CODELABS ACADEMY

Express Services

In this assignment, you will implement the services for the remaining functionalities (Profile management, challenge creation and listing, grading...).

Content management

1. Challenge creation

- Add a service to create the challenge and persist it in the database
- The following is an example of a challenge creation request:

Unset

```
{
  "title": "factorial",
  "category": "Math",
  "description": "### Problem Statement:\nCompute the factorial of a non-negative integer `n`, denoted as `n!`. The factorial of `n` is the product of all positive integers less than or equal to `n`.\n\n### Example:\nFor example, the factorial of `5` is `5! = 5 * 4 * 3 * 2 * 1 = 120`.\n\n### Constraints:\n- The input `n` is a non-negative integer.\n- `0 <= n <= 20`.\n\n### Approach:\nA simple approach to compute the factorial of `n` is to use recursion. We define a recursive function `factorial(n)` that returns the factorial of `n`. The base case of the recursion is when `n` is `0` or `1`, in which
```

```
case the factorial is `1`. Otherwise, we recursively
compute the factorial of `n-1` and multiply it by
`n`.\n\n### Implementation:\nTo implement this, we can
define a recursive function `factorial(n)` that takes a
non-negative integer `n` as input and returns its
factorial. In the function, we handle the base case when
`n` is `0` or `1`, and recursively call `factorial(n-1)`
for other values of `n`. Finally, we return the product of
`n` and the factorial of `n-1`.",

  "level": "Hard",

  "code": {

    "function_name": "factorial",

    "code_text": [

      {

        "language": "py",

        "text": "def factorial(n):\n    return 1"

      },

      {

        "language": "js",

        "text": "function factorial(n) {\n return 1\n}"

      }

    ],
```

```
    "inputs": [
      {
        "name": "n",
        "type": "number"
      }
    ]
  },
  "tests": [
    {
      "weight": 0.8,
      "inputs": [
        {
          "name": "n",
          "value": 5
        }
      ],
      "output": 120
    }
  ]
```

```
}
```

- Make sure to return proper responses upon successful creation and report any errors

3. Challenge Listing

- Implement a service to list all challenges.
- If the requester is a coder, display all available challenges.
- If the requester is a manager, display only challenges created by them.
- Include additional relevant information for each challenge, such as its `solution_rate`, representing the percentage of coders who correctly solved the challenge.
- If the requester is a coder, display the `status` of each challenge as well. Possible values for challenge status are:

Status	Description
`Waiting`	The challenge has not been attempted or solved by that coder yet
`Attempted`	The challenge has been attempted which means that the coder has made a submission but it didn't pass
`Completed`	The coder has successfully completed the challenge

- The same thing should be done for getting the challenge by id.

4. Categories Listing

- Add a service that queries the database to fetch all the categories.

CODELABS ACADEMY

Grading

In this part, you are going to implement another feature, which is grading a submission. This is the most important function in the system. You are not going to implement the actual code runner, you are going to use an already existing code runner that we created for you. It runs test cases for codes written in `javascript` and `python`.

The code runner functions as a web service, offering a RESTful API for seamless communication. Your role will be to interface with this runner, transmitting both the submitted code and accompanying test cases. Subsequently, upon receiving this data, the runner will execute the code against the provided tests, generating comprehensive reports detailing the outcomes of these evaluations.

- First, make sure to protect the route for the grading service so that only coders are able to submit a code for grading.
- Add the service that communicates with the code runner and calculate the final score. These are important things to consider while writing the service.
- We don't allow coders to submit correct solutions twice. So you should send a response indicating that the challenge has already been solved.
- The grader should take the submitted code and invoke the code runner service.
- You should, based on the language provided in the submission, create a grading payload and send it to the `code test runner` (Its URL is provided below).
- If the tests fail then the submission fails and the score remains 0.
- If the tests pass then the submission passes and the score should be calculated using this formula: `score = sum over all test cases`

`(test case weight * 100)`.

- This score should be added to the total score of the user.
- Make sure to return proper responses in case of errors.

The data provided to the grading service has the following shape:

```
Unset
{
  lang: "py" | "js".    // This is the language of the code
  code: "..."           // The coders' code
  challenge_id: "...",  // The challenge id
}
```

The following is an example of a code submission:

```
Unset
{
  "challenge_id": "65feaac34c7c0fa50a47fb3e",
  "lang": "py",
  "code": "def factorial(n):\n\tif n == 0: return 2
\n\treturn n * factorial(n-1)"
}
```

Code test runners

The `code test runner` is available via this endpoint:

```
Unset
POST https://runlang-v1.onrender.com/run
```

An example of a request to the `code test runner` is the following:

```
Unset
{
  "lang": "js",
  "code": "function factorial(n) {\n    if (n === 0) return 1;\n    return n * factorial(n - 1);\n}",
  "func_name": "factorial",
  "tests": [
    {
      "_id": "1234654654654d",
      "inputs": [{ "value": 1 }],
      "output": 1
    },
    {
      "_id": "1234654654654",
      "inputs": [{ "value": 5 }],
      "output": 120
```

```
    },
    {
      "_id": "12346546544d",
      "inputs": [{ "value": 4 }],
      "output": 24
    }
  ]
}
```

Description of code runner request:

Field	Description
`lang`	The type of programming language of the code
`func_name`	The same as `function_name` of the challenge's code
`code`	The coder's submitted code
`tests`	A list of test cases, each test case with its `_id` to help the runner differentiate between tests, the `inputs` which is an array of objects with a field named `value,` and finally the `output` which is the expected output of the test case

The test runner will output either a success or an error message. In case of a success, it should return a report of the tests.

