

back-end

Server & Templating

lab 2/8

Stand-up!

Show what
you did

issues

feedback

- ❖ Werk vanuit **1 repo** voor alle vakken van blok tech
- ❖ Ook geen subfolders per vak
- ❖ Dubbelcheck **je linkjes** in je issue, en **files op GH**
- ❖ Let goed op de **metainfo in package.json**
- ❖ Stop node_modules en .DS_Store **in .gitignore**
- ❖ **Index.js of server.js** staat in de root van je folder
- ❖ Schrijf in **ES6 stijl**
- ❖ **Global** versus **local** && **dep** versus **devdep**

today

~~I. Stand-up~~

II. Webserver

- Request & Response
- Express
- Routes & Static & 404

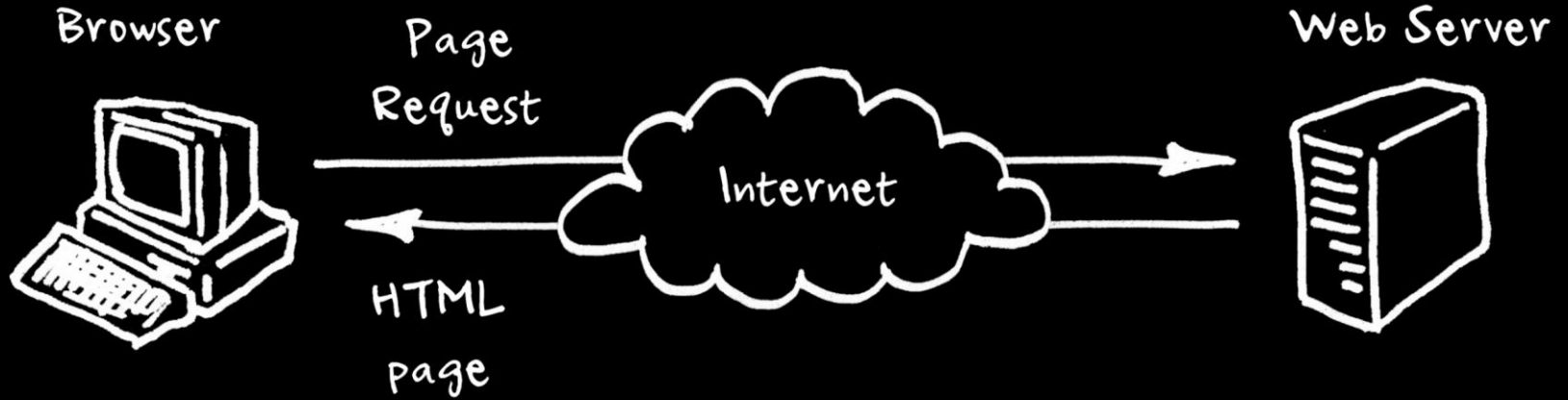
III. Templating

Webserver

Req & Res

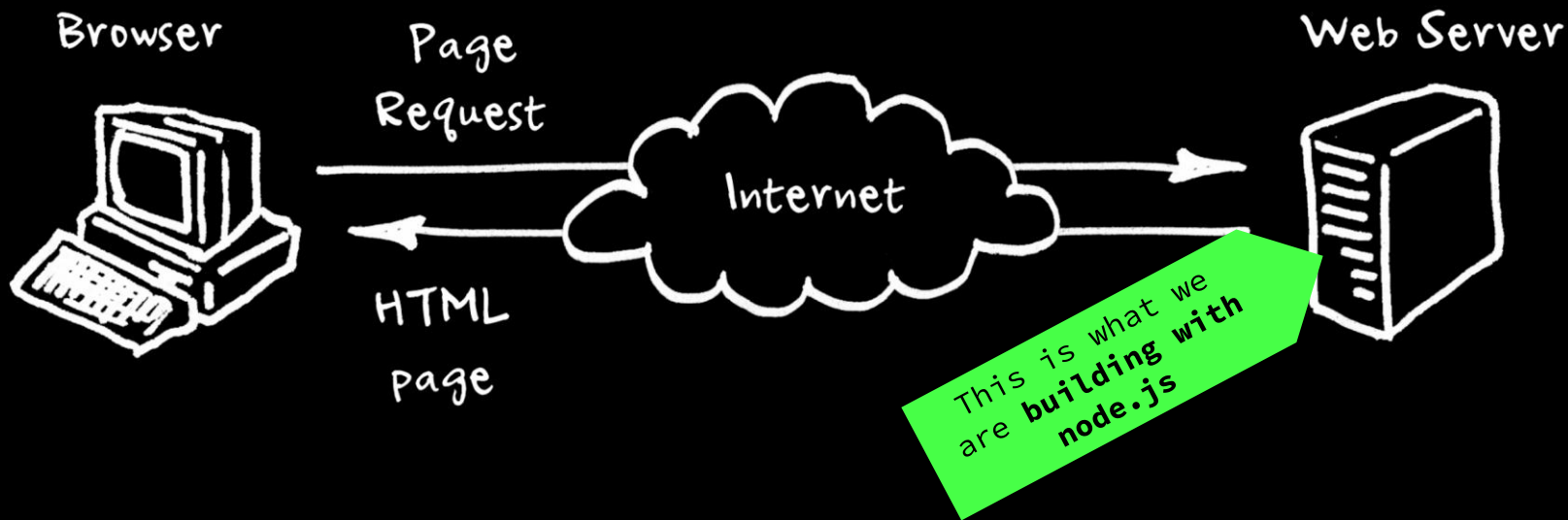
Servers

flow



Servers

flow



Servers

client/server

Client (request)

