

Testowanie aplikacji dla programistów frontendowych

Wiodące frameworki do automatyzacji testów aplikacji internetowych i ich przyszłość oparta na testowaniu niskokodowym i sztucznej inteligencji

Eran Kinsbruner



Tytuł oryginału: A Frontend Web Developer's Guide to Testing: Explore leading web test automation frameworks and their future driven by low-code and AI

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-9864-1

Copyright © Packt Publishing 2022. First published in the English language under the title 'A Frontend Web Developer's Guide to Testing – (9781803238319)'.

Polish edition copyright © 2023 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/tesapl>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<https://ftp.helion.pl/przyklady/tesapl.zip>

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	11
O autorze	13
O korektorze	14
Wstęp	15

Część I. Frontendowe testowanie stron internetowych

Rozdział 1. Metody testowania z wykorzystaniem różnych przeglądarek	23
Przegląd krajobrazu internetowego	24
Typy aplikacji internetowych	25
Tradycyjne aplikacje internetowe	25
Responsywne aplikacje internetowe	26
Progresywne aplikacje internetowe	28
Typy testów dla aplikacji internetowych	30
Testy funkcjonalne aplikacji internetowych	31
Testy нефункционалне aplikacji internetowych	31
Zastosowanie przeglądarek interfejsowych i bezinterfejsowych podczas tworzenia i testowania aplikacji	33
Wybór między przeglądarkami interfejsowymi i bezinterfejsowymi	34
Frameworki testowania przy użyciu przeglądarek bezinterfejsowych	36
Podsumowanie	39

Rozdział 2. Wyzwania stojące przed frontendowymi programistami aplikacji internetowych	40
Wyzwania związane z tworzeniem aplikacji internetowych	41
Jakość a cykl wydawniczy	43
Wyzwania związane z pokryciem dostępnych platform i systemów operacyjnych	43
Wyzwania niefunkcjonalne w tworzeniu aplikacji internetowych	48
Wyzwania związane z wydajnością	48
Wyzwania związane z ułatwieniami dostępu	49
Wyzwania związane z zapewnianiem zgodności aplikacji internetowych	51
Podsumowanie	53
Rozdział 3. Najlepsze frameworki do automatyzacji testów aplikacji internetowych	54
Przegląd rynku testowania aplikacji internetowych	55
Pierwsze kroki z frameworkiem Selenium WebDriver	56
Konfigurowanie komponentu WebDriver	57
Selenium Grid	60
Pierwsze kroki z frameworkiem Cypress	63
Ważne funkcjonalności Cypressa	65
Pierwsze kroki z frameworkiem Google Puppeteer	68
Pierwsze kroki z frameworkiem Microsoft Playwright	69
Podsumowanie	72
Rozdział 4. Dopasowanie odpowiednich osób i przypadków użycia do frameworków testowych	73
Wymagania techniczne	74
Przegląd osób testujących strony internetowe	74
Przypadki użycia i zagadnienia dotyczące wyboru solidnego frameworku do automatyzacji testów	76
Kwestie społecznościowe	76
Możliwości testowania na dużą skalę	77
Integracje stosu narzędzi i wtyczki	77
Łatwość użytkowania i popularność	77
Wielokrotny użytek i łatwość utrzymywania	78
Raportowanie, analiza testów i sztuczna inteligencja	78
Macierz oceny frameworków testowych	79
Podsumowanie	83
Rozdział 5. Wprowadzenie do wiodących frontendowych frameworków do tworzenia aplikacji internetowych	85
Wymagania techniczne	86
Wprowadzenie do wiodących frameworków do tworzenia aplikacji internetowych	86
Wytyczne dotyczące wyboru frameworku do tworzenia aplikacji internetowych	87
ReactJS	88
AngularJS	91
Vue.js	93

Ember.js	95
Svelte	97
Podsumowanie	99

Część II. Strategia ciągłego testowania dla programistów aplikacji internetowych

Rozdział 6. Filary strategii programistycznych testów aplikacji internetowych 103

Filary planu i strategii testowania aplikacji internetowej	104
Poznaj użytkowników docelowych	105
Opracuj plan testów	105
Przygotuj stos narzędzi i środowiska	106
Ustal kryteria i cele jakości	106
Określ porządek chronologiczny i harmonogram	106
Wykonuj, monitoruj, mierz i dokumentuj	106
Pomiar sukcesu strategii ciągłego testowania	107
Studium przypadku — strategia testowania rzeczywistej aplikacji internetowej	110
Podsumowanie	111

Rozdział 7. Podstawowe funkcjonalności wiodących frameworków do automatyzacji testów javascriptowych 112

Porównanie funkcjonalności frameworków do automatyzacji testów	113
Testy wizualne	113
Testowanie interfejsów API	116
Obsługiwane języki programowania	118
Testowanie urządzeń mobilnych	119
Testy wydajnościowe	121
Testowanie ułatwień dostępu	124
Testowanie żądań sieciowych i atrapy usług	125
Praca z elementami	127
Istotne zdarzenia wymagające ponownej ewaluacji frameworków do automatyzacji testów	133
Podsumowanie	134

Rozdział 8. Mierzenie pokrycia testowego aplikacji internetowej 135

Wprowadzenie do pokrycia kodu i pokrycia testowego	136
Pokrycie testowe	136
Pokrycie kodu	137
Narzędzia mierzenia pokrycia kodu JavaScriptu dla programistów aplikacji internetowych	138
Pomiar pokrycia kodu JavaScriptu za pomocą narzędzia Istanbul i frameworku Cypress	139
Uzupełnianie pokrycia kodu pokryciem testowym	142
Podsumowanie	143

Część III. Przewodnik po frontendowych frameworkach do automatyzacji javascriptowych testów aplikacji internetowych

Rozdział 9. Praca z frameworkiem Selenium	147
Wymagania techniczne	147
Framework Selenium i jego komponenty	148
Selenium WebDriver	148
Zaawansowane funkcjonalności Selenium	151
Różne metody testowania z wykorzystaniem Selenium	155
Przyszłość frameworku Selenium	159
Podsumowanie	160
Rozdział 10. Praca z frameworkiem Cypress	161
Wymagania techniczne	162
Pierwsze kroki z Cypressem	162
GUI frameworku Cypress	162
IDE i wiersz poleceń Cypressa	164
Zaawansowane funkcjonalności automatyzacji testów Cypressa	167
Ponawianie testów Cypressa	167
Korzystanie z namiastek, szpiegów i zegarów w Cypresie	168
Uruchamianie Cypressa w ramach potoku CI	170
Testowanie komponentów	171
Cypress Studio	172
Wtyczki Cypressa	175
Testowanie API z wykorzystaniem Cypressa	176
Przyszłość frameworku Cypress	178
Podsumowanie	179
Rozdział 11. Praca z frameworkiem Playwright	180
Wymagania techniczne	181
Pierwsze kroki z Playwrightem	181
Zaawansowane funkcjonalności automatyzacji testów Playwrighta	184
Narzędzie Inspector Playwrighta	184
Emulowanie urządzeń mobilnych	186
Adnotacje testowe Playwrighta	186
Testowanie API przy użyciu Playwrighta	188
Asercje Playwrighta	189
Atrapy żądań sieciowych Playwrighta	190
Obiektowy Model Strony Playwrighta	191
Raporty testowe Playwrighta	191
Test runnery Playwrighta	192
Trace viewer Playwrighta	193
Zaawansowane konfiguracje Playwrighta	194
Integracja Playwrighta z CI	196
Przyszłość frameworku Playwright	197
Podsumowanie	199

