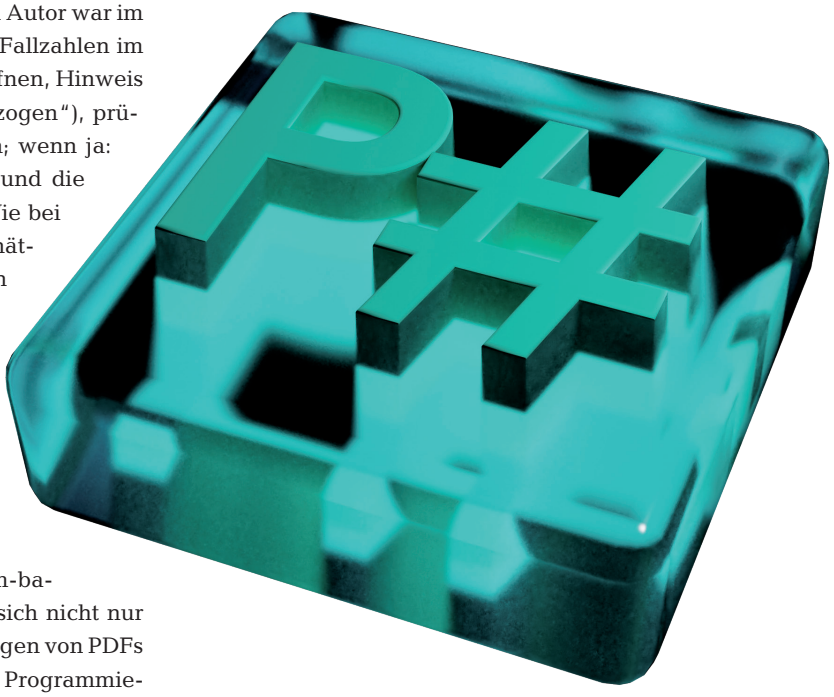
WEBCRAWLING MIT PUPPETEER SHARP

Lesen und verarbeiten

Daten aus Webseiten herausziehen ist mit Puppeteer Sharp eine einfache Übung.



Der Mensch ist ein Gewohnheitstier. Für den Autor war im vergangenen Jahr das Prüfen der Corona-Fallzahlen im Landkreis zur Routine geworden: Bookmark öffnen, Hinweis wegklicken („die Führerscheinstelle ist umgezogen"), prüfen, ob die Zahlen bereits aktualisiert wurden; wenn ja: mehrmals scrollen, um die aktuelle Inzidenz und die Fallzahlen für den eigenen Ort zu ermitteln. Wie bei so vielen Meetings auch galt hier: Eine E-Mail hätte genügt. Alles, was es dazu braucht, ist ein Programm, das die Informationen aus der Website liest, verarbeitet und – bei neuen Daten – eine E-Mail erzeugt und verschickt.

Um das automatische Navigieren durch Webseiten und das Auslesen von Informationen (Crawling) geht es in diesem dritten Teil der Serie über Puppeteer Sharp. Puppeteer Sharp ist eine Portierung von Googles Node.js-Bibliothek Puppeteer, mit der sich Chromium-basierte Browser fernsteuern lassen. Das eignet sich nicht nur für das Testen von Webseiten [1] oder das Erzeugen von PDFs aus HTML-Vorlagen [2], sondern auch für die Programmierung einfacher Webcrawler, die automatisch Informationen aus Webseiten herausziehen.

Crawling unterscheidet sich im Grunde nur wenig von End-to-End-Tests, geht es doch auch hier darum, Webseiten aufzurufen, mit ihnen zu interagieren und ihre Inhalte abzurufen. Die Grundlagen, insbesondere aus dem ersten Teil der Serie [1], werden daher nicht wiederholt. Die hier gezeigten Codebeispiele sollten jedoch auch ohne das Lesen der ersten beiden Teile verständlich sein. Wie in den beiden vorherigen Artikeln gilt: Es geht technisch auch mit weniger Overhead, aber für die Umsetzung ist es – dank der Möglichkeit, dem Browser bei der Arbeit zuzuschauen – deutlich einfacher.

