COMP 3123 - FULL STACK DEVELOPMENT I

INTRODUCTION TO MONGO DB

- MongoDB is a free and open-source NoSQL document database used commonly in modern web applications.
- MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- MongoDB works on concept of collection and document.

INTRODUCTION TO MONGO DB

Database

- Database is a physical container for collections.
- Each database gets its own set of files on the file system.
- ***A single MongoDB server typically has multiple databases.***

Collection

- Collection is a group of MongoDB documents.
- ***It is the equivalent of an RDBMS table***.
- A collection exists within a single database.
- Collections do not enforce a schema.
- Documents within a collection can have different fields.
- Typically, all documents in a collection are of similar or related purpose.
- They can hold multiple JSON documents.

Document

- ***A document is a set of key-value pairs***.
- Documents have dynamic schema.
- Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Fields

- 'Fields' or attributes are similar to columns in a SQL table.

Schema

- While Mongo is schema-less, SQL defines a schema via the table definition.
- A Mongoose 'schema' is a document data structure (or shape of the document) that is enforced via the application layer.

Models

- 'Models' are higher-order constructors that take a schema and create an instance of a document equivalent to records in a relational database.

NOSQL DOCUMENTS VS. RELATIONAL TABLES IN SQL

RDBMS	Mongo DB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

NOSQL DOCUMENTS VS. RELATIONAL TABLES IN SQL

MONGO

PEOPLE

```
{
	"Id": 1,
	"FirstName": "Ada",
	"LastName": "Lovelace",
	"Email": "ada.lovelace@gmail.com",
	"Phone": [{
			"Home": "+1.123.456.7890"
		},
		{
			"Work": "+1.111.222.3333"
		}
	]
}
```

```
{
	"Id": 2,
	"FirstName": "Grace",
	"LastName": "Hopper",
	"Email": "grace.hopper@gmail.com"
}
```

```
{
	"Id": 3,
	"FirstName": "Kathy",
	"LastName": "Sierra",
	"Email": "kathy.sierra@gmail.com"
}
```

SQL

PERSON

Id	FirstName	LastName	Email
1	Ada	Lovelace	ada.lovelace@gmail.com
2	Grace	Hopper	grace.hopper@gmail.com
3	Kathy	Sierra	kathy.sierra@gmail.com

PHONE_NUMBER

PersonId	PhoneId	Phone Number	Type
1	1	+1.123.456.7890	Home
1	2	+1.111.222.3333	Work

EXAMPLE OF THE DOCUMENT STRUCTURE - BLOG SITE

```
{
    _id: ObjectId(7df78ad8902c)
    title: 'MongoDB Overview',
    description: 'MongoDB is no sql database',
    by: 'tutorials point',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100,
    comments: [
        {
            user:'user1',
            message: 'My first comment',
            dateCreated: new Date(2011,1,20,2,15),
            like: 0
        },
        {
            user:'user2',
            message: 'My second comments',
            dateCreated: new Date(2011,1,25,7,45),
            like: 5
        }
    ]
}
```

MONGODB DATATYPES

String
This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.

Integer
This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

Boolean
This type is used to store a boolean (true/ false) value.

Double
This type is used to store floating point values.

Min/ Max keys
This type is used to compare a value against the lowest and highest BSON elements.

Arrays
This type is used to store arrays or list or multiple values into one key.

Timestamp
ctimestamp. This can be handy for recording when a document has been modified or added.

Object
This datatype is used for embedded documents.

Null
This type is used to store a Null value.

Symbol
This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

Date
This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

Object ID
This datatype is used to store the document's ID.

Binary data
This datatype is used to store binary data.

Code
This datatype is used to store JavaScript code into the document.

Regular expression
This datatype is used to store regular expression.

WRITING QUERIES AND MONGODB

INSERT

UPDATE

DELETE

SELECT

CRUD

INSTALLATION AND SETUP

You can choose from **one of the following options** (**we are using option #2**):

1. Download the appropriate MongoDB version for your Operating System from the MongoDB Website and follow their installation instructions
2. Create a free shared clusters on mongoDB Cloud Atlas
3. Create a free sandbox database subscription on mLab
4. Install Mongo using Docker if you prefer to use docker

