## Model design

```
model User {
  id            Int            @id @default(autoincrement())
  firstname     String
  lastname      String
  email         String         @unique
  password      String
  role          String         @default("User")
  avatar        String?
  createdAt     DateTime       @default(now())
  updatedAt     DateTime       @updatedAt
  banned        Boolean        @default(false)
  // Relations
  codeTemplates CodeTemplate[]
  blogPosts     BlogPost[]
  comments      Comment[]
  reports       Report[]

  Rating Rating[]
}

model CodeTemplate {
  id          Int        @id @default(autoincrement())
  title       String
  description String
  tags        String
  code        String
  language    String
  isForked    Boolean    @default(false)
  author      User       @relation(fields: [authorId], references: [id])
  authorId    Int
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt
  // Relations
  blogPosts   BlogPost[]
}

model BlogPost {
  id           Int       @id @default(autoincrement())
  title        String
  description  String
  content      String
  tags         String
  createdAt    DateTime  @default(now())
  updatedAt    DateTime  @updatedAt
  hidden       Boolean   @default(false) // New field to indicate if a post is hidden

  // Relations
  user          User           @relation(fields: [userId], references: [id])
  userId        Int
  codeTemplates CodeTemplate[] // Blog post can reference multiple code templates
  comments      Comment[]
  ratings       Rating[]
```

```
  Report        Report[]
}

model Comment {
  id              Int       @id @default(autoincrement())
  content         String
  author          User      @relation(fields: [authorId], references: [id])
  authorId        Int
  blogPost        BlogPost  @relation(fields: [blogPostId], references: [id])
  blogPostId      Int
  parentComment   Comment?  @relation("CommentReplies", fields: [parentCommentId], references: [id])
  parentCommentId Int?
  replies         Comment[] @relation("CommentReplies")
  createdAt       DateTime  @default(now())
  updatedAt       DateTime  @updatedAt

  Rating Rating[]

  Report Report[]
}

model Report {
  id              Int       @id @default(autoincrement())
  reason          String
  additionalInfo  String?
  user            User      @relation(fields: [userId], references: [id])
  userId          Int
  blogPost        BlogPost? @relation(fields: [blogPostId], references: [id])
  blogPostId      Int?
  comment         Comment?  @relation(fields: [commentId], references: [id])
  commentId       Int?

  @@unique([blogPostId, commentId]) // Ensures that a report targets either a BlogPost or a Comment, no
}

model Rating {
  id        Int      @id @default(autoincrement())
  value     Int // 1 for upvote, -1 for downvote
  createdAt DateTime @default(now())

  // Relations
  user       User      @relation(fields: [userId], references: [id])
  userId     Int
  blogPost   BlogPost? @relation(fields: [blogPostId], references: [id])
  blogPostId Int?
  comment    Comment?  @relation(fields: [commentId], references: [id])
  commentId  Int?
}
```

# API Endpoints

## User

### API Endpoint: Add Avatar

- **Endpoint**: `/api/users/addavatars`
- **Method**: `POST`
- **Description**: Uploads a profile picture (avatar) for a user. The uploaded file is saved on the server, and the file path is returned in the response.

**Request**

- **Headers**:
  - `Content-Type`: `multipart/form-data`
- **Payload**:
  - Form-data with the following key:
    - `addavatars`: The file input for the avatar image.

**Example Request in Postman**

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/users/addavatars`
3. **Body**:
   - Select `form-data`
   - Key: `addavatars` (type: File)
   - Value: [Choose a file to upload]

**Example Request Body (Form-data)** Key: addavatars Type: File Value: [Select a file]

**Example Response**

```
{
  "message": "Profile picture uploaded successfully!",
  "path": "/uploads/IMG_6073.JPG"
}
```

**Response Codes**

- `200 OK`: Profile picture uploaded successfully.
- `400 Bad Request`: No file uploaded under the `addavatars` key.
- `405 Method Not Allowed`: Request method is not `POST`.
- `500 Internal Server Error`: An error occurred while parsing or saving the file.

**API Endpoint: User Login**

- **Endpoint**: `/api/users/login`
- **Method**: `POST`
- **Description**: Authenticates a user by verifying their email and password. Returns an access token and sets a refresh token as an HTTP-only cookie.

