Introduction to Computer Science I
COMP 2406 – Fall 2019

# MongoDB and Node

Dave McKenney
david.mckenney@carleton.ca

## Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

**Use MongoDB inside a Node app**

## Remember...

Remember – all input/output in Node is asynchronous by default

So prepare to work with callback functions

Mongo, however, handles the coordination of multiple reads/writes happening at once (which is an improvement over file-based storage)

## Getting Started with Mongo in Node

To use Mongo inside of a Node.js app:

1. Run the Mongo daemon as before
2. Install the Mongo package: `npm install mongo`
3. Require Mongo in your server:
   `const mongo = require('mongo');`

## Getting Started with Mongo in Node

The Mongo package includes a MongoClient

This is generally what you will use to interact with the underlying Mongo database

```
const client =
require("mongodb").MongoClient;
```

## Getting Started with Mongo in Node

**Once you have a MongoClient variable, you can connect to the database and start issuing queries**

**See ex1-mongo-connect.js to ex9-mongo-delete.js**

## Databases in Apps

Going through the entire connection process for each query is tedious in an app

Ideally, we will connect to our database once, and continue to use that connection until the app stops

## Databases in Apps

We can do this by creating a variable to reference the database object

We can then use that database object whenever we need to perform a query

## Databases in Apps

**Furthermore, we can wait to tell our server to listen until the database connection has been established**

**See ex10-database-server.js**

## Databases in Apps

Another issue: what about sharing a database across multiple modules?

For example, across different routers in an Express-based app?

## Databases in Apps

One solution is to create a module that is solely responsible for providing database connectivity

See database-sharing-module example

This method is a little bit messy, there is a better way

## Databases in Apps

**We can use built-in Express functionality to share a variable across our entire app**

**In the main Express app, we have access to an `app.locals` object**

**This object can be accessed in other routers using req.app.locals or res.app.locals**

## Databases in Apps

So we can initialize the database in the main Express app and set a value in app.locals

We can then access this value from other routers

See database-sharing-express example

## Databases in Apps

We previously saw that Mongo creates its own unique IDs

We can make use of these IDs in our app - this way we don't have to create our own scheme

See database-ids-example

## Databases in Apps

**Finally, we can look at incorporating a database into one of our existing apps**

**We will add some database support into the store app from the Express/Template Engine lectures**

## Databases in Apps

The first step: connect to the database in the main store app

Get the "store" database and save it into app.locals

Add a step to 'upsert' a main page configuration document into the 'config' collection
(really only for convenience in this example)

Modify the index page to load the featured products and store motto from the database

## Databases in Apps

Next, update the GET /products route

Now we can query the database for products

With modular code, we don't have to change that much

## Databases in Apps

Update the GET /products/pid route

Search the database for the given product ID and render the product if a match is found

## Questions

Questions?

Next week: Mongoose