Instituto Politécnico Nacional

Escuela Superior de Cómputo

Unidad de aprendizaje: Desarrollo Web

Profesor: Efren Cruz Carlin

Integrantes:

García Saavedra Armando
López Fabián Jesús Manuel
Orozco Guillén Jorge David
Rodríguez Rojas Isai Neftali

Fecha: 08/01/2024

Documentación del proyecto final

Models

public class Productor

Representa la información de un productor en el sistema.

Propiedades:

- Nombre_Usuario: Nombre de usuario del productor.
- Correo: Correo electrónico del productor.
- Contrasena: Contraseña del productor.
- Nombre: Nombre del productor.
- Apellido_Paterno: Apellido paterno del productor.
- Apellido_Materno: Apellido materno del productor.
- Fecha_nacimiento: Fecha de nacimiento del productor.
- Telefono: Número de teléfono del productor.
- Num_ext: Número exterior del domicilio del productor.
- Calle: Calle del domicilio del productor.
- Ciudad: Ciudad del domicilio del productor.
- Codigo_Postal: Código postal del domicilio del productor.

public class Producto

Representa la información de un producto disponible en el sistema.

Propiedades:

- ID_producto: Identificador único del producto.
- Nombre_Producto: Nombre del producto.
- Descripcion: Descripción del producto.
- Precio: Precio del producto.
- URL: URL de la imagen asociada al producto.

public class Direccion

Propiedades:

- id: Identificador único de la dirección.
- Num_ext: Número exterior de la dirección.
- Calle: Calle de la dirección.
- Ciudad: Ciudad de la dirección.
- Codigo_Postal: Código postal de la dirección.

public class Compra

Representa la información de una compra realizada en el sistema.

Propiedades:

- NumCompra: Número único de la compra.
- FechaCompra: Fecha de la compra.
- Cantidad: Cantidad de productos comprados.
- ID_cliente: Identificador del cliente asociado a la compra.
- ID_productor: Identificador del productor asociado a la compra.
- ID_producto: Identificador del producto comprado.

public class Cliente

Representa la información de un cliente en el sistema.

Propiedades:

- Nombre_Usuario: Nombre de usuario del cliente.
- Correo: Correo electrónico del cliente.
- Contrasena: Contraseña del cliente.
- Nombre: Nombre del cliente.
- Apellido_Paterno: Apellido paterno del cliente.
- Apellido_Materno: Apellido materno del cliente.
- Fecha_nacimiento: Fecha de nacimiento del cliente.

RescueMarket DTOs

DTO_Productor

Representa la información de un productor en el mercado de rescate.

Atributos:

- `Correo`: Correo electrónico del productor.
- `Nombre_Usuario`: Nombre de usuario del productor.
- `Nombre`: Nombre completo del productor.
- `Telefono`: Número de teléfono del productor.
- `Num_ext`: Número exterior de la dirección del productor.
- `Calle`: Nombre de la calle del productor.

- `Ciudad`: Ciudad donde reside el productor.
- `Codigo_Postal`: Código postal de la ubicación del productor.

DTO_Producto

Representa la información de un producto disponible en el mercado de rescate.

Atributos:

- `ID_producto`: Identificador único del producto.
- `Nombre_Producto`: Nombre del producto.
- `Descripcion`: Descripción del producto.
- `Precio`: Precio del producto.
- `URL`: URL relacionada con el producto.

DTO_Compra

Representa la información de una compra realizada en el mercado de rescate.

Atributos:

- `NumCompra`: Número único de la compra.
- `FechaCompra`: Fecha en que se realizó la compra.
- `Cantidad`: Cantidad de productos comprados.
- `ID_producto`: Identificador único del producto comprado.

DTO_Cliente

Representa la información de un cliente en el mercado de rescate.

Atributos:

- `Correo`: Correo electrónico del cliente.
- `Nombre_Usuario`: Nombre de usuario del cliente.
- `Nombre`: Nombre completo del cliente.

Estos Data Transfer Objects (DTOs) están diseñados para facilitar la transferencia de datos entre capas o componentes en una aplicación, proporcionando una estructura clara y coherente para la información relacionada con productores, productos, compras y clientes en el contexto del mercado de rescate.

Controladores

ProductorController

El controlador `ProductorController` proporciona endpoints para gestionar la información de los productores en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /Productores

- Descripción: Obtiene la lista de todos los productores en el mercado.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos Productor.

