NativeScript.org

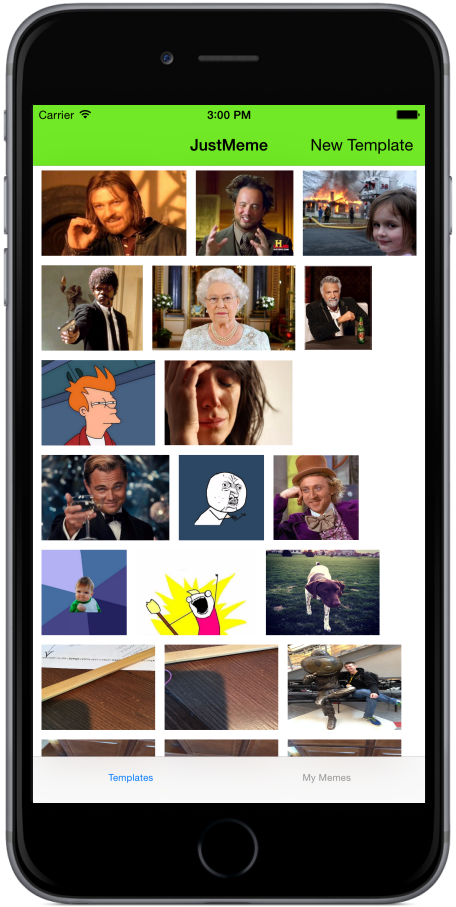
Going Beyond The {N} Basics

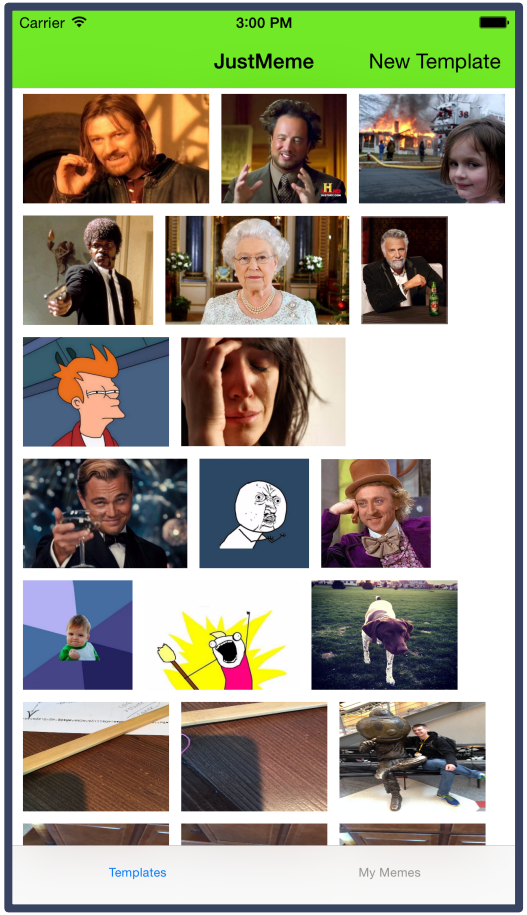
Summer of NativeScript

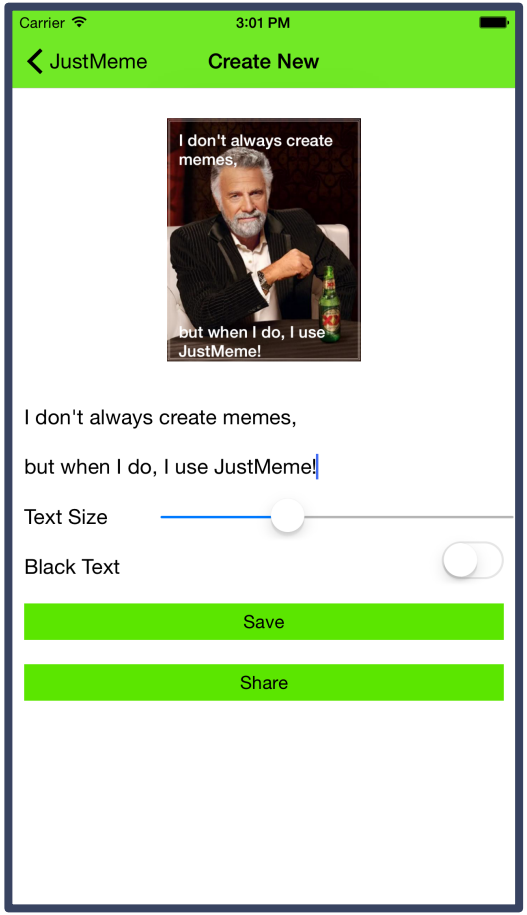
Clark Sell, TJ VanToll

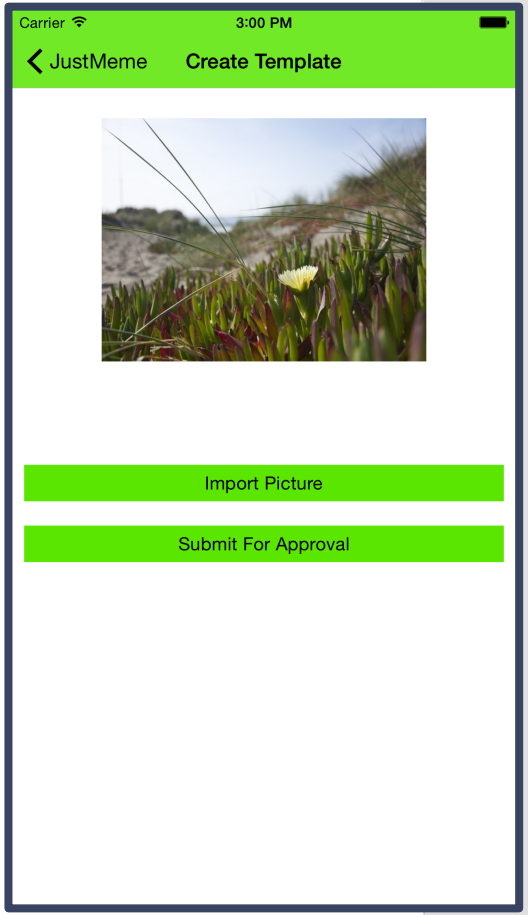

#worksOnMyPhone

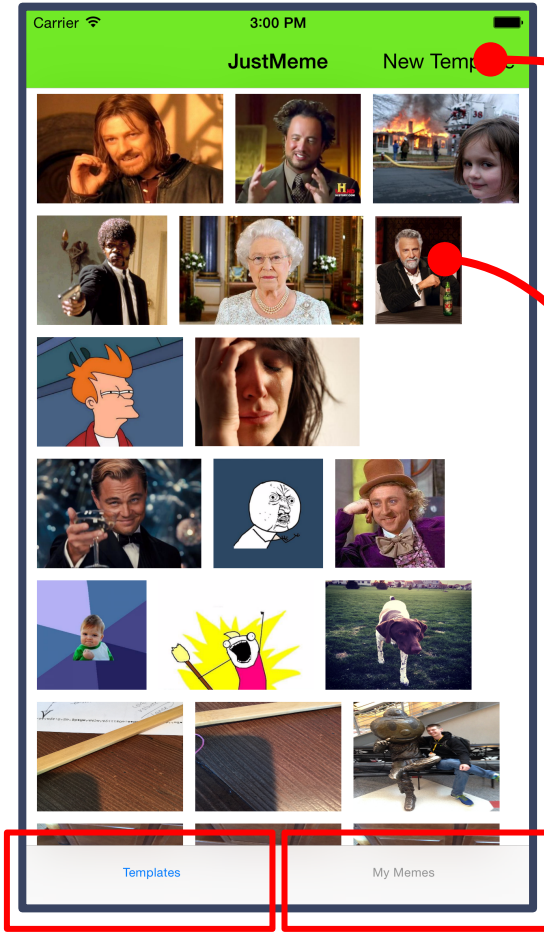
JustMeme

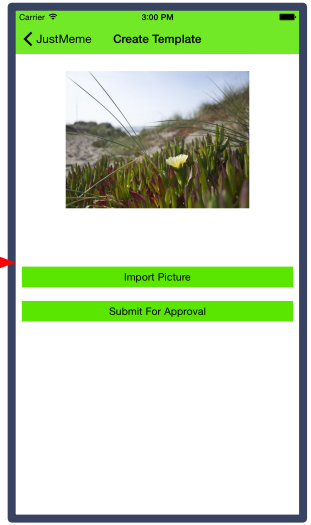


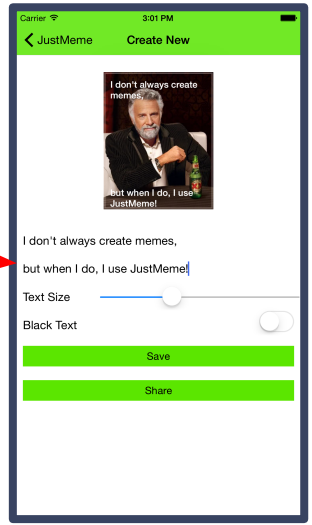

Carrier
3:00 PM
JustMeme
New Template
Templates
My Memes


Carrier
3:01 PM
JustMeme
Create New
