

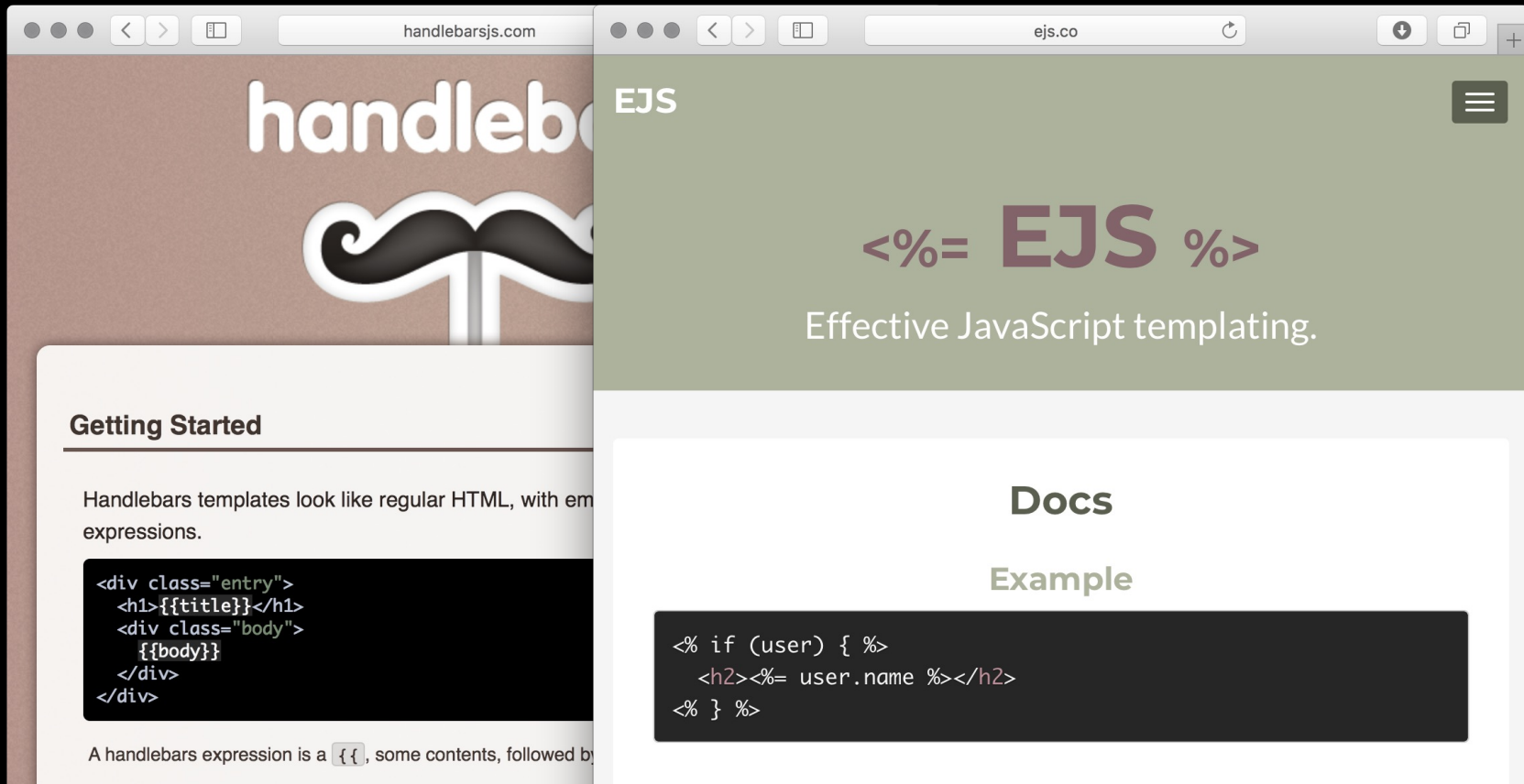
# back-end

HTTP & Forms

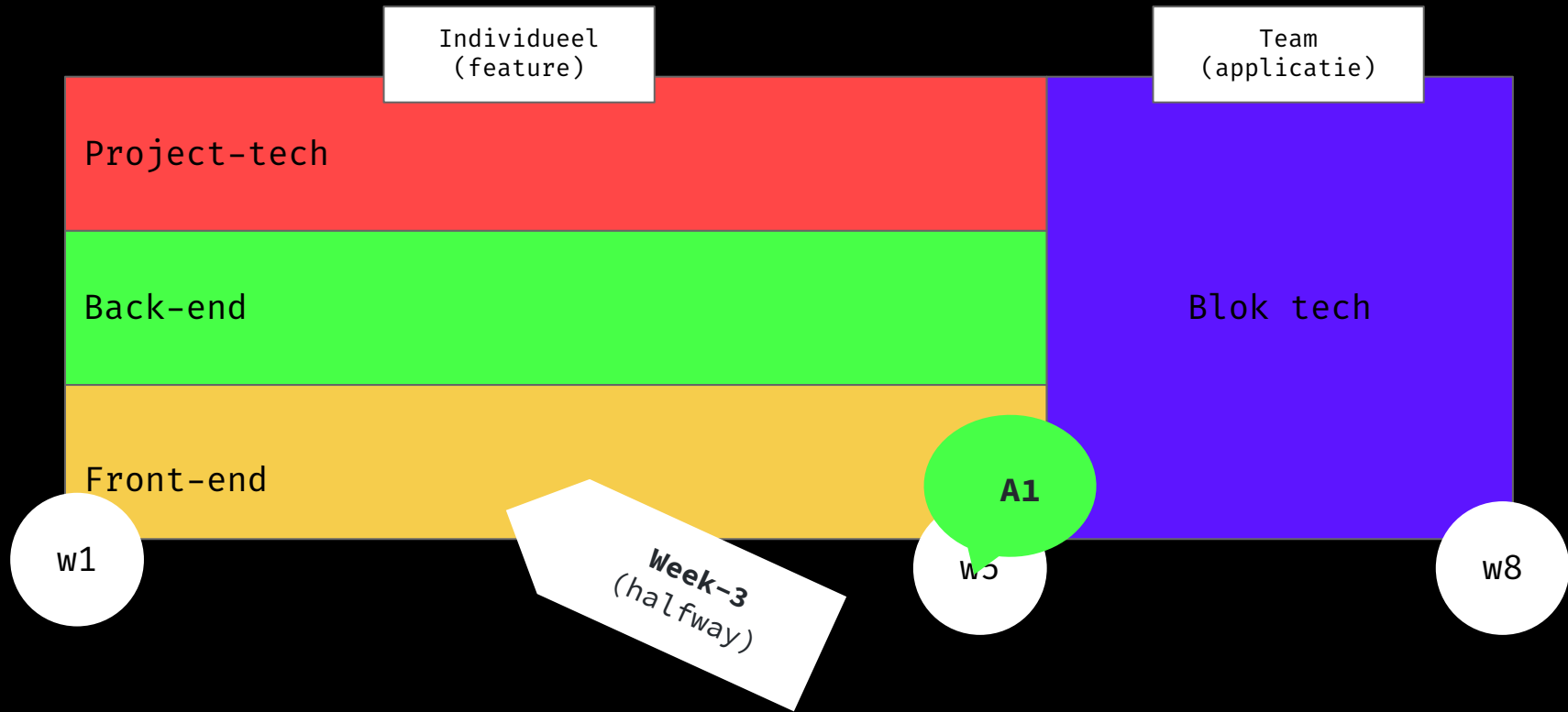
lab 3/8

Stand-up!

Show what  
you did



Live demo **templating**



# today

~~I. Stand-up~~

II. HTTP

III. Forms (+ files)

IV. Connect



HTTP

# http

?

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. [...]

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.

