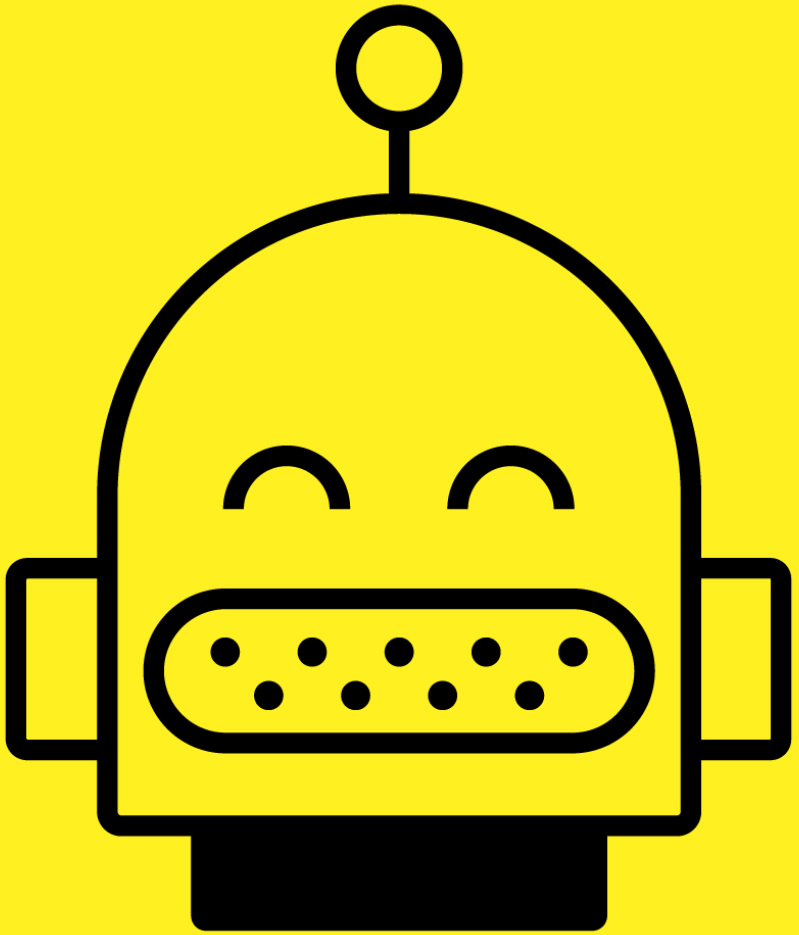
Communication and Multimedia Design | Hogeschool van Amsterdam



Project Tech

Les 2.3 verdieping

Collegejaar 23/24

Creating Tomorrow

Planning

(1)

Verdieping frontend
[zelfstandig]

ListJS

Verdieping backend

Routes & static

(2)

Pauze

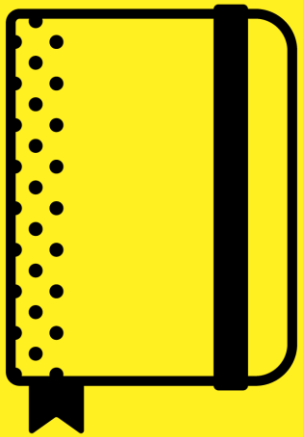
(3)

Verdieping frontend
[zelfstandig]

