



# Flask

## Routing - Form verwerking

Labo 04

# Content

<b>LEERDOELEN.....</b>	<b>4</b>
<b>1      VOORAF .....</b>	<b>5</b>
<b>2      INSTELLEN VAN EEN VIRTUAL ENVIRONMENT (VENV) – WINDOWS .....</b>	<b>6</b>
2.1    Aanmaken venv voor de lessen FSWD - Windows.....	6
2.1.1   Mappenstructuur voor lessen FSWD.....	6
2.1.2   Venv aanmaken - Windows .....	6
2.1.3   Packages toevoegen aan venv - Windows.....	7
<b>3      INSTELLEN VAN EEN VIRTUAL ENVIRONMENT (VENV) - MACOS .....</b>	<b>9</b>
3.1    Aanmaken venv voor de lessen FSWD - macOS .....	9
3.1.1   Mappenstructuur voor lessen FSWD.....	9
3.1.2   Venv aanmaken - macOS .....	9
3.1.3   Packages toevoegen aan venv - macOS.....	10
<b>4      OEFENING 1: WINKEL .....</b>	<b>12</b>
4.1    Front-end  .....	12
4.2    Back-end  .....	13
4.2.1   Voorbereiding.....	13
4.2.2   Opdracht.....	13
4.2.3   Testen .....	14
4.2.4   'Hacken' met Postman.....	15
<b>5      OEFENING 2: TRACK AND TRACE .....</b>	<b>18</b>
5.1    Back-end  .....	18
5.1.1   Voorbereiding.....	18
5.1.2   Opdracht.....	19
5.2    Front-end  .....	19
5.2.1   Voorbereiding.....	19
5.2.2   Opdracht.....	20
5.3    Thuisopdracht.....	22
5.3.1   Back-end  .....	22
5.3.2   Front-end  .....	22
5.3.3   Resultaat.....	22
5.3.4   Uitbreiding.....	23
5.3.5   Extra uitdaging.....	23
<b>6      OEFENING 3: PUNTELIJST .....</b>	<b>24</b>
6.1    Back-end  .....	24
6.1.1   Opdracht.....	24
6.2    Front-end  .....	25

6.2.1	Opdracht.....	25
-------	---------------	----

## Leerdoelen

---

- Een virtual environment instellen op je **back-end**.
- In de **back-end** een route instellen die *GET* en/of *POST method* aanvaardt
- Een route programmeren die data van een formulier verwerkt in de **back-end** en een correct response terugstuurt naar de **front-end**.
- Een route programmeren die data uit een dictionary opvraagt in de **back-end**. Via een call de data ophalen en weergegeven in de **front-end**.
- De response statuscode instellen van een response in de **back-end**.
- De header van een response aanpassen in de **back-end** zodat je geen CORS errors krijgt.
- In de **front-end** het click-event van een knop opvangen om nadien in interactie te gaan met de **back-end**.
- In de **front-end** het input-event van een form element opvangen om nadien in interactie te gaan met de **back-end**.
- Het uitzicht van de **front-end** veranderen door de classlist van een element aan te passen via Javascript.
- Het uitzicht van de **front-end** veranderen door de innerHTML en value property van een element aan te passen via Javascript.

# 1 Vooraf

We gebruiken momenteel de extension *Live Server* in VS Code om de **front-end** code te testen. Deze biedt de mogelijkheid om onze (statische) webpagina's te openen via een lokale webserver.  
De pagina's worden geopend via het `http://` protocol op de localhost.

We zien dat

- De **Live Server** extension start op de localhost een webserver op poort **:5500**.
- De **Werkzeug** server van **Flask** start, wanneer je `app.py` runt, een server op onze localhost poort **:5000**

Dit zijn met andere woorden twee totaal verschillende webservers die op dezelfde machine draaien.

- We zullen voor de **Front-end** gebruik maken van JS en HTML die we opstarten via de *Live Server* in VS Code.
- We zullen voor de **Back-end** gebruik maken van Flask en Python die we opstarten via de *Werkzeug* server in VS Code door het python project te runnen in de terminal.

## 2 Instellen van een virtual environment (venv) – Windows

Voor de werkwijze op een laptop met macOS ga naar hoofdstuk 3 Instellen van een virtual environment (venv) - macOS

Voor de back-end code gaan we verschillende packages toevoegen aan de python interpreter. Een package toevoegen doe je in het command prompt via het commando `pip`.

Het nadeel van al de packages toe te voegen aan de default python interpreter (C:\Program Files\Python 3.x) is dat je de standaard interpreter “vervult” met allerlei packages die je wil testen.

Een betere manier is:

- Om voor elk nieuw project een kopie (*een venv: virtual environment*) te nemen van de default python interpreter.
- De venv in dezelfde folder als het VS Code project te bewaren.
- Het VS Studio project te koppelen aan de venv.
- Alle packages die je nodig nodig (denkt) te hebben toe te voegen aan de venv.

Het voordeel van op deze manier te werken is:

- De default interpreter wordt niet vervuld met onnodige packages.
- Als je het project kopieert naar een andere computer wordt de venv (met alle packages) mee gekopieerd.
- Heb je verkeerde packages toegevoegd, of krijg je conflicten, dan kan je eenvoudig de venv verwijderen maar blijft de default interpreter “clean”.

### 2.1 Aanmaken venv voor de lessen FSWD - Windows

Om niet per week een nieuwe venv aan te moeten maken, plaatsen we alle projecten in 1 “root” map. In deze “root” map plaatsen we een venv.

Vanaf nu openen we dus altijd deze “root” map.

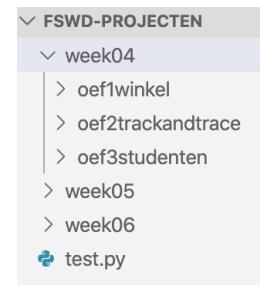
#### 2.1.1 Mappenstructuur voor lessen FSWD

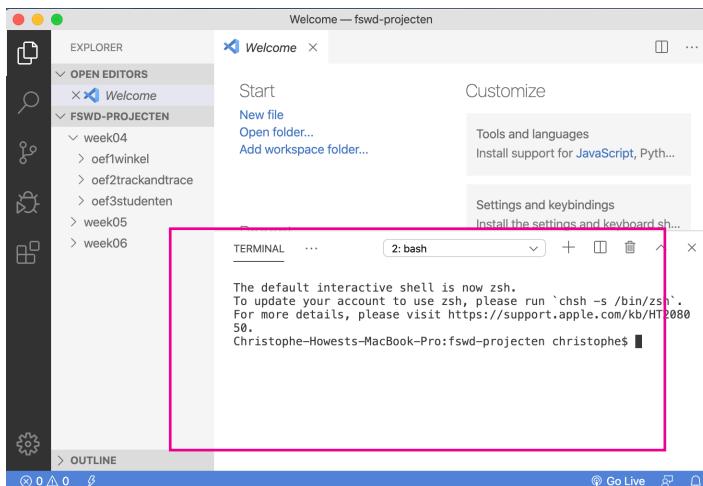
- Maak een folder FSWD-projecten aan.
- Maak een folder week04, week05,... aan in de folder FSWD-projecten.
- Plaats de labo oefeningen in week04.
- Open de folder FSWD-projecten in VS Code.



