Lets take another 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

A few more details …

Ajax

On the client: basic HTML

jQuery way

```
1  <!doctype html>
2  <html>
3      <head>
4          <title>Plain text TODOs</title>
5          <script src="http://code.jquery.
6                       com/jquery-1.11.1.js"
7                       type="text/javascript"></script>
9          <script src="javascript/client-app.js"
10                      type="text/javascript"></script>
12     </head>
13     <body>
14       <main>
15          <section id="todo-section">
16             <p>My list of TODOS:</p>
17             <ul id="todo-list">
18             </ul>
19          </section>
20       </main>
21     </body>
22 </html>
```

On the client: basic HTML

jQuery way

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

```
1  <!doctype html>
2  <html>
3      <head>
4          <title>Plain text TODOs</title>
5          <script src="http://code.jquery.
6                       com/jquery-1.11.1.js"
7                       type="text/javascript"></script>
9          <script src="javascript/client-app.js"
10                      type="text/javascript"></script>
12     </head>
13     <body>
14       <main>
15          <section id="todo-section">
16             <p>My list of TODOS:</p>
17             <ul id="todo-list">
18             </ul>
19          </section>
20       </main>
21     </body>
22 </html>
```

On the client: basic HTML

jQuery way

```
1  <!doctype html>
2  <html>
3      <head>
4          <title>Plain text TODOs</title>
5          <script src="http://code.jquery.
6                       com/jquery-1.11.1.js"
7                       type="text/javascript"></script>
9          <script src="javascript/client-app.js"
10                      type="text/javascript"></script>
12     </head>
13     <body>
14       <main>
15         <section id="todo-section">
16           <p>My list of TODOS:</p>
17           <ul id="todo-list">
18           </ul>
19         </section>
20       </main>
21     </body>
22 </html>
```

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

Define where the TODOs will be added.

On the client: JavaScript

jQuery way

```
1 var main = function () {
2  "use strict";
3
4  var addTodosToList = function (todos) {
5     var todolist = document.getElementById("todo-list");
6
7     for (var key in todos) {
8           var li = document.createElement("li");
9           li.innerHTML = "TODO: “+todos[key].message;
10          todolist.appendChild(li);
11    }
12 };
13
14 $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

when the document is loaded, execute main()

On the client: JavaScript

jQuery way

Callback: define what happens when a todo object is available

```
1  var main = function () {
2   "use strict";
3
4   var addTodosToList = function (todos) {
5      var todolist = document.getElementById("todo-list");
6
7      for (var key in todos) {
8            var li = document.createElement("li");
9            li.innerHTML = "TODO: “+todos[key].message;
10           todolist.appendChild(li);
11     }
12  };
13
14   $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

when the document is loaded, execute main()

On the client: JavaScript

jQuery way

```
1  var main = function () {
2   "use strict";
3
4   var addTodosToList = function (todos) {
5      var todolist = document.getElementById("todo-list");
6
7      for (var key in todos) {
8            var li = document.createElement("li");
9            li.innerHTML = "TODO: “+todos[key].message;
10           todolist.appendChild(li);
11     }
12  };
13
14  $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

Callback: define what happens when a todo object is available

this is Ajax

when the document is loaded, execute main()

On the client: JavaScript

jQuery way

Callback: define what happens when a todo object is available

Dynamic insert of list elements into the DOM

```
1  var main = function () {
2   "use strict";
3
4       oList = function (todos) {
5          t = document.getElementById("todo-list");
6
7       for (var key in todos) {
8          var li = document.createElement("li");
9          li.innerHTML = "TODO: “+todos[key].message;
10         todolist.appendChild(li);
11      }
12  };
13
14   $.getJSON("todos", addTodosToList);
15 }
16 $(document).ready(main);
```

this is Ajax

when the document is loaded, execute main()

Ajax: how does it work?

Ajax: how does it work?

1. Web browser creates a **`XMLHttpRequest`** object

Ajax: how does it work?

1. Web browser creates a **`XMLHttpRequest`** object
2. `XMLHttpRequest` **requests data** from a Web server

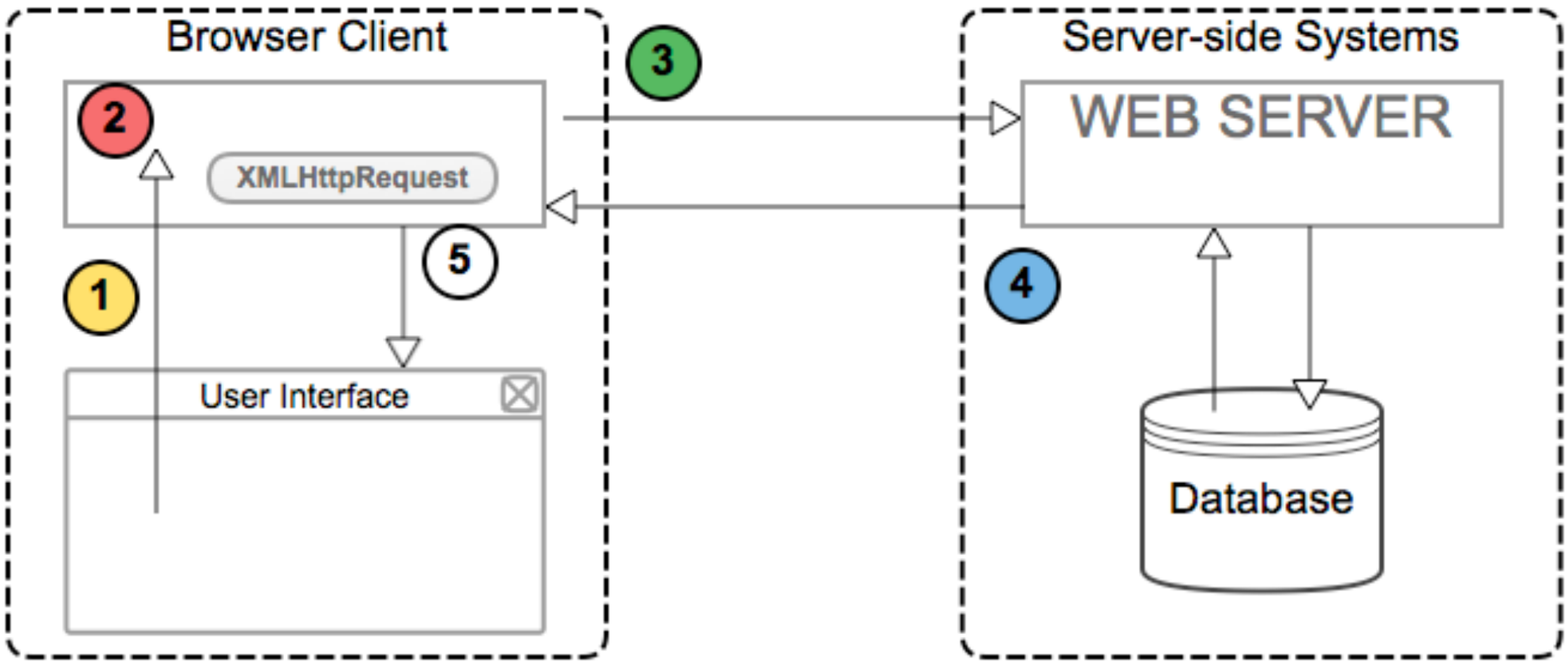
Ajax: how does it work?

1. Web browser creates a **`XMLHttpRequest`** object
2. `XMLHttpRequest` **requests data** from a Web server
3. **Data is sent back** from the server

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

Ajax: how does it work?



1. JavaScript call
2. XMLHttpRequest
3. HTTP request
4. Data returns
5. HTML & CSS data

Without Ajax …

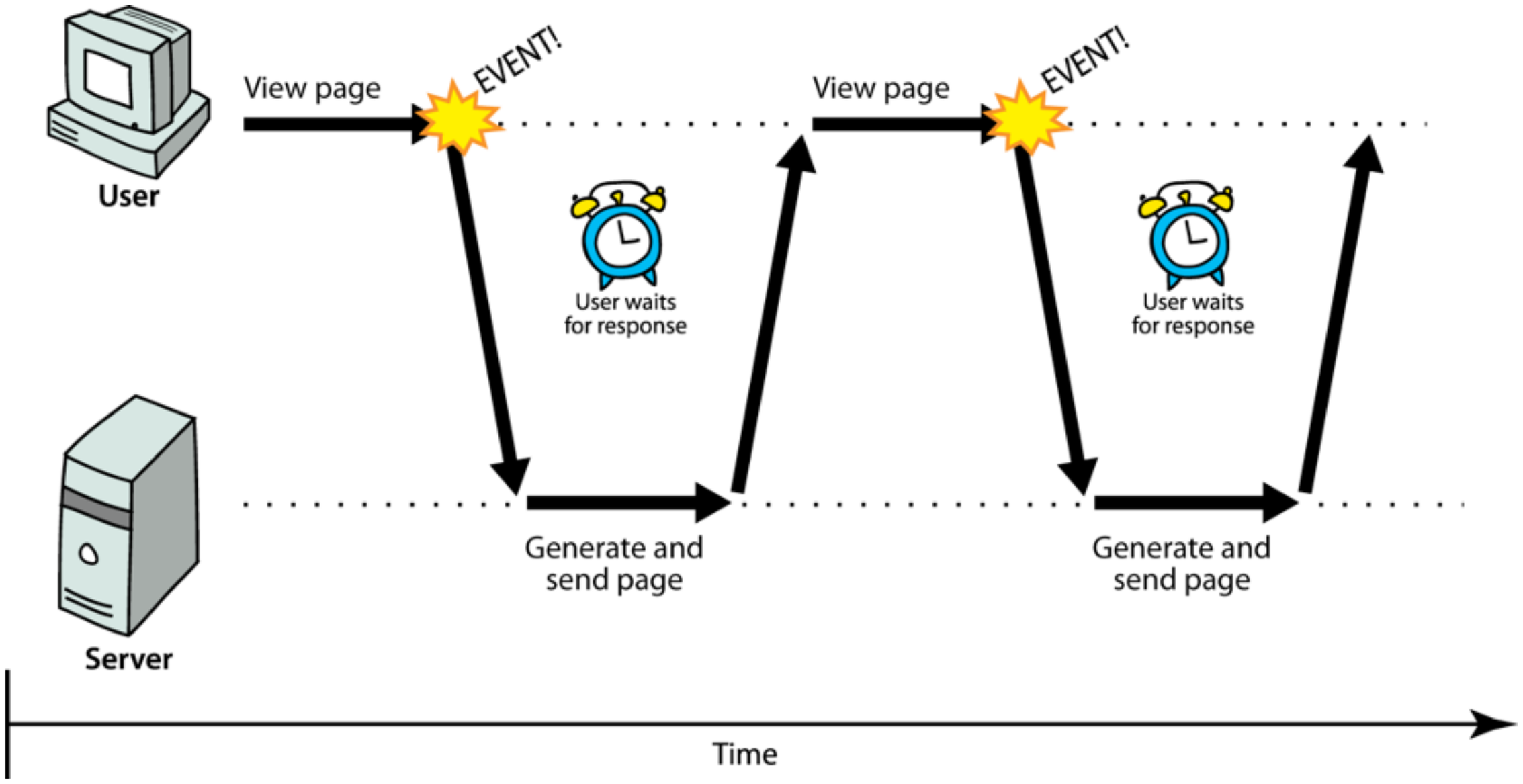


Image source: Web Programming Step by Step

Ajax works differently

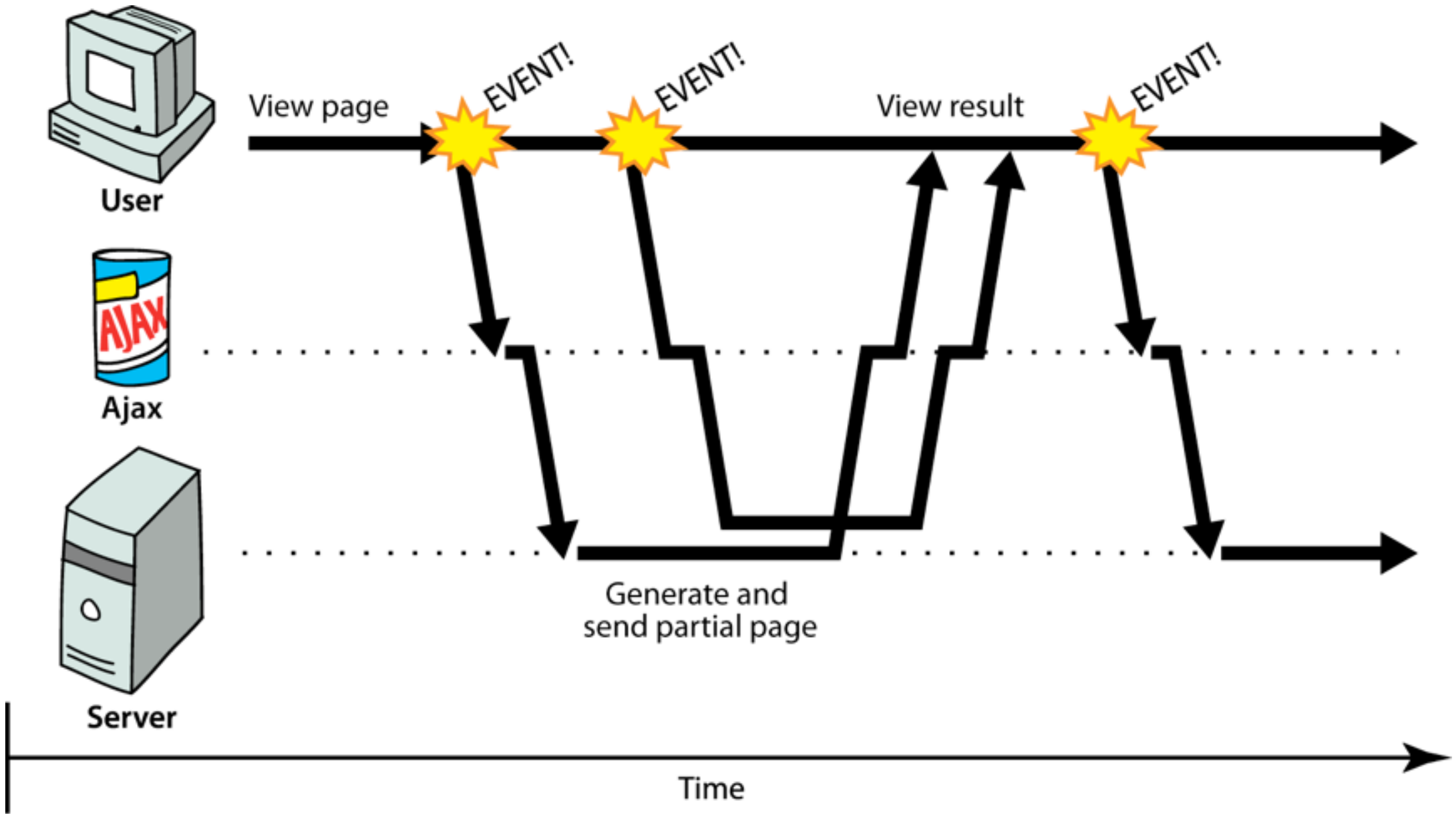


Image source: Web Programming Step by Step

Ajax: synchronous request

plain JS

