2018

Zusammenfassung WEB3

SEP 2018

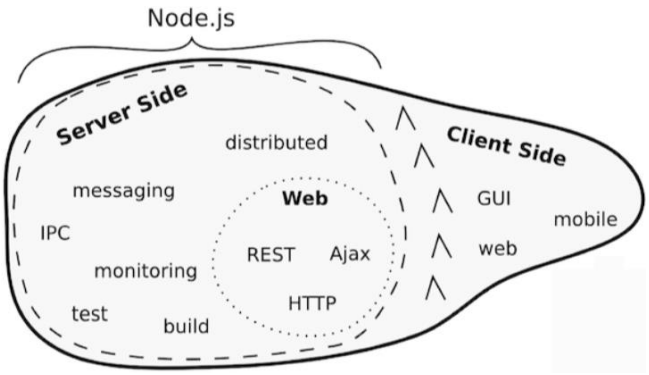
PASCAL BRUNNER

1 Inhaltsverzeichnis

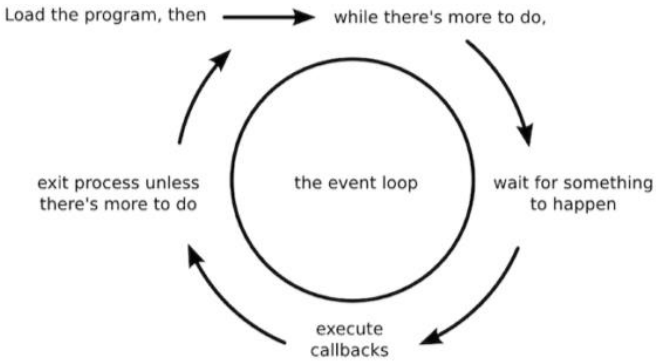
Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

2 NodeJS

- Was ist Node.js?
  - Es ist JavaScript, jedoch im Backend
  - Node.js basiert auf einer idiomatischen, asynchronen, ereignisgesteuerten Architektur



- Was ist npm?
  - Node Packaged Modules mit über 700'000 Moduls
  - Ist die Standardlösung für die Installation für Node
- Was ist das Prinzip von Node.js?
  - Event Loop → Kernfunktionalität von Node
    - Solange noch etwas zu tun ist, läuft der Loop weiter
    - Wenn Event eintritt, hört es auf den Callback
  - Operationen können schnell eingeplant werden, wenn neue Ereignisse auftauchen
  - Alles läuft parallel ausser der Code selber
  - Single-Threaded und Hochparellel



const → setzt eine Variable mit einem konstanten Wert

require() → function pulls in a Node module and returns it → Output plain old JS object

watch() → polls target file for changes

- Event bus
  - Mit Event-Bus kann man Events durchreichen und auf diese dann an unterschiedlichen Orten abhören

Um potentiellen Konflikten zwischen verschiedenen Programmteilen aus dem Weg zu gehen, hat node.js Module, welche nach dem Module Design Pattern funktionieren (ursprünglich angedacht für die Enkapsulation von private und public)

Modules

- Ein Modul ist ein .js-File
- Wenn Node ein neues File lädt, so kreiiert es einen neuen Scope
- Vom inneren des Plugins ist die Aussenwelt unsichtbar, ausser für benötigte andere Module

module.exports → kreiiert ein Modul-Objekt

exports → shortway für module.exports

Webserver

Node.js hat ein "http-Modul» welches für das Erstellen eines WebServers verwendet werden kann.

```
1. const http = require('http');
2. const server = http.createServer(function(req, res) {
3.                   res.writeHead(200, {'Content-Type': 'text/plain'});
4.                   res.end('Hello World\n');
5.                 });
6.
7. server.listen(8080, function(){
8.   console.log('ready captain!');
9. });
```

