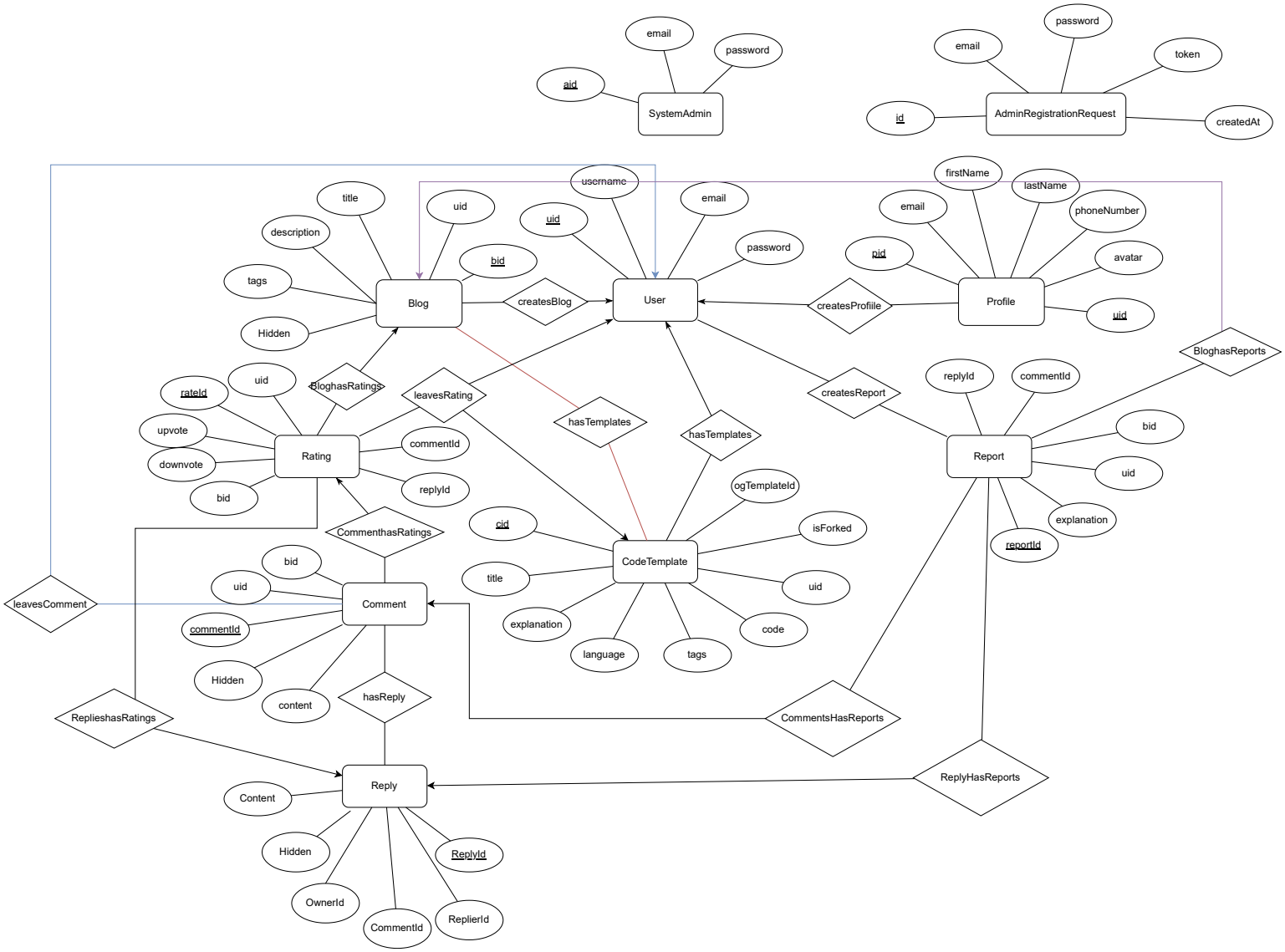
email
password
aid
SystemAdmin
password
email
token
id
AdminRegistrationRequest
createdAt
username
email
firstName
lastName
phoneNumber
email
title
uid
uid
description
password
pid
avatar
bid
tags
Blog
createsBlog
User
createsProfile
Profile
Hidden
uid
BloghasReports
uid
BloghasRatings
leavesRating
createsReport
replyId
commentId
rateId
hasTemplates
hasTemplates
bid
upvote
Rating
commentId
Report
bid
downvote
uid
ogTemplateId
replyId
explanation
CommenthasRatings
cid
isForked
reportId
CodeTemplate
bid
title
uid
uid
Comment
explanation
code
commentId
language
tags
Hidden
content
hasReply
RepliieshasRatings
CommentsHasReports
ReplyHasReports
Reply
Content
Hidden
ReplyId
OwnerId
CommentId
ReplierId
leavesComment

ERDiagram

ChatGpt was utilized to generate the documentation

Models Design

1. User

Description: Represents users of the application.
Attributes:

- `uid`: Unique identifier (Primary Key).
- `username`: Unique username for the user.
- `email`: Unique email for user identification.
- `password`: User's password.
- `profile`: Optional reference to the user profile.
- `codeTemplates`: Array of code templates created by the user.
- `blog`: Array of blogs written by the user.
- `comments`: Array of comments made by the user.
- `ratings`: Array of ratings given by the user.
- `reports`: Array of reports filed by the user.
- `ownedReplies`: Replies made by the user (as the owner).
- `replies`: Replies made by the user (as a replier).

---

2. AdminRegistrationRequest

Description: Represents requests for admin registration.
Attributes:

- `id`: Unique identifier (Primary Key).
- `email`: Unique email for the admin request.
- `password`: Password for the admin account.
- `token`: Unique token for the request.
- `createdAt`: Timestamp of when the request was created.

---

3. SystemAdmin

Description: Represents system administrators.
Attributes:

- `aid`: Unique identifier (Primary Key).
- `email`: Unique email for the admin.
- `password`: Password for the admin account.

---

4. Profile

Description: Stores user profile information.
Attributes:

- `pid`: Unique identifier (Primary Key).
- `email`: User's email.
- `firstName`: User's first name.
- `lastName`: User's last name.
- `avatar`: URL to the profile picture.
- `phoneNumber`: User's phone number.
- `uid`: Unique user ID (Foreign Key to User).

---

5. CodeTemplate

Description: Represents a code template created by a user.
Attributes:

- `cid`: Unique identifier (Primary Key).
- `title`: Title of the code template.
- `explanation`: Optional explanation of the code.
- `language`: Programming language of the template.
- `tags`: Tags associated with the template.
- `code`: The actual code.
- `uid`: Unique user ID (Foreign Key to User).
- `isForked`: Indicates if the template is a fork of another.
- `ogTemplateId`: ID of the original template if it's a fork.
- `forks`: Array of forks of this template.
- `ogTemplate`: Reference to the original template.

---

6. Blog

Description: Represents a blog post created by a user.
Attributes:

- `bid`: Unique identifier (Primary Key).
- `title`: Title of the blog post.
- `description`: Description of the blog post.
- `tags`: Tags associated with the blog.
- `Hidden`: Indicates if the blog is hidden.
- `uid`: Unique user ID (Foreign Key to User).
- `comments`: Array of comments made on the blog.
- `ratings`: Array of ratings given to the blog.
- `reports`: Array of reports filed against the blog.
- `codeTemplates`: Array of code templates associated with the blog.

---

7. Comment

Description: Represents a comment made on a blog.
Attributes:

- `commentId`: Unique identifier (Primary Key).
- `bid`: Blog ID (Foreign Key to Blog).
- `uid`: User ID (Foreign Key to User).
- `Hidden`: Indicates if the comment is hidden.
- `content`: The content of the comment.
- `ratings`: Array of ratings given to the comment.
- `reports`: Array of reports filed against the comment.
- `replies`: Array of replies to the comment.

---

8. Reply

Description: Represents a reply to a comment.
Attributes:

- `replyId`: Unique identifier (Primary Key).
- `ownerId`: User ID of the comment owner (Foreign Key to User).
- `commentId`: Comment ID (Foreign Key to Comment).

- `replierId`: User ID of the replier (Foreign Key to User).
- `content`: The content of the reply.
- `ratings`: Array of ratings given to the reply.
- `reports`: Array of reports filed against the reply.

---

9. Rating

Description: Represents a rating given by a user.
Attributes:

- `rateId`: Unique identifier (Primary Key).
- `upvote`: Indicates if the rating is an upvote.
- `downvote`: Indicates if the rating is a downvote.
- `uid`: User ID (Foreign Key to User).
- `bid`: Blog ID (Foreign Key to Blog, optional).
- `commentId`: Comment ID (Foreign Key to Comment, optional).
- `replyId`: Reply ID (Foreign Key to Reply, optional).optional).

