2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)

Light-weight and Scalable Hierarchical-MVC Architecture for Cloud Web Applications

Meng Ma[1], Jun Yang[2], Ping Wang[1,4,3,*], Weijie Liu[4] and Jingzhuo Zhang[4]
[1] National Engineering Research Center for Software Engineering, Peking University
[2] Baidu, Inc.
[3] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
[4] School of Software and Microelectronics, Peking University
{mameng, pwang, dataliu, zhangjingzhuo}@pku.edu.cn, harttle@baidu.com

***Abstract*—Nowadays, with the prevalence of cloud computing, the demand for modular and scalable Web application development technologies is urgent. Dynamic contents and ubiquitous user interactions make Web applications increasingly complicated. A majority of current web applications leverage a Model-View-Controller (MVC) architectural style. Since the MVC triad does not provide feature-based modularization, Web applications in pure MVC style are experiencing scalability and maintainability issues. In this paper, we propose a light-weight and scalable hierarchical-MVC architecture for Web application development in Cloud environment, named Web Module Definition (WMD), which supports feature-based modularization and application structure. In WMD, the entire Web application is decomposed into interconnect WMD modules, which contains controllers and views for a single feature. WMD modules are able to include and extend others to handle complex business logic. In the meantime, we provide a Web application framework implementation supporting WMD-based architecture, and present a demonstration website using WMD.**

***Index Terms*—Web Application, Cloud Computing, MVC, Modularization**

I. Introduction

Since 1994 when the Internet became available to the public, it has become a platform for a large number of sophisticated Web applications [1]. The increasing complexity of Web applications requires more structured application architectures. In the era of cloud computing, modular and lightweight development framework can effectively improve the efficiency of Web application development. Traditionally, the most commonly used architecture is the MVC (Model-View-Controller) triad [2]. However, the absence of feature-based modularization in MVC leads to poor scalability, especially for cloud Web pages development. Meanwhile, back-end and front-end components of Web applications are organized separately in most frameworks. Comparing to the application back-end, the front-end components are rarely organized or well-structured. Therefore, maintainability issues will emerge when Web applications become more complex.

In this paper we present a Web Module Definition (WMD) based HMVC (Hierarchical-Model-View-Controller) architecture to support high-efficiency feature-based modularization in cloud Web applications. WMD defines the structure and dependency of Web modules. A Web module is an independent program unit for a single feature. Web modules can be either deployed independently, or composed to another Web module. Thus, WMD-based architecture provides better scalability and maintainability in both back-end and front-end components. The contributions of our work are threefold:

- We summarize existing Web application architecture styles including MVC and its variants. Web front-end modular approaches in open-source communities are also been analyzed. The limitations of the traditional MVC triad are identified and the merits within HMVC design pattern are discussed in this work.
- We propose a Web Module Definition (WMD) to specify Web application units supporting feature-based modularization. We then present a WMD-based HMVC architecture for Web applications to provide better scalability and maintainability.
- We provide a WMD-compliant Web Application Framework (WAF) implementation for the proposed architecture. We provide a demonstration prototype developed under WAF, and discussions about its scalability and maintainability.

The rest of this paper is organized as follows: In section II, we summarize Web application architecture styles and modular approaches. In section III and IV, we propose the WMD-based Web application architecture and corresponding implementation. Section V provides a demonstration Web application developed under WAF. Section VI concludes the paper and discusses the future works about WMD.

II. Related Works

A. The MVC Design Pattern

The MVC (Model-View-Controller) design pattern was first described by Krasner and Pope in the Smalltalk system [3]. MVC is often used to implement software with user interfaces. It divides the entire application into three interactive parts, as in Fig. 1 [4]:

- Model is the underlying data structures of the information represented in GUI.
- View is the representation within the GUI. Multiple Views for the same information are possible.

*Corresponding author: *Ping Wang (pwang@pku.edu.cn).*

978-1-7281-1661-7/19/$31.00 ©2019 IEEE
DOI 10.1109/CSCloud/EdgeCom.2019.00017

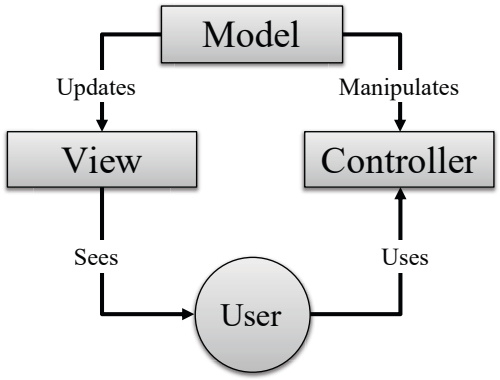

Fig. 1. Model-View-Controller Design Pattern.

