WBE: WEB-ENTWICKLUNG

ABSCHLUSS

ÜBERSICHT

- React: Konzept und Bibliothek
- Ausblick: Weitere Themen rund ums Web
- Abschluss, Feedback
- Anhang: Themenliste WBE

ÜBERSICHT

- React: Konzept und Bibliothek
- Ausblick: Weitere Themen rund ums Web
- Abschluss, Feedback
- Anhang: Themenliste WBE

REACT

- React als Biblithek zum Bau von UIs
- Bisher in diesem Semester:
  - Hinweise zum Aufbau einer solchen Bibliothek
  - Eigene Implementierung eines Mini-React (SuiWeb)
- Es ist Zeit, React noch etwas genauer anzusehen

REACT

„A JavaScript library for building user interfaces”

- Declarative
- Component-Based
- Learn Once, Write Anywhere

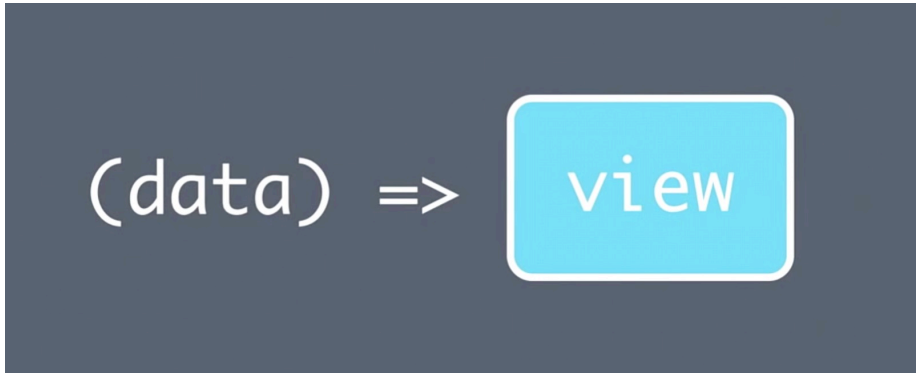
Facebook, Instagram
2013 vorgestellt

https://react.dev

Speaker notes

- Kein Mega-Framework
- Keine "full-stack"-Lösung

React.js verpackt eine imperative API (DOM) in eine deklarative API.



Wenn Sie React.js einmal ausprobieren möchten, gibt es verschiedene Möglichkeiten:

https://react.dev/learn/installation

Die React-Entwickler empfehlen mittlerweile, eines der auf React basierenden Frameworks wie Next.js, Remix, Gatsby oder Expo zu verwenden. Leider ist dadurch der Einstieg mit einem überschaubarem Setup schwieriger geworden.

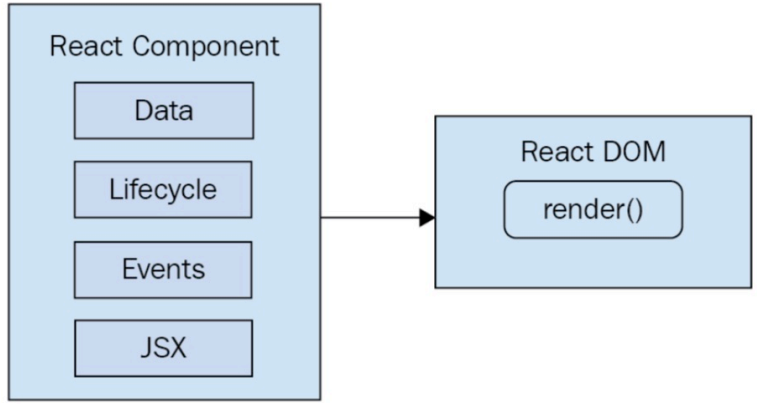
Eine Alternative ist, *Vite* zu verwenden:

```
# npm 7+, extra double-dash is needed:
npm create vite@latest my-react-app -- --template react
```

https://vitejs.dev/guide/

ZWEI TEILE

- React DOM
  - Performs the actual rendering on a web page
- React Component API
  - Data to be rendered
  - Lifecycle methods
  - Events: respond to user interactions
  - JSX: syntax used to describe UI structures



Speaker notes

Einfachheit als Vorteil

- Eine überschaubare API kann ein Vorteil sein
- Intern ist trotzdem viel zu tun

Zeitlicher Ablauf

- React-Komponenten verwenden Daten, welche ihnen übergeben werden
- Daten stellen den dynamischen Askekt des UI dar
- Ergebnis: Sammlung von gerenderten UI-Komponenten

React is simple, because it doesn’t have a lot of moving parts

KOMPONENTEN UND KLASSEN

```
// ES5
var HelloComponent = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>
  }
})

// ES6
class HelloComponent extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>
  }
}

// Function Component
const HelloComponent = (props) => {
  return (<div>Hello {props.name}</div>)
}
```

KOMPONENTEN

```
const MyComponent = () => (
  <section>
    <h1>My Component</h1>
    <List data={["Maria", "Hans", "Eva", "Peter"]} />
  </section>
)

const List = ({data}) => (
  <ul>
    { data.map(item => (<li key={item}>{item}</li>)) }
  </ul>
)

const root = createRoot(document.getElementById('app'))
root.render(
  <MyComponent />
)
```

