

RESTful Web Service



Descripción general

La aplicación contiene la funcionalidad para poder insertar, actualizar, eliminar y consultar los siguientes componentes:

Objeto	Descripción
Person	Contiene los datos de identidad (nombre, apellido, correo, fecha de ingreso, registroactivo). Este es el registro principal, sin el identificador PersonId no se permitiran registrar los componentes siguientes.
Address	Contiene los datos de ubicación/dirección de la persona registrada, depende del registro principal (person) para poder relacionar persona-dirección con el identificador PersonId.
Product	Contiene los datos para ingresar un producto relacionado con el registro persona, de igual forma, depende del registro principal (person) para poder relacionar persona-producto con el identificador PersonId.

Funcionamiento



El registro completo de estos datos se realiza utilizando el controlador PersonCompleteController en una sola operación, el cual tiene los distintos métodos para consultar uno o todos los registros de tipo person-address-product.

Para esto, utiliza un DataTransferObject (DTO) el cual contiene los 3 objetos a recibir, esto para poder manejar cada una de las operaciones (get, post, put y delete).

GET

Al consultar la ruta **/api/personscomplete** obtendremos una respuesta compuesta de la siguiente forma:

Search Postman

GET GET PersonsCom... GET GET PersonCompl... POST POST PersonCo... PUT PUT PersonCompl... DELETE DELETE PersonCo... + ... No Environment

PersonsProductsAPI / PersonComplete / GET PersonsComplete

GET https://localhost:44346/api/personscomplete Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (10) Test Results Status: 200 OK Time: 6 ms Size: 1.74 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "person": {
3     "id": 18,
4     "firstName": "Peter",
5     "middleName": null,
6     "lastName": "Peterson",
7     "email": "peterpet5@gmail.com",
8     "dateAdded": "2017-06-15T00:00:00",
9     "isActive": true
10  },
11  "address": {
12    "id": 12,
13    "addressLine1": "211bis, rue des Peupliers",
14    "addressLine2": null,
15    "stateOfProvince": "Yveline",
16    "city": "Versailles ",
17    "postalCode": "78100",
18    "personId": 18
19  },
20  "product": {
21    "id": 16,
```

POST

Para hacer la carga de un registro completo, se requiere recibir un objeto JSON de esta forma:

The screenshot shows the Postman interface with a POST request to `https://localhost:44346/api/personscomplete`. The request body is a JSON object:

```
1 {
2   "Person": {
3     "firstName": "Christian",
4     "middleName": null,
5     "lastName": "Northwind",
6     "email": "chrisnorth2@gmail.com",
7     "dateAdded": "2017-05-13T00:00:00",
8     "isActive": true
9   },
10  "Address": {
11    "addressLine1": "2, rue Lamarck",
12    "addressLine2": null,
13    "stateOfProvince": "Seine Saint Denis",
14    "city": "Saint-Denis ",
15    "postalCode": "93400"
16  },
17  "Product": {
18    "name": "HL Road Frame -- Red, 58",
19    "number": "FR-R92R-58",
20    "numberInStock": 20,
21    "standardCost": 1059.31
22  }
23 }
```

The response is a 200 OK status with a message: `"Record was added successfully!"`. The status bar at the bottom shows the time as 10:52 a.m. on 07/05/2021.

Objeto JSON:



```
{
  "Person": {
    "firstName": "Christian",
    "middleName": null,
    "lastName": "Northwind",
    "email": "chrisnorth2@gmail.com",
    "dateAdded": "2017-05-13T00:00:00",
    "isActive": true
  },
  "Address": {
    "addressLine1": "2, rue Lamarck",
    "addressLine2": null,
    "stateOfProvince": "Seine Saint Denis",
    "city": "Saint-Denis ",
    "postalCode": "93400"
  },
}
```

