Visual Statement

NoSQL Data Storage

MongoDB Project

April 10, 2014

Bobby Esfandiari
Stefan Schielke
Nicole Saat

Table of Contents

Timeline

Week #	Dates	Sprint #
Weeks 3 and 4	Jan 20 – Feb 2	Sprint 1

Become familiar with MongoDB and its functionality.

- Reviewed MongoDB Tutorial videos as well as NoSQL Distilled.
- Tested understanding by creating test databases, test collections, and test documents

Ensure MongoDB is ready to gather and store existing JSON BLOB Data.

- Took existing JSON BLOB Data from client and imported it into MongoDB.
- Created test JSON BLOB Data with multiple documents and imported them into MongoDB

Have MongoDB storing data properly

- Tested changing data existing in documents as well as copying existing documents into new collections
- Tested storing data as new objects, storing objects in arrays, and storing multiple levels of objects and arrays
- Tried to create objects in new collections with the same unique ID to begin versioning testing

Other Sprint Details:

- Created Requirements document and presented to client
- Created Analysis document
- Created Design document
- Began research on indexing, other databases' pros and cons, and versioning to discuss with client in next sprint
    - These analyses have been included in the Analysis doc. These will be discussed with the client to find the best possible choice in regards to the client's needs
- Began research on Multi-tenancy options to discuss with client in next sprint

Week #	Dates	Sprint #
Weeks 5 and 6	Feb 3 – Feb 16	Sprint 2
Have all necessary data stored in MongoDB • This has been postponed due to changes in data format. Due to limitations with indexing/size restrictions/data storage in MongoDB, team will be re-evaluating storage capabilities based on "clean data" • "Clean data" will be provided to team in coming week; as well as current data will be stripped down to pertinent information. Incorporate Multi-tenancy and compartmentalization of data, and maintain this functionality • Virtualized multi-tenancy will be incorporated based on having unique ID's for each state, region, agency, etc. that will be referenced based on a user's privileges. This will allow data to be compartmentalized based on API and UI search functionality Ensure versioning capabilities are maintained • Implemented Vermongo methodology in which a shadow collection houses previous document versions. Ensure Indexing is incorporated as per client's requested format • This has been postponed due to changes in data format. There are limitations for indexing when data with unused fields is incorporated. "Clean data" will be provided to the team and this will be re-evaluated in coming sprints Other Sprint Details: • Mid-term review was completed Feb. 14. Client has expressed satisfaction with progress and has agreed to continue • Team has found limitations based on Indexing that will be pushed into later sprints		
Week 7	Feb 17 – Feb 23	N/A
Reading Break. MongoDB Milestone should be completed by this week. • This milestone has been completed with the exception of Indexing and Data Storage. The team is continuing to evaluate these topics Team will prepare to start developing API and querying functionality with existing data. This will include learning C# programming techniques • Team has loaded Visual Studio 2012 onto all personal machines, and have begun to implement C# and MongoDB Driver functionality. Create a game plan in regards to difficult API querying techniques • This will be evaluated further in coming sprints Other Sprint Details: • Team has started evaluation of Azure hosting. Team is awaiting approval for trial license for Azure to incorporate as per client's request • Team has stripped current JSON BLOB data of redundant data for testing • Team is awaiting more test data from client for testing		

Week #	Dates	Sprint #
Weeks 8 and 9	Feb 24 – Mar 9	Sprint 3
API created to submit data and update data elements. • Created. API for basic searching functionality • Created for querying known fields without full-text search Part 2: Ensure Indexing is incorporated as per client's requested format • Still being tested. Part 2: Have all necessary data stored in MongoDB • Data has been created and imported for stripped form and unstripped form		
Weeks 10 and 11	Mar 10 – Mar 23	Sprint 4
API for complete search functionality • Implemented. Currently search is returning _id field of documents, field return will be implemented next sprint API querying must be complete at the end of this stage. • Partially completed. Querying for single and multiple items has been implemented, return fields will be changed next sprint The complete functioning prototype • Complete other than return fields implemented next sprint. Presentation-ready GUI will be completed next sprint Part 2: Ensure Indexing is incorporated as per client's requested format • Full Text Index has been implemented in the API to search keywords without knowing field. Tests will be run next sprint on 5 million record file with index and non-index to see speed		
Weeks 12 and 13	Mar 24 – Apr 6	Sprint 5
Sprint dedicated to graphical interface and to account for any delays in milestones previously listed • Partially completed. The rest of the GUI will be completed before presentation April 10 API to return field of searched data • Completed. Field is returning path of searched data, as discussed with client Data retrieved on speed of indexed and non-indexed fields • Partially completed. Testing has been run over weekend and will be compiled April 7 (Monday) Code sent to client by Friday, Apr 4 as requested • Completed Time permitting, complete documentation • Partially completed. Documentation is expected to be completed before presentation April 10 Other Sprint Details: • Testing has been tailored to requests from client. Results will be documented		
Week 14	Apr 7 – Apr 13	N/A
Presentation April 10		

Requirements

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	23-Jan-2014	Initial Requirements	Bobby Esfandiari Stefan Schielke Nicole Saat
2	24-Jan-2014	Updated the “Update an existing Document” use case to exclude deleting/include versioning Added Version Control to requirements definition Updated Glossary	Bobby Esfandiari Stefan Schielke Nicole Saat
3	15-Mar-2014	Updated the following: Query Documents	Bobby Esfandiari Stefan Schielke Nicole Saat

Introduction

MongoDB is an open-source, NoSQL database that syncs with JSON data. It provides flexible capabilities for storage, as well as compatibility with a wide range of programming languages.

The aim of this project is to provide a “proof of concept” representation of MongoDB with specific sample data sets. This concept will either confirm or refute the client’s request to populate MongoDB with existing JSON BLOB data and provide querying functionality.

Glossary

Collection: Similar to a table in a relational database – a group of documents in MongoDB

Document: Similar to a row or record in a relational database – an instance in MongoDB. This will refer to each JSON BLOB

Field: Similar to a column in a relational database – a category in MongoDB

_id: Unique identifier in MongoDB. Cannot be changed after creation and cannot be duplicated

Customer: Agency using the product

Users: Employees at the agency using the product

Key Notes

User Interface

The client has advised the team that the user interface is not crucial to the final product. If the client adopts the product, they will be rendering their own user interface.

User Requirements Definition

The project must be able to achieve the following:

Multi-tenancy and Protection of Data

The database must allow for compartmentalization of data for storage and retrieval by specified customers. Information stored is sensitive, and must be accessible only to the approved customer.

Version Control

The database must provide version control for any document being updated. When querying, it should only query the existing version, but the database must allow versions to be saved when updated.

API Approach for Data Management

The database must be accessible through various API methods and queries. These must include:

- submission of data into MongoDB
- update data elements associated within MongoDB
- search data fields within the documents
- aggregate querying
    - return a list of all reports where weather conditions were dry for example

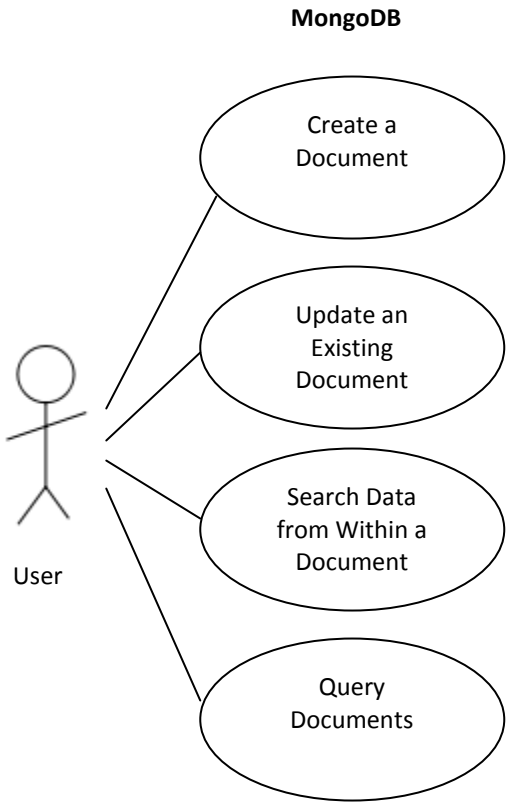
Functional Requirements

1. User to Create a Document
    a. Add document to database
2. User to Update an Existing Document
    a. Add information to document
    b. Update existing information in document
3. User to Search Data Within an Existing Document
    a. Request a field
    b. Request multiple fields
4. User to Query Data about Existing Documents
    a. Query documents with data in a known field
    b. Query documents with data in an unknown field

System Requirements Specifications

Use Cases

1. High Level Use Case

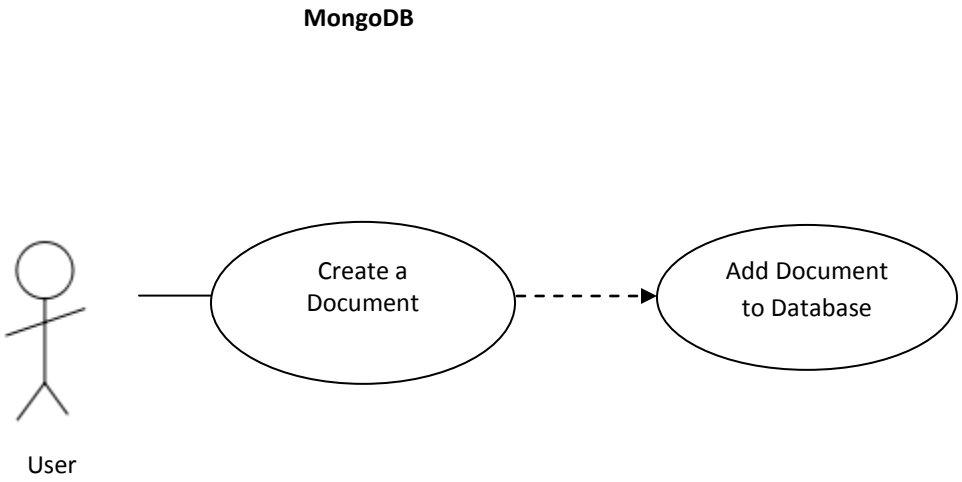


2. Create a Document

Description: The Create a Document use case enables a user to add a new document to the database.

Step-by-Step Description

1. Add document to database
    a. Populate necessary fields for item, save as document



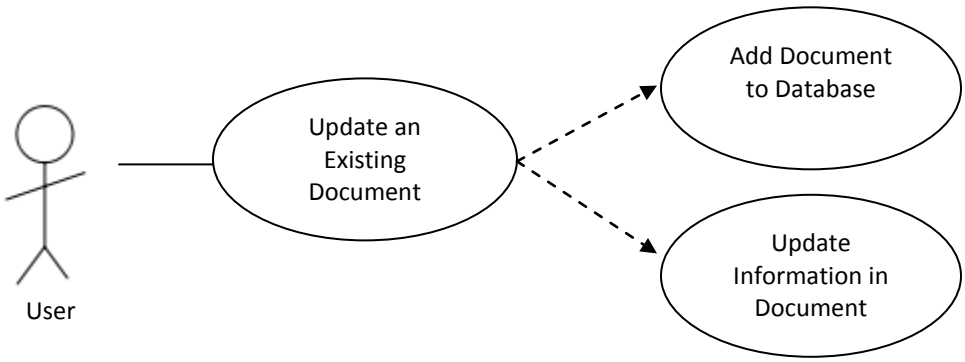
3. Update an Existing Document

Description: The Update a Document use case enables a user to add or update information in an existing document.

Step-by-Step Description

1. Add information to document
    a. Check to see if information is currently in field to update. If so, change information instead
    b. Save data under specific field in document
2. Update information in document
    a. Check to see if information is currently in field to update. If not, add information instead
    b. Set version of existing document
    c. Update information

MongoDB



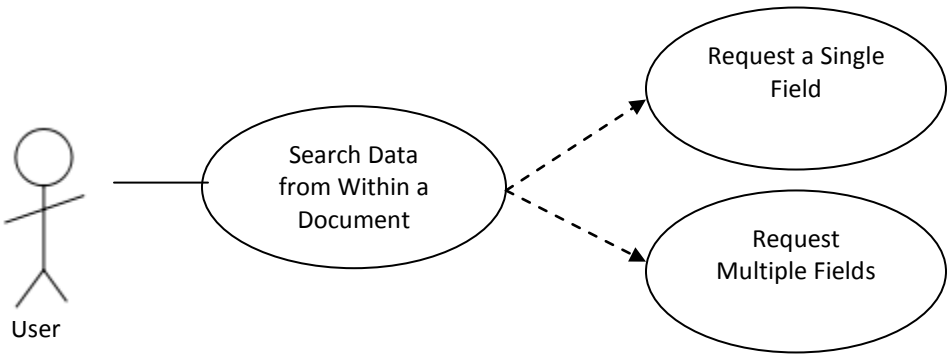
4. Search Data from Within a Document

Description: The Search Data from Within a Document use case enables a user to request specific data about a document. This could include any of the fields the document currently holds

Step-by-Step Description

1. Request a field
    a. Look up document with _id or query based on other field data
    b. Request data from specified field of document
    c. Output data
2. Request multiple fields at once
    a. Look up document with _id or query based on other field data
    b. Request data from specified fields of document
    c. Output data for each field

MongoDB

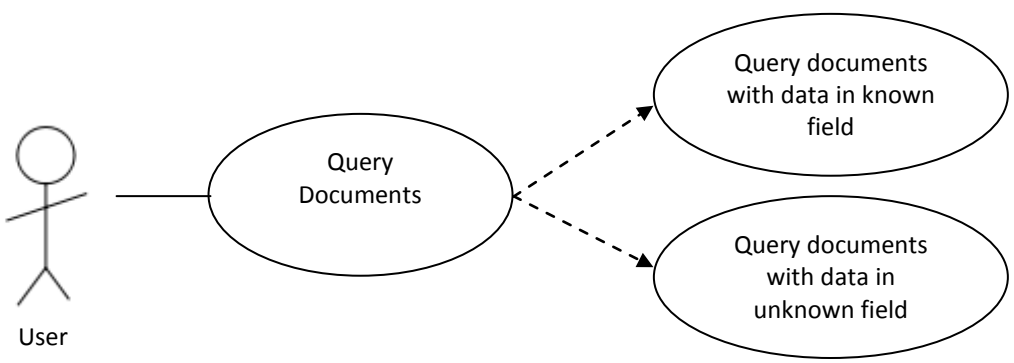


5. Query Documents

Description: The Query Documents use case enables a user to find all documents that contain specified data. This data could be in a known field, or unknown field within documents.