➔ Dieser Server kann nur auf request mit hello World antworten

http request Lifecycle

1. ein http-Client (z.B: Webbrowser) leitet einen http Request ein

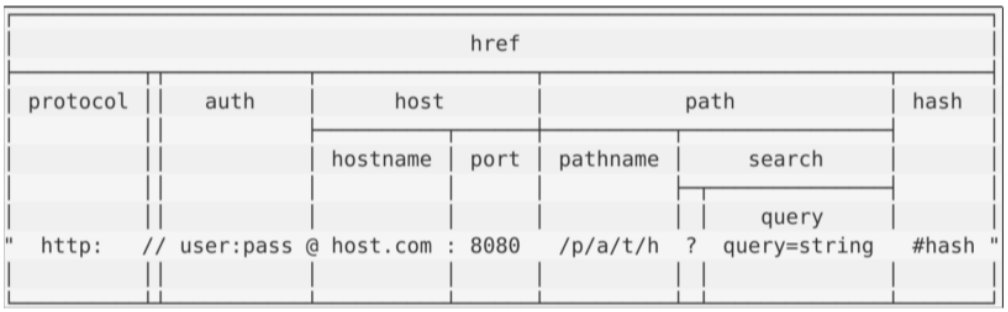
2. Node akzeptiert die Verbindung und die einkommende Daten werden an den http Server übermittelt

3. http Server parsed und führt den request callback aus

4. Der Request Callback wird mit der App-Logik ausgeführt

5. Die Antwort wird zurück via den http Server gesendet

Mit url.parse kann man die versch. URL Komponenten extrahieren



Mit cwd() kann man den current working directory abfragen um den root des Pfades zu finden

Verbesserter, aber simpler static web server

```
1.  const http = require('http'),
2.        fs = require('fs'),
3.        url = require('url'),
4.        path = require('path');
5.
6.  http.createServer(function(request, response) {
7.    var uri = url.parse(request.url).pathname;
8.    var filename = path.join(process.cwd(), uri);
9.    console.log("serving: " + filename);
10.   fs.readFile(filename, 'binary', function(err, file) {
11.     response.writeHead(200);
12.     response.write(file, "binary");
13.     response.end();
14.   })
15. }).listen(8080)
```

➔ Noch nicht wirklich robust

Jedes npm-package hat ein package.json File welches das Package beschreibt und die Dependencies angegeben sind.

```
1.  {
2.    "name": "foo",
3.    "version": "1.0.0",
4.    "description": "A foo package",
5.    "main": "index.js",
6.    "author": "demo@zhaw.ch",
7.    "license": "GPL"
8.  }
```

Mit --save bei npm install, update das package.json file automatisch mit den korrekt Dependencies

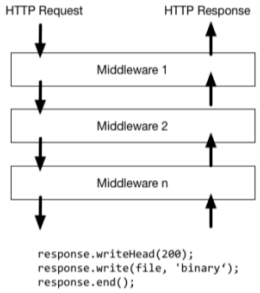
Mit npm start, kann man angeben, was gestartet werden soll

```
1.  {
2.    "name" : "simple-web-server",
3.    "scripts" : {
4.      "start" : "node web_server.js"
5.    }
6.  }
```

Express

- Ist ein kleines, schneller und erweiterbares Framework für das Erstellen eines Web- und Applikationsservers
- Npm install --save express

Die Express-Funktionalität wird durch die sogenannte Middleware bereitgestellt. Es können mehrere Middlewares (Bspw. Webserver, Authentication, Caching, Logging etc.) hinteinander geschalten werden. Diese asynchronen Funktionen manipulieren die request und response Objekte.



- get(), put(), post(), del()
- app.get() sagt Express wie ein http-Get Request zum /api/:name -Pfad behandelt werden soll
- :name → named route parameter
- Wenn die API getroffen wird, wird dieser Teil der URL in req.params verfügbar
- Das Programm hört auf den TCP Port 8080 für einkommende http Requests

Um nun einen vollfunktionsfähigen WebServer zu erhalten:

app.use(express.static('public')) → erstellt statische Middleware und speichert Files im Public

Der Webserver kann man danach mit npm starten

```
1. "scripts": {
2.   "start": "node ./express_server.js"
3. }
```

➔ Wenn man Änderungen am Code vornimmt, muss man den Server neustarten

Mit nodemon (kurz für Node Monitor) kann man diesen Prozess automatisieren -> Startet automatisch neu, wenn es Änderungen am Code gab.