I don't always create memes,
but when I do, I use JustMeme!
I don't always create memes,
but when I do, I use JustMeme!
Text Size
Black Text
Save
Share


Carrier
3:00 PM
JustMeme
Create Template
Import Picture
Submit For Approval


Carrier
3:00 PM
JustMeme
Templates
My Memes

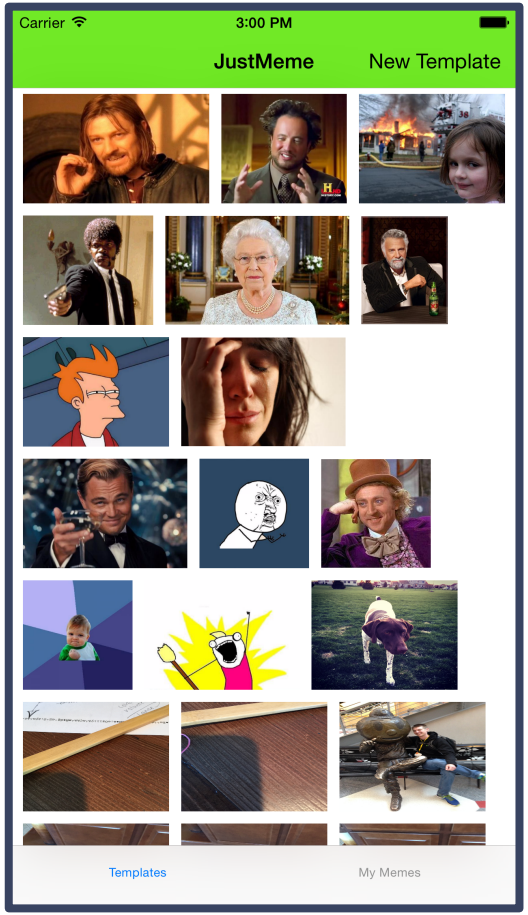

Carrier
3:00 PM
JustMeme
Create Template
Import Picture
Submit For Approval


Carrier
3:01 PM
JustMeme
Create New
I don't always create memes,
but when I do, I use JustMeme!
Text Size
Black Text
Save
Share

