Smitch

Backend Developer

Problem Statement

Problem Statement **1**

Diving Deeper **2**

Tools & Technologies involved 2

Requirements 2

Evaluation Metrics **5**

Submission Guidelines **5**

1. Problem Statement

Create a backend application to track power usage for users. You are required to provide a set of APIs for users to perform the following tasks

- User Registration
- User Login
- Authentication
- List Sessions
- User Logout
- Add Power Usage (PU)
- List Power Usage
- List Power Usage Day Wise
- Get Power Usage Streak *

*Note: Sections mentioned with * are not expected from junior-level developers*

2. Diving Deeper

a. Tools & Technologies involved

i. Web Server - Node.js (usage of TypeScript is optional)
ii. Routing Framework - Express
iii. Database - PostgreSQL or MySQL (use Sequelize ORM if needed)
iv. Version Control - Git
v. Documentation - Postman or Swagger
vi. Unit Tests - Jest *
vii. Caching - Redis *
viii. Containerization - Docker *

b. Requirements

i. All of the APIs that are created should follow REST standards
ii. All of the relational database tables that are getting created are expected to be normalized and follow standards
iii. User Registration
    - The user should be able to register a new account
    - At the time of registration, the following information is obtained
        - *Display Name*
        - *User Name* (should be unique across the system)
        - *Email ID* (should be unique across the system)
        - *Mobile Number* (should be unique across the system)
        - *Password*

- If the user is already registered, send the appropriate HTTP status code and error message for the same
- For each user registration, a UUID should be created. This *User ID* would be used at the time of generating the *Access Token*

iv. User Login
- The user should be able to log in with the registered account
- Users could use either *User Name* or *Email ID* or *Mobile Number* and *Password* to login the account
- If the password does not match, send the appropriate HTTP status code and error message
- Upon successful login, an *Access Token* is sent back as a response which would be consumed by the other APIs
- *Access Token* is a JSON Web Token (JWT) should have a validity of 5 minutes with the *User ID* as payload
- The user should also be able to perform multiple logins which would create multiple sessions for the user (i.e. user should be able to use any of the *Access Token* generated)

v. Authentication
- All other APIs other than *User Registration* and *User Login* should be authenticated
- APIs should accept *Access Token* as one of the headers to authenticate the user
- Only if the validation is successful, allow the API to perform its purpose otherwise return the appropriate status code and error message

vi. List Sessions
- The user should be able to get all his active sessions
- The response may include information such as
  - *Time* at which the login is performed
  - *User Agent* name (this would usually be passed by default from all types of clients like browser, postman, etc)

vii. User Logout
- Users should be able to log out of a particular session
- After successful logout, *Access Token* associated with the login should be invalidated (i.e. with this *Access Token* user should be not able to access the Authenticated APIs)
- Users should be able to log out of all sessions as well

viii. For the below set of APIs, let us consider the following scenario
- Let us imagine a factory that manufactures spare parts. Every employee there has a dedicated smart power socket into which any appliance in the factory could be plugged in. Whenever employees use their sockets, power consumption by that appliance would be tracked. Upon switching off the socket,

power consumption details (From Time to Time, Duration, Units Consumed) are sent to the server via API. Employees could use the socket multiple times in a day with various types of appliances (low-power, high-power, mid-power). There would also be instances where employees do not use the socket for the entire day or could have gone on vacation.

ix. Add Power Usage (PU)
- API should be able to add a power usage entry
- At the time of adding a power usage, following information are obtained
  - *From Time*
  - *To Time*
  - *Duration*
  - *Unit Consumed*
  - *Appliance Type* (‘low-power’, ‘high-power’, ‘mid-power’)
- Appropriate HTTP status code and messages should be sent based on success or failure

x. List Power Usage
- The user should be able to list all power usage entries for a given time range
- API could get *Start Time* and *End Time* as input and return respective power usage entries as a response
- Appropriate HTTP status code and messages should be sent in case of failure

xi. List Power Usage Day Wise
- Similar to *List Power Usage* API but the response should be returned day wise
- For example: If *Start Time* is "2021-09-16T18:30:00.000Z"’ and the *End Time* is "2021-09-20T18:29:59.999Z" the response should return data for 4 days. Every day’s unit consumed and duration should be added up and sent as a response

xii. Get Power Usage Streak *
- Users should be able to get their power usage streak
- Streak is the total no. of days since the last zero power usage day
- For example: If there are power usage entries for yesterday and the day before, then the current streak is 2. If there are no power usage entries for the day before yesterday, but entries are present yesterday then the current streak is 1. If there is no entry present yesterday, then irrespective of the day before, the current streak resets to 0.

Current Day - 2	Current Day - 1	Current Day	Streak
PU present	PU present	Any	2

PU not present	PU present	Any	1
Any	PU not present	Any	0

For all three scenarios in the table, the assumption is that the API is getting hit on the current day.

3. Evaluation Metrics

Below are some of the metrics that would be evaluated from this assignment

a. Clean bug-free code that satisfies the requirements
b. Modularity (directory structure, module/package structure, etc)
c. Readability (variable naming, naming convention, easy-to-read logic, inline code comments, etc)
d. Documentation
    i. README.md
    ii. Inline code documentation
    iii. Swagger / Postman
    iv. Git conventional commit messages (clear commit message at completion of every working snippet/module/functionality)
e. Best practices and standards (adherence to industry best practices in terms of code, design choices, and tool/library selection choices)
f. Additional Metrics *
    i. Ability to architect code and database in a way that could be easily scalable
    ii. Ability to test the code with unit/integration tests
    iii. Ability to perform caching mechanism to optimize the code wherever necessary
    iv. Ability to containerize the application for seamless deployment

4. Submission Guidelines

a. Upload the project to a GitHub **private** repository and share the repository to *gowtham.r@mysmitch.com* by the given deadline
b. Add a README file with instructions to run the project and special instructions in case of any
c. Deadline: Refer to Email
d. P.S Please do not commit and push the complete project at one go. We would like to see how you approach the problem, once you finish a module/function or fix a bug, do a commit with an appropriate commit message