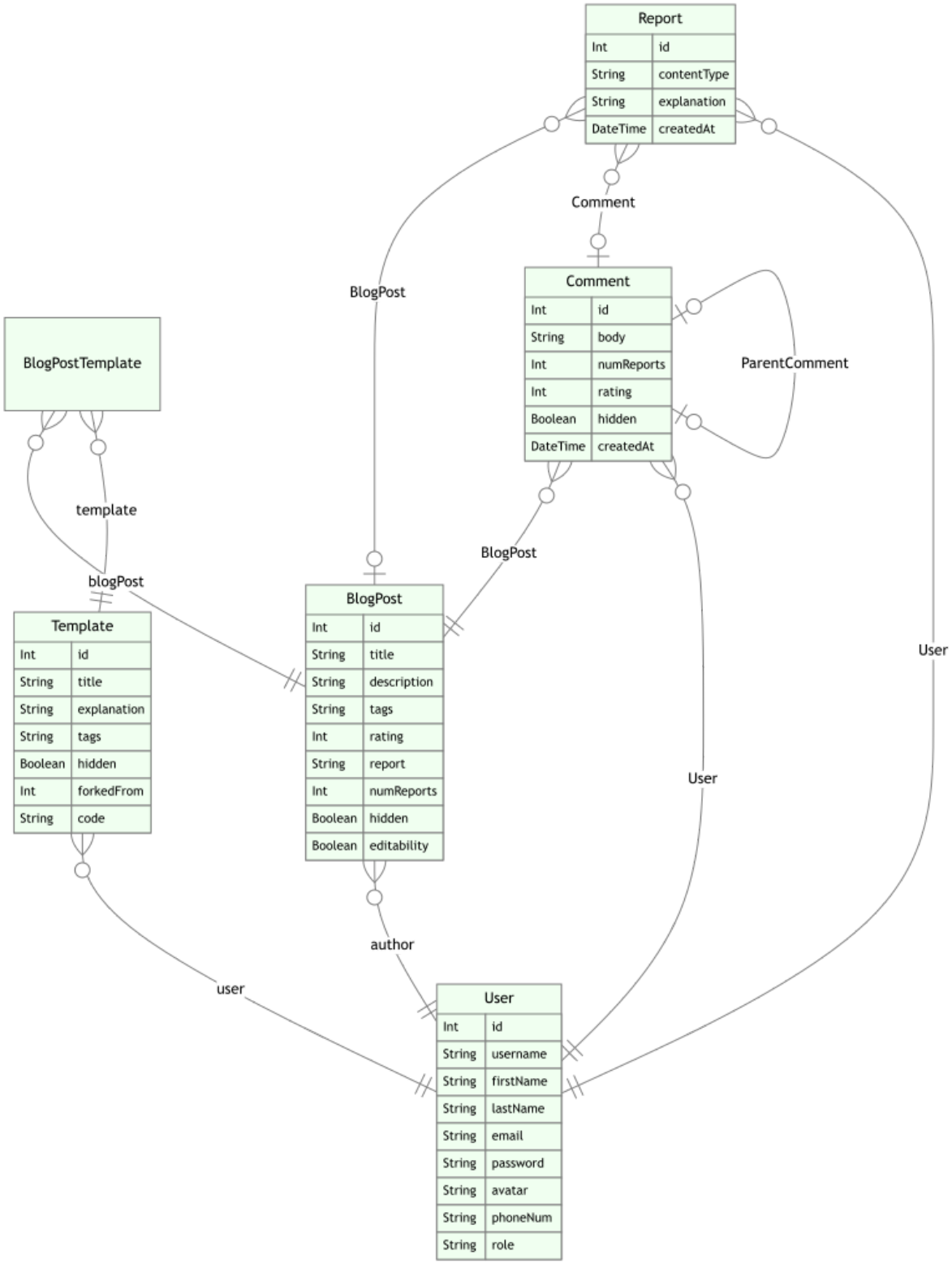
Scriptorium: Project Part 1 Documentation

CSC309H1F

Professor Kianoosh Abbasi

Daniel Kaloshi (kaloshid, 1009520751)
Parth Vats (vatspart, 1007294642)
Mykola Zhuk(zhukmyko, 1007315512)

Database Diagram



Admin user credentials created during startup.sh:

Username**:** ***admin***

Password**:** ***adminpassword***

API Documentation

/api/users/signup

Description: This endpoint allows users to sign up by providing their username, first name, last name, email, password, avatar URL, and phone number.

