Spaß mit JavaScript

Servus!

Ich bin Matthias.

CEO Frischluft Medien OG

Lehrender FH-Hagenberg

Trainer Coders.Bay Linz

Organisatorisches
Modus und Beurteilung

Weil's sein muss

Anwesenheit & Durchführung

- (Hoffentlich 🙏) Vor Ort alternativ über MS-Teams
- Vier Blöcke - Übungsteil muss ggf. zuhause fertiggestellt werden.
- Tutor (Daniel Saiz) kann unterstützen

Inhalt & Erwartungen

- Wir behandeln mehrere fortgeschrittene Konzepte in **kurzer Zeit**
- Schwierigkeitsgrad **hoch**
- **Eigeninitiative** wird notwendig sein
- => Wir lernen richtiges **Experten-Zeug**!

Ablauf und Benotung

- In vier Blöcken erstellen wir **unser eigenes JS-Framework**
- Die Übungen **bauen** jeweils **aufeinander auf**
- **Musterlösung** zum vorherigen Block wird bereitgestellt (Github)
- Übungen werden von **euch selbst** bewertet (zählt nicht zur Note)
- Klausur am Ende des Kurses

Organisatorisches
GitHub

Keine Angst, wir klonen nur

How to git clone

1. Install git (if not installed yet)
   a. Windows: Click this link
   b. macOS: `$ git --version`
   c. Linux: `$ sudo apt install git-all`
2. Create new (empty) folder
3. `$ git clone https://github.com/MatthiasNeuwersch/FH-Hagenberg.CWP4`

Let’s have a look! 

JavaScript Rückblick

Was können wir schon?

JavaScript Überblick

Warum nicht Pascal?

The need of JavaScript

1996	2006	2016
JS erscheint. Ein paar Leute surfen im Netz. Hauptsächlich lesen sie statische Inhalte. JS kann auch was	Internet ist relativ weit verbreitet, viele PCs surfen im Web. JS ist dienlich für z.B. Form-Validation oder Eventhandler	Jeder hat Internet in der Hosentasche. Alles passiert online (Banking, Filme, Musik, Telefonieren, Spiele...) -> häufig im Browser

Proof?

1996	2022
Nintendo N64 erscheint	Browser: Drakensang Online
Regionales Kino mit 2 Säälen	Browser: Netflix / Prime
Radio mit Antenne und Batterie	Browser: Spotify
Röhrenfernseher im Esszimmer	Browser: YouTube
Sumsi Spardose / Sparbuch	Browser: ELBA / George

Proof?

1996	2022
Microsoft Office Suite	Office 365 / Google Suite
Wechselrahmen	Google Drive / One Drive
Morpheus / Limewire /WinAmp	Spotify
Microsoft Encarta	Wikipedia
Babylon	Google Translate

JavaScript in 2022

Rich Software

Anspruchsvolle Software wird heute auch in JavaScript programmiert.

All Sorts of Devices

Egal welches Betriebssystem, egal ob SmartPhone, Tablet oder PC.

It's EVERYWHERE!

Alles gibt's heut auch im Browser!

JavaScript
Warum Frameworks?

Wieso, weshalb, warum?

JavaScript Frameworks
Walk of ~~Sh~~Fame

Die Crème de la Crème

Angular.JS

Großer Name

Google treibt die Entwicklung von Angular.js voran.

Besonderheit

Arbeitet mit TypeScript, ist vermutlich das etablierteste Framework

Downside

Angular zählt als Schwergewicht

Vue.js

Großer Name

Evan You - nachdem er bei **Google** aufgehört hat - entwickelte Vue.js

Besonderheit

Simplicity.

Downside

Nicht für ganz große Software geeignet, hat schwierigkeiten beim Skalieren.

React.JS

Großer Name

react.js wurde von FaceBook entwickelt. Auch Uber und Airbnb verwenden react.

Besonderheit

Behandelt nur die View aus MVC. Muss also um andere Komponenten erweitert werden

Downside

Unidirectional Databinding. Verwendet JSX.