**Request**

- **Headers**:
  - `Content-Type`: `application/json`
- **Payload**:

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

**Example Request in Postman**

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/users/login`
3. Example Payload:

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

**Example Response**

```
{
  "accesstoken": "jwt-access-token"
}
```

**Response Codes**

- `200 OK`: User authenticated successfully. An access token is returned, and a refresh token is set as a cookie.
- `400 Bad Request`: Email or password is missing.
- `401 Unauthorized`: Invalid credentials or the user is banned.
- `405 Method Not Allowed`: Request method is not `POST`.
- `500 Internal Server Error`: An unexpected error occurred.

**API Endpoint: User Profile**

- **Endpoint**: `/api/users/profile`
- **Methods**: `GET`, `PUT`
- **Description**: Handles fetching and updating user profile information based on the provided JWT token.

**Request**

- **Headers**:
  - `Authorization: Bearer <jwt-token>`
  - `Content-Type: application/json` (for `PUT` requests)

**GET Request**

- **Description**: Fetches the user's profile information based on the JWT token.

**Example Request**:

```
GET /api/users/profile HTTP/1.1
Host: localhost:3000
Authorization: Bearer <jwt-token>
```

**Example Response**:

```
{
  "firstname": "a",
  "lastname": "b",
  "email": "user@example.com"
}
```

**PUT Request**

- **Description**: Updates the user's profile information.
- **Payload**:

```
{
  "firstname": "c",
  "lastname": "d",
  "email": "user@example.com"
}
```

**Example Request**:

```
PUT /api/users/profile HTTP/1.1
Host: localhost:3000
Authorization: Bearer <jwt-token>
Content-Type: application/json

{
  "firstname": "John",
  "lastname": "Doe",
  "email": "user@example.com"
}
```

**Example Response**:

```
{
  "firstname": "John",
  "lastname": "Doe",
  "email": "user@example.com"
}
```

**Response Codes**

- `200 OK`: Successfully fetched or updated the user profile.
- `401 Unauthorized`: Missing or invalid token, or token has expired.
- `404 Not Found`: User not found.
- `405 Method Not Allowed`: Method not allowed.
- `500 Internal Server Error`: Unexpected error during processing.

**API Endpoint: User Registration**

- **Endpoint**: `/api/users/register`
- **Method**: `POST`
- **Description**: Registers a new user by storing their information in the database with a hashed password.

**Request**

- **Headers**:
  - `Content-Type: application/json`
- **Payload**:

```
{
  "firstname": "John",
  "lastname": "Doe",
  "email": "user@example.com",
```

```
  "password": "password123",
  "role": "User" // Optional, defaults to "User"
}
```

**Example Request in Postman**

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/users/register`
3. **Body**:
   - Select `raw`
   - Choose `JSON` format
   - Example Payload:

```
{
  "firstname": "John",
  "lastname": "Doe",
  "email": "user@example.com",
  "password": "password123",
  "role": "User"
}
```

**Example Response**

```
{
  "user": {
    "id": 1,
    "firstname": "John",
    "lastname": "Doe",
    "email": "user@example.com",
    "role": "User",
    "createdAt": "2024-11-03T14:35:23.898Z",
    "updatedAt": "2024-11-03T14:35:23.898Z"
  }
}
```

**Response Codes**

- `201 Created`: User registered successfully.
- `400 Bad Request`: Missing required fields (`firstname`, `lastname`, `email`, or `password`), or the email already exists.
- `405 Method Not Allowed`: Request method is not `POST`.
- `500 Internal Server Error`: Error hashing the password or unexpected server error.

**API Endpoint: Admin Protected Middleware**

- **Endpoint**: `/api/admin/protected`
- **Purpose**: Middleware to protect admin routes. It verifies the JWT token and checks if the user has admin privileges.

**Functionality**

- **Authorization Header**: `Bearer <jwt-token>`
- **Checks**:
  - Valid JWT token.
  - User role is `Admin`.

**Response Codes**

- `401 Unauthorized`: No token provided.
- `403 Forbidden`: Invalid token or access denied (non-admin).
- `403 Forbidden`: Token expired.

**API Endpoint: Get Reports**

- **Endpoint**: `/api/admin/getReports`
- **Method**: `GET`
- **Description**: Fetches all reports including details about the associated blog posts or comments.

**Request**

- **Headers**:
  - `Authorization: Bearer <admin-jwt-token>`

**Example Request**:

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/admin/getReports`
3. Authorization: Bearer

**Example Response**:

```
[
  {
    "id": 1,
    "reason": "Inappropriate content",
    "blogPost": { "id": 123, "title": "Sample Blog Post" },
    "comment": { "id": 456, "content": "Sample Comment" }
  }
]
```

**Response Codes**

- `200 OK`: Successfully fetched reports.
- `405 Method Not Allowed`: Request method is not `GET`.
- `500 Internal Server Error`: Failed to fetch reports.

