

Wie installiere ich den Matomo-Tracking-Code auf WordPress?

Wie installiere ich den Matomo-Tracking-Code auf WordPress?

Es ist ganz einfach, Matomo-Tracking zu Ihrer WordPress-Site hinzuzufügen. Sie können entweder das einfache WordPress-Plugin verwenden (mit den folgenden Schritten), den JavaScript-Code installieren ([mit dieser Anleitung](#)) oder einen Image-Beacon installieren ([mit dieser Anleitung](#)).

Anforderungen

- Ein Matomo-Konto: [Cloud](#) oder [On-Premise \(selbst gehostet\)](#) .
- Administratorkonto in WordPress
- [WordPress.com](#)- Site mit einem Business-, Commerce- oder Enterprise-Plan oder eine [selbst gehostete](#) WordPress-Site.

Schritte zum Starten des Trackings in Matomo

1. Installieren Sie das Plugin „WP-Matomo“ in Ihrem WordPress.
 - Melden Sie sich als Administrator bei Ihrer

WordPress-Site an.

- Klicken Sie im linken Menü auf „Plugins“ > „Neu hinzufügen“ (wenn Sie das Menü „Plugins“ nicht sehen können, können Sie stattdessen [diese Schritte befolgen, um den Matomo-Tracking-Code manuell in Ihre WordPress-Theme-Dateien einzufügen](#)).
- Suchen Sie im Feld „Keyword“ auf der Seite „Plugins hinzufügen“ nach „Connect Matomo“.
- Klicken Sie neben dem Plugin „WP-Matomo“ auf „Jetzt installieren“.
- Nachdem das Plugin nun erfolgreich installiert wurde, klicken Sie auf „Aktivieren“.

2. Klicken Sie im linken Menü auf „Einstellungen“ > „WP-Matomo“.
3. Wählen Sie im Dropdown-Menü „Matomo-Modus“ die Option „Selbst gehostet (HTTP-API, Standard)“ aus.
4. Geben Sie im Textfeld „Matomo-URL“ beispielsweise Ihre Matomo-URL ein <https://analytics.example.com>.
5. Geben Sie im Textfeld „Auth-Token“ Ihr Matomo-Authentifizierungstoken ein. [So finden Sie das Auth-Token](#) .
6. Bestätigen Sie, dass das Kontrollkästchen „Automatische Konfiguration“ aktiviert ist, und klicken Sie auf „Änderungen speichern“.
7. Sie sollten die Meldung „WP-Matomo wurde erfolgreich mit Matomo verbunden“ sehen. Wenn die Meldung nicht angezeigt wird, stellen Sie sicher, dass die in den vorherigen Schritten angegebenen Werte korrekt sind, und versuchen Sie es erneut.
8. Klicken Sie auf die Registerkarte „Tracking aktivieren“.
9. Wählen Sie im Dropdown-Menü „Tracking-Code hinzufügen“ die Option „Standard-Tracking“ aus.
10. Klicken Sie auf „Änderungen speichern“ und eine Erfolgsmeldung wird angezeigt.

Herzlichen Glückwunsch, Sie sollten jetzt erfolgreich Besucher mit Matomo Analytics über Ihre WordPress-Site verfolgen! Viel Spaß bei der Analyse.

Deutsches Whistleblower-Gesetz tritt in Kraft

Ein neues Gesetz soll Hinweisgeber vor Repressalien und Vergeltungsmaßnahmen schützen. Es verpflichtet außerdem Unternehmen, technische Systeme und Prozesse für solche Hinweise einzuführen.

[Mehr lesen ...](#)

Cyberisiko-Check für KKUs nach DIN

Kleine Unternehmen sind genauso von Cyberangriffen bedroht wie die großen, haben häufig aber weder Mittel noch speziell geschultes Personal, um sich zu schützen. Ein neuer Standard soll ihnen zu einer Einschätzung der Risiken und entsprechenden Schutzmaßnahmen verhelfen.

[Mehr lesen ...](#)

KI-Regulierung nimmt international Konturen an

Die Grundlagen des europäischen KI-Gesetzes stehen, in Einzelfragen dürften die verantwortlichen EU-Gremien in nächster Zeit noch nachjustieren.

[Mehr lesen ...](#)

Einheitsstrafen DSGVO

Neue europäische Regeln für DSGVO-Bußgelder

Erstmals haben sich die europäischen Datenschutzbehörden auf gemeinsame Grundsätze geeinigt, nach denen Bußgelder zu verhängen sind. Diese lassen den Behörden in den einzelnen Ländern aber nach wie vor große Spielräume.

Von Joerg Heidrich

Link zu den EDSA-Guidelines: ct.de/yfyw

[Mehr lesen ...](#)

Google Analytics 4 & DSGVO: Ihre Datenschutz-Checkliste für die Umstellung am 1. Juli 2023

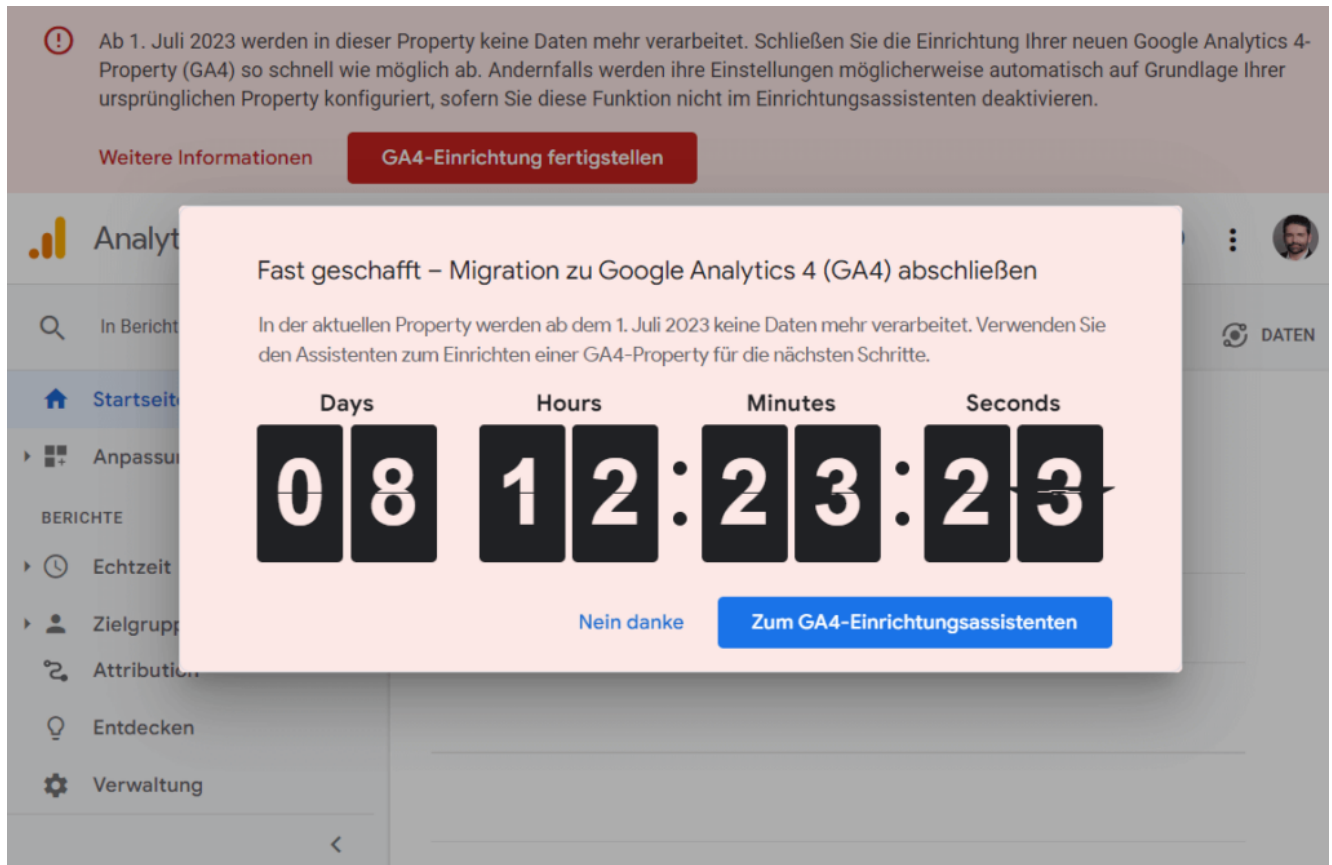
Ab dem 1. Juli 2023 stellt Google seinen Dienst Google Analytics dauerhaft auf die neue Version "Google Analytics 4" (kurz "GA4") um. Mit dieser Änderung kommen nicht nur technische, sondern auch rechtliche Veränderungen auf Nutzer von Google Analytics zu.

In diesem Beitrag erfahren Sie, was sich datenschutzrechtlich ändert, ob und wie Sie Google Analytics 4 einsetzen dürfen. Am Ende des Beitrags erhalten Sie eine Checkliste mit den wichtigsten Prüfpunkten vor dem Einsatz von Google Analytics 4.

Wie erfolgt die Umstellung auf Google Analytics 4?

Sollten Sie bisher noch keine eigenen Schritte zur Umstellung unternommen haben, so hat Google bereits im März 2023 automatisch Google Analytics 4 für Sie eingerichtet, basierend auf Ihren bisherigen Einstellungen und Zielen für Universal Analytics.

Bis Ende Juni 2023 haben Sie noch die Möglichkeit, Daten in der alten Version (Universal Analytics) zu sammeln und zu nutzen. Danach beginnt eine Übergangszeit von sechs Monaten, in der Sie auf alte Daten zugreifen können. Ab dann müssen Sie Google Analytics 4 einsetzen.



Was ist technisch neu an Google Analytics 4?

Universal Analytics und Analytics 4 sind beide Versionen von Googles Analysetool, jedoch mit technischen Unterschieden.

Universal Analytics fokussiert sich auf Sitzungen und Nutzerinteraktionen auf einzelnen Geräten. Im Gegensatz dazu konzentriert sich Analytics 4 auf die Nachverfolgung von Nutzern über verschiedene Geräte hinweg und ermöglicht eine detailliertere Erfassung des Nutzerverhaltens durch sogenannte "Events", was tiefergehende Einblicke in ihr Verhalten bietet.

Kurzum, Analytics 4 ist die neuere Version, die detailliertere und geräteübergreifende Daten liefert. Umgekehrt hat Google aber auch die Datenschutzaspekte verbessert.

Versionsbezeichnung: In der Praxis verwendet man die Versionsbezeichnungen selten, sondern spricht schlicht von "Google Analytics". Ab Juli 2023 ist damit automatisch Google

Analytics 4 gemeint.

Ist Google Analytics 4 datenschutzrechtlich sicherer?

Google Analytics 4 erleichtert zwar die Nachverfolgung des Nutzerverhaltens, bietet aber zugleich neue Datenschutzfunktionen:

- **Cookie-los:** Mit Google Analytics 4 können Tracking-Verfahren eingerichtet werden, die ohne Cookies auskommen.
- **Serverseitiges Tracking:** Nutzerdaten können auf dem eigenen Server pseudonymisiert werden, bevor sie an Google übermittelt werden.
- **Standort EU:** Google zufolge werden alle Daten von Endgeräten in der EU auf Servern innerhalb der EU gespeichert und verarbeitet.
- **IP-Kürzung in der EU:** Im Gegensatz zu Universal Analytics erfolgt die Kürzung der IP-Adressen der Nutzer auf EU-Servern von Google. Das bedeutet, dass ein entsprechend pseudonymer Datensatz in die USA übermittelt wird.
- **Kürzere Aufbewahrungsfristen:** Im Gegensatz zur bisherigen Standardlöschfrist von 26 Monaten bei Universal Analytics erlaubt GA4 keine längere Aufbewahrung als 14 Monate für Daten.
- **Google Signals:** Durch die [Deaktivierung von Google Signals](#) kann die Verknüpfung mit einem Google-Konto verhindert werden.
- **Geo- und Geräteinformationen:** Die Genauigkeit der gesammelten Geo- und Geräteinformationen kann angepasst werden.

Die genannten Maßnahmen tragen zur Verbesserung des Datenschutzniveaus bei der Verwendung von Google Analytics 4

bei. Beachten Sie jedoch, dass die Nutzung von Google Analytics damit nicht automatisch zulässig und rechtssicher wird.

Daten aus Social Media automatisiert herunterladen

Das Automatisieren von Downloads spielt bei der Auswertung von Daten eine große Rolle. Oftmals liegen riesige Datenmengen vor, die allerdings auf mehrere Dateien aufgeteilt wurden. Das betrifft vor allem Daten, die von diversen Diensten wie einem Fahrradverleih zur Verfügung gestellt werden. Andererseits sind Daten von sozialen Medien gefragt, anhand derer sich beispielsweise erkennen lässt, wie beliebt ein Post, Tweet oder Video ist.

Für etliche Anbieter von sozialen Medien existieren bereits Bibliotheken, die Zugriff auf das API eines sozialen Netzwerks ermöglichen [1]. Bei diversen Downloadproblemen ist allerdings die Verwendung der allgemeineren Requests-Bibliothek erforderlich [2]. Requests beschäftigt sich nämlich mit den *POST*- und *GET*-Anfragen, die an HTTP-Server übermittelt werden. Anschließend kann die gewünschte Seite oder Datei mittels geeigneter Funktion heruntergeladen werden (**Abb. 1**).

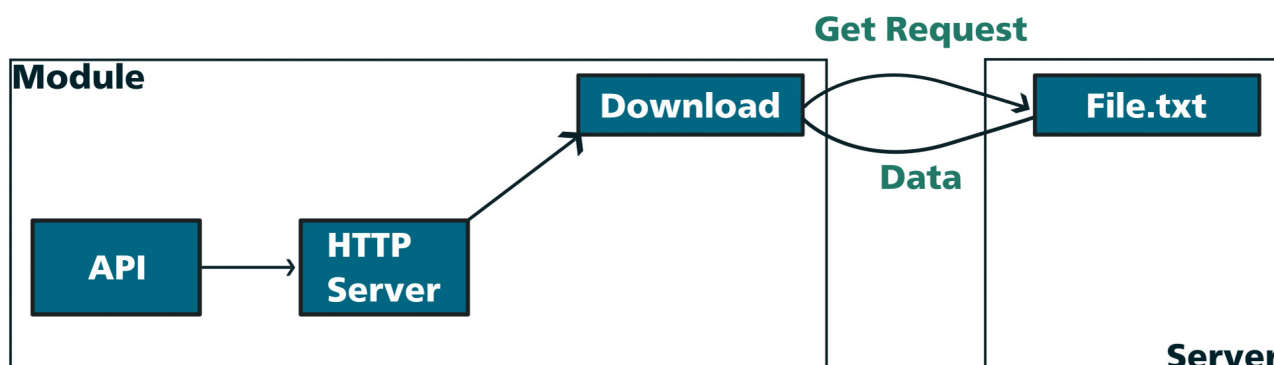


Abb. 1: Requests-Bibliothek

Alles in allem belegen diverse Statistiken, dass das Datenvolumen exponentiell wächst, womit es für Anwender unausweichlich wird, das Herunterladen von Daten beziehungsweise den Zugriff auf Daten zu automatisieren [3].

Soziale Medien

Üblicherweise platzieren Unternehmen beziehungsweise Organisationen Links zu den sozialen Netzwerken auf der eigenen Webseite, was vor allem auf die Kontaktseite zutrifft. So befinden sich die Icons der bekannten Anbieter von sozialen Medien eingebettet auf der Kontaktseite. Andererseits ist es üblich, die Icons im Footer einer x-beliebigen Seite der Organisation zu platzieren. Der dafür erforderliche HTML-Code, um das Icon eines Anbieters für soziale Medien auszugeben, könnte wie folgt aussehen:

```
<a                                target="_blank"
href="https://www.facebook.com/GradeSaverLLC">
    
</a>
```

Sobald ein Anwender auf eines dieser Icons klickt, leitet der Browser den Anwender auf die Social-Media-Seite der Organisation weiter. Die Reichweite des Links umfasst dabei das komplette Social-Media-Icon.

Um Links aus Webseiten extrahieren zu können, eignet sich das Paket „Extract Social Media“ in der Version 0.4.0. Zusätzlich ist der Einsatz des Requests-Pakets erforderlich. Mit pip lassen sich die Pakete wie folgt installieren [4]:

```
pip install extract-social-media
pip install requests
```

Wird nun ein URL der selbstdefinierten Methode *get_links* übergeben, parst sie die Webseite nach möglichen Links und gibt diese aus (Listing 1).

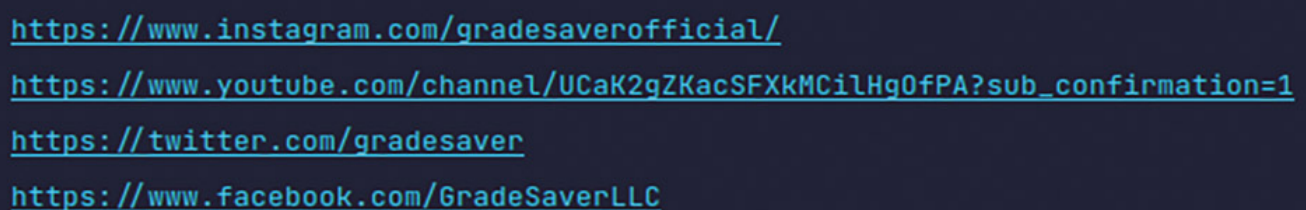
Listing 1

```
import requests
from extract_social_media import find_links_tree
from html_to_etree import parse_html_bytes

def get_links(url):
    res = requests.get(url)
    tree = parse_html_bytes(res.content,
res.headers.get('content-type'))
    link_set = set(find_links_tree(tree))
    for s in link_set:
        print(s)

get_links('https://www.gradesaver.com/contact')
```

In der Methode *get_links* wird zunächst eine *GET*-Anfrage *requests.get* an die Webseite gesendet. Der daraufhin übermittelte Inhalt der Webseite wird in der Variable *res* gespeichert. Anschließend fischt die Methode *find_links_tree* die Links heraus und speichert sie in einem Set ab (**Abb. 2**).



```
https://www.instagram.com/gradesaverofficial/
https://www.youtube.com/channel/UCaK2gZKacSFXkMCilHg0fPA?sub\_confirmation=1
https://twitter.com/gradesaver
https://www.facebook.com/GradeSaverLLC
```

Abb. 2: Extrahierte Links einer Webseite

Dateidownload

Das Herunterladen von Dateien lässt sich unter Python in mehreren Schritten bewerkstelligen, wobei das Python-Skript davon ausgeht, dass die Links zu den Dateien bereits vorhanden sind. Vor allem die Auswertung von Dateien wird durch den automatisierten Download erleichtert, da sich heruntergeladene

Text- oder CSV-Dateien im Anschluss in einen Dataframe importieren lassen. Für den Download mit Python werden die Importe aus Listing 2 gebraucht.

Listing 2

```
import numpy as np
import pandas as pd
import requests
import zipfile
import os
import glob
```

Das Python-Skript legt zunächst einen Ordner an, um die Dateien dort abzuspeichern. Der Ordner wird lediglich dann erstellt, wenn er noch nicht existiert:

```
def make_dir(folder):
    if not os.path.exists(folder):
        os.makedirs(folder)
```

Beim anschließenden Herunterladen der Dateien wird durch eine Liste mit Links *links* iteriert, wobei erneut von der Requests-Bibliothek in Version 2.28.2 Gebrauch gemacht wird (Listing 3). Um den Fortschritt des Downloads anzuzeigen, wird auf Python-Bordmittel zurückgegriffen. So wird der Index des aktuellen Links ermittelt, wobei die Anzeige zusätzlich die Gesamtzahl der Links zusammen mit der Antwort des Servers beinhaltet. In der Produktion sollte die *GET*-Anfrage *requests.get* unter Berücksichtigung eines Timeout ausgegeben werden. Dadurch wartet das Python-Skript lediglich für eine bestimmte Zeit in Sekunden auf eine Antwort des Servers, um anschließend weiterzumachen. Ansonsten besteht die Gefahr, dass sich das Programm aufhängt [5]. Mit den letzten zwei Zeilen werden die Dateien im Ordner abgespeichert (**Abb. 3**).

Listing 3

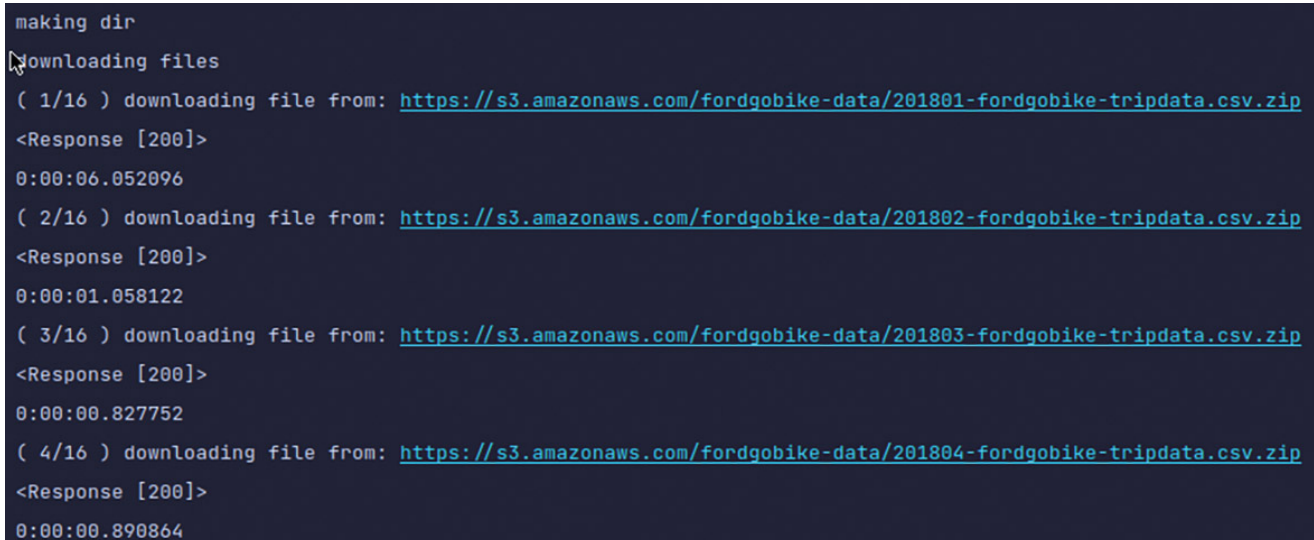
```
# downloads all zip files that are mentioned in the list:
def download(links, folder):
    size = len(links)
```

```

for l in links:
    i = links.index(l)
    msg = '( '+str(i+1)+'/'+str(size)+' ) '+'downloading file
from: '+l
    response = requests.get(l,timeout=30)
    print(msg)
    print(response)
    print(response.elapsed)

    with open(os.path.join(folder, l.split('/')[-1]),
mode='wb') as file:
        file.write(response.content)

```



```

making dir
downloading files
( 1/16 ) downloading file from: https://s3.amazonaws.com/fordgobike-data/201801-fordgobike-tripdata.csv.zip
<Response [200]>
0:00:06.052096
( 2/16 ) downloading file from: https://s3.amazonaws.com/fordgobike-data/201802-fordgobike-tripdata.csv.zip
<Response [200]>
0:00:01.058122
( 3/16 ) downloading file from: https://s3.amazonaws.com/fordgobike-data/201803-fordgobike-tripdata.csv.zip
<Response [200]>
0:00:00.827752
( 4/16 ) downloading file from: https://s3.amazonaws.com/fordgobike-data/201804-fordgobike-tripdata.csv.zip
<Response [200]>
0:00:00.890864

```

Abb. 3: Fortschrittsanzeige der Downloads

Das Programm könnte mit Listing 3 vorbei sein. Allerdings lässt sich das Programm noch weiter ausbauen, indem beispielsweise heruntergeladene ZIP-Archive automatisch entpackt werden (Listing 4). Die Methode `extract` erledigt genau das, indem sie zusätzlich die entpackten Dateien an einem beliebigen Ort auf der Festplatte ablegt.

