DEMO PROJECT

SHOPSYNC

April 5 2024

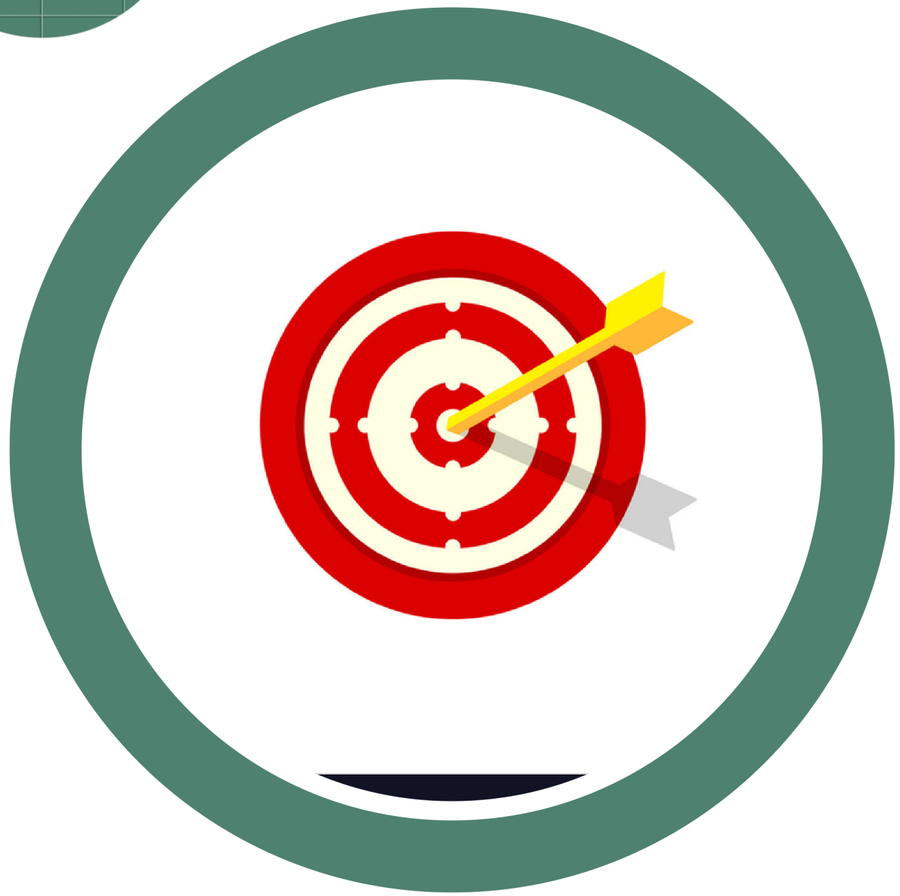
BY KARTHIKEYAN M

Project Overview

An interactive application that allows shopOwners to register and perform operations effectively and efficiently.

- INTRODUCTION
- PROJECT CORNERSTONES
- BASIC AUTH
- SHARDING
- INDEXING
- REDIS
- RATELIMIT

Introduction



Basic Auth - To apply basic authentication system to allow users to signup and login with email and password and give access token to access APIs

MongoDB - Leveraging MongoDB retrieval and insertion with sharding and indexing for faster access and efficient storage structure

Redis - A cache mechanism for fast retrieval and efficient processing of requests also used as session management

Efficient search - To efficiently search for shopOwners within certain distance and to find shopOwners of certain age within certain country with help of indexing

RateLimiting - To limit the number of requests and prevent DDoS attacks and reduce load on the server

Project Cornerstones

The below are the concepts used within the project to achieve the missions and use cases of the project

- **BASIC AUTH**
- **INDEXING**
- **SHARDING**
- **REDIS**
- **RATE LIMITING**
- **JWT TOKENS WITH RSA 256 ENCRYPTION**

Basic Auth

This allows the user to register in our database with email and password where email will be unique and password will be hashed