[wikipedia.org](https://en.wikipedia.org)

# http

url

A Uniform Resource Locator (URL) [...] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

[wikipedia.org](https://wikipedia.org)



# http

# url

protocol

subdomain

domain

port

path

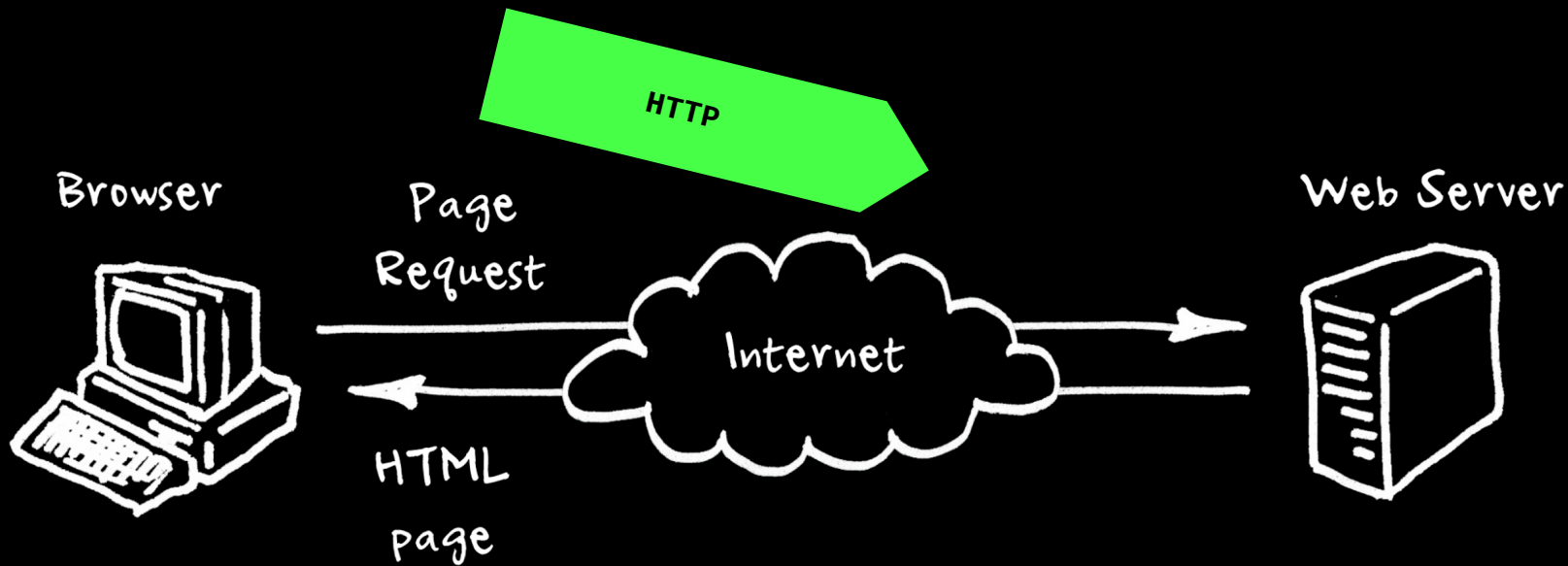
query

fragment

<http://test.example.com:3000/users/search?q=test&w=all#results>

# http

req/res



# http

response

HTTP/1.1

**status code &  
status message**

Date: Mon, 19 Feb 2018 12:00:00 GMT

Last-Modified: Tue, 13 Feb 2018 20:18:22 GMT

Content-Length: 29769

Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)

# http

## status

| Category                               | Range | Example          |
|--|-------|------------------|
| ❖ <b>Information</b><br>Protocols      | 1..   | 101 Switching    |
| ❖ <b>Success</b><br>Created            | 2..   | 200 OK, 201      |
| ❖ <b>Redirect</b><br>Permanently       | 3..   | 301 Moved        |
| ❖ <b>Client error</b><br>404 Not Found | 4..   | 400 Bad Request, |
| ❖ <b>Server Error</b><br>Server Error  | 5..   | 500 Internal     |

# http

## methods

- ❖ **Create:** PUT, POST
- ❖ **Read:** GET
- ❖ **Update:** PATCH
- ❖ **Delete:** DELETE

# http

get

```
>      /users/1 HTTP/1.1
> Host: example.com
>
< HTTP/1.1 200 OK
<
< {"id":1,"name":"Anna","age":22}
```

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The prompt is 'bash'. The command '\$ curl example.com/users/1' has been executed, and the output is a JSON object: '{"id":1,"name":"Anna","age":22}'. The prompt '\$' is visible on the next line.

```
bash
$ curl example.com/users/1

{"id":1,"name":"Anna","age":22}

$
```

