Scriptorium Project Part 1

Instructor: Kianoosh Abbasi

CSC309H1: Programming on the Web

Fall 2024

Project Part 1

Abdullah Safi

Victor Zheng

William Liu

Table of Contents

Setting Up

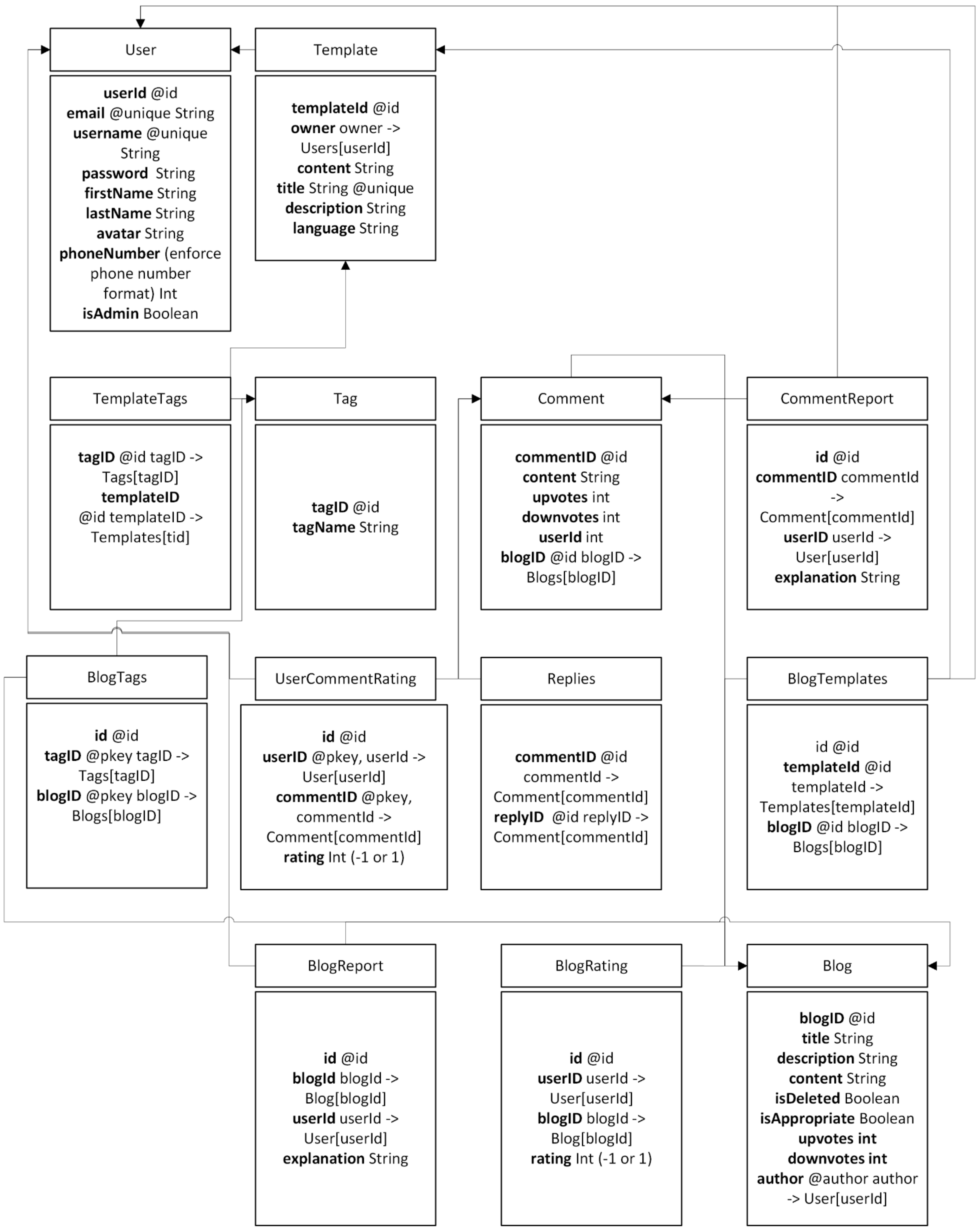
Run Setup.sh
Checks if all dependencies are installed and installs npm packages, conducts DB migrations, and checks the versions of gcc, g++, python3, and java.

Run run.sh
Builds the server then runs the server.

Admin Login:

email: admin@example.com
password: Password1!

Database Design


User
userId @id
email @unique String
username @unique String
password String
firstName String
lastName String
avatar String
phoneNumber (enforce phone number format) Int
isAdmin Boolean
Template
templateId @id
owner owner -> Users[userId]
content String
title String @unique
description String
language String
TemplateTags
tagID @id tagID -> Tags[tagID]
templateID @id templateID -> Templates[tid]
Tag
tagID @id
tagName String
Comment
commentID @id
content String
upvotes int
downvotes int
userId int
blogID @id blogID -> Blogs[blogID]
CommentReport
id @id
commentID commentId -> Comment[commentId]
userID userId -> User[userId]
explanation String
BlogTags
id @id
tagID @pkey tagID -> Tags[tagID]
blogID @pkey blogID -> Blogs[blogID]
UserCommentRating
id @id
userID @pkey, userId -> User[userId]
commentID @pkey, commentId -> Comment[commentId]
rating Int (-1 or 1)
Replies
commentID @id commentId -> Comment[commentId]
replyID @id replyID -> Comment[commentId]
BlogTemplates
id @id
templateId @id templateId -> Templates[templateId]
blogID @id blogID -> Blogs[blogID]
BlogReport
id @id
blogId blogId -> Blog[blogId]
userId userId -> User[userId]
explanation String
BlogRating
id @id
userID userId -> User[userId]
blogID blogId -> Blog[blogId]
rating Int (-1 or 1)
Blog
blogID @id
title String
description String
content String
isDeleted Boolean
isAppropriate Boolean
upvotes int
downvotes int
author @author author -> User[userId]

