Universidad de Nariño.
Ingeniería de Sistemas.
Diplomado de actualización en nuevas tecnologías para el desarrollo de Software.

Código: 22036268
Nombres: Alexander Luna

INFORME TALLER UNIDAD 3 FRONTEND

Iniciar un proyecto en Angular

Inicialmente se crea un nuevo proyecto mediante el comando: *ng new FruverFrontend* aplicando Angular Routing y estilos CSS.

```
PS C:\TallerFruverAlex\Frontend> ng new FruverFrontend
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE FruverFrontend/angular.json (2741 bytes)
CREATE FruverFrontend/package.json (1046 bytes)
CREATE FruverFrontend/README.md (1068 bytes)
CREATE FruverFrontend/tsconfig.json (901 bytes)
CREATE FruverFrontend/.editorconfig (274 bytes)
CREATE FruverFrontend/.gitignore (548 bytes)
```

Se crearán los archivos y carpetas del proyecto base. Se procede a crear los diferentes modelos, servicios y componentes, así como las rutas necesarias para el proyecto.

Crear los modelos en Angular

Se crea la carpeta ‘Models’ dentro de la ruta FruverFrontend/src/app/shared/Models para definir los modelos necesarios para la aplicación.

Crear el modelo ‘ProductoModel’ en el que se definen los siguientes atributos:

Producto.model.ts U

src > app > shared > Models > Producto.model.ts > ...

```ts
export class ProductoModel {
    constructor(

        public idProducto: number,
        public nombre: string,
        public precio: number,
        public detalle: string,
        public tipo: string,
        public imagen: string
```

Crear el modelo 'UsuarioModel', el cual tendrá los siguientes atributos:

src > app > shared > Models > Usuario.model.ts > ...

```ts
export class UsuarioModel {
    constructor(
        public idUsuario: number,
        public email: string,
        public password: string,
        public nombres: string,
        public direccion: string,
        public rol: string
```

Crear el modelo 'CompraModel', el cual tiene los atributos:

src > app > shared > Models > Compra.model.ts > ...

```ts
export class CompraModel {
    constructor(
        public idCompra: number,
        public idUsuario: number,
        public valorTotal: number,
        public fecha: Date,
    ) {
```

Crear el modelo 'ProductoCompraModel':

src > app > shared > Models > ProductoCompra.model.ts > ...

```ts
export class ProductoCompraModel {
    constructor(
        public idProductoCompra: number,
        public idCompra: number,
        public idProducto: number,
        public cantidad: number,
        public valorProductoCompra: number,
    ) {
```

Y un modelo para los productos del carrito de compras: 'ProductoCarritoModel':

```
export class ProductoCarritoModel {
    constructor(
        public idProductoCarrito:number,
        public idUsuario: number,
        public idProducto: number,
        public cantidad: number,
        public valorProductoCarrito: number,
```

Crear los servicios en Angular

Se deben crear los servicios para la aplicación, que servirán para las llamadas a los diferentes métodos del Backend y la gestión del carrito de compras.

Crear el servicio para el modelo 'ProductoModel' mediante: ng generate service shared/Services/producto

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng generate service shared/Services/producto
CREATE src/app/shared/Services/producto.service.spec.ts (367 bytes)
CREATE src/app/shared/Services/producto.service.ts (137 bytes)
```

En el archivo creado se implementan los métodos necesarios para comunicar con Backend.

En la clase del servicio se define la url para establecer conexión con el Backend, esto se hará con cada servicio.

```
export class ProductoService {
  BASE_URL = 'http://localhost:3000';

  constructor(private http: HttpClient) {

  }
```

Luego se agregan los métodos dependiendo de lo que se necesite en la aplicación:

```
obtenerProductos(){
  return this.http.get<ProductoModel[]>(`${this.BASE_URL}/productos`);
}
obtenerProductoId(idProducto:number){///Un solo producto
  return this.http.get<ProductoModel[]>(`${this.BASE_URL}/productos/${idProducto}`);
}
agregarProducto(producto:ProductoModel){
  return this.http.post<ProductoModel>(`${this.BASE_URL}/productos`,producto);
}
actualizarProducto(producto: ProductoModel) {
  return this.http.put<ProductoModel>(`${this.BASE_URL}/productos/${producto.idProducto}`,producto);
}
borrarProducto(idProducto: string) {
  return this.http.delete<ProductoModel>(`${this.BASE_URL}/productos/${idProducto}`);
}
```

Crear el servicio para el modelo ‘UsuarioModel’ mediante: ng generate service shared/Services/usuario

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng generate service shared/Services/usuario
CREATE src/app/shared/Services/usuario.service.spec.ts (362 bytes)
CREATE src/app/shared/Services/usuario.service.ts (136 bytes)
```

