This document displays different ways I would interact with my redis data structures.
NOTE: In my CRUD application, I only interacted with the data structures pertaining to the users and portfolios collection.

Caching User Data

HGETALL user:${userId}

Purpose:
Retrieve all fields and values of the hash stored at key user:${userId}. If the user data is cached, this command retrieves it, avoiding a database query.

Usage:
Used in the GET /users/:id endpoint to quickly fetch user data if it has been cached, significantly reducing response time and database load.

HSET user:${userId} field1 value1 field2 value2 ...

Purpose:
Set multiple fields to their respective values in the hash stored at key user:${userId}. This is used to cache user data right after fetching it from MongoDB.

Usage:
This command is executed when user data is fetched from MongoDB and not found in the cache, caching it for subsequent access.Found in app.js file in CRUD app.

Invalidating Cached Data

DEL user:${userId}

Purpose:
Remove the cached data at key user:${userId} when user data is updated or deleted.

Usage:
Ensures that any updates or deletions are consistent across the database and the cache by removing outdated cached data. Found in app.js file in CRUD app.

Caching Portfolio Data

GET portfolio:${userId}
SET portfolio:${userId} "JSON_string" EX 3600

Purpose:
GET is used to retrieve portfolio data from the cache. SET with EX option is used to cache this data with an expiration time, ensuring it doesn't become stale.

Usage:
These commands are utilized to manage caching for portfolio data associated with a user. Found in app.js file in CRUD app.

Invalidating Portfolio Data

DEL portfolio:${userId}

Purpose:
Invalidate the cached portfolio data when updates or deletions occur.

Usage:
Used post-update or deletion of a portfolio to ensure the cache does not hold outdated information, aligning the cache content with the database. Found in app.js file in CRUD app.

Maintain Real-Time Property Value Rankings

Purpose: Track and retrieve top properties by value in real-time.

Update Property Value: ZADD property_values 1200000 property101
Retrieve Top 10 Properties by Value: ZREVRANGE property_values 0 9 WITHSCORES
Remove Property from Rankings: ZREM property_values property101

Tenant Interaction Log

Purpose: Log recent tenant interactions for operational insights.

Log Interaction: LPUSH tenant_interactions:tenant456 "inquiry about lease renewal"
Retrieve Last 5 Interactions: LRANGE tenant_interactions:tenant456 0 -4
Delete Interaction History: DEL tenant_interactions:tenant456

Financial Report Quick Access

Purpose: Cache financial report summaries for fast access.

Cache Report Summary: HSET report_summary:report789 income 80000 expenses 50000 summary "Net positive"
Get Report Summary: HGETALL report_summary:report789
Clear Report Cache: DEL report_summary:report789

Add/Update Portfolio in Leaderboard

Purpose: Maintain an up-to-date leaderboard of portfolios sorted by total value.

ZADD portfolio_leaderboard {total_value} {portfolio_id}
ZADD portfolio_leaderboard 1500000 portfolio101
ZADD portfolio_leaderboard 2000000 portfolio102

Retrieve Top N Portfolios

Purpose: Fetch the top portfolios for display on dashboards or reports.

ZREVRANGE portfolio_leaderboard 0 {0-9} WITHSCORES

Remove a Portfolio from Leaderboard

Purpose: Remove a portfolio from the leaderboard when it is deleted or no longer tracked.

ZREM portfolio_leaderboard {portfolio_id}
ZREM portfolio_leaderboard portfolio101

Redis usage in app.js:

```
app.get('/users/:id', async (req, res) => {
   const cacheKey = `user:${req.params.id}`;
   try {
       let user = await redisClient.hGetAll(cacheKey);
       if (Object.keys(user).length) {
           console.log('Retrieving from cache');
           return res.json(user);
       }
       user = await User.findById(req.params.id);
       if (!user) {
           res.status(404).send('User not found');
       } else {
           await redisClient.hSet(cacheKey, user.toObject());
           res.json(user);
       }
   } catch (err) {
```

```
        res.status(500).json(err);
    }
});

app.put('/users/:id', async (req, res) => {
    try {
        const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true
});
        if (!user) {
            res.status(404).send('User not found');
        } else {
            await redisClient.del(`user:${req.params.id}`);
            res.json(user);
        }
    } catch (err) {
        res.status(400).json(err);
    }
});

app.delete('/users/:id', async (req, res) => {
    try {
        const user = await User.findByIdAndDelete(req.params.id);
        if (!user) {
            res.status(404).send('User not found');
        } else {
            await redisClient.del(`user:${req.params.id}`);
            res.send({ message: 'User deleted successfully' });
        }
    } catch (err) {
        res.status(500).json(err);
    }
});

// Fetch portfolios by user_id
app.get('/portfolios/user/:userId', async (req, res) => {
    const cacheKey = `portfolio:${req.params.userId}`;  // Define a unique key for
caching
    try {
        // Try to get the portfolio data from Redis first
        let portfolios = await redisClient.get(cacheKey);
        if (portfolios) {
            console.log('Retrieving portfolios from cache');
```

```
            return res.json(JSON.parse(portfolios));  // Parse and return the cached
data
        }

        // If not in cache, fetch from MongoDB
        portfolios = await Portfolio.find({ user_id: req.params.userId });
        if (portfolios.length === 0) {
            res.status(404).json({message: 'No portfolios found for this user. Please
add a new portfolio.', canCreate: true});
        } else {
            await redisClient.set(cacheKey, JSON.stringify(portfolios), 'EX', 3600);
            res.json(portfolios);
        }
    } catch (err) {
        console.error('Error retrieving portfolios:', err);
        res.status(500).send({ message: "Error retrieving portfolios", error: err });
    }
});




app.put('/portfolios/:portfolioId', async (req, res) => {
    try {
        const portfolio = await Portfolio.findOneAndUpdate({ portfolio_id:
req.params.portfolioId }, req.body, { new: true });
        if (!portfolio) {
            return res.status(404).send({ message: 'Portfolio not found' });
        } else {
            // Invalidate the cache after updating
            const cacheKey = `portfolio:${portfolio.user_id}`;
            await redisClient.del(cacheKey);
            res.json(portfolio);
        }
    } catch (err) {
        console.error('Error updating portfolio:', err);
        res.status(500).send({ message: 'Error updating portfolio', error: err });
    }
});
```