```
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);
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

```
1  var ajax = new XMLHttpRequest();
2
3  //function to be called when the data has
4  arrived
5  ajax.onreadystatechange = function() {
6
7     //the only state we care about
8     if(ajax.readyState==4) {
9        /*
10        * process received data
11        */
12    }
13 }; //end of function
14
15 ajax.open("GET", "url", true); //true indicates
16                                 //asynchronous request
17
18 ajax.send(null);
```

Ajax: an asynchronous request

plain JS

```
1  var ajax = new XMLHttpRequest();
2
3  /                                  data has
4  a
5  ajax.onreadystatechange = function() {
6
7      //the only state we care about
8      if(ajax.readyState==4) {
9          /*
10          * process received data
11          */
12     }
13 }; //end of function
14
15 ajax.open("GET", "url", true); //true indicates
16                                 //asynchronous request
17
18 ajax.send(null);
```

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

What does the “same-origin” policy refer to?

A. A Web server permits scripts contained in page A to access data contained in page B only if they originate from the same client.
B. A Web browser permits scripts contained in page A to access data contained in page B only if they originate from the same server-side component.
C. A Web browser permits a Web server running on a URI scheme/hostname/port to only access data from a server running on the same host (on a different port).
D. A Web server permits browsers A and B to exchange data only if they originate from the same client.

Ajax security

- Conveniently we always requested data from "our" Web server
- **Security restriction of Ajax**: can only fetch files from the same Web server as the calling page (*Same-origin policy*)
  - Same origin when protocol, port, and host are the same for two pages
- Ajax **cannot** be executed from a Web page stored locally on disk

Ajax security

- Conveniently we always requested data from "our" Web server
- **Security restriction of Ajax**: can only fetch files from the same Web server as the calling page (*Same-origin policy*)
  - Same origin when protocol, port, and host are the same for two pages
- Ajax **cannot** be executed from a Web page stored locally on disk

The course book explains how to get around this restriction. Don't forget to switch this safety restriction back on!!!

Organization and reusability of node.js code

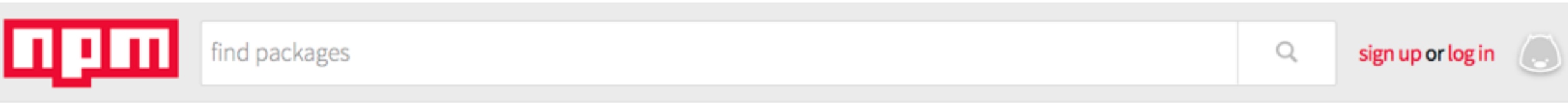
So far…

- All server-side code maintained **within a single file**
- Possible for small projects
- Larger projects suffer in this setting
  - **Debugging** is cumbersome
  - **Team-work** is cumbersome
  - **Programming** is cumbersome

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 the **npm**
  - Distribution of modules to other developers is made easy

node.js modules



```
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@1.0.5
│ │ │ ├┬ 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
│ │ ├┬ home-or-tmp@1.0.0
│ │ │ ├── os-tmpdir@1.0.1
│ │ │ └── user-home@1.1.1
│ │ └┬ source-map-support@0.2.10
│ │  └── source-map@0.1.32
│ ├┬ babel-runtime@6.11.6
│ │ └── regenerator-runtime@0.9.5
│ ├── babel-template@6.9.0
│ ├┬ babel-traverse@6.12.0
│ │ └── globals@8.18.0
│ ├┬ babel-types@6.11.1
│ │ └── to-fast-properties@1.0.2
│ ├── babylon@6.8.4
│ ├── convert-source-map@1.3.0
│ ├┬ debug@2.2.0
│ │ └── ms@0.7.1
```

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@1.0.5
│ │ │ ├┬ 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
│ │ ├┬ home-or-tmp@1.0.0
│ │ │ ├─ os-tmpdir@1.0.1
│ │ │ └─ user-home@1.1.1
│ │ └┬ source-map-support@0.2.10
│ │   └─ source-map@0.1.32
│ ├┬ babel-runtime@6.11.6
│ │ └─ regenerator-runtime@0.9.5
│ ├─ babel-template@6.9.0
│ ├┬ babel-traverse@6.12.0
│ │ └─ globals@8.18.0
│ ├┬ babel-types@6.11.1
│ │ └─ to-fast-properties@1.0.2
│ ├─ babylon@6.8.4
│ ├─ convert-source-map@1.3.0
│ ├┬ debug@2.2.0
│ │ └─ ms@0.7.1
```

npm list

5 lines (4 sloc) | 82 Bytes

```
1  'use strict';
2  module.exports = Number.isNaN || function (x) {
3          return x !== x;
4  };
```

```
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@1.0.5
│ │ │ ├┬ 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
```

npm list

5 lines (4 sloc) | 82 Bytes

```
1  'use strict';
2  module.exports = Number.isNaN || function (x) {
3          return x !== x;
4  };
```

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.

Ex.11 - Mark

- [x] Asynchronous
- [ ] Synchronous

Working on `Asynchronous'

```
1  var http = require('http');
2
3  http.createServer(function (req, res) {
4    setTimeout(function () {
5      res.writeHead(200, {'Content-Type': 'text/plain'});
6      res.end('Hello World');
7    }, 2000);
8  }).listen(8000);
9
10 console.log('Server running at http://127.0.0.1:8000/');
```

Ranges | Occurences

- var http = require('http'); 0:0 - 0:27
- http.createS 2:0 - 2:17
- setTimeout 3:2 - 3:12
- res.writeHead(200, {'Content-… 4:4 - 5:27
- console.log('Server running a… 9:0 - 9:56
- function (req, res) { 2:18 - 2:39

A file-based module system

A file-based module system

- **A file is its own module**; no pollution of the global namespace

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-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

A file-based module system

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

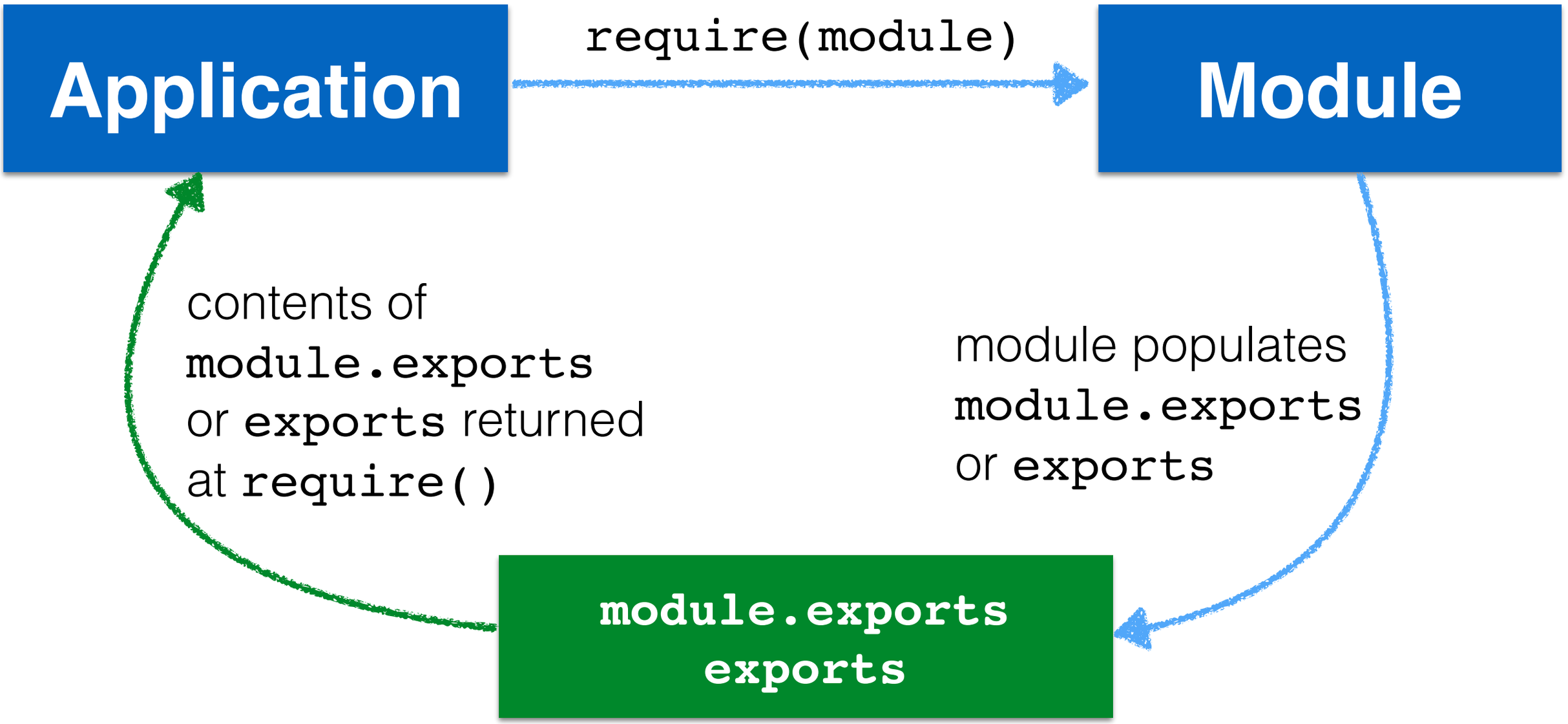
- **A file is its own module**; no pollution of the global namespace
- A file accesses its module definition through the **`module`** variable
- The export of the current module is determined by the **`module.exports`** variable (or its alias **`exports`**)

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.exports or exports



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);
```

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`**

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`**

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!
```

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!
```

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]
```

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]
```

A first example

foo.js

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

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

bar.js

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

```
Hi from foo!
```

```
[Function]
```

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

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!");
}
```

A first example

foo.js

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

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

bar.js

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

```
Hi from foo!
```

```
[Function]
```

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

```
ReferenceError
```

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
```

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

`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
```

constants.js

```
module.exports.pi = 3.1415;
module.exports.one = 1;
module.exports.login = "root";
module.exports.password = "root";
```

bar.js

