Build a Single Page App with Ember.js and Sinatra

Who am I?

Chris Meadows
Director of Software Engineering & Senior Developer
Cloudswell
Email: meadoch1@gmail.com
Twitter: @meadoch1

1996-2013 Professional application development with most all the buzzwords

Aug 2012 – today: Developed, deployed, and am enhancing SPA in ExtJS

Credit where Due!

Luke Melia - @lukemelia
Ryan Bates - @rbates, railscasts.com
Peepcode - @peepcode, peepcode.com
Yehuda Katz - @wycats, yehudakatz.com

Agenda

What’s Ember.js?
How do I make it work?
Q&A

What is Ember.js?

"A framework for creating ambitious web applications"

- from emberjs.com

Say what?

Client-side Javascript MVC framework

Other examples
Backbone, Knockout, ExtJS

Background

Grew from Sproutcore 2.0

Main devs:
Yehuda Katz
Tom Dale

Active development sponsored by Tilde

Focus

Developer Productivity

Pervasive Conventions

Less code required

Elimination of boilerplate*

Easy Development of Complex Web Apps

Long lived states

Very different flow than request/response

(*almost)

Where's the Magic?

Dynamic Runtime Code Generation

Proxies

Observers

Where will I get stuck?

Learn (& follow) the Conventions!