Request a resource

# http

# post

```
> /users HTTP/1.1
> Host: example.com
>
> {"name":"Bisma","age":19}
>
< HTTP/1.1 201 Created
< Location: /users/2
<
< {"id":2,"name":"Bisma","age":19}
```



```
bash
$ curl example.com/users \
  → --request POST \
  → --data \
  → '{"name":"Bisma","age":19}'

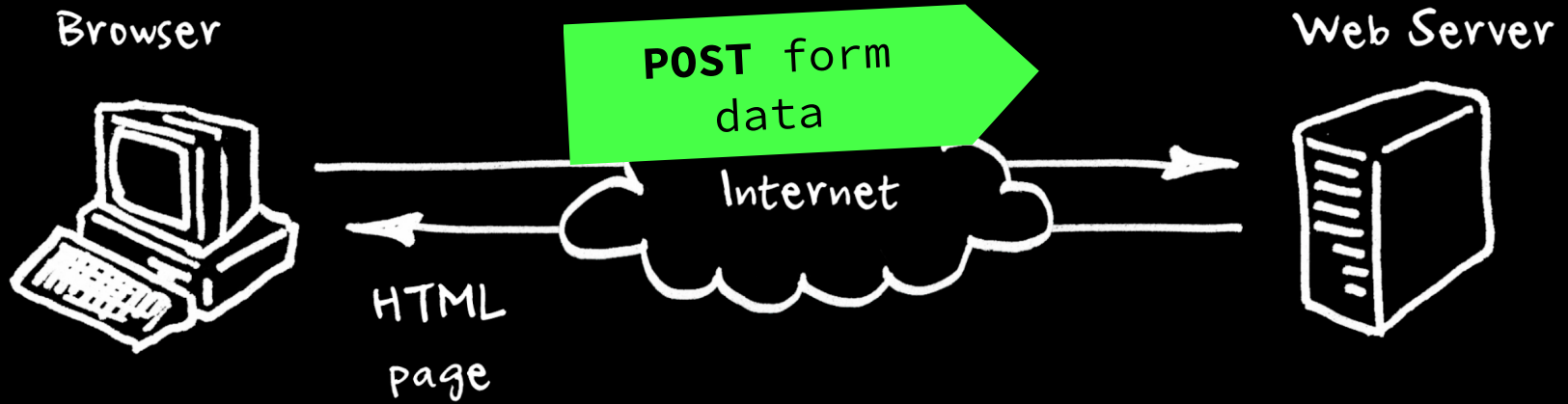
{"id":2,"name":"Bisma","age":19}

$
```

## Submit a resource

# http

get/post





# http

post

- ❖ **Text** text/plain, text/html, text/javascript, ...
- ❖ **Image** image/jpeg, image/png, ...
- ❖ **Audio** audio/webm, audio/ogg, ...
- ❖ **Video** video/webm, video/ogg, ...
- ❖ **Application** application/pdf, application/octet-stream,...

# Forms

The background is a solid black field. It is populated with numerous small, light gray geometric elements. These include triangles of various sizes and orientations, some of which are nested or overlapping. There are also several wavy, horizontal lines scattered across the canvas. The overall effect is a complex, textured pattern that resembles a stylized, abstract landscape or a microscopic view of a material.

localhost:8000/add

# Add a new movie

Title

Plot

Description

add

Add a movie

Express

view/add.ejs

```
<% include head.ejs %>
<title>Add a movie - My movie website</title>
<h1>Add a new movie</h1>
<form action=/add-movie method=post>
  <label>Title <input type=text name=title></label>
  <label>
    Plot (short)
    <input type=text name=plot>
  </label>
  <label>
    Description (long)
    <textarea
      name=description
      rows=5
    ></textarea>
  </label>
  <input type=submit value=Add>
</form>
```

The **ACTION** is the URL  
to receive the request

Send a **POST** request...

