REDIS DATABASE VISUALIZER FOR ReJSON AND REDISGRAPH MODULE (RDBVIZ)

Project report submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

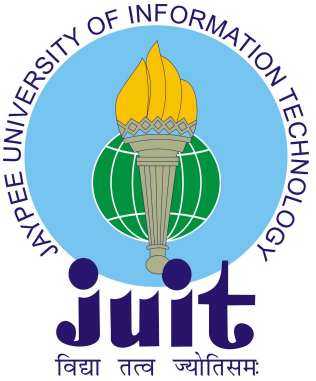
COMPUTER SCIENCE AND ENGINEERING

By

HIMANK JOG (151212)

UNDER THE GUIDANCE OF

Mr. BOPAIAH MEKERIRA



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

TABLE OF CONTENTS

DECLARATION BY THE SCHOLAR

We hereby declare that the work reported in the B-Tech thesis entitled **"REDIS DATABASE VISUALIZER FOR ReJSON AND REDISGRAPH MODULE"** submitted at **Jaypee University of Information Technology, Waknaghat, India,** is an authentic record of my work carried out under the supervision of **MR. BOPAIAH MEKERIRA**. We have not submitted this work elsewhere for any other degree or diploma.

Himank Jog, 151212

Department of Computer Science and Engineering
Jaypee University of Information Technology
Waknaghat, India

Dated:

CERTIFICATE

I hereby declare that the work presented in this report entitled "REDIS DATABASE VIZUALIZER FOR ReJSON AND REDISGRAPH MODULE" in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is an authentic record of my own work carried out over a period from February 2019 to May 2019 under the supervision of Mr. Bopaiah Mekerira, Technical Delivery Owner, HashedIn Technologies. The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Himank Jog, 151212

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr. Bopaiah Mekerira
Technical Delivery Owner
HashedIn Technologies
Bangalore
Dated:

ACKNOWLEDGEMENT

It is a pleasure to express my deep appreciation and gratitude to my mentor and to guide Mr. BOPAIAH MEKERIRA, Technical DELIVERY OWNER, HashedIn Technologies, Bangalore, Karnataka. His dedication and keen interest, and primarily, his overwhelming attitude to help his team had been solely and mainly responsible for the completion of my work. Timely advice, meticulous review and scholarly advice have helped me a lot.

I owe a deep sense of gratitude to all my team, respectful and supportive, who helped me to participate in the very early stages of project development and I expect them to continue to guide me constantly. Their prompt inspirations, timely suggestions with kindness, enthusiasm and dynamism allowed us to complete my thesis.

I thank very much MR. SRIPATHI KRISHNAN, Chief Technical Officer, HashedIn Technologies, Bangalore, Karnataka, for his help and cooperation throughout my development period.

HIMANK JOG (151212)

LIST OF ABBREVIATIONS AND ACRONYMS

ABBREVIATION	MEANING
RDBTools	Redis DataBase Tools
CLI	Command Line Interface
GUI	Graphical User Interface
JS	JavaScript
Redis	REmote DIctionary Service
JSON	JavaScript Object Notation

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The report covers my work on creating a visualization tool, RDBVIZ, that makes it easier for the user to interact with the non-native data stored in the Redis Database. Redis is an in-memory database that is used for majorly for the caching purpose that makes is faster than all the other databases present in the market. Specific to the ReJSON and RedisGraph module, the tool allows user to perform not only the basic operations of Create, Read, Update and Delete, but also allows the user to navigate through the data seamlessly and filter the data based on the user input. Starting with the ReJSON, it provides the ability to store the data in the JSON format. Second, RedisGraph allows the user to store the data in the form of a graph, thus acting as an in-memory Graph Database. As a future prospect, this tool will be integrated with the company's product RDBTools.

CHAPTER – 1

INTRODUCTION

1.1 About the Company

HashedIn Technologies is a software development company that focuses on developing deep tech SAAS products and platforms. It majorly deals in the service sector, working with companies to develop Intelligent Saas products. Along with the services team, the company also maintains two of its own products that are MeetNotes and RDBTools.

1.2 Understanding Redis

Redis, as the name comes from **RE**mote **DI**ctionary **S**ervice. It is an open source database server that is mostly used as a caching service. Started by Salvatore Sanfilipo, originally written to improve the performance of a real-time web analytics startup by him. The entire database, as it has come to be known as, is written purely in C language. It is an in-memory data structure store, therefore all the data is stored in the RAM. Thereby making it faster to perform CRUD operations on the data. The only drawback is that the RAM is a volatile memory, therefore all the data is gone once the machine reboots. Hence, it is used for the caching purposes.

1.2.1 Introduction to Redis

1.2.1.1 Data structures in Redis

An in-memory data structure store, used for various purposes in the software development cycle. Majorly it is used as database service, cache service or as a message broker. Natively it supports a variety of data structures that help in storing data. These help in storing data according to the requirement of the user. It uses the

Key Value pair technique to store data where key is a “String” and value is one of the supported data structures.

Some of the supported data structures are as follows

- STRINGS
    - An array that stores characters.
    - The most basic data structure in Redis.
    - As a result used as a key to store values for the Key-Value Pair.
    - **Operations: [GET, SET]**
        - Sample Workflow

            > SET MYVAL somevalue

            OK

            > GET MYVAL

            “Somevalue”

- SETS
    - Collection of strings.
    - Unordered in nature.
    - Used for mathematical set operations such as Intersection, Union and Difference.
    - **Operations: [SADD, SISMEMBER, SMEMBERS, SINTER, SUNIONSTORE, SPOP, SCARD]**
        - Sample Workflow

            > **SADD TESTSET 5 6 7**

            (integer) 3

            > **SMEMBERS TESTSET**

            1. 5
            2. 6
            3. 7

            > **SISMEMBER TESTSET 5**

(integer) 1

> **SISMEMBER TESTSET 8**

(integer) 0

> **SINTER RANDOMTAG:2:TOPIC**

> SADD DECK A1 A2 A3 B1 B2 B3

(integer) 6

> **SUNIONSTORE PLAY:1:DECK DECK**

(integer) 6

> **SPOP PLAY:1:DECK**

"A2"

> **SPOP PLAY:1:DECK**

"B1"

> **SCARD PLAY:1:DECK**

(integer) 4

- SORTED SETS
    - Amalgamation of a Set and a Hash.
    - It conceptualizes the idea of score.
    - We provide the value and along with its score.
    - The values are sorted on the basis of their score.
    - **Operations: [ZADD, ZRANGE, ZREVRANGE, ZRANGEBYSCORES, ZREMRANGEBYSCORE, ZRANK]**
        - Sample Workflow

