CSC309 - PP1 Documentation

Isabella Nguyen (1009051482), Alessia Ruberto (1008910680), Erin Kim (1008933950)

November 5, 2024

Admin Credentials

User: "admin"
Pass: "admin"

Code Writing And Execution

API End Point: api/code-writing-and-execution/execute-code (POST only)
Description: Execute code in various programming languages with optional standard input. Supported languages include Python, Java, C, JavaScript, and C++.
Parameters:

- **code** (string, required): The code to execute as plaintext.
- **language** (string, required): The language to execute the code in.
- **stdin** (string, optional): The input to pass to the code.

Example Request:

```
{
    "language": "python",
    "stdin": "Hellooooo",
    "code": "user_input = input()  # Print the input back\nprint(f'{user_input} from python!')"
}
```

Example Response:

```
{
    "output": "Hellooooo from python!\r\n"
}
```

Accounts

Sign Up

API End Point: api/user/register (POST only)
Description: This API endpoint is used for user registration. It allows clients to create a new user by providing necessary details.
Parameters:

- **username** (string, required): The unique username for the new user.
- **password** (string, required): The password for the new user.

- **firstName** (string, optional): The first name of the user.
- **lastName** (string, optional): The last name of the user.
- **email** (string, required): The email address of the user.
- **avatar** (integer, optional): The avatar ID (defaults to 1 if not provided).
- **role** (string, optional): The role of the user (defaults to "USER" if not provided).

Example Request:

```
{
  "username": "newuser",
  "password": "securePassword123",
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "avatar": 2,
  "role": "ADMIN"
}
```

Example Responses:
successful:

```
// HTTP/1.1 200 OK
{
  "id": 1,
  "username": "newuser",
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "avatar": 2,
  "role": "ADMIN",
  "avatarUrl": "/avatars/2.png"
}
```

required fields are missing:

```
//HTTP/1.1 400 Bad Request
{
  "message": "Please provide all the required fields"
}
```

username already exists:

```
//HTTP/1.1 400 Bad Request
{
  "error": "Username already exists"
}
```

Notes:
The avatar field is optional; if not provided, it defaults to 1.
The role defaults to "USER" if not specified.
Passwords are hashed before being stored in the database for security.

Login

API End Point: POST /api/user/login
Description: This API endpoint is used for user login. It allows clients to authenticate users by validating their credentials and generating access and refresh tokens.
Parameters:

- **username** (string, required): The username of the user.
- **password** (string, required): The password of the user.

Example Request:

```
{
  "username": "existinguser",
  "password": "correctPassword123"
}
```

Example Response:
success:

```
//HTTP/1.1 200 OK
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Invalid username or password:

```
// HTTP/1.1 401 Unauthorized
{
  "message": "Invalid credentials"
}
```

Required fields are missing:

```
// HTTP/1.1 400 Bad Request
{
  "message": "Please provide all the required fields"
}
```

Notes:
The access token is used for authenticating subsequent requests, while the refresh token can be used to obtain a new access token when the original expires.

Refreshing Access Token

API End Point: POST /api/user/refresh
Description: This endpoint is used to refresh an access token by validating a provided refresh token. If the refresh token is valid, a new access token is generated and returned.
Parameters:

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

Example Request:

```
{
  "refreshToken": "dfkjew423434iu8rekjgenrgldfgmnsdghstehs34y65k"
}
```

Example Response:
success:

```
// HTTP/1.1 200 OK
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Refresh token missing:

```
// HTTP/1.1 400 Bad Request
{
  "error": "Refresh token missing"
}
```

Invalid refresh token:

```
// HTTP/1.1 401 Unauthorized
{
  "message": "Invalid or expired refresh token"
}
```

Editing Profile

API End Point: PUT /api/user/userId
Description: This file defines an API endpoint that allows users to edit their profiles. User can update their personal information, including first name, last name, email, and avatar.
Parameters:

- URL Parameters
  - id (integer, required): The unique identifier of the user whose profile is being updated.
- Request Headers
  - Authorization (string, required): A bearer token used to authorize the request.
- Parameters
  - **firstName** (string, optional): The new first name for the user.
  - **lastName** (string, optional): The new last name for the user.
  - **email** (string, optional): The new email address for the user.
  - **avatar** (integer, optional): The new avatar ID for the user.

