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: 978-83-246-3946-5

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_ebook
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Dla mojej fantastycznej żony Heather
oraz dla naszego małego skarbu, Scarlett.

Spis treści

Słowo wstępne **15**

1. Podstawowe pojęcia związane z kontenerem aplikacji społecznościowych **21**

Czym jest kontener aplikacji społecznościowych? 22
- Profil użytkownika 23
- Znajomi i powiązania użytkownika 24
- Strumień aktywności użytkowników 24

Implementacja zastrzeżonych i otwartych standardów 25
- Implementacja zastrzeżona 25
- Implementacja typu open source 26
- Dlaczego w tej książce zostaną omówione otwarte standardy? 27

Wbudowana aplikacja — tworzenie rozwiązań w ramach czarnej skrzynki 27

Wbudowane zabezpieczenia aplikacji 29
- Ataki XSS 30
- Zasada tego samego pochodzenia i starsze przeglądarki 30
- Pobieranie plików bez wiedzy użytkownika 31
- Zabezpieczanie aplikacji 31

Aplikacja zewnętrzna — integracja danych serwisu społecznościowego poza kontenerem 31

Widoki aplikacji 32
- Widok domowy (mały) 33
- Widok profilu (mały) 34
- Widok kanwy (duży) 35
- Domyślny widok (dowolny) 36

Zagadnienia związane z uprawnieniami aplikacji 37

Aplikacje strony klienckiej kontra aplikacje serwera 38
- Stosowanie systemów szablonów w warstwie znaczników 38
- Stosowanie mieszanego środowiska serwera i klienta 39
- Opóźnianie ładowania mniej ważnej treści 40

Kiedy dobra aplikacja okazuje się zła? 40
- Przenośna aplikacja z animacjami 41
- Niedopracowany widok 42

Aplikacja kopiująca widoki 43
Aplikacja prezentująca zbyt dużo informacji 43
Nierentowna aplikacja 44
Aplikacja informacyjna 45
Studia przypadków dla modeli aplikacji 46
Studium przypadku: gra społecznościowa ze znajomymi 46
Studium przypadku: aplikacje sprzedaży produktów 50
Studium przypadku: aplikacje uwzględniające położenie użytkownika 53
Krótkie wskazówki na początek 56
Należy zdefiniować docelowych odbiorców 57
Możliwie wczesne budowanie punktów integracji z serwisem społecznościowym 57
Budowanie z myślą o elementach komercyjnych 57
Tworzenie dopracowanych, atrakcyjnych widoków 58

2. Odwzorowywanie relacji użytkowników na podstawie grafu powiązań społecznościowych 59

Graf powiązań społecznościowych w internecie 59
Stosowanie grafu rzeczywistych powiązań społecznościowych w wirtualnym świecie 61
Automatyczne dzielenie użytkowników na klastry 62
Prywatność i bezpieczeństwo 62
Budowanie zaufania 63
Udostępnianie prywatnych danych użytkownika: model opt-in kontra model opt-out 63
Model udostępniania za zgodą użytkownika (opt-in) 63
Model wyłączania udostępniania na wniosek użytkownika (opt-out) 64
Zrozumienie modelu relacji 65
Model śledzenia 65
Model połączeń 66
Model grupowy 67
Relacje kontra podmioty 71
Budowanie związków społecznościowych — analiza grafu powiązań społecznościowych Facebooka 72
Budowanie na bazie rzeczywistych tożsamości 72
Zrozumienie najskuteczniejszych kanałów komunikacji 73
Budowanie grup użytkowników 74
Unikanie grafów nieistotnych powiązań społecznościowych 74
Wskazywanie lubianych i nielubianych podmiotów za pośrednictwem protokołu OpenLike 75
Integracja widgetu OpenLike 75
Sposób prezentowania oznaczeń „Lubię to” 76
Podsumowanie 76

3. Tworzenie podstawowych elementów platformy aplikacji społecznościowych 79
Czego nauczysz się w tym rozdziale? 79
Apache Shindig 79
Konfiguracja kontenera Shindig 80
Instalacja kontenera Shindig w systemie Mac OS X (Leopard) 81
Instalacja kontenera Shindig w systemie Windows 84
Testowanie instalacji kontenera Shindig 86
Partuza 87
Wymagania 88
Instalacja kontenera Partuza w systemie Mac OS X (Leopard) 88
Instalacja kontenera Partuza w systemie Windows 91
Testowanie instalacji kontenera Partuza 96
Specyfikacja gadżetu OpenSocial w formacie XML 96
Konfigurowanie aplikacji za pomocą węzła ModulePrefs 97
Elementy Require i Optional 98
Element Preload 98
Element Icon 99
Element Locale 99
Element Link 100
Definiowanie preferencji użytkownika 101
Wyliczeniowe typy danych 103
Treść aplikacji 103
Definiowanie widoków treści 104
Treść wbudowana kontra treść zewnętrzna 110
Budowanie kompletnego gadżetu 111

4. Definiowanie funkcji za pomocą odwołań JavaScriptu do elementów standardu OpenSocial 115
Czego nauczysz się w tym rozdziale? 115
Dołączanie bibliotek JavaScriptu z funkcjami standardu OpenSocial 116
Dynamiczne ustawianie wysokości widoku gadżetu 117
Umieszczanie animacji Flash w ramach gadżetu 118
Wyświetlanie komunikatów dla użytkowników 119
Tworzenie komunikatu 120
Określanie położenia okien komunikatów 123
Definiowanie stylów komunikatów i okien 125
Zapisywanie stanu z preferencjami użytkownika 127
Programowe ustawianie tytułu gadżetu 129
Integracja interfejsu użytkownika gadżetu z podziałem na zakładki 130
Podstawowy gadżet 131
Tworzenie zakładki na podstawie kodu języka znaczników 131
Tworzenie zakładki w kodzie JavaScriptu 132
Uzyskiwanie i ustawianie informacji na temat obiektu TabSet 134

Rozszerzanie kontenera Shindig o własne biblioteki języka JavaScript 136
Budowanie kompletnego gadżetu 140
Przygotowanie specyfikacji XML gadżetu 140
Wyświetlanie gadżetu przy użyciu kontenera Shindig 144

5. Przenoszenie aplikacji, profili i znajomych 145
Czego nauczysz się w tym rozdziale? 145
Ocena obsługi standardu OpenSocial 145
Podstawowe elementy specyfikacji OpenSocial 147
Specyfikacja podstawowego serwera API 148
Specyfikacja podstawowego kontenera gadżetów 148
Specyfikacja serwera społecznościowego interfejsu API 149
Specyfikacja kontenera gadżetów społecznościowych 149
Specyfikacja kontenera OpenSocial 150
Tworzenie rozwiązań dla wielu kontenerów i przenoszenie aplikacji 150
Stosowanie mieszanego środowiska klient-serwer 151
Oddzielanie funkcji społecznościowych od podstawowego kodu aplikacji 151
Unikanie znaczników właściwych konkretnym kontenerom 151
Przenoszenie aplikacji z Facebooka do kontenera OpenSocial 152
Stosowanie ramek iframe dla konstrukcji niebędących aplikacjami społecznościowymi 152
Wyodrębnianie logiki funkcji Facebooka 153
Oddzielenie kodu znaczników (wizualizacji) od logiki programu 153
Stosowanie punktów końcowych REST zamiast języka FQL 153
Stosowanie implementacji z zasadniczą częścią kodu po stronie serwera 154
Personalizacja aplikacji na podstawie danych zawartych w profilu 154
Obiekt Person 154
Metody wymiany danych obiektu Person 155
Pola dostępne w ramach obiektu Person 160
Rozszerzanie obiektu Person 183
Uzyskiwanie profilu użytkownika 189
Promowanie aplikacji z wykorzystaniem znajomych użytkownika 191
Generowanie żądań dotyczących znajomych użytkownika 192
Budowanie kompletnego gadżetu 193
Specyfikacja gadżetu 193
Kod języka znaczników 194
Kod języka JavaScript 195
Uruchamianie gadżetu 197

6. Aktywność użytkowników, publikowanie powiadomień aplikacji i żądanie danych w kontenerze OpenSocial 199
Czego nauczysz się w tym rozdziale? 200
Promocja aplikacji za pomocą strumienia aktywności w kontenerze OpenSocial 200
Personalizacja aplikacji na podstawie powiadomień w strumieniu aktywności 201
Generowanie powiadomień w celu zwiększania liczby użytkowników 202

Pasywne i bezpośrednie publikowanie powiadomień aplikacji 205
Bezpośrednie publikowanie powiadomień aplikacji 206
Pasywne publikowanie powiadomień aplikacji 207
Zrównoważone publikowanie powiadomień 209
Generowanie żądań AJAX i żądań dostępu do danych zewnętrznych 210
Generowanie standardowych żądań dostępu do danych 211
Umieszczanie treści w żądaniach danych 212
Używanie autoryzowanych żądań do zabezpieczania połączeń 213
Budowanie kompletnego gadżetu 221

7. Zaawansowane techniki OpenSocial i przyszłość tego standardu 225
Czego nauczysz się w tym rozdziale? 225
Potokowe przesyłanie danych 225
Rodzaje żądań danych 228
Udostępnianie danych dla żądań zewnętrznych 233
Korzystanie z potokowego przesyłania danych po stronie klienta 234
Obsługa błędów generowanych przez potok danych 237
Parametry dynamiczne 238
Szablony OpenSocial 240
Alternatywny model kodu języka znaczników i danych 241
Wyświetlanie szablonów 243
Wyrażenia 247
Zmienne specjalne 248
Wyrażenia warunkowe 250
Przetwarzanie treści w pętli 253
Łączenie potokowego przesyłania danych i szablonów 258
Pozostałe znaczniki specjalne 260
Biblioteki szablonów 262
Interfejs API języka JavaScript 265
Kilka dodatkowych znaczników — język znaczników OpenSocial 270
Wyświetlanie nazwiska użytkownika — znacznik os:Name 271
Lista wyboru użytkownika — znacznik os:PeopleSelector 271
Wyświetlanie odznaki użytkownika — znacznik os:Badge 272
Ładowanie zewnętrznego kodu HTML — znacznik os:Get 272
Obsługa lokalizacji za pomocą pakietów komunikatów 272
Biblioteki API protokołu OpenSocial REST 275
Dostępne biblioteki 275
Przyszłość standardu OpenSocial: obszary rozwoju 276
Kontenery korporacyjne 276
Mobilna rewolucja 277
Rozproszone frameworki internetowe 277
Standard OpenSocial i rozproszone frameworki internetowe 277
Standard Activity Streams 278
Protokół PubSubHubbub 278

Protokół Salmon 279
Protokół Open Graph 280
Budowanie kompletnego gadżetu 281

8. Zagadnienia związane z bezpieczeństwem aplikacji społecznościowych 287
Czego nauczysz się w tym rozdziale? 287
Wykonywanie zewnętrznego kodu za pośrednictwem ramek iframe 288
Bezpieczny model — projekt Caja 288
Dlaczego warto używać kompilatora Caja? 289
Rodzaje ataków — jak Caja chroni użytkownika? 289
Przekierowywanie użytkowników bez ich zgody 290
Śledzenie historii przeglądarki użytkownika 290
Wykonywanie dowolnego kodu za pomocą funkcji document.createElement 291
Rejestrowanie klawiszy naciskanych przez użytkownika 291
Konfiguracja kompilatora Caja 293
Przetwarzanie skryptów za pomocą kompilatora Caja z poziomu wiersza poleceń 295
Zabezpieczanie kodu HTML-a i JavaScriptu 295
Zmiana docelowego formatu kodu 300
Uruchamianie kompilatora Caja z poziomu aplikacji internetowej 301
Stosowanie kompilatora Caja dla gadżetu OpenSocial 303
Dodawanie kompilatora Caja do gadżetu 303
Praktyczny przykład 304
Wczesne wykrywanie niebezpiecznych elementów JavaScriptu za pomocą narzędzia JSLint 305
Eksperymenty w środowisku Caja Playground 306
Wskazówki dotyczące pracy w środowisku Caja 306
Implementacja modułowego kodu — kompilatora Caja nie należy stosować dla całego projektu 307
Stosowanie wstępnie przetworzonych bibliotek JavaScriptu 308
Nie należy używać Firebuga dla przetworzonego kodu źródłowego JavaScriptu 309
Nie należy umieszczać zdarzeń w kodzie języka znaczników 309
Centralizacja kodu JavaScriptu — stosowanie wyłącznie żądań danych i kodu języka znaczników 311
Lżejsza alternatywa dla kompilatora Caja: narzędzie ADsafe 312
ADsafe kontra Caja — którego narzędzia używać? 313
Jak zaimplementować środowisko ADsafe? 314
Konfiguracja obiektu środowiska ADsafe 314
Obiekt DOM 315
Wybór konkretnych węzłów DOM za pomocą zapytań 317
Praca z obiektami pakietów 321
Dołączanie zdarzeń 327
Definiowanie bibliotek 328

Budowanie kompletnego gadżetu 329
Źródło danych 330
Sekcja nagłówkowa: dołączane skrypty i style 330
Ciało: warstwa języka znaczników 332
Ciało: warstwa języka JavaScript 332
Ostateczny wynik 334
Podsumowanie 335

9. Zabezpieczanie dostępu do grafu powiązań społecznościowych za pomocą standardu OAuth 337
Punkt wyjścia — uwierzytelnianie podstawowe 337
Implementacja uwierzytelniania podstawowego — jak to działa? 338
Wady stosowania uwierzytelniania podstawowego 339
Standard OAuth 1.0a 340
Przepływ pracy w standardzie OAuth 1.0a 341
Standard OAuth z perspektywy użytkownika końcowego 348
Dwuetapowa autoryzacja OAuth kontra trzyetapowa autoryzacja OAuth 350
Przykład implementacji trzyetapowej autoryzacji OAuth 354
Narzędzia i wskazówki związane z diagnozowaniem problemów 369
OAuth 2 373
Przepływ pracy w standardzie OAuth 2 373
Przykład implementacji: Facebook 381
Przykład implementacji: żądanie dodatkowych informacji na temat użytkownika w procesie autoryzacji OAuth w serwisie Facebook 392
Przykład implementacji: aplikacja z perspektywy użytkownika końcowego 394
Wskazówki dotyczące diagnozowania problemów z żądaniami 396
Podsumowanie 400

10. Przyszłość serwisów społecznościowych: definiowanie obiektów społecznościowych za pośrednictwem rozproszonych frameworków sieciowych 401
Czego nauczysz się w tym rozdziale? 401
Protokół Open Graph — definiowanie stron internetowych jako obiektów społecznościowych 402
Wzloty i upadki metadanych 403
Działanie protokołu Open Graph 403
Implementacja protokołu Open Graph 404
Rzeczywisty przykład: implementacja protokołu Open Graph w serwisie Facebook 410
Praktyczna implementacja: odczytywanie danych protokołu Open Graph ze źródła w internecie 413
Wady protokołu Open Graph 419
Strumienie aktywności: standaryzacja aktywności społecznościowych 420
Dlaczego warto zdefiniować standard dla aktywności? 421
Implementacja standardu Activity Streams 421

Typy obiektów 424
Czasowniki 426
WebFinger — rozszerzanie grafu powiązań społecznościowych na podstawie adresów poczty elektronicznej 429
Od finger do WebFinger: geneza protokołu WebFinger 429
Implementacja protokołu WebFinger 430
Wady protokołu WebFinger 432
Protokół OExchange — budowanie grafu udostępniania treści społecznościowych 433
Jak działa protokół OExchange? 433
Zastosowania protokołu OExchange 434
Implementacja protokołu OExchange 435
Protokół PubSubHubbub: rozpowszechnianie treści 440
Jak działa protokół PubSubHubbub? 441
Zalety z perspektywy wydawców i subskrybentów 443
Serwery hubów i usługi implementacji 445
Biblioteki przepływu pracy 445
Budowanie wydawcy w języku PHP 446
Budowanie wydawcy w języku Python 448
Budowanie subskrybenta w języku PHP 450
Budowanie subskrybenta w języku Python 452
Protokół Salmon: ujednolicenie stron konwersacji 455
Działanie protokołu Salmon 455
Budowanie rozwiązań na bazie protokołu PubSubHubbub 457
Ochrona przed nadużyciami i spamem 458
Przegląd implementacji 459
Podsumowanie 460

11. Rozszerzanie grafu powiązań społecznościowych za pomocą standardu OpenID 461

Standard OpenID 461
Klucz do sukcesu — decentralizacja 462
Udoskonalenia względem tradycyjnego logowania 462
Dostęp do istniejącej bazy danych użytkowników i grafu powiązań społecznościowych 462
Czy już teraz dysponuję implementacją standardu OpenID? Gdzie mam jej szukać? 463
Procedura uwierzytelniania OpenID 464
Krok 1.: żądanie logowania przy użyciu identyfikatora OpenID 464
Krok 2.: operacja odkrywania w celu wyznaczenia adresu URL punktu końcowego 465
Krok 3.: żądanie uwierzytelnienia użytkownika 466
Krok 4.: udostępnienie stanu sukcesu lub niepowodzenia 467
Dostawcy OpenID 469
Omijanie problemów odkrywania domen w standardzie OpenID 469

Rozszerzenia standardu OpenID 471
Rozszerzenie Simple Registration 472
Rozszerzenie Attribute Exchange 473
Rozszerzenie Provider Authentication Policy Extension 479
Aktualnie tworzone rozszerzenia 483
Przykład implementacji: OpenID 484
Implementacja standardu OpenID w języku PHP 485
Implementacja standardu OpenID w języku Python 497
Typowe błędy i techniki diagnostyczne 508
Niezgodność adresu URL wywołań zwrotnych 509
Brak możliwości odkrycia identyfikatora OpenID 509
Podsumowanie 510

12. Uwierzytelnianie hybrydowe — wygoda użytkownika i pełen dostęp do profilu 511

Rozszerzenie hybrydy standardów OpenID i OAuth 511
Istniejące implementacje 512
Kiedy należy używać standardu OpenID, a kiedy jego hybrydy ze standardem OAuth? 512
Pytania, na które warto sobie odpowiedzieć przed wybraniem właściwego rozwiązania 512
Zalety i wady: standardowa implementacja OpenID 513
Zalety i wady: uwierzytelnianie hybrydowe 514
Przebieg uwierzytelniania w modelu hybrydowym na bazie standardów OpenID i OAuth 515
Kroki 1. i 2.: odkrywanie (pierwsze dwa kroki procedury OpenID) 516
Krok 3.: akceptacja uprawnień przez użytkownika 516
Krok 4.: przekazanie stanu akceptacji/odrzucenia żądania OpenID i parametrów rozszerzenia hybrydowego 517
Krok 5.: wymiana wstępnie zaakceptowanego tokenu żądania na token dostępu 519
Krok 6.: generowanie podpisanych żądań dostępu do chronionych danych użytkownika 520
Przykład implementacji: OpenID, OAuth i Yahoo! 521
Konfiguracja aplikacji: uzyskanie kluczy standardu OAuth na potrzeby procesu uwierzytelniania hybrydowego 521
Implementacja uwierzytelniania hybrydowego w języku PHP 522
Implementacja uwierzytelniania hybrydowego w języku Python 533
Podsumowanie 546

Dodatek A Podstawowe zagadnienia związane z budową aplikacji internetowych 547

Dodatek B Słownik pojęć 563

Skorowidz 567

Słowo wstępne

Swoją przygodę z aplikacjami społecznościowymi rozpocząłem w momencie, w którym serwis Facebook udostępnił platformę programistyczną (w 2007 roku), stwarzając takim ludziom jak ja możliwość operowania na bogatych danych społecznościowych. Na podstawie tych danych można było skutecznie popularyzować nowe rozwiązania i efektywniej personalizować ustawienia. W tym czasie budowałem aplikacje społecznościowe dla serwisu *CBSSports.com*, które na podstawie informacji o użytkownikach wzbogacały dane fikcyjnych sportów, tworząc wysoce spersonalizowane reguły rozgrywki.

