HTML5 Programming Week

Kleiner Technologie-Überblick

Unser Job

- Kennenlernen von HTML5 und CSS3
- **Webentwicklung nach allen Regeln der Kunst!**
- Basteln einer Beispielapp (RoomR)

Der Plan

1. Programmierer programmieren
2. Organisatoren organisieren
3. **Erklärbären erklären**

Ausrüstungscheck

- Rechner mit IDEs u.Ä.?
- Node.js installiert?
- Git installiert?
- PhantomJS installiert?

Die Technik

- Buildsystem: Grunt [1]
- CSS-Präprozessor: Stylus [2] + Nib [3]
- Programmiersprache: CoffeeScript [4] + Docco [5]
- Modulsystem: AMD [6] + RequireJS [7]
- Testsystem: QUnit [8] + PhantomJS [9]
- Webframework: Backbone [10] + Underscore [11] + jQuery [12]
- Ein Haufen **HTML5** und **CSS3**
- **Denken wie ein moderner Frontend-Webentwickler!**

[1] http://gruntjs.com/
[2] http://learnboost.github.com/stylus/
[3] http://visionmedia.github.com/nib/
[4] http://coffeescript.org/
[5] http://jashkenas.github.com/docco/
[6] https://github.com/amdjs/amdjs-api/wiki/AMD
[7] http://requirejs.org/
[8] http://docs.jquery.com/QUnit
[9] http://code.google.com/p/phantomjs/
[10] http://backbonejs.org/
[11] http://underscorejs.org/
[12] http://jquery.org/

Moderne Frontend-Nerds

- **Navigieren scheinbar unorganisiert durch ein gewaltiges Chaos.**
- Eigenschaften:
  - Kleine vor großen Lösungen (Unix-Philosophie)
  - Schicke Tools lieber als funktionale
  - Zuweilen Aufmerksamkeitsspanne einer Waldameise
- Das sind alles **Methoden** zum Umgang mit dem Chaos!

The secret to writing large apps in JavaScript is **not to write large apps**. Write many small apps that can talk to each other.

- @derickbailey

Die Vorlage

- Beispielapp herunterladen: `git clone https://github.com/SirPepe/HTML5Workshop2.git`
- Abhängigkeiten installieren: `npm install`
- Testbuild: `grunt`
- **Klappt das?**

Die Technik im Detail

Der Buildprozess

- Grunt [13] : taskbasiertes CLI-Tool auf Basis von Node.js
- Vorteile: Simpel, taskbasiert, ein paar eingebaute Tasks, JavaScript
- Nachteile: Nagelneu, JavaScript
- Task-Repo: github.com/gruntjs/grunt-contrib [14]

[13] https://github.com/cowboy/grunt
[14] https://github.com/gruntjs/grunt-contrib

So funktioniert Grunt

1. Tasks als JS-Files oder NPM-Module anlegen
2. Konfiguration in `grunt.js` im Projektverzeichnis (Details später)
3. Aufruf: `~/Projektverzeichnis $ grunt`

Alles klar zum Buildprozess?

Next: CSS

CSS-Sprache: Stylus [15]

- Auf Node.js basierender **CSS-Präprozessor**
- Kompiliert eine eigene Syntax zu CSS3
- Features: Variablen, Selektorverschachtelung Funktionen, freie Syntax, Modulsystem …
- Vorteil: Extrem mächtig
- Nachteil: Nicht einfach zu beherrschen

[15] http://learnboost.github.com/stylus/

```
// Variablen, Farbmanipulation
colorNormal = darken(green, 10)
colorActive = #C00

/* Links in Kopf- und Fußbereich */
#header, #footer
  a
    &:link, &:visited
      text-decoration underline
      color colorNormal
    &:hover, &:active
      text-decoration none
      color colorActive
```

```
#header a:link,
#footer a:link,
#header a:visited,
#footer a:visited {
  text-decoration: underline;
  color: #004d00;
}
#header a:hover,
#footer a:hover,
#header a:active,
#footer a:active {
  text-decoration: none;
  color: #c00;
}
```

```
// @extends
form
  input[type=text]
    padding: 5px
    border: 1px solid #eee
    color: #ddd
textarea
  @extends form input[type=text]
  padding: 10px
```

```
form input[type=text],
form textarea {
  padding: 5px;
  border: 1px solid #eee;
  color: #ddd;
}
textarea {
  padding: 10px;
}
```

```
// Vendor-Prefix-Funktion
vendor(prop, args...)
  for pref in webkit moz ms o
    {'-' + pref + '-' + prop}: args
  {prop}: args

#Foo
  vendor(border-radius, 8px)
```

```
#Foo {
  -webkit-border-radius: 8px;
  -moz-border-radius: 8px;
  -ms-border-radius: 8px;
  -o-border-radius: 8px;
  border-radius: 8px;
}
```

