УДК 004.94
Бортник К. Я., Волошин М. М.
Луцький національний технічний університет

СТВОРЕННЯ МУЛЬТИПЛАТФОРМНИХ ДОДАТКІВ ЗА ДОПОМОГОЮ JAVASCRIPT

Бортник К. Я., Волошин М. М. Створення мультиплатформних додатків за допомогою JAVASCRIPT. У даній статті оглянуто переваги та недоліки мультиплатформних додатків написаних з використанням мови програмування JavaScript. Виконано порівняння існуючих засобів для створення таких додатків.
Ключові слова: JavaScript, мультиплатформні додатки, веб-технології, мобільна розробка

Бортник К. Я., Волошин М. М. Создание мультиплатформенной приложений с помощью JAVASCRIP. В данной статье рассмотрено преимущества и недостатки мультиплатформенных приложений написанных с использованием языка программирования JavaScript. Выполнено сравнение существующих инструментов для создания таких приложений.
Ключевые слова: JavaScript, мультиплатформенные приложения, веб-технологии, мобильная разработка

Bortnyk K. Y., Voloshyn M. M. Creating a multiplatform applications with JAVASCRIPT. This article describe the advantages and disadvantages of multiplatform applications written with the programming language JavaScript. A comparison of existing tools to create such applications.
Keywords: JavaScript, cross-platform applications, Web-technologies, mobile development

Мультиплатформне програмне забезпечення – це програмне забезпечення, що може працювати на більше ніж одній апаратній платформі або операційній системі. Його можна поділити на два види: перший вимагає незалежної збірки або компіляції, другий – може запускатись безпосередньо на будь-яких платформах без попередніх підготовок, таке програмне забезпечення створюється на інтерпретованих мовах програмування.

Створювати програми таким чином доцільно лише коли планується їх видавництво на більше ніж одній платформі. Це пропонує єдину спільну частину коду, за рахунок чого тривалість розробки менша ніж при створенні на кожну платформу окремо. Розробка програми ведеться на одній технології, тому в цьому випадку не доведеться наймати відповідних фахівців при розширені підтримуваних систем.

Разом з перевагами, такого підходу наявні недоліки та складнощі:

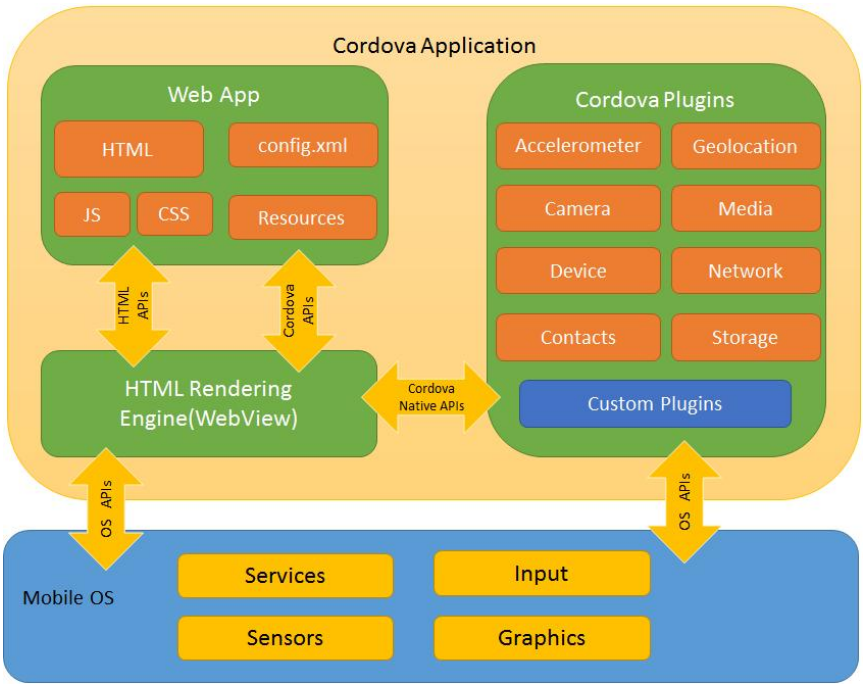
1) тестування дещо складніше, оскільки поведінка платформи при одній і тій ж події відрізняється від інших. Це є потенційним місцем де можуть міститись неочевидні та слабо помітні помилки.
2) часто використовувані засоби не можуть забезпечити повним, передбаченим системою, функціоналом. Але такі обмеження вирішуються написанням відсутніх частин платформно-орієнтованому коді.
3) кожна платформа має унікальний графічний інтерфейс, з яким програма не завжди гармонічно вписуватиметься. Слідування вимогам дизайну платформи робить програму більш передбаченою і послідовною, простішою у використанні.
4) суттєвою проблемою також є швидкодія та ефективність таких програм. Інтерпретатори та віртуальні машини повинні бути трансльовані в виконавчий код кожного разу коли програма запускається. Це потребує накладних витрат.