Dopiero w 2008 roku, kiedy dołączyłem do zespołu odpowiedzialnego za integrację produktów partnerskich z usługą Yahoo! Developer Network, po raz pierwszy odkryłem zalety modelu open source w świecie wytwarzania aplikacji społecznościowych (na przykładzie projektu OpenSocial). Największą zaletą standardu OpenSocial była nie tyle możliwość wdrażania tej samej aplikacji w wielu kontenerach OpenSocial (z czasem okazało się, że ta koncepcja była nietrafiona), ile możliwość budowy aplikacji społecznościowych przy użyciu narzędzi open source oraz poznania wszystkich aspektów funkcjonowania dostępnych platform. Poświęciłem sporo czasu poznawaniu i analizie technik wykorzystywania relacji budowanych przez użytkowników w internecie do wzbogacania i lepszego personalizowania ich doznań w wirtualnym świecie. Właśnie wtedy rozpoczęła się moja kariera gorącego orędownika technologii społecznościowych open source.

Specyfikacja OpenSocial była dla mnie tylko początkiem wielkiej przygody — to dzięki niej poznałem kontener Shindig, standardy OpenID i OAuth (stworzone odpowiednio z myślą o uwierzytelnianiu i autoryzacji), technologie zabezpieczania zewnętrznego kodu Caja i ADsafe oraz nieco młodsze specyfikacje rozproszonych frameworków internetowych, jak Activity Streams, PubSubHubbub czy protokół Open Graph. Szybko zdałem sobie sprawę z tego, że istnieje mnóstwo technologii open source umożliwiających konstruowanie bogatych frameworków społecznościowych. Dostępne technologie i specyfikacje tworzą rozbudowane warstwy rozwiązań w wyjątkowo prosty sposób, na podstawie otwartych i zrozumiałych metodyk.

Właśnie te społecznościowe technologie i specyfikacje są tematem tej książki. Każdy rozdział przybliża nową warstwę składającą się na aplikacje i platformy społecznościowe, które cechują się ogromnym potencjałem rozwoju. Zaczniemy od przeanalizowania podstawowych zagadnień związanych z aplikacjami i kontenerami społecznościowymi, aby następnie przejść do szczegółowego omawiania technologii, których użyto do budowy tych rozwiązań. Po informacjach na temat podstawowych zagadnień przejdziemy do analizy technologii zabezpieczających zewnętrzny kod wykonywany w ramach kontenera oraz do omówienia technik zabezpieczania

danych użytkownika. Przyjrzymy się także standardowej architekturze logowania. Po przeanalizowaniu wszystkich tych skomplikowanych warstw przystąpimy do omawiania rozproszonych frameworków internetowych, które dobrze ilustrują próby standaryzacji technik rozpowszechniania aktywności oraz odkrywania rozbudowanych danych o użytkownikach za pośrednictwem serwisów internetowych lub adresów poczty elektronicznej. I wreszcie omówimy kilka bardzo obiecujących przyszłych standardów świata aplikacji społecznościowych.

Treść tej książki jest efektem wieloletniej pracy nad integracją różnych rozwiązań, przede wszystkim potencjału i funkcji technologii open source, oraz współpracy z innymi programistami i firmami zainteresowanymi tworzeniem bogatych rozwiązań społecznościowych na bazie serwisu Yahoo!. Praca nad tą książką była dla mnie źródłem ogromnej satysfakcji, ponieważ miałem przyjemność jednocześnie uczyć się i dzielić się swoją wiedzą na temat technologii integrujących rozwiązania społecznościowe, które są stosowane w prawdziwych aplikacjach i serwisach.

Kto powinien sięgnąć po tę książkę?

Ponieważ ta książka dotyczy wielu różnych aspektów wytwarzania społecznościowych aplikacji internetowych, specyfikacji kontenerów, architektur i standardów, może być ciekawym źródłem wiedzy dla odbiorców zainteresowanych różnymi zagadnieniami i dysponujących wiedzą fachową na różnych poziomach, w tym (choć nie tylko):

- programistów społecznościowych aplikacji internetowych, którzy tworzą rozwiązania dla takich serwisów jak Facebook, iGoogle, Orkut czy YAP (lub dowolnych innych serwisów społecznościowych oferujących obsługę zewnętrznych aplikacji);
- architektów platform aplikacji i inżynierów rozwiązań dla serwerów, którzy pracują nad platformami dla aplikacji społecznościowych;
- inżynierów interfejsu użytkownika, którzy chcą skorzystać z możliwości dostosowywania wyglądu aplikacji i skuteczniejszego adresowania przekazu na podstawie bogatych danych społecznościowych oferowanych przez omawiane technologie;
- miłośników technologii i programistów-hobbystów, którzy realizują projekty w niewielkiej skali (zwykle na własny użytek) i którzy chcą wykorzystać potencjał serwisów społecznościowych;
- miłośników technologii open source, którzy chcą lepiej zrozumieć działanie mechanizmów używanych do promocji danych i standardów społecznościowych;
- programistów aplikacji internetowych i zespołów projektowych zainteresowanych tworzeniem systemów członkowskich i mechanizmów uwierzytelniania;
- specjalistów i inżynierów ds. zabezpieczeń, którzy chcą pogłębić swoją wiedzę dotyczącą bezpieczeństwa aplikacji i serwisów społecznościowych.

Treść książki

W książce omówiono wiele technologii i narzędzi opracowanych z myślą o tworzeniu serwisów społecznościowych — od implementowania kontenerów i prostych aplikacji do budowy bogatych grafów powiązań społecznościowych.

Każdy rozdział w większym lub mniejszym stopniu bazuje na wiedzy na temat technologii społecznościowych przekazanej we wcześniejszych rozdziałach. Poniżej wymieniono najważniejsze zagadnienia prezentowane w poszczególnych rozdziałach tej książki:

Rozdział 1.
: Wprowadza podstawy aplikacji, systemów i rozwiązań open source, których opanowanie będzie bardzo pomocne podczas implementowania technologii prezentowanych w pozostałych rozdziałach książki.

Rozdział 2.
: Wyjaśnia zagadnienia związane z grafem powiązań społecznościowych. Rozdział zawiera szczegółową analizę poszczególnych cech tego grafu.

Rozdział 3.
: Ten rozdział można traktować jako punkt wyjścia dla procesu dalszego poznawania technik wytwarzania aplikacji społecznościowych. Omówiona została w nim konstrukcja kontenera społecznościowego pełniącego funkcję hosta dla zewnętrznych aplikacji.

Rozdział 4.
: Rozdział zawiera omówienie rozszerzeń i funkcji zaimplementowanych w bibliotekach standardu OpenSocial dla języka JavaScript.

Rozdziały 5. i 6.
: Te dwa rozdziały zawierają pogłębioną analizę specyfikacji OpenSocial. Znajdują się tu informacje na temat podstawowych aspektów działania platformy aplikacji społecznościowych, w tym implementacji grafu powiązań społecznościowych i modelu architektury danych.

Rozdział 7.
: W ostatnim rozdziale poświęconym standardowi OpenSocial omówione zostały takie zaawansowane zagadnienia jak szablony czy metody potokowego przesyłania danych. Rozdział zawiera też analizę przewidywanych kierunków rozwoju standardu OpenSocial.

Rozdział 8.
: W rozdziale omówione zostały modele zabezpieczania zewnętrznego kodu oraz techniki ochrony kontenera i jego użytkowników przed złośliwym oprogramowaniem (za pomocą systemów przetwarzających kod strony klienckiej).

Rozdział 9.
: W tym rozdziale znajdują się informacje na temat technik uwierzytelniania użytkowników i aplikacji za pomocą standardu OAuth. W rozdziale opisana została zarówno specyfikacja OAuth 1, jak i nowsza wersja OAuth 2.

Rozdział 10.
: Rozdział zawiera szczegółowe omówienie eksperymentalnych i nowych technologii umożliwiających konstruowanie grafów powiązań społecznościowych, strumieni aktywności i rozproszonych frameworków sieciowych.

Rozdziały 11. i 12.
: Te dwa ostatnie rozdziały zostały poświęcone uwierzytelnianiu użytkowników i zabezpieczaniu tego procesu za pomocą standardu OpenID oraz hybrydy standardów OpenID i OAuth.

Stosowanie stosu technologii open source

Ponieważ ta książka ma na celu przede wszystkim prezentację podstawowych technik tworzenia aplikacji społecznościowych, kontenerów i grafów powiązań społecznościowych przy użyciu dostępnych rozwiązań open source, warto wymienić te technologie.

Do najważniejszych technologii open source, które zostaną omówione w tej książce, należą:

- standard OpenSocial umożliwiający przetwarzanie grafu powiązań społecznościowych i wytwarzanie aplikacji;
- kontenery Shindig i Partuza jako implementacje standardu OpenSocial;
- standard OAuth umożliwiający zabezpieczanie aplikacji i procesu autoryzacji użytkowników;
- standard OpenID upraszczający uwierzytelnianie użytkownika oraz rozszerzenie hybrydy standardów OpenID i OAuth;
- kompilatory Caja i ADsafe zabezpieczające kod strony klienckiej;
- protokół Open Graph umożliwiający przetwarzanie obiektów społecznościowych;
- specyfikacja Activity Streams jako podstawowa technologia publikacji treści aktywności;
- protokół WebFinger, czyli technika odkrywania publicznych danych użytkownika na podstawie jego adresu poczty elektronicznej;
- standard OExchange jako sposób udostępniania dowolnych adresów URL dowolnym innym usługom w internecie;
- protokół PubSubHubbub umożliwiający rozpowszechnianie wśród wielu serwisów subskrybenckich dyskusji prowadzonych przez użytkowników w serwisie dostawcy;
- protokół Salmon jako rozwinięcie protokołu PubSubHubbub, dzięki któremu jest możliwe scalanie dyskusji prowadzonych przez użytkowników w serwisie wydawcy i w wielu serwisach subskrybentów.

Przy okazji omawiania wymienionych technologii będziemy porównywać dostępne rozwiązania z wieloma zastrzeżonymi standardami stosowanymi przez najważniejszych przedstawicieli tej branży. Taki model prezentacji wiedzy pozwoli lepiej zrozumieć potencjał rozwiązań open source oraz skutki wykorzystywania poszczególnych technologii.

Konwencje stosowane w książce

W książce zastosowano następujące konwencje typograficzne:

Tekst pogrubiony
: W ten sposób są zapisywane nowe terminy.

Kursywa
: Kursywa jest stosowana dla adresów poczty elektronicznej, nazw plików, rozszerzeń plików, ścieżek i katalogów, a także tytułów menu, opcji menu, przycisków i skrótów klawiszowych (jak *Alt* czy *Ctrl*).

`Czcionka stałej szerokości`
: W ten sposób są zapisywane polecenia, opcje, przełączniki, zmienne, atrybuty, klucze, funkcje, typy, klasy, przestrzenie nazw, metody, moduły, właściwości, parametry, wartości, obiekty, zdarzenia, metody obsługujące zdarzenia, znaczniki języków XML i HTML, makra, zawartość plików oraz dane wynikowe zwracane przez polecenia.

`Pogrubiona czcionka stałej szerokości`
: W ten sposób są oznaczane polecenia i inne fragmenty, które należy stosować w niezmienionej formie.

`Czcionka stałej szerokości pisana kursywą`
: Kursywa służy do wyróżniania fragmentów, które należy zastąpić konkretnymi wartościami.

W ten sposób są oznaczone wskazówki, sugestie i ogólne uwagi.

W ten sposób są oznaczone potencjalne zagrożenia.

Stosowanie przykładów kodu

Ta książka ma ułatwić czytelnikowi realizację konkretnych zadań. Ogólnie kod źródłowy prezentowany w książce można swobodnie stosować w programach i dokumentacji. Nie oczekuję wniosków o zgodę, chyba że czytelnik planuje ponowną publikację istotnych fragmentów tego kodu. Na przykład pisanie programów obejmujących wiele fragmentów kodu dołączonego do tej książki nie wymaga dodatkowych pozwoleń. Zgoda wydawcy jest natomiast wymagana w przypadku sprzedaży lub dystrybucji płyt CD-ROM z przykładami zaczerpniętymi z tej książki. Zgoda nie jest konieczna także w przypadku cytowania tekstu tej książki lub zawartych w niej przykładów kodu. Jest jednak niezbędna, jeśli istotna część przykładowego kodu z tej książki ma trafić do dokumentacji jakiegoś produktu.

Będziemy wdzięczni za stosowne przypisy (które nie są wymagane). Przypis zwykle obejmuje tytuł, autora, wydawcę i numer ISBN.

Każdy, kto nie jest pewien, czy planowany sposób użycia przykładowego kodu mieści się w granicach, które nie wymagają dodatkowej zgody, może skontaktować się z wydawcą.

Podziękowania

Po pierwsze, chciałbym podziękować mojej żonie, Heather, za cierpliwość wykazywaną przez te miesiące, kiedy zarywałem noce w związku z pracą nad tą książką. Dziękuję też za nieustające wsparcie.

Chciałbym podziękować także Mary Treseler z wydawnictwa O'Reilly za cenne wskazówki i odpowiedzi na moje niezliczone pytania oraz za przeprowadzenie mnie przez trudny proces tworzenia tej książki.

Dziękuję także Rachel Monaghan, redaktor tej książki, której jestem wdzięczny za nadanie rozdziałom właściwej formy.

Chciałbym też wyrazić wdzięczność wszystkim recenzentom książki: Matthew Russellowi, Billowi Dayowi, Henry'emu Saputrze, Markowi Weitzlowi i Josephowi Caterze. Dziękuję wam wszystkim za wskazanie usterek, zanim zyskały nieśmiertelność w druku, za sugerowanie bezcennych poprawek w tekście, za pokazywanie treści, która nie była dostatecznie dobra, aby trafić do tej książki.

Jestem wdzięczny także moim rodzicom i siostrze, którzy zawsze oferowali mi wsparcie i którzy nauczyli mnie, że dzięki ciężkiej pracy mogę osiągnąć dosłownie wszystko.

I wreszcie wielkie podziękowania należą się Havi Hoffman, która odpowiada za program Yahoo! Press w firmie Yahoo!. Bez jej pomocy i wsparcia ta książka mogłaby nigdy nie powstać.

ROZDZIAŁ 1.

Podstawowe pojęcia związane z kontenerem aplikacji społecznościowych

Rosnąca popularność takich serwisów społecznościowych jak Facebook, LinkedIn, MySpace, Yahoo! Application Platform (YAP) i setek innych rozwiązań na całym świecie pokazuje, jak dalece zmieniły się sposoby korzystania użytkowników z internetu i jak sam internet komunikuje się ze swoimi użytkownikami. Statyczne strony internetowe to relikt przeszłości. Został on bezpowrotnie zastąpiony koncepcją serwisów i aplikacji, których standardowe działanie polega na dostarczaniu użytkownikom funkcji dostosowanych do ich indywidualnych preferencji. Internet szybko przybrał postać wielkiej społeczności ludzi, którzy doceniają wartość doświadczeń i interakcji społecznych za pośrednictwem odpowiednich serwisów. Tak jak w rzeczywistym świecie, życie w internecie obejmuje różne formy i cele komunikacji, w tym komunikację z przyjaciółmi i wymianę informacji na gruncie zawodowym. Ludzie instynktownie budują odpowiednie kategorie zachowań społecznych i czerpią rozmaite korzyści z kontaktów z innymi ludźmi w poszczególnych przestrzeniach (prywatnej, biznesowej itp.).

Właśnie z myślą o tych potrzebach są tworzone aplikacje serwisów społecznościowych. Programiści aplikacji społecznościowych mogą pomóc w odkrywaniu i czerpaniu korzyści z komunikacji za pośrednictwem internetu. W tradycyjnym modelu programiści budowali swoje produkty, wdrażali je i próbowali dostosowywać zachowania użytkowników do przyjętych wcześniej założeń. Serwisy społecznościowe pozwalają programistom dostosowywać same aplikacje do preferencji użytkowników, ponieważ już w momencie tworzenia oprogramowania istnieje przestrzeń obejmująca nie tylko liczną bazę użytkowników, ale też znane, sprawdzone zjawiska społeczne. Wspomniana przestrzeń stanowi swoisty **kontener aplikacji społecznościowych**.

W tym rozdziale zostanie omówionych wiele zagadnień. Spróbuję też odpowiedzieć na następujące pytania:

- Czym są kontenery aplikacji społecznościowych i jakie mają funkcje?
- Co różni otwarte i zastrzeżone standardy?
- Jakie są różnice dzielące poszczególne typy środowisk wytwarzania aplikacji i na które zagrożenia związane z bezpieczeństwem należy zwracać szczególną uwagę?
- Z jakich elementów składa się interfejs użytkownika aplikacji?

- Czym są i do czego są wykorzystywane uprawnienia aplikacji?
- Których błędów (na przykładach rzeczywistych serwisów i systemów) należy szczególnie unikać podczas projektowania własnych aplikacji?
- Które spośród rzeczywistych modeli aplikacji sprawdziły się w przeszłości?
- Czy istnieją jakieś krótkie wskazówki, które można by przekazać jeszcze przed przystąpieniem do pracy?

W pierwszej części tego rozdziału wyjaśnię, czym właściwie jest kontener aplikacji.

Książka zawiera wiele przykładowych gadżetów, aplikacji i programów. Dla wygody czytelnika dodałem wszystkie najważniejsze przykłady do repozytorium Github, tak aby można je było łatwo integrować i wdrażać. Repozytorium jest dostępne pod adresem *https://github.com/jcleblanc/programming-social-applications*.

Czym jest kontener aplikacji społecznościowych?

Serwisy społecznościowe stanowią nieodłączną część codziennego życia — niemal każdy korzysta z Facebooka do komunikowania się z przyjaciółmi i rodziną, z serwisu LinkedIn do komunikowania się ze współpracownikami i partnerami biznesowymi itp. Serwisy tego typu na stałe wpisały się w nasze codzienne zwyczaje. Serwisy społecznościowe próbują dodatkowo zwiększyć swój udział w bazie użytkowników między innymi poprzez oferowanie podmiotom zewnętrznym możliwości budowania aplikacji, które będą działały w ramach tych serwisów.

Na podstawowym poziomie aplikacje tego typu mogą udostępniać użytkownikom funkcje oryginalnego serwisu społecznościowego wraz z elementami, które wcześniej nie były dostępne w tym serwisie. W pewnych przypadkach takie aplikacje mogą nawet stanowić punkty integracji odpowiednich serwisów.

Serwis występujący w roli hosta takiej dodatkowej aplikacji (czyli udostępniający dane społecznościowe swojej bazy użytkowników w celu przetwarzania ich przez tę aplikację) jest w istocie **kontenerem** tej aplikacji. Relacje łączące ten kontener z dodatkową aplikacją przynoszą korzyści obu stronom:

- Kontener ma większą wartość z perspektywy swoich użytkowników, ponieważ obsługuje dodatkową treść dołączaną do profili bądź nowe mechanizmy łączenia tej treści, co ostatecznie wydłuża czas spędzany przez użytkowników w serwisie.
- Aplikacja zyskuje nowy, szeroki rynek, na którym może promować swoją treść. Co więcej, aplikacja od samego początku może korzystać z grafu połączeń społecznościowych zbudowanego na poziomie kontenera. Aplikacja może wykorzystać te połączenia do przyciągania nowych użytkowników do oryginalnego serwisu (kontenera) lub budowy dodatkowej bazy użytkowników własnych usług.

Jednym z przykładów korporacyjnego kontenera aplikacji społecznościowych jest oprogramowanie firmy Jive Software. Firma mogłaby oczywiście opracować własne rozwiązania do badania opinii użytkowników, jednak zdecydowała się udostępnić platformę dla niezależnych programistów konstruujących aplikacje — w tej sytuacji odpowiednie funkcje są oferowane przez aplikację firmy SurveyGizmo. Na tak zbudowanych relacjach korzystają oba przedsiębiorstwa.

