Blok 1 – Client Server Interaction

Maaike van Putten
Software developer & trainer
www.brightboost.nl

Sessie 3: Van multi-page naar single page

AJAX, JSON, en DOM Manipulatie

Overview

HTTP & HTML FORMS

State management

AJAX & refresh components

PRG, validation & error handling

Agenda van vandaag

- Het probleem: page refreshes
- HTTP request/response herhaling
- JavaScript in de browser
- AJAX: asynchrone communicatie
- JSON: data format
- Fetch API: requests maken
- Servlets die JSON teruggeven
- DOM manipulatie
- Component gedachte
- **Veel praktijk!**

Flash back (literally)

Laten we kijken naar de dag 2 app...

- Screen flitst wit bij elke actie
- Hele pagina wordt opnieuw geladen
- Langzaam (vooral bij slechte connectie)

Moderne web apps

Hoe voelt Gmail? Google Maps? Spotify?

Kenmerken:

- ✓ Geen pagina refreshes
- ✓ Instant feedback
- ✓ Smooth transitions
- ✓ Voelt als een desktop app

Hoe doen ze dat?

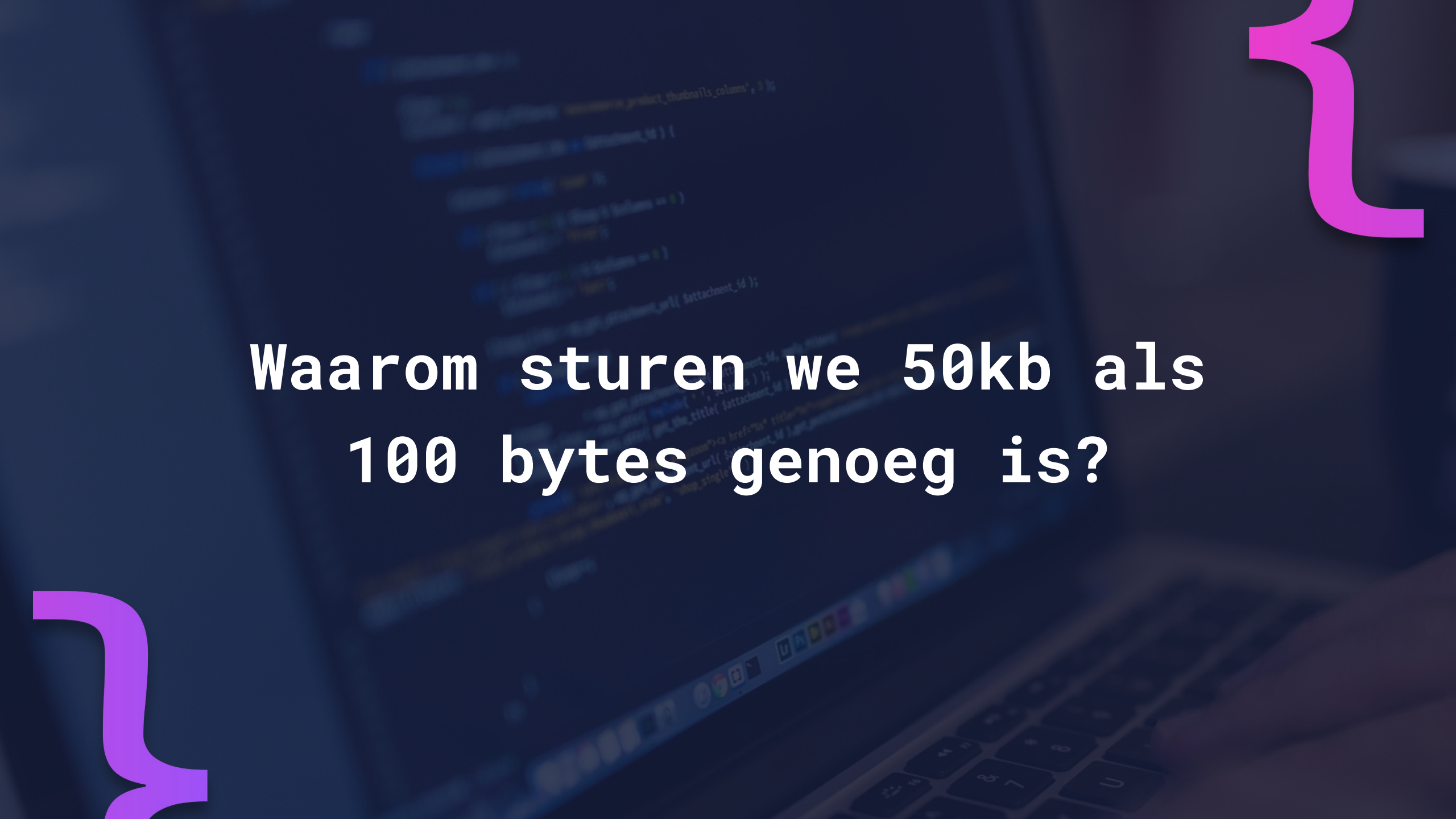
```
Browser --GET /step1--> Server
Browser <--HTML/CSS/JS-- Server (50kb)

Browser --POST data--> Server
Browser <--HTML/CSS/JS-- Server (50kb)

Browser --GET /step2--> Server
Browser <--HTML/CSS/JS-- Server (50kb)
```

