Todd Gavin
DSCI550
Professor Wengsheng Wu
28 April 2023

Emulating Firebase Realtime Database

1. Purpose

The purpose of this project was to emulate a large data storage system such as Firebase's Realtime Database using alternate systems such as MongoDB's NoSQL database and Python Flask for the RESTful server.

Link to GitHub Repository: https://github.com/todd-gavin/DSCI551-FirebaseEmulator
Link to Google Drive Submission: https://drive.google.com/drive/folders/1sV78mQyJaovyRtlS4UVRnskFi-6bI1GI?usp=sharing
Link to YouTube project DEMO video: https://youtu.be/S5IYBkvT2Zs

2. Implementation and Architecture

flaskServer.py

a. This Python code implements a RESTful API using the Flask framework that interacts with a MongoDB database to perform CRUD (Create, Read, Update, and Delete) operations. The architecture of the code can be described as follows:
    i. Import necessary libraries and functions:
        1. Import the necessary functions from the custom `mongoDB_driver` module, which manages the connection and interaction with the MongoDB database.
        2. Import the `Flask` class and the `request` object from the `flask` module to create the web application and handle incoming requests.
        3. Import the `json` module for processing JSON data.
    ii. Initialize the Flask application and create a MongoDB client:
        1. Instantiate a Flask app object.
        2. Connect to the MongoDB server using the `connectMongoDB` function and store the client object.
        3. Set up the database, collection, and document filter for reading/writing data.
    iii. Pre-process incoming requests:
        1. Use the `before_request` decorator to ensure the content-type of incoming POST, PATCH, and PUT requests is set to ``'application/json'`.
    iv. Define the request handler function:

1. Create a single function, `handle_request`, to handle all incoming requests for the RESTful API.
2. Use the `route` decorator to define the default and custom routes, and specify the allowed HTTP methods (GET, POST, PUT, PATCH, DELETE).
3. Inside the function, parse the path parameters, extract the JSON path, and retrieve the data from the request,
4. Based on the HTTP method, execute the respective CRUD function:
    a. GET: Retrieve data from the MongoDB database.
    b. POST: Create new data in the MongoDB database.
    c. PUT: Update the entire data object in the MongoDB database.
    d. PATCH: Update specific fields of the data object in the MongoDB database.
    e. DELETE: Remove data from the MongoDB database.

v. Run the Flask application:
    1. Use the `app.run()` function to start the Flask web server, specifying the desired port number (5000).

mongoDB_driver.py

b. This code provides an interface for connecting to a MongoDB server, as well as performing CRUD (Create, Read, Update, and Delete) operations on the data stored in the database. The code uses the `pymongo` library to interact with MongoDB and `certifi` library for secure SSL/TLS certificate handling. The architecture of this code is as follows:
    i. Connection to MongoDB:
        1. The `connectMongoDB` function is responsible for connecting to a MongoDB instance using a URI that includes the username, password, and other necessary information for authentication. The `certifi.where()` method is used to obtain the CA certificate needed for SSL/TLS communication. The `ping` command is used to check if the connection was successful.
    ii. Accessing a specific database, collection, and document:
        1. The `db_collection_document` function takes the `client`, `db_name`, `collection_name`, and `objectId` as inputs and returns references to the database, collection, and a filter for the specified document.
    iii. Custom sorting:
        1. The `custom_sort_key` function sorts the data according to the following order: numbers, letters, special characters, and data types (like dicts, lists, and tuples).
    iv. Filtering data:
        1. The `helper_filter` function applies filtering operations based on the provided filter parameters. The function handles various cases like ordering by keys or values, equalTo, limitToFirst, limitToLast, startAt, and endAt, among others.

v. CRUD operations: The code contains functions to perform CRUD operations on the MongoDB data:
    1. `get`: Retrieves data from the collection based on the document filter, JSON path, and additional filter parameters. It uses the `helper_filter` function to apply the required filters.
    2. `put`: Updates the data in the MongoDB collection by replacing the specified JSON path with the new data. The `helper_Update_PUT` function is used to create the update command.
    3. `post`: Adds new data to the MongoDB collection, creating a new unique key if necessary. The `helper_Update_POST` function is used to create the update command.
    4. `patch`: Updates the data at the specified JSON path, either overwriting the existing data or creating new data if it doesn't exist.
    5. delete`: Removes data at the specified JSON path from the MongoDB collection.

