COEN 168/268

Mobile Web Application Development

MVC in JavaScript and Ember.js Introduction

Peter Bergström (pbergstrom@scu.edu)

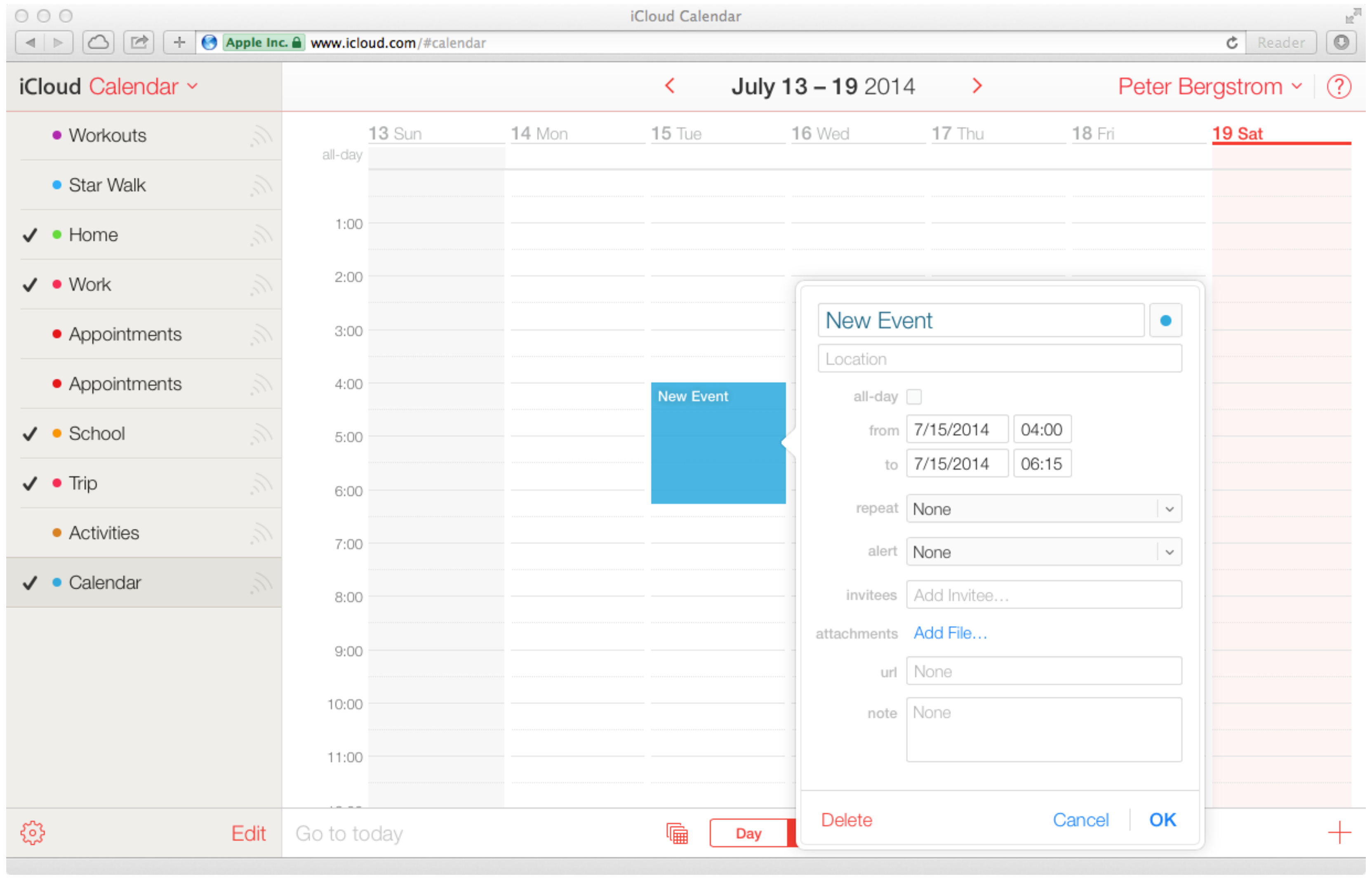
Santa Clara University

Modern Web Browsers

- Faster rendering:
  - DOM manipulation, CSS rendering, GPU acceleration, etc
- Faster JavaScript:
  - More efficient interpreters, Just-In-Time (JIT) compilation
- Rich HTML 5 APIs
  - Videos, audio, drag & drop, location services, local storage, etc

Modern Web Browsers Allow For Building Native-Style Apps

For Example, *iCloud.com Calendar*



New Event
Location
all-day ☐
from 7/15/2014 04:00
to 7/15/2014 06:15
repeat None
alert None
invitees Add Invitee…
attachments Add File…
url None
note None

I use this example because I worked on it when I was at Apple

These Web Apps Are Complex

As apps become more complex:

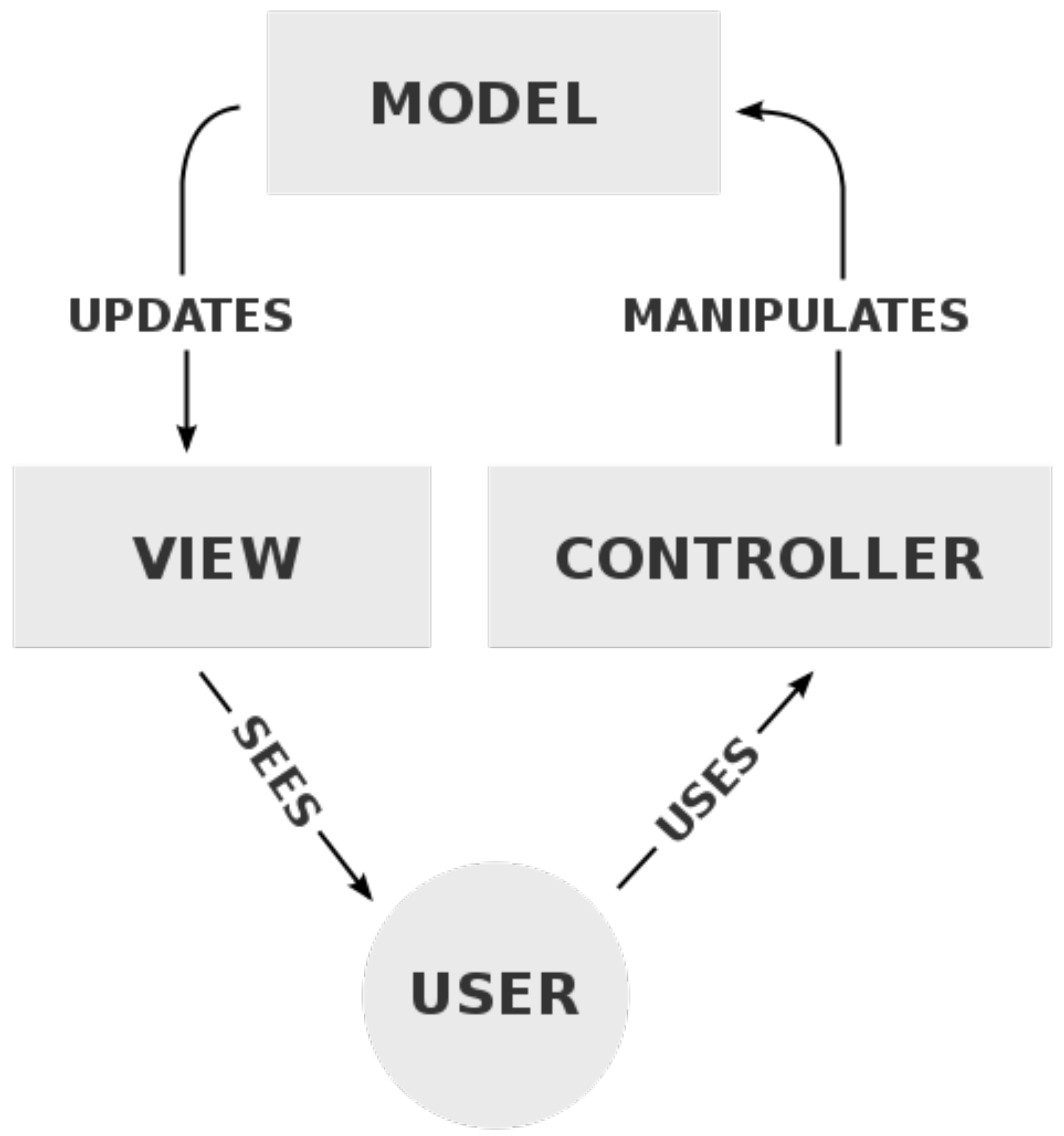
- Number of lines of code increases
- Number of files increase
- Number of developers increase
- Number of designers, PMs, QA increase

This Might Get Out of Control

- Using basic JavaScript or even jQuery is not viable for complex apps
- You need to make sure that your code is:
    - Modular
    - Maintainable
    - Reusable
    - Extendable

Solve This Problem By Using Design Patterns!

