МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования Российской Федерации
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»
«ИНСТИТУТ УПРАВЛЕНИЯ И ЦИФРОВЫХ ТЕХНОЛОГИЙ»

Кафедра: «Цифровые технологии управления транспортными процессами»

Отчёт
По дисциплине: «Сервис-ориентированное программирование»
Hypertext Application Language.

Выполнил: Каюн М. Р.
УВП-411

Принял:
ст. п., Заманов Е. А.
асс., Афонина А. А.

Москва 2022

Оглавление

1. Задание.

Развить веб-приложение Auto, расширив его путём добавления новой сущности «владелец».

Продемонстрировать работу интерфейсов, отдающих RESTful ответы.

2. Выполнение

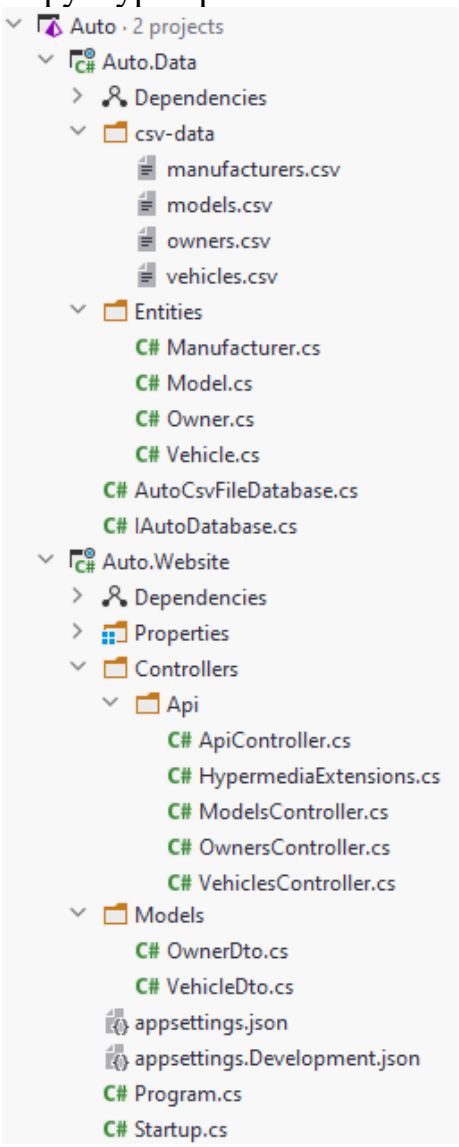
Hypertext Application Language (далее – HAL) – это стандарт для определения гипермедиа, таких как ссылки на внешние ресурсы в форматах JSON или XML.

Вся логика реализована в контроллерах без вынесения в сервисный слой. Контроллер получает DTO (Data transfer object). Контроллер может возвращать данные, используя HAL и пагинацию. При изменении или добавлении объекта «владелец» для передачи данных используется DTO, в остальных случаях ответ отдаётся в соответствии со стандартом HAL.

Для минимизации данных полей «actions» в ответе, все запросы принимаются с типом POST. Это позволяет упразднить структуру ответа, не указывая дополнительно тип запроса.

В контроллере реализован базовый CRUD-функционал.

Структура проекта:



В решении присутствуют два проекта. Auto.Data – это проект, предоставляющий контекст для работы с данными, которые хранятся в csv-файлах. В ходе выполнения был добавлен файл owners.csv. Формат записей в файле: Антонов,Александр,Викторович,barlow@hotmail.com,AA07AMM

Где:

1) Антонов,Александр,Викторович – ФИО владельца;
2) barlow@hotmail.com – адрес электронной почты владельца;
3) AA07AMM – регистрационный номер транспортного средства, которое выставлено на продажу владельцем на нашем сервисе. Поле может быть пустым, заполнение необязательно.

