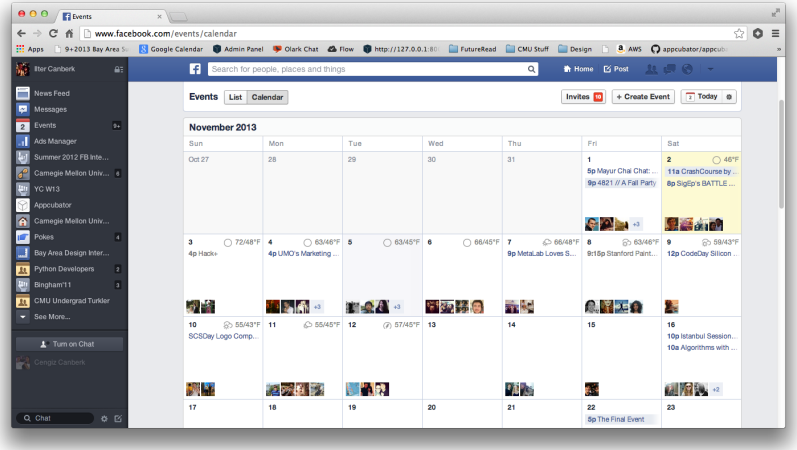
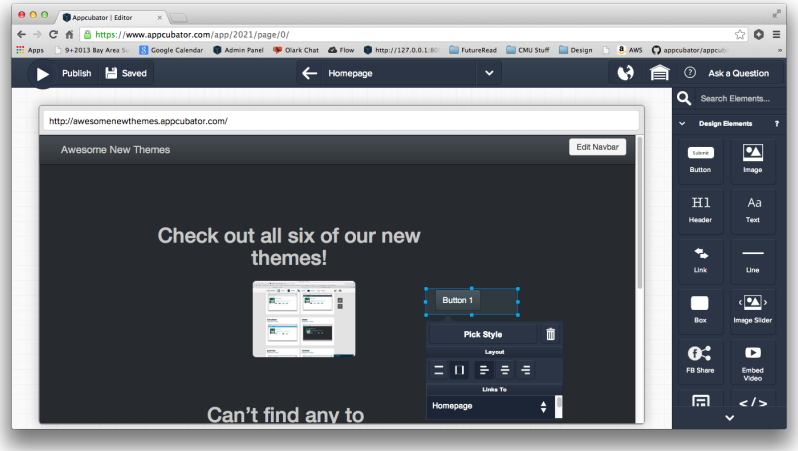
Client-Side Webapps and Backbone.js

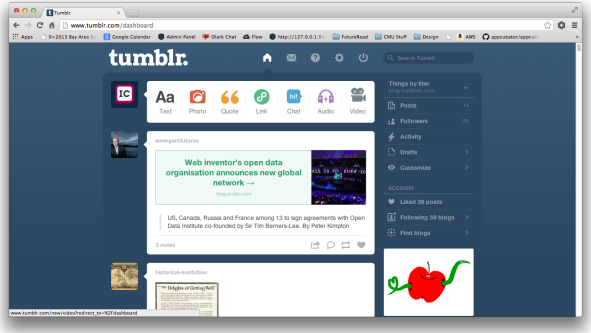
Ilter Canberk, Appcubator

- Junior, ECE student at CMU
- Interned at Facebook at some point
- Co-founder of Appcubator



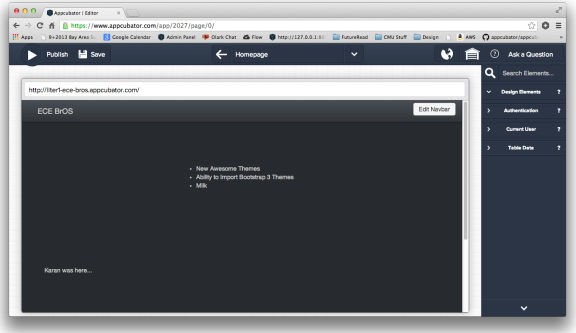


Web-apps



Static

More-dynamic/Ajaxified



Fully-Client Side

Static Sites

Simple

Loads Fast

You don’t really have much to worry about…

Ajaxified Sites

Pros:

Page loads are fast

Cons:

JS needs to query the whole DOM tree.

Harder to retain references to the elements.

Fully Client-Side

Pros:

Highly Dynamic

Easier to maintain the interactivity

Faster page switch

Cons:

(Sometime really) Slow initial page load

So what’s the right way

- Just keep it static if you’re not going to write more than 300 lines of Javascript
- Ajaxify stuff if there is a lot of content on a single page and you still want to have the smooth page switches
- If >80% of your page is dynamically loaded content, you might as well make it completely client-side.

Some of the MVC frameworks

Ember.js (64 kb)

Angular.js (81kb)

Backbone.js (19 kb)

For more: http://todomvc.com/

Why Backbone?

- It’s light weight
- Doesn’t do magic
- Let’s you have your own architecture

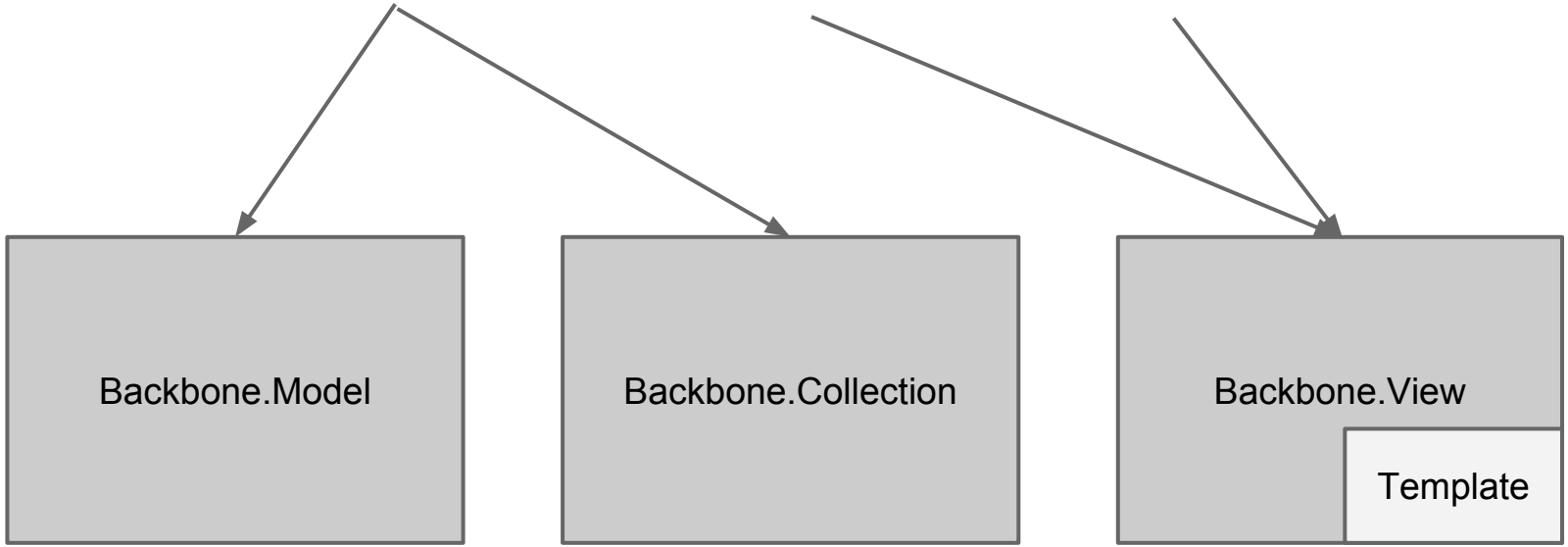
- Awesome inheritance

Dependencies

jQuery - easy to use (quite slow…)

underscore.js - utility-belt library for JavaScript that provides a lot of the functional programming support

Model - View - Template



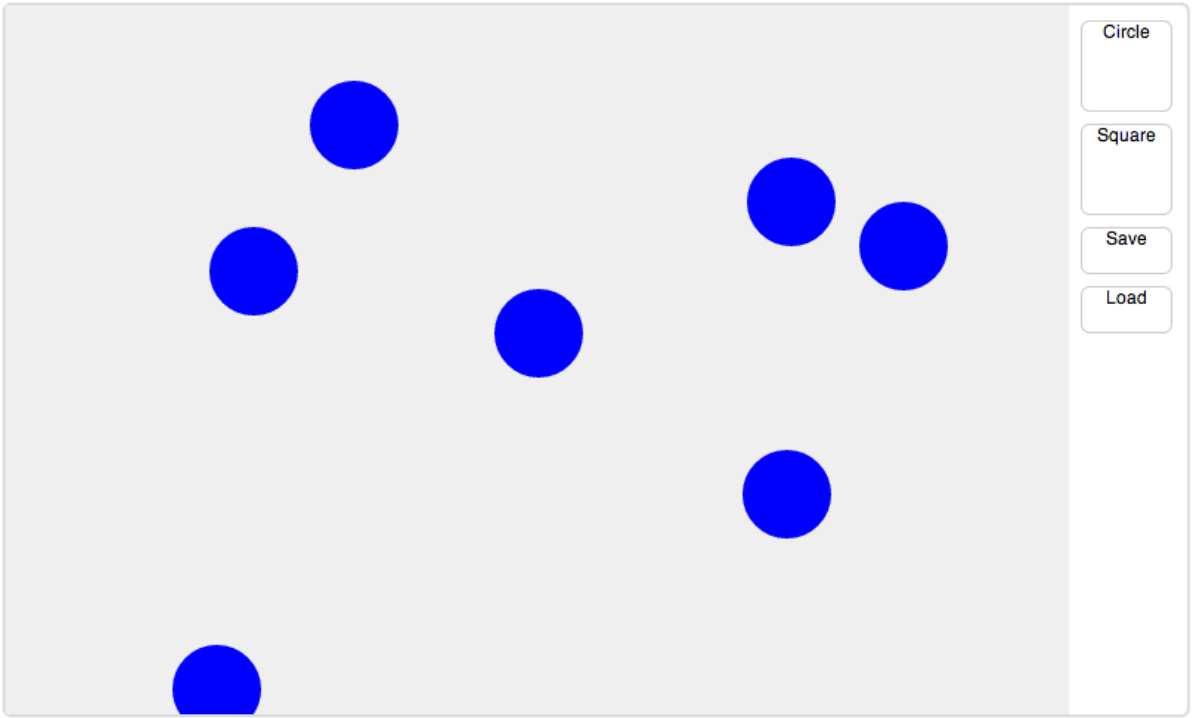
Please download/clone...

https://github.com/icanberk/BackboneDemo

Your best friend

http://backbonejs.org/

Bunch of shapes



Models (*{objects in json}*)

- They define the structure of the data.
- They have properties
- Properties can be primitives, other *models* or *collections*.

BackboneDemo / js / data / **ShapeModel.js**

```
var ShapeModel = Backbone.Model.extend({

    initialize: function(coords) {
        this.set('xCoord', coords.xCoord);
        this.set('yCoord', coords.yCoord);
    },

    setType: function(str) {
        this.set('type', str);
    },

    toJSON: function() {
        var json = _.clone(this.attributes);
        // need to call toJSON on nested collections and models
        return json;
    }
});
```

More sample code

Normal JS

```
var shapeModel1 = {};


shapeModel1.type = ‘circle’;
console.log( shapeModel1.type ); // circle
```

Backbone Way

```
var shapeModel1 = new ShapeModel();


shapeModel1.set(‘type’, ‘circle’);
console.log( shapeModel1.get(‘type’);) // `circle`
```

But why set and get?

It is more safe.

It let’s backbone model do some validations before changing the property.

It makes the model emit/trigger events that other modules can listen to. (Will talk more about the observer pattern in views)

Collection *([arrays in json])*

- They contain and maintain set of models.
- There are pre-built functions to manipulate, or get models.
- You are also free to define your own

BackboneDemo / js / data / **ShapeCollection.js**

```
var ShapeCollection = Backbone.Collection.extend({
    model: ShapeMode,

    getCircles: function() {
        /* some code */
    },
    getSquares: function() {
        /* some code */
    }
});
```

Views

- They render DOM elements and retain references to them.
- They generally correspond to one model. (e. g. ShapeModel -> ShapeView)
- They need to have an hierarchy of the layout. (superviews/subviews)

BackboneDemo / js / views / **CircleView.js**

```
var CircleView = Backbone.View.extend({

    className: 'shape-widget circle',
    events: {
      'click .blabla' : 'functionyouwanttocall'
      // TODO: remove the element on click
    },
    initialize: function(shapeModel) {
      _.bindAll(this);
      this.model = shapeModel;
    },
    render: function() {

      this.el.style.left = this.model.get('xCoord') + 'px';
      this.el.style.top  = this.model.get('yCoord') + 'px';
      return this;
    }
  });
```

Observer Pattern = Less Coupling

BackboneDemo / js / views / **BoardView.js**

```
initialize: function() {
    _.bindAll(this);

    this.model = new BoardModel({});

    // listeners (read(or ask me) more about observer pattern)
    this.listenTo(this.model.get('shapes'), 'add', this.renderShape);

}
```

I can just remove this BoardView module, and nothing in the codebase will break.

How to put these modules together?

```
<script src="js/libs/module1.js"></script>
<script src="js/libs/module2.js"></script>
<script src="js/libs/module3.js"></script>
.
.
.
.
.
.
.
.
.
.
.
<script src="js/libs/module99.js"></script>
<script src="js/libs/module100.js"></script>
```

Do I know if module60.js is loaded?

File Tree from Appcubator code-base

```
├── AnalyticsView.js
├── AppInfoView.js
├── AppRouter.js
├── CollaboratorsView.js
├── DeployView.js
├── DownloadModalView.js
├── GarageView.js
├── InvitationsView.js
├── OverviewPageView.js
├── PluginsView.js
├── QuickTour.js
├── RedoController.js
├── RouteLogger.js
├── ShareModalView.js
├── SimpleTwitterTour.js
├── SoftErrorView.js
├── Striper.js
├── ThemeDisplayView.js
├── ThemesGalleryView.js
├── TwitterTour.js
├── WorldView.js
├── dicts
│   ├── constant-containers.js
│   ├── default-uielements.js
│   └── page-templates.js
├── editor
│   ├── CustomWidgetEditorModal.js
│   ├── EditorGallerySectionView.js
│   ├── EditorGalleryView.js
│   ├── EditorView.js
│   ├── FacebookShareEditor.js
│   ├── FooterEditorView.js
│   ├── FooterView.js
│   ├── GuideView.js
│   ├── ImageSliderEditorView.js
│   ├── KeyDispatcher.js
│   ├── LinkEditorView.js
│   ├── MarqueeView.js
│   ├── MouseDispatcher.js
│   ├── MultiSelectorView.js
│   ├── NavbarEditorView.js
│   ├── NavbarView.js
│   ├── PageTemplatePicker.js
│   ├── PickCreateFormEntityView.js
│   ├── QueryEditorView.js
│   ├── SearchEditorView.js
│   ├── ToolBarView.js
│   ├── VideoEmbedEditor.js
│   ├── WidgetClassPickerView.js
│   ├── WidgetContainerView.js
│   ├── WidgetContentEditorView.js
│   ├── WidgetCustomView.js
│   ├── WidgetEditorView.js
│   ├── WidgetEditorViewProxy.js
│   ├── WidgetFormView.js
│   ├── WidgetLayoutEditorView.js
│   ├── WidgetListView.js
│   ├── WidgetSelectorView.js
│   ├── WidgetView.js
│   ├── WidgetsManagerView.js
│   ├── editor-templates.js
│   ├── form-editor
```

…. Could not fit all

Require.js

JavaScript file and module loader.

>> import utilities.py
>> #include “utilities.c”

For Javascript:

var ViewModule = require(‘views/ViewModule’);

Task 1:

Making the canvas responsive.

Hint: On click to `.canvas` BoardView should call addShape function.

Task 2:

Bind load and save buttons to corresponding functions.

Hint: looks at how alert button is bound.

Task 3:

Make the Circle button highlighted, if the mode is “circle”. Make the Square button highlighted, if the mode is “square”.

Hint: You can just use ‘active’ class.

Task 4:

Implement SquareView.js.

Task 5:

Implement remove functionality. Circles and Squares should remove themselves from the board when clicked.

Thanks for listening!

Send your questions to ilter@appcubator.com