Rechtliche Grenzen

Webseiten glänzen oft nicht mit der Offenheit, die sich Nutzer wünschen. Hier können Crawler helfen, Inhalte zu überwachen und sich über Änderungen informieren zu lassen, wenn die Webseiten hierfür selbst keine Funktionen dafür zur Verfügung stellen, zum Beispiel RSS-Feeds. Auch Wettbewerbsanalysen lassen sich eventuell per Crawler bewerkstelligen: Einfach alle Preise per Crawler im Minutentakt von der Webseite der Konkurrenz herunterladen und auswerten. Technisch ist das kein Problem – aber rechtlich?

Der Autor ist kein Jurist, weshalb das Folgende auch keine Rechtsberatung ist, sondern nur ein erster Hinweis. Im Zweifel gilt: Fragen Sie Ihre Rechtsabteilung oder einen Anwalt. Nur weil jemand eine Information ins Internet stellt, heißt das nicht, dass er oder sie damit alle Rechte an diesen Inhalten abgeben. Bevor Sie eine Website crawlen, sollten Sie prüfen, ob das überhaupt erlaubt ist. Anhaltspunkte dafür können die allgemeinen Geschäftsbedingungen oder Nutzungsbedingungen für die Site liefern.

Gibt es ein Captcha vor einer Website wie in Bild 1 [3], ist dies ein eindeutiges Stoppschild. Sein Zweck ist es, Crawler an der Nutzung der Website zu hindern. Ist die Website technisch gut gepflegt, findet sich im Root-Verzeichnis der Site eine Textdatei namens *robots.txt*; Listing 1 zeigt als Beispiel diese Datei von Spiegel.de. Sie enthält Regeln, an die sich Crawler halten sollen (laut Wikipedia ist der „Robots Exclusion Standard" ein Quasi-Standard [4]).

Das Spiegel-Beispiel ist vergleichsweise freizügig, hier ist allen User-Agents alles erlaubt, nur einige Links im Verzeichnis *gutscheine* sowie eine bestimmte PDF-Datei sollen nicht gecrawlt werden. Die Sitemap-Befehle sind Hinweise auf Seiten, die Crawlern explizit Seiten vorschlagen, die sie auswerten sollen. Eine weitere Einführung zum Aufbau und Inhalt der *robots.txt* liefert Selfhtml.org [5]. Es kann auch helfen, Webseitenbetreiber direkt zu fragen, ob man bestimmte Dinge per Crawler abfragen und weiterverwenden darf.

Gutes Benehmen

Durch Parallelisierung und bei entsprechend leistungsfähiger eigener Internetanbindung kann ein Crawler in wenigen

Sekunden selbst riesige Webseiten auslesen. Aus „Opfersicht“ kann sich so etwas schlimmstenfalls wie eine DDoS-Attacke anfühlen und die Verfügbarkeit der Webseite für andere Nutzer einschränken. In Zeiten von Functions as a Service und entsprechenden Abrechnungsmodellen kann ein Crawler aber auch real messbare Kosten erzeugen.

Crawlen Sie also auf verantwortungsbewusste Art und Weise. Das bedeutet: Machen Sie kurze Pausen zwischen einzelnen Seitenaufrufen und rufen Sie nicht zu viele Seiten parallel auf. Das führt dann zwar zu längeren Durchlaufzeiten für den Crawler, verhindert aber Lastspitzen bei der Website. Zusätzlich reduzieren Sie dadurch das Risiko, dass Ihre IP-Adresse (temporär) durch den Webseitenbetreiber geblockt wird.

Grundsätzlich sollte sich ein Crawler auch als solcher erkennbar machen. Dazu sollte der User-Agent entsprechend gesetzt sein. Zum Glück bietet Puppeteer Sharp hierfür einfache Lösungen an.

Listing 1: Die robots.txt von Spiegel.de

```
User-agent: *
Allow: /
Disallow: /*CR-Dokumentation.pdf$
Disallow: /gutscheine/suche?
Disallow: /gutscheine/*?code=*
Disallow: /gutscheine/*&code=*

Sitemap: https://www.spiegel.de/sitemaps/news-de.xml
Sitemap: https://www.spiegel.de/sitemaps/videos/sitemap.xml
Sitemap: https://www.spiegel.de/plus/sitemap.xml
Sitemap: https://www.spiegel.de/sitemap.xml
Sitemap: https://www.spiegel.de/gutscheine/sitemap.xml
```

Ein erstes Beispiel

Das Szenario aus der Einleitung dient als Aufhänger für den ersten Crawler. In Bild 2 ist eine vereinfachte Variante der Webseite dargestellt. Die gelben Markierungen sind nicht auf der Webseite vorhanden, sondern wurden nachträglich eingefügt, um die gewünschten Informationen zu markieren, die es auszulesen gilt: a) die Anzahl der Neuinfektionen, b) das Datum der letzten Webseiten-Aktualisierung und c) der aktuelle Inzidenzwert.

