O'REILLY®

Interfejs API
Strategia programisty

POZNAJ POTENCJAŁ INTERFEJSÓW API!

Daniel Jacobson,
Greg Brail, Dan Woods

Helion

Tytuł oryginału: APIs: A Strategy Guide

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-0555-7

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

Printed in Poland.

Spis treści

ROZDZIAŁ 5.

Kluczowe zasady projektowania interfejsów API

W początkowych etapach rozwoju projekt interfejsów API stanowi sztukę. Niemniej jednak zdobyto wystarczające doświadczenie do tego, aby opracować wartościową listę błędów do unikania, a także zasady zapewniające powodzenie. Żeby właściwie zaprojektować interfejs API, konieczne będzie uwzględnienie większości zagadnień poruszonych w tej książce. Na projekt mają wpływ przyjęta strategia oraz grupa odbiorców, jak również zasoby biznesowe planowane do udostępnienia. Oczywiście dla projektu znaczenie ma też wybrana metoda technologiczna. W tym rozdziale skoncentrowano się na następujących trzech obszarach:

- projektowanie interfejsu API dla różnych grup odbiorców;
- zapewnienie przeglądu kwestii technologicznych;
- wyróżnienie ogólnych zagadnień projektowych.

Strategia dotycząca interfejsów API skupia się głównie na utworzeniu co najmniej jednego kanału z interfejsem API w celu ułatwienia osiągnięcia przyjętych celów biznesowych. Gdy ludzie korzystają z interfejsu API, tak naprawdę nie zastanawiają się nad wartością jego samego. Zainteresowani są wartością, jaką uzyskają za pośrednictwem interfejsu, czyli zasobami biznesowymi (produktami i usługami) udostępnianymi za jego pomocą. Przy projektowaniu interfejsów API próbujemy udostępnić zasoby biznesowe projektantom, tak aby mogli tworzyć aplikacje zapewniające wartości użytkownikom końcowym, którzy następnie będą mieli możliwość uzyskania określonego rodzaju wartości pozwalającej zakończyć cykl korzyści związanych z interfejsem API.

Ogólnie rzecz biorąc, najlepsze interfejsy API są starannie zaprojektowane, łatwe do opanowania, z logiczną strukturą i wewnętrznie spójne. Dzięki temu projektanci mogą stwierdzić, jak ze względnym powodzeniem wyeliminować wszelkie niejasności. Interfejs API powinien być trochę jak samochód. Każde auto wyposażone jest w kierownicę oraz pedały hamulca i gazu. Możesz uznać, że światła awaryjne, otwieranie bagażnika lub radio są trochę inne w różnych modelach samochodów, ale do rzadkości należy sytuacja, w której doświadczony kierowca nie wie, jak prowadzić wypożyczony pojazd.

Projektowanie interfejsów API dla konkretnych grup odbiorców

Jak w przypadku dowolnej kampanii o charakterze biznesowym interfejs API wymaga zdefiniowania grupy odbiorców. Okazuje się, że w odniesieniu do interfejsów API należy rozważyć dwie podstawowe grupy.

Projektanci (lub nawet szerzej rzecz ujmując, zespoły techniczne i tworzące produkty) to *bezpośrednia grupa odbiorców*, która korzysta z interfejsu API i buduje na jego bazie aplikacje.

Użytkownicy końcowi to *pośrednia grupa odbiorców*, która stosuje aplikacje utworzone za pomocą interfejsu API.

Aby określić grupę odbiorców, należy się najpierw zastanowić nad sformułowaniem wizji dotyczącej interfejsu API. Pozwoli to stwierdzić, co jest celem interfejsu. W dalszej kolejności można ustalić, którzy odbiorcy spośród tych ich typów skorzystają z interfejsu API oraz jak musi wyglądać model interakcji. Czy interfejs jest projektowany dla działów, które bazują na informacjach z systemu ERP? Czy może interfejs ma być przeznaczony dla projektantów aplikacji dla urządzeń przenośnych, najważniejszych 10 partnerów firmy, czy sprzedawców stowarzyszonych z firmą?

Przeprowadź analizę segmentacji użytkowników końcowych i (lub) projektantów oraz ustal priorytety dla segmentów. Jeśli początkowa odpowiedź wskazuje na wszystkich jako odbiorców, sugerujemy trochę staranniejsze przemyślenie tej kwestii, przynajmniej w przypadku określania priorytetów.

Wyzwaniem jest projektowanie zarówno z myślą o projektantach tworzących interfejs API, jak i użytkownikach końcowych, którzy będą stosować interfejs API. Zajmiemy się teraz przybliżeniem obu zagadnień.

Projektowanie pod kątem projektantów

Projektantów można podzielić na wiele typów i podkategorii. Interfejs API może być tworzony z myślą o nielicznej, bardzo specyficznej grupie projektantów o wysokich kompetencjach używających konkretnej technologii lub dla każdego na świecie, komu firma chciałaby zapewnić dostęp do swoich zasobów biznesowych. W związku z tym warto zaznajomić się z segmentacją kategorii projektantów. Dzięki temu wybór platformy technicznej stanie się bardziej oczywisty.

Inaczej mówiąc, zespoły budujące interfejsy API będą zwykle zmuszone do dokonywania wielu wyborów dotyczących technologii. Protokół SOAP czy REST? Format XML czy JSON? Protokół OAuth czy coś innego? Odpowiedzi na te pytania będą bardziej niż cokolwiek innego zależne od odbiorców, którzy skorzystają z interfejsu API. Prezentujemy jednak kilka rekomendacji.

Nasze rekomendacje technologiczne dla interfejsów API

Jeśli nie ma żadnych konkretnych powodów, aby postąpić inaczej, oto pierwsze możliwości, jakie powinny zostać obecnie wybrane przez dowolny zespół tworzący interfejs API:

- „Pragmatyczna" usługa REST na potrzeby struktury, ponieważ ułatwia opanowanie, wykorzystanie i rozszerzenie interfejsu API w większym stopniu niż inne technologie, takie jak SOAP (w dalszej części książki zostanie omówiona usługa REST i jej „pragmatyczny" wariant).
- JSON jako format danych pobieranych i zwracanych przez interfejs API, gdyż upraszcza programistom generowanie danych i korzystanie z nich.
- Protokół OAuth do zabezpieczania, ponieważ zapobiega propagacji haseł w internecie, a jednocześnie obsługuje różne metody uwierzytelniania użytkownika końcowego.

Jeśli te technologie nie są Ci jeszcze znane, nie przejmuj się. W dalszej części rozdziału zajmiemy się usługą REST i formatem JSON, a protokół OAuth wraz z innymi zagadnieniami związanymi z zabezpieczeniami zostanie omówiony w rozdziale 6.

Co więcej, opisujemy te technologie jako pierwszą opcję wyboru, a nie jako jedyną. Nowoczesne serwery aplikacji i platformy z interfejsem API umożliwiają obsługiwanie więcej niż jednej takiej technologii. Na przykład

zapewniają one interfejs API, który wspiera technologie XML i JSON, albo udostępniają interfejsy API protokołu SOAP i usługi REST.

Najlepsze procedury techniczne związane z interfejsem API firmy Tumblr

Czy chciałbyś się podzielić jakimikolwiek najlepszymi procedurami technicznymi związanymi z interfejsem API firmy Tumblr?

Zależy nam na tym, aby interfejs API był łatwy do opanowania bez użycia dokumentacji. Dzięki temu, jak zostały zaprojektowane stosowane przez nas identyfikatory URI, powinno być po prostu możliwe przejście do poziomu wiersza poleceń.

Staramy się tylko w nieznacznym stopniu modyfikować interfejs API. Jedną z wielkich ironii losu związanych z witryną internetową jest to, że mogę wprowadzić w niej zmiany w dowolnym żądanym momencie, a z kolei interfejs API jest wyjątkowo delikatny w tym sensie, że może spowodować, iż przestaną działać aplikacje projektantów korzystających z tego interfejsu. Kontrola wersji to trwający proces.

Derek Gottfrid, dyrektor ds. produktów w firmie Tumblr

Choć warto zacząć od technologii REST, JSON i OAuth przy rozważaniu zastosowania interfejsu API, są powody, aby nie decydować się na te rozwiązania.

Usługa REST może powodować mniej elastyczne interakcje między klientami a interfejsem API. Mogą być dostępne lepsze metody osiągnięcia podobnego wyniku, bo dające większą efektywność.

Prostszy model obiektowy formatu JSON jest preferowany w wielu sytuacjach w celu zmniejszenia liczby bajtów, co poprawia wydajność dostarczania bajtów za pomocą protokołu HTTP. Język XML stwarza potencjał użycia bardziej rozbudowanych znaczników i opcji semantycznych. Ponadto w przypadku formatu XML istnieje znacznie więcej standardów organizowania treści niż dla formatu JSON. Oznacza to, że aplikacje analizujące oraz inne biblioteki kodu mogą zwiększyć efektywność projektowania aplikacji.

Choć protokół OAuth znakomicie nadaje się do zabezpieczania transakcji interfejsu API, najlepiej dostosowany jest do potrzeb dużych grup nieznanych projektantów, którzy próbują użyć interfejsu API firmy. Jeśli odbiorcy docelowi to niewielka grupa projektantów firmy, protokół OAuth może okazać się przesadą.

Kluczem do podjęcia takich decyzji jest zrozumienie tego, kto będzie korzystać z interfejsu API, a także tego, w jaki sposób takie osoby będą tworzyć aplikacje interesujące dla użytkowników końcowych.

Projektowanie pod kątem użytkowników aplikacji

Im lepiej możesz zrozumieć, jaki typ dostępu jest wymagany przez użytkowników końcowych, tym więcej informacji uzyskasz na temat tego, co ma zostać uwzględnione w interfejsie API, oraz tego, jak w największym stopniu ułatwić dostęp. Firmy publikujące interfejsy API powinny się zastanowić, czego użytkownicy oczekują od ich zasobów biznesowych i jak mogą zapewnić dostęp, który spowoduje zwiększenie wykorzystania interfejsu API.