Niet zo: HTTP cyclus - huidige situatie

Probleem is dat elke keer de hele pagina opnieuw laadt!

Bij elke request sturen we:

- HTML (verandert niet)
- CSS styling (verandert niet)
- JavaScript code (verandert niet)
- **Alleen de data verandert!**


Waarom sturen we 50kb als
100 bytes genoeg is?

```
Browser --GET /survey.html--> Server
Browser <--HTML/CSS/JS------- Server (1x, 50kb)

Browser --POST {name:"Jan"}--> Server
Browser <--{success:true}---- Server (100 bytes)

Browser --POST {city:"NY"}---> Server
Browser <--{success:true}---- Server (100 bytes)
```

De nieuwe aanpak

Oplossing: pagina 1x laden, daarna alleen data!

Web page ingrediënten

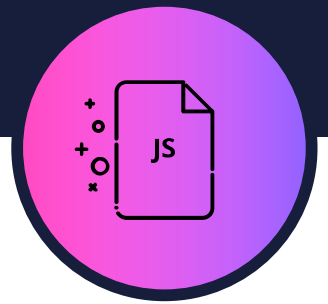
JavaScript draait in de browser en kan de pagina aanpassen!

HTML
Structuur

CSS
Uiterlijk

JavaScript
Gedrag

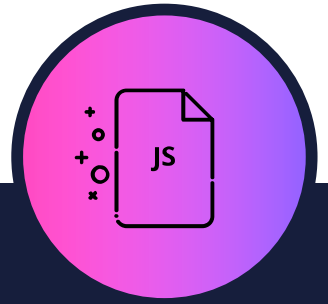
```
// Element ophalen
document.getElementById("title");

// Content aanpassen
element.textContent = "Nieuwe tekst";

// Style aanpassen
element.style.backgroundColor = "red";
```



DOM - Document Object Model

JavaScript’s representatie van de HTML pagina

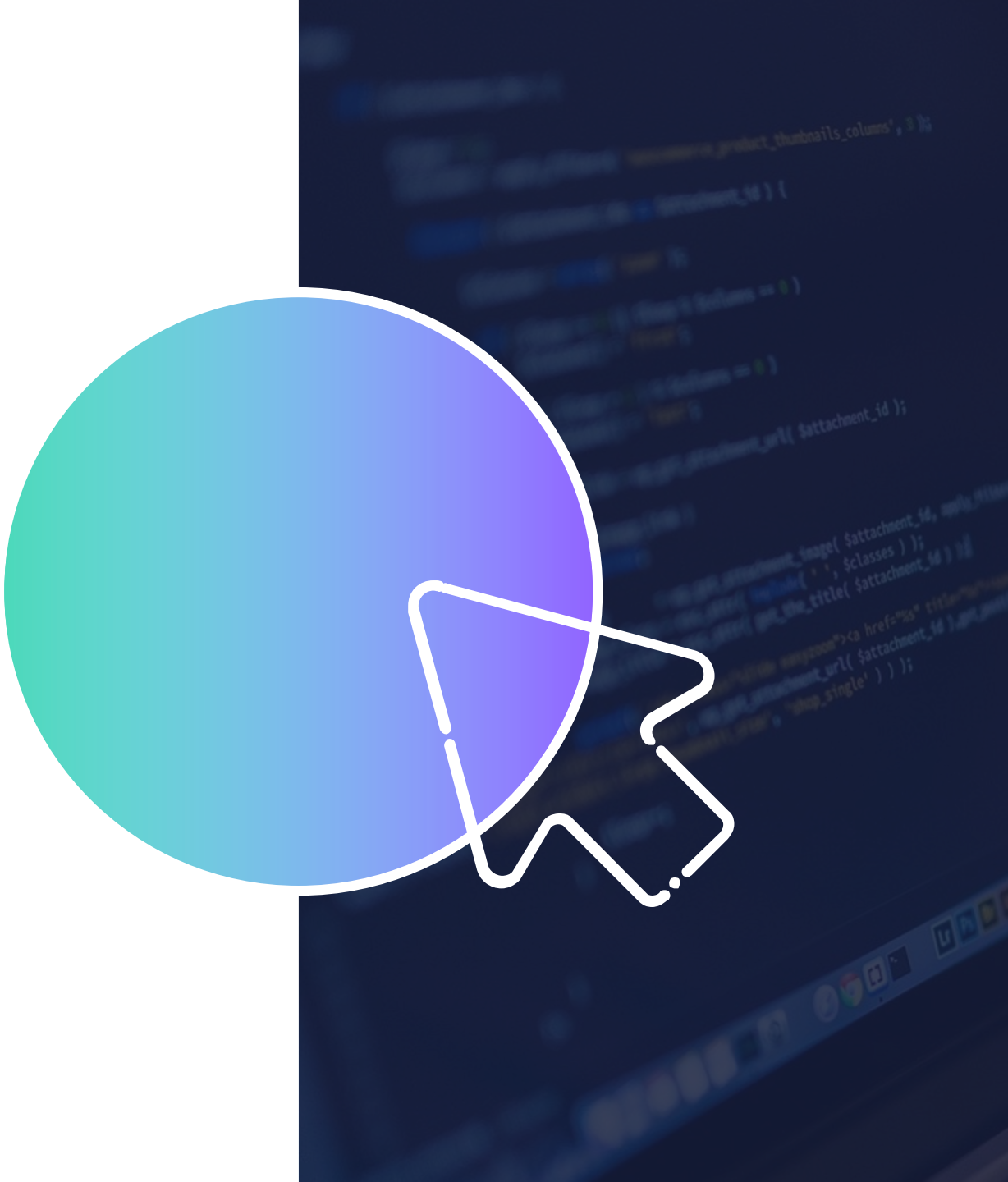
DOM manipulatie patronen




```
// Element tonen/verbergen
element.style.display = "block";
element.style.display = "none";

// CSS class toevoegen/verwijderen
element.classList.add("active");
element.classList.remove("active");

// Content aanpassen
element.textContent = "Nieuwe tekst";
element.innerHTML = "<strong>HTML</strong>";
```