Allowed Methods: POST

Request:

Payload Structure:

- `username` (string) - The username of the user.
- `firstName` (string) - The first name of the user.
- `lastName` (string) - The last name of the user.
- `email` (string) - The email address of the user.
- `password` (string) - The password for the user account.
- `avatar` (string) - The URL of the user's avatar.
- `phoneNum` (string) - The phone number of the user.
- Can also include role (string) - The user role (defaults to USER)

Example: {

```
"username": "danny",

"firstName": "Daniel",

"lastName": "Kal",

"email": "danny@gmail.com",

"password": "123",
```

```
    "avatar": "http://localhost:3000/avatars/avatar1.png",

    "phoneNum": "+123456789"

}
```


Response:

The response body will be in JSON format and will include the following fields:

- `id` (number) - The unique identifier for the user.
- `username` (string) - The username of the user.
- `firstName` (string) - The first name of the user.
- `lastName` (string) - The last name of the user.
- `email` (string) - The email address of the user.
- `password` (string) - The password for the user account.
- `avatar` (string) - The URL of the user's avatar.
- `phoneNum` (string) - The phone number of the user.
- `role` (string) - The role of the user.

Example:


```
{

    "id": 1,

    "username": "danny",

    "firstName": "Daniel",

    "lastName": "Kal",

    "email": "danny@gmail.com",

    "password": "$2b$10$g2IVKk5whkGjJzlEr4nbPO.JMHq7BHF5rs7zwibrMImmUmjsA3MQO",

    "avatar": "http://localhost:3000/avatars/avatar1.png",

    "phoneNum": "+123456789",

    "role": "USER"

}
```

/api/users/login

Description: This endpoint is used to authenticate a user and obtain access and refresh tokens.

Allowed Methods: POST

Request:

The request should include a JSON payload in the raw request body type with the following parameters:

- `username` (string): The username of the user.
- `password` (string): The password of the user.

Example:

```
{

    "username": "danny",

    "password": "123"

}
```

Response:

The response will have a status code of 200 and a content type of application/json.

Example:

```
{

"accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJkYW5ueSIsInJvbGUiOiJV
```

U0VSIiwiZXhwaXJlc0F0IjoiMTczMDUxODgyNjk2MDNoIiwiaWF0IjoxNzMwNTE4ODI2LCJleHAiOjE3MzA1Mjk2MjZ9.9AWtCCalqIPSF7CBgorxH0ENkBlKoAx9l7dE-s0YJK0",

"refreshToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJkYW5ueSIsInJvbGUiOiJVU0VSIiwiZXhwaXJlc0F0IjoiMTczMDUxODgyNjk2MDNoIiwiaWF0IjoxNzMwNTE4ODI2LCJleHAiOjE3MzA3NzgwMjZ9.Jd3ybt61VJOCx5yM_PCvvojEv-_ISPn2_xFCZhATiVA"

/api/users/refresh

Description: This endpoint is used to refresh the user access token by providing the refresh token.

Allowed Methods: POST

Request:

Payload Structure:

- `refreshToken` (string, required): The refresh token used to obtain a new access token.

Must also provide Authorization (as seen below)

`Example:`

Key: Authorization, Value: Bearer {{accessToken}}

```
{
    "refreshToken": "{{refreshToken}}"
}
```

Response:

The response will be a JSON object with the following schema:

JSON

{

"accessToken": "string"

}

- `accessToken` (string): The new access token obtained after refreshing.

Example:


```
{

    "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJkYW5ueSIsInJvbGUiOiJV
U0VSIiwiaWF0IjoxNzMwNTIwNTY4LCJleHAiOjE3MzA1MzEzNjh9.gJ0zNwMERCBNRkBp34xVvK9yCTPH398Ky
IHpIT-yjH0"

}
```

/api/users/edit-profile

Description: This endpoint allows the user to update their profile information via an HTTP PUT request.

