

UNIWERSYTET GDAŃSKI

Wydział Matematyki, Fizyki i Informatyki

Artur Rybak

nr albumu: 179796

**Projektowanie RESTful API na
potrzeby aplikacji mobilnych**

Praca magisterska na kierunku:

INFORMATYKA

Promotor:

dr Włodzimierz Bzyl

Gdańsk 2015

Streszczenie

Problematyka tej pracy dotyczy schematu komunikacji między urządzeniami mobilnymi a serwerem, jaki narzuca projekt interfejsu w postaci WebAPI. Przeanalizuję konsekwencje, jakie niesie za sobą obranie podejścia RESTful jako wzorca projektowego oraz gdy grupa klientów ograniczona zostanie do aplikacji mobilnych, tj. takich, które uruchamiane są z poziomu urządzeń przenośnych: telefonów, tabletów, zegarków, opasek, odzieży. Kierując się potrzebami oraz wymaganiami wpisanymi w sprzęt, przedstawię oraz zutylizuję w istniejącym projekcie praktyki, powszechnie uważane za właściwe. Słuszność rzeczonych praktyk potwierdzona będzie oszczędnym gospodarowaniem baterią urządzenia, minimalizowaniem zużycia dostępnego pakietu danych czy poprawnym reagowaniem na osiągalność i topologię sieci.

Modelowe przykłady API które oferują firmy takie, jak Facebook, Twitter, Flickr czy Google posłużą jako punkt odniesienia oraz źródło wzorców, którymi powiniśmy się kierować podczas tworzenia własnego WebAPI. Skupię się na najczęstszych problemach i możliwych rozwiązaniach struktury interfejsu, ekstensywnym wykorzystaniu protokołu HTTP oraz powszechnych rozwiązaniach dla sztandarowych problemów takich jak autentykacja, wersjonowanie, rozszerzalność, monitoring czy obsługa błędów.

Wreszcie, zajmę się analizą wdrożonego interfejsu, na kilku przykładach po-

staram się przetransformować API w ten sposób, by intuicyjnie spełniało założenia REST. Przedstawię, jak moja implementacja odbiega od modelowej oraz ocenię czy stuprocentowa utylizacja podejścia RESTful jest zawsze wskazana i wymagana.

Słowa kluczowe

WebAPI, RESTful, aplikacje, mobilne, telefony, komunikacja, wzorce

Spis treści

Wprowadzenie	7
1. Rozumienie WebAPI	10
1.1. Jakie API jest użyteczne?	13
1.2. Praca z Web API	15
1.2.1. Podstawowa struktura składniowa	18
1.2.2. Inspekcja i symulowanie żądań przychodzących i wychodzących	20
1.3. API First	24
2. Projekt WebAPI	29
2.1. Wartość biznesowa	29
2.2. Budowa poprawnego modelu	29
2.2.1. Rzecznowniki - zasoby, kontra czasowniki - akcje	29
2.2.2. Często popełniane błędy	29
2.2.3. Ograniczenia	29
3. Wykonanie WebAPI	30
3.1. Implementacja na wybranej platformie	30

3.2. Składnia zapytań i odpowiedzi	30
3.3. Utylizacja protokołu HTTP	30
3.4. Obsługa błędów	30
3.5. Wersjonowanie i rozszerzalność	30
4. Ewaluacja WebAPI	31
4.1. Porównanie funkcjonalne istniejącego web serwisu i RESTful API	31
4.2. Testowanie	31
4.2.1. Behawioralne	31
4.2.2. Wydajnościowe	31
4.2.3. Narzędzia testowania	31
Zakończenie	32
A. Listingi	33
Bibliografia	34
Spis rysunków	37
Oświadczenie	42

Wprowadzenie

Urządzenia mobilne opanowały sposób w jaki komunikujemy się ze światem. Smartfon jest nam bliższy niż niejeden z członków rodziny. Nie ma w tym jednak nic dziwnego, skoro stał się naszą bramą do poznania, doświadczania i dzielenia się światem i przeżyciami. Mobilność sięga jednak coraz dalej. Telefon, tablet, zegarek, opaska na nadgarstku, okulary, buty, a niedługo pozostałe części naszej garderoby są lub będą częścią świata mobilnego. Niezaprzeczalny jest wpływ telefonu i tabletu na nasz rytm życia, wobec czego, biorąc pod uwagę powszechność rozwiązań mobilnych, postanowiłem przyjrzeć się kluczowej kwestii, jaką jest wydajny, dobrze zaprojektowany interfejs komunikacyjny, pozwalający trzymać w ryzach nawet najbardziej zaawansowany system.

Smartfon - będący naszym osobistym centrum sterowania musi w sposób efektywny porozumiewać się z użytkownikiem oraz światem zewnętrznym. Ubogie byłyby bowiem aplikacje mobilne z których korzystamy gdyby nie pomoc web serwisów. Z jednej strony dostarczają nam one niezbędnych informacji o pogodzie, ruchu drogowym, zmianach cen ropy czy naszych osiągnięciach na siłowni. Z drugiej konsumują to wszystko, co chcemy zachować dla siebie lub czym chcemy pochwalić się przed innymi. Zmieniają sposób w jaki jesteśmy połączeni z ludźmi z najbliższego otoczenia ale i najdalszych zakątków globu.

Intuicyjnie, jako ludzie, potrafimy się porozumiewać. W większości sytuacji wiemy, czy to, co i o czym mówimy kogoś rani, czy bawi. Wiemy, czy dialog z naszym rozmówcą "klei się" w całość, czy raczej jest to nerwowe odbijanie pytań i odpowiedzi. Całą naszą umiejętności i intuicję w tym zakresie opieramy na doświadczeniu. O ile nie jest więc dla nas problemem wysyłanie i odbieranie prostych komunikatów, o tyle zdolność do porozumiewania się między narodami wspólnym językiem biznesowym z zachowaniem wrażliwości na kulturę i poglądy rozmówców jest niezaprzeczalnie wyzwaniem.

Podobnie jest z aplikacjami mobilnymi: o ile łatwo jest z pomocą dostępnych narzędzi wysłać wiadomość w świat a może nawet odebrać odpowiedź na zadane pytanie, o tyle trudniej robić to w sposób wydajny i dobrze ustrukturyzowany. W ciągu ostatnich lat udało się nam jednak wypracować kilka reguł i wytycznych, które czynią komunikację szybką, czytelną, bezpieczną. Reguły, które czynią komunikację wydajną.

Od trzech lat usilnie przyglądam się światu przez pryzmat telefonu i jego możliwości. Przez ten czas dostrzegłem różne próby komunikacji, gdzie jako medium służy smartfon. Pierwotna jego funkcja - możliwość wykonywania połączeń głosowych za pośrednictwem sieci GSM jest dobrze zdefiniowana i wykorzystana ale funkcjonalnie mało rozszerzalna. Krótkie wiadomości tekstowe urozmaicili formę komunikacji ale prawdziwą rewolucję przyniósł tani mobilny internet oraz upo-

wszechnienie się darmowych punktów dostępowych sieci WiFi. Procentowy udział w ruchu internetowym, jaki jest kreowany przez platformy mobilne wzrasta z upływem czasu. Dzieje się to za sprawą coraz doskonalszych aplikacji, przemyślanych interfejsów i wygodzie dostępu do informacji, jaką zyskujemy będąc w podróży czy odpoczywając po ciężkim dniu na kanapie, bo i ta nie jest naturalnym środowiskiem w którym korzystamy z komputera stacjonarnego czy laptopa.