Summary List of API Endpoints

Signup/Login/User API

POST /api/user/signup

- Used for signing up users
- Adding people to database

POST /api/user/login

- Authenticates user to the system
    - JWT framework for authentication
    - Header authentication
- Returns true/false login
- Checks database to see if the user is logged in

POST /api/user/logout

- Logs out the current user

PATCH /api/user/profile

- Ability to edit the user
- Editing different fields through one request body

GET /api/user/data

- Retrieves the users data

Execute API

POST /api/code/execute

- Add language to the header
- Add body of the script
- Response:
    - Standard output on success
    - Standard error on failure
- Status code indicates pass/fail
- Response original highlighted + output in different fields

- Text input boxes for standard input before. JSON object with multiple possible values.
    - {stdin: [“object”, “test”, “write”]}

- Generates a process (max 10 for instance)
- Initialize each separately
- Time limit for each process to kill it
    - 10 seconds
    - Need the PID
    - Prevent I/O operations
    - Memory limit

Templates API

- GET /api/templates/[id]
    - Content of each template
    - Page links to associated blog posts
    - Open to everyone
    - Returns the template for execution
- GET /api/templates?params
    - Return a list of ids that match the query
    - Param by title and tags
        - Front end should display each id as a card with a title, tags, and content summary
- POST /api/templates/create
    - Title, explanation, and tags
    - Authenticated user
    - Needs to return template ID, which is a value visible to the user
- PATCH /api/templates/[id]
    - Edit the template
    - Authenticated users and only owner can edit
    - Allows users to also add tags
- PATCH /api/templates/[id]
    - Delete the template
    - Authenticated users and only owner can delete
    - Automatically deletes associated tags
- POST /api/templates/[id]

    - Forks the template (only for authenticated users)
        - Get notified that its a forked version
    - Response: template body which can be displayed by the front end on the homepage

Blogs API

- POST /api/blogs/create
    - Body with title, description, tag
    - Links by user provided the template ID, which will be visible to them
    - Authentication
    - Returns the ID of the newly created blog
- DELETE /api/blogs/[id]
    - Delete blog from the DB
- PATCH /api/blogs/[id]
    - Editing the blog
- GET /api/blogs/search?params
    - Search parameters
        - Content (string)
        - Title (string)
        - Tags: array of tag names (strings)
        - Template titles (strings) contained in the blog
    - Empty then return all with upvotes and downvotes
    - Needs pagination
- GET /api/blogs/[id]
    - Returns the blog and everything related to it so that frontend can display it. Only returns direct comments, replies are not returned.
    - Includes ratings of blogs and comments too
- POST /api/blogs/[id]/report
    - Explanation, userID of person making the report
    - commentID (otherwise blog)
- GET /api/blogs/reports
    - Reports of comments and blogs that are received
    - Only accessible by Admins
    - Return blogs in sorted order of most reports
- PATCH /api/blogs/reports
    - Flag a content as inappropriate or appropriate

- POST /api/blogs/[id]/comment
    - Comment on the blog
    - Ability to link to another comment
    - Adding comments or replying to comment
        - If replying, include the comment ID you are replying to
- POST /api/blogs/[id]/comment/rate
    - Rate the comment specified in the request body
    - Check if the user has already rated. If they haven't, add the rating. If they have, ensure that the current rating is the only one in the DB (they cannot double vote, and they cannot vote up and down at the same time).
- GET /api/blogs/[id]/comment/replies
    - Retrieves the direct replies for the comment based on provided comment id
- POST /api/blogs/[id]/rate
    - Handle both add and remove upvote
    - Check if the user has already rated. If they haven't, add the rating. If they have, ensure that the current rating is the only one in the DB (they cannot double vote, and they cannot vote up and down at the same time).
    - User rating for the blog
- GET/api/blogs/ranked
    - Get all blogs sorted by ratings
- GET/api/blogs/[id]/ranked
    - Get all comments sorted by ratings

Tags API

- GET /api/tags
    - Get all existing tags
    - No auth required
- GET /api/tags/[id]
    - Get all items linked to this tag
    - No auth required

Detailed Endpoint Descriptions

Note: Example requests and responses generated by ChatGPT, documentation formatted by ChatGPT. Reviewed by us and modified to ensure accuracy.

Documentation Structure:

/api/endpoint

Description: …
Allowed Methods: …
Payload: list description of request body payload…
Headers: …

Example Request:
Example complete request…

Example Response:
Example complete response…

User API

/api/user/signup

Description: This endpoint allows a new user to sign up by providing their email, username, password, and name details. The user's password is securely hashed before storing it in the database. If the email or username is already in use, appropriate error messages will be returned. Returns the created user, without their password or refresh token data.
Allowed Methods: POST
Payload:

- email: String (required) - The user's email address. Must be a valid email format.
- username: String (required) - The desired username. Must be at least 3 characters long and only contain alphanumeric characters or underscores.
- password: String (required) - The user's password. Must meet the following criteria:
  - At least 8 characters long
  - Contains at least one uppercase letter, one lowercase letter, one number, and one special character
- firstName: String (required) - The user's first name. Must only contain letters (including accented characters), spaces, or hyphens.
- lastName: String (required) - The user's last name. Must only contain letters (including accented characters), spaces, or hyphens.