- ❖ Also known as user-agent: acts for the user
- ❖ Often a web browser
- ❖ Client asks for something: the request

Server (response)

- ❖ Connected machine waiting for requests
- ❖ Can be one machine, multiple machines, or multiple servers per machine
- ❖ Server sends something back: the response

Servers

request url



Servers

response



Servers

headers

HTTP/1.1 200 OK

Date: Mon, 19 Feb 2018 15:40:02 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)

Servers

response

HTTP/1.1 200 OK

Date: Mon, 19 Feb 2018 15:40:02 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)



body

index.js

```
var http = require('http')

http.createServer(onrequest).listen(8000)

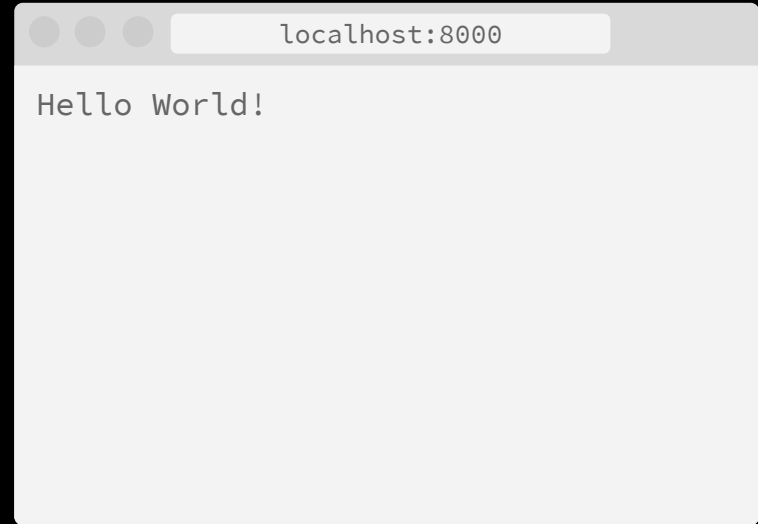
function onrequest(req, res) {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/html')
  res.send('Hello World!\n')
}
```

Handle each request by
sending **“Hello World”**



```
bash
$ node index.js
```

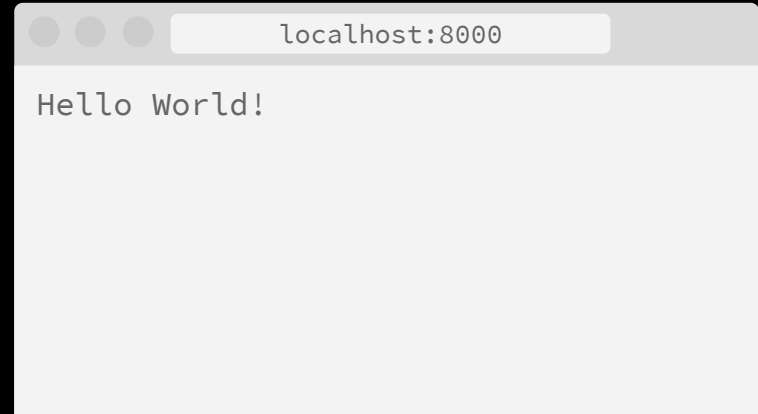
Now we have a server!





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 '\$ node index.js' has been entered. A green arrow points from the terminal towards the browser window on the right.

```
bash
$ node index.js
```



Now we have a server!

Note: this is a pretty simple server **that actually misses a lot of features:** routes, static files, mime types, streaming etc.

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

Express

expressjs.com


Express

Home Getting started Guide API reference Advanced topics Resources

Express 4.16.1

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

 Express 4.16.0 contains important security updates.
For more information on what was added in this release, see the [4.16.0 changelog](#).

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs


With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Frameworks

Many [popular frameworks](#) are based on Express.



expressjs.com

?

Express

?

- ❖ **Web Applications:** Express is a minimal and flexible Node web application framework that provides a robust set of features for web and mobile applications
- ❖ **APIs:** With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy

expressjs.com

index.js

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

express()
  .get('/', onhome)
  .listen(8000)

function onhome(req, res) {
  res.send('<h1>Hello Client</h1>\n')
}
```

Express

index.js

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

Require **express** package

```
express()
```

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

```
  .listen(8000)
```

```
function onhome(req, res) {
```

```
  res.send('<h1>Hello Client</h1>')
```

```
}
```

Express

index.js

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

express()
  .get('/', onhome)
  .listen(8000)

function onhome(req, res) {
  res.send('<h1>Hello Client</h1>')
}
```

Handle each request by
sending "Hello World"

Express

express

middleware

Middleware functions are functions that have **access to the request and the response object**. [...] Middleware can **make changes** to the request and response objects.

