OKIEM EKSPERTA

Helion

Testowanie i rozwój API w Postmanie

Łatwe tworzenie, testowanie, debugowanie i zarządzanie API

Wydanie II

Dave Westerveld

packt

Tytuł oryginału: API Testing and Development with Postman: API creation, testing, debugging, and management made easy, 2nd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-2478-9

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/teroz2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Rozdział 2

Projektowanie i dokumentowanie API

Uważam, że projektowanie, testowanie i dokumentacja są w tworzeniu wysokiej jakości produktu równie istotne jak samo programowanie. Być może są nawet ważniejsze. Może się wydawać zaskakujące, że książka o testowaniu poświęca tyle uwagi dokumentacji i projektowaniu, ale obie te praktyki mają kluczowy wpływ na jakość końcowego produktu.

Dobry projekt może znacznie ułatwić pracę programistom i testerom. Projektowanie czegoś nie polega tylko na tym, aby wyglądało to ładnie, ale przede wszystkim na tym, aby przynosiło satysfakcję. My, ludzie, lubimy rzeczy, które dobrze wyglądają, więc projekty czasem koncentrują się na wyglądzie i odczuciach, jakie wywołują. Jednak nawet w przypadku interfejsów programowania aplikacji (API), w których nie ma za bardzo „co oglądać", dobry projekt może sprawić, że korzystanie z nich będzie o wiele przyjemniejsze, a tym samym poprawi ich jakość.

Dobra dokumentacja znacznie ułatwia pracę programistom i testerom. Sam zmagałem się z API, do których nie było dokumentacji, a nawet jeśli była, to okazała się niepoprawna. To znacznie utrudnia pracę z takim API i jego testowanie. Można stracić godziny na próbach zrozumienia, jak wykonać wywołanie API, tylko dlatego, że brakuje jednego parametru niezbędnego do jego działania. W przypadku dobrej dokumentacji to można ustalić w ciągu zaledwie kilku minut.

W tym rozdziale przedstawię reguły dotyczące dokumentowania i projektowania API. Przy okazji omówię następujące zagadnienia:

- określanie przeznaczenia API,
- tworzenie użytecznych API,
- dokumentowanie API (za pomocą Postmana),
- przykład projektowania API (ćwiczenie praktyczne).

Wprawdzie przedstawiony materiał może wydawać się nieco abstrakcyjny i teoretyczny, ale zamieściłem kilka ćwiczeń, które pomogą zrozumieć, jak wykorzystać te koncepcje w praktyce. Zachęcam, by poświęcić czas na ich wykonanie. Dzięki lekturze rozdziału będziesz w stanie wykorzystać omówione reguły projektowania do tworzenia wartościowych rozwiązań dla API, nad którymi obecnie pracujesz. Zdobędziesz też solidne

podstawy, które przydadzą się, gdy trzeba będzie opracować zupełnie nowe API. Zamieszczone tutaj informacje pomogą również rozpocząć pracę z dokumentacją API w Postmanie i pokażą, jak używać języka specyfikacji RAML do projektowania i modelowania API oraz automatycznego dodawania żądań do Postmana.

Wymagania techniczne

Kod wykorzystany w rozdziale znajduje się w repozytorium GitHub dostępnym pod adresem *https://github.com/PacktPublishing/API-Testing-and-Development-with-Postman-Second-Edition/tree/main/Chapter02*.

Rozpocznij od wyznaczenia celu

Nie tworzymy API tylko dlatego, że to świetna zabawa. Tworzymy je, aby osiągnąć konkretny cel. Mają one pomóc nam lub naszym użytkownikom rozwiązać jakiś problem. Może się to wydawać oczywiste, ale bądźmy szczerzy: zbyt często o tym zapominamy.

Projektowanie to trudna dziedzina. Być może w Twojej firmie pracują projektanci (podobnie jak w mojej). Osoby ukierunkowane techniczne, np. testerzy i programiści, czasami lekceważą pracę projektantów, uważając, że „tylko upiększają rzeczy". Jednak w rzeczywistości dobre projektowanie jest bardzo wymagające. Dobry projekt tworzy coś, co jest dostosowane do swojego celu. Ta książka nie jest poświęcona teorii projektowania (jeśli chcesz dowiedzieć się więcej na ten temat, sięgnij po pozycje takie jak *The Design of Everyday Things* Dona Normana). Natomiast jeśli zależy Ci na wysokiej jakości interfejsie programowania aplikacji, musisz poświęcić trochę czasu na przemyślenie celu tego API i sposobu, w jaki je zaprojektujesz, aby ten cel osiągnąć.

Świadomość, że trzeba coś zrobić, to ważny pierwszy krok. Niestety, wiele prezentacji, artykułów i książek na tym poprzestaje. Ale jaki z tego pożytek? Jesteś przekonany, że rozsądne jest projektowanie API z myślą o wyznaczonym celu, ale co nim jest? Co zrobić, jeśli go nie znasz? Jak określić przeznaczenie budowanego API?

Określanie przeznaczenia API

Chcesz określić cel swojego API, ale zastanawiasz się, jak to zrobić? Istnieje kilka prostych strategii, które możesz wykorzystać w tym celu. Pierwszą rzeczą jest zadawanie pytań. Jeśli Ty lub Twój zespół tworzycie API, prawdopodobnie ktoś Was o to poprosił. Porozmawiaj z tą osobą! Ustal, dlaczego prosiła o opracowanie API. Co chce z nim zrobić? Jakie problemy jej zdaniem rozwiąże to API? Zadaj tego rodzaju pytania i spróbuj się dowiedzieć, jaki ma być cel opracowania API.

Persony

Inną prostą strategią, którą możesz zastosować, jest wykorzystanie person. Persona to po prostu fikcyjna postać, którą tworzysz, aby lepiej zrozumieć, kto może korzystać z Twojego API. Na przykład możesz utworzyć personę reprezentującą pracownika firmy, który będzie używał API podczas opracowywania interfejsów użytkownika, lub personę inżyniera oprogramowania pracującego dla jednego z Twoich klientów, który wykorzystuje API do tworzenia niestandardowej aplikacji. Przeanalizowanie różnych typów użytkowników i rozważenie, w jaki sposób mogliby korzystać z API, pomoże zrozumieć cel, jakiemu ono służy.

