Comp 388/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2015 - Week 13

Dr Nick Hayward

Contents

Final Presentation & Report

- team presentation on 4th December @ 2.45pm
- team report due on 11th December by 5.15pm

Final Assessment Outline

- continue to develop your app concept and prototypes using Apache Cordova
- implement a custom Cordova plugin for either of the following native Mobile OSs
    - *Android*
    - *iOS*
    - *Windows Phone*
- working app
- explain design decisions
    - *outline what you chose and why?*
    - *what else did you consider, and then omit? (again, why?)*
- which platform/s did you choose, and why?
- which concepts could you abstract for easy porting to other platform/OS?
- describe patterns used in design of UI and interaction

Cordova app - Plugins

intro

- developing custom plugins for Cordova, and by association your apps
    - *a useful skill to learn and develop*
- it is not always necessary to develop a custom plugin
    - *to produce a successful project or application*
    - *dependent upon the requirements and constraints of the project itself*
- use and development of Cordova plugins is not a recent addition
- with the advent of Cordova 3 plugins have started to change
    - *introduction of Plugman and the Cordova CLI helped this change*
- plugins are now more prevalent in their usage and scope
    - *their overall implementation has become more standardised*

Cordova app - Plugins

structure and design - part 1

- as we start developing our custom plugins
  - *makes sense to understand the structure and design of a plugin*
- what makes a collection of files a plugin for use within our applications
- we can think of a plugin as a set of files
  - *as a group extend or enhance the capabilities of a Cordova application*
- already seen a number of examples of working with plugins
  - *each one installed using the CLI*
  - *its functionality exposed by a JavaScript interface*
- a plugin could interact with the host application without developer input
- majority of plugin designs provide access to the underlying API
  - *provide additional functionality for an application*

Cordova app - Plugins

structure and design - part 2

- a plugin is, therefore, a collection of contiguous files
  - *packaged together to provide additional functionality and options for a given application*
- a plugin includes a `plugin.xml` file
  - *describes the plugin*
  - *informs the CLI of installation directories for the host application*
  - *where to copy and install the plugin's components*
  - *includes option to specify files per installation platform*
- a plugin also needs at least one JavaScript source file
  - *file is used within the plugin*
  - *helps define methods, objects, and properties required by the plugin*
  - *source file is used to help expose the plugins API*

Cordova app - Plugins

structure and design - part 3

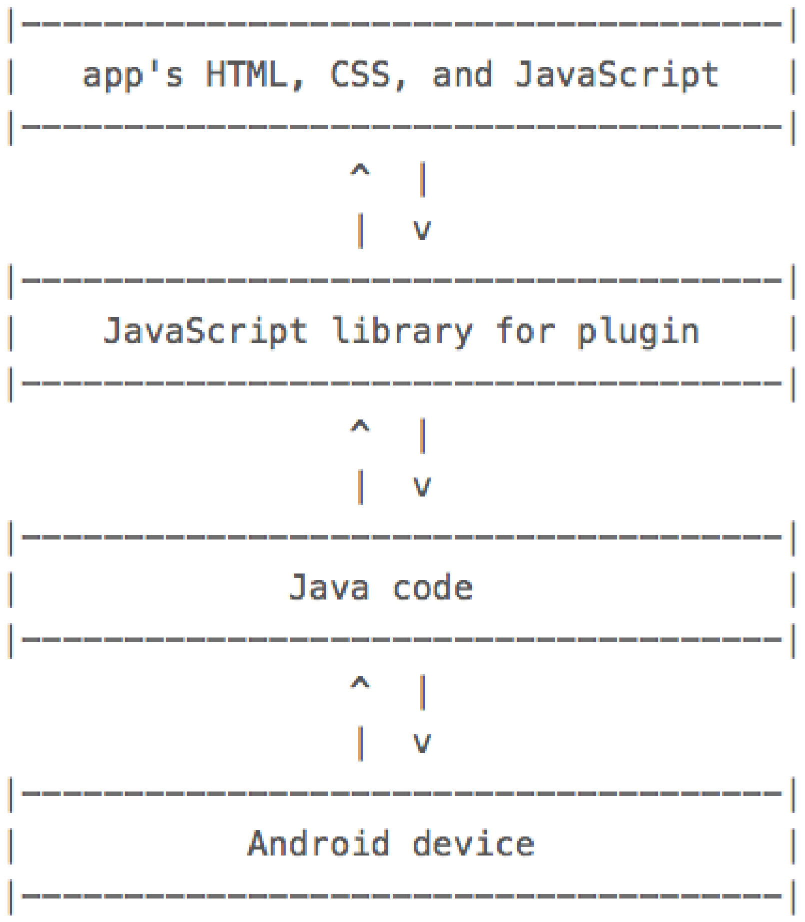
- within our plugin structure
  - *easily contain all of the required JS code in one file*
  - *divide logic and requirements into multiple files...*
- structure depends on plugin complexity and dependencies
- eg: we could bundle other jQuery plugins, handlebars.js. maps functionality...
- beyond the requirement for a `plugin.xml` and plugin JS source file
  - *plugin's structure can be developer specific*
- for most plugins, we will add
  - *native source code files for each supported mobile platform*
  - *may also include additional native libraries*
  - *any required content such as stylesheets, images, media...*

Cordova app - Plugins

architecture - Android

- we can choose to support one or multiple platforms for an application
- consider a plugin for Android
  - *we can follow a useful, set pattern for its development*
- android plugin pattern
  - *application's code makes a call to the specific JS library, API*
  - *plugin's JS then sends a request down the chain*
  - *request sent to specific Java code written for supported versions of Android*
  - *Java code communicates with the native device*
  - *upon success, any return is then handled*
  - *return passed up the plugin chain to the app's code for Cordova*
- bi-directional flow from the Cordova app to the native device, and back again

Image - Cordova Plugin Architecture - Android



```
|-------------------------------------|
|   app's HTML, CSS, and JavaScript   |
|-------------------------------------|
                ^  |
                |  v
|-------------------------------------|
|    JavaScript library for plugin    |
|-------------------------------------|
                ^  |
                |  v
|-------------------------------------|
|              Java code              |
|-------------------------------------|
                ^  |
                |  v
|-------------------------------------|
|            Android device           |
|-------------------------------------|
```

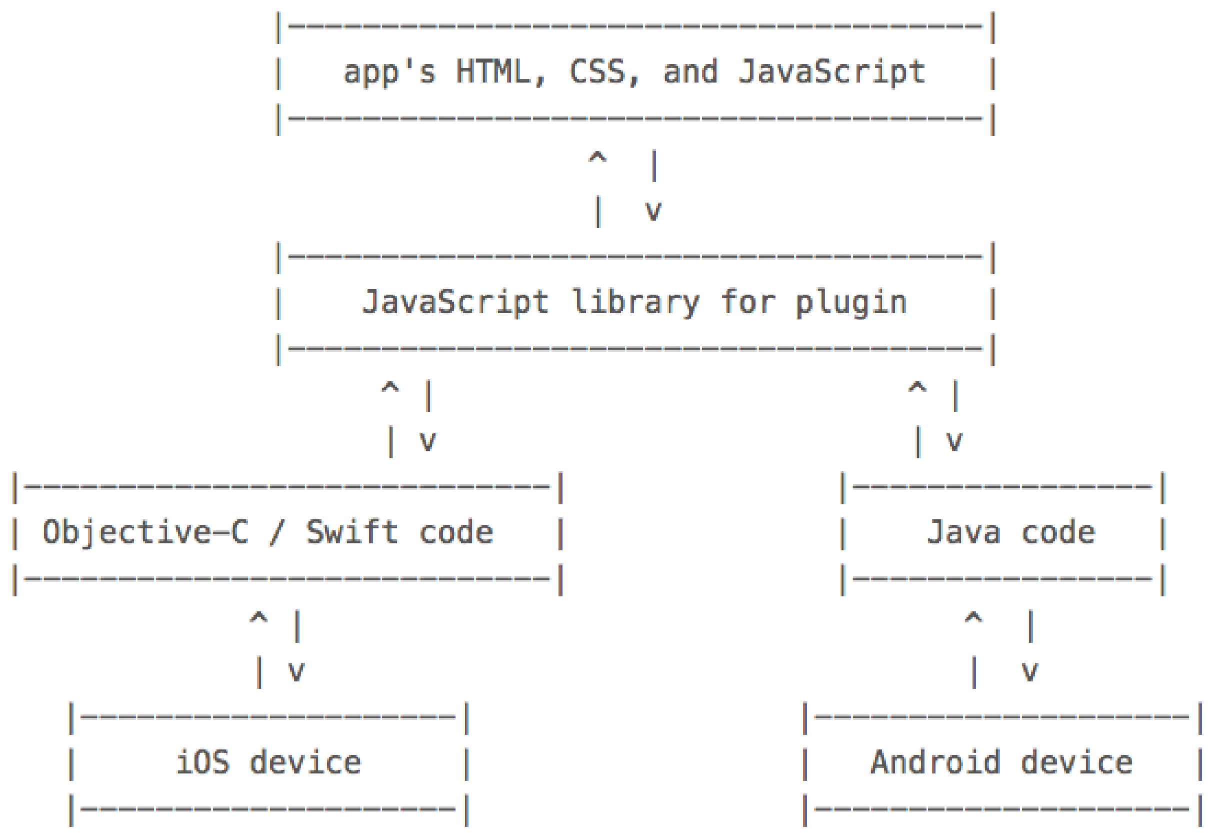
Cordova Plugin Architecture - Android

Cordova app - Plugins

architecture - cross-platform

- update our architecture to support multiple platforms within our plugin design
- maintain the same exposed app content
  - *again using HTML, CSS, and JavaScript*
- maintain the same JavaScript library, API for our plugin
- add some platform specific code and logic for iOS devices
  - *add native Objective-C/Swift code and logic*
- inherent benefit of this type of plugin architecture
  - *the plugin's JavaScript library*
- as we support further platforms
  - *plugin's JavaScript library should not need to change per platform*

Image - Cordova Plugin Architecture - Cross-platform


app's HTML, CSS, and JavaScript
JavaScript library for plugin
Objective-C / Swift code
Java code
iOS device
Android device

Cordova Plugin Architecture - Cross-platform

Cordova app - Plugins

Plugman utility - part 1

- for many plugin tasks in Cordova we can simply use the CLI tool
- we can also use the recent *Plugman* tool
  - *useful for the platform-centric workflow*
- *Plugman* tool helps us develop custom plugins
  - *helps create simple, initial template for building plugins*
  - *add or remove a platform from a custom plugin*
  - *add users to the Cordova plugin registry*
  - *publish our custom plugin to the Cordova plugin registry*
  - *likewise, unpublish our custom plugin from the Cordova plugin registry*
  - *search for plugins in the Cordova plugin registry*

Cordova app - Plugins

Plugman utility - part 2

- need to install *Plugman* for use with Cordova
  - *use NPM to install this tool*

```
npm install -g plugman
```

- OS X may need `sudo` to install
- `cd` to working directory for our new custom plugin
  - *now create the initial template*

```
plugman create --name cordova-plugin-test --plugin_id org.csteach.plugin.Test --plugin_ver
```

- with this command, we are setting the following parameters for our plugin
  - *`--name` = the name of our new plugin*
  - *`--plugin_id` = sets an ID for the plugin*
  - *`--plugin_version` = sets the version number for the plugin*
- also add optional metadata, such as author or description, and path to the plugin...
- new plugin directory containing
  - *`plugin.xml`, www directory, `src` directory*

Cordova app - Plugins

Plugman utility - part 3

- using `plugman`, we can also add any supported platforms to our custom plugin

```
// add android
plugman platform add --platform_name android
// add ios
plugman platform add --platform_name ios
```

- command needs to run from the working directory for the custom plugin
- template creates plugin directories

```
|- plugin.xml
|- src
   |- android
      |- Test.java
|- www
   |- test.js
```

