

**Universidad ORT Uruguay**

**Facultad de Ingeniería**

**Bernard Wand Polak**

## **Diseño de Aplicaciones II**

### **Anexo 3 - Diseño del Front-end**

**<https://github.com/ORT-DA2/164660-185720-234059>**

**Santiago Alvarez – N° Estudiante: 185720**

**Juan Castro – N° Estudiante: 164660**

**Marcelo Tilve – N° Estudiante: 234059**

<b>Core UI</b>	<b>3</b>
<b>Angular Date Range Picker</b>	<b>4</b>
<b>Angular QR Code Generator</b>	<b>5</b>
<b>Componentes Custom</b>	<b>5</b>
<b>Servicios Custom</b>	<b>6</b>
<b>Diseño de paths y ruteo</b>	<b>7</b>
<b>Manejo de Roles (Guardas)</b>	<b>7</b>
<b>Uso de Pipes</b>	<b>8</b>

## Core UI

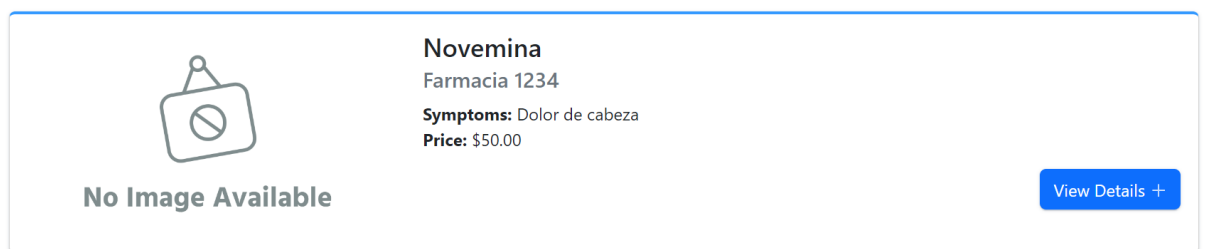
Se tomó la decisión de utilizar Angular [Core UI](#) como librería de componentes, si bien existen librerías similares como lo pueden ser Angular Material, PrimeNG o Nebular, se concluyó que los componentes proporcionados por ésta como también la facilidad para entender la documentación disponible se adaptan a lo necesitado para realizar este obligatorio.

Se utilizó una gran cantidad de módulos de Core UI, algunos de los más destacados son:

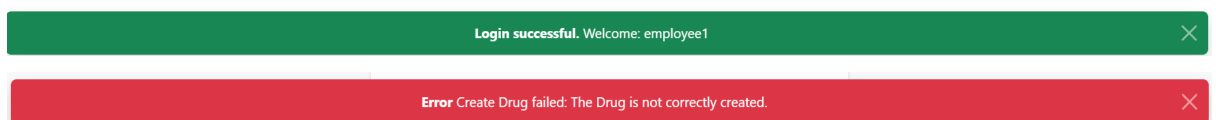
- Badge Module



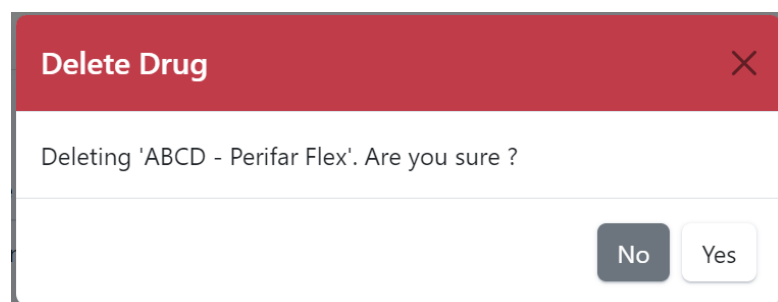
- Card Module






- Toast Module



- Modal Module





- Table Module


#	Drug	Price	Quantity	Total	
1	Novemina	\$50.00	5	\$250.00	
2	Perifar Flex	\$125.00	1	\$125.00	
3	Aspirineta	\$50.00	1	\$50.00	
<b>Total</b>				<b>\$425.00</b>	


