Intro to JavaScript

Jussi Pohjolainen

Agenda

- JavaScript Today
- JavaScript as a Language
- DOM and JQuery
- Building Frontend using AngularJS
- Building Backend using NodeJS
    - ExpressJS and MongoDB

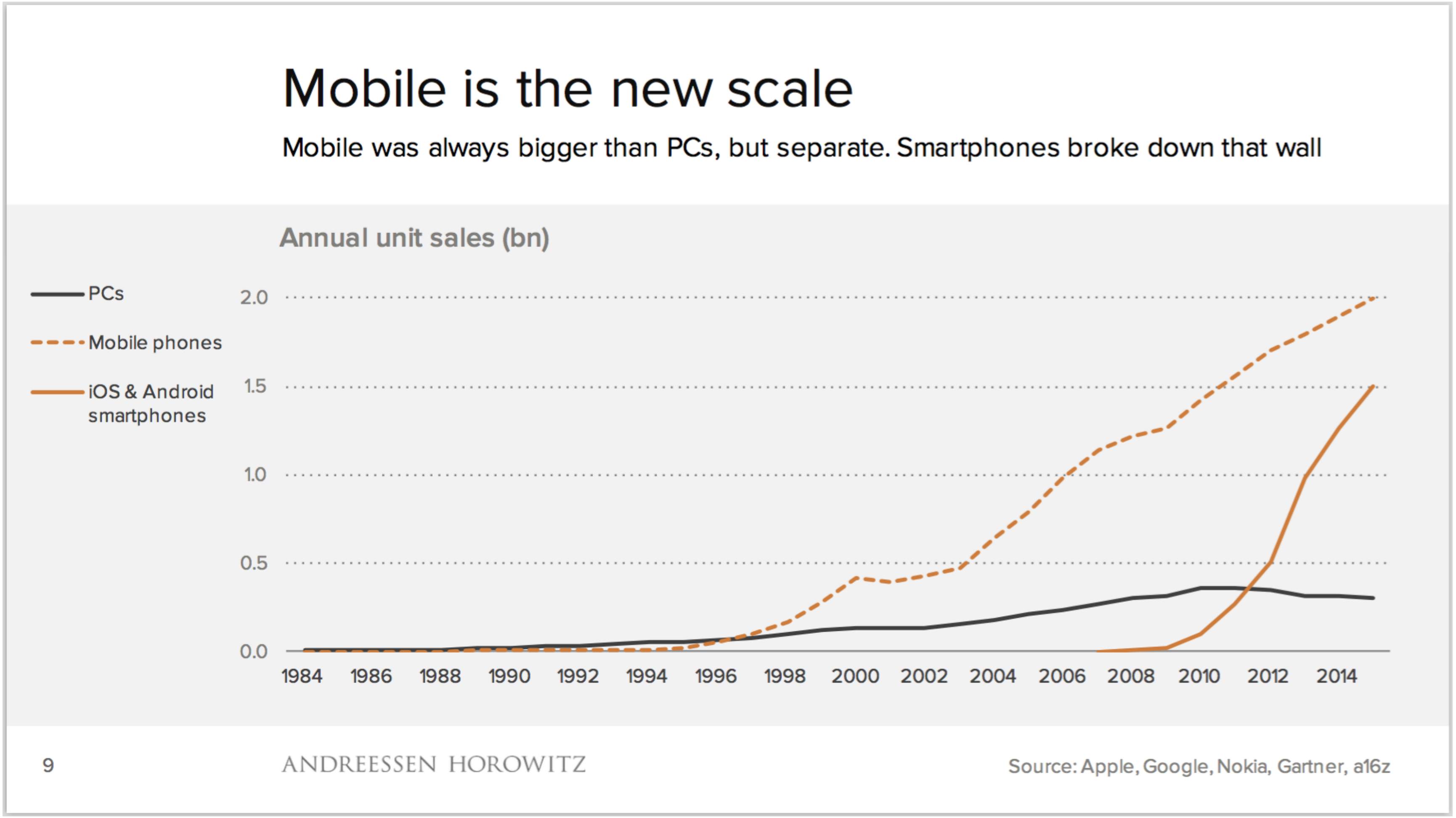
JavaScript History

- Developed in **ten days in May 1995 (Netscape)**
- LiveScript -> JavaScript; just for marketing reasons
- In 1996 Microsoft released it's port of JS called **JScript**
  - **Browser wars, IE vs Netscape**
  - **=> "Designed for IE/Netscape" web pages**
- In 1997 first standard of JavaScript: **EcmaScript** which today is the core of JS

JavaScript today

- The key language to implement **web applications**
- Building **universal Windows 10** applications
- Possible candidate for **cross-platform mobile development**
  - Cordova / Phonegap, Ionic
- Also **server side** development (Backend)
- Replacing / Complementing XML for **data transfer** (REST + JSON)

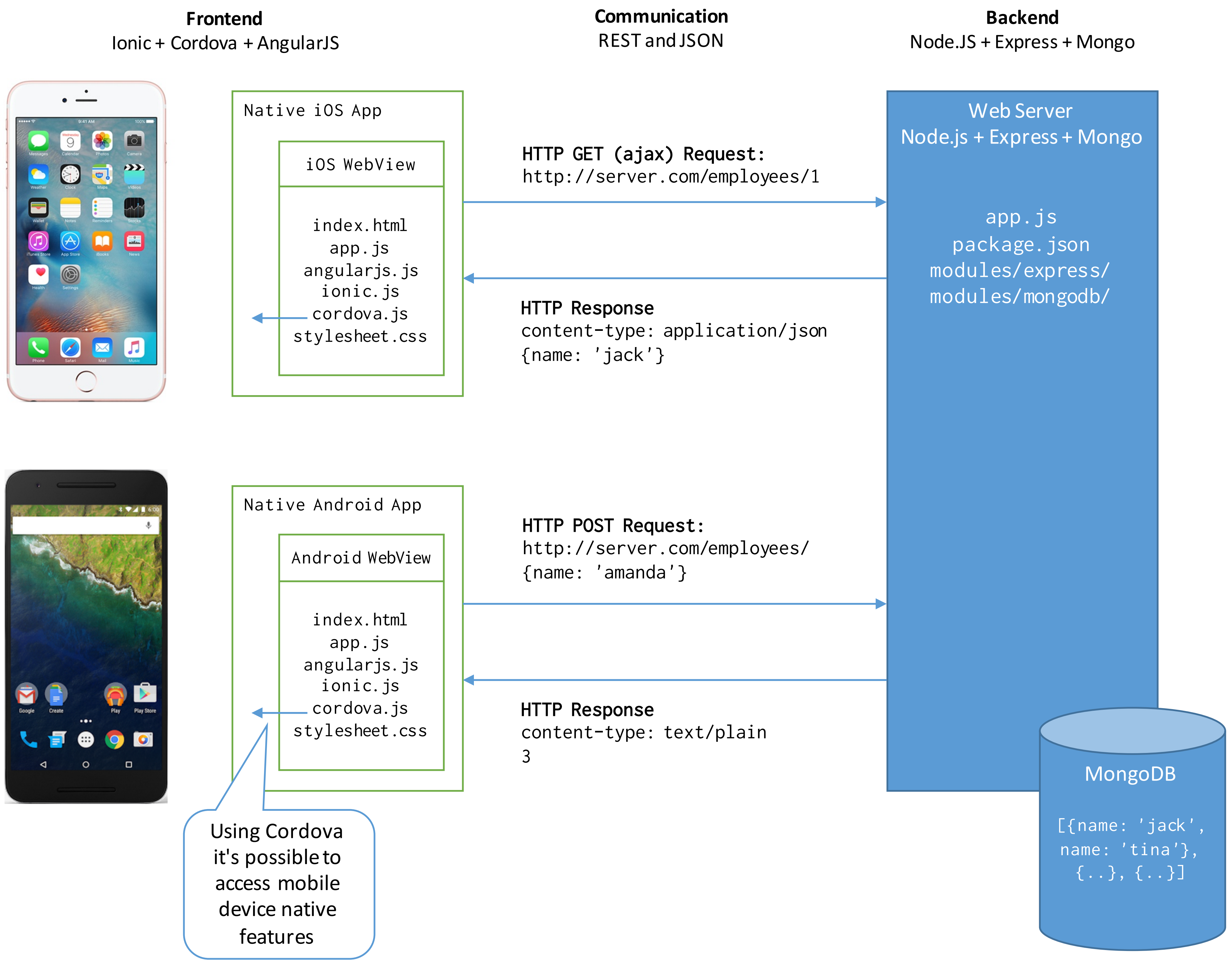
Mobile is the New Desktop


Mobile is the new scale

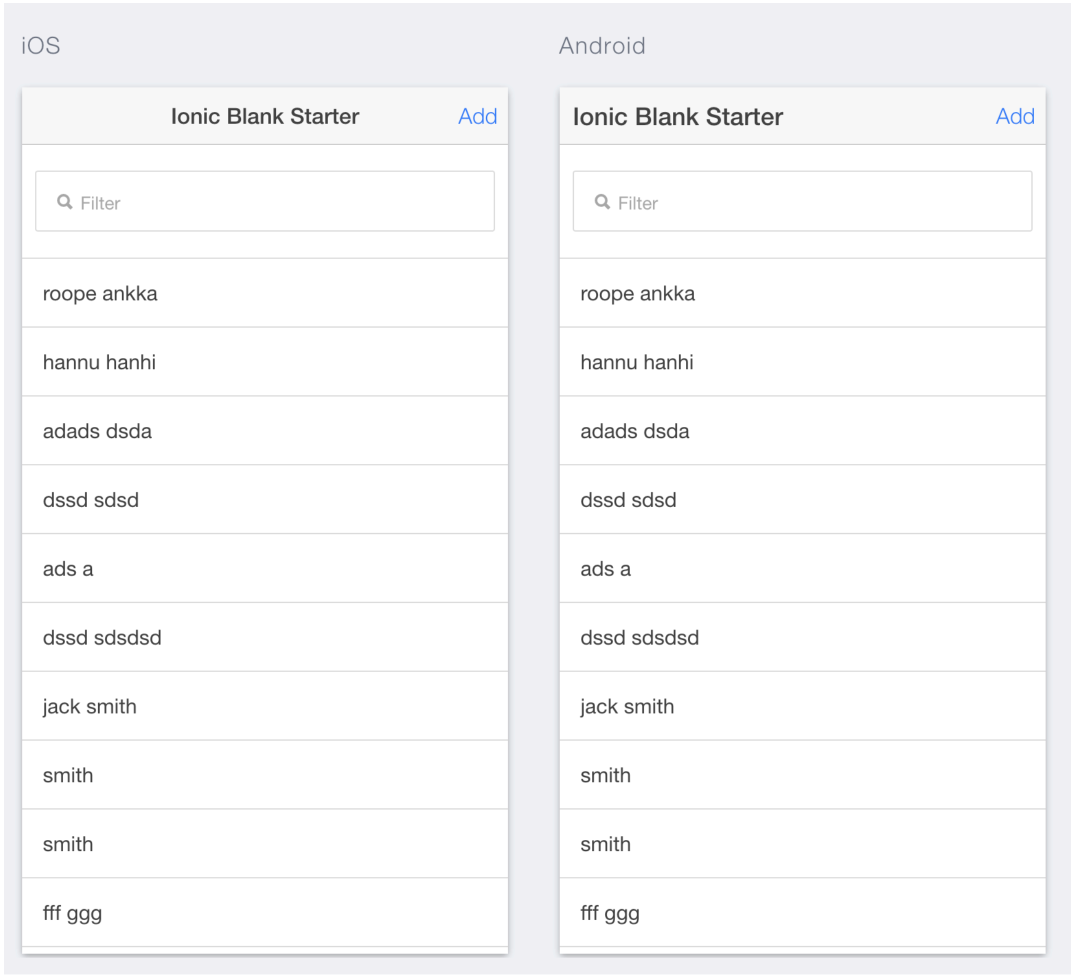
Mobile was always bigger than PCs, but separate. Smartphones broke down that wall

9 ANDREESSEN HOROWITZ Source: Apple, Google, Nokia, Gartner, a16z

http://ben-evans.com/benedictevans/2015/11/7/mobile-ecosystems-and-the-death-of-pcs


Frontend
Ionic + Cordova + AngularJS
Communication
REST and JSON
Backend
Node.JS + Express + Mongo
Native iOS App
iOS WebView
index.html
app.js
angularjs.js
ionic.js
cordova.js
stylesheet.css
HTTP GET (ajax) Request:
http://server.com/employees/1
HTTP Response
content-type: application/json
{name: 'jack'}
Web Server
Node.js + Express + Mongo
app.js
package.json
modules/express/
modules/mongodb/
Native Android App
Android WebView
index.html
app.js
angularjs.js
ionic.js
cordova.js
stylesheet.css
HTTP POST Request:
http://server.com/employees/
{name: 'amanda'}
HTTP Response
content-type: text/plain
3
MongoDB
[{name: 'jack', name: 'tina'}, {..}, {..}]
Using Cordova it's possible to access mobile device native features

Frontend



iOS

Ionic Blank Starter
Add

Filter

roope ankka
hannu hanhi
adads dsda
dssd sdsd
ads a
dssd sdsdsd
jack smith
smith
smith
fff ggg

Android

Ionic Blank Starter
Add

Filter

roope ankka
hannu hanhi
adads dsda
dssd sdsd
ads a
dssd sdsdsd
jack smith
smith
smith
fff ggg

JavaScript Language

JavaScript as Language

- High-level, multi-paradigm, dynamic, untyped and interpreted programming language
  - Very **easy to learn, hard to master**
- Because of the nature of JS, several frameworks available
  - SPA (angularjs), TDD (QUnit), Doc (JSDoc) ...
- Usually combined with other technologies such as HTML5 and CSS

Java vs JavaScript

- Despite the naming, **Java** and **JavaScript** are totally **different** programming languages!
  - Like "Car" and "Carpet"
- Back in the days
  - "JavaScript is essentially a toy, designed for writing small pieces of code, used by inexperienced programmers" – "Java is a real programming language for professionals"
- Today JavaScript is not overlooked!

