DATAART

SPB .NET Meetup #8

Topic

# RESTful API

## Best practices, versioning, design documentation

Speaker

Vyacheslav Mikhaylov (vmikhaylov@dataart.com)

# О чем доклад?

- Что такое API и зачем он нужен?
- Основы REST
- На чем реализовывать?
- Бест-практики
- Как проектировать?
- Документация и версионирование

DATAART

# Что такое API?

**A**plication
**P**rogram
**I**nterface

Набор правил и механизмов

(на самом деле это все знают)

DATAART

# Почему хороший API это важно?

- Простота использования и поддержки
- Конверсия в среде разработчиков (потребителей).
- Больше бользователей ☺ API ->  выше популярность сервиса
- Лучше структура -> лучше изоляция компонентов
- API это UI для разработчиков

DATAART

# Какие виды API бывают?

- Web service APIs
  - XML-RPC and JSON-RPC
  - SOAP
  - REST
- WebSockets APIs
- Library-based APIs
  - Java Script
- Class-based APIs
  - C# API, Java
- OS function and routines
  - Access to file system
  - Access to user interface
- Object remoting APIs
  - CORBA
  - .Net remoting
- Hardware APIs
  - Video acceleration (OpenCL…)
  - Hard disk drives
  - PCI bus
  - …

DATAART

# Какие виды API нас интересуют?

Web service APIs

- XML-RPC and JSON-RPC
- SOAP – Simple ☺ Object Acces Protocol
- REST

DATAART

# Что такое REST?

**Re**presentative
**S**tate
**T**ransfer

Это не протокол. И не стандарт. Это архитектурный стиль

(это тоже все знают)

DATAART

# Принципы REST?

- Клиент-серверная архитектура
- Любые данные являются ресурсом
- Любой ресурс имеет ID
- Ресурсы связаны между собой
- Используются стандартные методы HTTP
- Сервер не хранит состояние

DATAART

# Чем REST хорош?

- Он простой!
- Переиспользуем существующие стандарты
- REST базируется на HTTP => доступны все плюшки
  - Кеширование
  - Масштабирование
  - Минимум накладных расходов
  - Стандартные коды ошибок
- Очень хорошая распространённость (даже IoT)

DATAART

# Best-practices (независимые от технологий)

- SSL everywhere
- Documentation & Versioning
- POST, PUT should return data
- Filtering, sorting, pagination
- Support MediaType
- Pretty print & gzip
- Standard caching by ETag & Last-Modified
- Use standard error codes and predefined error format

DATAART

# Свойства HTTP методов

| HTTP Method | Idempotent | Safe |
|---|---|---|
| OPTIONS | Yes | Yes |
| GET | Yes | Yes |
| HEAD | Yes | Yes |
| PUT | Yes | No |
| POST | No | No |
| DELETE | Yes | No |
| PATCH | Yes | No |

DATAART

# Что такое RESTful API?

Это такой сервис, который удовлетворяет принципам REST


DATAART

# Выбираем технологию

## WCF Services

- – webHttpBinding only(а зачем тогда остальные?)
- – Поддерживаются только HTTP Get & POST (и все)
- + Разные форматы XML, JSON, ATOM

## Web Api

- + Очень простой
- + Open source
- + Все возможности HTTP
- + Все возможности MVC
- + Легкий (не жирный ☺)
- + Тоже поддерживает кучу форматов