В контексте AutoCsvFileDatabase немного скорректирован метод поиска транспортного средства. В случае неудачного поиска генерируется исключение с сообщением о том, что транспортное средство не найдено.

```
public Vehicle FindVehicle(string registration)
{
    var vehicle = vehicles.FirstOrDefault(e :KeyValuePair<string,Vehicle> => e.Key == registration).Value;
    if (vehicle == default)
    {
        throw new Exception( message: $"Авто с номером {registration} не найдено");
    }

    return vehicle;
}
```

Добавлена строка заполнения данных о владельцах из файла.

```
public AutoCsvFileDatabase(ILogger<AutoCsvFileDatabase> logger)
{
    this.logger = logger;
    ReadManufacturersFromCsvFile( filename: "manufacturers.csv");
    ReadModelsFromCsvFile( filename: "models.csv");
    ReadVehiclesFromCsvFile( filename: "vehicles.csv");
    ReadOwnersFromCsvFile( filename: "owners.csv");
    ResolveReferences();
}
```

В качестве ключа используется ФИО с разделителем &.

```
1 usage
private void ReadOwnersFromCsvFile(string filename)
{
    var filePath :string = ResolveCsvFilePath(filename);
    foreach (var line :string in File.ReadAllLines(filePath))
    {
        var tokens :string[] = line.Split( separator: ",");
        var owner = new Owner( firstName: tokens[0],  middleName: tokens[1],  lastName: tokens[2],  email: tokens[3]);

        var vehicle = this.vehicles // Dictionary<string,Vehicle>
            .FirstOrDefault(e :KeyValuePair<string,Vehicle> => tokens[4] == e.Key).Value;
        owner.Vehicle = vehicle;

        owners[owner.GetFullName] = owner;
    }
}
```

```
[Newtonsoft.Json.JsonIgnore]
11 usages
public string GetFullName => $"{FirstName}&{MiddleName}&{LastName}";
```

В интерфейсе контекста выставлены контракты для манипулирования данными о владельцах.

```
2 usages  1 implementation
public void CreateOwner(Owner owner);
2 usages  1 implementation
public void UpdateOwner(Owner owner, string name);
2 usages  1 implementation
public void DeleteOwner(Owner owner);
```

Создано отдельное Dto для добавления и обновления «владельца»

```
3 usages
public class OwnerDto
{
    public OwnerDto()
    {
    }
    public OwnerDto(string firstName, string middleName, string lastName, string email, string regCodeVehicle = null)
    {
        FirstName = firstName;
        MiddleName = middleName;
        LastName = lastName;
        Email = email;
        RegCodeVehicle = regCodeVehicle;
    }
    4 usages
    public string FirstName { get; set; }
    4 usages
    public string MiddleName { get; set; }
    4 usages
    public string LastName { get; set; }

    3 usages
    public string Email { get; set; }

    5 usages
    public string? RegCodeVehicle { get; set; }

    [Newtonsoft.Json.JsonIgnore]
    1 usage
    public string GetFullName => $"{FirstName}&{MiddleName}&{LastName}";
}
```

Создан контроллер.

```
[Route( template: "api/[controller]")]
[ApiController]
public class OwnersController : ControllerBase
{
    private readonly IAutoDatabase _context;

    public OwnersController(IAutoDatabase context){...}

    [HttpPost]
    [Produces( contentType: "application/hal+json")]
    public async Task<IActionResult> Get(int index = 0, int count = 10){...}

    [HttpPost]
    [Produces( contentType: "application/hal+json")]
    [Route( template: "{name}")]
    1 usage
    public async Task<IActionResult> GetByName(string name){...}

    [HttpPost]
    [Produces( contentType: "application/hal+json")]
    [Route( template: "add")]
    public async Task<IActionResult> Add([FromBody] OwnerDto ownerDto){...}

    [HttpPost]
    [Produces( contentType: "application/hal+json")]
    [Route( template: "delete/{name}")]
    public async Task<IActionResult> Remove(string name){...}

    [HttpPost]
    [Produces( contentType: "application/hal+json")]
    [Route( template: "update/{name}")]
    public async Task<IActionResult> Update(string name, [FromBody] OwnerDto owner){...}

    private Vehicle ParseVehicle(dynamic href){...}

    1 usage
    private Owner CreateOwner(OwnerDto owner, Vehicle vehicle){...}

    3 usages
    private dynamic GetResource(Owner owner){...}

    2 usages
    private dynamic GetResource(Owner owner, string name = null){...}

    1 usage
    private dynamic Paginate(string url, int index, int count, int total){...}
}
```