---

10. Report

Description: Represents a report filed against a blog, comment, or user.
Attributes:

- `reportId`: Unique identifier (Primary Key).
- `content`: The content of the report.
- `uid`: User ID (Foreign Key to User).
- `bid`: Blog ID (Foreign Key to Blog, optional).
- `commentId`: Comment ID (Foreign Key to Comment, optional).
- `replyId`: Reply ID (Foreign Key to Reply, optional).

Prisma Schema Relationships

1. User

- **One-to-Many** with `Profile`: A user can have one profile.
- **One-to-Many** with `CodeTemplate`: A user can create multiple code templates.
- **One-to-Many** with `Blog`: A user can write multiple blogs.

- **One-to-Many** with `Comment`: A user can make multiple comments.
- **One-to-Many** with `Rating`: A user can give multiple ratings.
- **One-to-Many** with `Report`: A user can file multiple reports.
- **One-to-Many** with `Reply` (as owner): A user can own multiple replies.
- **One-to-Many** with `Reply` (as replier): A user can reply to multiple comments.

2. AdminRegistrationRequest

- **None**: This model does not establish any relationships with other models.

3. SystemAdmin

- **None**: This model does not establish any relationships with other models.

4. Profile

- **One-to-One** with `User`: Each profile is associated with one user.

5. CodeTemplate

- **Many-to-One** with `User`: Each code template is created by one user.
- **One-to-Many** with `Blog`: A code template can be associated with multiple blogs.
- **Many-to-One** with `CodeTemplate` (ogTemplate): A code template can be a fork of one original template.
- **One-to-Many** with `CodeTemplate` (forks): An original template can have multiple forks.

6. Blog

- **Many-to-One** with `User`: Each blog is written by one user.
- **One-to-Many** with `Comment`: A blog can have multiple comments.
- **One-to-Many** with `Rating`: A blog can receive multiple ratings.
- **One-to-Many** with `Report`: A blog can receive multiple reports.
- **Many-to-Many** with `CodeTemplate`: A blog can have multiple code templates associated with it.

7. Comment

- **Many-to-One** with `Blog`: Each comment is associated with one blog.
- **Many-to-One** with `User`: Each comment is made by one user.
- **One-to-Many** with `Reply`: A comment can have multiple replies.

- **One-to-Many** with `Rating`: A comment can receive multiple ratings.
- **One-to-Many** with `Report`: A comment can receive multiple reports.

8. Reply

- **Many-to-One** with `Comment`: Each reply is associated with one comment.
- **Many-to-One** with `User` (as owner): Each reply can be owned by one user.
- **Many-to-One** with `User` (as replier): Each reply can be made by one user.
- **One-to-Many** with `Rating`: A reply can receive multiple ratings.
- **One-to-Many** with `Report`: A reply can receive multiple reports.

9. Rating

- **Many-to-One** with `User`: Each rating is given by one user.
- **Many-to-One** with `Blog`: A rating can be associated with one blog.
- **Many-to-One** with `Comment`: A rating can be associated with one comment.
- **Many-to-One** with `Reply`: A rating can be associated with one reply.

10. Report

- **Many-to-One** with `User`: Each report is filed by one user.
- **Many-to-One** with `Blog`: A report can be associated with one blog.
- **Many-to-One** with `Comment`: A report can be associated with one comment.
- **Many-to-One** with `Reply`: A report can be associated with one reply.

API DOCUMENTATION

Blog Endpoints

Base URL for Blogs:

```
http://localhost:3000/api/Blogs
```

1. Create a New Blog Post

- **Endpoint**: `/api/Blogs`
- **Method**: `POST`
- **Description**: Creates a new blog post. Only authenticated users with the role `"USER"` are authorized to create a blog.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>`

Request Body

Field	Type	Description
`title`	`string`	Title of the blog post.
`description`	`string`	Description of the blog post.
`tags`	`string`	JSON array of tags for the blog.
`codeTemplateIds`	`array` of `number`	Array of template IDs associated with the blog.

Example Request

```
POST /api/Blogs
Authorization: Bearer <JWT_TOKEN>

{
    "title": "my blog",
    "description": "this is my first blog :)",
    "tags": "[\"Python\"]",
    "codeTemplateIds": [1]
}
```

Example Response: 201 - Created

```
{
    "bid": 3,
    "title": "my blog",
    "description": "this is my first blog :)",
    "tags": "[\"Python\"]",
    "Hidden": false,
    "uid": 1
}
```

Response Codes

- **201 Created**: Blog post created successfully.
- **400 Bad Request**: Missing or incorrect fields in the request body.
- **401 Unauthorized**: Invalid or missing JWT token.
- **403 Forbidden**: User does not have permission to create a blog.
- **500 Internal Server Error**: Error occurred while creating the blog.

---

2. Get Blog Posts

- **Endpoint**: `/api/Blogs`
- **Method**: `GET`
- **Description**: Retrieves a list of blog posts with optional filters, pagination, and visibility settings based on user authentication and role.

Authentication

- Optional: a valid authentication token (accessToken) stored in cookies or in header as bearer token.

Query Parameters

Parameter	Type	Description
`bid`	`number`	Filters by a specific blog ID.
`title`	`string`	Searches for blogs containing this substring in the title.
`description`	`string`	Searches for blogs containing this substring in the description.
`tags`	`string`	JSON array of tags to filter by (e.g., `["API", "Web"]`). A single tag as a string is also accepted.
`uid`	`number`	Filters by the user ID of the blog creator.
`codeTemplateIds`	`string`	JSON array of code template IDs associated with the blog. A single ID as a number is also accepted.
`page`	`number`	Page number for pagination (default: 1).
`limit`	`number`	Number of results per page (default: 10).

Filtering and Visibility Rules

- **Authenticated Users with `USER` Role**:
  - Can view both public and their own private/hidden blogs.
  - Filters based on visibility (`Hidden: false`) or by creator (`uid: user.uid`).
- **Unauthenticated Users or Visitors**:
  - Only public (unhidden) blogs are visible (`Hidden: false`).

Example Request

```
GET /api/Blogs?title=my blog&tags=["Python"]&page=1&limit=10
Authorization: Bearer <JWT_TOKEN>
```

Example Response (200 - OK)

```
[
    {
        "bid": 2,
        "title": "my blog",
        "description": "this is my first blog :)",
        "tags": "[\"Python\"]",
        "Hidden": false,
        "uid": 1
    }
]
```

Response Codes

- **200 OK**: Blogs retrieved successfully.
- **400 Bad Request**: Invalid query parameter format (e.g., non-JSON tags or codeTemplateIds).
- **500 Internal Server Error**: Error occurred while retrieving blogs.

---

3. Update Blog

- **Endpoint**: `/api/Blogs/{id}`
- **Method**: `PUT`
- **Description**: Allows an authenticated user to update a blog's details. Only the blog owner can make updates.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies or in header as bearer token.
- Users must have the `USER` role to update a blog.

Request Body

Parameter	Type	Required	Description
`title`	`string`	Optional	The updated title of the blog.
`description`	`string`	Optional	The updated description of the blog.
`tags`	`string`	Optional	A JSON array of tags associated with the blog.

`codeTemplateIds`	`number[]`	Optional	Array of code template IDs to associate with the blog.

Example Request

```
PUT /api/Blogs/1
Authorization: Bearer <JWT_TOKEN>

{
  "title": "Updated Blog Title!",
  "description": "updated description of my blog!!",
  "tags": "[\"Web\"]",
  "codeTemplateIds": [1]
}
```

Example Response (200 - OK)

```
{
    "bid": 1,
    "title": "Updated Blog Title!",
    "description": "updated description of my blog!!",
    "tags": "[\"Web\"]",
    "Hidden": false,
    "uid": 1,
    "codeTemplates": [
        {
            "cid": 1,
            "title": "my python code",
            "explanation": "this is my code :)",
            "language": "Python",
            "tags": "[\"Python\"]",
            "code": "for i in range(5): print(i * i)",
            "uid": 1,
            "isForked": false,
            "ogTemplateId": null
        }
    ]
}
```

Responses

- **200 OK**: Blog updated successfully.
- **400 Bad Request**: Blog is hidden and cannot be edited.
- **401 Unauthorized**: Authentication token is missing or invalid.
- **403 Forbidden**: Only the owner of the blog can update it.
- **404 Not Found**: Blog not found.