➔ Npm install -g nodemon
➔ Nodemon express_server.js

3 SPA (single-page Web Application)

1. User navigates to new view

2. Additional content is requested using CHR (often via REST)

3. AJAX makes communication and processing asynchronous

4. User has improved usability and responsiveness without interruption

MVC Design Pattern

Ein sehr beliebtes Design-Pattern ist MVC (Model-View-Controller) → Es verbessert die Anwendungsorganisation durch die Trennung.

Business Daten (**Model**) von User Interface (**Views**) und verwalten der Logik aufgrund von User Inputs und Koordination zwischen Model und View (**Controllers**)

1. Models

    - repräsentiert einen spezifizierten Bereich von Daten

    - Vergleichbar mit einer Klasse: User, Photo, Todo, Note

    - Models können Observers benachrichtigen, wenn deren Status geändert hat

2. Views

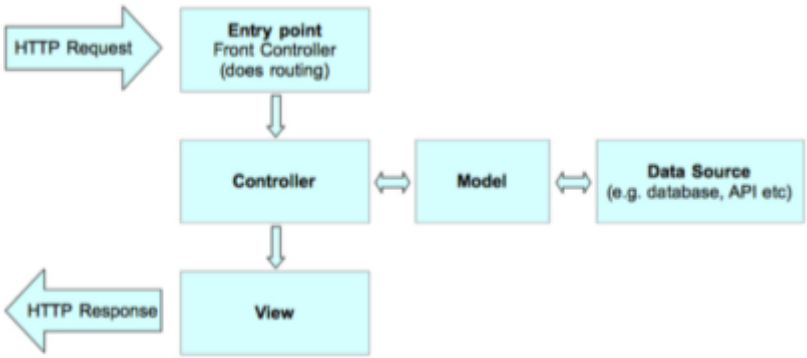
    - bilden in der Regel das Interface / Oberfläche (markup / templates)

    - beobachten Models, aber kommunizieren nicht direkt mit ihnen

3. Controllers

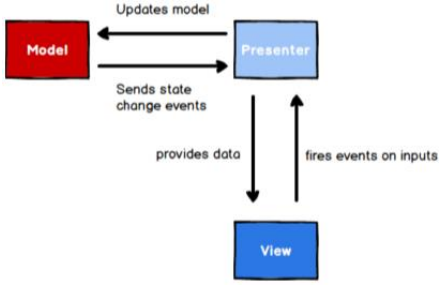
    - Behandeln den Input (klicken, Benutzer Interaktionen) und updaten die Models

→ Benutzereingaben werden von Controllern bearbeitet, die Modelle aktualisieren. Views beobachten Modelle und aktualisieren die Benutzeroberfläche, wenn Änderungen auftreten.



MVP Design Pattern

- MVP (Model View Presenter) ist eine Ableitung von MVC
- Wird vor allem für das bilden von Interfaces verwendet
- Der Presenter übernimmt die Funktion des Mittelmanns



- Das **Model** ist eine Schnittstelle, die die Daten definiert, die in der Benutzeroberfläche angezeigt oder anderweitig bearbeitet werden sollen.
- Der **Presenter** wirkt auf das Model und die View. Es ruft Daten aus dem Model ab und bereitet sie für die Anzeige in der View auf.
- Die **View** ist eine Schnittstelle, die Daten (das Model) anzeigt und Benutzerbefehle (Ereignisse) an den Presenter weiterleitet, um auf diese Daten zu reagieren.

Es gibt sehr viele Frameworks, was jedoch die meisten gemeinsam haben ist : Components

- Was ist ein Component?
    - Eine einzelne Softwarekomponente ist ein Softwarepaket, ein Webservice, eine Webressource oder ein Modul, das eine Reihe von verwandten Funktionen (oder Daten) kapselt.
    - .... wird oft gesagt, dass Komponenten modular und zusammenhängend sind.

