howest
hogeschool

Recap – part 2 + Leaflet

Frontend en backend

Inhoudsopgave

1 Leerdoelen

- In de **frontend** een read gebruiken.
- In de **frontend** een create doorsturen via een form en POST method.
- In de **frontend** een update gebruiksvriendelijke weergeven en doorsturen via de fetch.
- In de **frontend** een delete doorsturen via de fetch.
- In de **frontend** gebruik maken van een javascript library om geodata op een kaart te tonen.

2 Vooraf

Het labo van deze week bestaat uit twee delen.

- We breiden het labo van vorige week uit met een webpagina om een update uit te voeren. Je bekijkt de oplossing van vorige week van de delete en insert.
- Nadien leer je werken met Leaflet. Dit is – net zoals Google Maps – een library om kaarten van opensource aanbieders op je website te tonen. Hiervan vind je videomateriaal terug op Leho.
- *Volgende week pas je leaflet toe op de NMBS-oefening.*

> 👨‍🏫 Leertip:
>
> In het eerste deel van het labo is veel voorgetypt voor jou. Typ niet alles “klakkeloos” over, maar probeer – net zoals in een live labo – alles stap voor stap op te bouwen en probeer voor jezelf te begrijpen waarom je deze zaken doet.

3 CRUD acties

Vorige labo heb je reeds

- De code geschreven om een trein te verwijderen via het click event op het vuilnisbakje. (DELETE)
- De code geschreven om een trein toe te voegen. (CREATE)

Bekijk deze code in het startbestand dat je vandaag krijgt, vergelijk ze met de code die jij had geschreven en pas aan waar nodig.

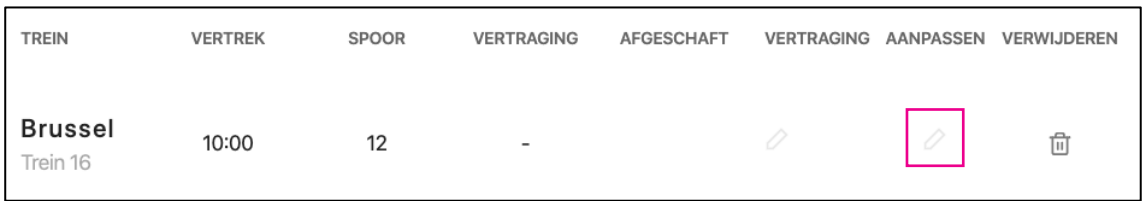
3.1 Update van een volledige record – een rit aanpassen

Je werkt de functionaliteit van je webapplicatie verder af met het aanpassen van een trein (UPDATE)

Als je een record (trein) wil updaten in de database moet je UX-gewijs aan enkele zaken denken.

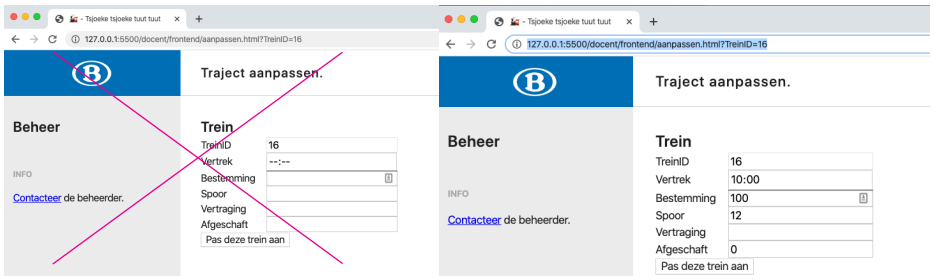
3.1.1 Voordat je kan 'Updaten' moet je eerst 'Readen'

- Als de gebruiker op de link klikt om een trein te wijzigen, wordt hij doorverwezen naar een nieuwe webpagina `http://127.0.0.1:5500/frontend/aanpassen.html?TreinID=16`.
  Via de querystring wordt meegegeven welke trein er moet worden aangepast.
  We kiezen hier niet toevallig om de Primary Key (PK) mee te geven. Via de PK kan elke record (trein) uniek worden aangesproken.