ZUSTAND

```
const Counter = () => {
  const [state, setState] = useState(1)
  const handler = () => setState(c => c + 1)

  return (
    <h1 onClick={handler} style={{userSelect:"none",cursor:"pointer"}}>
      Count {state}
    </h1>
  )
}

const root = createRoot(document.getElementById('counter'))
root.render(
  <Counter />
)
```

PROPERTIES

```
1  const MyButton = (props) => {
2    const { disabled, text } = props
3    return (
4      <button disabled={disabled}>{text}</button>
5    )
6  }
7
8  const root = createRoot(document.getElementById('counter'))
9  root.render(
10   <main>
11     <MyButton text='My Button' disabled=true />
12   </main>
13 )
```

UND SONST

- Funktions- und Klassenkomponenten unterstützt
- Funktionskomponenten mit Hooks (u.a. State Hook)
- Diverse Optimierungen: virtuelles DOM, Fibers
- Entwicklertools, React Devtools
- Serverseitiges und clienseitiges Rendern
- Komponententechnologie auch für native iOS und Android Apps verwendbar (React Native)

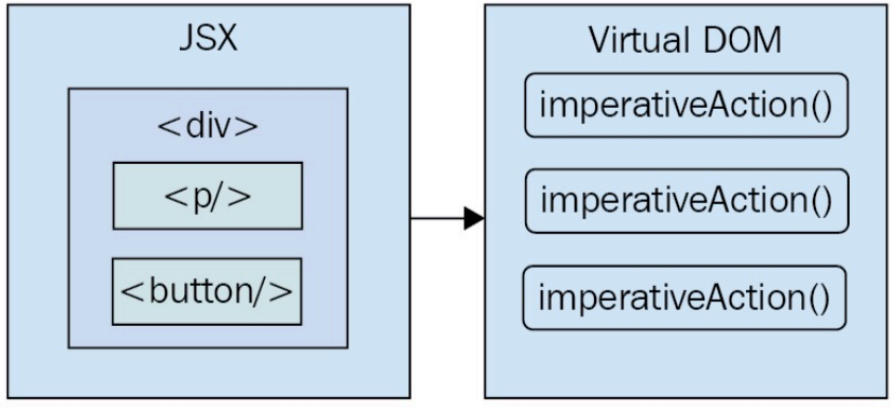
Speaker notes

PERFORMANZ

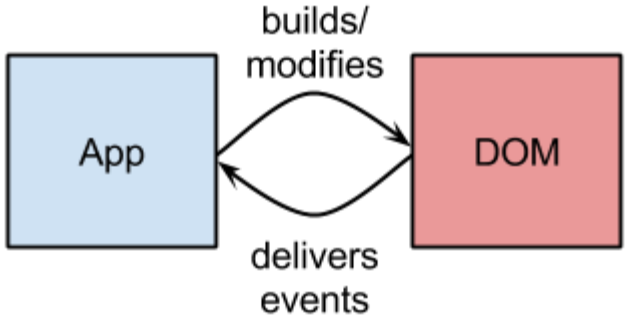
- Das ist die Herausforderung eines deklarativen Ansatzes
- Komplettes Neu-Rendern ins DOM wirkt sich schlecht auf die Performanz aus
- Traditioneller Ansatz: deklarative Templates (z.B. Handlebars) kombiniert mit imperativem Code, um die dynamischen Aspekte des UI zu implementieren

VIRTUELLES DOM

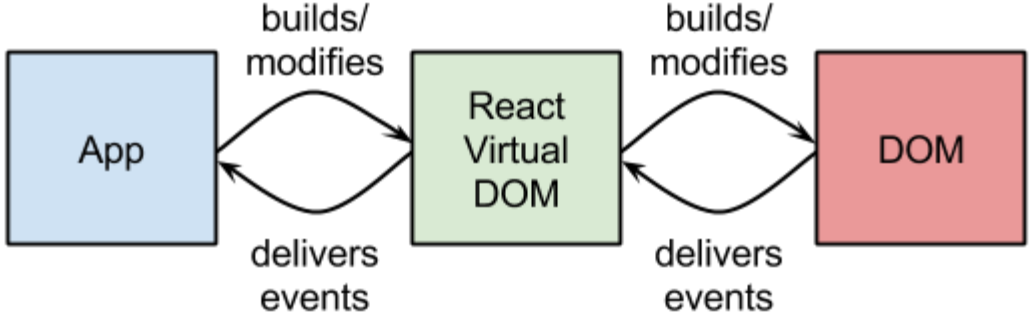
- Baumstruktur, welche das DOM im Speicher repräsentiert
- Berechnet, welche DOM-Inhalte angepasst werden müssen
- Nur die nötigen DOM-Operationen werden durchgeführt
- Applikations-UI kann bei Änderung neu generiert werden
- Verwaltung des Applikationszustands so deutlich einfacher
- Ermöglicht auch das Rendern auf dem Server



Traditionelles Modell: App passt DOM direkt an (z.B. mit Hilfe von jQuery). Das wird schnell unübersichtlich für Entwickler. Ausserdem sind DOM-Updates teuer im Ressourcenverbrauch.



