Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 141 (2018) 454–459

Procedia Computer Science

www.elsevier.com/locate/procedia

The 4th International Workshop on Adults Use of Information and Communication Technologies in Healthcare (auICTH 2018)

Return of the JS: Towards a Node.js-Based Software Architecture for Combined CMS/CRM Applications

Fabian Kaimer[a], Philipp Brune[a]

[a]*Neu-Ulm University of Applied Sciences, Wileystraße 1, 89231 Neu-Ulm, Germany*

Abstract

While the use of server-side JavaScript in combination with the Node.js framework for implementing web applications is getting more and more common in practice, its implications for the evolution of web application architectures have rarely been studied in the scientific literature. In particular, the combination of components and their interplay for building pure JavaScript business applications has only rarely been investigated so far. Therefore, in this paper a software architecture for a real-world online service network application with a combined CMS/CRM functionality is presented. It is evaluated by a prototypical implementation of relevant core functionalities. Results indicate the feasibility and potential of the approach.

Keywords: Web Development, JavaScript, Node.js, Software Architecture, CMS, Service Networks

1. Introduction

When Ryan Dahl introduced Node.js in 2009, the interest in server-side utilization of JavaScript increased noticeably - in business as well as in science. Node.js, which is running on Google's V8 JavaScript implementation, introduced the concept of a non-blocking I/O-eventing model, working on a single thread instead of using multiple threads, as known from traditional web-servers. This single-threaded event loop leads to an improved consumption of hardware resources and a significantly higher possible number of concurrent client connections, making Node.js a considerable alternative for server-side web application development [1].

Apart from the performance-related benefits, the usage of JavaScript for advanced application frontends is widely common nowadays. Thus, using it for backend-development too, massively simplifies the whole process of web-application development. Additionally, Node.js comes with its own package manager "npm", which enables the dis-

* Corresponding author. Tel.: +49-731-9762-1503 ; fax: +49-731-9762-1599.
E-mail address: Philipp.Brune@hs-neu-ulm.de

tribution and uncomplicated installation of third-party libraries and their dependencies to extend existing development projects [2].

Considering all advantages Node.js, it is not surprising that many big players from various industries, like Uber [3], PayPal [4] or Netflix [5], have adopted it for their own projects.

However, even though many modern web-applications are built upon a Node.js architecture, only few results about respective architectural approaches have been published, in particular from a scientific perspective. The existing scientific literature mainly either deals with performance-related issues of Node.js or evalutes actual software-projects, which usually exhibit a low level of abstraction when it comes to Node.js architectural considerations.

Therefore, in this paper an examplified Node.js-based software architecture for a real-world web application is proposed and evaluated by means of a proof-of-concept implementation.

The rest of this paper is organized as follows: In section 2 the relevant literature is analysed in detail. The project contextof the study is presented in section 3, and the proposed application architecture is described in section 4. The architecture is evaluated by a proof-of-concept implemenation illustrated in section 5. We conclude with a summary of our findings.

2. Related Work

Research on Customer Relationship Management (CRM) systems has shifted in focus during the last years. Concerning non-business but tech-related aspects of CRM applications, especially the interest in cloud-based solutions has increased [6, 7]. With respect to architecture design for CRM systems, many web-based approaches exist [8, 9, 10], but none of them make use of server-side JavaScript or Node.js.

The major scientific interest in (Web) Content Management Systems (CMS) existed in the late 1990s and early 2000s [11, 12]. More recent literature on CMS focuses on performance-related [13] and security-related issues [14, 15], as well as on architectural design approaches [16, 17]. Nevertheless, none of the publications proposes a Node.js- or JavaScript-based approach.

The number of peer-reviewed publications concerning Node.js and server-side JavaScript has increased in recent years. Though, the majority of researches focuses on technical aspects of the platform, like performance and scalability [18, 19, 20] or security [21, 22, 23], rather than investigating its actual capabilities in building web-applications.

Some authors address with the use of Node.js in actual projects [24, 25, 26], but they mainly focus on the projects and their implications itself [27], rather than describing the software architecture built on Node.js. Publications which actually propose or discuss potential web-application architectures on a meta-level are scarce [28]. Proposals for such architectures in a CRM or CMS application context do not exist yet.