Rozdział 12. Praca z frameworkiem Puppeteer	200
Wymagania techniczne	201
Pierwsze kroki z Puppeteerem	201
Zaawansowane funkcjonalności automatyzacji testów Puppeteera	205
Przestrzenie nazw Puppeteera	205
Praca z elementami przy użyciu Puppeteera	206
Testy obciążeniowe Puppeteera	209
Puppeteer i Cucumber BDD	209
Testy ułatwień dostępu Puppeteera	210
Śledzenie aplikacji internetowych przy użyciu Puppeteera	211
Testy API Puppeteera	212
Puppeteer i narzędzia dla programistów Google'a	212
Integracja Puppeteera z frameworkiem CodeceptJS	213
Integracja Puppeteera z CI	215
Przyszłość frameworku Puppeteer	217
Podsumowanie	219
Rozdział 13. Uzupełnianie testów opartych na kodzie automatyzacją niskokodową	220
Podstawowe funkcjonalności narzędzi do testowania niskokodowego i bezkodowego	221
Przegląd narzędzi bezkodowych w krajobrazie open source	224
Narzędzia bezkodowe na licencji open source	224
Wiodące komercyjne narzędzia do bezkodowego testowania aplikacji internetowych	230
Narzędzie Perfecto Scriptless Web	231
Narzędzie Testim do bezkodowego testowania aplikacji internetowych	234
Narzędzie Mabl do bezkodowego testowania aplikacji internetowych	237
Podsumowanie	242
Rozdział 14. Podsumowanie	244
Główne wnioski z książki	245
Przydatne materiały referencyjne	246
Dla frameworku Cypress	246
Dla frameworku Playwright	247
Dla frameworku Selenium	247
Dla frameworku Puppeteer	247

Metody testowania z wykorzystaniem różnych przeglądarek

W ciągu ostatnich kilku lat technologia internetowa znacznie się rozwinęła. Użytkownicy końcowi mają teraz do czynienia z zupełnie nowym poziomem dojrzałych aplikacji internetowych w postaci responsywnych i progresywnych aplikacji Reacta czy Fluttera. W związku z tym postępem technologicznym programiści mają coraz większe problemy z zapewnianiem stałej jakości swoich aplikacji internetowych, niezależnie od tego, na których platformach (mobilnych, stacjonarnych czy na obu) są one używane.

W tym rozdziale omówię najbardziej zaawansowane technologie internetowe i typy aplikacji internetowych, z którymi możesz się zetknąć, w tym typy responsywne i progresywne. Postaram się wskazać główne trendy, które zazwyczaj wpływają na twórców aplikacji internetowych, oraz różne typy testów istotnych dla takich aplikacji. Internetowy krajobraz oferuje programistom szeroki zakres typów aplikacji internetowych pisanych przy użyciu różnych frameworków. Aplikacje responsywne i progresywne oraz aplikacje frameworków Flutter i React Native stanowią jedynie niewielki podzbiór. Skoncentruję się na głównych typach aplikacji i wskażę różnice między nimi, abyś jako programista frontendowy mógł łatwiej dobrać odpowiednie działania związane z testowaniem.

W tym rozdziale zostaną omówione następujące tematy:

- przegląd krajobrazu internetowego,
- typy aplikacji internetowych,
- typy testów dla aplikacji internetowych,
- zastosowanie przeglądarek interfejsowych i bezinterfejsowych w tworzeniu i testowaniu aplikacji internetowych.

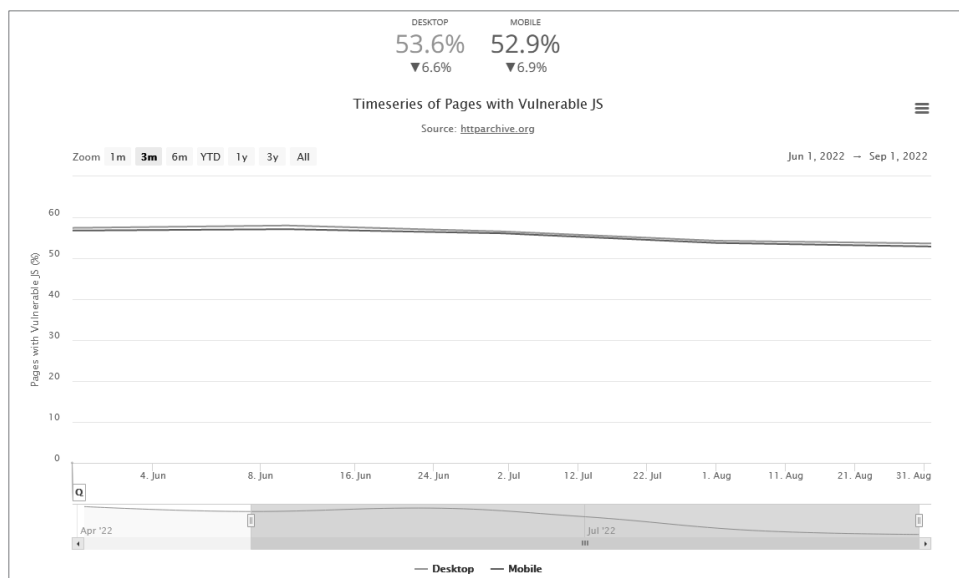
Przegląd krajobrazu internetowego

W porównaniu ze stanem sprzed kilku lat krajobraz internetowy jest w tej chwili na najbardziej zaawansowanym etapie. Obecnie aplikacje internetowe mogą wykorzystywać unikatowe możliwości różnych przeglądarek internetowych, jeszcze niedawno niedostępne. Dzięki funkcjonalnościom, takim jak m.in. interakcja z usługami lokalizacyjnymi i kamerami, oraz opcjom instalowania na smartfonach, tabletach lub laptopach aplikacje internetowe zaczynają się coraz bardziej upodabniać do aplikacji mobilnych.

Oprócz wspomnianych kwestii istnieją jednak jeszcze inne ważne aspekty krajobrazu internetowego. Należą do nich stale zmieniające się technologie internetowe, które są dostępne dla twórców aplikacji sieciowych.

Według serwisu Hackr.io (<https://hackr.io/blog/web-development-frameworks>) twórcy aplikacji internetowych mają szeroki wybór narzędzi. Mając do dyspozycji takie frameworki jak m.in. ExpressJS, Angular, React, Vue i Ember, programiści mogą wybrać najbardziej odpowiadającą im technologię.

Ponieważ technologie internetowe rozwijają się i są wykorzystywane w różnych strategiach sprzedaży wielokanałowej, dużym wyzwaniem stają się również coraz to nowsze wektory ataków i stale rosnąca liczba luk w zabezpieczeniach. Jak pokazują dane serwisu HTTP Archive dotyczące bieżącego monitorowania trendów internetowych (<https://httparchive.org/reports/state-of-the-web>), ok. 53% indeksowanych stron zawiera co najmniej jedną znaną lukę w zabezpieczeniach w zewnętrznym JavaScript (rysunek 1.1).



Rygunek 1.1. Strony internetowe z podatnym na ataki kodem JavaScriptu
(źródło: <https://httparchive.org/reports/state-of-the-web>)

Oprócz rozwoju technologii internetowych i poziomu dojrzałości funkcji przeglądarek dodatkowym obszarem, który zmienił się całkowicie zarówno pod względem świadomości, jak i znaczenia, jest kwestia ułatwienia dostępu do aplikacji internetowych. Organizacje tworzące aplikacje internetowe muszą obecnie przestrzegać rygorystycznych norm dostępności na komputerach stacjonarnych i urządzeniach mobilnych. Nieprzestrzeganie tych wytycznych, takich jak § 508 amerykańskiej ustawy o rehabilitacji, amerykańska ustawa o niepełnosprawnych (ang. *Americans with Disabilities Act* — ADA) i wytyczne dotyczące ułatwień dostępu do zawartości sieci (ang. *Web Content Accessibility Guidelines* — WCAG), może skutkować nałożeniem na organizację wysokiej kary finansowej i zniszczeniem reputacji marki.

