DEV 1

Arianne Arcebal

USERS MICROSERVICE

users.py

`POST` `/registration`

Purpose
Creates a new user.

Description
Accesses the users table in the database to create a new user. The user-provided email and username must not already exist in the database. The user-provided password is hashed before being stored.

Parameters
The following parameters are required to successfully register. It accepts **JSON** format.

Parameter	Description
`email`	The user's email used to sign in to the account. Must be unique.
`password`	Chosen password used to access the account.
`username`	The user's display name. Must be unique.

Quick Example

```
curl -X POST -H 'Content-Type: application/json' -d '{"email":"ari@test.com", "password":"password", "name":"ari"}' http://127.0.0.1:5000/registration
```

This creates an account for a user with the email `ari@test.com`, password `password`, and username `ari`.

`GET` `/users/view`

Purpose
Views a user by id.

Description
Displays the username of a given id.

Parameters

The following parameters are required to view a specific user. It is passed through the **URL**.

Parameter	Description
`id`	The id of the user account the user wishes to view.

Quick Example

```
curl http://127.0.0.1:5000/users/view?id=1
```

Displays the username for the user with an `id` of `1`.

`PUT` `/users/settings`

Purpose

Changes an existing user's password.

Description

Requires authentication. Successful authentication allows the user to change their current password. The new password is hashed before being stored in the database. Unsuccessful authentication does not change the password.

Parameters

The following parameters are required to change the user's password. It accepts **JSON** format.

Parameter	Description
`new-password`	The new password the user wishes to change from their current password

Quick Example

```
curl -X PUT -u ari@test.com:password -H 'Content-Type:
application.json' -d '{"new-password":"12345"}'
http://127.0.0.1:5000/users/settings
```

This changes the user password for `ari@test.com` from `password` to `12345`.

`DELETE` `/users/settings`

Purpose
Deletes an existing user.

Description
Requires authentication. Successful authentication allows the user to delete their own account. Unsuccessful authentication does not delete the user.

Quick Example
```
curl -X DELETE -u ari@test.com:password
http://127.0.0.1:5000/users/settings
```

This deletes the user `ari@test.com` from the database.

ARTICLES MICROSERVICE

articles.py

`POST` `/articles/new`

Purpose
Creates a new article.

Description
Requires authentication. Successful authentication allows the user to create a new article. The creation date and time are automatically taken from the system. The author name is the username of the authenticated user. Title and content are provided by the user. Unsuccessful authentication does not create a new article.

Parameters
The following parameters are required to create a new article. It accepts **JSON** format.

Parameter	Description
`title`	The title of the article.
`content`	The body of the article.

Quick Example

```
curl -X POST -u ari@test.com:password -H 'Content-Type: application/json' -d '{"title":"Hello World!", "content":"Lorem ipsum dolor sit amet."}' http://127.0.0.1:5001/articles/new
```

Assuming the username for `ari@test.com` is `ari`, this creates an article by the author `ari` with the title `Hello World!` and content `Lorem ipsum dolor sit amet.` The date and time will vary depending on the current system date and time.

`GET` `/articles/view`

Purpose
Views one article.

Description
Displays the data of one an article specified by its id. The results are displayed in a JSON format.

Parameters
The following parameters are required to view a specific article. It is passed through the **URL**.

Parameter	Description
`id`	The id of the article the user wishes to view.

Quick Example
```
curl http://127.0.0.1:5001/articles/view?id=1
```

This displays the article with an `id` of `1`.

`PUT /articles/view`

Purpose
Edits an existing article.

Description
Requires authentication. Successful authentication allows the user to edit an article specified by its id if it was posted by the authenticated user. The date and time is automatically taken from the system to update when the article was last modified. The new title and content are provided by the user. Unsuccessful authentication or mismatching article author and username does not allow the user to edit an article.

Parameters
The following articles are required to edit a specific article. It is passed through the **URL**.

Parameter	Description
`id`	The id of the article the user wishes to edit.

