Złożenie pracy online:
2013-06-17 20:47:24
Kod pracy:
9656
Kod załącznika:
9644

Albert Wajda
(nr albumu: 18982*INF/INŻ)

Praca inżynierska

Mobilny system do obsługi zamówień w restauracjach

Mobile system that supports food ordering in restaurants

Wydział: Nauk Społecznych i Informatyki
Kierunek: Informatyka
Specjalność:
Promotor: dr Henryk Telega

Nowy Sącz, 2013

SPIS TREŚCI

WSTĘP

Wprowadzenie

Zamawianie towarów i usług (w szczególności jedzenia) przez telefon to czynność, która od lat nie zmieniła swojej formy. Większość restauracji udostępnia swoje menu w formie elektronicznej, które czasami jest także dostępne on-line. Czy to jednak to nie za mało jak na dzisiejsze czasy? W dobie telefonów komórkowych wyposażonych w odbiorniki GPS, łączność bezprzewodową (modemy, karty sieciowe), działających coraz częściej w oparciu o system operacyjny, umożliwiający instalowanie aplikacji, ogromnym przeoczeniem wydaje się być niedostosowanie tego procesu do możliwości technologicznych.

Większość danych wymagana do złożenia zamówienia jest przechowywana w pamięci urządzenia, m.in. numer telefonu osoby zamawiającej, dane osobowe użytkownika (osoby składającej zamówienie), a nawet adres dostawy zamówienia. Wykorzystanie urządzenia jako dostawcy części danych oznacza oszczędność czasu zarówno użytkownika, jak i drugiej strony czyli w tym przypadku restauracji - odebranie zamówienia następuje natychmiast po jego złożeniu, pracownik nie musi poświęcać swojego czasu na telefoniczne odbieranie zamówień.

Cel i zakres pracy

Celem pracy jest:

- zaprojektowanie i utworzenie aplikacji mobilnej dla systemu Android
- zaprojektowanie interfejsu serwera dla aplikacji
- zapewnienie stabilnej komunikacji pomiędzy aplikacją mobilną a serwerem

Aplikacja współpracować będzie z serwerem, który zostanie wykonany w ramach innej pracy inżynierskiej. Umożliwiać będzie składanie zamówień w dowolnej restauracji dodanej do bazy danych. Podczas pierwszego uruchamiania pobierze ona dane użytkownika zapisane w systemie Android, które posłużą do indentyfikacji klienta po stronie serwera.

Po uruchomieniu aplikacja ustali aktualną lokalizację użytkownika, a na jej podstawie dokona filtracji danych z serwera (ang. location-aware application). Każda restauracja ma ustaloną odgórnie maksymalną odległość dowozu zamówienia - dlatego aplikacja obliczać będzie odległość użytkownika od restauracji, tak aby zaprezentować mu tylko te, w zasięgu których się znajdzie. Dane o restauracjach i ich menu pobierane będą ze specjalnie przygotowanego serwera, który zapewniać będzie szybkość przesyłu danych i prostotę komunikacji, m.in. dzięki zastosowaniu wzorca architektury REST.

Po filtracji danych użytkownik będzie miał możliwość rozpoczęcia składania zamówienia w dowolnie wybranej przez siebie restauracji. Dodatkowo będzie miał także możliwość uzyskania trasy dojazdu do niej z aktualnego miejsca, dzięki wykorzystaniu funkcjonalności Google Maps. Dane o lokalizacji restauracji po stronie serwera przechowywane będą w formie współrzędnych geograficznych, a adres będzie uzyskiwany z wykorzystaniem Google Geocoder'a - narzędzia umożliwiającego odwrotne geokodowanie - czyli zamianę tych danych na dokładny adres (ulicę, miasto, kod pocztowy, a także nr lokalu).

Proces składania zamówienia będzie prosty i intuicyjny - menu podzielone będzie na kategorie, do których przypisane zostaną konkretne produkty/posiłki/dania wraz ze zdjęciem i opisem. Przeglądanie menu bazować będzie na gestach (swipe gesture) i obsłudze maksymalnie dwóch przycisków. Ceny produktów nie będą pobierane razem z menu ze względu na ograniczenie ilości przesyłanych danych (jeden produkt może mieć wiele cen w zależności od rozmiaru). W momencie gdy użytkownik zainteresowany będzie wybranym produktem, jego cena zostanie pobrana z serwera. Po pobraniu cen, użytkownik będzie miał możliwość dodania produktu do zamówienia (dowolnej ilości). W każdej chwili będzie miał także możliwość podglądu swojego zamówienia, zmiany ilości poszczególnych produktów lub ich usunięcia.

Zamówienie dodatkowo będzie zawierało dane identyfikacyjne użytkownika – numer telefonu, adres email (który zostanie automatycznie pobrany przez aplikację z systemu Android) oraz współrzędne geograficzne pobrane z modułu GPS, określające miejsce dostarczenia zamówienia. Indetyfikacja użytkowników po stronie serwera będzie się opierać na adresie email, ponieważ aby używać systemu Android należy zalogować się do niego poprzez konto email – oznacza to, że adres nie wymaga już dodatkowej weryfikacji, gdyż posłużył do uruchomienia urządzenia. Wyklucza to konieczność rejestracji i logowania użytkownika, a to z kolei skraca czas realizacji zamówienia.

Zakres prac:

- zaprojektowanie przejrzystego i spójnego interfejsu użytkownika
- zaprojektowanie i implementacja bazy danych SQLite (tabele muszą mieć identyczną strukturę jak po stronie serwera)
- wykorzystanie klas frameworku Spring (RestTemplate) dla uproszczenia obsługi wymiany danych z serwerem zbudowanym w architekturze REST
- impementacja własnych interfejsów do obsługi wywołań zwrotnych (ang. callback)
- wykorzystanie narzędzia Google Geocoder do procesu odwrotnego geokodowania
- wykorzystanie klas LocationListener'a wchodzących w skład systemu Android do obłsugi modułu GPS jako części funkcjonalności aplikacji
- wykorzystanie biblioteki Sherlock ActionBar dla zapewnienia kompatybilności aplikacji ze starszymi wersjami systemu (poniżej Androida 3.0)
- wykorzystanie tzw. fragmentów (Fragments) do budowania ekranów użytkownika - wycinka części widoku, za którego obsługę odpowiada osobna klasa
- stworzenie nawigacji po menu bazującej na gestach (swipe gesture)
- zbudowanie mechanizmów służących do pobierania wybranych informacji z urządzenia (systemu) w celu identyfikacji użytkownika
- wykorzystanie biblioteki GSON do parsowania danych pobranych z serwera w formacie tekstowym JSON na obiekty własnego typu
- wykorzystanie silnika Jackson Mapper z biblioteki Spring do serializacji i deserializacji danych
- implementacja własnych klas umożliwiających zautomatyzowanie procesów pobierania, wysyłania oraz zapisywania danych (m.in. menedżer pobierania, bazy danych oraz aktualizacji)
- wykorzystanie SharedPreferences jako sposobu na przechowywanie prostych danych w postaci klucz-wartość takich jak np. ustawienia aplikacji
- stworzenie funkcjonalności pozwalającej na zapis danych w postaci plików w wewnętrznej pamięci urządzenia (np. zdjęcia produktów)

- stworzenie metod automatyzujących pomniejszanie obrazków pobranych z serwera, dopasowującej je do rozdzielczości ekranu urządzenia na którym zostaną pobrane

Serwer dla aplikacji został zaprojektowany i wykonany jako temat odrębnej pracy dyplomowej. Dane pomiędzy urządzeniami są przesyłane w formacie JSON, który zapewnia wysoką wydajność zarówno serwera jak i aplikacji mobilnej (zwłaszcza parsowanie danych). Serwer implementuje wzorzec architektury oprogramowania REST, charakteryzujący się m.in. własnym sposobem wywoływania usług i przesyłania parametrów.

Serwer został stworzony z wykorzystaniem silnika Google App Engine, oferującego także hosting aplikacji w chmurze. Zarządzanie danymi na serwerze odbywa się poprzez dedykowany, webowy system zarządzania treścią (CMS).