Java vs JavaScript

- Java
  - **Object-oriented** language, with set of libraries and virtual machine platform
  - Apps are compiled to bytecode
  - Write once, run anywhere
- JavaScript
  - Multiparadigm language
  - Small API for text, arrays, dates, regex – no I/O, networking, storage, graphics...
  - Standardized by EcmaScript
  - Usually run in host environment that offers another API
  - Different environments (browsers) can be frustrating

Quick example of JS and HTML

```html
<!DOCTYPE html>
<html>
  <head>
    <title>
      Title
    </title>

    <meta charset="UTF-8" />
    <style media="screen"></style>

  </head>

  <body>
    <script>
      document.write("Hello World");
    </script>
  </body>

</html>
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>
      Title
    </title>

    <meta charset="UTF-8" />
    <style media="screen"></style>

  </head>

  <body>
    // External test.js containing "document.write(..)"
    <script src="test.js"></script>
  </body>

</html>
```

Development Tools

- Each browser holds a **JS Engine** with Debugging capabilities
  - **V8** (Chrome), **SpiderMonkey** (Firefox), **Nitro** (Safari), **Chakra** (Microsoft Edge)
- Possible to install only the JS Engine and use it via CLI
  - Node.js uses V8!
- IDEs
  - WebStorm, Eclipse + JSDT, Netbeans, Visual Studio, Visual Studio Code

Intro to programming with JS

References

- EcmaScript 5.1 Language Spesification
  - http://www.ecma-international.org/ecma-262/5.1/
- EcmaScript 6 Language Spesification (June 2015)
  - http://www.ecma-international.org/ecma-262/6.0/index.html
- Really good JavaScript Reference from Mozilla
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
- W3Schools have lot of JS stuff too but remember
  - http://meta.stackoverflow.com/questions/280478/why-not-w3schools-com

Primitive Types

- JavaScript is **loosely typed language!**
- Six primitive datatypes
  - `Boolean` (true, false)
  - `Null` (value that isn't anything)
  - `Undefined` (undefined value)
  - `Number` - floating point number (64 bit)
  - `String` (16 bit UNICODE)
  - `Symbol` (ECMAScript 6)
- And Objects (all the rest)

Primitive Types - Testing

```
var booleanValue    = true;
var stringValue     = "some text";
var undefinedValue;
var numberValue     = 4;
var nullValue       = null;

// boolean
console.log(typeof booleanValue)
// string
console.log(typeof stringValue)
// undefined
console.log(typeof undefinedValue)
// number
console.log(typeof numberValue)
// object (language design error in ECMAScript!)
console.log(typeof nullValue)
// false
console.log(null instanceof Object)
```

String

- Series of characters
- Indexes zero based, first char 0, second 1
- Examples
  - `var mj1 = ”hello”;`
  - `var mj2 = ”hello \”world\””;`
- Methods available
  - `charAt(), replace(), split()`
- See:
  - `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String`

Automatic type conversion

```
5 + null     // returns 5        because null is converted to 0
"5" + null   // returns "5null"  because null is converted to "null"
"5" + 1      // returns "51"     because 1 is converted to "1"
"5" - 1      // returns 4        because "5" is converted to 5
```

About Numbers

- `Number(value)`, converts entire string
  - `var i = Number(”12”);`
- `parseInt(value[, radix])`, converts **start of the string**
  - `var i = parseInt(”12px”, 10);`
  - Radix?
    - 10 => integer number, 8 => octal number, 16 => hexadecimal
- `NaN (Not a Number)`, check using `isNaN(variable)`
  - Result of erroneous operations

Date

```
var today = new Date();
var birthday = new Date('December 17, 1995 03:24:00');
var birthday = new Date('1995-12-17T03:24:00');
var birthday = new Date(1995, 11, 17);
var birthday = new Date(1995, 11, 17, 3, 24, 0);

var unixTimestamp = Date.now(); // in milliseconds
```

Intl - object

- Intl object is a **namespace** for internalization API
  - NumberFormat, DateTimeFormat
- Example

```
var today = new Date();

var options = { year: "2-digit",
               month: "2-digit",
                 day: "2-digit",
                hour: "2-digit",
              minute: "2-digit",
        timeZoneName: "short" };

var americanDateTime = new Intl.DateTimeFormat("fi-FI",
options).format;

console.log(americanDateTime(today)); // 15-11-05 12:42 GMT+2
```

Regex

- Regular expressions are patterns used to match character combinations in strings
- In JS regex is an object
  - `var re = new RegExp("ab+c");`
- For better performance
  - `var re = /ab+c/;`
- Example

```
var regex = new RegExp("java");

if("javascript".match(regex)) {
    console.log("Match found!");
}
```

Indexed Collection: Arrays

- Arrays are objects but in addition they have more methods and auto-increment key-values
  - `var stuff = [”a”, ”b”, ”c”]`
- The keys are now 0, 1 and 2
- The length of the array is 3 and **it's linked to numerical properties**

Array Usage

```
var stuff = ["a", "b", "c"]

console.log(stuff[0]);      // a
console.log(stuff[1]);      // b
console.log(stuff[2]);      // c
console.log(stuff.length);  // 3

// Array's length and numerical properties are connected
stuff.push("d")
console.log(stuff.length);   // 4

stuff["key"] = "value";
console.log(stuff);          // [ 'a', 'b', 'c', 'd', key: 'value' ]
console.log(stuff.length);   // 4!

delete stuff["key"];
console.log(stuff);          // [ 'a', 'b', 'c', 'd' ]

stuff[4] = "e";
console.log(stuff);          // [ 'a', 'b', 'c', 'd', 'e' ]
console.log(stuff.length);   // 5


stuff = ["a", "b", "c"];
stuff[9] = "e";
console.log(stuff);          // [ 'a', 'b', 'c', , , , , , , 'e' ]
console.log(stuff.length);   // 10
```

JavaScript Objects

- Object is an collection of key – value pairs (properties
  - `var car = { brand: ”Ford”, year: 2015 }`
- Possible to create properties dynamic
  - `var car = new Object(); car.brand = ”Ford”;`
- Possible to use also bracket notation
  - `car[”year”] = 2015`
- Delete key-value pair
  - `delete car[”year”]`

Keyed Collections: Map

- EcmaScript 6: Map object for simple key-value pairs

```
var sayings = new Map();
sayings.set("dog", "woof");
sayings.set("cat", "meow");

for (var [key, value] of sayings) {
  console.log(key + " goes " + value);
}
```

- Map object advantages over Objects:
  - Map keys can be anything, in objects, it's string
  - Map has size information
  - Iteration in maps are the insert order

Keyed Collections: Set

- Set object is a collection of values

```
var mySet = new Set();
mySet.add(1);
mySet.add("some text");
mySet.add("foo");

mySet.has(1); // true
mySet.delete("foo");
mySet.size; // 2

for (var item of mySet)
    console.log(item);
```

- Set object advantages over arrays:
  - Only unique values
  - Checking element if exists is faster
  - Can delete values, in array's its splice

Expressions and Operators

- Assignment
  - `var x = 5; x++; x += 10;`
- Comparison
  - `==, ===, <, >, <=, => ...`
- Arithmetic
  - `+, /, -, * ...`
- Logical
  - `&&, ||, !`

Input & Output – Depends!

- **In Browsers**
  - Input: HTML Forms or `window.prompt("", "");`
  - Output: DOM manipulation, `document.write("")` or `window.alert("");`
- **In V8/NodeJS or Rhino/Spidermonkey**
  - Output to CLI: `print("..");`
  - Input from CLI: Is not that easy...
- **Debugging**
  - `Console.log("");`

Demo: Browser Environment

```html
<!DOCTYPE html>
<html>
  <head>
    <title>
      Embed Example
    </title>

    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script></script>

  </head>

  <body>
    <input type="button" onclick="window.alert('Hello World!');"
value="Show Alert Box" />
  </body>

</html>
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>
      Embed Example
    </title>

    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script>

    function showAlert()
    {
        window.alert("Hello World!");
    }

    </script>

  </head>

  <body>
    <input type="button" onclick="showAlert();" value="Show Alert Box" />
  </body>

</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Embed Example
    </title>

    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script>

    function askQuestion()
    {
        var name = window.prompt("Please enter your name","Harry Potter");

        // If name was empty or cancel was pressed
        if (!(name == "" || name == null))
        {
            window.alert("Hello " + name + "! How are you today?");
        }
    }

    </script>

  </head>

  <body>
    <input type="button" onclick="askQuestion();" value="Question for Me" />
  </body>

</html>
```

Control structures

Business as Usual...

Statements: Should be Trivial

- `if`
- `switch/case`
- `while`
- `do/while`
- `for`
- `break`
- `continue`
- `return`
- `try/throw/catch`

true or false?

```
var myArray1 = [false, null, undefined, "", 0,
NaN];

// EcmaScript 5 feature, iterate the array
myArray1.forEach(function(entry)
{
    if(entry)
    {
        print(entry);    // Here?
    }
});
```

true or false?

```
var myArray1 = ["false", "0", "undefined", "NaN"];

myArray1.forEach(function(entry)
{
    if(entry)
    {
        print(entry);
    }
});
```

true or false?

```
var value = 0;
if(value = 0)
{
    print("A");
}

if(value == 0)
{
    print("B");
}

if("0" == 0)
{
    print("C");
}

if("0" === 0)
{
    print("D");
}
```

About for in

- For in iterates enumerable properties **of an object in arbitrary order!**
- Example
    - `for(variable in object) {  … }`
- You can iterate also array, but remember that order is not guaranteed

JS Functions

Functions

- Functions are fundamental part of JS coding
- Functions are objects! It's possible to pass them as variables
- If function is object's property, it's a method
- Functions uses **variable hoisting**

Output?

```
var x = 10;

function test() {
    document.write(x);
    if(true) {
        var x = 5;
    }
    document.write(x);
}
```

Hoisting

- Variables are moved on top of the function!

```
function test() {
    var x;
    if(true) {
        x = 5;
    }
    document.write(x);
}
```

Single Var Pattern

```
function test() {
    var a = 1,
        b = 2,
            …;

    // rest of the function
}

test();
```

FD, FE, NFE

- It gets harder when a variable is
  - Function declaration (FD)
  - Function expression (FE)
- Rules
  - **Function declaration is not hoisted**
  - **Function expression is hoisted**

Function Declaration: this works!

```
function testC()
{
    print(foo());
    function foo()
    {
        return 5;
    }
}
<=>
function testD()
{
    function foo()
    {
        return 5;
    }
    print(foo());
}
```

Function Declaration: this works!

```
function testA()
{
    print(foo());
    var foo = function()
    {
        return 5;
    }
}
<=>
function testB()
{
    var foo;
    print(foo());
    foo = function()
    {
        return 5;
    }
}
```

Functions

- Every function in JS is Function object
  - Can be passed as arguments
  - Can store name / value pairs
  - Can be anonymous or named
- Usage (Don’t use this, it’s not efficient)
  - `var myfunction = new Function(”a”,”b”, ”return a+b;”);`
  - `print(myfunction(3,3));`
- Only functions have scope, regular {blocks) don’t
- Inner function can have access to outer function’s properties and parameters

Returning a Inner Function

```
function makeFunc() {
    var name = "Hello World";
    function displayName() {
        print(name);
    }

   return displayName;
}

// Is “Hello World” printed?
var myFunc = makeFunc();
myFunc();
```

OO in JS

About Objects

- Everything (except basic types) are objects
  - Including functions and arrays
- Object contains properties and methods
  - Collection of name-value pairs
  - Names are strings, values can be anything
  - Properties and methods can be added at runtime
- Objects can inherit other objects

Object Literal

```
var mystring = "hello!";
var myarray = ["element1", "element2"];
var circle1 = {radius: 9, getArea : someFunction };
var circle2 = {
   radius: 9,
   getRadius: function() {
        return this.radius;
   }
}
```

No Classes!

