Comp 322/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2018 - Week 14

Dr Nick Hayward

Final Demo and Presentation

- presentation and demo - live working app...
  - *due on Tuesday 4th or Thursday 6th December 2018 @ 2.30pm*
- continue to develop your app concept and prototypes
  - *develop application using any of the technologies taught during the course*
  - *again, combine technologies to best fit your mobile app*
- produce a working app
  - *as far as possible try to create a fully working app*
  - *explain any parts of the app not working...*
- explain choice of technologies for mobile app development
  - *e.g. data stores, APIs, modules, &c.*
- explain design decisions
  - *outline what you chose and why?*
  - *what else did you consider, and then omit? (again, why?)*
- which concepts could you abstract for easy porting to other platform/OS?
- describe patterns used in design of UI and interaction
- consider outline of content from final report outline
  - ...

All project code must be pushed to a repository on GitHub.

Final Report

Report due on Saturday 15th December 2018 by 6.15pm

- final report outline - coursework section of website
  - *PDF*
  - *group report*
  - ***extra individual report*** *- optional*
- include repository details for project code on GitHub

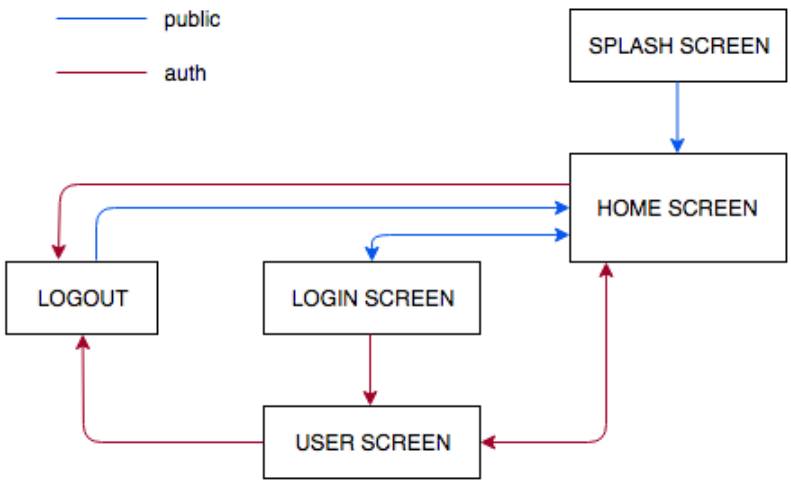
Cross-platform - navigation & data

app structure - public and auth routes

- a more detailed example might include multiple navigation paths
  - *paths relative to user authentication, data, options...*
  - *e.g. app loads with Splashscreen, then redirects to Home Screen.*
- from the *Home Screen*
  - *a user has option to follow public or authenticated routes*
  - *each route will require navigation support*
- authenticated route may contain a minimum set of screens, e.g.
  - *logout*
  - *user*
- public route will often comprise bulk of app's screens, e.g.
  - *login*
  - *data such as a rendering of data store records &c.*
  - *search*
  - *timeline*
  - *maps*
  - *...*
- some crossover between public and authenticated routes
- authenticated user may gain extra features, e.g.
  - *access to specific data for their personal account*
  - *options such as messaging and customisation.*

