# Comp 422 - Software Development for Wireless and Mobile Devices

Fall Semester 2016 - Week 10 Notes

Dr Nick Hayward

# Contents

- Server-side considerations
  - *SQL or NoSQL*
  - *Redis, MongoDB*
  - *other options*
- Cordova app - NoteTaker
- Cordova app = plugins
- custom plugins
- ...

# Server-side considerations - data storage

***SQL or NoSQL***

- common database usage and storage
  - *often thought solely in terms of SQL, or structured query language*
- SQL used to query data in a relational format
- relational databases, for example MySQL or PostgreSQL, store their data in tables
  - *provides a semblance of structure through rows and cells*
  - *easily cross-reference, or relate, rows across tables*
- a relational structure to map authors to books, players to teams...
  - *thereby dramatically reducing redundancy, required storage space...*
- improvement in storage capacities, access...
  - *led to shift in thinking, and database design in general*
- started to see introduction of non-relational databases
  - *often referred to simply as* ***NoSQL***
- with NoSQL DBs
  - *redundant data may be stored*
  - *such designs often provide increased ease of use for developers*
- some NoSQL examples for specific use cases
  - *eg: fast reading of data more efficient than writing*
  - *specialised DB designs*

# Server-side considerations - data storage

***Redis - intro***

- Redis provides an excellent example of NoSQL based data storage
- designed for fast access to frequently requested data
- improvement in performance often due to a reduction in perceived reliability
  - *due to in-memory storage instead of writing to a disk*
- able to flush data to disk
  - *performs this task at given points during uptime*
  - *for majority of cases considered an in-memory data store*
- stores this data in a **key-value** format
  - *similar in nature to standard object properties in JavaScript*
- Redis often a natural extension of conventional data structures
- Redis is a good option for quick access to data
  - *optionally caching temporary data for frequent access*

# Server-side considerations - data storage

***MongoDB - intro***

- MongoDB is another example of a NoSQL based data store
  - *a database that enables us to store our data on disk*
- unlike MySQL, for example, it is not in a relational format
- MongoDB is best characterised as a **document-oriented** database
- conceptually may be considered as storing objects in collections
- stores its data using the BSON format
  - *consider similar to JSON*
  - *use JavaScript for working with MongoDB*

# Server-side considerations - data storage

***MongoDB - document oriented***

- SQL database, data is stored in tables and rows
- MongoDB, by contrast, uses **collections** and **documents**
- comparison often made between a collection and a table
- **NB:** a document is quite different from a table
- a document can contain a lot more data than a table
- a noted concern with this document approach is duplication of data
- one of the trade-offs between NoSQL (MongoDB) and SQL
- SQL - goal of data structuring is to normalise as much as possible
- thereby avoiding duplicated information
- NoSQL (MongoDB) - provision a data store, as easy as possible for the application to use

# Server-side considerations - data storage

***MongoDB - BSON***

- BSON is the format used by MongoDB to store its data
- effectively, JSON stored as binary with a few notable differences
  - *eg: `ObjectId` values - data type used in MongoDB to uniquely identify documents*
  - *created automatically on each document in the database*
  - *often considered as analogous to a primary key in a SQL database*
- `ObjectId` is a large pseudo-random number
- for nearly all practical occurrences, assume number will be unique
- might cease to be unique if server can't keep pace with number generation...
- other interesting aspect of `ObjectId`
  - *they are partially based on a timestamp*
  - *helps us determine when they were created*

# Server-side considerations - data storage

***MongoDB - general hierarchy of data***

- in general, MongoDB has a three tiered data hierarchy

1. database
   - *normally one database per app*
   - *possible to have multiple per server*
   - *same basic role as DB in SQL*
2. collection
   - *a grouping of similar pieces of data*
   - *documents in a collection*
   - *name is usually a noun*
   - *resembles in concept a table in SQL*
   - *documents do not require the same schema*
3. document
   - *a single item in the database*
   - *data structure of field and value pairs*
   - *similar to objects in JSON*
   - *eg: an individual user record*

# Server-side considerations - data storage

***Firebase - mobile platform - what is it?***

- other data store and management options now available to us as developers
- depending upon app requirements consider
  - *Firebase*
  - *RethinkDB*
- as a data store, Firebase offers a hosted NoSQL database
  - *data store is JSON-based*
  - *offering quick, easy development from webview to data store*
- syncs an app's data across multiple connected devices in milliseconds
  - *available for offline usage as well*
- provides an API for accessing these JSON data stores
  - *real-time for all connected users*
- Firebase as a hosted option more than just data stores and real-time API access
- Firebase has grown a lot over the last year
  - *many new features announced at Google I/O conference in May 2016*
  - *analytics, cloud-based messaging, app authentication*
  - *file storage, test options for Android*
  - *notifications, adverts...*

# Server-side considerations - data storage

***RethinkDB - realtime JSON - what is it?***

- RethinkDB describes itself as,

> *open source, scalable JSON database built from the ground up for the realtime web*

- RethinnkDB can be setup on a server, as a cloud service...
  - *offers flexibility, customisation, performance benefits to different teams and apps*
- paradigm shift is how an app can consume data with RethinkDB
- mobile app can now consume a continuous stream of data
  - *pushed real-time from a RethinkDB data store*
  - *create real-time, scalable apps*
- use this type of real-time model for various types of apps, e.g.
  - *gaming apps, including multi-player polling and communication*
  - *live updates for auctions, sales, and other marketplaces...*
- RethinkDB inherently different from real-time sync APIs
  - *closer to a standard database in its underlying structure, options, and general functionality*
  - *developer can use queries such as table joins, geospatial queries, subqueries...*
- build mobile apps to scale to open thousands of concurrent feeds on a single instance
- leverage clusters to enable hundreds of thousands of concurrent feeds

# Server-side considerations - data storage

***working with mobile cross-platform designs***

- how can we use Redis, MongoDB, and other data store technologies with Cordova?
- considerations for a multi-platform structure
  - *data*
  - *models*
  - *views*
- authentication
  - *user login*
  - *accounts*
  - *data*

# Cordova app - NoteTaker - v1

***notes app - intro***

## Some initial considerations for IndexedDB

- connecting
- event listeners
- creating a new DB
- adding object stores
- transactions
- adding data, reading data...
- index and querying...