---

**API Endpoint: Manage Blog Post Visibility**

- **Endpoint**: `/api/admin/manageBlogPost`
- **Method**: `PUT`
- **Description**: Updates the visibility status of a blog post.

**Request**

- **Headers**:
  - `Authorization: Bearer <admin-jwt-token>`
- **Payload**:

```
{
  "id": 123,
  "hide": true
}
```

**Example Request**:

```
{
  "id": 123,
  "hide": true
}
```

**Example Response**:

```
{
  "id": 123,
  "hidden": true
}
```

### Response Codes

- `200 OK`: Successfully updated the blog post visibility.
- `400 Bad Request`: Invalid `id` or `hide` value.
- `405 Method Not Allowed`: Request method is not `PUT`.
- `500 Internal Server Error`: Failed to update visibility.

---

### API Endpoint: Manage Users

- **Endpoint**: `/api/admin/manageUsers`
- **Methods**: `PUT`, `DELETE`
- **Description**:
  - **PUT**: Updates a user's role.
  - **DELETE**: Bans a user.

### PUT Request (Update User Role)

- **Headers**:
  - `Authorization: Bearer <admin-jwt-token>`
- **Payload**:

```
{
  "email": "user@example.com",
  "role": "Moderator"
}
```

**Example Request**:

```
{
  "email": "user@example.com",
  "role": "User"
}
```

**Example Response**:

```
{
  "email": "user@example.com",
  "role": "User"
}
```

**DELETE Request (Ban User)**

- **Headers**:
  - `Authorization: Bearer <admin-jwt-token>`
- **Payload**:

```
{
  "email": "user@example.com"
}
```

**Example Request**:

```
{
  "email": "user@example.com"
}
```

**Example Response**:

```
{
  "message": "User banned successfully"
}
```

**Response Codes**

- `200 OK`: Successfully updated the user role or banned the user.
- `400 Bad Request`: Invalid or missing payload.
- `405 Method Not Allowed`: Request method is not `PUT` or `DELETE`.
- `500 Internal Server Error`: Failed to update the user role or ban the user.

**API Endpoint: Create Comment**

- **Endpoint**: `/api/comments/creatcomments`
- **Method**: `POST`
- **Description**: Creates a new comment on a blog post.

**Request**

- **Headers**:
  - `Authorization: Bearer <jwt-token>`
  - `Content-Type: application/json`
- **Payload**:

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

**Example Request in Postman**

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/comments/creatcomments`
3. **Body**:
   - Select `raw`
   - Choose `JSON` format
   - Example Payload:

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

### Example Response

```
{
  "message": "Comment created successfully",
  "comment": {
    "id": 1,
    "content": "This is a comment.",
    "blogPostId": 1,
    "authorId": 2,
    "createdAt": "2024-11-03T14:35:23.898Z"
  }
}
```

### Response Codes

- `201 Created`: Comment created successfully.
- `400 Bad Request`: Blog post ID or content is missing.
- `401 Unauthorized`: Missing or invalid token.
- `405 Method Not Allowed`: Request method is not `POST`.
- `500 Internal Server Error`: Failed to create comment.

---

### API Endpoint: Delete Comment

- **Endpoint**: `/api/comments/deletecomments`
- **Method**: `DELETE`
- **Description**: Deletes a comment. Only the author or an admin can delete a comment.

### Request

- **Headers**:
  - `Authorization: Bearer <jwt-token>`
  - `Content-Type: application/json`
- **Payload**:

```
{
  "id": 1
}
```

### Example Request in Postman

1. **Method**: `DELETE`
2. **URL**: `http://localhost:3000/api/comments/deletecomments`
3. **Body**:
   - Select `raw`
   - Choose `JSON` format
   - Example Payload:

```
{
  "id": 1
}
```

### Example Response

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

### Response Codes

- `200 OK`: Comment deleted successfully.
- `400 Bad Request`: Comment ID is missing.
- `401 Unauthorized`: Missing or invalid token.
- `403 Forbidden`: User is not authorized to delete the comment.
- `404 Not Found`: Comment not found.
- `405 Method Not Allowed`: Request method is not `DELETE`.
- `500 Internal Server Error`: Failed to delete comment.

---

### API Endpoint: Get Comments

- **Endpoint**: `/api/comments/getcomments`
- **Method**: `GET`
- **Description**: Retrieves all comments for a specific blog post. Admins can view hidden comments as well.

