Internet rzeczy

Budowa sieci z wykorzystaniem technologii webowych i Raspberry Pi

Dominique D. Guinard
Vlad M. Trifa

Helion
MANNING

Tytuł oryginału: Building the Web of Things: With examples in Node.js and Raspberry Pi

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-2968-3

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/intrze.zip

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

Printed in Poland.

Spis treści

2 Witaj, świecie WWW rzeczy!

Zawartość rozdziału:

- Szybki rzut oka na różne poziomy architektury WWW rzeczy.
- Uzyskiwanie dostępu do urządzeń przy użyciu protokołu HTTP, adresów URL, technologii WebSocket oraz przeglądarek WWW.
- Stosowanie API typu REST do pobierania danych w formacie JSON.
- Poznawanie zapisu webowej semantyki.
- Tworzenie pierwszej fizycznej aplikacji typu *mashup*.

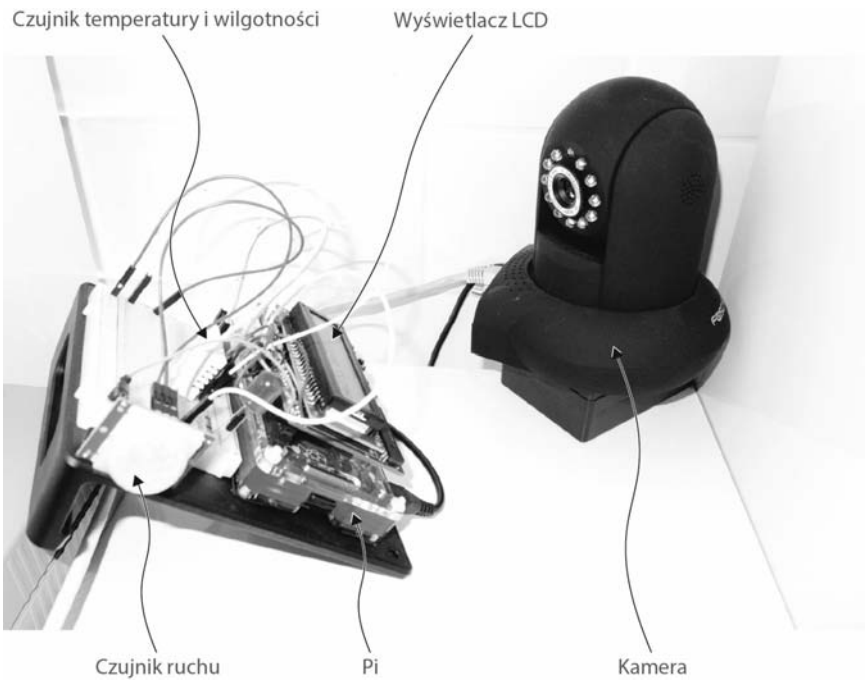
Zanim rzucimy się na głęboką wodę architektury WWW rzeczy i pokażemy, w jaki sposób można ją zaimplementować od samych podstaw, chcielibyśmy dać czytelnikowi przedsmak tego, czym jest WoT. Ten rozdział został napisany jako zestaw ćwiczeń, w ramach których napiszemy małe i proste aplikacje webowe korzystające z danych generowanych przez rzeczywiste urządzenia. Każde z tych ćwiczeń będzie łagodnym ukazaniem problemów oraz zagadnień technicznych, z którymi stykamy się podczas konstruowania urządzeń podłączonych do WWW oraz aplikacji wchodzących z nimi w interakcję.

W tym rozdziale czytelnik będzie miał okazję, by zakasać rękawy, wziąć się do roboty i napisać kilka prostych (a także parę nieco trudniejszych) aplikacji WoT. A co, jeśli czytelnik nie ma odpowiedniego urządzenia? To żaden problem: może skorzystać z naszego! Aby umożliwić wykonanie ćwiczeń zaprezentowanych w tym rozdziale bez konieczności kupowania jakiegokolwiek urządzenia, podłączyliśmy nasze urządzenie do internetu i zapewniliśmy dostęp do tego urządzenia. Oczywiście jeśli czytelnik posiada już odpowiednie urządzenie, to będzie mógł pobrać kody źródłowe aplikacji przedstawionych w tym rozdziale i uruchomić je na własnym urządzeniu. Sposób uruchamiania kodu na urządzeniu został szczegółowo opisany w rozdziale 7.

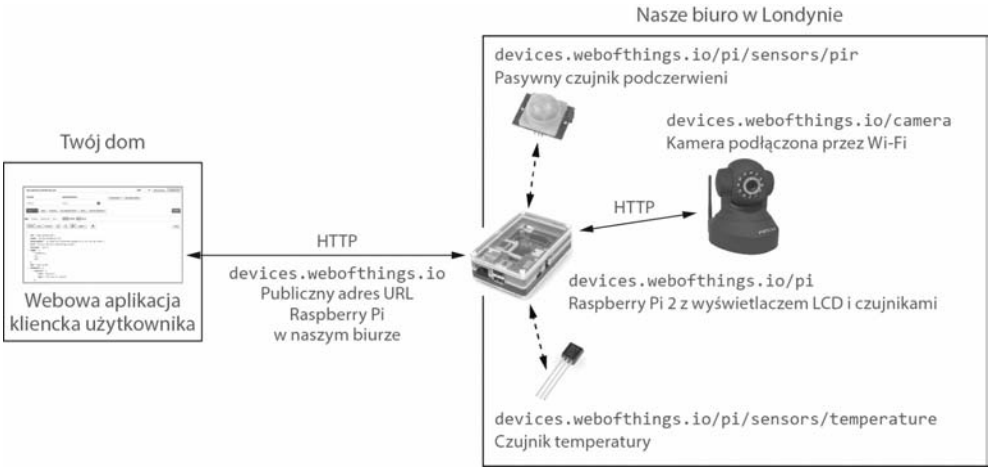
2.1. Poznajemy urządzenie WWW rzeczy

Jak już wspominaliśmy, ten rozdział składa się z serii krótkich i przyjemnych ćwiczeń. Każde z nich pozwala na interakcję z rzeczywistym urządzeniem WoT, działającym w naszym biurze przez 24 godziny na dobę. Dzięki temu czytelnik będzie mógł wykonywać ćwiczenia bez konieczności posiadania rzeczywistego urządzenia.

Urządzenie, które podłączyliśmy w naszym biurze do internetu i udostępniliśmy, to Raspberry Pi 2 (dla rodziny i przyjaciół po prostu Pi), przedstawione na rysunku 2.1 i opisane bardziej szczegółowo w rozdziale 4. Jeśli czytelnik nigdy wcześniej nie widział Raspberry Pi, to może je sobie wyobrazić jako płytkę drukowaną wielkości karty kredytowej, z procesorem i kilkoma czujnikami, podłączoną kablem Ethernet do sieci lokalnej i internetu. W naszym systemie Pi działa jako brama zapewniająca możliwość interakcji z kilkoma podłączonymi do niego czujnikami i urządzeniami. Takie bramy zostały szczegółowo opisane w rozdziale 7., a na razie wystarczy wiedzieć, że na naszym Pi został uruchomiony serwer WWW zapewniający możliwość zdalnego dostępu do zasobów komputera, jak pokazaliśmy na rysunku 2.2.



Rysunek 2.1. Raspberry Pi oraz kamera internetowa używane w ćwiczeniach i działające w naszym biurze w Londynie



Rysunek 2.2. Konfiguracja urządzeń i czujników stosowanych w ćwiczeniach przedstawionych w tym rozdziale

W czasie gdy pisaliśmy tę książkę, nasze Raspberry Pi było wyposażone w wyświetlacz ciekłokrystaliczny (LCD), kamerę, czujnik temperatury i czujnik RIP (pasywny czujnik podczerwieni, czyli czujnik ruchu). W przyszłości chcielibyśmy dodać do tego systemu kolejne czujniki oraz inne elementy, dlatego też gorąco zachęcamy do przeprowadzania eksperymentów wykraczających poza ćwiczenia zamieszczone w tym rozdziale. Czytelnik zapewne prędko się zorientuje, że różne techniki i wzorce opisane w tej książce pozwolą bardzo szybko dostosować przedstawione tu przykłady do każdego urządzenia, czujnika i przeznaczenia.

2.1.1. Podejrzany: Raspberry Pi

Raspberry Pi zaprezentujemy bardziej szczegółowo w rozdziale 4., więc na razie to, co czytelnik musi wiedzieć jego temat, ogranicza się do informacji, że jest to mały komputer, do którego można podłączać wiele różnych czujników i akcesoriów. Udostępnia on wszystkie możliwości, których można oczekiwać od komputerów biurkowych, cechuje się jednak mniejszym zapotrzebowaniem na energię oraz znacznie mniejszymi wymiarami. Co więcej, korzystając z portów GPIO, można do niego dołączać wszelkiego rodzaju cyfrowe czujniki, elementy sygnalizacyjne i wykonawcze. *Elementy wykonawcze i sygnalizacyjne* (ang. *actuators*) to ogólny termin, którego w tej książce używamy do określania wszystkich elementów podłączanych do urządzenia mających wpływ na realny świat. Są to elementy, które na przykład włączają określone diody LED, wyświetlają tekst na panelach LCD, obracają motor, otwierają drzwi, odtwarzają muzykę itd. W świecie WWW rzeczy takie elementy wykonawcze i sygnalizacyjne aktywuje się w taki sam sposób, w jaki wysyła się żądania zapisu do odpowiedniego API — używa się w tym celu protokołu HTTP. A teraz wróćmy do zapowiadanych ćwiczeń. W pierwszej kolejności czytelnik będzie musiał pobrać kody przykładów przedstawionych w dalszej części rozdziału — można je znaleźć na serwerze FTP wydawnictwa Helion, pod adresem *ftp://ftp.helion.pl/przyklady/intrze.zip*.

Repozytorium z kodami można także skopiować na swój własny komputer; w skład tego repozytorium wchodzi kilka katalogów, po jednym dla każdego rozdziału książki. Ćwiczenia zaprezentowane w tym rozdziale znajdują się w katalogu *rozdzial2-witaj-wot/klient*. Jeśli ktoś będzie się zastanawiał, co z kodami aplikacji działającej po stronie serwera, to nie musi się tym przejmować! W dalszej części książki dokładnie pokażemy, jak należy się nimi zająć.

Jak zdobyć kody źródłowe przykładów przedstawionych w tym rozdziale

Do synchronizacji kodu pomiędzy naszymi komputerami a naszym Pi wykorzystamy serwis GitHub[a]. Alternatywnym rozwiązaniem jest serwis Bitbucket[b], który także działa i jest dość podobny do GitHuba. Oba te serwisy są oparte na systemie kontroli wersji Git. Spolonizowane kody przykładów prezentowanych w tej książce są dostępne na serwerze FTP wydawnictwa Helion, pod adresem *ftp://ftp.helion.pl/przyklady/intrze.zip*, a oryginalne kody źródłowe można znaleźć w repozytorium w serwisie GitHub (jak również na stronie *http://book.webofthings.io/*). Przykłady do tego rozdziału znajdują się w katalogu *rozdzial02-witaj-wot*.

Jeśli czytelnik nie zna systemu Git ani jego poleceń, to i tak nie ma się czym przejmować: w internecie można znaleźć bardzo wiele informacji na ten temat. Na wszelki wypadek poniżej przedstawiliśmy kilka najistotniejszych poleceń systemu Git:

- `git clone` — pobiera kopię repozytorium na lokalny komputer. W przypadku pobierania kodów do tej książki konieczne jest zastosowanie opcji `recursive`, która dodatkowo pobierze także wszystkie projekty podrzędne: `git clone https://github.com/webofthings/wot-book --recursive`.
- `git commit -a -m "komunikat..."` — zatwierdza lokalne zmiany w kodzie.
- `get push origin master` — przesyła ostatnio zatwierdzone lokalne zmiany do zdalnego repozytorium (`origin`), zapisując je w gałęzi `master`.

[a] GitHub jest bardzo popularnym internetowym serwisem do zarządzania kodem źródłowym. Korzysta z niego wiele projektów typu *open source*, ponieważ — jak by to powiedzieć — jest on niesamowity. Doskonałe wprowadzenie do sposobów wykorzystywania tego serwisu można znaleźć na stronie *http://bit.ly/intro-git*.

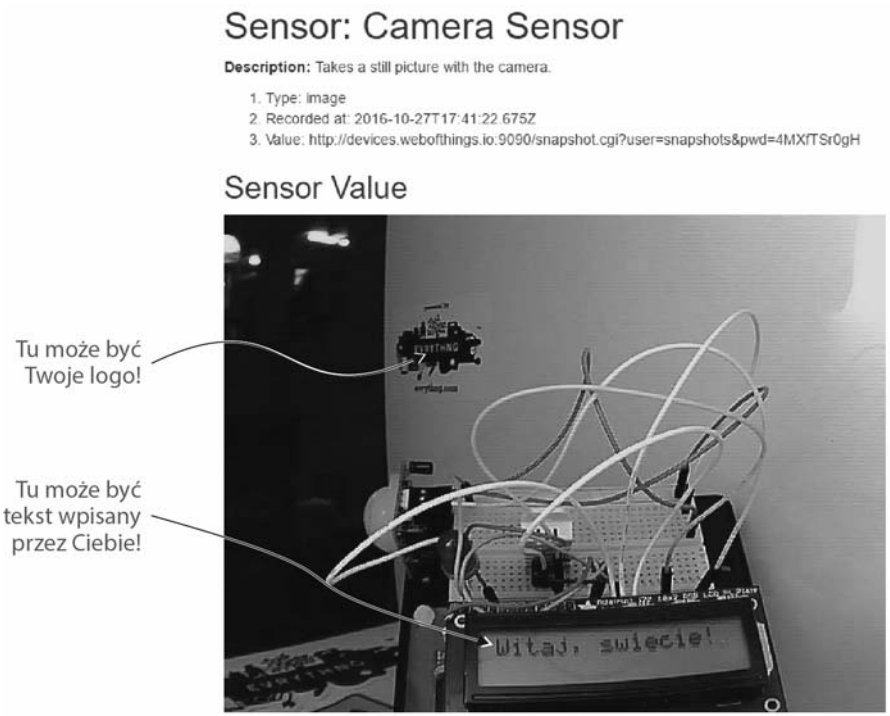
[b] *https://bitbucket.com*.

