FIREMONGO

Firebase Realtime Database Emulator

FINAL PROJECT REPORT

APRIL 28, 2023

UNIVERSITY OF SOUTHERN CALIFORNIA

Kayvan Shah – kpshah@usc.edu – 1106650685 – Group 19

TABLE OF CONTENTS

INTRODUCTION

This project is an emulation of Firebase Realtime DB's RESTful API using FastAPI and MongoDB as the database. It enables users to store and retrieve data using CRUD operations and can handle concurrent requests. The project is scalable, secure, and efficient, emulating the key functionalities of Firebase Real-time Database. It is deployed on Okteto Cloud and Docker, which provide an isolated development environment that can be easily shared with a team. The API mimics the behavior of Firebase endpoints, and data is stored using MongoDB Atlas, a cloud-based database service.

TEAM MEMBERS

Sr No.	**Name**
1	Kayvan Shah
Group	19

PROJECT DETAILS

Title	**FireMongo**
Name	**Firebase Emulator**
About	**Firebase Realtime Database RESTful API Emulation**
GitHub Repo (Public)	https://github.com/KayvanShah1/firebase-realtime-db-emulator
Google Drive Link	https://drive.google.com/drive/folders/1EYhpnZKPlnQeufIIBwUM8f-GZtdPSY6s?usp=sharing

REQUIREMENTS

Requirements on your prototype system (database server):

- RESTful API which supports functions in Firebase RESTful API, which include:
  PUT, GET, POST, PATCH, DELETE, and filtering functions:
  - orderBy=”$key”/”$value”/”name”
  - limitToFirst/Last
  - equalTo
  - startAt/endAt.
- Store JSON data in another database
- It should have a proper index created in the database to support orderBy. For example, for orderBy=”name” on users.json, it should create an index on the name.
- A command-line interface that allows users to query/update the content of the database using the curl command (similar to that in Firebase), for example:
  - curl -X GET ‘http://localhost:5000/users.json?orderBy=”name”&limitToFirst=5’
  - curl -X PUT ‘http://localhost:5000/users/200.json’ -d ‘{“name”: “john”, “age”: 25}’
- **Note**: *the command should return data/response in JSON format like that in Firebase*

PROJECT PLAN

TIMELINE

Week	Dates	Tasks
Week 1	Feb 13-Feb 19	Finalizing the tech stack Going through the tutorials Design API Creating a Git Repo & project's directory structure Sample data
Week 2	Feb 20-Feb 26	Data 3odelling PUT request function POST request function
Week 3	Feb 27-Mar 5	GET request function and filters
Week 4	Mar 6-Mar 12	PATCH request function DELETE request function
Week 5	Mar 13-Mar 19	Deployment on a free site hosting platform OR using Docker Test using "curl"
Week 6	Mar 20-Mar 26	Midterm Progress Report TESTING + BUG FIXES Documentation – Docstrings, Readme & Setup
Week 7	Mar 27-Apr 2	TESTING Video Documentation
Week 8	Apr 3-Apr 9	Final Report
Week 9	Apr 10-Apr 16	BUFFER TIME
Week 10	Apr 17-Apr 23	BUFFER TIME

MILESTONES

NAME	STATUS
FINALIZING THE TECH STACK	COMPLETED
API DESIGN	COMPLETED
DATA MODELING	COMPLETED
- **V1**	COMPLETED
- **V2**	COMPLETED
REPOSITORY DIRECTORY STRUCTURE	COMPLETED
ENDPOINTS	COMPLETED
- **V1**	COMPLETED
- **V2**	COMPLETED
TEST CURL COMMANDS	COMPLETED
LANDING PAGE	COMPLETED
DEPLOYMENT	COMPLETED
DOCUMENTATIONS	COMPLETED

TASK LEVEL PROGRESS

Some milestones are tasks by themselves, so they are not repeated below.

NAME	STATUS
ENDPOINTS VERSION 1	DEPRECATED
1. POST	COMPLETED
2. PUT	COMPLETED
3. PATCH	COMPLETED
4. DELETE	COMPLETED
5. GET	BLOCKED
ENDPOINTS VERSION 2	COMPLETED
1. POST	COMPLETED
2. PUT	COMPLETED
3. PATCH	COMPLETED
4. DELETE	COMPLETED
5. GET	COMPLETED
DOCUMENTATION	COMPLETED
1. DOCSTRINGS	COMPLETED
2. API DOCS	COMPLETED
DEPLOYMENT	COMPLETED
1. DOCKER	COMPLETED
2. HOSTING	COMPLETED
TESTING	COMPLETED
1. CURL	COMPLETED
2. DEPLOYMENT	COMPLETED

LEARNINGS & EXPERIENCES

VERSION 1

CHALLENGES & OUTCOMES

- Data model used in version 1 of endpoints didn't turn out to be feasible when retrieving data from the client end.
    - Followed a nested document structure, where every document in a collection had its schema.
    - Used a single collection for housing all the incoming data.
    - Create, Update & Delete operations were simplified using this data model.
    - Read operation turned out to be complicated, which involved writing complex queries on the database server side and writing complex filter logic to get the desired results.
    - The retrieval approach failed for basic filters and hence deprecated it.

TIMELINE CATCHUP & MITIGATION

- Unexpected challenges pushed some important & secondary tasks to the upcoming week nearing the deadline and stressing the workload. Hopefully, a buffer time estimate becomes helpful here.
- Implement the ideas for a new data model such that indexing and querying data is easier by utilizing the prowess of the multiple Mongo Collections housing documents following similar JSON schema.

VERSION 2

CHALLENGES & OUTCOMES

- The revamped data model effectively utilizes the robust querying and indexing features of MongoDB, improving the read and write operations to match the capabilities of Firebase.
- While the majority of data filtering and querying is handled by MongoDB for robustness, there are certain scenarios where the logic needs to be executed on the server side for optimal performance.
    - If one request data from within a document applying orderBy and other filtering queries.
    - If one fetches data from the root, i.e, on the database level.

LIMITATIONS

- Firebase Realtime Database and MongoDB have different data models, and as a result, there are some root-level data operations that are possible with Firebase but not with MongoDB.
- In Firebase Realtime Database, data is stored as a JSON tree structure, where each node is a key-value pair. The root node is the topmost node in the tree structure. Some of the root-level data operations that are possible with Firebase but not with MongoDB include:
    - Setting data with a single call:
        - In Firebase, you can set data at the root level with a single call, which automatically creates a new node if it doesn't exist. In MongoDB, you would have to create a new document and insert it into the collection.
    - Updating data with a single call:
        - In Firebase, you can update data at the root level with a single call, which automatically updates the data if it exists. In MongoDB, you would have to use the update() method to update a document in the collection.
    - Deleting all data with a single call:
        - In Firebase, you can delete all the data at the root level with a single call. In MongoDB, you would have to delete each document in the collection individually.
- However, it is important to note that MongoDB provides more flexibility in querying and filtering data, as well as more powerful indexing and aggregation capabilities, which can be leveraged to provide more complex data operations.

