



College of Science and Technology
Rinchending: Bhutan

DBS101

Database Systems Fundamentals

SS(2024)

Practical{8} Report

Submitted By;

Student Name : Tashi Penjor

Enrollment No.: 02230306

Programme : BESWE

Date : 24/04/2024



Royal University of Bhutan



འབྲུག་རྒྱལ་ཁོན་གཙུག་ལག་སློབ་ཤེེ།

College of Science and Technology Rinchending: Bhutan



Table of content

SL.No	Topics	Page number
1	Guided session	2 - 5
2	Conclusion	6
3	Redis Pub/Sub	7-8
4	Redis Message Broker	9-12
5	Redis caching	12-13

Topic : Guided session

Task 1 : In this task the command is given to store the data in the Redis. Where the hash is named student and creates the entries for multiple students each unique with ID and each student's entry stores details including name, age, grade, GPA, attendance, address, contact number and the email address.

```
Administrator: Command Prompt - redis-cli
ity" contact "123-456-7890" email "john.doe@example.com"
(integer) 8
127.0.0.1:6379> HSET student:002 name "Alice Smith" age 16 grade "10th" GPA 3.5 attendance 92 address "456 Oak Ave
, Town" contact "987-654-3210" email "alice.smith@example.com"
(integer) 8
127.0.0.1:6379> HSET student:003 name "Michael Johnson" age 18 grade "12th" GPA 4.0 attendance 98 address "789 Elm
St, Village" contact "555-123-4567" email "michael.johnson@example.com"
(integer) 8
127.0.0.1:6379> HSET student:004 name "Emily Brown" age 17 grade "11th" GPA 3.7 attendance 94 address "101 Pine Rd
, County" contact "222-333-4444" email "emily.brown@example.com"
(integer) 8
127.0.0.1:6379> HSET student:005 name "David Wilson" age 16 grade "10th" GPA 3.2 attendance 90 address "777 Cedar
Ln, Suburb" contact "777-888-9999" email "david.wilson@example.com"
(integer) 8
127.0.0.1:6379>
```

Task 2 : In this the command is given to retrieve the specific data from the hash. The command uses HMGET keyword to retrieve the specific data. If the data is not stored in the hash, a null value is returned.

```
127.0.0.1:6379> HMGET student:001 name GPA
1) "John Doe"
2) "3.8"
127.0.0.1:6379>
```

Results: 25. Scanned 25 / 25

jobQueue	16%	4
leaderboard	4%	1
session	60%	15
student	20%	5

HASH	001	No limit	206 B
HASH	002	No limit	212 B
HASH	003	No limit	222 B
HASH	004	No limit	214 B
HASH	005	No limit	217 B

HASH student:001

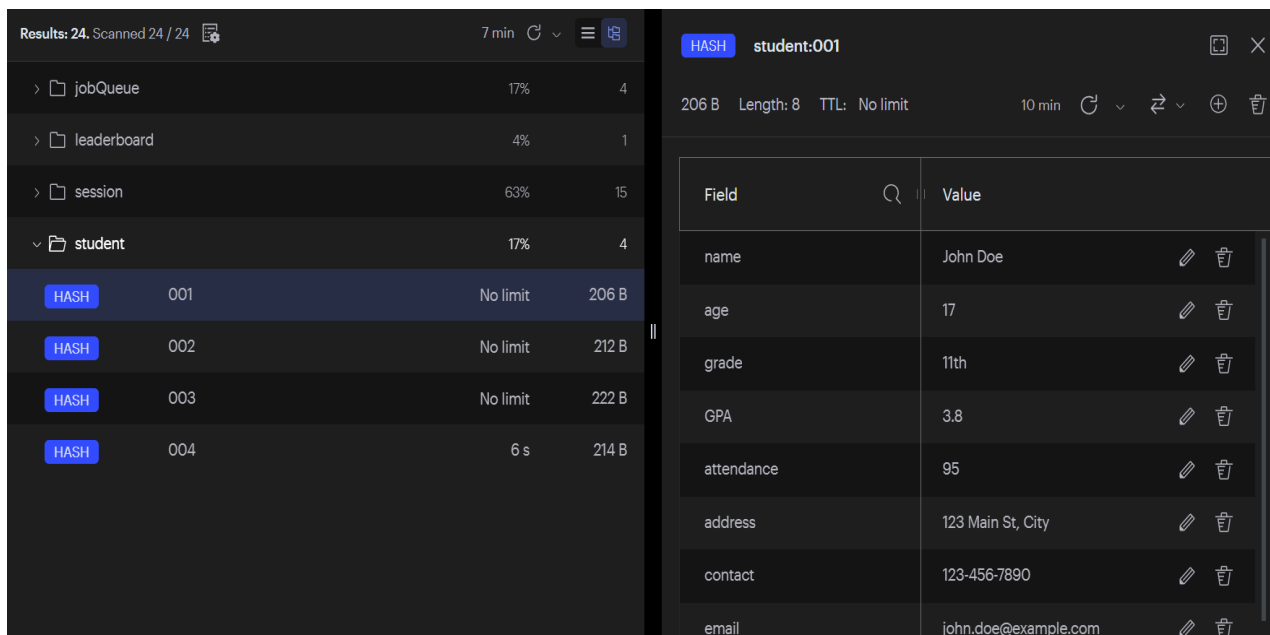
206 B Length: 8 TTL: No limit

Field	Value
name	John Doe
age	17
grade	11th
GPA	3.8
attendance	95
address	123 Main St, City
contact	123-456-7890
email	john.doe@example.com

