Creating, Inserting and Finding in MongoDB
Start up a new database by switching to it.

NOTE: The db does not exist until you create a collection:

use lessondb
Show the current db by running db:
db
Insert data into the lessondb database with this command.

NOTE: This will create the collection automatically.

ALSO, TAKE NOTE: the contents of the insert are basically a JS object, and include an array:

db.places.insert({"continent": "Africa", "country":"Morocco", "majorcities": ["Casablanca", "Fez", "Marrakech"]})

As a class, come up with 3-5 more countries and insert them into the db using the same syntax as above.

Observe where the data was entered in the MongoDB instance (in mongod).

Find all data in a Collection with db.[COLLECTION_NAME].find().

NOTE: the MongoDB _id was created automatically.

This id is specific for each doc in the collection:

db.places.find()

Adding .pretty() makes the data more readable:
db.places.find().pretty()

Find specific data by matching a field:
db.places.find({"continent": "Africa"})
db.places.find({"country": "Morocco"})

Try a few queries with the examples we came up with as a class.

Also, pick something that will find more than one entry so we can see how it will return all matches.

Find specific data by matching an _id:

db.places.find({_id:[COPY AN OBJECTID FROM THE PREVIOUS FIND RESULTS]})

Example: db.places.find({_id: ObjectId("5416fe1d94bcf86cd785439036")})

# MongoDB Create/Insert/Find

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 17\17-NoSQL\01-Activities>mongo
MongoDB shell version v5.0.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("cded9b1b-2f81-4734-9611-93d275762a84") }
MongoDB server version: 5.0.2
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
---
The server generated these startup warnings when booting:
        2021-08-14T10:14:59.618-04:00: Access control is not enabled for the database. Read and write access to data
 and configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

```
> use exampledb
switched to db exampledb
> db
exampledb
> db.places.insert({"country": "USA", "state": "North Carolina", "majorcities": ["Charlotte",
"Wilmington", "Asheville"]})
WriteResult({ "nInserted" : 1 })
> db.places.find()
{ "_id" : ObjectId("612cbcacd1ce45449d51fe83"), "country" : "USA", "state" : "North Carolina",
 "majorcities" : [ "Charlotte", "Wilmington", "Asheville" ] }
> db.places.find().pretty()
{
        "_id" : ObjectId("612cbcacd1ce45449d51fe83"),
        "country" : "USA",
        "state" : "North Carolina",
        "majorcities" : [
                "Charlotte",
                "Wilmington",
                "Asheville"
        ]
}
```

Updating, Deleting and Dropping in MongoDB

Make sure you are using the database, lessondb, that we created earlier.

db
use lessondb

Updating
We update data using db.[COLLECTION_NAME].update()
db.places.update({"country": "Morocco"}, {$set: {"continent": "Antarctica"}})

Note that the above will only update the first entry it matches.

To update multiple entries, you need to add {multi: true}

db.places.update({"country": "Morocco"}, {$set: {"continent": "Antarctica"}}, {multi: true})

Recall from the earlier demo the structure of our document:
db.places.insert({"continent": "Africa", "country": "Morocco", "majorcities": ["Casablanca", "Fez", "Marrakech"]})

What do you think will happen when you run the following command, even though there is not a capital field in the document?
db.places.update({"country": "Morocco"}, {$set: {"capital": "Rabat"}})

Answer: $set will create the field capital

The newly created field can now be updated with the same command:

db.places.update({"country": "Morocco"}, {$set: {"capital": "RABAT"}})

We can update the values in an array with $push:
db.places.update({"country": "Morocco"}, {$push: {"majorcities": "Agadir"}})

Deleting
We delete an entry with db.[COLLECTION_NAME].remove()
db.places.remove({"country": "Morocco"})

We can also empty a collection with db.[COLLECTION_NAME].remove()
db.places.remove({})

Dropping
We drop a collection with db.[COLLECTION_NAME].drop()
db.places.drop()

To drop a database:
db.dropDatabase()

# MongoDB Update/Delete

```
> db.places.update({"state": "North Carolina"}, {$set: {"country": "England"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.places.find().pretty()
{
        "_id" : ObjectId("612cbcacd1ce45449d51fe83"),
        "country" : "England",
        "state" : "North Carolina",
        "majorcities" : [
                "Charlotte",
                "Wilmington",
                "Asheville"
        ]
}
> db.places.update({"state": "North Carolina"}, {$push: {"majorcities": "Greensboro"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.places.find().pretty()
{
        "_id" : ObjectId("612cbcacd1ce45449d51fe83"),
        "country" : "England",
        "state" : "North Carolina",
        "majorcities" : [
                "Charlotte",
                "Wilmington",
                "Asheville",
                "Greensboro"
        ]
}
```

Sorting in MongoDB
The format of a sort follows db.COLLECTION_NAME.find().sort ({FIELD:1})

A value of 1 is for ascending order.

A value of -1 is for descending order.

NOTE: Remember to add .pretty() afterwards so the results are readable!

Create a new db named zoo and insert at least 5 animals with the following attributes:
numLegs - a integer that points to the number of legs.

class - a string that points to the animal's class ("reptile", "mammal" etc).

weight - a integer that points to the animals weight.

name - a string that points to the animal's name.