- Controller is responsible to receive operations from user (the client) and convert them into operations on Model and View.

One of the key benefit of MVC design pattern is the separation of information representation and underlying data structures. This supports multiple views for the same information, without changing the underlying data structures. As information representation (View) is unaware of logic defined in controllers, views could be reused in different controllers. Hence, MVC is also capable for developing web applications in cloud infrastructure [18].

Based on the concept of Model-View-Controller separation, MVC-styled Web Application Frameworks tend to organize models, controllers and views separately. While the controllers (and views) for different application features are organized in same namespace. This leads to a point of failure when application features scale out.

B. The Variants of MVC

Today almost all Web Application Frameworks are claimed to support MVC design pattern. The traditional pure MVC is not suitable for Web environment since HTTP is stateless and Web applications are forced to be partitioned into client-side and server-side. To alleviate this problem, a variety of variants MVC structure are proposed:

MTV (Model-Template-View) [5] is widely adopted in Web Application Frameworks, such as Django [6]. Model is responsible for underlying data structures and database interaction. View is similar to the Controller in traditional MVC, accept HTTP requests, do business logic, and respond views to the requests. Template is an abstraction of views, used to render dynamic HTML

MVVM (Model-View-ViewModel) [7] is first developed by Ken Cooper and Ted Peters from Microsoft for .NET WPF and Silverlight. Today MVVM is applied to Web front-end MVC frameworks, such as AngularJS [8]. The most creative part of MVVM design pattern is the ViewModel, which is an abstraction for View states and behaviors. With the help of ViewModel, the underlying architectures are able to implement data binding between Models and Views.

HMVC (Hierarchical-Model-View-Controller) [9] is also called Layered MVC. With HMVC architectures, the entire Web application is break down into interactive MVC sub-structures, as shown in Fig. 2. HMVC is one of the responsibility-based architectures, in which different application features are divided into different sub-structures. In contrast of cloud-based MVC [18], HMVC is more powerful in handling the complexity of cloud Web applications, especially for data management, user interaction, and control flows. Thus HMVC design pattern can provide better scalability than traditional MVC.

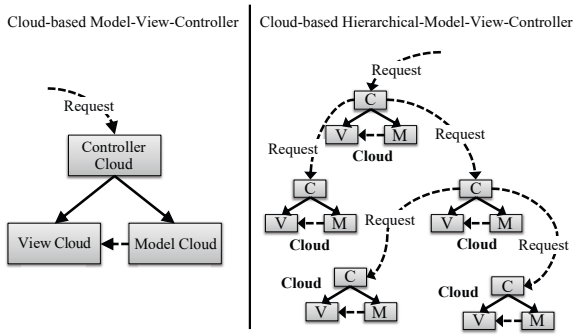

Fig. 2. Differences between cloud-based MVC and cloud-based HMVC design pattern.

C. Web Front-end Modularization

Web frontend (client-side) is composed by HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript [10]. Most of these programming languages are not designed for modular or object-oriented programming. As a result, various tools and pre-compile languages are emerging to help with the problems of pure HTML, CSS and JavaScript.

For HTML, most of the Web Application Frameworks provide one or more template systems to implement HTML modularization. A Web template system use a template engine to combine tem-plates with context data into a HTML file. The context data usually comes from Models and the out-put of business logic in the Controllers. Template systems allow Web application to define reusable HTML partials, and its layouts. In terms of application features, partials and layouts can only provide a limited level of modularization. Since they are compiled by template systems before passed to Controllers, which means the Controller is still responsible for the whole compiled HTML page.

For CSS, processors like LESS [11] and SASS [12] are proposed, devoted to modular (or even object-oriented) language extension to CSS. As for JavaScript, there're module definitions like CommonJS [13] and AMD (Asynchronous Module Definition) [14] to provide file-level JavaScript modularization.

Despite of these tools and approaches, Web frameworks tend to leave CSS and JavaScript modularization to the developer. This is originated from the traditional thin-client design philosophy which is widely adopted by Web Application

Frameworks, which means models and controllers are defined in the server, and views are defined in the client (typically, the browser). With the increasing complexity of application features, pure back-end architectures won't scale well.