OVERALL LEARNINGS & EXPERIENCES

- Choosing the right data model is crucial for the success of any application. It's important to consider the nature of the data, the type of queries that will be performed, and the expected traffic and workload.
- Filtering and querying data can be a challenging task, especially when dealing with large datasets. It's important to have a good understanding of the available querying and filtering capabilities of the chosen database system.
- Every database system has its limitations and trade-offs. It's important to understand these limitations to make informed decisions.
- Time management is crucial for the success of any project. It's important to allocate enough time for unexpected challenges and prioritize tasks based on their importance and urgency.
- Effective planning and understanding requirements, prioritizing tasks, breaking them down into smaller pieces, creating a project timeline, reviewing, and adjusting the plan as needed, and communicating effectively with team members are essential for the success of any project.

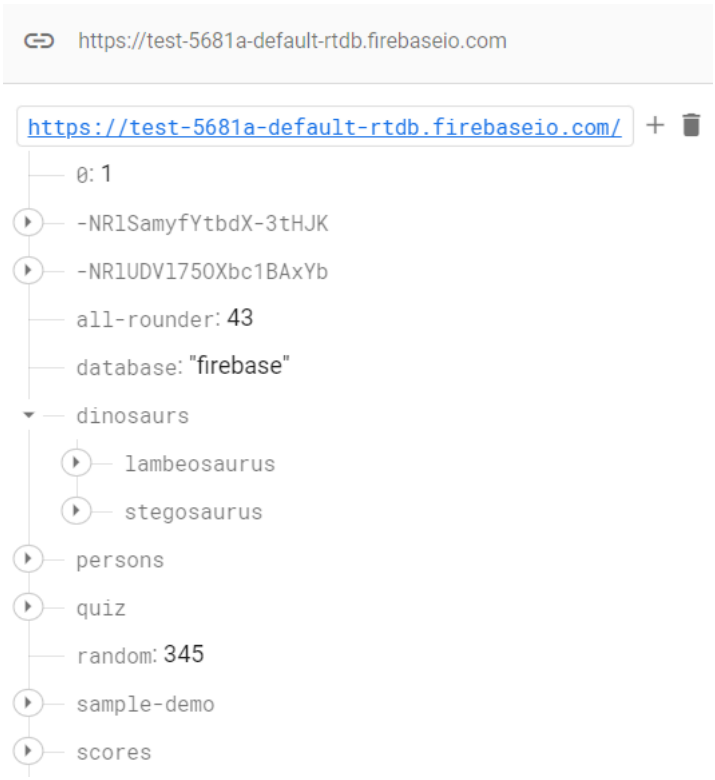
IMPLEMENTATION

TECH STACK

TECH NAME	DESCRIPTION
PYTHON	• Popular programming language with a large community and extensive library support • Provides many built-in data structures and data manipulation capabilities. • Supports both object-oriented and functional programming paradigms. • Good for scripting, automation, and building web applications
FASTAPI	• Supports asynchronous programming, which can improve performance for I/O-bound tasks. • Built on top of the Starlette framework, which provides many useful features such as automatic request validation, dependency injection, and support for web sockets. • Provides automatic API documentation using the OpenAPI specification. • Easy to use and can be deployed easily on many platforms.
HTML5, CSS	• Easy to build static webpages for prototype projects
MONGODB ATLAS	• A cloud-hosted version of the popular NoSQL document-oriented database MongoDB • Provides automatic scaling, backups, and monitoring. • Has a flexible schema-less data model that can handle complex data structures. • Provides a powerful query language and indexing system for fast data retrieval. • Offers many integrations with other cloud services and platforms
OKTETO CLOUD	• A Kubernetes-based development platform for cloud-native applications • Provides a fully managed Kubernetes cluster and development environment. • Allows developers to build, test, and deploy their applications in the cloud with ease. • Provides automatic scaling, load balancing, and high availability. • Supports many popular programming languages and frameworks
DOCKER	• A containerization platform that allows developers to package their applications into lightweight, portable containers. • Provides an isolated environment for running applications, which makes it easy to deploy and manage them across different environments. • Provides many useful features such as versioning, networking, and security. • Can be used to build, ship, and run applications anywhere, from local machines to the cloud.

DESIGN

DATA MODEL



In Firebase Realtime DB,

- Entire database is a single JSON documents.
- Has key value pairs.
- Nested up to 32 level but is never recommended.