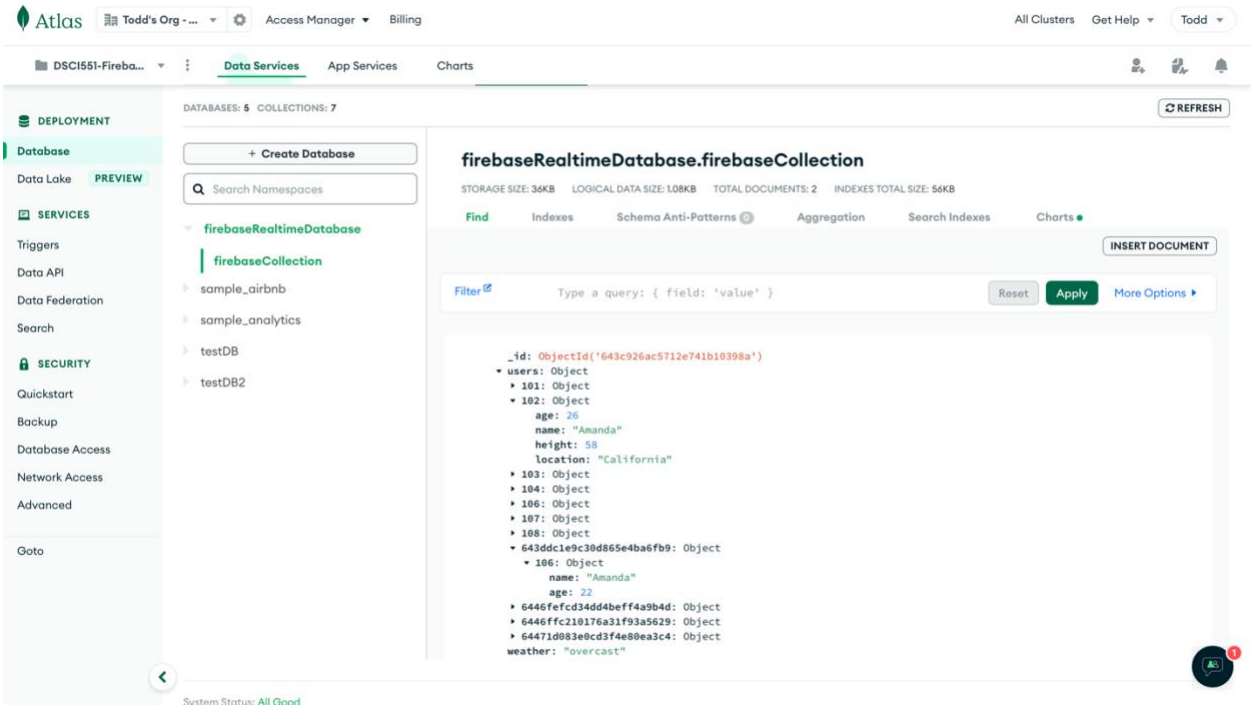
This code effectively implements a RESTful API to communicate with a MongoDB database using the Flask web framework. The API enables users to perform various CRUD operations using CURL on the data stored in the MongoDB database, allowing for flexible and efficient data management.

3. Learning Experiences

a. Data validation and error handling:
    i. This project demonstrates the importance of validating input data and handling errors. For example, checking if `limitToFirst` and `limitToLast` are integers and ensuring that only one of them is specified. Proper error handling not only improves the code's reliability but also makes it easier to debug and maintain.
b. Code modularity and organization:
    i. The project is divided into different functions that handle specific tasks such as connecting to the database, filtering data, and performing CRUD operations. This modular design enhances code readability, maintainability, and makes it easier to extend the codebase in the future.
c. Custom sorting:
    i. Implementing a custom sort function (`custom_sort_key()`) allows the code to sort data by numbers, letters, special characters, and data types. This is an excellent example of how to create custom sorting logic tailored to specific project requirements.
d. Working with MongoDB:
    i. The project offers an opportunity to learn how to work with MongoDB, a popular NoSQL database. This includes connecting to a MongoDB instance using pymongo, defining and working with databases and collections, and performing CRUD operations.