Tworząc personę, warto wyjść poza aspekty techniczne, które dana osoba może znać lub wykorzystywać. Zastanów się nad jej celami, motywacjami i tym, co może ją frustrować. Pomocne jest też dodanie osobistych szczegółów, które sprawią, że persona będzie bardziej autentyczna. Na przykład czy lubi psy albo czy ma dzieci? Co czyni ją wyjątkową? Innymi słowy — jakim jest człowiekiem? Zapisywanie takich szczegółów może się wydawać nieco dziwne, ale pomaga lepiej wczuć się w tę personę. Dzięki temu łatwiej będzie spojrzeć na świat jej oczami i zrozumieć, co jest dla niej ważne. Im lepiej to zrozumiesz, tym skuteczniej będziesz w stanie określić cel tworzonego API.

Dlaczego?

Gdy ustalasz przeznaczenie API, podstawowe pytanie, które się wówczas pojawia, brzmi „dlaczego?". Dlaczego ma być utworzone dane API? Odpowiedź prawie zawsze wiąże się z rozwiązaniem jakiegoś problemu. Świetnym sposobem na ustalenie przeznaczenia API jest określenie, jaki problem ono rozwiązuje. Czy ułatwia opracowanie elementów interfejsu użytkownika? Czy umożliwia klientom dostosowywanie aplikacji do własnych potrzeb? Czy pozwala zewnętrznym programistom korzystać z Twojej platformy? Czy upraszcza integrację z innymi aplikacjami? Jaki problem rozwiązuje tworzone API? Odpowiedź na te pytania to dobry punkt wyjścia, aby zrozumieć przeznaczenie dla tworzonego API.

Uwaga

Ćwiczenie polegające na odkrywaniu przeznaczenia API nie dotyczy tylko nowych interfejsów. Jeśli pracujesz z istniejącym API, jego zrozumienie zapewnia ogromną wartość. Może być już za późno na radykalną zmianę projektu API. Nie ma większego zagrożenia dla jakości produktu niż podejmowanie nieskutecznych prób pomagania użytkownikom w rozwiązywaniu ich problemów. W najgorszym razie zrozumienie istniejącego API, które testujesz, pomoże ustalić, które błędy są ważne, a które mogą mieć mniejsze znaczenie. Pewnej wprawy wymaga nabycie umiejętności rozróżnienia, którymi błędami trzeba się pilnie zająć. Zrozumienie przeznaczenia API na pewno bardzo w tym pomoże.

Wypróbuj to

Właśnie przekazałem kilka praktycznych wskazówek, ale nie przyniosą one wiele pożytku, jeśli ich nie zastosujesz. Niniejsza książka nie służy tylko do przedstawienia teorii; ma na celu pomóc w lepszym testowaniu API. Nieco później omówię wybrane

narzędzia, jakie Postman oferuje w zakresie projektowania API. Jednak wcześniej zachęcam do zrobienia sobie przerwy na wypróbowanie tego, o czym właśnie mówiliśmy.

Weź istniejące API, nad którym pracujesz, i spróbuj opisać jego przeznaczenie w dwóch lub trzech zdaniach. W trakcie tego procesu możesz zastosować wymienione tutaj kroki.

1. Wskaż co najmniej dwóch kluczowych zainteresowanych danym API. Zrób to w formie odpowiedzi na pytanie „Kto chce (lub chciał) opracowania tego API?". Zapisz imiona tych osób.
2. Jeśli to możliwe, porozmawiaj z zainteresowanymi i zapytaj ich, jakie ich zdaniem powinno być przeznaczenie danego API i dlaczego chcą, żeby je opracować. Zapisz ich odpowiedzi.
3. Utwórz co najmniej jedną, a najlepiej dwie persony opisujące typy osób, które Twoim zdaniem będą korzystać z API. Jaki jest ich poziom umiejętności? Jakie zadania starają się wykonać? W jaki sposób pomoże im to API?
4. Zapisz, jakie problemy Twoim zdaniem rozwiąże budowane API.
5. Teraz przejrzyj wszystkie zebrane informacje. Podsumuj je w dwóch lub trzech zdaniach, które wyjaśniają przeznaczenie API.

Dzięki wykonaniu tego ćwiczenia dobrze zrozumiesz przeznaczenie danego API. Ostateczny rezultat może wyglądać mniej więcej tak:

„To API zostało opracowane, aby umożliwiało komunikację między naszą usługą a innymi usługami w firmie. Będzie ono głównie wykorzystywane przez innych programistów w naszej organizacji. To wewnętrzny interfejs API, dostępny wyłącznie w ramach naszej wewnętrznej sieci firmowej.

Jego celem jest przekazywanie do innych części naszego systemu informacji o stanie określonych zdarzeń w usłudze. Wprawdzie co jakiś czas będzie sprawdzana część punktów końcowych, ale ogólnie rzecz biorąc, nie przewiduje się dużego obciążenia".

Zrozumienie przeznaczenia API ma kluczowe znaczenie. Jednak aby osiągnąć ten cel, API musi być użyteczne. W następnym podrozdziale wyjaśnię, co sprawia, że API jest łatwe w użyciu.

Tworzenie użytecznych API

Użyteczność polega na znalezieniu równowagi między udostępnieniem zbyt wielu kontrolek a udostępnieniem zbyt małej liczby. Osiągnięcie tego stanu jest niezwykle trudne. W skrajnych przypadkach łatwo zauważyć, że ta równowaga zostaje zaburzona. Weźmy na przykład API Metropolitan Museum of Art, które udostępnia informacje o różnych dziełach sztuki znajdujących się w zbiorach muzeum. Gdyby API oferowało tylko jedno wywołanie zwracające wszystkie dane, udostępniałoby zbyt mało kontrolek. Konieczność przetworzenia tak dużej ilości informacji sprawiłaby, że korzystanie z API straciłoby sens. Z drugiej strony, gdyby API udostępniało oddzielny punkt końcowy dla

każdego elementu metadanych w systemie, znalezienie właściwego punktu końcowego z poszukiwanymi informacjami byłoby bardzo trudne. Aby skorzystać z takiego systemu, trzeba byłoby przyswoić zbyt wiele informacji.

Jeśli chcesz osiągnąć odpowiednią równowagę, musisz dobrze przemyśleć rozwiązanie. Upewnij się, że budowane API dostarcza użytkownikom wystarczająco szczegółowych danych, które są im potrzebne do wykonywania zadań (tutaj przydaje się znajomość przeznaczenia API), ale nimi nie przytłacza. Innymi słowy, zachowaj jak największą prostotę.

