

Opdracht Python, PowerShell en een REST API

D. Leeuw

19 november 2025

0.5.1

© 2025 Dennis Leeuw



Dit werk is uitgegeven onder de Creative Commons BY-NC-SA Licentie en laat anderen toe het werk te kopiëren, distribueren, vertonen, op te voeren, en om afgeleid materiaal te maken, zolang de auteurs en uitgever worden vermeld als maker van het werk, het werk niet commercieel gebruikt wordt en afgeleide werken onder identieke voorwaarden worden verspreid.

1 Over dit Document

1.1 Leerdoelen

Uit het Kwalificatie Dossier, P2-K1 Ontwikkelt digitale informatievoorzieningen:

- Heeft kennis van één of meer programmeer- of scripttalen, zoals C#, Java, Javascript, C++, Python, PHP, Bash of PowerShell
- Heeft kennis van één of meerdere dataformaten, zoals JSON, YAML, XML
- Kan data ophalen uit of verzenden naar verschillende platformen, diensten of apparaten door gebruik van REST APIs

1.2 Voorkennis

Voordat de student aan deze opdracht kan beginnen dient deze kennis te hebben van:

- Python
- PowerShell
- de werking van een webserver
- wat REST is
- JSON en dit te kunnen lezen

2 Python: FastAPI

FastAPI is een Python implementatie van een REST server. Het geeft je de mogelijkheid om snel en simpel je eigen REST server te bouwen.

```
pip install "fastapi[standard]"
```

Voor meer informatie over FastAPI kan je terecht op <https://fastapi.tiangolo.com/>.

`pip` plaatst FastAPI in een eigen directory. Deze directory is standaard niet toegevoegd aan je PATH en dus kan je `fastapi` alleen opstarten door het hele pad op te geven. Dat is niet handig dus gaan we het pad toevoegen aan onze PATH variabele in ons PowerShell profile. Het pad naar je profile staat in de variabele `profile` dus die kunnen we gebruiken:

```
notepad.exe $profile
```

Voeg een regel toe met het pad dat je hebt gekregen van `pip` (jouw pad zal anders zijn dan in het gegeven voorbeeld)

```
$env:PATH += "C:\Users\dleeu\AppData\...\local-packages\Python313\Scripts"
```

Elke keer als je hierna PowerShell opstart zal dit pad worden toegevoegd PATH zodat de `fastapi`-server gevonden wordt. Je kan dit testen door een tweede console voor PowerShell te openen en in te typen:

```
$env:PATH
```

Een webserver heeft natuurlijk een start-pagina nodig en die gaan we maken in Python. Maak een nieuw tekstbestand met de naam `main.py` en zet daarin:

```
from typing import Union

from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```

Nu kunnen we de server opstarten:

```
fastapi dev main.py
```

Dit zou een overzicht zoals dit op moeten leveren:

```

PS C:\Users\dleeu> fastapi.exe dev .\main.py

FastAPI Starting development server 🚀

Searching for package file structure from directories with __init__.py files
Importing from C:\Users\dleeu

module 📄 main.py

code Importing the FastAPI app object from the module with the following code:

from main import app

app Using import string: main:app

server Server started at http://127.0.0.1:8000
server Documentation at http://127.0.0.1:8000/docs

tip Running in development mode, for production use: fastapi run

Logs:

INFO Will watch for changes in these directories: ['C:\\Users\\dleeu']
INFO Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO Started reloader process [11252] using WatchFiles
INFO Started server process [5844]
INFO Waiting for application startup.
INFO Application startup complete.

```

3 REST-API test

De `main.py` code luistert op de web-interface naar “/” en naar “/items/itemID”. Dit is dus zoals we de server kunnen aanroepen.

De meest simpele test die we kunnen doen is onze web-browser openen en in de adresbalk <http://127.0.0.1:8000/> invoeren. We benaderen dan de FastAPI server met de vraag om het / pad op te voeren. Als alles goed gegaan is dan zal de server antwoorden met:

```
{"Hello": "World"}
```

Windows 10 en 11 hebben standaard `curl` geïnstalleerd staan. `cURL` is een commandline tool om tegen web-servers te praten zonder alle overhead van een web-browser. Het is de tool voor de systeembeheerder, ontwikkelaar om REST-API's te testen. Meer informatie over `curl` kan je vinden op <https://curl.se/>. Wij beginnen met een simpele test in PowerShell:

```
curl.exe -X GET http://127.0.0.1:8000/
```

Vergeet de `.exe` niet anders praat je tegen de cmdlet `Invoke-WebRequest` inplaats van tegen de echte `curl` en dat levert hele andere resultaten op als we echt de opties van `curl` gaan gebruiken.

De optie `-X` geeft ons de mogelijkheid om aan `curl` te vertellen met welke method er tegen de webserver gepraat moet worden, in dit geval gebruiken we het `GET` method.

Als we dit op een tweede PowerShell console uitvoeren dan krijgen we hetzelfde antwoord als wat we in onze web-browser al gekregen hebben.

We kunnen met `curl` ook de tweede mogelijkheid van onze `main.py` opvragen:

```
curl.exe -X GET http://127.0.0.1:8000/items/5?q=somequery
```

Dit zou het volgende antwoord moeten opleveren:

```
{"item_id": 5, "q": "somequery"}
```

We zien dat we een JSON list terug krijgen met het `item_id` en de query die we hebben opgegeven.

Met `CTRL` + `C` kunnen we de REST-API server stoppen.

4 Opdracht

We gaan de `main.py` aanpassen zodat we de volgende functionaliteit krijgen:

1. We willen opdrachten per operating system aan de server kunnen geven. Voorlopig ondersteunen we alleen Windows:
 - <http://127.0.0.1/windows?<opdracht>=<functie>>
2. We gaan voor deze opdrachten werken met de uitkomsten van het PowerShell commando dat we geschreven hebben bij de PowerShell argumenten opdracht.
3. Via de REST-API vragen we de gegevens op.
 - <http://127.0.0.1/windows?version=CurrentBuild>
 - <http://127.0.0.1/windows?version=DisplayVersion>
 - <http://127.0.0.1/windows?version=ProductName>
4. Herschrijf de Python `main.py` met functies die de gevraagde informatie via REST in JSON produceert. Gebruik de subprocess module om het PowerShell script aan te roepen.