```
var constants1 = require('./constants');
constants1.password = "admin";
var constants2 = require('./constants');
console.log(constants2.password);
var constants3 = require('./constants');
constants2.pi = 3;
console.log(constants3.pi);
```

What is the console output of `node bar.js` ?

`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`

- **`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');
};
```

equivalent

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

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

`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');
};
```

equivalent

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

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

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

constants.js

```
exports = function() {
   return {
      pi: 3.1415,
      one: 1,
      login: "root",
      password: "root"
   }
};
```

bar.js

```
var constants1 = require('./constants');
console.log(constants1);
constants1["password"] = "admin";
var constants2 = require('./constants');
console.log(constants2["password"]);
```

What is the console output of `node bar.js` ?

Creating a module

A module can be

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

```
 1 function roundGrade(grade) {
 2     return Math.round(grade);
 3 }
 4
 5 function roundGradeUp(grade) {
 6     return Math.round(0.5+parseFloat(grade));
 7 }
 8 exports.maxGrade = 10;
 9 exports.roundGradeUp = roundGradeUp;
10 exports.roundGradeDown = function(grade) {
11     return Math.round(grade-0.5);
12 }
```

grades.js

Creating a module

A module can be

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

```
 1 function roundGrade(grade) {
 2     return Math.round(grade);
 3 }
 4
 5 function roundGradeUp(grade) {
 6     return Math.round(0.5+parseFloat(grade));
 7 }
 8 exports.maxGrade = 10;
 9 exports.roundGradeUp = roundGradeUp;
10 exports.roundGradeDown = function(grade) {
11     return Math.round(grade-0.5);
12 }
```

not exposed in this module;
application cannot use it

grades.js

Creating a module

A module can be

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

```
 1 function roundGrade(grade) {
 2     return Math.round(grade);
 3 }
 4
 5 function roundGradeUp(grade) {
 6     return Math.round(0.5+parseFloat(grade));
 7 }
 8 exports.maxGrade = 10;
 9 exports.roundGradeUp = roundGradeUp;
10 exports.roundGradeDown = function(grade) {
11     return Math.round(grade-0.5);
12 }
```

not exposed in this module;
application cannot use it

determines what exactly is
exposed to the outer world

Using a module

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var grading = require("./grades");
5  var app;
6
7  var port = process.argv[2];
8  app = express();
9  http.createServer(app).listen(port);
10
11 app.get("/round", function (req, res) {
12   var query = url.parse(req.url, true).query;
13   var grade = ( query["grade"]!=undefined) ?
14                 query["grade"] : "0";
15   res.send("Rounding up: " +
16            grading.roundGradeUp(grade) +", and down: "+
17            grading.roundGradeDown(grade));
18 });
```

Using a module

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4 var grading = require("./grades");
5 var app;
6
7 var port = process.argv[2];
8 app = express();
9 http.createServer(app).listen(port);
10
11 app.get("/round", function (req, res) {
12  var query = url.parse(req.url, true).query;
13  var grade = ( query["grade"]!=undefined) ?
14                query["grade"] : "0";
15  res.send("Rounding up: " +
16           grading.roundGradeUp(grade) +", and down: "+
17           grading.roundGradeDown(grade));
18 });
```

adding our module (current directory)

Using a module

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var grading = require("./grades");
5  var app;
6
7  var port = process.argv[2];
8  app = express();
9  http.createServer(app).listen(port);
10
11 app.get("/round", function (req, res) {
12   var query = url.parse(req.url, true).query;
13   var grade = ( query["grade"]!=undefined) ?
14                 query["grade"] : "0";
15   res.send("Rounding up: " +
16            grading.roundGradeUp(grade) +", and down: "+
17            grading.roundGradeDown(grade));
18 });
```

adding our module (current directory)

accessing module functions

Using a module

require returns the contents of the exports object

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var grading = require("./grades");
5  var app;
6
7  var port = process.argv[2];
8  app = express();
9  http.createServer(app).listen(port);
10
11 app.get("/round", function (req, res) {
12   var query = url.parse(req.url, true).query;
13   var grade = ( query["grade"]!=undefined) ?
14                 query["grade"] : "0";
15   res.send("Rounding up: " +
16            grading.roundGradeUp(grade) +", and down: "+
17            grading.roundGradeDown(grade));
18 });
```

adding our module (current directory)

accessing module functions

constants.js

```
module.exports = function() {
    return {
        pi: 3.1415,
        one: 1,
        login: "root",
        password: "root"
    }
};
```

bar.js

