos anteriormente añadiendo además una librería para la gestión del carrito *bumbummen99*.

Esta librería nos permite trabajar más fácilmente los artículos que el usuario añade a su cesta

```

class CartComp extends Component
{
    public function render()
    {
        $cart = Cart::content();
        return view('livewire.cart-comp', compact('cart'));
    }

    public function lessQty($rowId)
    {
        $article= Cart::get($rowId);
        if($article->qty != 1){
            $newQty = $article->qty - 1;
            Cart::update($rowId,$newQty);
        }
    }

    public function moreQty($rowId)
    {
        $article= Cart::get($rowId);
        $newQty = $article->qty + 1;
        Cart::update($rowId,$newQty);
    }

    public function delete($rowId)
    {
        Cart::remove($rowId);
    }
}

```

El componente se compone de dos métodos para el incremento y el decremento de los artículos haciendo uso del método **get()** que nos proporciona la librería. Además de métodos para actualizar y eliminar los productos del carrito.

Test para livewire

Al igual que en resto del desarrollo de la aplicación se han llevado a cabo distintas pruebas que corroboren su correcto funcionamiento. Para esta tarea livewire nos provee de una serie de métodos con los que poder realizar pruebas sobre sus componentes ya sean llamadas a métodos, establecer el valor de las variables, simular emits entre otros.

Veremos a continuación los distintos casos de uso que se nos han presentado al momento de implementar livewire en la aplicación y cómo se han codificado sus correspondientes pruebas.

FilterManageOrdersTest.php (línea 26)

```
/** @test */
public function it_filters_the_order_status()
{
    $this->actingAs($this->getAdmin());

    $userPepe = $this->createUser('Pepe López');

    $collected = Order::factory()->collected()->create(['user_id' => $userPepe->id,]);
    $readyToCollect = Order::factory()->readyToCollect()->create(['user_id' => $userPepe->id,]);
    $notCollected = Order::factory()->notCollected()->create(['user_id' => $userPepe->id,]);

    Livewire::test(OrderManageSidebar::class)->set('orderStatus', 'recogido')->assertViewHas('orders', function
    ($orders) use ($collected, $readyToCollect, $notCollected){
        return (true == $orders->contains($collected))
            && (false == $orders->contains($readyToCollect))
            && (false == $orders->contains($notCollected));
    });
}
```

En esta prueba correspondiente al filtrado de órdenes podemos ver cómo a partir de **Livewire::test()** podemos asignar valor a sus variables internas. Para ello pasamos como argumento el nombre de nuestro componente seguido de ::class para acceder al mismo y a continuación con su método set establecemos clave y valor a actualizar. Al igual que en pruebas ordinarias livewire trae un método **assertViewHas()** con el que comprobamos que la vista contenga o no una colección determinada.

ManageOrdersTest.php (línea 65)

```
$order = Order::factory()->readyToCollect()->notPayedYet()->create([
    'id' => '1',
    'user_id' => $userPepe->id,
    'order_code' => 'R23200000',
]);
Livewire::test('order-manage-detail-view')->emit('orderSelected', $order->id)
    ->assertSeeInOrder([
        'Pepe López',
        'pepe@mail.com',
        '¡RECOGIDO!',
        'REPORTAR',
        'Código pedido: R232',
        'Pendiente',
        'Sin pagar',
    ])->call('setOrderAsCollected', '1')
    ->assertSeeInOrder([
        'Pepe López',
        'pepe@mail.com',
        'RECOGIDO',
        'Código pedido: R232',
        'Recogido',
        'Pagado',
    ]);
});
```

Este extracto pertenece a una prueba encargada de comprobar que el contenido seleccionado en el componente sideBar se refleja en detailsView y desde él se ejecuta la acción de marcar pedido como recogido. De igual forma desde el método **test()** que nos provee livewire llamamos esta vez a **emit()** para que simule la sección de un pedido, una vez seleccionado comprobamos que los cambios se reflejan en el orden deseado.

Para simular que el administrador marca el pedido recurrimos al método **call()** y pasamos a este el nombre del método que queremos simular y el parámetro de ser necesario. Una vez ejecutado podemos revisar con **asssertSeeInOrder()** si la vista muestra el contenido acorde a la acción tomada.

ShowManageOrdersListTest.php (línea 62)

```
$response = $this->get(route('orders.manage'));
$response->assertStatus(200)
    ->assertSeeLivewire('order-manage-side-bar')
    ->assertSeeLivewire('order-manage-detail-view');
$response->assertSeeInOrder([
    'border-red-600',
    'PEDIDO: A100',
    'text-red-300',
    'Pepe López',
    '(2 Artículos)',
    'hace 35 minutos',
    'PEDIDO: B340 (RECOGIDO)',
    'green-600',
    'Juan López',
    '(1 Artículos)',
    'hace 1 segundo',
]);
});
```

Podemos además comprobar la presencia de componentes enteros usando **assertSeeLivewire()** directamente desde un petición http y pasándole como parámetro al método el nombre del componente, en este caso su vista asociada.