Data exchange

Not a traditional web site mentality

"If something is very difficult, you're probably doing it wrong" - Peepcode

And now for something completely different…

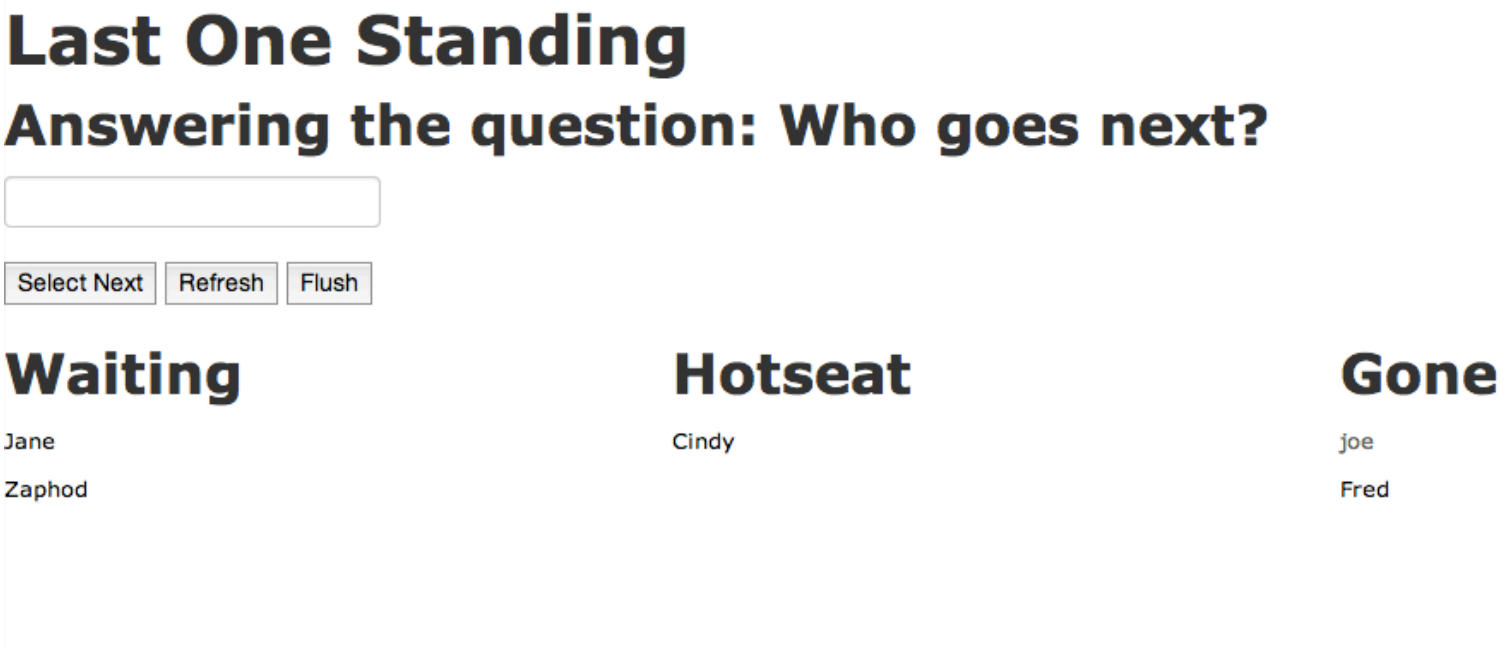
Example Application

Last One Standing
Standup “who goes next” randomizer

Participants
have names and a state

States
there are 3: waiting, hotseat, and gone

Example Application Screenshot



Now back to your
regularly scheduled program

After one more sidenote…

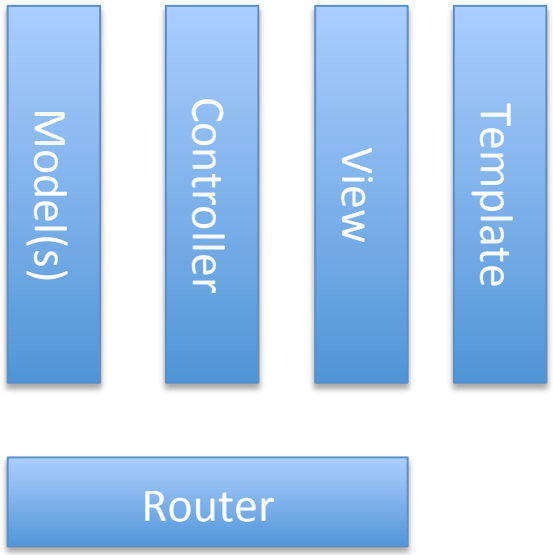
MVC(R)

No “pure” MVC

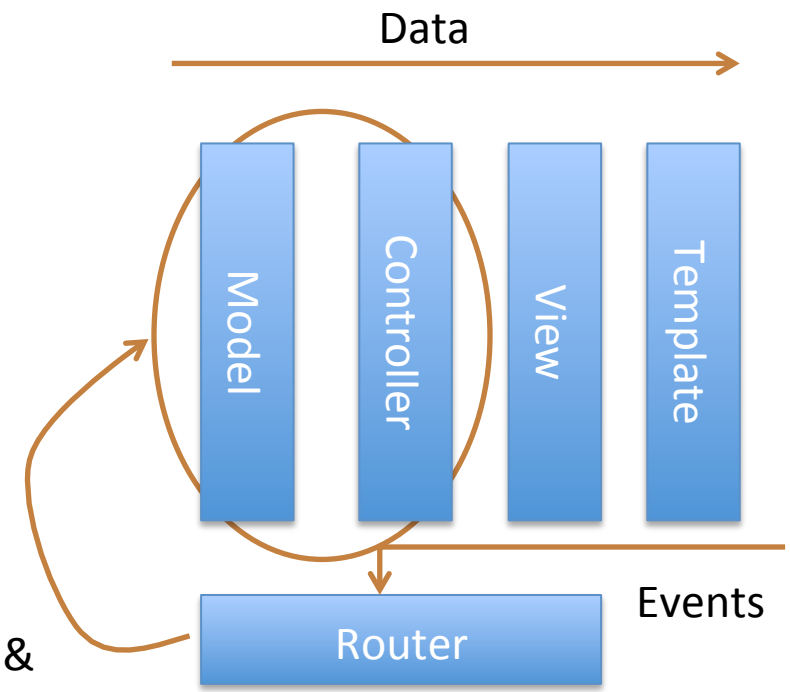
R = Router

How do I make it work?

