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

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.

Section 2 presents the context where `x-project` is positioned, overviewing the CMS evolution; section 3 introduces `x-project`, its philosophy and architecture. Section 4 shows a case study in which is implemented a blog using x-project toolkit. Finally, section 5 draws conclusions.

DocEng2015 Sep 8–11, 2015, Lausanne, Switzerland

DOI: 10.475/123_4

2. EVOLUTION OF CMSS

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

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.

3. X-PROJECT TOOLKIT

`x-project` is a set of libraries and API-centric HTML5 Web Components based Web Application and CMS framework.

"Everything is an element", even an AJAX request. Every part of the website is encapsulated inside an element. Even functional components such as HTTP services (AJAX calls), pages and routing logic are encapsulated inside elements. We push Web Components' philosophy beyond all limits so that users can reach a very high level of customization and ease of use.

Component-based Software Engineering denotes the process of building software by (re)using pre-built software components. The main benefit of component-based software is the "time-to-market" [14], thus reducing the cost of developing the software. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies. Additionally, a component is a software element that conforms to a component model and can be independently deployed and composed [1]. Component based software is developed interconnecting building blocks, therefore, a high degree of reusability and modularity is achieved by this type of applications [6].

3.1 Architecture

`x-project` architecture is client-server. The full technology stack is JavaScript based.

Server-side is based on Node.js.

Node.js (http://nodejs.org) is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.

Express.js (http://expressjs.com) is a minimal and flexible Node.js web application framework that provides a set of features for web and mobile applications.

Loopback (http://loopback.io) is a set of Node.js modules to develop Web Servers applications. An application interacts with data sources through the LoopBack model API, available locally within Node.js, remotely over REST.

MongoDB (https://www.mongodb.org/) is an open-source document database that provides high performance, high availability, and automatic scaling.

Client-side is based on Web Components.

Web Components are a collection of standards which are working their way through the W3C and landing in browsers at the moment. In a nutshell, they allow us 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://customelemen io that already counts thousands of open source user-contributed custom elements.

3.2 Server-side

The Web Application development is document-driven.

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

In `x-project`, server-side, the documents that drive the development are the schema of the models of the application.

These are json document. Each document represent a model and has 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.

`x-project` (using Loopback) generates model's API from the models schemas, to let CRUD operations on models (e.g. a model `Author` that describe a blog author generates the following HTTP RESTful API: `GET /api/authors/` to get all authors, `POST /api/authors/` to add a new author, `GET /api/authors/:author_id` and `PUT /api/authors/:author_id` to get (or to update) the author with id "`author_id`". Since an blog author have blog posts (its model has a relation one-to-many to model `Post`) there are also the API to handle author's posts: `GET /api/authors/:author_id/posts`, etc.

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 API are RESTful, cookie free, signed by authentication token.

`x-project` 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. By default, only user with role `admin` can create/update/delete models in the applications.

3.2.1 Third party services

`x-project` is designed to be connected to microservices. These tools are accessible extending the Web Application API.

An essential feature of a CMS is media storage (documents, images and videos) [2].

`x-project` provides a remote storage service implemented as a direct upload module to AWS Amazon S3 storage service. The media to upload don't pass through the server but are sent directly to AWS using a signed request. The signature of the request is provided by the module. The module has to be configured with the AWS bucket settings: bucket

name, public and secret keys.

3.3 Client-side

Client-side can be divided in two parts: `admin part` and `user part`. `admin part` is automatically generated from the models schemas. It let the admin to manage (via CRUD operations) the models. `user part` depends on the type of the Web Application that has been implemented. It is the part the final user interact with.

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

3.3.1 Elements for local routing

`x-project` provide elements to perform local routing to implement Single Page Application pattern.

`<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 the extension of the native anchor element (`<a>`). It prevents the default behavior, blocking page request to the server and letting to manage the routing locally.

3.3.2 Elements for API requests

`x-projects` provide a set of elements to handle collections and models API.

`<api-collection-get>` retrieve a collection of models. It has input attributes to handle filters and pagination and an output attributes that stores the collection.