# Cordova app - NoteTaker - v1

***notes app - basic requirements***

## Start building initial **Trip Notes** app,

- Cordova base app created
- add support for required platforms
  - *Android & Browser (for testing...)*
- add any initial plugins
- set icon and splashscreens in `config.xml`

## Then build out initial app,

- add home screen
  - *initial design and layout*
  - *initial widgets and elements*
  - *add some basic styling*
- add IndexedDB support
  - *create base **object stores***
  - *load some notes*
- add some more UI options
  - *change view of notes - grid, list...*
  - *sort and filter notes*
- view single note

## Then, move on to v2...

# Cordova app - NoteTaker - v1

***notes app - icon and splashscreen***

- add the splashscreen plugin,

```
cordova plugin add cordova-plugin-splashscreen
```

- then update `config.xml` file
  - *add support for required splashscreens and icons*
- set a value for the splashscreen timeout
  - *controls default AutoHide for splashscreen*

```
<preference name="SplashScreenDelay" value="3000" />
```

- option can be overridden
  - *either programmatically in JS*
  - *or in the `config.xml` file, e.g.*

```
<preference name="AutoHideSplashScreen" value="false" />
```

- default value set to `value="true"`
  - *hides splashscreen at specified value for delay*
- if `AutoHideSplashScreen` set to false
  - *needs to be hidden programmatically in the app's JS*

# Cordova app - NoteTaker - v1

***notes app - `config.xml` - splashscreens and icons***

## Add splashscreens and icons to the `config.xml` file

```
<platform name="android">
    <icon density="ldpi" src="resources/android/icon/drawable-ldpi-icon.png" />
    <icon density="mdpi" src="resources/android/icon/drawable-mdpi-icon.png" />
    <icon density="hdpi" src="resources/android/icon/drawable-hdpi-icon.png" />
    <icon density="xhdpi" src="resources/android/icon/drawable-xhdpi-icon.png" />
    <icon density="xxhdpi" src="resources/android/icon/drawable-xxhdpi-icon.png" />
    <icon density="xxxhdpi" src="resources/android/icon/drawable-xxxhdpi-icon.png" />
    <splash density="land-ldpi" src="resources/android/splash/drawable-land-ldpi-screen.png" />
    <splash density="land-mdpi" src="resources/android/splash/drawable-land-mdpi-screen.png" />
    <splash density="land-hdpi" src="resources/android/splash/drawable-land-hdpi-screen.png" />
    <splash density="land-xhdpi" src="resources/android/splash/drawable-land-xhdpi-screen.png" />
    <splash density="land-xxhdpi" src="resources/android/splash/drawable-land-xxhdpi-screen.png" /
    <splash density="land-xxxhdpi" src="resources/android/splash/drawable-land-xxxhdpi-screen.png
    <splash density="port-ldpi" src="resources/android/splash/drawable-port-ldpi-screen.png" />
    <splash density="port-mdpi" src="resources/android/splash/drawable-port-mdpi-screen.png" />
    <splash density="port-hdpi" src="resources/android/splash/drawable-port-hdpi-screen.png" />
    <splash density="port-xhdpi" src="resources/android/splash/drawable-port-xhdpi-screen.png" />
    <splash density="port-xxhdpi" src="resources/android/splash/drawable-port-xxhdpi-screen.png" /
    <splash density="port-xxxhdpi" src="resources/android/splash/drawable-port-xxxhdpi-screen.png
</platform>
<preference name="SplashScreenDelay" value="3000" />
```

***n.b.*** *we'll initially set `SplashScreenDelay` in the `config.xml` file...*

# Cordova app - NoteTaker - v1

***notes app - initial home screen***

## Initial requirements for app's home screen

***Places***

- header & navbar
- title
- icons for *create note*, *menu*...
- main content
- heading
- grid for notes
- footer

***Functionality***

- view all notes
- single note with title, snippet, &c.
- no. of notes
- switch layout of notes
- grid, list, &c.
- filter and sort notes
- ...

