CSC309 Scriptorium Documentation

Authors:

- **Daniel Hyunseo Lee**
- **Jodhvir Singh Bassi**
- **David ZHEGLOV**

Course: CSC309
Project: Scriptorium

Models

1) User

- Description: Represents users of the Scriptorium platform, including their personal details and relationships to content.
- Fields:

1. id: Unique identifier for each user (auto-incrementing integer).
2. email: User's email address (unique).
3. passwordHash: Hashed password for secure authentication.
4. firstName: User's first name.
5. lastName: User's last name.
6. avatar: Optional URL to the user's avatar.
7. phoneNumber: Optional user's phone number.
8. role: Role of the user (default is "USER").
9. createdAt: Timestamp for when the user was created (default to now).
10. updatedAt: Timestamp for when the user was last updated (automatically updated).

- Relations:

1. templates: Templates created by the user.
2. blogPosts: Blog posts authored by the user.
3. comments: Comments made by the user.
4. reports: Reports submitted by the user.
5. votes: Votes made by the user on blog posts and comments.

2) Template

- Description: Represents templates available for use in blog posts, containing code and explanations.
- Fields:

1. id: Unique identifier for each template (auto-incrementing integer).
2. title: Title of the template.
3. code: The code content of the template.
4. explanation: Optional explanation of the template's purpose.
5. userId: Foreign key referencing the user who created the template.
6. createdAt: Timestamp for when the template was created (default to now).
7. updatedAt: Timestamp for when the template was last updated (automatically updated).

- Relations:

1. user: Relation to the user who created the template.
2. blogPosts: Blog posts that utilize this template.
3. tags: Many-to-many relation to tags associated with this template.

3) Tag

- Description: Represents tags used to categorize blog posts and templates.
- Fields:

1. id: Unique identifier for each tag (auto-incrementing integer).
2. name: Name of the tag (unique).

- Relations:

1. templates: Many-to-many relation to templates.
2. blogPosts: Many-to-many relation to blog posts.

4) BlogPost

- Description: Represents blog posts created by users, containing title, description, and associated metadata.
- Fields:

1. id: Unique identifier for each blog post (auto-incrementing integer).
2. title: Title of the blog post.
3. description: Content description of the blog post.
4. userId: Foreign key referencing the user who created the blog post.
5. hidden: Indicates if the blog post is hidden (default is false).
6. createdAt: Timestamp for when the blog post was created (default to now).
7. updatedAt: Timestamp for when the blog post was last updated (automatically updated).
8. upvotes: Number of upvotes received by the blog post (default is 0).
9. downvotes: Number of downvotes received by the blog post (default is 0).

- Relations:

1. user: Relation to the user who authored the blog post.
2. templates: Templates associated with the blog post.
3. comments: Comments made on the blog post.
4. reports: Reports submitted against the blog post.
5. tags: Many-to-many relation to tags associated with this blog post.
6. votes: Votes cast on this blog post.

5) Comment

- Description: Represents comments made by users on blog posts, allowing for discussion and interaction.
- Fields:

1. id: Unique identifier for each comment (auto-incrementing integer).
2. content: The content of the comment.
3. userId: Foreign key referencing the user who made the comment.
4. blogPostId: Foreign key referencing the blog post to which this comment belongs.
5. parentId: Optional ID of a parent comment for replies (for threaded comments).
6. hidden: Indicates if the comment is hidden (default is false).
7. createdAt: Timestamp for when the comment was created (default to now).
8. updatedAt: Timestamp for when the comment was last updated (automatically updated).
9. upvotes: Number of upvotes received by the comment (default is 0).
10. downvotes: Number of downvotes received by the comment (default is 0).

- Relations:

1. user: Relation to the user who made the comment.
2. blogPost: Relation to the blog post on which the comment was made.
3. reports: Reports submitted against the comment.
4. votes: Votes cast on this comment.

6) Report

- Description: Represents reports submitted by users to flag inappropriate content (either comments or blog posts).
- Fields:

1. id: Unique identifier for each report (auto-incrementing integer).
2. reason: Explanation for why the content is being reported.
3. userId: Foreign key referencing the user who submitted the report.
4. blogPostId: Optional foreign key referencing a blog post being reported.
5. commentId: Optional foreign key referencing a comment being reported.

6. createdAt: Timestamp for when the report was created (default to now).

