howest
hogeschool

Recap

Frontend en backend

Inhoudsopgave

1 Leerdoelen

- Een volledige **backend** schrijven met de correct CRUD datarepository en API routes.
- Een **frontend** maken die gegevens uit de API routes opvraagt uit de **backend**.
- De **frontend** javascript code structureren met **regions** zodat deze overzichtelijk wordt.
- Werken met **dynamische events** in de **frontend** javascript code.
- In de **frontend** gebruik maken van het **data-** attribuut in HTML en Javascript.

2 Vooraf

Deze week maken we een herhalingsoefening waarbij we de kennis van de voorbije vijf weken toepassen en ook herhalen.

Je zal ook twee nieuwe “concepten” gebruiken tijdens de opdracht.

- De opdracht wordt gemaakt als een *single page website*.
- De EventListeners worden dynamisch aangemaakt.

2.1 Structuur van een Javascript file voor een single page

2.1.1 Werk met regions

De volledige opdracht wordt gemaakt als een *single page website*.
Hierdoor wordt de Javascript snel complex.

Een goede tip is om gebruik te maken van `#region` in het JS-bestand.
VS Code biedt ons de mogelijk om een region “dicht” te klappen.

Op deze manier houden we het overzicht en plaatsen we gemeenschappelijk code bij elkaar.

```
//#region een willekeurige naam

//#endregion
```

We verdelen onze javascript file in volgende regions.

```
//#region ***  DOM references                          **********
//#endregion
//#region ***  Callback-Visualisation - show___       **********
//#endregion
//#region ***  Callback-No Visualisation - callback___ **********
//#endregion
//#region ***  Data Access - get___                    **********
//#endregion
//#region ***  Event Listeners - listenTo___           **********
//#endregion
//#region ***  INIT / DOMContentLoaded                 **********
//#endregion
```

2.1.2 Naamgeving

Ondanks dat het geen “regel” is, raden we aan om te werken met een consequente naamgeving.

- frontend/index.html
- frontend/script/dataHandler.js
- frontend/script/app.js

2.1.3 Structuur app.js

- **Verwijs je naar een DOM element gebruik dan de prefix *html_***

Gebruik je verschillende malen hetzelfde DOMObject, dan ben je beter om deze globaal te declareren zodat je er - in om het even welke function - naar kan verwijzen.
Opgepast hierbij: de declaratie is globaal, maar de initialisatie (het toewijzen) gebeurt pas in de `init()`, want anders ben je niet zeker dat de elementen al aangemaakt zijn in de DOM.

Door met een extra if-structuur te controleren of `html_kleuren` niet `null` is, ben je zeker dat de `querySelector()` een html-element heeft gevonden. `getKleuren()` wordt dus enkel maar uitgevoerd als de querySelector het element vond.
Je vermijdt dus om later volgende error te krijgen "Cannot set property 'innerHTML' of null".

```
//#region ***  DOM references ***
let html_kleurenHolder, html_uitvoerHolder;
//#endregion

//#region ***  INIT / DOMContentLoaded  ***
const init = function() {
  console.info("DOM geladen");
  html_kleurenHolder = document.querySelector(".js-kleuren");
  html_uitvoerHolder = document.querySelector(".js-uitvoer");

  if (html_kleurenHolder) {
    getKleuren();
  }
};

document.addEventListener("DOMContentLoaded", init);
//#endregion
```

- **Om een API aan te spreken gebruiken we de prefix *get***

```
//#region ***  Data Access - get___ ***
const getKleuren = function() {
  handleData("http://linknaar.json", showKleuren);
};
//#endregion
```

- **Kan de callbackgegevens toevoegen aan de DOM.**

```
//#region ***  Callback-Visualisation - show___ ***
const showKleuren = function(data) {
  const arrKleuren = data.colors;
  for (const kleur of arrKleuren) {
    console.info(`${kleur.color}`);
  }
  listenToClickKleur();
};
//#endregion
```

