User Guide for Go-DB

Simple guide for using the go-db application. Go-db mimics a simple document database using golang, and allows interaction through a RESTful API. This document will go over how to interact with the RESTful API, and also give a high level overview of the application itself.

Overview

The golang database follows a simple document database like mongodb. It is logically separated into this hierarchy of logical abstractions:

Database => Collection => Document

Database being the highest level of abstraction. It simply contains a name, and then a list of collections inside of it. The collection is similar to a database as in the collection is simply its name, and then a list of documents inside. Document is the final and most specific layer of abstraction. It contains an object id which is a unique identifier in the form of a UUID, and then a data field the stores the document given by the user.

The golang application has a list of RESTful APIs that allow the user to Create, Read, Update, and Destroy resources inside of the session. They will be described in greater detail later in this document.

Objects

This will go over the models used in the application that are used to store data, and send/receive data via the api.

- Database
    - Descrption: database struct used to represent a database object inside of the golang api.
    - Body:
        - Name
            - Type: string
            - Description: the name of the database
        - Collections
            - Type: array of structs
            - Description: the array of collections this database will hold
- Collection
    - Description: the collections that will contain all of the document objects
    - Body:
        - Name
            - Type: string
            - Description: the name of the collection
        - Documents
            - Type: array of structs
            - Descriptions: the array of structs that will contain its structs
- Document
    - Description: The document that will actually store the data from the user
    - Body:
        - ObjectID
            - Type: UUID
            - Description: unique identifier for document
        - Data
            - Type: raw json message
            - Description: stores the json of whatever the post operation was.
- Search
    - Description: allows user to search documents by keyword
        - Keyword (json field is search)
            - Type: string
            - Description: keyword you want to search for

API

This will go over all the routes in the API. You could also import the postman collections from the docs/testing folder in the repository's root.

- /db
    - Method: POST
        - Description: posts and creates a new database to the session
        - Request body:
            - Name
                - Type: string
                - Description: the name of the data base you want to create.
    - Method: GET
        - Description: gets all of the current databases inside of the session
            - No request body
- /db/{dbName}
    - URL parameters
        - dbName
            - Type: string
            - Description: name of the database you want to update, get, or delete
    - Method: GET
        - Description: get specific database from session by name
        - Request Body:
            - Name
                - Type: string
                - Description name of the database to get
    - Method: PUT
        - Description: update an existing database in the session. The name in the path is the current name of the database, and the name in the request body
        - Request Body:
            - Name
                - Type: string
                - Description: the new name you want to rename the database to.

    - Method: DELETE
        - Description: deletes the database by dbName in the path
        - No request body
- /db/{dbName}/collection
    - URL Parameters
        - dbName
            - Type: string
            - Description: the database name that you want to add or get collections from
    - Method: POST
        - Description: create a new collection on the database specified in the URL
        - Request Body:
            - Name
                - Type: string
                - Description: the name of the collection you want to create
    - Method: GET
        - Description: gets all of the collections listed in the specified database.
        - No request body
- /db/{dbName}/collection/{collectionName}
    - URL Parameters
        - dbName
            - Type: string
            - Description: the database name that you want to modify
        - collectionName
            - Type: string
            - Description: the collection name that you want to read, update, or destroy
    - Method: GET
        - Description: Gets a collection by the specified name in the path
        - No request body
    - Method: DELETE
        - Description: Deletes a collection by the name specified in the path
        - No request body

    - Method: PUT
        - Description: update the name of a collection in the session
        - Request Body:
            - Name
                - Type: string
                - Description: The name in the path is the old name of the collection, and the name inside of the request body is the new name the user desires
- /db/{dbName}/collection/{collectionName}/document
    - URL Parameters
        - dbName
            - Type: string
            - Description: the database name that you want to modify
        - collectionName
            - Type: string
            - Description: the collection name that you want to read, update, or destroy
    - Method: POST
        - Description: Creating a new document inside of the collection and database specified in the URL
        - Request Body:
            - Data
                - Type: json object
                - Description: the document you want to add to the collection. It expects a json object
    - Method: GET
        - Description: Gets all of the documents inside of the specified collection.
        - No request body

- /db/{dbName}/collection/{collectionName}/document/search
    - URL Parameters
        - dbName
            - Type: string
            - Description: the database name that you want to modify
        - collectionName
            - Type: string
            - Description: the collection name that you want to read, update, or destroy
    - Method: GET
        - Description: search all documents by keywords and returns any matches it may find
        - Request Body:
            - search
                - Type: string
                - Description: the string you want to search all the documents for
- /db/{dbName}/collection/{collectionName}/document/{id}
    - URL Parameters
        - dbName
            - Type: string
            - Description: the database name that you want to modify
        - collectionName
            - Type: string
            - Description: the collection name that you want to read, update, or destroy
        - ID
            - Type: string
            - Description: the UUID generated by the golang webserver that is the unique identifier for a specific document
    - Method: GET
        - Description: Gets a specific document by the ID provided.
        - No request body
    - Method: DELETE
        - Description: Deletes a specific document by the provided ID
        - No request body

- Method: PUT
    - Description: Update an existing document
    - Request Body
        - data
            - Type: JSON Object
            - Description: data you want to overwrite in specific document.

Important note

If there is any confusion about the spelling case or format of data check out the postman collections in docs/testing or checkout the annotation in the model package in src/pkg/model/model.go

Testing

This is the one section I wish I would have fleshed out more. Normally I automate test using ginkgo, and mock the backend or any tools I use using testify's mockery. However I was running out of time, and couldn't flesh it out to a degree that I felt comfortable presenting.

I did however include my postman tests that I used during the entire development of the database. You can import the collections from the folder */docs/testing* and follow the API guide up top, and try all of the endpoints making sure they work as intended. I did include tests inside of the requests to check to make sure you are getting the correct http status in return.