In diesem Fall ist die Herausforderung nicht die Navigation durch eine komplexe Webseite, sondern die verlässliche Extraktion der gewünschten Informationen aus einem Fließtext. Die erste (fachliche) Aufgabe besteht darin, die entsprechenden Textmuster zu erkennen, die auf die gewünschten Daten hinweisen.

Das Datum (b) lässt sich relativ einfach identifizieren, wenn man einmal davon ausgeht, dass davor immer „Datenstand: “ steht. Gleiches gilt für die Inzidenz (c), die nach dem Text „7-Tages-Inzidenz beträgt “ zu finden ist. Bei den Neuinfektionen (a) ist es allerdings nicht ganz so einfach: Wie verhält es sich, wenn es nur einen oder sogar keine weiteren Fälle mehr gibt?

Reguläre Ausdrücke als Rettung?

In allen drei Fällen lassen sich reguläre Ausdrücke definieren, welche die gewünschten Daten aus dem Fließtext extrahieren. Im vorliegenden Fall sind die drei Absätze der Webseite (Bild 2) jeweils in einem <*p*>-Element untergebracht. Für das Beispiel genügt es, die gesuchten Stellen im gesamten Text zu suchen.

Das hängt jedoch sehr stark von der Webseite ab, die durchsucht wird. Die drei regulären Ausdrücke für dieses Beispiel sind: ▶

Captchas sind für Crawler ein klares Haltesignal (Bild 1)

Fallzahlen im Landkreis Musterkreis (Coronavirus)

Die Zahlen werden montags bis freitags am späten Nachmittag aktualisiert.

Fallzahlen Infizierte insgesamt:

Im Landkreis Musterkreis wurden seit Montag, 14 Uhr, **7 weitere Fälle** einer Infektion mit dem neuartigen Coronavirus gemeldet. Damit gibt es seit Meldung des ersten Falls im Landkreis Musterkreis am 4. Februar 2020 insgesamt **15.188 bestätigte Infektionsfälle.** Aktuell sind **101 Personen infiziert**. Insgesamt wurden 2.610 Mutationsfälle bestätigt.

Insgesamt gelten **14.797 Personen** als statistisch **genesen**. Darin enthalten sind alle Personen, bei denen der Beginn der Quarantäne 14 Tage oder länger zurückliegt. Die tatsächliche Krankheitsdauer variiert, daher kann es in Einzelfällen sein, dass Personen auch nach Ablauf der Quarantäne noch Symptome haben, andere Infizierte sind bereits nach wenigen Tagen wieder gesund. (Datenstand: 15.06.2021).

Die 7-Tages-Inzidenz ergibt sich aus der Anzahl der für die letzten sieben Tage neu gemeldeten Fälle pro 100.000 Einwohner. Die vom Robert Koch-Institut (RKI) für den Landkreis Musterkreis ermittelte **7-Tages-Inzidenz** beträgt **12,3.** (15.06.2021)

Inzidenzwerte der vergangenen 7 Tage:

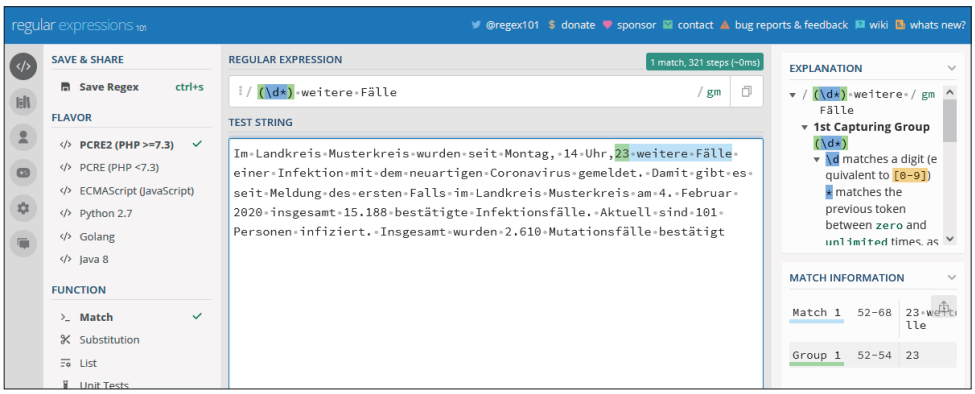
14.06.: 15,7
13.06.: 15,1
12.06.: 15,4
11.06.: 14,6
10.06.: 13,1
09.06.: 16,0
08.06.: 16,5

Die per Crawler auszulesende Website: Die gelben Hervorhebungen gehören nicht dazu, sondern dienen nur der Verdeutlichung (Bild 2)

```
var neueFaelleRegex =
  new Regex(
  @"(\d*) weitere Fälle");

var datumRegex =
  new Regex(@"Datenstand: "
  + @"(\d{1,2}\.\d{2}\.202"
  + @"[1,2])");

var inzidenzRegex =
  new Regex(
  @"7-Tages-Inzidenz "
  @"beträgt (\d{1,4},"
  @"\d{0,2})\.");
```



Regex101.com hilft bei der Entwicklung von regulären Ausdrücken (Bild 3)

Wer wie der Autor reguläre Ausdrücke gleichzeitig bewundert und verflucht (und deren Funktion nach dem Schreiben sofort wieder vergisst), wird anfänglich wohl hier die meiste Zeit bei der Crawler-Entwicklung benötigen. Dienste wie Regexr.com oder Regex101.com erleichtern einem dabei das Leben. Auf beiden Webseiten können Sie den Zieltext hineinkopieren und in der darüberliegenden Textbox Ihren regulären Ausdruck entwickeln. Sie erhalten eine Vorschau,

Listing 2: Der Corona-Crawler

```
using PuppeteerSharp;
using System;
using System.IO;
using System.Text;
using System.Text.RegularExpressions;

using var browser =
    await Puppeteer.LaunchAsync(new LaunchOptions
{
  Headless = true, //Browser nicht anzeigen
  SlowMo = 40,
  ExecutablePath = @"C:\Program Files (x86)\"
    + @"Microsoft\Edge\Application\msedge.exe"
});
var page = await browser.NewPageAsync();
var userAgent = await browser.GetUserAgentAsync();
await page.SetUserAgentAsync($"{userAgent} TestBot");

await page.GoToAsync(
  "https://www.janschreier.de/Fallzahlen1.html");
string[] paragraphs = await page.
  EvaluateExpressionAsync<string[]>(
  @"Array.from(document.querySelectorAll('p'))."
  # @"map(p => p.innerText)");

// Ab hier erfolgt die Evaluation unabhängig
// von PuppeteerSharp
var datumRegex = new Regex(
  @"Datenstand: (\d{1,2}\.\d{2}\.202[1,2])");
var datumText = FindFirstRegexInStringArray(
  paragraphs, datumRegex);

if (datumText == null)
{
  Console.WriteLine("Abbruch! Datum nicht gefunden");
}
else
{
  var neueFaelleRegex = new Regex(
    @"(\d*) weitere Fälle");
  var neueFaelle = FindFirstRegexInStringArray(
    paragraphs, neueFaelleRegex);
  Console.WriteLine($"{datumText} : {neueFaelle ??"
    # " 'neue Fälle nicht gefunden'}");

  var inzidenzRegex = new Regex(@"7-Tages-Inzidenz "
    + @"beträgt (\d{1,4},\d{0,2})\.");
  var inzidenz = FindFirstRegexInStringArray(
    paragraphs, inzidenzRegex);
  Console.WriteLine($"{datumText} : {inzidenz ?? "
    + $"'Inzidenz nicht gefunden'}");
}

static string? FindFirstRegexInStringArray(
    string[] paragraphs, Regex regex)
{
  foreach (string paragraph in paragraphs)
  {
    var regexMatch = regex.Match(paragraph);
    if (regexMatch.Success)
    { return regexMatch.Groups[1].ToString(); }
  }
  return null;
}
```

welche Inhalte der reguläre Ausdruck erfasst, sowie umfangreiche Erklärungen zu den einzelnen Parametern, siehe Bild 3.