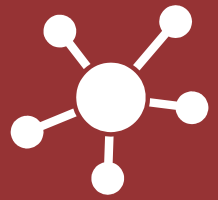
kwm.JS

Großer Name	Besonderheit	Downside
Du!	Regional Echt Rarität	Noch nicht fertig 🤦

JavaScript Frameworks

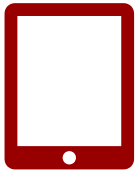
Konzepte

Was macht mich zum Framework?



Konzepte
Single Page Applications

Überblick / Technolekt



SPA - Was ist das?

Index.html

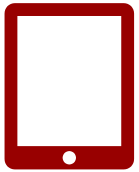
Applikation läuft in **einer einzigen** .html Datei.

DOM Manipulation

Erfolgt ausschließlich durch JavaScript; wird ggf. **über API nachgeladen.**

Navigation

Mittels Router: Entweder durch **History API** oder durch **Hash Based Routing.**



SPA - Vorteile?

User Experience

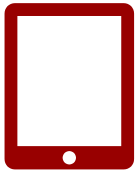
Während Bedienung **kein sichtbarer Pagereload** oder Refresh.

Performance

Es **muss nicht** bei jedem Request das gesamte Dokument **neu geladen werden.**

Datascope

Das **Datenmodell** besteht view-übergreifend über die **gesamte Session** hinweg.



SPA - Nachteile?

SEO

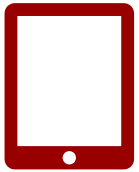
Nur ein **Teil** des Contents ist **für Suchmaschinen sichtbar**.

Preloading

Je nach Anwendung müssen bereits zum Einstieg **viele Daten vorgeladen** werden.

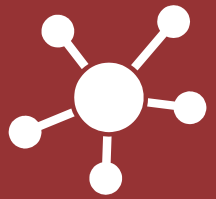
Aufwand

In der Erstellung **aufwändiger**, als statische HTML-Seite



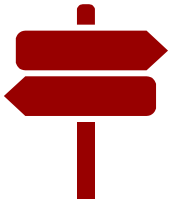
SPA - How to?

1. Einstieg auf index.html
2. Laden von Router in JS
3. URL-Änderungen werden vom Router abgefangen; Entsprechende Views werden angezeigt.
4. Content wird ggf. dynamisch über AJAX nachgeladen



Konzepte
Routing

Technolekt / Setup



Router - Was ist das?

Navigation in SPA

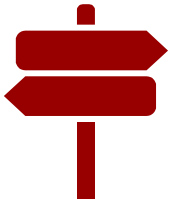
SPA hat ja eigentlich nur eine URL.
Router ermöglicht Navigation in SPA

Ist quasi der View-Manager

Sucht die **passende View** zur Anfrage raus und zeigt diese an.

Unzertrennlich

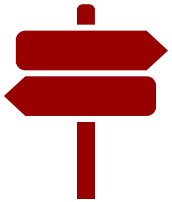
Routes und Router bilden gemeinsam das Konzept des "**Routing**"



Router in echt

- Hat ein **Array mit allen Routen**
- Hat zusätzlich **Spezialrouten** (Startseite, 404-Seite etc.)

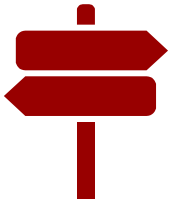
```
export default class Core_SPA_Router{
  constructor(views){
    this.routes = views;
    this.homeRoute = views[0];
    this.init();
  }
```



Router in echt

- Router wird mit **Listener** für das Event **hashchange** ausgestattet.
- Wenn sich **Hash** ändert, tritt der Router mit "**changeView()**" in Aktion.

```
init(){
  window.removeEventListener('hashchange', this.changeView);
  window.addEventListener('hashchange',this.changeView.bind(this));
  this.changeView();
}
```



Routes - Was ist das?