- Form Module


### Create Drug
















☐ Prescription





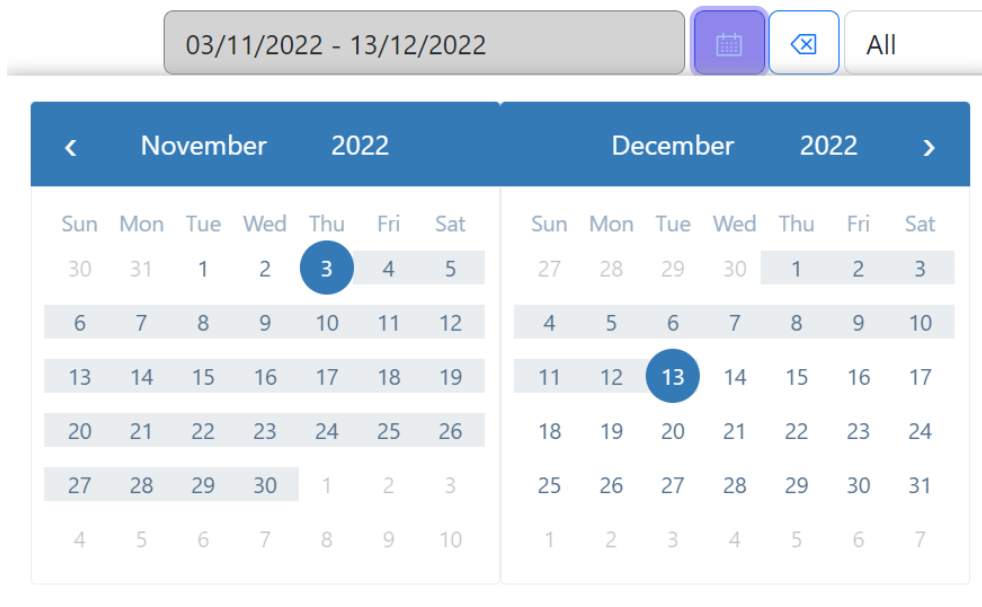
Create Drug

- Icons Module



## Angular Date Range Picker

Se agregó como librería externa el date range picker de [ngx-bootstrap](https://ng-bootstrap.github.io/), que permite importar a nuestra aplicación un datepicker por rango de fechas, lo que en determinados listados es muy práctico a la hora de filtrar.



## Angular QR Code Generator

Se agregó como librería externa [angularx-qrcode](#), que permite generar códigos QR a partir de strings de manera rápida y en el momento. Si bien esto no es un requerimiento, nos pareció didáctico e interesante darle la funcionalidad extra al usuario anónimo de que al momento de rastrear el estado de sus compras (“/home/tracking”) pueda visualizar el código QR de rastreo de sus 5 últimas compras consultadas.

### Tracking

Code


7T14V5UI5Z7JE3RD

Track your purchase 📄

**Buyer:** jose3@gmail.com  
**Date:** 24/10/2022 03:10  
**Total:** \$375.00

Code	Name	Quantity	Unit Price	Total	Pharmacy	Status
XDEA	Novemina	15	\$50.00	\$750.00	Farmacia 1234	Rejected
ABCD	Perifar Flex	3	\$125.00	\$375.00	Farmacia 1234	Approved

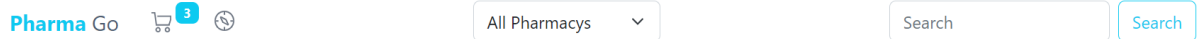
Last 5 tracked codes

#	Tracking Code	QR
1	7T14V5UI5Z7JE3RD	

## Componentes Custom

Se crearon algunos componentes re-utilizables, los mismos se encuentran en la carpeta “custom” y son comunes a varios conjuntos de páginas dentro de la aplicación, un ejemplo de

esto es la barra de header que es común a todas las pantallas a que pueden acceder los usuarios anónimos al realizar una compra.



```
<app-custom-header  
[icons]="true" [select]="true" [search]="true">  
</app-custom-header>
```

El componente tiene la funcionalidad de mostrar u ocultar los elementos del mismo, es decir, en el “/home” se desea que los usuarios puedan filtrar por farmacia y buscar medicamentos. En el checkout de la aplicación (“/home/cart/cho”) se desea no mostrar nada de esto de forma de que el usuario no tenga ningún tipo de distracción (no sacarlo del flujo directo de compra) al momento de finalizar la compra.

