Webentwicklung

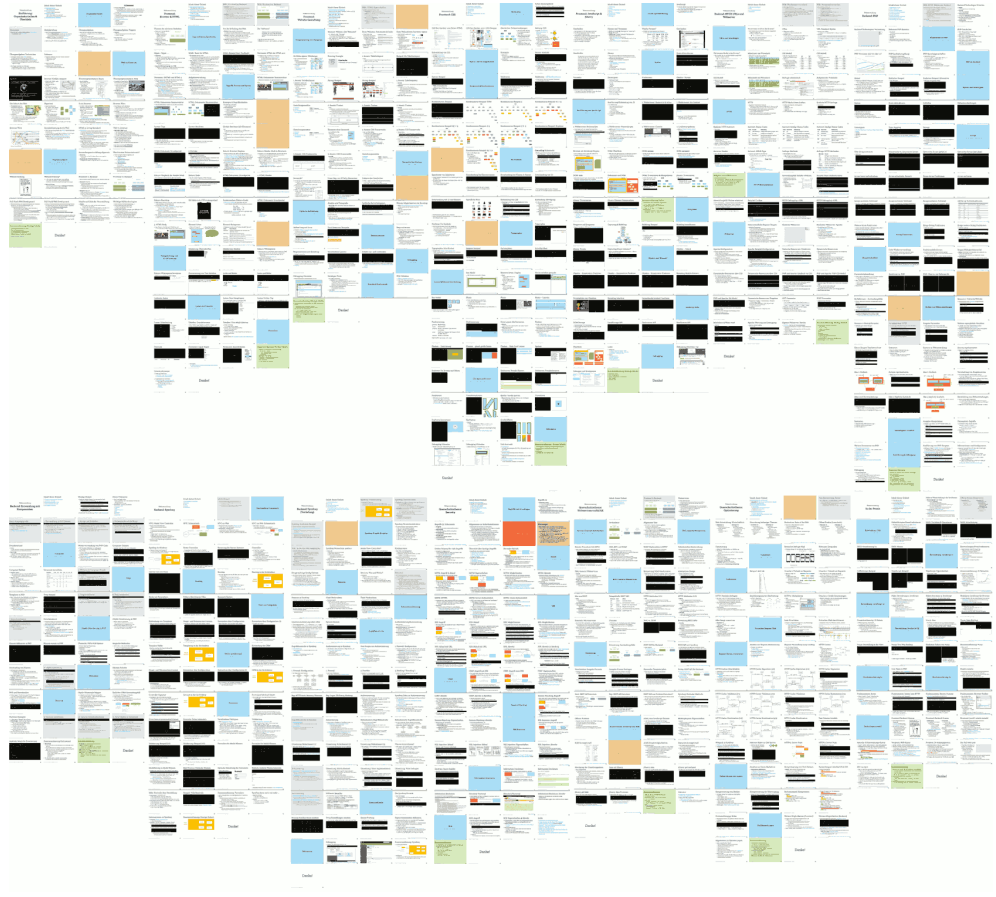
Schnelldurchgang & Abschluss

Inhalt dieser Einheit

1. Der Stoff im Schnelldurchgang
2. Nicht behandelte Themen
3. Ask me Anything
4. Warme Worte zum Schluss

Uff!

- Das war viel Stoff →
- 14 Einheiten
  - über 600 Inhalts-Folien
- **Wichtig:** Trennung von …
  - grundlegende Ideen
  - technische Details
    - (siehe: viele `Beispiele`)
- Gehen wir das mal durch
  - Verweise auf Einzelvorträge jeweils unten auf den Folien



Was ist Webentwicklung?

- **Anwendungs-Perspektive** (Webentwicklung):
  - Entwurf & Implementierung von Websites & Webanwendungen
    - für Erreichung der Anwender-Ziele *(funktionale Anf.)*
    - unter Berücksichtigung *nicht-funktionaler Anforderungen*
      - wie etwa (Informations-)Sicherheit und Performance
  - Als verteilte Systeme mit *Webtechnologien*
    - Kommunikation über **HTTP**, oft: Client-Server-Architektur
    - Client mit **HTML**, **CSS** und **JavaScript**
- **Technisch-betriebliche Perspektive** (Webentwicklung):
  - effiziente Entwicklung, Wartung und Betrieb *solcher* Systeme
    - Kenntnis & Abwägung techn. Möglichkeiten, Technologieauswahl
    - Hilfstechnologien: Versionsverwaltung, Deployment, Tests
  - (denn: es gibt Unterschiede zur allg. Anwendungsentwicklung)

