Assignment 8

Part1: Using Sequelize Define the following models (3 Grades):

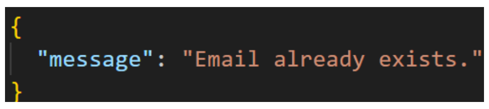
Users (using define) (0.5 Grade)	Posts (using init) (0.5 Grade)	Comments (using init) (0.5 Grade)
• **id** (Primary Key, Auto Increment) • **name** (VARCHAR) • **email** (VARCHAR, Unique) • **role** (ENUM: user, admin) • **createdAt** (Date) • **updatedAt** (Date)	▪ **id** (Primary Key, Auto Increment) ▪ **title** (VARCHAR) ▪ **content** (TEXT) ▪ **userId** (Foreign Key to Users) ▪ **createdAt** (Date) ▪ **updatedAt** (Date)	• **id** (Primary Key, Auto Increment) • **content** (TEXT) • **postId** (Foreign Key to Posts) • **userId** (Foreign Key to Users) • **createdAt** (Date) • **updatedAt** (Date)

Must Implemented:
1. Apply built-in validation to validate the email format and apply soft-delete (paranoid) to post table. **(0.5 Grade)**
2. Add a custom validation method "checkPasswordLength" that ensure the password length greater than 6 characters. **(0.5 Grade)**
3. add a custom validation method "checkNameLength" to a "beforeCreate" hook that ensure the name of the user is greater than 2 characters. **(0.5 Grade)**

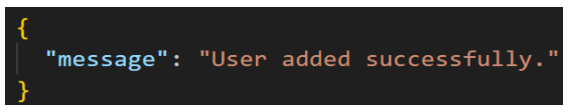
Part 2: APIs (⚠️ Ensure you apply the folder structure we discussed)

A- User APIs (2 Grades):

1. **Create a new user (using build and save) (make sure that the email does not exist before) (Don't forget to Handle validation errors). (0.5 Grade)**
   - **URL:** POST /users/signup



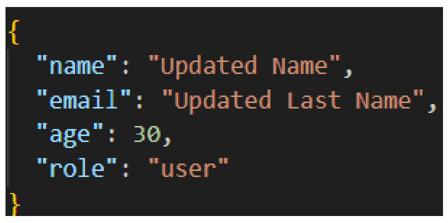
```
{
  "message": "Email already exists."
}
```



```
{
  "message": "User added successfully."
}
```

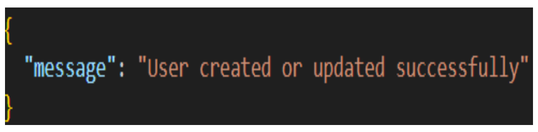
2. **Create or update based on PK and use skip validation option. (0.5 Grade)**
   - **URL:** PUT /users/:id

Input



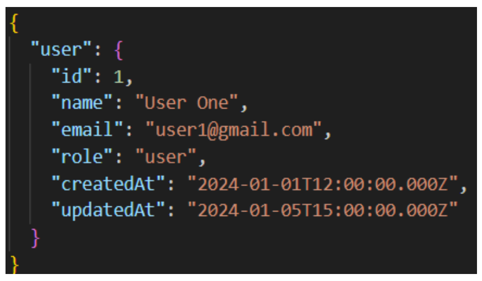
```
{
  "name": "Updated Name",
  "email": "Updated Last Name",
  "age": 30,
  "role": "user"
}
```

Output

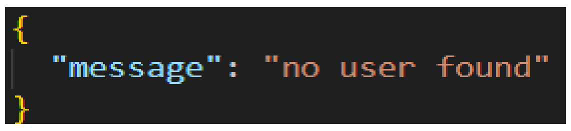


```
{
  "message": "User created or updated successfully"
}
```

3. **Write an API endpoint to find a user by their email address. (0.5 Grade)**
   - **URL:** GET /users/by-email (for example /user/by-email?email=user1@gmail.com)