Pharma Go

Este tipo de componentes reutilizables también se los puede ver en los footers o en los toast que se muestran en diferentes flujos dentro de la aplicación.

## Servicios Custom

Se crearon 2 servicios que se inyectan dentro de los providers del app.module, estos son:

- StoreManager

El StoreManager se creó con la idea de tener un wrapper del Local Storage, y que en caso de ser necesario y se necesite cambiar, por ejemplo, al Session Storage, el cambio deba hacerse en un solo archivo y no altere la lógica del resto de la aplicación.

- CommonService

El CommonService nace con la necesidad de poder disparar eventos desde un componente hacia otro, para esto se decidió utilizar el [BehaviorSubject](#) de ‘rxjs’ para que luego de una determinada acción de un usuario emitir y estar continuamente escuchando eventos y actualizando la UI.

De esta forma para varias funcionalidades dentro de la aplicación como lo pueden ser actualizar la cantidad de diferentes ítems comprados por un usuario anónimo en su carrito de compras o mostrar los diferentes Toast de éxito o error al realizar una acción al llamar a un servicio del backend.

## Diseño de paths y ruteo

Se decidió que al momento del Login automáticamente y dependiendo del rol del usuario se redirige a los paneles con la funcionalidades específicas de ese usuario, es decir, si el usuario es:

- Administrador: se redirige a **“/admin”** (Administrator Panel) con la página de inicio para este rol donde se muestran sus funcionalidades específicas.
- Employee: se redirige a **“/employee”** (Employee Panel) con la página de inicio para este rol donde se muestran sus funcionalidades específicas.
- Owner: se redirige a **“/owner”** (Owner Panel) con la página de inicio para este rol donde se muestran sus funcionalidades específicas.

Por otro lado, todo tipo de usuario puede acceder a **“/login”**, **“/register”** y a todo el flujo de compra para usuarios Anónimos que inicia en el **“/home”** de PharmaGo y que le permite comprar items de diferentes farmacias.

## Manejo de Roles (Guardas)

Para el manejo de roles se creó la guarda [AuthenticationGuard](#) la misma obtiene la lista de roles que se asigna en el [app-routing](#) a cada ruta dentro de la aplicación.

Ejemplo:

Al momento de hacer el “Login”, se guarda en el Local Storage el objeto devuelto por el backend, que contiene token de autorización, rol y nombre de usuario.

```
{  
  "token": "e9e0e1e9-3812-4eb5-949e-ae92ac931401",  
  "role": "Employee",  
  "userName": "employee1"  
}
```

Luego cuando un usuario desea ingresar a una ruta se verifica el mismo en la guarda, en este caso se envía el rol “Employee” para la ruta **“/employee”** a la guarda.

```
'employee',  
component: EmployeeComponent,  
canActivate: [AuthenticationGuard], data: {roles: ['Employee']}
```

De esta forma dentro del `AuthenticationGuard` se verifica que el rol guardado en el Local Storage coincida con el rol de la ruta a la que se quiere acceder. Si coincide, se devuelve “true” y se permite acceder a la ruta dentro de la aplicación, en caso contrario se devuelve “false” y se re-dirige a “/unauthorized” (401).

En la llamada a todos los servicios se consulta el Local Storage y se obtiene el token de autorización, de esta forma se lo envía en el *Header* : *Authorization* para su posterior validación en el filtro del backend.

## Uso de Pipes

Se utilizaron 2 pipes de Angular para dar formato a los valores a mostrarse en las diferentes pantallas, estos fueron:

- Currency

```
{{ drug.price | currency }}
```

Esto se traduce agregando el signo “\$” delante del monto y formateando el mismo separando decimales con punto (“.”) y miles con coma (“,”).

\$3,375.00

- Date

```
{{ purchase.purchaseDate | date: "dd/MM/yyyy HH:mm" }}
```

Esto genera formatear la fecha devuelta por el backend al formato: día/mes/año horas:minutos.

05/10/2022 10:21

- Uppercase

Esto permite pasar a mayúscula el texto devuelto por el backend.

```
{{ parameter.inputName | uppercase }}
```