Code Academy

Lecturer

Rokas Slaboševičius

Repetition + further learning

Data

Today you will learn

01 HTTP requests and ways to transfer information.

02 Endpoint'ai.

03 Services and their registration.

04 Different lifecycles.

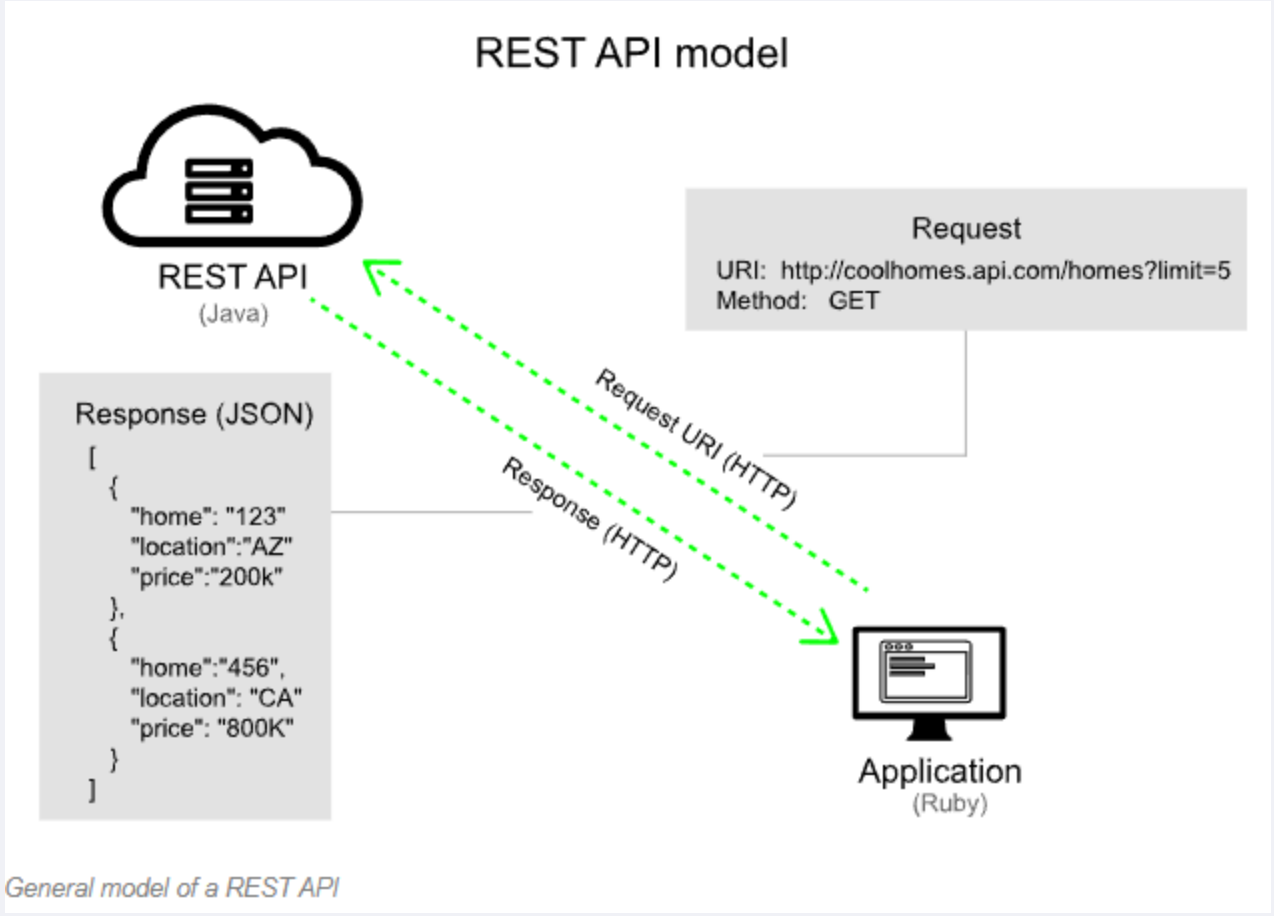
05 Singleton vs Static

Repetition and learning

- HTTP requests and ways to transfer information.
- Endpoint'ai.
- Servicing and recording.
- Different lifecycles.

HTTP requests and ways to transfer information

HTTP GET request:



General model of a REST API

HTTP requests and ways to transfer information

HTTP POST request:

1

```
POST /v1/summaries/plain-text
Host: api.jizt.it
Content-Type: application/json

{
  "source": "Very long, boring text.",
  "model": "t5-large",
  "language": "en",
  "params": {
    "relative_min_length": 0.1,
    "relative_max_length": 0.4,
    "do-sample": true,
    "num-beams": 5,
    ...
  }
}
```