### Request

- **Headers**:
  - `Authorization: Bearer <jwt-token>`
- **Query Parameters**:
  - `blogPostId`: The ID of the blog post for which comments are being retrieved.

**Example Request**:

```
GET /api/comments/getcomments?blogPostId=1
Host: localhost:3000
Authorization: Bearer <jwt-token>
```

### Example Response

```
[
  {
    "id": 1,
    "content": "This is a comment.",
    "blogPostId": 1,
    "authorId": 2,
    "createdAt": "2024-11-03T14:35:23.898Z"
  }
]
```

### Response Codes

- `200 OK`: Comments retrieved successfully.
- `400 Bad Request`: Blog post ID is missing.
- `401 Unauthorized`: Invalid or expired token.
- `405 Method Not Allowed`: Request method is not `GET`.
- `500 Internal Server Error`: Failed to fetch comments.

### API Endpoint: Submit Report

- **Endpoint**: `/api/reports/report`
- **Method**: `POST`
- **Description**: Submits a report for a specific content (either a blog post or a comment) indicating the reason for reporting.

### Request

- **Headers**:
  - `Content-Type: application/json`
- **Payload**:

```
{
  "contentId": 123,
  "contentType": "BlogPost", // or "Comment"
  "reason": "Inappropriate content",
  "additionalInfo": "Contains offensive language",
  "userId": 1
}
```

### Example Request in Postman

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/reports/report`
3. **Body**:
   - Select `raw`
   - Choose `JSON` format
   - Example Payload:

```
{
  "contentId": 123,
  "contentType": "BlogPost",
  "reason": "Inappropriate content",
  "additionalInfo": "Contains offensive language",
  "userId": 1
}
```

### Example Response

```
{
  "message": "Report submitted successfully",
  "report": {
    "id": 1,
    "reason": "Inappropriate content",
    "additionalInfo": "Contains offensive language",
    "userId": 1,
    "blogPostId": 123,
    "createdAt": "2024-11-03T14:35:23.898Z"
  }
}
```

### Response Codes

- `201 Created`: Report submitted successfully.

- `400 Bad Request`: Missing required fields (`contentId`, `contentType`, or `reason`) or invalid `contentType`.
- `405 Method Not Allowed`: Request method is not `POST`.
- `500 Internal Server Error`: Failed to submit the report.

### API Endpoint: Save a code template

- **Endpoint**: `/api/codeTemplate/save`
- **Method**: `POST`
- **Description**: This API provides a POST endpoint to create a new code template in the database. Users must be authorized with a valid JWT token, passed in the Authorization header as Bearer . Upon successful validation, the endpoint accepts parameters such as title, description, tags, code, language, and authorId. The authorId must match the ID in the decoded token payload; otherwise, the request will be denied with a 403 Forbidden status. The endpoint responds with the newly created template object upon success (201 Created) or an appropriate error code (401 Unauthorized, 403 Forbidden, 400 Bad Request, 405 Method Not Allowed, or 500 Internal Server Error) if there are issues.

### Request

- **Payload**:

```
{
  "title": "Example Template",
  "description": "This is an example template description.",
  "tags": "example, code",
  "code": "console.log('Hello, World!');",
  "language": "JavaScript",
  "authorId": 1
}
```

### Example Request in Postman

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/codeTemplate/save`

### Example Response

```
{
  "id": 1,
  "title": "Example Template",
  "description": "This is an example template description.",
  "tags": "example, code",
  "code": "console.log('Hello, World!');",
  "language": "JavaScript",
  "authorId": 1
}
```

### Response Codes

- `201 Created`: Template created successfully.

```
{
  "id": 1,
  "title": "Example Template",
  "description": "This is an example template description.",
  "tags": "example, code",
  "code": "console.log('Hello, World!');",
```

```
  "language": "JavaScript",
  "authorId": 1
}
```

- `400 Bad Request`: One or more required fields (`title`, `description`, `tags`, `code`, `language`, `authorId`) are missing in the request body.

```
{
  "error": "All fields are required"
}
```

- `401 Unauthorized`: Missing or invalid token in the Authorization header.

```
{
  "error": "Unauthorized: No token provided"
}
```

or

```
{
  "error": "Invalid token"
}
```

- `403 Forbidden`: The user ID in the token does not match the `authorId` provided in the request body.

```
{
  "error": "Forbidden: You do not have permission to perform this action"
}
```