Image - Navigation

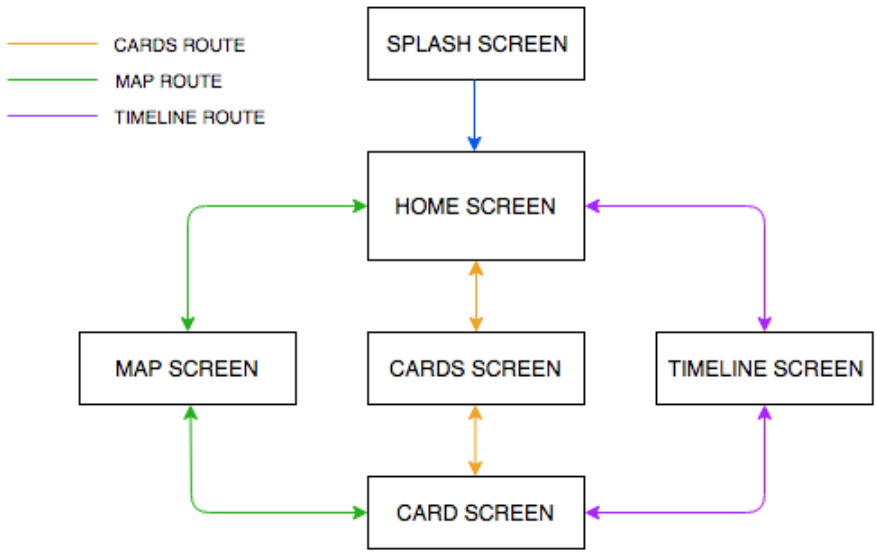
user auth



Navigation - user auth

Image - Navigation

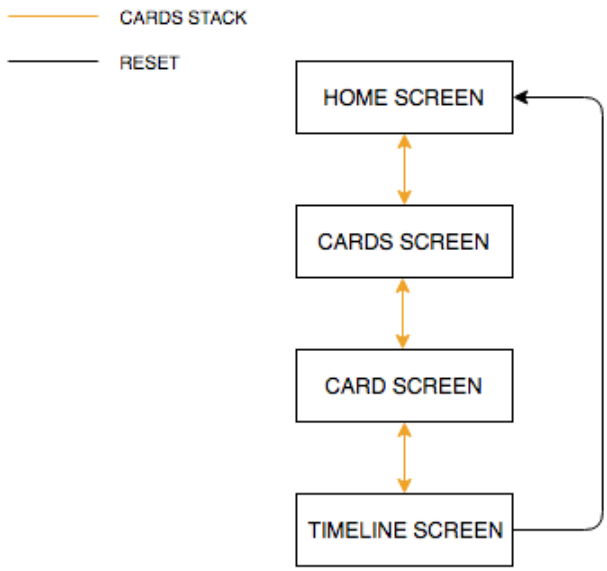
app routes



Navigation - app routes

Image - Navigation

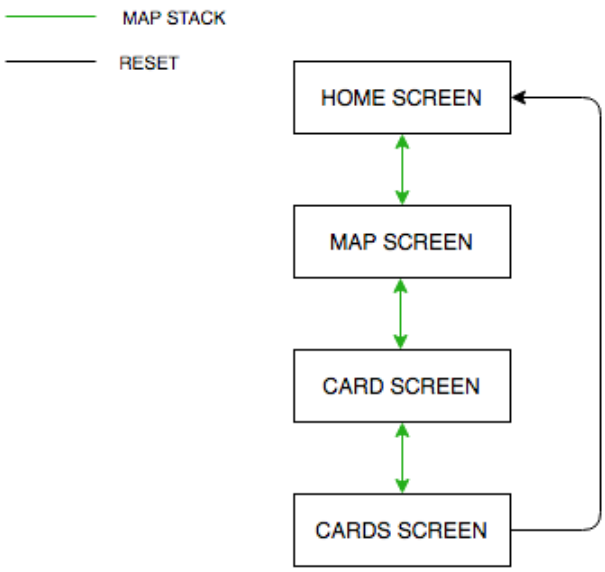
paths and stacks



Navigation - stack example - cards

Image - Navigation

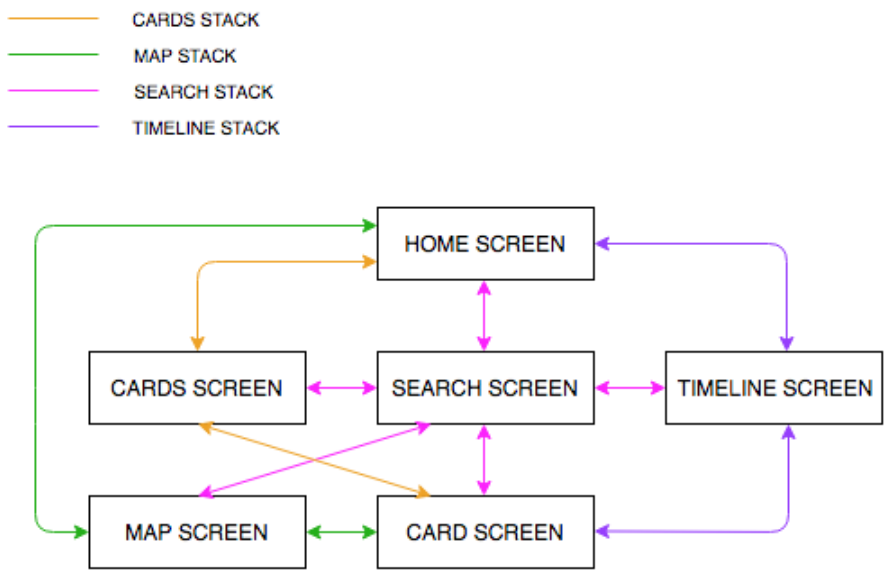
paths and stacks



Navigation - stack example - map

Fun Exercise - Navigation Stacks

app routes



Navigation - app routes - search

Consider the following relative to the outline of stacks for the app,

- How do we reconcile the option to switch to a search screen?
  - *i.e. how will it change the requirements for each of the stacks?*
- What are the benefits of multiple stacks?
- What role would a reset route play in easing stack navigation?
- what are the benefits of limiting user routes relative to stack navigation?
  - *could we improve app usage and performance by restricting certain routes?*

~ 10 minutes...

Cross-platform - modular design

ES Module pattern - intro

- React Native modules use ES6 module system
  - *Cordova may also use this module structure*
- simpler and easier to work with than CommonJS
  - *in most examples...*
- JavaScript `strict` mode is enabled by default
- `strict` mode helps with language usage - check for poor usage
  - *stops hoisting of variables*
  - *variables must be declared*
  - *function parameters must have unique name*
  - *assignment to read-only properties throws errors*
  - ...
- modules are exported with `export` statements
- modules are imported with `import` statements

Cross-platform - modular design

ES Module pattern - `export` statements

- ES6 modules are individual files
  - *expose an API using `export` statements*
- declarations are scoped to the local module
- e.g. variables declared inside a module
  - *not available to other modules*
  - *need to be explicitly exported in module API*
  - *need to be imported for usage in another module*
- export statements may only be added to *top-level* of a module