PASSWORD

Password being hashed with help of bcrpyt with 10 rounds of hash

EMAIL

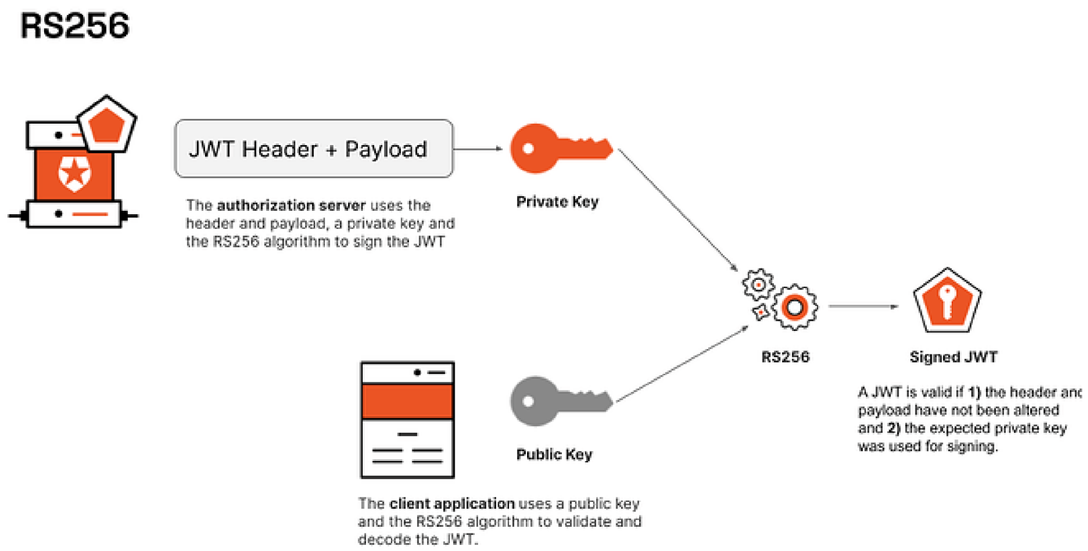
A valid email of type string to be provided that must be unique

RESULT

An access token will be provided with payload as userId (_id)

JWT Token

- **Digital Signature** - RS256 JWTs use digital signatures for authentication, ensuring data integrity and security.
- **Private Key Signing -** Tokens are signed with a private key, verifying authenticity.
- **Public Key Verification** - Verification is done using the public key, confirming validity without revealing sensitive data.
- **Header-Payload-Signature -** RS256 JWTs consist of three parts: Header, Payload, and Signature, ensuring tamper-proof authentication.
- **Enhanced Security -** RS256 provides robust security, mitigating risks associated with unauthorized access and tampering.



- **Standardization -** RS256 is a widely accepted standard algorithm, ensuring compatibility and trust across various platforms and frameworks.
- **Payload Flexibility -** JWT payload allows for flexible inclusion of custom claims, enabling versatile usage scenarios beyond authentication, such as authorization and information exchange.

INDEXING

- **Enhanced Query Performance -** Indexes in MongoDB significantly speed up query operations by efficiently locating specific data without scanning every document in a collection.
- **Support for Complex Queries -** MongoDB indexing supports complex queries, including text searches and queries on embedded documents and arrays, facilitating rich data exploration.
- **Efficient Sorting and Aggregation -** Indexes enable quick sorting of query results, allowing for fast data aggregation and analysis, crucial for reporting and analytics.
- **DataStructure** - B-Tree Indexing

SINGLE FIELD

Index with one field can be either ascending or descending

COMPOUND

Index with more than 1 field and order of field also matters - asc/desc

GEOSPATIAL

Works on coordinates with lat/lon for geography based searching

HASHED INDEX

A type of shard index that can be used as shard key for sharding purpose

Indexing Analysis

COLSCAN VS IXSCAN

Query Performance Summary

- **1** documents returned
- **100885** documents examined
- **30 ms** execution time
- **Is not** sorted in memory
- **0** index keys examined
- ⚠ No index available for this query.