> **ZADD HACKERSLIST 1 "ALAN KAY"**

(integer) 1

> **ZADD HACKERSLIST 2 "SOPHIE WILSON"**

(integer) 1

> **ZADD HACKERSLIST 3 "RICHARD STALL"**

(integer) 1

> **ZADD HACKERSLIST 4 "ANITA BORG"**

(integer) 1

> **ZADD HACKERSLIST 5 “YUKIHIRO”**

(integer) 1

> **ZADD HACKERSLIST 6 “HEDY LAMARR”**

(integer) 1

> **ZADD HACKERSLIST 7 “CALUDE SANNON”**

(integer) 1

> **ZRANGE HACKERSLIST 0 2**

1) **“ALAN KAY”**
2) **“SOPHIE WILSON”**

> ZREVRANGE 0 2

1) **“CALUDE SANNON”**
2) **“HEDY LAMARR”**

> ZRANGEBYSCORE HACKERSLIST 0 3

1) **“ALLAN KAY”**
2) **“SOPHIE WILSON”**
3) **“RICHARD STALL”**

> ZREMRANGEBYSCORE HACKERSLIST 0 7

(integer) 7

> **ZRANK HACKERSLIST YUKIHIRO**

(integer) 5

- HASHES
  - It is the usual collection of field-value pairs.
  - There’s a unique key that maps to a value that is stored as a string.
  - **Operations: [HMSET, HGET, HMGET, HINCRYBY, HGETALL]**
    - Sample Workflow

> HMSET CLIENT:10 NAME TEST YEAR 1997

OK

> **HGET CLIENT:10 NAME**

"TEST"

> HGETALL CLIENT:10

1) **"**Name**"**
2) **"**TEST**"**
3) **"**YEAR**"**
4) **"**1997**"**

> HMGET CLIENT:10 NAME YEAR RANDOM

1) **"**TEST**"**
2) **"**1997**"**
3) (nil)

> HINCRBY CLIENT:10 YEAR 20

(integer) 2017

- LISTS
  - Implemented as linked lists within Redis.
  - Insertion is constant time.
  - Access is linear time.
  - **Operations: [RPUSH, LPUSH, RPOP, LRANGE]**
    - Sample Workflow

> **RPUSH TEST A**

(integer) 1

> **RPUSH TEST RANDOM**

(integer) 1

> **LPUSH TEST B**

(integer) 1

> **LRANGE TEST 0 -1**

1) **"**B**"**
2) **"**A**"**

3) **“**RANDOM**”**

> RPOP TEST

“RANDOM”

- BITMAPS
    - Bit-oriented operations in disguise of n actual data type.
    - Helpful in saving space when storing certain type of information.
    - By default the value on a position is not set i.e 0.
    - **Operations: [SETBIT, GETBIT, BITCOUNT, BITOP, BITCOUNT, BITPOS]**
        - Sample Workflow

> **SETBIT BITS 11 1**

(integer) 1

> **SETBIT BITS 13 1**

(integer) 1

> **GETBIT BITS 11**

(integer) 1

> **BITCOUNT BITS**

(integer) 2

- HYPERLOGLOGS
    - This is a special kind of data structure specific to a single task of counting unique elements.
    - It is based on Memory-Precision Trade off.
    - The Redis implementation takes an approximation based approach.
    - The standard error in Redis implementation is less than 1%.
    - In the worst case, as calculated, constant amount of memory i. 12KB is needed.
    - This is a lot less if there are few elements.
    - **Operations: [PFADD, PFCOUNT]**
        - Sample Workflow

> PFADD HLL P Q R S T P

(integer) 1

> PFCOUNT HLL

(integer) 5

- GEOSPATIAL INDEXES
    - It is used to add the specified geospatial items that includes latitude, longitude and name to the declared key.
    - It provides with the functionality to query for the radius or geo radius by number.
    - It provides the functionality to identify the distance between two places.
    - **Operations: [GEOADD, GEODIST, GEORADIUS]**
        - Sample Workflow

> GEOADD Bangalore 56.787655 76.345443 ”Chennai” 74.987389 12.253647 “Pune”

(integer) 2

> GEODIST Bangalore Chennai Pune

“1662.8767”

> GEORADIUS Bangalore 15 87 400 km

1) “**Chennai**”

- STREAMS
    - It is a new data type that models a LOG DATA STRUCTURE in a way that is supposed to be more abstract in nature.
    - The continuous streaming data is stored and analyzed in real time with this data structure
    - **Operations: [XADD, XLEN, XRANGE]**
        - Sample Workflow

> XADD thisstream * sensor-id 4321 temperature 31.8

1572930283738-0

> XLEN thisstream

(integer) 1

> **XRANGE thisstream - +**

1) **1)** 1572930283738-0
2) 1) "sensor-id"
   2) "4321"
   3) "temperature"
   4) "31.8"

1.2.1.2 Inbuilt-Tools in Redis

It provides many in-built tools and services to make the task of the user easier. Some of these services are

- **Replication**
  -
  - Ability to replicate data from master to slave nodes.
  - Whenever a link is established, the slave will automatically start replicating the data form the master to itself to become the exact copy of the master regardless of whatever state the master is in.
  - This replication works using 3 main mechanisms
    i. In a well established connection
       - Master keeps the slave nodes updated
    ii. In a broken connection (Possible Partial Resynchronization)
       - Slave tries for partial resynchronization.
       - It tries to obtain the commands or part of commands that were missed by slave due to disconnection.
    iii. In a broken connection (Impossible Partial Resynchronization)

- Full resynchronization is done.
- Master creates a snapshot of all the data at present and transfers it to the slave.

- **Lua Scripting**
  - There is an inbuilt Lua interpreter in Redis.
  - The argument EVAL is the shorthand for the Lua script in Redis.
  - Therefore, there is no need to define Lua function.
  - Arguments can be accessed by Lua using the global variable KEYS.
  - Additional arguments can be accessed by Lua using global variable ARGV.
  - Sample Workflow

    > eval "return {KEYS[1], ARGV[1]}" 1 key1 firstARG

    1) "Key1"
    2) "firstARG"
  - **Calling Redis commands from a Lua script**

    i. redis.call()

    - Returns the result of the evaluation.
    - If an error is raised, it will raise a Lua error.
    - This forces EVAL to return an error.
    - This error is returned to the command caller.

    ii. redis.pcall()

    - Similar to redis.call(), returns the result of evaluation.
    - If an error is raised, it will trap the error.
    - Return Lua table representing the error.

- The conversion table from Redis to Lua

Redis	Lua
Redis Integer	Lua Number
Redis Bulk	Lua String
Redis Multi Bulk	Lua Table
Redis Status	Lua Table with a single OK field containing status
Redis Error	Lua Table with a single ERR field containing the error
Redis Nil Bulk and Multi Bulk	Lua False Boolean type

- The conversion table from Lua to Redis

Lua	Redis
Lua Number	Redis Integer
Lua String	Redis Bulk
Lua Table (array)	Redis Multi Bulk
Lua Table with a single OK field	Redis Status
Lua table with a single ERR field	Redis error
Lua Boolean False	Redis Nil Bulk

- **Atomicity of the scripts**
    i. Similar Lua Interpreter in used in Redis to run all the commands.
    ii. It is the responsibility of Redis to guarantee that a script is executed in an atomic way.
    iii. No parallel script or a command of Redis will not be in an execution phase while a script is in the execution phase.
    iv. This enforces the script in the running phase to not be slow cause all the other functions will be halted due to Atomicity.

- **Error Handling**
    i. If redis.call() function is used, the error will stop the execution of the currently running script.
    ii. This is done in a way that it becomes obvious to the user that the error was generated by a script.
    iii. Similarly if redis.pcall() is used, no error is raised.
    iv. Rather an error object is returned.