2.2. Ćwiczenie 1. Przeglądanie urządzenia na WWW rzeczy

Poznawanie WWW rzeczy zaczniemy od prostego ćwiczenia — jedyne, co trzeba w nim zrobić, to kliknąć kilka odnośników na stronie wyświetlonej w przeglądarce. Chcielibyśmy w ten sposób pokazać przede wszystkim to, że w świecie WWW rzeczy urządzenia mogą jednocześnie oferować wizualny interfejs użytkownika (strony WWW), pozwalający ludziom je kontrolować oraz wchodzić z nimi w interakcję, oraz API — interfejs programowania aplikacji — zapewniający analogiczne możliwości innym komputerom i aplikacjom.

2.2.1. Część 1. WWW jako interfejs użytkownika

W tym ćwiczeniu czytelnik użyje przeglądarki WWW do wejścia w interakcję z rzeczywistym urządzeniem WoT podłączonym do internetu w naszym biurze. W pierwszej kolejności warto przyjrzeć się konfiguracji systemu, używając do tego kamery internetowej (patrz rysunek 2.3). W tym celu należy uruchomić ulubioną przeglądarkę i w niej



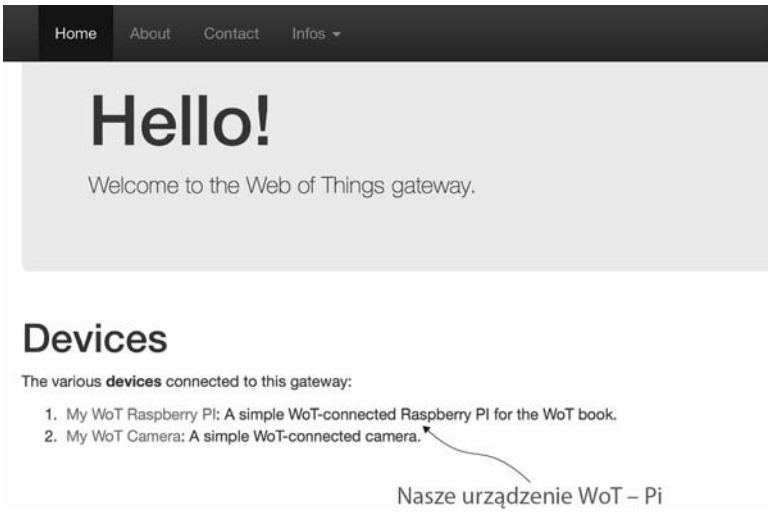
Rysunek 2.3. Strona WWW kamery wchodzącej w skład naszego zestawu. Zdjęcia są wykonywane przez kamerę na bieżąco

wyświetlić najnowsze zdjęcie wykonane przez naszą kamerę, dostępne na stronie *http://devices.webofthings.io/camera/sensors/picture/*. Na tej stronie zawsze jest prezentowane najnowsze zdjęcie zrobione przez naszą kamerę, dzięki czemu czytelnik może zobaczyć urządzenie, na którym będzie przeprowadzał eksperymenty (warto to robić w nocy — wtedy jest jeszcze zabawniej!). Oczywiście sama kamera nie będzie widoczna na zdjęciu.

Łatwo zauważyć, że podany adres URL ma określoną strukturę. Spróbujmy zatem trochę z tym adresem poeksperymentować i użyć jego podstawowej postaci, która pozwoli przejść na stronę główną bramy pozwalającej na przeglądanie urządzeń działających w naszym biurze i interakcję z nimi (patrz rysunek 2.4). W tym celu należy wpisać w przeglądarce następujący adres URL: *http://devices.webofthings.io*.

Ten adres URL zawsze przekieruje na *stronę główną* bramy działającej w naszym biurze, która prezentuje listę wszystkich podłączonych urządzeń. W tym przypadku widać, że są do niej podłączone dwa urządzenia:

- Raspberry Pi z różnymi czujnikami, elementami sygnalizacyjnymi i wykonawczymi;
- kamera internetowa (ta, której przed chwilą użyliśmy).



Rysunek 2.4. Strona HTML bramy do naszego urządzenia WoT. Dwa odnośniki umieszczone u dołu pozwalają przejść na strony urządzeń podłączonych do tej bramy

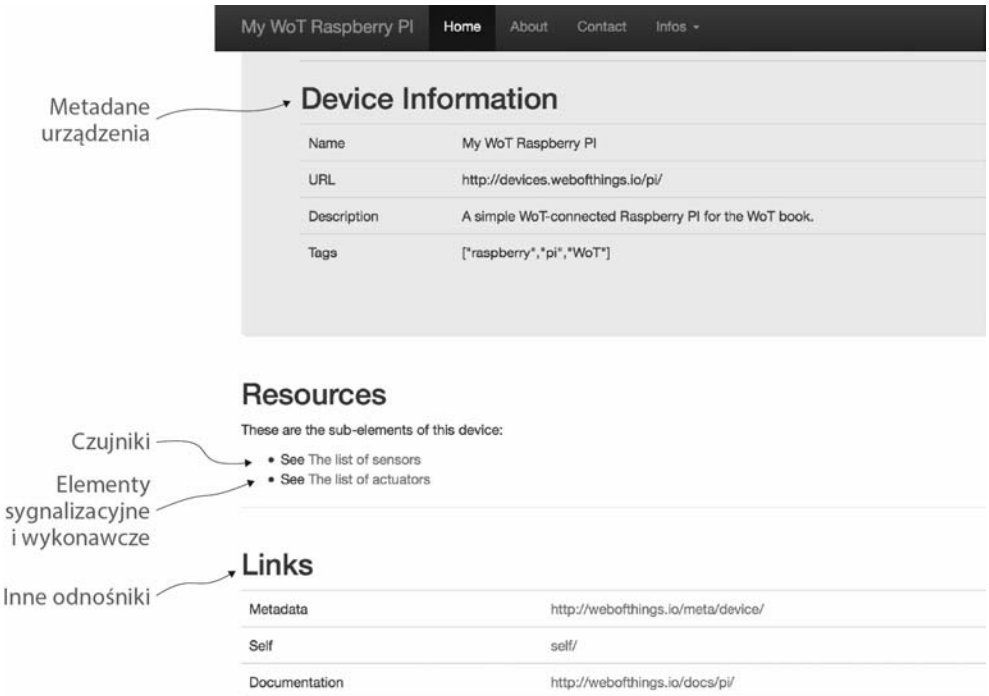
Trzeba zwrócić uwagę na to, że strona ta jest generowana automatycznie, na podstawie urządzeń, które są faktycznie fizycznie podłączone do bramy, dlatego może się zdarzyć, że będzie na niej wyświetlonych kilka dodatkowych urządzeń, o ile tylko je podłączymy. Właśnie tak! Choć wygląda ona na zwyczajną stronę WWW, to jednak w rzeczywistości odczytuje *realne* dane przesyłane w *czasie rzeczywistym* przez *realne* urządzenia działające w naszym *realnym* biurze!

Spróbujmy teraz kliknąć odnośnik *My WoT Raspberry Pi* (moje Raspberry Pi), aby przejść na stronę główną samego urządzenia. Ponieważ używamy do tego przeglądarki, z łatwością możemy zauważyć, że adres URL wyświetlanej w niej strony zmienił się na *http://devices.webofthings.io/pi*, jak pokazaliśmy na rysunku 2.5.

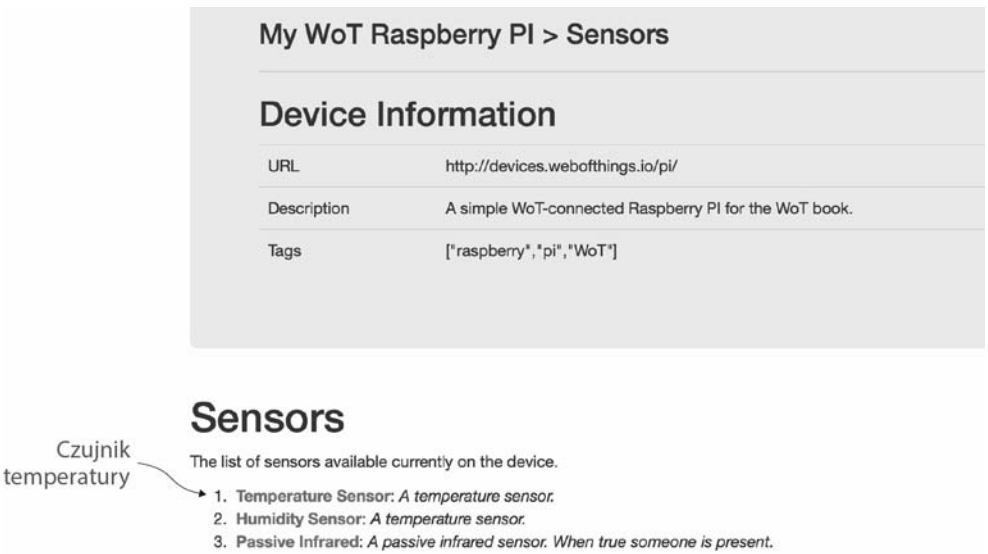
To kolejna strona główna — tym razem samego urządzenia. W tym przypadku wystarczyło dopisać na końcu adresu URL bramy fragment `/pi`.

Wróćmy teraz do samej strony: warto spróbować wskazać myszą różne umieszczone na niej odnośniki, aby przyjrzeć się dokładniej ich strukturze, a następnie kliknąć odnośnik *The list of sensors* (lista czujników). Także to kliknięcie spowoduje zmianę adresu URL strony wyświetlonej w przeglądarce, tym razem na *http://devices.webofthings.io/pi/sensors* (patrz rysunek 2.6).

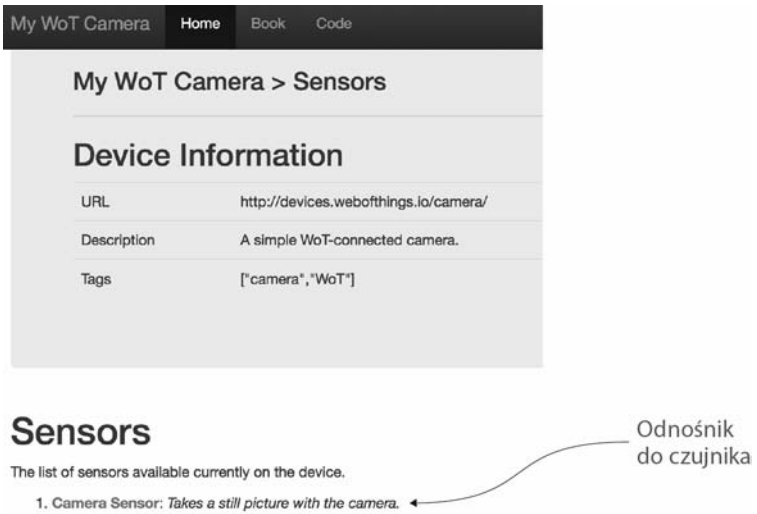
Jak na razie wszystko jest bardzo proste: poprosiliśmy o wyświetlenie strony WWW prezentującej listę czujników (`/sensors`) podłączonych do Raspberry Pi (`/pi`), które z kolei jest podłączone do bramy `deviced.webofthings.io`. Warto sobie przypomnieć, że do bramy jest także podłączona kamera, dlatego zamieniając w adresie URL `/pi/` na `/camera/`, można bezpośrednio wyświetlić listę czujników kamery: *http://devices.webofthings.io/camera/sensors* (patrz rysunek 2.7).



Rysunek 2.5. Strona główna naszego Raspberry Pi. U dołu tej strony umieszczona jest lista odnośników pozwalających na przeglądanie i poznawanie różnych zasobów podłączonych do urządzenia; w tym przypadku są to czujniki, elementy sygnalizacyjne i wykonawcze



Rysunek 2.6. Lista czujników podłączonych do naszego Pi. Można kliknąć każdy z nich, by wyświetlić najnowszą wartość zarejestrowaną przez dany czujnik



Rysunek 2.7. Strona czujników kamery. W tym przypadku jest tylko jeden taki czujnik; reprezentuje on zdjęcie wykonane przez kamerę

Wróćmy teraz do listy czujników naszego Pi i przyjrzyjmy się im nieco dokładniej. Obecnie dostępne są tylko trzy czujniki: temperatury, wilgotności i pasywny czujnik podczerwieni. Po kliknięciu odnośnika *Temperature Sensor* (czujnik temperatury) w przeglądarce zostanie wyświetlona strona tego czujnika, prezentująca aktualnie zmierzoną przez niego wartość. I w końcu, tak samo jak w przypadku czujników, można wyświetlić listę elementów sygnalizacyjnych i wykonawczych podłączonych do naszego Pi, a następnie wyświetlić stronę ze szczegółowymi informacjami o wybranym urządzeniu — *Actuator Details* (parz rysunek 2.13) — dostępną pod adresem *http://devices.webofthings.io/pi/actuators/display*.

Jeśli chodzi o prezentację, to do naszego Raspberry Pi został podłączony prosty wyświetlacz LCD, pozwalający na wyświetlanie tekstów. Jego możliwości wykorzystamy w ćwiczeniu 2.4. Prezentowane są także informacje na temat tego komponentu naszego zestawu, a w szczególności tekst aktualnie prezentowany wyświetlaczu, dane o API pozwalającym na przesyłanie własnego tekstu na wyświetlacz oraz formularz do wysyłania tego tekstu. Na razie formularz nam się nie przyda, jednak wrócimy do niego w podrozdziale 2.4.