Images baked into the App

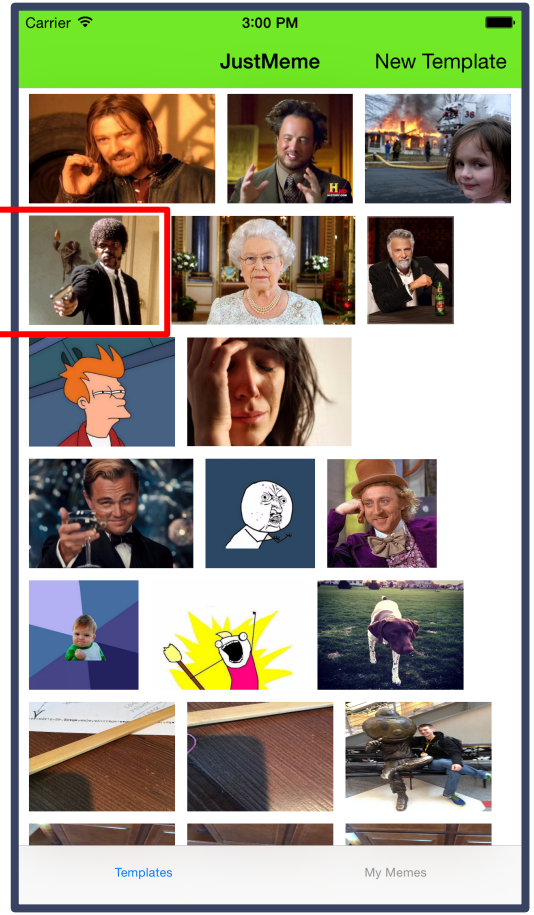
Images you saved locally

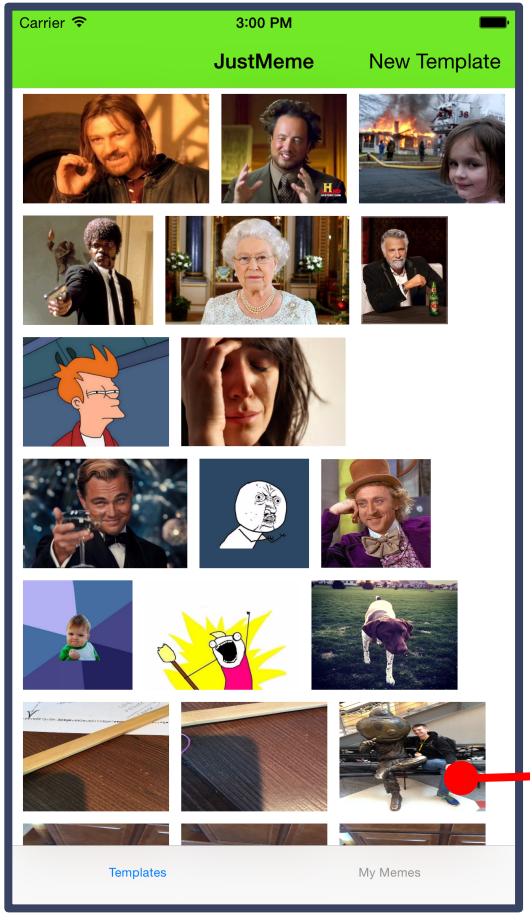
Images pulled from the cloud and saved locally



What is going on here???

1. Load the page
2. Cleanup any elements
3. Read local storage locations
4. Loop on items found
5. Create image elements
6. Assign a gesture to the image element
7. Assign a callback
8. Add to the right layout
    a. Let the WrapLayout do the work.

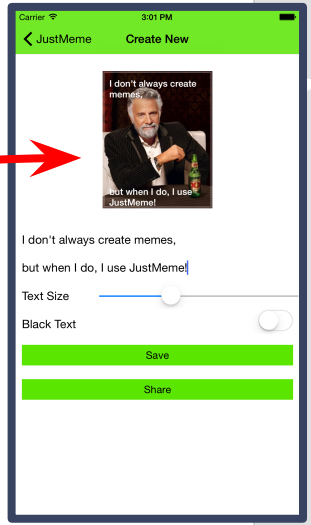




oh those callbacks....

"I don't normally just call functions, but when I do I expect you to pass me along."

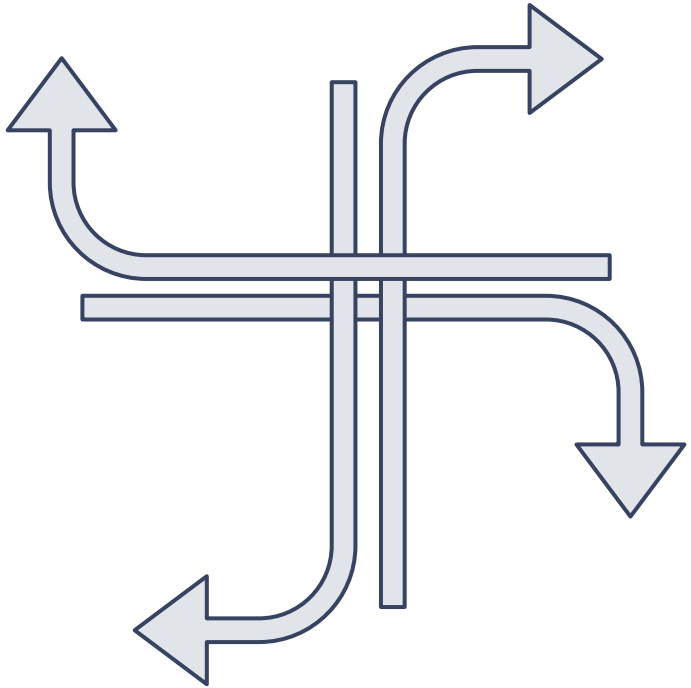
ImageSource FTW!