Web Components beinhalten im Wesentliche folgende 4 Standards

- Custom Elements
    - Gibt Entwicklern die Möglichkeit ein eigenes DOM-Element zu kreiieren
- Templates
    - Deklaration von Elementen die beim Laden noch nicht verwendet werden, aber in HTML geparsed und zur Laufzeit der Website zur Verfügung stehen
    - Aktuell einziger finalisierte Standard
- HTML Imports
    - Erlaubt es innerhalb von HTML Dokumente andere HTML Dokumente zu importieren
    - Ermöglicht das Teilen von HTML, CSS und JS Code ohne AJAX
- Shadow DOM
    - Ermöglicht das Einbinden von unterschiedlichen Ressourcen bspw. Youtube, FB etc. mit <iframe>

4 RIOT

- Custom Tags möglich
- Einfache API

Custom Tags in Riot werden *mounted* in der HTML-Seite

```
1.  <!doctype html>
2.  <html>
3.    <head>
4.      <title>Hello Riot</title>
5.      <meta charset="utf-8" />
6.    </head>
7.    <body>
8.
9.      <sample></sample>
10.
11.     <script type="riot/tag" src="sample.tag.html"></script>
12.     <!-- include riot.js -->
13.     <script src="js/riot+compiler.min.js"></script>
14.     <!-- mount the tag -->
15.     <script>riot.mount('sample')</script>
16.   </body>
17. </html>
```

Riot lädt einen sample.tag.html und fügt den <sample> tag im body der Seite ein.

```
<sample>
  <h1> { message } </h1>
  <script>
     this.message = "Hello World.";
  </script>
</sample>
```

Die Leerzeichen zwischen {} sind notwendig!

```
<clock>
  <div> { time } </div>
  <script>
   var tag = this;

   tag.tick = function() {
     tag.update({ time: new Date() });
   }

   var timer = setInterval(tag.tick, 1000);
  </script>
</clock>
```

```
<body>
  <clock></clock>
  <clock></clock>
  <script type="riot/tag" src="clock_1.tag.html"></script>
  <!-- include riot.js -->
  <script src="js/riot+compiler.min.js"></script>
  <!-- mount the tag -->
  <script>riot.mount('clock')</script>
</body>
```

Die verschiedenen Tags können auf unterschiedliche Art und Weise gemounted werden:

```
// mount all tags
riot.mount("*");
// mount element with id
riot.mount("#some-element");
// mount specific tags
riot.mount("clock, tz-selector, comments")
```

Tag Lifecycle

1. Tag wurde erstellt

2. Das Javascript des Tags wurde ausgeführt

3. Der HTML Ausdruck wurde berechnet

4. Der Tag wurde gemounted und der mount-Event wurde durchgeführt

Wenn der Tag gemounted wurde, gibt es folgende Update-Ausdrücke:

1. Automatischer Update durch Event-Handler

2. Wenn this.update aufgerufen auf dem aktuellen Tag aufgerufen wurde

3. Wenn this.update auf einem Parent-Tag aufgerufen wurde

4. Wenn riot.update aufgerufen wurde → updated alle Ausdrucke auf dieser Seite

```
this.on('before-mount', function() {
  // before the tag is mounted
})
this.on('mount', function() {
  // right after the tag is mounted on the page
})
this.on('update', function() {
  // allows recalculation of context data before the update
})
this.on('updated', function() {
  // right after the tag template is updated after an update call
})
this.on('before-unmount', function() {
  // before the tag is removed
})
this.on('unmount', function() {
  // when the tag is removed from the page
})
// curious about all events ?
this.on('*', function(eventName) {
  console.info(eventName)
})
```

Mixin

Mit Mixin kann Code über mehrere Tags geteilt werden.

Tbd

Expressions

Man kann sein HTML mit JS Ausdrücke ergänzen

```
{ title || 'Untitled' }
{ results ? 'ready' : 'loading' }
{ new Date() }
{ message.length > 140 && 'Message is too long' }
{ Math.round(rating) }
```

Es ist zu empfehlen, dass man seine Expressions im Tag selber kurz hält und die Berechnungen eher separat durchführt

```
<my-tag>

  <!-- the `val` is calculated below .. -->
  <p> { val } </p>

  // ..on every update
  this.on('update', function() {
    this.val = some / complex * expression ^ here;
  })
</my-tag>
```