# Cordova app - NoteTaker - v1 - jQuery Mobile

***notes app - index.html - part 1***

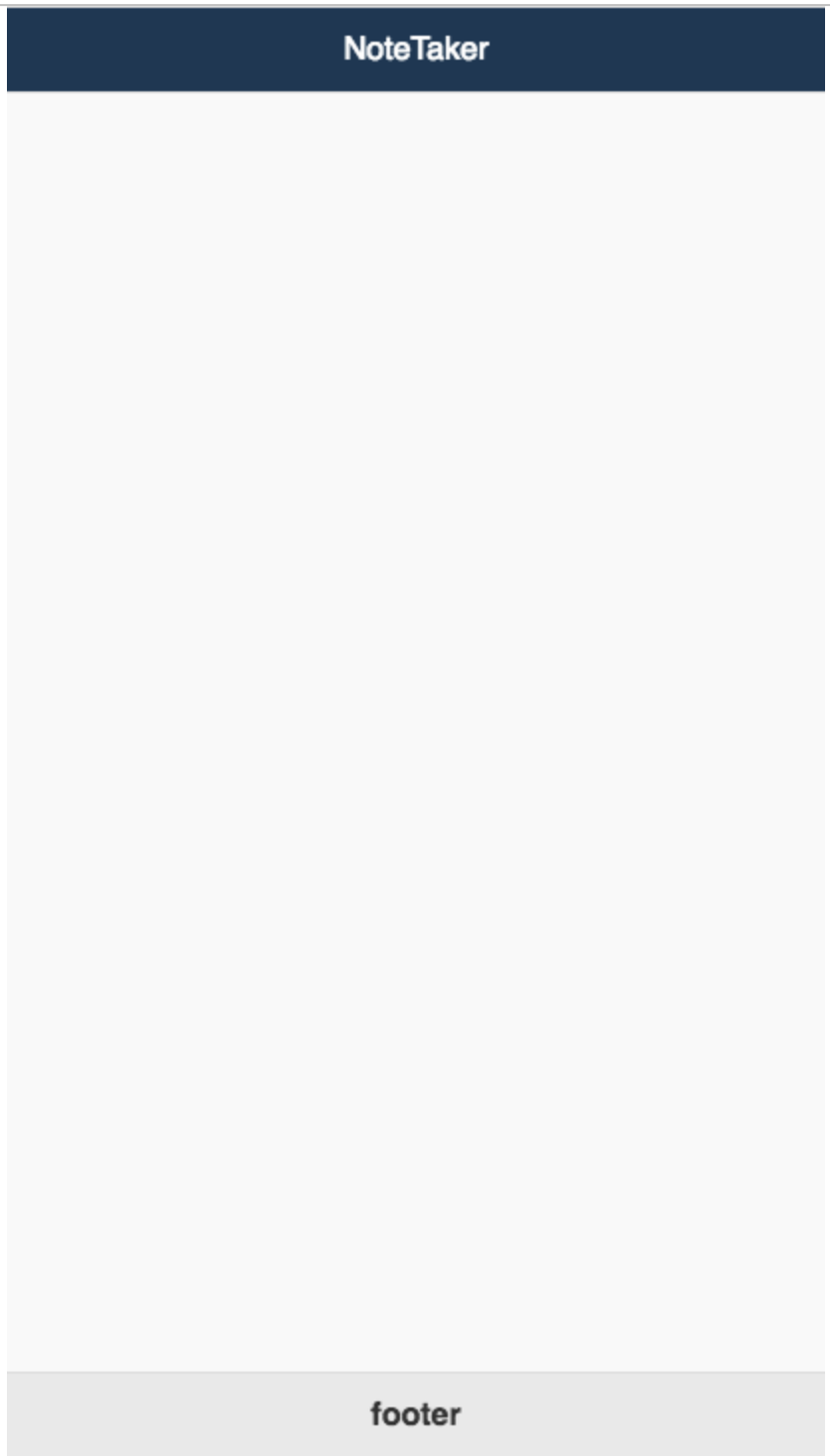
- update app's `index.html` page using jQuery Mobile structure

```
<body>
    <!-- app pages -->
    <!-- page1 - home screen -->
    <div data-role="page" id="home">
        <div data-role="header">
            <h3>NoteTaker</h3>
        </div><!-- /header -->
        <div role="main" class="ui-content">
            ...
        </div><!-- /content -->
        <div data-role="footer" data-position="fixed" class="page-footer" >
            <h5>footer</h5>
        </div><!-- /footer -->
    </div><!-- /page1 - home screen -->
</body>
```

- modify the initial CSS for the app, and add specifics for jQuery Mobile

```
/* remove default all uppercase...*/
body {
    text-transform: none;
}
/* customise page header */
.ui-page .ui-header {
    background-color: #1a3852;
}
/* customise header title */
.ui-header .ui-title {
    font-weight: normal;
    text-shadow: none;
    color: #fff;
}
```

# Image - NoteTaker - Home Screen 1 - jQuery Mobile




NoteTaker app - initial home screen 1

# Cordova app - NoteTaker - v1 - OnsenUI setup

- now start to introduce alternative UI frameworks for developing cross-platform apps
  - *first option is **OnsenUI***
- supports development with
  - *JavaScript*
  - *Angular 1 & 2*
  - *React*
- setup initial Cordova project, add platforms, plugins &c.
- then add OnsenUI to newly created project
  - *use either NPM or Bower*
- install Bower using the following terminal command

```
npm install -g bower
```

- use Bower to install UI components and dependencies for building our OnsenUI projects

```
cd www
bower install onsenui
```

- `bower_components` directory created in the project's `www` directory

# Cordova app - NoteTaker - v1 - OnsenUI

***add OnsenUI files to project***

- we need to add OnsenUI to our project
  - *add framework's CSS and JS*

```
...
<head>
...
<!-- setup css -->
<link rel="stylesheet" type="text/css" href="css/index.css">
<link rel="stylesheet" type="text/css" href="bower_components/onsenui/css/onsenui.css">
<link rel="stylesheet" type="text/css" href="bower_components/onsenui/css/onsen-css-components.css
<link rel="stylesheet" type="text/css" href="css/style.css">
<!-- setup js -->
<script type="text/javascript" src="cordova.js"></script>
<script type="text/javascript" src="bower_compoents/onsenui/js/onsenui.js"></script>
<script type="text/javascript" src="js/app.js"></script>
<title>NoteTaker</title>
</head>
...
```

- adding required OnsenUI framework CSS and JavaScript
  - *allow us to use specific OnsenUI elements, structures, methods...*
- update these files to support custom themes and styles
  - *create using the* ***OnsenUI Theme Roller***
  - *add app specific styles to* `css/style.css`

# Cordova app - NoteTaker - v1 - OnsenUI

***OnsenUI concepts - `ons` object, elements, page content...***

- `ons` object is exposed as an integral part of working with OnsenUI
  - *part of the core library*
  - *use with available exposed methods - e.g. with tabbar, page, various utilities...*
- OnsenUI specific elements are all custom
  - *defined with tag prefix of `<ons-`*
  - *elements still include attribute and class patterns*
  - *provide properties and events as expected for standard HTML*
- still traverse an OnsenUI created DOM as expected
- OnsenUI includes many UI components
  - *e.g. `<ons-navigator>` component provides management of page stack and app navigation*

```
<ons-navigator id="navigator" page="page1.html"></ons-navigator>
```

- also add many types of pre-built OnsenUI components, e.g.
  - *buttons, dialogs, notifications*
  - *toolbars, pages, splitters, tabs*
  - *forms, lists...*
- helper properties and functions such as *infinite scroll*
  - *provide pre-built ways to manage content, rendering, layout, and general development...*

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - index.html - part 1***

- add a page using the <ons-page> element
  - *becomes initial container for a single page*
  - *root element for other page elements*
- add a toolbar to the top or foot of a page
  - *using the <ons-toolbar> element*

```
<!-- page root -->
<ons-page>
  <!-- page toolbar -->
  <ons-toolbar>
  ...
  </ons-toolbar>
</ons-page>
```

