

# Web Framework Bouwen – Opdrachten



## Wat we gaan doen

We gaan ons eigen web app framework maken! Inspiratie hiervoor is Apache Wicket. Dit wordt een soort Apache Wicket, maar dan "ultra light" omdat we natuurlijk het wel wat moeten versimpelen om in 4 middagen + 1 dag iets dergelijks te bouwen.

Web frameworks zoals Wicket laten ons een HTML/CSS/JS app maken middels het schrijven van HTML en Java.

Een belangrijk kenmerk van het framework dat we gaan maken is dat deze **component-based** is. Dat wil zeggen dat pagina's uit herbruikbare componenten bestaan zoals formulieren, tabellen, knoppen. Ieder component heeft zijn eigen gedrag en zijn eigen HTML. Op deze manier kunnen we stukken code herbruiken en hebben we een meer modulaire architectuur wat het testen vergemakkelijkt.

## Wat er nodig is

Om dit te maken hebben we een aantal dingen nodig:

- **Servlet foundation** - centrale servlet die alle requests afhandelt
- **Component model** - basis Component class waar alles van overerft
- **Rendering systeem** - componenten kunnen zichzelf naar HTML renderen
- **State management** - component state bewaren tussen requests
- **Event handling** - knoppen en forms kunnen callbacks triggeren

## Hoe we het gaan doen

De tijdlijn van 4 middagen is ambitieus. Het beste advies is op dit moment dan ook **KISS** (Keep It Simple, Stupid). We bouwen het framework op in de volgende stappen:

### Dag 1: Foundation - HTTP begrijpen

- Servlets maken die GET/POST requests afhandelen
- HTML forms verwerken
- Data handmatig doorgeven tussen pagina's
- *Doel: De pijn voelen van stateless HTTP*

### Dag 2: State oplossen

- Sessions implementeren voor data opslag
- JWT tokens voor authenticatie
- Eerste abstractie: Page classes die state beheren
- *Doel: Begrijpen waarom frameworks state management nodig hebben*

### **Dag 3: Van multi-page naar single-page**

- AJAX/XMLHttpRequest toevoegen
- Servlets die JSON retourneren
- Component model: classes die zichzelf kunnen renderen als HTML en/of JSON
- DOM manipulatie met JavaScript
- *Doel: Client-server communicatie patronen leren*

### **Dag 4: Production-ready maken**

- Server + client-side validatie
- Post-Redirect-Get pattern
- Error handling en user feedback
- Back button handling
- Component lifecycle hooks
- *Doel: Idiot proof applicaties bouwen*

### **Dag 5: Opschonen**

- Patronen herkennen in je code
- Gemeenschappelijke code extraheren naar base classes
- Demonstreren dat je "framework" herbruikbaar is
- *Doel: Zien hoe frameworks ontstaan uit real-world problemen*

### **Progressie samengevat**

Week 1: Plain servlets

Week 2: Sessions/JWT

Week 3: AJAX/Components

Week 4: Validatie/Patterns

Week 5: Framework in gebruik

### **Bonus challenges (als je tijd over hebt):**

- HTML templating met attributes
- Bidirectional mapping tussen HTML en Java componenten
- URL-based component callbacks
- Unit tests voor je framework



# Opdracht dag 1: HTTP + HTML Forms

**Doel van de dag:** Een simpele multi-step form applicatie bouwen die de HTTP request/response cycle demonstreert.

## Wat je gaat maken

Een vragenlijst/enquête applicatie met meerdere opeenvolgende formulieren (bijvoorbeeld: stap 1 = naam/email, stap 2 = adres, stap 3 = voorkeuren, stap 4 = overzicht).

## Technische vereisten

### 1. Servlet voor GET requests

- Maak een servlet die HTML forms kan tonen
- Gebruik `@WebServlet` annotatie of `web.xml` configuratie
- Implementeer `doGet()` methode

### 2. Servlet voor POST requests

- Implementeer `doPost()` methode voor form submissions
- Extract request parameters met `request.getParameter()`
- Bijvoorbeeld: `String name = request.getParameter("name");`

### 3. Data tussen paginas doorgeven

- Query params worden gebruikt om data tussen requests door te geven (via GET/redirect)
- Hidden fields worden gebruikt om data in forms mee te sturen (via POST)

```
<input type="hidden" name="name" value="Jan">
<input type="hidden" name="email" value="jan@example.com">
```