Listing 4

```

# Extracts all contents from zip file
def extract(folder):
    all_files = glob.glob(folder + "/*.zip")
    archive = folder + '/' + working_path
    for f in all_files:
        with zipfile.ZipFile(f, 'r') as myzip:

```

```
myzip.extractall(path=folder)
```

Handelt es sich bei den heruntergeladenen Dateien um Datenreihen, die beispielsweise als Textdatei oder im CSV-Format vorliegen, dann bietet es sich an, diese Daten in einem Dataframe zu importieren (Listing 5). So iteriert die Methode *merge* durch alle Dateien vom Typ CSV. Danach werden die Datenreihen schrittweise in einen einzigen Dataframe geladen.

Listing 5

```
# merges all csv files into one dataframe
def merge(folder):
    all_files = glob.glob(folder + "/*.csv")
    li = []
    for filename in all_files:
        df = pd.read_csv(filename, index_col=None, header=0)
        li.append(df)
    fordgobike = pd.concat(li, axis=0, ignore_index=True)
    # displays the columns and their datatypes
    fordgobike.info()
```

Listing 6 beherbergt die Testdaten. So können Sie dort entnehmen, wie die Liste mit den Links definiert worden ist und wie sich die einzelnen Methoden aufrufen lassen (vgl. *test_files_download*).

Listing 6

```
folder_name = 'fordgobike'
working_path = 'archive'
# urls of zip files
urls =
['https://s3.amazonaws.com/fordgobike-data/201801-fordgobike-tripdata.csv.zip',
 'https://s3.amazonaws.com/fordgobike-data/201802-fordgobike-tripdata.csv.zip',
 'https://s3.amazonaws.com/fordgobike-data/201803-fordgobike-tripdata.csv.zip',
 'https://s3.amazonaws.com/fordgobike-data/201804-fordgobike-tripdata.csv.zip',
 'https://s3.amazonaws.com/fordgobike-data/201805-fordgobike-tripdata.csv.zip']
```

```

ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201806-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201807-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201808-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201809-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201810-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201811-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201812-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201901-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201902-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201903-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201904-fordgobike-tr
ipdata.csv.zip']
def test_files_download():
    print('making dir')
    make_dir(folder_name)
    print('downloading files')
    download(urls, folder_name)
    print('extracting files')
    extract(folder_name)
    print('merging files into to dataframe')
    merge(folder_name)

```

Vimeo-Bot

Vimeo ist ein beliebter Videodienst. Die gute Nachricht ist, dass es bereits eine Bibliothek gibt, die das Vimeo-API implementiert [6]. In Version 0.4.1 lässt sich die Bibliothek Vimeo Downloader wie folgt installieren:

```

pip install vimeo-downloader

```

Zusätzlich braucht Ihr Programm den folgenden Import, um auf die Methoden und Attribute des Vimeo-API zuzugreifen:

```
from vimeo_downloader import Vimeo
```

Um mehr über ein Video zu erfahren, können Sie zunächst die Metadaten eines Videos abrufen (Listing 7). Sobald ein Objekt vom Typ *Vimeo* durch Übergabe eines Links initialisiert wird, lassen sich der Titel sowie die Zahl der Likes und Views abrufen. Zusätzlich können Sie alle Kategorien des Metaobjekts anzeigen lassen (**Abb. 4**).

Listing 7

```
def print_info(url):
    v = Vimeo(url)
    meta = v.metadata
    print('Title:', meta.title)
    print('Number of likes: ', meta.likes)
    print('Number of views: ', meta.views)
    print(meta._fields)
    return v
```



```
Title: The Science Catalyst Live Show:Chemistry Experiment
Number of likes: 1
Number of views: 1242
('id', 'title', 'description', 'url', 'upload_date', 'thumbnail_small', 'thumbnail_medium', 'thumbnail_large', 'user_id', 'user_name', 'user_url', 'user_portrait_small', 'user_portrait_medium', 'user_portrait_large', 'user_por
```

Abb. 4: Metadaten eines Vimeo-Videos

Damit Sie nun einen Vimeo-Bot generieren können, ist es lohnenswert, den Bot auf eine Liste von Links anzuwenden. So initialisiert die Schleife jedes Mal das *Vimeo*-Objekt bei Übergabe des aktuellen Links. Dabei können Sie gegebenenfalls den Titel anpassen. So lassen sich beispielsweise alle Sonderzeichen sowie Leerzeichen mit geeigneten Methoden entfernen. Den Titel für einen Link rufen Sie zunächst über die Metadaten ab, indem Sie der Methode *get_title* das zuvor initialisierte Vimeo-Objekt übergeben:

```
def get_title(v):
    meta = v.metadata
    return meta.title
```

Danach können Sie den Titel eines Videos mittels der Methode `set_filename` anpassen (Listing 8). Bei dieser Methode werden alle Sonderzeichen einschließlich der Leerzeichen entfernt. Anschließend prüft die Methode die Länge des Strings und kürzt ihn gegebenenfalls [7].

Listing 8

```
def set_filename(s):
    max_filename = 255
    normal_string = "".join(ch for ch in s if ch.isalnum())
    stripped_s = normal_string.strip()
    split_string = stripped_s[:max_filename]
    return split_string
```

Das eigentliche Herunterladen des Videos findet in der Methode `download` unter Übermittlung des *Vimeo*-Objekts, des *Download*-Ordners sowie eines neuen Dateinamens statt. Normalerweise beherbergt der Aufruf von `vimeo.streams` eine Liste voller Streams, die für den jeweiligen Titel verfügbar sind. Allerdings handelt es sich beim letzten Listenelement um den Stream mit der höchsten Auflösung:

```
def download(vim,path,filename):
    stream = vim.streams
    best_stream = stream[-1]
    best_stream.download(download_directory=path,
filename=filename)
```

Alternativ lässt sich über die Liste verfügbarer Streams iterieren, um den gewünschten Stream herauszufiltern. Sofern Sie beispielsweise einen Stream mit einer Auflösung von 720 herunterladen wollen, können Sie wie in Listing 9 vorgehen [8].

Listing 9

```
for s in stream:
    if s.quality == '720p':
        s.download(download_directory='video',
filename=v.metadata.title)
```

```

        break
    else:
        print('quality not found')

```

Die Methode `download_all` lädt schließlich alle Videos herunter, indem auf die zuvor erwähnten Hilfsmethoden zurückgegriffen wird (Listing 10). Abgesehen davon, beherbergt diese Methode eine Fortschrittsanzeige, die den aktuellen Download samt Downloadraten ausgibt (**Abb. 5**).

Listing 10

```

vimeo_path = '/home/Videos/vimeo'

v_1 = 'https://vimeo.com/136491689'
v_2 = 'https://vimeo.com/509738789'
v_3 = 'https://vimeo.com/546042556'
v_4 = 'https://vimeo.com/58326464'
v_5 = 'https://vimeo.com/396053468'
v_6 = 'https://vimeo.com/560619626'
v_7 = 'https://vimeo.com/4510860'

videos = [v_1, v_2, v_3, v_4, v_5, v_6]

def download_all(path,urls):
    size = len(urls)
    for url in urls:
        v = Vimeo(url)
        title = get_title(v)
        new_title = set_filename(title)
        i = urls.index(url)
        print('Downloading video ('+str(i+1)+'/'+str(size)+')')
        download(v,path,new_title)

```



```

Downloading video (1/6)
STEMExploreChemistryUniversityofSheffield.mp4: 16% | 18278/117102 [00:27<02:36, 632.47KB/s]

```

Abb. 5: Download von Vimeo-Videos

YouTube-Bot

Genauso wie für Vimeo existiert eine Bibliothek, die Zugriff auf das YouTube-API ermöglicht. Aktuell liegt `pytube` in der Version 12.1.2 vor und lässt sich wie folgt installieren [9]:

```
pip install pytube
```

Bei Bots können Sie abgesehen von Listen Dateien einlesen und durch diese zeilenweise iterieren, um aus dem aktuellen Link ein *youtube*-Objekt zu generieren. In der Regel verfügt ein YouTube-Link über etliche Streams, sodass es empfehlenswert ist, die Auswahl weiter einzuschränken. So lässt sich angeben, dass beispielsweise lediglich Streams mit der Dateiendung *.mp4* heruntergeladen werden. Die Einstellung *progressive=true* sorgt dafür, dass lediglich Streams ausgewählt werden, die sowohl eine Audio- als auch Videospur beinhalten. Die Anweisung *streams.order_by(,resolution')* wählt schließlich unter allen noch verfügbaren Streams das Video mit der höchsten Auflösung aus. Abgesehen davon beinhaltet die *filter*-Methode weitere Parameter, die in Listing 11 jedoch nicht erscheinen [10].

Mit der Downloadmethode *stream.download()* aus dem YouTube-API wird der ausgewählte Stream heruntergeladen [11]. Dabei lässt sich der Download weiter einstellen, indem auf geeignete Parameter zurückgegriffen wird. So kann zusätzlich der Pfad angegeben werden. Der *Timeout*-Parameter hingegen gibt an, wie lange auf eine Antwort vom Server gewartet wird. Daneben lässt sich die Zahl der Downloadversuche weiter eingrenzen (**Abb. 6**).

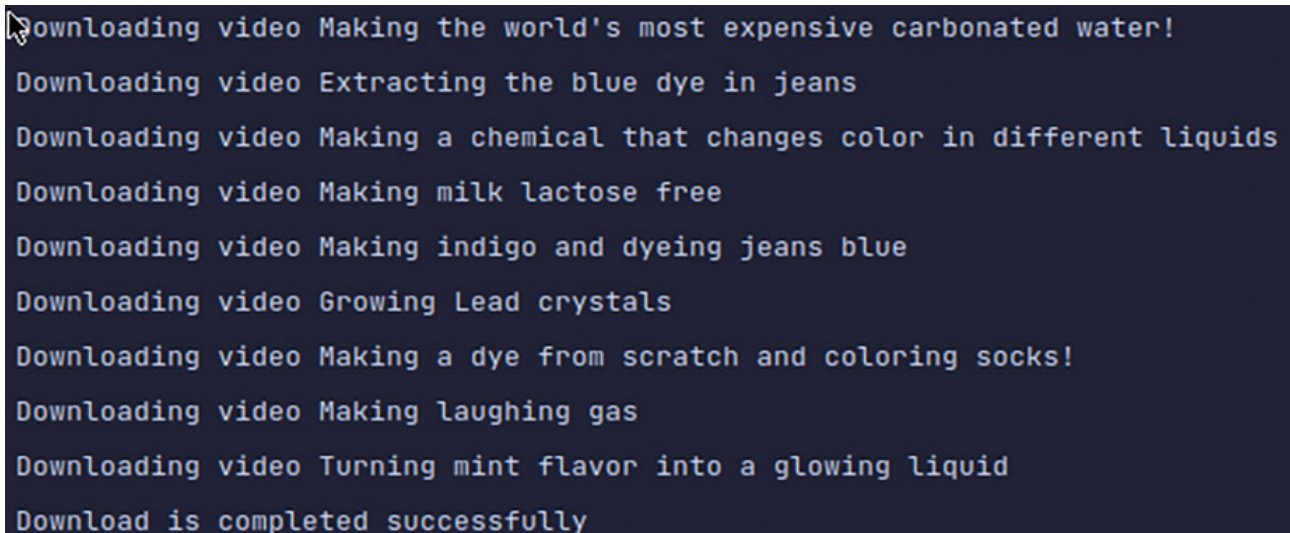
Listing 11

```
def download_yt():
    link = open('/home/Videos/chemistry/links_file.txt',
mode='r')
    SAVE_PATH = '/home/Videos/chemistry'
    for i in link:
        try:
            youtubeObject = YouTube(i)
```

```

    d_video = youtubeObject.streams.filter(progressive=True,
file_extension='mp4').order_by('resolution').desc().first()
    print ('Stream: ' + d_video)
    print('Downloading video: ' + d_video.title)
    d_video.download(SAVE_PATH, timeout=30, max_retries=3)
except:
    print("An error has occurred")
print("Download is completed successfully")

```



```

Downloading video Making the world's most expensive carbonated water!
Downloading video Extracting the blue dye in jeans
Downloading video Making a chemical that changes color in different liquids
Downloading video Making milk lactose free
Downloading video Making indigo and dyeing jeans blue
Downloading video Growing Lead crystals
Downloading video Making a dye from scratch and coloring socks!
Downloading video Making laughing gas
Downloading video Turning mint flavor into a glowing liquid
Download is completed successfully

```

Abb. 6: Download von YouTube-Videos ohne Fortschrittsanzeige

Twitter-Bot

Twitter-Seiten erhalten eine Vielzahl an Informationen, angefangen von Tweets, Retweets, Followern, bis hin zu eingebetteten Videos etc. Dabei ermöglicht die Tweepy-Bibliothek 4.13.0 Zugriff auf das Twitter-API, sodass sich damit Etliches an Daten herunterladen lässt [12]. Um Zugriff auf das Twitter-API zu erhalten, brauchen Sie allerdings einen Entwickleraccount von Twitter. Anhand der folgenden Schritte können Sie einen Entwickleraccount bei Twitter beantragen:

- normales Benutzerkonto auf der Twitter-Seite erstellen [13]
- Benutzerkonto für Entwickler einrichten und sich für den erweiterten Zugang bewerben [14]
- Tokens, Secrets sowie Schlüssel generieren

Als Nächstes installieren Sie die Tweepy-Bibliothek:

```
pip install tweepy
```

Wenn Twitter Ihren Antrag genehmigt hat, sollten Sie über diverse Tokens, Schlüssel sowie Secrets verfügen. Sie können dafür sorgen, dass die Accountdaten griffbereit sind, indem Sie Ihre Zugangsdaten in einer Konfigurationsdatei speichern. Diese Datei enthält Schlüssel-Wert-Paare und ist unter Python wie in **Abbildung 7** aufgebaut. Anschließend speichern Sie die Konfigurationsdatei mit der Dateiendung *.cfg* ab. Angenommen der Dateiname lautet *config.cfg*, dann lässt sich die Konfigurationsdatei wie in Listing 12 einlesen. Die Schlüssel-Wert-Paare eines Abschnitts (*section*) werden dabei in einem Dictionary abgelegt [15].

Listing 12

```
import configparser
```

```
def get_config_dict(section):
    config = configparser.RawConfigParser()
    config.read(r'config.cfg')
    if not hasattr(get_config_dict, 'config_dict'):
        get_config_dict.config_dict = dict(config.items(section))
    return get_config_dict.config_dict
```

```
[TWITTER]
```

```
c_key = HIDDEN
```

```
c_secret = HIDDEN
```

```
a_token = HIDDEN
```

```
a_secret = HIDDEN
```

```
b_token = HIDDEN
```

Abb. 7: Konfigurationsdatei unter Python

Um etwas bei Twitter automatisieren zu können, ist es erforderlich, sich erst einmal zu authentifizieren. So liest die Methode *create_api* zunächst die zuvor generierten Tokens, Schlüssel und Secrets unter Angabe des Abschnitts ein (Listing 13). Die eigentliche Authentifizierung beim Twitter-API erfolgt mittels der Methode *api.verify_credentials()*, wobei die zurückgegebene API-Variable für weitere Aktionen gebraucht wird [16].

Listing 13

```
import tweepy as tw
from tweepy import OAuthHandler

def create_api():
    config_details = get_config_dict('TWITTER')
    c_key = config_details['c_key']
```

```

c_secret = config_details['c_secret']
a_token = config_details['a_token']
a_secret = config_details['a_secret']
auth = OAuthHandler(c_key, c_secret)
auth.set_access_token(a_token, a_secret)
api = tw.API(auth, wait_on_rate_limit=True)
try:
    api.verify_credentials()
except Exception as e:
    print("Error creating API")
    raise e
print("API created")
return api

```

Bei Datenanalysen ist es relevant, Tweets zu einem bestimmten Hashtag zu analysieren. Die Suchergebnisse lassen sich dabei in einem Dataframe ablegen, um hinterher in der Lage zu sein, große Datenmengen zu analysieren. Bei dieser Suche erfolgt die Authentifizierung mit dem Bearer-Token (Inhabertoken), das ebenfalls über die Konfigurationsdatei abrufbar ist. So nimmt die Methode `get_tweets` zunächst die Authentifizierung vor (Listing 14). Danach sucht die Methode `client.search_recent_tweets` nach Tweets der letzten sieben Tage, indem zuvor die Suchanfrage `query` definiert wird. Die Suchanfrage beinhaltet neben dem Hashtag auch die Sprache der in Frage kommenden Tweets, wobei die Suche lediglich aus `#<Suchwort>` bestehen kann. Dabei wird die maximale Zahl an Suchergebnissen auf `100` begrenzt. Außerdem lässt sich in der Suchmethode `client.search_recent_tweets` definieren, welche Felder eines Tweets gespeichert werden (**Abb. 8**). Zusätzlich lassen sich die Tweets in einem Dataframe ablegen [17].

Listing 14

```

def get_tweets(search):
    config_details = get_config_dict('TWITTER')
    token = config_details['b_token']
    client = tw.Client(bearer_token=token)
    # What to search for
    query = '#' + search + ' -is:retweet lang:de'

```

```

max_results = 100
# We grab tweets + context annotations + create date + user
name of tweeter
tweets = client.search_recent_tweets(query=query,
tweet_fields=['context_annotations', 'created_at'],
user_fields=['name'], expansions='author_id',
max_results=max_results)
for tweet in tweets.data:
    print(tweet.text)
    if len(tweet.context_annotations) > 0:
        print(tweet.context_annotations)
# Return the tweet data as a dataframe
df2 = pd.DataFrame(tweets.data).astype(str)
# Collect the user IDs
df_users = pd.DataFrame(tweets.includes['users'])

```

```

API created
Tweet-Text:
Die 15-Millionen-Forderung ist der #Eintracht zu hoch, berichtet @Sport1.

#sge #vfb #mavropanos
https://t.co/8vpmz0Mh
Annotation:
[{'domain': {'id': '3', 'name': 'TV Shows', 'description': 'Television shows from around the world'}, 'entity': {'id': '10039036407', 'name': 'Bundesliga Soccer', 'description': 'Soccer action from the German Bundesliga.'}},
Tweet-Text:
News #Mavropanos: #SGE kennt den Preis, der #VFB will über 15 Millionen Euro. Interesse ist zwar da, aber diese Summe ist für die #Eintracht zu teuer. 000 https://t.co/SzVQpr08So
Annotation:
[{'domain': {'id': '3', 'name': 'TV Shows', 'description': 'Television shows from around the world'}, 'entity': {'id': '10039036407', 'name': 'Bundesliga Soccer', 'description': 'Soccer action from the German Bundesliga.'}},
Tweet-Text:
#SGEVFB #Mavropanos hat zwar noch kein Tor geschossen, aber wenn ich seine Leistung richtig deute, will er zu uns.
Annotation:
[{'domain': {'id': '6', 'name': 'Sports Event', 'entity': {'id': '1618941262048780272', 'name': 'Eintracht Frankfurt vs VfB Stuttgart'}}, {'domain': {'id': '10', 'name': 'Person', 'description': 'Named people in the world'}}]

```

Abb. 8: Tweets zum Hashtag #Mavropanos zusammen mit Annotationen

Abgesehen von Tweets basierend auf Hashtags ist es möglich, alle möglichen Tweets eines Twitter-Accounts herunterzuladen. Hierfür wird angenommen, dass die Tweet-ID eines Tweets zusammen mit weiteren Kategorien wie *Timestamp*, *Text* etc. in einer CSV-Datei gespeichert ist (**Abb. 9**). Dieser Twitter-Bot ist dann in der Lage, die Tweets von einem Account anhand der ID zu identifizieren und herunterzuladen.

```

tweet_id,in_reply_to_status_id,in_reply_to_user_id,timestamp,source,text,retweeted_status_id,retweeted_status_user_id,retweeted_status_timestamp,expanded_urls,rating_numerator,rating_denominator,name,doggo,fiio
89242064355336193,,2017-08-01 18:20:56 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","This is Phineas. He's a mystical boy. Only ever appears in the hole of a donut. 13/10 https://t.co/MgUwQ76dJU,,https://twitter.com/dog_rates/status/89242064355336193/photo/1,13,10,Phineas,None,None,None,None
892177421306343426,,2017-08-01 00:17:27 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","This is Tilly. She's just checking pup on you. Hopes you're doing ok. If not, she's available for pats, snugs, boops, the whole bit. 13/10 https://t.co/0Xxu71qeIV,,https://twitter.com/dog_rates/status/892177421306343426/photo/1,13,10,Tilly,None,None,None,None
891815181378084864,,2017-07-31 00:10:03 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","This is Archie. He is a rare Norwegian Pouncing Corgo. Lives in the tall grass. You never know when one may strike. 12/10 https://t.co/WuZnhtVJB,,https://twitter.com/dog_rates/status/891815181378084864/photo/1,12,10,Archie,None,None,None,None
8918095527888686,,2017-07-30 15:58:51 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","This is Daria. She commenced a snooze mid meal. 13/10 happens to the best of us https://t.co/ID36da7qLQ,,https://twitter.com/dog_rates/status/8918095527888686/photo/1,13,10,Daria,None,None,None,None
891327558926688256,,2017-07-29 16:00:24 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","This is Franklin. He would like you to stop calling him ""cute."" He is a very fierce shark and should be respected as such. 12/10 #BarkWeek https://t.co/AtUzn9177F,,https://twitter.com/dog_rates/status/891327558926688256/photo/1,https://twitter.com/dog_rates/status/891327558926688256/photo/1,12,10,Franklin,None,None,None,None
891087950875887856,,2017-07-29 06:08:17 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","Here we have a majestic great white breaching off South Africa's coast. Absolutely n°okin breathtaking. 13/10 (G: tucker.marlo) #BarkWeek https://t.co/K0q4FD0Rmh,,https://twitter.com/dog_rates/status/891087950875887856/photo/1,13,10,None,None,None,None,None
890971913173991426,,2017-07-28 15:27:12 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","Meet Jax. He enjoys ice cream so much he gets nervous around it. 13/10 help Jax enjoy more things by clicking below https://t.co/Zr4HwFAsIH https://t.co/LV3BRMhxt1,,https://gofundme.com/ydvewe-surgery-for-jax,https://twitter.com/dog_rates/status/890971913173991426/photo/1,13,10,Jax,None,None,None,None
890729181411237888,,2017-07-28 06:22:40 +0000,"ca href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone<a>","When you watch your owner call another dog a good boy but then they turn back to you and say you're a great boy. 13/10 https://t.co/v0n0NBcwq,,https://twitter.com/dog_rates/status/890729181411237888/photo/1,https://twitter.com/dog_rates/status/890729181411237888/photo/1,13,10,None,None,None,None,None

```

Abb. 9: CSV-Datei zusammen mit den Tweet-IDs eines Twitter-

Accounts

Die selbstdefinierte Methode `save_data_to_file` ermöglicht unter Angabe des API, dem Pfad zum Twitter-Archiv sowie dem Zielpfad für die Tweets das Herunterladen von Tweets zu automatisieren (Listing 15). Nach der Authentifizierung wird zunächst ein Twitter-Archiv namens `twitter-archive-enhanced.csv` eingelesen. Anschließend werden vom zuvor erzeugten Dataframe die Tweet-IDs extrahiert und in einer Liste abgelegt. Durch diese Liste wird anschließend iteriert, wobei pro Tweet eine JSON-Datei heruntergeladen wird. Die heruntergeladenen JSON-Dateien landen später alle in der Textdatei `tweet_json.txt`. Anhand einer selbstdefinierten Fortschrittsanzeige weiß der Benutzer sofort Bescheid, welche Tweets sich herunterladen lassen und bei welchen Tweets der Download scheitert (**Abb. 10**). Besteht der Download hingegen aus mehreren Tausend Tweets, kommt das Rate Limit von Twitter zum Tragen und der Download wird in bestimmten Zeitintervallen für einige Minuten unterbrochen. Dadurch wird der Download einer großen Menge Tweets in die Länge gezogen.