Obecnie twórcy aplikacji internetowych powinni być wyposażeni w szerszą wiedzę i lepsze narzędzia, a ponadto odbywać ciągłe szkolenia z zakresu jakości tych aplikacji. Ma to na celu zagwarantowanie solidności aplikacji i uniknięcie wprowadzenia ryzyka biznesowego dla marki organizacji — związanego z jakością, bezpieczeństwem, ułatwieniem dostępu, dostępnością lub zgodnością z takimi współczynnikami jak różne rozmiary i rozdzielczości ekranu.

Po tym krótkim przeglądzie aktualnego krajobrazu internetowego przyjrzyjmy się różnym typom aplikacji internetowych i ich znaczeniu z perspektywy rozwoju i testowania oprogramowania.

Typy aplikacji internetowych

Tworząc aplikację internetową na ciągle zmieniającym się rynku cyfrowym, programiści mają do wyboru tradycyjną, responsywną lub progresywną aplikację internetową. Każdy wybór ma swoje zalety, wady i implikacje technologiczne, np. język programowania aplikacji, platformy docelowe, na których będzie działać, oraz odpowiedni dla danej aplikacji stos technologiczny.

Progresywna aplikacja internetowa (ang. *Progressive Web Application* — PWA), która ma działać zarówno jako aplikacja internetowa, jak i mobilna, może być rozwijana w języku JavaScript, ale testowanie jej na prawdziwych urządzeniach mobilnych i w przeglądarkach będzie wymagało mieszanki różnych frameworków, takich jak Selenium, Appium itd.

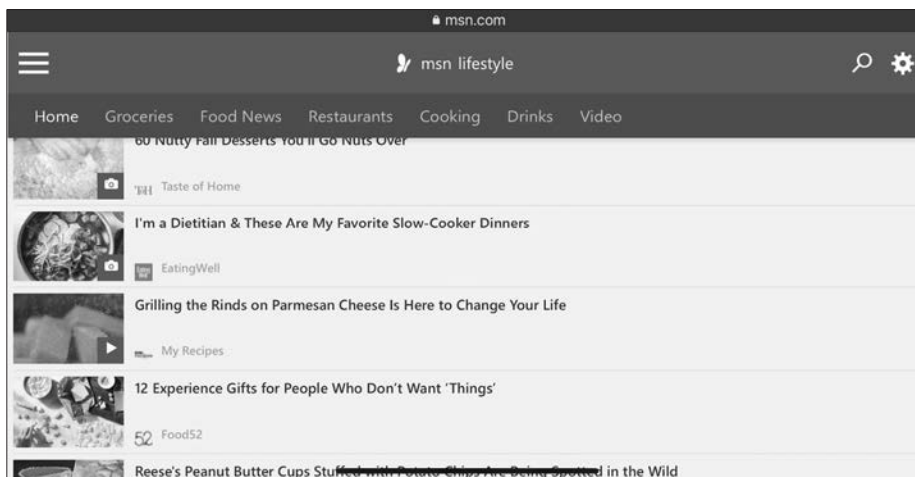
Przyjrzyjmy się bliżej poszczególnym typom aplikacji.

Tradycyjne aplikacje internetowe

Najbardziej podstawowy typ aplikacji internetowej to aplikacja tworzona i projektowana z przeznaczeniem do działania na komputerach stacjonarnych (np. na maszynach z systemem Windows 11 z przeglądarką Edge lub z systemem macOS z przeglądarką Safari). Chociaż takie aplikacje są w pełni obsługiwane na urządzeniach mobilnych (np. smartfonach i tabletach), nie są projektowane jako responsywne.

Jeżeli otworzysz witrynę MSN (<http://msn.com>) w przeglądarce Chrome na komputerze stacjonarnym z systemem Windows, będziesz mógł pracować w witrynie, poruszać się po niej i korzystać z niej zgodnie z wymaganiami. Jeśli jednak wybierzesz w menu przeglądarki opcję *Więcej*

narzędzi/Narzędzia dla developerów (lub wciśniesz przycisk *F12* na klawiaturze), zobaczysz przycisk *Przełącz pasek narzędzi urządzenia*, który ma pomóc programistom i testerom w sprawdzaniu poprawności ich aplikacji internetowych z responsywnego punktu widzenia. Gdy w tej opcji menu wybierzesz urządzenie iPhone, zobaczysz, że witryna nie jest gotowa dla urządzeń mobilnych. Tak naprawdę na górnym panelu tej aplikacji internetowej znajduje się nawet łącze do pobrania odpowiedniej aplikacji mobilnej zarówno dla Androida, jak i systemu iOS (rysunek 1.2).

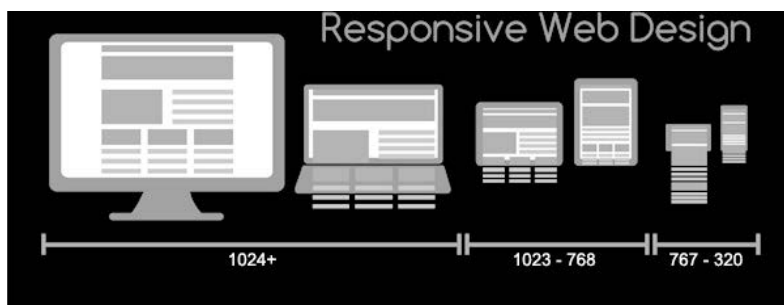


Rysunek 1.2. Aplikacja internetowa MSN uruchomiona na urządzeniu iPhone 13 Pro Max w trybie orientacji poziomej

Taki typ aplikacji internetowej jest opracowywany, testowany i przeznaczony przede wszystkim dla użytkowników komputerów stacjonarnych. Ponadto wymaga on od organizacji utrzymywania natywnej lub hybrydowej aplikacji charakterystycznej dla urządzeń mobilnych, aby zapewnić prawidłowe korzystanie ze wszystkich urządzeń cyfrowych.

Responsywne aplikacje internetowe

W przeciwieństwie do tradycyjnych aplikacji internetowych, **responsywne aplikacje internetowe** (ang. *Responsive Web Application*) można z reguły dostosowywać do rozmiarów i rozdzielczości ekranów komputerów stacjonarnych i urządzeń mobilnych. Taka unikatowa konstrukcja pozwala programistom aplikacji internetowych udostępniać swoje aplikacje przez maksymalną liczbę kanałów cyfrowych przy użyciu pojedynczej bazy kodu i spójnego doświadczenia użytkownika. Nie jest to jednak takie proste. Oprócz unikatowego przepływu pracy programistycznej responsywne aplikacje internetowe wymagają m.in. uzgodnionej strategii zawartości, czyli określenia najbardziej odpowiednich treści, które mają być wyświetlane w obszarze widocznym po załadowaniu strony i poza nim (rysunek 1.3). Takie widoczne zawartości na wszystkich rodzajach ekranów muszą być stale utrzymywane, optymalizowane i testowane, aby zapewnić rozwój i sukces firmy.



Rysunek 1.3. Projektowanie responsywnych stron internetowych dla laptopów, tabletów i smartfonów; rysunek autorstwa Muhammada Rafizeldiego (Mrafizeldi) pobrany ze strony https://commons.wikimedia.org/wiki/File:Responsive_Web_Design_for_Desktop,_Notebook,_Tablet_and_Mobile_Phone.png i udostępniony na licencji CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/deed.en>)

Responsywne aplikacje internetowe są o wiele bardziej wszechstronnym typem aplikacji, który zapewnia oczywiste korzyści zarówno dla programistów, jak i użytkowników końcowych. Utrzymywanie pojedynczej bazy kodu i automatyczne serwowanie dowolnego rozmiaru lub dowolnej rozdzielczości ekranu to opłacalne metody tworzenia oprogramowania.

Na rynku pojawiły się również nowe rodzaje platform cyfrowych, takie jak składane smartfony i urządzenia domowe (np. Google Hub i Amazon Echo Show). Tego typu aplikacje muszą być stale aktualizowane, aby zapewnić ciągłą podróż użytkownika przez wszystkie platformy.