Trudno w tym przypadku o generalizowanie, ponieważ często nie jest oczywiste, jakiego rodzaju aplikacje będą tworzone. Poza tym możesz tylko w zarysie wyobrażać sobie skrywane potrzeby użytkowników końcowych dotyczące zasobów biznesowych. Co więcej, jeśli po udostępnieniu interfejsy API odniosą sukces, zwykle będą stanowić inspirację dla nowych pomysłów i zastosowań, które być może nie były rozważane w fazie projektowania. Dotyczy to zarówno prywatnych, jak i publicznych interfejsów API.

Na przykład popularne funkcje, takie jak pobieranie z serwisu Twitter wszystkich wpisów jednej osoby (interfejs API firmy Twitter) lub tworzenie mapy z naniesionym na niej położeniem (interfejs API serwisu Google Maps), mogą być wykorzystywane z dużą łatwością, ponieważ wiele aplikacji oferuje realizowanie tych prostych działań. Inne, bardziej złożone funkcje zostały dodane później bądź okazały się przydatne w wyniku ciągłego powiększania się bazy użytkowników. Końcowy wniosek jest taki, że im więcej uzyskasz informacji o populacji użytkowników końcowych i rodzaju wymaganych przez nich aplikacji, tym lepiej będziesz poinformowany na etapie decydowania o tym, jakie funkcje mają zostać zaoferowane jako pierwsze.

Najlepsze praktyki związane z projektem interfejsów API

Z doświadczenia wiemy, że w przypadku interfejsów API część rzeczy się udaje, a część nie. W tym podrozdziale przedstawiamy kilka propozycji, które uważamy za solidne i najlepsze praktyki związane z projektem interfejsów API.

Organizacja NPR zna swoich odbiorców

Organizacja NPR opracowała swój program związany z interfejsem API, bazując na dobrej znajomości odbiorców docelowych. Zasadniczo z dalekowzroczności organizacji korzystają następujące cztery grupy:

- *Pracownicy* organizacji NPR to największa grupa używająca interfejsu API, która obsługuje całą infrastrukturę serwisu NPR.org, główną witrynę internetową, a także inne zasoby cyfrowe. Początkowo interfejs był stosowany do usprawniania systemu zarządzania treścią CMS wspierającego witrynę internetową. Miało to na celu umożliwienie użycia niestandardowych kanałów informacyjnych opracowywanych przez dział redakcyjny i projektowy. Największą korzyścią wynikającą z zastosowania interfejsu API było utworzenie wielu aplikacji firmowanych przez organizację NPR dla urządzeń przenośnych i innych platform. W efekcie po upływie niecałego roku osiągnięto 100-procentowy wzrost liczby wyświetleń stron dla wszystkich zasobów cyfrowych.
- *Stacje członkowskie organizacji NPR* stanowią istotne grono odbiorców. Używają one interfejsu API do pobierania treści i tworzenia dla swoich społeczności stron zespolonych z treścią o zasięgu lokalnym i ogólnokrajowym.
- Interfejs API organizacji NPR stworzył nowe możliwości dla *partnerów*, ponieważ wiązały się z nim niskie koszty integracji. Oznaczało to, że już istniejące rozwiązania również były łatwiejsze do utrzymania i rozwijania, gdyż kanał komunikacji był bardziej niezawodny niż wcześniej, kiedy rozwiązania często były obsługiwane za pomocą niestandardowych kanałów informacyjnych XML, kanałów RSS lub innych mniej niż optymalnych metod.
- Czwarta grupa korzystająca z interfejsu API organizacji NPR to *ogół społeczeństwa*. Rozszerzenie interfejsu ułatwiło wsparcie misji organizacji NPR, która oferuje usługę publiczną, a ponadto zapewniło zestaw widgetów i narzędzi wspomagających tworzenie zewnętrznych witryn internetowych wykorzystujących treść tej organizacji w innowacyjny sposób.

Odróżnij własny interfejs API

Podstawowa sprawa: dlaczego projektant powinien użyć Twojego interfejsu API? Czym się on różni od innych rozwiązań? Dlaczego należałoby z niego skorzystać?

Oto kilka możliwych sposobów odróżnienia własnego interfejsu API:

- Dane są unikalne, bardziej kompletne lub dokładniejsze niż w przypadku interfejsu konkurencji.
- Oferowane jest lepsze wsparcie bądź łatwiejszy proces rejestrowania.
- Interfejs API jest solidniejszy, bardziej niezawodny, ciekawszy albo szybszy niż rozwiązanie alternatywne.
- Zaproponowane warunki są atrakcyjniejsze dla projektantów. Być może oferujesz większy darmowy ruch sieciowy związany z danymi lub lepsze stawki.

Zaakcentuj to, dlaczego ktoś powinien używać Twojego interfejsu API, a nie innego rozwiązania. Może nim być publiczny interfejs API konkurencji, alternatywne źródło danych lub starsza (i bardziej znajoma) metoda powiązana z prywatnym interfejsem API. Możliwe jest też rozróżnienie ofert i elementów motywacyjnych związanych z Twoim interfejsem API. Jedna ze strategii polega na zaoferowaniu różnych wariantów cenowych zależnych od skali bądź typu wykorzystania. Taka strategia umożliwia pozyskanie większej liczby poziomów subskrybentów interfejsu. Część wywołań interfejsu API może być bezpłatna, a część może wymagać dokonania płatności. Możesz zezwolić na darmowe użycie interfejsu w przypadku publicznej aplikacji, natomiast pobierać opłatę, gdy wykorzystywany jest on wewnątrz firmy.

Zapewnij łatwość testowania i używania interfejsu API

Jeśli Twój interfejs API jest jedną z wielu opcji, jego potencjalni użytkownicy mogą poświęcić na jego wypróbowanie tylko kilka minut. Jeżeli nie mogą stwierdzić prawie natychmiastowych postępów, zainteresują się czymś innym. W przypadku udanych programów związanych z interfejsami API eliminowane są wszystkie bariery odnoszące się do użytkowania. Istnieje kilka metod, które to umożliwiają.

W odniesieniu do prywatnych interfejsów API ważne jest zademonstrowanie prawdziwej wartości wynikającej z uruchomienia systemu lub aplikacji na bazie interfejsu API. Na przykład interfejs może oferować znacznie bardziej elastyczny dostęp do treści, lepszą ogólną wydajność, efektywniejsze cykle projektowania albo inną korzyść. Aby przyspieszyć proces adaptacji interfejsu API, zidentyfikuj konkretny przypadek użycia znaczącej aplikacji i zadbaj o uwzględnienie go w środowisku produkcyjnym. Gdy to nastąpi, możliwe będzie dostrojenie tego przypadku zastosowania i zaprezentowania

jego wartości. Po wdrożeniu możliwego do zademonstrowania przypadku użycia łatwiejsze będzie wskazanie go jako przykładu, który może zostać wykorzystany przez innych. Ponadto projektanci, którzy zaimplementowali ten pierwszy przypadek użycia, ułatwią proces promowania, pod warunkiem że ich doświadczenie z nim związane jest pozytywne. Nie zapomnij postarać się o poparcie odpowiednich osób z zarządu i (lub) zespołów zarządzających, aby usprawnić sobie dalsze działania.

W przypadku publicznych interfejsów API jedną z metod postępowania jest oferowanie określonego poziomu darmowego dostępu. Wielu projektantów nawet nie rozważy użycia interfejsu, jeśli będzie on zapewniać tylko płatne opcje. Na tym etapie cyklu pozyskiwania klientów projektant może nie wiedzieć, czy Twój interfejs spełnia jego wymagania lub czy model biznesowy jego aplikacji będzie uwzględniać płatny dostęp.

Programy, które odniosły sukces, sprawiają też, że testowanie interfejsu API jest wyjątkowo proste i szybkie. Gdy to tylko możliwe, zapewniają one natychmiastową satysfakcję. Projektantom wyświetlającym interfejs API serwisu Twitter prezentowana jest konsola, która pozwala na błyskawiczne testowanie. Firma Twitter idzie o krok dalej, oferując operacje interfejsu, które nie wymagają żadnego rejestrowania ani uwierzytelniania (jest to na przykład operacja powiązana z publiczną osią czasu, umożliwiająca projektantom uzyskanie dostępu do ostatnich aktualizacji statusu dla wszystkich publicznych użytkowników na całym świecie).

Jeśli warunkiem zastosowania interfejsu API jest rejestracja, udoskonal proces. Nie jest dobrym pomysłem zadawanie więcej niż kilku pytań kwalifikujących lub implementowanie procesu zatwierdzania, w ramach którego musisz skontaktować się z projektantem. Zanim to nastąpi, projektant może już stracić zainteresowanie interfejsem API.

Inaczej sytuacja wygląda w przypadku prywatnych interfejsów API, dla których niezbędna jest oficjalna umowa partnerska. Im więcej informacji możesz zapewnić w celu zachęcenia partnerów do zarejestrowania, tym lepiej.

Zapewnij łatwość zrozumienia interfejsu API

Najbardziej udane interfejsy API są projektowane intuicyjnie. Prostota jest kluczem, pewnikiem dotyczącym nie tylko tego, na co interfejs pozwala, ale też sposobów prezentowania funkcji projektantom. Znakomitym przykładem jest interfejs API usługi Facebook Graph. Opis tego interfejsu czyta się jak książkę. Aby zrozumieć, jakie są jego możliwości, niezbędna jest bardzo ograniczona dokumentacja.