Zielpunkte

Routes sind **Zielpunkte** in unserer App, **fast so wie URLS**

History API

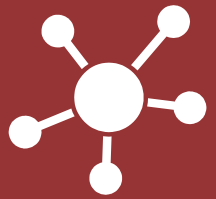
Kann über **window.history** angesteuert werden.

Hash Based

Kann über **window.location.hash** angesteuert werden. Liefert alles **hinter #** in der URL

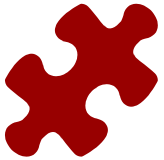
kwmJS Routes

- Wir verwenden **Hash-Based Routing**
- z.B. www.kwmjs.at/#**/login**
  - Router fängt "/login" ab und ruft die dazu passende View auf.
- Wie in URLs, können auch GET Parameter übergeben werden
  - z.B. www.kwmjs.at/#/login**?lang=en**



Konzepte
Translation Engine

Technolekt / Setup



Translator - Was ist das?

Dolmetscher

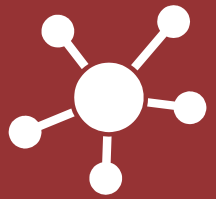
Liefert **Textausgaben** zu Suchindizes in gewünschter Sprache

Resources

Beinhalten die Übersetzungen. Können wir als **simples JavaScript Objekt** abbilden

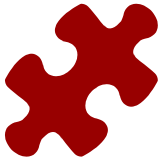
Key-Value Pairs

Key für alle Sprachen **gleich**, Value Sprachen-spezifisch



Konzepte
Template Engine

Technolekt / Setup



Templates - Was ist das?

Schablonen

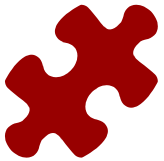
Templates sind **Markup-Schablonen,** die wir vorbereiten können.

Dynamise me

Die Schablonen können wir **dynamisch** mit Inhalten **befüllen**.

Format

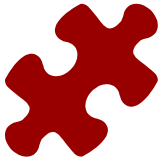
Frei wählbar, etwa als .txt oder .tpl



Templates how to?

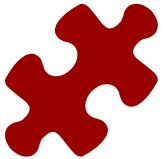
1. Vorbereitung von HTML-Markup
2. Maskieren von dynamischen Teilen durch Sonderzeichen
3. Template-Engine ersetzt maskierte Teile durch Variablen-Inhalt

```
<div id="hotel-list">
  <div class="hotel-container">
    <h1><&>hotel-name<&></h1>
    <span class="rating"><&>hotel-rating<&></span>
    <span class="price"><&>hotel-price<&></span>
    <span class="city"><&>city<&></span>
    <span class="country"><&>country<&></span>
  </div>
</div>
```



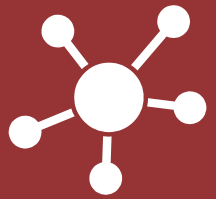
kwmJS
Templates

- Wir verwenden eine **vereinfachte** Form
- Maskieren durch **<&>**dyanmischer_inhalt**<&>**
- Maskieren durch **<%>**übersetzungen**<%>**
- <&>Maskierter Inhalt<&> wird **je nach Objekt** anders ausgefüllt
- <%>Maskierter Inhalt<%> wird **je nach Sprache anders** ausgefüllt
- Wir kombinieren **Template-Engine** mit **Translation-Engine**



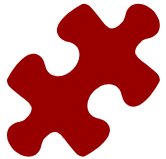
kwmJS Templates

```
<div id="login">
    <form id="login-form" class="received">
        <input id="login-username" required type="text" placeholder="<%>username<%>*"/>>
        <input id="login-password" required type="password" placeholder="<%>password<%>*"/>
        <input id="login-submit" type="submit" value="<%>login<%>"/>
    </form>
</div>
```



Konzepte
Databinding

Technolekt / Setup



Databinding - Was ist das?