Example Request:

```
{
  "firstName": "Jane",
  "lastName": "Doe",
  "email": "jane.doe@example.com",
  "avatar": 2
}
```

Example Responses:
success

```
//HTTP/1.1 200 OK
{
  "message": "Profile updated successfully",
  "user": {
    "id": 1,
    "firstName": "Jane",
    "lastName": "Doe",
    "email": "jane.doe@example.com",
    "avatar": 2,
    "avatarUrl": "/avatars/2.png"
  }
}
```

Access token invalid or expired:

```
// HTTP/1.1 401 Unauthorized
{
  "message": "Invalid or expired access token"
}
```

Code Templates

Creating template, Searching through templates
index.js Overview:
This file defines an API endpoint for managing code templates. It handles the creation of new templates, listing all templates, and searching templates by title, explanation, or tags.

Creating Template

API End Point: POST /api/templates
Description: Create a new code template with a title, code content, and optional tags and explanation.

- Request Headers
  - Authorization (string, required): A bearer token used to authorize the request.
- Parameters
  - **title** (string, required): The title of the code template.
  - **explanation** (string, optional): A description or explanation of the template.
  - **codeContent** (string, required): The actual code content to be saved.
  - **tags** (array of strings, optional): An array of tags associated with the template.
  - **language** (string, required): The programming language of the code content.

Example Request:

```
{
  "title": "My First Template",
  "explanation": "This is a simple template.",
  "codeContent": "console.log('Hello, world!');",
  "tags": ["JavaScript", "HelloWorld"],
  "language": "JavaScript"
}
```

Example Responses:
success

```
// HTTP/1.1 200 OK
{
  "message": "Template created successfully.",
  "template": {
    "id": 1,
    "title": "My First Template",
    "explanation": "This is a simple template.",
    "authorId": 123,
    "codeId": 456,
    "tags": [
      { "id": 1, "name": "JavaScript" },
      { "id": 2, "name": "HelloWorld" }
    ],
    "code": { "filePath": "/path/to/code.js", "language": "JavaScript" }
  }
}
```

auth token missing:

```
//HTTP/1.1 401 Unauthorized
{
  "message": "Login required to create template."
}
```

Template Search

API End Point: GET /api/templates
Description: Retrieve a list of code templates, optionally filtered by a search query and paginated.
Parameters:

- **query** (string, optional): A search term to filter templates by title, explanation, or tags.
- **page** (integer, optional): The page number for pagination (default is 1).
- **limit** (integer, optional): The number of templates per page (default is 10).

Example Request:

```
GET /api/templates?query=JavaScript&page=1&limit=10
```

Example Response:

```
//HTTP/1.1 200 OK
[
  {
    "id": 1,
    "title": "My First Template",
    "explanation": "This is a simple template.",
    "authorId": 123,
    "codeId": 456,
    "tags": [
      { "id": 1, "name": "JavaScript" },
      { "id": 2, "name": "HelloWorld" }
    ],
    "code": { "filePath": "/path/to/code.js", "language": "JavaScript" }
  },
  // ...other templates
]
```

Note: here, your account would have userId of 123.

Editing code template

API End Point: PUT /api/templates/templateId
Description: Updates an existing code template. Note: only the owner of the template can edit their code templates.
Parameters:

- URL Parameters
  - templateId (integer, required): The unique identifier of the template to update.
- Request Headers
  - Authorization (string, required): A bearer token used to authorize the request.
- Parameters
  - **title** title (string, required): The new title of the template.
  - **explanation** (string, optional): The new description of the template.
  - **codeContent** (string, required): The updated code content to be saved.
  - **tags** (integer, optional): An array of tags associated with the template.
  - **language** (integer, required): The programming language of the code.