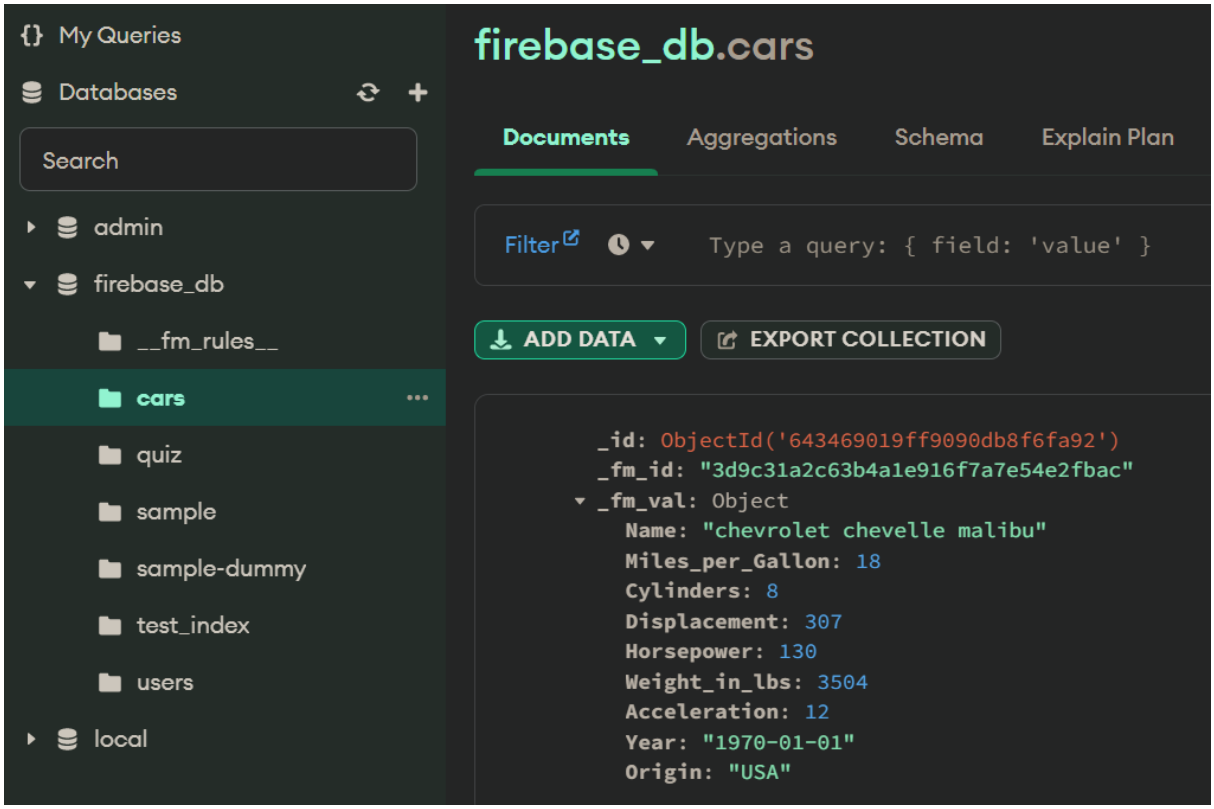
In Mongo DB,

- It is a document store where every document is a BSON.
- In a cluster
    - Multiple databases.
    - A database has multiple collections.
    - A collection comprises of JSON documents.

Here we consider a database as treated a top-level entity, where collection is a second level index which holds multiple documents. To replicate the JSON response and make it easy to parse, we create a generic document structure having.

- _id = Mongo ID
- fm_id = FireMongo ID which acts an emulated key
- fm_val = Value

Firebase also allows pushing value at the root level index, while this isn't directly possible with MongoDB – being a document store. It can be achieved with a workaround, but this feature doesn't have any plausible value or application, and this isn't recommended as a best practice when modeling a data warehouse or data store.



API DESIGN

ENDPOINTS PATH COMPONENTS

ROOT LEVEL

Path	Description
/	• data = a JSON object • collection = data.keys(), a JSON object • _fm_id = collection.keys() • _fm_val = collection.values()

COLLECTION LEVEL

Path	Description
/collection	• data = a JSON object • collection = data.keys(), a JSON object • _fm_id = collection.keys() • _fm_val = collection.values()
/collection/a	• collection = collection • _fm_id = a • nested_key = _fm_val
/collection/a/b/c	• collection = collection • _fm_id = a • nested_key = _fm_val.b.c

LIST OF ENDPOINTS

1. /.json – To perform root level read and write operation with special cases and limitation.
2. /.{path}.json – To perform collection/nested level read and write operation.

SAVE DATA

Below are the endpoints to be used for writing to the database, i.e., saving and deleting data.

Request Type	Endpoint	Path Parameters	Query Parameters
POST	/.json	-	data
POST	/{path}.json	path	data
PUT	/.json	-	data
PUT	/{path}.json	path	data
PATCH	/{path}.json	path	data
DELETE	/.json	-	-
DELETE	/{path}.json	path	-

RETRIEVE DATA

Below are the endpoints to be used for reading from the database, i.e., fetching data.

Request Type	Endpoint	Path Parameters	Query Parameters
GET	/.json	-	orderBy startAt endAt equalTo limitToFirst limitToLast
GET	/{path}.json	path	orderBy startAt endAt equalTo limitToFirst limitToLast

SET RULES

To order/sort data by “$key”, “$value”, or child key, one needs to set rules by creating an index for the key. These are some additional endpoints to get, set and delete rules.

Request Type	Endpoint	Path Parameters	Query Parameters
GET	/get-rules	-	-
PUT	/set-index	path	data
DELETE	/delete-index	path	-

APP LAYOUT AND DIRECTORY STRUCTURE

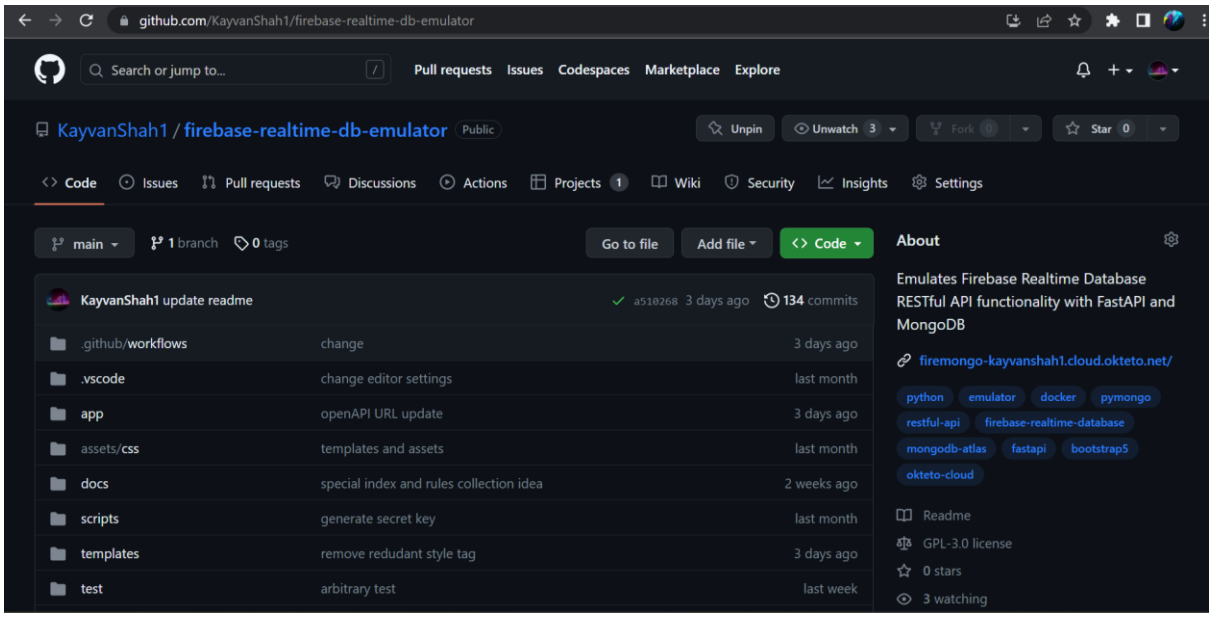
```
firebase-realtime-db-emulator
│   .env – Environment variables for the project
│   .dockerignore – Mention files to be ignored by docker-daemon while building Image
│   dev-requirements.txt – Development dependencies – formatters, linters
│   Dockerfile – Build docker image
│   docker-compose.yml – Configuration for Building Docker containers
│   okteto.yml – Deployment configuration for Okteto Cloud
│   README.md – Setup Documentation
│   requirements.txt – App dependencies
├──app – Root folder for backend application
│   │   main.py – Entrypoint file of the server
│   ├──api – Routes definition
│   │   ├──v1
│   │   │   ├──endpoints
│   │   ├──v2
│   │   │   ├──endpoints
│   ├──core – App settings and configurations
│   ├──crud – CRUD utils
│   ├──db – Database configuration
│   ├──schemas – Validation schemas
├──assets – Static files like CSS and images
│   └──css
├──docs – Other documents – ideation, conceptual
├──scripts – Shell scripts
├──templates – HTML Templates
│   └──includes – Generic templates
└──test – Testing scripts
```