En el archivo creado se implementan los métodos necesarios para comunicar con Backend.

```
obtenerUsuarioId(idUsuario: number){
  return this.http.get<UsuarioModel[]>(`${this.BASE_URL}/usuario/${idUsuario}`);
}
obtenerUsuarioEmail(email:string){
  return this.http.get<UsuarioModel[]>(`${this.BASE_URL}/usuario_email/${email}`);
}
agregarUsuario(usuario:UsuarioModel){
  return this.http.post<UsuarioModel>(`${this.BASE_URL}/usuario`,usuario);
}
```

Crear el servicio para el modelo ‘CompraModel’ mediante: ng generate service shared/Services/compra

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng generate service shared/Services/compra
CREATE src/app/shared/Services/compra.service.spec.ts (357 bytes)
CREATE src/app/shared/Services/compra.service.ts (135 bytes)
```

En el archivo generado se implementan los métodos necesarios para comunicar con Backend.

```
obtenerCompraId(idCompra: number){
  return this.http.get<CompraModel[]>(`${this.BASE_URL}/compra/${idCompra}`);
}
obtenerComprasUsuario(idUsuario:number){
  return this.http.get<CompraModel[]>(`${this.BASE_URL}/compras_usuario/${idUsuario}`);
}
agregarCompra(compra:CompraModel){
  return this.http.post<CompraModel>(`${this.BASE_URL}/compra`,compra);
}
```

Crear el servicio para el modelo ‘ProductoCompraModel’ mediante: ng generate service shared/Services/productoCompra

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng generate service shared/Services/productoCompra
CREATE src/app/shared/Services/producto-compra.service.spec.ts (398 bytes)
CREATE src/app/shared/Services/producto-compra.service.ts (143 bytes)
```

En el archivo generado se implementan los métodos necesarios para comunicar con Backend.

```
obtenerCompraId(idCompra: number){
  return this.http.get<ProductoCompraModel[]>(`${this.BASE_URL}/productos_comprados/${idCompra}`);
}
agregarCompra(productoCompra:ProductoCompraModel){
  return this.http.post<ProductoCompraModel>(`${this.BASE_URL}/producto_compra`,productoCompra);
}
```

Crear el servicio para el modelo ‘ProductoCarritoModel’ mediante: ng generate service shared/Services/productoCarrito

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng generate service shared/Services/productoCarrito
CREATE src/app/shared/Services/producto-carrito.service.spec.ts (403 bytes)
CREATE src/app/shared/Services/producto-carrito.service.ts (144 bytes)
```

Y en el archivo generado se definen los métodos necesarios para gestionar el carrito de compras:

```
obtenerCarrito(idUsuario: number){
  return this.http.get<ProductoCarritoModel[]>(`${this.BASE_URL}/productos_carrito/${idUsuario}`);
}
agregarProductoAlCarrito(productoCarrito: ProductoCarritoModel) {
  return this.http.post<ProductoCarritoModel>(`${this.BASE_URL}/producto_carrito`, productoCarrito);
}
actualizarProductoCarrito(productoCarrito: ProductoCarritoModel){
  return this.http.put<ProductoCarritoModel>(`${this.BASE_URL}/producto_carrito/${productoCarrito.
  idProductoCarrito}`,productoCarrito);
}
quitarProductoDelCarrito(idProductoCarrito: number){
  return this.http.delete<ProductoCarritoModel>(`${this.BASE_URL}/producto_carrito/$
  {idProductoCarrito}`);
}
vaciarCarrito(idUsuario: number){
  return this.http.delete<string>(`${this.BASE_URL}/carrito/${idUsuario}`);
}
```

Creacion de los componentes

Primero agregaremos la dependencia de Bootstrap en el archivo index.html para aplicar estilos de este framework a la aplicación:

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>FruverFrontend</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUVM" crossorigin="anonymous">
</head>
<body>
  <app-root></app-root>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4Bkz" crossorigin="anonymous"></script>
</body>
</html>
```

Y en el archivo app.component.html borramos todo su contenido y agregamos la línea: <router-outlet></router-outlet>

```
<router-outlet></router-outlet>
```

Ahora, crearemos un componente para mostrar la página de inicio de la aplicación: ng generate component inicio

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng generate component inicio
CREATE src/app/inicio/inicio.component.html (21 bytes)
CREATE src/app/inicio/inicio.component.spec.ts (559 bytes)
CREATE src/app/inicio/inicio.component.ts (202 bytes)
CREATE src/app/inicio/inicio.component.css (0 bytes)
UPDATE src/app/app.module.ts (475 bytes)
```

Creamos un componente para el login-registro de usuarios: ng g c login-registro

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng g c login
CREATE src/app/login/login.component.html (20 bytes)
CREATE src/app/login/login.component.spec.ts (552 bytes)
CREATE src/app/login/login.component.ts (198 bytes)
CREATE src/app/login/login.component.css (0 bytes)
UPDATE src/app/app.module.ts (553 bytes)
```