---

4. Get a Single Blog

- **Endpoint**: `/api/Blogs/{id}`
- **Method**: `GET`
- **Description**: Retrieves a single blog by its ID.

Example Request

```
GET /api/Blogs/3
```

Example Response (200 - OK)

```
{
    "bid": 3,
    "title": "Blog Title!",
    "description": "updated description of my blog!!",
    "tags": "[\"python\"]",
    "Hidden": false,
    "uid": 1
}
```

Responses

- **200 OK**: Returns the blog data if found.
- **404 Not Found**: Blog not found.
- **500 Internal Server Error**: Something went wrong during retrieval.

---

5. Delete Blog

- **Endpoint**: `/api/Blogs/{id}`
- **Method**: `DELETE`
- **Description**: Allows an authenticated user to delete a blog. Only the blog owner can delete it.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies or in header as bearer token.
- Users must have the `USER` role to delete a blog.

Example Request

```
DELETE /api/Blogs/3
Cookie: accessToken=<valid_token>
```

Example Response (200 - OK)

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

Responses

- **200 OK**: Blog deleted successfully.
- **401 Unauthorized**: Authentication token is missing or invalid.
- **403 Forbidden**: Only the owner of the blog can delete it.
- **404 Not Found**: Blog not found.

Error Responses

- **401 Unauthorized**: Returned when authentication is required but no valid token is provided.
- **403 Forbidden**: Returned when the user lacks permission (not the owner or incorrect role).
- **404 Not Found**: Returned when the specified blog ID does not exist.
- **500 Internal Server Error**: General error during processing.

---

6. Get Blogs Sorted By Rating

- **Endpoint**: `/api/Blogs`
- **Method**: `GET`
- **Description**: Retrieves a paginated list of blogs with their associated ratings and calculates a score based on upvotes and downvotes.

Query Parameters

Parameter	Type	Required	Default	Description
`page`	`number`	No	`1`	The page number to retrieve (1-based index).
`limit`	`number`	No	`10`	The number of blogs to return per page.

Example Request

```
GET /api/Blogs/sortByRatings
```

Example Response (200 - OK)

```
[
    {
        "bid": 1,
        "title": "my blog",
        "description": "this is my first blog :)",
        "tags": "[\"Python\"]",
        "Hidden": false,
        "uid": 1,
        "ratings": [
            {
                "rateId": 1,
                "upvote": true,
                "downvote": false,
                "uid": 1,
                "bid": 1,
                "commentId": null,
                "replyId": null
            }
        ],
        "score": 1
    },
    {
        "bid": 2,
        "title": "my blog",
        "description": "this is my first blog :)",
        "tags": "[\"Python\"]",
        "Hidden": false,
        "uid": 1,
        "ratings": [],
        "score": 0
    }
]
```


Responses

- **200 OK**: Returns a list of blogs for the specified page, including their score based on ratings.
- **500 Internal Server Error**: Returns an error message if something goes wrong during the retrieval process.
- **405 Method Not Allowed**: Returned when a method other than `GET` is requested.

---

7. Get Filtered Blogs Sorted by Number of Reports for Admin

- **Endpoint**: `/api/Blogs/sortByReports`
- **Method**: `GET`
- **Description**: Retrieves a list of blogs filtered by various parameters and sorted by the number of reports, accessible only to users with the `ADMIN` role.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>` (Cookies also accepted)

Query Parameters

Parameter	Type	Required	Description
`bid`	`number`	No	The unique identifier of the blog to retrieve.
`title`	`string`	No	A string to filter blogs by title (partial match).
`description`	`string`	No	A string to filter blogs by description (partial match).
`tags`	`string`	No	A JSON string of tags to filter blogs (can handle single tags).
`uid`	`number`	No	The unique identifier of the user who authored the blogs.
`codeTemplateIds`	`string`	No	A JSON string of code template IDs to filter blogs.
`page`	`number`	No	The page number for pagination (default is `1`).
`limit`	`number`	No	The number of blogs to return per page (default is `10`).

Example Request

```
GET /api/Blogs/sortByReports
Cookie: accessToken=<valid_token>
```

Example Response

```
[
    {
        "bid": 1,
        "title": "my blog",
        "description": "this is my first blog :)",
        "tags": "[\"Python\"]",
        "Hidden": false,
        "uid": 1,
        "reports": [
            {
                "reportId": 1,
                "uid": 1,
                "bid": 1,
```

```
                "commentId": null,
                "replyId": null,
                "explanation": "this blog post has inappropriate content."
            },
            {
                "reportId": 2,
                "uid": 1,
                "bid": 1,
                "commentId": null,
                "replyId": null,
                "explanation": "this blog post has inappropriate content."
            }
        ]
    },
    {
        "bid": 2,
        "title": "my blog",
        "description": "this is my first blog :)",
        "tags": "[\"Python\"]",
        "Hidden": false,
        "uid": 1,
        "reports": []
    }
]
```


---

8. Rate a Blog

- **Endpoint**: `/api/Blogs/[id]/rate`
- **Method**: `POST`
- **Description**: Allows a user to upvote or downvote a blog post. Only authenticated users with the `USER` role can rate a blog.

Request Parameters

- **Path Parameters**:
  - `id` (integer): The unique identifier of the blog to be rated.
- **Body Parameters**:
  - `upvote` (boolean): Set to `true` if the user wants to upvote, otherwise `false`.
  - `downvote` (boolean): Set to `true` if the user wants to downvote, otherwise `false`.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies.
- Users must have the `USER` role to rate a blog.

Example Request

```
POST /api/Blogs/1/rate
Cookie: accessToken=<valid_token>

{
    "upvote": true,
    "downvote": false
}
```

Example Response

```
{
    "message": "Rating updated successfully",
    "rating": {
        "rateId": 1,
        "upvote": true,
        "downvote": false,
        "uid": 1,
        "bid": 1,
        "commentId": null,
        "replyId": null
    }
}
```

Responses

- **200 OK**: Returns a success message and the updated or created rating object.
- **401 Unauthorized**: Returned if the authentication token is missing.
- **403 Forbidden**: Returned if the user does not exist, does not have the `USER` role, or if the token is invalid/expired.
- **404 Not Found**: Returned if the specified blog does not exist.
- **500 Internal Server Error**: Returned if an unexpected error occurs during rating.

Code Template Endpoints

Base URL for Blogs:

```
http://localhost:3000/api/CodeTemplates
```

1. Create a New Code Template

- **Endpoint**: `/api/CodeTemplates`
- **Method**: `POST`

- **Description**: Creates a new Code Template. Only authenticated users with the role `"USER"` are authorized to create a blog.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>`

Request Body

Field	Type	Description
`title`	`string`	Title of the blog post.
`description`	`string`	Description of the blog post.
`tags`	`string`	JSON array of tags for the blog.
`codeTemplateIds`	`array` of `number`	Array of template IDs associated with the blog.

Request Body Parameters

- `title` (string, required): The title of the code template.
- `explanation` (string, optional): An explanation or description of the code template.
- `language` (string, required): The programming language of the code.
- `tags` (array of strings, required): Tags associated with the code template.
- `code` (string, required): The actual code content.

Example Request

```
POST /api/codeTemplates
Cookie: accessToken=<valid_token>

{
    "title": "Example Template",
    "explanation": "This is an example explanation",
    "language": "JavaScript",
    "tags": ["example", "template"],
    "code": "console.log('Hello, World!');"
}
```

Example Response

```
{
    "cid": 1,
    "title": "Example Template",
    "explanation": "This is an example explanation",
    "language": "JavaScript",
    "tags": ["example", "template"],
```

```
    "code": "console.log('Hello, World!');",
    "uid": 5,
    "isForked": false,
    "ogTemplateId": null
}
```

Responses

- **201 Created**: Returns the created code template object.
- **400 Bad Request**: Returned if required fields are missing in the request body.
- **401 Unauthorized**: Returned if the access token is missing.
- **403 Forbidden**: Returned if the user's role is not `USER` or if the token is invalid/expired.
- **500 Internal Server Error**: Returned if there is an error while saving the code template.

---

2. Get Code Template

- **Endpoint**: `/api/CodeTemplates`
- **Method**: `GET`
- **Description**: Retrieves a list of Code Templates with optional filters, pagination, and visibility settings based on user authentication and role.

Query Parameters

Parameter	Type	Required	Description
`cid`	`integer`	No	Filter by code template ID.
`title`	`string`	No	Filter by title containing this string.
`language`	`string`	No	Filter by programming language.
`tags`	`string[]`	No	Filter by tags associated with the template.
`code`	`string`	No	Filter by code containing this string.
`uid`	`integer`	No	Filter by user ID.
`page`	`integer`	No	Page number for pagination (default: 1).
`limit`	`integer`	No	Number of items per page (default: 10).

