Junior Backend Developer - Practical Assignment

Description:

This assignment was designed to cover a series of aspects that we consider relevant for the existing role such as the ability to plan and execute, technical expertise, and focus on real-world solutions among other things.

The scope of this project is to evaluate the candidate's skills and not to deliver a full production-level project. We are aware that most candidates have a busy schedule so please focus on the right balance between showcasing your knowledge and time.

Assignment #1:

Count the number of Duplicates

Write a function **count_duplicates(str)** that will return the count of distinct case-insensitive alphabetic characters and numeric digits that occur more than once in the input string. The input string can be assumed to contain only alphabets (both uppercase and lowercase) and numeric digits.

Example:

count_duplicates("abcde") -> 0 # no characters repeats more than once
count_duplicates("aabbcde") -> 2 # 'a' and 'b'
count_duplicates("aabBcde") -> 2 # 'a' occurs twice and 'b' twice (`b` and `B`)
count_duplicates("indivisibility") -> 1 # 'i' occurs six times
count_duplicates("Indivisibilities") -> 2 # 'i' occurs seven times and 's' occurs twice
count_duplicates("aA11") -> 2 # 'a' and '1'
count_duplicates("ABBA") -> 2 # 'A' and 'B' each occur twice

Requirements:

- Ruby 2.7
- Write 5-10 test cases with Rspec or Unit::Test (optional)

Assignment #2:

Table Structure

Table: (**posts**)
- title - required
- body - required
- mark_for_deletion
- created_at

Table (**comments**):
- post_id - FK posts
- body
- created_at

Short Description

Implement a simple blog site with posts and comments using Ruby and Rails framework.
Implement post comments CRUD API

Requirements:
- Ruby 2.7
- Rails 6.1
- PostgreSQL any version
- Try to write test cases for **API** with Rspec (optional)

1. Implement simple Post and Comments pages

You can use **Bootstrap** to implement these UI components or any other library.

Posts:

- On the site main page show a Post List table (display only the last 10 posts, mark_for_deletion=false). Columns to display: id, title, created_at, actions (links to current post Edit, Delete)
- Give an opportunity to add a post (form with inputs: title, body, mark_for_deletion)
- Give an opportunity to edit, update or delete posts

Comments:

- Give an opportunity to add comments (form with inputs: body) to the post.
- Add Comments List on the post page.
- Give an opportunity to edit, update or delete comments

`2. Implement CRUD comments API`

`2.1 Fetch last 10 post where mark_for_deletion=false, order by created at desc`

GET - api/v1/posts

Response:

```
{
        status: 200,
        data: [
                { id: 1, title: 'Title', body: 'Body' , created_at: '2021-02-18' },
                { id: 2, title: 'Title', body: 'Body' , created_at: '2021-02-18' }
        ]
}
```

`2.2 Fetch last 10 post comments for current post`

GET - api/v1/posts/:post_id/comments

Response:

```
{
        status: 200,
        data: [
                { id: 1, post_id: 1, body: 'Body' , created_at: '2021-02-18' },
                { id: 2, post_id: 1, body: 'Body' , created_at: '2021-02-18' }
        ]
}
```

2.3 Implement add comment route

POST - api/v1/posts/:post_id/comments

```
Content-Type: application/json
{
        post_id: 1,
        body: ‘Comment’
}
```

Response examples:

On Success:

```
{
        status: 200,
        data: [
                { id: 3, post_id: 1, body: ‘Comment’ , created_at: ‘2021-02-18’ }
        ]
}
```

On Error:

```
{
   status: 422/500,
   errors: [
           { body: "Can’t be blank"}
   ]
}
```

2.4 Implement Update route

PUT - api/v1/posts/:post_id/comments/:comment_id

Content-Type: application/json

```
{
        post_id: 1,
        comment_id: 3
        body: ‘Comment Update’
}
```

On Success:

```
{
        status: 200,
        data: [
                { id: 3, post_id: 1, body: ‘Comment Update’ , created_at: ‘2021-02-18’ }
        ]
}
```

On Error:

```
{
   status: 422/500/404,
   errors: [
          { msg: "Comment not found"}
   ]
}
```

2.5 Implement delete comment route

DELETE - api/v1/posts/:post_id/comments/:comment_id
Content-Type: application/json
{
	post_id: 1,
	comment_id: 3
}

On Success:

{
	status: 200
}

On Error:
{
  status: 422/500/404,
  errors: [
	{ msg: "Comment not found"}
  ]
}

Deliverable:

- ➢ Project files: please, upload the project to a Github repository.Work with git as you usually do with the commit/pull/push routine so that we can see all the commits and work process;
- ➢ Setup instructions (if needed);
- ➢ Project Documentation:
  - Summary;
  - Utilized stack/tools;
  - In case you have integrated other code that has not been written by you, please mark it;
  - Development timeline – a simple list of tass and the time spent on each;
  - Any additional information that you consider relevant for this project;