Creamos un componente para el Navbar que el usuario visualizará siempre: ng g c navbar

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng g c navbar
CREATE src/app/navbar/navbar.component.html (21 bytes)
CREATE src/app/navbar/navbar.component.spec.ts (559 bytes)
CREATE src/app/navbar/navbar.component.ts (202 bytes)
CREATE src/app/navbar/navbar.component.css (0 bytes)
UPDATE src/app/app.module.ts (635 bytes)
```

Creamos un componente para mostrar el carrito de compras: ng g c carrito

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng g c carrito
CREATE src/app/carrito/carrito.component.html (22 bytes)
CREATE src/app/carrito/carrito.component.spec.ts (566 bytes)
CREATE src/app/carrito/carrito.component.ts (206 bytes)
CREATE src/app/carrito/carrito.component.css (0 bytes)
UPDATE src/app/app.module.ts (721 bytes)
```

Generamos un componente para comprar: ng g c comprar

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng g c comprar
CREATE src/app/comprar/comprar.component.html (22 bytes)
CREATE src/app/comprar/comprar.component.spec.ts (566 bytes)
CREATE src/app/comprar/comprar.component.ts (206 bytes)
CREATE src/app/comprar/comprar.component.css (0 bytes)
UPDATE src/app/app.module.ts (807 bytes)
```

Generar un componente para que el administrador pueda visualizar los productos y gestionarlos: ng g c productos

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng g c productos
CREATE src/app/productos/productos.component.html (24 bytes)
CREATE src/app/productos/productos.component.spec.ts (580 bytes)
CREATE src/app/productos/productos.component.ts (214 bytes)
CREATE src/app/productos/productos.component.css (0 bytes)
UPDATE src/app/app.module.ts (901 bytes)
```

Un componente para agregar o editar los productos: ng g c editar-productos

```
PS C:\TallerFruverAlex\Frontend\FruverFrontend> ng g c editar-productos
CREATE src/app/editar-productos/editar-productos.component.html (31 bytes)
CREATE src/app/editar-productos/editar-productos.component.spec.ts (623 bytes)
CREATE src/app/editar-productos/editar-productos.component.ts (241 bytes)
CREATE src/app/editar-productos/editar-productos.component.css (0 bytes)
UPDATE src/app/app.module.ts (1021 bytes)
```

Configurar las Rutas

Se definen las rutas con las que trabajará la aplicación en el archivo app-routing.module.ts

```
const routes: Routes = [
  {path:'', component:InicioComponent},
  {path:'login', component:LoginComponent},
  {path:'carrito', component:CarritoComponent},
  {path:'comprar', component:ComprarComponent},
  {path:'productos', component:ProductosComponent},
  {path:'agregar', component:EditarProductosComponent},
  {path:'editar/:idProducto', component:EditarProductosComponent},
  {path:'**', redirectTo:'', pathMatch:'full'},

];
```

Configuración de los componentes

Componente para gestionar productos:
Se mostrarán los productos en una tabla, en la cual se visualizarán sus datos y se podrá modificar o eliminar cada producto. Para esto, en el componente html se define la cabecera de la tabla con los atributos de cada producto:

```
<div class="table-responsive">
    <table class="table table-bordered ">
        <thead>
            <tr class="text-center">
                <th class="id">ID</th>
                <th class="imagen">Imagen</th>
                <th>Nombre</th>
                <th class="tipo">Precio</th>
                <th class="detalle">Detalle</th>
                <th class="tipo">Tipo</th>
                <th>Acciones</th>
            </tr>
        </thead>
```

Se listan los productos mediante **ngFor** y en las celdas se colocan **textareas** que permitan modificar los datos de los productos. A través de **ngModel** se manejan estos atributos:

```
<tr *ngFor="let producto of productos | async">
    <td>{{ producto.idProducto }}<img [src]="producto.imagen" alt="Producto" width="50"></td>
    <td><textarea type="text" [(ngModel)]="producto.imagen" class="form-control" name="imagen" required></textarea></td>
    <td><textarea type="text" [(ngModel)]="producto.nombre" class="form-control" name="nombre" required></textarea></td>
    <td><textarea type="number" [(ngModel)]="producto.precio" class="form-control" name="precio" required></textarea></td>
    <td><textarea [(ngModel)]="producto.detalle" class="form-control" name="detalle" required></textarea></td>
    <td><select class="form-control" id="tipo" name="tipo" [(ngModel)]="producto.tipo" required>
            <option [ngValue]="null">Seleccione una opción</option>
            <option [ngValue]="'Frutas'">Frutas</option>
            <option [ngValue]="'Verduras'">Verduras</option>
            <option [ngValue]="'Granos'">Granos</option>
            <option [ngValue]="'Hortalizas'">Hortalizas</option>
            <option [ngValue]="'Hierbas'">Hierbas</option> </select></td>
```