Example Request:
```
{
  "email": "user@example.com",
  "username": "new_user123",
  "password": "Password@123",
  "firstName": "John",
  "lastName": "Doe"
}
```

Example Response:
```
{
  "message": "User created",
  "user": {
    "userId": 1,
    "email": "user@example.com",
    "username": "user123",
    "firstName": "John",
    "lastName": "Doe",
    "avatar": null,
    "phoneNumber": null,
    "isAdmin": false,
    "createdAt": "2024-11-01T12:00:00.000Z",
    "updatedAt": "2024-11-01T12:00:00.000Z"
  }
}
```

/api/user/refresh

Description: This endpoint allows a registered user to refresh their login by providing their refresh token. If the credentials are valid, a new access token and a new refresh token are generated and returned. The tokens are used for session management, with the access token expiring after a short period and the refresh token allowing the user to obtain a new access token. The refresh token is stored in a database. Refresh token is updated for 1 hr and access token for 15 mins.
Allowed Methods: POST
Payload:

- Authorized user header

Example Request:
POST /api/user/refresh

Example Response:
```
{
  "userId": 7,
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2s….",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2V….."
}
```

/api/user/login

Description: This endpoint allows a registered user to log in by providing their email and password. If the credentials are valid, an access token and a refresh token are generated and returned. The tokens are used for session management, with the access token expiring after a short period and the refresh token allowing the user to obtain a new access token. The refresh token is stored in a database.
Allowed Methods: POST
Payload:

- email: String (required) - The user's email address. Must be a valid email format.
- password: String (required) - The user's password.

Example Request:

```
POST /api/user/login
{
  "email": "user@example.com",
  "password": "Password@123"
}
```

Example Response:

```
{
  "message": "Login successful",
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzjhvjhvjhbbJHBGhXVCJ9..."
}
```

/api/user/logout

Description: This endpoint logs out an authenticated user by invalidating their refresh token, setting it to null in the server. The user must provide a valid access token in the request headers. If the user is already logged out or unauthorized, appropriate error messages will be returned.
Allowed Methods: POST
Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.

Example Response:
```
{
  "message": "Logged out successfully"
}
```

/api/user/profile

Description: This endpoint allows an authenticated user to update their profile information. The user must provide a valid Bearer token in the request headers. The request body can include fields such as email, username, password, first name, last name, avatar, and phone number. Appropriate validation will be applied to each field, and any errors will be returned if the input is invalid or if the update fails due to uniqueness constraints. The updated user is returned in the response body, excluding the user's hashed password and refresh token.
Allowed Methods: PATCH
Payload:
- email: String (optional) - The user's email address. Must be a valid email format.
- username: String (optional) - The user's username. Must contain only alphanumeric characters and underscores, with a minimum length of 3 characters.
- password: String (optional) - The user's password. Must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, one number, and one special character.
- firstName: String (optional) - The user's first name. Must contain only alphabetic characters, accented characters, spaces, or hyphens.
- lastName: String (optional) - The user's last name. Must contain only alphabetic characters, accented characters, spaces, or hyphens.
- avatar: String (optional) - The filename of the user's avatar image. Must be a valid PNG file that exists on the server.
- phoneNumber: String (optional) - The user's phone number in the format XXX-XXX-XXXX.

Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.

Example Request:
```
{
  "email": "newemail@example.com"
  "username": "newusername",
  "phoneNumber": "123-458-8888",
  "avatar": "bear.png"
}
```

Example Response:

```
{
  "message": "Profile updated",
  "user": {
     "userId": 8,
     "email": "newemail@example.com",
     "username": "newusername",
     "firstName": "Will",
     "lastName": "Liu",
     "avatar": "bear.png",
     "phoneNumber": "123-458-8888",
     "isAdmin": false,
     "createdAt": "2024-10-30T18:34:14.066Z",
     "updatedAt": "2024-10-30T19:21:10.176Z"
  }
}
```

/api/user/data

Description: This endpoint allows retrieval of basic data about the user, such as their userId, email, username, firstName, lastName, avatar, phoneNumber, isAdmin status, and user creation/revision time. Note that the refresh token and hashed password are not returned. This serves as the primary way to retrieve information about the user.
Allowed Methods: GET

Example Response:

```
{
  "message": "Profile fetched",
  "user": {
     "userId": 1,
     "email": "admin@example.com",
     "username": "admin",
     "firstName": "John",
     "lastName": "Doe",
     "avatar": "bear.png",
     "phoneNumber": "123-456-6789",
     "isAdmin": true,
     "createdAt": "2024-01-05T10:30:00.000Z",
     "updatedAt": "2024-11-03T17:56:04.739Z"
  }
}
```

Execute Code API

/api/code/execute

Description: This endpoint executes code provided by the user in a specified programming language and returns the output. It supports various languages and handles standard input. The response structure is consistent, even in error cases.
Allowed Methods: POST
Payload:

- code: String (required) - The code to be executed.
- language: String (required) - The programming language of the code. Accepted values: "python", "javascript", "java", "c", "cpp".
- input: String (optional) - Standard input to be passed to the program, if necessary. If not passed, it will default to an empty string.

Returns:

- success: Boolean - Indicates whether the execution was successful.
- output: String - The standard output that results from executing the program.
- error: String - The error(s) that result from executing the program (if any).
- warnings: String - Any warnings generated during compilation (if applicable).

Example Request:
```
{
  "code": "print('Hello, World!')",
  "language": "python",
}
```

Example Response (Success):
```
{
  "success": true,
  "output": "Hello, World!\n",
  "error": "",
  "warnings": ""
}
```

Example Response (Error):
```
{
  "success": false,
  "output": "",
  "error": "Execution timed out (10 second limit)",
  "warnings": ""
}
```

Templates API

/api/templates/

Description: Returns a paginated list of templates, each with details about its content, tags, and metadata. Ability to search for templates using content, title, or tags.