#### 2.1.2 Venv aanmaken - Windows

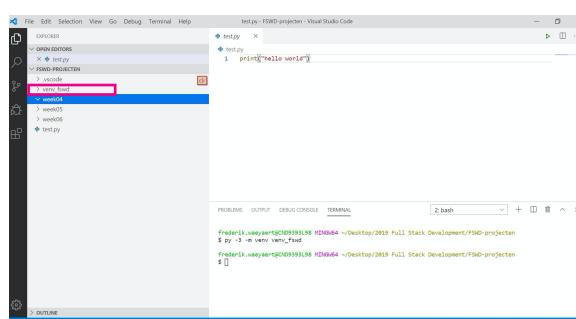
- Zorg dat de folder (FSWD-projecten) waarin je de venv wil aanmaken openstaat in VS Code.
- Maak in de root van het project een test.py bestand aan met de inhoud `print("hello world")`;
- Open een nieuwe terminal in VS Code





- Geef volgend commando in om een nieuwe *virtual environment* - op basis van de default Python interpreter - te maken met als naam venv\_fswd.  
Je zal zien dat er een folder bijkomt in het project. Deze folder stelt de venv voor.

```
py -m venv venv_fswd
```



- Verander de interpreter van het project via **ctrl+shift+p** Python: select interpreter naar de net aangemaakte *virtual environment*: ./venv\_fswd/bin/python

```
Python 3.8.1 64-bit ('venv_fswd': venv)
./venv_fswd/bin/python
```

Je zal zien dat er in het bestand .vscode/settings.json een regel is bijgekomen.  
“`python.pythonPath`” deze verwijst naar je venv.

Eventueel kan je dit ook doen (als er een .py bestand openstaat) via de blauwe balk links onderaan.

- Om zeker te zijn dat je straks de packages op de correcte interpreter installeert, run je even test.py. Je zal zien dat er onderaan een nieuwe terminal wordt geopend.  
De naam van de venv staat nu vooraan elk commando. Het is belangrijk dat hier (`venv_fswd`) staat.

```
(venv_fswd) frederik.waeyaert@PCNAAM $
```

### 2.1.3 Packages toevoegen aan venv - Windows

Voer volgende commando's uit om de correcte packages te installeren op de venv. Je gebruikt hiervoor pip. Pip is een package installer die wordt geïnstalleerd bij de installatie van Python.

- Zorg dat je de laatste versie van pip hebt geïnstalleerd.

```
(venv_fswd) pip install pip --upgrade
```

- Controleer welke packages er reeds geïnstalleerd zijn op deze venv.  
Je krijgt een lijst te zien met alle geïnstalleerde packages in de terminal. De lijst is redelijk beperkt.

```
(venv_fswd) pip list
```

- Installeer de packages om met Flask te werken en de correcte CORS headers mee te sturen in de responses van de webserver.

```
(venv_fswd) pip install flask  
(venv_fswd) pip install flask-cors
```

- Installeer de packages autopep8, om de python code volgens PEP8 afspraken te formateren.

```
(venv_fswd) pip install autopep8
```

- Controleer opnieuw welke packages er geïnstalleerd zijn op de venv.  
De lijst zal nu veel uitgebreider zijn dan daarnet.

```
(venv_fswd) pip list
```

Tip: Heb je onvoldoende rechten op de console? Geef je PowerShell dan extra rechten.

```
set-executionpolicy remotesigned  
bevestig eventueel met (Y)
```

### 3 Instellen van een virtual environment (venv) - macOS

Voor de werkwijze op een laptop met Windows ga naar hoofdstuk 2 Instellen van een virtual environment (venv) – Windows

Voor de back-end code gaan we verschillende packages toevoegen aan de python interpreter. Een package toevoegen doe je in het command prompt via het commando `pip`.

Het nadeel van al de packages toe te voegen aan de default python interpreter (`usr\bin\Python`) is dat je de standaard interpreter “vervult” met allerlei packages die je wil testen.

Een betere manier is:

- Om voor elk nieuw project een kopie (*een venv: virtual environment*) te nemen van de default python interpreter.
- De venv in dezelfde folder als het VS Code project te bewaren.
- Het VS Studio project te koppelen aan de venv.
- Alle packages die je nodig nodig (denkt) te hebben toe te voegen aan de venv.

Het voordeel van op deze manier te werken is:

- De default interpreter wordt niet vervuld met onnodige packages.
- Als je het project kopiëert naar een andere computer wordt de venv (met alle packages) mee gekopieerd.
- Heb je verkeerde packages toegevoegd, of krijg je conflicten, dan kan je eenvoudig de venv verwijderen maar blijft de default interpreter “clean”.

#### 3.1 Aanmaken venv voor de lessen FSWD - macOS

Om niet per week een nieuwe venv aan te moeten maken, plaatsen we alle projecten in 1 “root” map. In deze “root” map plaatsen we een venv.

Vanaf nu openen we dus altijd deze “root” map.

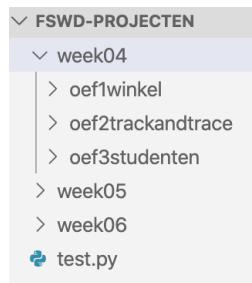
##### 3.1.1 Mappenstructuur voor lessen FSWD

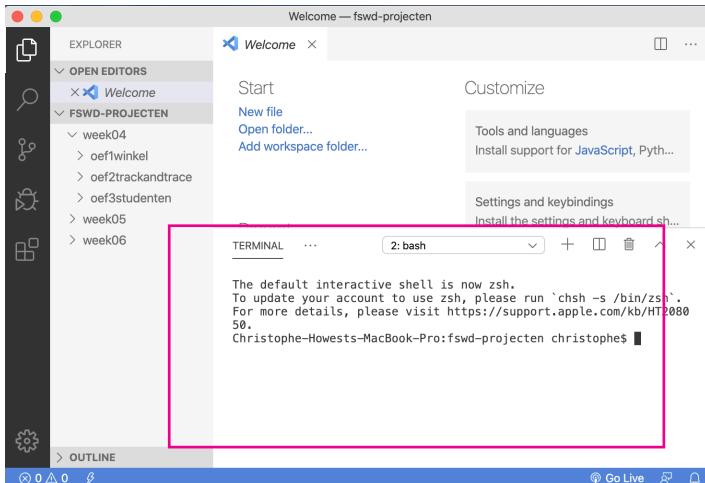
- Maak een folder FSWD-projecten aan.
- Maak een folder week04, week05,... aan in de folder FSWD-projecten.
- Plaats de labo oefeningen in week04.
- Open de folder FSWD-projecten in VS Code.