Example Request:

```
{
  "title": "Updated Template",
  "explanation": "Updated explanation.",
  "codeContent": "console.log('Updated code.');",
  "tags": ["JavaScript", "Updated"],
  "language": "JavaScript"
}
```

Example Response:
success:

```
//HTTP/1.1 200 OK
[
  {
    "id": 1,
    "title": "My First Template",
    "explanation": "This is a simple template.",
    "authorId": 123,
    "codeId": 456,
    "tags": [
      { "id": 1, "name": "JavaScript" },
      { "id": 2, "name": "HelloWorld" }
    ],
    "code": { "filePath": "/path/to/code.js", "language": "JavaScript" }
  },
  // ...other templates
]
```

not author:

```
//HTTP/1.1 401 Unauthorized
{
  "message": "Only the authors are allowed to edit templates. Please fork instead."
}
```

Deleting code template

API End Point: DELETE /api/templates/templateId
Description: Deletes a code template by its ID. The user must be authenticated to delete a template.
Parameters:

- URL Parameters
  - templateId (integer, required): The unique identifier of the template to update.
- Request Headers
  - Authorization (string, required): A bearer token used to authorize the request.

Example Request:

```
DELETE /api/templates/1
```

Example Responses:
success:

```
HTTP/1.1 204 No Content
```

not the author

```
//HTTP/1.1 401 Unauthorized
{
  "message": "Only the authors are allowed to delete templates."
}
```

unauthorized

```
//HTTP/1.1 401 Unauthorized
{
  "message": "Authentication required to delete template, please login."
}
```

template not found

```
//HTTP/1.1 404 Not Found
{
  "message": "Template for deletion not found."
}
```

Fork code template

API End Point: POST /api/templates/templateId

Description: Creates a fork of an existing code template. The language of the code cannot be changed when forking. Only users can fork templates, visitors receive a message to login or create account.
Note: you cannot edit the tags when forking
Parameters:

- URL Parameters
  - id (integer, required): The unique identifier of the template to fork.
- Request Headers
  - Authorization (string, required): A bearer token used to authorize the request.
- Parameters
  - **title** (string, optional): The new title of the forked template. Has a default title, so is not required
  - **explanation** (string, optional): The explanation for the forked template. Has a default explanation.
  - **codeContent** (string, required): The code content to be saved in the forked template.

Example Request:

```
{
  "title": "Forked Template",
  "explanation": "This is a forked template.",
  "codeContent": "console.log('Forked code.');"
}
```

Example Response:
success:

```
// HTTP/1.1 201 Created
{
  "message": "Template forked successfully.",
  "template": {
    "id": 2,
    "title": "Forked Template",
    "explanation": "This is a forked template.",
    "authorId": 456,
    "codeId": 789,
    "tags": [],
    "isForked": true
  }
}
```

not a user

```
// HTTP/1.1 401 Unauthorized
{
  "message": "Please create an account to save the forked template."
}
```

Missing required fields

```
//HTTP/1.1 400 Bad Request
{
  "message": "Please provide all the required fields for forking."
}
```

Blogs

API End Point: POST /api/blogs
Description: Allow users to post blogs with links to code templates
Parameters:

- Header: Authorization - Bearer token for user authentication.
- Body:
  - title - string: Title of the blog post.
  - description - string: Brief description of the blog post.
  - content - string: Main content of the blog post.
  - tags - array of strings: Tags for categorization (e.g., ["JavaScript", "Tutorial"]).
  - codeTemplates - array of strings (URLs): URLs of code templates to be linked with the post.