2.2.2. Część 2. WWW jako API

W części 1. weszliśmy w interakcję z WWW rzeczy z poziomu przeglądarki WWW. Został także zaprezentowany sposób, w jaki internauci mogą poznawać zasoby urządzenia (czujniki, elementy sygnalizacyjne i wykonawcze itd.) oraz w jaki można wchodzić w interakcję z tymi zasobami za pośrednictwem stron WWW. Wszystkie te czynności są wykonywane poprzez przeglądanie zasobów fizycznego urządzenia, dokładnie tak samo jak przeglądamy strony na zwyczajnych witrynach WWW. Co jednak zrobić w przypadku, kiedy chcemy, by zamiast użytkownika te same operacje wykonywało inne urządzenie

lub aplikacja — automatycznie i bez żadnej ingerencji człowieka? Jak można ułatwić wszystkich klientom WWW odnajdywanie urządzenia, zrozumienie, co ono robi, poznanie jego API, określenie, jakie polecenia można do niego przesyłać, itd.?

W dalszej części książki pokażemy i wyjaśnimy szczegółowo, jak można wykonywać wszystkie te operacje. Na razie jednak ograniczymy się jedynie do pokazania, w jaki sposób wykorzystanie WWW ułatwia obsługę zarówno ludzi, jak i aplikacji. W tym celu zaprezentujemy, co „widzi" inne urządzenie lub aplikacja podczas przeglądania zasobów naszego urządzenia.

Do wykonania tego ćwiczenia czytelnik będzie potrzebował przeglądarki Chrome z zainstalowanym rozszerzeniem Postman[1] — jednym z naszych ulubionych. Jeśli ktoś preferuje pracę z poziomu wiersza poleceń, to równie dobrze może wykorzystać program cURL[2]. Postman to wygodna, niewielka aplikacja, która może znacznie ułatwić korzystanie z webowych API, gdyż pozwala na proste wysyłanie żądań HTTP, ustawianie przeróżnych opcji tych żądań, takich jak nagłówki i treści, itd. Dodatek Postman bardzo ułatwi czytelnikowi życie podczas lektury tej książki, dlatego naprawdę warto go zainstalować już teraz.

W poprzednim punkcie rozdziału — części 1. — przeglądarka pełniła funkcję zwyczajnego klienta, żądającego od serwera WWW przesyłania treści. Przeglądarka automatycznie prosi, by zwracanymi treściami były dokumenty HTML, które po przesłaniu z serwera do przeglądarki są w niej wyświetlane.

W tym punkcie rozdziału zrobimy niemal to samo, ale tym razem poprosimy, by serwer zwracał dokumenty w formacie JSON, a nie strony HTML. JSON jest najprawdopodobniej najpopularniejszym formatem wymiany danych używanym obecnie w internecie. Ma on bardzo prostą składnię, dane zapisane w tym formacie są zwarte i nie zajmują wiele miejsca, co sprawia, że w porównaniu ze swoim rodzicem — formatem XML — JSON jest bardzo efektywny. Co równie ważne, dane w formacie JSON mogą być bez trudu odczytywane i zapisywane przez ludzi, a także analizowane i generowane przez komputery, co z kolei sprawia, że jest on *optymalnym* formatem do wymiany danych na WWW rzeczy. Proces przesyłania prośby o zastosowanie konkretnego kodowania jest w specyfikacji HTTP 1.1 nazywany *negocjowaniem zawartości* (ang. *content negotiation*) — zagadnienie to omówimy w rozdziale 6.

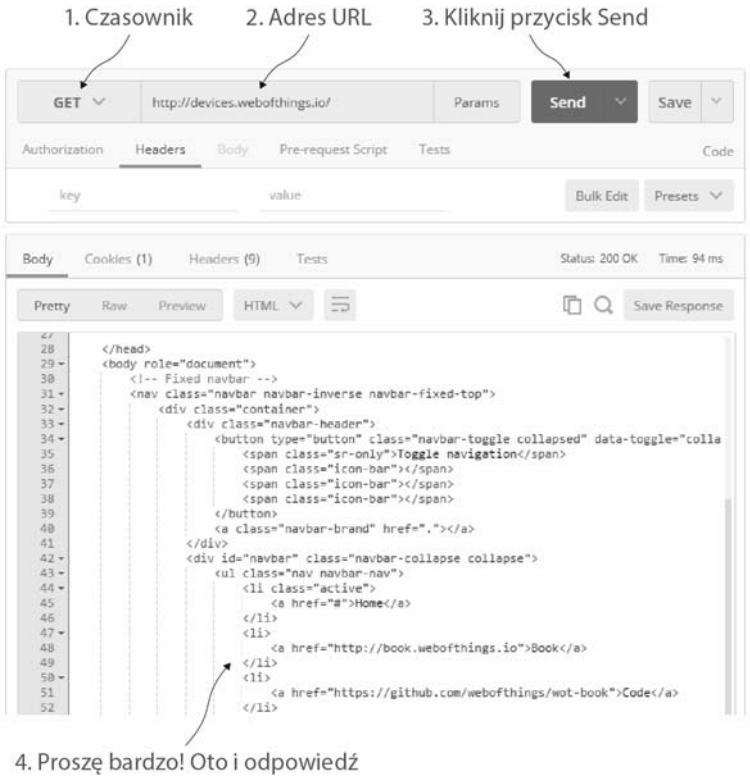
KROK 1. POBIERANIE LISTY URZĄDZEŃ Z BRAMY

Podobnie jak w poprzednim przypadku, także i teraz w celu pobrania listy urządzeń prześlemy żądanie GET na stronę główną bramy. Aby to zrobić, należy wpisać w rozszerzeniu Postman adres URL tej bramy i kliknąć przycisk *Send*, jak pokazaliśmy na rysunku 2.8.

---

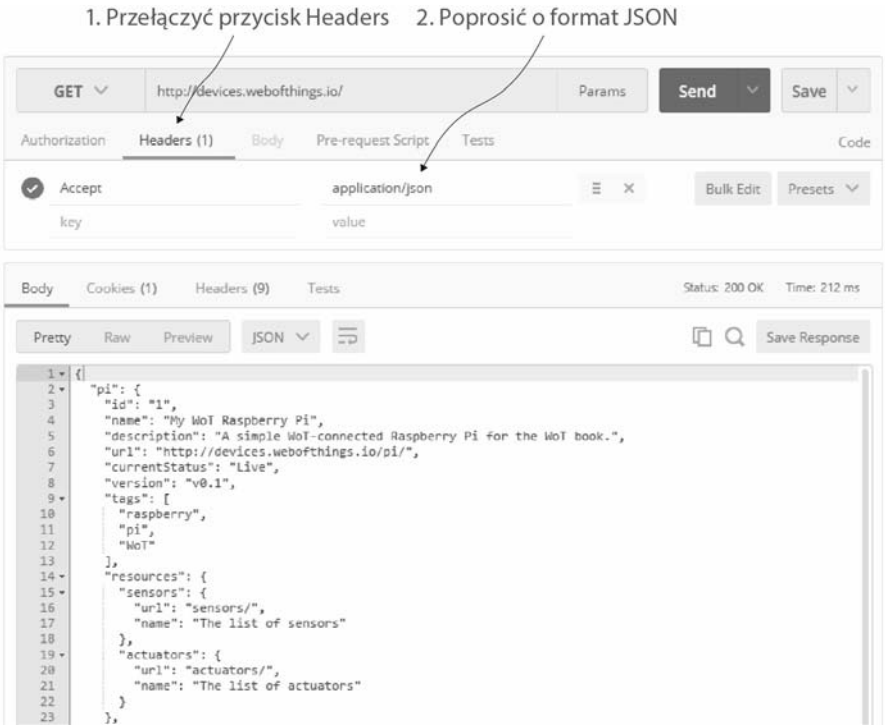
[1] Można je pobrać ze strony *http://www.getpostman.com/*.

[2] cURL to program używany z poziomu wiersza poleceń pozwalający na przesyłanie danych przy użyciu różnych protokołów, w tym także przy użyciu protokołu HTTP. Jeśli nie został on domyślnie zainstalowany na komputerze użytkownika, to bez żadnych problemów można go zainstalować samodzielnie, wybierając odpowiednią wersję: na komputery Mac, systemy Linux oraz Windows. Witryna poświęcona temu programowi znajduje się pod adresem *http://curl.haxx.se/*.



Rysunek 2.8. Pobieranie strony głównej bramy przy użyciu rozszerzenia Postman. Wykorzystujemy w tym celu żądanie HTTP GET (1), podając w nim adres URL bramy (2). Treść odpowiedzi będzie zawierać dokument HTML (4)

Ponieważ większość serwerów WWW domyślnie zwraca dokumenty HTML, w obszarze prezentującym zawartość odpowiedzi zobaczymy kod strony WWW (4). Dokładnie to dzieje się domyślnie za każdym razem, gdy odwołujemy się do serwera WWW, używając przeglądarki. Teraz, aby zamiast kodu HTML pobrać dane w formacie JSON, trzeba kliknąć przycisk *Headers* i dodać nagłówek o nazwie `Accept` i wartości `application/json`. Potem można ponownie kliknąć przycisk *Send*, jak pokazaliśmy na rysunku 2.9. Dodanie tego nagłówka do żądania można by porównać z powiedzeniem serwerowi: „Słuchaj, jeśli możesz, to zwróć mi, proszę, wyniki zapisane w formacie JSON". Ponieważ brama obsługuje zwracanie takich odpowiedzi, tym razem w obszarze zawartości odpowiedzi zostanie wyświetlony kod JSON; będzie on stanowił zrozumiały dla komputerów odpowiednik pobranej wcześniej strony WWW, choć w tym przypadku będzie on zawierał samą treść, bez żadnych elementów wizualnych (czyli kodu HTML).



Rysunek 2.9. Pobieranie listy urządzeń podłączonych do bramy z użyciem rozszerzenia Postman. W tym przypadku zastosowany został dodatkowo nagłówek `Accept` o wartości `application/json`, stanowiący prośbę o przekazanie wyników zapisanych w formacie JSON

Kod JSON zwrócony w zawartości odpowiedzi to zrozumiały dla komputerów opis wszystkich urządzeń podłączonych do bramy; poniżej pokazaliśmy, jak wygląda ten kod.

```
{
  "pi": {
    "id": "1",
    "name": "My WoT Raspberry Pi",
    "description": "A simple WoT-connected Raspberry Pi for the WoT book.",
    "url": "http://devices.webofthings.io/pi/",
    "currentStatus": "Live",
    "version": "v0.1",
    "tags": [
      "raspberry",
      "pi",
      "WoT"
    ],
    "resources": {
      "sensors": {
        "url": "sensors/",
        "name": "The list of sensors"
      },
      "actuators": {
```

```
        "url": "actuators/",
        "name": "The list of actuators"
      }
    },
    "links": {
      "meta": {
        "rel": "http://book.webofthings.io",
        "title": "Metadata"
      },
      "doc": {
        "rel": "https://www.raspberrypi.org/products/raspberry-pi-2-model-b/",
        "title": "Documentation"
      },
      "ui": {
        "rel": ".",
        "title": "User Interface"
      }
    }
  },
  "camera": {
    [ ... opis obiektu kamery... ]
  }
}
```

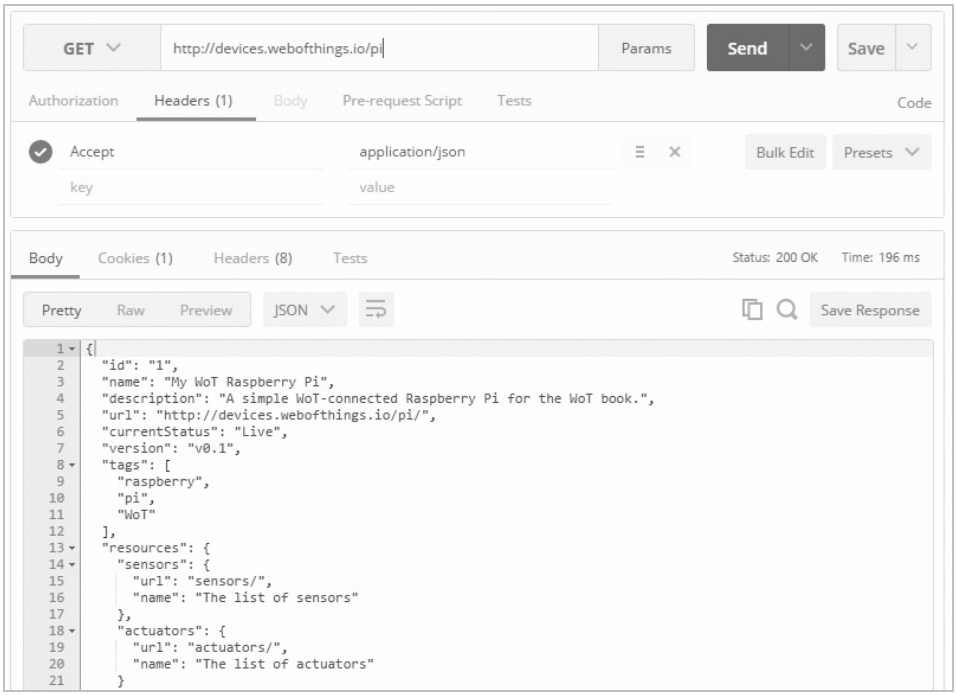
W powyższym dokumencie JSON można wskazać dwa elementy pierwszego poziomu (`pi` oraz `camera`), które reprezentują dwa urządzenia podłączone do bramy, jak również kilka bardziej szczegółowych informacji na ich temat, takich jak ich adres URL, nazwa, identyfikator i opis. Nie trzeba się przejmować, jeśli nie wszystkie zamieszczone tu informacje są zrozumiałe — po przeczytaniu kilku kolejnych rozdziałów wszystko stanie się całkowicie jasne.