Allowed Methods: PUT

Request:

The user can include any of email, password, avatar, and phoneNum in their request body. All other user attributes will remain unchanged. The request should be sent to http://localhost:3000/api/users/edit-profile. The request body should be in raw JSON format.

Must also provide Authorization (as seen below).

Example:

Auth Type: Bearer Token, Token: {{accessToken}}

```
{
    "email": "danny@yahoo.com",
    "avatar": "http://localhost:3000/avatars/avatar2.png"
}
```

Response Example:

```
{
```

```
    "id": 1,
    "username": "danny",
    "firstName": "Daniel",
    "lastName": "Kal",
    "email": "danny@yahoo.com",
    "password": "$2b$10$g2IVKk5whkGjJzlEr4nbPO.JMHq7BHF5rs7zwibrMImmUmjsA3MQO",
    "avatar": "http://localhost:3000/avatars/avatar2.png",
    "phoneNum": "+123456789",
    "role": "USER"
}
```

/api/blogPost

Description: This endpoint allows the user to create a new blog post by sending a POST request to the specified URL. The request should include the title, description, tags, and an array of template IDs linked to the blog post (optional).

The GET request retrieves blog posts based on specified tags, title, description, and/or ‘links’ to templates.

Allowed Methods: POST, GET

Request (for POST):

Payload Structure:

- title (string): The title of the blog post.
- description (string): The content or description of the blog post.
- tags (string): Tags associated with the blog post.
- linkToTemplates (array of integers): An array of template IDs linked to the blog post.

Must also provide Authorization (as seen below).

Example:

Auth Type: Bearer Token, Token: {{accessToken}}

```
{

    "title": "Blog Post 1",
```

```
    "description": "This is my first blog post.",
    "tags": "blog, intro",
    "linkToTemplates": [1, 2]
}
```

Response (for POST):

The response will be a JSON object with the following properties:

- id (number): The unique identifier for the blog post.
- title (string): The title of the blog post.
- description (string): The content or description of the blog post.
- tags (string): Tags associated with the blog post.
- userId (number): The user ID associated with the blog post.
- rating (number): The rating of the blog post.
- report (null): A report associated with the blog post, if any.
- numReports (number): The number of reports for the blog post.
- hidden (boolean): Indicates if the blog post is hidden.
- editability (boolean): Indicates if the blog post is editable.
- linkToTemplates (array): An array of objects containing blogPostId and templateId linked to the blog post.

Example:

```
{
    "id": 1,
    "title": "Blog Post 1",
    "description": "This is my first blog post.",
    "tags": "blog, intro",
    "userId": 1,
    "rating": 0,
```

```
    "report": null,
    "numReports": 0,
    "hidden": false,
    "editability": true,
    "linkToTemplates": [
        {
            "blogPostId": 1,
            "templateId": 1
        },
        {
            "blogPostId": 1,
            "templateId": 2
        }
    ]
}
```


Request (for GET):

Query Parameters: title, description, tags, linkToTemplates. All are optional, can include more than 1 or none at all. It also supports pagination by allowing for “page” and “limit” query parameters to be passed into the Request URL.

No authorization required.

Example:

http://localhost:3000/api/blogPost?tags=blog&title=Post&linkToTemplates=1

Response (for GET):

Page 1 with limit 10 is returned by default.

Example:


```
{
    "page": 1,
    "limit": 10,
    "data": [
        {
            "id": 1,
            "title": "Blog Post 1",
            "description": "This is my first blog post.",
            "tags": "blog, intro",
            "userId": 1,
            "rating": 0,
            "report": null,
            "numReports": 0,
            "hidden": false,
            "editability": true,
            "linkToTemplates": [
                {
                    "blogPostId": 1,
                    "templateId": 1
                },
                {
                    "blogPostId": 1,
```

```
                "templateId": 2
            }
        ]
    },
    {
        "id": 2,
        "title": "Blog Post 2",
        "description": "This is my second blog post.",
        "tags": "blog, cool",
        "userId": 1,
        "rating": 0,
        "report": null,
        "numReports": 0,
        "hidden": false,
        "editability": true,
        "linkToTemplates": [
            {
                "blogPostId": 2,
                "templateId": 1
            }
        ]
    }
]
}
```

/api/blogPost/[id]

Description: This endpoint allows the logged-in user to update, vote on, or delete an existing blog post belonging to them.