CSS-Erweiterungen: Nib [16]

- Kein NIH für DRY!
- Library mit nützlichen Defaults, Mixins und Funktionen
- Besten Überblick bietet das Git-Repo selbst: github.com/visionmedia/nib/tree/master/lib/nib [17]

[16] https://github.com/visionmedia/nib
[17] https://github.com/visionmedia/nib/tree/master/lib/nib

```
// Fertiger Clearfix
@import "nib/clearfix"

#wrapper
  clearfix()
```

```
// Automatische Vendor-Prefixes
@import "nib/vendor"

*
  box-sizing border-box
#wrapper
  border-radius(8px)
  box-shadow(0.2em 0.2em 1.5em #999);
```

```
// Kürzere Position-Angaben
@import "nib/position"

#foo
  absolute top left
```

Stylus und Nib in der App

- Stylus und Nib sind fertig installiert
- Ort der Haupt-Datei: `src/style/main.styl`
- Ort für Module: `src/style/lib/`
- Ort für Bilder: `src/style/img/`
- How to: **Änderungen machen, speichern, Build durchführen, fertig!** (Output wird automatisch zusammengefügt und optimiert)

```
// Stylus im Buildprozess
stylus: {
  main: {
    file: 'src/style/main.styl', // Quelldatei
    dest: 'app.css',             // Zieldatei
    options: {                   // Zusatzoptionen
      url: {
        paths: [ 'src/style' ]   // Pfad(e) für Bilder
      },
      compress: true             // Code-Komprimierung an/aus
    }
  }
},
```

Alles klar zu Stylus und Nib?

Next: CoffeeScript

Programmiersprache: CoffeeScript [18]

- **Kleine Programmiersprache, die zu JavaScript kompiliert wird**
- Einflüsse: Ruby, Python
- Features: Bereinigte Syntax, Klassensystem, viele kleine Sprachfeatures
- Vorteil: Lässt viele Probleme von JS verschwinden, mittlerweile recht populär [19]
- Nachteil: Geschmacksfrage, resultierender Code schwerer zu debuggen

[18] http://coffeescript.org/
[19] https://github.com/languages

Warum kein JavaScript?

```
// Was kommt hier raus?
42.toString();

// True oder false?
new Boolean(false) == true;

// True oder false?
{} == 1;

// Was passiert hier?
if(true){
    function foo(){
        return "wtf";
    }
}
foo();
```

JS == WTF

```
// Was kommt hier raus?
42.toString(); // SyntaxError: Unexpected token ILLEGAL

// True oder false?
new Boolean(false) == true; // true

// True oder false?
{} == 1; // Weder noch! SyntaxError: Unexpected token ==

// Was passiert hier?
if(true){
    function foo(){
        return "wtf";
    }
}
foo(); // Abhängig von der JS-Engine (verboten in ES5)
```

Kein Problem in CoffeeScript

```
# Was kommt hier raus?
42.toString() # String "42"

# True oder false?
{} == 1 # false

# True oder false?
new Boolean false == true # false - Vergleich mit ===

# Was passiert hier?
if true
  foo = ->
    return "wtf"
foo()  # String "wtf"
```

CoffeeScript-Features

```
# Automatische var-Deklaration
# Sparsame Funktions-Syntax
# Strukturierung durch Whitespace
# Vorgabewerte für Parameter
# Automatisches return
square = (x = 42) ->
  x * x

# Runde Klammern optional
ergebnis = square 1337
```

CoffeeScript-Features

```
# Kommata (und Semikolons) vor Zeilenumbrüchen optional
books = [
  { name: 'Dune', rating: 5 }
  { name: 'Red Mars', rating: 4 }
  { name: 'Altered Carbon', rating: 3 }
]

# Einfache Comprehensions
good = (b.name for b in books when b.rating > 3)
```

CoffeeScript-Features

```
# `unless` als Gegenstück zu `if` (auch: `until` für `while`)
# Kein Klammer-Infarkt mehr bei Callbacks
# Existenz-Operator (`?`)
# Postfix-If (und `unless`)
unless offline
  $.get 'foo/ajax.txt', (result) ->
    alert result if result?
```

