IN4MATX 133: User Interface Software

Lecture:
Mobile Design Principles
& SASS

Goals for this lecture

By the end of this lecture, you should be able to…

- Follow high-level guidelines for developing mobile interfaces
- Find and interpret platform-specific human interface guidelines
- Differentiate iOS and Android platform guidelines
- Explain why we might use a preprocessor like SASS for CSS
- Use SASS variables, mixins, nesting, and operators to simplify CSS

What makes a good user experience?

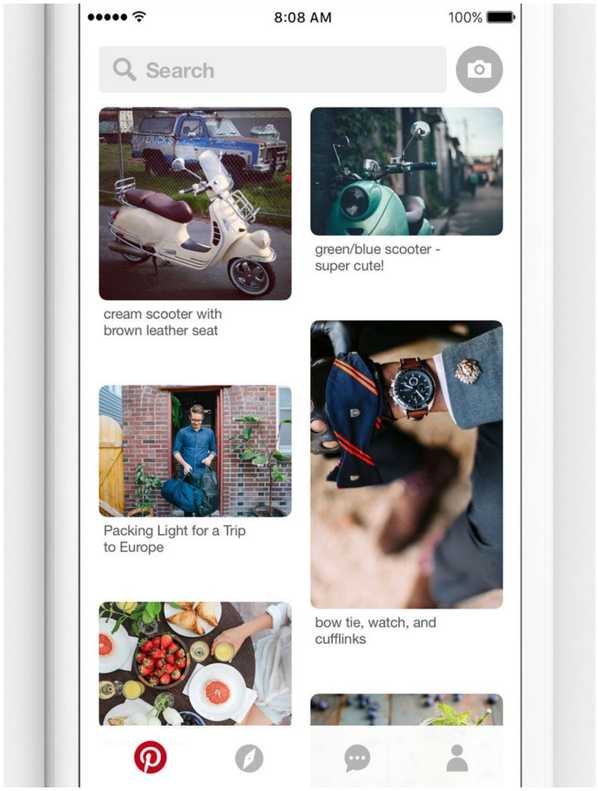
- It’s not just the UI
  - The experience begins with the first time you launch an app or go to a website
- There are several components here
  - Initial impression (boot up to app start)
  - User interface
  - Visual design
  - Information presentation
  - The physical device and how it is used with the app

A few principles of mobile design

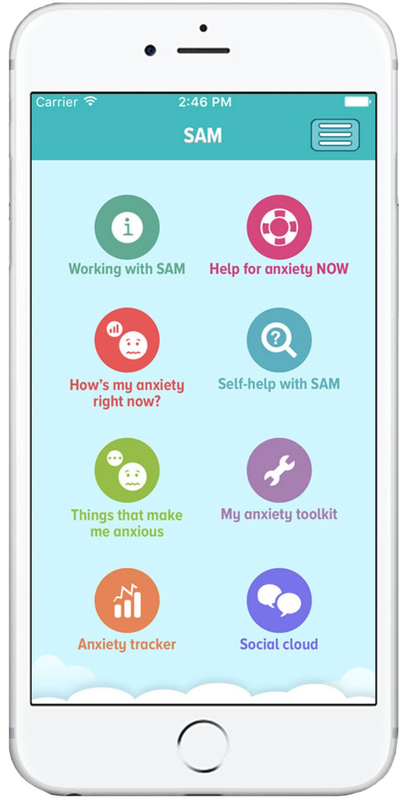
- A useful initial view
- The “uh-oh” button
- Error prevention
- Follow platform conventions

A useful initial view

- Give users clear calls to action
- Put useful content on the homepage
  - Pinterest's images
  - Put more than navigation buttons
- Make it easy to get back to the homepage
  - Bottom navbar, side navigation menu