Step-by-Step Description

1. Query documents with data in a known field
    a. Search for documents with a specified field containing specified data
    b. Return _id, parameter of search, record, and date of record for each document found
2. Query documents with data in an unknown field
    a. Search for documents with specified data
    b. Return _id, parameter of hit, record, and date of record for each document found



Analysis

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	25-Jan-2014	Initial Analysis	Bobby Esfandiari Stefan Schielke Nicole Saat
2	1-Feb-2014	Updates to Analysis, including: • Recommendation/Choice of Versioning	Bobby Esfandiari Stefan Schielke Nicole Saat
3	5-Feb-2014	Updates to Analysis, including: • Indexing	Bobby Esfandiari Stefan Schielke Nicole Saat
4	8-Feb-2014	Updates to Analysis, including: • Multi-Tenancy evaluation and choice • Version Control choice • Indexing Choice	Bobby Esfandiari Stefan Schielke Nicole Saat
5	12-Feb-2014	Updates to Analysis, including: • Indexing options and choice	Bobby Esfandiari Stefan Schielke Nicole Saat
6	17-Feb-2014	Updates to Analysis, including: • Multi-tenancy choice • Version Control Choice • Indexing	Bobby Esfandiari Stefan Schielke Nicole Saat
7	28-Feb-2014	Updates to Analysis, including: • Added Insert Document API	Bobby Esfandiari Stefan Schielke Nicole Saat
8	7-Mar-2014	Updates to Analysis, including: • Indexing choice and review	Bobby Esfandiari Stefan Schielke Nicole Saat
9	15-Mar-14	Updates to Analysis, including: • Glossary • Indexing Choice • Insert Documents • Querying	Bobby Esfandiari Stefan Schielke Nicole Saat
10	4-Apr-14	Updates to Analysis, including: • Indexing Choice • Insert Documents API – moved to design doc	Bobby Esfandiari Stefan Schielke Nicole Saat

Introduction

MongoDB is an open-source, NoSQL database that syncs with JSON data. It provides flexible capabilities for storage, as well as compatibility with a wide range of programming languages.

This analysis document has been developed to weigh options in regards to plausible databases, as well as which procedures will be executed to create the final product.

Glossary

Collection: Similar to a table in a relational database – a group of documents in MongoDB

Document: Similar to a row or record in a relational database – an instance in MongoDB. This will refer to each JSON BLOB

Field: Similar to a column in a relational database – a category in MongoDB

_id: Unique identifier in MongoDB. Cannot be changed after creation and cannot be duplicated

Customer: Agency using the product

Users: Employees at the agency using the product

Parameter of Hit: Field in which queried data was found

Key Notes

Hosting

The chosen database must be able to be hosted on Azure.

Analysis Definition

The project must be able to achieve the following:

Multi-Tenancy and Protection of Data

The database must allow for compartmentalization of data for storage and retrieval by specified customers. Information stored is sensitive, and must be accessible only to the approved customer.

Version Control

The database must provide version control for any document being updated. When querying, it should only query the existing version, but the database must allow versions to be saved when updated.

API Approach for Data Management

The database must be accessible through various API methods and queries. These must include:

- submission of data into the database
- update data elements associated within the database
- search data fields within the documents
- aggregate querying
    - return a list of all reports where weather conditions were dry, for example

To ensure the best possible results are obtained for the completed project, we will analyze the pros and cons of possible solutions to confirm the best result.

Database Choice

The client requires a database that has specific functionality within the final product. To choose the correct database, the following characteristics must be taken into account:

- Azure-compatible for hosting
- NoSQL for flexible data storage and querying
- Provides multi-tenancy capabilities due to sensitive data that cannot be accessed by incorrect customers
- Version control for updates to the database documents
- Multiple querying capabilities to run queries on multiple fields at once

The following databases were evaluated.

Database	Azure Compatible	NoSQL	Multi-tenancy Capabilities	Version Control	Multiple Queries
MongoDB	Yes	Yes	Yes – see "multi-tenancy" evaluation below	Partly – see "version control" evaluation below	Yes – see "querying" evaluation below

Please see the following links, which outline different databases to be evaluated:

- http://nosql.findthebest.com/d/d/Disk
- http://www.drdobbs.com/database/nosql-options-compared/240151198?pgno=1

Based on review (including the above documents), we believe the best possible option is MongoDB due to the requirements necessary for the client.

Multi-Tenancy

There are various methods to invoke multi-tenancy in MongoDB:

Method	Pros	Cons
Separate database for each customer	• Allows for multiple collections inside database • Heightened security – no overlap of data	• May be more difficult for client to query multiple customers • Each database would have to be initialized with keys
Separate collection for each customer	• Only one database for all customers • Can query data only on specific clients	• There would be an excessive amount of collections if shadow collections are implemented with Vermongo
Single collection for all customers	• "Overlord scenario" can be accounted for, in which different levels of querying will be fulfilled	• Will be more complicated to ensure compartmentalization of data

Recommendation

Based on the review listed above and team's understanding of client requirements, we recommend using separate databases for each customer. There is a heightened level of security using databases instead of collections, and will be more straightforward to keep track of each customer and their current/shadow collections for document storage

Choice

Upon review with the client, the single collection paradigm will be used. In this scenario there will be one database and collection for all agencies (plus one extra shadow collection for version control), and compartmentalization will be handled through document querying.

After reviewing this methodology, we have found any collection that is between 250MB and ~400MB cannot be sharded. Therefore, we have suggested each state has its own collection – this will account for the "overlord scenario" (detailed in querying section), as well as keeping collection size low.

Version Control

There are various methods to invoke version control in MongoDB:

	Method	Pros	Cons
1	Store object inside each document to store past data[1]	• The data would be stored with the current document, without access from the user • The object could store each past item with either a count or timestamp to show last version	• Could be slow depending on how many updates must be done when updating the document • Can make documents very large
2	Incorporate a "published", "draft", and "history" object under each document	• Same pros as method 1 • Allows for a draft to be saved without being queried	• Same cons as method 1
3	Use "Vermongo" methodology[2]	• No constraints on what kind of document must be versioned • Saves all versioned documents in a "shadow collection" mimicked to the live collection • Does not affect querying of current data • Separates current data from versioned data which will eliminate querying past data	• Documents must be updated one by one • A fair bit of overhead could slow system down • Can only update when you know the current version number • No document compression so each version takes up same amount of space

Recommendation

Based on the review listed above and team's understanding of client requirements, we recommend using the Vermongo methodology. This allows for the simple transfer of data between collections without loss of document data. This also provides easier querying in the current collection for customers and users, as querying does not need to handle different document versions.

Choice

Test cases of Vermongo were provided to the client, who approved the idea.

The team has clarified with the client that current documents may or may not be stored the shadow as well (this is up to the team's design). The team has decided to exclude the current document from the shadow collection to keep storage sizes low.

[1] See the following link for explanation of this method: http://stackoverflow.com/questions/4185105/ways-to-implement-data-versioning-in-mongodb

[2] See the following link for explanation of this method: https://github.com/thiloplanz/v7files/wiki/Vermongo

Indexing

There are various ways to use indexing in MongoDB:

	Method	Pros	Cons
1	Compound Index on Name and Value	• Not a lot of necessary change to existing data set as index is on existing data • Use a list to store properties • eleMatch() is quick	• Needs many indexes • findOne() is slow
2	Basic Multi-Key	• Query for a document containing anything in indexed-value array • One index needed	• Must update existing data • Cannot use eleMatch() since there is more than one value in array
3	Advanced Multi-Key	• employs pros of compound indexing and basic multi-key • can match fields without knowing what they are in advance	• must update existing data, will be more complicated • documents take more space due to more key values
4	Full Text Search	• Indexes all data	• Large index

Recommendation

Based on our research and the client's needs, we recommend the advanced multi-key methodology. Although it causes a lot more data to be stored, there is less overhead when searching data, especially with large data sets.

Choice

Upon review with client, Full Text Search will be used

Although this will cause higher storage for keys, the amount of storage necessary compared to developer time to maintain other indexing strategies is minimal.

Note: See Review Below

Review

- Upon review, Advanced Multi-Key Indexing is a less plausible choice than the full-text search. The Key-Value pairs can cause issues in creating documents and returning data, as only one key-value pair will be stored for each object.

The team will be re-evaluating this section based on the Mid-Term review.

- Since the Mid-Term Review, the team has been evaluating plausible options. The client's needs include searching for an item without knowing the field – which can only be done with a full-index search, or an array of indexes. The client wishes to minimize developer time updating the system, so the full-text's dynamic indexing is the best choice.

Querying

Querying will allow a user to search for related documents containing specified data. The user may or may not know what field they are searching within.

Querying with Known Field to Search

If the user is aware of the field they are searching in, it will be incorporated in the document search. All documents found with specified data in the specified field will be queried, and the _id field of all found documents will be returned (as well as other declarative data for the user).

Querying with No Known Field to Search

If the user does not know what field they wish to search but knows the data they would like to find, a full-text search will be done on all documents. All documents with specified data found anywhere in the document will be queried. The _id field and the parameter of hit will be returned for each found document (as well as other declarative data for the user).

Design

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	1-Feb-2014	Initial Design Document	Bobby Esfandiari Stefan Schielke Nicole Saat
2	8-Feb-2014	Updated Design, including: • Multi-tenancy redone based on client analysis discussion • Indexing added	Bobby Esfandiari Stefan Schielke Nicole Saat
3	17-Feb-2014	Updated Design, including: • Multi-Tenancy explanation • Setting up Vermongo • Indexing	Bobby Esfandiari Stefan Schielke Nicole Saat
4	7-Mar-2014	Updated Design, including: • Setting up Vermongo • Vermongo API • Appendix Example Vermongo	Bobby Esfandiari Stefan Schielke Nicole Saat
5	15-Mar-14	Updated Design, including: • Design Definition • Multi-Tenancy explanation/API • Indexing • API and Data Management	Bobby Esfandiari Stefan Schielke Nicole Saat
6	4-Apr-14	Updated Design, including: • Multi-Tenancy API • Indexing explanation/API • Querying	Bobby Esfandiari Stefan Schielke Nicole Saat

Introduction

MongoDB is an open-source, NoSQL database that syncs with JSON data. It provides flexible capabilities for storage, as well as compatibility with a wide range of programming languages.

This design document has been developed to capture the reasoning and functionality behind the product design.

Glossary

Collection: Similar to a table in a relational database – a group of documents in MongoDB

Document: Similar to a row or record in a relational database – an instance in MongoDB. This will refer to each JSON BLOB

Field: Similar to a column in a relational database – a category in MongoDB

_id: Unique identifier in MongoDB. Cannot be changed after creation and cannot be duplicated

Customer: Agency using the product

Users: Employees at the agency using the product

Parameter of Hit: Field in which queried data was found

Key Notes

None

Design Definition

The design of the project is crucial to its functionality and ease-of-use both for the client, and for the client's customers and users. There are specific areas in which the design is important for the workings of the final product. These areas have been evaluated in the Analysis and Requirements documents.

Design can be categorized under the following:

- JSON BLOB Data Storage
- Multi-Tenancy and Protection of Data
- Version Control
- Indexing
- Querying

JSON BLOB Data Storage

JSON BLOB data is stored as a BSON BLOB (binary form of JSON data): the primary storage technique for MongoDB. MongoDB can import and export files with JSON BLOB data, as well as print JSON data in a user-friendly format with specific commands.

Importing existing JSON BLOB file data[3]

Importing existing data from a JSON file consists of the following command:

mongoimport --host <hostname> --db <dbName> --collection <collectionName> < <path> --jsonArray

This command specifies the host, the database and collection to import to, the directory location (path) of the existing file, and a final command to import data whilst ignoring return characters (to import data that has been organized for user ease rather than one line per object).

Example:

mongoimport --host 127.0.0.1 --db customerOne --collection current < C:\data\db\FieldReport.json --jsonArray

```
C:\mongodb\bin>mongoimport --host 127.0.0.1 --db customerOne --collection curren
t < C:\data\db\FieldReport.json --jsonArray
connected to: 127.0.0.1
Sat Feb 01 13:58:43.870 imported 1 objects
```

API

Multiple Documents at Once

When developing the API for inserting documents into MongoDB, we have found that the way JSON BLOBs are entered can cause errors. JSON BLOB data that will be made into documents must be a single object with a beginning and end bracket pair, regardless of whether it is on one line or multiple lines. This can become problematic when entering more than one document at once.
To handle this, the team found that coding through bracket pairing is simplest, in which data is read into a string, and a counter handles finding a beginning and end bracket pair per object.

public static void insertMultipleDocsFromFile(string, MongoCollection<BsonDocument>)

Insert Single Document at a Time

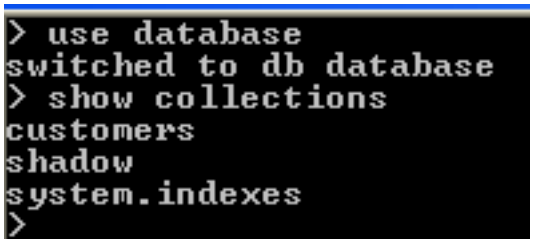
This will be the format most used in the final product, as one document will be created at a time in industry. The document must create a shadow document to prepare for versioning when the document is updated.

public static void insertDocFromFile(string, MongoCollection<BsonDocument>)

[3] Please note that the team recommends importing straight into MongoDB and not through API for the initial import of data.

Multi-Tenancy and Protection of Data

The Analysis indicated that the best approach for multi-tenancy is to create a single collection for each state. This will allow for different levels of customers (for example, state departments, county offices, etc.) to be able to query data with minimal impact to product design. A current and shadow collection will still be kept to control versioning of documents.