D. Component-based Web Application Frameworks

Since developing a holistic Web application using MVC leads to potential maintainability and scalability issues, component-based Web Application Frameworks emerged to help building scalable Web applications without pain. In component-based Web Application Frameworks, developers take care of a set of application components instead of the whole application in traditional MVC frameworks. Notable component-based Web Application Frameworks includes Flask Blueprint [15] and Kohana [16] framework for PHP. All these frameworks have their own ways to construct or extend an application, thus provide a lot of inspiration in how to architect scalable Web applications.

Blueprint [15] is a concept in Python Flask framework for making application components and supporting common patterns within an application or across applications. Blueprint applies coarse grained modularization to Web applications. A component is a sub-application with its own view functions, templates, and static assets registered in one or more URL paths. A controller (view function) in Blueprint are still responsible for a whole Web page, scalability issues within complex feature-rich Web pages remains unsolved.

Kohana framework [16] is a HMVC styled Web Application Framework for PHP. Kohana propose the concept of Cascading Filesystem to achieve responsibility-based modularization. Within HMVC frameworks like Kohana, one controller is responsible for exactly one component in the requested Web page, and there's a hierarchy of controllers (or components) for one HTTP request. HMVC is powerful for building scalable Web applications, while existing HMVC Web frameworks rarely take CSS and JavaScript modularization into account.

III. WMD-based Web Application Architecture

In order to solve scalability issues associated with controllers and views in complicated Web applications, and provide better maintainability for front-end components, we propose the concept of Web Module Definition (WMD) and a WMD-based Web application architecture.

A. Goals

The goals of the WMD-based Web application architecture include:

- **Scalability.** The architecture should support large numbers of components (Web Modules), or interactions among them. With this architectural feature, building complex Web applications can be painless.
- **Maintainability.** The architecture should simplify modifications of the deployed Web application including fault correction and performance improvement. This could be done by strict modularization of Web Modules, and provide unified modularization for both back-end and front-end components.
- **Simplicity.** Simplicity has become a key feature for Web application frameworks since the success of Ruby on Rails. In the meantime, the increasing number of Web Modules should not lead to significant performance degradation.

B. Elements and Configurations

The WMD-based Web application architecture include four part of elements (as shown in Fig. 3):

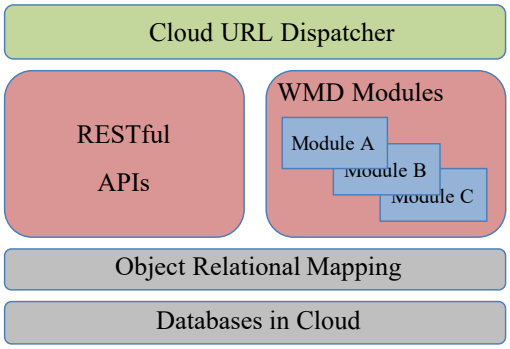

Fig. 3. WMD-based Web Application Architecture.

- **URL dispatcher.** URL dispatcher is the entry of the entire cloud Web application. It's responsible for dispatch incoming HTTP request to corresponding APIs or WMD root modules.
- **RESTful APIs.** Representational state transfer (REST) [17] is a Web architectural style proposed by R. T. Fielding. REST is proved successful in building HTTP APIs. HTTP API is essential in modern Web applications, as the emerging of AJAX technology and client-side Apps.
- **WMD Modules.** Responsibility-based Web modules that comfort WMD can be used to build scalable feature-rich Web pages. Each WMD module should contain all elements needed to act as a complete Web component, typically including: view functions, templates, style sheets, and client-side scripts.
- **Databases and Object Relational Mapping (ORM)** correspond to the Model in traditional MVC triad. In MVC design pattern, Models are unaware of Views and Controllers, which means Mod-els are decoupled naturally in architectures comfort MVC. ORM is a common middleware in Web applications developed in Object-Oriented language. ORM makes it possible to include in each entity attribute that is responsible for the physical location of data in distributed cloud databases.

Fig. 3 illustrates the configuration of the architectural elements above. URL Dispatcher is a unified entry for incoming HTTP requests in Web applications. Based on URL, the HTTP request is dis-patched to RESTful APIs or one of the WMD

root modules. RESTful API has been a common practice in Web community.