- We willen dezelfde javascript file (app.js) gebruiken om deze pagina dynamisch te maken. Daarom doen we in DOMContentLoaded een extra controle. We kijken vanaf welke pagina het javascript wordt geladen.
  - Via de `querySelector` zoeken of een `.js-` klasse aanwezig is op de pagina en bewaren het resultaat in een variabele.
    - Via een `if`-structuur controleren we of de variabel niet `undefined` is. Als de variabele is opgevuld, dan weten we dat het element met klasse `.js-` op de webpagina bestaat.
      En hebben we dus de correcte pagina bepaald.

- In aanpassen.html, komt er een formulier tevoorschijn om de aanpassingen van een treinrit door te geven.
    - Hier is het gebruiksvriendelijk om de waardes die in de database aanwezig zijn reeds in te vullen in de verschillende tekstvakken. ✅



- Omdat je een bestaande treinrit wil updaten, vul je dus in de elementen op het formulier de huidige `values` van de gekozen *treinrit* in.
    - Hiervoor zal je eerst moeten weten van welke treinrit je gegevens wil tonen. Hiervoor gebruik je het TreinID (*PK*) van een trein. Deze waarde is beschikbaar in de *querystring*.

    - In JS kan je de querystring aanspreken met URLSearchParams, zoals in de theorieles gezien.

```
let urlParams = new URLSearchParams(window.location.search);
let treinid = urlParams.get("TreinID");
```

    - Als de key `TreinID` bestaat *(niet undefined is)* in de querystring dan roep je via `handleData()`de gegevens op van deze ene bepaalde trein.
        - Indien de key in de querystring niet bestaat dan verwijs je de bezoeker terug naar index.html

    - Bij een antwoord van de server toon je de binnengekomen gegevens in het formulier via de functie `showTrain()`
        - Het id van de trein plaats je in een tekstvak. Dit tekstvak stel je via het attribuut `readonly` in, zodat de waarde niet aangepast kan worden.
          Een PK wil je natuurlijk niet laten aanpassen door een bezoeker, maar moeten we straks wel doorsturen met het PUT request naar de API.
        - Vervolgens vul je het `value` attribute van alle form elementen op met de gegevens uit het jsonobject.

https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams
https://developer.mozilla.org/en-US/docs/Web/API/Window/location

app.js

```
//#region *** Callback-Visualisation - show___ ***
const showTrain = function(jsonObject) {
  console.log(jsonObject);
  document.querySelector("#idtrein").value      = jsonObject.idtrein;
  document.querySelector("#afgeschaft").value   = jsonObject.afgeschaft;
  document.querySelector("#bestemmingID").value = jsonObject.bestemmingID;
  document.querySelector("#spoor").value        = jsonObject.spoor;
  document.querySelector("#vertraging").value   = jsonObject.vertraging;
  document.querySelector("#vertrek").value      = jsonObject.vertrek;
};

//#region *** Data Access - get___ ***
const getTrain = function(treinid) {
  handleData(`http://127.0.0.1:5000/api/v1/treinen/${treinid}`, showTrain);
};

//#region *** INIT / DOMContentLoaded ***
const init = function() {
  console.log("🚂", "https://www.youtube.com/watch?v=8oVTXSntnA0");
  // Get some DOM, we created empty earlier.
  html_destinationHolder = document.querySelector(".js-destinations");
  html_adaptTrain = document.querySelector(".js-adapttrain");

  if (html_destinationHolder) {
    //deze code wordt gestart vanaf index.html
    getDestinations();
    listenToClickAddTrain();
  }
  if (html_adaptTrain) {
    //deze code wordt enkel gestart vanaf aanpassen.html
    let urlParams = new URLSearchParams(window.location.search);
    let treinid = urlParams.get("TreinID");
    if (treinid) {
      getTrain(treinid);
    } else {
      window.location.href = "index.html";
    }
  }
```