Quelle: Intro

Anwendungsperspektive (1)

Erfüllung funktionaler Anforderungen

Technische Grundlagen

- **URL**: Uniform Resource Locator
  - ein *Uniform Resource Identifier* (URI) mit Ortsangabe
  - bezeichnet eine **Ressource**, häufig Hypertext
- **Hypertext**:
  - nicht-lineares Dokument, verknüpft durch **Hyperlinks**
- **HTML**: *Hypertext Markup Language*
  - Auszeichnungssprache um Hypertext zu verfassen
- **HTTP**: *Hypertext Transfer Protocol*
  - TCP-basiertes Protokoll zum Zugriff auf Ressourcen

Quelle: Intro, HTML, HTTP

HTTP

- **Request**/**Response**-Schema
- Format: Start Line, opt. **Header**, Leerzeile, opt. **Body**
- Anfrage, oder **Request**:
  - von Client an Host
  - **Zugriffsmethode** und Ressource in Start Line
- Antwort, oder **Response**:
  - vom Host an Client
  - **Statuscode** in Start Line
  - Ressourcen-Inhalte im Body
- Header: Schlüssel-Wert-Paare mit **Meta-Informationen**
- Request-**Parameter**:
  - Als Query-Teil der Ressource (“GET-Parameter”)
  - Im Body (“POST-Parameter”)

Quelle: HTTP

HTTP-Methoden & -Statuscodes

- Charakterisierung der verschiedenen HTTP-Methoden
  - **Sicher**: keine Nebeneffekte außer Auslieferung
  - **Idempotent**: Mehrfache Aufrufe haben gleiche Nebeneffekte wie einfacher Aufruf
  - (die anderen Methoden haben keinen speziellen Namen)
- Status-Code-Bereiche
  - `1xx`: Status-Informationen
  - `2xx`: Anfrage erfolgreich
  - `3xx`: Client muss aktiv werden (z.B. Umleitung)
  - `4xx`: Fehler auf Client-Seite
  - `5xx`: Fehler auf Server-Seite

Quelle: HTTP

Einige HTTP-Header-Felder

- Details zur Anfrage:
  - Request mit `Host`, `Accept`, `Accept-Langauge`
  - Response mit `Content-Type` und `Content-Length`
- Authentisierung: (durch Server initiert)
  - `Authorization` ← `WWW-Authenticate`
- Cookies: (durch Server initiert)
  - `Cookie` ← `Set-Cookie`
- Kompression: (durch Client initiert)
  - `Accept-Encoding` → `Content-Encoding`
- Caching:
  - in Request: `If-Modified-Since`
  - in Response: `Cache-Control`, `Age`, `Last-Modified`

Quelle: HTTP, Optimierungen (Caching & Kompression)

Vier Grundbausteine des Web

- Protokoll: **HTTP**
- Inhalte (aus Client-Sicht): HTML, CSS, JavaScript
  - **HTML**: Strukturiert Inhalte & verweist auf weitere Ressourcen
  - **CSS**: optische Gestaltung
  - **JavaScript**: Interaktion und Programmlogik beim Client
- Standardisierung:
  - HTML & CSS: W3C und WhatWG (“Living Standard”)
  - JavaScript: Ecma

Quelle: HTML, CSS, JavaScript

HTML

- Eigenschaften der Sprache bzw. von HTML-Code:
  - **deklarativ**, **wohlgeformt**, **valide**, **semantisch**
- Arbeitsweise des **Browsers**:
  - **Code mit Tags** → **DOM mit Elementen** → grafische Darstellung
- **Syntax** und Element-Arten:
  - Allgemein: **Dokumentstruktur**, Head und Body
  - Meta: **Verweise** auf andere Ressourcen, **Metadaten**
  - Semantische Elemente:
    - **Überschriften, Absätze, Verweise, Tabellen, Listen, Formulare**
    - HTML 5: Kopf- und Fußbereich, Haupt- und Nebenbereiche, Abschnitte, Navigationsmenüs
  - Optische und stilistische Elemente:
    - **Bildereinbindung, einfache Typografie**
    - HTML 5: Medieneinbindung (Audio und Video)
  - **Semantikfreie Elemente**: `div` und `span`