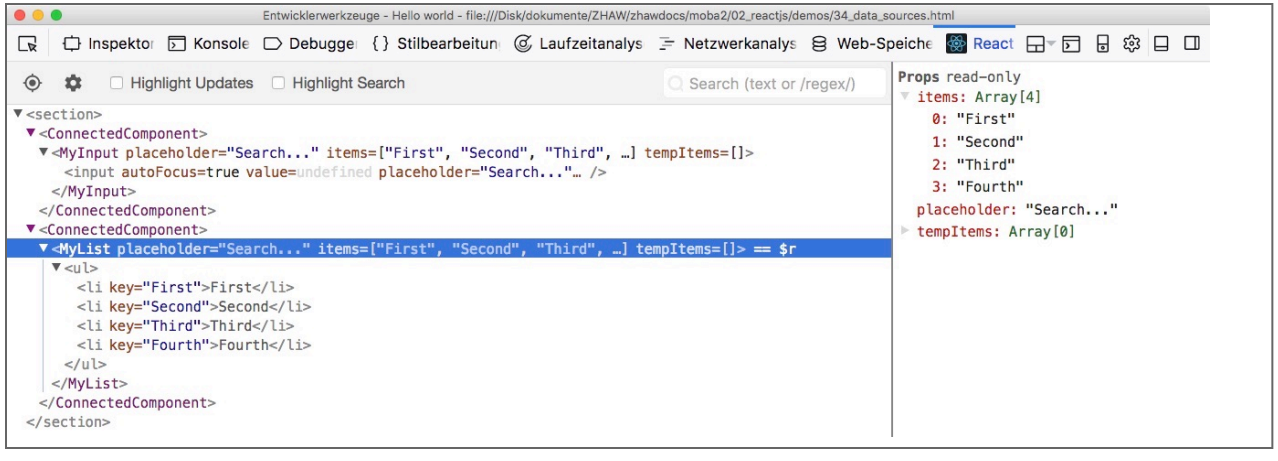
Virtuelles DOM:



Updates im virtuellen DOM werden nicht unmittelbar im Browser-DOM nachgeführt, sondern synchronisiert mit Viewport-Updates. Dann wird auch nur die berechneten Unterschiede zum vorangehenden Zustand als Aktualisierungen im Browser-DOM angewendet. Das selbst zu machen wäre aufwändig und fehleranfällig. Es ist eine Aufgabe des Frameworks.

REACT DEVTOOLS

- Browser-Erweiterung (Firefox, Chromium)
- Untersuchen der Komponentenhierarchie möglich



https://github.com/facebook/react/tree/master/packages/react-devtools

WAS IST NUN REACT?

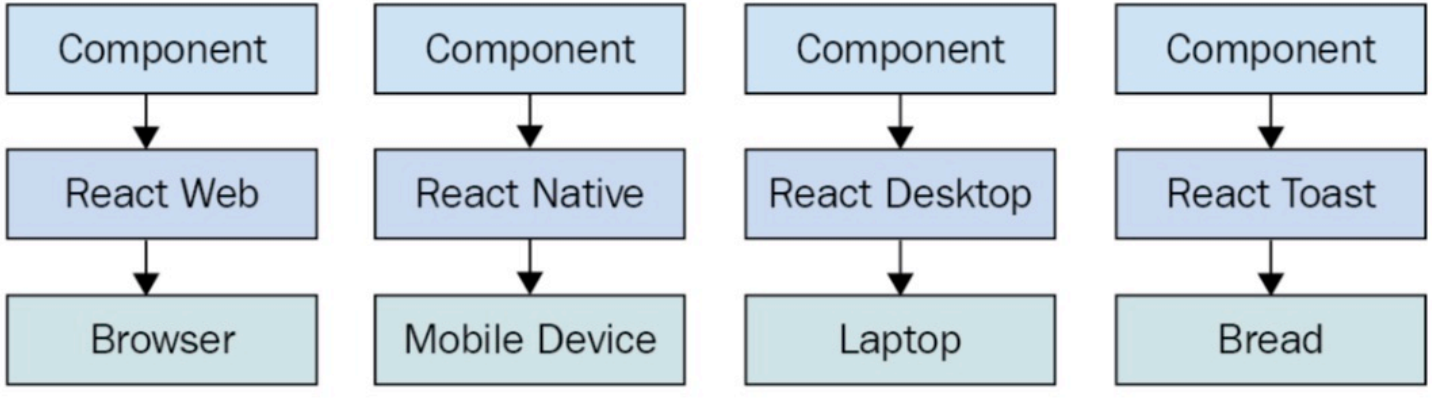
- React bildet die View einer Applikation
- Nicht (nur) Bibliothek, sondern in erster Linie Konzept
- Unterstützt das Organisieren von Vorlagen in Komponenten
- Das virtuelle DOM sorgt für schnelles Rendern

POWER OF COMPONENTS

- Kleinere Einheiten entwickeln
- Weniger Abhängigkeiten
- Einfacher zu verstehen, zu pflegen, zu testen
- Komponentendesign: für genau eine Sache verantwortlich
- Zustand in wenigen Komponenten konzentrieren