Application Components



Application Components Information Flow

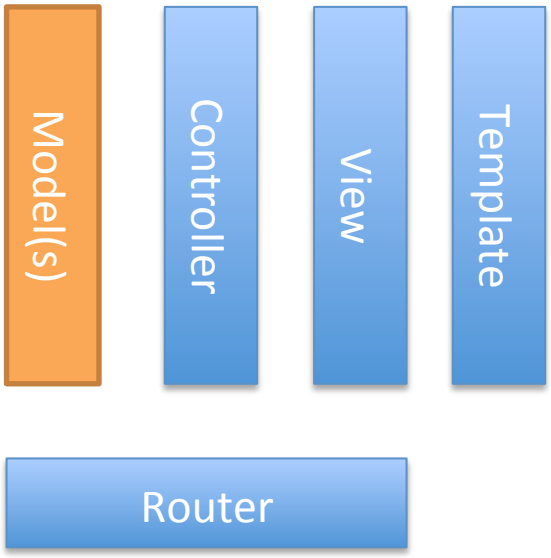


Router updates models & controllers based on events

Application Components Information Lifecycles

Component	Data Lifecycle
Models	The life of the application
Controllers	The life of the session
Views	The life of the page view
Templates	The life of the page view
Router	State machine for session

Application Components
Models



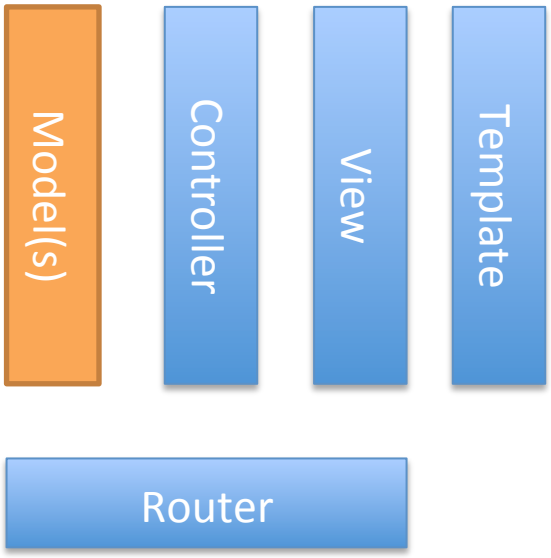
Hold, retrieve, and store data

Lifetime of application

Ignorant of Controllers, Views, app state

Can depend on other models

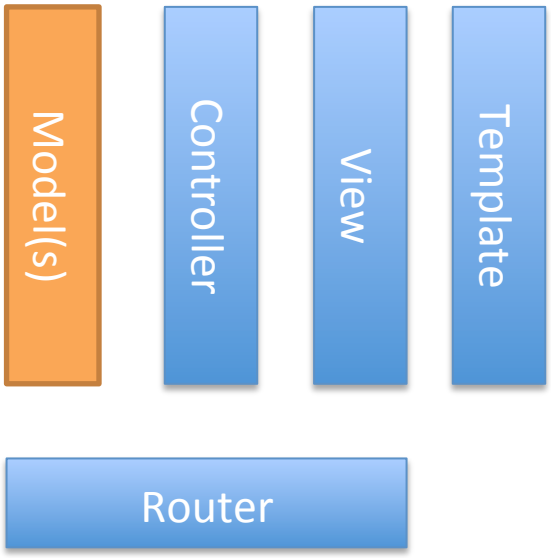
Application Components
Models



Data frequently retrieved through API utilizing an Adapter

Models should go through an Identity Map

Application Components
Models



To work with the router a model must implement:

find(id) – look up and initialize a model based on an id

then(success, failure) – promise pattern

Extend Ember.Object (unless using ember-data)

