PiazzaApi Documentation

Packages Details

Introduction:

The "piaaza-api" is a Node.js application designed to serve as the backend for a Piazza platform. It provides functionalities for user authentication, data validation, and interaction with a MongoDB database. This application is built using the Express.js framework, making it easy to create robust and scalable web APIs.

Packages Used:

1. **cookie-parser**:

   Description: Middleware for parsing cookies attached to the client's request.
   Purpose: Used for handling HTTP cookies, often used for user authentication.

2. **cors**:

   Description: Middleware for enabling Cross-Origin Resource Sharing (CORS).
   Purpose: Allows or restricts requested resources on a web server based on where the HTTP request was initiated.

3. **dotenv**:

   Description: Zero-dependency module that loads environment variables from a .env file.
   Purpose: Simplifies environment variable management, particularly useful for configuration settings.

4. **express**:

   Description: Web application framework for Node.js.
   Purpose: Used to build scalable and maintainable web applications by providing a set of features for web and mobile applications.

5. **express-validator**:

Description: A set of Express.js middlewares that wraps validator.js validator and sanitizer functions.
Purpose: Simplifies the validation and sanitization of incoming requests.

6. **google-auth-library**:

Description: Library for handling Google's OAuth 2.0 authentication.
Purpose: Allows the application to integrate Google Sign-In for user authentication.

7. **jsonwebtoken**:

Description: Library for creating and verifying JSON Web Tokens (JWT).
Purpose: Used for creating secure tokens for user authentication and authorization.

8. **moment**:

Description: A library for parsing, validating, manipulating, and formatting dates.
Purpose: Simplifies working with dates and times in the application.

9. **mongodb**:

Description: The official MongoDB driver for Node.js.
Purpose: Enables the application to interact with a MongoDB database, performing operations like CRUD (Create, Read, Update, Delete).

10. **mongoose**:

Description: Elegant MongoDB object modeling for Node.js.
Purpose: Provides a schema-based solution to model the application's data and interact with MongoDB.

11. **selfsigned**:

Description: Generates self-signed certificates.
Purpose: Useful for local development where HTTPS is required.

These packages collectively empower the "piaaza-api" to handle user authentication, data validation, and database operations effectively.

Environment Variables Details

1 **MONGODB_ATLAS**:

- **Description**: This is the URI (Uniform Resource Identifier) for your MongoDB Atlas database.
- **How to Obtain**:
  - Go to [MongoDB Atlas](https://www.mongodb.com/cloud/atlas).
  - Sign in or create an account.
  - Create a new cluster and obtain the connection string.

JWT_SECRET:

Description: Secret key used to sign JSON Web Tokens (JWT) for authentication.
How to Obtain:
Generate a strong, random string. This secret should be kept confidential.

PORT:

Description: The port on which your backend server will run.
Default Value: 8000 (You can change it if needed).

CLIENT_ID:

Description: Google OAuth (google-auth-library) client ID for authentication.
How to Obtain:
Go to [Google Cloud Console](https://console.cloud.google.com/).
Create a new project and configure the OAuth consent screen.
Create credentials and obtain the client ID.

CLIENT_SECRET:

Description: Google OAuth client secret for authentication.
How to Obtain:
- Same as CLIENT_ID, you can find it in the credentials created on the Google Cloud Console.

Note: Ensure that you keep your `JWT_SECRET`, `CLIENT_ID`, and `CLIENT_SECRET` confidential. Never expose them in your frontend code or public repositories. Use environment variables or a secure configuration management system to handle these sensitive values.

Auth Endpoint

OAuth2 Google Authentication - Initiate Authorization

Note: To make authentication via google u need to integrate this end point in “signIn with google " button which will redirect you to google authentication process.

Endpoint Documentation: /api/auth/

Endpoint Description:

This endpoint initiates the OAuth2 authorization process for Google authentication. It returns the authorization URL that the client application should redirect the user to for authentication.

Request:

- Method: POST
- Headers:
  - Access-Control-Allow-Origin: http://localhost:5173
  - Access-Control-Allow-Credentials: true
  - Referrer-Policy: no-referrer-when-downgrade

Response:

- Body:{

"url" "Authorization URL"

}

Usage:

Client Application Integration:
- When a user intends to log in via Google, make a POST request to this endpoint from the client application.