Demo

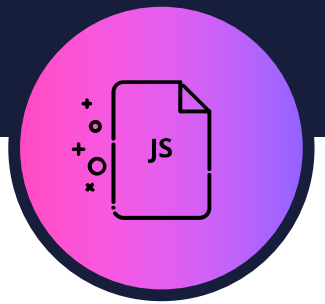
Editing websites using the console and JS



Oefening

Doe de exercise 1 “DOM manipulation” uit het mini exercises document.

```
// Button click
button.addEventListener("click", function () {
alert("Geklikt!");
});


// Form submit
form.addEventListener("submit", function (event) {
event.preventDefault(); // Stop normale submit!
// Doe iets anders...
});
```



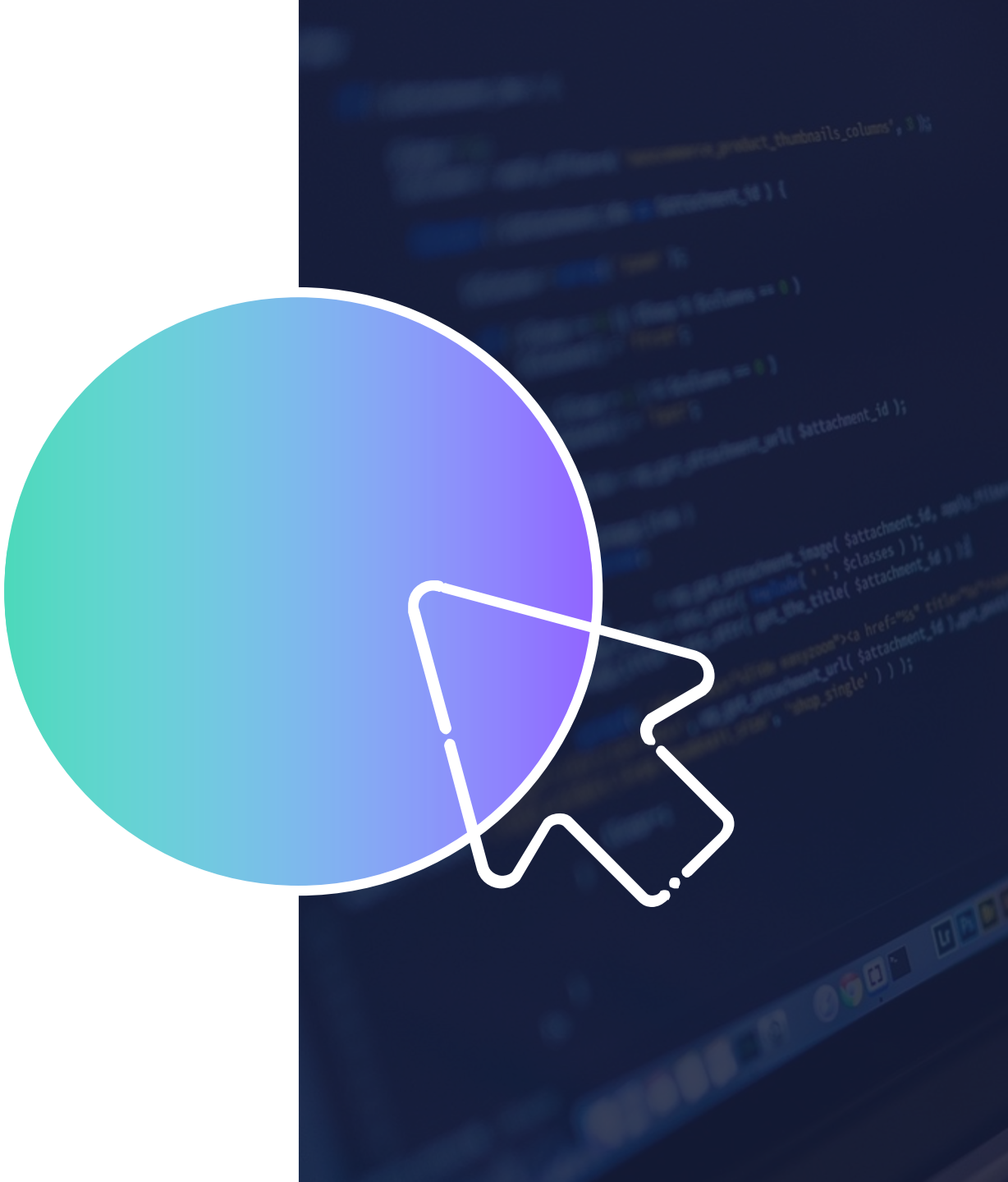
Event listeners

- JavaScript reageert op gebruikersacties

Belangrijk: `preventDefault()` stop normaal browsergedrag en dat hebben we meestal nodig bij submit!

Demo

beschrijving van demo



Oefening

Doe mini exercise 2 “Light bulb”

Project structuur

```
src/main/webapp/
├── index.html
├── css/
│   └── style.css
└── js/ ← Hier komen je .js files!
    └── survey.js
```

In HTML

```
<script src="/js/survey.js"></script>
```

Veelgemaakte fouten

Let op deze dingen!

× verkeerde path:
```
<script src="survey.js">
```
✓ Correcte path:
```
<script src="/js/survey.js">
```

× Vergeten leading slash (/)

✓ Altijd beginnen met /

Tip

Hard refresh met Ctrl+F5 (Windows) of Cmd+Shift+R (Mac)