- update our toolbar with a menu icon, title, and search icon

```
<!-- page toolbar -->
<ons-toolbar>
    <div class="left">
    <ons-toolbar-button>
      <!-- hamburger menu -->
      <ons-icon icon="md-menu"></ons-icon>
    </ons-toolbar-button>
  </div>
  <div class="center">NoteTaker</div>
  <div class="right">
    <ons-toolbar-button>
      <!-- search option -->
      <ons-icon icon="md-search"></ons-icon>
    </ons-toolbar-button>
  </div>
</ons-toolbar>
```

# Image - NoteTaker - add initial navbar - OnsenUI


NoteTaker app - add initial navbar

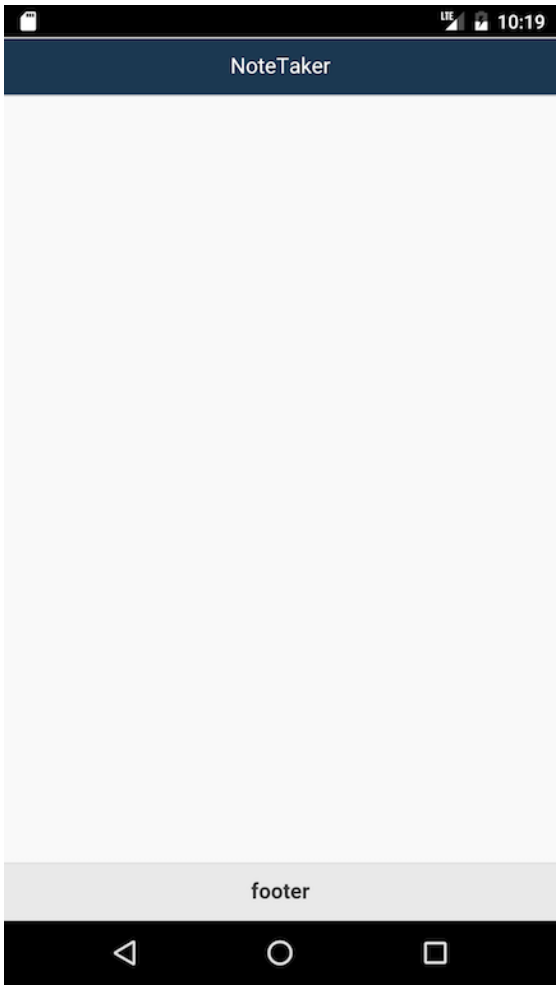
# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - index.html - part 2***

## initial homepage,

```
<ons-page id="home">
  <!-- page toolbar -->
  <ons-toolbar>
    <div class="left">
      <ons-toolbar-button>
        <!-- hamburger menu -->
        <ons-icon icon="md-menu"></ons-icon>
      </ons-toolbar-button>
    </div>
    <div class="center">NoteTaker</div>
    <div class="right">
      <ons-toolbar-button>
        <!-- search option -->
        <ons-icon icon="md-search"></ons-icon>
      </ons-toolbar-button>
    </div>
  </ons-toolbar>
  <!-- home page content -->
  <h4>notes</h4>
  ...
  <ons-button id="push-button">Create</ons-button>
</ons-page>
```

# Image - NoteTaker - Statusbar - default

| jQuery Mobile | OnsenUI |
| --- | --- |
|  |  |

# Cordova app - NoteTaker - v1

***notes app - statusbar***

- add Cordova's `statusbar` plugin
  - *helps customise statusbar at the top of the UI*

```
cordova plugin add cordova-plugin-statusbar
```

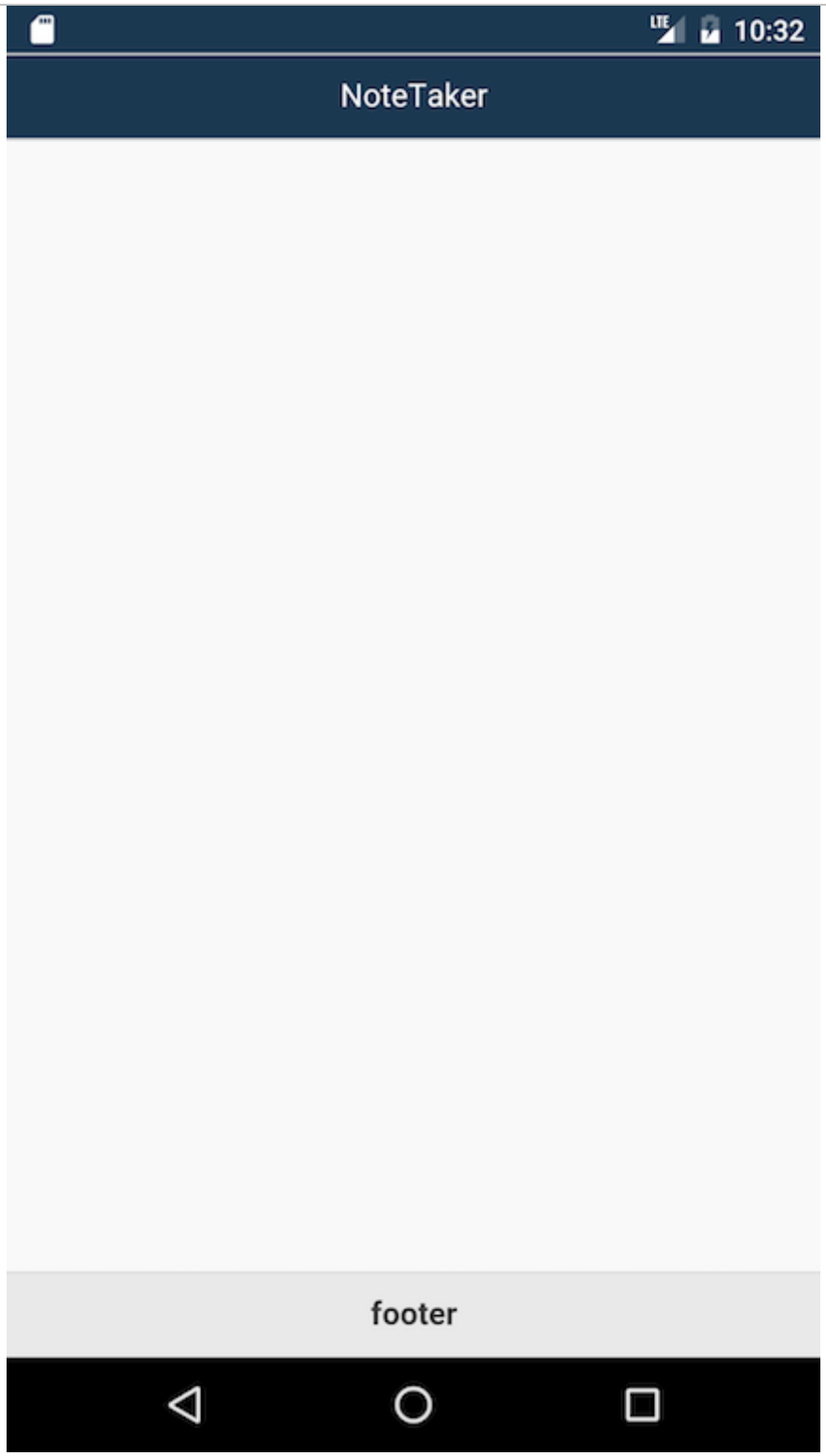
- update app's `config.xml` file to modify background colour

```
<preference name="StatusBarBackgroundColor" value="#1a3852" />
```