  - *e.g. not in function expression *&c.*
- cannot dynamically define and expose API using methods
  - *unlike CommonJS module system - Node.js &c.*

Cross-platform - modular design

ES Module pattern - `export default`

- common option is to export a default binding, e.g.

```
export default `hello world`
```

```
export default {
    name: 'Alice',
    place: 'Wonderland'
}
```

```
export default [
    'Alice', 'Wonderland'
]
```

```
export default function name() {
    ...
}
```

Cross-platform - modular design

ES Module pattern - bindings

- ES modules export **bindings**
  - *not values or references*
- e.g. an export of `count` variable from a module
  - *`count` is exported as a binding*
  - *export is bound to `count` variable in the module*
  - *value is subject to changes of `count` in module*
- offers flexibility to exported API
  - *e.g. `count` might originally be bound to an object*
  - *then changed to an array...*
- other modules consuming this export
  - *they would see change as `count` is modified*
  - *modified in module and exported...*
- **n.b.** take care with this usage pattern
  - *useful for counters, logs &c.*
  - *can cause issues with API usage for a module*

Cross-platform - modular design

ES Module pattern - named export

- we may define bindings for export
- instead of assigning properties to implicit export object
  - *e.g.*

```
export let counter = 0
export const count = () => counter++
```

- cannot refactor this example for named export
  - *syntax error will be thrown*
  - *e.g.*

```
let counter = 0
const count = () => counter++
export counter // this will return syntax error
export count
```

- rigid syntax helps with analysis, parsing
  - *static analysis for ES modules*

Cross-platform - modular design

ES Module pattern - export lists

- lists provide a useful solution to previous refactor issue
- syntax for list export easy to parse
- export lists of named *top-level* declarations
  - *variables &c.*
- e.g.

```
let counter = 0
const count = () => counter++
export { counter, count }
```