RESULTS

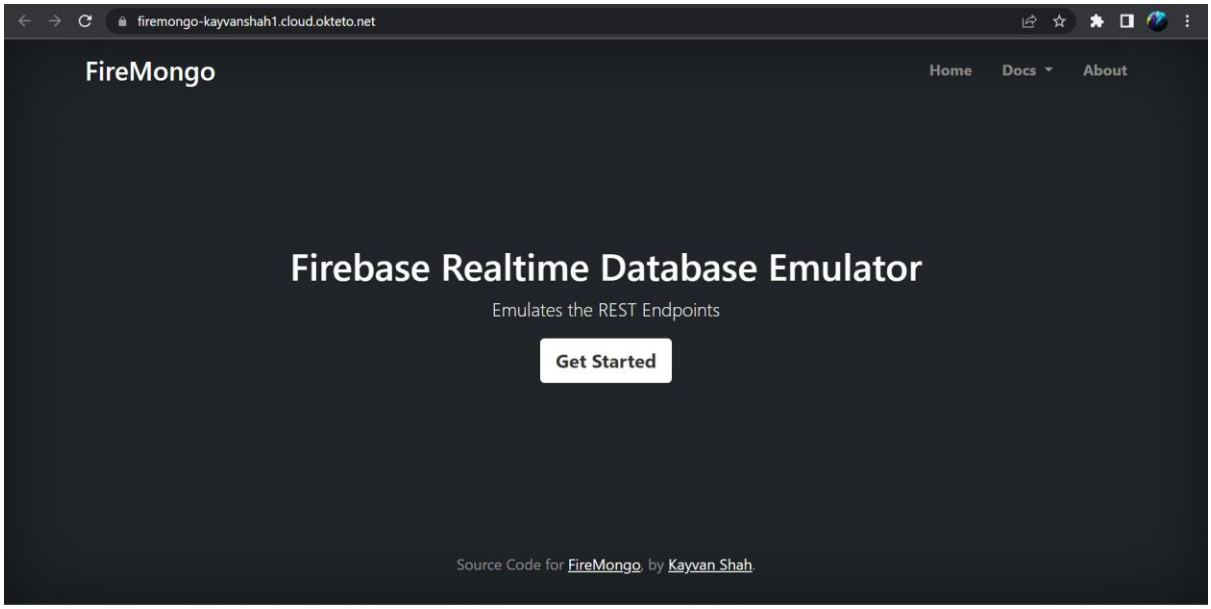
GITHUB REPOSITORY

Link to Public GitHub Repository: https://github.com/KayvanShah1/firebase-realtime-db-emulator

This is a public repository with documentation and details about cloning, setting up the development environment, installing dependencies and deploying it locally, on Docker or on a free hosting platform like Okteto Cloud.