Application Components
Models – Ember-data

An ORM built for Ember

Implements Store (including an Identity Map), Adapters, and Serializer

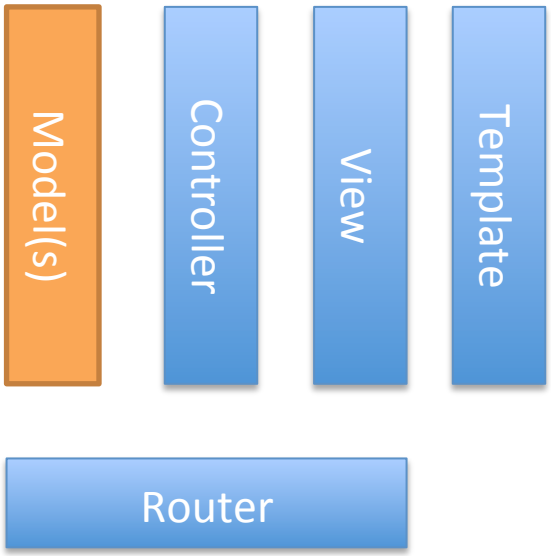
RESTAdapter & BasicAdapter

Supports associations between models

Implements the Router required methods

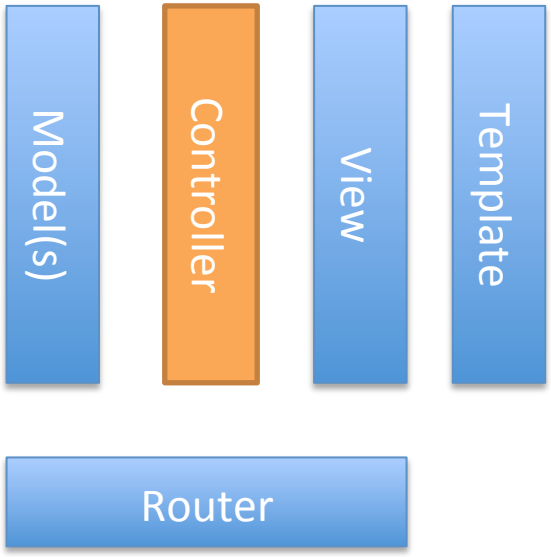
In “beta” phase, but usable

Application Components
Models




```
1  App.Participant = DS.Model.extend({
2      name: DS.attr('string'),
3      state: DS.attr('string')
4  });
```

Application Components
Controllers

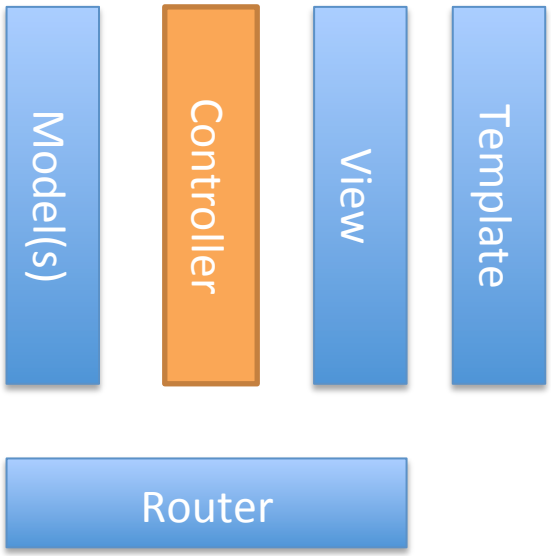


Make model data accessible to view
Handle events
Store “transitory” data

Life of session

Lazily instantiated once upon need

Application Components
Controllers



Included Controllers:

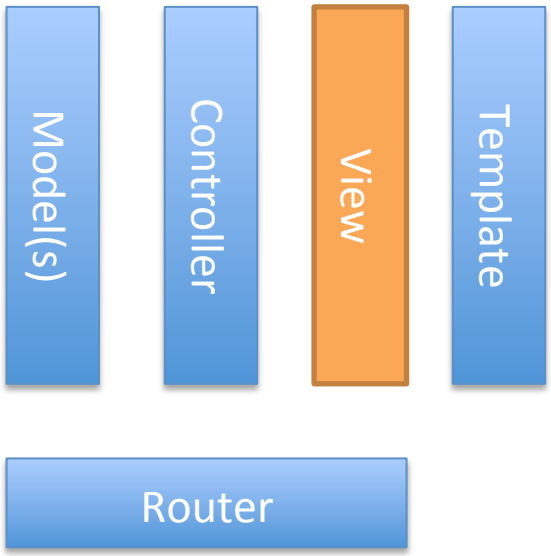
Ember.Controller
Ember.ObjectController *
Ember.ArrayController *

* Allows proxying of underlying model

Application Components
Controllers

```
App.ParticipantsController = Ember.ArrayController.extend({
    content: [],
    addParticipant: function() {
        var newParticipant = App.Participant.createRecord( {
            name: this.get('newParticipantName'),
            state: 'Waiting'
        });
        this.set('newParticipantName', '');
        this.get('store').commit();
    }
});
```

