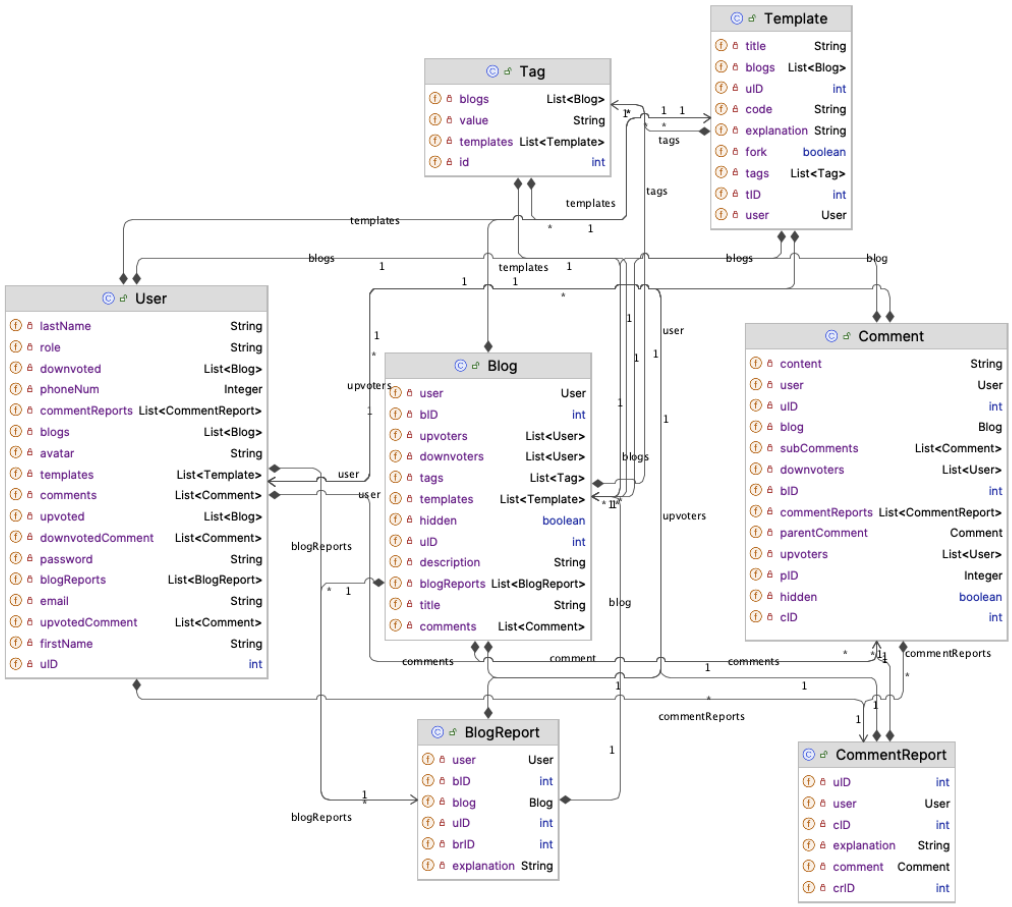
Documentation

Cynthia Zhou and William Lam

Model Design



User

The User model is central to the system, reflecting the core entity that interacts with various aspects of the platform.

- **User-Blog Connection (blogs)**: This one-to-many relationship allows a single user to author multiple blogs. Each blog is linked back to its creator, ensuring that content ownership is clear.
- **User-Template Connection (templates)**: Users can create reusable templates, fostering knowledge sharing and code reuse. The relationship is one-to-many since a user can create multiple templates.
- **User-Comment Connection (comments)**: Users can leave multiple comments on different blogs. This one-to-many relationship captures user engagement and participation in discussions.
- **User-Report Connections (commentReports, blogReports)**: Users can report content (blogs or comments). These one-to-many relations ensure accountability and track which user reported which content for moderation purposes.
- **Voting Connections (upvoted, downvoted, upvotedComment, downvotedComment)**: These many-to-many relationships capture the dynamics of content interaction, allowing users to express approval or disapproval of blogs and comments, which helps in content ranking and visibility.

Tag

Tags are used to categorize and enhance the discoverability of content.

- **Tag-Blog Connection (blogs)**: A many-to-many relationship, where a single blog can have multiple tags, and each tag can be associated with multiple blogs. This structure supports robust content categorization and filtering.
- **Tag-Template Connection (templates)**: Similar to blogs, templates can be categorized using tags, aiding in search and discovery. This encourages better organization and helps users quickly find templates relevant to their needs.

Template

Templates serve as reusable code or content snippets.

- **Template-User Connection (user)**: Each template is authored by a user, captured through a one-to-many relationship. This linkage promotes accountability and allows users to showcase their contributions.

- **Template-Blog Connection (blogs)**: Templates can be associated with blogs, fostering reuse of code or content structures. This many-to-many relationship supports collaborative content creation and consistency across blogs.

Blog

The Blog model represents the main content in the system.

- **Blog-User Connection (user)**: Each blog is linked to a user, establishing clear ownership. This one-to-many relationship enables tracking of all content authored by a user.
- **Blog-Tag Connection (tags)**: Tags enhance the discoverability of blogs by categorizing them. This many-to-many relationship allows blogs to be flexibly categorized under multiple relevant tags.
- **Blog-Template Connection (templates)**: Blogs can utilize templates, enabling content creators to streamline their writing process and maintain consistency. The many-to-many relationship reflects the potential for multiple templates to be used across different blogs and vice versa.
- **Blog-Comment Connection (comments)**: Each blog can have multiple comments, fostering user interaction and engagement. This one-to-many relationship allows the platform to support discussions around each blog post.
- **BlogReport Connection (blogReports)**: Enables users to report blogs, creating a channel for moderation. The one-to-many relationship tracks all reports against a specific blog.
- **Voting Connections (upvoters, downvoters)**: These many-to-many relationships capture user feedback, helping to surface popular or high-quality content while demoting less favorable content.

Comment

Comments facilitate discussions and user interaction.

- **Comment-User Connection (user)**: Each comment is authored by a user, establishing accountability and promoting user engagement.
- **Comment-Blog Connection (blog)**: Comments are tied to specific blogs, maintaining the context of discussions. This one-to-many relationship enables rich, focused conversations on blog posts.
- **Nested Comments (subComments, parentComment)**: The self-referential relationship allows for threaded conversations, where comments can have replies, promoting detailed discussions.

- **CommentReport Connection (commentReports)**: Users can report comments, supporting moderation efforts. This one-to-many relationship tracks reports associated with specific comments.
- **Voting Connections (upvoters, downvoters)**: Similar to blogs, comments also benefit from user feedback via upvotes and downvotes, which helps highlight valuable contributions and manage content quality.

CommentReport

Handles reports of inappropriate or problematic comments.

- **CommentReport-Comment Connection (comment)**: Each report is linked to a specific comment, ensuring that moderation actions can be accurately targeted.
- **CommentReport-User Connection (user)**: Tracks which user submitted the report, providing accountability and allowing for follow-up or further investigation.

BlogReport

Manages reports on blogs.

- **BlogReport-Blog Connection (blog)**: Each report directly associates with a specific blog, helping moderators pinpoint the content in question.
- **BlogReport-User Connection (user)**: Identifies the user who submitted the report, maintaining accountability and enabling better handling of the moderation process.

Summary

This schema design balances several key aspects:

- **Accountability and Attribution**: By linking all content (blogs, templates, comments) back to users, the system ensures clear ownership and accountability.
- **Discoverability and Organization**: Tags and templates enable robust categorization and reuse, making content easier to find and manage.
- **User Engagement**: Comments and voting systems foster interaction, allowing users to contribute to and shape the community.
- **Moderation**: Reporting mechanisms and hidden fields support content moderation, ensuring that the platform maintains quality and appropriateness.
- **Scalability**: The many-to-many and one-to-many relationships are designed to handle growing content and user interactions efficiently.

Scriptorium

Welcome to the Scriptorium Backend API documentation.

user

This folder contains API endpoints related to user management functions within the Scriptorium Backend. It includes operations such as user registration, login, profile editing, and the display of all blogs and templates created by the user.

POST sign up

http://localhost:3000/api/user/signup

It supports content type json for standard registration data and multipart/form-data for registrations including a profile picture upload using Multer.

Payload:

• firstName (string): First name of the user.

• lastName (string): Last name of the user.

• email (string): Email address of the user, must be unique.

• password (string): Password for the user account.

• phoneNum (integer, optional): Phone number of the user.

• role (string, default “user”): Role of the user in the system.

• avatar (file, optional): Profile picture to upload.

Responses:

• **201 Created**:

`{ "message": "User registered successfully", "user": { "uID": 5, "firstName": "test", "lastName": "admin", "email": " admin1234@gmail.com ", "password": "$2b$10$cmzZbv1xNWQQkz3X6vVe4.WySpGnvkasIqMSxsK/7jG7gX5eFybq6", "avatar": "public/avatar/default.jpg", "phoneNum": 1234, "role": "admin" } }`

• **400 Bad Request**: Missing required fields or incorrect data format.

