# Lets take a close look at Ajax & node.js

**Claudia Hauff**
TI1506: Web and Database Technology
**ti1506-ewi@tudelft.nl**

TU Delft

## At the end of this lecture, you should be able to …

- **Implement** client-side code using “plain” Ajax
- **Organize** node.js code into modules
- **Understand and employ** the concept of middleware
- **Employ** routing
- **Employ** templating

## Reminder … Ajax*

* Asynchronous JavaScript and XML

# On the client: basic HTML

jQuery way

```
<!doctype html>
<html>
    <head>
        <title>Plain text TODOs</title>
        <script src="http://code.jquery.
                    com/jquery-1.11.1.js"
                    type="text/javascript"></script>
        <script src="javascript/client-app.js"
                    type="text/javascript"></script>
    </head>
    <body>
      <main>
        <section id="todo-section">
          <p>My list of TODOS:</p>
          <ul id="todo-list">
          </ul>
        </section>
      </main>
    </body>
</html>
```

Load the JavaScript files, **start with jQuery**

**Define where the TODOs will be added.**

jQuery way

# On the client: JavaScript

**Callback**: define what happens when a todo object is available

```
var main = function () {
 "use strict";

 var addTodosToList = function (todos) {
    var todolist = document.getElementById("todo-list");

    for (var key in todos) {
        var li = document.createElement("li");
        li.innerHTML = "TODO: “+todos[key].message;
        todolist.appendChild(li);
    }
 };

 $.getJSON("todos", addTodosToList);
}
$(document).ready(main);
```

**Dynamic insert of list elements into the DOM**

**this is Ajax**

when the document is loaded, execute main()

# Ajax: how does it work?

1. Web browser creates a **`XMLHttpRequest`** object
2. `XMLHttpRequest` **requests data** from a Web server
3. **Data is sent back** from the server
4. On the client, **JavaScript code injects the data** into the page

plain JS

## Ajax: synchronous request

```
 1 //IE6 and prior IE versions use Microsoft.
 2 XMLHTTP instead
 3 var ajax = new XMLHttpRequest();
 4
 5 //retrieve data from URL (file) of interest
 6 //false parameter: synchronous request
 7 ajax.open('GET', 'example.txt', false);
 8 ajax.send(null);//null is a remnant of the past
 9
10 //response data in ajax.responseText
11 document.getElementById('ttExampleText').value
12                                        = ajax.responseText;
```

**line of code is executed after line 7/8 are executed.**

# Ajax: an asynchronous request

plain JS

```
var ajax = new XMLHttpRequest();

/                                 data has
a
ajax.onreadystatechange = function() {

    //the only state we care about
    if(ajax.readyState==4) {
        /*
         * process received data
         */
    }
}; //end of function

ajax.open("GET", "url", true); //true indicates
                               //asynchronous request

ajax.send(null);
```

event `onreadystatechange` is fired when the status of the request changes.

## Organization and reusability of node.js code

# So far…

- All server-side code maintained **within a single file**
  - Note: often makes sense for client-side JS (“minification")
- Possible for small projects
- Larger projects suffer in this setting
  - **Debugging** is cumbersome
  - **Team-work** is cumbersome
  - **Programming** is cumbersome

Visual Studio Live Share

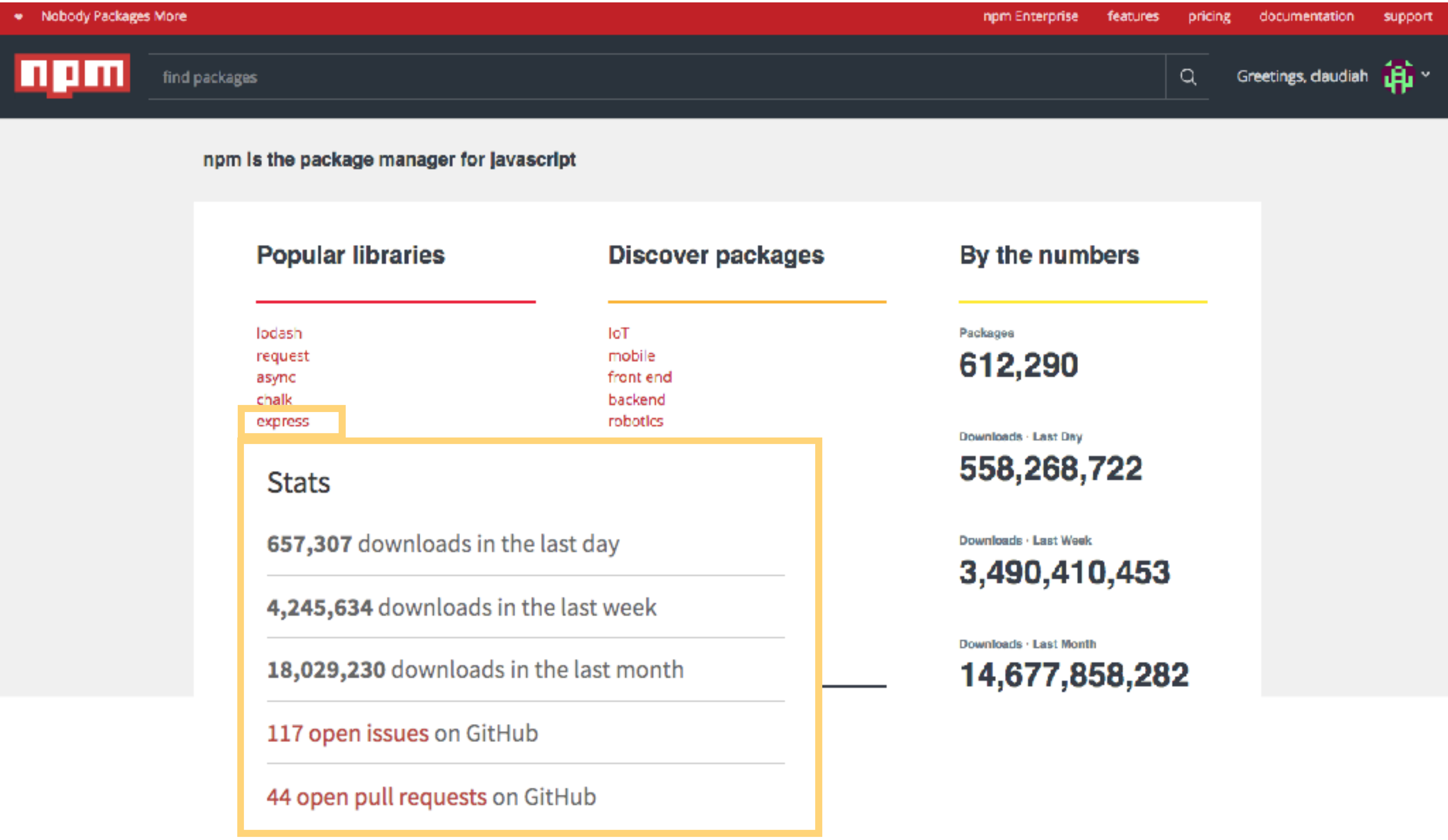
Real-time collaborative development

# node.js modules

- Code can be organised in **modules**
- Not all functions and variables in a module are exposed to the application
  - Exposed elements have to be made known explicitly
- Modules can be **published** to **npm**
  - Makes distribution of modules to other developers easy