POST /Productores:

- Descripción: Agrega un nuevo productor al mercado.
- Método HTTP:POST
- Parámetros de Entrada: Productor en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /Productores:

- Descripción: Actualiza la información de un productor existente en el mercado.
- Método HTTP: PATCH
- Parámetros de Entrada: Productor con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /Productores:

- Descripción: Elimina un productor del mercado.
- Método HTTP: DELETE
- Parámetros de Entrada: Productor a ser eliminado en formato JSON (FromBody).

- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia de un productor antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `Productor`.

Este controlador facilita la gestión de productores en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con los productores en el mercado de rescate.

ProductoController

El controlador `ProductoController` proporciona endpoints para gestionar la información de productos en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /Productos:

- Descripción: Obtiene la lista de todos los productos disponibles en el mercado.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos Producto.

POST /Productos:

- Descripción: Agrega un nuevo producto al mercado.
- Método HTTP: POST
- Parámetros de Entrada: Producto en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /Productos:

- Descripción: Actualiza la información de un producto existente en el mercado.
- Método HTTP: PATCH

- Parámetros de Entrada: Producto con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /Productos:

- Descripción: Elimina un producto del mercado.
- Método HTTP: DELETE
- Parámetros de Entrada: Producto a ser eliminado en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia de un producto antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `Producto`.

Este controlador facilita la gestión de productos en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con los productos en el mercado de rescate.

LoginProductorController

El controlador `LoginProductorController` maneja las operaciones relacionadas con el inicio de sesión de los productores en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por este controlador:

POST /LoginProductor:

- Descripción: Permite que un productor inicie sesión verificando las credenciales proporcionadas.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `LoginRequestProductor` en formato JSON (FromBody) que contiene el correo electrónico (`Correo`) y la contraseña (`Contrasena`) del productor.

- Retorno: JsonResult indicando si las credenciales son válidas (true) o no (false).

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- El método `LoginProductor` verifica la existencia de un productor en la base de datos que coincida con el correo electrónico y la contraseña proporcionados.
- Si se encuentra un productor con las credenciales válidas, el método retorna `true`; de lo contrario, retorna `false`.
- Este controlador se utiliza para autenticar a los productores antes de permitirles el acceso a ciertas funcionalidades del sistema.

Este controlador proporciona una interfaz para el inicio de sesión de los productores y es esencial para garantizar la seguridad y el acceso controlado a las operaciones en el mercado de rescate.

LoginClienteController

El controlador `LoginClienteController` se encarga de las operaciones relacionadas con el inicio de sesión de los clientes en el mercado de rescate. Aquí se presenta un resumen de las funcionalidades ofrecidas por este controlador:

POST /LoginCliente:

- Descripción: Permite que un cliente inicie sesión verificando las credenciales proporcionadas.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `LoginRequestCliente` en formato JSON (FromBody) que contiene el correo electrónico (`Correo`) y la contraseña (`Contrasena`) del cliente.
- Retorno: JsonResult indicando si las credenciales son válidas (true) o no (false).

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- El método `LoginCliente` verifica la existencia de un cliente en la base de datos que coincida con el correo electrónico y la contraseña proporcionados.
- Si se encuentra un cliente con las credenciales válidas, el método retorna `true`; de lo contrario, retorna `false`.

- Este controlador se utiliza para autenticar a los clientes antes de permitirles el acceso a ciertas funcionalidades del sistema.

Este controlador proporciona una interfaz para el inicio de sesión de los clientes y es esencial para garantizar la seguridad y el acceso controlado a las operaciones en el mercado de rescate.

DireccionController

El controlador `DireccionController` administra las operaciones relacionadas con las direcciones en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /Direcciones:

- Descripción: Obtiene la lista de todas las direcciones registradas en el sistema.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos `Direccion`.

POST /Direcciones:

- Descripción: Agrega una nueva dirección al sistema.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `Direccion` en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /Direcciones:

- Descripción: Actualiza la información de una dirección existente en el sistema.
- Método HTTP: PATCH
- Parámetros de Entrada: Objeto `Direccion` con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /Direcciones:

- Descripción: Elimina una dirección del sistema.
- Método HTTP: DELETE

- Parámetros de Entrada: Objeto `Direccion` a ser eliminado en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia de una dirección antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `Direccion`.

Este controlador facilita la gestión de direcciones en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con las direcciones en el mercado de rescate.

