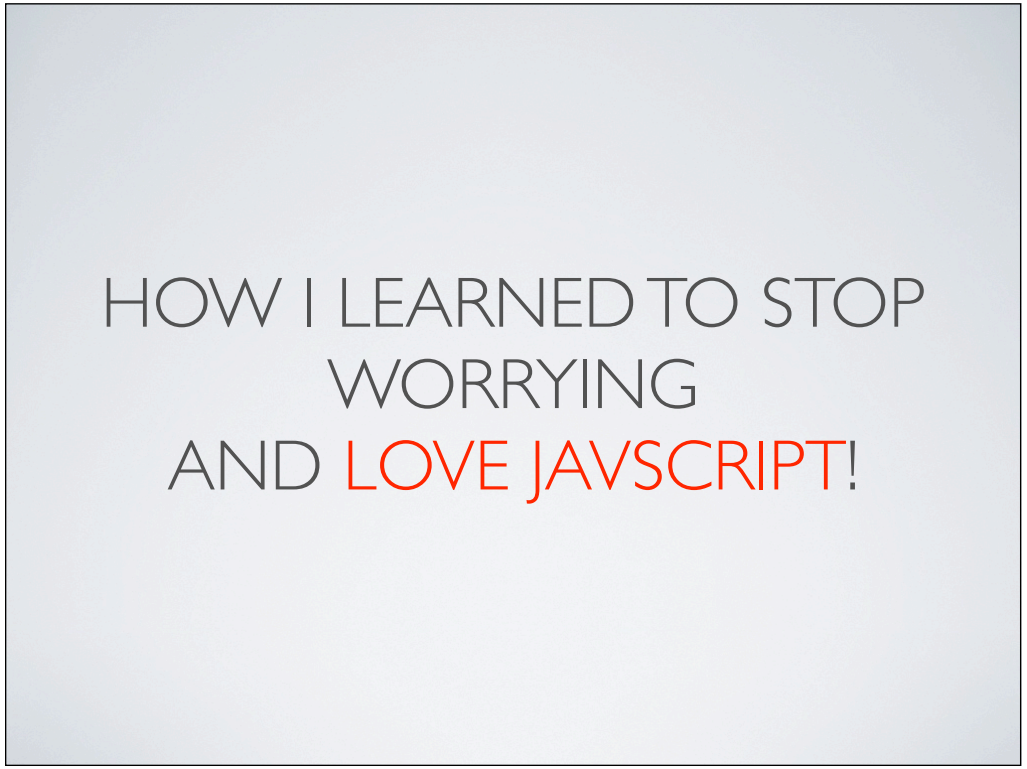
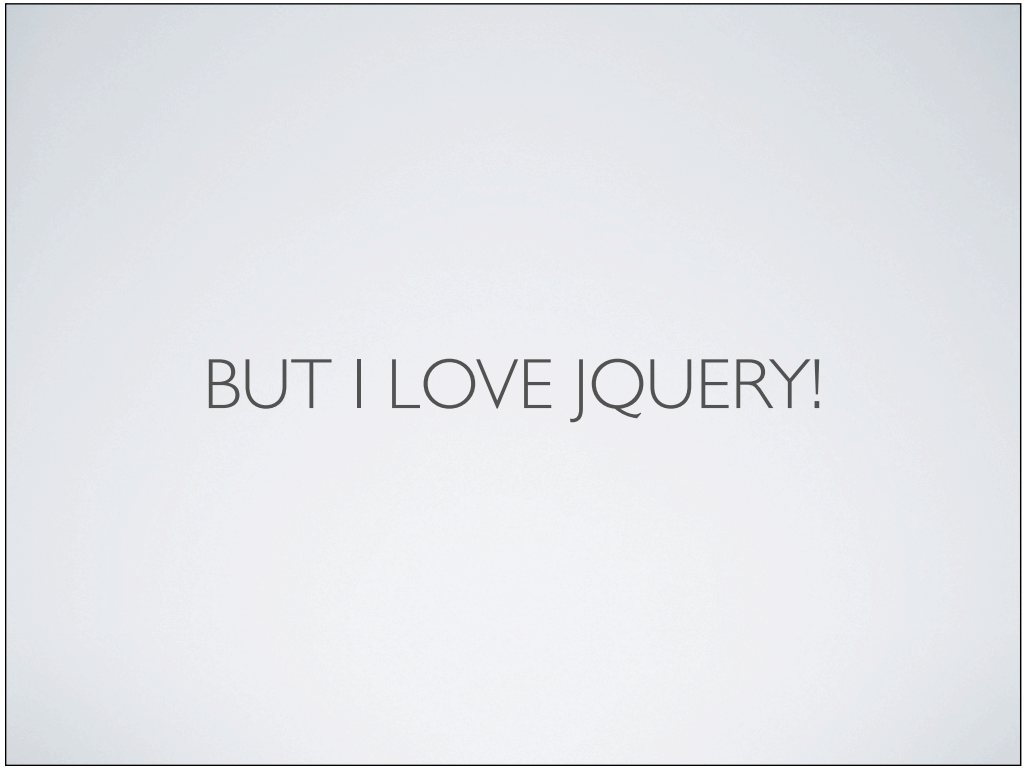
Prateek Dayal, 4 years with Ruby on Rails, CoFounder SupportBee, A help desk Software for SME, like the ones using Basecamp, Also did Muziboo and run HackerStreet .. Will be talking about ... how many of you love js?