- three important files that will help us develop our custom plugin
  - *`plugin.xml` file for general definition, settings...*
  - *`Test.java` contains the initial Android code for the plugin*
  - *`test.js` contains the plugin's initial JS API*

Cordova app - Plugins

Plugman utility - part 4

- now update plugin's definition, settings in `plugin.xml` file
  - *helps us define the general structure of our plugin*
- within the `<plugin>` element, we can identify our plugin's metadata
  - *`<name>`, `<description>`, `<licence>`, and `<keywords>`*
- need to clearly define and structure our JS module
  - *corresponds to a JS file for our plugin*
  - *helps expose the plugin's underlying JS API*
- `<clobbers>` element is a sub-element of `<js-module>`
  - *inserts JS object for plugin's JS API into application's window*
- update `target` attribute for `<clobbers>` adding required window value

```
<clobbers target="window.test" />
```

- now corresponds to object defined in `www/test.js` file
- exported into app's window object as `window.test`
  - *access underlying plugin API using this `window.test` object*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 1

- majority of Cordova plugins include native code
  - *for platforms such as Android, iOS, Windows Phone...*
  - *not a formal requirement for plugins*
- start by developing our custom plugin using JavaScript
  - *eg: create a custom plugin to package a JavaScript library*
  - *or a combination of libraries*
  - *create a structured JS plugin for our application*
- start by creating a simple JavaScript only plugin
  - *helps demonstrate plugin development*
  - *general preparation and usage*
- need to quickly update our `plugin.xml` file
  - *correctly describe our new plugin*

```
<description>output a daily random travel note</description>
```

Cordova app - Plugins

Test plugin 1 - JS plugin - part 2

- now start to modify our plugin's main JS file, `www/test.js`
- use this JS file to help describe the plugin's primary JS interface
  - *developer can call within their Cordova application*
  - *helps them leverage the options for the installed plugin*
- by default, when Plugman creates a template for our custom plugin
  - *includes the following JS code for `test.js` file*

```
var exec = require('cordova/exec');

exports.coolMethod = function(arg0, success, error) {
    exec(success, error, "test", "coolMethod", [arg0]);
};
```

Cordova app - Plugins

Test plugin 1 - JS plugin - part 3

- part of the default JS code
  - *created based upon the assumption we are creating a native plugin*
  - *eg: for Android, iOS platforms...*
- loads the `exec` library
  - *then defines an export for a JS method called* `coolMethod`
- as we develop a native code based plugin for Cordova
  - *need to provide this method for each target platform*
- working with a JS-only plugin, simply export a function for our own plugin
- now update this JS file for our custom plugin

```
module.exports.dailyNote = function() {
return "a daily travel note to inspire a holiday...";
}
```

- to be able to use this plugin
  - *a Cordova application simply calls* `test.dailyNote()`
  - *the note string will be returned*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 4

- simply exposing one test method through the available custom plugin
- easily build this out
  - *expose more by simply adding extra exports to the* `test.js` *file*
- also add further JS files to the project
  - *also export functions for plugin functionality*
- need to update our plugin to work in an asynchronous manner
  - *a more Cordova like request pattern for a plugin*
- when the API is called
  - *at least one callback function needs to be passed*
  - *then the function can be executed*
  - *then passed the resulting value*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 5

```
module.exports = {

  // get daily note
  dailyNote: function() {
      return "a daily travel note to inspire a holiday...";
  },

  // get daily note via the callback function
  dailyNoteCall: function (noteCall) {
    noteCall("a daily travel note to inspire a holiday...");
  }
};
```

- exposing a couple of options for requests to the plugin
- now call `dailyNote()`
  - *get the return result immediately*
- call `dailyNoteCall()`
  - *get the result passed to the callback function*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 6

- now need to test this plugin, and make sure that it actually works as planned
- first thing we need to do is create a simple test application
  - *follow the usual pattern for creating our app using the CLI*
  - *add our default template files*
  - *then start to add and test the plugin files*

```
cordova create customplugintest1 com.example.customplugintest1 customplugintest1
```

- also add our required platforms,

```
cordova platform add android
```

Cordova app - Plugins

Test plugin 1 - JS plugin - part 7

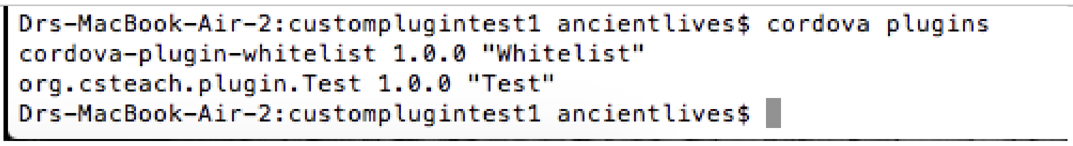
- we can then add our new custom plugin

```
cordova plugin add ../custom-plugins/cordova-plugin-test
```

- currently installing this plugin from a relative local directory
- when we publish a plugin to the Cordova plugin registry
  - *install custom plugin using the familiar pattern for standard plugins*
- we can now check the installed plugins for our custom plugin

```
cordova plugins
```

Image - Cordova Custom Plugin



```
Drs-MacBook-Air-2:customplugintest1 ancientlives$ cordova plugins
cordova-plugin-whitelist 1.0.0 "Whitelist"
org.csteach.plugin.Test 1.0.0 "Test"
Drs-MacBook-Air-2:customplugintest1 ancientlives$
```

Cordova Installed Plugins

Cordova app - Plugins

Test plugin 1 - JS plugin - part 7

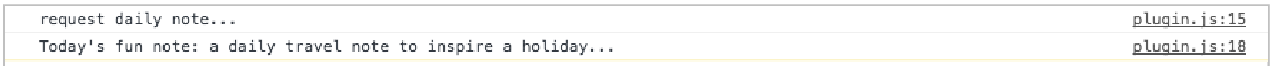
- now need to setup our home page,
- add some jQuery to handle events
- then call the exposed functions from our plugin
- start by adding some buttons to the home page

```
<button id="dayNote">Daily Note</button>
<button id="dayNoteSync">Daily Note Async</button>
```

- then update our app's `plugin.js` file
  - *include the logic for responding to button events*
  - *then call plugin's exposed functions relative to requested button*

```
//handle button tap for daily note - direct
$("#dayNote").on("tap", function(e) {
  e.preventDefault();
  console.log("request daily note...");
  var note = test.dailyNote();
  var noteOutput = "Today's fun note: "+note;
  console.log(noteOutput);
});
```

Image - Cordova Custom Plugin



```
request daily note...                                              plugin.js:15
Today's fun note: a daily travel note to inspire a holiday...      plugin.js:18
```

Cordova Custom Plugin - Direct Request

Cordova app - Plugins

Test plugin I - JS plugin - part 8

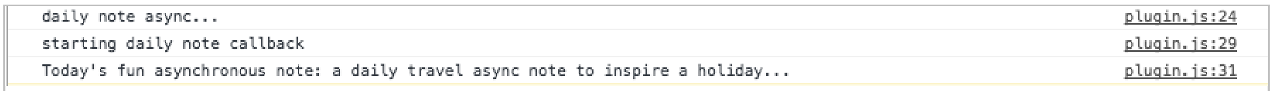
- request asynchronous version of daily note function from plugin's exposed API
- add an event handler to our `plugin.js` file
  - *responds to the request for this type of daily note*

```
//handle button press for daily note - async
$("#dayNoteSync").on("tap", function(e) {
  e.preventDefault();
  console.log("daily note async...");
  var noteSync = test.dailyNoteCall(noteCallback);
});
```

- then add the callback function

```
function noteCallback(res) {
  console.log("starting daily note callback");
  var noteOutput = "Today's fun asynchronous note: "+ res;
  console.log(noteOutput);
}
```

Image - Cordova Custom Plugin



```
daily note async...                                                          plugin.js:24
starting daily note callback                                                 plugin.js:29
Today's fun asynchronous note: a daily travel async note to inspire a holiday...   plugin.js:31
```

Cordova Custom Plugin - Async Request

Cordova app - Plugins

Test plugin 2 - Android plugin - part 1

- now setup and tested our initial JS only plugin application
- JS only can be a particularly useful way to develop a custom plugin
- often necessary to create one using the native SDK for a chosen platform
  - *eg: a custom Android plugin*
- now create a second test application
  - *then start building our test custom Android plugin*

```
cordova create customplugintest2 com.example.customplugintest2 customplugintest2
```

- add test template to application

Cordova app - Plugins

Test plugin 2 - Android plugin - part 2

- start to consider developing our custom Android plugin
- Android plugins are written in Java for the native SDK
- build a test plugin to help us understand process for working with native SDK
- test a few initial concepts for our plugin
  - *processing user input,*
  - *returning some output to the user*
  - *some initial error handling*

Cordova app - Plugins

Test plugin 2 - Android plugin - part 3

- now consider setup of our application to help us develop a native Android plugin
- three parts to a plugin that need concern us as developers

```
|- plugin.xml
|- src
   |- android
      |- Test2.java
|- www
   |- test2.js
```

- then add our required platforms for development

```
// add android
plugman platform add --platform_name android
```

- focus on the Android platform for the plugin

Cordova app - Plugins

Test plugin 2 - Android plugin - part 4

- start to build our native Android plugin
- begin by modifying the `Test2.java` file
- Cordova Android plugins require some default classes

```
import org.apache.cordova.CordovaPlugin;
import org.apache.cordova.CallbackContext;
```

- our Java code begins importing required classes for a standard plugin
- these include Cordova required classes
  - *required for general Android plugin development*

Cordova app - Plugins

Test plugin 2 - Android plugin - part 5

- now start to build our plugin's class
- start by creating our class, which will extend CordovaPlugin

```
public class Test2 extends CordovaPlugin {
  ...do something useful...
}
```

- then start to consider the internal logic for the plugin
- each Android based Cordova plugin requires an `execute()` method
- this method is run
  - *whenever our Cordova application requires interaction or communication with a plugin*
  - *this is where all of our logic will be run*

```
@Override
public boolean execute(String action, JSONArray args, CallbackContext callbackContext)
throws JSONException {
    if (action.equals("coolMethod")) {
        String message = args.getString(0);
        this.coolMethod(message, callbackContext);
        return true;
    }
    return false;
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 6

- for the execute method
  - *passing an action string*
  - *tells plugin what is being requested*
- plugin uses this requested action
  - *checks which action is being used at a given time*
  - *eg: plugins will often have many different features*
- code within `execute()` method needs to be able to check the required action
- now update our `execute()` method,

```java
@Override
public boolean execute(String action, JSONArray args, CallbackContext callbackContext)
throws JSONException {
    if (ACTION_GET_NOTE.equals(action)) {
        JSONObject arg_object = args.getJSONObject(0);
        String note = arg_object.getString("note");
    }
    String result = "Your daily note: "+note;
    callbackContext.success(result);
    return true;
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 7

- with our updated `execute()` method
  - *if the request action is `getNote`*
  - *our Java code grabs requested input from JSON data structure*
- current test plugin has a single input value
- if we started to build out the plugin
  - *eg: requiring additional inputs*
  - *we could grab them from the JSON as well*
- we've also added some basic error handling
- able to leverage the default `callbackContext` object
  - *provided by the standard Cordova plugin API*
- able to simply return an error to the caller
  - *if an invalid action is requested*
- one of the good things about developing an Android plugin for Cordova
  - *majority of plugins follow a similar pattern*
  - *main differences will be seen within the `execute()` method*

Cordova app - Plugins

Test plugin 2 - Android plugin - part 8

```
package org.csteach.plugin;
import org.apache.cordova.CallbackContext;
import org.apache.cordova.CordovaPlugin;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class Test2 extends CordovaPlugin {

  public static final String ACTION_GET_NOTE = "dailyNote";

    @Override
    public boolean execute(String action, JSONArray args, CallbackContext callbackContext)
    throws JSONException {
        if (ACTION_GET_NOTE.equals(action)) {
            JSONObject arg_object = args.getJSONObject(0);
            String note = arg_object.getString("note");
        String result = "Your daily note: "+note;
        callbackContext.success(result);
        return true;
    }
    callbackContext.error("Invalid action requested");
    return false;
    }
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 9

- need to update the JavaScript for our plugin
  - *helps us expose the API for the plugin itself*
- first thing we need to do is create a primary object for our plugin
- then use this to store the APIs needed to be able to request and use our plugin