• **409 Conflict**: Email already registered.

• **500 Internal Server Error**: Issues with file upload, database access, or password hashing.

Body raw (json)

```
json

{
    "firstName":"test",
    "lastName":"admin",
    "email":"admin1234@gmail.com",
    "password":"123",
    "phoneNum":1234,
    "role":"admin"
}
```

POST login

http://localhost:3000/api/user/login

This endpoint authenticates a user by their email and password. It validates the user credentials and returns access and refresh tokens upon successful authentication.

Payload :

• email (string): The user's email address.

• password (string): The user's password.

Responses:

• **200 OK**:

`{ "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzExNjMsImV4cCI6MTczMDY3Mjk2M30.XAWdaethqNIYf-4M-X2awM1JgQ5BDn46Bfhpe6AOboA", "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzExNjMsImV4cCI6MTczMTI3NTk2M30.STBTw4acY8eqXmR6IWrmbvF02PmYThbMMQreAc65MnI" }`

• **400 Bad Request**: Missing either email or password.

• **401 Unauthorized**: Invalid credentials.

• **405 Method Not Allowed**: If any method other than POST is used.

Body raw (json)

```json
{
    "email":"admin123@gmail.com",
    "password":"123"
}
```

POST refresh token

http://localhost:3000/api/user/refresh

This endpoint validates the user's access token provided in the authorization header and genreates new tokens.

Authorization:

• Authorization (string): Bearer access token obtained during user login.

Responses:

• **200 OK**: Successful validation.

`{ "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzE0ODksImV4cCI6MTczMDY3MzI4OX0.3dZBQpCs98bnpkuBNCJHO_ZElUCJHX3D2nO5BlnXalA", "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzE0ODksImV4cCI6MTczMTI3NjI4OX0.OnmX7ZcBtNOhrC-SX5TnN84tazXcN0P2ntJkr88HTWs" }`

• **401 Unauthorized**: Returned if the token is invalid or expired.

• **405 Method Not Allowed**: Returned if the request method is not POST.

GET view profile

http://localhost:3000/api/user/profile

Retrieves the authenticated user's profile information. It verifies the user's identity using an authorization token and returns the user's profile details.

Authorization:

Requires a valid authorization token provided in the request header.

Responses:

• **200 OK**: Successfully retrieves the user profile.

`{ "message": "User profile retrieved successfully", "user": { "uID": 4, "firstName": "teacct", "lastName": "user221", "email": " admin123@gmail.com ", "password": "$2b$10$SGAns/dN5Jq5LSq2dBGk3.8/TIxK1RjxDGvhQmQVJ/oWTYJOMIzWO", "avatar": "public/avatar/default.jpg", "phoneNum": 1234, "role": "admin" } }`

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

PATCH update profile

http://localhost:3000/api/user/profile

Allows the authenticated user to update their profile information. It handles data in both son and multipart/form-data formats, allowing updates to fields like firstName, lastName, phoneNum, and the profile avatar.

Authorization:

Requires a valid authorization token provided in the request header.

Payload:

• firstName: Optional string, must be non-empty if provided.

• lastName: Optional string, must be non-empty if provided.

• phoneNum: Optional number.

• avatar: Optional file, handled via multipart/form-data.

Responses:

• **200 OK**: Profile updated successfully.

`{ "message": "Profile updated successfully", "user": { "uID": 4, "firstName": "updated", "lastName": "name", "email": " admin123@gmail.com ", "password": "$2b$10$SGAns/dN5Jq5LSq2dBGk3.8/TIxK1RjxDGvhQmQVJ/oWTYJOMIzWO", "avatar": "public/avatar/default.jpg", "phoneNum": 123424, "role": "admin" } }`

• **400 Bad Request**: Validation errors for input data.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

Body raw (json)

json

```
{
    "firstName":"updated",
    "lastName":"name",
    "phoneNum":123424
}
```

GET view blogs by user

http://localhost:3000/api/user/blog?page=1&pageSize=5

Retrieves a paginated list of blogs created by the authenticated user. This endpoint verifies the user's identity using an authorization token and returns blogs associated with the user, including related tags and templates.

Authorization:

Requires a valid authorization token provided in the request header.

Query Parameters:

• page: Specifies the page number in the pagination sequence.

• pageSize: Specifies the number of blog entries per page.

Responses:

• **200 OK**: Successfully retrieves the list of blogs along with pagination details.

`{ "blogs": [{ "bID": 12, "title": "testing1", "description": "description 8", "hidden": false, "uID": 4, "tags": [{ "id": 1, "value": "Python" }, { "id": 2, "value": "Java" }, { "id": 4, "value": "python" }, { "id": 6, "value": "easy" }], "templates": [] }], "totalCount": 1, "page": 1, "pageSize": 5 }`

• **400 Bad Request**: Missing or invalid pagination parameters.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

PARAMS

page	1
pageSize	5

GET view templates by user

http://localhost:3000/api/user/template?page=1&pageSize=2

Retrieves a paginated list of templates created by the authenticated user. This endpoint verifies the user's identity using an authorization token and returns templates associated with the user, including related tags.

Authorization:

Requires a valid authorization token provided in the request header.

Query Parameters:

• page: Specifies the page number in the pagination sequence.

• pageSize: Specifies the number of template entries per page.

Responses:

• **200 OK**: Successfully retrieves the list of templates along with pagination details.

`{ "templates": [ { "tID": 2, "title": "", "explanation": null, "code": null, "fork": false, "uID": 4, "tags": [] }, { "tID": 3, "title": "cads", "explanation": "fat", "code": "dfsdf", "fork": false, "uID": 4, "tags": [] } ], "totalCount": 6, "page": 1, "pageSize": 2 }`

• **400 Bad Request**: Missing or invalid pagination parameters.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

PARAMS

page	1
pageSize	2

code execution

Executes code submitted by the user for 10 seconds, supporting various programming languages.

Request Body:

• code: The source code to execute.

• input: Input parameters for the code, if required.

• language: Programming language of the code (e.g., "javascript", "python").

Responses:

• **200 OK**: Returns the output of the executed code.

```
{ "output": "Hello from Python\n" }
```

• **400 Bad Request**: If the specified programming language is unsupported or got error when executing the code.

```
{ "error": "Failed to execute code: Execution timed out" }
```

• **405 Method Not Allowed**: If the HTTP method is not POST.

POST Python No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from Python\n"}
```

Body raw (json)

```
json
{
  "code": "print('Hello from Python')",
  "input": "",
  "language": "python"
}
```

POST Python w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello, PythonUser1 and PythonUser2\n" }
```

Body raw (json)

json

```
{
  "code": "input_values = [input() for _ in range(2)]; print(f'Hello, {input_values[0]} and {in
  "input": ["PythonUser1", "PythonUser2"],
  "language": "python"
}
```

POST JavaScript No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from JavaScript\n" }
```

Body raw (json)

json

```
{
  "code": "console.log('Hello from JavaScript')",
  "input": "",
  "language": "javascript"
}
```

POST JavaScript w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "\n" }
```

Body raw (json)

json

```
{
  "code": "const readline = require('readline'); const rl = readline.createInterface({ input: r
```

```
  "input": ["JavaScriptUser1", "JavaScriptUser2"],


  "language": "javascript"
}
```

POST Java No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from Java\n" }
```

Body raw (json)

```
json

{
  "code": "public class Main { public static void main(String[] args) { System.out.println(\"He
  "input": "",
  "language": "java"
}
```

POST Java w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{
"output": "Hello, JavaUser1 and JavaUser2\n"
}
```

Body raw (json)

```
json

{
  "code": "import java.util.Scanner; public class Main { public static void main(String[] args)
```

```
  "input": ["JavaUser1", "JavaUser2"],


  "language": "java"
}
```

POST C No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from C\n" }
```

Body raw (json)

json

```
{
  "code": "#include <stdio.h> \n int main() { printf(\"Hello from C\\n\"); return 0; }",
  "input": "",
  "language": "c"
}
```

POST C w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello, CUser1 and CUser2\n" }
```

Body raw (json)

json

```
{
  "code": "#include <stdio.h>\n int main() { char input1[50], input2[50]; scanf(\"%s\", input1)
  "input": ["CUser1", "CUser2"],
  "language": "c"
}
```

POST C++ No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from C++\n" }
```

Body raw (json)

json

```
{
  "code": "#include <iostream> \n int main() { std::cout << \"Hello from C++\" << std::endl; re
  "input": "",
  "language": "cpp"
}
```

POST C++ w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello, CppUser1 and CppUser2\n" }
```

Body raw (json)

json

```
{
  "code": "#include <iostream>\n int main() { std::string input1, input2; std::cin >> input1 >>
  "input": ["CppUser1", "CppUser2"],
  "language": "cpp"
}
```

POST Timeout Error w/ Interpreted Language

http://localhost:3000/api/code/execute

Expected output: 400 Bad Request

```
{ "error": "Failed to execute code: Execution timed out" }
```

Body raw (json)

```json
{
  "code": "while True: pass",
  "input": "",
  "language": "python"
}
```

POST Timeout Error w/ Compiled Language

http://localhost:3000/api/code/execute

Expected output: 400 Bad Request

```
{ "error": "Failed to execute code: Execution timed out" }
```

Body raw (json)

```json
{
  "code": "#include <stdio.h>\nint main() { while (1) {} return 0; }",
  "input": "",
  "language": "c"
}
```

POST Division By Zero error - Python

http://localhost:3000/api/code/execute

Expected output: 400 Bad Request

{ "error": "Failed to execute code: Runtime error: Division by zero detected in the code." }

Body raw (json)

```
json
{
  "code": "print(0/0)",
  "input": "",
  "language": "python"
}
```

code templates

This endpoint in the API manages template-related operations including creation, deletion, search, and updating of templates. It supports various methods and ensures that the user is authenticated before performing some actions.

