1 Streszczenie

Poniższa praca zawiera opis biblioteki „Ash" służącej do funkcjonalnego testowania hybrydowych aplikacji mobilnych stworzonych przy użyciu Adobe PhoneGap lub Apache Cordova. Ash pozwala na testowanie zachowania się aplikacji w różnorodnych realistycznych scenariuszach, które bywają trudne do symulowania przez inne narzędzie, takich jak poruszanie się użytkownika, obrót ekranu, utrata dostępu do sieci oraz inne. Dzięki wykorzystaniu hybrydowego charakteru aplikacji możliwa jest emulacja zachowania, co wpływa na większy realizm testów, a tym samym na wiarygodność ich wyników. Innymi zaletami bliblioteki są elastyczna struktura testów, która ułatwia utrzymywanie testów i budowanie złożonych scenariuszy z prostych testów-kroków oraz możliwość wykorzystania asercji w aplikacji poza testami. Możliwe jest także wykorzystanie Ash do testowania mobilnych wersji stron internetowych.

2 Wprowadzenie

2.1 Hybrydowe aplikacje mobilne

Mówiąc o natywnej aplikacji mobilnej mamy na myśli apliakcję tworzoną z myślą o konkretnej platformie (Android, iOS, itp.) przy użyciu narzuconych przez twórcę platformy narzędzi (Java, Objective-C, itd.). Aplikacje mobile web są to z kolei aplikacje webowe zoptymalizowane po kątem urządzeń natywnych. Adobe PhoneGap oraz jej odpowiednik o otwartym źródle Apache Cordova, to dwie popularne biblioteki pozwalajace na tworzenie hybrydowych aplikacji mobilnych, tj. aplikacji które łączą w sobie zalety aplikacji natywnych oraz aplikacji typu mobile web. Zasada działania tego typu aplikacji jest w założeniu prosta i polega na wykorzystaniu komponentów, które dalej nazywać będziemy WebView. Kontrolki te są dostępne na każdej nowoczesnej platoformie i pozwalają nam na wyświetalnie stron internetowych z wnętrza natywnych aplikacji mobilnych. W momencie startu aplikacja hybrydowa tworzy WebView oraz ładuje do niego zasoby z określonego adresu (lokalnego lub zdalnego), które są wyświetlane w WebView. Najczęściej tym zasobem jest aplikacja stworzona przy użyciu technologii webowych.

	Aplikacje natywne	Aplikacje hybrydowe	Aplikacje mobile web
Narzędzia	Zależne od platformy	Narzędzia webowe	Narzędzia webowe
Instalacja	Wymagana	Wymagana	Nie wymagana
Wydajność	Bardzo dobra	Ograniczona przez przeglądarkę	Ograniczona przez przeglądarkę
Możliwość monetyzacji	Pobranie, reklamy, subskrybcje	Pobranie, reklamy, subskrybcje	Reklamy, subskrybcje
Dostęp do urządzenia	Tak, pełen	Najważniejsze funkcjonalności	Ograniczony

Takie podejście ma wiele zalet. Dzięki temu, że do tworzenia aplikacji hybrydowych wykorzystywane są technologie znane z zastosowań intenetowych, koszt tworzenia aplikacji i czas dostarczenia gotowego rozwiązania na rynek są znacznie zredukowane. Aplikacje tego typu są też z założenia wieloplatformowe. Używając PhoneGap z tego samego kodu źródłowego możemy stworzyć aplikacje na platformę Android, iOS, Blackberry, WebOS, Windows Phone, Symbiana i Bada. Dodatkowo popularność narzędzi internetowych ułatwia znalezienie właściwych programistów.

2.2 Wady podejścia hybrydowego

Największymi wadami podejścia hybrydowego są słabsza wydajność, błędy pojawiające się tylko na określonych urządzeniach oraz różnorodne oczekiwania użytkowników różnych platform. Użytkownik instalując aplikację hybrydową w taki sam sposób co natywną spodziewa się, że będzie ona działać niczym natywna. Tymczasem dodatkowy narzut hybrydy, często w połączeniu z niechlujnie przygotowaną i nie zoptymalizowaną aplikacją, prowadzi do mniej responsywnego interfejsu użytkownika i w konskewencji do frustracji użytkownika. Częstym problemem przy tworzeniu aplikacji mobilnych są błędy pojawiające się tylko na określonych urządzeniach. Przy podejściu hybrydowym problem jest o tyle bardziej widoczny, że najczęściej tworzymy rozwiązanie które ma być przenośne nie tylko pomiędzy urządzeniami w ramach jednej platformy, ale także między platformami. Co z oczywistych względów zwiększa przestrzeń urządzeń, które trzeba uzwględnić. Trzecim problemem hybryd jest kwestia tworzenia interfejsu użytkownika. Dostawcy systemów operacyjnych dla urządzeń mobilnych publikują

zalecenia, co do tego jak powinien wyglądać interfejs aplikacji działającej pod danym systemem. Zalecenia te są specyficzne dla platformy i często wzajemnie się wykluczają. Przygotowanie jednej szaty graficznej i jednego interfejsu może zostać źle odebrane przez użytkowników spodziewających się wyglądu dostosowanego do platformy. Niestety stworzenie kilku wersji interfejsu jest dużo bardziej pracochłonne i skomplikowane, niweczy także podstawową zaletę apliakcji hybrydowych – przenośność. Są to poważne niedostatki, ale braki te można zniwelować poprzez skuteczne narzędzia.