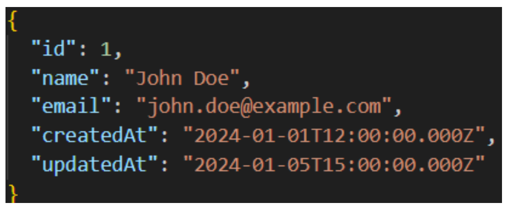


```
{
  "user": {
    "id": 1,
    "name": "User One",
    "email": "user1@gmail.com",
    "role": "user",
    "createdAt": "2024-01-01T12:00:00.000Z",
    "updatedAt": "2024-01-05T15:00:00.000Z"
  }
}
```

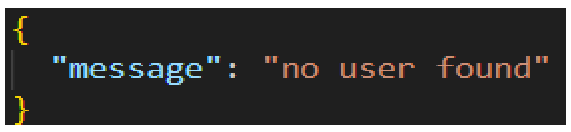


```
{
  "message": "no user found"
}
```

4. **Retrieve a user by their PK, excluding the "role" field from the response. (0.5 Grade)**
   - **URL:** GET /user/:id



```
{
  "id": 1,
  "name": "John Doe",
  "email": "john.doe@example.com",
  "createdAt": "2024-01-01T12:00:00.000Z",
  "updatedAt": "2024-01-05T15:00:00.000Z"
}
```

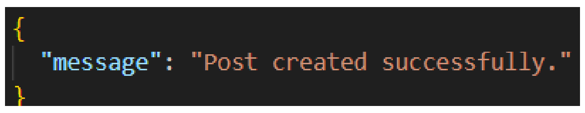


```
{
  "message": "no user found"
}
```

Assignment 8

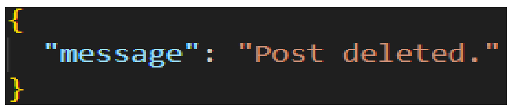
B- Post APIs (2 Grades):

1. **Create new Post (using new instance and save) (Get the post data from the body). (0.5 Grade)**
   - **URL:** POST /posts

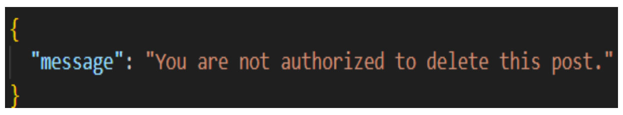


```
{
  "message": "Post created successfully."
}
```

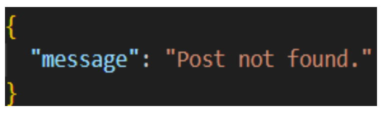
2. **Delete a post by its id (Ensure that only the owner of the post can perform this action) (0.5 Grade)**
   - **URL:** DELETE /posts/:postId



```
{
  "message": "Post deleted."
}
```

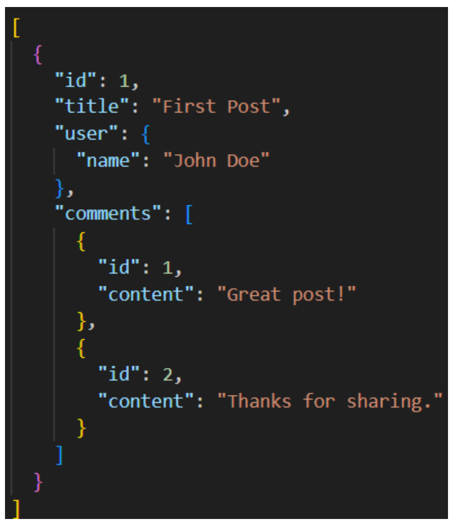


```
{
  "message": "You are not authorized to delete this post."
}
```



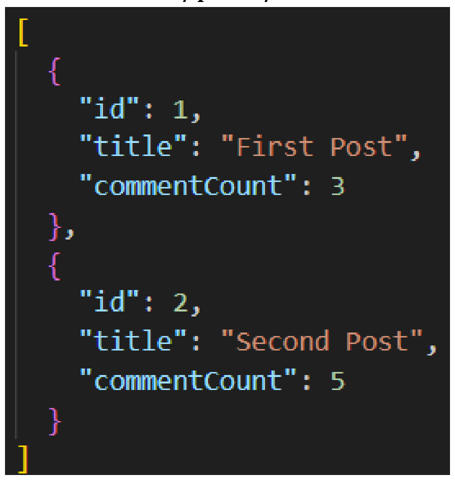
```
{
  "message": "Post not found."
}
```