Application Components
Views

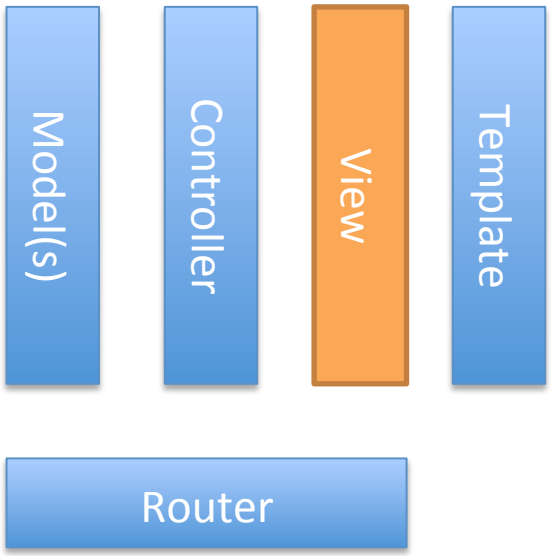


DOM Interaction
Browser events -> semantic events
Frequently optional

Lifetime of page view

Should bind to only one controller

Application Components
Views



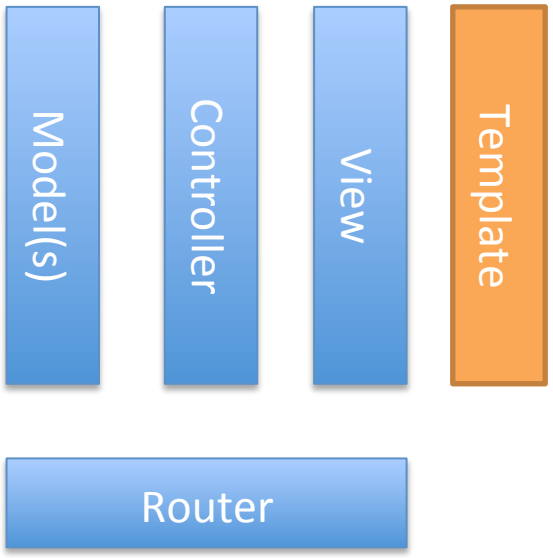
Events Surfaced

Common DOM events (keyDown, mouseMove, touchStart…)

“Workflow” events – didInsertElement, willDestroyElement, etc

Full list at http://emberjs.com/api/classes/Ember.View.html

Application Components
Templates

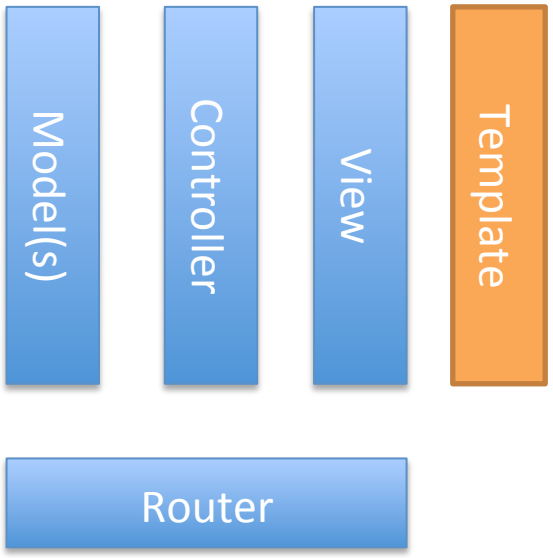


HTML Output
Utilize data bindings

Lifetime of page view

Uses Handlebars

Application Components
Templates




```
<div class ="row-fluid">
<div class="span6">
<ul>
   {{#each item in controller }}
   <li>{{ item }}</li>
   {{/each }}
</ul>
</div>
<div class="span6">
     {{outlet}}
</div>
</div>
```

Application Components
Templates

Conditionals

```
1  {{#if participant}}
2    Hello {{participant.name}}
3  {{else}}
4    Please choose a participant.
5  {{/if}}
```