2.3 Propozycje rozwiązania problemów

Ash ma za zadanie pomóc rozwiązać problemy zwiąne z wydajnością aplikacji oraz z błędami pojawiającymi się tylko na określonych konfiguracjach sprzętowych. Aby zapewnić wysoką wydajność działania aplikacji konieczny jest rygor w tworzeniu wydajnego oprogramowania oraz możliwość testowania jej w sytuacjach które mogę uwydatnić problemy z wydajnością. Dzięki funkcyjnemu podejściu do testowania oprogramowania Ash pozwala na zbieranie informacji o responsywności interfejsu oraz realistycznie symulować scenariusze dużego obciążenia. Programista korzystający z Ash ma możliwość zdefiniowania w scenariuszach maksymalnego czasu przebeigu testu, jeśli test nie zakończy się w założonym czasie test nie powiedzie się. Niska responsywność interfejsu traktowana jest na równi z błędami logiki czy prezentacji. Wymusza to na twórcy zadbanie o szybkość reakcji interfejsu. Ash oferuje także możliwość chwilowego wyłączenia lub opóźnienia dostępu do sieci. Sytuacje w których dostępność do internetu jest ograniczona są dość częste, jednak niewiele aplikacji jest pisana biorąc to pod uwagę, jeszcze mniej jest testowana pod tym kątem. Bardzo często problemy z dotępem objawiają się kiepską wydajnością interfejsu lub długimi przestojami na ekranach ładowania. Ash pozwala twórcom świadomie zmierzyć się z tym problemem. Ash wyposażony jest w mechanizmy pozwalajace na łatwe uruchomienie na aplikacji na wielu urządzeniach naraz, fizycznych jak i wirtualnych, także w zdalnych lokalizacjach. Sprawia to, że użytkownicy są mają możliwość masowego uruchamiania testów na wszystkich dostępnych im urządzeniach oraz są bardziej skłonni skorzystać z testów w czaseie swojej pracy. Możliwość zdalnego uruchomienia testów daje możliwość stworzenia rozproszonej bazy urządzeń, co pozwoli na testowanie także nietypowych konfiguracji.

2.4 Dlaczego Apache Cordova

Jako bazę do implementacji wybrałem Apache Cordova. Apache Cordova jest to wersja PhoneGap, którą korporacja Adobe (właściel praw do PhoneGap) udostępniła fundacji Apache. Od tego momentu ten wariant technologii udostępniany jest zasadzie otwartego źródła. Pomimo tego oba projekty są niemal identyczne, a jedyna różnica faktyczna różnica między nimi jest natury prawnej. Głównym powodem wyboru tej technologii jest spory udziała w rynku hybrydowych aplikacji mobilnych, dynamizm rozwoju oraz liczna społeczność.

2.5 Propozycje rozwiązania problemów

Ash ma za zadanie pomóc rozwiązać problemy zwiąne z wydajnością aplikacji oraz z błędami pojawiającymi się tylko na określonych konfiguracjach sprzętowych. Aby zapewnić wysoką wydajność działania aplikacji konieczny jest rygor w tworzeniu wydajnego oprogramowania oraz możliwość testowania jej w sytuacjach które mogę uwydatnić problemy z wydajnością. Dzięki funkcyjnemu podejściu do testowania oprogramowania Ash pozwala na zbieranie informacji o responsywności interfejsu oraz realistycznie symulować scenariusze dużego obciążenia. Programista korzystający z Ash ma możliwość zdefiniowania w scenariuszach maksymalnego czasu przebeigu testu, jeśli test nie zakończy się w założonym czasie test nie powiedzie się. Niska responsywność interfejsu traktowana jest na równi z błędami logiki czy prezentacji. Wymusza to na twórcy zadbanie o szybkość reakcji interfejsu. Ash oferuje także możliwość chwilowego wyłączenia lub opóźnienia dostępu do sieci. Sytuacje w których dostępność do internetu jest ograniczona są dość częste, jednak niewiele aplikacji jest pisana biorąc to pod uwagę, jeszcze mniej jest testowana pod tym kątem. Bardzo często problemy z dotępem objawiają się kiepską wydajnością interfejsu lub długimi przestojami na ekranach ładowania. Ash pozwala twórcom świadomie zmierzyć się z tym problemem. Ash wyposażony jest w mechanizmy pozwalajace na łatwe uruchomienie na aplikacji na wielu urządzeniach naraz, fizycznych jak i wirtualnych, także w zdalnych lokalizacjach. Sprawia to, że użytkownicy są mają możliwość masowego uruchamiania testów na wszystkich dostępnych im urządzeniach oraz są bardziej skłonni skorzystać z testów w czaseie swojej pracy. Możliwość zdalnego uruchomienia testów daje możliwość stworzenia rozproszonej bazy urządzeń, co pozwoli na testowanie także nietypowych konfiguracji.

2.6 Propozycje rozwiązania problemów

Jako bazę do implementacji wybrałem Apache Cordova. Apache Cordova jest to wersja PhoneGap, którą korporacja Adobe (właściel praw do PhoneGap) udostępniła fundacji Apache. Od tego momentu ten wariant technologii udostępniany jest zasadzie otwartego źródła. Pomimo tego oba projekty są niemal identyczne, a jedyna różnica faktyczna różnica między nimi jest natury prawnej. Głównym powodem wyboru tej technologii jest spory udziała w rynku hybrydowych aplikacji mobilnych, dynamizm rozwoju oraz liczna społeczność.

2.7 Test Driven Development

Test Driven Development jest to metodologia wytwarzania oprogramowania opierająca się na krótkich cyklach programistycznych, o których często mówimy red-green-refactor:

- RED Programista pisze testy automatyczne, które pokrywają funkcjonalność, która została wyspecyfikowana, ale nie jest jeszcze zaimplementowana. Na tym etapie testy nie przechodzą
- GREEN Następnie programista implementuje brakujące funkcjonalności. Programista pisze kod, aż wszystkie testy napisane w poprzednim kroku
- REFACTOR Po zakończeniu implementacji tworzone są dodatkowe testy do istniejących już funkcjonalności. Sama funkcjonalność też jest udoskonalona

Dzięki zastosowaniu krótkich cykli oraz automatycznych testów metodologia ta pozwala na tworznie oprogramowania, które ma przejrzystą architekturę oraz spełnia wszystkie założenia. Testy automatyczne, które powstają podczas cykl dają pewność, że w przypadku, gdy kod aplikacji zmieni się potencjalne błędy zostaną od razu wychwycone.