ListJS

Verdieping backend

Templating



Theorie

Node & Express

Node.js

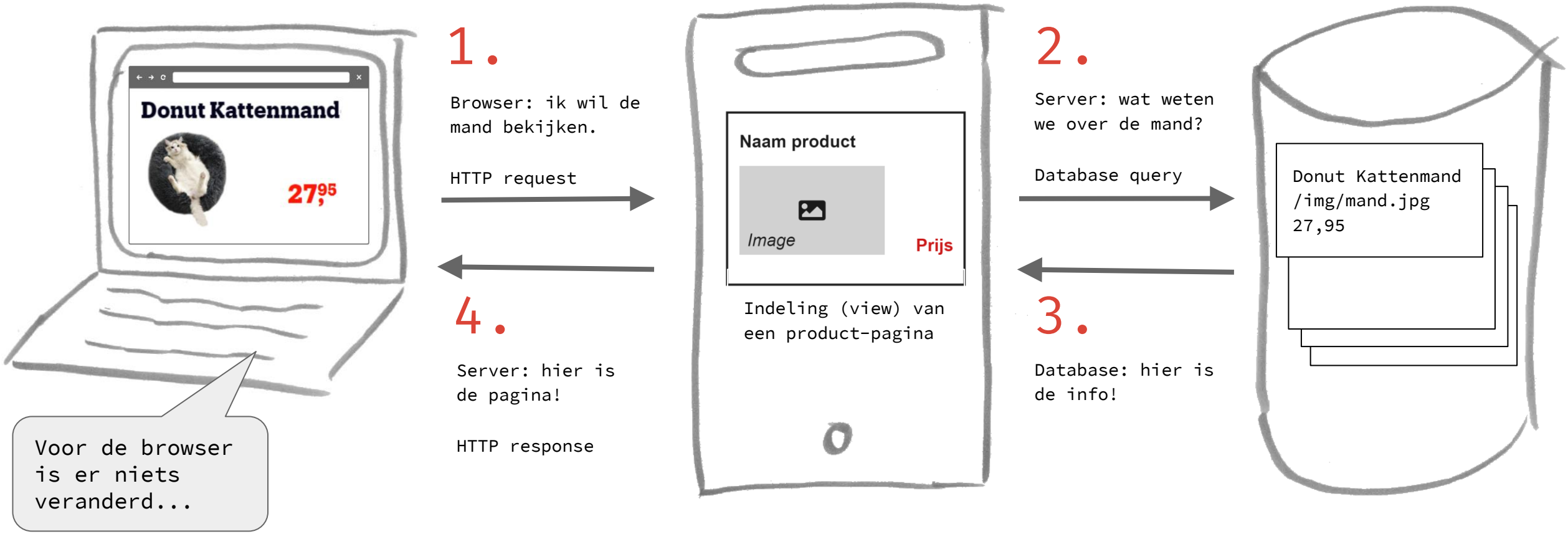
- Bij de back-end specialisatie bouwen we zelf onze eigen webserver met Node.js
- We geven in onze code aan welke HTTP requests naar bepaalde URLs door de server moeten worden afgehandeld
- En welke HTTP response de server dan moet terugsturen

Site met Node.js

Client-side / Front-end
(browser)

Server-side / Back-end
(Webserver met Node.js)

Database
(MongoDB)



Javascript in frontend vs backend

Node.js gebruikt JavaScript. Je kunt dus je bestaande code-kennis gebruiken.

Maar… deze JavaScript code draait dus op de webserver en niet in de browser!

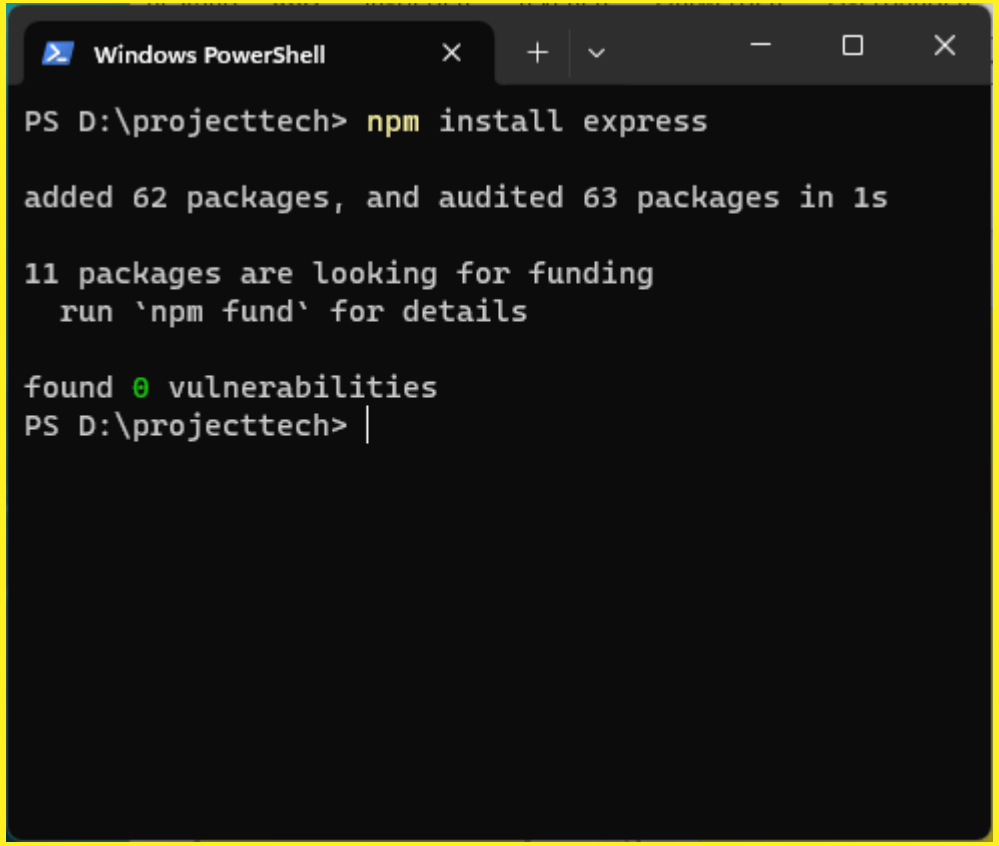
Een paar verschillen:

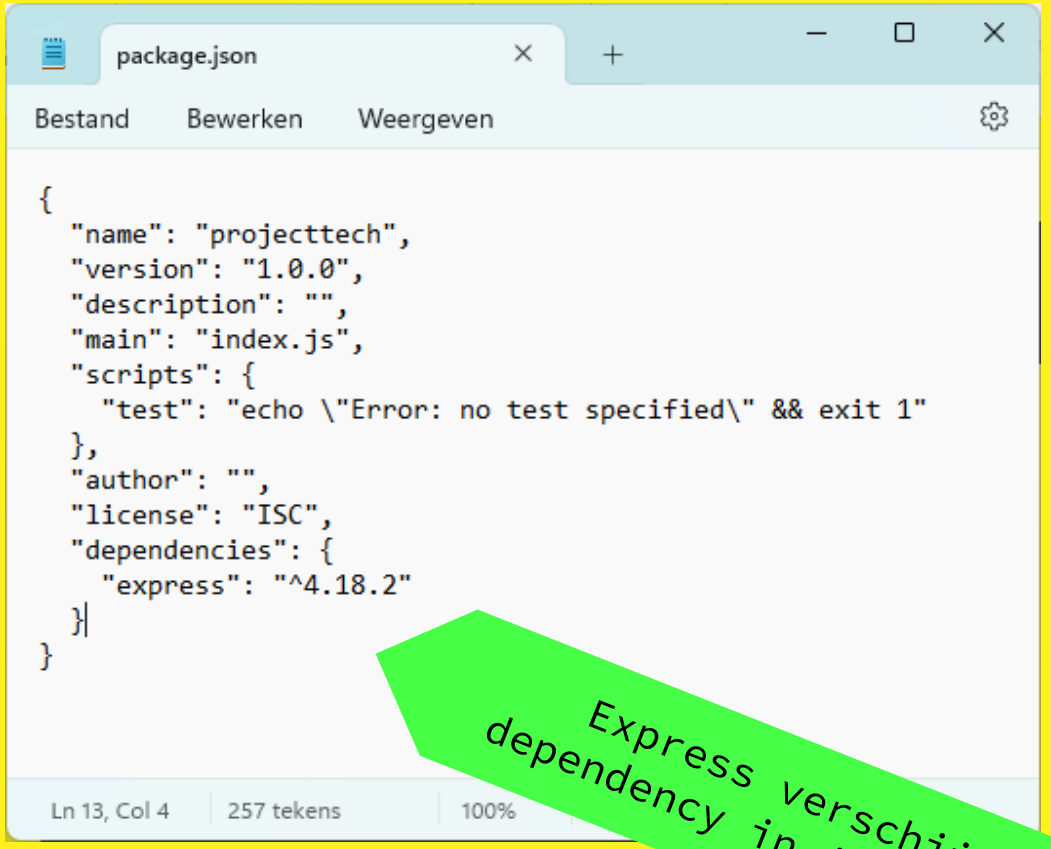
- console.log output is niet zichtbaar in de browser, maar in de terminal
- je hebt geen toegang tot de DOM (er is geen DOM op de server)
- je hebt wel toegang tot het filesystem en kunt bestanden lezen / wegschrijven

