CSC309 Endpoint Documentation

By: Emily Zhou, Kevin Wang, Benson Chou

> Note that all endpoints must be preceded by ‘api’ (eg. if the endpoint is /auth/signup, the full path is http://localhost:3000/api/auth/signup)

Note that on the git commits, “linux” is a linux VM that Kevin installed, so commits from “linux” are from Kevin.

The startup.sh script will create an admin user:

- username: admin
- password: admin

PLEASE run the startup.sh script as `root` . Otherwise, it will not set things up.

IF this fails, please use the signup/login/makeadmin endpoints to make an admin manually.

Table of Contents

Notes on Design Decisions

Note that content — posts, comments and replies — are stored in one table in the DB, with separate tables containing metadata pertaining to each content type. These pieces of content can be referenced with their contentID, for the purposes of voting, reporting and hiding the content. This will be different than postID, commentID etc, which some endpoints use, so it's important to make that distinction.

Many of these endpoints require Bearer Token Authentication, and some endpoints require the user to be an admin to authenticate.

Login and Authorization (Kevin)

POST: `/auth/signup`

Allows users to signup for a new (non-admin) account.

Request Body (JSON)

Name	Data Type	Description
username	string	Username of the user to create. Must be unique
password	string	Password of the user to create
firstName	string	First name
lastName	string	Last Name
email	string	Email, must be valid email address, must be unique
phone	string(optional)	Phone number, must be 10 digits in a row, no spaces

Response Body (JSON)

Name	Data Type	Description
message	string	Message from the server, for successful requests
error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

```
{
    "username": "user1",
    "password": "abc",
    "firstName": "Kevin",
    "lastName": "Wang",
    "email": "kevin@gmail.com",
    "phone": "1234567890"
}
```

Sample Response Body (JSON)

```
{
    "message": "Could not create user"
}
```

POST: `/auth/login`

Allows users to login with username/password, returning access and refresh tokens for Token Auth.

Request Body (JSON)

Name	Data Type	Description
username	string	Username of the user to create
password	string	Password of the user to create

Response Body (JSON)

Name	Data Type	Description
accessToken	string	Access token
refreshToken	string	Refresh token
error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

```
{
    "username": "user1",
    "password": "abc"
}
```

Sample Response Body (JSON)

```
{
    "accessToken": "thisisnotarealtoken",
    "refreshToken": "thisisnotarealtoken"
}
```

POST: `/auth/refresh`

Refreshes the access token, given the refresh token for Token Auth

Request Body (JSON)

Name	Data Type	Description
refreshToken	string	Refresh token

Response Body (JSON)

Name	Data Type	Description
accessToken	string	New access token
error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

```
{
    "refreshToken": "notarealrefreshtoken"
}
```

Sample Response Body (JSON)

```
{
    "accessToken": "notarealaccesstoken"
}
```

POST: `/api/auth/addphoto`

Adds a profile photo to a given account

Request Body (JSON)

Add a photo in the formData. Bearer authorization is required.

Response Body (JSON)

Name	Data Type	Description
message	string	Message from the server, for successful requests
error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

Upload photo

Sample Response Body (JSON)

```
{
    "message": "Success"
}
```

PUT: `/auth/makeadmin`

Attempts to make a user into an admin. Bearer authentication is required, plus a password for authorization in the request body.

Normally, we should be setting admin privileges in the server through the database, and it doesn't make sense to expose an API endpoint that would do that. However, for the purposes of testing this project, we made a convenient endpoint for this purpose.

Request Body (JSON)

Name	Data Type	Description

password	string	The magic word for privilege escalation

Response Body (JSON)

Name	Data Type	Description
message	string	Message from the server, for successful requests
error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

```
{
    "password": "admin"
}
```

Sample Response Body (JSON)

```
{
    "message": "Success"
}
```

Code Templates (Emily)

PUT: `/template/save/[templateID]`

This API endpoint allows authenticated users to update an existing code template by providing a template ID and optional fields (code, title, explanation, tags) to modify. Only the owner of the code template can perform the update.

