Model Design

1. User

    - `id`: Integer, primary key, auto-incremented.
    - `email`: String, unique; used for user authentication.
    - `password`: String; stores the user's password.
    - `firstName`: String; the user's first name.
    - `lastName`: String; the user's last name.
    - `avatar`: Optional String; stores a link or file path for the user's avatar image.
    - `phone`: Optional String; the user's phone number.
    - `isAdmin`: Boolean; indicates if the user has administrative privileges (default: `false`).
- **Relationships**:
    - `templates`: One-to-many relation with `Template`; each user can create multiple templates.
    - `posts`: One-to-many relation with `Post`; each user can create multiple posts.
    - `comments`: One-to-many relation with `Comment`; each user can leave multiple comments.
    - `ratings`: One-to-many relation with `Rating`; users can rate multiple items.
    - `reports`: One-to-many relation with `Report`; users can report multiple items.

2. Template

- **Fields**:
    - `id`: Integer, primary key, auto-incremented.
    - `title`: String; the name or title of the template.
    - `code`: String; contains the code associated with this template.
    - `explanation`: Optional String; a description or explanation of the template's purpose or functionality.
- **Relationships**:
    - `user`: Many-to-one relation with `User`; each template is owned by a single user.
    - `tags`: Many-to-many relation with `Tag`; each template can have multiple tags for categorization.

3. Tag

- **Fields**:
    - `id`: Integer, primary key, auto-incremented.
    - `name`: String; the name of the tag (e.g., "JavaScript", "Beginner").
- **Relationships**:

    - `templates`: Many-to-many relation with `Template`; a tag can be assigned to multiple templates, enabling flexible categorization.

4. Post

- **Fields**:
    - `id`: Integer, primary key, auto-incremented.
    - `title`: String; title of the post.
    - `description`: String; description or content of the post.
    - `userId`: Integer; foreign key relating the post to a user.
- **Relationships**:
    - `user`: Many-to-one relation with `User`; each post is created by a single user.

5. Comment

- **Fields**:
    - `id`: Integer, primary key, auto-incremented.
    - `content`: String; the content of the comment.
    - `userId`: Integer; foreign key relating the comment to a user.
    - `postId`: Integer; foreign key relating the comment to a post.
- **Relationships**:
    - `user`: Many-to-one relation with `User`; each comment is made by a single user.
    - `post`: Many-to-one relation with `Post`; each comment is associated with a single post.

6. Rating

- **Fields**:
    - `id`: Integer, primary key, auto-incremented.
    - `value`: Integer; represents the rating value (e.g., 1 to 5).
    - `userId`: Integer; foreign key linking the rating to the user who gave it.
    - `templateId`: Integer; foreign key linking the rating to a specific template.
- **Relationships**:
    - `user`: Many-to-one relation with `User`; each rating is given by a single user.
    - `template`: Many-to-one relation with `Template`; each rating is associated with a single template.

7. Report

- **Fields**:
    - `id`: Integer, primary key, auto-incremented.
    - `reason`: String; description of the reason for the report.
    - `userId`: Integer; foreign key linking the report to the user who reported it.

    - `templateId`: Integer; foreign key linking the report to a specific template.
- **Relationships**:
    - `user`: Many-to-one relation with `User`; each report is filed by a single user.
    - `template`: Many-to-one relation with `Template`; each report is associated with a specific template
    -

API Endpoints

1. Signup Endpoint

- **URL**: `/api/auth/signup`
- **Method**: `POST`
- **Description**: Registers a new user in the system.

Payload:

```
{
  "email": "string (required)",
  "password": "string (required)",
  "firstName": "string (required)",
  "lastName": "string (required)",
  "phone": "string (optional)",
  "isAdmin": "bool (optional)",

}
```

Example Request:

```
{
  "email": "test@example.com",
  "password": "password123",
  "firstName": "John",
  "lastName": "Doe",
  "phone": "1234567890"
}
```

Example Response:

```
{
  "message": "User created successfully",
  "user": {
    "id": 1,
    "email": "test@example.com",
    "firstName": "John",
```

```
    "lastName": "Doe",
    "phone": "1234567890",
    "isAdmin": false

  }
}
```

Example Request:

```
{
  "email": "ADMIN",
  "password": "ADMIN",
  "firstName": "ADMIN",
  "lastName": "ADMIN",
  "phone": "ADMIN",
  "isAdmin": true
}
```

Example Response:

```
{
  "message": "User created successfully",
  "user": {
    "id": 2,
    "email": "ADMIN",
    "firstName": "ADMIN",
    "lastName": "ADMIN",
    "phone": "ADMIN",
    “isAdmin”: true
  }
}
```

- **Error Responses**:
  - `400`: Missing required fields or user already exists.
  - `500`: Server error during user creation.

