Y STEM and Chess

Database Documentation

Samson Zhang | sz7651@rit.edu

September 19, 2023

1 Overview

Disclaimer: I did not design/implement the database system. The information stated in this document represents only the best of my knowledge of the system, which might be incomplete or incorrect at certain places.

1. We are using MongoDB, and accessing it in our code using mongoose.js
2. The relevant files can be found in the middlewareNode folder

1.1 Web App, Middleware, Database

With the help of Mengqi Wu, I was able to test and find that the database is actually hosted remotely. This means that any changes you make to the database will reflect to other people as well.

I previously guessed that changes are local because we local host the middlewareNode. But we now know that that's not the case.

My current guess is that we local host the middlewareNode so we can run database commands through the routes defined in the middleware:

$$\text{web app} \xrightarrow[\text{Request}]{\text{HTTP}} \text{middleware} \xrightarrow[\text{Command}]{\text{Run}} \text{database}$$

$$\text{web app} \xleftarrow[\text{Response}]{\text{HTTP}} \text{middleware} \xleftarrow{\text{Result}} \text{database}$$

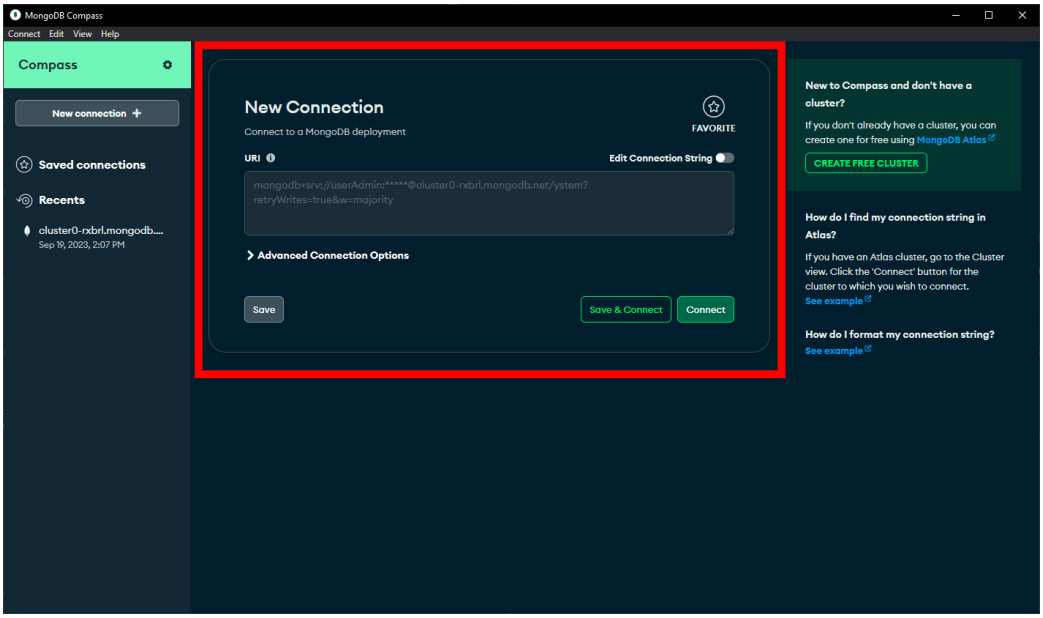
2 Direct Connection

Database connection URL:

```
mongodb+srv://userAdmin:l2W50UIqrscqWcXM@cluster0-rxbrl
.mongodb.net/ystem?retryWrites=true&w=majority
```