- **LRU Cache**
    - The main purpose is to use Redis as an LRU cache.
    - The Least Recently Used eviction policy is the one majorly used, there are other methods available.
    - The is a memory usage limit to a certain amount that is fixed.
    - **Maxmemory Configuration Directive**
        i. Used to limit the memory usage to a fixed amount.
        ii. To set the fixed amount we need to edit *redis.conf* file.

            maxmemory 100mb

        iii. Setting *maxmemory* to 0 describes that there are no memory limits.
        iv. For a 64bit system this is a default behaviour.

v. For a 32bit system, implicitly the memory limits to 3GB.

- **Eviction Policies**
  i. When the Redis reaches the *maxmemory* limit, then the behaviour is configured using the *maxmemory-policy* configuration directive.
  ii. There are following policies available as of now

    - **Noeviction**
      - Lets the user know through an error that memory limit has been reached and the command the user is trying to execute may cause in usage of more memory.
    - **Allkeys-lru**
      - It suggests to remove the Less Recently Used keys first to make some space for the new data added
    - **Volatile-lru**
      - Similar to **Allkeys-lru**, it also evicts to remove the Less Recently Used (LRU) keys first.
      - The only keys that are in the expire set are considered for removal.
      - This is done in order to make some space for the new incoming data.
    - **Allkeys-random**

- The keys are evicted in the random order.
- It is done in order to make space for the new data to be added.

- **Volatile-random**

  - The keys are evicted in the random order.
  - It is done in order to make space for the new data to be added.
  - The keys to be evicted are from the expire set.

- **Volatile-ttl**

  - The keys are evicted in using the Shorter Time to Live First policy.
  - This is done in order to make some space for the new data to be added.

- **How the Eviction Process Works**
  i. When a client runs a new command, it results in adding more data.
  ii. Redis constantly checks the usage of the memory, and as soon as it exceeds the *maxmemory* limit, it starts to evict keys according to the policy.
  iii. A new command is executed, so on and so forth.

- **Approximated LRU algorithm**
  i. Within Redis, it is not an exact implementation of the actual LRU algorithm.

ii. It isn't able to pick up the *best candidate* but banks up on the approximation to pick up the next candidate key to evict.

iii. The main reason why we use the Redis LRU Algorithm for eviction policies is that we can tune the precision of the algorithm by changing the quantity of the samples we choose to check for every eviction.

iv. The reason why the true LRU algorithm is not used is because it uses comparatively more memory.

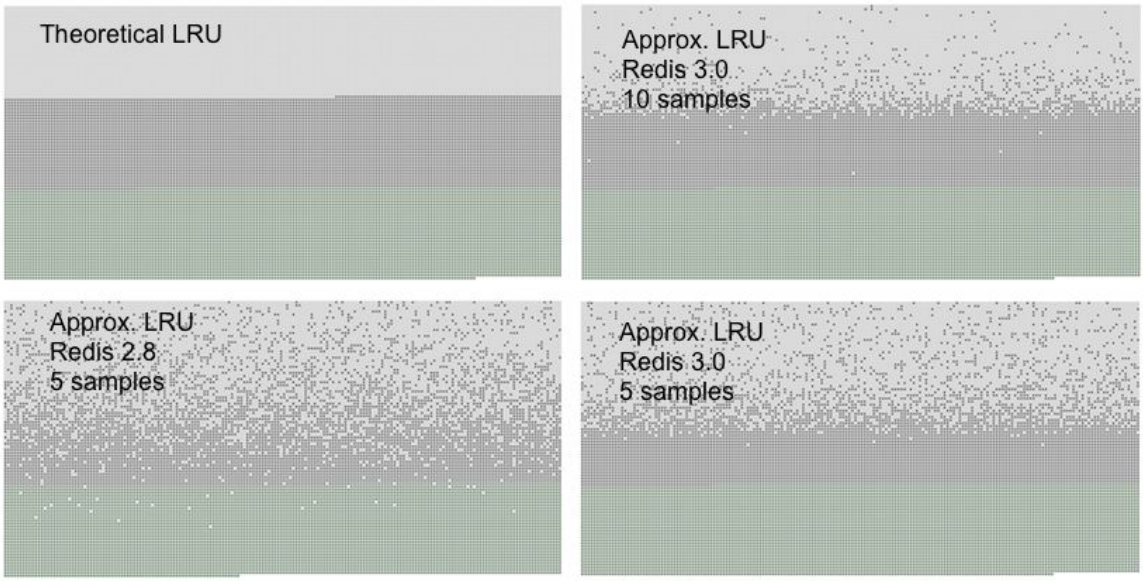


Figure 1.

LRU approximation compared with true LRU

- **The new LFU mode**

  i. With the new version rolling out, Least Frequently Used eviction mode will be available.

  ii. Its performance is same as LRU, but for certain cases LFU works more efficiently than the LRU,

  iii. Policies for LFU mode are

  - volatile-lfu
  - Allkeys-lfu

iv. The default configuration is as follows
    - Saturate the counter at, around, 1,000,000 requests.
    - The counter is decayed every minute.

- **TRANSACTIONS**
    - There are only 4 commands to take care of the Transactions.
        i. MULTI
        ii. EXEC
        iii. DISCARD
        iv. WATCH
    - All the above mentioned commands are serialized, that implies that they are sequentially executed.
    - This implies that no request can be issued when another request is in the middle of the execution.
    - These commands are run/executed as a single isolated operation.
    - It is made sure that either all the commands are processed or none, thus making it atomic along with isolation as well.
    - MULTI command is used to make the multiple commands execute in atomic fashion.
    - The EXEC command is used to execute all the transactions that were pushed in to the queue of the MULTI.
    - DISCARD is used to discard a current session of commands in execution or pre-execution phase.
    - WATCH is used to keep a check on a key and track changes, either all the changes will be applied or none.
    - **Error inside a TRANSACTION**
        i. There could be an error during queuing of the commands, so this may result in an error when executing the EXEC command.

ii. This could be because of several reasons such as, a syntactically wrong command or some condition where memory usage limit is being exceeded.

iii. An error may occur after the EXEC command is executed, such as accessing or setting wrong values for wrong keys.

- **Modules**
  - There are only few few data structures native to Redis as discussed in section 1.2.1.
  - So, in order to provide support for data structures for specific operations, module of that specific data structure is introduced.
  - It provides the ability to work with non-native data structures in order to utilize the speed of Redis for other tasks than just the ones that are natively supported
  - Following tables explains some of the supported modules and their usage.

MODULE	USE CASE
neural-redis	Online trainable neural networks as Redis data types.
RediSearch	Full-Text search over Redis.
RedisJSON	A JSON data type for Redis.
rediSQL	Provides full SQL capabilities embedding SQLite.
redis-cell	Provides rate limiting in Redis as a single command.
RedisGraph	It is an implementation of graph database with a Cypher-based querying language that uses sparse adjacency matrices.
RedisML	Machine Learning Model Server.
RedisTimeSeries	Time-series data structure for redis.