- One of the simplest way to create object
  - `var obj = new Object();`
  - `obj.x = 10;`
  - `obj.y = 12;`
  - `obj.method = function() { … }`
- This adds dynamically two properties to the obj – object!
- Object is built – in data type

“Class”

- To define a class, define a function

```
function Foo() {
    this.x = 1;
    this.y = 1;
}
```

- `var obj = new Foo();`
- **Internally a Object is created**

Example

```
function Circle(radius)
{
        this.radius = radius;
        this.getArea = function()
        {
            return (this.radius * this.radius) * Math.PI;
        };
}

var myobj = new Circle(5);
document.write(myobj.getArea());
```

About Namespaces

- Avoid polluting global scope
  - Use namespaces!
  - Helps avoid clashes between your code and third-party libraries
- Namespaces **don’t have dedicated syntax built into the language**
- It’s possible to **get same benefits** by creating **single global object** and add all other objects and functions to this object

EcmaScript 5

EcmaScript

- Ecma Standard is based on JavaScript (Netscape) and JScript (Microsoft)
- Development of the standard started in 1996
- First edition 1997
- Support
  - http://kangax.github.com/es5-compat-table/
  - http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Object Types

- Native (Core Javascript)
  - ECMAScript standard: Array, Date..
- Host
  - The host environment, for example browser: window, DOM objects

EcmaScript

- Goal
  - Fix “bad” parts of JS while maintaining compatible with EcmaScript 5
- Introduces **Strict mode**
  - *Removes features from the language!* Raises errors that **were okay in non strict mode**
  - Backward compatible
  - Add “use strict”, in function or global scope
- EcmaScript supports non-strict mode, **but it’s depricated!**

Strict “mode”

- Detection of bad/dangerous programming practices
  - with() statement prevented
  - Assignment to non-declared variables prevented (i = 3)
  - Eval is prohibited
  - Lot of other issues..
- See ES5 specification page 235

Enable strict mode

```
> cat strictmode.js
// This is just a string, backward compatible!
"use strict";

i = 0;

> rhino strictmode.js
js: uncaught JavaScript runtime exception:
ReferenceError: Assignment to undefined "i" in
strict mode
```

Global and local

```
// GLOBAL, everything is strict:
"use strict";

// LOCAL, function is strict
function foo()
{
   "use strict";
   //.. strict function
}
```

Code Quality: JSLint

JSLint

- JSLint is JS code quality tool made by Douglas Crockford
    - http://jslint.com
- Inspects and warns about potential problems
- **“Will hurt your feelings”**
- Excepts that your code is in “strict mode”
- Can be used via website or command line (installation required)
- Command line tool (Java wrapper for JSLint)
    - http://code.google.com/p/jslint4java/

Any problems in the code?

```
function sum (a, b)
{
    s = a + b;
    return s;
}

x = sum(5,5);

print (x);
```

JSLint

```
function sum (a, b)
{
    ”use strict”;
    var s = a + b;
    return s;
}

var x = sum(5,5);

print (x);
```

BOM vs DOM?

JavaScript

- JavaScript = ECMAScript + Host Objects
- In Browsers, host objects are
  - **BOM** (Browser object model)
  - **DOM** (Document object model)

Objects

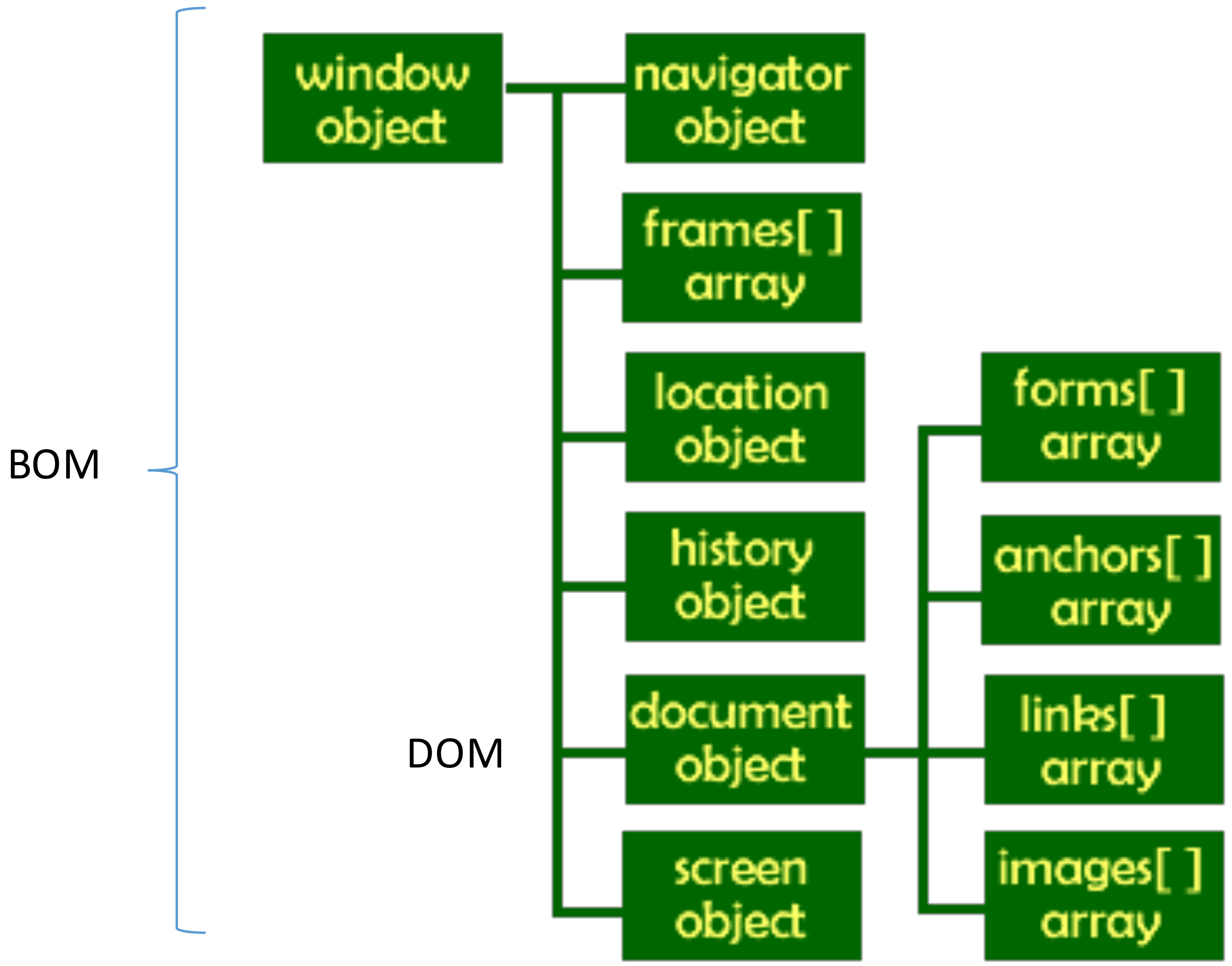


Image: http://www.daaq.net/old/javascript/index.php?page=the+js+bom

DOM

W3C DOM

- DOM – Document Object Model – cross-platform and language-independent convention for interacting with objects in HTML and in XML.
- With DOM you can manipulate html/xml document! Dynamic html!
- Public interface available: `http://www.w3.org/DOM/DOMTR`

W3C DOM Levels

- **(DOM Level 0 and Intermediate DOM)**
  - Not W3C Standards, used in Netscape 2 (Level 0) and Netscape 4 (Intermediate DOM)
- **DOM Level 1**
  - 1998: Ability to change entire HTML or XML document
- **DOM Level 2**
  - 2001: Introduces “getElementById” function, **event model** and support for XML namespaces
- **DOM Level 3**
  - 2004: XPath, keyboard event handling
- **Support varies!**
  - `http://www.webbrowsercompatibility.com/dom/desktop/`

Node

- In DOM, each object is Node
- In this
  - `<p>Hello</p>`
- You have two nodes 1) element node p 2) text node Hello
- Text node is *child node* of p element. P element is *parent node of* the text node

Node Example

```
<p>This is a <strong>paragraph</strong></p>
```

```
          <p>
           |
     ---------------
    |              |
This is a       <strong>
                   |
                   |
                paragraph
```

Attribute Node

```
<a href="http://www.company.fi">Company</a>
```

```
          <a> ------------------
           |                 |
         TAMK               href
                             |
                     http://www.company.fi
```

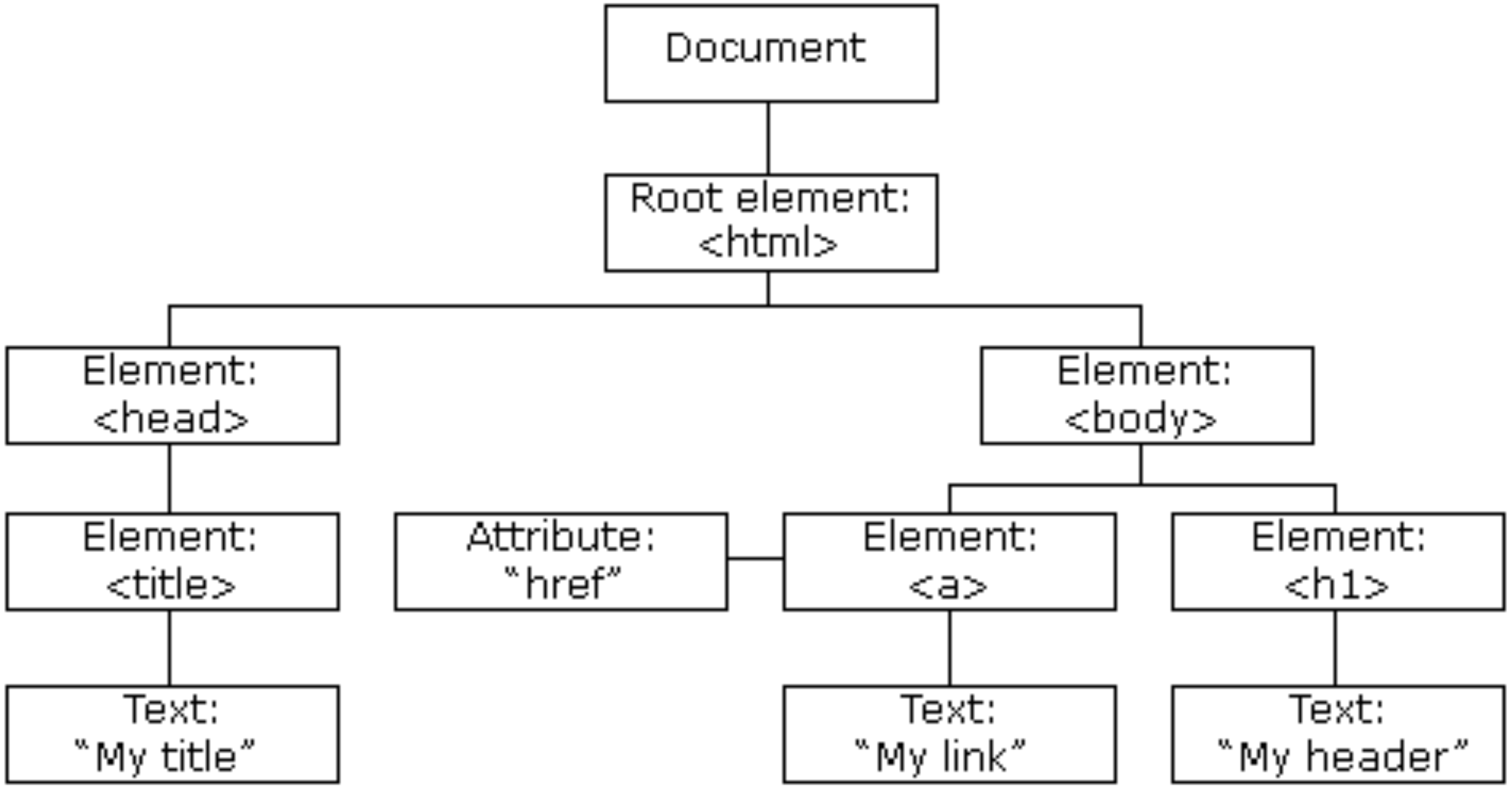
Nodes

- Element node: p, img, a
- Text node: *sometext*
- Attribute node: *src*

DOM Level 1: To Access DOM tree

- X can be some node
  - `var parent = x.parentNode;`
  - `var firstchild = x.childNodes[0];`
- How to get reference to x?