- **Voegt de callback geen gegevens toe aan de DOM.**
  Gebruik de prefix *callback*

```
//#region ***  Callback-No Visualisation - callback___  ***
//#endregion
```

- **Een Eventlistener toevoegen aan een dynamisch object.**
  Gebruik de prefix *listenTo* en gebruik een *anonymous* function in de eventlistener.
  Je zit dat dat deze functie werd opgeroepen in showKleuren(). Eerst voeg je de elementen met een `.js-` *klasse toe via innerHTML aan de DOM. Nadien koppel je een eventlistener aan alle elementen met een* `.js-` *klasse.*

```
//#region ***  Event Listeners - listenTo___  ***
const listenToClickKleur = function() {
  const buttons = document.querySelectorAll(".js-kleurelement");
  for (const b of buttons) {
    b.addEventListener("click", function() {
      const gekozenCode = this.getAttribute("data-code");
      const gekozenCategorie = this.dataset.category;

      html_uitvoerHolder.innerText = `${gekozenCategorie}`;
    });
  }
};
//#endregion
```

2.2 Eventlisteners gebruiken op een dynamisch opgebouwd element met het data-attribute

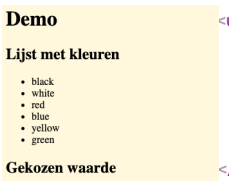
Soms wil je op de website “iets” (black,white, red, blue,…) tonen maar wil je “iets anders” (#000, #FFF, #FF0, #00F,…) bijhouden “achter de schermen”.

Bijvoorbeeld een id, een kleurcode, een categorie,…

Hiervoor gebruiken we het data attribute. Lees hier meer over op:
https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes.

In het voorbeeld tonen we de kleurnaam in de browser maar houden de hexadecimale code en categorie bij.

We **tonen** de *kleurnamen* maar we **houden** de *categorie* en *hexcode* **bij** in een `data-` attribute.
Deze twee gegevens tonen we na het klikken op een kleurnaam uit de lijst.




```
<ul class="js-kleuren">
  <li class="js-kleurelement" data-code="#000" data-category="hue">black</li>
  <li class="js-kleurelement" data-code="#FFF" data-category="value">white</li>
  <li class="js-kleurelement" data-code="#FF0" data-category="hue">red</li>
  <li class="js-kleurelement" data-code="#00F" data-category="hue">blue</li>
  <li class="js-kleurelement" data-code="#FF0" data-category="hue">yellow</li>
  <li class="js-kleurelement" data-code="#0F0" data-category="hue">green</li>
</ul>
```

Hiervoor passen we de javascript code als volgt aan. We zien dat we twee mogelijkheden hebben om in javascript het `data-` *attribute* aan te spreken.

- Via `this.getAttribute('data-naam')`
- of `this.dataset.naam`

Aan jou de keuze welke werkwijze je verkiest.