Użyteczna struktura API

Jednym ze sposobów na utworzenie użytecznego API jest stosowanie jako nazw punktów końcowych wyłącznie *rzeczowników*. Jeśli chcesz, aby użytkownicy łatwo zrozumieli dane API, zbuduj je w zgodzie z obiektami istniejącymi w systemie. Na przykład, jeśli użytkownicy API mają mieć możliwość pobrania informacji o studentach w systemie edukacyjnym, nie definiuj punktu końcowego o nazwie `/getAllStudents` (pol. *pobierz wszystkich studentów*). Zamiast tego utwórz punkt końcowy `/students` i wywołuj go za pomocą metody `GET`. Jednym z powodów takiego podejścia jest łatwość dodawania nowych studentów. Gdy punkt końcowy nazywa się `/students`, możesz go łatwo wywołać z użyciem metody `POST`, aby w ten sposób dodać nowego studenta. Natomiast w przypadku punktu końcowego o nazwie `/getAllStudents` użycie wymienionej metody w celu dodawania nowych elementów byłoby bardzo nienaturalne. Zastanów się nad tym, wypowiadając to na głos. Zdanie „Chcę utworzyć nowy obiekt studenta" brzmi naturalnie, w przeciwieństwie do „Chcę utworzyć obiekt pobierania wszystkich studentów".

Tworzenie punktów końcowych o nazwach opartych na rzeczownikach pomoże również w bardziej naturalnym ustrukturyzowaniu danych. Na przykład, jeśli masz punkt końcowy `/students`, wówczas dla każdego studenta możesz łatwo dodać punkt końcowy, który będzie miał postać `/students/{studentId}`. Taka struktura kategoryzacji to kolejna reguła przydatna w projektowaniu API i dlatego warto o niej pamiętać.

Opracowanie tego rodzaju struktury odwzorowuje układ API na rodzaje informacji, które są potrzebne użytkownikowi danego API. Dzięki temu znacznie łatwiej będzie znaleźć odpowiednie informacje.

Taka struktura sprawdza się całkiem dobrze, ale czy rzeczywiście odpowiada temu, jak użytkownicy będą korzystać z API? Jeśli szukam informacji o studencie, czy będę znał jego identyfikator w API? Być może, ale bardziej prawdopodobne jest to, że będę znał np. jego imię (ang. *name*). Czy w takim razie powinniśmy zmodyfikować strukturę przez dodanie kolejnego punktu końcowego, np. `/students/name`? A co, jeśli chcemy znaleźć wszystkich studentów w określonym wieku (ang. *age*)? Czy powinniśmy dodać jeszcze jeden punkt końcowy, `/students/age`? Doskonale widzisz, do czego zmierzam — sytuacja może szybko stać się skomplikowana.

W tym miejscu przydatne okazują się **parametry zapytania**. Taki parametr określa sposób, w jaki można uzyskać podzbiór kategorii na podstawie jej właściwości. Zamiast tworzyć osobne punkty końcowe `name` i `age` w kategorii `students`, jak w poprzednich

przykładach, możemy po prostu użyć parametrów zapytania. W takim przypadku wywołanie mogłoby mieć postać `/students?name='JanKowalski'` lub `/students?age=25`. Parametry zapytania pomagają zachować prostotę i logikę punktów końcowych, a zarazem zapewniają użytkownikom elastyczność w efektywnym pobieraniu interesujących ich informacji.

Dobre komunikaty o błędach

Dobrze zaprojektowane API powinno pomagać użytkownikom, gdy popełniają błędy. Dlatego też w odpowiedziach udzielanych za żądania należy zwracać odpowiednie **kody stanu HTTP**. Jeśli żądanie jest źle sformatowane, API powinno zwrócić kod błędu 400. Nie będę tu wymieniać wszystkich kodów stanu HTTP, ponieważ łatwo je znaleźć w internecie. Ważnym aspektem jakości jest to, aby API zwracało kody adekwatne do rodzaju udzielanej odpowiedzi. To istotne kryterium oceny jakości API.

Oprócz kodów stanu HTTP niektóre API mogą zwracać również komunikaty informujące o możliwych krokach, jakie można podjąć w celu rozwiązania problemu. Z jednej strony taka możliwość może okazać się niezwykle pomocna, choć z drugiej należy uważać, aby nie ujawnić zbyt wielu informacji osobom, które mogłyby je wykorzystać w niecnych celach.

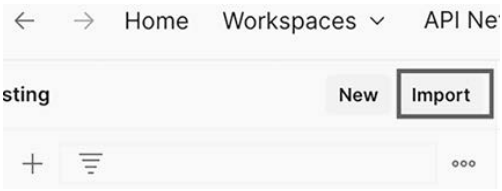
Dokumentowanie API (za pomocą Postmana)

Często pomijanym, lecz niezwykle istotnym aspektem dobrej jakości API jest jego odpowiednie dokumentowanie. Czasami testerzy i programiści mogą traktować dokumentację jako coś spoza ich obszaru kompetencji. Jeśli masz zespół, który przygotowuje dokumentację do API, to oczywiście dobrze jest pozwolić mu się tym zająć, przy czym nie traktuj dokumentacji jako czegoś mniej ważnego. Dokumentacja jest często pierwszym elementem, z którym zetknie się użytkownik Twojego API. Zatem jeśli nie wskaże mu właściwego kierunku, może poczuć się zdezorientowany i sfrustrowany. Niezależnie od tego, jak dobrze zaprojektowano interfejs programowania aplikacji, użytkownicy będą potrzebowali pewnej formy dokumentacji, która pomoże im zrozumieć, co on potrafi i jak z niego korzystać.

Tworzenie dokumentacji w Postmanie

Postman umożliwia tworzenie dokumentacji bezpośrednio w programie. W poprzednim rozdziale pokazałem, jak utworzyć żądanie do API GitHuba. Oczywiście GitHub ma własną dokumentację API, ale w tym miejscu na podstawie tego żądania wyjaśnię, jak można opracować ją w Postmanie. Jeśli wcześniej nie utworzyłeś kolekcji przeznaczonej do eksperymentów z GitHubem, możesz ją zaimportować z repozytorium GitHuba, które zawiera przykładowe materiały do tego rozdziału.

1. Ze strony *https://github.com/PacktPublishing/API-Testing-and-Development-with-Postman-Second-Edition/tree/main/Chapter02* pobierz plik kolekcji *GitHub API Requests.postman_collection.json*.
2. W programie Postman kliknij przycisk *Import* widoczny na górze okna, jak pokazałem na rysunku 2.1.