Zaprojektowanie serwera jest ściśle powiązane ze sposobem implementacji aplikacji mobilnej, dlatego też w ramach tej pracy dodatkowo należy:

- zaprojektować strukturę danych przechowywanych w klasach wspólnych dla serwera i aplikacji mobilnej
- zaprojektować interfejs serwera, aby aplikacja mogła komunikować się z serwerem (pobierać i wysyłać dane zgodnie z przeznaczeniem)

ROZDZIAŁ 1. TECHNOLOGIE UŻYTE W PRACY

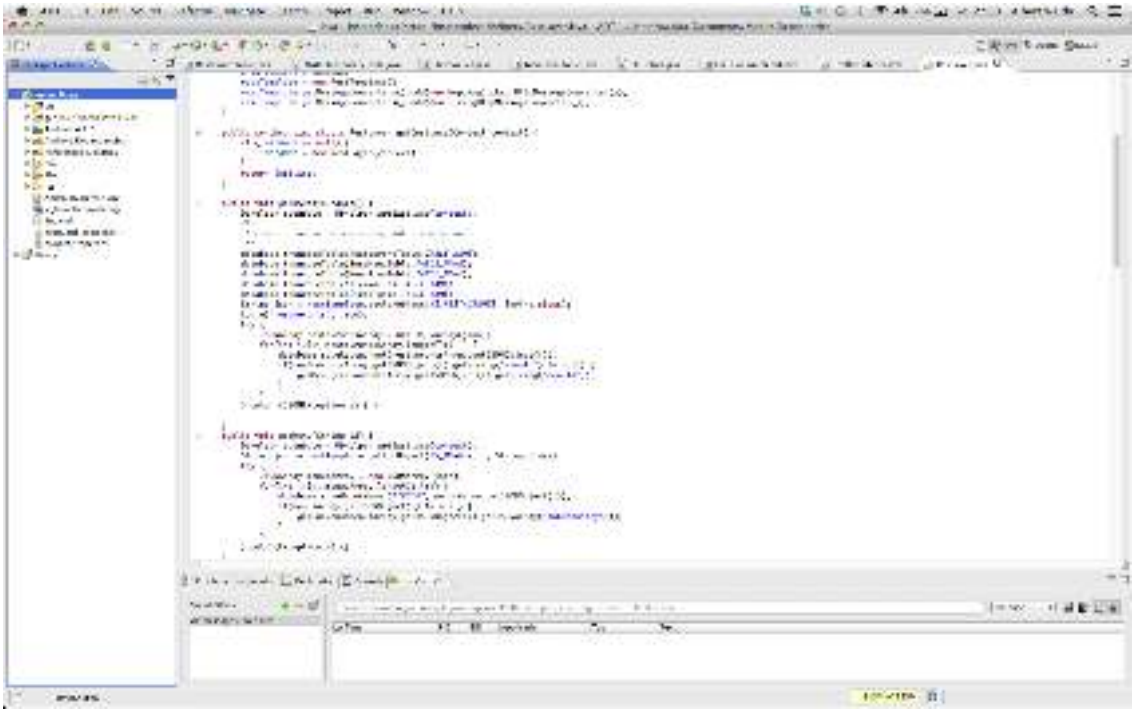
1.1. Język programowania Java

Java to obiektowy język programowania stworzony przez grupę roboczą pod kierunkiem Jamesa Goslinga z firmy Sun Microsystems. Java jest językiem tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną.

Wybór Javy przez Google jako języka do tworzenia aplikacji dla systemu Android może mieć wiele powodów. Niewątpliwą zaletą jest fakt, że dzięki użyciu Javy nie zachodzi potrzeba ponownej kompilacji kodu natywnego (tutaj będzie wyjaśnienie co to jest kod natywny) dla różnych platform sprzętowych, a pamiętać należy, że Android działa na wielu, różniących się od siebie platformach. Dodatkowo, Java ma silnie wsparcie ze strony społeczności open-source, co oznacza ogromną liczbę bibliotek i narzędzi, pozwalających nie tylko skrócić czas pracy nad aplikacją, ale także zapewnić stabilność dzięki wykorzystaniu sprawdzonych rozwiązań.

Java to także bezpieczeństwo - Android pozwala na uruchamianie niepodpisanych cyfrowo aplikacji (niepochodzących bezpośrednio ze sklepu Google Play), które skutecznie mogłyby zniszczyć system lub mieć znaczący wpływ na jego działanie. Wykonywanie kodu aplikacji w wirtualnej maszynie Javy gwarantuje, że odbędzie się to bez wpływu na jądro systemu operacyjnego.

1.2. Środowisko programistyczne Android Developer Tools



Główne okno projektu w Android Developer Tools – po lewej stronie znajduje się eksplorator plików w katalogu z projektem, w prawej części zawartość otwartych plików.

Jest to zestaw narzędzi programistycznych do tworzenia aplikacji dla systemu Android przygotowany do pobrania w formie wtyczki, dla zintegrowanego środowiska programistycznego Eclipse.

Rozszerza ona możliwości Eclipse o funkcje związane z tworzeniem aplikacji Android, interfejsów użytkownika, debugowania oraz eksportowania pakietów aplikacyjnych gotowych do użytku.

Narzędzia niezbędne do tworzenia aplikacji dla systemu Android są zintegrowane ze środowiskiem Eclipse i dostępne z poziomu menu, co znacznie ułatwia pracę. Programista nie musi także przełączać się w tryb wiersza poleceń aby wykonać podstawowe czynności, jak np. instalacja i uruchomienie aplikacji w emulatorze.

Zaprojektowanie Android Developer Tools jako wtyczki do Eclipse jest sprytnym rozwiązaniem ze strony Google. Większość programistów Java miała do czynienia w przeszłości z tym środowiskiem, więc nabycie wprawy w obsłudze narzędzia nie powinno zająć dużo czasu. Ponadto jest gwarantem, że programista otrzyma funkcjonalność stabilnego środowiska programistycznego.

1.3. Dalvik Debug Monitor Server

Android Developer Tools wyposażone zostało w Dalvik Debug Monitor Server (DDMS) - narzędzie do analizy i kontroli urządzeń aktualnie podłączonych do komputera. Umożliwia ono podgląd w czasie rzeczywistym:

- uruchomionych aplikacji
- działających procesów
- alokacji pamięci
- statystyk dotyczących wykorzystania sieci bezprzewodowych
- przeglądarkę plików zapisanych w pamięci urządzenia i na karcie pamięci

Dodatkowo narzędzie ma możliwośc kontroli emulatora (Android Virtual Device, którego opis znajduje się w kolejnym podrozdziale). Pozwala na wygenerowanie sztucznego zachowania urządzenia dla następujących sytuacji:

- połączenia nadchodzącego
- nadchodzącej wiadomości tekstowej (SMS)
- otrzymania pakietu lokalizacji urządzenia z odbiornika GPS

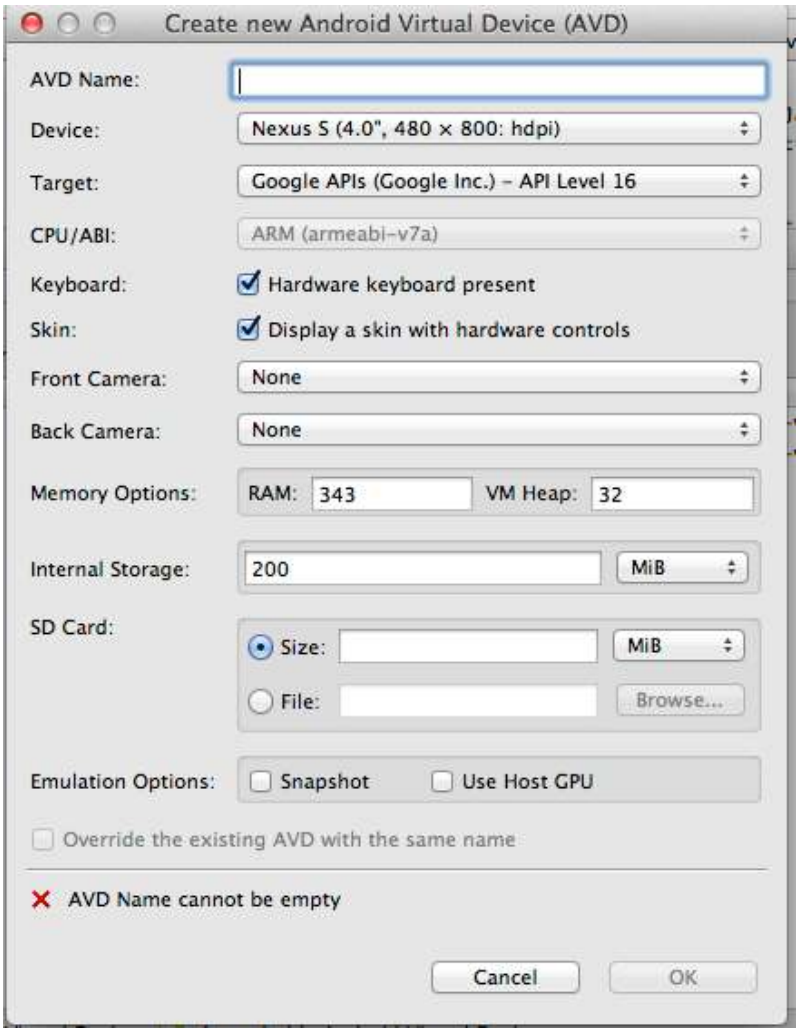
1.4. Android Virtual Device

Android Virtual Device – podgląd aplikacji w emulowanym urządzeniu.

