
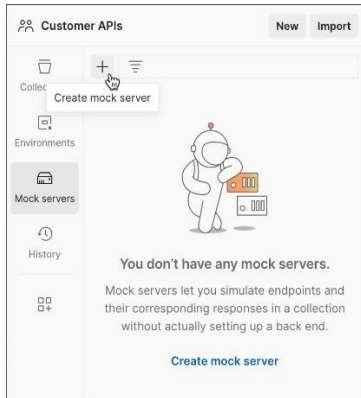


Hands-on 1: Introductie (à 10 min.)

Een simpele mock server in Postman.

Een nieuwe mock server aanmaken in Postman

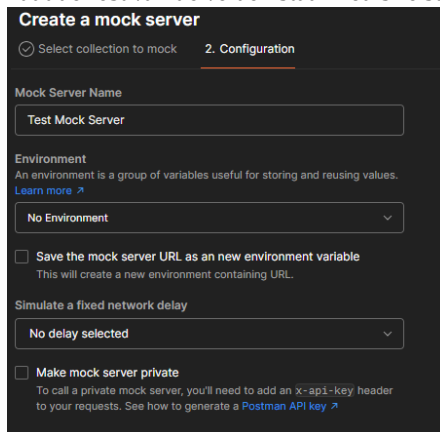
1. Open Postman op delagroup.postman.co en login met je Dela Entra ID
2. Ga naar Workspaces en kies My Workspace
3. Ga naar mock servers
 - a. Als deze niet te zien is, ga dan eerst naar  om de workspaces sidebar te configureren.
4. Klik op +, hiermee begin je met het maken van een nieuwe mock server



5. Selecteer "Create a new collection".
6. Geef de mock server een naam, bijvoorbeeld "Test Mock Server".
7. Voeg een **GET** request method toe, waarbij het pad van de Request URL **users** is.
8. Voeg de volgende response body toe

```
[
  { "id": 1, "name": "Alice" },
  { "id": 2, "name": "Bob" }
]
```

9. Laat de rest van de velden staan zoals ze staan.



10. Klik op "Create Mock Server".

Een request naar de mock server sturen

1. Open de zojuist gemaakte mock server in Postman, en kopieer de gegenereerde URL (onderaan in blauw, of open de collectie).
2. Ga naar het tabblad "Collections" in Postman. Zie je nu je nieuw gemaakte mock server?
3. Als het goed is, zie je nu binnen deze collectie de **GET** met daarachter **users**, de call die je net hebt aangemaakt. Klik deze aan.
4. In de balk zie je nu **GET {url}/users**. Klik op send.
5. Wat krijg je terug? Controleer of de response de body teruggeeft die we verwachten.
6. Probeer ook wat er gebeurt wanneer je **{url}/users** in je browser plakt.
7. Probeer ook wat er gebeurt wanneer je de request vanuit een andere pc stuurt. Draait de mock server lokaal?

Hands-on 2: Ontwikkelen (à 20 min)

Frontend ontwikkelen met behulp van een mock server.

Als developer kan het voorkomen dat je wil starten met het ontwikkelen van de frontend van een applicatie, maar dat dit nog niet mogelijk is omdat de backend, waarvan de frontend afhankelijk is, nog niet beschikbaar is. Dit kan bijvoorbeeld zo zijn omdat ook de backend nog in de beginfase van ontwikkeling zit. Wanneer het API contract bekend is, kan een oplossing voor de ontwikkeling van de frontend zijn om de backend te mocken. Dat gaan we in deze hands-on nabootsen.

Vorbereiding

Benodigheden: Node.js (en npm), IDE (bijv. VS Code)

1. Clone de frontend repository van <https://github.com/VinceVerspeek/product-list-app-handson.git>
2. Open de repository in je IDE
3. Open een terminal in de project folder, en run het commando `npm install`.
4. Run nu het commando `npm start`. Wat gebeurt er nu? Kijk ook in je browser console. Gaat alles goed?

Mock server implementatie

Voor het opzetten van de mock server maken we gebruik van `json-server`. `json-server` is een tool waarmee je snel een volledige REST API kunt simuleren op basis van een eenvoudig JSON-bestand. Het ondersteunt alle CRUD-operaties (GET, POST, PUT, DELETE) zonder dat je zelf een backend hoeft te ontwikkelen. Hiermee kun je frontend-ontwikkeling voortzetten terwijl de echte backend nog niet klaar is.

1. Installeer `json-server`, door het volgende commando in de project folder te draaien vanuit de terminal:

```
npm install json-server
```

2. Maak een file genaamd `db.json` aan in de root folder van het project. Deze file representeert de database van onze gemockte backend. Vul de json file met de volgende content:

```
{
  "products": [
    {
      "id": "1",
      "name": "Een kilo kaas",
      "description": "We doen de boodschappen"
    },
    {
      "id": "2",
      "name": "Een liter melk",
      "description": "Elke dag weer naar de supermarkt"
    }
  ]
}
```

- Voeg zelf nog een product toe met id "3", naam "Een blikje soep", en description "Je weet dat ik niet kiezen kan".