Document object



Access

```
var title = document.firstChild.firstChild.lastChild;
// document.html.head.title

var title = document.firstChild.childNodes[0].childNodes[0];
```

Getting Element Easier Way

```
var x = document.getElementsByTagName('strong')[0]

var x = document.getElementById('hereweare');
```

Changing the Node

```
// <a href="" id="someId">Link</p>
var x = document.getElementById('someId');
x.firstChild.data = "Hello!";
x.setAttribute("href", "http:/…");
```

Inner HTML

```
// <a href="" id="someId">Link</p>
var x = document.getElementById("someId");
x.innerHTML = "Hello!";
```

Creating and Removing Nodes

```
var x = document.createElement('hr');
document.getElementById('inserthrhere').appendChild(x);

var node = document.getElementById('inserthrhere')
node.removeChild(node.childNodes[0]);

var x = document.createTextNode('SomeText');
document.getElementById('hereweare').appendChild(x);
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <meta http-equiv="content-type" content="application/xhtml+xml; charset=utf-8" />
     <script type="text/javascript">
    //<![CDATA[
    function change()
    {
        // Get list of ALL <h1> - elements
        var listOfHeading1 = window.document.getElementsByTagName("h1");
        // Find the first <h1> - element in the list
        var heading1       = listOfHeading1[0];
        // Get the child - element of the first <h1> - element (Text)
        var text           = heading1.firstChild;
        // Replace the text
        text.data = "Hello from JS!";
    }
    //]]>
</script>
</head>
<body>

<h1>Title</h1>
<input type="button" onClick="change();" value="click!"/>
</body>
</html>
```

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <meta http-equiv="content-type" content="application/xhtml+xml; charset=utf-8" />
     <script type="text/javascript">
    //<![CDATA[
    function change()
    {
        // Reference to the table - element
        var table = document.getElementById("mytable");
        var tr      = document.createElement("tr");         // <tr>
        var td1     = document.createElement("td");         // <td>
        var td1Text = document.createTextNode("New Cell"); // "New Cell"
        td1.appendChild(td1Text);                           // <td>New Cell</td>

        var td2     = document.createElement("td");         // <td>
        var td2Text = document.createTextNode("New Cell"); // "New Cell"
        td2.appendChild(td2Text);                           // <td>New Cell</td>

        tr.appendChild(td1);
        tr.appendChild(td2);
        table.appendChild(tr);
    }

    //]]>
</script>

</head>
<body>

<table id="mytable" border="1">
<tr><td>&nbsp;</td><td>&nbsp;</td></tr>
<tr><td>&nbsp;</td><td>&nbsp;</td></tr>
<tr><td>&nbsp;</td><td>&nbsp;</td></tr>
</table>

<input type="submit" onClick="change();" value="Add Row"/>

</body>
</html>
```

JQuery – DOM is Easy!

Motivation

- **Motivation**
  - Simple things may require lot of coding
  - Common browsers are different and implementation varies
- **Solution, use a framework**
  - **jQuery** is a fast and concise JavaScript Library that simplifies **HTML document traversing**, event handling, animating, and Ajax interactions for rapid web development.
- jQuery is the most used JS library

jQuery

- jQuery at it's core is a DOM manipulation library
- Simplifies the syntax for finding, selecting and manipulating these DOM elements
- Features
  - DOM selection
  - DOM manipulation
  - Events, effects and animation
  - AJAX

How?

- Download jQuery file
  - `http://jquery.com/download/`
- Make your HTML5 page and reference to the file in script block
- Make your code and use JQuery functions!

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

 // When document is ready to be manipulated
 jQuery(document).ready( pageReadyToBeManipulated );

 function pageReadyToBeManipulated() {
     // If link is clicked
     jQuery("a").click( linkClick );
 }

 function linkClick(event) {
     alert("Thanks for visiting!");
     // Prevent the default action
     event.preventDefault();
 }

</script>
```

Some Basic Syntax

- JQuery can be used in two ways:
  - `JQuery()` or `$()`
- $ is an alias to JQuery()! $ more commonly used

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
//<![CDATA[

 // When document is ready to be manipulated
 $(document).ready( pageReadyToBeManipulated );

 function pageReadyToBeManipulated() {
     // If link is clicked
     $("a").click( linkClick );
 }

 function linkClick(event) {
     alert("Thanks for visiting!");
     // Prevent the default action
     event.preventDefault();
 }

 //]]>
</script>
```

```
// USING ANONYMOUS FUNCTIONS and document - obj is not
// needed..
 <script type="text/javascript" src="jquery.js"></script>

 <script type="text/javascript">
 //<![CDATA[

  $().ready(function(){
      $("a").click(function(event){
          alert("Thanks for visiting!");
          event.preventDefault();
      });
  });

  //]]>
 </script>
```

Getters in the Traditional Way

- `getElementsById`
- `getElementsByTagName`
- `getAttribute`

JQuery and Selectors

- Select all h1 elements
  - `$("h1")`
- Select the first one
  - `$("h1")[0]`
- Add contents
  - `$("h1")[0].html = "hello!";`
- Lot of different selectors
  - `http://api.jquery.com/category/selectors/`

Creating Elements in Traditional Way

- `createElement`
- `createTextNode`
- `setAttribute`
- `appendChild`
- `removeChild`

JQuery Insert

```
$().ready(function(){
        $("a").click(function(event){

            // Insert the new element after element with id here
            $("<p>New Element</p>").insertAfter("#here");

            event.preventDefault();
        });
    });
```

Manipulation Functions

- `.addClass()`
- `.after()`
- `.append()`
- `.css()`
- …
- See: `http://api.jquery.com/category/manipulation/`

Ajax

```
<script src="jquery-2.1.4.min.js"></script>

  <script>

  $().ready(function(){

    $("button").click(function(){
        var ajaxRequestObject = {url: "some-file.txt",
                                 success: function(result) {
                                        $("#div1").html(result);
                                     }
                                };

        $.ajax(ajaxRequestObject);
    });
  });
  </script>
</head>

<body>
<button text="click">Fetch Content</button>

<div id="div1"></div>
```

Frontend using AngularJS

Angular JS

- **Single Page App Framework** for JavaScript
- Implements client-side **Model-View-Whatever** pattern
  - Some call it MVC, some MVVM, it does not matter:
  - **Separation** of presentation from **business logic** and **presentation state**
- **No direct DOM** manipulation, less code
- Support for all major browsers
- Supported by Google
- Large and fast growing community

First Example – Template

Template

```
<!DOCTYPE html>
<html ng-app>
  <head>
    <title>
      Title
    </title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="angular.min.js"></script>
  </head>
  <body>
    <!-- initialize the app -->
    <div>
      <!-- store the value of input field into a variable name -->
      <p>Name: <input type="text" ng-model="name"></p>
      <!-- display the variable name inside (innerHTML) of p -->
      <p ng-bind="name"></p>
    </div>
  </body>
</html>
```

Directive

Download this file from:
https://angularjs.org/

Directive

Basic Concepts

- **1) Templates**
  - HTML with additional markup, directives, expressions, filters ...
- **2) Directives**
  - Extend HTML using `ng-app`, `ng-bind`, `ng-model`
- **3) Filters**
  - Filter the output: `filter`, `orderBy`, `uppercase`
- **4) Data Binding**
  - Bind model to view using expressions `{{ }}`

2) Directives

- **Directives** apply special behavior to attributes or elements in HTML
  - Attach behaviour, transform the DOM
- Some directives
  - `ng-app`
    - Initializes the app
  - `ng-model`
    - Stores/updates the value of the input field into a variable
  - `ng-bind`
    - Replace the text content of the specified HTML with the value of given expression

About Naming

- AngularJS HTML Compiler supports multiple formats
  - `ng-bind`
    - Recommended Format
  - `data-ng-bind`
    - Recommended Format to support HTML validation
  - `ng_bind, ng:bind, x-ng-bind`
    - Legacy, don't use

Lot of Built in Directives

- `ngApp`
- `ngClick`
- `ngController`
- `ngModel`
- `ngRepeat`
- `ngSubmit`
- `ngDblClick`
- `ngMouseEnter`
- `ngMouseMove`
- `ngMouseLeave`
- `ngKeyDown`
- `ngForm`

2) Expressions

- Angular **expressions** are JavaScript-like code snippets that are usually placed in bindings
  - `{{ expression }}.`
- Valid Expressions
  - `{{ 1 + 2 }}`
  - `{{ a + b }}`
  - `{{ items[index] }}`
- Control flow (loops, if) are not supported!
- You can use **filters** to format or filter data

Example

```
<!DOCTYPE html>
<html ng-app>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
  </head>
  <body>
    <div>
      <p>Number 1: <input type="number" ng-model="number1"></p>
      <p>Number 2: <input type="number" ng-model="number2"></p>
      <!-- expression -->
      <p>{{ number1 + number2 }}</p>
    </div>
  </body>
</html>
```

Directive

Directive

Expression

Valid HTML5 version

```
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
  </head>
  <body>
    <div>
      <p>Number 1: <input type="number" data-ng-model="number1"></p>
      <p>Number 2: <input type="number" data-ng-model="number2"></p>
      <!-- expression -->
      <p>{{ number1 + number2 }}</p>
    </div>
  </body>
</html>
```

Add data-

Add data-

ng-init and **ng-repeat** directives

```
<html data-ng-app="">
<head>
  <title>Title</title>
  <meta charset="UTF-8" />
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-init="names = ['Jack', 'John', 'Tina']">
    <h1>Cool loop!</h1>
    <ul>
      <li data-ng-repeat="name in names">{{ name }}</li>
    </ul>
  </div>
</body>
</html>
```

3) Filter

- With **filter,** you can **format or filter** the output
- **Formatting**
  - `currency, number, date, lowercase, uppercase`
- **Filtering**
  - `filter, limitTo`
- **Other**
  - `orderBy, json`

Using Filters - Example

```
<!DOCTYPE html>
<html data-ng-app="">
<head>

  <title>Title</title>
  <meta charset="UTF-8">
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>

<body>
  <div data-ng-init="customers = [{name:'jack'}, {name:'tina'}]">
    <h1>Cool loop!</h1>
    <ul>
      <li data-ng-repeat="customer in customers | orderBy:'name'">
          {{ customer.name | uppercase }}</li>
    </ul>
  </div>
</body>
</html>
```

Using Filters - Example

```
<!DOCTYPE html>

<html data-ng-app="">
<head>
  <title>Title</title>
  <meta charset="UTF-8">
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>
<body>
  <div data-ng-init=
  "customers = [{name:'jack'}, {name:'tina'}, {name:'john'}, {name:'donald'}]">
    <h1>Customers</h1>