Quick Example
```
curl -X PUT -u ari@test.com:password -H 'Content-Type: application/json' -d '{"title":"Editing test.", "content":"This is the new content."}' http://127.0.0.1:5001/articles/view?id=1
```

This edits the article with an `id` of `1` and changes the title to `Editing test.` and content to `This is the new content.`, if the user who posted it was `ari@test.com`. The last modified date and time varies depending on the current system date and time.

`DELETE` /articles/view

Purpose
Deletes an existing article.

Description
Requires authentication. Successful authentication allows the user to delete an article specified by its id if it was posted by the authenticated user. Unsuccessful authentication or mismatching article author and username does not allow the user to delete the article.

Parameters
The following articles are required to edit a specific article. It is passed through the **URL**.

Parameter	Description
`id`	The id of the article the user wishes to edit.

Quick Example
```
curl -X DELETE -u ari@test.com:password
http://127.0.0.1:5001/articles/view?id=1
```

This deletes the article with an `id` of `1` if it was posted by `ari@test.com`.

`GET` /articles/view/all

Purpose
Views all articles.

Description
Displays all articles stored in the database. The results are displayed in a JSON format and is in order by id.

Quick Example
```
curl http://127.0.0.1:5001/articles/view/all
```

This displays all articles.

`GET` /articles/view/recent

Purpose
Views most recent articles by creation date.

Description
Displays the *n* most recent articles. The results are displayed in a JSON format and is listed in order by creation date.

Parameters
The following articles are required to view the *n* most recent articles. It is passed through the **URL**.

Parameter	Description
`amount`	The number of recent articles the user wishes to view

Quick Example
```
curl http://127.0.0.1:5001/articles/view/recent?amount=3
```

Displays the 3 most recent articles according to creation date.

`GET` `/articles/view/recent/meta`

Purpose
Views metadata of the most recent articles by creation date.

Description
Displays the metadata for the *n* most recent articles. The results are displayed in a JSON format and is listed in order by creation date.

Parameters
The following articles are required to view the *n* most recent articles. It is passed through the **URL**.

Parameter	Description
`amount`	The number of recent articles the user wishes to view

Quick Example
```
curl http://127.0.0.1:5001/articles/view/recent/meta?amount=3
```

Displays the metadata for the 3 most recent articles according to creation date.

DEV 2

Henrik Vahanian

TAGS MICROSERVICE

tags.py

`POST` `/tags/new`

Purpose
Add tags for a new URL

Description
Requires authentication. Accesses the tags table in the database to add new tag to a new URL. If the article does not exist, the new article is added with the new tag. If the article exists, will return status 409. If the article exists and user is not the owner of article, will return status 403.

Parameters
The following parameters are required to successfully post a tag to a new URL. It accepts **JSON** format.

Parameter	Description
`id`	New URL id
`category`	New tag to add to URL id

Quick Example

```
curl -X POST \
    'http://localhost:5003/tags/new?id=36' \
    -H 'Authorization: Basic aGVsbG9AZ21haWwuY29tOmhlbGxv' \
    -H 'Content-Type: application/json' \
    -H 'Postman-Token: 4f6e1add-13c4-4a4b-b537-b160668edee5' \
    -H 'cache-control: no-cache' \
    -d '{"category": "nature"}'
```

This creates a tag `nature` for the new URL with id `36`.

`POST` `/tags/exists`

Purpose
Add tags to an existing URL.

Description
Requires authentication. Accesses the tags table in the database to add new tag to an existing URL. If the URL does not exist, it will return a 404 status code. If the URL exists, it will add the new tag to the URL and return status 409. If the URL exists and user is not the owner of article, will return status 403.

Parameters
The following parameters are required to successfully post a tag to an existing URL. It accepts **JSON** format.

