Sean Gallagher

Data Access Layer

CS-340: Client/Server Development

**Milestone One: Data Access Layer**

The specification for this first milestone called for an implementation of the data access layer for a RESTful application, implementing the four basic CRUD operations associated with database access. Implementation is in Javascript, to better leverage the familiarity of front-end and full-stack developers with the language, and is written to take advantage of the MongoDB Native Javascript driver's asynchronous nature. For simplicity—and because this is a relatively constrained use case—the entire data access layer is implemented in a single Javascript source file, `dataAccess.js`. This file is included with the submission, and is listed in its entirety in this document. In the interest of keeping this submission as concise as possible, only `dataAccess.js` is included in the submission; unit tests, as well as the `mile1.js` test script that produced the output in Figures 1–6, are available from the project's Github repository.

```
#!/usr/bin/env node

const config = require('../config.json')
const MongoClient = require('mongodb').MongoClient

/**
 * Count the number of documents matching the query
 *
 * @param {Object} query - a MongoDB query document
 * @param {Object} [opts] - optional settings
 * @param {string} [opts.dbName] - the name of the target database
 * @param {string} [opts.colName] - the name of the target collection
 *
 * @returns {number} - the number of matching docs
 */
async function countMatching (query, opts = config) {
  const {
    dbName = config.dbName,
    colName = config.colName
  } = opts

  const client = initClient()
  await client.connect()
  const col = client.db(dbName).collection(colName)

  const numResults = await col.countDocuments(query)
  client.close()
```

```
  return numResults
}

/**
 * Create one or more new records with the specified document(s)
 *
 * @param {(Object|Object[])} document - the document to insert into the db
 * @param {Object} [opts] - optional settings
 * @param {string} [opts.dbName] - the name of the target database
 * @param {string} [opts.colName] - the name of the target collection
 *
 * @returns {Object} - the insert op result object returned from MongoDB
 */
async function dataCreate (document, opts = config) {
  const {
    dbName = config.dbName,
    colName = config.colName
  } = opts

  const client = initClient()
  await client.connect()
  const col = client.db(dbName).collection(colName)
  const result = Array.isArray(document)
    ? await col.insertMany(document)
    : await col.insertOne(document)
  await client.close()
  const resultOk = result.result.ok === 1

  return resultOk
}

/**
 * Delete records matching a specified query
 *
 * @param {Object} query - a MongoDB query document
 * @param {Object} [opts] - optional settings
 * @param {number} [opts.limit] - maximum number of matching results to delete
 * @param {string} [opts.dbName] - the name of the target database
 * @param {string} [opts.colName] - the name of the target collection
 *
 * @returns {Object} - the delete op result object returned from MongoDB
 */
async function dataDelete (query, opts = config) {
  const {
    dbName = config.dbName,
    colName = config.colName
  } = opts

  const client = initClient()
  await client.connect()
  const col = client.db(dbName).collection(colName)
  const result = (dataRead(query).length > 1)
    ? await col.deleteOne(query)
    : await col.deleteMany(query)
```

```
  await client.close()
  return result
}

/**
 * Perform a read operation and return the results as an array
 *
 * @param {Object} query - a MongoDB query document
 * @param {Object} [opts] - optional settings
 * @param {number} [opts.limit] - the number of results to return
 * @param {string} [opts.dbName] - the name of the target database
 * @param {string} [opts.colName] - the name of the target collection
 *
 * @returns {Object[]} - an array of matching documents from the collection
 */
async function dataRead (query, opts = config) {
  const {
    limit = config.limit,
    dbName = config.dbName,
    colName = config.colName
  } = opts

  const client = initClient()
  await client.connect()
  const col = client.db(dbName).collection(colName)
  const result = await col.find(query).limit(limit).toArray()
  await client.close()
  return result
}

/**
 * Perform an update operation on documents found within the database
 *
 * @param {Object} query - a MongoDB query document
 * @param {Object} updates - a MongoDB update document
 * @param {Object} [opts] - optional settings
 * @param {string} [opts.dbName] - the name of the target database
 * @param {string} [opts.colName] - the name of the target collection
 */
async function dataUpdate (query, updates, opts = config) {
  const {
    dbName = config.dbName,
    colName = config.colName
  } = opts

  const client = initClient()
  await client.connect()
  const col = client.db(dbName).collection(colName)

  const numMatches = await countMatching(query, opts)

  const result = (numMatches > 1)
    ? await col.updateMany(query, updates)
    : await col.updateOne(query, updates)
```

```
136
137   await client.close()
138   return result
139 }
140
141 /**
142  * Initialize the MongoClient
143  *
144  * @param {Object} [conf] - a Config object with connection URI component info
145  * @returns {MongoClient} - a MongoClient object initialized with the config
146  */
147 function initClient (conf = config) {
148   const url = (
149     'mongodb://' +
150     `${conf.connectURI.user}:${conf.connectURI.pass}` + // Authentication creds
151     `@${conf.connectURI.host}:${conf.connectURI.port}` + // Host name and port
152     `/?authSource=${conf.connectURI.auth}` // Authentication db
153   )
154
155   const client = new MongoClient(url, { useUnifiedTopology: true })
156
157   return client
158 }
159
160 module.exports = {
161   countMatching: countMatching,
162   dataCreate: dataCreate,
163   dataDelete: dataDelete,
164   dataRead: dataRead,
165   dataUpdate: dataUpdate,
166   initClient: initClient
167 }
```