Test Driven Developement jako metodologia jest powiązana z Extreme Programming oraz z koncepcją "Tests First", która zakłada że zestawy testów automatycznych powinny być tworzone jeszcze przed napisaniem tesotwanej aplikacji, co skutkuje wysoką jakością tworzonego oprogramowania.

Test Driven Developement zostało spopularyzowane przez Kenta Becka.

Więcej informacji można znaleźć na:

http://www.agiledata.org/essays/tdd.html http://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx

2.8 Hierarchia testów

Wyróżniamy wiele rodzajów testów:

- Jednostkowe. Testy obejmujące poszczególne funkcje na najniższym poziomie. Ich wykonywanie musi być możliwie jak najszybsze. Nie powinny korzystać z zewnętrznych zasobów
- Integracyjne. Ten rodzaj testów pokrywa wiele komponentów aplikacji na raz i testuje nie tylko, czy każdy z poszczególnych działa poprawnie w izolacji, ale także czy komunikacja między nimi jest poprawna
- Funkcyjne. Testy tego typu sprawdzają, czy interfejs prezentowany użytkownikowi jest poprawny oraz czy interakcja z użytkownikiem działa poprawnie. U podstaw działanie tych testów leży symulowanie zachowania użytkownika
- A/B. Tak zwane testy A/B polegają na testowaniu oprogramowania pod kątem wygody i praktyczności interfejsu
- Wydajnościowe. Jak sama nazwa wskazuje służą one do sprawdzenia wydajności aplikacji
- oraz inne ...

W przypadku poprawnie zarządzanego projektu trzy pierwsze rodzaj tworzę hierarchię zwaną "piramidą testowania". Dwa pozostałe z wymienionych przeze mnie rodzajów pełni funkcję uzupełniającą.

TODO OBRAZEK PIRAMIDY

Biblioteka Ash została stworzona z testowaniem funkcjonalnym na myśli.

3 Sposób użycia

3.1 Getting Started

Aby dołączyć Ash do swojego projektu wydaj następujace polecenie[1]:

cordova plugins add https://github.com/pjazdzewski1990/Ash

Wtyczka zostanie zainstalowana, a do globalnej przestrzeni dodany zostanie obiekt Ash, który udostępnia wszystkie metody biblioteki.

Punktem startowym Ash jest metoda loadTests. Metoda ta przyjmuje tablicę ścieżek do plików zawierających kod z testami i po wywołaniu dynamicznie dodaje wskazane pliki do dokumentu w ramach którego został wywołany. Pliki z testami należy napisać samodzielnie pamiętając, żeby były one zamknięte w ramach self-invocing-function. Celem takiego podejście służy do tego, żeby umożliwić użytkownikowi wpięcie i odpięcie testów z aplikacji oraz aby testy niepotrzebnie nie zaśmiecały aplikacji produkcyjnej.

Projekt demonstrujący użycie Ash oraz jego możliwości jest dostępny pod adresem

https://github.com/pjazdzewski1990/AshDemo

3.2 Hello World

3.3 Filozofia

3.4 Ograniczenia

Ash został stworzony w sposób na tyle elastyczny, aby był w stanie integrować się najróżniejszymi projektami. Niestety nie zawsze możliwe było uwzględnienie wszystkich możliwości, dlatego konieczne było przyjęcie pewnych założeń co do projektu który ma być testowany przez Ash. Zrozumienie tych założeń jest istotne do sprawnego posługiwania się biblioteką, zrozumienia przyczyn trudności w jej używaniu oraz przy podejmowania decyzji czy dołączyć Ash do swojego projektu.

Założnie Ash co do projektu testowanego:

- wykonany w technologii Adobe Phonegap/Apache Cordova - istnieje możliwość aby w przyszłości testować także aplikacje typu mobile web, ale na obecnym etapie nie jest to celem Ash
- single page app - Ash zakłada, że aplikacja testowana będzie pisana zgodnie z najnowszymi trendami w branży oraz zaleceniami twórców Phonegap
- android - obecnie Ash pozwala testować jedynie aplikacje bazujące na Androidzie jako platformie, gdyż ta jest dominująca na rynku, ale w niedalekiej przyszłości planowana jest także wersja na iOS

4 Przegląd funkcjonalności

4.1 Zmiana położenia ekranu

4.1.1 Sposób użycia

Pozwala ustawić ekran w określonym położeniu - landscape/horizontal lub portrait/vertical.

Ash.orientationHorizontal().then(...)

Ash.orientationVertical().then(...)

4.1.2 Opis

Smartfony i tablety ze względu na swój mobilny charakter oraz dotykowy ekran bardzo często używane są w pozycjach i w sposób, o których twórcom nawet się nie śniło. Najczęściej rozróżniamy dwie pozycje definiujące położnie ekranu wzgledem użytkownika

- pionowe - gdy TODO

[1] Jeśli nie masz jeszcze zainstalowane Cordova CLI spójrz tutaj https://github.com/apache/cordova-cli

- poziome - gdy TODO

Każda dobrze wykonana aplikacja powinna uwzględniać te dwie pozycje. W każdej z nich przestrzeń, którą mamy do dyspozycji jest w prawdzie taka sama, ale jej posterzganie przez użytkownika jest inne, dlatego należy inaczej rozmieścić elementy interfejsu.