Listing 15

```
import tweepy as tw
from tweepy import OAuthHandler
import json
from timeit import default_timer as timer
import pandas as pd

def save_data_to_file(api, sFrom, sTo):
    df = pd.read_csv(sFrom)
    tweet_ids = df.tweet_id.values
    len(tweet_ids)
    count = 0
    fails_dict = {}
    start = timer()
    # Save each tweet's returned JSON as a new line in a .txt
    file
    with open(sTo, 'w') as outfile:
        # This loop will likely take 20-30 minutes to run because
```

of Twitter's rate limit

```
for tweet_id in tweet_ids:
    count += 1
    print(str(count) + ": " + str(tweet_id))
    try:
        tweet = api.get_status(tweet_id,
tweet_mode='extended')
        print("Success")
        json.dump(tweet._json, outfile)
        outfile.write('\n')
    except tw.errors.TweepyException as e:
        print("Fail")
        fails_dict[tweet_id] = e
    pass
end = timer()
print(end - start)
print(fails_dict)
```


Success

888: 759923798737051648

Fail

889: 759846353224826880

Success

890: 759793422261743616

Success

891: 759566828574212096

Fail

892: 759557299618865152

Success

893: 759447681597108224

Success

894: 759446261539934208

Success

895: 759197388317847553

Success

896: 759159934323924993

Success

897: 759099523532779520

Success

898: 759047813560868866

Success

899: 758854675097526272

Success

900: 758828659922702336

Success

901: 758740312047005698

Rate limit reached. Sleeping for: 447

Abb. 10: Download von

Tweets



Abgesehen vom Schreiben bietet Anzela Minosi Dienstleistungen auf Legiit.com an. Dort erstellt Anzela Datenanalysen, Datenbanksoftware, Python-Skripte sowie Kommandozeilentools für den Raspberry Pi. Bevor Anzela sich selbständig gemacht hat, war sie zehn Jahre lang in den Automobil-, Bildungs- und Telekommunikationsbranchen tätig, wo sie diverse IT-Tätigkeiten ausübte: Support, Softwaretests sowie Webentwicklung. Anzela verbringt ihre Freizeit gerne an der ligurischen Küste, fährt Fahrrad und spielt Retrospiele auf dem Raspberry Pi. Sie steht für Redaktionsprojekte sowie für persönliche Beratungsgespräche zur Verfügung.

Links & Literatur

[1] PyPi: <https://pypi.org/>

[2] Requests:
<https://www.geeksforgeeks.org/get-post-requests-using-python/>

[3] Datenvolumen: <https://firstsiteguide.com/big-data-stats/>

[4] Extract Social Media:
<https://pypi.org/project/extract-social-media/>

[5] Requests:
<https://requests.readthedocs.io/en/latest/user/quickstart/>

[6] Vimeo: <https://pypi.org/project/vimeo-downloader/>

[7] Sonderzeichen in Strings:

<https://www.scaler.com/topics/remove-special-characters-from-string-python/>

[8] Vimeo - Download :
<https://jakeroid.com/blog/how-to-download-vimeo-video-using-python/>

[9] pytube: <https://pytube.io/en/latest/>

[10] Streams: <https://pytube.io/en/latest/user/streams.html>

[11] YouTube - Download :
<https://www.geeksforgeeks.org/pytube-python-library-download-youtube-videos/>

[12] Tweepy: <https://pypi.org/project/tweepy/>

[13] Twitter: <https://twitter.com>

[14] Entwicklerportal :
<https://developer.twitter.com/en/support/twitter-api/developer-account>

[15] Konfigurationsdatei :
<https://stackoverflow.com/questions/19379120/how-to-read-a-config-file-using-python>

[16] Twitter - Bot :
<https://realpython.com/twitter-bot-python-tweepy>

[17] Tweet - Suche :
<https://towardsdatascience.com/how-to-access-data-from-the-twitter-api-using-tweepy-python-e2d9e4d54978>

Mobile App Entwicklung im Überblick

Es scheint fast für jeden etwas dabei zu sein. So oder ähnlich kann man die Situation beschreiben, wenn es um Technologien, Frameworks und Vorgehensweisen zur Entwicklung von Mobile-Apps geht. Wir haben versucht, den Überblick zu behalten.

Redet man über die Entwicklung von Apps für mobile Systeme, dann meint man damit die Entwicklung für Android und iOS. Doch wie kommt man zu einer solchen App? Hier können wir gleich am Anfang des Artikels festhalten, dass es für dieses Ziel eine sehr große Anzahl sehr unterschiedlicher Technologien und Vorgehensweisen gibt. Wir möchten in diesem Artikel einen Überblick über die verschiedenen Möglichkeiten geben und dabei versuchen, diese unterschiedlichen Ansätze nach bestimmten Kriterien zu systematisieren. Starten wir zunächst mit den Kriterien.

Systematik der Entwicklungsansätze

Bei der Entwicklung von Mobile-Apps kann man unterschiedliche Kriterien für die Systematisierung heranziehen. Nach den folgenden Kriterien ist eine Einteilung möglich

- nach der Art der entstehenden App, d. h. nativ, webbasiert oder hybrid
- nach der verwendeten Programmiersprache und den zum Einsatz kommenden Framework
- nach dem Umfang der Toolunterstützung, d. h. einer codebasierten Erstellung oder dem Einsatz von RAD- oder Low-Code-Tools
- nach dem Zielsystem oder ob mit Cross-Platform- bzw. Cross-Device-Programmierung mehrere Systeme aus einer gemeinsamen Quellcodebasis angesprochen werden

Weitere Kategorien sind sicherlich denkbar, die meisten relevanten Ansätze dürften sich jedoch hierunter subsumieren lassen. Die folgenden Textabschnitte stellen einige interessante Ansätze vor und versuchen sie entsprechend einzuordnen. Das wird mit Blick auf die Kriterien nicht immer überschneidungsfrei gelingen, dennoch dürfte auf diese Weise ein umfassender Überblick über die aktuellen Technologien zur Entwicklung von mobilen Apps entstehen. Starten wir mit dem naheliegendsten Kriterium, der Art der entstehenden App.

Native Apps

„Entscheidend ist, was hinten rauskommt“, meinte der ehemalige Bundeskanzler Helmut Kohl – zwar in einem vollständig anderen Zusammenhang, wir können diesen Ausspruch jedoch auch auf den Bereich der Entwicklung von Mobile-Apps übertragen. Man kann technologisch grundsätzlich drei Arten von Apps unterscheiden: native Apps, Web-Apps und hybride Apps.

Native Apps sind die Applikationen, deren Programmcode unmittelbar durch das Betriebssystem auf dem mobilen Gerät ausgeführt wird. Sie laufen in keinem Browser, sondern greifen direkt auf die Programmierschnittstellen von Android und iOS zurück. Mit dem direkten Zugriff auf die APIs der Systeme ist eine vollumfängliche Nutzung aller Geräte- und Hardwarefunktionen möglich. Beispielsweise kann man alle Sensoren ohne Einschränkungen verwenden. Auch die Ausführungsgeschwindigkeit der App ist die bestmögliche. Native Apps können offline betrieben werden und man kann sie über die App-Stores vertreiben. Wie gelangt man zu diesen nativen Apps? Als Erstes sind hier die von den Herstellern der Betriebssysteme vorgesehenen Vorgehensweisen und Tools zu nennen. Für Android ist es die Entwicklungsumgebung Android Studio [1] mit der Programmiersprache Kotlin bzw. Java (**Abb. 1**).

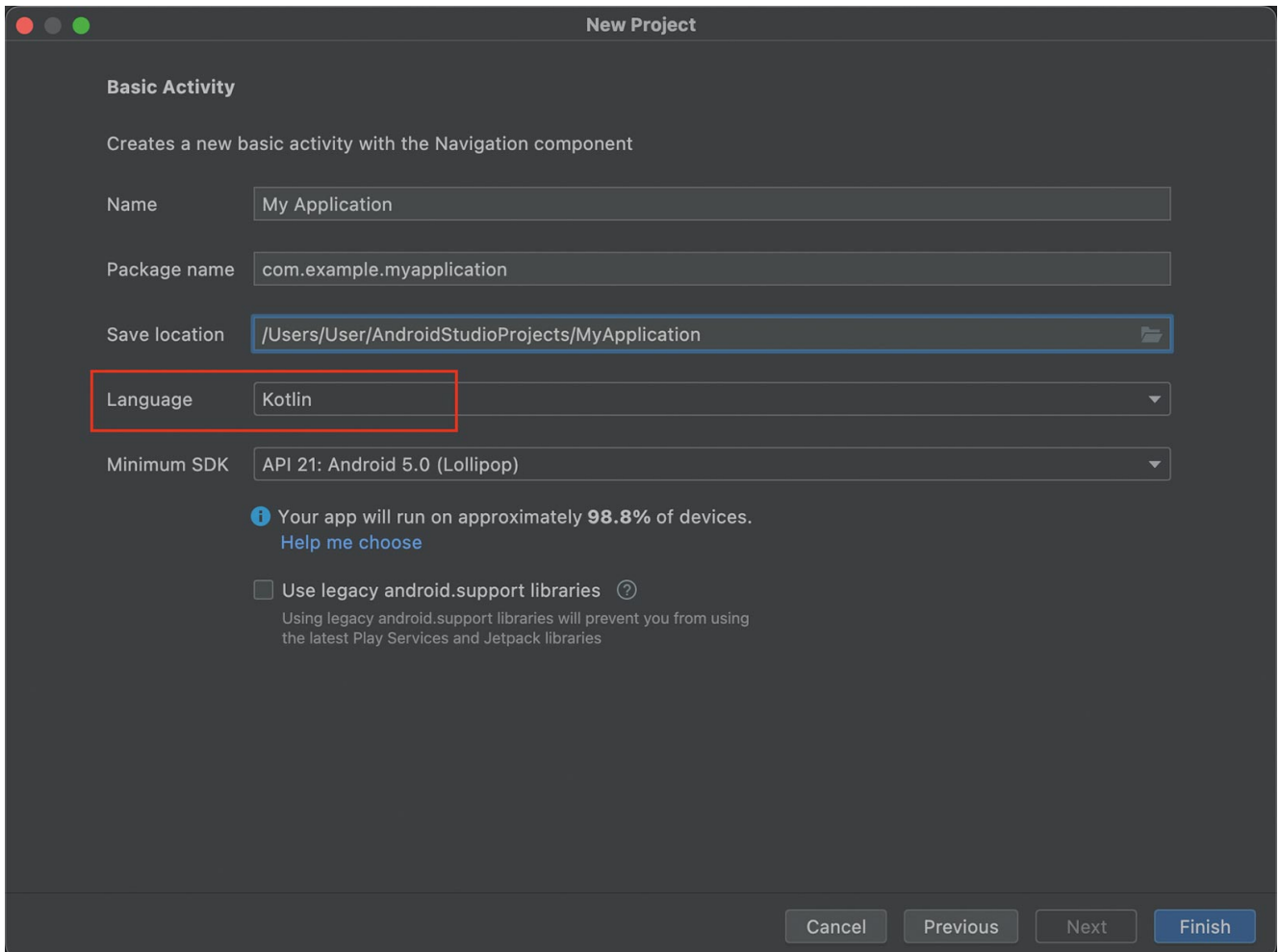


Abb. 1: Projektassistent von Android Studio für die Entwicklung von nativen Android-Apps

Die Auswahl der Programmiersprache können Sie beim Anlegen eines Projekts treffen. Beide Sprachen können jedoch auch gemischt genutzt werden. In der Entwicklungsumgebung steht u. a. ein Designer zur Verfügung, um die Oberfläche zu gestalten. Mit diesem Entwicklungsansatz können Sie auch für alle anderen Formfaktoren von Android-Devices entwickeln, beispielsweise für Android TV, Android for Car und verwandte Betriebssysteme wie Wear OS (Smartwatch) und Chrome OS. Um direkt gegen das Android SDK zu programmieren, könnte man auch eine andere Entwicklungsumgebung verwenden, die meisten Entwickler:innen werden jedoch Android Studio einsetzen.

Kommen wir zu iOS. Hier wird Xcode [2] als integrierte Entwicklungsumgebung benutzt. Diese läuft ausschließlich auf macOS und man benötigt daher zwingend einen Mac. Apple erlaubt keine andere Vorgehensweise, um das App-Package final für das

Zielsystem zu erstellen. Während der Entwicklung kann man sich ggf. mit Hilfe eines in die Cloud ausgelagerten Remote-Apple behelfen. Schauen Sie sich dabei beispielsweise die Optionen unter [3] an. Hier können Sie einen Mac mieten und mit Xcode arbeiten. Auch das Ausführen eines Simulators ist möglich, der Bildschirm wird dazu über das Netzwerk auf den eigenen Entwicklungsrechner projiziert. Dieses Vorgehen dürfte interessant sein, wenn man mit Hilfe eines anderen Ansatzes (Cross-Platform, hybrid) eine App für iOS erstellen möchte und sich dabei die Anschaffung von mehreren Macs (bei Teamarbeit) wirtschaftlich nicht lohnt. Zurück zur iOS-Entwicklung: Als Programmiersprache wird Swift verwendet. Das User Interface wird in SwiftUI deklarativ im Quellcode erstellt. Swift und SwiftUI haben Objective-C und die Nutzung von Storyboards für das Erstellen der Benutzeroberfläche weitgehend abgelöst. Die Live-Preview in Xcode zeigt bereits während des Entwurfs und der Entwicklung eine Vorschau des User Interface an und beschleunigt damit die Entwicklung maßgeblich (**Abb. 2**).

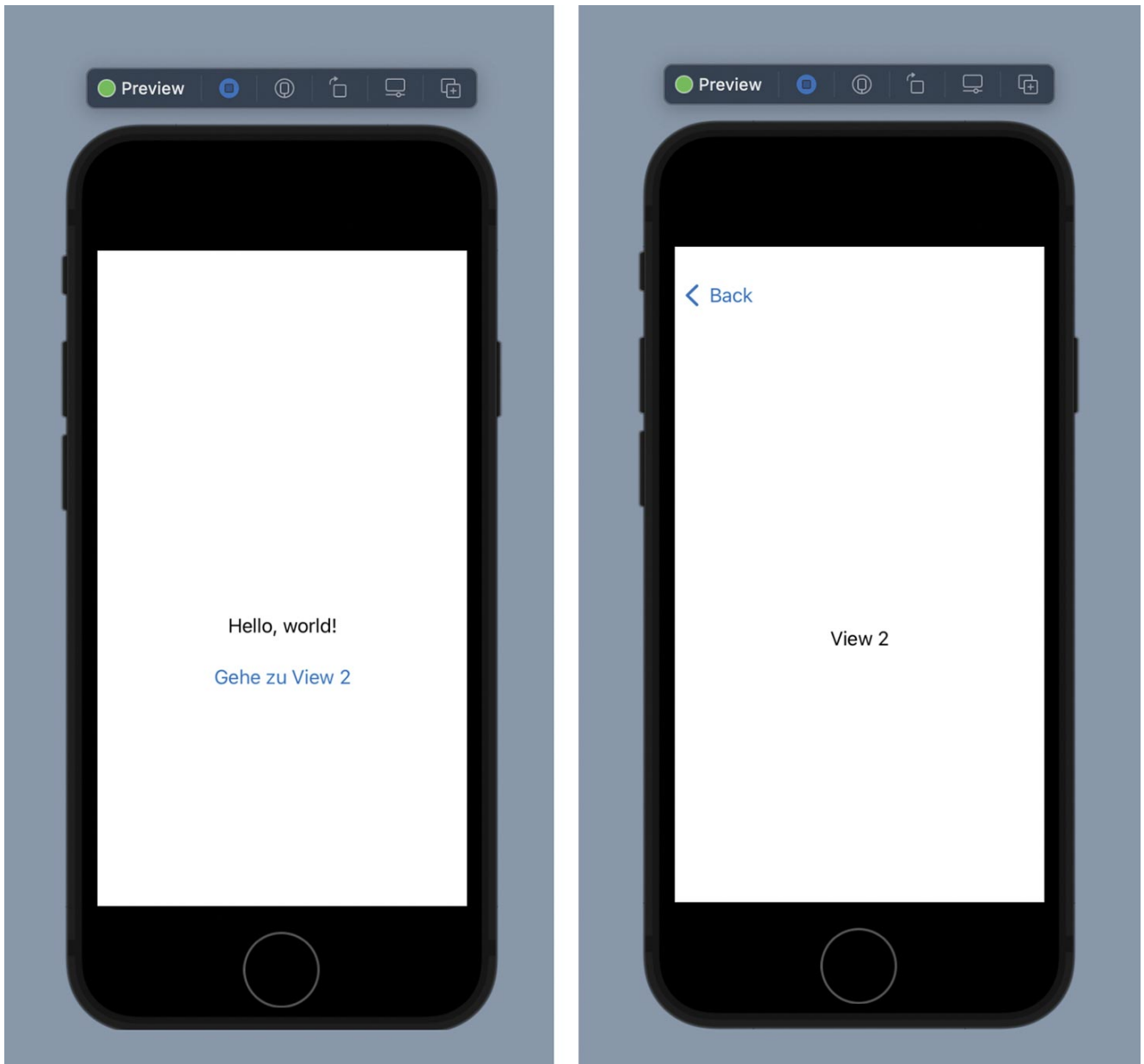


Abb. 2: Live-Preview von Xcode zur Entwicklung von nativen iOS-Apps

Es besteht kein Zweifel: Wenn Sie alle Features von Android und iOS nutzen bzw. die neuste Hardware adressieren und keinerlei Kompromisse bei der Gestaltung des User Interface machen wollen, dann bleibt Ihnen nur die Entwicklung mit diesen beiden von den Herstellern Google und Apple vorgeschlagenen Ansätzen. Will man beide Plattformen bedienen, dann muss die App zweifach programmiert werden. Es ist ein nahezu doppelter Aufwand, denn der Quellcode lässt sich zwischen den Systemen kaum teilen, wenn man von einigen übergreifenden Konzepten absieht. Hinzu kommt der Umstand, dass Android-Entwickler:innen meistens nicht über ausreichende

Kennnisse in der nativen iOS-Entwicklung verfügen und umgekehrt. Man braucht daher auch zwei Teams.

Auch wenn man sich für die Cross-Platform- oder hybride Entwicklung von Apps entscheidet, ist es hilfreich, mit den nativen Ansätzen zu experimentieren. Gelegentlich hackelt es bei der Cross-Platform-Programmierung, man muss einige plattformspezifische Einstellungen vornehmen oder man muss Plattformcode schreiben, um beispielsweise einen Sensor zu erreichen. Das geht dann nur mit Hilfe von Kenntnissen der APIs der Plattformen. Es ist auf jeden Fall nützlich, sich Wissen über iOS und Android anzueignen.

Cross-Platform- und Cross-Device-Programmierung

Es ist ein ewiger Entwicklerwunsch: „Write once, run anywhere (WORA)“. Cross-Platform-Programmierung verfolgt das Ziel, eine App zu erstellen, die auf unterschiedlichen Betriebssystemen, zum Beispiel Android und iOS, ausgeführt werden kann. Cross-Device-Programmierung geht dabei noch einen Schritt weiter. Wir adressieren hier nicht nur unterschiedliche Betriebssysteme, sondern auch verschiedene Formfaktoren der Geräte, beispielsweise Smartphone, Tablet, PC, Notebook, TV usw. Nach dieser Lesart ist die Cross-Platform-Programmierung also ein Teil der Cross-Device-Entwicklung.

Unter Cross-Platform-Programmierung verstehen wir Ansätze mit dem Ziel, Apps für Android und iOS aus einer gemeinsamen Quellcodebasis zu generieren. Die entstehenden Apps sollten dabei im Ergebnis einer nativen App möglichst nahekommen. Web-Apps bzw. hybride Apps, die wir bei einer weitergehenden Betrachtung auch als Cross-Platform bezeichnen könnten, meinen wir damit explizit nicht. Sehen wir uns einige Möglichkeiten an.

Ein neuerer Ansatz ist .NET MAUI von Microsoft [4]. Man

erreicht damit die Systeme iOS, Android, Windows und macOS und über den Herstellersupport von Samsung auch Tizen. Entwickelt wird in der Programmiersprache C# und das User Interface wird mittels XAML deklarativ erstellt. Das funktioniert für alle Zielsysteme, wobei plattformspezifische Anpassungen möglich sind. Zur Gestaltung der Oberfläche stehen visuelle Controls zur Verfügung, die während des Kompilierens auf die User-Interface-Elemente der Zielsysteme gemappt werden. Statt eines grafischen Designers wird mit dem Hot-Reload-Feature gearbeitet, d. h., Änderungen am Quellcode werden während der Entwicklung direkt in die laufende App übernommen (**Abb. 3**). .NET MAUI ist der Nachfolger des Frameworks Xamarin [5] und eine Migration für bestehende und weiterzuführende Projekte sollte geprüft werden.

