3-2-2023

Proyecto web API

para Scent-ists (velas perfumadas)

Eva Aramburu Domínguez
FEMCODERS F5

Indice

Negocio elegido y fundamentación del producto
- Diseño de tablas
- Tecnologías, lenguajes y versiones

Documentación funcional de la web API
- Objetivos
- Sobre la arquitectura
- Funcionalidades

Extensiones posibles, correcciones y mejoras

Muestra visual
- Anexo

Negocio elegido y fundamentación del producto

1.

Este producto ha sido diseñado para un pequeño negocio de comercialización de velas aromáticas llamado *Scent-ists*. Para ellos, hemos desarrollado una API que contempla la particularidad de sus productos, como son los registros de peso y aroma.

En términos más generales, nuestro producto está orientado al pequeño comercio en vías de digitalización, cuyo foco de interés esté en la gestión de sus productos, compradores y pedidos. No obstante, cuenta con una estructura de entidades que aspira a ser versátil y capaz de adaptarse a las demandas del cliente. Por ejemplo, permitiendo incluir nuevos administradores o gestores en el negocio, o llevando a cabo un registro de compradores habituales.

Cabe mencionar que esta es una versión inicial del producto. A pesar de ello, ya estaría listo para ofrecer unas operaciones de gestión mínimas, como la inserción, edición y desactivación de entidades) para un negocio de las características anteriormente mencionadas.

Diseño de tablas

El diseño de nuestras tablas distingue tres tipos de usuario: User(refiere a aquellas entidades que acceden a nuestra aplicación), Admin(entidad reservada a administradores del negocio) y Buyer(usuarios compradores o clientes de la marca *Scent-ists*).

Por otra parte, contamos con otras dos clases: una dedicadas a la gestión de los productos del negocio(Items) y otra reservada para la gestión de los pedidos de los clientes(Orders).

Hemos generado una relación entre tablas donde los pedidos quedan vinculados al comprador(Buyers) a través del atributo userId, y a la tabla de productos(Items) a través del itemId. [1]

Tecnologías, lenguajes y versiones

Este producto ha sido desarrollado sobre Visual Studio 2022 y Microsoft SQL Management Server con lenguaje C#, .Net Core 6 como framework y su librería Entity Framework 7.

[1] Consultar 4.2 Anexo para consultar el diagrama relacional de las tablas

Documentación funcional de la web API

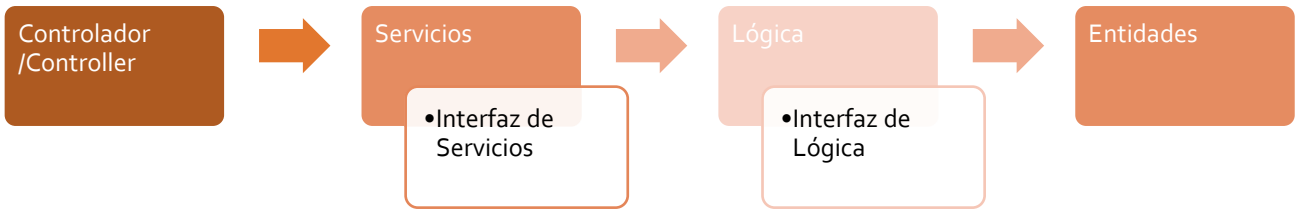
La siguiente documentación pretende presentar en detalle los métodos y funcionalidades de nuestro producto.

Objetivos

Nuestro producto pretende responder y adaptarse a las necesidades de un pequeño negocio local dedicado a la comercialización de velas perfumadas o aromáticas. Entre sus objetivos destacan 1) ofrecer una primera solución API que permita a nuestro cliente gestionar de manera simple y eficaz las entidades relacionadas con el funcionamiento básico de su negocio: productos, compradores y pedidos; 2) facilitar el proceso de digitalización de los negocios de nuestros clientes a través de una integración gradual de tecnologías y funcionalidades; 3) apoyar a nuestros clientes durante dicho proceso de digitalización gracias a la aportación de recursos de orientación como la presente documentación funcional.

Sobre la arquitectura

Nuestro producto cuenta con una arquitectura que distingue (actualmente) seis entidades con un máximo de siete métodos por cada una de ellas. Cada método ha sido construido siguiendo un esquema controlador / servicios / lógica / entidades. La creación e implementación de interfaces sobre las clases Servicios y Lógica tiene como objetivo presentar sus operadores como roles abstractos, y así mejorar la independencia de sus funciones de aquellos objetos encargados de llevarlas a cabo. De esta forma, no solo logramos que nuestra aplicación sea más autónoma y, por lo tanto, más adaptable, sino que además nos va a facilitar su mantenimiento y revisión a lo largo del tiempo.