Data will only be accessible based on a user's privileges – this will be handled by instantiating an object in each document housing ID fields for the document's agency, county, region, state, etc. When a user tries to query documents for reports, their access will be tested against these ID fields, and only data pertaining to their access level will be retrieved.

API

Multi-tenancy in the API is invoked by declaring access variables for a user at any given time. This includes three variables: one for agency, one for county, and one for state. If a query is done just on a state, the agency and county are left null, and so on for each level of the "overlord scenario" search.

Example with sample data:

private static string agency = "GHA10001";

private static string county = "XYZ101";

private static string state = "CA";

These variables are then used when a search is to take place. The confirmAccess() method is called, which checks to see if these values are listed, and sets up a query parameter that filters all search results to only contain these values.

public static IMongoQuery confirmAccess()

Version Control

The Analysis indicated that Vermongo is the accepted version control ideology to be used in this project.

Vermongo Methodology

Vermongo uses a technique that involves two separate collections for version control. The original collection contains all current versions of documents. The shadow collection contains all previous versions of each document.

Each document stored in the shadow has the same _id (unique ID) as the current document, but instead has an array that stores objects. Each object in the array is a previous version of said document. This allows for snapshots of documents to be stored separately from the current collection, but can be retrieved whenever necessary.

Setting Up Vermongo

Setting up Vermongo has two different processes – initial setup of collections, and creating new documents.

The initial setup consists of creating a shadow collection of the current collection. Invoking the following commands for each document achieves this:

Save variables current document ID, and copy of document

var document = db.<collection>.findOne({<search parameter>})

var id = db.<collection>.findOne({<search parameter>})._id

Move to shadow collection

Create new document with same ID as original

db. <shadow>.save({_id:id})

Create a version variable and snapshot array to store document versions

db. <shadow>.update({_id:id}, {$set:{version:0, snapshot[]}})

This will duplicate the collection's _id fields into the new shadow collection.

Note: For the initial database set-up, the most efficient way to create the shadow collection is to import the current collection, export it to retrieve all _id's, parse the _id field out for each document and create a new JSON file that includes this command for each document:

{_id: { "$oid" : "<_id of document>" },version:0, snapshot:[]}

Invoking Vermongo

Vermongo uses the following functions to execute[4]:

Step 1: Update the current document with the new changes. The existing document is already in the shadow collection.

Step 2: Save variables of the complete document, and current document ID

var document = db.<collection>.findOne({<search parameter>})

var id = db.<collection>.findOne({<search parameter>})._id

Step 3: Move to the shadow collection

db.<shadow>

Step 4: Push the document into the snapshot array and increment the version number by one. Please note that the version number will coincide with the array position of the document for easier access

db.<shadow>.update({_id:id}, {$inc:{version:1}, $push:{snapshot:document}})

API

Vermongo in the API is invoked by:

- querying the _id field in the collection
- saving a version of the current document
- querying the _id field of the shadow
- pushing a passed version of the document into the snapshot array
- incrementing the version number.

Method(s):

updateDoc(ObjectId, string, string, MongoCollection<BsonDocument>, MongoCollection<BsonDocument>)

saveVersion(BsonDocument, ObjectId, MongoCollection<BsonDocument>)

[4] See Design Appendix for example of Vermongo process

Indexing

The Analysis indicated that full text indexing is the accepted indexing method to be used in this project.

This indexing technique indexes all fields and data within a document. This allows for flexible indexing to occur, as fields do not need to be known beforehand.

With this, we employ the following index per collection in MongoDB:

```
db.collection.ensureIndex ( { “$**” : ”text” } , { name : “TextIndex” } )
```

API

Indexes created are used during querying in the API. This is invoked during the fullTextSearch method when using the search command:

```
var textSearchCommand = new CommandDocument
{
   {"text",collection.Name},
   {"search", search},
   {"limit", -1},
   {"filter", BsonValue.Create(confirmation)}
};
```

This invokes the index “text” in the collection, the search as the search parameters passed, no limit[5] for results, and the filter created for the agency/county/state (as listed in Multi-Tenancy). This command is then used to run the full text search.

[5] MongoDB automatically sets a limit to only return 100 documents with each search. Limits can be specified with a specific number, or eliminated with -1

Querying

Querying is one of the key components to the final project. Users should be able to request documents with specified data, which will be returned. Querying should allow for two different options – knowing the field to search, and not knowing the field to search.

Queries will involve three steps

- building the query, which involves filters, indexes, and searchable data
- executing the search, which involves running the search and retrieving all relevant documents
- returning the data, which includes gathering relevant data to be returned to the user, and printing data to the user if necessary

API

Queries, as noted above, can either involve knowing the field to search (the key-value pair), or not specifying the field to search.

Key-Value Pair

If the key-value pair is known, the following method will be invoked:

public static string[] queryOneField(string, string, MongoCollection<BsonDocument>)

This method passes the key, value, and collection of the search. This will return[6] the _id of relevant documents.

Full Text Search

If the key-value pair is not known, the following method will be invoked:

public static List<Results> fullTextSearch(string[], MongoCollection<BsonDocument>, MongoDatabase)

This method passes a string of parameters to search (if multiple search values), the collection of the search, and the database of the search. This will return a list of Results objects.

Inside of the fullTextSearch method, there are various supporting methods. These include:

- confirm access method to create filter based on agency/county/state (as noted in multi-tenancy section

  public static IMongoQuery confirmAccess()
- convert search string method to take all passed search parameters and make them into one searchable string for use in fullTextSearch

  public static string convertSearchString(string[,])

[6] Since the field is already known, the path is not returned for this search

- filter document method that takes the resulting documents from search (one at a time) and parses out pertinent data. This method returns a Results object to be added to list of results

  public static Results filterDocument(BsonValue, string[,])

Results Class

The Results class constructs Results objects that store pertinent information about each document found in a query. The Results object represents one document, and contains the following information:

public string IdDoc	//unique ID of document
public string[][] ItemPathDoc	//all paths indicating where queried data is found
public string[] ReportNumbers	//report numbers associated with document

IdDoc stores the unique identifier for the document

ItemPathDoc is a jagged array that stores the paths in which all searched data was found in a document. The path includes report numbers.

ReportNumbers stores all report numbers associated with that document.

Design Appendix

Vermongo Process Example

Existing Collection

```
> db.current.find().forEach(printjson)
{
        "_id" : ObjectId("52e061dee9e1000c678f1000"),
        "FirstName" : "Bruce",
        "LastName" : "Wayne",
        "Email" : "bwayne@Wayneenterprises.com"
}
{
        "_id" : ObjectId("52e061dee9e1000c678f1001"),
        "FirstName" : "Lucius",
        "LastName" : "Fox",
        "Email" : "lfox@Wayneenterprises.com"
}
{
        "_id" : ObjectId("52e061dee9e1000c678f1002"),
        "FirstName" : "Dick",
        "LastName" : "Grayson",
        "Email" : "dgrayson@Wayneenterprises.com"
}
```

Step 1: Save variables of the complete document, and current document ID

Note: The search parameter may be the unique ID if initially searched this way

var document = db.batman.findOne({FirstName: "Bruce"})

var id = db.batman.findOne({FirstName: "Bruce"})._id

Step 2: Move to the shadow collection

db.shadow

Step 3: Push the document into the snapshot array and increment the version number by one. Please note that the version number will coincide with the array position of the document for easier access

db.shadow.update({_id:id}, {$inc:{_version:1}, $push:{snapshot:document}})

```
> db.shadow.find().forEach(printjson)
{
        "_id" : ObjectId("52e061dee9e1000c678f1001"),
        "version" : 0,
        "snapshot" : [ ]
}
{
        "_id" : ObjectId("52e061dee9e1000c678f1002"),
        "version" : 0,
        "snapshot" : [ ]
}
{
        "_id" : ObjectId("52e061dee9e1000c678f1000"),
        "snapshot" : [
                {
                        "_id" : ObjectId("52e061dee9e1000c678f1000"),
                        "FirstName" : "Bruce",
                        "LastName" : "Wayne",
                        "Email" : "bwayne@Wayneenterprises.com"
                }
        ],
        "version" : 1
}
```

Step 4: Change “Bruce” Wayne to “Bob” Wayne

db.batman.update({FirstName: “Bruce”}, {$set:{FirstName: “Bob”}})

Implementation

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	5-Apr-2014	Initial Implementation	Babak Esfandiari Stefan Schielke Nicole Saat
2	6-Apr-2014	Update Implementation, including: • API • System	Babak Esfandiari Stefan Schielke Nicole Saat
3	7-Apr-2014	Update Implementation, including: • System	Babak Esfandiari Stefan Schielke Nicole Saat

Introduction

MongoDB is an open-source, NoSQL database that syncs with JSON data. It provides flexible capabilities for storage, as well as compatibility with a wide range of programming languages.

This implementation document has been developed to indicate how the project has been set up, both with the version information of hardware/software, as well as with the API organization.

Glossary

Collection: Similar to a table in a relational database – a group of documents in MongoDB

Document: Similar to a row or record in a relational database – an instance in MongoDB. This will refer to each JSON BLOB

Field: Similar to a column in a relational database – a category in MongoDB

_id: Unique identifier in MongoDB. Cannot be changed after creation and cannot be duplicated

Customer: Agency using the product

Users: Employees at the agency using the product

Parameter of Hit: Field in which queried data was found

Key Notes

Hosting

The chosen database must be able to be hosted on Azure.

Methods

When discussing implementation of API below, any related methods that are described are indicated by square brackets. These methods can be found in further parts of the document.

Implementation – System

To implement and maintain the project, we have separated tools into two different categories – computer hardware, MongoDB and API. MongoDB relates to the database itself, including data storage, importing and exporting data, and running some forms of testing. API relates to running queries, some forms of testing, and user interaction with the underlying database.

Computer Hardware

The project has been run on four different systems, however all testing was done on one machine.

Details of this machine include:

Windows 7 64 Bit Home Premium Service Pk. 1

Dell Studio XPS 9100

Intel® Core™ i7 CPU X980 @ 3.33GHz

12 GB Ram

4TB Hard Drive Space

This machine was the closest to that of a dedicated server running MongoDB.

MongoDB

The team has been using MongoDB[7] version 2.4.9 for the duration of the project – the newest stable version.

Installation

We have installed it by downloading the MongoDB package from:

http://www.mongodb.org/downloads

and following the installation instructions:

http://docs.mongodb.org/manual/installation/

[7] See http://docs.mongodb.org/manual/ for MongoDB Manual

Running

To run MongoDB, a command prompt should be opened and the path should be set to the location of the MongoDB server.

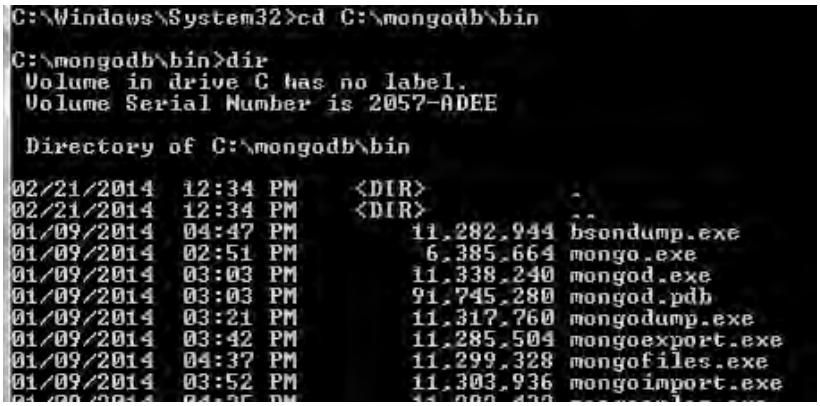
Example:

Once there, the following command must be prompted to allow the server to run while enabling the full text search for querying[8]:

mongod --setParameter textSearchEnabled=true

Importing Data

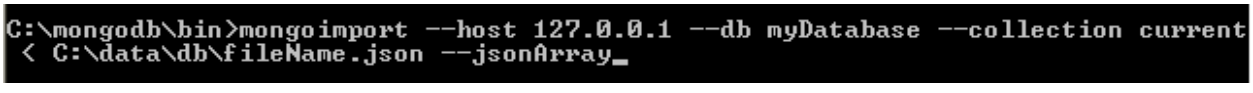
To do an initial import of data, open a command prompt and navigate to the location of MongoDB.



Once there, the following command should be entered to import data:

mongoimport --host myHost --db databaseName --collection collectionName < pathToFile.json

--jsonArray



[8] Please note this is due to full text search being in a beta stage. Once fully released, this command will be incorporated into the run command for mongod

This command specifies the host server, the database and collection to import to, the path of the JSON file to be imported, and allowing data to be imported that has objects on multiple lines.

Please note that there are various parameters that can be used with mongoimport. Type mongoimport for a list of all possible commands.

Importing directly through MongoDB will be quicker for the initial transfer of data. After the majority of files are imported to MongoDB, the API can be used.

When the data is imported to the collection, the shadow collection should also be imported. Please see the "Setting Up Vermongo" section under Design.

Creating Full Text Index

Once the data has been imported, the index must be created for future querying. Open another command prompt, navigate to the MongoDB folder, and open a Mongo client by typing *mongo.* Once there, navigate to the collection by entering the following[9]:

use <databaseName>

db.<collectionName>

Once in the correct collection, the following command will create the index:

*db.<collectionName>.ensureIndex({ "$**":"text" } , { name:"TextIndex" })*

Once the index is created, it will maintain itself as new documents are added to the collection.

API

The team has been using Microsoft Visual Studio 2012 Professional for the duration of the project.

Drivers

Two drivers were used for the implementation of the C# MongoDB API. These two drivers are the MongoDB C# Driver version 1.8.3, and the Newtonsoft JSON.net Driver version 6.0.1 – both accepted by MongoDB.

[9] Note: anything with <> will be replaced with the proper name

To install these two drivers, follow the subsequent steps:

Open MongoDB API Solution in Visual Studio

Click Tools > Library Package Manager > Package Manager Console

Enter the following two commands:

Install-Package mongocsharpdriver

Install-Package Newtonsoft.Json

Package contents can be found at http://www.nuget.org/packages/mongocsharpdriver/ and http://www.nuget.org/packages/Newtonsoft.Json/

Implementation – API

The API has been developed to handle different categories of interaction. These categories include:

- Test methods (to read in data for use in API representation)
- Connect/Disconnect from MongoDB server
- Insert new documents into the database
- Update existing documents in the database
- Querying
- Support Methods (conversions, etc.)