POST create template

http://localhost:3000/api/templates

Allow logged in user to post code template.

Authorization:

Requires a valid authorization token provided in the request header.

Request Parameters:

• title: Title of the template (required).

• explanation: A brief description or explanation of the template (optional).

• tags: An array of tag values related to the template (optional).

• code: The actual code snippet for the template (optional).

• fork: Boolean indicating if the template is a fork of another template (defaults to false).

Responses:

• **201 Created**: Returns a success message and the newly created template.

{ "message": "Template created", "template": { "tID": 14, "title": "New Title !!", "explanation": "Updated explanation", "code": "Updated code", "fork": true, "uID": 4 } }

{ "explanation": "Updated explanation", "code": "Updated code", "fork": true, "uID": 4 }

• **400 Bad Request**: Missing required fields or invalid input format.

• **401 Unauthorized**: No authorization token provided or token validation failed.

• **409 Conflict**: A template with the given title already exists.

Body raw (json)

```json
{
  "title": "New Title !!",
  "explanation": "Updated explanation",
  "tags": ["updatedTag"],
  "code": "Updated code",
  "fork": true
}
```

GET search template

http://localhost:3000/api/templates?title =title&tags=python&tags=java&explanation=what&page=&limit=3

Search template based on given input. All given condition (case insensitive) must be matched to be shown in the search result.

Query Parameters:

• page: Specifies the page number in the pagination sequence (default: 1).

• pageSize: Specifies the number of template entries per page (default: 10).

• title: Filters templates based on title inclusion (optional).

• tags: Filters templates based on associated tags (optional, can be multiple strings).

• explanation: Filters templates based on text included in the explanation (optional).

Responses:

• **200 OK**: Successfully retrieves the list of templates along with pagination details. (not the output for the Query Params show below)

{ "templates": [{ "tID": 1, "title": "Updated Title 1", "explanation": "Updated explanation", "code": "Updated code", "fork": false, "uID": 1 }, { "tID": 2, "title": "", "explanation": null, "code": null, "fork": false, "uID": 4 }, { "tID": 3, "title": "cads", "explanation": "fat", "code": "dfsdf", "fork": false, "uID": 4 }], "totalPages": 5, "currentPage": 1 }

• **400 Bad Request**: Invalid query parameters.

• **401 Unauthorized**: No authorization token provided or token is invalid.

• **404 Not Found**: No templates found or user not found.

PARAMS

title	title
tags	python
tags	java
explanation	what
page	
limit	3

DELETE delete template create by user

http://localhost:3000/api/templates?tID=4

Delete a template created by the user.

Authorization

Bearer token required for verifying user authentication.

Parameters

• tID: The ID of the template to be deleted.

Responses

• **200 OK**: Returns a success message indicating that the template was successfully deleted.

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

• **400 Bad Request**:

• Returns an error if the tID is not provided or cannot be parsed as an integer.

• Returns an error if the user does not exist in the database.

• Returns an error if the tags is more than 10.

• **401 Unauthorized**: Returns an error if no authorization token is provided.

• **403 Forbidden**: Returns an error if the user does not have permission to delete the specified template.

• **404 Not Found**: Returns an error if the specified template is not found in the database.

PARAMS

tID	4

PATCH edit template created by user

http://localhost:3000/api/templates

Update a template created by the user. If a field is not provided, make it maintain as the orginal.

Authorization

Bearer token required for verifying user authentication.

Request Body

• tID (Integer): The unique identifier for the template to be updated. Required.

• title (String): The new title of the template.

• explanation (String): Detailed explanation or description of the template.

• tags (Array of Strings): Tags associated with the template for categorization or search.

• code (String): The code snippet or content of the template.

• fork (Boolean): Indicates whether the template is a fork of another template.

Responses

• **200 OK**: Returns a success message indicating that the template was successfully updated.

```
{ "message": "Template updated successfully" }
```

• **400 Bad Request**: Invalid query parameters.

• **401 Unauthorized**: No authorization token provided or token is invalid.

• **404 Not Found**: No templates found or user not found.

Body raw (json)

json

```
{
  "title": "Update",
  "explanation": "exp",
  "tags": ["JS","1","2","3","4", "5", "6", "342"],
  "code": "console.log('Hello World')",
  "fork": true,
  "tID": 7
}
```


blog

This endpoint in the API manageblog-related operations including creation, deletion, search, vote, and edit. It supports various methods and ensures that the user is authenticated before performing some actions.

GET search blogs

http://localhost:3000/api/blog/search?method=controversial&title=bloa

Search for Blogs based on input. All search criteria must be satisfied for a blog to be shown in the output.

Query Parameters:

- **title** (optional): Filters posts containing the specified title substring.
- **content** (optional): Filters posts containing the specified content substring.
- **tags** (optional): Filters posts associated with the specified tags (array of strings or a single string).
- **templates** (optional): Filters posts associated with specified templates (array of strings or a single string).
- **method** (optional): Determines the sorting method of the posts (controversial, popular, or default by newest).
- **page** (default: 1): Specifies the page number in the result set.
- **pageSize** (default: 5): Specifies the number of posts per page.

Responses:

- **200 OK**: Returns a list of blog posts that match the criteria.

`[{ "bID": 9, "title": "bloafawww", "description": "description 8", "hidden": false, "uID": 1, "tags": [{ "id": 1, "value": "Python" }, { "id": 2, "value": "Java" }, { "id": 4, "value": "python" }], "templates": [], "user": { "uID": 1, "firstName": "updated", "lastName": "name", "email": " user1@gmail.com ", "password": "$2b$10$dp731kUjvbVmzAyprEM2p.fI5F.NRDtBDQyiEPYOtjWoejw/IrH22", "avatar": "public/avatar/default.jpg", "phoneNum": 123424, "role": "user" }, "_count": { "upvoters": 0, "downvoters": 0, "comments": 0 } }]`

• **400 Bad Request**:

• Returns an error if page or pageSize are not positive integers.

• Returns an error if there is an invalid JSON format for tags or templates.

• Returns an error if tags or templates are not arrays or a single string.

• Returns an error for incorrect data types or invalid values for title and content.

• **401 Unauthorized**:

• Returns an error if no authorization token is provided or the token validation fails.

• **405 Method Not Allowed**:

• Returns an error if the request method is not GET.

Sorting:

• **controversial**: Sorts posts by the sum of upvoters, downvoters, and comments.

• **popular**: Sorts posts by the number of upvoters.

• **default**: Sorts posts by the newest based on their ID.

PARAMS

tags	python
tags	Java
method	controversial
title	bloa

POST create blog

http://localhost:3000/api/blog/create

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **title** (String): The title of the blog post. This is a required field and must be a non-empty string.

• **description** (String): A detailed description or body of the blog post. This is a required field and must be a non-empty string.

• **tags** (Array of Strings): A list of tags associated with the blog post. Optional field, defaults to an empty array, cannot contain more than 10 tags.

• **templates** (Array of Strings): A list of template titles associated with the blog post. Optional field, defaults to an empty array, cannot contain more than 10 templates.

Responses:

• **201 Created**: Returns a success message and details of the newly created blog post.

`{ "message": "Blog created successfully", "blog": { "bID": 13, "title": "testingsdvcsd", "description": "description 8", "hidden": false, "uID": 4, "tags": [{ "id": 1, "value": "Python" }, { "id": 2, "value": "Java" }, { "id": 4, "value": "python" }, { "id": 6, "value": "easy" }], "templates": [] } }`

• **400 Bad Request**:

• Title and description are required and must be strings.

• Tags and templates must be valid arrays of strings and cannot exceed 10 items each.

• One or more specified templates do not exist.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User associated with the token does not exist.

• **405 Method Not Allowed**: Method used is not POST.

• **409 Conflict**: A blog with the provided title already exists.

• **500 Internal Server Error**: Unable to create blog post due to a database error.

Body raw (json)

```json
{
    "title":"testingsdvcsd",
    "description":"description 8",
    "tags":["Python", "Java", "python","easy"],
    "templates":[]
}
```

POST vote blog

http://localhost:3000/api/blog/vote

Logged in user can vote. If user already voded on the blog, overwrote it with the new request.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to vote on. Required.

• **voteType** (String): The type of vote to cast. Accepts either “upvote” or “downvote”. Required.