KROK 2. POBIERANIE INFORMACJI O KONKRETNYM URZĄDZENIU

A teraz spróbujmy zmienić adres URL w rozszerzeniu Postman tak, by wskazywał on na nasze Raspberry Pi (to ten sam adres, którego użyliśmy w poprzednim punkcie rozdziału — w części 1.), a następnie kliknijmy przycisk *Send*, jak pokazano na rysunku 2.10.

Obecnie treść odpowiedzi zawiera obiekt JSON z informacjami o urządzeniu Raspberry Pi, takimi samymi jak te, które zobaczyliśmy już wcześniej; ponadto, jak widać na poniższym przykładzie, obiekt `resources` zawiera właściwości `sensors` oraz `actuators`:

```
"resources": {
  "sensors": {
    "url": "sensors/",
    "name": "The list of sensors"
  },
  "actuators": {
    "url": "actuators/",
    "name": "The list of actuators"
  }
}
```



Rysunek 2.10. Pobieranie danych o urządzeniu Raspberry Pi zapisanych w formacie JSON. Informacje zwrócone w odpowiedzi zawierają metadane dotyczące urządzenia oraz odnośniki do jego zasobów

KROK 3. POBIERANIE LISTY CZUJNIKÓW PODŁĄCZONYCH DO URZĄDZENIA

Aby pobrać listę czujników podłączonych do urządzenia, należy — podobnie jak wcześniej — dodać do jego adresu URL końcówkę `/sensors` i tak zmienione żądanie przesłać, używając rozszerzenia Postman. W tym przypadku żądanie HTTP `GET` zwróci w odpowiedzi następujący dokument zapisany w formacie JSON:

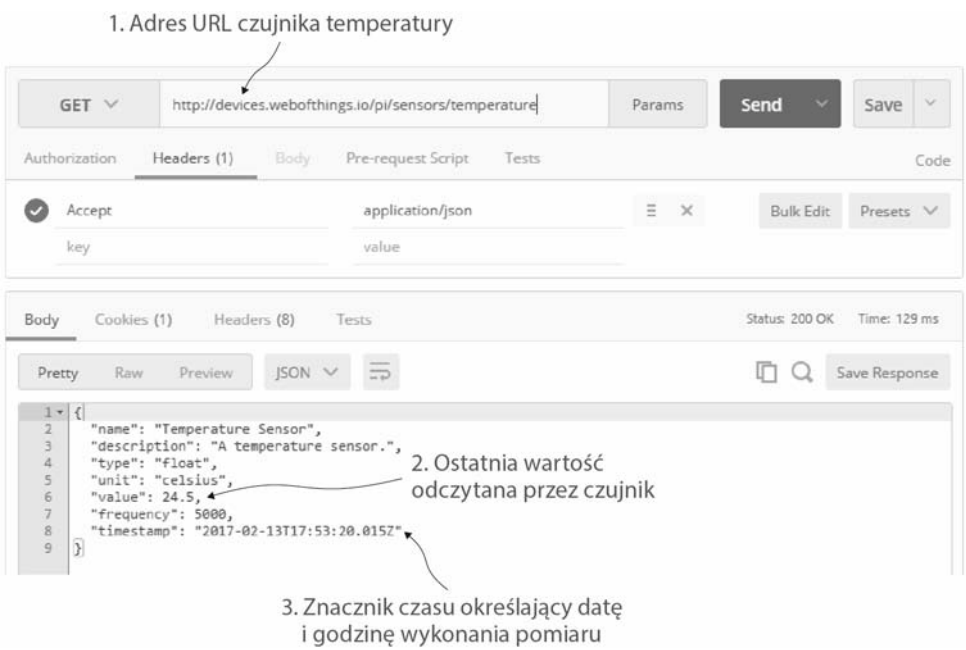
```
{
  "temperature": {
    "name": "Temperature Sensor",
    "description": "A temperature sensor.",
    "type": "float",
    "unit": "celsius",
    "value": 23.4,
    "timestamp": "2017-02-13T14:39:17.240Z",
    "frequency": 5000
  },
  "humidity": {
    "name": "Humidity Sensor",
    "description": "A temperature sensor.",
    "type": "float",
    "unit": "percent",
    "value": 38.9,
    "timestamp": "2017-02-13T14:39:17.240Z",
    "frequency": 5000
```

```
  },
  "pir": {
    "name": "Passive Infrared",
    "description": "A passive infrared sensor. True when someone present.",
    "type": "boolean",
    "value": true,
    "timestamp": "2017-02-13T14:39:17.240Z",
    "gpio": 20
  }
}
```

Jak pokazuje powyższy przykład, do urządzenia zostały podłączone trzy czujniki (są to odpowiednio: czujnik temperatury — `temperature`, czujnik wilgotności — `humidity`, czujnik ruchu — `pir`), a w odpowiedzi zostały umieszczone szczegółowe informacje o nich, jak również ostatnia zmierzona przez nie wartość.

KROK 4. POBIERANIE SZCZEGÓŁOWYCH INFORMACJI O CZUJNIKU

I w końcu pobierzemy szczegółowe informacje o konkretnym czujniku. W tym celu należy dodać do adresu URL w rozszerzeniu Postman końcówkę `/temperature` i ponownie kliknąć przycisk *Send*. Adres URL powinien teraz mieć postać: *http://devices.webofthings.io/pi/sensors/temperature*, jak pokazaliśmy na rysunku 2.11.



Rysunek 2.11. Pobieranie obiektu reprezentującego czujnik temperatury podłączonego do Raspberry Pi. Jak widać, uzyskane informacje obejmują aktualny odczyt czujnika (23,4 stopnia Celsjusza) oraz datę i godzinę wykonania pomiaru (17:53, 13 lutego 2017 r.)

Wykonanie powyższego żądania zwróci dokładne informacje dotyczące czujnika temperatury, a w szczególności ostatnią zmierzoną przez niego wartość (pole `value`). Aby pobrać wyłącznie wartość pomiaru, do powyższego adresu URL czujnika należy dodać końcówkę `/value` (rozwiązanie to działa także w przypadku pozostałych czujników):

```
{
  "value":22.4
}
```

2.2.3. Wnioski

Teraz nadeszła pora na to, by czytelnik na własną rękę poeksperymentował i wypróbował różne adresy URL przedstawione wcześniej w tym ćwiczeniu. Warto zwrócić uwagę na to, czym one się od siebie różnią i jaką mają strukturę, poprzeglądać zasoby urządzenia, spróbować zrozumieć, jakie dane udostępniają poszczególne czujniki, w jakim formacie dane te są zapisywane itd. Dodatkowo można przyjrzeć się urządzeniom elektronicznym znajdującym się w naszym otoczeniu: urządzeniom gospodarstwa domowego w kuchni, telewizorowi oraz systemowi audio-wideo w salonie, systemowi do zamawiania kawy, elektronicznemu systemowi informacji kolejowej — zależnie od tego, gdzie czytelnik czyta tę książkę. A teraz spróbujmy sobie wyobrazić, że wszystkie te urządzenia i usługi mogłyby mieć podobną strukturę: adresy URL, treści, ścieżki itd. Spróbujmy odwzorować tę strukturę z wykorzystaniem formatu JSON przedstawionego w tym rozdziale: zapisać adresy URL oraz zwracane obiekty JSON.

Wszystko, co zobaczyliśmy w tym ćwiczeniu, pokazuje, że zarówno ludzie, jak i aplikacje mogą pobierać dane, używając dokładnie tych samych adresów URL, lecz nieco innych formatów wyjściowych (dla ludzi bardziej odpowiedni jest format HTML, a dla aplikacji — JSON). Oczywiście dane zwracane w obu tych formatach są identyczne, dzięki czemu programiści aplikacji bez trudu mogą zmieniać stosowane formaty. To tylko jeden z przykładów pokazujących, jak proste — a jednocześnie potężne — są technologie związane z WWW. Dzięki niezwykle popularnym standardom, takim jak HTTP czy adresy URL, interakcja z rzeczywistym światem z poziomu przeglądarek WWW stała się naprawdę łatwa. Znacznie więcej informacji na temat tych koncepcji zostało przedstawionych w rozdziale 6. tej książki.

2.3. Ćwiczenie 2. Pobieranie danych z czujników w WWW rzeczy

W ćwiczeniu 1. poznaliśmy strukturę urządzenia WoT oraz sposób jego działania. W szczególności przekonaliśmy się, że każdy element takiego urządzenia jest po prostu zasobem, posiada unikalny adres URL i mogą się do niego odwoływać zarówno ludzie, jak i komputery, aby odczytywać i zapisywać dane. Teraz czas założyć kapelusz programisty i zabrać się do pisania pierwszej aplikacji wchodzącej w interakcję z urządzeniem należącym do WWW rzeczy.

2.3.1. Część 1. Odczyt bieżącej wartości czujnika

W celu wykonania tego ćwiczenia należy przejść do pobranego z serwisu GitHub katalog o nazwie *rozdzial02-witaj-wot/klient*. W tym katalogu należy dwukrotnie kliknąć plik *ex2.1-polling-temp.html*, by otworzyć go w nowoczesnej przeglądarce[3]. Ta strona wyświetla wartość czujnika temperatury podłączonego do Pi w naszym biurze i aktualizuje tę wartość co pięć sekund, pobierając w tym celu dane w formacie JSON (dokładnie tak, jak pokazano na rysunku 2.11).

Do pobierania danych z naszego Pi strona używa biblioteki jQuery[4]. Otwórzmy teraz ten plik w ulubionym edytorze, aby przyjrzeć się jego kodowi źródłowemu. Należy w nim zwrócić uwagę na dwie rzeczy:

- znacznik `<h2>`, pokazujący, gdzie zostanie zapisana bieżąca wartość czujnika;
- funkcję JavaScript o nazwie `doPoll()`, która odczytuje wartość czujnika z Raspberry Pi, wyświetla ją, a następnie, po upływie pięciu sekund, wywołuje samą siebie. Kod tej funkcji został przedstawiony na poniższym listingu 2.1.

Listing 2.1. Pobieranie wartości z czujnika temperatury

```
$(document).ready(
  function doPoll() {
    $.getJSON('http://devices.webofthings.io/pi/sensors/temperature',
      function (data) {
        console.log(data);
        $('#temp').html(data.value + ' ' + data.unit);
        setTimeout(doPoll, 5000);
    });
});
```

Czekamy, aż strona zostanie pobrana, a następnie wywoła funkcję doPoll().

Używamy funkcji pomocniczej do generowania żądań AJAX pobierających z czujnika temperatury dane w formacie JSON.

Ta funkcja zostanie wywołana, kiedy nadejdzie odpowiedź.

Wybieramy element HTML o identyfikatorze "temp" i aktualizujemy jego zawartość, używając wyrażeń data.value (wartość) oraz data.unit (jednostka), korzystających z informacji zwróconych w danych JSON przesłanych z naszego urządzenia.

Funkcja doPoll() ustawia czasomierz, który wywoła tę samą funkcję po upływie pięciu sekund (5000 milisekund).

Podczas tworzenia (a szczególnie testowania!) aplikacji internetowych bardzo przydatna będzie możliwość wyświetlania treści z poziomu kodu JavaScript poza stroną WWW wyświetloną w przeglądarce. Do tego celu można wykorzystać konsolę JavaScript. Aby ją wyświetlić w przeglądarce Chrome, należy kliknąć w dowolnym miejscu strony prawym przyciskiem myszy, a następnie z wyświetlonego menu podręcznego wybrać opcję *Zbadaj* — spowoduje to wyświetlenie konsoli JavaScript poniżej obszaru prezentującego kod HTML bieżącej strony. Instrukcja `console.log(data)` wyświetla na konsoli obiekt JSON `data` pobrany z serwera.

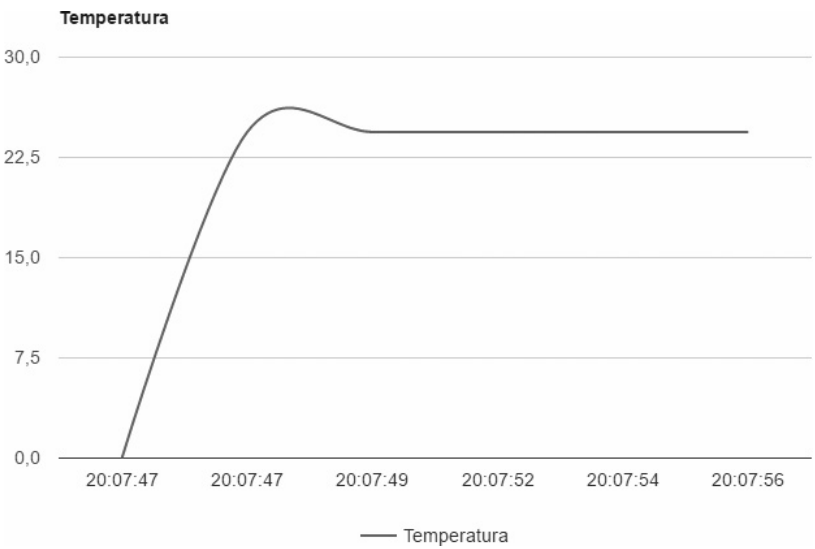
---

[3] Wszystkie przykłady zostały gruntownie przetestowane w przeglądarkach Firefox (>41) oraz Chrome (>46), dlatego też sugerujemy, by czytelnik zainstalował na swoim komputerze jedną z nich. Przykłady powinny także działać w przeglądarce Safari (>9). Jeśli ktoś koniecznie chce używać przeglądarki Internet Explorer, to musi pamiętać, że będzie potrzebował wersji IE 10 lub nowszej — w starszych wersjach tej przeglądarki przykłady nie będą działać.

[4] jQuery jest bardzo wygodną biblioteką języka JavaScript, która znacznie ułatwia wykonywanie na stronach WWW wielu czynności, takich jak korzystanie z API typu REST, manipulowanie elementami HTML, obsługa zdarzeń itd. Więcej informacji na temat tej biblioteki można znaleźć na stronie *http://jquery.com/*.

