INFO3139 Lab 6

Rev 1.0

General JavaScript Topic #5 – When to use Array.map (…)

This method takes an array and applies some procedure to its elements so that you get **a new array** with modified elements. For example:

1. Create a new folder called week4\class1

2. Create a file called **map_example1.js** with the following code:

```
let arr = [2, 3, 5, 7];
let newArray = arr.map((element) => element * 2);
console.log(`Original Array Contents - ${arr}`);
console.log(`New Array Contents - ${newArray}`);
```

3. Then execute the code:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node map_example1
Original Array Contents - 2,3,5,7
New Array Contents - 4,6,10,14
```

4. Note, we could still code a map loop without even defining a new array. Add another file called **map_example2.js** with the following (notice that it doesn't define a new array).

```
let arr = [2, 3, 5, 7];
arr.map((element) => {
  let x = element * 2;
});
console.log(`Original Array Contents - ${arr}`);
```

5. Execute the new code:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node map_example2
Original Array Contents - 2,3,5,7
```

According to MDN, .map method builds a new array, using it when you aren't using the returned array is considered **an anti-pattern** so you shouldn't use it in those instances. Instead use the forEach or for...of constructs we looked at in an earlier class instead.

MongoDB cont'd

Last class we set up the environment variables and configuration to allow us to connect to a database and add a document to a collection. We will get a little more ambitious and add a series of documents from an array and utilize both **.map** and **Promise.allSettled** techniques.

1. Copy the week3\class2\.env, config.js and db_routines.js into week4\class1.

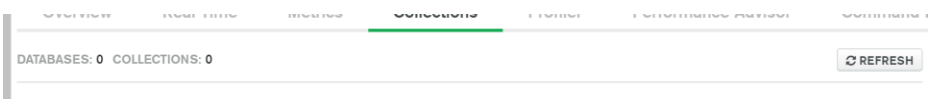
2. Add another file called week4\class1**mongo_exercise3.js** with the following (change my name to yours):

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const addSomeUsers = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();
    let resultArray = someUsers.map(async (user) => {
      let result = await dbRtns.addOne(db, "users", user);
      result.value.acknowledged
        ? console.log(`added document to users collection`)
        : console.log(`document not added to users collection`);
    });
    resultArray.forEach((result) => console.log(result));
  } catch (err) {
    console.log(err);
  } finally {
    process.exit();
  }
};

addSomeUsers();
```

3. Run the mongo_exercise3.js code now:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise3
establishing new connection to Atlas
Promise { <pending> }
Promise { <pending> }
Promise { <pending> }
```

4. Look at the Atlas console now:



Hmm, not what we were expecting. We're experiencing **a common problem** when using the **.map method asynchronously**. We would expect to see 3 console.logs indicating the document was or was not added, but we actually don't see any. The .map iterator in this case actually returned an Array of **unresolved** promises.

To get the code to behave like we think it should, we must incorporate our old friend **Promise.allSettled**. It allows us to wait for all the promises to be resolved before we can make use of the resulting Array.

5. Add another file called **mongo_exercise4.js** with the following:

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const addSomeUsers = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();
    let resultArray = await Promise.allSettled(
      // don't await this because we don't need any results immediately
      someUsers.map((user) => dbRtns.addOne(db, "users", user))
    );

    resultArray.forEach((result) => {
      result.value.acknowledged
        ? console.log(
            `Promise ${result.status} and document added to users collection`
          )
        : console.log(
            `Promise ${result.status} and document not added to users collection`
          );
    });
    let count = await dbRtns.count(db, "users");
    console.log(
      `there are currently ${count} documents in the user collection`
    );
    process.exit(0);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

addSomeUsers();
```

6. Execute the **mongo_exercise4.js** code now:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise4
establishing new connection to Atlas
Promise fulfilled and document added to users collection
Promise fulfilled and document added to users collection
Promise fulfilled and document added to users collection
there are currently 3 documents in the user collection
```

7. Refresh the Atlas console once more and you will see the expected results:

So, we've successfully transferred the array data up into the info3139db.users collection.

8. To delete the collection's contents before loading any data we need to add a new method to the **db_routines.js** files with the following code (do not forget to **update the export**):

```
const deleteAll = (db, coll) => db.collection(coll).deleteMany({});
```

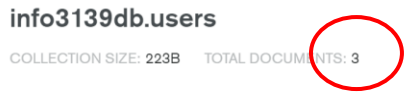
9. Add another file called week4\class1**mongo_exercise5.js** with the following (again, change the last name with yours):

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const reloadUsers = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();

