Od implementacji prostych aplikacji do budowy bogatych grafów powiązań społecznościowych

Programowanie

aplikacji na serwisy społecznościowe

HELION

O'REILLY® | YAHOO! PRESS

Jonathan LeBlanc

Tytuł oryginału: Programming Social Applications: Building Viral Experiences with OpenSocial, OAuth, OpenID, and Distributed Web Framework

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 978-83-246-3944-1

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/prapse.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 6.

Aktywność użytkowników, publikowanie powiadomień aplikacji i żądanie danych w kontenerze OpenSocial

Do największych wyzwań stojących przed programistami budującymi aplikacje społecznościowe należy właściwa promocja tych aplikacji i efektywne korzystanie z zewnętrznych źródeł danych. Prawidłowe stosowanie zewnętrznych źródeł danych jest warunkiem utworzenia bogatego zbioru funkcji, które przyciągną uwagę użytkowników i pozwolą zbudować bazę lojalnych klientów.

Wielu programistów traktuje kwestie publikowania komunikatów aplikacji i analizy aktywności jako nieistotny szczegół — uważają oni, że architektura, która ma na celu zwiększanie bazy aktywnych użytkowników, nie ma większego znaczenia. Okazuje się jednak, że standardowe metody promocji aplikacji oferowane przez kontenery, czyli na przykład galerie, nie stanowią efektywnych form zachęcania użytkowników do instalowania kolejnych rozwiązań. W wielu przypadkach galerie są przepełnione i nieczytelne — nierzadko obejmują tysiące aplikacji uporządkowanych w taki sposób, aby na najcenniejszych, najwyższych pozycjach znajdowały się najbardziej popularne produkty. Z perspektywy nowego programisty taki kształt galerii oznacza nie tylko konieczność konkurowania z wieloma aplikacjami, ale też wizję startu z najmniej atrakcyjnej pozycji. Sytuacja przypomina trochę problem historii kredytowej — brak kredytów jest często traktowany na równi z nieterminowo spłacanymi kredytami.

Warto więc rozważyć promocję aplikacji z wykorzystaniem aktywności samych użytkowników, przy zastosowaniu przemyślanego modelu udostępniania informacji o produktach. Takie rozwiązanie umożliwia przekazywanie linków do aplikacji bezpośrednio w strumieniu codziennej aktywności użytkownika. Innym czynnikiem ułatwiającym przyciąganie uwagi użytkowników jest regularnie odświeżana, atrakcyjna treść aplikacji (uzyskiwana za pośrednictwem żądań danych). W ten sposób można nie tylko podnieść liczbę użytkowników instalujących aplikację, ale także zwiększyć ich aktywność.

Czego nauczysz się w tym rozdziale?

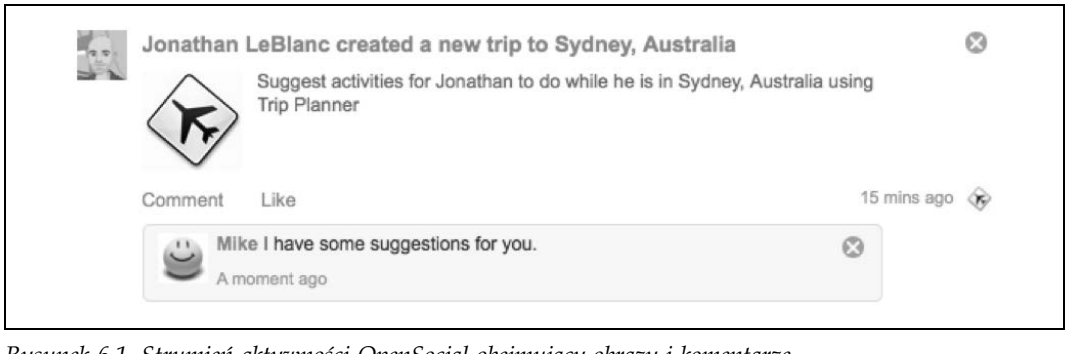
W rozdziale 5. omówiliśmy metody personalizacji i dostosowywania aplikacji na podstawie informacji zawartych w profilu użytkownika oraz techniki promocji z wykorzystaniem listy znajomych użytkownika. W tym rozdziale wspomniane zagadnienia zostaną dodatkowo rozwinięte — skoncentrujemy się na reprezentacji aktywności użytkowników w kontenerze OpenSocial i na technikach tworzenia aplikacji oferujących szerokie możliwości konfiguracji i dostosowywania (dzięki bogatym źródłom danych budowanym przy użyciu żądań do źródeł zewnętrznych). W tym rozdziale zostaną omówione następujące zagadnienia:

- personalizacja stanu aplikacji pod kątem użytkownika na podstawie jego aktywności społecznościowej;
- poszerzanie bazy użytkowników poprzez generowanie czynności;
- wyjaśnienie modeli pasywnego i bezpośredniego publikowania powiadomień aplikacji;
- generowanie żądań danych niezbędnych do budowy bogatych źródeł danych i zwiększania liczby aktywnych użytkowników;
- tworzenie uwierzytelnionych (podpisanych) żądań danych dla zapewnienia bezpieczeństwa (poprzez weryfikację danych uwierzytelniających i źródeł danych).

Po opanowaniu wymienionych zagadnień i technik będzie można przystąpić do budowy aplikacji społecznościowych dobrze przygotowanych do promocji wśród użytkowników.

Promocja aplikacji za pomocą strumienia aktywności w kontenerze OpenSocial

Jednym z najskuteczniejszych narzędzi w rękach programistów aplikacji społecznościowych jest możliwość wysyłania powiadomień (aktualizacji) do **strumienia aktywności** użytkownika. Strumień aktywności (patrz rysunek 6.1) jest centralnym obszarem powiadomień kierowanych do użytkownika aplikacji i jego znajomych. Strumień aktywności stanowi też główny kanał komunikacji z użytkownikami kontenera. Za pośrednictwem tego medium można promować aplikację poprzez rozsyłanie zachęt trafiających do jak największej liczby użytkowników. Oznacza to, że strumień aktywności umożliwia nieporównanie skuteczniejsze zwiększanie liczby użytkowników aplikacji niż zwykła galeria aplikacji.



Rysunek 6.1. Strumień aktywności OpenSocial obejmujący obrazy i komentarze

W przypadku większości kontenerów aplikacji społecznościowych elementy strumienia aktywności obejmują następujące dane:

- tytuł opisujący czynność użytkownika;
- link do źródła aktualizacji (powiadomienia), na przykład do samej aplikacji;
- opis obejmujący dodatkowe informacje na temat danej aktualizacji lub zachęta do działania (na przykład do zainstalowania aplikacji przez pozostałych użytkowników);
- opcjonalne elementy multimedialne, jak animacja czy obraz, które mogą dodatkowo przyciągnąć uwagę użytkowników;
- komentarze lub oznaczenia „lubię to" od znajomych użytkownika.

Tylko dobre zrozumienie składników strumienia aktywności umożliwi programiście pełne wykorzystanie potencjału tego strumienia. Standard OpenSocial udostępnia dwa narzędzia do operowania na powiadomieniach w strumieniu aktywności — pierwsze z nich umożliwia programistom wykorzystywanie istniejących elementów strumienia do personalizacji aplikacji, drugie umożliwia generowanie nowych elementów strumienia w celu przyciągania nowych użytkowników lub zwiększania zaangażowania dotychczasowych.

Personalizacja aplikacji na podstawie powiadomień w strumieniu aktywności

Jeśli profil obejmuje dane wybrane przez samego użytkownika, które w dodatku odzwierciedlają sposób postrzegania użytkownika przez niego samego, strumień aktywności dobrze ilustruje to, co robi i co lubi ten użytkownik. Strumień aktywności obejmuje takie informacje jak wykaz instalowanych aplikacji, lista aktualizacji (powiadomień) tych aplikacji oraz informacje o statusie i profilu. Strumień aktywności jest też bezcennym źródłem dodatkowych danych, które pozwalają lepiej ocenić internetowe zwyczaje, preferencje i uprzedzenia użytkownika. Strumień aktywności stosowany łącznie z informacjami zawartymi w profilu użytkownika stanowi dla programisty wprost doskonałą okazję do przygotowania treści i reklam pod kątem konkretnego użytkownika.

Strumień aktywności użytkownika jest jednym z najcenniejszych źródeł informacji w każdym serwisie społecznościowym. Za pośrednictwem tego strumienia programista ma dostęp do takich informacji jak to, kiedy użytkownik wysyła wiadomości, z kim się kontaktuje, co robi i z jakich aplikacji korzysta.

