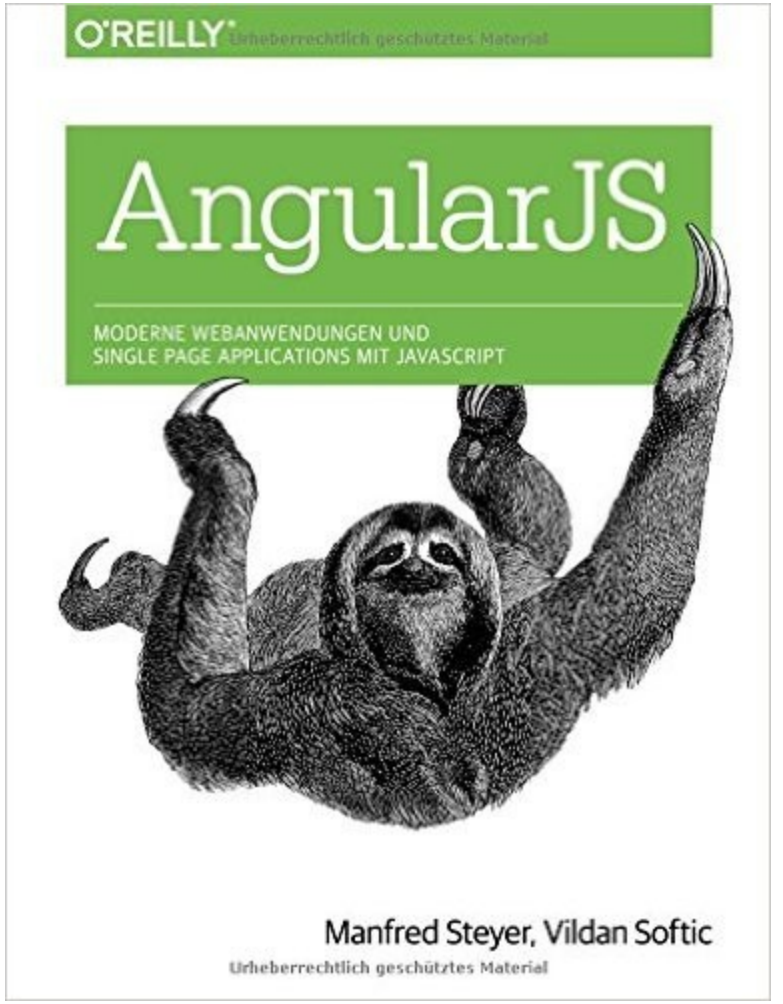
Vildan Softic | Ranorex GmbH

JavaScript beyond the Basics

Vildan Softic

- Graz / Österreich
- Campus02 IT & IT-Marketing
- Senior Software Developer
- @ Ranorex GmbH
- Core Team Mitglied Aurelia
- (aurelia.io, I18N, Animations)
- Consulting und JavaScript Training
- (http://angular.at/)
- AngularJS Buch



Ranorex Online BETA

Log in

Automated web testing reinvented.

Effortlessly record workflows on your website or web application to see if they look and work as expected. You can reuse your recordings across all operating systems and popular browsers, or share them with anyone. Now's the time to dramatically improve the quality of your web presence.

e.g. www.ranorex.com

Start testing for free

No plugin. No installation. Just online.

Ranorex Online

- Purer JavaScript Wahnsinn
  - Event sniffing
  - Monkey Patching
  - Tabübergreifende Kommunikation
  - CORS IFrames !!!
  - Native / synthetische Events

Ranorex Online

- C# Mono on Linux / Win
- Redis / Mongo
- Git push → deploy (CI, CD)
- Containers

Ranorex Online

- No Plugins, no install
- Free Beta ☺

ranorex.io

Agenda

- VanillaJS, das bessere „Framework“
- MicroTask vs MacroTask Queue
  - setTimeout vs Promise
- HTML5 API
  - Page Visibility API
  - DOM Mutation Observers
- ESNext Sprachfeatures
  - Async / Await
  - Immutable Data Structures
- Zusammenfassung