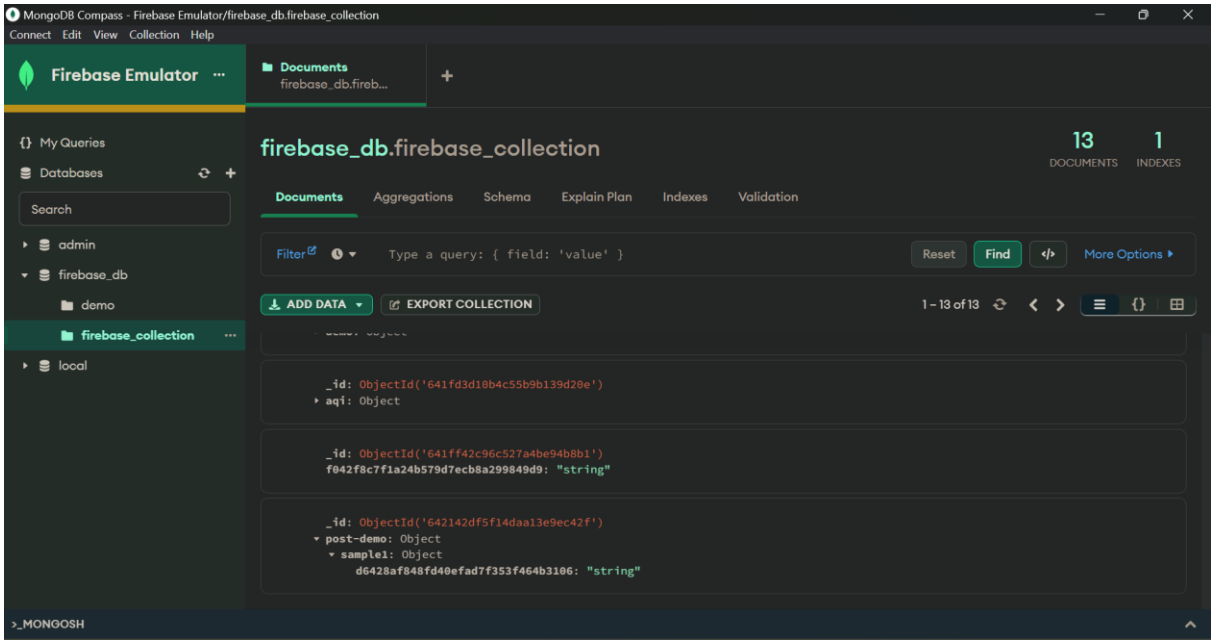


LANDING PAGE

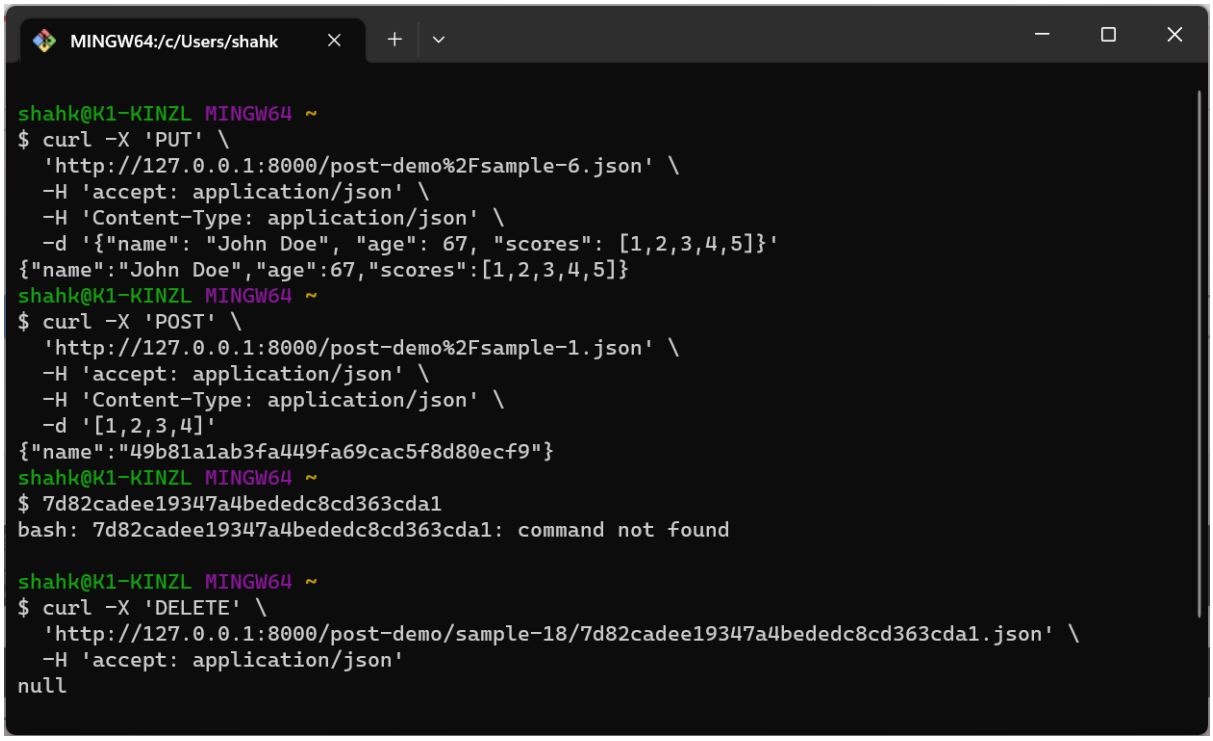


VERSION 1

MONGO ATLAS CLUSTER



COMMAND LINE INTERFACE



VERSION 2

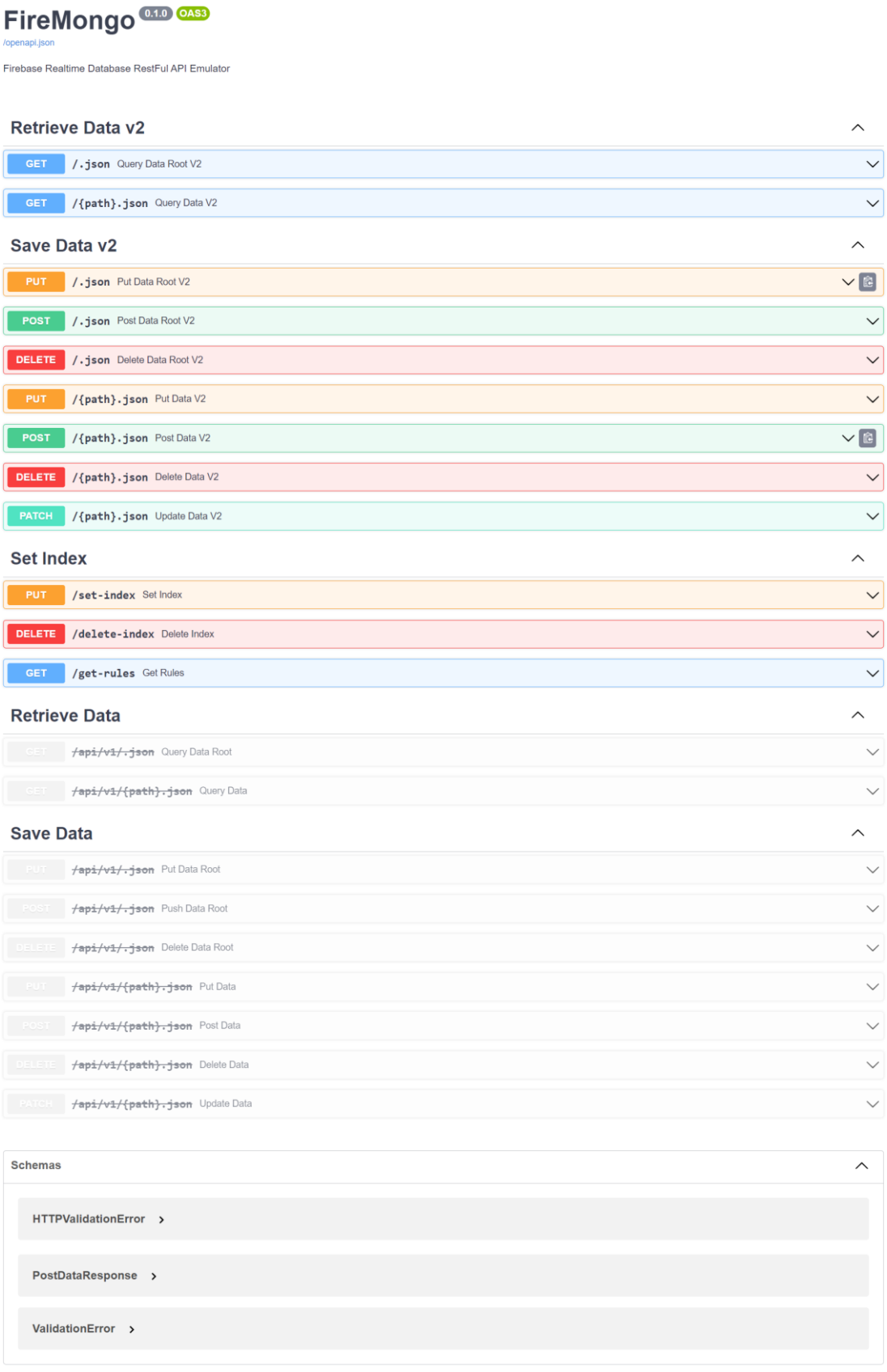
CURL CLI IMPLEMENTATION

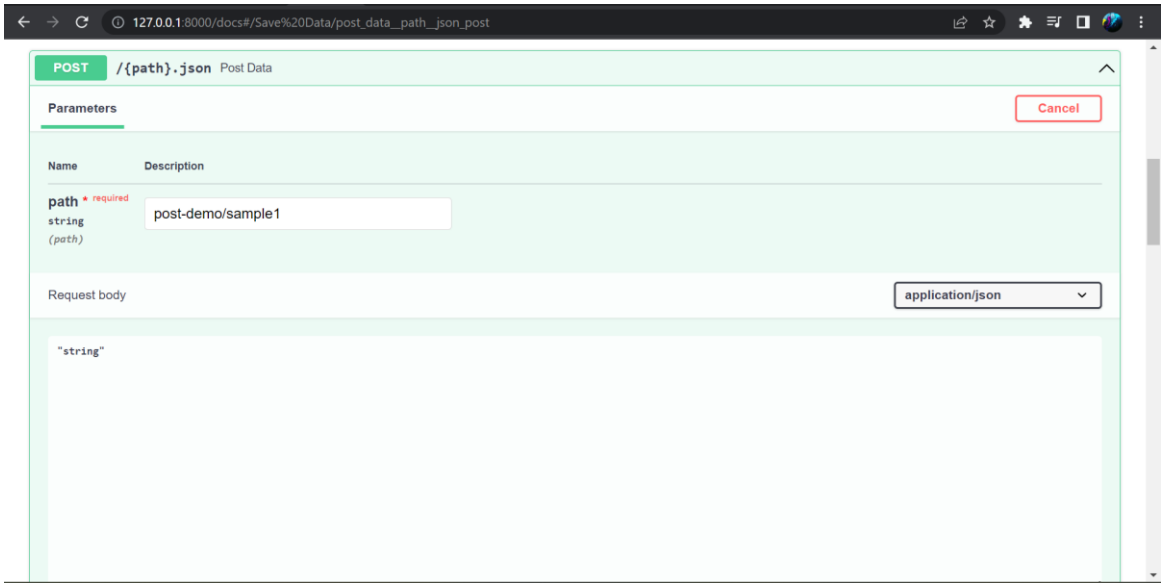
```
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' \
'http://127.0.0.1:8000/nested-users.json' \
-H 'accept: application/json'
{"1":{"userId":1,"name":{"first":"Krish","last":"Lee"},"contact":{"phoneNumber":"123456","emailAddress":"krish.lee@learningcontainer.com"},"age":54},"145":{"userId":145,
"name":{"first":"Liam","last":"Smith"},"contact":{"phoneNumber":"123456","emailAddress":"liam.smith@example.com"},"age":76},"2":{"userId":2,"name":{"first":"racks","last
":"jacson"},"contact":{"phoneNumber":"123456","emailAddress":"racks.jacson@learningcontainer.com"},"age":42},"23":{"userId":23,"name":{"first":"Darshan","last":"Gala"},"
contact":{"phoneNumber":"7850756485","emailAddress":"gala.darshan@gmail.com"},"age":26},"290":{"userId":290,"name":{"first":"Ava","last":"Johnson"},"contact":{"phoneNumb
er":"234567","emailAddress":"ava.johnson@example.com"},"age":7},"3":{"userId":3,"name":{"first":"denial","last":"roast"},"contact":{"phoneNumber":"33333333","emailAddres
s":"denial.roast@learningcontainer.com"},"age":87},"315":{"userId":315,"name":{"first":"Noah","last":"Williams"},"contact":{"phoneNumber":"345678","emailAddress":"noah.w
illiams@example.com"},"age":43},"36":{"userId":36,"name":{"first":"John","last":"Doe"},"contact":{"phoneNumber":"7134568907","emailAddress":"john78@gmail.com"},"age":57}