Test Methods

There are two test methods that are represented in the API: these two methods are to symbolize data passed to the API. Currently they read data in from a JSON file, but in the future this data will most likely be passed from the user interface.

Insert Multiple Documents from File

void insertMultipleDocsFromFile(string fileName, MongoCollection<BsonDocument> collectionName, MongoCollection<BsonDocument> shadowCollectionName)

This method reads in data from a passed filename and parses it into JSON strings. It uses bracket pairing with a count value as well as confirming bracket pairs exist to indicate where a JSON object begins and ends. Once this is completed, it is converted to a BSON Document [convertStringToDoc] and inserted into the passed collection [insertDoc].

We expect this method to be minimally used in the future, as any large data imports are recommended to go through MongoDB specifically, and updates from the customer will be one document at a time.

Insert Single Document From File

void insertNewDocFromFile(string fileName, MongoCollection<BsonDocument> collectionName, MongoCollection<BsonDocument> shadowCollectionName)

This method is one that most closely resembles what would be used in future API installments. This method takes one document at a time and reads it into a JSON string, which is then converted to a BSON Document [convertStringToDoc] and inserted into the passed collection [insertDoc].

Server Methods

void serverConnect()

void serverDisconnect(MongoServer serverName)

The serverConnect and serverDisconnect methods both allow communication to occur between the client (API) and server (MongoDB). The serverConnect method creates a new client, gets a server for the client, and establishes a connection. The serverDisconnect breaks this connection by disconnecting the passed server name.

Insert Methods

Insert Document into Collection

void insertDoc(BsonDocument document, MongoCollection<BsonDocument> collection, MongoCollection<BsonDocument> shadowCollection)

To insert a new document for the first time, the method takes the document and uses an insert command on the collection to add it. To handle versioning, a new BSON document is created with the documents _id, a version variable set to 0, and a snapshot array to store further versions of the document. This BSON document is added to the shadow collection.

Update Methods

Update Document in Collection

void updateDoc(string id, string fieldToUpdate, string update, MongoCollection<BsonDocument> collection, MongoCollection<BsonDocument> shadowCollection)

This method is used to update documents in a given collection. The method uses the passed _id of a document, queries it, and first saves the current version [saveVersion] of the document in the shadow collection. Then the method updates the document's specified field with the update value in the main collection. If no document was found with the _id, it prints an error.

Save Version of an Updated Document

void saveVersion(BsonDocument version, string id, MongoCollection<BsonDocument> shadowCollection)

To ensure updates do not overwrite information for a document, this method is used to store past versions of a document. When an update is to occur [updateDoc], the current version of the document is passed, along with its _id. This _id is searched in the shadow collection, and the current document is "pushed" into the next snapshot array position. The version variable is incremented to indicate what version has been stored.

Querying Methods

Query One Pre-Defined Field

string[] queryOneField(string field, string itemToQuery, MongoCollection<BsonDocument> collection)

This method indicates that searches can be done on specific passed fields. The method gathers the field to search and parameter, and searches all representations of that data in the collection. A string array of matching _id fields is returned.

Although this will not be used extensively in future development (due to knowing the field), this method is proof that queries can be done on a specific field.

Query One to Many Unknown Parameters

List<Results> fullTextSearch(string[,] searchFields, MongoCollection<BsonDocument> collection, MongoDatabase database)

This method allows a user to search for keywords without knowing what fields they are looking for. The method passes a two-dimensional array (row 1 with keywords, row 2 indicating a "type" and/or "context ID" if necessary) to search. The method confirms the user access filter [confirmAccess], and converts the keywords to a search string [convertSearchString], which is then used to build the query. The query is then run.

The results of the query return as an array, so each returning document is converted to a JSON string. This string is passed to be filtered [filterDocument], and the Results object for that document is returned. Each document is then put on the Returns List and returned to the main class.

This method is the one that will be most widely used in future development.

Support Methods

In the API there are various methods known as "support methods" that serve as converters, and other basic tasks that support the bulk API methods.

Convert JSON String to BSON Document

BsonDocument convertStringToDoc(string line)

To add documents to a collection, the document must be in BSON format. This method takes a JSON string, makes it into a JSON object, which can then be converted to a BSON document and returned.

Confirm Access for Information Security

IMongoQuery confirmAccess()

To ensure sensitive data is not transferred between customers, an access filter is created for all queries. Any time a query [fullTextSearch] is run, this method is invoked to get the filter. This method checks three global private variables (agency, county and state) to see which exist. For those that do exist (and are not null), a query command is created. This is passed back to the query method, which is then implemented as a filter in the full text search.

Convert Search Parameters into Searchable String

string convertSearchString(string[,] fieldsToSearch)

Since there can be multiple parameters to search in any given query [fullTextSearch], the parameters must be formatted to one able-to-be-queried string. This method loops through all passed parameters and formats them into one string that an "and" operation can be implemented in the full text search. This string is returned for implementation in query.

Print Document to User

void printDocToUser(BsonDocument document)

In most searches, only specific fields are necessary to be returned. However, if a full document and all contents must be printed, this method is invoked. It is passed a document, and each key-value pair is printed to the user.

Print Query Results to User

void printResults(List<Results> pathOfHit, string[,] fieldsToSearch)

When a query [fullTextSearch] is run, the results are saved as a linked list of Results objects [Results]. To indicate to the user what the query results were, this method is invoked. The method prints (for each document that was returned) the _id of the document, the report numbers associated in that document, and the path(s) to each search parameter that was requested.

Filter Document for Pertinent Data

Results filterDocument(BsonValue document, string[,] fieldsToSearch)

When a document is returned as a query result [fullTextSearch], this method filters through each document to return pertinent data to the user. It passes back a Results object [Results] with the _id, path(s) to each search parameter that was requested, and the reports associated with the document.

The method starts by converting the document to a JSON string to be parsed. This string is then parsed by keeping track of bracket pairs (for each level of nesting for data), key-value pairs, and path information for each value. The method loops through the string by looking for bracket pairs (which increments a count for nesting level), elements (either keys or values), and stores elements as either path elements or value elements. When a value is found that matches a search value, the current path data is stored for that searched field. When a closing bracket is found, the current path has elements removed that were contained in that level of nesting, as they are not relevant to the next element's path. When a type or contextID is found, it is saved in the path to indicate specific data about the element returned: for example, a last name can belong to a defendant or officer – the type and contextID will indicate which it is.

This method is put together in such a way that it will work with the current data format. However, when the full text search in MongoDB is rolled out after the beta stage, we expect changes to the C# driver that will allow filtering to be done with built-in methods.

Return Class

The Results class contains necessary information about queried documents. It contains the following variables:

public string IdDoc

public string[][] itemPathDoc

public string[] ReportNumbers

Each document that is queried should return the _id of the document (to be referenced later), the paths of where searched data was found (as a jagged array with as many rows as there are search parameters), and all report numbers associated with that document.

Testing

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	7-Apr-2014	Initial Testing Documentation	Babak Esfandiari Stefan Schielke Nicole Saat
2	9-Apr-2014	Querying Testing	Babak Esfandiari Stefan Schielke Nicole Saat

Introduction

MongoDB is an open-source, NoSQL database that syncs with JSON data. It provides flexible capabilities for storage, as well as compatibility with a wide range of programming languages.

The aim of this project is to provide a “proof of concept” representation of MongoDB with specific sample data sets. This concept will either confirm or refute the client’s request to populate MongoDB with existing JSON BLOB data and provide querying functionality.

Glossary

Collection: Similar to a table in a relational database – a group of documents in MongoDB
Document: Similar to a row or record in a relational database – an instance in MongoDB. This will refer to each JSON BLOB
Field: Similar to a column in a relational database – a category in MongoDB
_id: Unique identifier in MongoDB. Cannot be changed after creation and cannot be duplicated
Customer: Agency using the product
Users: Employees at the agency using the product

Key Notes

When doing a Full Text Index search, there is a 16MB restriction return on documents. This has been noted by the team to be approximately 1920 records. In the future, the skip() option can be used to skip found documents and re-start the search from that value to attain all results. This is not a factor for querying with a known field.

When doing Full Text Index searches, the database automatically caches searches. Therefore, two test cases were run four times to find the difference in time to search: Test Case 25, 28, 30, and 31.

Test Cases

MongoDB

Test Case #: MI1	Test Case Name: MongoDB Insert 1
Short Description: Insert of data into MongoDB. Insert provided JSON BLOB data into a test collection	

Pre-conditions:

- Creation of the test database and collection

Step	Action	Expected System Response	Pass / Fail	Comment
1	Move to correct collection	System will move to collection in which JSON data will be inserted	Pass	
2	Run a count() command to ensure collection is empty	System will output 0 to indicate no documents in collection	Pass	
3	Open mongoimport terminal	Terminal is opened and indicates it is connected to the database	Pass	
4	Run command to enter JSON file into collection	Terminal will advise that the data has been entered	**FAIL**	Researched issue. Found that new line characters or return characters can cause issues converting to BSON. Must use extra command (--jsonArray) for import
5	Re-run command to enter JSON file into collection	Terminal will advise the data has been entered	Pass	

Post-conditions:

1. JSON data is successfully added to the collection

API Querying

Please note: All API Querying occurs in the following way:

- Index is a Full Text Search with unknown field
- Non-Index is a query with known field. Filter for Non-Index is path to field, as necessary for nested searches in MongoDB

Small Collection

There are 3,998 documents in the small database, all of randomly generated “true” values.

Test Case 1

	Test Case	Defendant named Barry	
Index	**Search Value**	BARRY	
	Filter	Defendant	
	Result	13	Note: Vehicle Registration is also included under Defendant, so larger value is found with Full Text Search
	Time Taken	00:00:00.1892078	
Non-Index	**Search Value**	BARRY	
	Filter	values.DomainObjects.values.FirstName values.DomainObjects.values.LastName	Note: Since the path is necessary, team had to search both First Name and Last Name for results
	Result	1 6	
	Time Taken	00:00:00.0866219 00:00:00.0894739	
Findings	The search for the Defendant.name (First or Last) on the non-indexed data provides the data quicker on a small data set a we are specifying the path where to look in the search field. As the database gets larger these results flip and the indexed database results become quicker.		

Test Case 2

	Test Case	Defendant named John	
Index	**Search Value**	JOHN	
	Filter	Defendant	
	Result	12	Note: Defendant is also included under Vehicle Registration, so larger value is found with Full Text Search
	Time Taken	00:00:00.2140707	
Non-Index	**Search Value**	JOHN	
	Filter	values.DomainObjects.values.FirstName values.DomainObjects.values.LastName	Note: Since the path is necessary, team had to search both First Name and Last Name for results
	Result	6 1	
	Time Taken	00:00:00.0870904 00:00:00.0816852	
Findings	The search for the Defendant.name (First or Last) on the non-indexed data provides the data quicker on a small data set a we are specifying the path where to look in the search field. As the database gets larger these results flip and the indexed database results become quicker.		

Test Case 3

	Test Case	Leslie (no specification)	
Index	**Search Value**	LESLIE	
	Filter		
	Result	8	Note: With no filter, results were returned for both Officer and Defendant
	Time Taken	00:00:00.1342891	
Non-Index	**Search Value**	LESLIE	
	Filter	values.DomainObjects.values.FirstName values.DomainObjects.values.LastName	Note: Since the path is necessary, team had to search both First Name and Last Name for results. This search was only for Defendant
	Result	2 1	
	Time Taken	00:00:00.0849495 00:00:00.0814729	
Findings	The search for the Defendant.name (First or Last) on the non-indexed data provides the data quicker on a small data set a we are specifying the path where to look in the search field. As the database gets larger these results flip and the indexed database results become quicker.		

Test Case 4

	Test Case	Officer name Sally	
Index	**Search Value**	SALLY	
	Filter	Officer	
	Result	7	Note: Filter for Officer will only return Officer name (no other locations for data)
	Time Taken	00:00:00.2396340	
Non-Index	**Search Value**	SALLY	
	Filter	values.DomainObjects.values.IssuingOfficer.FirstName values.DomainObjects.values.IssuingOfficer.LastName	Note: Since the path is necessary, team had to search both First Name and Last Name for results
	Result	6 1	
	Time Taken	00:00:00.0888608 00:00:00.0824019	
Findings	The search for the Officer.name (First or Last) on the non-indexed data provides the data quicker on a small data set a we are specifying the path where to look in the search field. As the database gets larger these results flip and the indexed database results become quicker.		

Test Case 5

	Test Case	Color Blue	
Index	**Search Value**	BLUE	
	Filter		
	Result	437	Note: With no filter, results were returned for Name, Eye Color, and Vehicle Color
	Time Taken	00:00:04.2569686	
Non-Index	**Search Value**	BLUE	
	Filter	values.DomainObjects.values.EyeColor.values.Description	Note: Since the path is necessary, team searched only Eye Color
	Result	330	
	Time Taken	00:00:00.3943691	
Findings	The indexed data returns a larger data set as the full index will search through every field searching for the value 'Blue'. For the non-indexed as the path must be specified, in this case 'Eye Color', a smaller data set is returned in a shorter amount of time.		

Test Case 6

	Test Case	Color Brown	
Index	**Search Value**	BROWN	
	Filter		
	Result	1332	Note: With no filter, results were returned for Name, Eye Color, Hair Color, and Vehicle Color
	Time Taken	00:00:08.1589170	
Non-Index	**Search Value**	BROWN	
	Filter	values.DomainObjects.values.HairColor.values.Description	Note: Since the path is necessary, team searched only Hair Color
	Result	659	
	Time Taken	00:00:00.7437299	
Findings	The indexed data returns a larger data set as the full index will search through every field searching for the value 'Brown'. For the non-indexed as the path must be specified, in this case 'Hair Color', a smaller data set is returned in a shorter amount of time.		