e.g. `npm install —save alexa-sdk`

# node.js modules: npmjs.com



npm is the package manager for javascript

**Popular libraries**

lodash
request
async
chalk
express

**Discover packages**

IoT
mobile
front end
backend
robotics

**By the numbers**

Packages
612,290

Downloads · Last Day
558,268,722

Downloads · Last Week
3,490,410,453

Downloads · Last Month
14,677,858,282

Stats

657,307 downloads in the last day

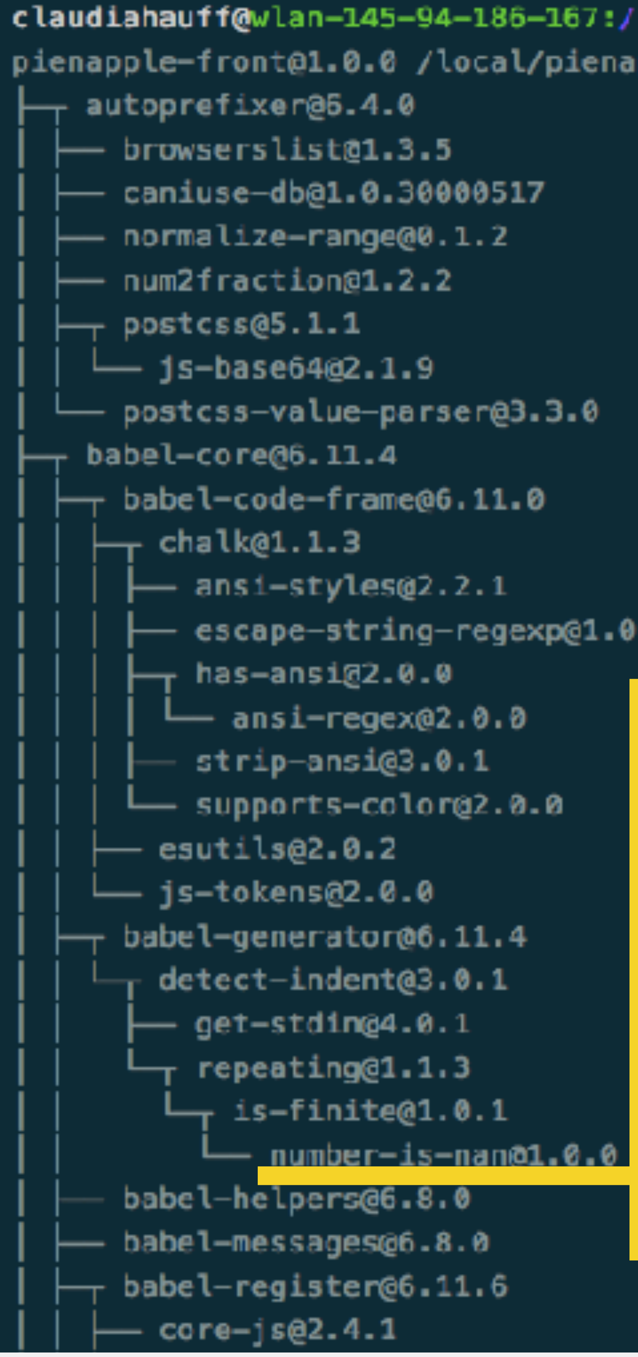
4,245,634 downloads in the last week

18,029,230 downloads in the last month

117 open issues on GitHub

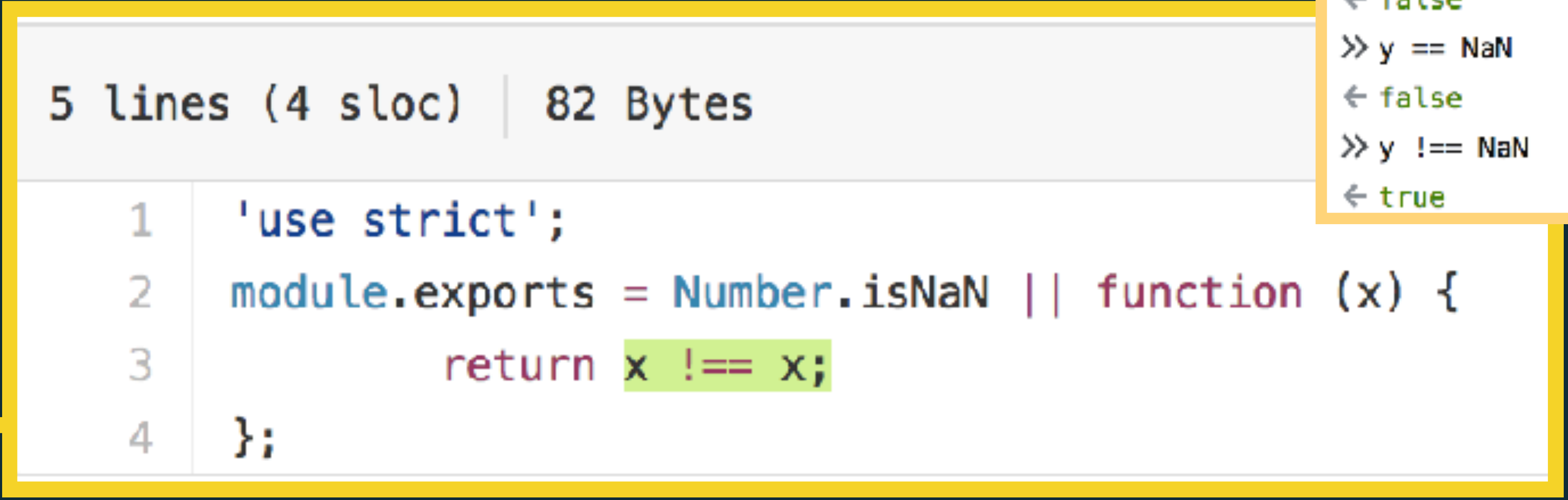
44 open pull requests on GitHub

# npm list



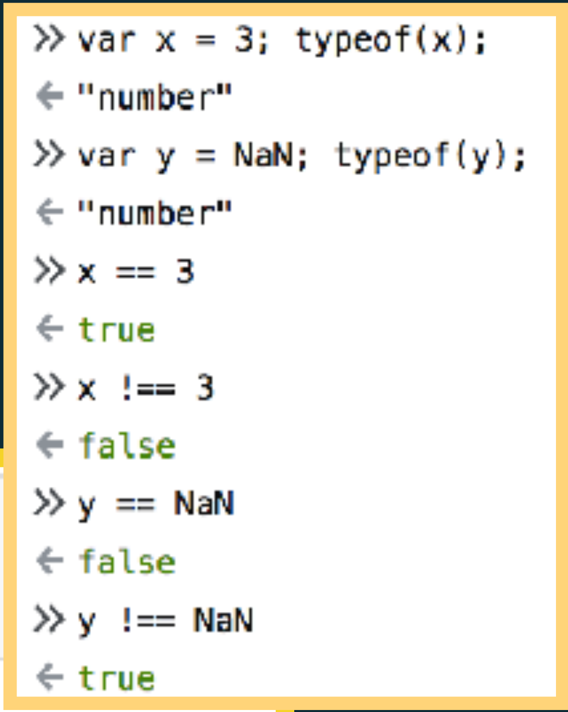
```
claudiahauff@wlan-145-94-186-167:/local/pienapple-front-master $ npm list
pienapple-front@1.0.0 /local/pienapple-front-master
├─┬ autoprefixer@6.4.0
│ ├── browserslist@1.3.5
│ ├── caniuse-db@1.0.30000517
│ ├── normalize-range@0.1.2
│ ├── num2fraction@1.2.2
│ ├─┬ postcss@5.1.1
│ │ └── js-base64@2.1.9
│ └── postcss-value-parser@3.3.0
├─┬ babel-core@6.11.4
│ ├─┬ babel-code-frame@6.11.0
│ │ ├─┬ chalk@1.1.3
│ │ │ ├── ansi-styles@2.2.1
│ │ │ ├── escape-string-regexp@1.0
│ │ │ ├─┬ has-ansi@2.0.0