VanillaJS aka pure JavaScript

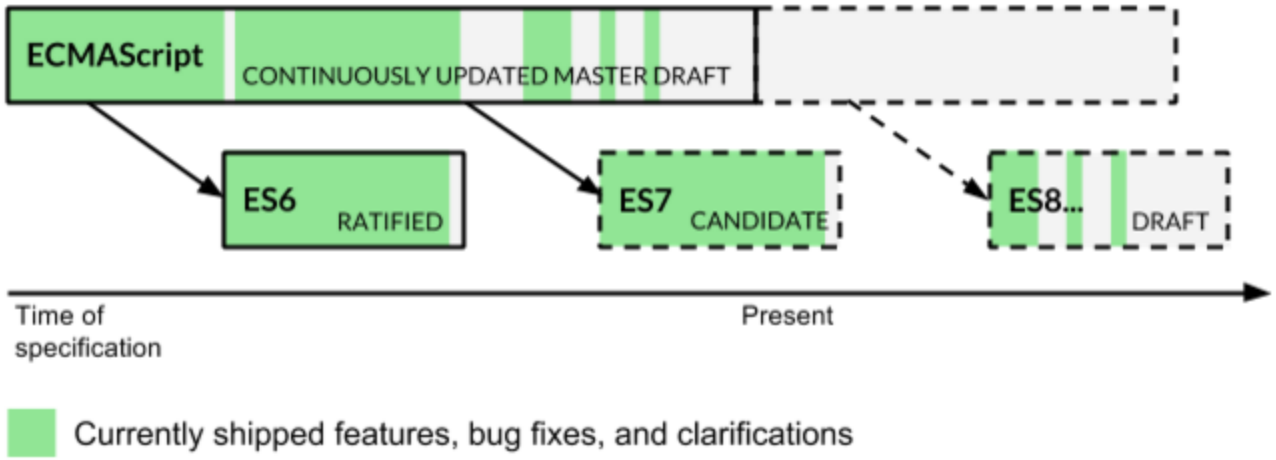
Warum VanillaJS?

- Kein Transpiler / Superset
- ES6 und ESNext
- Performance, performance, performance !
- Grundlagen vs. spezifisches Wissen
- Reduzierter Lock-In Effekt
- Neueste Trends direkt ohne „Wrapper“ verwenden

Warum VanillaJS?

- Front & Backend (NodeJS)
- Riesige Auswahl an Modulen (npm)
- Feature-Deprecation-State über Jahre hinweg
  - synchrone Ajax Anfragen
  - keyCode vs which
- Neue Domänen (Mongo, ReactNative, IoT ...)

Version Support



- https://kangax.github.io/compat-table/es6/
- http://caniuse.com/

Neues schon heute?

- BabelJS: https://babeljs.io/
- Playground: https://babeljs.io/repl/

- Typsichere Sprache
- Superset von JS
- ES6 / ESNext

TypeScript

HTML5 API

HTML5 APIs

- Semantik
- Leistung & Integration
- Konnektivität
- Offline Support
- Multimedia
- 3D, Graphiken & Effekte
- Native Funktionen
- Styling

HTML5 APIs

- <video>, <time>, …
- WebWorker, History
- WebSockets
- AppCache, IndexedDB
- Camera API
- WebGL, SVG
- Touch, Geolocation, ...
- CSS 3

687 APIs

https://developer.mozilla.org/en-US/docs/Web/API

Betrachtete APIs

- Page Visibility API
- DOM Mutation Observer

Page Visibility API

- Auskunft ob eine Website sichtbar / aktiv ist.
- Funktioniert mit Tabs / Fenstern
- Abrufbar über Event **visibilitychange**

Page Visibility API

- Use Cases:
  - Pausieren von Audio/Video Wiedergabe
  - Unterbrechen von Polling Tasks
  - Pausieren von Animationen

Verbreitung

Chrome 33, Opera 12.10, Firefox 18,

- Safari 7 & IE 10
- Android 4.4, Safari Mobile 7

- Polyfills: https://gist.github.com/addyosmani/1122546

DEMO

DOM Mutation Observer

- Auskunft über Veränderungen des DOM
- Granularität konfigurierbar
  - ChildList
  - Attribute
  - Subtree (Änderungen an ChildNodes)
  - CharacterData (Text, Comment, ...)