##### 3.1.2 Venv aanmaken - macOS

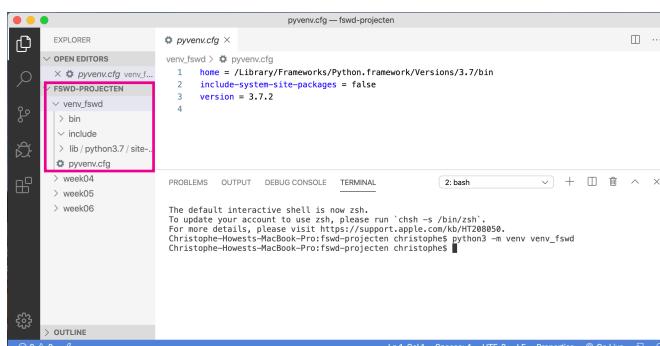
- Zorg dat de folder (FSWD-projecten) waarin je de venv wil aanmaken openstaat in VS Code.
- Maak in de root van het project een test.py bestand aan met de inhoud `print("hello world")`;
- Open een nieuwe terminal in VS Code





- Geef volgend commando in om een nieuwe *virtual environment* - op basis van de default Python interpreter - te maken met als naam `venv_fswd`.  
Je zal zien dat er een folder bijkomt in het project. Deze folder stelt de venv voor.

```
python3 -m venv venv_fswd
```



- Verander de interpreter van het project via **ctrl+shift+p** Python: select **interpreter naar de net aangemaakte virtual environment**: `./venv_fswd/bin/python`

```
Python 3.8.1 64-bit ('venv_fswd': venv)
./venv_fswd/bin/python
```

Je zal zien dat er in het bestand `.vscode/settings.json` een regel is bijgekomen.  
“`python.pythonPath`” deze verwijst naar je venv.

Eventueel kan je dit ook doen (als er een `.py` bestand openstaat) via de blauwe balk links onderaan.

- Om zeker te zijn dat je straks de packages op de correcte interpreter installeert, run je even `test.py`. Je zal zien dat er onderaan een nieuwe terminal wordt geopend.  
De naam van de venv staat nu vooraan elk commando. Het is belangrijk dat hier (`venv_fswd`) staat.

```
(venv_fswd) Christophe-Howests-MacBook-Pro:fswd-projecten christophe$
```

### 3.1.3 Packages toevoegen aan venv - macOS

Voer volgende commando's uit om de correcte packages te installeren op de venv. Je gebruikt hiervoor pip. Pip is een package installer die wordt geïnstalleerd bij de installatie van Python.

- Zorg dat je de laatste versie van pip hebt geïnstalleerd.

```
(venv_fswd)      pip install pip --upgrade
```

- Controleer welke packages er reeds geïnstalleerd zijn op deze venv.  
Je krijgt een lijst te zien met alle geïnstalleerde packages in de terminal. De lijst is redelijk beperkt.

```
(venv_fswd)      pip list
```

- Installeer de packages om met Flask te werken en de correcte CORS headers mee te sturen in de responses van de webserver.

```
(venv_fswd)      pip install flask  
(venv_fswd)      pip install flask-cors
```

- Installeer de packages autopep8, om de python code volgens PEP8 afspraken te formateren.
- Controleer opnieuw welke packages er geïnstalleerd zijn op de venv.  
De lijst zal nu veel uitgebreider zijn dan daarnet.

```
(venv_fswd)      pip list
```

## 4 Oefening 1: Winkel

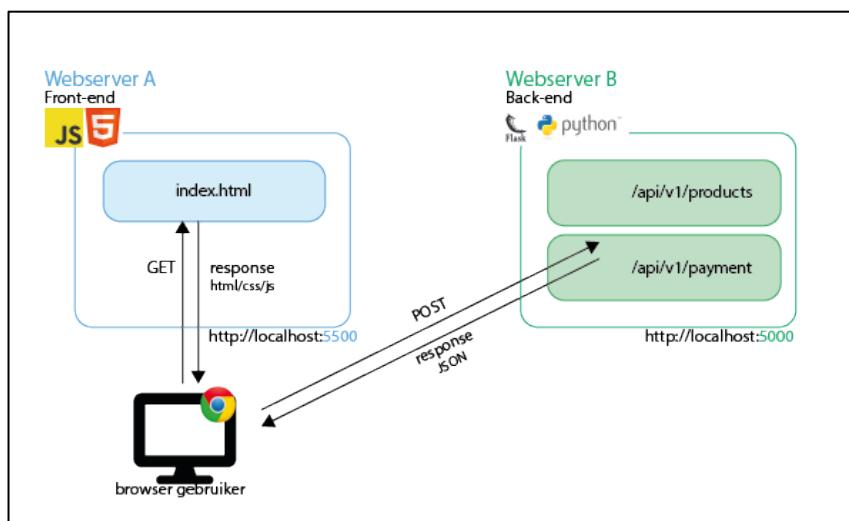
In de eerste oefening gaat de gebruiker via de browser in interactie met de **website A**, deze draait op webserver A.

In onze situatie is het adres van deze webserver [localhost:5500/index.html](http://localhost:5500/index.html)

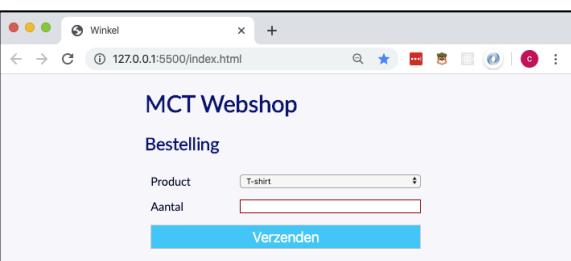
Op index.html staat een formulier dat zijn data POST naar het endpoint dat draait op **Webserver B**. Dit gebeurt bij het submitten van de form.