Android Virtual Device to nic innego jak emulator urządzeń z systemem Android. Umożliwia uruchamianie aplikacji z poziomu Android Developer Tools jak również z wiersza poleceń. Narzędzie to wchodzi w skład Android SDK .

Programista ma możliwość pobrania obrazu dowolnej wersji systemu Android, która została oficjalnie opublikowana przez Google. Poza wyborem systemu, należy także stworzyć konfigurację sprzętową, na którą składa się:

- rozdzielczość i gęstość wyświetlacza
- procesor
- urządzenie odpowiedzialne za dostarczenie obrazu do przedniej i tylnej kamery
- ilość pamięci RAM
- ilość pamięci wewnętrznej urządzenia
- zewnętrzna karta pamięci która ma być zainstalowana w urządzeniu

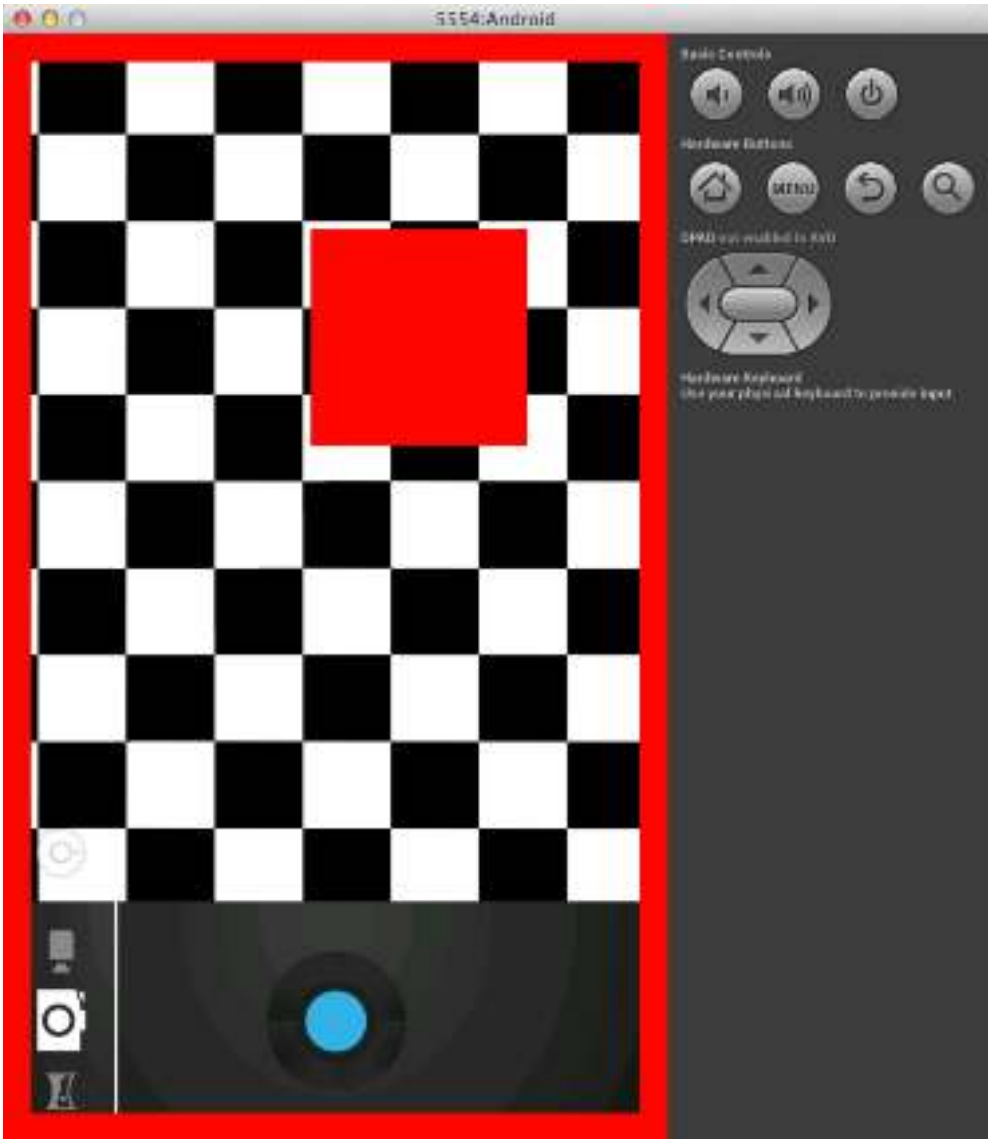


Okno kreatora nowego urządzenia w Android Virtual Device.

Android Virtual Device domyślnie oferuje 19 gotowych konfiguracji sprzętowych, przeważnie różniących się konfiguracją wyświetlacza (gęstością i rozdzielczością). Jest to szczególnie przydatne podczas testów aplikacji, chcąc zapewnić wsparcie dla jak największej ilości urządzeń. Emulator umożliwia uruchomienie i pracę na kilku wirtualnych urządzeniach jednocześnie.

Android Virtual Device pozwala na stworzenie dowolnej ilości własnych konfiguracji urządzeń, dzięki czemu istnieje możliwość dokładnego przetestowania aplikacji pod wybranymi wersjami systemu, bez konieczności posiadania fizycznych urządzeń.

Niestety, narzędzie nie zastępuje całkowicie fizycznego urządzenia, ponieważ część funkcjonalności nie działa - jak na przykład symulowanie połączeń wychodzących. Również, mimo możliwości wyboru urządzenia odpowiedzialnego za dostarczenie obrazu do przedniej oraz tylnej kamery, podczas testowania aplikacji wykorzystującej obraz z kamer, okazuje się, że podgląd ten nie jest dostępny. Wygląda to na błąd ze strony Google.



Brak podglądu obrazu z kamery w emulowanym urządzeniu mimo poprawnej konfiguracji w kreatorze.

Niewątpliwie przydatną funkcją okazuje się emulacja sygnału GPS. Opcja ta jest nieco ograniczona, ale zdecydowanie ułatwia testowanie aplikacji bazującej na usługach lokalizacji. Pozwala zaoszczędzić sporo czasu, ponieważ istnieje możliwość zaimportowania wyznaczonej wcześniej za pomocą Google Maps trasy, która to będzie przesyłana do urządzenia w postaci kolejnych punktów lokalizacyjnych, w zadanym interwale czasowym. Ograniczeniem natomiast jest brak możliwości przesyłania do urządzenia informacji o jakości sygnału GPS, którą to większość uwzględnia podczas swojego działania (przeważnie aby wyeliminować przekłamania powstające podczas bazowania na słabym sygnale). Tak więc do testowania funkcjonalności opartych

o lokalizację, niezbędne czasem okazuje się przygotowanie takiej wersji aplikacji, aby nie sprawdzała jakości sygnału.

1.5. Logcat

Okno konsoli Logcat'a wyświetlające komunikaty różnego rodzaju.

Logcat jest narzędziem zbierającym i wyświetlającym informacje wyjściowe z systemu, pochodzące z aktualnie działających aplikacji lub procesów. Programista, korzystając ze specjalnej klasy Log, ma możliwość wypisania dowolnej informacji w postaci tekstowej na konsoli Logcat'a, określając dodatkowo jej rodzaj (dla każdego przewidziany jest inny kolor tekstu, co pomaga w skanowaniu konsoli wzrokiem w poszukiwaniu np. błędów).

Sprawne używanie tego narzędzia pozwala na dokładne zrozumienie kodu programu, aktualnego stanu urządzenia, czy też wyświetlania informacji o aktualnie występujących błędach bez konieczności przerywania wykonania programu (np. dzięki wykorzystaniu bloku try-catch).

Konsola umożliwia filtrowanie informacji nie tylko wg rodzaju (np. wyświetlanie wyłącznie błędów), ale także aplikacji z której pochodzą.

Często popełniane błędy, jak np. brak deklaracji odpowiednich uprawnień i późniejsza próba wykorzystania ich w kodzie, skutkuje wymuszonym zamknięciem aplikacji na urządzeniu. W pierwszej chwili nie do końca może być zrozumiałe, dlaczego poprawnie składniowo napisany kod powoduje błąd. Google jednak doskonale wykorzystał Logcat'a, jako miejsce do wyświetlania podpowiedzi dla programisty o możliwych przyczynach błędu.

Przykład braku deklaracji uprawienia aplikacji do wykorzystania lokalizacji i podgląd z Logcata informujący o możliwej przyczynie błędu.

1.6. SQLite Manager

System Android dostarcza dwa mechanizmy przechowywania danych (poza plikami), z których każdy ma inne zastosowanie.