    <ul>
      <li data-ng-repeat="customer in customers | orderBy:'name' | filter:'john'">{{
      customer.name | uppercase }}</li>
    </ul>
  </div>
</body>
</html>
```

Using Filters – User Input Filters the Data

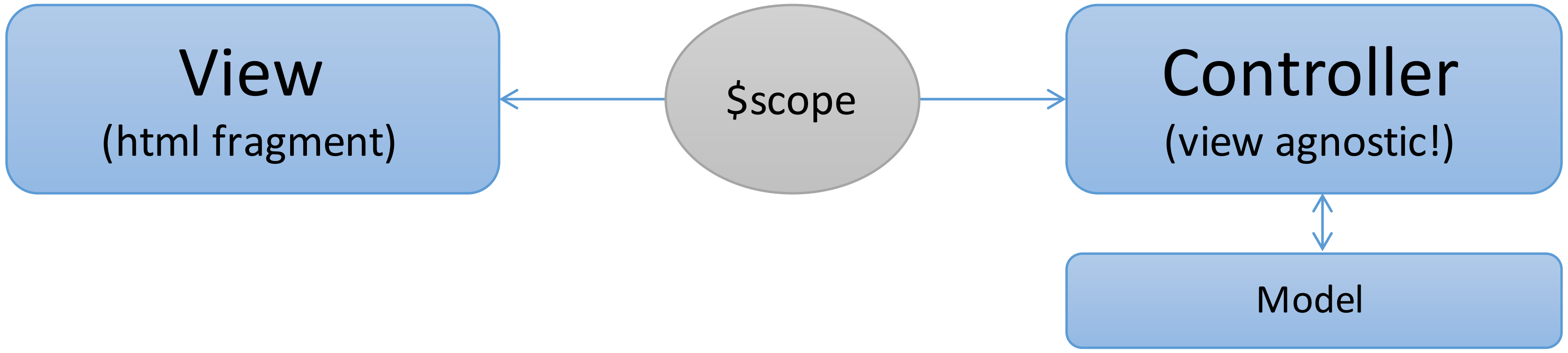
```
<!DOCTYPE html>

<html data-ng-app="">
<head>
  <title>Title</title>
  <meta charset="UTF-8">
  <script src="../angular.min.js" type="text/javascript">
</script>
</head>
<body>
  <div data-ng-init=
  "customers = [{name:'jack'}, {name:'tina'}, {name:'john'}, {name:'donald'}]">
    <h1>Customers</h1>

    <input type="text" data-ng-model="userInput" />
    <ul>
      <li data-ng-repeat="customer in customers | orderBy:'name' | filter:userInput">{{
      customer.name | uppercase }}</li>
    </ul>
  </div>
</body>
</html>
```

Views, Controllers, scope

Model – View - **Controllers**

- **Controllers** provide the **logic** behind your app.
    - So use controller when you need logic behind your UI
- Use **ng-controller** to define the controller
- Controller **is a JavaScript Object, created by** standard **JS object constructor**

View, Controller and Scope



$scope is an object that can *be used to communicate* between View and Controller

controller.js

```
// Angular will inject the $scope object, you don't have to
// worry about it! By using $scope, you can send data to
// view (html fragment)
function NumberCtrl ($scope) {

   // $scope is bound to view, so communication
   // to view is done using the $scope
   $scope.number = 1;

   $scope.showNumber = function showNumber() {
     window.alert( "your number = " + $scope.number );
  };
}
```

Warning, this will not work from AngularJS 1.3.
We will see later on how this is done using module

```html
<!DOCTYPE html>
<html data-ng-app="">
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
    <script src="controller.js"></script>

  </head>
  <body>
    <div>
      <div data-ng-controller="NumberCtrl">
        <p>Number: <input type="number" ng-model="number"></p>
        <p>Number = {{ number }}</p>
        <button ng-click="showNumber()">Show Number</button>
      </div>
    </div>
  </body>
</html>
```

Define the Controller implemented in controller.js

Access $scope.showNumber()

Access $scope.number

Modules

Modules

- **Module** is an reusable container for different features of your app
    - **Controllers**, services, filters, directives...
- All **app controllers** should **belong to a module**!
    - More **readability**, global **namespace clean**
- Modules can be loaded in any order
- We can build our **own filters** and **directives**!

`angular.module`

- The `angular.module` is a global place for creating, registering and retrieving Angular modules
- Creating a new module
  - `var myModule = angular.module('myMod', []);`
- The second argument ([]) defines dependent modules – *which modules should be loaded first before this*

Template for Controllers

```
// Create new module 'myApp' using angular.module method.
// The module is not dependent on any other module
var myModule = angular.module('myModule',
                              []);

myModule.controller('MyCtrl', function ($scope) {
    // Your controller code here!
});
```

Creating a Controller in Module

```
var myModule = angular.module('myModule',
                              []);

myModule.controller('MyCtrl', function ($scope) {

    var model = { "firstname": "Jack",
                  "lastname": "Smith" };

    $scope.model = model;
    $scope.click = function() {
        alert($scope.model.firstname);
    };
});
```

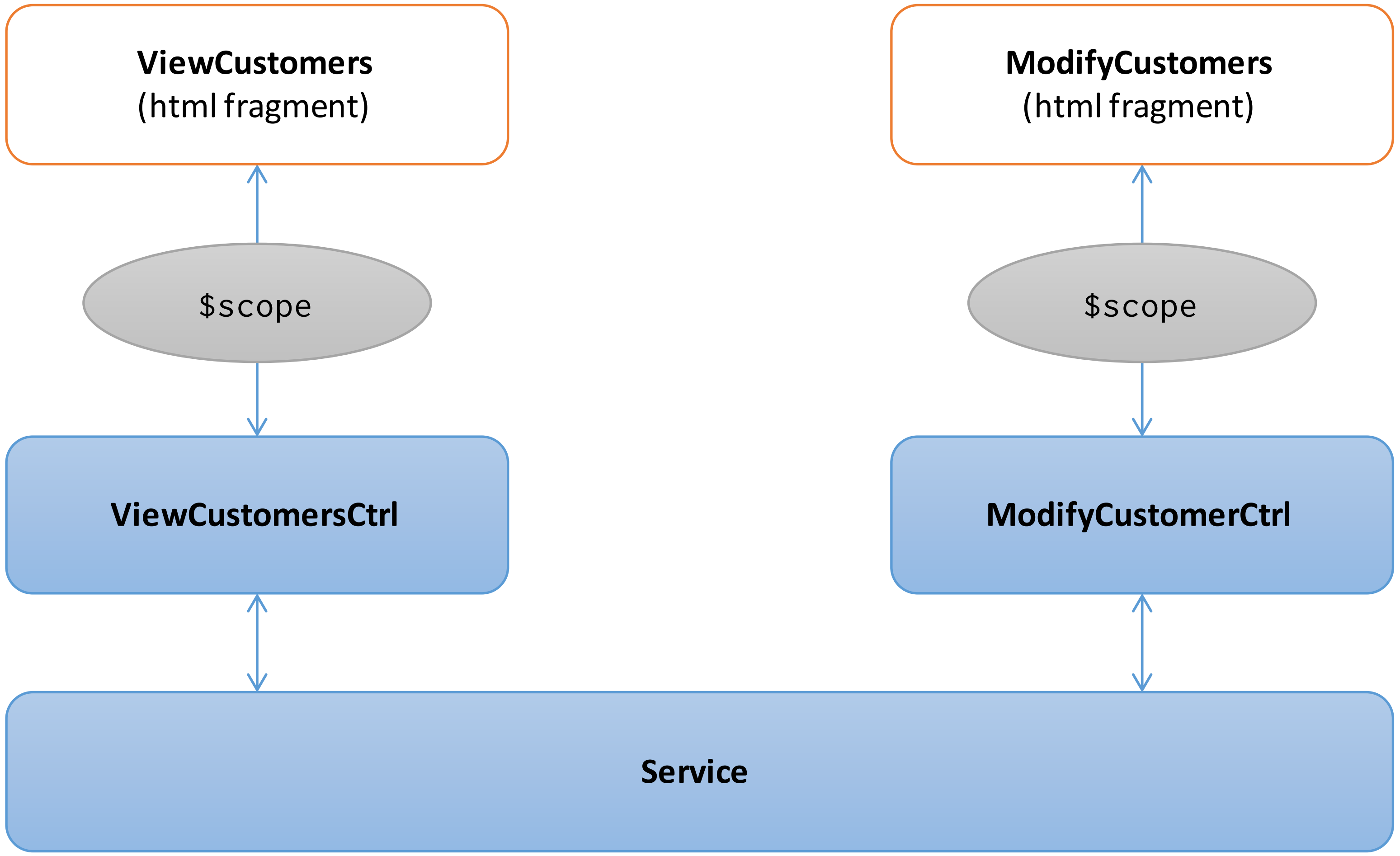
```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <meta charset="UTF-8" />
    <style media="screen"></style>
    <script src="../angular.min.js"></script>
    <script src="mymodule.js"></script>

  </head>
  <body>
    <div ng-app="myModule"
      <div ng-controller="MyCtrl">
        <p>Firstname: <input type="text" ng-model="model.firstname"></p>
        <p>Lastname: <input type="text" ng-model="model.lastname"></p>
        <p>{{model.firstname + " " + model.lastname}}</p>

        <button ng-click="click()">Show Number</button>

     </div>
    </div>
  </body>
</html>
```

This is now the model object from MyCtrl. Model object is shared with view and controller

Services

- View-independent **business logic** should **not be** in a controller
  - Logic should be in **a service component**
- **Controllers** are **view specific**, **services** are **app-spesific**
  - We can move from view to view and service is still alive
- Controller's responsibility is to bind model to view. Model can be fetched from service!
  - Controller is not responsible for manipulating (create, destroy, update) the data. **Use Services instead!**
- AngularJS **has many built-in services**, see
  - http://docs.angularjs.org/api/ng/service
  - Example: `$http`

Services



ViewCustomers
(html fragment)
ModifyCustomers
(html fragment)
$scope
$scope
ViewCustomersCtrl
ModifyCustomerCtrl
Service

Service, Provider, Factory?

- Three "Service" Types:
  1. **Factory**

     Creates **and returns a** factory service object
  2. **Service**

     Declares a function that is invoked **with new - keyword**
  3. **Provider**

     Can be configured

AngularJS Custom Services using Factory

```
// Let's add a new controller to MyApp. This controller uses Service!
myApp.controller('ViewCtrl', function ($scope, CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Let's add a new controller to MyApp. This controller uses Service!
myApp.controller('ModifyCtrl', function ($scope, CustomerService) {
    $scope.contacts = CustomerService.contacts;
});