Datenschicht (State)

Sind die Rohdaten, meist verwaltet vom Model

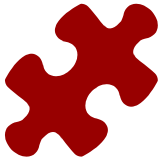
Userinterface (UI)

Ist zugleich Präsentation und Eingabecontroller (bspw. Formular) der Daten.

Binding

Verändert sich das eine, soll sich auch das andere verändern.

Uniderectional / Biderectional



Databinding - Wie geht das?

Eventdriven

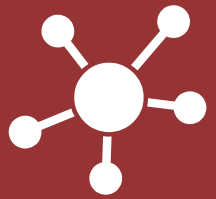
Werden Daten verändert, wird ein entsprechendes Event gefeuert. Dazupassend gibt es listener.

Pub/Sub Model

Wir arbeiten nach dem Prinzip des Publisher / Subscriber models, sodass Änderungen in der ganzen App sichtbar werden.

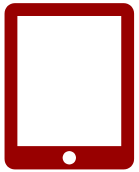
Bi-Directional

Verändern wir das Model, soll sich die View aktualisieren. Verändern wir Daten über die View, soll sich das Model aktualisieren.



Konzepte
MV(C)

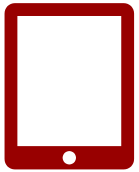
Technolekt / Setup



Model View Controller

- Trennung von Datenmodell und Präsentation
- Ermöglicht unterschiedliche Präsentationen eines Datenmodells
- Modularisiert die Applikation => Erhöhte Skalierbarkeit

Model



Daten

Alle Lese- und Schreibvorgänge für Daten laufen über das Model.

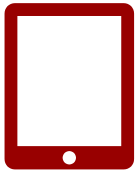
Unabhängig

Hat keine Informationen über den Controller oder die Views

Zentrale Komponente

Bildet den Angelpunkt des MVC-Patterns.

View



Präsentation

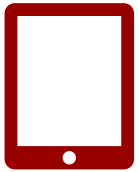
Übernimmt die konkrete Ausgabe der Daten, in welcher Form auch immer.

Unabhängig

Mehrere Views können voneinander unabhängig denselben Datenbestand repräsentieren.

Sichtbar

Ist jener Teil der Applikation, der für den Benutzer sichtbar ist.



Controller

Schnittstelle

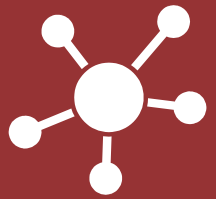
Übernimmt die Kommunikation mit Views und Model

Eingabefilter

Nimmt User-Input entgegen, behandelt diesen und kommuniziert dann an Model oder View

Verzichtbar

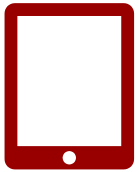
Der Controller wird manchmal auch weg gelassen.



Konzepte

Application Program Interface

Der Dreh- und Angelpunkt des Datenverkehrs



API

Was ist das?

Datenschnittstelle

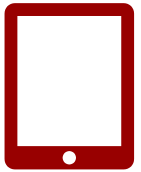
Dient dazu, **Daten** anzufragen und übermitteln zu können.

Daten á la carte

Ist wie eine **Speisekarte**. Man sieht, was es gibt und kann daraus **bestellen**. Die Zubereitung sieht man nicht.

Skalierbar

APIs sollten immer so ausgelegt sein, dass **mehrere** Clients **zugleich** damit arbeiten können.



APIs

Beispiele?

Business Software

z.B. Anbindung eines **Webshops** an ein **WAWI** um Lagerbestände zu synchronisieren

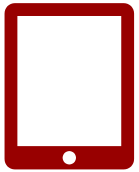
Öffentliche Daten

z.B. **Wetter-API** oder **Corona-API**. Services um anderen Daten zur Verfügung zu stellen.

Hardware APIs

z.B. **Lego Mindstorms** oder **Kaffeemaschinen**.

How about some **Restpresso**?



APIs
Nachteile?

Security

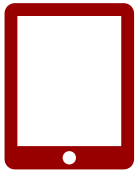
Man öffnet ein Tor und muss dieses entsprechend absichern.