Quelle: HTML, Websites

CSS

- DOM-Knoten haben dutzende **Eigenschaften**
  - Formulierung von **Regeln** zum Setzen der Eigenschaftswerte
  - Auswahl von Knoten-Teilmenge durch **Selektoren**
    - Ausdruck von Knoten-Beziehung im DOM: **Kombinatoren**
- **Vererbungsschema**, Bestimmung der Werte pro Knoten:
  - kein Selektor passt: Erben von DOM-Vorfahren
  - genau eine Regel: anwenden
  - mehrere Regeln: **Kaskade**
    1. **Ursprung** der Regel (Dokument > Anwender > Browser, bzw. `!important`)
    2. **Spezifizität** der Selektoren
       - inline (keine Selektoren nötig) ist am spezifischsten
       - sonst: *ID* > *Klassen & Attribute* > *Elementnamen*
    3. Reihenfolge (letzte Regel gilt)
- Zentral für Layout: **Box-Modell**

Quelle: CSS

JavaScript

- Funktionsumfang:
  - Lesender und schreibender **Zugriff auf DOM**
  - Zugiff auf **Browser-APIs**
- **Ausführung** von JavaScript im Browser:
  - einmalig **beim Laden** des HTML-Dokuments
    - Funktionen definieren, Variablen und Event-Listener initialisieren
  - **Ereignisgetrieben**, basierend auf dutzenden DOM-Ereignissen
    - vorher registrierte Event-Listener werden aktiv
- **Event-Modell** mit zwei Phasen:
  1. **Capturing**: Top-Down, von Wurzel zum betreffenden Element
  2. **Bubbling**: Buttom-Up, von Element über Eltern zur Wurzel

Quelle: JavaScript

Arten von Ressourcen

- **statische** vs. **dynamische Ressourcen**
  - Server-Sicht: Was tun bei HTTP-Anfrage?
    - Dateiinhalte direkt von Festplatte ausliefern, oder
    - Programm muss Antwort vorverarbeiten oder generieren
  - für Client nicht unterscheidbar
- für ältere Websites:
  - meiste Ressourcen (HTML, CSS, JS) statisch
  - evtl. einige dynamisch (Kontaktformular: Anfrage mit Effekt)
- für viele Webanwendungen:
  - HTML-Ressourcen sind dynamisch (vom Backend generiert)
  - CSS und JavaScript statisch (Backend nicht involviert)
    - (ein Backend kann aber auch CSS & JS dynamisch erzeugen)

Quelle: Backend

Backend

- Muss HTTP implementieren, Sprache egal
  - über **CGI**: alles, was Text-Ausgaben erzeugen kann
  - meistens: webtaugliche Sprachen mit Webserver-Anbindung
- Weitverbreitete Sprachen für Backend-Entwicklung:
  - **PHP, Python, Ruby, Java, C#**, **Node.js** (serverseitiges JavaScript)
- **Auswahlkriterien** für Backendtechnologie:
  - (Lizenz-)Kosten, Hosting, Community, Bibliotheken & Frameworks, betrieblicher Kontext, Arbeitsmarkt
- **PHP**: Eigenschaften, Syntax und Standardbibliothek

Quelle: Backend, PHP

Backend-Frameworks

- Typische Architektur von Backend-Systemen:
  - inspiriert von **MVC**-Design-Pattern
- Bekannte Vertreter:
  - **Symfony** (PHP)
  - Ruby on Rails (Ruby)
  - Django (Python)
- Allgemeiner Kontrollfluss bei Anfrage:
  1. **Routing**: Mapping von HTTP-Anfragen
  2. **Controller**: zentrale Logik für Anfrage-Behandlung
  3. **Model**: Datenhaltung mit OR-Mapper
  4. **View**: Render-Logik, Erzeugen einer (HTML-)Ausgabe
- Leitspruch: *Konvention über Konfiguration*

Quelle: Backend, Symfony

Verschiedene Begriffe, gleiche Ideen

	Routing	Controller	Model	View
	Routing	Controller/Action	Entity	Template
	Routing	Controller/Action	Model	View
	URLconf	View	Model	Template