- `405 Method Not Allowed`: Request method is not POST.

```
{
  "error": "Method Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "success": false,
  "message": "Internal Server Error"
}
```

**API Endpoint: Show a code template**

- **Endpoint**: `/api/codeTemplate/show`
- **Method**: `GET`
- **Description**: This API provides a GET endpoint for retrieving code templates based on specific query criteria. Users can specify the search criterion using the options and info query parameters. The options parameter can be userId, title, or tags, and the info parameter provides the corresponding value. For example, when options=userId, info should be an integer representing the author's ID. When options=title, info should be the template's title. When options=tags, info should contain a tag to search within the tags field. The endpoint responds with a list of matching templates (200 OK). If options is invalid, it returns a 400 Bad Request. Supported error responses include 405 Method Not Allowed for unsupported HTTP methods and 500 Internal Server Error for any server-side issues.

**Request**

- **Payload**:

```
GET /api/codeTemplate?options=userId&info=1
```

### Example Request in Postman

1. **Method**: `GET`
2. **URL**: `http://localhost:3000/api/codeTemplate/show`

### Example Response

```
{
        "id": 1,
        "title": "Sample Template",
        "description": "A sample template description.",
        "tags": "example, code",
        "code": "console.log('Hello, World!');",
        "language": "JavaScript",
        "authorId": 1
    },
```

### Response Codes

- `200 OK`: Successfully retrieved the requested code templates.

```
[
  {
    "id": 1,
    "title": "Sample Template",
    "description": "This is a sample template description.",
    "tags": "example, code",
    "code": "console.log('Hello, World!');",
    "language": "JavaScript",
    "authorId": 1
  },
  {
    "id": 2,
    "title": "Another Template",
    "description": "Another example template",
    "tags": "sample, code",
    "code": "print('Hello from Python!')",
    "language": "Python",
    "authorId": 2
  }
]
```

- `400 Bad Request`: Invalid options parameter provided in the query.

```
{
  "error": "Method Not Allowed"
}
```

- `405 Method Not Allowed`: Request method is not GET.

```
{
  "error": "Method Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "success": false,
```

```
  "message": "Internal Server Error"
}
```

### API Endpoint: Delete a code template

- **Endpoint**: `/api/codeTemplate/delete`
- **Method**: `DELETE`
- **Description**: This API provides a DELETE endpoint for deleting an existing code template. Users must be authorized with a valid JWT token, passed in the Authorization header as Bearer . The id of the code template to delete should be provided as a query parameter. The endpoint first checks if the specified template exists; if not, it returns a 404 Not Found response. If the template exists, it is deleted, and a success message is returned.

### Request

- **Payload**:

```
DELETE /api/codeTemplate/delete?id=1
```

### Example Request in Postman

1. **Method**: `DELETE`
2. **URL**: `http://localhost:3000/api/codeTemplate/delete`

### Example Response

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

### Response Codes

- `200 OK`: Template deleted successfully.

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

- `401 Unauthorized`: Missing or invalid token in the Authorization header.

```
{
  "error": "Unauthorized: No token provided"
}
```

- `404 Not Found`: Template with the specified `id` does not exist.

```
{
  "error": "Template not found"
}
```

- `405 Method Not Allowed`: Request method is not DELETE.

```
{
  "error": "Method Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "success": false,
```

```
  "message": "Internal Server Error"
}
```

### API Endpoint: Update a code template

- **Endpoint**: `/api/codeTemplate/update`
- **Method**: `PATCH`
- **Description**: This API provides a PATCH endpoint to update an existing code template. Users must be authorized with a valid JWT token, passed in the Authorization header as Bearer . The id of the code template to update should be provided as a query parameter. In the request body, fields such as title, description, tags, code, and language can be included to specify which attributes of the template to update. Only fields provided in the request body will be updated, while any fields left out will remain unchanged.

### Request

- **Payload**:

```
{
  "title": "Updated Title",
  "description": "Updated description",
  "tags": "updated, example",
  "code": "console.log('Updated Code');",
  "language": "JavaScript"
}
```

### Example Request in Postman

1. **Method**: `PATCH`
2. **URL**: `http://localhost:3000/api/codeTemplate/update`

### Example Response

```
{
  "title": "Updated Title",
  "description": "Updated description",
  "tags": "updated, example",
  "code": "console.log('Updated Code');",
  "language": "JavaScript"
}
```

### Response Codes

- `201 Created`: Template updated successfully.

```
{
  "id": 1,
  "title": "Updated Template Title",
  "description": "Updated template description",
  "tags": "updated, example",
  "code": "console.log('Updated Code');",
  "language": "JavaScript",
  "authorId": 1
}
```