### 4. HTML generatie

- Gebruik `response.setContentType("text/html")`
- Schrijf HTML via `PrintWriter: response.getWriter().println("<html>...")`
- Afrader: het gebruik van JSP files (dit is weliswaar simpeler maar daardoor minder leerzaam)

## Concrete taken

1. **Setup project**
  - Maven project met servlet-api dependency
  - Basic project structuur opzetten
2. **Eerste servlet maken**
3. **Minimaal 3-4 stappen implementeren**
  - Elke stap is een apart servlet of één servlet met state parameter
  - Data doorgeven via URL params of hidden fields
4. **Simpele validatie** (optioneel voor dag 1)
  - Check of required fields niet leeg zijn
  - Toon foutmelding als validatie faalt

## Leerdoelen

- **Begrijpen dat HTTP stateless is** (elke request is onafhankelijk)
- **Ervaren dat data doorgeven lastig is** en dat er veel boilerplate code nodig is
- **Handmatig parameter handling is foutgevoelig** door type conversies, null checks, etc.

## Verwachte problemen (dat is goed!)

- "Waarom moet ik steeds dezelfde data meesturen?"
- "Dit is veel code voor iets simpels"
- "Wat als de gebruiker de back button gebruikt?"



## Opdracht dag 2: State management

Refactor de dag 1 app om de pijn van stateless HTTP op te lossen door state management toe te voegen. Je leert het verschil tussen cookies, sessions en JWT tokens door ze voor verschillende doeleinden te gebruiken.

## Wat je gaat maken

Dezelfde enquête app van dag 1, maar nu met:

- **Cookies** voor gebruikersvoorkeuren (taal, thema)
- **Sessions** voor enquête data opslag
- **Authenticatie** (login systeem - keuze tussen sessions of JWT)
- **Page abstractie** die state beheert

## Waarom deze verdeling?

Dit is hoe echte web frameworks zoals Wicket het doen:

- **Cookies** = client-side preferences die je wilt onthouden over meerdere bezoeken
- **Sessions** = server-side application state die tijdelijk is (zoals formulier data)
- **Authenticatie** = wie is de gebruiker? (kan via sessions of JWT)

## Technische vereisten

### 1. Cookies voor gebruikersvoorkeuren

#### Wat zijn cookies?

- Kleine stukjes data die de browser onthoudt
- Worden bij elke request meegestuurd
- Blijven bestaan na het sluiten van de browser (als je dat wilt)

#### Wat je moet implementeren:

- **Taal selectie** (bijvoorbeeld: nl, en)
  - Cookie naam: `language`
  - Gebruik deze cookie om je HTML in de juiste taal te tonen
  - Maak een klein dropdown menu: " Nederlands | English"
- **Thema selectie** (bijvoorbeeld: light, dark)
  - Cookie naam: `theme`
  - Pas je CSS aan op basis van het thema
  - Maak een toggle button: " Light | Dark"

#### Belangrijke cookie eigenschappen:

- `setMaxAge(seconds)` - Hoe lang blijft de cookie bestaan?
  - `-1` = tot browser wordt gesloten (session cookie)
  - `0` = verwijder de cookie
  - `> 0` = aantal seconden
- `setPath("/")` - Voor welke URLs geldt de cookie?
- `setHttpOnly(true)` - Cookie niet toegankelijk via JavaScript (veiliger)
- `setSecure(true)` - Alleen versturen via HTTPS

### 2. Sessions voor enquête data

#### Wat zijn sessions?

- Server-side opslag (data blijft op de server)
- Browser krijgt alleen een session ID cookie
- Perfect voor tijdelijke application state

## Waarom sessions voor formulier data?

- Data is tijdelijk (alleen tijdens deze enquête)
- Bevat mogelijk gevoelige info
- Moet verdwijnen als gebruiker klaar is

## Wat je moet implementeren:

### 1. Refactor je Day 1 code:

- Verwijder alle hidden fields
- Verwijder alle URL parameters voor data doorgeven
- Gebruik `session.setAttribute()` in plaats daarvan

### 2. Session lifecycle beheren:

- Bij stap 1: nieuwe session starten
- Bij elke stap: data toevoegen aan session
- Bij laatste stap: session data tonen en dan `session.invalidate()`

### 3. Session timeout configureren (optioneel):

## 3. Authenticatie (keuze tussen Sessions of JWT)

### Waarom authenticatie?

- Alleen ingelogde gebruikers mogen de enquête invullen
- We willen weten wie de enquête heeft ingevuld