Specyfikacja OpenSocial definiuje standardowe metody uzyskiwania tych szczegółowych danych:

```
// uzyskuje dane o aktywności bieżącego użytkownika
osapi.activities.get({userId: '@viewer ', count: 20}).execute(function(result){
   if (!result.error){
      var activities = result.list;
      var html = ' ';

      // buduje tytuł i adres URL dla każdej odkrytej czynności
      for (var i = 0; i < activities.length; i++){
         html += 'Tytuł czynności: ' + activities[i].title +
                 'Adres URL czynności: ' + activities[i].url;
      }
   }
});
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

W ramach tego żądania wywołano metodę `osapi.activities.get(...)`, aby zasygnalizować potrzebę pobrania strumienia aktywności użytkownika. Obiekt JSON, który przekazano za pośrednictwem parametru tej metody, reprezentuje identyfikator użytkownika (wskazujący użytkownika aktualnie korzystającego z aplikacji) oraz liczbę oczekiwanych czynności (w tym przypadku równą 20).

Po zwróceniu odpowiedzi dla tego żądania można przystąpić do analizy i dowolnego wykorzystywania poszczególnych czynności.

Generowanie powiadomień w celu zwiększania liczby użytkowników

Wiele kontenerów aplikacji społecznościowych udostępnia przepełnione, nieczytelne galerie, w których nowe aplikacje są spychane na najmniej atrakcyjne pozycje. W ten sposób twórcy kontenerów zapobiegają przesłanianiu podstawowych funkcji społecznościowych przez okna dodatkowych aplikacji. Taki kształt typowych galerii stanowi poważny problem dla programistów aplikacji — skoro budowane aplikacje trafiają do podzakładek lub innych trudno dostępnych miejsc w ramach kontenera, jak można dotrzeć do nowych użytkowników?

Jedną z najlepszych metod zachęcania użytkowników do instalowania aplikacji jest promocja produktu za pośrednictwem strumieni aktywności użytkowników. Strumień aktywności to jedna z niewielu dróg docierania do użytkowników (jeśli oczywiście sam kontener nie udostępnia atrakcyjnych miejsc dla okien aplikacji). Większość programistów, którzy zdecydowali się umieścić w strumieniu aktywności nowe, odpowiednio atrakcyjne i przyciągające uwagę powiadomienia, obserwowała nieporównanie większy wzrost zainteresowania swoimi produktami niż po umieszczeniu tych samych aplikacji w galerii.

Okazuje się, że specyfikacja OpenSocial definiuje prostą metodę języka JavaScript obsługującą umieszczanie nowych czynności w strumieniu aktywności. Za pomocą tej metody programista może promować swoją aplikację, kierując do użytkowników specjalnie przygotowane komunikaty.

Umieszczanie komunikatu w strumieniu aktywności użytkownika

W kontenerze zgodnym ze standardem OpenSocial 0.9 do umieszczania aktualizacji (powiadomień) w strumieniu aktywności użytkownika służy metoda `osapi.activities.create(...)`. Metoda umożliwia programiście łatwe wysyłanie komunikatów z aplikacji do strumienia aktywności lub dowolnego innego kanału przekazywania powiadomień, który jest obsługiwany przez dany kontener.

Metoda `osapi.activities.create(...)` otrzymuje na wejściu jeden parametr — obiekt JSON zawierający parametry żądania dotyczącego elementów strumienia aktywności (patrz tabela 6.1).

Za pomocą parametrów opisanych w tabeli 6.1 można zbudować blok języka JavaScript umieszczający nową aktualizację w strumieniu aktywności użytkownika:

Tabela 6.1. Parametry żądania elementu strumienia aktywności obsługiwane przez metodę osapi.activities.create

Parametr	Opis
`activity`	Obiekt `Activity` standardu OpenSocial definiujący treść wysyłanego powiadomienia.
`auth`	Obiekt `AuthToken` definiujący rodzaj autoryzacji (na przykład `HttpRequest.Authorization`).
`appId`	Łańcuch identyfikatora wskazujący aplikację, która wysłała dane powiadomienie (aktualizację). Kontener może użyć tego identyfikatora do automatycznego wyświetlenia w ramach powiadomienia szczegółowych informacji o aplikacji i linków prowadzących na jej stronę.
`groupId`	Identyfikator grupy, do której należy wysłać nowe powiadomienie (na przykład `@self`).
`userId`	Identyfikator użytkownika, do którego ma zostać przypisana tworzona aktualizacja (na przykład `@me`, `@viewer` lub `@owner`). Ten parametr może mieć postać łańcucha lub tablicy łańcuchów.

```
// umieszcza nowe powiadomienie w strumieniu aktywności bieżącego użytkownika
osapi.activities.create({
   userId: "@viewer",
   groupId: "@self",
   activity: {
      title: "Moja aplikacja robi mnóstwo przydatnych rzeczy",
      body: "<a href='http://www.mysite.com'>Kliknij tutaj</a>, aby uzyskać więcej informacji",
      url: "http://www.mysite.com/"
   }
}).execute();
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Aby wygenerować powiadomienie (aktualizację), w powyższym kodzie wywołano metodę `osapi.activities.create(...)` i przekazano na jej wejściu odpowiednio przygotowany obiekt JSON. W ramach tego obiektu należy wskazać strumień aktywności, do którego ma trafić nowa aktualizacja (w tym przypadku będzie to strumień bieżącego użytkownika aplikacji), grupę docelową (tutaj `self`) oraz obiekt `activity` reprezentujący treść powiadomienia. W powyższym kodzie obiekt `activity` zawiera tytuł, adres URL wskazywany przez link w tytule oraz opis (ciało) aktualizacji. Ciało aktualizacji może obejmować niewielki podzbiór znaczników języka HTML, w tym `<b>`, `<i>`, `<a>` oraz `<span>`. Wykonanie tego kodu spowoduje umieszczenie nowej aktualizacji w strumieniu aktywności wskazanego użytkownika.

Ustawianie priorytetu aktualizacji

Podczas umieszczania w strumieniu aktywności powiadomienia dla użytkownika należy zagwarantować możliwość wysyłania powiadomień aplikacji w imieniu samego użytkownika, nawet jeśli ten użytkownik wprost nie przekazał aplikacji odpowiednich uprawnień. Takie działanie jest możliwe dzięki priorytetom czynności.

Aby ustawić priorytet powiadomienia, należy ustawić opcjonalną flagę `priority`. To pole logiczne może zawierać albo wartość `0` (niski priorytet), albo wartość `1` (wysoki priorytet). Stosowana wartość powinna zależeć od tego, czy użytkownik będący adresatem powiadomienia nadał aplikacji odpowiednie uprawnienia. Innym ważnym czynnikiem jest implementacja samego kontenera. Jeśli programista zdefiniował wysoki priorytet (`1`) i jeśli użytkownik nie dał aplikacji uprawnień do umieszczania aktualizacji w jego imieniu, aplikacja podejmie próbę załadowania mechanizmu uwierzytelniania, aby zapytać użytkownika o zgodę na umieszczenie nowego

elementu w strumieniu aktywności. Jeśli programista ustawił niski priorytet (0) i jeśli użytkownik nie dał aplikacji niezbędnych uprawnień, aktualizacja zostanie zignorowana, a użytkownik nie zostanie zapytany o zgodę na umieszczenie nowej aktualizacji w strumieniu aktywności.

Ustawienie flagi `priority` wymaga umieszczenia odpowiedniej wartości w obiekcie JSON przekazywanym podczas tworzenia żądania:

```
// umieszcza nowe powiadomienie z wysokim priorytetem w strumieniu aktywności bieżącego użytkownika
osapi.activities.create({
   userId: "@viewer",
   activity: {
      title: "Więcej informacji można znaleźć na moim blogu",
      url: "http://www.nakedtechnologist.com/",
      priority: 1
   }
}).execute();
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

W powyższym przykładzie użytkownik, który nie nadał aplikacji uprawnień do umieszczania powiadomień w swoim strumieniu aktywności, jest proszony o zgodę na dodanie nowej aktualizacji. Pytanie o zgodę może być wyświetlone, jeśli na przykład użytkownik nie zezwolił danej aplikacji na dostęp do swoich danych społecznościowych lub jeśli przegląda tę aplikację w trybie podglądu.

Wzbogacanie aktualizacji o treści multimedialne

Dołączanie elementów multimedialnych do powiadomień umożliwia wzbogacenie interakcji z treścią umieszczaną w strumieniu aktywności i dużo skuteczniejsze przyciąganie uwagi użytkownika niż w przypadku standardowego tekstu i linków. Stosowanie elementów multimedialnych jest więc świetnym sposobem zwiększania liczby odbiorców, którzy zdecydują się na bliższe poznanie i zainstalowanie promowanej aplikacji.

Żądanie wysłania powiadomienia obejmuje opcjonalne pole `mediaItems`, w którym programista może umieszczać obrazy, pliki audio i zapisy wideo wzbogacające treść aktualizacji.

Do tworzenia elementów multimedialnych w ramach kontenera OpenSocial służy metoda `opensocial.newMediaItem(...)`, która otrzymuje na wejściu typ MIME definiujący rodzaj dodawanej treści oraz adres URL samej treści, na przykład obrazu:

```
// tworzy nowy element multimedialny dla obrazu
var imageUrl = "http://www.mysite.com/image.jpg";
var mediaImg = opensocial.newMediaItem("image/jpeg", imageUrl);
var mediaObj = [mediaImg];

// buduje listę parametrów dla nowego powiadomienia
var params = {};
params[opensocial.Activity.Field.TITLE] = "Wysłanie obrazu";
params[opensocial.Activity.Field.URL] = "http://www.myserver.com/index.php";
params[opensocial.Activity.Field.BODY] = "Test: <b>1, 2, 3</b>";
params[opensocial.Activity.Field.MEDIA_ITEMS] = mediaObj;
var activityObj = opensocial.newActivity(params);
```

```
// generuje żądanie utworzenia nowego powiadomienia
osapi.activities.create({
   userId: "@viewer",
   activity: activityObj
}).execute();
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Proces dołączania obiektu reprezentującego treść multimedialną w powyższym przykładzie składa się z trzech kroków. W pierwszym kroku należy utworzyć nowy obiekt elementu multimedialnego za pomocą metody `opensocial.newMediaItem(...)`, która tworzy odpowiednią strukturę. Pierwszy parametr tej metody, w tym przypadku `image/jpeg`, reprezentuje typ MIME, czyli rodzaj tworzonych danych. Drugi parametr zawiera łańcuch adresu URL wskazującego odpowiedni obraz. W powyższym przykładzie zastosowano typowe rozwiązanie dla dołączania treści multimedialnych do powiadomienia, czyli utworzono tablicę obejmującą poszczególne pola nowego obiektu.

W drugim kroku należy utworzyć obiekt `activity` obejmujący wszystkie dane składowe, w tym obiekt treści multimedialnych. Parametry składające się na ten obiekt można zdefiniować w formie obiektu JSON lub za pomocą metody `opensocial.newActivity(...)`, która generuje odpowiednią strukturę. W roli podstawowych danych tekstowych obiektu powiadomienia użyto tytułu, ciała i adresu URL. W powyższym kodzie dodano też element multimedialny za pośrednictwem pola `opensocial.Activity.Field.MEDIA_ITEMS`, któremu przypisano tablicę zawierającą obiekt multimedialny utworzony w poprzednim kroku. Prezentowany kod wywołuje następnie metodę `opensocial.newActivity(...)`, aby wygenerować strukturę nowego powiadomienia.