An example of a successful runner response

Unset

```
{
  "status": "passed",
  "message": "<message here>",
  "test_results": [
    {
      "message": "",
      "status": "passed",
      "test_id": "1234654654654d",
      "time": 4
    },
    {
      "message": "",
      "status": "passed",
      "test_id": "1234654654654",
      "time": 1
    },
    {
      "message": "",
```

```
      "status": "passed",
      "test_id": "12346546544d",
      "time": 0
    }
  ]
}
```

Description of code runner response

Field	Description
`status`	Whether the submission is `passed` or `failed`
`test_results`	An array that contains each test its status whether it's `passed` or `failed`, the `test_id`, and the `time` (to be ignored) which is time spent in running that test
`message`	A string containing an optional message from the runner (for example, it might indicate that the code could be evaluated due to syntax errors)

Leaderboard

In this part of the assignment, you will implement the leaderboard ranking functionality.

1. **Leaderboard**

- First, make sure to protect the route for this service so that only coders are allowed to see the leaderboard.
- Add the service to get the leaderboard.
- The leaderboard should list all the coders sorted by their score (from the highest to the lowest).
- You should add the number of `solved_challenges` for each coder in the response. Solved challenges are the challenges from the coder's correct submissions.

2. Top K coders

- Protect the route to this service so that only the coder is allowed to list the top k coders
- Add the service for that controller to get the top k coders.
- The code of that service should be similar to the previous leaderboard task, you just need to `limit` the number of returned objects to `k`.

System Statistics

In this assignment, you are going to implement various system statistics (or analytics) such as the heatmap to describe how many recent correct submissions are made by a specific coder.

Also how many challenges the coder has solved for various difficulty levels and the trending categories.

> Note: In this assignment, some operations require the use of the powerful MongoDB aggregation pipeline either for performance or elegance. For which we are going to provide details about the pipeline's stages.

1. Solved Challenges Statistics

for this endpoint, we want something that looks like this as output.

```
Unset
{
```

```
  "totalEasySolvedChallenges": 10,
  "totalModerateSolvedChallenges": 2,
  "totalHardSolvedChallenges": 1,
  "totalEasyChallenges": 111,
  "totalModerateChallenges": 40,
  "totalHardChallenges": 5
}
```

Where

Field	Description
`totalEasySolvedChallenges`	The total number of `Easy` solved challenges
`totalModerateSolvedChallenges`	The total number of `Moderate` solved challenges
`totalHardSolvedChallenges`	The total number of `Hard` solved challenges
`totalEasyChallenges`	The total number of `Easy` challenges that are available in the platform

`totalModerateChallenges`	The total number of `Moderate` challenges that are available in the platform
`totalHardChallenges`	The total number of `Hard` challenges that are available in the platform

- First, make sure to protect the route for this service so that only coders are allowed to see the statistics.
- Add the service for that controller.
- To get the analytics data as described in the previous data example, you need first to get the total number of challenges for each level and then, for each difficulty level, get the number of correct submissions.

2. Trending categories statistics

For this, we want response data similar to the following:

```
Unset
[
  {
    "category": "Data structure",
    "count": 300
  },
  {
    "category": "Graphs",
    "count": 100
  },
```

```
  {
    "category": "Tree",
    "count": 5
  },
  {
    "category": "Math",
    "count": 1
  }
]
```

- First, protect the route of the trending categories.
- Add the service for that controller.
- You can use here the aggregation pipeline of Mongodb by composing the following stages:
- Filter submissions to include only those that were correctly passed to ensure that only passed submissions are considered for further processing in the pipeline.
- Perform a lookup (similar to `SQL join` operation) from submissions to challenges.
- Next, you can group the result by `category` and cumulate the number of occurrences denoted by `count`.
- Next, you can add the `category` field to the result which is the one expected in the output
- Next, you have to sort the results based on the `count` in descending order.
- Finally, you have to `project` the result and keep the `category` and `count` fields only.

3. **Heatmap**

For the submission strikes heatmap, we want a result like this

```
Unset
[
  {
    "date": "2024/04/01",
    "count": 10
  },
  {
    "date": "2024/04/01",
    "count": 10
  }
]
```

The user can specify the `start_date` and `end_date` as query arguments in the request to get strikes within that date range.

- Protect the route of the heatmap endpoint.
- Add the service for that controller
- To implement the service, make sure you transform `date` inputs which are strings to be `Date` objects (you can use `new Date(date as string)`). If the dates are not specified, the `start_date` should be set to the current date minus a year, and the `end_date` should be set to the current date.
- Similar to the categories trends you can use the aggregation pipelines with the following stages:

- First, you need to filter the passed submissions based on the coder and the submission date which should be between the start and end dates.
- Next, you add a date field to the result out of the `submittedAt` field and make it follow this format: `YYYY/mm/dd`.
- Next, you have to group the resulting documents by the `date` field and calculate the `count` of submissions for each unique `date`.
- Finally, you have to `project` the result and keep the `date` and `count` fields only.

Profile management

1. Get a user profile

- Similarly, this endpoint should be protected, so that both managers and coders are allowed to view their profiles.
- Create the service to get the profile.
- If the coder is the entity who made the request then you should calculate the rank of the coder and add it to the response.

1. User profile update

- Similarly, this endpoint should be protected, so that both managers and coders are allowed to update their profiles.
- Create the service to update the profile.

Note: Updating the avatar will be tackled in a later assignment once we discuss cloud services. Because we want the image to be uploaded to a cloud service and we store the provided URL in the database.