Funkcje orientationHorizontal oraz orientationVertical ustawiają ekran na pozycjach, odpowiednio, poziomej oraz pionowej względem domyślnej pozycji "zero" urządzenia. Pozycja ta może, ale nie musi być tą pozycją w której w momencie wołania znajduje się ekran.

Należy pamiętać, że poza kontekstem orientationX status ekranu nie jest określony. Może być on dowolny. W szczególności wykonanie testu z orientation nie musi po fakcie nie musi przywracać poprzedniego ustawienia. Celem takiego podejścia jest wymuszenie na użytkowniku biblioteki zamykanie całego kodu związanego z położeniem w kontekście obrotu ekranu.

Praktycznym podejściem może być sprawdzanie jak na zmianę ekranu reagują pojedyncze części ekranu zamiast całość. innym podejściem jest przetestowanie jak aplikacja reaguje na zmianę położenia ekranu a następnie na jego powrót. Można to zrealizować w ten sposób:

```
//example.js

Ash.orientationHorizontal().then(function(msg){
    //sprawdz czy interfejs jest ulozony dla pozycji poziomej
  }).then(
    Ash.orientationVertical
  ).then(function(msg){
    //sprawdz czy interfejs jest ulozony dla pozycji pionowej
    Ash.endTest();
  });
};
```

4.2 Dostęp do sieci

4.2.1 Opis

Daje możliwość manipulowania dostępem do sieci - opóźniać reakcję, zatrzymywać oraz przywracać połączenie.

4.2.2 Sposób użycia

4.3 Dostęp do systemu plików

4.3.1 Opis

Daje dostęp do systemu plików - pozwala na symulowanie dostępu do lokalnych plików oraz ich uploadu.

4.3.2 Sposób użycia

4.4 Symulacja ruchu

4.4.1 Opis

Ash dostarcza API umożliwiające symulowanie położenia lub ruchu między dwoma punktami.

4.4.2 Sposób użycia

4.5 Obsługa przycisku powrotu

4.5.1 Sposób użycia

Ta funkcjonalność symuluje naciśnięcie przycisku powrotu na telefonie.

Ash.pressBack().then(...)

4.5.2 Opis

Integralną cześcią sytemu Android jest przycisk powrotu do poprzedniego ekranu. Specyfikacja systemu określa dokładnie

TODO: umieścić wycinek speca tutaj

Niezależnie od dostawcy sprzętu, implementacji czy wersji systemu zastosowanie przycisku zawsze jest takie samo - pozwolić użytkownikom na łatwą nawigację w aplikacji, która będzie spójna pomiędzy najróżniejszymi aplikacjami. Bardziej precyzyjnie nawigacja w tym kontekście oznacza możliwość powrot do poprzedniej aktywności lub ekranu prezentowanego użytkownikowi. W przypadku aplikacji hybrydowych opartych na Javascript oraz technologiach webowych powrót do poprzedniej aktywności oznacza cofnięcie stanu WebView do poprzedniego adresu URL. Wszystkie adresy dowiedzane przechowywane są na stosie, a ich unikalnośc determinowana jest przez 3 cześci adresu: hosta, ścieżkę oraz tzw. hash.

```
//example.js

var pressBackTest = function(){
  var pageOne = 1;
  var pageTwo = 2;
  // function app.mySwipe.slide(A, x) nawiguje do podstrony A w ciagu x milisekund,
  // po zakonczonej nawigacji hash adresu jest zmieniany
  app.mySwipe.slide(pageOne, 1);
  app.mySwipe.slide(pageTwo, 1);
  app.mySwipe.slide(pageTwo + 1, 1); // wykonujemy 3 ruchy, zeby moc cofnac sie 2 razy i wrocic do
      punktu wyjscia

  Ash.pressBack().then(function(msg){
    Ash.equal("#" + pageTwo, window.location.hash);
  }).then(
    Ash.pressBack
  ).then(function(){
    Ash.equal("#" + pageOne, window.location.hash);
    Ash.endTest();
  });
};
```

Ash pozwala na cofanie się aż do szczytu stosu historii aplikacji. W szczególności od razu po uruchomieniu aplikacji nie możliwe jest wykonie pressBack. W takiej sytuacji pressBack nie powiedzie się, ale nie dojdzie także do wyjścia z aplikacji.

Testowanie obsługi przycisku powrotu na olbrzymie znaczenie w przypadku apliakcji mobilnych oraz tzw. "Single Page Applications". Tworząc aplikacje bardzo wiele uwagi jest poświęcane sytuacjom kiedy użytkownik porusza się niejako "w głąb" aplikacji otwiera kolejne ekrany na zaplanowanej ścieżce zmieniając jej stan. Właśnie globalny stan bywa w takich sytuacjach problemem. Bardzo często bywa, że użytkownik przechodzi przez ekrany A i B, w czasie swojego pobytu na B zmienia globalny stan, ale zamiast przechodzić dalej cofa się do A, które od czasu zmiany stanu zachowuje się inaczej niż zakładano. Tworząc aplikację należy zabezpieczyć się przed takimi sytuacjami. Ash.pressBack pozwala na przetestowanie czy taka sytuacja ma miejsce.

5 Zastosowania

Ash został pomyślany jako biblioteka, która jest w stanie oprócz dostarczania "tradycyjnej" infrastruktury testowej, tzn.:

- asercji
- funkcji porównujących
- loggerów
- narzędzi do budowanie testów

- ustandaryzowanych wyjątków

ułatwić także tesotwanie sytuacji, które są albo trudne do odtworzenia

- operacje na plikach
- badanie widoczności elementu na ekranie

albo w poza możliwościami typowych rozwiązań testujących

- ustawienie ekranu urządzenia
- dostęp do sieci
- symulowanie ruchu

Ash może z powodzeniem być stosowany zarówno jako osobny (standalone) i samodzielny framework testujący, jak i uzupełnienie innych rozwiązań. W ogólnym ujęciu Ash jest przeznaczony do fukcyjnego testowania aplikacji, tzn. aplikacja jest testowana od strony interfejsu użytkownika, co ma swoje zalety (najbardziej wiarygodnie symuluje użytkownika) jak i wady (interfejs często podlega zmianom). Jak zostało wcześniej wspomniane testy funkcyjne dają najlepsze efekty, gdy są uzupełniane przez testy jednostkowe.

