Proyecto 1 trimestre
Lucas Coronilla Salmeron

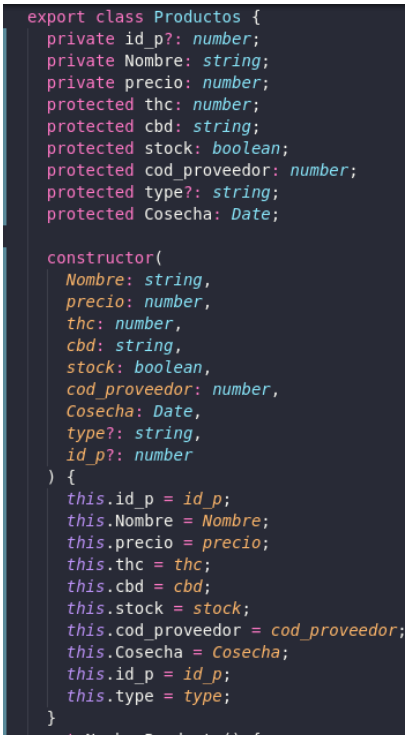
1-Introducción al proyecto.

Mi proyecto trata sobre un sistema de login y compras con carrito de una asociación cannábica en el cual se busca otorgar a cada usuario la comodidad de loguearse y que todo lo que realice se guarde en su cuenta y sea accesible a lo largo del tiempo.

2-Clases.

Tendremos dos superclases las cuales son clientes y productos se relacionan mediante una relación 1:N.

*Producto => de producto se extienden 2 subclases las cuales son plantas y extractos las cuales cuentan con métodos enfocados a la programación de objetos haciendo un sistema de productos escalable.



```
export class Productos {
  private id_p?: number;
  private Nombre: string;
  private precio: number;
  protected thc: number;
  protected cbd: string;
  protected stock: boolean;
  protected cod_proveedor: number;
  protected type?: string;
  protected Cosecha: Date;

  constructor(
    Nombre: string,
    precio: number,
    thc: number,
    cbd: string,
    stock: boolean,
    cod_proveedor: number,
    Cosecha: Date,
    type?: string,
    id_p?: number
  ) {
    this.id_p = id_p;
    this.Nombre = Nombre;
    this.precio = precio;
    this.thc = thc;
    this.cbd = cbd;
    this.stock = stock;
    this.cod_proveedor = cod_proveedor;
    this.Cosecha = Cosecha;
    this.id_p = id_p;
    this.type = type;
  }
```

```ts
export class Plantas extends Productos {
  private Genetica: Genetica;
  private humedad: Number;
  private Apta_para_extracto: boolean;

  constructor(
    Nombre: string,
    precio: number,
    thc: number,
    cbd: string,
    stock: boolean,
    cod_proveedor: number,
    Genetica: Genetica,
    humedad: Number,
    Apta_para_extracto: boolean,
    Cosecha: Date,
    id_p?: number,
    type?: string
  ) {
    super(Nombre, precio, thc, cbd, stock, cod_proveedor, Cosecha, type, id_p);
    this.Genetica = Genetica;
    this.humedad = humedad;
    this.Apta_para_extracto = Apta_para_extracto;
  }
```

```ts
export class Extracto extends Productos {
    private N_apaleo: number;
    private mutable: boolean;
    private variedad: string;

    constructor(
      Nombre: string,
      precio: number,
      thc: number,
      cbd: string,
      stock: boolean,
      cod_proveedor: number,
      Cosecha: Date,
      N_apaleo: number,
      mutable: boolean,
      variedad: string,
      id_p?: number,
      type?: string
    ) {
      super(Nombre, precio, thc, cbd, stock, cod_proveedor, Cosecha, type, id_p);
      this.N_apaleo = N_apaleo;
      this.mutable = mutable;
      this.variedad = variedad;
    }
```