Express

- Je kunt in Node vanaf 0 je eigen webserver bouwen
- Maar dat gaan we niet doen ;)
- Herinner je je NPM?
- Express is een populaire module, waar veel functionaliteit voor een webserver al in zit

Express installeren





Express verschijnt als dependency in je package.json

server.js

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

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

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

1. Require **express** package

2. Handle each HTTP GET request by **sending "Hello World"**

3. Start webserver on port 8000

Express

Opdracht: Hello World

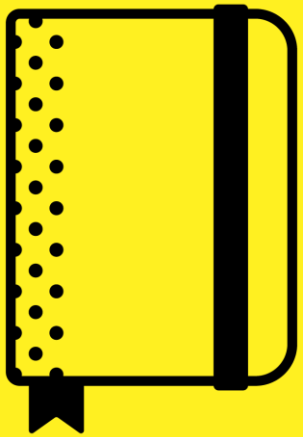
Krijg deze 1e webserver werkend op je eigen computer

1. open een terminal
2. maak een nieuwe folder voor dit node project (mkdir)
3. ga naar deze folder (cd)
4. start een nieuw node project (npm init)
5. installeer express (npm install express)
6. maak een index.js met de code van je server
7. start de server (node index.js)
8. doe vanuit je browser een http request naar deze server op de URL localhost:8000

server.js

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

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

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

Werkt het niet? Kijk eens of er wat in je terminal staat?



Theorie

Routes

Routes

GET product/catbed

GET profile/ivo

GET about-us

GET hatseflats

product/****

profile/****

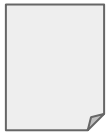
about-us

everything else

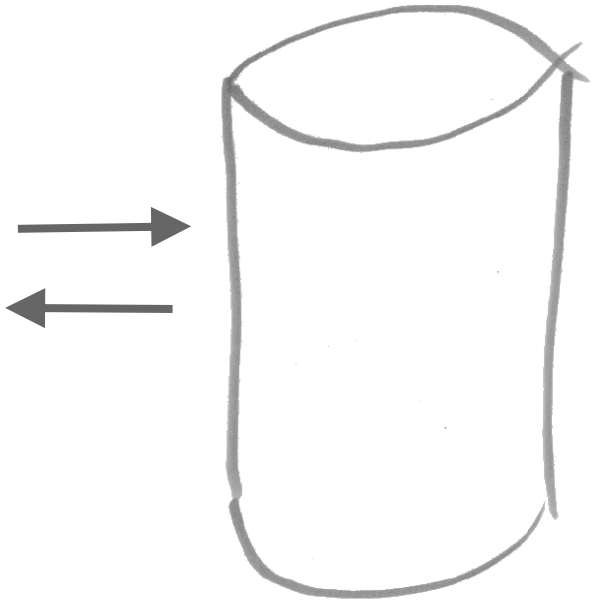
1. *Get info for product*
2. *Render product view*

1. *Get info for user*
2. *Render user view*

Serve static HTML page



Error 404



Client

Server
(Node.js & Express)

Database

express

routing

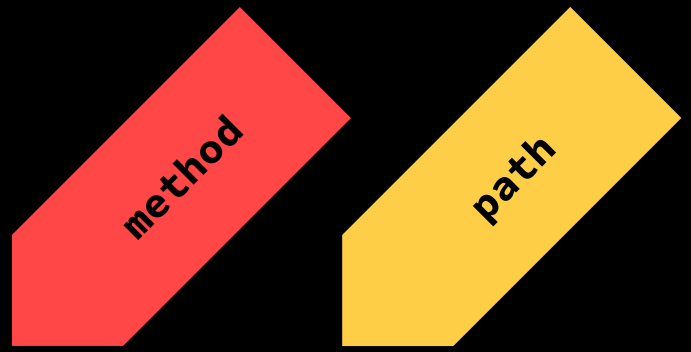
Routing refers to determining **how an application responds to a client request to a particular endpoint**, which is a URI (or path) and a specific HTTP request method (GET, POST).