Task 3: In this task the command given is to manipulate the data in the hash structure where it updates the GPA for the student ID 001 and it updates the attendance for the same student. On top of that it also deletes the data for the student ID 005.

```
127.0.0.1:6379> HSET student:001 GPA 3.7
(integer) 0
127.0.0.1:6379> HMSET student:001 GPA 3.7 attendance 95
OK
127.0.0.1:6379> FT.SEARCH idx:student "*" LIMIT 0 0
(error) ERR unknown command `FT.SEARCH`, with args beginning with: `idx:student`, `*`, `LIMIT`, `0`, `0`,
127.0.0.1:6379> DEL student:005
(integer) 1
127.0.0.1:6379> FT.SEARCH idx:student 005 RETURN 2 name grade
(error) ERR unknown command `FT.SEARCH`, with args beginning with: `idx:student`, `005`, `RETURN`, `2`, `name`, `grade`,
127.0.0.1:6379> EXPIRE "student:004" 30
(integer) 1
127.0.0.1:6379> FT.SEARCH idx:student "David Wilson" RETURN 2 name grade
(error) ERR unknown command `FT.SEARCH`, with args beginning with: `idx:student`, `David Wilson`, `RETURN`, `2`, `name`, `grade`,
127.0.0.1:6379>
```

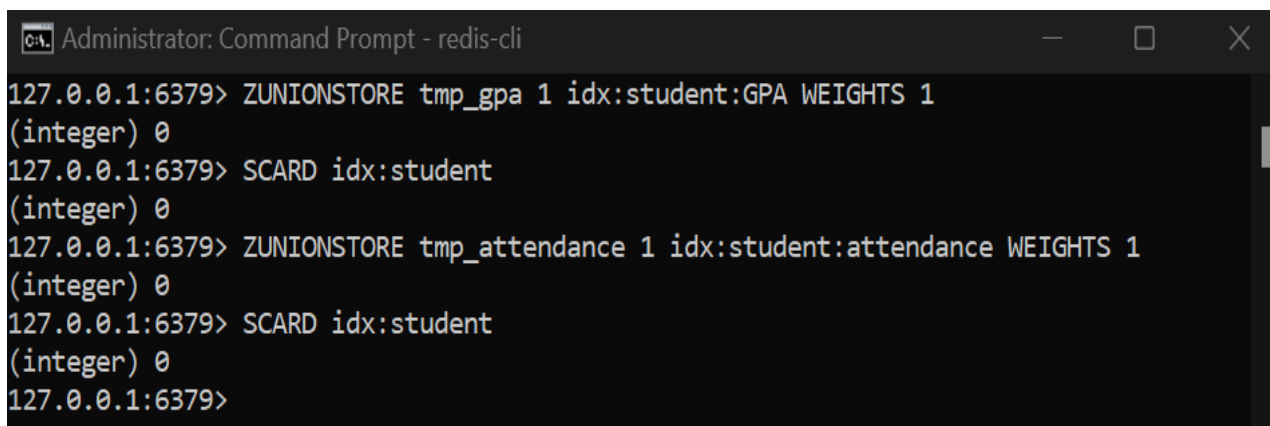
After manipulating the data for student ID 004 the expiration time is set for 30 seconds. So after 30 seconds the data associated with the student ID 004 will be automatically removed from the database.



The screenshot shows the Redis GUI interface. On the left, a tree view displays the database structure with folders for jobQueue, leaderboard, session, and student. The 'student' folder is expanded, showing a list of student records. The record for student ID 004 is highlighted, showing a TTL of 6 s and a size of 214 B. On the right, the details for 'student:001' are displayed, showing a size of 206 B, length of 8, and TTL of No limit. The details table lists fields and their values:

Field	Value
name	John Doe
age	17
grade	11th
GPA	3.8
attendance	95
address	123 Main St, City
contact	123-456-7890
email	john.doe@example.com

Task 4 : In this task the command given creates a temporary sorted set using the ZUNIONSTORE tmp_gpa and tmp_attendance.



```
Administrator: Command Prompt - redis-cli
127.0.0.1:6379> ZUNIONSTORE tmp_gpa 1 idx:student:GPA WEIGHTS 1
(integer) 0
127.0.0.1:6379> SCARD idx:student
(integer) 0
127.0.0.1:6379> ZUNIONSTORE tmp_attendance 1 idx:student:attendance WEIGHTS 1
(integer) 0
127.0.0.1:6379> SCARD idx:student
(integer) 0
127.0.0.1:6379>
```

Task 5 : In this task the command given is to manipulate and analyze the data stored in the hash set by calculating the average GPA, calculating the average attendance and dividing the sum of attendance percentages by the number of students.

```
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>redis-cli
127.0.0.1:6379> ZUNIONSTORE tmp_gpa 1 idx:student:GPA WEIGHTS 1

(integer) 0
127.0.0.1:6379> SCARD idx:student
127.0.0.1:6379> ZUNIONSTORE tmp_attendance 1 idx:student:attendance WEIGHTS 1
(integer) 0
127.0.0.1:6379> SCARD idx:student
(integer) 0
127.0.0.1:6379> ZCOUNT idx:student:grade 9 9
(integer) 0
127.0.0.1:6379> ZCOUNT idx:student:grade 10 10
(integer) 0
127.0.0.1:6379> ZCOUNT idx:student:grade 11 11
(integer) 0
127.0.0.1:6379> ZCOUNT idx:student:grade 12 12
(integer) 0
127.0.0.1:6379> |
```