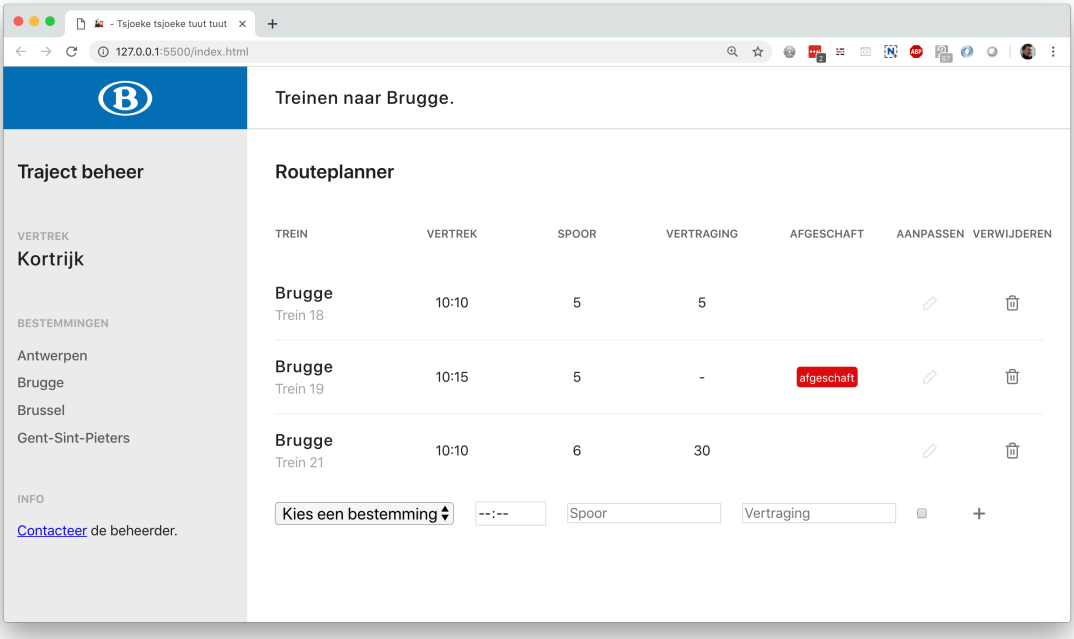
```
//#region ***  Callback-Visualisation - show___ ***
const showKleuren = function(data) {
  const arrKleuren = data.colors;
  let kleurenHTML = "";
  for (const kleur of arrKleuren) {
    kleurenHTML += `<li class="js-kleurelement" data-code="${kleur.code.hex}" data-
category="${kleur.category}">${kleur.color}</li>`;
  }
  html_kleurenHolder.innerHTML = kleurenHTML;
  listenToClickKleur();
};
//#endregion
```

```
//#region ***  Data Access - get___ ***
const getKleuren = function() {
  console.info("kleuren worden geladen van API");
  handleData("data/kleuren.json", showKleuren);
};
//#endregion

//#region ***  Event Listeners - listenTo___ ***
const listenToClickKleur = function() {
  const buttons = document.querySelectorAll(".js-kleurelement");
  for (const b of buttons) {
    b.addEventListener("click", function() {
      const gekozenCode = this.getAttribute("data-code");
      const gekozenCategorie = this.dataset.category;

      html_uitvoerHolder.innerText = `De gekozen kleur heeft een Hex waarde van
${gekozenCode} en behoort tot category ${gekozenCategorie}`;
    });
  }
};
//#endregion
```

3 Oefening

Download de startbestanden van Leho en plaats deze - net zoals de voorbije weken - in de folder projecten_FSWD onder week 06. Op deze manier zal je ook nu de venv met de correcte packages gebruiken.

Bij deze herhalingsoefening is het de bedoeling dat je een onepagewebsite bouwt waarin je de treintijden toont naar verschillende plaatsen.



3.1 Stappenplan:

3.1.1 Database

Importeer de database database/treinen.sql via MySQL Workbench.

3.1.2 Backend

Bouw de **backend** door volgende functies aan te maken in de `DataRepository.py` en stel de bijhorende routes in in `app.py`:

Controleer in `config.py` of de settings voor de verbinding met je database juist staan.

Voorzie in `DataRepository.py` volgende static methods voor de CRUD bewerkingen. Vergeet niet om met SQL parameters te werken, zodat je SQL veilig is.

`read_bestemmingen()`	Returnt de info van alle **bestemmingen**.
`read_treinen()`	Returnt de info alle **treinen**.
`read_trein(treinid)`	Returnt de info van **1 trein** die voldoet aan het meegegeven **treinid**.

`read_treinen_met_bestemming(bestemmingid)`	Returnt de info van **alle treinen**, samen met de info van **de bestemming, toon dit enkel van de treinen die naar 1 bestemming rijden.** Gebruik hiervoor een INNER JOIN, zodat je gegevens kan opvragen van de trein en de bestemming.
	`SELECT *`
	`FROM treinen t INNER JOIN bestemmingen b`
	`ON t.bestemmingID = b.idbestemming`
	`WHERE idbestemming = %s`

`create_trein(…,…,…,…,…)`	Maakt een nieuwe trein aan.
`update_trein(…,…,…,…,…)`	Past de gegevens aan van 1 trein.
`delete_trein(treinid)`	Verwijdert 1 trein op basis van het id.