DOM Mutation Observer

- Löst Mutation Events (DOM3) ab

```
element.addEventListener("DOMNodeInserted", function (ev) {
   //  ...
}, false);
```

-
- Liefert Array von MutationRecords zurück
- Event Callbacks sind MicroTasks !!!

DOM Mutation Observer

- Use cases:
- Reagieren auf Veränderungen des DOM
  - durch 3rd Party Module
  - Browser Extensions

Verbreitung

-
- Chrome 26, Opera 15, Firefox 14
- Safari 6 & IE 11
- Chrome Android 26, Safari Mobile 6/7
-
- Polyfills: https://github.com/Polymer/MutationObservers

DEMO

MicroTask vs MacroTask Queue

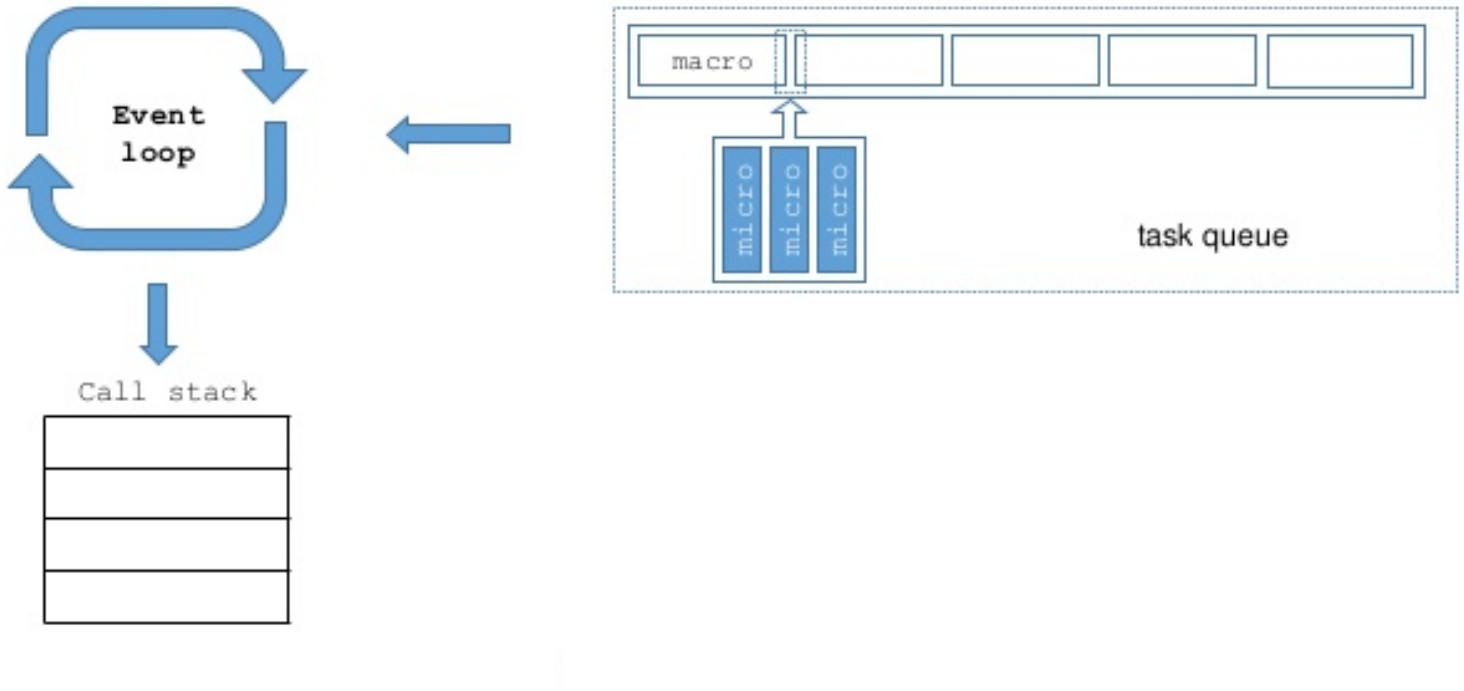
MacroTask Queue

- Ein Task nach dem anderen
- Nach jedem EventLoop
- Queueing mittels setTimeout / setInterval

MicroTask Queue

- Werden am Ende des aktuellen MacroTasks ausgeführt
- Reihen sich hinten nach bestehenden ein
- Auch wenn aus einem vorhergehendem MicroTask heraus erstellt
- Garantierte Ausführung aller MicroTasks vor nächstem MacroTask

JavaScript EventLoop / TaskQueue



Was sind Promises?