- status bar now matches aesthetics of app's colour scheme
- many other options detailed in the Cordova API

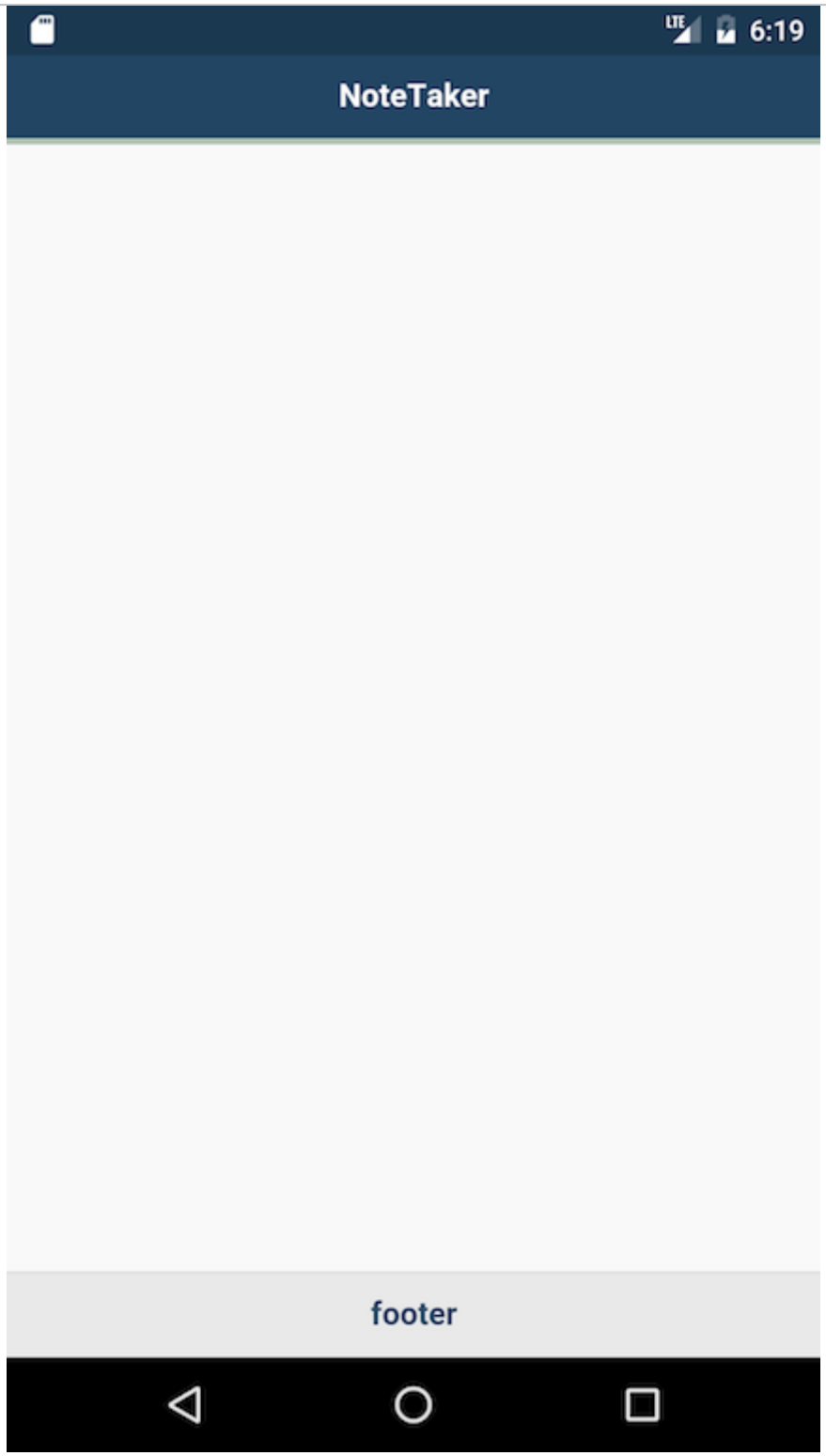
## ***n.b.*** *Cordova Plugin Statusbar*

# Image - NoteTaker - Statusbar - custom




NoteTaker app - custom statusbar

# Image - NoteTaker - Statusbar - custom




NoteTaker app - custom statusbar

# Cordova app - NoteTaker - v1 - jQuery Mobile

***notes app - index.html - part 2***

- start to add content to home screen
  - *including create button for new note, grid layout for notes, second page for create note form...*

```
<!-- header -->
<div data-role="header" class="ui-nodisc-icon">
    <h3>NoteTaker</h3>
    <a href="#create" data-transition="slideup"
    class="ui-btn ui-icon-plus ui-btn-icon-notext ui-btn-right">create</a>
</div><!-- /header -->
```