2.3.2. Część 2. Pobieranie danych z czujnika i rysowanie ich wykresu

No dobrze, ale w niektórych sytuacjach konieczne będzie wyświetlanie czegoś więcej niż jedynie ostatniej wartości odczytanej z czujnika, na przykład potrzebne będzie narysowanie wykresu tych wartości zarejestrowanych w ciągu ostatniej godziny. Otwórzmy zatem drugi plik HTML w tym ćwiczeniu (o nazwie *ex-2.2-polling-temp-chart.html*). To nieco bardziej złożony przykład, który przechowuje 10 ostatnich wartości odczytanych z czujnika temperatury i wyświetla je w formie wykresu. Kiedy otworzymy ten drugi plik w przeglądarce, na stronie zostanie narysowany wykres aktualizowany co dwie sekundy, taki jak ten przedstawiony na rysunku 2.12.



Rysunek 2.12. Co dwie sekundy do wykresu dodawana jest automatycznie nowa wartość odczytana z urządzenia

Wykres jest tworzony przy użyciu Google Charts[5] — niewielkiej i prostej biblioteki JavaScript służącej do sporządzania wszelkiego typu wykresów i diagramów. Sposób jej użycia został zaprezentowany na zamieszczonym poniżej listingu opatrzonym komentarzami.

Listing 2.2. Pobieranie danych z czujnika i prezentowanie ich na wykresie

```
$(document).ready(function () {
  var maxDataPoints = 10;
  var chart = new google.visualization.LineChart($('#chart')[0]);
  var data = google.visualization.arrayToDataTable([
    ['Czas', 'Temperatura'],
    [getTime(), 0]
  ]);
  var options = {
```

Inicjalizacja wykresu.

Utworzenie tablicy, w której będą zapisywane kolejne punkty wykresu.

Konfiguracja parametrów wykresu.

[5] *http://developer.google.com/chart/*.

```
    title: 'Temperatura',
    curveType: 'function',
    animation: {
      duration: 1000,
      easing: 'in'
    },
    legend: {position: 'bottom'}
  };

  function addDataPoint(dataPoint) {
    if (data.getNumberOfRows() > maxDataPoints) {
      data.removeRow(0);
    }
    data.addRow([getTime(), dataPoint.value]);
    chart.draw(data, options);
  }

  function getTime() {
    var d = new Date();
    return d.toLocaleTimeString();
  }

  function doPoll() {
    $.getJSON('http://devices.webofthings.io/pi/sensors/temperature/value',
      function (result) {
        addDataPoint(result);
        setTimeout(doPoll, 2000);
      });
    }

    doPoll();
  });
```

Dodanie danych kolejnego punktu do tablicy z danymi wykresu oraz usunięcie najstarszego punktu, jeśli będzie to konieczne (jeżeli liczba punktów przekroczy 10).

Przerysowanie wykresu w celu uwzględnienia nowego punktu.

Odczyt wartości z czujnika temperatury (jak w poprzednim przykładzie).

Kiedy będzie dostępna nowa wartość odczytana z czujnika, użyjemy jej w wywołaniu funkcji addDataPoint().

2.3.3. Część 3. Aktualizacja danych na bieżąco

W poprzednim przykładzie pobieranie danych z czujnika temperatury podłączonego do Raspberry Pi działało doskonale. Jednak zastosowane rozwiązanie wydawało się trochę nieefektywne, nieprawdaż? Czy nie byłoby lepiej, gdyby nasz skrypt, zamiast samodzielnie pobierać wartość czujnika co każde dwie sekundy lub jakiś inny okres, był *informowany* o wszelkich zmianach mierzonej temperatury i wyłącznie o nich?

Zgodnie z bardziej wyczerpującymi informacjami na ten temat zamieszczonymi w rozdziale 6. ta zmiana to podstawowa różnica pomiędzy modelem standardowo używanym na WWW oraz nowym, sterowanym zdarzeniami modelem działania aplikacji współpracujących z czujnikami bezprzewodowymi. Na razie przedstawimy jeden ze sposobów rozwiązania tego problemu, polegający na wykorzystaniu stosunkowo nowej technologii internetowej: *WebSocket*. Ogólnie rzecz ujmując, WebSocket to zespół prostych, lecz dających duże możliwości mechanizmów, których serwery WWW mogą używać do przekazywania powiadomień do klientów; mechanizmy te zostały wprowadzone w standardzie HTML5.

Standard WebSocket składa się z dwóch części: pierwsza z nich działa na serwerze, a druga po stronie klienta. Ponieważ w naszym przypadku część serwerowa została już zaimplementowana, musimy się zająć jedynie częścią kliencką. Kliencki API technologii

WebSocket korzysta z języka JavaScript i jest stosukowo prosty i nieskomplikowany. Poniższe dwa wiersze kodu to wszystko, czego potrzeba, by nawiązać połączenie z serwerem WebSocket i wyświetlać na konsoli otrzymywane od niego komunikaty.

Listing 2.3. Nawiązywanie połączenia z serwerem z użyciem technologii WebSocket i wyświetlanie komunikatów

```
var socket = new WebSocket('ws://ws.webofthings.io');
socket.onmessage = function (event) {console.log(event);};
```

Wróćmy do naszego przykładu i dwukrotnie kliknijmy plik *ex-2.3-websockets-temp-graph.html*, aby otworzyć go w przeglądarce. Na stronie zobaczymy dokładnie ten sam wykres co w poprzednim ćwiczeniu, jednak za kulisami będzie on działał zupełnie inaczej. Przyjrzyjmy się zatem kodowi tego przykładu, przedstawionemu na poniższym listingu 2.4.

Listing 2.4. Utworzenie połączenia WebSocket i odczytywanie zmian temperatury na bieżąco

```
var socket = new
     WebSocket('ws://devices.webofthings.io/pi/sensors/temperature');

socket.onmessage = function (event) {
  var result = JSON.parse(event.data);
  addDataPoint(result);
};

socket.onerror = function (error) {
  console.log('WebSocket error!');
  console.log(error);
};
```

Utworzenie subskrypcji WebSocket pobierającej dane z czujnika temperatury. Koniecznie należy zwrócić uwagę na zastosowanie protokołu WebSocket (ws://...).

Zarejestrowanie anonimowej funkcji, która ma być wywoływana po odebraniu komunikatu WebSocket.

Zarejestrowanie kolejnej anonimowej funkcji, która będzie wywoływana w przypadku wystąpienia błędów komunikacji WebSocket.

W tym przykładzie nie odpytujemy czujnika cyklicznie przez pobieranie jego bieżącego odczytu, lecz rejestrujemy zainteresowanie otrzymywaniem informacji o aktualizacjach mierzonej wartości — w tym celu, korzystając z API WebSocket, tworzymy subskrypcję punktu końcowego `/sensors/temperature`. Kiedy serwer będzie dysponował nową zmierzoną wartością temperatury, wyśle ją do klienta (naszej przeglądarki WWW). Zdarzenie to zostanie przechwycone przez zarejestrowaną funkcję anonimową, a najnowsza zmierzona wartość temperatury zostanie dodana do obiektu zdarzenia przekazywanego do tej funkcji jako jej parametr.

2.3.4. Wnioski

Cofnijmy się nieco i zastanówmy nad tym, co zrobiliśmy w ostatnim ćwiczeniu: udało nam się skomunikować z osadzonym urządzeniem (Raspberry Pi), które może się znajdować po drugiej stronie kuli ziemskiej (o ile ktoś nie ma szczęścia i nie mieszka w deszczowej i pięknej Anglii). Z poziomu strony WWW byliśmy w stanie pobierać dane z czujnika podłączonego do urządzenia i cyklicznie — w regularnych odstępach czasu — wyświetlać je na wykresie. To naprawdę całkiem nieźle jak na stronę WWW składającą się z 60 wierszy kodów HTML, JavaScript i CSS. Ale to nie wszystko: potrzebowaliśmy mniej niż 10 wierszy kodu JavaScript, by subskrybować powiadomienia z naszego Pi przy wy-

korzystaniu technologii WebSocket i wyświetlać temperaturę na bieżąco. W ramach rozszerzenia dla tego ćwiczenia czytelnik może spróbować napisać prostą stronę WWW, która będzie automatycznie pobierać zdjęcia robione przez naszą kamerę (sugerujemy przy tym, by nie robić tego z częstotliwością 25 klatek na sekundę!).

Jeśli to było pierwsze spotkanie czytelnika z WWW rzeczy, to w oczy powinna mu się rzucić przede wszystkim duża prostota przedstawianych przykładów. Załóżmy na chwilkę, że nasze Pi nie byłoby podłączone do internetu przy wykorzystaniu HTTP, JSON-a czy WebSocket, lecz przy użyciu „klasycznego" stosu do wymiany danych pomiędzy aplikacjami opartego na formacie XML, takiego jak DPWS (jeśli czytelnik nigdy o nim nie słyszał, to nie ma się czym przejmować, bo właśnie o to chodzi!). Najprościej rzecz ujmując, nie bylibyśmy w stanie porozumiewać się z urządzeniem bezpośrednio z przeglądarki, a przynajmniej nie tak łatwo. Konieczne byłoby napisanie aplikacji przy użyciu jakiegoś bardziej skomplikowanego języka niższego poziomu, takiego jak C lub Java. Nie można by było przy tym korzystać z powszechnie znanych i popularnych rozwiązań, jak adresy URL, języki HTML, CSS oraz JavaScript. I właśnie o to chodzi w WWW rzeczy: o zapewnienie możliwości programowania oraz dostępu do rzeczywistych przedmiotów poprzez przybliżenie ich szerokim rzeszom programistów aplikacji webowych, czyli osób mających do czynienia z rozwiązaniami i technologiami, w których obecnie wprowadzanych jest najwięcej innowacji.

Jak już wcześniej zaznaczyliśmy, z tej książki czytelnik dowie się znacznie więcej na temat sztuki tworzenia API do wchodzenia w interakcję z fizycznymi przedmiotami. W rozdziale 6. przyjrzymy się dokładniej protokołom HTTP i REST, formatowi JSON, jak również zagadnieniom związanym z interakcjami z WWW rzeczy w czasie rzeczywistym, a w rozdziale 7. wyjaśnimy, jak tworzyć bramy, które pozwolą innym protokołom i systemom korzystać z zalet, jakie zapewniają technologie związane z WWW.

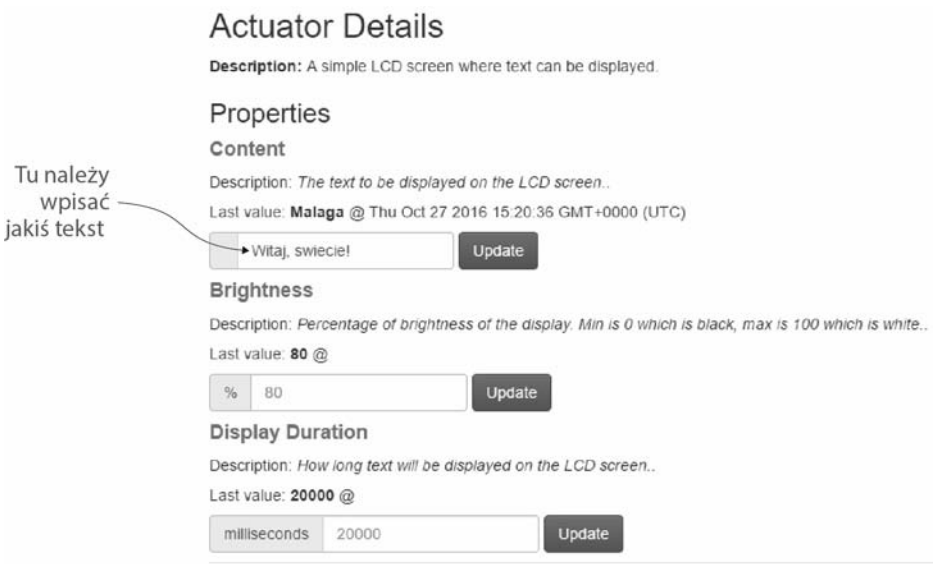
2.4. Ćwiczenie 3. Działania w realnym świecie

Do tej pory przedstawiliśmy jedynie różne sposoby „odczytu" danych czujników udostępnianych przez urządzenia podłączone do WWW. A co z operacjami „zapisu"? Na przykład moglibyśmy chcieć przesłać na urządzenie polecenie zmiany parametru konfiguracyjnego. W innych przypadkach może się pojawić konieczność kontroli serwomechanizmu (na przykład w celu otworzenia drzwi garażu lub wyłączenia świateł).

2.4.1. Część 1. Zastosowanie formularza do zmiany tekstu na wyświetlaczu

Aby pokazać, w jaki sposób można wysyłać polecenia do elementów wykonawczych i sygnalizacyjnych, w tym ćwiczeniu przedstawimy prostą stronę pozwalającą na przesyłanie tekstu, który będzie wyświetlany na ekranie LCD podłączonym do naszego Raspberry Pi. Aby szybko przetestować te możliwości funkcjonalne, wystarczy otworzyć w przeglądarce stronę WWW ekranu LCD na naszym Pi, która jest dostępna pod adresem *http://devices.webofthings.io/pi/actuators/display*.

Na tej stronie (przedstawionej na rysunku 2.13) prezentowane są różne *właściwości* wyświetlacza LED. Pierwszą z nich jest `brightness` (jasność), którą można by zmieniać, jednak nie zapewniliśmy takiej możliwości, przez co właściwość ta ma charakter danej tylko do odczytu. Kolejną właściwością jest `content` (zawartość) i to ją będziemy chcieli ustawiać w tym ćwiczeniu. I w końcu ostatnią właściwością jest `duration`, która określa, jak długo podany tekst będzie prezentowany na naszym ekranie LCD. Można teraz skorzystać z rozszerzenia Postman, aby pobrać obiekt JSON zawierający różne informacje o ekranie LCD, używając w tym celu adresu URL podanego w poprzednim akapicie i postępując zgodnie z instrukcjami zamieszczonymi w pierwszym ćwiczeniu w tym rozdziale.