3. **Retrieve all posts, including the details of the user who created each post and the associated comments. (Show only for the post the “id, title”, and for user “id, name”, and for the comments “id, content”) (0.5 Grade)**
   - **URL:** GET /posts/details



```
[
  {
    "id": 1,
    "title": "First Post",
    "user": {
      "name": "John Doe"
    },
    "comments": [
      {
        "id": 1,
        "content": "Great post!"
      },
      {
        "id": 2,
        "content": "Thanks for sharing."
      }
    ]
  }
]
```

4. **Retrieve all posts and count the number of comments associated with each post. (0.5 Grade)**
   - **URL:** GET /posts/comment-count

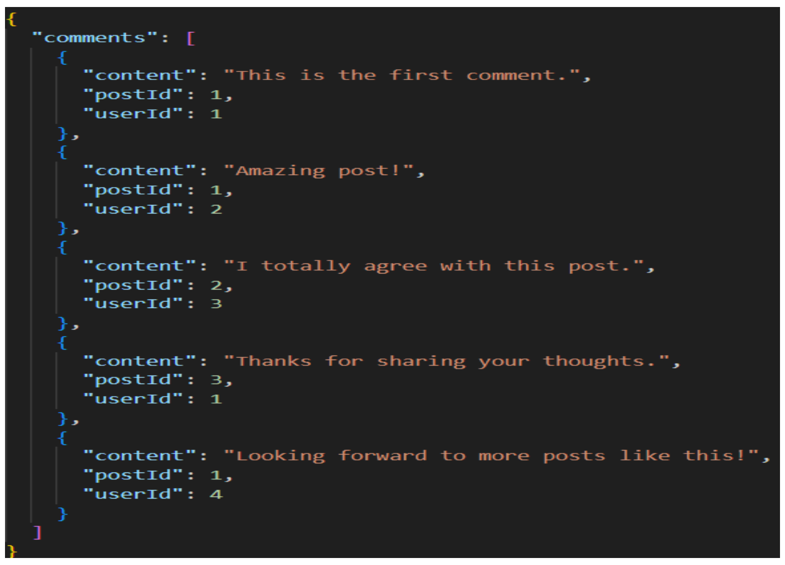


```
[
  {
    "id": 1,
    "title": "First Post",
    "commentCount": 3
  },
  {
    "id": 2,
    "title": "Second Post",
    "commentCount": 5
  }
]
```

C- Comment APIs (3 Grades):

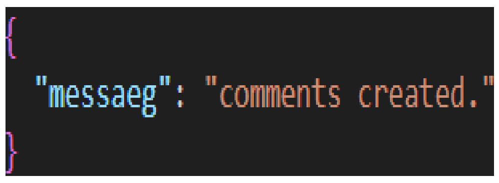
1. **Create a bulk of Comments. (0.5 Grade)**
   - **URL:** POST /comments

Input



```
{
  "comments": [
    {
      "content": "This is the first comment.",
      "postId": 1,
      "userId": 1
    },
    {
      "content": "Amazing post!",
      "postId": 1,
      "userId": 2
    },
    {
      "content": "I totally agree with this post.",
      "postId": 2,
      "userId": 3
    },
    {
      "content": "Thanks for sharing your thoughts.",
      "postId": 3,
      "userId": 1
    },
    {
      "content": "Looking forward to more posts like this!",
      "postId": 1,
      "userId": 4
    }
  ]
}
```

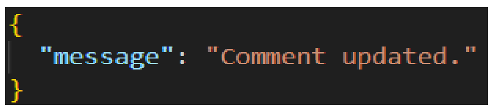
Output



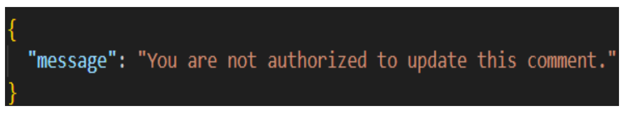
```
{
  "messaeg": "comments created."
}
```