Query Parameters

Name	Type	Description

templateID	number (Required)	The unique ID of the code template to be updated.

Request Body (JSON)

Name	Type	Description
code	string (Optional)	The updated source code of the template
title	string (Optional)	The updated title of the template
explanation	string (Optional)	The updated explanation or description for the template
language	string (Optional)	The updated language of the template.
tags	array of string (Optional)	A list of tags to associate with the template. The old tags will be cleared and replaced with the new tags. Tags not found will be created

Example of request body:

```
{
  "code": "function example() { return 'Hello World'; }",
  "title": "Updated Example",
  "explanation": "This template prints 'Hello World'.",
  "language": "Javascript",
  "tags": ["javascript", "example"]
}
```

Response (JSON)

- **200 Success**: Returns the updated code template object.
- **400 Bad Request**: If templateID is missing from the request.
- **401 Unauthorized**: If the user is not logged in or the JWT is invalid.
- **403 Forbidden**: If the user is not the owner of the code template.
- **404 Not Found**: If the specified code template does not exist.

- **405 Method Not Allowed**: If a method other than PUT is used.
- **500 Internal Server Error**: If an error occurs during the update process.

POST: `/template/save`

This API endpoint allows logged-in users to create a new code template, either from scratch or by forking an existing template. The new template is associated with the logged-in user. If a forkID is provided, the template is a fork of an existing template.

Request Body (JSON)

Name	Type	Description
code	string	The source code of the new template. Required if not forking from an existing template.
title	string	The title of the new template. Required if not forking from an existing template.
explanation	string (Optional)	A description or explanation for the code template.
language	string (Optional)	The updated language of the template. Mandatory if creating, optional if forking.
tags	array of string (Optional)	A list of tags to associate with the template. Tags not found will be created.
forkID	number (Optional)	The ID of an existing template to fork. If provided, the new template will copy the code, title, and explanation from the forked template, but tags will not be copied.

Example request body:

```
// creating a new template
{
  "code": "function example() { return 'Hello World'; }",
  "title": "Example Template",
```

```
  "explanation": "This template prints 'Hello World'.",
  "language": "Javascript",
  "tags": ["javascript", "example"]
}

// forking a template
{
  "forkID": 1
}
```

Response (JSON)

- **201 Created**: Returns the newly created code template object.
- **400 Bad Request**: If required fields (like code and title) are missing when creating from scratch, or if the forkID is invalid.
- **401 Unauthorized**: If the user is not logged in or the JWT is invalid.
- **403 Forbidden**: If the user is not the owner of the template when attempting to fork.
- **404 Not Found**: If the template to be forked does not exist.
- **405 Method Not Allowed**: If a method other than POST is used.
- **500 Internal Server Error**: If an error occurs during the creation or forking of the template.

GET: `/template/search/[templateID]`

This API endpoint allows users to retrieve a code template by its unique ID. The request must specify the template ID in the query parameters. The endpoint returns the full details of the code template if found.

Query Parameters

Name	Type	Description

templateID	number (Required)	The unique ID of the code template to be retrieved.

Example request:

```
GET .../api/template/search/1
```

Response (JSON)

- **200 Success: Returns the code template object associated with the given ID, with the following fields:**
  - id: The template's unique identifier.
  - code: The source code of the template.
  - title: The title of the template.
  - explanation: The description or explanation for the template.
  - author: The author's **username**.
  - tags: An array of string tags associated with the template.

  Example success response:

```
{
  "id": 1,
  "code": "function example() { return 'Hello World'; }",
  "title": "Example Template",
  "explanation": "This template prints 'Hello World'.",
  "author": "john_doe",
  "tags": ["javascript", "example"]
}
```

- **400 Bad Request**: If the id is missing from the request.
- **404 Not Found**: If the specified code template does not exist.
- **405 Method Not Allowed**: If a method other than GET is used.