RedisBloom	Scalable Bloom filters.
cthulhu	Extend Redis with JavaScript modules.
redis-cuckoofilter	Hashing-function agnostic Cuckoo filters.
RedisAI	Provides support to serve tensors and execute deep learning graphs.
redis-roaring	CRoaring library is used to implement roaring bitmap commands for Redis.
redis-tdigest	This is used for accurate online accumulation of certain statistics that are usually rank-based.
Session Gate	Session management with multiple payloads achieved using token that were cryptographically signed tokens.
countminsketch	An apporximate frequency counter.
ReDe	Low Latancy timed queues (Dehydrators).
topk	An almost deterministic top k elements counter.
commentDis	Add comment syntax to your redis-cli scripts.

1.2.1.3 ReJSON and RedisGraph

ReJSON

This is a module provided by RedisLabs that supports the data structure for a JSON type object. It implements ECMA-404. ECMA-404 is a standard meant for the JSON data interchange as a native type. It allows the user to perform the basic operations such as storing, fetching and updating the values stored in JSON object from the Redis Key.

Primary Features:

- Full support of the JSON standard
- For selecting elements inside documents JSONPath-like syntax is used.
- In order to allow fast access to sub-elements the documents are stored as binary data in a tree structure.
- For all JSON values types, there are Typed atomic operation.

Launching the ReJSON module

- Preferred through the Docker Image provided by RedisLabs.
- Following command is used to run the Docker Image

```
docker run -p 6379:6379 --name redis-redisjson redislabs/rejson:latest
```

Working with the module.

Basic functionalities

```
127.0.0.1:6379> JSON.SET foo . '"bar"'
OK
127.0.0.1:6379> JSON.GET foo
"\"bar\""
127.0.0.1:6379> JSON.TYPE foo .
string
```

Extra functions provided

```
127.0.0.1:6379> JSON.STRLEN foo .
3
127.0.0.1:6379> JSON.STRAPPEND foo . '"baz"'
6
127.0.0.1:6379> JSON.GET foo
"\"barbaz\""
```

Working with arrays

```
127.0.0.1:6379> JSON.SET arr . []
OK
127.0.0.1:6379> JSON.ARRAPPEND arr . 0
(integer) 1
127.0.0.1:6379> JSON.GET arr
"[0]"
127.0.0.1:6379> JSON.ARRINSERT arr . 0 -2 -1
(integer) 3
127.0.0.1:6379> JSON.GET arr
"[-2,-1,0]"
127.0.0.1:6379> JSON.ARRTRIM arr . 1 1
1
127.0.0.1:6379> JSON.GET arr
"[-1]"
127.0.0.1:6379> JSON.ARRPOP arr
"-1"
127.0.0.1:6379> JSON.ARRPOP arr
(nil)
```

Working with the JSON style object

```
127.0.0.1:6379> JSON.SET obj . '{"name":"Leonard Cohen","lastSeen":1478476800,"loggedOut": true}'
OK
127.0.0.1:6379> JSON.OBJLEN obj .
(integer) 3
127.0.0.1:6379> JSON.OBJKEYS obj .
1) "name"
2) "lastSeen"
3) "loggedOut"
```

RedisGraph

It is the module that provides the support to utilize the Redis as a graph database.

It is the first of its type in the context of queryable Property Graph database that uses sparse matrices in order to represent the adjacency matrix in graphs as well as linear algebra to query the graph.

Primary Features

- Based on the Property Graph Model.

- Attributes can be assigned to Nodes and Relationships that are basically vertices and edges respectively.
- Node(vertices) can be labeled.
- A relationship type can be associated with a relationship.
- Sparse adjacency matrices are used to represent the graphs.
- The query language to query the graph database is Cypher.
- The queries in Cypher are translated to linear algebra expressions.

Launching the RedisGraph module

- Preferred through the Docker Image provided by RedisLabs.
- Following command is used to run the Docker Image

```
docker run -p 6379:6379 -it --rm redislabs/redisgraph
```

Working with the module

Basic working functionalities

```
$ redis-cli
127.0.0.1:6379> GRAPH.QUERY MotoGP "CREATE (:Rider {name:'Valentino Rossi'})-[:rides]->(:Team {name:'Yamaha'})
1) (empty list or set)
2) 1) Labels added: 2
   2) Nodes created: 6
   3) Properties set: 6
   4) Relationships created: 3
   5) "Query internal execution time: 0.399000 milliseconds"
```

Other functionalities

```
127.0.0.1:6379> GRAPH.QUERY MotoGP "MATCH (r:Rider)-[:rides]->(t:Team) WHERE t.name = 'Yamaha' RETURN r,t"
1) 1) 1) "r.name"
      2) "t.name"
   2) 1) "Valentino Rossi"
      2) "Yamaha"
2) 1) "Query internal execution time: 0.122000 milliseconds"
```

1.3 Introduction to ReactJS

ReactJS is a frontend framework that was developed by Facebook. It's declarative states that it makes it painless to create the User Interfaces that are interactive in nature. It provides the developer with the ability to design simple views for each of the state in the application and React's framework will efficiently update and render the required components when the data is updated.

It's design ensures that the build encapsulated components that are designed to manage their own states and thereby composing them to make complex User Interfaces.

The following diagram explains the workflow of a React Framework rendering

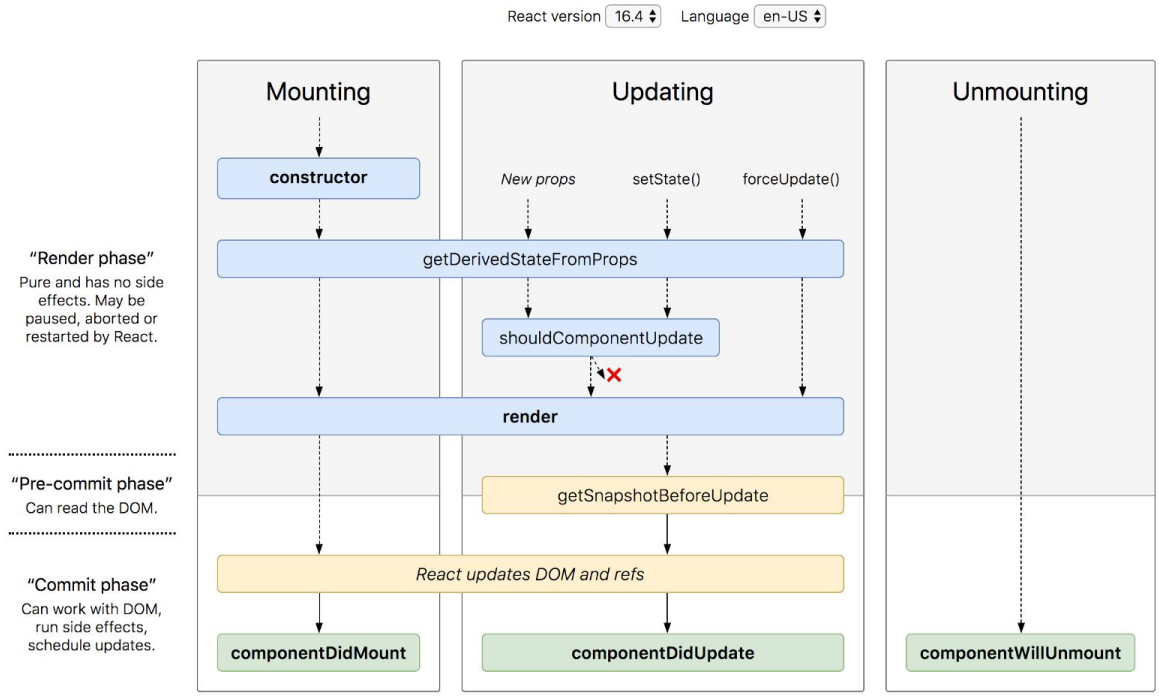


Figure 2

1.4 Introduction to Django

Django is a Model-View-Template framework that takes care of the backend part of a server. It is a high-level Python Web Framework that is designed to encourage rapid development and clean, pragmatic design.

