

# Labo Gebruikersinterfaces

## Reeks 4: Ajax

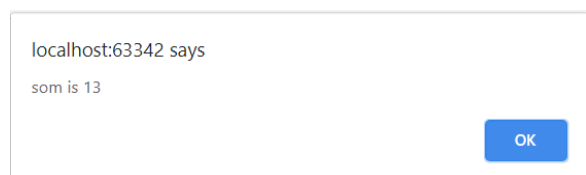
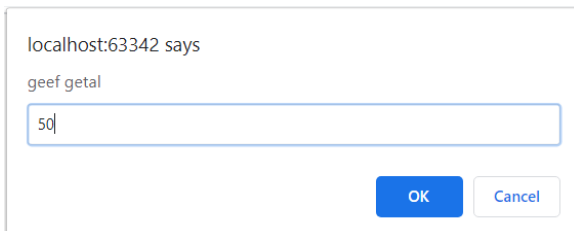
### 1 Voorbereidende oefening op promises

In dit labo werk je in IntelliJ IDEA, Ultimate edition. Ben je geregistreerd als student (met je ugent-mailadres) dan is deze versie gratis.

Als voorbereidende oefening op promises maak je een kleine applicatie, die via dialoogvensters de gebruiker om input vraagt. Maak een index-pagina zonder inhoud, leg de link naar een JavaScript-bestand, en schrijf daar code die aan de hand van promises volgende functionaliteit heeft:

- de gebruiker krijgt tot drie keer toe de vraag om een getal in te geven;
- als de gebruiker drie keer een getal opgeeft, krijgt hij nadien de som te zien;
- indien de gebruiker echter geen getal ingeeft of indien de som groter zou worden dan 100, dan breekt het programma af met een foutmelding.

Werk met twee functies/promises: de functie `leesGetal()` vraagt via een promptvenster een getal aan de gebruiker; de functie `telBij(getal)` telt een getal op bij de voorlopige som.



## 2 Inleiding hoofdopdracht

In de vorige labo's maakten we kennis met HTML5 en Javascript. De combinatie van beiden laat toe om complexe client-side webapplicaties te maken.

In deze reeks implementeren we een blog met een achterliggend Content Management System. De beheerder van de blog zal eenvoudig met een webformulier nieuwe artikels kunnen toevoegen aan de blog zonder dat er iets moet veranderen aan de broncode. Al deze functionaliteit wordt geïmplementeerd via een REST API. Vanuit Javascript kunnen we aan de hand van Ajax requests communiceren met de backend. Hiervoor gebruiken we de nieuwe “fetch” api.

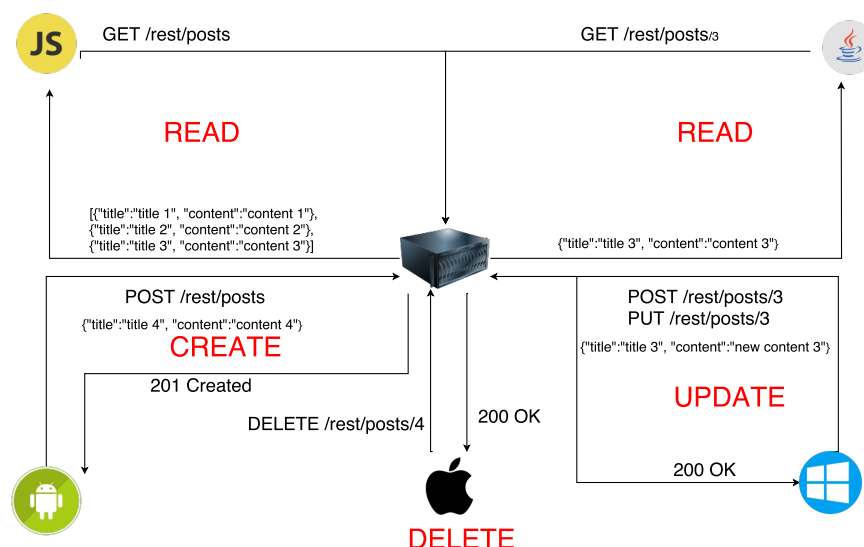
## 3 REST API

De back-end applicatie houdt de blog posts bij. De front-end applicatie kan requests sturen naar de back-end om een blog post op te vragen, aan te maken, aan te passen of te verwijderen. De back-end zal iedere request beantwoorden met een response. Deze communicatie tussen client en server verloopt via boven vernoemde REST API. Deze laatste is een populaire standaard bij de ontwikkeling van web applicaties.