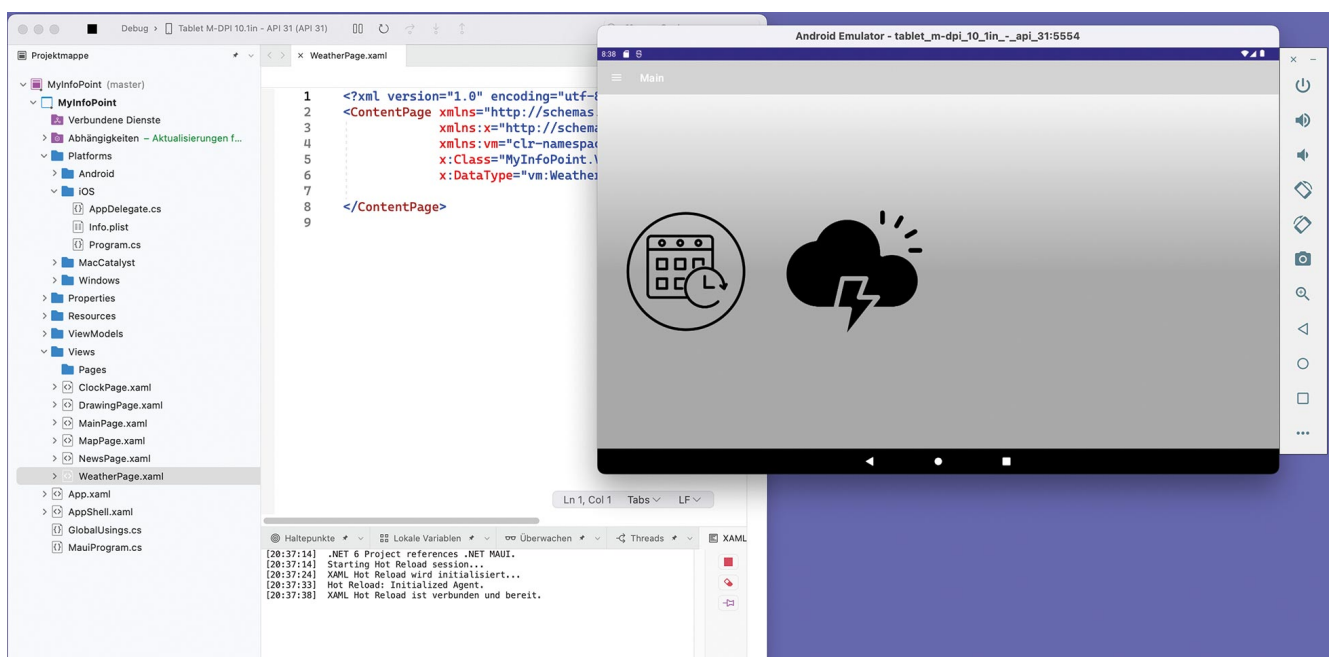


Abb. 3: Hot Reload beschleunigt die Entwicklung von MAUI-Apps (Visual Studio for macOS)

Aus dem Hause Google stammt der Ansatz Flutter [6]. Mit Flutter kann man Apps für nahezu alle relevanten Clientsysteme erstellen, d. h. für Desktop, Mobile und Web und damit auch Apps für Android und iOS. Als Programmiersprache wird Dart verwendet. Diese Sprache wurde ebenfalls von Google entwickelt. Es entstehen native Apps. Das Rendering des User Interface erfolgt über die C++-2D-Rendering-Engine Skia, die

u. a. auch in Google Chrome, Mozilla Firefox und anderen Programmen verwendet wird. Flutter zeichnet die User-Interface-Controls eigenständig und stellt dazu vielfältige Widgets zur Implementierung der Benutzeroberfläche zur Verfügung. Die Entwicklung mit dem Flutter SDK kann auf unterschiedlichen Systemen durchgeführt werden (Windows, macOS, Linux, Chrome OS). Auch bei der Wahl des Editors bzw. der Entwicklungsumgebung ist man flexibel, beispielsweise Android Studio (bevorzugt) oder Visual Studio Code.

Mit RAD Studio [7] gibt es einen weiteren Ansatz, um eine App für mehrere Plattformen zu erstellen. Programmiert wird in der Sprache Delphi (Object Pascal) und man kann Anwendungen für die Systeme Windows, macOS, Linux, iOS und Android erstellen. Die Entwicklungsumgebung bietet einen grafischen Designer, in der die Benutzeroberfläche der App vollständig aus visuellen Controls zusammengesetzt wird (**Abb. 4**).

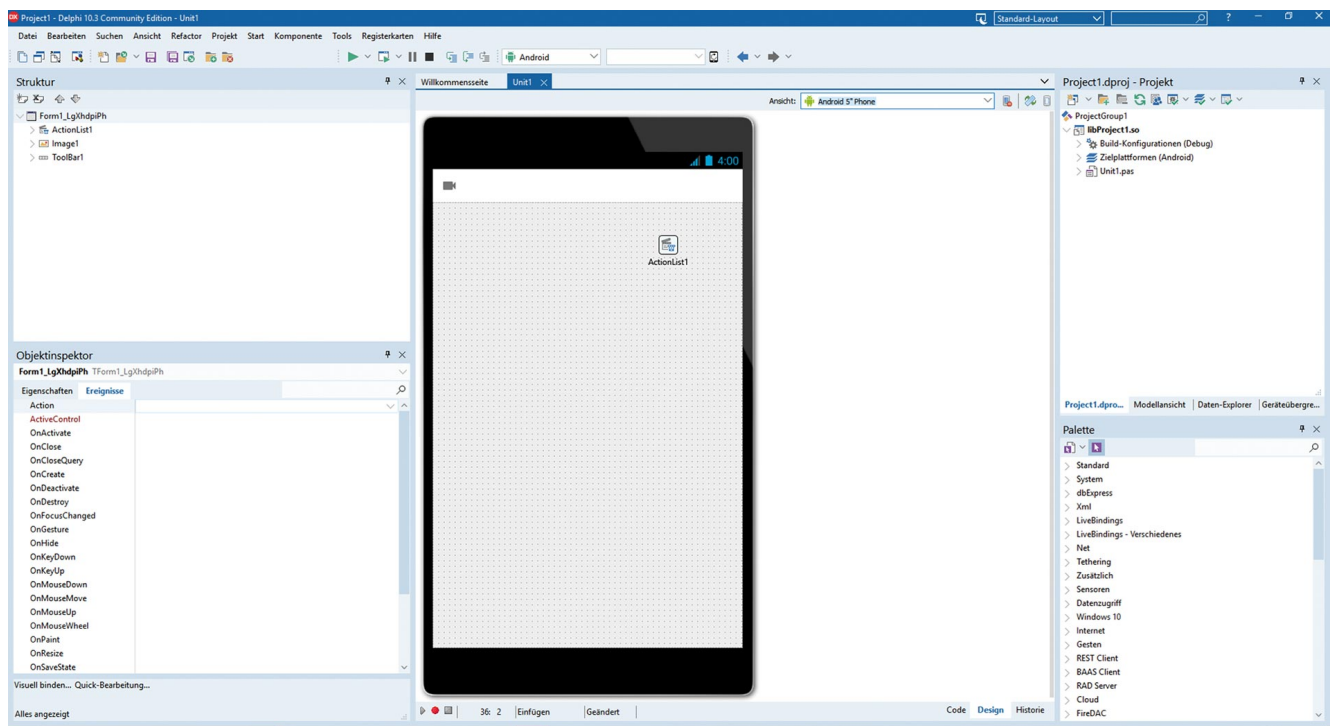


Abb. 4: Grafischer Designer in RAD Studio zur UI-Erstellung mit FireMonkey

Nichtvisuelle Controls stehen beispielsweise für den Zugriff auf Gerätefunktionen (z. B. Dateisystem) oder Sensoren (z. B. Bluetooth) zur Verfügung. Es handelt sich um native Apps, die

direkt auf den Zielsystemen ausgeführt und in die App Stores eingestellt werden können. RAD Studio nutzt für das Rendering des UI das geräteübergreifende Grafikframework Fire Monkey, das die systemeigenen Grafikbibliotheken der Zielsysteme verwendet. Die besprochenen und weitere Ansätze zur geräte- und plattformübergreifenden Entwicklung haben wir für Sie in Tabelle 1 zusammengefasst.

Ansatz/IDE	Zielsystem	Programmier-sprache/Framework	Betriebssystem	Hersteller	Hinweise	Link
Android Studio	Android	Kotlin Java	Windows Linux macOS	Google	der direkte und offizielle Weg zu einer nativen App für Android Zugriff auf alle APIs der Plattform keine Einschränkungen bei den Features und den Kompatibilitäten	https://developer.android.com/studio
Xcode	iOS	Swift SwiftUI	macOS	Apple	der direkte und offizielle Weg zu einer nativen App für iOS Zugriff auf alle APIs der Plattform keine Einschränkungen bei den Features und den Kompatibilitäten	https://developer.apple.com/xcode/
.NET MAUI/ Visual Studio	iOS Android Windows (WinUI 3) macOS (Tizen)	C# XAML .NET-Plattform	Windows macOS	Microsoft	native Apps für die genannten Zielsysteme aus einer gemeinsamen Quellcodebasis UI wird deklarativ mittels XAML erstellt Plattformanpassungen und plattformspezifischer Code sind möglich Einsatz von Controls für das UI Nachfolger von Xamarin.	https://dotnet.microsoft.com/en-us/apps/maui
Xamarin	iOS Android Windows (UWP) (Tizen macOS 10.13 GTK# WPF)	C# XAML .NET-Plattform	Windows macOS	Microsoft	gemeint ist hier die Entwicklung mit Xamarin.Forms technologischer Vorgänger von .NET MAUI, wird aktuell noch gepflegt eine Migration nach .NET MAUI ist zu prüfen	https://dotnet.microsoft.com/en-us/apps/xamarin
RAD Studio	iOS Android Windows macOS Linux	Delphi	Windows	Embarcadero	RAD-Tool zur Entwicklung von App für unterschiedliche Zielsysteme, u. a. Mobile-Apps für iOS und Android grafischer Designer, in dem mittels Definition von Eigenschaften die Benutzeroberfläche erstellt wird	https://www.embarcadero.com/products/rad-studio
Flutter	iOS Android Windows macOS Linux Web	Dart	Windows macOS Linux Chrome OS	Google	geräteübergreifende Programmierung von Apps für diverse Clientbetriebssysteme und das Web programmiert wird in der Sprache Dart User Interface wird mittels Widgets erstellt und eigenständig gezeichnet	https://flutter.dev/

Ansatz/IDE	Zielsystem	Programmiersprache/Framework	Betriebssystem	Hersteller	Hinweise	Link
Native-Script	iOS Android	JavaScript TypeScript	Windows macOS Linux	ursprünglich Telereik; aktuell Community	Erstellung nativer Apps für Android und iOS es werden die APIs der Zielsysteme verwendet Programmierung in JavaScript (TypeScript) User Interface kann mit Hilfe von Komponenten erstellt werden (XML)	https://nativescript.org
React-Native	iOS Android	JavaScript TypeScript React	Windows macOS Linux	Metaplattform	Ermöglicht das Erstellen von mobilen Apps für Android und iOS mit Hilfe des React-Frameworks	https://reactnative.dev

Tabelle 1: Übersicht über Ansätze zur geräte- bzw. plattformübergreifenden Entwicklung

Web-Apps

Nicht immer ist es notwendig oder sinnvoll, eine native App zu erstellen. Der Anteil der Nutzer:innen mit mobilen Endgeräten nimmt stetig zu. Gerade in der jüngeren Generation wird heute bereits ein Großteil der Aufgaben im Internet über das Smartphone erledigt. Wird eine Webseite bzw. -applikation für die Nutzung auf einem Smartphone optimiert (Mobile First), dann handelt es sich um eine Web-App. Eine solche Web-App läuft direkt im Browser und wird nicht installiert. Nach der Eingabe der Internetadresse ist sie sofort nutzbar. Da die Applikation auf dem Server läuft, ist eine ständige Datenverbindung notwendig. Dadurch, dass keine Installation auf dem Zielgerät notwendig ist, können diese Apps auch nicht in den Store eingestellt werden. Auch wollen Nutzer:innen diese oftmals ad hoc verwenden, zum Beispiel, um einmalig ein Ticket zu buchen, und eine Installation einer App aus dem Store ist daher nicht zumutbar.

Technisch betrachtet sind Web-Apps meist clientseitige Single-Page Applications. Für eine gute Reaktionsfähigkeit und eine bestmögliche User Experience werden diese typischerweise mit den bekannten Webframeworks Angular, Vue oder React entwickelt. Im Kern basieren sie technisch auf HTML (Struktur), CSS (Layout, Design) und JavaScript (Interaktion). Durch die Verwendung von auf die mobile Nutzung ausgerichteten UI-Bibliotheken kann man das Design der Web-Apps weitgehend

identisch zu einer nativen App gestalten. Ein Beispiel für eine solche UI-Bibliothek ist OnsenUI [8]. OnsenUI bietet einen umfassenden Satz von grafisch reichhaltigen UI-Komponenten, die speziell für Mobile-Apps entwickelt wurden. Diese orientieren sich an den Design-Guidelines von Android und iOS. Das Framework steht unter einer Open-Source-Lizenz (Apache v2) zur Verfügung und kann mit Vue, Angular und React eingesetzt werden.

Beispiele zeigen, dass man mit Web-Apps heute das Feeling (Design, Handling) von nativen Apps nahezu erreicht. **Abbildung 5** zeigt die Ansicht einer Wetter-App, die mit Onsen UI gestaltet wurde [9].

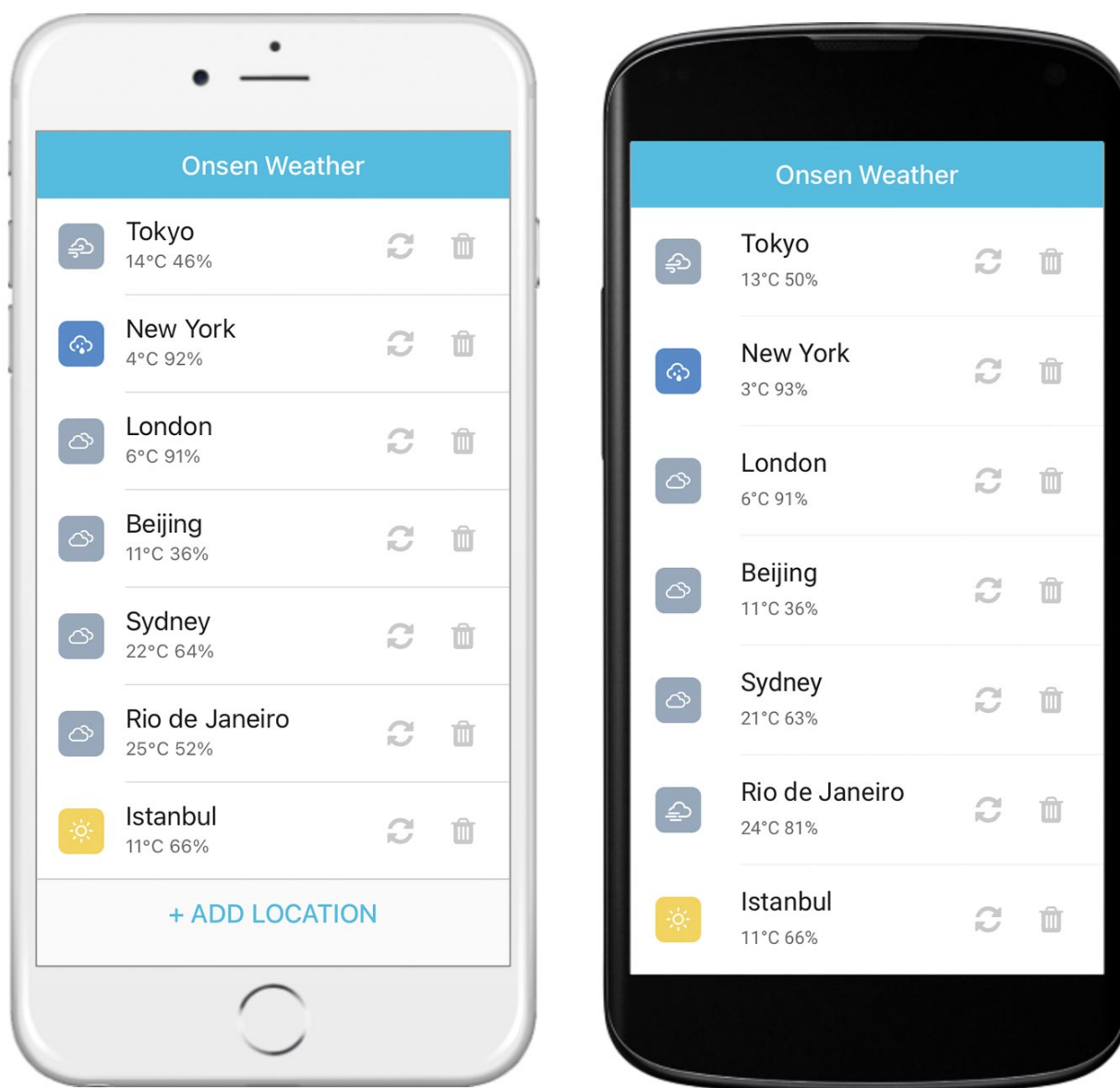


Abb. 5: Wetter-App mit Onsen UI

Da es sich um eine Webapplikation handelt, kann man diese direkt im Browser eines jeden beliebigen Systems aufrufen und starten. Interessant ist dabei auch, dass man das Design an iOS bzw. Android anpassen kann. Wie man sieht, kommt man dem nativen App-Erlebnis hier sehr nahe. Eine Alternative für die Optimierung des UI für die mobile Nutzung bietet auch das Framework Ionic [10], das mit React, Vue und Angular kombiniert werden kann. Ein Beispiel für eine mobile App mit einem Kalender-Control, implementiert mit Ionic und Angular, zeigt **Abbildung 6**.

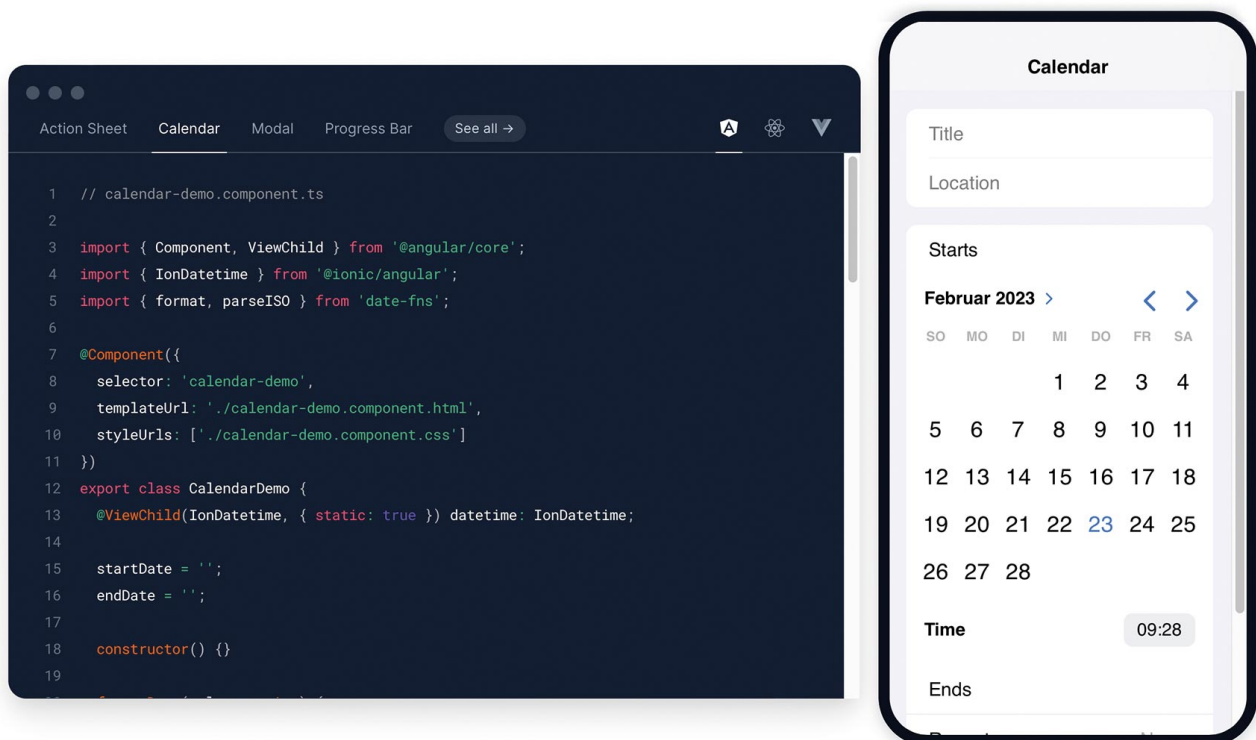


Abb. 6: App mit Kalender-Control mit dem Ionic-Framework

Diese UI-Frameworks bieten Lösungen für die Gestaltung der mobilen App. Da diese direkt im Browser des Systems läuft, sind Geräte- und Systemzugriffe zunächst auf das beschränkt, was über den Browser unmittelbar möglich ist. Das JavaScript API zur Ortung kann beispielsweise ohne Einschränkungen genutzt werden. Dagegen ist ein Zugriff auf die Sensoren eines Smartphones eingeschränkt bzw. nahezu ausgeschlossen.

Hier kommen wir jedoch zu zwei weiteren wichtigen Erweiterungsmöglichkeiten. Eine Web-App, die auf einen mobilen Device ausgeführt wird, kann mit Hilfe einer Progressive Web App (PWA) und mittels eines nativen App-Containers (hybride App) erweitert werden. Mit einer PWA ist es möglich, eine Web-App auch mit Offlinefunktionalität auszustatten. Ebenso kann man für eine solche PWA auch ein Icon auf den Home Screen des mobilen Geräts ablegen. Bei einer hybriden App wird die Web-App im internen Browser einer nativen App verpackt und erhält dann über diesen Zugriff auf die Geräte- und Hardwarefeatures des Smartphones oder Tablets. Wir werden im kommenden Abschnitt noch genauer auf diese Technologie eingehen.

Üblicherweise entwickelt man Web-Apps mit Webtechnologien und -frameworks. Es gibt jedoch auch zu dieser Vorgehensweise Alternativen. An Entwickler:innen, die in der Java-Welt zu Hause sind, richtet sich Vaadin [11]. Mit Hilfe dieses Frameworks kann man eine Web-App mit Java-Quellcode generieren, ohne direkt HTML, CSS und JavaScript schreiben zu müssen. Es steht auch hier eine Reihe von UI-Komponenten zur Verfügung, um die Benutzeroberfläche zu gestalten.

Vaadin kann man auch innerhalb der integrierten Entwicklungsumgebung RapidClipse [12] nutzen. Rapid-Clipse ist eine IDE, die auf Eclipse aufsetzt und u. a. einen grafischen Designer für Vaadin bietet, mit dem man die Benutzeroberflächen intuitiv gestalten kann. Für die Entwicklung von Mobile-Apps (Mobile First) gibt es ein nützliches Feature: Durch einen Button im GUI Builder wird die erstellte Seite lokal gehostet. Für diese Seite wird dann ein QR-Code generiert, den man mit dem Smartphone einscannen kann (**Abb. 7**). Die Seite wird anschließend direkt auf dem mobilen Gerät angezeigt. Voraussetzung ist, dass sich der Entwicklungsrechner mit dem lokalen Server und das Smartphone im gleichen lokalen Netzwerk befinden.