Express



index.js

```
const express = require('express')
```

```
...
```

```
express()
```

```
  .use(express.static('static'))
```

```
  .use(express.urlencoded({extended: true}))
```

```
  .set('view engine', 'ejs')
```

```
  .set('views', 'view')
```

```
  .get('/', movies)
```

```
  .get('/:id', movie)
```

```
  .get('/add', form)
```

```
  .post('/add-movie', add)
```

```
  .listen(8000)
```

```
...
```

Middleware: parses form data

Handle a **post** request to  
**/add-movie**

Express

index.js

```
express()
  .use(express.urlencoded({extended: true}))

...

function add(req, res) {
  var id = slug(req.body.title).toLowerCase()

  data.push({
    id: id,
    title: req.body.title,
    plot: req.body.plot,
    description: req.body.description
  })

  res.redirect('/') + id
}

...
```

Parsed form data is stored in  
**req.body**

Express

# form

slug

```
// Files
express-server/
├── node_modules/
├── static/
│   └── index.css
├── view/
│   ├── add.ejs
│   ├── detail.ejs
│   ├── head.ejs
│   ├── list.ejs
│   └── not-found.ejs
├── index.js
└── package.json
```

```
bash

$ npm install slug

+ slug@0.9.0
added 1 package from 1 contributor in 1.214s

$
```

slug makes a string (such as a title) URL safe.

Express

The background is a solid black field. It is populated with numerous small, light green geometric shapes and wavy lines. These elements are scattered across the entire frame, creating a textured, abstract pattern. The shapes include triangles, squares, and circles, some of which are solid, while others are outlines. The wavy lines are thin and fluid, resembling stylized waves or motion lines. The overall effect is a dynamic and modern aesthetic.

# Files



# files

```
// Files
express-server/
├─ node_modules/
├─ static/
│   ├─ index.css
│   └─ index.js
├─ view/
│   ├─ add.ejs
│   ├─ detail.ejs
│   ├─ head.ejs
│   ├─ list.ejs
│   ├─ not-found.ejs
│   └─ tail.ejs
├─ index.js
└─ package.json
```

multer

```
bash
$ npm install multer

+ multer@1.3.0
added 20 packages from 13 contributors and audited 20 packages in 3.041s

$
```

**multer is middleware for  
handling multipart/form-data**

Express

# files

# folder

```
// Files
express-server/
├─ node_modules/
├─ static/
│  ├─ index.css
│  ├─ index.js
│  └─ upload/
├─ view/
│  ├─ add.ejs
│  ├─ detail.ejs
│  ├─ head.ejs
│  ├─ list.ejs
│  ├─ not-found.ejs
│  └─ tail.ejs
├─ index.js
└─ package.json
```

We'll upload files to **static/upload**

```
bash
$ npm install multer

+ multer@1.3.0
added 20 packages in 3.041s
```

Express

view/add.ejs

```
<% include head.ejs %>
<title>Add a movie - My movie website</title>
<h1>Add a new movie</h1>
<form
  action=/add-movie
  method=post
  enctype=multipart/form-data
>
  <label>Title <input name=title></label>
  <label>
    Cover
    <input name=cover type=file accept=image/*>
  </label>
  <label>
    Plot (short)
    <input name=plot>
  </label>
  ...
  <input type=submit value=Add>
</form>
<% include tail.ejs %>
```

Accept only images

Express

index.js

```
...
const multer = require('multer')

const upload = multer({dest: 'static/upload/'})

express()
  .post('/add-movie', upload.single('cover'), add)
  ...

function add(req, res) {
  ...
  data.push({
    ...
    cover: req.file ? req.file.filename : null,
    ...
  })
}

...
```

**multer sets req.file**

Express

localhost:8000/add

# Add a new movie

Wonder Woman

Diana, an Amazonian  
warrior...

Cover **wonder-woman.jpg**

When a pilot crashes and  
tells of conflict in the  
outside world, Diana, an  
Amazonian warrior in  
training, leaves home ...

We can add files!

Express

# Expert tip 0

multer

Don't use multer's upload for forms without a file <input> – things will break!



```
// upload.single()
```

# Expert tip 1

multer

Multer will store uploaded files for us on the webserver in the folder we specify, so we don't need to worry about that. We just need to remember the filename.



```
req.file.filename
```

# recap



```
req.params
```

```
req.body
```

```
req.query
```

```
req.file
```

Question: what is the difference between ...?





Break!

# Questions:

Where will you use a form in your assignment?