Kontener aplikacji społecznościowych zwykle obejmuje przynajmniej trzy kategorie informacji o społeczności użytkowników. Każda z tych kategorii może być z powodzeniem wykorzystywana przez dodatkowe aplikacje:

Profil użytkownika
: Informacje wpisane przez samego użytkownika na swój temat.

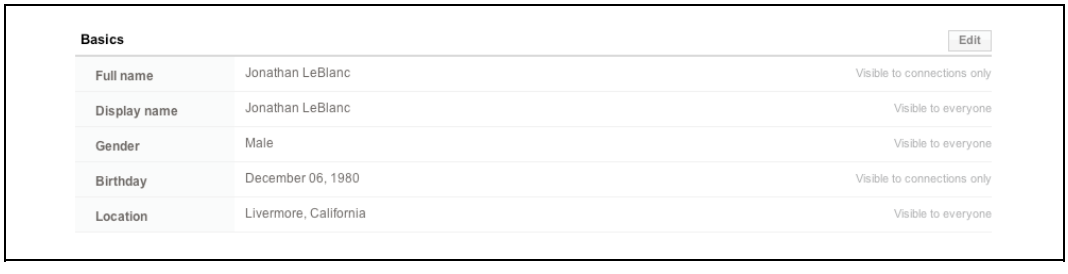
Znajomi i powiązania
: Graf społeczności użytkowników w formie rozbudowanej sieci wzajemnie powiązanych kontaktów.

Strumień aktywności
: Wiadomości dla użytkownika, czyli w istocie zbiorcza perspektywa jego aktywności w sieci (z uwzględnieniem powiadomień o aktualizacjach danych znajomych i wzajemnych powiązań).

Każdy z tych elementów ułatwia dostosowywanie kontenera aplikacji społecznościowych do rzeczywistych potrzeb użytkowników. Co jeszcze ważniejsze, wymienione elementy stanowią gotowy punkt wyjścia dla programistów aplikacji, którzy mogą niemal od razu dotrzeć do licznej grupy potencjalnych odbiorców swoich produktów i aplikacji (bez gotowego kontenera musieliby tworzyć własne serwisy prezentowania informacji i budować zupełnie nowe grafy powiązań społecznościowych).

Profil użytkownika

Profil użytkownika (patrz rysunek 1.1) składa się z danych osobowych, jak nazwisko, data urodzenia, strona internetowa, zainteresowania, zdjęcia, miejsce zamieszkania i mnóstwo innych szczegółów udostępnianych znajomym (lub całemu światu, zależnie od wybranych ustawień ochrony prywatności).



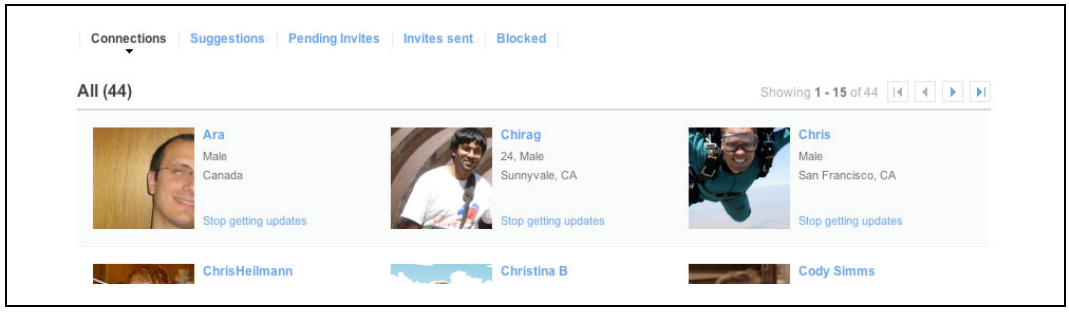
Rysunek 1.1. Podstawowy profil użytkownika

Z perspektywy programisty aplikacji profil użytkownika jest bezcennym źródłem wiedzy, którą można wykorzystać do konstruowania aplikacji oferujących spersonalizowane funkcje (projektowane z myślą o ściśle wyselekcjonowanych grupach odbiorców). Wielu użytkowników kontenerów aplikacji internetowych jest gotowych udostępniać jak najwięcej informacji na swój temat. Chcą mieć własny kąt w internecie, skąd będą mogli się komunikować z przyjaciółmi, gdzie będą przechowywać zdjęcia lub podejmować dowolne inne działania społecznościowe, które wydają im się najwłaściwsze. Co więcej, wiele kontenerów udostępnia statystyki stopnia kompletności profilu, dodatkowo zachęcając użytkowników do tworzenia pełnych profili i maksymalnego zwiększania ich zaangażowania w działanie tych kontenerów. Z perspektywy

kontenerów takie działanie ułatwia rozwój bazy zaangażowanych użytkowników i podnosi liczbę aktywnych użytkowników, co z kolei jest korzystne dla programistów aplikacji, którzy dążą do personalizacji funkcji oferowanych poszczególnym użytkownikom.

Znajomi i powiązania użytkownika

Znajomi użytkownika i jego powiązania stanowią podstawę grafu społeczności w ramach kontenera aplikacji społecznościowych. Ludzie tworzący swoje profile dodają do swoich sieci powiązań przyjaciół, członków rodzin, współpracowników i wiele innych osób, z którymi czują się w ten czy inny sposób związani (w świecie rzeczywistym lub tylko wirtualnym). Przykładową wizualizację powiązań społecznościowych użytkownika pokazano na rysunku 1.2.



Rysunek 1.2. Znajomi przypisani do profilu w serwisie społecznościowym

Użytkownicy budujący swoje relacje w świecie wirtualnym zwykle dzielą te powiązania na kategorie, jak przyjaciele, rodzina czy współpracownicy. Podczas tworzenia aplikacji dobre rozumienie istoty tych kategorii może znacznie ułatwić identyfikację najlepszych metod kierowania oferowanej treści do właściwych odbiorców.

Strumień aktywności użytkowników

Jednym z najważniejszych aspektów interakcji z serwisem społecznościowym jest strumień aktywności użytkowników, czyli w istocie kanały powiadamiania o zdarzeniach. Taki kanał (patrz rysunek 1.3) pozwala prezentować użytkownikowi nie tylko zestawienie jego własnych czynności i aktualizacji statusu, ale też zbiór powiadomień o zdarzeniach dotyczących powiązań i znajomych.

Aplikacje w ramach kontenera często nie są promowane w najważniejszych, najbardziej widocznych miejscach na ekranie. Oznacza to, że warunkiem pozyskania odbiorców w wielu przypadkach jest skuteczne wykorzystywanie funkcji, które cieszą się największym zainteresowaniem poszczególnych użytkowników.

Ponieważ właśnie strumień aktywności jest podstawowym punktem interakcji z użytkownikami, właśnie w tym obszarze należy szukać sposobów docierania do odbiorców. Skuteczne włączenie czynności promujących aplikację do strumienia aktywności umożliwia programiście docieranie do zupełnie nowej grupy odbiorców (potencjalnych użytkowników) — znajomych i powiązań bieżącego użytkownika.

Rysunek 1.3. Strumień aktywności społecznościowej użytkownika

Implementacja zastrzeżonych i otwartych standardów

Skoro serwisy społecznościowe stale walczą o dominację w wirtualnym świecie, pytanie o implementację zastrzeżonych lub otwartych standardów w ramach tworzonego kontenera jest nieuniknione. Czy należy zaimplementować własne, niestandardowe rozwiązania na potrzeby wszystkich aspektów kontenera aplikacji społecznościowych, czy raczej wystarczy postępować według gotowej specyfikacji (opracowanej przy udziale najważniejszych graczy w tej branży)?

Obie metody implementacji mają swoje wady i zalety, zatem warto im się przyjrzeć nieco bliżej.

Implementacja zastrzeżona

Implementacja otwartej specyfikacji kontenera wymuszającej obsługę wielu różnych aspektów regionalnych i wymagań nie zawsze jest korzystna. W takim przypadku budowa własnego, niestandardowego oprogramowania pod kątem konkretnych potrzeb wydaje się lepszym rozwiązaniem. Taki model ma wiele zalet z perspektywy programistów implementujących kontener:

- Tak budowane oprogramowanie będzie lepiej dostosowane do potrzeb i wymagań konkretnego kontenera, co pozwoli ograniczyć liczbę zbędnych funkcji.
- Baza kodu jest niezależna od wymagań tej czy innej otwartej specyfikacji. Oznacza to, że jeśli kod będzie wymagał zmiany niezgodnej z początkowo wybraną specyfikacją, będzie można wprowadzić niezbędną modyfikację bez konieczności odpowiedniego dostosowywania otwartej specyfikacji (i włączenia zmian do przyszłych wersji standardu) i zarządzania ewentualnymi różnicami pomiędzy kontenerem a specyfikacją (wskutek wprowadzania nowych wersji).

Wymienione cechy mają decydujące znaczenie dla wielu producentów oprogramowania. W ten sposób można sprawnie realizować projekty niezależnie od świata zewnętrznego. Okazuje się jednak, że opisany model ma także pewne wady:

- Cały kod kontenera należy opracować od podstaw. Skoro baza kodu ma zastrzeżony, niestandardowy charakter, przeprowadzenie każdej aktualizacji i wprowadzenie nowej funkcji wymaga czasu i sporych nakładów pracy.

- Firma musi sama zapewniać wsparcie dla programistów, którzy będą budowali aplikacje ponad tą platformą. Grupa specjalistów od integracji z tak przygotowaną platformą z natury rzeczy nie obejmuje innych firm i programistów, ponieważ nikt inny nie miał okazji implementować rozwiązań według tej samej specyfikacji.

Facebook to przykład kontenera w dużej mierze zaimplementowanego na bazie zastrzeżonej technologii, którą zaprojektowano specjalnie z myślą o tej platformie. Przykład Facebooka pokazuje, że model zastrzeżonej implementacji może osiągnąć niebywały sukces, jednak wymaga mnóstwo pracy, sprawnej inżynierii i czasu programistów.

W ostatnich latach można zaobserwować integrację tego serwisu z pewnymi projektami open source polegającą na włączaniu tych otwartych rozwiązań do zastrzeżonego stosu Facebooka (przykładami takich projektów są standard OAuth 2.0 i protokół Open Graph Protocol — oba zostaną omówione w dalszej części tej książki).

Implementacja typu open source

Niewielkie firmy programistyczne (lub wręcz indywidualni programiści), które chcą skorzystać z osiągnięć licznej społeczności inżynierów i wiedzy dotyczącej kontenerów społecznościowych i wytwarzania aplikacji, mogą dużo zyskać dzięki modelowi otwartych standardów. Możliwość korzystania z doświadczeń i wniosków najtęższych umysłów w tej branży znacznie ułatwia tworzenie rozbudowanych narzędzi i specyfikacji dla dowolnego kontenera społecznościowego lub aplikacji społecznościowej.

Taki model ma wiele zalet:

- Specyfikacje i narzędzia budowane przez społeczności open source zwykle są dziełem wielu różnych programistów, którzy postrzegają to oprogramowanie z odmiennych perspektyw. Ten model wprost doskonale sprawdza się podczas tworzenia rozbudowanych rozwiązań dla wielu standardowych problemów, co bez istniejących specyfikacji wymagałoby mnóstwa czasu programistów (budujących własne, zastrzeżone rozwiązania).
- Otwarte specyfikacje są stale rozwijane. Jeśli firma budująca aplikację społecznościową nie jest aktywnym uczestnikiem prac nad tymi specyfikacjami, aktualizacje i nowe funkcje są wprowadzane niezależnie od tej firmy i jej produktów. Oznacza to, że firma nie musi poświęcać swoich zasobów inżynierskich na uzupełnianie produktu o nowe funkcje. Po wydaniu nowej wersji specyfikacji zespoły implementujące dany produkt muszą tylko przeanalizować swoje narzędzia pod kątem zgodności z wymaganiami najnowszej specyfikacji. Mimo że opisany model wymaga poświęcania części czasu programistów na analizę specyfikacji, wszystkie problemy dotyczące zabezpieczeń, nowych funkcji i aktualizacji są identyfikowane i rozwiązywane przez twórców specyfikacji.
- Społeczność oferująca wsparcie dotyczące oprogramowania open source zwykle jest dość liczna. Co więcej, udostępniana dokumentacja najczęściej jest na tyle rozbudowana, że można w niej znaleźć wiele przykładów i przypadków użycia.

Jak widać, największą zaletą projektów open source jest ustawiczne zaangażowanie społeczności twórców w rozwój otwartych specyfikacji. Warto jednak pamiętać, że podobnie jak w przypadku implementacji zastrzeżonych standardy open source mają kilka istotnych wad:

- Rozwiązania tego typu nie są budowane specjalnie z myślą o konkretnym kontenerze. Mimo że niektóre specyfikacje (na przykład OpenSocial) definiują metody na potrzeby inte-

gracji z wybranymi elementami standardów (potrzebnymi w konkretnej implementacji), nawet te elementy obejmują wiele przypadków użycia, które nie są potrzebne w niestandardowych kontenerach czy aplikacjach.

- Aktualizacje specyfikacji zwykle są wynikiem procedur głosowania w ramach społeczności, gdzie każdy ma jeden głos i może wskazać, które kierunki rozwoju wydają mu się najwłaściwsze. Ta forma wyboru rozwiązań w pewnych przypadkach jest korzystna, ale oznacza też, że nie wszystkie rozwiązania oczekiwane przez tę czy inną firmę trafią do podstawowej specyfikacji.

Mimo tych wszystkich wad wiele kontenerów buduje się właśnie na bazie projektów open source. Do firm stosujących ten model należą tacy potentaci jak Yahoo!, Google, Hi5 czy LinkedIn, a także producenci rozwiązań korporacyjnych, w tym między innymi Jive, IBM i Atlassian.

Dlaczego w tej książce zostaną omówione otwarte standardy?

Kiedy przystępowałem do omawiania szczegółowych rozwiązań, musiałem wybrać jeden z dwóch modeli — albo projekty open source, albo pojedynczą, zastrzeżoną implementację kontenera. Zdecydowałem się omówić model na bazie otwartych standardów kontenerów aplikacji społecznościowych i metod wytwarzania, ponieważ ta książka nie ma dotyczyć jednego kontenera. Nie chciałem ograniczać zakresu tematycznego tego tekstu do jednej zastrzeżonej platformy, która w jednej chwili może zostać gruntownie przebudowana przez swoich twórców. Co więcej, taka formuła książki zbyt wąsko prezentowałaby zagadnienia związane z serwisami społecznościowymi.

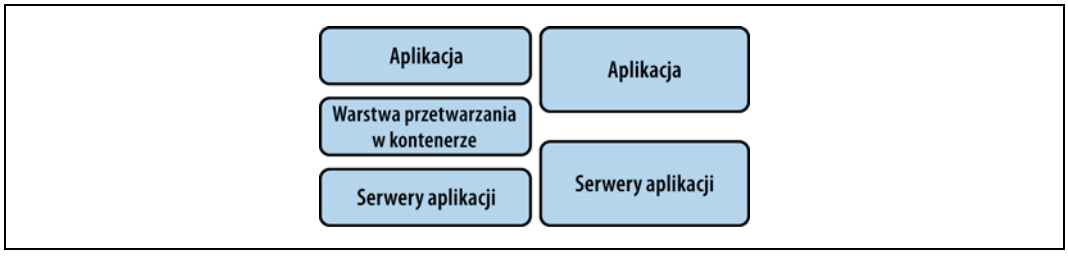
Moim głównym celem jest zaprezentowanie procesów tworzenia i stosowania kontenerów aplikacji społecznościowych oraz wyjaśnienie, jak należy budować aplikacje w ramach tych kontenerów. Rozwiązania dostępne w projektach open source najlepiej ilustrują aktualny stan serwisów społecznościowych, niezależnie od tego, czy istnieje choć jeden kontener implementujący wszystkie funkcje oferowane przez jeden projekt open source.

Wbudowana aplikacja — tworzenie rozwiązań w ramach czarnej skrzynki

Jednym z najważniejszych aspektów tworzenia aplikacji dla kontenera aplikacji społecznościowych jest brak tradycyjnego środowiska działania tych aplikacji, gdzie wystarczy zapewnić prawidłowe ładowanie aplikacji i niezawodne działanie serwera. W tradycyjnych środowiskach programiści muszą uwzględniać (i — w razie konieczności — modyfikować) tylko wymienione zmienne.

Na rysunku 1.4 pokazano różnice dzielące tradycyjne środowisko wytwarzania aplikacji (po prawej stronie) i środowisko tzw. „czarnej skrzynki", czyli kontenera aplikacji społecznościowych (po lewej stronie). Przedstawione zestawienie jest bardzo ogólne, a każda warstwa może zawierać wiele mechanizmów przetwarzania, filtrowania i udostępniania danych.

Różnicą jest środkowa warstwa środowiska kontenera aplikacji społecznościowych. Budowa aplikacji społecznościowych, które mają działać w ramach kontenera, polega na tworzeniu oprogramowania ponad infrastrukturą definiowaną przez ten kontener. Zamiast bezpośredniego



Rysunek 1.4. Ładowanie aplikacji w kontenerze (po lewej stronie) i w tradycyjnym środowisku serwera (po prawej stronie)

udostępniania kodu i funkcji przez serwery aplikacji wspomniana infrastruktura udostępnia treść, która jest następnie przetwarzana przez kontener. Kontener może następnie filtrować tę treść w celu wyeliminowania złośliwego kodu, nieobsługiwanych funkcji i innych elementów, tak aby właściwa aplikacja otrzymała gotową, sterylną treść.

Oznacza to, że programista aplikacji korzysta z gotowego kodu opracowanego przez innych programistów, zatem każda zmiana w tych procesach będzie miała bezpośredni wpływ na przebieg komunikacji pomiędzy serwerem a kontenerem lub na sposób przetwarzania danych dla samej aplikacji. Poniżej wymieniono zmiany na poziomie kontenera, które w największym stopniu wpływają na aplikację:

Aktualizacje kontenera
: Aktualizacje kontenera są prawdziwą zmorą programistów pracujących w środowiskach czarnych skrzynek. Takie aktualizacje mogą ujawniać nowe błędy lub powodować problemy związane ze zgodnością wstecz.

Czas pracy kontenera
: Jeśli nie działa kontener, nie działa także aplikacja w ramach tego kontenera.

Zmiany obsługiwanych funkcji
: Kontenery mogą zmieniać zakres obsługiwanych funkcji, co może wpływać na działanie aplikacji. Kiedy na przykład z Twittera usunięto w 2010 roku obsługę prostej autoryzacji (przy użyciu nazwy i hasła użytkownika), aplikacje zbudowane na bazie tej platformy wymagały implementacji obsługi modelu uwierzytelniania Open Authorization (OAuth).

Uszkodzone funkcje
: Obsługa niektórych funkcji kontenera używanych przez aplikację (na przykład niestandardowych znaczników, punktów końcowych REST itp.) może być czasowo niedostępna w związku z aktualizacją.

Podczas tworzenia aplikacji dla tego rodzaju środowisk można podejmować wiele działań, aby nie dać się zaskoczyć zmianom kontenera i wymienionym powyżej zdarzeniom:

Należy analizować blogi, listy dyskusyjne, wpisy na Twitterze i inne kanały komunikacji poświęcone kontenerowi
: Kiedy twórcy kontenera aktualizują swoją platformę, zwykle ogłaszają nowe wydanie w formie odpowiedniego wpisu na blogu ze szczegółowym opisem wprowadzonych zmian. Jeśli więc programista śledzi odpowiednie źródła informacji, może analizować zmiany i dostosowywać swoje oprogramowanie już w momencie wdrażania nowego wydania (lub nawet z pewnym wyprzedzeniem, jeśli twórcy kontenera wcześniej informują o plano-

wanych zmianach). I wreszcie wielu producentów kontenerów używa Twittera do informowania o przestojach, aktualizacjach lub wykrytych błędach. Twitter umożliwia śledzenie istotnych zdarzeń z dokładnością niemal co do minuty i jako taki powinien być pierwszym źródłem wiedzy o problemach dotyczących platformy.