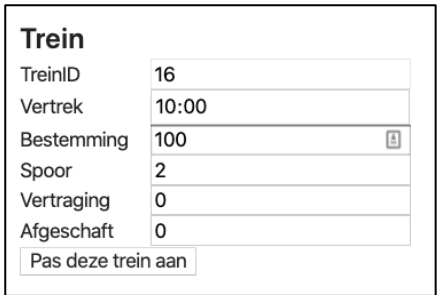
3.1.2 Na het 'Readen' kan je 'Updaten'

Het updaten volgt dezelfde werkwijze als de voorbije labo's:

1. Je koppelt een `click` event aan een knop
2. Je maakt een jsonobject aan om mee te geven met de payload van de fetch call.

3. Je spreekt het PUT endpoint aan van de API.
   (In het adres van het endpoint noteer je het id van de trein die je wil aanpassen)
4. Je schrijft de callbackfuncties zodat de bezoeker terug naar index.html gestuurd wordt, bij een geslaagde update.

Voor de eenvoud van de oefening, vul bij vertraging altijd een waarde in.
(Ook al is de vertraging 0 minuten) op deze manier krijg je geen errors, en hoef je geen extra controles te programmeren.

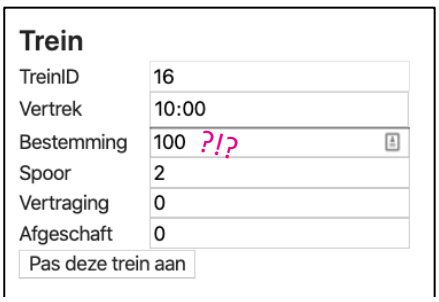



```
//#region ***  Event Listeners - listenTo___ ***
const listenToClickAdaptTrain = function() {
  let treinid = document.querySelector("#idtrein").value;
  const jsonobject = {
    idtrein:      treinid,
    afgeschaft:   document.querySelector("#afgeschaft").value,
    bestemmingID: document.querySelector("#bestemmingID").value,
    spoor:        document.querySelector("#spoor").value,
    vertraging:   document.querySelector("#vertraging").value,
    vertrek:      document.querySelector("#vertrek").value
  };
  console.log(jsonobject);
  handleData(`http://127.0.0.1:5000/api/v1/treinen/${treinid}`, callbackAdaptTrain,
  callbackAdaptTrainError, "PUT", JSON.stringify(jsonobject));
};
```

```
//#region ***  INIT / DOMContentLoaded  ***
const init = function() {
  console.log("🚂", "https://www.youtube.com/watch?v=8oVTXSntnA0");
  // Get some DOM, we created empty earlier.
  html_destinationHolder = document.querySelector(".js-destinations");
  html_adaptTrain = document.querySelector(".js-adapttrain");

  if (html_destinationHolder) {…
  }
  if (html_adaptTrain) {
    //deze code wordt enkel gestart vanaf aanpassen.html
    let urlParams = new URLSearchParams(window.location.search);
    let treinid = urlParams.get("TreinID");
    if (treinid) {
      getTrain(treinid);
      document.querySelector(".js-adapttrain-btn").addEventListener("click", function() {
        listenToClickAdaptTrain();
      });
    } else {
      window.location.href = "index.html";
    }
  }
```

3.1.3 Maak je formulier nog gebruikersvriendelijk

Het zal je al opgevallen zijn dat je momenteel het *id* van de bestemming uit je hoofd moet kennen. Dit is in strijd met alle regels van UX.



We kunnen het tekstvak voor het *bestemmingsid* vervangen met een keuzelijst en deze opvullen bij het laden van de pagina met alle bestemmingen.

- We tonen de bestemming als `innerHTML` en als `value` kiezen we voor het *bestemmingsid*.
  Het is nog steeds nodig om als value het bestemmingsid te blijven gebruiken: omdat dit de waarde is die wordt weggeschreven in de database als Foreign Key in de tabel treinen.