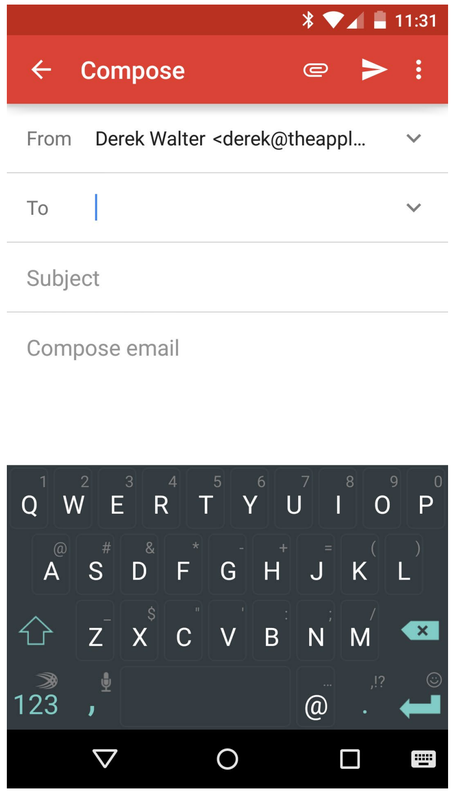
Pinterest



Anxiety management app

The “uh-oh” button

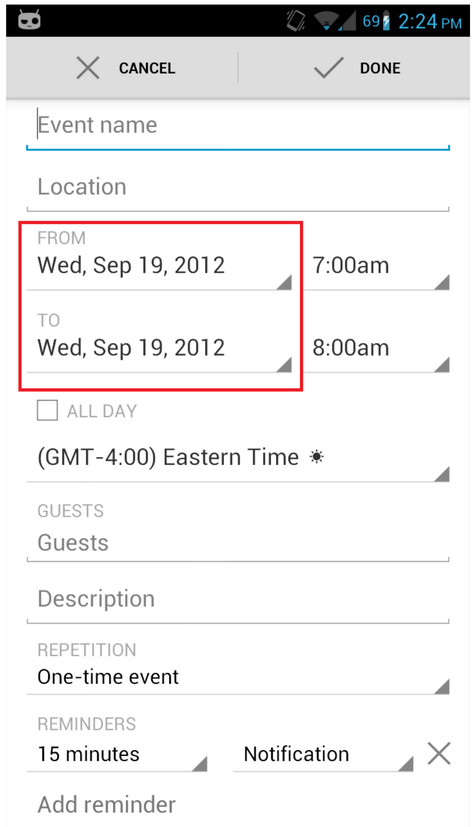
- Functions and buttons are often pressed by mistake
- Undo and redo should be easy
  - Gmail: “undo send”
- Navigating back a page should be easy
  - Breadcrumbs or back buttons (top left)

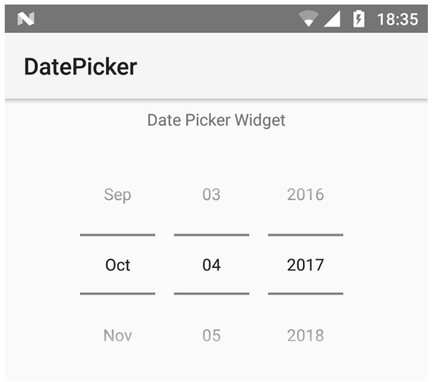


Gmail

Error prevention

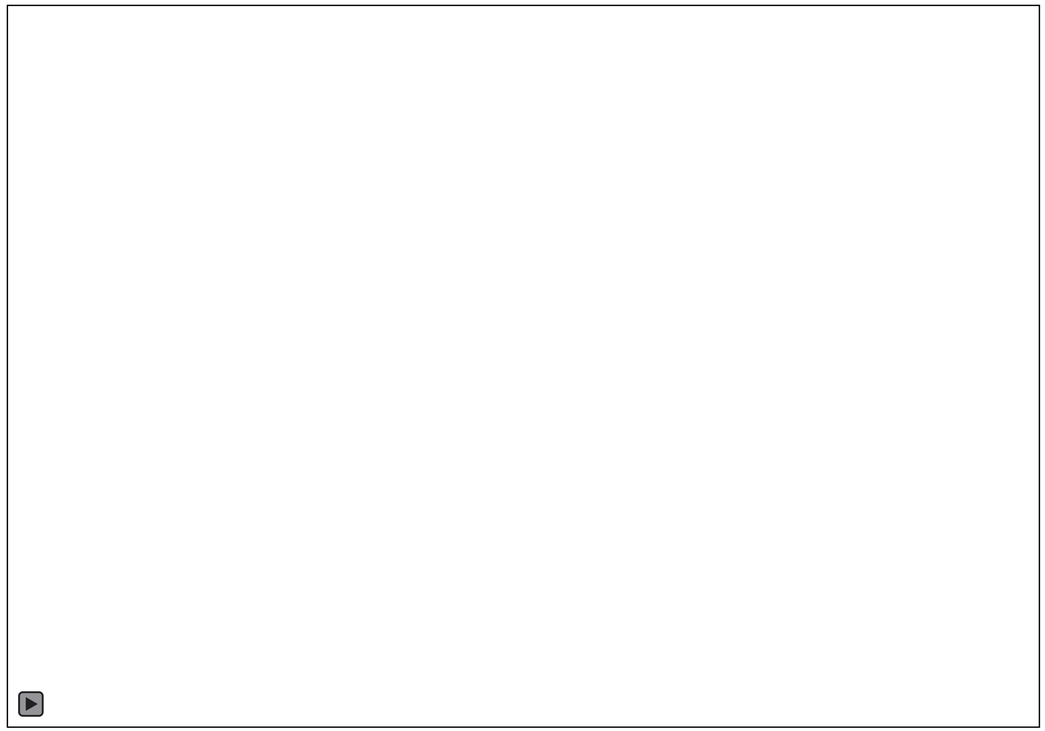
- Providing input with small devices is difficult
  - Add in as much assistance as possible to aid with input
- Add input checks
  - How many digits are in that phone number? Credit card number?
- Use appropriate widgets
  - Date/time spinner
  - Sliders





Follow platform conventions

- Users should not have to wonder whether different words, situations, icons, or actions mean the same thing
- Users should not have to remember app-specific navigation



iOS and Android platform conventions: Human Interface Guidelines

Human interface guidelines

- Created by web/mobile platform developers (Google, Apple)
- Key features:
  - Define rules for visual design and style
  - Specify interactions
  - Establish layout techniques
  - Provide consistency across the platform