5.1 Problemy związane z obecnymi rozwiązaniami

Obecnie istniejące rozwiązania pozwalajace na funkcyjne testowanie hybrydowych aplikacji mobilnych nie są wystarczające. Większość z nich wywodzi się z technologii webowych co ogranicza ich zastosowanie tylko do tej części aplikacji, która odpowiada aplikacji typu mobile web. Rozwiązania te nie są w stanie wykorzystać przewagi jaką daje hybrydowe podejście Apache Cordova, nie są w stanie ani przetestować ani skorzystać z natywnych podwalin aplikacji hybrydowej. W naturalny sposób ogranicza to ich możliwości oraz zastosowanie w kontekście rozważanych aplikacji. Naturalną wadą wielu bibliotek do funkcyjnego testowania aplikacji jest ich wrażliwość na zmiany interfejsu użytkownika. W przypadku większości aplikacji warstwa interfejsu jest tą częścią, która zmienia się najczęściej. W konsekwencji pożądaną własnością dobrej biblioteki do testowania funkcyjnego jest abstrakcja pozwalajaca na oderwanie się od detali interfejsu użytkownika oraz prostota pozwalająca na łatwą zmianę testów, gdyby zaszła taka potrzeba.

5.2 Ash jako rozwiązanie

Ash jest biblioteką służącą do funkcyjnego testowania hybrydowych aplikacji mobilnych, która stara się minimalizować niedostatki tego podejścia poprzez pełne wykorzystanie możliwości dawanych przez hybrydowy charakter testowanych aplikacji.

Ash działa na zasadzie wtyczki do Apache Cordova jest więc w stanie komunikować się z platformą sprzętową poprzez Native Bridge. Daje to możliwość manipulowanie ustawieniami czy zasobami urządzenia np. uzyskania dostępu do systemu plików czy ustawień połączenia. System operacyjny na którym uruchomiona jest aplikacja hybrydowa ma spory wpływ na to jak zachowa się aplikacja, niestety tradycyjne podejścia nie pozwalają na przetestowanie ich wpływu na aplikację. Co innego Ash, który udostępnia funkcjonalności pozwalające na wywoływanie określonych zachowań urządzenia.

Zewnętrzne API Ash napisane jest w jezyku Javascript i także w tym języku pisane są testy. Jestem zdania, że Javascript ze względu na swój charakter pozwoli w prosty i przejrzysty sposób wyrazić testy oraz będą one na tyle elastyczne, że wymagane zmiany będą mogły być wprowadzane niewielkimi nakładami sił. API Ash kładzie nacisk na na powtórne wykorzystywanie kodu oraz korzystanie ze wzorca Page Object, co pozwala na zapanowanie nad potrzebą aktualizacji testów do szybko zmieniającej się aplikacji.

5.3 RWE - real world example

6 Architektura

Ash dystrybuowany jest jako wtyczka do Apache Cordova z tego korzysta on także ze wszystkich mechanizmów typowych dla tego typu rozwiązań. Po dołączniu wtyczki do projektu np. za pomocą Cordova Command Line Interface w projekcie dostępny staje się globalny obiekt Ash, udostępniający wszystkie funkcjonalności biblioteki.

Pola globalnego obiektu można podzielić na 3 grupy:

- proste funkcjonalności działające w sposób synchroniczny
- fukcje działające asynchronicznie i udostępniające złożoną funkcjonalność
- prywatne, wewnętrzne pola oraz funkcje biblioteki

Warto podkreślić, że trzecia grupa nie wchodzi w skład oficjalnego, publicznego API Ash. Oznacza to, że o ile wiele z tych funkcjonalności czy helperów może być przydatna dla twórców testów, to nie rozsądnym jest korzystanie z nich bezposrednio, gdyż w przyszłych wersjach aplikacji w zależności od potrzeb kod ten może zostać zmieniony w sposób, który zmieni jego zachowanie lub/oraz API, co spowoduje niekompatybilność kodu testów z nową wersją Ash.

6.1 Spojrzenie z lotu ptaka

Ash definiuje funkcje pozwalajace na uruchamianie testów w kontekście pewnego zdarzenia sprzętowego, tj. możemy przygotować test, co do którego będziemy pewni, że w czasie jego działania będą zachodzić określone okoliczności. Możemy na przykład stworzyć test, który zostanie wykonany w sytuacji braku sieci np. w ten sposób:

```
 //example.js

//zdefiniuj kontekst braku sieci oraz uruchom w jego ramach przekazana funkcje
Ash.noNetwork().then(function(){
    // tu mozemy umiescic kod testu
    // zostanie on wykonany z kontekscie braku dostepu do sieci

    //tu powinien znalezc sie kod testujacy zachowanie aplikacji
});
```

Blok then pozwala zdefiniować co nastąpi po tym, jak sieć zostanie wyłączona. Jest to bardzo ważny aspekt biblioteki. Mamy pewność, że przez cały czas działania funkcji zachodzić będzie określony warunek. Warto zwrócić uwagę na przykładzie, że o ile w ramach kontektu stan sieci jest precyzyjnie zdefiniowany, to poza tymi ramami już tak nie musi być, stan możmy traktować jako dynamiczny lub nieokreślony. Dodatkowo należy pamiętać, że zmiana ustawień urządzenia w ramach kontekstu nie musi pociągać za sobą jego zmiany po wykonaniu testów. Ma to duże znaczenie z perspektywy testowania aplikacji, gdyż niejako wymusza, żeby w ramach jednego konteksu testować jedynie te rzeczy, które są z nim związane, a elementy interfesju które od nie zależą poza tym kontekstem.