Należy opanować strukturę raportowania o błędach kontenera
Twórcy niektórych kontenerów używają otwartych systemów śledzenia błędów; inni umożliwiają zgłaszanie i śledzenie błędów w systemach za pośrednictwem odpowiednich forów. Jeśli standardowe kanały komunikacji nie zapewniają niezbędnych informacji, należy korzystać właśnie z narzędzi do raportowania o błędach.

Należy budować narzędzia testujące poszczególne funkcje
Wiele kontenerów nie oferuje dostępnych z zewnątrz zautomatyzowanych pakietów testów, które umożliwiałyby łatwe sprawdzanie, czy wszystkie funkcje działają prawidłowo. Budowanie aplikacji zgodnie z zasadami wytwarzania sterowanego testami pozwala stworzyć wyczerpujący pakiet tzw. testów przekrojowych (ang. *end-to-end test*) niezbędnych do wygodnego weryfikowania dostępności funkcji i błyskawicznego wykrywania zmian w platformie.

Mimo że wytwarzanie oprogramowania w modelu czarnej skrzynki bywa trudniejsze niż w przypadku tradycyjnych metod budowy aplikacji, warto też wspomnieć o zaletach tego trybu pracy. W tradycyjnych środowiskach wytwarzania należałoby tworzyć lub integrować wszystkie elementy niezbędne do uruchamiania społecznościowych funkcji na własnych serwerach. W środowisku czarnej skrzynki aplikacja jest budowana ponad stale aktualizowanym i rozwijanym kontenerem. To kontener odpowiada za doskonalenie oferowanych funkcji oraz za testy jakościowe i przekrojowe. Oznacza to, że programiści aplikacji nie muszą tracić czasu na rozwój samej platformy i mogą się koncentrować wyłącznie na poszczególnych funkcjach swojej aplikacji.

Wbudowane zabezpieczenia aplikacji

Aplikacje społecznościowe działające ponad swoimi kontenerami stanowią poważne zagrożenie dla bezpieczeństwa tych kontenerów. Skoro kontenery mają pełnić funkcje hostów dla tych aplikacji, muszą akceptować wykonywanie zewnętrznego kodu w ramach swoich stron. Warto więc odpowiedzieć sobie na pytanie, jak obsługiwać te dodatkowe aplikacje bez ryzyka osłabienia bezpieczeństwa użytkowników kontenera aplikacji społecznościowych.

W przeszłości podejmowano wiele prób ograniczania tego ryzyka. Niektóre kontenery zachęcają programistów do budowania aplikacji przy użyciu odpowiednio zabezpieczonego podzbioru elementów HTML-a i JavaScriptu — takie rozwiązanie gwarantuje kontenerom, że wykonywany kod nie zawiera potencjalnie groźnych elementów. Alternatywnym wyjściem jest stosowanie mechanizmów filtrowania i modyfikowania kodu (na przykład Caja lub ADsafe), które modyfikują kod aplikacji, pozostawiając tylko bezpieczny podzbiór funkcji i eliminując wszystkie znaczniki i elementy stanowiące potencjalne zagrożenie. Odpowiednie technologie zostaną szczegółowo omówione w punkcie „Zabezpieczanie aplikacji" w dalszej części tego podrozdziału oraz w rozdziale 8. (poświęconym metodom wytwarzania bezpiecznych aplikacji).

Mimo dużej liczby metod zabezpieczania aplikacji znaczniki `iframe` wciąż zachowują status najbardziej popularnego rozwiązania (znaczniki `iframe` są stosowane w zdecydowanej większości

kontenerów). Zalety używania tych znaczników są dość oczywiste — implementacja obsługi tej metody w kodzie kontenera jest dość prosta, a same znaczniki zapewniają programistom aplikacji maksimum swobody przy minimalnych ograniczeniach.

Okazuje się jednak, że swoboda programistów budujących treść aplikacji wewnątrz ramek `iframe` jest też największą wadą tej metody. Ta swoboda może być sporym ułatwieniem dla złośliwych programistów, którzy mogą wykorzystać doskonale znane luki tego rozwiązania (opisane w poniższych punktach).

Ataki XSS

Ataki **XSS** (od ang. *cross-site scripting*) stanowią podstawowe zagrożenie dla bezpieczeństwa aplikacji internetowych (szczególnie tych uwięzionych wewnątrz kontenerów). Ataki XSS należą do najbardziej popularnych sposobów wykorzystywania luk w zabezpieczeniach aplikacji internetowych. Atakujący stosuje tę metodę do wstrzykiwania skryptów strony klienckiej do kodu stron przeglądanych następnie przez pozostałych użytkowników. Skrypty umieszczone w ten sposób na stronie mogą zostać użyte do obejścia mechanizmów kontroli dostępu (na przykład z wykorzystaniem zasady tego samego pochodzenia — ang. *same-origin policy*).

Skutki korzystania z serwisu, który został skutecznie zaatakowany przy użyciu metody XSS, mogą być bardzo różne — od zwykłej irytacji do poważnego zagrożenia dla bezpieczeństwa, w tym przechwycenia przez atakującego szczegółów logowania użytkownika, danych karty kredytowej, danych osobowych i informacji o wszystkich osobistych kontaktach użytkownika za pośrednictwem serwisu społecznościowego.

Prostym przykładem ataku XSS jest implementacja w ramach aplikacji internetowej reklamy, która umożliwia zewnętrznym reklamodawcom wykonywanie własnego kodu w serwisie. Reklamy są co prawda formą „ataków" XSS za zgodą właściciela serwisu, ale w większości przypadków właściciel strony może ufać reklamodawcy.

Mimo że opisana metoda ataku dotyczy wszystkich aplikacji internetowych, w przypadku kontenerów aplikacji społecznościowych akceptujących kod zewnętrzny ochrona przed tym zagrożeniem wymaga specjalnych działań i mechanizmów kontrolnych.

Zasada tego samego pochodzenia i starsze przeglądarki

Zasada tego samego pochodzenia (ang. *same-origin policy*) jest bardzo ważnym aspektem interakcji użytkownika z serwisem internetowym lub aplikacją internetową. Bez implementacji zasady tego samego pochodzenia aplikacje ładowane wewnątrz ramki `iframe` nie mogłyby uzyskiwać dostępu nie tylko do struktury DOM macierzystego serwisu, ale też do jego plików cookie czy danych formularza.

Współczesne przeglądarki internetowe dość dobrze radzą sobie z implementacją zasady tego samego pochodzenia, eliminując opisane problemy (także w kontekście tworzenia aplikacji społecznościowych), jednak niektóre starsze przeglądarki nie gwarantują skutecznej ochrony przed naruszaniem tej zasady.

Mimo że starsze przeglądarki cieszą się coraz mniejszą popularnością wśród użytkowników, warto mieć na uwadze zagrożenia związane z ich stosowaniem.

Pobieranie plików bez wiedzy użytkownika

Ataki poprzez pobieranie plików bez wiedzy użytkownika (ang. *drive-by download*) polegają na stosowaniu procesów pobierających treść ze złośliwych serwisów na komputer nieświadomego użytkownika. Problem nie dotyczy tylko zabezpieczeń znacznika `iframe`, jednak ponieważ znacznik `iframe` umożliwia programiście wykonywanie dowolnego kodu w ramach kontenera, właśnie w tym modelu ryzyko podobnych ataków jest szczególnie wysokie.

Ataki poprzez pobieranie plików bez wiedzy użytkownika mogą naśladować funkcje wyskakujących okien. Próba zamknięcia takiego okna może spowodować przypadkowe pobranie do systemu użytkownika oprogramowania szpiegującego, złośliwego oprogramowania lub wirusów. Wyskakujące okna mogą sprawiać wrażenie komunikatów o błędach, reklam lub dowolnych innych popularnych komunikatów, które nie wzbudzają podejrzeń użytkownika. Ponieważ atak jest inicjowany przez czynność samego użytkownika, kontener traktuje pobieranie złośliwych plików jako celowe, przemyślane działanie.

Opisany sposób przeprowadzania ataków to tylko jedna z metod, której może użyć twórca złośliwego oprogramowania. Ataki poprzez pobieranie plików bez wiedzy użytkownika przybierają wiele form i stanowią poważny problem wszędzie tam, gdzie zewnętrzny kod może być swobodnie wykonywany w ramach kontenera aplikacji.

Zabezpieczanie aplikacji

Istnieje wiele sposobów zabezpieczenia kodu z zewnątrz (dostarczonego przez niezależnych producentów) wykonywanego w ramach kontenera. Dwa dostępne rozwiązania, produkty Caja i ADsafe, realizują to zadanie w szczególny sposób.

Caja jest kompilatorem języka JavaScript, który przebudowuje kod części frontowej ładowany w ramach kontenera. W czasie procesu przebudowy tego kodu kompilator Caja eliminuje potencjalnie niebezpieczne elementy, tworząc tzw. bezpieczny kod Caja. Aplikacja współpracująca z kontenerem, który stosuje kompilator Caja, ma pośredni dostęp do struktury DOM macierzystej strony kontenera. Takie rozwiązanie umożliwia skuteczne zabezpieczanie wszystkich żądań.

ADsafe nie przebudowuje kodu aplikacji, a jedynie usuwa wszystkie funkcje języka JavaScript uważane za potencjalnie niebezpieczne. Stosowanie produktu ADsafe jest mniej kłopotliwe niż używanie kompilatora Caja (gruntownie przebudowującego kod źródłowy), jednak ADsafe nie zapewnia bezpieczeństwa na poziomie oferowanym przez kompilator Caja.

Oba te rozwiązania zostaną szczegółowo omówione w rozdziale 8.

Aplikacja zewnętrzna — integracja danych serwisu społecznościowego poza kontenerem

Do tej pory koncentrowałem się na konstruowaniu aplikacji działających w ramach kontenera. Okazuje się jednak, że nie jest to jedyny kontekst funkcjonowania aplikacji społecznościowych.

Większość kontenerów oferuje dostęp do swoich danych społecznościowych (i danych kontenerów) za pośrednictwem punktów końcowych URI. Z myślą o wygodzie programistów aplikacji wspomniane punkty końcowe zwykle są opakowywane w ramach łatwiejszych w użyciu

konstrukcji, na przykład żądań OpenSocial języka JavaScript lub specjalnych znaczników kontenera. Dzięki temu aplikacje mogą bezpiecznie uzyskiwać dostęp do danych użytkowników serwisów społecznościowych (przetwarzanych w czasie wyświetlania aplikacji). Okazuje się jednak, że te same punkty końcowe umożliwiają programistom aplikacji działających poza kontenerami operowanie na danych społecznościowych tych kontenerów. Oznacza to, że aplikacje tego typu mogą oferować funkcje na bazie tych danych społecznościowych, mimo że nie są budowane ani nie działają w ramach kontenera.

Aby zapewnić bezpieczny dostęp do tej warstwy i właściwie chronić dane społecznościowe użytkowników przed atakami, wiele kontenerów stosuje implementacje mechanizmów zabezpieczeń, na przykład standard OAuth. Standard OAuth jest w większym lub mniejszym stopniu implementowany przez wiele najbardziej popularnych kontenerów aplikacji społecznościowych, w tym przez kontenery Facebook, YAP, iGoogle, Orkut i MySpace.

Korzystanie z funkcji społecznościowych kontenera ułatwia programistom wychodzenie poza granice samego kontenera i błyskawiczną budowę bogatego grafu społeczności na potrzeby swoich aplikacji i serwisów internetowych (bez konieczności długotrwałego konstruowania własnego grafu powiązań dopiero po wdrożeniu aplikacji).

Oprócz operowania na danych społecznościowych poza kontenerem programista może też stosować rozmaite dodatkowe technologie, na przykład mechanizm rejestracji użytkowników przy użyciu nazw i haseł ze struktury logowania w kontenerze. Istnieje technologia open source nazwana OpenID (*Open Identification*), która umożliwia implementację właśnie takiej struktury logowania. Brak konieczności tworzenia nowego konta w serwisie może podnieść społecznościową wartość aplikacji i ograniczyć odsetek użytkowników rezygnujących z jej stosowania w trakcie procesu rejestracji. Po zalogowaniu się użytkownika (za pomocą technologii OpenID) serwis może oferować mechanizm modyfikowania i dostosowywania profilu użytkownika.

Połączenie tych dwóch technologii (OpenID i OAuth) w ramach hybrydowego procesu autoryzacji umożliwia programiście skonstruowanie struktury logowania eliminującej problem porzucania serwisu w trakcie rejestracji (OpenID) i wykorzystującej punkty końcowe URI kontenera aplikacji społecznościowych do wstępnego określania ustawień profilu użytkownika i stosowania bogatych danych społecznościowych udostępnianych przez kontener (OAuth).

Techniki implementowania standardów OAuth i OpenID w kontekście aplikacji i serwisów zewnętrznych zostaną omówione w dalszych rozdziałach tej książki.

Widoki aplikacji

Widoki umożliwiają aplikacji interakcję z użytkownikiem w ramach kontenera aplikacji społecznościowych. Kontener może oferować co najmniej jeden widok, dla którego programista aplikacji może przygotowywać swoją treść. Właśnie widoki umożliwiają użytkownikowi przeglądanie i używanie funkcji aplikacji za pośrednictwem różnych stron w ramach kontenera.

Ogólnie wszystkie widoki należą do jednego z dwóch typów:

Mały widok
: Widoki tego typu zwykle podlegają ograniczeniom dotyczącym wielkości i zakresu oferowanych funkcji. Małe widoki zwykle są prezentowane w profilu użytkownika lub w jego spersonalizowanym widoku domowym (dostępnym tylko dla tego użytkownika). Poje-

dyncza strona może zawierać wiele małych widoków (zależnie od liczby aplikacji zainstalowanych przez użytkownika).

Duży widok
: Widok tego typu ma w założeniu całkowicie angażować użytkownika. Duży widok rzadko podlega ścisłym ograniczeniom i jako taki udostępnia te same funkcje, które programista ma do dyspozycji podczas budowy aplikacji na potrzeby zewnętrznego serwisu (ma nad nim pełną kontrolę). Duże widoki zwykle obejmują tylko jedną aplikację na stronie, zatem użytkownicy korzystający z tych widoków najczęściej są pochłonięci tylko jedną aplikacją.

W większości współczesnych kontenerów aplikacji społecznościowych do oddzielania widoków od właściwych kontenerów używa się znaczników `iframe`. Mimo że opisany model zapewnia pewną ochronę dla kontenera i jego użytkowników, z pewnością nie jest to w pełni bezpieczne środowisko. W wielu przypadkach aplikacje są ładowane dopiero po pełnym załadowaniu kontenera, tak aby zapobiec spadkowi wydajności stron w przypadku jednoczesnego stosowania wielu aplikacji.

Widok domowy (mały)

Widok domowy aplikacji zwykle ma postać małego widoku udostępniającego osobistą, ogólną treść dostosowaną do preferencji bieżącego użytkownika. Innymi słowy, widok domowy nie jest dostępny dla żadnego innego użytkownika (w tym dla znajomych i powiązań bieżącego użytkownika). W większości przypadków ten rodzaj widoku stanowi centralny punkt interakcji użytkownika z kontenerem. To w tym widoku użytkownik zapoznaje się z powiadomieniami o czynnościach powiązanych użytkowników, zdarzeniach, zdjęciach itp. Widok domowy może zawierać wiele aplikacji (zależnie od kontenera pełniącego funkcję hosta dla tego widoku). Miejsce tego widoku w ramach kontenera pokazano na rysunku 1.5.

Rysunek 1.5. Widok domowy aplikacji

Widok domowy zwykle oferuje niewielkie okno (w formie małego widoku) dla pełnej aplikacji. Wiele kontenerów narzuca pewne ograniczenia dla tego widoku, na przykład dotyczące zbioru znaczników języka HTML, elementów arkuszy stylów CSS i bezpiecznych znaczników definiowanych przez sam kontener (niezbędnych do uzyskiwania dostępu do informacji społecznościowych, jak listy rozwijane zaproszeń czy dane użytkowników). Wiele kontenerów ściśle określa zasady stosowania skryptów języka JavaScript i technologii Flash przede wszystkim z myślą o wydajności i bezpieczeństwie.

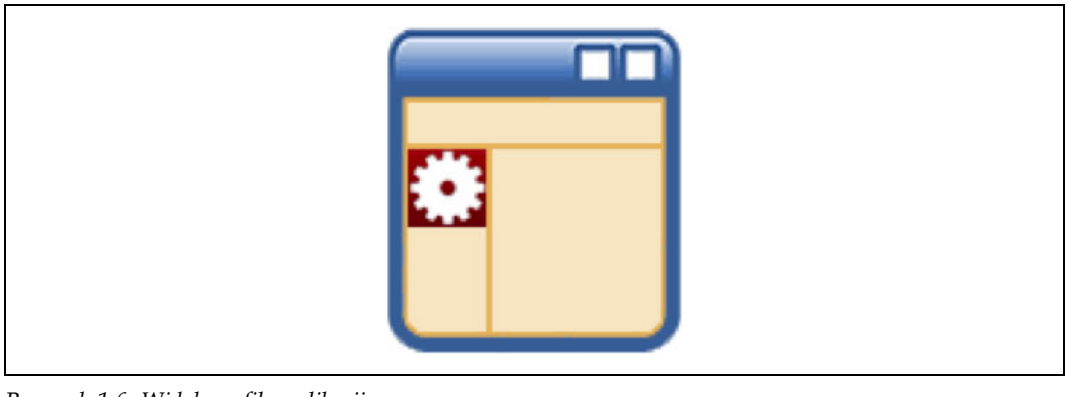
Ponieważ ten widok często jest pierwszym punktem interakcji użytkownika z aplikacją w ramach kontenera, bardzo ważne jest prezentowanie możliwie wielu funkcji, aby zachęcić użytkownika do przejścia do dalszych, bardziej szczegółowych widoków aplikacji. Wspomniane ograniczenia małego widoku skłaniają wielu programistów do niemal całkowitego ignorowania tej formy prezentacji i poświęcania całej uwagi pełnowartościowemu dużemu widokowi. W zdecydowanej większości przypadków takie nastawienie programistów powoduje, że małe widoki ograniczają się do zbioru zachęt do działania (ang. *call to action*), czyli w praktyce do przejścia do właściwych dużych widoków. Tak uproszczone widoki nie stanowią jednak odpowiednio skutecznych bodźców.

Każdy widok, który może poprawić pierwsze wrażenie użytkownika aplikacji, zasługuje na nie mniej uwagi niż właściwe funkcje tej aplikacji. Mógłbym powtarzać to bez końca — odpowiednio zaprojektowany mały widok może skutecznie podnieść liczbę aktywnych użytkowników codziennie korzystających z aplikacji, spotęgować ich zaangażowanie i wreszcie podnieść łączną liczbę użytkowników, zwiększając potencjał biznesowy całej aplikacji.

Mały widok aplikacji powinien prezentować możliwie atrakcyjną, przyciągającą uwagę treść (na przykład nowe możliwości, które użytkownicy zyskają dzięki aplikacji). Mały widok nigdy nie powinien być bezpośrednią kopią właściwego pełnego widoku i powinien udostępniać podstawowe funkcje, których stosowanie nie wymaga pełnego zaangażowania, a jednocześnie przynosi pewne korzyści użytkownikom.

Widok profilu (mały)

Widok profilu jest dostępny dla wszystkich i obejmuje dane upublicznione przez użytkownika. Dostęp do tego widoku ma każdy, komu pozwoli na to właściciel profilu. W niektórych kontenerach widok użytkownika może obejmować aplikacje dodane przez użytkownika — zainstalowane aplikacje są wówczas widoczne dla wszystkich pozostałych użytkowników przeglądających ten profil. W tym kontekście szczególnie popularne są aplikacje wyświetlające wysłane i otrzymane podarunki, prezentujące informacje o użytkownikach lub aplikacje notatek z funkcją sprawdzania dostępności — wszystkie te rozwiązania umożliwiają interakcję zewnętrznych użytkowników (na przykład znajomych) z aplikacjami zainstalowanymi przez właściciela profilu. Miejsce tego widoku w ramach kontenera pokazano na rysunku 1.6.