Example Request

```
GET /api/codeTemplates?page=1&limit=5&language=Python
```

Example Response (200 - OK)

```
[
    {
        "cid": 1,
        "title": "my python code",
        "explanation": "this is my code :)",
        "language": "Python",
        "tags": "[\"Python\"]",
        "code": "for i in range(5): print(i * i)",
        "uid": 1,
        "isForked": false,
        "ogTemplateId": null
    }
]
```

Response Codes

- **200 OK**: Code Templates retrieved successfully.
- **400 Bad Request**: Invalid query parameter format (e.g., non-JSON tags or codeTemplateIds).
- **500 Internal Server Error**: Error occurred while retrieving Code Templates.

---

3. Update Code Templates

- **Endpoint**: `/api/CodeTemplates/{id}`
- **Method**: `PUT`
- **Description**: Allows an authenticated user to update a Code Template's details. Only the Code Template owner can make updates.

Authentication

- Requires a valid authentication token (`accessToken`) stored in cookies or in the header as a bearer token.
- Users must have the `USER` role to update a Code Template.

Request Body

Parameter	Type	Required	Description
`title`	`string`	No	The updated title of the code template.
`explanation`	`string`	No	The updated explanation of the code template.
`language`	`string`	No	The updated programming language of the template.

`tags`	`string[]`	No	An array of tags associated with the code template.
`code`	`string`	No	The updated code in the template.

Example Request 200-OK

```
PUT /api/CodeTemplates/1
Cookie: accessToken=<your_access_token>

{
    "title": "Updated code template!!",
    "explanation": "i just updated my explanation of my code template!",
    "code": "#include <stdio.h>\nint main() { printf(\"Hello from C\\n\"); retur
    "tags": "[\"Python\"]",
    "language": "C"
}
```

Example Response

```
{
    "cid": 1,
    "title": "Updated code template!!",
    "explanation": "i just updated my explanation of my code template!",
    "language": "C",
    "tags": "[\"Python\"]",
    "code": "#include <stdio.h>\nint main() { printf(\"Hello from C\\n\"); retur
    "uid": 1,
    "isForked": false,
    "ogTemplateId": null
}
```

Responses

- **200 OK**: Code Template updated successfully.
- **400 Bad Request**: Invalid input data.
- **401 Unauthorized**: Authentication token is missing or invalid.
- **403 Forbidden**: Only the owner of the code template can update it.
- **404 Not Found:** Code Template not found.

---

4. Retrieve a Code Template

- **Endpoint**: `/api/CodeTemplates/{id}`
- **Method**: `GET`
- **Description**: Retrieves the details of a specific Code Template by its ID, including associated blog posts.

Authentication

- No authentication required.

Query Parameters

Parameter	Type	Required	Description
`id`	`integer`	Yes	The ID of the Code Template to retrieve.

Example Request

```
GET /api/CodeTemplates/1
```

Example Response


```
{
    "cid": 1,
    "title": "Updated code template!!",
    "explanation": "i just updated my explanation of my code template!",
    "language": "C",
    "tags": "[\"Python\"]",
    "code": "#include <stdio.h>\nint main() { printf(\"Hello from C\\n\"); retur
    "uid": 1,
    "isForked": false,
    "ogTemplateId": null,
    "blogs": []
}
```


---

5. Delete a Code Template

- **Endpoint**: `/api/CodeTemplates/{id}`
- **Method**: `DELETE`
- **Description**: Allows an authenticated user to delete a Code Template by its ID. Only the owner of the Code Template can perform this action.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies or in the header as a bearer token.
- Users must have the `USER` role to delete a Code Template.

Query Parameters

Parameter	Type	Required	Description
`id`	`integer`	Yes	The ID of the Code Template to delete.

Example Request

```
DELETE /api/code-templates/1
Authorization: Bearer <JWT_TOKEN>
```

Example Response

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

Reponses

- **200 OK**: Code Template deleted successfully.
- **401 Unauthorized**: Authentication token is missing or invalid.
- **403 Forbidden**: Only the owner of the Code Template can delete it.
- **404 Not Found**: Code Template not found.
- **500 Internal Server Error**: Something went wrong while trying to delete the Code Template.

---

6. Get Current User's Code Templates

- **Endpoint**: `/api/code-templates`
- **Method**: `GET`
- **Description**: Allows an authenticated user to retrieve their own Code Templates, with optional filtering by various parameters.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies or in the header as a bearer token.
- Users must have the `USER` role to access their Code Templates.

Query Parameters

Parameter	Type	Required	Description
`id`	`integer`	Optional	The ID of the Code Template to filter by.
`title`	`string`	Optional	The title to filter Code Templates by, searching for templates that contain this title.

`language`	`string`	Optional	The programming language to filter Code Templates by, searching for templates that contain this language.
`tags`	`string`	Optional	Tags to filter Code Templates by, searching for templates that contain these tags.
`code`	`string`	Optional	Code content to filter Code Templates by, searching for templates that contain this code.
`page`	`integer`	Optional	The page number for pagination (default is 1).
`limit`	`integer`	Optional	The number of Code Templates to retrieve per page (default is 10).

Example Request

```
GET /api/code-templates?page=1&limit=10&title=python
Authorization: Bearer <JWT_TOKEN>
```

Example Reponse


```
[
    {
        "cid": 1,
        "title": "Updated code template!!",
        "explanation": "i just updated my explanation of my code template!",
        "language": "C",
        "tags": "[\"Python\"]",
        "code": "#include <stdio.h>\nint main() { printf(\"Hello from C\\n\"); re
        "uid": 1,
        "isForked": false,
        "ogTemplateId": null
    }
]
```


Responses

- **200 OK**: Code Templates retrieved successfully.
- **401 Unauthorized**: Authentication token is missing or invalid.
- **403 Forbidden**: Only users can access their own Code Templates.
- **500 Internal Server Error**: Something went wrong while trying to retrieve the Code Templates.
- **405 Method Not Allowed**: The requested method is not supported for this endpoint.

---

7. Fork a Code Template

- **Endpoint**: `/api/code-templates`
- **Method**: `POST`
- **Description**: Allows an authenticated user to create a fork of an existing Code Template. This is useful for modifying and saving templates while preserving the original.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies or in the header as a bearer token.
- Users must have the `USER` role to save templates.

Request Body

Parameter	Type	Required	Description
`cid`	`number`	Yes	The ID of the Code Template to be forked.

Example Request

```
POST /api/code-templates
Authorization: Bearer <JWT_TOKEN>

{
  "cid": 1
}
```

Example Response

```
{
    "message": "Template saved as a fork!",
    "template": {
        "cid": 2,
        "title": "Updated code template!!",
        "explanation": "i just updated my explanation of my code template!",
        "language": "C",
        "tags": "[\"Python\"]",
        "code": "#include <stdio.h>\nint main() { printf(\"Hello from C\\n\"); re
        "uid": 1,
        "isForked": true,
        "ogTemplateId": 1
    }
}
```

Responses

- **201 Created**: Forked Code Template created successfully.
- **401 Unauthorized**: Authentication token is missing or invalid.

- **403 Forbidden**: Invalid or Expired token.
- **500 Internal Server Error**: Something went wrong while trying to fork the Code Template.
- **405 Method Not Allowed**: The requested method is not supported for this endpoint.

Code Execution Endpoints

Base URL for Code Execution:

```
http://localhost:3000/api/execution/executionCode
```

1. Execute Code

- **Endpoint**: `/api/execution/executeCode`
- **Method**: `POST`
- **Description**: Executes code in the specified programming language. Users provide the code they intend to run. Standard input (stdin) can also be specified.

Request Body

Field	Type	Description
`code`	`string`	Code to execute
`language`	`string`	Programming language of the code (`python` , `java` , `c` , `cpp` , `javascript`).
`stdinInput`	`string`	Optional. Standard input for the code execution.

Example Request: Execute Python Code

```
POST /api/execution/executeCode
Content-Type: application/json

{
  "code": "name = input('Enter your name: ')\nprint(f'Hello, {name}!')",
  "langauge": "python",
  "stdinInput": "Suhani"
}
```

Example Response

```
{
    "output": "Enter your name: Hello, Suhani!\n"
```

```
}
```

Replies Endpoints

Base URL for Comments:

```
http://localhost:3000/api/replies
```

1. Post a Reply to a Comment

- **Endpoint**: `/api/replies`
- **Method**: `POST`
- **Description**: Allows a user to post a reply to a specific comment.

