Home Assignment

Target Audience: Backend Developer

Instructions

Here are the guidelines which you should follow while doing the assignment.

1. The assignment consists of three parts:
    a. Problem Solving
    b. REST API interaction
    c. Adding Persistence Layer [BONUS SECTION]
2. Each part has an objective to be tested and must be attempted only after the previous section is completed.
3. Please ensure that the coding conventions, directory structure and build approach to your project follow the conventions set by popular open source projects in the language that you're using.
4. Please ensure that you write comprehensive Unit Tests
5. You have to solve the problem in **Java or GoLang** language
6. Please use **Git for version control** and make incremental changes

Submission

1. We expect 3 different code versions which can be run independently of the other. Name them SUBMISSION_A, SUBMISSION_B and SUBMISSION_C respectively.
2. We expect you to **send us a single standard zip or tarball** of all submission folders when you're done. That includes Git metadata(the .git folder) in the tarball so we can look at your commit logs and understand how your solution evolved. **Granular and descriptive comments are a huge plus**.
3. Each submission folder should have its own readme on how to execute and test. We strongly advise to double check if code is runnable on foreign system by following the readme. If your submission is not runnable, it cannot be graded.
4. Please do not check in class files, jars or other libraries or output from the build process. Use a standard build automation and dependency system like ant/maven/rake/gradle etc
5. Please do not make either your solution or this problem statement publicly available by, for example, using GitHub or bitbucket or by posting this problem to a blog or forum.

How will your assignment be tested?

1. Your submissions will be imported and run in a linux based operating system.
2. All the 3 sections will be tested separately. We will see comprehensibility of your unit tests and run your APIs for validating them.
3. We are really interested in your low level design, so structure your code in such a way that it can easily be extended to support feature extension/modifications
4. Look out for corner cases, error handling and code structure!

Problem Statement

We need to create a task management system. It includes the following functionalities:

1. There are two basic entities: TASK and USER
2. A task is an entity representing some information which needs some action.
3. Implement a method/interface so user can register themselves
   [create a valid entry in this system]
4. A new user can be registered in the system
5. A registered user can create a new task
6. A task can be assigned to a given registered user
7. A registered user can see task status and its details
8. A registered user can update the task details/status

Part A:: Problem Solving

- Use appropriate data structure to represent Task and User
- Create appropriate functions to support the above mentioned functionalities
- Write appropriate unit test cases with maximum code coverage

Part B::REST API Interaction

In this section, our main objective is to test your ability to write APIs according to the guidelines given. Each API has expected payload and expected response. There are some choices of json responses that you need to make (explained in each API).

- POST /register_user to register a new user
    - PAYLOAD
      ```
      {
        "NAME": "USER_1",
        "EMAIL": "user_1@true-fan.com"
      }
      ```
    - EXPECTED JSON RESPONSE
      ```
      {
        "OPERATION": "REGISTER_USER",
        "OUTPUT": "USER <USER_NAME> IS REGISTERED WITH ID <USER_ID>",
        "DATA": "<user_json_with_id>",
      }
      ```
      `Fields in user_json_with_id (must include the id of the user created) is your decision`

- POST /create_task/<userId> to create a task by the given user. This should throw proper response if the user is not registered
    - PAYLOAD

```
{
  <task_json>
}
```

    - EXPECTED JSON RESPONSE

```
{
  "OPERATION": "CREATE_TASK",
  "OUTPUT": "USER <USER_ID> CREATED A NEW TASK <TASK_ID>",
  "DATA": "<task_json_with_id>"
}
```

Fields in task_json and task_json_with_id (must include the id of the task) is your decision and may vary according to your representation of task

- GET /tasks/creator/<userId> to return all tasks created by user
    - EXPECTED JSON RESPONSE

```
{
  "OPERATION": "CREATE_TASK",
  "DATA": "LIST OF <task_json_with_id>"
}
```

DATA is a list of task_json_with_id (must include the id of the task. Defining task_json is up to you

- POST /assign_task/<taskId>/<userId> to assign a task to the user
    - EXPECTED JSON RESPONSE

```
{
  "OPERATION": "ASSIGN_TASK",
  "OUTPUT": "TASK_ID> ASSIGNED TO <USER_ID>"
  "DATA": "<task_json_with_asignee>"
}
```

Fields in task_json_with_asignee is your decision (must include the id and asignee of the task)

- GET /tasks/assignee/<userId> to return all tasks [alongwith task details such as status, creator, etc] assigned to given user
    - EXPECTED JSON RESPONSE

```
{
  "OPERATION": "MY_TASKS",
  "DATA": "LIST OF <task_json_with_id>"
}
```

DATA is a list of task_json_with_id (must include the id of the task. Defining task_json is up to you

Part C::Adding Persistence Layer [BONUS SECTION]

- Add the ability to your functions to persist the data
- The user and tasks are now supposed to be saved in some schema [MySQL or Mongo]
- Run all the above created APIs after adding the persistence layer in the same order as in PART B. Idea is to create a data snapshot
- Any data modification, query made should reflect/read from this data source
- Include any create schema/table commands in readme
- Include the snapshot of your database after these steps (using mysqldump or mongodump) inside the tar/zip created for this section.

--------------------------------------XXXXXXXXXXXXXXXX---------------------------------------