`update_trein_vertraging (idtrein, vertraging)`	Past enkel het databaseveld `vertraging` van 1 bepaalde trein aan.
	`UPDATE treinen`
	`SET vertraging = %s`
	`WHERE idtrein = %s`

Voorzie in `app.py` de correcte routes voor je API. Binnen elke route spreek je de net geschreven methodes aan uit de `DataRepository` aan.

`/api/v1/bestemmingen`	`GET`	`read_bestemmingen()`
`/api/v1/treinen`	`GET` `POST`	`read_treinen()` `create_trein()`

`/api/v1/treinen/<trein_id>`	`GET`
	`PUT`
	`DELETE`
		`read_trein()`
		`update_trein()`
		`delete_trein()`

`/api/v1/treinen/<trein_id>/vertraging`	`PUT`	`update_trein_vertraging(idtrein, vertraging)` *De vertraging komt binnen in de "payload" van de request.*
`/api/v1//treinen/bestemming/<bestemming_id>`	`GET`	`read_treinen_met_bestemming(bestemmingid)`

Voorzie waar nodig Errorhandling zoals je in het vorige labo aanleerde. Denk er ook aan om de correcte HTMLstatuscode terug te sturen bij elk response.

3.1.3 Postman testing

Test telkens deze routes aan de hand van Postman.
Importeer hiervoor de Postman Collection `FSWD-labo06-TREIN.postman_collection.json` die je in het bronmateriaal terugvindt.

1


```
{
  "bestemmingen": [
    {
      "idbestemming": 100,
      "stad": "Brussel"
    },
    {
      "idbestemming": 101,
      "stad": "Antwerpen"
    },
    {
      "idbestemming": 102,
      "stad": "Gent-Sint-Pieters"
    },
    {
```

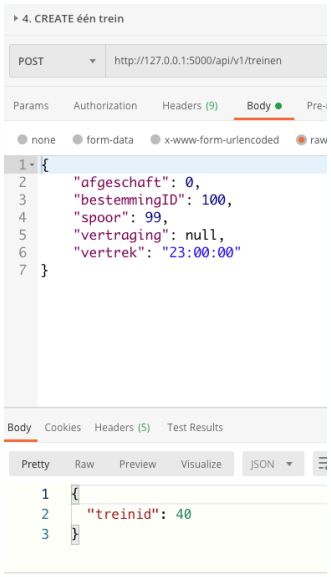

2


```
{
  "treinen": [
    {
      "afgeschaft": 0,
      "bestemmingID": 100,
      "idtrein": 16,
      "spoor": 12,
      "vertraging": null,
      "vertrek": "10:00"
    },
    {
      "afgeschaft": 0,
      "bestemmingID": 101,
      "idtrein": 17,
      "spoor": 11,
```

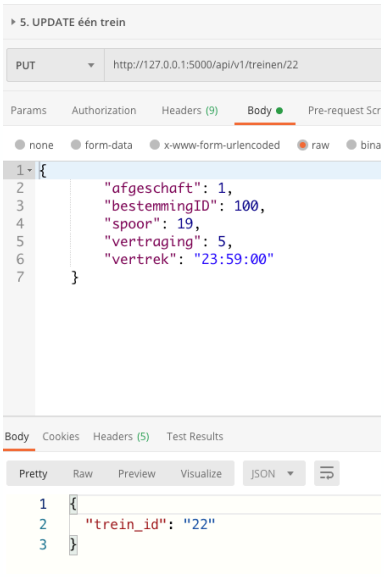

3


```
{
  "afgeschaft": 0,
  "bestemmingID": 100,
  "idtrein": 16,
  "spoor": 12,
  "vertraging": null,
  "vertrek": "10:00"
}
```