Example:
```
{
  "name": "Panda",
  "numLegs": 4,
  "class": "mammal",
  "weight": 254
}
```

Using the animals collection that you created:

Sort by id:
The id contains a timestamp, so sorting by id will sort by when they were entered to the database.
```
db.animals.find().sort({ _id:1 });
db.animals.find().sort({ _id:-1 });
```

Sort by an integer - numLegs:
```
db.animals.find().sort({ numLegs:1 });
db.animals.find().sort({ numLegs:-1 });
```

Sort by a string - class:
```
db.animals.find().sort({ class:1 });
db.animals.find().sort({ class:-1 });
```

# MongoDB Sort

```
const express = require("express");
const mongojs = require("mongojs");

const app = express();

const databaseUrl = "zoo";
const collections = ["animals"];

const db = mongojs(databaseUrl, collections);

db.on("error", (error) => {
  console.log("Database Error:", error);
});

app.get("/", (req, res) => {
  res.send("Hello world");
});

app.get("/all", (req, res) => {
  db.animals.find({}, (err, data) => {
    if (err) {
      console.log(err);
    } else {
      res.json(data);
    }
  });
});

// 1: Name: Send JSON response sorted by name in ascending order, e.g. GET "/name"
app.get("/name", (req, res) => {
  db.animals.find().sort({ name: 1 }, (err, found) => {
    if (err) {
      console.log(err);
    } else {
      res.json(found);
    }
  });
});
// 2: Weight: Send JSON response sorted by weight in descending order, , e.g. GET "/weight"
app.get("/weight", (req, res) => {
  db.animals.find().sort({ weight: -1 }, (err, found) => {
    if (err) {
      console.log(err);
    } else {
      res.json(found);
    }
  });
});
// Set the app to listen on port 3000
app.listen(3000, () => {
  console.log("App running on port 3000!");
});
```

# Mongo CRUD pt 1

```
const actionBtn = document.getElementById("action-button");
// new item
const makeNote = document.getElementById("make-new");
// clear all items
const clear = document.getElementById("clear-all");
// delete an item
const results = document.getElementById("results");
const status = document.getElementById("status");

function getResults() {
  // Empty any results currently on the page
  $("#results").empty();
  // Grab all of the current notes
  $.getJSON("/all", function(data) {
    // For each note...
    for (var i = 0; i < data.length; i++) {
      // ...populate #results with a p-tag that includes
              the note's title and object id
      $("#results").prepend("<p class='data-entry' data-id=" +
              data[i]._id + "><span class='dataTitle' data-id=" +
        data[i]._id + ">" + data[i].title + "</span><span class='delete'>
              X</span></p>");
    }
  });
}

// Runs the getResults function as soon as the script is executed
getResults();

// When the #make-new button is clicked
$(document).on("click", "#make-new", function() {
  // AJAX POST call to the submit route on the server
  // This will take the data from the form and send it to the server
  $.ajax({
    type: "POST",
    dataType: "json",
    url: "/submit",
    data: {
      title: $("#title").val(),
      note: $("#note").val(),
      created: Date.now()
    }
  })
  // If that API call succeeds, add the title and a delete button
              for the note to the page
    .then(function(data) {
    // Add the title and delete button to the #results section
      $("#results").prepend("<p class='data-entry' data-id=" +
              data._id + "><span class='dataTitle' data-id=" +
      data._id + ">" + data.title + "</span><span class='delete'>
              X</span></p>");
      // Clear the note and title inputs on the page
      $("#note").val("");
      $("#title").val("");
    });
});

// When the #clear-all button is pressed
$("#clear-all").on("click", function() {
  // Make an AJAX GET request to delete the notes from the db
  $.ajax({
    type: "DELETE",
    dataType: "json",
    url: "/clearall",
    // On a successful call, clear the #results section
    success: function(response) {
      $("#results").empty();
    }
  });
});

// When user clicks the delete button for a note
$(document).on("click", ".delete", function() {
  // Save the p tag that encloses the button
  var selected = $(this).parent();
  // Make an AJAX GET request to delete the specific note
  // this uses the data-id of the p-tag, which is linked to the specific note
  $.ajax({
    type: "DELETE",
    url: "/delete/" + selected.attr("data-id"),
    // On successful call
    success: function(response) {
      // Remove the p-tag from the DOM
      selected.remove();
      // Clear the note and title inputs
      $("#note").val("");
      $("#title").val("");
      // Make sure the #action-button is submit (in case it's update)
      $("#action-button").html("<button id='make-new'>Submit</
button>");
    }
  });
});
// When user click's on note title, show the note, and allow for updates
$(document).on("click", ".dataTitle", function() {
  // Grab the element
  var selected = $(this);
  // Make an ajax call to find the note
  // This uses the data-id of the p-tag, which is linked to the specific note
  $.ajax({
    type: "GET",
    url: "/find/" + selected.attr("data-id"),
    success: function(data) {
      // Fill the inputs with the data that the ajax call collected
      $("#note").val(data.note);
      $("#title").val(data.title);
      // Make the #action-button an update button, so user can
      // Update the note s/he chooses
      $("#action-button").html("<button id='updater' data-id='" +
              data._id + "'>Update</button>");
    }
  });
});

// When user click's update button, update the specific note
$(document).on("click", "#updater", function() {
  // Save the selected element
  var selected = $(this);
  // Make an AJAX POST request
  // This uses the data-id of the update button,
  // which is linked to the specific note title
  // that the user clicked before
  $.ajax({
    type: "POST",
    url: "/update/" + selected.attr("data-id"),
    dataType: "json",
    data: {
      title: $("#title").val(),
      note: $("#note").val()
    },
    // On successful call
    success: function(data) {
      // Clear the inputs
      $("#note").val("");
      $("#title").val("");
      // Revert action button to submit
      $("#action-button").html("<button id='make-new'>
              Submit</button>");
      // Grab the results from the db again, to populate the DOM
      getResults();
    }
  });
});
```