Y en cada renglón de la tabla se tendrán los botones que permiten ejecutar las funciones para editar y eliminar:

```
<button class="btn btn-primary btn-sm" (click)="guardarProducto(producto)" [disabled]="producto.imagen === '' ||
producto.nombre === '' || producto.precio == 0 || producto.detalle === '' || producto.tipo === ''">Guardar</button>&
nbsp;
<button class="btn btn-danger btn-sm" (click)="borrarProducto(producto.idProducto)">Eliminar</button>
```

Además, se dejará un botón para agregar productos, el cual redirige al componente de agregar:

```
<h2 class="text-center">Gestión de Productos en Inventario</h2><br>
<div class="text-right">
    <h4>Total: {{total}}</h4>
    <button class="btn btn-primary" [routerLink]="['/agregar']">Agregar nuevo</button>
</div>
<br>
```

En el componente de typescript se definen las funciones para mostrar, editar y eliminar productos:

Se implementa el método OnInit para cargar los productos registrados y el total:

```
ngOnInit() {
  this.productos = this.productoService.obtenerProductos();
  this.productos.subscribe(productos => {
    this.total = productos.length;
  });
}
```

Se define el método **guardarProducto**, el cual recibe un producto de la tabla, el cual será modificado:

```
guardarProducto(producto: ProductoModel) {
  console.log(producto);
  this.productoService.actualizarProducto(producto).subscribe(data => {
    console.log(data);
    alert(`Se actualizó el producto ${data.idProducto}:${data.nombre}`);
    this.ngOnInit();
    // this.router.navigate(['/productos']);
    //Aqui mejor redireccionar a buscar el id del producto, agregar funcion
  });
}
```

Se define el método **borrarProducto**, el cual recibe un id del producto de la tabla, el cual será eliminado:

```
borrarProducto(idProducto: number) {
  console.log(idProducto);
  this.productoService.borrarProducto(idProducto).subscribe(data => {
    alert(`Se eliminó correctamente el producto ${idProducto}: ${data.nombre}`)
    console.log("Registro eliminado correctamente");
    this.ngOnInit();
  });
```

La interfaz es la siguiente:

Componente para agregar productos:
Se mostrará un formulario, en la cual se deben ingresar los datos del producto para ser agregado. Para esto, en el componente html se define el formulario con los atributos de cada producto:

```
<form (ngSubmit)="agregarProducto()" #agregarForm="ngForm">
    <div class="form-group">
        <label for="nombre">Nombre</label>
        <input type="text" class="form-control" id="nombre" name="nombre"
            [(ngModel)]="producto.nombre" required>
    </div>
    <div class="form-group">
        <label for="precio">Precio</label>
        <input type="number" class="form-control" id="precio" name="precio"
            [(ngModel)]="producto.precio" required>
    </div>
    <div class="form-group">
        <label for="detalle">Detalle</label>
        <textarea class="form-control" id="detalle" name="detalle" [(ngModel)]="producto.detalle"
            rows="3" required></textarea>
    </div>
```

```
<button type="submit" class="btn btn-primary" [disabled]="!agregarForm.form.valid">Agregar
    Producto</button>
```

El botón será deshabilitado hasta completar todos los campos.

En el componente typescript se definen las funciones para agregar un producto:

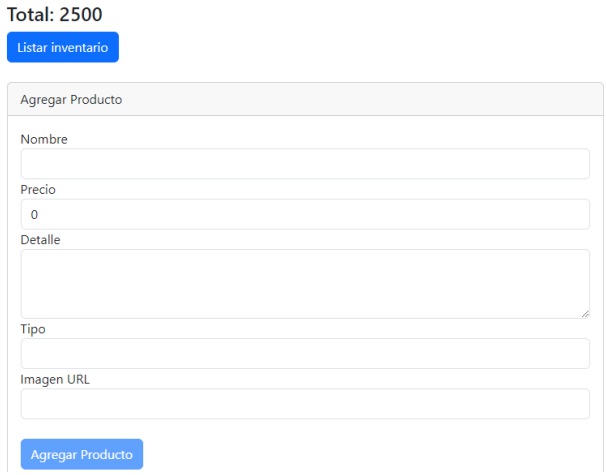
Se crea una instancia del modelo Producto para guardar los datos y mandarlos a Backend a través del servicio construido:

```
producto = new ProductoModel(0, "", 0, "", "", "");

constructor(private productoService: ProductoService, private router: Router) { }

agregarProducto() {
  console.log(this.producto);
  this.productoService.agregarProducto(this.producto).subscribe(data => {
    console.log(data);
    alert("See agregó el producto:" + this.producto.idProducto);
    this.router.navigate(['/productos']);
  });
}
```

La interfaz se ve de la siguiente manera:



Componente para login-registro de usuarios:

Se mostrarán dos formularios en tarjetas tipo Card, en las cuales se llenan los datos del usuario. Para esto, en el componente html se define la card con los atributos necesarios:

FORMULARIO PARA INICIAR SESION:

```
<form class="card" (ngSubmit)="login()" #loginForm="ngForm">
  <!-- Formulario de Inicio de Sesión -->
  <div class="col-6">
    <h3 class="text-center">Iniciar Sesión</h3>
    <hr><br>
    <h6 class="text-success text-center">{{mensajeSesionExitosa}}</h6>
    <h6 class="text-danger text-center">{{mensaje}}</h6><br>
    <div class="form-group">
      <label for="email">Email</label>
      <input type="email" class="form-control" id="email" name="email" [(ngModel)]="email"
        placeholder="Ingresa tu email" required>
      <div *ngIf="loginForm.submitted && loginForm.controls['email'].invalid" class="text-danger">
        Email inválido o requerido.
      </div>
    </div>
    <div class="form-group">
      <label for="password">Contraseña</label>
      <input type="password" class="form-control" id="password" name="password" [(ngModel)]
      ="password"
        placeholder="Ingresa tu contraseña" required>
      <div *ngIf="loginForm.submitted && loginForm.controls['password'].invalid" class="text-danger">
        Contraseña requerida.
```

En el componente typescript se define la función de login que permita redireccionar a la página de inicio si el rol del usuario es tipo ‘cliente’ o a la página de gestión de productos si es tipo ‘administrador’:

```
login() {
  console.log("Email:" + this.email);
  console.log("Pass:" + this.password);
  this.mensajeSesionExitosa = '';
  this.mensaje = '';

  // Llamamos al servicio para obtener el usuario por email
  this.usuarioService.obtenerUsuarioEmail(this.email).subscribe(
    (usuario) => {
      // Verificamos si se encontró un usuario con ese email
      console.log(usuario);
      if (usuario) {
        const usuarioEncontrado = usuario;
        // Verificamos si la contraseña coincide
```

```
// Verificamos si la contraseña coincide
if (usuarioEncontrado.password == this.password) {
  // Inicio de sesión exitoso
  // Guardamos el id del usuario en localStorage para mantener la sesión
  localStorage.setItem('idUsuario', usuarioEncontrado.idUsuario.toString())
  console.log('Inicio de sesión exitoso');
  this.mensajeSesionExitosa = 'Inicio de sesión exitoso';

  //Verificar si es admin o cliente
  if (usuarioEncontrado.rol == 'admin') {
    this.router.navigate(['/productos']);
  } else {
    this.router.navigate(['/inicio']);
  }
```

```
    } else {
      console.log('Contraseña incorrecta');
      this.mensaje = "Contraseña incorrecta";
    }
  } else {
    console.log('Usuario no encontrado');
    this.mensaje = 'Usuario no encontrado';
  }
```

Hay que tener en cuenta que, para el manejo de sesiones, se guarda el id del usuario en localStorage para utilizarlo en cada componente.

FORMULARIO PARA REGISTRO:

```
<form class="card" (ngSubmit)="registro()" #registroForm="ngForm">

  <!-- Formulario de Registro -->
  <div class="col-6">
    <h3 class="text-center">Registro</h3>
    <hr><br>
    <div class="form-group">
      <label for="emailRegistro">Email</label>
      <input type="email" class="form-control" id="emailRegistro" name="email" required [(ngModel)]="usuario.email" placeholder="Ingresa tu
      email">
    </div>
    <div class="form-group">
      <label for="passwordRegistro">Contraseña</label>
      <input type="password" class="form-control" id="passwordRegistro" name="password" required [(ngModel)]="usuario.password"
      placeholder="Ingresa tu contraseña">
    </div>
```

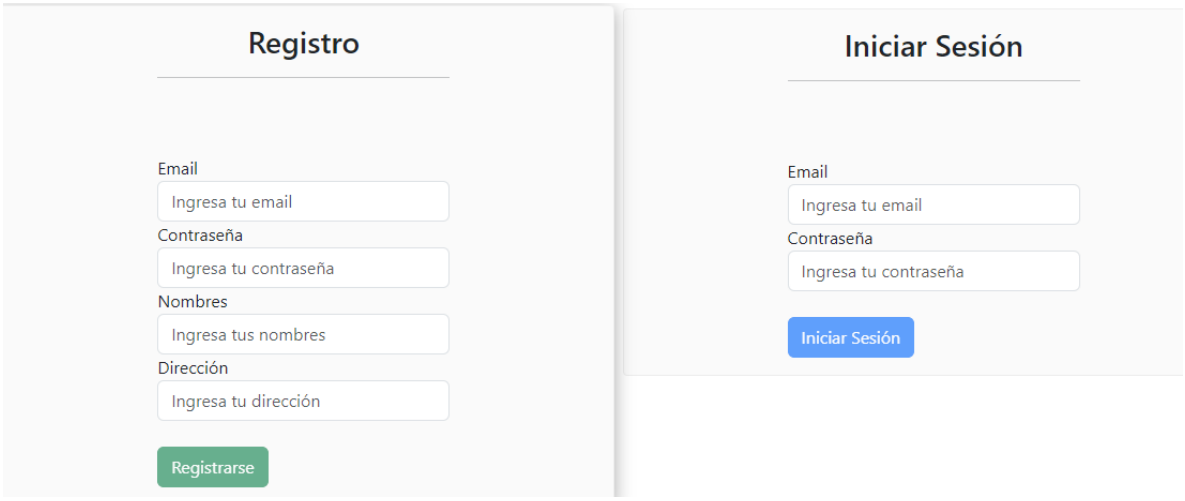
El botón será deshabilitado hasta completar los campos.

```
<button type="submit" class="btn btn-success" [disabled]="!registroForm.form.valid">Registrarse</button>
```