Poniżej przedstawiłem listę elementów, które składają na plan testowy dla solidnego **projektowania responsywnych stron internetowych** (ang. *Responsive Web Design* — RWD), który powinien być wykonywany w sposób ciągły zarówno przez programistów, jak i inżynierów testowych:

- Testowanie kompatybilności najważniejszych przeglądarek na komputery stacjonarne, wersji systemów operacyjnych i urządzeń mobilnych.
- Przeprowadzanie testów wizualnych obejmujących różne układy, rozmiary ekranu, rozdzielczości i języki, aby zapewnić prawidłowe wyświetlanie wszystkich elementów graficznych na tych platformach.
- Przeprowadzanie testów funkcjonalnych e2e wszystkich przepływów biznesowych, linków, formularzy i innych zależności internetowego interfejsu użytkownika.
- Testowanie dostępności stron na różnych platformach.
- Przeprowadzanie testów wydajnościowych po stronie klienta.
- Przeprowadzanie testów obciążeniowych w okresach najwyższego i normalnego natężenia ruchu.
- Testowanie w różnych warunkach środowiskowych (zarówno internetowych, jak i mobilnych), obejmujące sieci, pracę z czujnikami, zdarzenia przychodzące, zdarzenia ze wskazaniem lokalizacji itd.

Progresywne aplikacje internetowe

Progresywne aplikacje internetowe (ang. *Progressive Web Application* — PWA) to jeden z najbardziej zaawansowanych typów aplikacji internetowych o unikatowych cechach. Ten typ aplikacji został opracowany przez firmę Google i przyjęty przez wszystkich innych dostawców przeglądarek, w tym Apple'a, Microsoft i Mozillę. PWA to aplikacje zbudowane na podstawie bazy kodu responsywnych aplikacji internetowych, umożliwiające użytkownikom mobilnym zainstalowanie łącza internetowego na swoich urządzeniach z systemami Android i iOS (rysunek 1.4). Użytkownicy mogą wtedy korzystać z tych aplikacji w trybie offline za pośrednictwem różnych czujników z dostępem do funkcji mobilnego systemu operacyjnego, takich jak lista kontaktów, kamera, lokalizacja itd.



Rysunek 1.4. Ikony PWA na urządzeniu mobilnym; rysunek „Progressive web apps on my home screen” autorstwa Jeremy’ego Keitha pobrany ze strony <https://www.flickr.com/photos/adactio/42535353742> i udostępniony na licencji CC BY 2.0 (<https://creativecommons.org/licenses/by/2.0/>)

Aplikacje PWA można instalować za pomocą linku na dowolnym urządzeniu mobilnym z systemem iOS lub Android, a także na laptopach hybrydowych z systemem Windows, takich jak Microsoft Surface. Po ich zainstalowaniu użytkownik może je uruchamiać z ekranu głównego urządzenia mobilnego i cieszyć się unikatowymi funkcjonalnościami aplikacji, które są przypisywane komponentowi **ServiceWorker** wbudowanemu w każde PWA. Na rysunku 1.4 ikona *Twitter Lite* to skrót aplikacji PWA, która została zainstalowana na urządzeniu mobilnym z poziomu przeglądarki internetowej.

Wątki robocze usług

Wątki (procesy) robocze usług (ang. *service workers*) to uruchamiane w tle skrypty przeglądarki użytkownika, umożliwiające twórcom aplikacji internetowych dodawanie funkcjonalności, takich jak powiadomienia push, buforowanie w trybie offline, angażowanie czujników urządzeń mobilnych i serwer proxy, który może obsługiwać różne żądania sieciowe z aplikacji internetowej. PWA wykorzystują wątki robocze usług do wzbogacania aplikacji internetowych działających na urządzeniach mobilnych.

Używając pojedynczej bazy kodu, aplikacje PWA oferują użytkownikom możliwość korzystania z aplikacji internetowej na dowolnym ekranie komputera stacjonarnego lub urządzenia mobilnego z dodatkowymi funkcjonalnościami, takimi jak buforowanie offline, powiadomienia push, obsługa czujników (lokalizatora, kamery i mikrofonu) oraz dostęp do listy kontaktów. Dzięki obsłudze tak wielu funkcjonalności twórcy aplikacji internetowych mogą łatwo i natychmiastowo wdrażać swoje aplikacje z pominięciem sklepów aplikacji mobilnych.

Google i inni dostawcy przeglądarek dostarczają narzędzia do walidacji PWA (w postaci funkcjonalności narzędzi dla deweloperów dostępnych w przeglądarce), a także inne wytyczne i zalecane praktyki. Programiści mogą generować pliki *MANIFEST.MF* i pisać w JavaScriptcie skrypty wątków roboczych usług dodawane do aplikacji internetowych. Wiele przedsiębiorstw — np. Instagram, Lyft, Twitter, eBay itd. (<https://www.pwastats.com/>) — w różnych segmentach rynku już wdrożyło tę technologię w swoich aplikacjach internetowych i codziennie dostrzega liczne korzyści. Jako lider w tego typu aplikacjach firma Google opracowała doskonale wytyczne (<https://web.dev/progressive-web-apps/>) oraz listę kontrolną do weryfikowania zgodności z wymaganiami PWA.

Na plan testowy progresywnych aplikacji internetowych składają się wspomniane wcześniej elementy planu testowego dla solidnego RWD wraz z następującymi scenariuszami charakterystycznymi dla PWA:

- Sprawdzanie poprawności plików manifestów PWA (wskazujących adres URL strony głównej, motyw, ikonę itd.).
- Walidacja wątków roboczych usług PWA, która obejmuje kluczowe komponenty aplikacji, w tym rejestrację powiadomień push, funkcjonalności buforowania itp.
- Walidacja instalowania i funkcjonowania aplikacji PWA na całej platformie i równolegle z aplikacjami natywnymi.
- Sprawdzanie, czy PWA zapewniają niestandardową stronę do obsługi użytkowników w trybie offline.
- Sprawdzanie, czy PWA współpracują z dowolnym typem urządzeń wejściowych, takich jak mysz, klawiatura, rysik lub ekran dotykowy.
- Sprawdzanie, czy PWA współpracują poprawnie ze wszystkimi czujnikami urządzeń mobilnych, takimi jak lokalizator, mikrofon, kamera itp.
- Testowanie PWA pod kątem wszystkich zewnętrznych zależności, takich jak integracja z mediami społecznościowymi (np. Facebookiem i LinkedInem), inne interfejsy API, usługi analityczne itd.
- Sprawdzanie, czy zabezpieczenia, reguły prywatności i uprawnienia aplikacji PWA są zgodne z wymaganiami Apple'a i Google'a.

Jak mogłeś się zorientować, jest to nadzbiór filarów planu RWD, uwzględniający dodatkowe testy charakterystyczne dla PWA.

Poniższy fragment kodu JavaScriptu przedstawia rejestrację podstawowego wątku roboczego usługi w ramach budowania fundamentu aplikacji PWA:

```

If ('serviceWorker' in navigator) {
  window.addEventListener('load', function(){
    navigator.serviceWorker.register('/sw.js').then(function(registration) {
      // Rejestracja się powiodła
      console.log('ServiceWorker zarejestrowany z zakresem:', registration.scope);
    } function(err){
      // Rejestracja się nie powiodła ☹️
      console.log('ServiceWorker nie został zarejestrowany:', err);
    });
  });
};

```

Skoro poznałeś już główne typy aplikacji internetowych dostępne dla programistów, omówię teraz najważniejsze typy testów, które należy uwzględnić przy wydawaniu każdej wersji aplikacji internetowej.