3. Start de mock server.

```
npx json-server --watch db.json --port 3001
```

4. Bekijk de output van het vorige commando. Waar draait de server? Gebruik deze info om de juiste `API_URL` aan te geven in `src/api/productApi.js`
5. Indien je deze had afgesloten, run opnieuw `npm start`. Werkt het nu wel?
6. De knoppen `edit` en `delete` werken nog niet. Probeer in `productApi.js` de functies `updateProduct` en `deleteProduct` te implementeren:
 - `updateProduct`:
 - Welke HTTP methode heb je nodig voor updaten?
 1. *Hint: je gebruikt hier json-server, zoek daar in de documentatie*
 - De functie moet het `productId` van het te updaten product krijgen, en een object dat de `updatedProduct` data bevat. Zorg ervoor dat je met headers duidelijk maakt aan de server dat de gestuurde data in JSON formaat is.
 - De functie hoeft geen waarde te retourneren (waarom niet?).
 - `deleteProduct`:
 - Welke HTTP methode heb je nodig voor verwijderen?
 - De functie moet het `productId` van het te updaten product krijgen.

Reflectie

Denk erover na wat voordelen zouden kunnen zijn van het gebruiken van een mock server ten opzichte van een echte backend bij het ontwikkelen van frontend.

Hands-on 3: Testen (à 20 min)

Testen met behulp van een mock server

Vorbereiding

1. Clone de repository van <https://github.com/VinceVerspeek/todo-list-testing-handson.git>
2. Open de repository in je IDE
3. Open een terminal in de project folder, en run het commando `npm install`.
4. Run het commando `npx playwright install`.
5. Run nu het commando `npm start`. Wat heeft de gemaakte website voor doel?

Testen met de echte API

1. Kijk in `./test/todos.spec.js` welke test al bestaat voor de website. Wat checkt deze test?
2. Open een nieuwe terminal, en draai met het volgende commando deze specifieke test in het project.

```
npx playwright test -g "fetches and displays todos from real API"
```

Of gebruik onderstaand commando voor alle tests in dit project (nu alleen nog deze)

```
npx playwright test
```

- Slaagt of faalt de test? Indien hij faalt, probeer de test dan te verbeteren en draai hem opnieuw.

Testen met een mock server

1. Installeer json-server, door het volgende commando in de project folder te draaien vanuit de terminal:

```
npm install json-server
```
2. Zoals in de vorige hands-on: maak een db.json die todo's zal bevatten aan in de root folder van het project. Voeg een enkele array toe met de naam `todos`. Geef de objecten de properties `id` en `title`. Maak een aantal voorbeeld objecten.
 - a. *Hint: je mag gerust spieken bij de vorige opdracht*
3. We willen dat wanneer we gaan testen met de mock server, de applicatie gebruikmaakt van de mock server data voor het draaien van de testen.
 - a. Ga naar `package.json`, en voeg in het `"scripts"` object onder de regel van `"start"` het volgende toe. Wat doet deze regel?

```
"start:mock": "cross-env REACT_APP_TODO_URL=http://localhost:3001/todos react-scripts start",
```
 - b. Ga naar `./src/components/TodosList.js` en kijk naar de regel waar `const todoUrl` gezet wordt. Momenteel is dit een vaste waarde, namelijk de echte API. We willen vanuit het npm command de url van de mock server kunnen accepteren. Pas deze variabele daartoe op de volgende manier aan:

```
const todoUrl = process.env.REACT_APP_TODO_URL || "https://jsonplaceholder.typicode.com/todos";
```
4. Ga naar `./test/todos.spec.js` en maak een nieuwe test description en daarin een test aan voor de mock server. Test hierbij net als bij de test met de echte API of de eerste todo is wat we verwachten.
 - a. Start de mock server met:

```
npx json-server --watch db.json --port 3001
```
 - b. Stop de draaiende applicatie (let op: dus niet de mock server uit de vorige stap!)
 - c. Start de applicatie opnieuw, maar nu met:

```
npm run start:mock
```
 - d. Draai de gemaakte nieuwe test met:

```
npx playwright test "naam van nieuwe test"
```
5. **Optioneel:**
Schrijf een mock server test die controleert of de todos array die opgehaald wordt van de mock server de juiste lengte heeft.
6. **Optioneel:**
Schrijf een mock server test die controleert of de mock todos in de juiste volgorde weergegeven worden.

Reflectie

Denk erover na wat voordelen zouden kunnen zijn van het gebruiken van een mock server ten opzichte van een echte backend bij het testen van software.