,"4":{"userId":4,"name":{"first":"devid","last":"neo"},"contact":{"phoneNumber":"222222222","emailAddress":"devid.neo@learningcontainer.com"},"age":23},"403":{"userId":4
03,"name":{"first":"Emma","last":"Jones"},"contact":{"phoneNumber":"456789","emailAddress":"emma.jones@example.com"},"age":15},"41":{"userId":41,"name":{"first":"Vadilal
","last":"Amratlal"},"contact":{"phoneNumber":"9650756485","emailAddress":"vadilal.amratlal@gmail.com"},"age":67},"5":{"userId":5,"name":{"first":"jone","last":"mac"},"c
ontact":{"phoneNumber":"111111111","emailAddress":"jone.mac@learningcontainer.com"},"age":19},"512":{"userId":512,"name":{"first":"Oliver","last":"Brown"},"contact":{"ph
oneNumber":"567890","emailAddress":"oliver.brown@example.com"},"age":32},"698":{"userId":698,"name":{"first":"Sophia","last":"Davis"},"contact":{"phoneNumber":"678901","
emailAddress":"sophia.davis@example.com"},"age":24},"79":{"userId":79,"name":{"first":"Parsimon","last":"Red"},"contact":{"phoneNumber":"4572135709","emailAddress":"pars
imon.red@usc.edu"},"age":63}}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'POST' \
'http://127.0.0.1:8000/nested-users.json' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
            "userId": 7423,
            "name": {
                "first": "Mia",
                "last": "Brownlee"
            },
            "contact": {
                "phoneNumber": "9876354321",
                "emailAddress": "mia.brown@learningcontainer.com"
            },
            "age": 21
        }'
{"name":"45180c77d78546d38678ed5c4fc68ff9"}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'PUT' \
'http://127.0.0.1:8000/nested-users/698/age.json' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d 144
144(firemongo)
```