Do przechowywania par klucz-wartość, czyli niewielkiej ilości danych zostały wprowadzone tzw. Shared Preferences. Są one wykorzystywane głównie do przechowywania ustawień aplikacji.

Z kolei do przechowywania dużej ilości uporządkowanych danych wprowadzono bazy danych SQLite. Biblioteka implementuje silnik SQL (ang. Structured Query Language). Zawartość bazy przechowywana jest w postaci pliku binarnego. Przeglądanie zawartości bazy danych na urządzeniu bez nadania praw administratora systemu (root) jest niemożliwe. Ułatwieniem jest dostęp do pliku bazy danych na wirutalnym urządzeniu, dzięki czemu istnieje możliwość sprawdzenia struktury i zawartości zewnętrznym programem.

1.7. Wzorzec architektury REST

W celu uproszczenia interfejsu serwera i komunikacji pomiędzy aplikacją a serwerem, zaimplementowano wzorzec architektury oprogramowania REST. Charakteryzuje go inny sposób umieszczania parametrów wywołania, w ścieżce adresu URL:

Wywołanie klasyczne:

```
http://instant-pizza.appspot.com/menu?id=1234&order=alfa
```

Wywołanie RESTful:

```
http://instant-pizza.appspot.com/menu/1234/alfa
```

ROZDZIAŁ 2. IMPLEMENTACJA WAŻNYCH CZĘŚCI APLIKACJI

2.1. Aktywności w systemie Android

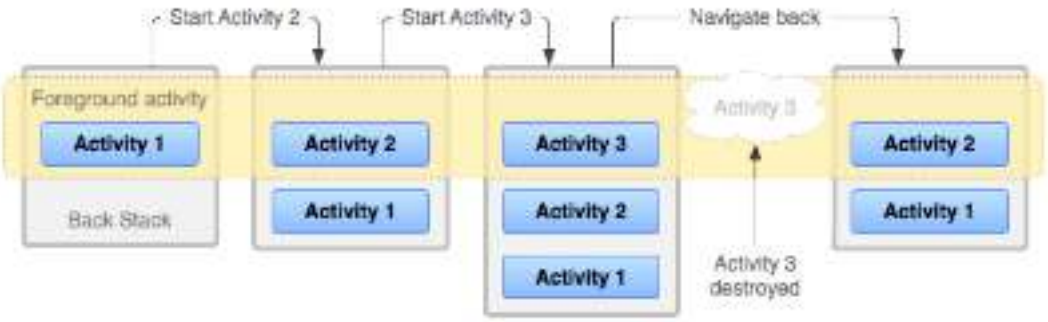
Aktywność (ang. Acitivity) jest zasadniczą jednostką pracy w systemie Android. Można ją porównać do pojęcia procesu istniejącego w systemie desktopowym, jednak z małym zastrzeżeniem - mianowicie przyjęto, iż jedna aktywność powinna odpowiadać za obsługę maksymalnie jednego widoku w aplikacji. Nie jest to jednak odgórnym wymogiem, bowiem istnieje możliwość implementacji aktywności obsługującej wszystkie widoki aplikacji.

Każda aplikacja może posiadać wiele aktywności. W uruchamianiu innych aktywności z pomocą przychodzą intencje/zamiary (ang. Intent). Wartym podkreślenia jest fakt, że platfroma pozwala na uruchamianie aktywności także z innych aplikacji (np. pozycja menu przenosi do profilu użytkownika z wykorzystaniem aplikacji Facebook).

Informacja o każdej aktywności musi znaleźć się w pliku manifestu aplikacji, gdzie należy także wskazać główną aktywność, czyli uruchamianą domyślnie (najczęściej po wyborze aplikacji z menu).

Organizacja aktywności w systemie odbywa się za pomocą stosu aktywności. Nowo uruchamiana aktywność jest dokładana na wierzch stosu, a poprzednia jest zatrzymywana. Zatrzymanie aktywności oznacza, że system zachowuje stan jej interfejsu użytkownika, po to aby w momencie użycia przycisku „wstecz“ przez użytkownika, aktywność która jest na szczycie stosu mogła zostać z niego zdjęta (aktywność zostaje „zniszczona“ (ang. destroyed)), a poprzednia aktywność zostaje wznowiona (stan interfejsu użytkownika zostaje przywrócony)[1].

[1] http://developer.android.com/reference/android/app/Activity.html



Prezentacja procesu tworzenia nowych aktywności i odkładania ich na stos. W momencie użycia przycisku „wstecz" przez użytkownika, aktualna aktywność jest "niszczona" (ang. destroyed), a oprzednia wznawiana.

2.2. Wywołania zwrotne

Wywołania zwrotne (ang. callback) są techniką programowania stanowiącą odwrotność wywołania funkcji. Użytkownik rejestruje jedynie funkcję, która ma być później wywołana natomiast jej wykonanie następuje w stosownym dla siebie czasie. Rejestrowanie funkcji polega na stworzeniu własnej klasy i zdefiniowaniu w niej odpowiedniej metody:

```
public interface OnTaskCompleted {
   public abstract void onTaskCompleted();
}
```

Na potrzeby stworzenia aplikacji zaimplementowano menedżer pobierania korzystający z wywołań zwrotnych. Aktywność która korzysta z menedżera pobierania, przekazuje jako argument referencję do siebie samej, a dzięki temu że implementuje interfejs OnTaskCompleted, który jest z kolei tzw. słuchaczem (ang. listener), menedżer będzie w stanie wywołać na nim odpowiednią metodę publiczną – onTaskCompleted().

Definicja konstruktora klasy UpdateManager:

```
public UpdateManager(Context context, int updateAction,
String itemId, String message, OnTaskCompleted
listener)
```

Po wykonaniu odpowiednich czynności (aktualizacji konkretnych danych), menedżer dokonuje wywołania zwrotnego na obiekcie typu OnTaskCompleted:

```
protected void onPostExecute(Void result) {
   (...)
   if (listener != null)
        listener.onTaskCompleted();
}
```

W sytuacji gdy dane nie zostały pobrane z serwera bądź są nieaktualne - nie następuje utworzenie widoku dla aktywności (np. listy restauracji). W momencie gdy menedżer aktualizacji zakończy swoją pracę, czyli dane zostaną pobrane i zapisane w lokalnej bazie danych, wówczas aktywność zostaje powiadomiona o możliwości utworzenia kompletnego widoku.

2.3. Menedżer aktualizacji

W projektowanej aplikacji zachodzi konieczność implementacji jednostki odpowiedzialnej za pobieranie najświeższych danych z serwera, dostępnej z poziomu aktywności. Android dostarcza model pojedynczego wątku, tworzonego podczas uruchomienia aplikacji - jest to tzw. UI Thread (User Interface Thread), inaczej zwany wątkiem głównym.

Oznacza to, że wszystkie procesy wykonywane w aplikacji realizowane są synchronicznie. Wykonanie jakiejkolwiek dłuższej operacji, jak np. połączenie

z serwerem, pobranie danych i zapisanie ich do bazy danych nie jest możliwe w wątku głównym, ponieważ brak odpowiedzi aplikacji przez co najmniej 5 sekund system traktuje jako błąd, wymuszając jej zamknięcie. Istnieje kilka możliwości rozwiązania tego problemu - użycie wątków (Threads) znanych z innych języków lub użycie mechanizmu zaimplementowanego w klasie AsyncTask. Ten drugi jest wręcz idealny do wykorzystania przez menedżer aktualizacji, ponieważ w trakcie pobierania danych wymagane jest wyświetlenie odpowiedniego komunikatu użytkownikowi, a klasa AsyncTask ma możliwość dokonywania modyfikacji w wątku interfejsu (czyli wyświetlania komunikatu, interakcji z użytkownikiem). Menedżer aktualizacji będzie więc rozszerzeniem klasy AsyncTask.

Dzięki wykorzystaniu wywołań zwrotnych, menadżer aktualizacji po pomyślnym pobraniu i zapisaniu żądanych danych, informuje swojego rejestratora o zakończeniu wykonywania operacji.

Definicja konstruktora klasy UpdateManager:

```
public UpdateManager(Context context, int updateAction,
String itemId, String message, OnTaskCompleted listener);
```

- context (*Context*)– kontekst aplikacji
- updateAction (*int*) – numer odpowiadający akcji zdefiniowanej w menadżerze (dopuszczalne wartości: 0 – aktualizacja restauracji, 1 – aktualizacja menu, 2 – aktualizacja podmenu, 3 – aktualizacja rozmiarów i cen wybranego produktu, 4 – aktualizacja historii zamówień). Wartości zdefiniowane są jako zmienne stałe,
- itemId (*int*) – (niewymagane) identyfikator zasobu do aktualizacji
- message (*String*) – treść wiadomości która zostanie wyświetlona użytkownikowi podczas aktualizacji
- listener (*OnTaskCompleted*) – obiekt odpowiedzialny za odebranie informacji o zakończeniu aktualizacji

Przykładowe wywołanie aktualizacji z poziomu aktywności:

```
new UpdateManager(this, UpdateManager.UPDATE_ORDERS, null,
"Pobieranie historii zamówień", this);
```

Efekt działania klasy UpdateManager widoczny dla użytkownika aplikacji

2.4. Menedżer pobierania

Jest to singleton, którego głównym zadaniem jest obsługa dwukierunkowej komunikacji z serwerem na którą składa się pobieranie (restauracji, menu i pozycji menu, zdjęć) oraz wysyłanie danych (zamówienia).