Weitere Framework-Funktionen

- (jeweils auf Basis von HTTP und HTML)
- **Formulare**
  - programmatische Erzeugung, inkl. HTML-Ausgabe
  - Validierung von Eingaben, Fehlerbehandlung
  - Abbildung auf Daten-Modell
- **Rechte-Verwaltung**
  - Authentisierung & Autorisierung (Rollen & Attribute)
- **Session-Verwaltung**
  - Clients wiedererkennen
  - längere Interaktionen als Anfrage/Antwort
- **Internationalisierung**
  - Mehrsprachigkeit

Quelle: Symfony Überblick und Vertiefung

WebServices

- **WebServices** für zwei **Szenarien**:
  - Backend für weitere Clients öffnen
  - Nutzung fremder Dienste
- Zwei Architekturen: **XML** und **REST**
- **AJAX** im Frontend für **vielgestaltige Interaktion**

Quelle: WebServices & AJAX

Anwendungsperspektive (2)

Erfüllung nicht-funktionaler Anforderungen

Informationssicherheit

- **Schutzziele**
  - Vertraulichkeit, Integrität, Authentizität, Verfügbarkeit
- für Web-Entwicklung relevante **Angriffsmodelle**
  - MITM, XSS, CSRF, Session-Hijacking, SQL-Injection
- und **Gegenmaßnahmen**
  - Vertrauensmodell
  - Verschlüsselung, Ein- und Ausgabe-Filterung, Tokens

Quelle: Security

Performance und Usability

- **Optimierungen**
  - **Zusammenhang:** Browser-Funktionsweise ↔ Ladezeiten
    - **Kritischer Request-Pfad**
    - Bedeutung von Request-Zahlen und Datenvolumen
  - Strategien zur Minimierung von **Request-Zahlen**
    - HTTP-Caching: Modelle **Expiration** & **Validation**
    - **Bündeln** von Ressourcen: Grafiken, CSS und JavaScript
  - Strategien zur Minimierung von **Datenvolumen**
    - **Minify** für Textformate
    - Komprimierung allen Payloads
  - **Betriebswirtschaftliche Perspektive**
- **Responsiveness**:
  - Anpassung der Darstellung auf Endgerät

Quelle: Optimierung, Website-Gestaltung

Technisch-betriebliche Perspektive

Praktiken guter Web-Entwicklung

Trennung von Belangen

- **Templates:**
  - Vorlagen für textuelle Ausgaben (meist HTML)
  - haben Platzhalter für Anzeige von Variablen
    - (Herkunft der Daten spielt keine Rolle)
  - minimale Logik (Schleifen, Bedingungen)
  - Vererbung und Fragment-Einbindung
- **OR-Mapper:**
  - abstrahieren von Tabellen, Queries auf Objekten
  - erlauben Objekt-Orientierte Geschäftslogik
  - verschiedene Datenbanken für Entwicklung und Betrieb

Quelle: PHP-Komponenten

Wiederverwendung

- **Wiederverwendung** von Komponenten
  - PHP: Abhängigkeitsverwaltung mit *Composer*
  - JavaScript: ditto mit *NPM* bzw. *Yarn*
  - Allgemeine Funktionsweise:
    - Abhängigkeiten selbst nicht unter Versionskontrolle, sondern
    - nur *Verweise* und Versionsbereiche definiert in JSON-Datei
    - (konkrete Versionen ermittelt und hinterlegt in LOCK-Datei)
    - Herunterladen in aktuelle Umgebung mit *Install*-Befehl
- **Wiederverwendung** von Bibliotheken
  - *jQuery* für JavaScript, *Bootstrap* für CSS