Einsehbarkeit

Man gibt Preis, welche Daten man zur Verfügung hat und muss sich evtl. dafür rechtfertigen (z.B. Facebook)

Aufwand

APIs ain't building themselves



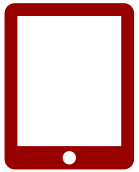
APIs Allgemein

- Aufruf meist **Asynchron**

  => Arbeiten mit **Callbacks** oder **Promises** (bevorzugt)

- Es gibt unterschiedliche **Standards** (z.B. **rest**) und **Protokolle** (z.B. **soap**)

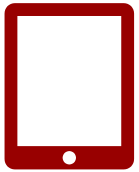
  => Entscheidend für das verwendete Datenformat (z.B. **json** oder **xml**)



APIs in fast echt

(Vereinfachtes Beispiel eines API-Calls)

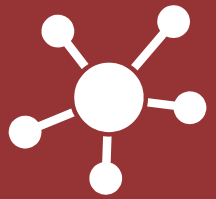
```
static getData(target){
   return new Promise((resolve, reject)  => {
       fetch(target)
           .then(response=>response.json())
           .then(data=>{
               console.log(data);
              resolve();
           });
   })
}
```



APIs in unecht

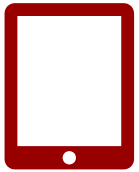
```
function resolveWebApplication(){
   initTemplate();
   doAllTheNeccessaryStuff();
   console.log("Hey, App is looking so fresh! Let's add some Data!");
   API.getData().then(function(data){
      includeDataToTemplate(data);
   });
   console.log("Oh Snap! This line is executed before the Data is included?");
}
```

API-Call kann unbestimmt lange benötigen. **.then** symbolisiert hier die **Asynchronität**.



Konzepte
Asynchronität - AJAX

Mehr als nur ein Putzmittel



AJAX

Was ist das?

Name ist Programm

Asynchronous **J**avaScript **a**nd **X**ML.

Zur Laufzeit

Ermöglicht **HTTP-Requests** mittels JavaScript → **Kein Neuladen** nötig.

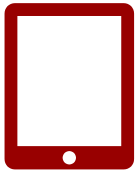
GET/**POST**/**PUT**/**DELETE** möglich

Schon 2005 nix Neues

Kombinierte Anwendung von **HTML**, **JS**, **DOM**, **XML**, **XSLT** und **XMLHttpRequest**

AJAX in hardmode

```
function oldSchoolXHR(){
   var xhttp = new XMLHttpRequest();
   xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
         var result = xhttp.responseText;
      }
   };
   xhttp.open("GET", "filename", true);
   xhttp.send();
}
```

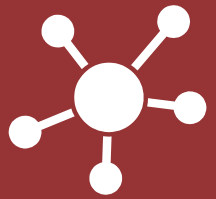


AJAX in jQuerymode

```
function jQueryAjaxRequest(){
   $.ajax({
      dataType: "json",
      url: url,
      data: data,
      success: function(result) {...}
   });
}
```

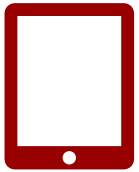
AJAX in easymode

```
fetch(target).then(response=>response.json())
      .then(data=>{  resolve(data);  });
```



Konzepte
Asynchronität - Promises

I **promise**, this will be complex!



Promise

Was ist das?

Friendly Placeholder

Für **Ergebnis** oder **Fehler** einer asynchronen Funktion, bis das Zutreffende dann soweit ist.

Wie alles in JS...