Rysunek 2.1. Przycisk Import w oknie Postmana

3. W wyświetlonym oknie dialogowym kliknij przycisk *Choose Files* (pol. *wybierz pliki*), przejdź do katalogu zawierającego pobrany plik kolekcji i wskaż go.
4. Kliknij przycisk *Import*, a Postman automatycznie zaimportuje kolekcję.

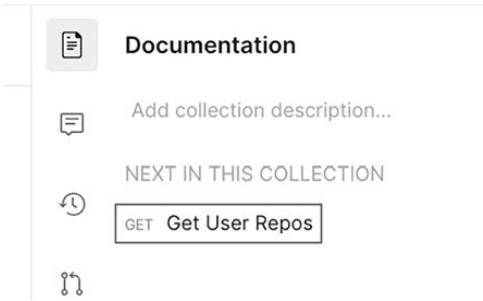
Po skonfigurowaniu kolekcji możesz utworzyć dokumentację przez wykonanie następujących kroków:

1. Przejdź do kolekcji `GitHub API Requests` i ją kliknij.
2. Teraz musisz kliknąć łącze *View complete documentation* (pol. *wyświetl pełną dokumentację*).

To spowoduje wyświetlenie panelu *Documentation*.

Jeśli chcesz, to w górnej części okna możesz wprowadzić dokumentację dla całej kolekcji. Wprawdzie czasami może to być przydatne, ale w tym przypadku mamy tylko jedno żądanie, więc prawdopodobnie nie będzie tutaj zbyt wielu aspektów do opisania. Zamiast tego możesz przejść na dół okna i wprowadzić dokumentację jedynie dla żądania **`Get User Repos`** (pol. *pobierz repozytoria użytkownika*).

Zauważ, że aby przejść do żądania, możesz również kliknąć je w panelu widocznym po prawej stronie okna, jak pokazałem na rysunku 2.2.



Rysunek 2.2. Przejście do dokumentacji żądania

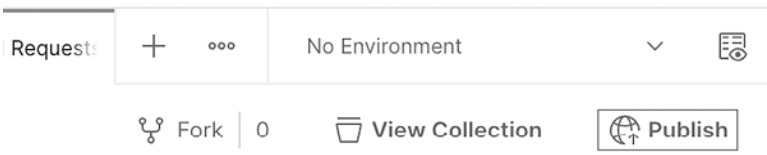
W omawianym przypadku kolekcja składa się z tylko jednego żądania, więc żeby je odszukać, wystarczy przewinąć zawartość okna. Jednak gdy masz wiele żądań, szybkie przejście do interesującego Cię elementu umożliwia nawigacja skokowa. Aby edytować dokumentację, po prostu kliknij obszar poniżej adresu URL żądania, a następnie możesz zacząć dodawać opis bezpośrednio w wyświetlonym polu tekstowym.

Uwaga

W opisie API w Postmanie można stosować składnię Markdown. Jeśli jej nie znasz, to wyjaśniam, że ma ona postać prostego zestawu reguł służących do formatowania tekstu. Doskonałym zasobem informacji o dostępnych poleceniach Markdown jest ściągawka, którą znajdziesz pod adresem *https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet*.

Przystępujemy do utworzenia dokumentacji dla tego punktu końcowego.

1. Na tym etapie wpisz po prostu coś prostego, aby wypróbować sposób tworzenia dokumentacji. Na przykład możesz wprowadzić zdanie typu `Ten punkt końcowy pobiera informacje o repozytoriach danego użytkownika`.
2. Gdy klikniesz gdzieś poza panelem edycji, Postman automatycznie zapisze wprowadzone zmiany. Jednak tak utworzona dokumentacja jest obecnie dostępna tylko w danej kolekcji. Możesz ją upublicznić przez jej opublikowanie.
3. Aby opublikować dokumentację, kliknij w górnej części okna ikonę *Publish*, jak pokazałem na rysunku 2.3.



Rysunek 2.3. Opublikowanie dokumentacji

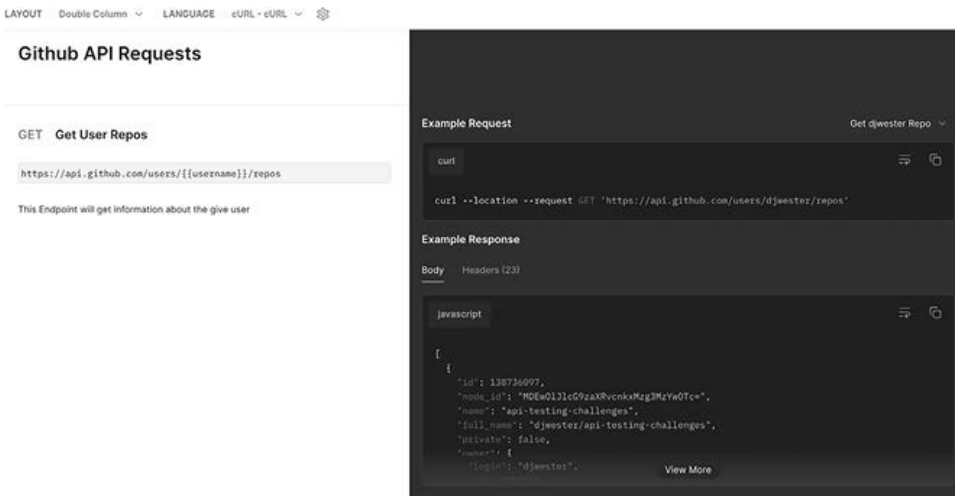
4. To spowoduje wyświetlenie strony internetowej, na której można wybrać różne opcje związane z nadaniem stylu dla utworzonej dokumentacji. Po skonfigurowaniu ustawień na tej stronie przewiń ją w dół i kliknij przycisk *Publish Collection* (pol. *opublikuj kolekcję*), aby udostępnić tę dokumentację innym użytkownikom.
5. Na opublikowanej stronie znajdziesz adres URL, pod którym będzie dostępna utworzona przed chwilą dokumentacja. Kliknij to łącze, aby wyświetlić przejrzystą stronę dokumentacji, którą możesz udostępnić innym.

Przeglądając tę dokumentację, można zauważyć, że jest ona dość skromna. Jedną z możliwości jej usprawnienia jest dodanie przykładowego żądania wraz z udzielaną na nie odpowiedzią. Dodawanie przykładów w programie Postman jest bardzo łatwe i sprowadza się do wykonania wymienionych tutaj kroków.