Authentication

- Requires a valid authentication token (accessToken) stored in cookies.
- Users must have the `USER` role to post replies.

Request Body

Parameter	Type	Required	Description
`commentId`	`number`	Yes	The ID of the comment to which the reply is made.
`content`	`string`	Yes	The content of the reply being posted.

Example Request

```
POST /api/replies
Authorization: Bearer <JWT_TOKEN>

{
    "commentId": "1",
    "content": "I don't agree with your comment."
}
```

Example Reponse

```
{
    "replyId": 1,
    "ownerId": 1,
    "commentId": 1,
    "replierId": 1,
    "content": "I don't agree with your comment.",
```

```
    "Hidden": false
}
```

Reponses

- **201 Created**: Reply successfully created.
- **400 Bad Request**: Missing required fields in the request body.
- **401 Unauthorized**: Authentication token is missing or invalid.
- **403 Forbidden**: Only users can post replies.
- **404 Not Found**: The specified comment does not exist.
- **500 Internal Server Error**: An error occurred while processing the request.
- **405 Method Not Allowed**: The requested method is not supported for this endpoint.

---

2. Get Replies for a Comment

- **Endpoint**: `/api/replies`
- **Method**: `GET`
- **Description**: Retrieves replies associated with a specific comment, with support for filtering and pagination.

Authentication

An authentication token (`accessToken`) is optional. Visitors can view unhidden replies.

Query Parameters

Parameter	Type	Required	Description
`ownerId`	`number`	No	Filter replies by the ID of the owner of the reply.
`replierId`	`number`	No	Filter replies by the ID of the user who replied.
`commentId`	`number`	No	Filter replies by the ID of the comment.
`page`	`number`	No	The page number for pagination (default is 1).
`limit`	`number`	No	The number of replies to return per page (default is 10).

Example Request

```
GET /api/replies
Authorization: Bearer <JWT_TOKEN>
```

Example Response

```
{
    "replies": [
        {
            "replyId": 1,
            "ownerId": 1,
            "commentId": 1,
            "replierId": 1,
            "content": "I don't agree with your comment.",
            "Hidden": false,
            "owner": {
                "uid": 1,
                "username": "user1",
                "email": "user1@mail.com",
                "password": "$2b$10$nryoR5t5Obv2QF8EEP6g3uTxwyGju4LfrAdxGPXR/TI.
            },
            "replier": {
                "uid": 1,
                "username": "user1",
                "email": "user1@mail.com",
                "password": "$2b$10$nryoR5t5Obv2QF8EEP6g3uTxwyGju4LfrAdxGPXR/TI.
            }
        }
    ],
    "totalReplies": 1,
    "currentPage": 1,
    "totalPages": 1
}
```


Responses

- **200 OK**: Returns a list of replies for the specified comment.
- **400 Bad Request** Invalid query parameters.
- **401 Unauthorized**: Authentication token is invalid.
- **404 Not Found**: The specified comment does not exist.
- **500 Internal Server Error**: An error occurred while processing the request.
- **405 Method Not Allowed**: The requested method is not supported for this endpoint.

---

3. Delete a Reply

- **Endpoint**: `/api/replies`
- **Method**: `DELETE`
- **Description**: Deletes a specific reply based on its ID.

Authentication

An authentication token (`accessToken`) is required. Only users can delete replies.

Path Parameters

Parameter	Type	Required	Description
`id`	`number`	Yes	The ID of the reply to be deleted.

Example Request

```
DELETE /api/replies/1
Authorization: Bearer <JWT_TOKEN>
```

Example Response

```
{
    "message": "Reply deleted successfully."
}
```

Responses

- **200 OK**: Successfully deleted the reply.
- **401 Unauthorized**: Authentication token is required.
- **403 Forbidden**:
  - "Forbidden: Only users can delete replies." (User role is not "USER")
  - "You can't delete someone else's reply." (User is not the replier)
- **404 Not Found**: The specified reply does not exist.
- **500 Internal Server Error**: Failed to delete the reply.

---

4. Update a Reply

- **Endpoint**: `/api/replies`
- **Method**: `PUT`
- **Description**: Updates the content of a specific reply based on its ID.

Authentication

An authentication token (`accessToken`) is required. Only users can edit replies.

Path Parameters

Parameter	Type	Required	Description
`id`	`number`	Yes	The ID of the reply to be updated.

Request Body

Parameter	Type	Required	Description
`newReplyContent`	`string`	Yes	The new content for the reply.

Example Request

```
PUT /api/replies/1
Authorization: Bearer <JWT_TOKEN>
Content-Type: application/json

{
    "newReplyContent": "This is the updated reply content."
}
```

Example Response (200-OK)

```
{
    "message": "Reply updated successfully.",
    "updatedReply": {
        "replyId": 1,
        "ownerId": 1,
        "commentId": 1,
        "replierId": 1,
        "content": "This is the updated reply content.",
        "Hidden": false
    }
}
```

---

5. Get Replies Sorted by Ratings

- **Endpoint**: `/api/replies/sortByRatings`
- **Method**: `GET`
- **Description**: Retrieves replies associated with a specific comment, sorted by ratings.

Query Parameters

Parameter	Type	Required	Description
`commentId`	`number`	Yes	The ID of the comment for which replies are retrieved.
`page`	`number`	No	The page number for pagination (default is 1).
`limit`	`number`	No	The number of replies to return per page (default is 10).

Example Request

```
GET /api/replies/sortByRatings?commentId=1&page=1&limit=10
```

Example Response

```
[
    {
        "replyId": 1,
        "ownerId": 1,
        "commentId": 1,
        "replierId": 1,
        "content": "I agree with your comment",
        "Hidden": false,
        "ratings": [],
        "score": 0
    }
]
```

Responses

- **200 OK**: Returns a list of replies sorted by their score (upvotes - downvotes) for the specified comment.
- **405 Method Not Allowed**: The specified method is not allowed for this endpoint.
- **400 Bad Request**: "Please provide commentId" if commentId is not provided.
- **500 Internal Server Error**: "Failed to filter replies" if an error occurs during processing.

---

6. Get Replies Sorted by Number of Reports

- **Endpoint**: `/api/replies/sortByReports`
- **Method**: `GET`
- **Description**: Retrieves replies associated with optional filters, sorted by the number of reports.

Query Parameters

Parameter	Type	Required	Description
`ownerId`	`number`	No	Filter replies by the ID of the owner of the reply.
`replierId`	`number`	No	Filter replies by the ID of the user who replied.
`commentId`	`number`	No	Filter replies by the ID of the comment.
`page`	`number`	No	The page number for pagination (default is 1).
`limit`	`number`	No	The number of replies to return per page (default is 10).

Authentication

- **Token**: An authentication token (`accessToken`) is required. Only users with the ADMIN role can access this endpoint.

Example Request

```
GET /api/replies/sortByReports
```

Example Response (200-OK)

```
{
    "replies": [
        {
            "replyId": 1,
            "ownerId": 1,
            "commentId": 1,
            "replierId": 1,
            "content": "I agree with your comment",
            "Hidden": false,
            "reports": [
                {
                    "reportId": 3,
                    "uid": 1,
                    "bid": null,
                    "commentId": null,
                    "replyId": 1,
                    "explanation": "this blog post has inappropriate content."
                }
            ]
        }
    ],
    "totalReplies": 1,
    "currentPage": 1,
    "totalPages": 1
}
```

Responses

- **200 OK**: Returns a list of replies sorted by the number of reports.
- **401 Unauthorized**: "Authentication token is required" if no token is provided.
- **403 Forbidden**: "Forbidden: Only admins can hide content" if the user does not have the ADMIN role.
- **405 Method Not Allowed**: The specified method is not allowed for this endpoint.
- **500 Internal Server Error**: "Something went wrong." if an error occurs during processing.

---

7. Rate Replies

- **Endpoint**: `/api/replies/[id]/rate`
- **Method**: `POST`
- **Description**: Allows users to rate a reply by providing an upvote or downvote.

URL Parameters

Parameter	Type	Required	Description
`id`	`number`	Yes	The ID of the reply to be rated.

Request Body

Parameter	Type	Required	Description
`upvote`	`boolean`	No	Indicates if the reply is upvoted (true for upvote).
`downvote`	`boolean`	No	Indicates if the reply is downvoted (true for downvote).

Authentication

- **Token**: An authentication token (`accessToken`) is required. Only users with the USER role can access this endpoint.