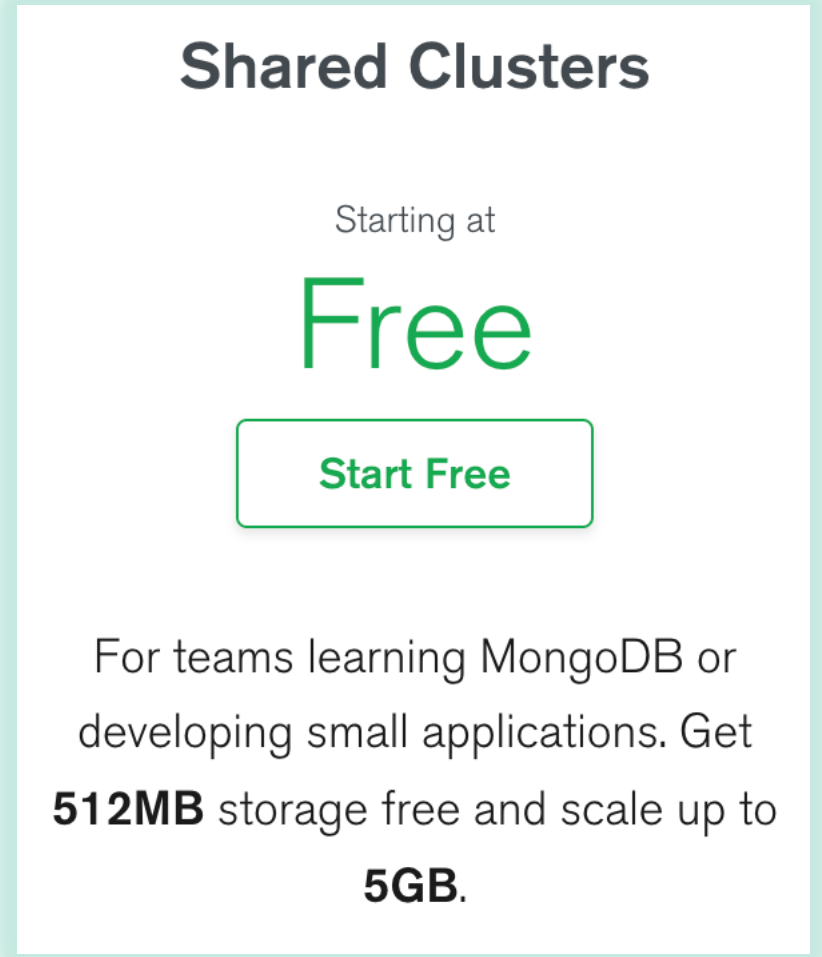
GET STARTED WITH CLOUD ATLAS

MongoDB Atlas provides an easy way to host and manage your data in the cloud.

Please follow the guides from creating an Atlas cluster, connecting to it, inserting data, and querying data.