Parameter	Description
`id`	Existing URL id
`category`	New tag to add to URL id

Quick Example
```
curl -X POST \
    'http://localhost:5003/tags/new?id=2' \
    -H 'Authorization: Basic aGVsbG9AZ21haWwuY29tOmhlbGxv' \
    -H 'Content-Type: application/json' \
    -H 'Postman-Token: d7087e94-90f6-44e7-8d65-6fa4d0f8b01b' \
    -H 'cache-control: no-cache' \
    -d '{"category": "nature"}'
```

This adds the tag `nature` to the existing URL with id `2`

`DELETE /tags/delete`

Purpose
Remove one or more tags from an individual URL

Description
Requires authentication. Successful authentication allows the user to delete tags if they are the owner of the URL. Unsuccessful authentication does not allow the user to delete the tag.

Parameters
The following parameters are required to successfully remove a tag from a URL. It accepts **JSON** format.

Parameter	Description
`id`	URL id
`category`	Tag to delete from URL id

Quick Example

```
curl -X DELETE \
        'http://localhost:5003/tags/delete?id=2' \
        -H 'Authorization: Basic aGVsbG9AZ21haWwuY29tOmhlbGxv' \
        -H 'Content-Type: application/json' \
        -H 'Postman-Token: 922a40ef-3ecc-4595-9334-5efe512b590d' \
        -H 'cache-control: no-cache' \
        -d '{"category": "nature"}'
```

This deletes the category `nature` from the URL id `2`

`GET /tags`

Purpose
Retrieve the tags for an individual URL.

Description
Lists all tags that pertain to the desired article.

Parameters
The following parameters are required to successfully retrieve tags from a URL. It accepts **JSON** format.

Parameter	Description
`id`	URL id

Quick Example

```
curl -X GET \
        'http://localhost:5003/tags?id=5' \
        -H 'Content-Type: application/json' \
```

```
    -H 'Postman-Token: 5753900f-c955-4f3f-9c42-28de2aae9310' \
    -H 'cache-control: no-cache'
```

This lists all tags that pertain to URL id `5`

`GET /tags/articles`

Purpose
Retrieve a list of URLs with a given tag.

Description
Lists URLs that pertain to the selected tag.

Parameters
The following parameters are required to successfully retrieve articles pertaining to a tag. It accepts **JSON** format.

Parameter	Description
`category`	Tag name

Quick Example

```
curl -X GET \
    http://localhost:5003/tags/articles \
    -H 'Content-Type: application/json' \
    -H 'Postman-Token: d4cbb978-3934-442c-8231-274fcb51266a' \
    -H 'cache-control: no-cache' \
    -d '{"category": "nature"}'
```

This lists all the URLs that pertain to the selected tag.

COMMENTS MICROSERVICE

comments.py

`POST` `/comments/new`

Purpose
Post a new comment on a URL.

Description
The system will post a comment to the desired URL. If a user is logged in, that user will be assigned the author of that comment. If no user is logged in, the author of the comment will be Anonymous. If no Basic Auth is used at all, the author of the comment will be Anonymous.

Parameters
The following parameters are required to successfully post a new comment. It accepts **JSON** format.

Parameter	Description
`id`	URL id
`content`	The content of the comment

Quick Example

```
curl -X POST \
    'http://localhost:5002/comments/new?id=2' \
    -H 'Authorization: Basic aGVsbG9AZ21haWwuY29tOmhlbGxv' \
    -H 'Content-Type: application/json' \
    -H 'Postman-Token: 83c4d450-dbc2-4435-85e5-47a486caa564' \
    -H 'cache-control: no-cache' \
    -d '{"content": "This is a comment"}'
```

This posts the comment `This is a comment` to the desired URL id `2`

`DELETE` `/comments/delete`

Purpose
Delete an individual comment.

Description
Requires authorization. The system will check if the user is the author of the comment before it allows for deletion. If the author of the comment is Anonymous, the system will check if the user is the author of the article before it allows for deletion.

Parameters
The following parameters are required to successfully delete a comment. It accepts **JSON** format.

Parameter	Description
`commentid`	Comment id

Quick Example
```
curl -X DELETE \
     'http://localhost:5002/comments/delete?id=2' \
     -H 'Authorization: Basic aGVsbG9AZ21haWwuY29tOmVsbG8=' \
     -H 'Postman-Token: 64c00797-8f0f-4c90-8d69-0bf21283bcd0' \
     -H 'cache-control: no-cache'
```

This deletes the comment id `2`.