Primary Features

- It provides the tools and services that speed up the process from concept to completion phase.
- It comes with inbuilt security so that the developer has no overhead for managing the security part.
- It is highly scalable and hence is supported by many large scale enterprises in the industry.

Following workflow diagram describes how Django works as an MVT framework.

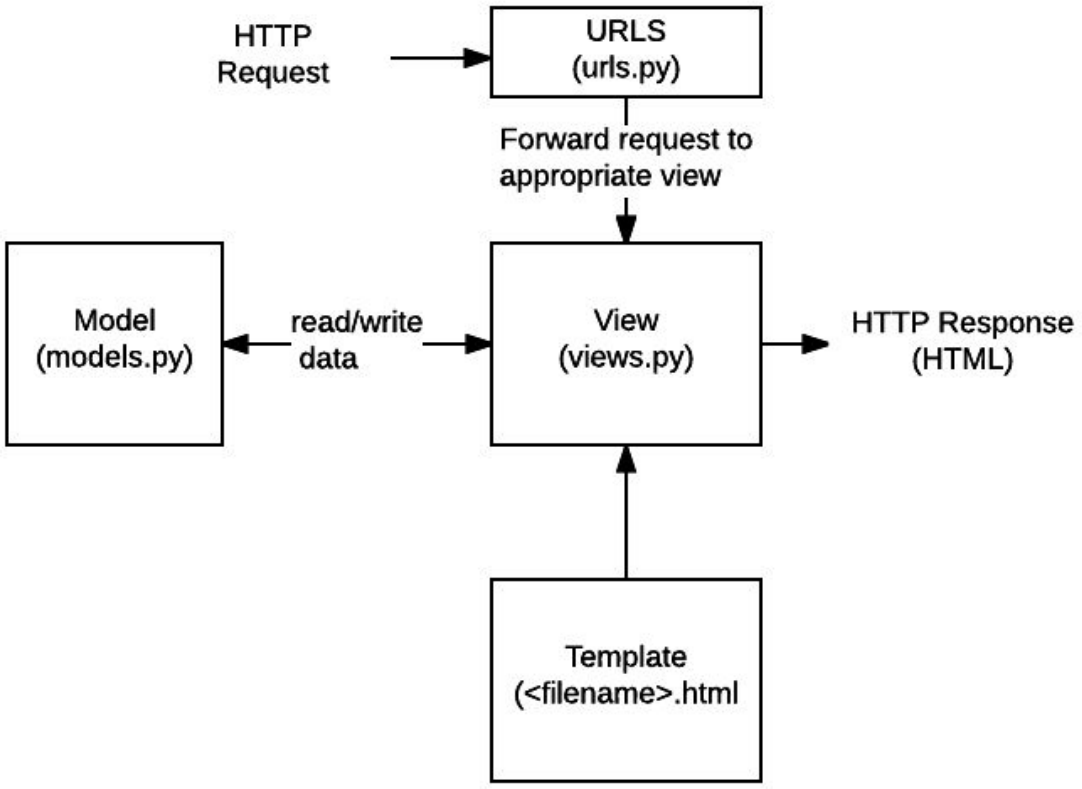


Figure 3

1.5 Introduction to Heroku

Heroku is a cloud application platform as a service that provides the ability for deployment of an application online. We can also define Heroku as a container-based cloud PaaS (Platform as a Service).

Developer can scale, manage and deploy modern apps on this platform. It offers the developers an elegant, easy to use and a flexible platform to work on.

It supports deployment of applications based on various technologies such as Node.js, Ruby, Java Go, Scala, Python Clojure.

Following workflow describes the Development Team Workflow

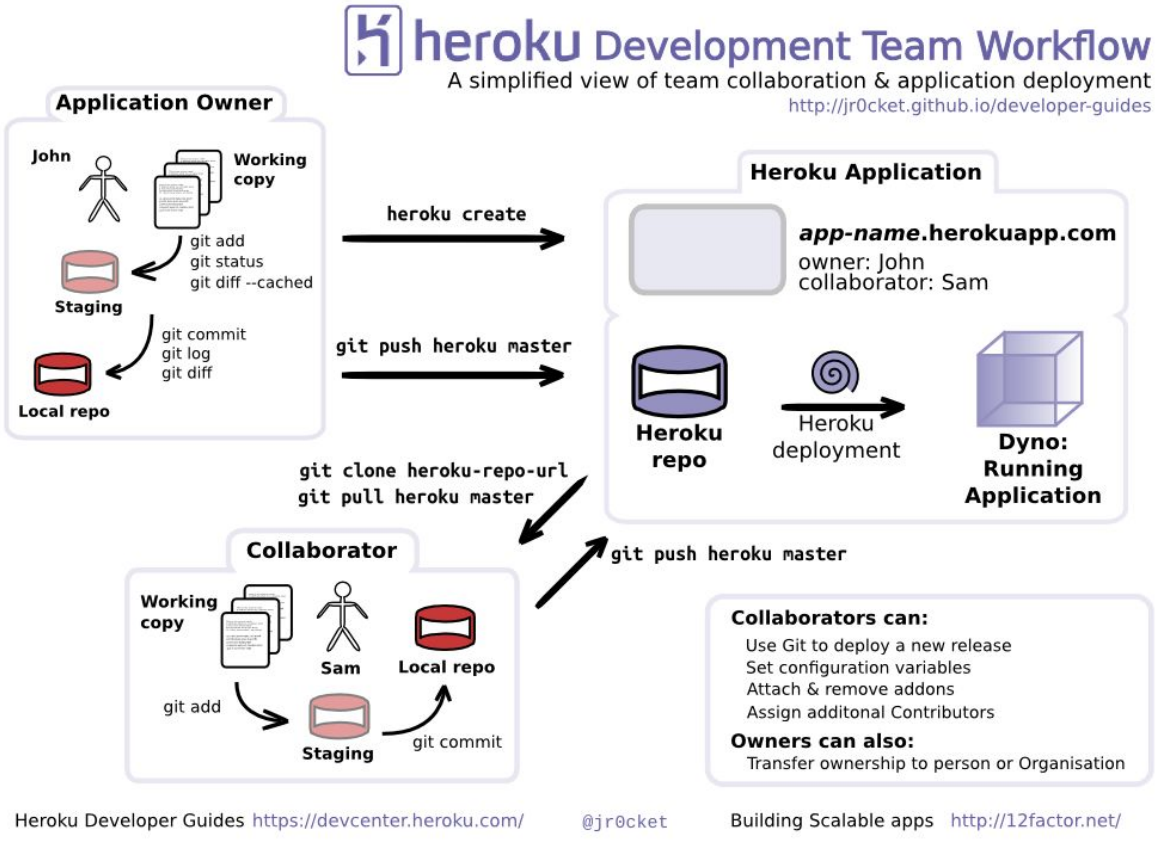


Figure 4

CHAPTER – 2

LITERATURE SURVEY

For the development of this tool the basic working and understanding of various frameworks as in how frontend rendering, backend api creation and working of the Redis was required. In order to gain an insight in to all of these, All the required knowledge base has been gained from the following documents

2.1 Redis Documentation

This document/documentation provides all the required knowledge to get started with Redis not only as a Database but as other services mentioned in the introduction. It gives the list of all the commands, services, features and APIs provided by the Redis.