Jako developer aplikacji mobilnych jestem świadomy, że sukces napisanych przeze mnie programów nie ma jednego źródła. Składają się na niego: przemyślany i przyjemny design aplikacji, rozpoznanie, zrozumienie i prawidłowe modelowanie potrzeb użytkownika oraz przypadków użycia. To i tak tylko wierzchołek góry lodowej. Doświadczenie jednak podpowiada, że dobrze zaprojektowana komunikacja klient (aplikacja mobilna) - serwer (w postaci WebAPI) wiele rzeczy upraszcza i pozwala się skupić na problemach funkcjonalnych, zamiast marnotrawić czas na kolejne techniczne *gotchas*.

ROZDZIAŁ 1

Rozumienie WebAPI

Zanim skupimy się na kwestiach technicznych oraz na projektowaniu API, powinniśmy wiedzieć, po co w ogóle chcemy stwarzać możliwość na manipulację naszymi zasobami za pomocą zewnętrznego interfejsu. Pytanie zgoła podstawowe. Istotne jest jednak byśmy wracali do niego często w trakcie projektowania czy implementacji. Łatwo bowiem zgubić ideę którą chcieliśmy się podzielić podczas gdy pochłonie do reszty walka z przeciwnościami technicznymi oraz dylematy związane ze strukturą naszego interfejsu.

Projektowanie API nie jest żadnym wyjątkiem. Podobnie jest z tą pracą magisterską, podobnie z każdym pomysłem na biznes. Cel powinien nie tylko uświetać środki ale i wyznaczać dalszą drogę. Cel powinien być zawsze widoczny.

By zrozumieć jak dobrze projektować API, postarajmy się więc najpierw zrozumieć jakie są wymagania i oczekiwania stawiane przed jego twórcami.

"Wielu chciało GROM rozwiązać. Bo był inny, a w naszym kraju na inność patrzy się podejrzliwie..."

JAKUB NOCH
2 godziny temu

f 76
t
g+
e

– W GROM najistotniejszy nie jest stopień, a to, co kto ma w głowie – mówi naTemat dowódca tej jednostki, ptk Piotr Gaśtał. • Zrzut ekranu z kanalu YouTube.com/GRAK

– W GROM na przykład nie ma typowego wojskowego drylu, stukania obcasami na każdym kroku. Jesteśmy dla siebie partnerami, rozmawiamy ze

ZOBACZ TAKŻE:

- Jak zarabiać na swoich pasjach? Zainwestowała w siebie – odniesta sukces
- Wino i sery idealnym połączeniem? To kulinarny trend, który pokochali Polacy
- Samochody elektryczne przyszłością polskich miast?
- Jak w 3 krokach rozpocząć biznes w internecie?
- Kup bilet i jedź do Sopotu. Na Dancing. I koniecznie zabierz babcię

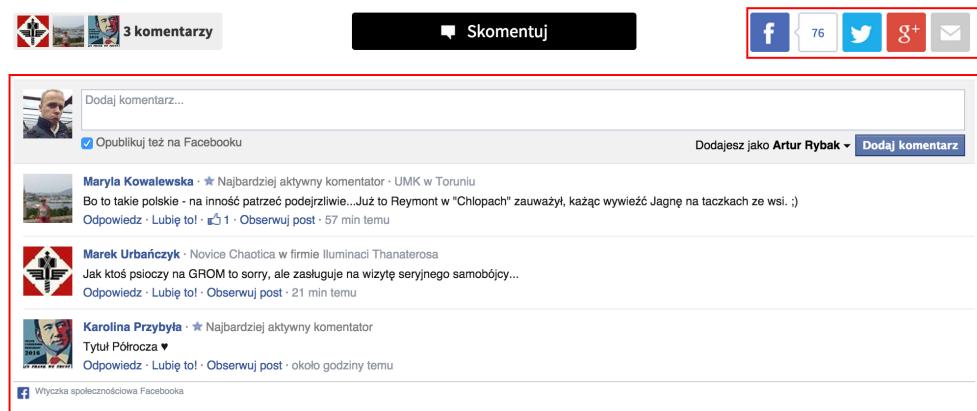
Rysunek 1.1. Strona artykułu w serwisie *na:temat* z przyciskami społecznościowymi

Interpretacja: Jak wiele podobnych stron informacyjnych *na:temat* oferuje przyciski społecznościowe, z których każdy powiązany jest z API serwisu macierzystego, tutaj: *Facebook*, *Twitter*, *Google+*

Źródło: *na:temat* [online], [dostęp: 16 czerwca 2015] dostępny w internecie:

<http://natemat.pl/145775>

W powyższym przykładzie widzimy, że twórcy serwisu *na:temat* umożliwiają swoim czytelnikom rozpowszechnianie treści za pomocą przycisków społecznościowych. Jest to jeden z najprostszych sposobów budowanie zasięgu treści i docieranie do szerokiego grona odbiorców. W ten sposób autor artykułu, czy wydawca stawia na treściwe i wzbudzające emocje artykuły z którymi czytelnik się identyfikuje. Co za tym idzie sam staje się przekaźnikiem, poprzez który serwis dociera do znajomych i obserwatorów swoich dotychczasowych odbiorców.



Rysunek 1.2. Sekcja komentarzy do artykułu w serwisie *na:temat*

Interpretacja: Niektóre z serwisów zrzucają całe funkcjonalności i interakcje z użytkownikami na boki API społecznościowych. Tutaj: wykorzystanie formularza komentarzy zintegrowanego z *Facebookiem*

Źródło: *na:temat* [online], [dostęp: 16 czerwca 2015] dostępny w internecie:

<http://natemat.pl/145775>

na:temat postanowiło oddać w ręce zewnętrznego serwisu całą sekcję komentarzy, która z mozołem powielana jest przez wiele portali. Jak można zaobserwować na rysunku 1.2, serwis zyskuje gotowe rozwiązanie, z całym dobrodziejstwem internetarza ale i ograniczeniami. Zdecydowanie mniej tu komentarzy, które są wulgarnie, opryskliwe i niekulturalne, ponieważ każdy z imienia i nazwiska podpisuje się pod swoimi słowami. Ogranicza to ilość potrzebnej moderacji i przenosi tę odpowiedzialność na Facebooka. Minusem jest mała elastyczność wyglądu tej sekcji i brak kontroli oraz niechciane treści reklamowe, które Facebook może dołączać do swojego produktu.

1.1. Jakie API jest użyteczne?

Dobrze zaprojektowane API pozwala deweloperom na tworzenie aplikacji *manashup*, tj. takich, które łączą w sobie cechy kilku serwisów. Z pewnością istnieje uzasadnienie dla istnienia aplikacji, oferujących funkcjonalność specyfczną dla ich dziedziny. Nie potrzebują one elastyczności i gdyby porównać je do garniturów, to mielibyśmy do czynienia jedynie z sztywnymi na miarę. Takie aplikacje mogą bez problemów komunikować się z interfejsami bazującymi na akcjach lub za pomocą SOAP API. Pewną nadmiarowością byłoby także tworzenie generycznego serwisu tylko na potrzeby jednej aplikacji. Niesie to za sobą wymierne korzyści, gdy budujemy system mocno powiązany. Pozwala *ścinac zakręty* i skracac proces powstawania aplikacji.