- Relations:

1. user: Relation to the user who submitted the report.
2. blogPost: Optional relation to the blog post being reported.
3. comment: Optional relation to the comment being reported.

7) Vote

- Description: Represents votes made by users on blog posts and comments to express approval or disapproval.
- Fields:

1. id: Unique identifier for each vote (auto-incrementing integer).
2. type: Type of vote, either "upvote" or "downvote".
3. userId: Foreign key referencing the user who cast the vote.
4. blogPostId: Optional foreign key referencing a blog post being voted on.
5. commentId: Optional foreign key referencing a comment being voted on.
6. createdAt: Timestamp for when the vote was cast (default to now).

- Relations:

1. user: Relation to the user who cast the vote.
2. blogPost: Optional relation to the blog post being voted on.
3. comment: Optional relation to the comment being voted on.

- Constraints:

1. Unique constraint to ensure that each user can only vote once per blog post or comment (combination of userId, blogPostId, and commentId).

USERS API ENDPOINTS

Signup Procedure

- **Description:** This endpoint allows users to sign up by providing their first name, last name, email, and password.
- **URL**: http://localhost:3000/api/users/signup
- **Method**: POST

Payload:
json

```
{
  "firstName": "csc",
  "lastName": "309",
```

```
  "email": "name@csc309.com",
  "password": "password"
}
```

Response:
json

```
{
  "message": "User created successfully",
  "user": {
    "id": 2,
    "email": "name@csc309.com",
    "passwordHash": "<hashed_password>",
    "firstName": "csc",
    "lastName": "309",
    "avatar": null,
    "phoneNumber": null,
    "role": "USER",
    "createdAt": "2024-11-04T22:20:57.945Z",
    "updatedAt": "2024-11-04T22:20:57.945Z"
  }
}
```

Expected Response: A success message with user details including ID, email, passwordHash, and timestamps.

Login Procedure

- **Description:** This endpoint allows users to log in by providing their email and password.
- **URL**: http://localhost:3000/api/users/login
- **Method**: POST

Payload:
json

```
{
  "email": "name@csc309.com",
  "password": "password"
```

}

Response:
json

```
{
  "token": "<your_jwt_token>"
}
```

Expected Response: A JWT token which can be used to authenticate subsequent requests.

Get Profile

- **Description:** Retrieves the profile details of the authenticated user.
- **URL**: http://localhost:3000/api/users/profile
- **Method**: GET
- **Authorization**: Bearer <your_jwt_token>
- **Payload**: None

Response:
json

```
{
  "id": 2,
  "firstName": "csc",
  "lastName": "309",
  "email": "name@csc309.com",
  "avatar": null,
  "phoneNumber": null,
  "createdAt": "2024-11-04T22:20:57.945Z",
  "updatedAt": "2024-11-04T22:20:57.945Z"
}
```

- **Expected Response**: User profile information including ID, firstName, lastName, email, avatar, phoneNumber, and timestamps.

Update Profile

- **Description:** Allows the user to update their profile details such as first name, last name, avatar, and phone number.
- **URL**: http://localhost:3000/api/users/profile
- **Method**: PUT
- **Authorization**: Bearer <your_jwt_token>

Payload:
json

```
{
  "firstName": "John",
  "lastName": "Doe",
  "avatar": "https://example.com/avatar.jpg",
  "phoneNumber": "+1234567890"
}
```

Response:
json

```
{
  "message": "Profile updated successfully",
  "user": {
    "id": 2,
    "email": "name@csc309.com",
    "passwordHash": "<hashed_password>",
    "firstName": "John",
    "lastName": "Doe",
    "avatar": "https://example.com/avatar.jpg",
    "phoneNumber": "+1234567890",
    "role": "USER",
    "createdAt": "2024-11-04T22:20:57.945Z",
    "updatedAt": "2024-11-04T22:31:46.908Z"
  }
}
```

Expected Response: A success message along with the updated user profile.