Do jego implementacji wykorzystano klasę RestTemplate [2], pochodzącej z biblioteki Spring[3]. Spring jest szkieletem tworzenia aplikacji w języku Java. Klasa RestTemplate zapewnia dostęp HTTP do danych po stronie klienta.

Obsługuje sześć metod HTTP:

- DELETE
- GET
- HEAD
- OPTIONS
- POST
- PUT

Pobrane dane zostają przekazane z menedżera do klasy zajmującej się obsługą bazy danych.

2.5. Menedżer zapisu/odczytu bazy danych

Projektowana aplikacja przechowuje pobrane z serwera dane w pamięci urządzenia. Android standardowo wyposażony jest w lekki i szybki system zarządzania bazą danych SQLite, który zostanie wykorzystany jako magazyn przechowywania danych tekstowych. Struktura danych po stronie serwera (udostępniania dla urządzeń komunikujących się z nim) ma swoje odzwierciedlenie w strukturze bazy danych, gdzie każdej klasie odpowiada jedna tabela.

Menedżer obsługujący zapis i odczyt danych z bazy danych zawiera publiczne metody wykorzystywane najczęściej przez menedżer pobierania.

[2] http://blog.springsource.org/2009/03/27/rest-in-spring-3-resttemplate/
[3] http://www.springsource.org/

2.6. Wykorzystanie klasy Geokodera

Każda restauracja dodawana za pomocą systemu zarządzania treścią (CMS - Content Management System) musi mieć zdefiniowane współrzędne geograficzne (długość i szerokość). Dane te są pobierane przez urządzenia z serwera i zapisywanie w lokalnej bazie danych. Do tłumaczenia współrzędnych geograficznych (ang. reverse geocoding) na adres formalny - ulicę, nr domu/mieszkania/lokalu, miasto i kod pocztowy - wykorzystano klasę geokodera (Geocoder) i metodę wstecznego geokodowania.

Do prawidłowego działania klasa Geocoder wymaga części funkcjonalnej, która nie wchodzi w skład systemu Android, dlatego podczas korzystania z niej niezbędne jest zapewnienie połączenia z internetem.

Po stronie serwera wykorzystana została metoda geokodowania, która odbywa się w celu przetłumaczenia współrzędnych geograficznych na adres formalny. Są to współrzędne miejsca z którego wysłano zamówienie (położenie geograficzne urządzenia, a więc długość i szerokość geograficzna), które zostały pobrane automatycznie przez aplikację w trakcie składania zamówienia przez użytkownika. Bardzo ważne jest, aby ich dokładność była możliwie największa (do kilku metrów), w przeciwnym wypadku geokodowanie takich danych nie powinno się odbywać, ponieważ rezultat obarczony będzie dużym prawdopodobieństwem błędu.

2.7. Dane w formacie JSON – mapowanie na obiekty Java

Wszystkie dane - poza grafikami - serwer prezentuje w tekstowym formacie JSON (JavaScript Object Notation). W celu szybkiej konwersji danych tekstowych na obiekty Java wykorzystano w aplikacji bibliotekę Gson. Pozwala na dwukierunkową konwersję, a jej zaletą jest brak wymogu umieszczania adnotacji w definicji klasy (dzięki temu można nią konwertować dane do obiektów ze skompilowanych bibliotek). Jedynym wymogiem jest zgodność nazw pól składowych klasy z nazwą wartości w JSONie. W przypadku gdy nie wszystkie wartości znajdują swoje odzwierciedlenie w polach składowych klasy, wówczas biblioteka je pomija.

Definicja klasy OrderItem:

```
public class MenuItem {
  private String id;
  private String name;
  private String description;
  private String components;
  public MenuItem() {

  }
}
```

Przykładowe dane z serwera:

```
{"id":13, "name":"Produkt 13",
"description":"Opis produktu o id 13",
"components":"brak"}
```

Konwersja:

```
Gson converter = new Gson();
MenuItem mItem = converter.fromJson(json, MenuItem.class);
```

2.8. Biblioteka Action Bar Sherlock

Action Bar czyli tłumacząc dosłownie „pasek akcji“ jest systemowym komponentem, którego zadaniem jest ułatwienie użytkownikowi nawigacji po aplikacji oraz zapewnienie szybkiego dostępu do kontekstowego menu. Został wprowadzony dla systemu Android od wersji 3.0 (API 11), niestety nie jest on kompatybilny

z wcześniejszymi wersjami. Chcąc zapewnić wspieranie jak najszerszej gamy urządzęń przez aplikację, a co za tym idzie – jak największej ilości wersji systemu, niezbędne jest wykorzystanie bibilioteki emulującej pasek akcji w starszych wersjach systemu.

Action Bar Sherlock jest biblioteką stworzoną przez Jake Whartona, udostępnianą na licnecji open-source, która pozwala wykorzystać funkcjonalność Action Bar'a w aplikacjach dla starszych wersji systemu. W zależności od wersji systemu, biblioteka wykorzystuje natywny pasek akcji (w wersjach powyżej 3.0), lub implementuje bardzo podobne rozwiązanie dla niższych wersji systemu.[4]

W aplikacji „InstantPizza" wykorzystano pasek akcji, który prezentuje kontekstowe menu, a dzięki bibliotece Action Bar Sherlock – zapewniono kompatynilność aplikacji z systemami w wersji niższej niż 3.0.

2.9. Wykorzystanie lokalizacji

Lokalizacja urządzenia zostanie wykorzystania w aplikacji do określenia miejsca dostarczenia zamówienia oraz do aktualizacji położenia zamówienia dostarczanego. Aktywność która będzie ją pobierała, musi implementować interfejs LocationListener[5], a także należy nadać jej odpowiednie uprawienia w pliku AndroidManifest.xml.
Przykład definicji klasy implementującej LocationListener:

```
public class LocalizatorActivity extends Activity
implements LocationListener {
  (...)
}
```

Interfejs ten zawiera cztery metody publiczne, które muszą zostać nadpisane przez klasę implementującą. Są to odpowiednio:

- onProviderDisabled - moduł GPS został wyłączony
- onProviderEnabled - moduł GPS został włączony
- onLocationChanged - moduł GPS odebrał lokalizację
- onStatusChanged - moduł GPS nie może odebrać

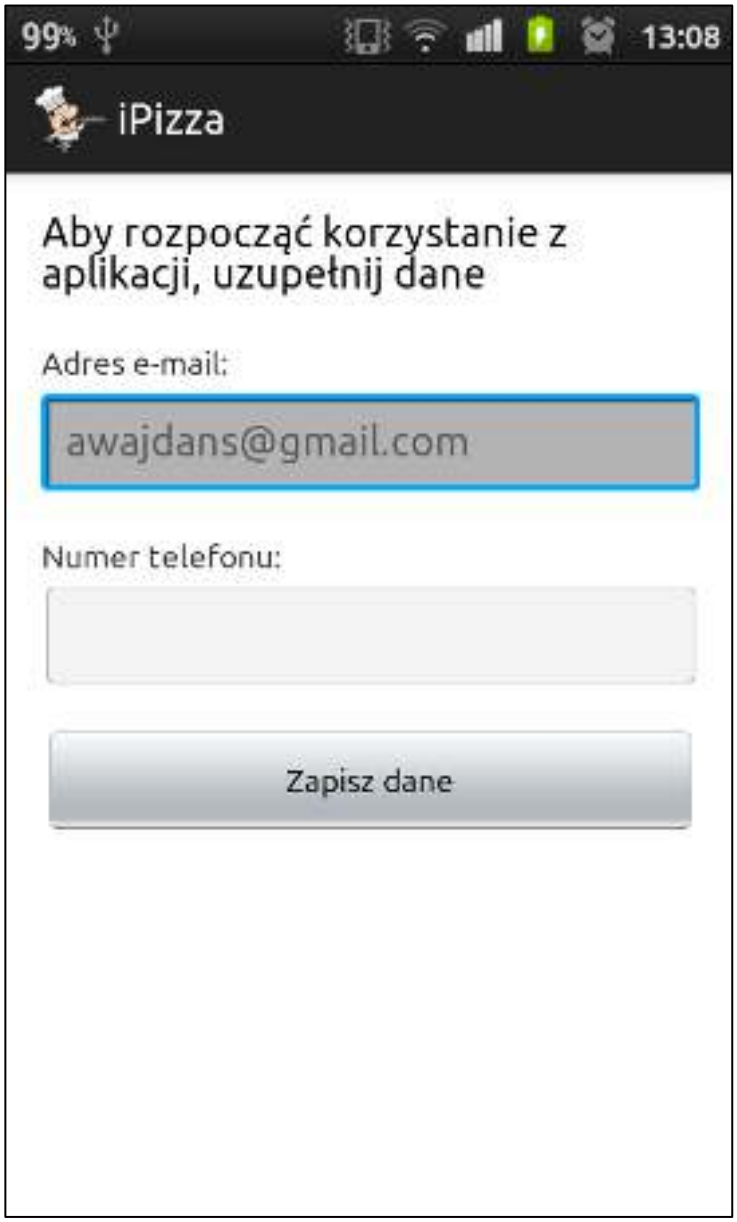
[4] http://actionbarsherlock.com/index.html
[5] http://developer.android.com/reference/android/location/LocationListener.htm