Разбор реализации отдачи HAL. Рассмотрение метода GetByName.

```
public async Task<IActionResult> GetByName(string name)
{
    dynamic result;
    try
    {
        var item:dynamic = GetResource(_context.FindOwnerByName(name), name);
        if (item == null)
        {
            throw new Exception(message:"Владелец с таким именем не найден");
        }
        var total:int = _context.CountOwners();
        result = new
        {
            total,
            item
        };
        return Ok(result);
    }
    catch (Exception e)
    {
        result = new {message = e.Message};
    }

    return BadRequest(result);
}
```

С самого начала создаётся переменная result под ответ. Если в ходе выполнения что-то пошло не так, то в эту переменную будет записано сообщение об ошибке.

Весь процесс поиска происходит в блоке try. В item присваивается ресурс, содержащий всю необходимую информацию об искомом объекте.

Вся логика скрыта в методе GetResource.

```
2 usages
private dynamic GetResource(Owner owner, string name = null)
{
    if (name != null && owner.GetFullName != name)
    {
        return null;
    }
    var pathOwner = "/api/owners/";
    var pathVehicle = "/api/vehicles/";
    var ownerDynamic :dynamic = owner.ToDynamic();

    dynamic links = new ExpandoObject();
    links.self = new
    {
        href = $"{pathOwner}{owner.GetFullName}"
    };
    if (owner.Vehicle != null)
        links.vehicle = new
        {
            href = $"{pathVehicle}{owner.Vehicle.Registration}"
        };

    ownerDynamic._links = links;
    ownerDynamic.actions = new
    {
        update = new
        {
            href = $"/api/owners/update",
            accept = "application/json"
        },
        delete = new
        {
            href = $"/api/owners/delete/{owner.GetFullName}"
        }
    };
    return ownerDynamic;
}
```

При формировании структуры ответа проверяется, есть ли у владельца авто, выставленное на продажу, если его нет, то поле vehicle в структуру ответа не добавляется.

При запросе нескольких владельцев используется пагинация.

```
[HttpPost]
[Produces(contentType: "application/hal+json")]
public async Task<IActionResult> Get(int index = 0, int count = 10)
{
    dynamic result;
    try
    {
        var items :IEnumerable<dynamic> = _context.ListOwners().Skip(index).Take(count).Select(GetResource);
        var total :int = _context.CountOwners();
        var links :dynamic = Paginate(url: "/api/owners", index, count, total);

        result = new
        {
            links, index, count, total, items
        };
        return Ok(result);
    }
    catch (Exception e)
    {
        result = new {message = e.Message};
    }

    return BadRequest(result);
}
```

Метод Paginate производит «расстраничивание» списка, позволяя не выгружать целиком все данные, которые у нас есть, а передавать клиенту лишь часть из них с указанием ссылок на другие страницы.

Метод Add выполняет добавление владельца и проверяет данные о транспортном средстве на корректность. Если указанного авто нет в базе, то регистрации нового владельца не происходит. Если добавляется владелец без указания авто, то регистрация происходит без связывания с конкретным транспортным средством.

```
[HttpPost]
[Produces(contentType: "application/hal+json")]
[Route(template: "add")]
public async Task<IActionResult> Add([FromBody] OwnerDto ownerDto)
{
    dynamic result;
    try
    {
        Vehicle vehicle = null;
        if (!string.IsNullOrEmpty(ownerDto.RegCodeVehicle))
        {
            vehicle = _context.FindVehicle(ownerDto.RegCodeVehicle);
        }

        var ownerInContext = _context.FindOwnerByName(ownerDto.GetFullName);
        if (ownerInContext == null)
        {
            Owner newOwner = CreateOwner(ownerDto, vehicle);
            result = new
            {
                message = "Создан новый владелец",
                owner = GetResource(newOwner)
            };
            return Ok(result);
        }
        result = new { message = "Владелец с таким именем уже существует", owner = GetResource(ownerInContext) };
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        throw;
    }

    return BadRequest(result);
}
```