Authorization URL:
- Extract the url from the response.
- Redirect the user to the provided authorization URL.

User Authentication:
- The user will be prompted to log in to their Google account (if not already logged in) and grant necessary permissions.

Callback Handling:
- After successful authentication, Google will redirect the user back to the specified redirect URI (config.google.redirectUri).

Server-Side Handling:
- The server should handle the callback, exchanging the received authorization code for access and refresh tokens.

Further Operations:
- Once tokens are obtained, the server can use them to access user data from Google.

Additional Notes:

- `Ensure that the client application is allowed to make requests from http://localhost:5173 (configured in Access-Control-Allow-Origin header).`

Sample CURL Request:

```
curl -X POST https://localhost:8000/api/auth/ -H "Access-Control-Allow-
Origin: http://localhost:5173" -H "Access-Control-Allow-Credentials: true"
-H "Referrer-Policy: no-referrer-when-downgrade"
```

Post Endpoint

Create Post:

Endpoint Documentation: `/api/post`

Endpoint Description:

This endpoint is responsible for creating a new post. It requires authentication, and the newly created post is associated with the authenticated user. Optionally, the post can have an expiration time, after which it will be automatically marked as "Expired." The endpoint returns details of the created post upon successful creation.

Request:

- Method: POST
- URL: `http://localhost:8000/api/post`
- Headers:
    - Content-Type: application/json
    - Authorization: Bearer `<access_token>`
        - (Include a valid access token obtained during user authentication)
- Body:

```
"title"  "Sample Post"
"topics"   "Technology"  "Programming"
"body"  "Lorem ipsum dolor sit amet, consectetur adipiscing elit..."
"expirationTime"  "2023-12-31T23:59:59.999Z"
```

-

Request Validation:

- The request must include a valid JWT access token in the Authorization header for authentication.
- The request body should contain the required fields: `title`, `topics`, `body`, and optionally, `expirationTime`.

Response:

- Status Code: 201 Created
- Body:

```
"success"  true
"message"  "Post created successfully"
"newPost"
"_id"  "post_id"
"title"  "Sample Post"
"topics"  "Technology"  "Programming"
"body"  "Lorem ipsum dolor sit amet, consectetur adipiscing elit..."
"expirationTime"  "2023-12-31T23:59:59.999Z"
"owner"  "user_id"
"status"  "Live"
"createdAt"  "timestamp"
"updatedAt"  "timestamp"
```

-

Response Details:

- If the post creation is successful, it returns a success message and details of the newly created post.
- The `newPost` object contains the post's `_id`, `title`, `topics`, `body`, `expirationTime`, `owner`, `status`, `createdAt`, and `updatedAt`.

Post Expiration:

- If `expirationTime` is provided, a background process is initiated to automatically update the post's status to "Expired" after the defined expiration time.

Error Handling:

- If there is an authentication failure (no access token or invalid token), a 401 Unauthorized response is returned.
- If the request body is missing required fields or there is an internal server error during post creation, a 500 Internal Server Error response is returned.

Security Measures:

- Authentication is required to create a post, ensuring that only authenticated users can create posts.
- Background process for post expiration provides automated handling of post status updates.

Sample CURL Request:

```
curl -X POST http://localhost:8000/api/post -H "Content-Type: application/json" -H
"Authorization: Bearer <access_token>"    '{"title": "Sample Post", "topics":
["Technology", "Programming"], "body": "Lorem ipsum...", "expirationTime": "2023-
12-31T23:59:59.999Z"}'
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.
- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Get All Post by Query:

Endpoint Documentation: `/api/post/`

Endpoint Description:

This endpoint retrieves all posts related to a specific topic. The topic is extracted from the request query parameters. The endpoint validates the provided topic against a predefined list of allowed topics and returns the posts along with their details.

Request:

- Method: GET
- URL: `http://localhost:8000/api/post/`
    - Include the desired topic as a query parameter (e.g., `http://localhost:8000/api/post/?topic=Politics`)

Request Parameters:

- Query Parameters:
    - `topic` (string): The topic for which posts are to be retrieved.

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"posts"

"_id"  "post_id_1"
"title"  "Sample Post 1"
"topics"   "Politics"
"body"  "Lorem ipsum dolor sit amet..."
"createdAt"  "timestamp"
"updatedAt"  "timestamp"



"_id"  "post_id_2"
"title"  "Sample Post 2"
"topics"   "Politics"
"body"  "Lorem ipsum dolor sit amet..."
"createdAt"  "timestamp"
"updatedAt"  "timestamp"
```

Response Details:

- If the request is successful, it returns a list of posts related to the specified topic.
- Each post includes details such as `_id`, `title`, `topics`, `body`, `createdAt`, and `updatedAt`.