- updates header - adds *plus* icon for *create note* option
- need to match app's aesthetics
  - *need to modify svg properties for underlying icon*

```
/* custom svg for button - plus - custom fill colour = 1f4463*/
.ui-icon-plus:after {
    background-image: url("...polygon%20fill%3D%22%231f4463...");
}
```

- modify *polygon fill* to fit our app's colour scheme

```
polygon%20fill%3D%22%231f4463
```

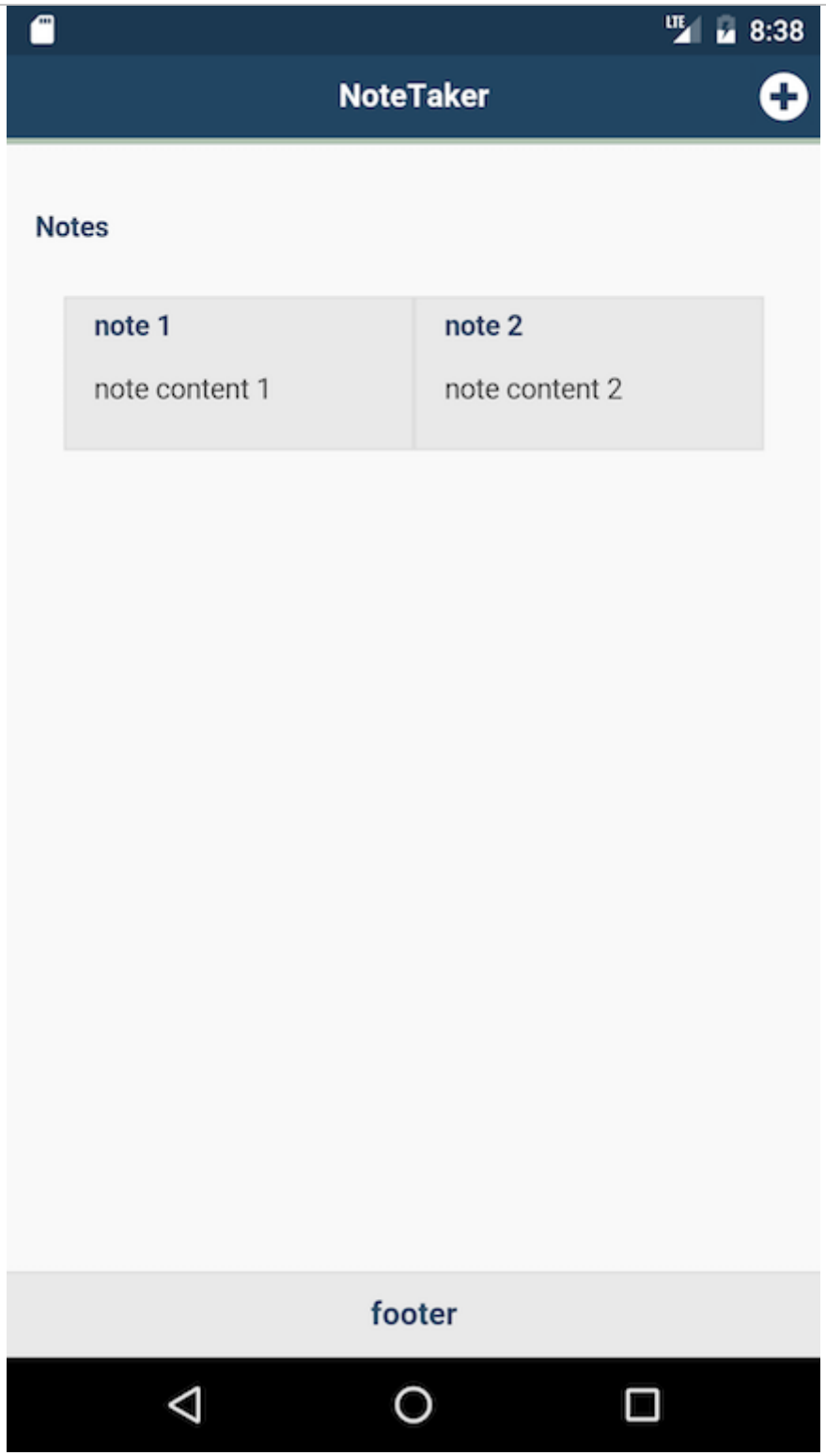
# Cordova app - NoteTaker - v1 - jQuery Mobile

***notes app - index.html - part 3***

- notes rendered by default using a grid pattern
  - *contains note's title and snippet of content*
- grid uses the following pattern
  - *abstract in JS as we read notes from specified data store*

```
<div role="main" class="ui-content">
    <h4>Notes</h4>
    <!-- output available notes -->
    <div id="notes" class="ui-body">
        <!-- grid layout for notes -->
        <div class="ui-grid-a">
            <!-- left column -->
            <div class="ui-block-a">
                <div class="ui-bar ui-bar-a">
                    <h5>note 1</h5>
                    <p>note content 1</p>
                </div>
            </div>
            <!-- right column -->
            <div class="ui-block-b">
                <div class="ui-bar ui-bar-a">
                    <h5>note 2</h5>
                    <p>note content 2</p>
                </div>
            </div>
        </div>
    </div>
</div><!-- /content -->
```

# Image - NoteTaker - update `index.html` - jQuery Mobile




NoteTaker app - update `index.html`

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - index.html - part 3***

- add a second page to allow a user to create their notes
- follow the same pattern as the home page

```
<ons-page id="create">
  <ons-toolbar>
    <div class="left"><ons-back-button>Back</ons-back-button></div>
    <div class="center"></div>
  </ons-toolbar>
  <!-- create note page -->
  <h4>create note</h4>
  ...
</ons-page>
```

- update toolbar relative to page requirements

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - page lifecycle***

## <ons-page> element provides the following events at different points during the lifecyle of a page

- `init`
  - *fired after page is added to DOM*
- `destroy`
  - *fired prior to page being removed from the DOM, and just before the page is destroyed*
- `show`
  - *fired every time <ons-page> comes into the view*
- `hide`
  - *fired every time <ons-page> disappears from the view*