En el componente typescript se define la función para registro del usuario, en la cual se valida que el correo ingresado no exista en la base de datos, se asigna el rol de 'cliente' y al no existir errores se guarda los datos del usuario a través del servicio, y se guarda el id generado de la base de datos en localStorage para redirigir a la página de inicio.

```
registro() {
  console.log(this.usuario);
  this.mensajeEmail = '';
  this.usuarioService.obtenerUsuarioEmail(this.usuario.email).subscribe((usuario) => {
    if (usuario) {//si existe este correo mostramos el mensaje
      this.mensajeEmail = 'El correo ingresado ya existe, debe elegir otro';
    } else {
      this.usuarioService.agregarUsuario(this.usuario).subscribe((usuario) => {
        const usuarioCreado = usuario;
        localStorage.setItem('idUsuario', usuarioCreado.idUsuario.toString());
          console.log('Inicio de sesión exitoso');
          this.mensajeSesionExitosa = 'Inicio de sesión exitoso';
          this.router.navigate(['/inicio']);
      });
    }
  });
```

La interfaz se ve así:



Componente para inicio de usuarios:
Se mostrarán los productos en tarjetas tipo **Card**, en las cuales se visualizarán sus datos y se podrá agregar una cantidad de cada producto. Para esto, en el componente html se define la card con los atributos de cada producto:

```
<div class="card-container">
  <form *ngFor="let producto of productos | async" class="card"  #productoForm="ngForm">
    <img src="{{producto.imagen}}" alt="img" width="200">

    <span>{{producto.idProducto}}. {{producto.nombre}}: $ {{producto.precio}} <span class="small">[{{producto.tipo}}]</span>
      <hr><br>
      <p>{{producto.detalle}}</p><br>
      <label class="form-label" for="cantidad">Cantidad
        <input type="number" name="cantidad" id="cantidad" style="width: 80px;" min="1" value="1" required
               [(ngModel)]="cantidades[producto.idProducto]">
      </label><br>
      <button type="button" class="btn btn-primary" (click)="agregarProducto(producto)" [disabled]="!productoForm.form.valid">Agregar al
      carrito</button>
    </span>
  </form>
</div>
```

En el evento **OnInit()** se obtienen los productos de Backend para guardarlos en la variable que se itera en las tarjetas:

```
ngOnInit() {
  this.productos = this.productoService.obtenerProductos();
}
```

Mediante el evento **ngSubmit** se llama a la función para agregar el producto al carrito

```
agregarProducto(producto: ProductoModel) {
  console.log(producto);
  const cantidad = this.cantidades[producto.idProducto];
  //Hay que sacar el id del usuario en sesion desde la base de datos o tener el sesion su id
  const id = localStorage.getItem('idUsuario');
  if (id) {
    this.productoCarrito.idUsuario = parseInt(id);
    this.productoCarrito.cantidad=cantidad;
    this.productoCarrito.producto = producto;
    this.productoCarrito.idProducto = producto.idProducto;
    this.productoCarrito.valorProductoCarrito = producto.precio * this.productoCarrito.cantidad;

    console.log(this.productoCarrito);

    this.productoCarritoService.agregarProductoAlCarrito(this.productoCarrito).subscribe(prod => {
      alert("Se agregó el producto al carrito");
    });
  }
}
```

La interfaz se ve asi:

Componente gestionar productos del carrito:

Se mostrarán los productos del carrito en una tabla, en la cual se visualizarán sus datos y se podrá modificar una cantidad de cada producto o sacarlo del carrito. Para esto, en el componente html se define la tabla con los atributos de cada producto:

```
<table class="table table-bordered">
  <thead>
    <tr>
      <th>Imagen</th>
      <th>Nombre</th>
      <th>Precio</th>
      <th>Cantidad</th>
      <th>Subtotal</th>
      <th></th>
    </tr>
  </thead>
```