- Ein Objekt das einen möglichen künftigen Wert oder Exception darstellt
- Asynchron
- Drei Zustände:
  - Pending
  - Fulfilled
  - Rejected

Was sind Promises?

```
Promise {
   [[PromiseStatus]]: "pending",
   [[PromiseValue]]: undefined
}
```

- Alternative zu Callbacks
- Statt Funktionen aufzurufen, können Werte zurückgegeben werden.

Async mit Callbacks

```
function asyncJob ( cb ) {
  setTimeout( () =>  {
    cb("Done");
  }, 2000);
}

function done(msg) {
  console.log(msg);
}

asyncJob (done);
// -> Done [nach 2s]
```

Async mit Promises

```
function asyncJob () {
  return new Promise( (resolve, reject) => {
    setTimeout( () =>  {
      resolve("Done");
    }, 2000);
  });
}

asyncJob ().then( (msg) => { console.log(msg); });
// -> Done [nach 2s]
```

Statische Funktionen

- Promise.all(iterable) : Promise<any[]>
- Promise.race(iterable) : Promise<any>
- Promise.reject(reason) : Promise<any> / rejected
- Promise.resolve(value) : Promise<any> / resolved

Verbreitung

Chrome 32, Opera 19, Firefox 29,

- Safari 8 & Microsoft Edge
- Polyfills: https://github.com/stefanpenner/es6-promise#readme

DEMO

ES6 & ESNext Sprachfeatures

ES6 & ESNext Sprachfeatures

ES6

- Classes
- Const / Let
- Rest / Spread
- Template Literals
- Modules
- Promises

ESNext

- SharedMem/Atomics
- String.padding
- Object static methods
- Async / Await

Async / Await

- Syntax Sugar für Promises

```
async function asynchron() {
  return "Hello World";
}
```

```
function asynchron() {
  return Promise.resolve("Hello World");
}
```

```
asynchron();
> Promise {[[PromiseStatus]]: "resolved", [[PromiseValue]]: "Hello World"}
```

Async / Await

- Error handling mit try / catch

```
async function add(a, b) {
  if (isNaN(parseInt(a)) || isNaN(parseInt(b))) {
    throw new Error("Keine gültigen Zahlen übergeben");
  }

  return a + b;
}
```

```
add(1, 2);
> Promise {[[PromiseStatus]]: "resolved", [[PromiseValue]]: 3}

add(1, "a");
> Promise {[[PromiseStatus]]: "rejected", [[PromiseValue]]: Error: Keine gültigen Zahlen übergeben ...}
```

Async / Await

- await nur innerhalb async function

```
add(1, 2) + add(3,4);
> "[object Promise][object Promise]"
```

```
async function sum() {
  return await add(1, 2) + await add(3, 4);
}

sum();
> Promise {[[PromiseStatus]]: "resolved", [[PromiseValue]]: 10}

await sum();
> Uncaught SyntaxError: Unexpected identifier
```

DEMO

Immutable Data Structures

- Einmal erzeugt, nie mehr verändert
- Reduktion von Seiteneffekten
- Bestandteil purer / reiner Funktionen
- Fördern Komposition

Immutable Data Structures

- Referentielle Transparenz
  - add(1, 2) === 3
- Vereinfachte Persistenz
  - persistent data structures
  - Garbage Collection (orig + changes)
- Vereinfachtes Diffing
  - Vgl. Objekte vs Zugriffe nachverfolgen
- Unterstützen Lazy Operations
  - Beliebig kombinierbar
  - Memoization mit WeakMap

DEMO

Zusammenfassung

- JavaScript wird immer mächtiger
- Grundlagen sind für alle Frameworks anwendbar
- Solides Verständnis von asynchronen Tasks ist unersetzbar
- Sprachfeatures erleichtern die Arbeit
- Mehr APIs für zahlreiche UseCases
- Immutability vereinfacht die Nachvollziehbarkeit einer Applikation

Vildan Softic | Ranorex GmbH

Vielen Dank!

FRAGEN?