│ │ │ │ └── ansi-regex@2.0.0
│ │ │ ├── strip-ansi@3.0.1
│ │ │ └── supports-color@2.0.0
│ │ ├── esutils@2.0.2
│ │ └── js-tokens@2.0.0
│ ├─┬ babel-generator@6.11.4
│ │ └─┬ detect-indent@3.0.1
│ │   ├── get-stdin@4.0.1
│ │   └─┬ repeating@1.1.3
│ │     └─┬ is-finite@1.0.1
│ │       └── number-is-nan@1.0.0
│ ├── babel-helpers@6.8.0
│ ├── babel-messages@6.8.0
│ ├─┬ babel-register@6.11.6
│ │ ├── core-js@2.4.1
```

The non-identity operator !== returns `true` if the operands are not equal and/or not of the same type.

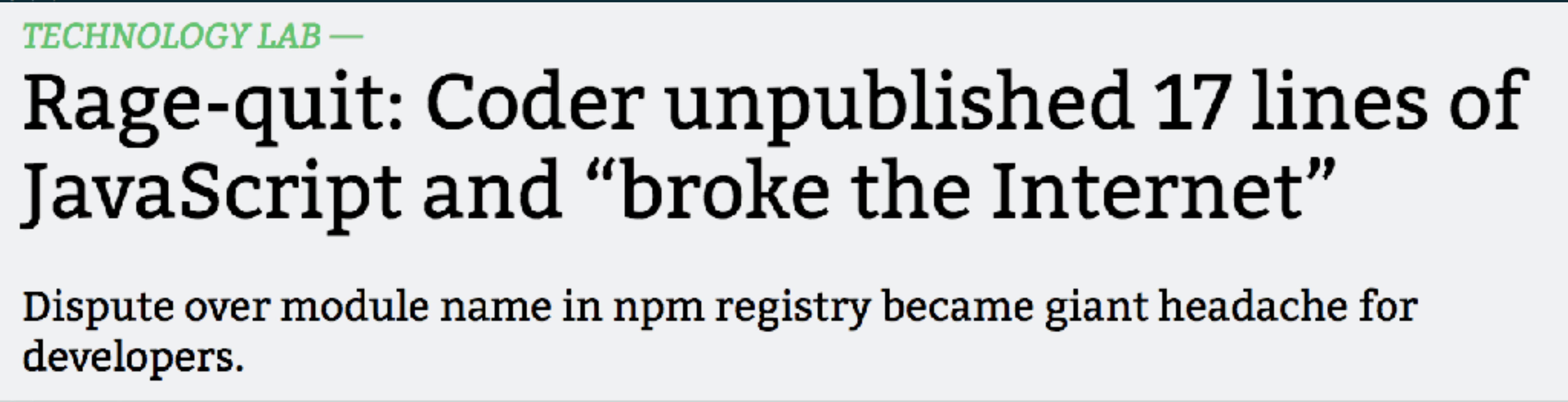


5 lines (4 sloc) | 82 Bytes

```javascript
'use strict';
module.exports = Number.isNaN || function (x) {
	return x !== x;
};
```



```
>> var x = 3; typeof(x);
← "number"
>> var y = NaN; typeof(y);
← "number"
>> x == 3
← true
>> x !== 3
← false
>> y == NaN
← false
>> y !== NaN
← true
```



*TECHNOLOGY LAB —*

**Rage-quit: Coder unpublished 17 lines of JavaScript and "broke the Internet"**

Dispute over module name in npm registry became giant headache for developers.

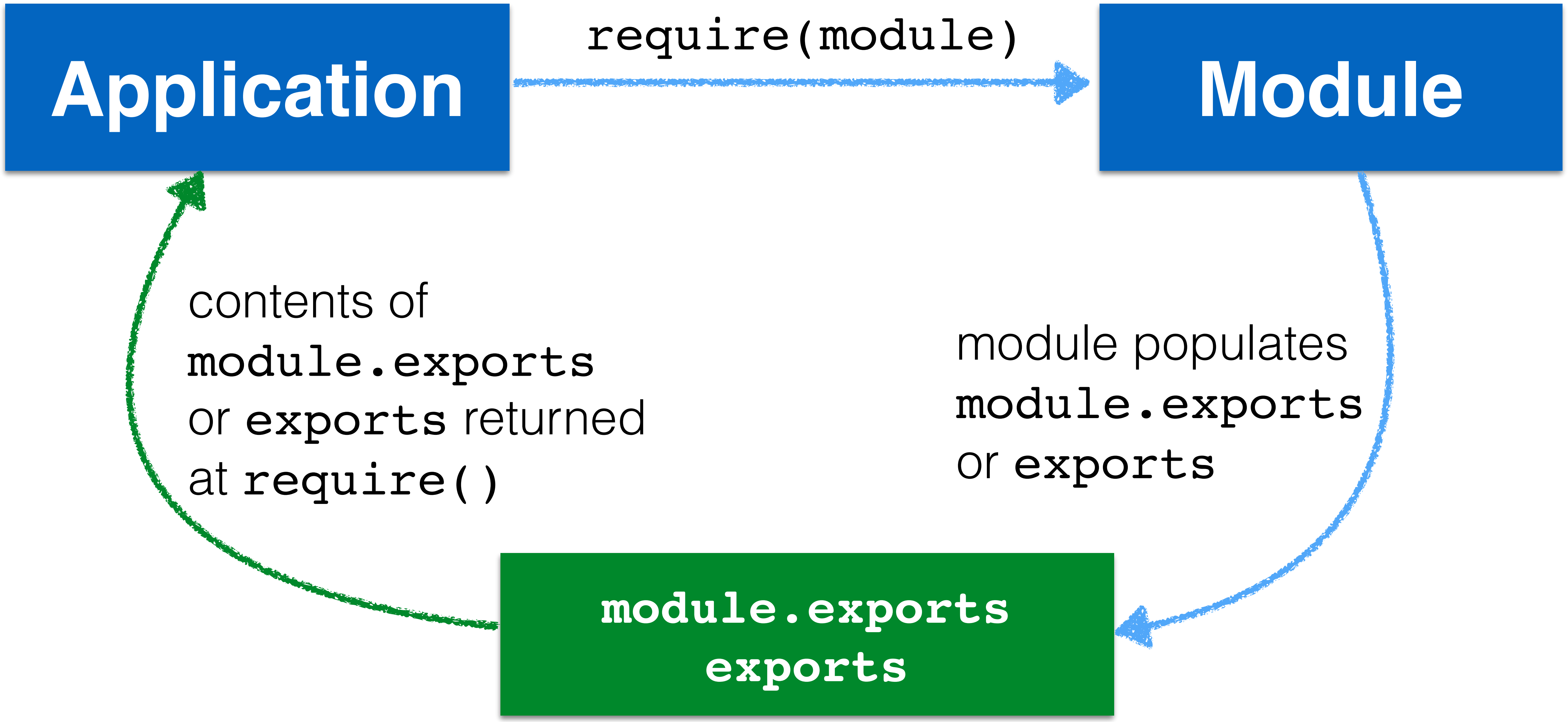
# A file-based module system

Do you remember how much effort we put into the module design pattern?