- also rename binding for export, e.g.

```
let counter = 0
const count = () => counter++
export { counter, count as increment }
```

- define `default` with export list, e.g.

```
let counter = 0
const count = () => counter++
export { counter as default, count as increment }
```

Cross-platform - modular design

ES Module pattern - `export from ...`

- expose another module's API using `export from...`
  - *i.e. a kind of pass through...*
- e.g.

```
export { increment } from './myCounter.js'
```

- bindings are not imported into module's local scope
- current module acts as conduit, passing bindings along export/import chain...
- module does not gain direct access to `export from ...` bindings
  - *e.g. if we call `increment` it will throw a `ReferenceError`*
- aliases are also possible for bindings with `export from...`
  - *e.g.*

```
export { increment as addition } from './myCounter.js'
```

Cross-platform - modular design

ES Module pattern - `import` statements

- use `import` to load another module
- `import` statement are only allowed in top level of module definition
  - *same as `export` statements*
  - *helps compilers simplify module loading &c.*
- import default exports
  - *give default export a name as it is imported*
  - *e.g.*

```
import counter from './myCounter.js'
```

- importing binding to `counter`
- syntax different from declaring a JS variable

Cross-platform - modular design

ES Module pattern - `import` named exports

- also imported any named exports
  - *import more than just default exports*
- named import is wrapped in braces
  - *e.g.*

```
import { increment } from './myCounter.js'
```

- also import multiple named exports
  - *e.g.*

```
import { increment, decrement } from './myCounter.js'
```

- import aliases are also supported
  - *e.g.*

```
import { increment as addition } from './myCounter.js'
```

- combine default with named
  - *e.g.*

```
import counter, { increment } from './myCounter.js'
```

Cross-platform - modular design

ES Module pattern - `import` with wildcard

- we may also import using the *wildcard* operator
  - *e.g.*

```
import * as counter from './myCounter.js'
counter.increment()
```

- name for wildcard import acts like object for module
- call module exports on wildcard


```
import * as counter from './myCounter.js'
counter.increment()
```


- common pattern for working with libraries &c.

Cross-platform - modular design

ES Module pattern - benefits & practical usage

- offers ability to explicitly publish an API
  - *keeps module content local unless explicitly exported*
- similar function to *getters* and *setters*
  - *explicit way in and out of modules*
  - *explicit options for reading and updating values...*
- code becomes simpler to write and manage
  - *module offers encapsulation of code*
- import binding to variable, function &c.
  - *then use it as normal...*
- removes need for encapsulation in main JS code
  - *e.g. with patterns such as IIFE...*
- *n.b.* need to be careful how we use modules
  - *e.g. priority for access, security, testing &c.*
  - *all now moved to individual modules...*

Mobile Design & Development - Modular Designs

Fun Exercise

Four apps with variant designs,

- Modular designs - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/modular/
  - *Anatomy*
  - *Home Design*
  - *Reminders*
  - *Watches*

For each design, consider the following

- define perceived modules for each app
  - *where might you use a module?*
- what type of modules can you define in each app?
  - *e.g. logical, structural, design, performance...*
- from a developer perspective
  - *consider primary modular groupings*
  - *does each module purpose help with testing?*
  - *can each module be decoupled from app?*
    - e.g. test and use outside of current app...

~ 10 minutes

Cordova - Extra options - build and customisation

config.xml

- `config.xml` generated as part of Cordova CLI `create` command
- additional preferences we can consider in the metadata
- modify values of these preferences
  - *configure and setup our app with greater precision and customisation*
- Cordova uses `config.xml` file to help setup structures within an app
- standard metadata for author, description, app name, and ID
- additional, useful preferences, e.g.
  - *specifying the default start file as the app loads,*
  - *a security setting for resource access*
  - *a minimum API for building the app*
  - ...