Responses:

• **200 OK**:Returns a success message and the updated vote count of the blog post.

`{ "message": "Successfully updated downvote", "blog": { "bID": 3, "title": "fdfvs", "description": "asfcedfvVARVRG", "hidden": false, "uID": 1, "_count": { "upvoters": 0, "downvoters": 2 } } }`

• **400 Bad Request**: Returns an error if the vote type is invalid or missing.

• **401 Unauthorized**: Returns an error if the authorization token is invalid or missing.

• **404 Not Found**: Returns an error if the user or blog post is not found.

• **405 Method Not Allowed**: Returns an error if the request method is not POST.

Body raw (json)

```
json

{"bID": 3,"voteType":"downvote"}
```

PATCH edit blog

http://localhost:3000/api/blog/edit

Allow user to edit blog they created.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to be updated. Required.

• **title** (String): The new title for the blog post. Optional.

• **description** (String): The new description or content for the blog post. Optional.

• **tags** (Array of Strings): A new list of tags to replace any existing tags. Optional.

• **templates** (Array of Integer IDs): A new list of template IDs to associate with the blog post. Optional.

• **hidden** (Boolean): The visibility status of the blog post. Optional.

Responses:

• **200 OK**:

• Returns a success message and the updated details of the blog post.

• Example Message: "Blog updated successfully"

• **400 Bad Request**:

• Returns an error if required fields are missing or malformed.

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have permission to edit the specified blog post.

• Example Message: "You do not have permission to edit this blog post"

• Returns an error if the blog post is marked as hidden and cannot be edited.

• Example Message: "You cannot edit this blog post"

• **404 Not Found**:

• Returns an error if the user or the specified blog post cannot be found.

• Example Message: "User not found"

• Example Message: "Blog post not found"

• **500 Internal Server Error**:

• Returns an error if there is an unexpected issue during the update process.

• Example Message: "Unable to update blog post, database error."

Body raw (json)

json

```
{"bID": 4, "title":"new title 1", "tags" : ["c++"]}
```

DELETE delete blog

http://localhost:3000/api/blog/delete?bID=5

Allow user to delete blog they created.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Query Parameters:

• **bID** (Integer): The unique identifier of the blog post to be deleted. Required.

Responses:

• **200 OK**:

• Returns a success message indicating the blog post was successfully deleted.

• Example Message: "Delete successfully"

• **400 Bad Request**:

• Returns an error if the blog ID (bID) is not provided or is in an invalid format.

• Example Message: "Invalid or missing blog ID"

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have permission to delete the specified blog post.

• Example Message: "You do not have permission to delete this blog post"

• **404 Not Found**:

• Returns an error if the user or the specified blog post cannot be found.

• Example Message: "User not found"

• Example Message: "Blog post not found"

• **405 Method Not Allowed**:

• Returns an error if the method used is not DELETE.

• Example Message: "Method DELETE Not Allowed"

• **500 Internal Server Error**:

• Returns an error if there is an unexpected issue during the deletion process.

• Example Message: "Unable to delete blog post, database error."

PARAMS

bID	5

GET get blog

http://localhost:3000/api/blog?bID=3

Allows fetching of a blog post along with its comments based on provided sorting and pagination parameters.

Query Parameters:

• **bID** (Integer): The unique identifier of the blog post to retrieve. Required.

• **method** (String, optional): Sorting method for the comments (controversial, popular). Default is sorting by comment ID in ascending order.

• **page** (Integer, optional): The page number of comments to fetch. Default is 1.

• **pageSize** (Integer, optional): The number of comments per page. Default is 5.

Responses:

• **200 OK**:

• Returns the blog post and a paginated list of its comments, including the user details and vote counts for each comment.

• **400 Bad Request**:

• Returns an error if the bID is not provided or is not a valid integer.

• Returns an error if page or pageSize are not positive integers.

• Example Messages:

• "Blog ID is required."

• "Blog ID must be a valid integer."

• "Page and pageSize must be positive integers."

• **404 Not Found**:

• Returns an error if no blog post or no comments are found for the given blog ID.

• Example Messages:

• "Blog post not found."

• "Comment not found."

• **405 Method Not Allowed**:

• Returns an error if the request method is not GET.

• Example Message: "Method Not Allowed"

PARAMS

bID	3

comment

This endpoint in the API manage comment-related operations including creation, search, and vote. It supports various methods and ensures that the user is authenticated before performing some actions.

POST create comment

http://localhost:3000/api/comment/create

Allows a logged-in user to post comments on a blog post or reply to an existing comment.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to comment on. Required.

• **content** (String): The content of the comment. Must be a non-empty string. Required.

• **pID** (Integer, optional): The unique identifier of the parent comment, if replying to an existing comment.

Responses:

• **201 Created**:

• Returns a success message and the details of the newly created comment.

• Example Message: "Comment created successfully"

• **400 Bad Request**:

• Returns an error if the blog ID or content is missing or invalid, or if the content is empty.

• Example Message: "Blog ID and content are required and content must be a non-empty string."

• Returns an error if the parent comment does not belong to the provided blog ID when replying to a comment.

• Example Message: "Parent comment does not belong to the provided blog."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the user or the specified blog post/comment cannot be found.

• User not found: "User not found"

• Blog post not found or parent comment not found: "Parent comment not found"

• **405 Method Not Allowed**:

• Returns an error if the method used is not POST.

• Example Message: "Method Not Allowed"

Body raw (json)

```
json

{"bID":3, "content":"7", "pID":14}
```

POST vote comment

http://localhost:3000/api/comment/vote

Allows a logged-in user to cast a vote (upvote or downvote) on a comment.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **cID** (Integer): The unique identifier of the comment to vote on. Required.

• **voteType** (String): The type of vote to cast. Accepts “upvote” or “downvote”. Required.

Responses:

• **200 OK**:

• Returns a success message and the updated details of the comment including the current counts of upvotes and downvotes.

• Example Message: "Successfully updated upvote"

• **400 Bad Request**:

• Returns an error if the vote type is invalid or missing.

• Example Message: "Invalid vote type specified. Must be either 'upvote' or 'downvote'."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the specified comment cannot be found.

• Example Message: "Comment not found"

• **403 Forbidden**:

• Returns an error if the user does not have permission to vote, such as attempting to change a vote improperly or when the comment is locked from voting.

• Example Message: "You do not have permission to delete this blog post"

Body raw (json)

```json
{"cID":15, "voteType":"upvote"}
```

GET get comment

http://localhost:3000/api/comment?cID=14&page=1&method=controversial

Allows fetching of sub-comments related to a parent comment, with options for sorting and pagination.

Method:

• **GET**

URL:

• /api/comments/subcomments

Query Parameters:

• **cID** (Integer): The unique identifier of the parent comment for which sub-comments are to be retrieved. Required.

• **method** (String, optional): Method for sorting the sub-comments (controversial, popular). Default sorting is by the sub-comment ID in ascending order.

• **page** (Integer, optional): The page number of sub-comments to fetch. Default is 1.

• **pageSize** (Integer, optional): The number of sub-comments per page. Default is 5.

Responses:

• **200 OK**:

• Returns a list of sub-comments, each including details about the user who posted and vote counts.

• **400 Bad Request**:

• Returns an error if the comment ID is missing or not a valid integer, or if pagination parameters are invalid.

• Example Messages:

• "Comment ID must be a valid integer."

• "Page and pageSize must be positive integers."

• **404 Not Found**:

• Returns an error if no sub-comments are found for the given parent comment ID.

• Example Message: "Comment not found."

• **405 Method Not Allowed**:

• Returns an error if the request method is not GET.

• Example Message: "Method Not Allowed"

Sorting :

• **Controversial**: Sorts sub-comments based on a combination of upvotes, downvotes, and the presence of further sub-comments to highlight discussions with mixed reactions.

• **Popular**: Sorts sub-comments by the number of upvotes to showcase the most appreciated responses.

• **Default**: Sub-comments are sorted by their unique ID, reflecting the order in which they were added.

PARAMS

cID	14
page	1
method	controversial

report

This endpoint in the API manage report-related operations including report comment and blog, delete report, view report and hide blog and comment for admin. It supports various methods and ensures that the user is authenticated before performing some actions.

POST report comment

http://localhost:3000/api/report/comment

Creates a report for a specific comment.

Request Body:

• **cID** (Integer): The unique identifier of the comment to report. Required.

• **explanation** (String): The reason or explanation for the report. Required.

Responses:

• **201 Created**:

• Returns a success message and details of the newly created comment report.

• Example Message: "Comment report created successfully"

• **400 Bad Request**:

• Returns an error if the comment ID or explanation is missing.

• Example Message: "Comment ID and explanation are required."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the specified comment does not exist.

• Example Message: "Comment post not found"

• **409 Conflict**:

• Returns an error if a report has already been submitted by the user for the same comment.

• Example Message: "You have already reported this comment."

Body raw (json)

```json
{"cID":17, "explanation":"i don't like this"}
```

DELETE delete comment report

http://localhost:3000/api/report/comment?cID=16

Deletes a previously submitted comment report.