Typy testów dla aplikacji internetowych

Na testowanie oprogramowania składa się kilka typów testów: funkcjonalne, API, integracyjne, нефункционалне, jednostkowe, dostępności i testy eksploracyjne ad hoc. W tym rozdziale omówię tylko ogólne testy funkcjonalne i нефункционалне, natomiast dalej w książce przy okazji przeglądu konkretnych frameworków do automatyzacji testów przyjrzymy się również testom API i pozostałym typom testów. Do każdego z tych typów można zastosować podział w ramach tradycyjnej piramidy testowej oraz zakres na podstawie tego, czy jest to wydanie, czy mała poprawka.

W tym podrozdziale wskażę najważniejsze aspekty testowania dla wszystkich wymienionych typów w kontekście nowoczesnych aplikacji internetowych.

W przypadku aplikacji internetowych, które są przeznaczone do pracy na dowolnym komputerze stacjonarnym oraz mobilnym systemie operacyjnym i urządzeniu, pokrycie testami wszystkich aspektów aplikacji ma kluczowe znaczenie dla zapewnienia najwyższej klasy doświadczenia użytkownika.

Aplikacje internetowe opierają się na ciągłych interakcjach między komponentami, interfejsami użytkownika (warstwą prezentacji), bazami danych, mikrouslugami komunikującymi się przez bramę API z innymi komponentami, serwerami, różnymi interfejsami API, takimi jak płatności, uwierzytelnianie, przepływ pracy biznesowej (warstwa biznesowa) itd.

Jak wspomniałem wcześniej, testowanie tego typu aplikacji, które mają wiele warstw architektury, jest trudnym zadaniem, zwłaszcza w kontekście programowania zwinnego (ang. *agile software development*) i DevOps. W cyklu rozwoju oprogramowania bierze udział duża liczba osób, wiele razy dziennie zatwierdzone są różne zmiany kodu (tj. żądania *pull*), a wszystko to powinno zostać odpowiednio sklasyfikowane i przetestowane.

Testy funkcjonalne aplikacji internetowych

Przyjrzyjmy się kluczowym obszarom w kategorii testów funkcjonalnych aplikacji internetowych. Pamiętaj, że takie scenariusze testowe można podzielić na różne rodzaje testów, w tym testy kondycji, regresji, dymne, integracyjne i testy użyteczności przeprowadzane przez klienta. Zakres zestawu testowego powinien być określony przez **cykl rozwoju oprogramowania** (ang. *Software Development Life Cycle* — SDLC), zmiany wprowadzane w ramach iteracji oprogramowania oraz historię usterek aplikacji internetowej (tj. stabilność i niezawodność).

Na poniższej liście umieściłem kilka sugerowanych filarów testowych, które powinieneś wziąć pod uwagę w planach testowania aplikacji internetowych. Niezależnie od tego, czy będziesz przeprowadzać walidację tych scenariuszy za pomocą testów ręcznych, czy zautomatyzowanych, są to podstawowe filary, które zapewniają prawidłowe działanie aplikacji internetowych w całej interakcji użytkownika z daną witryną:

- Sprawdzanie działania linków do stron internetowych w całej witrynie, w tym:
 - linków nawigacji,
 - linków do mediów społecznościowych,
 - linków mailto.
- Przetestowanie dla odpowiednich stron witryny internetowych formularzy, takich jak formularze rejestracyjne i formularze zamówień:
 - testowanie pól formularza (dodatnie lub ujemne wartości wejściowe),
 - weryfikacja wypełniania pól obowiązkowych,
 - wysyłanie formularzy na wszystkich platformach (mobilnych i stacjonarnych).
- Testowanie reguł aplikacji internetowej, dotyczących plików cookie:
 - sprawdzenie usuwania plików cookie po wyczyszczeniu internetowej pamięci podręcznej.
- Weryfikacja całego przepływu biznesowego użytkownika w ramach witryny internetowej:
 - testowanie działania wszystkich linków wewnętrznych i nawigacji użytkownika,
 - testowanie interfejsu użytkownika i układu,
 - testowanie usług lokalizacyjnych,
 - testowanie kompatybilności strony internetowej we wszystkich rozmiarach i rozdzielczościach ekranu,
 - testowanie użyteczności i doświadczenia użytkownika.

Testy нефunkcjonalne aplikacji internetowych

W celu zagwarantowania wysokiej jakości aplikacji internetowej niezbędne jest uzupełnienie testów funkcjonalnych o testy нефunkcjonalne. Ostatecznie nie ma znaczenia, czy nieprawidłowe działanie aplikacji w środowisku produkcyjnym będzie spowodowane awarią funkcjonalną, czy problemami z dostępnością generowanymi przez usterki związane z obciążeniem.

Uwzględnienie wszystkich rodzajów testów w ramach zadań związanych z **ciągłą integracją** (ang. *continuous integration* — CI) odróżnia wydajny zespół zwinny od powolnego. Takie typy testów powinny obejmować zarówno testy funkcjonalne, jak i нефункционалne, które są uruchamiane za pośrednictwem frameworków automatyzacji przy wszelkich zmianach kodu.

Jeśli chodzi o działania związane z testami нефункционалnymi, zespoły powinny zazwyczaj uwzględnić testy bezpieczeństwa (zarówno statyczne, jak i dynamiczne), testy wydajności i obciążenia oraz testy ułatwienia dostępu. W niektórych praktykach zespoły mogą zaliczać testy ułatwienia dostępu do testów funkcjonalnych, ale niezależnie od kategorii, do której zaliczone zostaną te typy testów, wszystkie są istotne. Ponadto wykonywanie jak największej liczby dostarczających wartość typów testów decyduje o gotowości i wysokiej jakości aplikacji internetowej — oczywiście jeśli wszystkie zakończą się powodzeniem 😊

Testowanie bezpieczeństwa

Testy bezpieczeństwa obejmują następujące elementy:

- zapewnienie autoryzacji dostępu do zabezpieczonych stron,
- uniemożliwienie użytkownikom pobierania zastrzeżonych plików bez odpowiednich uprawnień dostępu i uwierzytelniania,
- automatyczne zamykanie sesji przy bezczynności użytkownika,
- przekierowywanie witryn na strony szyfrowane protokołem SSL przy użyciu certyfikatów SSL,
- stosowanie sprawdzonych w branży testów, takich jak OWASP Top 10, CWE, CERT itd.,
- stosowanie standardów jakości kodu, takich jak JSLint (<https://www.jshint.com/>), w ramach statycznych testów bezpieczeństwa aplikacji (ang. *Static Application Security Testing* — SAST) i dynamicznych testów bezpieczeństwa aplikacji (ang. *Dynamic Application Security Testing* — DAST).

Testowanie wydajności i obciążenia

Testy wydajności i obciążenia obejmują następujące elementy:

- pomiar czasów reakcji aplikacji względem wzorców i współczynników KPI w zależności od różnych warunków sieciowych (dla aplikacji internetowych i mobilnych),
- testowanie obciążenia aplikacji internetowej, aby określić jej zachowanie w warunkach normalnych (wydajność w przypadku pojedynczego użytkownika) i dużego natężenia ruchu (milionów wirtualnych użytkowników),
- przeprowadzanie stress testów aplikacji internetowej w celu określenia jej wartości granicznej, gdy zostanie poddana wyższemu niż normalne obciążeniu w godzinach szczytu,
- określanie, w jaki sposób witryna odzyskuje sprawność po awariach lub problemach z dostępnością.

Testowanie ułatwień dostępu

Testy ułatwień dostępu obejmują:

- uwzględnienie najbardziej rozpowszechnionych norm ułatwień dostępu: WCAG, ADA, § 508 oraz ACA (w Kanadzie),
- przeprowadzenie testów ułatwień dostępu dla różnych platform i języków (dla aplikacji internetowych i mobilnych),
- zapewnienie odpowiednich identyfikatorów ułatwień dostępu (elementów internetowych) w celu usprawnienia automatyzacji testów.