Clientes =>la cual se dividirá en clientes normales y Mayoristas así diferenciaremos y podremos aplicar descuento a mayoristas que compran productos a gran escala.

```
export class Cliente {
    _id?: number;
    protected _nombre: string;
    protected _apellidos: String;
    protected _dni: String;
    protected _nombreUsuario: string;
    protected _Contraseña: string;
    private _pedidos: Array<number> = [];
    private _gramos: Array<number> = [];
    private _recibo: Boolean;
    private _status: Boolean;
    type: string;

    constructor(
        _nombre: string,
        _apellido: string,
        _dni: string,
        _nombreUsuario: string,
        _Contraseña: string,
        _pedidos: Array<number>,
        _gramos: number[],
        _recibo: boolean,
        _status: boolean,
        id?: number
    ) {
```

```
export class Mayoristas extends Cliente {
    private numEmpresa: string;
    type: string;

    constructor(
        _nombre: string,
        _apellido: string,
        _dni: string,
        _nombreUsuario: string,
        _Contraseña: string,
        _pedidos: Array<number>,
        _gramos: number[],
        _recibo: boolean,
        _status: boolean,
        numEmpresa: string,
        id?: number
    ) {
        super(
            _nombre,
            _apellido,
            _dni,
            _nombreUsuario,
            _Contraseña,
            _pedidos,
            _gramos,
            _recibo,
            _status,
            id
        );
```

La última clase será pedidos que es donde guardaremos los datos de las compras que los clientes confirmen en el carrito.

```
export class Pedidos {

    private pedidos : number[]
    private gramos: number[]
    private fecha?: Date
    private total :number
    private cliente: number

    constructor (pedidos : number[] , gramos: number[] , cliente: number , fecha?:Date){

        this.pedidos = pedidos;
        this.gramos = gramos;
        if (fecha === undefined){
        this.fecha = new Date()
        }else{
            this.fecha = fecha
        }
        this.cliente = cliente
    }
```

3-Métodos interesantes.

Exist: Mediante este método de la clase clientes podemos elegir pasarle o no el nombre de un usuario lo que nos permitirá utilizarlo sobre un objeto cliente o sus subclases o mediante ClientFunc el cual es un objeto clientes vacío llamaríamos esta función pasandole un nombre de usuario devolviendo si existe o no , así conseguimos un código organizado , escalable y reducimos la repetición de código.

```
Exist(user?: string) {
    const promise = new Promise<boolean>(async (resolve, reject) => {
        let exist = 'a';
        if (user != undefined) {
            exist = await clientModel.findOne({ _nombreUsuario: user });
        } else {
            exist = await clientModel.findOne({
                _nombreUsuario: this._nombreUsuario,
            });
        }
        if (exist != null) {
            resolve(true);
        } else {
            resolve(false);
        }
    });
    return promise;
}
```

creator: Dentro de cada clase hay un método creator el cual puede recibir o no una id ya sea porque tengamos que recoger un objeto de una query el cual posee id , o a la hora de crearlo sin id.

```
//funciones tecnicas:
creator(
    _nombre: string,
    _apellido: string,
    _dni: string,
    _nombreUsuario: string,
    _Contraseña: string,
    _pedidos: number[],
    _gramos: number[],
    _recibo: boolean,
    _status: boolean,
    id?: number
) {
    if (id === undefined) {
        return new Cliente(
            _nombre,
            _apellido,
            _dni,
            _nombreUsuario,
            _Contraseña,
            _pedidos,
            _gramos,
            _recibo,
            _status
        );
    }
    return new Cliente(
        _nombre,
        _apellido,
        _dni,
        _nombreUsuario,
        _Contraseña,
        _pedidos,
        _gramos,
        _recibo,
        _status,
        id
    );
}
```

Tiempo: Es un método de la clase pedidos el cual teniendo en cuenta que la empresa tendría un tiempo de envío estipulado de 12 dias de envio calcula la diferencia de tiempo y indica si el producto esta retrasado o no.

```
tiempo(number: number){
    let actualDate = new Date()
    if (this.fecha !== undefined){
    let dateDifference =  actualDate.getTime() - this.fecha.getTime()
    dateDifference = Math.round((dateDifference / (1000*60*60*24)))
    console.log(`pedido ${number}:`)
    if (dateDifference < 12){
        console.log(`a su pedido le quedan ${12 - dateDifference} para llegar`)
    }else{
        console.log(`su pedido esta retrasado por ${dateDifference - 12} dias`)
    }
    }
}
```