Application Components
Templates

Loops & Link to

```
<h2>Hotseat</h2>
{{#each participant in controller }}
  {{#if participant.isHotseat}}
    <p>{{#linkTo "participants.show" participant }} {{participant.name}}{{/linkTo}}</p>
  {{/if}}
{{else}}
  <p>No Participants Today</p>
{{/each }}
```

Application Components
Templates
Bound Expressions

```
<div id="avatar">
    <img {{bindAttr src="avatarUrl"}}>
</div>


<div {{bindAttr class=":message color"}}>
    <p>Display with color</p>
</div>

<div class="message red">
    <p>Display with color</p>
</div>


<div {{bindAttr class=":message isAlert"}}>
    <p>I {{#if isAlert}}am{{else}}am not{{/if}} important</p>
</div>

<div class="message">
    <p>I am not important</p>
</div>

<div class="message is-alert">
    <p>I am important</p>
</div>
```

Application Components
Templates

Actions & Outlets

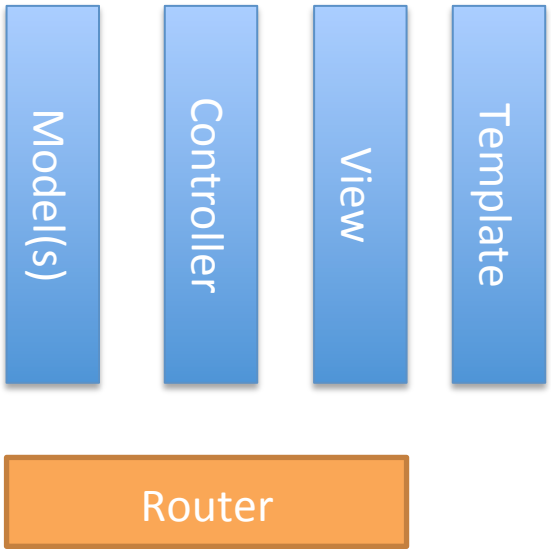
```
<div class="row-fluid">
    <div class="span12">
        <p>{{view Ember.TextField valueBinding="newParticipantName" action="addParticipant"}}</p>
        <p><button {{action drawWinner}}>Select Next</button>
        <button {{action refresh}}>Refresh</button>
        <button {{action flush}}>Flush</button>
        </p>
        {{outlet}}
    </div>
</div>
```

Application Components
Templates

Partials

```
1  <div class ="row-fluid">
2    <div class="span4">
3      {{ partial "participantsWaiting" }}
4    </div>
5    <div class="span4">
6      {{ partial "participantHotseat" }}
7    </div>
8    <div class="span4">
9      {{ partial "participantsNotWaiting" }}
10   </div>
11 </div>
```

Application Components
Router



Map URL to state & objects

Handles data loading for controllers

Application Components
Router

Load model data into controller model/content

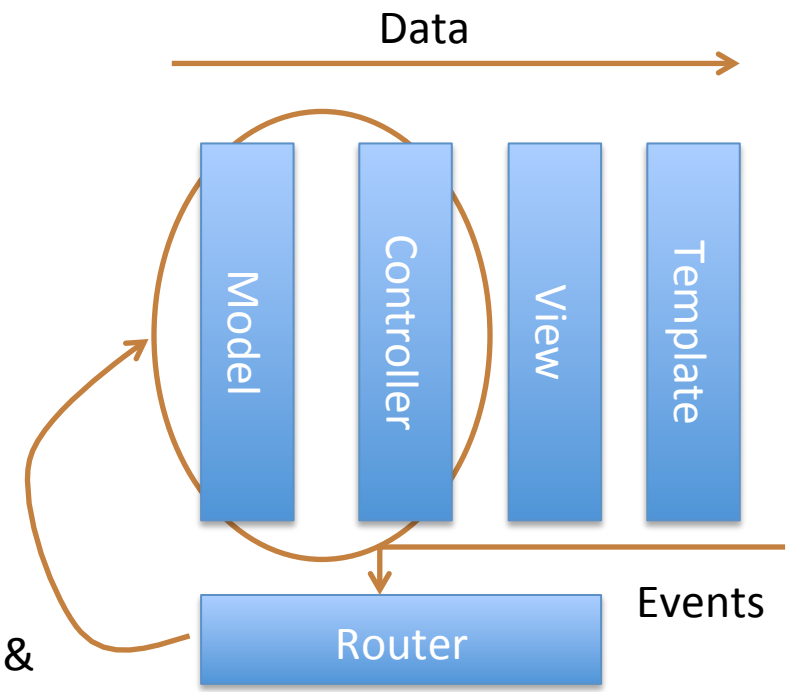
```
App.ParticipantsIndexRoute= Ember.Route.extend({
    model: function() {
        return App.Participant.find();
    }
});
```