Example Request:

```
{
  "title": "Very exciting blog post",
  "description": "Yadaya",
  "content": "Coding is so fun (this project was so much work and its only half)",
  "tags": ["java", "dynamic programming", "slay"],
  "codeTemplates": ["http://localhost:3000/api/templates/1"]
}
```

Example Response:
Successful

```
{
    "id": 6,
    "title": "Very exciting blog post",
    "createdAt": "2024-11-04T19:38:15.462Z",
    "authorId": 2,
    "description": "Yadaya",
    "content": "Coding is so fun (this project was so much work and its only half)",
    "ratingScore": 0,
    "hidden": false,
    "tags": [
        {
            "name": "dynamic programming"
        },
        {
            "name": "java"
        },
        {
            "name": "slay"
        }
    ],
    "codeTemplate": [
        {
            "id": 1,
            "title": "my first template",
            "explanation": "this is a test template for demonstration.",
            "codeId": 1,
            "authorId": 1,
            "createdAt": "2024-11-04T16:25:47.987Z",
            "updatedAt": "2024-11-04T16:25:47.987Z",
            "isForked": false
        }
    ]
}
```

Error: Visitor (not signed in)

```
{
    "error": "Please sign in to create a blog post"
}
```

Error: Empty Required fields
Example Request:

```
{
  "title": "Very exciting blog post",
  "description": "Yadaya",
  "content": "",
  "tags": ["java", "dynamic programming", "slay"],
  "codeTemplates": ["http://localhost:3000/api/templates/1"]
}
```

Example Response:

```
{
    "error": "Blog Posts must have a title, description, and valid content."
}
```

Failure - User tries to edit hidden content
Example Response:
API End Point: GET /api/blogs
Description: Retrieve a list of visible blog posts, optionally filtered by a search query matching titles, content, tags, or associated code templates. Results are sorted by rating score in descending order.
Parameters:

- **Query Parameters:**
  - **paramName** - *string* (optional): A search term to filter blog posts by matching title, content, tags, or code template titles/explanations.

Example Request:

```
http://localhost:3000/api/blogs/?query=java
```

Example Response:

```
[
    {
        "id": 5,
        "title": "Very exciting blog post",
        "createdAt": "2024-11-04T19:36:13.259Z",
        "authorId": 2,
        "description": "Yadaya",
        "content": "Coding is so fun (this project was so much work and its only half)",
        "ratingScore": 0,
        "hidden": false,
        "rating": [],
        "tags": [
            {
                "name": "dynamic programming"
            },
            {
                "name": "java"
            },
            {
                "name": "slay"
            }
        ],
        "codeTemplate": [
            {
                "id": 1,
                "title": "my first template",
                "explanation": "this is a test template for demonstration.",
                "codeId": 1,
                "authorId": 1,
                "createdAt": "2024-11-04T16:25:47.987Z",
                "updatedAt": "2024-11-04T16:25:47.987Z",
                "isForked": false
            }
        ]
    }
```

API End Point: PUT /api/blogs/id
Description: Updates an existing blog post with a new title, description, content, tags, and code templates. The user must be authenticated and must be the author of the blog post to make changes. Content cannot be edited if the post is hidden.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Path Parameter:** *id* - Integer, the ID of the blog post to update.
- **Body Parameters:**
  - **title** - *string* (optional): New title of the blog post.
  - **description** - *string* (optional): New description of the blog post.
  - **content** - *string* (optional): Updated content of the blog post.
  - **newTemplates** - *array of strings* (optional): URLs of code templates to associate with the blog post.

  - **newTags** - *array of strings* (optional): Tags to categorize the blog post.

Successful
Example Request:

```
{
  "title": "Updated Blog Title",
  "description": "An updated description of the blog post.",
  "content": "Here is the new content of the blog post.",
  "newTemplates": ["http://localhost:3000/api/templates/1"],
  "newTags": ["JavaScript", "Node.js", "Prisma"]
}
```

Example Response:

```
{
    "message": "Blog updated successfully",
    "updatedBlog": {
        "id": 1,
        "title": "Updated Blog Title",
        "createdAt": "2024-11-04T01:45:10.089Z",
        "authorId": 1,
        "description": "An updated description of the blog post.",
        "content": "Here is the new content of the blog post.",
        "ratingScore": 0,
        "hidden": false
    }
}
```