Human interface guidelines

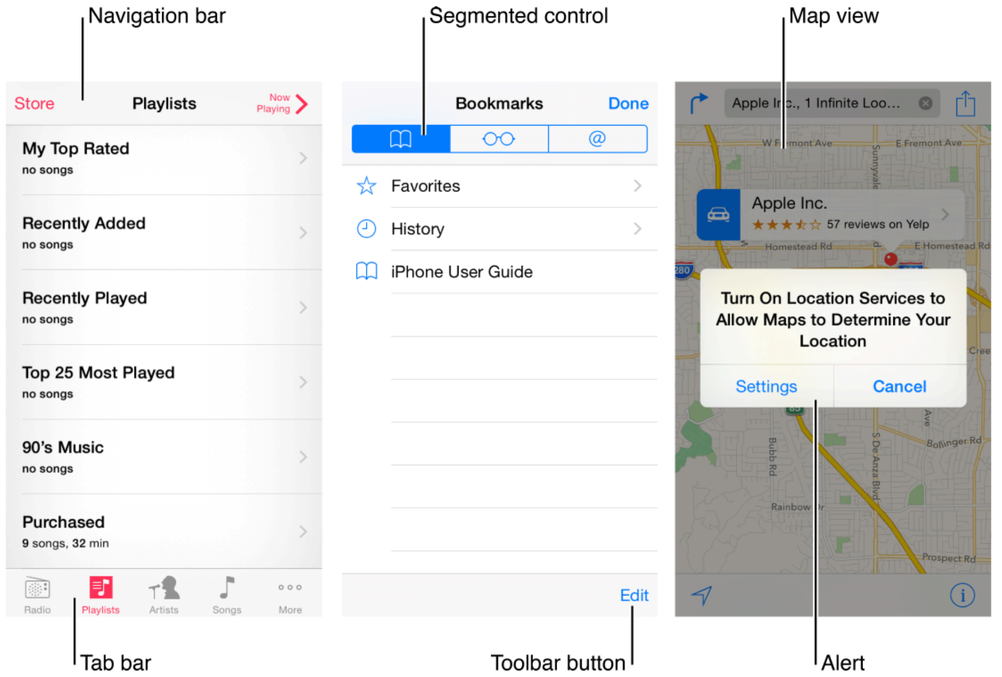
- HIGs are recommendations; you can choose to ignore them
  - The goal is to create an optimal experience for a device or platform
  - These guidelines most often follow best practices

iOS Human Interface Guidelines

- Content over UI
- Use the whole screen
- Single / simple colors
- Borderless buttons and widgets

Navigation

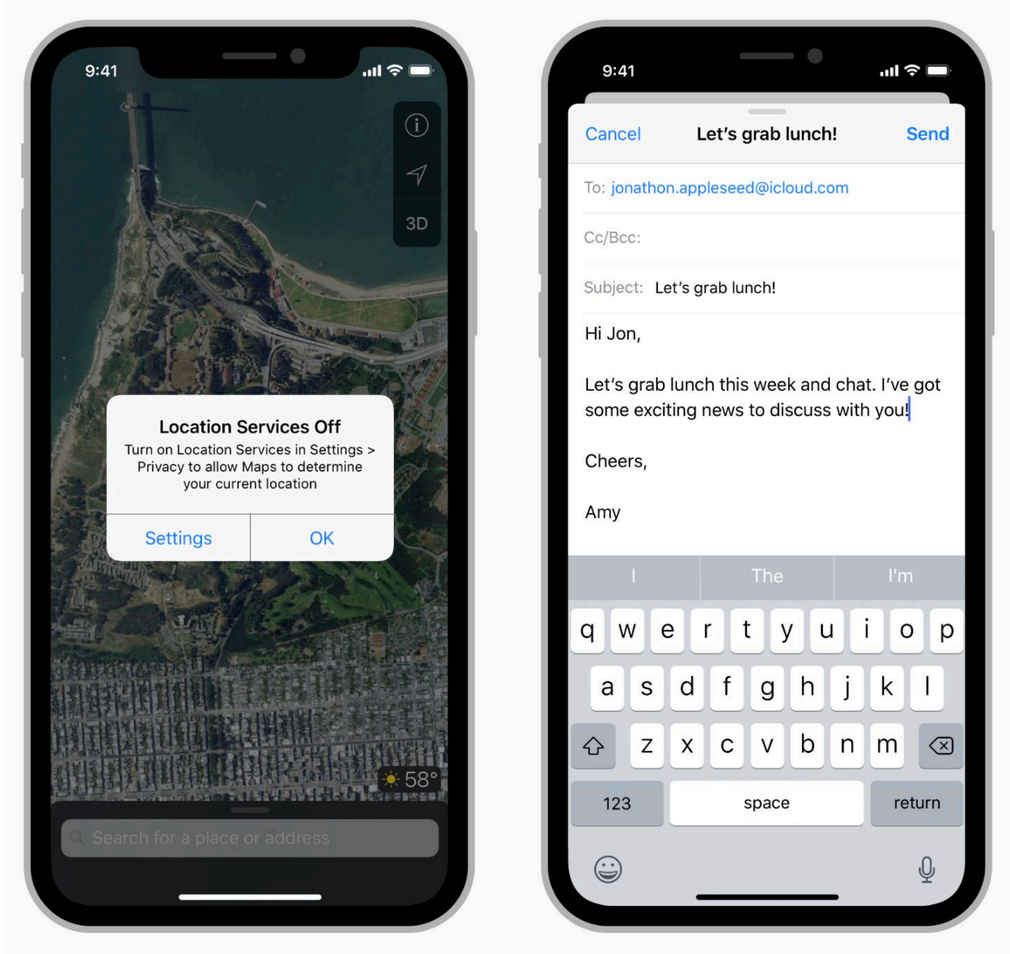
- Should be “natural”
- Use a navigation bar to traverse a hierarchy of data
- Use a tab bar for several peer categories
- Use a new page when that page is an instance of an item for another page