Rysunek 1.6. Widok profilu aplikacji

Ten widok jest stosowany w kontenerach rzadziej niż standardowy mały widok (widok domowy) i duży widok (tzw. widok kanwy), ale jeśli już jest dostępny, ma spory potencjał i może być źródłem wielu korzyści. W przeciwieństwie do widoku domowego bezpośrednim odbiorcą tego małego widoku jest grupa wszystkich użytkowników przeglądających profil danego użytkownika. Odbiorcy będą widzieli treść ładowaną przez aplikację i zachęcającą w ten czy inny sposób do jej użycia. Ponieważ w ten sposób można trafić do odbiorców powiązanych z bieżącym użytkownikiem (właścicielem profilu), opisywany widok jest wprost doskonałą okazją do nakłaniania nowych użytkowników do zainstalowania aplikacji. Warto wykorzystać tę okazję do oferowania nowych metod nawiązywania kontaktu z oryginalnym użytkownikiem (poza standardowymi sposobami dostępnymi w danym kontenerze aplikacji społecznościowych). Udostępniane metody mogą na przykład wiązać się z upominkami dla pary kontaktujących się użytkowników, z pomocą w grach, w których uczestniczą ci użytkownicy, lub prezentacją położenia obu użytkowników na mapie. Kiedy nowy użytkownik korzysta z tak udostępnianej funkcji aplikacji, jest proszony o zainstalowanie tej aplikacji we własnym profilu — w ten sposób można pozyskać nowego użytkownika.

Ponieważ widok profilu należy do kategorii małych widoków z możliwością interakcji ze światem zewnętrznym, zwykle podlega tym samym ograniczeniom co widok domowy. Sprawne posługiwanie się wszystkimi dostępnymi widokami w celu zapewnienia użytkownikowi możliwie bogatych doznań (umożliwienia interakcji na poziomie dowolnego widoku, w którym zdecyduje się skorzystać z oferty aplikacji) jest kluczem do sukcesu. Użytkownicy nigdy nie korzystają z aplikacji dokładnie tak, jak założyli twórcy oprogramowania, zatem warto z wyprzedzeniem planować sposoby oferowania przydatnych funkcji na możliwie wielu płaszczyznach (najlepiej we wszystkich widokach).

Widok kanwy (duży)

Widok kanwy, czyli duży widok, jest w istocie pełnowartościowym widokiem aplikacji. Większość kontenerów nie narzuca żadnych ograniczeń w zakresie stosowania kodu JavaScriptu ani animacji Flash w ramach tego widoku, zatem widok kanwy stwarza doskonałą okazję do prezentowania funkcji, treści i narzędzi użytkownikom końcowym. Ten widok jest sercem aplikacji i jako taki zapewnia dostęp do najważniejszych funkcji oferowanych przez tę aplikację.

W przeciwieństwie do małego widoku aplikacji widok kanwy nie jest wyświetlany wraz z pozostałymi aplikacjami; widok kanwy zajmuje większość widoku kontenera aplikacji społecznościowych. Oznacza to, że widok kanwy oferuje twórcy aplikacji mnóstwo przestrzeni złożonej z pikseli w pionie i poziomie (patrz rysunek 1.7). Z perspektywy programisty aplikacji użytkownicy, którzy korzystają z tego widoku, są już odpowiednio zaangażowani (albo poprzez bezpośrednie odwiedzenie odpowiedniej strony, albo w wyniku skutecznych bodźców i zachęt do działania zawartych na którymś z małych widoków). W zdecydowanej większości przypadków użytkownik, który trafia na ten widok, jest już mocno zainteresowany odpowiednią aplikacją.

Warto przy tej okazji podkreślić, że widok kanwy jest najlepszą formą interakcji użytkownika i aplikacji. Nadmierne komplikowanie doznań, których dostarcza on użytkownikowi, nadużywanie informacji społecznościowych (na przykład poprzez prezentowanie zbyt wielu komunikatów w strumieniu aktywności) lub brak dostatecznie widocznych, wygodnych funkcji niemal na pewno zniechęcą użytkownika do dalszego korzystania z aplikacji.

Rysunek 1.7. Widok kanwy aplikacji

Najlepsze aplikacje zapewniają w tym widoku nie tylko bogate, odpowiednio zaprojektowane atrakcje, ale też oferują przemyślane połączenia pomiędzy dużym widokiem a mniejszymi widokami. Jeśli na przykład użytkownik skorzystał z małego widoku aplikacji, wpisał jakieś informacje na formularzu i wysłał ten formularz, aby przejść do widoku kanwy, aplikacja powinna zareagować prezentacją stanu oczekiwanego przez tego użytkownika, na przykład informacji o wyniku przetwarzania danych formularza.

Innym bardzo ważnym składnikiem widoku kanwy są elementy potwierdzające, że użytkownik wciąż korzysta z kontenera społecznościowego, czyli na przykład możliwość uzyskania dostępu do profilu użytkownika lub listy znajomych. Jeśli aplikacja wymaga do działania informacji o użytkowniku, na przykład o jego zainteresowaniach, nazwisku czy identyfikatorze, należy uzyskać te informacje z danych podanych przez samego użytkownika w profilu. Żądanie ponownego wpisywania informacji na potrzeby rejestracji w usłudze bez choćby wstępnego wypełnienia pól formularza na podstawie dostępnych danych jest najkrótszą drogą do utraty użytkownika.

Domyślny widok (dowolny)

Ostatni widok, który jest obsługiwany przez wiele kontenerów aplikacji społecznościowych, w rzeczywistości nie jest widokiem — pełni raczej funkcję stanu widoku. **Widok domyślny** (tzw. **podgląd**) aplikacji wyświetla treść kierowaną do użytkowników, którzy jeszcze nie zainstalowali danej aplikacji lub którzy nie są zalogowani w danym serwisie społecznościowym. Ten widok należy do przestrzeni publicznej i jako taki może nie mieć bezpośredniego dostępu do informacji o profilu użytkownika (podobnie jak publicznie dostępna wersja profilu użytkownika zawiera tylko niewielki podzbiór informacji na temat właściciela profilu).

Ten specyficzny stan widoku może występować w dowolnym innym tradycyjnym widoku, w tym w widoku profilu i widoku kanwy. Ponieważ jednak widok domowy zwykle prezentuje osobisty profil zalogowanego użytkownika, w tym przypadku występowanie domyślnego stanu zdarza się dość rzadko. W przypadku wylogowanego użytkownika korzystanie z osobistych informacji jest niemożliwe, ponieważ niezbędne dane są po prostu nieznane. Występowanie tego stanu w małym widoku może wynikać z próby wyświetlenia profilu użytkownika przez innego użytkownika; mimo że aplikacja zostanie wyświetlona, samo zdarzenie

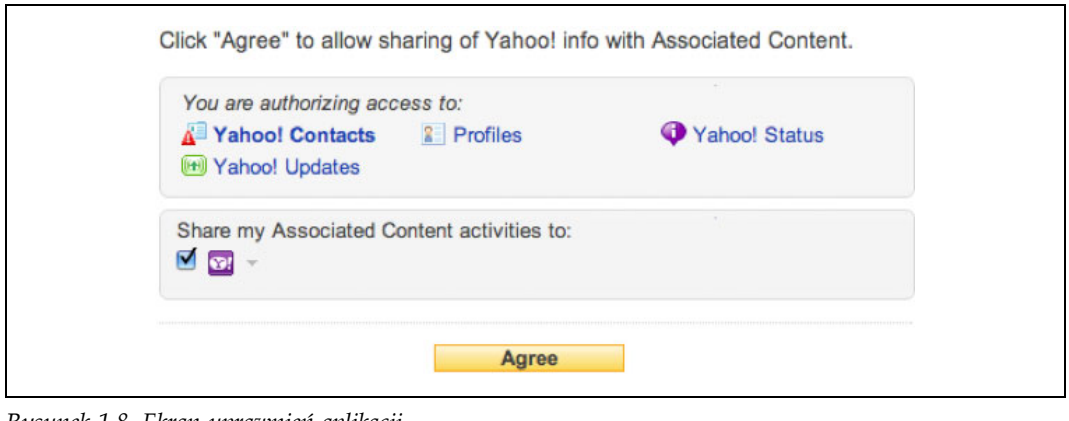
otwarcia tego widoku nie musi oznaczać pozyskania zainteresowanego użytkownika. Jeśli jednak opisany stan występuje w widoku kanwy aplikacji, prawdopodobieństwo pozyskania nowego użytkownika zainteresowanego dodatkowymi szczegółami jest całkiem wysokie.

Z myślą o występowaniu tego stanu w mniejszych widokach aplikacji warto opracować metodę możliwie szybkiego kierowania odwiedzających do widoku kanwy aplikacji, tak aby przekonać ich do zainstalowania aplikacji (lub metodę stopniowego zwiększania ich zaangażowania, by chwilę później zachęcić użytkowników do przejścia do widoku kanwy). Ponieważ użytkownik nigdy nie korzystał z aplikacji i nie podał danych osobowych, które można by wykorzystać w widoku aplikacji, właściwa personalizacja tego stanu jest utrudniona. Budowanie zaangażowania jest bardzo ważne także w opisanych przypadkach, jednak tym razem zasadniczym celem twórcy aplikacji jest zachęcenie użytkownika do przejścia z domyślnego stanu do stanu instalacji.

Wystąpienie tego stanu w większym widoku kanwy najprawdopodobniej oznacza, że użytkownik szuka informacji o aplikacji przed podjęciem decyzji o jej ewentualnej instalacji. W takim przypadku warto spróbować nakłonić użytkownika do interakcji z aplikacją (w formie, w jakiej to będzie możliwe) i jednocześnie zachęcać go do zalogowania się w serwisie i przeprowadzenia właściwej instalacji. Powiązanie stanu instalacji kontenera z funkcjami aplikacji jest dobrym sposobem płynnego pozyskiwania nowych użytkowników. Jeśli użytkownik decyduje się na interakcję z kontrolkami i preferencjami aplikacji, można spróbować skierować go do stanu instalacji kontenera i przeprowadzić przez ten proces, tak aby mógł przystąpić do właściwego korzystania z aplikacji.

Zagadnienia związane z uprawnieniami aplikacji

Aplikacje działające w ramach kontenera aplikacji społecznościowych domyślnie nie mają dostępu do profili użytkowników, strumieni aktywności czy danych o znajomych — taki dostęp stanowiłby poważne zagrożenie dla bezpieczeństwa macierzystego serwisu. Kontenery zwykle nakładają na aplikacje obowiązek definiowania **zakresu uprawnień** wymaganych do prawidłowego funkcjonowania. Zakres uprawnień obejmuje wszystkie żądania, które będą wymagały akceptacji użytkownika instalującego aplikację (patrz rysunek 1.8).



Rysunek 1.8. Ekran uprawnień aplikacji

Uprawnienia dostępu do danych społecznościowych zwykle dotyczą podstawowych informacji na temat użytkownika, na przykład:

- profilu,
- strumienia aktywności,
- znajomych.

Każdy kontener może definiować inne uprawnienia lub nawet obsługiwać wiele uprawnień niezwiązanych z danymi społecznościowymi udostępnianymi w ramach tego kontenera. Po zaakceptowaniu zakresu uprawnień przez użytkownika kontener może podejmować odpowiednie działania w jego imieniu. W przypadku wymienionych powyżej uprawnień dostępu do danych społecznościowych odpowiednie działania mogą obejmować:

- Pobieranie przez aplikację wszystkich informacji przechowywanych w ramach profilu użytkownika.
- Śledzenie przez aplikację wszystkich czynności składających się na strumień aktywności i umieszczanie w tym strumieniu nowych czynności w imieniu użytkownika.
- Uzyskiwanie przez aplikację danych zawartych w profilach wszystkich znajomych danego użytkownika. Mimo że znajomi użytkownika sami nie musieli dodać tej aplikacji, każdy użytkownik deklarujący przyjaźń czy znajomość z innym użytkownikiem w rzeczywistości potwierdza pewną relację zaufania, a instalowana aplikacja tylko wykorzystuje tę deklarację. W szczególności użytkownik powierza zaufanemu znajomemu swoje dane osobowe. Skoro znajomy użytkownika deklaruje zaufanie do jakiejś aplikacji i godzi się na powierzenie jej własnych danych osobowych, istnieje swoisty most zaufania łączący tę aplikację ze znajomymi użytkownika. Niektóre kontenery ograniczają zakres informacji społecznościowych udostępnianych na podstawie tak deklarowanych relacji zaufania.

Wymienione uprawnienia często są zabezpieczane przy użyciu mechanizmów autoryzacji, na przykład technologii OAuth, które uzależniają możliwość nadania uprawnień instalowanej aplikacji od uprzedniego logowania samego użytkownika.

Aplikacje strony klienckiej kontra aplikacje serwera

Już na początku prac nad aplikacją dla serwisu społecznościowego każdy programista zadaje sobie pytania, jak najlepiej zapisywać treść budowanej aplikacji. Czy należy użyć mechanizmów zapisywania treści po stronie klienta, korzystając z licznych narzędzi klienckich udostępnianych przez większość kontenerów? A może należy zapisywać dane po stronie serwera, aby zagwarantować możliwie wysoką skalowalność i wydajność? Innymi słowy, które narzędzia najlepiej sprawdzają się podczas realizacji tego zadania? Na niektóre z tych pytań należy odpowiedzieć jeszcze przed przystąpieniem do właściwej budowy aplikacji.

Zanim programista zacznie tworzyć aplikację, powinien przeanalizować kilka czynników decydujących o skalowalności i ogólnej wydajności budowanego oprogramowania.

Stosowanie systemów szablonów w warstwie znaczników

Mimo że zintegrowanie aplikacji z niektórymi systemami szablonów wymaga dodatkowego czasu programistów, stosowanie tych systemów znacznie podnosi skalowalność środowiska aplikacji. Krótko mówiąc, szablony umożliwiają programistom oddzielanie logiki aplikacji od

jej wizualnej warstwy. Systemy szablonów upraszczają tworzenie aplikacji i usprawniają współpracę w ramach zespołu programistów, ułatwiają diagnozowanie środowiska aplikacji oraz umożliwiają skalowanie produktu (w razie potrzeby).

Nie ma wątpliwości, że niektóre systemy szablonów wprowadzają do kodu aplikacji mnóstwo niepotrzebnych elementów — w tej sytuacji warto odpowiedzieć sobie na pytanie, który z tych systemów rzeczywiście okaże się przydatny podczas budowy określonej aplikacji. W przypadku niewielkich aplikacji stosowanie pełnowartościowych, rozbudowanych systemów szablonów nie przyniesie spodziewanych korzyści, ponieważ rozwiązania tego typu planuje się z myślą o projektach wielkiej skali; podobnie w przypadku wielkich aplikacji stworzenie niestandardowego systemu szablonów wymagałoby sporych nakładów.

Ponownie programiści powinni zadać sobie pytanie dotyczące przede wszystkim potrzeb tworzonego produktu. Przy okazji omawiania w tej książce technik stosowania różnych standardów open source podczas tworzenia aplikacji zostaną opisane między innymi systemy szablonów oferowane za pośrednictwem frameworku aplikacji OpenSocial.

Stosowanie mieszanego środowiska serwera i klienta

Jak już wspomniałem, na początku projektu polegającego na budowie aplikacji programiści zwykle muszą odpowiedzieć sobie na pytania dotyczące właściwych metod konstruowania logiki tej aplikacji. Programiści zazwyczaj wybierają rozwiązania wygodne dla nich samych, nierzadko ignorując inne, równie korzystne opcje tylko dlatego, że nie są przyzwyczajeni do alternatywnych rozwiązań.

Czy podczas tworzenia aplikacji lepiej wybrać implementację obciążającą przede wszystkim stronę klienta, czy system działający głównie po stronie serwera? Odpowiedź jest krótka — oba modele mają pewne zalety w wymiarze wydajności, funkcjonalności i prostoty samego wytwarzania aplikacji.

Większość kontenerów oferujących możliwość wytwarzania aplikacji ponad swoimi platformami udostępnia zbiory bezpiecznych znaczników, dzięki którym programista może łatwo wizualizować dane społecznościowe użytkowników bez konieczności wywoływania serwera. Taka możliwość znacznie ułatwia tworzenie części frontowej aplikacji, ponieważ to kontener odpowiada za przetwarzanie danych społecznościowych. Co więcej, przetwarzanie danych w tym trybie nie wymaga zgody użytkownika na uzyskiwanie dostępu do danych osobowych przez aplikację. Bezpieczne znaczniki to tylko jedno z wielu ułatwień dostępnych dla programistów pracujących nad systemem frontowym — zarządzanie przetwarzaniem danych społecznościowych przez sam kontener dodatkowo umożliwia programistom konstruowanie wysoce społecznościowych aplikacji.

Także alternatywne rozwiązanie, czyli model budowy aplikacji działającej po stronie serwera, ma dość oczywiste zalety z perspektywy programistów. Po pierwsze, przetwarzanie po stronie serwera nie jest uzależnione od zgodności poszczególnych przeglądarek (na przykład od obsługi języka JavaScript). Środowisko serwera pozostaje spójne niezależnie od przeglądarki, w której użytkownik wyświetla daną aplikację. Na działanie aplikacji nie mają też wpływu ustawienia przeglądarki, na przykład wyłączona obsługa JavaScriptu. Co więcej, przetwarzanie danych po stronie serwera umożliwia programiście buforowanie wyników w pamięci podręcznej i — tym samym — skrócenie czasu odpowiedzi na żądania danych.

Kiedy w takim razie należy stosować implementację po stronie serwera, a kiedy implementację po stronie klienta? Przetwarzanie danych jest bardziej efektywne po stronie serwera, a mechanizmy buforowania odpowiedzi dodatkowo skracają czas obsługi żądań danych. Implementacja po stronie klienta umożliwia stosowanie niestandardowych znaczników i uzyskiwanie dostępu do pozostałych danych społecznościowych kontenera, ponieważ sam kontener może efektywnie analizować i wyświetlać osobiste dane użytkownika bez konieczności wysyłania dodatkowych żądań przez aplikację. Najlepszym wyjściem jest więc zastosowanie mieszanego środowiska klienta-serwera, które stwarza największe możliwości korzystania z narzędzi kontenera.

Opóźnianie ładowania mniej ważnej treści

Jeśli kontener aplikacji społecznościowych stosuje restrykcyjne zabezpieczenia w związku z ładowaniem treści, aplikacje operujące na dużej ilości tej treści zwykle są ładowane wolniej i są narażone na przekraczanie limitów czasowych (jeżeli kontener wymusza ładowanie całej treści aplikacji w określonym czasie). Problem jest jeszcze poważniejszy w przypadku aplikacji ładującej całą treść na podstawie jednego żądania kierowanego na serwer, co zwykle wymaga długotrwałego przetwarzania przed zwróceniem kompletnego zbioru znaczników. Przetwarzanie wszystkich danych aplikacji w ramach jednego żądania pozwala jednak ograniczyć liczbę żądań protokołu HTTP generowanych przez aplikację i — tym samym — nieznacznie podnieść wydajność. W przypadku wielu stosunkowo niewielkich aplikacji opisany model jest więc uzasadniony. Przy aplikacjach operujących na dużej ilości treści takie rozwiązanie może przynieść więcej szkód niż pożytku, ponieważ prędzej czy później aplikacja osiągnie punkt, w którym nie będzie mogła załadować żadnych danych w ramach kontenera.

Skutki tego zjawiska można stosunkowo łatwo ograniczyć, stosując mechanizm opóźnionego ładowania danych. Ponieważ większość kontenerów ładuje aplikację na stronie dopiero po zakończeniu ładowania treści samej strony, aplikacja zwykle ma niewiele cennego czasu na przyciągnięcie uwagi użytkownika. Aplikacja ładowana przez osiem sekund (już po wyświetleniu znaczników właściwej strony) zwykle traci szansę na dotarcie do użytkownika, który albo przewija stronę, albo przystępuje do innych czynności w ramach swojego profilu.