Express

expressjs.com

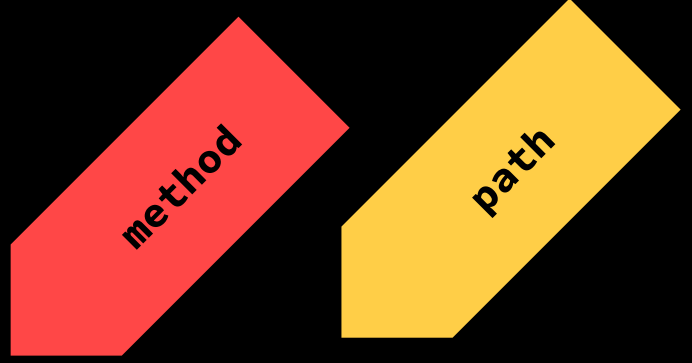
express

methods




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

express



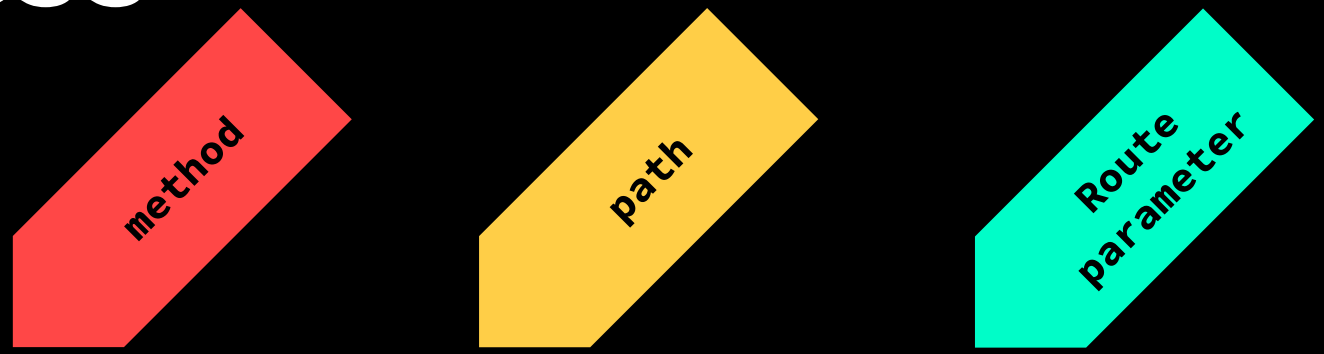
methods

```
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

methods




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

Access route parameter with
req.params.movieId

server.js

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

app
  .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>')
}
```

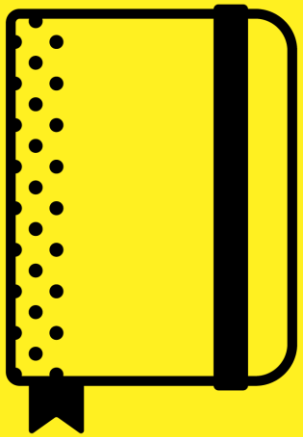
We have a **route!**

localhost:8000

Hello World!

localhost:8000/about

About me



Theorie

Static

Dynamic vc Static content

- De webserver kan HTTP requests dynamisch afhandelen. De HTML kan er anders uit zien, afhankelijk van info uit de database, voorkeuren van de gebruiker, route parameters in de URL etc.
- Dat is niet altijd nodig. Static content is altijd hetzelfde. Denk aan plaatjes, fonts, stylesheets, javascript files die in de browser zullen draaien

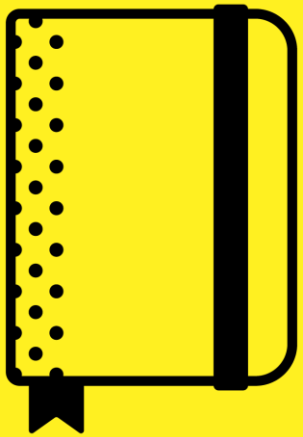
Static content

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

Maak in je project een apart mapje voor alle static content

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

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

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

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

Use **static** middleware



Theorie

Templating

Dynamische pagina’s

localhost:8000

Hello {{ username }}!

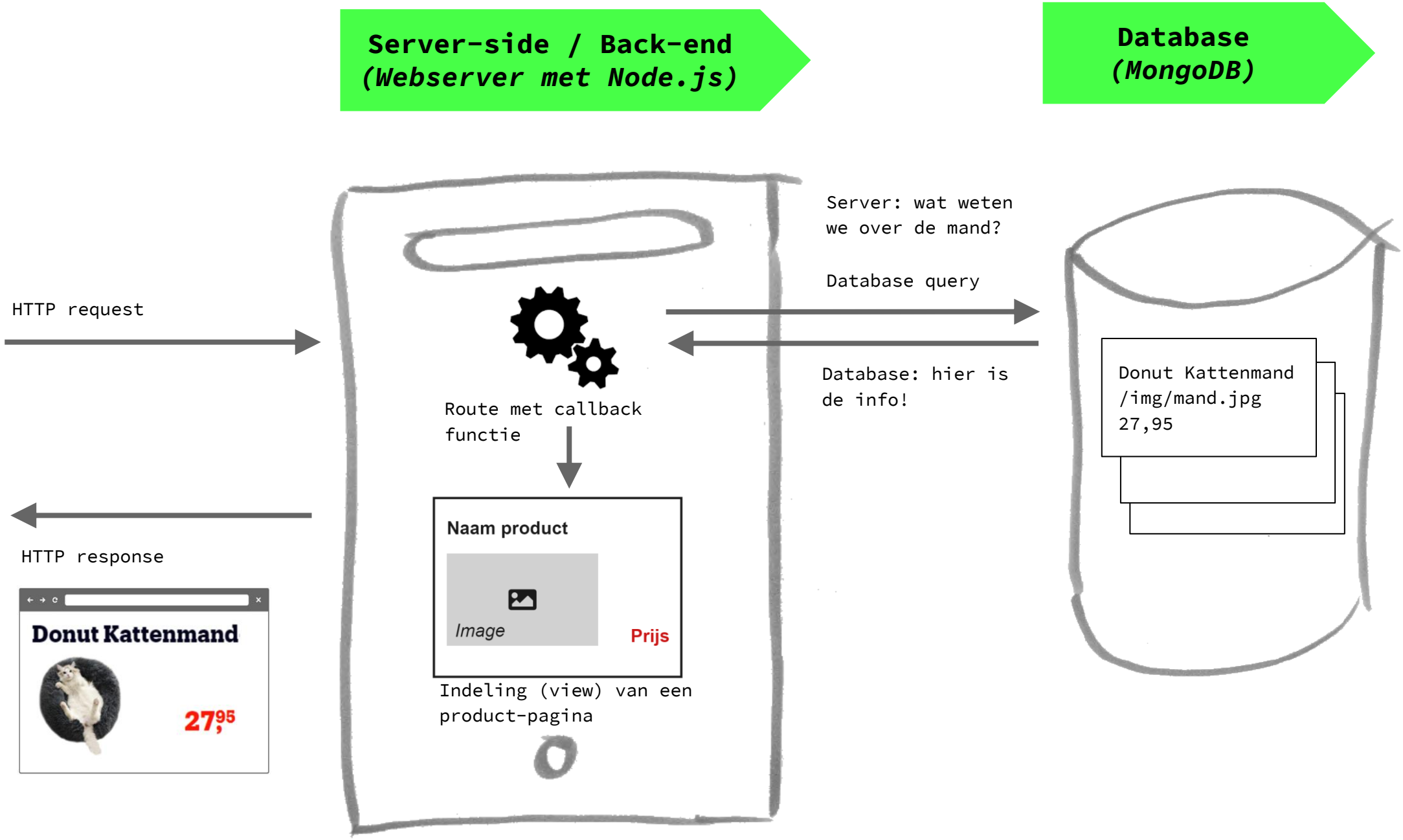
Nice to meet you!

Hoe maken we dynamische pagina’s?
Gevuld met **user data**?

Dynamische pagina’s

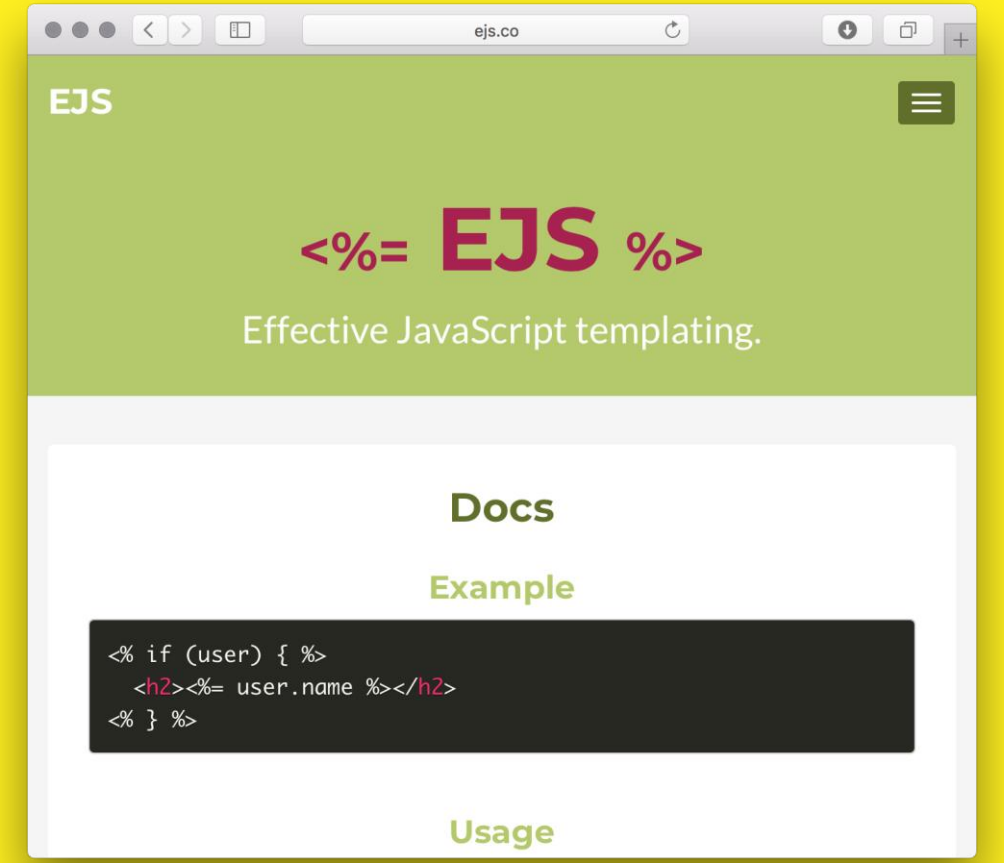
Antwoord: met routes en templates

- De route roept een callback functie aan. Deze callback functie verzamelt eerst alle nodige data
- Vervolgens wordt een *templating engine* aangeroepen. Deze gebruikt een template (indeling) van de pagina – een *view* – en vult daarin de data in om een complete HTML te maken.


Server-side / Back-end
(Webserver met Node.js)
Database
(MongoDB)
HTTP request
Server: wat weten we over de mand?
Database query
Database: hier is de info!
Donut Kattenmand
/img/mand.jpg
27,95
Route met callback functie
Naam product
Image
Prijs
Indeling (view) van een product-pagina
HTTP response
Donut Kattenmand
27,95

Templating engine

- Er zijn meerdere templating engines
- Wij gebruiken in dit project EJS



templating

views

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

Maak in je project een apart mapje voor je views

bash

```
$ npm install ejs

+ ejs@2.5.7
added 1 package in 0.811s

$
```

Installeer EJS

templating

views

view/detail.ejs

```
<!doctype html>

<title><%= data.title %> - Movie Site</title>

<h1><%= data.title %></h1>
<p><%= data.description %></p>
<p><a href="/">Back to list</a></p>
```

Een view bestaat uit HTML met stukjes template code ertussen

De dynamische code begint met <% en eindigt met %>

In dit geval vullen we dynamisch de waardes van een aantal variabelen in

templating

partials

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

view/detail.ejs

```
<!doctype html>

<title><%= data.title %> - Movie Site</title>

<%- include('menu.ejs') %>

<h1><%= data.title %></h1>
<p><%= data.description %></p>
<p><a href="/">Back to list</a></p>
```

met partials kun je stukjes code herbruiken

Elke page heeft het zelfde menu voor de navigatie

server.js

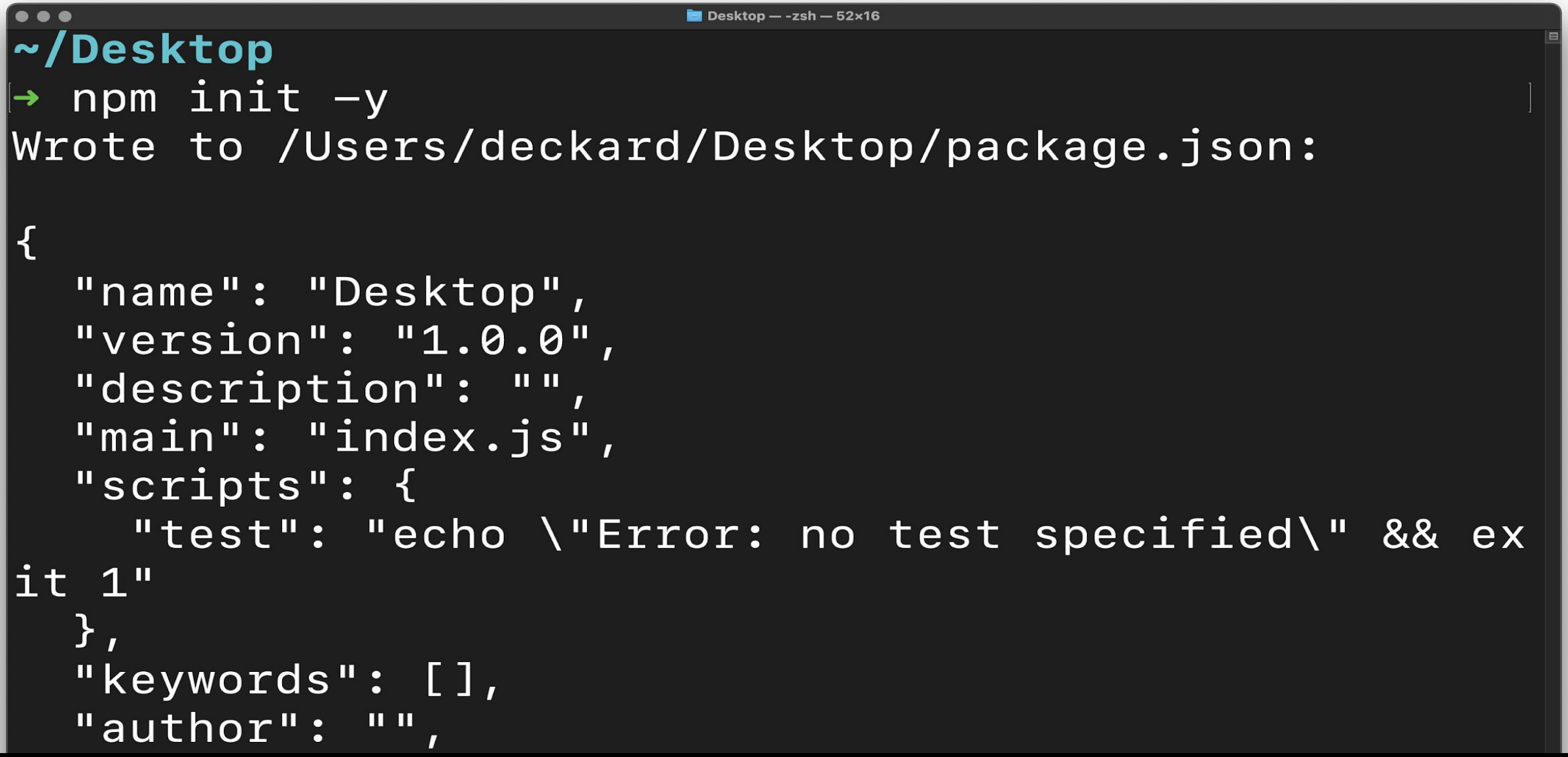
```
app
  .set('view engine', 'ejs')
  .set('views', 'view')

function movie(req, res, next) {
  let movie = {
    title: 'The Shawshank Redemption',
    description: 'Andy Dufresne is a young and …'
  }

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

Configureer EJS

Callback functie verzamelt data

En roept templating engine aan

- **detail.ejs** is de te renderen view
- **movie** is de variabele die wordt meegegeven aan de view
- deze komt nu in de view beschikbaar als de variabele **data**, dus in de view kunnen we **data.title** en **data.description** gebruiken



Live demo **express**