CompraController

El controlador `CompraController` maneja las operaciones relacionadas con las compras en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /Compras:

- Descripción: Obtiene la lista de todas las compras registradas en el sistema.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos `Compra`.

POST /Compras:

- Descripción: Agrega una nueva compra al sistema.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `Compra` en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /Compras:

- Descripción: Actualiza la información de una compra existente en el sistema.
- Método HTTP: PATCH
- Parámetros de Entrada: Objeto `Compra` con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /Compras:

- Descripción: Elimina una compra del sistema.
- Método HTTP: DELETE
- Parámetros de Entrada: Objeto `Compra` a ser eliminado en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia de una compra antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `Compra`.

Este controlador facilita la gestión de compras en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con las compras en el mercado de rescate.

ClienteController

El controlador `ClienteController` gestiona las operaciones relacionadas con los clientes en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /Clientes:

- Descripción: Obtiene la lista de todos los clientes registrados en el sistema.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos `Cliente`.

POST /Clientes:

- Descripción: Agrega un nuevo cliente al sistema.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `Cliente` en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /Clientes:

- Descripción: Actualiza la información de un cliente existente en el sistema.
- Método HTTP: PATCH
- Parámetros de Entrada: Objeto `Cliente` con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /Clientes:

- Descripción: Elimina un cliente del sistema.
- Método HTTP: DELETE
- Parámetros de Entrada: Objeto `Cliente` a ser eliminado en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia de un cliente antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `Cliente`.

Este controlador facilita la gestión de clientes en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con los clientes en el mercado de rescate.

DTOProductorController

El controlador `DTOProductorController` gestiona las operaciones relacionadas con los Productores en el mercado de rescate utilizando DTOs (Data Transfer Objects). A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /DTO_Productores:

- Descripción: Obtiene la información de un Productor en formato DTO utilizando su dirección de correo electrónico.
- Método HTTP: GET
- Parámetros de Consulta: Correo electrónico (`correo`) del Productor.
- Retorno: JsonResult que contiene el objeto `DTO_Productor`.

POST /DTO_Productores:

- Descripción: Agrega un nuevo Productor en formato DTO al sistema.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `DTO_Productor` en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /DTO_Productores:

- Descripción: Actualiza la información de un Productor en formato DTO existente en el sistema.
- Método HTTP: PATCH
- Parámetros de Entrada: Objeto `DTO_Productor` con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /DTO_Productores:

- Descripción: Elimina un Productor en formato DTO del sistema.
- Método HTTP: DELETE
- Parámetros de Entrada: Correo electrónico del Productor en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia de un Productor antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `DTO_Productor`.

Este controlador facilita la gestión de Productores en formato DTO en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con los Productores en el mercado de rescate.

DTOProductoController

El controlador `DTOProductoController` se encarga de gestionar las operaciones relacionadas con los Productos en formato DTO (Data Transfer Objects) en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por el único método presente:

GET /DTO_Productos:

- Descripción: Obtiene la lista de Productos en formato DTO.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos `DTO_Producto`.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- El método hace uso de Entity Framework Core para obtener la lista de productos en formato DTO.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `DTO_Producto`.

Este controlador facilita la obtención de información sobre los productos en formato DTO en el sistema, permitiendo a otras partes del sistema consumir esta información de manera eficiente. Es útil especialmente cuando se requiere una representación simplificada de los productos en comparación con la entidad completa del modelo.

DTOCompraController

El controlador `DTOCompraController` gestiona las operaciones relacionadas con las Compras en formato DTO (Data Transfer Objects) en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por el único método presente:

GET /DTOCompras:

- Descripción: Obtiene la lista de Compras en formato DTO.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos `DTO_Compra`.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- El método hace uso de Entity Framework Core para obtener la lista de compras en formato DTO.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `DTO_Compra`.

Este controlador facilita la obtención de información sobre las compras en formato DTO en el sistema, permitiendo a otras partes del sistema consumir esta información de manera eficiente. Es útil especialmente cuando se requiere una representación simplificada de las compras en comparación con la entidad completa del modelo.

DTOClienteController

El controlador `DTOClienteController` gestiona las operaciones relacionadas con los Clientes en formato DTO (Data Transfer Objects) en el mercado de rescate. A continuación, se presenta un resumen de las funcionalidades ofrecidas por cada método:

GET /DTOClientes/GetClientesDTO:

- Descripción: Obtiene la lista de Clientes en formato DTO.
- Método HTTP: GET
- Retorno: JsonResult que contiene la lista de objetos `DTO_Cliente`.

