Using MongoDB and Mongoose with a MEAN Stack Implementation of the NEXUS-PORTAL-DOORS System

Jason J. Liu, Daniel Yang, Adam Craig, and Carl Taswell

Introduction

For researchers, primary research is only the first stepping stone to proving or disproving a hypothesis. One successful experiment proves little to nothing in the grand scheme of the scientific community. However, the weakness of a single experiment is covered by the power of the meta-analysis, usually completed by a third party. Meta-analysis requires researchers to analyze results from muliple primary articles and assess whether or not the result is accurate, show if the hypothesis presented are proven or disproven, and provide a degree of error. However, despite the necessity of meta-analyses, it continues to be time consuming and ineffective at connecting with all points of data. The miniscule amount of secondary articles pale in comparison to the literal thousands of primary articles which are published each day. In order to take advantage of all the new information which is provided and to create more effective secondary analyses, a semantic web solution provides the necessary tools in order to take full statistical advantage that meta-analysis offers.

One such implementation of a meta-analysis system is the NEXUS-PORTAL-DOORS (NPD) system. In order to support PORTAL registries and DOORS directories, a database is necessary in order to support and store the data. In order to take full advantage of a tried and true framework, MEAN stack serves as the ideal system by thoroughly integrating client-side, server-side, and database processes. As part of that stack, MongoDB alongside Mongoose enables a heirarchical database for storing resource metadata

Methods

For this new NEXUS-PORTAL-DOORS (NPD) system implementation, MEAN (MongoDB, Express, Angular, Node.js) stack is the most updated and contributed fullstack javascript framework. Both backend and front end will rely heavily on node.js, an open-source runtime environment which enables scalable web applications and servers. Used with Express and other packages, node.js enables the user to organize a web application into a MVC (model-view-controller) architecture. In accordance with the "M" from MEAN stack, MongoDB serves as the database for the NPD. As a noSQL database, MongoDB relies on collecetions and documents in contrast to tables and rows. However, MongoDB offers no implementation of queries or updates to the database. Mongoose, a popular object relational mapper for MongoDB, enables connection to a MongoDB database from javascript, allowing model abstraction, large scale queries, data validation, and more. It also enables the user to create schemas, which help generate a structured database.

In the NPDS system, Problem Oriented Registry of Tags And Labels (PORTAL) registers resource labels and tags, inserting or updating them into the database while the Domain Ontology Oriented Resource System (DOORS) publishes resource locations and descriptions with mapping of labels to locations for the semantic web, retrieving stored data from the database. Both PORTAl and DOORS can be optimized for faster read/write if organized in a heirarchical architecture organized through XML. Within the database, the collections for each of these PORTAL, DOORS, AND NEXUS are named portalResources, doorsResources, and nexuResources, respectfully. The database schema for these collections consists of a String handle, which acts as a quick identifer, a String resourceAuthor, which identifiers the user who created the record, and a series of Javascript Objects, which contains metadata information about the record.

- Entity metadata: stores information about the object itself (tags, labels, descriptions)
- Record metadata:stores information about entity for storage purposes (timestamps, which registry or directory)
- Infoset metadata: stores information for management and analysis (status, validation times, info from reasoning machine)
- Representation metadata: shortened version of resource, composed of entity, record, and infoset metadata
- Message metadata: messages between servers or client

In the previous implementation of NPDS, the Microsoft SQL database contained resources in multiple tables with each row containing a small piece of information which

would be used to build an XML document. Furhter information would be kept in more tables, all of which would be accessed through primary keys. The extra time required to create, store, and build a resource from the spread of information is inefficient. However, MongoDB's documentless database structure enables the same metadata resource information to be kept in XML format. However, XML format is not supported by any databases, including Mongoose/MongoDB. Therefore, it is necessary to convert the XML into a data type which is recognized by Mongoose and able to be stored by MongoDB, of which JSON is the most reliable. In the new MEAN stack implmementation for NPDS, the xml record is generated from name, author, recordAuthor, nature, and type in order to create the metadata types mentioned above in XML format. The XML metdata is then converted into JSON and stored in MongoDB to be stored as a Javascript Object. Although each metadata type contains smaller information types, they are simply kept in the appopriate metadata type. This idea of splitting up the individual section down to a few sections was chosen because it enables a degree of modularity for future editing, but retains the accessbility that is offered by storing a resource as XML into MongoDB. In order to read or edit a record, the JSON is converted back into XML using the json2xml npm package.

It is also vital that NEXUS, PORTAL, and DOORS sections of the NPDS system be able to communicate with one another. By using Mongoose, the schema of a collection can be modeled to point at another collection. Combined with queries, this enables a user to communicate and exchange information between NEXUS, PORTAL, and DOORS.