```
var noteplugin = {
... do something useful...
}

module.exports = noteplugin;
```

- current API will support one action, our `getNote` action

```
getNote:function(note, successCallback, errorCallback) {
...again, do something useful...
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 10

- communication between JavaScript and the native code in the Android plugin
  - *performed using the `cordova.exec` method*
- method is not explicitly defined within our application or plugin
- when this code is run within the context of our Cordova application
  - *the `cordova` object and the required `exec()` method become available*
  - *they are part of the default structure of a Cordova application and plugin*
- now add our `cordova.exec()` method

```
cordova.exec(
...add something useful...
);
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part II

- now pass our `exec()` method two required argument
  - *represents necessary code for success and failure*
- basically telling Cordova how to react to a given user action
- then tell Cordova which plugin is required
  - *and associated action to pass to the plugin*
- also need to pass any input to the plugin
- updated `exec()` method is as follows

```
cordova.exec(
  successCallback,
  errorCallback,
  'Test2',
  'getNote',
  [{
  "note": note
  }]
);
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 12

- plugin's JavaScript code should now look as follows

```
var noteplugin = {

  getNote:function(note, successCallback, errorCallback) {

    cordova.exec(
      successCallback,
      errorCallback,
      'Test2',
      'getNote',
      [{
        "note": note
      }]
    );

  }
}

module.exports = noteplugin;
```

Cordova app - Plugins

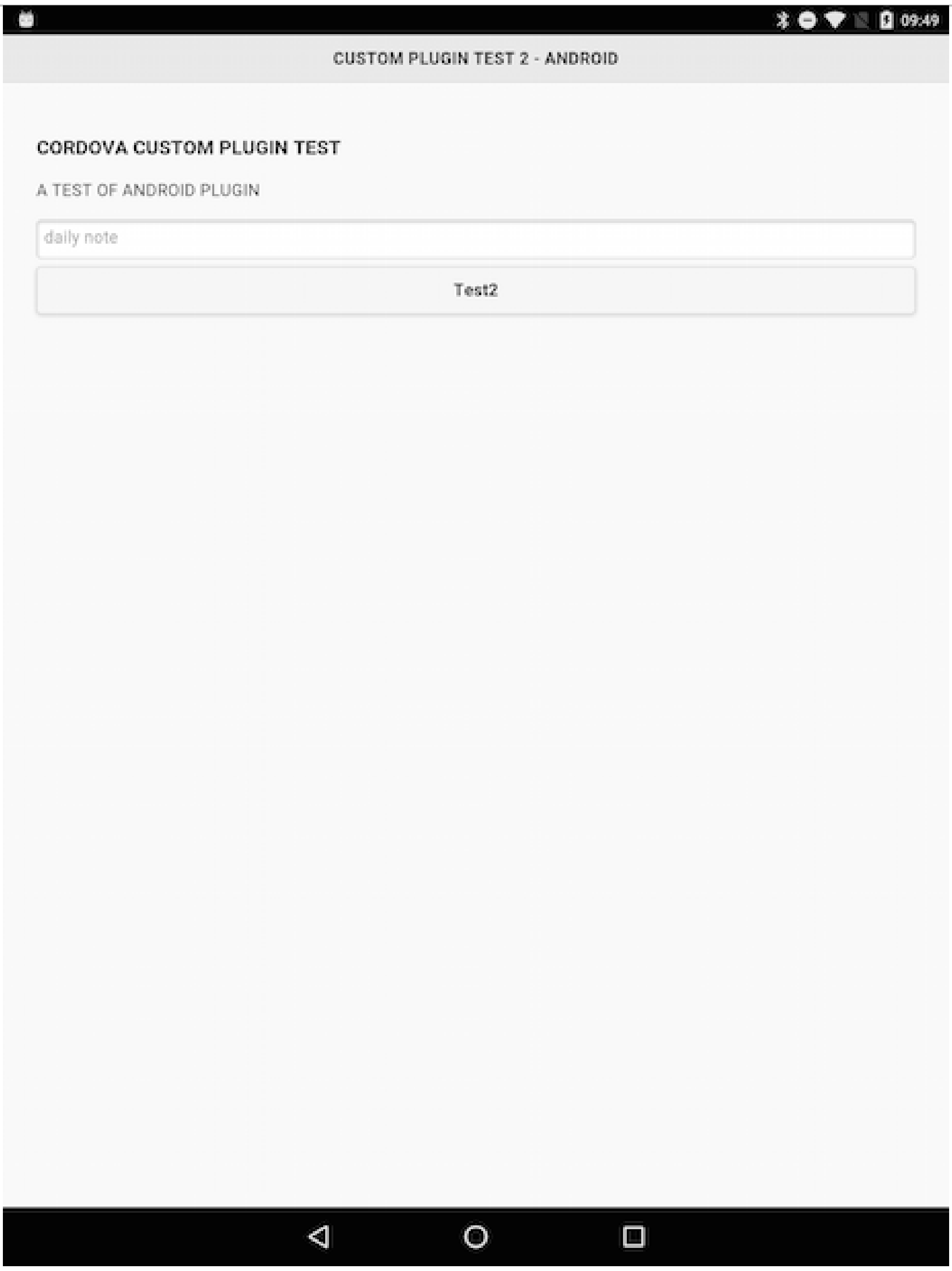
Test plugin 2 - Android plugin - part 13

- now need to test our plugin with our application
- update our home page to allow a user to interact with our new custom plugin
- add an input field for the user requested note
- add a button to submit the request itself

```
<input type="text" id="noteField" placeHolder="daily note">
<button id="testButton">Test2</button>
```

- exposed plugin API will be able to respond
  - *use the input data from the user*
  - *then pass to the native Android plugin*

Image - Cordova Custom Plugin 2



Cordova Custom Plugin 2 - HTML Update

Cordova app - Plugins

Test plugin 2 - Android plugin - part 14

- update app's `plugin.js` to handle user input
  - *then process for use with our custom plugin*
- still need to wait for the `deviceready` event to return successfully
- then we can start to work with our user input and custom plugin
- our native Android plugin's API is similarly exposed using the window object

```
window.test2
```

- we can then execute it from our application's JS

```
windows.test2.getNote
```

- then pass the requested note data to the API
- define how we're going to work with success and error handlers
  - *render the returned value to the application's home page*

```
window.test2.getNote(note,
  function(result) {
    console.log("result = "+result);
    $("#note-output").html(result);
  },
  function(error) {
    console.log("error = "+error);
    $("#note-output").html("Note error: "+error);
  }
);
```

Cordova app - Plugins

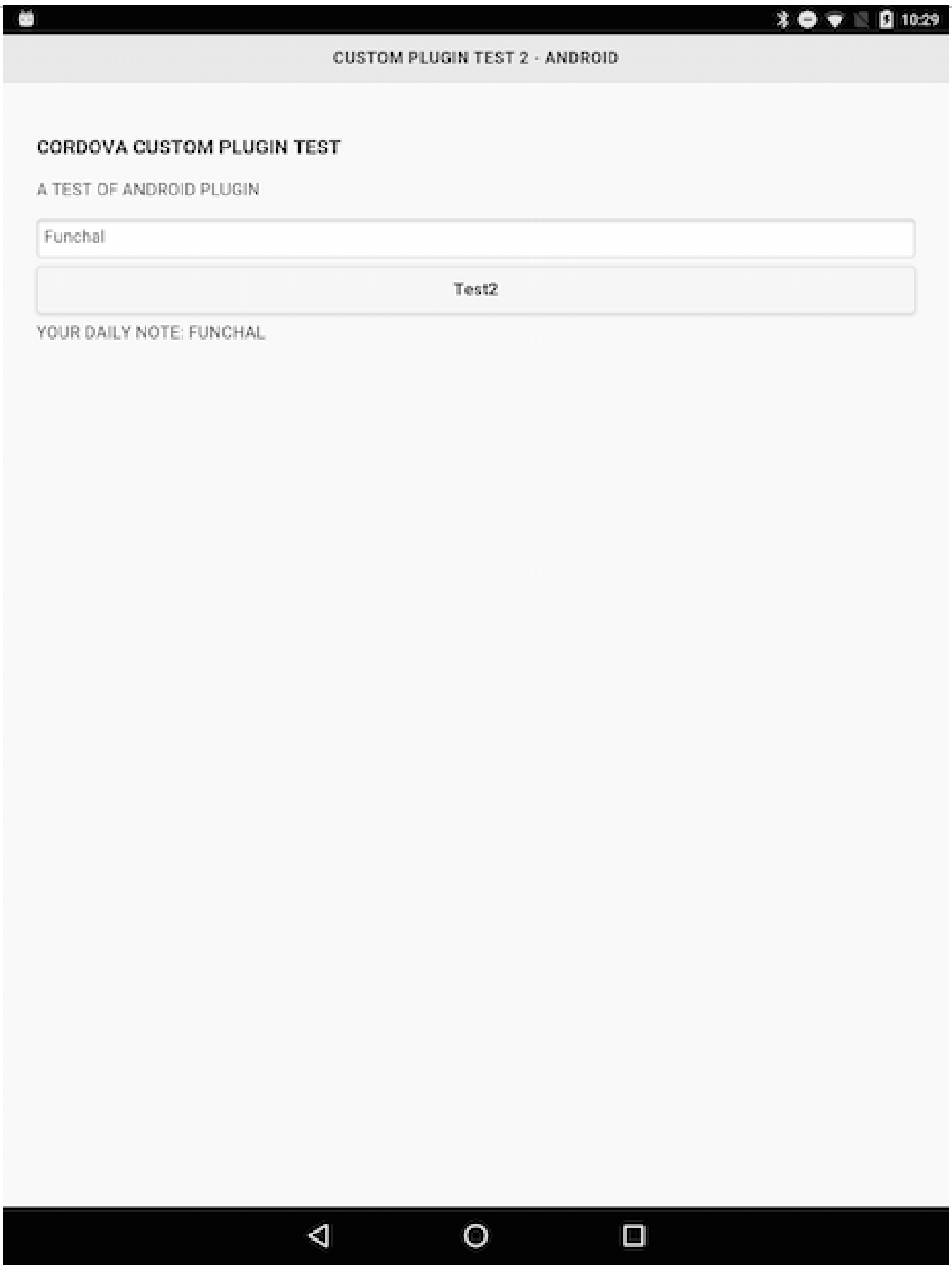
Test plugin 2 - Android plugin - part 15

```
function onDeviceReady() {

//handle button press for daily note - direct
$("#testButton").on("tap", function(e) {
  e.preventDefault();
  console.log("request daily note...");
  var note = $("#noteField").val();
  console.log("requested note = "+note);
  if (note === "") {
    return;
  }
  window.test2.getNote(note,
    function(result) {
      console.log("result = "+result);
      $("#note-output").html(result);
    },
    function(error) {
      console.log("error = "+error);
      $("#note-output").html("Note error: "+error);
    }
  );
});

}
```

Image - Cordova Custom Plugin 2



CORDOVA CUSTOM PLUGIN TEST

A TEST OF ANDROID PLUGIN

Funchal

Test2

YOUR DAILY NOTE: FUNCHAL

Cordova Custom Plugin 2 - Android plugin output

Cordova app - Plugins

Summary of custom plugin development

- an initial template for a custom plugin can be created using the *Plugman* tool
- create JS only custom plugins
- create native SDK plugins
  - *eg: Android, iOS, Windows Phone...*
- custom plugin consists of
  - *plugin.xml*
  - *JavaSript API*
  - *native code*
- create the plugin separate from the application
  - *then add to an application for testing*
  - *remove to make changes, then add again...*