Jak wspomniałem wcześniej, w planie testów każdej organizacji powinno zostać uwzględnione połączenie testów eksploracyjnych i testów zautomatyzowanych zarówno dla funkcjonalnych, jak i niefunkcjonalnych obszarów aplikacji internetowej. Ponadto testy należy przeprowadzać w sposób ciągły w celu dostosowywania się do najnowszych zmian w aplikacji internetowej, wad wykrytych w środowisku produkcyjnym, zmian na rynku platform, takich jak nowe wersje systemu operacyjnego lub urządzeń mobilnych, oraz zmian w standardach branżowych, takich jak ułatwienia dostępu i nowe reguły bezpieczeństwa.

Dalej w książce omówię konkretne przykłady i narzędzia pomocne przy przeprowadzaniu większości typów testów zarekomendowanych w tym podrozdziale.

Skoro poznałeś już podstawowe typy testów, które mają zastosowanie do aplikacji internetowych, w następnym podrozdziale skupię się na głównych różnicach między używaniem przeglądarek interfejsowych i bezinterfejsowych podczas faz programowania i testowania.

Zastosowanie przeglądarek interfejsowych i bezinterfejsowych podczas tworzenia i testowania aplikacji

Oprócz wyboru frameworków i języków programowania twórcy i testerzy aplikacji internetowych mają również możliwość wyboru, czy do testowania swoich aplikacji internetowych używać przeglądarek z załadowanym graficznym interfejsem użytkownika (ang. *Graphical User Interface* — GUI), czy bez niego. Mówimy wtedy odpowiednio o przeglądarkach interfejsowych (ang. *headed browser*) i bezinterfejsowych (ang. *headless browser*, zwanych również potocznie przeglądarkami bezgłowymi).

Przeglądarki bezinterfejsowe

Przeglądarka bezinterfejsowa to normalna przeglądarka, która podczas działania nie ładuje po prostu swojego graficznego interfejsu użytkownika.

Decyzja dotycząca sposobu użycia przeglądarki zależy od celów, jakie chce osiągnąć programista lub tester. W kolejnym punkcie omówię te metody nieco szerzej.

Wybór między przeglądarkami interfejsowymi i bezinterfejsowymi

Korzystanie z przeglądarki bezinterfejsowej może być niezwykle korzystne, gdy nie ma potrzeby eksplorowania żadnych elementów lub działań w interfejsie przeglądarki, a głównym celem jest zweryfikowanie jedynie poprawności uruchamiania testów lub wykonywania innych działań w przeglądarce. Przeglądarki bezinterfejsowe mogą być bardzo opłacalne również w przypadku równoległego uruchamiania ogromnego zestawu testów, w których nie ma potrzeby *przeglądania* interfejsu przeglądarki. Takie wykonywanie będzie przebiegać znacznie szybciej ze względu na mniejsze użycie pamięci i innych zasobów sprzętowych wykorzystywanych zwykle przez przeglądarki interfejsowe, które do wyrenderowania stron HTML-owych potrzebują ponadto czasu na zainicjowanie środowiska. Można rozważyć także przypadek użycia dla testowania paralelnego na dużą skalę jako części regresji budowanej w ramach CI po testach opartych na interfejsie użytkownika dla różnych przeglądarek lub równoległe z nimi.

Zwróć uwagę, że programiści i testerzy nie mogą polegać wyłącznie na testowaniu z wykorzystaniem przeglądarek bezinterfejsowych, które jest trudniejsze do debugowania i nie zawsze ujawnia rzeczywiste wrażenia użytkownika końcowego na różnych platformach. Połączenie testów interfejsowych z testami bezinterfejsowymi powinno być strategicznie zaplanowane i przeprowadzane przez różne zespoły. Wszyscy główni dostawcy przeglądarek, w tym Google, Mozilla i Microsoft, oferują opcję bezinterfejsową, którą użytkownik końcowy może włączyć za pomocą flag wiersza poleceń lub w ramach różnych frameworków automatyzacji testów, takich jak Selenium i Puppeteer.

Selenium oferuje dobry zestaw przykładów kodu, które można wykorzystać do uruchomienia dowolnej obsługiwanej przeglądarki internetowej w trybie interfejsowym lub bezinterfejsowym. Poniżej pokazałem przykładową konfigurację frameworku Selenium 4 (<https://github.com/SeleniumHQ/selenium/blob/trunk/javascript/node/selenium-webdriver/chrome.js#L333>), który uruchamia przeglądarkę Chrome w trybie bezinterfejsowym:

```
let driver = new Builder()
  .forBrowser('chrome')
  .setChromeOptions(new chrome.Options().headless())
  .build();
```

Framework Selenium omówię dalej w książce i dowiesz się wtedy, jak używać go zarówno w trybie interfejsowym, jak i bezinterfejsowym. Większość frameworków testowych, takich jak Selenium, Playwright i Cypress, obsługuje te dwie metody testowania aplikacji internetowych.

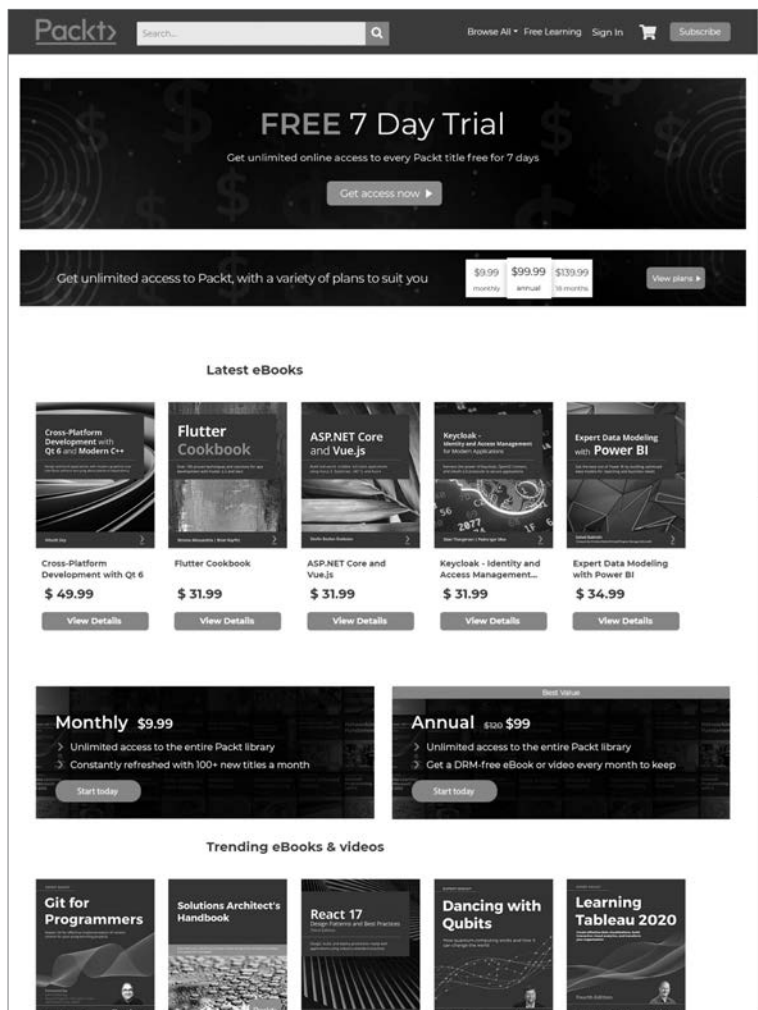
Aby korzystać z różnych przeglądarek z poziomu interfejsu wiersza poleceń, programiści i testerzy mogą wykorzystać dziesiątki opcji do robienia zrzutów ekranu, zdalnego debugowania aplikacji internetowej itd.