Удаление, метод Remove.

```
[HttpPost]
[Produces(contentType: "application/hal+json")]
[Route(template: "delete/{name}")]
public async Task<IActionResult> Remove(string name)
{
    var owner = _context.FindOwnerByName(name);
    _context.DeleteOwner(owner);
    return Ok(owner);
}
```

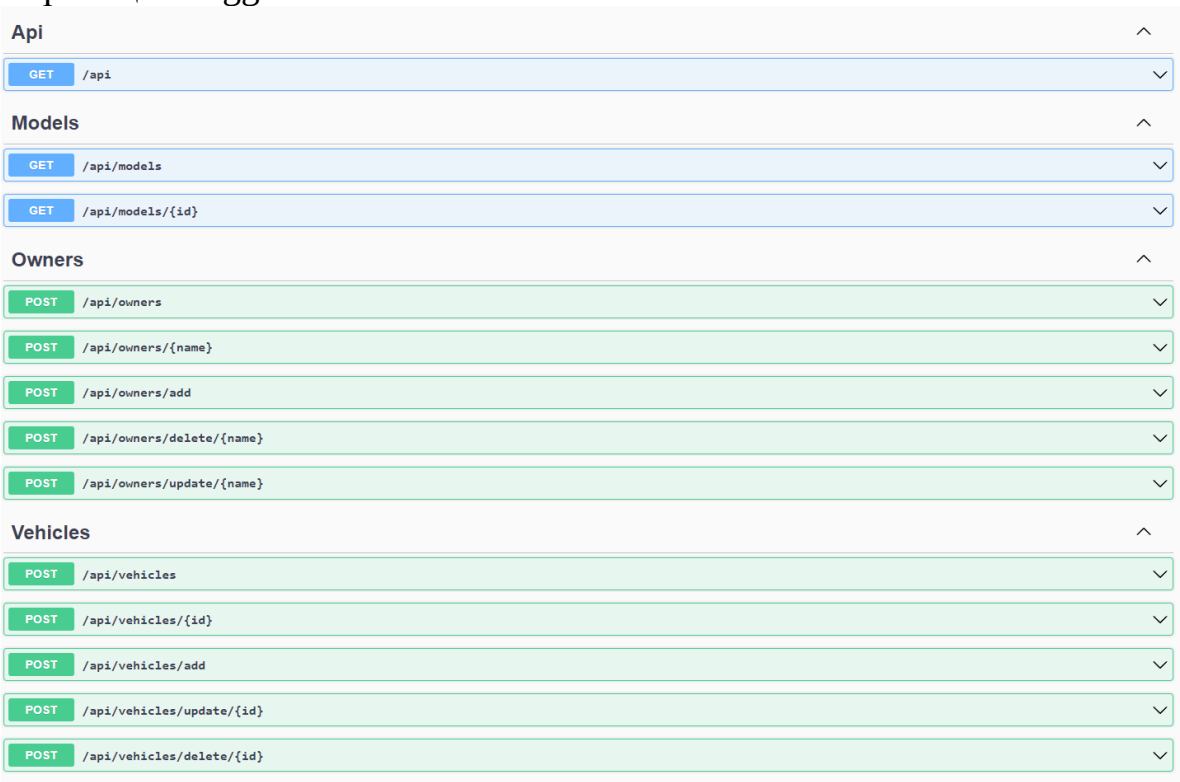
Обновление, метод Update.

```
[HttpPost]
[Produces(contentType: "application/hal+json")]
[Route(template: "update/{name}")]
public async Task<IActionResult> Update(string name, [FromBody] OwnerDto owner)
{
    dynamic result;
    try
    {
        Vehicle vehicle = null;
        if (!string.IsNullOrEmpty(owner.RegCodeVehicle))
        {
            vehicle = _context.FindVehicle(owner.RegCodeVehicle);
        }

        var ownerInContext =
            _context.FindOwnerByName(name);
        if (ownerInContext == null)
        {
            result = new
            {
                message = "Такого владельца нет. Воспользуйтесь методом add",
            };
            return BadRequest(result);
        }

        var oldName:string = ownerInContext.GetFullName;

        ownerInContext.FirstName = owner.FirstName;
        ownerInContext.MiddleName = owner.MiddleName;
        ownerInContext.LastName = owner.LastName;
        ownerInContext.Email = owner.Email;
        ownerInContext.Vehicle = vehicle;

        _context.UpdateOwner(ownerInContext, oldName);

        return await GetByName(ownerInContext.GetFullName);
    }
    catch (Exception e)
    {
        result = new {message = e.Message};
    }

    return BadRequest(result);
}
```