ShowManageOrdersListTest.php (linea 189)

```
Livewire::test('order-manage-detail-view')->emit('orderSelected', $firstOrder->id)
->assertSeeInOrder([
    'Pepe López',
    'pepe@mail.com',
    '¡RECOGIDO!',
    'REPORTAR',
    'Código pedido: R232',
    'Pendiente',
    'Sin pagar',
    $firstOrderDate->toDayDateTimeString(),
])->assertDontSee([
    'Juan López',
    'juan@mail.com',
    'Código pedido: B445',
    'Recogido',
    'Pagado',
    $secondOrderDate->toDateTimeString()
])->assertSeeHtml('<span class="mr-2">¡RECOGIDO!</span>')
->assertSeeHtml('<span class="mr-2">REPORTAR</span>')
->assertDontSeeHtml('<p class="font-semibold text-xl text-green-400"> RECOGIDO</p>');
```

De ser necesario livewire nos permite además comprobar la existencia de etiquetas completas dentro de nuestra vista, para ello se hace uso del método **assertSeeHtml()**.

Laravel Breeze. Sistema de inicio de sesión

Laravel Breeze es una implementación minimalista y sencilla de todas las características de autenticación de Laravel, incluyendo el login, registro, el restablecimiento y confirmación de la contraseña y la verificación por correo electrónico. Breeze viene con una serie de vistas por defecto y se componen de simples plantillas Blade estilizadas con Tailwind CSS, las cuales se han reutilizado dentro de lo posible para otras funcionalidades en nuestro proyecto.

Breeze proporciona un cómodo punto de partida para comenzar una nueva aplicación Laravel con autenticación de usuarios, y también es una gran opción para los proyectos que planean llevar las plantillas **BLADE** al siguiente nivel con Laravel **LIVEWIRE**.

Breeze ha sido instalado en nuestro proyecto gracias a **COMPOSER**, corriendo el siguiente comando en nuestro espacio de trabajo: ‘*composer require laravel/breeze --dev*’.

Ya instalado, podemos observar que el proceso ha creado una serie de secciones nuevas en nuestro proyecto:

- Un cómputo de controladores que registran y manejan las solicitudes de inicio de sesión, así como ‘Requests’, funciones que se encargan de validar y procesar los datos a partir de un formulario a través de las vistas que nos ofrece Breeze.
- Vistas maquetadas con componentes de **BLADE** y **LIVEWIRE**. Estas vistas vienen acompañadas con un ‘Middleware’, capaz de controlar el acceso a su renderizado si el usuario está autenticado o no. Este ‘Middleware’ lo podemos encontrar en la carpeta ‘app/http/middleware/Authenticate’:

```

<?php

namespace App\Http\Middleware;

use Illuminate\Auth\Middleware\Authenticate as Middleware;

class Authenticate extends Middleware
{
    /**
     * Get the path the user should be redirected to when they are not authenticated.
     *
     * @param \Illuminate\Http\Request $request
     * @return string|null
     */
    protected function redirectTo($request)
    {
        if (! $request->expectsJson()) {
            return route('login');
        }
    }
}

```

- Breeze nos deja el trabajo ‘masticado’ para que nuestro sistema de autenticado sea prácticamente automático, por lo que también nos provee de tests que demuestran el correcto funcionamiento de esta librería. Estos tests, controlan la entrada de datos por inicio de una sesión, registro, verificación de emails y recuperación y confirmación de contraseñas. No hemos tenido que manipular mucho el código de esta sección, salvo por una circunstancia. La librería **LARATRUST** requería la emulación de un inicio de sesión de un usuario con un ‘rol’ específico para que los tests no tuvieran huecos vacíos. He aquí una captura de los requisitos a añadir, mediante un test de ejemplo en cada prueba, para que estas pasaran junto con las nuevas funciones que ofrece **LARATRUST**, que más adelante explicaremos cómo se complementan con **BREEZE**:

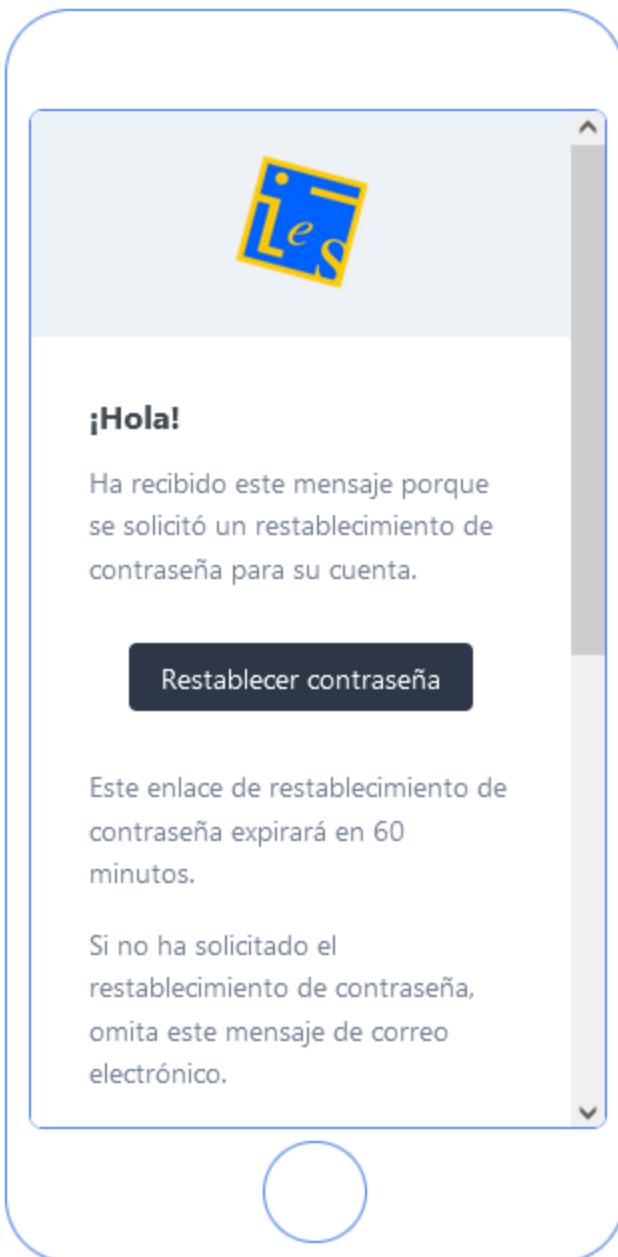
```
public function test_email_verification_screen_can_be_rendered()
{
    $nre = Nre::factory()->create();
    $user = User::factory()->create([
        'nre_id' => $nre->id,
        'email_verified_at' => null,
    ]);

    $response = $this->actingAs($user)->get('/verify-email');
```

Bien, una vez tengamos todo a punto, solo nos falta establecer un servidor de correo donde poder gestionar la entrada y salida de correos al usuario. Laravel provee multitud de configuraciones para una gran variedad de servidores. El que hemos usado nosotros ha sido '*Mailtrap.io*' junto con la configuración 'smtp'. En el archivo .env de nuestro proyecto, establecemos el servidor mail de la siguiente manera:

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=cc6a49b4223e97
MAIL_PASSWORD=c89ff2a9b48ad5
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=mail_man@mail.com
```

Encontrándonos '*logueados*' en '*mailtrap.io*' podemos visualizar los correos que llegan, así como su '%' de compatibilidad en navegadores de correo móviles, tablets y de escritorio. He aquí una captura de la versión final.



Y una captura donde se comprueba la compatibilidad total con clientes de correo:

The screenshot shows a Mailtrack.io report for an email sent from Cantina IES Cierva. The report includes a search bar and navigation links for 'HTML', 'HTML Source', 'Text', 'Raw', 'Spam Analysis', 'HTML Check' (which is active), and 'Tech Info'. The main content area displays a large green circle with '83.8%' labeled 'MARKET SUPPORT'. Below this are two tables: one for 'Clients' and another for 'Found on lines: 10, 20'. The 'Clients' table lists various email clients and their usage percentages. The 'Found on lines' table lists specific client and device combinations.

| Clients | Count |
|-----------------------------|-------|
| Gmail Android | 1 |
| Gmail Desktop Webmail | 1 |
| Gmail iOS | 1 |
| Gmail Mobile Webmail | 1 |
| Outlook Android | 1 |
| Outlook iOS | 1 |
| Outlook Outlook.com | 1 |
| Outlook Windows | 1 |
| Outlook Windows 10 Mail | 1 |
| Yahoo! Mail Android | 1 |
| Yahoo! Mail Desktop Webmail | 1 |
| Yahoo! Mail iOS | 1 |
| AOL Desktop Webmail | 1 |
| AOL iOS | 1 |
| Mail.ru Desktop Webmail | 1 |
| Mozilla Thunderbird macOS | 6 |
| ProtonMail Desktop Webmail | 1 |
| ProtonMail iOS | 1 |
| SFR Android | 1 |
| SFR iOS | 1 |

| Found on lines | Count |
|----------------|-------|
| 10 | 20 |

Si se desea acceder y visualizar de primera mano el proceso de envío y recepción de correo, siéntase libre de iniciar sesión en '[Mailtrack.io](#)' con las siguientes credenciales:

- Correo electrónico: *togaloj575@svlog.com*
- Contraseña: *laravelcantina*

Laratrust. Gestión de permisos de usuarios en nuestro proyecto.

Funcionamiento interno de laratrust:

Internamente **LARATRUST** crea tablas relacionadas en nuestra base de datos utilizando sus propios seeders. Las tablas son las siguientes:

role: Contiene los roles creados en nuestra aplicación, en nuestro caso 3:

| | | | id | name | display_name | description | created_at | updated_at |
|--------------------------|--|--|--|-------------|---------------------|--------------------|-------------------|---------------------|
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 1 | administrator | Administrator | Administrator | 2021-12-13 18:43:41 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 2 | employee | Employee | Employee | 2021-12-13 18:43:41 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 3 | user | User | User | 2021-12-13 18:43:42 |

role_user: Es una tabla intermedia que almacena que rol tiene cada usuario, es la que se modifica cada vez que utilizamos `$user->attachRole('nombre del rol')`

Nuestra aplicación se basa en el uso de esas tablas, junto a su middleware (explicado posteriormente), pero genera otras dos tablas con un funcionamiento parecido para permisos individuales.

permissions: Contiene la definición de diferentes permisos:

| | <input type="button" value="←"/> <input type="button" value="→"/> | <input type="button" value="id"/> | <input type="button" value="name"/> | <input type="button" value="display_name"/> | <input type="button" value="description"/> | <input type="button" value="created_at"/> | <input type="button" value="updated_at"/> | |
|--------------------------|---|---------------------------------------|---------------------------------------|---|--|---|---|---------------------|
| <input type="checkbox"/> | <input type="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 1 administrator | Administrator | Administrator | 2021-12-13 18:43:41 | 2021-12-13 18:43:41 |
| <input type="checkbox"/> | <input type="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 2 employee | Employee | Employee | 2021-12-13 18:43:41 | 2021-12-13 18:43:41 |
| <input type="checkbox"/> | <input type="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 3 user | User | User | 2021-12-13 18:43:42 | 2021-12-13 18:43:42 |

permissions_users: La tabla intermedia que asigna estos permisos a usuarios.

permissions_roles: La tabla intermedia que asigna estos permisos a roles.

Estas tablas contienen la información de los roles que definamos y la asignación de permisos a cada rol o usuario.

Además podemos acceder fácilmente a la información utilizando métodos como `$user->role()` para saber el rol del usuario. También nos facilita herramientas para añadir lógica que mejore nuestra seguridad utilizando métodos que comprueben los roles o permisos del usuario, por ejemplo `$user->hasRole('administrador')` devolverá true si el usuario tiene ese rol.

Mostraremos mas adelante como utilizamos estas herramientas en la vista para limitar la visibilidad de los usuarios dependiendo de sus roles.

Uso de **LARATRUST** para control de credenciales en URLs:

Una de las herramientas que nos proporciona **LARATRUST** es un middleware que podemos utilizar en el archivo rutas web.php. Con esto conseguimos filtrar los roles a los que queremos proporcionar acceso a cada URL.

web.php

```
Route::get('/users', [UserController::class, 'index'])->middleware(['role:administrator|employee'])->name('users.index');
    Route::get('/users/newKey', [NreController::class, 'createSpecialUserCode'])->middleware(['role:administrator|employee'])->name(
'nres.createSpecialUserCode');
    Route::delete('/users/delete/{id}', [UserController::class, 'destroy'])->middleware(['role:administrator|employee'])->name(
'users.destroy');
    Route::get('/users/papelera', [UserController::class, 'index'])->middleware(['role:administrator|employee'])->name(
'users.trashed');
    Route::get('/users/{user}/show', [UserController::class, 'show'])->middleware(['role:administrator|employee'])->name(
'users.show');
    Route::get('/users/bann/{id}', [UserController::class, 'bann'])->middleware(['role:administrator|employee'])->name(
'users.bann');
);
    Route::put('/users/{id}/papelera', [UserController::class, 'restore'])->middleware(['role:administrator|employee'])->name(
'users.restore');
    Route::patch('/users/{user}/papelera', [UserController::class, 'trash'])->middleware(['role:administrator|employee'])->name(
'users.trash');
    Route::get('/users/createEmployee', [UserController::class, 'createEmployee'])->middleware(['role:administrator'])->name(
'users.createEmployee');
```

Solo los usuarios con los roles indicados al pueden acceder a las URLs.

Esto hay que tenerlo en cuenta a la hora de realizar test, lo cual hemos hecho utilizando usuarios con los roles que deseamos testear y usando \$this->actingAs(\$user).

UpdateUsersTest.php