../../db/dataAccess.js

```
[sean@Jotunheim] ~/.../cs340/cs340-project >>> ./tests/01-db/mile1.js
true
[
  {
    _id: 5ee30d44ae266f127dc34829,
    id: '12345-6789-TEST',
    certificate_number: 123456789,
    business_name: "Test Testington's Testing Supply Warehouse and Emporium",
    date: 2020-06-12T05:06:12.502Z,
    result: 'Overstocked On All Testing Supplies',
    sector: 'Testing and Evaluation - 123',
    address: {
      number: 1234,
      street: 'Testsylvania Avenue',
      city: 'Testerton',
      zip: 12345
    }
  }
]
CommandResult {
  result: { n: 1, nModified: 1, ok: 1 },
  connection: Connection {
    _events: [Object: null prototype] {
      commandStarted: [Function (anonymous)],
      commandFailed: [Function (anonymous)],
      commandSucceeded: [Function (anonymous)],
      clusterTimeReceived: [Function (anonymous)]
    },
    _eventsCount: 4,
    _maxListeners: undefined,
    id: 1,
    address: '127.0.0.1:27017',
    bson: BSON {},
    socketTimeout: 360000,
    monitorCommands: false,
    closed: false,
    destroyed: false,
    lastIsMasterMS: 1,
    [Symbol(kCapture)]: false,
    [Symbol(description)]: StreamDescription {
      address: '127.0.0.1:27017',
      type: 'Standalone',
      minWireVersion: 0,
      maxWireVersion: 8,
      maxBsonObjectSize: 16777216,
      maxMessageSizeBytes: 48000000,
      maxWriteBatchSize: 100000,
      compressors: []
    },
    [Symbol(generation)]: 0,
    [Symbol(lastUseTime)]: 1591938372851,
    [Symbol(queue)]: Map(0) {},
    [Symbol(messageStream)]: MessageStream {
      _readableState: [ReadableState],
      _events: [Object: null prototype],
```