Assignment 8

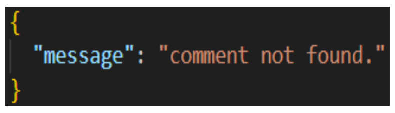
2. **Update the content of a specific comment by its ID. (Ensure that only the owner of the comment can perform this action) (The user id that wants to perform this action will be given in the body). (0.5 Grade)**
   - **URL:** PATCH /comments/:commentId
   - **Input from the body:** {“userId”:3, “content”:”updated”}



```
{
  "message": "Comment updated."
}
```



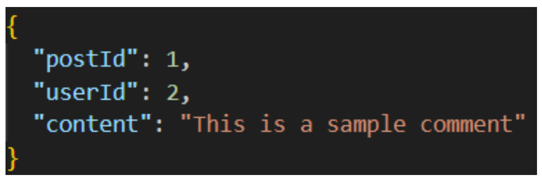
```
{
  "message": "You are not authorized to update this comment."
}
```



```
{
  "message": "comment not found."
}
```

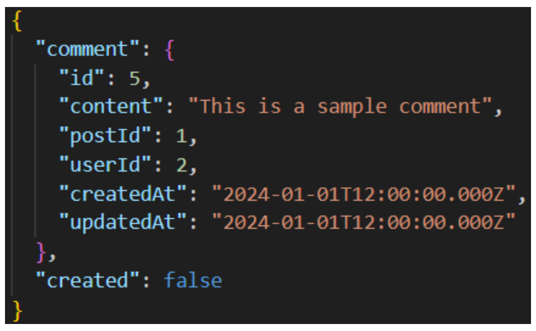
3. **find a comment for a specific post, user, and content. If the comment exists, return it, otherwise, create a new comment with the given details. (0.5 Grade)**
   - **URL:** POST /comments/find-or-create

Input



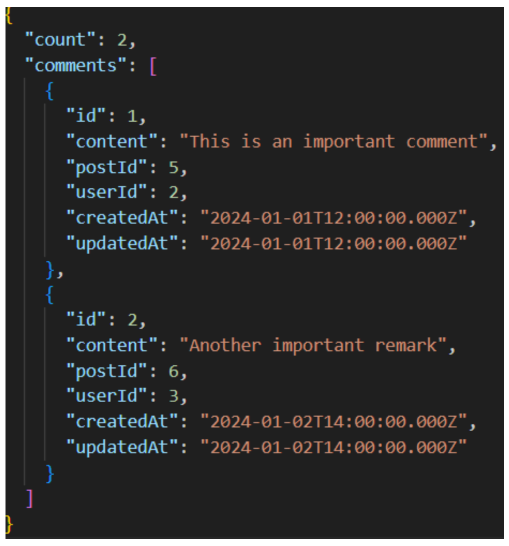
```
{
  "postId": 1,
  "userId": 2,
  "content": "This is a sample comment"
}
```

Output

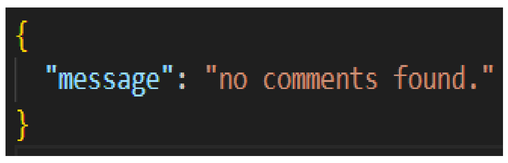


```
{
  "comment": {
    "id": 5,
    "content": "This is a sample comment",
    "postId": 1,
    "userId": 2,
    "createdAt": "2024-01-01T12:00:00.000Z",
    "updatedAt": "2024-01-01T12:00:00.000Z"
  },
  "created": false
}
```

4. **Retrieve all comments that contain a specific word in their content and return the number of comments matched (use find and count). (0.5 Grade)**
   - **URL:** GET /comments/search => (for example /comments/search?word=the)