1. Wróć do Postmana i przejdź do żądania `Get User Repos`.
2. Zmienną {{*username*}} w adresie URL zastąp poprawną nazwą użytkownika (możesz użyć własnej lub wpisać moją, czyli `djwester`). W kolejnych rozdziałach wyjaśnię sposób pracy ze zmiennymi, natomiast w tym momencie należy po prostu ręcznie podmienić tę wartość.
3. Kliknij przycisk *Send*, aby wykonać żądanie. Jeśli w odpowiedzi otrzymasz pustą tablicę, może to oznaczać, że Twoje ustawienia prywatności w serwisie GitHub uniemożliwiają zwrócenie oczekiwanych wyników. W takim przypadku użyj mojej nazwy użytkownika (`djwester`).
4. Teraz przejdź do odpowiedzi i kliknij przycisk *Save as example* (pol. *zapisz jako przykład*), jak pokazałem na rysunku 2.4.

Rysunek 2.4. Dodawanie przykładu w programie Postman

5. Postman automatycznie dołączy przykład do danego żądania. Wróć na stronę dokumentacji API i odśwież ją. Zobaczysz teraz przykładowe żądanie i udzieloną na nie odpowiedź, jak pokazałem na rysunku 2.5.



Rysunek 2.5. Przykład umieszczenia w dokumentacji API żądania i udzielonej odpowiedzi

Warto poświęcić więcej czasu na poznawanie sposobów dobrego dokumentowania API. Funkcjonalność oferowana przez Postmana sprawia, że jest to dość proste zadanie.

Niezależnie od tego, czy jesteś testerem czy programistą i czy pracujesz nad nowym API czy nad istniejącym już od jakiegoś czasu, zachęcam Cię do pewnego wysiłku mającego na celu odpowiednie udokumentowanie API. Dobra dokumentacja znacząco ułatwi pracę użytkownikom, a także pomoże Tobie i Twojemu zespołowi lepiej zrozumieć sposób działania danego API. Warto więc poświęcić na to czas, zwłaszcza gdy korzysta się z narzędzia takiego jak Postman, które bardzo ułatwia tworzenie dokumentacji API. Wprawdzie to nie jest książka o dokumentacji API, ale mimo to warto poświęcić kilka minut na omówienie wybranych kwestii związanych z tworzeniem dobrej dokumentacji API.

Dobre praktyki w zakresie dokumentowania API

Jedną z kluczowych reguł, o której należy pamiętać podczas tworzenia dokumentacji API, jest **spójność**. W tym i poprzednim rozdziale przedstawiłem wiele terminów związanych z API. Zawsze będą występować pewne różnice w stosowaniu tych pojęć, ale staraj się używać ich w sposób zgodny z tym, jak ma to miejsce w przypadku innych API. Należy również dążyć do osiągnięcia jak największej wewnętrznej spójności w dokumentacji. Oznacza to między innymi konsekwentne stosowanie tych samych terminów na określenie tych samych rzeczy w całej dokumentacji. Ważne jest także zachowanie spójności struktury API, co pozwoli uniknąć konieczności wyjaśniania podobnych kwestii na różne sposoby w poszczególnych sekcjach dokumentacji.

Utworzenie dokumentacji API może przynieść korzyści nie tylko czytającym, ale także Tobie. Czasami w trakcie tego zadania pewne rzeczy same rzucają się w oczy. Możesz zauważyć, że dwie podobne funkcje są opisane w zupełnie odmienny sposób. Dzięki temu dostrzegasz, gdzie należy coś poprawić lub zmienić w samym API, co stanowi kolejną zaletę dobrej dokumentacji API.

Kolejną bardzo ważną kwestią, o której należy pamiętać podczas tworzenia dokumentacji API, jest znaczenie przykładów. Wcześniej pokazałem, jak łatwo w Postmanie tworzy się przykład w dokumentacji. Warto to wykorzystać. Mówi się, że obraz jest wart tysiąca słów. Uważam, że przykład w dokumentacji API jest wart prawie tyle samo. Niektóre kwestie trudno jest dobrze wytłumaczyć jedynie słowami i dopiero zobaczenie czegoś w praktyce pozwala użytkownikom zrozumieć sposób, w jaki to działa. Nie należy lekceważyć siły dobrze przygotowanego przykładu.

Ostatnia rzecz, o której warto pamiętać podczas tworzenia dokumentacji API, to problem dotyczący wszystkich rodzajów dokumentacji — dezaktualizacja. Kod się zmienia, a to dotyczy również kodu interfejsu programowania aplikacji. Z czasem funkcjonalność i sposób działania API będą ewoluować. Jeśli nie będziesz na bieżąco aktualizować dokumentacji, aby odzwierciedlała te zmiany, szybko przestanie ona być przydatna.

Wcześniej pokazałem, jak można utrzymywać dokumentację razem z testami oraz jak generować przykłady bezpośrednio na podstawie żądań w Postmanie. Wykorzystaj te możliwości i zadbaj o to, aby dokumentacja Twojego API zawsze była aktualna. Dużą część pracy Postman wykona za Ciebie, automatycznie aktualizując całą dokumentację przy każdej publikacji.

Istnieją narzędzia do pracy ze specyfikacją, które mogą pomóc w automatycznym generowaniu dokumentacji. Tego rodzaju narzędzia pomagają utrzymać dokumentację i testy na bieżąco z najnowszymi zmianami wprowadzanymi w kodzie. W rozdziale 4. wybrane z nich omówię bardziej szczegółowo. Skoncentruję się przy tym szczególnie na specyfikacji OpenAPI, która jest potężnym i popularnym narzędziem przeznaczonym do tworzenia specyfikacji API. Oczywiście istnieją też inne narzędzia tego typu.

W społeczności zajmującej się tworzeniem API toczy się dyskusja, która odzwierciedla szersze debaty w całym środowisku programistycznym. Spór sprowadza się do tego, ile czasu należy poświęcić na wstępne projektowanie. Niektórzy twierdzą, że skoro jedynym dobrym oprogramowaniem jest to, które pomaga użytkownikom rozwiązać problem, należy stosować podejście typu „najpierw wydaj", pozwalające szybko wprowadzić w życie własne pomysły. Następnie uzupełniamy projekt na podstawie rzeczywistych potrzeb i oczekiwań klientów. Z kolei inni opowiadają się za podejściem typu „najpierw zaprojektuj", w którym rygorystycznie definiuje się sposób działania API, jeszcze zanim przystąpi się do utworzenia jakiegokolwiek kodu. Jak zwykle w takich przypadkach, najlepiej jest unikać skrajności i znaleźć złoty środek między tymi dwoma podejściami.