Loops

Mit each, kann man einen Loop auslösen. Bspw. um sämtliche Tasks zu laden

```
<todo>
  <ul>
    <li each="{ items }" class="{ completed: done }">
      <input type="checkbox" checked="{ done }"> { title } </input>
    </li>
  </ul>

  this.items = [
    { title: 'First item', done: true },
    { title: 'Second item' },
    { title: 'Third item' }
  ]
</todo>
```

Ref

Elemente mit dem ref-Attribut werden automatisch zu diesem Kontext gebunden, weshalb man einen einfachen Kontext dazu aufbauen kann.

```
<login>
 <form id="login" onsubmit={ submit }>
   <input ref="username">
   <input ref="password">
   <button ref="submit">
 </form>

 // access above HTML elements
 this.submit = function() {
   var form = this.login,
       username = this.refs.username.value,
       password = this.refs.password.value,
       button = this.refs.submit
 };
</login>
```

EventHandlers

- Beginnen jeweils mit on (onclick, onsubmit etc.)
- This referenziert zur aktuellen Tag-Instanz
- Nach dem der Handler aufgerufen wird, wird automatisch this.update() aufgerufen und das UI wird angepasst.

```
<login>
  <form onsubmit={ submit }>
  </form>

  var tag = this;
  // this method is called when above form is submitted
  tag.submit = function(e) {
    e.preventDefault();
  }
</login>
```

- E.preventDefault(); stellt sicher, dass die Seite nicht geladen wird

Conditionals

In den Tags kann man auch eigene Bedingungen einbauen

```
<div if={ is_premium }>
  <p>This is for premium users only</p>
</div>
```

Nesting von Objekten

Grundsätzlich ist das Nesting von Objekten möglich, jedoch nicht empfehlenswert, da es eine starke Abhängigkeit zwischen den beiden Komponenten gibt

Observable

```
function Car() {
  // Make Car instances observable
  riot.observable(this)

  // listen to 'start' event
  this.on('start', function() {
    // engine started
  })
}

// make a new Car instance
var car = new Car()

// trigger 'start' event
car.trigger('start')
```

EventBus

Mit einem EventBus kann man den Inhalt bzw. den Event einer Komponente an eine andere durchreichen.

EventBus erstellen und diesen an die TodoListKomponente durchreichen

```
1. var bus = riot.observable();
2. riot.mount('todolist', { bus : bus } );
```

ToDoCollection umschreiben, so dass dieser vom Bus weiss

```
 1. class TodoCollection {
 2.   constructor(bus) {
 3.     // ...
 4.     this.bus = bus;
 5.   }
 6.   // ...
 7.   // Adds a model to the collection and persists it
 8.   add(model) {
 9.     this.collection.push(model);
10.     this.save();
11.     this.bus.trigger('collectionUpdated');
12.   }
13.   // Fetch models from localStorage into collection
14.   fetch() {
15.     this.collection =
    JSON.parse(localStorage.getItem(this.localStorage_key)) || [];
16.     this.bus.trigger('collectionUpdated');
17.   }
```

TodoComponent umschreiben, damit dieser auf Events reagieren kann

```
 1. var tag = this;
 2.
 3. this.bus = opts.bus;
 4.
 5. this.todos = new TodoCollection(bus);
 6. // return all todos
 7. this.allTodos = function() {
 8.   return this.todos.all();
 9. }
10.
11. this.bus.on('collectionUpdated', function() {
12.   tag.update();
13. });
```

RiotJS Router

- erlaubt es uns den Fragementteil der URL anzupassen
- man wird informiert, wenn ein Fragmentteil ändert
- den aktuell URL-Fragmentteil zu untersuchen

```
1. riot.route(function(collection, id, action) {
2.  // this is the callback
3. });
```

Wenn die Seite z.B. mit http://localhost:8000#project/123/edit aufgerufen wird, zieht der Callback mit den folgenden Parameter collection: project, id: 123, action:edit

Man muss den Router explizit mit *riot.route.start(true)* starten bzw. mit *riot.route.stop()* anhalten

RiotGear

Hat diverse Komponenten welche für RiotJS verwendet werden können, ohne dass man diese neu erstellen muss → Library

