CS 572 Modern Web Applications

Najeeb Najeeb, PhD (najeeb@miu.edu)

V1.0.0

JavaScript Full Stack Development

- MongoDB
  - NoSQL database (document store)
  - Stores JSON documents
- Express
  - JavaScript web framework
  - On top of Node
- Angular
  - JavaScript UI framework
  - Single Page Applications
- Node
  - JavaScript server-side platform
  - Single threaded, fast and scalable

Roadmap and Outcomes

- Node.js: write asynchronous (non-blocking) code. Understand node platform to start a project.
- Express: setup express and get requests and send back responses. REST API.
- MongoDB: what NoSQL DB looks like. Full API interacting with DB.
- AngularJS: Investigate AngularJS and architect it. A single page application.
- MEAN application: Learn by example. We will create a MEAN Games application.

NoSQL DB

NoSQL Database Types

- Key-value store, ArangoDB
  - Store unique key and value, high scalability for caching (session management)
- Document store, MongoDB
  - Store semi-structured data in document format, no schema insert(mobile applications)
- Wide- column store, Amazon DynamoDB
  - Store in columns not rows, fast (catalogs, recommendation engines)
- Graph databases, Amazon Neptune
  - Store data as nodes and edges, show connections (reservation systems)
- More

Document Store vs Relational DB

RELATIONAL DB

STUNDET_ID	NAME	GPA
1	Jack	3.0
2	Jill	3.3
3	John	2.8

ID	COURSE_NAME	STUDENT_ID
1	Software Engineering	1
2	Web Programming	2
3	Algorithms	2

DOCUMENT STORE


```
[
 { "StudentID" : 1,
   "Name" : "Jack",
   "GPA" : 3.0,
   "Courses" : [
    { "ID" : 1,
      "CourseName" : "Software Engineering" } ] },
 { "StudentID" : 2,
   "Name" : "Jill",
   "GPA" : 3.3 },
   "Courses" : [
    { "ID" : 2,
      "CourseName" : "Web Programming" },
    { "ID" : 3,
      "CourseName" : "Algorithms" } ] },
 { "StudentID" : 3,
   "Name" : "John",
   "GPA" : 2.8 }
]
```

NoSQL DB Design

- What is all the data you wish to output (at once) on a pages.
  - Put that information in one place.
- If on some page you wish to display some of the information from another document.
  - Add what needs to be displayed and include an ID to link to the other document.
- Optimize for the most common operation.
  - Reduce updates for the most common changeable items.
  - Increase speed of displaying most common pages.
- Keep number of Collections (Tables) to a minimum.
- Try to reduce each page to one collection (or minimum number of joined collections)
- Most common operations must run faster (even at the expense of less common operations)

MongoDB

MongoDB Collections

REVIEW.JSON


```
[
{ "ReviewID" : 1,
  "Title" : "Good Game.",
  "Review" : "I enjoyed the game.",
  "Stars" : 4,
  "Game" : {
    "ID" : 1,
    "Name" : "Trains"}
},
{ "ReviewID" : 2,
  "Title" : "Too Long.",
  "Review" : "The game is nice, but it was too long.",
  "Stars" : 3,
  "Games" : {
    "ID" : 2,
    "Name" : "Monopoly"}
}]
```


GAME.JSON


```
[{ "ID" : 1,
   "Name" : "Trains",
   "Price" : 48.82,
   "MinPlayers": 2,
   "MaxPlayers": 4,
   "EstimatedTimeToPlay": 45,
   "ReleaseYear": 2013},
{  "ID" : 2,
   "Name" : "Monopoly",
   "Price" : 29.97,
   "MinPlayers": 2,
   "MaxPlayers": 8,
   "EstimatedTimeToPlay": 180,
   "ReleaseYear": 1933},
{  "ID" : 3,
   "Name" : "Risk",
   "Price" : 20.99,
   "MinPlayers": 2,
   "MaxPlayers": 6,
   "EstimatedTimeToPlay": 120,
   "ReleaseYear": 1959}
]
```

How to Design a Document

- Why not have one Collection and store everything in it?
  - Not good logically and performance.
  - Hard to maintain.
- A review is for a game, so why not only have one Collection of Games.
  - A review can exists by itself.
  - Get all positive reviews, negative, …
  - A Game could also have several reviews.
- Collections may reference each other.
- You do not use a collection to get data from another collection.
  - What you want from another collection embed in your collection.

JSON and BSON

- JSON is what you use in your application.
- JSON is a close representation of what MongoDB stores.
- BSON is Binary-JSON, it is what MongoDB uses.
- BSON not human readable but maintains the flexibility and ease of use of JSON plus the speed of binary format.
- MongoDB accepts JSON and returns JSON (but stores it as BSON).

JSON ID

- MongoDB creates unique ID for a document when created.
- _id property is what MongoDB creates.
- The value is ObjectId("5f9aef68980db44d37c1aaed") unique combination of time (Unix epoch) , machine ID, process ID, and counter.

Install and Work With MongoDB

- Install from MongoDB website (www.mongodb.com/try/download)
- Running MongoDB
  - mongo --version
  - mongo
  - exit (or Ctrl + C)
- Create Database
- Create Collection
- Retrieve Collection

MongoDB
Database
Collection

List all databases on your system

```
show dbs
admin 0.000GB
config 0.000GB
local   0.000GB
```

Select database to work with

```
use local
switched to db local
```

Create new database, make sure it does not exist

```
use newTestDb
switch to newTestDB
```

Note: new database not created until you add a collection to it.

Get the current database being used

```
db (or db.getName();)
newTestDB
```

Delete database

```
db.dropDatabase();
{ "dropped" : "newTestDb", "ok" : 1}
```

MongoDB
Database
Collection

List collections in current database

```
use local
show collections
startup_log
use newTestDB
show collections
```

Create collection

```
db.createCollection("technology")
{ "ok" : 1}
```

Delete collection

```
db.technology.drop()
true
```

CRUS

Crteate

Read

Update

Delete

Add document in current collection

```
db.technology.insert(
... {
... name : "MongoDB",
... role : "Database"
... }
... );
WriteResult({ "nInserted" : 1 })
```

List documents in current collection

```
db.technology.find();
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB",
"role" : "Database" }
db.technology.find().pretty();
```

Insert multiple documents at once

```
db.technology.insert([{name : "Express", role: "Web application server"},
... {name : "Angular", role: "Front-end framework"},
... {name : "Node.js", role: "Platform"}]);
BulkWriteResult({ ... "nInserted" : 3 ... })
```

CRUS

Crteate

Read

Update

Delete

List all documents in current collection

```
db.technology.find();
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB",
"role" : "Database" }
db.technology.find().pretty();
```

List based on document id in current collection

```
db.technology.find({"_id" : ObjectId("5f9aef68980db44d37c1aaed")});
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB",
"role" : "Database" }
```

List based on name in current collection

```
db.technology.find({"name" : Angular")});
{ "_id" : ObjectId("5f9af651980db44d37c1aaef"), "name" : "Angular",
"role" : "Front-end framework" }
```

Sorting, 1 for assending -1 for decending

```
db.technology.find().sort({"name" : 1});
```

Limit returned fields, projection (the second parameter in find).

```
db.technology.find({}, {"name" : true});
db.technology.find({}, {"name" : true, "_id" : false});
```

CRUS

Crteate

Read

Update

Delete

Update a document , finds the documents of interest the updates them. The first parameter is the query, the second is the data to set.

```
db.technology.update( {"name" : "Angular"}, {$set : {"name" : AngularJS"} } );
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Update more than one document at once

