SERVICE DESIGN AND ENGINEERING

REST – DATA SERVICES

Riccardo RATTA - Emiliano TOLOTTI - Andrea BALASSO

REST – BRIEF SUMMARY

- By using the term REST we indicate an architectural style
- *RESTful* web services conform to the REST architectural style
- Today we focus on services that deal with data

RESOURCE REPRESENTATION

RESOURCE IDENTIFICATION

LINKS AND CONNECTEDNESS

STATELESSNESS

UNIFORM INTERFACE

PROTEZIONE CIVILE
Presidenza del Consiglio dei Ministri
Dipartimento della Protezione Civile

EMERGENZA CORONAVIRUS

Italiano - English

Dati COVID-19 Italia

License Creative Commons Attribution 4.0 International | last commit today

Modifiche ai dataset completate - Avviso
I dati legacy, con la strutturazione delle cartelle dei vari dataset, sono disponi
saranno alimentate fino a venerdì 31/07

The legacy data, with the structure of the dataset folders, are available in th
until Friday 31/07

Sito del Dipartimento della Protezione Civile - Emergenza Coronavirus: la risp

Il 31 gennaio 2020, il Consiglio dei Ministri dichiara lo stato di emergenza, p
del rischio sanitario connesso all'infezione da Coronavirus. Al Capo del Dip
Borrelli, è affidato il coordinamento degli interventi necessari a fronteggiar
Le principali azioni coordinate dal Capo del Dipartimento sono volte al soc
eventualmente interessata dal contagio, al potenziamento dei controlli ne
continuità con le misure urgenti già adottate dal Ministero della salute, al
nei Paesi a rischio e al rimpatrio dei cittadini stranieri nei Paesi di origine

Per informare i cittadini e mettere a disposizione i dati raccolti, utili ai s
Dipartimento della Protezione Civile ha elaborato un cruscotto geogra
http://arcg.is/C1unv (versione desktop) e http://arcg.is/081a51 (versio
CC-BY-4.0, le seguenti informazioni aggiornate quotidianamente alle
del Capo Dipartimento):

CONTEXT

Data Sources

CONTEXT – WHAT WE WANT THE API TO DO

- The first step was defining which data we want the API to expose
- We found a good collection of data regarding the COVID-19 in Italy
  - github.com/pcm-dpc/COVID-19

CONTEXT – WHAT WE WANT THE API TO DO

- …however, such data is stored in .csv files
- The task was then to collect the content of these files
- And share it through an API

dpc-covid19-ita-andamento-nazionale-20201025.csv	2020-10-25
dpc-covid19-ita-andamento-nazionale-20201026.csv	2020-10-26
dpc-covid19-ita-andamento-nazionale-20201027.csv	2020-10-27
dpc-covid19-ita-andamento-nazionale-20201028.csv	2020-10-28
dpc-covid19-ita-andamento-nazionale-20201029.csv	2020-10-29
dpc-covid19-ita-andamento-nazionale-20201030.csv	2020-10-30
dpc-covid19-ita-andamento-nazionale-20201031.csv	2020-10-31
dpc-covid19-ita-andamento-nazionale-20201101.csv	2020-11-01
dpc-covid19-ita-andamento-nazionale-20201102.csv	2020-11-02
dpc-covid19-ita-andamento-nazionale-20201103.csv	2020-11-03
dpc-covid19-ita-andamento-nazionale-20201104.csv	2020-11-05
dpc-covid19-ita-andamento-nazionale-20201105.csv	2020-11-05
dpc-covid19-ita-andamento-nazionale-20201106.csv	2020-11-06
dpc-covid19-ita-andamento-nazionale-20201107.csv	2020-11-07
dpc-covid19-ita-andamento-nazionale-latest.csv	2020-11-07
dpc-covid19-ita-andamento-nazionale.csv	2020-11-07