All information regarding the programming with Redis, redis modules API, Tutorials, Frequently Asked Questions, Administration, Embedded, IoT, Troubleshooting, Redis Cluster, Specifications, Resources and Use Cases were all provided by this documentation.

2.2 ReactJS Documentation

ReactJS documentation provided with all the in hand tutorials and resources required to build User Interface and Improved User Experience, i.e UI/UX. Starting with the concepts of the ReactJS as to how it works and renders the components to all the way to building the basic first app on react, every information and tutorial was provided by this document.

The introduction and examples for working with Redux in ReactJS are the only source of truth for Redux as it is a complicated tool/feature to implement and work around with.

2.2 Django Documentation

Described as the best documentation for any backend framework yet, it provided with all the knowledge required to develop not only the RESTful APIs for the project but also provided with the space to deal with the business logic and data source as well.

With the help of this document all the underlying code for the project was developed that defined how the structure of the project might end up like. With the Domain Model to work with, it not only provided the required overview to the structure of the project but also equipped me with the in depth knowledge of the working and hence I was able to use this to minimize the complexity of the project.

2.3 Heroku Documentation(Python)

The documentation provided with all the necessary steps and knowledge required to deploy a Python application on the platform. All the workflow and procedure required for an application to be deployed and run online with a domain name attached to it, this all was gathered from this documentation.

CHAPTER – 3

SYSTEM DEVELOPMENT

3.1 System Design

The model is highly inspired from the Domain Model. It is a 3-Tier structure that has a front end framework ReactJS to deal with the User Interface and User Experience parts of the project followed by the backend framework Django that provided the suitable environment for development of the RESTful APIs for the project followed by the Redis as database that was the backbone of the entire project as its interaction had to transformed from CLI to GUI.

The instruction flow

- User clicks provides some input from the frontend.
- User then performs some action on the frontend that triggers some event.
- The event decides if an API call is required for the task or not.
- The API is called.
- The corresponding Transaction Script to the API runs at the back end.
- Transaction Script could have two tasks, either to deal with the data source i.e Redis database or do some calculations, in the end it returns the requested response.
- The front end receives the output from the back end and handles it accordingly
- If the output received from the backend is an error, then front end accordingly handles the error in a way that it doesn't affects the User Experience negatively.
- If the output received is to be displayed to the user then it is rendered accordingly and sent to the user

Following architectural model visualizes the workflow of the system design.

Architectural Diagram

Following diagram describes the architectural overview of the system design.

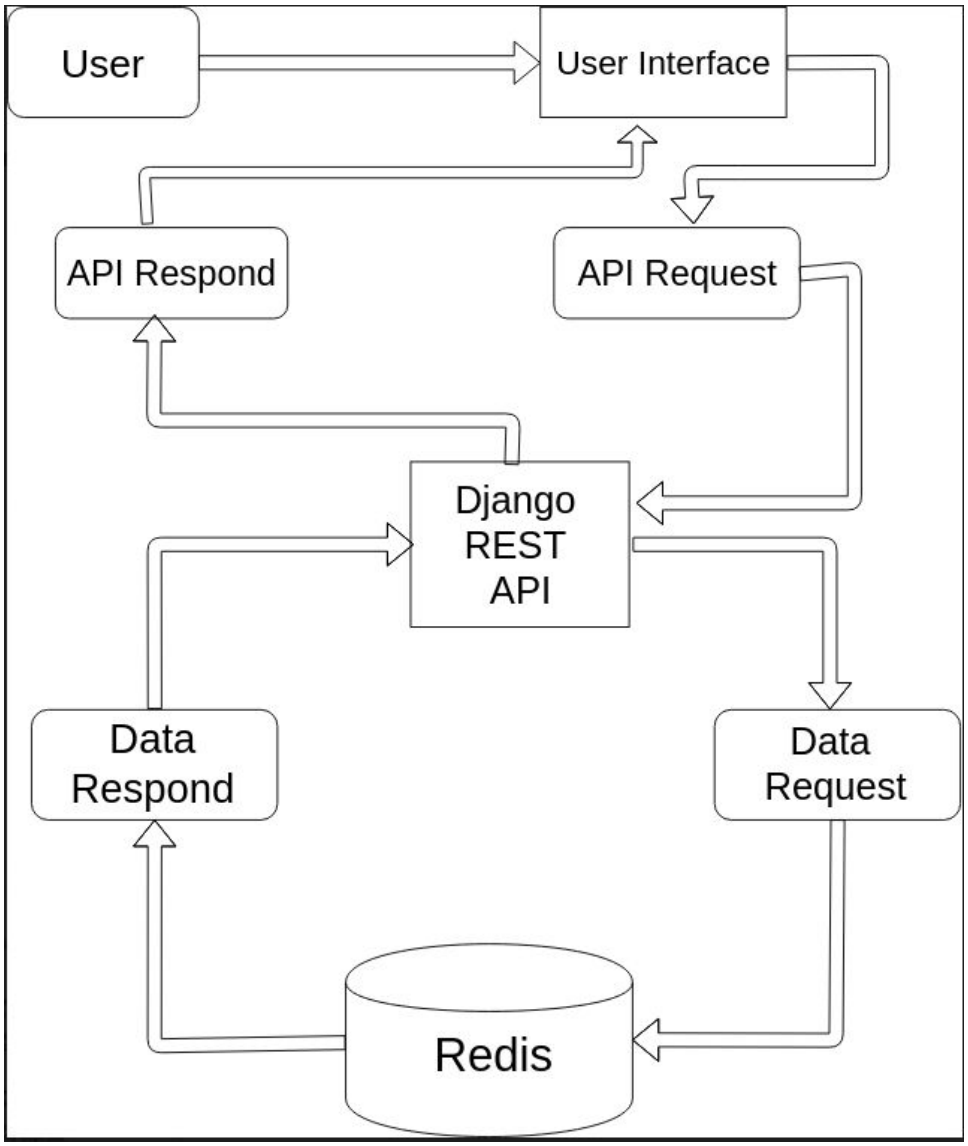


Figure 5

3.2 Front End

To handle the front end flow, ReactJS framework was used. There were other options available such as AngularJS and VueJS, but due to complexity of AngularJS and unfamiliarity of VueJS, A light weight familiar framework i.e ReactJS was chosen.