Model-View-Controller Design Pattern



`Model-View-Controller` Design Pattern

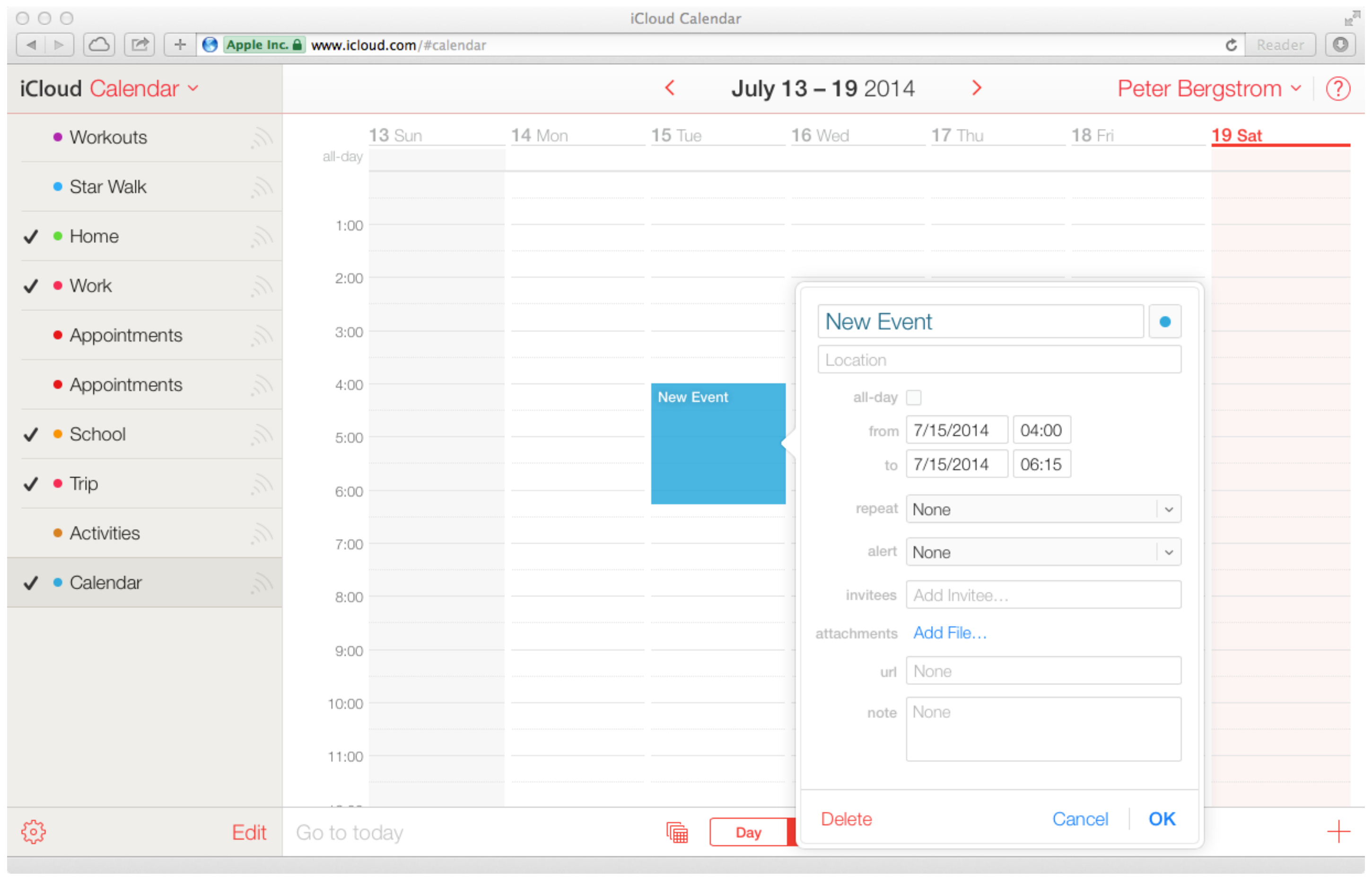
- **Controller**: the user *uses* the controller to *change* the model
- **Model**: the controller *manipulates* the model and the model *updates* the view
- **View**: the view *requests* data from model to *generate* the UI displayed to the user

There Are Many JavaScript MVC Frameworks

- Some are lightweight
- Some are heavy
- Some are restrictive
- Some are open ended
- Some are opinionated

However, most build upon on lower level frameworks such as `jQuery`

Back to *iCloud.com Calendar*



New Event
Location
all-day
from 7/15/2014 04:00
to 7/15/2014 06:15
repeat None
alert None
invitees Add Invitee…
attachments Add File…
url None
note None

Uses the `SproutCore` JavaScript MVC framework that I used and worked on back in the day when I was at Apple

iCloud.com Calendar

Demo

`SproutCore` Is A **Heavy** MVC Framework

- Not made for mobile
- Contains the kitchen sink in terms of functionality
- Good for large apps, but too complicated for small apps
- It's old now

However, there are other frameworks!

There Are Many Modern MVC Frameworks…

Backbone.js, AngularJS, Ember.js, KnockoutJS, Dojo, YUI, Agility.js, Knockback.js, CanJS, Maria, Polymer, React, cujoJS, Montage, Sammy.js, Stapes, Epitome, soma.js, DUEL, Kendo UI, PureMVC, Olives, PlastronJS, Dijon, rAppid.js, DeftJS + ExtJS, Aria Templates, Enyo + Backbone.js, SAPUI5, Exoskeleton, Atma.js, Ractive.js, ComponentJS, Vue.js, React + Backbone.js

- Taken from TodoMVC.com

There Are so Many That We Can't Talk about All of Them

Let's Look At TodoMVC.com

TodoMVC

Helping you **select** an MV* framework

Which JavaScript MVC Framework Should You Use?