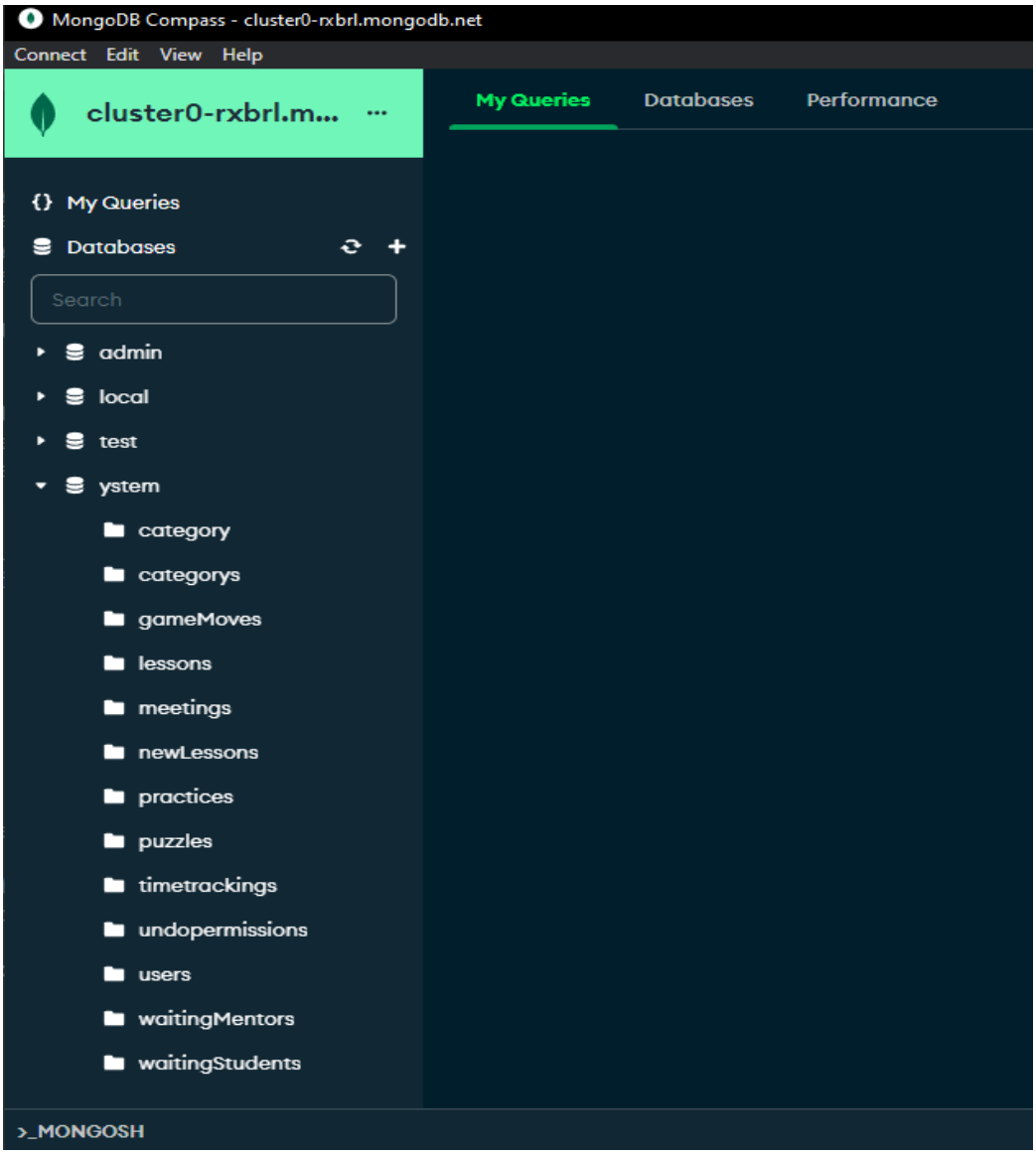
To directly connect to the database, use the connection url as provided above.

You should be able to use any tools you prefer to connect to the database. We will use MongoDB Compass in this document:

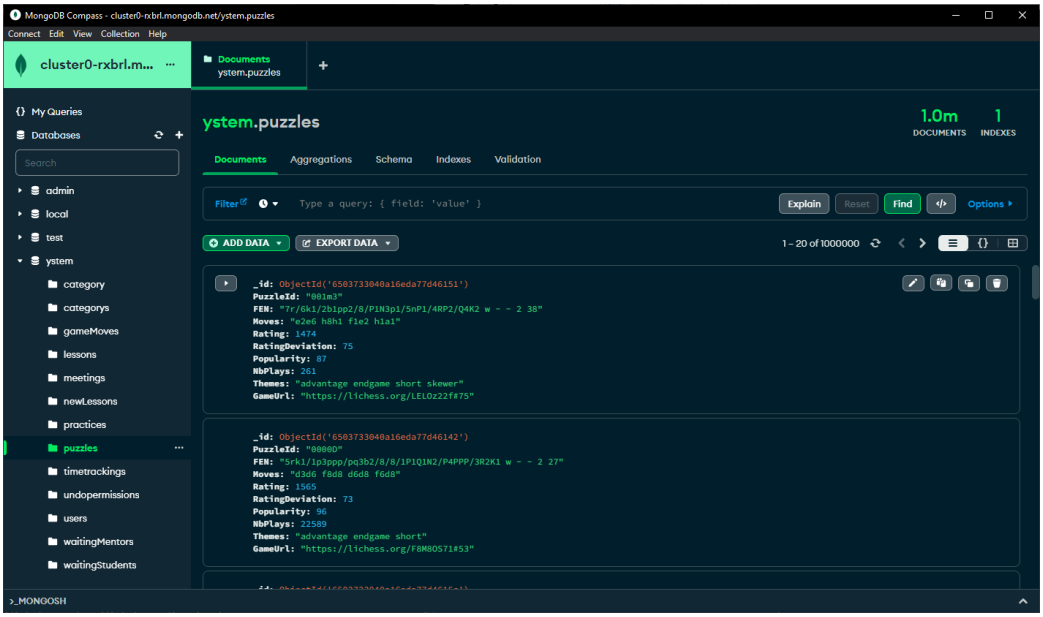


3 MongoDB Compass

Once connected in MongoDB Compass, you should be able to see a list of databases on the left of the app, they each contain some collections. We mainly care about the ystem database, which contains relevant collections to for development:



Clicking on a collection will open it, and you will be able to see the data stored in the collection, you can add and export data pretty easily as well:



You can add a collection into a database by clicking the plus sign next to the database:

4 Adding New Collection

Normally, adding a collection can be easily done through MongoDB Compass (or whatever tool you decide to use). However, in order to use it in our code, we will need to do a few more things.