https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/navigation/

Modals

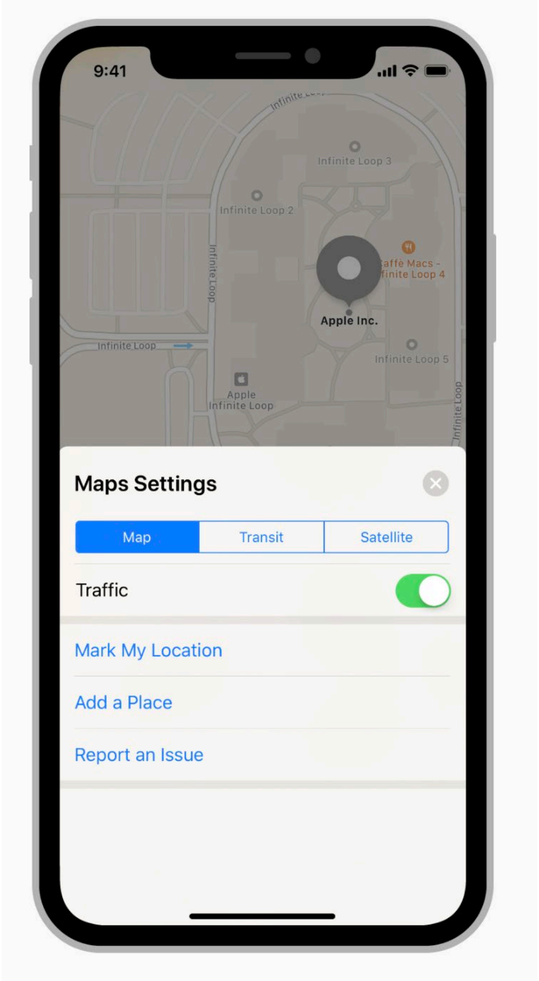
- Grab control of the experience until they are dismissed
- Meant to grab attention for doing one small, specific task
- Make sure the user can back out
- Respect notification wishes
- Use sparingly



https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/modality/

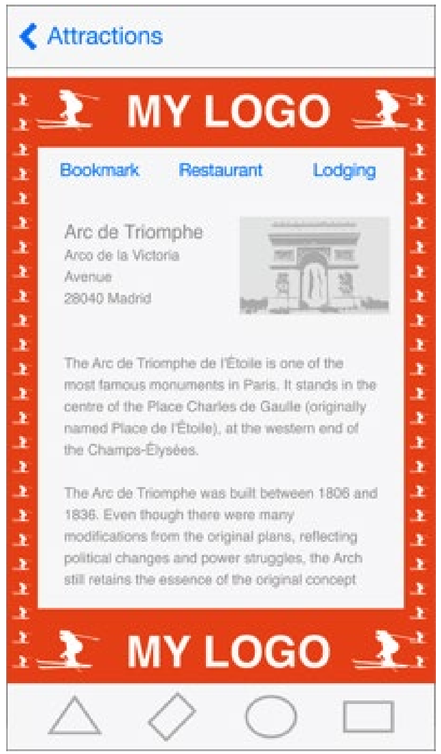
Interactivity

- Use a key color to denote interactive elements
- Denote “active” and “inactive” components differently
- Be aware of colorblindness



Branding

- It’s important to be distinctive…
- But be careful not to pull a user out of the iOS experience
- Your app does not have to look like a default app, but…



Color and Typography

- Colors are great for grabbing attention, but can be overused
- Use complementary colors
  - Palette definers like paletton.com
- Use a single typeface (font), if possible
  - Built-in fonts are just fine
  - Use font size, and color and weight (bold) to highlight information

https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/color/

Icons

- A good icon is important
- Keep background simple
- Only use words if they are essential or part of a logo
- Leave your icon out of the interface
- When appropriate, use system icons in the interface itself
  - Use as intended

Icon	Name	Meaning
	Action (Share)	Shows a modal view containing share extensions, action extensions, and tasks, such as Copy, Favorite, or Find, that are useful in the current context.
	Add	Creates a new item.
	Bookmarks	Shows app-specific bookmarks.
	Camera	Takes a photo or video, or shows the Photo Library.
Cancel	Cancel	Closes the current view or ends edit mode without saving changes.
	Compose	Opens a new view in edit mode.
Done	Done	Saves the state and closes the current view, or exits edit mode.
Edit	Edit	Enters edit mode in the current context.
	Fast Forward	Fast-forwards through media playback or slides.

https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/app-icon/

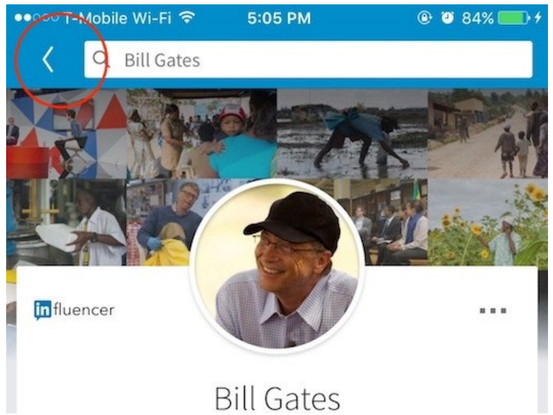
Google Material Design

- Philosophy: interface should look like layers on a sheet of paper
    - Have 3D depth and motion
- Follows many of the same patterns as iOS design in terms of interaction
    - Limited use of modals
    - Use color to emphasize content
    - Be subtle with branding
- But, there are a few key differences

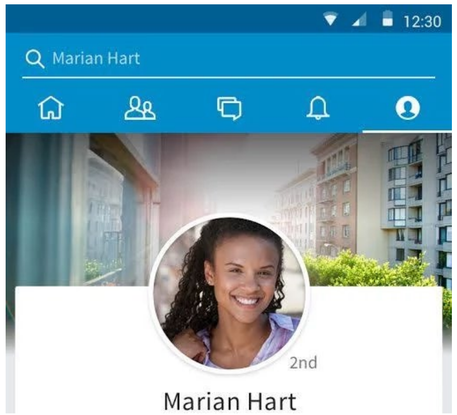
https://material.io/design/

Universal navigation bar

- Android "used to have” a navigation bar at the bottom of the screen
  - Sometimes it’s a hardware button, sometimes done in software?
  - But it’s always present
  - How is it handled now?
- iOS implements “back” in-app
  - Has it changed?