Application Components
Router

Redirection

```
1  App.IndexRoute = Ember.Route.extend({
2      redirect: function() {
3          this.transitionTo('participants');
4      }
5  });
```

Application Flow



Router updates models & controllers based on events

Application Components Information Lifecycles

Component	Data Lifecycle
Models	The life of the application
Controllers	The life of the session
Views	The life of the page view
Templates	The life of the page view
Router	State machine for session

Application Structure

Few Files

- All JS in one file
- All templates and html in a second file

Pros

simple

no “build” required

Cons

bad organization for non-trivial projects

Broken up

- Each class in it’s own file
- Each template in a .handlebars file
- Separate directories to store components (ie views, models, etc)

Pros

modular file organization

Cons

lots of files to handle

need a “build” process (largely automated by tools though)

Application Bootstrap

Need to define the following:

"App"
App.Router
App.Store

Must include all files manually

Application Bootstrap
Define “App”

Provide namespacing for objects

```
1   App = Ember.Application.create();
```

Needs to be instantiated before the other js files are loaded

Application Bootstrap
Define Router

Similar to Rails routes.rb

Maps URLs to objects through the naming conventions

```
App.Router.map(function() {
    // index template will be displayed when / is visited
    this.route('index', {path: '/'});

    // participants.index template will be displayed when /participants is visited
    this.resource('participants', function() {

        // participants.show template will be displayed when /participants/1 is visited
        this.route('show', {path: ':participant_id' });
    });
    //both participants routes above will be wrapped in the participants template since they are nested
});
```

Application Bootstrap Naming Conventions

```
1  App.Router.map(function() {
2      // index template will be displayed when / is visited
3      this.route('index', {path: '/'});
4
5      // participants.index template will be displayed when /participants is visited
6      this.resource('participants', function() {
7
8          // participants.show template will be displayed when /participants/1 is visited
9          this.route('show', {path: ':participant_id' });
10     });
11     //both participants routes above will be wrapped in the participants template since they are nested
12 });
```

URL	Route Name	Controller	Route	Template
/	index	IndexController	IndexRoute	index
NA	participants	ParticipantsController	ParticipantsRoute	participants
/participants	participants.index	ParticipantsIndexController	ParticipantsIndexRoute	participants/index
/participants/6	participants.show	ParticipantsShowController	ParticipantsShowRoute	participants/show

Application Bootstrap
Define Store

Defines how data will be retrieved

Mainly used with ember-data

```
1  App.Store = DS.Store.extend({
2      revision: 11,
3      adapter: 'DS.RESTAdapter',
4      url: 'http://127.0.0.1:3000'
5  });
```

Application Bootstrap
Define Store

Sidenote

If using ember-data, set adapter to "DS.FixtureAdapter" to get easy test data loading

```
App.Store = DS.Store.extend({
    revision: 11,
    adapter: 'DS.FixtureAdapter'
});

App.Participant.FIXTURES = [
    {
        id: 1,
        name: "Larry",
        state: "Waiting"
    },{
        id: 2,
        name: "Moe",
        state: "Going"
    },{
        id: 3,
        name: "Curly",
        state: "Gone"
    },{
        id: 4,
        name: "Shemp",
        state: "Waiting"
    }
];
```

Show the application!

Questions?

More info

Dig into emberjs.com for more depth

Google “ember.js”

I’d be glad to talk

Chris Meadows
meadoch1@gmail.com
@meadoch1 (Twitter)

https://github.com/meadoch1/lms