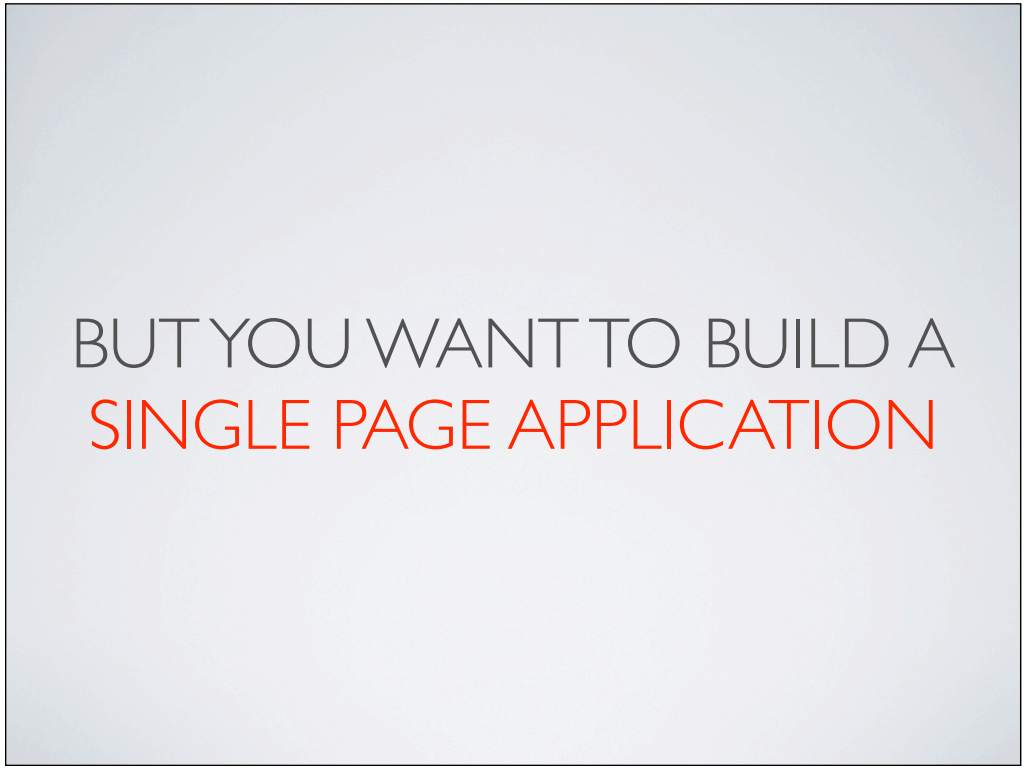


As Douglas Crockford puts it, most widely installed language but most misunderstood or not understood at all. A lot of care for Rails but not so much for JS etc