https://www.mongodb.com/pricing

https://docs.atlas.mongodb.com/getting-started/

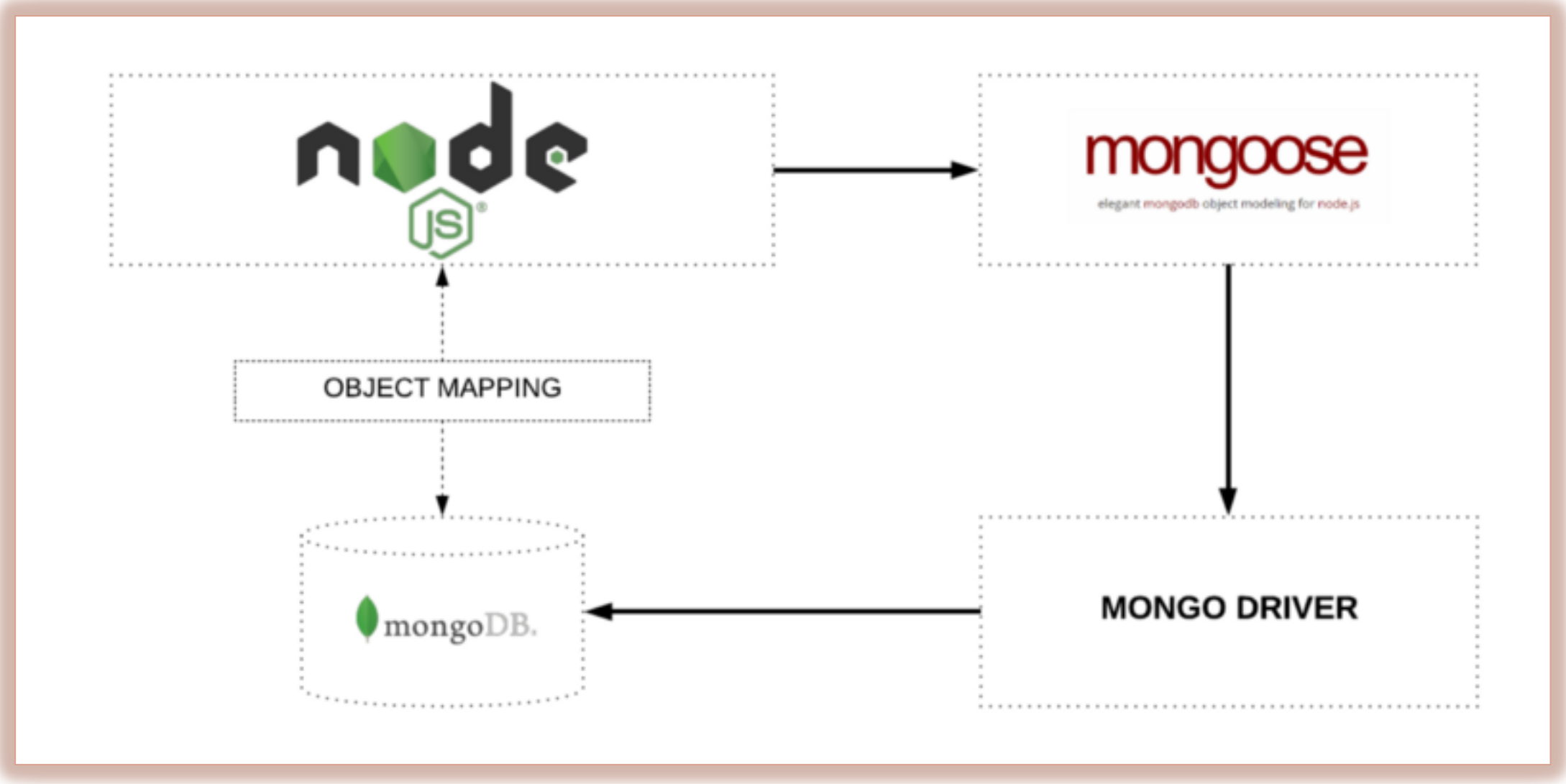


MONGOOSE

- Mongoose provides a straight-forward, schema-based solution to model your application data.
- Mongoose is an **Object Data Modeling (ODM)** library for MongoDB and Node.js.
- It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.
- It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

```
$ npm i express mongoose --save
```

OBJECT MAPPING BETWEEN NODE AND MONGODB MANAGED VIA MONGOOSE



CREATING SERVER

```
const express = require('express');
const mongoose = require('mongoose');
const foodRouter = require('./routes/foodRoutes.js');

const app = express();
app.use(express.json()); // Make sure it comes back as json

mongoose.connect('mongodb+srv://<UserName>:<Password>@cluster0-8vkls.mongodb.net/<DBName>?retryWrites=true&w=majority', {
  useNewUrlParser: true
});

app.use(foodRouter);

app.listen(8081, () => { console.log('Server is running...') });
```

DEFINING YOUR SCHEMA

./models/food.js

```
const mongoose = require('mongoose');

const FoodSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true,
    lowercase: true
  },
  calories: {
    type: Number,
    default: 0,
    validate(value) {
      if (value < 0) throw new Error("Negative calories aren't real.");
    }
  },
});

const Food = mongoose.model("Food", FoodSchema);
module.exports = Food;
```

CRUD OPERATIONS WITH MONGOOSE

- While there are many querying functions available, here are the ones we'll be using in this example:
- **find()**
  - Returns all objects with matching parameters so .find({ name: fish }) would return every object named fish and an empty object will return everything.
- **save()**
  - Save it to our Atlas database.
- **findByIdAndDelete()**
  - Takes the objects id and removes from the database.
- **findByIdAndUpdate()**
  - Takes the objects id and an object to replace it with.
- **deleteOne()** and **deleteMany()**
  - Removes the first or all items from the database.

Web Reference Link

CREATING NEW DOCUMENTS

```
app.post('/food', async (req, res) => {
  const food = new foodModel(req.body);

  try {
    await food.save();
    res.send(food);
  } catch (err) {
    res.status(500).send(err);
  }
});
```

Pass Date as JSON (application/json)

```
{
  "name": "snails",
  "calories": "100"
}
```

http://localhost:8081/food

QUERYING

```
const express = require('express');
const foodModel = require('../models/food');
const app = express();

app.get('/foods', async (req, res) => {
  const foods = await foodModel.find({});

  try {
    res.send(foods);
  } catch (err) {
    res.status(500).send(err);
  }
});

module.exports = app
```

http://localhost:8081/foods

DELETING

```
app.delete('/food/:id', async (req, res) => {
  try {
    const food = await foodModel.findByIdAndDelete(req.params.id)

    if (!food) res.status(404).send("No item found")
    res.status(200).send()
  } catch (err) {
    res.status(500).send(err)
  }
})
```

http://localhost:8081/food/5d1f6c3e4b0b88fb1d257237

UPDATING

```
app.patch('/food/:id', async (req, res) => {
  try {
    await foodModel.findByIdAndUpdate(req.params.id, req.body)
    await foodModel.save()
    res.send(food)
  } catch (err) {
    res.status(500).send(err)
  }
})
```

Pass Date as JSON (application/json)

```
{
  "name": "snails",
  "calories": "100"
}
```

http://localhost:8081/food/5d1f6c3e4b0b88fb1d257237

THANK YOU