```
db.technology.update({},{$set:{"language":JavaScript"}},{multi:true}  );
{ "_id" : ObjectId("5f9aef68980db44d37c1aaed"), "name" : "MongoDB", "role" : "Database", "language" : "JavaScript" }
{ "_id" : ObjectId("5f9af651980db44d37c1aaee"), "name" : "Express", "role" : "Web application server", "language" : "JavaScript" }
{ "_id" : ObjectId("5f9af651980db44d37c1aaef"), "name" : "Angular", "role" : "Front-end framework", "language" : "JavaScript" }
{ "_id" : ObjectId("5f9af651980db44d37c1aaf0"), "name" : "Node.js", "role" : "Platform", "language" : "JavaScript" }
```

CRUS

Crteate

Read

Update

Delete

Delete document from collection, you provide a query object

```
db.technology.remove( { "name" : "Express" })
WriteResult({ "nRemoved" : 1 })
```

```
db.technology.remove( {})
```

This will remove all the documents from the collection :(

Import & Export Data

BSON

Import

Export

Import MongoDB data from BSON file

```
mongorestore --db newTestDB2 --gzip dump/newTestDb
2020-11-01T13:46:25.982-0800    building a list of
collections to restore from dump/newTestDb dir
2020-11-01T13:46:25.987-0800    reading metadata for
newTestDb2.technology from
dump/newTestDb/technology.metadata.json.gz
2020-11-01T13:46:26.058-0800    restoring
newTestDb2.technology from
dump/newTestDb/technology.bson.gz
2020-11-01T13:46:26.076-0800    no indexes to restore
2020-11-01T13:46:26.076-0800    finished restoring
newTestDb2.technology (4 documents)
2020-11-01T13:46:26.076-0800    done
```

BSON

Import

Export

Export MongoDB data as BSON file

```
mongodump --db newTestDB
writting newTestDb.technology to
done dumping newTestDb.technology (4 documents)
/dump/newTestDb/technology.bson
```

Compress the BSON output data

```
mongodump --db newTestDB --gzip
/dump/newTestDb/technology.bson.gz
```

JSON

Export

Import

Export MongoDB data as JSON file (for a collection only)

```
mongoexport --db newTestDB --collection technology
{"_id":{"$oid":"5f9aef68980db44d37c1aaed"},"name":"MongoDB","role":"Databas
e","language":"JavaScript"}

{"_id":{"$oid":"5f9af651980db44d37c1aaee"},"name":"Express","role":"Web
    application server","language":"JavaScript"}

{"_id":{"$oid":"5f9af651980db44d37c1aaef"},"name":"Angular","role":"Front-end
    framework","language":"JavaScript"}

{"_id":{"$oid":"5f9af651980db44d37c1aaf0"},"name":"Node.js","role":"Platform","l
    anguage":"JavaScript"}

2020-11-01T13:54:18.608-0800    exported 4 records
```

Export to file

```
mongoexport --db newTestDB –collection technology --out output/technology.json
exported 4 records
```

Export as an array

```
mongoexport --db newTestDB –collection technology --out output/technology.json
--jsonArray --pretty
exported 4 records
```

JSON

Export

Import

Import MongoDB data from JSON file

```
mongoimport --db newTestDB3 --collection technology --
jsonArray output/technology.json
```

imported 4 documents

Connecting MongoDB to NodeJS

MongoDB & NodeJS

- Installing mongoDB driver in our app.
- Createing reusable connections.
- Defining connection string.
- Accessing connections from controllers.
- Best practices while doing all this.

Connect to DB

Install driver

Connections

Use DB

Install MongoDB native driver

```
npm install mongodb --save
```

```
mongodb@2.1.7 node_modules/mongodb
```

Connect to DB

Install driver

Connections

Use DB

Create file to manage connections,
File api/data/dbconnection.js

```
var MongoClient= requires("mongodb").MongoClient;
var dbName= "meanGamesDb";
var dburl= "mongodb://localhost:27017/"+dbName;
var_connection= null;
var open= function(){
    MongoClient.connect(dburl,{useUnifiedTopology: true}, function(err, client){
        if(err){
            console.log("DB connection failed");
            return;
        }
        _connection= vlirny.db(dbName);
        console.log("DB connection open", _connection);
    });
};
var get= function(){
    return _connection;
};
module.exports= {
    opend : open,
    get : get
};
```

Connect to DB

Install driver

Connections

Use DB

Open the connection as soon as the application starts, app.js