Stare powiedzenie dotyczące projektów, które odnosi się też do interfejsów API, brzmi: „Spraw, aby interfejs był tak prosty, jak to możliwe, lecz nie zbyt prosty". Twórcy publikujący interfejsy API często dołączają za wiele funkcji lub takie funkcje, które tak naprawdę są nieodpowiednie. Na przykład mieliśmy do czynienia z publicznymi interfejsami API zawierającymi funkcje, które przeznaczone były wyłącznie do użytku wewnętrznego (takie jak tabele kodów wewnętrznych).

Kin Lane, promotor projektantów w firmie Mimeo, twierdzi, że mniejsza liczba funkcji będzie korzystna dla projektu interfejsu API. „Moja rada jest następująca: prostsze znaczy lepsze. Usługa REST, format JSON czy inne proste i zrozumiałe usługi. Nie próbujcie zbytnio komplikować interfejsu. Skoncentrujcie się na podstawowych komponentach tworzących interfejs API. Zróbcie jedno, ale naprawdę dobrze, a ponadto dołączcie do tego prostą dokumentację i przykładowe kody. Są to fundamenty interfejsu API".

Oprócz oferowania najmniejszego, ale jednocześnie w miarę możliwości najbardziej rozbudowanego zestawu operacji kluczowe znaczenie ma też określenie struktury interfejsu API w przystępny sposób. Jest to ogromna zaleta interfejsów API, które zgodne są z opisanymi dalej wzorcami „pragmatycznej" usługi REST. Tego rodzaju interfejsy API są łatwe do opanowania, ponieważ prezentują swoje operacje w postaci zrozumiałych dla człowieka identyfikatorów URI, które mogą być testowane za pomocą prostych narzędzi, takich jak przeglądarka internetowa.

Może się wydawać, że podczas tworzenia interfejsu API nieistotne jest to, aby cechował się on intuicyjnością, łatwością obsługi i elegancją. Jeśli interfejs API zapewnia odpowiednią funkcjonalność, to czy projektanci i tak nie będą z niej korzystać? Może to mieć sens do momentu pojawienia się konkurencji dla interfejsu. Twórcy aplikacji są wybredni, zadufani w sobie, a przede wszystkim działają w pośpiechu. Elegancko zaprojektowany interfejs API, którego obsługa sprawi przyjemność projektantom, zyska zwolenników i zadowolonych użytkowników w stopniu, na jaki nie pozwolą działania cechujące się mniejszą starannością.

Nie rób niczego dziwnego

Wyróżnić można także inne aspekty prostoty, które mogą przyczynić się do ułatwienia adaptacji Twojego interfejsu API. Jednym z czynników powodzenia jest uczestniczenie w spotkaniach, które mogą być już znane projektantom.

Jest to szczególnie ważne w branży zabezpieczeń. Wiele interfejsów API oferuje niestandardowe lub złożone schematy zabezpieczeń, które wymagają od projektantów opanowania ich od podstaw, co jest znaczną przeszkodą na drodze do decyzji o adaptacji interfejsu. Zastanów się nad użyciem powszechnych standardów, takich jak protokół OAuth bądź innych ogólnie znanych schematów zabezpieczeń. Dzięki temu nie tylko możesz zwiększyć szanse adaptacji interfejsu, ale też zmniejszyć własny (oraz projektantów) nakład pracy związany z projektowaniem.

Mniej znaczy więcej

Udane interfejsy API często na początku zapewniają absolutnie minimalną liczbę funkcji, a później z upływem czasu i po uzyskaniu opinii stopniowo dodawane są kolejne.

Po pierwsze, gdy już udostępniono interfejs API, nie ma odwrotu. Oznacza to, że po opublikowaniu funkcjonalności i zbudowaniu na ich bazie aplikacji przez projektantów bardzo, ale to bardzo trudno wycofać takie funkcje. W przypadku witryny internetowej wymagane jest jedynie ukrycie funkcji. Ograniczenie funkcjonalności interfejsu API wiąże się z niemiłą decyzją o dalszym wspieraniu funkcji do momentu, aż kworum projektantów zacznie używać następnej ulepszonej wersji tej funkcji lub jej iteracji. Może to również spowodować ryzyko problemów z aplikacjami projektantów i wynikającego z tego ich gniewu.

Po drugie, całkiem prawdopodobne jest to, że klienci będą się domagać zmiany rozwoju interfejsu API w innych kierunkach, niż wcześniej sobie w ogóle wyobrażano. Dotyczy to zwłaszcza prywatnych interfejsów API, w przypadku których wewnętrzne zespoły projektantów mają bliższe relacje z dostawcą interfejsów API, a także większy wpływ. O czymś takim nieustannie się dowiadujemy. Niezależnie od tego, czy chodzi o strategię, typ projektanta, rodzaj aplikacji, czy o konkretną funkcję, która okazuje się wartościowa, interfejs API często rozwija się w innym kierunku, niż przewidywano. W takiej sytuacji rozpoczęcie od minimalnego poziomu funkcji może dać możliwość najbardziej przejrzystego i najszybszego rozwoju produktu.

Prawdopodobnie istnieje więcej powodów niż podane, ale ogólnie rzecz biorąc, prawie zawsze dobrym rozwiązaniem będzie rozpoczęcie od mniejszej liczby funkcji i dodawanie kolejnych w razie potrzeby. Jest to zgodne z zasadą, która głosi, że mniej znaczy więcej.

Ukierunkowanie na konkretny segment projektantów

Przed zastanowieniem się nad sposobem uruchomienia interfejsu API pomyśl o tym, kto ma być jego odbiorcą i jakie działanie ma zostać przez niego podjęte.

Jak w przypadku wszystkich dobrych programów marketingowych kampania marketingowa związana z interfejsem API może być znacznie ułatwiona przez ustalenie konkretnego segmentu projektantów lub aplikacji, które mają zostać uwzględnione. Uprości to doprecyzowanie strategii, taktyki, zasobów i metody pomiaru skali powodzenia.

Kontaktując się z firmami w trakcie uruchamiania interfejsu API, często zadajemy następujące pytanie: „Kim są odbiorcy docelowi?". Jeśli w odpowiedzi usłyszymy, że wszyscy, będziemy zmartwieni. Podobnie jeśli zadamy pytanie: „Jakiego rodzaju aplikacje mają zostać zbudowane?", będziemy zasmuceni po usłyszeniu, że chodzi o wszystkie rodzaje.

Z czego to wynika? Tak naprawdę trudno stworzyć interfejs API, który spełni wymagania każdego możliwego elementu oraz przypadku użycia. Jeśli nawet otrzymano by idealny interfejs API, nie jest możliwy skuteczny marketing dla tych wszystkich segmentów.

Po części atrakcyjność interfejsu API, prywatnego lub publicznego, polega na tym, że umożliwia on projektantom wprowadzanie innowacji w sposób, jakiego dostawca interfejsu może nie przewidzieć. Oznacza to, że podczas uruchamiania interfejsu ważne jest określenie oczekiwań odnośnie do tego, jak usługa będzie prawdopodobnie używana. Dzięki temu możliwe będzie zapewnienie interfejsu API, który spełni wymagania dotyczące przypadków użycia. Dla każdego z nich ustal, jacy prawdopodobnie będą odbiorcy docelowi, a następnie projektuj pod ich kątem.

Po odniesieniu sukcesu w przypadku pierwszego segmentu możesz dodać nowych partnerów, którzy z kolei umożliwią rozszerzenie projektu o wsparcie ich konkretnych potrzeb. Jeśli uruchamiany jest publiczny interfejs API, przeprowadź analizy dotyczące demografii segmentów projektantów według języka lub typu platformy aplikacji.

Kwestie techniczne projektu interfejsów API

W tym podrozdziale opisano kwestie projektowe natury filozoficznej i technicznej, które w dużym stopniu wpływają na sposób funkcjonowania interfejsu API.

Usługa REST

Różne warianty usługi REST (*Representational State Transfer*) to obecnie preferowana metoda tworzenia interfejsów API. Styl bazujący na tej usłudze został opracowany w ramach pracy doktorskiej przez Roya Fieldinga, który był jednym z twórców protokołu HTTP.

Zasadniczo Fielding zaproponował użycie tego protokołu do komunikacji między komputerami. W konsekwencji usługa REST oparta jest na standardzie HTTP. Korzystając ze składników protokołu HTTP, usługa dzieli przestrzeń nazw na zestaw „zasobów" opartych na unikalnych wzorcach identyfikatorów URI. Ponadto usługa REST stosuje standardowe metody protokołu HTTP (`GET`, `POST`, `PUT` i `DELETE`) do odwzorowywania operacji na bazie tych „zasobów". Te standardowe metody dokonują odwzorowania na funkcje *create* (utwórz), *read* (odczytaj), *update* (aktualizuj) i *delete* (usuń), które są znane generacjom programistów w postaci skrótu CRUD.

Identyfikatory URI i adresy URL: czym się różnią?

W świecie standardów internetowych identyfikator URI (*Uniform Resource Identifier*) to ogólne odwołanie do „zasobu" dostępnego w sieci. Może to być bardzo konkretne odwołanie opisujące protokół sieciowy, który ma zostać użyty do uzyskania dostępu do odwołania w sieci, metodę zapewniającą ten dostęp oraz miejsce, w którym odwołanie ma być szukane. Na przykład *http://helion.pl/ksiazki/autocad-2014-pl-andrzej-pikon,ac23pl.htm* to identyfikator URI. Identyfikator może być też znacznie ogólniejszy. Jeden z jego podzbiorów jest identyfikowany przez skrót URN (*Uniform Resource Name*), który po prostu identyfikuje unikalny identyfikator obiektu.