DATAART

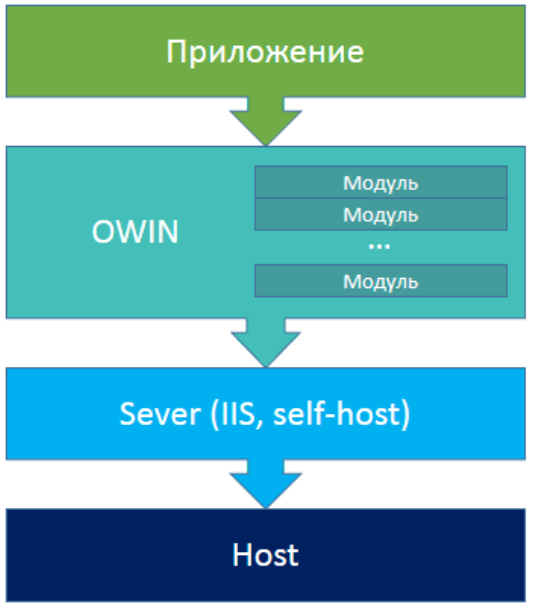
# Выбираем хостинг для WebApi

- ASP.NET MVC
- OWIN – Open Web Interface for .Net
  - IIS
  - Self-hosted
- Azure

DATAART

# Идея OWIN

- Это спецификация (не библиотека и не платформа)
- Устраняет сильную связанность веб приложения с реализацией сервера




DATAART

# Katana – реализация OWIN от Microsoft

```
[assembly: OwinStartup(typeof (Startup))]
namespace RestApiDemo
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            var config = new HttpConfiguration();
            config.MapHttpAttributeRoutes();
            app.UseWebApi(config);
        }
    }
}
```

DATAART

# Проектируем интерфейс

- Все ресурсы в REST существительные (множественное число)
- Корневые сущности API
  - GET /stations - Все вокзалы
  - GET /stations/123  - Информация по вокзалу с ID = 123
  - GET /trains - Все поезда
- Зависимые сущности
  - GET /stations/555/departures – поезда уходящие с вокзала 555

# Простейший контроллер

```
[RoutePrefix("stations")]
public class RailwayStationsController : ApiController
{
    [HttpGet]
    [Route]
    public IEnumerable<RailwayStationModel> GetAll()
    {
        return testData;
    }

    RailwayStationModel[] testData = /*initialization here*/
}
```

DATAART

# “Много” данных?

- 100?
- 1000?
- 1000000?

> 100 редко нужно на клиенте

DATAART

# OData (www.odata.org)

```
[RoutePrefix("stations")]
public class RailwayStationsController : ApiController
{
    [HttpGet]
    [Route]
    [EnableQuery]
    public IQueryable<RailwayStationModel> GetAll()
    {
        return testData.AsQueryable();
    }

    RailwayStationModel[] testData = /*initialization here*/
}
```

DATAART

# Параметры запросов

| Query Option | Sample |
| --- | --- |
| $filter | Stations?$filter=Name eq 'Московский вокзал'<br>Stations?$filter=contains(Name, 'Лад') |
| $select | Stations?$select=Name, Id |
| $orderby | Stations?$orderby=Name desc |
| $top | Trains?$top=40. |
| $skip | Trains?$skip=1000&$top=40 |

DATAART

# EnableQuery Attribute

- AllowedArithmeticOperators
- AllowedFunctions
- AllowedLogicalOperators
- AllowedOrderByProperties
- AllowedQueryOptions
- EnableConstantParameterization
- EnsureStableOrdering
- HandleNullPropagation
- MaxAnyAllExpressionDepth
- MaxExpansionDepth
- MaxNodeCount
- MaxOrderByNodeCount
- MaxSkip
- MaxTop
- PageSize

# Примеры запросов REST

- GET /stations– получить все вокзалы
- GET /trains – расписание всех поездов
- GET /stations/555/arrivals
- GET /stations/555/departures

# Зависимый контроллер

```
[RoutePrefix("stations/{station}/departures")]
public class TrainsFromController : TrainsController
{
    [HttpGet]
    [Route]
    [EnableQuery]
    public IQueryable<TrainTripModel> GetAll(int station)
    {
        return GetAllTrips().Where(x => x.OriginRailwayStationId == station);
    }
}
```

DATAART

# Константы для роутинга

```
public static class TrainsFromControllerRoutes
{

    public const string BasePrefix =
        RailwayStationsControllerRoutes.BasePrefix +
        "/{station:int}/departures";

    public const string GetById = "{id:int}";
}
```