Een request van de client naar de server bestaat steeds uit een aantal elementen:

- HTTP verb: welke operatie wil je uitvoeren? Er zijn een aantal mogelijkheden:
  - GET: verkrijgen van een resource (bv. blog posts opvragen)
  - POST: het aanmaken van een resource (bv. aanmaken van een nieuwe blog post)
  - PUT: een resource updaten (bv. aanpassen van een blog post)
  - DELETE: verwijderen van een resource (bv. een blog post verwijderen)
- Header: hier heeft de client de mogelijkheid om extra informatie over de request mee te sturen naar de server.
- Path naar de resource: welke data worden door de client opgevraagd of aangepast?
- Body (optioneel): data.

Onderstaande figuur toont hoe de REST API door verschillende clients kan gebruikt worden. De blog posts worden voorgesteld in JSON formaat.



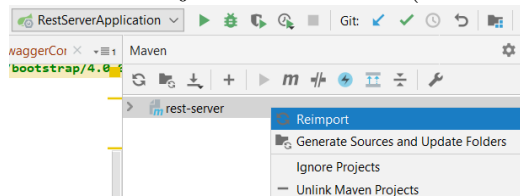
## 4 Starten van de back-end applicatie

Het bouwen van een back-end applicatie behoort niet tot de scope van deze cursus en daarom krijgen jullie van ons een werkende back-end applicatie. Op Ufora vinden jullie een IntelliJ project dat deze applicatie bevat. De back-end applicatie zal niet op een server runnen maar lokaal op jullie pc.

► Open het project in IntelliJ. Indien er foutmeldingen verschijnen zoals

```
Error(3, 52) java: package org.springframework.beans.factory.annotation does not exist
Error(4, 46) java: package org.springframework.context.annotation does not exist
Error(5, 46) java: package org.springframework.context.annotation does not exist
Error(6, 78) java: package org.springframework.security.config.annotation.authentication.builders does not exist
Error(7, 75) java: package org.springframework.security.config.annotation.method.configuration does not exist
Error(8, 67) java: package org.springframework.security.config.annotation.web.builders does not exist
Error(9, 67) java: package org.springframework.security.config.annotation.web.configuration does not exist
Error(10, 72) java: package org.springframework.security.config.annotation.web.configuration does not exist
```

dan herlaad je de rest-server (rechtermuisklik; dit duurt even):



Heb je later nog last met het starten van de server, sluit dan het project en heropen het.

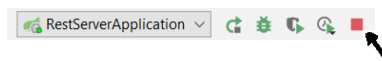
► Start de server  en bekijk de logs. Als alles goed ging, heeft onze applicatie een Tomcat Server op poort 8080 gestart. <http://localhost:8080>

Als je hier foutmeldingen krijgt, kijk dan op de laatste bladzijde van deze opgave voor tips.

Nog enkele tips voor we starten met coderen.

- Als je code aangepast hebt, maar bij controle van het resultaat op de website (localhost:8080) merk je geen verschil, dan zit de oude code misschien nog in de cache van de webbrowser. Om zeker te zijn dat je de nieuwe code runt, moet je vier stappen uitvoeren:

1. stop de server in IntelliJ

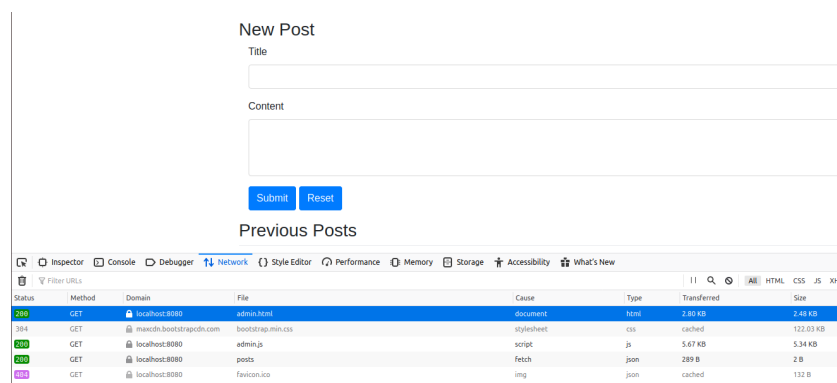


2. herlaad de webpagina in de browser met als resultaat  This site can't be reached