Test Case 7

	Test Case	Color Green	
Index	**Search Value**	Green	
	Filter		
	Result	767	Note: With no filter, results were returned for Name, Eye Color, and Vehicle Color
	Time Taken	00:00:06.1412973	
Non-Index	**Search Value**	Green	
	Filter	values.DomainObjects.values.Color.values.Description	Note: Since the path is necessary, team searched only Vehicle Color
	Result	140	
	Time Taken	00:00:00.2136043	
Findings	The indexed data returns a larger data set as the full index will search through every field searching for the value 'Green'. For the non-indexed as the path must be specified, in this case 'Vehicle Color', a smaller data set is returned in a shorter amount of time.		

Test Case 8

	Test Case	Vehicle with Most Violations	
Index	**Search Value**	Array of all listed Vehicles	Team could not search for Vehicle with Wild Card and pool values based on each model
	Filter	Vehicle	
	Result	105 / Lexus	Note: each vehicle was queried, and a max was taken
	Time Taken	00:00:25.4179783	
Non-Index	**Search Value**	Array of all listed Vehicles	
	Filter	values.DomainObjects.values.Make.values.Description	Note: Since the path is necessary, team searched only Vehicle Make
	Result	105 / Lexus	
	Time Taken	00:00:06.6981264	
Findings	A loop through an array of values had to be queried on, and then a max value calculated as each element was queried. Both databases provided the same results with the non-indexed data been quicker.		

Test Case 9

	Test Case	Search both Brown and Cleveland	
Index	**Search Value**	BROWN, CLEVELAND	
	Filter		No Filter for either value
	Result	3	
	Time Taken	00:00:00.2884578	
Non-Index	**Search Value**		Cannot Search Two Values at Once
	Filter		
	Result		
	Time Taken		
Findings	The non-indexed database is not possible to query based on multiple items. As a result only the non-indexed database could return results.		

Test Case 10

	Test Case	Search both Brown and Ford	
Index	**Search Value**	BROWN, FORD	
	Filter		No Filter for either value
	Result	34	
	Time Taken	00:00:00.5833813	
Non-Index	**Search Value**		Cannot Search Two Values at Once
	Filter		
	Result		
	Time Taken		
Findings	The non-indexed database is not possible to query based on multiple items. As a result only the non-indexed database could return results.		

Test Case 11

	Test Case	Who has more Violations – Males or Females?	
Index	**Search Value**	MALE FEMALE	These were done in separate searches, and the values were then compared
	Filter	Gender	
	Result	MAXED 1920 MAXED 1920	Note: There is a 16MB limit return on searches, so each capped out
	Time Taken	00:00:23.6080721	
Non-Index	**Search Value**	MALE FEMALE	These were done in separate searches, and the values were then compared
	Filter	values.DomainObjects.values.Gender.values.Description	
	Result	M: 2942 F: 3043	
	Time Taken	00:00:06.3912234	
Findings	The gender is also stated in the Vehicle Attributes for a person. As a result the findings are greater than the number of records in the database. Each gender had to be looped over in order to obtain a count. The counts were then compared to obtain a record of the greatest violations. To obtain an accurate number the count can be divided in half and the resulting count will be the number of violations by sex.		

Test Case 12

	Test Case	Blue Vehicles	
Index	**Search Value**	BLUE	
	Filter	Vehicle	
	Result	119	
	Time Taken	00:00:04.2777163	
Non-Index	**Search Value**	BLUE	
	Filter	values.DomainObjects.values.Color.values.Description	
	Result	119	
	Time Taken	00:00:00.1996922	
Findings	The search for the Vehicles.Blue on the non-indexed data provides the data quicker on a small data set as we are specifying the path where to look in the search field. As the database gets larger these results flip and the indexed database results become quicker.		

Test Case 13

	Test Case	Red Vehicles	
Index	**Search Value**	RED	
	Filter	Vehicle	
	Result	137	
	Time Taken	00:00:04.9601473	
Non-Index	**Search Value**	RED	
	Filter	values.DomainObjects.values.Color.values.Description	
	Result	137	
	Time Taken	00:00:00.2060518	
Findings	The search for the Vehicles.Red on the non-indexed data provides the data quicker on a small data set as we are specifying the path where to look in the search field. As the database gets larger these results flip and the indexed database results become quicker.		

Large Collection

There are ~5 million documents in the large database, all have randomly generated “true” values.

Test Case 14

	Test Case	All instances of Brown	
Index	**Search Value**	BROWN	
	Filter		
	Result	MAXED 1920	
	Time Taken	00:54:45.3398343	
Findings	When searching for Brown in the full text search, all record containing ‘Brown’ will be returned. This will include documents where ‘Brown’ is in Eye Color, Hair Color, Last Name, and Vehicle Color. As a result the data set becomes very large and overloads the 16MB capacity.		

Test Case 15

	Test Case	All instances of Cleveland	
Index	**Search Value**	CLEVELAND	
	Filter		
	Result	1415	
	Time Taken	00:01:56.7082507	
Findings	When searching for Cleveland, it could have been the name of a defendant or officer – by not specifying the parameter of defendant or officer, all possible documents were returned. This search took just under 2 minutes for 5 million records.		

Test Case 16

	Test Case	All instances of Brown and Cleveland	
Index	**Search Value**	BROWN, CLEVELAND	
	Filter		
	Result	496	
	Time Taken	00:51:52.6548793	
Findings	In this case, two parameters were entered and an "and" operation was done in regards to returning results. When this search occurred, both parameters had to be satisfied. Since brown could be multiple values (name, eye color, hair color, vehicle color), it could have slowed the search process – the index would have to search all documents to compare. As compared with *Test Case 14* and *Test Case 15* (Brown and Cleveland separately), the results only differed by 4 minutes. This search took just under 52 minutes to execute for 5 million records.		

Test Case 17

	Test Case	Defendant named Reed	
Index	**Search Value**	REED	
	Filter	Defendant	
	Result	1409	
	Time Taken	00:02:08.5668583	
Findings	When searching for Reed, it could have been the name of a defendant or officer – by specifying the parameter of defendant, fewer documents were returned. This search took 2 minutes for 5 million records.		

Test Case 18

	Test Case	Driver's License Number	
Index	**Search Value**	156231218	
	Filter		
	Result	1	
	Time Taken	00:00:00.0942730	
Findings	Since only one unique driver's license was searched in this case, this search took a very short amount of time. Only one records out of 5 million was returned, and it took .09 seconds to execute.		

Test Case 19

	Test Case	Officer named Cleveland	
Index	**Search Value**	156231218	
	Filter	Officer	
	Result	624	
	Time Taken	00:02:15.3536764	
Findings	When searching for Cleveland, it could have been the name of a defendant or officer – by specifying the parameter of officer, fewer documents were returned. This search took 2 minutes for 5 million records.		

Test Case 20

	Test Case	Report Number	
Index	**Search Value**	10000085	
	Filter	ReportNumber	
	Result	2	
	Time Taken	00:00:00.3120430	
Findings	Since only one unique report number was searched in this case (that could be affiliated with up to two other cases), this search took a very short amount of time. Only two records out of 5 million were returned, and it took .3 seconds to execute.		

Test Case 21

	Test Case	Incident Date	
Index	**Search Value**	2012-11-13T00:00:00	
	Filter	IncidentDate	
	Result	213	
	Time Taken	00:21:10.8545360	
Findings	In this search, all incidents were queried for the data November 13, 2012. This search took a comparable amount of time for one search value, at 21 minutes.		

Test Case 22

	Test Case	Who has more Violations – Males or Females?	
Index	**Search Value**	MALE FEMALE	These were done in separate searches, and the values were then compared
	Filter	Gender	
	Result	MAXED 1920 MAXED 1920	Note: There is a 16MB limit return on searches, so each capped out
	Time Taken	02:36:43.0056389	
Findings	The gender is also stated in the Vehicle Attributes for a person. As a result the findings are greater than the number of records in the database. Each gender had to be looped over in order to obtain a count. The counts were then compared to obtain a record of the greatest violations. To obtain an accurate number the count can be divided in half and the resulting count will be the number of violations by sex.		

Test Case 23

	Test Case	Vehicle with Most Violations	
Index	**Search Value**	Array of all listed Vehicles	Team could not search for Vehicle with Wild Card and pool values based on each model
	Filter	Vehicle	
	Result	MAXED 1920	Note: each vehicle was queried, and a max was taken
	Time Taken	07:38:26.2062748	
Findings	In this search, an array of vehicles was queried against the collection to see which had the most violations. To achieve this, each vehicle had to be queried one at a time, and the results were then compared to find a max value, which was then returned. This search took over 7.5 hours to complete due to the multitude of searches and filters to go through with a database of approx. 5 million records.		

Test Case 24

	Test Case	Blue Vehicles	
Index	**Search Value**	BLUE	
	Filter	Vehicle	
	Result	1089	
	Time Taken	00:44:33.0032618	
Findings	In this search, the colour “blue” was filtered by vehicle color. Therefore, no eye color/names were returned. This search took a significant amount of time, since the index had to first find blue, then filter by vehicle.		

Test Case 25

Note: This search was done twice in a row to indicate change in speed when caching

		NOT CACHED		CACHED	
	Test Case	All Instances of James		All Instances of James	
Index	**Search Value**	JAMES		JAMES	
	Filter	Agency, County, State		Agency, County, State	
	Result	807		807	
	Time Taken	00:04:38.0305041		00:00:05.9888026	
Index	**Search Value**	JAMES		JAMES	
	Filter	County, State	Note: No Agency Filter	County, State	Note: No Agency Filter
	Result	1611		1611	
	Time Taken	00:04:48.8355858		00:00:16.7921402	
Index	**Search Value**	JAMES		JAMES	
	Filter	State	Note: No Agency or County Filter	State	Note: No Agency or County Filter
	Result	MAXED 1920		MAXED 1920	
	Time Taken	00:05:01.4363549		00:00:28.9300647	
Findings	To account for the “overlord scenario”, tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added. In this search, a second search was run directly after which cached the search results from the “not cached” search. This decreased the time to search significantly, from 4 minutes 38 seconds to only 5 seconds.				

Test Case 26

	Test Case	All Instances of Cleveland	
Index	**Search Value**	CLEVELAND	
	Filter	Agency, County, State	
	Result	714	
	Time Taken	00:02:13.4008224	
Index	**Search Value**	CLEVELAND	
	Filter	County, State	Note: No Agency Filter
	Result	1415	
	Time Taken	00:00:09.0590592	
Index	**Search Value**	CLEVELAND	
	Filter	State	Note: No Agency or County Filter
	Result	MAXED 1920	
	Time Taken	00:00:12.0753874	
Findings	To account for the “overlord scenario”, tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added.		

Test Case 27

	Test Case	All Instances of Brown	
Index	**Search Value**	BROWN	
	Filter	Agency, County, State	
	Result	MAXED 1920	
	Time Taken	00:57:18.3984168	
Index	**Search Value**	BROWN	
	Filter	County, State	Note: No Agency Filter
	Result	MAXED 1920	
	Time Taken	00:53:12.8640815	
Index	**Search Value**	BROWN	
	Filter	State	Note: No Agency or County Filter
	Result	MAXED 1920	
	Time Taken	00:58:12.6472907	
Findings	To account for the “overlord scenario”, tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added. All instances of brown can return for multiple fields (name, eye color, hair color, vehicle color), and so this search maxed out on all three searches		

Test Case 28

Note: This search was done twice in a row to indicate change in speed when caching

		NOT CACHED		CACHED	
	Test Case	All Instances of Dale		All Instances of Dale	
Index	**Search Value**	DALE		DALE	
	Filter	Agency, County, State		Agency, County, State	
	Result	957		957	
	Time Taken	00:03:20.2236574		00:00:06.1238212	
Index	**Search Value**	DALE		DALE	
	Filter	County, State	Note: No Agency Filter	County, State	Note: No Agency Filter
	Result	MAXED 1920		MAXED 1920	
	Time Taken	00:00:12.2037331		00:00:11.9062245	
Index	**Search Value**	DALE		DALE	
	Filter	State	Note: No Agency or State Filter	State	Note: No Agency or County Filter
	Result	MAXED 1920		MAXED 1920	
	Time Taken	00:00:11.8481232		00:00:11.8999094	
Findings	To account for the “overlord scenario”, tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added. In this search, a second search was run directly after which cached the search results from the “not cached” search. This decreased the time to search significantly, from 3 minutes 20 seconds to only 6 seconds.				

Test Case 29

	Test Case	All Instances of Roxy	
Index	**Search Value**	ROXY	
	Filter	Agency, County, State	
	Result	287	
	Time Taken	00:00:59.4718520	
Index	**Search Value**	ROXY	
	Filter	County, State	Note: No Agency Filter
	Result	573	
	Time Taken	00:01:03.0616869	
Index	**Search Value**	ROXY	
	Filter	State	Note: No Agency or State Filter
	Result	1753	
	Time Taken	00:01:13.6014596	
Non-Index	**Search Value**	ROXY	Note: Roxy was done just to compare search times with Index in Large Collection
	Filter	values.DomainObjects.values.FirstName	
	Result	2638	
	Time Taken	00:11:43.2630435	
Findings	To account for the "overlord scenario", tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added. A no-index search of Roxy was run to indicate the time difference for non-indexed searches in large collections. Results were 10 minutes different, with indexed being quicker by 10 times.		

Test Case 30

Note: This search was done twice in a row to indicate change in speed when caching

		NOT CACHED		CACHED	
	Test Case	All Instances of Gus		All Instances of Gus	
Index	**Search Value**	GUS		GUS	
	Filter	Agency, County, State		Agency, County, State	
	Result	310		310	
	Time Taken	00:01:23.4977288		00:00:02.0542500	
Index	**Search Value**	GUS		GUS	
	Filter	County, State	Note: No Agency Filter	County, State	Note: No Agency Filter
	Result	632		632	
	Time Taken	00:00:04.0031085		00:00:03.8905900	
Index	**Search Value**	GUS		GUS	
	Filter	State	Note: No Agency or State Filter	State	Note: No Agency or County Filter
	Result	1877		1877	
	Time Taken	00:00:11.2874361		00:00:11.2258087	
Findings	To account for the “overlord scenario”, tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added. In this search, a second search was run directly after which cached the search results from the “not cached” search. This decreased the time to search significantly, from 1 minutes 23 seconds to only 2 seconds.				