DATAART

# Зависимый контроллер еще раз

```
[RoutePrefix(TrainsFromControllerRoutes.BasePrefix)]
public class TrainsFromController : TrainsController
{
    [HttpGet]
    [Route]
    [EnableQuery]
    public IQueryable<TrainTripModel> GetAll(int station)
    {
        return GetAll().Where(x => x.OriginRailwayStationId == station);
    }
}
```

DATAART

# Базовый CRUD

- POST – создать новую сущность
  - POST /Stations – JSON описание сущности целиком. Действие добавляет новую сущность в коллекцию
  - Возвращает созданную сущность
- PUT – изменить сущность
  - PUT /Stations/12 – Изменить сущность с ID = 12.
  - Возвращает измененную сущность
- DELETE
  - DELETE /Stations/12 – Удалить сущность с ID = 12.

DATAART

# Еще примеры CRUD

- POST /Stations – Добавляем вокзал
- POST /Stations/1/Departures – Добавляем информацию об отправлении с вокзала 1
- DELETE /Stations/1/Departures/14 – Удаляем запись об отправлении с вокзала 1
- GET /Stations/33/Departures/10/Tickets – Список проданных билетов для отправления 10 с вокзала 33

DATAART

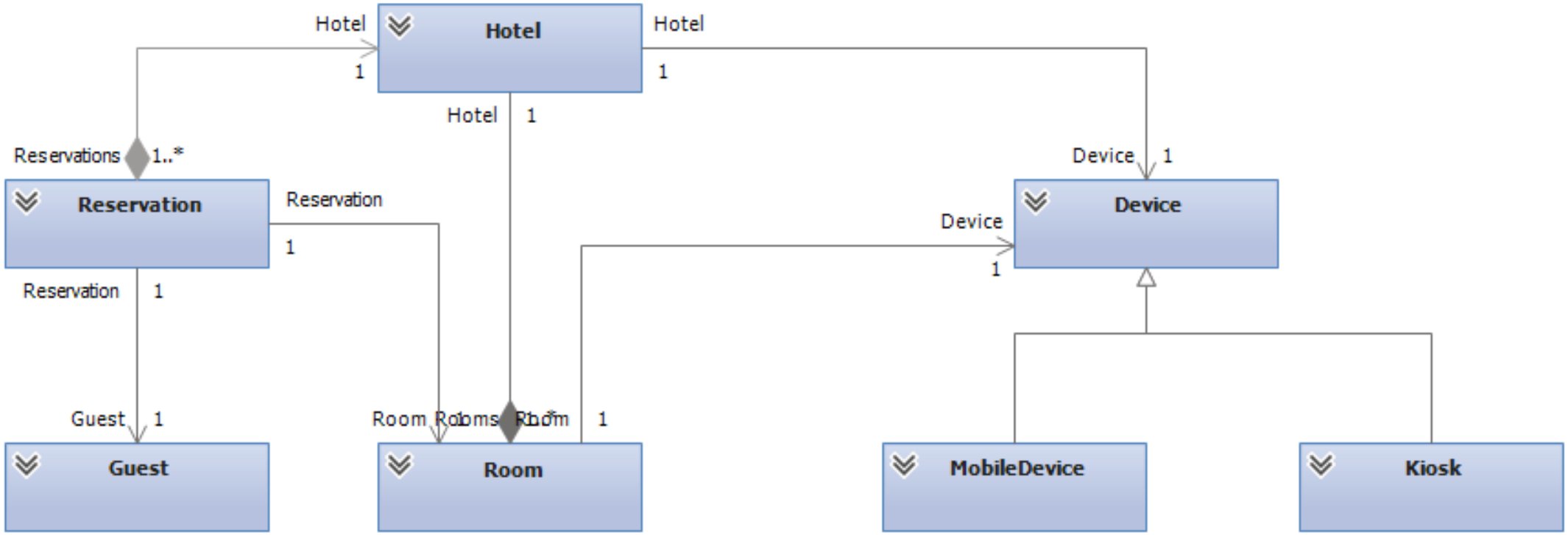
# Naming anti-patterns

- GET /Stations/?op=departure&train=11
  - действия в query string
- GET /Stations/DeleteAll
  - реальный пример из жизни :)
  - борьба с кешированием
- POST /GetUserActivity
  - пост нужен был из-за параметров запроса в body
- POST /Stations/Create
  - действие указано в составе URL - избыточно