En el cuerpo de la tabla se mostrarán los datos de cada producto y los botones que servirán para cambiar la cantidad o eliminar el producto del carrito:

```
<tbody>
  <tr *ngFor="let prodCart of productos | async">
    <td><img src="{{prodCart.Producto.imagen}}" alt="Producto" width="50"></td>
    <td>{{ prodCart.Producto.nombre }}</td>
    <td>{{ prodCart.Producto.precio }}</td>
    <td style="max-width: 30%;">
      <div class="row">
        <div class="col-4">
          <input type="number" class="form-control" style="width: 80px;" [(ngModel)]="prodCart.cantidad" min="1" (input)="actualizar
          (prodCart)" (change)="actualizar(prodCart)">
        </div>
        <div class="col-3"><button class="btn btn-success btn-sm" (click)="actualizar(prodCart)" [disabled]="prodCart.
        cantidad<1">Actualizar</button>
        </div>
      </div>
    </td>
    <td>$ {{ (prodCart.Producto.precio * prodCart.cantidad) }}</td>
    <td>
      <button class="btn btn-danger btn-sm" (click)="eliminar(prodCart)">Eliminar</button>
    </td>
  </tr>
</tbody>
```

Además, se tiene un botón con el que se puede vaciar el carrito y un botón con el que se procede a la compra:

```
<div class="text-right">
  <button class="btn btn-danger" (click)="vaciarCarrito()">Vaciar</button>
</div><br>
```

```
<div class="text-right">
  <h4>Total: $ {{valorTotal}}</h4>
  <button class="btn btn-primary" (click)="finalizarCompra()" data-bs-toggle="modal"
  data-bs-target="#exampleModal">Proceder a Comprar</button>
</div>
```

En el componente typescript se definen las funciones para llevar a cabo el proceso de gestión de productos del carrito. En el método **OnInit()** se obtienen los productos que tenga registrados el usuario que tenga iniciado sesión y se calcula el valor total del carrito.

```
productos: Observable<ProductoCarritoModel[]> | undefined;
valorTotal = 0;

constructor(private productoCarritoService: ProductoCarritoService) { }

ngOnInit() {
  const id = localStorage.getItem('idUsuario');
  ///Buscar el id del usuario que esta en sesion, sino no mostrará productos
  if (id) {
    const idUsuario = parseInt(id);
    console.log(idUsuario);
    this.productos = this.productoCarritoService.obtenerCarrito(idUsuario);
    this.productos.subscribe((data) => {
      console.log(data);
      this.valorTotal = data.reduce((total, productoCarrito) => {
        return total + Number(productoCarrito.valorProductoCarrito);
      }, 0);
      console.log('Valor Total:', this.valorTotal);
    });
  }
}
```

El método **actualizar()** recibe de parámetro el producto del carrito que será modificado para cambiar su cantidad y valor:

```
actualizar(producto: ProductoCarritoModel) {
  producto.valorProductoCarrito = producto.Producto.precio * producto.cantidad;
  console.log(producto.valorProductoCarrito);
  this.productoCarritoService.actualizarProductoCarrito(producto).subscribe(data => {
    console.log("Actual");
    console.log(data);
    this.ngOnInit();
  },
    (error) => {
      console.log('Error al actualizar el carrito:', error);
    });
}
```

El método **eliminar()** recibe el producto que será sacado del carrito, para esto se utiliza su id:

```
eliminar(producto: ProductoCarritoModel) {
  this.productoCarritoService.quitarProductoDelCarrito(producto.idProductoCarrito).subscribe(data => {
    this.ngOnInit();
  },
    (error) => {
      console.log('Error al eliminar el producto:', error);
    });
}
```

Y el método **vaciarCarrito()** eliminará todos los productos del carrito del usuario, para esto se necesita obtener el id del usuario en sesión:

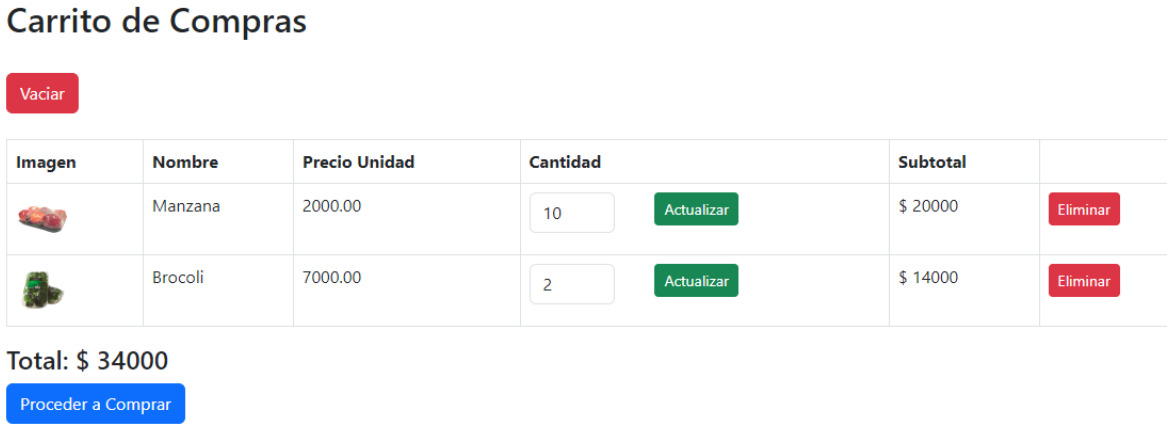
```
vaciarCarrito() {
  const id = localStorage.getItem('idUsuario');
  if (id) {
    const idUsuario = parseInt(id);
    this.productoCarritoService.vaciarCarrito(idUsuario).subscribe(data => {
      this.ngOnInit();
    },
      (error) => {
        console.log('Error al eliminar el producto:', error);
      });
  }
}
```