3. start de server in IntelliJ 

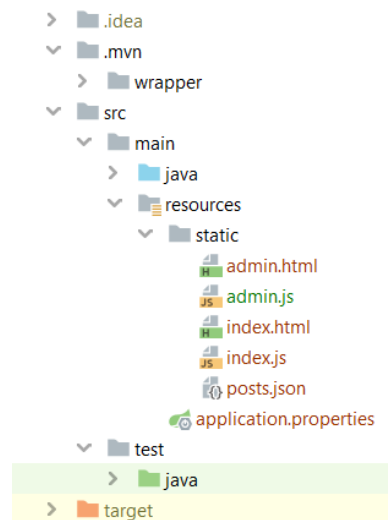
4. keer terug naar de webpagina

- Je kan de network monitor in de development tools gebruiken om te zien welke data doorgestuurd wordt. Hieronder een voorbeeld van Firefox.



## 5 Gegeven Html- en JavaScript-bestanden

- Als het project in IntelliJ geopend is, bekijk je links in het Project-venster de bestandsstructuur. We passen enkel de html- en js-bestanden aan. (Het bestand `admin.js` staat er misschien nog niet, dat kan je later aanmaken.)



## 6 Ophalen van blogposts met AJAX

Nu staat er één (TEST-)blogpost hardgecodeerd in de html-broncode. We willen de blogposts dynamisch ophalen via AJAX. Voorlopig laten we de gegeven back-end application nog even links liggen (we halen de data dus *nog niet* op vanop die back-end), en werken we met dummy data die in het bestand "`posts.json`" terug te vinden zijn.

- Werk in het bestand `index.js`. Schrijf de JavaScript-code die bij het laden van `index.html` een AJAX request doet naar dit bestand. Gebruik de developer tools om te debuggen. Maak gebruik van de "fetch" api. Omdat we de gegevens uit een bestand halen, geven we de bestandsnaam mee door aan de `fetch()`-methode: `fetch('posts.json')`.

Je kan handig gebruik maken van de ES6 template literals: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template\\_literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals). Merk op: deze literals worden geopend en afgesloten met een backward single quote.

- Verwijder die ene hardgecodeerde blogpost uit het Html-bestand en zorg ervoor dat de data uit de json getoond wordt.

Merk op, we werken niet met modules, dus vermeld bij de link naar het script expliciet dat de code pas geëvalueerd mag worden nadat de webpagina volledig geladen is. Enkel indien er effectief gebruik gemaakt wordt van modules, zal de extra vermelding `type="module"` ook impliciet de evaluatie uitstellen tot de webpagina geladen is.

Heb je (geneste) elementen aan het document toegevoegd, of Html-code? (De voorkeur gaat uit naar dat eerste.)

## 7 Toevoegen van blogposts met AJAX

Bekijk de administration-pagina. (Als je moet inloggen, gebruik je login en paswoord **admin**. Later zien we waar dit ingesteld werd.) Op deze pagina kan een administrator blogposts toevoegen. Na het ingeven van titel en inhoud, zorgt de knop **Submit** voor twee zaken:

1. de blogpost wordt gepost op de server (zodat hij later opgehaald kan worden in de index-pagina), en
2. de blogpost wordt onderaan op de admin-pagina toegevoegd, voorzien van delete- en edit-knoppen.

← → ↻ localhost:8080/admin.html

Home About Contact Administration

### A blog about Lorem Ipsum

added blogpost

#### New Post

Title

Derde blogpost

Content

Dit is de inhoud van de derde blogpost.

Submit Reset

#### Previous Posts

Title		
Eerste blogpost	delete	edit
Tweede blogpost	delete	edit

We pakken het coderen hiervan in stukjes aan. Werk in het bestand **admin.js** (maak dit aan als het er nog niet is, en voorzie de juiste link in **admin.html**).

♦ Het hoofdprogramma voorziet alvast volgende handelingen:

1. Beide alerts die nu nog verschijnen (de groene en rode banner) worden verborgen. Gebruik het **display**-stijlelement.
2. De submit-knop krijgt een nieuwe **EventListener**, zodat de standaardactie overschreven wordt. Wat de **EventListener** precies moet doen, wordt later beschreven - gebruik dus voorlopig een verwijzing naar een nog te implementeren functie (geef ze de naam **formSubmit**).