- **500 Internal Server Error**: If an error occurs while fetching the template.

GET: `/template/search`

This API endpoint allows users to search for code templates based on various criteria, such as title, explanation, tags, or user ID. The results include the code template details, associated tags, and the author.

Query Parameters

Name	Type	Description
titleContains	string (Optional)	Filters templates where the title contains the specified string. The search is case-insensitive.
explanationContains	string (Optional)	Filters templates where the explanation contains the specified string. The search is case-insensitive.
tags	array of string (Optional)	Filters templates that are associated with **all** the specified tags.
userID	number (Optional)	Filters templates that are created by the specified user (author).
pageNumber	number (Optional)	Specifies the page of results to retrieve. Starts at 1 for the first page.
pageSize	number (Optional)	Defines the number of items to return per page.

Example request:

```
GET .../api/template/search?titleContains=example&tags=javascri
```

Response (JSON)

- **Success 200**: Returns a list of code templates that match the provided filters. Each template object includes:
  - id: The template's unique identifier.

- code: The source code of the template.
- title: The title of the template.
- explanation: The description or explanation for the template.
- author: The author's **username**.
- tags: An array of string tags associated with the template.

Example success response:

```
[
    {
        "id": 1,
        "code": "function example() { return 'Hello World'; }
        "title": "Example Template",
        "explanation": "This template prints 'Hello World'.",
        "language": "Javascript",
        "authorID": 1,
        "forkID": null,
        "tags": [
            "example"
        ],
        "author": {
            "username": "user1"
        }
    }
]
```

- **400 Bad Request**: If the query parameters are invalid.
- **405 Method Not Allowed**: If a method other than GET is used.
- **500 Internal Server Error**: If an error occurs during the search process.

Code Execution (Emily)

POST: `/code/execute`

This API endpoint allows users to submit code in various programming languages for compilation and execution on the server. Supported languages include C, C++, Java, Python, and JavaScript. The endpoint handles compilation or interpretation, executes the code, and returns the resulting output (stdout) or errors (stderr).

Request Body (JSON)

Name	Type	Description
code	string (Required)	The source code in the specified language to be compiled and executed.
language	string (Required)	Specifies the programming language of the provided code. Valid options: 'c', 'cpp', 'java', 'python', 'javascript'.
stdin	string (Optional)	Input data to be provided to the program during execution.

Example request body:

```
{
  "code": "#include <stdio.h>\nint main() { printf(\"Hello, Wor
  "language": "c",
  "stdin": ""
}
```

Response (JSON)

- **200 Success**: Returns the result of the executed code.

  Response Body:

Field	Type	Description
return	number	The return code of the execution. A value of 0 indicates success.

stdout	string	The output from the program execution (standard output).
stderr	string	Any error messages generated during the compilation or execution of the program (standard error).

Example success response:

```
{
  "return": 0,
  "stdout": "Hello, World!\n",
  "stderr": ""
}
```

- **400 Bad Request**: If the input is invalid, such as missing code or unsupported language.
- **401 Unauthorized**: If the user is not authenticated (if required by the environment or system policy).
- **403 Forbidden**: If the user does not have permission to execute the given command (if permissions are enforced).
- **405 Method Not Allowed**: If a method other than POST is used.
- **500 Internal Server Error**: If an error occurs during code compilation or execution. This may include a compilation error or runtime error within the program, with details provided in the stderr.