`<api-collection-post>` add a new model to the collection. It has an input attribute that refers to the model to save.

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

`<api-model-(get|put|delete)>` retrieve, update or delete a model. These have an input attribute that refers to the model.

For example, the following element perform an `HTTP GET` request to `/api/Posts` using filter and returning the first 10 items of the second page and the total number of items (helpful for pagination element).

```
<api-collection-get name="Posts"
  perpage="10" page="2" filter="{{filter}}"
  collection="{{posts}}" count="{{count}}">
</api-collection-get>
```

3.3.3 Elements for User management

`x-project` provides a set of elements to manage User. `<x-login>` is a form (with `email` and `password` input elements) that let the user login in the application. Once logged in, every API request is signed with a token that refers to the current user, for ACL reasons.

3.3.4 Elements for forms composition

`x-project` provides a set of input elements to create forms. `<x-input>`, `<x-textarea>`, `<x-number>`, `<x-date>`, are respectively associated to short text, long text, number, and date; `<x-location>`, based on Google Place APIs, allows to choose locations using autocomplete add-on; `<x-file>`, based on the `x-project` direct upload module, allows to upload files. `<x-form>`, given a model schema, generates a form to edit models that match that schema; it is a composition of the input elements.

3.3.5 Elements for layout and style

`x-project` provides a set of elements for layout, such as: `<x-page>`, `<x-header>`, `<x-navbar>`, `<x-sidebar>`, `<x-footer>`, respectively to define the structure of the page, to add header, navbar, sidebar and footer.

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

4. CASE STUDY

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

4.1 server-side

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

`Post` model (defined below) represent a blog post.

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

`Tag` model represent a tag in a post.

`Author` model represent a blog author. It has properties that describe an author, such as `full_name`, and one `has_many` relation to `Post` model.

4.2 client-side

Client-side pages are encapsulated in elements that extend `<x-page>` element. These pages can be divided in two parts: Admin and User.

4.2.1 Admin part

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

`<page-collections>` is the main page. It show the collections of the app. In this case, these are `Authors`, `Posts` and `Tag`.

`<page-collection>` show the model instances of a collection.

```
<page-collection>
  <api-collection-get
    name="{{collection_name}}"
    filter="{{filter}}"
    collection="{{collection}}">
  </api-collection-get>
  <part-collection-filter
    name="{{collection_name}}"
```

```
    filter="{{filter}}">
  </part-collection-filter>
  <part-list
    list="{{collection}}">
  </part-list>
  <part-paginator
    list="{{list}}"
    filter="{{filter}}"
    current="{{page}}">
  </part-paginator>
</page-collection>
```

`<page-model-edit>` presents the forms to update a model. The form is automatically generated from the model schema. This page is composed by an `<api-model-get>` element that retrieve the model to edit. The model schema (retrieved with the model) is passed to a `<x-form>` element that presents an input element for each property of the model. The type of the input element corresponds to the type of the property (e.g. a *boolean* property is editable via a `<x-checkbox>` element).

4.2.2 User part

The *User part* must be designed by the developer. It consists of the following pages: `<page-author>`, `<page-posts>` and `<page-post>`.

`<page-posts>` show the list of posts. It use `<api-collection-get>`, `<x-paginator>`, and `<x-list>` to retrieve, paginate and list the posts.

`<page-post>` show a post. It is accessible via `<x-route path="/posts/:post_id" page="page-post">`. It use `<api-model-get>` to retrieve the post using the `post_id` parameter matched in the url.

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

5. CONCLUSION

In the present time open source CMS has gained a big market. Generally all CMSs fulfill common task of content like create, edit, publish. Lots of varieties are available based on functionality and platform, such as *good user support*, *security aspects*, *plug-ins ecosystem*, *documentation*, etc.

[5] propose different performance criteria like *page load time*, *page size*, *number of request*, *number of bootstrap resources files*.

[2] propose about thirty features criteria that could be classified in: *Admin Management*, *Data Management*, *User Management*, *UI Management*, *Web Content Management*, *Multimedia Data Management*.

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.