Fashion Cloud

Cache API

Your task is to build a REST API that exposes methods to interact with a cache that you will build. You will have to use **Node.js** and **Express.js** to build the API and **MongoDB** to store the cache data in. The cache does not have another data source in the background that is cached. **All data returned by the cache is random dummy data.** You do not need to build a frontend. The API shall be used with tools like curl or Postman.

Following features have to be implemented for the cache:

- Add an endpoint that returns the cached data for a given key
    - If the key is not found in the cache:
        - Log an output “Cache miss”
        - Create a random string
        - Update the cache with this random string
        - Return the random string
    - If the key is found in the cache:
        - Log an output “Cache hit”
        - Get the data for this key
        - Return the data
- Add an endpoint that returns all stored keys in the cache
- Add an endpoint that updates the data for a given key
- Add an endpoint that removes a given key from the cache
- Add an endpoint that removes all keys from the cache

Following additional features have to be also included:

- The number of entries allowed in the cache is limited. If the maximum amount of cached items is reached, some old entry needs to be overwritten (Please explain the approach of what is overwritten in the comments of the source code)
- Every cached item has a Time To Live (TTL). If the TTL is exceeded, the cached data will not be used as it has expired. If you should request expired keys again, a new random value will then be generated (just like cache miss). The TTL will be reset on every read/cache hit

Keep this in mind when you build the cache:

- Create a public Github repo for the code. Commit your progress with good commit message and in sensible chunks
- It’s up to you what Version of Javascript to use. Feel free to use ES2016 or other tools like Typescript or Flow. It just needs to be able to run on the defined node version
- Use standards and best practices as best as you can
- Create the necessary npm scripts. Make sure that in the end it will at least support the standard commands “install”, “start”, “test”
- Make sure it works in a Unix environment

- The only dependency to the machine shall be MongoDB. When I will run your code on my machine I will have a MongoDB Instance running. Every other dependency should be resolved by npm install
- Test the code in a way that seems appropriate to you in a professional environment
- Think about the names of the endpoints, the HTTP methods to use (GET, PUT, POST, DELETE, etc.) and the structure of the returned data
- Use the appropriate status codes (also in case of an error)
- Structure the code as if it was a huge application and make sure that with the used structure the code will still be readable and maintainable when it has grown to that point
- Pretend you have a big team and make it really easy for them to set up your environment. Think about what data should be configurable in different environments (e.g. db connection)

Happy coding :-D