Our Order Of Operations For Today

- Navigation
- Events
- Layout
- Modules / Controls
- Data Binding
- CSS

Navigation



Navigation

What do we really care about?

- How to get from here to there?
- How can I pass something along the way?
- What events do I pass during my trip?
- Get me back home!

Navigation - the {N} way

1. Get the frame
2. Set some context
3. Navigate

```
var frameModule = require("ui/frame");

frameModule.topmost().navigate({
    moduleName: "./components/create-meme/create-meme",
    context: imageSource,
    animated: true
});
```

Navigation - take me back

Easily pop back to the last thing in the stack.

1. Get the top frame
2. goBack()

```
var frameModule = require("ui/frame");
frameModule.topmost() .goBack();
```

References

- {N} Architecture and Navigation
- Frame Module

- Navigation in JustMeme
- JustMeme - Navigate
- JustMeme - goBack()

Lifecycle Events

Lifecycle Events

Application Level

- Events which fire only based on the scope of the application itself.

View Level

- Events which fire every time a page is brought into “focus”.

** there are also events on things like controls*

Application Scope

- onLaunch
- onResume
- onSuspend
- onExit
- onUncaughtError
- onLowMemory

**not in order of execution*

Wiring it up!

1. Get the application object
2. Assign your function

app.js

```
var application = require("application");

application.onExit = function () {
    var analyticsMonitor = require("./shared/analytics");
    analyticsMonitor.stop();
};
```

Page / View Level Events

- loaded
- navigatingTo
- navigatedTo
- navigatingFrom
- navigatedFrom
- unloaded

* *in order of execution*

Wiring up it up!

[yourView].xml

```
<Page
    xmlns="http://www.nativescript.org/tns.xsd"
    loaded="load"
    unloaded="unloaded"
    navigatedTo="navigatedTo">
```

[yourView's].js

```
exports.load = function(args) {
    //Seriously Awesome Stuff
};
```

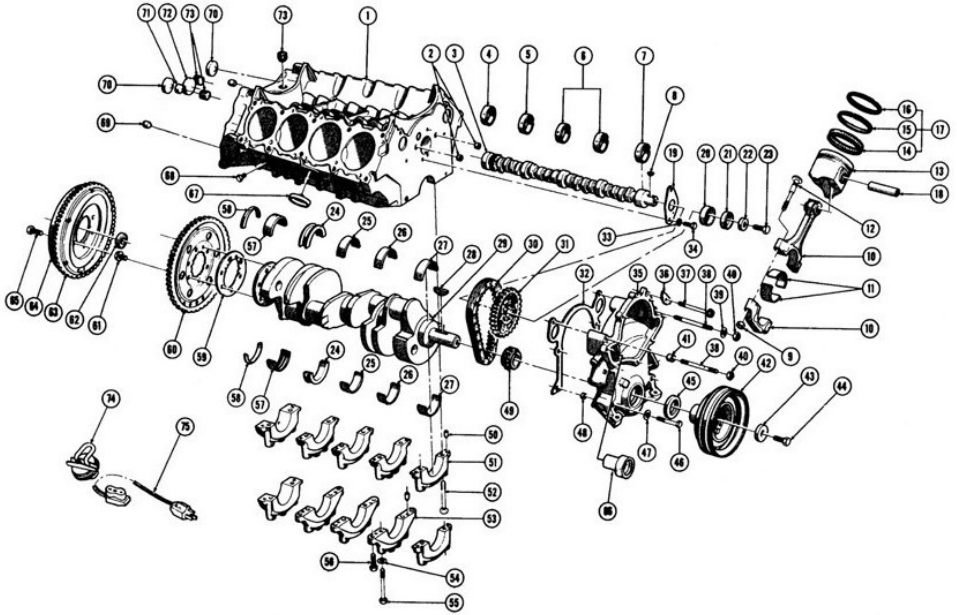
References

- Application Level Events
  - Application Callbacks
- View Events

- Page Events
- Weak Events

- JustMeme - App Events
- JustMeme - View Events

Layout*ing*



Layouting

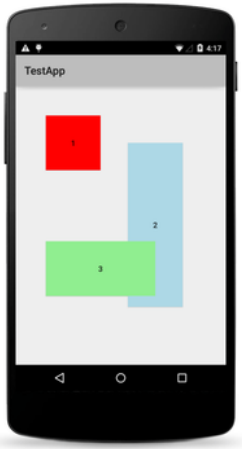
Layouting is the process of measuring and positioning the child views of a Layout container. Layouting is an ***intensive*** process whose speed and performance depend on the count of the children and the complexity of the layout container.

For example, a simple layout container such as AbsoluteLayout might perform better than a more complex layout container, such as GridLayout.

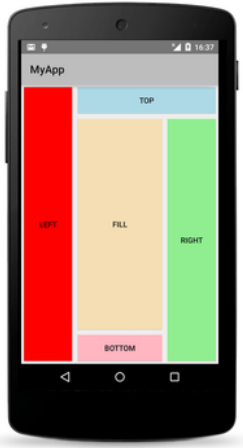
Layouting completes in two passes - a measure pass and a layout pass. Every layout container provides its own onMeasure() and onLayout() methods to achieve its own specific layouting.