Allowed Methods: GET

Payload / Query Parameters:

- ○ page (optional, integer): The page number for pagination.
- ○ limit (optional, integer): Number of templates to return per page.
- ○ Title, tags, or content can be used to filter for items under this endpoint
    - ■ Title to filter based on templates' title
    - ■ Content to filter based on templates' content
    - ■ Tags to filter based on templates' tags

Returns:

- templates: Array of template objects, each with:
    - templateId (integer): Unique identifier of the template.
    - userId (integer): ID of the user who created the template.
    - content (string): Template content or code.
    - title (string): Template title.
    - description (string): Brief description of the template.
    - language (string): Programming language of the template.
    - createdAt (string, ISO format): Timestamp of creation.
    - updatedAt (string, ISO format): Timestamp of the last update.
    - templatesTags: Array of tags associated with the template, where each tag includes:
        - templateTagId (integer): Unique identifier for the template-tag relation.
        - tagId (integer): Unique identifier for the tag.
        - templateId (integer): ID of the template associated with this tag.
        - tag: An object containing:
            - tagId (integer): Tag's unique identifier.
            - tagName (string): Name of the tag.

- totalPages (integer): Total number of pages in the result set.
- page (integer): Current page number.
- limit (integer): Maximum number of templates per page.

Example Request:

GET /api/templates?page=1&limit=10&title=python

Example Response:

```
{
  "templates": [
    {
      "templateId": 4,
      "userId": 1,
      "content": "print(\"hello world\")",
      "title": "Python Hello World Example 441423",
      "description": "This program prints out hello world to users.",
      "language": "python3",
      "createdAt": "2024-11-03T19:28:20.438Z",
      "updatedAt": "2024-11-03T19:28:20.438Z",
      "templatesTags": []
    },
  ],
  "totalPages": 1,
  "page": 1,
  "limit": 10
}
```

/api/templates/create

Description: Allows an authenticated user to create a new template by submitting content, metadata, and associated tags.

Allowed Endpoints: POST

Headers:

- **Authentication Required**: Yes (only authenticated users can create templates). Only authenticated users; user ID is inferred from the authenticated session

Payload / Request Body:

The request should include the following JSON structure:

- title (string): The title of the template, which should be unique. Example: "Python Hello World Example {{rand_unique_id}}"
- content (string): The code or content of the template. Example: "print(\"hello world\")"
- language (string): The programming language of the template. Example: "python3"
- description (string): A brief description of what the template does. Example: "This program prints out hello world to users."
- tags (array of strings): Tags associated with the template for categorization and easy searching. Example: ["hello world", "python", "test"]

Response/Return Body

On successful creation, the API returns a JSON object containing the newly created template's details along with the associated tags.

- **Fields**:
  - template (object): Contains information about the newly created template:
    - templateId (integer): The unique identifier of the template.
    - userId (integer): The ID of the user who created the template.
    - content (string): The template content.
    - title (string): The title of the template.
    - description (string): The description provided for the template.
    - language (string): The programming language of the template.
    - createdAt (string, ISO format): The timestamp when the template was created.
    - updatedAt (string, ISO format): The timestamp of the most recent update.
  - tags (array of strings): Array of tags associated with the template, reflecting those sent in the request.

Example Request:

```
{
  "title": "Python Hello World Example {{rand_unique_id}}",
  "content": "print(\"hello world\")",
  "language": "python3",
  "description": "This program prints out hello world to users.",
  "tags": ["hello world", "python", "test"]

}
```

Example Response:

```
{

  "template": {
    "templateId": 12,
    "userId": 1,
    "content": "print(\"hello world\")",
    "title": "Python Hello World Example 120757",
    "description": "This program prints out hello world to users.",
    "language": "python3",
    "createdAt": "2024-11-03T19:35:08.417Z",
    "updatedAt": "2024-11-03T19:35:08.417Z"
  },
  "tags": [
    "hello world",
    "python",
    "test"
  ]
}
```

/api/templates/[id] - Fork a Template

Description: Allows an authenticated user to create a fork of an existing template. The new template will have the same content and attributes as the original, but it will be assigned a new unique identifier and can be modified independently.

Allowed Methods / Endpoint: POST /api/templates/[id] (where [id] is the ID of the template to be forked)

Headers & Payload:

- **Authentication Required**: Yes (only authenticated users can fork templates)
- **Permissions**: Only authenticated users

Response

On successful forking, the API returns a JSON object containing the details of the newly created forked template. To ensure uniqueness, the title contains the date that it was forked.

- **Fields**:
    - templateId (integer): The unique identifier for the new forked template.
    - userId (integer): The ID of the user who created the fork.
    - content (string): The content of the original template, which remains unchanged in the fork.
    - title (string): The title of the forked template. It may include additional context, such as the original title with a note indicating it is a fork, along with a timestamp.
    - description (string): The description of the template, inherited from the original.
    - language (string): The programming language of the template, inherited from the original.
    - createdAt (string, ISO format): The timestamp when the fork was created.
    - updatedAt (string, ISO format): The timestamp of the most recent update, which will initially be the same as the creation time.

Example Request:

POST /api/templates/[id]

Example Response:

```
{
   "templateId": 23,
   "userId": 1,
   "content": "print(\"hello world\")",
   "title": "Python Hello World Example 86003 (Forked) Sun Nov 03 2024 16:40:30
GMT-0500 (Eastern Standard Time)",
   "description": "This program prints out hello world to users.",
   "language": "python3",
   "createdAt": "2024-11-03T21:40:30.597Z",
   "updatedAt": "2024-11-03T21:40:30.597Z"
}
```

/api/templates/[id] - Update Template

Description: Allows an authenticated user to modify an existing template. The user can update the content, language, description, and tags of the template. Only owner can edit their own template.