Test Case 31

Note: This search was done twice in a row to indicate change in speed when caching

		NOT CACHED		CACHED	
	Test Case	All Instances of John		All Instances of John	
Index	**Search Value**	JOHN		JOHN	
	Filter	Agency, County, State		Agency, County, State	
	Result	545		545	
	Time Taken	00:03:28.8898696		00:00:06.4007438	
Index	**Search Value**	JOHN		JOHN	
	Filter	County, State	Note: No Agency Filter	County, State	Note: No Agency Filter
	Result	1144		1144	
	Time Taken	00:03:41.1002118		00:00:18.3666524	
Index	**Search Value**	JOHN		JOHN	
	Filter	State	Note: No Agency or State Filter	State	Note: No Agency or County Filter
	Result	MAXED 1920		MAXED 1920	
	Time Taken	00:03:53.2980738		00:00:30.1894926	
Non-Index	**Search Value**	JOHN		Note: John was done just to compare search times with Index in Large Collection	
	Filter	values.DomainObjects.values.FirstName			
	Result	5153			
	Time Taken	00:15:10.3288994			

Findings	To account for the “overlord scenario”, tests were run with filters by agency (all 3), county (last 2), and state (last 1). As noted, results became larger the more customers were added. In this search, a second search was run directly after which cached the search results from the “not cached” search. This decreased the time to search significantly, from 3 minutes 28 seconds to only 6 seconds. A no-index search of John was run to indicate the time difference for non-indexed searches in large collections. Results were 12 minutes different, with indexed being quicker by 4 times.

Lessons Learned

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	8-Apr-2014	Initial Lessons Learned	Babak Esfandiari Stefan Schielke Nicole Saat

Introduction

MongoDB is an open-source, NoSQL database that syncs with JSON data. It provides flexible capabilities for storage, as well as compatibility with a wide range of programming languages.

This “lessons learned” document illustrates the positive and negative conclusions found by the team throughout the project.

Glossary

Collection: Similar to a table in a relational database – a group of documents in MongoDB

Document: Similar to a row or record in a relational database – an instance in MongoDB. This will refer to each JSON BLOB

Field: Similar to a column in a relational database – a category in MongoDB

_id: Unique identifier in MongoDB. Cannot be changed after creation and cannot be duplicated

Customer: Agency using the product

Users: Employees at the agency using the product

Parameter of Hit: Field in which queried data was found

Key Notes

None

Lessons Learned

There are various “lessons learned” found by the team during the project duration. These can be categorized into the following:

- Structure of Data
- Queries and Sample Cases
- Azure-Hosting
- Full Text Search

Structure of Data

The data structure is key to the efficiency of MongoDB querying and indexing. To search for values without knowing the field in advance, a full text search must be implemented (fields cannot use wildcard expressions). Since the full text search indexes all data in a collection (excluding “stop words” like a, an, the, etc.), this can cause the size to grow exponentially if non-pertinent data is included.

Data organization is the largest factor in querying efficiency. Since the project has developed into MongoDB being specifically a search bank, we would change the organization of the data.

Currently when queries are returned, the resulting document is parsed as a string to retrieve important data (path information, etc.). Nested data that is inside more than one object cannot be retrieved natively in the MongoDB C# API, and cannot be searched by those parameters (for example, Defendant’s Last Name). If this data was organized differently, the MongoDB C# API can be used more effectively, and data can be retrieved as BSON values rather than parsed strings. By having only one level of nested data (rather than multiple objects and/or arrays inside other objects and/or arrays), not only will the index be smaller and the querying be faster, but also the API can then parse data out of the object in BSON format.

For example, to retrieve a Defendant’s last name, the system is currently set up with this path:

values.DomainObjects.values.LastName

If data was restructured, the path could be:

Defendant.LastName

We recommend structuring the data with only one level of nesting. Please see below for an example:


```
                    LEVEL 1             LEVEL 2
{
        _id : “JKE10001-XYZ101-NY-XYZ101-NY-10000004”,
        ReportNumbers : [
                            "10000004",
                            "10000005",
                            "10000006"
                            ],
        Defendant : {
                            FirstName : “John”,
                            LastName : “Doe”,
                            Gender : “Male”,
                            GenderCode : “M”,
                            EyeColor : “Brown”
                                    …
                            },
        Vehicle: {
                            RegistrationNumber : “9HP SX6”,
                                    …
                    }
        …
}
```

Queries and Sample Cases

For the duration of the project, the team developed in sprints. For the first one and a half months, the team focused on learning MongoDB functionality, and developed an understanding of the NoSQL Database. Due to the amount of learning required (as MongoDB and C# were both new to team members), the team left future sprint details until a later time, unless deemed necessary to assist in the current sprint.

Building of the API and querying was allocated two sprints, beginning the last week of February. Due to the size and scope of these tasks, details were pushed into the last "cushion" sprint. The team noted that, when building the API to return the path of query results, they were required to parse query results as a string.

If the team accounted for specific queries in the early development of the API, they may have saved time creating this parsing method by not making changes to the method after errors were encountered.

Azure-Hosting

One of the requirements for the project was to ensure the final product was Azure-compatible. The MongoDB website notes it is, as well as Azure has noted "promotional codes" for MongoDB on their website[10]. The team tried various ways to coordinate a trial version of Azure, whether through an educational license or through Visual Statement.

Due to restrictions on time, the team was unable to incorporate this into the final product. If repeated, the team would have exhausted avenues for Azure hosting earlier.

Full Text Search

The full text search was a key requirement for the project, since field searches did not include wildcards – therefore, fields had to be known to search data. Since the MongoDB full text search function was still in the beta stage, the team was required to use other parsing techniques rather than pre-defined methods native to the MongoDB C# driver.

The team faced various difficulties with the full text search, including filtering and path returns of where the data was found. These difficulties included querying and data structure, as noted above. However, when the full text search has developed out of its beta stage and is rolled out (version 2.6 is currently in unstable format), more methods to fulfill these tasks without parsing a string and working around the returned data are expected to develop.

[10] Please note the client contacted Azure, and they advised the promotional code was not applicable at this time.

Future

Document History

Revision History

Revision #	Revision Date	Description of Change	Author(s)
1	9-Apr-2014	Initial Future Items	Babak Esfandiari Stefan Schielke Nicole Saat

Future of MongoDB Project

Now that the project has been completed, the team would like to advise of opportunities for future development.

MongoDB version 2.6 was released on Tuesday, April 8, 2014, which allows for more functionality and changes to the current project. Relevant tools that have been incorporated in MongoDB version 2.6[11] are:

Full Text Search

- Text Search is incorporated by default: therefore does not need to be declared when opening MongoDB server.
- $text operator can be used in the aggregation pipeline command

Insert and Update Improvements

- When updating a document, the order of fields is now preserved: originally, the order changed based on the update
    - This was of little importance to the team as the NoSQL format will still return specified document information as needed for the queries

Security Improvements

- SSL connections between client and server
- Improved client authentication

Query Engine Improvements

- Index Intersection created to use more than one index
- Index filters incorporated

[11] Please see http://docs.mongodb.org/manual/release-notes/2.6/

Appendix

Weekly Documentation

Week 1

Week 1: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:

During the past week, our team has started to gain information and complete housekeeping tasks to prepare for the coming weeks. We have met with the client, gathered pertinent information in regards to the project, and are prepared to start documentation and product creation.

Task Completion from Last Week

- Team Meeting – Thursday, Jan 9
  - Discuss goals and what is expected of each team member
  - Discuss understanding of project outline
  - Discuss using Agile Development methodology
- First Client Meeting – Friday, Jan 10
  - Discussed the following:
    - Questions from team in regards to programming language
    - Current BLOB storage techniques
    - Project expectations in regards to scope and use of final product
- Installation of MongoDB and Visual Studio on personal machines
- Review of client-provided supplemental document titled "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence"

Task for This Week

- Become familiar with MongoDB/Visual Studios, and their functionalities
- Review provided documentation of current JSON BLOB storage from client. Discuss how to apply this information in MongoDB
- Begin to develop a plan in regards to how the program will be implemented, milestones and their expected dates of completion
- Meet with professor to discuss project status and timeframe
- Contact client at end of week to schedule next bi-weekly meeting

Additional Information

None this week.

Known issues / things blocking progress

None this week.

NoSQL Data Storage

Meeting 1 Minutes

January 10, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Stefan Schielke, Bobby Esfandiari, Nicole Saat
Next meeting:	TBA

I. Announcements

Introduction of members for both Team and Client

II. Discussion

- Introduction to Client needs in regards to the project
  - Project is developed to provide querying functionality with MongoDB to existing JSON BLOB Data
- Discussion of current JSON BLOB data
  - Client introduced JSON BLOB and components, including unique identifiers for data
  - Client has sent copy of JSON BLOB Data to team members for review
- Programming languages to be used
  - Client recommends C# for API, but may also accept Java due to lack of knowledge from team members of C#
  - JSON BLOB Data must be stored in MongoDB
- Client recommends team members read “NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence”
  - Team members have received copies and have begun to review
- Discussed brief outline of how project will progress
  - Both Team and Client agree that API Functionality will occur at a later date, and is best to start with simple functionality and work up from there
- Client will have high levels of involvement during process, as client requests continuous updates
  - Client has requested team discuss progress during development as they may be able to provide valuable insight to potential roadblocks before they occur

III. Next Meeting

- Meetings will occur bi-weekly, or more if deemed necessary
- Meetings do not currently have a set time. Team will contact client one week before to schedule an appropriate time to meet

Week 2

Week 2: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:

This week has allowed for the team to make headway toward development of the prototype. The team has developed a projected timeline, as well as discussed how the information will be implemented into MongoDB. The team has become familiar with MongoDB and how to use it in regards to the project scope. This information will be used for the coming sprint staring Monday, January 20.

Task Completion from Last Week

- Become familiar with MongoDB and functionality
  - Reviewed MongoDB video tutorials released
- Reviewed provided documentation of current JSON BLOB storage from client. Discussed how to apply this information in MongoDB
  - Brief discussion with group. Will be defined more in-depth during next sprint
- Begin to develop a plan in regards to how the program will be implemented, milestones and their expected dates of completion
  - Created timeline
- Meet with professor to discuss project status and timeframe
  - Met with Kevin. Project is currently on schedule and is on path toward successful completion
- Contact client at end of week to schedule next meeting

Task for This Week

Note: information in this section will be completed over a two-week iteration.

- Start developing in MongoDB.
- Test MongoDB's functionality with newly-gained knowledge from last iteration
- Transfer existing JSON data provided by client into MongoDB
- Test MongoDB to ensure JSON data is stored correctly
- If necessary, create new JSON sample data to be used in MongoDB

Additional Information

- Requested weekly instead of bi-weekly meetings with client
  - Discussed with client. Bobby will maintain weekly meetings, Stefan and Nicole will be present whenever possible

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#
- Understanding exact scope/minimal requirements of project
  - Discussed with client and reviewed
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
- Change of JSON BLOB data type from current version (known) to new version (unknown)
  - Reviewed. Should be of minimal impact due to NoSQL Data Storage in MongoDB

NoSQL Data Storage

Meeting 2 Minutes

January 17, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari
Next meeting:	January 24, 2014: 10AM

I. Announcements

II. Discussion

- Presented time table and progress report to client
  - Andrew approved time current timetable.
  - He likes the way we've broken down our tasks.
- Discussed future meetings and they are satisfied with alternating formal meetings with the whole team and less formal meetings with only Bobby present on a weekly basis.
- Chad Ludwig (Visual Statement CTO) will most likely be present for the next meeting.
  - Send out an email for confirming the next meeting's date and time.
- Make sure to keep Chad appraised of our progress and include him in emails.
- They don't have any additional test files, so we have to generate our own.
  - The one we were given was randomly generated.
- We don't have to be limited by MongoDB as a solution, so if we come across something better in our research we're free to explore it if we have the time and resources.
- Discussed again why they are not currently using MongoDB and it was due to the team's lack of knowledge in Mongo and inability to dedicate time and resources towards learning it at the time.

III. Next Meeting

Week 3

Week 3: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week was filled with learning the capabilities and functionality of MongoDB. The team was able to successfully import sample data from client and manipulate entered data. In the coming week, the team will be doing additional research of MongoDB's functionality based on specific client criteria.

Task Completion from Last Week
Note: information in this section will be completed over a two-week iteration.

- Start developing in MongoDB.
- Test MongoDB's functionality with newly-gained knowledge from last iteration
- Transfer existing JSON data provided by client into MongoDB
- Test MongoDB to ensure JSON data is stored correctly
- If necessary, create new JSON sample data to be used in MongoDB

Task for This Week

- Research the following to provide status update at next week client meeting:
    - Multi-tenancy options
    - Other Azure-hosted database options
    - Indexing capabilities
    - Versioning capabilities
- Create Analysis Document based on research above an current project status

Additional Information

- Requirements document has been created and reviewed with client
- Team is on time for proposed schedule

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#
    - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage
- Understanding exact scope/minimal requirements of project
    - Resolved. Has been documented in Requirements Document
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
- Spacing of existing data to be imported/exported into and from Mongo
    - Resolved. This has been resolved via the --jsonArray command
- Versioning is not specifically provided as a part of MongoDB's base functionality
- Data provided by client will only be sample data. This will not represent true data from system

Week Three Meeting Agenda

3/28/2014
10:30 AM

Attendees: Andrew Challenger, Auzeb Manzoor, Chad Ludwig
Bobby Esfandiari, Nicole Saat, Stefan Schielke

Topics:

Introduction

Current status of project in regards to milestones document

Requirements Document

Review first draft of requirements doc.