Allowed Methods: PUT, PATCH, DELETE

Request (for PUT):

Allows user to update title, description, tags, and/or ‘links’ to templates of a particular blog post

Payload Structure:

title (string, optional): The updated title of the blog post.

Description (string, optional): The updated description of the blog post.

tags (string, optional): The updated tags for the blog post.

linkToTemplates (array of numbers, optional): The updated list of template IDs linked to the blog post.

Must also provide Authorization (as seen below).

Example:

Auth Type: Bearer Token, Token: {{accessToken}}

```
{
    "title": "Updated New Blog",
    "tags": "blog, template, update",
```

```
    "linkToTemplates": [2]
}
```

Response Example:

```
{
    "id": 3,
    "title": "Updated New Blog",
    "description": "I like Template 1 very much!",
    "tags": "blog, template, update",
    "userId": 2,
    "rating": 0,
    "report": null,
    "numReports": 0,
    "hidden": false,
    "editability": true,
    "linkToTemplates": [
        {
            "blogPostId": 3,
            "templateId": 2
        }
    ]
}
```

Request (for PATCH):

This endpoint is used to update the user vote for a specific blog post. The “vote” attribute can be set to either “upvote” or “downvote”.

Must also provide Authorization (as seen below).

Example:

Auth Type: Bearer Token, Token: {{accessToken}}

```
{
    "vote": "upvote"
}
```

Response Example (for PATCH):


```
{
    "id": 3,
    "title": "Updated New Blog",
    "description": "I like Template 1 very much!",
    "tags": "blog, template, update",
    "userId": 2,
    "rating": 1,
    "report": null,
    "numReports": 0,
    "hidden": false,
    "editability": true,
    "upvotedByUsers": [
        {
            "id": 2,
            "username": "john",
```

```
            "firstName": "John",
            "lastName": "Johnson",
            "email": "john@gmail.com",
            "password":
"$2b$10$N/.CCKP.ah1faRyLHl7pguD0HcuW3ZMxwatTOKM1ZVWzyElC3l7XS",
            "avatar": "http://localhost:3000/avatars/avatar3.png",
            "phoneNum": "+1 (647) 234-567",
            "role": "USER"
        }
    ],
    "downvotedByUsers": []
}
```


Request (for DELETE):

The API endpoint sends an HTTP DELETE request to http://localhost:3000/api/blogPost/[id] to delete the blog post with [id] . Upon successful deletion, the response will be in the form of a JSON object with a "message" key.

Must also provide Authorization (as seen below).

Example:

Auth Type: Bearer Token, Token: {{accessToken}}

http://localhost:3000/api/blogPost/3

Response Example (for DELETE):


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

/api/blogPost/sorted

Description: This endpoint retrieves a sorted list of blog posts based on a specified ‘sort’ parameter.

Allowed Methods: GET

Request:

No request body parameters are required for this request.

- `sort` (query parameter): Specifies the value based on which the blog posts are sorted. Must be set to either “valued” or “controversial”.

It also supports pagination by allowing for “page” and “limit” query parameters to be passed into the Request URL.

No authorization required.

Example:

http://localhost:3000/api/blogPost/sorted?sort=valued

Response:

If “sort” is set to “valued”, the blog posts with the highest rating are returned first. If “sort” is set to “controversial”, the blog posts with the lowest rating are returned first. Page 1 with a limit of 10 is returned by default.

Example:

{

```
"page": 1,
"limit": 10,
"data": [
    {
        "id": 1,
        "title": "Blog Post 1",
        "description": "This is my first blog post.",
        "tags": "blog, intro",
        "userId": 1,
        "rating": 0,
        "report": null,
        "numReports": 0,
        "hidden": false,
        "editability": true
    },
    {
        "id": 2,
        "title": "Blog Post 2",
        "description": "This is my second blog post.",
        "tags": "blog, cool",
        "userId": 1,
        "rating": 0,
        "report": null,
        "numReports": 0,
```

```
            "hidden": false,
            "editability": true
        },
        {
            "id": 3,
            "title": "Updated New Blog",
            "description": "I like Template 1 very much!",
            "tags": "blog, template, update",
            "userId": 2,
            "rating": -1,
            "report": null,
            "numReports": 0,
            "hidden": false,
            "editability": true
        }
    ]
}
```

/api/blogPost/template-mention

Description: This endpoint retrieves a list of blog posts that mention a specific template in either their title or their description.