Oto prosta opcja wiersza poleceń, która wykorzystuje zbudowaną na Chromium bezinterfejsową przeglądarkę Microsoft Edge do przechwycenia zrzutu ekranu strony głównej witryny PacktPub:

```
Msedge --headless --disable-gpu --screenshot=c:\[.]\packt.png --window-size=1280,1696 https://www.packtpub.com/
```

Przed uruchomieniem powyższego polecenia upewnij się, że umieściłeś przeglądarkę Edge w systemowej ścieżce środowiskowej.

Jak widać na rysunku 1.5, przeglądarka przechwyciła stronę główną z rozmiarem okna podanym w wierszu poleceń.



Rysunek 1.5. Zrzut ekranu strony internetowej PacktPub wykonany za pomocą wiersza poleceń bezinterfejsowej przeglądarki Edge

Frameworki testowania przy użyciu przeglądarek bezinterfejsowych

Skoro dowiedziałeś się już, czym jest szybkie, ekonomiczne i dość łatwe w użyciu środowisko przeglądarki bezinterfejsowej, przyjrzyjmy się frameworkowi automatyzacji o nazwie **Puppeteer** (<https://developers.google.com/web/tools/puppeteer>), który dobrze współpracuje z przeglądarką bezinterfejsową Chrome. Puppeteer jest biblioteką środowiska Node opracowaną przez Google'a i wyposażoną w wysokopoziomowy interfejs API do sterowania bezinterfejsową przeglądarką Chrome za pośrednictwem protokołu DevTools. Ma wszystkie zalety przeglądarki Chrome, w tym przysyłanie formularzy i wprowadzanie danych przez użytkownika, a także dodatkowe funkcjonalności charakterystyczne dla przeglądarki bezinterfejsowej, takie jak m.in. pomiar wydajności działania aplikacji internetowej.

Microsoft rozwija analogiczny, oparty na Puppeteerze framework o nazwie **Playwright**. Omówię go szerzej dalej w książce.

Aby rozpocząć korzystanie z tego rozwiązania bezinterfejsowego, uruchom następujące polecenie `npm install`:

```
npm install puppeteer
```

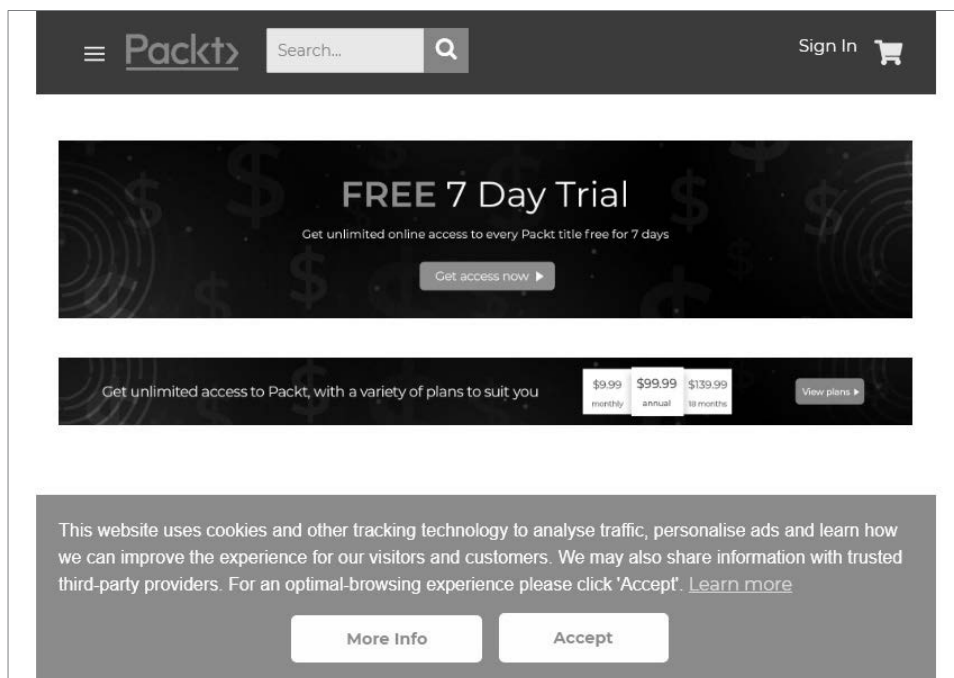
Po zainstalowaniu tego rozwiązania możesz pisać skrypty w języku JavaScript i korzystać z interfejsów API dostępnych w tym frameworku. Używając poniższego fragmentu kodu jako przykładu, możesz automatycznie przejść do określonej witryny i zrobić zrzut ekranu:

```
const puppeteer = require('puppeteer');
(async() => {
  const browser = await puppeteer.launch({headless:false}); // Wartość domyślna to true
  const page = await browser.newPage();
  await page.goto('https://www.packtpub.com');
  await page.screenshot({path: 'Packt.png'});
  await browser.close();
})();
```

W przypadku ustawienia dla flagi `headless` wartości `false` wykonanie kodu spowoduje uruchomienie wbudowanej przeglądarki Chrome.

Na rysunku 1.6 przedstawiłem zrzut ekranu z wykonania powyższego przykładu kodu.

Poprzedni przykład jest prostym przypadkiem użycia Puppeteera. Framework ten może jednak rozszerzać możliwości protokołu DevTools i generować za pomocą zautomatyzowanego kodu plik archiwum HTTP (HAR) do analizy bezpieczeństwa, wydajności i innego ruchu internetowego. W najnowszej wersji frameworku Selenium 4 oraz w Cypressie, aby skorzystać z niektórych funkcjonalności Puppeteera, możesz również używać protokołu **CDP** (ang. *Chrome DevTools Protocol*).



Rysunek 1.6. Zrzut ekranu strony głównej Packta wykonany za pomocą javascriptowego testu Puppeteera w trybie bezinterfejsowym

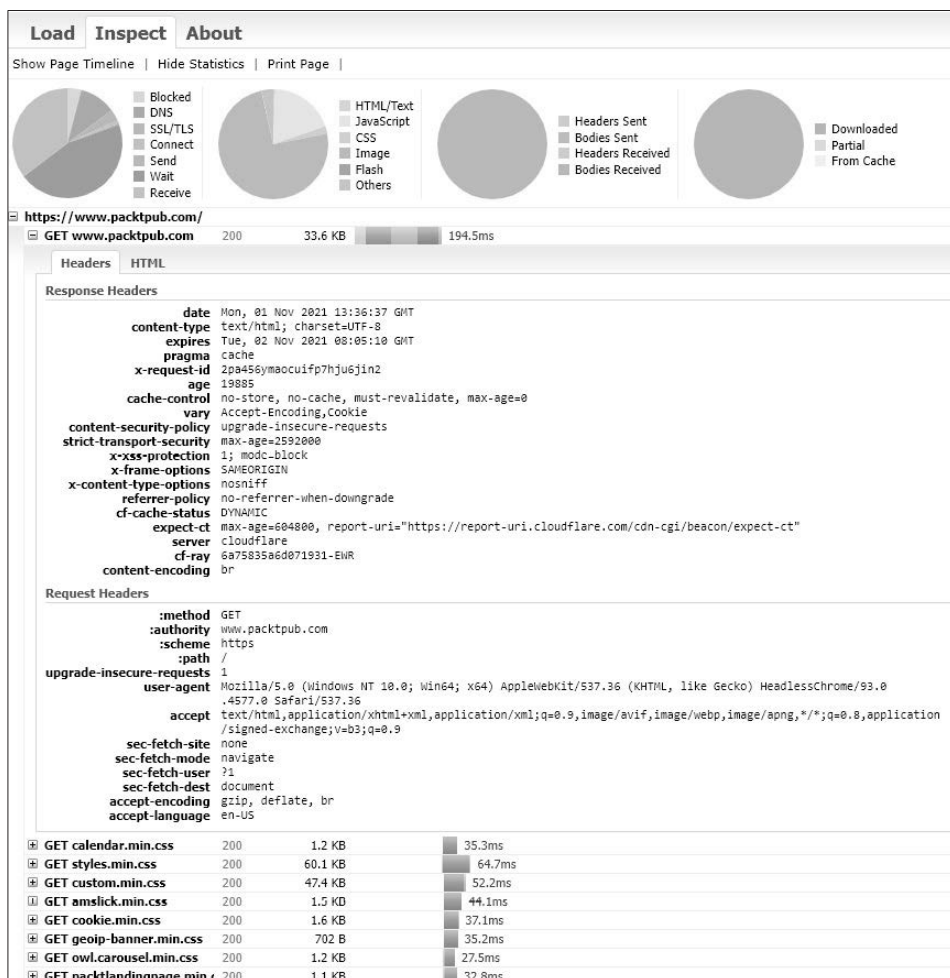
W celu wygenerowania pliku HAR w ramach skryptu automatyzacji testów powinieneś w swoich scenariuszach automatyzacji testów uwzględnić następujące linie kodu (po zainstalowaniu modułu `puppeteer-har` Node'a):

```
const puppeteerHar = require('puppeteer-har');
const har = new puppeteerHar(page);
await har.start({path: 'results.har'});
await page.goto('https://www.packtpub.com');
await har.stop();
```