Основними критеріями вибору способу написання прикладних додатків є: знання платформ на яких вони повинні працювати, врахування можливих складностей реалізації функціоналу на тій чи іншій платформі, бюджет розробки.

У мобільній розробці використовуються мови програмування Objective-C, Swift, Java, C# – такі програми називають рідними (від англ. – native) або платформно-орієнтованими. Додатки, що пишуться не лише на рідному коді, зокрема на JavaScript, називаються гібридними.

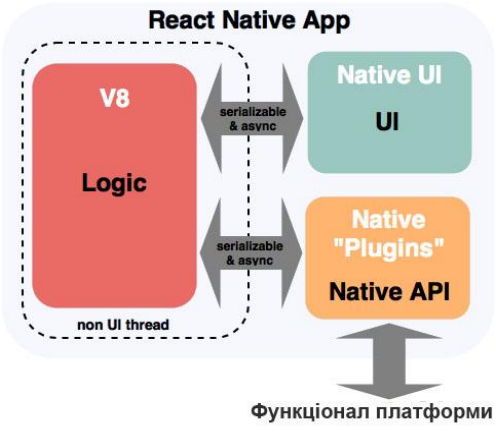
Такий підхід широко застосовується у багатьох програмах, в тому числі Facebook, Instagram, Uber, Google Maps, Discovery VR. Для розробки створені спеціальні набори інструментів: Apache Cordova, PhoneGap, React Native, Electron, Phaser, Titanium. Частина цих технологій використовує спеціальний компонент WebView (UIWebView у iOS), що дозволяє вміщувати веб-сторінки і викликати функції з рідного коду платформи. Він забезпечує роботу звичних веб-технологій всередині додатків.

Apache Cordova формує у WebView контент, для взаємодії з ресурсами платформ (такі як камера, мережа, сховище зберігання, контакти та ін.) передбачені плагіни. Схематичне зображення архітектури, що пропонує Cordova, наведено нижче.



Бібліотека Native script дозволяє робити мультиплатформні додатки, використовуючи XML, CSS, JavaScript. Native script для формування представлення використовує елементи рідного інтерфейсу. Доступ до таких ресурсів як камера, gps забезпечується з коробки. Варто зауважити, що такі додатки працюватимуть на пристроях починаючи з Android 4.2 і вище, та iOS 7.1 і вище.

В React Native (RN) на противагу Apache Cordova логіка програма пишеться і працює на JavaScript, при тому, що інтерфейс повністю рідний. В Cordova при виникнені нової події потік блокується і передається управління на JavaScript-код, очікуючи його інструкцій, React Native виконує код в окремому фоновому потоці, взаємодіючи з головним потоком асинхронно, тобто в потоці збирається ряд команд до головного і певний момент відправляється згрупований запит. Загальна схема роботи зображена нижче.