verCarrito: Es un método de la clase clientes la cual permite ver de forma organizada el carrito y trabaja con 2 arrays los cuales son 2 arrays de objetos de plantas y extractos los cuales han sido obtenidos anteriormente mediante una query y separados. Utilizaremos métodos en arrays tales como find que es muy bueno para trabajar con array de objetos y reduce el cual sirve para reducir elementos de un array a un solo valor.

```
79
80      verCarrito(plantas: Plantas[], extractos: Extracto[]) {
81          let pedidos: Array<number> = this.pedidos;
82          let gramos: Array<number> = this.gramos;
83          let x = 0;
84          let total = 0;
85          let totalIva = 0;
86
87          if (pedidos.length !== 0) {
88              let gram = gramos.reduce((a, b) => a + b);
89
90              for (let pedido of pedidos) {
91                  let temp: Plantas | undefined = plantas.find(
92                      (planta) => planta.id == pedido
93                  );
94                  if (temp !== undefined) {
95                      console.log(
96                          `${x}.- ${gramos[x]} gramos de ${
97                              temp.NombreProducto
98                          } por un precio total de ${temp.totalprice(gramos[x], this.type)}€`
99                      );
100                     total = temp.totalprice(gramos[x],this.type) + total;
101                     totalIva = temp.totalIva(gramos[x], this.type)
102                 } else {
103                     let temp: Extracto | undefined = extractos.find(
104                         (extracto) => extracto.id == pedido
105                     );
106                     if (temp !== undefined) {
107                         console.log(
108                             `${x}.- ${gramos[x]} gramos de ${
109                                 temp.NombreProducto
110                             } por un precio total de ${temp.totalprice(gramos[x],this.type)}€`
111                         );
112                         total = temp.totalprice(gramos[x],this.type) + total;
113                         totalIva = temp.totalIva(gramos[x],this.type) + totalIva
114                     }
115                 }
116                 x++;
117             }
118             console.log(`un total de ${gram} gramos por ${total}€`);
119             console.log(`iva_incluido:${totalIva}`)
120         } else {
121             console.log('el carrito esta vacio');
122             return false;
123         }
124     }
```

PrecioTotal= Un método que aplica polimorfismo , cuando se ejecuta en un objeto tipo planta añade condiciones elevando el precio depende de las características de la planta mientras que para los demás productos lo hará de forma normal además de ofrecer un descuento depende del tipo de usuario que seas.

```
totalprice(gramos: number, type: string) {
  if (type == 'M') {
    if (this.Apta_para_extracto) {
      return this._precio * gramos * 1.25 * 0.75;
    } else {
      return this._precio * gramos * 0.75;
    }
  } else {
    if (this.Apta_para_extracto) {
      return this._precio * gramos * 1.25;
    } else {
      return this._precio * gramos;
    }
  }
}
```

```
totalprice(grams: number, type: string) {
  if (this.type == 'M') {
    return this._precio * grams * 0.75;
  } else {
    return this._precio * grams;
  }
}
```

4-Aportacion Personal.

auto-increment ID => un plugin de mongoose el cual se aplica en el esquema añadiendo automáticamente un id autoincremental evitando el tener que hacer funciones para generar uno , o insertarlo manualmente.

```
//Al crear la conexion inizializamos el plugin para tener id autoincrementables
autoIncrement.initialize(connection);

export const ClienteSchema = new Schema({
    _id: { type: String },
    _nombre: { type: String, unique: true },
    _apellidos: { type: String },
    _dni: { type: String },
    _nombreUsuario: { type: String },
    _Contraseña: { type: String },
    _pedidos: { type: Array },
    _gramos: { type: Array },
    numEmpresa: { type: String },
    _recibo: { type: Boolean },
    _status: { type: Boolean },
    type: { type: String },
});

ClienteSchema.plugin(autoIncrement.plugin, 'Cliente');
export const clientModel: Cliente | any = connection.model<Cliente>(
    'cliente',
    ClienteSchema
);
```

Sistema de login => Cada cliente cuenta con un valor booleano llamado status el cual se actualiza cuando inicias secion con ese cliente permitiendote hacer todas las operaciones desde un cliente y luego cambiar , además que en caso de desconectarse sin cerrar sesion como status seguirá en true , no tendrás que volver a loguearte evitando errores.