The 5 Layouts

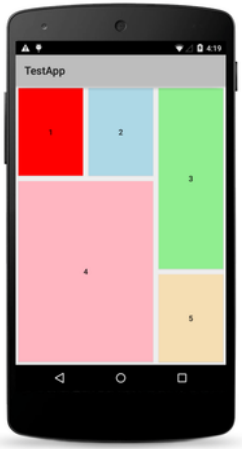
Absolute



Dock

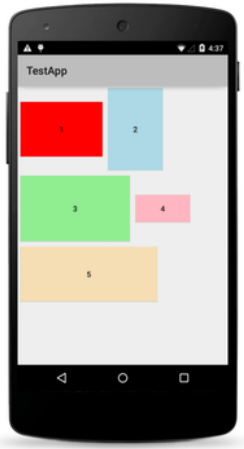


Grid



Stack

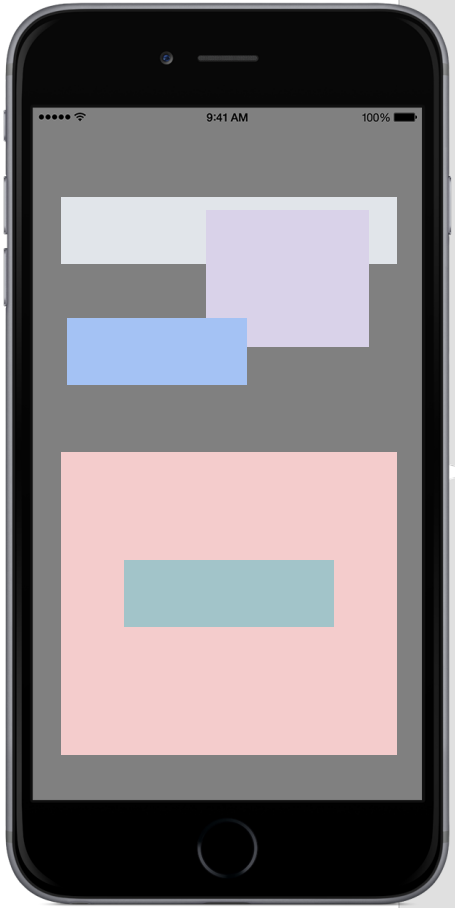
Wrap



AbsoluteLayout

This layout lets you set exact locations (left/top coordinates) for its children.

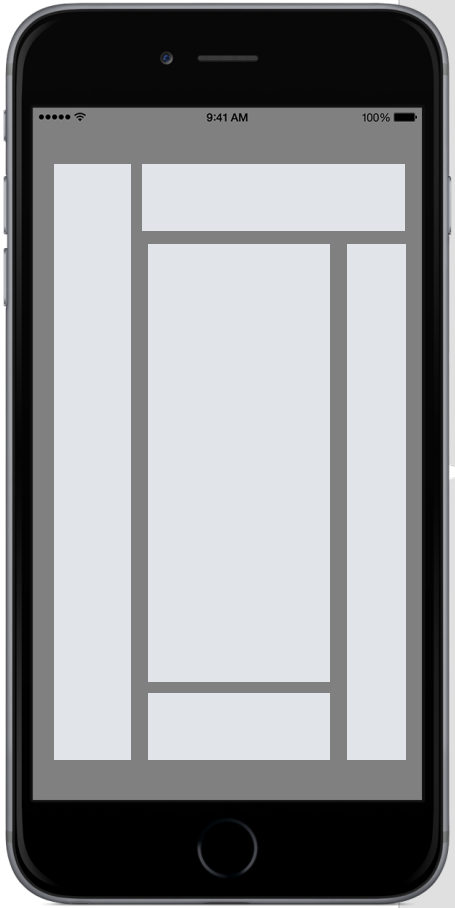
```
var absoluteLayout = new absoluteLayoutModule.AbsoluteLayout();
absoluteLayout.width = 230;
absoluteLayout.height = 230;

var label;
label = new labelModule.Label();
label.width = 100;
label.height = 100;
label.style.backgroundColor = new colorModule.Color("Red");

absoluteLayoutModule.AbsoluteLayout.setLeft(label, 10);
absoluteLayoutModule.AbsoluteLayout.setTop(label, 10);
absoluteLayout.addChild(label);
```



DockLayout

This layout arranges its children at its outer edges and allows its last child to take up the remaining space.

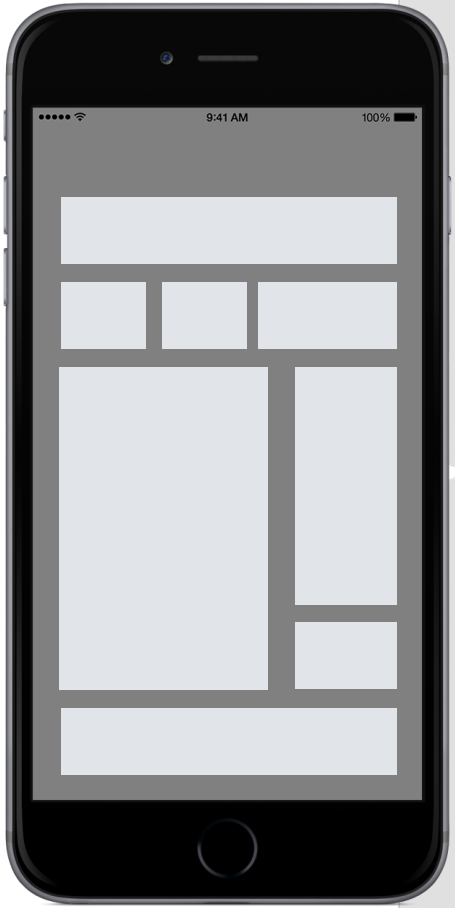
```
<Page>
 <DockLayout stretchLastChild="true" >
   <Button dock="left" text="left" />
   <Button dock="top" text="top" />
   <Button dock="right" text="right" />
   <Button dock="bottom" text="bottom" />
   <Button text="fill" />
 </DockLayout >
</Page>
```