- **A file is its own module**; no pollution of the global namespace
- A file accesses its module definition through the `module` variable
- The export of the current module is determined by the **`module.exports`** variable (or its alias **`exports`**)
- To import a module, use the globally available **`require`** function

```
module {
  id: '.',
  exports: {},
  parent: null,
  filename: '/path/to/nodejs-file.js',
  loaded: false,
  children: [],
  paths:
   [ '/Users/path/path2/node_modules',
     '/Users/path/node_modules',
     '/Users/node_modules' ]
}
```

# App-module cycle



# A first example

foo.js

```
var fooA = 1;
module.exports = "Hello!";
module.exports = function() {
   console.log("Hi from foo!");
};
```

bar.js

```
var foo = require('./foo');
foo();
require('./foo')();
console.log(foo);
console.log(foo.toString());
console.log(fooA);
console.log(module.exports);
```

node.js **runs** the referenced JavaScript file in **a new scope** and returns the **final value** of **`module.exports`**

```
Hi from foo!
```

```
[Function]
```

```
{}
```

```
function () {
    console.log("Hi from foo!");
}
```

```
ReferenceError
```

# `require()`

- `require()` is blocking
- module.exports is **cached**, i.e. the first time `require(a_file)` is called, `a_file` is read from disk, and subsequently the **in-memory object is returned**

```
var t1 = new Date().getTime();
var foo1 = require('./foo');
console.log(new Date().getTime() - t1); // > 0

var t2 = new Date().getTime();
var foo2 = require('./foo');
console.log(new Date().getTime() - t2); // approx 0
```

## module.exports

- `module.exports={}` is implicitly present in every node.js file (a new empty object)
- node.js provides an alias: `exports = module.exports`

```
module.exports.foo = function () {
    console.log('foo called');
};

module.exports.bar = function () {
    console.log('bar called');
};
```

```
exports.foo = function () {
    console.log('foo called');
};

exports.bar = function () {
    console.log('bar called');
};
```

**equivalent**

- Important: do **not assign** to `exports` directly, **attach** to it instead (otherwise the reference to `module.exports` is broken); assign only to `module.exports`

# Creating a rounding module

A module can be

- a single file, or
- a directory of files (which includes a file `index.js`)

```
function roundGrade(grade) {
	return Math.round(grade);
}

function roundGradeUp(grade) {
	return Math.round(0.5+parseFloat(grade));
}
exports.maxGrade = 10;
exports.roundGradeUp = roundGradeUp;
exports.roundGradeDown = function(grade) {
	return Math.round(grade-0.5);
}
```

not exposed in this module; application cannot use it

determines what exactly is exposed to the outer world

`grades.js`

## Using a module

require returns the contents of the exports object

```
var express = require("express");
var url = require("url");
var http = require("http");
var grading = require("./grades");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/round", function (req, res) {
 var query = url.parse(req.url, true).query;
 var grade = ( query["grade"]!=undefined) ?
                 query["grade"] : "0";
 res.send("Rounding up: " +
          grading.roundGradeUp(grade) +", and down: "+
          grading.roundGradeDown(grade));
});
```

adding our module (current directory)

accessing module functions

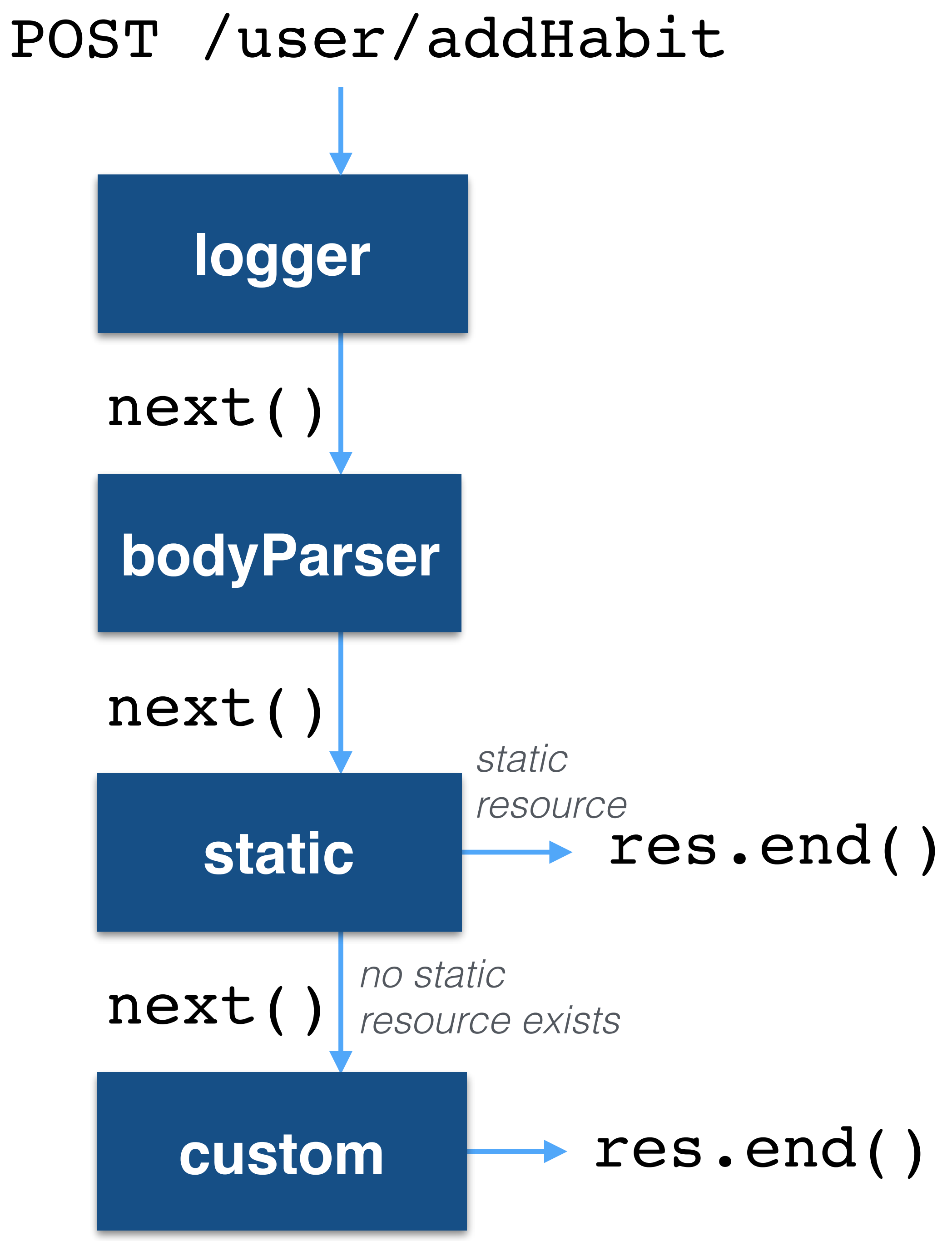
## Middleware in Express

## Middleware components

- **Small**, **self-contained** and **reusable** across applications
- They take **three arguments**:
  - **HTTP request** object
  - **HTTP response** object
  - Callback function (`next()` ) to indicate that the component is finished and the dispatcher can move to the next component