Jeśli jednak chcemy pobudzić społeczność i dać jej narzędzie, które będzie mogła KREATYWNIĘ wykorzystać, to być może wystawienie REST API jest dla nas najlepszym wyborem. Do najbardziej rozchwytywanych API należą te, oferowane przez serwisy społecznościowe takie, jak *Facebook*, *Twitter* czy *LinkedIn*. Deweloper, który otrzymuje takie narzędzie, ma natychmiast dostęp do danych o milionach a nawet miliardach użytkowników. Oczywiście „za darmo” otrzymujemy jedynie dane publiczne, tj. takie, które użytkownik sam zgodził się udostępniać zarówno i wszystkim. Istnieje jednak kilka sposobów, by poprosić użytkownika o bardziej newralgiczne, dokładniejsze informacje. Wiąże się to z reguły z autentykacją

(m.in OAuth o którym w kolejnych rozdziałach) i założeniem konta w serwisie, który będzie potrafił jednoznacznie określić, która aplikacja odpowiada za aktywność w serwisie. Przydaje się to szczególnie wtedy, gdy twórcy aplikacji zaczynają obchodzić regulamin lub wprost go lekceważyć i wykorzystują naiwność internautów, ich słabości, ciekawość tylko po to by przechwycić cenne informacje, pieniądze lub zaatakować kolejną, spersonalizowaną dawką spamu. Twórcy API, którym zależy na budowaniu pozytywnego wizerunku swojego produktu mogą wówczas takie wystąpienia szybko ukröcić, wyłączając dostęp do API dla konkretnych aplikacji. Bywa, że w przypadku bardzo obieganych serwisów jest to proces zautomatyzowany, sterowany algorytmami z dziedziny sztucznej inteligencji.

MOST POPULAR APIs

1. Facebook	Track this API	6. LinkedIn	Track this API
2. Google Maps	Track this API	7. Kayak	Track this API
3. Skype	Track this API	8. Waze	Track this API
4. Netflix	Track this API	9. Yahoo Weather	Track this API
5. Telegram	Track this API	10. Pinterest	Track this API

Rysunek 1.3. Ranking najpopularniejszych API z których korzystają deweloperzy.

Interpretacja: Najpopularniejsze serwisy webowe oferują najbardziej rozchwytywane API. Zarówno ze względu na bogatą treść jaką za ich pomocą „wyciągnąć” ale i środki jakie czołowi gracze inwestują w rozwój swoich platform. Owocuje to przemyślanym i responsywnym interfejsem a także znaczną penetracją rynku.

Źródło: ProgrammableWeb [online], [dostęp: 16 czerwca 2015] dostępny w internecie:
<http://www.programmableweb.com/apis>

1.2. Praca z Web API

Praca z Web API wymaga od programisty zrozumienia platformy: jej struktury danych, sposobu interakcji, kroków wymaganych do wydobycia lub przetworzenia interesujących go w danym momencie informacji. Niezbędne jest zapewnienie w takim wypadku miejsca, gdzie zagubiony programista będzie mógł usystematyzować swoją wiedzę o możliwościach, ograniczeniach, wymaganiach i najprostszych sposobach korzystania z danego API. Wszelakie narzędzia, przykładowy kod, tutorial, samouczki są nieodzowną pomocą w takich przypadkach. Często, w przypadku raczy wielkości Facebooka strony deweloperskie urastają do całkiem sporego rozmiaru, tworząc cały ekosystem. Społeczność, wśród której możemy bez skrępowania zadawać pytania i poszukiwać odpowiedzi.

Rysunek 1.4 przedstawia narzędzie deweloperskie stworzone przez Twilio. Jest to firma, która zajmuje się dostarczaniem API dla rozmów telefonicznych i SMSów. Celem Twilio jest, by każdy, kto odwiedza stronę główną mógł w ciągu 5 minut wykonać testową rozmowę lub wysłać smsa za pomocą oferowanego API. To doskonaly sposób na zaangażowanie potencjalnych klientów i deweloperów, którzy jak się okazuje - z niezwykłą łatwością mogą korzystać z dobrodziejstw SMSów i rozmów głosowych w swoich produktach. Być może także początkowy plan by tylko 5-10 minut rozejrzeć się na tronie Twilio zaowocuje w programiście chęcią

zainwestowania czasu, który pozwoli mu na poznanie i zrozumienie „co oni mogą dla mnie zrobić”.

Request

Curl Ruby PHP Python Node.js Java C#

Note: Toggle showing your **Auth Token** by [clicking here](#)

[Check out the curl manpage](#)

```
curl -X POST 'https://api.twilio.com/2010-04-01/Accounts/ACb92118e87631a105a67f326a74d508ff/Messages.json' \
--data-urlencode 'To=+48[REDACTED]8' \
--data-urlencode 'From=+48[REDACTED]8' \
--data-urlencode 'Body=Testing API for master thesis' \
-u ACb92118e87631a105a67f326a74d508ff:[AuthToken]
```

[Make Request](#)

Note: This request costs money. You can find pricing information [here](#)

Rysunek 1.4. Podgląd pierwszego testowego żądania z Twilio API

Interpretacja: Twilio umożliwia „wyklikanie” i wypełnienie formularza online, który zostaje „w locie” przetłumaczony na żądanie. Możemy zobaczyć przykładową implementację w kilku najbardziej popularnych językach programowania a także za pomocą programu *curl*.

Źródło: własne

Mnie zajęło 7 minut wysłanie pierwszego SMSa, za pomocą Twilio API. I choć może to poniżej celu jaki sobie ta firma stawia, to z pewnością efekt został osiągnięty. W głowie zakiełkowało już co najmniej 10 pomysłów, jak takie API można wykorzystać i gdzie mógłbym je wpleść do moich aplikacji. Rysunek 1.5 jest dowodem na to, że można. Ponadto w informacji zwróconej otrzymujemy kod informujący o utworzonej encji (**201**) **CREATED** oraz samą encję w postaci JSONa.

Response 201

CREATED - The request was successful. We created a new resource and the response body contains the representation.

```
{  
    "sid": "SM1b2e48d4cca24833949d1a0cfb85c654",  
    "date_created": "Mon, 24 Aug 2015 13:40:19 +0000",  
    "date_updated": "Mon, 24 Aug 2015 13:40:19 +0000",  
    "date_sent": null,  
    "account_sid": "ACb92118e87631a105a67f326a74d508ff",  
    "to": "+485 [REDACTED] 8",  
    "from": "+487 [REDACTED]",  
    "body": "Testing API for master thesis",  
    "status": "queued",  
    "num_segments": "1",  
    "num_media": "0",  
    "direction": "outbound-api",  
    "api_version": "2010-04-01",  
    "price": null,  
    "price_unit": "USD",  
    "error_code": null,  
    "error_message": null,  
    "uri": "/2010-04-  
01/Accounts/ACb92118e87631a105a67f326a74d508ff/Messages/SM1b2e48d4cca24833949d1a0cfb85c654.json",  
    "subresource_uris": {  
        "media": "/2010-04-  
01/Accounts/ACb92118e87631a105a67f326a74d508ff/Messages/SM1b2e48d4cca24833949d1a0cfb85c654/Media.js  
on"  
    }  
}
```