Figure 1: Output from test script showing successful execution of Create, Read, Update, and Delete operations using functions from `dataAccess.js` (1 of 6)

```
    _events: [Object: null prototype],
    _eventsCount: 7,
    _maxListeners: undefined,
    _writableState: [WritableState],
    allowHalfOpen: true,
    bson: BSON {},
    maxBsonMessageSize: 67108864,
    [Symbol(kCapture)]: false,
    [Symbol(buffer)]: [BufferList]
  },
  [Symbol(stream)]: Socket {
    connecting: false,
    _hadError: false,
    _parent: null,
    _host: 'localhost',
    _readableState: [ReadableState],
    _events: [Object: null prototype],
    _eventsCount: 7,
    _maxListeners: undefined,
    _writableState: [WritableState],
    allowHalfOpen: false,
    _sockname: null,
    _pendingData: null,
    _pendingEncoding: '',
    server: null,
    _server: null,
    timeout: 360000,
    _peername: [Object],
    [Symbol(async_id_symbol)]: 264,
    [Symbol(kHandle)]: [TCP],
    [Symbol(kSetNoDelay)]: true,
    [Symbol(lastWriteQueueSize)]: 0,
    [Symbol(timeout)]: Timeout {
      _idleTimeout: 360000,
      _idlePrev: [TimersList],
      _idleNext: [TimersList],
      _idleStart: 1060,
      _onTimeout: [Function: bound ],
      _timerArgs: undefined,
      _repeat: null,
      _destroyed: false,
      [Symbol(refed)]: false,
      [Symbol(asyncId)]: 282,
      [Symbol(triggerId)]: 264
    },
    [Symbol(kBuffer)]: null,
    [Symbol(kBufferCb)]: null,
    [Symbol(kBufferGen)]: null,
    [Symbol(kCapture)]: false,
    [Symbol(kBytesRead)]: 0,
    [Symbol(kBytesWritten)]: 0
  },
  [Symbol(ismaster)]: {
    ismaster: true,
    maxBsonObjectSize: 16777216,
```

Figure 2: Output from test script showing successful execution of Create, Read, Update, and Delete operations using functions from `dataAccess.js` (2 of 6)

```
      maxBsonObjectSize: 16777216,
      maxMessageSizeBytes: 48000000,
      maxWriteBatchSize: 100000,
      localTime: 2020-06-12T05:06:12.782Z,
      logicalSessionTimeoutMinutes: 30,
      connectionId: 2422,
      minWireVersion: 0,
      maxWireVersion: 8,
      readOnly: false,
      ok: 1
    }
  },
  message: BinMsg {
    parsed: true,
    raw: <Buffer 3c 00 00 00 4e 31 00 00 1f 00 00 00 dd 07 00 00 00 00 00 00 27
 00 00 00 10 6e 00 01 00 00 00 10 6e 4d 6f 64 69 66 69 65 64 00 01 00 00 00 01 6f
6b ... 10 more bytes>,
    data: <Buffer 00 00 00 00 00 27 00 00 00 10 6e 00 01 00 00 00 10 6e 4d 6f 64 6
9 66 69 65 64 00 01 00 00 00 01 6f 6b 00 00 00 00 00 00 00 f0 3f 00>,
    bson: BSON {},
    opts: { promoteLongs: true, promoteValues: true, promoteBuffers: false },
    length: 60,
    requestId: 12622,
    responseTo: 31,
    opCode: 2013,
    fromCompressed: undefined,
    responseFlags: 0,
    checksumPresent: false,
    moreToCome: false,
    exhaustAllowed: false,
    promoteLongs: true,
    promoteValues: true,
    promoteBuffers: false,
    documents: [ [Object] ],
    index: 44
  },
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
CommandResult {
  result: { n: 1, ok: 1 },
  connection: Connection {
    _events: [Object: null prototype] {
      commandStarted: [Function (anonymous)],
      commandFailed: [Function (anonymous)],
      commandSucceeded: [Function (anonymous)],
      clusterTimeReceived: [Function (anonymous)]
    },
    _eventsCount: 4,
    _maxListeners: undefined,
    id: 1,
    address: '127.0.0.1:27017',
    bson: BSON {},
```