```
var constants1 = require('./constants')();
constants1["password"] = "admin";
var constants2 = require('./constants')();
console.log(constants2["password"]);
var constants3 = require('./constants')();
constants2["pi"] = 3;
console.log(constants3["pi"]);
```

What is the console output of `node bar.js` ?

Express and Connect

What is Connect?

What is Connect?

- **Connect**: **framework** whose components ("**middleware**") are used to create Web application logics (inspired by Ruby's Rack framework)

What is Connect?

- **Connect**: **framework** whose components ("**middleware**") are used to create Web application logics (inspired by Ruby's Rack framework)
- Middleware: **function** which **intercepts** the HTTP request & response objects, **executes logic** and ends the response or passes them to the next component

What is Connect?

- **Connect**: **framework** whose components ("**middleware**") are used to create Web application logics (inspired by Ruby's Rack framework)
- Middleware: **function** which **intercepts** the HTTP request & response objects, **executes logic** and ends the response or passes them to the next component
- **Dispatcher** connects the components (thus "Connect")

What is Connect?

- **Connect**: **framework** whose components ("**middleware**") are used to create Web application logics (inspired by Ruby's Rack framework)
- Middleware: **function** which **intercepts** the HTTP request & response objects, **executes logic** and ends the response or passes them to the next component
- **Dispatcher** connects the components (thus "Connect")
- Typical components: request logging, request body parsing, session managing, etc.

What is Connect?

Express is built on top of Connect.

- **Connect**: **framework** whose components ("**middleware**") are used to create Web application logics (inspired by Ruby's Rack framework)
- Middleware: **function** which **intercepts** the HTTP request & response objects, **executes logic** and ends the response or passes them to the next component
- **Dispatcher** connects the components (thus "Connect")
- Typical components: request logging, request body parsing, session managing, etc.

What is Connect?

Express is built on top of Connect. **True up to express@3.**

- **Connect**: **framework** whose components ("**middleware**") are used to create Web application logics (inspired by Ruby's Rack framework)
- Middleware: **function** which **intercepts** the HTTP request & response objects, **executes logic** and ends the response or passes them to the next component
- **Dispatcher** connects the components (thus "Connect")
- Typical components: request logging, request body parsing, session managing, etc.

What is Connect?

Express is built on top of Connect. **True up to express@3.**

Express and Connect are now decoupled (express@4), in practice few things have changed.

- Middleware: **function** which **intercepts** the HTTP request & response objects, **executes logic** and ends the response or passes them to the next component
- **Dispatcher** connects the components (thus “Connect”)
- Typical components: request logging, request body parsing, session managing, etc.

Connect example

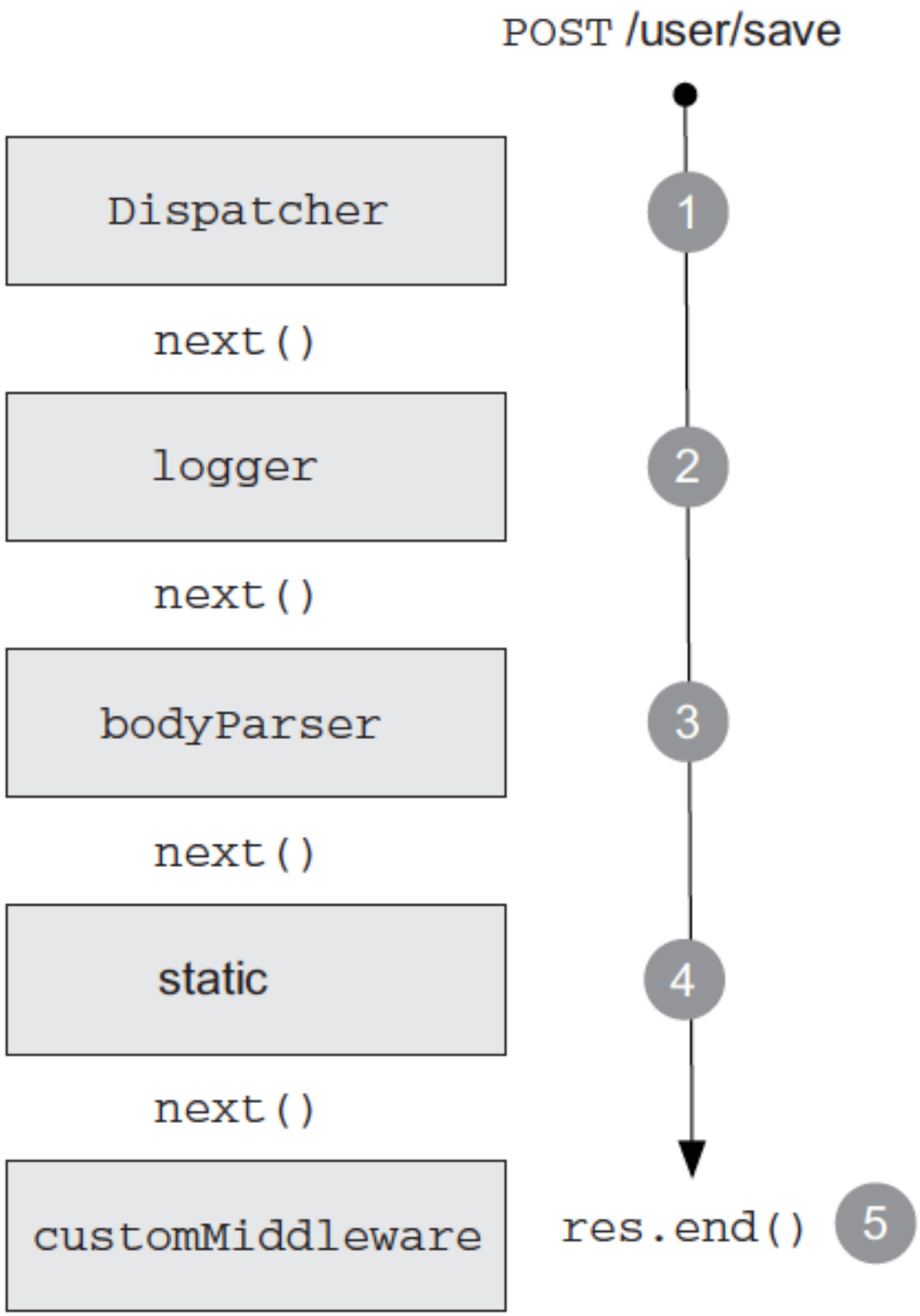
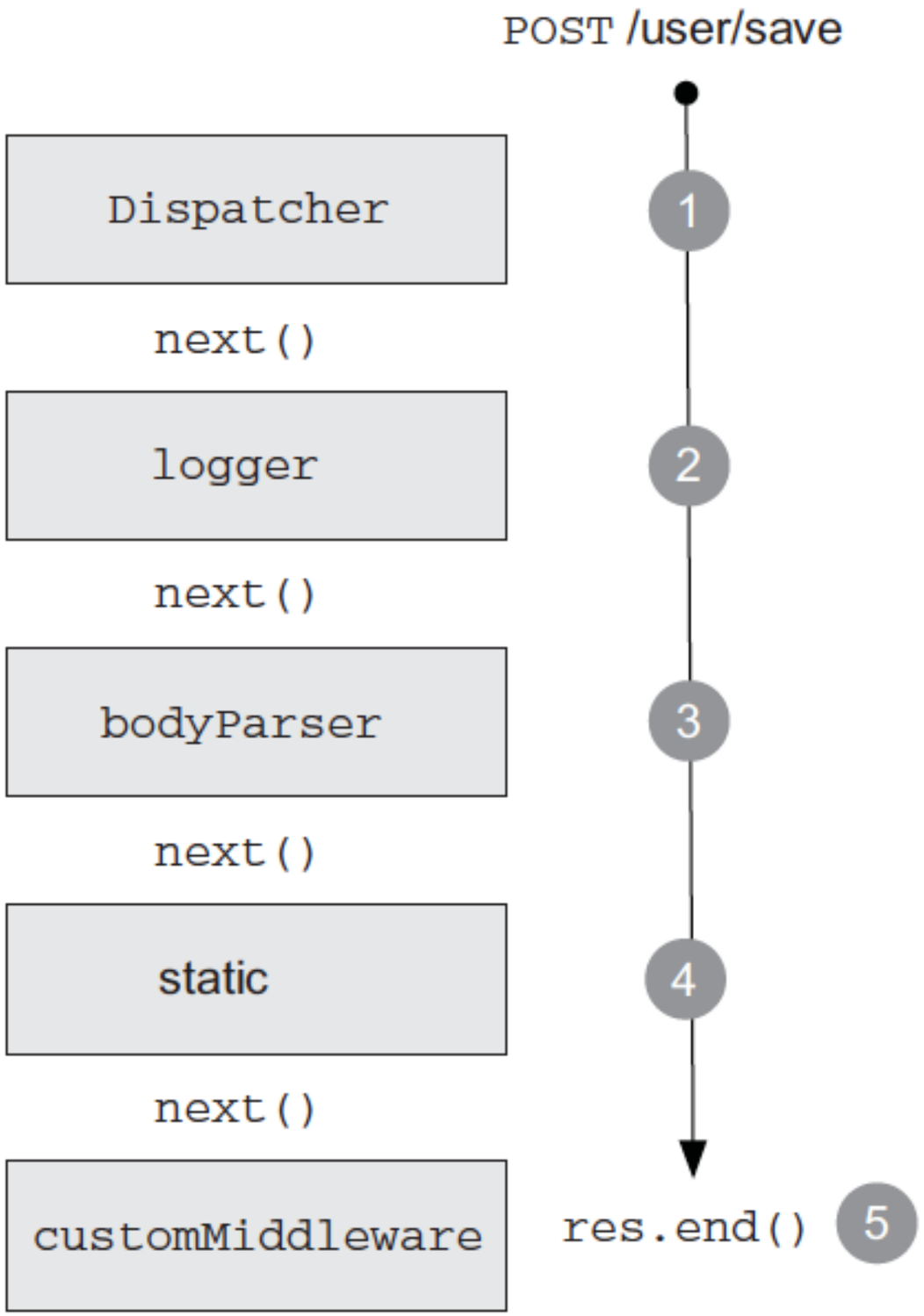


Image source: *Node.js in Action*, page 124

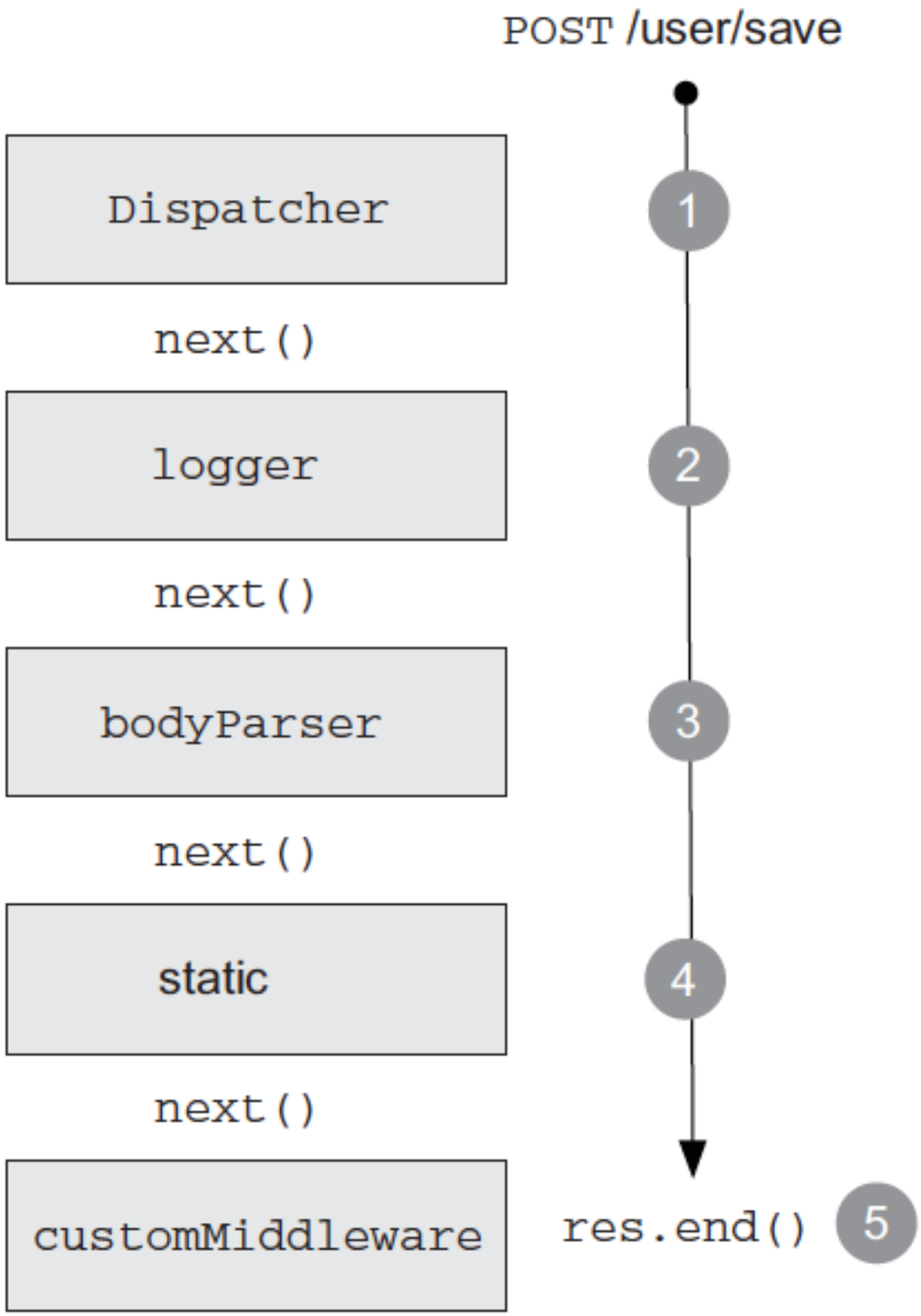
Connect example



1. Dispatcher receives request and passes it to the first middleware

Image source: *Node.js in Action*, page 124

Connect example

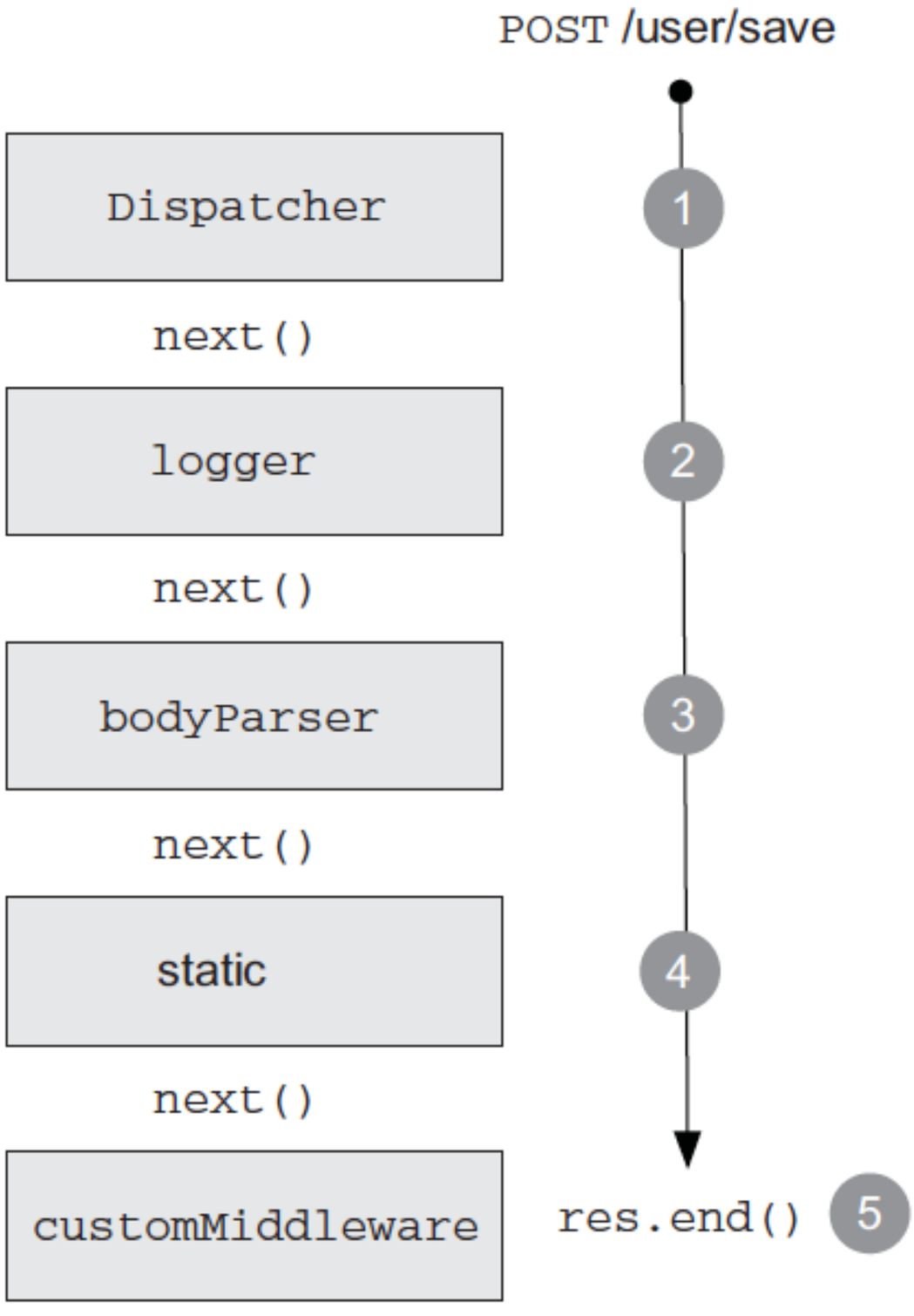


1. Dispatcher receives request and passes it to the first middleware

2. Request is logged and passed to the next middleware using next()

Image source: *Node.js in Action*, page 124

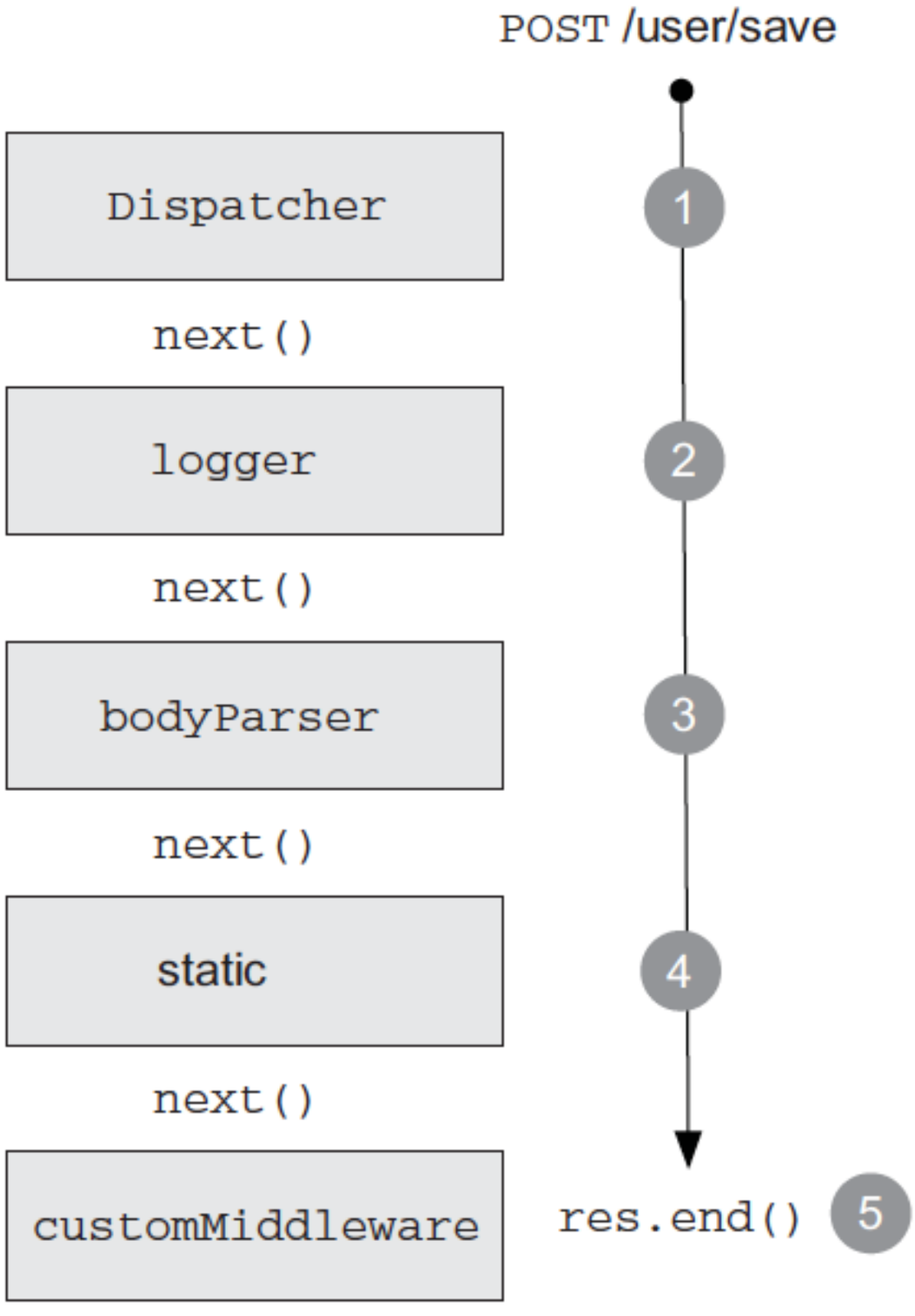
Connect example



1. Dispatcher receives request and passes it to the first middleware
2. Request is logged and passed to the next middleware using next()
3. Request body is parsed if any exists and then passed to the next middleware using next()

Image source: *Node.js in Action*, page 124

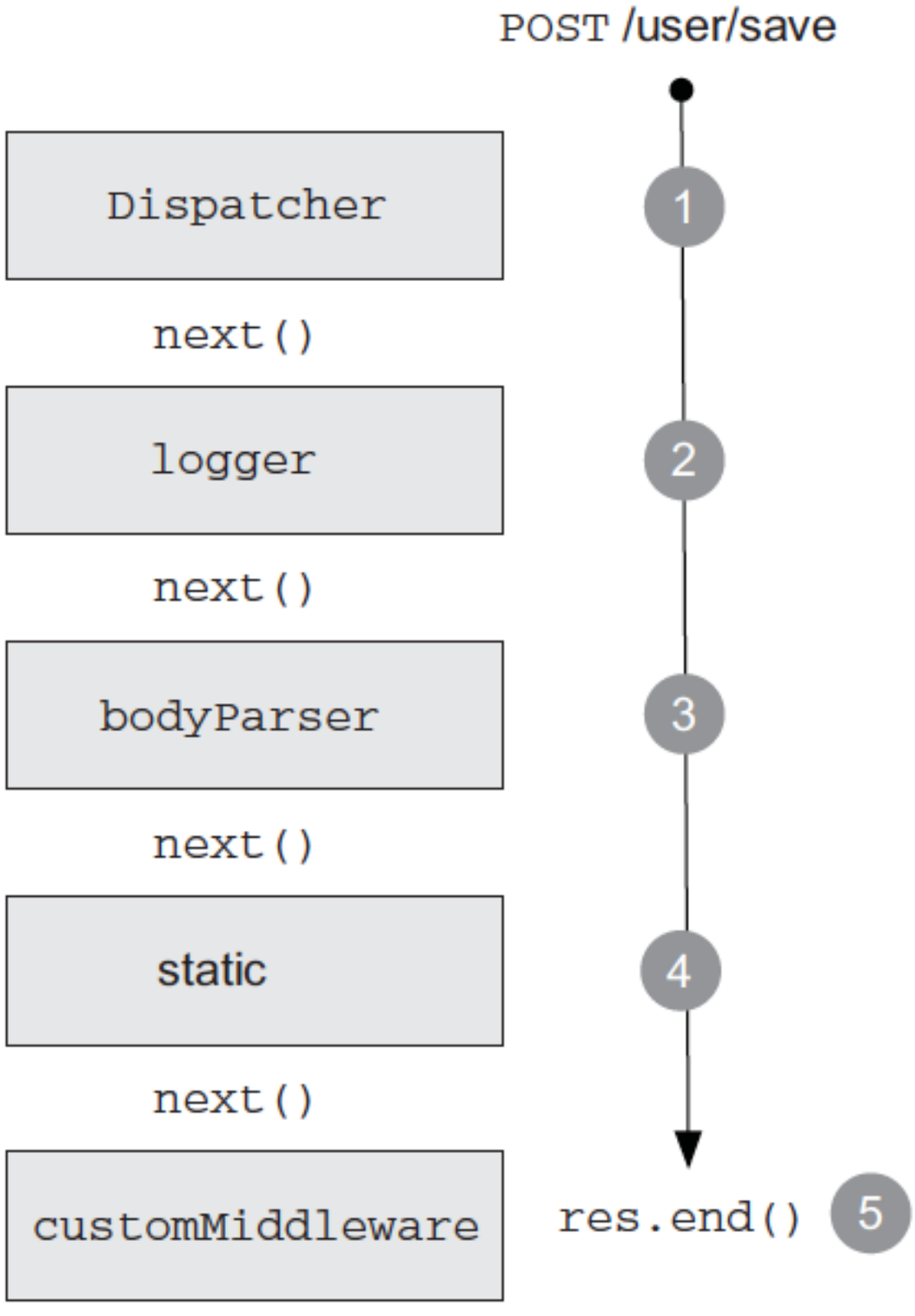
Connect example



1. Dispatcher receives request and passes it to the first middleware
2. Request is logged and passed to the next middleware using next()
3. Request body is parsed if any exists and then passed to the next middleware using next()
4. If request is for a static file, response is sent with that file and next() is not called; otherwise, the request moves to the next middleware

Image source: *Node.js in Action*, page 124

Connect example



1. Dispatcher receives request and passes it to the first middleware
2. Request is logged and passed to the next middleware using next()
3. Request body is parsed if any exists and then passed to the next middleware using next()
4. If request is for a static file, response is sent with that file and next() is not called; otherwise, the request moves to the next middleware
5. Request is handled with a custom middleware and response is ended

Image source: *Node.js in Action*, page 124

Middleware components

Middleware components

- Middleware components 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 components

- Middleware components 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 components are **small**, **self-contained** and **reusable** across applications

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

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