Rysunek 1.5. Odpowiedź z API Twilio, która potwierdza wysłanie testowego SMSa

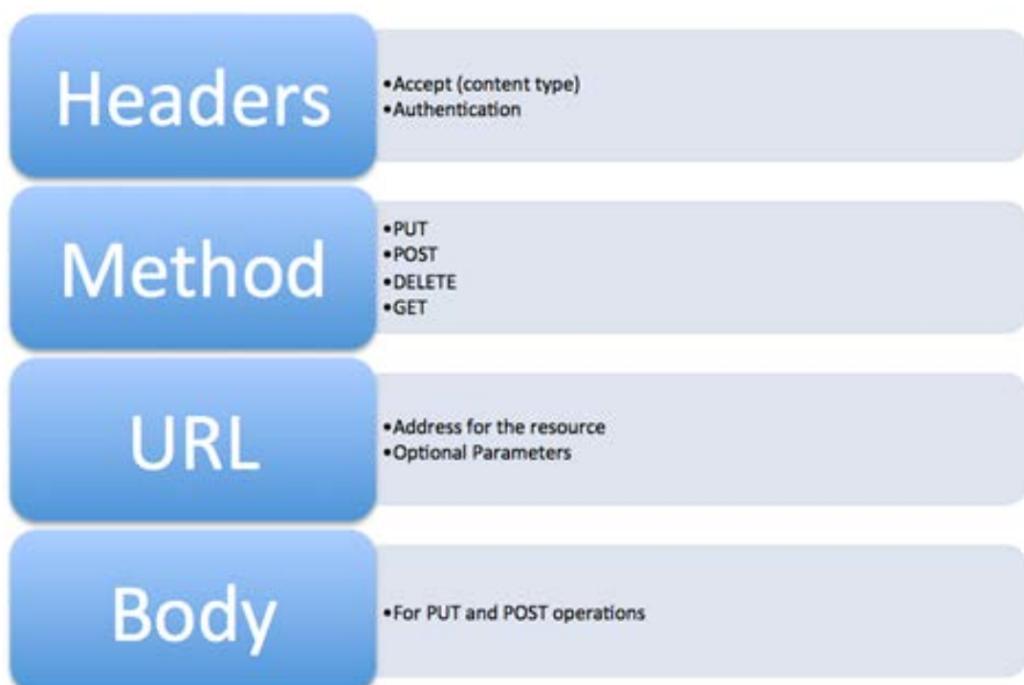
Interpretacja: W odpowiedzi widzimy, czego możemy się spodziewać w zwrocie z Twilio API. Warto zwrócić uwagę na to, że nawet bez zagłębiania się w strukturę łatwo domniemać znaczenie poszczególnych pól.

Źródło: własne

Tak łatwe wprowadzenie jak w przypadku Twilio, generuje wymierną wartość marketingową. Deweloper, który w pozostałych sytuacjach zapytałby „Dlaczego ma mnie to obchodzić?”, co sprowadza się do dwóch innych pytań: „Co mogę

Z TYM zrobić” i „Jak mogę TO zrobić?” w tym momencie zobaczy nie kolejne przeszkody a klocki, budulec jego przyszłych aplikacji.

1.2.1. Podstawowa struktura składniowa



Rysunek 1.6. Elementy składowe żądania HTTP

Interpretacja: Podstawowa struktura żądania zawiera informację o nagłówkach, metodzie, URLu (adresie) oraz ciele.

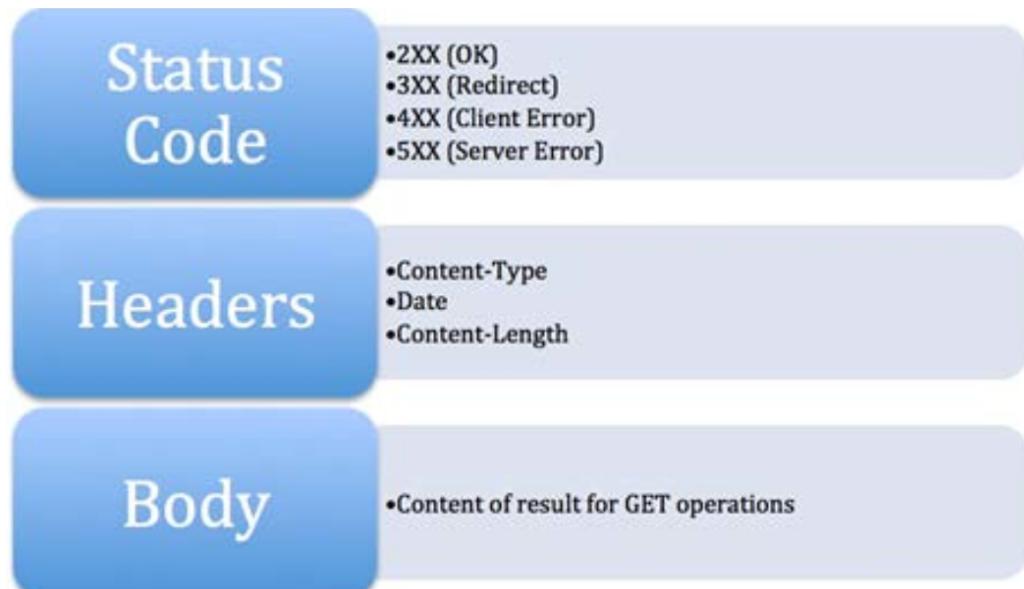
Źródło: [1], wersja IV, s. 27

Podczas gdy mówiąc o API możemy mieć na uwadze skomplikowane systemy, Web API są interfejsami tak prostymi jak to tylko możliwe. Podstawowymi akcjami przezeń oferowanymi są akcje **CRUD** - Create Read Update Delete. Ponadto

interfejsy Web API utylizują cechy protokołu HTTP a co za tym idzie spodziewają się, że żądanie, tak jak na rysunku 1.6 będzie zawierało informacje o adresie, nagłówkach, metodzie oraz w przypadku metod PUT/PATCH/POST ciało żądania. Oczywiście jest to tylko konwencja i nikt nie zabroni nam obsłużyć żądań na własną rękę w zupełnie dowolny sposób. Należy jednak pamiętać, że jest to niezgodne z intuicją i powszechną praktyką. Co za tym idzie utrudnia zrozumienie intencji naszego API i podwyższa „próg wejścia” potrzebny do tego by z API korzystać. Największe zdziwienie może tutaj budzić odniesienie do metody PATCH, która nie była pierwotnie częścią standardu HTTP, została jednak wprowadzona dokumentem RFC 5789 [2] w marcu 2010 roku. Szczególnie ważna ze względu na platformy mobilne, których zadaniem jest minimalizacja transferu.

Postępowanie według wzorca pozwala deweloperom poczynić pewne założenia. Jak choćby te, że metoda POST stworzy nowy zasób, PUT uaktualni już istniejący, PATCH uaktualni niektóre własności/wartości zasobu, GET dokona odczytu a DELETE usunie go całkowicie. Programista będzie również domniemywać unicjalność zasobu występującego pod danym adresem.