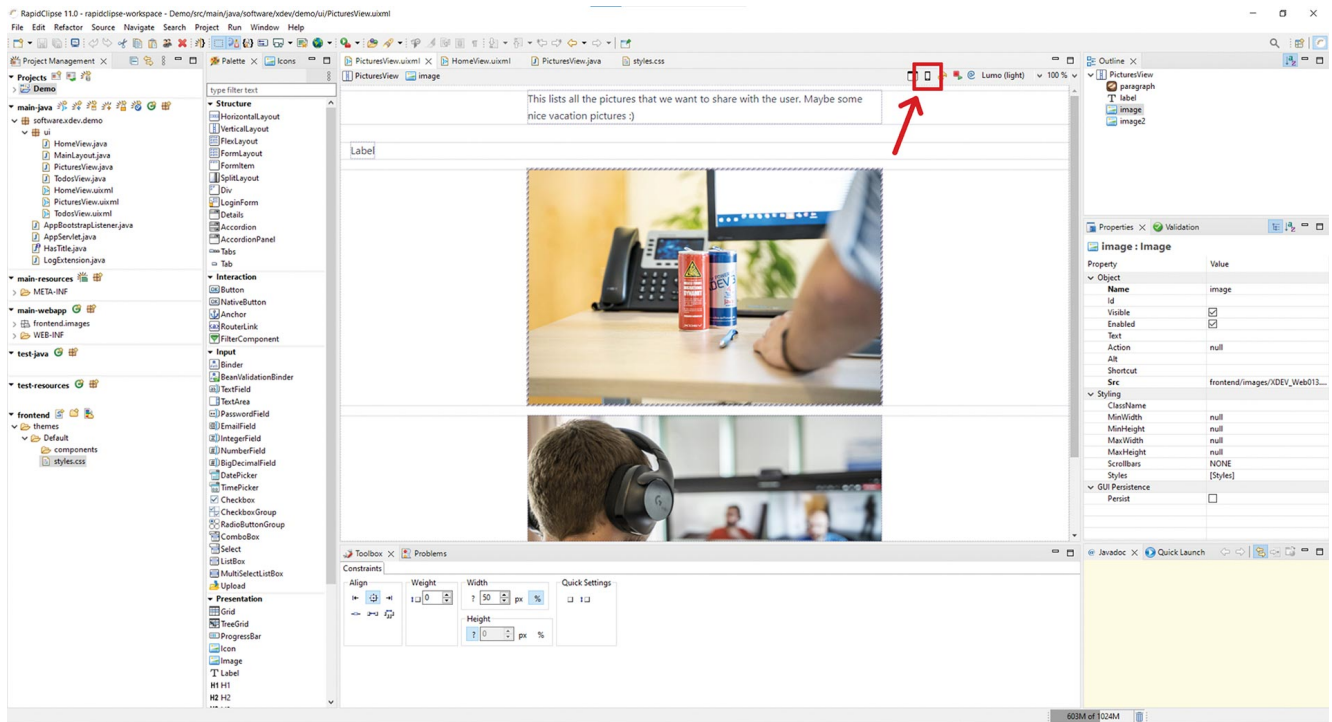


Abb. 7: Web-App mit RapidClipse und Vaadin erstellen und für mobile Geräte testen

Hybride Apps

Das Prinzip ist einfach: Wir haben einen nativen App-Container, der auf das jeweilige Zielsystem angepasst ist. Innerhalb dieses Containers läuft ein Browser (ohne Möglichkeit der Änderung des URL), in dem die Web-App ausgeführt wird. Für das Erstellen der App steht das gesamte Spektrum der bisher behandelten Technologien zur Verfügung. Der Container fungiert als Schnittstelle zwischen den APIs des betreffenden Betriebssystems und der App und ermöglicht dieser einen Zugriff auf die Gerätefunktionen und Sensoren des mobilen Geräts. Das bekannteste Framework ist Cordova [13]. Die Architektur ist in **Abbildung 8** zu sehen.

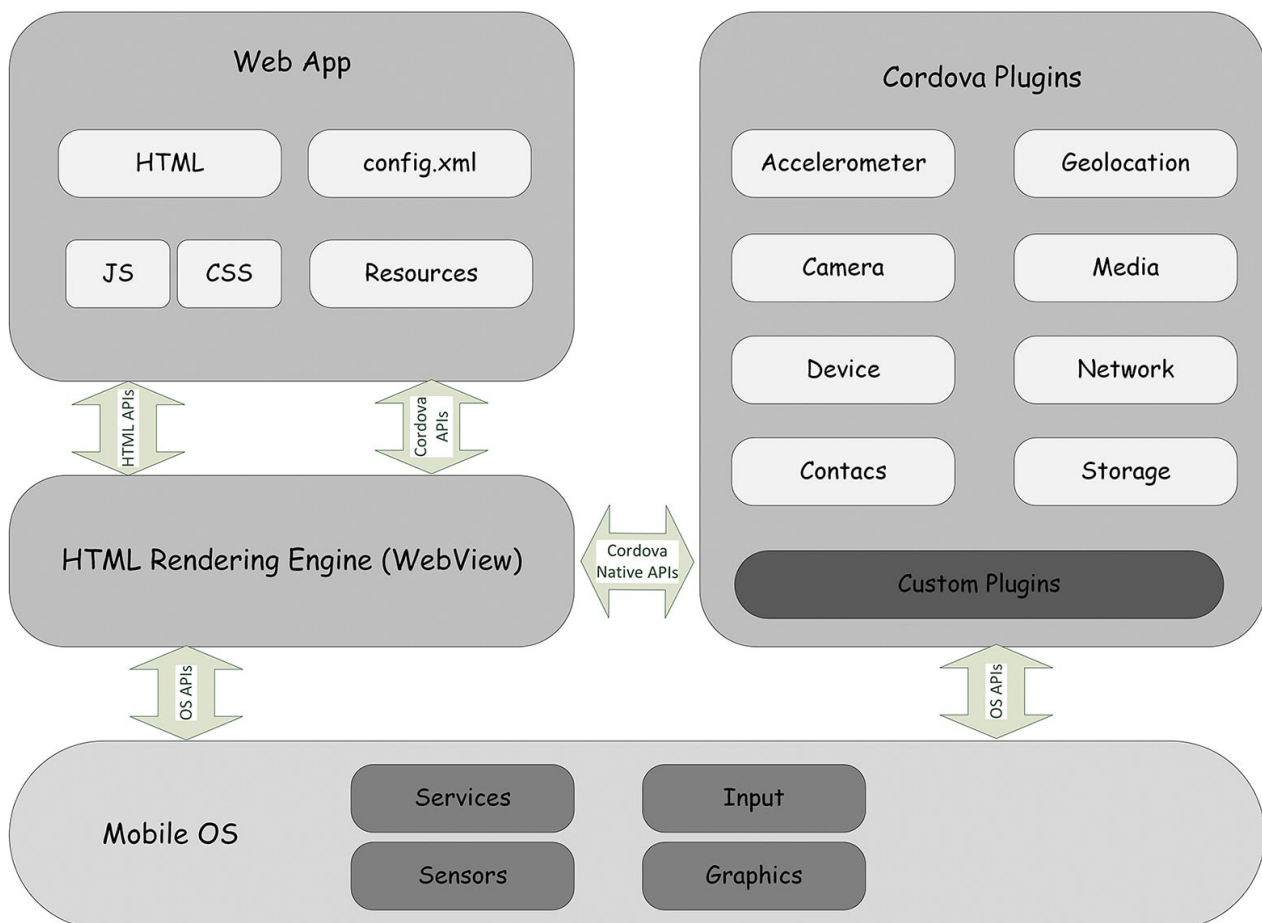


Abb. 8: Die Architektur von Cordova für hybride Apps

Über Plug-ins wird der Zugriff auf die Systemfunktionen sichergestellt. Hybride Apps erscheinen für das System wie eine native Anwendung – sie können offline genutzt werden und man kann sie in den App Stores einstellen. Das Prinzip der hybriden App ist also eine interne Web-View, die eine Webapplikation anzeigt.

Auch hier gibt es weitere Technologien zur Umsetzung. Beispielsweise geht das Framework Wisej.NET einen anderen Weg: Hier wird die Webapplikation auf Basis von .NET erstellt und auf einem Webserver gehostet. Als Entwicklungsumgebung kommt Visual Studio zum Einsatz und man arbeitet mit visuellen Komponenten für die Gestaltung der Oberfläche. Das Framework übersetzt die .NET-Applikation, die auf dem Server läuft, in eine Single-Page App. Diese kann wiederum in einer Web-View auf den mobilen Geräten angezeigt werden. Dazu bietet der Hersteller vorbereitete native Apps, die man individuell an

die Web-App anpassen kann. Gleichwohl sorgt diese Web-View mittels Wisej Mobile dafür, dass man aus der Web-App Zugriff auf das API des mobilen Geräts hat. Die Besonderheit dieses Ansatzes liegt in der Erstellung von Web-Apps. Dazu kann, vergleichbar mit Windows Forms, in Visual Studio per Drag and Drop die Oberfläche mit einem grafischen Designer konfiguriert werden. Die Businesslogik wird in C# programmiert, wodurch es nicht notwendig ist, HTML-, CSS- und Java-Script-Code zu schreiben. Informationen zum Erstellen von Mobile-Apps mit Wisej.NET finden Sie unter [14].

Weitere Ansätze

Haben wir jetzt alle Vorgehensweisen zum Erstellen einer Mobile-App beisammen? Zumindest die grundsätzlichen Ansätze zur Entwicklung von nativen, webbasierten und hybriden Apps, einschließlich der plattformübergreifenden Entwicklung haben wir aufgeführt. Dennoch gibt es weitere Technologien. Daher folgen in diesem Abschnitt einige weitere Tools, mit deren Hilfe man ebenfalls zu einer Mobile-App kommt:

- *Qt*: Das plattformübergreifende GUI-Toolkit kann auch dazu genutzt werden, Mobile-Apps für Android und iOS zu erstellen. Die Programmlogik wird in C++ erstellt und das User Interface mittels QML deklariert. Ein grafischer Designer (Qt-Creator) erleichtert die Gestaltung. Informationen für den Einstieg findet man in der Dokumentation [15].
- *Xojo*: Hierbei handelt es sich um ein RAD-Tool zur Anwendungsentwicklung für sehr unterschiedliche Zielsysteme. Der Schwerpunkt liegt auf einer beschleunigten Entwicklung mit Hilfe eines grafischen Designers und vorgefertigten Komponenten. Die Businesslogik wird mit Hilfe eines Basic-Dialektes erstellt. Gemäß der Dokumentation kann man mit Hilfe von Xojo auch Apps für iOS erstellen. Android-Apps stehen

auf der Roadmap für künftige Releases der Entwicklungsumgebung. Informationen zur App-Entwicklung für iOS findet man in der Onlinedokumentation unter [16].

- *NativeScript* [17]: Ein Open-Source-Framework zum Entwickeln von Apps für iOS und Android. Als Programmiersprachen werden JavaScript und TypeScript eingesetzt. Der Einsatz von Frameworks wie Vue oder Angular wird unterstützt. Es entstehen native Apps, die die APIs von Android und iOS verwenden. Zur Gestaltung des User Interface gibt es entsprechende Komponenten. Die Oberfläche kann in XML oder in JavaScript bzw. TypeScript kodiert werden.
- *React Native* [18]: React Native setzt auf React auf und erlaubt, native Apps mit JavaScript für Android und iOS zu realisieren.
- *Kotlin Multiplatform* [19]: Dieses Framework geht einen anderen Weg. Im Fokus der Cross-Platform-Entwicklung steht die Entwicklung der Geschäftslogik mit Hilfe einer gemeinsamen Codebasis. Und das UI? Das wird weiterhin nativ mit den Technologien von Android und iOS erstellt. Als Programmiersprache kommt Kotlin zum Einsatz, als Entwicklungsumgebung Android Studio.

Wenn Sie im Netz suchen, werden Sie noch weitere, durchaus interessante Ansätze finden, um eine mobile App für Android oder iOS zu erstellen. Nicht erwähnt haben wir hier Low-Code-Tools. Auch mit diesen Entwicklungstools kommt man bei speziellen Anforderungen ans Ziel. Für alle webbasierten Technologien gilt: Packt man die Web-App in einen nativen Container, kann man sie über die hybride Technologie dem nativen App-Feeling deutlich näherbringen.

Fazit

Lassen wir diesen Artikel mit einem allgemein bekannten Spruch

ausklingen: „Wer die Wahl hat, hat die Qual“. Das gilt in der Tat für die Auswahl der Entwicklungsansätze für das Mobile-App-Development. Doch diese Qual hat einen großen Vorteil: Entwickler:innen können sich den Technologiestack heraussuchen, der den eigenen Kenntnissen in Sachen Programmiersprache, Framework und bisherigen Erfahrungen am nächsten kommt. An erster Stelle der Auswahl wird i. d. R. der geforderte Funktionsumfang der App stehen. Ebenso ist es wichtig, wie oft man eine App für die Mobile-Systeme erstellen möchte. Ist es nur ein Nebenprodukt, dann tut es vielleicht eine webbasierte App. Regelmäßige App-Entwickler werden dagegen meist zu den Cross-Platform-Ansätzen greifen, um keine oder nur wenige Einschränkungen auf den Zielsystemen in Kauf nehmen zu müssen.

Muss die bestmögliche Plattformanpassung erreicht werden, dann bleibt nur die Entwicklung mit den Werkzeugen der Hersteller, in diesem Fall separat für iOS und Android. Für die meisten Zwecke dürfte man jedoch mit der Cross-Platform- bzw. hybriden Entwicklung gut bedient sein.



Dr. Veikko Krypczyk ist Softwareentwickler, Trainer und Fachautor und u.a. auf die Themen WinUI 3 und .NET MAUI spezialisiert. Sein Wissen gibt er über Fachartikel, Seminare und Workshops gern an Interessierte weiter und steht mit seiner Expertise auch für eine individuelle Unterstützung in Projekten zur Verfügung.



Elena Bochkor arbeitet am Entwurf und Design mobiler Anwendungen und Webseiten. Beginnend bei einer systematischen Nutzerforschung, über visuelle Prototypen bis hin zu einem barrierefreien Design gestaltet sie die User Experience moderner digitaler Produkte.

Trainings und Workshops zu diesen Themen können Sie unter <https://wp.larinet.com> anfragen und die Agenda vorab einsehen.

Links & Literatur

- [1] <https://developer.android.com/studio>
- [2] <https://developer.apple.com/xcode/>
- [3] <https://www.macincloud.com>
- [4] <https://dotnet.microsoft.com/en-us/apps/maui>
- [5] <https://dotnet.microsoft.com/en-us/apps/xamarin>
- [6] <https://flutter.dev>
- [7] <https://www.embarcadero.com/products/rad-studio>
- [8] <https://onsen.io>
- [9] <http://argelius.github.io/react-onsenui-redux-weather/demo.html>
- [10] <https://ionicframework.com>
- [11] <https://vaadin.com>

- [12] <https://rapidclipse.com>
 - [13] <https://cordova.apache.org>
 - [14] <https://docs.wisej.com/mobile>
 - [15] <https://www.qt.io/product/mobile-app-development/>
 - [16] https://documentation.xojo.com/getting_started/quickstarts/ios_quickstart.html
 - [17] <https://nativescript.org>
 - [18] <https://reactnative.dev>
 - [19] <https://kotlinlang.org/lp/mobile/>
-

Ein breiteres Spektrum – komplexe Rendering Patterns

Nachdem im ersten Teil die Grundlagen für ein besseres Verständnis davon gelegt wurden, wie Inhalte im Web geladen und gerendert werden, wenden wir uns nun komplexeren Ansätzen zu.

Im ersten Teil dieser Artikelserie haben wir die Geschichte des Webs und die daraus resultierende Entstehung verschiedener Rendering Patterns betrachtet. Dabei wurde auf die grundlegenden Patterns wie SSG (Static Site Generation), SSR (Serverseitiges Rendering) und CSR (Clientseitiges Rendering) eingegangen. Ihre Vor- und Nachteile wurden unter anderem mit Hilfe der drei Web-Vitals-Metriken TTFB (Time to First Byte), FCP (First Contentful Paint) und TTI (Time to Interactive)

unter sechs Schwerpunkten bewertet.

In diesem Teil werden darauf aufbauende Patterns behandelt, die die bisherigen Ansätze verbessern, neue Ideen hinzufügen oder Patterns kombinieren. Zuerst betrachten wir ein Pattern, das das statische Rendering verbessert. Anschließend werden wir uns mit Möglichkeiten beschäftigen, die Nachteile von clientseitig gerenderten Single-Page Applications (SPAs) zu reduzieren.

Incremental Static Regeneration

Incremental Static Regeneration (ISR), oder auch unter der Abkürzung iSSG (Incremental Static Site Generation) bekannt, ist eine erweiterte Variante des statischen Renderings (SSG), das wir bereits aus dem ersten Teil der Serie kennen. Es stellt eine Art Hybrid zwischen statischem und serverseitigem Rendering dar. ISR eliminiert einen der größten Nachteile von SSG: die lineare Skalierung der Build-Zeit mit der Anzahl der Seiten, wodurch in großen Projekten selbst kleinste Anpassungen einer Seite einen langwierigen Build auslösen.

Stattdessen werden die einzelnen Seiten nun inkrementell erzeugt. ISR ermöglicht es, neue Seiten nach dem Build einzubinden oder bestehende Seiten zu aktualisieren, indem die statische Generierung pro Seite zur Laufzeit angestoßen wird. Wird eine Seite angefordert, die zum Zeitpunkt des Build noch nicht generiert wurde, wird bei der klassischen statischen Variante in der Regel eine Fehlerseite mit HTTP404 (Not Found) zurückgegeben. Bei ISR hingegen wird die Generierung der Webseite bei der ersten Anfrage an den Server angestoßen (**Abb. 1**). Während des Generierungsprozesses sieht der Nutzer eine Ladeanimation oder einen Platzhalter. Da die Inhalte statisch sind, ist der Generierungsprozess sehr schnell. Sobald die Generierung abgeschlossen ist, wird die Seite ausgeliefert (geringes TTFB) und gecacht. Da alle Inhalte des HTML-Dokuments bereits gerendert sind, ergibt sich ein

schneller FCP und eine direkt interaktive Seite (TTI).

<https://phpconference.com/session-qualification/ipc-webentwicklung/?layout=contentareafeed&widgetversion=0&utmtrackerversion=1&seriesId=XWrH7HpMZMrnZNHid>

Bei der Generierung wird ein Zeitstempel hinterlegt, der angibt, wie lange die Seite aktuell ist. Jede weitere Anfrage an diese zuvor unbekannte Seite wird nun aus dem serverseitigen Cache mit der soeben generierten Seite beantwortet. Um die Aktualität der Inhalte zu gewährleisten, wird nach einer definierten Zeit (Zeitstempel plus konfigurierbarer Zeitraum) ab der nächsten Anfrage die Aktualität der Seite validiert. Dabei sollte der Zeitraum sinnvoll gewählt werden. Als Faustregel gilt: Je öfter sich der Inhalt der Seite ändert, desto kürzer sollte das Intervall sein. Ist die Seite im Cache nicht mehr aktuell ist, weil sich der Inhalt geändert hat, wird im Hintergrund eine Generierung der Seite angestoßen. In der Zwischenzeit werden Anfragen weiterhin aus dem Cache mit den veralteten Daten bedient. Dieser Ansatz wird als Stale-while-revalidate [1] bezeichnet. War die Generierung erfolgreich, wird der Cacheeintrag für diese Seite invalidiert und durch die neue Seite ersetzt. Falls ein Fehler aufgetreten ist, können Anfragen immerhin noch mit dem alten Eintrag beantwortet werden.

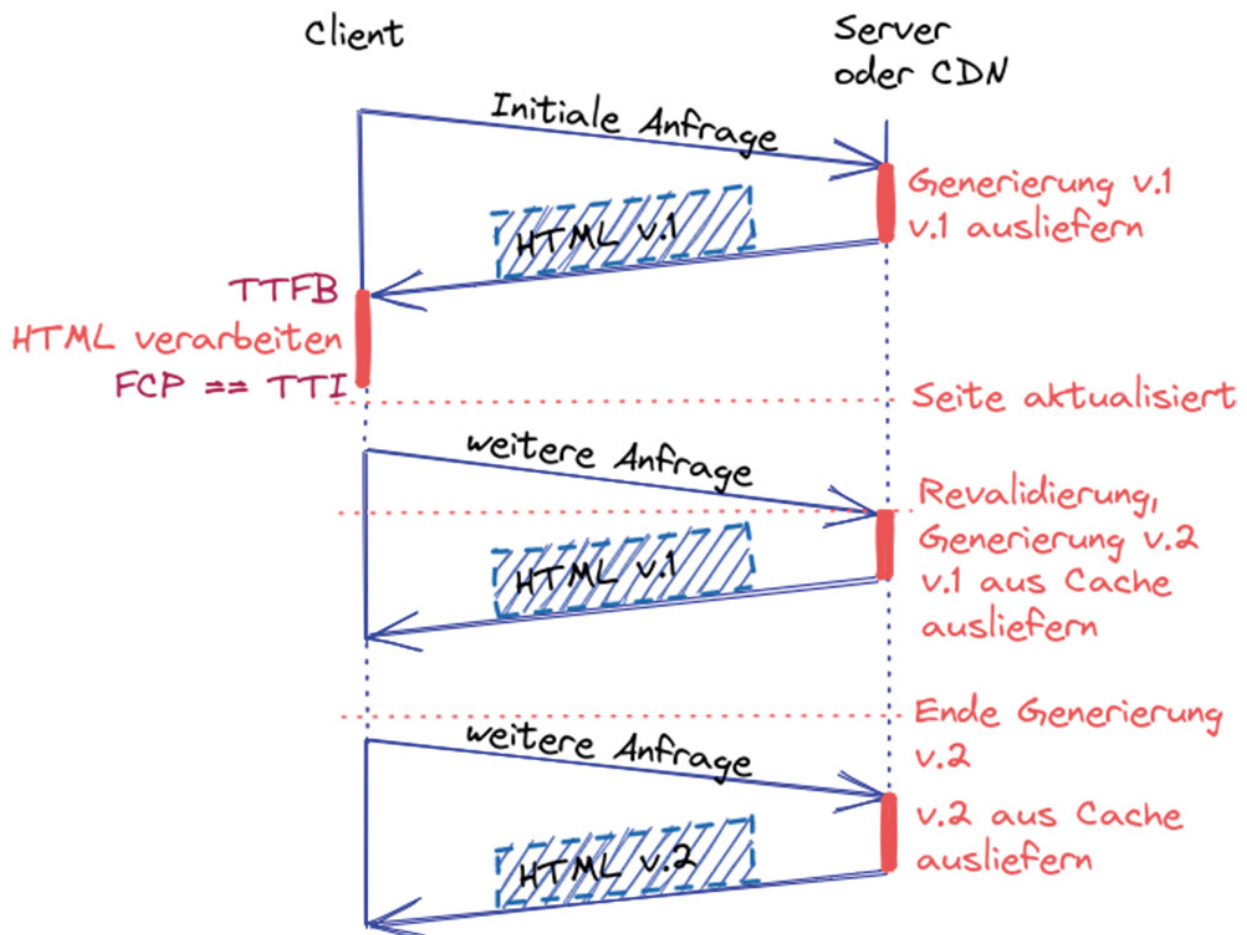


Abb. 1: Client-Server-Kommunikation von ISR mit Stale-while-revalidate-Mechanismus

Neben dieser passiven Art der Revalidierung existiert auch ein aktiver Ansatz. Dieser wird als On-Demand Revalidation bezeichnet. Damit wird es ermöglicht, den Cache manuell zu revalidieren. Dazu kann z. B. nach der Aktualisierung des Inhalts einer Webseite in einem CMS ein serverseitiges API aufgerufen werden, das die Revalidierung anstößt. Diese Variante sollte verwendet werden, wenn sich der Inhalt nur von Zeit zu Zeit ändert. Selbstverständlich können beide Varianten auch kombiniert eingesetzt werden.

Das inkrementelle Vorgehen hat den Vorteil, dass zur Build-Zeit nicht alle Seiten auf einmal gebaut werden müssen und Anpassungen schneller produktiv sind. Das soll am Beispiel einer E-Commerce-Webseite mit 100 000 Produkten demonstriert werden. Wir nehmen an, dass die Generierung einer Produktseite

im optimistischen Mittel lediglich zehn Millisekunden dauert. Ein kompletter Build würde also knapp 17 Minuten in Anspruch nehmen. Für eine Webseite, die schnell auf Preisanpassungen reagieren muss, könnte das bereits zu lang sein. Eine Lösung wäre, nur die 1 000 populärsten Produkte zum Zeitpunkt des Build mit ISR zu generieren. Die restlichen 99 000 Produkte können just in time erzeugt werden, sobald sie angefordert werden. Das würde die Build-Zeit auf wenige Sekunden reduzieren.