Posts (Benson + Kevin)

Our posts system treats posts, comments, and replies to those comments(replies) as pieces of **content**. All **content** can be voted on, reported or hidden. Each content type has specific additional fields(e.g. post has an additional title field)

POST: `/post/search` (Kevin)

Searches through all posts, returning paginated and filtered results

Request Body (JSON)

Name	Data Type	Description
title	string(Optional)	Text that should be contained in the post's title
content	string(Optional)	Text that should be contained in the post content
tags	list[string](Optional)	Tags that the post should contain
templates*	List[string or int](Optional)	Templates that we want the post to mention. Each item in the list represents a template ID. eg. [1, 2] or [”1”, “2”]
fromUser	bool(Optional)	whether to only show posts from the authenticated user(if logged in)
showHidden	bool(Optional)	whether to show hidden posts(if logged in as admin)
pageNumber	int(Optional, default 1)	page number, for pagination
pageSize	int(Optional, default 10)	page size, for pagination
sortMode	string in [mostUpvoted, mostDownvoted, mostReported] (Optional, default creation time)	Sorting method

*NOTE FOR TEMPLATES: “mentioning” a template in a post will be done by using @<templateID> in the content text. We plan to add hyperlinks to these templates in the frontend, and do not specifically validate that these links will lead to an actual template(e.g. on the web, there can be broken hyperlinks.

Response Body (JSON)

Name	Data Type	Description
error	string	Error message from the

Name	Data Type	Description
		server, for failed requests
content	list[PostDTO]	list of post data, if any were found

PostDTO:

Name	Data Type	Description
contentID	int	id of the piece of content
title	string	title of the post
_count	{comments: int}	number of comments
content	ContentDTO	information about the content of the post
tags	list[TagDTO]	information about the tags

ContentDTO

Name	Data Type	Description
text	string	text of the content
creationTime	datetime	creation time
author	{username: string, id: int}	Data about the author
_count	{upvotes: int, downvotes: int, reports?: int}	number of upvotes/downvotes, and reports if the searching user is an admin

TagDTO

Name	Data Type	Description
tag	{name: string}	name of the tag

Sample Request Body(JSON)

```
{
    "title": "",
    "content": "",
```

```
    "tags": ["tag1", "tag2"],
    "templates": [],
    "fromUser": false,
    "showHidden": true,
    "pageNumber": 1,
    "pageSize": 5,
    "sortMode": "mostUpvoted"
}
```

Sample Response Body (JSON)

```
[
    {
        "contentID": 1,
        "title": "How to create a C program",
        "content": {
            "text": "Just do it !",
            "creationTime": "2024-10-19T22:30:53.533Z",
            "author": {
                "username": "user1",
                "id": 1
            },
            "_count": {
                "upvotes": 1,
                "downvotes": 0,
                "reports": 0
            }
        },
        "tags": [
            {
                "tag": {
                    "name": "tag1"
                }
            },
        ],
```

```
        "_count": {
            "comments": 2
        }
    }
]
```

GET: `/post/search/[postID]` (Benson)

This API endpoint allows users to retrieve a post by its unique postID. The request must specify the post ID in the query parameters. The endpoint returns the full details of the post if found. The posts, comments, and replies associated with it will be hidden if they are set to isHidden and the user cannot access them.

Query Parameters

Name	Type	Description
postID	number (Required)	The unique ID of the post to be retrieved. Note that this is not the contentID, which is used in other endpoints

Example request:

```
GET .../api/post/search/1
```

Response (JSON)

- **200 Success:** Returns the post object associated with the given ID.

Name	Data Type	Description
error	string	Error message from the server, for failed requests
content	list[PostDTO]	list of post data, if any were found

PostDTO:

Name	Data Type	Description
contentID	int	id of the piece of content
title	string	title of the post
_count	{comments: int}	number of comments
content	ContentDTO	information about the content of the post
tags	list[TagDTO]	information about the tags
comments	list[CommentsDTO]	Information about the comments

ContentDTO

Name	Data Type	Description
text	string	text of the content
creationTime	datetime	creation time
author	{username: string, id: int}	Data about the author
_count	{upvotes: int, downvotes: int, reports?: int}	number of upvotes/downvotes, and reports if the searching user is an admin