W ostatnim kroku należy przygotować żądanie utworzenia powiadomienia. Na wejściu metody `create` przekazano identyfikator użytkownika, do którego ma trafić powiadomienie (w tym przypadku będzie to bieżący użytkownik aplikacji), oraz utworzony wcześniej obiekt samego powiadomienia.

Wymienione kroki wystarczą do wysłania powiadomienia (aktualizacji) obejmującego obraz. Wstawienie zapisu wideo lub strumienia audio wymaga przeprowadzenia identycznej procedury — należy tylko pamiętać o ustawieniu właściwego adresu URL i typu MIME dla prezentowanych danych.

Pasywne i bezpośrednie publikowanie powiadomień aplikacji

Istnieją dwie główne kategorie metod promowania aplikacji społecznościowych: **bezpośrednie** i **pasywne publikowanie powiadomień**. Wybór właściwego rozwiązania zależy od okoliczności, w jakich aktualizacja będzie publikowana w imieniu użytkownika, oraz od tego, czy użytkownik ma świadomość generowania tej aktualizacji.

Istnieje kilka alternatywnych modeli publikowania powiadomień aplikacji. Jak już wspomniałem, wielu programistów promuje swoje aplikacje, umieszczając powiadomienia w strumieniach

aktywności użytkownika, tak aby odpowiednie komunikaty były prezentowane wszystkim znajomym tego użytkownika. Programiści stosujący tę metodę są przekonani, że więcej powiadomień oznacza większą widoczność oferty. Przyjmijmy, że użytkownik ma zainstalowanych co najmniej pięć aplikacji, a każda z nich wysyła maksymalną możliwą liczbę powiadomień, które w dodatku są rozsyłane do wszystkich znajomych tego użytkownika. Co będzie, jeśli kontener oferuje mechanizm ukrywania wszystkich tego rodzaju powiadomień, aby zapobiec zasypywaniu swojego strumienia aktywności natrętnymi ofertami? Okazuje się, że niemal wszystkie kontenery obsługujące aplikacje społecznościowe *umożliwiają* ukrywanie tego rodzaju powiadomień, zatem programiści muszą bardzo ostrożnie planować dobór wiadomości i częstotliwość ich wysyłania. W takim przypadku należy rozważyć wybór bezpośredniego lub pasywnego publikowania powiadomień. Obie opcje mają swoje zalety i wady, które zostaną szczegółowo omówione w poniższych punktach.

Bezpośrednie publikowanie powiadomień aplikacji

Bezpośrednie publikowanie powiadomień aplikacji polega na umieszczaniu w strumieniu aktywności komunikatów (aktualizacji) zależnie od czynności podejmowanych przez użytkownika, za jego wiedzą i zgodą. Użytkownik godzi się na ten model, akceptując odpowiednią opcję, korzystając z funkcji oferowanych przez tę aplikację lub wybierając jakąś formę nagrody lub wsparcia w ramach tej aplikacji.

Podstawowym argumentem na rzecz mechanizmu bezpośredniego publikowania powiadomień jest to, że użytkownik wie o działaniach podejmowanych przez aplikację w jego imieniu i — tym samym — mniejsze jest prawdopodobieństwo wyłączenia (ukrycia) powiadomień wysyłanych przez aplikację (w konfiguracji kontenera lub samej aplikacji) lub wręcz odinstalowania aplikacji z powodu utraty zaufania do jej działań. Utrzymanie relacji zaufania pomiędzy użytkownikiem a aplikacją jest bardzo ważne, jeśli wokół tej aplikacji ma być budowana szersza społeczność. Programista powinien robić wszystko, aby użytkownik był przekonany o możliwości korzystania z aplikacji bez obaw o złośliwe czy ryzykowne działania.

Największą wadą bezpośredniego publikowania powiadomień aplikacji jest ograniczona liczba generowanych aktualizacji. Jeśli aplikacja nie dość skutecznie zachęca użytkownika do działań, które powodują wysyłanie powiadomień, większość użytkowników nie będzie godziła się na umieszczanie nowych elementów w swoich strumieniach aktywności, zatem przekaz promujący aplikację nie będzie trafiał do ich znajomych. Użytkowników kontenera aplikacji społecznościowych bardzo szybko zniechęca nadmiar powiadomień wysyłanych do znajomych przez zainstalowaną aplikację. Ponieważ większość kontenerów udostępnia funkcje blokowania umieszczania takich powiadomień w strumieniu użytkownika, programista, który w ten sposób chce promować swoją aplikację, musi stale mieć na uwadze ryzyko przekroczenia akceptowanej liczby aktualizacji.

Aby ograniczyć to ryzyko, należy dobrze *zaplanować*, jakie zdarzenia w ramach aplikacji będą stanowiły dobrą okazję do jej promowania (zarówno z perspektywy użytkownika, którego strumień aktywności zostanie użyty, jak i z punktu widzenia znajomych tego użytkownika, którzy otrzymają odpowiednie powiadomienia). Umieszczanie w strumieniu aktywności użytkownika zbyt dużej liczby aktualizacji jest najkrótszą drogą do całkowitego zablokowania powiadomień generowanych przez tę aplikację. Powiadomienia należy umieszczać w strumieniu aktywności *z umiarem*.

Bezpośrednie publikowanie powiadomień aplikacji polega na prezentowaniu zachęt do działania, które przekonają użytkownika do opublikowania aktualizacji aplikacji w jego strumieniu aktywności:

```
<div id="msgNode"></div>
<div id="shareMsg">
   Przekaż znajomym, że zaktualizowałeś swój profil i zarobiłeś 5 dolarów w walucie aplikacji!<br />
   <button onclick="addActivity();">Publikuj</button>
</div>

<script type="text/javascript">
// generuje żądanie utworzenia nowego powiadomienia
function addActivity(){
   osapi.activities.create({
      userId: "@viewer",
      activity: {
         title: "Zaktualizowałem swój profil - kliknij, aby przeczytać o tych aktualizacjach",
         url: "http://www.container.com/myapp"
      }
   }).execute(function(){
      // powiadomienie zostało opublikowane — wyświetla się komunikat o pomyślnym zakończeniu operacji
      document.getElementById("msgNode").innerHTML = "Twoja wiadomość została opublikowana";
      document.getElementById("shareMsg").style.display = "none";
      // kod dodający 5 dolarów waluty aplikacji do profilu użytkownika
   });
}
</script>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Po rozłożeniu tego przykładu na elementy składowe łatwo zauważyć, że użytkownik otrzymał komunikat tekstowy i przycisk z zachętą do opublikowania informacji o aktualizacji swojego profilu. Kliknięcie przycisku *Publikuj* powoduje, że odpowiednie powiadomienie jest umieszczane w jego strumieniu aktywności, a sam użytkownik otrzymuje potwierdzenie tej aktualizacji.

Pasywne publikowanie powiadomień aplikacji

W przeciwieństwie do bezpośredniego publikowania powiadomień pasywne publikowanie aktualizacji polega na umieszczaniu komunikatów w strumieniu aktywności użytkownika (i w jego imieniu) bez każdorazowego informowania i pytania o zgodę tego użytkownika. Model pasywnego publikowania powiadomień można obserwować w takich aplikacjach jak FourSquare czy Gowalla, które rozsyłają informacje o miejscu pobytu użytkownika za każdym razem, gdy użytkownik loguje się w tych serwisach. Użytkownik co prawda ma świadomość, że aplikacja umieszcza w strumieniu aktywności odpowiednie powiadomienia w jego imieniu (wcześniej wyraził na to zgodę), ale sam w żaden sposób nie uczestniczy w procesie publikowania poszczególnych aktualizacji.

Ta metoda publikacji powiadomień ma swoje zalety i wady. Użytkownik formalnie zezwolił aplikacji na publikowanie powiadomień w swoim imieniu, jednak programista aplikacji ma wolną rękę w kwestii wykorzystania tej zgody. Największą zaletą tego modelu jest pewność, że określone czynności użytkownika spowodują opublikowanie właściwych powiadomień

(w przeciwieństwie do modelu bezpośredniej publikacji, gdzie jest wymagana każdorazowa zgoda użytkownika). Oznacza to, że aplikacja działająca według tego modelu może publikować znacznie więcej aktualizacji w nadziei na dotarcie do większej liczby odbiorców.

Największa zaleta tego modelu jest jednocześnie jego zasadniczą wadą — źródłem problemów jest liczba powiadomień wysyłanych w imieniu użytkownika *bez jego każdorazowej zgody i bez jego udziału w tym procesie*. Takie działanie ma kilka negatywnych aspektów:

- Użytkownik musi z wyprzedzeniem wyrazić swoje zaufanie do aplikacji, aby umożliwić jej dostęp do swojego profilu społecznościowego i podejmowanie działań w jego imieniu. Naruszenie tej relacji zaufania poprzez wysyłanie zbyt wielu aktualizacji może spowodować ukrycie wszystkich powiadomień, wycofanie zgody na publikację aktualizacji w imieniu użytkownika lub wręcz całkowite odinstalowanie aplikacji.
- Istnieje wiele aplikacji publikujących liczne aktualizacje w strumieniach aktywności swoich użytkowników. Wszystkie te aktualizacje są widoczne dla znajomych tych użytkowników. Jeśli aplikacja umieszcza w strumieniu aktywności użytkownika zbyt wiele aktualizacji, jego znajomi najprawdopodobniej ukryją powiadomienia generowane przez tę aplikację lub wręcz ją odinstalują. Oznacza to, że zbyt duża liczba aktualizacji wiąże się z ryzykiem utraty potencjalnej bazy użytkowników.

Warunkiem skutecznego publikowania powiadomień (niezależnie od wybranego modelu) jest umiar. Nie należy nadużywać zaufania użytkownika i wysyłać zbyt wielu aktualizacji — w przeciwnym razie użytkownik bezpowrotnie straci zaufanie do aplikacji.

Zdarzeniem wywołującym pasywną publikację powiadomienia może być prosta aktualizacja profilu użytkownika, która powoduje umieszczenie odpowiedniego komunikatu w strumieniu aktywności:

```
<!-- WSTAWIĆ: elementy formularza niezbędne do aktualizacji profilu -->
Zaktualizuj swój profil
<button onclick="updateProfile();">Aktualizuj profil</button>

<script type="text/javascript">
// funkcja aktualizująca profil użytkownika
function updateProfile (){
   // WSTAWIĆ: skrypty generujące żądanie aktualizacji profilu użytkownika
   // generuje żądanie publikacji powiadomienia o aktualizacji profilu
   osapi.activities.create({
      userId: "@viewer",
      activity: {
         title: "Zaktualizowałem swój profil - kliknij, aby przeczytać o tych aktualizacjach",
         url: "http://www.container.com/myapp"
      }
   }).execute();
</script>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Powyższy przykład pod wieloma względami przypomina kod pokazany przy okazji omawiania bezpośredniego publikowania powiadomień. Jedyna różnica polega na tym, że doda-

nie nowego powiadomienia jest teraz ściśle związane z aktualizacją profilu, a użytkownik nie jest informowany o umieszczeniu nowego powiadomienia w strumieniu aktywności (nie jest też pytany o zgodę).

Przykładowa aplikacja udostępnia wiele pól formularza, za pośrednictwem którego użytkownik może zaktualizować swój profil. Po wprowadzeniu zmian użytkownik może kliknąć przycisk zapisujący nowe ustawienia. Kliknięcie tego przycisku powoduje wywołanie funkcji, która wysyła na serwer żądanie aktualizacji profilu. Po wysłaniu tego żądania odpowiednie powiadomienie jest umieszczane w strumieniu aktywności użytkownika (w jego imieniu) bez żadnej informacji czy ostrzeżenia dla samego użytkownika.

Zrównoważone publikowanie powiadomień

Jednym ze sposobów wykorzystania zalet obu modeli (bezpośredniego i pasywnego publikowania powiadomień) i jednocześnie unikania ich wad jest próba zintegrowania obu rozwiązań w ramach mechanizmu **zrównoważonego publikowania powiadomień**. Proponowana technika, jeśli jest prawidłowo stosowana, pozwala zagwarantować publikację określonej liczby powiadomień w związku z działaniami użytkownika, przy jednoczesnym zachowaniu relacji zaufania łączącej aplikację i tego użytkownika.

Technika zrównoważonego publikowania powiadomień bazuje na kilku podstawowych założeniach. Programista powinien zacząć od stosowania modelu pasywnego publikowania powiadomień i na podstawie zebranych doświadczeń określić, które czynności użytkownika powinny powodować publikowanie aktualizacji. Jeśli powiadomienia publikowane w tym trybie są skojarzone z częstymi czynnościami, aplikacja zasypie strumień aktywności użytkownika (bez jego wiedzy) mnóstwem komunikatów. Pasywne publikowanie powiadomień należy stosować tylko dla najważniejszych działań, jak wykonanie przez użytkownika czasochłonnego zadania, uzyskanie rzadkiej odznaki czy wprowadzenie istotnych zmian w jego profilu lub treści aplikacji. Takie rozwiązanie pozwala jednocześnie zagwarantować promocję pewnej liczby zdarzeń i wyeliminować ryzyko zmonopolizowania strumienia aktywności użytkownika.

Dla wszystkich pozostałych zdarzeń, które powinny powodować umieszczanie komunikatów w strumieniu aktywności, należy stosować technikę bezpośredniej publikacji powiadomień. Powiadomienia publikowane w tym trybie mogą dotyczyć zaproszeń, próśb o pomoc lub określoną treść kierowanych do znajomych, udostępniania znajomym treści aplikacji itp. Zachęty do działania mogą być bardziej kuszące, jeśli zawierają informacje o korzyściach wynikających ze zgody na publikację aktualizacji — może to być zastrzyk wirtualnej gotówki lub dostęp do rozmaitych ulepszeń.

Rozsądne gospodarowanie aktualizacjami i utrzymanie relacji zaufania łączącej aplikację z użytkownikiem może bardzo ułatwić budowanie bogatej bazy promocji aplikacji przy użyciu strumienia aktywności. W idealnych warunkach to użytkownicy będą promowali aplikację wśród swoich znajomych, publikując powiadomienia dotyczące wykonywanych przez siebie czynności.

Generowanie żądań AJAX i żądań dostępu do danych zewnętrznych

W trakcie normalnego funkcjonowania programu programista często staje przed koniecznością modyfikacji kodu aplikacji lub rozszerzenia źródeł danych serwera (na przykład bazy danych) o nową treść. Aby uprościć to zadanie, specyfikacja OpenSocial definiuje metody obiektu http dostępne za pośrednictwem standardowej biblioteki JavaScriptu.

Programiści mogą używać tej metody do generowania żądań REST (`GET`, `PUT`, `POST` i `DELETE`) pomiędzy aplikacją a serwerem w celu zmodyfikowania stanu działającego systemu bez wpływu na doznania użytkowników.

Do generowania tych żądań służą następujące metody obiektu `osapi.http`:

- `osapi.http.get(`*`url`*`, `*`parametry`*`)`
- `osapi.http.put(`*`url`*`, `*`parametry`*`)`
- `osapi.http.post(`*`url`*`, `*`parametry`*`)`
- `osapi.http.delete(`*`url`*`, `*`parametry`*`)`

Oprócz adresu URL, na który ma zostać wysłane żądanie http, na wejściu wymienionych metod można dodatkowo przekazywać wiele różnych parametrów. Dostępne parametry opisano w tabeli 6.2.

Tabela 6.2. Parametry żądań obiektu http

Parametr	Opis
`authz` *(łańcuch)*	Metoda autoryzacji używana podczas wysyłania danych na serwer. Parametr może mieć wartość `none` (domyślnie), `signed` lub `oauth`.
`body` *(łańcuch)*	Stosowane tylko dla żądań `PUT` i `POST`. Dane wysyłane na serwer w ramach żądania.
`format` *(łańcuch)*	Format zwracanych danych. Parametr może mieć wartość `json` (domyślnie) lub `text`.
`headers` *(łańcuch lub tablica łańcuchów)*	Opcjonalne nagłówki wysyłane wraz z żądaniem danych.
`oauth_service_name` *(łańcuch)*	Element `service` w specyfikacji gadżetu, który ma zostać użyty w tym żądaniu. Domyślną wartością tego parametru jest łańcuch pusty ("").
`oauth_token_name` *(łańcuch)*	Token standardu OAuth stosowany w tym żądaniu. Domyślną wartością tego parametru jest łańcuch pusty ("").
`oauth_request_token` *(łańcuch)*	Token wstępnie zaakceptowany przez dostawcę (dla treści będącej przedmiotem żądania).
`oauth_request_token_secret` *(łańcuch)*	Tajny klucz skojarzony z tokenem `request_token`.
`oauth_use_token` *(łańcuch)*	Określa, czy w żądaniu należy użyć tokenu standardu OAuth. Parametr może mieć wartość `always`, `if_available` lub `never`.
`refreshInterval` *(liczba całkowita)*	Okres, w którym kontener może przechowywać zwrócone dane w swojej pamięci podręcznej.
`sign_owner` *(wartość logiczna)*	Określa, czy żądanie powinno być autoryzowane (podpisane) i czy ma obejmować identyfikator właściciela. Parametr domyślnie ma wartość `true`.
`sign_viewer` *(wartość logiczna)*	Określa, czy żądanie powinno być autoryzowane (podpisane) i czy ma obejmować identyfikator bieżącego użytkownika. Parametr domyślnie ma wartość `true`.

Znaczna część żądań dotyczących danych, które nie wymagają zabezpieczeń, obejmuje zaledwie kilka z opisanych parametrów, w tym `format`, `body` (w przypadku żądań `POST` i `PUT`) oraz `refreshInterval` (w celu poprawienia wydajności).

Parametry `authz`, `sign_*` oraz `oauth_*` stosuje się w sytuacji, gdy żądanie danych wymaga zabezpieczenia i gdy odbiorca musi potwierdzić tożsamość nadawcy tego żądania.

Generowanie standardowych żądań dostępu do danych

Jeśli budowany serwis wymaga bezpiecznego przesyłania danych, dostęp do zdecydowanej większości tych danych będzie się odbywał za pośrednictwem standardowych żądań serwera RESTful. Do generowania tych standardowych żądań służy wywołanie `osapi.http.`*`metoda`*, gdzie słowo *`metoda`* należy zastąpić typem żądania, czyli `get`, `put`, post lub `delete`.

Ponieważ omawiane rozwiązania wymagają stosowania funkcji biblioteki `osapi` języka JavaScript, należy dodać wyrażenie `Require`, aby było możliwe korzystanie z metod generowania żądań. Po udostępnieniu wspomnianych metod można przystąpić do utworzenia praktycznego przykładu, który będzie pobierał dane z zewnętrznego źródła i wyświetlał uzyskaną w ten sposób treść w oknie aplikacji.