Jeśli najważniejsza treść aplikacji jest ładowana w ramach pierwszego żądania protokołu HTTP (na przykład gdy to żądanie dotyczy podstawowych danych o preferencjach użytkownika lub tak konkretnych informacji jak punkty czy status), szanse skutecznego przyciągnięcia uwagi użytkownika są znacznie większe. Po zwróceniu uwagi użytkownika aplikacja może dalej wysyłać na serwer żądania HTTP w celu uzyskania dodatkowej, mniej ważnej treści i stopniowo ładować kolejne elementy danych.

Opisana technika pozwala możliwie szybko zainteresować użytkownika funkcjami aplikacji i — tym samym — zwiększyć liczbę aktywnych użytkowników tej aplikacji.

Kiedy dobra aplikacja okazuje się zła?

Mimo najlepszych intencji programisty czasem dobra aplikacja nie przynosi spodziewanych efektów wskutek wyboru niewłaściwej architektury, niedostatecznej skalowalności lub zbyt ograniczonego wykorzystania dostępnych narzędzi społecznościowych. Okazuje się jednak,

że wystarczy odrobina planowania przed przystąpieniem do właściwego tworzenia aplikacji, aby zbudować skalowalny rdzeń programu, uwzględnić dostępne uchwyty społecznościowe niezbędne do przyciągania ruchu i — w dłuższej perspektywie — zarobić pieniądze.

Przed przystąpieniem do realizacji projektu programiści powinni zadać sobie kilka pytań:

- Dla jakiej grupy odbiorców jest tworzona dana aplikacja? Czy w przyszłości aplikacja będzie rozszerzana z myślą o innych grupach odbiorców lub innych językach?
- Jak ma być skalowany szkielet serwera pod kątem obsługi chwilowych wzrostów obciążenia, jeśli z czasem aplikacja zyska popularność?
- Które funkcje społecznościowe kontenera można wykorzystać do przyciągania nowego ruchu i zachęcania użytkowników do instalowania aplikacji?
- Jak można zarobić na tej aplikacji?

To tylko niektóre podstawowe pytania, na które należy sobie odpowiedzieć jeszcze przed przystąpieniem do budowy aplikacji. O sukcesie aplikacji decydują właśnie solidne podstawy i przemyślany plan na przyszłość.

Warto przeanalizować kilka przypadków użycia ilustrujących wpływ tych aspektów na przebieg wytwarzania aplikacji.

Przenośna aplikacja z animacjami

Zdolność szybkiego dostosowywania oprogramowania do różnych środowisk i przenośność pomiędzy kontenerami to specjalność wielu producentów oprogramowania. Umiejętność sprawnego tworzenia kompletnych aplikacji, w tym szybkiego opanowania nowej platformy, przygotowania ramki `iframe` odwołującej się do serwerów producenta oprogramowania i ładowania animacji Flash, na pierwszy rzut oka zapewnia przewagę wydawcy oprogramowania, jednak taki model wytwarzania ma wiele wad w kontekście aplikacji społecznościowych.

Jak już wspomniałem, wiele kontenerów aplikacji społecznościowych narzuca ograniczenia (w tym ograniczenia dotyczące animacji Flash i skryptów języka JavaScript) na treść prezentowaną w małych widokach aplikacji. Z perspektywy producenta, który zasadniczą treść aplikacji udostępnia w formie animacji Flash, takie ograniczenia oznaczają, że mały widok albo ma postać reklamy nakłaniającej użytkownika do odwiedzenia dużego widoku lub serwisu producenta, albo ma niewielki związek z samą aplikacją. W tej sytuacji trudno oczekiwać, by mały widok oferował funkcje podobne do tych dostępnych w dużym widoku.

Drugim istotnym problemem są funkcje społecznościowe. Przenośność to nie jedyny aspekt, który należy mieć na uwadze podczas tworzenia aplikacji z myślą o wielu kontenerach. Aplikacje na bazie animacji Flash zbyt często są ściśle uzależnione od funkcji społecznościowych określonego kontenera, w tym uchwytów aktywności (używanych do prezentowania powiadomień), grafów znajomych (używanych do promocji aplikacji), a nawet prostych systemów przekazywania komunikatów (wykorzystywanych do przyciągania dodatkowego ruchu i zachęcania do instalacji). Zaangażowanie użytkowników tego rodzaju aplikacji zwykle jest dość płytkie. Aplikacje cechują się dość szybką wymianą użytkowników i ostatecznie tracą popularność wskutek wyczerpania się grup potencjalnych odbiorców i — tym samym — możliwości rozwoju. Bez społecznościowych punktów zaczepienia i odpowiednich widoków aplikacja z natury rzeczy nie może w dłuższej perspektywie utrzymać zaangażowania użytkowników.

Wytwarzanie aplikacji dla wielu kontenerów powinno polegać na budowie warstwy abstrakcji działającej pomiędzy kontenerem a samą aplikacją. Tylko w ten sposób można zbudować jedną, wysoce społecznościową aplikację, aby następnie uzupełniać ją o punkty zaczepienia dla funkcji i danych społecznościowych dowolnego kontenera zintegrowanego z tak skonstruowaną architekturą.

Niedopracowany widok

Wspomniałem już, że odpowiednio zaprojektowane małe widoki mogą bardzo pomóc w budowaniu popularności i pogłębianiu zaangażowania użytkowników. Okazuje się, że to samo dotyczy większych widoków. Niektórzy producenci oprogramowania woleliby pracować tylko nad większymi widokami; inni najchętniej ograniczyliby się do tworzenia małych widoków, ponieważ w większości przypadków to one są prezentowane użytkownikom jako pierwsze.

Warto najpierw przeanalizować kwestię małych widoków aplikacji. Wspomniałem już, że niektóre kontenery nakładają na małe widoki pewne ograniczenia, które uniemożliwiają na przykład stosowanie kodu języka JavaScript lub animacji Flash. Ograniczenia tego typu są głównym czynnikiem zniechęcającym programistów do tworzenia bogatych i atrakcyjnych małych widoków. Skoro właśnie mały widok jest pierwszym punktem interakcji użytkownika i aplikacji, to gdy jest on odpowiednio zaprojektowany i atrakcyjny, może mieć zasadniczy wpływ na wzrost popularności tej aplikacji. Jeśli jednak dany użytkownik regularnie korzysta z aplikacji, należy tak zmienić mały widok, aby raczej zachęcał użytkownika do dalszej interakcji za pośrednictwem dużego widoku. W przypadku gry funkcję takiej zachęty mogą pełnić bieżące statystyki, zestawienie ich ze statystykami znajomych lub informacje o nowych elementach, które użytkownik może zdobyć podczas dalszej rozgrywki. W przypadku aplikacji wspierających prace biurowe mały widok może przypominać o najbliższych zadaniach, prezentować zestawienia tych czynności z zaplanowanymi zadaniami znajomych lub udostępniać proste funkcje aplikacji.

Wielu programistów nie tworzy dużych widoków, ponieważ budowane aplikacje oferują na tyle proste i nieliczne funkcje, że ograniczenia obowiązujące w kontenerze nie stanowią żadnego problemu. Taka postawa jest przejawem dość wąskiego postrzegania aplikacji społecznościowej i zwykle oznacza, że twórca tej aplikacji nie przewiduje jej rozwoju w przyszłości — nie dostrzega dodatkowych funkcji i danych, których mogą oczekiwać użytkownicy aplikacji. Większość kontenerów nie oferuje opcji wyłączania widoków i wyświetla puste strony w miejsce niezdefiniowanych widoków. Oznacza to, że programista musi dodatkowo podejmować próby ukrywania brakujących widoków przed użytkownikami (przechodzącymi pomiędzy widokami), co może wymagać dodatkowych nakładów (zależnie od implementacji mechanizmów przełączania widoków w ramach kontenera). W opisanym scenariuszu sam programista zawęża ofertę kierowaną do użytkowników do mniejszych widoków zawierających mniej funkcji, ponieważ chce uniknąć kosztów związanych z budową naprawdę atrakcyjnej aplikacji, która możliwie najpełniej wykorzysta istniejące funkcje kontenera. Użytkownicy błyskawicznie odkrywają, że aplikacja próbuje dostarczać im płytką, łatwo dostępną treść lub że korzystanie z niej jest możliwe tylko w jednym miejscu.

Aplikacja kopiująca widoki

Każdy programista wie, jak kuszące jest kopiowanie kodu jednego widoku do wszystkich pozostałych widoków. Celem programistów jest udostępnienie użytkownikowi możliwie wielu funkcji (i jak największej liczby widoków) możliwie niewielkim kosztem (rozumianym jako zasoby inżynierskie). Kopiowanie jednego widoku we wszystkich miejscach, gdzie może być dostrzeżony przez użytkownika, jest jednak dalece niepożądane.

Problem w tym, że użytkownicy błyskawicznie odkryją stosowanie tej techniki i wyrobią sobie negatywne zdanie o aplikacji, jeśli po przejściu pomiędzy małym a dużym widokiem (lub odwrotnie) otrzymają dokładnie tę samą treść. Wielu użytkowników traci zaufanie do aplikacji już w momencie napotkania pierwszego elementu sprawiającego wrażenie uszkodzonego lub niedopracowanego. Skoro programista nie zechciał poświęcić swojego czasu na dopracowanie swojej aplikacji, czy można mieć pewność, że poważnie potraktował choćby kwestię ochrony informacji społecznościowych?

Warto najpierw przeanalizować aspekt relacji zaufania w opisanym scenariuszu. Jeśli budowana aplikacja ma na celu albo zarabianie pieniędzy dzięki aktywności użytkowników, albo kierowanie użytkowników do jakiegoś źródła, ostatnią rzeczą, na którą może sobie pozwolić programista, jest utrata zaufania użytkownika. Jeśli użytkownik aktywnie przechodzący pomiędzy widokami stale otrzymuje ten sam ekran z tymi samymi funkcjami, zaczyna podejrzewać, że aplikacja nie działa prawidłowo — że zdarzyło się coś, co nie powinno mieć miejsca. Użytkownik z pewnością będzie o tym pamiętał, kiedy dojdzie do punktu, w którym aplikacja zażąda płatności. Ostatecznym celem producenta aplikacji jest budowanie relacji zaufania potrzebnej do uzyskania od użytkowników zwrotu z inwestycji (w tej czy innej formie).

Warto jeszcze zwrócić uwagę na różnice dzielące poszczególne widoki. Jeśli mały widok jest kopiowany do dużego widoku (lub odwrotnie), programista prędzej czy później napotka jeden z opisanych poniżej problemów:

- Jeśli aplikacja jest budowana pod kątem określonej szerokości widoku (nie zawiera elastycznej obsługi wymiarów), w jednym z widoków będzie sprawiała wrażenie za małej lub za dużej. Jeśli treść jest zbyt duża dla widoku, użytkownik będzie musiał przewijać ramkę widoku. W obu przypadkach jego doznania będą dalekie od ideału.
- Jeśli aplikacja dysponuje mechanizmami elastycznego dostosowywania szerokości (z dzieleniem wierszy w razie braku przestrzeni), można znacznie złagodzić skutki części opisanych powyżej problemów. Nawet wspomniane mechanizmy nie eliminują jednak problemu zbyt dużej i zbyt małej ilości treści. Jeśli aplikacja prezentuje zbyt rozbudowaną treść, mały widok sprawia wrażenie przepełnionego; podobnie, jeśli aplikacja oferuje zbyt mało treści, duży widok sprawia wrażenie pustego i niedopracowanego.

Niezależnie od tego, który z opisanych powyżej scenariuszy dotyczy realizowanego projektu, opracowanie rozwiązań dla wspomnianych problemów będzie się wiązało z kosztami zbliżonymi do kosztów opracowania drugiego widoku.

Aplikacja prezentująca zbyt dużo informacji

Kolejnym przypadkiem użycia, który warto przeanalizować, jest aplikacja nadużywająca strumienia aktywności użytkownika. Bezmyślne zarzucanie użytkownika powiadomieniami o wszystkich możliwych zdarzeniach powoduje, że aplikacja jest postrzegana wyłącznie jako

źródło wiadomości (w tym wiadomości niepotrzebnie ujawnianych znajomym użytkownika). Każdy, kto kiedykolwiek używał aplikacji w ramach kontenera aplikacji społecznościowych, zapewne zetknął się z odpowiednimi przykładami.

Istotą problemu jest nieprzemyślany wybór rodzajów informacji udostępnianych znajomym bieżącego użytkownika. Jak reaguje użytkownik otrzymujący ciągłe powiadomienia od aplikacji, której sam nigdy nie widział ani nie instalował? Większość użytkowników odruchowo wyłącza wszystkie powiadomienia pochodzące z tej aplikacji (jeśli ma do dyspozycji odpowiednie ustawienia) lub wręcz wyłącza wszystkie powiadomienia od danego użytkownika. W obu przypadkach reakcja użytkownika eliminuje ważny kanał komunikacji, który powinien łączyć tę aplikację z bazą potencjalnych użytkowników. Nadmierna liczba powiadomień utrudnia więc docieranie do nowych użytkowników. Istnieje zaledwie kilka wartościowych kanałów dostępnych dla twórców aplikacji i umożliwiających trafianie do potencjalnych użytkowników, zatem dopuszczenie do utraty jednego z tych kanałów (bodaj najważniejszego) może mieć fatalny wpływ na czas życia aplikacji.

W dalszej części tej książki zostaną omówione alternatywne metody udostępniania informacji, które jednocześnie umożliwiają osiąganie celów stawianych sobie przez programistów i nie zaśmiecają strumienia aktywności użytkowników.

Nierentowna aplikacja

Jednym z najsmutniejszych doznań w świecie aplikacji jest obserwowanie, jak świetny pomysł został zaprzepaszczony wskutek złej realizacji. Programiści wciąż tworzą bardzo udane aplikacje, całkowicie zapominając o kwestii zarabiania pieniędzy na tych projektach (dzięki użytkownikom korzystającym z ich produktów). Producenci, którzy nie sprzedają swoich aplikacji czołowym firmom branży oprogramowania, muszą zadbać o środki na pokrycie kosztów obsługi rosnącej liczby użytkowników przez serwer.

Nawet jeśli nie jest możliwe zaimplementowanie od razu wszystkich technik generowania przychodów, warto od początku zaplanować stopniową rozbudowę aplikacji o nowe rozwiązania. Należy zidentyfikować najważniejsze obszary aplikacji, które mogą w ten czy inny sposób generować zwrot z inwestycji. Wyznaczenie sobie jasnych celów, w tym czasu i sposobu integracji elementów komercyjnych w ramach aplikacji, bez wątpienia powinno być częścią długoterminowej strategii biznesowej. Producent aplikacji może na przykład zaplanować stworzenie wewnętrznego sklepu, którego oferta będzie prezentowana dopiero po osiągnięciu określonej liczby użytkowników lub pewnego progu użytkowników codziennie korzystających z aplikacji.

Sposób oferowania komercyjnych usług jest równie ważny jak sama koncepcja zarabiania dzięki użytkownikom. Jeśli producent aplikacji chce żądać dodatkowych opłat, musi przekonać użytkowników, że skorzystanie z oferty przyniesie im korzyści (niezależnie od tego, czy chodzi o dodatkowe funkcje, czy o nową treść). Aplikacja powinna z jednej strony oferować przydatne funkcje także tym użytkownikom, którzy nie są skłonni wydawać dużo pieniędzy lub wręcz nie chcą ponosić żadnych kosztów, ale z drugiej strony musi też jasno wskazywać wymierne korzyści wynikające z dodatkowych funkcji (na przykład z wykupienia konta premium).

Istnieje wiele technik spieniężania doznań użytkownika; kilka z nich wymieniono poniżej:

- Sklep w ramach gry może oferować nową treść i elementy rozgrywki rozszerzające doznania użytkownika aplikacji. Taki sklep zawiera wyłącznie wirtualne „towary", bez żadnych rzeczywistych przedmiotów.
- Platformy reklamowe w ramach aplikacji społecznościowych. Istnieje wiele platform reklamowych, które zaprojektowano z myślą o aplikacjach działających w ramach kontenera aplikacji społecznościowych.
- Sklep internetowy oferujący w ramach produktu rzeczywiste przedmioty albo związane z daną aplikacją, albo dostosowane do informacji społecznościowych podanych przez użytkownika (w ten sposób można kierować ofertę do określonych grup).

Opisane techniki są powszechnie stosowane w wielu sprawdzonych aplikacjach (w ramach wielu popularnych kontenerów aplikacji społecznościowych). Taktowne, przemyślane techniki sprzedaży (zmyślnie wplecione w logikę aplikacji) nie muszą być irytujące i — tym samym — nie muszą narażać aplikacji na utratę bazy użytkowników.

Aplikacja informacyjna

Ostatnim typem aplikacji, który warto przeanalizować, jest typ dość często stosowany przez wydawców podejmujących próby publikowania informacji w alternatywnych źródłach, na przykład za pośrednictwem aplikacji w ramach kontenerów społecznościowych. W wielu przypadkach wydawcy chcieliby po prostu wykorzystać gotowy kod XML-a lub kanał RSS z wiadomościami, zastosować arkusz stylu i zamieścić link do oryginalnego źródła, czyli zwykle własnego serwisu internetowego. Opisany model aplikacji z pewnością umożliwia łatwą i szybką integrację, ale nie zawsze jest skuteczny.

Powody, dla których wydawcy decydują się na implementację tego rodzaju aplikacji, zwykle są dość proste. Albo nie chcą angażować zbyt dużo zasobów inżynierskich w aplikację, dopóki nie zobaczą wymiernych korzyści (zwrotu z inwestycji na przykład w formie większej liczby aktywnych użytkowników), albo wolą nie udostępniać całej oferowanej treści za pośrednictwem innego serwisu, gdzie nie mają pełnej kontroli nad mechanizmami śledzenia zachowań użytkowników i działaniami marketingowymi.

Po pierwsze, wydawcy stosujący tę metodę zwykle nie decydują się na opracowanie wszystkich widoków aplikacji lub wypełniają wszystkie widoki tą samą treścią. Negatywne skutki stosowania tego modelu aplikacji omówiłem już w punkcie „Aplikacja kopiująca widoki" (we wcześniejszej części tego rozdziału), zatem nie będę powtarzał tych wniosków w tym miejscu.

Obok kwestii widoków największą wadą tego modelu aplikacji jest stosunkowo płytka implementacja, która nie ma większej wartości z perspektywy użytkownika. W tym przypadku jedyną różnicą dzielącą korzystanie z aplikacji od czytania tej samej treści za pośrednictwem kanału RSS jest styl prezentowania informacji. Co więcej, korzystanie z linków przenoszących użytkownika aplikacji z kontenera aplikacji społecznościowych do serwisu wydawcy jest dość irytujące. Użytkownicy korzystają z aplikacji głównie po to, aby mieć dostęp do interesującej ich treści w jednym miejscu; nie chcą, aby ich doznania estetyczne były gwałtownie zmieniane tylko dlatego, że chcą się zapoznać z rozwinięciem treści.

Tworzenie aplikacji informacyjnych jest o tyle specyficzne, że większość ich użytkowników woli uzyskiwać informacje bezpośrednio ze swoich ulubionych źródeł. Wiele lat temu udowodniono, że zamknięty model wytwarzania aplikacji, gdzie technologia ani dane nie są

udostępniane poza firmą, nie zdaje egzaminu. Otwarcie dostępu do danych i źródeł pozwala dotrzeć do zupełnie nowych grup odbiorców. Właściwe traktowanie tych grup umożliwia pozyskanie lojalnych czytelników, co z kolei przekłada się na sukces komercyjny aplikacji.