**Webserver B** zal uiteindelijk de data verwerken en wegschrijven naar de terminal. In de toekomst zal dit naar een database gebeuren.

Tenslotte stuurt de **webserver B** een response in JSON naar de browser.



### 4.1 Front-end

 oef1winkel/front

Open index.html uit de folder oef1winkel/front in **VS Code**, dit onderdeel is reeds uitgewerkt voor ons.

Het bevat een formulier met een select en input html element.

- Verken de form-tag via de DevTool:
  - Welke method is ingesteld?
  - Welke action is ingesteld? Waarom staat het volledige http:// adres ingesteld?
  - Wat is het verschil tussen de name en het id attribute?
  - Wat is de betekenis van de attributen min, max, required?
- Verzend het formulier.
  - Wat gebeurt er? Waarom?
- Vul als aantal -100 in.
  - Wat gebeurt er? Waarom?

## 4.2 Back-end



### oef1winkel/back

Open app.py in **VS Code**. Je zal zien dat er nog geen routes zijn geprogrammeerd.

#### 4.2.1 Voorbereiding

Er is reeds:

- Een dictionary aangemaakt waarmee je straks de prijs kan opvragen.
- De correcte imports van flask en flask\_cors staan in app.py.

#### 4.2.2 Opdracht

- Activeer, onmiddellijk nadat de variabele `app` wordt ingesteld, de CORS headers door de constructor van de CORS klasse op te roepen en de variabele `app` hieraan mee te geven als parameter.

```
app = Flask(__name__)
CORS(app)
```

- Voeg een route toe `/api/v1/products`
  - Deze route zal de **GET** method aanvaarden.
  - Deze route geeft als response de volledige dictionary terug (in json formaat) en de response code 200.
- Voeg een route toe `/api/v1/payment`
  - Deze route zal de **POST** method aanvaarden.
  - Print, ter controle, alle data die binnenkomt via `request.form` in de terminal van VS Code.
  - Bewaar de gegevens (aantal en id) die je nodig hebt uit de POST request in twee lokale variabelen zodat je deze straks eenvoudig kan oproepen.
  - Zoek de bijhorende prijs in de dictionary.
  - Bereken het totaal te betalen bedrag.
  - Schrijf naar de terminal van VS code de boodschap “Besteld: aantal stuks \t naam --> totaal bedrag”
  - Als het gelukt is stuur je een JSONresponse met code 201 naar de browser.

```
{
    status: "success"
}
```

- Indien er een error optreedt in de programmeercode sturen we een JSON response met code 500.

```
{
    status: "error"
}
```

In de volgende les CRUD zullen we zien dat we het response meer kunnen standariseren.

### 4.2.3 Testen

Test de code door zowel de **front-end** als de **back-end** op te starten. Op welke poorten draaien beide?

- Start de front-end via **Live Server Extension**
- Start de back-end via **Debug > Run and Debug > Python Bestand**.
  - Kies NIET voor Flask in de keuzelijst maar voor Python Bestand!!!
  - Maak eventueel een launch.json file aan zodat je niet iedere keer dezelfde keuze moet maken.

Interpreteer de output in het terminalvenster van **VS Code**.

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Geef in de **back-end** code een extra parameter mee met `app.run()` zodat de foutberichten worden getoond in de browser.

```
if __name__ == '__main__':
    app.run(debug=True)
```

Stop de **back-end** server via **ctrl+c** en herstart hem. Interpreteer de output in het terminalvenster van **VS Code**.

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 128-350-968
```

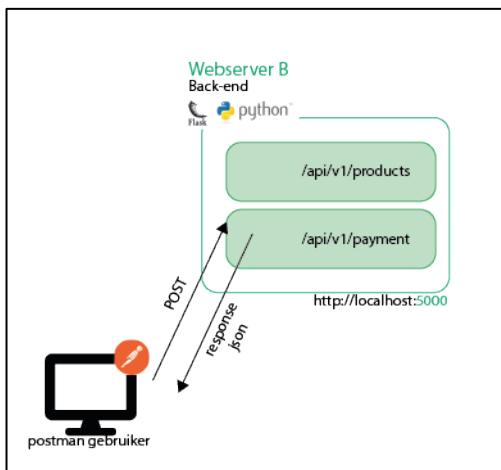
Doe in de **front-end** een bestelling van 10 pulls. In de terminal van de **back-end** zie je volgende output.

```
Volgende data komt binnen in de POST request
ImmutableMultiDict([('product', '101'), ('aantal', '10'), ('send', 'Verzenden')])
*** Besteld 10 stuks    pull --> 220
127.0.0.1 - - [22/Feb/2020 11:29:51] "POST /api/v1/payment HTTP/1.1" 201 -
```

Doe in de **front-end** een bestelling van -100 pulls. Je zal zien dat de bestelling nooit verstuurd wordt naar de **back-end**.

We stellen ons de vraag of onze oplossing veilig is. In de **front-end** beperken we toch de invoer voor de gebruiker.... Dus veilig?!?

#### 4.2.4 ‘Hacken’ met Postman



We zagen reeds in week 2 dat we via Postman requests kunnen versturen naar een endpoint.

Deze techniek kan mis/gebruikt worden om verkeerde informatie te versturen naar de endpoints van een API. Zonder dat de input gecontroleerd wordt aan de kant van de **front-end**.

Doe eerst een **GET** request naar het endpoint: `/api/v1/products`  
Je krijgt een JSONobject terug

A screenshot of the Postman interface. The method dropdown says 'GET'. The URL field contains 'http://127.0.0.1:5000/api/v1/products'. On the right, a blue 'Send' button is visible.

Doe vervolgens een **GET** request naar het endpoint `/api/v1/payment`.  
Je krijgt volgende response. Hoe komt dit? Wat is de status code?

A screenshot of the Postman interface. The method dropdown says 'GET'. The URL field contains 'http://127.0.0.1:5000/api/v1/payment'. The status bar at the top right shows 'Status: 405 METHOD NOT ALLOWED'. The response body is displayed in 'Pretty' format:

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
2 <title>405 Method Not Allowed</title>
3 <h1>Method Not Allowed</h1>
4 <p>The method is not allowed for the requested URL.</p>
```

Doe vervolgens een **POST** request naar het endpoint `/api/v1/payment`.  
Je krijgt volgende response. Hoe komt dit? Wat is de status code?