iOS

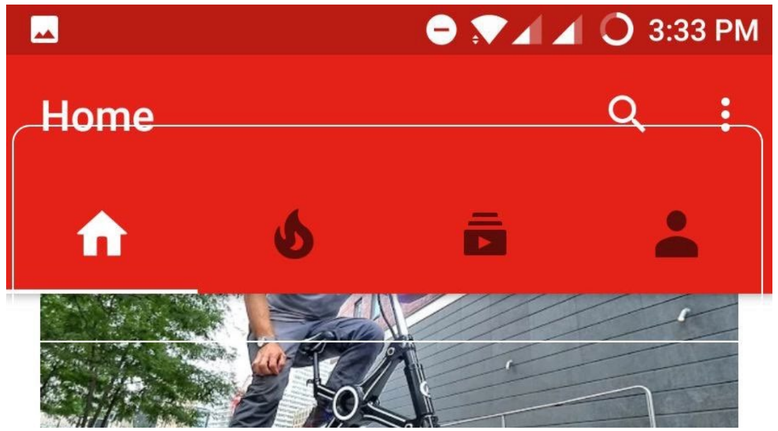


Android

https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7

In-app navigation

- On top in Android, on bottom in iOS
  - Why the difference?
  - Has it changed?
- Android only shows icons
- iOS icons have labels



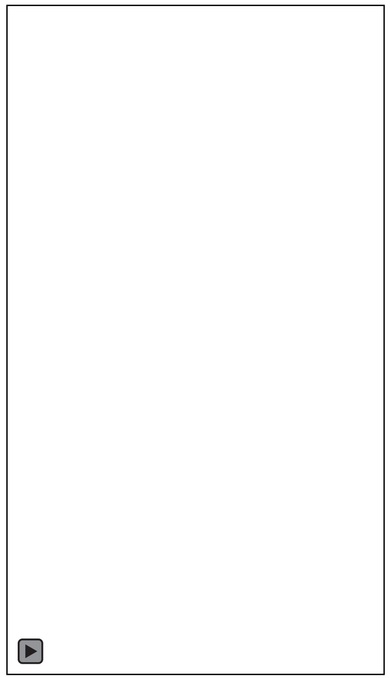
Android

iOS

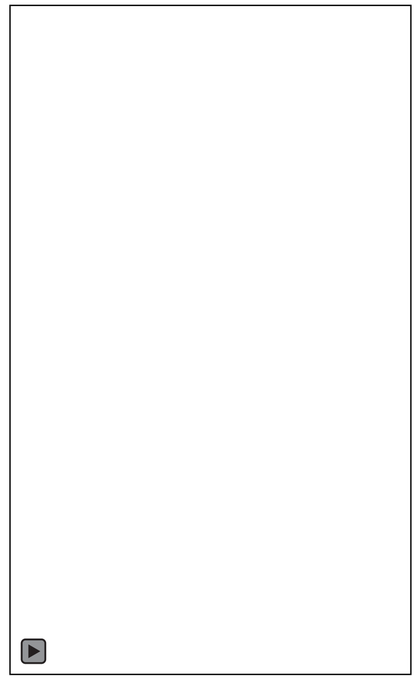
https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7

Swiping

- On Android, ~~swiping moves the user between tabs~~
- On iOS AND NOW Android, swiping takes the user back a screen
- Android's ~~always-present back button allows this navigation~~



Android

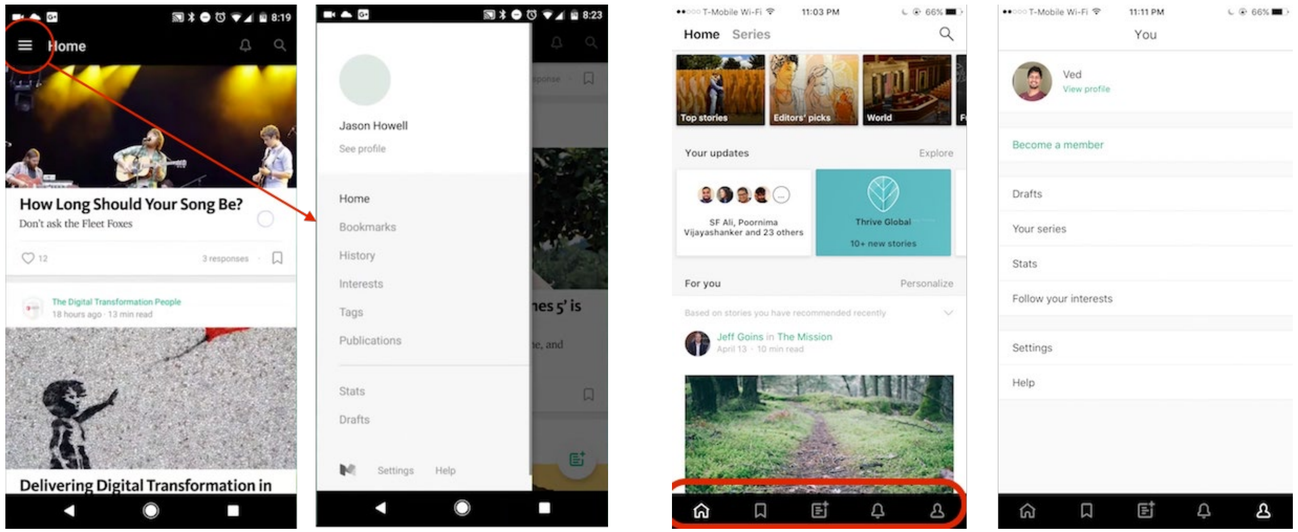


iOS

https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7

App settings

- Android apps usually load settings from a “hamburger” button in the top left
- iOS typically have settings as an item on the navigation bar