Failure - User tries to edit a blog that doesn't belong to them
Example Request:

```
    {
  "title": "Updated Blog Title",
  "description": "An updated description of the blog post.",
  "content": "Here is the new content of the blog post.",
  "newTemplates": ["http://localhost:3000/api/templates/20"],
  "newTags": ["JavaScript", "Node.js", "Prisma"]
}
```

Example Response:

```
   {
    "error": "You do not have access to edit this blog post"
}
```

API End Point: DELETE /api/blogs/id
Description: Deletes an existing blog post. Only the author of the blog post, verified through authentication, is permitted to delete it.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Path Parameter:** *id* - Integer, the ID of the blog post to delete.

Example Response:

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

Comments

API End Point: GET /api/comments
Description: Retrieves a paginated list of top-level comments for a specific blog post. Each comment includes its replies, sorted by rating score in descending order.
Parameters:

- **Query Parameters:**
  - **blogPostId** - *integer* (required): The ID of the blog post for which to fetch comments.
  - **page** - *integer* (optional): The page number for pagination. Default is 1.
  - **limit** - *integer* (optional): Number of comments to retrieve per page. Default is 10.

API End Point: POST /api/comments?blogPostId={blogPostId}
Description: Allows authenticated users to post a comment on a specified blog post. The comment can be a top-level comment or a reply to another comment.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Query Parameter:** *blogPostId* - *integer* (required): The ID of the blog post to which the comment will be added.
- **Body Parameters:**
  - **content** - *string* (required): The content of the comment.
  - **parentId** - *integer* (optional): The ID of the parent comment if this comment is a reply.

Example Request:

```
{
    "content": "so lame",
    "authorId": 1,
    "parentId": null
}
```

Example Response:

```
{
    "message": "Comment added successfully",
    "comment": {
        "id": 6,
        "content": "so lame",
        "authorId": 2,
        "blogPostId": 2,
        "parentId": null,
        "ratingScore": 0
    }
}
```

Failure: Visitor tries to leave a comment
Example Request:

```
{
    "content": "so lame",
    "authorId": 1,
    "parentId": null
}
```

Example Response:

```
{
    "error": "Please log in to post a comment"
}
```

API End Point: GET /api/comments/id
Description: Retrieves a specific comment by ID, including user information and any replies with their respective users.
Parameters:

- **Path Parameter:** *id* - *integer* (required): The ID of the comment to retrieve.

API End Point: PUT /api/comments/id
Description: Updates the content of a specific comment. Only the content field can be modified, and the request should be authorized.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Path Parameter:** *id* - *integer* (required): The ID of the comment to update.
- **Body Parameters:**
  - **content** - *string* (required): The new content for the comment.

Example Request:

```
{
    "content": "this is an updated comment"
}
```

Example Response:

```
{
    "message": "Comment updated successfully",
    "updated": {
        "id": 1,
        "content": "this is an updated comment",
        "authorId": 1,
        "blogPostId": 2,
        "parentId": null,
        "ratingScore": 0
    }
}
```

API End Point: DELETE /api/comments/id
Description: Deletes a specific comment by ID. This action requires authorization.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Path Parameter:** *id* - *integer* (required): The ID of the comment to delete.

Example Response:

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

Ratings

API End Point: POST /api/ratings
Description: Allows authenticated users to rate a specific blog post or comment. Users can upvote or downvote, and the rating is adjusted if the user changes their vote.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Body Parameters:**
  - **id** - *integer* (required): The ID of the content (either a blog post or a comment) to be rated.
  - **contentType** - *string* (required): The type of content being rated. Valid values are ‘"BlogPost"‘ or ‘"Comment"‘.
  - **vote** - *integer* (required): The rating value, ‘1‘ for an upvote or ‘-1‘ for a downvote.