A screenshot of the Postman interface. The method dropdown says 'POST'. The URL field contains 'http://127.0.0.1:5000/api/v1/payment'. The status bar at the top right shows 'Status: 500 INTERNAL SERVER ERROR'. The response body is displayed in 'Pretty' format:

```
1 {
2   "status": "error"
3 }
```

We stellen Postman zodanig in dat er een **POST**-request verzonden wordt met de correcte form gegevens. Dit doe je via body > form-data

- Waar halen we de correcte benamingen om bij *key* in te vullen?
- Vanwaar komt de statuscode van de response?

POST http://127.0.0.1:5000/api/v1/payment

Body (form-data)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> product	102...	
<input checked="" type="checkbox"/> aantal	10	

Status: 201 CREATED Time: 51ms Size: 202 B Save Response

```

1 {
2   "status": "succes"
3 }

```

Controleer de terminal van de **back-end** code

```
****Volgende data komt binnen in de POST request
ImmutableMultiDict([('product', '102'), ('aantal', '10')])
*** Besteld 10 stuks koffie tas --> 110
```

Wat zou er gebeuren als we toch een negatief aantal doorsturen via Postman?

POST http://127.0.0.1:5000/api/v1/payment

Body (form-data)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> product	102	
<input checked="" type="checkbox"/> aantal	-10	

Status: 201 CREATED Time: 34ms Size: 202 B Save Response

```

1 {
2   "status": "succes"
3 }

```

Test dit door na te gaan wat er naar de terminal van de **back-end** wordt weggeschreven.  
Hoeveel zou er van de VISA-kaart worden afgetrokken?

```
****Volgende data komt binnen in de POST request
ImmutableMultiDict([('product', '102'), ('aantal', '-10')])
*** Besteld -10 stuks koffie tas --> -110
```

We leren hieruit dat valideren **enkel** via Javascript/HTML niet veilig is. Daarom moet je én aan de kant van de browser én aan de kant van de server de binnenkomende data valideren.

- We passen de code in de **back-end** zodanig aan dat we
  - Controleren of het product ID in de dictionary staat
  - én of het aantal groter dan 0 en kleiner dan 100 is.
  - Wordt er niet voldaan dan geef je een 400 response code en volgend json response.

Controleer na deze aanpassingen de response in Postman.

POST payment

payment

POST http://127.0.0.1:5000/api/v1/payment

Send Save

Params Authorization Headers Body Pre-request Script Tests Cookies Code Comments (0)

Body form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION	...	Bulk Edit
product	100			
aantal	-20			
Key	Value	Description		

Body Cookies Headers (4) Test Results Status: 400 BAD REQUEST Time: 10 ms Size: 223 B Save Download

Pretty Raw Preview JSON

```
1 {  
2   "message": "Doorgegeven waarden zijn niet geldig",  
3   "status": "error"  
4 }
```

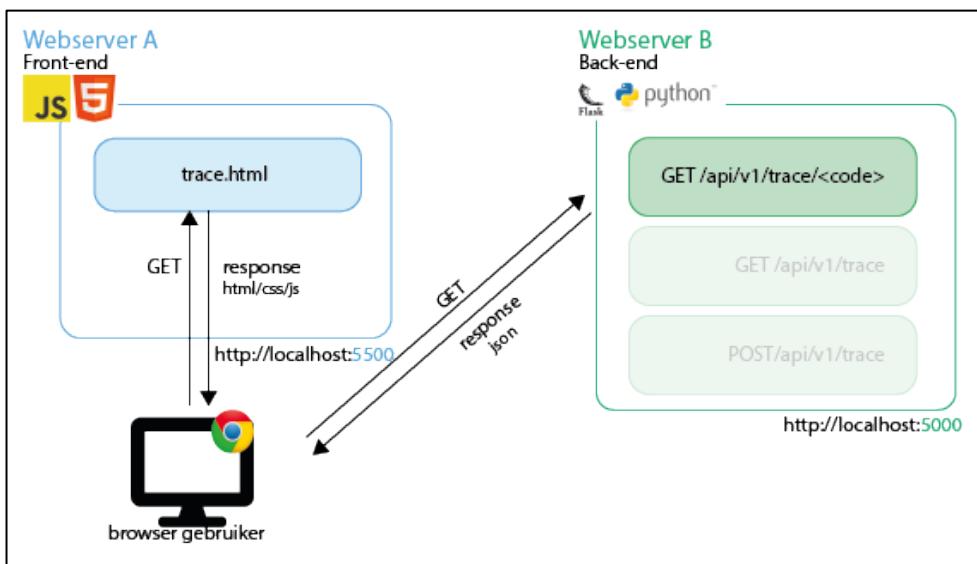
## 5 Oefening 2: Track and Trace

### oef2TrackAndTrace

In de tweede oefening werk je een "Track and Trace"-applicatie uit om een postpakket te lokaliseren.

Het is de bedoeling dat:

- De gebruiker in de **front-end** een tracking nummer ingeeft.
- Dit tracking nummer wordt, via een fetch call, verstuurd naar het endpoint `/api/v1/trace/<code>` van de api op de **back-end**.
  - Omdat je het stuurt via Javascript, ben je niet verplicht om het action attribute van een form-tag te gebruiken. Het volstaat om het click-event van de button op te vangen van javascript.
- De **back-end** haalt de bijhorende tracking informatie uit de variabele `dict_postpakketten` en stuurt deze als response (JSON formaat) naar de HTML pagina.
- De **front-end** ontvangt de data, toont deze data in het HTML document en past de innerHTML en de css klasse van de verschillende html elementen aan om visueel de correcte informatie weer te geven.



### 5.1 Back-end

#### oef2TrackAndTrace/back

##### 5.1.1 Voorbereiding

```
dict_postpakketten = {
    "s007": {
        "naam": "laprudence",
        "postcode": "9000",
        "afgifte": "2019-01-01",
        "sorteercentrum": "2019-01-02",
        "onderweg": None,
        "bezorgd": None
    },
    "s008": {
        "naam": "waeyaert",
        "postcode": "9500"
    }
}
```

De "status" van alle pakketten wordt bewaard in de variabele `dict_postpakketten`.

Aan de hand van de datum kan je zien waar het pakket zich bevindt. Het pakket s007 is op 2 januari afgeleverd in het sorteercentrum. De laatste twee stappen staan nog op None.

Er is reeds een route geschreven: `/api/v1/trace`

De route `/api/v1/trace` geeft een JSON object als response met alle postpakketten die te traceren zijn.

Deze route aanvaardt op dit moment enkel de GET method.

Controleer in Postman de correcte werking van dit endpoint. Welk verschil zie je in de value van “onderweg”, is de waarde nog steeds None in Postman?