DANGER
FLAMEWAR
AHEAD

Why not AngularJS?
Why not Backbone.js?
Why not KnockoutJS?
Why Not Agility.js?
Why not ExtJS?
Why not Kockback.js?
Why not CanJS?
Why not SproutCore?
Why not Polymer?
Why not Sammy.js?
Why not PureMVC?
Why not...

Which Should **We* Choose?

- There are so many good frameworks out there
- For this course, we don't have time to cover more than one
- Picking one shouldn't be the trigger of a flame war (please)

ember

Why `Ember.js`?

- As good as any framework out there
- What you will learn about `Ember.js` will translate to other JavaScript frameworks
- It's one of the most powerful (yet opinionated) frameworks out there so you will learn a lot
- I know it well
- Because.

ember

`Ember.js` is

A framework for creating **ambitious** *web applications*

Some `Ember.js` Background

- Open source framework using the MVC architectural pattern created by Yehuda Katz (of Ruby on Rails & jQuery) and Tom Dale in 2011
- `Ember.js` originated from the `SproutCore 2.0` project at Strobe, Inc
- Strobe Inc was a startup founded by Charles Jolley, the creator of `SproutCore` where myself, Yehuda, Tom, and others worked
- `SproutCore` was developed at Apple when Charles, Tom, myself and others worked there on .Mac, MobileMe, & iCloud

I Write Apps, I Don't Write Frameworks

- Most of the apps that I wrote while at Apple and Strobe were written in parallel with the development of `SproutCore`, `SproutCore 2.0` and `Ember.js`
- I view my contribution to `Ember.js` and `SproutCore` as an architect and developer of apps that **challenged** the framework creators to come up with good solutions to common problems
- I have added some code here and there, but the VAST majority of the frameworks have been **written by others who all deserve the credit**

Ember.js` is a MVC Framework

- Uses the Model-View-Controller architectural pattern
- Uses proven patterns and common idioms from native development frameworks, such as iOS
- As a heritage in `SproutCore` which was influenced by Cocoa development
- As a result, a lot of the concepts translate to native app development, which is useful

`Ember.js` Core Concepts

Templates

- Uses the **Handlebars** template language which extends the `Mustache` template syntax
- Uses **plain HTML** and expressions such as `{{firstName}}`
- Describes the user interface in your application using good semantics
- Each template is backed by a model object
- When the model object changes, the template updates automatically

Templates

An example Handlebars template might look like this:

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

Router

- Monitors the browsers `window.location.hash`
- Translates URLs into templates (or nested templates)
- `Ember.js` makes it easy to restore state just from the URL
- Can be used as a light-weight state management system

Components

- Custom HTML backed by JavaScript
- Easy to access in Handlebars templates
- Allows you to create reusable components that have complex behaviors

Models

- A model is a JavaScript object that stores persistent state
- Are displayed via templates
- Models are usually loaded via AJAX from a back end service
- `Ember.js` does not care what kind of back end service it is or what it is written in

Controllers

- Stores application state
- Templates can have a controller in addition to a model if you want to do more complex operations
- Templates can access controller properties
- Controllers can respond to actions defined in the HTML

We will go through each of the core concepts in the next couple of lectures following somewhat the format of the `Ember.js` guides