Vergeet je select-tag geen `name` en `id`-attribute te geven. Het `class`-attribute heb je nodig om de control aan te spreken in JS.

Het is belangrijk dat je in de DOMContentLoaded eerst alle destinations opvraagt en toont op het formulier en pas nadien de trein opvraagt en zijn values toont op de elementen van het formulier.

Omdat onze fetch-calls asynchroon werken, zijn we enkel op het einde van de functie `showDestionationsForUpdate()` zeker dat de keuzelijst is opgevuld.
Daarom verplaatsen we de oproep van de functie `showTrain()` naar het einde van de functie `showDestinationsForUpdate()`.

Want: voordat je in `showTrain()`, de value van #bestemmingID kan instellen, moet de keuzelijst natuurlijk eerst opgevuld zijn met option elementen!

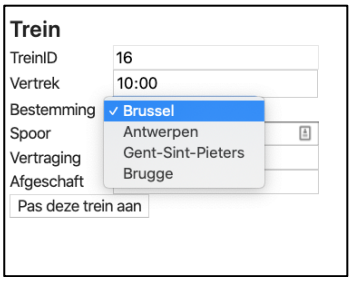
aanpassen.html?TreinId=x

```html
<div class="c-form__item">
  <label for="bestemmingID">Bestemming</label>
  <!-- <input type="text" name="bestemmingID" id="bestemmingID" /> -->
  <select name="bestemmingID" id="bestemmingID" class="js-destionations-for-update"></select>
</div>
```

app.js

```
//#region ***  Callback-Visualisation - show___ ***
const showDestinationsForUpdate = function(jsonObject) {
  //Toon dropdownbox
  console.log("****");
  console.log(jsonObject);
  let htmlstring_options = "";

  for (const bestemming of jsonObject.bestemmingen) {
    htmlstring_options += `<option value="${bestemming.idbestemming}">${bestemming.stad}</option>`;
  }
  document.querySelector(".js-destionations-for-update").innerHTML = htmlstring_options;

  //haal trein op
  let urlParams = new URLSearchParams(window.location.search);
  let treinid = urlParams.get("TreinID");
  getTrain(treinid);
};

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

//#region ***  INIT / DOMContentLoaded  ***
  if (html_adaptTrain) {
    //deze code wordt enkel gestart vanaf aanpassen.html
    let urlParams = new URLSearchParams(window.location.search);
    let treinid = urlParams.get("TreinID");
    if (treinid) {
      getDestinationsForUpdate();
      // getTrain(treinid);

      document.querySelector(".js-adapttrain-btn").addEventListener("click", function() {
        listenToClickAdaptTrain();
      });
    } else {
      window.location.href = "index.html";
    }
  }
```

Resultaat



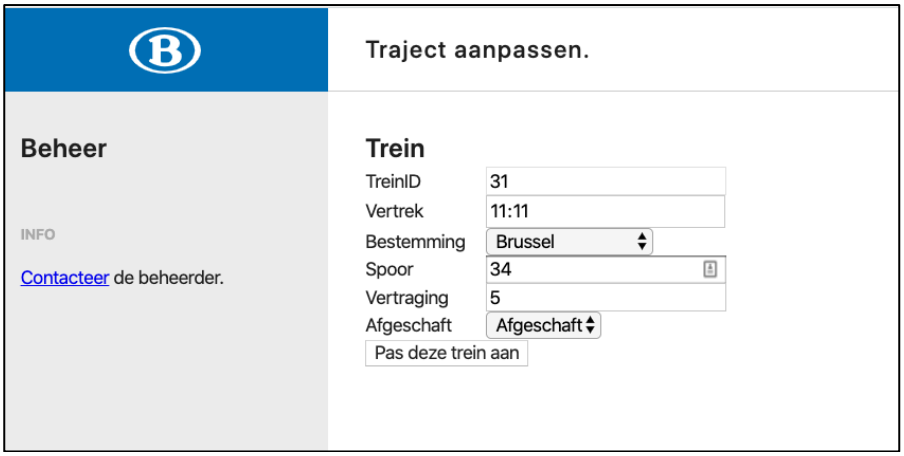

```
<select name="bestemmingID" id="bestemmingID" class="js-destionations-for-update">
    <option value="100">Brussel</option>
    <option value="101">Antwerpen</option>
    <option value="102">Gent-Sint-Pieters</option>
    <option value="103">Brugge</option>
  </select>
```