- `404 Not Found`: Template with the specified `id` does not exist.

```
{
  "error": "Template not found"
}
```

- `400 Bad Request`: Missing or invalid fields in the request body.

```
{
  "error": "Invalid request: Required fields are missing or invalid."
}
```

- `405 Method Not Allowed`: Request method is not PATCH.

```
{
  "error": "Method Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "success": false,
  "message": "Internal Server Error"
}
```

### API Endpoint: Execute a code template

- **Endpoint**: `/api/codeTemplate/execution`
- **Method**: `POST`
- **Description**: This API provides a GET endpoint to fetch a code template by its ID and execute the code based on its specified programming language. The endpoint supports various languages including JavaScript, Python, Java, C, and C++. Users can specify the id of the template they want to execute through the query parameter. The endpoint retrieves the template from the database, writes the code to a temporary file, and runs it using appropriate system commands. After execution, it returns the output of the code or any error messages encountered during execution. Note that only supported languages will be executed; unsupported languages will return a 400 error.

### Request

- **Payload**:

```
GET /api/codeTemplate/execution?id=1
```

### Example Request in Postman

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/codeTemplate/execution`

### Example Response

```
{
  "output": "Hello, World!"
}
```

### Response Codes

- `201 OK`: Template fetched and code executed successfully.

```
{
  "output": "Hello, World!"
}
```

- `400 Bad Request`: If the provided language is unsupported:

```
{
  "error": "Unsupported language"
}
```

- `400 Bad Request`: If there is an error during code execution:

```
{
  "error": "Execution error: <error message>"
}
```

- `404 Not Found`: Template with the specified `id` does not exist.

```
{
  "error": "Template not found"
}
```

- `405 Method Not Allowed`: Request method is not GET.

```
{
  "error": "Method Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "success": false,
  "message": "Internal Server Error"
}
```

### API Endpoint: Create a Blog Post

- **Endpoint**: `/api/blogpost/create`
- **Method**: `POST`
- **Description**: This API provides a POST endpoint to create a new blog post. Users must be authorized with a valid JWT token, passed in the Authorization header as Bearer . In the request body, fields such as title, description, content, and tags are required to create a post, while codeTemplateId is optional and allows associating existing code template with the blog post. #### Request
- **Authorization**: Bearer token
- **Payload**:

```
{
  "title": "Sample Blog Post",
  "description": "This is a description of the blog post.",
  "content": "Content for the blog post goes here.",
  "tags": "sample, blog, post",
  "codeTemplateIds": 1 //optional Id for associated code template
}
```

### Example Request in Postman

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/codeTemplate/execution`
3. **Headers** `Authorization: Bearer <JWT_TOKEN>`

### Example Response

- `Success (201 Created)`:Blog post created successfully.


```
{
  "success": true,
  "data": {
    "id": 1,
    "title": "Sample Blog Post Title",
    "description": "This is a sample description for the blog post.",
    "content": "Here is the main content of the blog post.",
    "tags": "sample, blog, post",
    "createdAt": "2024-11-02T19:15:54.584Z",
    "updatedAt": "2024-11-02T19:15:54.584Z",
    "userId": 3,
    "codeTemplates": [
      {
        "id": 1,
        "title": "Sample Code Template 1",
        "description": "A sample code template.",
        "code": "console.log('Hello World');",
        "language": "JavaScript"
      },
      {
        "id": 2,
        "title": "Sample Code Template 2",
        "description": "Another code template example.",
        "code": "print('Hello World')",
        "language": "Python"
      }
    ]
  }
}
```


### Response Codes

- `201 Created`: Blog post created successfully
- `400 Bad Request`:If any required fields are missing or invalid.


```
{
  "error": "All fields (title, description, content, tags) are required"
}
```


- `401 Unauthorized`: Authorization token is missing or invalid.


```
{
  "error": "Unauthorized: Invalid or expired token"
}
```


- `405 Method Not Allowed`: Request method is not POST.


```
{
  "error": "Method <method> Not Allowed"
}
```


- `500 Internal Server Error`: An unexpected error occurred.


```
{
  "error": "Failed to create blog post"
```

```
}
```

### API Endpoint: Create a Blog Post

- **Endpoint**: `/api/blogpost`
- **Method**: `GET`
- **Description**: This endpoint allows visitors to search for blog posts based on various criteria. The user can provide any combination of the query parameters title, content, tags, or codeTemplate to search for relevant blog posts. Each parameter performs a partial (case-insensitive) match, allowing flexible search functionality.