Je hebt **twee opties** voor authenticatie. Kies er één:

### Optie A: Session-based authenticatie (Eenvoudiger, aanbevolen)

#### Hoe werkt het?

1. Gebruiker logt in met username/password
2. Server controleert credentials
3. Bij success: `session.setAttribute("username", username)`
4. Bij volgende requests: check of `session.getAttribute("username")` bestaat

### Optie B: JWT-based authenticatie (Geavanceerder, stateless)

#### Hoe werkt het?

1. Gebruiker logt in met username/password

2. Server controleert credentials
3. Bij success: genereer JWT token met username erin
4. Token opslaan in cookie
5. Bij volgende requests: valideer token uit cookie

## 4. Page abstractie toevoegen

### Waarom een Page class?

- DRY principe: gemeenschappelijke code voor alle pagina's
- Centraal punt voor session management
- Eerste stap naar je framework!

### Voorbeeld Page class:

```
public abstract class Page {  
    protected HttpServletRequest request;  
    protected HttpServletResponse response;  
    protected HttpSession session;  
  
    public Page(HttpServletRequest request, HttpServletResponse response) {  
        this.request = request;  
        this.response = response;  
        this.session = request.getSession();  
    }  
  
    // Elke page moet zichzelf kunnen renderen  
    public abstract void render() throws IOException;  
  
    // Helper methods die elke page kan gebruiken  
    protected void setSessionData(String key, Object value) {  
        session.setAttribute(key, value);  
    }  
  
    protected Object getSessionData(String key) {  
        return session.getAttribute(key);  
    }  
}
```

```

protected String getUsername() {
    return (String) session.getAttribute("username");
}

protected String getLanguage() {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if ("language".equals(cookie.getName())) {
                return cookie.getValue();
            }
        }
    }
    return "nl"; // default
}

protected String getTheme() {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if ("theme".equals(cookie.getName())) {
                return cookie.getValue();
            }
        }
    }
    return "light"; // default
}

protected PrintWriter getWriter() throws IOException {
    response.setContentType("text/html; charset=UTF-8");
    return response.getWriter();
}
}

```



## Voorbeeld van een concrete page:

```
public class Step1Page extends Page {

    public Step1Page(HttpServletRequest request, HttpServletResponse response) {
        super(request, response);
    }

    @Override
    public void render() throws IOException {
        PrintWriter out = getWriter();
        String language = getLanguage();
        String theme = getTheme();
        String username = getUsername();

        // Gebruik language en theme om je HTML te genereren
        out.println("<html>");
        out.println("<head>");
        out.println("<style>");
        if ("dark".equals(theme)) {
            out.println("body { background: #222; color: #eee; }");
        } else {
            out.println("body { background: #fff; color: #222; }");
        }
        out.println("</style>");
        out.println("</head>");
        out.println("<body>");

        if ("en".equals(language)) {
            out.println("<h1>Welcome " + username + "!</h1>");
            out.println("<p>Step 1: Personal information</p>");
        } else {
            out.println("<h1>Welkom " + username + "!</h1>");
            out.println("<p>Stap 1: Persoonlijke informatie</p>");
        }
    }
}
```

```

    }

    out.println("<form method='post' action='/survey/step1'>");
    // ... form fields ...
    out.println("</form>");
    out.println("</body></html>");
}
}

```

### Servlet die je Page gebruikt:

```

@WebServlet("/survey/step1")
public class Step1Servlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        Step1Page page = new Step1Page(request, response);
        page.render();
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        // Verwerk form data
        String name = request.getParameter("name");
        String email = request.getParameter("email");

        // Sla op in session
        HttpSession session = request.getSession();
        session.setAttribute("name", name);
        session.setAttribute("email", email);

        // Ga naar volgende stap
        response.sendRedirect("/survey/step2");
    }
}

```

# Concrete taken

## 1. Cookies implementeren

### Taken:

- Maak een servlet `/preferences` die GET en POST afhandelt
- Implementeer taal selectie (nl/en) met cookie opslag
- Implementeer thema selectie (light/dark) met cookie opslag
- Pas je enquête pagina's aan om de taal cookie te gebruiken
- Pas je CSS aan op basis van de thema cookie
- Voeg een preference menu toe aan elke pagina (bijvoorbeeld in de header)

### Test:

- Verander je taal, refresh de pagina > taal blijft hetzelfde
- Sluit je browser, open opnieuw > taal is nog steeds je keuze
- Verander thema > CSS past direct aan

## 2. Sessions implementeren

### Taken:

- Refactor je Day 1 code: verwijder alle hidden fields
- Refactor je Day 1 code: verwijder alle URL parameters voor data
- Gebruik `session.setAttribute()` om formulier data op te slaan
- Bij laatste stap: toon alle verzamelde data uit de session
- Na voltooien enquête: `session.invalidate()` en toon "bedankt" pagina
- Test met meerdere browser tabs: zie je dat ze dezelfde session delen?