Create Index | Refresh

Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	3.4 MB	18 (since Thu Apr 04 2024)	UNIQUE
age_1_country_-1	REGULAR	2.3 MB	2 (since Thu Apr 04 2024)	COMPOUND
country_1	REGULAR	2.3 MB	0 (since Thu Apr 04 2024)	
country_hashed	HASHED	2.4 MB	0 (since Thu Apr 04 2024)	
shopDetails.coordinates_2dsphere	GEOSPATIAL	2.3 MB	2 (since Thu Apr 04 2024)	

Query Performance Summary

- **1** documents returned
- **1** documents examined
- **0 ms** execution time
- **Is not** sorted in memory
- **1** index keys examined

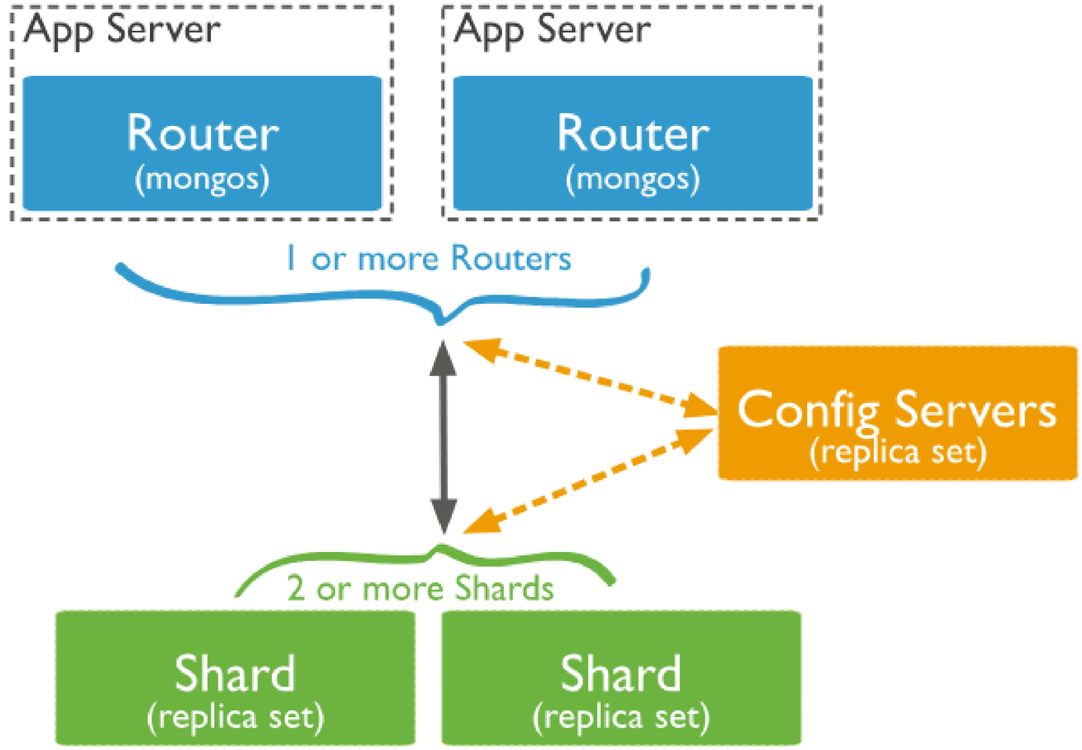
Query used the following index:

email ↑

Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	3.4 MB	18 (since Thu Apr 04 2024)	UNIQUE
age_1_country_-1	REGULAR	2.3 MB	2 (since Thu Apr 04 2024)	COMPOUND
country_1	REGULAR	2.3 MB	0 (since Thu Apr 04 2024)	
country_hashed	HASHED	2.4 MB	0 (since Thu Apr 04 2024)	
email_1	REGULAR	2.3 MB	2 (since Fri Apr 05 2024)	
shopDetails.coordinates_2dsphere	GEOSPATIAL	2.3 MB	2 (since Thu Apr 04 2024)	

SHARDING

- **Horizontal Scalability -** MongoDB sharding distributes data across multiple servers, allowing for horizontal scaling to accommodate large datasets and high traffic loads.
- **Automatic Data Distribution -** Sharding automatically distributes data across shards based on a shard key, ensuring even data distribution and efficient query routing.
- **Improved Performance -** By distributing data and query load across multiple shards, MongoDB sharding improves read and write performance, reducing latency and improving throughput.
- **High Availability -** Sharding enhances fault tolerance and availability by replicating data across multiple shards, ensuring data redundancy and minimizing downtime.



Data Partitioning - Sharding partitions data into smaller chunks called shards, allowing for efficient data distribution and management across multiple servers, optimizing resource utilization and improving scalability.