5 Test Driven Development (TDD)

1. Lege das nächste Arbeitspaket fest
2. Schreibe Testfälle dazu, welche den Code spezifizieren
3. Schreibe den eigentliche Code um die Testfälle zu bestehen
4. Wenn alle Testfälle erfolgreich sind, gehe zurück und Refactore den Code

```
1.  describe("Persistence", function() {
2.    it("In the beginning, there is emptiness", function() {
3.      expect(todo_collection.all()).toEqual([]);
4.    });
5.    it("When a Model is added to the Collection, we can find it",
    function() {
6.      var todo = { name: "Test Model", done: false };
7.      todo_collection.add(todo);
8.      expect(todo_collection.all()[0]).toEqual(todo);
9.    });
10.   it("Models within a Collection are persisted in localStorage",
    function() {
11.     var todo = { name: "Test Model", done: false };
12.     todo_collection.add(todo);
13.     // Manually delete what's currently in memory
14.     todo_collection.collection = [];
15.     expect(todo_collection.collection).toEqual([]);
16.     // However, `fetch()` should query localStorage and bring the todo
    back
17.     todo_collection.fetch();
```

Jasmine Framework mit npm installieren

npm install jasmine -g //installieren

jasmine init //spec Ordner erstellen

cd example_code

nodemon todo_server.js //Nodemon to reasrt

nodemon -x jasmine // Running tests on any file change

```
var request = require("request");
describe("API", function() {
  it("returns an empty result on GET /todos", function(done) {
    request("http://localhost:8000/todos", function(error, res, body) {
      expect(JSON.parse(body)).toEqual({todos: []});
      done();
    });
  });
  it("saves and returns the todo on POST /todos", function(done) {
    request({
      method: "POST",
      url: "http://localhost:8000/todos",
      json: {todo: "buy milk"}
    }, function(error, res, body) {
      expect(body).toEqual({todo: 'buy milk'});
      done();
    });
  });
});
```

- Jasmine unterstützt auch Asynchronität
  - beforeAll
  - afterAll
  - beforeEach
  - afterEach
  - it
  - done

6 Verwaltung von Dependencies

Grössere Web-Projekte können sehr schnell, sehr viele Abhängigkeiten haben. Damit man diese nicht alle manuell verwalten mus gibt es Stacks wie bspw. Yarn oder Webpack, welche die Aufgabe für einen erledigen.

- Yarn
    - Alternative zu npm
- Webpack
    - Bundling
    - Cross-compilation

7 Sessions

- http ist stateless → jede Anfrage an den Server ist unabhängig der anderer
- Wir wollen den User jedoch eingelogged halten bspw. beim Webshop → Session
- Session sind Daten welche zu einem bestimmten User gehören

Cookies

- Kann bis zu 4kb Datenspeichern
- Bei jeder Authentifizierungsanfrage sendet der Client das Cookie via http-Headers
    - Der Server liest das Cookie, genehmigt die Session-Data und gibt Antwort
- Where to store this session data is up to you
    - By default. Express will keep the data in memory, but this does not readily scale
    - Once you have more than one Node process, the session data should be stored in a shared place

```
1.  const
2.      express = require('express'),
3.      //...
4.      session = require('express-session'),
5.      app = express();
6.
7.  app.use(session({
8.      secret: 'impossible to predict'
9.      // NB: Do not hard-code secrets or check them in
10. }));
```

➔ Die Session kann wie ein normales JS Objekt verwendet werden

```
1.  app.get('/hitcount', function(req, res) {
2.      var session = req.session;
3.      if (session.hitcount) {
4.          session.hitcount++;
5.      } else {
6.          session.hitcount = 1;
7.      }
8.
9.      res.status(200).json({ 'hits_in_session' : session.hitcount })
10. });
```

Benutzer Authentifikation kann nun mittels Modulen bspw. Passport

8 Style Guide

- Das DOM wird nicht mehr im klassichen Sinne manipuliert, sondern RIOT erstellt ein virtuelles DOM → Updating

9 Deployment

A solid hosting setup includes (1/2)