### Test:

- Vul stap 1 in, ga naar stap 2 > data van stap 1 is beschikbaar
- Sluit browser tab (niet browser), open nieuwe tab > session data is weg
- Wacht 30 minuten zonder activiteit > session is expired

## 3. Authenticatie implementeren (kies Optie A of B)

### Taken (algemeen):

- Maak een login pagina (`/login`) met username/password form
- Maak een `LoginServlet` die credentials controleert
- Kies: implementeer **Optie A** (sessions) **of Optie B** (JWT)
- Maak een `AuthenticationFilter` die `/survey/*` beschermt

- Maak een logout functionaliteit
- Test: probeer `/survey/step1` te openen zonder login > redirect naar `/login`

#### Als je Optie A kiest (sessions):

- Bij succesvolle login: `session.setAttribute("username", username)`
- Filter checkt: `session.getAttribute("username") != null`
- Bij logout: `session.invalidate()`

#### Als je Optie B kiest (JWT):

- Maak `JwtUtil` class met `generateToken()` en `validateTokenAndGetUsername()`
- Bij succesvolle login: genereer token en sla op in `auth_token` cookie
- Filter haalt token uit cookie en valideert deze
- Bij logout: verwijder cookie met `setMaxAge(0)`

## 4. Page abstractie toevoegen

#### Taken:

- Maak een abstract `Page` base class
- Implementeer helper methods: `getLanguage()`, `getTheme()`, `getUsername()`
- Maak concrete Page classes voor elke stap (Step1Page, Step2Page, etc.)
- Refactor je servlets om Page classes te gebruiken
- Elke Page heeft een `render()` methode die HTML genereert

#### Test:

- Code duplicatie moet drastisch afgenomen zijn
- Alle pagina's gebruiken dezelfde helpers voor cookies/sessions

## Opdracht dag 3: AJAX en XMLHttpRequest



**Doel van de dag:** Transformeer de multipage form naar een Single Page Application met dynamische updates.

### Wat je gaat maken

De enquête wordt één pagina waar form secties dynamisch laden/updaten zonder page reloads.

## Technische vereisten

1. JavaScript XMLHttpRequest
2. Servlets die JSON retourneren
3. DOM manipulatie

## Concrete taken

1. **HTML structuur aanpassen**
  - Één pagina met meerdere `<div>` secties
  - Initieel alleen eerste sectie zichtbaar
  - Andere secties `style="display:none"`
2. **JavaScript functies schrijven**
  - `submitStep(stepNumber)` - submit data via AJAX
  - `showStep(stepNumber)` - toon/verberg secties
  - `updateFeedback(message)` - toon success/error messages
3. **API endpoints maken**
  - `/api/step1` - POST endpoint voor stap 1
  - `/api/step2` - POST endpoint voor stap 2
  - etc.
  - Elk endpoint retourneert JSON
4. **JSON parsing library toevoegen**
  - Google Gson of Jackson
  - Maven dependency toevoegen
5. **Component rendering uitbreiden**
  - Componenten kunnen zowel HTML als JSON output genereren
  - `component.render()` voor server-side HTML
  - `component.toJSON()` voor AJAX responses

## Leerdoelen

- **Client-server communicatie patronen**
- **JSON serialization/deserialization**
- **Asynchrone requests vs synchrone page loads**
- **Separation of concerns**, in dit geval data vs presentatie

## Meer nodig? (optioneel)

- Fetch API gebruiken in plaats van XMLHttpRequest (moderner)
- Loading spinners tijdens AJAX calls
- Error handling voor network failures



## Opdracht dag 4: Production-Ready Patterns

**Doel van de dag:** De applicatie robuust maken met validatie, proper navigation, en error handling.

### Wat je gaat maken

Een gepolijste enquête app met:

- Server + client-side validatie
- Post-Redirect-Get pattern
- User-friendly error handling
- Proper back button support