W historii internetu termin adresu URL (*Uniform Resource Locator*) często był używany do odwoływania się do typu identyfikatora URI, który obejmuje protokół sieciowy. Z technicznego punktu widzenia takie zastosowanie jest poprawne, ale w świecie standardów internetowych miało miejsce stopniowe przechodzenie do wykorzystywania skrótu URI w przypadku takich odwołań. W niniejszej książce będziemy się trzymać takiej definicji i konsekwentnie używać terminu URI.

W usłudze REST identyfikator URI w unikalny sposób odwołuje się do zasobu, obiektu lub kolekcji obiektów. Fielding sformalizował tę strukturę i zapewnił ją w postaci prostej metody projektowania interfejsu API, który będzie działać w przypadku każdego komputera bądź systemu operacyjnego. Na przykład program na komputerze A wymaga sprawdzenia listy

klientów. Dysponuje on informacją o istnieniu na komputerze zasobu zdefiniowanego przez identyfikator URI, w którym może uzyskać dostęp do listy. Wszystkie działania, które mogą być realizowane w odniesieniu do „klienta" obiektu, takie jak usuwanie lub dodawanie, są dostępne za pośrednictwem odsyłaczy i reprezentowane jako dane XML (zgodnie z pierwotną propozycją Fieldinga dane XML i hipertekst są kluczowymi elementami usługi REST).

Obecnie prawie każda platforma informatyczna może komunikować się z serwerem HTTP. Najlepsze interfejsy API bazujące na usłudze REST potrzebują do działania niewiele więcej niż podstawowej obsługi protokołu HTTP. Porównaj to z wcześniejszymi rozwiązaniami, takimi jak SOAP, które do połączenia się z serwerem wymagają złożonego stosu klienta.

Niestety, podobnie jak każde złożone zagadnienie informatyczne termin REST powoduje niejasności i dyskusję. Czasami używany jest niewłaściwie, a czasami zwolennicy usługi REST zbytnio przesadzają z budowaniem świadomości tego, że niepoprawne jest stosowanie go w mniej czystej formie. W tym miejscu chcemy się zająć związanymi z tym wątpliwościami.

Usługa REST w czystej formie

W swojej najczystszej formie usługa REST jest zgodna z wytycznymi zawartymi w pracy doktorskiej Fieldinga, a także w nowszych pracach i wpisach zamieszczonych na blogu. Centralne znaczenie dla stylu usługi REST ma pojęcie kryjące się pod skrótem HATEOAS (*Hypermedia as the Engine of Application State*).

Interfejs API, który przestrzega reguł HATEOAS, identyfikuje się sam za pomocą kontraktu bardzo różniącego się od kontraktów innych typów interfejsów API. Zamiast definiowania listy działań, jakie klient może zrealizować w dokumencie statycznym, taki interfejs API wymaga od używającego go klienta wykrycia funkcji zapewnianych przez interfejs. Klient, który korzysta z interfejsu opartego na usłudze REST, łączy się najpierw z serwerem i wykonuje metodę `GET` dla głównego identyfikatora URI. Z kolei ten identyfikator zwraca listę dodatkowych identyfikatorów URI, które mogą być używane na potrzeby dodatkowych operacji itp.

Inaczej mówiąc, klient interfejsu API REST realizuje następujące czynności:

`GET` *http://api.myapi.com/*
: Metoda zwraca dokument „powitalny", który zawiera listę dodatkowych identyfikatorów URI.

`GET` *http://api.myapi.com/customers*
: Klient stwierdza, że jeden z odsyłaczy w dokumencie „powitalnym" opisuje sposób uzyskania listy klientów, dlatego wywołuje ten identyfikator URI (jeśli odsyłacz ten okazał się niepoprawny, klient nie powinien go używać — technologia HATEOAS wyklucza trwałe umieszczanie identyfikatorów URI po stronie klienta).

`POST` *http://api.myapi.com/customers*
: Klient stwierdza, że metoda `GET` z poprzedniego kroku zwróciła inny odsyłacz do identyfikatora URI, który może być stosowany do dodania nowego klienta, dlatego klient wywołuje ten odsyłacz.

Klient nie zmienia się
: Rzecz w tym, aby nigdy nie umieszczać identyfikatora URI na trwałe, lecz by wykrywać go za pomocą interfejsu API.

Innymi słowy, klient, który przestrzega wszystkich reguł usługi REST, zachowuje się dokładnie tak jak człowiek przeglądający witrynę internetową, korzystając wyłącznie z wyświetlanych odsyłaczy (klient ignorujący wytyczne HATEOAS zachowuje się z kolei jak użytkownik, który tworzy zakładki dla konkretnych stron znajdujących się głęboko w strukturze identyfikatorów URI witryny; w pewnym momencie takie zakładki przestaną się sprawdzać).

Klienty i serwery zgodne z regułami HATEOAS są rzeczywiście skalowalne i rozszerzalne. Serwer może zmienić postać i funkcjonalność interfejsu API, a nawet usunąć określone funkcje bez negatywnego wpływu na klienta, ponieważ został on tak zbudowany, aby dynamicznie dostosowywał się do zmian po stronie serwera.

Pragmatyczna usługa REST

Jeśli w przeszłości korzystano z interfejsów API REST, treść wcześniejszego punktu może brzmieć obco. Prawdopodobnie używano interfejsów API REST, które wcale nie działają w sposób opisany w tym punkcie. Być może zamiast stosowania jedynie odsyłaczy zwróconych przez serwer przyzwyczajono się do trwałego wstawiania identyfikatorów URI. Czasami ma to miejsce, ponieważ tak zwane interfejsy API REST wcale nie bazują na usłudze REST, lecz po prostu jako metodę komunikacji wykorzystują format JSON lub XML za pośrednictwem protokołu HTTP (wkrótce zostanie opisany taki interfejs API).

Częściej jednak jest tak, gdyż interfejsy API były świadomie tworzone w innym celu. Są one zgodne z regułami REST, lecz nie wszystkimi. Takie interfejsy API są łatwe do opanowania i obsługi, a ponadto reprezentują większość publicznych interfejsów. W odniesieniu do tych interfejsów API będziemy używać terminu pragmatycznej usługi REST.

Dlaczego wiele interfejsów API zostało zaprojektowanych w taki „pragmatyczny" sposób? Po części wynika to z faktu, że zasady HATEOAS stawiają programiście po stronie klienta poprzeczkę bardzo wysoko. Programista, który nieumyślnie lub celowo trwale umieści ścieżkę identyfikatora URI w aplikacji, może w przyszłości doznać szoku, a zespół tworzący interfejs API po stronie serwera może po prostu poinformować klienta o braku możliwości zapewnienia zgodności ze specyfikacją.

Takie interfejsy API mogą też być projektowane w ten sposób, gdyż nowoczesna technologia związana z interfejsami API (obejmująca użycie serwerów mediacji między klientem a serwerami, które udostępniają interfejsy API umożliwiające ich dostawcom dynamiczne ponowne zapisywanie identyfikatorów URI i treści) sprawia, że bardziej funkcjonalne jest utrzymywanie spójnej struktury identyfikatorów URI.

Choć teoretycznie HATEOAS to dobra metoda w przypadku projektowania interfejsu API, w praktyce może nie mieć zastosowania. Ważne jest uwzględnienie odbiorców interfejsu API i ich możliwych metod budowania aplikacji bazujących na interfejsie, a także wzięcie tego pod uwagę przy podejmowaniu decyzji projektowych. W niektórych sytuacjach HATEOAS może nie być właściwym wyborem.

Zasady pragmatycznej usługi REST

W wariancie pragmatycznym usługi REST wykorzystywane są jej najlepsze elementy. W tym przypadku zdano sobie sprawę z tego, że programistom zależy na jak najszybszym zaznajomieniu się z możliwościami interfejsu API, a ponadto chcą z nich skorzystać bez konieczności pisania dużej ilości dodatkowego kodu.

Proponujemy następujące zasady pragmatycznej usługi REST:

Identyfikatory URI są istotne
: Dobrze opracowany wzorzec identyfikatorów URI powoduje, że interfejs API jest łatwy do zastosowania, poznania i rozszerzenia, tak jak ma to miejsce w przypadku starannie zaprojektowanego interfejsu API powiązanego z tradycyjnym językiem programowania. W usłudze REST w czystej postaci reguła ta jest zastępowana przez HATEOAS.

Parametry są istotne
: Użyj standardowego i łatwego do zidentyfikowania zestawu opcjonalnych operacji dla każdego wywołania interfejsu API.

Format danych jest istotny
: Spraw, aby programiści bez trudu mogli zrozumieć, jakiego rodzaju danych oczekuje interfejs API, jakiego typu dane zostaną przez niego zwrócone, a także jak to zmienić.

Kody powrotu są istotne
: Użyj kodu 404, gdy na przykład ścieżka nie prowadzi do rzeczywistego obiektu lub kolekcji, a nie zwracaj ogólnego kodu błędu, który wymaga właściwego zinterpretowania przez użytkownika.

Wszystko inne powinno być ukryte
: Informacje o zabezpieczeniach, ograniczaniu transferu, routingu itp. mogą i powinny być ukrywane w nagłówkach protokołu HTTP.

Ustal przejrzyste konwencje oznaczania wersji
: Czy na przykład identyfikator URI powinien zawierać numer wersji albo czy powinien on być parametrem? Jaka wersja powinna zostać dostarczona w przypadku niespełnienia dowolnego z tych warunków? Oczywiście w tym przypadku przyjmuje się, że Twój interfejs API w ogóle korzysta z wersji.