Warto zaznaczyć, że konteksty można ze sobą łączyć, aby tworzyć bardziej złożone scenariusze. Takie złączenie realizujemy poprzez wielokrotne użycie funkcji 'then' wraz z funkcjami, które chcemy wywyłać. Pozwala to na zdefiniowanie na przykład takiego kontekstu:

```
 //example.js

Ash.noNetwork().then(
    Ash.orientationHorizontal
).then(function(arg){
    // w tym momencie mozemy byc pewni, ze zachodza dwa warunki:
    // 1) dostep do sieci zostal odciety
    // 2) ekran znajduje sie w pozycji horyzontalnej

    //tu powinien znalezc sie kod testujacy zachowanie aplikacji
});
```

możemy także wywołania funckcji Ash przeplatać innym kodem JavaScript np. w ten sposób:

```
 //example.js

Ash.orientationHorizontal().then(function(msg){
    // tu mozemy umiescic kod JavaScript

    // ekran jest w pozycji horyzontalnej
```

```
    // tu mozemy umieccic kod JavaScript
    Ash.orientationVertical().then(function(){
           // tu mozemy umiescic kod JavaScript

                  // ekran jest w pozycji wertykalnej

           // tu mozemy umiescic kod JavaScript
    });
});
```

6.2 Promises

Wewnętrznie Ash korzysta z mechanizmu obietnic tzw. Promise. Promises jest to mechanizm pozwalający programistom w lepszy prostszy i bardziej przejrzysty sposób panować nad asynchronicznością. Dotychasowe podejście do kodu wykonującego się w sposób asynchroniczny wymagało oprócz parametrów wywołania przekazania także dwóch wywołań zwrotnych. Jednego wołanego w przypadku sukcesu oraz drugiego w przypadku wystąpienia błędu. Promise działają inaczej wywołanie promise zwraca obiekt o którym możemy myśleć jak o obietnicy, że kiedyś będzie on zawierał wynik jakiejś funkcji. Z jednej strony ułatwia to myślenie o asynchronicznych wywołaniach, z drugiej pozwala nam na traktowanie przyszłych zdarzeń jak obiektów. To znaczy, że możemy przypiąć do nich jedną lub wiele funkcji, które będą wołane dopiero gdy wewnętrzne obliczenia zostaną zakończone. Możliwe jest też łatwe szeregowanie obetnic, co w przypadku tradycyjnego podejścia prowadziło do tzw. "callback hell", czyli zagłębiania funkcji w funkcjach do tego stopnia że stają się one absolutnie nieczytelne.

Dokładny opis mechanizmu, jego wymagania oraz zastosowania można znaleźć na stronie projektu

http://promises-aplus.github.io/promises-spec/

Na chwilę obecną nawet w samym JavaScript istnieje wiele implementacji specyfikacji Promises. W Ash pierwotnie zastosowano bibliotekę o nazwie promises.js dostępną pod adresem

https://github.com/then/promise/

Za tym wyborem przemawiał niewielki rozmiar kodu (co ogranicza narzut związany z dodawaniem dodatkowej biblioteki do projektu korzystajacego z Ash) oraz bogactwo funkcjonalności (promises.js wiernie implementuje specyfikację oraz wzbogacając ją o kilka nowych możliwości). Niestety szybko okazało się, że funkcjonalności oferowane przez promises.js nie do końca odpowiadają wymaganiom projektu. Biblioteka nie została jednak porzucona, ale zamiast tego zaadaptowana do specyficznych potrzeb Ash.

6.3 Wzorzec Object Pages w Ash

W celu nadania testom struktury oraz zwiększenia elastyczności testów biblioteka Ash korzysta ze wzorca projektowego Object Pages znanego na przykład z biblioteki Selenium.

https://code.google.com/p/selenium/wiki/PageObjects

lub

http://docs.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern

Page Object reprezentuje stan interfejsu użytkownika, pojedynczego komponentu lub zbioru komponentów. W praktyce Ash jako Page Object traktuje każdy obiekt, który definiuje dwie bezparametrowe funkcje zwracające wartości logiczne:

- validate
- goto

Metoda validate wywoływana jest w celu sprawdzenia, czy obecny UI spełnia wymagania danego Page Object. Innymi słowy, czy metoda ta pozwala nam ustalić, czy znajdujemy się na właściwej podstronie.

Metoda goto pozwala na przejście na ekran definowany przez dany obiekt. Metoda ta może wewnątrz korzystać z validate w celu upewnienia się czy już nie znajdujemy się na właściwej stronie.

Oczywiście Page Objects można w sobie zagłębiać, na przykład jeden Page Object może zależeć od kilku mniejszych i jego metoda validate zakłada, że metody validate podstron będą spełnione.

Prosty Page Object w Ash może wyglądać w ten sposób:

```
//example.js
var somePageObject = {
    validate: function(){
        var screen = document.getElementById('elementId');
        return Ash.isVisible(screen);
     },
     goto: function(){
        if(!this.validate()) app.mySwipe.slide(0, 1);
        return true;
     }
};
```

6.4 Organizacja testów. Run oraz Play

Ash oferuje dwa podejścia do uruchamiania testów 'Run', o którym dalej będziemy nazywać "uruchamianiem testów" oraz 'Play', co dla rozróżnienia nazywiemy "odgrywaniem scenariuszy".