The middlewareNode folder contains the following (blue means folder):

- config
- controllers
- models
- node_modules
- routes
- template
- utils
- dockerfile
- package-lock.json
- package.json
- server.js

The folders/files that we are interested in are the following:

- models
- routes
- server.js

4.1 models

The modles folder stores the schema of each collection that we are using. For example, here is content of puzzles.js in the model folder:

```
const mongoose = require("mongoose");
const { Schema, model } = mongoose;

// Schema with the attributes matching data provided by the
// lichess puzzles API
const puzzleSchema = new mongoose.Schema(
  {
    puzzleId: {
        type: String,
        required: true,
    },
    FEN: {
        type: String,
        required: true,
    },
```

```
    moves: {
        type: String,
        required: true
    },
    rating: {
        type: Number,
    },
    ratingDeviation: {
        type: Number,
    },
    popularity: {
        type: Number,
    },
    nbPlays: {
        type: Number,
    },
    themes: {
        type: String,
    },
    gameUrl: {
        type: String,
    },
    openingTags: {
        type: String,
    },
  },
  { versionKey: false }
);

module.exports = puzzles = model("puzzles", puzzleSchema);
```

4.2 server.js

After specifying the schema of the collection, we still need to add it to our routes. This is done so that we can access the collection at the specified route in our code.

To do so, we need to add the route to server.js, which currently looks like this (notice the section commented with "Define Routes"):

```
const express = require('express')
const connectDB = require('./config/db')
const passport = require('passport')
require('./config/passport.js')
const app = express()
const cors = require('cors')
const config = require('config')

//Enable Cors
app.use(cors(config.get('corsOptions')))

// Connect Database
connectDB()

// Init Middleware
app.use(express.json({ extended: false }))

app.get('/', (req, res) => res.send('API Running'))

// Define Routes
app.use(passport.initialize())
app.use(passport.session())
app.use('/user', require('./routes/users'))
app.use('/category', require('./routes/categorys'))
app.use('/meetings', require('./routes/meetings'))
app.use('/auth', require('./routes/auth'))
app.use('/timeTracking', require('./routes/timeTracking'))
app.use('/puzzles', require('./routes/puzzles'))
const PORT = process.env.PORT || 8000

app.listen(PORT, () => console.log(`Server started on port ${PORT}`))
```

4.3 routes

Now, for any specific features requiring us to access the database in our code, we will need to define routes. For example, the following is the content of puzzles.js in the routes folder:

```
const express = require('express')
const router = express.Router()
const puzzles = require('../models/puzzles')

// Get all puzzles
router.get('/list', async (req, res) => {
  try {
      const puzzlesArray = await puzzles.find({}, {_id: 0 });
      res.status(200).json(puzzlesArray);
  } catch (error) {
    console.error(error.message)
    res.status(500).json('Server error')
  }
})

module.exports = router
```

There is only one route defined, which is /list. Factor in the localhost, port number, and the fact that this is a route from puzzles, the full route is then localhost:8000/puzzles/list.

When you access this link, you should see the result returned by the puzzles.find({}, {_id: 0}) command, which gets all data entries from the puzzles collection.

To to run any specific database commands on a collection, you start by defining a new route in the corresponding collection js file. You can reference the code in the previously implemented file in the routes folder.

5 Accessing Database At Runtime

With the collections and the routes set up, we can now access the database at runtime with code. We will continue the puzzles/list example from the previous section here.

Let's say you have:

1. Connected to the database and added puzzles data
2. Written the puzzles collection schema puzzles.js in the models folder
3. Added the puzzles route to server.js
4. Defined the /list route in puzzles.js in the routes folder

How can we run the command and get all of the puzzle in the puzzle page backend? With a HTTP request.

Referencing the code from play-nolog.component.ts, we can use the following function to send HTTP requests:

```
private httpGetAsync(theUrl: string, method: string = 'POST', callback) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function () {
      if (xmlHttp.readyState == 4 && xmlHttp.status == 200)
        callback(xmlHttp.responseText);
    };
    xmlHttp.open(method, theUrl, true); // true for asynchronous
    xmlHttp.send(null);
}
```

Below is the example of using it to get all data from the puzzles collection:

```
this.httpGetAsync(
  `${environment.urls.middlewareURL}/puzzles/list`,
  'GET',
  (response) => {
    var response = JSON.parse(response);
    var jsonData = []
    for (var i = 0; i < response.length; i++) {
      jsonData.push(response[i]);
    }
    this.ps.puzzleArray = jsonData;
    this.shuffleArray(this.ps.puzzleArray);
  }
)
```

The code snippet above sends a HTTP GET request to the specified route (localhost:8000/puzzles/list). With the response, it parses it and updates the puzzleArray.

The general structure for the function parameter is:

```
this.httpGetAsync(
    'route',
    'request type',
    (response) => {
        'code to execute'
    }
)
```

For examples of HTTP POST requests, you can refer to play-nolog.components.ts.