Android iOS

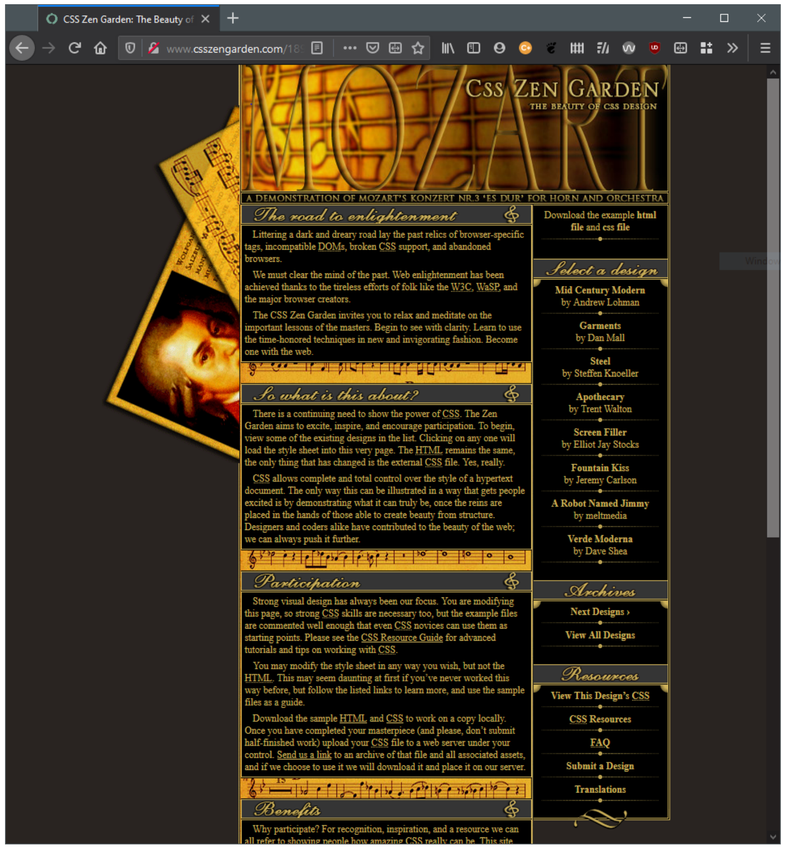
https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7

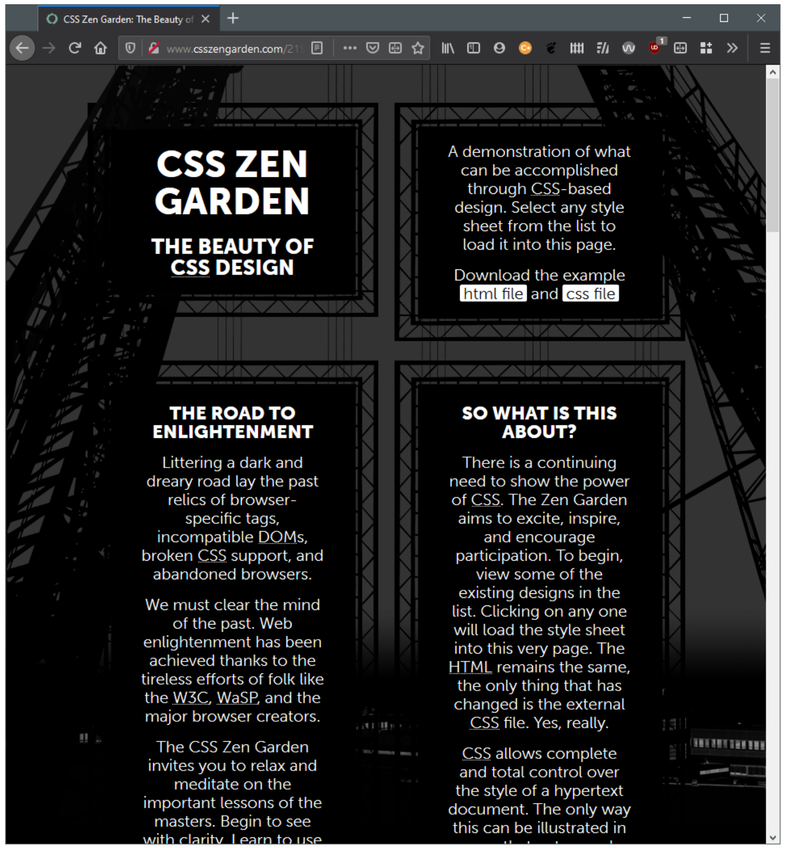
Uniformity

- There are always exceptions
- Not all apps vary the interaction
  and UI design patterns for each platform
- Understand the conventions for each platform, don’t stray too far.