```
$this->actingAs($admin)->from(route('users.edit', ['user' => $oldUser]))
    ->put(route('users.update', ['user' => $oldUser]), $this->withData([
        'user_credit' =>10,
    ]));
```

Si lo que se desea es limitar la funcionalidad del usuario ‘logeado’ en nuestra web, podemos hacerlo mediante directivas personalizadas de **BLADE**

```

1  @role('user')      <div class="({ (Auth::user())->banned ? 'user-banned' : '' }) pt-16 pb-1 flex flex-col items-center transition-all duration-500">
2    
3    <span class="font-light py-1 text-sm main-color-blue-text font-bold dark:main-color-yellow-text transition duration-500">{{ (Auth::user())->name }}</span>
4    <span class="font-light py-1 text-2xs main-color-blue-text dark:main-color-yellow-text transition duration-500">Strikes: {{ (Auth::user())->ban_strikes }}</span>
5    <span class="font-light py-1 text-2xs main-color-blue-text dark:main-color-yellow-text transition duration-500">
6      @switch((Auth::user())->ban_strikes)
7        @case(0)
8          |Sigue portándose así de bien!
9          @break
10         @case(1)
11          |Sigue portándose así de bien!
12          @break
13         @case(2)
14          @case(2)
15          |Cuidado! No queremos que te vayas...
16          @break
17         @case(3)
18          |No has dado la taya...
19          @break
20      @endswitch
21    </span>
22    <div class="pt-1 w-full">
23      <meter class="rounded-lg w-full main-color-yellow-text" min="0" max="3" value="{{ (Auth::user())->ban_strikes }}" low="1" high="2" optimum="0"></meter>
24    </div>
25  </div>
26 @endrole

```

Solo aquel usuario logeado con ese rol podrá visualizar el contenido que se muestra en la anterior imagen. De lo contrario, si se posee otro rol, podemos condicionar estas vistas de otra forma, sin la necesidad de directivas **BLADE**:

```

1  @if(Auth::user() && (Auth::user())->isAn('administrator|employee'))
2    <ul>
3      <li>
4        <a class="({ (request()->routeIs('orders.manage')) | (request()->is('manageOrders/*')) ? 'currentActive transform translate-x-3 transition duration-500' : 'transform
5           <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
6             <path d="M20 6H17V4C17 2.89 16.11 2 15 2H7C7.89 2 2.89 7 4V6C2.89 6 2 6.89 2 8V19C2.89 20.11 2.89 21 4 21H20C21.11 21 22 20.11 22 19V8C22 6.89 21.11
7               </svg>
8             <span class="ml-4 font-light text-xs main-color-blue-text uppercase dark:main-color-yellow-text transition duration-500"> Administrar Pedidos</span>
9           </a>
10      </li>
11      <li>
12        <a class="({ (request()->routeIs('users.index')) | (request()->is('users/*')) ? 'currentActive transform translate-x-3 transition duration-500' : 'transform
13           <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
14             <path d="M19 3H14.82C14.4 1.84 13.3 1 12 C10.7 1 9.6 1.84 9.18 3H3C9.3 3 3.9 3 5V19C3 20.1 3.9 21 5 21H9C20.1 21 21 20.1 21 19V5C21 3.9 20.1 3 19
15               </svg>
16             <span class="ml-4 font-light text-xs main-color-blue-text uppercase dark:main-color-yellow-text transition duration-500"> Usuarios</span>
17           </a>
18      </li>
19      <li>
20        <a class="({ (request()->routeIs('articles.index')) | (request()->is('food/*')) ? 'currentActive transform translate-x-3 transition duration-500' : 'transform
21           <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
22             <path d="M8.06 22.99H19.72C28.56 22.99 21.25 22.35 21.35 21.53L23 5.05H18V16.03V5.05H11.06L11.36 7.39C13.07 7.86 14.67 8.71 15.63 9.65C17.07 11.07
23               </svg>
24             <span class="ml-4 font-light text-xs main-color-blue-text uppercase dark:main-color-yellow-text transition duration-500"> Artículos</span>
25           </a>
26      </li>
27    </ul>
28  @else
29    <!-- ... -->

```

LARATRUST comprobará si el rol del usuario autenticado es el de un empleado o el de un administrador mediante la función ‘isAn()’ (existen otras como: ‘isA()’, ‘hasRole()’, ‘isAbleTo()’ o ‘hasPermission()’, que realizan funciones parecidas.), y se adentrará en la condición. Por otra parte, si el usuario autenticado no tuviera dichos roles, el código

mostrado en la anterior imagen no se mostrará, dotando de privacidad de contenido para cada rol que asignemos a los usuarios.

Si por lo general, necesitamos obtener aquellos usuarios que posean un tipo de rol u otro, podemos hacerlo con las queries que nos ofrece **LARATRUST**:

```
$users = User::whereRoleIs('admin')->orWhereRoleIs('regular-user')->get();
```

Se almacenará en `$users` todos aquellos usuarios que posean el rol '`admin`' o el rol '`regular-user`'.

PL/SQL

Con el fin de aligerar la carga de trabajo de la aplicación hemos recurrido a la implementación de triggers y procedimientos almacenados en la base de datos para los campos que tengan una mayor tasa de operaciones como la gestión del stock de los productos o la suspensión de la cuenta en función del número de faltas acumuladas por los usuarios de forma automática.

Para la implementación de estos últimos desde laravel se han realizado a partir de dos migraciones: *create_cantina_triggers.php* y *create_cantina_stored_procedures.php* siendo una para triggers y otra destinada para los procedimientos almacenados

create_cantina_stored_procedures.php (línea 30)

```
DB::unprepared("DROP TRIGGER IF EXISTS update_articles_stock");
DB::unprepared($trigger_update_articles_stock);
```

Dentro del método **up()** llamamos desde la instancia DB a su método **unprepared()** con la que ejecutaremos las instrucciones sql. En este ejemplo un drop inicial en caso de que exista y la variable que contiene el trigger.

create_cantina_stored_procedures.php (línea 58)

```
public function down()
{
    DB::unprepared('DROP TRIGGER IF EXISTS update_articles_stock');
    DB::unprepared("DROP TRIGGER IF EXISTS restore_articles_stock");
}
```

De igual manera que en resto de migraciones definimos el método **down()** con los drops correspondientes. A continuación veremos con más detalle el funcionamiento de estos últimos.

Update articles stock

```
DELIMITER //
CREATE TRIGGER update_articles_stock AFTER INSERT ON article_order
FOR EACH ROW
BEGIN

DECLARE v_order_status VARCHAR(45);
DECLARE v_previous_stock int;

SELECT o.order_status, a.stock
FROM orders o, articles a
WHERE o.id = new.order_id
AND new.article_id = a.id
INTO v_order_status, v_previous_stock;

IF (v_previous_stock-new.quantity) >= 0 THEN
    IF v_order_status ='pendiente' THEN
        UPDATE `articles` SET `stock` = (v_previous_stock-new.quantity), `deleted_at` = NULL WHERE `articles`. `id` = new.article_id;
    END IF;
END IF;
END //
DELIMITER ;
```

El siguiente trigger se encarga de retirar el stock vacante de los productos cuando se realiza una inserción en la tabla de órdenes bajo el estado de pendiente. Evita además que se pueda actualizar un stock por debajo de cero para evitar inconsistencias dentro de la base de datos.

Restore articles stock

```
DELIMITER //
CREATE TRIGGER restore_articles_stock AFTER UPDATE ON orders
FOR EACH ROW
BEGIN
    DECLARE v_order_status VARCHAR(45);

    SELECT o.order_status
    FROM orders o
    WHERE o.id = NEW.id
    INTO v_order_status;

    IF OLD.order_status = 'pendiente' THEN
        IF v_order_status = 'no_recogido' THEN
            CALL restore_stock(NEW.id);
            CALL report_user_order(NEW.user_id);
        ELSEIF v_order_status = 'cancelado' THEN
            CALL restore_stock(NEW.id);
        END IF;
    END IF;

END //
DELIMITER ;
```

Al contrario que el anterior trigger este se encarga de restaurar el stock de todos aquellos productos de una orden que haya sido cancelada o no recogida. Este solo se dispara sobre una actualización a la tabla orders. Se comprueba que su estado anterior fuese pendiente y después vuelve a comprobar el estado con el que ha sido actualizado y llama a los procedimientos que tienen que intervenir. En caso de cancelación del pedido solo se restaura el stock. En el supuesto donde el usuario no recoge un pedido se procede de igual forma pero además se realiza un reporte de este.

Restore stock