### Query Request

- `title` (optional): Searches for blog posts containing the specified text within the title.
- `content` (optional): Searches for blog posts containing the specified text within the content.
- `tags` (optional): Searches for blog posts that include the specified tags.
- `codeTemplate` (optional): Searches for blog posts that reference a code template with a matching title.

### Example Request in Postman

1. **Method**: `GET`
2. **URL**: `http://localhost:3000/api/blogpost?title=Sample+Title&tags=example`

### Example Response

- **Response (200 OK)**: Blog posts retrieved successfully.


```
{
    "success": true,
    "data": [
        {
            "id": 2,
            "title": "Updated Title",
            "description": "Updated description of the blog post.",
            "content": "Updated content for the blog post.",
            "tags": "updated, tags",
            "createdAt": "2024-11-02T19:15:54.584Z",
            "updatedAt": "2024-11-02T20:26:39.629Z",
            "userId": 3,
            "codeTemplates": [
                {
                    "id": 1,
                    "title": "this is code 2",
                    "description": "second save",
                    "tags": "remove",
                    "code": "print('bye')",
                    "language": "python",
                    "isForked": false,
                    "authorId": 3,
                    "createdAt": "2024-11-02T18:37:51.670Z",
                    "updatedAt": "2024-11-02T18:37:51.670Z"
                }
            ]
        }
    ]
}
```

**Response Codes**

- `200 OK`: Blog posts retrieved successfully.
- `400 Bad Request`: No search criteria provided.


```
{
  "success": false,
  "message": "No search criteria provided"
}
```


- `404 Not Found`: No blog posts found for the provided criteria.


```
{
  "success": false,
  "message": "No blog posts found"
}
```


- `500 Internal Server Error`: An unexpected error occurred.


```
{
  "success": false,
  "message": "Failed to fetch blog posts"
}
```


**API Endpoint: Get Blog Posts Sorted by Rating**

- **Endpoint**: `/api/blogpost/sortedByRating`
- **Method**: `GET`
- **Description**: This endpoint retrieves all blog posts and their associated comments, sorted by rating. Each blog post and comment has an aggregated rating score calculated based on upvotes (1) and downvotes (-1). The response provides blog posts ordered by their rating score, with comments within each post also sorted by rating.

**Example Request in Postman**

1. **Method**: `GET`
2. **URL**: `http://localhost:3000/api/blogpost/sortedByRating`

**Example Response**

- **Response (200 OK)**: Blog posts with comments sorted by rating score.


```
{
    "success": true,
    "data": {
        "id": 3,
        "title": "Python guidline",
        "description": "A comprehensive guide to async programming in python",
        "content": "...",
        "tags": "python, guide",
        "createdAt": "2024-11-03T21:50:56.284Z",
        "updatedAt": "2024-11-03T21:50:56.284Z",
        "userId": 3,
        "ratings": [
            {
                "value": 1
            }
        ],
```

```
        "comments": [],
        "ratingScore": 1
    }
}
```

**Response Codes**

- `200 OK`: Blog posts and comments retrieved and sorted successfully.
- `405 Method Not Allowed`: Request method is not GET.

```
{
  "error": "Method GET Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred.

```
{
  "error": "Failed to fetch sorted blog posts"
}
```

**API Endpoint: Delete Blog Post**

- **Endpoint**: `/api/blogpost/delete`
- **Method**: `DELETE`
- **Description**: This endpoint allows authorized users to delete a blog post by its ID. Users must be authenticated with a valid JWT token, which is passed in the Authorization header as Bearer . Only the author of the blog post is permitted to delete it. The blog post ID should be provided as a query parameter in the request.

**Example Request in Postman**

1. **Method**: 'DELETE
2. **URL**: `http://localhost:3000/api/blogpost/delete?id=<blogPostId>`

- **Headers**:
- `Authorization: Bearer <token>`

**Example Response**

- **Response (200 OK)**: Blog post deleted successfully.
- **Response (403 Forbidden)**: User does not have permission to delete this blog post.

```
{
  "error": "Forbidden: You do not have permission to delete this blog post"
}
```

- **Response (404 Not Found)**: Blog post with the specified ID does not exist.

```
{
  "error": "Blog post not found"
}
```

- **Response (400 Bad Request)**:Missing or invalid fields in the request.