# Проектируем API

- Как связаны сущности API с доменной моделью?
  - Никак они не связаны ☺
- Как проектировать API если это не CRUD
  - Превращаем действия в команды на изменения

# Доменная модель



# Коротенько про DDD

- Bounded Context
- Aggregates
- Entities
- Values Objects

# Bounded context (BC)

- Изолированный поддомен
- Независимы друг от друга
- Имеют независимые модели (разные)
- BC <= component

# Aggregates

- Целостная (consistent) группа сущностей
- Цель – гарантироваться целостность и согласованность всех объектов
- Aggregate root (AR) – самый «главный» объект в группе
- Все изменения только через AR
- Сущности из разных Aggregate Root не могут ссылаться друг на друга

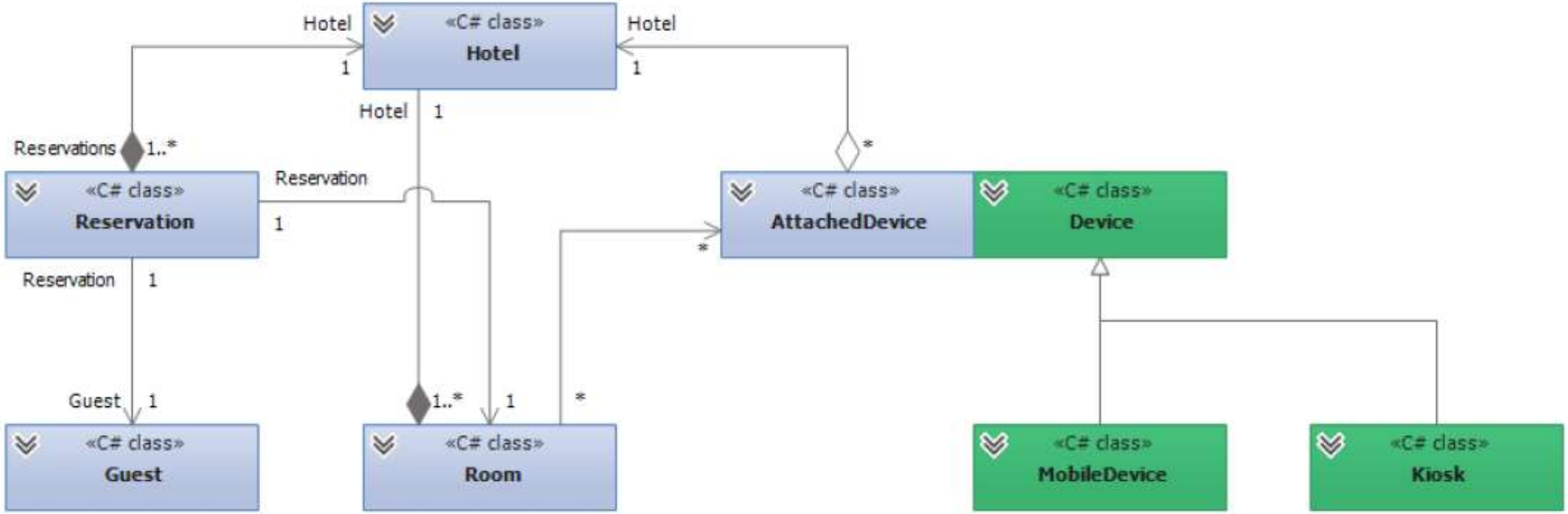
# Domain Entities

- Уникальны по ID
- Важно отличать один объекта от другого

# Values Objects

- Определен своими данными
- Уникальность не имеет значения

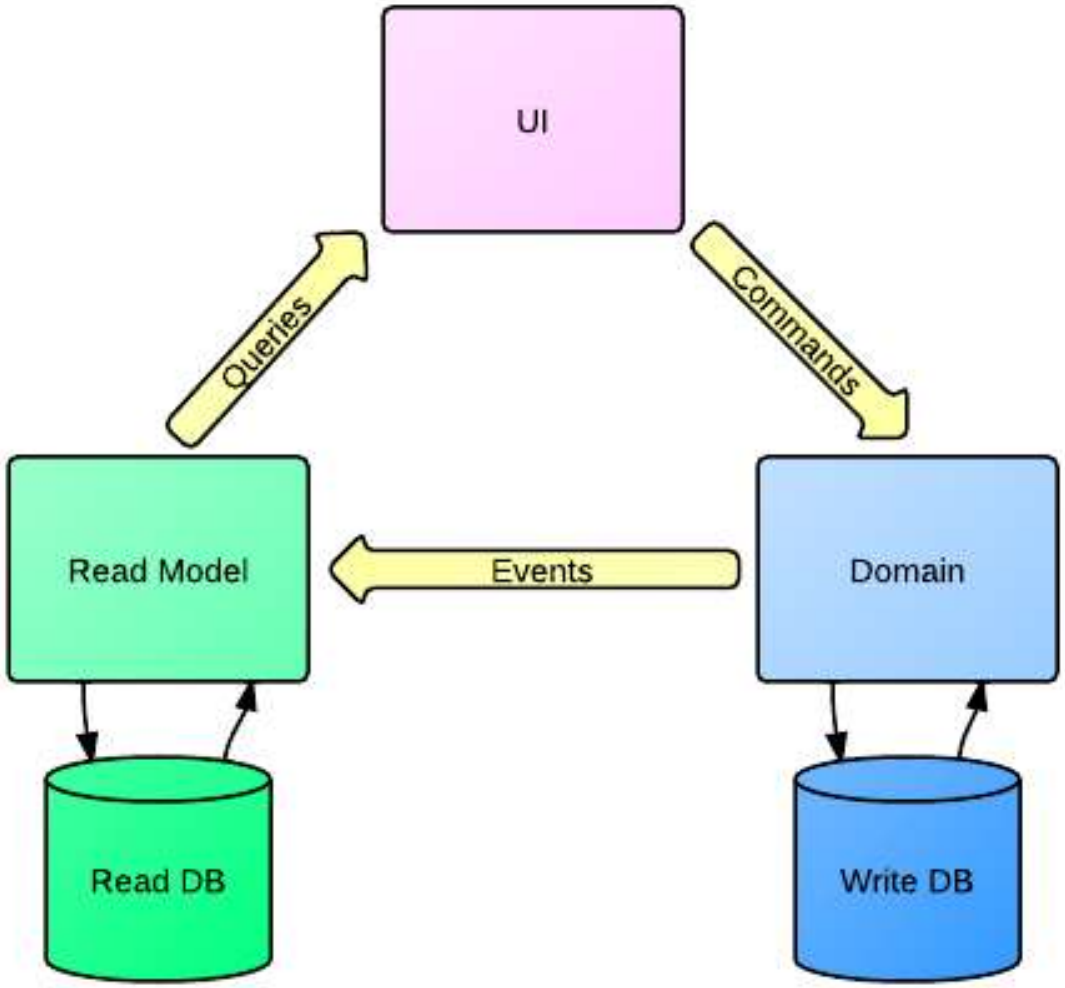
# Доменная модель



# Примеры запросов

- PUT /hotels/555/rooms/105/attachedDevices – заменить всю коллекцию привязанных устройств на новую
- POST /hotels/555/rooms/105/attachedDevices – привязать еще одно устройство
- DELETE /hotels/12 – удалить описание отеля с ID = 12
- POST /hotels/123/reservations – создать новую резервацию в отеле  id=123