```
 1 var connect = require('connect');
 2
 3 //a middleware logger component
 4 function logger(request, response, next) {
 5     console.log('%s\t%s\t%s', new Date(),
 6                 request.method, request.url);
 7     next();
 8 }
 9 var app = connect();
10 app.use(logger);
11 app.listen(3001);
```

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

```
1  var connect = require('connect');
2
3  //a middleware logger component
4  function logger(request, response, next) {
5      console.log('%s\t%s\t%s', new Date(),
6                  request.method, request.url);
7      next();
8  }
9  var app = connect();
10 app.use(logger);
11 app.listen(3001);
```

control returns to the dispatcher

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

```
1  var connect = require('connect');
2
3  //a middleware logger component
4  function logger(request, response, next) {
5      console.log('%s\t%s\t%s', new Date(),
6                  request.method, request.url);
7      next();
8  }
9  var app = connect();
10 app.use(logger);
11 app.listen(3001);
```

control returns to the dispatcher

register middleware component

Example: a simple logger

connect.js - Example8

EXPLORE

WORKING FILES

EXAMPLE8

.vscode

launch.json

connect.js

jsconfig.json

connect.js

```
1  var connect = require('connect');
2
3  //middleware 1
4  function logger(request, response, next) {
5      console.log('%s\t%s\t%s',
6          new Date(), request.method,
7          request.url);
8      next();
9  }
10 //middleware 2
11 function delimiter(request, response, next) {
12     console.log('+++++');
13     next();
14 }
15
16 var app = connect();
17 app.use(logger);
18 app.use(delimiter);
19 app.listen(4567);
```

1 0 Ln 8, Col 5 UTF-8 LF JavaScript

localhost:4567 Search

Example: a simple logger

connect.js - Example8

EXPLORE

WORKING FILES

EXAMPLE8

.vscode

launch.json

connect.js

jsconfig.json

connect.js

```
1  var connect = require('connect');
2
3  //middleware 1
4  function logger(request, response, next) {
5      console.log('%s\t%s\t%s',
6          new Date(), request.method,
7          request.url);
8      next();
9  }
10 //middleware 2
11 function delimiter(request, response, next) {
12     console.log('+++++');
13     next();
14 }
15
16 var app = connect();
17 app.use(logger);
18 app.use(delimiter);
19 app.listen(4567);
```

1 0 Ln 8, Col 5 UTF-8 LF JavaScript

localhost:4567 Search

Example: a simple logger and “Hello World” response

QuickTime Player File Edit View Window Help 71% Thu 22:47 Claudia Hauff

connect.js - Example8

Launch

VARIABLES

WATCH

CALL STACK

BREAKPOINTS

All exceptions

Uncaught exceptions

connect.js

```
1  var connect = require('connect');
2
3  //middleware 1
4  function logger(request, response, next) {
5      console.log('%s\t%s\t%s',
6          new Date(), request.method,
7          request.url);
8      next();
9  }
10
11
12 var app = connect();
13 app.use(logger);
14 app.listen(5678);
```

1 0 Ln 10, Col 1 UTF-8 LF JavaScript

New Tab

localhost:5678

Search

Example: a simple logger and “Hello World” response

QuickTime Player File Edit View Window Help 71% Thu 22:47 Claudia Hauff

connect.js - Example8

Launch

VARIABLES

WATCH

CALL STACK

BREAKPOINTS

All exceptions

Uncaught exceptions

connect.js

```
1  var connect = require('connect');
2
3  //middleware 1
4  function logger(request, response, next) {
5      console.log('%s\t%s\t%s',
6          new Date(), request.method,
7          request.url);
8      next();
9  }
10
11
12 var app = connect();
13 app.use(logger);
14 app.listen(5678);
```

1 0 Ln 10, Col 1 UTF-8 LF JavaScript

New Tab

localhost:5678

Search

A simple logger and “Hello World” response

```
1 var connect = require('connect');
2
3 function logger(request, response, next) { .... }
4
5 function helloWorld(request, response, next) {
6     response.setHeader('Content-Type',
7                        'text/plain');
8     response.end('Hello World!');
9 }
10
11 var app = connect();
12 app.use(logger);
13 app.use(helloWorld);
14 app.listen(3001);
```

A simple logger and “Hello World” response

```
1  var connect = require('connect');
2
3  function logger(request, response, next) { .... }
4
5  function helloWorld(request, response, next) {
6      response.setHeader('Content-Type',
7                         'text/plain');
8      response.end('Hello World!');
9  }
10
11 var app = connect();
12 app.use(logger);
13 app.use(helloWorld);
14 app.listen(3001);
```

No call to next! Control is not returned to dispatcher

A simple logger and “Hello World” response

```
1  var connect = require('connect');
2
3  function logger(request, response, next) { .... }
4
5  function helloWorld(request, response, next) {
6      response.setHeader('Content-Type',
7                         'text/plain');
8      response.end('Hello World!');
9  }
10
11 var app = connect();
12 app.use(logger);
13 app.use(helloWorld);
14 app.listen(3001);
```

No call to next! Control is not returned to dispatcher

any number of components can be registered

A simple logger and “Hello World” response

```
1  var connect = require('connect');
2
3  function logger(request, response, next) { .... }
4
5  function helloWorld(request, response, next) {
6      response.setHeader('Content-Type',
7                         'text/plain');
8      response.end('Hello World!');
9  }
10
11 var app = connect();
12 app.use(logger);
13 app.use(helloWorld);
14 app.listen(3001);
```

No call to next! Control is not returned to dispatcher

any number of components can be registered

their order matters

Example 8

Example: an authorisation component in Express

todo-server.js - Example8

EXPLORE

- WORKING FILES
- EXAMPLE8
  - .vscode
    - launch.json
  - client
  - jsconfig.json
  - todo-server.js

todo-server.js

```
1  /* global Buffer */
2  /* global __dirname */
3  //note: a close copy of Example6, only two middleware components added
4  var express = require("express");
5  var url = require("url");
6  var http = require("http");
7
8  var port = 3000;
9  var app = express();
10 app.use(express.static(__dirname + "/client"));
11
12 http.createServer(app).listen(port);
13
14 var todos = [];
15 var t1 = { message : "Maths homework due", type  : 1, deadline : "12/12/2015"};
16 var t2 = { message : "English homework due", type : 3, deadline : "20/12/2015"};
17 todos.push(t1);
18 todos.push(t2);
19
20 //clients requests todos
21 app.get("/todos", function (req, res) {
22     console.log("todos requested!");
```

claudiahauff — bash — Solarized Dark xterm-256color — 128×7

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

Example 8

Example: an authorisation component in Express

todo-server.js - Example8

EXPLORE

- WORKING FILES
- EXAMPLE8
  - .vscode
    - launch.json
  - client
  - jsconfig.json
  - todo-server.js

todo-server.js

```
1  /* global Buffer */
2  /* global __dirname */
3  //note: a close copy of Example6, only two middleware components added
4  var express = require("express");
5  var url = require("url");
6  var http = require("http");
7
8  var port = 3000;
9  var app = express();
10 app.use(express.static(__dirname + "/client"));
11
12 http.createServer(app).listen(port);
13
14 var todos = [];
15 var t1 = { message : "Maths homework due", type  : 1, deadline : "12/12/2015"};
16 var t2 = { message : "English homework due", type : 3, deadline : "20/12/2015"};
17 todos.push(t1);
18 todos.push(t2);
19
20 //clients requests todos
21 app.get("/todos", function (req, res) {
22     console.log("todos requested!");
```

claudiahauff — bash — Solarized Dark xterm-256color — 128×7

```
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

```
1 function setup(options) {
2     // setup logic
3     return function(req, res, next) {
4         // middleware logic
5     }
6 }
7
8 app.use( setup({ param1 : 'value1' }) );
```

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

```
1 function setup(options) {
2     // setup logic
3     return function(req, res, next) {
4         // middleware logic
5     }
6 }
7
8 app.use( setup({ param1 : 'value1' }) );
```

important: function call is made!

