Declarative Web-Applications without Server

Demonstration of How a Fully Functional Web-Application can be Built in an Hour with only HTML, CSS & <angular/> JavaScript Library

Miško Hevery
GetAngular.com
(BRAT Tech, LLC)
misko@hevery.com

Adam Abrons
GetAngular.com
(BRAT Tech, LLC)
adam@getangular.com

Abstract

Common wisdom in web-application design states that web-applications need to have (1) a presentation layer (on the browser), (2) a business-logic layer (on the server), and (3) a database. In such a design, almost all of the application behavior is specified on the server. Now, imagine what an application would look like if the application layer was moved from the server to the browser? The resulting system would have the browser communicating directly with the database server. In this system the behavior of the application would need to be fully described in the HTML itself. By embedding declarative tags in HTML the once static HTML can now become an interactive application capable of both persistence, and intelligent behavior. The resulting design allows a web-designer (not versed in server side technologies and languages) to build interactive web applications in hours rather than months, which is currently the norm with more traditional methods.

Categories and Subject Descriptors

D.3.3 Software, PROGRAMING LANGUAGES, Language Constructs and Features – Frameworks

General Terms Algorithms, Design, Languages.

Keywords Web-Applications; HTML & CSS; JavaScript; Declarative; AJAX; Web-Designer; Database; Template

1. What is <angular/>

<angular/>, consists of a JavaScript client library which adds new capabilities to the existing browsers, allowing the web-designer to become a web developer. In addition, <angular/> is also a service which provides persistence storage for the <angular/> web-applications. An <angular/> application consists of a collection of static HTML files which act as templates for the application and contain application behavior in the form of <angular/> directives. These directives allow data-binding to take place, making it easy to tie an input field in the HTML to a property in the database. Each HTML template must have the <angular/> JavaScript framework loaded in order to take advantage of the behavior.

1.1 Goals

Simple: Building web-applications should be as simple as: 1) creating an HTML template, 2) adding <angular/> JavaScript to HTML and 3) declaring the name of the database entity in the HTML template. As a web-application creator you should not need to know much beyond the HTML & CSS. In contrast, traditional methods of building even the simplest web-applications requires knowledge of client: HTML, CSS, JavaScript, HTTP; server: Java / PHP / Python / Ruby; database: SQL, and can take weeks to build. The goal of <angular/> is to make simple web-applications, which require basic CRUD (Create, Read, Update, Delete) operations, trivial to build.

Declarative: HTML is simple because it is declarative. JavaScript (like any other programing language) is too complex for most web-designers because it is procedural in nature. For this reason <angular/> too is declarative, making it easy to learn.

Bookmark-able: The page URL should contain all of the necessary information needed to restore the HTML template view to the current state. The page URL updates automatically as the user modifies it, and saves the data to the database.

Any Look&Feel: There are many existing services which allow CRUD operations on data. However, most of them are hosted on closed environments which limited freedom to set the look&feel of the form, and are often not embeddable in existing pages. In contrast, <angular/> gives full control of the look&feel to the web-designer.

Embed-able: The user should have the freedom to host the HTML templates anywhere, even in an existing web application. This allows the user to add application-like behavior to existing content such as blogs, CMS (Content Management Systems) or even emails. This means that the location of the HTML template and the <angular/> database need not be on the same domain.

Audit-able: The business rules execute on the client and therefore, they can not be trusted. As a result, a full audit of all modifications to the data is needed to prevent abuse.

Extensible: <angular/> is extensible. If it does not fulfill your needs out of the box, you cans always use JavaScript to extend functionality. This allows free mixing with existing JavaScript libraries and frameworks.

1.2 Non-Goals

Because <angular/> business rules run on the client, the code can not be trusted, and as a result applications where audit data is not sufficient and require stricter data control are not a good match.

1.3 Life-Cycle

The best way to describe an <angular/> application is to look at a typical <angular/> application life-cycle:

1. Browser loads an HTML template (normal browser behavior)
2. The <angular/> JavaScript, which is included in the template HTML script tag, examines the HTML DOM and looks for <angular/> directives in the form of tag attributes or text bindings. The <angular/> directives describe the application visual widgets, behavior, form validation, database persistence, and URL encoding strategy.
3. The initialization directives are processed first, and are used to declare the database entities and load any initial data from the database using the URL key-value pairs, this initializes the application model. The model is the truth of the current state of the application.
4. The application enters bind phase, where HTML UI elements and Widgets are bound to application model. For all practical purposes the model and the HTML rendering of the model become one. This means that any changes to the model are reflected in HTML structure, and conversely any changes in the HTML structure (such as entering form data) are reflected in the model.
5. User may trigger a save which persists the model to the database.

2. HTML as Template

A simple persistable hello-world form is show with <angular/> directives bolded:

```
<html>
 <head>
  <script src="http://misko.getangular.com/
          angular-1.0a.js#database=oopsla">
 </head>
 <body ng-entity="greeting=Greeting">
  Your name:
  <input type="text" name="greeting.name"/>
  <input type="submit" value="Save"/>
  <br/>
  Hello, {{name}}!
 </body>
</html>
```

The above example displays a form which asks for the user's name. Upon entering a name, such as '`John`', into the input field it is immediately visible in the greeting '`Hello, John!`', When the user hits the '`Save`' button, the data is persisted to the <angular/> database under the entity 'Greeting', and the URL is updated with '`greeting=123`'. This allows you to bookmark the URL for later access or email. When the URL is reopened, '`John`' is loaded from the database and bound to the appropriate locations on the page.

3. Persistence, Versioning and Query

An application may need a database to store data. Each database contains entities which are declared in the HTML templet. The entity is a collection of JSON documents which are dynamic (schema free i.e. each document can have a different format). Each document is stored as a list of unmodifiable versions allowing for a full audit of all of the edits by different users.

4. Bindings and Expressions

Using the `{{expression}}` notation anywhere in HTML will cause the express to be evaluated during the bind phase of the application execution, and be displayed. An example of the bind expression `{{total = qty * cost}}` calculates a single row in the invoice and saves it in total, in addition to displaying it to the user.

5. Filters

Filters allow the conversion of the model data into renderable presentation. For example, if you want to split `$10,000` between three people, a simple expression such as `{{10000/3}}` will evaluate to `3333.33333`. However, if you would like to display `$3,333.33`, this can be achieved with a currency filter such as `{{10000/3|currency}}`. <angular/> comes with a variety of the most common filters used to display data to users.

6. Widgets

HTML form widgets fall short of most users' expectations for today's interactive web-application. For this reason, <angular/> includes additional widgets such as repeaters, date-pickers, barcodes, charts, maps, package tracking, and form validation. New behavior can easily be added through JavaScript.

7. Security

<angular/> has security and authentication built into the database access and may prompt the user to log in. The security model prevents unauthorized access to your data (<angular/> does not control access to the templates, as they can be embedded). When creating a database with <angular/>, read/write permissions can be applied to the database to manage user rights. The HTML template is on a different domain than the <angular/> database, therefore, <angular/> must also supports cross-site communication. However, this opens <angular/> up for cross-site attacks. For this reason, an additional list of hosting sites, which are authorized to talk to database, must be specified when creating a database storage for an <angular/> application.

8. Glass Ceiling

Very rich applications can be created with nothing more than HTML and <angular/> directives. Examples of this include: invoicing, company directory, job estimation, and task lists, to name a few. At some point, it may become necessary to go beyond what the <angular/> directives allow. In this circumstance, the application developer can take full advantage of the JavaScript and use it together with <angular/> to build even more complicated applications.