Therefore, this paper adresses the question how an example state-of-the-art server-side JavaScript- and Node.js-based software architecture for a real-world Online Service Network Platform with a combined CMS/CRM functionality could be designed, by example of the so-called FISnet platform[1].

3. Project Context and Application Description

The main idea behind the FISnet application is to offer a web-based platform, where service-providers and service-consumers (being in the so-called transition age of 55-75 years) of health-related services are brokered and matched, either based on a precise search, performed by the service-consumer or automatically by a complex matching algorithm based on the service-consumer's specific needs and self-improving machine learning[2]. Figure 1 illustrates this functionality.

The first actual core functionality of the platform - besides the matching algorithm - is the presentation of the service-providers and their services. FISnet offers a CMS-like administration interface, which allows providers to comfortably enter their services into the application and publish a wide variety of additional multi-media information about them on a profile-based offer platform [28].

[1] http://www.fisnet.info
[2] http://www.fisnet.info

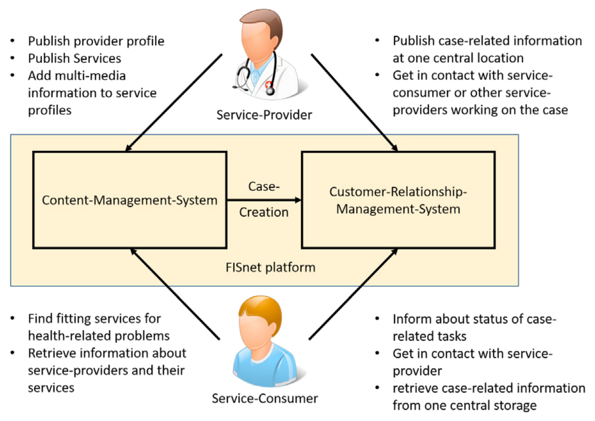
Publish provider profile
Publish Services
Add multi-media information to service profiles
Service-Provider
Publish case-related information at one central location
Get in contact with service-consumer or other service-providers working on the case
Content-Management-System
Case-Creation
Customer-Relationship-Management-System
FISnet platform
Find fitting services for health-related problems
Retrieve information about service-providers and their services
Service-Consumer
Inform about status of case-related tasks
Get in contact with service-provider
retrieve case-related information from one central storage

Fig. 1. The FISnet platform’s functional structure

When a service-consumer has chosen an appropriate offer, he may send a request to the provider, respectively providers in terms of a composite service, to make use of the respective offer. As soon as the request is accepted by all parties involved, a so-called case is generated that holds the service-provider(s) as well as the service-consumer and all service-related information. The case represents a centralized basis for all participants to communicate, share files, information and status concerning the subtasks of the overall service provision. This case-based service handling strongly resembles a CRM application. To fulfil the FISnet platform’s purpose, it is indispensable that both of the different application types - a CMS application as well as a CRM application - are interwoven with each other [28].

Several application requirements have already been deviated from the user group’s specific needs, which influence the platform’s actual architectural design:

- **REQ1:** A UI in particular suitable for elderly people in the transition age, usable on desktop as well as mobile devices [29]
- **REQ2:** A more enterprise-like desktop UI for the service providers and professional network coordinators
- **REQ3:** A web-based application to provide easy and ubiquitous access to the platform and avoid client-side software installation" [28]]
- **REQ4:** A high level of security and data integrity to ensure that the sensitive health-related customer data is conserved and protected

4. Software Architecture Design

Figure 2 illustrates the proposed architecture approach for the FISnet application, based upon modern JavaScript development frameworks for front-end- as well as for back-end development. As FISnet is such a highly customized hybrid application, using or adapting existing out-of-the-box back-end CMS or CRM solutions like Keystone.js[3], Ghost.js[4] or Enduro.js[5] was not an option.