http://emberjs.com/guides/

Now on to Setting Up `Ember.js`

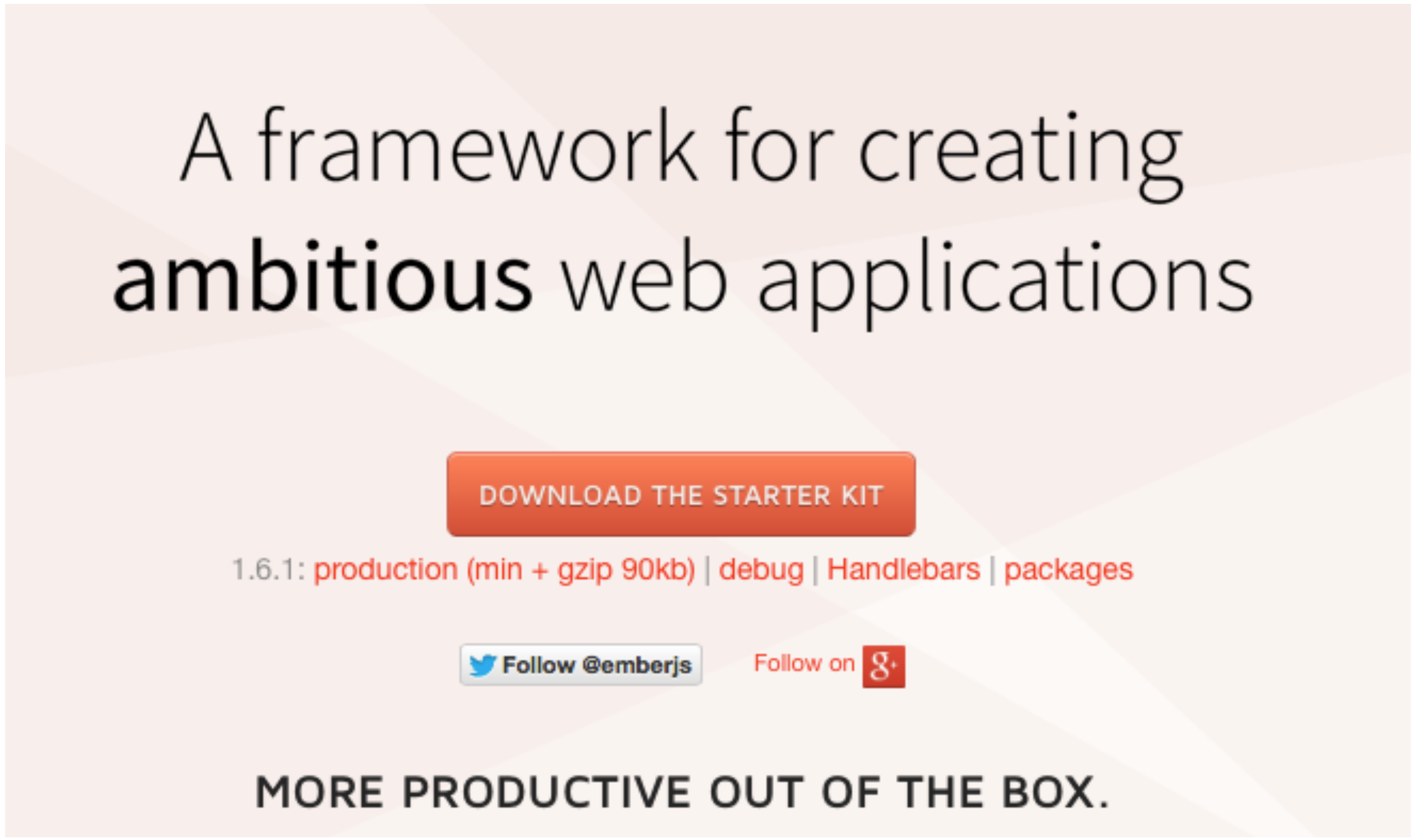
`Ember.js` Core Dependencies

As of `Ember.js` 1.9.1:

1. `jQuery` (v. 1.10.2)
    - http://jquery.com
2. `Handlebars` (v. 1.1.12)
    - http://handlebarsjs.com
3. Developers like you