Example Request

```
POST /api/replies/1/rate
Content-Type: application/json

{
    "upvote": true,
    "downvote": false
}
```

Example Reponse (200-OK)

```
{
    "message": "Rating updated successfully",
    "rating": {
        "rateId": 2,
        "upvote": true,
        "downvote": false,
        "uid": 1,
        "bid": null,
        "commentId": null,
        "replyId": 1
    }
}
```

Responses

- **200 OK**: Returns a success message and the updated rating object.
- **401 Unauthorized**: "Authentication token is required" if no token is provided.
- **403 Forbidden**: "Forbidden: Only users can rate replies" if the user is not found or does not have the USER role.
- **404 Not Found**: "Reply not found" if the specified reply ID does not exist.
- **405 Method Not Allowed**: The specified method is not allowed for this endpoint.
- **500 Internal Server Error**: Returns an error message if an error occurs during processing.

Report Endpoints

Base URL for Comments:

```
http://localhost:3000/api/Reports
```

1. Create Report

- **Endpoint**: `/api/reports`
- **Method**: `POST`
- **Description**: Allows users to create a report for a blog, comment, or reply.

Request Body

Parameter	Type	Required	Description
`bid`	`number`	No	The ID of the blog to report.
`commentId`	`number`	No	The ID of the comment to report.
`replyId`	`number`	No	The ID of the reply to report.
`explanation`	`string`	Yes	Explanation for the report.

Authentication

- **Token**: An authentication token (`accessToken`) is required. Only users with the USER role can access this endpoint.

Example Request

```
POST /api/Reports
Content-Type: application/json
```

```
{
    "bid": 1,
    "explanation": "Inappropriate content"
}
```

Example Response

```
{
    "reportId": 4,
    "uid": 1,
    "bid": 1,
    "commentId": null,
    "replyId": null,
    "explanation": "Inappropriate content"
}
```

Responses

- **201 Created**: Returns the created report object.
- **400 Bad Request**: "Either bid or commentId or replyId are required" if none are provided, or "You can't report a blog and comment and reply all at once."
- **401 Unauthorized**: "Authentication token is required" if no token is provided.
- **403 Forbidden**: "Forbidden: Only users can create reports" if the user does not have the USER role.
- **404 Not Found**: "Blog not found", "comment not found", or "reply not found" if the specified item does not exist.
- **500 Internal Server Error**: Returns an error message if an error occurs during processing.

---

2. Get Reports

- **Endpoint**: `/api/Reports`
- **Method**: `GET`
- **Description**: Allows admins to view reports with optional filtering.

Query Parameters

Parameter	Type	Required	Description
`reportId`	`number`	No	Filter reports by report ID.
`uid`	`number`	No	Filter reports by user ID.
`bid`	`number`	No	Filter reports by blog ID.

`commentId`	`number`	No	Filter reports by comment ID.
`replyId`	`number`	No	Filter reports by reply ID.
`page`	`number`	No	The page number for pagination (default: 1).
`limit`	`number`	No	The number of reports to return per page (default: 10).

Authentication

- **Token**: An authentication token (`accessToken`) is required. Only users with the ADMIN role can access this endpoint.

Example Request

```
GET /api/Reports
```

Example Response

```
[
    {
        "reportId": 1,
        "uid": 1,
        "bid": 1,
        "commentId": null,
        "replyId": null,
        "explanation": "this blog post has inappropriate content."
    }
]
```

Responses

- **200 OK**: Returns a list of reports matching the provided filters.
- **401 Unauthorized**: "Authentication token is required" if no token is provided.
- **403 Forbidden**: "Forbidden: Only admins can view reports" if the user does not have the ADMIN role.
- **500 Internal Server Error**: Returns an error message if an error occurs during processing.

---

3. Edit Report

Edit Report

- **Endpoint**: `/api/reports/:id`
- **Method**: `PUT`

- **Description**: Allows users to edit an existing report.

Request Body

Parameter	Type	Required	Description
`explanation`	`string`	Yes	The new explanation for the report.

Authentication

- **Token**: An authentication token (`accessToken`) is required. Only users with the USER role can edit reports.

Example Request

```
PUT /api/reports/1
Content-Type: application/json

{
    "explanation": "Updated explanation for the report."
}
```

Example Response

```
{
    "reportId": 1,
    "uid": 1,
    "bid": 1,
    "commentId": null,
    "replyId": null,
    "explanation": "I have updated my report"
}
```

Responses

- **200 OK**: Returns the requested report.
- **404 Not Found**: "Report not found" if the specified report does not exist.
- **500 Internal Server Error**: "Something went wrong." if an error occurs during processing.

---

4. Delete Report

- **Endpoint**: `/api/reports/:id`
- **Method**: `DELETE`
- **Description**: Allows users to delete an existing report.

Authentication

- **Token**: An authentication token (`accessToken`) is required. Only users with the USER role can delete reports.

Example Request

```
DELETE /api/reports/1
```

Example Response (200 - OK)

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

Responses

- **200 OK**:Returns a message confirming successful deletion.
- **401 Unauthorized**: "Authentication token is required" if no token is provided.
- **403 Forbidden**:
  - "Forbidden: Only users can delete reports" if the user does not have the USER role.
  - "You can't delete someone else's report" if the user is not the owner of the report.
- **404 Not Found**: "Report not found" if the specified report does not exist.
- **500 Internal Server Error**: Returns an error message if an error occurs during processing.

Admin Endpoints

Base URL for Admin Operations:

```
http://localhost:3000/api/admin
```

1. Admin Login

- **Endpoint**: `/api/admin/login`
- **Method**: `POST`
- **Description**: Authenticates an admin user by verifying their credentials and generates an access token and a refresh token if successful.

Request Body

Field	Type	Description

`email`	`string`	Email of the admin attempting to log in.
`password`	`string`	Password of the admin.

Example Request

```
POST /api/admin/login

{
  "email": "admin@example.com",
  "password": "adminpassword"
}
```

Example Response: 200 - OK

```
{
  "message": "Admin login successful",
  "accessToken": "<ACCESS_TOKEN>",
  "refreshToken": "<REFRESH_TOKEN>"
}
```

Cookies Set

- **`accessToken`**: A non-HTTP-only cookie with a 15-minute expiration, storing the access token for frontend access.
- **`refreshToken`**: An HTTP-only cookie with a 7-day expiration, storing the refresh token for backend use.

Response Codes

- **200 OK**: Admin successfully logged in, access and refresh tokens generated.
- **400 Bad Request**: Incorrect email or password.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred during the login process.

---

2. Admin Logout

- **Endpoint**: `/api/admin/logout`
- **Method**: `POST`
- **Description**: Logs out the admin user by clearing the access and refresh token cookies.

Example Request

```
POST /api/admin/logout
```

Example Response: 200 - OK

```
{
  "message": "Admin logged out successfully"
}
```

Cookies Cleared

- `accessToken`: HTTP-only cookie with immediate expiration to clear the access token.
- `refreshToken`: HTTP-only cookie with immediate expiration to clear the refresh token.

Response Codes

- **200 OK**: Admin successfully logged out and cookies are cleared.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred during the logout process.

---

3. Hide Content

- **Endpoint**: `/api/admin/hideContent`
- **Method**: `PUT`
- **Description**: Hides specified content (blog, comment, or reply) by marking it as hidden. Only admins are authorized to perform this action.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>`

Request Body

Field	Type	Description
`contentId`	`number`	ID of the content to be hidden.
`type`	`string`	Type of content to hide (`blog`, `comment`, or `reply`).

Example Request

```
PUT /api/admin/hideContent
Authorization: Bearer <JWT_TOKEN>

{
  "contentId": 123,
```

```
  "type": "blog"
}
```

Example Response: 200 - OK

```
{
  "message": "Blog content marked as hidden",
  "result": { /* Hidden content details */ }
}
```

Response Codes

- **200 OK**: Content marked as hidden successfully.
- **400 Bad Request**: Invalid content type or ID provided.
- **401 Unauthorized**: Authentication token is missing.
- **403 Forbidden**: Invalid token or user is not an admin.
- **405 Method Not Allowed**: HTTP method is not `PUT`.
- **500 Internal Server Error**: Error occurred while hiding content.

---

4. Register Admin with Email Approval

- **Endpoint**: `/api/admin/register_email_approval`
- **Method**: `POST`
- **Description**: Initiates the admin registration process by saving the request in the database and sending a confirmation email to the manager for approval.

Request Body

Field	Type	Description
`email`	`string`	Email of the admin requesting registration.
`password`	`string`	Password for the admin account.

