Magenic®

Angular.js and Typescript

From SPA to Enterprise App. Is it ready?

HTML5/JS: the answer to everything?

- No
  - Native platforms, at least to date, have a native experience. An app that needs access to system resources not exposed in a browser should be developed in their respective environment.
- Yes
  - Browser platforms are now exposing more and more functionality to their devices through the javascript api.
- It Depends…
  - The HTML5/JS route presents an attractive option for developing a visually attractive, maintainable, easily deliverable app.

HTML5/JS: the answer to Enterprise

- Easily deliverable for experiences on devices throughout your enterprise.
  - Using reactive design principles you can deliver to multiple screen formats
  - Platforms (like airwatch) allows you to deliver a secure platform across the internet/cell networks.
- Most developers know, or can easily learn the languages and principles.
  - But do current principles and methods in web development meet the challenges of complex application development?

HTML5/JS: How it gets it done

- Just as in the days of yore…
  - Failed attempts at writing good software (Bad software?) have taught us the lessons we need and have allowed patterns to emerge that help us to write great, consistent experiences for our stakeholders.
  - Frameworks emerge to support those patterns….
  - Thus Sprach Angularjs

Session Objectives

- Overview of Typescript
- Overview of Angular
- How Angular And Typescript play well together.
- Where we could use some help
- Architecture Caveats
- Questions and Answers

Things to know

- Assume knowledge of ASP.NET MVC (I'll be using v4 for the most part)
- Client is Javascript/HTML5 with a distinct separation of your Client Application Code/Library Code/Views
- Architecture: MV* (or Model View Star)
- Dependency Injection, used by both AngularJs and Typescript

Typescript: A quick Introduction

- C-Style language.
- Subset of the javascript language:
  - Valid javascript is valid Typescript.
- Typescript adds
  - Typing
  - Classes
  - Modules
  - Generics

Code Demonstration

Typescript

AngularJs: What can it do?

- Super Heroic:
  - MV*
    - Binding
    - Templating
  - Routing
  - Security
  - Components
    - Directives
  - HTTP services (including REST)
  - Animation
  - Testing/Mocking
  - Touch

AngularJs: Concepts

- Module (app)
- Controller
- Directive
- Service
- Dependency Injection
  - Uses CommonJS AMD like keys to define which modules Angular should resolve and then pass them

Code Demonstration

Basic Angular

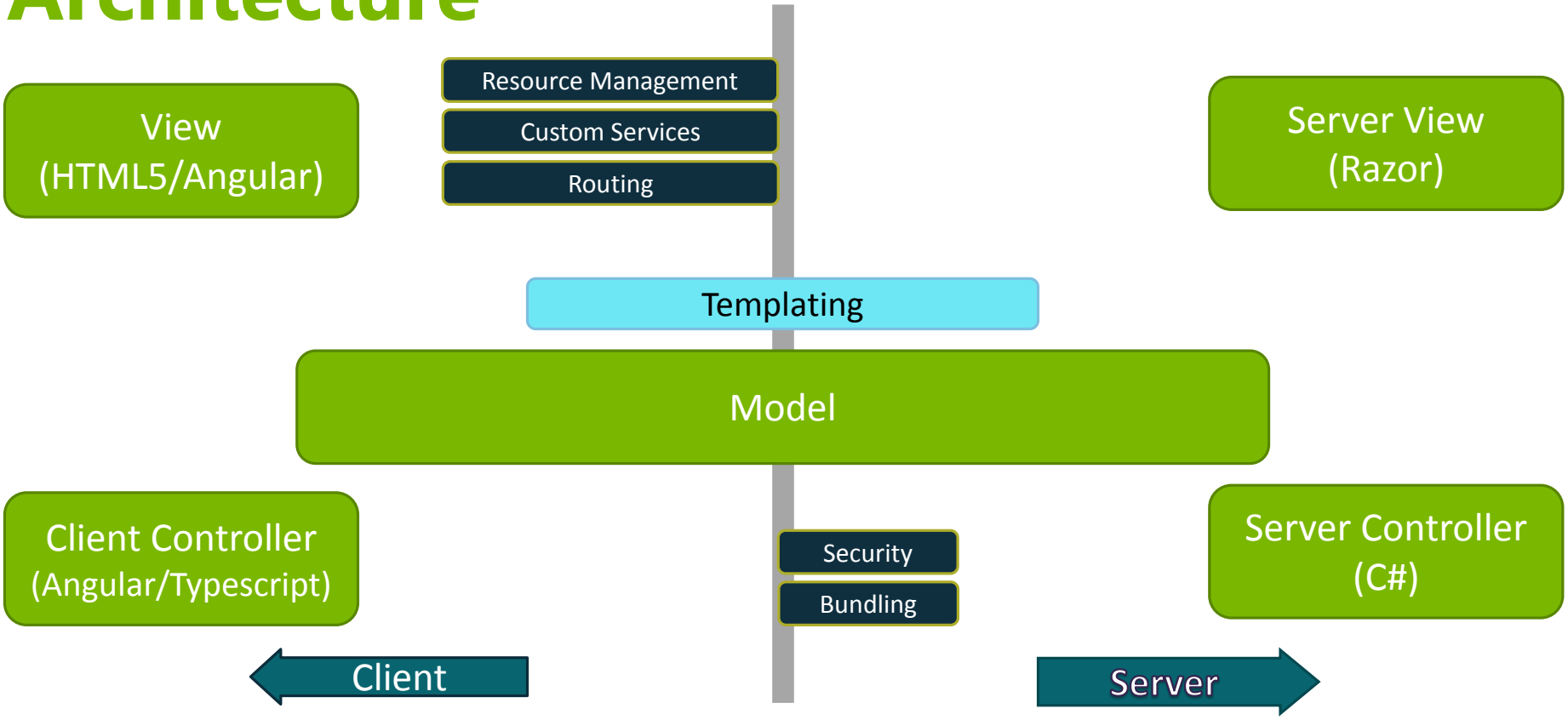
That's great! What does it mean?