Allowed Methods: GET

Request:

No request body parameters are required for this request.

Query parameter:

- `name` (string): The name of the template to be retrieved.

It also supports pagination by allowing for “page” and “limit” query parameters to be passed into the Request URL.

No authorization required.

Example:
http://localhost:3000/api/blogPost/template-mention?name=Template 1

Response:

Page 1 with limit 10 is returned by default. Returns all blog posts that mention the provided template.

Example:

```
{
```

```
    "page": 1,
    "limit": 10,
    "data": [
        {
            "id": 3,
            "title": "Updated New Blog",
            "description": "I like Template 1 very much!",
            "tags": "blog, template, update",
            "userId": 2,
            "rating": -1,
            "report": null,
            "numReports": 0,
            "hidden": false,
            "editability": true,
            "linkToTemplates": [
                {
                    "blogPostId": 3,
                    "templateId": 2
                }
            ]
        }
    ]
}
```

Comments

/api/comments

Allowed Methods: GET, POST

1. GET

Retrieve comments for a specific post, using postId.

Request Parameters

- `postId` (number): The ID of the post for which comments are being retrieved.

Response

If successful: The response will be a JSON object with the following schema:

```
{
  "comments": [...],
  "page": 0,
  "limit": 0
}
```

The "comments" key will contain an array of comments, while "page" and "limit" will provide pagination information.

Else: 404 Not Found for post that does not exist

2. POST

This endpoint allows the user to submit a comment for a specific post.

Request Body

- `postId` (Number): The ID of the post for which the comment is being submitted.
- `body` (String): The content of the comment.
- `parentCommentId` (Number): Optional. To link it to another comment

Response

The response will be in the form of a JSON object with the following properties:

- `id` (Number): The unique identifier for the comment.
- `postId` (Number): The ID of the post for which the comment was submitted.
- `userId` (Number): The ID of the user who submitted the comment.
- `body` (String): The content of the comment.
- `numReports` (Number): The number of reports received for the comment.
- `rating` (Number): The rating of the comment.
- `hidden` (Boolean): Indicates whether the comment is hidden or not.
- `parentCommentId` (Number or null): The ID of the parent comment, if any.
- `createdAt` (String): The timestamp when the comment was created.

JSON Schema for Response

If successful:

```
{  "id": { "type": "number" },
   "postId": { "type": "number" },
   "userId": { "type": "number" },
   "body": { "type": "string" },
   "numReports": { "type": "number" },
   "rating": { "type": "number" },
   "hidden": { "type": "boolean" },
   "parentCommentId": { "type": ["number", "null"] },
   "createdAt": { "type": "string" }
}
```

Else: 404 Not Found for post that does not exist; 401 Unauthorized for non-user usage

/api/comments/[id]

Allowed Methods: PUT, DELETE

1. PUT

Update a specific comment with the provided ID. The request body should contain the updated "body" of the comment.

Response

If successful: The response for this request is a JSON object with the following schema:

- `id` (number): The unique identifier of the comment.
- `postId` (number): The ID of the post to which the comment belongs.
- `userId` (number): The ID of the user who made the comment.
- `body` (string): The updated body of the comment.
- `numReports` (number): The number of reports received for the comment.
- `rating` (number): The rating of the comment.
- `hidden` (boolean): Indicates whether the comment is hidden or not.
- `parentCommentId` (number): The ID of the parent comment, if any.
- `createdAt` (string): The timestamp when the comment was created.

This schema can be used to validate the structure of the response object received after making the PUT request.

Else: 404 Not Found for comment that does not exist; 401 Unauthorized for non-comment-owner usage

2. DELETE

Delete a specific comment with the provided ID. The comment can only be deleted by the owner of the comment.

Response

The request returns a 204 No Content response upon successful deletion. If the user is unauthorized to delete the comment, the request returns a 401 Unauthorized response.

/api/comments/[id]/vote

Allowed Methods: POST

Vote (upvote/downvote) on a specific comment.

