An Efficient 2-Tier Architecture For Modern Web Development

Mathieu Nayrolles, Abdelwahab Hamou-Lhadj
SBA Lab, ECE Dept, Concordia University
Montréal, QC, Canada
{mathieu.nayrolles, wahab.hamou-lhadj}@concordia.ca

Marc Montagne, Vincent Satiat
Toolwatch
Lausanne, Switzerland
{marc, vincent}@toolwatch.io

***Abstract*—zsd**

***Index Terms*—Web Paradigm; Web Architecture; Two tier architecture**

I. Introduction

Modern web development is dominated by the three-tier architecture. In the three-tier architecture, the client (i.e, the browser) presents the user interface which has been generated according to business rules (i.e, dynamic web-page generation). The business rules might rely on stored date (i.e, a database) [1]. Each tier (presentation, application and data) is developed and maintained separately as interchangeable modules with well-defined interfaces. This provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application [2]. In addition to the flexibility during development and maintenance, the three-tier architecture also provides flexibility during operation. Indeed, each tier can be scale separately to meet the demand. Auto-scaling such architecture (i.e, adjusting the computational power of each tier without service interruption) is a popular area of research [3] and business [7].

While the three-tier architecture is popular and provide considerable advantages for developers and maintainers both, the model has been introduced more than twenty years ago. Despite the fact that this architecture has been refined and built upon during the last two decades, the fact remains that, it has been theorized at a time where the Internet user base was 40 millions [source], Netscape was dominating the browser market [source], Javascript [source], Java [source], Linux 1.2.0 [source] and Windows 95 were just hitting the shelves. In recent years, we assisted to the rise of two distinct set of technologies that could, if refined, replace the application layer in a classical 3-tier architecture: (a) Javascript MVC framework and (b) NoSQL database. Indeed, Javascript frameworks are now able to consume APIs [Angular, Backbone, React] and construct dynamic page web and NoSQL databases provide APIs for client to consume and support the map-reduce paradigm [CouchDb]. In theory, one could build a CRUD (create, read, update, delete) two-tier application with a Javascript Framework and a NoSQL database. Javascript applications are served statically to users (i.e, the pages are not dynamically generated) and then, use the Javascript engines of browser to construct html pages based on API calls to the NoSQL database.

Significant gaps exist, however, in these technologies for them to be able to remove entirely the application layer of the three-tier architecture: (a) read/write privileges, (b) account management and (c) schema management.

Indeed, NoSQL databases only provide privileges at the database level. Consequently, an user with the read (write) permission is able to read (write) all the database. Then, users have come to expect from every web-based service some kind of *reset password* functionality even thought it provide a single point of failure into the application security (i.e. the sent email) and the NoSQL databased nor the Javascript application are able to propose such a feature. Finally, NoSQL database are schema-less meaning that they do not have an enforceable definition of what should or should not be in the database. If the database API were to be accessible publicly, then, any malicious user could create database entries containing unexpectedly large documents resulting in a DOS (Denial Of Service) attack.

As a result, companies using Javascript application for their presentation layer and NoSQL database for their data layer are developing and maintaining an application layer using a server-side language.
As depicted by Figure 1, the application layer assess if the current user is authenticated and authorized to do a given action and forwards the call to the NoSQL API.

In order to assess these gaps, we created an add-on for a popular NoSQL database. More specifically, when a request is made to the NoSQL database through its HTTP-API, our add-on filters out unauthorized actions and ensures that the database access are not harmful. In addition, our add-on provides missing functionalities such as access rights, account and schema management by means of a JSON configuration file.

In this paper, we present how we overcome these limitations and created an efficient two-tiered architecture for web services. We experimented this new web-programming paradigm at Toolwatch; a web-based company that allows tens of thousands of clients to measure the accuracy of their mechanical watch

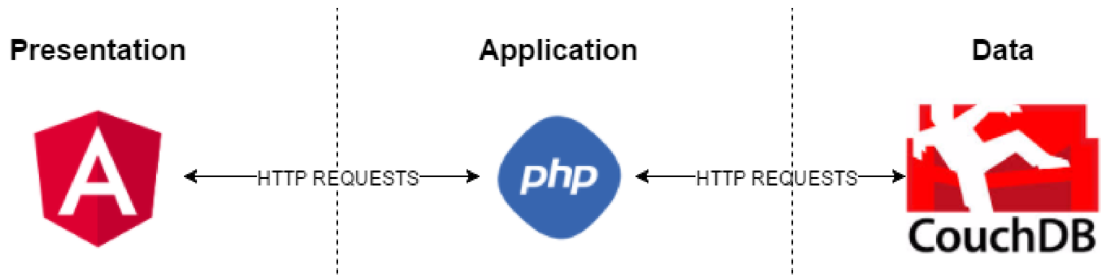

Fig. 1: Managing events happening on project tracking systems to extract defect-introducing commits and commits that provided the fixes

every day, and found that operation costs can be reduced by 63% while improving performances by 34%. In addition, the time required to develop new feature can be reduced by 41%.

The rest of this paper is organized as follows. Section II presents our approach while sections III and IV describe our case study setup and results. Then, section V presents the threats to validity. Finally, sections VI and VII present the related work and propose a conclusion to this paper.

II. A Modern 2-Tier Architecture

III. Case Study Setup

IV. Case Study Results

V. Threats to Validity

VI. Related Work

VII. Conclusion

References

[1] W. W. Eckerson, "Three tier client/server architectures: achieving scalability, performance, and efficiency in client/server applications," *Open Information Systems*, vol. 3, no. 20, pp. 46–50, 1995.

[2] R. Hirschfeld, "Three-tier distribution architecture," *Pattern Languages of Programs (PloP)*, 1996.

[3] J. C. Leite, D. M. Kusic, D. Mossé, and L. Bertini, "Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster," in *Proceeding of the 7th international conference on autonomic computing - icac '10*, 2010, p. 41.

[4] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *2010 11th ieee/acm international conference on grid computing*, 2010, pp. 41–48.

[5] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long, "Optimal Cloud Resource Auto-Scaling for Web Applications," in *2013 13th ieee/acm international symposium on cluster, cloud, and grid computing*, 2013, pp. 58–65.

[6] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures," in *2014 ieee international conference on cloud engineering*, 2014, pp. 195–204.

[7] R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers," in *2009 10th ieee/acm international conference on grid computing*, 2009, pp. 17–25.

[8] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the Cloud Computing era: A survey," in *2010 4th international universal communication symposium*, 2010, pp. 40–46.