```
DELIMITER $$  
CREATE PROCEDURE restore_stock (IN in_order_id int)  
BEGIN  
  
DECLARE v_article_id int;  
DECLARE v_quantity int;  
DECLARE v_current_stock int;  
DECLARE v_end_loop INTEGER DEFAULT 0;  
  
DECLARE cursor_articles_orders CURSOR FOR  
  
    select a.id, a.stock,ao.quantity from article_order ao, orders o, articles a  
    where ao.order_id = o.id  
    and ao.article_id = a.id  
    and ao.order_id = in_order_id;  
  
DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_end_loop = 1;  
  
OPEN cursor_articles_orders;  
articles_orders_loop:  
    LOOP  
        FETCH cursor_articles_orders INTO v_article_id, v_quantity, v_current_stock;  
        IF v_end_loop = 1 THEN  
            LEAVE articles_orders_loop;  
        END IF;  
        UPDATE `articles` SET `stock` = (v_current_stock + v_quantity), `deleted_at` = NULL WHERE `articles`.  
`id` = v_article_id;  
    END LOOP articles_orders_loop;  
  
    CLOSE cursor_articles_orders;  
END$$  
DELIMITER ;
```

Este procedimiento se encarga de restaurar el stock de todos los productos implicados en el pedido. Se realiza un cursor y una consulta inicial de los artículos y su stock dentro del pedido en cuestión. Una vez abierto el cursor por cada iteración se actualizará el valor del stock por cada producto.

Report user order

```
DELIMITER $$  
CREATE PROCEDURE report_user_order (IN in_user_id int)  
BEGIN  
  
DECLARE v_user_reports int;  
  
SELECT ban_strikes FROM users u WHERE u.id = in_user_id INTO v_user_reports;  
  
IF v_user_reports >= 3 THEN  
    UPDATE `users` SET `banned` = '1', `ban_strikes` = '0', `deleted_at` = NULL WHERE `users`.`id` = in_user_id;  
ELSE  
    UPDATE `users` SET `ban_strikes` = (v_user_reports + 1), `deleted_at` = NULL WHERE `users`.`id` = in_user_id  
AND banned = 0;  
END IF;  
  
END$$  
DELIMITER ;
```

Finalmente este procedimiento se encarga de reportar al usuario. En caso de que sus faltas anteriores sean menores al límite establecido sólo se actualizará el campo de strike. Si el usuario continúa acumulando strikes hasta rebasar el límite se procede a su suspensión de la aplicación, actualizando el campo ban a 1 y devolviendo sus strikes a 0 para que en el caso de ser restaurado comience con un historial limpio.

Problemas encontrados

- Cambio de valores por defecto en el archivo de configuración de **TAILWINDCSS** ‘tailwind.config.js’: debido a un problema con Node.js y las versiones de NPM, los cambios que realizáramos en este archivo serían muy limitados y específicos, ya que, a pesar de la gran versatilidad que presenta **TAILWINDCSS** en este ámbito, a la hora de cargar e iniciar el ‘Front-End’, este presentaba multitud de errores Todas las variantes a modificar en el apartado ‘theme’, debían de permanecer en el archivo ‘*/resources/css/app.css/’, de lo contrario, saltaría este error:

```
> development
> mix

✖ Mix
Compiled with some errors in 3.03s

ERROR in ./resources/css/app.css
Module build failed (from ./node_modules/mini-css-extract-plugin/dist/loader.js):
ModuleBuildError: Module build failed (from ./node_modules/postcss-loader/dist/cjs.js):
TypeError: variants.join is not a function
    at wrapWithVariants (/var/www/cantina/node_modules/tailwindcss/lib/util/wrapWithVariants.js:29:22)
    at addUtilities (/var/www/cantina/node_modules/tailwindcss/lib/util/processPlugins.js:86:70)
    at matchUtilities (/var/www/cantina/node_modules/tailwindcss/lib/util/processPlugins.js:148:9)
    at /var/www/cantina/node_modules/tailwindcss/lib/util/createUtilityPlugin.js:30:7
    at /var/www/cantina/node_modules/tailwindcss/lib/util/processPlugins.js:97:5
    at Array.forEach (<anonymous>)
    at _default (/var/www/cantina/node_modules/tailwindcss/lib/util/processPlugins.js:91:11)
    at /var/www/cantina/node_modules/tailwindcss/lib/processTailwindFeatures.js:64:54
    at LazyResult.runOnRoot (/var/www/cantina/node_modules/postcss/lib/lazy-result.js:339:16)
    at LazyResult.runAsync (/var/www/cantina/node_modules/postcss/lib/lazy-result.js:391:26)
    at processResult (/var/www/cantina/node_modules/webpack/lib/NormalModule.js:748:19)
    at /var/www/cantina/node_modules/webpack/lib/NormalModule.js:847:5
    at /var/www/cantina/node_modules/loader-runner/lib/LoaderRunner.js:399:11
    at /var/www/cantina/node_modules/loader-runner/lib/LoaderRunner.js:251:18
    at context.callback (/var/www/cantina/node_modules/loader-runner/lib/LoaderRunner.js:124:13)
    at Object.loader (/var/www/cantina/node_modules/postcss-loader/dist/index.js:142:7)