ROZDZIAŁ 3. PREZENTACJA APLIKACJI

3.1. Cechy i wygląd

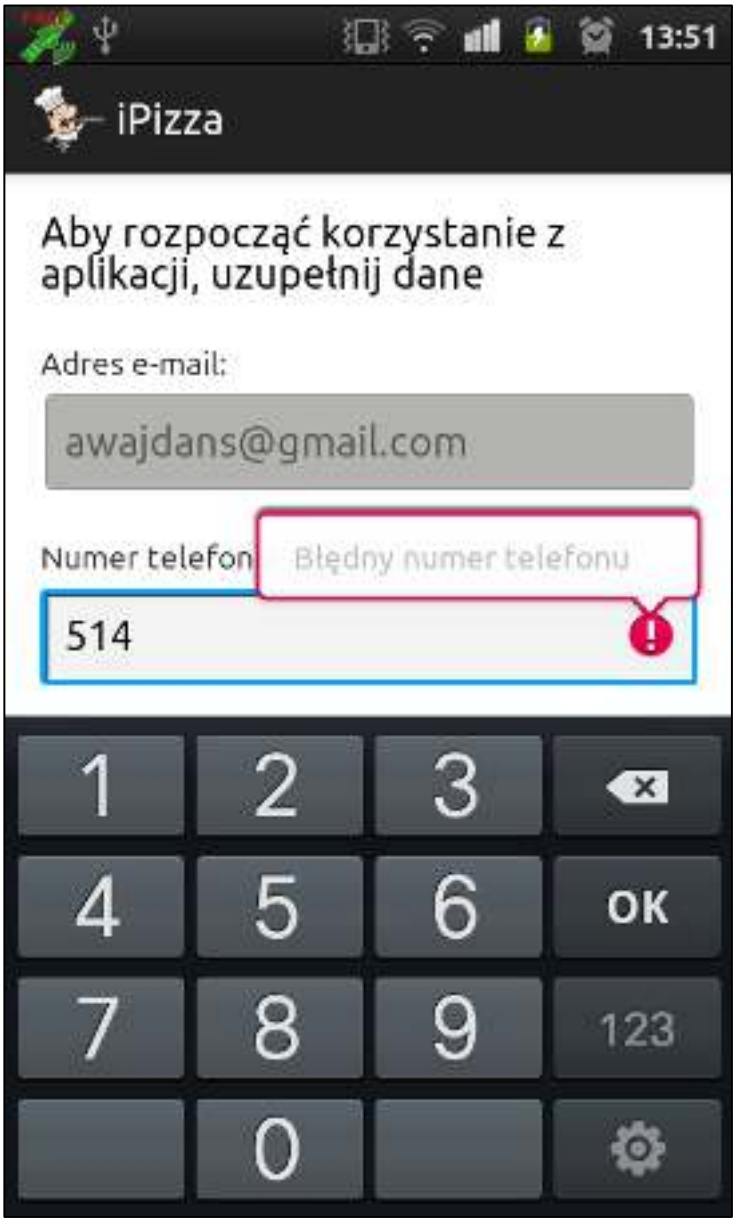
Pierwsze uruchomienie - szybka konfiguracja konta

Po zainstalowaniu aplikacji i pierwszym uruchomieniu, użytkownikowi prezentowany jest ekran z ustawieniami konta:



Zgodnie z założeniami, aplikacja ma na celu maksymalne uproszczenie procesu składania zamówienia, dlatego część danych jest pobierana automatycznie. Dzięki

wykorzystaniu adresu e-mail zapisanego w urządzeniu, adres ten nie wymaga dodatkowej weryfikacji. Do rozpoczęcia korzystania użytkownik musi wprowadzić jedynie numer telefonu, który będzie przypisany do jego zamówień. W zależności od urządzenia i wersji systemu Android, możliwe jest że aplikacja automatycznie pobierze numer telefonu - jednak w takiej sytuacji użytkownik także będzie miał możliwość jego edycji.

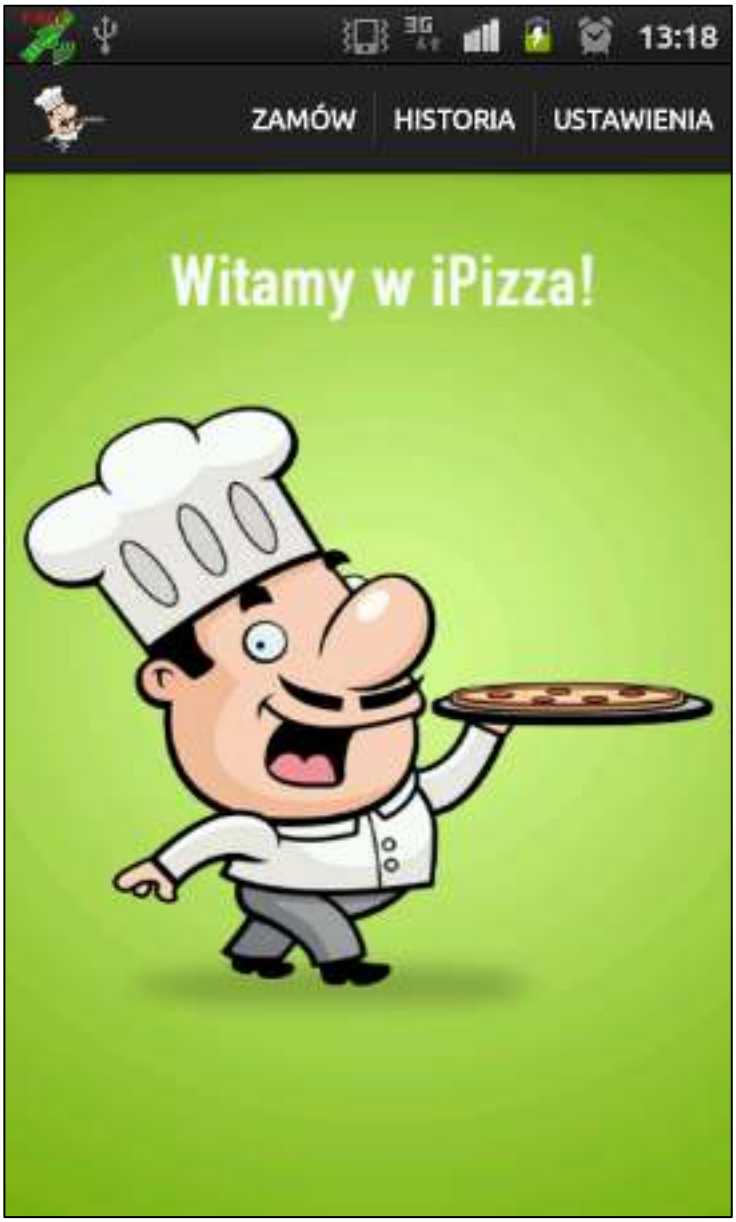


W trakcie wpisywania numeru telefonu jest on dynamicznie sprawdzany pod względem poprawności.