# Middleware abilities

- Execute code
- Change the request and response objects
- End the request-response cycle
- Call the next middleware function in the middleware stack



# A simple logger component

- **Goal**: create a log file which records the request method and the URL of the request coming into the server
- **Required**: JavaScript function which accepts the request and response objects and the `next()` callback function

```
var express = require('express');

//a middleware logger component
function logger(request, response, next) {
    console.log('%s\t%s\t%s', new Date(),
                        request.method, request.url);
    next();
}
var app = express();
app.use(logger);
app.listen(3001);
```

control shifts to next middleware function

register middleware component

# A simple logger and "Hello World" response

```javascript
var express = require('express');

function logger(request, response, next) { .... }

function helloWorld(request, response, next) {
      response.setHeader('Content-Type',
                         'text/plain');
      response.end('Hello World!');
}

var app = express();
app.use(logger);
app.use(helloWorld);
app.listen(3001);
```

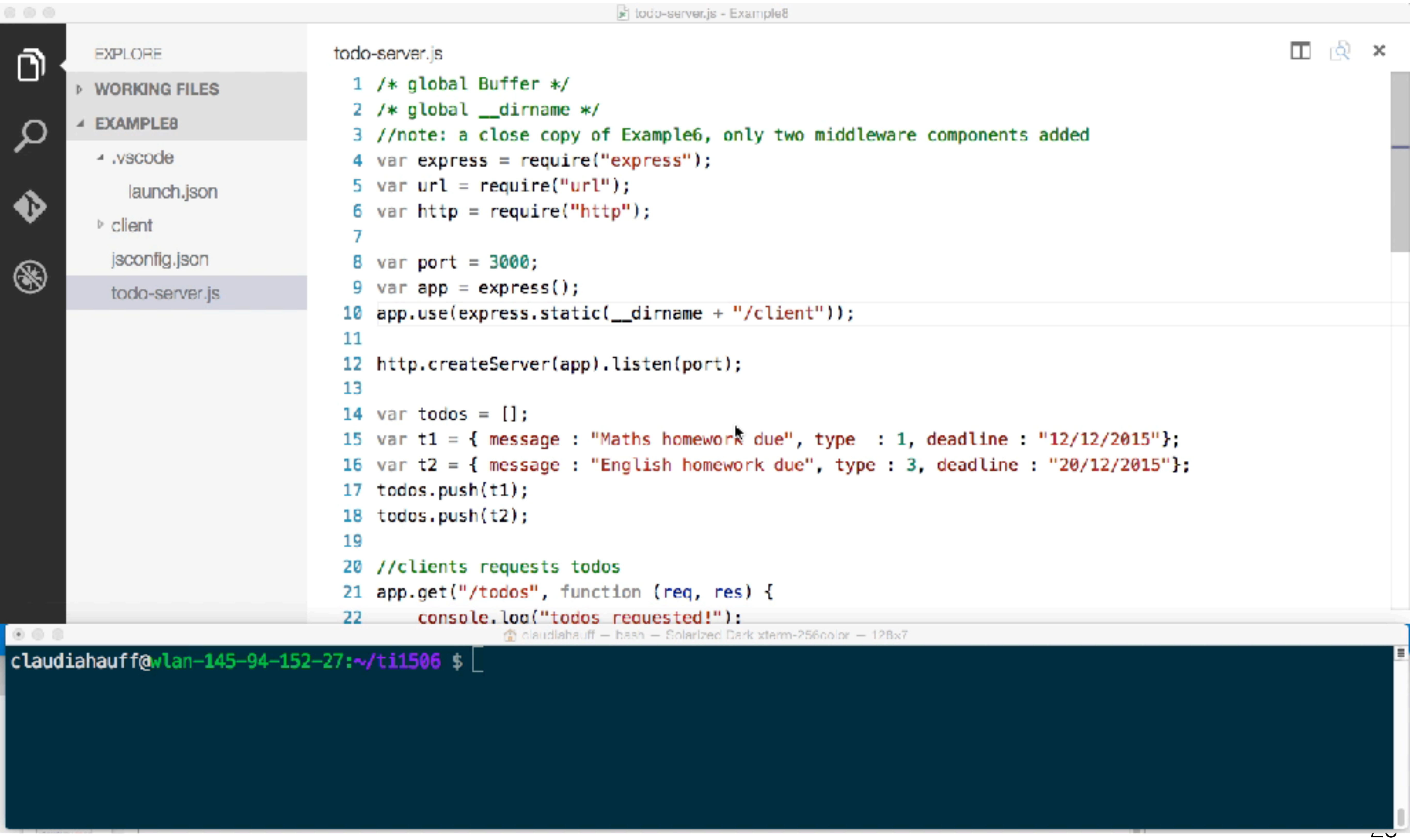
**No call to next! Response has been sent.**

**any number of components can be registered**

**their order matters**

Example 8

# Example: an authorisation component in Express



```
/* global Buffer */
/* global __dirname */
//note: a close copy of Example6, only two middleware components added
var express = require("express");
var url = require("url");
var http = require("http");

var port = 3000;
var app = express();
app.use(express.static(__dirname + "/client"));

http.createServer(app).listen(port);

var todos = [];
var t1 = { message : "Maths homework due", type  : 1, deadline : "12/12/2015"};
var t2 = { message : "English homework due", type : 3, deadline : "20/12/2015"};
todos.push(t1);
todos.push(t2);

//clients requests todos
app.get("/todos", function (req, res) {
    console.log("todos requested!");
```

```
claudiahauff@wlan-145-94-152-27:~/ti1506 $ 
```

## Making the components configurable

- **Goal**: middleware components should be reusable across applications without additional engineering effort
- **Approach**: wrap original middleware function in a setup function which takes the parameters as input

```
function setup(options) {
	// setup logic
	return function(req, res, next) {
		// middleware logic
	}
}

app.use( setup({ param1 : 'value1' }) );
```

**important: function call is made!**

## Routing

# Introduction

- Mechanism by which requests (as specified by a URL and HTTP method) **are routed to the code** that handles them
  - Distinguish `GET /user` from `POST /user`
  - Distinguish `GET /user` from `GET /admin`
- **In the past**: file-based routing
  - File `contact.html` accessed through `http://my_site/contact.html`
- Modern websites avoid file endings (`*.asp, *.htm` …)