Switching topics: SASS

Same page, different stylesheets





http://www.csszengarden.com/

CSS syntax

- **Selectors** specify which elements a **rule** applies to
- **Rules** specify what *values* to assign to different formatting **properties**

```
/* CSS Pseudocode */
selector {
  property: value;
  property: value;
  ...
}
```

One rule, many properties

Writing plain CSS

- Violates the “Don’t Repeat Yourself” principle of coding
- Many times we’re writing the same snippets of code for frequently used declarations

Example: fonts

- What if I want to switch from Lato to some other font?
- What if I want to make everything larger?
- I could have structured my CSS more efficiently, but at the core, it's inflexible
  - For example, I could have set Lato to be the default font for `cite`

```
cite > .series {
  font-family: 'Lato', sans-serif;
}

cite > a {
  font-family: 'Lato', sans-serif;
  font-size: 0.8em;
}

cite > .authors {
  font-family: 'ChaparralPro', serif;
  font-size: 1.1em;
}

cite > .title {
  font-family: 'Lato', sans-serif;
  font-weight: 700;
}
```

CSS preprocessors

- “Let you abstract key design elements, use logic, and write less code”
- Three widely used ones: SASS, Less, Stylus
- They all do pretty much the same thing
  - Angular and Ionic let you choose a preprocessor or use plain CSS

CSS preprocessors

Major features

- Variables
- Mixins
- Nesting
- Extensions
- Operators

Variables

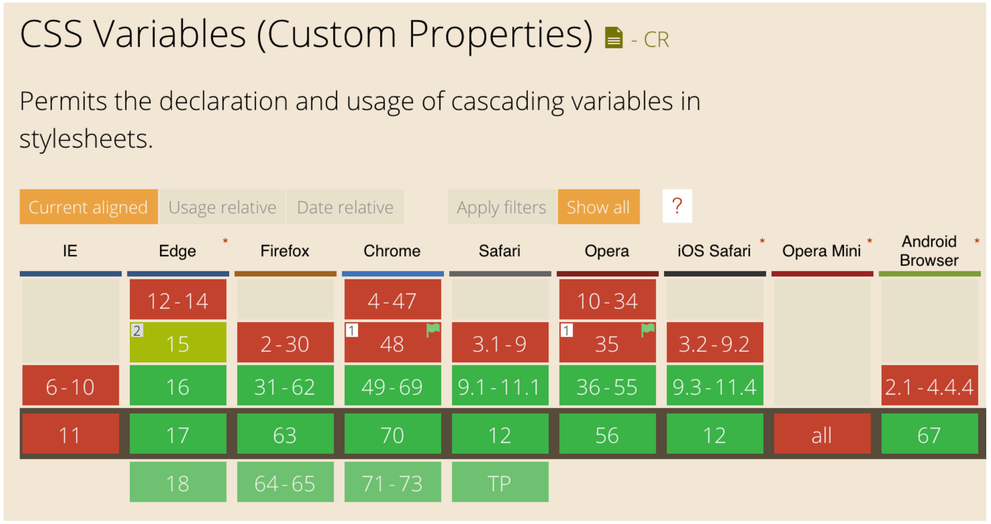
- Using variables with CSS preprocessors makes it easy to update colors, fonts, or other values throughout your entire stylesheet

```
//Sass Variables

$primary: #CC5533;

$font-base: 12px;

//Less Variables

@primary: #CC5533;

@font-base: 12px;

//Stylus Variables

primary = #CC5533

font-base = 12px
```

Variables

- As of 2016, variables are supported in plain old CSS
  - Many frameworks use these variables instead of preprocessor variables
- But preprocessors offer a lot more functionality

```
/*Declaring a variable*/
element {
  --main-bg-color: brown;
}

/*Using the variable*/
element {
  background-color: var(--main-bg-color);
}
```



https://www.caniuse.com/#search=css%20variables

Nesting

- You can nest selectors with preprocessors
- This means you can easily organize an entire hierarchy of selectors, including child elements

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li { display: inline-block; }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
   &:hover {
    text-decoration: underline;
   }
  }
}
```

Extensions

- The @extend property lets you share styles from one selector to another
- Use % to define an “abstract” style
- Can be inherited: can extend one style which extends another style

```
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}
.message {
  @extend %message-shared;
}
.success {
  @extend %message-shared;
  border-color: green;
}
.error {
  @extend %message-shared;
  border-color: red;
}
```

Mixins

- Mixins can produce functions which apply a set of styles when given arguments
- Similar to extensions, except arguments can be passed and no concept of inheritance

```
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}

.box { @include border-radius(10px); }
```

Operators

- Like most programming languages, CSS preprocessors can do math!
- This is especially great for setting a fixed value in a variable, like a font-size or padding, and then modifying it as you go along

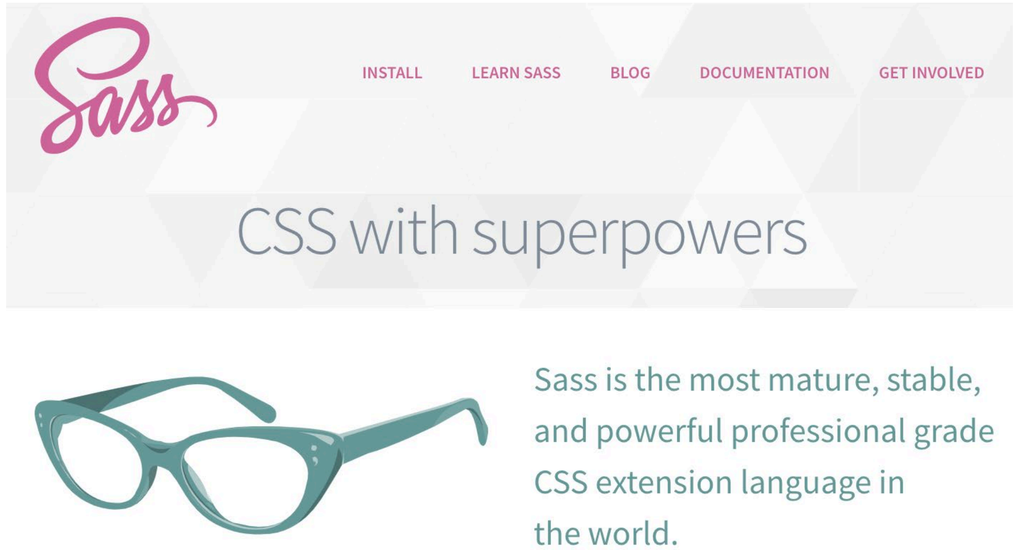
```
$container = 100%;

article[role="main"] {

  float: left;

  width: 600px / 960px * $container;