TECHNOLOGIES

- TypeScript
- Node.js
- Express

NODE.JS

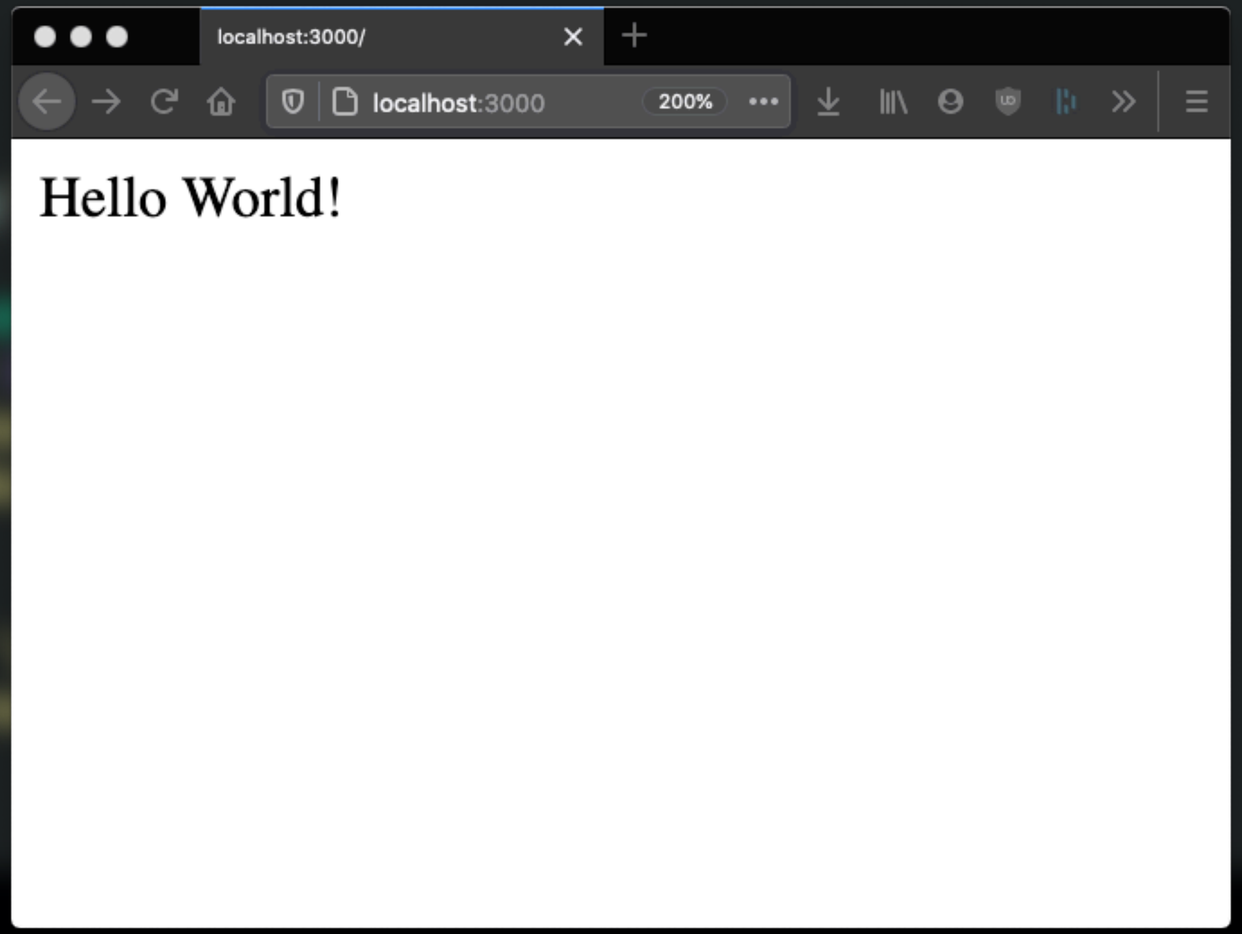
- JavaScript runtime environment
- Executes JavaScript outside a web browser
- Allows us to create REST APIs