Nach diesen Vorarbeiten kann es losgehen mit dem Schreiben des Crawlers, was in Listing 2 erfolgt. Zuerst wird ein Browser erzeugt, der im Headless-Modus operiert. Der Parameter *SlowMo = 40* sorgt dafür, dass zwischen einzelnen Interaktionen mit der Webseite 40 Millisekunden gewartet wird (weil beispielsweise jeder Tastendruck eine Aktion ist, sollte der Wert nicht zu groß sein). Anschließend öffnet der Code einen neuen Browser-Tab und erweitert die User-Agent-Kennung um den String „ *TestBot*“. Im Beispiel wird die Webseite nun geladen. Danach liest *EvaluateExpressionAsync()* die *innerText*-Werte als String-Array aus.

Die Startseite von Wikipedia.org zeigt ein Suchfenster mit Dropdown-Liste für die Sprachwahl (Bild 4)

Murphys Gesetz als Maxime

Die weitere Verarbeitung erfolgt ausschließlich in C# ohne die Hilfe von Puppeteer Sharp. Bei sämtlichen Werten, die ein Crawler erfasst, sollten Sie davon ausgehen, dass irgendwann etwas fehlschlagen wird, und entsprechend Vorsorge treffen. Im Beispiel werden die regulären Ausdrücke evaluiert und für jedes Ergebnis wird auch der Fall behandelt, dass der Ausdruck keine Daten gefunden hat (etwa, weil sich etwas auf der Webseite geändert hat, was Sie beim Entwickeln des Crawlers nicht vorhergesehen haben).

Im Fall eines fehlschlagenden regulären Ausdrucks ist es hilfreich, den Inhalt der Webseite vollständig zu speichern und in der Fehlermeldung auszugeben; das geht mittels *page.GetContentAsync()*. Dadurch lässt sich der Fehler später leichter nachvollziehen. Fehler im Übertragungsweg sind ebenfalls ein Problem, mit dem zu rechnen ist. Dazu ist der Einsatz eines zeitverzögerten Wiederholungsmechanismus sinnvoll. Die Zahl der Wiederholungen sollte jedoch gerade anfangs nicht zu hoch sein, um keine unnötige Last durch fehlerhafte Crawler zu erzeugen.

Sofern Sie die Daten aus den regulären Ausdrücken weiterverarbeiten, behandeln Sie die Daten wie Nutzereingaben und wenden entsprechende Sicherheitsmechanismen an, um keine Sicherheitslücken zu erzeugen. Auch das Speichern von Daten in CSV-Dateien kann zu Remote-Code-Execution-Angriffen führen, wie vor Kurzem die Luca-App öffentlichkeitswirksam demonstriert hat [6].

Komplexere Navigation

Im zweiten Beispiel wird Wikipedia aufgerufen, um dort die Bevölkerungszahl von Mallorca aus der spanischen Sektion des Online-Lexikons auszulesen. Wikipedia hat hierfür ein API, aber für ein frei zugängliches Beispiel zum Crawling eignet sich die Webseite, da die *robots.txt*-Datei entsprechendes Crawling erlaubt. Dazu soll Wikipedia.org (Bild 4) aufgerufen werden, die Sprache auf Spanisch eingestellt, „Mallorca“ in das Textfeld eingegeben und anschließend mittels Drücken der Eingabetaste die Suche ausgelöst werden. Auf der geladenen Mallorca-Seite soll dann im Infokasten auf der rechten Seite der Wert unter *Población* ausgelesen werden (siehe Bild 5 ganz unten). Der Infokasten ist eine HTML-Tabelle, der entsprechende Code zum Auslesen und zur Ausgabe der Werte ist in Listing 3 zu sehen.