Por último, contamos con un proyecto (librería de clases) de recursos(Resources), donde se pueden encontrar modelos de filtrado(Filter) y de solicitud(Request), que útiles para el desarrollo de funcionalidades como la selección de registros o la creación de plantillas para la solicitud y posterior inserción o modificación de registros.

Funcionalidades

Nuestro proyecto web API cumple con las siguientes funcionalidades en todas las entidades actualmente desarrolladas:

POST: método que permite insertar nuevos registros de cada entidad en la base de datos.

GET(ALL): método que permite recuperar o mostrar todos los registros de una entidad en una base de datos.

DEACTIVATE: método que permite desactivar un registro concreto en la base de datos.

Además, las entidades de productos (Items), usuarios(Users) y pedidos(Orders) cuentan con los siguientes métodos adicionalmente:

GET(SELECTED): método que permite recuperar o mostrar un registro concreto en base a su identificador o la aplicación de un modelo de filtrado específico.

GET(BY BRAND): método que permite recuperar o mostrar uno o varios registros que comparten la misma marca(Brand). Sólo aplicado a la entidad Items.

DELETE: método que permite eliminar de forma permanente un registro.

PATCH: método que permite editar un registro.

A modo de aclaración, hemos considerado oportuno distinguir un método que desactive registros y otro que los elimine por completo. Nuestra propuesta es aplicar los métodos de desactivación (Deactivate) a todas las

entidades, de modo que nuestro cliente cuente con la opción de recuperar dichos registros desactivados en un futuro, sin necesidad de volver a introducir sus datos. El método para borrar registros (Delete), sin embargo, sólo está presente por defecto en las entidades de productos (Items), usuarios (Users) y pedidos(Orders). Esta propuesta está sujeta a cambios y pretende ajustarse en todo momento a las necesidades y deseos de nuestro cliente.

Extensiones posibles, correcciones y mejoras

Si bien en este punto del desarrollo no cuenta con credenciales de seguridad, aspira a contar con ellos en una fase próxima. Esto significaría, entre otras mejoras, incluir un servicio de seguridad que valide credenciales de acceso, por ejemplo del tipo usuario-contraseña; incluir la encriptación de contraseñas, pasar parámetros FromHeader en los métodos, etc. En esta dirección, hemos desarrollado una clase UserRol que pretende gestionar en primera estancia dos tipos de roles dentro de la empresa: Administrador(Admin) o Comprador(Buyer). De esta forma las clases Admin y Buyer quedarían vinculadas a la clase UserRol a través del atributo idRol en cada una de sus tablas.

Por otro lado, podemos sugerir una serie de funcionalidades que podrían responder a necesidades concretas del cliente llegado el caso. Por ejemplo, creación de una nueva clase de usuario(User) o compradores(Buyer) y la aplicación de descuentos sobre el precio a los mismos, dependiendo de su nivel de registro, perfil de usuario (autónomo, mayorista, minorista…) o su identidad (física o jurídica).

Del mismo modo, estamos trabajando en la creación de métodos de inserción y almacenamiento de imágenes más concretamente dentro de la clase Items(productos).

Dado que la estamos presentando una versión todavía inicial de nuestro producto, esta sección queda abierta a las sugerencias y nuevas necesidades que nuestro cliente precise cubrir.

Muestra visual

A continuación, presentamos una exposición de los métodos desarrollados para cada una de las entidades propuestas en el proyecto web API para *Scent-ists*.

⇒ Endpoint /User

Reúne las solicitudes referidas a alta, baja, modificación y consulta de usuarios.

POST: permite insertar un nuevo registro del tipo usuario(User).

Request: parámetros FromBody (modelo UserRequest)

```
{
 "document": "string",
 "name": "string",
 "userEmail": "string",
 "birthday": "2023-02-02T13:55:57.325Z",
 "phone": 0,
 "password": "string"[2]
}
```

[2] Modelo UserRequest utilizado para introducir un nuevo usuario (User)

GET/User/GetAllUsers: reúne todos los registros de usuario(User)

Request: pasa sin parámetros adicionales.

GET/User/GetSelectedUser: recupera o muestra un registro usando su identificador(Id).

Request: parámetros FromQuery: "Id": int.

DELETE/User/DeactivateUser: desactiva un registro de un usuario a través de su identificador.

Request: parámetros FromQuery: "Id": int.

DELETE/User/DeleteUser: elimina el registro de un usuario a través de su identificador.

Request: parámetros FromQuery: "Id": int.

PATCH/User/UpdateUser: modifica el registro de un usuario recuperando una estructura JSON por defecto de la clase User. Requiere identificar este registro por defecto con el registro que se desea modificar a través de su "Id" durante la modificación.


```
{
  "id": 0,
  "idWeb": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "document": "string",
  "name": "string",
  "userEmail": "string",
  "phone": 0,
  "birthday": "2023-02-02T14:14:03.992Z",
  "password": "string",
  "insertDate": "2023-02-02T14:14:03.992Z",
  "updatedDate": "2023-02-02T14:14:03.992Z",
  "isActive": true
}
```


[3] Estructura JSON por defecto de la clase User

⇒ Endpoint /Admin

Reúne las solicitudes referidas a alta, baja, modificación y consulta de administradores o trabajadores (Admin).