Cordova - Extra options - build and customisation

config.xml

- default start file will be specified as `index.html` in the config
- also update this value to a different file,

```
<content src="custom.html" />
```

- also update app's settings to define access privileges and domains for remote resources
  - *e.g. CSS stylesheets, JavaScript files, images, remote APIs, servers...*
  - *specifically remote resources that are not bundled with the app itself*
- Cordova refers to this setting as a **whitelist**
  - *now been moved to a specific plugin*
  - *added by default as we create an app*
- default value for this setting is global access, e.g.

```
<access origin="*" />
```

- this setting will be OK for many apps

Cordova - Extra options - build and customisation

config.xml

- may need to restrict access, e.g.
  - *due to user input in our app*
  - *remote loading of data*
  - ...
- might consider restricting our app to specific domains
- add as many `<access>` tags as necessary for our app

```
<access origin="http://www.test.com" />
<access origin="https://www.test.com" />
```

- allows our app to access anything on this domain
  - *including secure and non-secure requests*
- also add subdomains relative to a given domain
  - *simply by prepending a wildcard option*

```
<access origin="http://*.test.com" />
<access origin="https://*.test.com" />
```

- we can now update our app to restrict access to specific, required domains
  - *e.g. remote APIs, servers hosting a DB...*

Cordova - Extra options - build and customisation

config.xml

- also add further metadata and preferences to help customise our app
- already seen preferences for icons, splashscreens...
- also add further settings for
  - *plugins*
  - *specific installed and supported platforms*
  - *general preferences for all platforms*
  - *or restrict to a single platform*
- for general preferences there are five global options to consider, e.g.
  - *BackgroundColor*
  - *Android and iOS - specific fixed background colour*
  - *DisallowOverscroll*
  - *Android and iOS - prevent a rendered app from moving off the screen*
  - *Fullscreen*
  - *Android (but not iOS) - determine screen usage for an app*
  - *e.g. useful for kiosk style apps...*
  - *HideKeyboardFromAccessoryBar*
  - *iOS (but not Android) - hiding an additional toolbar above a keyboard*
  - *Orientation*
  - *Android (but not iOS) - locking an app's orientation*

Cordova - Extra options - build and customisation

config.xml

- add any necessary preferences using the `<preference>` element in our `config.xml` file

```
<preference name="fullscreen" value="true" />
```

- add as many preferences as necessary for our app's configuration
- customise our preferences for a specific platform
  - *e.g. restricting a preference to just Android or iOS*

```
<platform name="android">
  <preference name="DisallowOverscroll" value="true" />
</platform>
```

Cordova - Extra options - build and customisation

merge options

- many Cordova apps developed using a single code base
- with platform specific preferences and UI customisations
- may prefer to create a distinction in the app's design or functionality
- use **merges** options to create platform specific code, files...
- create a new folder called `merges` in our app's root directory
  - *not the www directory*
- use `merges` folder to add platform specific requirements
  - *e.g. css stylesheets*
- add sub-directory to `merges` for each supported platform
- when we build our Cordova app
  - *Cordova will check for a `merges` directory for each platform*
  - *files will replace existing in www directory*
  - *new files added to www directory*

```
config.xml
|-- hooks
|-- merges
    |__ android
    |__ ios
|-- platforms
|-- plugins
|-- www
```

Cordova - Extra options - build and customisation

merge options

- example usage might include specific stylesheets per platform
- e.g. in our app's `index.html` file add a link to a CSS stylesheet
- stylesheet file added as usual to our app's `www` directory
  - *leave this CSS file blank for the overall project*
- then add matching CSS file to each platform directory in `merges` folder
- CSS file then added to our platform specific app as it is built by Cordova

```
config.xml
|-- hooks
|-- merges
    |__ android
        |__ css
            |__ platform.css
    |__ ios
|-- platforms
|-- plugins
|-- www
    |__ css
        |__ platform.css
    |__ ...
```

- allows us to add specific
  - *styling, layout, and design requirements*
  - *for each supported platform*
- quick and easy option for platform customisation