    // clean out collection before adding new users
    let results = await dbRtns.deleteAll(db, "users");
    console.log(
      `deleted ${results.deletedCount} documents from users collection`
    );

    let resultArray = await Promise.allSettled(
      // don't await this because we don't need any results immediately
      someUsers.map((user) => dbRtns.addOne(db, "users", user))
    );

    resultArray.forEach((result) => {
      result.value.acknowledged
        ? console.log(
            `Promise ${result.status} and document added to users collection`
          )
        : console.log(
            `Promise ${result.status} and document not added to users collection`
          );
    });
    let count = await dbRtns.count(db, "users");
    console.log(
      `there are currently ${count} documents in the user collection`
    );
    process.exit(0);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

reloadUsers();
```

10. Then execute the mongoex5.js code and you should see:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise5
establishing new connection to Atlas
deleted 3 documents from users collection
Promise fulfilled and document added to users collection
Promise fulfilled and document added to users collection
Promise fulfilled and document added to users collection
there are currently 3 documents in the user collection
```

And refreshing the console results in:



11. Adding users one at a time is inefficient, let's do a **bulk insert** this time. Add the following method to db_routines and update the export:

```
const addMany = (db, coll, docs) => db.collection(coll).insertMany(docs);
```

12. Add another file called week4\class1**mongo_exercise6.js** with the following (change the name again):

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const bulkLoadUsers = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();
    // clean out collection before adding new users
    let results = await dbRtns.deleteAll(db, "users");
    console.log(
      `deleted ${results.deletedCount} documents from the users collection`
    );
    results = await dbRtns.addMany(db, "users", someUsers);
    console.log(
      `added ${results.insertedCount} documents to the user collection`
    );
    process.exit(0);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

bulkLoadUsers();
```

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise6
establishing new connection to Atlas
deleted 3 documents from the users collection
added 3 documents to the user collection
```

13. Now that we have added a single and an array of users, let's **retrieve one**. Add the following method to **db_routines.js** and update the exports array accordingly. Notice the **criteria** reference here provides an object to do the search on (e.g. my name in the example code in step 12).

```
const findOne = (db, coll, criteria) => db.collection(coll).findOne(criteria);
```

14. Add another file called week4\class1**mongo_exercise7.js** with the following (again, change my name to yours):

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const bulkLoadUsersAndFindOne = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();

    // clean out collection before adding new users
    let results = await dbRtns.deleteAll(db, "users");
    console.log(
      `deleted ${results.deletedCount} documents from users collection`
    );
    results = await dbRtns.addMany(db, "users", someUsers);
    console.log(
      `added ${results.insertedCount} documents to the user collection`
    );

    let someUser = await dbRtns.findOne(db, "users", { name: "Evan Lauersen" });

    console.log(
      `User ${someUser.name} was found. This user's email address is ${someUser.email}`
    );

    process.exit(0);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

bulkLoadUsersAndFindOne();

processDb();
```

15. Execute the mongo_exercise7.js code and you should see:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise7
establishing new connection to Atlas
deleted 3 documents from users collection
added 3 documents to the user collection
User Evan Lauersen was found. This user's email address is el@abc.com
```

16. The next method to add to **db_routines.js** will return **all documents** that meet a certain condition from a collection **(criteria**) and return only certain fields from the document (**projection**). If you do not add anything as criteria or projection you will be returned all documents and all fields in those documents:

```
const findAll = (db, coll, criteria, projection) =>
  db
    .collection(coll)
    .find(criteria)
    .project(projection)
    .toArray();
```

17. Add week4\class1**mongo_exercise8.js** with the following code to test this new method **and leave the criteria and project objects empty**:

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const bulkLoadAndFindUsers = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();

    // clean out collection before adding new users
    let results = await dbRtns.deleteAll(db, "users");
    console.log(
      `deleted ${results.deletedCount} documents from users collection`
    );
    results = await dbRtns.addMany(db, "users", someUsers);
    console.log(
      `added ${results.insertedCount} documents to the user collection`
    );
    let someUser = await dbRtns.findOne(db, "users", { name: "Evan Lauersen" });
    let allDbUsers = await dbRtns.findAll(db, "users", {}, {}); // empty criteria and projection

    allDbUsers.forEach((user) =>
      console.log(`user ${user.name} is in the collection`)
    );

    process.exit(0);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

bulkLoadAndFindUsers();
```

18. Then execute mongo_exercise8.js:

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise8
establishing new connection to Atlas
deleted 3 documents from users collection
added 3 documents to the user collection
user Jane Doe is in the collection
user John Smith is in the collection
user Evan Lauersen is in the collection
```

19. Add week4\class1**mongo_exercise9.js** with the following code to show an example of both **criteria** and **projection**. In this example we are going to return

all documents that have an email **that contain the letter j (criteria)** and **only return the email address field** (**projection**).

```
import * as dbRtns from "./db_routines.js";

const rawJSON = `[{"name":"Jane Doe", "age":22, "email": "jd@abc.com"},
                  {"name":"John Smith", "age":24, "email": "js@abc.com"},
                  {"name":"Evan Lauersen", "age":30, "email": "el@abc.com"} ]`;

const bulkLoadAndFindByCriteria = async () => {
  let someUsers = JSON.parse(rawJSON);

  try {
    const db = await dbRtns.getDBInstance();

    // clean out collection before adding new users
    let results = await dbRtns.deleteAll(db, "users");
    console.log(
      `deleted ${results.deletedCount} documents from users collection`
    );

    results = await dbRtns.addMany(db, "users", someUsers);
    console.log(
      `added ${results.insertedCount} documents to the user collection`
    );

    let allJEmails = await dbRtns.findAll(
      db,
      "users",
      { email: /j/ }, // only have addresses contain a j - criteria
      { email: 1 } // only return the email field - projection
    );
    console.log(
      `There are ${allJEmails.length} documents in users with a j in the email, they are:`
    );
    allJEmails.forEach((user) => console.log(`\t${user.email}`));
    process.exit(0);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

bulkLoadAndFindByCriteria();
```