Ook met het tekstvak voor een afgeschafte treinrit kunnen we iets gelijksaardigs doen. Nu mogen we in aanpassen.html de keuzelijst wel hard coderen, want er is geen relatie naar een andere tabel in database die verwijst naar deze waarde (zoals die er wel was in het vorige voorbeeld met bestemmingen).

We geven, de `option`-tags de values 0 en 1. (0 =rijdt, 1 = afgeschaft). `ShowTrains()` zal er straks voor zorgen dat de juiste value wordt geselecteerd uit deze keuzelijst. In `app.js` hoeven we dus niets te veranderen.

aanpassen.html?TreinID=x

```
<div class="c-form__item">
  <label for="afgeschaft">Afgeschaft</label>
  <!-- <input type="text" name="afgeschaft" id="afgeschaft" /> -->
  <select name="afgeschaft" id="afgeschaft">
    <option value="0">Rijdt</option>
    <option value="1">Afgeschaft</option>
  </select>
</div>
```



3.2 Updaten van één veld uit een record - een vertraging toevoegen

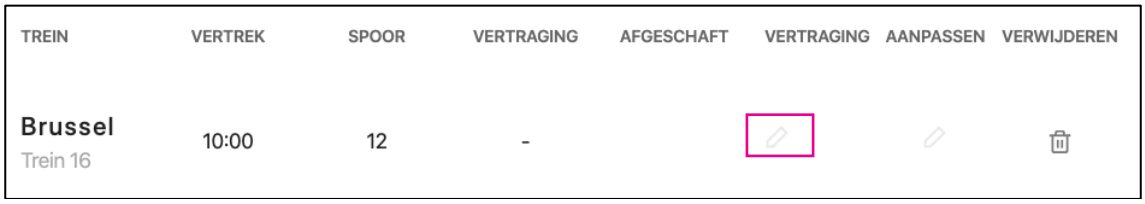
Een update (PUT) naar een database sturen moet - zoals in het vorige voorbeeld - niet altijd voor het volledige record.
Soms wil je maar 1 veld aanpassen. (Bijvoorbeeld een bericht 'liken', je stem uitbrengen op een poll, een bericht gelezen plaatsen, een vertraging van een treinrit toevoegen, …).

In zo'n situatie schrijf je, in de backend, een route en functie die niet de volledige record update.
Een voorbeeld hiervan is de route `/api/v1/treinen/<trein_id>/vertraging` die je in het vorige labo schreef. De route gebruikt enkel het `trein_id` uit het endpoint en de `vertraging` in de payload om een record te updaten.

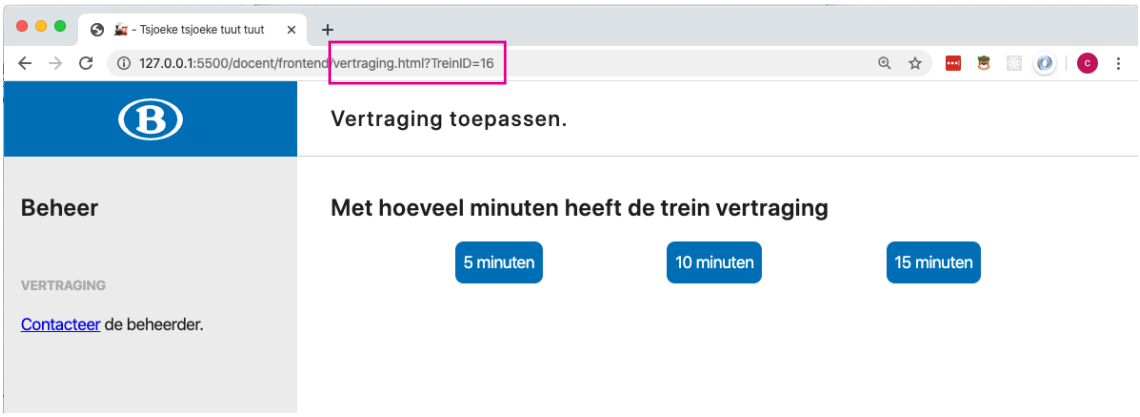
backend/app.py

```python
@app.route(endpoint + '/treinen/<trein_id>/vertraging', methods=['PUT'])
def set_trein_vertraging(trein_id):
    if request.method == 'PUT':
        gegevens = DataRepository.json_or_formdata(request)
        data = DataRepository.update_trein_vertraging(
            trein_id, gegevens['vertraging'])
        return jsonify(trein_id=trein_id), 200
```