SHARDING CONFIGURATION

MONGOD --CONFIGSVR --PORT 28041 --BIND_IP LOCALHOST --REPLSET CONFIG_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/CONFIGSRV &

MONGOD --CONFIGSVR --PORT 28042 --BIND_IP LOCALHOST --REPLSET CONFIG_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/CONFIGSRV1 &

MONGOD --CONFIGSVR --PORT 28043 --BIND_IP LOCALHOST --REPLSET CONFIG_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/CONFIGSRV2 &

MONGOSH --HOST LOCALHOST --PORT 28041

MONGOD --SHARDSVR --PORT 28081 --BIND_IP LOCALHOST --REPLSET SHARD_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARDREP1 &

MONGOD --SHARDSVR --PORT 28082 --BIND_IP LOCALHOST --REPLSET SHARD_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARDREP2 &

MONGOD --SHARDSVR --PORT 28083 --BIND_IP LOCALHOST --REPLSET SHARD_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARDREP3 &

MONGOSH --HOST LOCALHOST --PORT 28081

MONGOD --SHARDSVR --PORT 29181 --BIND_IP LOCALHOST --REPLSET SHARD3_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARD3REP1 &

MONGOD --SHARDSVR --PORT 29182 --BIND_IP LOCALHOST --REPLSET SHARD3_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARD3REP2 &

MONGOD --SHARDSVR --PORT 29183 --BIND_IP LOCALHOST --REPLSET SHARD3_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARD3REP3 &

MONGOSH --HOST LOCALHOST --PORT 29181

MONGOD --SHARDSVR --PORT 29081 --BIND_IP LOCALHOST --REPLSET SHARD2_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARD2REP1 &

MONGOD --SHARDSVR --PORT 29082 --BIND_IP LOCALHOST --REPLSET SHARD2_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARD2REP2 &

MONGOD --SHARDSVR --PORT 29083 --BIND_IP LOCALHOST --REPLSET SHARD2_REPL --DBPATH /HOME/KARTHIKEYAN/SHARD-DEMO/SHARD2REP3 &

MONGOSH --HOST LOCALHOST --PORT 29081

MONGOS -CONFIGDB CONFIG_REPL/LOCALHOST:28041,LOCALHOST:28042,LOCALHOST:28043 --PORT 27012

SH.ADDSHARD("SHARD_REPL/LOCALHOST:28081,LOCALHOST:28082,LOCALHOST:28083")
SH.ADDSHARD("SHARD2_REPL/LOCALHOST:29081,LOCALHOST:29082,LOCALHOST:29083")
SH.ADDSHARD("SHARD_REPL/LOCALHOST:29181,LOCALHOST:29182,LOCALHOST:29183")

SH.ENABLESHARDING("DEMO")

SH.SHARDCOLLECTION("DEMO.HUSER", { _ID : 1 })

SHARDING TYPES

RANGE BASED

RANGE PARTITIONING

Data is partitioned based on a specific range of values, such as timestamps or alphabetical order, enabling efficient querying of related data subsets.

NATURAL DATA ORGANISATION

Range-based sharding aligns with the natural organization of data, making it intuitive and effective for range-based queries.

BALANCED DATA DISTRIBUTION

Range-based sharding ensures a balanced distribution of data across shards, preventing hotspots and optimizing performance

QUERY EFFICIENCY

Queries involving range-based criteria benefit from reduced query times due to the optimized data distribution across shards.

HASHED

UNIFORM DATA DISTRIBUTION

Hashed sharding evenly distributes data across shards based on a hash function, ensuring balanced workload distribution regardless of data values.

RANDOM SHARD ASSIGNMENT

Hashed sharding randomly assigns data to shards, minimizing hotspots and evenly distributing write operations, enhancing scalability.

LIMITED RANGE QUERIES

Hashed sharding is less suitable for range-based queries due to the random distribution of data across shards, impacting query efficiency.