Getting The `Ember.js` Starter Kit

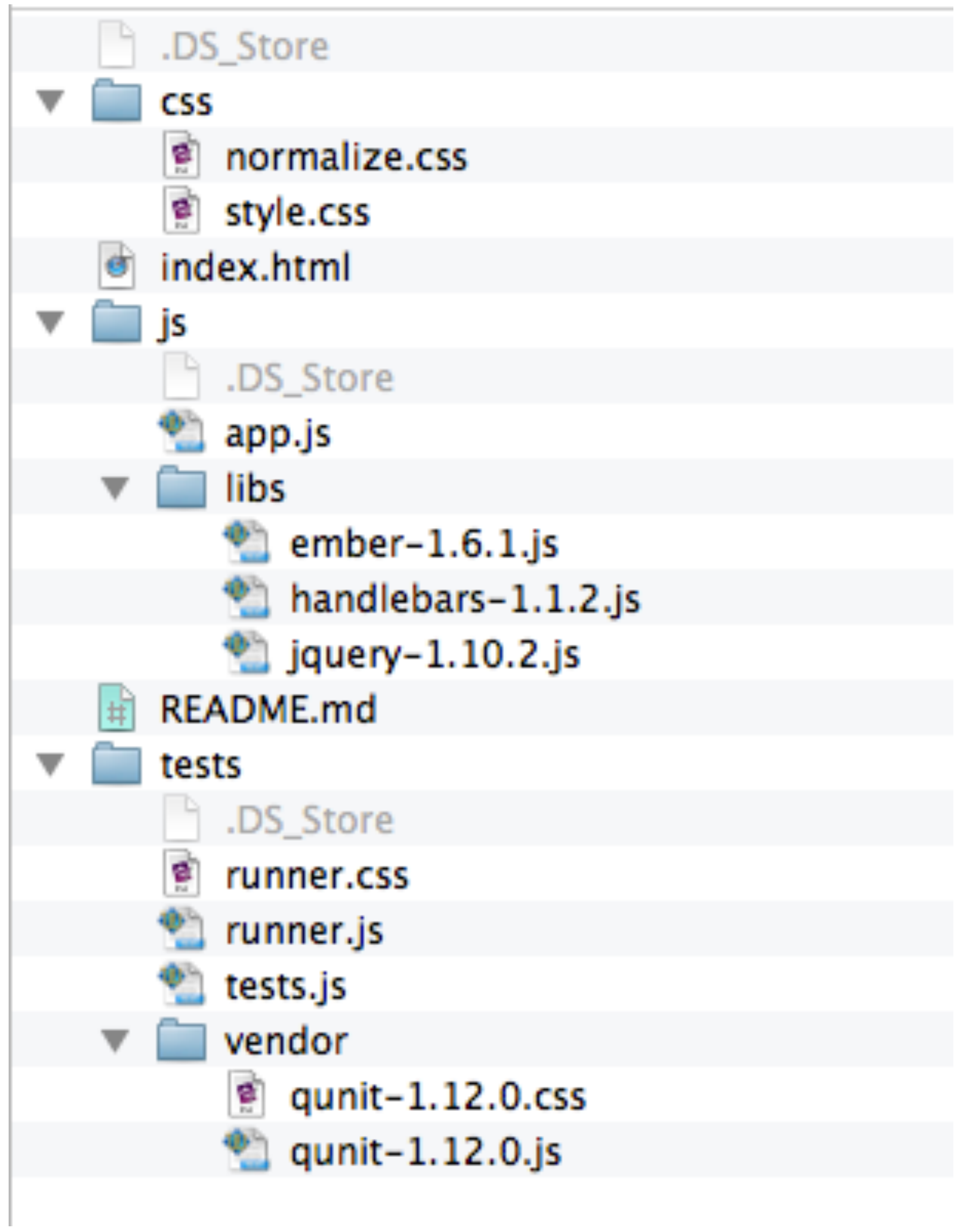
Download at `http://www.emberjs.com`


A framework for creating **ambitious** web applications

DOWNLOAD THE STARTER KIT
1.6.1: production (min + gzip 90kb) | debug | Handlebars | packages
Follow @emberjs
Follow on g+

MORE PRODUCTIVE OUT OF THE BOX.

What Does The Starter Kit Include?



```
.DS_Store
▼ css
    normalize.css
    style.css
index.html
▼ js
    .DS_Store
    app.js
  ▼ libs
      ember-1.6.1.js
      handlebars-1.1.2.js
      jquery-1.10.2.js
README.md
▼ tests
    .DS_Store
    runner.css
    runner.js
    tests.js
  ▼ vendor
      qunit-1.12.0.css
      qunit-1.12.0.js
```

Ember.js Starter Kit README Notes

- Start writing your app in `js/app.js`.
- Describe your application HTML in `index.html`.
- During development, you can link to `js/libs/ember-*.js` to get the unminified version of Ember.js.
- Add CSS to `css/style.css`.
- Open `index.html` in your browser.

Let's try it the Starter Kit

Demo

So, As Your App Grows You Need Structure

- The Starter Kit gives you a basic structure
- However, it won't scale very well as your app gets more advanced
- Therefore, we want to have a better directory structure that separates files into the M-V-C components and more on an object basis
- This means a more complex structure to add convenience

