Assigned: 27 September

Homework #4 – NoSQL League API

EE 547: Fall 2022

Due: Sunday, 09 October at 23:59. Late penalty: 10% per 24-hours before 11 October at 23:59. Submission instructions will follow separately on canvas.

1. Extend your HW #3 Node.js web application to use a Mongo database to persist changes.

A. Database Connection

Read Mongo database connection details from a JSON file. Use the file path `./config/mongo.json` relative to your index script. Assume the following default values if the file does not exist or in place of any empty fields:

```
{
  host: "localhost",
  port: "27017",
  db:   "ee547_hw",
  opts: {
    useUnifiedTopology: true
  }
}
```

You may assume the database has no access-control (*i.e.,* no user/pass). Exit with code 2 if the file is invalid JSON. Exit with code 5 if the file is valid but MongoDB connection fails for any reason.

B. Schema

Use a collection called `player` to store Player documents. Replace the integer `pid` in HW #2 with a Mongo `ObjectId` field called `_id`. Use `_id.toString()` as the player unique id.

Use the same document schema as the JSON database file in HW #2:

```
{
  _id:            ObjectId  player id
  fname:          string
  lname:          string
  handed:         string    L|R|A
  is_active:      boolean
  balance_usd:    string    currency string
}
```

Assume an empty league if the collection does not exist. Your server should create the collection on-demand. Immediately write any player data change to the database (create, delete, update,

etc.). Fetch data for every request using the most recent database state and do not use any caching facility.

C. Response Syntax

Use JSON for all (non-empty) responses. Use the following syntax for entity responses.

`Player[pid]`

```
{
  pid:            string    player id
  name:           string    "fname lname" - no trailing spaces
  handed:         string    left|right|ambi
  is_active       boolean
  balance_usd     string    currency string
}
```

D. Mongo Document "Base" Schema

You may extend the following base document schema. You may add fields or collections to maintain additional state within the same database but DO NOT MODIFY ANY ATTRIBUTES BELOW. Ensure your script accepts any document that satisfies the base schema as a valid document and infer reasonable defaults. Use a collection called `player` to store Player documents.

`player`

```
{
  _id:            ObjectId  player id
  balance_usd     string    currency string
  created_at      Date
  fname           string
  lname         string
  handed:         string    L|R|A
  is_active       boolean
}
```

2. **Node.js Practice: File access, timeouts, and intervals**

`setTimeout(cb, delay)` and `setInterval(cb, delay)` are Node.JS functions that are used to create programmatic timeouts for long-running processes and to avoid *zombie* operations. `fs.stat(file, cb)` and `fs.readFile(file, cb)` are two asynchronous filesystem functions to report information for a given filesystem path.

Implement the following function.

```
exports.fileCat = function(file1, file2, callback) {
  this.SEPARATOR  = ' ';   // space
  this.TIMEOUT_MS = 2000;  // 2.0 sec

  // complete me

}
```

- file1 <string>
- file2 <string>
- callback <function>
  - err <Error>
  - data <string>

Asynchronous.

Concatenate string content from file1 and file2 using SEPARATOR = ' '. If file1 or file2 do not exist the function periodically checks for its existence every 100ms. The callback gets two arguments err and data where data is the concatenated string. On failure the function should callback with Error object having message string: 'file1 not exist' | 'file2 not exist' | 'file1 and file2 not exist'.

The function does not perform any file read operations unless and until both files exist. The function ceases to monitor the existence of a file once it exists. The function terminates after a TIMEOUT_MS = 2000 millisecond timeout and generates an error.