Among the requirements for an appropriate framework were flexibility, scalability, security, extendability and a high degree in process automatization, like integrated object-relational-mapping (ORM) or command-line-interface-based API generation. One framework that meets all of these requirements is Sails.js. Sails.js is a lightweight server-side MVC framework, based upon Express.js. It offers an integrated ORM module called "Waterline", with several different database interfaces, as well as an integrated "Blueprint API", providing automated generation of routes,

[3] http://keystonejs.com/
[4] https://ghost.org/de/
[5] https://www.endurojs.com/

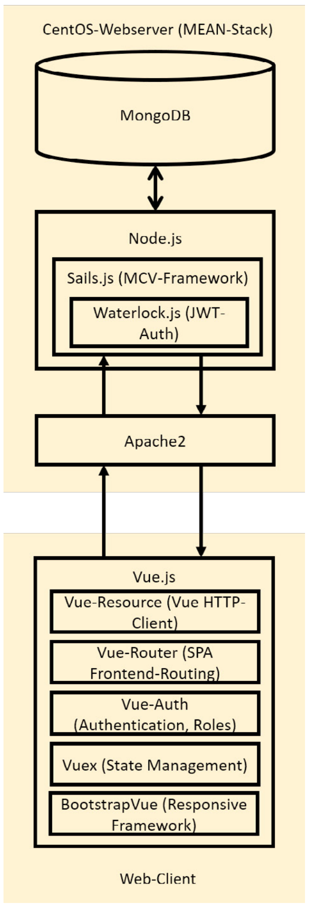
CentOS-Webserver (MEAN-Stack)
MongoDB
Node.js
Sails.js (MCV-Framework)
Waterlock.js (JWT-Auth)
Apache2
Vue.js
Vue-Resource (Vue HTTP-Client)
Vue-Router (SPA Frontend-Routing)
Vue-Auth (Authentication, Roles)
Vuex (State Management)
BootstrapVue (Responsive Framework)
Web-Client

Fig. 2. Overview of the proposed Node.js-based software architecture

controllers and controller methods for basic CRUD functions for all models created with the sails command line interface[6].

The FISnet application is built upon a MEAN-Stack on a CentOS 7 webserver [REQ3], which also runs the platform's affiliated MongoDB database. The preference for persistent data storing was given to MongoDB, as NoSQL databases have proven to have a higher data throughput, an improved performance and scalability and feature an easier key-value-storage system, in comparison to traditional relational databases like MySQL or MariaDB [REQ4] [30].

To meet the application's security requirements, Sails is extended by the Waterlock.js library, a user authentication tool built upon the concept of JSON web tokens (JWT). It enables server-side access restriction for unauthenticated users and Social-Login single-sign-on (SSO) [REQ4].

Additionally, an Apache2 webserver[7], operated as a reverse proxy, is superposed to the application, serving as a gatekeeper to all incoming and outgoing HTTP requests. Thereby, Apache's advanced security mechanisms, and its ability to serve separated static HTML files in case of a failure of the Sails application increase the application's security and performance even further [REQ4].

As the architectural approach needs to support a high level of application flexibility and short reaction- and feedback-durations, the best way to create the platform's frontend is to follow a single-page-application (SPA) architecture, built upon a JavaScript front-end framework, like Angular, React.js, Vue.js or Backbone.js [REQ1][REQ2].

[6] https://sailsjs.com/
[7] http://httpd.apache.org/

Vue.js was found to be highly suitable for building the FISnet application, as it features a modular development approach, a large number of external libraries and is more lightweight than Angular or React.js.

It is extended by Vue-Resource, an HTTP-client for standardized communication with the application back-end and Vuex, an application-overarching state management tool. Front-end authentication is handled by Vue-Auth, in JWT-based synchronization with Sails' Waterlock library, additionally providing role-based front-end access restriction in combination with Vue-Router. Vue-Router is a SPA front-end routing package for Vue, that enables the application to intercept and process URL calls at the front-end level instead of directly redirecting them to the application back-end.

Finally, Vue.js uses the BootstrapVue library, which represents a bootstrap adaption replacing jQuery by Vue-based components (e.g. modals, animations or front-end validation). BootstrapVue provides a large framework of modular, pre-built front-end components which ideally qualify for assembling the platform's CMS-based offer frontend and its CRM dashboard, optimized for service-providers and network-coordinators [REQ2].