  - nodejs.org

NODE.JS

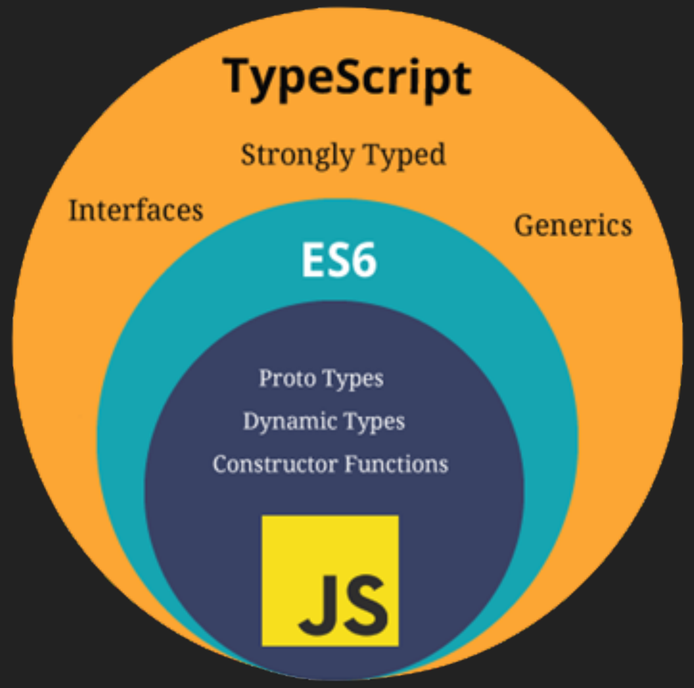
```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World');
});

server.listen(3000, '127.0.0.1', () => {
    console.log(`Server running ..`);
});
```

```
$ node index.js
```


localhost:3000/
localhost:3000
200%
Hello World!

TYPESCRIPT

- Superset of JavaScript
- For what we need to know, TypeScript is just JavaScript
- Since TypeScript compiles to JavaScript, browser runs JavaScript



TYPESCRIPT EXAMPLE

```
const user = {
    firstName: "Angela",
    lastName: "Davis",
    role: "Professor"
}

console.log(user.name)
```

TYPESCRIPT EXAMPLE

```
const user = {
    firstName: "Angela",
    lastName: "Davis",
    role: "Professor"
}

console.log(user.name)
```

```
Property 'name' does not exist on type '{ firstName: string;
lastName: string; role: string; }'.
```

NODE.JS VANILLA IS A BIT VERBOSE?

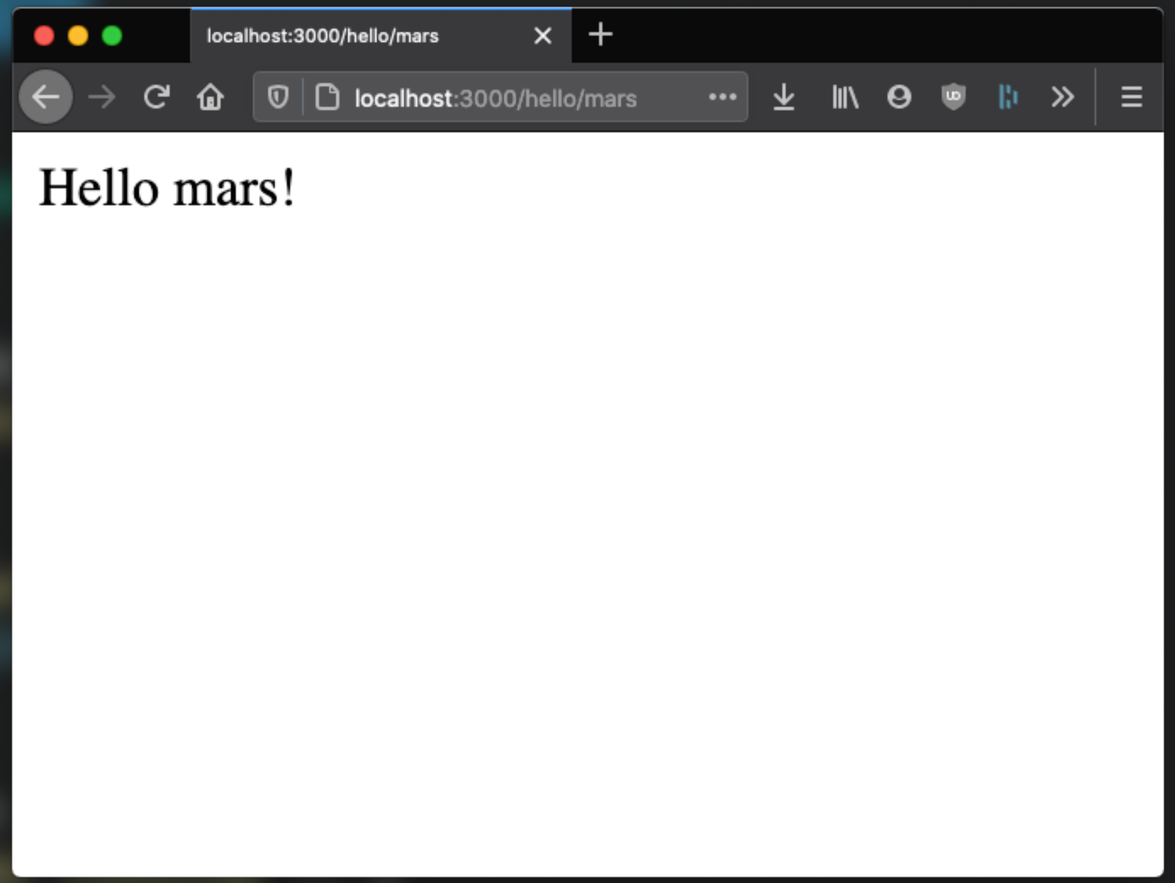
```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World');
});

server.listen(3000, '127.0.0.1', () => {
    console.log(`Server running ..`);
});
```