In order to create this implementation of MongoDB and Mongoose for a NPDS system, a template is required as the starting platform. Working with other parts of the NPDS implementation, MongoDB will serve two purposes. One is to contain users for an authentication system, working alongside PassportJS. PassportJS enables the creation of accounts under username and password. It can also be implemented to receive authentication from third parties sucha s Facebook and Twitter, a common practice nowadays. However, for our purposes, the primary advantage of using passport is the level of flexibility it offers. Its primary goal is to keep the complexity of user authentication simple and done in as a few lines of code as possible. This is done with "strategies," which are modules which enable the user to specifically designate which form of authentication they desire, without uncessary modules. PassportJS also supports persistent sessions. Lastly, PassportJS enables the database to store the username as a string, but encodes the password with a series of numbers, ensuring a level of security for the user. The second purpose of MongoDB in the template is to read/write/store metadata. Although the collection names will be changed upon the generation of each server from the template, the role will not. The pimary purpose of the database in the PORTAL registrar is to be able to store metadata resources with labels and optional tags. Meanwhile, the in the DOORS directories, the database must be able to retrieve locations and descriptions of resources, which are specified by references to an OWL ontology.

Results

Due to the flexbility of noSQL and especially lMongoDB's document based database, it offers a database storage which fit the purposes of storing metadata. Furthermore, the automation of this process is enabled by Mongoose, which also helps create a schema and acts as an ODM. The currently implementation proves the potential for a noSQL database such as MongoDB to play a role in the future of NPDS or other semantic web systems. Mongoose is able to generate and insert XML resources, as JSON, into the MongoDB database, as well as retrieving it and converting it back into XML form.

Discussion

Although the Mongoose and MongoDB has been proven to be able to create, store, and retrieve NPDS records, the ability and speed at which an XML record can be edited is still undertermined. Due to the nature of XML and the size of a full resource file, it may become difficult and time consuming to look for identifiers within the resource, edit it, save it, and later query the newly updated resource. Another way to take a step forward in storing semantic records is to XML can stored directly into MongoDB, although this would require an entirely new ODM entirely.

Conclusion

Previously, the NEXUS PORTAL DOORS System for a semantic web relied on Microsoft SQL in order to store records. However, while a standard SQL solution is viable, MongoDB, paired with Mongoose, enables a free, open sourced database while enjoying the benefits of MongooDB's sensible data storage and query capabilities.

Acknowledgments

This research was supported by the Brain Health Alliance Virtual Institute. We'd like to thank everyone who particpated in the 2016 program with us, especially Sujay Ratna and Ben Bae, who cooperated closely with us towards a semantic web solution.

References

1. Cattell, R. Scalable SQL and NoSQL data stores [Google Scholar]
2. Daniel Peng Frank Dabek, G. I. Large-scale Incremental Processing Using Distributed Transactions and Notifications [Google Scholar]
3. Elif Dede Daniel Gunter, R. S. C. L. R. Performance evaluation of a mongodb and hadoop platform for scientific data analysis [Google Scholar]
4. Ghislain Fourny Jsoniq The, S. N. G. F. JSONiq The SQL of NoSQL [Google Scholar]
5. Harpinder Kaur, J. S. Improvement in Load Balancing Technique for MongoDB Clusters [Google Scholar]
6. Jyotsna Talreja Wassan, A. P. Exploratory Implementation of Stream Clustering Algorithm using MongoDB [Google Scholar]
7. Ljiljana Stojanovic Nenad Stojanovic, R. V. Migrating data-intensive Web Sites into the Semantic Web [Google Scholar]
8. Pradeep Soni, N. S. Y. Quantitative Analysis of Document Stored Databases [Google Scholar]
9. Rajat Aghi Sumeet Mehta, R. C. S. C. N. B. A comprehensive comparison of SQL and MongoDB [Google Scholar]
10. Rupali Arora, R. R. A. Modeling and Querying Data in MongoDB [Google Scholar]
11. Sanobar Khan, P. V. M. SQL Support over MongoDB using Metadata [Google Scholar]
12. Stefanie Scherzinger, M. K. Managing Schema Evolution in NoSQL Data Stores [Google Scholar]
13. Taswell, C. Correction Corrections to DOORS to the Semantic Web and Grid With a PORTAL for Biomedical Computing 2008 [Google Scholar]
14. Taswell, C. DOORS to the semantic web and grid with a PORTAL for biomedical computing 2008 [Google Scholar]
15. Taswell, C. Article A Distributed Infrastructure for Metadata about Metadata: The HDMM Architectural Style and PORTAL-DOORS System 2010 [MDPI]
16. Zhengxiang Pan, J. H. DLDB: Extending Relational Databases to Support Semantic Web Queries 2003 [Google Scholar]
17. Dickey, J. Write modern web apps with the MEAN stack: Mongo, Express, AngularJS, and Node. js Education, P. (ed.) Pearson Education, 2014 [Google Scholar]
18. Pasquali, S. Mastering Nodejs Edward Gordan, G. W. (ed.) Packet Publishing Ltd., 2013 [Google Scholar]
19. Chodorow, K. and Dirolf, M. MongoDB: The Definitive Guide Steele, J. (ed.) O' Reilly, 2010 [Google Scholar]
20. Holmes, S. Mongoose for Application Development Mizen, G. (ed.) Packet Publishing Ltd., 2013 [Google Scholar]
21. Barrasa Rodriguez J., C. .. and Gmez-Prez, A. R2O, an extensible and semantically based database-to-ontology mapping language Springer-Verlag, 2004 [Google Scholar]
22. Kei-Hoi Cheung Andrew K. Smith, K. Y. Y. C. J. B. and Gerstein, M. B. Semantic Web Approach to Database Integration in the Life Sciences Revolutionizing Knowledge Discovery in the Life Sciences, 2007 [Google Scholar]