The key design point is the separation between APIs and WMDs, which significantly reduces the architectural complexity. APIs respond JSON or plain text to HTTP requests. WMDs respond HTML (the whole Web page) or HTML partials (a single Web component) with associated resources to HTTP requests. It's possible for WMD modules to include or extend others, this could be done by enforce HMVC-styled rendering and CSS/JS modularization. Fig. 4 depicts how WMD modules interact with others within a single Web page:

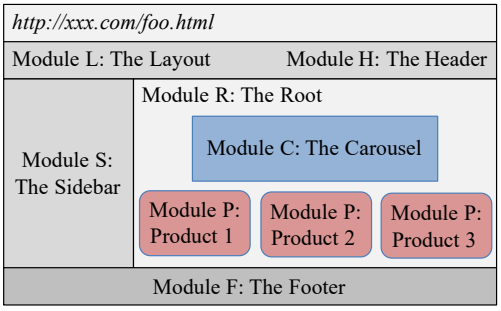

Fig. 4. A Sample Web Page Built with WMD Modules.

The URL *http://xxx.com/foo.html* is dispatched to a root WMD module R. Module R extends Module L for the layout, includes Module C for the carousel, and include Module P multiple times for a product list. Module L is a regular WMD module extended for a layout which includes Module H, S, and F.

The Web Modules R, L, C, P, H, S, and F are used as Web page components. When extended (Module L) or included (Module C, P, H, S, and F), the corresponding controllers should be called, style sheets should be applied accordingly, and client-side scripts should be correctly loaded and executed.

IV. Web Module Definition

Web Module Definition (WMD) is a specification that defines the structure and dependencies of a Web module. The entire application is decoupled into a set of Web modules. Each module can be developed, tested, and deployed independently.

A. The Structure of WMD Module

A WMD module is an independent Web component consists of controllers, templates, style sheets, and client-side scripts. The four parts of the WMD module along with the ORM layer form a MVC triad based on cloud infrastructure, as shown in Fig. 5.

Controllers in each module are responsible for the business logics and template context resolving of the current component. A Web page may contain multiple components hierarchically, thus it also may contain multiple controllers. Templates in each module is used to render the HTML for the module in the context resolved by corresponding controllers. Style sheets and client-side scripts are modularized with front-end modularization specifications introduced in Section II. C.

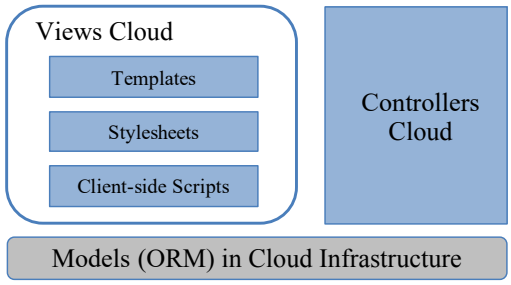

Fig. 5. MVC Triad Composed by A WMD Module and the ORM Layer in Cloud Infrastructure.

B. WMD Dependency Tree

WMD modules are Web components designed to work together. A WMD module is able to include or extend another. The relationship between WMD modules forms a dependency tree (WMD dependency tree), in which cyclic paths shall not exist. For instance, the WMD dependency tree of the WMD modules in Fig. 4 is shown as Fig. 6. The dependencies are classified into includes and extends.

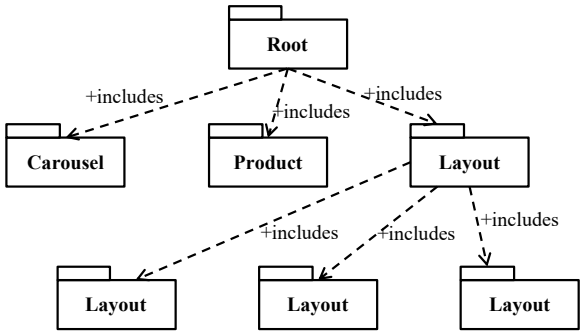

Fig. 6. The Dependency Tree of WMD Modules.

Web Application Frameworks that support WMD modules (WMD Frameworks) shall resolve the WMD dependency tree, and load WMD modules recursively.

C. Module Loading

A WMD module need to be loaded when it's included, extended (as dependency module), or dis-patched by the URL dispatcher (as root module).

WMD modules shall be loaded within a specific context for template rendering. For root modules, the context is null or an empty object. For dependency modules, the context should be bounded to the parent context where the current module is included or extended. The controller of the WMD module is executed immediately after parent context is resolved. The context resolved by the controller is merged into parent context before the template rendering.