EXPRESS.JS COUNTEREXAMPLE

```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.listen(3000, () => {
    console.log(`Server running ..`)
})
```

EXPRESS.JS ADD ROUTE

```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.get('/hello/:planet', (req, res) => {
    res.send(`Hello ${req.params.planet}!`)
})

app.listen(3000, () => {
    console.log(`Server running ..`)
})
```


localhost:3000/hello/mars
localhost:3000/hello/mars
Hello mars!

HANDS ON

Run the API

HANDS ON – RUN THE PROJECT

- Open the project's folder in the CLI
- Install the dependencies (if necessary)
  - npm install
- Run the server on localhost
  - npm run start

HANDS ON – RUN THE PROJECT

- The port is notified in the command line
- Open the browser
- Connect to localhost:port/regions to see that server is running

```python
@main.command()
@click.option('-a', '--all', is_flag=True)
@click.option('-l', '--latest', is_flag=True)
@click.option('-d', '--date')
def download(all, latest, date):
    # download command
    "Download data (by date, all, or latest)"

    if (all and latest) or (all and date) or (latest and date):
        click.echo("Multiple options not allowed")
        return
    elif all:
        click.echo("Downloading all days data...")
        query = ''
    elif latest:
        click.echo("Downloading latest day data...")
        query = '-latest'
    elif date:
        click.echo("Download data corrisponding to day " + date +
                   " in yyyymmdd format (Ex. 20200304 download for 
        query = '-' + date
    else:
        click.echo(click.get_current_context().get_help())
        return

    url = url_head + query + url_tail  # compose final url
    click.echo("Fetching: " + url)
    response = requests.get(url)  # http get request to reposito
    file_content = response.content
    file_content = header.encode(
        'UTF-8') + file_content.split(b'\n', 1)[1]  # replace 
    open(file, 'wb').write(file_content)  # write content to c


@main.command()
def send():
    # send command
    "Send data to db (put)"

    with open(file) as csvfile:  # open csv file
        reader = csv.DictReader(csvfile)
        field = reader.fieldnames  # get header
```

HANDS ON

Collect the data

RETRIEVE DATA FROM SOURCES

- The next step is to retrieve the data from the .csv files
- And properly format them to invoke the API PUT method
- The helper.py file allows us to simply do that