Ekran startowy aplikacji

Po zapisaniu danych użytkownikowi prezentowany jest ekran startowy aplikacji, z poziomu którego do wyboru ma trzy opcje:

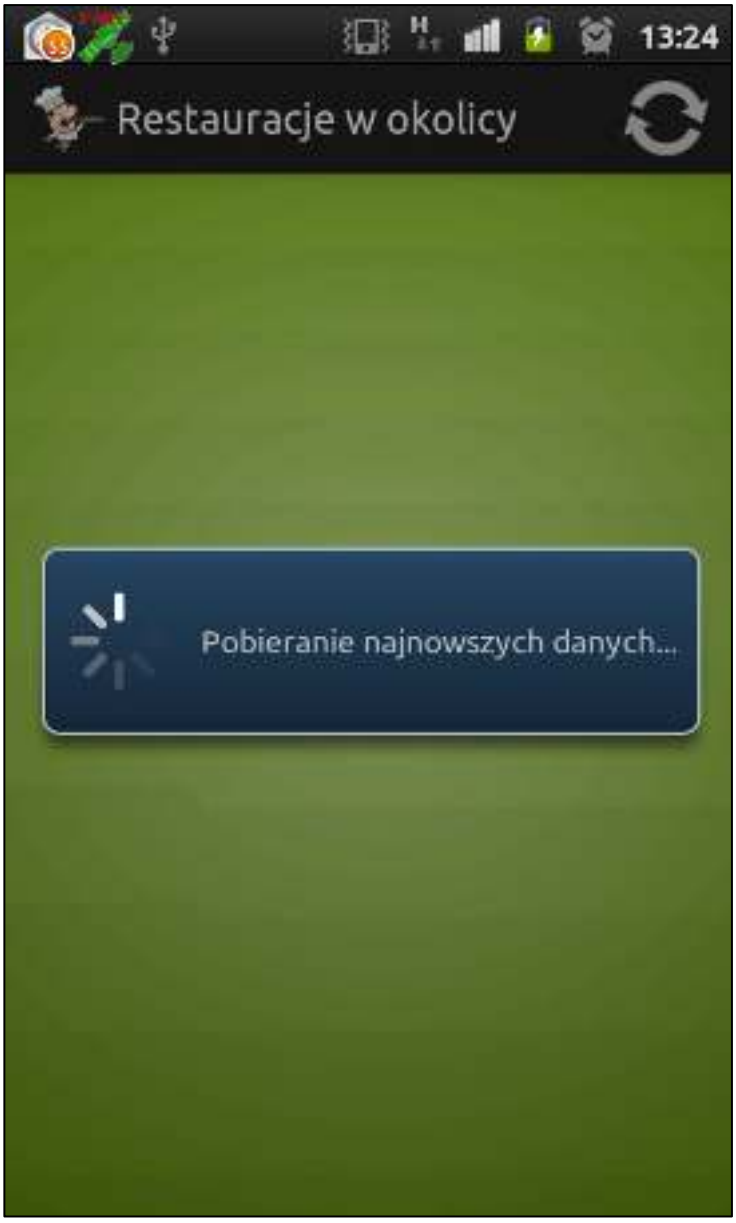
- rozpoczęcie nowego zamówienia
- sprawdzenie historii zamówień
- modyfikacja i podgląd ustawień konta



Składanie zamówienia

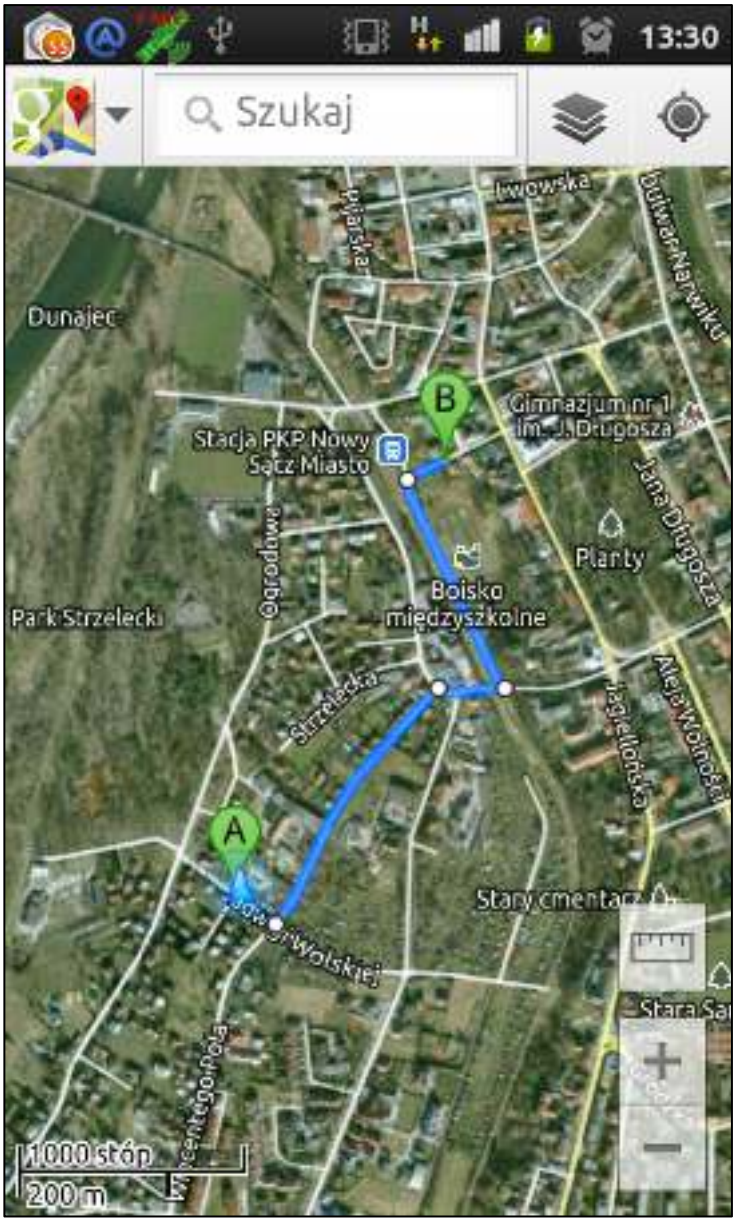
Po wyborze odpowiedniej opcji z górnego paska menu, charakterystycznego dla platformy Android – rozpoczyna się proces składania zamówienia. Rozpoczyna się od ustalenia lokalizacji użytkownika i poniższego ekranu:

Po ustaleniu lokalizacji użytkownika, automatycznie rozpoczyna się proces pobierania najnowszych danych z serwera.



Dane zostają zapisane w pamięci urządzenia i kolejno są poddawane analizie – aplikacja filtruje pobrane dane tak, aby były użyteczne dla użytkownika – wyświetla wszystkie restauracje, w zasięgu których użytkownik się znajduje (nie przekracza zasięgu dowozu).

Oprócz możliwości rozpoczęcia składania zamówienia w wybranej restauracji, użytkownik może także wytyczyć trasę dojazdu do niej z wykorzystaniem aplikacji Google Maps



Proces składania zamówienia

Po wybraniu restauracji w której będzie składane zamówienie użytkownik uzyskuje dostęp do oferty w postaci menu, pogrupowanego na kategorie (rodzaje dań/potraw). Liczba w nawiasie obok nazwy kategorii to ilość produktów w danej grupie.

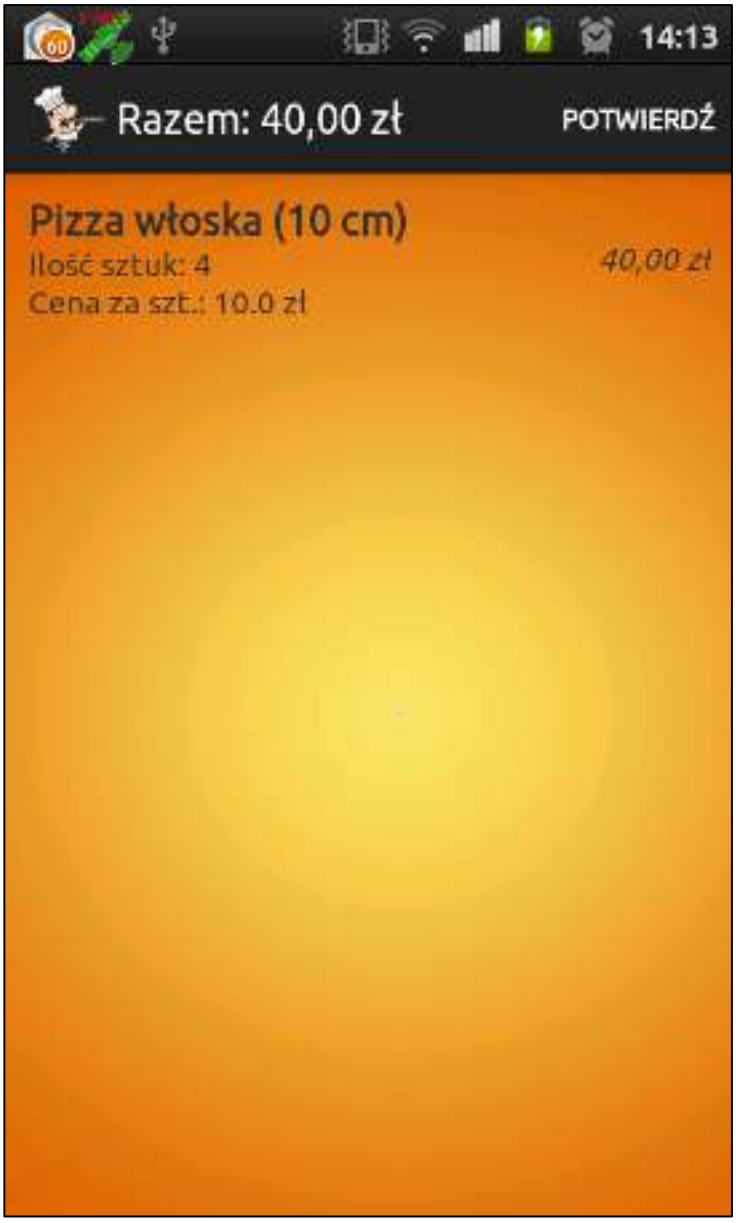
Po kliknięciu w przycisk „Pokaż pozycje menu“ użytkownik jest kierowany do listy produktów w wybranej kategorii. Przeglądanie menu odbywa się za pomocą gestów. Produktuy są prezentowane ze zdjęciem, nazwą oraz krótkim opisem.