Web applications today tend to use CDN to serve these static assets in production environment. In practice, style sheets and client-side scripts of all modules should be modularized and

concatenated into a single file respectively. The modularization for Style sheets could be done by class prefixing with corresponding WMD module identifier. Client-side scripts can be modularized with JavaScript Module Definitions like AMD (Asynchronous Module Definition) and CMD (Common Module Definition). In addition, client-side script need to be loaded according to the presence of WMD modules within the current Web page. Generally, a WMD-based JavaScript module loader is needed.

V. Implementation

In this section, we present an implementation of Web Application Framework supporting WMD-based architecture. This WMD Framework (named Brick.JS) is an open source project implemented in Node.js, hosted on Github: https://github.com/brick-js/brick.js, and available on NPM: https://www.npmjs.com/package/brick.js.

Brick.JS adopts a plugin architecture to support different template engines and CSS/JS processors. The core of Brick.JS implements WMD parser, URL dispatcher, and recursive WMD render. To achieve these, an open layered architecture is designed for Brick.JS core, as shown in Fig. 7.

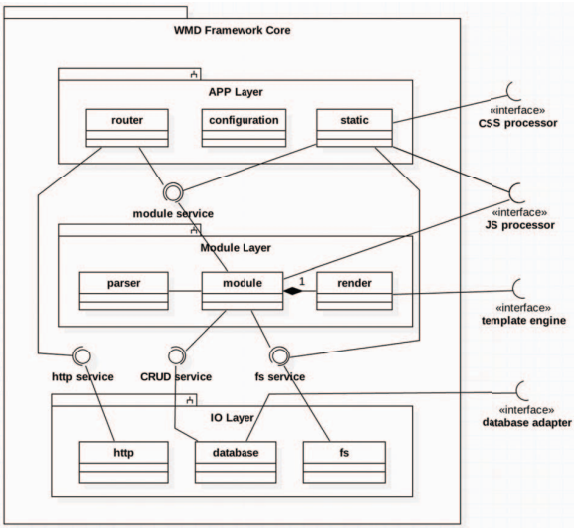

Fig. 7. Open Layered Architecture of Brick.JS.

The IO Layer implement HTTP, database, and file system services via Node.js API. The Module Layer is the key component for WMD module parsing and loading, WMD dependency tree resolving, and recursive rendering in runtime. There have been several plugins implemented for Brick.JS:

- Handlebars Template Engine: https://github.com/brick-js/brick-hbs
- Liquid Template Engine: https://github.com/brick-js/brick-liquid
- LESS processor: https://github.com/brick-js/brick-less

The demonstration Web application using Brick.JS is the website for the Intelligent Computing and Sensing Laboratory, Peking University. The source code is available on Github project: https://github.com/smart-sensing-lab/ics.pku.edu.cn

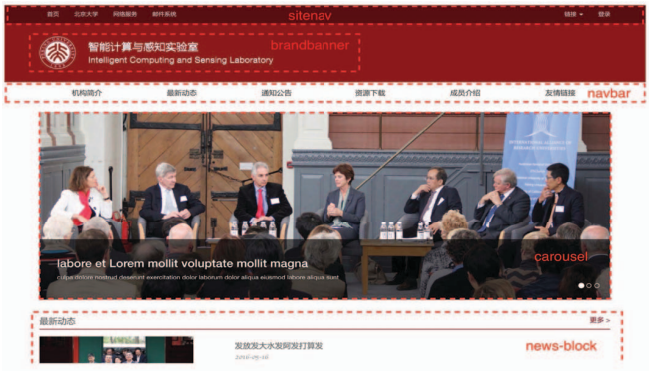

Fig. 8. WMD Modules in Homepage for http://ics.pku.edu.cn

The homepage for ics.pku.edu.cn and its corresponding page contents (marked by red-bordered rectangles) is shown as Fig. 8. This page consists of eight WMD modules, the dependency tree of which is shown as Fig. 9. The dependency tree is consists of a single root module, a layout module, and six component modules. The root module home extends from *default-layout* module. The home module *includes carousel*, *news-block*, and *navbar* modules as component modules. The default-layout includes *brandbanner*, *sitenav*, and *footer*.

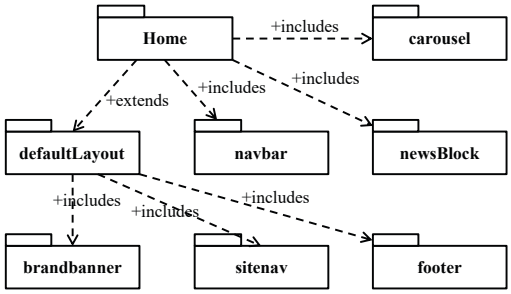

Fig. 9. The Dependency Tree for the Homepage.