`GET` `/comments/count`

Purpose
Retrieve the number of comments on a given URL.

Description
Will return the number of comments under the desired URL.

Parameters
The following parameters are required to successfully retrieve the number of comments under a given URL.

Parameter	Description
`id`	URL id

Quick Example

```
curl -X GET \
  'http://localhost:5002/comments/count?id=2' \
  -H 'Postman-Token: 2aa5e0e8-d1af-4c85-8f01-ce2cc37ca90a' \
  -H 'cache-control: no-cache'
```

This retrieves the number of comments under URL id `2`.

`GET` `/comments`

Purpose
Retrieve the n most recent comments on a URL.

Description
The system will take in the URL id and list the recent desired number of comments pertaining to that URL.

Parameters
The following parameters are required to successfully retrieve the n most recent comments under a given URL.

Parameter	Description
`id`	URL id
`amount`	Number of recent comments to list

Quick Example

```
curl -X GET \
  'http://localhost:5002/comments?id=2' \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 388e91dd-a69b-4fe6-9fa5-087e86481eb1' \
  -H 'cache-control: no-cache' \
  -d '{"amount": "2"}'
```

This retrieves the `2` most recent comments under URL id `2`

CUSTOM CLI COMMANDS

Users.py - register

Purpose
To register a user as root (testing purposes).

Description
Root user is able to register users to the database through the command line.

Parameters
The following parameters are required to successfully register a new user to the database.

Parameter	Description
`name`	Name of the user
`email`	Email address of the user
`password`	Password of the user used to login

Quick Example
`export FLASK_APP=users.py`
`flask register Bob bobby@gmail.com myPass`

This exports the `users.py` api and `flask` is used to call the definition register followed by the parameters `Bob bobby@gmail.com myPass` to register the new user

Users.py - delete

Purpose
To delete a user as root (testing purposes).

Description
Root user is able to delete users from the database through the command line.

Parameters
The following parameters are required to successfully delete a user from the database.

Parameter	Description
`email`	Email address of the user to delete

Quick Example
```
export FLASK_APP=users.py
flask delete bobby@gmail.com
```

This exports the users.py api and flask is used to call the definition delete followed by the parameter bobby@gmail.com to delete a user.

Articles.py - post

Purpose
For the root user to post articles to the database (for testing purposes).

Description
Root user is able to post articles to the database through the command-line.

Parameters
The following parameters are required to successfully post a new article to the database.

Parameter	Description
`title`	Title of the new article
`content`	Content contained in the new article

Quick Example
```
export FLASK_APP=articles.py
flask post aTitle SomeContent
```

This exports the articles.py api and flask is used to call the definition post followed by the parameters aTitle SomeContent to post the article to the database.

Articles.py - delete

Purpose
For the root user to delete articles from the database (for testing purposes).

Description
Root user is able to delete articles from the database through the command-line.

Parameters
The following parameters are required to successfully delete an article from the database.

Parameter	Description
`id`	The id of the article in the database

Quick Example

```
export FLASK_APP=articles.py
flask delete 3
```

This exports the articles.py api and flask is used to call the definition delete followed by the parameter 3 to delete the article from the database.

Tags.py - new

Purpose
For the root user to add tags to articles in the database (for testing purposes).

Description
Root user is able to add tags to articles in the database through the command-line.

Parameters
The following parameters are required to successfully add a tag to an article in the database.

Parameter	Description

`category`	Name of the tag to add
`id`	Article id to add the tag to

Quick Example
export FLASK_APP=tags.py
flask new nature 3

This exports the tags.py api and flask is used to call the definition new followed by the parameters nature 3 to add the tag to the article in the database.

Tags.py - delete

Purpose
For the root user to delete tags from articles in the database (for testing purposes).

Description
Root user is able to delete tags from articles in the database through the command-line.

Parameters
The following parameters are required to successfully delete a tag from an article in the database.