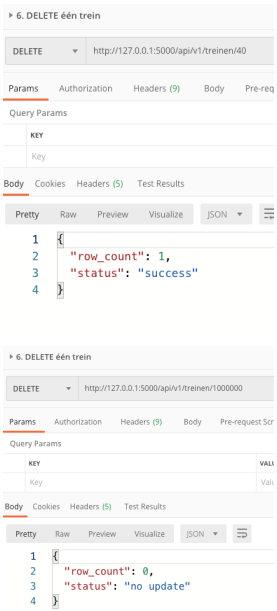

4



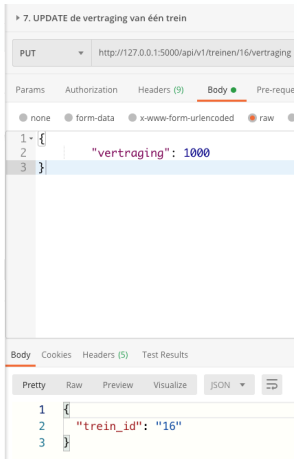
5



6



7



8


```
{
  "treinen": [
    {
      "afgeschaft": 0,
      "bestemmingID": 101,
      "idbestemming": 101,
      "idtrein": 17,
      "spoor": 11,
      "stad": "Antwerpen",
      "vertraging": null,
      "vertrek": "10:00"
    },
    {
      "afgeschaft": 0,
      "bestemmingID": 101,
```

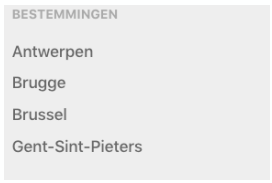
3.1.4 Frontend

Bouw de **frontend** aan de hand van de statische code die reeds aangemaakt is voor jou.

- **Bij het laden van de pagina**

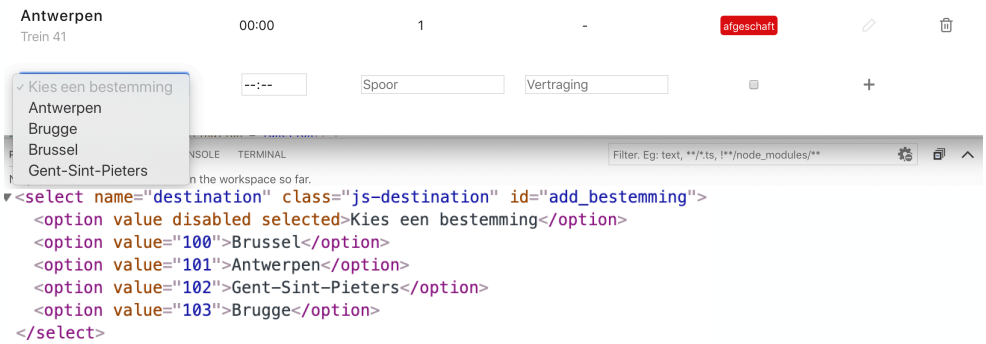
1. Bouw de navigatie op: Haal bij het laden van de pagina de verschillende bestemmingen op via de API en genereer dynamisch (niet hardcoded!) de navigatielinks aan de linkerzijde.

   - Haal de data op in een `get_____()` functie. Binnen deze functie roep je de `handleData()` op.
   - De binnengekomen jsondata van de `handleData()` functie zet je om naar HTML in de `show____()` functie.
     (dit is de callback functie bij een geslaagde fetch)
     - In `showDestination(jsonobject)` vul je de navigatie `.js-destinations` op met de correcte navigatie elementen `li.c-sidebar-item>button.js-station`. Doe dit binnen een lus die het jsonobject overloopt.
       Schenk aandacht aan het `data-` attribute, ook dit zal je correct moeten invullen.




```
<ul class="o-list c-sidebar-list js-destinations">
  <li class="c-sidebar-item">
    <button class="c-sidebar-button js-station" data-destination-id="100">Brussel
    </button>
  </li>
  <li class="c-sidebar-item">
    <button class="c-sidebar-button js-station" data-destination-id="101">Antwerpen
    </button>
  </li>
  <li class="c-sidebar-item">
    <button class="c-sidebar-button js-station" data-destination-id="102">Gent-Sint-
    Pieters</button>
  </li>
  <li class="c-sidebar-item">
    <button class="c-sidebar-button js-station" data-destination-id="103">Brugge
    </button>
  </li>
</ul>
```