Allowed Methods / Endpoint: PATCH /api/templates/[id] (where [id] is the ID of the template to be updated, e.g., 2)

Headers

- **Authentication Required**: Yes (only authenticated users can edit templates)
- **Permissions**: Only the owner of the template can edit it

Payload

Request Body

The request should include the fields that need to be updated in JSON format. Allows users to change content, language, description, title, or tags in the same format as /api/templates/create. Only item that cannot be modified is the userId.

- title (string): The title of the template, which should be unique. Example: "Python Hello World Example {{rand_unique_id}}"
- content (string): The code or content of the template. Example: "print(\"hello world\")"
- language (string): The programming language of the template. Example: "python3"
- description (string): A brief description of what the template does. Example: "This program prints out hello world to users."
- tags (array of strings): Tags associated with the template for categorization and easy searching. Example: ["hello world", "python", "test"]

Response

Upon successful update, the API returns the updated template information.

- **Fields** (Response Structure):
    - templateId (integer): The unique identifier for the updated template.
    - userId (integer): The ID of the user who owns the template.
    - content (string): The updated content of the template.
    - title (string): The title of the template (may remain unchanged unless specified).
    - description (string): The updated description of the template.
    - language (string): The programming language of the template.
    - createdAt (string, ISO format): The timestamp when the template was created.
    - updatedAt (string, ISO format): The timestamp of the most recent update.

Example Request Body:

```
{
   "content": "print(\"hello world\")\nprint(\"hello doctor.\")",
   "language": "python3",
   "description": "This program not only prints out hello world to users but also prints
out hello doctor.",
   "tags": ["new tag", "python remaster"]
}
```

Example Response:

```
{
   "templateId": 2,
   "userId": 1,
   "content": "print(\"hello world\")\nprint(\"hello doctor.\")",
   "title": "Original Title (if unchanged)",
   "description": "This program not only prints out hello world to users but also prints
out hello doctor.",
   "language": "python3",
   "createdAt": "2024-11-03T19:28:40.425Z",
   "updatedAt": "2024-11-03T21:40:30.597Z"
}
```

/api/templates/[id] - Delete Template

Description: Allows an authenticated user to delete an existing template. The user can only delete their own templates.

Allowed Methods / Endpoint: DELETE /api/templates/[id] (where [id] is the ID of the template to be deleted, e.g., 23)

Headers

- **Authentication Required**: Yes (only authenticated users can delete templates)
- **Permissions**: Only the owner of the template can delete it

Request Body / Payload

No request body is needed for this operation. Simply send the delete request to the specified endpoint.

Response

Upon successful deletion, the API returns a message confirming the deletion along with the details of the deleted template.

- **Response Structure**:
    - message (string): Confirmation message indicating successful deletion.
    - deletedTemplate (object): The details of the template that was deleted, including:
        - templateId (integer): The unique identifier of the deleted template.
        - userId (integer): The ID of the user who owned the template.
        - content (string): The content of the deleted template.
        - title (string): The title of the deleted template.
        - description (string): The description of the deleted template.
        - language (string): The programming language of the deleted template.
        - createdAt (string, ISO format): The timestamp when the template was created.
        - updatedAt (string, ISO format): The timestamp of the most recent update.

Example Response:

{

```
    "message": "Successfully deleted",
    "deletedTemplate": {
        "templateId": 23,
        "userId": 1,
        "content": "print(\"hello world\")",
        "title": "Python Hello World Example 86003 (Forked) Sun Nov 03 2024 16:40:30
GMT-0500 (Eastern Standard Time)",
        "description": "This program prints out hello world to users.",
        "language": "python3",
        "createdAt": "2024-11-03T21:40:30.597Z",
        "updatedAt": "2024-11-03T21:40:30.597Z"
    }
}
```

/api/templates/[id] - Get Template by ID

Description: Allows users to retrieve the details of a specific template using its unique identifier.

Allowed Methods / Endpoint: GET /api/templates/[id] (where [id] is the ID of the template to be retrieved, e.g., 19)

- **Authentication Required**: No (open to everyone)
- **Permissions**: None (anyone can view template details)

Request Body / Payload

N/A

Response

Upon successful retrieval, the API returns the details of the requested template, including any associated tags.

- **Response Structure**:
    - template (object): The details of the retrieved template, including:
        - templateId (integer): The unique identifier of the template.
        - userId (integer): The ID of the user who owns the template.
        - content (string): The content of the template.

        - title (string): The title of the template.
        - description (string): The description of the template.
        - language (string): The programming language of the template.
        - createdAt (string, ISO format): The timestamp when the template was created.
        - updatedAt (string, ISO format): The timestamp of the most recent update.
    - templateTags (array): An array of tags associated with the template (can be empty if no tags are present).

Example Response:

```
{
   "template": {
      "templateId": 19,
      "userId": 1,
      "content": "print(\"hello world\")",
      "title": "Python Hello World Example 334943 (Forked) Sun Nov 03 2024 16:35:56
GMT-0500 (Eastern Standard Time)",
      "description": "This program prints out hello world to users.",
      "language": "python3",
      "createdAt": "2024-11-03T21:35:56.738Z",
      "updatedAt": "2024-11-03T21:35:56.738Z"
   },
   "templateTags": []
}
```

Blogs API

/api/blogs/create

Description: This endpoint allows an authenticated user to create a new blog post. The user must provide a valid Bearer token in the request headers. The request body should include the title, description, content, tags, and linked template IDs. The title and description must meet specific validation criteria, and tags must be alphanumeric with no spaces. If the provided tags do not exist, they will be created. The specified templates however must exist; otherwise, an error will be returned. The blogId of the created blog will be returned.
Allowed Methods: POST
Payload:

- title: String (required) - The title of the blog post. Must be between 5 and 100 characters, allowing alphanumeric characters and spaces.
- description: String (required) - The description of the blog post. Must be between 0 and 100 characters, allowing alphanumeric characters, spaces, and basic punctuation.
- content: String (required) - The content of the blog post. Must be between 20 and 1000 characters, allowing alphanumeric characters, spaces, and basic punctuation.
- tags: Array of Strings (required) - An array of tags for the blog post. Each tag must be alphanumeric, no spaces, and between 1 and 20 characters.
- templateIds: Array of Integers (required) - An array of template IDs. Each ID must be a valid integer.

Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.

Example Request:
```
{
  "title": "My First Blog Post",
  "description": "This is a detailed description of my first blog post.",
  "content": "Welcome to my first blog post! I'm excited to share my thoughts and experiences with you. In this blog, I'll explore various topics, from personal anecdotes to commentary on current events. Join me on this journey as we learn and discuss together!",
  "tags": ["first", "blog", "post"],
  "templateIds": [1, 2, 3]
}
```

Example Response:

```
{
  "message": "Blog created",
  "blogId": 22
}
```

/api/blogs/[id]

Description: Performs tasks related to the blog of query param id based on the method
Allowed Methods: DELETE, PATCH, GET

DELETE:

Description: Delete a specific blog post by id in the query param. This requires authentication from the user who authored the blog. Sets the blogs isDeleted to true. Returns the updated blog in the response body.
Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.

Example Response:

```
{
  "message": "Blog deleted successfully",
  "updatedBlog": {
     "blogId": 12,
     "title": "My First Blog Post",
     "description": "This is a detailed description of my first blog post.",
     "content": "Welcome to my first blog post! I'm excited to share my thoughts
and experiences with you. In this blog, I'll explore various topics, from personal
anecdotes to commentary on current events. Join me on this journey as we
learn and discuss together!"
     "isDeleted": true,
     "isAppropriate": true,
     "createdAt": "2024-11-02T03:10:14.527Z",
    "updatedAt": "2024-11-02T03:37:52.413Z",
    "authorId": 8
  }
}
```

PATCH:

Description: Update an existing blog post based on query param blog id. This requires authentication from the user authoring the blog. Any attributes included in the payload will be updated. When updating tags and templateIds, the payload should include the new complete set of tags and/or linked templateIds, as the previous is deleted. The updated blog is returned in the response body.

Headers: Authorization: Bearer Token (required) - The access token for the authenticated user.

Payload:

- title (optional): New title for the blog (string, 5-100 characters).
- description (optional): New description for the blog (string, 0-100 characters).
- content (optional): New content for the blog (string, 20-1000 characters).
- tags (optional): Array of new tags (strings, 1-20 characters, alphanumeric).
- templateIds (optional): Array of new template IDs (integers).

Example Request:

```
{
  "title": "Updated Blog Title",
  "description": "This is the updated description of the blog.",
  "content": "This is the updated content of the blog.",
  "tags": ["tag1", "tag2"],
  "templateIds": [1, 2, 3]
}
```

Example Response:

```
{
  "message": "Blog updated successfully",
  "updatedBlog": {
    "blogId": 11,
        "title": "Updated Blog Title",
        "description": "This is the updated description of the blog.",
        "content": "This is the updated content of the blog.",
        "isDeleted": true,
        "isAppropriate": true,
        "createdAt": "2024-11-02T03:10:13.445Z",
        "updatedAt": "2024-11-02T04:04:18.957Z",
        "authorId": 8
    }
}
```

GET:

Description: Retrieves all information about a specific blog post based on the query param id, including comments/replies associated with the blog. If a comment is a reply, it will mention the comment id to which it is replying to - the comment which it is the reply child of. This request does not require authentication. The response body contains the retrieved blog data.

Example Response:

```
{
    "message": "Successfully retrieved blog",
    "blogDetails": {
        "blogId": 5,
        "title": "Tech Innovations",
        "description": "The latest trends in technology and what they mean for the
future.",
        "content": "In the fast-paced world of technology, innovation is a constant.
From AI advancements to breakthroughs in renewable energy, these trends are
reshaping industries and our daily lives. Artificial Intelligence is now more
integrated than ever, enhancing everything from healthcare to transportation.
The rise of smart devices is making our homes more interconnected, leading to
improved efficiency and convenience. As we look ahead, we must consider the
```

```
ethical implications of these technologies and how they impact our society.
Staying informed and adaptable is crucial in this rapidly evolving landscape."
    "isDeleted": false,
    "isAppropriate": true,
    "upvotes": 3,
    "downvotes": 2,
    "createdAt": "2024-11-02T18:46:34.466Z",
    "updatedAt": "2024-11-02T18:46:34.466Z",
    "authorId": 1,
    "Comment": [
      {
        "commentId": 6,
        "content": "an amazing comment",
        "upvotes": 2,
        "downvotes": 3,
        "user": {
          "username": "testuser2"
        },
        "CommentReplyChild": []
      },
      {
        "commentId": 7,
        "content": "a reply to the amazing comment",
        "upvotes": 3,
        "downvotes": 0,
        "user": {
          "username": "testuser2"
        },
                "CommentReplyChild": [
                    {
                        "commentId": 6
                    }
                ]
            },
            {
                "commentId": 8,
                "content": "a reply to the amazing comment",
                "upvotes": 0,
                "downvotes": 2,
                "user": {
                    "username": "testuser2"
                },
                "CommentReplyChild": [
                    {
                        "commentId": 6
                    }
                ]
            }
        ]
    }
}
```

/api/blogs/[id]/report

Description: Allows a user to report a specified blog or a comment on the blog (identified by the blog ID in the URL). Users must provide an explanation for the report. The userId is taken from the access token and should not be included in the payload.
Allowed Methods: POST

Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.
Payload:

- explanation: String (optional) - The reason for reporting the blog or comment.
- commentId: Int (optional) - The ID of the comment to report. If not provided, the report is a report on the blog with ID specified in the URL.