CoffeeScript-Features

```
zahl = Number window.prompt 'Gib eine Zahl ein'

# Alles ist ein Ausdruck (selbst try/catch)
# Fallthrough bei `switch` wird verhindert
status = switch zahl
  when 23 then 'ein Nerd'
  when 42 then 'ein Geek'
  else
    # Verkettete Vergleiche
    if 9000 > zahl > -9000
      'langweilig'
    else
      'größenwahnsinnig'

# String-Interpolation
window.alert "Du bist #{status}"
```

CoffeeScript-Features

```
class Tier                              # Klassen
  energie: 0
  constructor: (@name = 'Bernd') ->  # Parameter-Defaults
  friss: -> @energie++                  # @ = this.
  stirb: -> @energie = -1

class Schlange extends Tier
  friss: (opfer) ->
    opfer.stirb()
    super()                          # einfaches `super()`

bernd = new Tier()
august = new Schlange('August')

august.friss bernd
window.alert august.energie
```

CoffeeScript-Nachteile

- Große Ausdrucksfreiheit (mehr nachdenken erforderlich)
- Whitespace kann nerven (v.a. bei Wrapperfunktionen)
- Debuggt werden muss das kompilierte JavaScript

Dokumentation: Docco [20]

- **Ultra-Simple Dokumentationsgenerierung** mit Ausrichtung auf literate programming
- Kommentare können mit Markdown [21] formatiert werden

```
# Model für Pages
# ---------------

# [Im Beispiel](https://github.com/SirPepe/HTML5Workshop2/)
# wird der Inhalt der Hauptspalte durch ein **Model**
# repräsentiert.

PageModel = Backbone.Model.extend { }
Page = new PageModel()
```

[20] http://jashkenas.github.com/docco/
[21] http://daringfireball.net/projects/markdown/

As the years go by, more and more of my code comments are turning into complete sentences; grammar does enhance communication.

- @ID_AA_Carmack

CoffeeScript in der App

- CoffeeScript ist installiert
- App-Code: `src/script`
- Module: `src/script/lib`
- Tests: `test`
- Docs: `docs`
- How to: **Änderungen machen, speichern, Build durchführen, fertig!** (Output wird automatisch zusammengefügt und optimiert, Docs werden automatisch erzeugt und Tests durchgeführt)

```
// CoffeeScript im Buildprozess
coffee: {
  src: ['src/**/*.coffee'],  // Alle .coffee-Files in `src`
  test: ['test/**/*.coffee'] // Alle .coffee-Files in `test`
},
```

```
// Docco im Buildprozess
docco: {
  src: {
    files: [              // Zu dokumentierende Files
      'src/**/*.coffee',
      'grunt.js'
    ],
    dest: ''              // Zielverzeichnis für `/docs`
  }
},
```

Alles klar zu Stylus und Nib?

Next: Module

Modulsystem: AMD [22] + RequireJS [23]

- AMD = Community-Standard für JS-Module
- RequireJS = Scriptloader + Optimierer (auf Basis von UglifyJS)
- Features: Dependencymanagement, automatische Optimierung
- Nachteil: Boilerplate-Wrapperfunktion für den gesamten Modulcode, nicht alle Libs sind AMD-Ready

[22] https://github.com/amdjs/amdjs-api/wiki/AMD
[23] http://requirejs.org/

Das ist kein Modulsystem …

```
<script src="jquery.js"></script>
<script src="underscore.js"></script>
<script src="backbone.js"></script>
<script src="coffee-script.js"></script>

<script src="app.js"></script>
```

… das hingegen schon!

```
# modul.coffee
define ->
  return {
    getAntwort: -> 42
  }


# app.coffee
require ['modul'], (modulApi) ->
  window.alert modulAPI.getAntwort()


<!-- index.html -->
<script data-main="app.js" src="require.js"></script>
```

RequireJS-Optimizer

- CLI-Tool
- Fügt alle JS-Files zu einer einzigen Datei zusammen
- Optimiert den Code mit UglifyJS [24]

[24] https://github.com/mishoo/UglifyJS

AMD und RequireJS in der App

- Alle Vorlagen sind AMD-Module
- Alle Libraries sind angepasste AMD-Builds
- Die HTML-Datei verwendet RequireJS
- Der Buildprozess nutzt den Optimizer (Task grunt-require [25])
- How to: **Änderungen machen, speichern, Build durchführen, fertig!**

[25] http://asciidisco.github.com/grunt-requirejs/

```
// RequireJS im Buildprozess
requirejs: {
  baseUrl: './src/script',  // Script-Source-Verzeichnis
  paths: {                  // Pfade für Libraries mit doofen Dateinamen
    'jquery': 'lib/vendor/jquery-1.7.2',      // Versionsnummer
    'backbone': 'lib/vendor/backbone-amd',    // AMD-Build
    'underscore': 'lib/vendor/underscore-amd' // AMD-Build
  },
  name: 'main',             // Wurzelmodul
  out: 'app.js'             // Zieldatei
  //optimize: 'none'        // Optimierung deaktivieren
},
```

Alles klar zu Stylus und Nib?