TEMPLATES API ENDPOINTS

Create Template

- **Description:** Allows users to create a new code template with a title, explanation, tags, and code.
- **URL**: http://localhost:3000/api/templates
- **Method**: POST

Payload:
json

```
{
  "title": "Sample Template",
  "explanation": "This is a test template",
  "tags": ["example", "test"],
  "code": "print('Hello, world!')"
}
```

Response:
json

```
{
  "id": 1,
  "title": "Sample Template",
  "code": "print('Hello, world!')",
  "explanation": "This is a test template",
  "userId": 2,
  "createdAt": "2024-11-04T22:43:58.008Z",
  "updatedAt": "2024-11-04T22:43:58.008Z"
}
```

Search Templates

- **Description:** Search through templates by querying the title or explanation**.**
- **URL**: http://localhost:3000/api/templates/search?query=test
- **Method**: GET

Response:
json


```
[
  {
    "id": 1,
    "title": "Sample Template",
    "code": "print('Hello, world!')",
    "explanation": "This is a test template",
    "userId": 2,
    "createdAt": "2024-11-04T22:43:58.008Z",
    "updatedAt": "2024-11-04T22:43:58.008Z",
    "tags": [
      {
        "id": 1,
        "name": "example"
      },
      {
        "id": 2,
        "name": "test"
      }
    ]
  }
]
```


View All Templates

- **URL**: http://localhost:3000/api/templates
- **Method**: GET

Response:
json


```
[
  {
    "id": 1,
    "title": "Sample Template",
    "code": "print('Hello, world!')",
    "explanation": "This is a test template",
```

```
    "userId": 2,
    "createdAt": "2024-11-04T22:43:58.008Z",
    "updatedAt": "2024-11-04T22:43:58.008Z"
  }
]
```


Update Template

- **Description:** Allows users to update the title, code, explanation, and tags of a specific template.
- **URL**: http://localhost:3000/api/templates/3
- **Method**: PUT

Payload:
json


```
{
  "title": "Updated Title",
  "code": "console.log('Updated Code');",
  "explanation": "Updated explanation",
  "tags": ["updated", "new"]
}
```


Response:
json


```
{
  "id": 3,
  "title": "Updated Title",
  "code": "console.log('Updated Code');",
  "explanation": "Updated explanation",
  "userId": 2,
  "createdAt": "2024-11-04T23:01:18.957Z",
  "updatedAt": "2024-11-04T23:03:54.915Z"
}
```


Delete Template

- **URL**: http://localhost:3000/api/templates/3

- **Method**: DELETE

Response:
json

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

Blog Posts API Endpoints

1. Create a Blog Post

- **URL**: `http://localhost:3000/api/blogs`
- **Method**: `POST`
- **Description**: This endpoint allows authenticated users to create a blog post by providing a title, description, tags, and template IDs. The blog post will be associated with the user who creates it.

Request Body:

```
{
  "title": "New Blog Post",
  "description": "This is the description of the blog post.",
  "tags": ["tag1", "tag2"],
  "templateIds": [1, 2]
}
```

Response:

```
{
  "message": "Blog post created successfully",
  "blogPost": {
```

```
    "id": 1,
    "title": "New Blog Post",
    "description": "This is the description of the blog post.",
    "userId": 2,
    "hidden": false,
    "createdAt": "2024-11-04T23:09:03.676Z",
    "updatedAt": "2024-11-04T23:09:03.676Z",
    "upvotes": 0,
    "downvotes": 0
  }
}
```


2. Update a Blog Post

- **URL**: `http://localhost:3000/api/blogs/1`
- **Method**: `PUT`
- **Description**: Authenticated users can update their existing blog post using this endpoint. Users can modify the title, description, tags, and template associations.

Request Body:


```
{
  "title": "Updated Blog Post Title",
  "description": "This is the updated description of the blog post.",
  "tags": ["updatedTag1", "updatedTag2"],
  "templateIds": [1, 2]
}
```

Response:


```
{
  "message": "Blog post updated successfully",
  "blogPost": {
    "id": 1,
    "title": "Updated Blog Post Title",
    "description": "This is the updated description of the blog post.",
    "userId": 2,
    "hidden": false,
    "createdAt": "2024-11-04T23:09:03.676Z",
    "updatedAt": "2024-11-04T23:19:24.617Z",
    "upvotes": 0,
    "downvotes": 0
  }
}
```


3. Delete a Blog Post

- **URL**: `http://localhost:3000/api/blogs/3`
- **Method**: `DELETE`
- **Description**: Authenticated users can delete their own blog posts using this endpoint.

Response:


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

4. Retrieve Blog Posts Sorted by Date

- **URL**: `http://localhost:3000/api/blogs?search=select`
- **Method**: `GET`
- **Description**: Retrieve all blog posts sorted by creation date, including associated tags and templates.