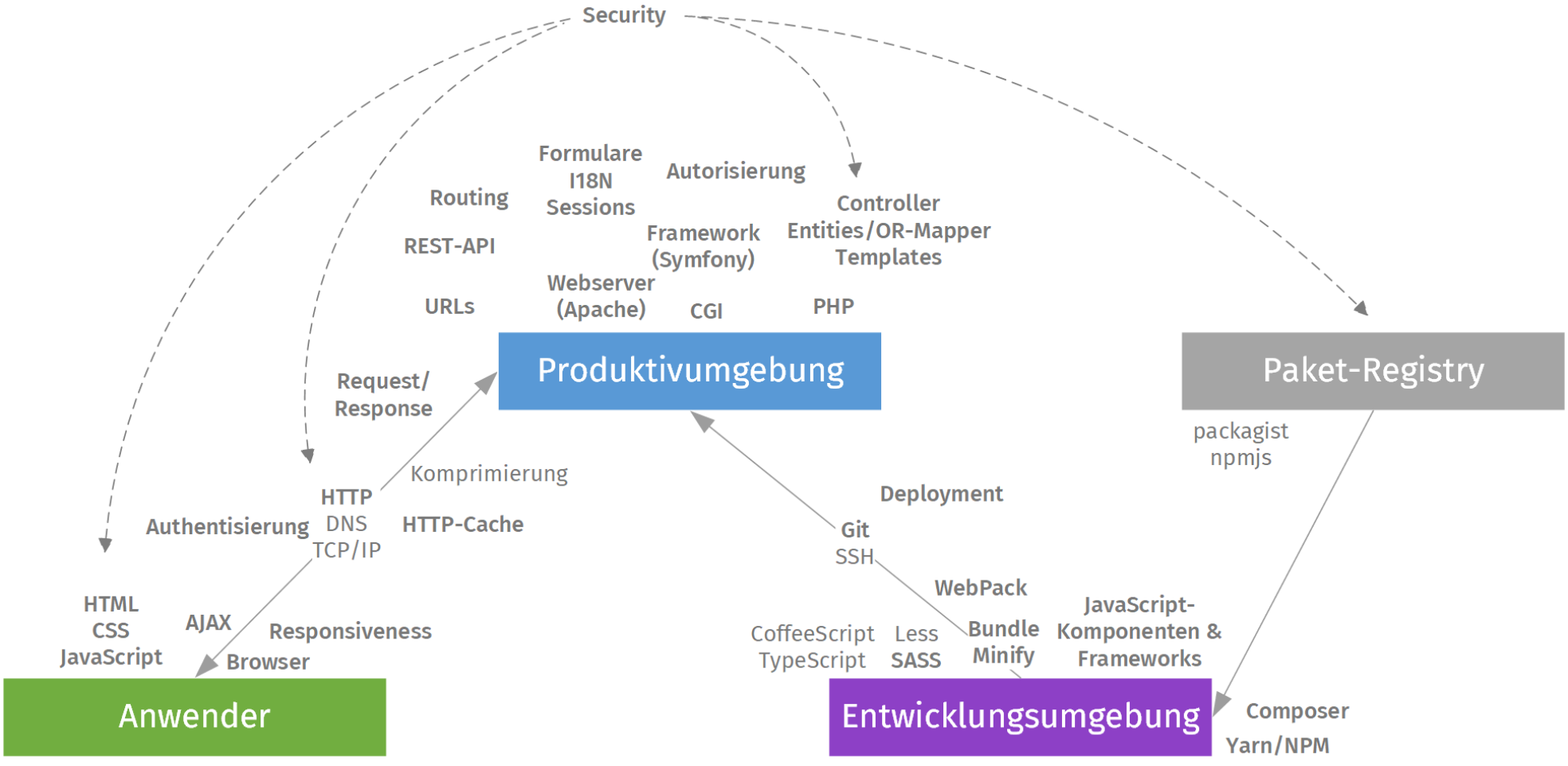
Quelle: PHP-Komponenten, JavaScript, Website-Gestaltung, JS-Komponenten

Entwicklungseffizienz

- Spracherweiterungen über Standards hinaus
  - **Besseres CSS** (SASS/Less/PostCSS)
  - **Besseres JavaScript** (CoffeeScript/TypeScript)
- Komponenten-basierte Entwicklung in JavaScript
  - ähnlich wie im Backend
  - allerdings Vorbereitung für Browser nötig
- Frontend-Frameworks
  - Kontrolle über DOM-Teil durch JavaScript-Objekt
  - One-Way und Two-Day **Data Binding**