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - JS and initial listeners***

- using OnsenUI and Cordova we need to consider specific events as an app starts...
- DOM loads - attach listener for Cordova's `deviceready`

```
//add listener to DOM - check deviceready event...continue with app logic
document.addEventListener('deviceready', onDeviceReady, false);
```

- JS checks require DOM has loaded
  - *otherwise we can't attach listeners for such events*
- `deviceready` event crucial to app loading
- if `deviceready` event not completed successfully
  - *not able to attach other listeners to the DOM*
- Cordova modifies the default event listener for an app's webview
  - *modifications include handling of special events*
  - *e.g. attaching and removing listeners...*
- as noted in the Cordova JS library,

> *Intercept calls to addEventListener + removeEventListener and handle deviceready, resume, and pause events.*

- `deviceready` event returns successful
  - *next listener needs to check that OnsenUI `init` event has fired*

```
document.addEventListener('init', function(event) {
  //check defined initial page for app...
  if (event.target.matches('#home')) {
     ...
```

```
  }
}, false);
```

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - notetaker.js***

- add the logic for our OnsenUI based app
- start by checking for Cordova's `deviceready` event
- then check the `init` event that OnsenUI provides
  - *use this to check for our home page*

```
...
//cordova - add listener to DOM & check deviceready event
document.addEventListener('deviceready', function(event) {
    //prevent any bound defaults
    event.preventDefault();
    console.log("cordova checked...device ready");

    //call as ons-page added to DOM...
    function onsInit(event) {
        //properties - initial page load
        var page = event.target;
    //check IndexedDB
    //set navigation
        //check for home page
        if (page.matches("#home")) {
        //ons.notification.alert('init checked...homepage ready');
            console.log("home page is now attached to the DOM...");
      } else {
            console.log("away from home page...");
        }
    }
    //onsen - init event is fired after ons-page attached to DOM...
    document.addEventListener('init', onsInit, false);
}, false);
...
```

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - navigation structure - part 1***

- two initial pages for our NoteTaker app - navigation from place to place
  - *update our `index.html` page's structure*
  - *updare app's logic in the `notetaker.js` file*
- provide support for multi-page navigation - three available navigation patterns
  - *add a navigator, tabbar, or splitter component*
  - *each component provides a **frame** for a defined place within our app*
  - *dynamically update the inner content of each frame*
- a **frame** normally contains a `<ons-page>` component
  - *we can also nest multiple navigation components*

# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - navigation structure - part 2***

- add a navigator component
  - *define each individual page within our single page app's `index.html`*
- defined each page for navigation using `<ons-template>` component
- updated structure is as follows

```
<ons-navigator id="navigator" page="home.html"></ons-navigator>
<!-- home page -->
<ons-template id="home.html">
  <ons-page id="home">
    ...
  </ons-page>
</ons-template>
<!-- create note page -->
<ons-template id="create.html">
  <ons-page id="create">
    ...
  </ons-page>
</ons-template>
```

- for a SPA
  - *each `<ons-page>` component's ID attribute replaces a full URL to a separate page*

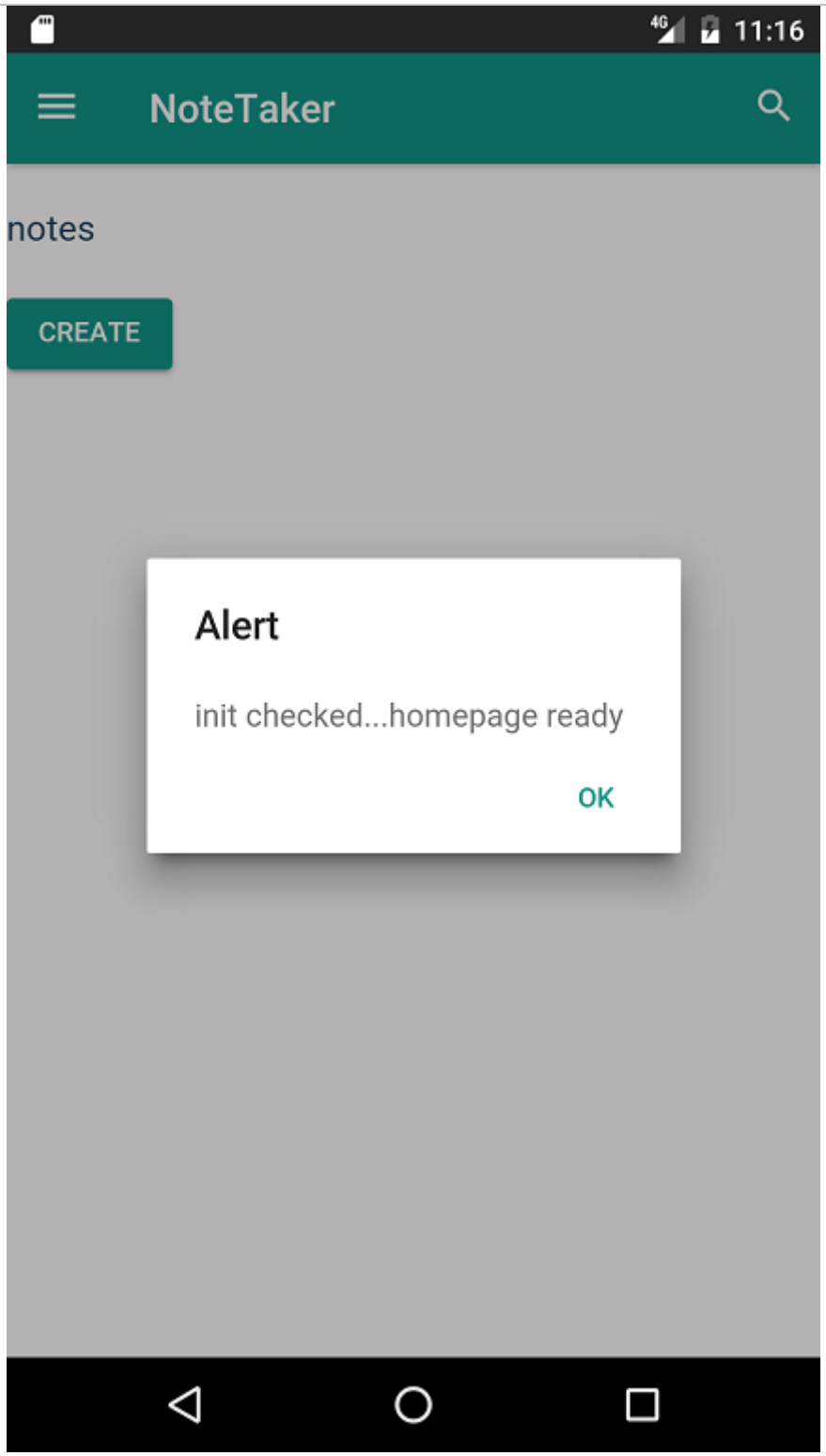
# Cordova app - NoteTaker - v1 - OnsenUI

