Redis Project: The Tommies Idea Board

By *Luca Comba*

Date: 2025-05-04

Quick Links

- GitHub repository
- tommies.peeperone.com

Introduction

This project explores Redis, an in-memory data structure store and database. It's known for its speed and can be used for handling various data structures in a simple key-value database.

The project focused on learning Redis, its history, core commands, and common use cases. To put into practice the learned material, a web application called "Tommies Idea Board" was developed. The Tommies Idea Board API was built in Golang, while the front end application was written using JavaScript and Vue. The board application allows users (Tommies) to share ideas, which are then stored and retrieved from the Redis database.

Why Redis? Redis was chosen because of its high performance. The Redis official documentation is very easy to read and understand. Redis is an open-source project, making the project free to develop. Its support for various data structures like Hashes was also a good fit for storing the idea data structure.

In reality, I was very interested in learning Redis because of its popularity in the industry. Redis is widely known and utilized across industries.

Minimal Goal Understand the fundamentals of Redis (commands, data types, persistence) and build a basic CRUD (Create, Read, Update, Delete) application using Redis as the database.

Optimal Goal Deploy the "Tommies Idea Board" application with a functional front-end, allowing multiple users to post and view ideas stored in Redis. Explore more advanced features or data models with Redis if time permitted. I strognly believe that anyone can develop any application if they had the time and resources.

Exploring

My previous database experience was with relational databases like PostgreSQL or MySQL and with non-relational databases like DynamoDB. This project

was an opportunity to explore the NoSQL landscape, specifically key-value and in-memory databases.

Initial Ideas Initial ideas involved exploring different NoSQL databases, a database that I have never used before like Cassandra or Redis.

Final Decision I really wanted to build a real application, like a website, therefore I prefered to use a database that was easy to use and understand. In that case, Redis was the best option.

The Redis Twitter-Clone pattern example documentation was extremely helpful as it provided a clear example of how to use Redis for a real-world application.

Building

As previously mentioned, the Redis Twitter-Clone pattern example gave me a clear idea on how to use Redis, therefore I decided to follow the tutorial and started to build the REST API using Go.

When the API server is started, it initializes the Redis connection and sets up the API routes. The API has endpoints for creating and retrieving ideas, as well as storing key-value pairs. The API uses the go-redis library to interact with Redis.

The API implements the following endpoints:

Endpoint	Description
`GET /store/{key}`	Retrieve a value by its key.
`GET /idea/{id}`	Retrieve a specific idea by its ID. The response will be a JSON object representing the idea.
`GET /ideas`	Retrieve all ideas. The response will be a JSON array of ideas.
`POST /idea`	Add a new idea. The request json body should contain the new idea.

Redis Data Model The `/store` endpoint access the Redis key-value store which follows the model

```
r.Client.Set(r.Ctx, "foo", "bar", 0)
r.Client.Set(r.Ctx, "key", "value", 0)
```

Which can be demonstrated with

```
$ curl http://localhost:8080/store/foo
Value from Redis: bar%
```

The `/idea` endpoint access the Redis Hashes which follows the model

```
firstIdea := Idea{
    ID:          1,
    Title:       "First Idea",
    Description: "This is my first idea.",
    Writer:      "Luca",
    Tags:        []string{"idea", "redis"},
}
r.Client.Set(r.Ctx, "idea_uid", firstIdea.ID, 0)
r.AddIdea(firstIdea)
```

The idea can be retrieved with

```
$ curl http://localhost:8080/idea/1
{
    "id":1,
    "timestamp":"1745381047",
    "title":"First Idea",
    "description":"This is my first idea.",
    "writer":"Luca",
    "tags":["idea","redis"]
}%
```

Docker The API is containerized using Docker, while Redis provides its own docker image. The Docker Compose file orchestrates the API and Redis containers, allowing for easy deployment and management of the application.

Frontend The front end of the application is built using Vue.js. It provides a simple interface for users to view and add ideas. The front end communicates with the API to retrieve and store ideas in Redis.

The front end can be accessed at https://www.tommies.peeperone.com

Discovering

What Went Well The project provided practical experience with Redis. Thanks to this project I was able to start exploring and using Redis, although due to time constraints and limited resources I was not able to produce a fully functional application.

What I Learned Before this project, my understanding of Redis was very limited to the name and its popularity. This project significantly advanced my knowledge by requiring hands-on implementation.

Challenges The application needs new features such as an user authentication system, more complex querying, allowing user interactions such as liking ideas or commenting an idea, but due to time constraints I was not able to implement them. All of these feature would require a more complex data model and a deeper understanding of Redis.

Unit testing are not implemented in the API. Unit tests are important for ensuring the correctness and reliability of the code.

Class Topics

1. Non Relational Databases From the class lecture number 12, I learned about the differences between relational and non-relational databases. Non-relational databases are designed to handle unstructured data and provide flexibility in data modeling. Redis is a key-value store that falls under the category of non-relational databases. Redis does not require fixed schemas and does not rely on Foreign Keys or Joins.

2. Key-Value Stores In lecture number 12, it was mentioned the nature of key-value stores. Redis is a key-value store, which means that it stores data as a collection of key-value pairs. Each key is unique and maps to a specific value.

3. NoSQL vs SQL Lecture 12 also covered the differences between SQL and NoSQL databases. SQL databases are structured and use a fixed schema, while NoSQL databases like Redis are more flexible and can handle unstructured data. Redis is a NoSQL database that provides high performance and scalability. In fact, Redis uses its own set of commands and uses different data structures such as Strings, Hashes, Lists, Sets, and Sorted Sets.

4. In-Memory Databases During the course it was discussed how database are storing their data in disk (Lecture 7, Chapter 8). Redis is an in-memory database, which means that it stores data in RAM for fast access. This allows Redis to provide high performance and low latency for read and write operations. However, it also means that data is lost if the server crashes or is restarted unless persistence options are configured.

5. Database Clients and APIs In lecture 10, REST api were discussed. This project implements a RESTful API in Golang. The API provides endpoints for creating and retrieving ideas, as well as storing key-value pairs. The API uses the go-redis library to interact with Redis.

Correctness and Completeness

Sources are corretly linked.

The project's source code is hosted on github.com/lukfd/tommies-idea

The front-end web application is accessible at https://www.tommies.peeperone.com.