What will be the 'method' in your form?

What will happen with the user input on the backend?

What will the user see after sending the form?

# input



Receive input from users on the server and manipulate that data for your own feature using HTTP request methods.

### Synopsis

- **Time:** 10:00h
- **Goals:** subgoal 4, subgoal 5, subgoal 6
- **Due:** before week 4

### Description

So far we only send data (response) to the client with our server. A one-sided conversation. Now the fun starts, it's time to actually start receiving data from users. For example; users can enter something into an input field or submit whole forms with file uploads.

The description of this assignment is quite vague since the end result will be very specific to your Job Story. Make sure you at least spend the

work on **input** and **connect**

# input



Receive input from users on the server and manipulate that data for your own feature using HTTP request methods.

### Synopsis

- **Time:** 10:00h
- **Goals:** subgoal 4, subgoal 5, subgoal 6
- **Due:** before week 4

### Description

**Note:** *Input* is quite a ‘**large**’ and ‘**vague**’ assignment since the end result will be very specific to your Job Story.



Break!

The background is a solid black field populated with numerous small, light green geometric shapes and lines. These include triangles of various sizes and orientations, wavy lines, and spiral patterns, scattered across the entire frame. The word "Connect" is centered in a bright green, sans-serif font.

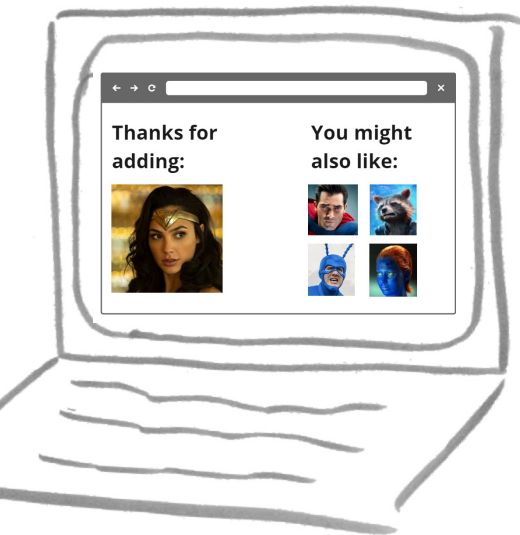
Connect

# Storing data in db

**Client**  
(browser)

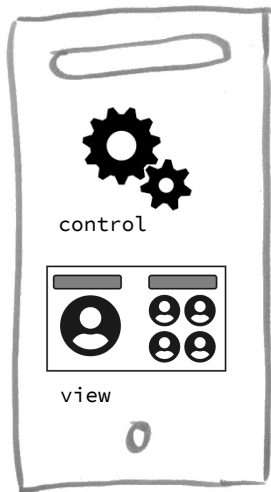
**Server**  
(Webserver met Node.js)