In total, there are 36 WMD modules in this demo Web application, including three layout mod-ules, 19 root modules, and 14 component modules. In Brick.JS, it's possible for a WMD module to contain no template files or controllers. It's useful for modules providing common JavaScript utilities or style set. The *utils*, *styles*, and *form* are WMD modules in such type.

VI. Conclusion and Future Work

Feature-based architecture applies separation of concerns and structural principles to achieve better scalability and maintainability of Web applications in clouds. In this paper, we propose Web Module Definition to describe what a reusable Web component should be composed by. A WMD module is a self-assembled structure containing both front-end and back-end parts. The WMD-based Web application architecture gives

a prototype of component-based scalable Web applications. In our implementation, Brick.JS, the WMD de-pendency tree is resolved locally. Generally, dependencies should be resolved and downloaded via network to support collaborative development and distributed deployment. Our future works include the improvement of dependency infrastructure, as well as the implementation of dependency management tools.

ACKNOWLEDGMENT

This work is supported in part by National Key R&D Program of China No.2016YFB0800603, No.2017YFB1200700, National Natural Science Foundation of China No.61701007, and National Key Laboratory of Science and Technology on Reliability and Environmental Engineering No.6142004180403.

REFERENCES

[1] Jazayeri, Mehdi. "Some trends in web application development." Future of Software Engineering, 2007. FOSE'07. IEEE, 2007.

[2] Wojciechowski, J. , et al. "MVC model, struts framework and file upload issues in web applications based on J2EE platform." Modern Problems of Radio Engineering, Telecommunications & Computer Science, International Conference IEEE, 2004:342-345.

[3] Krasner, Glenn E., and Stephen T. Pope. "A description of the model-view-controller user inter-face paradigm in the smalltalk-80 system." Journal of object oriented programming, 1988:26-49.

[4] Wikipedia. "Model-View-Controller." available online: https://en.wikipedia.org/wiki/Model-view-controller (accessed on 19 March 2019).

[5] Adrian Holovaty, et al., "The Django Book Chapter 5: Models." available online: https://djangobook.com/ (accessed on 19 March 2019).

[6] Django Software Foundation. "The Web Framework for Perfectionists with Deadlines." available online: https://www.djangoproject.com/ (accessed on 19 March 2019).

[7] Smith, Josh. "PATTERNS-WPF Apps with The Model-View-ViewModel Design Pattern." MSDN magazine 2009:72.

[8] Google. "AngularJS – Superheroic JavaScript MVM Framework." available online: https://angularjs.org (accessed on 19 March 2019)

[9] Cai, Jason, Ranjit Kapila, and Gaurav Pal. "HMVC: The layered pattern for developing strong client tiers." Java World. 2000: 07.

[10] W3C. "Web Design And Applications." available online: https://www.w3.org/standards/webdesign/ (accessed on 19 March 2019).

[11] The core LESS team. "Getting Started — Less.js" available online: http://lesscss.org (accessed on 19 March 2019).

[12] Hampton Catlin, Natalie Weizenbaum, Chris Eppstein." Sass: Syntactically Awesome Style Sheets" available online: http://sass-lang.com/ (accessed on 19 March 2019).

[13] Kevin Dangoor. "CommonJS: JavaScript Standard Library." available online: http://www.commonjs.org (accessed on 19 March 2019).

[14] AMDJS. "AMD amdjs/amdjs-api Wiki." available online: https://github.com/amdjs/amdjs-api/wiki/AMD (accessed on 19 March 2019).

[15] Armin Ronacher. "Modular Applications with Blueprints." available online: http://flask.pocoo.org/docs/0.10/blueprints/ (accessed on 19 March 2019).

[16] Kohana Team. "Kohana: The Swift PHP Framework." available online: https://kohanaframework.org (accessed on 19 March 2019).

[17] Fielding, Roy Thomas. "Architectural styles and the design of network-based software architectures." Dissertation of University of California, Irvine, 2000.

[18] Lin, Jyhjong, Lendy Chaoyu Lin, and Shiche Huang. "Migrating web applications to clouds with cloud-based MVC framework." 2016 International Symposium on Computer, Consumer and Control (IS3C). IEEE, 2016.