Synchrone communicatie

De Ober Analogie: Restaurant **zonder** ober (oude manier)

Je loopt naar de keuken

Je wacht tot het eten klaar is

Je loopt terug met dienblad

Restaurant is opnieuw ingericht en je moet opnieuw gaan zitten!

Asynchrone communicatie

De Ober Analogie: Restaurant **met** ober (AJAX)

Je blijft zitten

De ober loopt naar de keuken

Jij kunt blijven praten, drinken

Ober komt terug met eten

Asynchrone communicatie

De Ober Analogie: Restaurant **met** ober (AJAX)

De ober loopt naar de keuken

Jij kunt blijven praten, drinken

Ober komt terug met eten

Je bent nooit opgestaan!

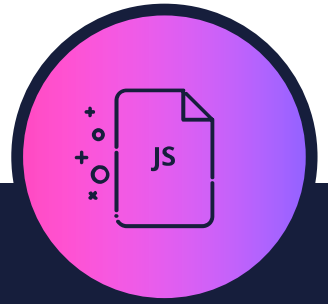
Asynchronous

Je doet andere dingen tijdens het wachten

Synchronous vs Asynchronous

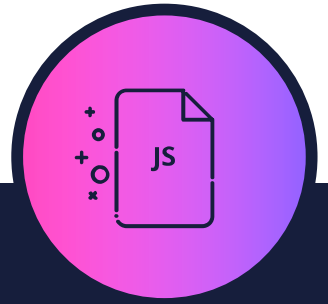
Code uitvoering

Synchronous (blocking)




```
console.log("1");
for (let i = 0; i < 1000000000; i++) {} // Wacht!
console.log("2");
// Output: 1, 2
```

Asynchronous (non-blocking)




```
console.log("1");
setTimeout(() => console.log("2"), 1000); // Wacht niet!
console.log(”3");
// Output: 1, 3, 2
```

Wat is AJAX?