index.html



vertraging.html?TreinID=x



Deze html pagina bereik je via index.html en bevat

- Een div met de `js-delaytrain` CSS-klasse. Op deze CSS-klasse kan je straks controleren in de `DOMContentLoaded`.
- Een unordered list die via CSS zodanig opgemaakt is dat het *visueel knoppen* zijn die naast elkaar staan. Elke listitem (knop) heeft dezelfde CSS klasse `js-delay-option` en een `data-delay` attribuut.
- Zoals je kan zien is alles "hard coded"

```
▼<div class="js-delaytrain"> == $0
  ▼<ul class="c-add-delay">
     <li class="c-add-delay__option js-delay-option" data-delay="5">5 minuten</li>
     <li class="c-add-delay__option js-delay-option" data-delay="10">10 minuten</li>
     <li class="c-add-delay__option js-delay-option" data-delay="15">15 minuten</li>
   </ul>
  </div>
```

app.js

- Bij het laden - `DOMContentLoaded`:
  - Kijk je of de klasse `js-delaytrain` aanwezig is.
  - Koppel je een click-event aan alle elementen met de klasse `js-delay-option`.
    - Denk goed na hoe je dit gaat doen? Kijk eens naar de vorige voorbeelden!
- In het `click`-event
  - Roepen we de functie `listenToClickDelay(this)` op. Als parameter geef je `this` mee. Zodat je deze straks kan gebruiken binnen in de functie om het data- attribuut op te halen.
- In `listenToClickDelay(btn)`
  - Komt er een parameter binnen – de `this` die je zojuist meegaf bij de oproep - die de knop voorstelt, we noemen de parameter dan ook `btn`. Zoals je kan zien stelt deze parameter het volledig element voor. Via `getAttribute()` kan je eventueel het `data-` attribuut opvragen.

```
const listenToClickDelay = function(btn) {
  console.log('geklikt op delay');
  console.log(btn);
  console.log(btn.getAttribute('data-delay'));
};
```

```
<li class="c-add-delay__option js-delay-option" data-delay="15">15 minuten</li>
15
>
```

  - Haal je uit de querystring, het treinID
  - Roep je via `handleData()` de API aan.
    - Je stelt de URL van het endpoint juist in (voeg het correcte id van de aan te passen treinrit toe in het endpoint)

- Haal je uit het element dat is aangeklikt - de parameter btn van de functie - de waarde die hoort bij `data-delay`. Je bewaart deze waarde in de variabele `delay`
- Maak een payload klaar met de variabele `delay` die er als volgt uitziet

```
const jsonobject = {
    vertraging: delay
  };
```

- Verstuurt deze payload naar het juiste endpoint.
- Komt er een geldige response terug van de API dan redirect je de gebruiker terug naar index.html

4 Leaflet

Zie voor de oefeningen op Leaflet de documentatie op https://leafletjs.com/reference-1.6.0.html en de videos die aangeboden worden op Leho. Werk de oefeningen uit naar het voorbeeld in de video. Volgende week zullen we een gelijkaardige oefening maken op de treinoefening.

howest
hogeschool

howest.be