Cordova app - Extras

Other UI Options - Ionic - part 1

- briefly consider option of using Ionic's framework
  - *for developing your UI for Cordova applications*
- Ionic is a HTML framework
  - *designed specifically for development of hybrid applications*
  - *including Cordova mobile applications*
- originally created by a group of developers called **Drifty**
- known to be simple to use and very fast
- Ionic provides
  - *overall UI framework*
  - *accompanying CLI*
- CLI is wrapper for Cordova CLI
- install Ionic using NPM

```
sudo npm install -g ionic
```

- start using Ionic at CLI with `ionic` command

Cordova app - Extras

Other UI Options - Ionic - part 2

- Ionic provides a number of useful starter templates
  - *use and modify for the development of our Cordova applications*
- create a new Ionic project
  - *use the following command at the CLI,*

```
ionic start csteach422 blank
```

- specify the project name
  - *in this example* `csteach422`
- required template for this project
- in this example `blank`
- templates include
  - *Tabs (default) - Demo*
  - *Sidemenu - Demo*
  - *Blank - Demo*
- Ionic CSS Styles - Demo
- Ionic creates a Cordova application
  - *with addition of support and styling for Ionic based UI*

Cordova app - Extras

Other UI Options - Ionic - part 3

- Ionic framework has now used Cordova to build the new project
- also added some Ionic specific components
  - *custom components to help with builds, UI framework updates...*
- Ionic adds platform support for iOS by default
  - *then we can the standard Android support*

```
ionic platform add android
```

- Ionic CLI commands closely match familiar Cordova commands
- a useful command

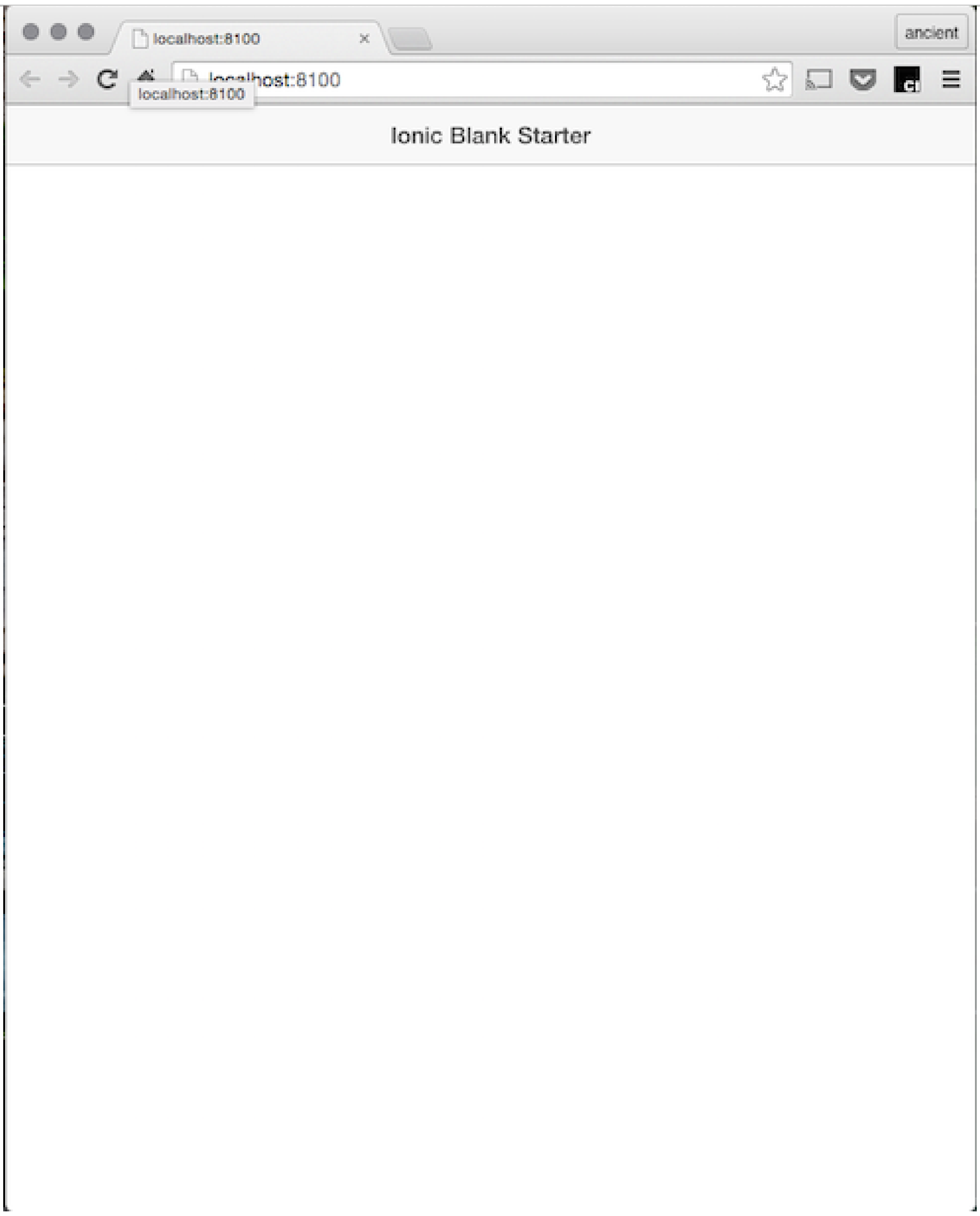
```
ionic serve
```

- start a local web server and test our project from the working directory
  - *CLI checks preferred server address, eg:* `localhost`
  - *loads project in default browser*