Speaker notes

- Es ist gar nicht so wichtig, was das Ziel der Ausgabe ist
- React ist potenziell für jede Art von UI geeignet



REACT NATIVE

- React ist nicht nur als Web-Framework geeignet
- Analog können auch native Apps damit erstellt werden
- React Native: Apps für iOS und Android entwickeln

„Learn once, write anywhere” (Facebook-Entwickler)

HAUPTKONZEPTE

- Klarer und einfacher Datenfluss:
  - Daten nach unten weitergegeben (props)
  - Ereignisse nach oben weitergegeben und dort behandelt
- Properties werden nicht geändert, Zustand ist veränderbar
- Zustand wird von Komponente verwaltet
- Es ist von Vorteil, die meisten Komponenten zustandslos zu konzipieren

Existing Frameworks Influenced: All of them

- Angular komplett überarbeitet
- Neue Frameworks entstanden: Vue.js, Svelte, ...
- Entwicklung nativer Mobil-Apps: SwiftUI, Compose
- ...

Speaker notes

INFORMATIONEN ZUR VERTIEFUNG

- React: Quick Start and Docs
  https://react.dev/learn
- Tutorial: Intro To React
  https://react.dev/learn/tutorial-tic-tac-toe
- Babel – a JavaScript compiler
  http://babeljs.io

WEITERES MATERIAL

- The React.js Way: Getting Started Tutorial
  http://blog.risingstack.com/the-react-way-getting-started-tutorial/
- OSCON 2014: How Instagram.com Works; Pete Hunt
  https://www.youtube.com/watch?v=VkTCL6Nqm6Y
- Removing User Interface Complexity, or Why React is Awesome
  http://jlongster.com/Removing-User-Interface-Complexity,-or-Why-React-is-Awesome

QUELLEN

- React – A JavaScript library for building user interfaces
  https://reactjs.org
- Adam Boduch: React and React Native, Second Edition, Packt Publishing, 2018

AUSZUG AUS FOLIEN VON WEB3:

Why are we not using Angular?

- As of today (09.10.2018), React has > 112’000 Stars (43% increase to 2017) and > 1’200 contributors on Github.
- Angular only has a third of the stars and half the contributors whilst being around for a longer time.
- What it does have is six times more bug reports (2200 vs. 300)

Angular is certainly popular and some big companies in Zürich started using it. But it’s certainly not more popular.😇

Technical Reasons

Apart from being less popular than React, there are reasons not to use Angular as educational software.

- Angular has a huge API surface, the learning curve is pretty bad
- Angular has lots of syntax
- It’s hard to extrapolate a well known paradigm (like MVC) from Angular
- Angulars’ API has been constantly shifting (not just from v1 to v2, but also in between the v1 versions)

Anyway, this is not meant to start a flame war. If you want to use Angular at home/business, please do so 😊

ÜBERSICHT

- React: Konzept und Bibliothek
- Ausblick: Weitere Themen rund ums Web
- Abschluss, Feedback
- Anhang: Themenliste WBE

HAUPTTHEMEN IN WBE

- JavaScript die Sprache (und Node.js)
- JavaScript im Browser
- Client-seitige Web-Apps

WEITERE THEMEN RUND UMS WEB

Rund ums Web gibt es noch viele spannende Themen...

Ein paar Anregungen sind auf den folgenden Slides zusammengestellt (ohne Anspruch auf Vollständigkeit)

HTML und CSS

- Grundlagen: als Vorkenntnisse für WBE
- Skript im Vorbereitungskurs (Moodle)
- Diverse Tutorials (ein paar im Kurs verlinkt)

▷ Vorbereitungskurs WBE

Web-Apps für Mobilgeräte

- Layout für verschiedene Devices (Smartphones, ...)
- Responsives Webdesign (u.a. Bilder)
- Web-APIs für Gerätesensoren
- Apps basierend auf React und Ionic
- React Native / Expo

▹ MOBA (?)

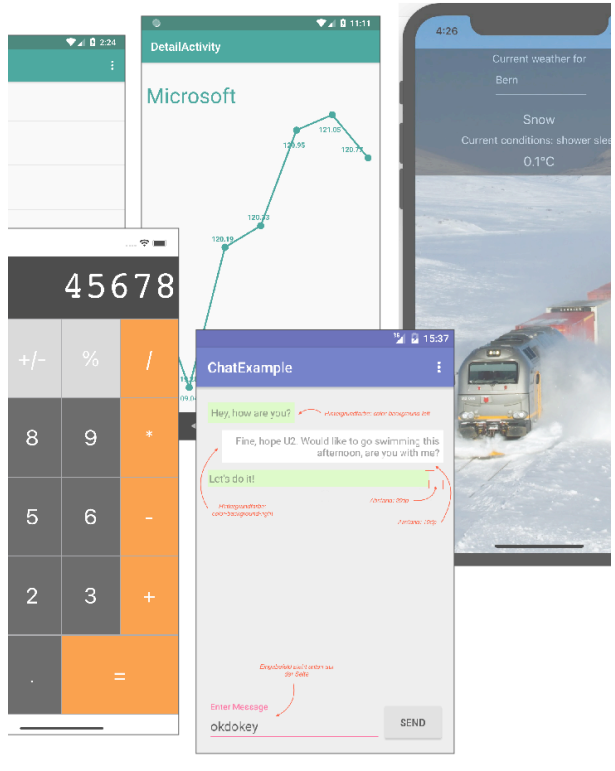
Mobile Applications (MOBA1/MOBA2)

- Mobile Layouts, CSS Flexbox
- Device APIs, Sensoren
- Web Components, React, Ionic
- React Native

und:

- Android native (Kotlin, Compose)
- iOS native (Swift, SwiftUI)

Weitere Durchführung nach HS25 noch offen...