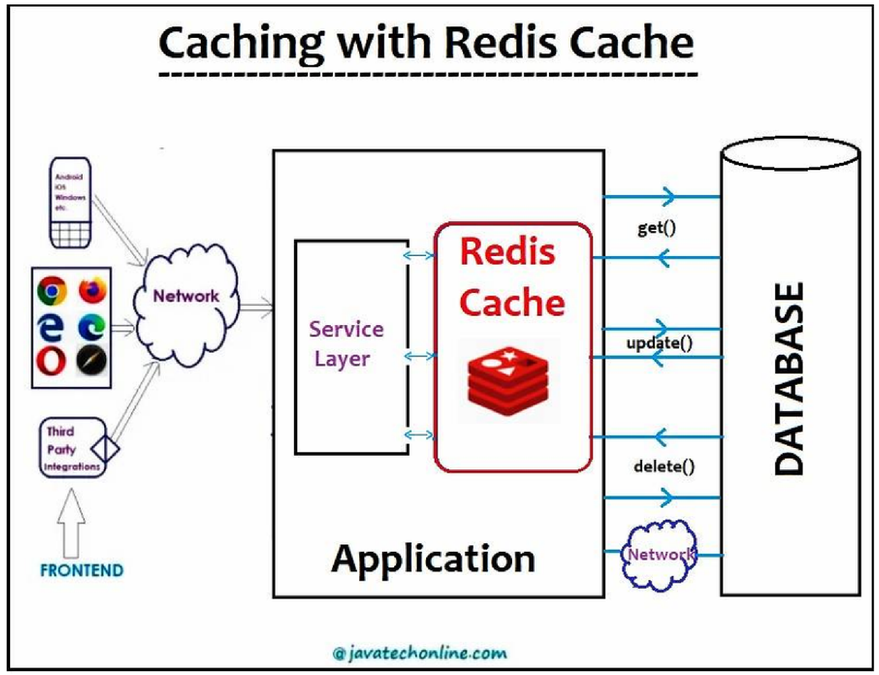
FLEXIBLE SCALABILITY

Hashed sharding offers flexibility in adding or removing shards without affecting existing data distribution, simplifying scalability operations.

REDIS

REmote DIctionary Server

- **In-Memory Data Store -** Redis is an in-memory database, storing data in RAM for lightning-fast access and retrieval.
- **Versatile Data Structures -** Redis supports various data types like strings, lists, sets, hashes, and more, enabling diverse data storage and manipulation.
- **High Performance -** Redis is renowned for its high throughput and low latency, making it ideal for use cases requiring real-time data processing and caching.
- **Pub/Sub Messaging -** Redis offers a publish/subscribe messaging system, facilitating real-time communication and event-driven architectures.
- **Advanced Features -** Redis includes advanced features like transactions, Lua scripting, and clustering, enhancing its usability for complex application scenarios.



- **Persistence Options** - Redis provides multiple persistence options, including snapshots and append-only files (AOF), ensuring data durability and recovery.
- **Scalability and High Availability**: Redis supports clustering and replication, enabling horizontal scaling and fault tolerance for demanding applications.

RateLimiting

A rate limiter is a critical tool in managing the flow of traffic or requests to a system, preventing overload and ensuring optimal performance. It enforces predefined limits on the number of requests a client can make within a specified timeframe.