1 ERROR in child compilations (Use 'stats.children: true' resp. '--stats-children' for more details)
webpack compiled with 2 errors
```

- Alpine Plugins, x-collapse: se quiso tratar este plugin como una alternativa más limpia a la directiva x-show, eliminando la necesidad del atributo x-transition, pudiendo así ser sustituido por x-collapse. Sin embargo, su inicialización en el ‘bundle’ de la aplicación, colapsaba en parte con la inicialización de ALPINE como tal: ‘window.Alpine = Alpine; Alpine.start()’ -> ‘Alpine.plugin(collapse)’ , dando lugar a la inutilización del plugin.

Conclusiones

El objetivo principal de este proyecto era el de crear una aplicación sencilla y práctica para el usuario con la que poder realizar pedidos en la cantina de forma cómoda y enfocada principalmente en los dispositivos móviles. Como resultado de nuestro trabajo hemos generado una aplicación funcional que permite mostrar el catálogo y generar pedidos de forma efectiva.

A lo largo del proyecto algunas ideas o expansiones de la aplicación han ido descartando ya fuera por limitaciones de tiempo o por falta de practicidad dentro de la web y su público objetivo. Algunas como el pago como Stripe o plataformas como Paypal no encajan en el marco de una aplicación de un instituto. En su lugar se optó por un sistema de crédito interno que puede ser recargado en la propia cantina del centro. Por otra parte el modelo de registro adoptado en un principio, basado en la autenticación de los usuarios a partir del nre resultó ser un limitante por ser demasiado restrictivo por lo que para solucionar este supuesto se implementó la generación de tickets equivalentes al campo nre. De esta forma aquel que lo desee puede solicitar uno de estos últimos para la creación de su cuenta. Si bien la solución no es la más efectiva por limitaciones temporales se decidió optar por ella, dejando como mejora a futuro implementar sistemas de autenticación basados en gmail o similares.

Durante el desarrollo de este proyecto hemos ido sorteando diferentes obstáculos como los referenciados en el anterior apartado pero en todo momento hemos procurado mantener el estándar TALL y el desarrollo guiado por pruebas aunque en ciertos momentos del mismo debido a los problemas encontrados, estos test se han realizado de forma posterior ya que a medida que indagamos cada vez más sobre tecnologías, en su mayoría novedosas para nosotros, iban surgiendo cambios o mejores formas de implementar las funcionalidades.

Pese a todo hemos cumplido con los objetivos propuestos inicialmente para este desarrollo y estamos satisfechos con el resultado final, habiendo creado una aplicación sencilla pero efectiva en su cometido que muestre su contenido de forma clara e intuitiva a los usuarios. Dejando una puerta abierta a nuevas funcionalidades si así se requiriese.

Herramientas utilizadas

- Visual Studio Code
- Docker Desktop
- Cmder.exe
- PhpStorm
- Laravel-Idea
- Laradock
- PowerShell
- Dbeaver
- FIGMA
- Git
- Github Desktop
- Composer
- Node.js

Referencias

Fuentes consultadas y referentes visuales.

Laravel: <https://laravel.com/>

Laravel-Livewire: <https://laravel-livewire.com/>

Alpine: <https://alpinejs.dev/>

TailwindCSS default config: '[Configuración por defecto TAILWINDCSS](#)'

Laravel-Breeze: <https://laravel.com/docs/8.x/starter-kits#laravel-breeze>

Laratrust:

<https://laratrust.santigarcor.me/docs/6.x/usage/roles-and-permissions.html#setting-things-up>

Librería del carrito:

<https://github.com/bumbummen99/LaravelShoppingcart>

Generador de datos Faker:

<https://faker.readthedocs.io/en/master/>

Provider de Faker para nombres de comida:

<https://github.com/jzonta/FakerRestaurant>

TailwindCSS Templates: [Wickedtemplates](#)

Intervention Image: <http://image.intervention.io/>

Ideas de diseño: <https://designcourse.com/>

Figma: <https://www.figma.com/>

Heroicons: <https://heroicons.com/>

Mailtrap: <https://mailtrap.io/>

Polacode: <https://marketplace.visualstudio.com/items?itemName=pnp.polacode>