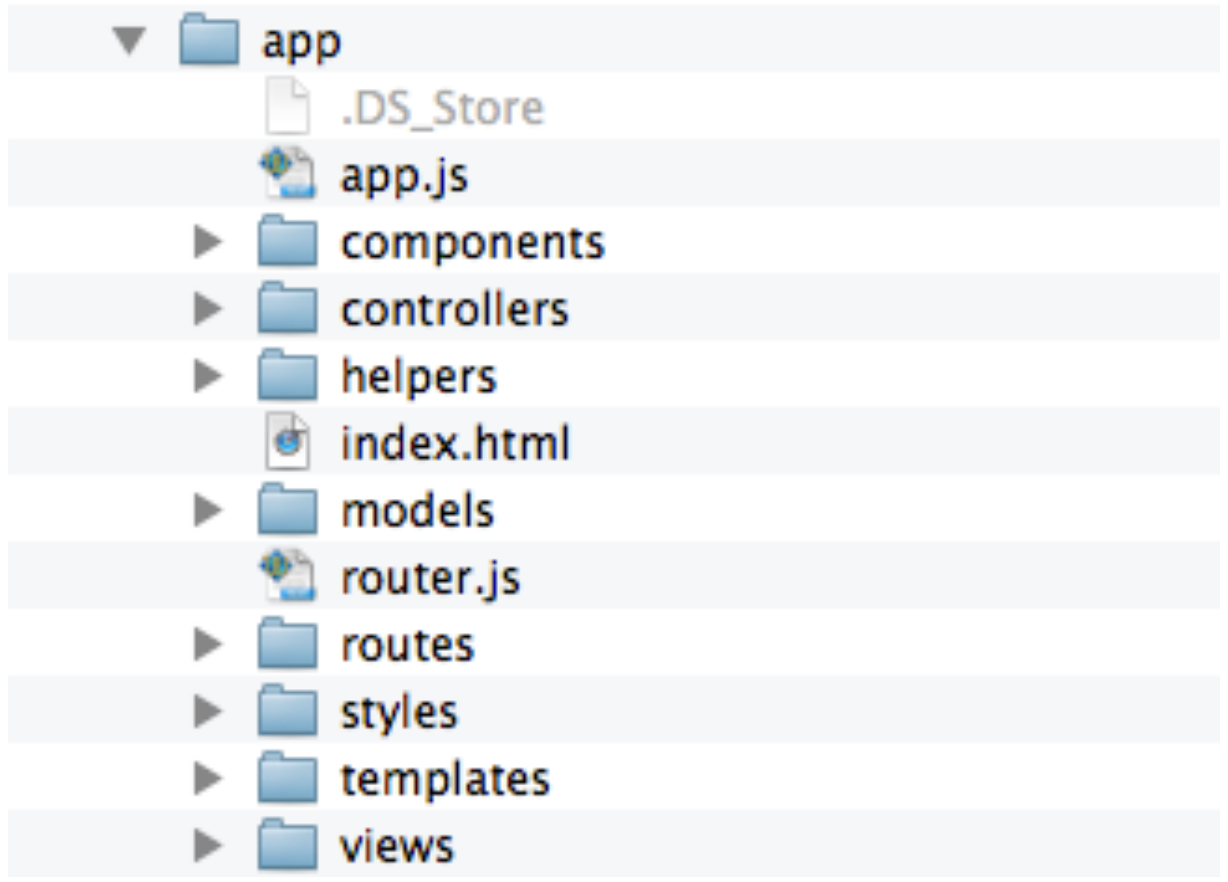
As Your App Grows You Have To Worry About:

Source control, minification of files, concatenation of files, linting, deployment, API stubbing, API proxying, testing, continuous integration, SASS/LESS management, package management, framework versioning, browser testing, and much more…

While still being stable, fast, and extensible

- Inspired by Stef Penner in his 2014 EmberConf talk: https://www.youtube.com/watch?v=4D8z3972h64

First Challenge: Splitting Your JS Out Into Functional Areas



```
app
    .DS_Store
    app.js
    components
    controllers
    helpers
    index.html
    models
    router.js
    routes
    styles
    templates
    views
```

This is one way to do it with the `Ember App Kit` structure...

A Sample `Ember.js` Directory Structure

- `app.js` -> The `Ember.Application` definition for your app
- `components/` -> any Ember.js component objects
- `controllers/` -> Contains any view controller objects
- `helpers/` -> Contains any `Handlebar` view helpers
- `index.html` -> The main `index` file for your app
- `models/` -> Contains any model objects

A Sample `Ember.js` Directory Structure, Cont'd

- `router.js` -> The `Router` object
- `routes/` -> Contains any `route` definitions
- `styles/` -> Contains any CSS or LESS files
- `templates/` -> Contains any `Handlebar` templates (instead of in `index.html`)
- `views/` -> Contains your view objects

So, How Do You Manage All These Files?

- You can do it yourself but you'll add a lot of disastrous and error prone glue code
- For performance, you need to actually load less files but with all the components
- Therefore, you need some tool to help you out to make your life easier

Some Challenges

1. Need to take in account dependencies
2. Need to find all files in the project
3. Need to update when things change
4. Need to help you do this without adding MORE overhead

Bring In `ember-cli`!

- `Ember.js command line utility
- Based on the `Ember App Kit` project structure template
- Offers:
  - Asset Compilation
  - Modules using the ES6 module transpiler
  - Testing using CLI
  - Dependency Management

Getting Started with `ember-cli`

Prerequisites

- Node.js: Get from http://nodejs.org
- PhantomJS: `npm install -g phantomjs`
- Ember CLI: `npm install -g ember-cli`
- Bower: `npm install -g bower`

Creating and Running New Project

In the command line: `ember new my-new-app`

This will create a new `my-new-app` folder and generate an application structure for you. Then, you can:

```
cd my-new-app
ember server
```