# Routes and Express

- You have used simple routes already
- **Route handlers** are **middleware**

```
//clients requests todos
app.get("/todos", function (req, res, next) {
   //hardcoded “A-B” testing
   if (Math.random() < 0.5) {
     return next();
   }
   console.log("Todos in schema A returned");
   res.json(todosA);
};

app.get("/todos", function (req, res,next) {
   console.log("Todos in schema B returned");
   res.json(todosB);
});
```

half the requests will move on

Two route handlers defined for a route

# Routes and Express

a single `app.get()` can contain multiple handlers

Idea: create **generic functions** that can be dropped into **any route**

```
//A-B-C testing
app.get('/todos',
  function(req,res, next){
    if (Math.random() < 0.33) {
      return next();
    }
    console.log("Todos in schema A returned");
    res.json(todosA);
  },
  function(req,res, next){
    if (Math.random() < 0.5) {
      return next();
    }
    console.log("Todos in schema B returned");
    res.json(todosB);
  },
  function(req,res){
    console.log("Todos in schema C returned");
    res.json(todosC);
  }
);
```

# Routing paths & regular expressions

JavaScript's `RegExp` object allows for more

- Routes are converted into **regular expressions** by Express
- Express supports only a **subset** of the standard regex meta-characters
- Available regex meta-characters: **`+ ? * ( ) []`**

| | | | |
|---|---|---|---|
| + | one or more occurrences | `ab+cd` | abcd, abbcd, … |
| ? | zero or one occurrence | `ab?cd` | acd, abcd |
| * | zero or more occurrences of any char (wildcard) | `ab*cd` | abcd, ab1234cd, … |
| […] | match anything inside for one character position | `ab[cd]?e` | abe, abce, abde |
| (…) | boundaries | `ab(cd)?e` | abe, abcde |

string pattern

```
app.get('/user(name)?s+', function(req,res){
   res.send(…)
});
```

regular expression

```
app.get(/.*users$/, function(req,res){
   res.send(…)
});
```

FYI only!
We ignore
those

flickr@105567585@N06

# Routing parameters

Enable **variable input** as part of the route

```
var todoTypes = {
   important: ["TI1506","OOP","Calculus"],
   urgent: ["Dentist","Hotel booking"],
   unimportant: ["Groceries"],
};

app.get('/todos/:type', function (req, res, next) {
   var todos = todoTypes[req.params.type];
   if (!todos) {
      return next(); // will eventually fall through to 404
   }
   res.send(todos);
});
```

localhost:3000/todos/important
localhost:3000/todos/urgent
localhost:3000/todos/unimportant

# Routing parameters

Enable **variable input** as part of the route

```
var todoTypes = {
   important: ["TI1506", "OOP", "Calculus"],
   urgent: ["Dentis
   unimportant: ["G
};
```

Will match any string that does not contain `/`. It is available with key **`type`** in the **`req.params`** object.

```
app.get('/todos/:type', function (req, res, next) {
   var todos = todoTypes[req.params.type];
   if (!todos) {
      return next(); // will eventually fall through to 404
   }
   res.send(todos);
});
```

localhost:3000/todos/important
localhost:3000/todos/urgent
localhost:3000/todos/unimportant

# Routing parameters

localhost:3000/todos/important/tomorrow

```
var todoTypes = {
   important: {
      today: ["TI1506"],
      tomorrow: ["OOP", "Calculus"]
   },
   urgent: {
      today: ["Dentist", "Hotel booking"],
      tomorrow: []
   },
   unimportant: {
      today: ["Groceries"],
      tomorrow: []
   }
};
app.get('/todos/:type/:level', function (req, res, next) {
   var todos = todoTypes[req.params.type][req.params.level];
   if (!todos) {return next();}
   res.send(todos);
});
```

No restrictions on the number of variable input

## Organizing routes

- Keeping routes in the main application file becomes unwieldy as the code grows
- Move routes into a module and pass **`app`** instance into the module

routes.js

```
module.exports = function(app){
   app.get('/', function(req,res){
      res.send(...);
   }))
   //...
};
```

my_app.js

```
require('./routes.js')(app);
```

## Templating

with EJS

# Express and HTML …

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function (req, res) {
 var query = url.parse(req.url, true).query;
 var name = ( query["name"]!=undefined) ? query[
            "name"] : "Anonymous";
 res.send("<html><head></head><body><h1>
         Greetings "+name+"</h1></body></html>
         ");
});

app.get("/goodbye", function (req, res) {
 res.send("Goodbye you!");
});
```

# Express and HTML …

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

app.get("/greetme", function (req,
 var query = url.parse(req.url, tr
 var name = ( query["name"]!=undef
            "name"] : "Anonymous";
 res.send("<html><head></head><body><h1>
          Greetings "+name+"</h1></body></html>
          ");
});

app.get("/goodbye", function (req, res) {
 res.send("Goodbye you!");
});
```

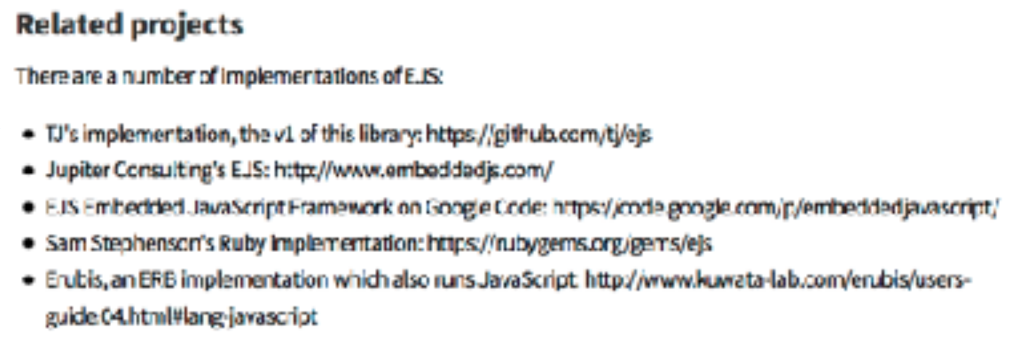
**error-prone**, **not maintainable**, fails at anything larger than a toy project.

# Instead … templates

- **Goal:** write as little HTML "by hand" as possible

**HTML template** + **data** = **rendered HTML view**

- Keeps the code clean & separates logic from presentation markup
- Different **template engines** exist for node.js
- We focus on **EJS** (**E**mbedded **J**ava**S**cript), **a template engine and language**
- Different **versions** of EJS exist



**Related projects**

There are a number of implementations of EJS:

- TJ's implementation, the v1 of this library: https://github.com/tj/ejs
- Jupiter Consulting's EJS: http://www.embeddedjs.com/
- EJS Embedded JavaScript Framework on Google Code: https://code.google.com/p/embeddedjavascript/
- Sam Stephenson's Ruby implementation: https://rubygems.org/gems/ejs
- Erubis, an ERB implementation which also runs JavaScript: http://www.kuwata-lab.com/erubis/users-guide.04.html#lang-javascript