Figure 3: Output from test script showing successful execution of Create, Read, Update, and Delete operations using functions from `dataAccess.js` (3 of 6)

```
    bson: BSON {},
    socketTimeout: 360000,
    monitorCommands: false,
    closed: false,
    destroyed: false,
    lastIsMasterMS: 1,
    [Symbol(kCapture)]: false,
    [Symbol(description)]: StreamDescription {
      address: '127.0.0.1:27017',
      type: 'Standalone',
      minWireVersion: 0,
      maxWireVersion: 8,
      maxBsonObjectSize: 16777216,
      maxMessageSizeBytes: 48000000,
      maxWriteBatchSize: 100000,
      compressors: []
    },
    [Symbol(generation)]: 0,
    [Symbol(lastUseTime)]: 1591938372951,
    [Symbol(queue)]: Map(0) {},
    [Symbol(messageStream)]: MessageStream {
      _readableState: [ReadableState],
      _events: [Object: null prototype],
      _eventsCount: 7,
      _maxListeners: undefined,
      _writableState: [WritableState],
      allowHalfOpen: true,
      bson: BSON {},
      maxBsonMessageSize: 67108864,
      [Symbol(kCapture)]: false,
      [Symbol(buffer)]: [BufferList]
    },
    [Symbol(stream)]: Socket {
      connecting: false,
      _hadError: false,
      _parent: null,
      _host: 'localhost',
      _readableState: [ReadableState],
      _events: [Object: null prototype],
      _eventsCount: 7,
      _maxListeners: undefined,
      _writableState: [WritableState],
      allowHalfOpen: false,
      _sockname: null,
      _pendingData: null,
      _pendingEncoding: '',
      server: null,
      _server: null,
      timeout: 360000,
      _peername: [Object],
      [Symbol(async_id_symbol)]: 357,
      [Symbol(kHandle)]: [TCP],
      [Symbol(kSetNoDelay)]: true,
      [Symbol(lastWriteQueueSize)]: 0,
      [Symbol(timeout)]: Timeout {
```

Figure 4: Output from test script showing successful execution of Create, Read, Update, and Delete operations using functions from `dataAccess.js` (4 of 6)

```
      [Symbol(timeout)]: Timeout {
        _idleTimeout: 360000,
        _idlePrev: [TimersList],
        _idleNext: [Timeout],
        _idleStart: 1160,
        _onTimeout: [Function: bound ],
        _timerArgs: undefined,
        _repeat: null,
        _destroyed: false,
        [Symbol(refed)]: false,
        [Symbol(asyncId)]: 379,
        [Symbol(triggerId)]: 357
      },
      [Symbol(kBuffer)]: null,
      [Symbol(kBufferCb)]: null,
      [Symbol(kBufferGen)]: null,
      [Symbol(kCapture)]: false,
      [Symbol(kBytesRead)]: 0,
      [Symbol(kBytesWritten)]: 0
    },
    [Symbol(ismaster)]: {
      ismaster: true,
      maxBsonObjectSize: 16777216,
      maxMessageSizeBytes: 48000000,
      maxWriteBatchSize: 100000,
      localTime: 2020-06-12T05:06:12.871Z,
      logicalSessionTimeoutMinutes: 30,
      connectionId: 2425,
      minWireVersion: 0,
      maxWireVersion: 8,
      readOnly: false,
      ok: 1
    }
  },
  message: BinMsg {
    parsed: true,
    raw: <Buffer 2d 00 00 00 60 31 00 00 2a 00 00 00 dd 07 00 00 00 00 00 00 18
 00 00 00 10 6e 00 01 00 00 00 01 6f 6b 00 00 00 00 00 00 00 f0 3f 00>,
    data: <Buffer 00 00 00 00 00 18 00 00 00 10 6e 00 01 00 00 00 01 6f 6b 00 00 0
0 00 00 00 00 f0 3f 00>,
    bson: BSON {},
    opts: { promoteLongs: true, promoteValues: true, promoteBuffers: false },
    length: 45,
    requestId: 12640,
    responseTo: 42,
    opCode: 2013,
    fromCompressed: undefined,
    responseFlags: 0,
    checksumPresent: false,
    moreToCome: false,
    exhaustAllowed: false,
    promoteLongs: true,
    promoteValues: true,
    promoteBuffers: false,
    documents: [ [Object] ],
```

Figure 5: Output from test script showing successful execution of Create, Read, Update, and Delete operations using functions from `dataAccess.js` (5 of 6)