Task 6 : In this task the command given is to manage the geospatial data. Redis stores geospatial data using the sorted set as the underlying data structure, with encoding and decoding of the location to allow users for efficient indexing, searching and sorting by location.

```
Microsoft Windows [Version 10.0.22621.3296]
127.0.0.1:6379> GEOADD idx:student:location 13.361389 38.115556 student_id1 151.2094 33.8650 student_id2
127.0.0.1:6379> GEORADIUS idx:student:location longitude latitude radius m
127.0.0.1:6379> GEORADIUS idx:student:location 13.361389 38.115556 200 km WITHCOORD WITHDIST
1) 1) "student_id1"
   2) "0.0001"
   3) 1) "13.36138933897018433"
127.0.0.1:6379> GEORADIUS idx:student:location 13.361389 38.115556 200 m WITHDIST
1) 1) "student_id1"
127.0.0.1:6379> GEORADIUS idx:student:location 13.361389 38.115556 200 m COUNT 5
127.0.0.1:6379> GEORADIUS idx:student:location 13.361389 38.115556 200 m SORT ASC
```

College of Science and Technology Rinchending: Bhutan

Conclusion :

The tasks outlined involve various operations with Redis, a powerful in-memory data structure store, used as a database, cache, and message broker. Here's a summary of each task:

Storing Data in Redis: The first task involves using the HSET command to store student data in a hash named student. Each student is uniquely identified by an ID, and their details include name, age, grade, GPA, attendance, address, contact number, and email address.

Retrieving Specific Data: The second task requires using the HMGET command to retrieve specific data from the hash. If the data is not present, a null value is returned.

Manipulating Data: This task involves updating the GPA and attendance for student ID 001 and deleting the data for student ID 005. Additionally, it sets an expiration time of 30 seconds for student ID 004's data, after which it will be automatically removed from the database.

Creating a Temporary Sorted Set: The fourth task uses the ZUNIONSTORE command to create a temporary sorted set named tmp_gpa and tmp_attendance, combining data from two existing sorted sets.

Analyzing Data: The fifth task involves manipulating and analyzing data stored in the hash set by calculating the average GPA and attendance, and dividing the sum of attendance percentages by the number of students. This requires extracting all relevant data, performing calculations, and possibly scripting or application logic to automate these operations.

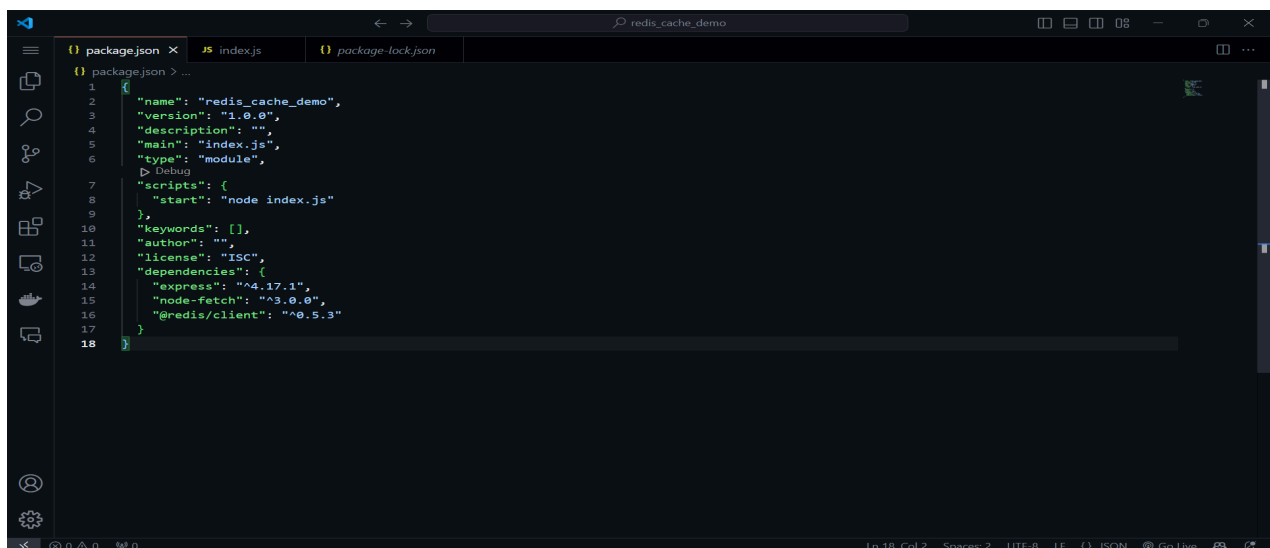
Managing Geospatial Data: The final task focuses on managing geospatial data using Redis's sorted set data structure. This involves encoding and decoding location data for efficient indexing, searching, and sorting by location.