// Creating a factory object that contains services for the
// controllers.
myApp.factory('CustomerService', function() {
    var factory = {};
    factory.contacts = [{name: "Jack", salary: 3000}, {name: "Tina", salary: 5000}, {name: 
"John", salary: 4000}];
    return factory;
});
```

Building Backend using NodeJS

Intro to Node.js

- Node.js cross-platform runtime environment for **server-side web apps ("backend")**
- Node.js apps are written in JavaScript
- Uses Google's V8 JS Engine
- Provides **built-in http server** without the need to use for example Apache HTTP Server
- Various modules
  - File System I/O, Networking (HTTP, TCP, UDP, DNS, SSL)
- Lot of frameworks built on top of Node.JS
  - Express.js for Web Apps
- Node.JS is **non-blocking, commands execute in parallel**

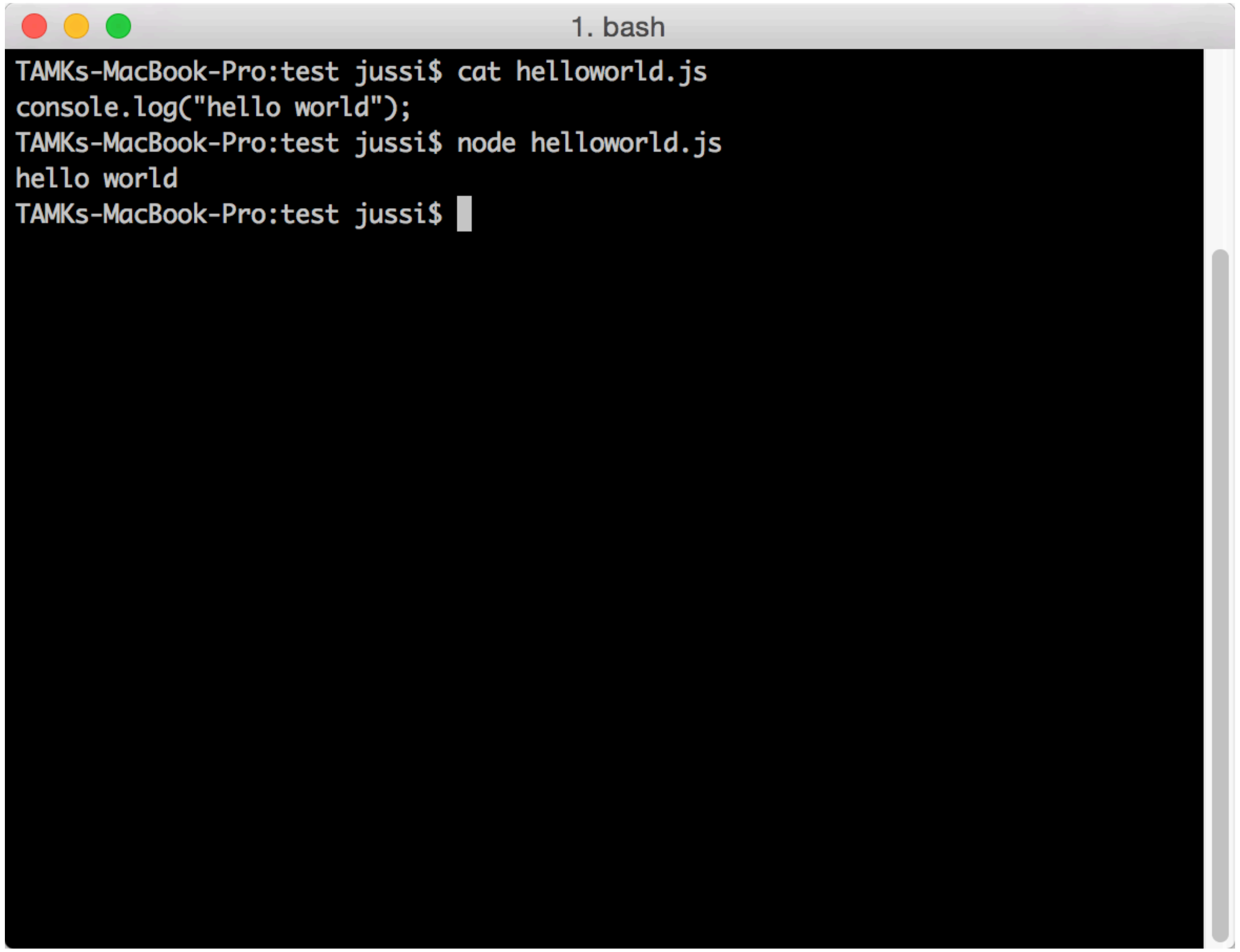
Node.JS Approach

- Usually we have Web server and some programming language: Apache Server combined with PHP
- Node.JS provides **both the server and the implementation** of the web application (JS)
    - One script handles all communication with the clients!
- Node.JS can communicate with Web Client, server filesystem, Restful APIs etc.
- Node.JS uses **event-driven model. IT'S FAST!**

Modules

- Node.js uses **module architecture**
- Each module contains set of functions you can use
  - **Http module contains functions specific to http**
- Node.js has **core modules** that are on by default
  - `https://nodejs.org/api`
- You must include a module
  - `var http = require('http');`
- To install new modules you can use npm
  - `npm install module_name`

Hello World

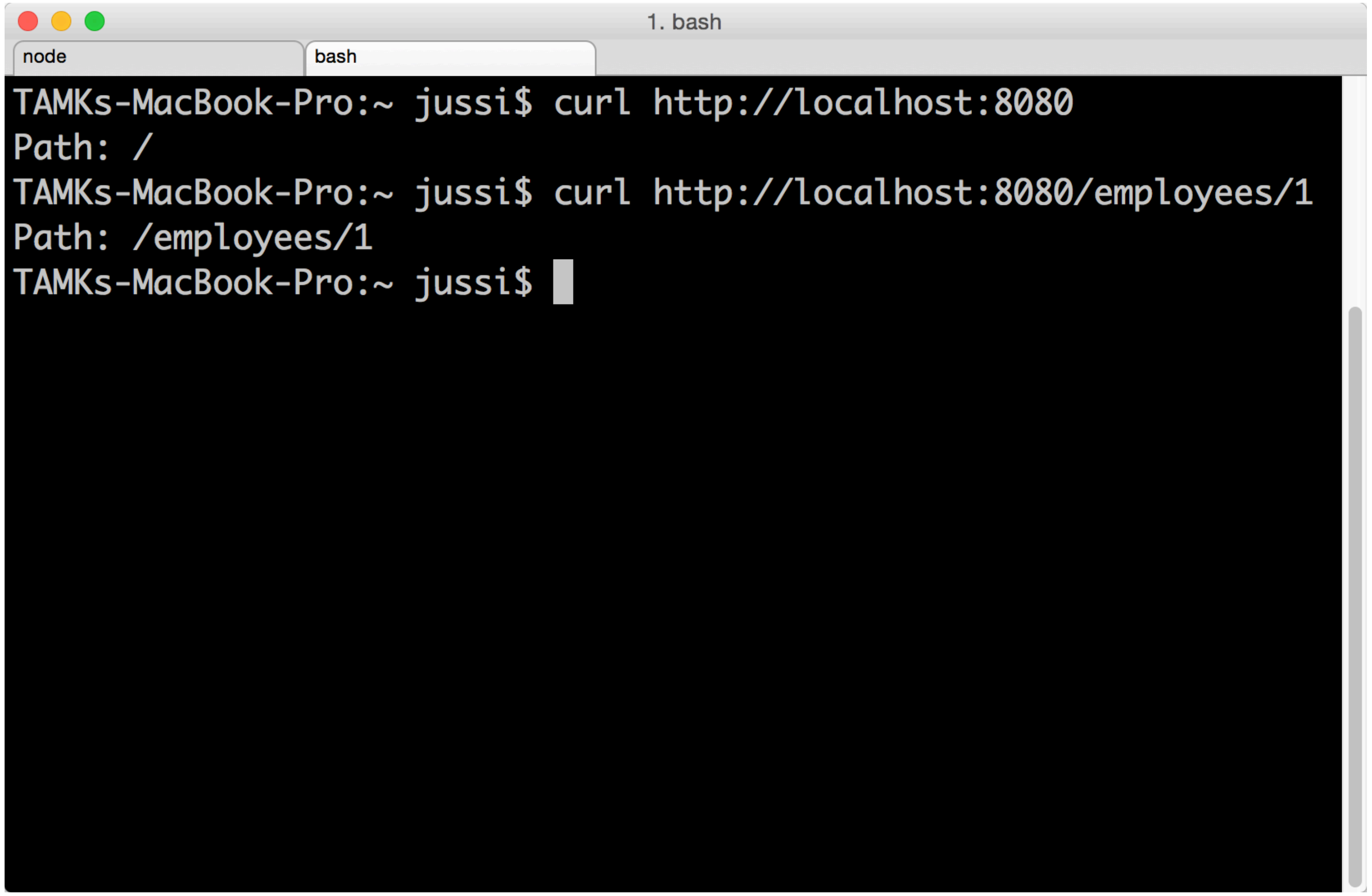


```
TAMKs-MacBook-Pro:test jussi$ cat helloworld.js
console.log("hello world");
TAMKs-MacBook-Pro:test jussi$ node helloworld.js
hello world
TAMKs-MacBook-Pro:test jussi$ 
```

Http Server

```
// Import http - module
var http = require('http');

function handleRequest(request, response){
    response.end('Path: ' + request.url);
}

// Create a server and provide a callback function
var server = http.createServer(handleRequest);

// Start the server in port 8080
server.listen(8080, function() {
    console.log("Server listening on: http://localhost:" + 8080);
});
```

cURL

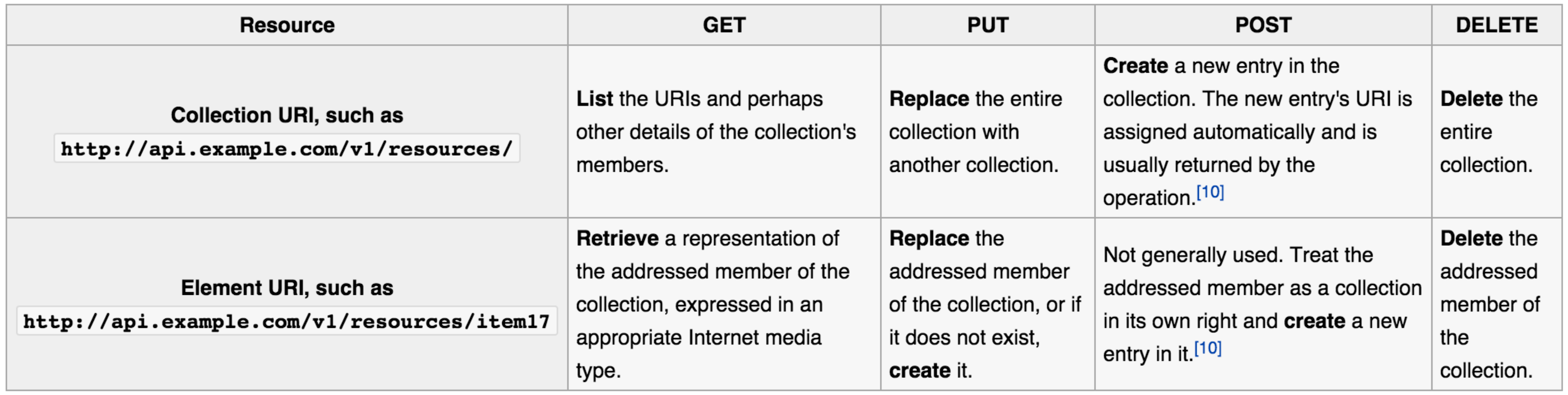


```
TAMKs-MacBook-Pro:~ jussi$ curl http://localhost:8080
Path: /
TAMKs-MacBook-Pro:~ jussi$ curl http://localhost:8080/employees/1
Path: /employees/1
TAMKs-MacBook-Pro:~ jussi$
```

Get and Post

```
"use strict";

// Import http - module
var http = require('http');

// Create a server and provide a callback function
var server = http.createServer(function (request, response) {
    response.end('Method: ' + request.method + "\n");
});

// Start the server in port 8080
server.listen(8080, function () {
    console.log("Server listening on: http://localhost:" + 8080);
});
```

REST Interface

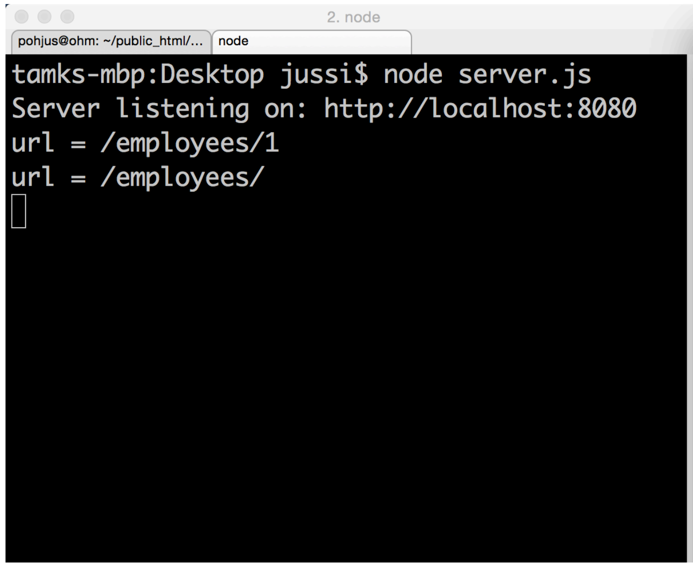

RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://api.example.com/v1/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[10]	**Delete** the entire collection.
Element URI, such as `http://api.example.com/v1/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[10]	**Delete** the addressed member of the collection.

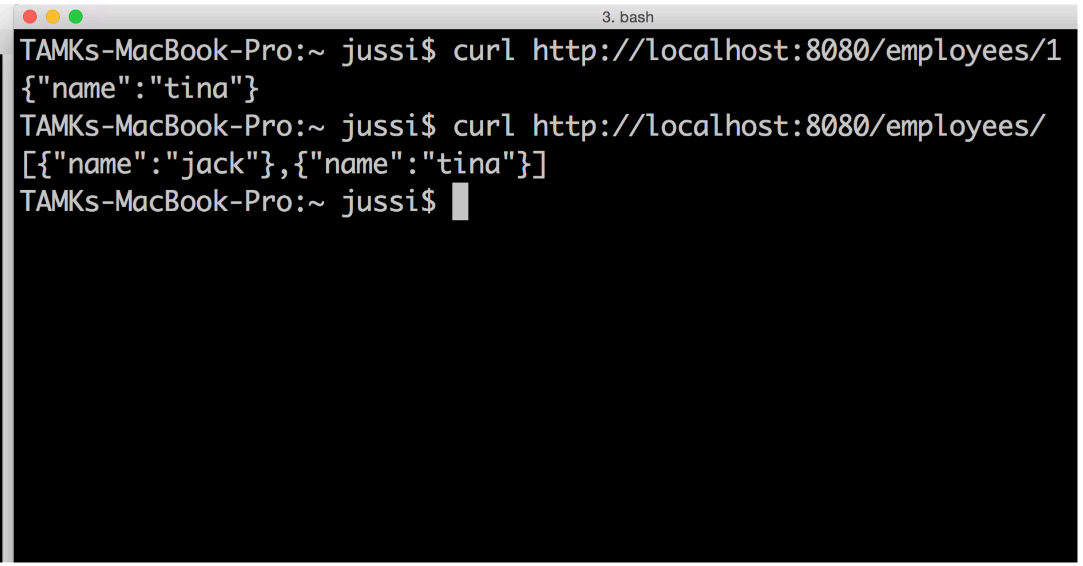
```
var dummyEmployeesData = [{name: "jack"}, {name: "tina"}];