Nowoczesne narzędzia mogą nam w tym pomóc. Na przykład istnieją narzędzia umożliwiające tworzenie prostych projektów, a następnie wykorzystywanie ich do zbierania opinii od klientów. Jednym z narzędzi przydatnych w trakcie projektowania API są tzw. **języki specyfikacji**. To po prostu zbiory reguł, które można wykorzystać do określenia sposobu działania interfejsów API. Przyglądając się metodom projektowania API, przyjrzyjmy się językowi specyfikacji, który ma na celu przeniesienie uwagi z samego dokumentowania interfejsów API na wspomaganie ich projektowania.

Język modelowania API typu RESTful

RAML, czyli ***RESTful API Modeling Language***, to język specyfikacji API, który — jak sama nazwa wskazuje — pomaga w modelowaniu API. Więcej informacji na jego temat znajdziesz w witrynie internetowej *https://raml.org/*. RAML oferuje narzędzia wspomagające projektowanie API, ale ich szczegółowe omówienie wykracza poza zakres tematyczny tej książki. W tym miejscu chcę jedynie wspomnieć o istnieniu tej specyfikacji i pokazać, jak możesz jej użyć do opracowania API spełniającego kryteria projektowe, o których mówiłem w tym rozdziale.

Rozpoczęcie pracy z językiem RAML jest bardzo proste — sprowadza się do uruchomienia edytora tekstu i wpisania kilku wierszy kodu. RAML zaprojektowano w taki sposób, aby pozostał czytelny dla człowieka. Dlatego specyfikacja jest napisana w prostym formacie tekstowym. RAML ma również strukturę hierarchiczną, co ułatwia tworzenie użytecznych struktur API, o których wcześniej wspominałem. W następnym podrozdziale przedstawię przykład użycia tego języka modelowania w pracy nad API i wykorzystania jego możliwości w trakcie pracy nad projektem w Postmanie. Następnie będziesz mieć okazję samodzielnego wypróbowania tej techniki.

Przykład projektowania API (ćwiczenie praktyczne)

Przedstawiłem już sporo teorii dotyczącej projektowania API, więc teraz chciałbym wyjaśnić, jak można do tego celu wykorzystać Postmana. Omówione tutaj reguły nie dotyczą tylko nowo tworzonych interfejsów API. W rzeczywistości stosowanie tych reguł także podczas testowania istniejącego API to świetny sposób na wykrycie potencjalnych zagrożeń dla jego wartości. Jednak żebyś mógł lepiej zrozumieć to zagadnienie, pokażę, jak zaprojektować interfejs API od początku. Jeśli zrozumiesz te reguły na takim przykładzie, poradzisz sobie z ich zastosowaniem również do istniejących interfejsów.

Studium przypadku — projektowanie API dla rozwiązania typu e-commerce

Wyobraźmy sobie, że chcemy zaprojektować API dla bardzo prostej aplikacji e-commerce. Ta aplikacja ma kilka produktów, które można przeglądać. Umożliwia również użytkownikom utworzenie profilu, z którego mogą korzystać podczas dodawania produktów do koszyka i dokonywania zakupów. Celem tego API jest udostępnienie danych w sposób, który mogą wykorzystać interfejsy użytkownika aplikacji zarówno internetowej, jak i mobilnej. Twój zespół właśnie otrzymał te informacje i musi opracować API, które spełni te wymagania.

Przejdźmy teraz przez ten proces i zobaczmy, jak zastosować omówione reguły projektowania. Zacznę od prostej definicji API w języku RAML. Pierwszym krokiem jest utworzenie pliku i określenie używanej wersji RAML. W tym celu w programie Visual Studio Code (możesz użyć dowolnego edytora tekstu) utworzyłem plik tekstowy o nazwie *E-Commerce_API-Design.raml*. Następnie na początku pliku dodałem odwołanie wskazujące, że będzie używana wersja 1.0 specyfikacji RAML.

```
#%RAML 1.0
---
```

Następnym krokiem jest nadanie tytułu dla API oraz skonfigurowanie podstawowego adresu URI. W tym celu dodałem do pliku następujące definicje.

```
title: E-Commerce API
baseUri: https://api.ecommerce.com/{version}
version: v1
```

Definiowanie punktów końcowych

To jest fikcyjne API, więc podstawowy adres URI nie wskazuje istniejącej witryny internetowej. Zwróć również uwagę na sposób określenia wersji. Skoro zdefiniowałem podstawy API, mogę rozpocząć projektowanie jego faktycznej struktury i dostępnych poleceń. Muszę zacząć od przeznaczenia tego API, którym jest obsługa zarówno witryny internetowej, jak i aplikacji mobilnej. W tym studium przypadku nie będę zagłębiać

się w szczegóły, takie jak tworzenie person. Wiemy jednak, że z tego API będą korzystać programiści frontendu, aby umożliwić wyświetlanie treści użytkownikom. Po przemyśleniu możemy założyć, że będą oni potrzebować dostępu do informacji o produktach, które zostaną wyświetlone użytkownikom. Ponadto muszą mieć możliwość dostępu do danych konta użytkownika, aby umożliwić użytkownikom tworzenie kont lub ich modyfikowanie. Wreszcie programiści frontendu potrzebują funkcjonalności dodawania i usuwania przedmiotów z koszyka na zakupy.

Mając te informacje dotyczące przeznaczenia API, można zastanowić się nad jego użytecznością i strukturą. Programiści frontendu prawdopodobnie będą oczekiwali punktu końcowego `/products`, który będzie używany do wyświetlania informacji o produktach. Ponadto oczekują punktu końcowego `/users` przeznaczonego do odczytywania danych użytkowników i zarządzania tymi danymi oraz punktu końcowego `/carts`, który pozwoli dodawać i usuwać przedmioty z koszyka na zakupy.

Te punkty końcowe to niejedyny sposób, w jaki można zaprojektować to API. Na przykład można połączyć punkty końcowe `carts` i `users`. Każdy koszyk na zakupy musi należeć do użytkownika, więc można zdecydować, że taki koszyk będzie właściwością danego użytkownika. Właśnie dlatego, że istnieją różne sposoby projektowania API, musimy rozważyć kwestie takie jak przeznaczenie interfejsu programowania aplikacji. W omawianym przypadku wiemy, że proces będzie wymagał regularnego dodawania i usuwania przedmiotów z koszyka na zakupy.