Sind auch Promises eigentlich nur **JavaScript Objekte**

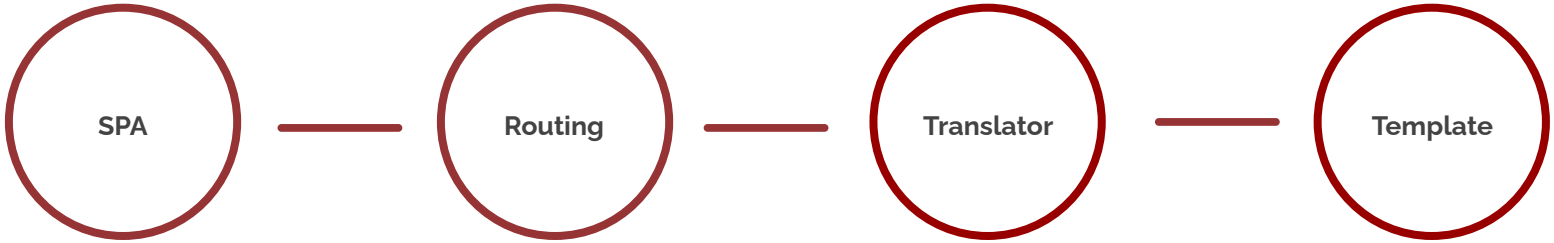
Status

Ein Promise ist immer entweder **Pending**, **Fullfilled** oder **Rejected**

Promise in echt

```
getCities(){
   return new Promise(resolve => {
       fetch(api_root+"posts")
           .then(response=>response.json())
           .then(data=>{
               let cities = [];
               for(let city of data)
                   cities.push(new City(city.acf));
               resolve(cities); });
   });
}
  async renderCities(){
       let cities = await window.Core.model.getCities();
       for(let city of cities)
           $("#cities_container").append(city.getListMarkup());
    }
```

Auf einen Blick



Single and ready to mingle

Die gesamte Applikation wird in einer einzigen Seite dargestellt. Teilbereiche der Applikation werden einzeln ausgetauscht.

Wo darf's hingehen?

Der Router dient zur Navigation zu unterschiedlichen "Seiten" in einer SPA, sowie zum Abfangen von ungültigen Seitenanfragen.

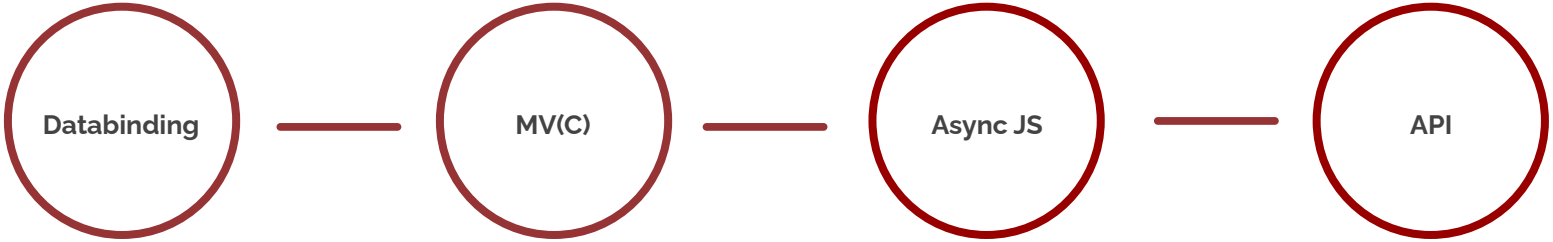
Der Polyglot

Die Translation Engine ermöglicht uns die Auslagerung von Ressourcen zur Übersetzung.

Der Gerät

Die Template Engine ermöglicht uns, statische Markup-Templates vorzubereiten, die dynamisch befüllt werden.

Auf einen Blick



Einfach unzertrennlich

Die Daten im Model und die sichtbaren Daten in den Views werden immer miteinander synchronisiert.

Most Valuable Concept

Das Model managed die Daten. Die View präsentiert sie. Der Controller regelt die Kommunikation zwischen den beiden.

I promise…

Jedes Promise liefert verlässlich ein **Ergebnis** oder einen **Fehler**. Das kann ich euch versprechen!

Die Datenspeisekarte

Regelt den Austausch von Daten über mehrere Systeme hinweg. Meist **REST** oder **SOAP**.