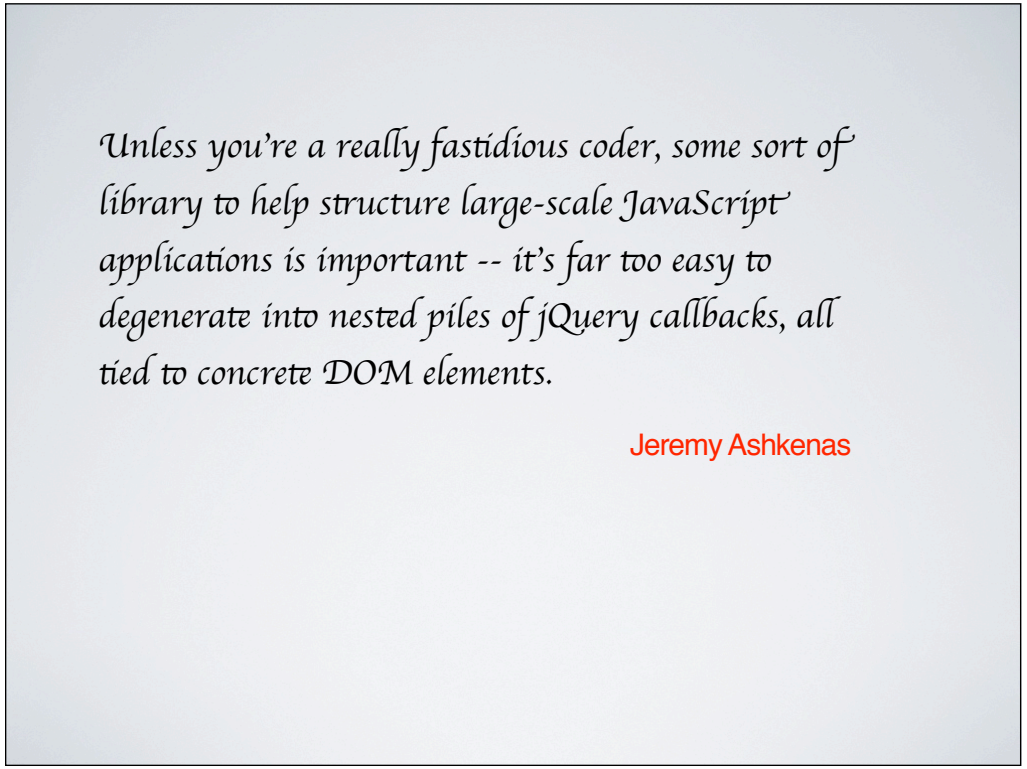
How many of you have used jQuery?

If you have to select a few elements and show/hide them or apply a css property etc



Something like Gmail. Incoming Emails affect multiple items in the view etc.


mobile
me



Choosing a framework is like choosing Ruby on Rails. It gives you a set of conventions and a place to put things. A lot of good choices have been already made for you.

Most of this talk is based on our experience of building SupportBee, The ticketing page is a single page app like gmail. The code is from SB

COMPLEX SINGLE PAGE APPS HAVE

- All of the client logic in Javascript
- Updates to many parts of the UI on changes to data
- Templating on the client side

Only communicate data with the server after being loaded.

TO NOT GO CRAZY
BUILDING IT, YOU NEED

- A MVC like pattern to keep the code clean
- A templating language like HAML/ERB to easily render view elements
- A better way to manage events and callbacks
- A way to preserver browser's back button
- Easy Testing :)

FORTUNATELY, THERE IS HELP!