# Mongo CRUD pt 2

```
const express = require("express");
const mongojs = require("mongojs");
const logger = require("morgan");

const app = express();

app.use(logger("dev"));

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

const databaseUrl = "notetaker";
const collections = ["notes"];

const db = mongojs(databaseUrl, collections);

db.on("error", (error) => {
  console.log("Database Error:", error);
});

app.get("/", (req, res) => {
  res.sendFile(path.join(__dirname + "./public/index.html"));
});

// 1. Save a note to the database's collection
// POST: /submit
app.post("/submit", (req, res) => {
  console.log(req.body);
  db.notes.insert(req.body, (error, data) => {
    if (error) {
      res.send(error);
    } else {
      res.send(data);
    }
  });
});

// 2. Retrieve all notes from the database's collection
// GET: /all
app.get("/all", (req, res) => {
  db.notes.find({}, (error, data) => {
    if (error) {
      res.send(error);
    } else {
      res.json(data);
    }
  });
});

// 3. Retrieve one note in the database's collection by it's ObjectId
// (remember, mongojs.ObjectId(IdYouWantToFind)
// GET: /find/:id
app.get("/find/:id", (req, res) => {
  db.notes.findOne(
    {
      _id: mongojs.ObjectId(req.params.id)
    },
    (error, data) => {
      if (error) {
        res.send(error);
      } else {
        res.send(data);
      }
    }
  );
});

// 4. Update one note in the database's collection by it's ObjectId
// (remember, mongojs.ObjectId(IdYouWantToFind)
// POST: /update/:id
app.post("/update/:id", (req, res) => {
  db.notes.update(
    {
      _id: mongojs.ObjectId(req.params.id)
    },
    {
      $set: {
        title: req.body.title,
        note: req.body.note,
        modified: Date.now()
      }
    },
    (error, data) => {
      if (error) {
        res.send(error);
      } else {
        res.send(data);
      }
    }
  );
});

// 5. Delete one note from the database's collection by it's ObjectId
// (remember, mongojs.ObjectId(IdYouWantToFind)
// DELETE: /delete/:id
app.delete("/delete/:id", (req, res) => {
  db.notes.remove(
    {
      _id: mongojs.ObjectID(req.params.id)
    },
    (error, data) => {
      if (error) {
        res.send(error);
      } else {
        res.send(data);
      }
    }
  );
});

// 6. Clear the entire note collection
// DELETE: /clearall
app.delete("/clearall", (req, res) => {
  db.notes.remove({}, (error, response) => {
    if (error) {
      res.send(error);
    } else {
      res.send(response);
    }
  });
});

// Listen on port 3000
app.listen(3000, () => {
  console.log("App running on port 3000!");
});
```

```
<!DOCTYPE html>
<html lang="en-us">
<head>
  <meta charset="UTF-8">
  <title>NoteTaker</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <style media="screen">
    body {
      text-align: center;
      background-color: cornflowerblue;
    }

    .primary {
      color: #64edbd;
    }

    h1 {
      font-size: 40px;
      color: white;
    }

    h2, h4 {
      color: white;
    }

    .delete {
      margin-left: 20px;
      color: red;
      cursor: pointer;
      text-align: right;
    }

    input {
      width: 400px;
      height: 50px;
      margin-left: auto;
      margin-right: auto;
      font-size: 42px;
      background-color: antiquewhite;
    }

    textarea {
      width: 400px;
      height: 400px;
      margin-left: auto;
      margin-right: auto;
      font-size: 24px;
      background-color: antiquewhite;
    }

    .data-title {
      cursor: pointer;
      font-size: 24px;
    }

    #results {
      width: 400px;
      height: 200px;
      overflow: auto;
      margin-left: auto;
      margin-right: auto;
      background-color: antiquewhite;
    }

    #action-button {
      display: inline-block;
    }

    #buttons {
      margin: 20px;
    }
  </style>
</head>
<body>
  <h1>Mongo Note App</h1>
  <h2>Status: <span id="status" class="primary">Creating</span></h2>
  <h4>Create a new note by submitting its title and content below</h4>
  <h4>Click on a note to edit its title or content</h4>
  <div id="user-input">
    <p>Title</p>
    <input type="text" id="title" />
    <br>
    <p>Note</p>
    <textarea id="note"></textarea>
    <div id="buttons">
      <div id="action-button">
        <button id="make-new">Submit</button>
      </div>
      <button id="clear-all">Delete All Notes</button>
    </div>
  </div>
  <div id="results"></div>
  <script src="app.js"></script>
</body>
</html>
```