GridLayout

This layout defines a rectangular layout area that consists of columns and rows.

```
<GridLayout columns="80, *, auto" rows="auto, *" >
  <Button col="0" />
  <Button col="1" />
  <Button col="2" />
  <Button row="1" colSpan="3" />
</GridLayout>
```



StackLayout

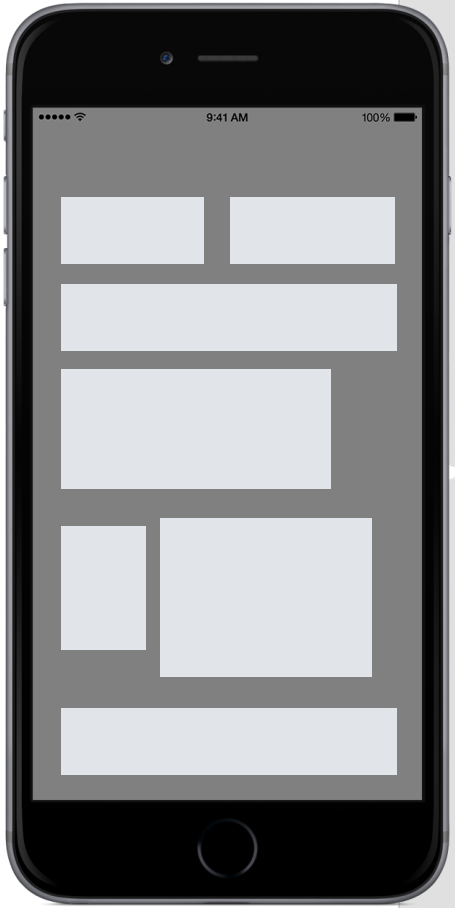
This layout arranges its children horizontally or vertically. The direction is set with the orientation property.

```
<Page>
  <StackLayout orientation="horizontal">
    <Label text="This is Label 1" />
    <Label text="This is Label 2" />
  </StackLayout>
</Page>
```

WrapLayout

This layout positions its children in rows or columns, based on the orientation property, until the space is filled and then wraps them on a new row or column.

```
<WrapLayout itemHeight="100" horizontalAlignment="center">
  <Label text="Label 1" />
  <Label text="Label 2" />
  <Label text="Label 3" />
  <Label text="Label 4" />
</WrapLayout>
```



Layout References

- Layouting

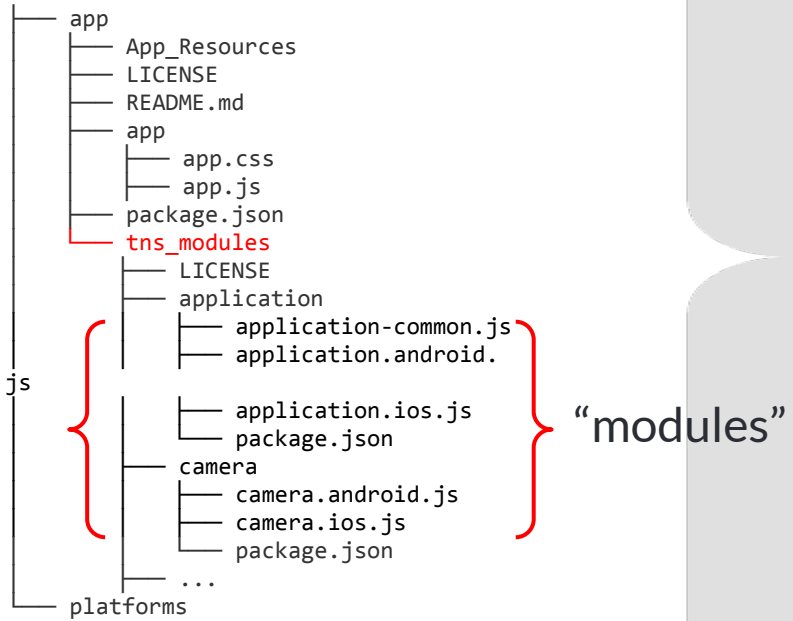
- JustMeme - WrapLayout
- JustMeme - A Grid in a Stack

~~Modules~~ Widgets

Module || Widget?

Well… both.

Generically speaking widgets are actually {N} modules. They work the same, are structured the same but a “widget” typically refers to a UI element. They are all contained in *tns_modules*.