Poniższy przykład kodu generuje żądanie `GET` wysyłane do serwisu Flickr przy użyciu języka YQL (od ang. *Yahoo! Query Language*). Żądanie zwraca fotografie pasujące do wyszukiwanego słowa Montreal. Otrzymane wyniki są następnie przetwarzane — każde zdjęcie jest otaczane znacznikami `<img>` i wyświetlane w ramach okna aplikacji:

```
<?xml version="1.0" encoding="utf-8"?>
<Module>
   <ModulePrefs title="Żądanie GET do serwisu Flickr za pomocą języka YQL">
      <Require feature="osapi"/>
   </ModulePrefs>
   <Content type="html" view="canvas">
      <![CDATA[
      <div id="imgContainer"></div>

      <script type="text/javascript">
      // Wywołanie zwrotne dla żądania GET
      function requestCallback(response){
         var photolist = response.content.query.results.photo, html = "";

         // przeszukuje w pętli kolejne obrazy i tworzy znaczniki <img>
         for (var i in photolist){
            if (photolist.hasOwnProperty(i)){
               html += "<img src='http://farm" + photolist[i].farm +
                       ".static.flickr.com/" + photolist[i].server +
                       "/" + photolist[i].id +
                       "_" + photolist[i].secret +
                       ".jpg' alt='" + photolist[i].title +"' /><br />";
               document.getElementById('imgContainer').innerHTML = html;
            }
         }
      }

      // generuje żądanie GET
      var url = "http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20flickr.photos.
      ➥search%20where%20text%3D%22Montreal%22&format=json";
      osapi.http.get({
         "href": url,
```

```
            "format": "json"
        }).execute(requestCallback);
        </script>
        ]]>
    </Content>
</Module>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

W pierwszym kroku zdefiniowano specyfikację gadżetu niezbędną do uruchomienia tego przykładu, w szczególności element `Require` dla biblioteki `osapi` (w ramach elementu `ModulePrefs`). W kolejnym kroku utworzono sekcję `Content` wyświetlającą widok kanwy aplikacji.

Właściwe wykonywanie programu rozpoczyna się w dolnej części sekcji `Content`. Zdefiniowano tam adres URL, na który zostanie wysłane żądanie (w tym przypadku użyto adresu usługi YQL z łańcuchem wyszukiwania dla serwisu Flickr). Bezpośrednio potem wywołano metodę `our osapi.http.get(...)`, aby zainicjować to żądanie. Na wejściu tej metody przekazano wspomniany adres URL (za pośrednictwem parametru `href`) i określono, że oczekiwanym wynikiem tego żądania jest obiekt JSON. Ostatnim elementem tej części kodu jest wysłanie żądania za pomocą metody `execute(...)`. Na wejściu tej metody przekazano referencję do funkcji wywołania zwrotnego, która ma zostać wykonana po zwróceniu odpowiedzi dla żądania.

Po zakończeniu przetwarzania żądania następuje wywołanie funkcji `requestCallback`, która otrzymuje na wejściu (za pośrednictwem parametru) obiekt odpowiedzi. Dalsza część kodu odpowiada za przetworzenie otrzymanych wartości — użyta pętla for przeszukuje poszczególne obiekty na liście.

Dla każdej fotografii znalezionej na liście jest generowany fragment kodu języka HTML ze znacznikami `<img>` i odwołaniem do odpowiedniego adresu URL w ramach serwisu Flickr. Po wygenerowaniu treści w formacie HTML dla wszystkich obrazów gotowy łańcuch jest umieszczany w węźle `div` skonfigurowanym na początku sekcji `Content`.

Umieszczanie treści w żądaniach danych

W niektórych przypadkach wraz z żądaniem należy wysłać na serwer określoną treść. Żądanie może dotyczyć na przykład umieszczenia na serwerze nowych ustawień konfiguracyjnych użytkownika, czyli aktualizacji rekordu tego użytkownika w bazie danych:

```
<label for="user"></label>
<input type="text" name="user" id="user" /><br />
<label for="pass"></label>
<input type="hidden" name="pass" id="pass" />
<button onclick="updateRecord();">Aktualizuj dane użytkownika</button>
<div id="response"></div>

<script type="text/javascript">
function updateRecord(){
   // ustawia adres URL i wysyłane dane
   var url = "http://www.mysite.com/updateUser.php";
   var postData = "user=" +
      encodeURIComponent(document.getElementById("user").value) + "&pass=" +
      encodeURIComponent(document.getElementById("pass").value);
```

```
    // wysyła obiekt danych pod wskazany adres URL (żądanie POST)
    osapi.http.post({
       "href": url,
       "body": postData,
       "format": "text"
    }).execute(function(response){
       document.getElementById("response").innerHTML = "Wysłano dane";
    });
}
</script>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

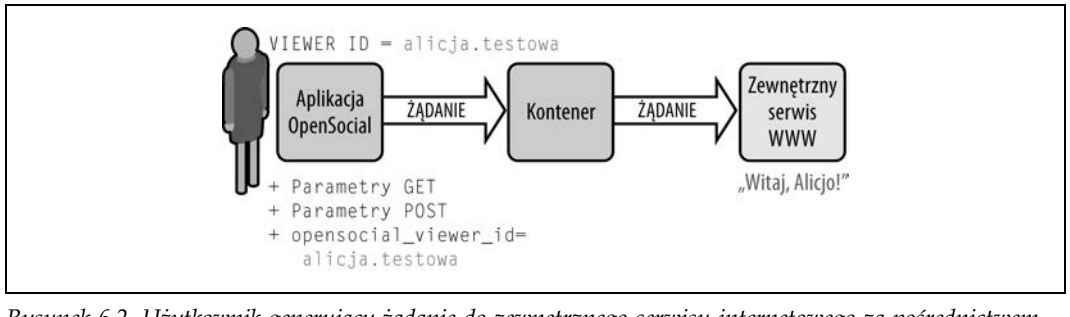
W ramach żądania POST umieszczono kod języka znaczników, aby umożliwić użytkownikowi podanie nazwy i hasła. Obie wartości reprezentują informacje, które zostaną przekazane na serwer w celu zaktualizowania rekordu użytkownika. W powyższym kodzie zdefiniowano też węzeł `div`, w którym będzie wyświetlane potwierdzenie wysłania żądania. Kliknięcie przycisku przez użytkownika spowoduje wywołanie funkcji `updateRecord()`.

W ciele funkcji `updateRecord()` ustawiono adres URL, na który zostanie wysłane żądanie, oraz wygenerowano pary klucz-wartość żądania `POST` na podstawie pól z danymi wejściowymi. Kolejne pary oddzielono znakiem `&`.

I wreszcie przedstawiony kod generuje żądanie `POST` protokołu HTTP za pomocą metody `osapi.http.post(...)` otrzymującej na wejściu adres URL, format i dane, które mają zostać wysłane w ramach tego żądania. Metoda `execute()` generuje żądanie i wywołuje funkcję zwrotną po jego przetworzeniu. Funkcja zwrotna umieszcza w węźle `div` prosty komunikat o pomyślnym przebiegu operacji.

Używanie autoryzowanych żądań do zabezpieczania połączeń

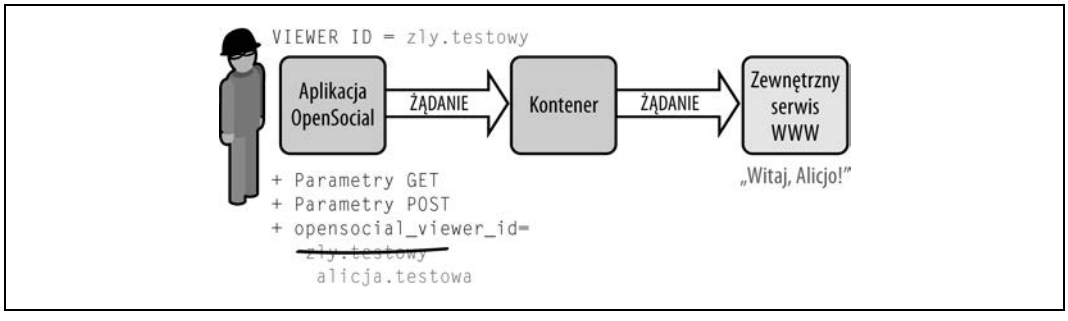
W przypadku standardowych żądań protokołu HTTP wysyłanych za pośrednictwem aplikacji korzystającej z metod obiektu `osapi.http` to kontener pełni funkcję pośrednika i przekazuje bezpośrednio na serwer wszystkie parametry dołączone do żądania. W tym modelu kontener w żaden sposób nie modyfikuje przekazywanych parametrów (patrz rysunek 6.2).



Rysunek 6.2. Użytkownik generujący żądanie do zewnętrznego serwisu internetowego za pośrednictwem kontenera bez autoryzacji OAuth

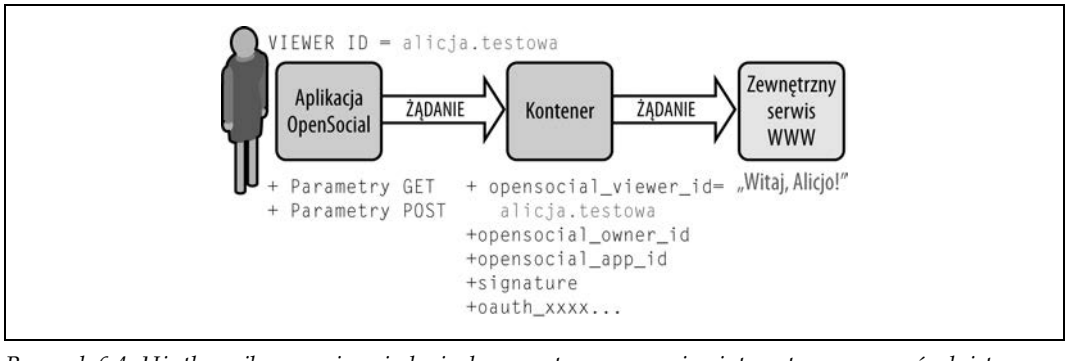
W przypadku pobierania danych z zewnętrznego źródła, które nie wymaga stosowania zabezpieczeń ani weryfikacji tożsamości użytkownika, opisany powyżej model (z niezabezpieczonym transferem danych) w zupełności wystarczy.