Założenia co do wartości zwracanych również będą bazować na protokole HTTP, tj. na kodzie zwracanym, nagłówkach oraz ciele odpowiedzi tak jak obrazuje to rysunek 1.7.



Rysunek 1.7. Element składowe odpowiedzi na żądanie HTTP

Interpretacja: Ustrukturyzowana odpowiedź na żądanie HTTP zawiera kod HTTP informujący o statusie, nagłówki oraz w niektórych przypadkach ciało.

Źródło: [1], wersja IV, s. 28

1.2.2. Inspekcja i symulowanie żądań przychodzących i wychodzących

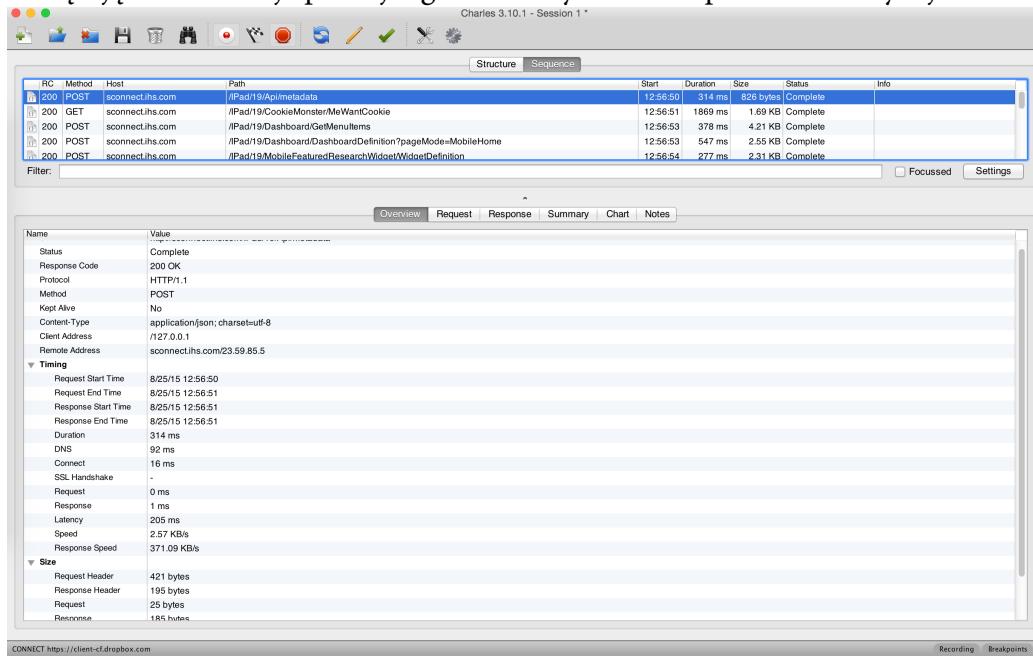
Codzienna praca z API, nawet jeśli nie jest ono przez nas rozwijane wymaga użycia narzędzi, które pozwolą na podglądarkanie żądań i odpowiedzi wraz z wszelkimi metadanymi. Każdy ma swoje ulubione narzędzia, jednak w tej pracy pozwolę sobie korzystać z takiego samego zestawu narzędzi, jak w przypadku mojej codziennej pracy nad aplikacjami mobilnymi.

Po pierwsze Charles [3]. Jest to proxy, poprzez które przechodzą wszelkie żądania z lokalnej maszyny a także wszelkie żądania urządzeń, symulatorów i emulatorów na których ustawiliśmy proxy w ten sposób, by wskazywało na maszynę z zainstalowanym Charlesem. Domyślnie program instaluje się na porcie 8888 a cechuje go prosty i elegancki wygląd a także duże możliwości konfiguracji: ograniczanie śledzonych żądań do wskazanych domen, możliwość symulowania niewydajnego łącza internetowego, stawianie breakpointów zarówno na żądaniach jak na odpowiedziach oraz ich modyfikacja „w locie”, przepisywanie żądań i odpowiedzi według ustalonych reguł a także bardzo przydatne mapowanie lokalne lub zdalne, polegające na odsyłaniu do wybranego pliku lub strony w przypadku spełnienia kryteriów. W praktyce pozwala to pracować z niestabilnymi środowiskami deweloperskimi poprzez przekierowywanie niektórych żądań do środowisk stabilniejszych. Naturalnie wszystko wymaga rozwagi i rozumnego wykorzystania, gdyż narzędzie może okazać się mieczem obosiecznym i doprowadzić do niestabilnego działania zarówno aplikacji klienckiej jak backendu. Alternatywami dla Charles'a mogą być *HTTPScoop* [4], *Wireshark* [5] czy *Fiddler* [6]

Na rysunku 1.8 widzimy przykładowy obraz jaki uzyskuje programista, który chce zbadać jakie są parametry żądań wysyłanych przez aplikację, oraz wszelkie detale dotyczące struktury i czasu wykonania. Umożliwia to łatwe wychwytywanie błędów, literówek, nadmiarowości a także jest doskonałym narzędziem do profi-

lowania czasów oczekiwania na odpowiedź. Dzięki temu widzimy jakie żądania są

obciążające dla naszej aplikacji i gdzie należy szukać usprawnień w wydajności.



Rysunek 1.8. Charles podczas pracy z jednym z API

Interpretacja: W górnej części znajdują się chronologicznie wykonywane żądania, które można zaznaczyć, by u dołu otrzymać szczegóły. Sposób prezentacji w poszczególnych zakładkach pozwala na czytelny, ukierunkowany obraz, gdzie skupić możemy się na parametrach żądania, parametrach odpowiedzi, nagłówkach, sposobie autentakcji, ciasteczkach, wykresach związanych z długością trwania poszczególnych żądań.

Źródło: własne

Po drugie cURL [7] i Apache™ JMeter. Funkcja, której najbardziej brakuje mi w Charlesie to możliwość łatwego komponowania żądań - ustawiania ich parametrów, zapisywanie, tworzenie scenariuszy. Tutaj moim wyborem do prosty rozwiązań jest zdecydowanie wbudowany w każdy uniksowopochodny system *cURL*. Wzięty z życia przykład żądania wygląda jak na listingu 1. Przy czym parametry

`$1 i $2` to login i hasło, które z powodu wymaganej poufności zostały ukryte. W zwrocie otrzymujemy json'a wraz informacjami statusie, co obrazuje rysunek 1.9.