Request Validation:

- The endpoint validates if the topic is provided and is one of the allowed topics (`['Politics', 'Health', 'Sport', 'Tech']`).
- If the topic is missing or invalid, a 400 Bad Request response is returned with a corresponding error message.

Error Handling:

- If there is an internal server error during post retrieval, a 500 Internal Server Error response is returned.
- If the provided topic is missing or invalid, a 400 Bad Request response is returned with a corresponding error message.

Security Measures:

- The endpoint ensures that only valid topics are accepted, preventing potential misuse or unauthorized access.

Sample CURL Request:

```
curl -X GET http://localhost:8000/api/post/?topic=Politics
```

Notes:

- Ensure that the server is running locally at [`http://localhost:8000`](http://localhost:8000).

Update Post:

Endpoint Documentation: `/api/post/:id`

Endpoint Description:

This endpoint allows the owner of a post to update its details. The post ID is extracted from the request parameters. The authenticated user is checked to ensure they are the owner of the post, and if so, the post details are updated based on the provided values in the request body.

Request:

- Method: PUT
- URL: `http://localhost:8000/api/post/:id`
  - Replace `:id` with the ID of the post to be updated (e.g., `http://localhost:8000/api/post/12345`)

Request Parameters:

- Path Parameters:
  - `:id` (string): The ID of the post to be updated.

Request Body:

- The request body can include any combination of the following fields to update the post:
  - `title` (string): The new title of the post.
  - `topics` (array): The new topics associated with the post.
  - `body` (string): The new body/content of the post.
  - `expirationTime` (string): The new expiration time for the post.

```
"title"  "New Title"
"topics"   "Politics"  "Technology"
"body"  "Updated content..."
"expirationTime"  "2023-12-31T23:59:59.999Z"
```

-

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
```

```
"message"  "Post updated successfully"
"updatedValues"
"title"  "New Title"
"topics"   "Politics"  "Technology"
"body"  "Updated content..."
"expirationTime"  "2023-12-31T23:59:59.999Z"
```

-

Response Details:

- If the request is successful, it returns a success message along with the updated values of the post.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.
- It verifies if the authenticated user is the owner of the post. If not, a 403 Forbidden response is returned.

Error Handling:

- If there is an internal server error during the update process, a 500 Internal Server Error response is returned.
- If the post is not found or the authenticated user is not the owner, appropriate error responses (404 or 403) are returned.

Security Measures:

- Only the owner of the post is allowed to update its details, preventing unauthorized modifications.

Sample CURL Request:

```
curl -X PUT http://localhost:8000/api/post/12345 -H "Content-Type:
application/json" -H "Authorization: Bearer <access_token>" -d '{"title": "New
Title", "topics": ["Politics", "Technology"], "body": "Updated content...",
"expirationTime": "2023-12-31T23:59:59.999Z"}'
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.

- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Delete Post:

Endpoint Documentation: `/api/post/:id`

Endpoint Description:

This endpoint allows the owner of a post to delete it. The post ID is extracted from the request parameters, and the authenticated user is verified to ensure they have the necessary permissions. The endpoint also handles the deletion of associated comments, likes, and dislikes before deleting the post.

Request:

- Method: DELETE
- URL: `http://localhost:8000/api/post/:id`
    - Replace `:id` with the ID of the post to be deleted (e.g., `http://localhost:8000/api/post/12345`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post to be deleted.