Image - Ionic Starter

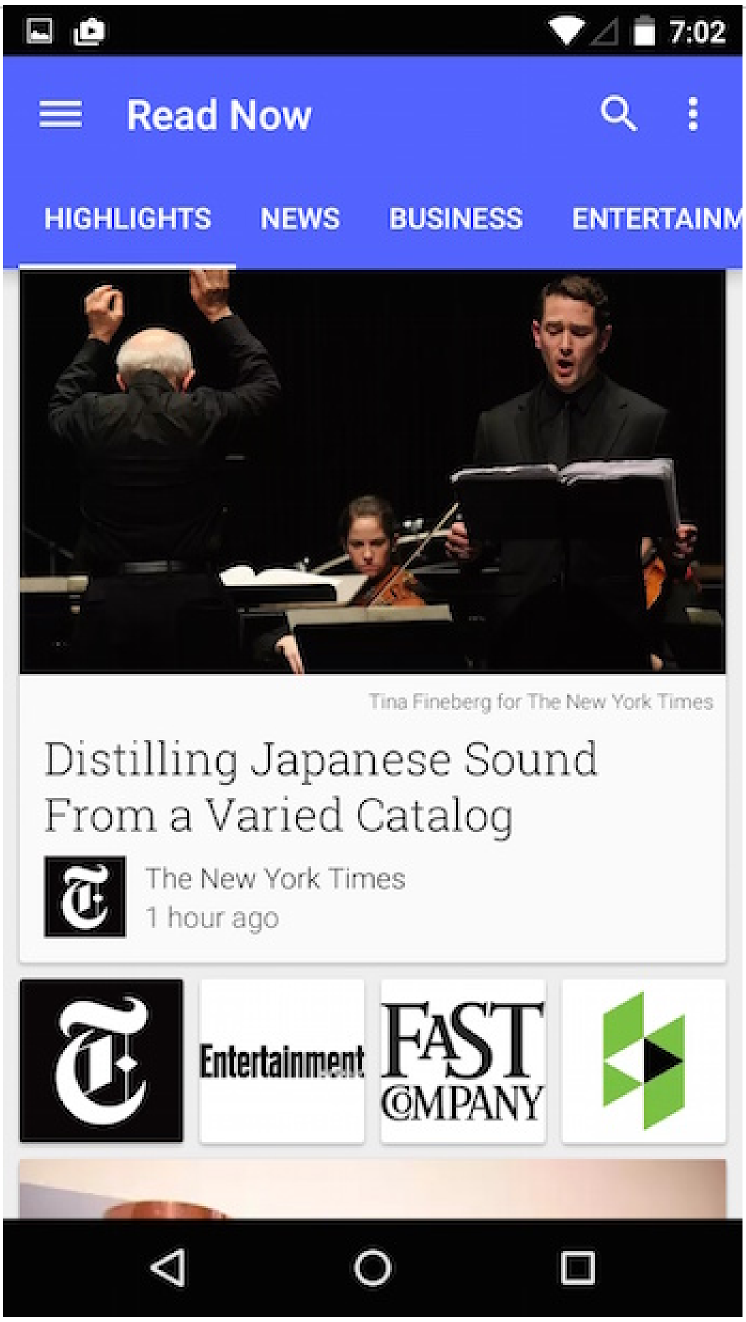
Ionic Starter on Android

Image - Ionic Starter



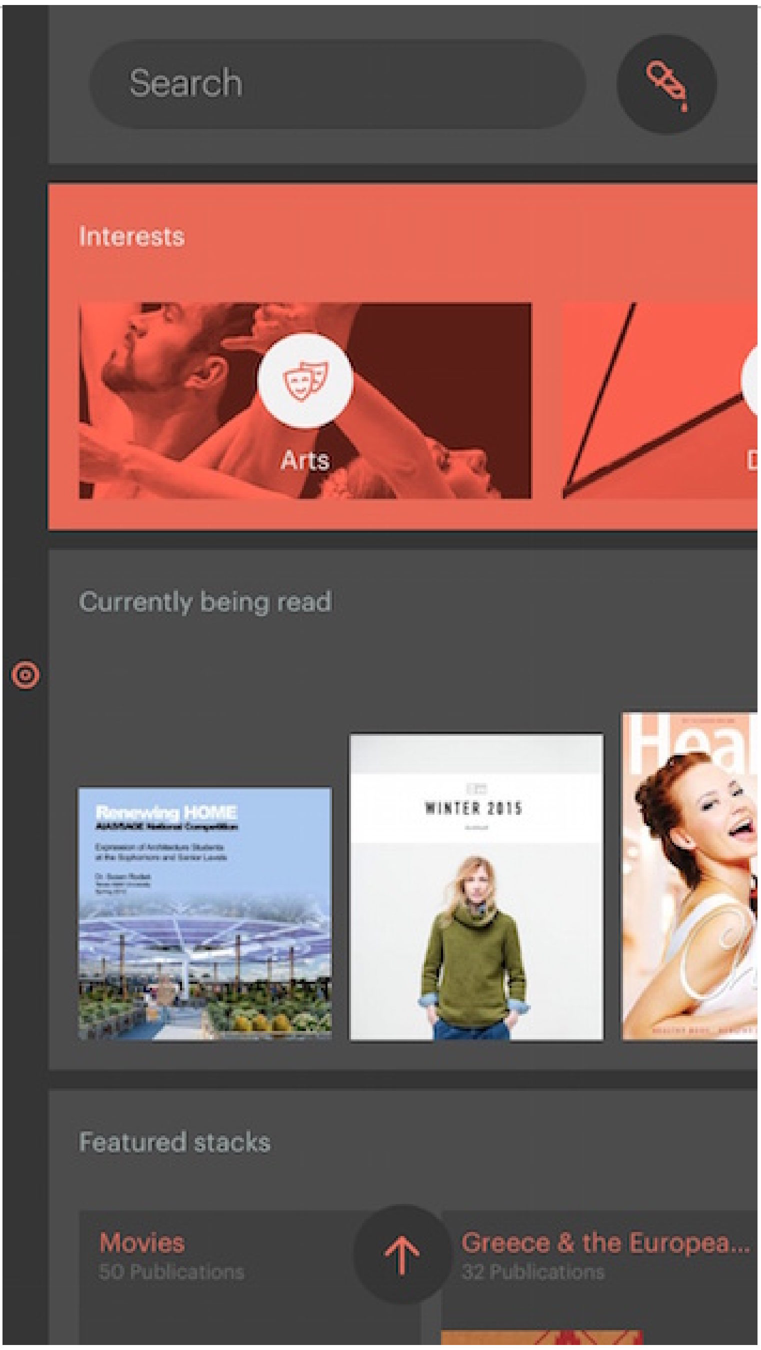
Ionic Starter in the browser

Considering mobile design patterns - screen 1



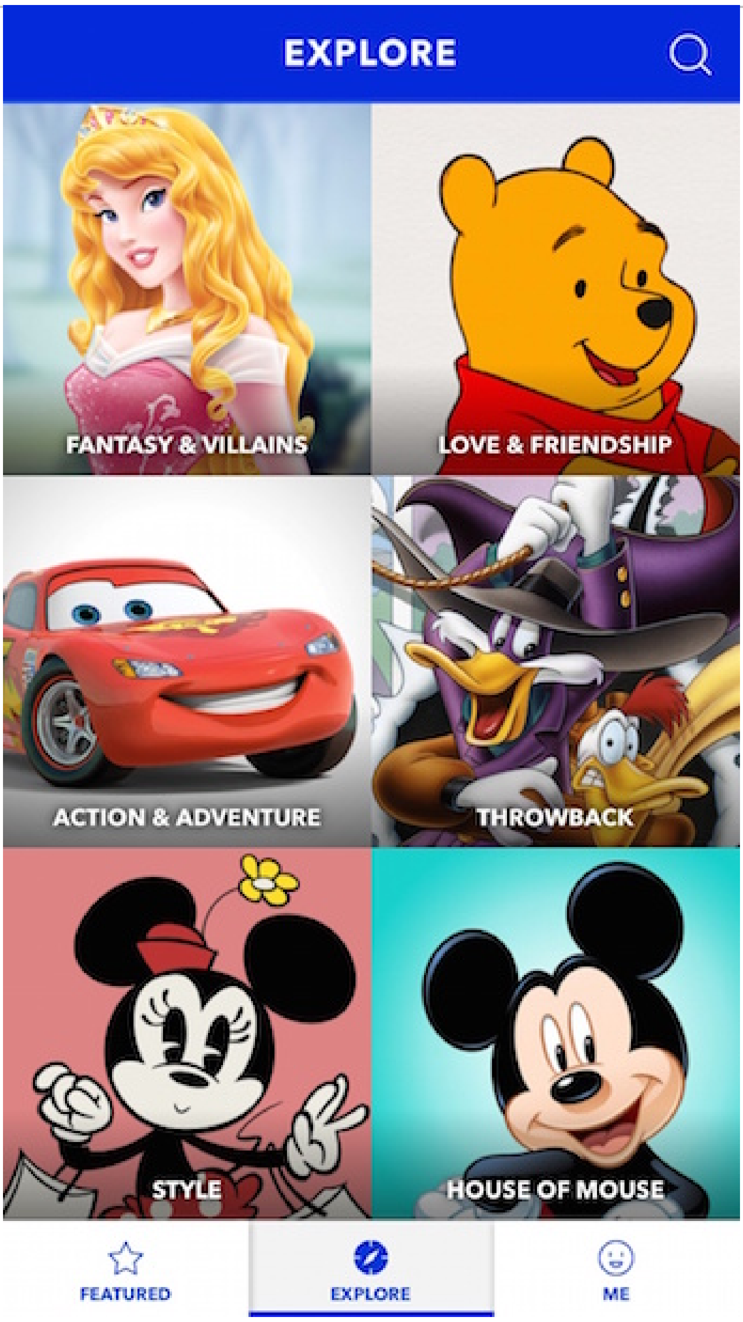
Google Play Newstand on Android

Considering mobile design patterns - screen 2



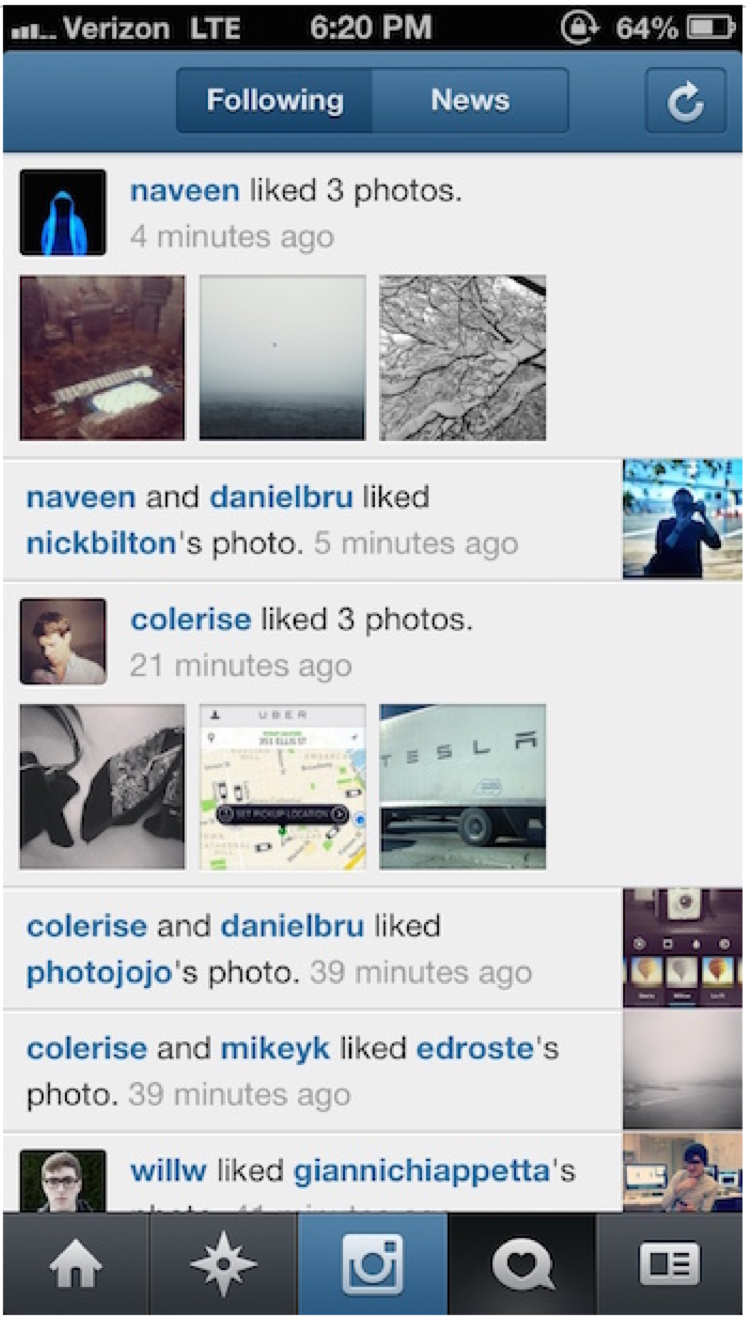
Issuu magazines on iOS

Considering mobile design patterns - screen 3



Disney Inquizitive

Considering mobile design patterns - screen 4



naveen liked 3 photos. 4 minutes ago

naveen and danielbru liked nickbilton's photo. 5 minutes ago

colerise liked 3 photos. 21 minutes ago

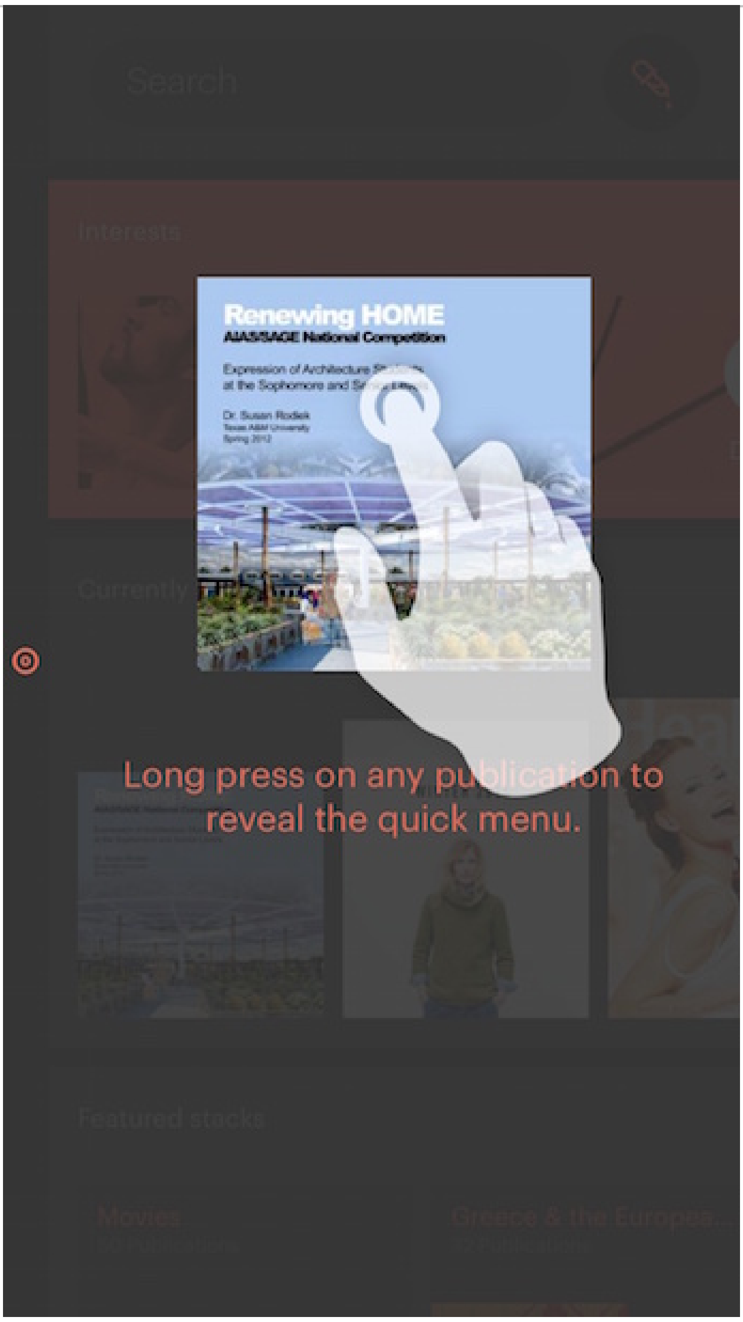
colerise and danielbru liked photojojo's photo. 39 minutes ago

