Blok 1 – Client Server Interaction

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

Overview

HTTP & HTML FORMS

State management

AJAX & refresh components

PRG, validation & error handling

Today's topics

HTTP Fundamentals

HTML Forms & Encoding

Request-Response Lifecycle

POST-Redirect-GET Pattern

Afternoon: Hands-on project

Learning objectives

Have a deep understanding of what happens when you click **submit.**

Implement the **HTTP fundamental concepts** in the afternoon project.

The big picture…

- HTTP foundations
- You use the functionality daily in the personal life
- Tools like Wicket rely on it under the hood
- You’re not likely to be doing this a lot in daily work, but it helps to debug and optimize

HTTP Fundamentals

What is HTTP?

HTTP = HyperText Transfer Protocal

A conversation protocol between client and server

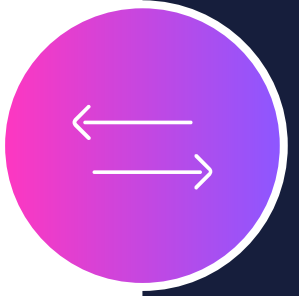
Client: "Give me the homepage"
Server: "Here's the HTML"

Request-Response model

HTTP text-based protocol

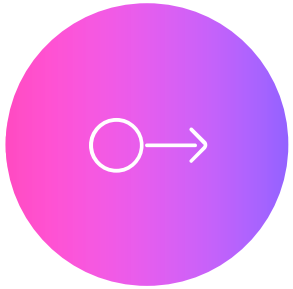
Client always initiates

Server always responds



One request = one response

HTTP is stateless



Every request is independent

Server has no memory between requests

Cookies and sessions to add state

HTTP methods – GET and POST

Aspect	GET	POST
Purpose	Retrieving data	Submitting / sending data
Data location	URL (visible)	Request body (hidden)
Bookmarkable	Yes	No
Cached	Yes	No
Idempotent (safe to repeat)	Yes	No (may cause changes)
Length limit	2000 chars	The sky

HTTP Request structure

```
POST /contact HTTP/1.1                          ← Request Line

Host: example.com                               ← Headers

Content-Type: application/x-www-form-urlencoded

Content-Length: 27

Cookie: sessionid=abc123


name=John&email=john@example.com                ← Body
```

YES! It’s just text!

Three parts of the request

Request line: Method + Path + HTTP Version

Headers: Metadata (content type, cookies, etc.)

Body: Actual data (only for POST/PUT)

HTTP Response structure

```
HTTP/1.1 200 OK                              ← Status Line

Content-Type: text/html; charset=utf-8       ← Headers

Content-Length: 1234

Set-Cookie: sessionid=xyz789


<html>...</html>                             ← Body
```

Three parts of the response

Status Line: HTTP Version + Status Code + Reason

Headers: Metadata about the response

Body: The actual content (HTML, JSON, etc.)

HTTP status codes

- 2xx Success
- 3xx Redirection
- 4xx Client error
- 5xx Server error

Exercise

See HTTP in action

HTML forms

The OG user interface.

Let’s understand forms at the HTTP level

HTML form example

```
<form action="/submit" method="POST">
    <input type="text" name="username" />
    <input type="email" name="email" />
    <button type="submit">Submit</button>
</form>
```

HTML form example

- action = Where to send the data
- method = How to send it (GET or POST)
- name - The parameter name (Without name, the field won't be submitted!)

```
<form action="/submit" method="POST">
    <input type="text" name="username" />
    <input type="email" name="email" />
    <button type="submit">Submit</button>
</form>
```

Form encoding

- Default forms are application/x-www-form-urlencoded

```
<input name="username"
value="John Doe" />
<input name="email"
value="john@example.com" />
```

Becomes :
username=John+Doe&email=john%40example.com

GET vs POST for Forms

- GET form:

```
<form action="/search" method="GET">
    <input name="q" value="java" />
</form>
```

Results in: https://example.com/search?q=java

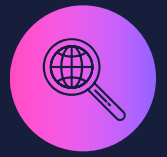
- POST form:

```
<form action="/search" method="POST">
    <input name="q" value="java" />
</form>
```

Results in: https://example.com/search (data in body)

When to use GET

Reading/retrieving data



Search queries

Filtering results

You want bookmarkable URLs

No sensitive data

Safe to repeat (idempotent)