TagDTO

Name	Data Type	Description
tag	{name: string}	name of the tag

CommentsDTO

Name	Data Type	Description
id	int	id of the comments
contentID	int	id of the content
postID	int	id of the post/reply this comment belongs to
content	ContentDTO	information about the content of the comment

ReplyDTO

Name	Data Type	Description
id	int	id of the reply
contentID	int	id of the content
commentID	int	id of the comment this reply belongs to
content	ContentDTO	information about the content of the comment


```
{
    "contentID": 1,
    "title": "User 2 post 1",
    "content": {
        "text": "post 1 @1 @4",
        "creationTime": "2024-10-22T04:00:52.996Z",
        "author": {
            "username": "user1",
            "id": 1
        },
        "_count": {
            "upvotes": 0,
            "downvotes": 0
        }
    },
    "tags": [
        {
            "tag": {
                "name": "tag1"
            }
        },
        {
            "tag": {
                "name": "tag3"
            }
```

```
        }
    ],
    "_count": {
        "comments": 1
    },
    "comments": [
        {
            "id": 4,
            "contentID": 7,
            "postID": 1,
            "content": {
                "text": "User 3 comment 4!",
                "author": {
                    "username": "user3"
                },
                "creationTime": "2024-10-22T17:59:17.689Z",
                "isHidden": true
            },
            "replies": []
        }
    ]
}
```


- **400 Bad Request**: If the id is missing from the request.
- **404 Not Found**: If the specified post does not exist or is hidden without admin access.
- **405 Method Not Allowed**: If a method other than GET is used.
- **500 Internal Server Error**: If an error occurs while fetching the post.

PUT: `/post/save/[contentID]` (Benson)

Updating the post with contentid [contentID]. Bearer authentication is required

Request Body (JSON)

Name	Type	Description
title	string (Optional)	The title of the post
description	string (Optional)	The post description or content.
tags	list[string] (Optional)	A list of tags that are related to the post

Sample Request Body (JSON)

```
{
    "title": "How to create a C program",
    "description": "Just do it !",
    "tags": ["C", "App"]
}
```

Response (JSON)

- 200 **Success**: Returns the result of the executed code.

Response Body:

Name	Type	Description
id	number	The id of the post
contentID	number	The contentID of the post
title	string	The title of the post

Example Success Response

```
{
    "id": 1,
    "contentID": 1,
    "title": "How to create a C program"
}
```

- **400 Bad Request**: If the input is invalid, such as title, description, and postID missing.

- **401 Unauthorized**: If the user is not authenticated (if required by the environment or system policy).
- **403 Forbidden**: If the user does not have permission to execute the given command (if permissions are enforced).
- **405 Method Not Allowed**: If a method other than PUT is used.
- **500 Internal Server Error**: If an error occurs while saving the post.

POST: `/post/save` (Benson)

Save a new post. Bearer authentication is required

Request Body (JSON)

Name	Type	Description
title	string (Required)	The title of the post
description	string (Required)	The post description or content.
tags	list[string] (Optional)	A list of tags that are related to the post

Sample Request Body

```
{
    "title": "How to create an Web App",
    "description": "You start by initializing a NextJS app",
    "tags": ["Test", "NextJS"]
}
```

Response (JSON)

- 200 **Success**: Returns the result of the executed code.

Response Body:

Name	Type	Description
id	number	The id of the post

contentID	number	The contentID of the post
title	string	The title of the post

Example Success Response

```
{
    "id": 6,
    "contentID": 9,
    "title": "How to create an Web App"
}
```

- **400 Bad Request**: If the input is invalid, such as title or description missing.
- **401 Unauthorized**: If the user is not authenticated (if required by the environment or system policy).
- **403 Forbidden**: If the user does not have permission to execute the given command (if permissions are enforced).
- **405 Method Not Allowed**: If a method other than PUT is used.
- **500 Internal Server Error**: If an error occurs while saving the post

POST: `/post/comment/create/[contentID]` (Benson)

Creating the comment for the parent (post or comment) with contentid [contentID]. Bearer authentication is required

Request Body (JSON)

Name	Type	Description
content	string (Required)	The content of the comment

Sample Request Body (JSON)

```
{
    "content": "This reply is awesome!"
}
```

Response (JSON)

- 200 **Success**: Returns the result of the executed code.