Response:


```
[
  {
    "id": 2,
    "title": "New Blog Post 2",
    "description": "This is the description of the blog post select.",
    "userId": 2,
    "hidden": false,
    "createdAt": "2024-11-04T23:15:32.622Z",
    "updatedAt": "2024-11-04T23:15:32.622Z",
    "upvotes": 0,
    "downvotes": 0,
    "user": {
      "firstName": "John",
      "lastName": "Doe",
      "avatar": "https://example.com/avatar.jpg"
    },
    "tags": [
      {
```

```
    "id": 3,
    "name": "tag1"
  },
  {
    "id": 4,
    "name": "tag2"
  }
],
"templates": [
  {
    "id": 1,
    "title": "Sample Template",
    "code": "print('Hello, world!')",
    "explanation": "This is a test template",
    "userId": 2,
    "createdAt": "2024-11-04T22:43:58.008Z",
    "updatedAt": "2024-11-04T22:43:58.008Z"
  },
  {
    "id": 2,
    "title": "Sample Template 2",
    "code": "print('Hello there!')",
    "explanation": "This is a test template 2",
```

```
      "userId": 2,
      "createdAt": "2024-11-04T22:54:18.632Z",
      "updatedAt": "2024-11-04T22:54:18.632Z"
    }
  ]
 }
]
```


5. Retrieve Blog Posts Sorted by Rating

- **URL**: `http://localhost:3000/api/blogs?sort=rating`
- **Method**: `GET`
- **Description**: Retrieve blog posts sorted by their rating (upvotes minus downvotes)

Votes API Endpoints

1. Vote on a Blog Post or Comment

- **URL**: `http://localhost:3000/api/votes`
- **Method**: `POST`
- **Description**: This endpoint allows users to upvote or downvote a blog post or comment. The vote can be toggled, meaning the vote is removed if the user votes again for the same item.

Request Body (Upvote Blog Post):


```
{
  "id": 1,
  "type": "upvote",
```

```
  "itemType": "blogPost"
}
```


Response (Vote Added):


```
{
  "message": "Voted upvote"
}
```


Response (Vote Removed on Second Click):


```
{
  "message": "upvote removed"
}
```


Request Body (Downvote Blog Post):


```
{
  "id": 2,
  "type": "downvote",
  "itemType": "blogPost"
}
```


Response:


```
{
  "message": "Voted downvote"
}
```


Comments API Documentation

Create Comment

- **URL**: `http://localhost:3000/api/comments/create`
- **Method**: `POST`
- **Description**: This endpoint allows authenticated users to create a new comment for a specific blog post. Users can optionally provide a `parentId` if the comment is a reply to another comment.

Request Body:


```
{
  "content": "This is a comment.",
  "blogPostId": 1,
  "parentId": null
}
```


Response:


```
{
  "message": "Comment created successfully",
  "comment": {
    "id": 1,
    "content": "This is a comment.",
    "userId": 2,
    "blogPostId": 1,
    "parentId": null,
    "createdAt": "2024-11-04T23:09:03.676Z",
    "updatedAt": "2024-11-04T23:09:03.676Z"
  }
```

}

Create Comment to another Comment

Request:

{
  "content": "This is the second comment.",
  "blogPostId": 1,
  "parentId" : 1
}

Response:

{
    "message": "Comment created successfully",
    "comment": {
        "id": 2,
        "content": "This is the second comment.",
        "userId": 2,
        "blogPostId": 1,
        "parentId": 1,
        "hidden": false,
        "createdAt": "2024-11-05T00:27:52.015Z",
        "updatedAt": "2024-11-05T00:27:52.015Z",
        "upvotes": 0,

```
    "downvotes": 0
  }
}
```


Search Comments

- **URL**: `http://localhost:3000/api/comments/search`
- **Method**: `GET`
- **Description**: This endpoint retrieves comments based on search criteria. Users can filter comments by content, specify pagination through `page` and `limit` query parameters, and sort by date or rating.

Query Parameters:

- `page`: The page number for pagination (default: 1)
- `limit`: The number of comments to return per page (default: 10)
- `search`: A search term to filter comments by content (optional)
- `sort`: The sorting criterion (`date` or `rating`, default is `date`)

Response:


```
[
 {
  "id": 1,
  "content": "This is a comment.",
  "userId": 2,
  "blogPostId": 1,
  "parentId": null,
  "createdAt": "2024-11-04T23:09:03.676Z",
```

```
    "user": {
      "firstName": "John",
      "lastName": "Doe",
      "avatar": "https://example.com/avatar.jpg"
    }
  }
]
```


Update Comment

- **URL**: `http://localhost:3000/api/comments/{id}/update`
- **Method**: `PUT`
- **Description**: This endpoint allows authenticated users to update the content of their own comment. Users can only update comments they have created.