Query Parameters:

• **cID** (Integer): The unique identifier of the comment report to delete. Required.

Responses:

• **200 OK**:

• Returns a success message indicating the comment report was successfully deleted.

• Example Message: "Comment report deleted successfully"

• **400 Bad Request**:

• Returns an error if the comment ID is missing.

• Example Message: "Comment ID is required for deletion."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the comment report cannot be found or is not owned by the user.

• Example Message: "Comment report not found or not yours to delete"

AUTHORIZATION Bearer Token

Token	<token>

PARAMS

cID	16

DELETE delete blog report

http://localhost:3000/api/report/blog?bID=3

Deletes a previously submitted blog report.

Query Parameters:

• **bID** (Integer): The unique identifier of the blog report to delete. Required.

Responses:

• **200 OK**:

• Returns a success message indicating the blog report was successfully deleted.

• Example Message: "Blog report deleted successfully"

• **400 Bad Request**:

• Returns an error if the blog ID is missing.

• Example Message: "Blog ID is required for deletion."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the blog report cannot be found or is not owned by the user.

• Example Message: "Blog report not found or not yours to delete"

PARAMS

bID	3

Body raw (json)

```
json

{"bID":4}
```

POST report blog

http://localhost:3000/api/report/blog

Creates a report for a specific blog post.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to report. Required.

• **explanation** (String): The reason or explanation for the report. Required.

Responses:

• **201 Created**:

• Returns a success message and details of the newly created blog report.

• Example Message: "Blog report created successfully"

• **400 Bad Request**:

• Returns an error if the blog ID or explanation is missing.

• Example Message: "Blog ID and explanation are required."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the specified blog does not exist.

• Example Message: "Blog post not found"

• **409 Conflict**:

• Returns an error if a report has already been submitted by the user for the same blog.

• Example Message: "You have already reported this blog."

Body raw (json)

```json
{"bID":3, "explanation":"I hate this"}
```

GET get all reports for admin

http://localhost:3000/api/report/admin?type=comment

Allows authorized administrators to retrieve blog or comment data based on specified criteria and pagination settings.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Query Parameters:

• **type** (String): Specifies the data type to retrieve (blog or comment). Required.

• **page** (Integer, optional): The page number of results to fetch. Default is 1.

• **pageSize** (Integer, optional): The number of items per page. Default is 5.

Responses:

• **200 OK**:

• Returns a list of blogs or comments based on the specified type and pagination settings.

• **400 Bad Request**:

• Returns an error if the type parameter is missing or invalid.

• Example Message: "Invalid type parameter"

• **401 Unauthorized**:

• Returns an error if the authorization token is missing or invalid.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have administrative permissions.

• Example Message: "Insufficient permissions"

PARAMS

type	comment

PATCH admin edit hidden status

http://localhost:3000/api/report/admin

Allows authorized administrators to update the visibility status of a specified blog or comment.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **id** (Integer): The unique identifier of the blog or comment to be updated. Required.

• **type** (String): Specifies whether the item is a blog or comment. Required.

• **hidden** (Boolean): The new visibility status to set. Required.

Responses:

• **200 OK**: Returns the updated blog or comment data.

• **400 Bad Request**:

• Returns an error if any parameters are missing or invalid.

• Example Messages:

• "Hidden must be defined as true or false."

• "Type must be 'blog' or 'comment'."

• **401 Unauthorized**:

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have the necessary permissions.

• Example Message: "Insufficient permissions"

• **404 Not Found**:

• Returns an error if the specified blog or comment does not exist.

• Example Messages:

• "Blog not found"

• "Comment not found"

Body raw (json)

```json
{
    "id": 14,
    "type": "blog",
    "hidden":true
}
```

Scriptorium

Welcome to the Scriptorium Backend API documentation.

user

This contains API endpoints related to user management functions within the Scriptorium Backend. It includes operations such as user registration, login, profile editing, and the display of all blogs and templates created by the user.

POST sign up

http://localhost:3000/api/user/signup

It supports content type json for standard registration data and multipart/form-data for registrations including a profile picture upload using Multer.

Payload:

• firstName (string): First name of the user.

• lastName (string): Last name of the user.

• email (string): Email address of the user, must be unique.

• password (string): Password for the user account.

• phoneNum (integer, optional): Phone number of the user.

• role (string, default “user”): Role of the user in the system.

• avatar (file, optional): Profile picture to upload.

Responses:

• **201 Created**:

`{ "message": "User registered successfully", "user": { "uID": 5, "firstName": "test", "lastName": "admin", "email": " admin1234@gmail.com ", "password": "$2b$10$cmzZbv1xNWQQkz3X6vVe4.WySpGnvkasIqMSxsK/7jG7gX5eFybq6", "avatar": "public/avatar/default.jpg", "phoneNum": 1234, "role": "admin" } }`

• **400 Bad Request**: Missing required fields or incorrect data format.

• **409 Conflict**: Email already registered.

Body raw (json)

```
json

{
    "firstName":"test",
    "lastName":"admin",
    "email":"admin1234@gmail.com",
    "password":"123",
    "phoneNum":1234,
    "role":"admin"
}
```

POST login

http://localhost:3000/api/user/login

This endpoint authenticates a user by their email and password. It validates the user credentials and returns access and refresh tokens upon successful authentication.

Payload :

• email (string): The user's email address.

• password (string): The user's password.

Responses:

• **200 OK**:

`{ "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzExNjMsImV4cCI6MTczMDY3Mjk2M30.XAWdaethqNIYf-4M-X2awM1JgQ5BDn46Bfhpe6AOboA", "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzExNjMsImV4cCI6MTczMTI3NTk2M30.STBTw4acY8eqXmR6IWrmbvF02PmYThbMMQreAc65MnI" }`

• **400 Bad Request**: Missing either email or password.

• **401 Unauthorized**: Invalid credentials.

• **405 Method Not Allowed**: If any method other than POST is used.

Body raw (json)

```json
{
    "email":"admin123@gmail.com",
    "password":"123"
}
```

POST refresh token

http://localhost:3000/api/user/refresh

This endpoint validates the user's access token provided in the authorization header and genreates new tokens.

Authorization:

• Authorization (string): Bearer access token obtained during user login.

Responses:

• **200 OK**: Successful validation.

`{ "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzE0ODksImV4cCI6MTczMDY3MzI4OX0.3dZBQpCs98bnpkuBNCJHO_ZElUCJHX3D2nO5BlnXalA", "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyZW1haWwiOiJhZG1pbjEyM0BnbWFpbC5jb20iLCJpYXQiOjE3MzA2NzE0ODksImV4cCI6MTczMTI3NjI4OX0.OnmX7ZcBtNOhrC-SX5TnN84tazXcN0P2ntJkr88HTWs" }`

• **401 Unauthorized**: Returned if the token is invalid or expired.

• **405 Method Not Allowed**: Returned if the request method is not POST.

GET view profile

http://localhost:3000/api/user/profile

Retrieves the authenticated user's profile information. It verifies the user's identity using an authorization token and returns the user's profile details.

Authorization:

Requires a valid authorization token provided in the request header.

Responses:

• **200 OK**: Successfully retrieves the user profile.

`{ "message": "User profile retrieved successfully", "user": { "uID": 4, "firstName": "teacct", "lastName": "user221", "email": " admin123@gmail.com ", "password": "$2b$10$SGAns/dN5Jq5LSq2dBGk3.8/TIxK1RjxDGvhQmQVJ/oWTYJOMIzWO", "avatar": "public/avatar/default.jpg", "phoneNum": 1234, "role": "admin" } }`

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

PATCH update profile

http://localhost:3000/api/user/profile

Allows the authenticated user to update their profile information. It handles data in both son and multipart/form-data formats, allowing updates to fields like firstName, lastName, phoneNum, and the profile avatar.

Authorization:

Requires a valid authorization token provided in the request header.

Payload:

• firstName: Optional string, must be non-empty if provided.

• lastName: Optional string, must be non-empty if provided.

• phoneNum: Optional number.

• avatar: Optional file, handled via multipart/form-data.

Responses:

• **200 OK**: Profile updated successfully.

`{ "message": "Profile updated successfully", "user": { "uID": 4, "firstName": "updated", "lastName": "name", "email": " admin123@gmail.com ", "password": "$2b$10$SGAns/dN5Jq5LSq2dBGk3.8/TIxK1RjxDGvhQmQVJ/oWTYJOMIzWO", "avatar": "public/avatar/default.jpg", "phoneNum": 123424, "role": "admin" } }`

• **400 Bad Request**: Validation errors for input data.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

Body raw (json)

json

```
{
    "firstName":"updated",
    "lastName":"name",
    "phoneNum":123424
}
```

GET view blogs by user

http://localhost:3000/api/user/blog?page=1&pageSize=5

Retrieves a paginated list of blogs created by the authenticated user. This endpoint verifies the user's identity using an authorization token and returns blogs associated with the user, including related tags and templates.

Authorization:

Requires a valid authorization token provided in the request header.

Query Parameters:

• page: Specifies the page number in the pagination sequence.

• pageSize: Specifies the number of blog entries per page.