W szczególności powyższe zasady sugerują następujące reguły dotyczące identyfikatorów URI:

- Ścieżki URI odwołujące się do kolekcji obiektów powinny uwzględniać rzeczownik w liczbie mnogiej, na przykład `/klienci`, aby odnosić się do zbioru wszystkich klientów.
- Ścieżki URI odwołujące się do pojedynczego obiektu powinny zawierać rzeczownik w liczbie pojedynczej, po którym następuje unikalny klucz podstawowy. Na przykład `/klienci/Bogdan` odwołuje się do klienta z podstawowym identyfikatorem Bogdan, a `/konta/123456` odnosi się do konta o numerze 123456.
- Poprawne jest rozpoczęcie identyfikatora URI od ścieżki identyfikującej, takiej jak ścieżka zawierająca numer wersji lub informacje o środowisku.
- Po ścieżce identyfikującej w identyfikatorze URI nie powinno się znaleźć nic innego, z wyjątkiem kolekcji i obiektów.

- Zestaw standardowych parametrów zapytania powinien być używany dla kolekcji w celu umożliwienia elementowi wywołującemu kontrolowania tego, jaka część kolekcji zostanie udostępniona. Na przykład parametr `count` pozwala określić liczbę obiektów, które zostaną zwrócone z dużej kolekcji, parametr `start` służy do ustalenia początkowego miejsca liczenia obiektów, a parametr `q` reprezentuje ogólne wyszukiwanie w dowolnej formie obejmujące kolekcję obiektów.

Sugerujemy również następujące reguły dotyczące obiektów:

- Pojedyncze obiekty powinny obsługiwać metodę `GET` dla odczytu, metodę `PUT` dla aktualizacji i metodę `DELETE` dla usuwania.
- Obiekty kolekcji powinny obsługiwać metodę `GET` w celu ponownego wczytania całości lub części kolekcji, a także metodę `POST`, aby dodać nowy obiekt do kolekcji.
- Pojedyncze obiekty mogą obsługiwać metodę `POST` w celu zapewnienia możliwości zmiany stanu. Za pomocą tej metody możesz na przykład wysłać nowy dokument JSON bądź XML do obiektu w celu zmiany określonych pól albo wywołać zmianę stanu lub działanie bez zastępowania całego obiektu.

Przykład: projektowanie z wykorzystaniem pragmatycznej usługi REST

W tabeli 5.1 pokazano interfejs API koszyka zakupów, który nie przestrzega konwencji usługi REST w czystej postaci. Interfejs nie jest zgodny ani z czystym, ani z pragmatycznym wariantem usługi REST.

Tabela 5.1. Niewłaściwa droga do usługi REST

Zadanie	Operacja	Identyfikator URI
Umieszczanie nowego produktu w koszyku	`POST`	*http://api.zakupy.com/WstawianieNowegoProduktu*
Usuwanie produktu z koszyka	`POST`	*http://api.zakupy.com/UsuwanieProduktu*
Wyświetlanie zawartości koszyka	`GET`	*http://api.zakupy.com/ZawartoscKoszyka?Identyfikator ↪koszyka=X*
Pobieranie produktu w koszyku	`GET`	*http://api.zakupy.com/WyswietlanieProduktu?Identyfikator ↪koszyka=X&identyfikatorproduktu=Y*
Usuwanie całego koszyka	`POST`	*http://api.zakupy.com/UsuwanieKoszyka*

Ten interfejs API nie jest trudny w użyciu, ale konieczne jest opanowanie poszczególnych operacji. Może to być kłopotliwe, jeśli istnieje wiele operacji lub interfejs API się rozwija. Wyobraź sobie, co by było, gdyby oprócz koszyka pojawiło się 50 innych typów obiektów, a wraz z nimi cały zestaw operacji. Z tego powodu opracowana dokumentacja interfejsu API byłaby obszerna, a ponadto wymagałaby przeszukiwania dla każdego nowego wywołania dotyczącego interfejsu. Projektant może być na przykład zmuszony do sprawdzenia, czy operacja `InsertNewItem` umieszcza jedynie produkt w koszyku, natomiast operacja `InsertNewItemIntoWishlist` musi zostać użyta dla listy życzeń klienta i tak dalej aż do znudzenia.

Łatwiejszy do opanowania jest koszyk zakupów bazujący na wariancie pragmatycznym usługi REST, który zaprezentowano w tabeli 5.2.

Tabela 5.2. Koszyk zakupów bazujący na wariancie pragmatycznym usługi REST

Zadanie	Operacja	Identyfikator URI
Umieszczanie nowego produktu w koszyku	`POST`	*http://api.zakupy.com/koszyk/Nazwakoszyka*
Usuwanie produktu z koszyka	`DELETE`	*http://api.zakupy.com/koszyk/Nazwakoszyka/produkt/Nazwaproduktu*
Wyświetlanie zawartości koszyka	`GET`	*http://api.zakupy.com/koszyk/Nazwakoszyka*
Pobieranie produktu w koszyku	`GET`	*http://api.zakupy.com/koszyk/Nazwakoszyka/produkt/Nazwaproduktu*
Zastępowanie produktu w całości	`PUT`	*http://api.zakupy.com/koszyk/Nazwakoszyka/produkt/Nazwaproduktu*
Usuwanie całego koszyka	`DELETE`	*http://api.zakupy.com/koszyk/Nazwakoszyka*

Powyższy zestaw wzorców identyfikatorów URI ułatwia rozszerzanie interfejsu API za pomocą prostych sposobów. Co na przykład będzie, jeśli w dowolnym momencie będą miały zostać wyświetlone wszystkie koszyki w systemie? Odpowiednie parametry zostaną dodane do identyfikatora URI *http://api.zakupy.com/koszyki* za pośrednictwem metody `GET` protokołu HTTP.

Parametry zapytania w dalszym ciągu pełnią ważną funkcję, czyli umożliwiają podanie dodatkowych opcji. Wyobraź sobie na przykład bardzo duży koszyk zakupów, dla którego wyniki mają być stronicowane. Aby przyjrzeć się produktom o numerach od 20 do 29, możesz użyć identyfikatora URI podobnego do następującego: *http://api.zakupy.com/koszyk/Nazwakoszyka?start=*
↪20&count=10.

Czasami usługa REST wymaga „odpoczynku"

Usługa REST to znakomity sposób zdefiniowania interfejsu API, który jest prosty do opanowania i rozwijania. Jednak użycie tylko usługi REST nie sprawi, że interfejs API będzie łatwy do nauczenia i wykorzystania, a ponadto dzięki temu nie stanie się on automatycznie efektywny. Na przykład interfejsy API REST często są projektowane do obsługi żądań w bardzo precyzyjny sposób, co powoduje pojawienie się wielu różnych zasobów, które przetwarzają specyficzne typy żądań. W rezultacie do zbudowania pojedynczej strony na urządzeniu przenośnym może być potrzebnych wiele osobnych żądań interfejsu API. Ponieważ każde takie żądanie pochodzące z urządzenia przenośnego zajmuje wiele czasu i wpływa na wykorzystanie energii baterii, drobiazgowy interfejs API zmniejsza wydajność i ma wpływ na komfort pracy użytkownika.

Możliwe jest jednak zaprojektowanie efektywniejszego interfejsu API bez całkowitego rezygnowania z usługi REST. Na przykład mogą zostać utworzone metazasoby wyższego poziomu, które w ramach jednego zasobu łączą kilka zasobów niższego poziomu. Inna metoda jest określana mianem żądania masowego. Aby na przykład pobrać jednocześnie 10 elementów z interfejsu API REST, wyślij jedno żądanie HTTP zawierające listę identyfikatorów URI, a następnie zwróć kolejno wszystkie odpowiedzi tak, jakby każda z ich dziesięciu została wygenerowana osobno.

Porównanie formatów XML i JSON

Początkowo w przypadku interfejsów API REST faktycznym standardem był format XML. Nawet obecnie istnieje wiele sensownych scenariuszy, w których format XML jest właściwym wyborem. Jako model danych format ten został użyty do stworzenia standardowych składni opisujących niektóre z najbardziej złożonych zbiorów danych na świecie, obejmujących kontrakty terminowe i polisy ubezpieczeniowe. Twórcy kodu XML mogą definiować składnię umożliwiającą pobieranie modeli informacji z różnych obszarów, łączenia ich w jeden złożony dokument bez obaw, że wystąpi między nimi konflikt, sprawdzania poprawności całości za pomocą standardowej technologii oraz transformowania i edytowania z wykorzystaniem zaawansowanych narzędzi. Firmy z branż, które wymagają tak dużych możliwości, naprawdę nie mogą się obejść bez formatu XML.

Większość interfejsów API ciągle korzysta z dość prostych języków programowania, dlatego format JSON (*JavaScript Object Notation*) będzie znacznie łatwiejszy w użyciu dla programistów po obu stronach interfejsu API.

Format JSON został opracowany przez Douglasa Crockforda, czyli jedną z osób związanych z początkami języka JavaScript. Crockford zdecydował się na stworzenie języka definicji danych obejmującego niewielki podzbiór elementów języka JavaScript, który wypełnia lukę między obiektami języków programowania i technologiami internetowymi. Zapewniając łatwość analizy i translacji, format JSON stał się popularną platformą dla interfejsów API, gdyż bez trudu może się komunikować z serwisami internetowymi i aplikacjami dla urządzeń przenośnych, które zwykle są tworzone za pomocą języka JavaScript. Ponadto format JSON jest znacznie zwięźlejszy niż format XML. Od momentu pojawienia się formatu JSON opracowano analizatory składni i generatory JSON dla większości języków programowania. Obecnie obsługa formatu JSON stanowi standardowy element wielu środowisk. Ponieważ format JSON został zaprojektowany jako podzbiór języka JavaScript, obiekty JSON z łatwością mogą być przekształcane w obiekty większości języków programowania bez konieczności dodatkowego analizowania kodu przez programistę. Inaczej sytuacja przedstawia się w przypadku formatu XML, który oferuje złożony i bogaty w możliwości model danych, zmuszając programistów do konwersji nawet niewielkich dokumentów XML do postaci obiektu i odwrotnie. Format XML zwiększa również złożoność mechanizmów analizowania składni, wprowadzając przestrzenie nazw, atrybuty, warianty kodowania tekstu itp.