When to use POST


Creating/modifying data


Submitting forms with sensitive data


File uploads


Large amounts of data


Actions with side effects


Should not be repeated accidentally

Exercise

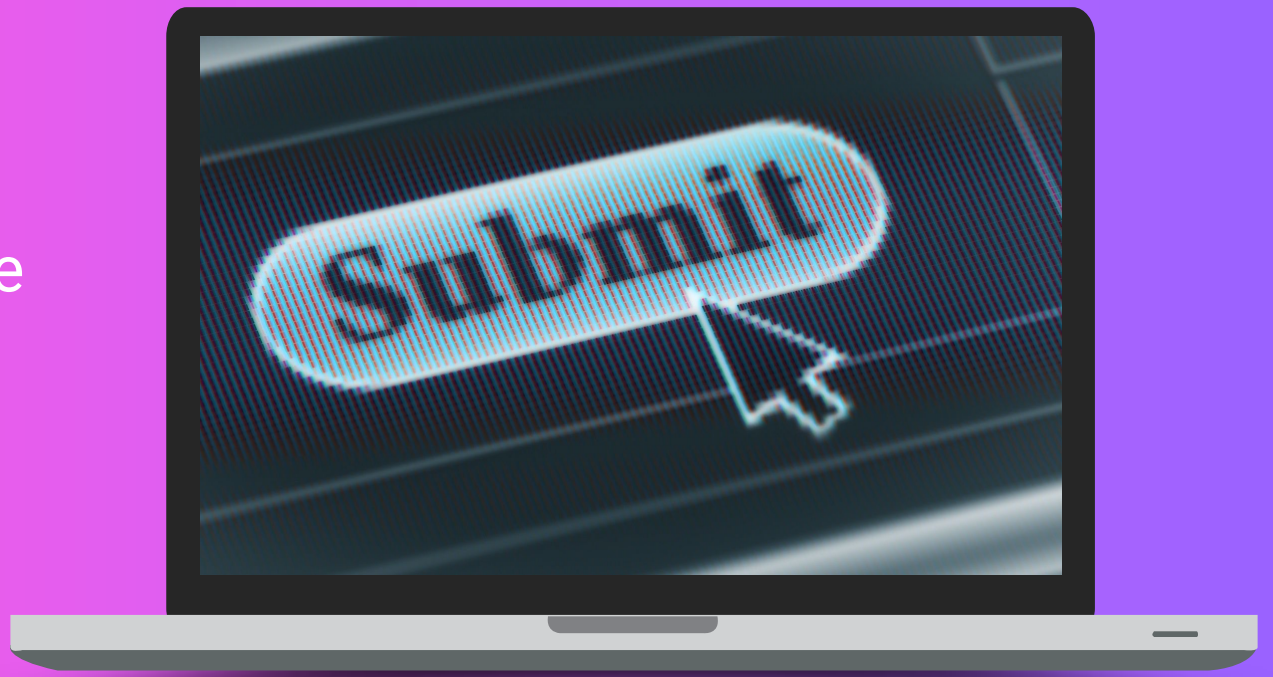
GET vs POST forms

Request-Response lifecycle

What happens when you click submit?

Step 1

User clicks the Submit button on the form



The browser then…

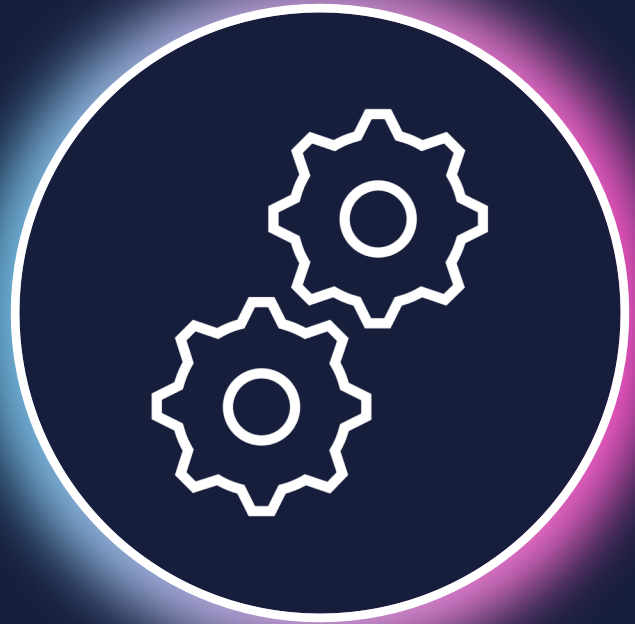
…collects form data (all fields with name).

…encodes data (URL encoding)

…sends HTTP request to server.

Next the server…

…receives and parses the request.



…processes data (validation, database, etc).

…generates response (HTML, redirect, etc).

Finally, the browser…

…receives the response.

…renders the result (new page, update DOM)

Each step can fail!

Request-Response cycle overview


```
Browser                          Server
   |                               |
   |----[1] POST /submit-------->|
   |    (form data)                |
   |                               |--[2] Parse request
   |                               |--[3] Validate data
   |                               |--[4] Process (save to DB)
   |                               |--[5] Generate HTML
   |                               |
   |<---[6] 200 OK---------------|
   |    (HTML response)            |
   |                               |
   |--[7] Render page              |
   |                               |
```

Servlets

Handling HTTP requests in Java

Servlet

- Java class that handles HTTP requests
- Generates dynamic content (e.g. HTML or JSON)
- Has access to HTTP request-response
- Basic building block of most frameworks (such as Wicket)

Exercise

Setting up servlets (incl server side multithreaded awareness)

The form resubmission problem

Safe to try this at home (you've probably seen it)

The process

1. Submit a form (POST)
2. Server processes and shows success page
3. Hit F5 (refresh)
4. Browser shows: "Confirm form resubmission"

Why does it show resubmission?

- Browser remembers it was a POST request
- Refresh = repeat the last request
- POST isn't idempotent = might cause duplicate action!

- What if it was a payment? You'd be charged twice… Oh no.

POST-Redirect-GET (PRG) Pattern

- Solution to form resubmission
- PRG pattern goes like this:
    1. POST /submit
    2. Process
    3. 303 redirect
    4. GET /success
    5. Return HTML
    6. User refreshes
    7. Just refreshes the GET

Why 303 specifically?

- 302 Found (old way): ambiguous and inconsistent across browsers
- 303 See other (correct way): it says, get this other resource and changes POST to GET
- 307 Temporary redirect: preserves method so POST stays POST
- Tools like Prometheus, Grafana, ELK stack

Exercise

See the browser warning upon resubmission

Exercise

Bonus bug

Next up:

Kaasus time!

Next up:

Session 2: State management

Questions or suggestions?

Maaike.vanputten@brightboost.nl

See you next time! 💖