The frontend was divided into two main components, **The Search and Create Panel** and **The Display Panel.**

3.2.1 The Search and Create Panel

Search panel was used for searching and Create Panel for creating new JSON key-value objects. It provided the user with the search result field that yielded all the keys present in the Redis database depending upon the search pattern provided by the user. Whereas the create part included two fields - **The Key Field, The Value Field.**
The Key Field takes the user input for the Redis Key that is supposed to map to the JSON object to be provided by the user.
The Value Field takes the user input for the JSON object to any level of indentation.

The search field makes an API call to the backend with the user provided pattern as a parameter. As the response arrives from the backend, the Search Result Display panel is populated with all the keys present in the Redis database matching the user provided pattern.
Whereas, in the creation field, as soon as the user submits the key-value pair, it is first sanitized by the frontend and then it makes and API call for insertion. Front end waits for the response from the back end, if the insertion is successful it updates the search panel, otherwise alerts the user that some defined error has occurred.

3.2.2 The Display Panel

Display Panel provided two types of views - **[JSON View, Graph View].**
The main focus of the entire project was to make it easier for the user to interact with the data that is being displayed in the Display Panel. The easier interaction implies that it becomes easier for the user to find the keys or nodes to modify or navigate through the data i.e cascading from parent key to the child key and it's values or moving from one key to another. Whereas in the Graph display, it motive was to make the node searching easier for the user and view relationships between the nodes in realtime and get the data of the node as well as modify it on the go.

3.2.2.1 Viewing the data

ReJSON

- The design is highly inspired from the works of compassDB.
- It provides the user with the adequate frontend that doesn't overwhelms the user with the entire data displayed at once.
- It moves in a layer-by-layer fashion, when the user first clicks the main Redis Key, it displays the first 20 keys in the first level of the object.
- Now user has the option to search for the key and reach directly for it or navigate to the desired key-value pair by clicking the key on the current level that reveals the further level keys that were encapsulated by the outer key.

RedisGraph

- The design is inspired from the design of Neo4j.
- It provides the user with the adequate frontend that results in the searched node/root node with maximum of 5 relationships at a time.
- User can navigate from node to node by expanding the graph and exploring other relationships with the current node.

3.2.2.2 Creating Data

ReJSON

- The key is provided in the key input field.
- The JSON object string is provided in the value field.
- The ReactJS typecasts the JSON object string into a JSON object.
- If this doesn't returns any error, the string is passed to along with and insertion API call, else the user is alerted regarding the same.

RedisGraph

- The user first defines the view in the display panel as the Edit View.
- Then user click to create a node.
- Now user is displayed a modal where user can provide any label and properties to the node.
- Similarly user can connect various nodes using an edge.
- As soon as n edge connects with exactly two nodes, it displays a modal that allows the user to input the information regarding the relationship between the two nodes.
- The input information contains the details about the label and the properties of the edge or the relationship.

3.2.3 Role of p5JS

- This library was used to successfully implement the concept of the RedisGraph to make the interaction with the graph data stored in Redis as seamless and flexible as possible.
- The p5JS component was divided in to two classes mainly the Node class that deals with the interaction of user with a node in the graph database and an Edge class that deals with the interaction of an edge or a relationship in the graph database.

3.2.4 System Design Issues

- The primary concern was to embed the p5JS code with the ReactJS.
- The secondary concern was to make the interaction as seamless, interactive and as flexible as possible.
- The tertiary concerns include, the deployment of the app in an SPA.

3.2.5 Solutions to the System Design Issues

- To embed the p5JS code, a wrapper class was created to embed the p5JS as a component to the React.
- The secondary issue was taken care of by the efficient UX design.
- The tertiary concern was taken care off by Webpack. Webpack was used to create a single bundle out of the ReactJS code. Therefore all the components were crunched into a single file. This helped in maintaining the main aspect of single deployment for the SPA.

3.3 Back End

To handle the back end flow, Django was chosen as a framework as it provided the create RESTful APIs comparatively fast. Along with the RESTful API creation, it was also used to deal with all the business logic and the Redis Database.

The back end was mainly implemented as a Model-View design, where the view was taking care of all the incoming requests i.e the APIs. To handle the business logic another layer was introduced that dealt with the Redis interaction part.

All the required unit test cases were created in order to make sure that adding/updating the functionality or API wouldn't affect the current workflow.

CHAPTER – 4

PERFORMANCE ANALYSIS

It gives an insight in the the breakdown of the entire application into various models that take care of the various parts and phases of the application independent of the other models in the design. The aim was to make the application as modular as possible as well as not compromising with the performance at the same time.

4.1 RDBTools

A product of HashedIn technologies for which the project RDBVIZ was developed to be used as a plugin to provide support for the non-native data types specifically ReJSON and RedisGraph module.

4.1.1 Establishing connection

The User Interface of the RDBTools provided the functionality to establish a connection to an already existing Redis Server/Database. Once this connection is established user can get a GUI look of the connected Redis Server.

4.1.2 ReJSON Module Support

The developed ReJSON module provided the ability to perform CRUD operations on the Redis Database in context to a JSON object. Prior to the support of the ReJSON module, a JSON format data was store in a string format. The issue with the string format was that, if there was any update, the entire string was required to be replaced in the database. It is no issue until we are dealing with the Redis Clusters dealing with Gigabytes of data every minute, then replacing the entire string was not the suitable operation as it contributed to the overhead to already increasing data.

ReJSON provided the ability to update the specific Key-Value pairs rather than updating the entire string, so it was a boost to the application and decreased the computation time significantly.

Following image visualizes the ReJSON support for the product.