Created by Charles Jolley, Used in Mobile Me, iWork.com, In Strobe Inc

Created by the founders of 280 North Inc. Used in GoMockingBird, Github Issues

BUT THEY ARE BIG

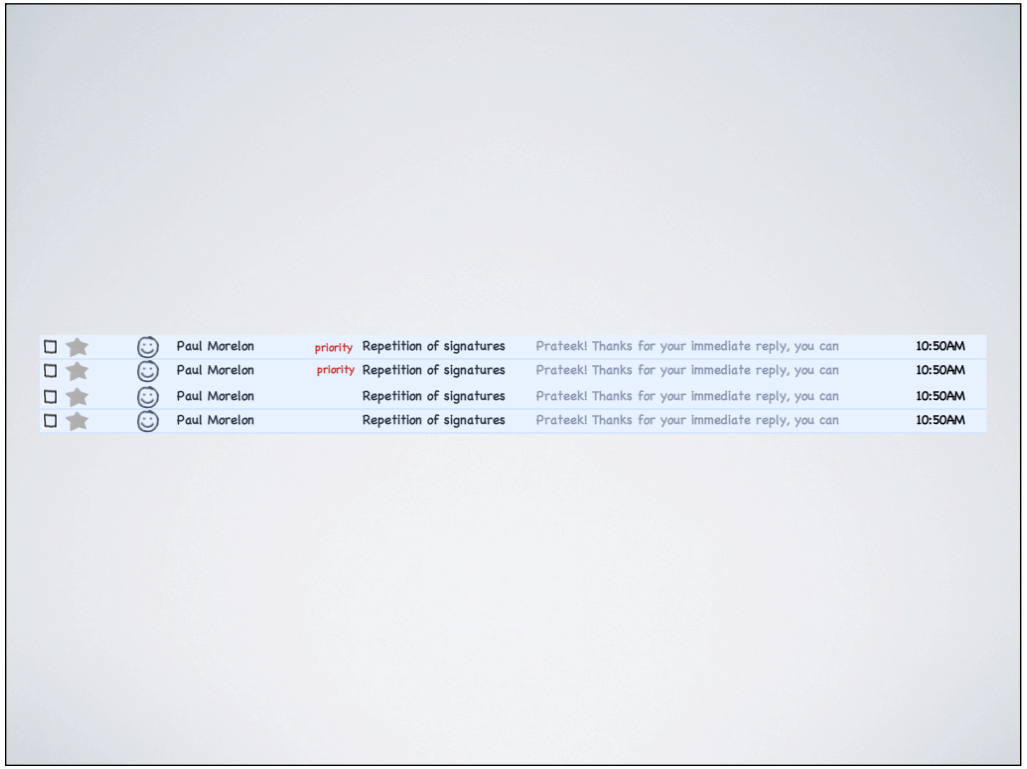
AND HAVE A
LEARNING CURVE

Done by Jeremy Ashkenas. He has also created coffeescript and jammit. Jammit and Backbone is part of the document cloud code. Basecamp mobile is built using Backbone

- 3.9 kb packed and gzipped
- Only dependency is underscore.js
- You need jQuery or Zepto for Ajax
- Annotated source code