College of Science and Technology

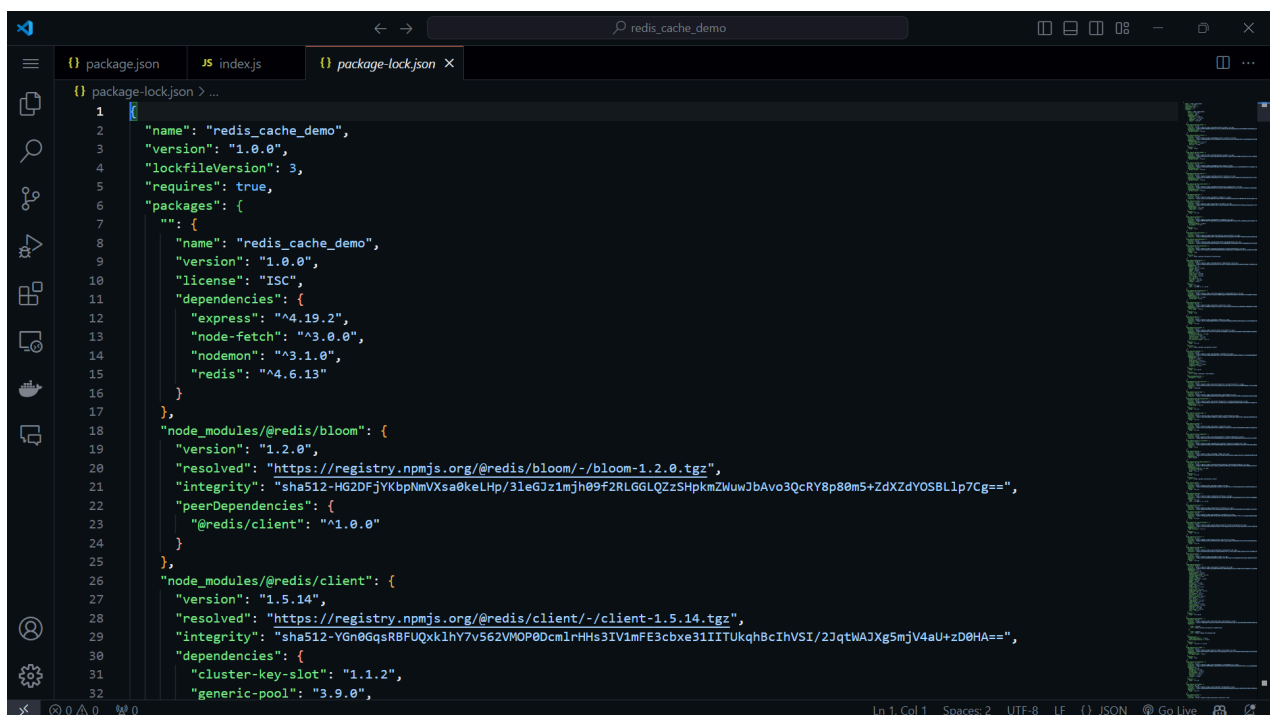
Rinchending: Bhutan

Redis Pub/Sub

Redis Pub/Sub is a messaging system that allows clients to publish messages to channels and subscribe to those channels to receive messages. This system is part of Redis's functionality as a message broker, supporting the publish/subscribe pattern for real-time communication between components of an application.



```
1 {
2   "name": "redis_cache_demo",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "type": "module",
7   "scripts": {
8     "start": "node index.js"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "express": "^4.17.1",
15    "node-fetch": "^3.0.0",
16    "redis/client": "^0.5.3"
17  }
18 }
```



```
1 {
2   "name": "redis_cache_demo",
3   "version": "1.0.0",
4   "lockfileVersion": 3,
5   "requires": true,
6   "packages": {
7     "": {
8       "name": "redis_cache_demo",
9       "version": "1.0.0",
10      "license": "ISC",
11      "dependencies": {
12        "express": "^4.19.2",
13        "node-fetch": "^3.0.0",
14        "nodemon": "^3.1.0",
15        "redis": "^4.6.13"
16      }
17    },
18    "node_modules/@redis/bloom": {
19      "version": "1.2.0",
20      "resolved": "https://registry.npmjs.org/@redis/bloom/-/bloom-1.2.0.tgz",
21      "integrity": "sha512-HG2DFjYKbpNmVXsa0keLHp/3leGJz1mjH09f2RLGGLQZzSHpkmZMwJbAvo3QcRY8p88m5+zDxDyOSBL1p7Cg==",
22      "peerDependencies": {
23        "@redis/client": "^1.0.0"
24      }
25    },
26    "node_modules/@redis/client": {
27      "version": "1.5.14",
28      "resolved": "https://registry.npmjs.org/@redis/client/-/client-1.5.14.tgz",
29      "integrity": "sha512-YGn0QsRBFUQxklhY7v562VMOP0Dcm1rHHs3IV1mFE3cbxe31IITUkqh8cIhVSI/23qtWAXJg5mjV4aU+zD0HA==",
30      "dependencies": {
31        "cluster-key-slot": "1.1.2",
32        "generic-pool": "3.9.0",

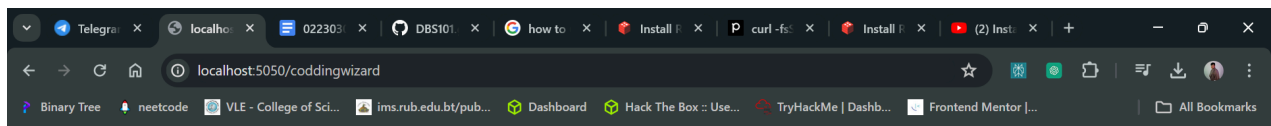
```




