

A very high level design of the rate limiter.

**Instructions to run the RateLimiter:**

1. In the attached zip , you will find RateLimitedAPI.jar
2. Use this command to start the server:

java -cp RateLimitedAPI.jar com.raviraj.hotel.query.ratelimtedapi.service.Startup config.txt hoteldb.csv

OR

If you clone the git repo, and run it from an IDE just run the com.raviraj.hotel.query.ratelimtedapi.service.Startup Java class. ( Make sure to give program arguments , the first argument is config to ratelimiter, second is the db csv file)

here hoteldb.csv is the csv file containing the data

config.txt is the config file for the rate limiter. This config file contains the APIs registered and

other configurations such as the data store used or rate limit functionality this project should use. Make sure hoteldb.csv and config.txt are in the same folder as the jar OR specifty the absolute path for these files.

1. Once the server is started you can hit the end points
2. <http://localhost:4444/hotelservice/city/Bangkok>
3. http://localhost:4444/hotelservice/room/Deluxe
4. <http://localhost:4444/hotelservice/city/Bangkok>?sort=ASC
5. <http://localhost:4444/hotelservice/room/Deluxe?sort=DESC>
6. This rate limiter runs with default config of 10 requests for 5 seconds for city API and 100 requests every 10 seconds for room api. You can change the config file and run the server again with different parameters to test it.
7. You can see that when the rate limit is below threshold the API returns proper response, in

Other cases we get RATE\_LIMIT\_EXCEEDED and the API pauses for 5 seconds.

1. The project also has a Integration Test case which tests the rate limit functionality of the API, you can find it in the code HotelServiceTest.java.
2. Creating the config file for RateLimiter:

The config file accepts 3 parameters

1. RATELIMITER.FUNCTION.TYPE , valid values are SLIDINGWINDOW\_WITHLOCK and SLIDINGWINDOW\_WITHOUTLOCK
2. RATELIMITER.DATAACCESS, valid values are INMEMORY , for running this rate limiter in a distributed environment , we need to use a persistent data store such as Cassandra or Redis and give a proper implementation for com.raviraj.hotel.query.ratelimiter.API.RateLimterDataAcess
3. To register a API with this rate limiter use

API=city,10,5 { API = API\_NAME, TIME\_WINDOW\_IN\_SECONDS,ALLOWED\_REQUESTS\_IN\_THE\_TIME\_WINDOW}

The above is self-explanatory with API name, followed by the time window limit and the requests allowed in the time limit.

As you can see, we can configure any number of APIs with this rate limiter with any different configurations for rate limits for each API.

1. I have provided a way to check Rate Limiting, You can run the Junit Test HotelServiceTest.
2. To check manually , either open the URLs given above in a browser or use the curl –X GET <http://localhost:4444/hotelservice/city/Bangkok>

**Assumptions and Architecture decisions:**

1. I am using a In-memory data store for storing the hoteldb.csv . However, the implementation is such that we can use a different database by plugging in a different database client.
2. I am suing a In-memory data store for storing the rate limiter data store, However, the implementation is such that we can use a different data base by implementing a different instance for com.raviraj.hotel.query.ratelimiter.API.RateLimterDataAcess
3. **This rate limiter can be used in a distributed environment,** by using the same implementation of RateLimitDataAccess across multiple servers running the API. If suppose, we write a implementation of com.raviraj.hotel.query.ratelimiter.API.RateLimterDataAcess

that talks to Cassandra , and this implementation is configured with same Cassandra end point across multiple servers , then all of them talk to same Cassandra tables , there by rate limiting the API, even when its run on different servers.

1. In the project, I am providing two implementations of Rate limit Functinality , one that uses Locks for every request, another implementation that doesn’t use locks for every request but periodically updates local counts to a global store. I have written the second implementation , so that we can achieve a better latency by sacrificing on the rate limit conditions. In the ideal world scenario, it might be beneficial to have a lower latency by relaxing the rate limit conditions. We might serve more requests than agreed , but that might not cost us a lot considering gain in the latency.
2. This rate limit can be used to rate limit Users/ Organizations using a given API. Each API has a API id, If we override the API ID and associate a user id along with a API ID , the rate limiting can then work even as user level.

**Git Repo ;** [**https://github.com/RavirajDataEngineering/rate-limiter**](https://github.com/RavirajDataEngineering/rate-limiter)

**The source code is also attached in the zip uploaded.**