MVC

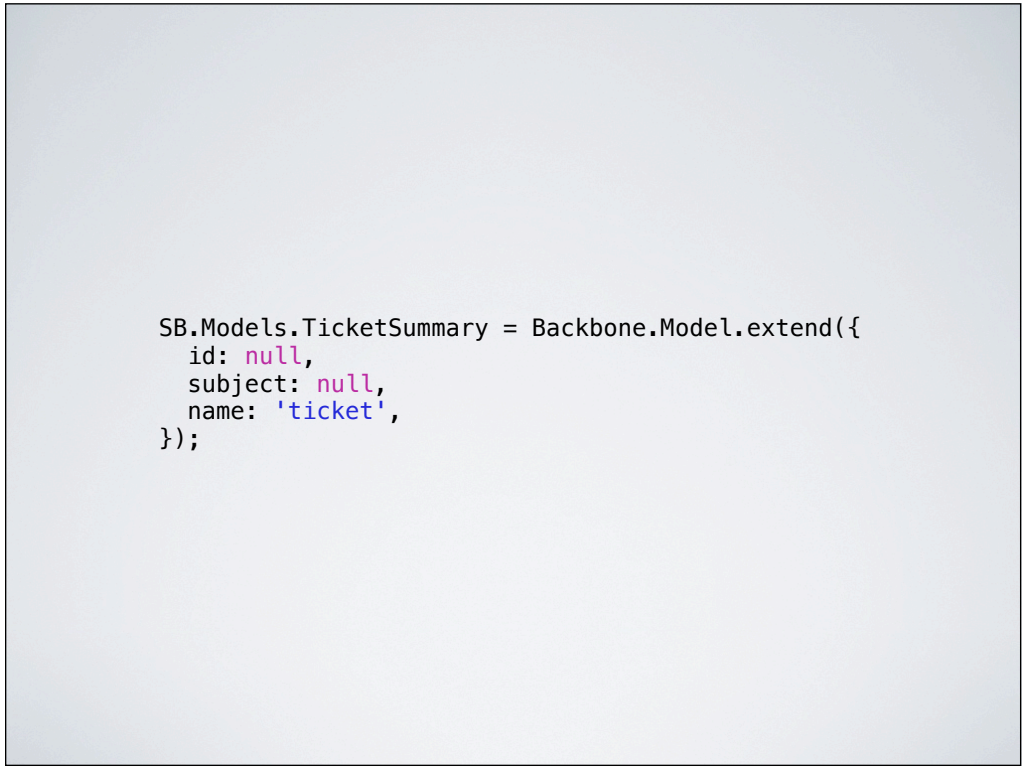
MODELS, VIEWS & COLLECTIONS



You get a json of tickets and display it. And on clicking the ticket the url should be changed and ticket should be displayed

MODELS

- Data is represented as Models
- Can be Created, Validated, Destroyed & Persisted on Server
- Attribute changes trigger a 'change' event



```
SB.Models.TicketSummary = Backbone.Model.extend({
  id: null,
  subject: null,
  name: 'ticket',
});
```

Unlike a Rails' model, there is no inbuilt support for Associations yet
You can initialize values in the initialize method
extend helps you setup the prototype chain so that you can again inherit from this model

COLLECTIONS

- A collection of models
- Triggers events like add/remove/refresh
- Can fetch models from a given url
- Can keep the models sorted if you define a comparator function

```
SB.Collections.TicketList = Backbone.Collection.extend
({
   model: SB.Models.TicketSummary,

   url: "/tickets",
   name: "tickets",

   initialize: function(models, options){
     // Init stuff if you want
   }
});
```

This is a collection of model SB.Models.TicketSummary

VIEWS

- They are more like Rails' Controllers
- Responsible for instantiating Collections and binding events that update the UI

```
SB.Views.TicketListView = Backbone.View.extend({

   tagName: 'ul',

   initialize: function(){
     this.ticketList = new SB.Collections.TicketList;
     _.bindAll(this, 'addOne', 'addAll');

     this.ticketList.bind('add', this.addOne);
     this.ticketList.bind('refresh', this.addAll);
     this.ticketList.bind('all', this.render);

     this.ticketList.fetch();

   },

   addAll: function(){
     this.ticketList.each(this.addOne);
   },

   addOne: function(ticket){
     var view = new SB.Views.TicketSummaryView({model:ticket});
     $(this.el).append(view.render().el);
   }
});
```

Views also instantiate other views, which can render templates

```
SB.Views.TicketSummary = Backbone.View.extend({

  initialize: function(){
    _.bind(this, 'render');
    if(this.model !== undefined){
      this.model.view = this;
    }
  },

  events: {
    'click': 'openTicket'
  },

  openTicket: function(){
    window.location = "#ticket/" + this.model.id;
  },

  render: function(){
    $(this.el).html(SB.Views.Templates['tickets/
summary'](this.model.toJSON()));
    return this;
  }

});
```

Event binding for click happens here
Renders a handlebars' template
The model stores a reference to this view. This reference can be used to remove/update view on changes to the attribute