Parameter	Description
`category`	Name of the tag to delete
`id`	Article id to delete the tag from

Quick Example
export FLASK_APP=tags.py
flask delete nature 3

This exports the tags.py api and flask is used to call the definition delete followed by the parameters nature 3 to delete the tag from the article in the database.

Comments.py - post

Purpose
For the root user to post a comment in the database (for testing purposes).

Description
Root user is able to post comments under specified articles in the database through the command-line.

Parameters
The following parameters are required to successfully post a comment to an article in the database.

Parameter	Description
`id`	Article id to post comment to
`content`	The data that the comment will contain

Quick Example
```
export FLASK_APP=comments.py
flask post 5 aComment
```

This exports the `comments.py` api and `flask` is used to call the definition `post` followed by the parameters `5 aComment` to post a comment to the article in the database.

Comments.py - delete

Purpose
For the root user to delete a comment in the database (for testing purposes).

Description
Root user is able to delete comments under specified articles in the database through the command-line.

Parameters
The following parameters are required to successfully delete a comment from an article in the database.

Parameter	Description
`id`	Comment id to delete

Quick Example

```
export FLASK_APP=comments.py
flask delete 5
```

This exports the `comments.py` api and `flask` is used to call the definition `delete` followed by the parameters `5` to delete a comment from the article in the database.

OPS

Kiren Syed

Procfile

Purpose

The purpose of this document is to declare the commands that should be run to enable all the microservices to run at the same time.

Instructions to deploy on Tuffix

-upload the procfile into the directory you have your programs in
-name the file: Procfile

-install **gem install foreman**
-after uploading procfile and installing foreman
-run the command: **foreman start**

Tavern Test Scenarios

Users Microservice

First test

See if an invalid user can change the password

Goal:

This microservice will return a status code of 401 because an invalid user cannot change a password.

Return:

401 status code

Second Test

Create a new user

Goal:

This microservice will return a status code of 200 because the proper steps were followed to create a user.

Return:

200 status code

Third Test

Change a valid user's password

Goal:

This microservice will return a status code of 200 once the user has been authenticated and has changed their password.

Return:

200 status code

Fourth Test

Delete a valid user

Goal

To delete an existing user. This will return a status code of 200.

Return

200 status code

Fifth Test

Try to change the password for deleted user

Goal

The goal of this test is to ensure deleted users can no longer change their passwords. This will return a status code of 401.

Return

401 status code

Articles Microservice

First test

attempt to post an article without authenticating

Goal:

This microservice will return a status code of 401 because one cannot post an article without authenticating

Return:

401 status code

Second Test

attempt to post an article using the wrong password

Goal:

This microservice will return a status code of 401 because one cannot post an article without authenticating

Return:

401 status code

Third Test

post an article successfully

Goal:

This microservice will return a status code of 201 once the user has been authenticated, the user can post articles.

Return:

201 status code

Fourth Test

retrieve the newly posted article

Goal

To get the newest article that has been posted. This will return a status code of 200

Return

200 status code

Fifth Test

check that the newly posted article is most recent

Goal

To show all the articles are being posted chronologically

Return

200 status code

Tags Microservice

First test

Add another tag to the article

Goal:

This microservice will return a status code of 201. The goal of this test is to check if one can add more tags to an existing articles.

Return:

201 status code

Second Test

Delete one of the tags from the article

Goal:

This microservice will return a status code of 200. The goal is to test if tags of an article can be deleted.

Return:

200 status code

Third Test

Add a tag to an article that doesn't exist

Goal:

This microservice will return a status code of 404. The goal of this is to test if an individual can add a tag to a non-existing article.

Return:

404 status code

Comments Microservice

First test

Post an anonymous comment on an article

stages:

Goal:

This microservice will return a status code of 201. The goal of this test is to check if one can add more tags to an existing articles.

Return:

201 status code

Second Test

Post an authenticated comment on an article

Goal:

This microservice will return a status code of 200. The goal is to test if tags of an article can be deleted.

Return:

201 status code

Third Test

Check that comments on the article were returned in order

Goal:

This microservice will return a status code of 404. The goal of this is to test if an individual can add a tag to a non-existing article.

Return:

200 status code