colerise and mikeyk liked edroste's photo. 39 minutes ago

willw liked giannichiappetta's

Instagram on iOS

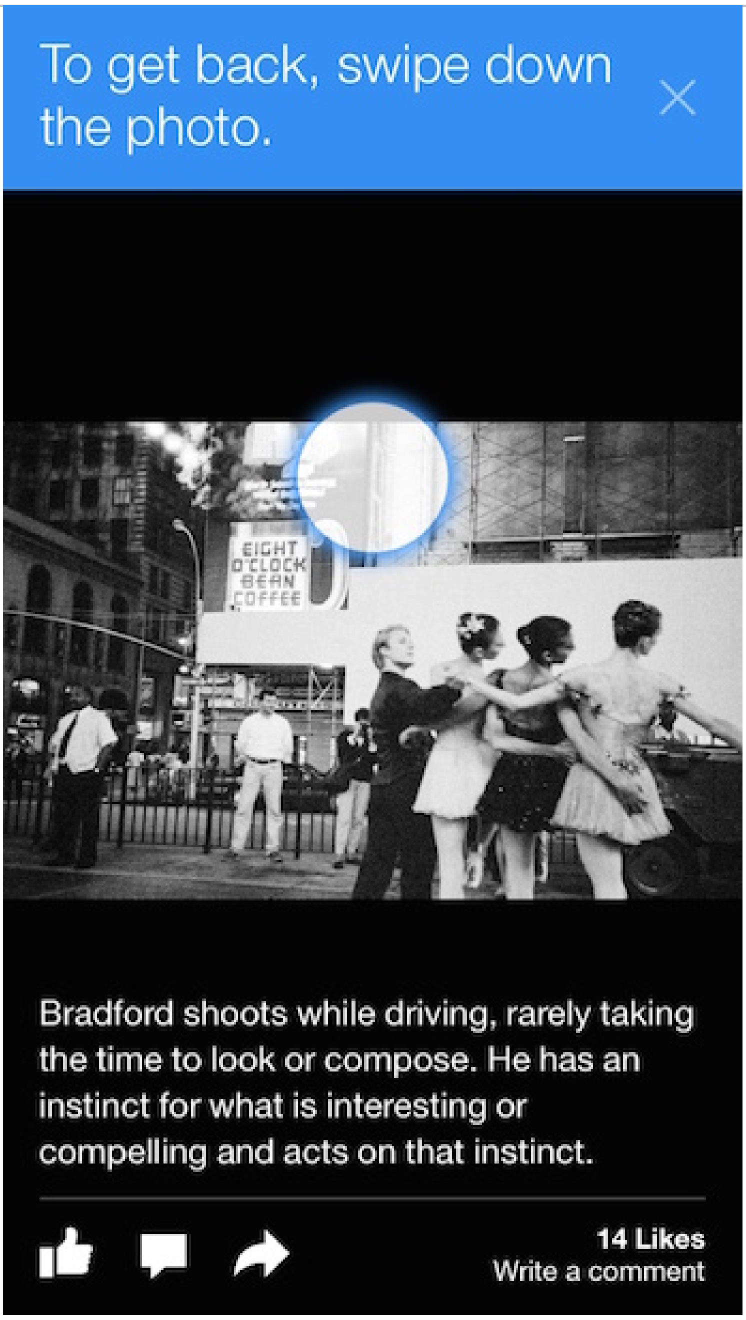
Considering mobile design patterns - screen 5



Long press on any publication to reveal the quick menu.

Issuu on iOS

Considering mobile design patterns - screen 6

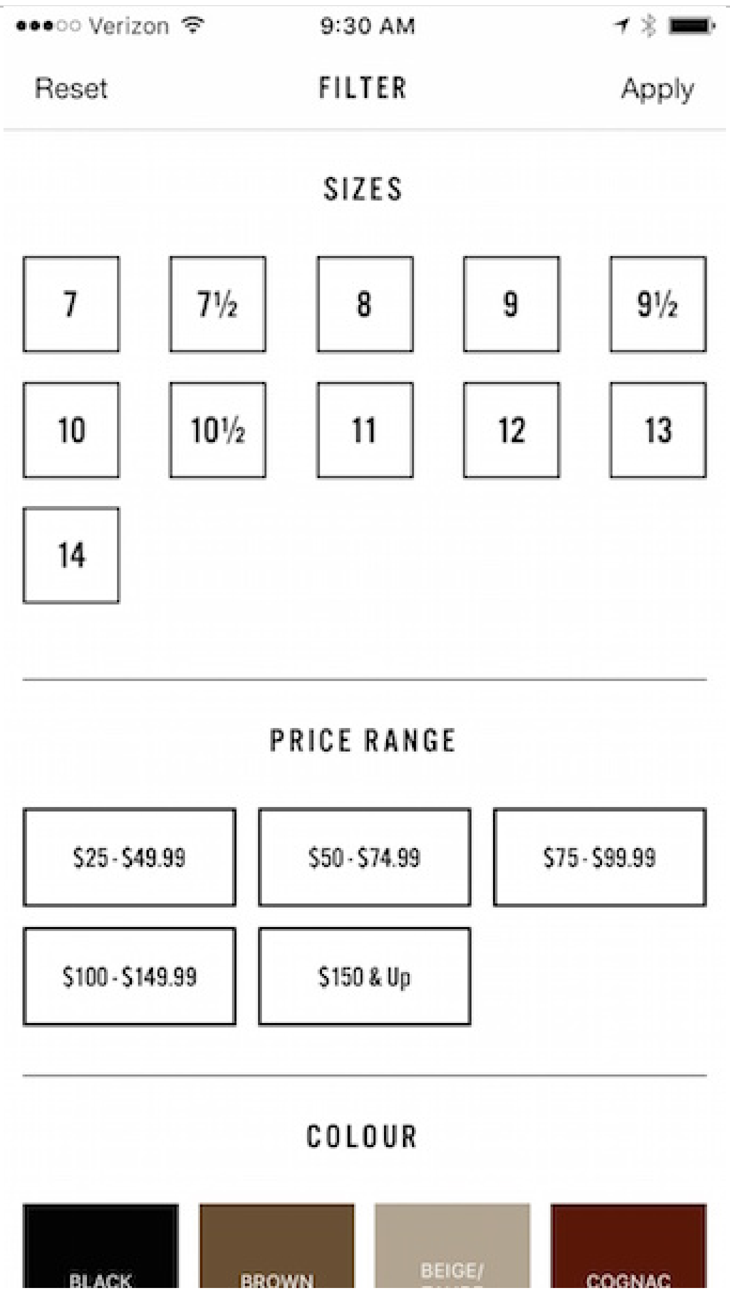


To get back, swipe down the photo.

Bradford shoots while driving, rarely taking the time to look or compose. He has an instinct for what is interesting or compelling and acts on that instinct.

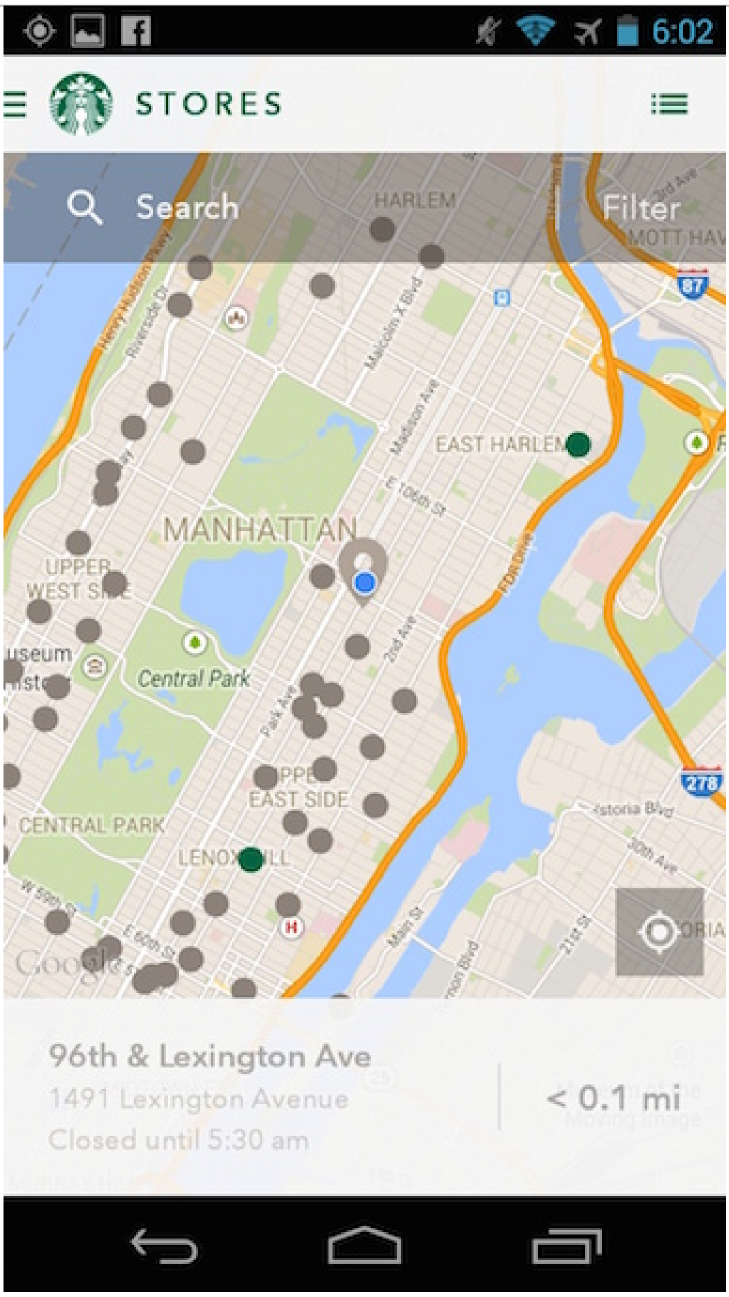
Paper- stories from Facebook

Considering mobile design patterns - screen 7



Aldo on iOS

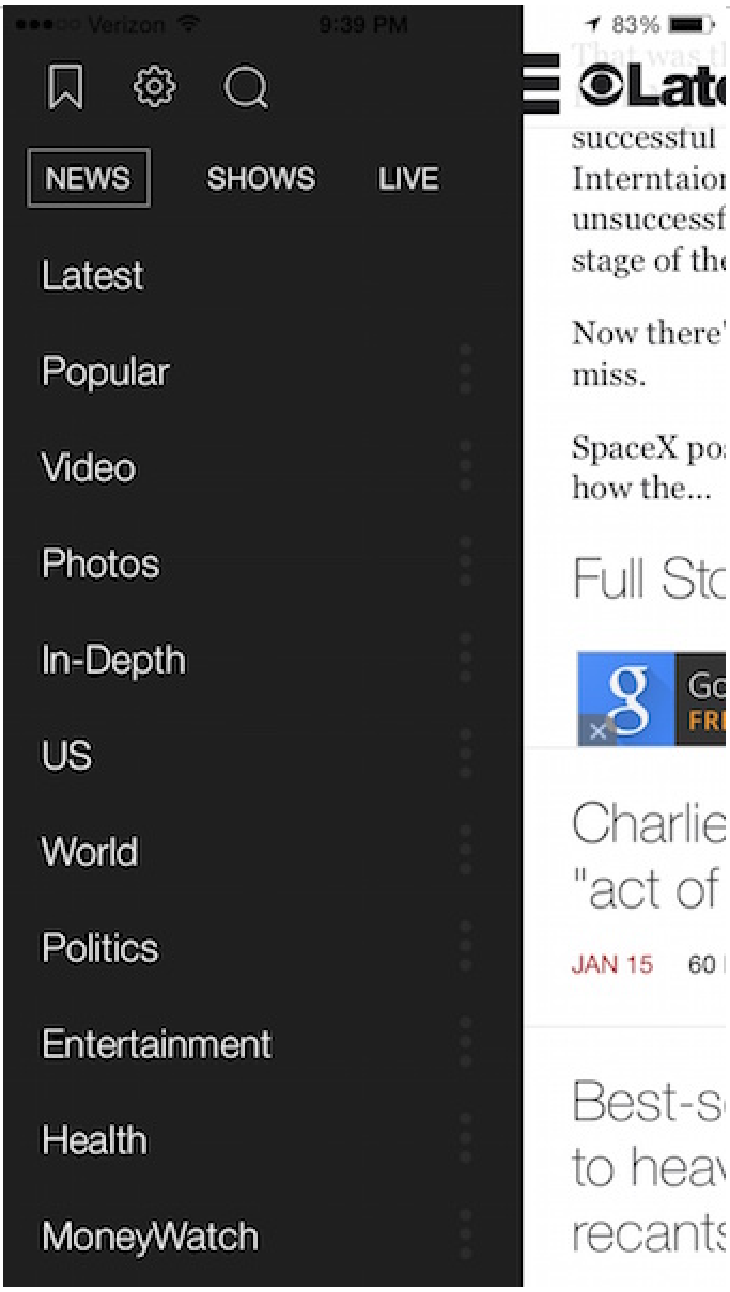
Considering mobile design patterns - screen 8