We now have a framework that supports some of the concepts that, as enterprise app developers, we need to meet the challenges in creating a complex application.

Review of Technologies Used

- AngularJs
  - Included ui-bootstrap library
- Twitter Bootstrap 3.1.1
- Typescript
  - Definitely typed libraries for AngularJs
- MVC4/WebAPI
- Neo4J – Not really relevant to this talk, but it was a good learning exercise.

Architecture



Server Side from 10,000 feet

Server View (Razor)

Model

Server Controller (C#)

- Decide where your script files will be served from
- I like /App for my application and /Scripts for my library.
- Setup MVC controllers for your server side logic and views
- Optionally set up Web api for your data operations.
- Leverage your shared _Layout.cshtml for the majority of your page.
- Models are standard POCOs. Validation can be handled through data but it takes a little wiring to validate them server side.

Client Side

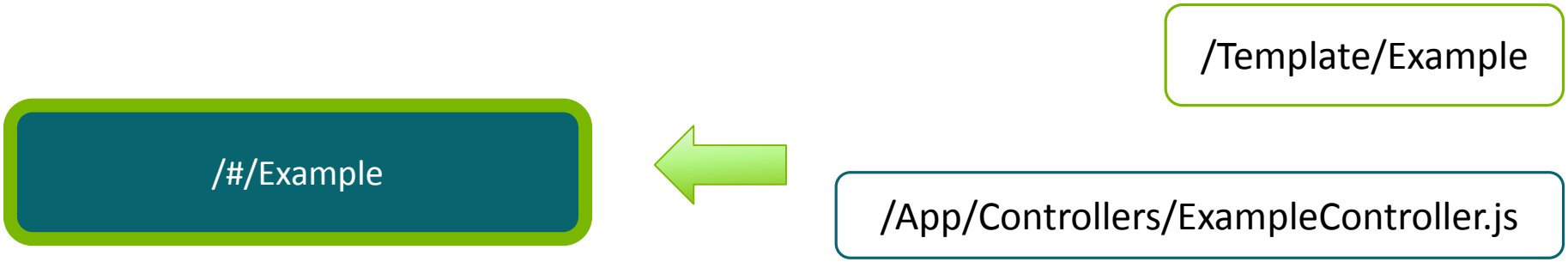
Client Controller
(Angular/Typescript)

Model

View
(HTML5/Angular)

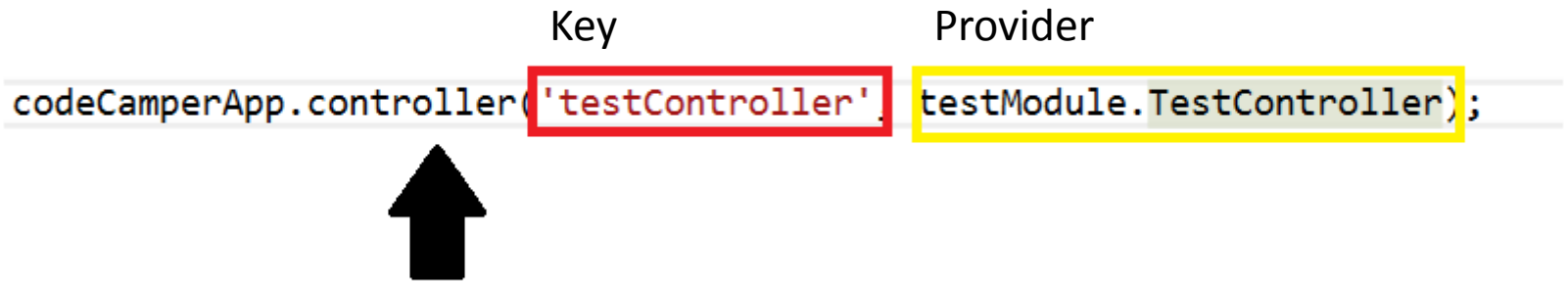
- Models take form in Hashes that are cast from interfaces in typescript that will pass back and forth, it is useful to use a T4 template to generate these interfaces.
- Client controllers have a view model mechanism (referred to as a scope) which we leverage to manage bindings on a page.
- Angular-Route can be used for routing and templating
  - In this app Templates are Razor partials and have a dummy controller to serve them.