Jeżeli użytkownik jest zainteresowany danym produktem, w prosty sposób może sprawdzić jego ceny (przeważnie produkty mają więcej niż jedną cenę – występują w wielu opcjach). Ceny produktu są pobierane na bieżąco.

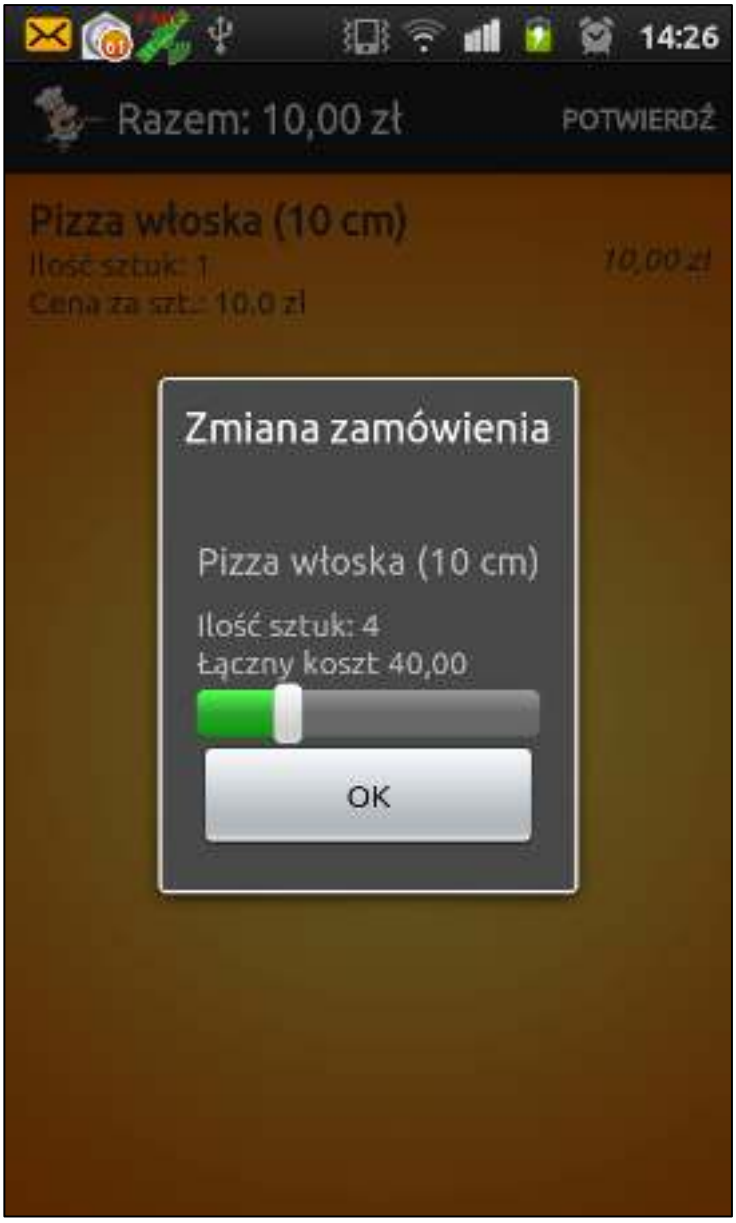
Po wyborze rozmiaru danego produktu, do zamówienia zostaje dodana jego jedna sztuka z odpowiadającą ceną. Jeżeli użytkownik ma przynajmniej jeden produkt w swoim zamówieniu, górny pasek zmienia swoją funkcję i stanowi skrócone podsumowanie zamówienia – wyświetla ilość produktów na zamówieniu. Dodatkowo w widoku pojawia się przycisk „Zobacz zamówienie“, który przeniesie użytkownika do szczegółowego podsumowania.

Podsumowanie zamówienia

Widok z podsumowaniem zamówienia zawiera listę, na której znajdują się wszystkie produkty dodane przez użytkownika, wraz z informacją o ilości sztuk, cenie za sztukę oraz sumarycznej cenie dla wybranego produktu, jak również sumą całego zamówienia.



Użytkownik ma możliwość edycji zamówienia – ilości poszczególnych produktów lub całkowitego ich usunięcia. W każdej chwili może także powrócić do menu wybranej restauracji, aby dodać nowe produkty.



Jeśli użytkownik zakończy tworzenie listy produktów, przechodzi do ekranu gdzie może zapoznać się z informacjami jakie zostaną przesłane z zamówieniem.

Po zatwierdzeniu tych danych, zamówienie zostaje przesłane do serwera.

Po pomyślnym przetworzeniu zamówienia po stronie serwera, aplikacja przenosi użytkownika do ekranu z historią zamówień, gdzie może on sprawdzać na bieżąco status zamówień.

PODSUMOWANIE

W ramach pracy dyplomowej zaprojektowano i stworzono mobilny system do składania zamówień w restauracjach. Główne założenia projektu, czyli prostota aplikacji, wykorzystanie danych i możliwości technologicznych dostępnych w urządzeniu przenośnym w celu skrócenia czasu składania zamówienia, zostały w pełni osiągnięte.

Aplikacja wraz z serwerem, który jest tematem osobnej pracy dyplomowej, tworzą spójną platformę która mogłaby zostać z powodzeniem wdrożona na rzeczywisty rynek. Zarówno od strony klienta - czyli użytkownika aplikacji mobilnej - wszystkie funkcjonalności potrzebne do sprawdzenia oferty restauracji oraz złożenia zamówienia zostały zaimplementowane. Dla restauracji zainteresowanych prezentowaniem swojej oferty oraz obsługą klientów kanałem mobilnym stworzono odpowiednie narzędzia, niezbędne do realizacji tych celów.

Istnieje duże prawdopodobnieństwo, że w niedalekiej przyszłości powstanie podobna platforma mobilna, jako odpowiedź na rosnące udziały rynku mobilnego w zamówieniach składanych drogą online.

System daje możliwość rozbudowy, ponieważ z czasem można wprowadzić program rabatowo-lojalnościowy: dla użytkowników - za składanie zamówień, a dla restauracji - za umieszczanie promocji w systemie. Można także podnieść jakość procesów towarzyszących składaniu zamówień, jak np. proces dostarczania zamówienia - poprzez stworzenie dedykowanej aplikacji dla dostawców, zintegrowanej z systemem- aktualizaującej stan zamówienia i przybliżony czas dostarczenia na podstawie odległości dostawcy od celu.

BIBLIOGRAFIA

1. Android Developers, Google Inc., http://developer.android.com/reference/android/app/Activity.html, odczyt: luty 2013
2. A. Poutsma, Go Pivotal Inc., http://blog.springsource.org/2009/03/27/rest-in-spring-3-resttemplate/, odczyt: luty 2013
3. Go Pivotal Inc., http://www.springsource.org/, odczyt: styczeń 2013
4. J. Wharton, http://actionbarsherlock.com, odczyt: marzec 2013
5. Android Developers, Google Inc., http://developer.android.com/reference/android/location/LocationListener.html, odczyt: kwiecień 2013
6. Android Developers, Google Inc., http://developer.android.com/training/best-ux.html, odczyt: styczeń 2013
7. R. Meier, Professional Android™ Application Development,Wiley Publishing Inc., 2010
8. E. Burnette, Hello, Android - Introducing Google's Mobile Development Platform (Third Edition), The Pragmatic Programmers LLC, United States of America 2011
9. S. Hashimi, S. Komatineni, D. MacLean, Pro Android 2, Wyd. Apress, 2010
10. B. Eckel, Thinking in Java (4th Edition), Pearson Education Inc., 2006