PREVENTING OVERLOAD

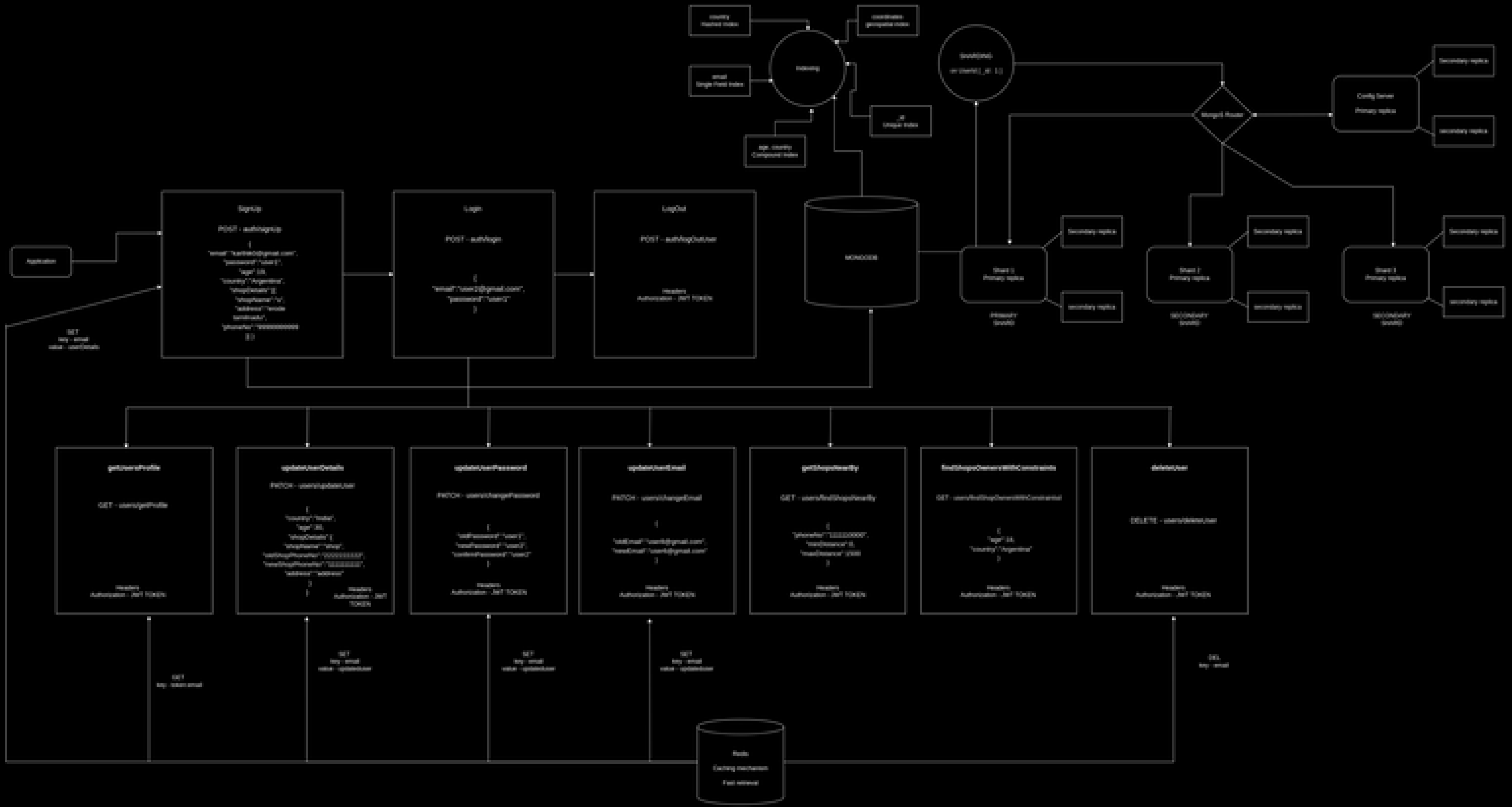
Protect systems from being overwhelmed by limiting the number of requests processed within a set timeframe, maintaining stability and preventing service degradation.

MITIGATING ABUSE

Serve as a defense mechanism against abusive or malicious behavior by imposing restrictions on the frequency of requests from individual clients, safeguarding system resources

OPTIMIZING RESOURCES

By regulating traffic flow, rate limiters optimize resource utilization, ensuring fair access to system resources and enhancing overall system efficiency.


Application
SignUp
POST - auth/signup
Login
POST - auth/login
LogOut
POST - auth/logOutUser
Headers
Authorization - JWT TOKEN
SET
key - email
value - userDetails
country
Hashed index
coordinates
geospatial index
Indexing
email
Single Field index
_id
Unique index
age, country
Compound index
SHARDING
MongoS Router
Config Server
Primary replica
Secondary replica
secondary replica
MONGODB
Shard 1
Primary replica
PRIMARY
SHARD
Shard 2
Primary replica
SECONDARY
SHARD
Shard 3
Primary replica
SECONDARY
SHARD
getUsersProfile
GET - users/getProfile
updateUserDetails
PATCH - users/updateUser
updateUserPassword
PATCH - users/changePassword
updateUserEmail
PATCH - users/changeEmail
getShopsNearby
GET - users/findShopsNearby
findShopsOwnersWithConstraints
deleteUser
DELETE - users/deleteUser
Headers
Authorization - JWT TOKEN
SET
key - email
DEL
key - email
Redis
Caching mechanism
Fast retrieval