Cordova - Extra options - build options

hooks

- we've been using Cordova's CLI tool to help
  - *create our apps, add platforms and plugins, build our apps...*
- we can customise the CLI tool using **hooks**
  - *scripts able to interact with the CLI tool for a given command and action*
- consider **Hooks** in two distinct scenarios
  - *before and after an action is executed by the CLI tool*
- for the CLI tool we might consider adding a **hook**
  - *before or after that command and action is called and executed*
- **hooks** might include automation of standard build options, tools, and commands
- e.g. automation of adding plugins to a project
  - *add a platform, and then add all required plugins using **hook***
- CLI tool checks for **hook** scripts in the `hooks` directory
- to add a **hook**
  - *create a sub-directory in the `hooks` directory - same name as a **hook***
  - *Cordova will then check for scripts to execute*
  - *scripts will be executed in alphabetical order by filename*
- **hooks** can be written in any language supported by the host computer

Cordova - Extra options - prepare for release

- finalise our Cordova app
- need to consider preparation and packing of the app
  - *ready for publication to one or more app stores*
- each major app store conceptually follows a pattern for release
- to prepare our app for publication
  - *begin by transitioning app from development version to a stable release version*
  - *app requires signing by developer with password*
  - *define ownership of app*
  - *accept responsibility for publication, contents...*
- submit the app to a store for publication
  - *required to provide descriptions for the app itself*
  - *provide a minimum of screenshots for general usage and prominent features*
  - *add supplementary information for publication of app*

Cordova - Extra options - prepare for release

Play Store

- releasing an Android app is considerably less involved than iOS
  - *developers can release and publish a vast array of application types*
- Play Store - division between preparation of the app, and then publication
- initial preparation
  - *begin by signing our app with a key - create using command line*
  - *use Cordova build tools to create a release build of our app*
- publication to store
  - *upload our app to Google's Play Store for publication*
  - *need to provide some additional supporting information*
  - *title for our app*
  - *icons*
  - *description*
  - *screenshots*
  - *...*
  - *then mark our app as published*

Cordova - Extra options - prepare for release

signing

- prepare our app for a store
  - *need to sign it using a key store and key prior to publication*
  - *key signs the app, which is saved in the keystore*
- sign our app using the Java tool, **keytool**

```
keytool -genkey -v -keystore my-app-ks.keystore -alias my-app-ks -keyalg RSA -keysize 2048
```

- command creates both the keystore and key for our app
- command arguments to consider for `-keystore` and `-alias`
- `my-app-ks.keystore`
  - *filename for the keystore*
  - *can be set to a preferred name for your app*
- `my-app-ks`
  - *name of the alias for the keystore*
  - *developer can specify their preferred name*
  - *can be a simple, plain text name for the keystore*

Cordova - Image - Keytool - Create a Keystore

```
Use "keytool -command_name -help" for usage of command_name
MacBook:networktestprod ancientlives$ keytool -genkey -v -keystore appks.keystore -alias appks -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Ancient Lives
What is the name of your organizational unit?
  [Unknown]:  Ancientlives
What is the name of your organization?
  [Unknown]:  Ancientlives
What is the name of your City or Locality?
  [Unknown]:  Chicago
What is the name of your State or Province?
  [Unknown]:  Illinois
What is the two-letter country code for this unit?
  [Unknown]:  IL
Is CN=Ancient Lives, OU=Ancientlives, O=Ancientlives, L=Chicago, ST=Illinois, C=IL correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=Ancient Lives, OU=Ancientlives, O=Ancientlives, L=Chicago, ST=Illinois, C=IL
Enter key password for <appks>
        (RETURN if same as keystore password):
[Storing appks.keystore]
```

Keytools - create a keystore

React JavaScript Library

Additional reading, material, and samples

- design thoughts
- event handling
- more composing components
- DOM manipulation
- forms
- intro to flux
- animations
- lots of samples...

References

- Cordova API docs
  - *config.xml*
  - *Globalization*
  - *Hooks*
  - *Merges*
  - *Network Information*
  - *Whitelisting*
- OnsenUI
  - *JavaScript Reference*
- React & React Native
  - *React DevTools*
  - *React Navigation*
  - *React Native Navigation*
  - *React Native Navigation - GitHub*