♦ Schrijf nu een paar hulpfuncties. Neem deze code over en vul aan:

```
let clearFields = () => {  
  };  
  
let success = (message) => {  
  };
```

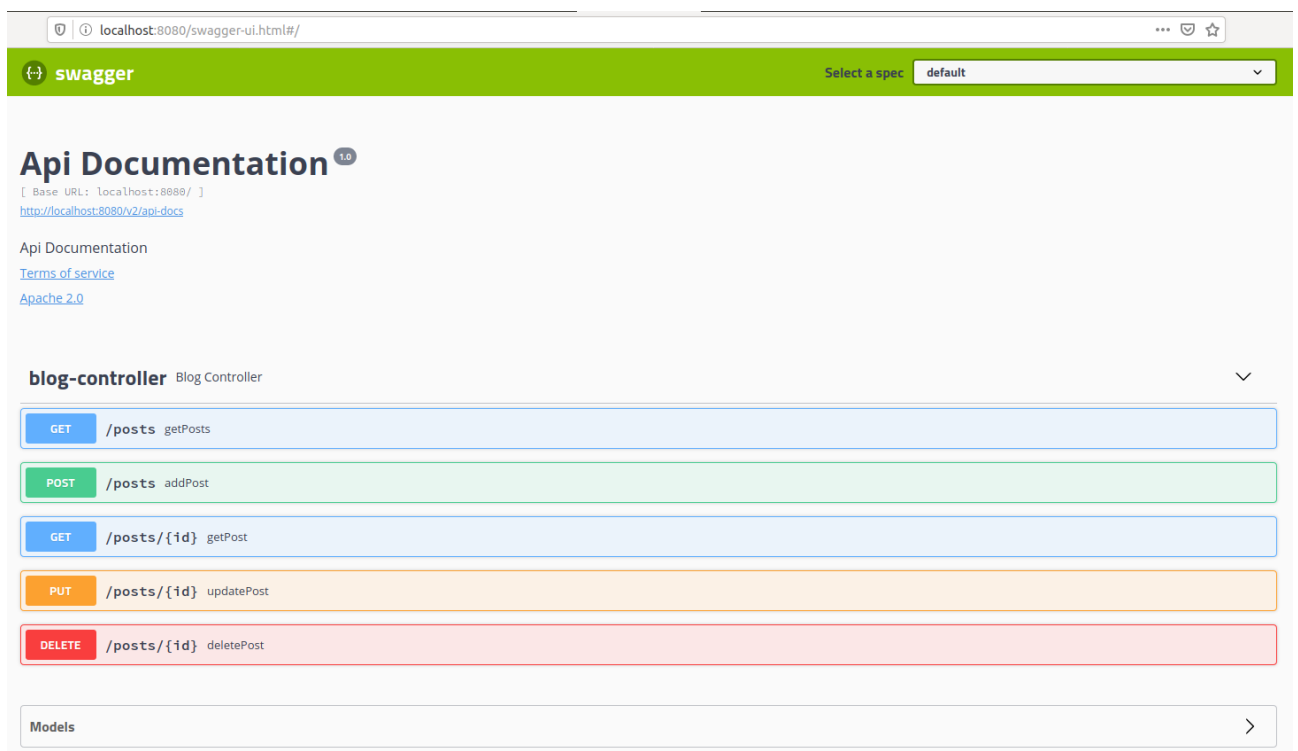
```
let fail = (message) => {
};

let refreshTable = () => {
};
```

1. De eerste functie maakt beide invulvelden (voor titel en inhoud) leeg.
2. De tweede functie zorgt ervoor dat de groene banner *succes alert* ingevuld wordt met de gegeven boodschap, en zichtbaar wordt op de webpagina. De rode banner wordt uiteraard onzichtbaar.
3. De derde functie is de tegenhanger van de vorige: enkel de rode banner wordt zichtbaar.
4. De laatste functie zal de tabel onderaan de admin-pagina (opnieuw) opstellen. Voorlopig *schetsen* we die functie, dat wil zeggen dat we ze nog niet de juiste functionaliteit geven maar enkel een voorlopige actie aan deze functie koppelen (bijvoorbeeld een boodschap loggen op de console via `console.log(...)`).

► Hier wordt beschreven wat de functie `formSubmit` moet doen.

