Lab 4
Due 3/21 before class

Part 1
Preload your Mongo database by calling one of your external APIs from Lab 3 one hundred times and loading those documents into a collection. You may modify the documents in any way you need to make the rest of this lab easier. All the documents should look the same. You can use Compass for this.

Part 2: Creating a database API
Create a new endpoint for your Node API: name it **/db**
Create a second new endpoint: a number that begins at 1 and increases from there so that endpoints like /db/50 and /db/75 are legal. Each **/db/:number** endpoint is tied directly to a specific document in the collection (you might consider adding a key:value pair to each document, where the value matches the :number, to make this easier for you).

The /db endpoint and the /db/:number endpoints should listen on all four HTTP verbs: GET, POST, PUT, and DELETE.

The GET request logic on /db/:number should fetch one specific document from the collection, whichever document corresponds to that number. If the GET request is on /db, you should get a listing of all valid document numbers in the collection (do not simply print every document!). This might *sound* as easy as remembering how many documents you have but it is not–what do you need to do if you have 100 documents and someone deletes document #42?

The POST request logic should allow you to add a new document to the collection. The document to add will be in the body of the POST request. This logic must be done on the /db endpoint only. A POST request to a /db/:number endpoint should result in an error (and not create a new document). You **cannot** reuse document numbers once assigned. This means if document #42 is deleted, the next POST request cannot use 42 as this new document's number.

The PUT logic on /db/:number should allow you to update an existing document. The desired changes to the document should be specified in the body of the PUT request. A PUT request on a document that does not exist should return an error. A PUT request on the /db endpoint should bulk update **all documents in the collection**.

The DELETE logic on /db/:number should allow you to delete an existing document. You don't need a body for this logic. A DELETE request on a /db/:number that does not exist **does not** return an error (it returns OK). A DELETE request on /db should mass delete **all documents in the collection**.

Part 3: A new React component

Make a new React component that presents an input box for :number (or 0 or empty for the /db endpoint), a large input field to be used for the body for POST and PUT requests, and four buttons: one for each HTTP verb. This component will talk to the API implemented in part 2. Display sensible output for successes and errors for all verb and endpoint combinations.

Part 4: .env file
You very much **do not** want to push your code to GitHub with your MongoDB password in it! You will want to create a .env file (and make sure .env is in your .gitignore file!) that looks something like this:

```
# .env file
MONGODB=<your mongodb connection string>
```

You can then start Node with:

```
node --env-file=.env server.js
```

And you will then have access to a process.env.MONGODB variable, whose contents are the mongodb connection string.

By the way, your mongodb connection string will look something like mongodb://<that url from atlas>/lab5 (assuming you named your database lab5).

README.md file
README.md files are not optional. Put in it all your citations, a running work log, where you got stuck, how you got unstuck, and anything you'd like us to consider for creativity. **You get a 0 for the lab if you don't have a README.md file and/or you don't cite your sources in your README.md file!**

Grading

Part 2	20 points
Part 3	10 points
Creativity	10 points
README.md	10 points
Total	**50 points**

If you are using Mongoose, first run `npm i mongoose` then in server.js:

```
const mongoose = require('mongoose');
await mongoose.connect('<your mongodb connection string>');
```

You then do need to define a schema with mongoose. You'll need to look at your JSON objects and see what's inside them. But for our simple course API from earlier in this semester, it would look something like:

```
const courseSchema = new mongoose.Schema({
  number: String,
  description: String
});
```

Now we compile that into a model:

```
const Course = mongoose.model('Course', courseSchema);
```

For reasons I’m not sure about, you use the *singular* version of your collection name as the first argument to mongoose.model but it is actually referencing a collection with the *plural* name: so in the line above, it is actually going to be using a collection named **Courses**. Be careful!

We can fetch documents with something like:

```
await Course.find({ number: 'ITWS 4500' });
```

And add new documents with something like:

```
const infosec = new Course({ name: 'ITWS 4370', description: 
'Information Systems Security' });
await infosec.save();
```