Request Body:


```
{
  "content": "This is the updated comment."
}
```


Response:


```
{
  "message": "Comment updated successfully",
  "comment": {
    "id": 1,
    "content": "This is the updated comment.",
```

```
    "userId": 2,
    "blogPostId": 1,
    "parentId": null,
    "createdAt": "2024-11-04T23:09:03.676Z",
    "updatedAt": "2024-11-04T23:19:24.617Z"
  }
}
```


Delete Comment

- **URL**: `http://localhost:3000/api/comments/{id}/delete`
- **Method**: `DELETE`
- **Description**: This endpoint allows authenticated users to delete their own comments. Users can only delete comments they have created.

Response:


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


Execute Code API

Description

This API endpoint allows users to execute a snippet of code in a specified programming language. It accepts the code, language, and any required input as parameters and returns the output or an error message based on the execution. This endpoint is useful for dynamically evaluating code snippets across various languages, including C, C++, Java, Python, and JavaScript.

- URL: `http://localhost:3000/api/code/executeCode`
- Method: POST

Request Body

- Code (string): The code snippet that the user wants to execute.
- Language (string): The programming language of the provided code (e.g., "c", "cpp", "java", "python", "javascript").
- Input (string): The inputs required by the code during execution, if any.

```
{

 "code": "#include <stdio.h>\nint main() { char input[100]; fgets(input,
sizeof(input), stdin); printf(\"Hello, %s!\\n\", input); return 0; }",

 "language": "c",

 "input": "C Programmer"

}
```

Response

The response will show the output of the user's code returned as a string.

```
{

   "output": "Hello, C Programmer\n!\n"

}
```

Report Posts API Endpoints

Create Report

- **URL**: `http://localhost:3000/api/reports/create`
- **Method**: `POST`
- **Description**: Allows an authenticated user to report an inappropriate blog post or comment with an explanation/reason

Request Body:

{

"reason": "Inappropriate content",

```
  "blogPostId": 1,
  "commentId": null
}
```


Response:


```
{
    "message": "Report submitted successfully",
    "report": {
        "id": 2,
        "reason": "Inappropriate content",
        "userId": 6,
        "blogPostId": 1,
        "commentId": null,
        "createdAt": "2024-11-05T00:57:33.353Z"
    }
}
```


View Report

- **URL**: `http://localhost:3000/api/reports`
- **Method**: `GET`
- **Description**: Allows an administrator to view a list of all reported blog posts and comments, sorted by the number of reports
- **Authorization**: Bearer Token (Admin access required)

Response:


```
{
    "reportedPosts": [
```

```
{
    "id": 1,
    "title": "BLOGPOST TITLE",
    "description": "BLOGPOST DESCRIPTION",
    "userId": 1,
    "createdAt": "2024-11-03T23:48:04.084Z",
    "updatedAt": "2024-11-03T23:47:37.866Z",
    "upvotes": 0,
    "downvotes": 0,
    "reports": [
        {
            "id": 1,
            "reason": "Inappropriate content",
            "userId": 6,
            "blogPostId": 1,
            "commentId": null,
            "createdAt": "2024-11-04T02:47:43.948Z"
        },
        {
            "id": 2,
            "reason": "Inappropriate content",
            "userId": 6,
            "blogPostId": 1,
```

```
            "commentId": null,
            "createdAt": "2024-11-05T00:57:33.353Z"
          }
        ],
        "user": {
          "firstName": "Daniel",
          "lastName": "Lee",
          "avatar": null
        }
      }
    ],
    "reportedComments": []
}
```


Expected Response: A list of reported blog posts and comments, including their reports and other metadata.

Hide Report

- **URL**: `http://localhost:3000/api/reports/hide`
- **Method**: `PATCH`
- **Description**: Allows an administrator to hide a reported blog post or comment, making it invisible to users except the author.
- **Authorization:** Bearer Token (Admin access required)

Request Body:


```
{
  "blogPostId": 1
```

}

Response:

{

“message”: “Content hidden successfully”

}

Expected Response: A success message indicating that the content has been hidden