# MongoJS Review pt.1

```
const express = require("express");
const mongojs = require("mongojs");
const logger = require("morgan");

const app = express();
app.use(logger("dev"));

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

const databaseUrl = "warmup";
const collections = ["books"];

const db = mongojs(databaseUrl, collections);

db.on("error", (error) => {
  console.log("Database Error:", error);
});

// Routes
// ======

// Post a book to the mongoose database
app.post("/submit", ({ body }, res) => {
  // Save the request body as an object called book
  const book = body;
  book.read = false;
  db.books.insert(book, (err, data) => {
    res.json(data);
  });
  db.books.save(book, (error, saved) => {
    if (error) {
      console.log(error);
    } else {
      res.send(saved);
    }
  });
});

// Find all books marked as read
app.get("/read", (req, res) => {
  db.books.find({ read: true }, (error, found) => {
    if (error) {
      console.log(error);
    } else {
      res.json(found);
    }
  });
});

// Find all books marked as unread
app.get("/unread", (req, res) => {
  db.books.find({ read: false }, (error, found) => {
    if (error) {
      console.log(error);
    } else {
      res.json(found);
    }
  });
});

// Mark a book as having been read
  // Remember: when searching by an id,
            the id needs to be passed in
  // as (mongojs.ObjectId(IdYouWantToFind))
app.put("/markread/:id", ({ params }, res) => {
  db.books.update(
    {
      _id: mongojs.ObjectId(params.id),
    },
    {
      $set: {
        read: true,
      },
    },
    (error, edited) => {
      if (error) {
        console.log(error);
        res.send(error);
      } else {
        console.log(edited);
        res.send(edited);
      }
    }
  );
});

// Mark a book as having been not read
app.put("/markunread/:id", ({ params }, res) => {
  db.books.update(
    {
      _id: mongojs.ObjectId(params.id),
    },
    {
      $set: {
        read: false,
      },
    },
    (error, edited) => {
      if (error) {
        console.log(error);
        res.send(error);
      } else {
        console.log(edited);
        res.send(edited);
      }
    }
  );
});

// Listen on port 3000
app.listen(3000, () => {
  console.log("App running on port 3000!");
});
```

# MongoJS Review pt.2

```
// Click Events

// Click event to add a book to the db
$("#addbook").on("click", function() {
  $.ajax({
    type: "POST",
    url: "/submit",
    dataType: "json",
    data: {
      title: $("#title").val(),
      author: $("#author").val(),
      created: Date.now()
    }
  })
    .then(function(data) {
      console.log(data);
      getUnread();
      $("#author").val("");
      $("#title").val("");
    }
    );
  return false;
});

// Click event to mark a book as read
$(document).on("click", ".markread", function() {
  var thisId = $(this).attr("data-id");
  $.ajax({
    type: "PUT",
    url: "/markread/" + thisId
  });
  $(this).parents("tr").remove();
  getRead();
});

// Click event to mark a book as not read
$(document).on("click", ".markunread", function() {
  var thisId = $(this).attr("data-id");
  $.ajax({
    type: "PUT",
    url: "/markunread/" + thisId
  });
  $(this).parents("tr").remove();
  getUnread();
});
```

```
// Load unread books and render them to the screen
function getUnread() {
  $("#unread").empty();
  $.getJSON("/unread", function(data) {
    for (var i = 0; i < data.length; i++) {
      $("#unread").prepend("<tr><td>" + data[i].title +
            "</td><td>" + data[i].author +
        "</td><td><button class='markread' data-id='" +
            data[i]._id + "'>Mark Read</button>
            </td></tr>");
    }
    $("#unread").prepend("<tr><th>Title</th>
            <th>Author</th>
            <th>Read/Unread</th></tr>");
  });
}

// Load read books and render them to the screen
function getRead() {
  $("#read").empty();
  $.getJSON("/read", function(data) {
    for (var i = 0; i < data.length; i++) {
      $("#read").prepend("<tr><td>" + data[i].title +
            "</td><td>" + data[i].author +
        "</td><td><button class='markunread' data-id='" +
            data[i]._id + "'>Mark Unread</button>
            </td></tr>");
    }
    $("#read").prepend("<tr><th>Title</th>
            <th>Author</th>
            <th>Read/Unread</th></tr>");
  });
}

// Calling our functions
getUnread();
getRead();
```

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="UTF-8">
    <title>Book Tracker</title>
    <style>
      #unread, #read {
        width: 40%;
        background-color: antiquewhite;
      }
      #unread { float: left; }
      #read { float: right; }
      table, td {
        border-width: 1px;
        border-style: solid;
        border-color: black;
      }
      #wrapper {
        width: 960px;
        padding-top: 10%;
        margin-left: auto;
        margin-right: auto;
        text-align: center;
      }
      body {
        font-family: sans-serif;
        background-color: #C0FFEE;
      }
      #results { margin-top: 50px; }
      #userinput { font-size: 24px; }
      input {
        width: 50%;
        height: 36px;
        font-size: 24px;
      }
    </style>
  </head>
  <body>
    <div id="wrapper">
      <h1>Book Tracker 4000</h1>
      <h2><i>The latest and greatest in book tracking technology!</i></h2>
      <form id="userinput">
        <input type="text" id="title" placeholder="Book Title"/>
        <br />
        <input type="text" id="author" placeholder="Book Author"/>
        <br />
        <button type="submit" id="addbook">Add New</button>
      </form>
      <div id="results">
        <table id="unread">
          <tr><th>Title</th><th>Author</th><th>Read/Unread</th></tr>
        </table>
        <table id="read">
          <tr><th>Title</th><th>Author</th><th>Read/Unread</th></tr>
        </table>
      </div>
    </div>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