1. Als het formulier ingediend wordt, wordt de gewone afhandeling van dit event verhinderd (`preventDefault`).



2. Onze back-end applicatie is er op voorzien dat we de API via swagger kunnen raadplegen. Via swagger kan je API resources raadplegen zonder daarvoor specifieke client-side code te schrijven. Een voorbeeld is te zien op bovenstaande figuur. Je kan zien welke data je moet meegeven en welke data je terugkrijgt wanneer de verschillende HTTP methodes gebruikt worden. Hier kan je ook het *path* naar de resources vinden. De Swagger UI is te vinden op: <http://localhost:8080/swagger-ui.html>.

3. Als het posten van de blog gelukt is, dan komt er nog een succesmelding (de groene banner verschijnt), de invulvelden worden leeggemaakt, en de tabel onderaan de admin-pagina wordt vernieuwd (ook al doet dit voorlopig niet veel).

4. Als het posten om een of andere reden niet gelukt is, dan verschijnt de rode banner.

♦ Je kan nu al uitproberen of het toevoegen van blogposts aan de server gelukt is. Daarvoor hergebruiken we de code die in het begin van het labo geschreven werd. In het bestand `index.js` werd het statische `json`-bestand geraadpleegd om informatie op te halen. Vervang de verwijzing naar dit bestand door de verwijzing `/posts`. Van zodra je een aantal blogposts hebt toegevoegd in de admin-webpagina, zouden ze nu moeten verschijnen op de startpagina van de website.

♦ Implementeer de methode `refreshTable`, die tot nu toe slechts geschetst werd. Daarvoor worden alle blogposts van de back-end application opgehaald. Van elke blogpost wordt de titel aan de tabel onderaan de `admin.html`-pagina toegevoegd, gevolgd door een delete- en edit-knop. Voorlopig moeten die knoppen nog niets doen.

Merk op: elke blogpost kreeg een titel en inhoud (`title` en `content`). Daarnaast voegt de back-end aan elke blogpost ook een uniek identificatienummer toe.

Willen we later een blogpost uit de tabel verwijderen of editeren, dan zou het makkelijk zijn indien de tabelrij ons zelf kan vertellen over welke blogpost in de back-end het gaat. Daarom voegen we aan de tag van de nieuwe tabelrij een nieuw attribuut toe, dat het uniek identificatienummer (`uuid`) van de blogpost bewaart. Om deze *custom data* aan de html-tag toe te voegen, gebruiken we het speciale `data-*`-attribuut. Meer informatie vind je op [https://www.w3schools.com/tags/att\\_global\\_data.asp](https://www.w3schools.com/tags/att_global_data.asp).

Een stukje van de nodige html-code:

```
<tr data-id=... >
  <td>...</td>
  <td><button class="btn_delete" onclick=...>delete</button></td>
  <td><button class="btn_edit" onclick=...>edit</button></td>
</tr>
```

## 8 Verwijderen en updaten van blogposts met AJAX

Tot nu toe hebben we aan de back-end data opgevraagd (GET) en data toegevoegd (POST). Nu willen we ook data aanpassen (PUT) en verwijderen (DELETE).

♦ Voeg de juiste eventListeners toe aan de delete- en edit-knoppen en implementeer deze.

Enkele tips.

1. Bij het verwijderen van een blogpost op een bepaalde rij, heb je het id nodig van deze blogpost. Dat id werd bewaard in de grootoudertag (`<tr data-id=...>`) van de delete-button-tag (`<button class="btn_delete">`). Gebruik de eigenschap `parentElement` van een DOM-element.
2. Je kan de waarde van het custom attribuut `data-id` opvragen aan de hand van zijn exacte naam. (Let op, die bevat een koppelteken!) Het kan ook anders: via de vraag *html custom-data how to find value* kom je op

<https://www.sitepoint.com/how-why-use-html5-custom-data-attributes/>, waar de eigenschap `dataset` besproken wordt. Gebruik de term `dataset` eventueel weer als zoekterm.

3. Bij het editeren van een blogpost wordt de juiste blogpost enkel opgehaald vanop de server, en titel en content ingevuld in de tekstvelden op de webpagina. De aanpassingen op de server worden pas gedaan bij het submitten van het formulier. Om daar een verschil te kunnen maken tussen bewaren van een nieuwe blogpost (POST) en updaten van een bestaande (PUT), heb je de globale variabele `saved_id` nodig. Als die `undefined` is, gaat het om een nieuwe blogpost. Is de gebruiker een blogpost aan het editeren, dan bevat deze variabele de id van deze blogpost.