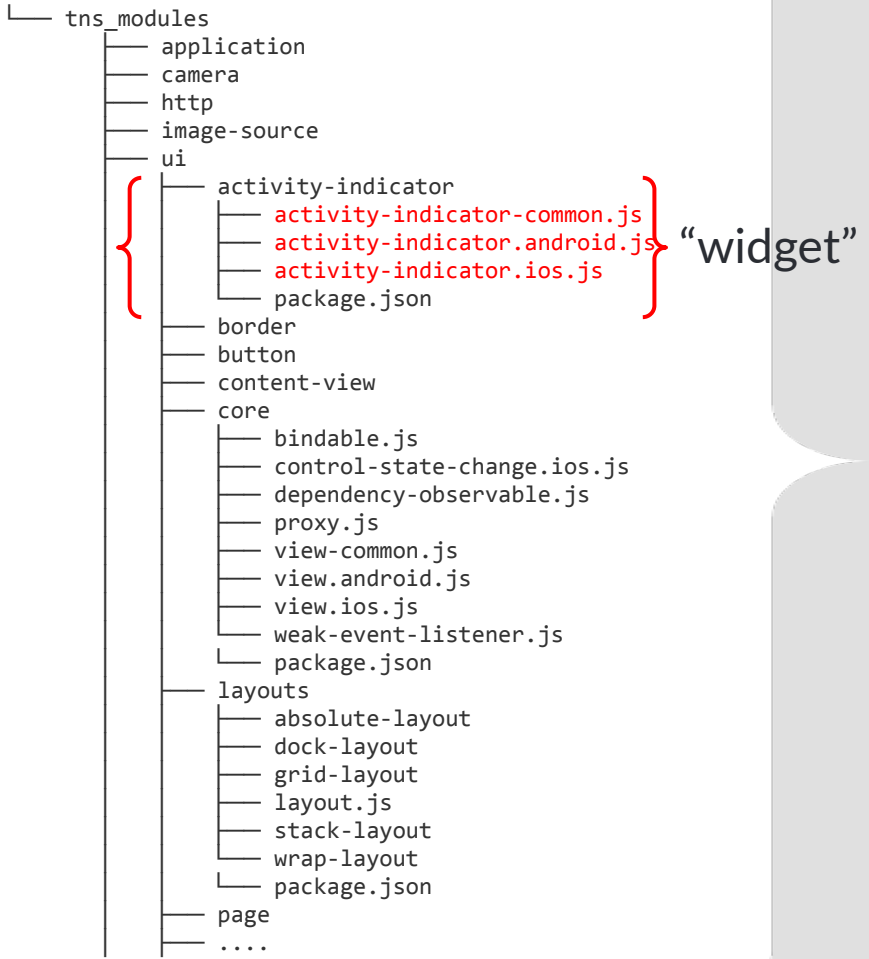


Where

Located in: tns_modules/ui

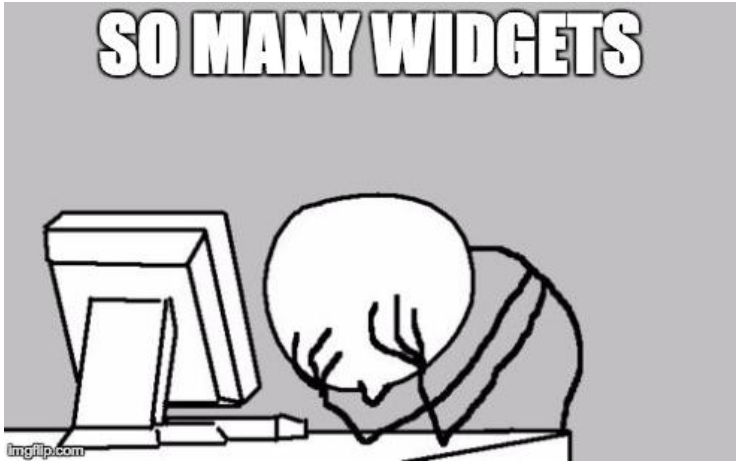
Typically follow:

- [name]-common.js
- [name].ios.js
- [name].android.js



What’s in our toolbox

```
└── tns_modules
    ├── color
    ├── image-source
    ├── text
    ├── ui
    │   ├── activity-indicator
    │   ├── border
    │   ├── builder
    │   ├── button
    │   ├── content-view
    │   ├── core
    │   ├── date-picker
    │   ├── dialogs
    │   ├── editable-text-base
    │   ├── enums
    │   ├── frame
    │   ├── gestures
    │   ├── image
    │   ├── image-cache
    │   ├── label
    │   ├── layouts
    │   ├── list-picker
    │   ├── list-view
    │   ├── page
    │   ├── placeholder
    │   ├── progress
    │   ├── scroll-view
    │   ├── search-bar
    │   ├── segmented-bar
    │   ├── slider
    │   ├── styling
    │   ├── switch
    │   ├── tab-view
    │   ├── text-base
    │   ├── text-field
    │   ├── text-view
    │   ├── time-picker
    │   ├── tool-bar
    │   └── web-view
```




MOAR ABSTRACTIONS?
imgflip.com

How to...

```
└── tns_modules
    ├── ui
    │   ├── label
    │   │   ├── label-common.js
    │   │   ├── label.android.js
    │   │   ├── label.ios.js
    │   │   └── package.json
```

view.xml

Just reference the *namespace*.

```
<Page xmlns="http://www.nativescript.org/tns.xsd">
  <StackLayout>
       <Label text="Label 1" />
       <Label text="Label 2" />
       <Label text="Label 3" />
  </StackLayout>
</Page>
```

view.js

Just reference the *module*.

```
var labelModule = require("ui/label");

var label = new labelModule.Label();
label.text = "{N}";
```

$(“#Label”)

No window, no problem.

1. Give the element an id.
2. Get It.
3. Use It.

```
<Page xmlns="http://www.nativescript.org/tns.xsd"
      loaded=”load” >

  <StackLayout>
       <Label id=”OhGetMePlease” text="{N}" />
       <Label text="Label 2" />
       <Label text="Label 3" />
  </StackLayout>
</Page>
```

```
var labelModule = require("ui/label");
var _page;

exports.load = function(args) {
  _page = args.object;
};

exports.OhTapMe = function(args){
  var myLabel = _page.getViewById("OhGetMePlease");
  myLabel.text = "{N} is awesome";
};
```

Going {N}ative

iOS: UINavigationController

```
if (applicationModule.ios) {

  var controller = frameModule.topmost().ios.controller;

  var navigationItem = controller.visibleViewController.navigationItem;
  navigationItem.setHidesBackButtonAnimated(true, false);

  var navBar = controller.navigationBar;
  navBar.barTintColor = UIColor.colorWithRedGreenBlueAlpha(.35, .90, .0, 1.0);
  navBar.barStyle = 0;
  navBar.tintColor = UIColor.blackColor();

  navBar.titleTextAttributes =
    new NSDictionary(
       [UIColor.blackColor()],
       [NSForegroundColorAttributeName]);
}
```

Module References

- Widgets

{{ Data Binding }}

{{ defined }}

Data binding is the process of connecting application user interface (UI) to a data object (business model).