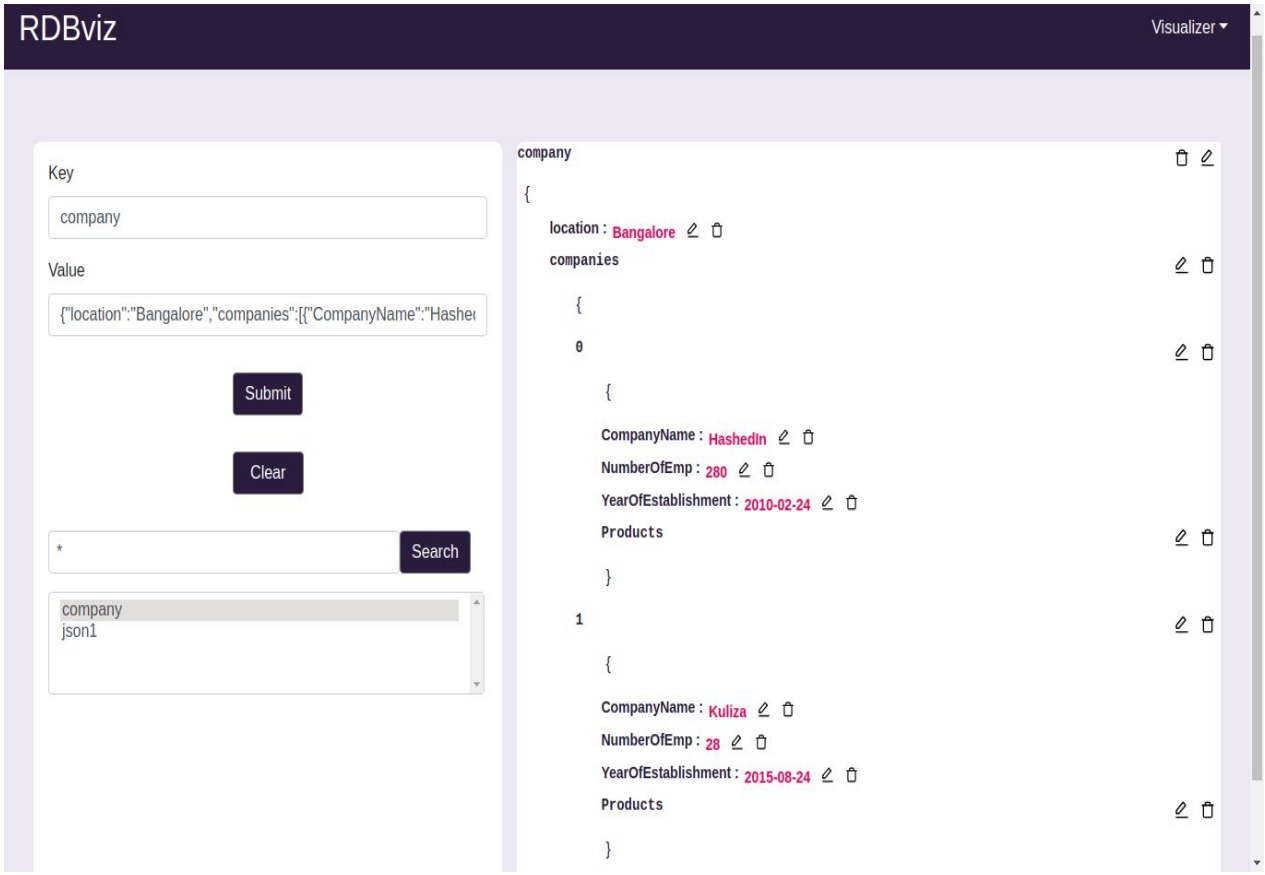


Figure 6

4.1.2 RedisGraph Module Support

As of now, officially, there is no other data visualization tool for visualizing the graph data stored in the the Redis. This was the major challenge as it was required to create something from scratch that hasn't been already done before.

Earlier there was no support for graph data type, but with the integration of the visualization tool for the Redis Graph module the RDBTools got the support for the Graph Data type.

Following image visualizes the Redis Graph support for the product.

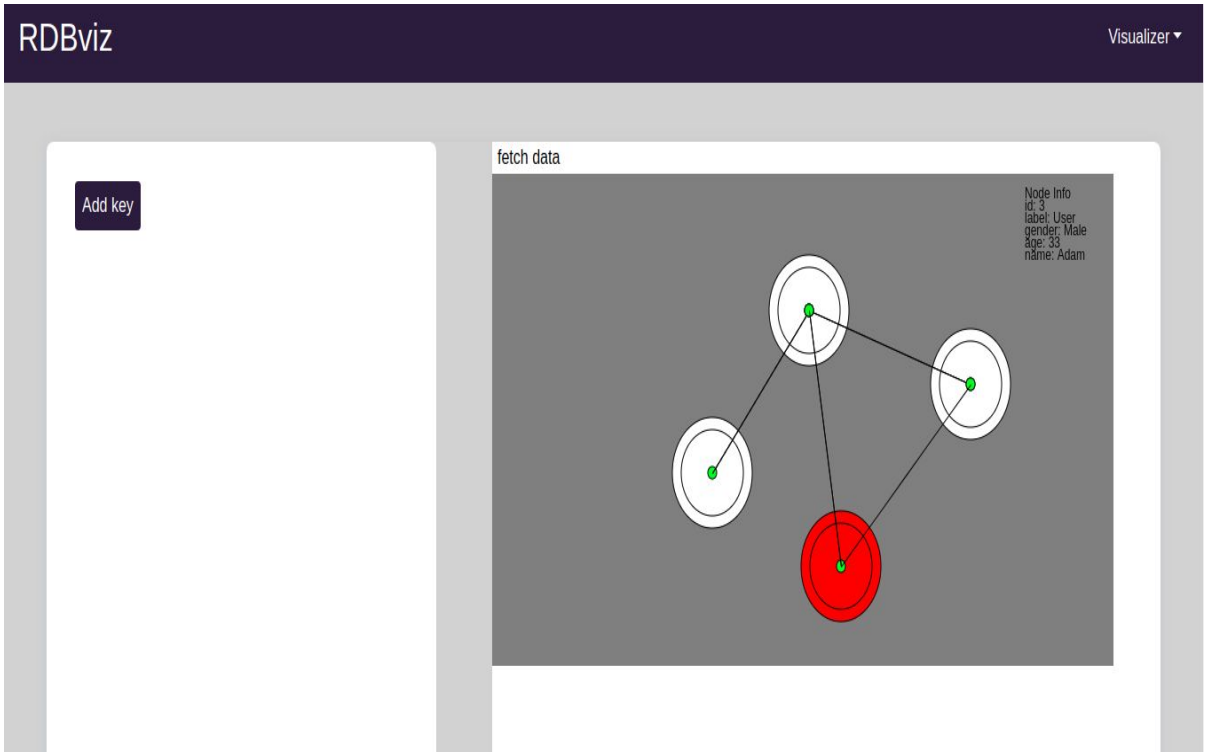


Figure 7

CHAPTER – 5

CONCLUSIONS

Working on a product provided me with an insight to the development and maintenance of a product on the industrial basis. My learning about the development and lifecycle management of a product has been of a great use for my other projects within the company.

Initially there was no tool for visualizing the data in the Redis Server/Database or as it may be described in layman terms that there was no GUI support for the Redis. RDBTools started with this idea in mind, to provide a tool for the users so that it becomes easier for the developers to interact with their data that is stored in the Redis Database.

With my work being in the field of non-native Redis data types, I had to deal with the creation of GUI support for modules provided by the redis, which had been done before. This was a project of first of its kind therefore it proved to be a challenging project. Moreover I was made the team lead for the project. WIth three additional team members to manage and complete the project on time gave me an opportunity to look at the management aspect of the project as well. Being the team lead I learnt the task estimation and team management as I had to make sure that everyone was working efficiently individually and as well as a team. Being at this position I had to keep myself updated with the front end, back end, integration and deployment aspect of the project.

With nearing deadlines, I learnt the stress management along with the completion of the task within the delta time window of the deadline. So it was a learning experience not only in context to the technology but the management as well.

REFERENCES

[1] https://rdbtools.com/docs/ **[RDBTools Documentation]**

[2] https://docs.djangoproject.com/en/2.2/ **[Django Documentation]**

[3] https://reactjs.org/docs/getting-started.html **[ReactJS Documentation]**

[4] https://oss.redislabs.com/redisjson/ **[ReJSON Documentation]**

[5] https://oss.redislabs.com/redisgraph/ **[RedisGraph Documentation]**

[6] https://p5js.org/get-started/ **[p5JS Documentation]**

[7] https://webpack.js.org/concepts **[Webpack Documentation]**