2. Login Endpoint

- **URL**: `/api/auth/login`
- **Method**: `POST`
- **Description**: Authenticates a user and issues a JWT.

Payload:

```
{
  "email": "string (required)",
  "password": "string (required)"
}
```

Example Request:

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

Example Response:

```
{
  "message": "Login successful",
  "token": "JWT Token Here"
}
```

- **Error Responses**:
  - `400`: Missing email or password.
  - `404`: User not found.
  - `401`: Invalid credentials.
  - `500`: Internal server error.

3. Logout Endpoint

- **URL**: `/api/auth/logout`
- **Method**: `POST`
- **Description**: Logs the user out by clearing the authentication cookie.
- **Payload**: None

Example Request:

```
{}
```

Example Response:

```
{
  "message": "Successfully logged out"
}
```

- **Error Responses**: None

4. Admin - Hide/Unhide Post

- **URL**: `/api/auth/admin-hide`
- **Method**: `PUT`
- **Description**: Allows an admin to hide or unhide a post.
- **Authentication**: Requires valid access or refresh token.

```
{

  "postId": "integer (required)",
  "isHidden": "boolean (required)"
}
```

Example Request:

```
{
  "postId": 1,
  "isHidden": true
}
```

Example Response:

```
{
  "id": 1,
  "isHidden": true
}
```

- **Error Responses**:
  - `401`: Authentication required.
  - `403`: Admin privileges required.
  - `400`: Missing `postId` or `isHidden`.
  - `500`: Error updating post.

5. Admin - Rank by reports

- **URL**: `/api/auth/admin-mod`
- **Method**: `GET`
- **Description**: Allows an admin to fetch posts and comments sorted by report count for moderation.
- **Authentication**: Requires valid access or refresh token.
- **Payload**: None
- **Query Parameters**:
  - `sort`: Sort order for reports, accepts `asc` or `desc`.

Example Request:

```
GET /api/auth/admin-mod?sort=desc
```

Example Response:

```
{
  "posts": [
    {
      "id": 1,
      "isHidden": false,
      "reportCount": 5
    },
    {
      "id": 2,
      "isHidden": false,
      "reportCount": 2
    }
  ],
  "comments": [
    {
      "id": 2,
      "reportCount": 1
    }
  ]
}
```

- **Error Responses**:
  - `401`: Authentication required.
  - `403`: Admin privileges required.
  - `500`: Error fetching moderation data.

6. Comment Endpoint - `/api/comment`

- **Description**: Allows authenticated users to post comments on an existing post, or reply to an existing comment.
- **Method**: `POST`
- **Payload**:
  - `postId` (integer): ID of the post to comment on.
  - `parentId` (integer, optional): ID of the parent comment if this is a reply.
  - `content` (string): The comment text.

Example Request:

```
{
```

```
  "postId": 123,
  "parentId": 45,
  "content": "This is a comment."
}
```

Example Response:

```
{
  "message": "Comment added successfully",
  "comment": { /* comment data */ }
}
```

7. Execute Code Endpoint - `/api/execute`

- **Description**: Executes code snippets in supported languages and returns the output.
- **Method**: `POST`
- **Payload**:
  - `code` (string): The code to execute.
  - `language` (string): Language of the code (`python`, `javascript`, `java`, `c`, `cpp`).
  - `input` (string, optional): Input for the code if required.

Example Request:

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

Example Response:

```
{
  "output": "Hello, World!\n",
  "error": null
}
```

8. Rating Endpoint - `/api/rating`

- **Description**: Allows authenticated users to upvote or downvote a post or comment.
- **Method**: `POST`
- **Payload**:
  - `postId` (integer, optional): ID of the post to rate.
  - `commentId` (integer, optional): ID of the comment to rate.

    - `upvote` (boolean): True for upvote.
    - `downvote` (boolean): True for downvote.