FIRST STEP

- We first import the useful packages
- We specify important variables such as the name of the file we'll create locally, the url from which we get data, the api port on which to send our request

```
import click
import requests
import csv
import json
from datetime import datetime

file = 'data.csv'
url_head = 'https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-regioni/dpc-covid19-ita-regioni'
url_tail = '.csv'
api_url = 'http://localhost:8080'
header_start = 4
header_end = 19
header = ("date,state,region_id,region_name,lat,lon,hospitalized_with_symptoms,intensive_care,total_hospitalized,"
          "home_isolation,total_positive,total_positive_variation,new_positives,resigned_cured,deceased,"
          "cases_from_suspected_diagnostic,cases_from_screening,total_cases,tampons,cases_tested,notes\n")
```

DOWNLOAD CONTENT FROM SOURCE

- We create a download function to download files from the data source
- We allow the user to specify different parameters to choose the file he wants to download

```
@main.command()
@click.option('-a', '--all', is_flag=True)
@click.option('-l', '--latest', is_flag=True)
@click.option('-d', '--date')
def download(all, latest, date):
    # download command
    "Download data (by date, all, or latest)"

    if (all and latest) or (all and date) or (latest and date):
        click.echo("Multiple options not allowed")
        return
    elif all:
        click.echo("Downloading all days data...")
        query = ''
    elif latest:
        click.echo("Downloading latest day data...")
        query = '-latest'
    elif date:
        click.echo("Download data corrisponding to day " + date +
                   " in yyyymmdd format (Ex. 20200304 download for 04 March 2020)...")
        query = '-' + date
    else:
        click.echo(click.get_current_context().get_help())
        return

    url = url_head + query + url_tail  # compose final url
    click.echo("Fetching: " + url)
    response = requests.get(url)  # http get request to repository file
    file_content = response.content
    file_content = header.encode(
        'UTF-8') + file_content.split(b'\n', 1)[1]  # replace original header
    open(file, 'wb').write(file_content)  # write content to csv file
```

SEND CONTENT TO API ENDPOINT

- The send function simply allows us to send a request to our API, automatically specifying the endpoint, the file with the content to send, automatically formatting the content of the file to our request body

```
@main.command()
def send():
    # send command
    "Send data to db (put)"

    with open(file) as csvfile:  # open csv file
        reader = csv.DictReader(csvfile)
        field = reader.fieldnames  # get header
        status_codes = dict()
        click.echo("Sending data...")

        for row in reader:  # for each row
            payload = {field[i]: my_num(row[field[i]])
                       for i in range(header_start, header_end)}  # create payload object
            date = datetime.fromisoformat(row['date'])
            url = api_url + '/region/' + \
                row['region_id'] + "/cases/" + str(date.year) + \
                "/" + str(date.month) + "/" + \
                str(date.day)  # compose final url
            response = requests.put(url, data=json.dumps(payload), headers={
                "content-type": "application/json"})  # http put request to api
            key = 'n of ' + str(response.status_code)
            status_codes[key] = status_codes[key] + \
                1 if key in status_codes else 1  # add response status code to dictionary

        click.echo("Status codes results: ")
        click.echo(status_codes)
```

EXERCISE 1 - HANDS ON

- Now that you know what the helper.py script does, it is time to use it!
  - Step 0 Make sure the API is running!!!
  - Step 1 Download the latest file and send it to the api
  - Step 2 Return the number of swabs for that day and write it in the chat

- Help You can perform in Postman a GET to endpoint /region/21/cases/2020
  - Swabs are the tests used to check if covid-positive (*translates to "tamponi"*)

```
import express from 'express'
import cors from 'cors'
import compression from 'compression'

import { Region } from './controllers'
import { d } from './helpers'
import { DateTime } from 'luxon'

const port = 8080

const app = express()

app.use(compression())
app.use(cors())
app.use(express.json())

app.get('/regions', (_, resp) => {
    resp.send(d.get('regions'))
})

app.get('/region/:regionId', (req, resp) => {
    const region = Region.fromId(parseInt(req.params.regionId))
    if (region) {
        return resp.send(region)
    } else {
        resp.status(404).send('Region not found')
    }
})

app.get('/region/:regionId/cases/:year', (req, resp) => {
    const region = Region.fromId(parseInt(req.params.regionId)
    if (region) {
        const year = region.year(parseInt(req.params.year))
        if (year) {
            return resp.send(year)
        } else {
            resp.status(404).send('Year not found')
        }
    } else {
        resp.status(404).send('Region not found')
    }
```