Jeśli dodasz powyższy kod do wcześniejszego przykładu zrzutu ekranu, zostanie wygenerowany plik *results.har* ze strony PacktPub. Do analizy wygenerowanego zasobu możesz użyć dowolnej przeglądarki plików HAR lub dodać do przeglądarki Google Chrome rozszerzenie HTTP Archive Viewer.

Podczas badania wygenerowanego pliku HAR możesz uzyskać wgląd w czasy wczytywania stron, statystyki stron, żądania witryn i szczegóły nagłówków odpowiedzi, jak pokazałem na rysunku 1.7.

Te spostrzeżenia możesz potem wykorzystać do zoptymalizowania wydajności witryny, wykrycia luk w zabezpieczeniach itd.



Rysunek 1.7. Zrzut ekranu pliku HAR strony głównej Packta, wygenerowanego za pomocą zautomatyzowanego skryptu Puppeteera

Jak wspomniałem wcześniej, firma Google zaprojektowała narzędzie przeglądarki bezinterfejsowej, aby pomóc programistom testować i debugować aplikacje internetowe. Aby skutecznie debugować aplikację internetową podczas pracy w trybie bezinterfejsowym, przeglądarki te zapewniają ponadto funkcjonalność zdalnego debugowania, której możesz używać ręcznie z poziomu wiersza poleceń lub w ramach zautomatyzowanego kodu JavaScriptu:

```
--remote-debugging-port=9222 (example)
```

Jeśli uruchomisz testy w trybie bezinterfejsowym i dodasz to polecenie, możesz skorzystać z interfejsowej przeglądarki Chrome, aby otworzyć stronę `http://localhost:9222` i sprawdzić wszystkie dane wyjściowe z wykonania testów.

Podsumowanie

W dzisiejszych czasach zbudowanie dobrej aplikacji internetowej jest trudniejsze niż kiedykolwiek ze względu na postępującą masową transformację cyfrową i ewentualne koszty związane ze zniszczeniem reputacji marki, gdy coś pójdzie nie tak. Zastosowanie wszystkich typów testów na wczesnych etapach rozwoju oprogramowania i wykorzystanie różnych metod, narzędzi i dostępnych w przeglądarce funkcjonalności może być świetnym początkiem w kontekście tworzenia planu jakości dla aplikacji internetowej. Taki plan musi obejmować wszystkie funkcjonalne i niefunkcjonalne aspekty testów. Ponadto należy wziąć pod uwagę narzędzia, które obniżają koszty i oszczędzają czas, takie jak testowanie przy użyciu przeglądarek bezinterfejsowych, narzędzia dla programistów aplikacji internetowych, pliki HAR i inne techniki, o których wspomniałem w tym rozdziale.

Na początku rozdziału opisałem zaawansowany krajobraz internetowy i nowe współczesne typy aplikacji. Potem zdefiniowałem i omówiłem pokrótce responsywne aplikacje internetowe (PWA) oraz wskazałem właściwy sposób zapewniania jakości tego typu aplikacji. Przedstawiłem ponadto różne typy testów dostępne dla programistów i inżynierów testowych oraz przeanalizowałem poszczególne typy testów w kontekście internetowego przypadku użycia.

Po omówieniu tych tematów wprowadziłem koncepcję korzystania z przeglądarek bezinterfejsowych w połączeniu z przeglądarkami interfejsowymi w ramach przepływu pracy rozwoju oprogramowania, aby usprawnić przekazywanie informacji zwrotnych, uwzględnić konfigurację, wydajność i stabilność środowiska oraz zwiększyć skuteczność debugowania w prawdziwych przeglądarkach.

Podsumowałem kilkoma stwierdzeniami dotyczącymi ogólnych kwestii testowania na różnych przeglądarkach.

Na tym kończy się ten rozdział! Mam nadzieję, że pomoże Ci to lepiej poznać krajobraz aplikacji internetowych i proces budowania odpowiedniej strategii testowania dla aplikacji internetowych, które będziesz tworzył.

W następnym rozdziale omówię kluczowe wyzwania, z jakimi muszą się mierzyć twórcy aplikacji internetowych, i spróbuję wyjaśnić przyczynę tego stanu rzeczy.

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Automatyzacja testów aplikacji — tak zagwarantujesz niezawodność!

Nawet bardzo doświadczony zespół niekiedy popełnia błędy, które mogą przesądzić o porażce obiecującego projektu. Aby uniknąć takich sytuacji, trzeba zadbać o odpowiednie testowanie kodu. To jednak bywa prawdziwym wyzwaniem dla frontendowców, którzy na co dzień skupiają się na innych aspektach pracy aplikacji. Obecnie deweloperzy mogą wybierać spośród rozlicznych narzędzi do testowania i wielu nowych metodyk. Na uwagę zasługują zwłaszcza frameworki służące do automatyzacji testów wieloprzeglądarkowych, dostępne na licencji open source.

Oto przewodnik po koncepcjach testowania i wiodących frameworkach, za pomocą których automatyzuje się testy aplikacji internetowych, takich jak Selenium, Cypress, Puppeteer i Playwright. Zaprezentowano w nim unikatowe funkcjonalności tych rozwiązań, ich wady i zalety, a także wyjaśniono zasady konfiguracji każdego z nich, aby testowanie przebiegało bezawaryjnie nawet po wprowadzeniu zmian w kodzie. Dzięki tej książce nie tylko wybierzesz najlepszy framework, ale także zintegrujesz go z przepływem pracy programowania frontendowego i utworzysz wstępny zestaw automatyzacji testów oparty na JavaScriptcie. Zapewni to szybką informację zwrotną przy zmianach w kodzie i zwiększy niezawodność automatyzacji testów.

W książce między innymi:

- wybór narzędzi do testowania aplikacji
- zaawansowana automatyzacja testów
- pomiar pokrycia kodu i pokrycia testowego w ocenie jakości aplikacji
- kompromisy przy wyborze narzędzi do testowania
- frameworki: Cypress, Selenium, Playwright i Puppeteer
- przegląd narzędzi do niskokodowego testowania aplikacji internetowych

Eran Kinsbruner jest autorem bestsellerów, doświadczonym programistą i zapalonym testerem oprogramowania. Pracował dla takich firm jak Sun Microsystems, Neustar, Texas Instruments i General Electric. Aktywnie działa w społeczności rozwijającej oprogramowanie. W 2021 roku był nominowany do nagrody DevOps Dozen w konkursie DevOps.com.

Helion 	KOD KORZYŚCI Śięgnij po więcej! 	
 helion.pl	ISBN 978-83-283-9864-1	
 HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	 9 788328 398641	
Cena: 69,00 zł		

Packt