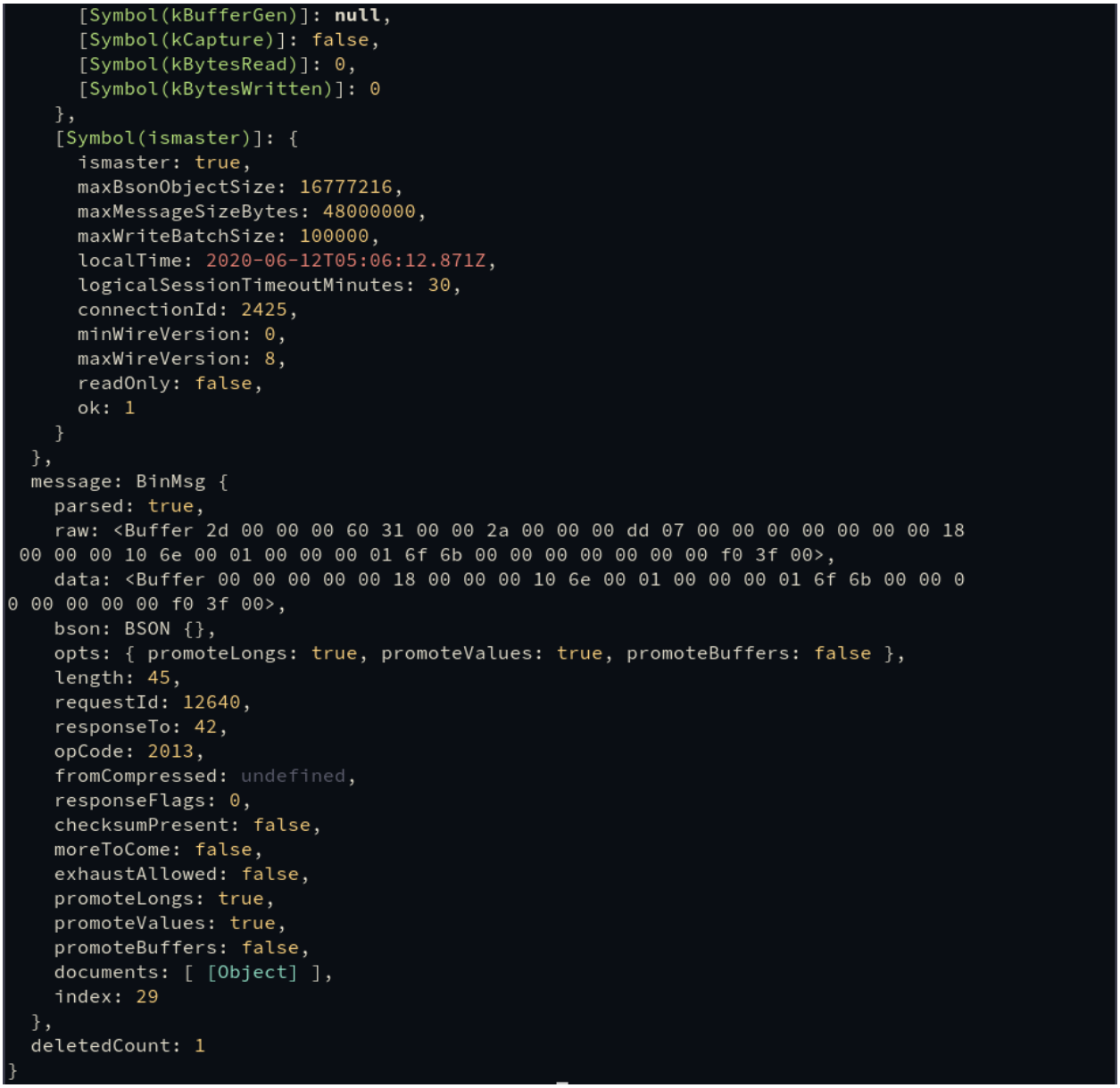


Figure 6: Output from test script showing successful execution of Create, Read, Update, and Delete operations using functions from `dataAccess.js` (6 of 6)