Express

expressjs.com

index.js

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

express()
  .get('/', onhome)
  .listen(8000)
  .use(compression)

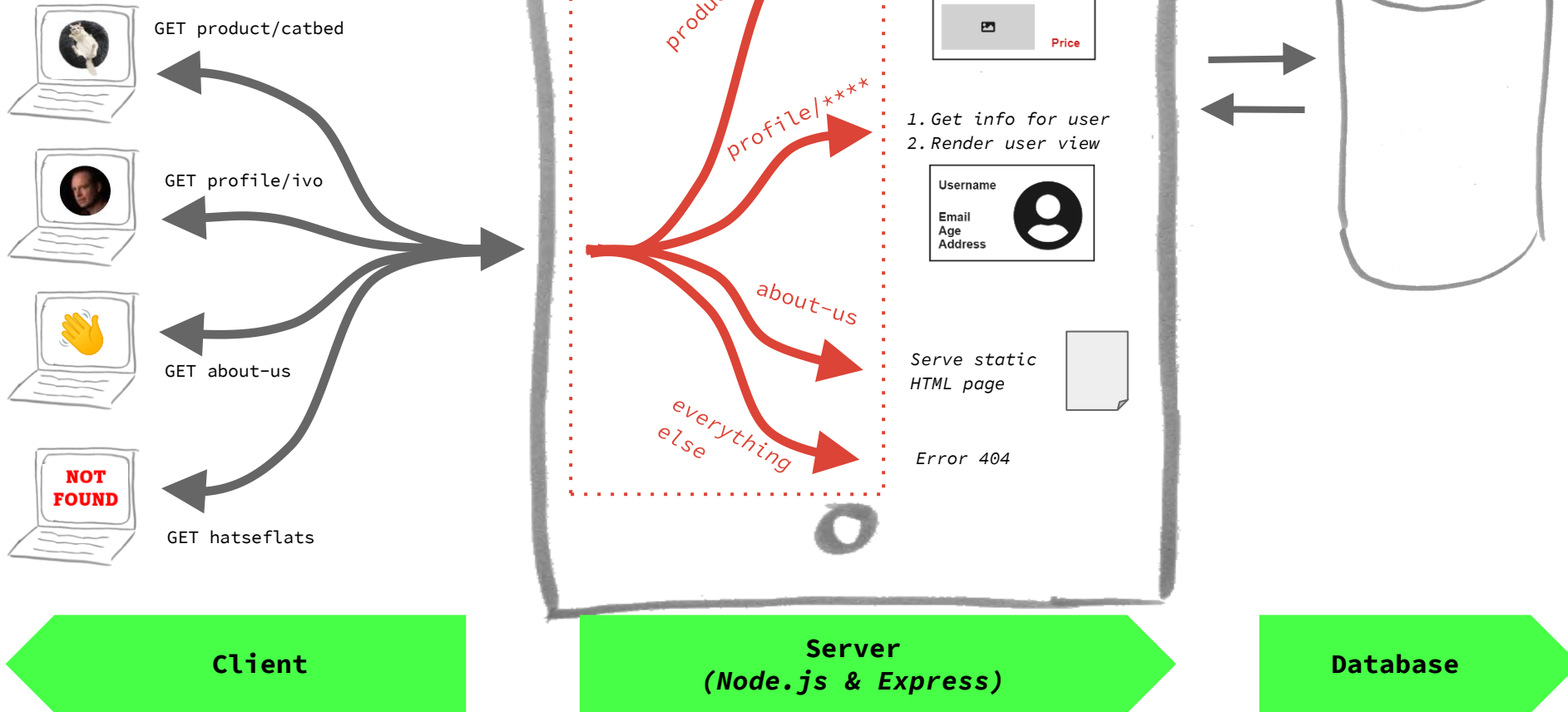
function onhome(req, res) {
  res.send('<h1>Hello Client</h1>')
}
```

For example **compression**
is middleware

Express

Routes & Static

Routes



express

routing

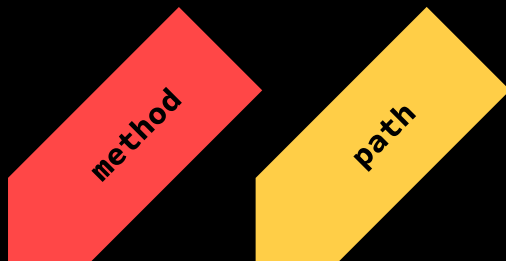
Routing refers to determining **how an application responds to a client request to a particular endpoint**, which is a URI (or path) and a specific HTTP request method (GET, POST).

Express

expressjs.com

express

methods



```
app.get(' / ', callback)
```

Express

express

method

path

methods

```
app.get('/', callback)
```

examples from imdb.com:

```
app.get('/', callback)
```

```
app.get('/conditions', callback)
```

```
app.get('/chart/top', callback)
```

```
app.get('/chart/boxoffice', callback)
```

```
app.get('/title/:movieID', callback)
```

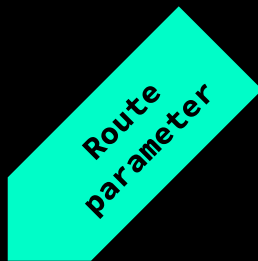
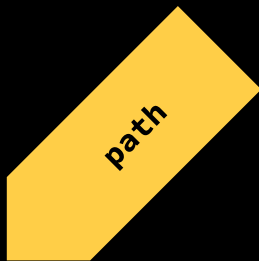
```
app.get('/title/:movieID/reviews', callback)
```