# CQRS




DATAART

# REST without PUT

- Change entity XXX => New COMMAND to change entity XXX
- Можно отслеживать статус выполнения
- Можно отменять команды (DELETE)
- Легко хранить историю изменений
- Пользователь сообщает о намерениях

# Fine Grained VS Coarse Grained

- Много маленьких объектов
- Бизнес логика уходит на сторону клиента
- Нужно знать как связаны объекты

- Сложно делать локальные изменения например
  - POST /blogs/{id}/likes
- Нужно отслеживать состояние на клиенте
- Большие объекты нельзя сохранить частично

# Версионирование

- Если вы однажды опубликовали контракт, то вы обязаны его соблюдать
- Braking changes можно делать только при изменении версии

DATAART

# Подходы к версионированию

| Type | Sample | Complexity |
| --- | --- | --- |
| URL | {host}/api/v2/… | Minimum |
| Custom Header | api-version:2 | Average |
| Custom Accept Header | Accept:application/vnd.trainmodel.v2+json | Maximum |

DATAART

# Подходы к версионированию

- http://aspnet.codeplex.com/SourceControl/latest#Samples/WebApi/NamespaceControllerSelector
- http://aspnet.codeplex.com/SourceControl/latest#Samples/WebApi/RoutingConstraintsSample
- http://www.strathweb.com/2015/10/global-route-prefixes-with-attribute-routing-in-asp-net-web-api/
- https://github.com/climax-media/climax-web-http

DATAART

# Библиотека Climax.Web.Http

- [VersionedRoute("v2/values", Version = 2)]

- config.ConfigureVersioning(
  versioningHeaderName: "version", vesioningMediaTypes: null);

- config.ConfigureVersioning(
  versioningHeaderName: null,
  vesioningMediaTypes: new [] { "application/vnd.model"});

DATAART

# Документация

Swagger & swashbuckle

- http://swagger.io/
- https://github.com/domaindrivendev/Swashbuckle

DATA ART

# Swashbuckle

```
httpConfiguration
        .EnableSwagger(c => c.SingleApiVersion("v1", ”Demo API"))
        .EnableSwaggerUi();
```

DATAART

```
public static void RegisterSwagger(this HttpConfiguration config)
{
    config.EnableSwagger(c =>
    {
        c.SingleApiVersion("v1", "DotNextRZD.PublicAPI")
            .Description("DotNextRZD Public API")
            .TermsOfService("Terms and conditions")
            .Contact(cc => cc
                .Name("Vyacheslav Mikhaylov")
                .Url("http://www.dotnextrzd.com")
                .Email("vmikhaylov@dataart.com"))
             .License(lc => lc.Name("License").Url("http://tempuri.org/license"));
                c.IncludeXmlComments(GetXmlCommentFile());
                c.GroupActionsBy(GetControllerGroupingKey);
                c.OrderActionGroupsBy(new CustomActionNameComparer());
                c.CustomProvider(p => new CustomSwaggerProvider(config, p));
            })
        .EnableSwaggerUi(
            c =>
            {
                c.InjectStylesheet(Assembly.GetExecutingAssembly(),
                    "DotNextRZD.PublicApi.Swagger.Styles.SwaggerCustom.css");
            });
        }
}
```

DATAART

```
public static void RegisterSwagger(this HttpConfiguration config)
{
    config.EnableSwagger(c =>
    {
        c.SingleApiVersion("v1", "DotNextRZD.PublicAPI")
            .Description("DotNextRZD Public API")
            .TermsOfService("Terms and conditions")
            .Contact(cc => cc
                .Name("Vyacheslav Mikhaylov")
                .Url("http://www.dotnextrzd.com")
                .Email("vmikhaylov@dataart.com"))
             .License(lc => lc.Name("License").Url("http://tempuri.org/license"));
               c.IncludeXmlComments(GetXmlCommentFile());
               c.GroupActionsBy(GetControllerGroupingKey);
               c.OrderActionGroupsBy(new CustomActionNameComparer());
               c.CustomProvider(p => new CustomSwaggerProvider(config, p));
            })
        .EnableSwaggerUi(
            c =>
            {
                c.InjectStylesheet(Assembly.GetExecutingAssembly(),
                    "DotNextRZD.PublicApi.Swagger.Styles.SwaggerCustom.css");
            });
        }
}
```