     - In `showDestination(jsonobject)` vul je tevens de keuzelijst onderaan de pagina op met de bestemmingen. Deze keuzelijst zal je later gebruiken om nieuwe treinen toe te voegen.
       Je voegt de `option` elementen met de correcte value toe aan `.js-destination`



- Gebruik een dynamische Eventlistener op de klasse van de navigatie.
  Gebruik hiervoor de functie `listenTo____()`
  Print *voorlopig* het bestemmingsid en bestemmingsnaam in de console van Chrome.

- **Controleer jezelf**

Als je bovenstaande stappen goed geprogrammeerd hebt, zal app.js volgende structuur hebben.
Kopieer deze niet onmiddellijk maar probeer deze eerst zelf op te bouwen!

```
//#region ***********  DOM references ***********
let html_destinationHolder, html_routeHolder, html_selectedCity,
html_destinationSelect, html_adaptTrain;
```

```
//#region ***  Callback-Visualisation - show___ ***
const showDestinations = function(jsonObject) {
  //Toon menu

  //Toon keuzelijst

  //Start met het luisteren naar click event van de net aangemaakte menu items
  listenToClickDestination();
};
```

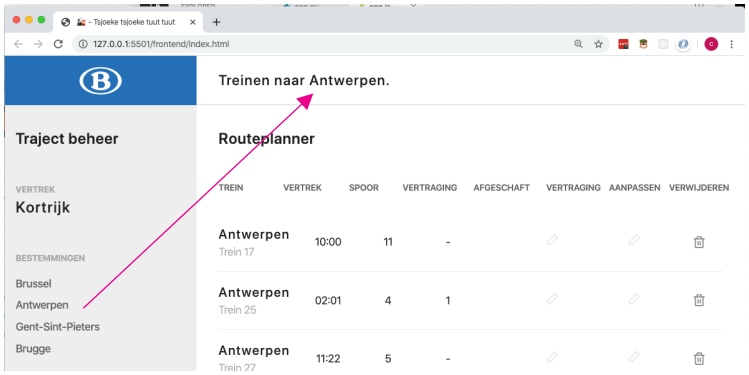
```
//#region ***  Data Access - get___ ***
const getDestinations = function() {
  handleData("http://127.0.0.1:5000/api/v1/bestemmingen", showDestinations);
};
```

```
//#region ***  Event Listeners - listenTo___ ***
const listenToClickDestination = function() {
  const buttons = document.querySelectorAll(".js-station");
  for (const btn of buttons) {
    btn.addEventListener("click", function() {
      //toon info in Chrome Console
      console.log(this);
      console.log(this.innerHTML);
    });
  }
};
```

```
//#region ***  INIT / DOMContentLoaded  ***
const init = function() {

  html_destinationHolder = document.querySelector(".js-destinations");
  html_routeHolder = document.querySelector(".js-trajects");
  html_selectedCity = document.querySelector(".js-departure");
  html_destinationSelect = document.querySelector(".js-destination");
  html_adaptTrain = document.querySelector(".js-adapttrain");

  if (html_destinationHolder) {
    getDestinations();
  }
};

document.addEventListener("DOMContentLoaded", init);
```