Example Request:

```
{
    "id": 3,
    "contentType": "Comment",
    "vote": -1
}
```

Example Response:

```
{
    "message": "Rating has been processed"
}
```

User - Report Inappropriate Content

API End Point: POST /api/reports
Description: Allows authenticated users to report inappropriate content, either a blog post or a comment. Each user can only submit one report per content item.
Parameters:

- **Header:** Authorization - Bearer token for user authentication.
- **Body Parameters:**
  - **contentId** - *integer* (required): The ID of the content (either a blog post or a comment) being reported.
  - **reason** - *string* (required): The reason for reporting the content.
  - **contentType** - *string* (required): The type of content being reported. Valid values are ‘"BlogPost"‘ or ‘"Comment"‘.

Example Request:

```
{
  "contentId": 6,
  "reason": "outdated",
  "contentType": "Comment"
}
```

Example Response:

```
{
    "message": "Comment has been reported successfully",
    "report": {
        "id": 1,
        "reporterId": 2,
        "reason": "outdated",
        "commentId": 6,
        "createdAt": "2024-11-04T18:01:51.041Z"
    }
}
```

Admin - Inappropriate Content Management

API End Point: GET /api/admin/inappropriate-blogs
Description: Allows admin users to retrieve a list of blog posts ordered by the number of reports, from most to least reported.
Parameters:

- **Header:** Authorization - Admin-level authentication is required.

Example Response:

```
[
    {
        "id": 2,
        "title": "Very exciting blog post",
        "createdAt": "2024-11-04T01:53:44.088Z",
        "authorId": 1,
        "description": "Yadaya",
        "content": "Coding is so fun (this project was so much work and its only half)",
        "ratingScore": 0,
        "hidden": true,
        "_count": {
            "reports": 6
        }
    },
    {
        "id": 3,
        "title": "Very exciting blog post",
        "createdAt": "2024-11-04T01:54:10.967Z",
        "authorId": 1,
        "description": "Yadaya",
        "content": "Coding is so fun (this project was so much work and its only half)",
        "ratingScore": 0,
        "hidden": false,
        "_count": {
            "reports": 0
        }
    },
    {
        "id": 4,
        "title": "Very exciting blog post",
        "createdAt": "2024-11-04T16:00:45.731Z",
        "authorId": 1,
        "description": "Yadaya",
        "content": "Coding is so fun (this project was so much work and its only half)",
        "ratingScore": -1,
        "hidden": false,
        "_count": {
            "reports": 0
        }
    }
]
```

Failure: Non-Admin tries to view reported content
Example Response

```
{
    "error": "Forbidden"
}
```

API End Point: PUT /api/admin/inappropriate-blogs
Description: Allows admin users to mark a specific blog post as hidden, making it no longer visible to regular users.
Parameters:

- **Header:** Authorization - Admin-level authentication is required.
- **Body Parameters:**
  - **blogId** - *integer* (required): The ID of the blog post to be hidden.

Example Request:

```
{
    "blogId": 2
}
```

Example Response:

```
{
    "message": "Content hidden successfully"
}
```

API End Point: GET /api/admin/inappropriate-comments
Description: Allows admin users to retrieve a list of comments ordered by the number of reports, from most to least reported. This endpoint is used to manage potentially inappropriate comments.
Parameters:

- **Header:** Authorization - Admin-level authentication is required.

Example Response:

```
[
    {
        "id": 6,
        "content": "so lame",
        "authorId": 2,
        "blogPostId": 2,
        "parentId": null,
        "ratingScore": 0,
        "_count": {
            "reports": 1
        }
    },
    {
        "id": 2,
        "content": "so lame",
        "authorId": 1,
        "blogPostId": 2,
        "parentId": null,
        "ratingScore": 0,
        "_count": {
            "reports": 0
        }
    },
    {
        "id": 3,
        "content": "so lame",
        "authorId": 1,
        "blogPostId": 2,
        "parentId": null,
        "ratingScore": 0,
        "_count": {
            "reports": 0
        }
    },
]
```

API End Point: PUT /api/admin/inappropriate-comments
Description: Allows admin users to mark a specific comment as hidden, making it no longer visible to regular users.
Parameters:

- **Header:** Authorization - Admin-level authentication is required.
- **Body Parameters:**
  - **commentId** - *integer* (required): The ID of the comment to be hidden.