Asynchronous JavaScript And XML

Asynchronous: wacht niet op response

JavaScript: draait in browser

And

XML (tegenwoordig Meestal JSON!)

AJAX = requests versturen vanuit JavaScript zonder page refresh

Waarom hebben we data format nodig?

Als we alleen data versturen, hoe ziet die eruit?

Optie 1: Plain text

```
"Jan,jan@email.com,Amsterdam"
```

Optie 2: XML

```
<user><name>Jan</name><city>Amsterdam</city></user>
```

Waarom hebben we data format nodig?

Als we alleen data versturen, hoe ziet die eruit?

Optie 1: Plain text

```
"Jan,jan@email.com"
```

Optie 2: XML

```
<user><name>Jan</name><email>…</email></user>
```

Optie 3: JSON

```
{"name": "Jan", "email": "jan@email.com"}
```

JSON

JavaScript Object Notation

JSON syntax

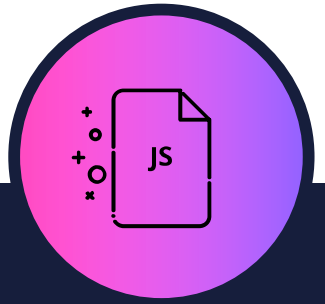
```json
{
  "key": "value",
  "number": 42,
  "boolean": true,
  "array": [1, 2, 3],
  "object": {
    "nested": "value",
    "nestedArray": [1, 2, 3]
  },
  "null": null
}
```

JSON regels

- ✓ Keys **altijd** tussen dubbele quotes: `"name"`
- ✓ Strings tussen dubbele quotes: `"jan"`
- ✓ Getallen zonder quotes: `42`
- ✓ Booleans: `true` of `false`
- ✓ Arrays: `[1, 2, 3]`
- ✓ Objects: `{"key", "value"}`

- × Geen trailing comma’s!
- × Geen comments mogelijk
- × Geen single quotes

Converteren tussen JSON en JS




```
// JavaScript object to JSON string
let obj = { name: "Jan", age: 25 };
let json = JSON.stringify(obj);
// Result: '{"name":"Jan","age":25}'


// JSON string to JavaScript object
let jsonString = '{"name":"Jan","age":25}';
let obj2 = JSON.parse(jsonString);
// Result: {name: "Jan", age: 25}
```

AJAX requests maken

Oudere manier

XMLHttpRequest

Nieuw manier

Fetch API

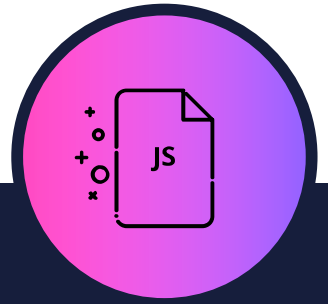
Demo

XMLHttpRequest demo

Oefening

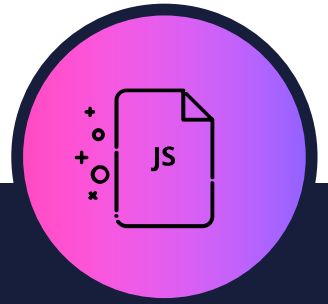
Recepten pagina maken! Volg mini exercise 3

Fetch API



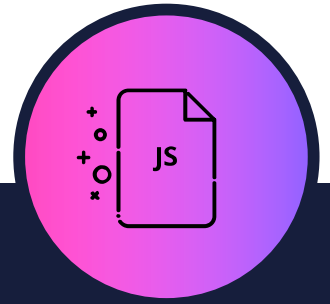

```
fetch("/api/data")
  .then((response) => response.json())
  .then((data) => {
    console.log(data);
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

Data ophalen van de server - GET request




```
fetch("/api/hello")
  .then((response) => response.json())
  .then((data) => {
    // Gebruik data hier
    console.log(data.message);
    document.getElementById("result").textContent = data.message;
  })
  .catch((error) => {
    console.error("Fout:", error);
  });
```

Data versturen naar de server - POST request




```
// FormData van een form element
let formData = new FormData(formElement);

fetch("/api/save", {
  method: "POST",
  body: formData,
})
  .then((response) => response.json())
  .then((data) => {
    if (data.success) {
      alert("Opgeslagen!");
    }
  })
  .catch((error) => console.error(error));
```

```
form.addEventListener("submit", function (event) {
  event.preventDefault(); // Belangrijk!