Listing 1. curl - podstawowe użycie

```
curl -v --basic --user $1:$2
→ sconnect.ihs.com/IPad/19/Dashboard/DashboardDefinition?pageMode=MobileHome
```

Konstrukcja tego szczególnego żądania jak i całego API, którego używam by najmniej nie należy do wzorów i nie powinna być naśladowana. W dalszych rozdziałach dowiesz się dlaczego.

```
curl -v --basic --user $1:$2
→ sconnect.ihs.com/IPad/19/Dashboard/DashboardDefinition?pageMode=MobileHome
```

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
X-Powered-By: ASP.NET
Date: Tue, 25 Aug 2015 18:09:47 GMT
Content-Length: 4260
Connection: keep-alive
Set-Cookie: IHS_SSO_SESS=0sx2FBqCBAUCA0jZMKMSzypGoyYbsRX-cx2dR0x2Bpx4x2B918x3DroPw14dBICG5j0n5J3E30x3Dx30-ZT4bQpF1WGNZAmCrAglwx3Dx3D-TBuJy8SQ19imJHQ1ZFQkAx3Dx3D; domain=.ihs.com; path=/; HttpOnly
Set-Cookie: IHS_CONNECT_SESS_STAGE=xhFR0qPmG09H0Qdp9eXA03a3f3d2c2110rKkDsJHJwIRScSm382ba1LxxH105w21082b52b5WTXN38gcrDKN%sb59HRd6jZ2np6z2bNDTCQc1qpmZ3M2wZo0wEMwRdz52p2T7P2b00%2b9bzoIUT0c881D2r2Wor11hpMBpdtLppwf6JmVPvxz2b9zFhsSwEv18VmnpV2ba1gfca1a1qJN02zpxhUGRoFE152h69JPKPGKs5wdkR6nrbrxG0sozKEpDFmIW0oX1VhZUmdb8p8d2gtW3dn0frAkWrIDvBMPtZPostSUK35Nszbv1Wwj0134jW1xLFNKR1jHjBu0PGCPgSf0W1W4Zc0l3y0v0j4E093fkqZ0mpeBHP1KpkpWj0h0eeZE0vP30kR140jEj52bvehosut7NP1AFJxBuikMvWtPV3B1k1M8SbsJ0fUr9dxecdU0%2FLJf12%2buaQUXHZ3MRYw6CT1W3Fw&MPd7nmII1tqY2FFJGrhliiwpqs3d; domain=.ihs.com; expires=Tue, 25-Aug-2015 18:39:46 GMT; path=/; HttpOnly
<
{"data": {"ToxonomyDefinition": {"ToxonomyTypeMobileDefinitions": [{"Dependent": null, "Description": null, "Min": null, "Max": null, "Name": "Domain", "PresentationStyle": "List", "ToxonomyKey": "d", "Final": false, "ExcludeFromCurrentSelectionDescription": true, "ExcludeFromInitialMessage": true}, {"Dependent": null, "Description": null, "Min": null, "Max": null, "Name": "Region", "PresentationStyle": "List", "ToxonomyKey": "r", "Final": false, "ExcludeFromCurrentSelectionDescription": true, "ExcludeFromInitialMessage": true}], "SelectEdToxonomy": {"d": [{"Id": "All", "Name": "All"}]}, "EntitledToxonomy": null, "WebToxonomyType": "MobileSectorAndRegion", "Final": false}, "PageMode": "MobileHome", "Name": null, "FoldCardId": null, "SavedDashboardId": null, "SubMenuItemDefinition": null, "AvailableWidgets": [{"Id": "MobileFeaturedResearchWidget", "Title": "Featured Research", "IsOpen": true, "IsTemplated": false, "ShowTitle": true, "BorderMargin": false}, {"Id": "MobileHeadlinePerspectiveWidget", "Title": "Headline Perspective", "IsOpen": true, "IsTemplated": true, "ShowTitle": true, "BorderMargin": true}, {"Id": "MobileHotTopicsWidget", "Title": "Hot Topics", "IsOpen": true, "IsTemplated": false, "ShowTitle": true, "BorderMargin": true}, {"Id": "MobileLatestHeadlineWidget", "Title": "Headline Analysis", "IsOpen": true, "IsTemplated": false, "ShowTitle": true, "BorderMargin": true}, {"Id": "MobileMostPopularWidget", "Title": "Most Popular on Connect", "IsOpen": true, "IsTemplated": true, "ShowTitle": true, "BorderMargin": true}], "LayoutDefinition": [{"Id": "A", "Children": [{"Id": "A1", "Children": [{"Id": "A11", "Children": []}], "SplitRatios": [100], "HorizontalSplit": true, "WidgetId": "MobileHeadlinePerspectiveWidget"}, {"Id": "A12", "Children": [], "SplitRatios": [100], "HorizontalSplit": true, "WidgetId": "MobileLatestHeadlineWidget"}], "SplitRatios": [60, 40], "HorizontalSplit": true}], "SplitRatios": [60, 40], "HorizontalSplit": true}}

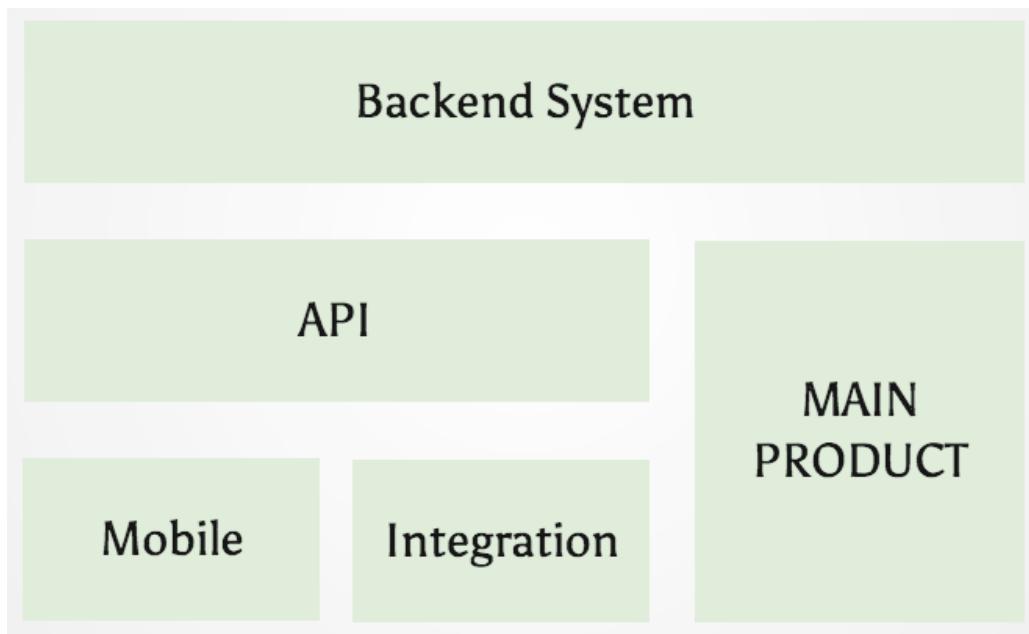
Rysunek 1.9. Wynik działania programu z listingu 1.

Interpretacja: W odpowiedzi otrzymaliśmy ciało w postaci json'a oraz informacje o statusie.

