Assignment #3: User Service with Real Databases

Goal

In this assignment, we will update our existing user service to utilize real databases, adding two new features along the way:

- Expiring sessions that are preserved beyond the lifetime of the app server, via Redis
- User data that are preserved beyond the lifetime of the app server, via Mongo

Tasks

You should start with the Assignment 2 version of your user service, and make the following modifications:

- Remove the local in-code storage of users, in favor of storing in MongoDB.
    - You can use either the **mongodb** or **mongoose** packages (not both!)
    - **You will connect to the remote database I provide for you.** Your per-user details are in your section of the Class OneNote in Teams, and the instructions were already included in the Mongo lab.
    - Your collection should be named for the function its used for - **users**
    - **Note:** you could use an "index" to restrict the creation of duplicate users (by username), but that will result in a Mongo error object that is difficult to analyze. Instead, consider checking for the existing of that username before creating a new user. Searching is easy in Mongo!
    - **Note:** Remember that Mongo operations are implemented with async/await. Therefore, you should **await** the result of the operation (i.e., await mongoClient.op()), and any function that includes an await – likely, your routes – should be decorated with **async** (i.e., async (req, res) => {}).
- Remove the local in-code storage of sessions, in favor of storing in Redis
    - You should use the **redis** (i.e. node-redis) package, which we used in the lab
    - You will connect to the localhost Redis server, under the default configuration.
    - You should scope your keys to sessions – perhaps something like **sessions:<sessionKey>**
- Implement a change in semantics for the Login API:
    - Sessions should expire after 10 seconds
        - Don't implement expiration yourself – let Redis do it! You utilized the EXPIRE function in the Redis lab.
    - A successful Login should remove any previous sessions for that user.
        - This requires you to store a lookup by user, **sessionsIdsByUserId:<userId>**, as you cannot search in Redis. The value would be the Redis key for the actual session – i.e. sessions:<sessionId>.

- You can delete keys in Redis using the **del** command, which is like HTTP DELETE – it deletes the key if exists, but it's perfectly happy if the key already doesn't exist.
- Note that each session must have a different key, even for the same user (per the unit tests)!

Aside from the redis package, and either the mongodb or mongoose package, you may not add any more packages to your project. In fact, you will likely not need the base64url package anymore. You could remove it with **npm uninstall base64url,** but that's not required. *You may not use connect-redis-sessions or any other package you may find that "handles sessions for you".*

Testing

Your user service will be tested with a Postman suite that is similar to the one for Assignment 2, though several tests have been added to it, or changed. The Postman tests are posted on Moodle alongside this document.

You should now be experienced with developing with unit tests. **As a result, the unit test grade is now all-or-nothing – you must pass all 85 tests, or you will receive 0/40 points for that section of the rubric.**

Note that the API paths should not change – they're still **v1**!

Submission

In addition to deploying and running on EC2, you will submit your code, similar to previous assignments with some changes.

You do need to update your primary code file to be named **assignment3.js.** My suggested method for this is:

1) Copy your assignment 2 content to a new directory. You could copy in Linux or via Windows Explorer, or simply unzip your submission. You should NOT copy the node_modules directory or if you do, you should delete it.
2) **del package*.json** in the new directory
3) Run **mv assignment2.js assignment3.js** to rename the file.
4) Run npm init again, ensuring that you enter **assignment3.js** as the entry point.
5) Run **npm install packageName** for each package you're using – likely express, uuid4, redis, and either mongodb or mongoose.

Your submission should be named **yourAlias-CS261-3.zip**, where yourAlias is the username you use to log into your email and lab PC.

The submission should include:

- **assignment3.js**, the primary Javascript file (note the rename!)
- Your other Javascript files (modules)
- package.json
- package-lock.json

… without any subdirectories *(including node_modules).*

Rubric

40 Points: Unit tests *all* pass (**85 total tests**)

The Assignment 3 set of unit tests will be run against your server in a configuration that matches what you set up in Assignment 2. If **all 85 tests** pass, then you will receive 40 points **(otherwise 0 points).**

30 points: MongoDB used correctly for user data

If your service is using the correct MongoDB database, provided for you, for user data, without any strange issues, you will generally receive all 30 points. Not using MongoDB at all will receive 0 points. Partial credit may be given in some cases.

30 points: Redis used correctly for *expiring* sessions

If your service is using Redis for sessions that expire after 10 seconds, without any strange issues, you will generally receive all 30 points. Not using Redis at all will receive 0 points. Partial credit may be given in some cases.

Submission Penalties

In addition to the grading rubric above, you can receive these penalties if your submission is not correct:

- -5 if your submission files are not named as above.
- -5 if your submission includes additional files.

I may apply other submission penalties if needed.

Technical Notes

- Remember – each time you restart your computer, your local Redis server won't work until you start it in WSL, using **sudo service redis-server start**
- **Tip:** You should make the expiration duration (i.e., 10 seconds) a const at the top of one of your primary files. You could leave it at something long, like 10000 seconds, to aid in debugging of your Redis implementation. Then, when you need to test expiration (and for submission), you can change the value to 10.
    - We will utilize a more powerful system for per-environment configurations later in the course.
- Your access to the Mongo server is not secure! Mongo does support SSL, but the configuration on the client and server would be complicated for our uses. Once we move our service to the cloud:
    - We won't need user-authentication for Mongo anymore, as the database will be running within the same trusted datacenter (in our configuration, the same machine) as the service itself.
    - We will use SSL to secure our service, via the nginx reverse-proxy service.