6.4.1 Run

Ten tryb oferuje proste API do szybkiego uruchamiania niezależnych od siebie testów. Sygnatura funkcji:

function Ash.run(array[function] testsArray, function failureCallback[, function successCallback])

Przykładowe zastosowanie:

```
//example.js

var exampleTests = [
    // tu umiesc funkcje z kodem testow
];

Ash.run(exampleTests, function(errorData){
  // ten callback zostanie uruchomiony, za kazdym razem gdy test sie nie powiedzie
}, function(successData){
  // ten callback jest opcjonalny, jesli zostanie podany to Ash uruchomi go za kazdym razem, gdy
      test sie powiedzie
});
```

Run pozwala na sekwencyjne uruchomienie przekazanych testów w kolejności zgodnej z indeksem tablicy. Ash nie czyni żadnych zabiegów, aby testy nie kolidowały i nie ingerowały w działanie innych testów, dlatego testy przekazywane winny być od siebie nawzajem niezależne. Dostarczane one są do Ash jako tablica bezparametrowych funkcji.

Ash definiuje zestaw funkcji wyzwalaczy wołanych podczas uruchomienia Run. Funkcje te znajdują się w globalnym obiekcie Ash.callbacks, ale mogą być swobodnie nadpisywane przez użytkowników. Ich wywołanie następuje w ściśle zdefiowanych momentach, dzięki czemu użytkownik Ash może lepiej kontrolować przebieg testów. Typowy przebieg wywołania Ash.run wygląda w ten sposób:

- Ash.callbacks.beforeClass
- Ash.callbacks.before
- test dostarczony przez użytkownika

- Ash.callbacks.after
- ....
- Ash.callbacks.before
- Test dostarczony przez użytkownika jest wywoływany
- Ash.callbacks.after
- Ash.callbacks.afterClass

Każdy z dostarczonych testów kończy się wywołaniem jednego z dostarczonych funckcji - obowiązkowego failureCallback wywoływanego jeśli z jakiegoś powodu test się nie powiedzie oraz opcjonalnego successCallback wołanego w momencie poprawnego wykonania się testu.

Do wywołań zwrotnych przekazywane są obiekty zawierające informacje o wykonaniu testu. Do errorCallback:

```
{
        "level": // string, wskazuje jak powazny jest problem, moze byc uzyty jako tag np. do
            filtrowania
        "code": // int, kod bledu
        "message": // string, komunikat bledu
        "url": // string, zawiera informacje w ktorym pliku doszlo do bledu
        "lineNumber": // int, zawiera informacje gdzie w pliku doszlo do bledu
}
```

Do successCallback:

```
{
        "length": // int, ile testow znajduje sie na tablicy przekazanej do run
        "index": // int, index obecnego testu w tablicy
}
```

6.4.2 Play

Play jest wyższą formą uruchamiania testów niż run, ale tylko w sensie, że bazuje na run. Najważniejszą z różnic pomiędzy run a play jest to, że play wykonuje uporządkowany i zsynchronizowany zestaw testów, w którym to zestawie istnieje ściśle określona kolejność oraz zależność pomiędzy kolejnymi testami-krokami. Play pozwala na uruchomienie przygotowanego wcześniej scenariusza.

Sygnatura funkcji play jest podobna do run:

function Ash.play(array[function] scenarioArray, function failureCallback[, function successCallback])

Scenariusze przekazujemy jako tablicę obiektów na przykład:

```
[
    {
        name: "First Step",
        where: somePageObject,
        what: [somePageTest],
        howLong: 1500 // in miliseconds
    },
    {
        name: "Second Step",
        where: otherPageObject,
        what: [otherPageTest],
        howLong: 1000 //in miliseconds
    }
];
```

każdy z obiektów musi defioniować następujące klucze:

- PageObject where - określa na jakim ekranie aplikacji ma zostać uruchomiony test
- Array[Function] what - zestaw bezparametrowych funkcji, które będą wywoływane jedna po drugiej
- Integer howLong - jak długo ma trwać wykonanie testów, jeśli przekroczony zostanie wyznaczony tu limit test zostaje uznany za niepowodzenie

dodatkowo może definiować:

- String name - nazwa kroku scenariusza, ma zastosowanie pomocnicze

Kroki scenariusza przekazane do play wykonywane są przy użyciu wcześniej opisanej metody run, jeden po drugim w kolejności od mniejszych indeksów. Po każdym pojedynczym wykonaniu porównywane są czasy realny oraz założony przekazany jako klucz 'howLong'. W przypadku, gdy czas realny jest większy niż założony test oznazany jest jako niepowodzenie. Biblioteka Ash definiuje takie surowe podejście do limitów czasowych, aby utrudnić deweloperom ignorowanie problemów z wydajnością ich aplikacji i niejako zmusić ich do zadbania o właściwą responsywność interfejsu użytkownika ich aplikacji. Page object where ma za zadanie ułatwić nawigację po aplikacji pomiędzy testami, tzn. test z danego kroku uruchamainy jest tylko wtedy, gdy znajdziemy się na ekranie związanym z danym testem. Kwestie sukcesu i niepowodzenia play są identyczne jak w przypadku run - przekazujemy failureCallback, który jest wywoływany po niepowodzeniu kroku oraz opcjonalny successCallback, który jest wołany gdy krok scenariusza powiedzie się.

Przykładowe wykonanie może wyglądać tak:

- Ash.callbacks.beforeClass
- scenario[0].where.goto
- scenario[0].where.validate
- Ash.run(scenario[0].what)
- porównanie czasu scenario[0].howLong z czasem wywołania scenario[0].what
- ....
- scenario[n].where.goto
- scenario[n].where.validate
- Ash.run(scenario[n].what)
- porównanie czasu scenario[n].howLong z czasem wywołania scenario[n].what
- Ash.callbacks.afterClass

Do wywołań zwrotnych przekazywane są obiekty zawierające informacje o wykonaniu testu, identycznie jak w przypadku metody run.