```
{
  "error": "Blog post ID is required"
}
```

**Response Codes**

- `200 OK`: Blog post deleted successfully.
- `403 Forbidden`: User does not have permission to delete the specified blog post.
- `404 Not Found`: Blog post with the specified ID does not exist.
- `405 Method Not Allowed`: Request method is not DELETE.

```
{
  "error": "Method DELETE Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "error": "Failed to delete blog post"
}
```

**API Endpoint: Update/Edit Blog Post**

- **Endpoint**: `/api/blogpost/[id]/edit`
- **Method**: `PUT`
- **Description**: This endpoint allows authenticated users to update an existing blog post. Users must provide a valid JWT token in the Authorization header as Bearer . Only the author of the blog post is permitted to update it. The blog post ID and the updated fields (title, description, content, tags) should be included in the request body.

**Example Request in Postman**

1. **Method**: `PUT`
2. **URL**: `http://localhost:3000/api/blogpost/<id>/edit`

**Request Headers**: - `Authorization: Bearer <token>`

**Payload**:

```
{
  "id": 1,
  "title": "Updated Title",
  "description": "Updated description of the blog post",
  "content": "Updated content for the blog post",
  "tags": "updated, tags"
}
```

**Example Response**

- **Response (200 OK)**: Blog post updated successfully.

```
{
  "success": true,
  "data": {
    "id": 1,
    "title": "Updated Title",
    "description": "Updated description of the blog post",
    "content": "Updated content for the blog post",
    "tags": "updated, tags",
    "userId": 3,
    "createdAt": "2024-11-03T02:10:18.000Z",
    "updatedAt": "2024-11-03T02:40:18.000Z"
```

```
  }
}
```

- **Response (403 Forbidden)**: User does not have permission to update this blog post.

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

- **Response (404 Not Found)**: Blog post with the specified ID does not exist.

```
{
  "error": "Blog post not found"
}
```

- **Response (400 Bad Request): Missing or invalid fields in the request body.

```
{
  "error": "All fields (id, title, description, content, tags) are required"
}
```

### Response Codes

- `200 OK`: Blog post updated successfully.
- `403 Forbidden`: User does not have permission to update the specified blog post.
- `404 Not Found`: Blog post with the specified ID does not exist.
- `405 Method Not Allowed`: Request method is not PUT.

```
{
  "error": "Method PUT Not Allowed"
}
```

- `500 Internal Server Error`: An unexpected error occurred while processing the request.

```
{
  "error": "Failed to update blog post"
}
```

### API Endpoint: Rate a Blog Post

- **Endpoint**: `/api/blogpost/[id]/rate`
- **Method**: `POST`
- **Description**: This endpoint allows authenticated users to rate a specific blog post by its id with either an upvote (1) or a downvote (-1). A valid JWT token is required in the Authorization header as Bearer . If the user has already rated the post, their rating will be updated; otherwise, a new rating will be created.

### Example Request in Postman

1. **Method**: `POST`
2. **URL**: `http://localhost:3000/api/blogpost/<id>/rate`

**Request Headers**: - `Authorization: Bearer <token>`

**Payload**:

```
{
  "value": 1  // Use 1 for upvote, -1 for downvote
}
```

**Example Responses**

- **Response (200 OK)**: Rating created or updated successfully.


```
{
  "success": true,
  "data": {
    "id": 1,
    "value": 1,
    "createdAt": "2024-11-03T21:52:08.364Z",
    "userId": 3,
    "blogPostId": 3,
    "commentId": null
  }
}
```


- **Response (400 Bad Request)**: Invalid rating value or missing required fields.


```
{
  "error": "Invalid rating value. Use 1 for upvote, -1 for downvote."
}
```


- **Response (401 Unauthorized)**: No token provided or invalid token.


```
{
  "error": "Unauthorized: No token provided"
}
```


- **Response (403 Forbidden)**: The user does not have permission to rate the blog post.


```
{
  "error": "Forbidden: You do not have permission to rate this blog post"
}
```


- **Response (404 Not Found)**: Blog post with the specified id does not exist.


```
{
  "error": "Blog post not found"
}
```


**Response Codes**

- `200 OK`: Rating was created or updated successfully.
- `400 Bad Request`: The rating value is invalid or required fields are missing.
- `401 Unauthorized`: User is not authenticated.
- `403 Forbidden`: User does not have permission to rate the blog post.
- `405 Method Not Allowed`: Request method is not POST


```
{
  "error": "Method POST Not Allowed"
}
```


- `500 Internal Server Error`: An unexpected error occurred while processing the request.


```
{
  "error": "Failed to rate blog post"
}
```