Response Body:

Name	Type	Description
id	number	The id of the post
contentID	number	The contentID of the post
postID	number	The id of the post
content.id	number	The contentID of the comment
content.text	string	The content of the comment
content.authorID	number	The user ID of the comment author
content.creationTime	string	The creation timestamp of the comment
content.isHidden	boolean	The boolean of whether a content is hidden or not

Example Success Response

```
{
    "id": 2,
    "contentID": 10,
    "postID": 1,
    "content": {
        "id": 10,
        "text": "This reply is awesome!",
        "authorID": 1,
        "creationTime": "2024-10-20T02:45:15.213Z",
        "isHidden": false
    }
}
```

- **400 Bad Request**: If the input is invalid, such as content missing.
- **401 Unauthorized**: If the user is not authenticated (if required by the environment or system policy).

- **403 Forbidden**: If the user does not have permission to execute the given command (if permissions are enforced).
- **405 Method Not Allowed**: If a method other than POST is used.
- **500 Internal Server Error**: If an error occurs while creating the comment.

POST: `/post/vote/[contentID]` (Benson)

Vote on content with contentID. Note that this can be a post, comment or reply.
Bearer authentication is required.
Note that each user can only vote each piece of content once

Request Body (JSON)

Name	Type	Description
state	number	The state of the vote (1 for upvote, -1 for downvote)

Sample Request Body

```
{
    "state": 1
}
```

Response (JSON)

- 200 **Success**: Returns the result of the executed code.

Response Body:

Name	Type	Description
id	number	The id of the post
userID	number	The id of the user
contentID	number	The contentID of the content (Post, comment, or reply)
upvote.id	number	The id of the vote
upvote.userID	number	The id of the user

upvote.contentID	number	The contentID of the voted content

Example Success Response

```
{
    "id": 1,
    "userID": 1,
    "contentID": 1,
    "upvote": {
        "id": 1,
        "userID": 1,
        "contentID": 1
    }
}
```

- **400 Bad Request**: If the input is invalid, such as state missing.
- **401 Unauthorized**: If the user is not authenticated (if required by the environment or system policy).
- **403 Forbidden**: If the user does not have permission to execute the given command (if permissions are enforced).
- **405 Method Not Allowed**: If a method other than POST or PUT is used.
- **500 Internal Server Error**: If an error occurs while creating a vote.

POST: `/post/report/[contentID]`

Report a piece of content with id [contentID]. Note that this can reference a post, comment or reply to a comment. Bearer authorization is required. Note that users cannot un-report a piece of content.

Request Body (JSON)

Name	Data Type	Description
reason	string(optional)	Reason for the report

Response Body (JSON)

Name	Data Type	Description
message	string	Message from the server, for successful requests
error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

```
{
    "reason": "This post has explicit content."
}
```

Sample Response Body (JSON)

```
{
        "message": "Successfully reported the post"
}
```

POST: `/post/hide/[contentID]`

Hide a piece of content with id [contentID]. Note that this can reference a post, comment or reply. Bearer authentication on an admin account is required.

Request Body (JSON)

Name	Data Type	Description
hide	boolean(optional)	true to hide the post, false to show the post. Defaults to true

Response Body (JSON)

Name	Data Type	Description
message	string	Message from the server, for successful requests

error	string	Error message from the server, for failed requests

Sample Request Body(JSON)

```
{
        "hide": true
}
```

Sample Response Body (JSON)

```
{
        "message": "Successfully hid the post"
}
```