### 5.1.2 Opdracht

Schrijf een extra route `/api/v1/trace/<code>` die enkel een GET method aanvaardt.

Zoek de informatie van een bepaald pakket op volgens de meegegeven code.

(bijvoorbeeld: `/api/v1/trace/s007`)

- Controleer of de binnenkomende tracking code tussen de keys staat van deze dictionary.
  - Als dit niet het geval is, returnen we een JSON response met code 404.

```
{  
    status: "error"  
}
```

- Bij een geldige tracking code, returnen we een JSON response met code 200. De inhoud van de response is:

```
{  
    "detail": {  
        "afgifte": "2018-11-11",  
        "bezorgd": "2018-11-17",  
        "naam": "waeyaert",  
        "onderweg": "2018-11-16",  
        "postcode": 8500,  
        "sorteercentrum": "2018-11-15"  
    },  
    "trackcode": "s008"  
}
```

Controleer in Postman de correcte werking van je endpoint. Bekijk aandachtig de teruggestuurde headers. Zullen we straks de endpoints zonder problemen kunnen aanspreken?

De oplossing is om er voor te zorgen dat de `Access-Control-Allow-Origin` header wordt meegestuurd met elke response die vertrekt van je webserver. Stel de CORS in.

Bekijk aandachtig de teruggestuurde headers in Postman. Zie je de verandering?

## 5.2 Front-end

 oef2TrackAndTrace/front

### 5.2.1 Voorbereiding

Voordat we beginnen bekijk eerst de gegeven HTML-structuur in `_layout.html`, dit ontwerp is je doorgestuurd door een front-end designer.

- We zien dat er geen form-tag aanwezig is.  
We zullen straks via javascript het click event opvangen van de input submit button.
- Verschillende html-elementen hebben een css klasse `js-*`.
- Bekijk in de DevTool hoe er in deze (voorlopig) statische pagina - via CSS klassen - onderscheid wordt gemaakt tussen de *huidige* “stap”, een “stap” die is *afgerond* en *toekomstige* stappen bij het verwerken van een postpakket.

```

<main class="c-tack">
  <h1 class="c-track__title">Track and Trace</h1>
  <section class="c-track__cover"></section>
  ▶<section class="c-track__search">...</section>
  ▶<section class="c-track__info-sender js-section-info-sender">...</section>
  ▶<section class="c-track__trace js-section-trace">
    ▶<div class="c-step c-step--done js-step-drop-off">...</div>
    ▶<div class="c-step c-step--done c-step--active js-step-warehouse">...</div>
    ▶<div class="c-step js-step-out-for-delivery">...</div>
    ▶<div class="c-step js-step-delivered">...</div>
  </section>
</main>

```

## 5.2.2 Opdracht

Je werkt in **index.html**. Dit is een identieke kopie van **\_layout.html**.

- Verwijder de css klassen `c-step--done` en `c-step--active` uit de div elementen.
- Verberg de section `.js-section-info-sender` en `.js-section-trace` door de extra css klasse `.u-hide` toe te voegen.

Inspecteer even de css code om te zien, wat deze klasse doet.

```

<main class="c-tack">
  <h1 class="c-track__title">Track and Trace</h1>
  <section class="c-track__cover"></section>
  ▶<section class="c-track__search">...</section>
  ▶<section class="c-track__info-sender u-hide js-section-info-sender">...</section>
  ▶<section class="c-track__trace u-hide js-section-trace">
    ▶<div class="c-step js-step-drop-off">...</div>
    ▶<div class="c-step js-step-warehouse">...</div>
    ▶<div class="c-step js-step-out-for-delivery">...</div>
    ▶<div class="c-step js-step-delivered">...</div>
  </section>
</main>

```

- Koppel `datahandler.js` en `trace_app.js` aan `index.html`.
  - In `datahandler.js` bevindt zich een functie `handleData()` die je kan **hergebruiken** om `fetch` uit te voeren.
  - In dit labo geven we steeds de eerste 3 parameters mee:
    - `url` = wat is de url van het API endpoint
    - `callback bij succes` = welke functie moet worden uitgevoerd als er een geldige response (response code 200 range) van de server teruggekomen is?
    - `callback bij een error` welke functie moet worden uitgevoerd als er een ongeldige response van de server teruggekomen is?
- Als de DOM geladen is, voeg je een `clickevent` toe aan de button `.js-search-btn`. In dit event:
  - Vraag je de waarde op die in het tekstvak staat van `.js-search-txt`.

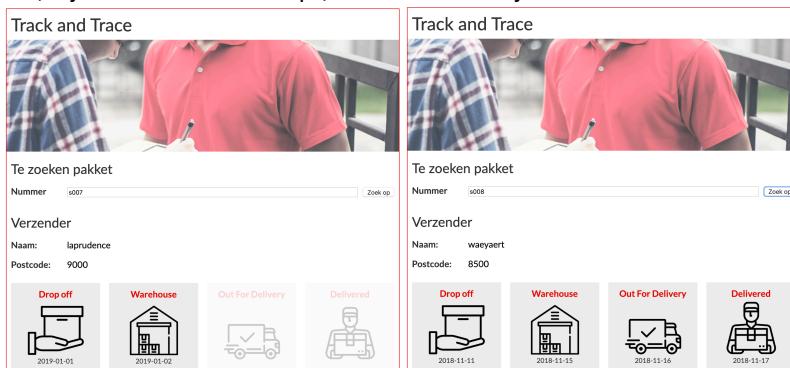
- Je gebruikt de property `value` in plaats van property `innerHTML` omdat het hier over een input element gaat.
- Bevraag je het endpoint, door de function `handleData()` aan te roepen.
  - Geef 3 parameters mee.
    - De url van je endpoint. De url zal als volgt opgebouwd zijn:  
``http://127.0.0.1:5000/api/v1/trace/${inhoudTxt}``
    - De functienaam die moet worden uitgevoerd als er een geldige response terugkomt. (een callback)
    - De functienaam die moet worden uitgevoerd bij een ongeldige response. (een callback)
- Schrijf de callback functie `callbackVerwerkStatus(jsonObject)`. Deze krijgt altijd een jsonobject mee als parameter.
- Schrijf de callback functie `callbackErrorStatus(response)`. Deze krijgt altijd een response object mee als parameter.