Next: QUnit

Testframework: QUnit [26]

- **Das Testsystem von jQuery**

```
# Das zu testende Modul laden
require ['testModule'], (testModule) ->
  'use strict'

  $(document).ready ->

    # `module()` teilt eine Testdatei in Module auf
    module 'Modul testModule'

    # In`test()` finden die eigentlichen Tests statt
    test 'Funktion testModule.foo()', ->
      equal testModule.foo(), 'foo'
      ok testModule.isAwesome
```

[26] http://docs.jquery.com/QUnit

Was kann QUnit?

- Einfache Assertions (`ok`, `equal`, `notEqual`)
- Deep und Strict Assertions (`deepEqual`, `notDeepEqual`, `strictEqual`, `notStrictEqual`)
- Exceptions: (`raises`)
- Asynchrone Tests (`start`, `stop`)

Das getestete Modul

```
define -> {
    foo: -> 'foo'
    isAwesome: yes
  }
```

QUnit HTML-Datei

```
<!doctype html>
<title></title>
<meta charset="utf-8">

<!-- Das allgegenwärtige jQuery -->
<script src="jquery.js"></script>

<!-- Das Testsystem selbst -->
<script src="qunit.js"></script>
<link rel="stylesheet" href="qunit.css">

<!-- Das Test-Script mit Require.JS laden -->
<script src="require.js" data-main="testModuleTest.js"></script>

<!-- Test-Markup, immer erforderlich -->
<h1 id="qunit-header">Test von testModule</h1>
<h2 id="qunit-banner"></h2>
<div id="qunit-testrunner-toolbar"></div>
<h2 id="qunit-userAgent"></h2>
<ol id="qunit-tests"></ol>
<div id="qunit-fixture"></div>
```

Fluch und Segen: Browser

- **QUnit läuft im Borwser!**
- Vorteil: Browser sind unsere Zielplattform, wir brauchen Tests mit DOM etc.
- Nachteil: Manuelles Aufrufen der Testseite im Browser ist blöd
- Lösung: Tests in PhantomJS [27] (Headless Browser) durchführen!

[27] http://code.google.com/p/phantomjs/

QUnit in der App

- Ein Grunt-Task ist eingerichtet
- Führt automatisch Tests mit allen HTML-Dateien in `test` durch
- How to: **Build durchführen, fertig!** (Grunt bricht ab wenn ein Test fehlschlägt)

```
// QUnit im Buildprozess
qunit: {
  all: ['test/*.html']
},
```

Alles klar zu Stylus und Nib?

Next: Backbone

Architektur: Backbone.js [28]

- Clientseitige „MVC“-Bausteine (MVP? Whatever …)
- Setzt auf jQuery [29] und Underscore.js [30] auf
- Backbone liefert: Basisklassen (Models, Collections, Router, Views) und Konventionen
- Backbone liefert nicht: Struktur/Framework (Erweiterungen in diese Richtung: Aura [31] , LayoutManager [32] , Marionette [33] , Chaplin [34] uvm.)

[28] http://backbonejs.org/
[29] http://jquery.com/
[30] http://underscorejs.org/
[31] https://github.com/addyosmani/backbone-aura
[32] https://github.com/tbranyen/backbone.layoutmanager
[33] https://github.com/derickbailey/backbone.marionette
[34] https://github.com/chaplinjs/chaplin

Backbone-Klassen in Aktion

```
# extend() erstellt ein anpassbares Abziehbild der
# Basisklasse Backbone.Model
Item = Backbone.Model.extend()

# Collections sammeln Models
ItemCollection = Backbone.Collection.extend {
  model: Item
}
Items = new ItemCollection()

# Views sind mehr Konvention als Code
ItemView = Backbone.View.extend {
  render: ->
    @$el.html @model.get('text')
    $('body').append @$el
    return this
}
```

Backbone-Klassen nutzen

```
# Auf neue Einträge in der Collection reagieren
Items.on 'add', (newModel) ->
  new ItemView({
    model: newModel
  }).render()

# Neue Models anlegen...
item1 = new Item().set('text', 'Hallo!')
item2 = new Item().set('text', 'Welt!')

# ... und in die Collection stecken
Items.add [item1, item2]
```