96th & Lexington Ave
1491 Lexington Avenue
Closed until 5:30 am
< 0.1 mi

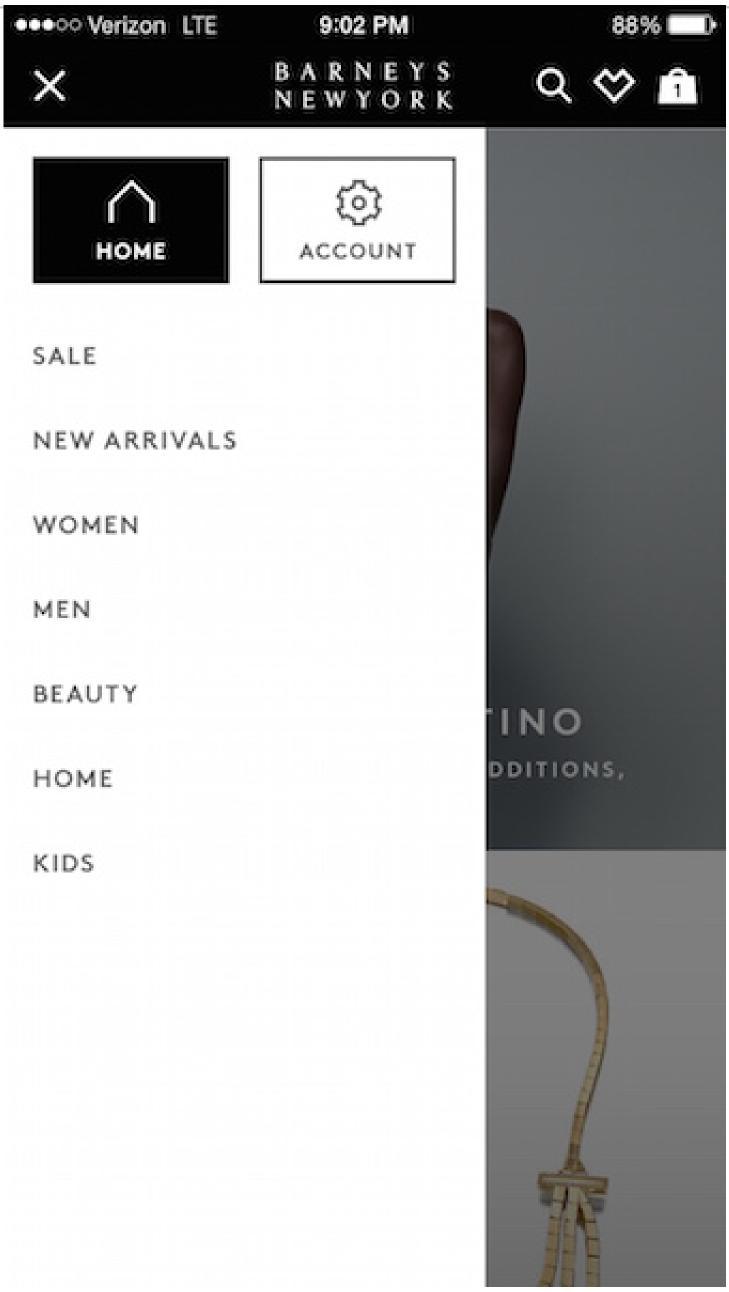
Starbucks on Android

Considering mobile design patterns - screen 9



CBS News on iOS

Considering mobile design patterns - screen 10



Barneys New York on iOS

Designing our app - I

Positive User Experience

- we need to be able to identify traits of a positive user experience
  - *conversely, understanding a negative experience is also helpful*
- application allows a user to feel they are in control
- helps develop a sense of confidence and competence with the application
- helps encourage high productivity and efficiency
  - *enables and encourages our user to develop a sense of* ***flow***
- allows simple, routine tasks to be completed as quickly and easily as possible
- produces valid, useful output for the user
- user feels confident with the validity of produced results, calculations...
- considered aesthetically pleasing
- exhibits acceptable, sufficient performance to avoid unnecessary delays and waiting
- stable and reliable for the user...no *blue screen of death*
- makes it easy for a user to correct or modify any errors, mistakes...
- inspires trust and confidence in the user with logical, well-ordered design, navigation...

Designing our app - 2

Negative User Experience

- application leaves a user with a sense of feeling a lack of control
- overwhelming the user, creating a sense of incompetence and inadequate ability
- hinders the user from improving productivity and general efficiency
    - *prevents a sense of **flow***
- simple tasks and routine patterns prove overly complicated for the user
- output from the application is flawed, incorrect, poorly formatted...
- the app may produce unreliable results and calculations
- the UI design is aesthetically dis-organised, cluttered, unappealing...
- slow in performing tasks, and exhibits unnecessary delays and lags in performance
- unstable, buggy, and prone to crashing...
    - *user loses data due to poor performance*
- **excessive complexity** and difficulty in general functionality
- **too much work** involved to use the application in general
- design that conflicts with a user's perception of previous applications, iterations of a design, and competing products

Designing our app - 3

Violating design principles

- issues that arise in usability
  - *consequence of poor interpretation, implementation, or misunderstanding general design principles*
- reconsider Norman's design principles
  - ***lack of consistency***
  - ***poor visibility***
  - ***poor affordance***
  - ***poor mapping***
  - ***insufficient feedback***
  - ***lack of constraints***

Designing our app - 4

interaction concept - part 1

- app's **interaction concept**
  - *basic summary of our base, fundamental idea of how the user interface will actually work*
  - *describes presentation of the UI to the user*
  - *general interaction concepts that allow a user to complete tasks*
- inherent benefit is that it will often highlight initial usability issues
  - *including navigation, workflow, and other carefully considered and planned interactions*
- every aspect cannot be defined and outlined at the initial design stage
- follow a more agile approach instead of formal specification documents
- prototyping a particularly effective method for
  - *testing different design ideas*
  - *receiving feedback through peer reviews and associated usability testing*
  - *representing and communicating intended design to a client etc*
- lightweight written records as supplemental and supporting material

Designing our app - 5

interaction concept - part 2

Analysis of interaction concepts

- interaction styles
- information architecture basics, which often include the following
  - *a data model*
  - *a naming scheme, or defined glossary of preferred names and labels*
  - *a navigation scheme*
  - *a search and indexing scheme*
- an outline of a framework for interactions and workflow
- an outlined concept for transactions and any necessary persistency
- AND, a framework for the general visual design of the application

Designing our app - 6

interaction style

- app's **interaction style**
  - *fundamental way it presents itself to a user to allow interaction with available functionality*
  - *many different concepts for interaction styles and overlap*
  - *many will employ a variety or combination of these interaction styles*
- an application might present the following styles to its users
  - ***menu driven options*** *- user is able to select options from menus, sub-menus*
  - ***forms*** *- user able to enter data, respond to queries by completing forms*
  - ***control panel options*** *- may show data visualisations, summaries, quick access options*
  - ***command line*** *- allows expert, power users to control the app using commands and queries*
  - ***conversational input*** *- user may interact in a back-and-forth dialogue or conversational style*
    - a sense of question asked and reply returned
  - ***direct manipulation*** *- direct user manipulation of objects within the app on the screen*
  - ***consumption of content*** *- app is simply a way to consume content*
    - eg: e-Book readers, music and video players...
- an app will normally use a combination of the above interaction styles

Image - Interaction Style

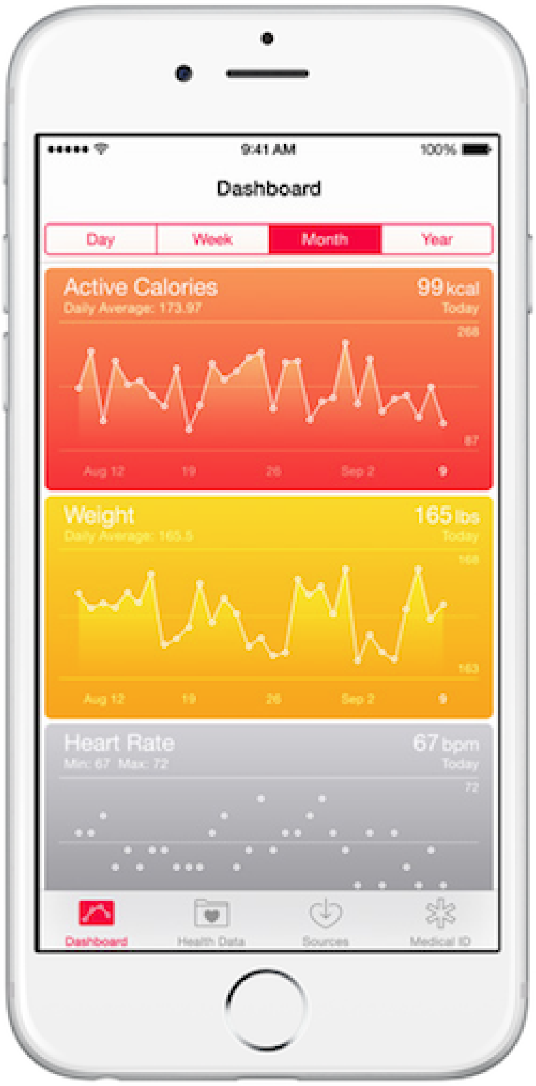
Interaction Style - Mobile Consideration

Designing our app - 7

information architecture - part 1

- concerned with the organisation of information into a perceived coherent structure
- structure is considered comprehensive, navigable, and in many situations searchable
  - *eg: concepts, entitites, relationships, functionality, events, content...*
- designing such information architecture requires the following considerations and implementation
  - *data model*
  - *naming scheme or glossary*
  - *names and titles for identification of places*
  - *navigation and location awareness*
  - *navigation map and associated mechanisms*
  - *breadcrumbs and navigation notifications*
  - *presentation of such places*
  - *searching*

Image - Information Architecture - Visualisations



Information Architecture - Apple Health

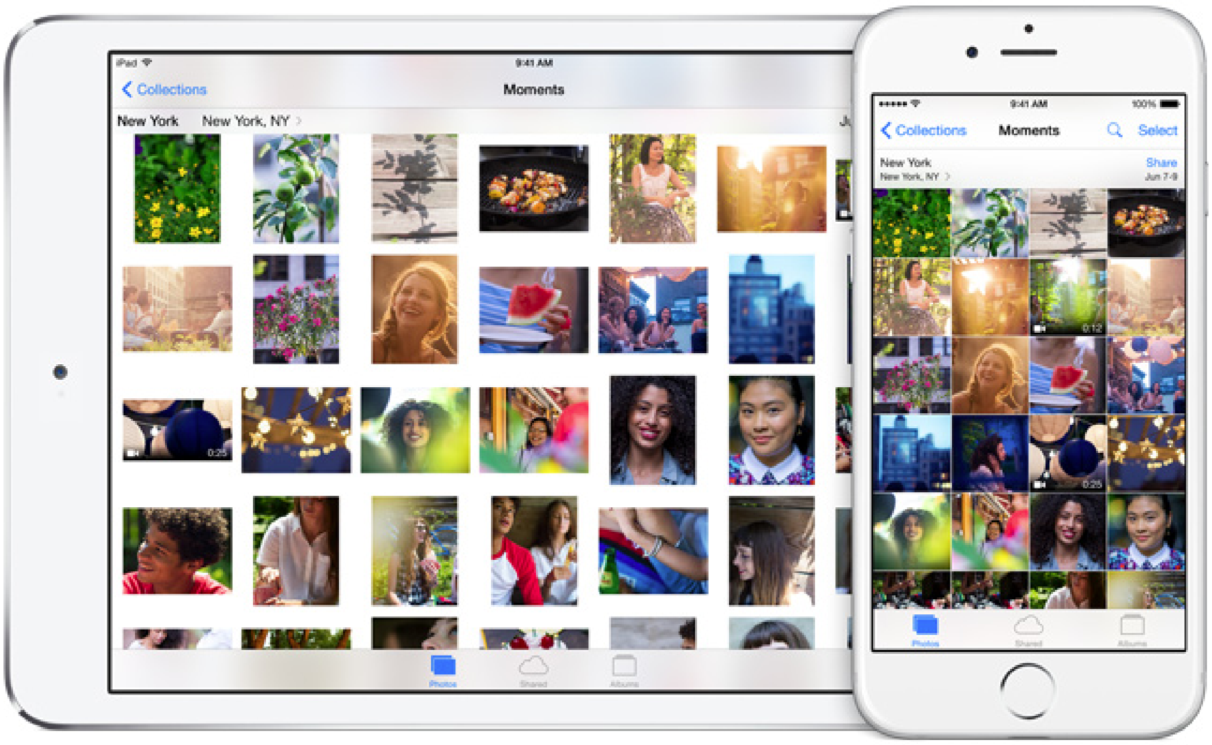
Designing our app - 8

information architecture - part 2

data model, naming scheme, naming places...