## 9 Extra oefening op promises

Herneem de voorbereidende oefening op promises. Het programma wordt licht gewijzigd. Nu willen we de gebruiker getallen opvragen die achtereenvolgens veelvoud zijn van de noemers 2, 3 en 4. Zolang de gebruiker aan onze vraag voldoet, tellen we het quotiënt van dit getal en de vooropgestelde noemer op bij de som. Indien de gebruiker geen (geldig) getal opgeeft, stopt het programma. Zorg dat de startnoemer (in ons voorbeeld: 2) in het programma makkelijk kan aangepast worden. Zo kan het programma ook vlot lopen om veelvouden van 6, 7 en 8 (of 45, 46 en 47) te verwerken.

localhost:63342 says  
geef getal (veelvoud van 2)

OK Cancel

localhost:63342 says  
som is 13

OK

localhost:63342 says  
Programma vroegtijdig gestop. Dit is geen veelvoud van 3: 16

OK

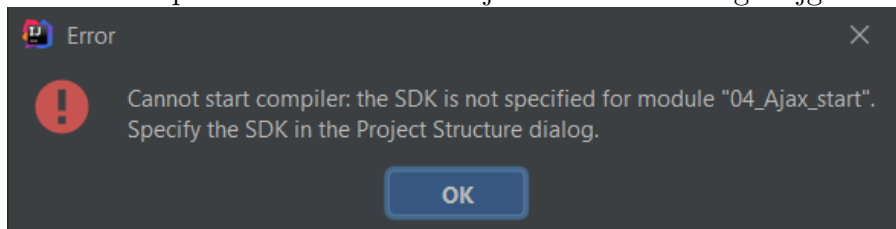
localhost:63342 says  
Programma vroegtijdig gestop. Dit is geen getal: x7

OK

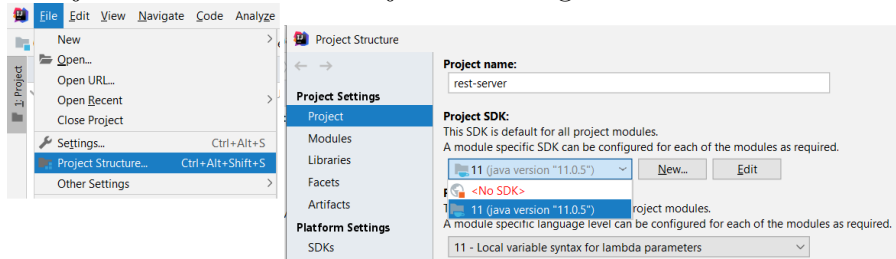
# Foutmeldingen en hun oplossingen

## Compiler start niet

Kan de compiler niet starten? Als je deze foutmelding krijgt



controleer je best of je met Java11 aan het werken bent. Open het venster Project Structure en kijk na. Eventueel moet je Java11 nog installeren.



## Poort al in gebruik

Indien poort 8080 al in gebruik is door een ander proces, krijg je deze foutmelding:

```
*****  
APPLICATION FAILED TO START  
*****
```

Description:

The Tomcat connector configured to listen on port 8080 failed to start. The port may already be in use or the connector may be misconfigured.

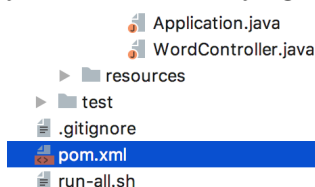
Action:

Verify the connector's configuration, identify and stop any process that's listening on port 8080, or configure this application to listen on another port.

Kopieer de foutmelding in een zoekmachine. Je zal aangeraden worden om command line te controleren welk proces op deze poort loopt. (Commando `netstat -aon` levert informatie over alle poorten.) Open dan de task manager en controleer in het tabblad *Details* met welk proces het gevonden 'Process Id' overeenkomt. In de task manager kan je dit proces stoppen (zoek eventueel eerst op of het een belangrijk proces zou kunnen zijn).

## 'Add Configuration' en rode cirkeltjes

Als er in plaats van  de boodschap **Add Configuration** staat, en een aantal java-bestanden zijn gemarkeerd met kleine (oranje)rode cirkeltjes



dan klik je rechts op het bestand `pom.xml` en kies je voor **Add as Maven Project** (onderaan). Zie ook [stackoverflow](https://stackoverflow.com), of gebruik de zoektermen `intellij red circle filename`.