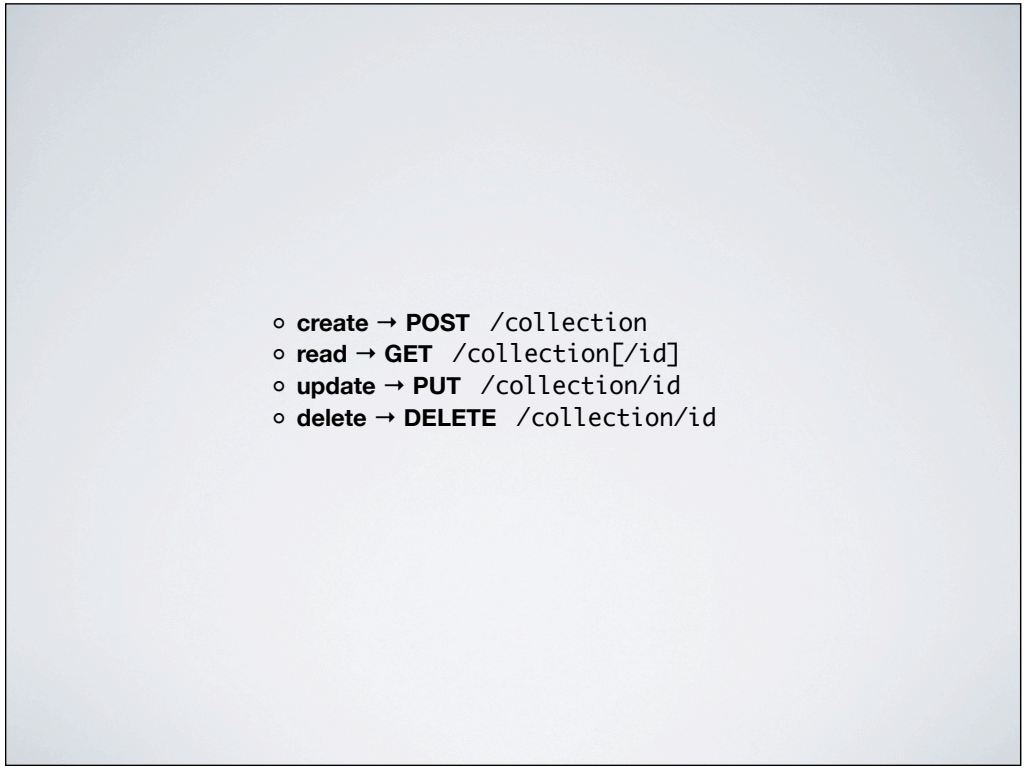
HANDLEBARS!

```
{{#ticket}}
  <div class="listing-cell name">
    <p>{{nameOrEmail requester}}
({{replies_count}})</p>
  </div>

  <div class="listing-cell subject">
     <a href="#{{id}}">
     {{#unread}}
     <b>{{subject}}</b>
     {{/unread}}
     {{^unread}}
     {{subject}}
     {{/unread}}
     </a>
  </div>
{{/ticket}}
```

Logicless template. Simple JSON parsing and if/else
nameOrEmail is a helper function that you can register

URLS

- Collections can have a url
- Models can have a url or they can derive it from collection's url