POST: inserta un nuevo administrador (Admin).

Request: sin parámetros adicionales. Introduce el administrador usando la estructura de la clase por defecto.

```
{
  "id": 0,
  "userId": 0,
  "name": "string",
  "companyName": "string",
  "companyEmail": "string",
  "isActive": true
}
```

GET/Admin/GetAllAdmins: recupera o muestra todos los registros de administradores(Admin).

Request: sin parámetros adicionales.

DELETE/Admin/DeactivateAdmin: desactiva un registro de administrador usando su identificador

Request: parámetros FromQuery: "Id": Int.

⇒ Endpoint /Buyer

Reúne las solicitudes referidas a alta, baja, modificación y consulta de compradores o clientes(Buyer).

POST: inserta un nuevo comprador (Buyer).

[4] Estructura JSON por defecto de la clase Admin

Request: sin parámetros adicionales. Introduce el comprador usando la estructura de la clase por defecto.

```
{
  "id": 0,
  "idRol": 0,
  "userName": "string",
  "address": "string",
  "orderId": 0,
  "isActive": true
}
```
[5]

GET/Buyer/GetAllBuyers: recupera o muestra todos los registros de compradores(Buyer).

Request: sin parámetros adicionales.

DELETE/Buyer/DeactivateBuyer: desactiva un registro de comprador usando su identificador.

Request: parámetros FromQuery: "Id": Int.

⇒ Endpoint /Item

Reúne las solicitudes referidas a alta, baja, modificación y consulta de productos (Item).

POST: permite insertar un nuevo registro del tipo producto(Item).

Request: parámetros FromBody (modelo ItemRequest)

```
{
  "name": "string",
  "weight": 0,
  "scent": "string",
  "brand": "string",
  "rawPrice": 0
}
```
[6]

[5] Estructura JSON por defecto de la clase Buyer
[6] Modelo UserRequest utilizado para introducir un nuevo producto (Item)

GET/Item/GetAllItems: recupera o muestra todos los registros de productos (Item).

Request: pasa sin parámetros adicionales.

GET/Item/GetSelectedItem: recupera o muestra un registro usando su identificador (Id).

Request: parámetros FromQuery: "Id": int.

GET/Item/GetItemByBrand: recupera o muestra registros de la clase productos (Item) que comparten el mismo atributo marca (Brand).

Request: parámetros FromQuery: "Brand": string.

DELETE/Items/DeactivateItem: desactiva un registro de un producto (Item) a través de su identificador.

Request: parámetros FromQuery: "Id": int.

DELETE/Items/DeleteItem: elimina el registro de un producto (Item) a través de su identificador.

Request: parámetros FromQuery: "Id": int.

PATCH/Item/UpdateItem: modifica el registro de un producto (Item) recuperando una estructura JSON por defecto de la clase Item. Requiere identificar este registro por defecto con el registro que se desea modificar a través de su "Id" durante la modificación.

```
{
  "id": 0,
  "idWeb": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "name": "string",
  "weight": 0,
  "scent": "string",
  "brand": "string",
  "stock": 0,
  "rawPrice": 0,
  "updateDate": "2023-02-02T18:55:45.158Z",
  "isActive": true,
  "isPublic": true
}
```
[7]

⇒ Endpoint /Order

Reúne las solicitudes referidas a la introducción, modificación y consulta de pedidos (Order).

POST: permite insertar un nuevo registro del tipo pedido (Order).

Request: sin parámetros adicionales.

```
{
  "id": 0,
  "idWeb": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "buyerId": 0,
  "amount": 0,
  "orderDate": "2023-02-02T19:00:05.668Z",
  "deliveryDate": "2023-02-02T19:00:05.668Z",
  "isPaid": true,
  "isShipped": true,
  "isActive": true
}
```
[8]

GET/Order/GetAllOrders: recupera o muestra todos los registros de pedidos (Order)

Request: pasa sin parámetros adicionales.

GET/Order/GetSelectedOrder: recupera o muestra un registro usando su identificador (Id).

[7] Estructura JSON por defecto de la clase Item

[8] Estructura JSON por defecto de la clase Order

Request: parámetros FromQuery: "Id": int.

DELETE/Order/DeactivateOrder: desactiva un registro de un pedido (Order) a través de su identificador.

Request: parámetros FromQuery: "Id": int.

DELETE/Order/DeleteOrder: elimina el registro de un pedido (Order) a través de su identificador.

Request: parámetros FromQuery: "Id": int.

