**Concurrency Exercise**

Implement a simple Node.js server implemented with Express and Mongoose that will expose the following rest API for a message object:

* Get message by Id
* Get all messages
* Put a message – update existing one
* Post a new message

The get operation by Id should support caching solution –

* get object will first check in cache whether the message exist.
* If it does exist in cache - get it from cache.
* If not – it will load from DB to cache .

Creating messages and updating messages should update the cache as well.

For caching :

* You can use LRU module as caching layer

<https://www.npmjs.com/package/lru-cache>

* For locking – use async-lock module, and lock by the object version.

<https://www.npmjs.com/package/async-lock>

**How to check your implementation:**

* Write a test that send 10 requests to server in parallel (use promise.all in test implementation) and make sure results are taken from cache for 9 requests and uploaded once to cache for one of the requests.
* Create a test with 3 requests that try to update the same object – make sure the changes does not override one another and that each request upload the update to cache.

**Message Fields:**

* Id – auto generated by DB
* Title – string
* Text – string
* Version – guide, cannot be changed from request or from any consumer. Auto generated on create and updated on every update by the server itself
  + Generate new version guide on every update
  + Use this version for locking the object in memory.

If you want to use a template and not write the entire server by yourselves, you can use the server in the following git repository:

<https://github.com/tamarstern/message_server_template_for_node_course>