College of Science and Technology

Rinchending: Bhutan

```
package.json JS index.js x package-lock.json
JS index.js > ...
1 import express from 'express';
2 import { createClient } from 'redis';
3 import fetch from 'node-fetch';
4
5 const PORT = process.env.PORT || 5050;
6 const REDIS_PORT = process.env.REDIS_PORT || 6379;
7 const client = createClient({ port: REDIS_PORT });
8
9 const app = express();
10
11 app.get('/:username', getRepos);
12
13 async function setResponse(username, problemsSolved, responseTime) {
14   return `<h2>${username} has solved ${problemsSolved} problems on LeetCode</h2>
15     <p>Server response time: ${responseTime} ms</p>`;
16 }
17
18 async function getRepos(req, res, next) {
19   try {
20     console.log('Fetching Data...');
21     const { username } = req.params;
22     const startTime = Date.now();
23
24     const cachedData = await client.get(username);
25     if (cachedData) {
26       const problemsSolved = parseInt(cachedData);
27       const responseTime = Date.now() - startTime;
28       const response = await setResponse(username, problemsSolved, responseTime);
29       res.send(response);
30     } else {
31       const response = await fetch('https://leetcode-api-faisalshohag.vercel.app/' + username);
32       const data = await response.json();
```



codingwizard has solved 35 problems on LeetCode

Server response time: 2 ms

College of Science and Technology

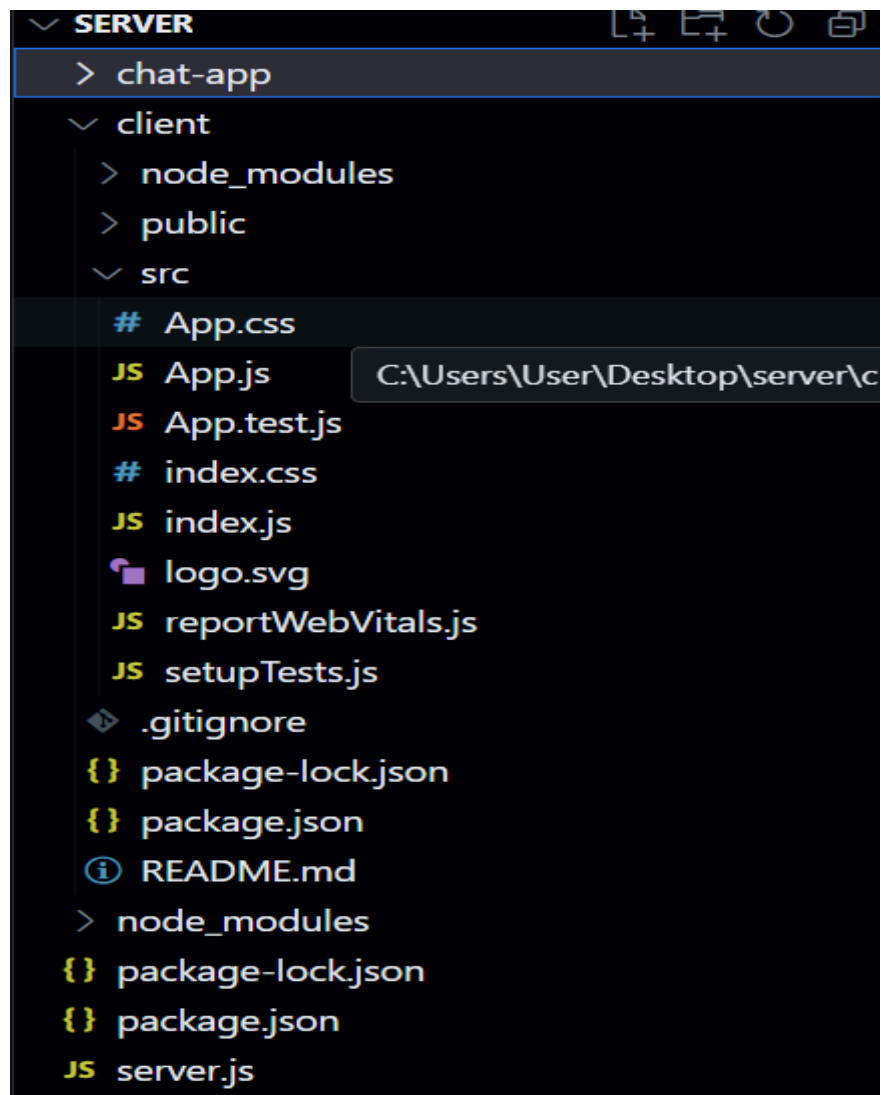
Rinchending: Bhutan

Topic : Redis Message Broker

A Message broker is an application that enables services to communicate with each other and share information.