# Mongoose

```
{
  "name": "mongoose_schema",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "watch": "nodemon server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.3",
    "mongoose": "^5.3.16",
    "morgan": "^1.9.1"
  }
}
```

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="UTF-8">
    <title>User Schema</title>
    <style>
      input {
        display: block;
        font-size: 24px;
        height: 36px;
      }
    </style>
  </head>
  <body>
    <form action="/submit" method="post">
      <input type="text" name="username" placeholder="username">
      <input type="password" name="password" placeholder="password">
      <input type="text" name ="email" placeholder="email">
      <input type="submit">
    </form>
  </body>
</html>
```

```
const express = require("express");
const logger = require("morgan");
const mongoose = require("mongoose");

const PORT = process.env.PORT || 3000;

const User = require("./userModel.js");
const app = express();

app.use(logger("dev"));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

mongoose.connect(process.env.MONGODB_URI ||
            "mongodb://localhost/userdb", { useNewUrlParser: true });
app.post("/submit", ({body}, res) => {
  User.create(body)
    .then(dbUser => {
      res.json(dbUser);
    })
    .catch(err => {
      res.json(err);
    });
});

app.listen(PORT, () => {
  console.log(`App running on port ${PORT}!`);
});
```

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

const UserSchema = new Schema({
  username: {
    type: String,
    trim: true,
    required: "Name is Required",
  },
  Password: {
    type: String,
    validate: [({ length }) => length >= 6, "Password should be longer."],
  },
  email: {
    type: String,
    match: [/.+@.+\..+/, "Please enter a valid e-mail address"],
  },
  userCreated: {
    type: Date,
    default: Date.now,
  },
});
const User = mongoose.model("User", UserSchema);

module.exports = User;
```

# Custom Methods

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <meta name="viewport">
  <title>Mongoose</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" cros-
sorigin="anonymous">
</head>
<body>
<div class="container col-md-4 col-md-offset-4">
  <form action="/submit" method="post">
    <h2>Create A User</h2>
    <input type="text" class="form-control" name="username" placeholder="username" autofo-
cus>
    <input type="text" class="form-control" name="password" placeholder="password" autofo-
cus>
    <input type="text" class="form-control" name="email" placeholder="email" autofocus>
    <button class="btn btn-lg btn-primary btn-block" id="submit" type="submit">Submit</button>
  </form>
</div>
</body>
</html>
```

```
const express = require("express");
const logger = require("morgan");

const mongoose = require("mongoose");
const PORT = process.env.PORT || 3000;

const User = require("./userModel.js");
const app = express();
app.use(logger("dev"));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(express.static("public"));
mongoose.connect(process.env.MONGODB_URI ||
            "mongodb://localhost/custommethods", { useNewUrlParser: true });

app.post("/submit", ({ body }, res) => {
  const user = new User(body);
  user.coolifier();
  user.makeCool();
  User.create(user)
    .then(dbUser => {
      res.json(dbUser);
    })
    .catch(err => {
      res.json(err);
    });
});

app.listen(PORT, () => {
  console.log(`App running on port ${PORT}!`);
});
```

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

const UserSchema = new Schema({
  username: {
    type: String,
    trim: true,
    required: "Username is Required"
  },
  password: {
    type: String,
    trim: true,
    required: "Password is Required",
    validate: [({ length }) => length >= 6, "Password should be longer."]
  },
  email: {
    type: String,
    unique: true,
    match: [/.+@.+\..+/, "Please enter a valid e-mail address"]
  },
  userCreated: {
    type: Date,
    default: Date.now
  },
  isCool: {
    type: Boolean,
    default: false
  }
});

UserSchema.methods.coolifier = function() {
  this.username = `${this.username}...the Coolest!`;
  return this.username;
};

UserSchema.methods.makeCool = function() {
  this.isCool = true;
  return this.isCool;
};
const User = mongoose.model("User", UserSchema);

module.exports = User;
```

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <meta name="viewport">
  <title>Mongoose Populate</title>
  <link rel="stylesheet" href=bootstrap
</head>
<body>
<div class="container col-md-4 col-md-offset-4">
  <form action="/submit" method="post">
    <h2>Create A Book</h2>
    <input type="text" class="form-control" name="author"
           placeholder="author" autofocus>
    <input type="text" class="form-control" name="title"
           placeholder="title" autofocus>
    <button class="btn btn-lg btn-primary btn-block" id="submit" type="submit">
           Submit</button>
    <p><a href="/books">See your books!</a></p>
    <p><a href="/library">See your library!</a></p>
    <p><a href="/populated">See your library, populated with books!</a></p>
  </form>
</div>
</body>
</html>
```

```
// Exporting an object containing all of our models
module.exports = {
  Book: require("./Book"),
  Library: require("./Library")
};
```

---

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const LibrarySchema = new Schema({
  name: {
    type: String,
    unique: true},
  books: [{
      type: Schema.Types.ObjectId,
      ref: "Book"}]});
const Library = mongoose.model("Library", LibrarySchema);

module.exports = Library;
```

---

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const BookSchema = new Schema({
  author: String,
  title: String});
const Book = mongoose.model("Book", BookSchema);

module.exports = Book;
```

# Populate

```
const express = require("express");
const logger = require("morgan");
const mongoose = require("mongoose");

const PORT = process.env.PORT || 3000;

const db = require("./models");
const app = express();

app.use(logger("dev"));