Example Request

```
POST /api/admin/register_email_approval

{
  "email": "newadmin@example.com",
  "password": "securepassword"
}
```

Example Response: 200 - OK

```
{
  "message": "Confirmation email sent to manager for approval"
}
```

Response Codes

- **200 OK**: Registration request saved and email sent successfully.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred while processing registration or sending email.

---

5. Finalize Admin Registration

- **Endpoint**: `/api/admin/register_final_process`
- **Method**: `GET`
- **Description**: Completes the admin registration process by verifying the confirmation token and creating the admin account.

Query Parameters

Parameter	Type	Description
`token`	`string`	Confirmation token provided in the approval email.

Example Request

```
GET /api/admin/register_final_process?token=<CONFIRMATION_TOKEN>
```

Example Response: 200 - OK

```
{
  "message": "Admin successfully registered",
  "admin": {
    "email": "admin@example.com",
    "id": 1
  }
}
```

Response Codes

- **200 OK**: Admin successfully registered and confirmation token is valid.
- **400 Bad Request**: Invalid or expired confirmation token.
- **405 Method Not Allowed**: HTTP method is not `GET`.
- **500 Internal Server Error**: Error occurred while confirming admin registration.

Authentication Endpoints

Base URL for Authentication Operations:

```
http://localhost:3000/api/auth
```

1. Refresh Access Token

- **Endpoint**: `/api/auth/refresh`
- **Method**: `POST`
- **Description**: Generates a new access token using a valid refresh token from cookies. Only valid users or admins can refresh their access tokens.

Cookies Required

- `refreshToken`: HTTP-only cookie containing the refresh token.

Example Request

```
POST /api/auth/refresh
```

Example Response: 200 - OK

```
{
  "accessToken": "<NEW_ACCESS_TOKEN>"
}
```

Response Codes

- **200 OK**: New access token generated and set in cookies.
- **401 Unauthorized**: Refresh token is missing.
- **403 Forbidden**: Invalid or expired refresh token.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred while refreshing the token.

User Endpoints

Base URL for User Operations:

```
http://localhost:3000/api/users
```

1. User Login

- **Endpoint**: `/api/users/login`
- **Method**: `POST`
- **Description**: Authenticates a user by verifying their credentials and generates both an access token and a refresh token upon successful login.

Request Body

Field	Type	Description
`username`	`string`	Username of the user attempting to log in.
`password`	`string`	Password of the user.

Example Request

```
POST /api/users/login

{
  "username": "exampleUser",
  "password": "userpassword"
}
```

Example Response: 200 - OK

```
{
  "message": "Login successful",
  "accessToken": "<ACCESS_TOKEN>",
  "refreshToken": "<REFRESH_TOKEN>"
}
```

Cookies Set

- **`accessToken`**: A non-HTTP-only cookie with a 15-minute expiration, storing the access token for frontend access.
- **`refreshToken`**: An HTTP-only cookie with a 7-day expiration, storing the refresh token for backend use.

Response Codes

- **200 OK**: User successfully logged in, access and refresh tokens generated.
- **400 Bad Request**: Incorrect username or password.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred during the login process.

---

2. User Logout

- **Endpoint**: `/api/users/logout`
- **Method**: `POST`
- **Description**: Logs out the user by clearing the access and refresh token cookies.

Example Request

```
POST /api/users/logout
```

Example Response: 200 - OK

```
{
  "message": "User logged out successfully"
}
```

Cookies Cleared

- **`accessToken`**: HTTP-only cookie with immediate expiration to clear the access token.
- **`refreshToken`**: HTTP-only cookie with immediate expiration to clear the refresh token.

Response Codes

- **200 OK**: User successfully logged out, cookies cleared.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred during the logout process.

---

3. User Registration

- **Endpoint**: `/api/users/register`
- **Method**: `POST`
- **Description**: Registers a new user with required details and creates an associated profile.

Request Body

Field	Type	Description
`username`	`string`	Username for the new user.
`email`	`string`	Email for the new user.
`password`	`string`	Password for the new user.
`firstName`	`string`	(Optional) First name of the user.

`lastName`	`string`	(Optional) Last name of the user.
`phoneNumber`	`string`	Phone number of the user.

Example Request

```
POST /api/users/register

{
  "username": "exampleUser",
  "email": "user@example.com",
  "password": "securepassword",
  "firstName": "John",
  "lastName": "Doe",
  "phoneNumber": "1234567890"
}
```

Example Response: 201 - Created

```
{
  "message": "User registered successfully",
  "user": {
    "username": "exampleUser",
    "email": "user@example.com",
    "profile": {
      "email": "user@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "avatar": "/avatars/avatar1.png",
      "phoneNumber": "1234567890"
    }
  }
}
```

Response Codes

- **201 Created**: User successfully registered.
- **400 Bad Request**: Missing required fields or username/email already exists.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred during the registration process.

---

4. Show User Profile

- **Endpoint**: `/api/users/showProfile`
- **Method**: `GET`
- **Description**: Retrieves the profile details of the authenticated user.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>` (Or `accessToken` cookie)

Example Request

```
GET /api/users/showProfile
Authorization: Bearer <JWT_TOKEN>
```

Example Response: 200 - OK

```
{
  "email": "user@example.com",
  "firstName": "John",
  "lastName": "Doe",
  "avatar": "/avatars/avatar1.png",
  "phoneNumber": "1234567890"
}
```

Response Codes

- **200 OK**: Profile successfully retrieved.
- **401 Unauthorized**: Authentication token is missing.
- **403 Forbidden**: Invalid or expired token.
- **404 Not Found**: Profile does not exist for the user.
- **405 Method Not Allowed**: HTTP method is not `GET`.
- **500 Internal Server Error**: Error occurred while fetching the profile.

---

5. Update User Avatar

- **Endpoint**: `/api/users/updateAvatar`
- **Method**: `PUT`
- **Description**: Updates the avatar of the authenticated user.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>` (Or `accessToken` cookie)

Request Body

Field	Type	Description
`avatar`	`string`	URL of the new avatar image.

Example Request

```
PUT /api/users/updateAvatar
Authorization: Bearer <JWT_TOKEN>

{
  "avatar": "/avatars/newAvatar.png"
}
```

Example Response: 200 - OK

```
{
  "message": "Profile photo updated successfully",
  "profile": {
    "uid": 1,
    "avatar": "/avatars/newAvatar.png"
  }
}
```

Response Codes

- **200 OK**: Avatar updated successfully.
- **400 Bad Request**: Avatar URL is missing.
- **401 Unauthorized**: Authentication token is missing.
- **403 Forbidden**: Invalid or expired token.
- **404 Not Found**: Profile does not exist for the user.
- **405 Method Not Allowed**: HTTP method is not `PUT`.
- **500 Internal Server Error**: Error occurred while updating the avatar.

---

6. Update User Profile

- **Endpoint**: `/api/users/updateProfile`
- **Method**: `PUT`
- **Description**: Updates the profile details (first name, last name, phone number) of the authenticated user.

Request Headers

- `Authorization` : `Bearer <JWT_TOKEN>` (Or `accessToken` cookie)

Request Body

Field	Type	Description
`firstName`	`string`	(Optional) Updated first name of the user.
`lastName`	`string`	(Optional) Updated last name of the user.

`phoneNumber`	`string`	(Optional) Updated phone number of the user.

Example Request

```
PUT /api/users/updateProfile
Authorization: Bearer <JWT_TOKEN>

{
  "firstName": "Jane",
  "lastName": "Smith",
  "phoneNumber": "0987654321"
}
```

Example Response: 200 - OK

```
{
  "message": "Profile updated successfully",
  "profile": {
    "uid": 1,
    "firstName": "Jane",
    "lastName": "Smith",
    "phoneNumber": "0987654321"
  }
}
```

Response Codes

- **200 OK**: Profile updated successfully.
- **401 Unauthorized**: Authentication token is missing.
- **403 Forbidden**: Invalid or expired token.
- **404 Not Found**: Profile does not exist for the user.
- **405 Method Not Allowed**: HTTP method is not `PUT`.
- **500 Internal Server Error**: Error occurred while updating the profile.

---

7. Update User Profile Photo

- **Endpoint**: `/api/users/updateProfilePhoto`
- **Method**: `POST`
- **Description**: Uploads and updates the profile photo for the authenticated user.

Request Headers

- `Authorization`: `Bearer <JWT_TOKEN>` (Or `accessToken` cookie)

Request Body

- `profilePhoto`: The profile photo file to upload. (Form-data key should be `profilePhoto`). File size is limited to 5MB.