Apps mit Webtechnologien

- Desktop-Applikationen mit Web-Technologien
  https://www.electronjs.org
  https://nwjs.io
- Basis für Applikationen wie VSCode
- Diverse weitere Frameworks in diesem Bereich
- Mobil-Applikationen mit Web-Technologien
  https://cordova.apache.org
  https://capacitorjs.com

WebAssembly (WASM)

- Bytecode zur Ausführung in Webbrowsern
- Ziel: höhere Performanz für Web-Applikationen
- Verschiedene Programmiersprachen kompilieren zu WASM
- Erste Version funktioniert in aktuellen Browsern bereits

https://webassembly.org

▷ PSPP (?)

JavaScript-Alternativen

- Werden nach JavaScript „kompiliert”
- TypeScript (Microsoft)
  - statisches Typenkonzept
- ReScript (ehemals ReasonML)
  - speziell für React-Ansatz geeignet
  - funktionaler Ansatz, an OCaml angelehnt
- ClojureScript (Lisp-Dialekt)

▷ PSPP (?)

Speaker notes

In PSPP werden u.a. Typenkonzepte, funktionale Programmierung und Lisp behandelt. Für einige Beispiele zur funktionalen Programmierung wird auch JavaScript verwendet. Von den hier erwähnten Programmiersprachen wird ausser JavaScript vor allem ClojureScript kurz thematisiert.

Funktionale Programmierung

- JavaScript ist eine Multiparadigmensprache
- Es eignet sich sehr gut für funktionale Programmierung *(higher order functions, partial application, currying, ...)*
- In WBE wird dieser Aspekt kaum thematisiert

▷ PSPP (?)
▷ FUP

Programmiersprachen und -Paradigmen (PSPP)

- Compiler, Bytecodes (WASM)
- Logische Programmierung (Prolog)
- Objektorientierte Programmierung (Smalltalk)
- Funktionale Programmierung (Lisp, Python)
- und: Modulkonzept, Scriptsprachen, Typenkonzepte

Weitere Durchführung nach HS25
noch offen...

Design, Usability, ...

- Grafische Gestaltung
  - Gestaltungsprinzipien
  - Farbenlehre
  - Typografie
- Usability
- Barrierefreiheit

▷ Vorbereitungskurs WBE (design-usability.pdf)

Zurück zu JavaScript ...

DOUGLAS CROCKFORD

Autor von: JavaScript: The Good Parts

> *„The idea of putting powerful functions and dynamic objects in the same language was just brilliant. That's the thing that makes JavaScript interesting."*

FullStack London 2018
https://www.youtube.com/watch?v=8oGCyfautKo

> *„My advice to everybody who wants to be a better programmer is to learn more languages. A good programming language should teach you. And in my career the language which has taught me the most was JavaScript."*

The Better Parts. JS Fest 2018
https://www.youtube.com/watch?v=XFTOG895C7c

ÜBERSICHT

- React: Konzept und Bibliothek
- Ausblick: Weitere Themen rund ums Web
- Abschluss, Feedback
- Anhang: Themenliste WBE

ÜBERBLICK WBE

Woche	Thema
1	Einführung, Administratives, das Web im Überblick
2	JavaScript: Grundlagen
3	JavaScript: Funktionen
4	JavaScript: Prototypen von Objekten
5	JavaScript: Asynchrones Programmieren
6	TypeScript: JavaScript mit Typen
7	JavaScript: Webserver
8-9	Browser-Technologien: JavaScript im Browser
10	Browser-Technologien: Client-Server-Interaktion
11-13	UI-Bibliothek: Komponenten, Implementierung, Einsatz
14	Abschluss: React, Feedback

WBE-ZIELE

In erster Linie:
Solide Kenntnisse in grundlegenden Web-Technologien, speziell JavaScript, denn dies ist die Programmiersprache des Web.

Grundlagen:
HTML und CSS als Basistechnologien des Web muss man natürlich auch kennen, um mit Webtechnologien entwickeln zu können.

Ausserdem:
Einen Überblick erhalten über einen für heutige Anforderungen relevanten Ausschnitt aus dem riesigen Gebiet der Web-Technologien.