With a correct data binding settings and if data object provides proper notifications then when data changes application UI will reflect changes accordingly (source to target).

Depending on settings and requirements there is a possibility to update data from UI to data object (target to source).

{{ black magic }}

noun

1. magic used for evil purposes; witchcraft; sorcery (contrasted with white magic).

Dictionary.com Unabridged
Based on the Random House Dictionary, © Random House, Inc. 2015.
Cite This Source

British Dictionary definitions for black magic

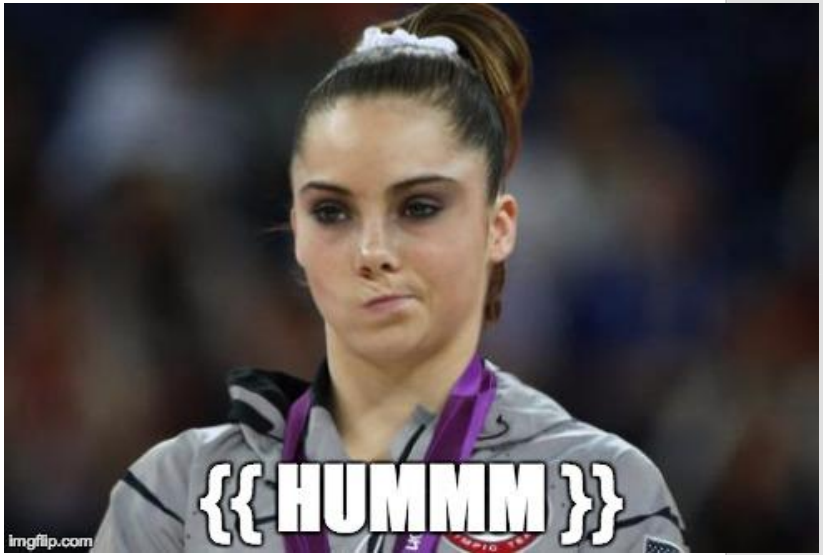
black magic in Technology

jargon
(Or " FM ") A technique that works, though nobody really understands why. More obscure than voodoo programming, which may be done by cookbook.
Compare black art, deep magic, and magic number.
(2001-04-30)

The Free On-line Dictionary of Computing, © Denis Howe 2010 http://foldoc.org
Cite This Source

{{ sayWAT }}

```
<Page xmlns="http://www.nativescript.org/tns.xsd"
      loaded=”load” >

  <StackLayout>
       <Label id=”OhGetMePlease” text="{{N}}" />
       <Label text="Label 2" />
       <Label text="Label 3" />
  </StackLayout>
</Page>
```



{{ bindingContext }}

```
<Page xmlns="http://www.nativescript.org/tns.xsd"
      loaded=”load” >

  <StackLayout>
       <Label id=”OhGetMePlease” text="{{N}}" />
       <Label text="Label 2" />
       <Label text="Label 3" />
  </StackLayout>
</Page>
```

```
var observableModule = require("data/observable");

var _viewData = new observableModule.Observable();
var _page;

exports.load = function(args) {
       _page = args.object;
       _viewData.set( "N", "awesome" );
       _page.bindingContext = _viewData;
};
```

{{ Express YOself }}

```
<Label text="{{ author ? 'by ' + author : '[no author]' }}" />
<Label text="{{ author || '[no author]' }}" />
```

{{ Event Binding }}

```
<Page xmlns="http://www.nativescript.org/tns.xsd"
      loaded=”load” >

<Button
       text="{{ N }}"
       tap="{{ myBtn }}" />
</Page>
```

```
var observableModule = require("data/observable");

var _viewData = new observableModule.Observable();
var _page;

exports.load = function(args) {
       _page = args.object;
       _viewData.set( "N", "awesome" );
       _page.bindingContext = _viewData;
};

exports.myBtn = function(args) {
       _viewData.set( "N", "CLICKED!!!" );
};
```

Data Binding References

- Data Binding
- UI Binding
- Observable
- ObservableArray

- JustMeme - {{ }}
- JustMeme - bindingContext

Style

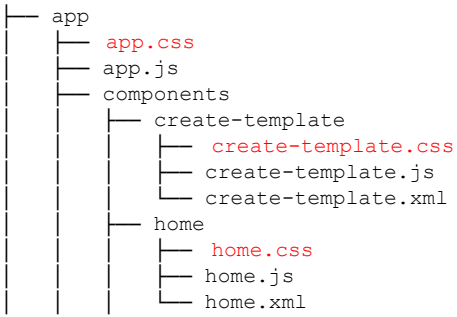
CSS - FTW!

You change the looks and appearance of views (elements) in a NativeScript application similarly to how you do it in a web application—using Cascading Style Sheets (CSS) or changing the style object of the elements in JavaScript.

Only a subset of the CSS language is supported.

Apply at 3 Different Levels

- **Application**
  - Applies to every application page
- **Page**
  - Applies to the page's UI views
- **InLine**
  - Applies directly to a UI view



Supported Selectors

Type

```
<Label />
```

```
Label { color: “white” }
```

Class

```
<Label cssClass=”ME” />
```

```
.ME { color: “white” }
```

ID

```
<Label id=”ME” />
```

```
#ME { color: “white” }
```

Style References

- UI Styling
- Supported Properties

- JustMeme - app's css
- JustMeme - home's css

What’s next?

- Screen Sizes
- Application Settings
- Custom Events
- Weak Events / Memory Leaks
- Dialogs / Action Sheets
- Gestures
- Placeholder

Just Remember
http://NativeScript.org