Routing


/Template/Example
/#/Example
/App/Controllers/ExampleController.js

Building your catalog, defining objects

- Angular.js modules use a standard signature for most of its provider methods that allow it to build a catalog.



Each provider method has a way that it handles instantiation
When the value is requested during injection.

Controllers

- The “C” in MVC, on our client, a controller is the “Code behind” for a given section on a webpage. This is done by tag on the webpage, i.e. or by the controller property in the hash expected on a .when method on the routing service.
- Has a $scope object where we expose our variables and methods.
- Additional services injectable
- In typescript we can use the following syntax to declare our controller and associated scope.
- Notice the use of Generics here.

```
export interface ICommentControllerScope extends
  ng.IScope
{
    comments: Array<CommentObject>
}

export class CommentController {
    static $inject = ['$scope']

    constructor($scope: ICommentControllerScope)
{
    }
}
```

Building Services

- Creating a service class is very similar to creating a controller. However, you'll use one of several methods to create an entry in your module catalog.
  - Value
  - Factory
  - Provider
  - Constant
  - Depending on your use case you'll decide which one to use.
    - Most typically I use factory, it allows for an array object initializer

Special Methods on the Module

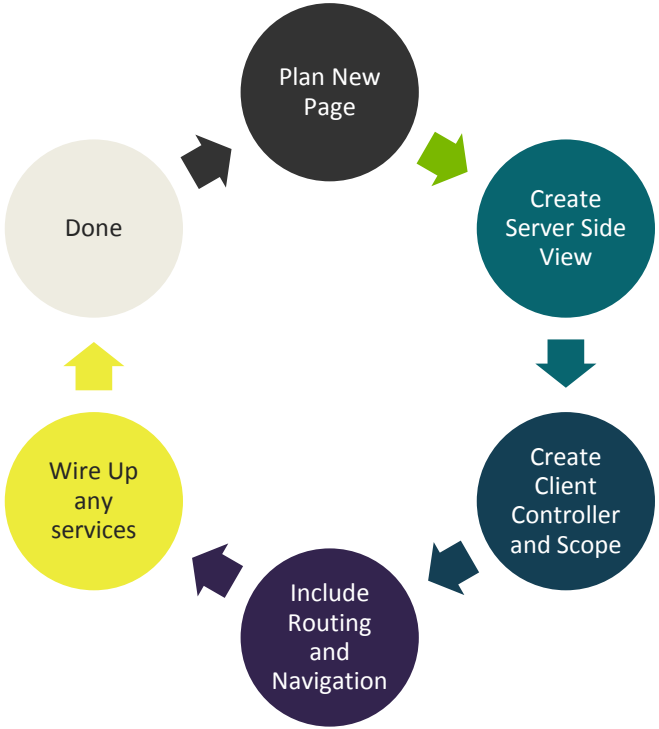
- Run
  - Follows the Angular DI recipe, a run method is run as an Initialization function.
    - I use this for initializing some of my services.
- Config
  - Also follows Angular DI recipe, Config is run as almost a constructor step.
  - This is typically where you'll provide your routing logic.

Angular and Typescript: Bootstrap

It’s an important step when working with Typescript, because we have to use two forms of Inversion of Control (require.js for Typescript, and angularjs.)

When loading the app, we load all our libraries, but we don’t immediately allow angular to resolve the app (ng-app) on our page. Rather, we use require to resolve all our dependencies, then “bootstrap” it in some logic at the global level.

Basic Development Workflow



- Now that we've compartmentalized our concerns We can simplify our development in to this rough workflow.

So much to cover, so little time

I hope to continue development on this app, and create a VSIX for the Angular/MVC/WebAPI template that I've developed.

Remember the angularjs site, and I will always try to answer email:

toddr@magenic.com, or faux.trot@gmail.com