```

"Product":{
"name": "HL Road Frame - Red, 58",
"number": "FR-R92R-58",
"numberInStock": 20,
"standardCost": 1059.31
}
}

```



Podemos notar que es un objeto compuesto de las tres entradas distintas (**person-address-product**), una vez recibiendo la entrada, **llamará a cada uno de los servicios encontrados como PersonRepository, AddressRepository y ProductRepository**, los cuales **contienen las distintas operaciones para crear, actualizar y eliminar** cada uno de los elementos.

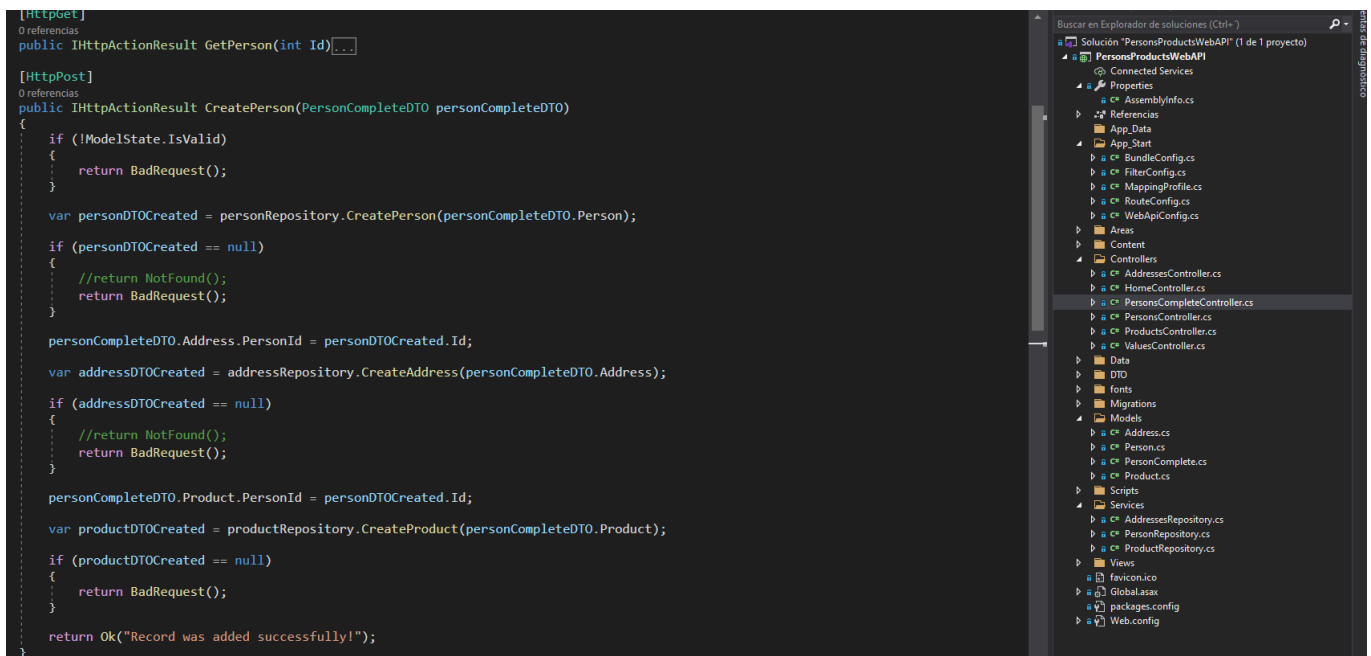
```

19
20
21 1 referencia
22 public AddressDTO GetAddress(int Id)
23 {
24     var addressInDb = _context.Addresses.SingleOrDefault(p => p.PersonId == Id);
25
26     if (addressInDb == null)
27     {
28         return null;
29     }
30     return Mapper.Map<Address, AddressDTO>(addressInDb);
31 }
32
33 1 referencia
34 public AddressDTO CreateAddress(AddressDTO addressDTO)
35 {
36     var addressInDb = Mapper.Map<AddressDTO, Address>(addressDTO);
37     _context.Addresses.Add(addressInDb);
38     _context.SaveChanges();
39
40     addressDTO.Id = addressInDb.Id;
41     return addressDTO;
42 }
43
44 1 referencia
45 public AddressDTO UpdateAddress(int Id, AddressDTO addressDTO)
46 {
47     var addressInDb = _context.Addresses.SingleOrDefault(p => p.PersonId == Id);
48
49     if (addressInDb == null)
50     {
51         return null;
52     }
53
54     Mapper.Map<AddressDTO, Address>(addressDTO, addressInDb);
55     _context.SaveChanges();
56     addressDTO.Id = addressInDb.Id;
57
58 }

```

En la instrucción POST, **insertará los datos de identidad de primera instancia, ya que sin este registro, no se podrán realizar las inserciones de los objetos tipo address ni product.**

Dentro del método POST se consumen cada uno de los servicios mencionados para poder insertar los componentes:



Debido a esto, es posible operar con las distintas funciones de REST: GET, POST, PUT y DELETE.