Ein weiterer Vorteil von ISR ist die Persistierung der Seiten zwischen verschiedenen Deployments. Das bedeutet, dass es möglich ist, sofort ein Rollback durchzuführen, ohne die generierten Seiten auszutauschen. Beispielsweise könnte nach dem Deployment mit der ID A auf der Webseite in Version 1 ein Tippfehler festgestellt werden. Dieser wird im CMS korrigiert. Durch die automatische Revalidierung ist kein erneutes Deployment notwendig. Es wird automatisch die aktualisierte Webseite in Version 2 erzeugt und im Cache abgelegt. Anschließend wird ein Feature entwickelt und in einem weiteren Deployment mit ID B bereitgestellt. Dieses Feature ist jedoch fehlerhaft. Daher geschieht ein Rollback zum Deployment A. Obwohl der Schreibfehler zum Zeitpunkt des Deployments A bestand, ist er nach dem Rollback nicht mehr vorhanden, da die Seite mit Version 2 unabhängig vom Deployment persistiert wurde. Dieser Mechanismus kann jedoch auch als Nachteil ausgelegt werden, da damit atomare, unveränderliche (immutable) Deployments nicht mehr garantiert werden können [2].

Ein Problem bei diesem Pattern ist, dass die ausgelieferten Seiten nicht für jeden Nutzer dieselben sind. Aufgrund des Stale-while-revalidate-Mechanismus kann es vorkommen, dass zwischen der Änderung der Webseite, der erneuten Validierung und der Generierung der neuen Version veraltete Inhalte ausgeliefert werden. In **Abbildung 1** liefert die zweite Antwort des Servers eine veraltete Version der Seite aus. Erst mit der

nächsten Anfrage wird der aktuelle Inhalt ausgeliefert. Das ist nicht optimal für den SEO-Score, tritt jedoch nur bei einem Bruchteil der Anfragen auf. Es hat aber größere negative Auswirkungen auf die User Experience und die Developer Experience, da Nutzer verschiedene Inhalte zu sehen bekommen und somit auch das Debugging erschwert wird. Schließlich erhöht dieses Pattern im Gegensatz zum klassischen statischen Rendering die Komplexität sowohl auf der Seite der Infrastruktur als auch auf der Seite der Developer Experience.

Aus diesen Gründen eignet sich Incremental Static Regeneration nicht für kleine Projekte. Es kann sogar völlig unnötig sein, wenn das Intervall für die Revalidierung größer als die gesamte Build-Zeit ist. Seine Stärken spielt es bei einer großen Anzahl statischer Seiten aus, z. B. bei E-Commerce-Webseiten. Es ist auch denkbar, dass nur stark frequentierte Seiten, wie z. B. Landing Pages, vorgerendert werden und die restlichen Seiten zur Laufzeit generiert werden. Ursprünglich stammt dieses Pattern vom auf React aufsetzenden Metaframework Next.js ab, findet jedoch auch Einzug in weitere Frameworks, wie beispielsweise dem Pendant Nuxt im Vue-Umfeld. Zusammengefasst sehen Sie die Vor- und Nachteile von ISR in Tabelle 1 und **Abbildung 2**.

Vorteile	Nachteile
performant	wenig Dynamik
niedrige TTFB	keine native User Experience
TTI = FCP	erhöhte Komplexität
SEO-freundlich	potenziell veraltete Seiten
CDN-fähig (Skalierbarkeit, Cache)	keine atomaren Deployments
mit deaktiviertem JS nutzbar	
wenige Angriffsvektoren	

Vorteile	Nachteile
kürzerer Build-Prozess als SSG	
geringerer Ressourcenverbrauch als SSG	
Fallback durch Cache	
Persistierung der Seiten unabhängig vom Deployment	

Tabelle 1: Vor- und Nachteile von ISR

■ Incremental Static Regeneration

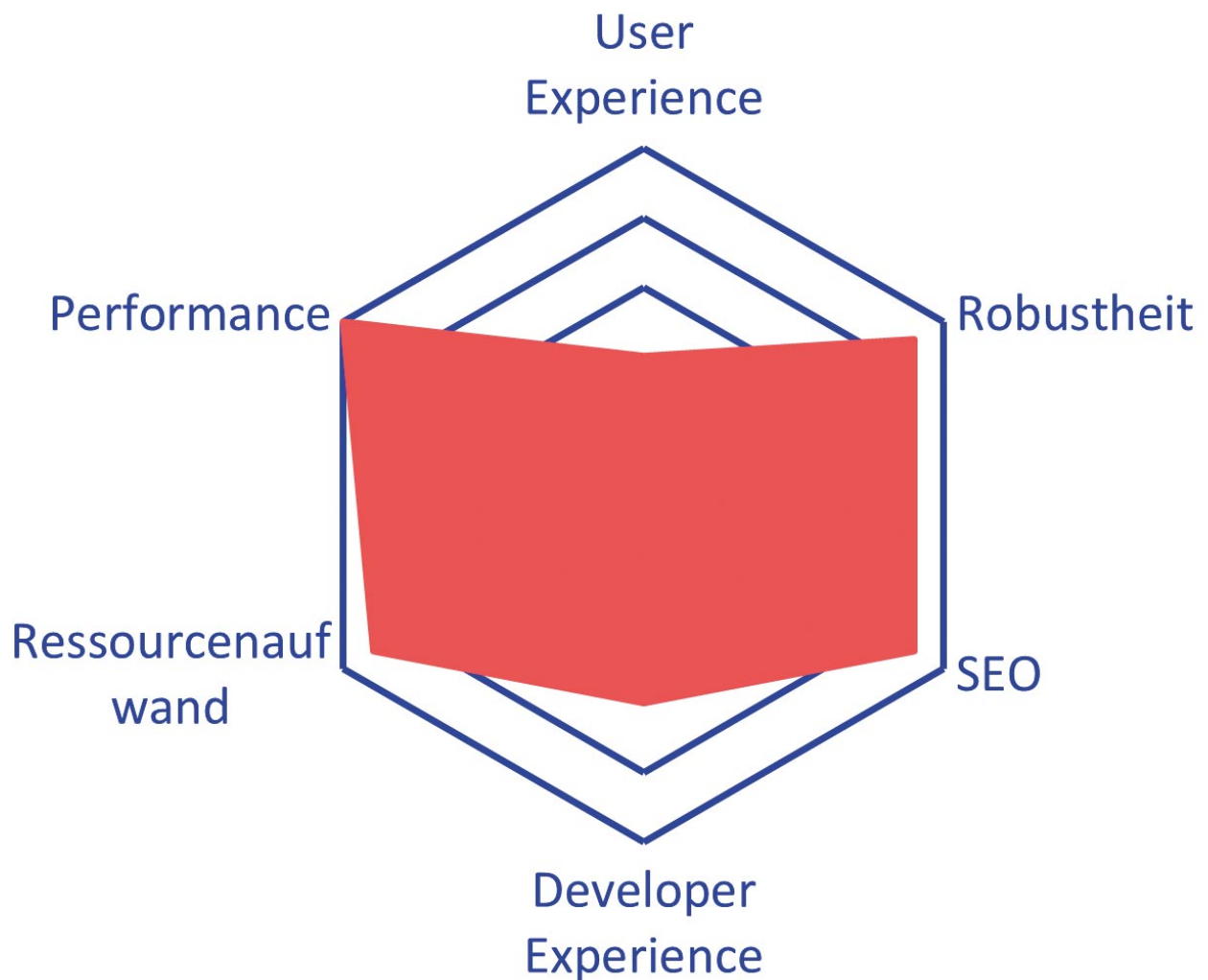


Abb. 2: Bewertung ISR: Developer und User Experience steigen im Vergleich zu SSG

Clientseitiges Rendering mit Prerendering

Einer der größten Nachteile von clientseitig gerenderten SPAs ist ihre schlechte SEO-Unterstützung. Das initiale HTML-Dokument enthält nur eine leere Hülle mit einem Einstiegspunkt für die JavaScript-Applikation. Crawler können daher keinen auswertbaren Inhalt finden. Um den SEO-Score zu verbessern, müsste bei der initialen Antwort des Servers mehr interpretierbarer Inhalt im Dokument geliefert werden.

Daher gibt es sogenannte Prerender Services oder -Tools, die als Middleware oder Plug-in in die bestehende Anwendungsinfrastruktur integriert werden können. Sie nutzen den Code der SPA, um daraus eine statische Version zu rendern. Dieses statische HTML wird dann initial ausgeliefert (**Abb. 3**). Anschließend wird der Java-Script-Code geladen, der die Kontrolle übernimmt.

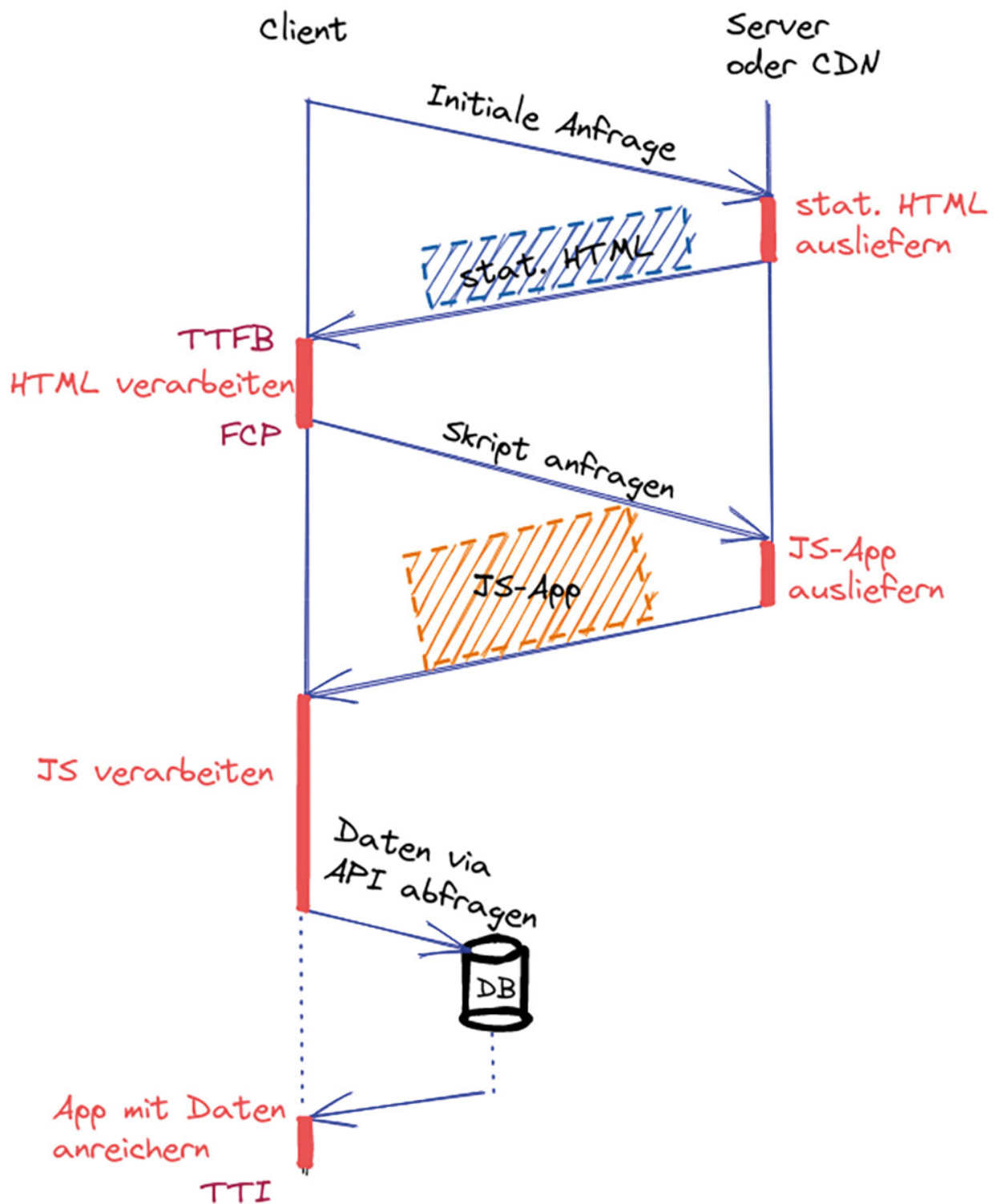


Abb. 3: Zuerst wird das statische HTML ausgeliefert, anschließend die SPA geladen

Das Prerendering kann zu unterschiedlichen Zeitpunkten erfolgen. Entweder vorab zur Build-Zeit oder automatisiert in einem definierten Intervall. Bei der Variante zur Build-Zeit wird die SPA einmalig vorgerendert. Dadurch wird der Build-

Prozess aufwendiger und die Inhalte können nur aktualisiert werden, wenn die Anwendung neu gebaut wird. Dieser Nachteil wird abgemildert, wenn die Anwendung stattdessen in einem festgelegten Intervall neu gerendert wird.

Da es sich beim Prerendering-Ansatz um eine Kombination aus dem CSR- und dem SSG-Pattern handelt, gelten größtenteils deren Vor- und Nachteile. So ist es beispielsweise nicht möglich, dynamische Inhalte vorab zur Build-Zeit zu rendern.

Prerendering steigert den SEO-Score. Darüber hinaus verbessert es die Zeit zum FCP, da die ersten Inhalte direkt nach der Interpretation des HTML-Dokuments gerendert werden, anstatt eine leere Seite anzuzeigen. Außerdem wird die Seite dadurch resistenter, da auch ohne JavaScript Inhalte gerendert werden und Inhalte nicht pro Anfrage gerendert werden müssen.

Durch die Verwendung eines externen Rendering-Diensts entsteht jedoch eine weitere Abhängigkeit, die außerhalb der eigenen Kontrolle liegt. Zusätzlich kann die Generierung zur Laufzeit zu einer längeren Antwortzeit (TTFB) führen, abhängig davon, wie schnell der Prerender Service ist.

Dieses Pattern kann zusätzlich um eine Laufzeitkomponente erweitert werden. Abhängig vom anfragenden User Agent wird unterschieden, ob die clientseitig gerenderte oder die statische Webseite ausgeliefert werden soll (**Abb. 4**). So erhält ein Crawler, für den JavaScript unerheblich ist, die statische Variante und ein normaler Nutzer die statische Seite mit dem SPA-Code. Das wird auch Dynamic Rendering [3] genannt.

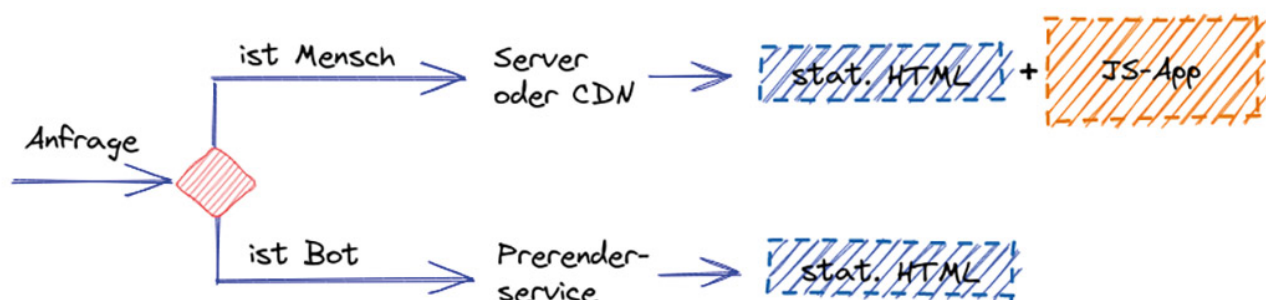


Abb. 4: Dynamic Rendering mit Prerender Service

Prerendering eignet sich, wenn SEO-Verbesserungen an einer bestehenden SPA vorgenommen werden sollen und eine Migration zu einer serverseitigen Lösung nicht im Verhältnis zum Aufwand steht. Vor allem dann, wenn die Seiten statische Daten beinhalten und sich deren Inhalte nur wenig ändern. Ein Dienst, der dieses Pattern anbietet, ist prerender.io [4]. In Tabelle 2 und **Abbildung 5** sehen Sie die Vor- und Nachteile von Prerendering auf einen Blick.

Vorteile	Nachteile
gute User Experience	TTI > FCP (verbesserte Zeit zum FCP gegenüber CSR)
Trennung zwischen Client- und Servercode	verteiltes mentales Modell
CDN-fähig	potenziell längere TTFB als bei CSR
SEO-freundlich	viele Angriffsvektoren
nachrüstbar	Abhängigkeit von externen Services
mit deaktiviertem JS eingeschränkt nutzbar	keine Dynamik

Tabelle 2: Vor- und Nachteile von Prerendering

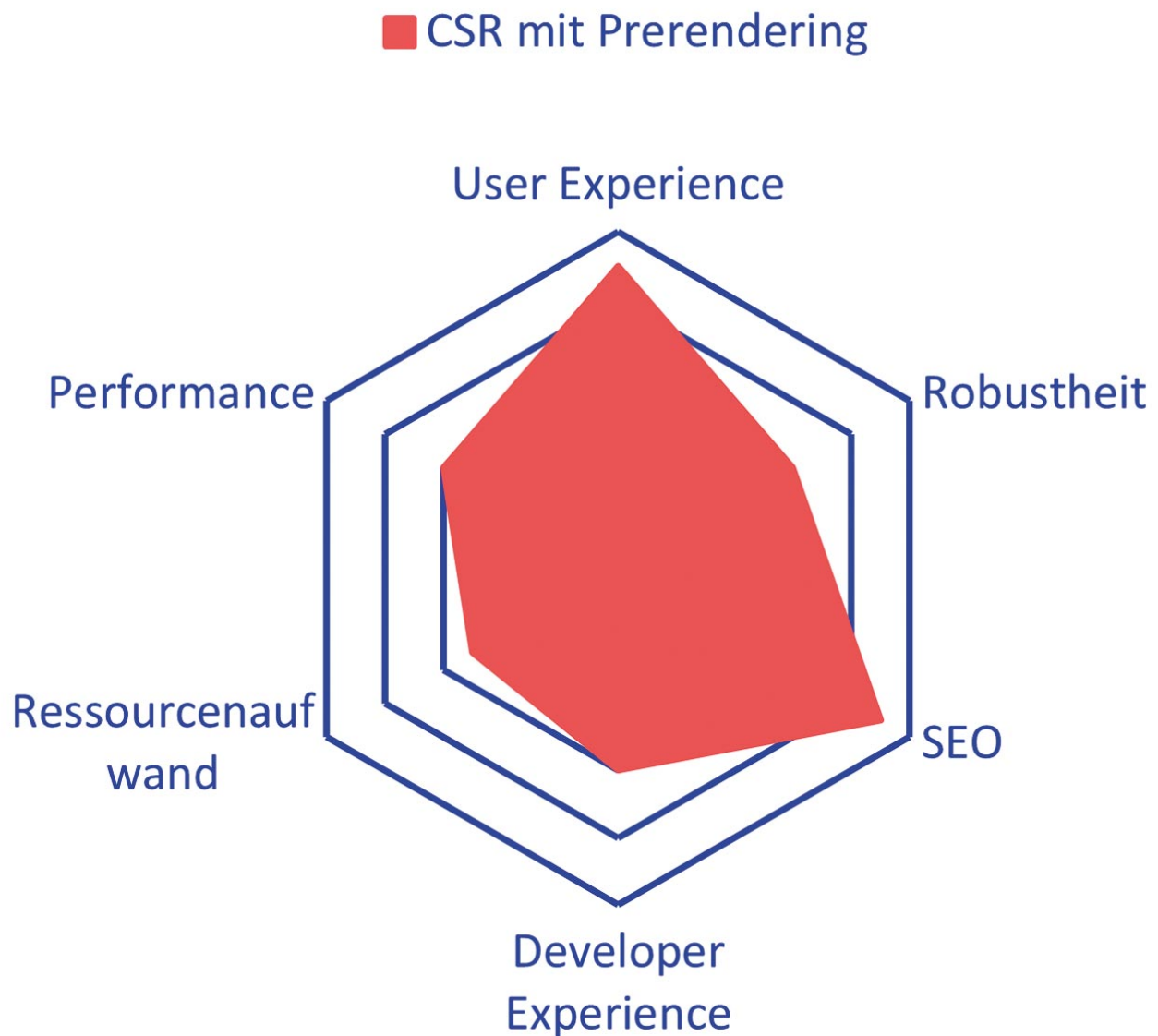


Abb. 5: Prerendering verbessert den SEO-Score erheblich

Serverseitiges Rendering mit Hydration

Neben dem Prerendering-Ansatz zur Build-Zeit gibt es noch eine weitere Möglichkeit, die Probleme von clientseitig gerenderten SPAs zu lösen: serverseitiges Rendering mit einem Hydrationsschritt. Hier findet das Rendering zur Laufzeit statt. Oft wird dieser Prozess auch als Hydration oder isomorphes bzw. universales Rendering bezeichnet. Das Pattern ist eine Kombination aus SSR und CSR.

Die Bezeichnung isomorph ist deshalb zutreffend, weil sowohl auf dem Server als auch auf dem Client der SPA-Code gerendert

wird. Begonnen wird dabei auf der Serverseite (**Abb. 6**). Dazu werden serverseitig die notwendigen Daten geladen und das DOM bzw. der Komponentenbaum der Anwendung damit angereichert. Anschließend wird daraus das HTML-Dokument gerendert und an den Client ausgeliefert. So erhält der Client schnell eine Seite mit interpretierbarem Inhalt, anstatt eine leere Seite anzuzeigen.