Okazuje się, że opisany model można dość łatwo przebudować, tak aby zapewniał użytkownikom wymierne korzyści. Wydawcy nie muszą udostępniać całej treści swoich wiadomości w kontekście tych aplikacji. Większość użytkowników przegląda tytuły i streszczenia, aby na tej podstawie ocenić, czy są zainteresowani pełną treścią wiadomości. Wydawcy powinni obsługiwać ten przypadek użycia, wyświetlając w widoku aplikacji co najwyżej dwa akapity niezbędne do przykucia uwagi użytkownika. Pod tą skróconą treścią można wyświetlić link do pełnej treści w witrynie internetowej wydawcy. Ta stosunkowo niewielka modyfikacja aplikacji ma zasadniczy wpływ na jej skuteczność.

Po drugie, w dużym widoku aplikacji programiści mogą integrować dodatkowe opcje konfiguracyjne, dzięki czemu użytkownicy będą otrzymywali tylko te dane, którymi są zainteresowani. Ten dodatkowy panel konfiguracyjny udostępni użytkownikom dodatkową warstwę funkcji, która ułatwi im uzyskiwanie wartościowych (z ich punktu widzenia) informacji. W ten sposób można też zwiększyć codzienną aktywność użytkowników.

Jak się okazuje, modyfikowanie opisanego modelu aplikacji jest wyjątkowo proste. Postrzeganie aplikacji jako interaktywnego obiektu (nie jako miejsca prezentowania informacji) może bardzo ułatwić jej rozwój, pozyskiwanie nowych użytkowników i zwiększanie liczby użytkowników odwiedzających oryginalny, źródłowy serwis wydawcy.

Studia przypadków dla modeli aplikacji

Skoro omówiłem już kilka podstawowych modeli aplikacji, które nie zdają egzaminu, i skoro wyjaśniłem, dlaczego tak budowane aplikacje nie przynoszą oczekiwanych efektów, czas przeanalizować trzy różne modele aplikacji, które sprawdzają się w świecie serwisów społecznościowych. W tym podrozdziale przedstawię konkretne firmy implementujące poszczególne modele.

Poniższe studia przypadków będą dotyczyły następujących modeli:

- aplikacji gier społecznościowych ze znajomymi;
- aplikacji sprzedaży produktów;
- aplikacji uwzględniających położenie użytkownika.

Ten podrozdział można traktować jako przegląd sprawdzonych modeli biznesowych, które zdały egzamin w świecie aplikacji, rozwiązań mobilnych i serwisów społecznościowych.

Studium przypadku: gra społecznościowa ze znajomymi

Pierwsze studium przypadku dotyczy aplikacji gry społecznościowej, której infrastruktura jest zbudowana na bazie interakcji społecznościowych użytkownika (czyli dwu- lub wielostronnych powiązań ze znajomymi). Budowa aplikacji na bazie rzeczywistych relacji łączących przyjaciół, rodziny, współpracowników i wszystkich innych uczestników strumienia aktywności jest doskonałą metodą konstruowania grafu powiązań na potrzeby gry społecznościowej.

Warto przeanalizować przykład jednego z największych przedstawicieli branży gier społecznościowych, czyli firmy Zynga. Firma ta zbudowała swój model biznesowy na tworzeniu wyjątkowo wciągających gier społecznościowych, które należą do najbardziej popularnych produktów na tym rynku. Gry tego typu można kochać lub nienawidzić, jednak sama firma Zynga osiągnęła jeden z największych sukcesów w branży gier społecznościowych, głównie dzięki takim tytułom jak FarmVille, CityVille czy Mafia Wars. Tacy producenci jak Zynga stosują wiele ciekawych taktyk budowania uzależniających gier, które przynoszą ich twórcom ogromne zyski.

Mimo że to studium przypadku dotyczy aplikacji gier społecznościowych, prezentowane zasady można z powodzeniem stosować w dowolnych aplikacjach społecznościowych korzystających z grafów powiązań (na przykład listy znajomych).

Zrozumieć grupę odbiorców

Precyzyjne zdefiniowanie docelowych grup odbiorców (na przykład cech demograficznych przyszłych użytkowników) powinno być jednym z pierwszych kroków w pracy każdego programisty aplikacji społecznościowej. Samo określenie, kto już teraz należy do grupy odbiorców aplikacji, nie wystarczy. Jeśli na przykład budowana aplikacja gry społecznościowej ma trafić do grupy, która miała już wielokrotnie kontakt z innymi, często świetnymi grami społecznościowymi, trudno oczekiwać zwrotu z inwestycji na poziomie innej, bardziej niszowej grupy.

Firma Zynga opanowała tę sztukę do perfekcji. W przeszłości gry internetowe były kierowane do konkretnych grup demograficznych, na przykład chłopców i mężczyzn w wieku od 14 do 25 lat, i były tworzone z myślą o najbardziej aktywnych użytkownikach. Firma Zynga zmieniła świat gier internetowych, kierując swoje produkty także do mniej aktywnych, weekendowych graczy. Firmie udało się z powodzeniem trafić do wielu młodszych i starszych użytkowników, a także użytkowników płci pięknej, niepasujących do stereotypu gracza. Firma Zynga stworzyła cały rynek gier społecznościowych dla dorywczych, weekendowych graczy.

Budowanie grafu powiązań w ramach gry

Nawet jeśli graf powiązań społecznościowych platformy, na której ma działać gra społecznościowa, nie ma większej wartości (jeżeli na przykład użytkownicy dysponują wieloma powiązaniami dwustronnymi z innymi użytkownikami, ale nie łączą ich z nimi żadne istotne, rzeczywiste relacje), nie należy rezygnować z budowania grafu powiązań społecznościowych w ramach aplikacji jako jednego z najważniejszych elementów tworzonej gry.

Firma Zynga udowodniła znaczenie powiązań społecznościowych przy okazji przenoszenia swoich aplikacji pomiędzy różnymi platformami, w tym platformami, które od bardzo niedawna miały postać portali społecznościowych, jak w przypadku serwisu YAP. Niezależnie od stanu i wartości powiązań społecznościowych w ramach platformy środowisko gry oferuje możliwość budowy grafu powiązań na poziomie niezbędnym do uatrakcyjnienia rozgrywki.

Powiązania społecznościowe można wykorzystać do stworzenia rozbudowanego mechanizmu zaproszeń, tak aby użytkownik mógł wysyłać do swoich znajomych powiadomienia z zachętą do przystąpienia do rozgrywki. Cały ten proces musi też przynosić jakieś korzyści samemu użytkownikowi; w przeciwnym razie będzie nieefektywny. Firma Zynga włączyła mechanizm

zaproszeń do rozgrywki w swoich produktach — w grze Mafia Wars gracz zwiększa siłę swojej organizacji przestępczej poprzez dodawanie nowych członków, a w grze FarmVille gracz uzyskuje pomoc w prowadzeniu swojej farmy. Możliwość uzyskiwania podobnych korzyści wskutek dodawania użytkowników motywuje graczy do poszukiwania kolejnych znajomych o zbliżonych zainteresowaniach, aby z ich pomocą poprawić własne wyniki. Mimo że ci znajomi nie muszą się znać osobiście, ich znajomość ma praktyczne znaczenie dla rozgrywki, ponieważ obie strony dzielą te same zainteresowania i wzajemnie korzystają na swojej znajomości.

Możliwość wzajemnej interakcji znajomych w grze

Poprzedni punkt można dodatkowo rozwinąć o korzyści płynące z wzajemnej interakcji użytkowników. Stworzenie możliwości interakcji społecznościowych w ramach gry jest równie ważne jak budowanie odpowiedniego grafu powiązań społecznościowych. Interakcja w ramach rozgrywki jest doskonałym sposobem zwiększania liczby aktywnych (najlepiej codziennie) użytkowników aplikacji.

Aby lepiej zilustrować tę koncepcję, warto raz jeszcze wrócić do przykładu gry Mafia Wars firmy Zynga. Gracz uczestniczący w rozgrywce wraz z przyjaciółmi może skorzystać z kilku wbudowanych mechanizmów, które operują bezpośrednio na grafie powiązań społecznościowych zbudowanym przez tego użytkownika w ramach gry:

- gracz może rozpocząć wykonywanie misji, podczas której będzie mógł korzystać z pomocy znajomych;
- gracz może prosić swoich znajomych o potrzebne elementy w grze (prezenty).

Przekazywanie prezentów jest jednym z najpopularniejszych aspektów tego rodzaju gier. Opcje przekazywania prezentów i proszenia o potrzebne elementy (na przykład raz dziennie) mogą zwiększyć przywiązanie użytkowników do swoich tożsamości w ramach gry i jednocześnie podnieść ich aktywność.

Krótko mówiąc, po stworzeniu kanałów przyciągania użytkowników do gry należy jeszcze opracować mechanizmy utrzymywania ich zaangażowania w rozgrywkę.

Jasne korzyści z działań podejmowanych w grze

Innym bardzo ważnym aspektem tego studium przypadku jest prezentowanie jasnych, czytelnych zachęt do dalszego korzystania z aplikacji (dla użytkowników, którzy zaczęli rozgrywkę). Innymi słowy, należy stworzyć użytkownikowi szanse wirtualnego rozwoju w ramach gry lub aplikacji.

Istnieje wiele różnych technik utrzymywania zainteresowania użytkownika w dłuższym terminie. Oto kilka z nich:

- Możliwość uzyskania coraz wyższych poziomów mocy w ramach gier zależnie od czasu gry i liczby działań wykonanych przez użytkownika.
- Możliwość doskonalenia postaci użytkownika w grze, tak aby użytkownik mógł łatwo skojarzyć czas poświęcany rozgrywce z postępami swojego awatara.
- Udostępnienie mechanizmów rozwoju w czasie rzeczywistym. Budowa nowych obiektów czy wzrost uprawy zboża w pewnym czasie zachęci użytkowników do wielokrotnego otwierania aplikacji i sprawdzania postępu tych procesów. Ten aspekt można dodatkowo

rozwinąć, czasowo udostępniając okno z możliwością wykonania jakiejś czynności po zakończeniu procesu budowy (na przykład zebrania plonu zanim uschnie na polu); jeśli gracz nie wykona tej czynności w określonym czasie, utraci efekt budowy. Takie rozwiązanie znacznie zwiększa zaangażowanie użytkowników we wspomnianych okresach.

Oferowanie tego rodzaju korzyści pomaga zwiększać liczbę graczy codziennie otwierających aplikację, co z kolei stwarza więcej okazji do komercyjnego wykorzystania ich zaangażowania.

Integracja kanałów społecznościowych za pośrednictwem poczty elektronicznej, powiadomień i czynności

Platformy aplikacji zwykle udostępniają programistom wiele cennych kanałów komunikacji z użytkownikami lub zapraszania innych użytkowników do gry. Mimo że wspomniane kanały należą do najważniejszych funkcji umożliwiających promocję aplikacji wśród użytkowników, dla większości programistów stanowią tylko pewien dodatek (którego implementacja jest mniej ważna niż rozwijanie samego produktu).

Warto przeanalizować przykład firmy Zynga. Wystarczy przez pewien czas monitorować aktywność aplikacji tej firmy, aby odkryć, że statyczny tekst powiadomień o aktualizacjach jest stale poddawany drobnym modyfikacjom. Firma Zynga stale doskonali tekst swoich powiadomień, ponieważ wie, że właśnie stosowany język decyduje o skuteczności pozyskiwania nowych użytkowników w wyniku informacji o aktualizacjach. Strumień aktywności użytkownika jest ściśle związany z tym kanałem powiadomień (odpowiednio zredagowane informacje są traktowane przez użytkownika na równi z informacjami o ważnych zdarzeniach dotyczących jego znajomych).

Platformy aplikacji zwykle udostępniają programistom alternatywne metody kontaktowania się z użytkownikami, w tym bezpośrednią korespondencję pocztą elektroniczną (inicjowaną przez użytkownika) oraz powiadomienia o czynnościach lub elementach, które wymagają akceptacji bądź odrzucenia przez użytkownika. W przeciwieństwie do strumienia aktywności użytkownika jako pasywnego mechanizmu zwiększania zaangażowania użytkowników lub pozyskiwania nowych użytkowników, wiadomości poczty elektronicznej i powiadomienia wymagają od użytkownika dodatkowych czynności. Niezbędna czynność może polegać na usunięciu wiadomości lub na wykonaniu proponowanych działań, jednak w obu przypadkach wiadomość wymaga uwagi adresata.

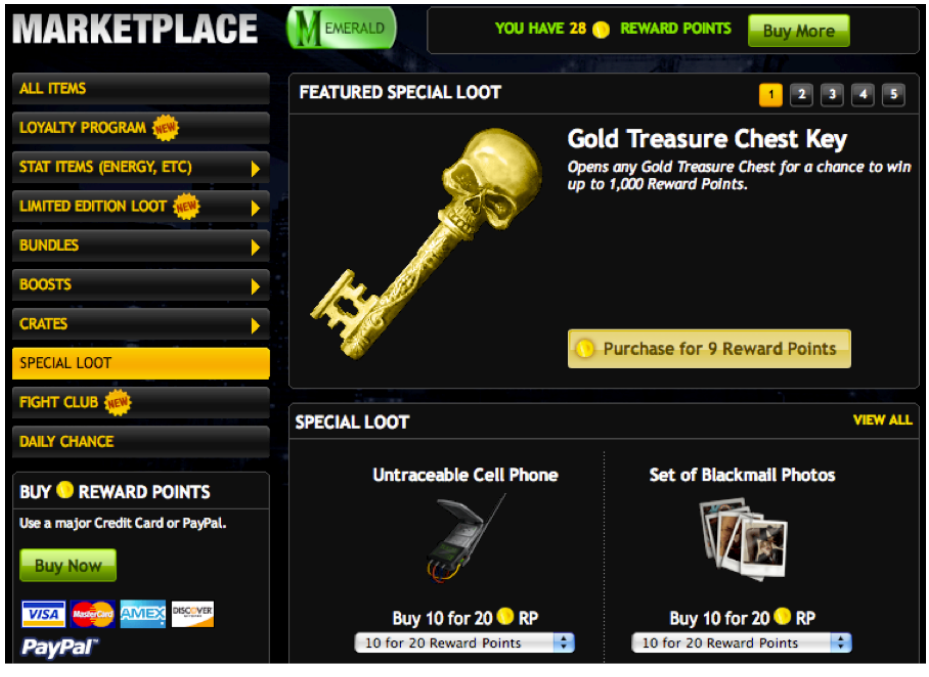
Nie należy nadużywać tych kanałów komunikacji i wysyłać do użytkowników zbyt wielu komunikatów. Nikt nie lubi aplikacji zarzucających skrzynkę pocztową niechcianymi wiadomościami — takie działanie z pewnością będzie miało negatywny wpływ na ocenę aplikacji.

Docelowo aplikacja powinna wykorzystywać wszystkie kanały komunikacji oferowane przez daną platformę. Architekturę aplikacji należy zaprojektować w taki sposób, aby wykorzystywała wszelkie okazje komunikacji z użytkownikami, ale też aby nie nadużywała żadnego z dostępnych kanałów. Warto analizować planowane formy komunikacji z perspektywy użytkowników aplikacji i ich komfortu.

Zarabianie na sprzedaży wirtualnych towarów

Jednym z najskuteczniejszych sposobów komercyjnego wykorzystania aplikacji jest oferowanie produktów, których tworzenie nie wiąże się z żadnymi kosztami. Właśnie w ten sposób zarabiają najbardziej dochodowe gry w tej branży.

Warto raz jeszcze wrócić do przykładu gry Mafia Wars firmy Zynga. W ramach gry użytkownik może kupować punkty nagród (patrz rysunek 1.9), czyli wirtualną walutę przydatną podczas rozgrywki. Gracze mogą używać tej wirtualnej waluty do kupowania dodatkowego wyposażenia i rozmaitych ułatwień w grze, aby uzyskać przewagę nad konkurencją. Firma Zynga zarabia pieniądze na czymś zupełnie wirtualnym, nienamacalnym, zatem koszty „produkcji" tych towarów są praktycznie zerowe (choć oczywiście firma ponosi koszty utrzymania serwerów, pracowników itp.).



Rysunek 1.9. Sklep dostępny w grze Mafia Wars firmy Zynga

Strumień pieniędzy płynący do producenta gry można poszerzać, oferując w wirtualnym sklepie nowe elementy — jedynym ograniczeniem tej oferty jest wyobraźnia inżynierów i projektantów aplikacji.

Studium przypadku: aplikacje sprzedaży produktów

Drugie studium przypadku będzie dotyczyło zupełnie innego rodzaju aplikacji niż przykłady gier społecznościowych omówione w poprzednim punkcie.

Zaledwie kilka lat temu na rynku pojawiły się takie firmy jak Groupon czy Living Social (oraz wiele firm stosujących zbliżone modele biznesowe), które zbudowały swoje imperia na bazie koncepcji oferowania towarów i usług po mocno obniżonych cenach.

W kolejnych podpunktach zostaną przedstawione wnioski płynące z doświadczeń tych firm, w tym elementy, które zdecydowały o ogromnym sukcesie ich aplikacji.

Gry to nie wszystko

Wielu użytkowników obserwuje nadmierne nasycenie serwisów społecznościowych grami. Warto więc podkreślić, że *gry to nie jedyna przestrzeń dla aplikacji społecznościowych*. Co więcej, wiele aplikacji stworzonych z myślą o typowych zadaniach biurowych czy sprzedaży produktów zarobiło miliony dla swoich twórców. Aplikacje tego typu, które z natury rzeczy są społecznościowe i ułatwiają codzienne życie użytkowników, mogą zyskać równie dużą popularność co gry. O ich sukcesie decydują następujące warunki:

- muszą rozwiązywać rzeczywiste problemy użytkowników;
- muszą odpowiadać na pytania, na które użytkownik nie może znaleźć odpowiedzi gdzie indziej (na przykład w publicznych elementach serwisu społecznościowego);
- muszą usprawniać sposób wykonywania standardowych czynności.

To tylko kilka z powodów, dla których użytkownicy decydują się na korzystanie z aplikacji poprawiających produktywność i aplikacji-sklepów.

Nowa realizacja starej koncepcji

O ogromnym sukcesie Groupona decyduje przede wszystkim wykorzystanie starego, doskonale znanego pomysłu (kuponów) i dostosowaniu go do współczesnego świata rozwiązań społecznościowych i mobilnych. Okazuje się jednak, że zamiast wykorzystać stary pomysł i uzupełnić go o pewne elementy społecznościowe i mobilne, Groupon całkowicie przebudował koncepcję kuponu.

Czytelnicy, którzy nie rozumieją idei Groupona, powinni wyobrazić sobie świat, w którym codziennie mają miejsce największe wyprzedaże (porównywalne z tzw. czarnym piątkiem w Stanach Zjednoczonych czy brytyjskimi wyprzedażami poświątecznymi).

To jeden z najważniejszych aspektów budowy aplikacji obejmującej elementy sprzedaży. Użytkownicy nie oczekują kolejnych sposobów realizacji tych samych działań — potrzebują raczej jasnego, odpowiednio atrakcyjnego uzasadnienia dla stosowania tej czy innej aplikacji. Takim uzasadnieniem może być właśnie **innowacyjność**. Produkt musi oferować nowy sposób realizacji znanych czynności. Musi zapewniać coś niepowtarzalnego w związku z oferowanymi towarami czy usługami.

Groupon nie tylko umożliwia użytkownikom znajdowanie interesujących ich towarów i usług w promocyjnych cenach, ale też proponuje im nowe produkty i doznania, z którymi być może nigdy wcześniej się nie zetknęli. Wielu użytkowników lubi korzystać z Groupona właśnie dlatego — dzięki temu serwisowi mogą robić to, czego nigdy wcześniej nie robili.

Właśnie te czynniki decydują o sukcesie Groupona jako doskonałego produktu.

Prowokowanie dyskusji w celu uzyskiwania i przekazywania informacji

Kolejną zaletą Groupona jest możliwość prowadzenia przez potencjalnych kupujących dyskusji na temat każdego dostępnego produktu i każdej usługi. Co ciekawe, udział w tych rozmowach biorą nie tylko użytkownicy, ale też pracownicy Groupona, a w wielu przypadkach także przedstawiciele firm oferujących produkty i usługi.