Coordenadas	39°37'00"N 2°59'00"E
Capital	Palma de Mallorca
Idioma oficial	Español y catalán *(dialecto mallorquín)*
Entidad	Isla
• País	España
• Comunidad aut.	Islas Baleares
• Provincia	Islas Baleares
Presidenta	Catalina Cladera Crespí (PSOE)
Senado	3 senadores
Parlamento aut.	33 diputados autonómicos
Consejo Insular	33 diputados
Superficie	Puesto 1.º
• Total	3640.11 km²
Altitud	
• Máxima	1436 m s. n. m.
• Mínima	0 m s. n. m.
Población (2019)	Puesto 2.º
• Total	923 608 hab.
• Densidad	255,14 hab/km²

Ausschnitt des Infokastens der Wikipedia-Seite zu Mallorca (Bild 5)

Den Überblick behalten

Listing 3 arbeitet bewusst mit Erweiterungsmethoden, um die Lesbarkeit des Codes zu erhöhen. Die Entwicklung von Crawlern erfolgt meist sehr prozedural, weil man sich Schritt für Schritt vorarbeitet. Der so entstehende Code sollte jedoch logisch gegliedert werden, um sich später schneller zurechtzufinden. Gerade bei Crawlern ist zu erwarten, dass Anpassungen notwendig sind, weil sich der Inhalte der Webseite ändert.

Die Klasse *WikipediaPageExtensions* liefert hierfür die Methoden *SetzeSpracheImSuchfeld()* und *GebeSuchBegriffEinUndStarteSuche()*. Erstere wählt die Sprache anhand des HTML-Option-Werts („*es*“ für Español, also Spanisch). Die zweite gibt den Suchwert in das Textfeld mit der ID *searchInput* ein und simuliert anschließend den Druck auf die Eingabetaste. ▶

Da der Zustand des *Page*-Objekts manipuliert wird, hält der Autor Erweiterungsmethoden hier für eine logischere Lösung als Funktionsaufrufe, die das *Page*-Objekt als Parameter erhalten.

Bitte warten

Vor der Eingabe des Suchbegriffs erzeugt *page.WaitForNavigationAsync()* einen neuen Task, der *Completed* ergibt, wenn die Navigation erfolgt ist. Die Navigation wird in diesem Fall mit der Eingabetaste ausgelöst (letzter Befehle in der Erweiterungsmethode *GebeSuchBegriffEinUndStarteSuche()*). Ohne dieses Warten würde das nächste Auslesen der Inhalte der Webseite fehlschlagen, da sich ihr Inhalt in einem ungültigen Zustand befindet. Werden Inhalte dynamisch nachgeladen, kann die Suche nach Werten in der Webseite fehlschlagen, wenn diese noch nicht geladen wurden. *WaitForSelector_Async()* umgeht dieses Problem. Das ist im Beispiel zwar nicht nötig, aber dennoch dargestellt, indem darauf gewartet wird, dass ein HTML-Element mit dem *class*-Attribut *infobox* vorhanden ist (die HTML-Tabelle, die ausgelesen soll, verfügt über diese Auszeichnung). Der restliche Code extrahiert zuerst den Tabelleninhalt zeilenweise in ein String-Array, in dem ein regulärer Ausdruck den gewünschten Wert sucht.

Fazit

Neben den gezeigten Beispiele unterstützt Puppeteer Sharp unter anderem noch weitere für das Crawling nützliche Funktionen. Dazu gehört das Auslesen und Setzen von (gespeicher-

Listing 3: Crawling der spanischen Wikipedia-Seite

```
using PuppeteerSharp;
using System;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

using var browser = await Puppeteer.LaunchAsync(
    new LaunchOptions
{
  Headless = true, SlowMo = 20,
  ExecutablePath = @"C:\Program Files (x86)\"
    + @"Microsoft\Edge\Application\msedge.exe"
});
var page = await browser.NewPageAsync();
var userAgent = await browser.GetUserAgentAsync();
await page.SetUserAgentAsync($"{userAgent} TestBot");

await page.GoToAsync("https://www.wikipedia.org/");

await page.SetzeSpracheImSuchfeld("es");

var navigationTask = page.WaitForNavigationAsync();
await page.GebeSuchBegriffEinUndStarteSuche(
  "mallorca");
await navigationTask;

await page.WaitForSelectorAsync(".infobox");
string[] tableRowTexts =
  await page.EvaluateExpressionAsync<string[]>(
  @"Array.from(document.querySelector('.infobox').
  querySelectorAll('tr')).map(tr => tr.innerText)");

var populationRegex = new Regex(
  @"Total\s([\d\s]*)hab\.");
var populationText = FindFirstRegexInStringArray(
  tableRowTexts, populationRegex);

if (populationText != null)
{
  Console.WriteLine($"Laut der spanischen "
    + $" Wikipedia hat Mallorca "
    + $"{populationText.Trim()} Einwohner.");
}

static string? FindFirstRegexInStringArray(
    string[] paragraphs, Regex regex)
{
  foreach (string paragraph in paragraphs)
  {
    var regexMatch = regex.Match(paragraph);
    if (regexMatch.Success)
    {
      return regexMatch.Groups[1].ToString();
    }
  }
  return null;
}

public static class WikipediaPageExtensions
{
  public static async Task SetzeSpracheImSuchfeld(
    + this Page page, string language) => await
    page.SelectAsync("#searchLanguage", language);

  public static async Task
      GebeSuchBegriffEinUndStarteSuche(
      this Page page, string searchstring)
  {
    var elementHandle = await page.
      QuerySelectorAsync("#searchInput");
    await elementHandle.TypeAsync(searchstring);
    await elementHandle.PressAsync(
      PuppeteerSharp.Input.Key.Enter);
  }
}
```