Example Request

```
POST /api/users/updateProfilePhoto
Authorization: Bearer <JWT_TOKEN>
Content-Type: multipart/form-data

{
  "profilePhoto": <file>
}
```

Example Response: 200 - OK

```
{
  "message": "Profile photo uploaded successfully",
  "profilePhoto": "/uploads/uniqueProfilePhoto.png"
}
```

Response Codes

- **200 OK**: Profile photo uploaded and updated successfully.
- **401 Unauthorized**: Authentication token is missing.
- **403 Forbidden**: Invalid or expired token.
- **405 Method Not Allowed**: HTTP method is not `POST`.
- **500 Internal Server Error**: Error occurred while uploading the profile photo.

Comments Endpoints

Base URL for Comments:

```
http://localhost:3000/api/comments
```

1. Rate a Comment

- **Endpoint**: `/api/comments/{id}/rate`
- **Method**: `POST`
- **Description**: Allows authenticated users to upvote or downvote a comment. If a rating already exists for the user and the comment, it updates the rating; otherwise, it creates a new rating.

Request Headers

- `Cookie`: `accessToken=<TOKEN>`

Request Body

Field	Type	Description
`upvote`	`boolean`	Indicates whether the comment is upvoted.
`downvote`	`boolean`	Indicates whether the comment is downvoted.

Example Request

```
POST /api/1/rate
Cookie: accessToken=<TOKEN>

{
  "upvote": true,
  "downvote": false
}
```

Example Response: 200 - OK

```
{
    "message": "Rating updated successfully",
    "rating": {
        "rateId": 1,
        "upvote": true,
        "downvote": false,
        "uid": 1,
        "bid": null,
        "commentId": 2,
        "replyId": null
    }
}
```

Response Codes

- **200 OK**: Rating updated successfully.
- **401 Unauthorized**: Authentication token is required
- **403 Forbidden:**: Only users can rate blogs
- **404 Not Found**: Comment not found.
- **500 Internal Server Error**

2. Edit a Comment

- **Endpoint**: `/api/comments/{id}`
- **Method**: `PUT`
- **Description**: Allows an authenticated user to edit the content of their own comment.

Request Body

Parameter	Type	Required	Description
`content`	`string`	No	The new content for the comment.

Example Request

```
PUT /api/comments/123
Cookie: accessToken=<YOUR_ACCESS_TOKEN>
Content-Type: application/json

{
    "content": "Updated comment content here"
}
```

Example Response: 200 - OK

```
{
    "commentId": 123,
    "content": "Updated comment content here",
    "uid": 1,
    "bid": 456
}
```

Response Codes

- **200 OK**: Comment updated successfully.
- **401 Unauthorized**: Authentication token is required
- **403 Forbidden:**: Only users can edit comments, or you cannot edit someone else's comment.
- **404 Not Found**: Comment not found.
- **500 Internal Server Error**: Error updating comment.

3. Delete a Comment

- **Endpoint**: `/api/comments/{id}`
- **Method**: `DELETE`
- **Description**: Allows an authenticated user to delete their own comment.

Example Request

```
DELETE /api/comments/123
Cookie: accessToken=<YOUR_ACCESS_TOKEN>
```

Example Response: 200 - OK

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

Response Codes

- **200 OK**: Comment deleted successfully.
- **401 Unauthorized**: Authentication token is required.
- **403 Forbidden:**: Only users can delete comments, or you cannot delete someone else's comment.
- **404 Not Found**: Comment not found.
- **500 Internal Server Error**: Error deleting comment.

---

4. Post a Comment

- **Endpoint**: `/api/comments`
- **Method**: `POST`
- **Description**: Allows an authenticated user to post a comment on a specific blog post, provided the blog is not hidden.

Example Request

```
POST /api/comments
Cookie: accessToken=<YOUR_ACCESS_TOKEN>
Content-Type: application/json

{
    "bid": 3,
    "content": "This is a comment on the blog post!"
}
```

{
"commentId": 45,
"bid": 3,
"content": "This is a comment on the blog post!",
"uid": 1,
"createdAt": "2024-10-10T12:45:30Z",
"updatedAt": "2024-10-10T12:45:30Z"
}

Response Codes

- **200 OK**: Comment posted successfully.
- **401 Unauthorized**: Authentication token is required.
- **403 Forbidden:**: Only users can post comments, or the blog is hidden.
- **500 Internal Server Error**: : Error unable to create comment.

5. Get Comments

- **Endpoint**: `/api/comments`
- **Method**: `GET`
- **Description**: Retrieves comments on a specific blog or by specific filters, including pagination. Only unhidden comments are shown to unauthenticated users.

Query Parameters

- `bid` (optional): Blog ID to filter comments by blog.
- `uid` (optional): User ID to filter comments by user.
- `commentId` (optional): Comment ID to retrieve a specific comment.
- `page` (optional): Page number for pagination (default: 1).
- `limit` (optional): Number of comments per page (default: 10).

Example Request

```
GET /api/comments?bid=3&page=2&limit=5
Cookie: accessToken=<YOUR_ACCESS_TOKEN>
```

Example Response: 200 - OK


```
{
    "comments": [
        {
            "commentId": 45,
            "bid": 3,
            "content": "This is a comment!",
            "uid": 1,
            "createdAt": "2024-10-10T12:45:30Z",
            "user": {
                "uid": 1,
                "username": "john_doe"
            },
            "ratings": [],
            "replies": []
        }
    ],
    "totalComments": 25,
```

```
    "currentPage": 2,
    "totalPages": 5
```

Response Codes

- **200 OK**: Comments retrieved successfully.
- **500 Internal Server Error**: : Error retrieving comments.

6. Sort Comments by Rating

- **Endpoint**: `/api/comments/sortByRating`
- **Method**: `GET`
- **Description**: Retrieves comments for a specific blog, sorted by their rating scores (upvotes minus downvotes), with pagination support.

Query Parameters

- `bid`: The ID of the blog for which comments are to be retrieved (required).
- `page` (optional): The page number for pagination (default: 1).
- `limit` (optional): The number of comments to return per page (default: 10).

Example Request

```
GET /api/comments/sortByRating?bid=3&page=1&limit=5
```

Example Response: 200 - OK

```
[
    {
        "commentId": 12,
        "bid": 3,
        "content": "Great blog post!",
        "uid": 2,
        "createdAt": "2024-10-01T10:20:30Z",
        "score": 5,
        "ratings": [
            {"upvote": true},
            {"upvote": true},
            {"downvote": false}
        ]
    },
    {
        "commentId": 15,
        "bid": 3,
        "content": "I found this very informative.",
        "uid": 1,
        "createdAt": "2024-10-02T12:15:45Z",
        "score": 3,
        "ratings": [
```

```
            {"upvote": true},
            {"downvote": false}
        ]
    }
]
```

Response Codes

- **200 OK**: No Comments retrieved and sorted successfully.
- **405 Method**: The request method is not supported.
- **500 Internal Server Error**: Failed to filter comments due to server error.

7. Sort Comments by Reports

- **Endpoint**: `/api/comments/sortByReports`
- **Method**: `GET`
- **Description**: Retrieves comments for a specific blog, sorted by the number of reports against them, with pagination support. Only accessible to users with an ADMIN role.

Query Parameters

- `bid`: The ID of the blog for which comments are to be retrieved (required).
- `uid` (optional): The ID of the user to filter comments by.
- `commentId` (optional): The ID of a specific comment to filter by.
- `page` (optional): The page number for pagination (default: 1).
- `limit` (optional): The number of comments to return per page (default: 10).

Example Request

```
GET /api/comments/sortByReports?bid=3&page=1&limit=5
```

Example Response: 200 - OK

```
    "comments": [
        {
            "commentId": 5,
            "bid": 3,
            "uid": 2,
            "content": "This comment is inappropriate.",
            "createdAt": "2024-10-01T10:20:30Z",
            "reports": [
                {"reportId": 1, "reason": "Spam"},
                {"reportId": 2, "reason": "Offensive Language"}
            ]
        },
        {
            "commentId": 6,
```

```
            "bid": 3,
            "uid": 1,
            "content": "I disagree with this post.",
            "createdAt": "2024-10-02T12:15:45Z",
            "reports": []
        }
    ],
    "totalComments": 2,
    "currentPage": 1,
    "totalPages": 1
}
```


Response Codes

- **200 OK**: Comments retrieved and sorted successfully.
- **401 Unauthorized**: Authentication token is required.
- **403 Forbidden**: Only admins can hide content.
- **405 Method Not Allowed**: The request method is not supported.
- **500 Internal Server Error**: Something went wrong while retrieving comments.