Więcej informacji o usłudze REST i formacie JSON

Treść oryginalnej rozprawy dotyczącej usługi REST dostępna jest pod adresem *http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm*.

Specyfikację formatu JSON można znaleźć pod adresem *http://json.org/*.

Kontrola wersji i projekt interfejsu API

Decyzja dotycząca tego, czy interfejs API ma mieć numer wersji, czy ma być pozbawiony wersji, stanowi ważną kwestię projektową, ponieważ aplikacje budowane na bazie interfejsu zależą od określonych funkcji działających w konkretny sposób. Oznacza to, że interfejs API jest „ucieleśnieniem" kontraktu między organizacją publikującą a projektantem. W miarę pojawiania się nowych wersji interfejsu API kontrakt ten powinien pozostać w niezmienionej postaci. Fakt ten wymaga rozważenia w kontekście ciągłych żądań użytkowników dotyczących aktualizacji funkcji oraz wewnętrznych możliwości firmy i chęci wspierania na bieżąco starszych zestawów funkcji. Zespoły projektujące interfejsy API są zwykle niewielkie i mają ograniczone możliwości obsługiwania nowych wersji. Projektanci nieustannie

oczekują nowszych i bardziej rozbudowanych wersji interfejsu API, natomiast użytkownicy końcowi spodziewają się ulepszanych aplikacji. Rzeczywistość jest taka, że zespół projektowy może jednocześnie obsługiwać tylko kilka wersji.

Istnieje wiele różnych metod rozwiązania tych problemów. Chcemy zaproponować jedną z nich, która składa się z trzech następujących drobnych kroków:

- W jakimś miejscu, które jest widoczne dla każdego klienta lub wywołania interfejsu API, dołącz wersję 1. (przeważnie informacja ta jest uwzględniana w ścieżce identyfikatora URI).
- Każdorazowo po zmodyfikowaniu interfejsu API za wszelką cenę próbuj unikać ciągłego zwiększania numeru wersji aż do wersji 2.
- Rozważ pominięcie informacji o wersji w identyfikatorze URI bądź parametrze, aby wskazać interfejsowi API, że ma skorzystać z pierwszej lub najnowszej wersji.

W przypadku tego rozwiązania identyfikowane są dwie kwestie. Po pierwsze, jest to konieczność intensywnej pracy w celu zachowania w niezmienionej postaci kontraktu interfejsu API. Kontrakt może być modyfikowany w sposób, który nie wpływa na działanie istniejących aplikacji. Aby jednak wprowadzić zmianę, która może spowodować, że aplikacje przestaną działać, niezbędne będzie jawne zaplanowanie takiej sytuacji. Po drugie, jest to zdanie sobie sprawy z tego, że czasami w okresie dostępności interfejsu API konieczne okaże się przyznanie do popełnienia błędu, co będzie wymagać rozpoczęcia od nowa z wykorzystaniem wersji o numerze 2.

Takie rozwiązanie zapewnia też elastyczność zmiany kontroli wersji, gdy zaistnieje taka potrzeba. Jeśli jednak projektant nie zmieni numeru wersji w swoim kodzie, jego żądania w dalszym ciągu będą obsługiwane bez zakłóceń.

Alternatywnie dla interfejsu API możesz opracowywać strategię dotyczącą wersji, a ponadto rozważać następujące kwestie:

- Czy numer wersji złożony jest z trzech lub czterech cyfr?
- Czy numer wersji zmienia się każdorazowo po wprowadzeniu nowej wersji interfejsu API?
- Czy w okresie istnienia interfejsu API albo firmy numer wersji będzie mieć kiedykolwiek postać dwucyfrową?
- Czy numer wersji oparty jest na dacie ostatniej modyfikacji interfejsu API?

Jeśli rozpatrujesz powyższe zagadnienia, prawdopodobnie dokonujesz niewłaściwego wyboru. Takie rozwiązania mogą się sprawdzić w przypadku aplikacji lub systemów, które mają postać bardzo małej, kontrolowanej dystrybucji. W odniesieniu do publicznych i prywatnych interfejsów API mogą jednakże spowodować mnóstwo problemów.

Zapamiętaj, że interfejs API to kontrakt. Publikując interfejs API i tworząc jego dokumentację, składasz również obietnicę tego, że bez wcześniejszego powiadomienia nie spowodujesz, iż istniejące aplikacje przestaną działać.

Numer wersji przekazuje zasadniczo następującą informację: „Proszę zakończyć korzystanie ze starego kontraktu i przejść na nowy". Informuje on też jednak o tym, że stary kontrakt jest nadal ważny. Opublikowanie nowej wersji każdorazowo będzie wymagać nakładu pracy od osób używających interfejsu API, co nie będzie dla nich powodem do zadowolenia. Bez wątpienia część z tych osób nigdy nie przejdzie na nową wersję, a to będzie oznaczać, że ich aplikacje przestaną działać, gdy w końcu wyłączysz obsługę starej wersji interfejsu API.

Z kolei im dłużej możesz korzystać ze zautomatyzowanego testowania regresyjnego w celu upewnienia się, że nigdy nie było konieczne zwiększania numeru wersji, tym mniej czasu poświęcisz na omawianie z klientami kwestii kontroli wersji. Oznacza to także, że niezbędne będzie wspieranie starszego kodu i zautomatyzowanych testów do momentu zaprzestania obsługi starszej wersji.

Zastanawiając się nad sposobem obsługi kontroli wersji, spróbuj odpowiedzieć na następujące pytania:

- Co się stanie ze starszymi wersjami?
- Co się stanie z aplikacjami zaprojektowanymi na bazie starszych wersji?
- Czy nowe wersje obsługują wszystkie funkcje starszych aplikacji albo czy klienci używający interfejsu API muszą zaktualizować swoje aplikacje, aby mieć możliwość korzystania z nowszych wersji?
- Czy dysponujesz planem minimalizowania liczby sytuacji, w których prosisz projektantów o aktualizowanie lub zmienianie wersji?
- Ile możesz utworzyć wyjątkowych wersji dla „szczególnych" partnerów? Jaki jest koszt obsługi związany z taką decyzją?

Strategia opracowywana pod kątem wycofywania starych wersji interfejsu API będzie mieć wpływ na strategię dotyczącą kontroli wersji. Użytkownicy interfejsu będą bardziej zadowoleni, jeśli udostępnisz przejrzyste reguły

określające, przez jaki czas będzie wspierana przestarzała wersja, zamiast przesłania jedynie tydzień wcześniej powiadomienia o zamiarze wyłączenia starej wersji lub wprowadzenia wielu zmian. Wcześniejsze przedstawienie takiej strategii wycofywania wersji przyczyni się do zdobycia zaufania i zachęci do adaptacji interfejsu API.

Godne uwagi jest to, że strategie związane z kontrolą wersji będą inne dla publicznych i inne dla prywatnych interfejsów API. W przypadku tych pierwszych społeczność projektantów to prawdopodobnie duża grupa potencjalnie nieznanych twórców aplikacji. Bardzo trudne może się okazać zrozumienie ich potrzeb, oczekiwań i możliwości dostosowania do zmian wprowadzonych w interfejsie API. Jednak w przypadku prywatnych interfejsów API zespoły korzystające z interfejsu są zwykle ze sobą bliżej powiązane, bardziej przygotowane na dokonywane w nim zmiany, bardziej elastyczne, a często motywujące do rozwijania interfejsu. Projektanci prywatnych interfejsów API mogą nie wymagać wcześniej czasu na przygotowanie do nowych wersji. Mogą oni nawet wcale nie potrzebować wielu wersji.

Zastosowanie warstwy mediacji

Zmiana wersji interfejsu API nie musi być bolesna dla użytkowników lub uciążliwa dla systemów. Technika nazywana mediacją może ograniczyć niektóre trudności towarzyszące niezbędnym zmianom wersji. Jest to możliwe, ponieważ, tak jak wcześniej wspomniano, interfejsy API są zwykle oparte na samoopisujących się formatach danych. Bez użycia jakiegokolwiek specjalnego kodu warstwa umieszczona między klientem interfejsu API a końcowym serwerem API może transformować każde żądanie i odpowiedź ze starej do nowej wersji, co nie wymaga wprowadzania żadnych zmian na serwerze.

W jaki sposób możesz udostępniać różne wersje tego samego interfejsu API i zarządzać nimi albo dokonywać „mediacji" (lub transformacji) zawartości i składni interfejsu API?

Alternatywne rozwiązania obejmują:

- obsługę wielu interfejsów API (uciążliwe);
- jak najdłuższe wstrzymywanie się z nową wersją i objęcie klientów modelem typu „jedna wielkość pasuje do wszystkiego" (bardziej uciążliwe);
- zapewnienie możliwości mediacji lub warstwy, która pozwala dokonywać transformacji między właściwościami interfejsu API, takimi jak protokół, dane, wersja i dane uwierzytelniające.

Użycie warstwy mediacji wraz ze strategią zamiast logiki biznesowej (więcej na ten temat napisano w dalszej części rozdziału) może ułatwić pracę w przypadku kontroli wersji (a także niektórych innych kluczowych kwestii, takich jak zabezpieczenia, ograniczenie transferu itp.).

Choć jest to przydatna technika, to jeśli zmiany powodują nowe bądź zmieniające się elementy odpowiedzi, mediacja może nie uchronić skutecznie projektantów aplikacji przed modyfikacjami w interfejsie API.