  // Nu doen we AJAX submit in plaats van normale submit
  let formData = new FormData(form);
  fetch("/api/save", {
    method: "POST",
    body: formData,
  })
    .then((response) => response.json())
    .then((data) => console.log(data));
});
```

Event PreventDefault

Stop normale form submit!

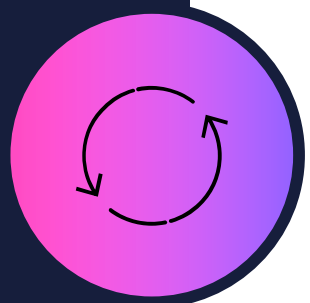
Zonder preventDefault: normale submit + page refresh!

Error handling

```
fetch("/api/save", {
  method: "POST",
  body: formData,
})
  .then((response) => response.json())
  .then((data) => {
    // Success!
    showSuccessMessage();
  })
  .catch((error) => {
    // Error! (network error, server down, etc.)
    showErrorMessage();
    console.error(error);
  });
```

Altijd error handling implementeren!

```
// Toon loading indicator
showLoading();

fetch("/api/save", {
  method: "POST",
  body: formData,
})
  .then((response) => response.json())
  .then((data) => {
    hideLoading(); // Verberg loading
    showSuccess();
  })
  .catch((error) => {
    hideLoading(); // Verberg loading
    showError();
  });
```



Loading states

Gebruiker laten weten dat er iets gebeurt

Oefening

Recepten pagina fetch versie! Volg mini exercise 4

Servlets: van HTML naar JSON

Oude manier vs nieuwe manier

Oude manier (day 1-2)

```java
// Java

response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("<h1>Hello</h1>");
out.println("</body></html>");
```

Nieuwe manier (day 3)

```java
// Java

response.setContentType("application/json");
PrintWriter out = response.getWriter();
out.println("{\"message\": \"Hello\"}");
```

Content-type belangrijk!

Browser moet weten wat het krijgt

Zonder content-type

```
// Java
response.getWriter().write("{\"success\": true}");
```

Met content-type

```
// Java
response.setContentType("application/json");
response.getWriter().write("{\"success\": true}");
```

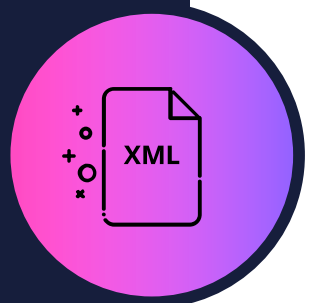
Content-type belangrijk!

Browser moet weten wat het krijgt

Met content-type

```
// Java
response.setContentType("application/json");
response.getWriter().write("{\"success\": true}");
```

Aha, dit is JSON!



Gson Library

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.10.1</version>
</dependency>
```

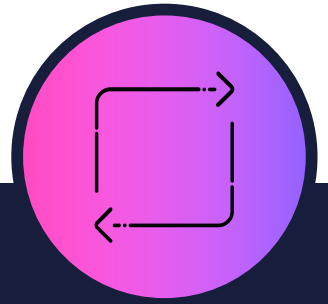
JSON genereren zonder fouten

Voorbeeld Maven dependency

Waarom Gson?