ALLGEMEINE BETRACHTUNG

- Themen, welche vertieft behandelt wurden

  Grösserer Block in mindestens einer Vorlesung, also nicht nur zwei bis drei Slides dazu, in der Regel auch im Praktikum thematisiert

- Themen welche nebenbei behandelt wurden

  Im Sinne von: das gibt’s auch, sollte man kennen, wenn man sich mit Webtechnologien beschäftigt, Einarbeitung nach Bedarf

ALLGEMEINE BETRACHTUNG

- Themen, welche vertieft behandelt wurden

  Mit diesen Themen sollte man sich auskennen (ein paar mehr Details im Anhang)

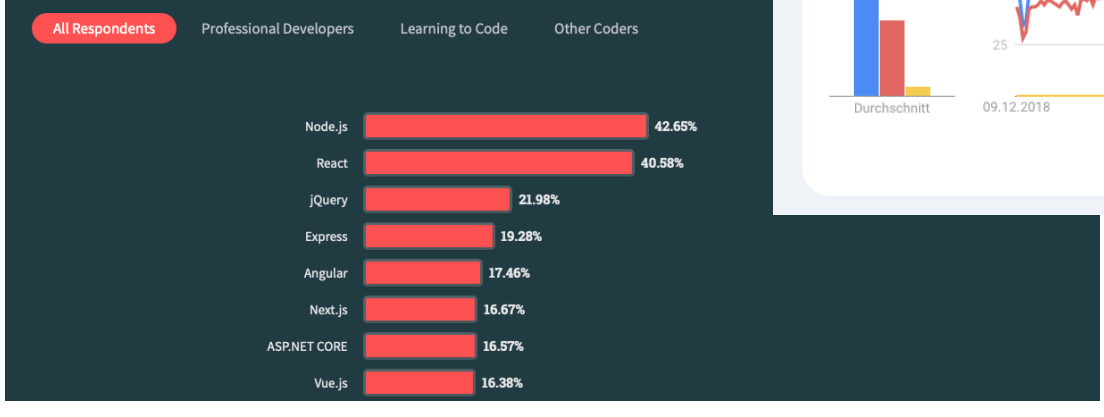
- Themen welche nebenbei behandelt wurden

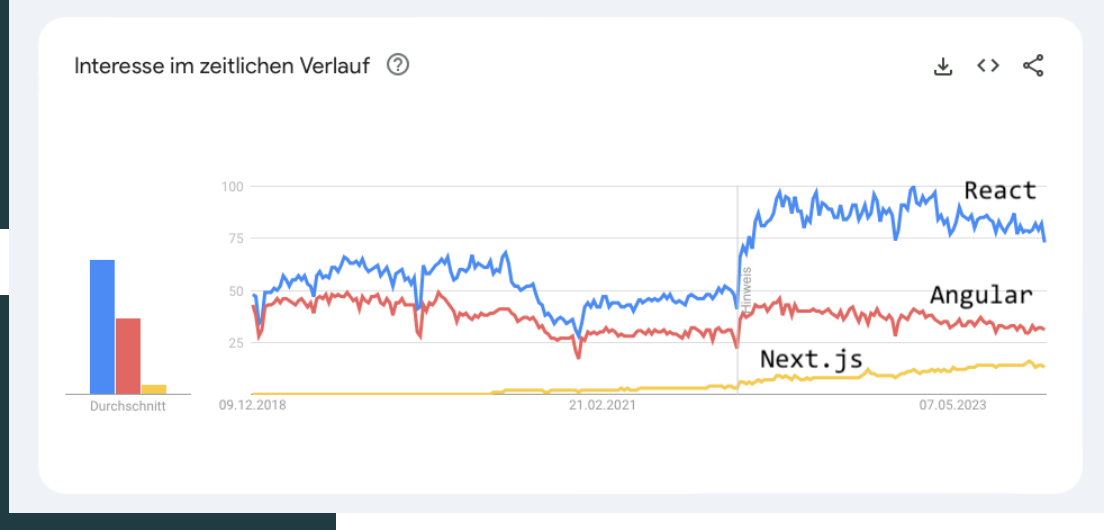
  Hier sollte man wissen, worum es geht, dazu gehören ein paar wesentliche Merkmale der Technologie, des Frameworks oder der Idee, aber Details sind hier nicht das Ziel

BITTE UM FEEDBACK

- Inhalte?
- Stoffumfang?
- Praktika?
- Art der Durchführung?

STACKOVERFLOW SURVEY, GOOGLE TRENDS





TIOBE Index for December 2023

Dec 2023	Dec 2022	Change	Programming Language	Ratings	Change
1	1		Python	13.86%	-2.80%
2	2		C	11.44%	-5.12%
3	3		C++	10.01%	-1.92%
4	4		Java	7.99%	-3.83%
5	5		C#	7.30%	+2.38%
6	7	↑	JavaScript	2.90%	-0.30%
7	10	↑	PHP	2.01%	+0.39%
15	23	⇈	Kotlin	0.92%	+0.34%
16	16		Delphi/Object Pascal	0.92%	+0.07%
17	15	↓	Swift	0.82%	-0.09%
18	20	↑	Rust	0.80%	+0.12%
38			TypeScript	0.36%	
39			Lisp	0.34%	
40			Haskell	0.31%	
41			ML	0.31%	

ÜBERSICHT

- React: Konzept und Bibliothek
- Ausblick: Weitere Themen rund ums Web
- Abschluss, Feedback
- Anhang: Themenliste WBE

ÜBERBLICK

- Ganzes Thema wichtig
  - inklusive Unterthema
- Thema teilweise wichtig
  - zum Beispiel dieses Unterthema
  - Unterthema: Überblick genügt
- Überblick genügt
  - Unterthema ebenso

Speaker notes

Die Themenliste in diesem Anhang soll als Orientierung dienen, welche Bedeutung die einzelnen Themen und Unterthemen für WBE – auch im Hinblick auf die Prüfung – haben. Das sollte eigentlich bereits aus der Art der Behandlung der Themen in Unterricht und Praktikum klar geworden sein.