GET /DTOClientes?correo={correo}:

- Descripción: Obtiene la información de un Cliente en formato DTO utilizando su dirección de correo electrónico.
- Método HTTP: GET
- Parámetros de Consulta: Correo electrónico (`correo`) del Cliente.
- Retorno: JsonResult que contiene el objeto `DTO_Cliente`.

GET /DTOClientes/GetClienteDTO_USER?user_name={user_name}:

- Descripción: Obtiene la información de un Cliente en formato DTO utilizando su nombre de usuario.
- Método HTTP: GET
- Parámetros de Consulta: Nombre de usuario (`user_name`) del Cliente.
- Retorno: JsonResult que contiene el objeto `DTO_Cliente`.

POST /DTOClientes:

- Descripción: Agrega un nuevo Cliente en formato DTO al sistema.
- Método HTTP: POST
- Parámetros de Entrada: Objeto `DTO_Cliente` en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación fue exitosa.

PATCH /DTOClientes?correo={correo}:

- Descripción: Actualiza la información de un Cliente en formato DTO existente en el sistema.
- Método HTTP: PATCH
- Parámetros de Consulta: Correo electrónico (`correo`) del Cliente.
- Parámetros de Entrada: Objeto `DTO_Cliente` con la información actualizada en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de actualización fue exitosa.

DELETE /DTOClientes:

- Descripción:Elimina un Cliente en formato DTO del sistema.
- Método HTTP: DELETE
- Parámetros de Entrada: Objeto `DTO_Cliente` en formato JSON (FromBody).
- Retorno: JsonResult indicando si la operación de eliminación fue exitosa.

Notas Adicionales:

- Se utiliza la clase `RMContext` para interactuar con la base de datos.
- Los métodos hacen uso de Entity Framework Core para realizar operaciones CRUD en la base de datos.
- En algunos métodos, se comprueba la existencia del Cliente antes de realizar la operación correspondiente.
- La información se transfiere entre la base de datos y el controlador mediante objetos de la clase `DTO_Cliente`.

Este controlador facilita la gestión de Clientes en formato DTO en el sistema, permitiendo la obtención, inserción, actualización y eliminación de información relacionada con los Clientes en el mercado de rescate.

Context

El contexto `RMContext` define el modelo de datos y la configuración de la base de datos para el mercado de rescate. A continuación, se presenta un resumen de las entidades y configuraciones del modelo:

Entidades Principales:

- Clientes: Representa a los clientes del mercado.
- Compras: Representa las compras realizadas en el mercado.
- Direcciones: Representa las direcciones asociadas a usuarios del mercado.
- Productos: Representa los productos disponibles en el mercado.
- Productores: Representa a los productores que ofrecen productos en el mercado.

Entidades DTO:

- DTO_Cliente: Representa la información del cliente en formato DTO.
- DTO_Compra: Representa la información de compra en formato DTO.
- DTO_Producto: Representa la información del producto en formato DTO.
- DTO_Productor: Representa la información del productor en formato DTO.

Configuraciones del Modelo:

- Clientes: Definida con claves y propiedades específicas.
- Direcciones: Definida con propiedades específicas.
- Productos: Definida con claves y propiedades específicas.
- Productores: Definida con claves y propiedades específicas.
- Compras: Definida con claves y propiedades específicas.
- DTO_Cliente: Definida con claves y propiedades específicas.
- DTO_Productor: Definida con claves y propiedades específicas.
- DTO_Compra: Definida con claves y propiedades específicas.
- DTO_Producto: Definida con claves y propiedades específicas.

Configuración de la Conexión a la Base de Datos:

- Utiliza MySQL como sistema de gestión de bases de datos.
- Conexión especificada en el método `OnConfiguring` del contexto.

Notas Adicionales:

- Se utilizan claves primarias, propiedades y configuraciones específicas para cada entidad.
- Las entidades y configuraciones se han definido utilizando las capacidades de Fluent API de Entity Framework Core.
- El contexto `RMContext` extiende la clase `DbContext` para proporcionar la funcionalidad necesaria para interactuar con la base de datos.
- Las propiedades de cada entidad mapean directamente a columnas en las tablas de la base de datos.

Este contexto facilita el acceso y la manipulación de datos relacionados con clientes, compras, direcciones, productos y productores en el mercado de rescate, así como la gestión de estas entidades en formato DTO.