DSRT Course Material 2015

Makzan

Version 1.0
October 4, 2015

Table of Contents

This is the course material I created for the cross-platform development class. in this course, we will explore different approaches and choices to build app for different platforms, mainly iOS, Android and web.

Chapter 1. Introduction

Getting ready.

1.1. How this course is constructed

There are different individual topics that we are going to cover in this course. You can find the course entry point at http://page.sh/dsrt2015

The entry point provides the outline links to each individual topic. It also comes with a log to show the changes I have made from time to time.

1.2. About this course

In my part of the course, we focus on different web technologies to create mobile web app.

You'll need to know basic HTML, CSS, and Javascript.

We will also explore how we can make use of native platform resources to create hybrid app.

You can refer to my past course materials on makzan.net for some basic skills.

1.3. What type of app you need to build?

Different types of apps requires different solutions which involves of different technologies.

You need to learn the advantages of each technology and know where to use the right one for different situations.

Roughly, we have the following common types of applications.

We explore several common types and see what technologies we shall use to make the app.

- Informative app

One common type is informative app. That the app organize and display information in a easy to read hierarchy.

- Utilities

Small utilities

- Productivity tools
- Photography

- Games

It depends. For serious game, obviously native gains the performance advantages. But for casual games, we can use canvas, webGL from HTML5 to build a smooth game.

- e-book

The web is good for displaying interactive content.

But be aware that if we need the

- Music

Web audio supports isn't really good. So native is really the technology we want to build music playback application.

For example, we sure need the iOS native API to implement the background playback function to allow listening to the music when the phone is locked.

But we actually can still make use of the web technologies somewhere. For example, we can build the interface in web view. There are connections between web and the native API.

- Entertainment

1.4. Why web technology?

Every device have a browser * not all. Apple TV OS doesn't have WebKit component.

We are familiar with web technology: HTML, CSS, JavaScript

1.5. Expectation: App vs web

- Get things done
- Usability
- Platform specific
- Device hardware
- Default offline
- Informative
- Hierarchy
- Readability

- Deep linking

You won't think that a website works in iPhone but not android. But you will find apps working on 1 platform only. Also, you think you can use any device to see the website and get what information you want

1.5.1. Web for app?

In my opinions, we will never archive app experience with web technology. That's same that we will never archive web's culture with app technology. They are different things. They are designed for different purposes. But, we can combine both by taking their advantages and avoiding their weaknesses.

1.5.2. The process of building native app vs web app

the graph

Using the pure web technology to build app is fast.

Using native technology to build app seemed to eb slow, but in the long run, it has more stable curve.

Actually there is no such thing about app must win or web must win. They can both win in their places and be campanion to each other.

Chapter 2. Getting sensor value

In this chapter, we explore different JavaScript API to get device sensors.

2.1. Getting device information in web

Getting accelerometer

Getting orientation

Getting gyroscope

Accepting photo upload

2.1.1. Getting orientation

2.1.2. Getting Gyroscope & Accelerometer

Getting raw data

```
window.addEventListener('devicemotion', function (e) {
  this.acceleration = e.acceleration;
};
```

The code example

We use a list to display the value history.

```
<ul id="x" class="history">
  <li>No Data Yet</li>
</ul>
```

We use an array to store the history of the value. And set a limit count to the array.

```
app.xValues = [];
window.addEventListener('devicemotion', function (e) {
  app.xValues.push(e.acceleration.x);
  var listLimit = 15;
  if (app.xValues.length > listLimit) {
    app.xValues.splice(0, 1); // remove the old records when list too long.
  }

  app.view.update();
});
```

```
app.view = {}
app.view.update = function(){
  this.renderHistory($('#x'), app.xValues);
}

app.view.renderHistory = function(element, data) {
  $(element).empty();
  for (var i = app.xValues.length - 1; i >= 0; i--) {
    $(element).append("<li>" + Math.floor(data[i]*1000) + "</li>");
  };
}
```

Note that this code only shows the handling of X value. There are Y and Z value in accelerometer sensor that share very similar logic.

2.1.3. Visualize the accelerometer value history a pie chart

We will put a canvas in each axis.

```
<div class='row'>
  <h2>X</h2>
  <canvas id='x-canvas' width='80' height='40'></canvas>
  <ul id='x' class='history'>
    <li>No Data Yet</li>
  </ul>
</div>
```

We will use a graph to show the historical values of the sensor.

```
// Graph
(function(){
  var app = this.app || (this.app={});

  // A bar chart for - Max Value to + Max Value.
  app.Graph = (function(){
    function Graph(canvasId, maxValue){
      this.canvas = document.getElementById(canvasId);
      if (this.canvas) {
        this.context = this.canvas.getContext('2d');
      }

      // set the Y unit scale.
      this.unit = this.canvas.height / 2 / maxValue; // divide by 2 because half for +
and half for -.
    }
    Graph.prototype.drawBar = function(x, value) {
      var c = this.context;
      c.beginPath();
      c.moveTo(x, this.canvas.height/2); // middle
      c.lineTo(x, this.canvas.height/2 - value * this.unit);
      c.lineWidth = 3;
      c.strokeStyle = "#222";
      c.stroke();
      c.closePath();
    };
    Graph.prototype.render = function(dataArray) {
      this.canvas.width = this.canvas.width;
      for (var i=0, len=dataArray.length; i<len; i++) {
        this.drawBar(i*5, dataArray[i]);
      }
    }
    return Graph;
  })();
}).call(this);
```

At last, we use the Graph to render the historical data.

```
var graphX = new app.Graph('x-canvas', 2.5);
graphX.render(app.xValues);
```

2.1.4. Inspecting the gyroscope sensor

You can test the app here: http://mztests.herokuapp.com/rotation/

The different is just change to listen to `deviceorientation` and use `e.alpha`, `e.beta`, `e.gamma` for the rotation axises.

```
window.addEventListener('deviceorientation', function (e) {
  app.xValues.push(e.alpha);
  app.yValues.push(e.beta);
  app.zValues.push(e.gamma);
  // ... code that is very similar to the accelerometer section.
}
```

Chapter 3. Mobile Navigation

3.1. Mobile navigation

3.1.1. Navigation difference between platform

The difference is huge Windows, Android, iOS It’s not about the style (skepticism or "flat", or depth) It’s about the user habits. It’s about the strategy (tabs on bottom, tabs at top, infinite canvas) It’s about the hardware buttons that affects the software design

3.1.2. Hamburger menu?

Conclusion: avoid it. At least don’t just put the hamburger menu there.

3.2. Case study navigation—Facebook

First 9 grid Then list view Then hamburger menu Then double hamburger menu Then removed main hamburger menu and back to tab

Chapter 4. Mobile Navigation

4.1. Accessibility—Making the web inclusive

One difference between web and app is that web pages should be designed to be inclusive. It doesn’t and never lock into perticular platform. It’s designed so that every one can get the information. This is why we need to handle accessibility in our web pages.

4.2. ARIA roles

4.2.1. Common ARIA Roles

- For structure
- For buttons
- For UI Components

4.2.2. UI Components

- Dialog
- Tab and Tabs Content

4.3. Color blind