Lab 3

Re-submit Assignment

Due Oct 14 by 5pm **Points** 100 **Submitting** a file upload

CS-546 Lab 3

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 promises for asynchronous code
- Continuing our exercises of only linking these modules together as needed

Packages you will use:

You will use the **node-uuid** package in order to generate unique id's to use as your identifiers.

You can read up on **node-uuid** (https://github.com/broofa/node-uuid) on the Github project page.

You will also use the **mongodb** (http://mongodb.github.io/node-mongodb-native/2.1/) package.

You may use the **lecture 4 code** (https://github.com/Stevens-CS546/CS-546-WS-Summer-1/tree/master/Lecture%20Code/lecture_04) as a guide.

You must save all dependencies to your package.json file

Database Structure

You will use a database with the following organization:

- The database will be called **lab3**
- The collection for todo items will be called **todoItems**

todo.js

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

createTask(title, description);

This function will return a promise that resolves to a newly created to-do list object, with the following properties:

```
{
    _id: "a unique identifier for the task; you will generate these using node-uuid",
    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**.

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");

let createdTask = todoItems.createTask("My First Task", "This is the first thing I need to do today");

createdTask.then((newTask) => {
    console.log(newTask);
});
```

getAllTasks();

This function will return a promise that resolves to an array of all tasks in the database.

getTask(id);

This function will return a promise that resolves to a task from the database, when given an id. For example, you would use this method as:

If no id is provided, the method should return a rejected promise.

If the task does not exist, the method should return a rejected promise.

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

let taskPromise = todoItems.getTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");

taskPromise.then((task) => {
    console.log(task);
})
```

completeTask(taskId)

This function will modify the task in the database. It will set `completed` to `true` and `completedAt` to the current time.

If no id is provided, the method should return a rejected promise.

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

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

let taskPromise = todoItems.getTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");

let finishedTask = taskPromise.then((task) => {
    return todoItems.completeTask(task._id);
});

finishedTask.then((task) => {
    console.log(task);
});
```

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 (https://docs.mongodb.com/manual/reference/operator/update/set/)** command to accomplish this as well.

removeTask(id)

This function will remove the task from the database.

If no id is provided, the method should return a rejected promise.

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

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

let removeTask = todoItems.removeTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");

let tryToGetTask = removeTask.then(() => {
    return todoItems.getTask("9714a17c-f228-49e9-a772-9086f5ff8bfb");
});

tryToGetTask.catch((error) => {
    // Should error out
    console.error(error);
})
```

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 tasks and log them

6. Complete the remaining task

7. Query and log the remaining task.

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, instead of throwing you should return a rejected promise (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/reject) .**
4. You **must remember** to update your package.json file to set `app.js` as your starting script!
5. You **must** submit a zip, rar, tar.gz, or .7z archive or you will lose points, named in the followign format: `LastName_FirstName_CS546_SECTION.zip` (or, whatever the file extension may be). You will lose points for not submitting an archive.