BluePrint® SPA

Front-End Architecture

Overview

Technology Stack

Version: first draft - in progress

Author: Pegah Tabrizi

With Microsoft planning on retiring Silverlight in near future and browsers starting to drop their support for Silverlight already, we have to adopt open web standards such as HTML5/JavaScript and replace Blueprint’s current offering that had been developed using Silverlight and along the way add support for tablet and mobile.

A Web based single page application (SPA) is an evolution away from the stateless page-redraw model that browsers were originally designed for but advancements in HTML5 and new JavaScript frameworks, enable browser to retain a single page and create a fluid user experience similar to a desktop application even when the application requires server communication.

Modern browsers that can parse HTML5 allow developers to shift the user interface and related application logic from web servers to the client and JavaScript libraries such as AngularJS, ReactJS, Ember, Meteor, … have adopted single page application (SPA) principles. HTML5 web-sockets also provide a bidirectional real-time client-server communication when needed.

Because SPA architecture moves the logic from the server to the client, the role of the web server can also evolve into a pure data API or web services enabling the server to be built as reusable discrete polyglot micro-services that are highly scalable, highly available and can independently be deployed in cloud environments.

# JavaScript SPA Frameworks: AngularJS & TypeScript

With AngularJS popularity and a large open source community behind it, not to mention the backing of Google, Angular has also been Blueprint’s JavaScript framework of choice. There is some uncertainty however with next version of Angular being completely different than Angular 1.x and because of it we need to design our code in a way that we are prepared for the changes which can be categorized into the following:

* Template Syntax
* ES6 and Modules
* Types
* Components
* Bindings

In Angular 2, “components” are the main way elements and logic are built on the page, and Angular 1.x’s directives, controllers, and scope are all combined into components. Using TypeScript with Angular 1.x provides a more clear migration path to Angular2.x

TypeScript is superset of ES5 and it wraps ES6 so we can use all ES6 features and on top of that it adds types and annotations that come very handy in large projects such as ours and can improve JavaScript development experience by providing type information and displaying API documentation (“intellisense”) based on type definition files provided by “Definetely Typed” repo. Using TypeLite we can also create TypeScript definitions (interfaces/enums) from our .NET classes.

TypeScript is transpiled into JavaScript so our workflow need to include this step. We also need write our controllers and services as TypeScript classes.

Using TypeScript decorators is a neat feature that Angular 2 uses which could make Angular1 code look like Angular2 but I think more than worrying about the looks and syntax, we need to use an architecture that is aligned with Angular2 which means no $scope anywhere and fully componentizing our app and thinking more about how components communicate with each other.

Looking into Functional Reactive Programming / React&Flux&Immutable objects and using React.js with Angular could be also something that worth looking into.

TODO – frontend technology stack / tools / workflows