Quelle: Moderne Frontend-Entwicklung

Mentales Modell



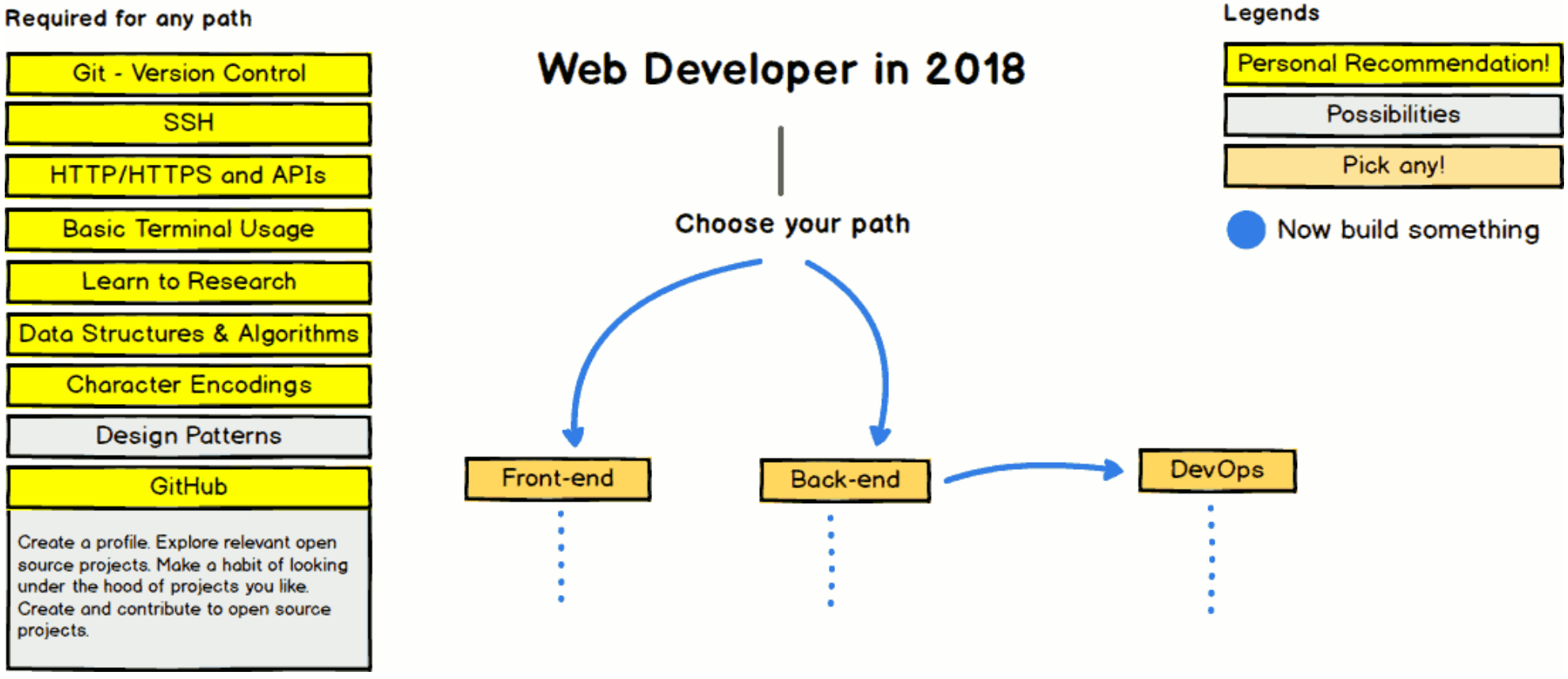
Ende

des Schnelldurchgangs

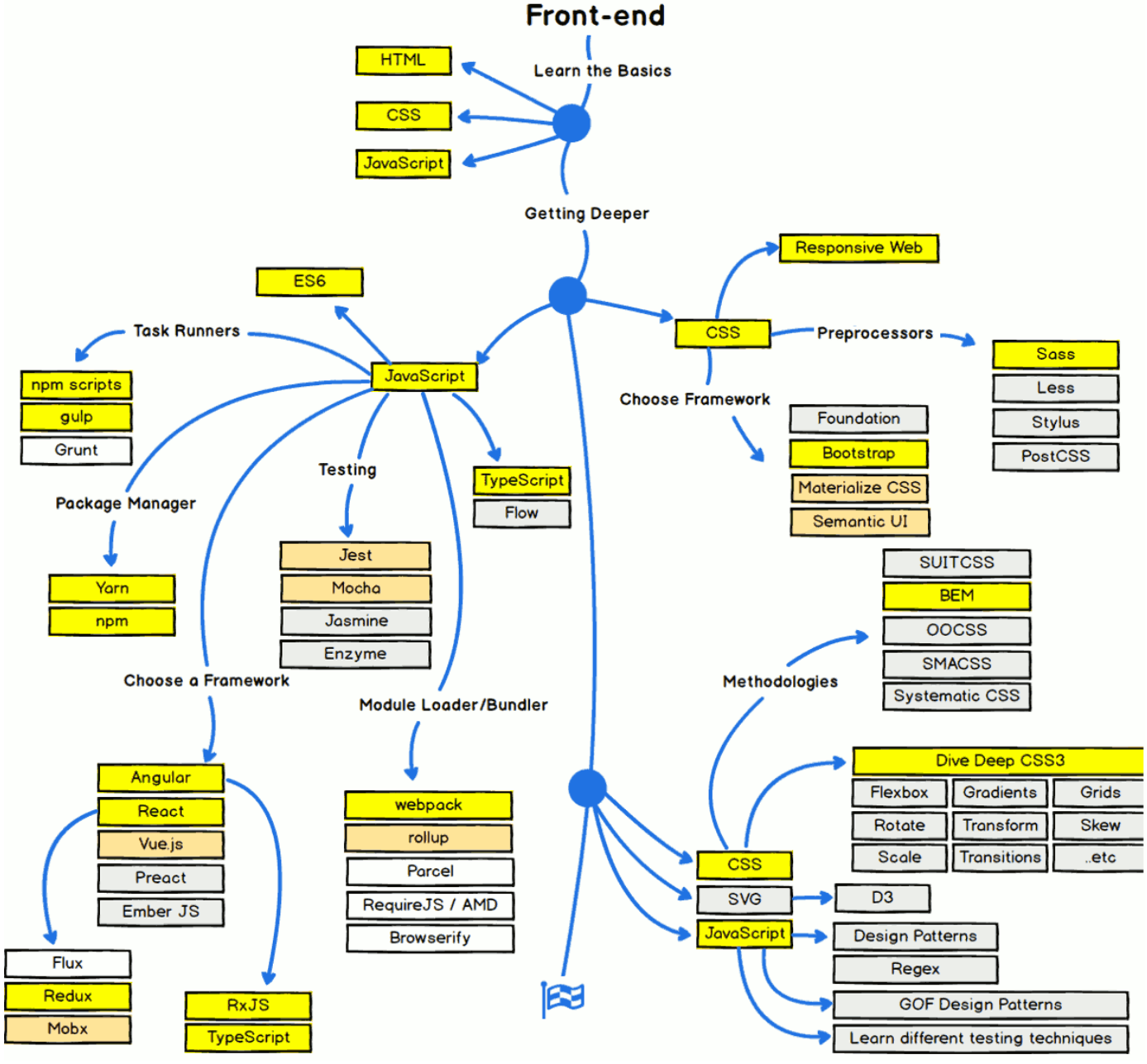
Einige nicht behandelte Themen

- Technologien:
  - Multimedia: Audio, Video und Grafiken
  - *WebSockets*: bidirektionale TCP-Verbindung
  - *WebAssembly*: Maschinen-Code für den Browser
- Webdesign: Gestalterische Aspekte
  - inkl. Mobile Web Design: besondere Anforderungen
- Integration mit anderen Systemen:
  - z.B. Mailing, Authentisierung (OAuth), GraphQL
- Strukturierung größerer Anwendungen
- “Containerisierung” mit *Docker*