Programiści będą zastanawiać się nad tym, co muszą zrobić w tych kategoriach. Zatem zmuszanie ich do korzystania z punktu końcowego `users` w celu zmiany zawartości koszyka na zakupy spowodowałoby zwracanie dodatkowych danych, które są niepotrzebne w tym kontekście, a także mogłoby wprowadzić pewne zamieszanie.

Po wybraniu punktów końcowych, które będą używane w budowanym API, należy je umieścić w pliku specyfikacji RAML. To po prostu kwestia zapisania ich do pliku i umieszczenia dwukropka na końcu nazwy każdego z nich.

```
/products:
/users:
/carts:
```

Definiowanie operacji

Oczywiście, musimy mieć możliwość wykorzystania tych punktów końcowych do pewnych działań. Jakie operacje, Twoim zdaniem, powinny być dostępne dla poszczególnych punktów końcowych? Zastanów się przez chwilę, jakie operacje byłyby przydatne w ich przypadku.

Moja początkowa koncepcja zakładała, że powinniśmy mieć następujące operacje dla poszczególnych punktów końcowych.

```
/products:
    get:
/users:
    get:
```

```
        post:
        put:
    /carts:
        get:
        post:
        put:
```

Zastanówmy się nad tym przez chwilę. Jeśli do pobierania informacji o koszykach na zakupy będzie służył tylko jeden punkt końcowy, `/carts`, wówczas będzie trzeba pobierać i aktualizować dane o wszystkich koszykach w systemie za każdym razem, gdy zajdzie potrzeba przeprowadzenia dowolnej operacji z którymkolwiek z nich. Powinienem się zatrzymać i lepiej przemyśleć to rozwiązanie. Nazwy punktów końcowych są tutaj w liczbie mnogiej i powinny reprezentować zbiory lub listy obiektów. Potrzebny jest jakiś sposób, który umożliwi pracę z pojedynczymi obiektami w poszczególnych kategoriach.

```
    /products:
        get:
        /{productId}:
            get:
    /users:
        post:
        get:
        /{username}:
            get:
            put:
    /carts:
        post:
        get:
        /{cartId}:
            get:
            put:
```

Zdefiniowałem tutaj **parametry URI**, które umożliwiają użytkownikom pobieranie informacji o konkretnym produkcie, użytkowniku lub koszyku na zakupy. Zauważ, że polecenia `POST` są przypisane do punktów końcowych kolekcji, ponieważ używanie metody `POST` w trakcie pracy z kolekcją spowoduje dodanie do niej nowego obiektu. Ponadto umożliwiam użytkownikom API pobieranie pełnej listy poszczególnych kolekcji, jeśli tego potrzebują.

Dodawanie parametrów zapytania

W poprzednim rozdziale omówiłem pojęcie parametrów zapytania. Patrząc na to z perspektywy użytkowników API, myślę, że warto byłoby zastosować parametr zapytania w punkcie końcowym koszyka na zakupy. Gdy użytkownik klika produkt i chce dodać go do koszyka, identyfikator produktu jest już znany na podstawie kliknięcia elementu. Jednak niedostępny może być identyfikator danego koszyka na zakupy. W takim przypadku konieczne byłoby przeszukanie kolekcji koszyków w celu znalezienia tego, który należy do bieżącego użytkownika. Można ułatwić to zadanie przez utworzenie parametru

zapytania. Ponownie kieruję się tutaj regułami projektowymi dotyczącymi użyteczności i przeznaczenia API, aby w ten sposób opracować dobry model działania tego API.

W kodzie RAML wystarczy utworzyć wpis parametru zapytania dla operacji, dla której chcemy dodać taki parametr.

```
/carts:
    post:
    get:
        queryParameter:
            username:
    /{cartId}:
        get:
        put:
```

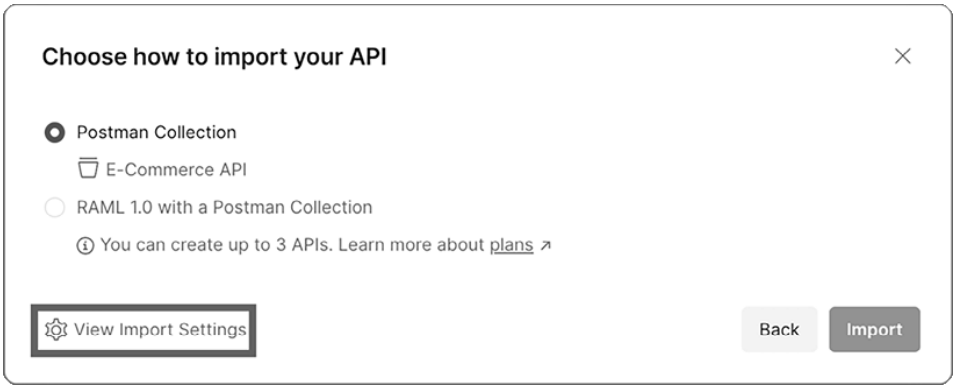
Strukturę API zaprojektowałem zgodnie z przedstawionymi wcześniej regułami. Mam nadzieję, że widzisz, jak potężne możliwości mają te reguły w zakresie zawężania szerokiego zakresu do czegoś znacznie łatwiejszego w zarządzaniu i zdecydowanie bardziej użytecznego. Podczas tworzenia specyfikacji RAML dla API nadal trzeba określić poszczególne atrybuty lub właściwości każdego z punktów końcowych i parametrów zapytań, które mają być utworzone. Nie zamierzam tutaj wchodzić we wszystkie związane z tym szczegóły. Jeżeli chcesz dowiedzieć się więcej na ten temat, skorzystaj z dostępnych samouczków RAML, które znajdziesz na stronie *https://raml.org/developers/raml-100-tutorial*. W naszym hipotetycznym API nie podałem wystarczająco dużo informacji, aby można było wypełnić wszystkie właściwości. Na przykład nie mamy dokładnych danych na temat poszczególnych produktów i użytkowników. W rzeczywistym rozwiązaniu tego rodzaju informacje prawdopodobnie byłyby pobierane z bazy danych, a następnie na ich podstawie powstałyby przykłady i atrybuty.

Korzystanie ze specyfikacji RAML w programie Postman

To może nie być pełna specyfikacja API, ale wystarczy nam do eksperymentów w Postmanie. Kliknij w Postmanie przycisk *Import* i wybierz utworzony wcześniej plik *.raml*. Przed zaimportowaniem zajrzyj do sekcji ustawień importu, do której masz dostęp za pomocą łącza *View Import Settings* (pol. *wyświetl ustawienia importowania*), jak pokazałem na rysunku 2.6.