```
app.get('/name/:celebID', callback)
```

Express

express

methods



```
app.get('/title/:movieID', callback)
```

Access route parameter with

```
req.params.movieId
```

Express

index.js

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

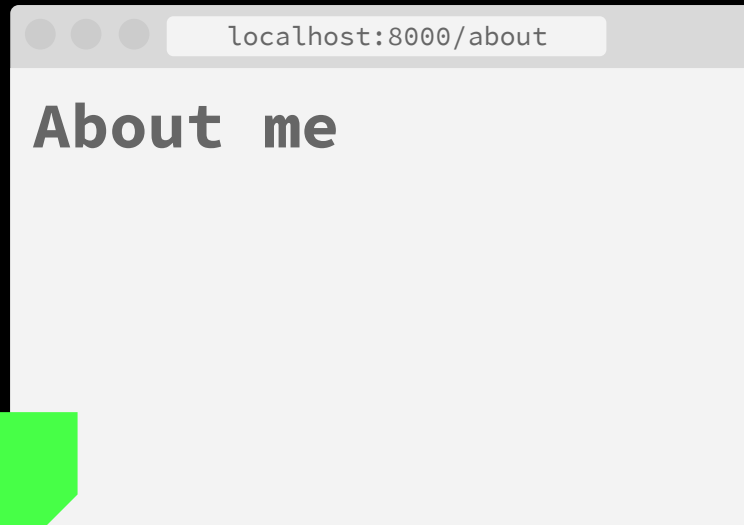
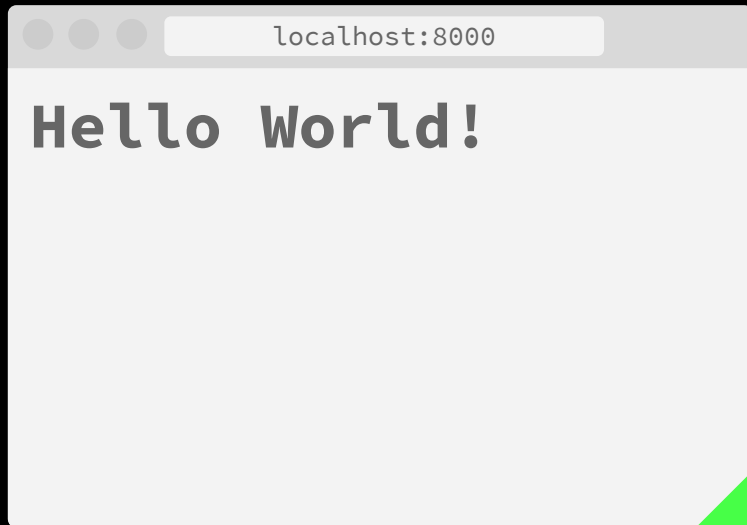
express()
  .get('/', onhome)
  .get('/about', onabout)

  .listen(8000)

function onhome(req, res) {
  res.send('<h1>Hello World!</h1>')
}

function onabout(req, res) {
  res.send('<h1>About me</h1>')
}
```

Express



We have a **route!**

express

static

Static files are files that are **not dynamically generated**. For example; .css, images, fonts. Express provides static middleware to look up files relative to the static directory.

Express

expressjs.com

express

folders

```
// Files
plain-server/
├─ index.js
├─ static/
│   └─ css          └─ style.css
│   └─ images       └─ kitten.jpg
```

express

folders

```
// Files
plain-server/
├─ index.js
├─ static/
│   ├─ css
│       └─ style.css
│   └─ images
│       └─ kitten.jpg
```

Note: everything in the **static** folder will be public!

index.js

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

express()
  .use('/static', express.static('static'))

  .get('/', onhome)
  .get('/about', onabout)

  .listen(8000)

function onhome(req, res) {
  res.send('<h1>Hello World!</h1>')
}

function onabout(req, res) {
  res.send('<h1>About me</h1>')
}
```

Use **static** middleware

Express

localhost:8000

Hello World!



404
Not Found

CSS and
Kittens!



Break!

~/Desktop

→ npm init -y

Wrote to /Users/deckard/Desktop/package.json:

```
{
  "name": "Desktop",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
  },
  "keywords": [],
  "author": "",
```

Live demo **express**

Serve



serve



In this assignment you'll build a static file server with a little help from Express.

Synopsis

- **Time:** 4:00h
- **Goals:** subgoal 3, subgoal 4
- **Due:** before week 3

Description

Create a server that handles routes and serves static files in Node.js. Use the `express` web framework. Make sure it does (atleast) the following three things:

1. **Basic routing:** Have a couple of different `routes` (e.g. `/about` `/login`) that are useful for your matching-application.
2. **Error handling:** Respond with a `404 Not Found` if you go to a route that doesn't exist.
3. **Serve static files :** such CSS but also media files such as images, video's or audio files.

work on serve



Break!

Templating

templating

static



This page is static, you type
html and you get the output.

templating

dynamic



What if you want to make them
dynamic? **Filled with user
data.**

templating

dynamic

Likes

Avatar



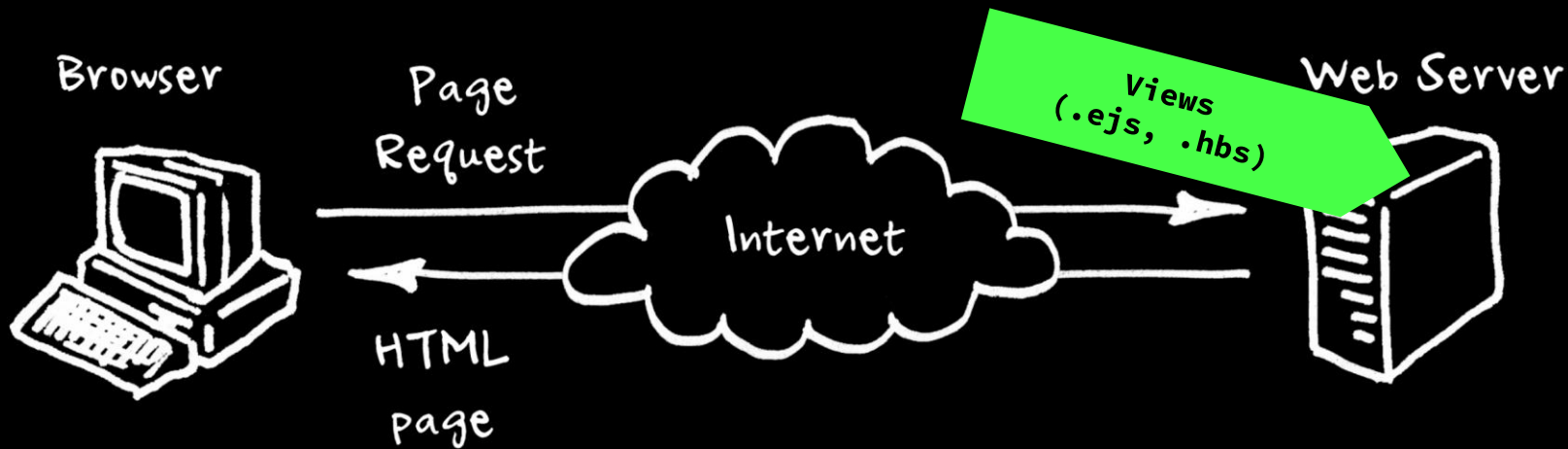
templating

?

A template engine enables you to use static template files in your application. [...] the template engine replaces variables in a **template file with actual values, and transforms the template into an HTML file sent to the client.**

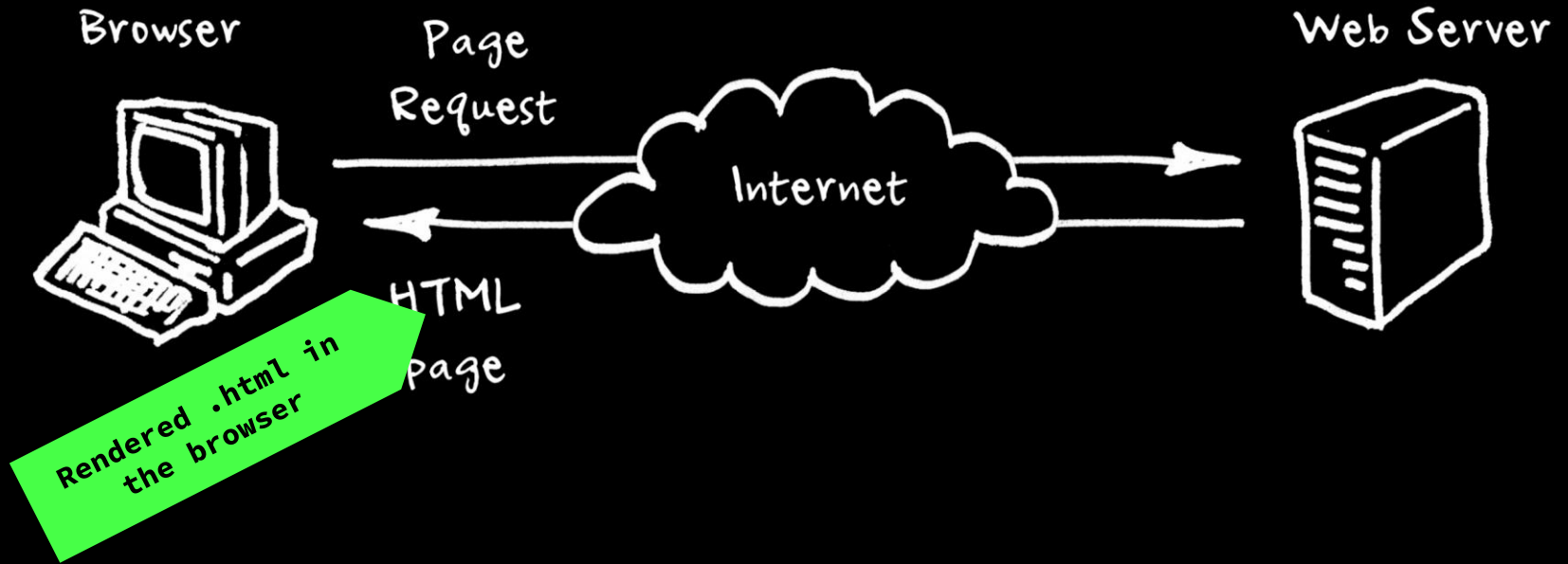
Express

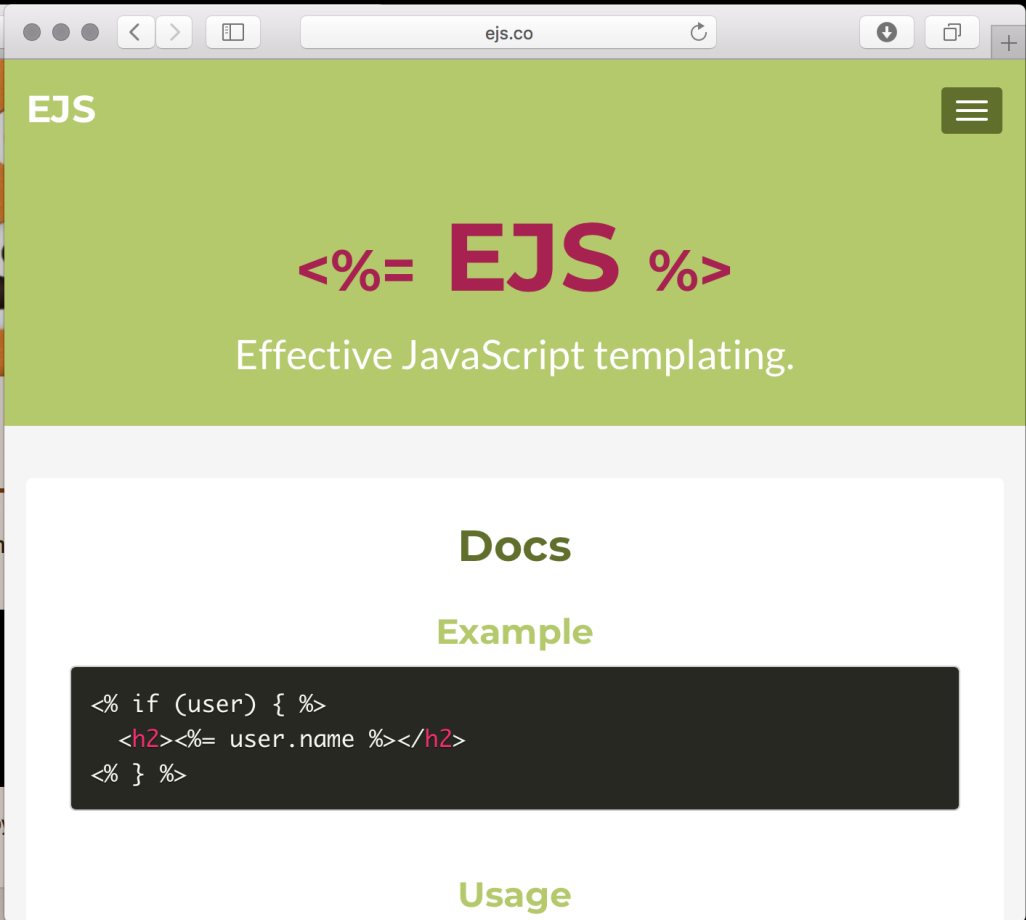
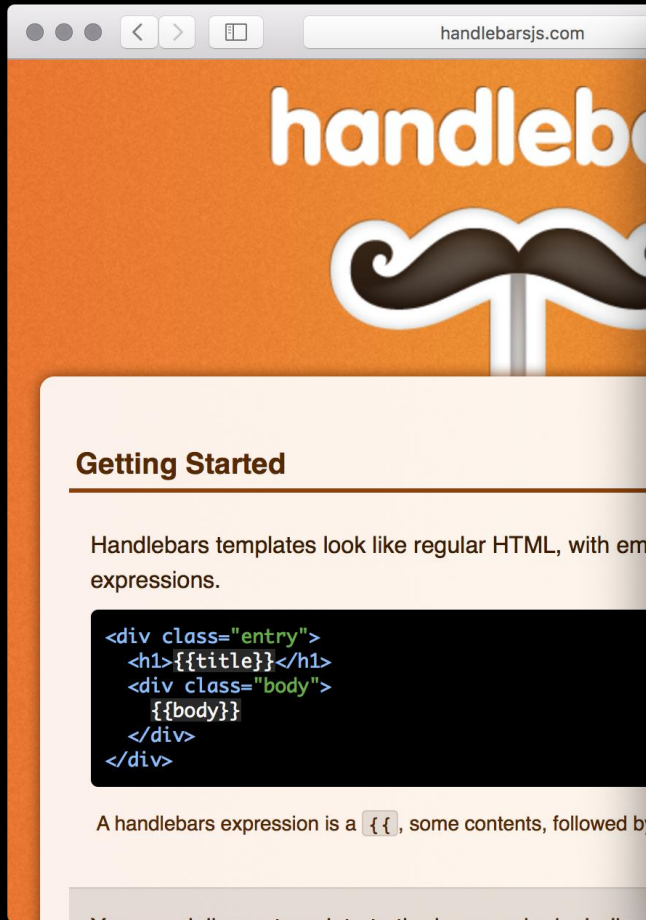
expressjs.com





Note: instead of storing .html files you store 'views' (templating files)





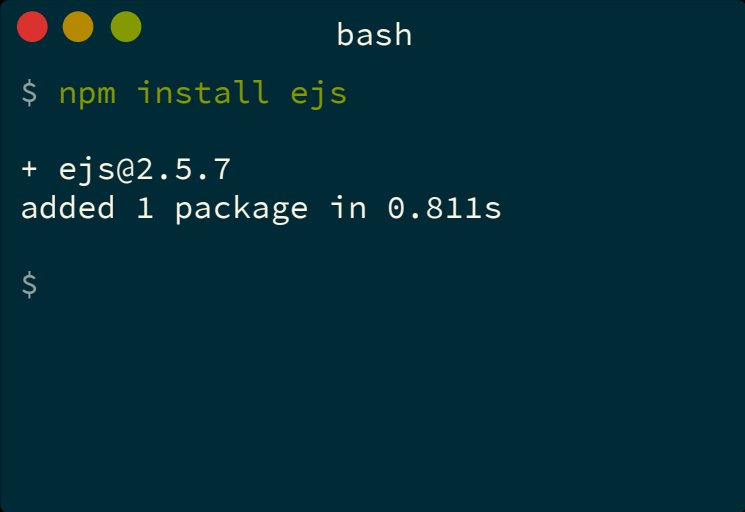
handlebarsjs.com

ejs.co

templating

views

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