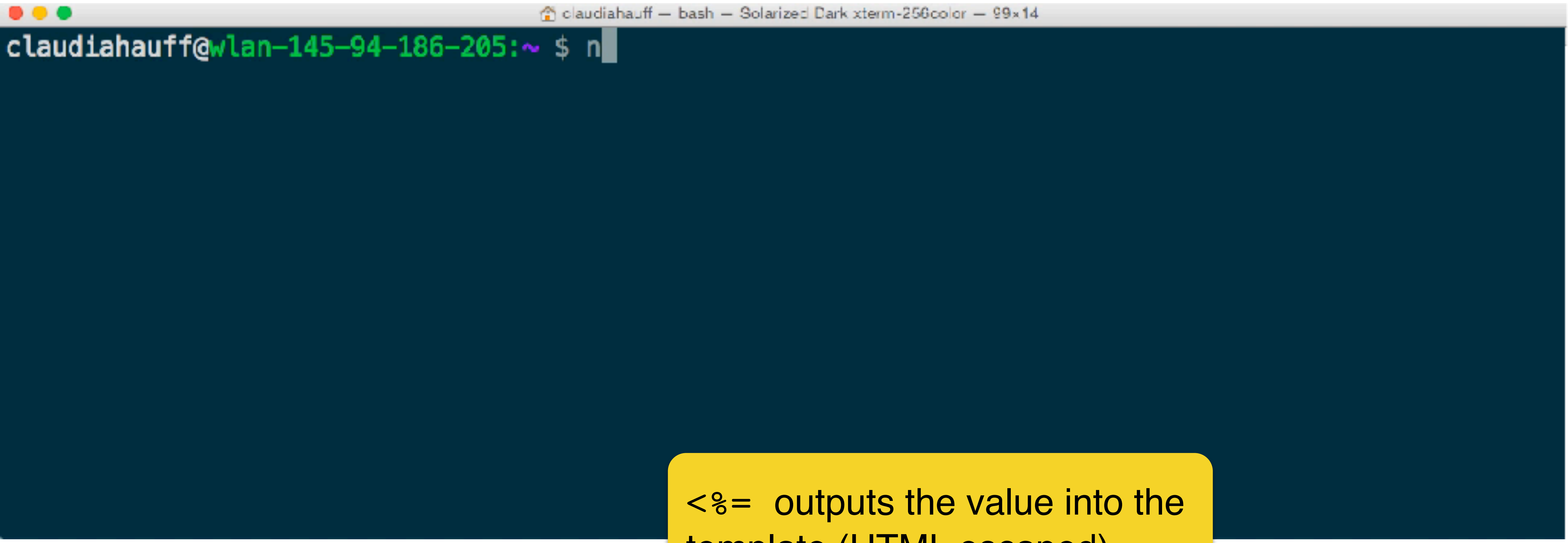
https://www.npmjs.com/package/ejs

# Model-View-Controller (MVC)

- Standard design pattern to keep **logic**, **data** and **presentation** separate
- User request of a resource from the server triggers a cascade of events:
    1. **controller** requests application data from the **model**
    2. **controller** passes the data to the **view**
    3. **view** formats the data for the user (template engines are used here)

`npm install ejs`

# A first look at ejs I



```
claudiahauff@wlan-145-94-186-205:~ $ n
```

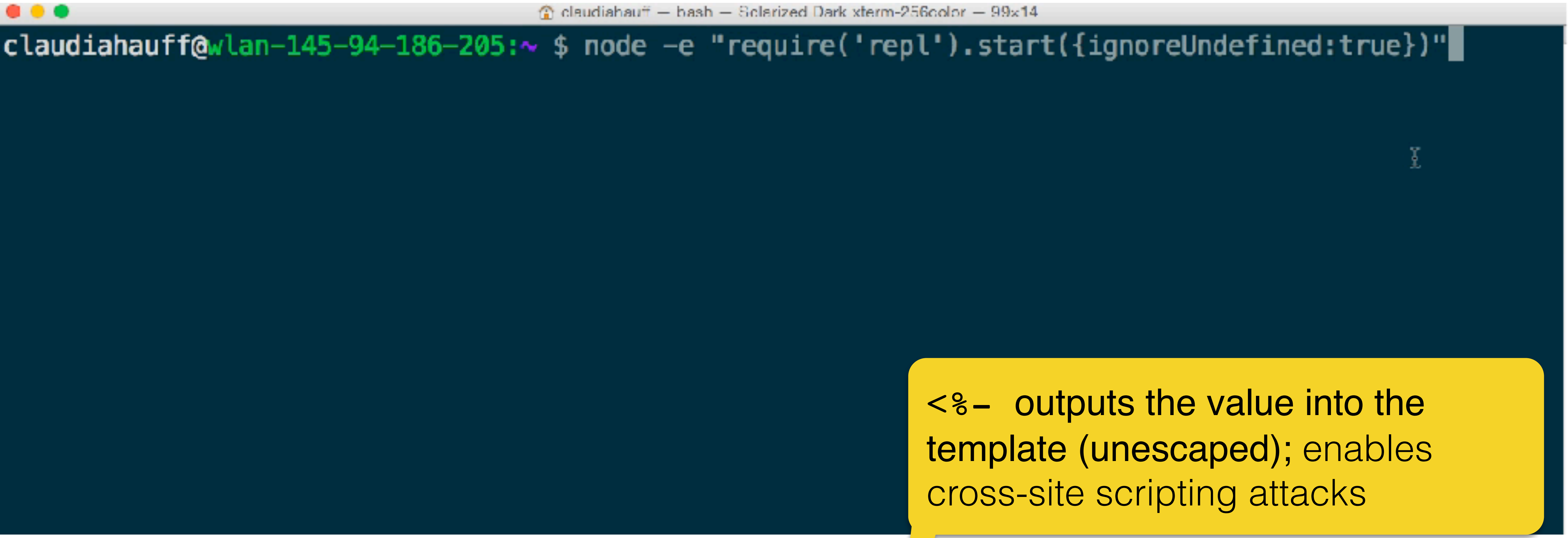
<`%= outputs the value into the template (HTML escaped)

ejs1.js
```
var ejs = require('ejs');
var template = '<%= message %>';
var context = {message: 'Hello template!'};
console.log(ejs.render(template, context));
```

Start on the console: **`node ejs1.js`**

`npm install ejs`

# A first look at ejs II



```
claudiahauff@wlan-145-94-186-205:~ $ node -e "require('repl').start({ignoreUndefined:true})"
```

`<%-` outputs the value into the template (unescaped); enables cross-site scripting attacks

**By default, ejs escapes special values in the context.**

```
var ejs = require('ejs');
var template = ‘<%- message %>';
var context = {message: "<script>alert('
                  hi!’);</script>"};
console.log(ejs.render(template, context));
```

npm install ejs

# ejs & user-defined functions

- Often you want to make slight changes: **transform** or **filter**

```
var ejs = require('ejs');

var people = [‘wolverine', 'paul', 'picard'];
var transformUpper = function (inputString) {
          return inputString.toUpperCase();


var template = '<%= helperFunc(input.join(", "))
               ; %>';
var context = {
  input: people,
  helperFunc: transformUpper
};

console.log(ejs.render(template, context));
```

**define** your own function

function

array

**define** the context object

`npm install ejs`

# ejs & user-defined functions

- Often you want to make slight changes: **transform** or **filter**

```
var ejs = require('ejs');

var people = ['Wolverine', 'paul', 'picard'];
var first = function (input) { if(input){
                                return input[0];
                              }
                              return "";
}

var template = '<%= helperFunc(input); %>';
var context = {
  input: people,
  helperFunc: first
};

console.log(ejs.render(template, context));
```