- Monitoring & Notifications
- Watchdog
- Automated Backups
- Automated Upgrades
- Security
  - *Firewall*
  - *Automated Updates to Operating System*
  - *Intrusion detection*
  - *Secure Login*
- Automated deployment of code and database schema with the ability to roll back to any version

A solid hosting setup includes (2/2)

- Configuration management for the server(s) itself
- Ability to provision and scale resources(HDD, CPU, RAM)
- Ability to provision and scale server/db instances including balancing between them
- Ability to address hardware issues quickly
- Failover when one host goes down with zero to little downtime / effort
- Great are: automated tests concluding in continuous deployment

Die verschiedenen Hosting-Möglichkeiten

- Eigene Hardware besitzen, schnelle Internetverbindung und alles im eigenen Keller verstauen
- Dezidierte Hardware, Unterhaltskosten und Schutz im Datencenter
- VM in der Cloud inklusive Skalierung, Backup und Monitoring
- Infrastructure as a Service (IaaS): Verschiedene Services für verschiedene Kosten wie Speicher, Datenbank etc.

Möglichkeiten für Back-End Deployment

- Amazon Elastic Beanstalk
- Google App Engine
- EngineYard
- Heroku

Container

- Es gibt die Möglichkeit seine Packages in der App inkl. Anforderungen in einen Container zu packen.
- Dies wird oftmals in Verbindung mit einem Docker gebracht

10 Web Security

- 96% aller Applikation haben Schwachstellen
    - Der Median liegt pro App bei 14
- Die Sicherheit ist schlussendlich abhängig von den Menschen, welche das Framework gebrauchen
- Statt nur eine sichere WebApp zu entwickeln, sollte man auf sämtlichen Ebenen Up to Date sein
    - Security mailing list abonnieren
    - Security blogs lesen
    - Update und security checks regelmässig durchführen
        - Gewissen können bspw. mit hakiri.io automatisiert werden

Session Management

- Cookies sollten einen Hash für die Werte und ID verwenden
- Jeder der in den Besitzt eines Cookies einen anderen kommt, kann sich als diesen identifizieren → Session Hijacking
- Viele Cross-Site Scripting (XSS) gehen auf die Cookies los

Session Storage

- Der Client kann alles in der Session sehen, denn dieser ist im Klartext abgelegt
- Es sollen keine «Geheimnisse» in einem Cookie gespeichert werden
    - Nur die ID wird abgelegt und zwar mit einem Hashwert
- Keine grosse Objekte in der Session speicher, stattdessen direkt auf der DB des Servers
- Wann immer möglich, auf Sessions verzichten
    - Stattdessen sollen JWT (JSON Web Tokens) verwendet werden

Cross-Site Request Forgery (CSRF)

- Der Angriff wird durch schädlichen Code oder Link auf einer Website durchgeführt
- Der Benutzer glaubt, dass er sich entsprechend anmelden muss
- Der Browser sendet bei jeder Anfrage an eine Domain automatisch das Cookie mit, wenn es sich um eine Domain handelt. Hat ein Cookie für diese Domain.
- Der umstrittene Punkt ist, dass es auch das Cookie sendet, wenn die Anfrage von einem andere Domäne

Cross-Site Scripting (XSS)

- Ist der häufigste Fall
- Bei diesem Angriff wird auf der Client-Side einen Code ausgeführt
- Bspw. Formularfeld. Die WebApp speichert dies und zeigt das Resultat an
- XSS kann das Cookies stehlen, Session Hijacken, zu einer Fakeseite weiterleiten, falsche Dinge anzeigen, Dinge auf der Website verändern oder eine Malware installieren

Here is a straightforward way to check for XSS

```
<script>alert('Hello');</script>
```

This does exactly the same, only in very uncommon places

```
<img src=javascript:alert('Hello')>
<table background="javascript:alert('Hello')">
```

➔ Nicht selberversuchen dies zu schützen, sondern auf bewährte APIs zugreifen, denn es ist schwer sämtliche Dinge abzufangen. Wie folgendes Bild zeigt

Example UTF-8 attack vector (this is still a simple obfuscation)

```
<img
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#
108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>
```

- The browser will render this, your manual escaping might not catch it

SQL Injection

- Durch einen User Input wird eine Datenbank abfrage getätigt