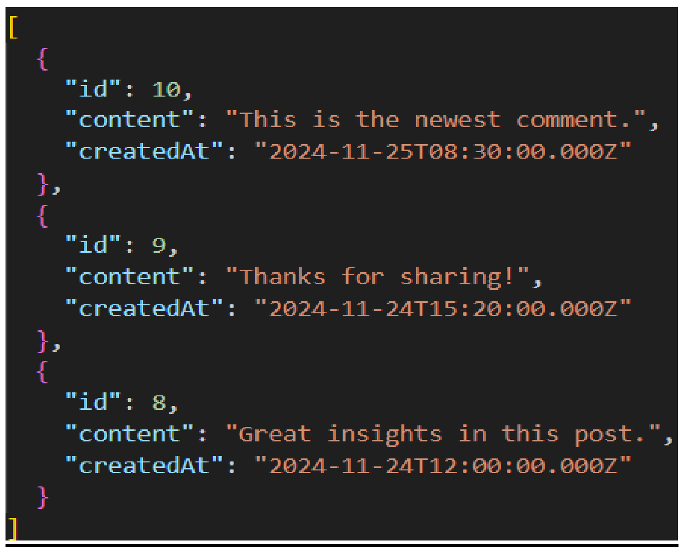


```
{
  "count": 2,
  "comments": [
    {
      "id": 1,
      "content": "This is an important comment",
      "postId": 5,
      "userId": 2,
      "createdAt": "2024-01-01T12:00:00.000Z",
      "updatedAt": "2024-01-01T12:00:00.000Z"
    },
    {
      "id": 2,
      "content": "Another important remark",
      "postId": 6,
      "userId": 3,
      "createdAt": "2024-01-02T14:00:00.000Z",
      "updatedAt": "2024-01-02T14:00:00.000Z"
    }
  ]
}
```



```
{
  "message": "no comments found."
}
```

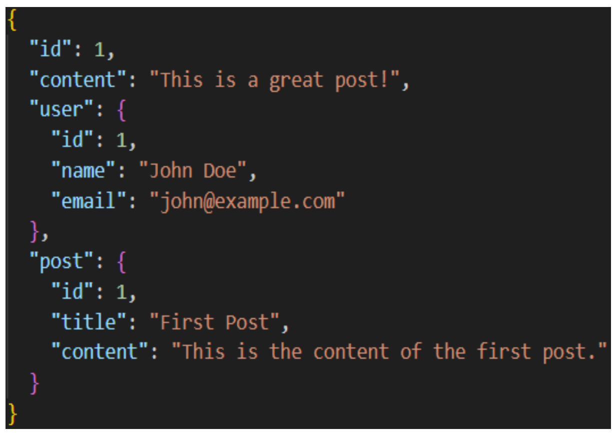
5. **Retrieve the 3 most recent comments for a specific post, ordered by creation date. (0.5 Grade)**
   - **URL:** GET /comments/newest/:postId



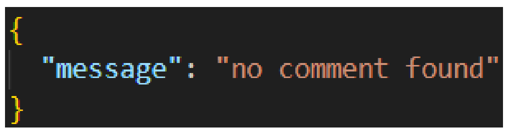
```
[
  {
    "id": 10,
    "content": "This is the newest comment.",
    "createdAt": "2024-11-25T08:30:00.000Z"
  },
  {
    "id": 9,
    "content": "Thanks for sharing!",
    "createdAt": "2024-11-24T15:20:00.000Z"
  },
  {
    "id": 8,
    "content": "Great insights in this post.",
    "createdAt": "2024-11-24T12:00:00.000Z"
  }
]
```

Assignment 8

6. **Get Specific Comment By PK with User and Post Information. (0.5 Grade)**
   - **URL:** GET /comments/details/:id



```
{
  "id": 1,
  "content": "This is a great post!",
  "user": {
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com"
  },
  "post": {
    "id": 1,
    "title": "First Post",
    "content": "This is the content of the first post."
  }
}
```



```
{
  "message": "no comment found"
}
```

⚠️ Important Notes about postman

1. **Name the endpoint with a meaningful name like 'Add User', not dummy names.**
2. **Save your changes on each request (ctrl+s).**
3. **Include the Postman collection link (export your Postman collection) in the email with your assignment link**

Bonus (2 Grades)

How to deliver the bonus?

1- Solve the problem Remove Element on **LeetCode**

2- Inside your assignment folder, create a **SEPARATE FILE** and name it "bonus.js"

3- Copy the code that you have submitted on the website inside "bonus.js" file