Outline the scope and important definitions in regards to MongoDB

Discuss current system's cumbersome issues for development with MongoDB project

---

API

Discuss future look of API and define API requirements

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage
Meeting 3 Minutes

January 24, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari
Next meeting:	TBA

I. Announcements

Introduction of team to Chad Ludwig

II. Discussion

- Presentation of Requirements document to client
  - Client clarified difference between user and customer.
  - Client advised to not worry about deleting documents, as all documents will be versioned and not dropped.
- Clarification of API/scope of project
  - Client has advised not to worry about any user interface functionality
  - Client advised team does not need to worry about user verification as this will be done at a higher level
  - Client clarified API requirements, and how the API should be present in the final product
    - Client discussed restful API as a possibility. More research needed on API implementation
- Discussion of Indexing and versioning
  - Client would ultimately like the ability to index and search data without knowing what is in the system.
    - Client suggested researching dynamic indexing possibilities, as well as plugins for MongoDB
    - More research needed on Indexing possibilities for next meeting
  - Client advised versioning is crucial to the final product. Any updates should take a snapshot of current version before update is made
    - When querying data only current version should be searched
    - More research is needed on versioning for next meeting
- Request for more sample data
  - Client advised they will provide a few more pieces of sample data, as well as less clean data

III. Next Meeting

- Client will provide more data
- Team will review the following topics and provide status reports:
  - Multi-tenancy options
  - Other Azure-hosted database options
  - Indexing capabilities
  - Versioning capabilities

Week 4

Week 4: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week involved understanding and development of Version Control and Indexing for MongoDB – two of the client-requested requirements for the final project. Team has also completed Sprint 2, and has a head start on requirements of Sprint 2 starting on Monday, February 3.

Task Completion from Last Week
Note: information in this section will be completed over a two-week iteration.

- Researched the following to provide status update at next week client meeting:
  - Multi-tenancy options
    - Discussed with Client
  - Indexing capabilities
    - Discussed with Client
  - Versioning capabilities
    - Discussed with Client
- Have all necessary data stored in MongoDB.
  - Received new test data from Client

Task for This Week
Note: information in this section will be completed over a two-week iteration.

- Implement the following for next week client meeting:
  - Multi-tenancy
  - Indexing capabilities
  - Versioning capabilities
- Incorporate new test data into database
- Incorporate multi-tenancy and compartmentalization of data, and maintain this functionality
- Ensure versioning capabilities are maintained
- Mid-Term Meeting with Client and Kevin

Additional Information

- Team is on time for proposed schedule

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#
  - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
  - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
  - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
  - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result

Week Four Progress Report

3/28/2014
10:30 AM

Attendees: Andrew Challenger, Auzeb Manzoor
Bobby Esfandiari

Topics:

Introduction

Discuss project progress with regards to topics from the last meeting.

Versioning

Look at versioning solution and its feasibility for the project.

Indexing

Discuss indexing strategies and receive feedback.

API

Discuss the MongoDB API and requirements for the project.

Security

Look at the sensitivity of sample data and whether additional security measures are needed.

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage

Meeting 4 Minutes

January 31, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari
Next meeting:	10:30 AM – Friday, February 7h, 2014

I. Discussion

- Versioning
  - The client approves of the use of Vermongo for versioning
  - It seems to meet their requirements
- Indexing
  - They like the idea of combining multi-keys and compound indexing
  - Meets with the requirements of having fewer indexes and not having to specify in advance
  - Would like to know if multiple levels of information can be incorporated into this method of data storage
  - Look into query efficiency
- API
  - They are okay with mongo commands until we can get it incorporated into a C# UI
- Security
  - No sensitive data is being transferred to us for testing
- New test data
  - They are working on putting together new test data (with more metadata) to send to us
- Chad was not present but will review the data and get back to us with any additional comments

II. Next Meeting

- Client will work on providing additional test data
- Still on the agenda for future meetings:
  - Multi-tenancy options
  - Other Azure-hosted database options
  - More research on indexing strategies
  - More versioning testing

Week 5

Week 5: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week involved meeting with the client in regards to researched data to complete the MongoDB sprints. The Client has provided our team with their preferences in regards to MongoDB functionality, which will be implemented and presented to the team at the Mid-Term Review this coming week

Task Completion from Last Week
Note: information in this section will be completed over a two-week iteration.

- Researched the following to provide status update at next week client meeting:
  - Multi-tenancy options
    - Discussed with Client
  - Indexing capabilities
    - Discussed with Client
  - Versioning capabilities
    - Discussed with Client
- Have all necessary data stored in MongoDB.
  - Received new test data from Client

Task for This Week
Note: information in this section will be completed over a two-week iteration.

- Implement the following for next week client meeting:
  - Multi-tenancy
  - Indexing capabilities
  - Versioning capabilities
- Incorporate new test data into database
- Incorporate multi-tenancy and compartmentalization of data, and maintain this functionality
- Ensure versioning capabilities are maintained
- Mid-Term Meeting with Client and Kevin

Additional Information

- Team is on time for proposed schedule

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#
  - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
  - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
  - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
  - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result

Week Five Progress Report

3/28/2014
10:30 AM

Attendees: Andrew Challenger, Auzeb Manzoor, Chad Ludwig
Bobby Esfandiari, Nicole Saat, Stefan Schielke

Topics:

Introduction

Current status of project in regards to milestones document

Requirements Document

Review requirements doc.

Analysis Document

Review first draft of analysis doc.

Discuss:

- Multi-tenancy options
- Version Control - Vermongo
- Indexing

API

Discuss possible test cases necessary for API testing of difficult issues

---

Test Data

Discuss test data provided by Visual Statement to be used in project

---

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage

Meeting 5 Minutes

February 7, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat
Next meeting:	11:30 AM – Friday, February 14th, 2014

I. Discussion

- Versioning
  - Vermongo methodology still to be used
  - Current collection should contain only current documents, shadow should only contain past documents
  - Size of collections is of lower priority than storage paradigm
- Indexing
  - Advanced Multi-Key Indexing is best option for project
  - Size of index is of lower priority than ability to index efficiently
- Security
  - No sensitive data is being transferred to us for testing
  - Risk of having missing data in test cases is understood by both client and team
- New test data
  - Test data should be provided by the end of today

II. Next Meeting

- Midterm review with client and Kevin

Week 6

Week 6: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week involved the Mid-Term review with the client and Kevin. We have completed Sprint 2 and are going in to Reading Break, which will involve learning about C#, Azure, and how to approach issues with Indexing and Data Storage in MongoDB.

Task Completion from Last Week
Note: information in this section will be completed over a two-week iteration.

- Implemented the following
  - Multi-tenancy – shown to the client and approved
  - Versioning capabilities – discussed with client and approved
- Incorporate multi-tenancy and compartmentalization of data, and maintain this functionality
- Ensure versioning capabilities are maintained
- Mid-Term Meeting with Client and Kevin

Task for This Week
Pushed from Last Week:

- Incorporated new test data into database – postponed due to changes in data format for storage
- Indexing capabilities – discussed with the client. Updates will be done to this section based on changes to data storage

- Reading Break – no scheduled sprint
- Team will become familiar with C# for upcoming API sprints
- Team will become familiar with Azure-hosted databases and will evaluate whether this can be incorporated for final product – client has given OK to exclude this depending on the amount of extra work needed
- Team will gather different data from client, as well as remove redundant data from current JSON BLOBs to work with for final product

Additional Information

- Team is on time for proposed schedule

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#
  - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)

- Indexing restrictions based on all choices for full-text search
    - Update: data format for storage will be re-evaluated this coming week to assist with this restriction
- Limitations for speed and storage in regards to indexing and searching
    - Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will be done this sprint

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
    - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
    - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
    - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
    - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
    - Client is aware of these issues and is ready to incorporate it into the final result

NoSQL Data Storage

Meeting 6 Minutes

February 14, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat
	Professor: Kevin O'Neil
Next meeting:	Friday, February 21st, 2014

I. Discussion

- Mid-Term Review
  - Client has expressed satisfaction based on current progress of project and wishes project to continue
- Versioning
  - Vermongo methodology still to be used
  - Versioning with Vermongo has been approved
- Indexing
  - Indexing will be reviewed in upcoming week(s) due to size and structure based on data format
  - Data format will be evaluated, which in turn will have indexing re-evaluated to find most optimal and realistic choice
- Multi-Tenancy
  - Multi-tenancy approach with comparison to unique ID field has been evaluated and discussed. This will be implemented further in the API/UI for user privileges
- New test data
  - "Clean data" will be provided to the team, and the team will strip down current data. This will be used to evaluate indexing and optimization of MongoDB

II. Next Meeting

- Updates based on data above

Week 7

Week 7: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week has been Reading Break, and a week without a scheduled sprint. This week involved our team researching C# and Azure hosting to implement for the coming API sprints. Our team was able to strip down the "messy data" into a skeleton to incorporate sample data for index testing.

Task Completion from Last Week

- Reading Break – no scheduled sprint
- Team has become familiar with C# for upcoming API sprints
- Team has requested to obtain a school-license of Azure to test a hosted database as per client's requirements. This should be updated in the coming week
- Team has removed redundant data from current JSON BLOB's to work with clean data for index testing

Task for This Week
Pushed from Last Week:

- Incorporated new test data into database – postponed due to changes in data format for storage
- Indexing capabilities – discussed with the client. Updates will be done to this section based on changes to data storage
- Team with gather different data from client

Note: Tasks in this section will be completed over two weeks

- Team will create API to submit data and update data elements.
- Team will create API for basic searching functionality

Additional Information

- Team has a couple of tasks pushed from previous sprint (indexing and data storage), but feel they are still on time for proposed completion

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure
    - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage, and have started learning C# and Azure
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
- Indexing restrictions based on all choices for full-text search
    - Update: team has stripped current JSON BLOB data for a clean skeleton. Tea should also be receiving new data from client to test this

- Limitations for speed and storage in regards to indexing and searching
    - Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will continue to be re-evaluated

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
    - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
    - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
    - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
    - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
    - Client is aware of these issues and is ready to incorporate it into the final result

Week 7 Meeting Agenda

3/28/2014
10:30 AM

Client: Andrew Challenger, Auzeb Manzoor

Team: Bobby Esfandiari

Topics:

Introduction

Current status of project in regards to milestones document

Indexing

Discuss text search and possible solutions through MongoDB

Windows Azure

Discuss hosting and free educator pass

Test Data

Team currently extracting usable test data from “messy” data

Discuss cleaner version

Roundtable

Any other topics not listed on this agenda

Week 8

Week 8: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week has been busy with a new sprint, and requirements passed from last sprint. We have begun creating the API for MongoDB data manipulation and querying, as well as continued to work on indexing.

Task Completion from Last Week
Note: Tasks in this section will be completed over two weeks

- Incorporated new test data into database – created a program to generate important data in documents to be added to the database as test data
- Indexing capabilities – discussed with the client again. Client would like full-text index search to be the priority plan, and mixed-key indexing as a last resort. More research needed on capabilities
- Azure Hosting Request – awaiting approval
- API coding has begun. Methods have been created to submit data and begin to query data elements

Task for This Week
Pushed from Last Week:

- Continue building API methods for submitting data, updating data elements, and basic querying
- Use newly-created test data and text indexing capabilities with both full-text index and multi-key index
- Azure Hosting Request – await approval
- Try to schedule a conference call with MongoDB engineers

Additional Information

- Team has a couple of tasks pushed from previous sprint (indexing and data storage), but feel they are still on time for proposed completion

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure
    - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage, and have started learning C# and Azure
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
- Indexing restrictions based on all choices for full-text search
    - Update: team has stripped current JSON BLOB data for a clean skeleton. Tea should also be receiving new data from client to test this
- Limitations for speed and storage in regards to indexing and searching

- Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will continue to be re-evaluated

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
  - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
  - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
  - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result

Week 8 Meeting Agenda

3/28/2014
10:30 AM

Client: Andrew Challenger, Auzeb Manzoor, Chad Ludwig

Team: Bobby Esfandiari, Nicole Saat

Topics:

Introduction

Current status of project in regards to milestones document

Indexing

Discuss text search size and efficiency, as well as testing using the cleaned up data

---

API

Talk about the current implementation of the API

Windows Azure

Discuss the status of hosting and testing

Contacting MongoDB

Investigating alternative methods of importing and indexing data in MongoDB

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage

Meeting 8 Minutes

February 28, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari, Nicole Saat
Next meeting:	Friday, March 4th, 2014

I. Discussion

- Indexing
  - Advised of test results from sample data (13MB collection vs. 15MB indexing)
    - Client has advised this still may be OK for requirements
  - Advised we would still look into Mixed Key Indexing
    - Client feels that this should be a last-resort, as they wish to eliminate restrictions on knowing what is in system (i.e. knowing which keys will search faster/slower)
  - Client wishes all items be searched quickly
- MongoDB
  - Advised team is still waiting to hear about MongoDB conference call
    - Client has advised it is OK to advise what company it is
- API
  - Provided update advising what has been completed on API
  - Advised client we are mid-sprint
- Azure
  - Advised client status of Azure license, and 3Gig size
- Sample Data
  - Advised of sample cases Stefan has created.
    - Client is happy that data is “true data”

II. Next Meeting

- Updates based on data above

Week 9

Week 9: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week has been completing a new sprint. We have begun creating the API for MongoDB data manipulation and querying, as well as continued to work on indexing.

Task Completion from Last Week

- Incorporated new test data into database – created a program to generate important data in documents to be added to the database as test data
    - Two types of data were created – one stripped of irrelevant data, and one unstripped. Programs were created to use true data to test real capabilities of the system
- MongoDB meeting – team was able to have a phone conversation with MongoDB regarding indexing. They recommended full-text search or indexing only pertinent data.
    - This was discussed with the client: they do not want to go the mixed-key route. Full-text must be tested for size and timing
- Indexing capabilities – Full-text search must be used. This will be incorporated into the API this coming sprint.
- Azure Hosting Request – awaiting approval
- API coding has begun. Methods have been created to submit data, query with passed fields (instead of free-text), and update data.