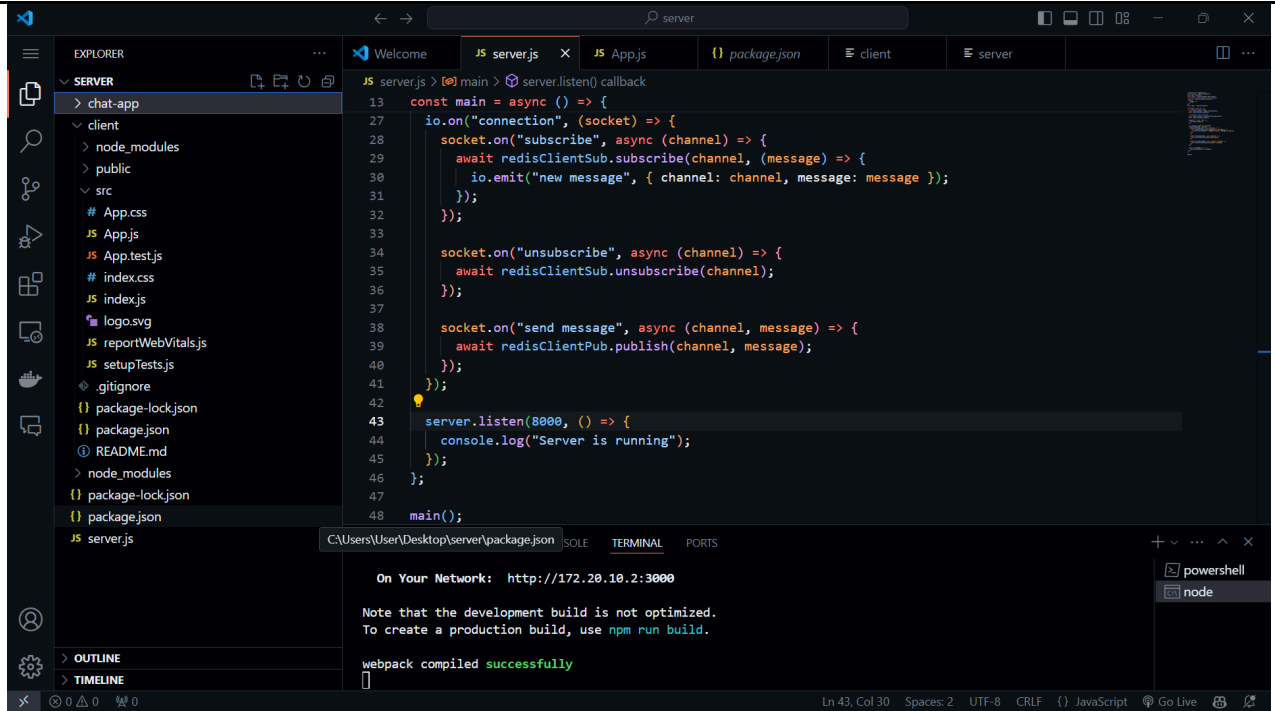
Now to implement a message broker using redis pub/sub for a simple chat application see the following steps.

The whole steps include setting up the server and installing the required independences.



College of Science and Technology

Rinchending: Bhutan



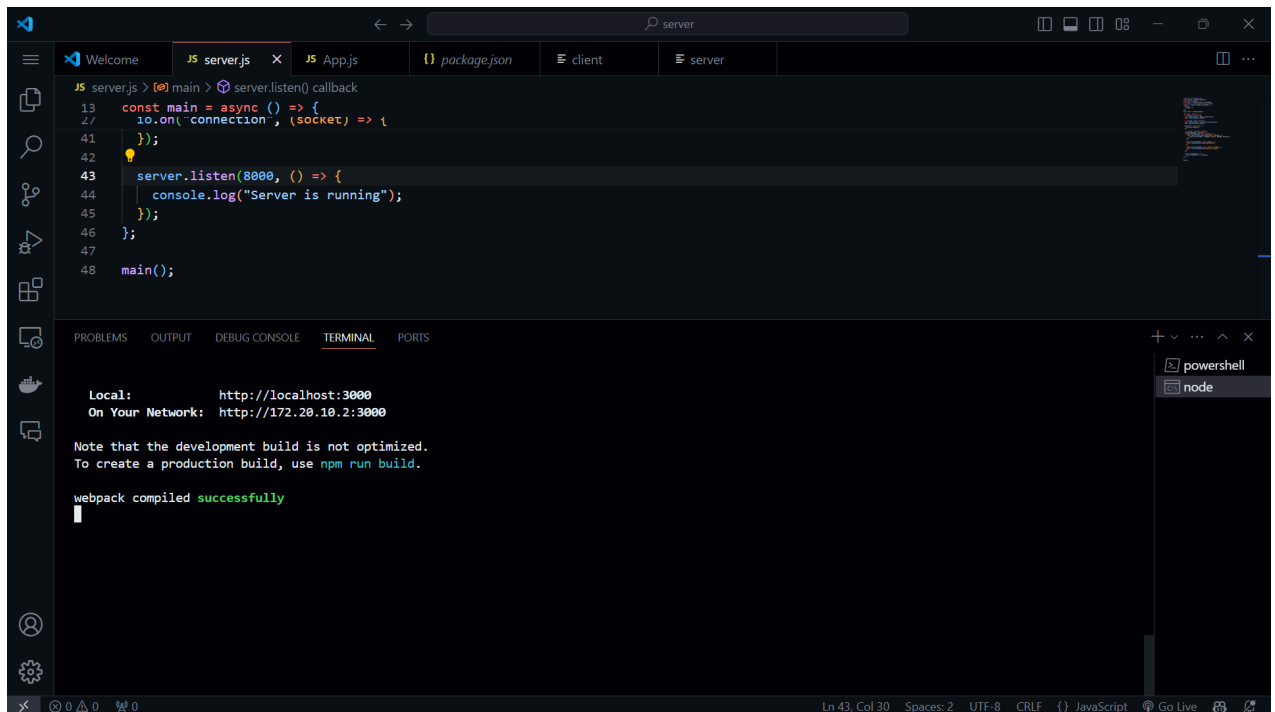
```

13  const main = async () => {
27    io.on("connection", (socket) => {
28      socket.on("subscribe", async (channel) => {
29        await redisClientSub.subscribe(channel, (message) => {
30          io.emit("new message", { channel: channel, message: message });
31        });
32      });
33    });
34    socket.on("unsubscribe", async (channel) => {
35      await redisClientSub.unsubscribe(channel);
36    });
37    socket.on("send message", async (channel, message) => {
38      await redisClientPub.publish(channel, message);
39    });
40  });
41  });
42  server.listen(8000, () => {
43    console.log("Server is running");
44  });
45  };
46  };
47  };
48  main();
  