*Let op: de parameter kan ook als undefined binnenkomen.*

- In de functie `callbackErrorStatus(response)`
  - Verberg de onderste 2 sections door de css klasse `u-hide` toe te voegen aan hun `classList`.
 

```
document.querySelector('.js-section-trace').classList.add('u-hide')
```
  - Als de parameter `response` bestaat en de status code is 404:
    - Verander je de value van `.js-search-txt` naar 'geen geldig postpakket code'.
    - Anders verander je de value van `.js-search-txt` naar 'fout bij opvragen'.
- In de functie `callbackVerwerkStatus(jsonObject)`
  - Maak je de html-elementen `.js-section-trace` en `.js-section-info-sender` zichtbaar. Dit kan je doen door uit de classlist, van deze html elementen, de css klasse `u-hide` te verwijderen.
 

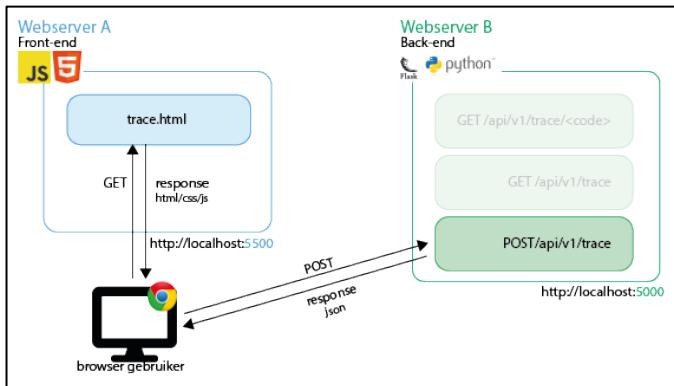
```
document.querySelector('.js-section-trace').classList.remove('u-hide')
```
  - Haal de gegevens (naam, postcode, datum, ...) op uit de parameter `jsonObject` en voeg deze toe via `innerHTML` aan de juiste html-elementen.
  - Door middel van extra css classes (`c-step--done` en `c-step--active`) toe te voegen aan de verschillende `div.js-step-*` html elementen; kunnen we de opacity van de iconen en de border-bottom aanpassen.
    - Hoe kan je in het JSON object afleiden dat we de "actieve stap" bereikt hebt?  
 Hoe vertalen we dit in een if-structuur?
  - Test de code door eerst het pakket s007 op te zoeken en nadien het pakket s008. We zien dat, bij de verschillende steps, de css classes blijven staan



Schrijf een functie `resetTraceSteps()` die de `c-step--done` en `c-step--active` css classes verwijderd. Roep deze functie op voordat de steps opvult.

- Werk de applicatie verder af door een extra EventListener (input) toe te voegen aan het tekstvak. Iedere maal er wordt getypt in het tekstvak, verberg je de onderste sections (.js-section-info-sender en .s-section-trace).

## 5.3 Thuisopdracht



### 5.3.1 Back-end

Er is reeds een route `/api/v1/trace` geschreven die de `GET`method aanvaardt. We zullen dezelfde route gebruiken om een postpakket toe te voegen.  
 Toevoegen zal altijd via de `POST`method verlopen.

Breed deze route uit, zodat deze de `POST` én `GET` method verwerkt.

- Pas de function decorator aan zodat beide methods worden aanvaard.
- Binnen de functie schrijf je een if-structuur om het onderscheid tussen beide requests te maken.
- Als het toevoegen gelukt is
  - Schrijf je een lijn weg naar de terminal van VS Code.
  - Return je een JSON object `{"status": "ok"}` en de code 202.

### 5.3.2 Front-end

Gebruik toevoegen.html, de tekstvelden en knop zijn reeds geschreven.  
 De action van deze form is je endpoint die je zonet schreef, de method is POST.

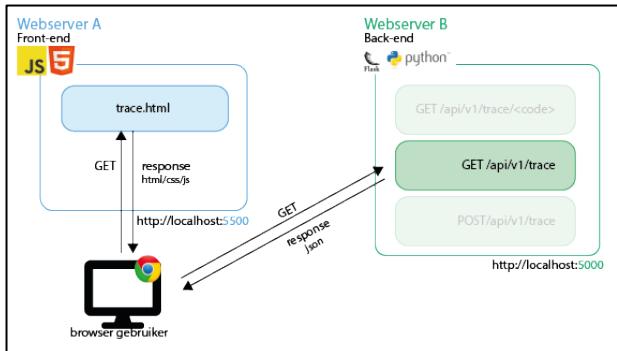
### 5.3.3 Resultaat

Vergeet niet dat de back-end moet opgestart zijn (herstart hem eventueel indien er wijzigingen zijn gemaakt in de Python code) om de front-end te testen!

Browser toevoegen.html	Terminal VS Code
<p><b>Browser toevoegen.html</b></p> <p>Track and Trace</p>  <p>Voeg een pakket toe aan de track and trace</p> <p>Nummer: 9999</p> <p>Naam: Johan</p> <p>Postcode: 1000</p> <p>Verzend naar bedien</p> <p>127.0.0.1:5000/api/v1/trace</p> <pre>{   "status": "ok" }</pre>	<p><b>Terminal VS Code</b></p> <p>Nieuw pakket toegekomen: s999: Johan naar 1000 127.0.0.1 - [23/Feb/2020 12:12:39] "POST /api/v1/trace HTTP/1.1" 202 -</p>

### 5.3.4 Uitbreiding

Je kan een front-end schrijven om de data van het endpoint `GET /api/v1/trace` te tonen, dit is een ideale herhaling van de leerstof van labo Week 02.



### 5.3.5 Extra uitdaging

Als extra uitdaging kan je proberen om het ingevoerde postpakket toe te voegen aan de `dict_postpakketten`. Zolang de server niet afsluit, blijft het nieuwe postpakket bewaard in de dictionary, je kan dit nieuw pakket nu ook opvragen via `index.html` via zijn nummer.  
Wat is de key, wat is de value die je bewaart in `dict_postpakketten`?

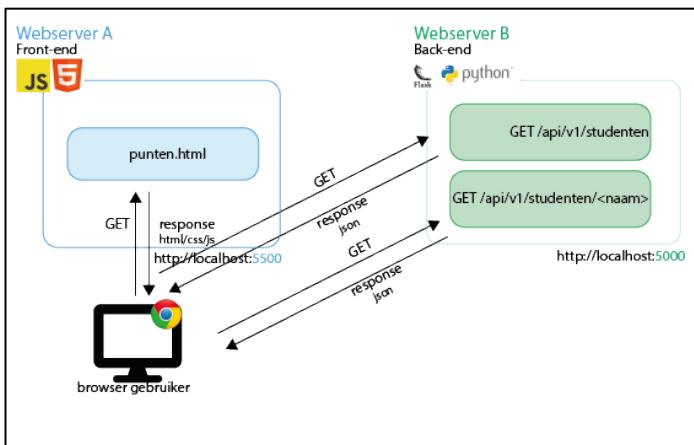
## 6 Oefening 3: Puntenlijst

### Oef3studenten

Je maakt een applicatie die – bij het laden – een keuzelijst toont van alle studenten die zijn bewaard in de dictionary `dict_punten`.