Routing

Introduction

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`

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`

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

Routes and Express

- You have used simple routes already

Routes and Express

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

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);
});
```

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);
});
```

Two route handlers defined for a route

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

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

Which route does this handler <u>not</u> match?

Routes and Express

```
//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);
   }
);
```

Routes and Express

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

```
//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);
   }
);
```

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

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

+	one or more occurrences
?	zero or one occurrence
*	zero or more occurrences of any character (wildcard)
[…]	match anything inside for one character position
(…)	boundaries

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);
});
```

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);
});
```

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);
});
```

Routing parameters

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

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

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);
});
```

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

Routing parameters

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

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

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);
});
```

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

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

Routing parameters

```
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);
});
```

Routing parameters

```
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);
});
```

Routing parameters

```
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

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


```
app.get('/whaa+[dt]s+upp*', function(req,res){
    res.send(…)
});
```

Which route does this handler not match?

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 …

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4 var app;
5
6 var port = process.argv[2];
7 app = express();
8 http.createServer(app).listen(port);
9
10 app.get("/greetme", function (req, res) {
11  var query = url.parse(req.url, true).query;
12  var name = ( query["name"]!=undefined) ? query[
13                "name"] : "Anonymous";
14  res.send("<html><head></head><body><h1>
15            Greetings "+name+"</h1></body></html>
16            ");
17 });
18
19 app.get("/goodbye", function (req, res) {
20  res.send("Goodbye you!");
21 });
```

Express and HTML …

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4 var app;
5
6 var port = process.argv[2];
7 app = express();
8 http.createServer(app).listen(port);
9
10 app.get("/greetme", function (req,
11  var query = url.parse(req.url, tr
12  var name = ( query["name"]!=undef
13              "name"] : "Anonymous";
14  res.send("<html><head></head><body><h1>
15           Greetings "+name+"</h1></body></html>
16           ");
17 });
18
19 app.get("/goodbye", function (req, res) {
20  res.send("Goodbye you!");
21 });
```

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

Instead … templates

Instead … templates

- **Goal:** write as little HTML “by hand” as possible

Instead … templates

- **Goal:** write as little HTML “by hand” as possible

Instead … templates

- **Goal:** write as little HTML “by hand” as possible

- 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

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)

Model-View-Controller (MVC)

- Standard design pattern to keep **logic**, **data** and **presentation** separate

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)

Model-View-Controller (MVC)

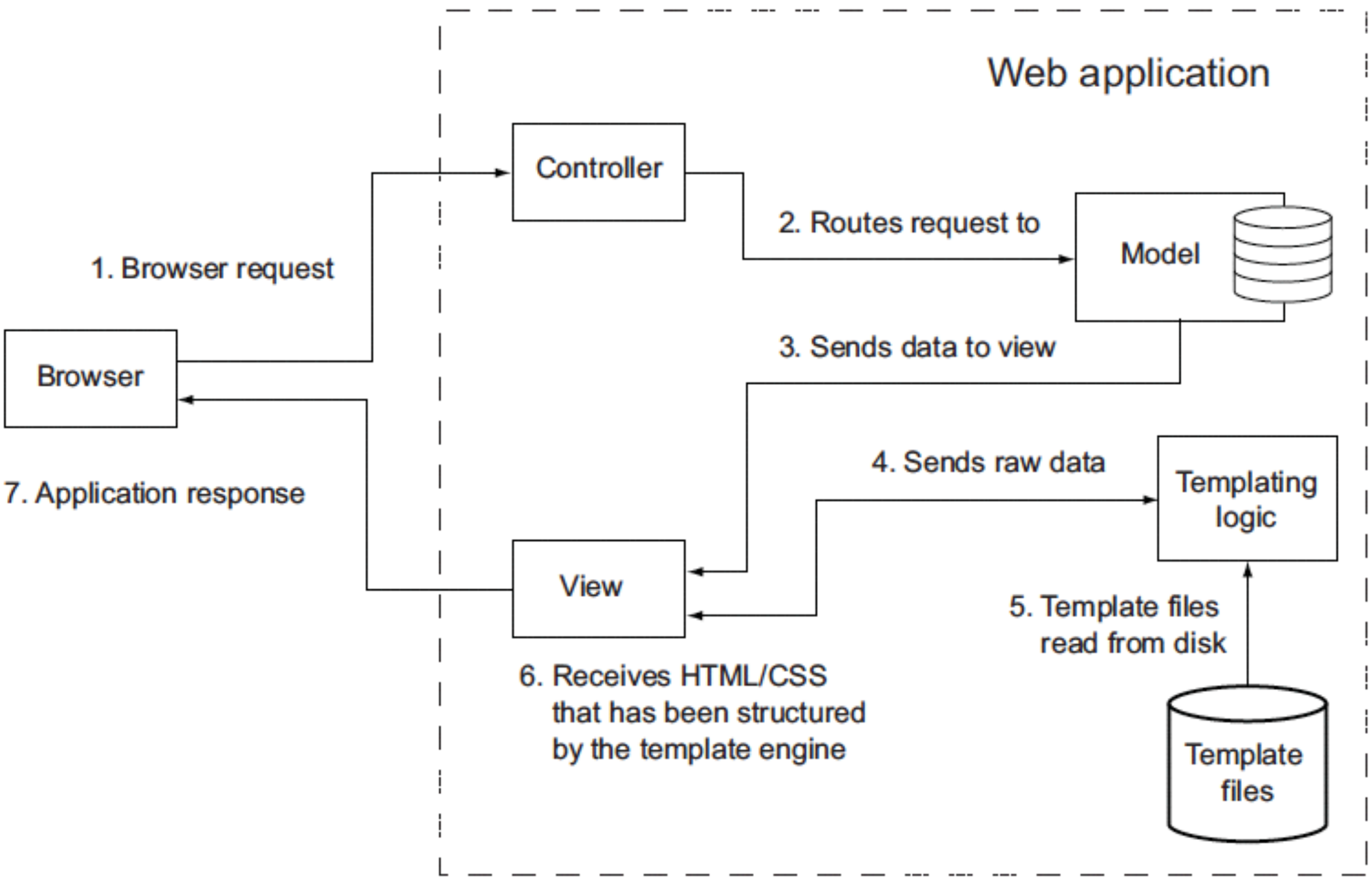
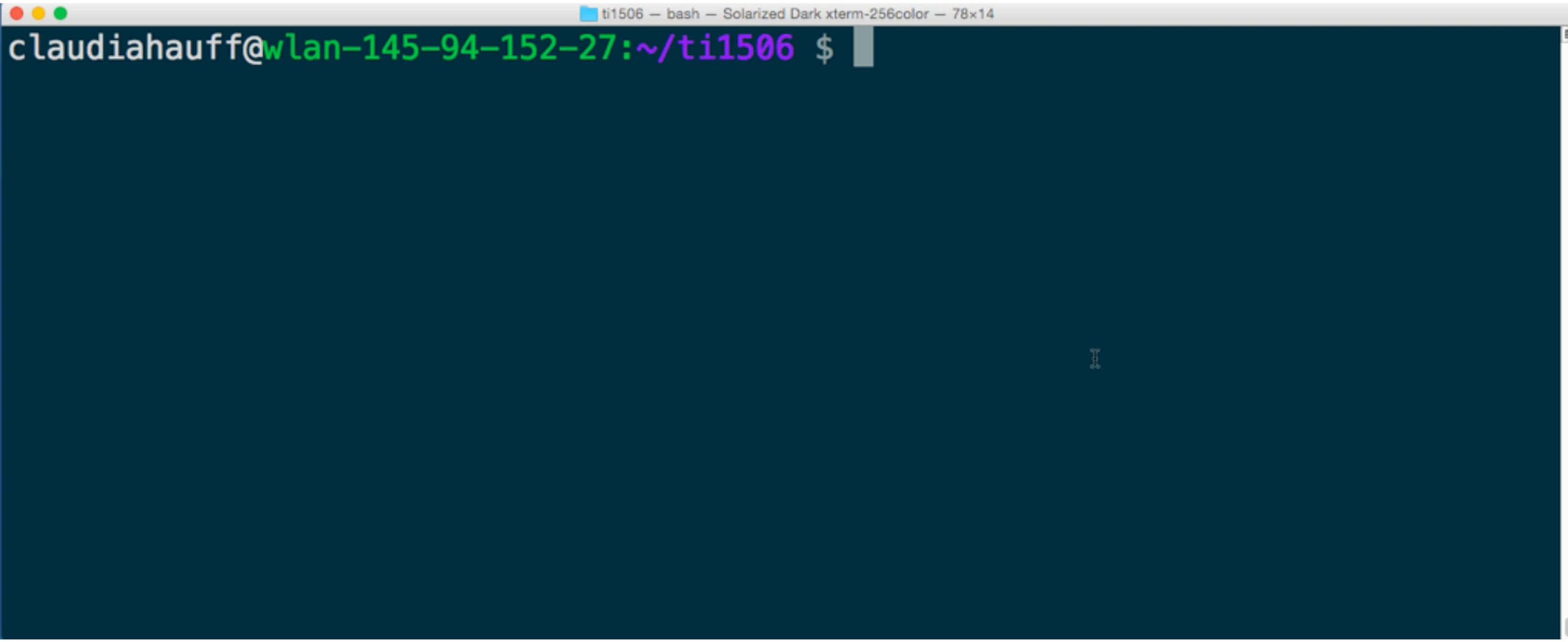