Upewnij się, że opcje generowania parametrów żądania i odpowiedzi są ustawione na użycie schematu (*Schema*). Wróć do panelu importu i kliknij przycisk *Import*. Postman przeanalizuje specyfikację i automatycznie utworzy kolekcję zawierającą zdefiniowane wywołania dla każdego z punktów końcowych i wariantów operacji, które zostały zdefiniowane w pliku specyfikacji. Prawda, że to świetne rozwiązanie?

Gdy już poznasz więcej funkcji Postmana, pokażę Ci, jak wykorzystać te koncepcje do głębszego zanurzenia się w proces projektowania API. Na razie jednak zachęcam Cię, żebyś zastanowił się nad tym, jak projektowanie API wpływa na jego jakość. Mam zresztą dla Ciebie małe wyzwanie.



Rysunek 2.6. Łącze umożliwiające przejście do ustawień dotyczących importowania

Modelowanie istniejącego projektu API

Dotychczas wyjaśniłem, jak podchodzić do projektowania API. Przedstawiłem kilka reguł, a następnie pokazałem przykład ich praktycznego zastosowania. To studium przypadku dotyczyło projektowania od podstaw zupełnie nowego API. Jednak często będzie trzeba pracować z już istniejącym API. Na szczęście te same reguły projektowania mogą pomóc w ulepszeniu istniejącego API. Aby je lepiej przyswoić, warto samodzielnie wykonać zamieszczone tutaj ćwiczenie. Zatem spróbuj zastosować wspomniane reguły w API, nad którym obecnie pracujesz. Jeśli Twoja firma nie ma API, którego można użyć podczas wykonywania tego ćwiczenia, możesz oczywiście skorzystać z jakiegoś publicznie dostępnego API.

Wykorzystując wybrane API, wykonaj wymienione tutaj kroki, aby zastosować wcześniej omówione reguły projektowania API.

1. Do pliku RAML dodaj wszystkie punkty końcowe. Upewnij się, że struktura w tym pliku odzwierciedla hierarchię API.
2. Poświęć chwilę na przemyślenie przeznaczenia API i oceń, czy bieżąca struktura pozwala osiągnąć ten cel. Czy istnieje inny sposób, w jaki można zaprojektować to API? Czy można usprawnić jego układ, aby ułatwiał osiągnięcie wyznaczonego celu?
3. Jak sądzisz, które operacje i parametry zapytania należy przypisać poszczególnym punktom końcowym? Utwórz kopię pliku, a następnie uzupełnij ją swoimi pomysłami dotyczącymi funkcjonalności danego API.
4. W oryginalnym pliku dodaj rzeczywiste operacje i parametry zapytań, które ma API. Jak się one mają do tych, które zostały umieszczone w kopii pliku?

Jeśli chcesz, możesz zaimportować plik do Postmana. W miarę poznawania funkcjonalności tego programu, m.in. związanej z testowaniem, będziesz mieć już gotową kolekcję żądań do wykorzystania.

Podsumowanie

Projektowanie API wymaga starannego przemyślenia. API to oprogramowanie, a celem tworzenia oprogramowania jest pomoc użytkownikom w rozwiązywaniu problemów. Zadaniem API jest rozwiązywanie konkretnych problemów, a im lepiej sobie z tym radzą, tym większa jest ich jakość. Jednym z czynników wpływających na jakość API jest jego dobry projekt. Dobrze zaprojektowane API to takie, które skutecznie realizuje postawione mu zadania. W tym rozdziale wyjaśniłem, jak przeanalizować przeznaczenie API przez zdefiniowanie person. Przedstawiłem również pytania, które pomagają ustalić, dlaczego dane API w ogóle powinno istnieć.

W rozdziale przedstawiłem również kilka sposobów strukturyzowania i dokumentowania API. Omówiłem specyfikację interfejsu programowania aplikacji i pokazałem, jak używać języka RAML do tworzenia specyfikacji opartych na projekcie. Oczywiście poruszone zagadnienia można było wypróbować w praktyce. Po opanowaniu podstaw reguł dotyczących projektowania API możemy nieco bardziej zagłębić się w języki specyfikacji interfejsów API i sposoby ich wykorzystania podczas tworzenia wysokiej jakości API. Tym zajmiemy się w następnym rozdziale.

API? Lepiej przetestować dziś niż debugować jutro!

Znaczenie interfejsów API systematycznie rośnie. Głównie za sprawą tego, że ułatwiają komunikację — zarówno zewnętrzną, jak i tę, która zapewnia spójne działanie elementów nowoczesnych systemów. I podobnie jak dobre porozumienie jest podstawą relacji międzyludzkich, tak samo sprawna komunikacja między usługami ma kluczowe znaczenie dla prawidłowego funkcjonowania aplikacji. Z tego powodu od jakości API zależy jakość systemu oprogramowania jako całości.

Dzięki tej praktycznej książce poznasz pełnię możliwości Postmana. Znajdziesz tu przystępnie wyjaśnione koncepcje, a także zestaw rzeczywistych przykładów, co ułatwi Ci korzystanie z tego narzędzia do tworzenia doskonale zaprojektowanych, udokumentowanych i przetestowanych interfejsów programowania aplikacji. Za pomocą praktycznych projektów nauczysz się dodawać automatyzację testów do istniejącego API. Poznasz również nowe funkcjonalności Postmana, stanowiące dużą pomoc w unikaniu błędów. W drugim, w pełni zaktualizowanym wydaniu znajdziesz nowe rozdziały dotyczące testowania przepływu pracy, tworzenia i używania serwerów imitacji, testowania bezpieczeństwa API, jak również testowania wydajności.

W książce:

- użycie Postmana do poprawy jakości interfejsu API
- użycie sieci GAN do przeprowadzania ataków i generowania deepfake'ów
- serwery mock w Postmanie i testowanie kontraktowe
- zasady testowania bezpieczeństwa
- funkcjonalne i niefunkcjonalne podejście do testowania API
- praca ze standardami przemysłowymi, na przykład przy użyciu OpenAI i imitacji

Dave Westerveld jest doświadczonym testerem oprogramowania, który pomaga innym specjalistom nadążać za dynamicznym światem IT. Tworzy popularne kursy wideo, przemawia na konferencjach i dzieli się wiedzą w podcastach i rozmowach online. Przez lata pełnił różne funkcje w branży — od testera eksploracyjnego, przez programistę automatyzacji testów, po specjalistę do spraw integracji API.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-2478-9
9 788328 924789
89,00 zł
packt