- identification and recording of the entities, attributes, and operations for each entity
- also includes identification of the relationships between the entities
- often argued that the data model is, in fact, part of the app's interaction concept
  - *perceived to help define the nature of the product*
- coherent and consistent naming scheme is important to aid user's mental model
- definition of official names for an app's key elements and processes
  - *can be formalised and recorded in the defined interaction concept*
- apps with specialised domains may require a glossary of names and labels
  - *helps define the official, preferred terminology*
  - *interaction concept may then link or reference this glossary*
- places within an app should be clearly named and labelled
  - *helps users determine what they are viewing and where in the app*
  - *helps users differentiate places and concepts within an app*
  - *clear naming of places helps define them in menus, instructions, help text...*
- user-defined place names are OK as well
  - *eg: a title of a document in an editing app*

Image - Information Architecture - Personal Naming Schemes



Information Architecture - Apple Photos

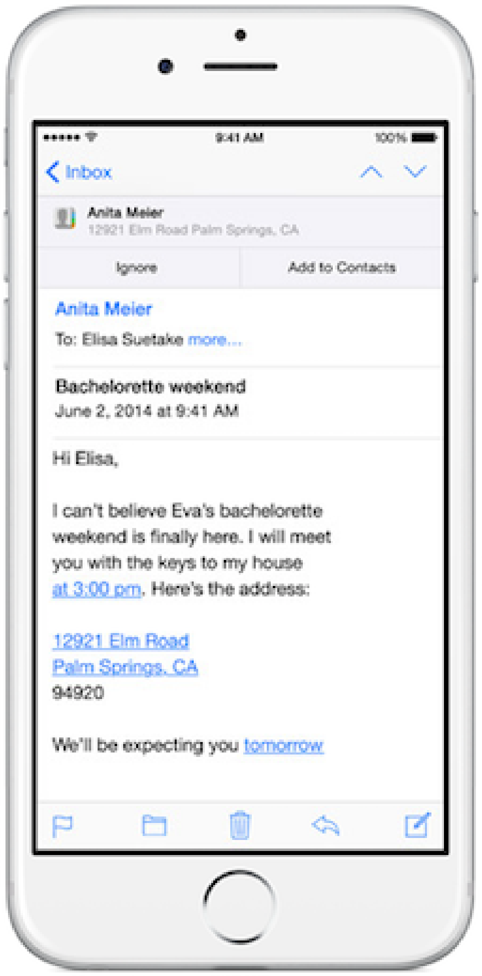
Designing our app - 9

information architecture - part 3

navigation and places

- app design often references navigation relative to defined places
  - *eg: in a web app places may be defined as pages or screens*
- not all places need to be user accessible
- places may also refer to sub-divisions such as panels, tabs, sub-sections...
  - *sub-sections may also include dialogs, image presentations etc*
- for apps with many places, a design should help users determine and differentiate
  - *where they are currently located within the app*
  - *where they can go next*
  - *how to easily get where they want to go*
- in addition to naming places, we need to consider their actual presentation as well
  - *how do we present different places to our users*
  - *view multiple places at once, or page/navigate through single places*
  - *can these places be resized, moved and rearranged, opened, closed, hidden, removed entirely...*
  - *can we relate content from one place to another*

Image - Information Architecture - Determining Places



Anita Meier
12921 Elm Road Palm Springs, CA

Anita Meier
To: Elisa Suetake more...

Bachelorette weekend
June 2, 2014 at 9:41 AM

Hi Elisa,

I can't believe Eva's bachelorette weekend is finally here. I will meet you with the keys to my house at 3:00 pm. Here's the address:

12921 Elm Road
Palm Springs, CA
94920

We'll be expecting you tomorrow

Information Architecture - Apple Mail

Designing our app - 10

information architecture - part 4

navigation map

- allow us to consider and define the places that may exist within our application
  - *the movements allowed from one to the other*
- beneficial if represented in a graphical manner within quick reference diagrams
- designing a complete navigation map at the design stage may be impractical and counter-productive
  - *initial map can always be expanded and modified as we develop the application.*
- some instances where a navigation map is simply impractical
  - *eg: dynamic applications, such as catalogues, wikis, some games...*
  - *many different links, pathways, and related material a user may generate*

Designing our app - 11

information architecture - part 5

navigation mechanisms

- many different ways for a user to switch places and content. A few defined examples include
  - ***bookmarks***
  - ***buttons***
  - ***events*** - *triggered by a user action or application process can show a notification or message window*
  - ***flow diagrams*** - *visualise steps and outcomes relative to the current complex process or workflow*
  - ***hierarchical structures*** - *eg: trees used to display hierarchical depth of data...*
  - ***history***
  - ***links***
  - ***maps*** - *data points represented geographically, or conceptual map of data, app domain...*
  - ***menus***
  - ***searching*** - *simple act of searching by keyword, selecting from a faceted list of terms...*
  - ***switching*** - *move between multiple places currently available within the UI*

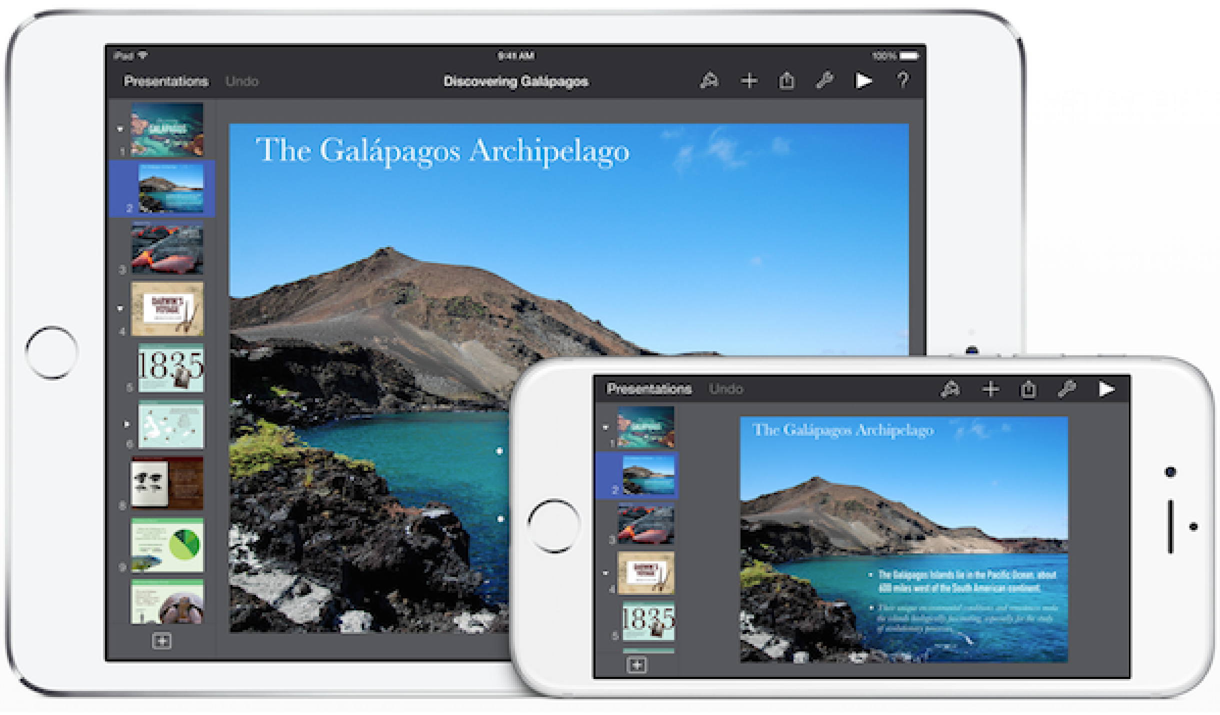
Designing our app - 12

information architecture - part 6

user location

- clearly identify a user's current location
- acts as a quick reminder to the user
  - *also creates a familiar contextual placeholder within the app*
- indicate the user's current location in a number of different ways
  - *clearly display the title or name of the current place with any associated contextual name*
  - *highlight the current place name or title on a visual map or flow diagram*
  - *include a representation of location on a visual flow diagram for a process of series of tasks*
  - *locate a current place within a defined hierarchical structure*
    - such as a tree representation of the current document or data...
- breadcrumb trail useful for hierarchical data representations
  - *benefit of acting as both location indicator and simple form of navigation*

Image - Information Architecture - User Location



Presentations
Undo
Discovering Galápagos
The Galápagos Archipelago
1835
Presentations
Undo
The Galápagos Archipelago
The Galápagos Islands lie in the Pacific Ocean, about 600 miles west of the South American continent.

Information Architecture - Apple Keynote

Designing our app - 13

interaction framework - part 1

considerations

- identify core sets of features, tasks, actions, operations, and processes
- consider series of use cases that follow and share similar patterns of interaction
  - *editing application may allow user interaction with many disparate tools and actions*
    - common menu structure, tools...variance is the selected tool itself
    - interaction will be able to follow a similar pattern
  - *we can also see this type of example with games*
    - many different levels, challenges, opponents
    - similar interaction concepts from level to level
- create an initial list or breakdown of these similar tasks or features
  - *then start to design an interaction framework to describe perceived commonalities*
    - such as the presentation and behaviour of the user interface
  - *this list allows us to*
    - understand how the application will fundamentally behave
    - ensure consistency across such similar tasks
  - *by simply documenting the commonalities between such tasks*
    - saves us from re-documenting the same aspects for individual tasks for our overall specs
- framework also useful for the development of the overall design and its technical underpinnings

Designing our app - 14

interaction framework - part 2

issues

- how tasks are started or triggered
  - *eg: user selecting an item on a menu...*
- required authorisations
- when and how tasks can be activated and any given cases where tasks may be disabled
- how and when the task is considered complete
- does the start or end of a task signal a change in any status, mode etc...
- what are the effects of the task on the system's data
  - *eg: is data saved automatically, does it persist or is it temporary*
  - *what happens if the task is abandoned*
  - *what happens if an error breaks the task...*

Designing our app - 15

interaction framework - part 3

data and persistency

- need to consider data transactions and persistency in an application
    - *eg: what, if any, of the application's data needs to saved or stored...*
- for the interface and interaction concepts
    - *consider how the actual saving of data works in the application*
    - *is the data generated by user interactions saved in a persistent store?*
    - *is the data saved in a temporary memory cache?*
    - *consider how such data saving and persistency is relayed to the user*
    - *are they aware that the data is being saved?*
    - *is it an explicit act in the interface design?*
    - *is it part of an auto-save option running as a background process?*

Designing our app - 16

interaction framework - part 4

data and persistency

- consider standard data design patterns that include validations of the data
  - *also consider accompanying error and notification messages*
- for the interface and interaction designs
  - *carefully plan how error messages are presented*
  - *whether the validation occurs on the client or server side*
- consider whether partial data for incomplete tasks is saved
- in the interface design, clearly identify potential *save points*
  - *helps correct notification to the user*
  - *we can also offer suggestions, reminders, completion estimates...*
  - *save points allow us to track current data*
  - *has it been saved recently?*
  - *is it a version or a re-write of saved data...*
  - *is it a persistent save or cached?*

References

- Cordova
  - *Plugin Development Guide*
  - *Plugman*
- Ionic
  - *Home*
  - *Docs*