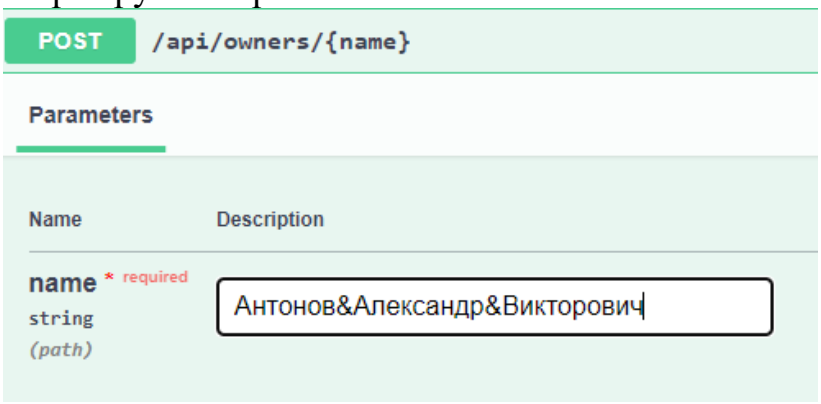
В методе Update производится проверка на корректность введенных данных об авто. Если такого авто нет, то обновления данных не происходит, клиенту возвращается ошибка. Если данные корректны, то происходит обновление данных, причём клиент должен указать старый ключ сущности, поскольку ключ формируется из полей объекта, а поля могут измениться. Для корректной работы обновления, необходимо знать старый ключ и в случае необходимости – перезаписать его в словаре контекста.

3. Демонстрация работы.

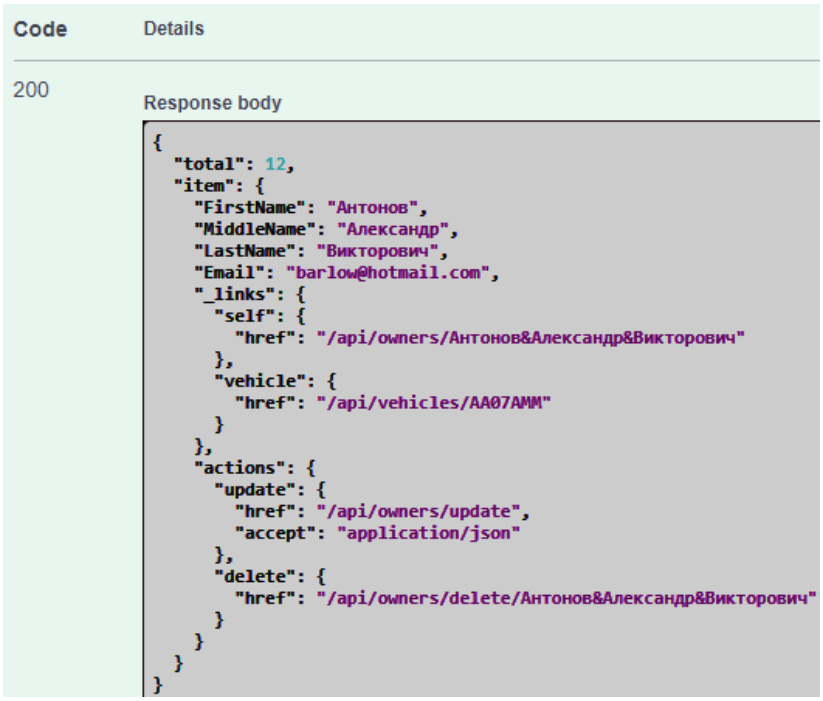
Страница swagger.