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

mongoose.connect(process.env.MONGODB_URI
           || "mongodb://localhost/populate",
           { useNewUrlParser: true });

db.Library.create({ name: "Campus Library" })
  .then(dbLibrary => {
    console.log(dbLibrary);
  })
  .catch(({message}) => {
    console.log(message);
  });

app.post("/submit", ({body}, res) => {
  db.Book.create(body)
    .then(({_id}) =>
           db.Library.findOneAndUpdate({},
           { $push: { books: _id } }, { new: true }))
    .then(dbLibrary => {
      res.json(dbLibrary);
    })
    .catch(err => {
      res.json(err);
    });
});

app.get("/books", (req, res) => {
  db.Book.find({})
    .then(dbBook => {
      res.json(dbBook);
    })
    .catch(err => {
      res.json(err);
    });
});

app.get("/library", (req, res) => {
  db.Library.find({})
    .then(dbLibrary => {
      res.json(dbLibrary);
    })
    .catch(err => {
      res.json(err);
    });
});

app.get("/populated", (req, res) => {
  db.Library.find({})
    .populate("books")
    .then(dbLibrary => {
      res.json(dbLibrary);
    })
    .catch(err => {
      res.json(err);
    });
});

app.listen(PORT, () => {
  console.log(`App running on port ${PORT}!`);
});
```

# Opening Indexes

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Making An IndexedDB Connection</title>
  </head>
  <body>
    <h1>Open Your Developer Tools > Application > IndexedDB</h1>
  </body>
</html>
<script>
  // We request a database instance.
  const request = indexedDB.open("firstDatabase", 1);

  // This returns a result that we can then manipulate.
  request.onsuccess = event => {
    console.log(request.result);
  };
</script>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Making An IndexedDB Connection</title>
  </head>
  <body>
    <h1>Open Your Developer Tools > Application > IndexedDB</h1>
    <script>
      const request = window.indexedDB.open("firstDatabase", 1);

      request.onsuccess = event => {
        console.log(request.result.name);
      };
    </script>
  </body>
</html>
```

- Use the [open](https://developer.mozilla.org/en-US/docs/Web/API/IDBFactory/open) docs to learn about the arguments it takes.

* You can `console.log` the `request` to so what attributes are available to you.

# Creating Object Stores

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Creating An Object Store</title>
  </head>
  <body>
    <h1>Open Your Developer Tools > Application > IndexedDB</h1>
    <script>
      // We request a database instance
      const request = window.indexedDB.open("toDoList", 1);

      // Create an object store inside the onupgradeneeded method.
      request.onupgradeneeded = ({ target }) => {
        const db = target.result;
        const objectStore = db.createObjectStore("toDoList");
      };

      // On success console the result.
      request.onsuccess = event => {
        console.log(request.result);
      };
    </script>
  </body>
</html>
```

Hint(s)
* Use the [open](https://developer.mozilla.org/en-US/docs/Web/API/IDBFactory/open) docs to learn about the arguments it takes.
* You can `console.log` the `request` to so what attributes are available to you.

Bonus
* Use the [keyPath](https://developer.mozilla.org/en-US/docs/Web/API/IDBObjectStore/keyPath) docs to research what a `keyPath` is and how to add it to your `objectStore`.

# Creating Indexes

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Creating Indexes</title>
  </head>

  <body>
    <h1>Open Your Developer Tools > Application > IndexedDB</h1>
    <script>

      const request = window.indexedDB.open("toDoList", 1);

      request.onupgradeneeded = ({ target }) => {
        const db = target.result;
        const objectStore = db.createObjectStore("todoList");
        objectStore.createIndex("icebox", "icebox");
        objectStore.createIndex("inprogress", "inprogress");
        objectStore.createIndex("complete", "complete");
      };

      request.onsuccess = event => {
        console.log(request.result);
      };

    </script>
  </body>
</html>
```

* Use the [createIndex](https://developer.mozilla.org/en-US/docs/Web/API/IDBObjectStore/createIndex) docs if you are stuck.

# Adding/ Getting Data

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Adding and Getting Data</title>
  </head>
  <body>
    <h1>Open Your Developer Tools > Application > IndexedDB</h1>
    <script>
      const request = window.indexedDB.open("toDoList", 1);

      // Create schema
      request.onupgradeneeded = event => {
        const db = event.target.result;

        // Creates an object store with a listID keypath that can be used to query on.
        const toDoListStore = db.createObjectStore("toDoList", {keyPath: "listID"});

        // Creates a statusIndex that we can query on.
        toDoListStore.createIndex("statusIndex", "status");
      }

      // Opens a transaction, accesses the toDoList objectStore and statusIndex.
      request.onsuccess = () => {
        const db = request.result;
        const transaction = db.transaction(["toDoList"], "readwrite");
        const toDoListStore = transaction.objectStore("toDoList");
        const statusIndex = toDoListStore.index("statusIndex");

        // Adds data to our objectStore
        toDoListStore.add({ listID: "1", status: "complete" });
        toDoListStore.add({ listID: "2", status: "in-progress" });
        toDoListStore.add({ listID: "3", status: "complete" });
        toDoListStore.add({ listID: "4", status: "backlog" });

        // Return an item by keyPath
        const getRequest = toDoListStore.get("1");
        getRequest.onsuccess = () => {
          console.log(getRequest.result);
        };

