VISUAL REGRESSION TESTING MIT

# BACKSTOPJS

# WIESO WESHALB WARUM

- Manuelles Testing
  - nervt
  - kostet viel Zeit
  - hängt davon ab, wer es durchführt
  - wird selten effizienter, je häufiger man es durchführt
  - hat den „ich weiß wo ich klicken muss Effekt“

# MEIN USECASES

- Umfangreiche TYPO3 Updates prüfen, insb. übernommene Projekte
  - Sichtbare Exceptions finden
  - Fehlende ViewHelper erkennen
  - Verschiedene Auflösungen testen
- Datenbankmigrationen testen: sieht es noch wie vorher aus?
- Deployments testen (derzeit noch mit VisualCeption umgesetzt)

# VISUAL REGRESSION TESTING IN KURZ

- Url aufrufen und Screenshot der Seite / von definierten Elementen anfertigen
- Code ändern
- Url erneut aufrufen, gleichen Screenshot anfertigen
- Screenshots vergleichen
- Abweichung > Threshold? ->  Fehler

# BISLANG AUSPROBIERT / VERWENDET

- Wraith: https://github.com/bbc/wraith
  nur auf den ersten Blick einfach… Probleme mit Ruby Updates und Gems traten häufig auf
- PhantomCSS / CasperJS: seit 2017 nicht mehr maintained https://github.com/HuddleEng/PhantomCSS
- VisualCeption / Codeception: braucht Selenium / WebDriver, Setup deutlich aufwendiger… https://github.com/Codeception/VisualCeption
  ***WARNING*** *This module can reduce the execution speed of acceptance tests. Use it only for visual regression test suite and not for regular end to end testing.*

# BACKSTOPJS

- Braucht kein Selenium Setup!
- Kann mit lokalen Browsern und Docker umgehen
- Hat einen Desktop und einen CI Modus
- Kann Puppeteer (Chrome https://github.com/puppeteer/puppeteer) und Playwright (Chrome, Edge, Firefox, Safari) verwenden https://playwright.dev/
- Hat sogar einen Udemy Kurs https://www.udemy.com/course/testes-de-regressao-visual-com-backstopjs/ (Spanisch)

# QUICKSTART

npm install -g backstopjs
backstop init
*backstop.json anpassen*
backstop reference
backstop test

# UND JETZT IN 10 MINUTEN

- backstop.json ohne scenarios + engine Skripte kopieren
- CSV mit URLs erzeugen
- backstop reference --config="backstop-sitemap.js"
  - https://gist.github.com/akiessling/f57bdf10b9f90431a8aa502db437de2f
- Die Config kann dynamisch als JavaScript-Modul erzeugt werden. Für noch mehr Flexibilität kann backstop auch in eigenen NodeJS Code eingebaut werden

# URLS.CSV ERZEUGEN? EASY :) LADEN UND FILTERN

- php sitemap-parser.php https://www.dmk-ebusiness.de/sitemap.xml
  cat urls.txt | grep "dmk-ebusiness.de" | sort | uniq > filtered.csv
- https://github.com/akiessling/crawler
  ./crawler crawl https://www.example.org  crawlresult.csv
  csvgrep -c status -m 200 crawlresult.csv | csvsort -c url | csvcut -c url | tail -n +2 > urls.csv
- curl https://www.dmk-ebusiness.de/sitemap.xml\?sitemap\=pages\&cHash\=d5af85692b989e540e2b992173491259 -s | xmlstarlet sel -t -v '//*[local-name()="loc"]'  |sed 's/ *//g’
- Oder für Spezialfälle über TypoScript: https://gist.github.com/akiessling/f57bdf10b9f90431a8aa502db437de2f#file-urls-typoscript

# WO WIRD ES TRICKY

- Jede Seite ist unterschiedlich:
  - lazy / async Inhalte, loading=„lazy“, Cookie Consent (für alle Domains freischalten!), Inhalte nach Interaktion anzeigen. Mögliche Lösung: https://gist.github.com/akiessling/f57bdf10b9f90431a8aa502db437de2f#file-onready-js
- Animationen / Content die durch Scrolling getriggert werden (https://github.com/garris/BackstopJS/issues/1111), Grafiken mit object-fit
- Docker Runner mit Apple Silicon https://github.com/garris/BackstopJS/issues/1300
- Nur fehlgeschlagene Scenarios wiederholen https://github.com/garris/BackstopJS/issues/639 –> JSON Result von *backstop test* parsen ggf. Lösung
- Ggf. CSP Probleme weil eigener JS Code injected wird
- Tausende Seiten am Stück vergleichen… –> ~bei 3000 war dann Schluss auf meinem System…

# NOCH MEHR TOOLS

- https://github.com/akiessling/backstopjs-demo
- https://gist.github.com/akiessling/f57bdf10b9f90431a8aa502db437de2f
- https://github.com/garris/BackstopJS
- https://github.com/mojoaxel/awesome-regression-testing