Task for This Week
Note: This sprint will be completed over two weeks

- API for complete search functionality (including full-text search)
- API querying must be complete at the end of this stage.
- The complete functioning prototype
- Confirm size for indexing with true data

Additional Information

- Team feels they are still on time for completion. Next sprint will include wrap-up for documentation, and any tasks pushed from this week

Known issues / things blocking progress

- API restrictions for importing data
    - C# driver has restrictions for importing data that has the same field name. This can be imported directly through Mongo. This will continue to be tested to try and find a way to import through C# API
- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure

    - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage, and have started learning C# and Azure
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
    - This should be addressed this sprint
- Indexing restrictions based on all choices for full-text search
    - Test data has been created both with stripped data and current form data
- Limitations for speed and storage in regards to indexing and searching
    - Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will continue to be re-evaluated

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
    - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
    - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
    - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
    - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
    - Client is aware of these issues and is ready to incorporate it into the final result

Week 9 Meeting Agenda

3/28/2014
10:30 AM

Client: Andrew Challenger, Auzeb Manzoor

Team: Bobby Esfandiari

Topics:

Introduction

Current status of project in regards to milestones document

Indexing

Discuss text search/index with regards to discussion with MongoDB support

---

Windows Azure

Still waiting on Azure approval for Educator pass

No promotion code from MongoDB

Test Data

Team is working on importing full data without stripping out any information

---

API

Discuss current progress on API implementation

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage

Meeting 9 Minutes

March 7, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari
Next meeting:	Friday, March 14th, 2014

I. Discussion

- Text Search/Indexing
  - Let the client know that MongoDB support has advised the following options:
    - Maintain an index on the most commonly used fields
    - If resources allow, run the full text index
    - Use Solr as a search platform
  - Client advised that they are strongly against an index that has to be maintained by a developer
    - Developer time is the most important resource for them
    - Need it to be fully dynamic
    - They looked at Solr before and it did not make sense for them to use both it and MongoDB
    - They believe Text Search is still our best option
- API
  - Provided update advising what has been completed on API
- Azure
  - Advised client status of Azure license,
    - They would like for us to start testing with Azure as soon as possible
    - If we do not hear back on the license status by next week, then we will have to look at other options (possibly through VS)
    - Advised client that MongoDB did not have access to a Promotion code for us
- Sample Data
  - Advised of new test data that does not need to strip out the metadata

II. Next Meeting

- Updates based on data above

Week 10

Week 10: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week includes the beginning of a new sprint. This sprint will involve having a working prototype that includes aggregate querying

Task Completion from Last Week
Note: This sprint will be completed over two weeks

- Unstripped data has been converted to create test data
  - This has been discussed with client, and client will be using only stripped data. This will be re-implemented to test size
- Azure has not contacted back regarding student license. Client provided credits for team to work with Azure
- API methods have been created for basic search functionality. Team has been researching implementing the full-text search in C#

Task for This Week
Note: This sprint will be completed over two weeks

- API for complete search functionality (including full-text search)
- API querying must be complete at the end of this stage.
- The complete functioning prototype
- Confirm size for indexing with true data

Additional Information

- Team feels they are still on time for completion. Next sprint will include wrap-up for documentation, and any tasks pushed from this week

Known issues / things blocking progress

- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure
  - Partially Resolved. Team has become familiar with functionality of MongoDB and JSON Data Storage, and have started learning C# and Azure
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
  - This should be addressed this sprint
- Indexing restrictions based on all choices for full-text search
  - Test data has been created both with stripped data and current form data
- Limitations for speed and storage in regards to indexing and searching
  - Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will continue to be re-evaluated

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
  - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
  - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
  - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result
- API restrictions for importing data - C# driver has restrictions for importing data that has the same field name. This can be imported directly through Mongo. This will continue to be tested to try and find a way to import through C# API
  - Resolved. Was able to convert data to a JObject before Bson Document and import will work.

Week 10 Meeting Agenda

3/28/2014
10:30 AM

Client: Andrew Challenger, Auzeb Manzoor, Chad Ludwig

Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat

Topics:

Introduction

Current status of project in regards to milestones document

Windows Azure

Still no word on Educator Pass

Created trial account using Visual Statement credentials

Test Data

Team created full test data

Continuing to test this data in various sizes to gather metrics

---

API

Discuss progress on API and the implementation of various search queries

---

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage
Meeting 10 Minutes

March 14, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor, Chad Ludwig
	Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat
Next meeting:	Friday, March 21st, 2014

I. Discussion

- Azure
  - Advised of current status of Azure
  - Client has provided team with a $220 credit user to test system
  - Client has been talking with Azure about the MongoDB promo code, but has received a response that it is "not ready for preview"
  - Until further information is obtained, will use Stefan's computer with 12G ram and large hardware space to simulate 5 million records
- API
  - Advised of current status of API (basic methods)
  - Client has requested to see current code. Will send by today evening
  - Client has asked whether API is separate level than UI or whether it is more incorporated. Discussed
- Current Data
  - Advised we have both stripped data and complete data in system to test.
  - Client has confirmed we will only be working with stripped data as a search bank, where the complete data will be retrieved from another location.
  - Will create more stripped data for testing
- Searching
  - Discussed Aggregate Querying as next step for coding API.
  - Client would like to see fields indicating where search word was found when querying
    - Example: Search "Blue" may return eye color, car color, last name, etc.
    - Query should return _id, parameter of hit, report number, and date
    - Search should be distinct (not return one document's _id multiple times if multiple hits)
    - Must return multiple reports in a document if there are multiple reports
  - Client suggested to start with returning a query and its "hit parameters" before distinct and multiple reports.

II. Next Meeting

- Updates based on data above

Week 11

Week 11: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week involved the end of our last scheduled coding sprint. We have completed most of the scheduled items, however a couple of items have been pushed to our "cushion" sprint as listed below. We have discussed the status of our project with the client, fine-tuned the requirements for completion, and set the final presentation date.

Task Completion from Last Week

- API for complete search functionality (including full-text search)
  - Full-Text search functionality has been implemented in the API.
  - The client has requested the returned data indicates which field the searched data was found in – this has been pushed to next sprint.
- API querying must be complete at the end of this stage.
  - Querying is completed minus the field data listed above.
- The complete functioning prototype
  - The team has a prototype that is returning the _id field of found documents.
  - As listed above, the only code to still be implemented is the return path
- Confirm size for indexing with true data
  - The index has been created for a 5-million record file. This has been documented and will be discussed with the client next week
- Provided client with current code and test data
  - Client has advised the current code is appropriate and does not foresee any issues.

Task for This Week
Note: tasks in this section will be completed over two weeks

- Sprint dedicated to graphical interface and to account for any delays in milestones previously listed
  - Presentation-appropriate UI is currently being developed
- Field of data searched return to user must be developed in API
- Test of speed between indexed 5-million record collection and non-indexed 5-million record collection to advise client

Additional Information

- Team feels they are still on time for completion. Although a couple of tasks have been pushed to this week, they are tasks the client agreed are non-crucial to the final product if not feasible. If issues arise, it will be documented for the client and discussed.

Known issues / things blocking progress

- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
  - This should be addressed this sprint with the field data to be returned.
- Limitations for speed and storage in regards to indexing and searching
  - Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will continue to be re-evaluated

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
  - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
  - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
  - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result
- API restrictions for importing data - C# driver has restrictions for importing data that has the same field name. This can be imported directly through Mongo. This will continue to be tested to try and find a way to import through C# API
  - Resolved. Was able to convert data to a JObject before Bson Document and import will work.
- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure
  - Resolved. Team has. Team has become familiar with functionality of MongoDB and JSON Data Storage, as well as C#. Due to time restrictions and communication with Azure, we will not be implementing in the final product.
- Indexing restrictions based on all choices for full-text search
  - The size has been addressed with the client and will be documented.

Week 11 Meeting Agenda

3/28/2014
12:00 PM

Client: Andrew Challenger, Auzeb Manzoor, Chad Ludwig

Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat

Topics:

Introduction

Current status of project in regards to milestones document

Windows Azure

No promotional code for MongoDB

Azure testing on hold due to time constraints

Test Data

Team has imported full data on test machine without stripping out any information

Indexing is currently underway

API

Full text search implemented

Discuss returned results within documents

Deliverables

Discuss expected project output in more detail

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage

Meeting 11 Minutes

March 21, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor
	Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat
Next meeting:	Friday, March 28th, 2014

I. Discussion

- Azure
  - Advised client that testing using Azure is currently on hold due to time and resource limitations
  - Team will try and incorporate some testing if time allows
  - The client is okay with this
- API
  - Advised of current status of API (basic methods & text search)
  - Client would like to see the code one week prior to presentation
- Current Data
  - Advised client of importing complete data in larger quantity for testing
    - Data is currently undergoing indexing
  - Client will create more stripped data for testing
- Searching
  - Client is okay with receiving a string of the element path pointing to the field involving the queried item, for returned results
    - They can handle further processing on their end
  - Next step is to implement searches using multiple fields and aggregate querying
- Deliverables
  - Client is happy with what we have so far
  - At the end of the project, they would like to receive the API and Documentation (with examples)
  - Discussed dates for final presentation
    - April 10th if feasible for all parties

II. Next Meeting

- Updates based on data above

Week 12

Week 12: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week involved the beginning of our final sprint. In this sprint the team is required to complete the API and all functionality, as well as begin to complete the report documentation. The team has laid out the requirements with the client.

Task Completion from Last Week
Note: tasks in this section will be completed over two weeks

- Presentation-appropriate UI is currently being developed
  - This is still being developed in the coming week
- Field of data searched return to user must be developed in API
  - Path of data is being returned by API
- Test of speed between indexed 5-million record collection and non-indexed 5-million record collection to advise client
  - Still being evaluated
- Met with client to discuss final requirements. See Meeting 12 Minutes for information

Task for This Week
Note: tasks in this section will be completed over two weeks

- Sprint dedicated to graphical interface and to account for any delays in milestones previously listed
  - Presentation-appropriate UI is currently being developed
- Report number of document must be returned in API
- Test of speed between indexed 5-million record collection and non-indexed 5-million record collection to advise client

Additional Information

- Team feels they are still on time for completion. This week is the final sprint, however there is no foreseeable issues with the API and/or documentation

Known issues / things blocking progress

- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
  - This should be addressed this sprint with the field data to be returned.
- Limitations for speed and storage in regards to indexing and searching
  - Update: Client has requested team re-evaluate these limitations based on newly structured BLOB data. Will continue to be re-evaluated

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
  - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
  - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
  - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines
  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result
- API restrictions for importing data - C# driver has restrictions for importing data that has the same field name. This can be imported directly through Mongo. This will continue to be tested to try and find a way to import through C# API
  - Resolved. Was able to convert data to a JObject before Bson Document and import will work.
- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure
  - Resolved. Team has. Team has become familiar with functionality of MongoDB and JSON Data Storage, as well as C#. Due to time restrictions and communication with Azure, we will not be implementing in the final product.
- Indexing restrictions based on all choices for full-text search
  - The size has been addressed with the client and will be documented.

Week 12 Meeting Agenda

3/28/2014
10:00 AM

Client: Andrew Challenger, Auzeb Manzoor

Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat

Topics:

Introduction

Current status of project in regards to milestones document

Documentation

Discuss the expected documentation design for deliverables at the end of the project

Test Data

Discuss the structure of data being used for testing as well as the required metrics for the culmination of the project

API

Discuss current progress on current API implementation, specifically the querying process with relation to the test data's format

Roundtable

Any other topics not listed on this agenda

NoSQL Data Storage

Meeting 12 Minutes

March 28, 2014

Present:	Client: Andrew Challenger, Auzeb Manzoor, Chad Ludwig
	Team: Bobby Esfandiari, Stefan Schielke, Nicole Saat
Next meeting:	Friday, April 4th, 2014

I. Discussion

- Test Data
  - Discussed the current test data format
  - Client has approved test data as is right now
  - Will remove related report numbers throughout data for presentation
- Documentation
  - Client requests documentation include “lessons learned” through project.
  - They advise to focus less on the analysis, and more on why items were chosen/not chosen, and data about roadblocks encountered
  - Client request we explain the items that are “hacked together”, as well as items that were easy to implement (or not implemented but should have been) and will not cause problems in the future
- API
  - Discussed the API and how data is being returned. Discussed how it is being returned with a path to the items.
  - Context ID must be incorporated to be returned
  - Report numbers must be incorporated to be returned

II. Next Meeting

- Updates based on data above

Week 13

Week 13: Weekly Report for COMP 4910 Project Course
NoSQL Data Storage

Summary:
This past week involved the end of our final sprint. In this sprint the team completed the base methods for the API and all functionality. The team began to complete documentation (as well as testing), and will be completing the UI next week. In the coming week, the team will be presenting the final product to the client and Kevin, as well as completing documentation.

Task Completion from Last Week
Note: tasks in this section will be completed over two weeks

- Sprint dedicated to graphical interface and to account for any delays in milestones previously listed
    - Presentation-appropriate UI is currently being developed
- Report number of document must be returned in API
    - This has been completed.
- Test of speed between indexed 5-million record collection and non-indexed 5-million record collection to advise client
    - This is being completed for April 7. Results will be noted in Testing documentation

Task for This Week

- Completion of documentation
- Completion of testing
- Completion of UI for API
- Presentation to client April 10

Additional Information

- Team is on time for completion

Known issues / things blocking progress

Resolved Known Issues

- Spacing of existing data to be imported/exported into and from Mongo
    - Resolved. This has been resolved via the --jsonArray command
- Understanding exact scope/minimal requirements of project
    - Resolved. Has been documented in Requirements Document
- Versioning is not specifically provided as a part of MongoDB's base functionality
    - Resolved. Team will be using "Vermongo" paradigm as reviewed with client
- Data Security in regards to unsecured machines

  - Client is providing just sample data – no restrictions for use on unsecured team machines
- Data provided by client will only be sample data. This will not represent true data from the system
  - Client is aware of these issues and is ready to incorporate it into the final result
- API restrictions for importing data - C# driver has restrictions for importing data that has the same field name. This can be imported directly through Mongo. This will continue to be tested to try and find a way to import through C# API
  - Resolved. Was able to convert data to a JObject before Bson Document and import will work.
- New concepts for team members: JSON Data Storage, MongoDB, C#, Azure
  - Resolved. Team has. Team has become familiar with functionality of MongoDB and JSON Data Storage, as well as C#. Due to time restrictions and communication with Azure, we will not be implementing in the final product.
- Indexing restrictions based on all choices for full-text search
  - The size has been addressed with the client and will be documented.
- Difficult issues with querying (i.e. Accident that involves two people with two different charges)
  - Resolved. Query results have been built to return report number in which data was found, as well as related reports
- Limitations for speed and storage in regards to indexing and searching
  - Resolved. We have discussed with client and have advised in documentation how long queries take based on full text search.