Noch ein Hinweis:

Zu Themen, welche als "Überblick genügt" angegeben sind, können durchaus auch Fragen in der Prüfung vorkommen. Diese gehen dann aber erstens weniger ins Detail und sind zweites nicht Voraussetzung fürs Bestehen der Prüfung.

GRUNDLAGEN: HTML & CSS

- Markup und HTML
  - Konzept von Markup verstehen
  - Eckpunkte der Entwicklung von HTML kennen
- Aufbau eines HTML-Dokuments
  - Grundbegriffe: Element, Tag, Attribut
  - Grundlegende Elemente: html, head, title, meta, body, p, div, span, p, img, h1, ..., ul, ol, li
  - Weitere Elemente: header, article, ...
  - Attribute: contenteditable, data-
  - Bild- und Grafikformate, SVG

GRUNDLAGEN: HTML & CSS

- Darstellung mit CSS
  - CSS mit HTML verbinden, CSS-Regeln
  - Selektoren
  - Einige Eigenschaften, Grössen- und Farbangaben
    (am besten an Beispielen und Aufgaben orientieren)
  - Schriften laden, Transitionen, Transformationen, Animationen
  - Weitere Eigenschaften
- Werkzeuge und Hilfsmittel

GRUNDLAGEN: HTML & CSS

- Das Box-Modell
  - overflow, width, height, margin, padding, border
  - border-radius, color, background-color
  - Farbverläufe, Sprites
- Positionierung und fliessende Boxen
  - position, float, clear, display (block, inline, none)

1. DAS WEB

- Internet und WWW
  - Einige Eckpunkte der Entwicklung kennen
- Client-Server-Architektur
  - Konzepte und wesentliche Tools kennen
  - User Agents, Webserver
  - URI/URL, IP-Adresse, Domain-Name
  - Grundzüge des HTTP-Protokolls

1. DAS WEB

- Die Sprachen des Web: HTML, CSS, JavaScript
  - Vorkenntnisse / Vorkurs
- Web-Standards und APIs
  - W3C und WHATWG kennen
  - clientseitige vs. serverseitige Technologien

2. JAVASCRIPT GRUNDLAGEN

- JavaScript und Node.js
  - Einige Eckpunkte der Entwicklung
  - Node.js als JavaScript-Laufzeitumgebung
  - Node.js Einsatz, REPL, NPM
  - console.log
- Werte, Typen, und Operatoren
  - Zahlen, typeof, Strings, logische Ausdrücke, ...

2. JAVASCRIPT GRUNDLAGEN

- Programmstruktur
  - Ausdruck vs. Anweisung
  - Syntax, Variablen, Kontrollstrukturen, Kommentare, ...
- Funktionen
  - Überblick, mehr später

2. JS: OBJEKTE UND ARRAYS

- Objekte
  - Objektliterale, Attribute, Methoden, ...
  - Methoden von `Object`: `assign`, `keys`, `values`
- Spezielle Objekte: Arrays
  - Array-Literale
  - Schleifen über Arrays
  - Array-Methoden: `slice`, `concat`, `Array.isArray`
  - Weitere Methoden schaut man bei Bedarf nach
- JSPN
  - `JSON.stringify`, `JSON.parse`

3. JS: FUNKTIONEN

- Funktionen definieren
  - Definition und Deklaration, Pfeilnotation
  - Gültigkeitsbereiche
- Parameter von Funktionen
  - Default-, Rest-Parameter, arguments
  - Spread-Operator
  - Arrays und Objekte destrukturieren
- Funktionen höherer Ordnung
  - Arrays: `forEach`, `filter`, `map`, `reduce`

3. JS: FUNKTIONEN

- Closures
  - Einsatz von Closures
  - Pure Funktionen
  - Funktionen dekorieren
  - Funktionales Programmieren
- Mehr zu Node.js
  - Konsole, Kommandozeilenargumente
  - Module in JavaScript
  - NPM, NPX

4. JS: MEHR ZU OBJEKTEN

- Werte- und Referenztypen
  - Unterschied verstehen
  - Wissen, welche Typen in JS Werte- und Referenztypen sind
- Vordefinierte Objekte
  - Wichtigste vordefinierte Objekte kennen
  - Methoden schaut man bei Bedarf nach

Zum vorletzten Punkt: Unterschied zwischen *in eigenem Code verwenden* und *in bestehendem Code verstehen*. Was ein `"Hello World".indexOf("ll")` bedeutet, sollte man sich schon vorstellen können.