Responses:

• **200 OK**: Successfully retrieves the list of blogs along with pagination details.

`{ "blogs": [{ "bID": 12, "title": "testing1", "description": "description 8", "hidden": false, "uID": 4, "tags": [{ "id": 1, "value": "Python" }, { "id": 2, "value": "Java" }, { "id": 4, "value": "python" }, { "id": 6, "value": "easy" }], "templates": [] }], "totalCount": 1, "page": 1, "pageSize": 5 }`

• **400 Bad Request**: Missing or invalid pagination parameters.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

PARAMS

page	1
pageSize	5

GET view templates by user

http://localhost:3000/api/user/template?page=1&pageSize=2

Retrieves a paginated list of templates created by the authenticated user. This endpoint verifies the user's identity using an authorization token and returns templates associated with the user, including related tags.

Authorization:

Requires a valid authorization token provided in the request header.

Query Parameters:

• page: Specifies the page number in the pagination sequence.

• pageSize: Specifies the number of template entries per page.

Responses:

• **200 OK**: Successfully retrieves the list of templates along with pagination details.

`{ "templates": [ { "tID": 2, "title": "", "explanation": null, "code": null, "fork": false, "uID": 4, "tags": [] }, { "tID": 3, "title": "cads", "explanation": "fat", "code": "dfsdf", "fork": false, "uID": 4, "tags": [] } ], "totalCount": 6, "page": 1, "pageSize": 2 }`

• **400 Bad Request**: Missing or invalid pagination parameters.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User not found.

PARAMS

page	1
pageSize	2

code execution

Executes code submitted by the user for 10 seconds, supporting various programming languages.

Request Body:

• code: The source code to execute.

• input: Input parameters for the code, if required.

• language: Programming language of the code (e.g., "javascript", "python").

Responses:

• **200 OK**: Returns the output of the executed code.

```
{ "output": "Hello from Python\n" }
```

• **400 Bad Request**: If the specified programming language is unsupported or got error when executing the code.

```
{ "error": "Failed to execute code: Execution timed out" }
```

• **405 Method Not Allowed**: If the HTTP method is not POST.

POST Python No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from Python\n"}
```

Body raw (json)

```
json
{
  "code": "print('Hello from Python')",
  "input": "",
  "language": "python"
}
```

POST Python w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello, PythonUser1 and PythonUser2\n" }
```

Body raw (json)

json

```
{
  "code": "input_values = [input() for _ in range(2)]; print(f'Hello, {input_values[0]} and {in
  "input": ["PythonUser1", "PythonUser2"],
  "language": "python"
}
```

POST JavaScript No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from JavaScript\n" }
```

Body raw (json)

json

```
{
  "code": "console.log('Hello from JavaScript')",
  "input": "",
  "language": "javascript"
}
```

POST JavaScript w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "\n" }
```

Body raw (json)

json

```
{
  "code": "const readline = require('readline'); const rl = readline.createInterface({ input: r
```

```
  "input": ["JavaScriptUser1", "JavaScriptUser2"],


  "language": "javascript"
}
```

POST Java No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from Java\n" }
```

Body raw (json)

```
json

{
  "code": "public class Main { public static void main(String[] args) { System.out.println(\"He
  "input": "",
  "language": "java"
}
```

POST Java w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{
"output": "Hello, JavaUser1 and JavaUser2\n"
}
```

Body raw (json)

```
json

{
  "code": "import java.util.Scanner; public class Main { public static void main(String[] args)
```

```
  "input": ["JavaUser1", "JavaUser2"],


  "language": "java"
}
```

POST C No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from C\n" }
```

Body raw (json)

json

```
{
  "code": "#include <stdio.h> \n int main() { printf(\"Hello from C\\n\"); return 0; }",
  "input": "",
  "language": "c"
}
```

POST C w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello, CUser1 and CUser2\n" }
```

Body raw (json)

json

```
{
  "code": "#include <stdio.h>\n int main() { char input1[50], input2[50]; scanf(\"%s\", input1)
  "input": ["CUser1", "CUser2"],
  "language": "c"
}
```

POST C++ No Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello from C++\n" }
```

Body raw (json)

json

```
{
  "code": "#include <iostream> \n int main() { std::cout << \"Hello from C++\" << std::endl; re
  "input": "",
  "language": "cpp"
}
```

POST C++ w/ Inputs

http://localhost:3000/api/code/execute

Expected output: 200 OK

```
{ "output": "Hello, CppUser1 and CppUser2\n" }
```

Body raw (json)

json

```
{
  "code": "#include <iostream>\n int main() { std::string input1, input2; std::cin >> input1 >>
  "input": ["CppUser1", "CppUser2"],
  "language": "cpp"
}
```

POST Timeout Error w/ Interpreted Language

http://localhost:3000/api/code/execute

Expected output: 400 Bad Request

```
{ "error": "Failed to execute code: Execution timed out" }
```

Body raw (json)

```json
{
  "code": "while True: pass",
  "input": "",
  "language": "python"
}
```

POST Timeout Error w/ Compiled Language

http://localhost:3000/api/code/execute

Expected output: 400 Bad Request

```
{ "error": "Failed to execute code: Execution timed out" }
```

Body raw (json)

```json
{
  "code": "#include <stdio.h>\nint main() { while (1) {} return 0; }",
  "input": "",
  "language": "c"
}
```

POST Division By Zero error - Python

http://localhost:3000/api/code/execute

Expected output: 400 Bad Request

{ "error": "Failed to execute code: Runtime error: Division by zero detected in the code." }

Body raw (json)

```
json
{
  "code": "print(0/0)",
  "input": "",
  "language": "python"
}
```

code templates

This endpoint in the API manages template-related operations including creation, deletion, search, and updating of templates. It supports various methods and ensures that the user is authenticated before performing some actions.

POST create template

http://localhost:3000/api/templates

Allow logged in user to post code template.

Authorization:

Requires a valid authorization token provided in the request header.

Request Parameters:

• title: Title of the template (required).

• explanation: A brief description or explanation of the template (optional).

• tags: An array of tag values related to the template (optional).

• code: The actual code snippet for the template (optional).

• fork: Boolean indicating if the template is a fork of another template (defaults to false).

Responses:

• **201 Created**: Returns a success message and the newly created template.

{ "message": "Template created", "template": { "tID": 14, "title": "New Title !!", "explanation": "Updated explanation", "code": "Updated code", "fork": true, "uID": 4 } }

{ "explanation": "Updated explanation", "code": "Updated code", "fork": true, "uID": 4 }

• **400 Bad Request**: Missing required fields or invalid input format.

• **401 Unauthorized**: No authorization token provided or token validation failed.

• **409 Conflict**: A template with the given title already exists.

Body raw (json)

```json
{
  "title": "New Title !!",
  "explanation": "Updated explanation",
  "tags": ["updatedTag"],
  "code": "Updated code",
  "fork": true
}
```

GET search template

http://localhost:3000/api/templates?title =title&tags=python&tags=java&explanation=what&page=&limit=3

Search template based on given input. All given condition (case insensitive) must be matched to be shown in the search result.

Query Parameters:

• page: Specifies the page number in the pagination sequence (default: 1).

• pageSize: Specifies the number of template entries per page (default: 10).

• title: Filters templates based on title inclusion (optional).

• tags: Filters templates based on associated tags (optional, can be multiple strings).

• explanation: Filters templates based on text included in the explanation (optional).

Responses:

• **200 OK**: Successfully retrieves the list of templates along with pagination details. (not the output for the Query Params show below)

{ "templates": [{ "tID": 1, "title": "Updated Title 1", "explanation": "Updated explanation", "code": "Updated code", "fork": false, "uID": 1 }, { "tID": 2, "title": "", "explanation": null, "code": null, "fork": false, "uID": 4 }, { "tID": 3, "title": "cads", "explanation": "fat", "code": "dfsdf", "fork": false, "uID": 4 }], "totalPages": 5, "currentPage": 1 }

• **400 Bad Request**: Invalid query parameters.

• **401 Unauthorized**: No authorization token provided or token is invalid.

• **404 Not Found**: No templates found or user not found.

PARAMS

title	title
tags	python
tags	java
explanation	what
page	
limit	3

DELETE delete template create by user

http://localhost:3000/api/templates?tID=4

Delete a template created by the user.

Authorization

Bearer token required for verifying user authentication.

Parameters

• tID: The ID of the template to be deleted.

Responses

• **200 OK**: Returns a success message indicating that the template was successfully deleted.

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

• **400 Bad Request**:

• Returns an error if the tID is not provided or cannot be parsed as an integer.

• Returns an error if the user does not exist in the database.

• Returns an error if the tags is more than 10.

• **401 Unauthorized**: Returns an error if no authorization token is provided.

• **403 Forbidden**: Returns an error if the user does not have permission to delete the specified template.

• **404 Not Found**: Returns an error if the specified template is not found in the database.

PARAMS

tID	4

PATCH edit template created by user

http://localhost:3000/api/templates

Update a template created by the user. If a field is not provided, make it maintain as the orginal.

Authorization

Bearer token required for verifying user authentication.

Request Body

• tID (Integer): The unique identifier for the template to be updated. Required.

• title (String): The new title of the template.

• explanation (String): Detailed explanation or description of the template.

• tags (Array of Strings): Tags associated with the template for categorization or search.

• code (String): The code snippet or content of the template.

• fork (Boolean): Indicates whether the template is a fork of another template.

Responses

• **200 OK**: Returns a success message indicating that the template was successfully updated.