DATAART

## RailwayStations : Information about all railway stations

Show/Hide | List Operations | Expand Operations

| | | |
|---|---|---|
| DELETE | /api/v1/stations | Delete all railway station |
| GET | /api/v1/stations | Get all railway stations |
| POST | /api/v1/stations | Create an railway station |
| PUT | /api/v1/stations | Update an railway station |
| DELETE | /api/v1/stations/{id} | Delete an railway station by Id |
| GET | /api/v1/stations/{id} | Get railway station by Id |
| DELETE | /api/v1/stations/{code} | Delete an railway station by railway station's code |
| GET | /api/v1/stations/{code} | Get railway station by internation railway station's code |
| GET | /api/v1/stations/{station}/arrivals | Retreive all trains arriving to the station specified |
| GET | /api/v1/stations/{station}/departures | Retreive all trains departuring from the railway station specified |
| GET | /api/v1/stations/{station}/arrivals/{id} | Retreive a particular train arriving to the station specified |
| GET | /api/v1/stations/{station}/departures/{id} | Retreive a particular train departuring from the railway station specified |

GET /api/v1/stations — Get all railway stations

## Implementation Notes

This method returns all available railway station without any trains

## Response Class (Status 200)

Model | Model Schema


```
[
  {
    "Id": 0,
    "Code": "string",
    "Name": "string",
    "CityId": 0,
    "Latitude": 0,
    "Longitude": 0,
    "Services": [
      {
        "Name": "string"
```


Response Content Type application/json ▼

Try it out!

POST /api/v1/stations — Create an railway station

## Response Class (Status 200)

Model | Model Schema

```
{
  "Id": 0,
  "Code": "string",
  "Name": "string",
  "CityId": 0,
  "Latitude": 0,
  "Longitude": 0,
  "Services": [
    {
      "Name": "string",
      "Description": "string"
```

Response Content Type application/json ▼

## Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| **model** | (required)<br>Parameter content type: application/json | **railway station data** | body | Model \| Model Schema<br>**RailwayStationModel {**<br>**Id** (integer, *optional*): Internal id of railway station in the database,<br>**Code** (string, *optional*): Public (extenal) code of railway station,<br>**Name** (string, *optional*): Translated to current language name of railway station,<br>**CityId** (integer, *optional*): Translated to current language name of railway station's city,<br>**Latitude** (number, *optional*): Latitude of railway station,<br>**Longitude** (number, *optional*): Longitude of railway station,<br>**Services** (Array[StationService], *optional*)<br>**}**<br>**StationService {**<br>**Name** (string, *optional*),<br>**Description** (string, *optional*),<br>**OpeningTime** (string, *optional*),<br>**ClosingTime** (string, *optional*)<br>**}** |

Try it out!

# Источники


- http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api
- http://www.strathweb.com/2015/10/global-route-prefixes-with-attribute-routing-in-asp-net-web-api/
- https://www.thoughtworks.com/insights/blog/rest-api-design-resource-modeling
- https://jacobian.org/writing/rest-worst-practices/
- http://piwik.org/blog/2008/01/how-to-design-an-api-best-practises-concepts-technical-aspects/
- http://www.toptal.com/api-developers/5-golden-rules-for-designing-a-great-web-api
- http://www.odata.org/
- http://owin.org/
- http://pietschsoft.com/post/2014/06/15/cqrs-command-query-responsibility-segregation-design-pattern
- https://blog.pivotal.io/pivotal-labs/labs/api-versioning


DATAART

# Thank you

To be continued…

DATAART