5. Proof-of-Concept Implementation

To demonstrate the actual functionality of our architectural design, a full-fledged front–to-back-end communication implemented for a "form-wizard", which provides rapid input form construction and data submission.

The wizard consists of interactive form inputs, featuring front-end validation both on the basis of BootstrapVue. The form submission only succeeds when the user is authenticated (a valid Vue-Auth JWT token is present for back-end communication and a Vuex-"auth" parameter is set to true to allow access to the form (if restricted) in combination with VueRouter).

The form definition is transferred to the wizard in the calling parent element as a JSON object. Besides the actual input definitions and their validation rules, the server-side target route for data submission with VueResource is defined in that JSON data. The server-side controllers, safeguarded by Waterlock, only allow to receive and process data on their model's automatically built Blueprint CRUD functions or custom controller functions if a valid JWT token is present. Finally, the data is passed to MongoDB for persisting it.

Even though the application is still under development and a validation of the overall CRM/CMS-hybrid concept is still pending, this working form-wizard component already proves the overall feasibility of our architectural approach and the successful interplay of the chosen components.

6. Conclusion

In conclusion, in this paper an example architecture design approach for a Node.js- and server-side JavaScript-based application has been presented, evalusted by a proof-of-concept implementation in the context of the CMS/CRM-hybrid FISnet service network platform.

The proposed architecture provides a secure and performant Vue.js-based SPA frontend in combination with a light-weight and well-structured Sails.js-based application backend. The selected solution has proven to be a useful and functional basis for building the underlying platform or develop other similiar applications upon it.

Further research is needed for a more comprehensive and holistic evaluation of the given approach (also in different contexts). In particular, the complete FISnet application still needs to be implemented and evaluated in comprehensive lab and field tests.

Acknowledgements

This research received funding from the German Federal Ministry of Education and Research (BMBF) under grant code 01FR14021 / 02K12B033. The responsibility for the content of this publication lies with the authors.

References

[1] Node.js;. Available from: https://nodejs.org/en/.
[2] Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing. 2010;14(6):80–83.