- Geen syntax fouten
- Automatisch escapen
- Type-safe
- Java object to JSON

Gson gebruik: Java object naar JSON



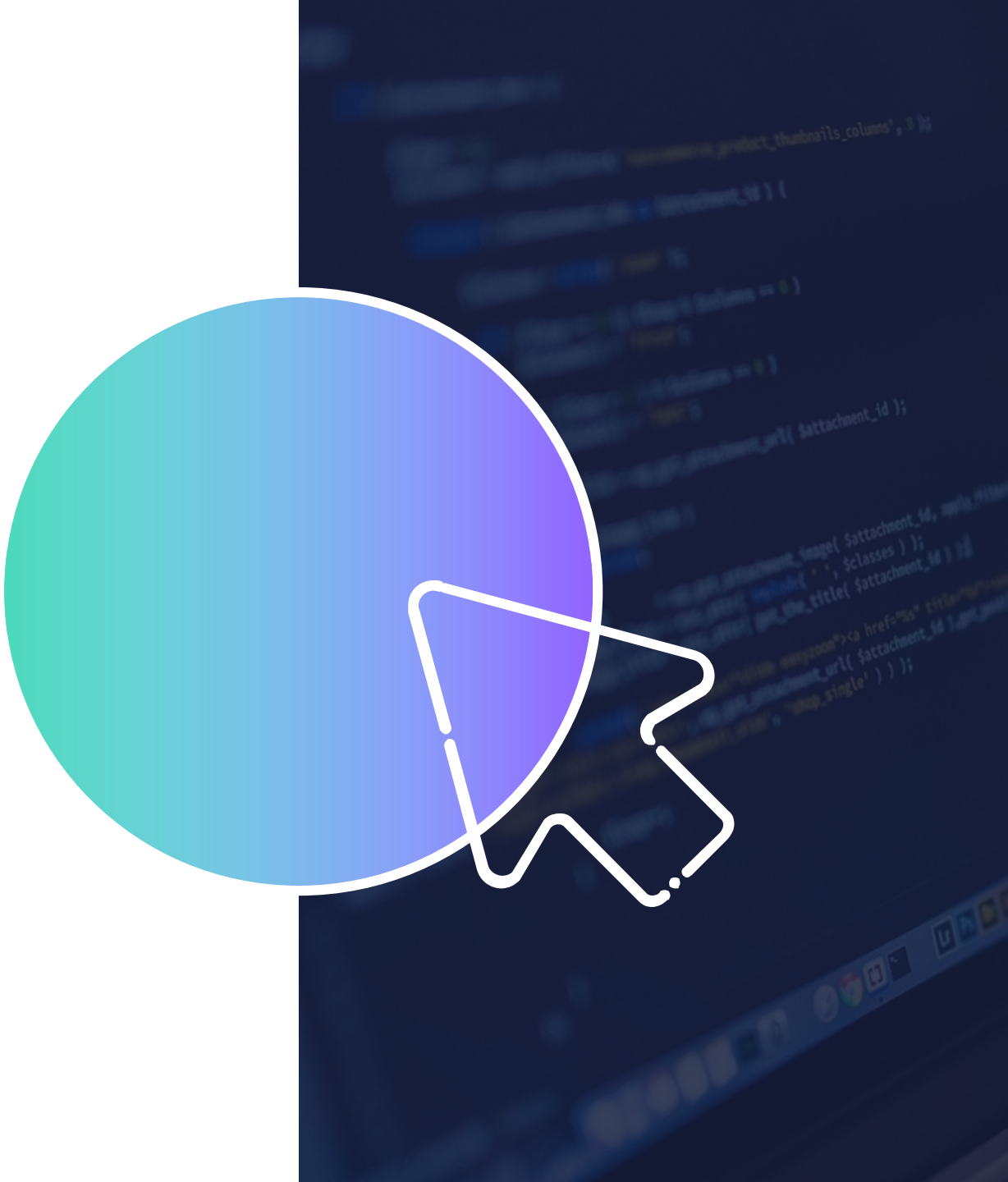
```java
// Java

import com.google.gson.Gson;

Gson gson = new Gson();


// Map naar JSON
Map<String, Object> response = new HashMap<>();
response.put("success", true);
response.put("message", "Saved!");
response.put("nextStep", 2);


String json = gson.toJson(response);
// Result: {"success":true,"message":"Saved!","nextStep":2}
```

Complete Servlet voorbeeld: JSON API endpoint

```
@WebServlet("/api/save-step1")
public class SaveStep1Servlet extends HttpServlet {
    private Gson gson = new Gson();

    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response) {
        String name = request.getParameter("name");

        // Save to session
        request.getSession().setAttribute("name", name);
        // Return JSON
        Map<String, Object> json = new HashMap<>();
        json.put("success", true);

        response.setContentType("application/json");
        response.getWriter().write(gson.toJson(json));
    }
}
```

Demo

Servlet & SPA



Browser DevTools - Network Tab

Debug je AJAX requests!

- Open DevTools: **F12**

- Network tab laat zien:
  - Alle HTTP requests
  - Request headers & body
  - Response headers & body
  - Status codes
  - Timing informatie

- **Filter op:** XHR/Fetch (alleen AJAX requests)

Oefening 1 - Hello AJAX

Opdracht:

Maak een pagina met een knop. Bij klik:

- JS stuurt request naar servlet
- Servlet geeft random getal terug (JSON)
- Toon getal op pagina zonder refresh

Leerdoelen:

- Eerste fetch request
- JSON response ontvangen
- DOM updaten

(30 minuten)

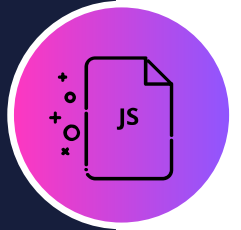
Multi-step form pattern

Steps wisselen zonder page refresh


```html
<div id="step1" class="step active">...</div>
<div id="step2" class="step">...</div>
<div id="step3" class="step">...</div>
```


```css
.step {
  display: none;
}
.step.active {
  display: block;
}
```

```javascript
function showStep(number) {
  document.querySelectorAll(".step").forEach((step) => {
    step.classList.remove("active");
  });
  document.getElementById("step" + number).classList.add("active");
}
```

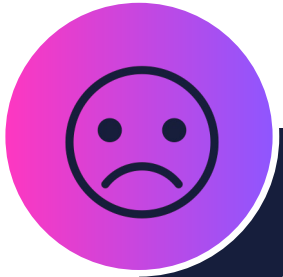


Wat gebeurt er precies?