PATCH/Order/UpdateOrder: modifica el registro de un pedido (Order recuperando una estructura JSON por defecto de la clase Order. Requiere identificar este registro por defecto con el registro que se desea modificar a través de su "Id" durante la modificación.

⇒ Endpoint /UserRol

Reúne las solicitudes referidas a la introducción y consulta de roles de usuario (UserRol).

POST: permite insertar un nuevo registro del tipo rol de usuario (UserRol).

Request: sin parámetros adicionales.

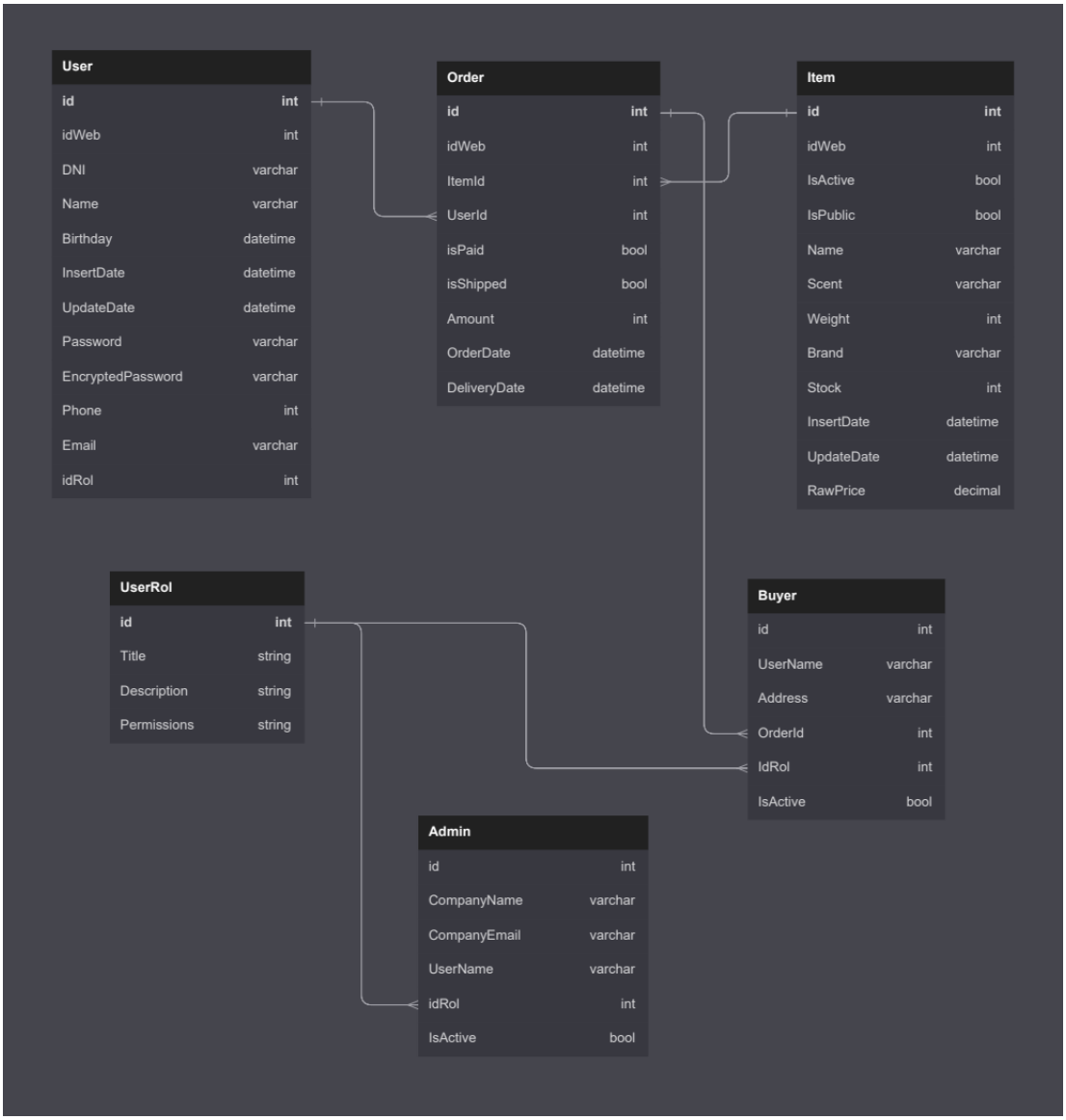
```
{
  "id": 0,
  "title": "string",
  "description": "string",
  "permissions": "string"
}
```
[9]

[9] Estructura JSON por defecto de la clase UserRol

GET/UserRol/GetAllUserRol: recupera o muestra todos los registros de roles de usuario (UserRol)

Request: pasa sin parámetros adicionales.

Anexo



1 Esquema relacional de tablas para el proyecto web API generado con db.diagram