ten) Cookies, beispielsweise um Sessions wieder aufzunehmen. Oder die Methode *NewPageAsync()*, mit der sich weitere Tabs erzeugen und zum parallelen Crawlen verwenden lassen. Den besten Überblick über diese und weitere Funktionen liefert das Buch „UI Testing with Puppeteer“ [7]; es beschäftigt sich zwar mit Puppeteer und nicht mit Puppeteer Sharp, aber in Verbindung mit der API-Dokumentation von Puppeteer Sharp lassen sich die Beispiele sehr gut übertragen.

Damit endet die Serie durch die Haupteinsatzgebiete von Puppeteer Sharp (und einigen hilfreichen Ergänzungen). Die Bibliothek bietet einfache Möglichkeiten, um mit UI-Tests zu beginnen. Auch PDF-Dokumente lassen sich mit wenig Aufwand erstellen und können dabei helfen, einen Wildwuchs an dezentralen Word-Vorlagen und Ordnern mit Word-Dateien als führender „Datenbank“ Einhalt zu gebieten.

Wer Inspiration für mögliche Crawling-Projekte sucht, findet diese sicherlich bei den Projekten der Open Knowledge Foundation [8]. Und auch David Kriesels Vortrag über Crawling und die dazugehörigen Auswertungen unter [9] sei jedem Datenanalytiker wärmstens empfohlen. ■

[1] Jan Hendrik Schreier, Mit Googles Werkzeugkiste, Web-UI- und End-to-End-Tests, dotnetpro 7/2021, Seite 71 ff., www.dotnetpro.de/A2107PuppeteerSharp

[2] Jan Hendrik Schreier, Chrome als PDF-Drucker,PDFs aus HTML-Templates erzeugen, dotnetpro 8/2021, Seite 82 ff., www.dotnetpro.de/A2108PuppeteerSharp

[3] Captcha bei Wikipedia, www.dotnetpro.de/SL2109PuppeteerSharp1

[4] Robots Exclusion Standard bei Wikipedia, www.dotnetpro.de/SL2109PuppeteerSharp2

[5] Selfhtml, Grundlagen/Robots.txt, www.dotnetpro.de/SL2109PuppeteerSharp3

[6] Patrick Beuth, Luca-App bereitet Hackern den Weg in die Gesundheitsämter, www.dotnetpro.de/SL2109PuppeteerSharp4

[7] Dario Kondratiuk, UI Testing with Puppeteer, Packt Publishing 2021, ISBN 978-1-80020-678-6

[8] Open Knowledge Foundation, Unsere Projekte, https://okfn.de/projekte/

[9] Chaos Computer Club, SpiegelMining – Reverse Engineering von Spiegel-Online (33c3), www.dotnetpro.de/SL2109PuppeteerSharp5

Jan Hendrik Schreier
ist Wirtschaftsinformatiker und lebt im Münchner Osten. Beruflich entwickelt er Line-of-Business-Anwendungen (WPF, Full-Stack). Sein Fokus liegt auf Datenstrukturen und fachlichem Prozessdesign.
mail@janschreier.de

dnpCode A2109PuppeteerSharp