```
{ "message": "Template updated successfully" }
```

• **400 Bad Request**: Invalid query parameters.

• **401 Unauthorized**: No authorization token provided or token is invalid.

• **404 Not Found**: No templates found or user not found.

Body raw (json)

json

```
{
  "title": "Update",
  "explanation": "exp",
  "tags": ["JS","1","2","3","4", "5", "6", "342"],
  "code": "console.log('Hello World')",
  "fork": true,
  "tID": 7
}
```


blog

This endpoint in the API manageblog-related operations including creation, deletion, search, vote, and edit. It supports various methods and ensures that the user is authenticated before performing some actions.

GET search blogs

http://localhost:3000/api/blog/search?method=controversial&title=bloa

Search for Blogs based on input. All search criteria must be satisfied for a blog to be shown in the output.

Query Parameters:

- **title** (optional): Filters posts containing the specified title substring.
- **content** (optional): Filters posts containing the specified content substring.
- **tags** (optional): Filters posts associated with the specified tags (array of strings or a single string).
- **templates** (optional): Filters posts associated with specified templates (array of strings or a single string).
- **method** (optional): Determines the sorting method of the posts (controversial, popular, or default by newest).
- **page** (default: 1): Specifies the page number in the result set.
- **pageSize** (default: 5): Specifies the number of posts per page.

Responses:

- **200 OK**: Returns a list of blog posts that match the criteria.

`[{ "bID": 9, "title": "bloafawww", "description": "description 8", "hidden": false, "uID": 1, "tags": [{ "id": 1, "value": "Python" }, { "id": 2, "value": "Java" }, { "id": 4, "value": "python" }], "templates": [], "user": { "uID": 1, "firstName": "updated", "lastName": "name", "email": " user1@gmail.com ", "password": "$2b$10$dp731kUjvbVmzAyprEM2p.fI5F.NRDtBDQyiEPYOtjWoejw/IrH22", "avatar": "public/avatar/default.jpg", "phoneNum": 123424, "role": "user" }, "_count": { "upvoters": 0, "downvoters": 0, "comments": 0 } }]`

• **400 Bad Request**:

• Returns an error if page or pageSize are not positive integers.

• Returns an error if there is an invalid JSON format for tags or templates.

• Returns an error if tags or templates are not arrays or a single string.

• Returns an error for incorrect data types or invalid values for title and content.

• **401 Unauthorized**:

• Returns an error if no authorization token is provided or the token validation fails.

• **405 Method Not Allowed**:

• Returns an error if the request method is not GET.

Sorting:

• **controversial**: Sorts posts by the sum of upvoters, downvoters, and comments.

• **popular**: Sorts posts by the number of upvoters.

• **default**: Sorts posts by the newest based on their ID.

PARAMS

tags	python
tags	Java
method	controversial
title	bloa

POST create blog

http://localhost:3000/api/blog/create

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **title** (String): The title of the blog post. This is a required field and must be a non-empty string.

• **description** (String): A detailed description or body of the blog post. This is a required field and must be a non-empty string.

• **tags** (Array of Strings): A list of tags associated with the blog post. Optional field, defaults to an empty array, cannot contain more than 10 tags.

• **templates** (Array of Strings): A list of template titles associated with the blog post. Optional field, defaults to an empty array, cannot contain more than 10 templates.

Responses:

• **201 Created**: Returns a success message and details of the newly created blog post.

`{ "message": "Blog created successfully", "blog": { "bID": 13, "title": "testingsdvcsd", "description": "description 8", "hidden": false, "uID": 4, "tags": [{ "id": 1, "value": "Python" }, { "id": 2, "value": "Java" }, { "id": 4, "value": "python" }, { "id": 6, "value": "easy" }], "templates": [] } }`

• **400 Bad Request**:

• Title and description are required and must be strings.

• Tags and templates must be valid arrays of strings and cannot exceed 10 items each.

• One or more specified templates do not exist.

• **401 Unauthorized**: Invalid or missing authorization token.

• **404 Not Found**: User associated with the token does not exist.

• **405 Method Not Allowed**: Method used is not POST.

• **409 Conflict**: A blog with the provided title already exists.

• **500 Internal Server Error**: Unable to create blog post due to a database error.

Body raw (json)

```json
{
    "title":"testingsdvcsd",
    "description":"description 8",
    "tags":["Python", "Java", "python","easy"],
    "templates":[]
}
```

POST vote blog

http://localhost:3000/api/blog/vote

Logged in user can vote. If user already voded on the blog, overwrote it with the new request.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to vote on. Required.

• **voteType** (String): The type of vote to cast. Accepts either “upvote” or “downvote”. Required.

Responses:

• **200 OK**:Returns a success message and the updated vote count of the blog post.

`{ "message": "Successfully updated downvote", "blog": { "bID": 3, "title": "fdfvs", "description": "asfcedfvVARVRG", "hidden": false, "uID": 1, "_count": { "upvoters": 0, "downvoters": 2 } } }`

• **400 Bad Request**: Returns an error if the vote type is invalid or missing.

• **401 Unauthorized**: Returns an error if the authorization token is invalid or missing.

• **404 Not Found**: Returns an error if the user or blog post is not found.

• **405 Method Not Allowed**: Returns an error if the request method is not POST.

Body raw (json)

```
json

{"bID": 3,"voteType":"downvote"}
```

PATCH edit blog

http://localhost:3000/api/blog/edit

Allow user to edit blog they created.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to be updated. Required.

• **title** (String): The new title for the blog post. Optional.

• **description** (String): The new description or content for the blog post. Optional.

• **tags** (Array of Strings): A new list of tags to replace any existing tags. Optional.

• **templates** (Array of Integer IDs): A new list of template IDs to associate with the blog post. Optional.

• **hidden** (Boolean): The visibility status of the blog post. Optional.

Responses:

• **200 OK**:

• Returns a success message and the updated details of the blog post.

• Example Message: "Blog updated successfully"

• **400 Bad Request**:

• Returns an error if required fields are missing or malformed.

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have permission to edit the specified blog post.

• Example Message: "You do not have permission to edit this blog post"

• Returns an error if the blog post is marked as hidden and cannot be edited.

• Example Message: "You cannot edit this blog post"

• **404 Not Found**:

• Returns an error if the user or the specified blog post cannot be found.

• Example Message: "User not found"

• Example Message: "Blog post not found"

• **500 Internal Server Error**:

• Returns an error if there is an unexpected issue during the update process.

• Example Message: "Unable to update blog post, database error."

Body raw (json)

json

```
{"bID": 7, "title":"new title 1", "tags" : ["c++"]}
```

DELETE delete blog

http://localhost:3000/api/blog/delete?bID=5

Allow user to delete blog they created.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Query Parameters:

• **bID** (Integer): The unique identifier of the blog post to be deleted. Required.

Responses:

• **200 OK**:

• Returns a success message indicating the blog post was successfully deleted.

• Example Message: "Delete successfully"

• **400 Bad Request**:

• Returns an error if the blog ID (bID) is not provided or is in an invalid format.

• Example Message: "Invalid or missing blog ID"

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have permission to delete the specified blog post.

• Example Message: "You do not have permission to delete this blog post"

• **404 Not Found**:

• Returns an error if the user or the specified blog post cannot be found.

• Example Message: "User not found"

• Example Message: "Blog post not found"

• **405 Method Not Allowed**:

• Returns an error if the method used is not DELETE.

• Example Message: "Method DELETE Not Allowed"

• **500 Internal Server Error**:

• Returns an error if there is an unexpected issue during the deletion process.

• Example Message: "Unable to delete blog post, database error."

PARAMS

bID	5

GET get blog

http://localhost:3000/api/blog?bID=3

Allows fetching of a blog post along with its comments based on provided sorting and pagination parameters.

Query Parameters:

• **bID** (Integer): The unique identifier of the blog post to retrieve. Required.

• **method** (String, optional): Sorting method for the comments (controversial, popular). Default is sorting by comment ID in ascending order.

• **page** (Integer, optional): The page number of comments to fetch. Default is 1.

• **pageSize** (Integer, optional): The number of comments per page. Default is 5.

Responses:

• **200 OK**:

• Returns the blog post and a paginated list of its comments, including the user details and vote counts for each comment.

• **400 Bad Request**:

• Returns an error if the bID is not provided or is not a valid integer.

• Returns an error if page or pageSize are not positive integers.

• Example Messages:

• "Blog ID is required."

• "Blog ID must be a valid integer."

• "Page and pageSize must be positive integers."

• **404 Not Found**:

• Returns an error if no blog post or no comments are found for the given blog ID.

• Example Messages:

• "Blog post not found."

• "Comment not found."

• **405 Method Not Allowed**:

• Returns an error if the request method is not GET.

• Example Message: "Method Not Allowed"

PARAMS

bID	3

comment

This endpoint in the API manage comment-related operations including creation, search, and vote. It supports various methods and ensures that the user is authenticated before performing some actions.

POST create comment

http://localhost:3000/api/comment/create

Allows a logged-in user to post comments on a blog post or reply to an existing comment.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to comment on. Required.

• **content** (String): The content of the comment. Must be a non-empty string. Required.

• **pID** (Integer, optional): The unique identifier of the parent comment, if replying to an existing comment.

Responses:

• **201 Created**:

• Returns a success message and the details of the newly created comment.

• Example Message: "Comment created successfully"

• **400 Bad Request**:

• Returns an error if the blog ID or content is missing or invalid, or if the content is empty.

• Example Message: "Blog ID and content are required and content must be a non-empty string."

• Returns an error if the parent comment does not belong to the provided blog ID when replying to a comment.

• Example Message: "Parent comment does not belong to the provided blog."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the user or the specified blog post/comment cannot be found.

• User not found: "User not found"

• Blog post not found or parent comment not found: "Parent comment not found"

• **405 Method Not Allowed**:

• Returns an error if the method used is not POST.

• Example Message: "Method Not Allowed"

Body raw (json)