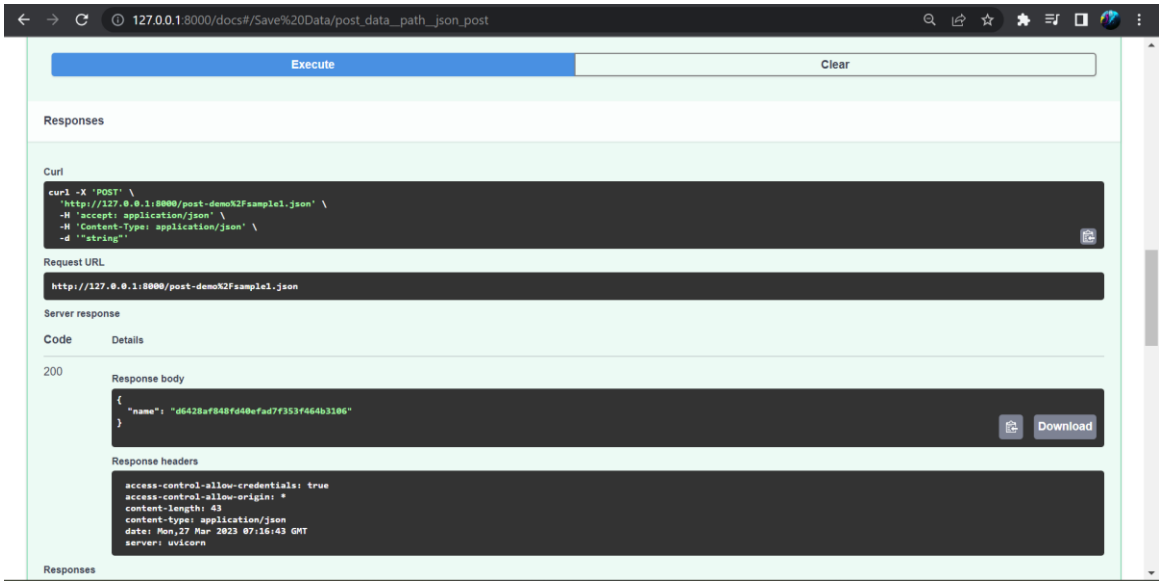
```
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' 'http://127.0.0.1:8000/nested-users/698.json' -H 'accept: application/json'
{"userId":698,"name":{"first":"Sophia","last":"Davis"},"contact":{"phoneNumber":"678901","emailAddress":"sophia.davis@example.com"},"age":144}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'PATCH' \
'http://127.0.0.1:8000/nested-users/698.json' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{"age": 34, "name/last":"Daviz" }'
{"age":34,"name/last":"Daviz"}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' 'http://127.0.0.1:8000/nested-users/698.json' -H 'accept: application/json'
{"userId":698,"name":{"first":"Sophia","last":"Daviz"},"contact":{"phoneNumber":"678901","emailAddress":"sophia.davis@example.com"},"age":34}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'PATCH' \
'http://127.0.0.1:8000/nested-users.json' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{"698/contact/phoneNumber": "6784563213", "512/age":35}'
{"698/contact/phoneNumber":"6784563213","512/age":35}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' 'http://127.0.0.1:8000/nested-users/698.json' -H 'accept: application/json'
{"userId":698,"name":{"first":"Sophia","last":"Daviz"},"contact":{"phoneNumber":"6784563213","emailAddress":"sophia.davis@example.com"},"age":34}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' 'http://127.0.0.1:8000/nested-users/512.json' -H 'accept: application/json'
{"userId":512,"name":{"first":"Oliver","last":"Brown"},"contact":{"phoneNumber":"567890","emailAddress":"oliver.brown@example.com"},"age":35}(firemongo)
```

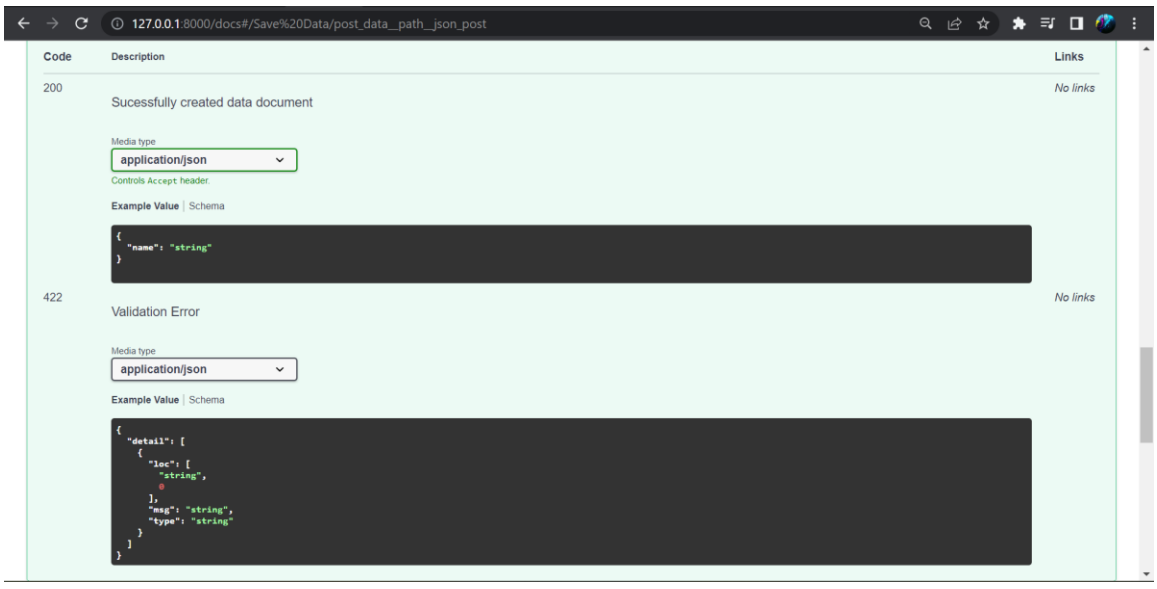
```
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'PUT' \
'http://127.0.0.1:8000/nested-users/512/extra.json' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d 'true'
1(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' 'http://127.0.0.1:8000/nested-users/512.json' -H 'accept: application/json'
{"userId":512,"name":{"first":"Oliver","last":"Brown"},"contact":{"phoneNumber":"567890","emailAddress":"oliver.brown@example.com"},"age":35,"extra":1}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'DELETE' \
'http://127.0.0.1:8000/nested-users/512/extra.json' \
-H 'accept: application/json'
{"45180c77d78546d38678ed5c4fc68ff9":{"userId":7423,"name":{"first":"Mia","last":"Brownlee"},"contact":{"phoneNumber":"9876354321","emailAddress":"mia.brown@learningconta
iner.com"},"age":21},"315":{"userId":315,"name":{"first":"Noah","last":"Williams"},"contact":{"phoneNumber":"345678","emailAddress":"noah.williams@example.com"},"age":43
},"512":{"userId":512,"name":{"first":"Oliver","last":"Brown"},"contact":{"phoneNumber":"567890","emailAddress":"oliver.brown@example.com"},"age":35},"79":{"userId":79,"
name":{"first":"Parsimon","last":"Red"},"contact":{"phoneNumber":"4572135709","emailAddress":"parsimon.red@usc.edu"},"age":63},"698":{"userId":698,"name":{"first":"Sophi
a","last":"Daviz"},"contact":{"phoneNumber":"6784563213","emailAddress":"sophia.davis@example.com"},"age":34}}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' \
'http://127.0.0.1:8000/nested-users.json?orderBy=%22name%2Ffirst%22&equalTo=%22Noah%22' \
-H 'accept: application/json'
{"315":{"userId":315,"name":{"first":"Noah","last":"Williams"},"contact":{"phoneNumber":"345678","emailAddress":"noah.williams@example.com"},"age":43}}(firemongo)
shahk@K1-KINZL MINGW64 ~/OneDrive - University of Southern California/Projects/firebase-realtime-db-emulator (main)
$ curl -X 'GET' \
'http://127.0.0.1:8000/sample-dummy.json?orderBy=%22%24value%22&limitToLast=2&startAt=%22s%22' \
-H 'accept: application/json'
{"0":"sram","2":"vram"}(firemongo)
```