Example Request:

```
{
    "commentId": 3
}
```

Example Response:

```
{
    "message": "Content hidden successfully"
}
```

Model

```
generator client {
  provider = "prisma-client-js"
```

```
}

datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

model User {
  id            Int          @id @default(autoincrement())
  username      String       @unique
  password      String
  firstName     String       @default("")
  lastName      String       @default("")
  email         String       @unique
  avatar        Int
  role          String       @default("USER")
  createdAt     DateTime     @default(now())
  comments      Comment[]
  blogPosts     BlogPost[]
  codeTemplates CodeTemplate[]
  ratings       Rating[]
  commentReports CommentReport[]
  BlogReports    BlogReport[]
}

model BlogPost {
  id              Int          @id @default(autoincrement())
  title           String
  createdAt       DateTime     @default(now())
  authorId        Int
  description     String
  author          User         @relation(fields: [authorId], references: [id], onDelete: Cascade)
  content         String
  codeTemplate    CodeTemplate[]
  ratingScore     Int @default(0)
  rating          Rating[]
  reports         BlogReport[]
  hidden          Boolean      @default(false)
  comments        Comment[]
  tags            Tag[]        @relation("BlogPostTags")
}

model CommentReport {
  id          Int       @id @default(autoincrement())
  reporter    User      @relation(fields: [reporterId], references: [id])
  reporterId  Int
  reason      String
  commentId   Int
  comment     Comment    @relation(fields: [commentId], references: [id])
  createdAt   DateTime   @default(now())
}

model BlogReport {
  id          Int       @id @default(autoincrement())
  reporter    User      @relation(fields: [reporterId], references: [id])
  reporterId  Int
  reason      String
  blogId      Int
  blog        BlogPost   @relation(fields: [blogId], references: [id])
  createdAt   DateTime   @default(now())
}

model CodeTemplate {
  id          Int          @id @default(autoincrement())
  title       String
  explanation String
  code        Code         @relation(fields: [codeId], references: [id])
  codeId      Int          @unique
  tags        Tag[]        @relation("TemplateTags")
  author      User         @relation(fields: [authorId], references: [id])
  authorId    Int
  blogs       BlogPost[]
  createdAt   DateTime     @default(now())
  updatedAt   DateTime     @updatedAt
  isForked    Boolean      @default(false)
}

model Comment {
  id           Int       @id @default(autoincrement())
  content      String
  author       User      @relation(fields: [authorId], references: [id])
  authorId     Int
  blogPostId   Int
  parentId     Int?
  ratingScore  Int @default(0)
  blogPost     BlogPost @relation(fields: [blogPostId], references: [id])
  parent       Comment?  @relation("ParentComment", fields: [parentId], references: [id])
  replies      Comment[] @relation("ParentComment")
  reports      CommentReport[]
  ratings      Rating[]
  hidden       Boolean    @default(false)
  @@index([parentId])
}

model Rating {
  id          Int       @id @default(autoincrement())
  user        User      @relation(fields: [userId], references: [id])
  userId      Int
  blogPostId  Int?
  blogPost    BlogPost? @relation(fields: [blogPostId], references: [id])
  commentId   Int?
  comment     Comment?  @relation(fields: [commentId], references: [id])
  rating      Int       // + 1 for upvotes, -1 for downvotes

  // ensure ratings can only be done once for a blog or a comment
  @@unique([userId, blogPostId])
  @@unique([userId, commentId])
}
```

```
model Tag {
  name            String            @id
  codeTemplates   CodeTemplate[]    @relation("TemplateTags")
  blogs           BlogPost[]        @relation("BlogPostTags")
}

model Code {
  id        Int      @id @default(autoincrement())
  filePath  String
  language  String
  codeTemplate  CodeTemplate?
  createdAt DateTime  @default(now())
}
```