Models inherit their urls from collections. For example

FINALLY, CONTROLLERS

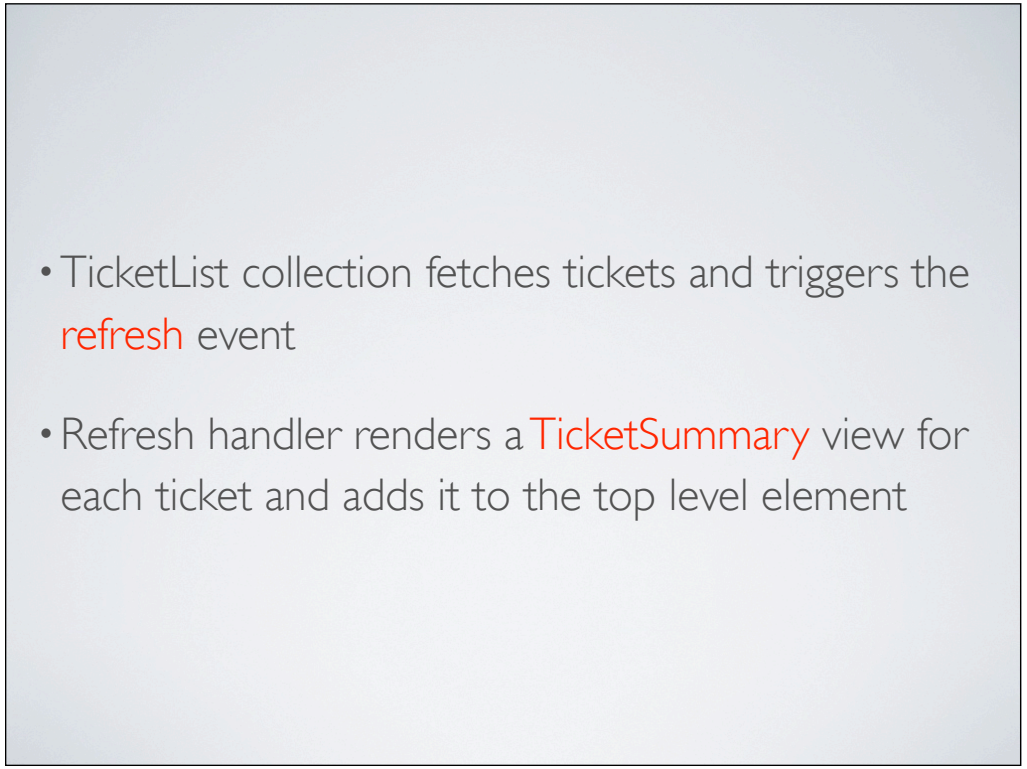
- Used for setting up the routes
- You can add new routes during runtime

```
SB.Controllers.AgentHome = Backbone.Controller.extend({
  routes: {
    "": "dashboard", // http://app_url
    ":id":  "openTicket" // http://app_url#id
  },

  dashboard: function(){
    var view = new SB.Views.TicketListView;
    $("#ticket").hide();
    $("#list").html(view.el);
    $("#list").show();
  },

  openTicket: function(id){
    var view = new SB.Views.TicketView({model : new
SB.Models.Ticket({id : id})});
    $("#list").hide();
    $("#ticket").html(view.el);
    $("#ticket").show();
  }
});
```

The openTicket route will be fired when the ticket is clicked

```
window.controller = new SB.Controllers.AgentHome
```

- Initiates a new controller instance
- which matches the dashboard route
- and creates a instance of TicketListView
- and a TicketList collection is created



The controller has already appended the top level element to the list element on the page and has made it visible. So when tickets are populated, they show up on the page

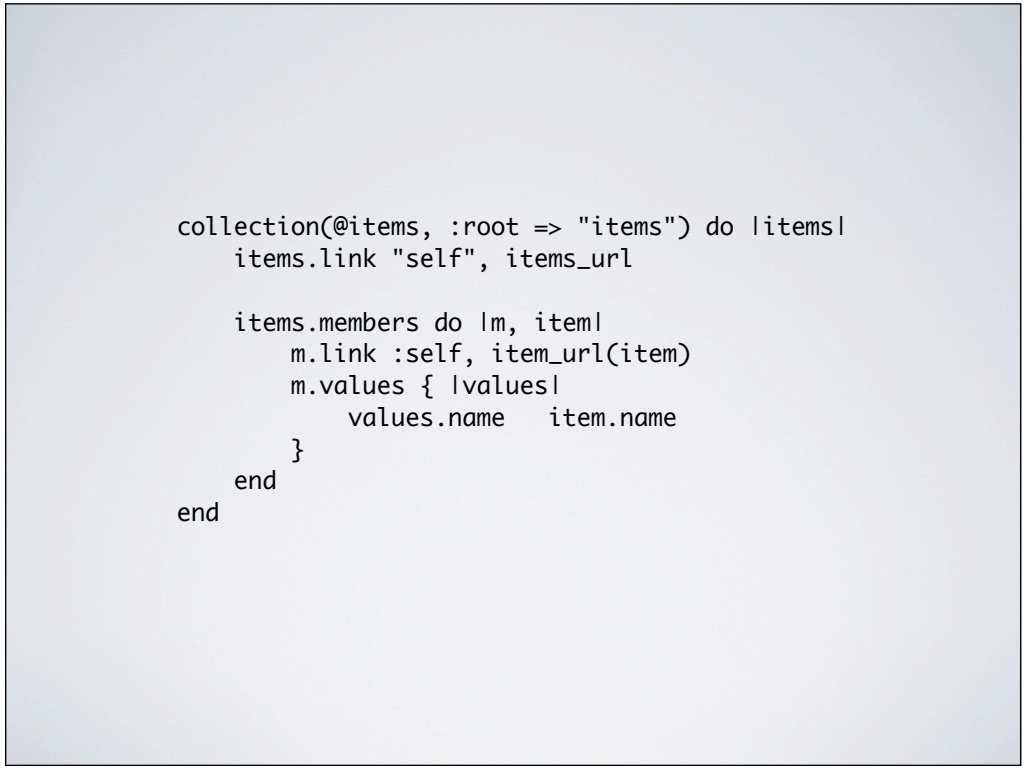
- Every Ticket Summary row has an onclick handler
- which changes to url to #ticket_id
- which matches the openTicket route