6.5 Obsługa błędów

Ash do obsługi błędów wykorzystuje wydarzenie 'window.onerror', które ma miejsce, kiedy wyjątek (lub błąd) nie zostanie złapany na żadnym poziomie stosu wywołania. Podczas uruchomienia zestawu testów korzystając z metod run lub play Ash podmienia zastane window.onerror na własną implementację. Zadaniem naszej implementacji jest przechwycenie błedu, jego obsłużenie łącznie w wywołaniem odpowiedniego wywołania zwrotnego oraz dalsze wykonywanie testów.

Każdy z obiektów rzucanych jako wyjątek jest ustandaryzowany i zawiera następujące pola:

```
{
        "level": // string, wskazuje jak powazny jest problem, moze byc uzyty jako tag np. do
            filtrowania
        "code": // int, kod bledu
        "message": // string, komunikat bledu
}
```

7 Wtyczki

Apache Cordova udostępnia swoje API na zasadzie wtyczek. Każda wtyczka może być indywidualnie instalowana oraz udostępnia określone API i związane z nim funkcjonalności. Takie podejście pozwala na rozbicie monolitycznego API na mniejsze części oraz zapewnia możliwość rozszerzenia platformy poprzez własne wtyczki.

Tak jest właśnie w przypadku Ash, który jest zaimplementowany jako wtyczka do platformy.

Więcej informacji można znaleźć w oficjalnej instrukcji dla twórców wtyczek:

http://cordova.apache.org/docs/en/edge/guide_hybrid_plugins_index.md.html#Plugin%20Development%20Guide
http://cordova.apache.org/docs/en/edge/guide_platforms_android_plugin.md.html#Android%20Plugins

7.1 Klasa CordovaPlugin

CordovaPlugin jest to klasa, która jest rozszerzana przez każdą wtyczkę. udostępnia ona dostęp do najważniejszych składowych aplikacji:

- CordovaWebView webView - komponent WebView w którym uruchomiona jest aplikacja
- CordovaInterface cordova - daje dostęp do zasobów platofrmy np. Activity oraz puli wątków

TODO prosty diagram klas dla commitu b872df0f314194ad50cbaa098ebbf717e53bb354
Punktem wejściowym dla kodu każdej wtyczki jest jest metoda exec:

```
public boolean execute(String action, JSONArray args, final CallbackContext callbackContext) throws
    JSONException
```

To właśnie ta metoda jest wywoływana, gdy warstwa Javascript zażąda wywołania metody.

7.2 Native bridge. Integracja z Apache Cordova

7.3 Wielowątkowość

8 Implementacja

Ten rozdziała zawiera informacje na temat szczegółów implementacyjnych biblioteki Ash.

8.1 Zmiana położenia ekranu

```
Ash.orientationHorizontal().then( ... )
Ash.orientationVertical().then( ... )
```

Konteksty orientationX działają na zasadzie przełączania ekranu z jednego położenia do innego. Na platformie Android jest to realizowane przez funkcję setRequestedOrientation klasy Activity do której uzyskujemy dostęp przez interfejs cordova.

```
this.cordova.getActivity().setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

Każdemu z widoków odpowiada stała zdefiniowana przez platformę Android

- orientationHorizontal - ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
- orientationVertical - ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

Obecna implementacja po zakończeniu testów zawartych w kontekście nie przywraca zastanego stanu ekranu.

8.2 Dostęp do sieci

8.3 Dostęp do systemu plików

8.4 Symulacja ruchu

8.5 Obsługa przycisku powrotu

```
Ash.pressBack().then( ... )
```

Ogólnie rzecz biorąc celem metody pressBack() jest powrót do poprzedniego ekranu w historii aplikacji. W praktyce mowa tutaj o adresach URL ładowanych przez WebView w czasie działania aplikacji. Większość aplikacji Cordova tworzona jest na zadadzie Single Page Apps, więc w poszczególne adresy będą różnić się jedynie wartością tzw. hashu a przejścia pomiędzy nimi będą polegać na płynnym przejściu, raczej niż załadowaniu nowej podstrony. Dane na temat historii przeglądania zbierane są w kontrolce WebView w formie stosu.

Na platformie Android aby spowodować powrót do poprzedniego ekranu należy wywołać lub przeciążyć metodę onBackPressed klasy Activity. W tym przypdaku to podejście nie działa, gdyż Activity w którym uruchomiona jest wtyczka jest inne niż to w którym działa aplikacja. Cofnięcie się w historii tego Activity prowadzi do zakończenia aplikacji.

Aby uniknąć tego typu problemów należy cofnąć się obrębie WebView a nie aktywności. Klasa CordovaPlugin daje dostęp do CordovaWebView. Ta klasa z kolei udostępnia metodę

```
// CordovaWebView.java
/**
 * Go to previous page in history. (We manage our own history)
 *
 * @return true if we went back, false if we are already at top
 */
public boolean backHistory()
```

pozwalającą na dostęp do historii i powrotu do poprzedniego adresu URL zapisanego na stosie odwiedzonych. Operowanie na zasobach związanych z działaniem aplikacji PhoneGap nie jest dozwolone, gdyż oznaczałoby to modyfikację zasobów z jednego wątku w ramach innego wątku. Co z oczywistych przyczyn prowadzi do awarii aplikacji. Dlatego też nasze operacje musimy wykonywać na wątku UI korzystając z metody runOnUiThread.

Na poziomie API wtyczka zwraca Promise, które jest rozwiązane gdy uda się wrócić do poprzedniego adresu ze stosu i odrzucony w systuacji, gdy stos jest pusty lub dojdzie do błędu w czasie wykonywania kodu wtyczki.

9 Narzędzia oraz środowisko programisty

9.1 Rozpraszanie wykonania testów

9.2 Tworzenie laboratorium urządzeń

10 Dalsze kierunki rozwoju