Y se tiene un metodo **mostrarCompra()**, el cual redirige a una página para mostrar la factura de la compra realizada. Se envía el id de la compra hecha.

```
mostrarCompra() {
  // Navegamos a la ruta /comprar con el idCompra como parámetro de consulta
  console.log(this.idCompra);
  this.router.navigateByUrl(`/comprar/${this.idCompra}`);

}
```

Y la interfaz se ve así:



Componente Mostrar Compra

Se muestra la factura de la compra realizada con los detalles de cada producto en una tabla. Para esto, se implementa una tabla en html:

```
<div *ngIf="productosCompra && total > 0">
  <h3>Productos de la Compra: {{total}}</h3>
  <table class="table">
    <thead>
      <tr>
        <th>ID</th>
        <th>Nombre</th>
        <th>Precio</th>
        <th>Cantidad</th>
        <th>Subtotal</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let productoCompra of productosCompra | async">
        <td>{{ productoCompra.Producto.idProducto }}</td>
        <td>{{ productoCompra.Producto.nombre }}</td>
        <td>${{ productoCompra.Producto.precio }}</td>
        <td>{{ productoCompra.cantidad }}</td>
        <td>${{ productoCompra.valorProductoCompra }}</td>
      </tr>
    </tbody>
  </table>
```

Y se muestran algunos datos como la fecha, valor de pago y datos del cliente:

```
<div>
  <h3 class="text-center">Información de la Compra</h3>
  <span><b> Fecha: </b>{{ fecha }}</span><br>
  <span><b>Valor Total: </b>${{ valorTotal }}</span><br>
  <span *ngIf="compra"><b>Cliente:</b> {{ compra.Usuario.nombres }}</span><br>
  <span *ngIf="compra"><b>Dirección:</b> {{ compra.Usuario.direccion }}</span><br>
  <span *ngIf="compra"><b>Correo:</b> {{ compra.Usuario.email }}</span><br>

</div>
```

En el componente typescript se definen las funciones para mostrar los productos comprados y detalles de la compra.

En el método **OnInit()** se recibe el id de la compra realizada para consultar los datos al Backend con ese id.

```
ngOnInit() {
  this.route.params.subscribe(params => {
    this.idCompra = params['idCompra'];
    console.log('idCompra recibido:', this.idCompra);
  });
```

Y se guardan los datos de la compra y de los productos de esa compra:

```
this.compraService.obtenerCompraId(this.idCompra).subscribe(compraObtenida => {
  this.compra = compraObtenida[0];
  console.log(this.compra);
  this.fecha = this.compra.fecha;
  this.valorTotal=this.compra.valorTotal;
  console.log(this.fecha);
  this.productosCompra = this.productoCompraService.obtenerProductoCompraId(this.idCompra);
  this.productosCompra.subscribe(productosCompra => {
    this.total = productosCompra.length;
  });
});
```

Y la interfaz se ve algo así:

Fecha: 2023-07-30T17:04:13.000Z
Valor Total: $23000.00
Cliente: Usuario1
Dirección: Pasto
Correo: sample@example.com

Productos de la Compra: 2

ID	Nombre	Precio	Cantidad	Subtotal
1	Manzana	$2000.00	1	$2000.00
2	Brocoli	$7000.00	3	$21000.00

Componente Navbar

En este componente se muestra una serie de botones los cuales servirán para navegar a la página inicial, al formulario de inicio de sesión y mostrar el carrito, además de mostrar la cantidad de productos en carrito:

En el componente typescript se calcula el número de productos que hay en carrito:

```
productos: Observable<ProductoCarritoModel[]> | undefined;
total = 0;
constructor(private productosCarritoService: ProductoCarritoService) { }


ngOnInit() {
  const id = localStorage.getItem('idUsuario');
  if (id) {
    const idUsuario=parseInt(id);
    this.productos = this.productosCarritoService.obtenerCarrito(idUsuario);
    this.productos.subscribe(carrito=>{
      this.total=carrito.length;
    })
  }
}
```

Y la visualización del navbar seria:

Con estos componentes se ha concluido la construcción del proyecto Frontend en Angular. En caso de necesitar mas funciones, se irán adicionando al proyecto.