**Database**  
(MongoDB)



1. Browser: here's form data about a new movie

HTTP POST request



2. Server: please store this info



3. Database: I did!



4. Server: now get me some related movies

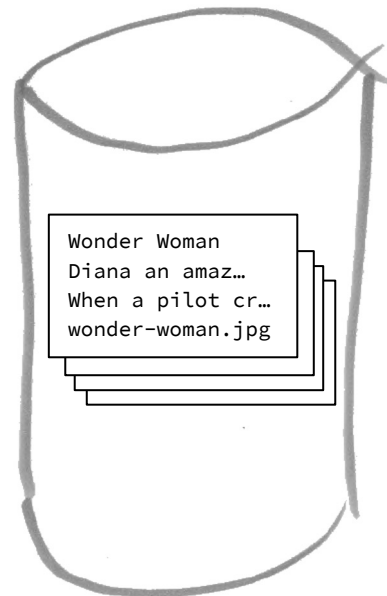


5. Database: here are the details



Server: here's a new page

6. HTTP response



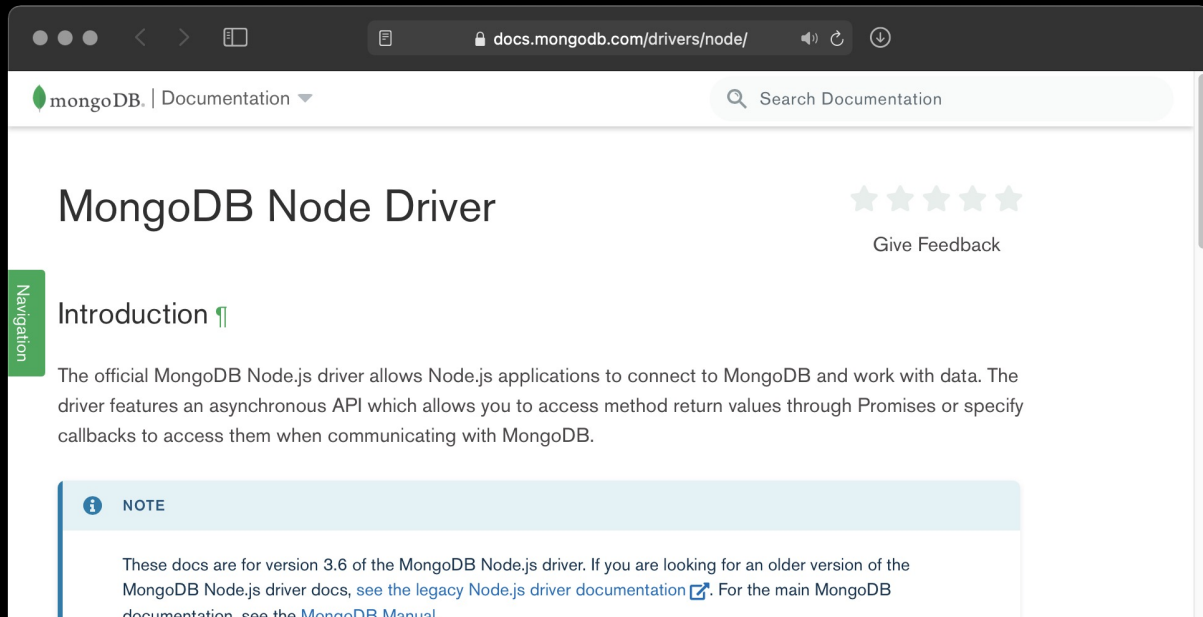
# connect

mongodb

MongoDB (from hum**ong**ous) is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. MongoDB is developed by MongoDB Inc. [...]

[wikipedia.org](https://www.wikipedia.org)





**Note:** there are a lot of small steps involved. Read the Mongo guides very carefully. If you miss a step everything will be broken.

# connect

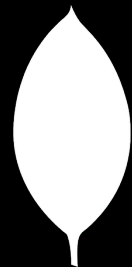
.env

```
// Files
mongodb-server/
├── node_modules/
├── static/
│   ├── index.css
│   ├── index.js
│   └── upload/
├── view/
│   ├── add.ejs
│   ├── detail.ejs
│   ├── head.ejs
│   ├── list.ejs
│   ├── not-found.ejs
│   └── tail.ejs
├── .env
├── index.js
└── package.json
```



.env

```
DB_HOST=localhost
DB_PORT=27017
DB_NAME=mymoviewebsite
DB_USERNAME=dandevri
```



# connect

.env

```
// Files
mongodb-server/
├─ node_modules/
├─ static/
│   ├─ index.css
│   ├─ index.js
│   └─ upload/
└─ view/
    ├─ add.ejs
    ├─ detail.ejs
    ├─ head.ejs
    ├─ list.ejs
    ├─ not-found.ejs
    └─ tail.ejs
```

```
.env

DB_HOST=localhost
DB_PORT=27017
DB_NAME=mymoviewebsite
DB_USERNAME=dandevri
```

**Note:** Never ever put your **host and password in code or on GitHub!** People will be able to access your database!

# connect

.gitignore

```
// Files
mongodb-server/
├── node_modules/
├── static/
│   ├── index.css
│   ├── index.js
│   └── upload/
├── view/
│   ├── add.ejs
│   ├── detail.ejs
│   ├── head.ejs
│   ├── list.ejs
│   ├── not-found.ejs
│   └── tail.ejs
└── .env

├── index.js
└── package.json
```

```
node_modules/
.DS_Store
.env
```





# exit;

see you in lab-4!