### Technische vereisten

1. Server-side validatie
2. Client-side validatie via AJAX
3. Post-Redirect-Get (PRG) pattern
4. Error handling
5. Back button handling

#### Optie A: History API

```
// Bij elke stap wijziging
history.pushState({step: 2}, '', '/survey?step=2');

// Luister naar back button
window.addEventListener('popstate', function(event) {
  if (event.state) {
    loadStep(event.state.step);
  }
});
```

#### Optie B: State management

- Bewaar huidige stap in session
- Bij page load: check welke stap actief moet zijn
- Laat gebruiker altijd teruggaan naar vorige stappen

#### 6. Bonus: CSRF protection

## Concrete taken

1. **Validation layer toevoegen**
  - Maak `Validator` class
  - Validatie regels voor elk veld
  - Server-side: valideer voor opslaan
  - Client-side: real-time feedback via AJAX
2. **PRG pattern implementeren**
  - Alle POST requests eindigen met redirect
  - Success/error messages via session flash attributes
  - Test: refresh na submit mag geen dubbele submit geven
3. **Error pages maken**
  - 404 pagina
  - 500 error pagina
  - User-friendly foutmeldingen
  - Logging van errors (bijv. naar console of file)
4. **Back button strategie kiezen**
  - History API voor SPA
  - Of: gebruiker mag altijd terug naar vorige stappen
  - Test grondig met browser back button
5. **Component lifecycle uitbreiden**
6. **CSRF tokens toevoegen** (bonus)
  - Token generator
  - Token validatie
  - Automatisch toevoegen aan forms

## Leerdoelen

- **Production concerns** zoals security, UX, error handling
- **PRG pattern** toepassen
- **Validatie architectuur**: client vs server
- **Browser quirks**: back button, refresh, bookmarks

## Opdracht dag 5: Wrap up en presentatie

**Doel van vandaag:** Project afronden en presenteren wat je geleerd hebt.

## Planning

### Individuele quiz

- Meerkeuzevragen over HTTP, sessions, AJAX, etc.



- Test je begrip van de concepten

### **Project afronden**

- Bugs fixen
- Code opschonen
- README schrijven
- Presentatie voorbereiden

### **Presentaties**

- Elk team max 20 minuten
- 15 min presentatie + 5 min Q&A

### **Afsluitende Kahoot**

- Leuke quiz om af te sluiten
- Mix van serieuze en fun vragen

## **Voorbeeld presentatie structuur**

### **1. Team intro (1 minuut)**

- Wie zijn jullie?
- Hoe was de samenwerking?

### **2. Product demo (3 minuten)**

- Live demo van de applicatie
- Laat zien wat het doet
- Highlight key features

### **3. Architectuur (4 minuten)**

- Toon architectuur diagram
- Leg component model uit
- Belangrijkste design decisions
- Code voorbeelden

### **4. Technical deep dive (4 minuten)**

- Kies 1-2 interessante technische aspecten
- Bijvoorbeeld: hoe werkt jullie rendering systeem?
- Of: hoe hebben jullie state management opgelost?



## 5. Lessen & uitdagingen (3 minuten)

- Wat hebben jullie geleerd?
- Tegen welke problemen liepen jullie aan?
- Hoe losten jullie die op?
- Wat zou je anders doen?

## 6. Q&A (5 minuten)

## Deliverables

1. **Werkende applicatie**
  - Kan lokaal draaien met `mvn jetty:run`
  - Minimaal de features van dag 1-4
2. **README.md**
  - Setup instructies
  - Architectuur uitleg
  - Hoe het framework te gebruiken
3. **Presentatie slides**
  - PowerPoint, Google Slides, of markdown
  - Architectuur diagrammen
  - Code snippets
4. **Source code**
  - Op GitHub of andere git repository
  - Duidelijke package structuur
  - Comments waar nodig

## Bonus uitdagingen (als je tijd over hebt)

1. **Maak een tweede demo app**
  - Bewijs dat je framework herbruikbaar is
  - Bijvoorbeeld: to-do lijst of contact form
2. **Unit tests**
  - Test je componenten
  - Test form validatie
  - Mock requests/responses
3. **Meer components**
  - Dropdown select
  - Checkboxes
  - Radio buttons
  - Date picker
4. **Template engine**
  - Externe HTML files



- Wicket-style `wicket:id` attributes
- Binding tussen HTML en Java