React Native забезпечує компонентний підхід до розробки, завдяки цьому додатки легко розширювати як власними компонентами, так і великою кількістю сторонніх. Передбачені такі готові компоненти: ActivityIndicator, Button, DatePickerIOS, DrawerLayoutAndroid, Image, KeyboardAvoidingView, ListView, MapView, Modal, Navigator, NavigatorIOS, Picker, PickerIOS,

ProgressBarAndroid, ProgressViewIOS, RefreshControl, ScrollView, SegmentedControlIOS, Slider, SnapshotViewIOS, StatusBar, Switch, TabBarIOS, TabBarIOS.Item, Text, TextInput, ToolbarAndroid, , TouchableWithoutFeedback, View, ViewPagerAndroid, WebView.

Оскільки платформи різні, то і компоненти, відповідно, відрізняються. Завдяки компонентам, що є абстракціями над рідним функціоналом, розмітка HTML та стилі CSS не використовуються для формування представлення. Це забезпечує більшу продуктивність ніж при використанні WebView. У випадку відсутності потрібного компоненту необхідно створювати власний на рівні платформи.

RN підтримує дві мобільні платформи – iOS та Android. Для розробки доступні як спільні для обох систем компоненти, так і індивідуальні.

React Native містить вбудований транслятор JavaScript специфікації ECMAScript 2015, завдяки якому код, написаний на найновішій реалізації, трансформується у поточний.

Також доступне зручне налагодження програм в браузері Chrome і миттєве перезавантаження. Для цього потрібно відкрити меню розробника і поставити галочки в пунктах Debug in Chrome і Enable Live Reload. Після чого можна побачити вивід console.log і відслідковувати помилки з консолі браузера. А миттєве перезавантаження пришвидшує та спрощує процес розробки, оскільки зроблені зміни в коді негайно застосовуються.

Нижче наведено приклад коду, що формує текст по центру екрану.

```
import React from 'react';
import {
  AppRegistry,
  StyleSheet,
  Text,
  View
} from 'react-native';

class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.mainText}>Simple text</Text>
      </View>
    )
  }
}
var styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  mainText: {
    fontSize: 20,
    textAlign: 'center',
  },
});

AppRegistry.registerComponent('App', () => App);
```

При розробці важливо передбачати розвиток функціоналу програми, і слідувати підходящій архітектурі. Компоненти повинні бути пов'язаними між собою правильним чином. Робота компонентів залежить від подій та даних. Існують два типи даних, що керуються компонентом: властивості props і стан state. Властивості props встановлюються батьківським компонентом і є незмінними протягом всього життєвого циклу компонента. Для даних, які можуть змінюватися з плином часу, потрібно використовувати state. Для спрощення управління потоком даних можна використовувати контейнер стану, такий як Redux.

Для настільних операційних систем також існує кілька інструментів створення програм на JavaScript. Гібридні програми для настільних комп'ютерів використовують Webkit та Node.js.

Electron містить в собі браузер Chromium та Node.js. Обробка коду та генерація представлення відбувається в Chromium, а робота з операційною системою – через Node.js.

Принцип роботи Electron заснований на двох типах процесів:

1) основний процес, який відповідає за інтеграцію та взаємодію з графічним інтерфейсом операційної системи. Такий процес може бути запущений тільки один раз на весь період роботи додатка.
2) процес формування представлення, що відповідає за відображення вікна браузера, це може бути сторінка додатка або будь-яка інша веб-сторінка. Таких процесів може бути декілька. За рендеринг вмісту відповідає основний процес.

На відміну від мобільних додатків, розмір таких програм буде значно більшим, оскільки міститиме повноцінний веб-браузер.

Підтримуються операційні системи OSX, Linux, Windows.

Підсумовуючи вище написане, можна сказати, що розробка мультиплатформних програм з використанням JavaScript є доцільною у багатьох випадках.

Література

1. React Native [Electronic resource] / React Native. – Mode of access: https://facebook.github.io/react-native/
2. Kyle Simpson. You Don't Know JS: ES6 & Beyond. – O`Reilly Media, 2015. – 228 p.
3. Bonnie Eisenman. Building Native Mobile Apps with JavaScript, 2015. – 272 p.
4. Architectural overview of Cordova platform [Electronic resource] / Apache Cordova. – Mode of access: https://cordova.apache.org/docs/en/latest/guide/overview/index.html