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"message"  "Post deleted successfully"
```

-

Response Details:

- If the request is successful, it returns a success message indicating that the post has been deleted.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.
- It verifies if the authenticated user is the owner of the post. If not, a 403 Forbidden response is returned.

Post Deletion Process:

- The endpoint deletes associated comments and reactions (likes/dislikes) before deleting the post to maintain data integrity.

Error Handling:

- If there is an internal server error during the deletion process, a 500 Internal Server Error response is returned.
- If the post is not found, the authenticated user is not the owner, or there are validation errors, appropriate error responses (404, 403, or 400) are returned.

Security Measures:

- Only the owner of the post is allowed to delete it, preventing unauthorized deletions.
- Associated comments, likes, and dislikes are deleted to maintain data consistency.

Sample CURL Request:

```
curl -X DELETE http://localhost:8000/api/post/12345 -H "Authorization: Bearer
<access_token>"
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.
- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Get Post by id:

Endpoint Documentation: `/api/post/:id`

Endpoint Description:

This endpoint retrieves the details of a specific post based on the provided post ID. The post ID is extracted from the request parameters, and the endpoint returns information about the post if it exists.

Request:

- Method: GET
- URL: `http://localhost:8000/api/post/:id`
    - Replace `:id` with the ID of the post to be retrieved (e.g., `http://localhost:8000/api/post/12345`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post to be retrieved.

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"post"
"_id"  "post_id"
"title"  "Sample Post"
"topics"  "Technology"  "Programming"
"body"  "Lorem ipsum dolor sit amet..."
"owner"
"_id"  "user_id"
"username"  "john_doe"

"likes"
"dislikes"
"comments"
```

```
 "createdAt"   "timestamp"
 "updatedAt"   "timestamp"
```

●

Response Details:

- If the request is successful, it returns details of the specified post, including `_id`, `title`, `topics`, `body`, `owner` (with `username`), `likes`, `dislikes`, `comments`, `createdAt`, and `updatedAt`.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.

Error Handling:

- If there is an internal server error during post retrieval, a 500 Internal Server Error response is returned.
- If the post is not found, a 404 Not Found response is returned.

Sample CURL Request:

```
curl -X GET http://localhost:8000/api/post/12345
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.

Reaction

Create Reaction:

Endpoint Documentation: `/api/post/:id`

Endpoint Description:

This endpoint allows users to create, update, or remove reactions (likes/dislikes) on a specific post. The post ID is extracted from the request parameters, and the type of reaction is obtained from the request body. The endpoint handles various scenarios, such as checking if the post exists, if it is live, and whether the user has already reacted to it.

Request:

- Method: POST
- URL: `http://localhost:8000/api/post/:id`
    - Replace `:id` with the ID of the post to be reacted to (e.g., `http://localhost:8000/api/post/12345`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post to be reacted to.

Request Body:

- The request body should include the type of reaction:
    - `type` (string): The type of reaction, either `'like'` or `'dislike'`.

```
"type"  "like"
```

-

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"message"  "Reaction added successfully"
```

-

Response Details:

- If the request is successful, it returns a success message indicating that the reaction has been added, updated, or removed.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.
- It verifies if the post is live. If not, a 400 Bad Request response is returned, indicating that reactions cannot be added to an expired post.

Reaction Handling:

- If the user has previously reacted to the post, the endpoint handles updating or removing the existing reaction based on the new reaction type.
- If the user has not reacted before, a new reaction is created.

Error Handling:

- If there is an internal server error during the reaction process, a 500 Internal Server Error response is returned.
- If the post is not found or is expired, appropriate error responses (404 or 400) are returned.

Security Measures:

- The endpoint ensures that reactions can only be added to live posts, preventing interactions with expired posts.
- User authentication is assumed to be handled by middleware (`req.user`), ensuring that only authenticated users can create reactions.

Sample CURL Request:

```
curl -X POST http://localhost:8000/api/post/12345 -H "Content-Type: application/json" -H
"Authorization: Bearer <access_token>" -d '{"type": "like"}'
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.
- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Get All Reactions:

Endpoint Documentation: `/api/post/list/:id`

Endpoint Description:

This endpoint retrieves the list of reactions (likes/dislikes) associated with a specific post. The post ID is extracted from the request parameters, and the endpoint returns details of each reaction, including the author, type, time left, and timestamp.

Request:

- Method: GET
- URL: `http://localhost:8000/api/post/list/:id`
  - Replace `:id` with the ID of the post to retrieve reactions for (e.g., `http://localhost:8000/api/post/list/12345`)

Request Parameters:

- Path Parameters:
  - `:id` (string): The ID of the post to retrieve reactions for.