}
```

Digging into SASS

SASS

Syntatically Awesome Style Sheets

- “SASS is the most mature, stable, and powerful professional grade CSS extension language in the world.”
- “SASS boasts more features and abilities than any other CSS extension language out there”
- It’s on their website, so it must be true!



http://sass-lang.com/

SASS

- File extension: `.scss`
- SASS is a superset of CSS
  - You can write any CSS in a SCSS document
- SCSS is transpiled to CSS
  - Just like TypeScript is transpiled to JavaScript

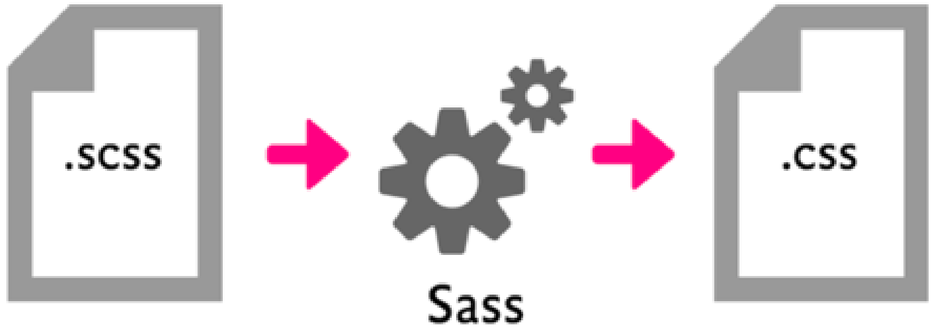


FIG 1: Sass converts its own "power syntax" to plain old CSS.

SCSS Syntax

- Looks very much like regular CSS
- Rules apply to a selector and are made in brackets
- Each rule ends with a semicolon
- SCSS adds variables, mixins, etc.

```
$font-stack:    Helvetica, sans-serif;

$primary-color: #333;



body {

  font: 100% $font-stack;

  color: $primary-color;

}
```

SCSS Syntax

.scss

```scss
$primary-color: #3bbfce;

$margin: 16px;

.content-navigation {

  border-color: $primary-color;

  color: darken($primary-color, 10%);

}

.border {

  padding: $margin / 2;

  margin: $margin / 2;

  border-color: $primary-color;

}
```

.sass

```sass
$primary-color: #3bbfce

$margin: 16px

.content-navigation

  border-color: $primary-color

  color: darken($primary-color, 10%)


.border

  padding: $margin/2

  margin:  $margin/2

  border-color: $primary-color
```

- There’s another style, called “sass syntax”, which looks more like python
- It’s older, uses the .sass extension
- Why is .scss better?
  - It’s a superset of CSS, rather than another syntax

Thoughts on CSS preprocessors

- Preprocessor functionality is slowly getting added to the CSS standard
  - CSS now supports variables, for example
- Does this mean that preprocessors will soon be obsolete?
  - Maybe. Or maybe they'll evolve, adding other kinds of new and better features
  - Transpiling languages are a great way to show the value of new features
  - And if they catch on enough, they get added into the standard
  - Who knows, maybe JavaScript will add typing from TypeScript

Goals for this lecture

By the end of this lecture, you should be able to…

- Follow high-level guidelines for developing mobile interfaces
- Find and interpret platform-specific human interface guidelines
- Differentiate iOS and Android platform guidelines
- Explain why we might use a preprocessor like SASS for CSS
- Use SASS variables, mixins, nesting, and operators to simplify CSS

How do I actually use a CSS preprocessor like SASS?

Installing SASS

- `npm install -g sass`
  - This version is written in JavaScript, which is slower
  - But it’s fine for the size of projects we’re working with
- `choco install sass`
- `brew install sass/sass/sass`
  - 3 times to make super duper sure
  - (just kidding, it’s how HomeBrew designates projects)

https://sass-lang.com/install

Manually transpiling

- You can use SASS with a plain-old HTML page!
- It just needs to be transpiled to CSS before getting loaded

Manually transpiling

- Transpile one file:
- `sass input.scss output.css`
- Watch one file for changes:
  - `sass —watch input.scss:output.scss`
- Watch a whole directory of SASS files:
  - `sass —watch path/sass-directory`

Automatic transpiling

- A lot of frameworks will automatically transpile `.scss` files when they build and run
- Angular and Ionic can include `.scss` files for every component and secretly transpile them to `.css`
    - A preprocessor is specified when the app is first created

Goals for this lecture

By the end of this lecture, you should be able to…

- Follow high-level guidelines for developing mobile interfaces
- Find and interpret platform-specific human interface guidelines
- Differentiate iOS and Android platform guidelines
- Explain why we might use a preprocessor like SASS for CSS
- Use SASS variables, mixins, nesting, and operators to simplify CSS