***notes app - navigation logic***

- update our app's logic to listen for navigation events
- add a custom function `onsNav()`
  - *sets required **stack-based** navigation*
  - *OnsenUI uses to keep track of page push and pop requests within an app*

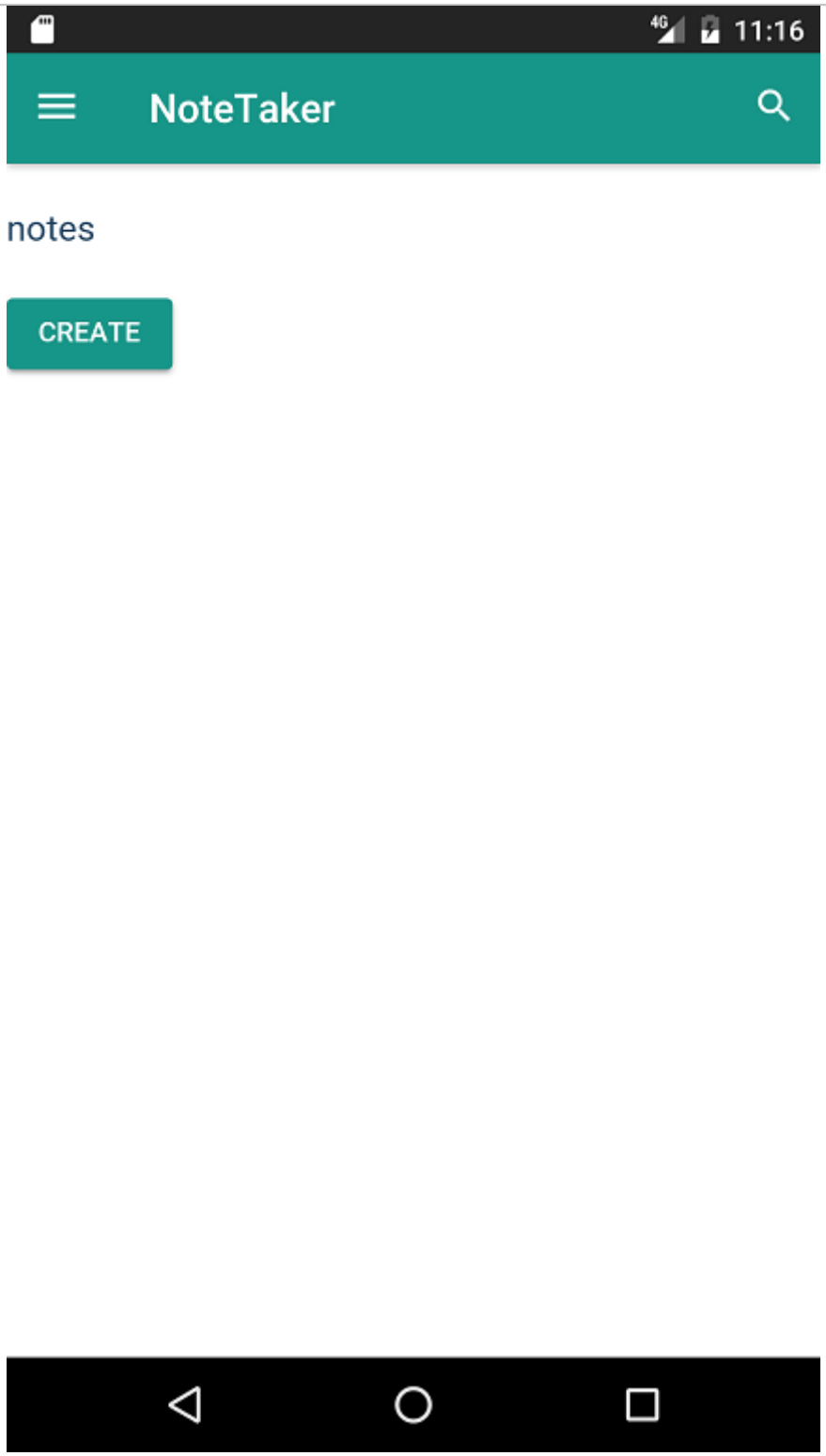
```
//onsen - set stack-based navigation
function onsNav(page) {
  if (page.id === 'home') {
    page.querySelector('#push-button').onclick = function() {
      document.querySelector('#navigator').pushPage('create.html', {data: {title: 'Create Note'}})
    };
  } else if (page.id === 'create') {
    page.querySelector('ons-toolbar .center').innerHTML = page.data.title;
      console.log("page title = "+ page.data.title);
  }
}
```

# Image - NoteTaker - check `init` event - OnsenUI




NoteTaker app - check init event

# Image - NoteTaker - load home page - OnsenUI




NoteTaker app - load home page

# Image - NoteTaker - add navigation - OnsenUI


NoteTaker app - add navigation

# References

- Cordova API
  - *Statusbar plugin*
  - *Storage*
- Whitelist plugin
- GitHub
- cordova-plugin-indexeddb
- MDN
- IndexedDB
- OnsenUI
- OnsenUI v2
- JavaScript Reference
- Theme Roller
- W3
- Web storage specification