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"reactions"

"author"  "user_id"
"type"  "like"
"timeleft"  "HH:mm:ss"
"timestamp"  "timestamp"
```

-

Response Details:

- If the request is successful, it returns details of reactions associated with the specified post.
- Each reaction includes `author` (user ID), `type` (like or dislike), `timeleft` (time left for the post to expire), and `timestamp` (when the reaction was added).

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.

Error Handling:

- If there is an internal server error during reaction retrieval, a 500 Internal Server Error response is returned.
- If the post is not found, a 404 Not Found response is returned.

Sample CURL Request:

```
curl -X GET http://localhost:8000/api/post/list/12345
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.

Comment

Create Comment:

Endpoint Documentation: `/api/post/:id`

Endpoint Description:

This endpoint allows users to create comments on a specific post. The post ID is extracted from the request parameters, and the comment body is obtained from the request body. The endpoint handles various scenarios, such as checking if the post exists, if it is live, and ensuring users cannot comment on their own posts.

Request:

- Method: POST
- URL: `http://localhost:8000/api/post/:id`
    - Replace `:id` with the ID of the post to comment on (e.g., `http://localhost:8000/api/post/12345`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post to comment on.

Request Body:

- The request body should include the comment body:
    - `body` (string): The content of the comment.

```
"body"  "This is a great post!"
```

-

Response:

- Status Code: 201 Created
- Body:

- json
-

```
"success"  true
"message"  "Comment added successfully"
"comment"

"author"  "user_id"
"body"  "This is a great post!"
"timeleft"  "HH:mm:ss"
"timestamp"  "timestamp"
```

-

Response Details:

- If the request is successful, it returns a success message indicating that the comment has been added. The response also includes an array of comments associated with the specified post.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.
- It verifies if the post is live. If not, a 400 Bad Request response is returned, indicating that comments cannot be added to an expired post.
- Users cannot comment on their own posts. If the authenticated user is the owner of the post, a 400 Bad Request response is returned.

Comment Creation Process:

- The endpoint creates a new comment object with details such as the author, comment body, time left for the post to expire, and timestamp.
- The new comment is added to the post's `comments` array.
- The comment count for the post is incremented.

Error Handling:

- If there is an internal server error during comment creation, a 500 Internal Server Error response is returned.
- If the post is not found, is expired, or the user is the owner, appropriate error responses (404, 400) are returned.

Security Measures:

- Users cannot comment on their own posts, preventing self-interaction.
- Only authenticated users can create comments.

Sample CURL Request:

bash

```
                                                          "Content-Type:
application/json"    "Authorization: Bearer <access_token>"    '{"body": "This is
a great post!"}'
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.
- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Update Comment:

Endpoint Documentation:

`/api/post/comment/:id/:commentId`

Endpoint Description:

This endpoint allows users to update a specific comment on a post. The post ID and comment ID are extracted from the request parameters, and the updated comment body is obtained from the request body. The endpoint handles various scenarios, such as checking if the post exists, if it is live, and ensuring users can only update their own comments.

Request:

- Method: PUT
- URL: `http://localhost:8000/api/post/comment/:id/:commentId`
    - Replace `:id` with the ID of the post, and `:commentId` with the ID of the comment to be updated (e.g., `http://localhost:8000/api/post/comment/12345/67890`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post containing the comment.
    - `:commentId` (string): The ID of the comment to be updated.

Request Body:

- The request body should include the updated comment body:
    - `body` (string): The new content of the comment.

```
"body"  "Updated comment text."
```

-

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"message"  "Comment updated successfully"
"updatedComment"
"_id"  "comment_id"
"author"  "user_id"
"body"  "Updated comment text."
"timeleft"  "HH:mm:ss"
"timestamp"  "timestamp"
```

-

Response Details:

- If the request is successful, it returns a success message indicating that the comment has been updated. The response also includes details of the updated comment.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.
- It verifies if the post is live. If not, a 400 Bad Request response is returned, indicating that comments cannot be updated on an expired post.
- Users can only update their own comments. If the authenticated user is not the author of the comment, a 400 Bad Request response is returned.

Comment Update Process:

- The endpoint finds the comment in the post's `comments` array based on the provided comment ID.
- It checks if the authenticated user is the author of the comment.
- If the conditions are met, it updates the comment body with the new content.
- The updated post is saved, and details of the updated comment are returned in the response.