Image source: Node.js in Action, page 265

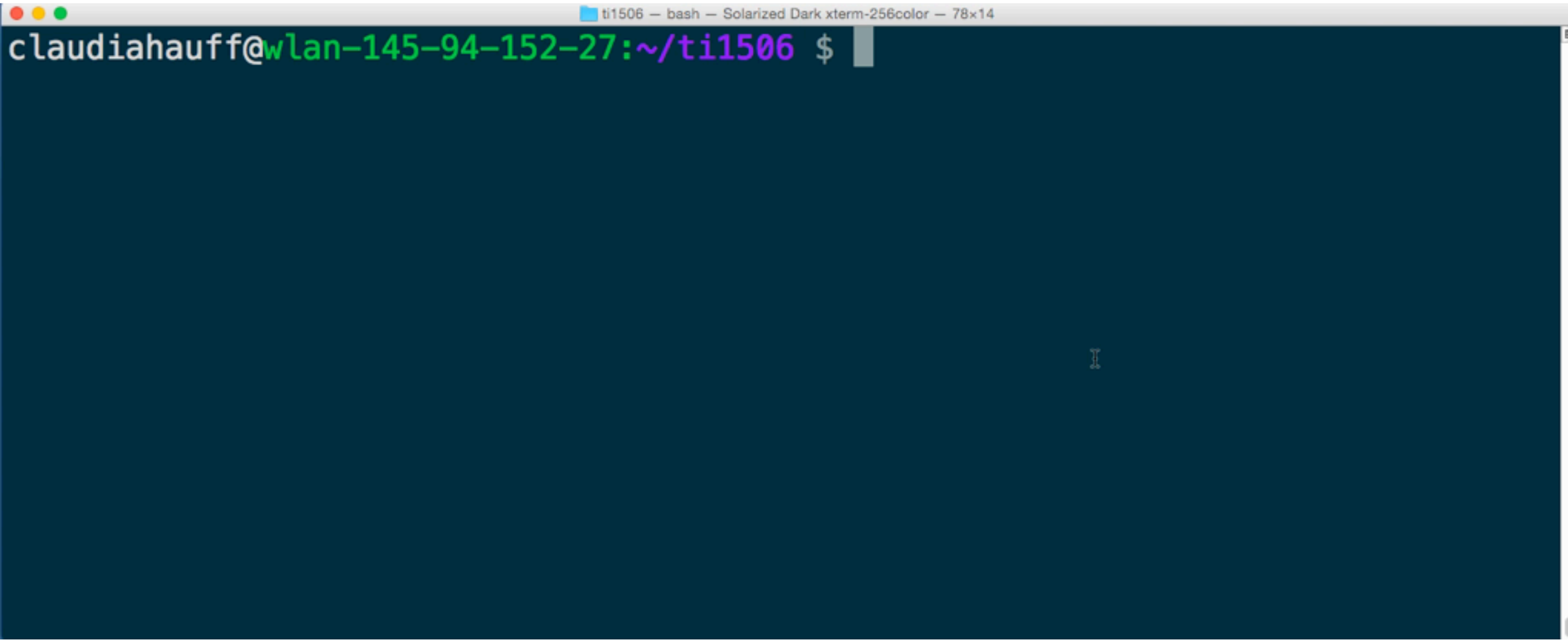
npm install ejs

A first look at ejs I



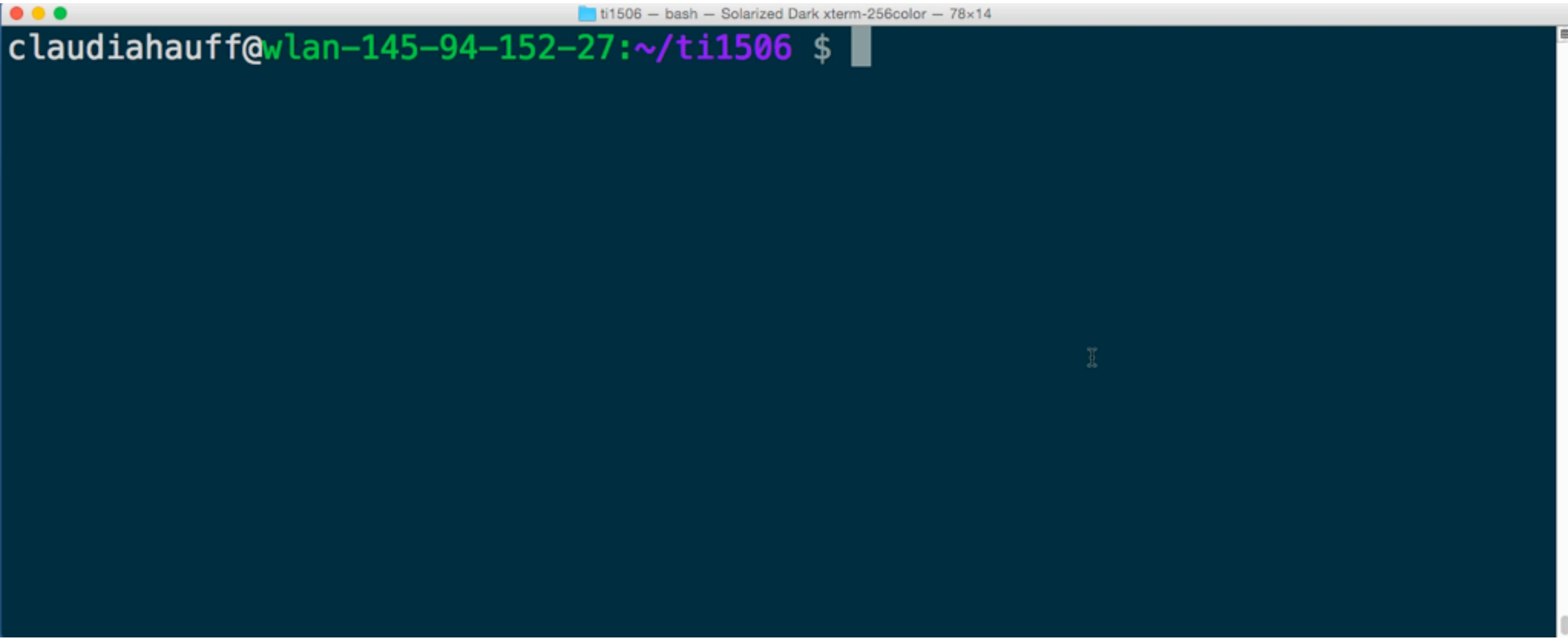
npm install ejs

A first look at ejs I



npm install ejs

A first look at ejs I



ejs1.js

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

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

npm install ejs

A first look at ejs I

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

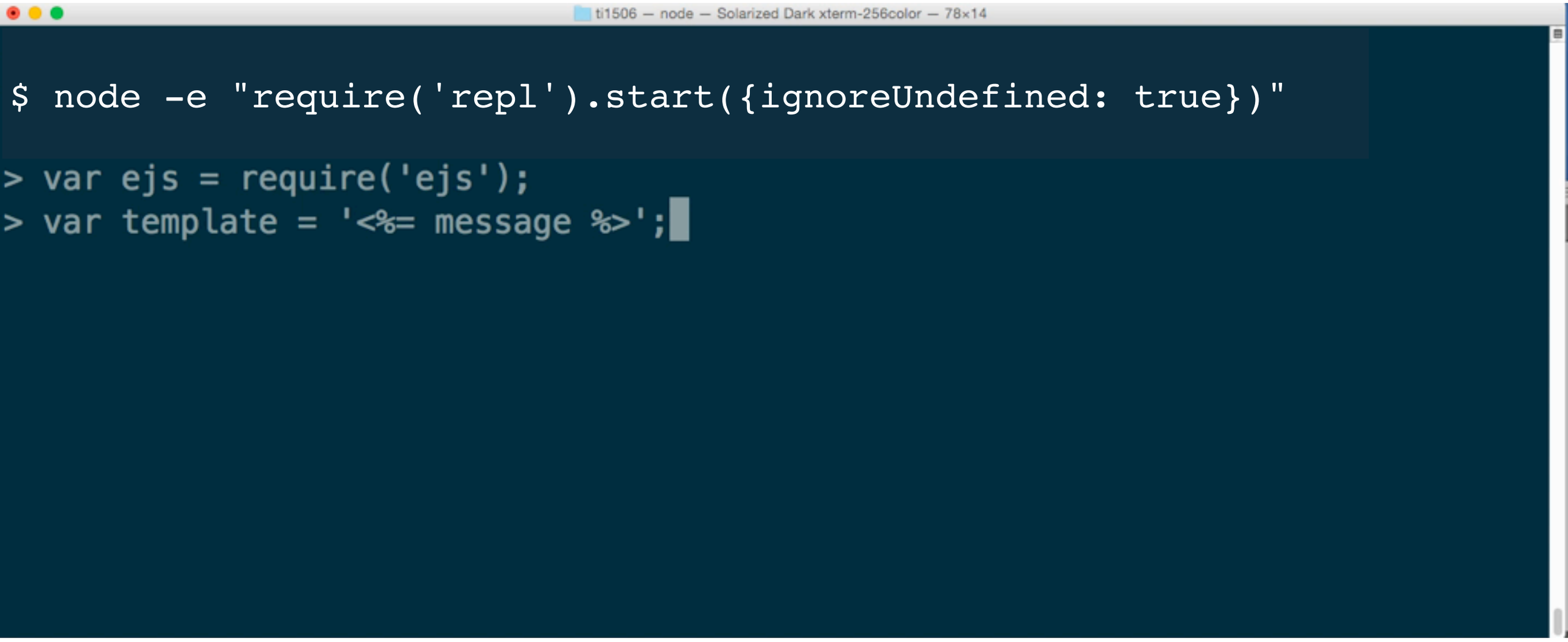
ejs1.js

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

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

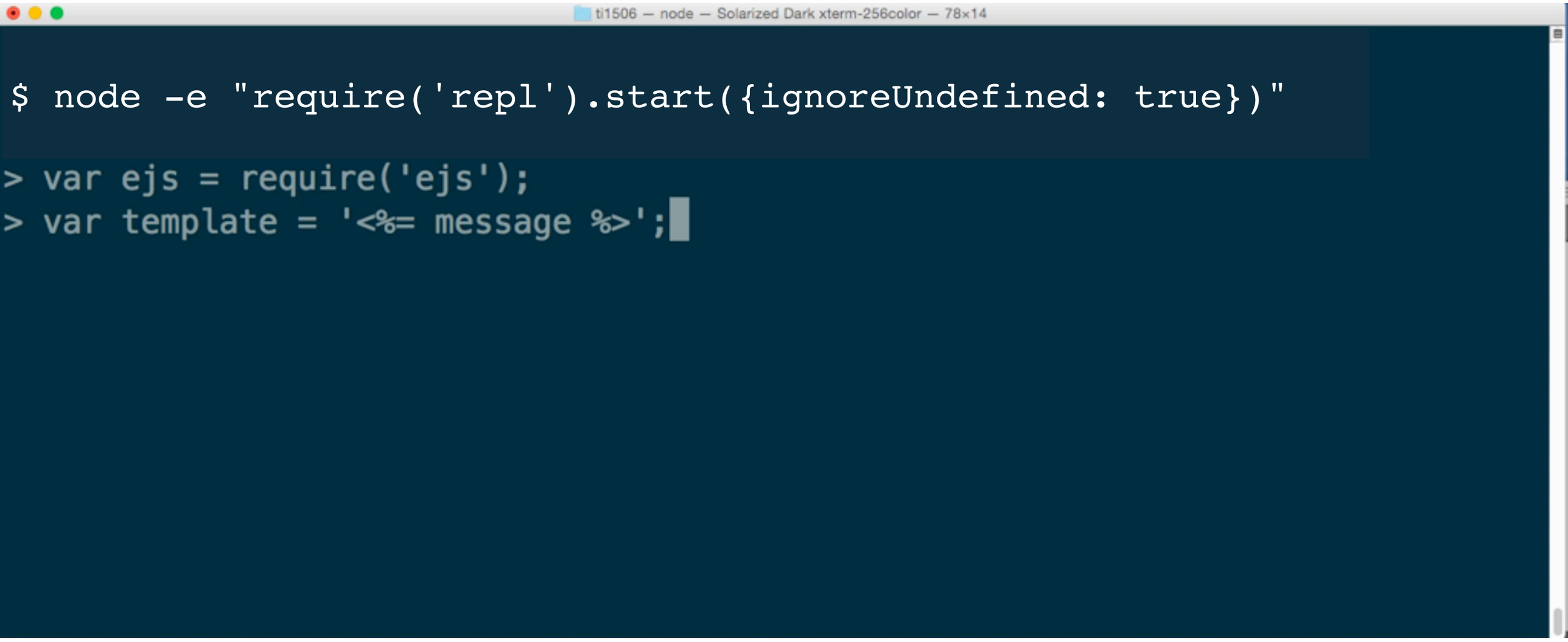
npm install ejs

A first look at ejs II



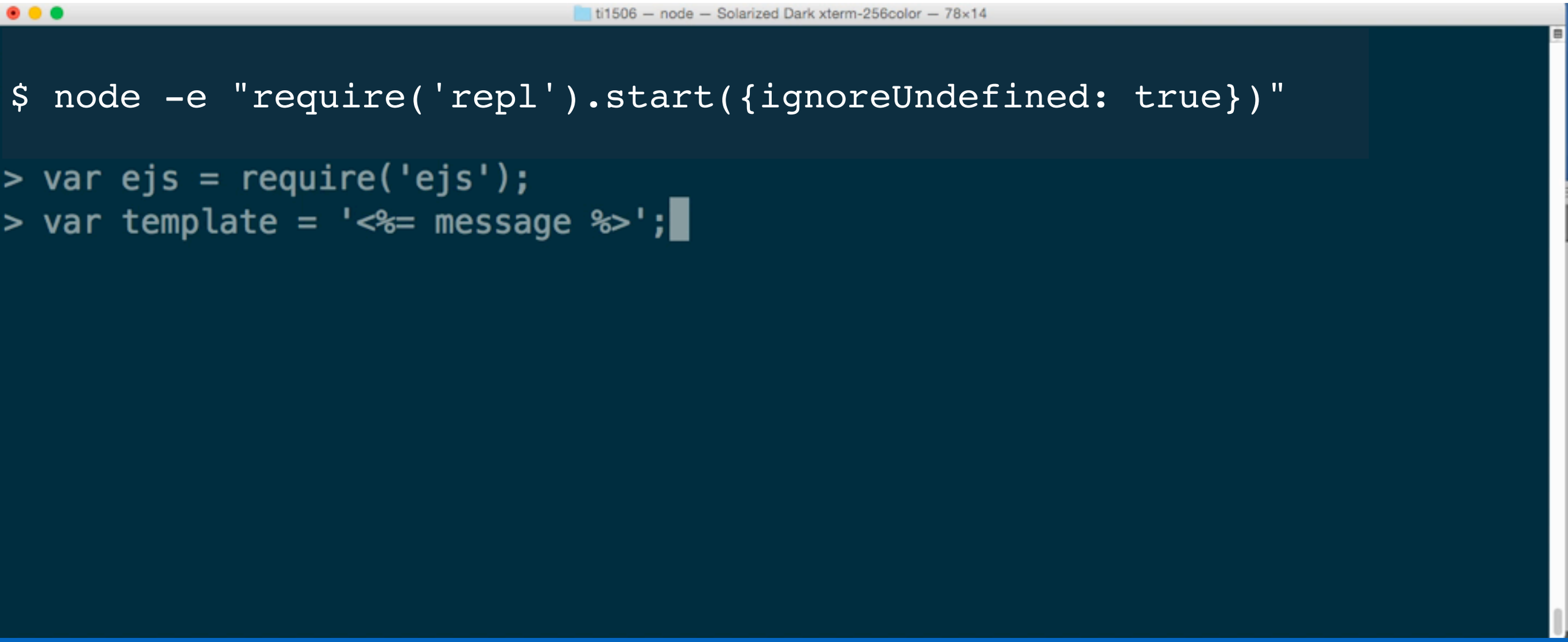
npm install ejs

A first look at ejs II



npm install ejs

A first look at ejs II



By default, ejs escapes special values in the context.

npm install ejs

A first look at ejs II

ti1506 — node — Solarized Dark xterm-256color — 78×14

```
$ node -e "require('repl').start({ignoreUndefined: true})"

> var ejs = require('ejs');
> var template = '<%= message %>';
```

By default, ejs escapes special values in the context.

