CS-546 Lab 4

Our First ToDo

For this lab, we're going to embark on a very common path that web developers start off on new technologies with: creating a to-do list! You will create a to-do list data module for this lab lab, and test it yourself.

The major concepts of this lab are:

- Seperating concerns into different modules:
  - Database connection in one module
  - Collections defined in another
  - Data manipulation in another
- Further practicing the usage of **async / await** for asynchronous code
- Continuing our exercises of linking these modules together as needed

Packages you will use:

You will use the **uuid** package in order to generate unique id's to use as your identifiers. You can read up on uuid (Links to an external site.)Links to an external site. on the Github project page. You may use either v4 or v1 ids.
You will also use the mongodb (Links to an external site.)Links to an external site. package.
You **may** use the lecture 4 code (Links to an external site.)Links to an external site. as a guide.

You must save all dependencies you use to your package.json file

Database Structure

You will use a database with the following organization:

- The database will be called **FirstName_LastName_lab4**
- The collection for todo items will be called **todoItems**

todo.js

In todo.js, you will create and export four methods:

async createTask(title, description);

This async function will resolve to the newly created to-do list object, with the following properties.

```
{
  _id: "a unique identifier for the task; you will generate these using uuid package",
  title: "the title of the task",
  description: "a descriptive bio of the task",
  completed: false,
  completedAt: null
```

```
}
```

This task will be stored in the **todoItems** collection.

Important Note: **you will create and set the `_id` field in the `createTask` method before you insert the document**.

Important Note: as you can tell, the parameters only provide a title and description. You must still set the other fields before inserting them into the database.

If the task cannot be created, the method should reject.

You would use it as:

```
const todoItems = require("./todo");

async function main() {
  const createdTask = await todoItems.createTask("My First Task", "This is the first thing I need to do today");
  console.log(createdTask);
}

main();
```

async getAllTasks();

This function will resolve to an array of all tasks in the database.

```
const todoItems = require("./todo");

async function main() {
  const getTasks = await todoItems.getAllTasks();
  console.log(getTasks);
}

main();
```

async getTask(id);

When given an id, this function will resolve to a task from the database.

If no id is provided, the method should reject.

If the task does not exist, the method should reject.

For example, you would use this method as:

```
const todoItems = require("./todo");
```

```
async function main() {
  const task = await todoItems.getTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");
  console.log(task);
}

main();
```

async completeTask(taskId)

This function will modify the task in the database. It will set `completed` to `true` and `completedAt` to the current time (Links to an external site.)Links to an external site..

If no id is provided, the method should reject.

If the task cannot be updated (does not exist, etc), this method should reject.

If the update is successful, this method will resolve to the updated task.

```
const todoItems = require("./todo");

async function main() {
  const task = await todoItems.getTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");
  const finishedTask = await todoItems.completeTask(task._id);
  console.log(finishedTask);
}

main();
```

Important note: for now, in `completeTask` you will want to get the task from the database, update the task in your JS code, and then run the update command.
If you would like to do something more advanced, you may also research using the $set (Links to an external site.)Links to an external site. command to accomplish this as well.

async removeTask(id)

This function will remove the task from the database.

If no id is provided, the method should reject.

If the task cannot be removed (does not exist), the method should reject.

If the removal succeeds, resolve to true.

```
const todoItems = require("./todo");

async function main() {
  const removeTask = await todoItems.removeTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");
```

```
  try {
    return await todoItems.getTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");
  } catch (error) {
   console.error(error);
  }
}

main();
```

app.js

For your app.js file, you will:

1. Create a task with the following details:

```
{
  title: "Ponder Dinosaurs",
  description: "Has Anyone Really Been Far Even as Decided to Use Even Go Want to do Look More Like?"
}
```

2. Log the task, and then create a new task with the following details:

```
{
  title: "Play Pokemon with Twitch TV",
  description: "Should we revive Helix?"
}
```

3. After the task is inserted, query all tasks and log them

4. After all the tasks are logged, remove the first task

5. Query all the remaining tasks and log them

6. Complete the remaining task

7. Log the task that has been completed with its new value.

General Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder
3. You must do basic error checking in each function

1. Check for arguments existing and of proper type.
2. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
3. If a function should return a promise, you should mark the method as an `async` function and return the value. Any promises you use inside of that, you should *await* to get their result values. If the promise should reject, ten you should throw inside of that promise in order to return a rejected promise automatically. Thrown exceptions will bubble up from any awaited call that throws as well, unless they are caught in the async method.
4. You **must remember** to update your package.json file to set `app.js` as your starting script!
5. You **must** submit a zip file named in the following format: `LastName_FirstName_CS546_SECTION.zip`