Request Body

- `vote` (string, required): The type of vote. Can be either "upvote" or "downvote".

Response

If successful: The response will be a JSON object with the following properties:

- `id` (number): The unique identifier of the comment.
- `postId` (number): The identifier of the post the comment belongs to.
- `userId` (number): The identifier of the user who posted the comment.
- `body` (string): The content of the comment.
- `numReports` (number): The number of reports the comment has received.
- `rating` (number): The rating score of the comment.
- `hidden` (boolean): Indicates if the comment is hidden.
- `parentCommentId` (number): The identifier of the parent comment, if it's a reply.
- `createdAt` (string): The timestamp of when the comment was created.

Else: 404 Not Found for comment that does not exist; 401 Unauthorized for non-user usage; 400 for trying to upvote/downvote more than once

JSON Schema

```
{
    "id": { "type": "number" },
    "postId": { "type": "number" },
    "userId": { "type": "number" },
    "body": { "type": "string" },
    "numReports": { "type": "number" },
    "rating": { "type": "number" },
    "hidden": { "type": "boolean" },
    "parentCommentId": { "type": ["number", "null"] },
    "createdAt": { "type": "string" }
}
```

Reports

api/reports

Allowed Methods: POST

Submit a report for a comment/post.

The request body should be in raw format and include the following parameters:

- `contentId` (Number): The ID of the content being reported, the corresponding comment or post ID.
- `contentType` (String): The type of content being reported, either “comment” or “post”.

An example of the response for this request is as follows:

```
{
 "id": 0,
 "blogPostId": null,
 "commentId": 0,
 "contentType": "",
 "explanation": null,
 "createdAt": "",
 "userId": 0
}
```

Here is the JSON schema for the response:

If successful:

```
{
  "id": {"type": "number"},
  "blogPostId": { "type": ["number", "null"] },
  "commentId": { "type": ["number", "null"] },
```

```
  "contentType": { "type": "string" },
  "explanation": { "type": ["string", "null"] },
  "createdAt": { "type": "string" },
  "userId": { "type": "number" }
}
```

Else: 404 Not Found for content that does not exist; 401 Unauthorized for non-admin usage

api/admin/reports

Allowed Methods: GET

For admins to retrieve reports based on the specified content type and sorting criteria.

Request Parameters

- `contentType` (query parameter) - Specifies the type of content for which reports are requested.
- `sortByReports` (query parameter) - Indicates whether the reports should be sorted based on the number of reports.

Response
If successful:

```
{
  "content": [...],
  "page": 1,
  "totalPages": 0
}
```

Else: 401 Unauthorized for non-admin usage

api/admin/reports/hide

Allowed Methods: POST

For admins to hide/unhide specific content (comment or post) from the visitors/users of the app.

Request Body

- `contentId` (number) - The unique identifier of the content to be hidden.
- `contentType` (string) - The type of content to be hidden.

Response
If successful:

```
{
  "id": { "type": "number" },
  "postId": { "type": "number" },
  "userId": { "type": "number" },
  "body": { "type": "string" },
  "numReports": { "type": "number" },
  "rating": { "type": "number" },
  "hidden": { "type": "boolean" },
  "parentCommentId": { "type": ["number", "null"] },   # If comment
  "createdAt": { "type": "string" }
}
```

Else: 404 Not Found for content that does not exist; 401 Unauthorized for non-admin usage

api/templates

Allowed Methods: POST

POST

This endpoint allows an authenticated user to create a new template in the system.

Request Headers

- `Authorization` (String): Bearer token for user authentication.

Request Body

- `title` (String, required): The title of the template.
- `explanation` (String, optional): Explanation or description of the template.
- `tags` (String or Array, optional): Tags associated with the template. If an array is provided, it will be converted to a comma-separated string.
- `code` (String, required): Code content of the template.

Response On success, returns a JSON object with the details of the newly created template:

- `id` (Number): The unique identifier for the template.
- `title` (String): The title of the template.
- `explanation` (String): Explanation or description of the template.
- `tags` (String): Tags associated with the template in a comma-separated format.
- `code` (String): Code content of the template.
- `userId` (Number): The ID of the user who created the template.
- `hidden` (Boolean): Defaults to `false`, indicating the template visibility.
- `forkedFrom` (Number or null): Indicates the template it was forked from, if any.