1. **Browser:** HTML pagina laden (eenmalig)
2. **Gebruiker:** Form invullen
3. **Browser:** Submit button klikken
4. **JavaScript:** Grijp in met p
5. **JavaScript:** Maak Form ch
6. **Servlet:** Data ontvang
7. **Servlet:** Opslaan in se
8. **Servlet:** JSON response sturen
9. **JavaScript:** Response ontvangen en parsen
10. **JavaScript:** DOM updaten (volgende step tonen)

Geen enkele page refresh!

User Experience vergelijking



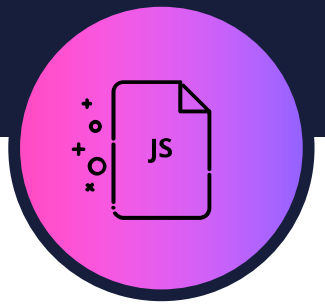
Day 1 (Multi-page)

- × Elke actie = page refresh
- × Screen flitst wit
- × Langzaam (50kb per request)
- × Data kwijt bij back button
- × Geen loading states mogelijk

Day 3 (Single-page)

- ✓ Geen page refreshes
- ✓ Smooth transitions
- ✓ Snel (100 bytes per request)
- ✓ State behouden
- ✓ Loading indicators

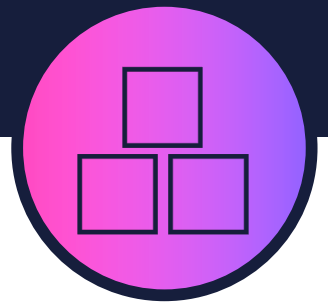
```
setupStep1();
setupStep2();
setupStep3();
// ... setupStep10();
```



Code organisatie

Probleem: 10 steps = 10x dezelfde code?

```
Component = HTML + Behavior + State
```



Code organisatie

Oplossing: Componenten!

Component voorbeeld: herbruikbare bouwstenen

```java
// Java
public abstract class Component {
    protected String id;

    // Render als HTML (initial load)
    public abstract String renderHtml();

    // Render als JSON (AJAX updates)
    public abstract String renderJson();

    // Handle events
    public abstract void handleEvent(String event);
}
```

Wicket's aanpak

Component-based framework

Wicket principe:

1. Component tree in session
2. Component re-render server-side
3. Stuur alleen gewijzigde HTML via AJAX
4. JavaScript update DOM

Jullie bouwen een “Wicket-light”!

Security & best practices

Altijd valideren op server!

- Client-side = UX (gebruiksvriendelijkheid)
- Server-side = Security (veiligheid)
- JavaScript kan uitgeschakeld worden!

HTTPS in productie

- Secure cookies
- Geen gevoelige data in URLs

Error handling

- Geen stack traces naar client
- Log errors server-side

CSRF protection

Cross-**S**ite **R**equest **F**orgery

Probleem:

- Externe site kan requests doen naar jouw server

Oplossing: CSRF tokens

- Token in form
- Server valideert token
- Token is uniek per sessie

Debugging tips

Browser console

- F12 → Console tab
- Zie JavaScript errors
- Zie `console.log()` output

Network tab

- Zie alle requests
- Check response bodies
- Check status codes

JSONLint

- Valideer je JSON
- Vind syntax fouten

Veelgemaakte fouten

Let hierop!

- × Vergeten `event.preventDefault()`
- × Verkeerde Content-Type
- × Niet `.json()` aanroepen op response
- × Geen error handling
- × CORS issues (gebruik http://localhost)
- × Browser cache (Ctrl+F5!)

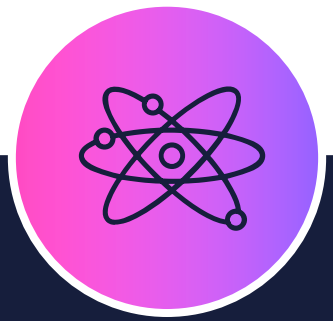
Next up:

Kaasus time!

Hoe verhoudt zich dit tot...

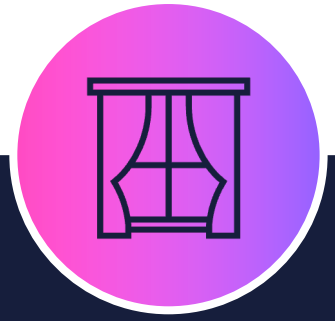
Wicket

- Components in session
- Server-side rendering
- AJAX updates



React/Vue

- Components in browser
- Client-side rendering
- Virtual DOM



Jullie framework

Mix van beide!

Next up:

Session 4: PRG, validation & error handling

Questions or suggestions?

Maaike.vanputten@brightboost.nl

See you next time! 💖