Rysunek 2.13. Szczegółowe informacje o wyświetlaczu LCD wraz z polami różnych właściwości, które można ustawiać, takimi jak wyświetlany tekst

```
{
  "name": "LCD Display screen",
  "description": "A simple display that can write commands.",
  "properties": {
    "brightness": {
      "name": "Brightness",
      "timestamp": "2017-02-13T21:06:02.913Z",
      "value": 80,
      "unit": "%",
      "type": "integer",
      "description": "Percentage of brightness of the display. Min is 0 which is black,
         max is 100 which is white."
    },
    "content": {
      "name": "Content",
      "timestamp": "2017-02-13T21:06:32.933Z",
      "type": "string",
      "description": "The text to display on the LCD screen."
```

```
    },
    "duration": {
      "name": "Display Duration",
      "timestamp": "2017-02-13T21:06:02.913Z",
      "value": 5000,
      "unit": "milliseconds",
      "type": "integer",
      "read-only": true,
      "description": "The duration for how long text will be displayed on the LCD screen."
    }
  },
  "commands": [
    "write",
    "clear",
    "blink",
    "color",
    "brightness"
  ]
}
```

Oczywiście wyświetlanie czegoś w naszym biurze byłoby mało interesujące, gdybyśmy nie zapewnili możliwości obejrzenia tego, co jest pokazane na ekranie LCD. Właśnie z tego powodu dodaliśmy do naszego zestawu kamerę internetową, która pokazuje ekran LCD podłączony do naszego Pi — dzięki niej zawsze można zobaczyć, co jest w danej chwili wyświetlane. Oto adres URL zapewniający możliwość obejrzenia tego zdjęcia: *http://devices.webofthings.io/camera/sensors/picture/*. A zatem do dzieła: proszę wyświetlić tę stronę, by zobaczyć najnowsze zdjęcie wykonane przez kamerę, takie jak to przedstawione na rysunku 2.3 (aby zobaczyć najnowsze zdjęcie, należy odświeżyć stronę w przeglądarce).

A teraz spróbujemy przesłać do Pi komunikat tekstowy, który zostanie wyświetlony na podłączonym do niego ekranie LCD. Właściwość `content` zawsze zawiera bieżący komunikat wyświetlany na ekranie LCD, a zatem aby go zaktualizować, należy wygenerować żądanie typu `POST`, którego zawartością będzie treść komunikatu, jaki ma zostać wyświetlony na ekranie (na przykład `"value": "Witaj, swiecie"`). Można wypróbować to samemu, używając rozszerzenia Postman, jednak prostszym rozwiązaniem będzie skorzystanie ze strony WWW wyświetlacza dostępnego pod adresem *http://devices.webofthings.io/pi/actuators/display*. Strona ta została przedstawiona na rysunku 2.13.

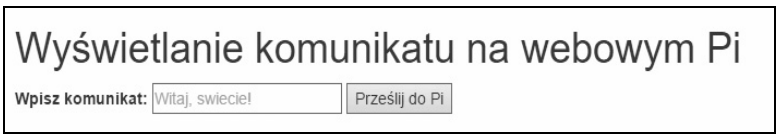
Omawiana strona prezentuje różne właściwości wyświetlacza LCD. Niektóre z nich pozwalają na ustawianie wartości, a inne są przeznaczone tylko do odczytu. My chcemy ustawić wartość właściwości `content`, spróbujmy zatem podać treść komunikatu w polu tekstowym i kliknąć przycisk *Update*. Jeśli wszystko pójdzie dobrze, to w przeglądarce powinny zostać wyświetlone przesłane dane JSON, takie jak te pokazane w poniższym przykładzie:

```
{
  "id":11,
  "messageReceived":"Lubie WoT, a nie VAT!",
  "displayInSeconds":20
}
```

Te zwrócone dane zawierają komunikat, który zostanie wyświetlony, unikalny identyfikator wiadomości oraz szacunkowe opóźnienie, z jakim treść komunikatu zostanie wyświetlona na ekranie LCD (w sekundach); dzięki tej ostatniej informacji czytelnik będzie wiedział, kiedy wyświetlić obraz z kamery, by zobaczyć podany tekst.

2.4.2. Część 2. Utworzenie własnego formularza do kontroli urządzenia

Teraz spróbujemy napisać prostą stronę HTML zawierającą formularz i pozwalającą na przesyłanie do urządzenia podłączonego do WWW wszelkiego rodzaju poleceń. Aby zobaczyć, jak taka strona może wyglądać, otwórzmy w przeglądarce plik *ex-3.1-actuator-form.html*. Postać tej przykładowej strony została przedstawiona na rysunku 2.14.



Rysunek 2.14. Ten prosty formularz HTML działający po stronie klienta pozwala na przesyłanie tekstu, który ma zostać wyświetlony na ekranie LCD podłączonym do naszego Pi

Strona zawiera pole tekstowe oraz przycisk *Prześlij do Pi*, a jej kod został zaprezentowany na zamieszczonym poniżej listingu 2.5. Dowolny tekst wpisany w polu tekstowym na tej stronie zostanie wyświetlony na ekranie LCD Raspberry Pi w naszym biurze. Prosimy przy tym o kulturalne zachowanie, gdyż API naszego Pi jest publicznie dostępny, a my nie bierzemy żadnej odpowiedzialności za to, co ludzie na nim wyświetlają.

Listing 2.5. Prosty formularz do przesyłania poleceń do elementu sygnalizacyjnego

```
<form action="http://devices.webofthings.io/pi/actuators/display/content/"
    method="post">
  <label>Wpisz komunikat:</label>
  <input type="text" name="value" placeholder="Witaj, swiecie!">
  <button type="submit">Prześlij do Pi</button>
</form>
```

To prosty formularz HTML, który przesyła dane, używając żądań HTTP typu `POST` (określonego za pomocą atrybutu `method`), pod wskazany adres URL (określony przy pomocy atrybutu `action`). Pole tekstowe nosi nazwę `value` (atrybut `name="value"`), dzięki czemu nasze Pi wie, jaki tekst ma wyświetlać. To rozwiązanie świetnie sprawdza się w przypadku prostych witryn WWW. Niestety, tym, czego nie widać, jest to, że przeglądarka nie wysyła (ani nawet nie daje takiej możliwości) danych w formacie JSON (co możemy bardzo prosto zrobić, używając rozszerzenia Postman), lecz w formacie określanym jako `application/x-www-form-urlencoded`. Pi musi rozumieć także ten format, a nie jedynie format `application/json`, by móc prawidłowo obsługiwać dane przesyłane z formularzy HTML.

Formularze HTML mogą używać tylko dwóch czasowników protokołu HTTP: `POST` oraz `GET`, nie mogą natomiast używać `DELETE` ani `PUT`. Naprawdę szkoda, że z jakichś niejasnych przyczyn zachowania zgodności wstecz nowoczesne przeglądarki WWW nie potrafią wysyłać zawartości formularzy HTML w formie obiektów JSON; ale cóż… takie jest życie!

Jak się przekonamy w dalszej części książki, zapewnienie, że wszystkie przedmioty tworzące WWW rzeczy będą w stanie odbierać i transmitować treści w formacie JSON, jest jednym z kluczowych warunków koniecznych do utworzenia prawdziwie otwartego ekosystemu. Właśnie z tego powodu pokażemy czytelnikowi, jak można przesyłać z formularzy HTML dane w formacie JSON (korzystając z połączenia technologii AJAX i języka JavaScript), gdyż stanowi to jeden z najważniejszych aspektów komunikacji z urządzeniami w WWW rzeczy.

Otwórzmy zatem plik *ex-3.2-actuator-ajax-json.html*, by przyjrzeć się temu samemu formularzowi, który w tej wersji jest obsługiwany przez rozbudowany fragment kodu JavaScript, przedstawiony na poniższym listingu:

Listing 2.6. Przesyłanie w żądaniu POST danych z formularza zapisanych w formacie JSON

```
(function($){ function processForm(e){
  $.ajax({
    url: 'http://devices.webofthings.io/pi/actuators/display/content/',
    dataType: 'json',
    method: 'POST',
    contentType: 'application/json',
    data: JSON.stringify({"value": $('#value').val()}),
    processData: false,
    success: function( data, textStatus, jQxhr ){
      $('#response pre').html( JSON.stringify( data ) );
    },
    error: function( jqXhr, textStatus, errorThrown ){
      console.log( errorThrown );
    }
  });
  e.preventDefault();
}
$('#message-form').submit(processForm);
})(jQuery);
```

Adres URL, na jaki zostanie wysłane żądanie.

Oczekiwany format danych wynikowych.

Czasownik HTTP określający sposób wysyłania żądania.

Sposób kodowania wysyłanych danych.

Wysyłane dane (faktyczna zawartość formularza).

Funkcja zwrotna wywoływana, jeśli żądanie zostanie prawidłowo obsłużone.

Funkcja zwrotna wywoływana, jeśli nie uda się prawidłowo obsłużyć żądania.

Funkcja processForm() zostanie wykonana, kiedy użytkownik naciśnie przycisk wysyłający formularz.

W powyższym przykładzie zdefiniowana została funkcja `processForm()`, która pobiera dane z formularza, zapisuje je w formie obiektu JSON, następnie przesyła do naszego Pi w żądaniu typu `POST` i w końcu wyświetla wyniki prawidłowo obsłużonego żądania (w przypadku wystąpienia błędu stosowny komunikat jest wyświetlany na konsoli). Parametr `url` określa adres URL punktu końcowego (ekranu podłączonego do naszego Pi), parametr `method` określa metodę HTTP używaną do przesyłania żądania, a parametr `contentType` — format danych przesyłanych na serwer (w tym przypadku jest to `application/json`). Ostatni wiersz kodu określa, że kliknięcia przycisku *Prześlij do Pi*, umieszczonego w formularzu `#message-form`, mają być obsługiwane przez funkcję `processForm()`.

W pliku *ex-3.2b-actuator-ajax-form.html* została umieszczona zmodyfikowana wersja tego samego przykładu, w której dane przesyłane do naszego Pi nie są zapisywane w formacie JSON, lecz w formacie `application/x-www-form-urlencoded`, czyli tym samym, który jest domyślnie używany przez formularze i został zastosowany w części 1. ćwiczenia 3.

2.4.3. Wnioski

W tym podrozdziale czytelnik poznał podstawowe sposoby wysyłania poleceń do urządzenia — przy wykorzystaniu formularza na stronie WWW, jak również przy użyciu API. Dodatkowo zamieszczone tu zostały informacje o ograniczeniach, wyzwaniach oraz problemach związanych z korzystaniem z nowoczesnej WWW (na razie nie ma się czym przejmować — w dalszej części książki pojawi się ich znacznie więcej!), a w szczególności o tym, że różne przeglądarki mogą na różne sposoby interpretować i implementować te same standardy. W końcu czytelnik dowiedział się, jak wykorzystać technologię AJAX, by ominąć wskazane wcześniej ograniczenia, oraz jak przesyłać do naszego Raspberry Pi polecenia zapisane w formacie JSON i jak zdalnie je kontrolować.

Mamy nadzieję, że po przeanalizowaniu tych ćwiczeń czytelnik przekonał się, że przesyłanie przez WWW poleceń do elementów wykonawczych i sygnalizacyjnych podłączonych do wszelkiego typu urządzeń jest bardzo proste — o ile tylko są one wszystkie podłączone do internetu i udostępniają ten sam, wspólny interfejs HTTP/JSON. Pozostaje jednak jeszcze jedno pytanie: w jaki sposób można odnaleźć urządzenia, zrozumieć ich API, określić, jakie funkcje one oferują, poznać parametry, które należy do nich przesyłać, ich typy, jednostki, ograniczenia itd.? Właśnie tymi wszystkimi kwestiami zajmiemy się w następnym podrozdziale.

2.5. Ćwiczenie 4. Informujemy świat o naszym urządzeniu

W poprzednich ćwiczeniach czytelnik dowiedział się, jak można zbadać zasoby udostępniane przez urządzenie w internecie oraz jak można ich używać w innych aplikacjach. Jednak we wszystkich tych przykładach zakładaliśmy, że czytelnik (zarówno on jako programista, jak i pisana przez niego aplikacja) *wie*, jakie pola mają zwracane obiekty JSON (na przykład te reprezentujące czujniki, elementy sygnalizacyjne czy też wykonawcze) oraz jakie jest ich znaczenie. Ale czy możliwe jest zdobycie takiej wiedzy? Co zrobić w przypadku, gdy jedyną informacją o urządzeniu, jaką dysponujemy, jest jego adres URL?

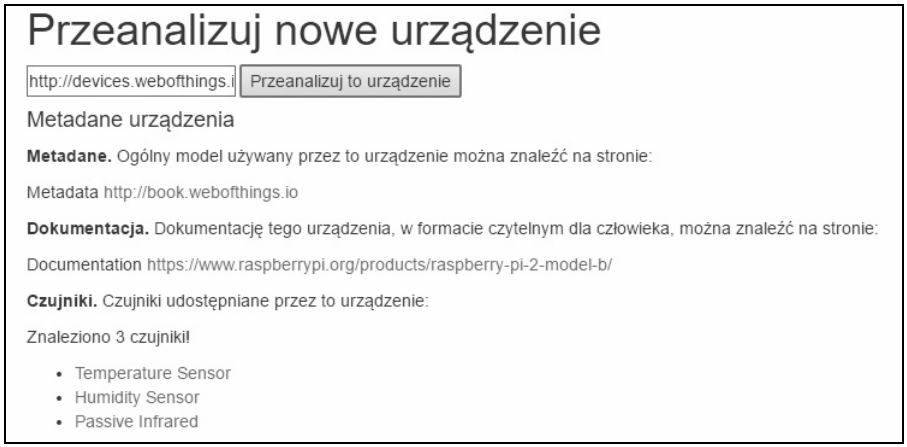
Wyobraźmy sobie, że chcielibyśmy napisać aplikację webową kontrolującą urządzenia domowej automatyki podłączone do sieci lokalnej. W jaki sposób moglibyśmy zapewnić, że aplikacja ta zawsze będzie działała prawidłowo, nawet na sieci należącej do kogoś innego, w przypadku gdy nie będziemy posiadali żadnej wiedzy o podłączonych do tej sieci urządzeniach?