Then access the app through the browser: `http://localhost:4200`

Let's Go Through The Important `ember-cli` Commands

ember new <app-name> <options...>

```
ember new <app-name> <options...>
  Creates a new folder and runs ember init in it.
  --dry-run (Boolean) (Default: false)
    aliases: -d
  --verbose (Boolean) (Default: false)
    aliases: -v
  --blueprint (gitUrl, Path) (Default: app)
    aliases: -b <value>
  --skip-npm (Boolean) (Default: false)
    aliases: -sn
  --skip-bower (Boolean) (Default: false)
    aliases: -sb
  --skip-git (Boolean) (Default: false)
    aliases: -sg
```

ember init <app-name> <options...>

```
ember init <glob-pattern> <options...>
  Creates a new ember-cli project in the current folder.
  aliases: i
  --dry-run (Boolean) (Default: false)
    aliases: -d
  --verbose (Boolean) (Default: false)
    aliases: -v
  --blueprint (Path)
    aliases: -b <value>
  --skip-npm (Boolean) (Default: false)
    aliases: -sn
  --skip-bower (Boolean) (Default: false)
    aliases: -sb
  --name (String) (Default: )
    aliases: -n <value>
```

ember serve <options...>

```
Builds and serves your app, rebuilding on file changes.
aliases: server, s
--port (Number) (Default: 4200)
  aliases: -p <value>
--host (String) (Default: 0.0.0.0)
  aliases: -H <value>
--proxy (String)
  aliases: -pr <value>, -pxy <value>
--insecure-proxy (Boolean) (Default: false) Set false to proxy self-signed SSL certificates
  aliases: -inspr
--watcher (String) (Default: events)
  aliases: -w <value>
--live-reload (Boolean) (Default: true)
  aliases: -lr
--live-reload-port (Number) (Defaults to port number + 31529)
  aliases: -lrp <value>
--environment (String) (Default: development)
  aliases: -e <value>, -dev (--environment=development), -prod (--environment=production)
--output-path (Path) (Default: dist/)
  aliases: -op <value>, -out <value>
```

ember build <options...>

```
ember build <options...>
  Builds your app and places it into the output path (dist/ by default).
  aliases: b
  --environment (String) (Default: development)
    aliases: -e <value>, -dev (--environment=development), -prod (--environment=production)
  --output-path (Path) (Default: dist/)
    aliases: -o <value>
  --watch (Boolean) (Default: false)
    aliases: -w
  --watcher (String)
```

ember test <options...>

```
Runs your apps test suite.
aliases: test, t
--environment (String) (Default: test)
  aliases: -e <value>
--config-file (String) (Default: ./testem.json)
  aliases: -c <value>, -cf <value>
--server (Boolean) (Default: false)
  aliases: -s
--port (Number) (Default: 7357) The port to use when running with --server.
  aliases: -p <value>
--filter (String) A regex to filter tests ran
  aliases: -f <value>
--module (String) The name of a test module to run
  aliases: -m <value>
--watcher (String) (Default: events)
  aliases: -w <value>
```

ember generate <blueprint> <options...>

```
ember generate <blueprint> <options...>
  Generates new code from blueprints.
  aliases: g
  --dry-run (Boolean) (Default: false)
    aliases: -d
  --verbose (Boolean) (Default: false)
    aliases: -v
  --pod (Boolean) (Default: false)
    aliases: -p
```

ember destroy <blueprint> <options...>

```
Destroys code generated by `generate` command.
aliases: d
--dry-run (Boolean) (Default: false)
  aliases: -d
--verbose (Boolean) (Default: false)
  aliases: -v
--pod (Boolean) (Default: false)
  aliases: -p
```

Additional `ember-cli` commands

```
ember help <command-name (Default: all)>
  Outputs the usage instructions for all commands or the provided command
  aliases: h, help, -h, --help
ember update
  Updates ember-cli to the newest available version.
ember version
  outputs ember-cli version
  aliases: v, version, -v, --version
```

Let's Create a New App Using `ember-cli`

Demo

Sample App Is Available Here:

http://coen268.peterbergstrom.com/resources/emberclisampleapp.zip

Using Modules & the Resolver

- Manages your dependencies
- Manages your load order of files
- Simple syntax that allows you to have objects NOT in the global namespace
- However, it means that you have to import and export dependencies

An Example: Book Model Object

In `models/book.js`:

```
// always import Ember if you need it
import Ember from "ember";

var Book = Ember.Object.extend({
  displayString: function() {
    return "%@ - %@".fmt(this.get('title'), this.get('pubYear'));
  }.property('title', 'pubYear');
});

export default Book;
```

An Example: Book Model Object Imported In BooksController

```
import Ember from "ember";
import Book from "./models/book";

var BooksController = Ember.Controller.extend({
// Use the Book model object here somewhere...
});

export default BooksController;
```

For More Info About `ember-cli`

Including most of the instructions I just gave:

http://iamstef.net/ember-cli/

Stef Penner talking about `ember-cli` in his 2014 EmberConf talk: https://www.youtube.com/watch?v=4D8z3972h64

ember

COEN 168/268

Mobile Web Application Development

MVC in JavaScript and Ember.js Introduction

Peter Bergström (pbergstrom@scu.edu)

Santa Clara University