INTRO TO DATABASES
CHEATSHEET

Connecting to MongoDB and using Mongoose, a NodeJS library that allows MongoDB integration

MongoDB Terminology

Database
A database is used to store all collections.

Collection
Stores a number of documents. A collection should store documents of the same type, although this constraint is not explicitly enforced by MongoDB.

Document
Each record in a collection is a document. Documents are composed of key / value pairs.

ObjectId
The ObjectId class is the default primary key for a MongoDB document and is usually found in the _id field in an inserted document. All documents are automatically assigned a unique id when added to a collection.

Schema Types

String – Text
Number – An integer
Date – A single moment in time
Buffer – Binary data
Boolean – One of {true, false}
Mixed – "Anything goes". If set to mixed, any other type will be accepted
ObjectId – A unique ID, typically used to identify documents
Array – A list of values

```
const someSchema = new Schema({
    names : [String],
    items : Array,
});
```

You can and should specify the type of the elements inside an array, as in this example. Names has a specified type of elements in the array while items does not.

Mongoose Terminology

Schema
A description of the data structure. A Schema defines the field names and their corresponding time, which helps with validation, defaults, and other options in our models.

Model
A model is the primary tool for interacting with MongoDB. It is a fancy constructor for a document. Models are responsible for creating and reading documents from the underlying MongoDB database.

Validation
By default, MongoDb does not require the documents in a collection to have the same schema. In other words, the documents do not have to have the same fields and data types. Validation allows you to define the structure of the documents in a collection.

If a document does not fit the schema, the insertion will be rejected.

Mongoose Installation

Add the Mongoose package to your project

```
$ npm install --save mongoose
```

Import Mongoose

The Mongoose package needs to be included in every JS file it is used in.

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

Imports the package and saves to the constant mongoose

Setting Up Atlas

Atlas is an online service that hosts Mongo databases. Sign up at *www.mongodb.com/cloud/atlas.* Create a cluster, a user with at least read write permissions, and connect to your project.

INTRO TO DATABASES
CHEATSHEET

Connecting to MongoDB and using Mongoose, a NodeJS library that allows MongoDB integration

Connect to MongoDB

Use your SRV from our MongoDB provider, Atlas, to connect with Mongoose.

```
const mongoConnectionURL = "mongodb+srv://
USERNAME:PASSWORD@DB.mongodb.net/test?
retryWrites=true";
const databaseName = "someDatabaseName";
const options = {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    dbName: databaseName,
};

mongoose.connect(mongoConnectionURL, options)
    .then(() => console.log("Connected."))
    .catch((error) => console.log(`Error
connecting to MongoDB ${error}`));
```

Here mongoURL is set to the SRV given for a cluster on Atlas

Creating a Schema

```
const StudentSchema = new mongoose.Schema({
    name    : String,
    age     : Number,
    classes : [String],
});
```

Here we have created a schema with attributes name, age, and class. Name is a string. Age is a number. And classes are an array of the specific type, String.

Creating a Model

A model is compiled from a Schema.

```
module.exports = mongoose.model("ModelName",
StudentSchema);
```

Here we created a model based on the schema we defined above. Remember that the first argument specifies the name of the *collection*.

Finding a Document

We will be using queries to find documents in a collection.

Defining a Query

A query describes how to filter documents to specify which documents to return.

```
const emptyQuery = {};
```

An empty query, as above, returns all documents.

```
const query = { name: "Tim", age: 21 };
```

This query would return all documents with the name Tim and the age 21.

Finding with a Query

Below are two ways to find using a query.

```
Student.find(query)
    .then((students) =>
     console.log(`found ${students.length}`));
```

Find returns a Promise with all documents that match the query. Here, our anonymous function takes the returned documents and logs some information.

```
Student.findOne(query)
    .then((student) =>
     console.log(`found ${student.name}`));
```

Find one returns only one document that matches the query.

INTRO TO DATABASES
CHEATSHEET

Connecting to MongoDB and using Mongoose, a NodeJS library that allows MongoDB integration

Finding Documents with a Certain Key-Value Pair

Below are two ways to find a document with a certain key-value pair.

```
Student.find({ key : someValue })
    .then((student) => console.log("Found"));
```

```
Student.find({})
    .where(key).equals(someValue)
    .then((student) => console.log("Found"));
```

Updating a Document

```
Student.findOne(query)
    .then((student) => {
        student.fieldToUpdate = newValue;
        student.save()
    });
```

We use find one to ensure that we only update one document.

Deleting a Document

```
Student.deleteOne(query)
    .then((student) => console.log("Deleted"));
```

This only deletes one document that matches the query.

```
Student.deleteMany(query)
    .then((student) =>
     console.log("Deleted many documents"));
```

This only deletes one document that matches the query.

Common Mongo Errors

Make sure you've installed and imported Mongoose.

If connection issues to Atlas, try:
* checking the validity of your Atlas SRV
* restarting your server and try again

Helpful Resources

https://mongoosejs.com/docs/index.html

https://gist.github.com/subfuzion/9236165

https://www.mongodb.com/collateral/quick-reference-cards