```
bash
$ npm install ejs

+ ejs@2.5.7
added 1 package in 0.811s

$
```

Express

templating

```
view/not-found.ejs
<!doctype html>
<link
  rel=stylesheet
  href=/index.css
>
<title>Not found - My movie
website</title>
<h1>Not found</h1>
<p>Uh oh! We couldn't find this
page!</p>
```

```
view/not-found.ejs
<!doctype html>
<link
  rel=stylesheet
  href=/index.css
>
<title><%= data.title %> - My
movie website</title>
<h1><%= data.title %></h1>
<p><%= data.description %></p>
<p><a href="/">Back to
list</a></p>
```

<%= is JavaScript
with output

%> stops JavaScript

templating

partials

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

Express

templating

partials

view/head.ejs

```
<!doctype html>
<link
  rel=stylesheet
  href=/index.css
>
```

view/not-found.ejs

```
<% include head.ejs %>
<title>Not found - My movie
website</title>
<h1>Not found</h1>
<p>Uh oh! We couldn't find this
page!</p>
```

Express

index.js

...

```
express()
  .use(express.static('static'))
  .set('view engine', 'ejs')
  .set('views', 'view')
```

...

```
function movies(req, res) {
  res.render('list.ejs', {data: data})
}
```

```
function movie(req, res, next) {
  ...
  res.render('detail.ejs', {data: movie})
}
```

```
function notFound(req, res) {
  res.status(404).render('not-found.ejs')
}
```

Express

index.js

...

```
express()
  .use(express.static('static'))
  .set('view engine', 'ejs')
  .set('views', 'view')
```

...

```
function movies(req, res) {
  res.render('list.ejs', {data: data})
}
```

```
function movie(req, res, next) {
  ...
  res.render('detail.ejs', {data: movie})
}
```

Note: Views are **rendered** (combined with data) on the server.
The resulting HTML is sent to the client.

```
index.js

var express = require('express')

var moviesList = [
  {
    id: 'evil-dead',
    title: 'Evil Dead',
    description: 'Five friends head to a remote ...'
  },
  {
    id: 'the-shawshank-redemption',
    title: 'The Shawshank Redemption',
    description: 'Andy Dufresne is a young and ...'
  }
]

express()
  .get('/', movies)
  .listen(8000)

function movies(req, res) {
  res.render('list.ejs', {data: moviesList})
}
```

Express

view/list.ejs