SWAGGER DOCUMENTATION




127.0.0.1:8000/docs#/Save%20Data/post_data__path__json_post
POST
/{path}.json
Post Data
Parameters
Cancel
Name
Description
path
* required
string
(path)
post-demo/sample1
Request body
application/json
"string"

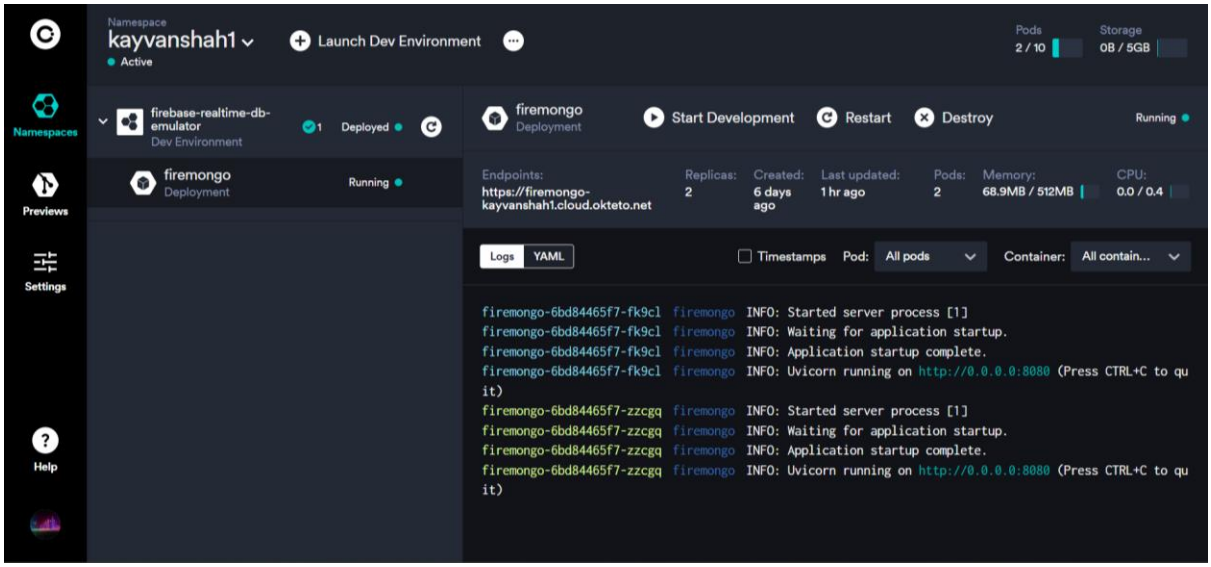

127.0.0.1:8000/docs#/Save%20Data/post_data__path__json_post
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/post-demo%2Fsample1.json' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '"string"'
Request URL
http://127.0.0.1:8000/post-demo%2Fsample1.json
Server response
Code
Details
200
Response body
{
"name": "d6428af848fd40efad7f353f464b3106"
}
Download
Response headers
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 43
content-type: application/json
date: Mon,27 Mar 2023 07:16:43 GMT
server: uvicorn
Responses


127.0.0.1:8000/docs#/Save%20Data/post_data__path__json_post
Code
Description
Links
200
Sucessfully created data document
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{
"name": "string"
}
422
Validation Error
No links
Media type
application/json
Example Value | Schema
{
"detail": [
{
"loc": [
"string",
0
],
"msg": "string",
"type": "string"
}
]
}

SERVER LOGS

```
INFO:     127.0.0.1:54496 - "PATCH /nested-users/698.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54498 - "GET /nested-users/698.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54499 - "PATCH /nested-users.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54501 - "GET /nested-users/698.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54502 - "GET /nested-users/512.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54505 - "GET /nested-users/512.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54506 - "GET /nested-users/512.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54508 - "PUT /nested-users/512/extra.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54509 - "GET /nested-users/512.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54510 - "DELETE /nested-users/512/extra.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54512 - "GET /nested-users/512.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:54517 - "GET /nested-users.json?orderBy=%22name%2Ffirst%22&limitToFirst=2 HTTP/1.1" 200 OK
INFO:     127.0.0.1:54525 - "PUT /set-index?path=nested-users HTTP/1.1" 200 OK
INFO:     127.0.0.1:54528 - "GET /nested-users.json?orderBy=%22name%2Ffirst%22&limitToFirst=2 HTTP/1.1" 200 OK
INFO:     127.0.0.1:54539 - "GET /nested-users.json?orderBy=%22name%2Ffirst%22&startAt=%22M%22&endAt=%22T%22 HTTP/1.1" 200 OK
INFO:     127.0.0.1:54541 - "GET /nested-users.json?orderBy=%22name%2Ffirst%22&equalTo=%22Noah%22 HTTP/1.1" 200 OK
INFO:     127.0.0.1:54544 - "GET /sample-dummy.json?orderBy=%22%24value%22&limitToLast=2&startAt=%22s%22 HTTP/1.1" 200 OK
```

OKTETO CLOUD DEPLOYMENT



FUTURE SCOPE

- Adding multiuser capabilities
  - A user can create his own project as in Firebase.
  - Users own a database with an account.
  - Signing with account triggers a dependent background operation which fetches the database name associated with user and configures the app to work with it.
- Add support to mimic root level of Firebase
  - Create some special collections and keys which can be associated with the root, easing the database transactions.

REFERENCES

[1] Firebase. (n.d.). Use the Firebase Realtime Database REST API
[2] The MongoDB documentation
[3] MongoDB Atlas. (2021). Cloud-hosted MongoDB
[4] Okteto. (2021). Okteto Cloud Documentation. Okteto Cloud.
[5] Sebastian Ramirez et al. FastAPI. 2020. [Online].
[6] Deta. (n.d.). Deta Space Documentation
[7] Docker. (2021). Docker Documentation
[8] Shah, K. (2021). KayvanShah1/blogAPI. GitHub.