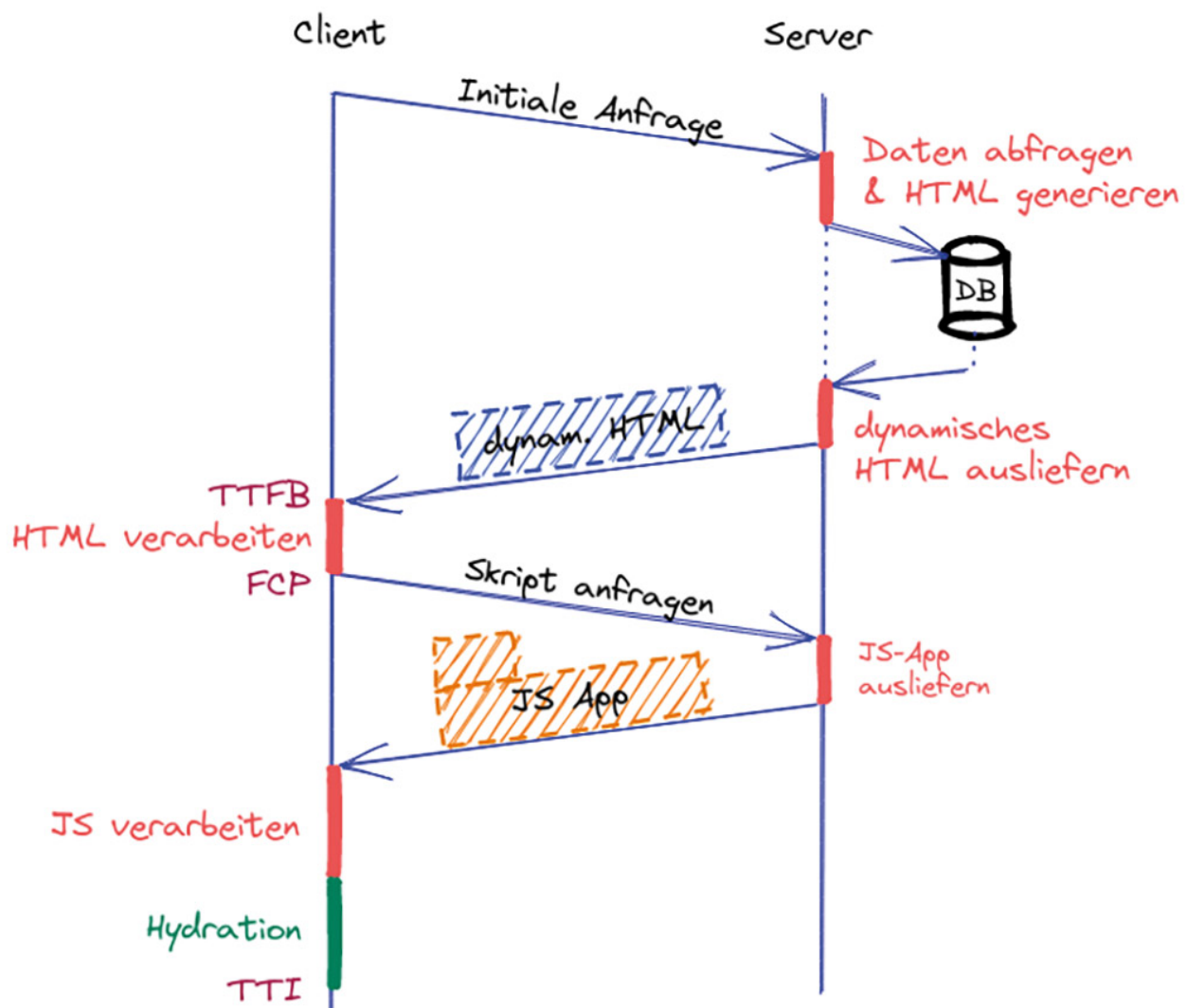


Abb. 6: Client-Server-Kommunikation – neu ist der Hydration-Schritt am Ende

Im zweiten Schritt wird clientseitig der JavaScript-Code für die SPA geladen und der Komponentenbaum erzeugt. Im Anschluss startet die Hydration, die die servergerenderte Anwendung auf dem Client wiederherstellt. Dazu wird die Initialisierung der gesamten Anwendung erneut abgespielt. Dabei wird das DOM mit

Event Handlern versehen und Zustände einzelner Komponenten wiederhergestellt. Es ist, als würde das zuvor „trockene“ HTML mit dem „Wasser“ der Interaktivität und des Event Handling hydriert werden [5]. Sobald dieser Prozess für alle Komponenten abgeschlossen ist, verhält sich die Anwendung wie eine klassische SPA, da ab diesem Zeitpunkt der JavaScript-Code die Kontrolle übernimmt.

Die Vorteile dieses Ansatzes liegen auf der Hand. Statt wie bei einer rein clientseitig gerenderten SPA können dynamische Inhalte schneller angezeigt werden und sind auch für Suchmaschinen indizierbar. Die Größe des Java-Script Bundle hat kaum Einfluss auf die FCP-Metrik, da bereits vor dem clientseitigen Laden des Bundles serverseitig generierte Inhalte gerendert werden. Nach dem Laden bestehen weiterhin die Vorteile des clientseitigen Rendering, wie z. B. eine native User Experience. Somit haben wir alle unsere Probleme gelöst. Oder? Der Schein trügt. Hydration ist nämlich ein ineffizienter und fehleranfälliger Prozess. Große Teile der Webseite sind doppelt vorhanden. Einmal im servergerenderten HTML-Dokument und einmal im JavaScript Bundle für den Client. Diese Doppelung erhöht den Datenverbrauch und die Auslastung des Clients, da sowohl das HTML als auch das JavaScript geladen, interpretiert und ausgeführt werden müssen. Je nach Endgerät kann dieser Vorgang längere Zeit in Anspruch nehmen.

Da die clientseitige Anwendung den vom Server generierten Code übernimmt, muss der Komponentenbaum des Servers genau mit dem des Clients übereinstimmen. Um das zu gewährleisten, muss vor der Hydration abgewartet werden, bis das JavaScript vollständig geladen wurde. Ansonsten kann es im besten Fall zu einer Verlangsamung der Applikation führen, im schlimmsten Fall könnten beispielsweise Event Handler für das falsche Element initialisiert werden [6]. Auch können Situationen auftreten, bei denen das initial vom Server erzeugte DOM zerstört wird und komplett neu gerendert werden muss.

Das größte Problem stellt das sogenannte Uncanny Valley dar.

Solange die SPA clientseitig nicht initialisiert ist (d. h., der Hydration-Schritt für alle Komponenten abgeschlossen ist), kann der Nutzer nicht mit der statischen Seite interagieren, obwohl diese bereits durch das serverseitig generierte HTML interaktiv erscheint.

Aus diesen Gründen ist Hydration ein komplexes Thema [7] und es werden inkrementelle Verbesserungen vorgenommen. Beispielsweise arbeitet das React-Team seit einigen Jahren an der Suspense-Architektur, die es erlaubt, nur Teile des Komponentenbaums zu rendern und zu hydrieren. Dieser Ansatz ist unter [5] sehr gut beschrieben. In den folgenden Abschnitten werden daher zwei weitere Varianten des Hydration-Patterns betrachtet, wodurch die Probleme reduziert werden sollen: progressive und partielle Hydration.

Hydration ist heutzutage bei vielen Frontend-Frameworks der Standard für die Entwicklung von Webapplikationen. Durch die dynamische Berechnung zur Laufzeit ist es weitläufig einsetzbar. Trotzdem ist es keine One-size-fits-all-Lösung. Für überwiegend statische Seiten könnte beispielsweise ISR genutzt werden. Alle großen JavaScript-Frameworks wie React oder Vue unterstützen Hydration. In der Regel wird jedoch auf die darauf aufbauenden Metaframeworks wie Next.js oder Nuxt zurückgegriffen, da diese Frameworks den gesamten Prozess rund um Hydration automatisieren. Zusammengefasst sehen Sie die Vor- und Nachteile von Hydration in Tabelle 3 und **Abbildung 7**.

Vorteile	Nachteile
performant	potenziell hohe TTFB
SEO-freundlich	TTI > FCP (Uncanny Valley)
gute User Experience	initiales Rendering abhängig von Konnektivität
dynamische, personalisierte Inhalte	Performance abhängig von Nutzerzahl und Serverstandort

Vorteile	Nachteile
mit deaktiviertem JS eingeschränkt nutzbar	ineffizient und fehleranfällig
	mangelnde CDN-Fähigkeit
	viele Angriffsvektoren

Tabelle 3: Vor- und Nachteile von Hydration

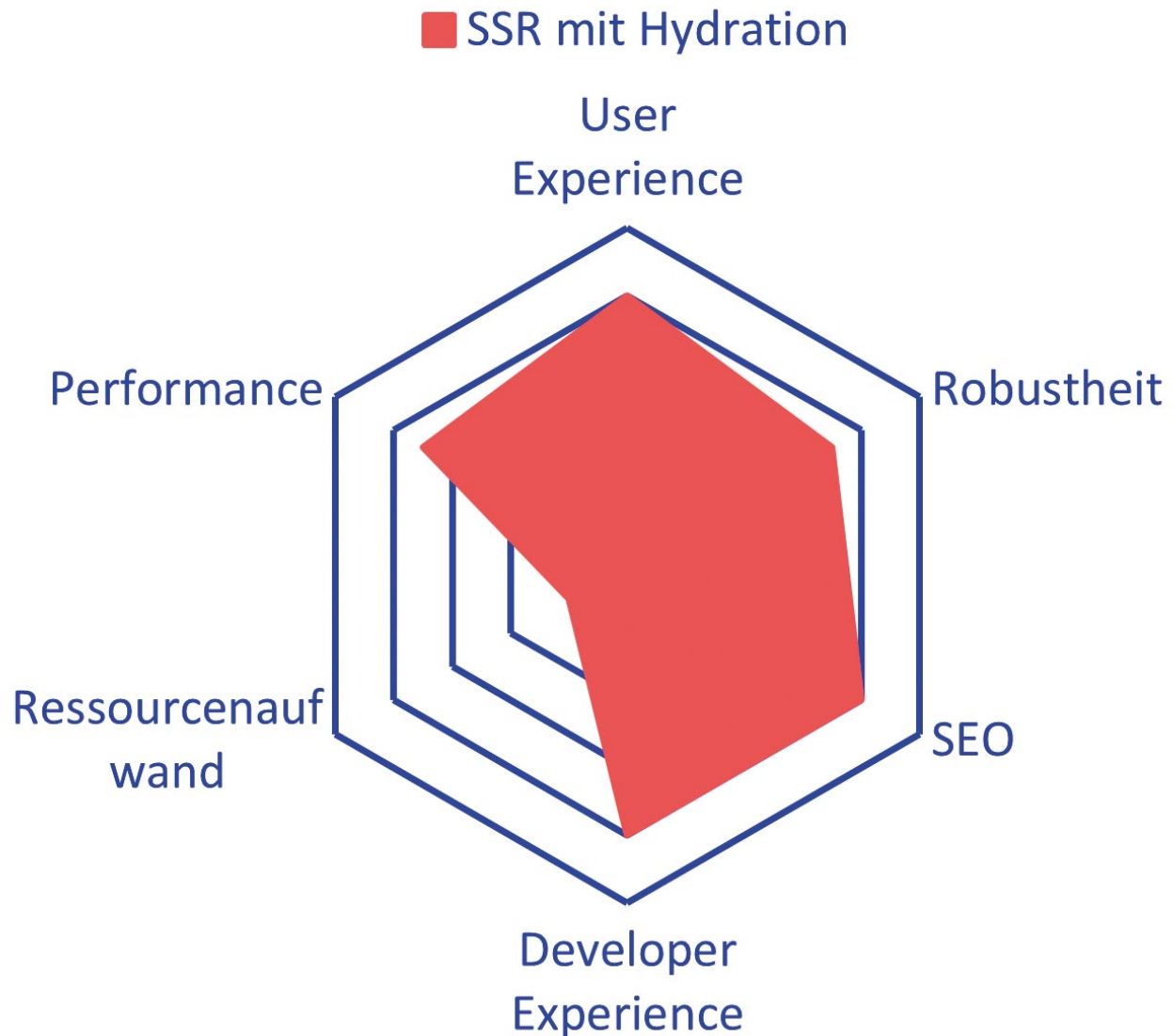


Abb. 7: Hydration ist ein guter Allrounder, wäre da nicht die verschwenderische Ressourcennutzung

Progressive Hydration

Wie wir gesehen haben, stellt die initiale Hydration aller Komponenten ein großes Problem dar. Es ist daher naheliegend

zu versuchen, diesen Aufwand zu reduzieren. Ein Ansatz dazu kann Lazy Loading sein. Diese Idee wurde bereits 2016 in einem Blogbeitrag [8] unter dem Namen „Progressive Booting“ beschrieben.

Anstatt den gesamten Komponentenbaum auf einmal zu hydrieren, werden zunächst nur die nötigsten Komponenten initialisiert. Die Hydration von weniger relevanten Zweigen im Komponentenbaum kann anfangs ausgesetzt und zu einem späteren Zeitpunkt angestoßen werden. Als Auslöser können hierfür der Viewport, die Interaktionswahrscheinlichkeit, oder die aktuelle Auslastung des Clients dienen. Beispielsweise kann eine Footer-Komponente, die sich außerhalb des Viewports befindet, erst zu dem Zeitpunkt hydriert werden, wenn sie sichtbar wird (**Abb. 8**). Hierfür wird der JavaScript Chunk für den Footer erst zu diesem Zeitpunkt angefordert.

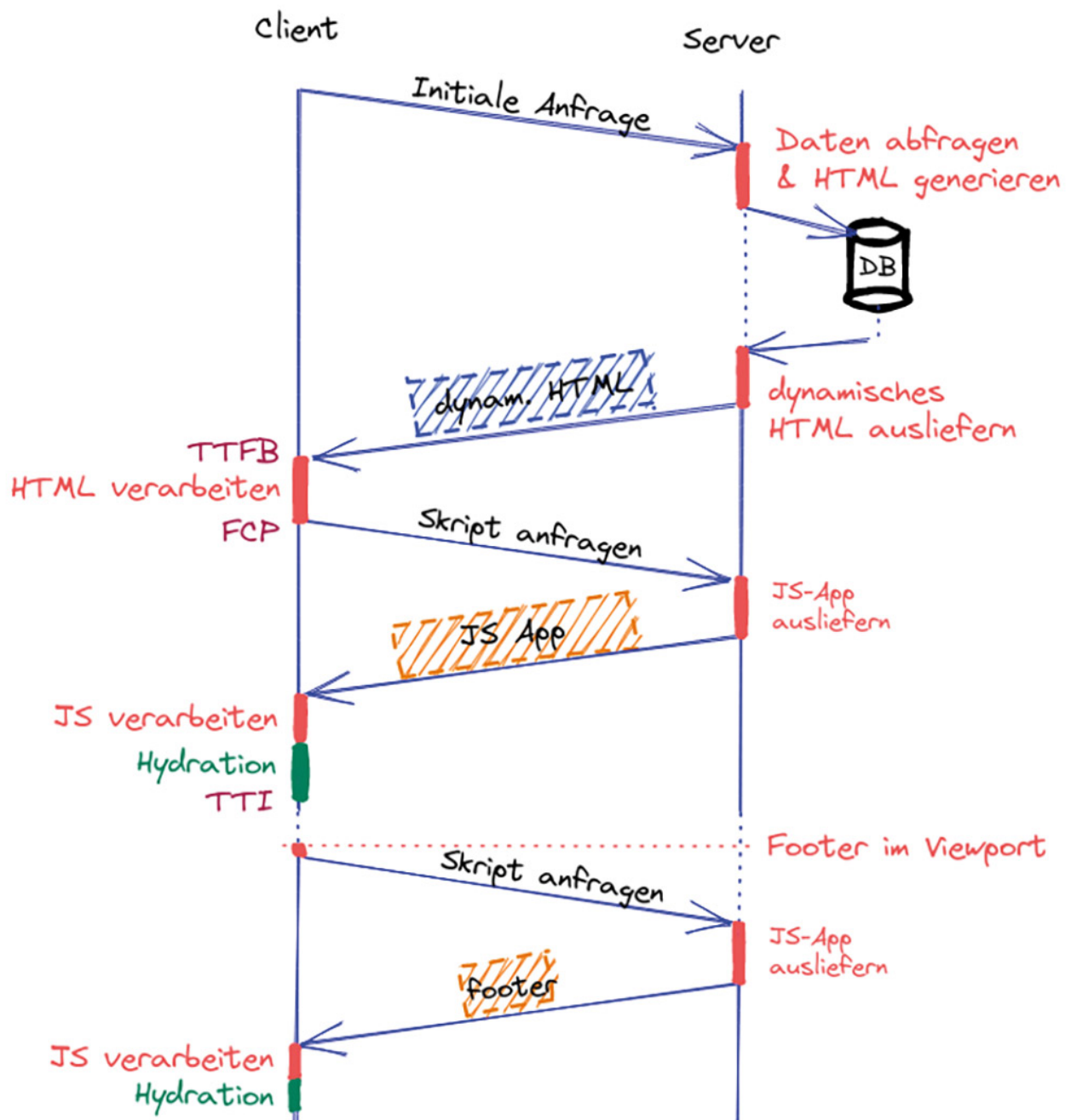


Abb. 8: Die Hydration für den Footer findet erst zu einem späteren Zeitpunkt statt

Für die Aufteilung der Applikation in einzelne Chunks müssen geeignete Grenzen und Prioritäten definiert werden. Diese können unter anderem vom Entwickler vorgegeben werden. Ein Beispiel hierfür sind Astros Clientdirektiven [9], die pro Komponente definiert werden können. Durch diesen progressiven Ansatz verringert sich die initiale Größe des JavaScript-Bundles, da bei der ersten Anfrage nur noch das Notwendigste versendet wird. Somit reduziert sich auch das Uncanny Valley,

da es die TTI für die Hauptkomponenten verbessert.

Wahre progressive Hydration ist aufgrund der vielen Fallstricke und Komplexität schwer zu erreichen. Ein Großteil der heutigen Frameworks basiert immer noch auf einem Top-down-Ansatz bei der Hydration, da diese Frameworks ursprünglich dazu gedacht waren, eine SPA zu bauen. Dadurch ist nur ein einzelner Einstiegspunkt für die Hydration gegeben. Somit muss ein großer Teil des Codes geladen und ausgeführt werden, selbst wenn lediglich eine Komponente am Ende des Komponentenbaumes hydriert werden muss. Für den progressiven Ansatz wären jedoch mehrere unabhängige Einstiegspunkte besser. Optimalerweise genau an der Stelle im Komponentenbaum, die hydriert werden soll.

Außerdem gestaltet es sich in der Praxis als schwierig, die richtige Grenze für die Chunks zu ziehen. Aufgrund der Eltern-Kind-Beziehungen kann es sein, dass die Grenze an einer Stelle gezogen wird, die von einem anderen Teil des Baumes abhängig ist. Daher ist es notwendig, auch diesen Teil zu hydrieren. Neben diesen Punkten gibt es noch weitere, die eine gute Umsetzung von progressiver Hydration erschweren. Für interessierte Leser empfehle ich die Quellen [7] und [10].

Progressive Hydration kann überall dort eingesetzt werden, wo auch Hydration in Frage kommt und die verbesserte Performance und Ressourcennutzung notwendig sind. Ein Framework, das progressive Hydration implementiert, ist das bereits erwähnte Astro. Aber auch andere Frameworks aus dem JavaScript-Ökosystem bieten diesen Ansatz an. Die Vor- und Nachteile sehen Sie zusammengefasst in Tabelle 4 und **Abbildung 9**.

Vorteile	Nachteile
siehe Tabelle 3	siehe Tabelle 3
effizienter durch kleineres initiales JavaScript Bundle	erhöhte Komplexität

Vorteile	Nachteile
geringere TTI, Reduktion des Uncanny Valley	

Tabelle 4: Vor- und Nachteile von progressiver Hydration

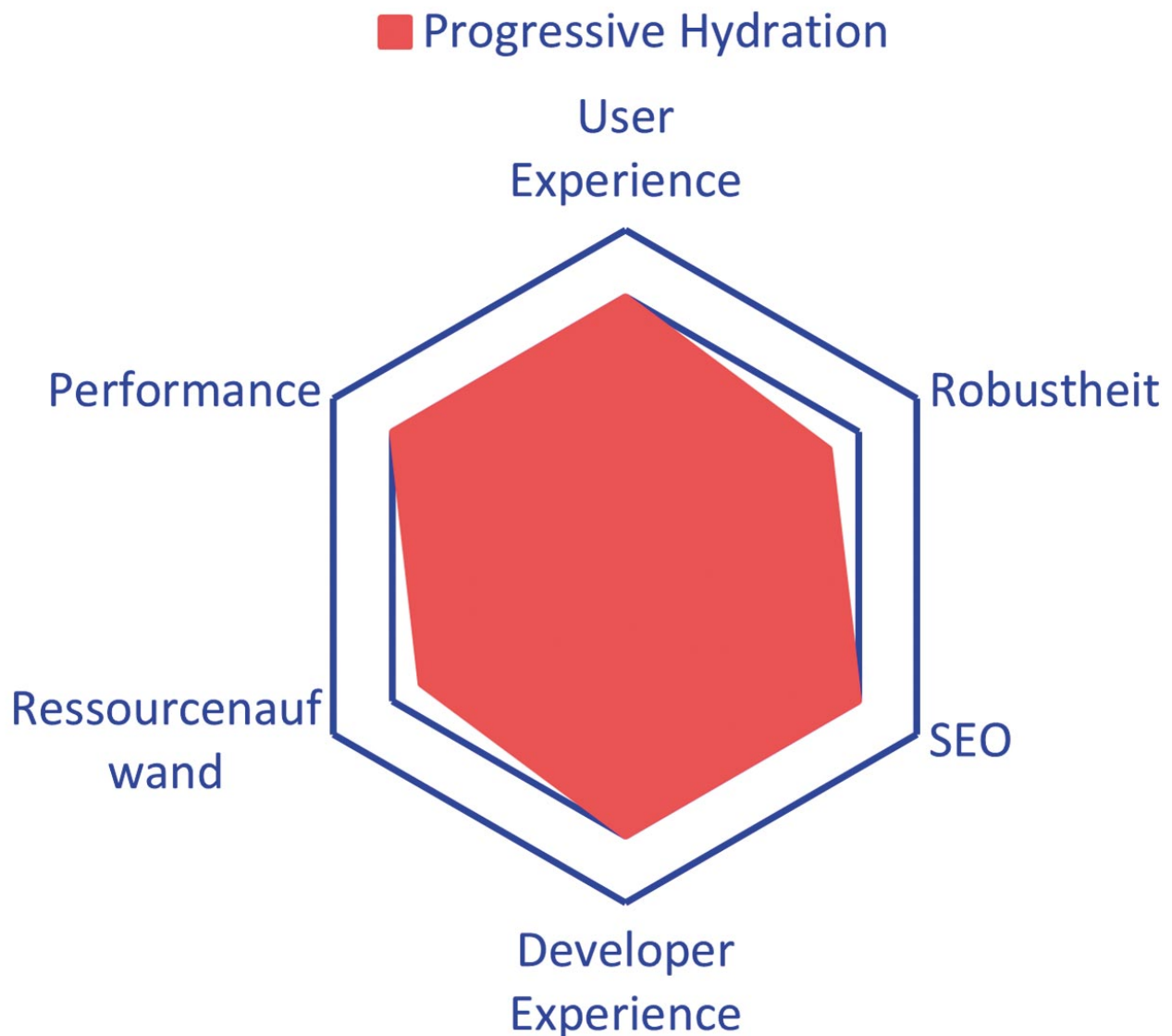


Abb. 9: Progressive Hydration reduziert den initialen Ressourcenaufwand

Partielle Hydration und Island Architecture

Partielle Hydration und Island Architecture werden häufig miteinander verwechselt oder als Synonyme verwendet. Die

Abgrenzung dieser beiden Begriffe ist nicht klar definiert. Das Ergebnis ist jedoch in beiden Fällen das gleiche. Im Gegensatz zum progressiven Ansatz, der sich auf den Zeitpunkt der Hydration auswirkt, ist es mit Hilfe der partiellen Hydration möglich, nur ausgewählte Teile des Komponentenbaums zu hydrieren.

Stellen Sie sich eine Anwendung vor, bei der nur zwei unabhängige kleine Teile interaktiv sind. Der Rest ist statisch. Wie zuvor bereits erläutert wurde, muss für diesen kleinen interaktiven Teil der gesamte Baum hydriert werden. Das ist ineffizient. Eine Lösung wäre, nur genau diese dynamischen Teile zu hydrieren (**Abb. 10**).

