# 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

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

Tiziano Sperati
Università Roma Tre
Dipartimento di Ingegneria
Università Roma Tre
Rome, Italy
sperati@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

One generally accepted definition of Content Management System is: a system that lets you apply management principles to content.

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

It is simple to elicit an evolutionary path in the history of management systems whose milestones are identifiable in *Joomla!*, *Wordpress* and *KeystoneJS*.

Alongside these milestones entire constellations of analogous experiences popped up, but we considered them not relevant since they borrowed main features and constitutive approaches from cited ones.

Starting from *Joomla!*, a framework that drove in the engineering into the world of web content management. Joomla! powers more than 2,7% of the largest 1,000,000 web sites in the world [13]. Anyway, nowadays, *Joomla!* results unwieldy and, due to its monolithic approach, not complied to current web features.

*Wordpress*, instead is used by more than 23.3% of the top 10 million websites (as of January 2015) [13]. Wordpress develop CMS's idea, driving in the intention to use CMSs to build Web Application. Wordpress, with its plugin, aims to limber user experience. The availability of more than 37,000 plugins, because it lets to create sites to non-experts too. Anyway, further customizations, other than the ones introduced by plugins, are difficult to deploy due to loosely code engeneering of this tool.

Finally, *KeystoneJs* stand in the last position. Minimal and agile, KeystoneJS, embody the new era of the CMS, letting the user to make his own personal web application.

In the spite of focusing on very minimal features, a mention goes to Ghost, a non-profit, open source blogging Node.js based platform, which represents a transversal experience, although minimal brings a new light way to blogging.

Eventually, `x-project`, for the reasons the will be clear in the remaining of this paper, could be thought as the last standing in this path of decreasing monolithic approach and increasing customization and code engineering.

To speed up web application development, frameworks mostly rely on external configuration files and less on procedural code [3].

The remainder of this document is organized as follows. In Section **??** we provide an overview of the `x-project` architecture. Section **??** is devoted to describe the advances in-

troduced by the `x-project` architecture. Finally, Section ?? reports a case study using the toolkit

## 2. WEB DEVELOPMENT CYCLE

We model the web development process as a four-steps procedure that can be applyied 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 modellization is based on the reasonable assumption that server side operation on data models are nowadays be sufficently explored, and as proven by the *Keystone.js* experience, at least one choice is available to automagically 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 definion**. Since CRUD operation coul not be enougth 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 assemplation**. 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 present are full stack *JavaScript*.

On the server-side they rely on *Node.js*, exploiting the power of the *Loopback* framework by *Strongloop*. As mention 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 others 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 represent a user, with properties `username`, `email` and `password` for login and the property `role` used by the ACL module.

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].

**webcomponent.js polyfills** enable Web Components in (evergreen) browsers that lack native support. Web Components specifications are currently W3C Working Draft, so they aren't fully supported across all major browsers. 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*.

**iron-elements** [4] is a library of utility Polymer elements from ajax requests to input elements. There are web repositories like http://component.kitchen and http://customelemer io that already counts thousands of open source user-contributed custom elements.

### 3.1 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` provide a set of Polymer element for local routing, API requests, User management, forms composition, layout and style.

#### 3.1.1 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.

```
<link rel="import"
  href="/elements/page-posts.html">
<link rel="import"
  href="/elements/page-post.html">

<x-router>
  <x-route route="posts" page="posts">
  <x-route route="posts/:id" page="post">
</x-route>
```

#### 3.1.2 Elements for API requests

These elements handle models API.

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

```
<api-collection-get name="{{name}}"
  filter="{{filter}}"
  page="{{page}}" perpage="{{perpage}}"
  collection="{{items}}" schema="{{schema}}"
  count="{{count}}">
</api-collection-get>
```

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

```
<api-collection-post name="{{name}}"
  model="{{model}}"></api-collection-post>
```

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

```
<api-collection-schema name="{{name}}"
  schema="{{schema}}"></api-collection-schema>
```

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

```
<api-model-get name="{{name}}"
  model-id="{{model_id}}"
  model="{{model}}" schema="{{schema}}">
</api-model-get>
```

`<api-model-put>` retrieve a model.

```
<api-model-put name="{{name}}"
  model="{{model}}"></api-model-put>
```

### 3.1.3 *Elements for lists and forms*

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

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

```
<x-input type="{{type}}" label="{{label}}"
  value="{{value}}"></x-input>
```

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

```
<x-form schema="schema"
  model="{{model}}"></x-form>
```

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

```
<x-filter schema="{{schema}}"
  filter="{{filter}}"></api-filter>
```

`<x-table>` generate dinamically (froma a model schema) a table of models.

```
<x-table schema="{{schema}}"
  collection="{{collection}}"></x-table>
```

### 3.1.4 *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.

### 3.1.5 *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.

```
<dom-module id="page-collection">
<template>
  <api-collection-get name="{{collection}}"
    filter="{{filter}}"
    collection="{{list}}">
  </api-collection-get>
  <x-filter schema="{{schema}}"
    filter="{{filter}}"></x-filter>
  <x-table schema="schema"
    list="{{list}}">
  </x-table>
  <x-paginator current="{{page}}">
  </part-paginator>
</template>
</dom-module>
```

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

```
<dom-module id="page-model-edit">
<template>
  <api-model-get name="{{collection}}"
    model-id="model_id"
    model="{{model}}" schema="{{schema}}">
  </api-model-get>
  <x-form
    schema="schema" model="model">
  </x-form>
  <api-model-put name="{{collection}}"
    model-id="{{model_id}}">
  </api-model-put>
</page-model-edit>
</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.

# 4. CASE STUDY

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

## 4.1 Models

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

```
{
  "name": "Post",
  "properties": {
    "title": { "type": "string" },
    "posted": { "type": "date" },
    "content": { "type": "text" },
    "permalink": { "type": "string" }
  },
  "relations": [{
    "name": "tags",
    "type": "has_many",
    "model": "Tag"
  }]
}
```

```
{
  "name": "Tag",
  "properties": {
    "name": { "type": "string" }
  }
}
```

It results in the following HTTP RESTful API.

```
GET|POST /api/Posts
GET|PUT|DELETE /api/Posts/:post_id
GET|POST /api/Tags
GET|PUT|DELETE /api/Tags/:tag_id
```

## 4.2 Pages

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

```
<dom-module id="page-posts">
<template>
  <api-collection-get name="Posts"
    page="{{page}}" perpage="10"
    collection="{{posts}}" count="{{count}}">
  </api-collection-get>
  <template is="dom-repeat" items="{{posts}}">
    <div>
      <h1>{{post.title}}</h1>
      <h2>by <span>{{post.author}}</span></h2>
      <h3>on <span>{{post.date}}</span></h3>
    </div>
  </template>
  <x-paginator perpage="10" total="{{count}}"
    current="{{page}}"></x-paginator>
</template>
</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.

```
<dom-module id="page-post">
<template>
  <api-model-get name="Posts"
    model-id="{{post_id}}" model="{{post}}">
  </api-model-get>
  <h1>{{post.title}}</h1>
  <h2>by <span>{{post.author}}</span></h2>
  <h3>on <span>{{post.date}}</span></h3>
  <div>{{post.content}}</div>
</template>
</dom-module>
```

## 5. 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.