Po pierwsze, konieczne będzie odnalezienie wszystkich urządzeń podłączonych do sieci (jest to tak zwany problem *wykrywania urządzeń*). Innymi słowy, trzeba się dowiedzieć, w jaki sposób aplikacja webowa może określić główne adresy URL wszystkich urządzeń w okolicy.

Po drugie, nawet jeśli uda się nam (w jakiś magiczny sposób) uzyskać główne adresy URL wszystkich okolicznych urządzeń, które mogą współpracować z WWW rzeczy, musimy odpowiedzieć sobie na pytanie, jak nasza aplikacja może się „dowiedzieć”, jakie czujniki, elementy sygnalizacyjne i wykonawcze są do tych urządzeń podłączone, jakich formatów należy używać podczas komunikacji z nimi, jakie jest znaczenie poszczególnych urządzeń, jakie są ich właściwości, pola itd.

Jak się przekonaliśmy w ćwiczeniu 2. (w punkcie 2.3.2), znając główny adres URL urządzenia, bez trudu można przejrzeć jego zasoby, pobrać dane dotyczące tego urządzenia oraz podłączonych do niego czujników, udostępnianych usług itd. Dla nas to proste zadanie, gdyż jesteśmy ludźmi, spróbujmy jednak wyobrazić sobie, co by się stało, gdybyśmy mieli do dyspozycji dokument JSON zawierający niezrozumiałe słowa i znaki i nie mieli żadnej dokumentacji wyjaśniającej jego znaczenie — w jaki sposób moglibyśmy się dowiedzieć, co takie urządzenie robi? I skąd moglibyśmy mieć pewność, że to w ogóle jest jakieś urządzenie?

Otwórzmy zatem plik *ex-4-parse-device.html*, aby wyświetlić formularz z wpisanym adresem URL naszego Pi (patrz rysunek 2.15), a następnie, by przekonać się, co potrafi ta strona, naciśnijmy przycisk *Przeanalizuj to urządzenie*.



Rysunek 2.15. Miniprzeglądarka, która analizuje metadane urządzenia i wyświetla uzyskane wyniki

Kod JavaScript umieszczony w pliku *ex-4-parse-device.html* odczytuje główny dokument Raspberry Pi (zapisany w formacie JSON) i na jego podstawie generuje prosty raport o urządzeniu i jego czujnikach, uzupełniony o odnośniki do dokumentacji urządzenia. W pierwszej kolejności przeanalizujmy kod HTML używany do generowania raportu; kod ten został przedstawiony na rysunku 2.7.

Listing 2.7. Prosta przeglądarka urządzeń

```
<form id="message-form">
  <input type="text" id="host" name="host" value="http://devices.webofthings.io/pi"
placeholder="The root URL of a WoT device"/>
  <button type="submit">Przeanalizuj to urządzenie</button>
</form>
<h4>Metadane urządzenia</h4>
<p>
<b>Metadane.</b> Ogólny model używany przez to urządzenie można znaleźć na stronie: <div
id="meta"></div>
</p>
<p>
<b>Dokumentacja.</b>
```

```
Dokumentację tego urządzenia, w formacie czytelnym dla człowieka, można znaleźć na stronie:
<div id="doc"></div>
</p>
<p>
<b>Czujniki.</b> Czujniki udostępniane przez to urządzenie:
<div id="sensors"></div>
</p>
<ul id="sensors-list">
</ul>
```

Na początku tego przykładu umieszczone zostały formularz, w którym można wpisać adres URL urządzenia, oraz przycisk powodujący wygenerowanie żądania. Poniżej znajduje się grupa elementów tekstowych HTML (z identyfikatorami `meta`, `doc` itd.), stanowiących miejsca, które potem zostaną uzupełnione informacjami uzyskanymi z urządzenia. Przeanalizujmy teraz sposób przetwarzania żądania AJAX, przedstawiony na poniższym listingu 2.8.

Listing 2.8. Pobieranie i przetwarzanie metadanych urządzenia zapisanych w formie danych JSON z użyciem żądań AJAX

```
(function ($) {
  function processForm(e) {

    var sensorsPath = '';

    $.ajax({
      url: $('#host').val(),
      method: 'GET',
      dataType: 'json',
      success: function (data) {
        $('#meta').html(data.links.meta.title + " <a href=\""
          + data.links.meta.rel + "\">" + data.links.meta.rel + "</a>");
        $('#doc').html(data.links.doc.title + " <a href=\"" + data.links.doc.rel
          + "\">" + data.links.doc.rel + "</a>");

        sensorsPath = data.url + data.resources.sensors.url;

        $.ajax({
          url: sensorsPath,
          method: 'GET',
          dataType: 'json',
          success: function (data) {
            var sensorList = "";

            $('#sensors').html("Znaleziono " + Object.keys(data).length + " czujniki!");

            for (var key in data) {
              sensorList = sensorList + "<li><a href=\"" + sensorsPath
                + key + "\">" + data[key].name + "</a></li>";
            }

            $('#sensors-list').html(sensorList);
          },
          error: function (data, textStatus, jqXHR) {
```

Generacja żądania GET, które pobierze dane urządzenia w formacie JSON i przetworzy je.

Aktualizacja elementów "meta" i "doc" poprzez zapisanie w nich odnośników przechowywanych w odebranym dokumencie JSON.

Zapisanie adresu URL zasobu czujników.

Generacja żądania GET pobierającego listę czujników podłączonych do urządzenia.

Funkcja zwrotna przetwarzająca dokument JSON z danymi o czujnikach; obiekt JSON z tymi informacjami jest zapisany jako 'data'.

Pętla przetwarzająca wszystkie czujniki.

Wyświetlenie listy w kodzie HTML strony.

```
            console.log(data);
          }
        });
      },
      error: function (data, textStatus, jqXHR) {
        console.log(data);
      }
    });

    e.preventDefault();
  }

  $('#message-form').submit(processForm);
})(jQuery);
```

Jak pokazuje powyższy listing, w pierwszej kolejności ustawiany jest adres URL pozwalający na pobranie głównego dokumentu urządzenia, zwracanego w formacie JSON (jest on pobierany przy użyciu wywołania `$('#host').val()`). Jeśli żądanie zostanie prawidłowo obsłużone, to zostanie wywołana funkcja zwrotna `success`, a przekazana do niej zmienna `data` będzie zawierać dokument JSON z danymi urządzenia (ten sam, który został przedstawiony w kroku 2. w punkcie 2.2.2). Te dane JSON są następnie analizowane, a funkcja pobiera z nich interesujące nas informacje — w naszym przypadku chodzi o element `links` zwróconego obiektu JSON (stąd zastosowanie wyrażenia `data.links`), zawierający odnośniki pozwalające na uzyskanie dokładniejszych informacji o urządzeniu. W elemencie tym znajdują się dane o następującej postaci:

```
"links": {
  "meta": {
    "rel": "http://book.webofthings.io",
    "title": "Metadata"
  },
  "doc": {
    "rel": "https://www.raspberrypi.org/products/raspberry-pi-2-model-b/",
    "title": "Documentation"
  },
  "ui": {
    "rel": ".",
    "title": "User Interface"
  }
}
```

Szczególnie interesujące są element `meta`, który zawiera odnośnik (zapisany jako wartość elementu `rel`) do ogólnego modelu używanego przed dane urządzenie (opisuje on gramatykę stosowaną do przedstawienia elementów danego urządzenia), oraz element `doc`, który z kolei zawiera odnośniki do dokumentacji urządzenia przeznaczonej dla ludzi i opisującej znaczenie (semantykę) urządzenia oraz prezentującej szczegółowe informacje o urządzeniu (jakie czujniki są do niego podłączone i co one mierzą).

Dokument metadanych przedstawiony w powyższym przykładzie nie jest niczym więcej jak zrozumiałym dla komputerów dokumentem modelu zapisanym w formacie JSON, pozwalającym użytkownikom na opisanie urządzenia WoT w strukturalny sposób, z uwzględnieniem definicji logicznych elementów, które muszą posiadać wszystkie urządzenia WoT.

Gdyby setki producentów urządzeń używały tego samego modelu danych do przekazywania informacji o usługach udostępnianych przez ich urządzenia, oznaczałoby to, że dowolna aplikacja mogłaby odczytać i przeanalizować ten plik JSON zwracany przez dowolne urządzenie i zrozumieć, co ono udostępnia (ile czujników jest do niego podłączonych, jak się te czujniki nazywają, jakie mają ograniczenia, jakiego są typu itd.).

No dobrze, a co z samymi czujnikami, elementami sygnalizacyjnymi i wykonawczymi? Element `links` zawiera jedynie metadane (takie jak dokumentacja) dotyczące urządzenia, ale nie jego rzeczywistą zawartość. Aby uzyskać informacje o czujnikach podłączonych do urządzenia, konieczne byłoby przeanalizowanie pola `sensors` elementu `resources`, a tak się składa, że właśnie za to odpowiada drugie wywołanie AJAX, które używając żądania `GET`, pobiera dane o czujnikach urządzenia. Kiedy dokument JSON z danymi czujników zostanie już pobrany, poszczególne czujniki są pobierane w pętli, która — korzystając z poniższego wzorca — generuje listę HTML z odnośnikami do poszczególnych czujników:

```
<li><a href=\""+sensorsPath+key+"\">"+data[key].name+"</a></li>
```

W tym przypadku `sensorsPath` jest adresem URL do zasobu czujników (w naszym przykładzie będzie to adres *http://devices.webofthings.io/pi/sensors*), do którego dodajemy identyfikator (`key`) każdego czujnika, a wyrażenie `data[key].name` reprezentuje nazwę danego czujnika.

2.5.1. Wnioski

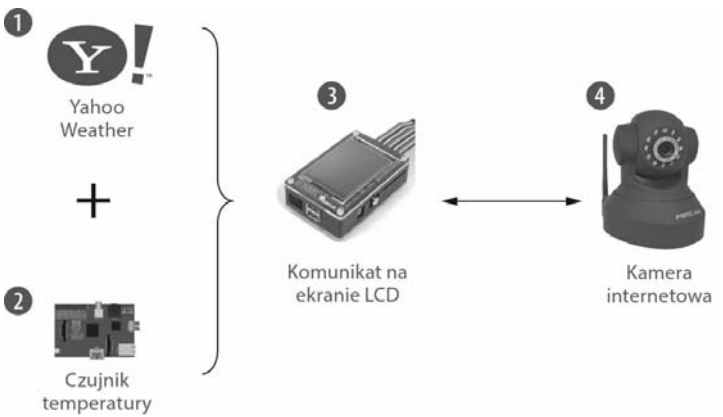
Jeśli czytelnik nie zrozumiał wszystkich szczegółów poprzednich ćwiczeń, to nic nie szkodzi — nie ma się czym przejmować! Najważniejsze jest to, że udało się przejść pierwszy, przyspieszony kurs semantycznej WWW (ang. *Semantic Web*), a może raczej kurs związany z trudnymi problemami, jakie stara się ona rozwiązywać. Powodem, dla którego czytelnik zapewne o niej wiele słyszał, lecz nie miał do tej pory okazji jej zobaczyć lub używać (jak również zrozumieć), jest to, że stawia ona przed komputerami i ludźmi piszącymi działające na tych komputerach programy bardzo trudne wyzwanie: niby w jaki sposób można wyjaśnić komputerom realny świat wraz z jego problemami egzystencjalnymi? No cóż, okazuje się, że jeszcze nie można nauczyć swojego komputera filozofii. Jednak, jak pokazaliśmy w tym rozdziale i pokażemy jeszcze raz nieco dokładniej w rozdziale 8., istnieje kilka magicznych sztuczek, które można z powodzeniem zastosować, by sprawić, że WWW i komputery będą trochę sprytniejsze.

A zatem pokazaliśmy już, jak urządzenia mogą informować o swoich możliwościach, danych i usługach, i to w sposób zrozumiały dla innych komputerów. Wykorzystanie do tego celu dobrze znanych wzorców stosowanych na WWW pozwoliło nam bez trudu napisać aplikację internetową będącą w interakcji z naszym urządzeniem. Niestety, nie ma pojedynczego standardu pozwalającego na definiowanie tych informacji w uniwersalny sposób, a zastosowanie formatu JSON jest rozwiązaniem wypracowywanym przez lata metodą prób i błędów. Aby udało się wyzwolić pełny potencjał WWW rzeczy, musimy mieć możliwość definiowania wszystkich szczegółowych informacji o obiektach z wykorzystaniem jednego modelu danych, posiadającego przejrzystą semantykę, którą bez ryzyka pojawienia się jakichkolwiek nieścisłości mogłyby zrozumieć wszystkie komputery i aplikacje. W rozdziale 8. pokażemy znacznie bardziej szczegółowo, jak można to osiągnąć, korzystając z technologii WWW oraz lekkich technologii semantycznej WWW.

2.6. Ćwiczenie 5. Tworzenie pierwszej fizycznej aplikacji typu mashup

W poprzednich ćwiczeniach przedstawiony został sposób odwoływania się do urządzenia podłączonego do WWW, przeglądania danych i usług, jakie to urządzenie udostępnia, oraz odczytu i zapisu tych danych. Natomiast w tym ćwiczeniu pokażemy czytelnikowi, jak może napisać swoją pierwszą aplikację typu mashup. Termin *mashup* pochodzi ze świata hip-hopu, w którym zaczęto nim określać piosenki składające się z fragmentów innych utworów. Analogicznie webowa aplikacja typu mashup pobiera dane z różnych źródeł, przetwarza je i łączy, tworząc w ten sposób coś zupełnie nowego.