Warto jednak przeanalizować nieco inny scenariusz generowania żądania. Przypuśćmy, że zamiast pobierać przypadkowe, niewrażliwe dane, generujemy żądanie `POST`, aby zaktualizować na serwerze dane użytkownika. Wszystkie parametry przekazywane na serwer, w tym identyfikator użytkownika, są ustawiane w kodzie samej aplikacji. Ponieważ żądanie nie jest w żaden sposób zabezpieczone, każdy użytkownik może odczytać te dane za pomocą Firebuga lub tak zmodyfikować żądanie, aby zmienić lub uzyskać informacje o innym użytkowniku. Na rysunku 6.3 pokazano żądanie wysłane na serwer, które sprawia wrażenie w pełni poprawnego, mimo że w rzeczywistości przekazuje kontrolę nad danymi użytkownika innemu, nieuprawnionemu użytkownikowi.



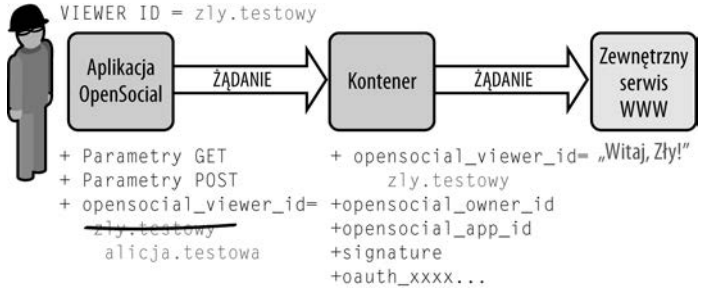
Rysunek 6.3. Złośliwy użytkownik generuje żądanie do zewnętrznego serwisu, skutecznie podszywając się pod innego użytkownika (wskutek braku autoryzacji OAuth)

Właśnie takim przypadkom można zapobiec, stosując autoryzowane żądania, które można generować za pomocą metod obiektu `osapi.http` standardu OpenSocial. Autoryzowane (podpisane) żądania umożliwiają programistom przekazywanie parametrów pomiędzy aplikacją a serwerem, jednak w tym przypadku kontener aplikacji jest nieporównanie bardziej aktywny. Kontener, do którego trafia autoryzowane żądanie (patrz rysunek 6.4), sprawdza tożsamość użytkownika występującego w roli nadawcy i dołącza do żądania odpowiedni identyfikator. Oprócz tego identyfikatora kontener dodaje do żądania zaszyfrowany skrót, który umożliwia weryfikację poprawności identyfikatora przez niezależną, zewnętrzną usługę.



Rysunek 6.4. Użytkownik generujący żądanie do zewnętrznego serwisu internetowego za pośrednictwem kontenera z wykorzystaniem autoryzacji OAuth

Dodatkowa weryfikacja oznacza, że atak polegający na próbie sfałszowania tożsamości użytkownika i wysłania na serwer danych uwierzytelniających innego użytkownika zakończy się niepowodzeniem, ponieważ sfałszowane dane i tak zostaną nadpisane przez dane uwierzytelniające właściwego użytkownika. Jeśli dane użytkownika zostaną potwierdzone, w ramach żądania można dodatkowo przekazać takie parametry jak identyfikator bieżącego użytkownika i właściciela czy identyfikator aplikacji (patrz rysunek 6.5).



Rysunek 6.5. Złośliwy użytkownik generuje żądanie do zewnętrznego serwisu, ale nie może skutecznie podszywać się pod innego użytkownika z powodu zastosowania mechanizmu autoryzacji OAuth

W przypadku wygenerowania autoryzowanego (podpisanego) żądania serwer docelowy zawsze otrzymuje następujące parametry:

`opensocial_owner_id`
: Unikatowy identyfikator właściciela aplikacji.

`opensocial_app_url`
: Pełny adres URL aplikacji, która wygenerowała dane żądanie.

Poza wymienionymi powyżej wymaganymi parametrami kontenery mogą też wysyłać dodatkowe informacje ułatwiające weryfikację, w tym:

`opensocial_viewer_id`
: Unikatowy identyfikator bieżącego użytkownika aplikacji.

`opensocial_instance_id`
: Określa, czy kontener powinien obsługiwać wiele instancji tej samej aplikacji. Za pośrednictwem tego parametru należy przekazać identyfikator instancji aplikacji wysyłającej dane żądanie. Pomiędzy parametrami `opensocial_instance_id` i `opensocial_app_url` można wskazać instancję konkretnej aplikacji działającej w kontenerze.

`opensocial_app_id`
: Unikatowy identyfikator aplikacji. Ten parametr służy przede wszystkim zachowaniu zgodności wstecz z wersją 0.7 specyfikacji OpenSocial.

`xoauth_public_key`
: Klucz publiczny użyty do podpisania danego żądania. Jeśli kontener nie stosuje kluczy publicznych do podpisywania żądań lub jeżeli wykorzystuje alternatywne metody przekazywania kluczy w ramach żądań, ten parametr można pominąć.

Oprócz wymienionych powyżej parametrów na serwer można dodatkowo wysyłać dane uwierzytelniające standardu OAuth umożliwiające weryfikację autoryzowanych żądań. Do tej grupy parametrów należą:

- `oauth_consumer_key`
- `oauth_nonce`
- `oauth_signature`
- `oauth_signature_method`
- `oauth_timestamp`
- `oauth_token`

Samo wygenerowanie podpisanego (autoryzowanego) żądania nie gwarantuje pełnej ochrony przed manipulacjami. Serwer docelowy żądania musi jeszcze wykonać dodatkowe kroki związane z weryfikacją autoryzowanego żądania, aby potwierdzić poprawność nadesłanych danych.

Generowanie autoryzowanego żądania

Utworzenie podpisanego (autoryzowanego) żądania wymaga połączenia standardowej składni żądań, którą stosowano już we wcześniejszych przykładach, z parametrem `authz`:

```
// generuje podpisane żądanie GET protokołu HTTP
osapi.http.get({
   'href' : 'http://www.mysite.com/editUser.php',
   'format' : 'json',
   'authz' : 'signed'
}).execute(callback);
```

W ramach tego żądania `GET` zdefiniowano adres URL, na który ma zostać wysłane podpisane żądanie, wskazano format danych i — co najważniejsze — określono, że żądanie ma być podpisane (parametr `authz`). Powyższy kod generuje więc podpisane żądanie `GET` protokołu HTTP.

Weryfikacja podpisanego żądania po stronie serwera

Jak już wspomniałem, samo wygenerowanie i wysłanie podpisanego żądania na serwer nie wystarczy do zagwarantowania poprawności tego żądania. Złośliwy użytkownik może podjąć próbę fałszowania także autoryzowanych żądań. Do weryfikacji podpisanych żądań pod kątem pochodzenia z właściwego źródła programista może wykorzystać podpis standardu OAuth.

Weryfikacja żądań przy użyciu tego mechanizmu będzie wymagała kilku dodatkowych elementów:

- Biblioteki OAuth (dostępnej na stronie *http://code.google.com/p/oauth/*) niezbędnej do weryfikacji żądań po stronie serwera. W tym przykładzie będzie stosowana biblioteka PHP OAuth 1.0 Rev A (dostępna pod adresem *http://oauth.googlecode.com/svn/code/php/*).
- Jeśli kontener stosuje metodę weryfikacji na bazie certyfikatu klucza publicznego, będzie potrzebny odpowiedni certyfikat. Listę certyfikatów z kluczami publicznymi dla wielu różnych kontenerów oraz adresy usług sprawdzania tych certyfikatów można znaleźć na stronie *https://opensocialresources.appspot.com/certificates/*. Wspomniany serwis należy jednak traktować wyłącznie jako pomocnicze źródło, ponieważ jego treść nie jest aktualizowana ani aprobowana przez twórców kontenerów. Aby zintegrować najlepsze mechanizmy zabezpieczeń, należy zapoznać się z dokumentacją kontenera i znaleźć najbardziej aktualny certyfikat klucza publicznego.

Weryfikacja podpisanego żądania składa się z dwóch kroków. Po pierwsze, w kodzie po stronie klienta należy odpowiednio przygotować podpisane żądanie na potrzeby skryptu po stronie serwera. Następnie trzeba przetworzyć parametry przesłane do skryptu po stronie serwera i sprawdzić ich poprawność za pomocą elementów biblioteki OAuth. Cały ten dwuetapowy proces został omówiony poniżej.

Generowanie podpisanego żądania w kodzie JavaScriptu

Wygenerowanie podpisanego żądania w warstwie kodu języka JavaScript (w ramach sekcji `Content` specyfikacji gadżetu) spowoduje przekazanie danych uwierzytelniających biblioteki OAuth, kontenera i użytkownika w celu sprawdzenia poprawności żądania po stronie serwera.