Returns:

- newCommentReport: CommentReport (if reporting a comment)
- newBlogReport: BlogReport (if reporting a blog)

Example Request (reporting a blog):

POST /api/blogs/1/report

```
{
   "explanation": "Inappropriate content"
}
```

Example Response (reporting a blog):

```
{
   "newBlogReport": {
        "userId": 3,
        "blogId": 1,
        "explanation": "Inappropriate content"
   }
}
```

Example Request (reporting a comment):

POST /api/blogs/1/report

```
{
   "commentId": 5,
   "explanation": "Spam"
}
```

Example Response (reporting a comment):

```
{
   "newCommentReport": {
        "userId": 3,
        "commentId": 5,
        "explanation": "Spam"
   }
}
```

/api/blogs/search

Description: Searches for blogs based on specified criteria, allowing users to filter results for more relevant content.
Allowed Methods: GET
Query Parameters:
 - title: String (optional) - Filter results by specific blog titles. Matches can be partial.
 - content: String (optional) - Filter results based on keywords or phrases found in the blog content. Matches can be partial.
 - tags: Array of Strings (optional) - Filter results to include only blogs that have specific tags associated with them.
 - templates: Array of Strings (optional) - Filter results to include only blogs that are linked to specified template names.
 - page: Integer (optional) - The page number to retrieve for paginated results, starting from 1.
 - limit: Integer (optional) - The maximum number of blogs to return per page, with a default limit set by the server if not specified.

Returns:

- blogs: Array of blogs that meet the given search criteria

Example Request:

```
GET /api/blogs/search?page=1&limit=10&title=Understanding
```

Example Response:

```
 {
   "blogs": [
     {
       "blogId": 5,
       "title": "Understanding Prisma Relations",
       "content": "A detailed guide...",
       "authorId": 1,
       "createdAt": "2024-11-01T12:00:00.000Z",
       "updatedAt": "2024-11-01T12:00:00.000Z"
     },
     {
       "blogId": 6,
       "title": "Understanding JavaScript Closures",
       "content": "Exploring closures in JavaScript...",
       "authorId": 2,
       "createdAt": "2024-10-30T09:00:00.000Z",
       "updatedAt": "2024-10-30T09:00:00.000Z"
     }
   ],
   "totalCount": 2,
   "totalPages": 1,
   "page": 1,
   "limit": 10
 }
```

/api/blogs/reports

Description: This endpoint is only accessible by Admins. Returns blogs and comments sorted by the number of reports (GET). Additionally, this endpoint allows for flagging content as appropriate or inappropriate (PATCH).

Allowed Methods: GET, PATCH

Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.

GET Request:
Returns:
- sortedBlogs: Array of blog objects sorted by the number of reports.
- sortedComments: Array of comment objects sorted by the number of reports.

Example Response (GET):

```
{
  "sortedBlogs": [
    {
      "blogId": 1,
      "title": "Blog Title 1",
      "reportCount": 5
    },
    // more blog objects...
  ],
  "sortedComments": [
    {
      "commentId": 1,
      "content": "Comment content 1",
      "reportCount": 3
    },
    // more comment objects...
  ]
}
```

PATCH Request:

Payload:

- id: Integer (required) - The ID of the content to be flagged.
- type: String (required) - The type of content, either "blog" or "comment".
- isAppropriate: Boolean (required) - Indicates whether the content is appropriate (true) or inappropriate (false).

Example Request (PATCH):

```
PATCH /api/blogs/reports
{
  "id": 1,
  "type": "blog",
  "isAppropriate": false
}
```

Example Response (PATCH):

```
{
  "updatedBlog": {
    "blogId": 1,
    "isAppropriate": false,
    // other blog properties...
  }
}
OR
{
  "updatedComment": {
    "commentId": 1,
    "isAppropriate": false,
    // other comment properties...
  }
}
```

/api/blogs/[id]/comment

Description: Creates a comment on the specified blog (identified by the blog ID in the URL). Ability to link to another comment by replying to it. The userId is taken from the access token and should not be included in the payload.
Allowed Methods: POST
Payload:
  - content: String (required) - The content of the comment.
  - commentId: Int (optional) - The ID of the comment to reply to. If specified, this comment will be a reply to the comment with this ID. If not specified, this comment will be a standalone comment on the blog specified with the ID in the URL.

Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.

Returns:
  - newComment: Comment (if creating a new comment)
  - newCommentReply: CommentReply (if creating a reply to an existing comment)

Example Request:

```
  POST /api/blogs/1/comment
  {
```

```
  "content": "This is a great blog post!",
  "commentId": 2,
}
```

Example Response:

```
{
  "newComment": {
    "commentId": 3,
    "content": "This is a great blog post!",
    "isAppropriate": true,
    "upvotes": 0,
    "downvotes": 0,
    "userId": 3,
    "blogId": 1,
    "createdAt": "2024-11-01T12:00:00.000Z",
    "updatedAt": "2024-11-01T12:00:00.000Z"
  },
  "newCommentReply": {
    "commentId": 2,
    "replyId": 3
  }
}
```

/api/blogs/[id]/comment/rate

Description: Allows a user to rate a comment on a specified blog (identified by the blog ID in the URL). Users can upvote, downvote, or remove their rating on a comment. The userId is taken from the access token and should not be included in the payload.
Allowed Methods: POST
Headers:
Authorization: Bearer Token (required) - The access token for the authenticated user.
Payload:

- rating: Int (required) - The rating value. Must be 1 for upvote, -1 for downvote, or 0 to remove the rating.
- commentId: Int (required) - The ID of the comment to be rated.