4. JS: PROTOTYPEN VON OBJEKTEN

- Prototypen und `this`
  - Bedeutung von `this` je nach Aufruf
  - Strict Mode
  - `call`, `apply`, `bind`
  - Prototyp eines Objekts, `Object.create`
  - Weitere Methoden (getPrototypeOf, getOwnPropertyNames) schlägt man bei Bedarf nach

4. JS: PROTOTYPEN VON OBJEKTEN

- Konstruktoren und Vererbung
  - Konstruktorfunktionen, `new`
  - Prototypenkette
- Gewohntere Syntax: Klassen
  - `class`, `extends`, `constructor`, ...
- Test-Driven Development
  - Konzept verstehen
  - Jasmine einsetzen können

5. JS: ASYNCHRONES PROGRAMMIEREN

- File API
  - Unterschied zwischen `fs.readFileSync` und `fs.readFile`
  - Streams und weitere Methoden
- Reagieren auf Ereignisse
  - Event Loop im Überblick
- Modul „events”
- Promises, Async/Await

6. TYPESCRIPT: JS MIT TYPEN

- TypeScript Überblick und Grundlagen
  - Einführung und Infrastruktur
  - Grundlagen von TypeScript
  - Funktionen, Klassen, Generics
- TypeScript im Detail
  - Erfahrung im Einsatz von TypeScript

7. JS: WEBSERVER

- Internet-Protokolle
  - Internet-Protokoll-Stack
  - Protokolle: FTP, SFTP, SSH
- Das HTTP-Protokoll
  - Grundlagen des Protokolls
  - HTTP-Methoden: GET, POST, PUT, PATCH, DELETE

7. JS: WEBSERVER

- Node.js Webserver
  - Web-Server, -Client, Streams: Code lesen können
  - Beispiel File-Server: Aufbau grob verstehen
- REST APIs
  - Konzept verstehen
  - Alternative GraphQL
- Express.js
  - Für einfache Aufgaben verwenden können
  - Reverse Proxy

8. BROWSER: JAVASCRIPT

- JavaScript im Browser
  - Überblick, ES-Module
- Document Object Model
  - Repräsentation im Speicher, Baumstruktur
  - Verschiedene Knotentypen, Knoten anlegen
  - Array-ähnliche Objekte, `Array.from`
  - Attribute: HTML-Attribute, `className`, `classList`, `style`
  - `requestAnimationFrame`
  - Überblick, was möglich ist (Details kann man nachschlagen)
  - DOM-Scripting-Code lesen können

8. BROWSER: JAVASCRIPT

- Vordefinierte Objekte
  - Allgemeine Objekte und Browser-Objekte
- CSS und das DOM
  - Layout-Angaben im DOM
  - class und style

9. BROWSER: JAVASCRIPT

- Event Handling im Browser
  - Events registrieren: `window.addEventListener`
  - Event-Handler und Event-Objekt
  - Event-Weiterleitung und Default-Verhalten
  - Events: `click`, weitere Events
- Kleiner Exkurs: jQuery
- Bilder und Grafiken
- Weitere Browser-APIs
  - WebStorage
  - History, Geolocation, Workers

10. BROWSER: CLIENT-SERVER

- Formulare
  - Element `form` mit Attributen `method`, `action`
  - Elemente `input`, `label` mit wichtigen Attributen
  - Mehr kann man bei Bedarf nachschlagen
  - Daten mit GET und POST übertragen
  - File-Input, GET und POST in Express
- Cookies, Sessions
  - Konzept verstanden

10. BROWSER: CLIENT-SERVER

- Ajax und XMLHttpRequest
  - Konzept verstanden
- Fetch API
  - Verwenden von `fetch` (Promise)
  - jQuery, Axios, CORS

11. UI-BIBLIOTHEK (1)

- Frameworks und Bibliotheken
  - Unterschied, Eckpunkte der Entwicklung
  - Model-View-Controller, Singe-Page Apps
- DOM-Scripting und Abstraktionen
  - Verschiedene Ansätze im Überblick
- JSX und SJDON
  - Vergleich der Notationen
- Eigene Bibliothek: SuiWeb
  - Ziel, Vorgehen

12. UI-BIBLIOTHEK (2)

- Erste Schritte
  - Interne Datenstruktur, `createElement`, `render`
  - Ansatz verstehen, Code lesen können
- Komponenten und Properties
  - Einsetzen können
  - Details wie sie implementiert sind weniger wichtig
- Darstellung von Komponenten
- Defaults und weitere Beispiele

13. UI-BIBLIOTHEK (3)

- Zustand von Komponenten
  - State-Hook, einsetzen können
  - Kontrollierte Eingabe
  - Details der Implementierung sind weniger wichtig
- Komponenten-Design
  - Container-Komponente
  - Lifecycle-Methoden, Effect-Hook
  - Aufteilen in Komponenten:
    Beispiel nachvollziehen können
  - Deklarativer vs. imperativer Ansatz

13. UI-BIBLIOTHEK (3)

- Ausblick: Optimierungsansätze
  - Aufteilen in Arbeitsschritte, asynchrones Abarbeiten
  - Render- und Commit-Phasen

14. ABSCHLUSS

- React: Konzept und Bibliothek
  - Klassenkomponenten
  - Weitere Konzepte
- Ausblick: Weitere Themen rund ums Web