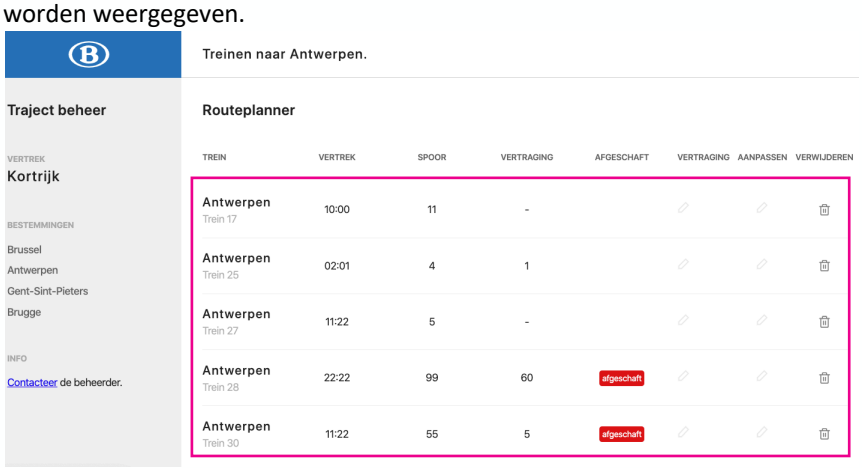
- **Bij het <u>klikken</u> op een navigatie (bestemming) element**

    - Zorg dat je aan de rechterzijde de weergave kan aanpassen na het klikken op een bestemming, zodat we kunnen zien wanneer de treinen naar de opgevraagde locatie zullen rijden. Gebruik hiervoor het `data-destination-id` attribute van de navigatie.
        - Pas de **titel** `.js-departure` aan naar de gekozen locatie (gebruik hiervoor de `innerHTML` van het aangeklikte navigatie element.



        - **Bewaar** ook het gekozen `data-destination-id` attribuut van het aangeklikte navigatie element in een globale **javascript variabele** `currentDestinationID`. Op deze manier weet je webpagina steeds welke bestemming er momenteel gekozen is.
        - Haal de data (de treinen die naar het bestemmingID rijden in de variabele `currentDestinationID`) op met een `get_____()` en bijhorende API call.
        - Zet de jsondata die binnenkomt in de callback om naar HTML met `show____()`

        - Bekijk goed onderstaande screenshot om te zien welke gegevens moeten worden weergegeven.

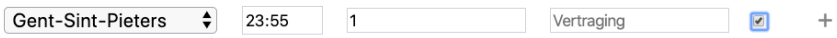


        De bijhorende html code voor zo 1 lijntje zal er als volgt uitzien.
        Schenk aandacht aan dat
        - De vertragingen doorlinkt naar een webpagina vertraging.html?TreinID=x
          ➔Werk dit volgende week verder uit
        - De update doorlinkt naar een webpagina aanpassen.html?TreinID=x
          ➔Werk dit volgende week verder uit
        - De delete “knop” een `data-` attribute heeft.

```
<div class="c-traject">
▼<div class="c-traject__info">
    <h2 class="c-traject__name">Antwerpen</h2>
    <p class="c-traject__train-id">Trein 28</p>
  </div>
  <div class="c-traject__departure">
                    22:22
                </div>
  <div class="c-traject__track">
                    99
                </div>
  <div class="c-traject__delay">
                60
                </div>
▼<div class="c-traject__cancelled">
    <span class="c-traject__cancelled-label">afgeschaft</span>
  </div>
▼<div class="c-traject__updatevertraging">
  ▼<a href="vertraging.html?TreinID=28">
    ▶<svg class="c-traject__updatevertraging-symbol" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="none" stroke="#222222" stroke-
    width="2" stroke-linecap="round" stroke-linejoin="arcs">…</svg>
    </a>
  </div>
▼<div class="c-traject__update">
  ▼<a href="aanpassen.html?TreinID=28">
    ▶<svg class="c-traject__update-symbol" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="none" stroke="#222222" stroke-width="2"
    stroke-linecap="round" stroke-linejoin="arcs">…</svg>
    </a>
  </div>
▼<div class="c-traject__delete">
  ▶<svg class="c-traject__delete-symbol" data-train-id="28" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="none" stroke="#222222"
  stroke-width="2" stroke-linecap="round" stroke-linejoin="arcs">…</svg>
  </div>
</div>
```

- Zorg dat je treinen kan verwijderen bij het klikken op het vuilnisbakje. Gebruik hiervoor een dynamische `Eventlistener` en `listenTo____()`
    - Handel de delete af binnenin de Eventlistener. Roep de API aan om een trein te verwijderen (welke request method heb je hiervoor nodig).
    - Haal na het deleten (een succes volle callback) de lijst met treinen naar de huidige bestemming opnieuw op. Denk even goed na waar je de huidige gekozen bestemming kan ophalen.