Returns:

- newRating: UserCommentRating (if creating a new rating or updating an existing rating)

Example Request:

POST /api/blogs/1/comment/rate

```
{
   "rating": 1,
   "commentId": 2
}
```

Example Response:

```
{
   "newRating": {
        "userId": 3,
        "commentId": 2,
        "rating": 1
   }
}
```

/api/blogs/[id]/comment/replies

Description: Returns all the replies to the comment with the comment id provided in the request body. Replies returned will have reply id, content, upvotes, downvotes, and the username of the author.
Allowed Methods: GET
Payload:
- commentId: Integer (required) - The ID of the comment to find replies to

Example Request:

```
{
  “commentId”: 2
}
```

Example Response:

```
{
  "message": "Successfully retrieved replies",
  "replies": [
     {
        "replyId": 4,
        "comment": {
           "content": "I disagree with some points made here.",
           "upvotes": 0,
           "downvotes": 3,
           "user": {
              "username": "johndoe"
           }
        }
     }
  ]
}
```

/api/blogs/[id]/rate

Description: Allows a user to rate a specified blog (identified by the blog ID in the URL). Users can upvote, downvote, or remove their rating. The userId is taken from the access token and should not be included in the payload.

Allowed Methods: POST

Headers:

Authorization: Bearer Token (required) - The access token for the authenticated user.

Payload:

- rating: Int (required) - The rating value. Must be 1 for upvote, -1 for downvote, or 0 to remove the rating.

Returns:

- newRating: BlogRating (if creating a new rating or updating an existing rating)

Example Request:

```
POST /api/blogs/1/rate
{
   "rating": 1
}
```

Example Response:

```
{
   "newRating": {
         "userId": 3,
         "blogId": 1,
         "rating": 1
   }
}
```

/api/blogs/ranked

Returns all blogs ranked by upvotes or downvotes.

Allowed Methods: GET

Query Parameters:

- order (optional): The order to sort blogs. Must be either "upvotes" or "downvotes" (default is "upvotes").
- ascending (optional): Sort order. Must be either "true" or "false" (default is "true").

Returns:

- rankedBlogs: An array of blogs sorted by the given criteria.

Example Request:

```
GET /api/blogs/ranked?order=upvotes&ascending=true
```

Example Response:

```
{
   "rankedBlogs": [
      {
         "blogId": 1,
         "title": "First Blog",
         "content": "Content of the first blog.",
         "upvotes": 100,
         "downvotes": 5
      },
      {
         "blogId": 2,
         "title": "Second Blog",
         "content": "Content of the second blog.",
         "upvotes": 50,
         "downvotes": 3
      }
   ]
}
```

/api/blogs/[id]/ranked

Returns all comments on the blog with the ID in the URL.

Allowed Methods: GET

Query Parameters:

- order (optional): The order to sort comments. Must be either "upvotes" or "downvotes" (default is "upvotes").
- ascending (optional): Sort order. Must be either "true" or "false" (default is "true").

Returns:

- rankedComments: An array of comments on the given blog sorted by the given criteria.

Example Request:

GET /api/blogs/1/ranked?order=upvotes&ascending=true

Example Response:

```
{
  "rankedComments": [
    {
      "commentId": 1,
      "blogId": 1,
      "userId": 2,
      "content": "This is a comment.",
      "upvotes": 10,
      "downvotes": 2
    },
    {
      "commentId": 2,
      "blogId": 1,
      "userId": 3,
      "content": "This is another comment.",
      "upvotes": 5,
      "downvotes": 1
    }
  ]
}
```

Tags API

/api/tags

Description: Returns a paginated list of existing tags.

Allowed Methods: GET

Payload / Query Params:

- page (optional, integer): The page number to retrieve (default is 1).
- limit (optional, integer): The maximum number of tags to retrieve per page (default is 10).

Returns / Response:

- tags: An array of tag objects, each containing:
  - tagId (integer): Unique identifier for the tag.
  - tagName (string): Name of the tag.
- totalPages (integer): Total number of pages available based on the current limit.
- page (integer): The current page number.
- limit (integer): The number of tags returned per page.

Example Request:

```
GET /api/tags?page=1&limit=10
```

Example Response:

```
{
  "tags": [
    {
      "tagId": 1,
      "tagName": "hello world"
    },
    {
      "tagId": 6,
      "tagName": "python remaster"
    },
  ],
  "totalPages": 1,
  "page": 1,
  "limit": 10
}
```

/api/tags/[id]

Description: Returns all blogs and templates linked to the tag with the specified ID.

Allowed Methods: GET

Payload:

- ○ id (integer): The unique identifier of the tag.

Returns:

- linkedBlogs: An array of blogs associated with the tag (if any).
- linkedTemplates: An array of templates associated with the tag (if any).

Example Request:

GET /api/tags/1

Example Response:

```
{
  "linkedBlogs": [],
  "linkedTemplates": [
    {
      "templateTagId": 1,
      "tagId": 1,
      "templateId": 6,
      "template": {
        "templateId": 6,
        "userId": 1,
        "content": "print(\"hello world\") test",
        "title": "Python Hello World Example 920101",
        "description": "This program prints out hello world to users.",
        "language": "python3",
        "createdAt": "2024-11-03T19:29:17.341Z",
        "updatedAt": "2024-11-03T20:20:22.847Z"
      }
    ]
}
```