Error Responses

- **400 Bad Request:** Missing required fields (title or code) or a template with the same title already exists.
  - `{ "message": "Title and code are required." }`
  - `{ "message": "A template with this title already exists." }`
- **405 Method Not Allowed:** When a non-POST method is used.
  - `{ "message": "Method not allowed." }`
- **500 Internal Server Error:** Server error while creating the template.
  - `{ "message": "Error creating template.", "error": "<error details>" }`

api/templates/public

Allowed Methods: GET

GET

This endpoint allows users to retrieve a paginated list of public templates. Optional search functionality is provided to filter templates by title, explanation, tags, or code content.

Query Parameters

- `search` (String, optional): A search term to filter templates by title, explanation, tags, or code content. Case-insensitive.
- `page` (Number, optional, default = 1): The page number for pagination.
- `limit` (Number, optional, default = 10): The maximum number of templates per page.

Response On success, returns a JSON object containing the list of templates along with pagination details.

- `templates` (Array): List of public templates.
    - Each template includes:
        - `id` (Number): The unique identifier of the template.
        - `title` (String): The title of the template.
        - `explanation` (String): The explanation or description of the template.
        - `tags` (String): Tags associated with the template.
        - `code` (String): Code content of the template.
        - `user` (Object): Information about the user who created the template.
            - `firstName` (String): The first name of the user.
            - `lastName` (String): The last name of the user.
            - `avatar` (String or null): URL to the user's avatar image, if available.
- `pagination` (Object): Details about pagination.
    - `totalItems` (Number): Total number of templates that match the search criteria.
    - `totalPages` (Number): The total number of pages based on `limit`.
    - `currentPage` (Number): The current page number.
    - `perPage` (Number): The number of templates per page.

Error Responses

- **405 Method Not Allowed:** When a non-GET method is used.
  - `{ "message": "Method not allowed." }`
- **500 Internal Server Error:** Error occurred while fetching the templates.
  - `{ "message": "Error fetching templates.", "error": "<error details>" }`

api/templates/[id]

Allowed Methods: PUT, PATCH, DELETE

Authentication

This endpoint requires JWT authentication. The user must be the owner of the template to access or modify it.

PUT / PATCH

This endpoint allows an authenticated user to update an existing template they own.

Request Headers

- `Authorization` (String): Bearer token for user authentication.

Request Parameters

- `id` (Number, required): The ID of the template to be updated.

Request Body

- `title` (String, optional): Updated title of the template.
- `explanation` (String, optional): Updated explanation or description of the template.
- `tags` (String or Array, optional): Updated tags associated with the template. If an array is provided, it will be converted to a comma-separated string.
- `code` (String, optional): Updated code content of the template.

Response On success, returns a JSON object with the details of the updated template:

- `id` (Number): The unique identifier for the template.

- `title` (String): The updated title of the template.
- `explanation` (String): The updated explanation or description of the template.
- `tags` (String): Updated tags in a comma-separated format.
- `code` (String): The updated code content of the template.

Error Responses

- **400 Bad Request:** A template with the same title already exists.
  - `{ "message": "A template with this title already exists." }`
- **403 Forbidden:** The user does not own the template.
  - `{ "message": "Forbidden" }`
- **405 Method Not Allowed:** When a non-PUT, non-PATCH, or non-DELETE method is used.
  - `{ "message": "Method not allowed." }`
- **500 Internal Server Error:** Error occurred while updating the template.
  - `{ "message": "Error updating template.", "error": "<error details>" }`

DELETE

This endpoint allows an authenticated user to delete a template they own.

Request Headers

- `Authorization` (String): Bearer token for user authentication.

Request Parameters

- `id` (Number, required): The ID of the template to be deleted.

Response On success, returns a JSON object indicating successful deletion:

- `{ "message": "Template deleted successfully." }`

Error Responses

- **403 Forbidden:** The user does not own the template.
  - `{ "message": "Forbidden" }`
- **405 Method Not Allowed:** When a non-DELETE, non-PUT, or non-PATCH method is used.