```
<!doctype html>
<link rel=stylesheet href=/index.css>
<title>Movies - My movie website</title>
<h1>Movies</h1>
<% for (var index = 0; index < data.length; index++) { %>
  <h2>
    <a href="/<%= data[index].id %>">
      <%= data[index].title %>
    </a>
  </h2>
  <p><%= data[index].description %></p>
<% } %>
```

<% is JavaScript
without output

<% stops JavaScript

Express

localhost:8000

Movies

A movie

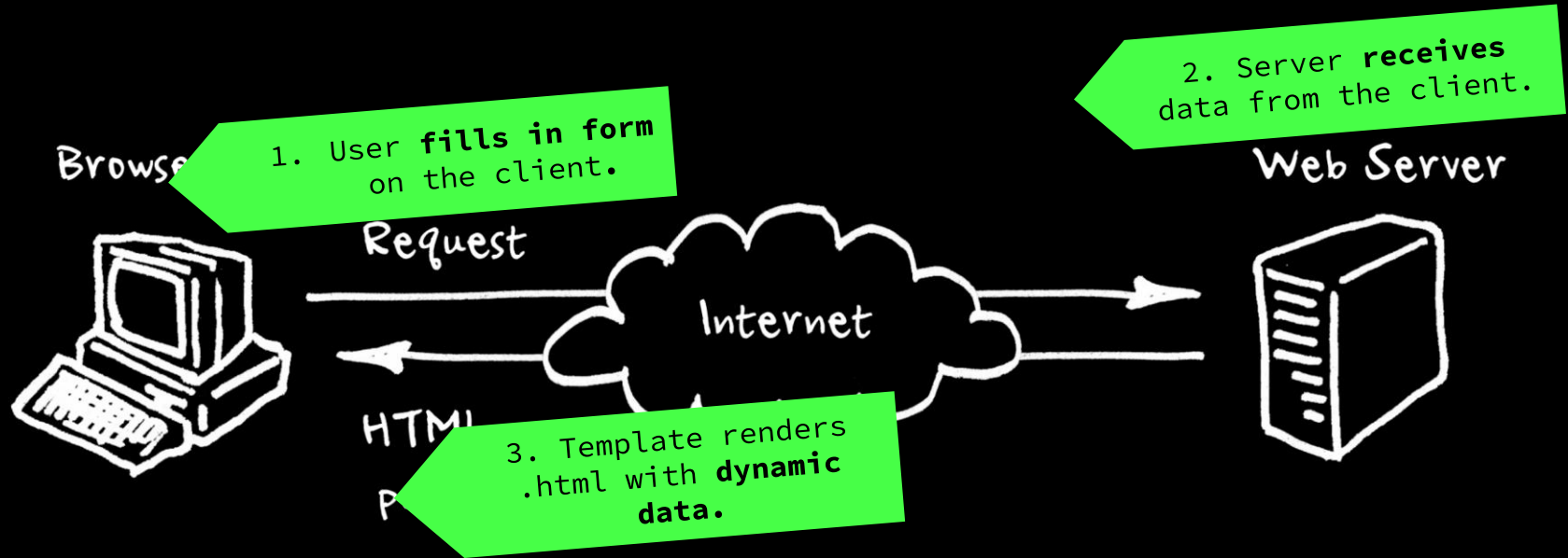
A short description

Another movie

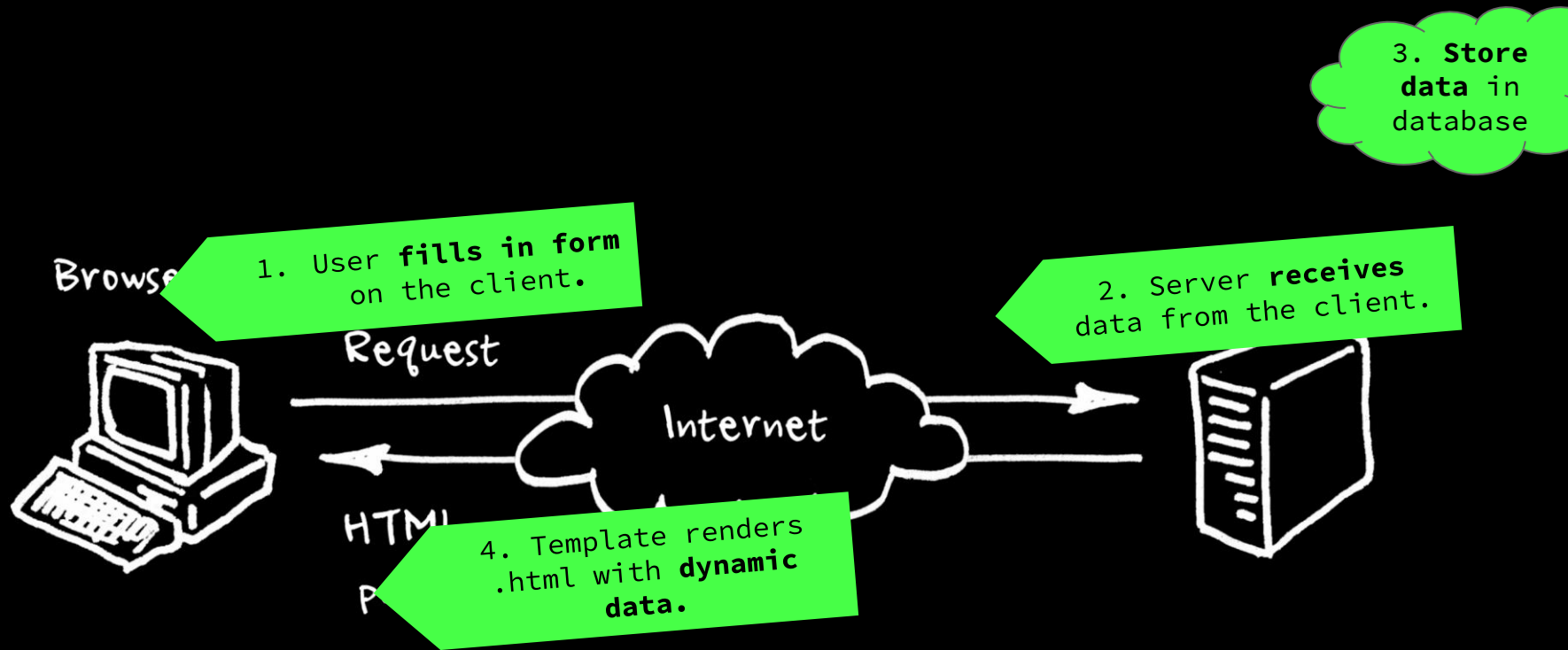
Another short description

List with movies

Express



Next up: get that data from user input.
So, for example, **a form!**



Next up: And then store that data into a **database**!

templating



Learn how to use a templating engine to dynamically render data and create components for your matching application.

Synopsis

- **Time:** 6:00h
- **Goals:** subgoal 4
- **Due:** before week 3

Description

We are slowly going to build the interface and components for your matching application. You already have a server up and running, now it's time to actually send dynamic HTML to the client using a templating engine.

1. Research different templating engines and read their documentation such as [pug](#), [ejs](#) or [handlebars](#). Document your research in your wiki. Pick one and install it in your project.
2. Then, create views and try to render a page using the templating engine. Start with 'simple' HTML pages and make sure you get

work on templating



exit;

see you in lab-3!