Coders App ExpressJS API

This assignment evaluates your understanding of creating and organizing Express applications, how Express manages routing, and creating validators for various API endpoints.

Throughout this assignment, you will adhere to the route, controller, and service architecture for the various API endpoints.

> Note: Services will be implemented in a later assignment as they require access to the data persistent layer.

Authentication and Profile Management

In this first part of the assignment, you'll be asked to implement routes and controllers with validation middlewares for authentication features for the different types of users, `Coders`, and `Managers` and some profile management operations.

1. Account Registration

- Add routes for `Coder` and `Manager` registration.
- Implement controllers for account registration.
- Apply the account registration validator to validate signup request data using Joi.

2. Account Login

- Add the routes for the `Coder` and `Manager` login.
- Add the controllers for account login.
- Add the login data validator in the controller to validate login request data.

3. Profile Management

- Add endpoints to get the details about the profiles of the `Coder` and the `Manager`.

- Add endpoints for both Coders and Managers to update general information (first name, last name, about).

Note: Updating the avatar will be discussed in a later assignment.

Content Management

In this second part of the assignment, you will implement the routes, controllers, and validators for challenges management functionalities.

1. Challenge Creation

- Add a route to create a challenge.
- Add the controller and create the validator for challenge creation.

Here's 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,
```

```
{
  "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
    }
  ]
}
```

As you can see, the challenge has a `title`, `description` (which is in a markdown format), `category`, and `level`.

And most importantly the code object has the code text as an array of the actual code and the language, and also the list of test cases along with inputs and the expected outputs.

> Note: This endpoint is solely intended for populating the database, as the challenge management will be rewritten in a subsequent assignment (NestJS assignment).

3. Challenge Listing

- Add a route to get all the challenges.
- Add the controller for that route.
- Make sure that this endpoint accepts a `category` query argument to filter challenges by their category.

4. Challenge Listing by Id

- Add a route to get a specific challenge by its id.
- Add the controller for that route.

5. Categories Listing

- Add a route to get all the existing categories.

- Add the controller for that route.

Grading Submission

In this part of the assignment, you are going to implement the routes and controller of the grading functionality

- Add a route to post a submission.
- Add the controller for that route.
- Add the validator to validate the submitted code before invoking the grader.

The requested data provided to the grader has the following shape:

```
Unset

{
  lang: "py" | "js".    // This is the language of the code (we support only
two languages)
  code: "..."           // The coders' code as text
  challenge_id: "...",  // The challenge id
}
```

An example of a code submission (to the code runner) is:

```
Unset

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

CODELABS ACADEMY

Leaderboard

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

1. Leaderboard

- Add a route to get the leaderboard
- Add the controller for that route

2. Top K coders

- Add a route to get the top k coders (k is a number passed as a query parameter and should be validated).
- Add the controller for that route

System statistics

In this last part of the assignment, you are going to implement routes and controllers to access 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 a coder has solved for various difficulty levels and the trending categories.

1. Solved Challenges Statistics

- Add a route to get the solved challenges statistics.
- Add the controller for that route.

2. Trending categories statistics

- Add a route to get the trending categories.
- Add the controller for that route

3. Heatmap

- Add the route to get the heatmap.

- Add the controller for that route, you should extract date (`start_date` and `end_date`) filters and pass them to be passed to the service