```

On Your Network: <http://172.20.10.2:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled **successfully**



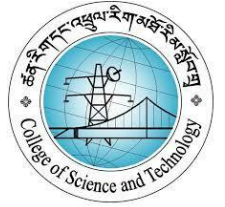
```

13  const main = async () => {
27    io.on("connection", (socket) => {
41    });
42  });
43  server.listen(8000, () => {
44    console.log("Server is running");
45  });
46  };
47  };
48  main();
  
```

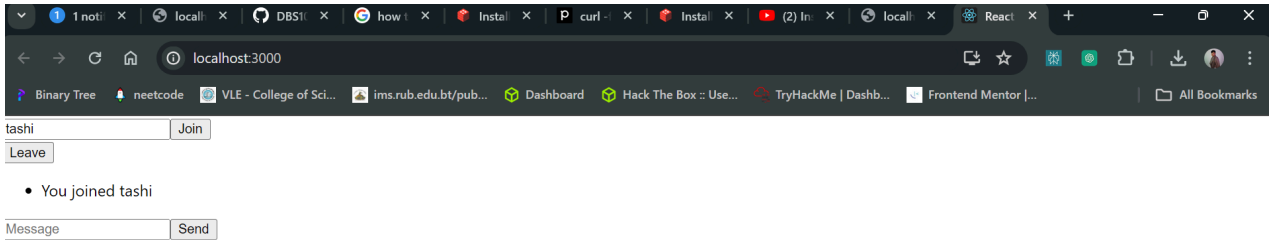
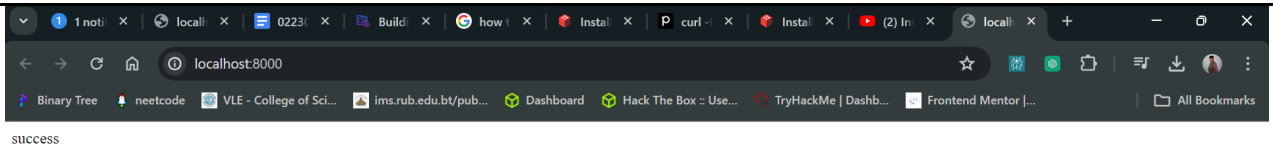
Local: <http://localhost:3000>
On Your Network: <http://172.20.10.2:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled **successfully**



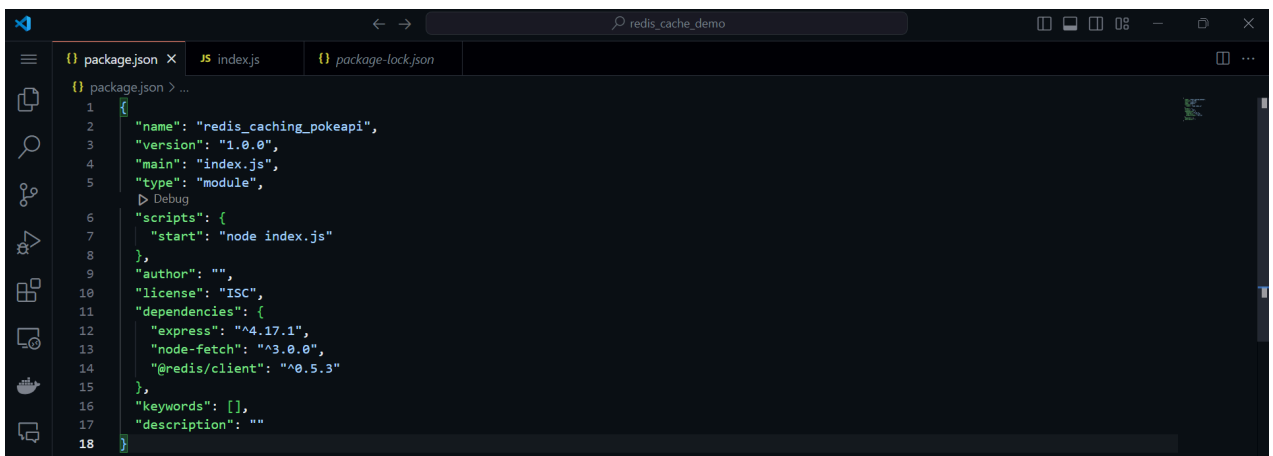
College of Science and Technology Rinchending: Bhutan