        // Return an item by index
        const getRequestIdx = statusIndex.getAll("complete");
        getRequestIdx.onsuccess = () => {
          console.log(getRequestIdx.result);
        };
      };
    </script>
  </body>
</html>
```

# Updating Data

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Adding and Getting Data</title>
  </head>
  <body>
    <h1>Open Your Developer Tools > Application > IndexedDB</h1>
    <script>
      const request = window.indexedDB.open("toDoList", 1);

      // Create schema
      request.onupgradeneeded = event => {
        const db = event.target.result;

        // Creates an object store with a listID keypath that can be used to query on.
        const toDoListStore = db.createObjectStore("toDoList", {
          keyPath: "listID"
        });

        // Creates a statusIndex that we can query on.
        toDoListStore.createIndex("statusIndex", "status");
      };

      // Opens a transaction, accesses the toDoList objectStore and statusIndex.
      request.onsuccess = () => {
        const db = request.result;
        const transaction = db.transaction(["toDoList"], "readwrite");
        const toDoListStore = transaction.objectStore("toDoList");
        const statusIndex = toDoListStore.index("statusIndex");

        // Adds data to our objectStore
        toDoListStore.add({ listID: "1", status: "complete" });
        toDoListStore.add({ listID: "2", status: "in-progress" });
        toDoListStore.add({ listID: "3", status: "complete" });
        toDoListStore.add({ listID: "4", status: "backlog" });

        // Opens a Cursor request and iterates over the documents.
        const getCursorRequest = toDoListStore.openCursor();
        getCursorRequest.onsuccess = e => {
          const cursor = e.target.result;
          if (cursor) {
            console.log(cursor.value);
            cursor.continue();
          } else {
            console.log("No documents left!");
          }
        };
      };
    </script>
  </body>
</html>
```

# Mini-Project pt.1

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
                initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <meta name="theme-color" content="#317EFB" />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/
                css/font-awesome.min.css"/>
    <link rel="stylesheet" href="/style.css" />
    <title>Budget Tracker</title>
  </head>
  <body>
    <div class="wrapper">
      <div class="total">
        <div>Your total is: $<span id="total">0</span></div>
      </div>
      <form id="budget-form">
        <label for="t-name">Transaction Name:</label>
        <input type="text" id="t-name" placeholder="Name" />
        <label for="t-amount">Transaction Amount:</label>
        <input type="number" min="0" id="t-amount"
                placeholder="Amount" />
        <div class="transaction-buttons">
          <button id="add-btn" type="submit"><i class="fa fa-plus"></i>
                Add Funds</button>
          <button id="sub-btn" type='submit'>
            <i class="fa fa-minus"></i> Subtract Funds
          </button>
        </div>
        <p role="alert" class="error"></p>
      </form>
      <div class="transactions">
        <table>
          <thead>
            <th>Transaction</th>
            <th>Amount</th>
          </thead>
          <tbody id="tbody"></tbody>
        </table>
      </div>
      <canvas id="my-chart"></canvas>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
    <script src="/db.js"></script>
    <script src="/index.js"></script>
  </body>
</html>
```

```
let transactions = [];
let myChart;

fetch('/api/transaction')
  .then((response) => response.json())
  .then((data) => {
    // save db data on global variable
    transactions = data;
    populateTotal();
    populateTable();
    populateChart();
  });

function populateTotal() {
  // reduce transaction amounts to a single total value
  const total = transactions
    .reduce((total, t) => {
      return total + parseFloat(t.value);
    }, 0)
    .toFixed(2);
  const totalEl = document.querySelector('#total');
  totalEl.textContent = total;
}

function populateTable() {
  const tbody = document.querySelector('#tbody');
  tbody.innerHTML = '';
  transactions.forEach((transaction) => {
    // create and populate a table row
    const tr = document.createElement('tr');
    tr.innerHTML = `
      <td>${transaction.name}</td>
      <td>${transaction.value}</td>
    `;
    tbody.appendChild(tr);
  });}

function populateChart() {
  // copy array and reverse it
  const reversed = transactions.slice().reverse();
  let sum = 0;
  // create date labels for chart
  const labels = reversed.map((t) => {
    const date = new Date(t.date);
    return `${date.getMonth() + 1}/${date.getDate()}/
${date.getFullYear()}`;
  });
  // create incremental values for chart
  const data = reversed.map((t) => {
    sum += parseInt(t.value);
    return sum;
  });
  // remove old chart if it exists
  if (myChart) {
    myChart.destroy();
  }
  const ctx = document.getElementById('my-chart').getContext('2d');

  myChart = new Chart(ctx, {
    type: 'line',
    data: {
      labels,
      datasets: [
        {label: 'Total Over Time',
          fill: true,
          backgroundColor: '#6666ff',
          data,
        },],},});}
function sendTransaction(isAdding) {
  const nameEl = document.querySelector('#t-name');
  const amountEl = document.querySelector('#t-amount');
  const errorEl = document.querySelector('form .error');
  // validate form
  if (nameEl.value === '' || amountEl.value === '') {
    errorEl.textContent = 'Missing Information';
    return;
  } else {
    errorEl.textContent = '';
  }
  // create record
  const transaction = {
    name: nameEl.value,
    value: amountEl.value,
    date: new Date().toISOString(),
  };
  // if subtracting funds, convert amount to negative number
  if (!isAdding) {
    transaction.value *= -1;
  }
  // add to beginning of current array of data
  transactions.unshift(transaction);
  // re-run logic to populate ui with new record
  populateChart();
  populateTable();
  populateTotal();
  // also send to server
  fetch('/api/transaction', {
    method: 'POST',
    body: JSON.stringify(transaction),
    headers: {
      Accept: 'application/json, text/plain, */*',
      'Content-Type': 'application/json',
    },})
    .then((response) => response.json())
    .then((data) => {
      if (data.errors) {
        errorEl.textContent = 'Missing Information';
      } else {
        // clear form
        nameEl.value = '';
        amountEl.value = '';
      }})
    .catch((err) => {
      // fetch failed, so save in indexed db
      console.log('save record');
      saveRecord(transaction);
      // clear form
      nameEl.value = '';
      amountEl.value = '';
    });
}