Skok na głęboką wodę: rezygnacja z wersji

W praktyce większość interfejsów API korzysta z kontroli wersji, aby ochronić ich klientów przed rozwijającym się interfejsem API (projektanci nie lubią interfejsów, które cały czas się zmieniają; dotyczy to nawet tych najbardziej znanych). Z kolei bardzo łatwo można przesadzić z kontrolą wersji, na skutek czego obsługa powiększającej się listy wersji może być bardzo utrudniona i czasochłonna. Na przykład interfejs API firmy Netflix obsługuje telewizory z technologią WiFi, które powiązane są z niezwykle specyficzną wersją tego interfejsu. Takie powiązanie czasami umieszczone jest w oprogramowaniu sprzętowym telewizora, a ponadto nie umożliwia wprowadzania zmian. Oczywiście większość osób kupuje telewizory z zamiarem użytkowania ich przez 7 – 10 lat. Oznacza to, że interfejs API, do którego odwołuje się telewizor, musi pozostać w niezmienionej postaci przez taki okres. Jeśli w ramach rozwoju interfejsu API firma Netflix każdego roku będzie publikować jego nowe wersje, ostatecznie może się okazać, że będzie wspierać tuzin lub więcej wersji interfejsu API! Taki model obsługi jest niemożliwy do utrzymania i z pewnością spowoduje wiele przestojów środowiska produkcyjnego.

Innym rozwiązaniem jest dążenie do całkowitego zrezygnowania ze stosowania wersji interfejsu API. Aby to osiągnąć, wymagana jest duża dyscyplina i dalekowzroczność.

Choć trudno podołać zwiększającemu się zapotrzebowaniu, z jakim ma do czynienia większość interfejsów API, jest to możliwe. Ponadto jest to cel warty osiągnięcia, ponieważ oferuje wiele korzyści. Współużytkowany system bazowy jest preferowany w przypadku zwiększającej się liczby odrębnych, lecz podobnych systemów. Dotyczy to szczególnie interfejsów API używanych przez dużą liczbę partnerów dostarczających urządzenia, z których część może nie zapewniać możliwości aktualizowania. Oznacza to, że takie urządzenia zawsze mogą wskazywać na tę samą wersję interfejsu API.

Interfejs API bez wersji organizacji NPR

Od początku swojego pojawienia się w 2007 r. prywatny interfejs API organizacji NPR pozbawiony był wersji. W 2008 r. udostępniono interfejs w wersji publicznej, dla którego również nie zastosowano wersji. Wszystkie aplikacje utworzone na bazie tego interfejsu API cechują się znakomitą stabilnością. Poza tym w ich przypadku można mieć pewność, że interfejs API będzie nadal rozwijany bez negatywnego wpływu na aplikacje.

Oto kilka kluczowych zasad, o których trzeba pamiętać przy rozważaniu tego rozwiązania:

- Dodawanie nowych funkcji do interfejsu API nie wymaga zwykle nowej wersji. Z tego względu stosuj następującą zasadę: Lepsza jest niekompletność niż niedokładność. Zasada ta sugeruje, że należy zrezygnować z funkcji w interfejsie API, jeśli przewiduje się spore szanse na to, iż będą one wymagać modyfikacji.
- Jeżeli to możliwe, projekt identyfikatorów URI i formaty odpowiedzi powinny być ogólne. Dla przykładu zamiast zwracania w wyniku pola o nazwie `prywatny_numer_telefonu` utwórz pole `numer_telefonu` i dodaj atrybut, który wyszczególnia typ (w tym przypadku byłby to typ `type="dom"`). Sprawi to, że dodawanie numerów telefonu dla nowej lokalizacji lub usuwanie prywatnego numeru telefonu nie wpłynie na projekt i strukturę interfejsu API.
- Istnieje tendencja polegająca na przypisywaniu interfejsowi API roli ogólnego potoku dystrybucji, który w równym stopniu spełnia cele wszystkich klientów. Zależnie od wymagań użytkowników interfejsu API coś takiego może utrudnić utrzymanie stabilnego modelu interfejsu. Jeśli nakład pracy związany z utrzymywaniem logiki biznesowej dla zbyt wielu urządzeń jest za duży, sensowne będzie utworzenie niestandardowych punktów końcowych API dla poszczególnych urządzeń (a nawet dla każdego żądania). Dzięki temu urządzenia będą traktowane w różny sposób, a ponadto możliwe będzie uniknięcie konieczności użycia wersji dla podstawowego interfejsu API podczas dodawania nowych funkcji w celu obsługi konkretnego urządzenia.
- Przeważnie mniejsze społeczności projektantów blisko powiązane z dostawcą interfejsu API (na przykład wewnętrzne zespoły projektantów tworzących oprogramowanie dla urządzeń przenośnych) są znacznie „odporniejsze" na zmieniający się interfejs. Oznacza to, że przy rozważaniu interfejsu API bez wersji kluczowe znaczenie ma określenie grupy

docelowej. W przypadku interfejsu API, takiego jak Google Maps, niemal niemożliwe jest wprowadzanie zmian bez kontroli wersji z powodu ogromnej liczby nieznanych projektantów, którzy z niego korzystają.

Zaletą interfejsu API bez wersji jest to, że nie jest wymagana obsługa wielu wersji systemu lub zapewnianie zgodności wstecz. Wyzwaniem związanym z takim interfejsem jest wymuszanie przez niego wcześniejszego rozważenia w szerszym zakresie możliwości adaptacji. Ponadto może być konieczne podjęcie niewygodnych decyzji, takich jak wstrzymanie funkcjonalności, które mogą być przydatne. Obsługa interfejsu API bez wersji jest często łatwiejsza do uzyskania w przypadku prywatnych interfejsów API niż w przypadku publicznych.

Projektowanie infrastruktury dla interfejsów API

W tym podrozdziale skoncentrujemy się na kwestiach projektowych związanych z infrastrukturą, które obejmują możliwość skalowania, buforowanie i ograniczanie transferu.

Jak duża powinna być infrastruktura interfejsów API? Podstawowa zasada brzmi: Projektuj pod kątem wymarzonej grupy odbiorców, ale uwzględniaj oczekiwane obciążenie. W przypadku niewielu programów związanych z interfejsami API dzienna liczba żądań w ciągu nocy zwiększa się z zera do 500 milionów. Nierozsądne będzie żądanie od zarządu zapewnienia dużego budżetu, jeśli wcześniejsze poniesienie wysokich kosztów nie jest uzasadnione ewidentnym przypadkiem biznesowym lub mocnym dowodem na to, że interfejs API będzie wymagał skalowania. Nawet usługa Twitter w początkowym etapie funkcjonowania miała wiele problemów ze skalowaniem (doprowadziły one do mnóstwa przestojów), ale była ona tak bardzo fascynująca i unikalna, że użytkownicy nie narzekali, gdy system okazał się niestabilny.

Centrum danych czy chmura?

Zwiększająca się liczba nowych witryn internetowych i interfejsów API działa z wykorzystaniem skalowalnej platformy usługi chmury, którą najczęściej jest platforma Amazon EC2. W przypadku takiej platformy wielkość początkowej bazy sprzętowej jest nieistotna, ponieważ może ona być szybko skalowana w górę lub w dół stosownie do potrzeb (czasami odbywa się to

dynamicznie, zależnie od sposobu skonfigurowania platformy). Większość dostawców interfejsów API będzie sobie znakomicie radzić, uruchamiając swoje systemy w usłudze chmury. Przeważająca liczba największych dostawców interfejsów API (na przykład Google, Yahoo! i Facebook) rezygnuje z centrów danych i stosuje sprzęt, którym bezpośrednio zarządza.

Niektóre firmy mają skonfigurowane centra danych i bardzo specyficzne wymagania dotyczące opóźnienia lub zgodności z wynikami audytu, które mogą zostać spełnione tylko przez zapewnienie fizycznego dostępu do sprzętu wykorzystywanego przez interfejs API do działania. W przypadku opóźnienia największy problem wielu dostawców interfejsów API ma związek z fizyczną odległością między serwerem a klientem. Z tego właśnie powodu część dostawców bazuje na sieciach dostarczania treści CDN (*Content Delivery Network*). W celu poradzenia sobie z opóźnieniem stosuje się również buforowanie.

Strategie buforowania

Z punktu widzenia systemów szybciej zawsze oznacza lepiej. Buforowanie może skrócić czasy odpowiedzi. Jeśli interfejs API jest powolny, korzystające z niego aplikacje także będą działać wolno. Jeżeli aplikacje nie mają dużej wydajności, użytkownicy końcowi z większym prawdopodobieństwem poszukają lepszych i szybszych opcji. W świecie internetowym długie czasy ładowania to jedna z podstawowych przyczyn opuszczania witryn przez internautów. Nie inaczej sprawa wygląda w świecie urządzeń przenośnych. Buforowanie to jeden ze sposobów zmniejszenia skali części problemów z wydajnością.

Pamiętaj o tym, że obecnie używane platformy interfejsów API są złożone z warstw, z których każda dysponuje pamięcią podręczną. Warstwa mediacji interfejsu API może buforować gotowe odpowiedzi interfejsu w pobliżu „granicy sieci", a serwer aplikacji może zawierać warstwę buforującą na potrzeby odpowiedzi interfejsu oraz warstwę służącą do buforowania wyników z bazy danych itp. Sieci CDN, takie jak Akamai, również pełnią swoją funkcję. Interfejsy API z treścią często będą zwracać odnośnik do treści, która z kolei przechowywana jest w sieci CDN.

Oczywiście w zależności od treści zmienna jest złożoność buforowania. Strona z prognozą pogody może być buforowana w prosty sposób. Prognozy rzadko aktualizowane są częściej niż raz na minutę. Z kolei buforowanie serwisu Twitter jest znacznie trudniejsze, ponieważ każdy użytkownik wyświetla inny strumień. Infrastruktura tego serwisu zawiera wiele warstw

buforowania, które dynamicznie dopasowują się do postaci zażądanej treści. Choć w przypadku wpisu w serwisie Twitter dotyczącego Kanyego Westa lub Ashtona Kutchera pamięć podręczna jest utrzymywana przez dłuższy czas z powodu liczby osób, które go żądają, dla każdej z tych gwiazd osie czasu są unikalne.