HANDS ON

API code

FIRST STEP

- First we import the packages that we need
- We also import other modules we created to manage specific logics
- We then specify the port where the app will run
- And put the Express application inside the app variable

```
import express from 'express'
import cors from 'cors'
import compression from 'compression'

import { Region } from './controllers'
import { d } from './helpers'
import { DateTime } from 'luxon'

const port = 8080

const app = express()

app.use(compression())
app.use(cors())
app.use(express.json())
```

GET METHOD

- We use app.get to define a get method in our API
- We specify the path relative to that method
  - We indicate a parameter in the URI with *:IDname*
- Then, parsing the request is fairly easy

```
app.get('/region/:regionId', (req, resp) => {
    const region = Region.fromId(parseInt(req.params.regionId))
    if (region) {
        return resp.send(region)
    } else {
        resp.status(404).send('Region not found')
    }
})
```

- We then specify the behavior of the method, what to return, and the status to send

PUT METHOD

- As we've seen, the parts of the URI with : in front are params
- We load in the DB the cases of the region :regionId, for the date :year, :month, :day
- We specify what the API should do with the request body relative to the URI params

```
app.put('/region/:regionId/cases/:year/:month/:day', async (req, resp) => {
    const region = Region.fromId(parseInt(req.params.regionId))
    if (region) {
        const dateTime = DateTime.local(
            parseInt(req.params.year), parseInt(req.params.month), parseInt(req.params.day))
        // check if the provided date makes sense
        if (dateTime.isValid) {
            return resp.send(await region.set(
                dateTime.year, dateTime.month, dateTime.day, req.body))
        } else {
            resp.status(400).send(`Provided date not valid: ${dateTime.invalidReason}`)
        }
    } else {
        resp.status(404).send('Region not found')
    }
})
```

- Note that we use a PUT method because if we try the same request multiple times, it is equivalent to a single request

ENDPOINTS

- The endpoints of our API are the following:
  - GET /regions
  - GET /region/:regionId
  - GET /region/:regionId/cases/:year
  - GET /region/:regionId/cases/:year/:month
    - ~~GET /region/:regionId/cases/:year/:month/:day~~
  - PUT /region/:regionId/cases/:year/:month/:day
  - PATCH /region/:regionId/cases/:year/:month/:day

EXERCISE 2 – CREATE A NEW ENDPOINT

- Our api does not accept* a specific day for a GET request :(
  Until now only years and months endpoints are accepted.
  - Implement /region/:regionId/cases/:year/:month/:day GET method
- Tip Write the endpoint that accepts also the day based on the other 2
- *Actually it did, but we wanted you to do it on your own

HANDS ON

Exercises

EXERCISE 3 – TEST YOUR SKILLS

- We have the data relative to latest day loaded in our db.
  - Write an endpoint that computes the percentage of the total_hospitalized over total_positive for a specific region and day
  - Write the number you get in the chat

- Example of execution:
  GET /region/:regionId/percentage/:year/:month/:day
  Response {
  "region": regionId, "year": year, "month": month,
  "day": day, "percentage": 0.15
  }

HOMEWORK - ASSIGNMENT

- We want to know how the number of cases in a region changed from specific day "start" and day “end"
- Write an api endpoint that performs the delta of the total_cases
- The endpoint should be of the form:
  GET /region/:regionId/delta?start=yyyy-mm-dd&end=yyyy-mm-dd
- Tip delta = end.total_cases - start.total_cases

HOMEWORK - ASSIGNMENT

- The response we expect is of the form:

```
{
    "region": regionId,
    "start year": year,
    "start month": month,
    "start day": day,
    "end year": year,
    "end month": month,
    "end day": day,
    "delta": number
}
```