    - `{ "message": "Method not allowed." }`
- **500 Internal Server Error:** Error occurred while deleting the template.
    - `{ "message": "Error deleting template.", "error": "<error details>" }`

api/templates/[id]/user

Allowed Methods: GET

Authentication

This endpoint requires JWT authentication. Only the authenticated user can access their templates through this endpoint.

GET

This endpoint allows an authenticated user to retrieve a paginated list of their own templates. An optional search functionality is provided to filter templates by title, explanation, or tags.

Request Headers

- `Authorization` (String): Bearer token for user authentication.

Query Parameters

- `search` (String, optional): A search term to filter templates by title, explanation, or tags. Case-insensitive.
- `page` (Number, optional, default = 1): The page number for pagination.
- `limit` (Number, optional, default = 10): The maximum number of templates per page.

Response On success, returns a JSON object containing the list of templates created by the authenticated user, along with pagination details.

- `templates` (Array): List of user-owned templates.
    - Each template includes:
        - `id` (Number): The unique identifier of the template.
        - `title` (String): The title of the template.

        - `explanation` (String): The explanation or description of the template.
        - `tags` (String): Tags associated with the template.
        - `code` (String): Code content of the template.
- `pagination` (Object): Details about pagination.
    - `totalItems` (Number): Total number of templates that match the search criteria.
    - `totalPages` (Number): The total number of pages based on `limit`.
    - `currentPage` (Number): The current page number.
    - `perPage` (Number): The number of templates per page.

Error Responses

- **405 Method Not Allowed:** When a non-GET method is used.
    - `{ "message": "Method not allowed." }`
- **500 Internal Server Error:** Error occurred while fetching the templates.
    - `{ "message": "Error fetching templates.", "error": "<error details>" }`

api/templates/fork/[id]

Allowed Methods: POST

Authentication

This endpoint requires JWT authentication. The user must be authenticated to fork a template.

POST

This endpoint allows an authenticated user to create a new template by forking an existing template. The new template will include all content from the original template, with modifications allowed for title, explanation, tags, and code.

Request Headers

- `Authorization` (String): Bearer token for user authentication.

Request Parameters

- `id` (Number, required): The ID of the template to be forked.

Request Body

- `title` (String, required): The title of the new forked template.
- `explanation` (String, optional): Explanation or description for the forked template.
- `tags` (String or Array, optional): Tags for the forked template. If an array is provided, it will be converted to a comma-separated string.
- `code` (String, required): Code content for the forked template.

Response On success, returns a JSON object containing the details of the newly created forked template:

- `id` (Number): The unique identifier for the forked template.
- `title` (String): The title of the forked template.
- `explanation` (String): The explanation or description of the forked template.
- `tags` (String): Tags associated with the forked template.
- `code` (String): Code content of the forked template.
- `userId` (Number): The ID of the user who created the forked template.
- `forkedFrom` (Number): The ID of the original template.

Error Responses

- **400 Bad Request:** Missing required fields (title or code) or a template with the same title already exists.
  - `{ "message": "Title and code are required." }`
  - `{ "message": "A template with this title already exists." }`
- **404 Not Found:** The original template does not exist.

    - `{ "message": "Template not found." }`
- **405 Method Not Allowed:** When a non-POST method is used.
    - `{ "message": "Method not allowed." }`
- **500 Internal Server Error:** Error occurred while forking the template.
    - `{ "message": "Error forking template.", "error": "<error details>" }`

api/execute

Allowed Methods: POST

Purpose

This endpoint allows users to execute code in various programming languages (JavaScript, Python, C, C++, and Java). The code can include optional input, and the response will return the code output or any error messages.

Request Headers

- `Content-Type` (String): `application/json`

Request Body

- `code` (String, required): The code to be executed.
- `input` (String, optional): Input provided for the code.
- `language` (String, required): The programming language of the code. Must be one of the following:
    - `javascript`
    - `python`
    - `c`
    - `cpp`
    - `java`

Error Responses

- **400 Bad Request:** Missing required fields (code or language) or invalid language specified.
  - `{ "error": "Valid code and language are required" }`
- **405 Method Not Allowed:** When a non-POST method is used.
  - `{ "message": "Method not allowed" }`
- **500 Internal Server Error:** An error occurred during code execution or compilation.
  - `{ "error": "Execution error: <error details>" }`