Takie rozmowy mają wiele zalet, w tym:

- Użytkownicy, którzy z różnych względów nie potrafią podjąć decyzji o zakupie produktu lub usługi, mogą bezpośrednio wyrażać i rozwiewać swoje wątpliwości dotyczące danej oferty.
- Przedstawiciele producenta aplikacji, jej użytkownicy i podmioty oferujące swoje produkty operują na tym samym poziomie i uczestniczą na równych prawach w wątkach dyskusji. To wprost doskonały sposób poprawienia komfortu potencjalnych kupujących w procesie lepszego poznawania oferowanych towarów i usług.

Takie metody jak obszary dyskusji nierzadko wymagają trochę czasu i wysiłku ze strony klientów (a także osób, które znają już oferowane produkty i dzielą się swoją wiedzą). Warto więc dodatkowo przygotować dział pytań i odpowiedzi (FAQ) dla produktów i usług, tak aby wyprzedzać wątpliwości potencjalnych kupujących i — tym samym — zwiększyć liczbę transakcji.

Przekazywanie usług w formie podarunków

Aplikacje poprawiające wydajność pracy oraz aplikacje sprzedające towary lub usługi mogą dodatkowo zwiększyć liczbę transakcji, oferując użytkownikom przekazywanie kupowanych produktów w formie prezentów dla innych użytkowników.

Jak pokazano na rysunku 1.10, w serwisie Groupon udało się wprost doskonale zaimplementować ten mechanizm. Nawet jeśli oferowany produkt nie interesuje bieżącego użytkownika, być może ten użytkownik zna kogoś, kto będzie zainteresowany tym produktem. Użytkownik może też dojść do wniosku, że oferowany produkt sprawdzi się w roli prezentu przy okazji jakiejś uroczystości czy imprezy. Samo udostępnienie użytkownikom alternatywnej opcji korzystania ze sprzedawanych produktów czy usług może znacznie podnieść sprzedaż.

Rysunek 1.10. Opcja prezentu („Kup na prezent!") w serwisie Groupon

Studium przypadku: aplikacje uwzględniające położenie użytkownika

W tym studium przypadku zostanie omówiony rodzaj aplikacji, który w ostatnich latach zyskał ogromną popularność: aplikacje uwzględniające położenie użytkownika.

Takie aplikacje jak Gowalla, FourSquare czy nawet Facebook Places doskonale ilustrują aktualne możliwości usług tego typu. W tym studium przypadku wymienione usługi będą stanowiły punkt wyjścia dla analizy aplikacji uwzględniających położenie użytkowników, w tym technik pozyskiwania nowych użytkowników, zwiększania liczby aktywnych użytkowników oraz oferowania komercyjnych usług dostosowanych do preferencji użytkowników.

Spotykanie przyjaciół

Jedną z największych zalet aplikacji uwzględniających położenie (geolokalizację) użytkowników jest dostęp do informacji o miejscu przebywania znajomych i przyjaciół. Od czasu do czasu każdy zadaje sobie pytanie: „ciekawe, co teraz robi mój przyjaciel?" lub „ciekawe, czy któryś z moich przyjaciół jest już w tym miejscu i czy mógłbym do niego dołączyć?". Aplikacje uwzględniające położenie użytkowników pozwalają błyskawicznie odpowiadać na podobne pytania. Funkcje śledzenia znajomych i analizy rzeczywistych miejsc, z których logują się w serwisie, dodatkowo podnosi społecznościową wartość aplikacji.

Należy brać pod uwagę, że znajomy użytkownika może już przebywać w innym miejscu niż to wskazywane przez aplikację (czyli miejsce ostatnio potwierdzone przez tego znajomego w usłudze). O ile usługi tego typu oferują opcje wskazywania miejsca przebywania, nie udostępniają opcji anulowania tego miejsca.

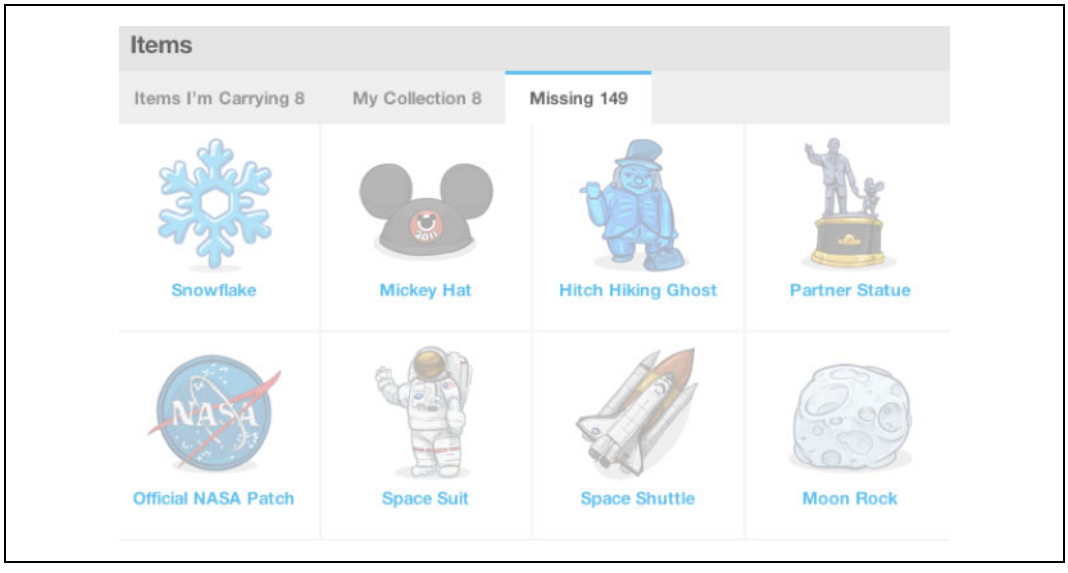
Mechanizmy interakcji użytkowników ze znajomymi według miejsca przebywania umożliwiają korzystanie z usług na wiele różnych sposobów (często wykraczających poza początkowe założenia twórców aplikacji). Te dodatkowe możliwości motywują użytkowników do znajdowania innowacyjnych metod, które w jeszcze większym stopniu będą odpowiadały ich potrzebom — będą więc spędzali więcej czasu z aplikacją.

Udostępnianie odznak i punktów

Jedną z metod zwiększania aktywności użytkownika aplikacji uwzględniającej położenie jest przyznawanie wirtualnych nagród za poszczególne czynności przy użyciu tej aplikacji. W ten sposób można utrzymać zaangażowanie użytkowników produktu przy minimalnych nakładach pracy programistów, którzy implementują to rozwiązanie.

Na przykład aplikacja Gowalla oferuje funkcję nadającą użytkownikom odznaczenia zależnie od miejsc, które odwiedzili i z których logowali się w serwisie. Każde nowe miejsce odwiedzane przez użytkownika jest nagradzane odznaką potwierdzającą obecność w danym punkcie. Użytkownik może też uzyskać wiele odznak jednocześnie, jeśli zgłosi swój pobyt w nowym stanie czy kraju. Od czasu do czasu wybrani użytkownicy mogą zdobywać dodatkowe, wirtualne nagrody (przy okazji logowania w nowych miejscach), które trafiają na ich oficjalne kolekcje.

Serwis Gowalla wyświetla także elementy pozostałe do zdobycia (patrz rysunek 1.11). Obawa przed utratą tych elementów skłania użytkowników do logowania się w wielu dodatkowych miejscach w nadziei na zachowanie dotychczasowych i zdobycie nowych odznaczeń.



Rysunek 1.11. Wykaz brakujących elementów w serwisie Gowalla

W przeciwieństwie do aplikacji gier społecznościowych aplikacje uwzględniające położenie użytkownika zwykle wykorzystują wirtualne towary raczej do zwiększania zaangażowania użytkowników niż do bezpośredniego zwiększania przychodów.

Oferowanie współzawodnictwa (tablice wyników)

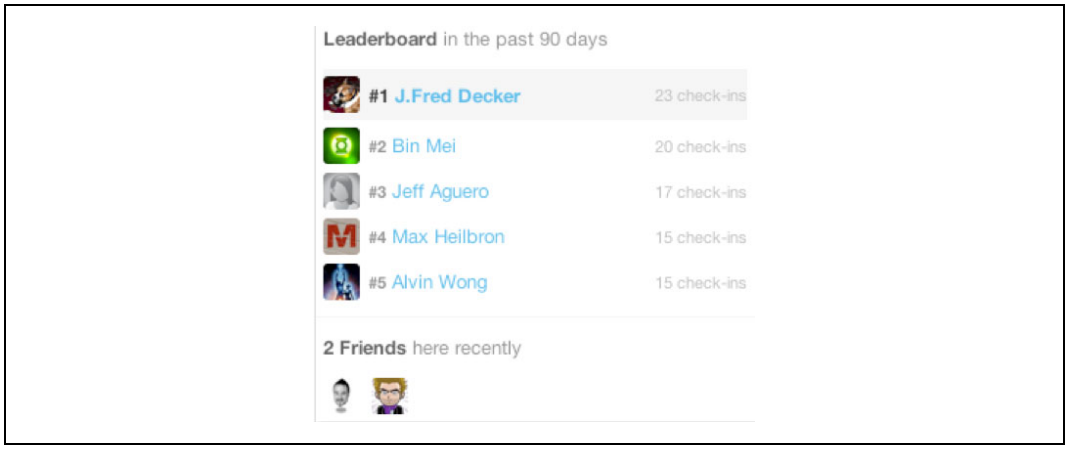
Czy jest coś złego w zdrowej rywalizacji? W przypadku aplikacji uwzględniających położenie użytkowników element współzawodnictwa jest wręcz pożądany.

Aplikacja Gowalla wprowadziła tablicę wyników (patrz rysunek 1.12), czyli listę użytkowników, którzy potwierdzili swój pobyt w określonym miejscu najwięcej razy. Podobnie w serwisie FourSquare jest na bieżąco budowana lista tzw. burmistrzów — najlepszy użytkownik otrzymuje tytuł burmistrza określonego miejsca.

Rozwiązania tego typu zachęcają użytkowników do częstszego korzystania z aplikacji i prób uzyskania najwyższych pozycji dla określonych miejsc (warto jednak mieć na uwadze ryzyko fałszywych zgłoszeń od użytkowników chcących szybko wejść na szczyt rankingu). Funkcja współzawodnictwa z innymi użytkownikami poprzez samo korzystanie z usługi jest jednym z najskuteczniejszych sposobów utrzymywania zaangażowania użytkowników.

Kierowanie reklam według lokalizacji i profilu

Jaka jest największa wartość informacji o miejscu, w którym użytkownik korzysta z danej usługi? Wiedza o tym, co otacza tego użytkownika. Połączenie tej wiedzy ze zgromadzonymi informacjami na temat nawyków użytkownika jest wprost doskonałym źródłem informacji o tym, jak przygotować treść maksymalnie dostosowaną do gustu i zainteresowań odbiorcy.



Rysunek 1.12. Tablica wyników w serwisie Gowalla

Przypuśćmy, że dysponujemy własną aplikacją umożliwiającą zgłaszanie miejsca pobytu przez użytkowników (jak w przypadku serwisów Gowalla i FourSquare) i że śledzimy historię miejsc, w których przebywał dany użytkownik. Na podstawie tych miejsc można spróbować określić zainteresowania użytkownika, na przykład ulubione potrawy (zależnie od odwiedzanych restauracji), ogólne przyzwyczajenia (według częstotliwości odwiedzania określonych miejsc) oraz godziny spożywania posiłków (zależnie od czasu przebywania w poszczególnych punktach). Połączenie wszystkich tych informacji pozwala prezentować użytkownikowi reklamy lokalnych firm posiadających ofertę najlepiej pasującą do tych kryteriów lub działających w tych samych branżach co firmy, z których usług już teraz korzysta użytkownik. Prezentowane reklamy można też zmieniać zależnie do czasu logowania użytkownika w serwisie.

System reklam można dodatkowo rozbudować. Skoro użytkownik korzysta z aplikacji uwzględniającej jego położenie, wiadomo, gdzie przebywa i jakie otoczenie lubi najbardziej. Jeśli użytkownik szuka restauracji, aplikacja może mu zasugerować lokale dopasowane do jego preferencji i położone w pobliżu miejsc, z których często logował się w systemie.

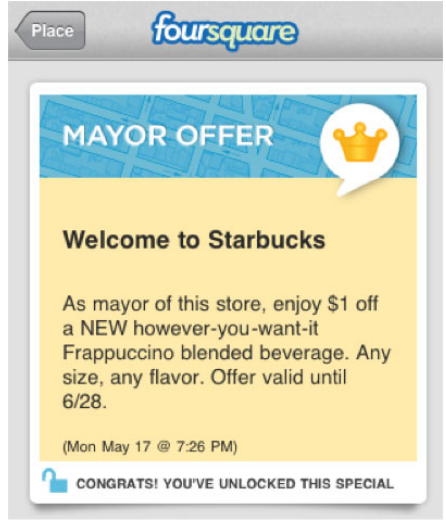
Mimo że przytoczony przykład jest bardzo konkretny, można bez trudu wskazać mnóstwo innych scenariuszy, w których miejsce przebywania użytkownika i historia odwiedzanych lokalizacji ułatwia przygotowywanie treści (w tym reklam) specjalnie z myślą o preferencjach odbiorcy. Dostosowanie reklam i sugestii do indywidualnych nawyków użytkownika znacznie zwiększa prawdopodobieństwo skorzystania z proponowanej oferty. Współpraca z lokalnymi firmami i reklamodawcami jest wyjątkowo efektywną strategią generowania sugestii uwzględniających przyzwyczajenia poszczególnych użytkowników.

Oferowanie promocji we współpracy z lokalnymi firmami

Warto teraz przeanalizować nieco inną formę komercyjnego wykorzystywania aplikacji geolokalizacyjnych — partnerstwo z lokalnymi firmami.

Aplikacja uwzględniająca fizyczne położenie użytkownika w rzeczywistym świecie stwarza wprost doskonałe możliwości budowania partnerstwa z lokalnymi firmami, które mogą oferować rabaty i zniżki użytkownikom aplikacji — taka współpraca jest korzystna zarówno dla właściciela aplikacji, jak i dla firm partnerskich.

Warto w tym kontekście przeanalizować przykład serwisu FourSquare. W maju 2010 roku firma nawiązała współpracę z siecią kawiarni Starbucks — użytkownicy serwisu, którzy najczęściej logowali się w lokalnych kawiarniach, otrzymywali atrakcyjne rabaty na jeden z nowych napojów (patrz rysunek 1.13). Z perspektywy firmy Starbucks takie rozwiązanie było doskonałą promocją nowego napoju wśród dużej grupy użytkowników, a serwis FourSquare skorzystał na tzw. promocji „lubię to”[1], zwiększonej aktywności użytkowników w określonych miejscach i budowie wizerunku serwisu, który nie służy tylko do zabawy i gier, ale może też przynieść użytkownikowi wymierne korzyści.



Rysunek 1.13. Oferta dla klientów sieci Starbucks i użytkowników serwisu FourSquare

Aplikacje geolokalizacyjne mogą oferować swoim użytkownikom korzyści zarówno w świecie wirtualnym, jak i w świecie rzeczywistym. Takie połączenie pokazuje użytkownikom, że ich aktywność w internecie może mieć ścisły związek z funkcjonowaniem w rzeczywistym świecie.

Krótkie wskazówki na początek

Budowanie aplikacji jest niełatwą sztuką, szczególnie jeśli zespół programistów jest poddawany presji szybkiego zwrotu z inwestycji. Przed przystąpieniem do właściwego procesu konstruowania aplikacji należy odpowiedzieć sobie na kilka praktycznych pytań dotyczących różnych aspektów funkcjonowania przyszłego produktu.

[1] Termin **promocja „lubię to”** (ang. *like promotion*) pochodzi od przycisku *Lubię to* dostępnego w serwisie Facebook. Przycisk umożliwia użytkownikowi przeglądającemu stronę lub produkt wyrażenie pozytywnej oceny i — tym samym — promowanie tej strony czy produktu wśród innych użytkowników tego serwisu społecznościowego.

Należy zdefiniować docelowych odbiorców

Najważniejsze pytanie, na które powinna odpowiedzieć sobie każda firma, brzmi: do kogo będzie kierowana oferta? To samo pytanie muszą stawiać sobie programiści aplikacji społecznościowych — twórcy aplikacji muszą zdefiniować i zrozumieć potrzeby użytkowników, których będą próbowali przekonać do swojego produktu. Dopiero po zidentyfikowaniu tej grupy można przystąpić do projektowania struktury mechanizmów odkrywania aplikacji, rozmieszczenia galerii zdjęć i obrazów oraz metod promocji.

Możliwie wczesne budowanie punktów integracji z serwisem społecznościowym

Jedną z pułapek, w które najczęściej wpadają twórcy aplikacji działających w ramach kontenerów społecznościowych, jest lekceważenie różnic dzielących te aplikacje od tradycyjnych, zamkniętych rozwiązań. Budowa prostego serwisu, który nie jest w żaden sposób związany z funkcjami społecznościowymi kontenera, nie obejmuje mechanizmów promocji przy użyciu strumieni aktywności użytkowników ani nie operuje na grafie powiązań społecznościowych, z pewnością nie wystarczy. Wymienione elementy społecznościowe mają zasadniczy wpływ na czas życia aplikacji i dynamikę liczby użytkowników.

Podczas planowania przyszłego działania aplikacji należy uwzględnić rolę informacji społecznościowych w ogólnej strukturze serwisu. W szczególności należy dokładnie przeanalizować możliwości promowania aplikacji według czynności podejmowanych przez użytkowników, wykorzystania grafu powiązań społecznościowych do promowania aplikacji wśród nowych użytkowników i użycia informacji zawartych w profilu do personalizacji oferowanych funkcji.

Budowanie z myślą o elementach komercyjnych

Kolejnym błędem popełnianym przez programistów jest brak projektu komercyjnego wykorzystania nawet najlepszych pomysłów i koncepcji. Już na etapie tworzenia aplikacji należy mieć na uwadze, jak docelowo mają one zarabiać pieniądze. Jeśli źródłem zarobku mają być reklamy, koniecznie należy zaplanować odpowiednią przestrzeń dla tych reklam.

Jeśli to nie reklamy mają być źródłem zysku, należy rozważyć wiele alternatywnych metod komercyjnego wykorzystania aplikacji. Dodatkowa treść wzbogacająca doznania użytkowników może okazać się świetnym sposobem poprawiania zadowolenia i jednocześnie może zapewniać użytkownikom swobodę w decydowaniu o sposobach korzystania z aplikacji.

Inną efektywną metodą jest zintegrowanie elementów komercyjnych z właściwymi funkcjami aplikacji. Na przykład niektórzy producenci gier oferują (za niewielką opłatą) dodatkowe punkty rozwoju, dzięki którym gracze mogą przyspieszyć produkcję, osiągnąć lepsze statystyki lub dodawać rozmaite usprawnienia. Większość graczy ograniczy się do korzystania z darmowych punktów, które można gromadzić w trakcie rozgrywki, jednak jakaś część zdecyduje się na zakup dodatkowych punktów, aby zyskać przewagę nad konkurentami.

Niezależnie od wybranej metody bardzo ważne jest przygotowanie strategii komercyjnego wykorzystania aplikacji jeszcze przed przystąpieniem do budowy samego oprogramowania.

Tworzenie dopracowanych, atrakcyjnych widoków

W tym rozdziale wspomniałem o znaczeniu widoków podczas budowy aplikacji społecznościowych. Należy maksymalnie wykorzystać wszystkie widoki dostępne w danym kontenerze aplikacji społecznościowych. Nie wystarczy umieszczenie całej treści we wszystkich widokach — lepszym rozwiązaniem jest stosowanie widoków, które będą się wzajemnie uzupełniały.

Na przykład widok kanwy może służyć do konfigurowania treści prezentowanej w pozostałych widokach. Oferowanie dynamicznych widoków, których zawartość może być zmieniana przez samego użytkownika, jest jednym z najlepszych sposobów budowania bazy aktywnych użytkowników aplikacji.