Wygenerowanie takiego żądania sprowadza się do utworzenia podpisanego żądania `osapi.`
`↪http.get` adresowanego do skryptu po stronie serwera (podobnie jak w podpunkcie „Generowanie autoryzowanego żądania" we wcześniejszej części tego podrozdziału):

```
<?xml version="1.0" encoding="utf-8"?>
<Module>
   <ModulePrefs title="Weryfikacja podpisanego żądania AJAX">
      <Require feature="opensocial-0.9"/>
      <Require feature="osapi"/>
   </ModulePrefs>
   <Content type="html" view="canvas">
      <![CDATA[
      <div id="validationResponse"></div>

      <script type="text/javascript">
      function dataCallback(response){
         document.getElementById("validationResponse").innerHTML =
            "Żądanie zweryfikowane jako: " + response.data.validation;
      }

      osapi.http.get({
         "href" : "http://www.mysite.com/validate.php",
         "format" : "text",
         "authz" : "signed"
      }).execute(dataCallback);
      </script>
      ]]>
   </Content>
</Module>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Sam gadżet zawiera wyrażenia `Require` dla żądania `osapi.http.get(...)` i sekcję `Content` niezbędną do wykonywania tych skryptów. We wspomnianej sekcji `Content` umieszczono węzeł `div`, w którym będzie wyświetlany komunikat o wyniku weryfikacji (komunikat będzie generowany przez funkcję wywołania zwrotnego oraz przez wywołanie zwrotne samego żądania protokołu HTTP). W aplikacji produkcyjnej należałoby sprawdzać komunikat i odpowiednio obsługiwać komunikat odesłany przez serwer w kodzie skryptu. Jeśli odpowiedź serwera jest prawidłowa i obejmuje nowy kod języka znaczników, można wstawić ten kod na stronie. Jeżeli jednak odpowiedź jest nieprawidłowa, należy wyświetlić stosowny komunikat o braku możliwości realizacji żądania (przynajmniej w danej chwili).

Wywołanie metody `osapi.http.get(...)`, które ma miejsce po wspomnianym wywołaniu zwrotnym, generuje podpisane żądanie `GET` do skryptu po stronie serwera.

Weryfikacja podpisanego żądania po stronie serwera (algorytm RSA-SHA1 z certyfikatem klucza publicznego)

Żądanie przekazywane (za pośrednictwem odpowiedniej warstwy kontenera) do skryptu po stronie serwera — w tym przypadku funkcję skryptu pełni plik *http://www.mysite.com/validate.php* — powinno obejmować wszystkie parametry niezbędne do weryfikacji, czyli dane uwierzytelniające kontenera, użytkownika i biblioteki OAuth.

Wiele popularnych kontenerów aplikacji społecznościowych stosuje certyfikaty z kluczami publicznymi do weryfikacji żądań przy użyciu algorytmu RSA-SHA1. W poniższym przykładzie żądanie zostanie sprawdzone z wykorzystaniem jednego z takich certyfikatów.

Jeśli kontener nie stosuje certyfikatów z kluczami publicznymi do weryfikacji żądań lub jeżeli programista woli użyć algorytmu HMAC-SHA1 zamiast RSA-SHA1, należy wygenerować klucz tajny na poziomie kontenera i wykorzystać ten klucz w miejsce certyfikatu klucza publicznego.

```
<?php
require_once("OAuth.php");

class buildSignatureMethod extends OAuthSignatureMethod_RSA_SHA1 {
   public function fetch_public_cert(&$request) {
      return file_get_contents("http://www.fmodules.com/public080813.crt");
   }
}

// konstruuje żądanie na podstawie parametrów POST i GET
$request = OAuthRequest::from_request(null, null, array_merge($_GET, $_POST));

// tworzy nową metodę podpisu na podstawie utworzonej klasy i certyfikatu klucza publicznego
$signature_method = new buildSignatureMethod();

// sprawdza podpis
@$signature_valid = $signature_method->check_signature($request, null, null, $_GET
["oauth_signature"]);

$response = array();
if ($signature_valid) {
   // sprawdza podpisane żądanie i wysyła komunikat o pomyślnej weryfikacji
   $response['validation'] = "valid";
} else {
   // sprawdza podpisane żądanie i wysyła komunikat o niepowodzeniu weryfikacji
   $response['validation'] = "invalid";
}

// wyświetla obiekt odpowiedzi
print(json_encode($response));
?>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Wyrażenie `require_once(...)` na początku tego przykładu wskazuje plik biblioteki OAuth, który pobrano wcześniej ze strony *http://oauth.googlecode.com/svn/code/php/*.

Warto na początku zwrócić uwagę na skrypt weryfikacji w postaci klasy `buildSignatureMethod`. Wspomniana klasa rozszerza klasę `OAuthSignatureMethod_RSA_SHA1` zdefiniowaną w pliku *OAuth.php* i zawiera zaledwie jedną funkcję odpowiedzialną za pobranie i zwrócenie zawartości pliku certyfikatu klucza publicznego. Danych zawartych w tym pliku nie należy pobierać za każdym razem, gdy jakieś podpisane żądanie wymaga weryfikacji. Należy raczej dodać te dane do pamięci podręcznej kluczy indeksowanej według wielu parametrów przekazywanych do skryptu po stronie serwera. Zawartość tej pamięci powinna być aktualizowana tylko w przypadku zmiany wspomnianych wartości. Przekazywane parametry obejmują:

- `xoauth_signature_publickey`
- `oauth_consumer_key`
- `oauth_signature_method`

W kolejnym kroku skonstruowano nowy obiekt żądania OAuth na podstawie parametrów `GET` i `POST` przesłanych w ramach żądania protokołu HTTP. Przekazane wartości obejmują parametry standardu OAuth i kontenera dołączone do żądania przez skrypt pośredniczący, który wygenerował to żądanie na poziomie kontenera. Obiekt żądania OAuth zostanie użyty do weryfikacji przekazanego podpisu. Za pomocą klasy uzyskującej certyfikat klucza publicznego można zbudować nowy podpis (na podstawie tego certyfikatu).

Metoda `check_signature(...)` jest wywoływana w celu sprawdzenia przekazanego podpisu i zapisania wyniku weryfikacji. Zależnie od efektu weryfikacji skrypt zapisuje komunikat o sukcesie bądź niepowodzeniu i odsyła do skryptu po stronie klienta obiekt odpowiedzi w formacie JSON.

Weryfikacja podpisanego żądania po stronie serwera (algorytm HMAC-SHA1)

Jeśli certyfikat klucza publicznego jest niedostępny dla kontenera, w którym działa aplikacja, można zastosować alternatywny model weryfikacji podpisanych żądań po stronie serwera. Zamiast algorytmu RSA-SHA1 należy wówczas użyć algorytmu HMAC-SHA1.

Zamiast posługiwać się certyfikatem klucza publicznego w roli metody weryfikacji żądania, programista może wygenerować nowy obiekt żądania OAuth na podstawie danych przesłanych w ramach podpisanego żądania:

```
<?php
require_once("OAuth.php");

$key = "TUTAJ NALEŻY ZDEFINIOWAĆ KLUCZ";
$secret = "TUTAJ NALEŻY ZDEFINIOWAĆ KLUCZ";

// Konstruuje nowy obiekt żądania na podstawie bieżącego żądania
$request = OAuthRequest::from_request(null, null, $_REQUEST);
$consumer = new OAuthConsumer($key, $secret, null);

// inicjalizuje metodę podpisu
$sig_method = new OAuthSignatureMethod_HMAC_SHA1();

// sprawdza przekazany podpis OAuth
$signature = $_GET['oauth_signature'];
$valid_sig = $sig_method->check_signature(
   $request,
   $consumer,
```

```
    null,
    $signature
);

// sprawdza, czy podpis został prawidłowo zweryfikowany
if (!$valid_sig) {
    // NIEPRAWIDŁOWY PODPIS — należy wygenerować odpowiedni komunikat o błędzie
} else{
    // PRAWIDŁOWY PODPIS — program może kontynuować działanie
}
?>
```

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Podobnie jak w poprzednim przykładzie, powyższy kod zaczyna się od dołączenia biblioteki OAuth języka PHP (dostępnej na stronie *http://oauth.googlecode.com/svn/code/php/*). Dzięki temu można utworzyć własny obiekt żądania OAuth i obiekt odbiorcy (tzw. konsumenta), po czym sprawdzić przekazany podpis. Oprócz wspomnianej biblioteki w kodzie dołączono odbiorcę żądania OAuth (obiekt odbiorcy żądania OAuth zostanie skonstruowany na podstawie zmiennych reprezentujących klucze).

W kolejnym kroku należy skonstruować obiekty żądania OAuth i jego odbiorcy. W tym celu wywołujemy najpierw metodę `OAuthRequest::from_request(...)`, aby zbudować obiekt żądania. Za pośrednictwem pierwszych dwóch parametrów przekazano wartość `null`, ponieważ w tym przypadku wspomniane parametry nie są wymagane. Parametry te reprezentują odpowiednio metodę protokołu HTTP i adres URL. Za pośrednictwem trzeciego parametru przekazano obiekt `$_REQUEST` zawierający wszystkie informacje na temat żądania OAuth, które są niezbędne do skonstruowania nowego obiektu. W kolejnym kroku utworzono nowy obiekt `OAuthConsumer`, przekazując na wejściu konstruktora klucz OAuth i klucz tajny. Trzecim parametrem tej metody jest adres URL wywołania zwrotnego dla procesu OAuth (w tym przypadku przekazano wartość `null`).

Kolejnym niezbędnym krokiem jest utworzenie nowego obiektu podpisu za pomocą wywołania konstruktora `OAuthSignatureMethod_HMAC_SHA1()`. Dzięki temu będzie możliwe porównywanie podpisu reprezentowanego przez ten obiekt z podpisem przekazanym w ramach autoryzowanego zdarzenia — na tej podstawie można stwierdzić, czy drugi podpis jest prawidłowy.

Kolejny blok kodu rozpoczyna się od odczytania podpisu OAuth przekazanego w ramach autoryzowanego żądania. Po odczytaniu tego podpisu nowy obiekt jest używany do wywołania metody `check_signature(...)`. Na wejściu tej metody (w roli listy parametrów) są przekazywane: obiekt żądania OAuth i obiekt odbiorcy OAuth, wartość tokenu (w tym przypadku jest niepotrzebna, stąd użyta wartość `null`) i wreszcie podpis przekazany w ramach żądania i będący przedmiotem porównania.

Po wykonaniu wszystkich tych zadań można wreszcie użyć zwróconej wartości do sprawdzenia, czy weryfikacja podpisu przebiegła pomyślnie (tj. czy podpis przekazany w ramach żądania był poprawny). Jeśli tak, program może kontynuować przetwarzanie podpisanego żądania. Jeśli nie, kod powinien wyświetlić komunikat o błędzie.