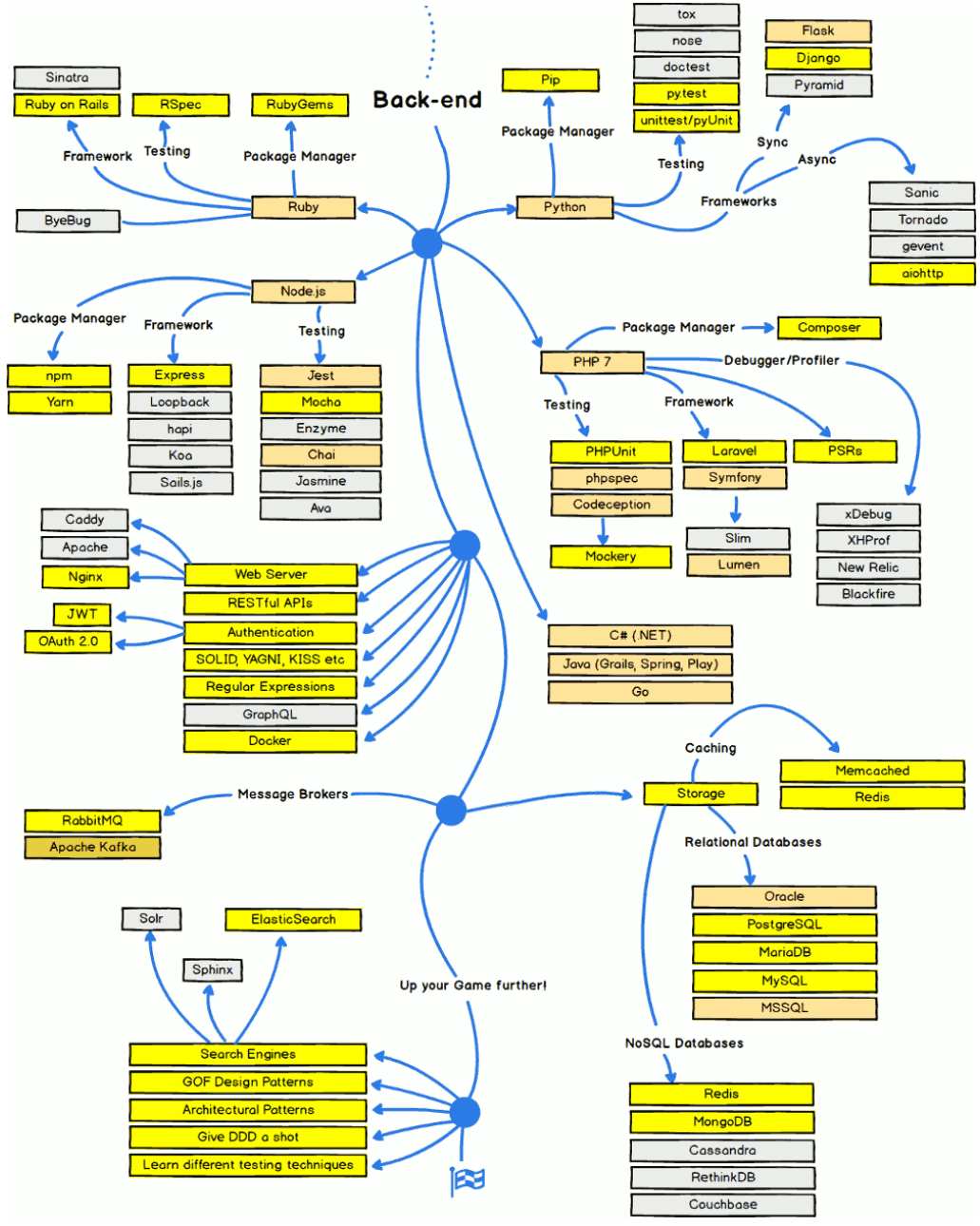
Web Developer Roadmap 2018



Quelle: https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d



Quelle: https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d



Quelle: https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d

Ask me Anything

- Mein Hintergrund:
  - Websites (HTML und CSS): **seit 2002**
  - Webentwicklung (PHP 4): **seit 2004**
  - professionell (wenn auch nur nebenberuflich): **seit 2006**
- Diese Veranstaltung: Ein Einstieg und Überblick
  - (zugegebenermaßen ein recht gründlicher)

Was wollten Sie schon immer mal wissen?

Warme Worte zum Schluss

1. Behalten Sie einen kühlen Kopf, und lassen sich nicht von der Neuigkeit von Technologien blenden.
2. Lernen Sie nie aus. Experimentierfreude und Spieltrieb gehören zu unserem Beruf.
   - *“Hierzulande musst du so schnell rennen, wie du kannst, wenn du am gleichen Fleck bleiben willst.”* (Lewis Caroll)
3. Sorgen Sie dafür, dass *Sie* Ihre Werkzeuge beherrschen.
   - IDE, Versionskontrolle, Kommandozeile, Debugging-Tools
4. Arbeiten Sie in kleinen Schritten mit je einem Anliegen.
5. Lesen Sie Fehlermeldungen.

Klausuren

- siehe: https://zieris.net/webdev/klausur
- Nächste Woche: Mittwoch, **31.01.2018**
  - Raum: **C 348**
  - Beginn: **12:15 Uhr**, spätestens 10 Minuten vorher da sein
  - Studierendenausweis nicht vergessen
  - keine Hilfsmittel (außer Deutsch-Wörterbuch)
- 2. Prüfungszeitraum: Montag, **19.03.2018**
  - Raum: **C 353**
  - Beginn: **12:15 Uhr**, spätestens 10 Minuten vorher da sein

Danke!