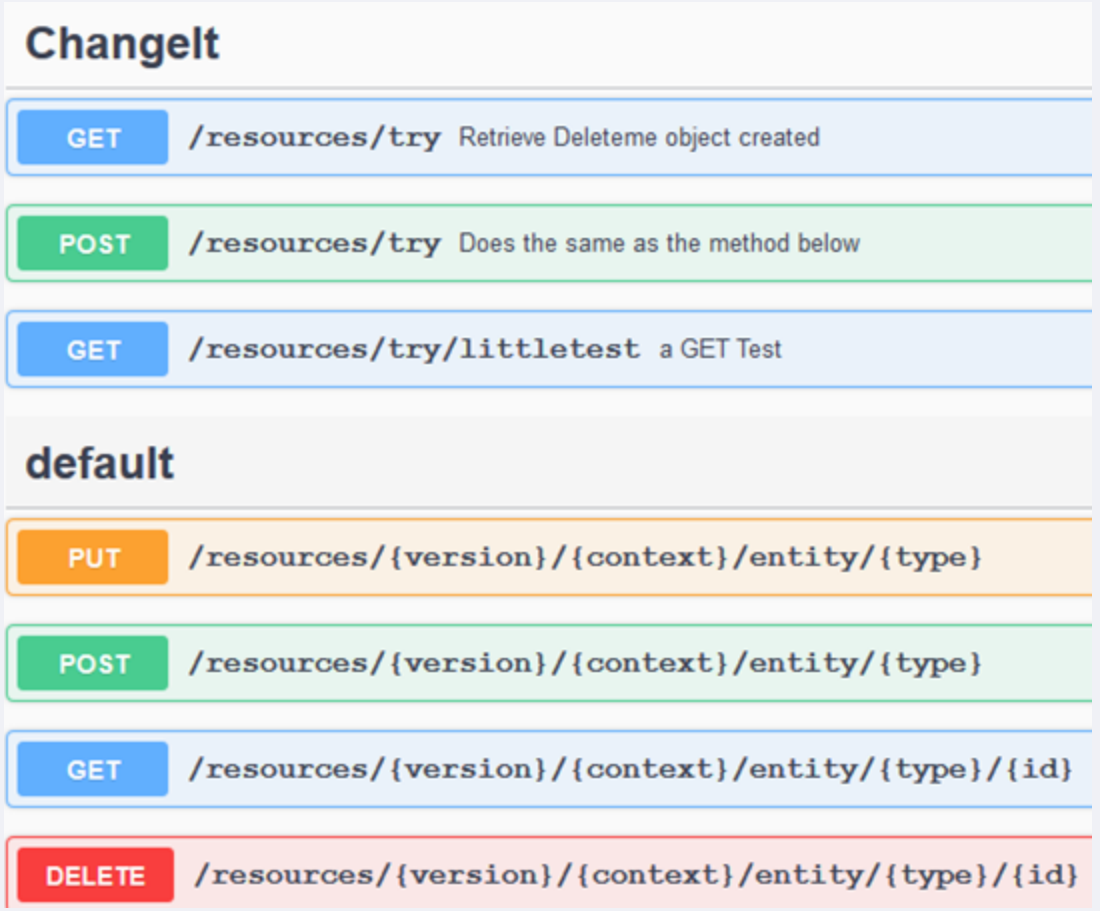
POST Request

```
{
  "summary_id": "b40d19c2af9c5096da9070",
  "started_at": "2020-12-30 17:51:21.49",
  "ended_at": "null",
  "state": "summarizing",
  "output": "null",
  ...
}
```

Response

Endpoint

Different endpoints:



Types of data transmission in the request

The main methods:

[FromRoute] get values from route, e.g. www.website.com/cars

[FromQuery] get values from URI query when we want to specify FromRoute search, e.g. /cars?color=blue

[FromBody] get values from the HTTP request body.

Additional methods:

[FromForm] receives the values of the data coming from a downloaded form whose content-type is 'application/x-www-url-formencoded', while e.g. FromBody reads in the default way, which is usually application/json

[FromHeader] gets values from the HTTP header.

[FromService] will get the injected value from the DI(Dependency injection) resolver (we're not worried about this yet)

For self-reading: https://www.dotnetcurry.com/aspnet/1390/aspnet-core-web-api-attributes

Servicing and recording.

There are several parts to service design:

1. Creating the interface
2. The creation of a class that implements that interface.
3. Registering a service in the IServiceCollection (usually in the Startup.cs file, in the ConfigureServices() method) by selecting the appropriate method for the situation (Scoped, Transient, Singleton)

```
public class TaskRepository : ITaskRepository
{
    private static readonly List<TodoTask> _tasks = new();

    public IEnumerable<TodoTask> SaveTask(TodoTask task)
    {
        _tasks.Add(task);
        return _tasks;
    }
}
```

```
services.AddSingleton<ITaskRepository, TaskRepository>();
```

```
private readonly ITaskRepository _taskRepository;

public TaskController(ITaskRepository taskRepository)
{
    _taskRepository = taskRepository;
}
```

When using inject, we don't forget to inject it through the constructor:

Servicing and recording.

AddTransient - The service is created **ONLY** when it is called.

AddScoped - A service is created for each HTTP request.

AddSingleton - The service is created once and lasts for the lifetime of the application.

Singleton vs Static.

Until now, we have been learning that in order to have a single instance of a resource throughout the whole program, we have to declare it as static.

Now there is a concept called Singleton, which is used for registering services when we want to keep a service alive for the lifetime of the application.

What is the difference between them?

First of all, you need to understand that **Singleton** is not a keyword like **static.** Singleton is a pattern.

Some of the differences in implementation:

1. A singleton can implement interfaces, inherit classes and be inherited itself. A static class cannot inherit.
2. The singleton class can be initialized lazily or asynchronously and loaded automatically when the resource is needed. A static class is loaded and initialized immediately.
3. A singleton class can have a constructor while a static class cannot.

More differences:

http://net-informations.com/faq/netfaq/singlestatic.htm

Repetition + learning

Task 1

- Let's build a small API again, using what we have learned.
- API safety cars.
- Exposable endpoints:
    - [GET]GetAllCars - return all cars
    - [GET]GetCarsByColor - returns cars by colour. Use the [FromQuery] attribute.
    - [POST]AddNewCar - Adds a new car (use CarDto, no Id).
    - [PUT]UpdateCar - Takes the car ID from the query parameter and the information from the [FromBody] parameter(Use CarDto, without Id).
    - [Delete] - Delete car by Id
- The class storing the information must be a Singleton (no static fields inside it either).