W tym ćwiczeniu utworzymy nie tylko webową aplikację typu mashup, lecz także *aplikację fizyczną*, czyli pobierającą dane z czujników rzeczywistych urządzeń podłączonych do WWW. Konkretnie rzecz biorąc, w tym ćwiczeniu spróbujemy pobrać dane o lokalnej temperaturze udostępniane przez usługę Yahoo! Weather, porównać je z wartościami mierzonymi przez czujnik temperatury podłączony do Raspberry Pi w naszym biurze w Londynie i wyświetlić wyniki na ekranie LCD, który także jest podłączony do naszego Pi. I w końcu, aby przekonać się, jak wygląda komunikat wynikowy, skorzystamy z API kamery internetowej, żeby zrobić zdjęcie ekranu LCD i opublikować je na stronie WWW! Cały ten proces został zilustrowany na rysunku 2.16.



Rysunek 2.16. Fizyczna aplikacja typu mashup. W pierwszej kolejności (1) pobieramy lokalną temperaturę z serwisu Yahoo! Weather, a następnie temperaturę zmierzoną przez czujnik podłączony do naszego Pi (2). Obie te temperatury są ze sobą porównywane, a komunikat z wynikiem jest wyświetlany na ekranie LCD podłączonym do naszego Pi (3). Kiedy komunikat jest już widoczny na ekranie, kamera internetowa robi jego zdjęcie (4) i wyświetla na stronie aplikacji

A zatem zacznijmy od otworzenia pliku *ex-5-mashup.html* w przeglądarce i edytorze. Kod JavaScript tej aplikacji, przedstawiony na listingu 2.9, jest nieco dłuższy niż kod wcześniejszych rozwiązań, jednak wcale nie jest bardziej złożony.

Listing 2.9. Funkcja mashup

```
$(document).ready(function () {
  var rootUrl = 'http://devices.webofthings.io';

  function mashup(name, location) {
    var yahooUrl = "https://query.yahooapis.com/v1/public/yql?q=select item from
    weather.forecast where woeid in (select woeid from geo.places(1) where text='" +
    location + "') and u='c'&format=json";
    $.getJSON(yahooUrl, function (yahooResult) {
      var localTemp = yahooResult.query.results.channel.item.condition.temp;
      console.log('Local @ ' + location + ': ' + localTemp);
      $.getJSON(rootUrl + '/pi/sensors/temperature', function (piResult) {
        console.log('Pi @ London: ' + piResult.value);
        publishMessage(prepareMessage(name, location, localTemp, piResult.value));
      });
    });
  }

  function publishMessage(message) {
    $.ajax(rootUrl + '/pi/actuators/display/content', {
      data: JSON.stringify({"value": message}),
      contentType: 'application/json',
      method: 'POST',
      success: function (data) {
        $('#message').html('Opublikowano na LCD: ' + message);
        $('#wait').html('Zdjęcie z kamery internetowej z Twoim komunikatem zostanie
        wyświetlone poniżej za ' + (data.displayInSeconds+2) + ' sekund.');
        console.log('Zdjęcie zostanie zrobione za ' + (data.displayInSeconds+2) + ' sekund...');
        setTimeout(takePicture, (data.displayInSeconds+2) * 1000);
      }
    });
  }

  function prepareMessage(name, location, localTemp, piTemp) {
    return name + '@' + location + ((localTemp < piTemp) ? ' < ' : ' > ') + piTemp;
  }

  function takePicture() {
    $.ajax({
      method: 'GET',
      url: rootUrl + '/camera/sensors/picture/',
      dataType: 'json',
      success: function (data) {
        console.log(data);
        $('#camImg').attr('src', data.value);
      },
      error: function (err) {
        console.log(err);
      }
    });
  }

  mashup('Renata', 'Gliwice, PL');
});
```

W pierwszej kolejności pobranie z serwisu Yahoo! temperatury mierzonej w miejscu, w którym znajduje się użytkownik.

Przygotowanie treści komunikatu i zastosowanie go do aktualizacji tekstu wyświetlanego na ekranie LCD.

Następnie pobranie temperatury zmierzonej przez Pi w naszym biurze w Londynie.

Wysłanie żądania POST z komunikatem do ekranu LCD.

Generacja tekstu do wyświetlenia, zawierającego imię użytkownika, jego lokalizację oraz temperaturę zmierzoną przez Pi.

Ustawienie czasomierza, który po upływie N sekund (kiedy ekran LCD zostanie już zaktualizowany) wywoła funkcję takePicture().

Pobranie zdjęcia zrobionego przez kamerę internetową w naszym biurze.

Aktualizacja znacznika HTML <img> i zapisanie w nim adresu URL zrobionego zdjęcia.

Funkcja `mashup()` odpowiada za wykonywanie różnych części naszej aplikacji. Pobiera ona dwa parametry: pierwszym z nich jest nazwa użytkownika, a drugim miejsce, w którym przebywa, zapisane w formacie `nazwa_miasta, kod_kraju` (na przykład Gliwice, PL; London, UK; New York, US). Następnie funkcja gromadzi niezbędne dane, wykonując w tym celu dwa żądania HTTP `GET`, z których każde prosi o przekazanie danych w formacie `application/json`. Pierwsze wywołanie korzysta z API usługi Yahoo! Weather, który — na podstawie podanej lokalizacji — zwraca informacje o bieżącej pogodzie i temperaturze w danym miejscu.

Po poprawnym obsłużeniu tego żądania (czyli wykonaniu określonej funkcji anonimowej) zostaje wywołana druga funkcja, która pobiera najnowszą wartość temperatury zmierzoną przez czujnik podłączony do naszego Pi; w tym celu wykonywane są te same operacje, które zostały opisane w punkcie 2.3.1.

Następnie wywoływana jest funkcja `prepareMessage()`, która formatuje komunikat i przekazuje go do kolejnej funkcji — `publishMessage()`. Ta ostatnia funkcja, korzystając z technologii AJAX, wykonuje żądanie HTTP `POST` i przekazuje w nim dane JSON zawierające komunikat do wyświetlenia na ekranie LCD. W tym przypadku wykonywane czynności są analogiczne do tych opisanych w ćwiczeniu 3., „Działania w realnym świecie".

Ponieważ wiadomość musi poczekać w kolejce, zanim zostanie wyświetlona, funkcja ustawia czasomierz, który z odpowiednim opóźnieniem wywoła funkcję `takePicture()`. Ta ostatnia funkcja wykonuje finalne żądanie HTTP `GET`, które pobiera zdjęcie ekranu LCD zrobione przez kamerę internetową. Zwrócone zdjęcie jest następnie dynamicznie dodawane do znacznika `<img>` na wyświetlanej stronie WWW.

Aby rozpocząć tę sekwencję rzeczywistych i wirtualnych zdarzeń, trzeba zrobić bardzo niewiele — wystarczy zmienić w edytorze wywołanie funkcji `mashup(x, y)`, podając w nim swoje własne imię oraz dane o miejscowości, w której się znajdujemy. Na przykład Renata z Gliwic w Polsce mogłaby wywołać tę funkcję w następujący sposób:

```
mashup('Renata', 'Gliwice, PL')
```

Teraz pozostaje już jedynie otworzyć plik w przeglądarce i gotowe! W ciągu kilku sekund zostanie w niej wyświetlone zdjęcie zrobione przez naszą kamerę internetową, prezentujące komunikat z podanym imieniem, wyświetlony na ekranie Raspberry Pi w naszym biurze.

2.6.1. Wnioski

W tym ćwiczeniu czytelnik stworzył webową, fizyczną aplikację typu mashup, korzystającą z danych pochodzących z różnych źródeł — fizycznych oraz wirtualnych działających w czasie rzeczywistym — i używającą prostego algorytmu w celu określenia, czy pogoda w miejscowości wskazanej przez użytkownika jest lepsza niż u nas (choć konkurowanie z pogodą londyńską z założenia jest trochę niesprawiedliwe). Zastanówmy się przez chwilę nad tą aplikacją. Do jej działania niezbędne są czujnik temperatury podłączony do urządzenia osadzonego, kamera internetowa, ekran LCD oraz internetowa usługa meteorologiczna; mimo to byliśmy w stanie stworzyć zupełnie nową aplikację, pisząc w tym celu niespełna 80 wierszy kodów HTML i JavaScript… i to włącznie z interfejsem

użytkownika! W dalszej części książki czyteln`ik dowie się znacznie więcej o aplikacjach tego typu, zwłaszcza w rozdziale 10., w którym przedstawimy różne dostępne narzędzia i techniki służące do tworzenia tych aplikacji.

2.7. Podsumowanie

- W tym rozdziale czytelnik miał okazję po raz pierwszy odbyć praktyczne spotkanie z urządzeniami podłączonymi do WWW działającymi w dowolnym miejscu na świecie; w jego trakcie mógł przeglądać ich metadane, zawartość, czujniki, elementy wykonawcze itd.
- Po zasobach urządzeń podłączonych do WWW można się poruszać tak jak po stronach witryn WWW. Dane zbierane z czujników w czasie rzeczywistym można pobierać, używając protokołu HTTP lub API WebSocket, podobnie jak wszelkie inne treści udostępniane w internecie.
- Podstawowe API związane z protokołem HTTP można zrozumieć znacznie łatwiej i szybciej niż przeróżne złożone protokoły powszechnie stosowane w internecie rzeczy.
- W ciągu zaledwie kilku minut czytelnik był w stanie opanować odczytywanie i zapisywanie danych na zdalnym urządzeniu przy użyciu żądań HTTP i rozszerzenia Postman do przeglądarki Chrome.
- Podłączanie realnego świata do WWW pozwala na błyskawiczne prototypowanie interaktywnych aplikacji z użyciem jedynie kilku wierszy kodów HTML i JavaScript.
- Kiedy dane i usługi oferowane przez różne urządzenia zostaną udostępnione jako zasoby webowe, pojawi się możliwość bardzo prostego tworzenia aplikacji typu mashup integrujących treści pochodzące z różnych źródeł.

Mamy nadzieję, że to pierwsze spotkanie z WWW rzeczy spodobało się czytelnikowi na tyle, by zachęcić go do przeczytania dalszej części książki i nauczenia się sposobów implementacji podobnych rozwiązań we własnych urządzeniach. W następnych rozdziałach powiemy, jak można zaimplementować język JavaScript na urządzeniach, oraz zamieścimy krótki i przystępny opis środowiska Node.js. Później zajmiemy się zagadnieniami związanymi z konfigurowaniem własnych urządzeń i zapewnieniem możliwości podłączania ich do WWW rzeczy. Pokażemy także, jak pisać i wdrażać aplikacje Node.js na urządzeniach Raspberry Pi, tak by czytelnik potrafił przygotować swoje pierwsze urządzenie podłączone do WWW rzeczy oraz dostosować przykłady prezentowane w tej książce do własnych potrzeb i zastosowań.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

T

U

W

Z

Ź

Ż

Internet rzeczy: fascynująca przyszłość, która zaczyna się dziś!

Internet rzeczy (Internet of Things – IoT) przynosi kolejną technologiczną rewolucję: oto coraz więcej przedmiotów, także sprzętów codziennego użytku, dzięki wbudowanym czujnikom i dostępowi do sieci może komunikować się z człowiekiem i z innymi przedmiotami. Możliwości takiej sieci są niewyobrażalne. Inżynierowie, którzy zajmują się tą koncepcją, rozwijają wiele różnych standardów mających służyć integracji IoT. Okazuje się jednak, że w chaosie konkurujących rozwiązań brakuje spojrzenia na internet rzeczy z nieco szerszej perspektywy, która pozwoliłaby na opracowanie pragmatycznej i strukturalnej metodologii tworzenia urządzeń i usług IoT.

Niniejszą książkę napisano dla osób, które zaczynają przygodę z internetem rzeczy. Zawarto tu informacje niezbędne do tworzenia prototypów urządzeń, produktów i aplikacji z wykorzystaniem infrastruktury WWW. Przedstawiono obszerne wprowadzenie w zagadnienia dotyczące internetu rzeczy. Znalazły się tu informacje o urządzeniach, czujnikach, standardach i potrzebnych narzędziach. Szczególnie dokładnie przedstawiono instrumentarium niezbędne do budowania WWW rzeczy — czyli warstwy aplikacji internetu rzeczy. Książka ta pozwoli na zrozumienie problemów dzisiejszego internetu rzeczy, poznanie dostępnych technik i ich wykorzystywanie.

W tej książce znajdziesz:

- omówienie koncepcji WWW rzeczy i internetu rzeczy
- sposoby wykorzystania Node.js do implementacji WWW rzeczy
- kwestie związane ze stosowaniem protokołu HTTP oraz API typu RESTful
- metody integracji BeagleBone, Intel Edison oraz Arduino z internetem rzeczy
- sposoby łączenia urządzeń i czujników (GPIO) z Raspberry Pi
- informacje o protokołach takich jak MQTT i CoAP i integracji ich z siecią rzeczy

Dr Dominique D. Guinard — jest pionierem koncepcji architektury internetu rzeczy. Uczestniczył w wielu projektach związanych z tym zagadnieniem: badał duże sieci RFID, zajmował się telefonami komórkowymi jako bramami IoT, a także integrował sieci czujników z oprogramowaniem SAP.

Dr Vlad M. Trifa — jest uznanym ekspertem w dziedzinie rozproszonych rozwiązań pomiarowych, a także integracji urządzeń interaktywnych z aplikacjami korporacyjnymi. Zajmował się bioakustyką, przetwarzaniem sygnałów, interakcjami robotów humanoidalnych i sieciami neuronowymi.

Helion
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

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

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

MANNING
ISBN 978-83-283-2968-3
cena: 67,00 zł