// Import http - module
var http = require('http');

// Create a server and provide a callback function
var server = http.createServer(function (request, response) {
    // url          = "/employees/1"
    console.log("url = " + request.url);

    // splittedURL = [employees","1"]
    var splittedURL = splitUrl(request.url);

    switch(request.method) {
        case "GET":
            if(splittedURL.length == 2) {
                // is "employees/"?
                if(isCollectionURL(splittedURL)) {
                    response.write( JSON.stringify(dummyEmployeesData) );
                }
                // is "employees/X"?
                if(isElementURL(splittedURL)) {
                    // take X from from employees/X
                    var index = splittedURL[1];
                    if(index >= 0 && index < dummyEmployeesData.length) {
                        response.write( JSON.stringify(dummyEmployeesData[index]) );
                    }
                }
            }
            break;
        case "POST":
            console.log("POST: ");
            break;
    }

    response.end();
});
```

```javascript
// Start the server in port 8080
server.listen(8080, function () {
    console.log("Server listening on: http://localhost:" + 8080);
});

function splitUrl(url) {
    // array = ["", "employees", "1"];
    var array = url.split("/");
    // splittedURL = [employees","1"]
    array.shift();
    return array;
}

function isCollectionURL(array) {
    return (array[0] == "employees" && array[1] == "");
}

function isElementURL(array) {
    return (array[0] == "employees" && isNumber(array[1]));
}

function isNumber(number) {
    if(number.match(/^\d+$/)){
        return true;
    } else {
        return false;
    }
}
```

Result



```
tamks-mbp:Desktop jussi$ node server.js
Server listening on: http://localhost:8080
url = /employees/1
url = /employees/
```

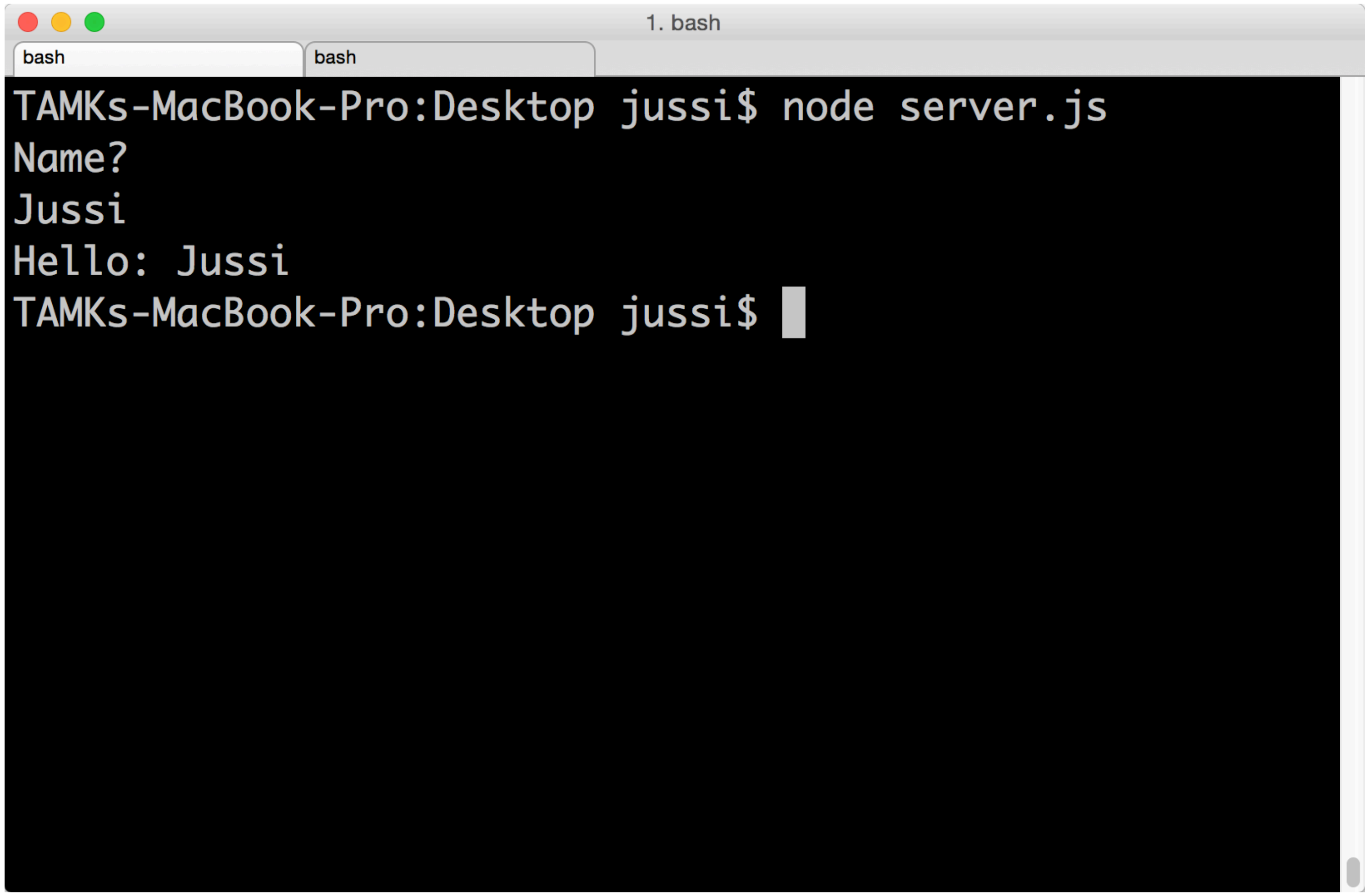


```
TAMKs-MacBook-Pro:~ jussi$ curl http://localhost:8080/employees/1
{"name":"tina"}
TAMKs-MacBook-Pro:~ jussi$ curl http://localhost:8080/employees/
[{"name":"jack"},{"name":"tina"}]
TAMKs-MacBook-Pro:~ jussi$
```

readline module

```
"use strict";

var readline = require('readline');

var rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.question("Name?\n", function(answer) {
    console.log("Hello:", answer);
    rl.close();
});
```

Usage of readline



```
TAMKs-MacBook-Pro:Desktop jussi$ node server.js
Name?
Jussi
Hello: Jussi
TAMKs-MacBook-Pro:Desktop jussi$ 
```

fs module

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

var dummyEmployeesData = [{name: "jack"}, {name: "tina"}];

fs.writeFile("./text.txt", JSON.stringify(dummyEmployeesData), function(err) {
    fs.readFile("./text.txt", function(err, data) {
        var array = JSON.parse(data);
        console.log(array[0].name);
    });
});
```

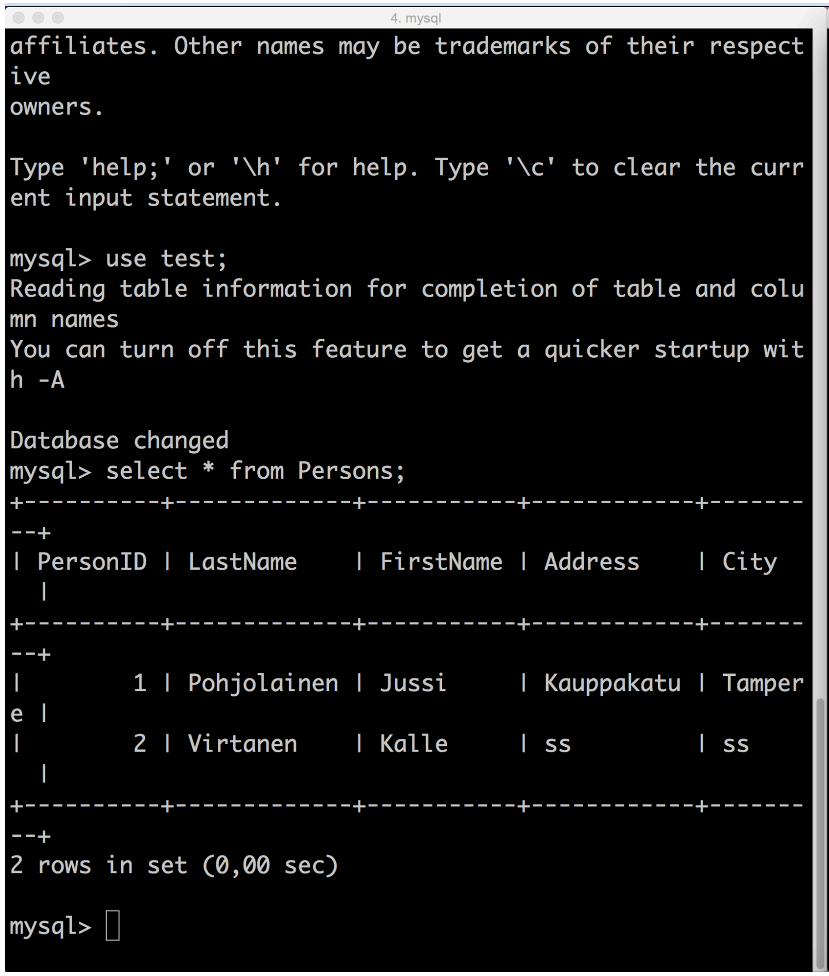
npm package manager

- Package manager for JS
  - `https://www.npmjs.com/`
- Node comes with npm package manager
  - `npm -version`
- npm makes it easy to share and reuse code
- Example:
  - https://www.npmjs.com/package/mysql
- Install
  - `npm install mysql`

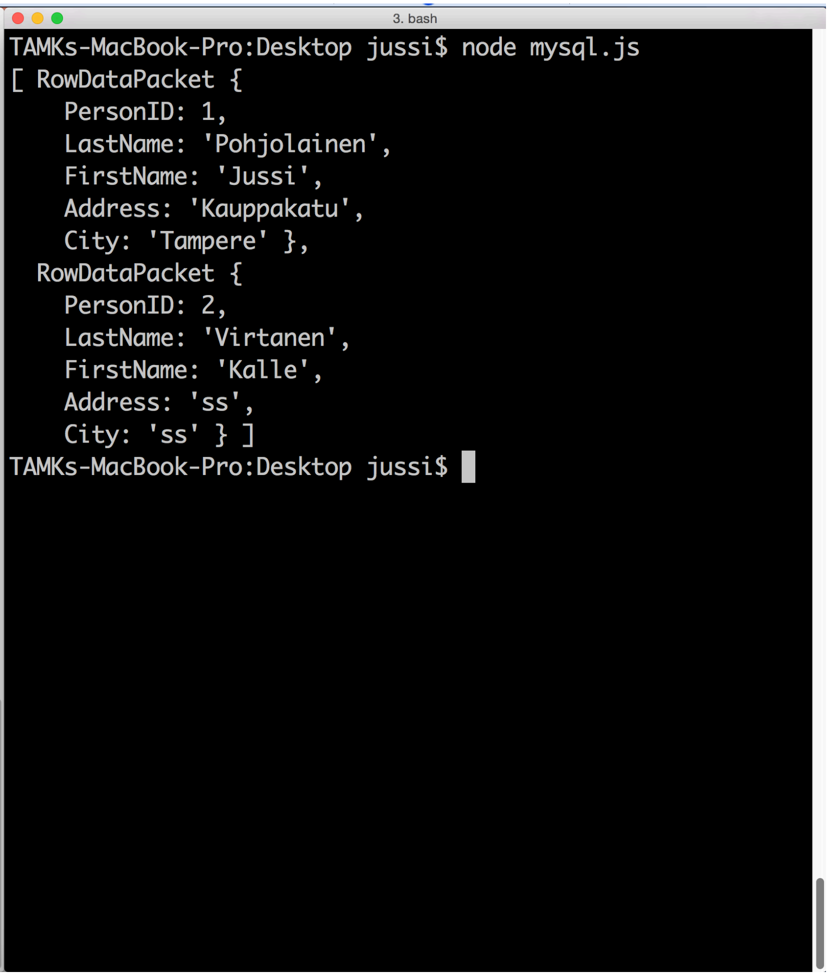
MySQL module

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host     : 'localhost',
  user     : '',
  password : '',
  database : 'test'
});

connection.connect();

connection.query('SELECT * FROM Persons', function(err, rows) {
    console.log(rows);
});

connection.end();
```

MySQL Usage



```
affiliates. Other names may be trademarks of their respect
ive
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the curr
ent input statement.