e. RESTful API design:
    i. The project demonstrates a simple RESTful API design using GET, PUT, POST, PATCH, and DELETE methods. Learning how to design and implement RESTful APIs is crucial for building modern web applications and microservices.
f. Filtering data:
    i. The project shows how to implement filtering logic, such as filtering based on `equalTo`, `startAt`, `endAt`, `limitToFirst`, and `limitToLast` parameters. This is a valuable learning experience as it shows how to parse and apply filtering logic on the server-side, which is a common requirement in many applications.
g. Managing nested data structures:
    i. The project handles nested data structures (dictionaries and lists) and demonstrates how to update and manipulate them using helper functions like `helper_Update_PUT()` and `helper_Update_POST()`.
h. Code commenting and logging:
    i. The project includes numerous comments and log statements that explain what the code is doing and provides insights into the flow of execution. This can be a useful learning experience for understanding the importance of properly documenting and logging code for future maintainability and collaboration.

4. MongoDB Document JSON Structure



For the structure of the MongoDB Database, to most accurately the “single document” structure of a Firebase Realtime Database, all data is store within a single document, in a single collection, in a single database, within one cluster on MongoDB. Therefore, the data is easily filterable and expandable. Due to the single document structure, data can be index on any path more easily than splitting up the data into multiple documents.

5. DEMO CURL Commands with Output

```
$ curl -X GET 'http://127.0.0.1:5000/.json?print=pretty'

{
   "users": {
      "101": {
         "age": 25,
         "gender": "Male",
         "name": "John"
      },
      "102": {
         "age": 26,
         "name": "Amanda",
         "height": 58,
         "location": "California"
      },
      "103": {
         "age": 12,
         "gender": "F",
         "name": "mary"
      },
      "104": {
         "age": 38,
         "name": "david smith sr"
      },
      "643ddc1e9c30d865e4ba6fb9": {
         "106": {
            "name": "Amanda",
            "age": 22
         }
      },
      "107": {
         "name": "john",
         "age": 32
      },
      "108": {
         "name": "john",
```

```
        "age": 32
      },
      "6446fefcd34dd4beff4a9b4d": {
        "108": {
          "name": "Johnny",
          "age": 33
        }
      },
      "106": {
        "age": 42,
        "name": "Grace"
      },
      "6446ffc210176a31f93a5629": {
        "weatherType": "Sunny"
      },
      "64471d083e0cd3f4e80ea3c4": {
        "109": {
          "name": "Miles",
          "age": 36
        }
      }
    },
    "weather": "overcast"
}

$ curl -X GET 'http://127.0.0.1:5000/users.json?orderBy="$key"&print=pretty'

[
    {
      "101": {
        "age": 25,
        "gender": "Male",
        "name": "John"
      }
    },
    {
      "102": {
        "age": 26,
        "name": "Amanda",
        "height": 58,
        "location": "California"
      }
    },
    {
```

```
    "103": {
      "age": 12,
      "gender": "F",
      "name": "mary"
    }
  },
  {
    "104": {
      "age": 38,
      "name": "david smith sr"
    }
  },
  {
    "106": {
      "age": 42,
      "name": "Grace"
    }
  },
  {
    "107": {
      "name": "john",
      "age": 32
    }
  },
  {
    "108": {
      "name": "john",
      "age": 32
    }
  },
  {
    "643ddc1e9c30d865e4ba6fb9": {
      "106": {
        "name": "Amanda",
        "age": 22
      }
    }
  },
  {
    "6446fefcd34dd4beff4a9b4d": {
      "108": {
        "name": "Johnny",
        "age": 33
      }
```

```
    }
  },
  {
    "6446ffc210176a31f93a5629": {
      "weatherType": "Sunny"
    }
  },
  {
    "64471d083e0cd3f4e80ea3c4": {
      "109": {
        "name": "Miles",
        "age": 36
      }
    }
  }
]
```


```
$ curl -X GET 'http://127.0.0.1:5000/users.json?orderBy="$key"&limitToFirst=2&limitToLast=3'

{'error': 'Only one of limitToFirst and limitToLast may be specified'}
```

Note: Top see more CURL commands, please refer to the project video.