Źródło: własne

1.3. API First

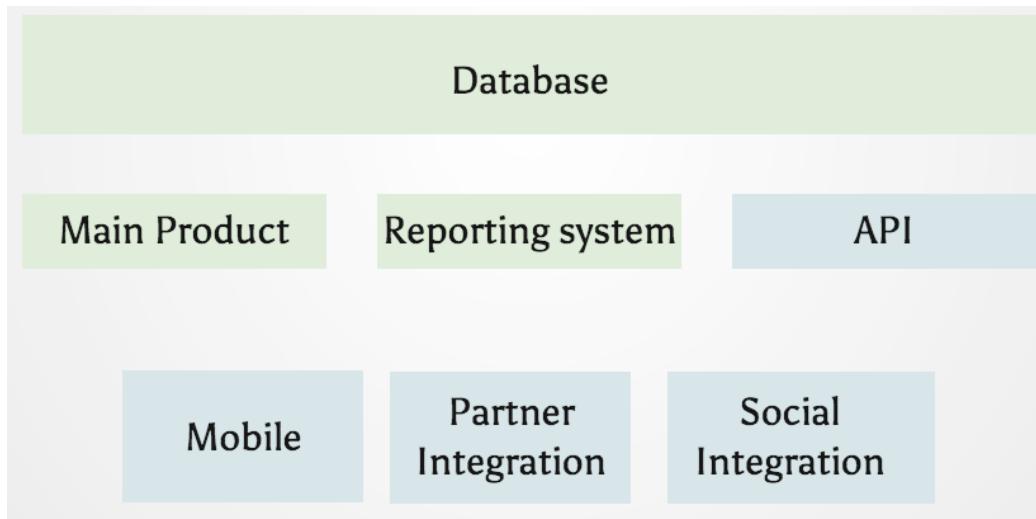
API first to strategia planowania całej linii produktów danej firmy, gdzie API są podstawą każdego produktu w przeciwieństwie do bycia osobnym, pobocznym produktem. By zrozumieć, dlaczego *API first* jest dobrym pomysłem, najpierw trzeba zrozumieć dlaczego API są tworzone w ogóle. Istnieje kilka modeli tworzenia interfejsów, jednak generalizując API serwuje to, co otrzyma z systemu stacjonującego backend i komunikuje się z nim bezpośrednio, równolegle z innymi produktami. Wynika z tego, że jeśli chcemy stworzyć kolejne aplikacje, musimy napisać więcej systemów, które będą bezpośrednio dotykać backendu ALBO rozszerzyć API w ten sposób, by wspierało oba alternatywne produkty. Co więcej, API jest często uważane za „wabik” lub „czynnik wyróżniający”, który pozwala zwabić klientów, jednak traktowane jest jako *nice to have* zamiast być ważnym składnikiem ekosystemu produktów w firmie. Takie podejście generuje poważne problemy, szczególnie uwzględniając zasoby, jako, że konkuruje z produktami przynoszącymi zauważalne przychody.



Rysunek 1.10. API jako produkt poboczny, podejście standardowe

Interpretacja: API jest osobną ścieżką w linii produktów firmy i używane jest do zapewnienia działania jednego lub kilku integracji, czasem klientów mobilnych.

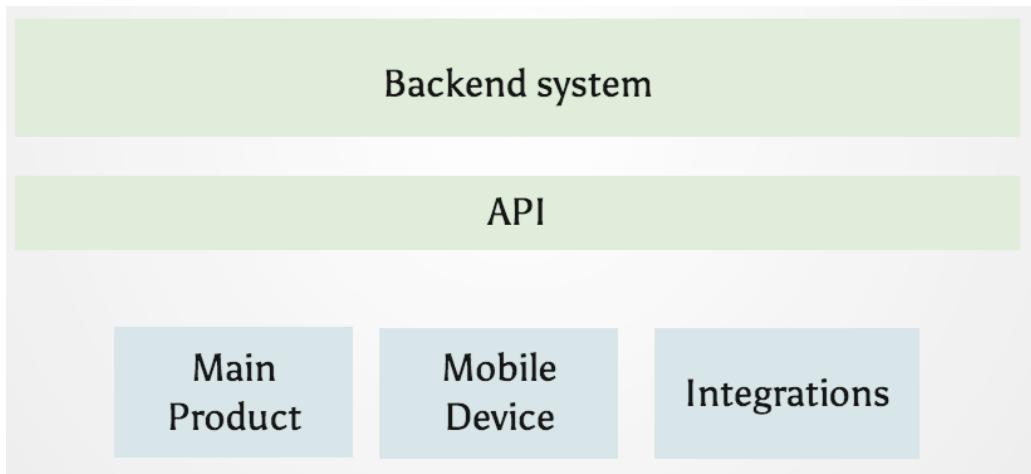
Źródło: własne



Rysunek 1.11. API jako jeden z wielu interfejsów

Interpretacja: API jest tylko jednym z interfejsów, jakie komunikują się z backendem i choć ma swoich klientów, to równolegle działają systemy komunikujące się z backendem w podobny lub wręcz zupełnie odmienny sposób.

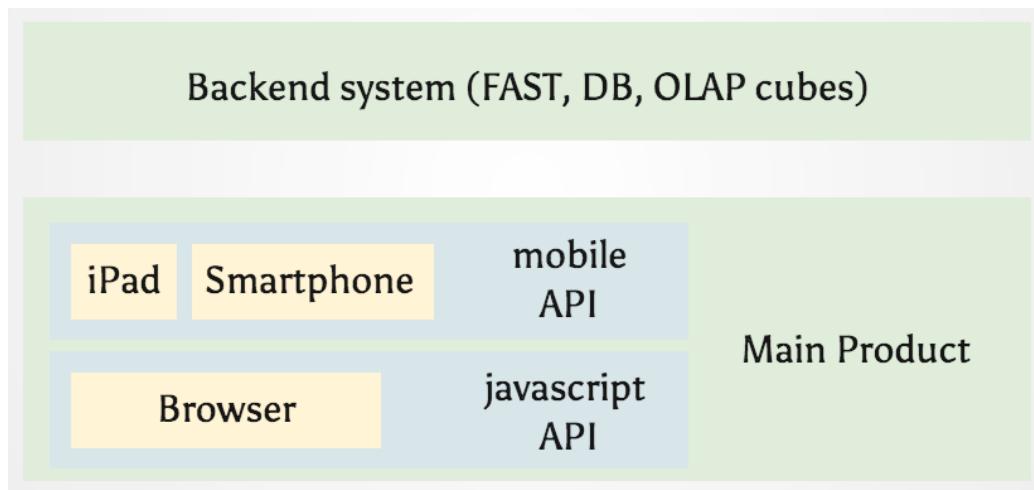
Źródło: własne



Rysunek 1.12. API first

Interpretacja: API jest pośrednikiem pomiędzy wszystkimi aplikacjami klienckimi a backendem, stanowi warstwę bez której komunikacja między tymi systemami nie może mieć miejsca.

Źródło: własne



Rysunek 1.13. API z mojej perspektywy

Interpretacja: API widziane z mojej perspektywy. Zaimplementowane jako *sideproduct* w strategii firmy. Służy do komunikacji backendu i platform mobilnych, przy czym samo nie jest osobnym byteam a wrasta w główny produkt - aplikację webową.

Źródło: własne

ROZDZIAŁ 2

Projekt WebAPI

2.1. Wartość biznesowa

2.2. Budowa poprawnego modelu

2.2.1. Rzecznowniki - zasoby, kontra czasowniki - akcje

2.2.2. Często popełniane błędy

2.2.3. Ograniczenia

ROZDZIAŁ 3

Wykonanie WebAPI

3.1. Implementacja na wybranej platformie

3.2. Składnia zapytań i odpowiedzi

3.3. Utylizacja protokołu HTTP

3.4. Obsługa błędów

3.5. Wersjonowanie i rozszerzalność

ROZDZIAŁ 4

Ewaluacja WebAPI

4.1. Porównanie funkcjonalne istniejącego web serwisu i RESTful API

4.2. Testowanie

4.2.1. Behawioralne

4.2.2. Wydajnościowe

4.2.3. Narzędzia testowania

Zakończenie

Podsumowanie

DODATEK A

Listingi

Bibliografia

- [1] Kirsten L. Hunter. *Irresistible APIs. Create Web APIs that developers will love.* Manning Publications, 2015.
- [2] Patch method for http [online], [dostęp: 25 sierpnia 2015] dostępny w internecie: <http://tools.ietf.org/html/rfc5789>.
- [3] Charles proxy [online], [dostęp: 25 sierpnia 2015] dostępny w internecie: <https://www.tuffcode.com/>.
- [4] Httpscoop [online], [dostęp: 25 sierpnia 2015] dostępny w internecie: <https://www.wireshark.org/>.
- [5] Wireshark [online], [dostęp: 25 sierpnia 2015] dostępny w internecie: <https://www.wireshark.org/>.
- [6] Fiddler [online], [dostęp: 25 sierpnia 2015] dostępny w internecie: <http://www.telerik.com/fiddler>.
- [7] curl [online], [dostęp: 25 sierpnia 2015] dostępny w internecie: <http://curl.haxx.se/>.
- [8] Kore.io [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <http://kore.io/>.

- [9] Best practices for designing a pragmatic restful api [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>.
- [10] Chen-Che Huang, Jiun-Long Huang, Chin-Liang Tsai, Guan-Zhong Wu, Chia-Min Chen, and Wang-Chien Lee. Energy-efficient and cost-effective web api invocations with transfer size reduction for mobile mashup applications. *Wireless Networks*, 20(3):361–378, April 2014.
- [11] How to design rest apis for mobile? [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <http://www.redotheweb.com/2012/08/09/how-to-design-rest-apis-for-mobile.html>.
- [12] Efficient mobile apis using apache thrift [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <http://blog.whitepages.com/reducing-data-with-apache-thrift/>.
- [13] Rest and the promise of secure and efficient application delivery [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <https://blog.akana.com/rest-and-the-promise-of-secure-and-efficient-application-delivery/>.
- [14] Creating an efficient rest api with http [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <http://mark-kirby.co.uk/2013/creating-a-true-rest-api/>.

- [15] How to handle challenges with api security and efficiency [online], [dostęp: 15 czerwca 2015] dostępny w internecie: <http://searchsoa.techtarget.com/feature/How-to-handle-challenges-with-API-security-and-efficiency>.
- [16] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly, 2013.
- [17] Walter Binder. *Designing and Implementing a Secure, Portable, and Efficient Mobile Agent Kernel: The J-SEAL2 Approach*. PhD thesis, der Technischen Universität Wien, April 2001.
- [18] How we moved our api from ruby to go and saved our sanity [online], [dostęp: 18 czerwca 2015] dostępny w internecie: <http://blog.parse.com/learn/how-we-moved-our-api-from-ruby-to-go-and-saved-our-sanity/>.

Spis rysunków

1.1. Strona artykułu w serwisie *na:temat* z przyciskami społecznościowymi

Interpretacja: Jak wiele podobnych stron informacyjnych *na:temat* oferuje przyciski społecznościowe, z których każdy powiązany jest z API serwisu macierzystego, tutaj: *Facebook*, *Twitter*, *Google+*

Źródło: na:temat [online], [dostęp: 16 czerwca 2015] dostępny w internecie: <http://natemat.pl/145775> 11

1.2. Sekcja komentarzy do artykułu w serwisie *na:temat*

Interpretacja: Niektóre z serwisów zrzucają całe funkcjonalności i interakcje z użytkownikami na baki API społecznościowych. Tużaj: wykorzystanie formularza komentarzy zintegrowanego z *Facebookiem*

Źródło: na:temat [online], [dostęp: 16 czerwca 2015] dostępny w internecie: <http://natemat.pl/145775> 12

1.3. Ranking najpopularniejszych API z których korzystają deweloperzy.

Interpretacja: Najpopularniejsze serwisy webowe oferują najbardziej rozchwytywane API. Zarówno ze względu na bogatą treść jaką za ich pomocą „wyciągnąć” ale i środki jakie czołowi gracze investują w rozwój swoich platform. Owocuje to przemyślanym i responsywnym interfejsem a także znaczną penetracją rynku.

Źródło: ProgrammableWeb [online], [dostęp: 16 czerwca 2015]
dostępny w internecie: <http://www.programmableweb.com/apis>

14

1.4. Podgląd pierwszego testowego żądania z Twilio API

Interpretacja: Twilio umożliwia „wyklikanie” i wypełnienie formularza online, który zostaje „w locie” przetłumaczony na żądanie. Możemy zobaczyć przykładową implementację w kilku najbardziej popularnych językach programowania a także za pomocą programu *curl*.

Źródło: własne 16

- 1.5. Odpowiedź z API Twilio, która potwierdza wysłanie testowego SMSa

Interpretacja: W odpowiedzi widzimy, czego możemy się spodziewać w zwrocie z Twilio API. Warto zwrócić uwagę na to, że nawet bez zagłębiania się w strukturę łatwo domniemać znaczenie poszczególnych pól.

Źródło: własne 17

- 1.6. Elementy składowe żądania HTTP

Interpretacja: Podstawowa struktura żądania zawiera informację o nagłówkach, metodzie, URLu (adresie) oraz ciele.

Źródło: [1], wersja IV, s. 27 18

- 1.7. Element składowe odpowiedzi na żądanie HTTP

Interpretacja: Ustrukturyzowana odpowiedź na żądanie HTTP zawiera kod HTTP informujący o statusie, nagłówki oraz w niektórych przypadkach ciało.

Źródło: [1], wersja IV, s. 28 20

1.8. Charles podczas pracy z jednym z API

Interpretacja: W górnej części znajdują się chronologicznie wykonywane żądania, które można zaznaczyć, by u dołu otrzymać szczegółowe. Sposób prezentacji w poszczególnych zakładkach pozwala na czytelny, ukierunkowany obraz, gdzie skupić możemy się na parametrach żądania, parametrach odpowiedzi, nagłówkach, sposobie autentakcji, ciasteczkach, wykresach związanych z dłużością trwania poszczególnych żądań.

Źródło: własne 22

1.9. Wynik działania programu z listingu 1.

Interpretacja: W odpowiedzi otrzymaliśmy ciało w postaci json'a oraz informacje o statusie.

Źródło: własne 23

1.10. API jako produkt poboczny, podejście standardowe

Interpretacja: API jest osobną ścieżką w linii produktów firmy i używane jest do zapewnienia działania jednego lub kilku integracji, czasem klientów mobilnych.

Źródło: własne 25

1.11. API jako jeden z wielu interfejsów

Interpretacja: API jest tylko jednym z interfejsów, jakie komunikują się z backendem i choć ma swoich klientów, to równolegle działają systemy komunikujące się z backendem w podobny lub wręcz zupełnie odmienny sposób.

Źródło: własne 26

1.12. API first

Interpretacja: API jest pośrednikiem pomiędzy wszystkimi aplikacjami klienckimi a backendem, stanowi warstwę bez której komunikacja między tymi systemami nie może mieć miejsca.

Źródło: własne 27

1.13. API z mojej perspektywy

Interpretacja: API widziane z mojej perspektywy. Zaimplementowane jako *sideproduct* w strategii firmy. Służy do komunikacji backendu i platform mobilnych, przy czym samo nie jest osobnym byteam a wrasta w główny produkt - aplikację webową.

Źródło: własne 28

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....

data

podpis