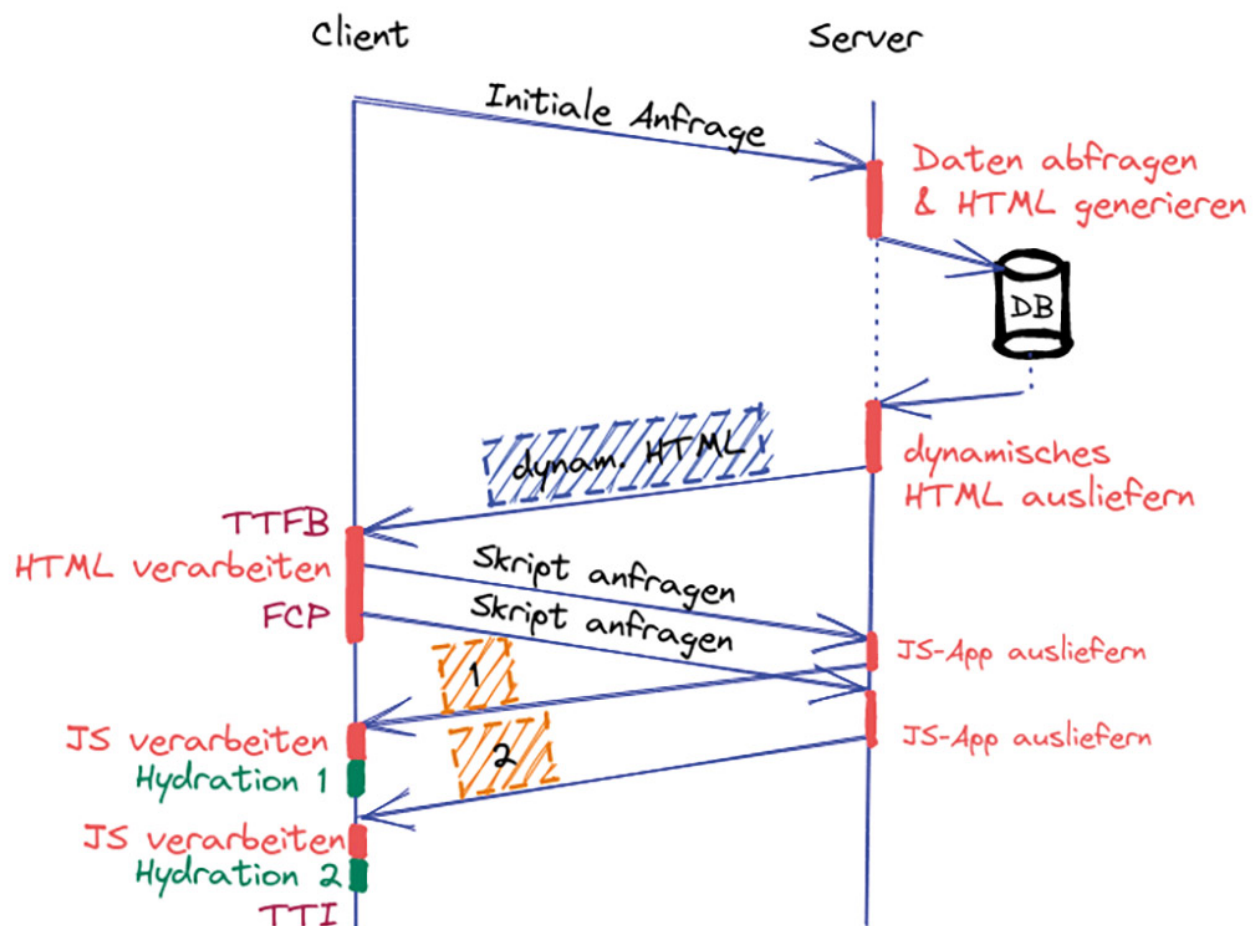


Abb. 10: Die Inseln werden unabhängig angefragt und verarbeitet

Dazu müssen geeignete Grenzen definiert werden. Das Ergebnis

ist einerseits ein statischer Teil aus HTML und andererseits dynamische Bereiche bestehend aus HTML und JavaScript. Letztere werden als Inseln bezeichnet. Damit verschwindet der zuvor bestehende Top-down-Ansatz des Komponentenbaumes, da nur noch unabhängige Teile mit verschiedenen Einstiegspunkten existieren. In unserem Beispiel hätten wir jetzt zwei unabhängige Inseln.

Für die Definition der Grenzen gibt es verschiedene Ansätze. Einige sind manueller Natur, andere arbeiten automatisch. Das bereits erwähnte Astro-Framework überlässt diese Aufgabe dem Entwickler. Dieser muss anhand der Clientdirektiven definieren, welche Teile nicht statisch sind. Andere Frameworks wie Fresh lösen das Problem, indem der Code für die Inseln in einem speziellen Ordner abgelegt werden muss. Wieder andere (z. B. Marko.js) erkennen die Inseln automatisch über einen Compiler.

Ähnlich wie beim progressiven Ansatz sinkt auch hier die Größe des clientseitigen Codes. Dadurch, dass sehr viel weniger teuer auszuführendes JavaScript ausgeliefert wird, ist die Anwendung schnell interaktiv. Das vermindert den Uncanny-Valley-Effekt. In Kombination mit progressiver Hydration kann dieser sogar ganz entfallen, da initial kein JavaScript mehr ausgeliefert werden muss. Die Seite ist somit wie beim statischen oder klassischen serverseitigen Rendering sofort interaktiv.

Da es sich um isolierte Inseln handelt, können sie auch unabhängig vom Rest der Seite geladen werden. Im Gegensatz zum Top-down-Ansatz verzögert keine Elternkomponente die Initialisierung einer Insel. Diese Vorteile werden jedoch durch eine erhöhte Komplexität erkaufte. Das manuelle Setzen von Grenzen kann fehleranfällig und kompliziert sein [11]. Einen Compiler zu schreiben, der dieses Problem löst, ist eine große Herausforderung, da die Architektur des Frameworks dies berücksichtigen muss.

Ein bereits erwähnter Vorteil der Inseln ist ihre Unabhängigkeit vom Rest der Seite, erschwert ist jedoch die Kommunikation zwischen den Inseln, z. B. um den globalen Anwendungszustand zu teilen. Im klassischen Ansatz können Daten über Props ausgetauscht werden. Da die Komponenten jedoch isoliert voneinander sind, ist dieser Mechanismus nicht mehr möglich. Daher wird eine zusätzliche Vermittlungsschicht benötigt, z. B. ein globaler Speicher für den Anwendungszustand, der von allen Inseln angesprochen werden kann.

Die partielle Hydration kann in ähnlicher Weise wie die progressive Hydration verwendet werden. Es ist jedoch zu bedenken, dass dieses Pattern die Developer Experience verringern kann, da es mehr Komplexität mit sich bringt. Außerdem ist dieses Pattern nicht für interaktionslastige Seiten geeignet, da solche sehr viele Inseln erfordern würden. Genutzt wird dieses Pattern bei den bereits erwähnten Frameworks Astro und Fresh. Die Vor- und Nachteile finden Sie in Tabelle 5 und **Abbildung 11** zusammengefasst.

Vorteile	Nachteile
siehe Tabelle 3	siehe Tabelle 3
effizienter durch kleineres JavaScript Bundle	erhöhte Komplexität
geringere TTI, Reduktion des Uncanny Valley	erschwerter Austausch zwischen Inseln
isolierte Komponenten	

Tabelle 5: Vor- und Nachteile partieller Hydration

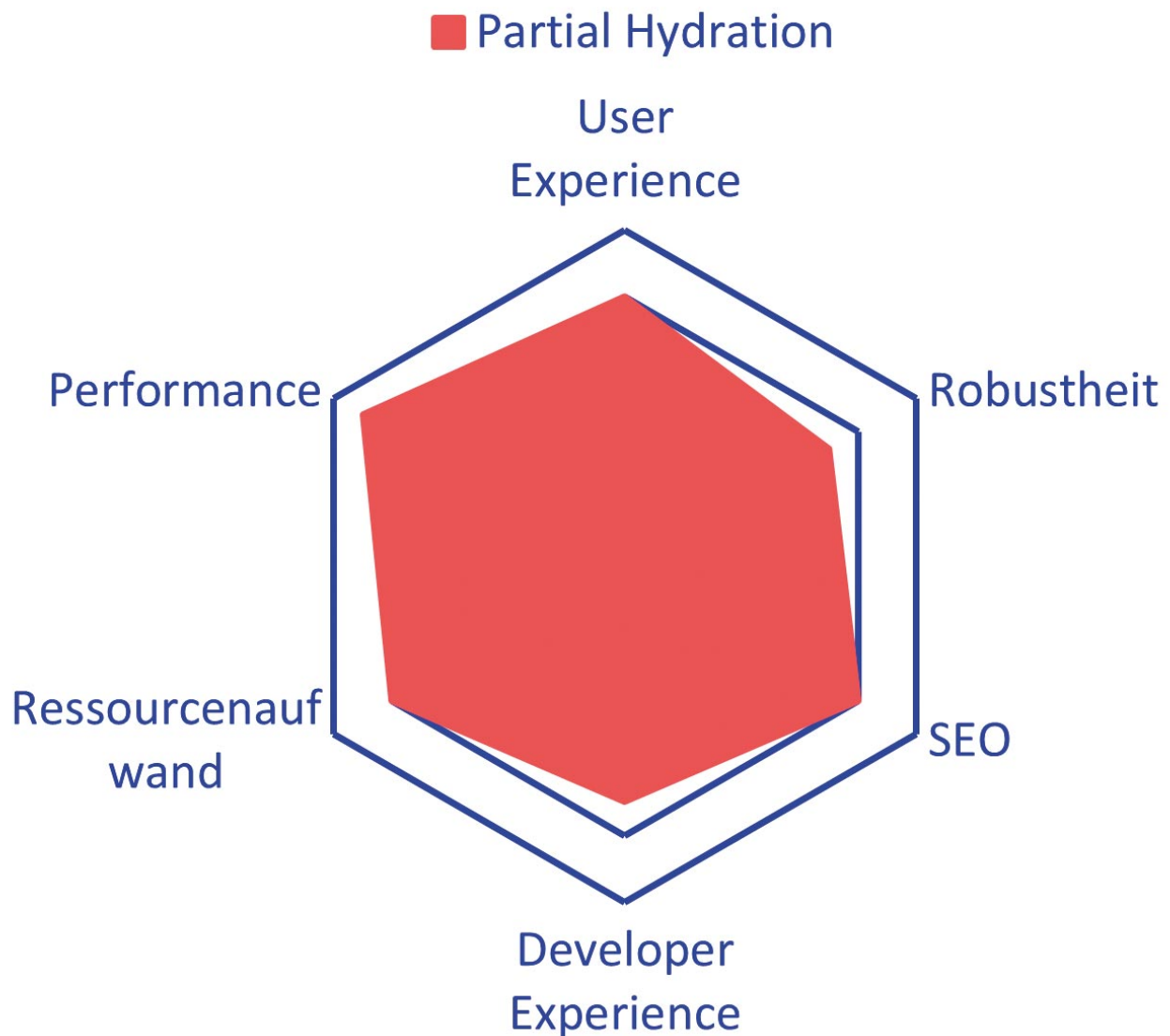


Abb. 11: Partielle Hydratation steigert die Performance und reduziert unnötige Daten

Fazit

Wie wir gesehen haben, versuchen alle diese Patterns, die Schwächen der drei grundlegenden Patterns zu mildern, indem sie verschiedene Ansätze daraus kombinieren und neue Ideen einbringen. Anhand der Bewertungen der jeweiligen Patterns ist zu erkennen, dass diese sich immer weiter verbessern. Besonders hervorzuheben sind die Sprünge in den Bereichen Developer Experience und User Experience. Das deckt sich auch mit der Entwicklung des Webs, die wir bereits im ersten Teil der Serie betrachtet haben.

Leider nimmt auch die Komplexität der Lösungen stetig zu. Das ist vor allem bei den Hydrations-Ansätzen zu beobachten, die heutzutage häufig den Standard in der Entwicklung mit JavaScript-Frameworks bilden. In letzter Zeit tauchen aber auch vermehrt neuartige Ansätze auf, die teilweise gänzlich ohne Hydratation auskommen, wie z. B. Resumability. Unter anderem auf dieses Pattern werden wir im kommenden und letzten Teil der Serie eingehen.



Julian Schäfer arbeitet als Full-Stack-Softwareentwickler bei der synyx GmbH & Co. KG in Karlsruhe. Daneben beschäftigt er sich mit Webtechnologien und versucht, dieses Wissen auch während seines Projektalltags weiterzugeben.

Links & Literatur

[1] <https://web.dev/stale-while-revalidate/>

[2] <https://www.netlify.com/blog/2021/03/08/incremental-static-regeneration-its-benefits-and-its-flaws/>

[3] <https://developers.google.com/search/docs/crawling-indexing/javascript/dynamic-rendering>

[4] <https://prerender.io>

[5] <https://github.com/reactwg/react-18/discussions/37>

[6] <https://www.joshwcomeau.com/react/the-perils-of-rehydration/>

[7] <https://dev.to/this-is-learning/why-efficient-hydration-in-javascript-frameworks-is-so-challenging-1ca3>

[8] <https://aerotwist.com/blog/when-everything-is-important-nothing-is/>

[9] <https://docs.astro.build/en/reference/directives-reference/#client-directives>

[10] <https://www.builder.io/blog/why-progressive-hydration-is-harder-than-you-think>

[11] <https://www.theguardian.com/info/2022/mar/25/react-islands-on-the-guardiancom>

Die drei größten Fehler bei agiler Transformation

Die drei Wege ins Glück, oder besser gesagt ins Unglück, sind, wie ein Unternehmen garantiert nicht agil wird. Wenn Sie keine schnellere Time-to-Market wünschen, nicht auf Ihre Kunden eingehen wollen, aber dennoch den Anschein erwecken möchten, agil zu sein und bemüht zu transformieren, folgen Sie einem der drei Wege – oder am besten folgen Sie allen dreien, nur um sicherzugehen.

Die letzten Jahre haben gezeigt: Wer auf dem Markt erfolgreich sein will, kommt um neue Formen der Unternehmensführung und Strukturierung nicht herum. In einer Zeit, in der die Märkte weniger vernetzt waren, Kunden nicht miteinander in Austausch gehen, also Erfahrungen zu Produkten teilen konnten, war es für die meisten Unternehmen einfacher. Auch konnten Methoden, die in einem anderen Unternehmen funktioniert haben, einfach auf das eigene übertragen werden. Nicht umsonst sind heute die meisten Unternehmen nach dem gleichen Muster aufgebaut, unterteilt nach Einheiten mit dem Ziel, möglichst effizient zu arbeiten. Und mit Effizienz ist in diesem Fall gemeint, effizient bzgl. der eigenen Aufgaben, nicht Richtung Kunde.

Dieses Verhalten, die Übernahme von Methoden, teilweise Best Practices genannt, lässt sich auch heute noch gut beobachten. Erfolgreiche Unternehmen kopieren einfach stumpf eine Idee und implementieren sie im eigenen Unternehmen, mit der Annahme, die gleichen Effekte wie im beobachteten Unternehmen zu erzielen. Die Implementierung einer Methode wird zum Selbstzweck. Manche Unternehmen gehen oberflächlich noch weiter und wählen einen der drei Wege ins Unglück.

- *Weg 1 – „Wir planen die Transformation“*: Natürlich muss es gut durchdacht und von A bis Z geplant sein. Wir können keine Transformation angehen, ohne diese geplant zu haben, wir müssen ja wissen, wann es losgeht, wann es beendet ist, also wann wir endlich agil sind und natürlich auch, was es kostet.
- *Weg 2 – „Wir brauchen ein eigenes Framework“*: Ist mal ein Plan gemacht (oder auch keiner), deutet sich in einem Unternehmen nur der Wille an, kommen sofort Bedenken und Bedenkenträger. Wie, wir wollen Scrum einführen? Das ist doch nur für Teams, wir sind hier ein Konzern und überhaupt geht das bei uns nicht, wir sind anders, wir müssen unser eigenes Framework entwickeln, das zu unseren Rahmenbedingungen passt.
- *Weg 3 – „Change-Team, mit Plan und Soll-Ist-Vergleich“*: Plan und eigenes Framework, natürlich muss der Plan auch kontrolliert werden und ein Soll-Ist-Vergleich stattfinden, dafür brauchen wir ein Change-Team, das sich dieser Thematik annimmt. Bitte das Change-Team auch damit beauftragen, ein Framework und einen Plan zu entwickeln. Und ganz wichtig: Das muss geheim bleiben, mit niemandem reden, das würde nur für Verunsicherung sorgen (es werden vielleicht Menschen entlassen oder Ähnliches). Wir müssen das erst gut durchdenken und dann alle mitnehmen, damit es auch klappt.

Ich glaube, die meisten von uns kennen solche Szenarien nur zu gut und haben erlebt, wie gut oder eher schlecht sie funktionieren.

Die ReG AG stand vor einigen Jahren vor einer ähnlichen Herausforderung (**Abb. 1**). Die Messe lief deutlich schlechter als erhofft, die Kunden waren überhaupt nicht begeistert vom vorgestellten Produkt. Der Vorstand hat zwei Maßnahmen ergriffen: Einen neuen Vertriebsvorstand geholt und Richard und seine designierte CTO, Silke, damit beauftragt, das Unternehmen zu transformieren. Richard kannte das laut eigener

Aussage bereits, für ihn war Agilität etwas für Hippies, die sich nicht an Zahlen messen lassen wollten. Daher forderte er vehement ein, dass die Transformation gründlich geplant wird.



Abb. 1: Die ReG AG steht vor der Herausforderung

Der erste Weg

Richard schlug folgenden Transformationsplan vor: Wir haben ein Kick-off-Meeting, zu dem alle kommen müssen. Im Kick-off bestimmen wir, welche Teams (falls sie schon existieren) jetzt nach Scrum oder anderen Frameworks arbeiten. Ganz wichtig: Wir überlassen es nicht den Teams, ihre Methode selbst zu wählen. Wir bestimmen, wer der Product Owner sein wird und wer der Scrum Master. Da die ReG AG hier keine Erfahrung hat, kaufen wir uns einen ein, oder schicken schnell jemanden auf die zweitägige Scrum-Master-Schulung, die reicht dann schon, um als Scrum Master im Unternehmen tätig zu sein. Wer wird der Product Owner? Nehmen wir am einfachsten den Projektleiter des

Teams, der hat das jetzt auch schon gut gemacht. Schulung für ihn sparen wir uns, das kann dann der Scrum Master richten, so anders ist Scrum nun auch wieder nicht.

Dann richten wir uns einige Meilensteine ein, um zu schauen, wie das Team oder die Teams nach Scrum arbeiten, in die ersten Reviews werden die Chefs eingeladen, die dann absegnen, dass die Arbeit gut, also richtig gemacht wurde. Die Vorschläge vom Scrum Master, auch Kunden einzuladen, wischen wir mit dem Kommentar vom Tisch, dass das bei uns nicht geht, wir sind die Experten, nicht die Kunden.

Der zweite Weg

Nachdem wir nun über den ersten Weg gesprochen haben, schauen wir uns den zweiten Weg zum (Un-)Glück an: Wir entwickeln unser eigenes Framework. Warum? Nun, Scrum kommt aus der Softwareindustrie und wir sind kein Softwareunternehmen, und selbst wenn wir eins sind, ging das vielleicht in Amerika, aber in Deutschland geht das nicht, schon allein wegen unserer Qualitätsansprüche; wir müssen schließlich am Ende genau wissen, was rauskommt. Also bilden wir ein Team aus Experten, kaufen Expertise hinzu, lesen viele Bücher.

Wir nehmen uns den Scrum Guide und schauen, was bei uns funktionieren könnte und was nicht. Wir ignorieren mal die Frage, ob es dann überhaupt noch eine Transformation gibt oder ob wir nur versuchen, Scrum in unsere Prozesse zu pressen (es sollte doch eher umgekehrt sein, die Prozesse und aktuellen Verhalten hinterfragen, um an dieser Stelle besser zu werden). Nein, wir schauen uns Scrum an und ändern es ab, und da Scrum nur für Teams ist, brauchen wir mehr. Wir nehmen z. B. SAFe. SAFe hat viele Ebenen, Rollen, Artefakte etc., aber auch diese passen den wenigsten Unternehmen. Scrum hätte ein kleines Team binnen Tagen oder Stunden anpassen (natürlich verschlimmbessern) können, bei SAFe braucht es schon länger. Nehmen wir uns wieder den Plan von Weg 1 vor und verschieben

den Kick-off um mindestens sechs Monate. In diesen sechs Monaten erarbeiten wir ein neues Framework, das genau auf uns zugeschnitten ist, was wir dann stolz beim Kick-off vorstellen.

Der dritte Weg

In den dritten Weg kann Weg 1 und 2 einfließen – der ultimative Weg zum Glück. Ein Change-Team oder Transformationsteam wird ins Leben gerufen und damit beauftragt, die Transformation zu planen, durchzuführen und zu kontrollieren. Dazu werden Weg 1 und Weg 2 angegangen, wenn möglich aber im Geheimen, um niemanden im Unternehmen mit halbgaren Ansätzen zu verunsichern. Ganz nach dem Ansatz von Taylor, das Denken (in dem Fall das Planen) und das Ausführen zu trennen. Es müssen für jeden Transformationsschritt KPIs festgelegt werden, etwa wie viele Scrum Master nach sechs Monaten existieren müssen, wie viele Teams, die nach Scrum arbeiten und so weiter. Die Teams müssen regelmäßig Folien zum Fortschritt der Implementationsmaßnahmen erstellen.

Fazit

Warum schafft es ein Unternehmen nicht, mit diesen drei Wegen agil zu werden? Weg 1 ist der Versuch einer deterministischen Planung, die im komplizierten Umfeld funktionieren mag, da hier Ursache und Wirkungsbeziehungen durch schlichtes Nachdenken (in diesem Fall durch Planen) erhoben werden können. Eine Transformation findet, da hier Menschen beteiligt sind und Menschen nicht planbar sind, im komplexen Umfeld statt, daher ist jeder Plan zum Scheitern verurteilt. Du kannst nicht mit einem Wasserfallplan agil werden.

Weg 2 impliziert etwas Ähnliches, Planbarkeit, und verwechselt Wissen mit Erfahrung. Um ein Framework anzupassen, benötige ich Erfahrung und nicht nur Wissen. Die meisten Menschen lernen gleich, nehmen wir als Beispiel das Kochen. Wenn ich

kochen lerne, und das erste Mal ein Rezept ausprobiere, koche ich es strikt nach Rezept. Über die Zeit nehme ich Anpassungen vor, ich habe Erfahrungen mit dem Rezept gesammelt und würze es vielleicht anders, je nach dem, was ich bevorzuge. Nach vielen weiteren Erfahrungen bin ich dann fähig, eigene Rezepte zu entwickeln. Kaum jemand würde auf die Idee kommen, bei ersten Kochversuch das Kochbuch zu lesen und Änderungen am Rezept zu machen. Was beim Kochen nicht funktioniert, funktioniert auch beim Entwickeln von Methoden für eine Transformation nicht. Folge und verstehe Shu Ha Ri!

Und der dritte Weg glaubt weiterhin daran, dass eine Trennung von Denken und Ausführen sinnvoll ist und verwechselt Output mit Outcome. Auch die ReG AG hat erkannt, dass ein Transformationsplan oder ein eigenes Framework nicht dabei helfen werden, das Unternehmen agil auszustellen. Wie entscheiden Sie sich?



René Schröder ist seit 2001 erfolgreich als Agile Coach und Trainer in unterschiedlichen Branchen unterwegs. Seine Erfahrungen teilt er gerne in diversen Kolumnen. Seit 2020 veröffentlicht er außerdem die Panda Story (http://regsus.de/panda_story/), eine Geschichte, in der Ray, Howard und Britta eine agile Transformation bei der ReG AG erleben.