Wordt er een student gekozen – input event - uit de keuzelijst, dan wordt er van die student de resultaten getoond op dezelfde pagina.

Er zullen dus twee fetch calls vertrekken vanaf je index.html naar de API.



### 6.1 Back-end



#### Oef3studenten/back

##### 6.1.1 Opdracht

Je maakt twee routes die beide een GET request afhandelen.

- `/api/v1/studenten`
- `/api/v1/studenten/<naam>`

Test de verschillende endpoints door een request te doen via Postman.

Zorg dat er geen problemen optreden met CORS. Hoe stel je dit in?

Het endpoint `/api/v1/studenten` geeft volgend JSON-object.

Tip: een list van enkel de keys uit een dictionary oprovragen doe je als volgt

```
lst_keys = list(dict_punten.keys())
```

```
{
  "studenten": [
    "Blanck",
    "Bodin",
    "Bogaert",
    "Bossuyt",
    "Brion",
    "Brock",
    "Bruggeman",
    "Brutyn",
    "Bulckaen"
  ]
}
```

Het endpoint `/api/v1/studenten/<naam>` geeft volgend JSON-object als er **Boddin** opgegeven is als `<naam>`. In het json response herhaal je de zoekterm bij de key naam, en toon je een list van de punten bij de key rapport.

```
{  
    "naam": "Boddin",  
    "rapport": [  
        {  
            "module": "Full-stack Web Dev",  
            "punt": 10  
        },  
        {  
            "module": "Sensors & interfacing",  
            "punt": 7  
        },  
        {  
            "module": "Data Management",  
            "punt": 10  
        },  
        {  
            "module": "UI Design",  
            "punt": 1  
        },  
        {  
            "module": "Project 1",  
            "punt": 9  
        }  
    ]  
}
```

## 6.2 Front-end

### Oef3studenten/front

#### 6.2.1 Opdracht

- **Voorbereiding**

Zorg dat app.js en datahandler.js zijn gekoppeld met index.html

- **Event: Laden**

Bij het laden van de pagina, halen we alle namen van de studenten op via de functie `handleData()` die je zojuist toevoegde via datahandler.js.

- Geef het endpoint en callback functie op.
- In de callback functie vul je – via de eigenschap `innerHTML` van het `select` element - de verschillende studenten op.
  - Kies zowel voor de *zichtbare tekst* als voor de `value` van de option-tag de naam van de student.

**Puntenboek**

**Kies een student**

Studenten Bruggeman 

```
<select name="studenten" class="js-studenten">  
    <option value="Blanck">Blanck</option>  
    <option value="Boddin">Boddin</option>  
    <option value="Bogaert">Bogaert</option>  
    <option value="Bossuyt">Bossuyt</option>  
    <option value="Brion">Brion</option>  
    <option value="Brock">Brock</option>  
    <option value="Bruggeman">Bruggeman</option>  
    <option value="Brutyn">Brutyn</option>  
    <option value="Bulckaen">Bulckaen</option>  
</select>
```

- **Event: Input** - Bij veranderen van student in de keuzelijst

Koppel een eventlistener die het veranderen van de input van de keuzelijst opvangt.

Bepaal eerst de aangeduide naam:

```

const veranderStudent = function() {
  const selectedIndex = this.selectedIndex;
  const arrOptions = this.options;
  const selectedValue = arrOptions[selectedIndex].value;
  handleData(`http://127.0.0.1:5000/api/v1/studenten/${selectedValue}` ,
  callbackToonPunten);
};

```

- Als je met een addEventListener werkt, dan stelt `this` het html element voor waaraan het event gekoppeld is. Hier `<select name="studenten" class="js-studenten">.....</select>`
- Bepaal de *geselecteerde index*.  
(Het hoeveelste option element is er gekozen (start met tellen vanaf 0))
- Maak een array met alle *mogelijke options*.
- Vraag vervolgens de *waarde* van de `<option>` met de *geselecteerde index* op uit de array.
- Nadat we de gekozen naam hebben bepaald, halen we via het endpoint `/api/v1/studenten/<naam>`, de bijhorende resultaten op.  
Overloop het rapport van de student en toon elk resultaat in een `<article>`, met bijhorende modulenaam en punt in `div.js-punten-placeholder`. Kijk aandachtig naar de html output en bepaal welk deel je in de for-lus noteert.

**Puntenboek**

### Kies een student

Studenten

#### Detail

Full-stack Web Dev	14
Sensors & interfacing	20
Data Management	19
UI Design	19
Project 1	17

```

<div class="js-punten-placeholder">
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Full-stack Web Dev </div>
    <div class="c-resultaat__punt"> 14 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Sensors & interfacing </div>
    <div class="c-resultaat__punt"> 20 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Data Management </div>
    <div class="c-resultaat__punt"> 19 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> UI Design </div>
    <div class="c-resultaat__punt"> 19 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Project 1 </div>
    <div class="c-resultaat__punt"> 17 </div>
  </article>
</div>

```

- Voeg tenslotte nog een extra css klasse bij het punt. Schrijf hiervoor een functie `bepaalPuntClass(punt)` die "`c-resultaat__punt--geslaagd`" of "`c-resultaat__punt--gebuisd`" returnt voor een punt groter of kleiner dan 10.

**Puntenboek**

### Kies een student

Studenten

#### Detail

Full-stack Web Dev	13
Sensors & interfacing	12
Data Management	12
UI Design	3
Project 1	12

```

<div class="js-punten-placeholder">
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Full-stack Web Dev </div>
    <div class="c-resultaat__punt c-resultaat__punt--geslaagd"> 13 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Sensors & interfacing </div>
    <div class="c-resultaat__punt c-resultaat__punt--geslaagd"> 12 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Data Management </div>
    <div class="c-resultaat__punt c-resultaat__punt--geslaagd"> 12 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> UI Design </div>
    <div class="c-resultaat__punt c-resultaat__punt--gebuisd"> 3 </div>
  </article>
  <article class="c-resultaat">
    <div class="c-resultaat__module"> Project 1 </div>
    <div class="c-resultaat__punt c-resultaat__punt--geslaagd"> 12 </div>
  </article>
</div>

```

# howest

hogeschool