Example Request:

```
{
  "postId": 123,
  "upvote": true
  “downvote”: false
}
```

Example Response:

```
{
  "postId": 123,
  "upvote": true
  “downvote”: false
}
```

Example Request:

```
{
  "commentId": 123,
  "upvote": true
  “downvote”: false
}
```

Example Response:

```
{
  "commentId": 123,
  "upvote": true
  “downvote”: false
}
```

9. Report Endpoint - `/api/report`

- **Description**: Allows authenticated users to report a post or comment for moderation.
- **Method**: `POST`
- **Payload**:
    - `postId` (integer, optional): ID of the post to report.
    - `commentId` (integer, optional): ID of the comment to report.
    - `reason` (string): Reason for reporting.

Example Request:

```
{
  "postId": 123,
```

```
  "reason": "Inappropriate content"
}
```
Example Response:
```
{
  "message": "Report submitted successfully",
  "report": {
                {
                      "postId": 123,
                      "reason": "Inappropriate content"
                }
            }
}
```

10. Sort Ratings Endpoint - `/api/posts/sort_ratings`

- **Description**: Retrieves a sorted list of posts based on net rating.
- **Method**: `GET`
- **Query Parameters**:
    - `sort` (string, optional): Sort order, either `asc` or `desc`.

Example Request:

```
GET /api/posts/sort_ratings?sort=desc
```
Example Response:

```
[
  { /* post data with netRating: 5 */ },
  { /* post data with netRating: 3 */ },
  { /* post data with netRating: 1 */ },
  { /* post data with netRating:-2 */ }
]
```

11. Browse Posts Endpoint - `/api/posts/browse`

- **Description**: Retrieves posts based on various filters like post ID, tag ID, template ID, or title.
- **Method**: `GET`
- **Query Parameters**:
    - `postid` (integer, optional): Filter by post ID.
    - `tagId` (integer, optional): Filter by tag ID.

    - `templateid` (integer, optional): Filter by template ID.
    - `title` (string, optional): Filter by title.

Example Request:

```
GET /api/posts/browse?tagId=5
```

Example Response:

```
[{
  "id": 123,
  "title": "My New Post",
  "description": "This is a description",
  "tags": [{“id”: 5, “id”: 1}],
  "templates": []

}]
```

12. Posts Endpoint - `/api/posts`

- **Description**: General endpoint for post operations such as fetching, creating, editing, and deleting posts.
- **Methods**:
    - **GET**: Fetches posts visible to the user.
    - **POST**: Creates a new post.
    - **PUT**: Updates a post if the user is the author.
    - **DELETE**: Deletes a post if the user is the author.
- **Payload (POST and PUT)**:
    - `title` (string): Title of the post.
    - `description` (string): Description of the post.
    - `tags` (array, optional): Tags to associate with the post.
    - `templates` (array, optional): Templates to associate with the post.

Example POST Request:

```
{
  "title": "My New Post",
  "description": "This is a description",
  "tags": [1, 2],
  "templates": [3]
}
```

Example Response:

```
{
  "id": 123,
  "title": "My New Post",
  "description": "This is a description",
  "tags": [{ /* tag data */ }],
  "templates": [{ /* template data */ }]

}
```

13. Posts Endpoint - `/api/users`

- **Description**: Fetches a list of all users, including their associated templates, posts, comments, ratings, and reports.
- **Method**: `GET`
- **Payload**: None

Example Request:

```
GET /api/users
```

Example Response:

```
{
  "status": 200,
  "data": [
    {
      "id": 1,
      "name": "User1",
      "templates": [...],
      "posts": [...],
      "comments": [...],
      "ratings": [...],
      "reports": [...]
    },
    ...
  ]
}
```

12. Template Operations by templateID - `/api/templates/[templateId]`

- **Description**: Performs `GET`, `PUT`, and `DELETE` operations on a specific template by `templateId`.
- **Allowed Methods**: `GET`, `PUT`, `DELETE`
- **Endpoints**:

Example Fetch Request:
Description: Retrieves a single template by its ID.

```
GET /api/templates/1
```

Example Response:

```
{
  "id": 1,
  "title": "Example Template",
  "code": "console.log('Hello World');",
  "explanation": "A simple Hello World template",
  "tags": [...],
  "user": {...}
}
```

Example Update Template Request:
Description: Updates a template if the authenticated user is the owner.

```
PUT /api/templates/1
```

Example Response:

```
{
  "status": 200,
  "data": {
    "id": 1,
    "title": "Updated Template Title",
    "code": "console.log('Updated Code');",
    "explanation": "Updated explanation",
    "tags": [...]
  }
}
```

Example Delete Request:
Description: Deletes a template if the authenticated user is the owner.

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

Example Response:

```
{
  "status": 200,
```

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

Admin User:

email: “ADMIN”
Password: “ADMIN”