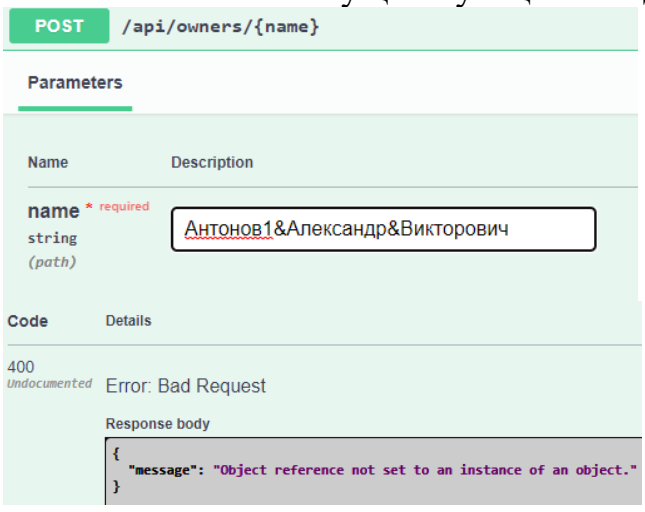
Попробуем извлечь одного владельца и посмотрим на структуру.
Формируем запрос:



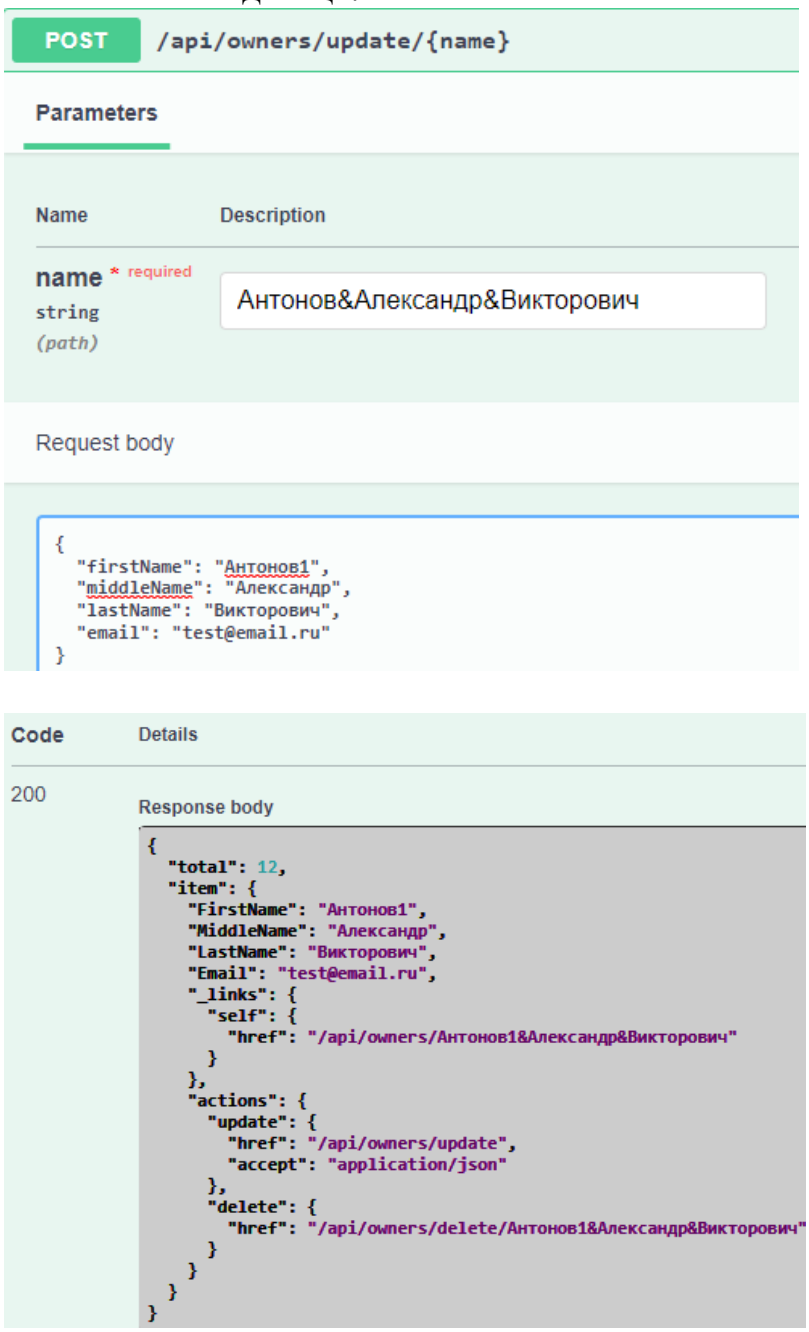
Ответ:



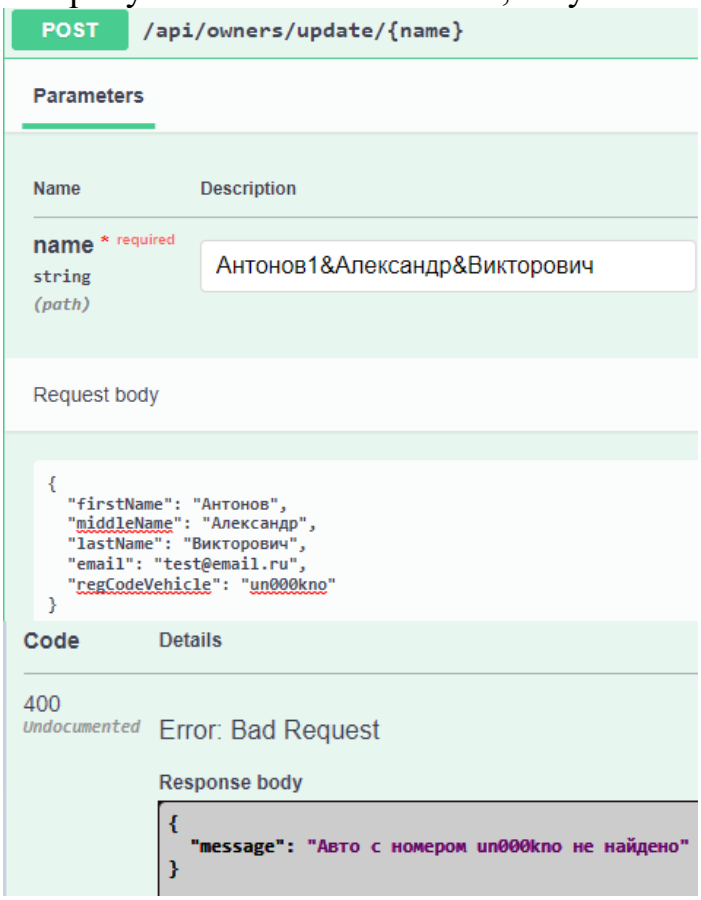
Попытка извлечь несуществующего владельца:



Изменение владельца.



Попробуем изменить его снова, но укажем несуществующий номер авто.



Проверим, данные о владельце не должны были измениться: