

back-end

Server & Templating

lab 2/8



Show what
you did

Stand-up!

today

I. ~~Stand-up~~

II. Request & Response

III. Express

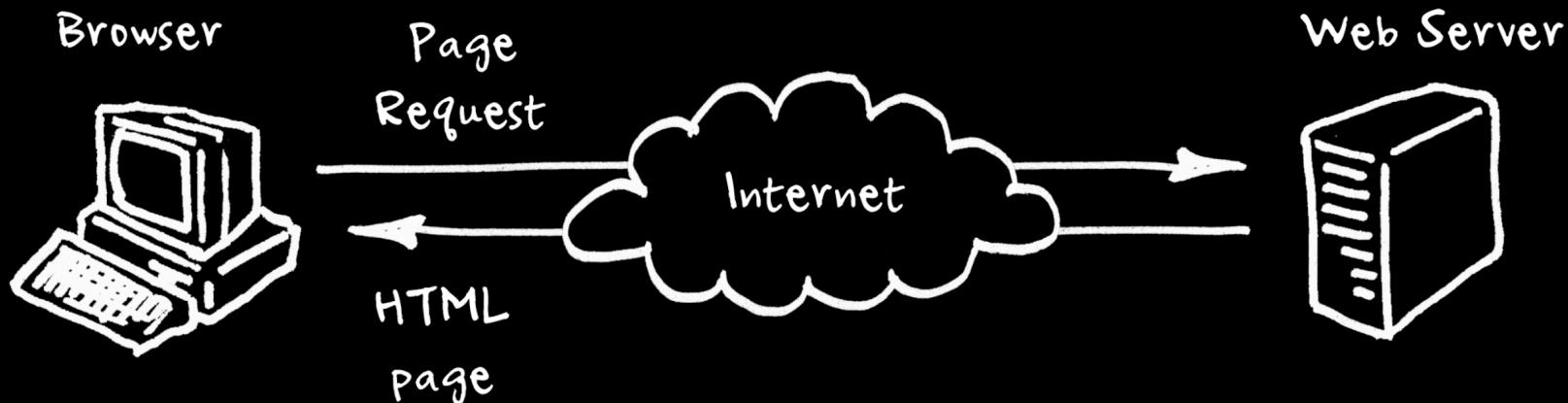
IV. Routes & static

V. Templating

Req & Res

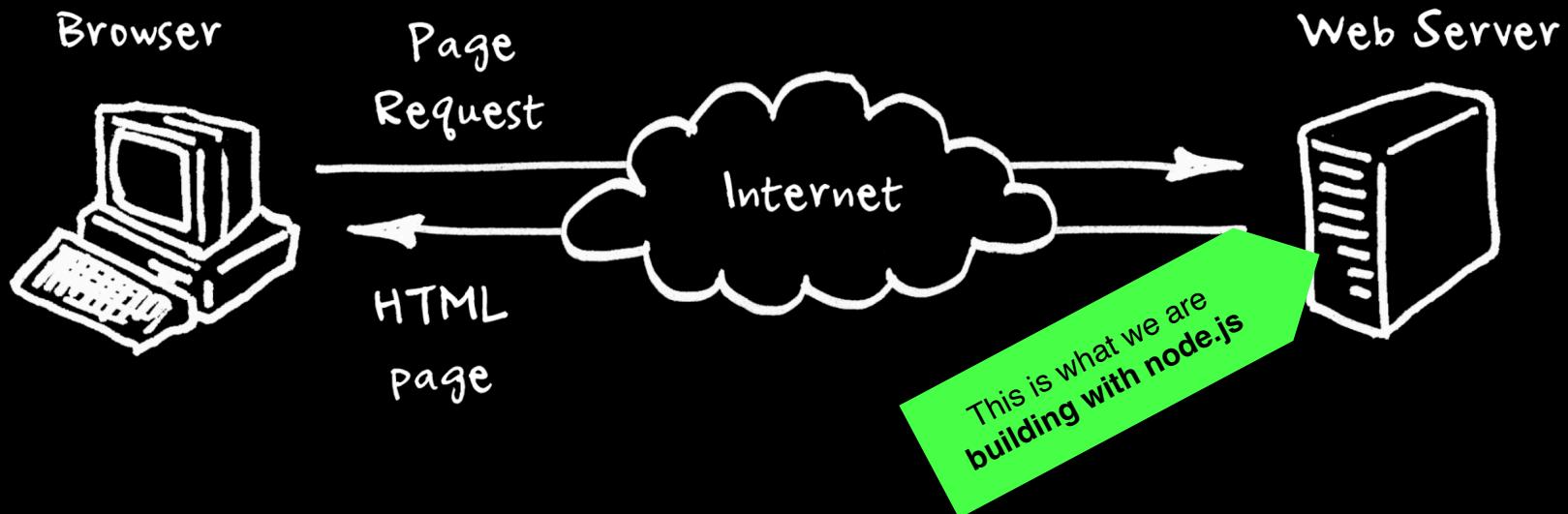
Servers

flow



Servers

flow



Servers

client/server

Client (request)

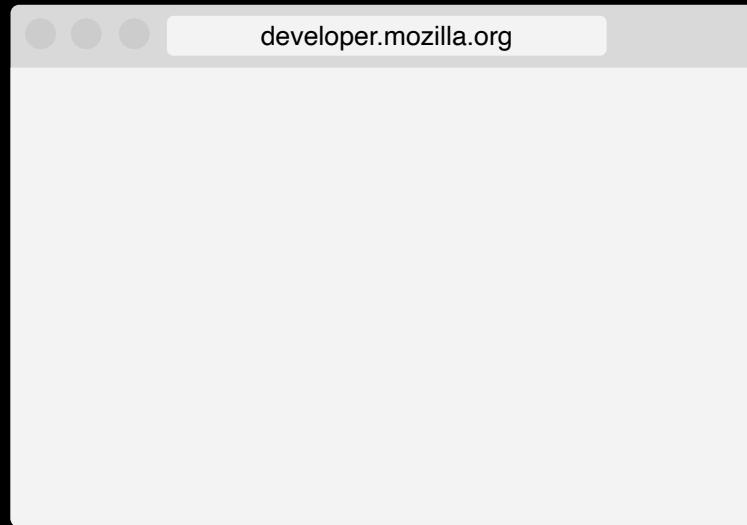
- ❖ Also know as user-agent: acts for the user
- ❖ Often a web browser
- ❖ Client ask for something: the request

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

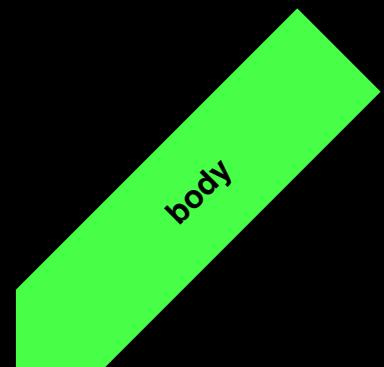
HTTP/1.1 200 OK

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)



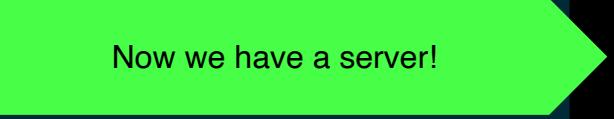
index.js

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

http.createServer(onrequest).listen(8000)

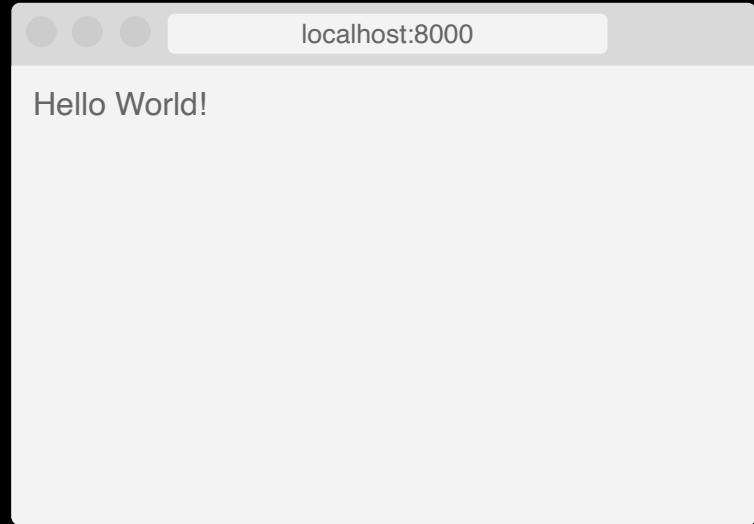
function onrequest(req, res) {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/html')
  res.end('Hello World!\n')
}
```

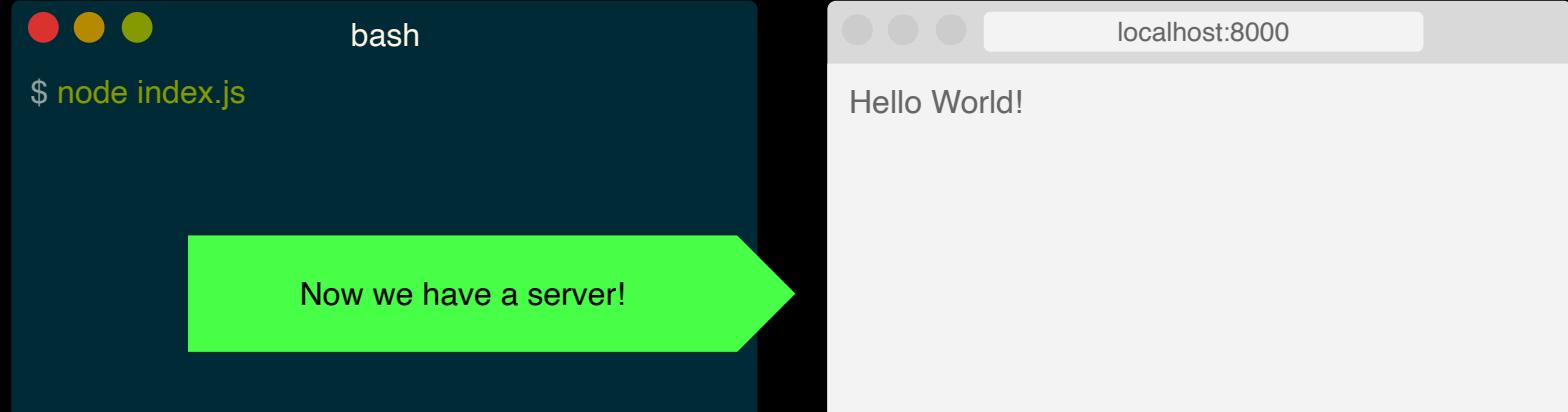
Handle each request by sending
“Hello World”



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

Now we have a server!





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.

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](#).

KEYNOTE: Express, State of the Union by D... Austin, Texas November 20–December 2, 2016

KEYNOTE: Express, State of the Union
Doug Wilson, Express

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(1900)

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

Express



index.js

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

express()
  .get('/', onhome)
  .listen(1900)

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

Require **express** package

Express



index.js

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

express()
  .get('/', onhome)
  .listen(1900)

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



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(1900)
  .use(compression)

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

For example **compression** is middleware

Express

Routes & Static

express

routing

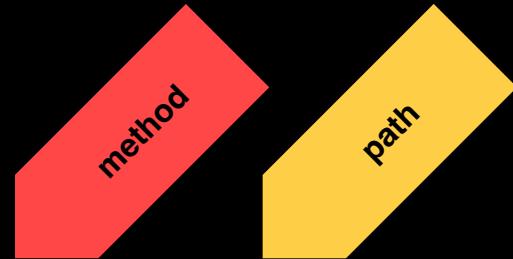
Routing refers to determining **how an application responds to a client request to a particular endpoint**, which is a URI (or path) and a specific HTTP request method (GET, POST).

Express

expressjs.com

express

methods



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

Express



index.js

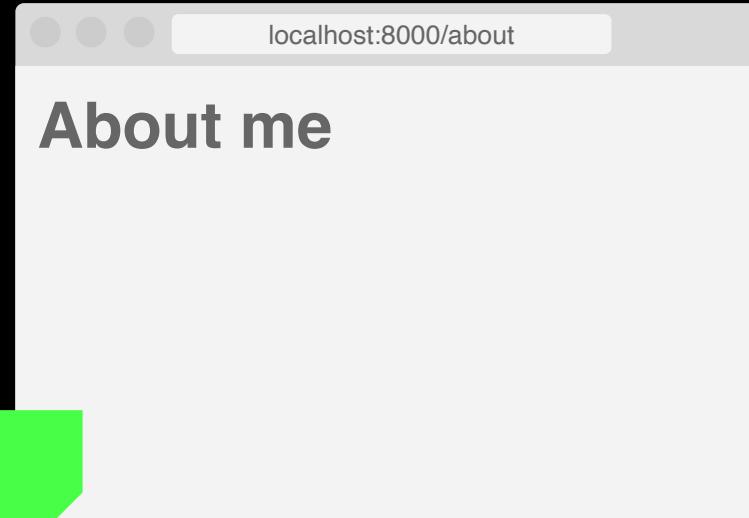
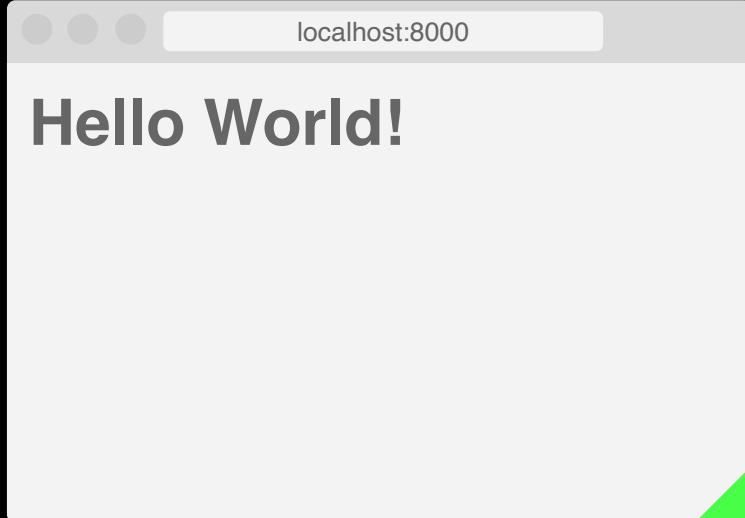
```
const express = require('express')

express()
  .get('/', onhome)
  .get('/about', onabout)
  .listen(1900)

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

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

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(1900)

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

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

Use **static** middleware

Express

localhost:8000

Hello World!



404

Not Found

CSS and
Kittens!

~/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**

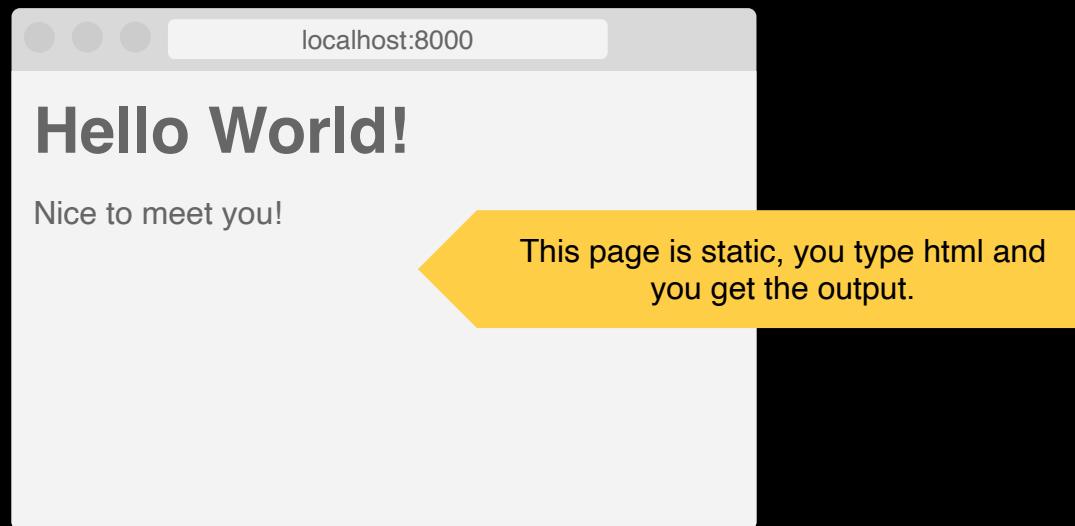


Break!

Templating

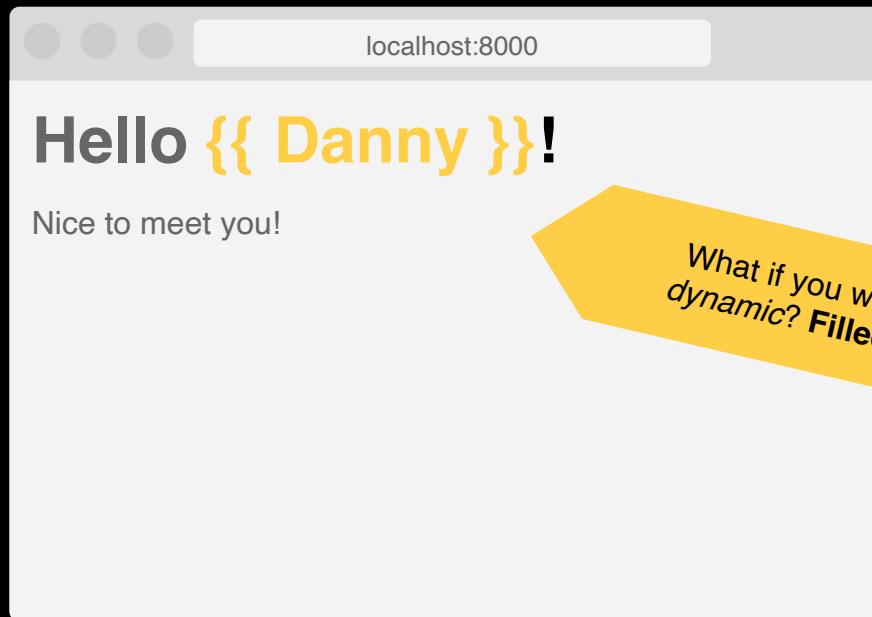
templating

static



templating

dynamic



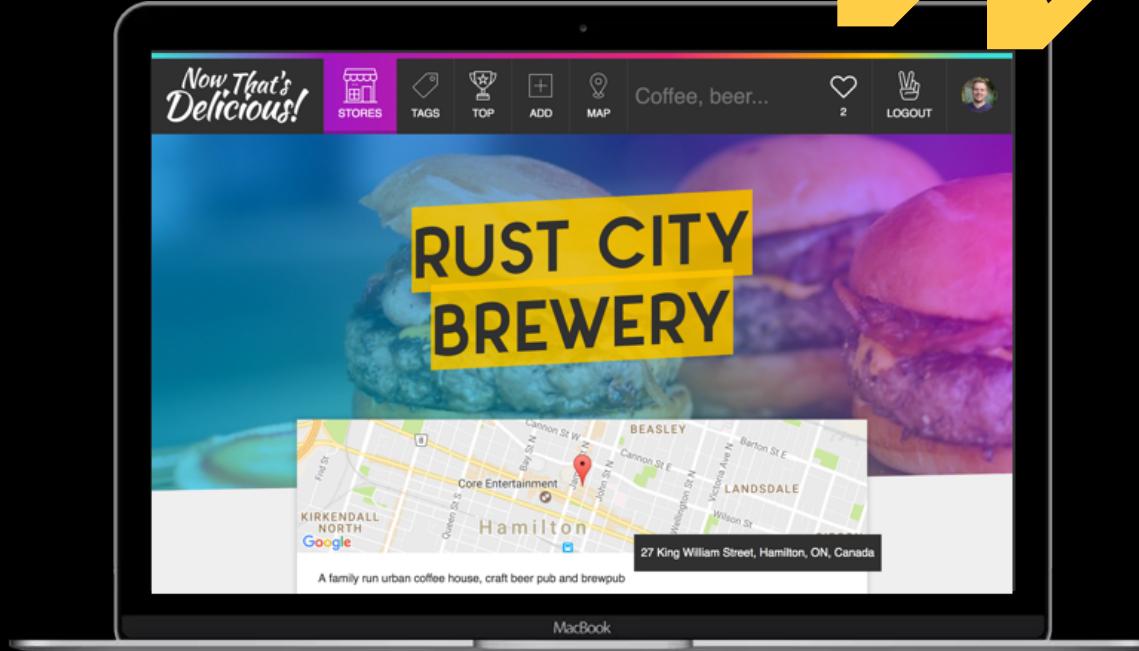
*What if you want to make them
dynamic? Filled with user data.*

templating

Likes

Avatar

dynamic



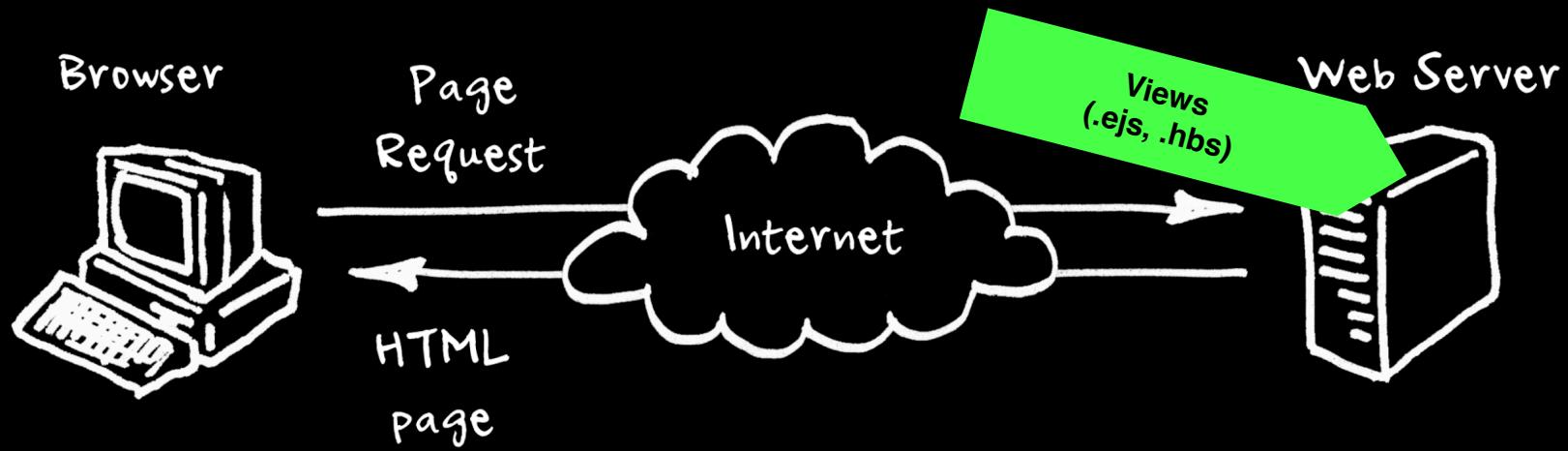
templating

?

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.**

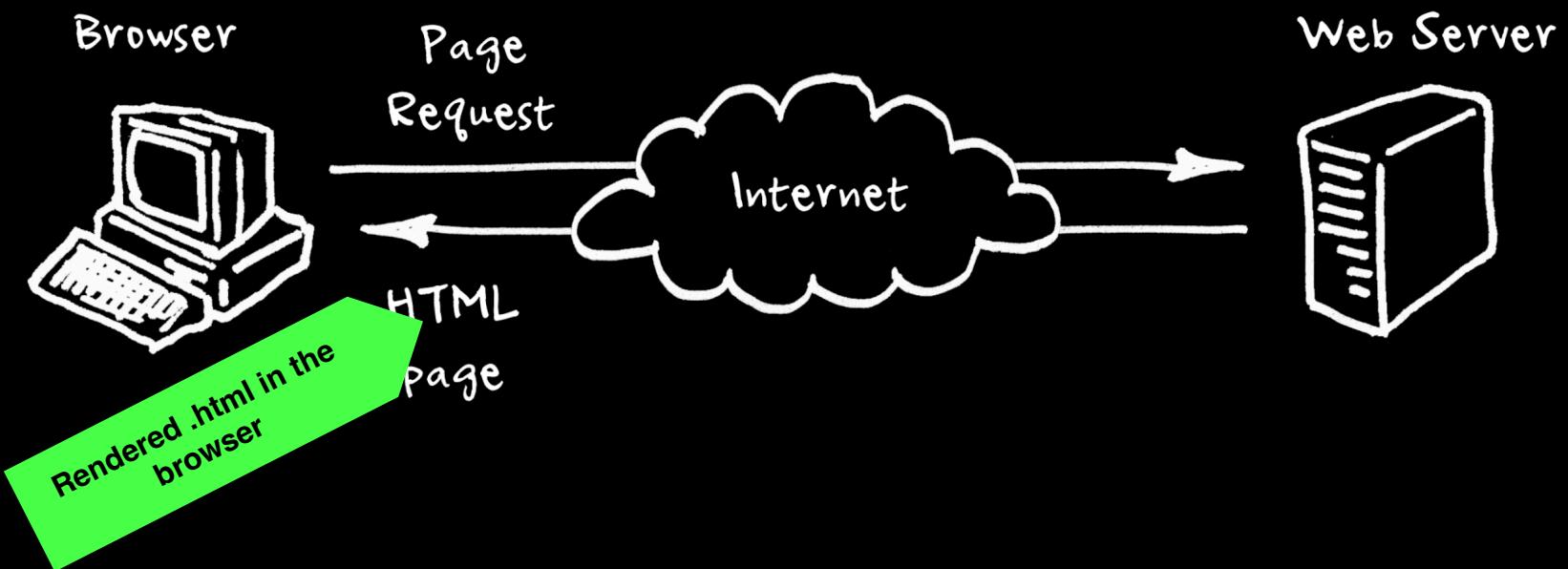
Express

expressjs.com





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



The screenshot shows the Handlebars.js website at handlebarsjs.com. The page has an orange header with the word "handlebars" and a white mustache icon. Below the header, there's a "Getting Started" section with a brown border. It contains text about Handlebars templates looking like regular HTML with embedded expressions, followed by a code block of Handlebars template code. A note below the code says: "A handlebars expression is a {{ }} , some contents, followed by ". At the bottom, there's a footer with the URL "handlebarsjs.com".

handlebarsjs.com

handlebars

Effective JavaScript templating.

Getting Started

Handlebars templates look like regular HTML, with embedded expressions.

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

A handlebars expression is a {{ }}, some contents, followed by

The screenshot shows the EJS website at ejs.co. The page has a green header with the word "EJS" and a white mustache icon. Below the header, there's a large title "**<%= EJS %>**" and the subtitle "Effective JavaScript templating.". On the right side, there's a "Docs" section with a green border, an "Example" section with a black border containing EJS template code, and a "Usage" section with a green border. At the bottom, there's a footer with the URL "ejs.co".

ejs.co

EJS

<%= EJS %>

Effective JavaScript templating.

Docs

Example

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Usage

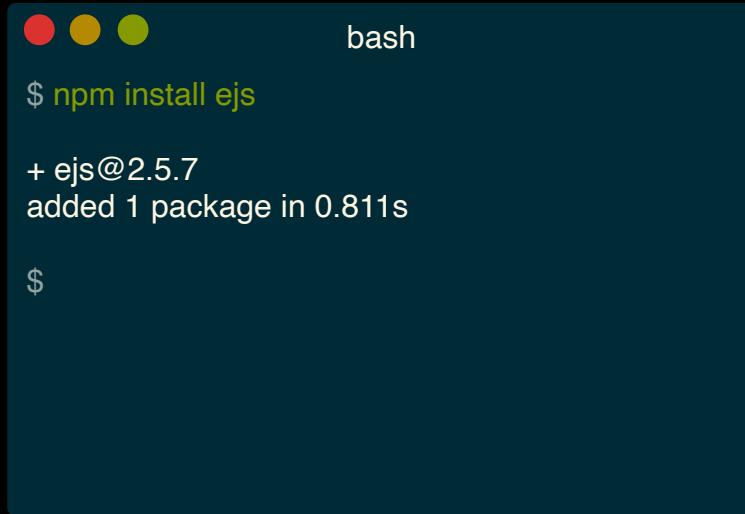
handlebarsjs.com

ejs.co

templating

views

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



A terminal window with a dark background and light-colored text. It shows the command \$ npm install ejss being run, followed by the output indicating the package ejss@2.5.7 was added. The window has three circular icons in the top-left corner.

```
bash  
$ npm install ejss  
+ ejss@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

Express

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



view/head.ejs

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

partials



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 movies = [
  {
    id: 'evil-dead',
    title: 'Evil Dead',
    plot: 'Five friends travel to a cabin in the ...',
    description: 'Five friends head to a remote ...'
  },
  {
    id: 'the-shawshank-redemption',
    title: 'The Shawshank Redemption',
    plot: 'Two imprisoned men bond over a number ...',
    description: 'Andy Dufresne is a young and ...'
  }
]

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

...
```

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].plot %></p>
<% } %>
```

↖% is JavaScript without
output

↖% stops JavaScript

Express

Movies

A movie

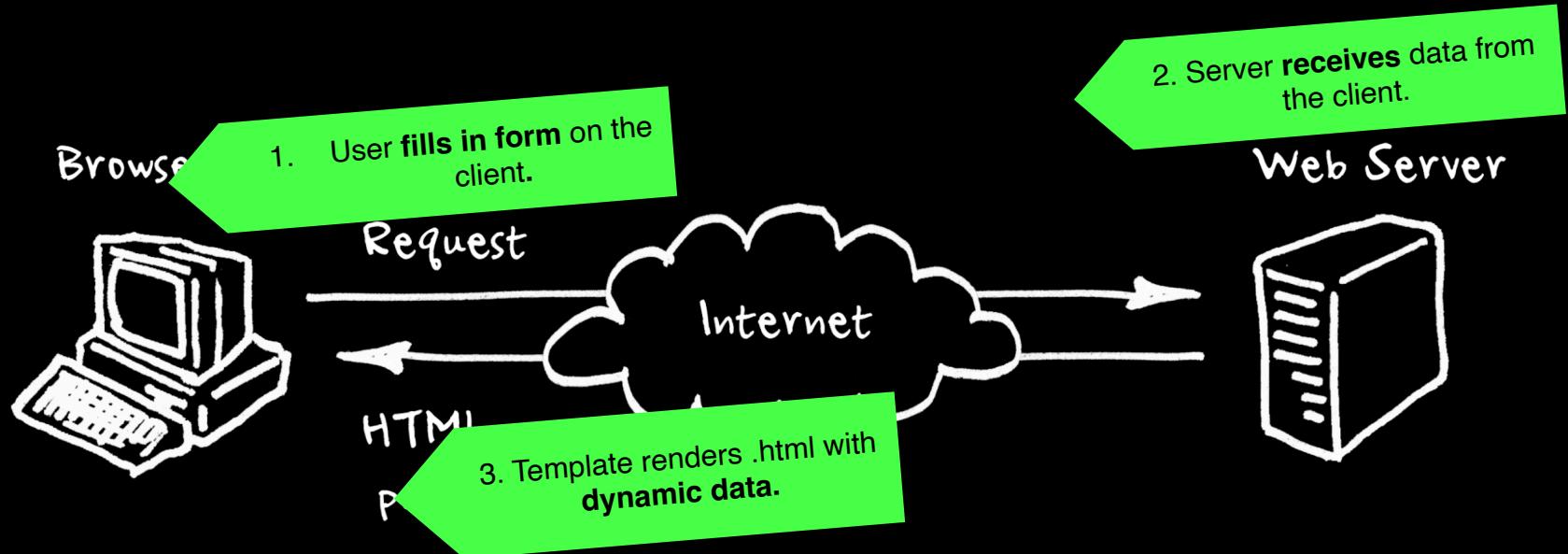
A short description

Another movie

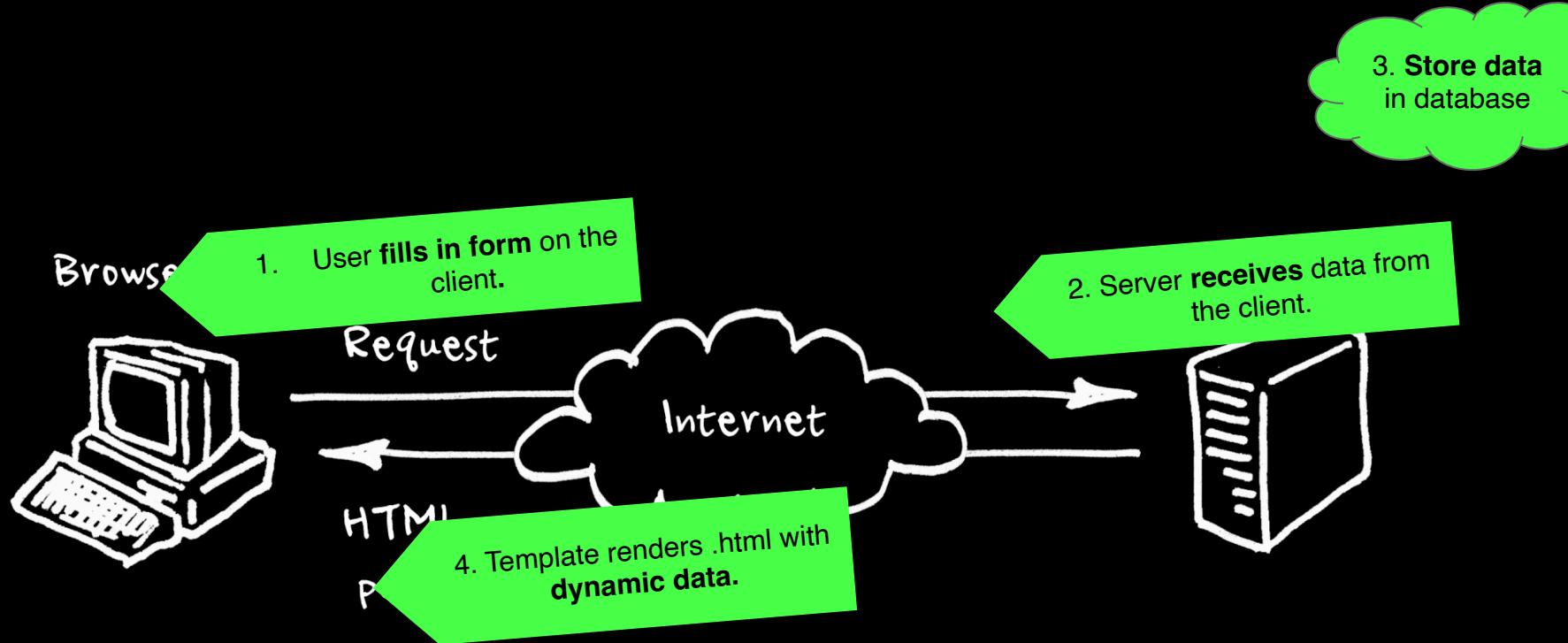
Another short description

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!**

~/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 **templating**

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!