- Zorg dat je een nieuwe trein kan toevoegen onderaan de pagina bij het klikken op het plusteken. Gebruik hiervoor een dynamische `Eventlistener` en `listenTo____()`
    - Handel het toevoegen af binnenin de Eventlistener. Kies de correcte request method en geef een correcte payload mee aan de API call.
    - Haal na het toevoegen de lijst met treinen op naar de bestemming die in de keuzelijst actief stond (dus van de trein die je net toevoegde).
    - Het is voor deze oefening niet nodig om datavalidatie te doen. Vul bij het testen dus altijd alle velden correct in.



- **Controleer jezelf**

Als je bovenstaande stappen goed geprogrammeerd hebt, zal app.js volgende structuur hebben.
Kopieer deze niet onmiddellijk maar probeer deze eerst zelf op te bouwen! Dit is de basisstructuur, je zal natuurlijk nog extra zaken moeten programmeren.

De code om de navigatie zichtbaar te maken is voor de eenvoud weggelaten.

```
let currentDestinationID; // is geen DOM reference maar globale variabele

//#region ***  DOM references ***
let html_destinationHolder, html_routeHolder, html_selectedCity,
html_destinationSelect, html_adaptTrain;
```

```
//#region ***  Callback-Visualisation - show___ ***
const showDestinations = function(jsonObject) {
  //Toon menu

  //Toon dropdownbox

  listenToClickDestination();
};
```

```
const showTrainsOnDestinations = function(jsonObject) {
  if (jsonObject.length === 0) {
    html_routeHolder.innerHTML = 'Geen treinen.';
    return;
  }
  // Vul de innerHTML op met alle treinen
  listenToClickRemoveTrain();
};
```

```
//#region ***  Callback-No Visualisation - callback___  ***
const callbackAddTrain = function(data) {
  console.log('ADD antw van server ');
  if (data.treinid > 0) {
    console.log('ADD gelukt');
  }
};
const callbackRemoveTrain = function(data) {
  console.log(REMOVE antw van server ');
};
```

```
//#region ***  Data Access - get___ ***
const getTrainsOnDestinations = function(idDestination) {
  handleData(`http://127.0.0.1:5000/api/v1/treinen/bestemming/${idDestination}`,
showTrainsOnDestinations);
};
```

```
//#region ***  Event Listeners - listenTo___ ***
const listenToClickDestination = function() {
  const buttons = document.querySelectorAll('.js-station');
  for (const btn of buttons) {
    btn.addEventListener('click', function() {
      const id = this.getAttribute('data-destination-id');
      currentDestinationID = id;
      getTrainsOnDestinations(id);
    });
  }
};
```

```
const listenToClickAddTrain = function() {
  const button = document.querySelector('.js-add-train');
  button.addEventListener('click', function() {
    console.log('toevoegen nieuwe trein');
    const jsonobject = {……}; // vul verder aan met payload
    console.log(jsonobject);
    handleData('http://127.0.0.1:5000/api/v1/treinen', callbackAddTrain, null,
'POST', JSON.stringify(jsonobject));
  });
};
```

```
const listenToClickRemoveTrain = function() {
  const buttons = document.querySelectorAll('.c-traject__delete-symbol');
  for (const b of buttons) {
    b.addEventListener('click', function() {
      const id = this.getAttribute('data-train-id');
      console.log('verwijder ' + id);
      handleData(`http://127.0.0.1:5000/api/v1/treinen/${id}`, callbackRemoveTrain,
null, 'DELETE');
    });
  }
};
```

howest
hogeschool

howest.be