[3] Lozinski L. The Uber Engineering Tech Stack, Part I: The Foundation; 2016. Available from: https://eng.uber.com/tech-stack-part-one/.
[4] Harrell J. PayPal Engineering, editor. Node.js at PayPal; 2013. Available from: https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/.
[5] Xiao Y. Node.js in Flames; 2014. Available from: techblog.netflix.com/2014/11/nodejs-in-flames.html.
[6] Manchar A, Chouhan A. Salesforce CRM: A new way of managing customer relationship in cloud environment. In: 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE; 2017. p. 1–4.
[7] Stephen WY, Cheng KL, Choy HY. An intelligent cloud-based customer relationship management system to determine flexible pricing for customer retention. In: 2016 Portland International Conference on Management of Engineering and Technology (PICMET). IEEE; 2016. p. 633–641.
[8] Vokorokos L, Hurtuk J, Cajkovsky M. Customer relationship management system proposal with emphasis on simplicity and security using distributed system architecture. In: 2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI). IEEE; 2014. p. 63–66.
[9] Wan S, Paris C, Georgakopoulos D. Social Media Data Aggregation and Mining for Internet-Scale Customer Relationship Management. In: 2015 IEEE International Conference on Information Reuse and Integration. IEEE; 2015. p. 39–48.
[10] Yuanyuan L, Guangjie C. Developing and Designing the Customer Relationship Management System of Small and Medium Enterprise Based on ActiveX Data Object Technology. In: 2012 Fifth International Conference on Intelligent Computation Technology and Automation. IEEE; 2012. p. 524–527.
[11] Barnes S, Goodwin S, Vidgen R. Web Content Management. BLED 2001 Proceedings 47. 2001;Available from: https://aisel.aisnet.org/bled2001/47.
[12] Jablonski S, Meiler C. Web-Content- Managementsysteme. Informatik-Spektrum. 2002;25(2):101–119.
[13] Mirdha A, Jain A, Shah K. Comparative analysis of open source content management systems. In: Krishnan N, editor. 2014 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). Piscataway, NJ: IEEE; 2014. p. 1–4.
[14] Contu CA, Popovici EC, Fratu O, Berceanu MG. Security issues in most popular content management systems. In: 2016 International Conference on Communications (COMM). Piscataway, NJ: IEEE; 2016. p. 277–280.
[15] Jerkovic H, Vranesic P, Dadic S. Securing web content and services in open source content management systems. In: Biljanovic P, editor. 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Piscataway, NJ: IEEE; 2016. p. 1402–1407.
[16] Islami RP, Mulyanto A. Component design of business process web content management system for online shop website. In: Proceedings of 2014 International Conference on Data and Software Engineering (ICODSE). Piscataway, NJ: IEEE; 2014. p. 1–8.
[17] Partheeban N, SankarRam N. e-Learning management system using web services. In: International Conference on Information Communication and Embedded Systems (ICICES), 2014. Piscataway, NJ: IEEE; 2014. p. 1–7.
[18] Lei K, Ma Y, Tan Z. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. In: 2014 IEEE 17th International Conference on Computational Science and Engineering. IEEE; 2014. p. 661–668.
[19] Chaniotis IK, Kyriakou KID, Tselikas ND. Is Node.js a viable option for building modern web applications? A performance evaluation study. Computing. 2015;97(10):1023–1044.
[20] Ogasawara T. Workload characterization of server-side JavaScript. In: 2014 IEEE International Symposium on Workload Characterization (IISWC). Piscataway, NJ and Piscataway, NJ: IEEE; 2014. p. 13–21.
[21] Brown F, Narayan S, Wahby RS, Engler D, Jhala R, Stefan D. Finding and Preventing Bugs in JavaScript Bindings. In: 2017 IEEE Symposium on Security and Privacy - SP 2017. Piscataway, NJ: IEEE; 2017. p. 559–578.
[22] Ojamaa A, Düüna K. Security Assessment of Node.js Platform. In: Venkatakrishnan V, Goswami D, editors. Information systems security. vol. 7671 of Lecture Notes in Computer Science. Berlin: Springer; 2012. p. 35–43.
[23] Pfretzschner B, Othmane Lb. Dependency-Based Attacks on Node.js. In: 2016 IEEE Cybersecurity Development. Piscataway, NJ: IEEE; 2016. p. 66.
[24] Cheng R, Scott W, Ellenbogen P, Howell J, Roesner F, Krishnamurthy A, et al. Radiatus. In: Aguilera MK, editor. Proceedings of the Seventh ACM Symposium on Cloud Computing. New York, NY: ACM; 2016. p. 237–250.
[25] Triglianos V, Pautasso C. Asqium: A JavaScript Plugin Framework for Extensible Client and Server-Side Components. In: Cimiano P, Frasincar F, Houben GJ, Schwabe D, editors. Engineering the Web in the Big Data Era. vol. 9114 of Lecture Notes in Computer Science. Cham: Springer International Publishing; 2015. p. 81–98.
[26] Chen X, Osakue D, Wang N, Parsaei H, Song G. Development of a remote experiment under a unified remote laboratory framework. QScience Proceedings. 2014;2014(3):7.
[27] Zhu Y, Richins D, Halpern M, Reddi VJ. Microarchitectural implications of event-driven server-side web applications. In: Prvulovic M, editor. MICRO 48. [Piscataway, NJ] and [Piscataway, NJ]: IEEE; 2015. p. 762–774.
[28] Brune P, Rockmann R. Towards An Application Architecture For A Smart Online Service Network Platform For The Elderly. Procedia Computer Science. 2017;113:442–447.
[29] Boll F, Brune P, Gewald H. Towards your Parents' Social Network Platform: Design of a User Interface for the Age of Retirement. Proceedings of the Annual Hawaii International Conference on System Sciences. Hawaii International Conference on System Sciences; 2017. .
[30] Jatana N, Puri S, Ahuja M, Kathuria I, Gosain D. A Survey and Comparison of Relational and Non-Relational Database. International Journal of Engineering Research & Technology (IJERT). 2012;2012(Vol. 1 Issue 6, August).