document.querySelector('#add-btn').addEventListener('click',
                function (event) {
  event.preventDefault();
  sendTransaction(true);
});

document.querySelector('#sub-btn').addEventListener('click',
                function (event) {
  event.preventDefault();
  sendTransaction(false);
});
```

```
let db;
let budgetVersion;

// Create a new db request for a "budget" database.
const request = indexedDB.open('BudgetDB', budgetVersion || 21);

request.onupgradeneeded = function (e) {
  console.log('Upgrade needed in IndexDB');
  const { oldVersion } = e;
  const newVersion = e.newVersion || db.version;
  console.log(`DB Updated from version ${oldVersion} to ${newVersion}`);
  db = e.target.result;
  if (db.objectStoreNames.length === 0) {
    db.createObjectStore('BudgetStore', { autoIncrement: true });
  }};

request.onerror = function (e) {
  console.log(`Woops! ${e.target.errorCode}`);
};

function checkDatabase() {
  console.log('check db invoked');
  // Open a transaction on your BudgetStore db
  let transaction = db.transaction(['BudgetStore'], 'readwrite');
  // access your BudgetStore object
  const store = transaction.objectStore('BudgetStore');
  // Get all records from store and set to a variable
  const getAll = store.getAll();
  // If the request was successful
  getAll.onsuccess = function () {
    // If there are items in the store, we need to bulk add them when we are back online
    if (getAll.result.length > 0) {
      fetch('/api/transaction/bulk', {
        method: 'POST',
        body: JSON.stringify(getAll.result),
        headers: {
          Accept: 'application/json, text/plain, */*',
          'Content-Type': 'application/json',
        },})
        .then((response) => response.json())
        .then((res) => {
          // If our returned response is not empty
          if (res.length !== 0) {
            // Open another transaction to BudgetStore with the ability to read and write
            transaction = db.transaction(['BudgetStore'], 'readwrite');
            // Assign the current store to a variable
            const currentStore = transaction.objectStore('BudgetStore');
            // Clear existing entries because our bulk add was successful
            currentStore.clear();
            console.log('Clearing store □');
          }});}};}

request.onsuccess = function (e) {
  console.log('success');
  db = e.target.result;
  // Check if app is online before reading from db
  if (navigator.onLine) {
    console.log('Backend online! □□');
    checkDatabase();
  }};

const saveRecord = (record) => {
  console.log('Save record invoked');
  // Create a transaction on the BudgetStore db with readwrite access
  const transaction = db.transaction(['BudgetStore'], 'readwrite');
  // Access your BudgetStore object store
  const store = transaction.objectStore('BudgetStore');
  // Add record to your store with add method.
  store.add(record);
};

// Listen for app coming back online
window.addEventListener('online', checkDatabase);
```

# Mini-Project pt.2

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

const transactionSchema = new Schema({
  name: {
    type: String,
    trim: true,
    required: "Enter a name for transaction"
  },
  value: {
    type: Number,
    required: "Enter an amount"
  },
  date: {
    type: Date,
    default: Date.now
  }
});
const Transaction = mongoose.model("Transaction", transactionSchema);

module.exports = Transaction;
```

```
{
  "name": "budget-app",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/coding-boot-camp/unit18hw.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/coding-boot-camp/unit18hw/issues"
  },
  "homepage": "https://github.com/coding-boot-camp/unit18hw#readme",
  "dependencies": {
    "express": "^4.17.1",
    "mongoose": "^5.5.15"
  }
}
```

```
const router = require("express").Router();
const Transaction = require("../models/transaction.js");

router.post("/api/transaction", ({ body }, res) => {
  Transaction.create(body)
    .then(dbTransaction => {
      res.json(dbTransaction);
    })
    .catch(err => {
      res.status(400).json(err);
    });
});

router.post("/api/transaction/bulk", ({ body }, res) => {
  Transaction.insertMany(body)
    .then(dbTransaction => {
      res.json(dbTransaction);
    })
    .catch(err => {
      res.status(400).json(err);
    });
});

router.get("/api/transaction", (req, res) => {
  Transaction.find({})
    .sort({ date: -1 })
    .then(dbTransaction => {
      res.json(dbTransaction);
    })
    .catch(err => {
      res.status(400).json(err);
    });
});

module.exports = router;
```

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

const PORT = process.env.PORT || 3000;

const app = express();

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(express.static("public"));

mongoose.connect(process.env.MONGODB_URI || "mongodb://localhost/budget", {
  useNewUrlParser: true,
  useFindAndModify: false
});

// routes
app.use(require("./routes/api.js"));

app.listen(PORT, () => {
  console.log(`App running on port ${PORT}!`);
});
```