Notice that the projection uses { email: 1} here the 1 means this is a field we want to return, so **the whole document is not returned** in this case, **just the email**.

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node mongo_exercise9
establishing new connection to Atlas
deleted 3 documents from users collection
added 3 documents to the user collection
There are 2 documents in users with a j in the email, they are:
        jd@abc.com
        js@abc.com
```

Lab 6 – 2.5%

Part A - (2%) - Using yargs, request an argument called **code** that represents a country code found in the iso countries JSON on github. The code is used to match the **alpha-2** property for each country:

https://raw.githubusercontent.com/elauersen/info3139/master/isocountries.json

Remember, to access a property with a hyphen you need to **use a different syntax**, you can't just use .alpha-2 but rather **["alpha-2"]** instead (for reference you can read over this StackOverflow article - https://stackoverflow.com/questions/13869627/unable-to-access-json-property-with-dash.)

Call an asynchronous routine in a file called **lab6.js** that will utilize the db_routines module we set up this class. Specifically do the following in this method:

1. Obtain the country iso data from the web at the GitHub location and place it in an array (refer to the code we used in week 3 class 1)
2. Delete any documents in a collection called **countries** on Atlas
3. Map through the new array from step 1 and create a series of new objects consisting **of only 2 properties** (name and code) whose data comes from the downloaded JSON
4. Bulk load the new array into a collection on Atlas called **countries**
5. When all the new objects have been added to the collection, search the Atlas collection for the code entered by the user and log the corresponding name (see format below)
6. Also log the total number of documents in the countries collection (see format below)

Your code **must utilize an .env file** (code your own corresponding **config.js**) file with the following constants (change the connection information to reflect your setup). The lab6 above method **should not** contain any hardcoded strings for database calls or string templates

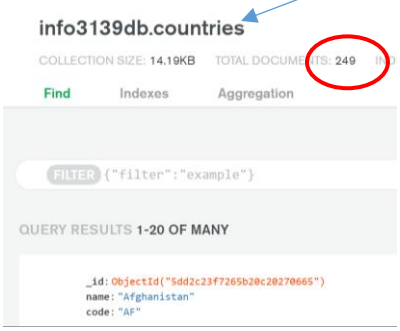
```
ISOCOUNTRIES=https://raw.githubusercontent.com/elauersen/info3139/master/isocountries.json
DBURL=mongodb+srv://info3139User:...@cluster0-qbwsa.mongodb.net/test?retryWrites=true&w=majority
DB=info3139db
COLLECTION=countries
```

Submit **in a single word document** the following:

1. A screen shot showing lab6 being **executed 3 times** using the codes:
    - CA, XX, AD

```
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node lab6 --code CA
establishing new connection to Atlas
there are currently 249 documents in the countries collection
deleted 249 documents from the countries collection
there are now 249 documents currently in the countries collection
The code CA belongs to the country of Canada
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node lab6 --code XX
establishing new connection to Atlas
there are currently 249 documents in the countries collection
deleted 249 documents from the countries collection
there are now 249 documents currently in the countries collection
The code XX is not a known country alpha-3 code
PS E:\winter2023\info3139\programming\nodeexercises\week4\class1> node lab6 --code AD
establishing new connection to Atlas
there are currently 249 documents in the countries collection
deleted 249 documents from the countries collection
there are now 249 documents currently in the countries collection
The code AD belongs to the country of Andorra
```

2. A screen shot showing the Atlas **countries** collection (remember the document should **only contain the _id, name and code properties**), showing the total documents and at least 1 full document in the screen shot:



3. The source code for lab6.js

Part B - (.5%) - Lab 6 Theory Quiz on FOL - 10 questions

Review

1. Do we need to have a resulting array declared in order to use the .map method?
2. What property of the returned variable from the dbRtns.deleteAll method did we check to ensure the delete worked?
3. What happens if you try to connect to the database and the database does not exist?
4. What collection method is called to lookup one document?
5. What export syntax is used in a module that has more than one method?
6. What does the .map iterator return when calling promises?
7. What mongo collection method is called to delete all documents in the collection?
8. What mongo collection method is called to create a single document?
9. What mongo collection method is called to create multiple documents from an array?
10. Using the Mongo Atlas dashboard how can I delete an existing collection or db?
11. Describe the JSON returned from the country ISO JSON on GitHub?