Budowanie kompletnego gadżetu

Kompletny kod źródłowy tego przykładu jest dostępny pod adresem *ftp://ftp.helion.pl/przyklady/prapse.zip*.

Skoro dysponujemy już wiedzą niezbędną do praktycznego stosowania funkcji społecznościowych gadżetu, warto wykorzystać zdobyte umiejętności i opracować prosty gadżet. Gadżet opracowany w tym podrozdziale będzie wyświetlał strumień aktywności znajomych bieżącego użytkownika i zdjęcia przypisane do ich profili oraz udostępniał użytkownikowi metodę dodającą nowe powiadomienie do jego strumienia.

Należy najpierw przygotować kod znaczników nowego gadżetu. W tym przykładzie będzie potrzebna tylko lekka biblioteka `osapi` języka JavaScript, zatem należy ją dodać w budowanej specyfikacji. Programista musi też zdefiniować widok, w którym będzie ładowany węzeł `Content`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
   <ModulePrefs title="Przykład dla rozdziału 6."
      description="Wyświetla uzyskane informacje społecznościowe i formularz dodawania aktualizacji">
      <Require feature="opensocial-0.9"/>
      <Require feature="osapi" />
   </ModulePrefs>
   <Content type="html" view="canvas">
   <![CDATA[
      <!-- treść widoku -->
   ]]>
   </Content>
</Module>
```

Należy teraz umieścić w węźle `Content` odpowiednie style i kod języka znaczników. Na potrzeby tego przykładu zostaną użyte style rozmieszczające elementy na stronie oraz ustawiające czcionkę, kolory i odstępy. Kod języka znaczników zbuduje kontener podzielony na dwie kolumny. Lewa kolumna będzie zawierała ostatnie aktualizacje dotyczące znajomych użytkownika; prawa kolumna będzie wyświetlała obrazy profili dwunastu spośród tych znajomych. Pod zdjęciami znajduje się formularz, w którym użytkownik może wpisać tytuł, opis i adres URL przeznaczone do opublikowania w strumieniu aktywności:

```
<style type="text/css">
div#gadget{ font:11px arial,helvetica,sans-serif; }
div#gadget div.header{ background-color:#858585;
                       color:#fff; font-weight:bold;
                       font-size:12px;
                       padding:5px;
                       margin:5px; }
div#gadget div#railRight{ width:360px;
                          float:right;
                          border:1px solid #858585;
                          margin:0 0 15px 15px;
                          padding:10px;
                          background-color:#eaeaea; }
div#gadget div#railRight div#friendLinks img{ border:0;
margin:5px;
width:50px;
height:50px; }
```

```
div#gadget div#railRight form{ margin:10px 5px; }
div#gadget div#railRight form label{ font-weight:bold; }
div#gadget div#railRight form input{ width:300px; }
div#gadget div#updates{ margin-left:5px;
                        margin-right:390px; }
div#gadget div#updates div.header{ margin:0; }
</style>

<div id="gadget">
   <div id="railRight">
      <div class="header">Pozostałe profile</div>
      <div id="friendLinks"></div>
      <div class="header">Wyślij powiadomienie do znajomych</div>
      <form name="addActivity" onSubmit="return false;">
         <label for="title">Tytuł:</label><br />
         <input type="text" name="title" id="title" /><br />
         <label for="description">Opis:</label><br />
         <input type="text" name="description" id="description" /><br />
         <label for="url">Adres URL:</label><br />
         <input type="text" name="url" id="url" /><br /><br />
         <button onclick="socialController.addActivity();">Dodaj aktualizację</button>
      </form>
   </div>
   <div id="updates">
      <div class="header">Powiadomienia od Twoich znajomych</div>
      <div id="updateContent"></div>
   </div>
</div>
```

Ostatnim składnikiem tego gadżetu jest warstwa kodu języka JavaScript. Sekcja skryptu obejmuje trzy funkcje odpowiedzialne za obsługę pobierania i ustawiania danych społecznościowych gadżetu. Wspomniane funkcje są stosowane w roli konstruktorów źródeł danych społecznościowych i mechanizmów promocji danych aplikacji wśród znajomych bieżącego użytkownika:

```
<script type="text/javascript">
var socialController = {
   // uzyskuje zdjęcia (obrazy) przypisane do profili znajomych
   fetchProfile: function(insertID){
      // generuje żądanie GET dotyczące profili dwunastu znajomych użytkownika
      osapi.people.get({userId: "@viewer",
                        groupId: "@friends",
                        count: 12}).execute(function(result){
         var friends = result.list;
         var html = '';

         // dla każdego znalezionego znajomego wyświetla obraz będący linkiem do odpowiedniego profilu
         for (var i = 0; i < friends.length; i++){
            html += "<a href='" + friends[i].profileUrl + "'><img src='"
                  + friends[i].thumbnailUrl + "' /></a>";
         }
         document.getElementById(insertID).innerHTML = html;
      });
   },

   // uzyskuje strumień aktualizacji dla znajomych
   fetchUpdates: function(insertID){
      // generuje żądanie GET dotyczące strumieni aktywności trzydziestu znajomych użytkownika
      osapi.activities.get({userId: "@viewer",
                            groupId: "@friends",
                            count: 30}).execute(function(result){
```

```
            var activities = result.list; var html = '';

            // dla każdej czynności tworzy tytuł będący linkiem do jej źródła
            for (var i = 0; i < activities.length; i++){
               html += "<p><a href='" + activities[i].url + "'>"
                     + activities[i].title + "</a><br /></p>";
            }
            document.getElementById(insertID).innerHTML = html;
         });
      },

      // umieszcza nowe powiadomienie w strumieniu aktywności bieżącego użytkownika
      addActivity: function(){
         osapi.activities.create({userId: "@viewer", groupId: "@self",
            activity: {
               title: document.getElementById("title").value,
               body: document.getElementById("description").value,
               url: document.getElementById("url").value
            }
         }).execute();
      }
   };

   // inicjalizuje żądania danych
   socialController.fetchProfile("friendLinks");
   socialController.fetchUpdates("updateContent");
   </script>
```

Funkcja `fetchProfile()` uzyskuje adresy URL i zdjęcia przypisane do profili znajomych bieżącego użytkownika. Pobrane informacje są następnie używane do utworzenia zbioru znaczników obrazów, które są umieszczane w kodzie języka HTML gadżetu.

Funkcja `fetchUpdates()` pobiera aktualizacje ze strumieni aktywności znajomych użytkownika, tworzy kod języka znaczników z tytułami w formie linków, po czym wstawia gotowy kod HTML-a do lewej kolumny aplikacji. Ostatnia funkcja, nazwana `addActivity()`, dodaje nowe powiadomienie do strumienia aktywności użytkownika (po wpisaniu tytułu, opisu i adresu URL w prawej kolumnie).

Dwa ostatnie wiersze tego bloku kodu JavaScriptu wywołują funkcje pobierające dane społecznościowe i wypełniające widok aplikacji (w czasie jej ładowania). Podczas ładowania gadżetu zostaną wyświetlone wszystkie elementy społecznościowe zdefiniowane w węzłach `div` (patrz rysunek 6.6).

Opisana aplikacja zawiera kilka podstawowych funkcji społecznościowych, które można z powodzeniem wykorzystywać do promocji własnych aplikacji i dostosowywania ich działania do potrzeb użytkowników. Wystarczy zastosować choćby część spośród zaproponowanych rozwiązań, aby lepiej odczytywać preferencje użytkowników, sprowokować ich do promowania aplikacji w naszym imieniu i uwzględnić w aplikacji wykaz znajomych użytkownika w ramach serwisu społecznościowego.

Rysunek 6.6. Przykładowy gadżet dla rozdziału 6. ilustrujący sposoby operowania na strumieniu aktywności i profilach społecznościowych

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

Ż

Programowanie

O'REILLY®

Programowanie aplikacji na serwisy społecznościowe

Serwisy społecznościowe w zasadzie z dnia na dzień opanowały internet. Mówią o nich wszyscy i korzystają z nich wszyscy! Przez krótki okres używały ich tylko osoby prywatne, jednak błyskawicznie ich potencjał dostrzegły także firmy. Jest to dla nich najprawdopodobniej najlepszy kanał komunikacji z klientami. Dlatego podczas tworzenia nowych rozwiązań czy nowych serwisów warto rozważyć integrację z popularnymi serwisami społecznościowymi oraz wprowadzenie własnych elementów tego typu.

To zadanie ma ułatwić platforma OpenSocial, na której koncentruje się ta książka. Dowiesz z niej, jak tworzyć niezależne aplikacje dla istniejących serwisów, jak budować grafy powiązań społecznościowych oraz tworzyć produkty spełniające oczekiwania samego autora jako użytkownika usług społecznościowych.

W trakcie lektury nauczysz się odwzorowywać relacje pomiędzy użytkownikami oraz dostosowywać dostarczane im treści na podstawie danych zawartych w ich profilach. Ponadto zdobędziesz solidną dawkę wiedzy na temat bezpieczeństwa oraz najlepszych technik autoryzacji użytkowników na platformie OpenSocial. Sprawdzisz, jak przenieść aplikację napisaną dla Facebooka na platformę OpenSocial, oraz poznasz niuanse konfigurowania środowiska produkcyjnego. Książka ta jest wyjątkową pozycją na rynku, poświęconą platformie OpenSocial. Wykorzystaj jej potencjał i stwórz nowatorskie oprogramowanie!

Poznaj możliwości platformy OpenSocial!

- Skonfiguruj środowisko produkcyjne
- Odwzoruj relacje pomiędzy użytkownikami
- Stwórz interesujące gadżety
- Skorzystaj z zaawansowanych mechanizmów identyfikacji
- Przenieś aplikację z serwisu Facebook na platformę OpenSocial

Postaw na otwarte standardy!

helion.pl
księgarnia internetowa

Nr katalogowy: 11678

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

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

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

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-246-3944-1

Cena 89,00 zł

9 788324 639441