SERVER SIDE

```
ActiveRecord::Base.include_root_in_json = false
```

Though you can keep this and add some parsing code to parse the json out on client side

- You can use as_json to customize the json responses
- Or you can use Restfulie
- Restfulie lets you easily put links in representations too



```
collection(@items, :root => "items") do |items|
    items.link "self", items_url

    items.members do |m, item|
        m.link :self, item_url(item)
        m.values { |values|
            values.name   item.name
        }
    end
end
```

Look at Rest in Practice or Roy Fielding’s blog for more info

NOT JUST FOR API BACKED MODELS

```
//window.AgentHomeController = new SB.Controllers.AgentHome;
window.mainView = new SB.Views.Main;
mainView.screens.add(new SB.Models.Screen({
  title: 'Dashboard',
  href: '#dashboard',
  listings: [
   {
      title: 'New Tickets',
      url: '/tickets?label=unanswered&replies=false'
   },
   {
      title: 'Ongoing Tickets',
      url: '/tickets?label=unanswered&replies=true'
   },
   {
      title: 'Starred Tickets',
      url: '/tickets?starred=true'
   }
   ]
}));
```

TESTING

JASMINE FTW!

```
beforeEach(function(){
   var ticket = new SB.Models.TicketSummary({id: 1,
                            subject: "A Ticket",
                            replies_count: 0,
                            requester: {id : 1,
                            email: 'requester@example.com',
                            name: 'Requester',
                            thumbnail: 'thumbnail-url'}
                            });
   this.view = new SB.Views.TicketSummary({model: ticket});
   this.view.render();
});

it("should render a ticket summary correctly", function(){
   expect($(this.view.el)).toHaveText(/A Ticket/);
   expect($(this.view.el)).toHaveText(/Requester/);
});

it("should change the url on click", function(){
   var currentLoc = window.location.toString();
   $(this.view.el).click();
   expect(window.location.toString()).toBe(currentLoc + "ticket#1");
});
```

SINON FOR MOCKS/STUBS

```
  beforeEach(function(){
    this.server = sinon.fakeServer.create();
    this.server.respondWith("GET", "/tickets_url_mock_server",
                    [200, { "Content-Type": "application/json" },
                   '{"tickets":[{"id": 1, "subject": "Ticket 1"},
  {"id": 2, "subject": "Ticket 2"}]}'
                  ]);
  });

  it("should instantiate view when tickets are fetched", function
  (){

     var viewMock = sinon.mock(SB.Views);

     viewMock.expects("TicketSummary").once().returns(new
  Backbone.View);
     viewMock.expects("TicketSummary").once().returns(new
  Backbone.View);

     var view = new SB.Views.TicketList({url: "/
  tickets_url_mock_server"});

     this.server.respond();
     viewMock.verify();
     expect($(view.render().el).children().length).toBe(2);
  });
```

QUESTIONS?

prateek@supportbee.com

@prateekdayal

http://supportbee.com