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

npm install ejs

A first look at ejs II

ti1506 — node — Solarized Dark xterm-256color — 78×14

```
$ node -e "require('repl').start({ignoreUndefined: true})"

> var ejs = require('ejs');
> var template = '<%= message %>';
```

By default, ejs escapes ext.

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

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

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = [‘wolverine', 'paul', 'picard'];
4  var transformUpper = function (inputString) {
5                         return inputString.toUpperCase();
6    }
7
8  var template = '<%= helperFunc(input.join(", "))
9                   ; %>';
10 var context = {
11   input: people,
12   helperFunc: transformUpper
13 };
14
15 console.log(ejs.render(template, context));
```

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = [‘wolverine', 'paul', 'picard'];
4  var transformUpper = function (inputString) {
5                         return inputString.toUpperCase();
6  }
7
8  var template = '<%= helperFunc(input.join(", "))
9                   ; %>';
10 var context = {
11   input: people,
12   helperFunc: transformUpper
13 };
14
15 console.log(ejs.render(template, context));
```

define your own function

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = [‘wolverine', 'paul', 'picard'];
4  var transformUpper = function (inputString) {
5                         return inputString.toUpperCase();
6  }
7
8  var template = '<%= helperFunc(input.join(", "))
9                   ; %>';
10 var context = {
11   input: people,
12   helperFunc: transformUpper
13 };
14
15 console.log(ejs.render(template, context));
```

define your own function

function

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = ['wolverine', 'paul', 'picard'];
4  var transformUpper = function (inputString) {
5                         return inputString.toUpperCase();
6  };
7
8  var template = '<%= helperFunc(input.join(", "))
9                  ; %>';
10 var context = {
11   input: people,
12   helperFunc: transformUpper
13 };
14
15 console.log(ejs.render(template, context));
```

define your own function

function

array

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = ['wolverine', 'paul', 'picard'];
4  var transformUpper = function (inputString) {
5                         return inputString.toUpperCase();
6  };
7
8  var template = '<%= helperFunc(input.join(", "))
9                  ; %>';
10 var context = {
11   input: people,
12   helperFunc: transformUpper
13 };
14
15 console.log(ejs.render(template, context));
```

define your own function

function

array

define the context object

What is shown in the console?

```
1  var ejs = require('ejs');
2
3  var people = ['Wolverine', 'paul', 'picard'];
4  var X = function (input) { if(input){
5                                  return input[0];
6                              }
7                                  return "";
8  }
9
10 var template = '<%= helperFunc(input); %>';
11 var context = {
12   input: people,
13   helperFunc: X
14 };
15
16 console.log(ejs.render(template, context));
```

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = ['Wolverine', 'paul', 'picard'];
4  var first = function (input) { if(input){
5                                     return input[0];
6                                 }
7                                 return "";
8  }
9
10 var template = '<%= helperFunc(input); %>';
11 var context = {
12   input: people,
13   helperFunc: first
14 };
15
16 console.log(ejs.render(template, context));
```

npm install ejs

ejs & user-defined functions

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

```
1  var ejs = require('ejs');
2
3  var people = ['Wolverine', 'paul', 'picard'];
4  var first = function (input) { if(input){
5                                     return input[0];
6                                  }
7                                  return "";
8  }
9
10 var template = '<%= helperFunc(input); %>';
11 var context = {
12   input: people,
13   helperFunc: first
14 };
15
16 console.log(ejs.render(template, context));
```

define your own function

npm install ejs

ejs and JavaScript

```
1 var ejs = require('ejs');
2 var template = '<%
                    movies.forEach(function(movie){
                       console.log(movie.title+"/"+movie.release);
                    })
                    %>';
3 var context = {'movies': [
4           {title:'The Hobbit', release:2014},
5           {title:'X-Men', release:'2016'},
6           {title:'Superman V', release:2014}
7 ]};
8 ejs.render(template, context);
```

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

npm install ejs

ejs and JavaScript

use <% for control-flow purposes; no output

```
1 var ejs = require('ejs');
2 var template = '<%
                    movies.forEach(function(movie){
                        console.log(movie.title+"/"+movie.release);
                    })
                  %>';
3 var context = {'movies': [
4           {title:'The Hobbit', release:2014},
5           {title:'X-Men', release:'2016'},
6           {title:'Superman V', release:2014}
7 ]};
8 ejs.render(template, context);
```

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

npm install ejs

ejs and JavaScript

use <% for control-flow purposes; no output

Array.prototype.forEach()*

```
1 var ejs = require('ejs');
2 var template = '<%
                    movies.forEach(function(movie){
                        console.log(movie.title+"/"+movie.release);
                    })
                  %>';
3 var context = {'movies': [
4           {title:'The Hobbit', release:2014},
5           {title:'X-Men', release:'2016'},
6           {title:'Superman V', release:2014}
7 ]};
8 ejs.render(template, context);
```

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

npm install ejs

ejs and JavaScript

use <% for control-flow purposes; no output

```
1 var ejs = require('ejs');
2 var template = '<%
                    movies.forEach(function(movie){
                        console.log(movie.title+"/"+movie.release);
                    })
                    %>';
3 var context = {'movies': [
4           {title:'The Hobbit', release:2014},
5           {title:'X-Men', release:'2016'},
6           {title:'Superman V', release:2014}
7 ]};
8 ejs.render(template, context);
```

Array.prototype.forEach()*

applied to each element

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

npm install ejs

ejs and JavaScript

use <% for control-flow purposes; no output

```
1 var ejs = require('ejs');
2 var template = '<%
                    movies.forEach(function(movie){
                        console.log(movie.title+"/"+movie.release);
                    })
                    %>';
3 var context = {'movies': [
4           {title:'The Hobbit', release:2014},
5           {title:'X-Men', release:'2016'},
6           {title:'Superman V', release:2014}
7 ]};
8 ejs.render(template, context);
```

Array.prototype.forEach()*

applied to each element

```
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.

What is shown in the console?

```
1  var ejs = require('ejs');
2  var template =
           '<% if(user) {
3             console.log("Hi "+user);
4             if(user.age>=18){
5                console.log("(adult)");
6             }
7             else {console.log("(minor)");}
8          } %>';
9  var context = {user: 'Tom', age:26,
10                  address:'Mekelweg 4'};
11 ejs.render(template, context);
```

flickr@stratman2

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

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port, function () {
9    console.log("Ready on port " + port);
10 });
11
12 var todos = [];
13 todos.push({ message: 'Final exam', dueDate: 'January 2016'  });
14 todos.push({ message: 'Prepare for assignment 6', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up for final exam', dueDate: '06/01/2016' });
16
17 app.set('views', __dirname + '/views');
18 app.set('view engine', 'ejs');
19
20 app.get("/todos", function (req, res) {
21   res.render('todos', { title: 'My list of TODOs', todo_array: todos });
22 });
23
```

Example 7

Example: exposing data to views

templates.js

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port, function () {
9    console.log("Ready on port " + port);
10 });
11
12 var todos = [];
13 todos.push({ message: 'Final exam', dueDate: 'January 2016'  });
14 todos.push({ message: 'Prepare for assignment 6', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up for final exam', dueDate: '06/01/2016' });
16
17 app.set('views', __dirname + '/views');
18 app.set('view engine', 'ejs');
19
20 app.get("/todos", function (req, res) {
21   res.render('todos', { title: 'My list of TODOs', todo_array: todos });
22 });
23
```

Example 7

Exposing data to views

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = [];
11 todos.push({ message: 'Midterm exam tomorrow',
12              dueDate: '01/12/2015'  });
13 todos.push({ message: 'Prepare for assignment
14              5', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up for final exam',
16              dueDate: '06/01/2016' });
17
18
19 app.set('views', __dirname + '/views');
20 app.set('view engine', 'ejs');
21
22 app.get("/todos", function (req, res) {
23     res.render('todos', { title: 'My list of
24                 TODOs', todo_array: todos });
25 });
```

Example 7

Exposing data to views

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = [];
11 todos.push({ message: 'Midterm exam tomorrow',
12              dueDate: '01/12/2015'  });
13 todos.push({ message: 'Prepare for assignment
14              5', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up for final exam',
16              dueDate: '06/01/2016' });
17
18
19 app.set('views', __dirname + '/views');
20 app.set('view engine', 'ejs');
21
22 app.get("/todos", function (req, res) {
23     res.render('todos', { title: 'My list of
24                 TODOs', todo_array: todos });
25 });
```

the list of todos we want to serve to the clients

Example 7

Exposing data to views

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = [];
11 todos.push({ message: 'Midterm exam tomorrow',
12              dueDate: '01/12/2015'  });
13 todos.push({ message: 'Prepare for assignment
14              5', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up f
16              dueDate: '06/01/2016
17
18
19 app.set('views', __dirname + '/views');
20 app.set('view engine', 'ejs');
21
22 app.get("/todos", function (req, res) {
23     res.render('todos', { title: 'My list of
24                 TODOs', todo_array: todos });
25 });
```

the list of todos we want to serve to the clients

informing express about the view templates

Example 7

Exposing data to views

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = [];
11 todos.push({ message: 'Midterm exam tomorrow',
12               dueDate: '01/12/2015'  });
13 todos.push({ message: 'Prepare for assignment
14               5', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up f
16               dueDate: '06/01/2016
17
18
19 app                                me + '/views');
20 app                               ejs');
21
22 app.ge( '/todos', function (req, res) {
23     res.render('todos', { title: 'My list of
24                   TODOs', todo_array: todos });
25 });
```

the list of todos we want to serve to the clients

informing express about the view templates

`render()` indicates the use of a template

Example 7

Exposing data to views

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = [];
11 todos.push({ message: 'Midterm exam tomorrow',
12              dueDate: '01/12/2015'  });
13 todos.push({ message: 'Prepare for assignment
14              5', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up f
16              dueDate: '06/01/2016
17
18
19 app                                me + '/views');
20 app                                ejs');
21
22 app.ge( '/todos', function (req, res) {
23     res.render('todos', { title: 'My list of
24                  TODOs', todo_array: todos });
25 });
```

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

Example 7

Exposing data to views

```
1  var express = require("express");
2  var url = require("url");
3  var http = require("http");
4  var app;
5
6  var port = process.argv[2];
7  app = express();
8  http.createServer(app).listen(port);
9
10 var todos = [];
11 todos.push({ message: 'Midterm exam tomorrow',
12              dueDate: '01/12/2015'  });
13 todos.push({ message: 'Prepare for assignment
14              5', dueDate: '05/01/2016' });
15 todos.push({ message: 'Sign up f
16              dueDate: '06/01/2016
17
18
19 app                                me + '/views');
20 app                                ejs');
21
22 app.ge( '/todos', function (req, res) {
23     res.render('todos', { title: 'My list of
24                          Os', todo_array: todos });
25 })
```

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

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4   <title><%= title %></title>
5  </head>
6  <body>
7   <h1>TODOs</h1>
8   <div>
9     <% todo_array.forEach(function(todo) { %>
10     <div>
11       <h3><%= todo.dueDate %></h3>
12       <p><%= todo.message %></p>
13     </div>
14    <% }) %>
15  </div>
16 </body>
17 </html>
```

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

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

A last word on templates

- ejs retains the **original HTML tags**
- Other template languages do not, **Jade** is a popular example here - which one to choose depends on your personal preferences

Jade template

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
         bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

rendered HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
         bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
      </p>
    </div>
  </body>
</html>
```

Source: http://jade-lang.com/

End of Lecture