Backbone und REST

```
# Collections sammeln Models
ItemCollection = Backbone.Collection.extend {
  model: Item
  url: '/api/items/'  # Wo liegt die API?
}
Items = new ItemCollection()

# `fetch()` holt Daten von der Datenquelle
# Das Config-Objekt entspricht dem von `jQuery.get()`
Items.fetch {
  success: (collection) ->
    collection.each (item) ->
      new ItemView({
        model: item
      }).render()
}
```

Daten speichern

```
<form method="post">
    <input type="text" name="title" placeholder="Titel" required>
    <select name="priority">
        <option value="0">Niedrig</option>
        <option value="1">Normal</option>
        <option value="2">Dringend</option>
    </select>
    <input type="submit">
</form>
```

Daten speichern

```
# Formular-Absenden abfangen
$('form').submit (evt) ->
  evt.preventDefault() # Absenden stoppen
  newItem = new Item() # Neues Model bauen...
  newItem.set 'title', $('input[name="title"]').val()
  newItem.set 'priority', $('select[name="priority"]').val()
  Items.add newItem    # ... der Collection hinzufügen...
  newItem.save()       # ... und speichern!
```

(Stellt euch hier einfach eine Demo des Gezeigten vor)

Gute Sachen

- Fertige Bausteine (Models, Views, Collections) mit nützlichen Defaults
- Viel nützliches Zubehör (Underscore, Events, Templating)
- Konventionen und gleichzeitig viele Freiheitenn

Offene Fragen

- Wie strukturieren wir übergeordnete Konzepte wie Module?
- Wie wird ein „Item“ repräsent? Seite? Widget?
- Wie managen wir Permissions und Authentifizierung?
- **Backbone gibt uns keine Antworten hierauf!**

Backbone in der App

```
# Spezieller AMD-Build von [amdjs](https://github.com/amdjs)
require ['backbone'], (Backbone) ->
  # Code hier
  # jQuery (`$`) und Underscore (`_`) werden von Backbone
  # als Abhängigkeiten geladen
```

```
// Pfade im Buildprozess
requirejs: {
  baseUrl: './src/script',  // Script-Source-Verzeichnis
  paths: {                  // Pfade für Libraries mit doofen Dateinamen
    'jquery': 'lib/vendor/jquery-1.7.2',      // Versionsnummer
    'backbone': 'lib/vendor/backbone-amd',    // AMD-Build
    'underscore': 'lib/vendor/underscore-amd' // AMD-Build
  },
  name: 'main',             // Wurzelmodul
  out: 'app.js'             // Zieldatei
  //optimize: 'none'        // Optimierung deaktivieren
},
```

Alles klar zu Stylus und Nib?

Next:

HTML5 & CSS3

HTML5-Technologien

- Web Sockets / Echtzeitkommunikation
- HTML5-Formulare + automatische Validierung
- Drag & Drop + Bildupload/Bearbeitung
- Offline-Technologien
- Geolocation / Navigation

CSS(3)-Technologien

- Präprozessor-Features
- Neue Selektoren
- Responsive bzw. Mobile Design mit Media Queries
- Animationen, Transitionen, Transformationen
- Print-Stylesheets

Viel Zeugs …

Beispielapp!

Beispiel-App

- `git clone https://github.com/SirPepe/HTML5Workshop2.git`
- `npm install`
- `grunt`
- Minimales Beispiel
- Fertiger Build-Prozess
- Gaaaanz viele Kommentare
- Alles bereit zum **kopieren und anpassen!**

Beispielapp im Detail

</walkthrough>

Alles klar zur Beispielapp?

Vorschläge / Herausforderungen

- Templatesystem austauschen; Underscore.Template stinkt, Handlebars [35] ist besser
- CoffeeScript-Dialekte anschauen (v.A. IcedCoffeeScript [36])
- Automatischer Build beim Speichern und Live-Reload nach Build (es gibt fertige [37] Tasks [38] dafür)
- Automatisches Multi-Geräte-Testing mit Bunyip [39] oder Distributed Continuous Integration mit TestSwarm [40]
- Weitere Ideen im Laufe der kommenden Tage

[35] http://handlebarsjs.com/
[36] http://maxtaco.github.com/coffee-script/
[37] https://github.com/cowboy/grunt/blob/master/tasks/watch.js
[38] https://github.com/webxl/grunt-reload
[39] http://ryanseddon.github.com/bunyip/
[40] https://github.com/jquery/testswarm/wiki

</reden>

<taten>