Error Handling:

- If there is an internal server error during comment update, a 500 Internal Server Error response is returned.
- If the post is not found, is expired, the user is the owner, or the user is not the author of the comment, appropriate error responses (404, 400) are returned.

Security Measures:

- Users can only update their own comments, preventing unauthorized modifications.
- Only authenticated users can update comments.

Sample CURL Request:

```
curl -X PUT http://localhost:8000/api/post/comment/12345/67890 -H "Content-Type:
application/json" -H "Authorization: Bearer <access_token>" -d '{"body": "Updated
comment text."}'
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.
- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Delete Comment:

Endpoint Documentation:

`/api/post/comment/:id/:commentId`

Endpoint Description:

This endpoint allows users to delete a specific comment on a post. The post ID and comment ID are extracted from the request parameters. The endpoint ensures that the comment exists, the authenticated user is the author of the comment, and then deletes the comment from the post.

Request:

- Method: DELETE
- URL: `http://localhost:8000/api/post/comment/:id/:commentId`
    - Replace `:id` with the ID of the post, and `:commentId` with the ID of the comment to be deleted (e.g., `http://localhost:8000/api/post/comment/12345/67890`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post containing the comment.
    - `:commentId` (string): The ID of the comment to be deleted.

Response:

- Status Code: 200 OK
- Body:

```
"success"  true
"message"  "Comment deleted successfully"
```

-

Response Details:

- If the request is successful, it returns a success message indicating that the comment has been deleted.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.
- It verifies if the comment exists in the post's comments array. If not, a 404 Not Found response is returned.
- The authenticated user must be the author of the comment; otherwise, a 403 Forbidden response is returned.

Comment Deletion Process:

- The endpoint finds the comment in the post's `comments` array based on the provided comment ID.
- It checks if the authenticated user is the author of the comment.
- If the conditions are met, it removes the comment from the post's comments array.
- The comment count for the post is decremented.
- The updated post is saved.

Error Handling:

- If there is an internal server error during comment deletion, a 500 Internal Server Error response is returned.
- If the post is not found, the comment is not found, or the user is not the author of the comment, appropriate error responses (404, 403) are returned.

Security Measures:

- Users can only delete their own comments, preventing unauthorized deletion.
- Only authenticated users can delete comments.

Sample CURL Request:

```
curl -X DELETE http://localhost:8000/api/post/comment/12345/67890 -H
"Authorization: Bearer <access_token>"
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.
- Replace `<access_token>` with a valid JWT access token obtained through user authentication.

Get All comments:

Endpoint Documentation: `/api/post/list/:id`

Endpoint Description:

This endpoint retrieves all comments for a specific post. The post ID is extracted from the request parameters, and the endpoint checks if the post exists. If the post exists, it returns a list of all comments associated with that post.

Request:

- Method: GET
- URL: `http://localhost:8000/api/post/list/:id`
    - Replace `:id` with the ID of the post (e.g., `http://localhost:8000/api/post/list/12345`)

Request Parameters:

- Path Parameters:
    - `:id` (string): The ID of the post for which comments are to be retrieved.

Response:

- Status Code: 200 OK
- Body:

```
 "success"  true
 "comments"
```

```
"_id"  "comment_id_1"
"author"  "user_id_1"
"body"  "Comment text 1"
"timeleft"  "HH:mm:ss"
"timestamp"  "timestamp_1"


"_id"  "comment_id_2"
"author"  "user_id_2"
"body"  "Comment text 2"
"timeleft"  "HH:mm:ss"
"timestamp"  "timestamp_2"
```

●

Response Details:

- If the request is successful and the post is found, it returns a success message along with an array of comments associated with the post.

Request Validation:

- The endpoint checks if the post with the provided ID exists. If not, a 404 Not Found response is returned.

Comments Retrieval Process:

- The endpoint finds the post in the database based on the provided post ID.
- If the post is found, it retrieves all comments associated with that post.

Error Handling:

- If there is an internal server error during the comments retrieval process, a 500 Internal Server Error response is returned.
- If the post is not found, a 404 Not Found response is returned.

Security Measures:

- No specific security measures are mentioned in the provided controller. However, it is assumed that users need appropriate permissions to access post comments.

Sample CURL Request:

```
curl -X GET http://localhost:8000/api/post/list/12345
```

Notes:

- Ensure that the server is running locally at `http://localhost:8000`.