```
require("./api/data/dbconnection.js).open();
```

Run

```
npm start
```

DB connection open ....

Check for error, change the port number in dbconnection.js and run again.

Connect to DB

Install driver

Connections

Use DB

Use the db connection in the controllers.
api/controllers/games.controllers.js

```
var dbConnection= require("../data/dbconnection.js");

… gameGetAll= ..
var db= dbConnection.get();
console.log("db", db);
```

Run on browser (http://localhost:3000/api/games)

```
npm start
db …
```

Opening db is asynchronous. So make sure you get it when you need it. Don't just open it at the start of the file.

Opening db connection is slow. Best to open it once at application start and reuse it.

No need for a global variable for db. Encapsulated in dbconnection.

Working with MongoDB in NodeJS

Query DB

GetAll

Pagination

GetOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
… gameGetAll= ..
var collection= db.collection("games");
// var docs= collection.find(); //Sync not good :(
collection.find().toArray(function(err, docs) {
    console.log("Found games", docs);
    res.status(200).json(docs);
}
```

Run on browser (http://localhost:3000/api/games)

```
npm start
```

Found games …

Query DB

GetAll

Pagination

GetOne

Use the db connection in the controllers.
api/controllers/games.controllers.js

```
… gameGetAll= ..
var collection= db.collection("games");
var offset= 0;
var count= 5;
if (req.query && req.query.offset) {
    offset= parseInt(req.query.offset, 10);
}
if (req.query && req.query.count) {
    count= parseInt(req.query.count, 10);
}
collection.find().skip(offset).limit(count).toArray(function(err, docs) {
    console.log("Found games", docs);
    res.status(200).json(docs);
}
```

Run on browser (http://localhost:3000/api/games)

```
npm start
Found games …
```

Query DB

GetAll

Pagination

GetOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
var ObjectId= require("mongodb").ObjectId;
… gameGetOne= ..
var db= dbConnection.get();
var collection= db.collection("games");
var gameId= req.params.gameId;
collection.findOne({_id : ObjectId(gameId)}, function(err,
doc) {
    console.log("Found game", doc);
    res.status(200).json(doc);
}
```

Run on browser (http://localhost:3000/api/games)

```
npm start
```

Found games …

Insert DB

Error Checking

InsertOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
var ObjectId= require("mongodb").ObjectId;
… gameAddOne= ..
var db= dbConnection.get();
var collection= db.collection("games");
if (req.body && req.body.name && req.body.starts) {
    console.log(req.body);
    res.status(200).json(req.body);
} else {
    console.log("Data missing from POST body");
    res.status(400).json({error : "Required data missing from
POST"});
}
```

Run app.boomerangapi.com/workspace on browser

```
npm start
```

error: "Required data missing from POST" …

Insert DB
Error Checking
InsertOne

Use the db connection in the controllers.

api/controllers/games.controllers.js

```
... gameAddOne= ..
var newGame;
if (req.body && req.body.name && req.body.starts) {
   newGame= req.body;
   newGame.price= parseFloat(req.body.price);
   collection.insertOne(newGame, function(err, response) {
      console.log(response.ops);
      res.status(201).json(response.ops);
   }
} ...
```

Run app.boomerangapi.com/workspace on browser

npm start

Found games ...

MongoDB & NodeJS

- We will not be using mongoDB directly from nodeJS.
- There is a much easier way to work with mongoDB from Node.
- We will use Mongoose.

Main Points

- MongoDB is a document-based NoSQL database. It is ideal for mobile application development.
- We can use mongoDB driver to connect to a MongoDB instance from our node code. You will need a connection, make sure you create only once and use it several times. Also make sure it is available when needed (since it is asynchronous).
- The best practice when working with mongoDB from your node code is to create a connection then have all your DB related code in controllers.