

x-project: a document-oriented toolkit to design and implement Web Applications based on HTML5 Web Components

Andrea D'Amelio
Università Roma Tre
Dipartimento di Ingegneria
Università Roma Tre
Rome, Italy
damelio@ing.uniroma3.it

Tiziano Sperati
Università Roma Tre
Dipartimento di Ingegneria
Università Roma Tre
Rome, Italy
sperati@ing.uniroma3.it

Enrico Marino
Università Roma Tre
Dipartimento di Ingegneria
Università Roma Tre
Rome, Italy
marino@ing.uniroma3.it

Federico Spini
Università Roma Tre
Dipartimento di Ingegneria
Università Roma Tre
Rome, Italy
spini@ing.uniroma3.it

ABSTRACT

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

CCS Concepts

•Information systems → Web applications; •Applied computing → Cartography; Format and notation; •Computer systems organization → Client-server architectures; Real-time system architecture;

1. INTRODUCTION

Intense work and researches around anatomy and operating of web applications have led to identify the operations that are identically performed by the (almost) total-

ity of them. It is essentially the case of procedures related to user and session management, data access policies and CRUD method on basic data models. Several software tools (e.g. KeystoneJS or LoopBack) are available nowadays to automatically handle these operations once a description of the data type to deal with (i.e. model schemas) has been provided. This approach is perfectly suitable to speed up web application development, mostly relying on external configuration files and less on procedural code [3].

Questa è solo l'ultima delle "automatizzazioni" che sono state introdotte nel corso degli anni. Da principio si è lavorato per facilitare la gestione del contenuto sul web, e diversi CMS di successo sono stati introdotti.

Il contributo principale presentato in questo articolo è costituito dalla individuazione di un processo di sviluppo web guidato dai documenti congiuntamente alla definizione di un toolkit che utilizzato come indicato di seguito permette l'effettivo utilizzo del processo identificato. Il web development cycle, è un processo definito in 4 fasi che può essere applicato ricorsivamente alle viste dell'applicazione web e a tutte le loro sottocomponenti. Il toolkit consiste di una libreria di Web Components che abilita allo sviluppo di SPA mediante la composizione (e parametrizzazione) di tags html.

questo tipo di sviluppo abilita ad un riuso estremo dei componenti della UI, ma anche per quanto riguarda nel caso di modelli di diffuso E/O FREQUENTE UTILIZZO.

dal momento che

lo sforzo di sviluppo deve concentrarsi nella definizione dei documenti che guidano lo sviluppo, ovvero lo schema dei modelli che descrivono i dati gestiti. Il toolkit fornirà quindi il supporto necessario alla gestione amministrativa dei modelli di dato definiti fornendo una interfaccia adeguata interfaccia.

The remainder of this document is organized as follows. In Section 2 we provide an overview of the proposed web development cycle process. Section 3 is devoted to describe the architecture and the technology stack exposed by applications developed with the toolkit, while section 4 presents the toolkit itself. Section 5 reports about a case-study appli-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DocEng2015 Sep 8–11, 2015, Lausanne, Switzerland

© 2015 ACM. ISBN 123-4567-24-567/08/06...\$15.00

DOI: 10.475/123_4

cation: it is shown how to build a CMS by means of the toolkit introduced in this paper. Finally, Section ?? proposes some conclusive remarks and future developments.

2. WEB DEVELOPMENT CYCLE

We model the web development process as a four-steps procedure that can be applied recursively to each page (or view) of a web application as well as to every single complex component (or widgets) of the page itself.

This modelization is based on the reasonable assumption that server side operation on data models are nowadays being sufficiently explored, and as proven by the *KeystoneJS* experience, at least one choice is available to automatically 1) generate server-side CRUD methods on models with ACL capabilities and 2) handle users and sessions, once a JSON description of data models and relations between them are provided to the system. This very JSON descriptor documents drive the whole process, actually composed by the following four steps.

1st step - JSON data model description. The JSON descriptors must be defined, specifying data type, relation, and user role read/write capabilities on particular portion of data.

2nd step - Model actions definition. Since CRUD operation could not be enough to describe all the needed operation further actions on models can be defined and exposed via http verbs.

3rd step - UI component definition. Then individual UI component can be defined, relying exclusively on CRUD operations and actions available on data models.

4th step - UI component assemblage. As last task, previously defined UI component have to be mounted to define application views. Assembly should be kept as simple as possible, in the case of x-project toolkit, it only consists of a juxtaposition of HTML5 tags.

3. ARCHITECTURE

Web applications developed exploiting x-project toolkit are full stack *JavaScript*.

On the server-side they rely on *Node.js*, exploiting the power of the *Loopback* framework by *Strongloop*. As mentioned above, the aim is to have a development process entirely document-driven, and those documents are the schemas of the models used by the application. These are JSON documents. Each document represents a model and presents the following fields: the **name** of the model, the set of **properties**, the list of **relations** to other models and the list of **ACL** (Access Control Layer) rules.

Loopback generates model's API from the models schemas, to let CRUD operations on models.

The API can be extended: the developer can add remote functions to models or add hooks to existing APIs to add behaviour before and/or after the API handler (to preprocess the request and/or postprocess the response).

The resulting API is RESTful, cookie free, signed by authentication token.

By default, applications have a built-in model that represents a user, with properties **username**, **email** and **password** for login and the property **role** used by the ACL module.

A very remarkable feature exposed by *Loopback* is that it abstracts from the particular DBMS utilized by the means of an indirection layer, allowing to choose the preferred one,

be it a noSQL or a graph-based one.

On the client-side, developed applications happen to be SPA (single page application) which exploit a variety of technologies, briefly described below.

Web Components are a collection of standards which are working their way through the W3C and landing in browsers at the moment. They allow to bundle markup and styles into custom HTML elements. *Custom Elements*[8], *HTML Imports*[9], *HTML Templates*[10], *Shadow DOM*[11]. Since these specifications are currently W3C Working Draft, they aren't fully supported across all major browsers, so as these technologies are implemented in browsers, the polyfills will shrink to gain the benefits of native implementations. [7]

Polymer library (<https://www.polymer-project.org/>) provides a thin layer of API on top of web components (native implementations and their polyfills) and several powerful features, such as custom events and delegation, mixins, accessors and component lifecycle functions, that makes it easier and faster to create Web Components. Similar to *Polymer* are *x-tag* and *Bosonic*. Web repositories <http://component.kitchen> and <http://customelements.io> already counts thousands of open source user-contributed custom elements.

4. X-PROJECT TOOLKIT

"Everything is an element", from an AJAX request to an entire web page. Every part of the website is encapsulated inside an element.

x-project provides a set of Polymer element for local routing, API requests, User management, forms composition, layout and style.

4.0.1 Elements for local routing

These elements can be used to perform local routing (for Single Page Application).

<x-router> implements local routing based on *HTML5 Push State API*.

<x-route> represents a route-to-page mapping. It has two input attributes: **route** and **page**. A route can be parametrized: parameters are sent as attributes to the corresponding page.

<x-link> is an extension of the anchor element **<a>** that prevents the default behavior when a click event occurs, blocking page request to the server and redirecting the request to the local router.

```
1 | <link rel="import"
2 |   href="/elements/page-posts.html">
3 | <link rel="import"
4 |   href="/elements/page-post.html">
5 |
6 | <x-router>
7 |   <x-route route="posts" page="posts">
8 |   <x-route route="posts/:id" page="post">
9 | </x-route>
```

4.0.2 Elements for API requests

These elements handle models API.

<api-collection-get> gets a collection of models.

```
1 | <api-collection-get name="{{name}}"
2 |   filter="{{filter}}">
```

```

3 | page="{{page}}" perpage="{{perpage}}"
4 | collection="{{items}}" schema="{{schema}}"
5 | count="{{count}}">
6 | </api-collection-get>

```

<api-collection-post> add a new model to the collection.

```

1 | <api-collection-post name="{{name}}"
2 |   model="{{model}}"></api-collection-post>

```

<api-collection-schema> retrieve a model schema.

```

1 | <api-collection-schema name="{{name}}"
2 |   schema="{{schema}}"></api-collection-schema>

```

<api-model-get> retrieve a model. **<api-model-delete>** delete a model.

```

1 | <api-model-get name="{{name}}"
2 |   model-id="{{model_id}}"
3 |   model="{{model}}" schema="{{schema}}">
4 | </api-model-get>

```

<api-model-put> retrieve a model.

```

1 | <api-model-put name="{{name}}"
2 |   model="{{model}}"></api-model-put>

```

4.0.3 Elements for lists and forms

These elements are used to create forms (even dynamically from a schema).

<x-input> is an extension of the input element. It's type can be **string**, **date**, **email**, **location**, **number**, **file**.

```

1 | <x-input type="{{type}}" label="{{label}}"
2 |   value="{{value}}"></x-input>

```

<x-form> generate dynamically (from a model schema) a form to create/update a model.

```

1 | <x-form schema="schema"
2 |   model="{{model}}"></x-form>

```

<x-filter> generate dynamically (from a model schema) a form to create an API filter.

```

1 | <x-filter schema="{{schema}}"
2 |   filter="{{filter}}"></api-filter>

```

<x-table> generate dynamically (from a model schema) a table of models.

```

1 | <x-table schema="{{schema}}"
2 |   collection="{{collection}}"></x-table>

```

4.0.4 Elements for layout and style

The style is based on **iron-flex-layout** [4], a CSS library of style mixins for cross-platform Flexible Box [12] layouts.

4.0.5 Admin pages

Client-side can be divided in two parts: **admin part** and **user part**.

The *Admin part* is automatically generated. It consists of the following pages: **<page-collections>**, **<page-collection>** and **<page-model-edit>**.

<page-collection> shows the model instances of a collection.

```

1 | <dom-module id="page-collection">
2 | <template>
3 |   <api-collection-get name="{{collection}}"
4 |     filter="{{filter}}"
5 |     collection="{{list}}">
6 | </api-collection-get>
7 | <x-filter schema="{{schema}}"

```

```

8 |   filter="{{filter}}"></x-filter>
9 | <x-table schema="schema"
10 |   list="{{list}}">
11 | </x-table>
12 | <x-paginator current="{{page}}">
13 | </part-paginator>
14 | </template>
15 | </dom-module>

```

<page-model-edit> shows the forms to update a model.

```

1 | <dom-module id="page-model-edit">
2 | <template>
3 |   <api-model-get name="{{collection}}"
4 |     model-id="model_id"
5 |     model="{{model}}" schema="{{schema}}">
6 | </api-model-get>
7 | <x-form
8 |   schema="schema" model="model">
9 | </x-form>
10 | <api-model-put name="{{collection}}"
11 |   model-id="{{model_id}}">
12 | </api-model-put>
13 | </page-model-edit>
14 | </dom-module>

```

The **user part** depends on the type of the Web Application that has been implemented. It is the part the final user interact with.

5. CASE STUDY

In this section we discuss the design and the implementation of a blog platform.

5.1 Models

For a blog platform the essential entities to model are: **Post** and **Tag**.

```

1 | {
2 |   "name": "Post",
3 |   "properties": {
4 |     "title": { "type": "string" },
5 |     "posted": { "type": "date" },
6 |     "content": { "type": "text" },
7 |     "permalink": { "type": "string" }
8 |   },
9 |   "relations": [{
10 |     "name": "tags",
11 |     "type": "has_many",
12 |     "model": "Tag"
13 |   }]
14 | }

```

```

1 | {
2 |   "name": "Tag",
3 |   "properties": {
4 |     "name": { "type": "string" }
5 |   }
6 | }

```

It results in the following HTTP RESTful API.

```

1 | GET|POST /api/Posts
2 | GET|PUT|DELETE /api/Posts/:post_id
3 | GET|POST /api/Tags
4 | GET|PUT|DELETE /api/Tags/:tag_id

```

5.2 Pages

<page-posts> show the list of posts. It is accessible via **/posts/** route.

```

1 | <dom-module id="page-posts">
2 | <template>
3 |   <api-collection-get name="Posts"
4 |     page="{{page}}" perpage="10"
5 |     collection="{{posts}}" count="{{count}}">
6 | </api-collection-get>

```

```

7 | <template is="dom-repeat" items="{{posts}}">
8 |   <div>
9 |     <h1>{{post.title}}</h1>
10 |     <h2>by <span>{{post.author}}</span></h2>
11 |     <h3>on <span>{{post.date}}</span></h3>
12 |   </div>
13 | </template>
14 | <x-paginator perpage="10" total="{{count}}"
15 |   current="{{page}}"></x-paginator>
16 | </template>
17 | </dom-module>

```

<page-post> show a post. It is accessible via /posts/:post_id route. The router pass the post_id parameter to the page.

```

1 | <dom-module id="page-post">
2 |   <template>
3 |     <api-model-get name="Posts"
4 |       model-id="{{post_id}}" model="{{post}}">
5 |     </api-model-get>
6 |     <h1>{{post.title}}</h1>
7 |     <h2>by <span>{{post.author}}</span></h2>
8 |     <h3>on <span>{{post.date}}</span></h3>
9 |     <div>{{post.content}}</div>
10 |   </template>
11 | </dom-module>

```

6. REFERENCES

- [1] G. T. Heineman and W. T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- [2] M. Nath and A. Arora. Content management system : Comparative case study. In *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, pages 624–627, July 2010.
- [3] V. Okanovic. Web application development with component frameworks. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*, pages 889–892, May 2014.
- [4] A. Osmani and E. Bidelman. iron-elements. Available at <https://github.com/PolymerElements/iron-elements>, 2015. [Online; accessed 15-May-2015].
- [5] S. Patel, V. Rathod, and S. Parikh. Joomla, drupal and wordpress - a statistical comparison of open source cms. In *Trendz in Information Sciences and Computing (TISC), 2011 3rd International Conference on*, pages 182–187, Dec 2011.
- [6] A. Repenning, A. Ioannidou, M. Payton, W. Ye, and J. Roschelle. Using components for rapid distributed software development. *Software, IEEE*, 18(2):38–45, Mar 2001.
- [7] Z. Rocha. Web components polyfills. Available at <http://webcomponents.org/polyfills/>, 2015. [Online; accessed 15-May-2015].
- [8] W3C. Custom elements. Available at <http://www.w3.org/TR/custom-elements/>, 2014. [Online; accessed 15-May-2015].
- [9] W3C. Html imports. Available at <http://www.w3.org/TR/html-imports/>, 2014. [Online; accessed 15-May-2015].
- [10] W3C. Html templates. Available at <http://www.w3.org/TR/html-templates/>, 2014. [Online; accessed 15-May-2015].
- [11] W3C. Shadow dom. Available at <http://www.w3.org/TR/shadow-dom/>, 2014. [Online; accessed 15-May-2015].
- [12] W3C. Css flexbox. Available at <http://www.w3.org/TR/css3-flexbox/>, 2015. [Online; accessed 15-May-2015].
- [13] W3Techs. Usage of content management systems for websites. Available at http://w3techs.com/technologies/overview/content_management/all/, 2015. [Online; accessed 15-May-2015].
- [14] L. Wu, R. De Matta, and T. Lowe. Updating a modular product: How to set time to market and component quality. *Engineering Management, IEEE Transactions on*, 56(2):298–311, May 2009.