BUILDING A PRODUCT WITH EMBER.JS AND FRIENDS

INTRODUCTION

JavaScript developer, open source enthusiast

work for Innovu, a data analytics firm

mildly social under @eliflanagan

WHY

because JS is fun

because web application JavaScript frameworks abound

todomvc is nice, but how does this correspond to building an actual product?

STACK / TOOLS

- node.js 6 + express API server + apache front end
- ember.js 2.4.x LTS with ember-data 2.4
- ember-cli 2.5.1

AN MVCISH FRAMEWORK

- top notch router `/some/path`
- ember-data as a data persistence library (the M)
- component driven development flow for the user interface (the V, eventually C)

HUMMINGWORDS

- DDAU (Data Down Actions Up)
- Glimmer engine
- JSONAPI (our API mostly conforms to)
- "happy paths"

WORKFLOW

ember-cli preferred tool for creating new ember projects

all application code in `app` directory

lets you write modern JavaScript throughout project (ember-cli-babel, Babel)

`ember build` outputs project to a folder, `dist` by default

DEV VIGNETTES

- a 2 factor login flow feature
- including external dependencies
- using `ember-cli` as a test tool

23 REASONS WHY ANGULAR *IS* BETTER THAN EMBER

A 2 FACTOR LOGIN FLOW

began with ember 1.12, then 1.13, then 2.x

initially built with controllers

refactored behavior into routes

ROUTE SNIPPET

```
import Ember from 'ember';

export default Ember.Route.extend({
    // various router hooks
    model() {},
    actions: {
      firstStep() {
        // ask API for next step
        // transition to other login.next
      }
    }
});
```

EXTENDING APP FUNCTIONALITY

add on ecosystem

including client side libraries

bower.json:

```
{
  "name": "ourapp",
  "dependencies": {
    "highcharts": "4.1.10"
  }
}
```

ember-cli-build.js

```
    app.import('bower_components/highcharts/highcharts.js');
```

USING AN NPM PACKAGE

ember-browserify to the rescue

package.json

```
  "dependencies": {
    "lodash": "^3.6.0"
  }
```

app/routes/application.js

```
import _ from 'npm:lodash';
import Ember from 'ember';
```

TESTING APPLICATION CODE

1. integration testing components
2. unit testing routes, utilities, helpers, controllers
3. acceptance testing for features
    - Juan Doe can log in and see a page

PASSING DATA AROUND, UPDATING THINGS

grappling with underlying observable system

applying DDAU

async-components

MODELING RELATIONAL DATA

learning JSONAPI

lacking strict DB structure to client side model definitions

ember-data seems better suited for traditional, integer based data structures

we often use composite primary keys, no default support

GENERAL WORKFLOW

2 package managers (bower and npm)

upgrading ember-cli a manual process

CONCLUSION

maturing

useful

room to grow

CREDITS AND MISC HYPERLINKS

- team I work with
- ember.js community via
  - http://discuss.emberjs.com/
  - https://embercommunity.slack.com/messages/help/
- background on LTS http://emberjs.com/blog/2016/02/25/announcing-embers-first-lts.html
- explanation of naming conventions in ember-cli: https://ember-cli.com/user-guide/#module-directory-naming-structure