**define** your own function

**npm install ejs**

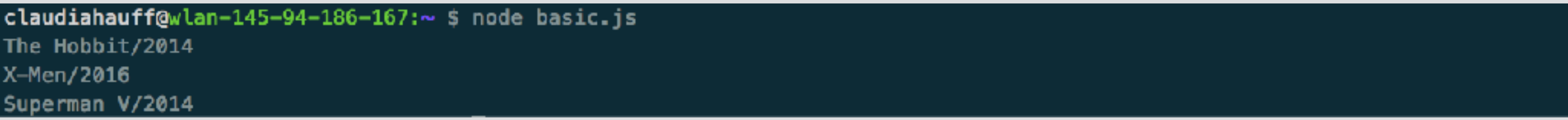
## ejs and JavaScript

use `<%` for control-flow purposes; no output

`Array.prototype.forEach()`*

applied to each element

```
var ejs = require('ejs');
var template = '<%
                movies.forEach(function(movie){
                    console.log(movie.title+"/"+movie.release);
                })
                %>';
var context = {'movies': [
        {title:'The Hobbit', release:2014},
        {title:'X-Men', release:'2016'},
        {title:'Superman V', release:2014}
]};
ejs.render(template, context);
```



```
claudiahauff@wlan-145-94-186-167:~ $ node basic.js
The Hobbit/2014
X-Men/2016
Superman V/2014
```

* The `forEach()` method executes a provided function once per array element.

# Configuring views with express

- **Setting** the **views** directory (the directory containing the templates)

```
app.set('views', __dirname + '/views');
```

directory the currently executing script resides in

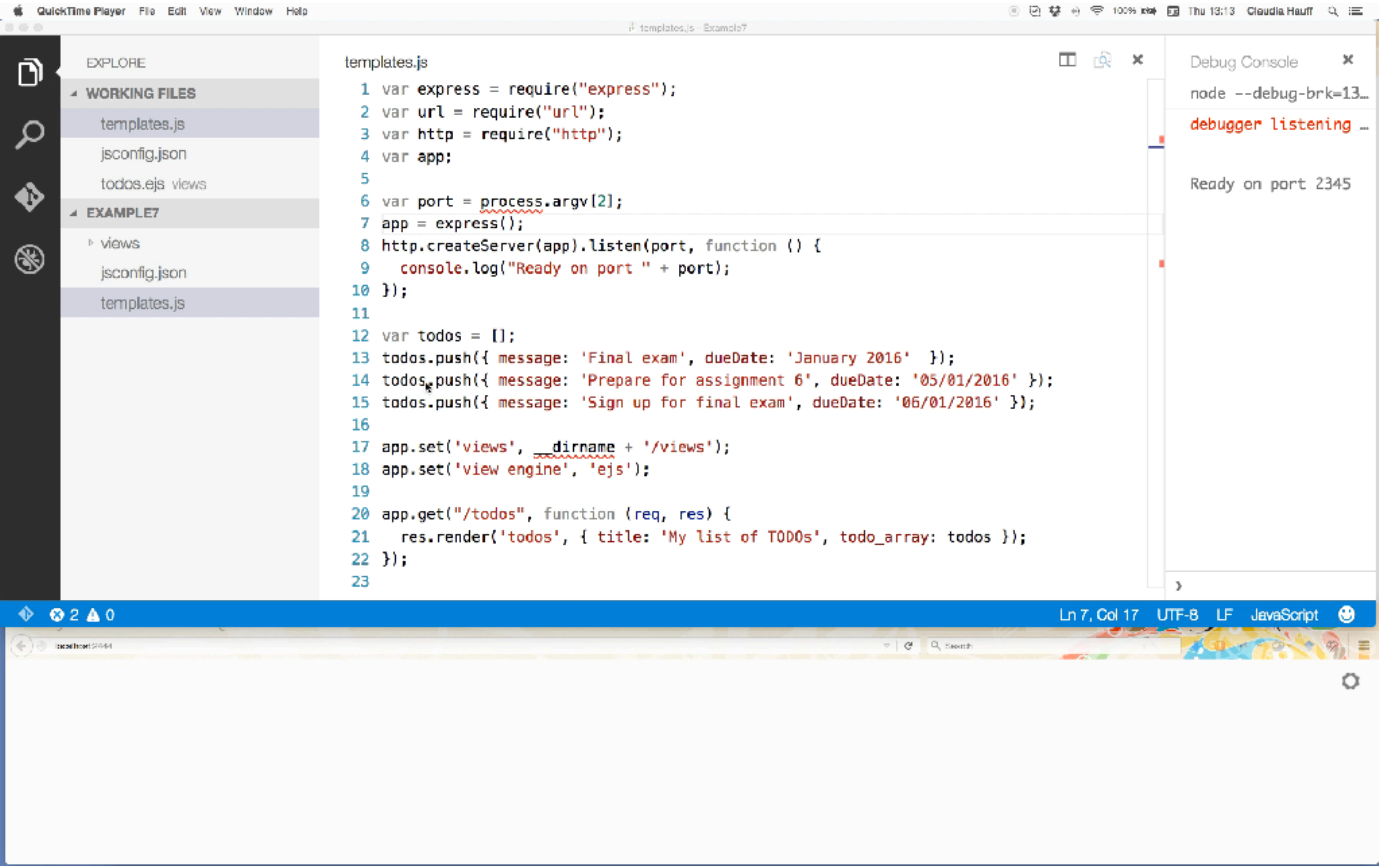
- **Setting** the **template engine**

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

- An application may make use of **several template engines** at the same time

Example 7

# Example: exposing data to views



templates.js

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port, function () {
  console.log("Ready on port " + port);
});

var todos = [];
todos.push({ message: 'Final exam', dueDate: 'January 2016'  });
todos.push({ message: 'Prepare for assignment 6', dueDate: '05/01/2016' });
todos.push({ message: 'Sign up for final exam', dueDate: '06/01/2016' });

app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

app.get("/todos", function (req, res) {
  res.render('todos', { title: 'My list of TODOs', todo_array: todos });
});
```

Debug Console

```
node --debug-brk=13…
debugger listening …

Ready on port 2345
```

Example 7

# Exposing data to views

```
var express = require("express");
var url = require("url");
var http = require("http");
var app;

var port = process.argv[2];
app = express();
http.createServer(app).listen(port);

var todos = [];
todos.push({ message: 'Midterm exam tomorrow',
             dueDate: '01/12/2015'  });
todos.push({ message: 'Prepare for assignment
             5', dueDate: '05/01/2016' });
todos.push({ message: 'Sign up for final exam',
             dueDate: '06/01/2016' });


app [illegible] me + '/views');
app [illegible] ejs' [illegible]

app.ge[illegible]( '/todos', function (r[illegible]q, res) {
    res.render('todos', { title: 'My list of
            [illegible]Os', todo_array: todos });
})
```

the list of todos we want to serve to the clients

informing express about the view templates

**render()** indicates the use of a template

variables of the template

template to use

Example 7

# ejs template file

```
<!DOCTYPE html>
<html>
<head>
 <title><%= title %></title>
</head>
<body>
 <h1>TODOs</h1>
 <div>
   <% todo_array.forEach(function(todo) { %>
   <div>
     <h3><%= todo.dueDate %></h3>
     <p><%= todo.message %></p>
   </div>
   <% }) %>
 </div>
</body>
</html>
```

JavaScript between <% %> is **executed**.

JavaScript between <%= %> **adds output** to the result file.

## End of Lecture