mysql> use test;
Reading table information for completion of table and colu
mn names
You can turn off this feature to get a quicker startup wit
h -A

Database changed
mysql> select * from Persons;
+----------+------------+-----------+-----------+-------
--+
| PersonID | LastName   | FirstName | Address   | City
  |
+----------+------------+-----------+-----------+-------
--+
|        1 | Pohjolainen | Jussi    | Kauppakatu | Tamper
e |
|        2 | Virtanen   | Kalle     | ss        | ss
  |
+----------+------------+-----------+-----------+-------
--+
2 rows in set (0,00 sec)

mysql> 
```



```
TAMKs-MacBook-Pro:Desktop jussi$ node mysql.js
[ RowDataPacket {
    PersonID: 1,
    LastName: 'Pohjolainen',
    FirstName: 'Jussi',
    Address: 'Kauppakatu',
    City: 'Tampere' },
  RowDataPacket {
    PersonID: 2,
    LastName: 'Virtanen',
    FirstName: 'Kalle',
    Address: 'ss',
    City: 'ss' } ]
TAMKs-MacBook-Pro:Desktop jussi$ 
```

Enhancing Backend with Express.js and MongoDB

Express.js

- **Express.js** is a Node.js web application server framework
- Gives you a layer of fundamental web application features
- To install:
  - `npm install express`

Web Server

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

// Listen requests for root (/)
app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  console.log('Server listening');
});
```

Creating package.json

- To create node.js app:
  - `npm init`
- Answer to the questions about your app
- To add module to your project and depencies
  - `npm install express --save`

package.json

```json
{
  "name": "routing",
  "version": "1.0.0",
  "description": "",
  "main": "routing.js",
  "scripts": {
    "start": "node routing.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.13.3"
  }
}
```

Routing

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

// respond with "Hello World!" on the homepage
app.get('/', function (req, res) {
  res.send('Hello World!');
});

// accept POST request on the homepage
app.post('/', function (req, res) {
  res.send('Got a POST request');
});

// accept PUT request at /user
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
});

// accept DELETE request at /user
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user');
});

var server = app.listen(3000, function () {
  console.log('Server listening');
});
```

Routing: regex

```
"use strict";
var express = require('express');
var app = express();

app.get(/\/employees\/1$/, function(req, res) {
  res.send('/employees/1');
});

var server = app.listen(3002, function () {
  console.log('Server listening');
});
```

Routing using String patterns

```
// will match acd and abcd
app.get('/employees/:id', function(req, res) {
  res.send('ab?cd');
});

// will match abcd, abbcd, abbbcd, and so on
app.get('/ab+cd', function(req, res) {
  res.send('ab+cd');
});

// will match abcd, abxcd, abRABDOMcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});

// will match /abe and /abcde
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
```

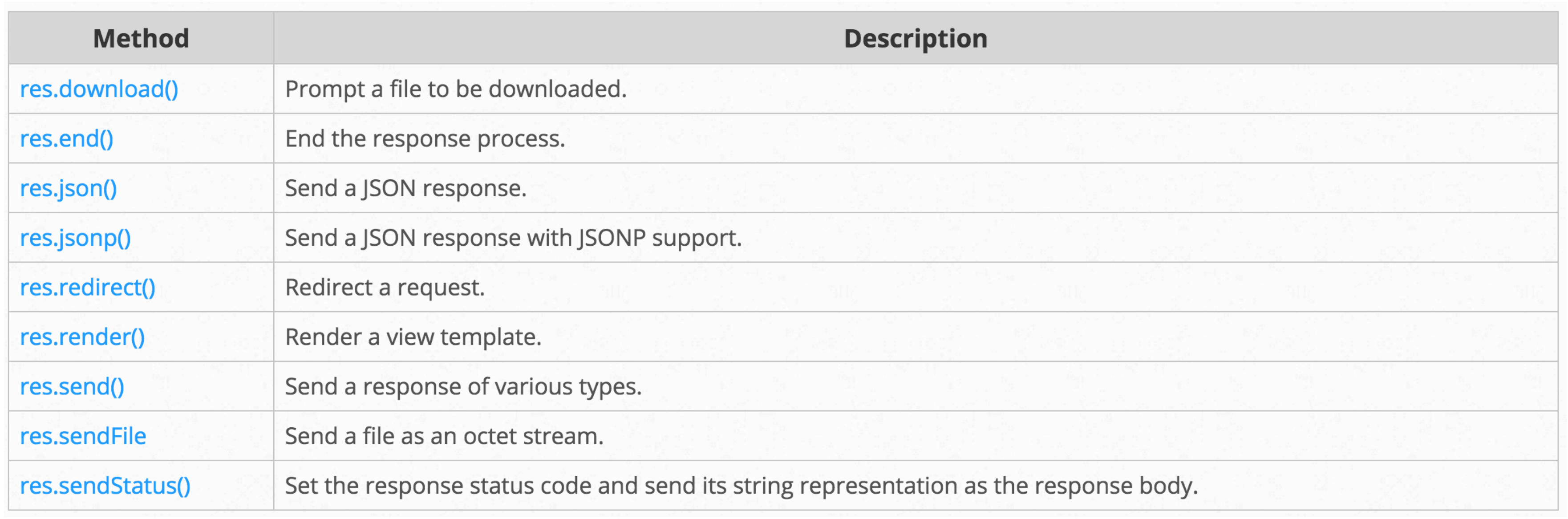
Uses path-to-regexp: `https://www.npmjs.com/package/path-to-regexp`

Routing using String patterns

```
// will match abcd, abbcd, abbbcd, and so on
app.get('/ab+cd', function(req, res) {
  res.send('ab+cd');
});

// will match abcd, abxcd, abRABDOMcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});

// will match /abe and /abcde
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
// will match employees/1
app.get('/employees/:id', function(req, res) {
  res.send(req.params.id);
});
```

Uses path-to-regexp: `https://www.npmjs.com/package/path-to-regexp`

Response methods



Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

Routing: string patterns and json

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

var data = [{name:"Tina"}, {name: "Jack"}];

// http://localhost:8080/employees or
// http://localhost:8080/employees/
app.get("/employees(\/)?$", function(req, res) {
  res.json(data);
});

// http://localhost:8080/employees/<INTEGER>
app.get("/employees\/[0-9]+?$", function(req, res) {
  res.json(data[0]);
});

var server = app.listen(3000, function () {
  console.log('Server listening');
});
```

Routing: result

```
curl -v http://localhost:3000/employees/

*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET /employees/ HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: application/json; charset=utf-8
< Content-Length: 33
< ETag: W/"21-erEHjuT/VGICHeMkZz+ZDw"
< Date: Sun, 20 Sep 2015 10:25:44 GMT
< Connection: keep-alive
<
* Connection #0 to host localhost left intact
[{"name":"Tina"},{"name":"Jack"}]
```

You can use parameters in String Patterns

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

var data = [{name:"Tina"}, {name: "Jack"}];

// http://localhost:3000/employees or
// http://localhost:3000/employees/
app.get("/employees(\/)?$", function(req, res) {
  res.json(data);
});

// http://localhost:8080/employees/<INTEGER>
app.get("/employees\/:id([0-9]+?)$", function(req, res) {
  res.json(data[req.params.id]);
});

var server = app.listen(3000, function () {
  console.log('Server listening');
});
```

Middleware

- Middleware is a **function** that receives request and response
- It may transform these objects before passing them to the next middleware function
- It may decide to write to the response or end the response

Example

```
function logger(req,res,next){
  console.log(new Date(), req.method, req.url);
  next();
}

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

// Application level middleware!
app.use(logger);

// ...
```

Example

```
var data = [{name:"Tina"}, {name: "Jack"}];

function validate(req, res, next) {
    if(req.params.id >= 0 && req.params.id < data.length) {
        next();
    } else {
        res.json({});
        res.end();
    }
}

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

app.get("/employees(\/)?$", function(req, res) {
  res.json(data);
});

app.get("/employees\/:id([0-9]+?)$", validate, function(req, res) {
  res.json(data[req.params.id]);
});

var server = app.listen(3000, function () {
  console.log('Server listening');
});
```

MongoDB

Intro to MongoDB

- Open source **document** database
- Database in MongoDB is a container for **collections**
- **Collection** is a group of MongoDB documents
- A record is a **document with field value pairs (json)**
- Documents are stored in **collections (tables in relational database)**

Install and usage

- See:
  - http://docs.mongodb.org/master/tutorial/
- Create directory for data
- Commands
  - `mongod – Starts process`
  - `mongo – Open CLI`

Commands

- Show all databases
  - `show databases`
- Create/use database
  - `use mydatabase`
  - *New database can be seen if it contains collections!*
- What database am I using
  - `db`
- Create new collection to current database with content
  - `db.newcollection.insert({name: ”Jack”})`
- Show collections
  - `show collections`

Commands

- Delete collection
  - `db.`*`mycollection`*`.drop()`
- Insert document to collection
  - `db.`*`mycollection`*`.insert({name: "Jack"})`
- Show all documents in a collection (Every document has unique id)
  - `db.`*`mycollection`*`.find()`
- Show particular document
  - `db.`*`mycollection`*`.find({name: "Jack"})`
- Import from file
  - `mongoimport --db test --collection mycollection --drop --file data.json`
  - Expects a JSON Object {} in file. **If array**, use --jsonArray to fetch array elements to different documents

Creating project in Node.JS

- Create project
  - `npm init`
- Add express and mongodb
  - `npm install express mongodb --save`
- Create javascript file that access the mongodb
  - `https://www.npmjs.com/package/mongodb`

Example: insert

```
var mongodb = require('mongodb')
var mongoClient = mongodb.MongoClient;

// See connection string
//    http://docs.mongodb.org/manual/reference/connection-string/
var url = 'mongodb://localhost:27017/test';

// Use connect method to connect to the Server
mongoClient.connect(url, function(err, db) {
  console.log("Connected correctly to server");

  var collection = db.collection('ankkalinna');

  collection.insert({name: "Aku"}, function(err, result) {
     if(err != null) {
        console.log("error");
     } else {
        console.log("success");
     }
     db.close();
  })
});
```

Example: find

```
var mongodb = require('mongodb')
var mongoClient = mongodb.MongoClient;

var url = 'mongodb://localhost:27017/test';

mongoClient.connect(url, function(err, db) {
  console.log("Connected correctly to server");

   var collection = db.collection('customers');

   // find and toArray
   // http://docs.mongodb.org/master/reference/method/cursor.toArray/
   collection.find({}).toArray(function(err, documents) {
        if(err == null) {
            console.log(documents);
        }

        db.close();
    })
});
```

Combine Express + MongoDB

rest.js

```
var express = require('express');
var myMongo = require('./my-mongo');

var app = express();

app.get("/employees(\/)?$", function(req, res) {
  myMongo.fetchFromDatabase({}, function(resultData) {
    res.json(resultData);
  });
});

app.get("/employees\/:id([0-9]+?)$", function(req, res) {
  var searchObject = {id: Number(req.params.id)};
  myMongo.fetchFromDatabase(searchObject, function(resultData) {
    res.json(resultData[0]);
  });
});

var server = app.listen(3000, function () {
  console.log('Server listening');
});
```

my-mongo.js

```
var mongodb = require('mongodb')

var mongoClient = mongodb.MongoClient;
var url = 'mongodb://localhost:27017/test';

function fetchFromDatabase(searchParameter, callback) {

    mongoClient.connect(url, function(err, db) {

       var collection = db.collection('customers');

       // find and toArray
       // http://docs.mongodb.org/master/reference/method/cursor.toArray/
       collection.find(searchParameter).toArray(function(err, documents) {
            if(err == null) {
                callback(documents);
            }

            db.close();
        })
    });
}

exports.fetchFromDatabase = fetchFromDatabase;
```