Redis caching

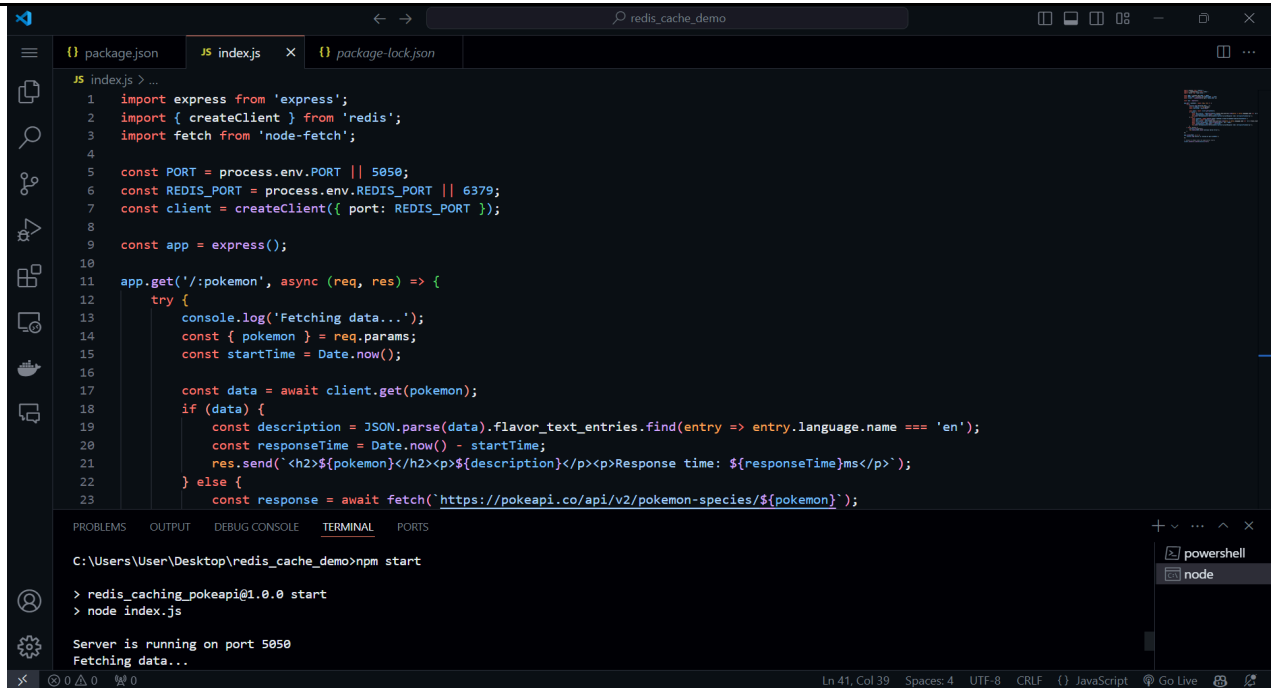
In this task it includes the implementation of caching the poke api using the redis.

The steps include creating a new package.js for the Pokemon API and a different server for the Pokemon API to fetch the details. Finally when I ran the server the Pokemon name and the details was displayed.



College of Science and Technology

Rinchending: Bhutan



```

1 import express from 'express';
2 import { createClient } from 'redis';
3 import fetch from 'node-fetch';
4
5 const PORT = process.env.PORT || 5050;
6 const REDIS_PORT = process.env.REDIS_PORT || 6379;
7 const client = createClient({ port: REDIS_PORT });
8
9 const app = express();
10
11 app.get('/:pokemon', async (req, res) => {
12   try {
13     console.log('Fetching data...');
14     const { pokemon } = req.params;
15     const startTime = Date.now();
16
17     const data = await client.get(pokemon);
18     if (data) {
19       const description = JSON.parse(data).flavor_text_entries.find(entry => entry.language.name === 'en');
20       const responseTime = Date.now() - startTime;
21       res.send(`<h2>${pokemon}</h2><p>${description}</p><p>Response time: ${responseTime}ms</p>`);
22     } else {
23       const response = await fetch(`https://pokeapi.co/api/v2/pokemon-species/${pokemon}`);

```

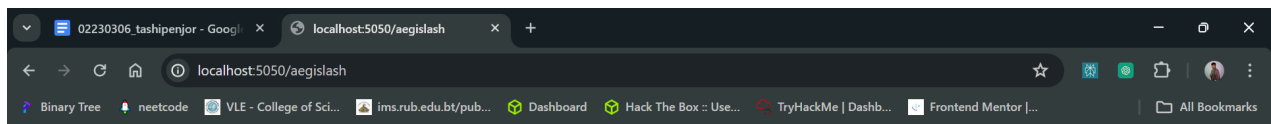
Terminal output:

```

C:\Users\User\Desktop\redis_cache_demo>npm start
> redis_caching_pokeapi@1.0.0 start
> node index.js

Server is running on port 5050
Fetching data...

```



aegislash

Generations of kings were attended by these Pokémon, which used their spectral power to manipulate and control people and Pokémon.

Response time: 514ms