Oto kilka ogólnych rad dotyczących buforowania interfejsów API:

Najpierw dokonaj pomiaru
: Używana infrastruktura rejestrowania informacji bądź analizowania interfejsów API powinna umożliwić stwierdzenie, jakie wywołania interfejsu API są najpopularniejsze, a jakie mają najdłuższy czas. W pierwszej kolejności zakresem buforowania należy objąć najwolniejsze i najczęściej używane wywołania interfejsu API zwracające wynik, który nie zmienia się zbyt często.

Nie zapomnij o unieważnianiu
: Najtrudniejszym elementem buforowania nie jest podejmowanie decyzji o tym, co ma być buforowane, lecz określenie czasu, przez jaki proces ten ma trwać, a także ustalenie momentu unieważnienia buforowania. W tym przypadku konieczne jest uważne planowanie i testowanie. Czy treść w pamięci podręcznej będzie unieważniana na podstawie znacznika czasu w odpowiedzi? Czy możliwe jest użycie stałego czasu? Czy jedno wywołanie interfejsu API (na przykład `PUT` lub `DELETE`) może unieważnić pamięć podręczną?

Monitoruj działającą pamięć podręczną
: Nieefektywna pamięć podręczna z niskim wskaźnikiem trafień nie wpłynie pozytywnie na wydajność, a nawet może ją pogorszyć.

Doświadczenie firmy AccuWeather w zakresie globalnego dostarczania interfejsu API

Z jakimi problemami miała do czynienia firma AccuWeather w procesie globalnego dostarczania interfejsu API?

Lokalizacja. Interfejs API oferujemy w 37 różnych językach. Tłumaczenie obejmuje nawet poprawny styl dla konkretnego rynku lokalnego.

Wydajność i czas odpowiedzi to ważne kwestie dla klientów globalnych. Ostatnio przekroczyliśmy miliard wywołań interfejsu API dziennie.

Przy globalnym dostarczaniu istotne są kwestie związane z buforowaniem, a zwłaszcza optymalizowanie pamięci podręcznej pod kątem odpowiedniego poziomu szczegółowości systemu GPS. Nie jest konieczneponowne korzystanie z magazynu danych w celu uzyskania danych pogodowych na przesadnym poziomie szczegółowości systemu GPS.

Reguły biznesowe są odmienne w różnych obszarach geograficznych. Nasza marka wymaga właściwego tłumaczenia w każdym segmencie (na przykład przedrostek *accu-* może mieć różne znaczenia na rynku lokalnym).

Wyjątkowym wyzwaniem może być też wspieranie projektantów z całego świata. Dokładamy wszelkich starań, aby nasze materiały były w równym stopniu przejrzyste dla projektantów posługujących się różnymi językami.

Chris Patti, dyrektor ds. technologii, firma AccuWeather

Kontrolowanie ruchu sieciowego generowanego przez interfejsy API

Ograniczanie transferu to mechanizm kontroli zastosowany przez właścicieli interfejsów API w celu zapobiegnięcia przeciążeniu systemu przez ruch sieciowy związany z danymi interfejsu API, a także powiązania wykorzystania interfejsu z konkretnymi wynikami biznesowymi. Istnieją różne podkategorie limitów transferu. Okazuje się, że całe zagadnienie jest ściśle powiązane z ogólnym zarządzaniem ruchem sieciowym. Wszystkie dostępne opcje zostaną omówione w rozdziale 8.

Zanim jednak przejdziemy do szczegółów technicznych, ważne jest wyjaśnienie, jak ograniczenia transferu na poziomie biznesowym, które są określane mianem **przydziałów**, wpływają na projekt interfejsu API.

Przydział to limit transferu, który powiązuje wynik biznesowy z konkretną liczbą transakcji. Celem przydziału jest utrzymanie strategii biznesowej powiązanej z wykorzystaniem interfejsu API. Na przykład w serwisie Twitter projektanci aplikacji mogą sprawdzać swoje osie czasu od 150 do 350 razy w ciągu godziny, zależnie od bieżącego stanu infrastruktury serwisu. Osoby, które przekroczą ten przydział, otrzymają komunikat o błędzie. Częstym zastosowaniem przydziału jest podział projektantów na segmenty, z których każdy ma inny przydział, a tym samym odmienne powiązanie z interfejsem API. Powszechną sytuacją jest na przykład oferowanie dostępu do interfejsu API z niewielkim przydziałem dowolnemu projektantowi, który założy konto. Aby jednak uzyskać wyższy limit transferu, niezbędna będzie dodatkowa weryfikacja, a nawet opłata.

Podobnie jak w przypadku sprzętu i infrastruktury konieczność zastosowania przydziału ma wiele wspólnego z właściwościami skalowania i wykorzystania. Pytania przewodnie brzmią: „Jak wartościowe są moje informacje? Co się stanie, gdy interfejs API odniesie ogromny sukces?”. Na przykład niektóre interfejsy API z treścią są w pełni otwarte, nie żądają żadnego uwierzytelniania użytkowników, dlatego też muszą być chronione przed zbyt dużym ruchem sieciowym. Serwis Twitter wymaga limitu transferu z powodu prawdopodobieństwa tego, że użytkownicy mogą sprawdzać swoje osie czasu 1000 razy na sekundę. Infrastruktura nie jest w stanie przyjąć tak dużego ruchu sieciowego.

Oczywiście możliwe jest zrezygnowanie z przydziałów. Czasami są one stosowane w sposób karny, przez co usługa staje się prawie niezdatna do użycia. Na skutego tego utrudnione staje się testowanie usługi. Ostatecznie projektanci mogą z niej zrezygnować. Zachęcamy do pewnej elastyczności. Bez wątpienia transfery zwiększają się, gdy projektanci testują swoje aplikacje. Mechanizm obciążania opłatą projektantów za transfery większe niż normalnie może ułatwić zarządzanie takimi żądaniami.

Nie zapomnij o tym, że przydziały są też korzystne w przypadku prywatnych interfejsów API (nawet w obszarze chronionym przez korporacyjną zaporę firewall), ponieważ mogą być pomocne w zmniejszaniu ryzyka. Na przykład przedsiębiorstwo może rozważać udostępnienie części swoich „klejnotów koronnych” jako interfejsu API, aby przyspieszyć tworzenie przyszłych produktów biznesowych. Jednakże te „klejnoty” bazują na kosztownej i kluczowej dla firmy infrastrukturze. Wdrażając przydziały, zespół, który jest w posiadaniu interfejsu API, ma możliwość udostępnienia treści lub usługi na potrzeby wprowadzania wewnętrznych innowacji przy jednoczesnym zmniejszaniu ryzyka występowania wewnętrznych problemów oraz ograniczaniu zagrożenia w postaci nieprzyjemnego spotkania z zespołem ds. operacyjnych firmy! Inaczej mówiąc, efektywne użycie przydziałów może pomóc wewnętrznemu zespołowi rozwijającemu interfejs API w zapewnieniu przedsiębiorstwu elastyczności internetu.

Skorowidz

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ż

O'REILLY®

Interfejs API. Strategia programisty

Internet to gigantyczna sieć połączonych ze sobą urządzeń. Jego potencjał wykorzystują wszyscy i każdy jest świadom tego, że pojedyncze urządzenie bez połączenia z siecią nic nie znaczy. Podobnie jest z systemami informatycznymi. Możliwość integracji z siecią, tworzenia rozszerzeń oraz wymiany danych pomaga twórcom aplikacji rozwinąć skrzydła i odnieść sukces na szeroką skalę. Zastanawiasz się, jak otworzyć Twój system na świat? Interfejs API to jedyna droga!

Sięgnij po tę książkę i przekonaj się, jak przygotować wygodny interfejs API, z którego programiści będą korzystali z przyjemnością. Na kolejnych stronach znajdziesz kluczowe zasady projektowania interfejsów API, sposoby zabezpieczania API oraz zarządzania użytkownikami. Ponadto dowiesz się, jak zarządzać ruchem sieciowym, obsługiwać interfejs API oraz mierzyć sukces Twojego API. Na sam koniec zobaczysz, jak zaangażować projektantów w proces adaptacji. Ta książka jest doskonałym źródłem informacji dla wszystkich osób chcących zrozumieć, czym są interfejsy API, jak wykorzystać drzemiący w nich potencjał oraz jak uniknąć typowych zagrożeń i problemów. Twoja lektura obowiązkowa!

Dzięki tej książce:

- poznasz możliwości interfejsów API
- zrozumiesz związaną z nimi strategię biznesową
- unikniesz typowych problemów i zagrożeń
- ustalisz, czy Twój interfejs API odniósł sukces
- zaangażujesz społeczność projektantów w jego rozwój

Wszystko, co powinieneś wiedzieć o interfejsach API!

Daniel Jacobson – dyrektor w Netflix odpowiedzialny za API. Wcześniej był liderem zespołu tworzącego API dla NPR. Jest autorem wielu prezentacji, prowadzi blog.

Greg Brail – obecnie pełni funkcję CTO w firmie Apigee, tworzącej narzędzia do zarządzania interfejsami API. Wcześniej pracował dla TransactPlus przy tworzeniu infrastruktury do przesyłania wiadomości (JMS). Pracował również dla IBM i Citibanku.

Dan Woods – przedsiębiorca, prelegent, autor wielu artykułów związanych z biznesem i branżą IT. Jest autorem lub współautorem ponad 20 książek, a także twórcą systemów informatycznych do publikacji treści oraz pomagających w działalności finansowej.

Helion

33519 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-283-0555-7

9 788328 305557

Informatyka w najlepszym wydaniu

cena 39,90 zł