```
json

{"bID":3, "content":"7", "pID":14}
```

POST vote comment

http://localhost:3000/api/comment/vote

Allows a logged-in user to cast a vote (upvote or downvote) on a comment.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **cID** (Integer): The unique identifier of the comment to vote on. Required.

• **voteType** (String): The type of vote to cast. Accepts “upvote” or “downvote”. Required.

Responses:

• **200 OK**:

• Returns a success message and the updated details of the comment including the current counts of upvotes and downvotes.

• Example Message: "Successfully updated upvote"

• **400 Bad Request**:

• Returns an error if the vote type is invalid or missing.

• Example Message: "Invalid vote type specified. Must be either 'upvote' or 'downvote'."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the specified comment cannot be found.

• Example Message: "Comment not found"

• **403 Forbidden**:

• Returns an error if the user does not have permission to vote, such as attempting to change a vote improperly or when the comment is locked from voting.

• Example Message: "You do not have permission to delete this blog post"

Body raw (json)

```json
{"cID":15, "voteType":"upvote"}
```

GET get comment

http://localhost:3000/api/comment?cID=14&page=1&method=controversial

Allows fetching of sub-comments related to a parent comment, with options for sorting and pagination.

Method:

• **GET**

URL:

• /api/comments/subcomments

Query Parameters:

• **cID** (Integer): The unique identifier of the parent comment for which sub-comments are to be retrieved. Required.

• **method** (String, optional): Method for sorting the sub-comments (controversial, popular). Default sorting is by the sub-comment ID in ascending order.

• **page** (Integer, optional): The page number of sub-comments to fetch. Default is 1.

• **pageSize** (Integer, optional): The number of sub-comments per page. Default is 5.

Responses:

• **200 OK**:

• Returns a list of sub-comments, each including details about the user who posted and vote counts.

• **400 Bad Request**:

• Returns an error if the comment ID is missing or not a valid integer, or if pagination parameters are invalid.

• Example Messages:

• "Comment ID must be a valid integer."

• "Page and pageSize must be positive integers."

• **404 Not Found**:

• Returns an error if no sub-comments are found for the given parent comment ID.

• Example Message: "Comment not found."

• **405 Method Not Allowed**:

• Returns an error if the request method is not GET.

• Example Message: "Method Not Allowed"

Sorting :

• **Controversial**: Sorts sub-comments based on a combination of upvotes, downvotes, and the presence of further sub-comments to highlight discussions with mixed reactions.

• **Popular**: Sorts sub-comments by the number of upvotes to showcase the most appreciated responses.

• **Default**: Sub-comments are sorted by their unique ID, reflecting the order in which they were added.

PARAMS

cID	14
page	1
method	controversial

report

This endpoint in the API manage report-related operations including report comment and blog, delete report, view report and hide blog and comment for admin. It supports various methods and ensures that the user is authenticated before performing some actions.

POST report comment

http://localhost:3000/api/report/comment

Creates a report for a specific comment.

Request Body:

• **cID** (Integer): The unique identifier of the comment to report. Required.

• **explanation** (String): The reason or explanation for the report. Required.

Responses:

• **201 Created**:

• Returns a success message and details of the newly created comment report.

• Example Message: "Comment report created successfully"

• **400 Bad Request**:

• Returns an error if the comment ID or explanation is missing.

• Example Message: "Comment ID and explanation are required."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the specified comment does not exist.

• Example Message: "Comment post not found"

• **409 Conflict**:

• Returns an error if a report has already been submitted by the user for the same comment.

• Example Message: "You have already reported this comment."

Body raw (json)

```json
{"cID":17, "explanation":"i don't like this"}
```

DELETE delete comment report

http://localhost:3000/api/report/comment?cID=16

Deletes a previously submitted comment report.

Query Parameters:

• **cID** (Integer): The unique identifier of the comment report to delete. Required.

Responses:

• **200 OK**:

• Returns a success message indicating the comment report was successfully deleted.

• Example Message: "Comment report deleted successfully"

• **400 Bad Request**:

• Returns an error if the comment ID is missing.

• Example Message: "Comment ID is required for deletion."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the comment report cannot be found or is not owned by the user.

• Example Message: "Comment report not found or not yours to delete"

AUTHORIZATION Bearer Token

Token	<token>

PARAMS

cID	16

DELETE delete blog report

http://localhost:3000/api/report/blog?bID=3

Deletes a previously submitted blog report.

Query Parameters:

• **bID** (Integer): The unique identifier of the blog report to delete. Required.

Responses:

• **200 OK**:

• Returns a success message indicating the blog report was successfully deleted.

• Example Message: "Blog report deleted successfully"

• **400 Bad Request**:

• Returns an error if the blog ID is missing.

• Example Message: "Blog ID is required for deletion."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the blog report cannot be found or is not owned by the user.

• Example Message: "Blog report not found or not yours to delete"

PARAMS

bID	3

Body raw (json)

```
json

{"bID":4}
```

POST report blog

http://localhost:3000/api/report/blog

Creates a report for a specific blog post.

Request Body:

• **bID** (Integer): The unique identifier of the blog post to report. Required.

• **explanation** (String): The reason or explanation for the report. Required.

Responses:

• **201 Created**:

• Returns a success message and details of the newly created blog report.

• Example Message: "Blog report created successfully"

• **400 Bad Request**:

• Returns an error if the blog ID or explanation is missing.

• Example Message: "Blog ID and explanation are required."

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid or missing.

• Example Message: "Invalid or missing authorization token"

• **404 Not Found**:

• Returns an error if the specified blog does not exist.

• Example Message: "Blog post not found"

• **409 Conflict**:

• Returns an error if a report has already been submitted by the user for the same blog.

• Example Message: "You have already reported this blog."

Body raw (json)

```json
{"bID":3, "explanation":"I hate this"}
```

GET get all reports for admin

http://localhost:3000/api/report/admin?type=comment

Allows authorized administrators to retrieve blog or comment data based on specified criteria and pagination settings.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Query Parameters:

• **type** (String): Specifies the data type to retrieve (blog or comment). Required.

• **page** (Integer, optional): The page number of results to fetch. Default is 1.

• **pageSize** (Integer, optional): The number of items per page. Default is 5.

Responses:

• **200 OK**:

• Returns a list of blogs or comments based on the specified type and pagination settings.

• **400 Bad Request**:

• Returns an error if the type parameter is missing or invalid.

• Example Message: "Invalid type parameter"

• **401 Unauthorized**:

• Returns an error if the authorization token is missing or invalid.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have administrative permissions.

• Example Message: "Insufficient permissions"

PARAMS

type	comment

PATCH admin edit hidden status

http://localhost:3000/api/report/admin

Allows authorized administrators to update the visibility status of a specified blog or comment.

Authorization:

• Requires a valid authorization token provided in the Authorization header.

Request Body:

• **id** (Integer): The unique identifier of the blog or comment to be updated. Required.

• **type** (String): Specifies whether the item is a blog or comment. Required.

• **hidden** (Boolean): The new visibility status to set. Required.

Responses:

• **200 OK**: Returns the updated blog or comment data.

• **400 Bad Request**:

• Returns an error if any parameters are missing or invalid.

• Example Messages:

• "Hidden must be defined as true or false."

• "Type must be 'blog' or 'comment'."

• **401 Unauthorized**:

• **401 Unauthorized**:

• Returns an error if the authorization token is invalid.

• Example Message: "Invalid or missing authorization token"

• **403 Forbidden**:

• Returns an error if the user does not have the necessary permissions.

• Example Message: "Insufficient permissions"

• **404 Not Found**:

• Returns an error if the specified blog or comment does not exist.

• Example Messages:

• "Blog not found"

• "Comment not found"

Body raw (json)

```json
{
    "id": 14,
    "type": "blog",
    "hidden":true
}
```