Microservices API Documentation

users.py, articles.py, tags.py, comments.py, rss.py

CPSC 476

Arianne Arcebal
Kiren Syed
Henrik Vahanian

Table of Contents

.

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 \
  http://localhost:5000/registration \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 233f59f5-51ac-456d-a75f-efcd9ddd410a' \
  -H 'cache-control: no-cache' \
  -d '{"email":"ari@test.com", "password":"password",
"name":"ari"}'
```

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

PUT `/users/change-password`

Purpose
Changes an existing user's password.

Description

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 \
  http://localhost:5000/users/change-password \
  -H 'Authorization: Basic YXJpQHRlc3QuY29tOnBhc3N3b3Jk' \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 6e8d9f29-6213-48ac-9b8d-756a246303b0' \
  -H 'cache-control: no-cache' \
  -d '{"new-password": "12345"}'
```

This changes the password for the user currently signed in to `12345`.

DELETE `/users/delete-account`

Purpose
Deletes an existing user.

Description
Allows the user to delete their own account

Quick Example

```
curl -X DELETE -u ari@test.com:password
http://127.0.0.1:5000/users/delete-account
```

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

ARTICLES MICROSERVICE

articles.py

POST `/articles/new`

Purpose

Creates a new article.

Description

Allows the user to create a new article. The creation date and time are automatically taken from the system. The author name is the email 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
```

This creates an article by the author `ari@test.com` 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/<articleid>`

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
`articleid`	The uuid of the article the user wishes to view.

Quick Example
```
curl http://127.0.0.1:5001/articles/1
```

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

PUT `/articles/<articleid>`

Purpose
Edits an existing article.

Description
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
`articleid`	The uuiid 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/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/<articleid>`

Purpose
Deletes an existing article.

Description
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
`articleid`	The uuid 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/1
```

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

GET `/articles/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/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/recent?amount=3
```

Displays the 3 most recent articles according to creation date.

GET /articles/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/recent/meta?amount=3
```

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

TAGS MICROSERVICE

tags.py

POST `/articles/<articleid>/tagged`

Purpose
Add tags to a URL.

Description
Accesses the tags table in the database to add new tag to an existing URL. If the URL exists, it will add the new tag to the URL and return status 201. The parameter can be a list of categories

Parameters
The following parameters are required to successfully post a tag to an existing URL. It is passed through the **URL** and in **JSON** format.

Parameter	Description
`articleid`	Article uuid. Passed through URL.
`category`	New tag to add to URL id. Passed through JSON.

Quick Example

```
curl -X POST \
      'http://localhost:5003/articles/2/tagged \
      -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”, “wildlife”]}'
```

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

DELETE `/articles/<articleid>/tagged`

Purpose
Remove one or more tags from an individual URL

Description
Deletes the specified tags on the given article.

Parameters
The following parameters are required to successfully remove a tag from a URL. It is passed through the **URL** and in **JSON** format.

Parameter	Description
`articleid`	Article uuid. Passed through URL
`category`	Tag to delete from URL id. Passed through JSON

Quick Example

```
curl -X DELETE \
      'http://localhost:5003/articles/2/tagged' \
      -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 `/articles/<articleid>/tagged`

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 is passed through the **URL**.

Parameter	Description
`articleid`	Article uuid

Quick Example

```
curl -X GET \
      'http://localhost:5003/articles/5/tagged' \
```

```
    -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` `/tagged/<category>`

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 is passed through the **URL**.

Parameter	Description
`category`	Tag name

Quick Example

```
curl -X GET \
    http://localhost:5003/tagged/nature \
    -H 'Content-Type: application/json' \
    -H 'Postman-Token: d4cbb978-3934-442c-8231-274fcb51266a' \
    -H 'cache-control: no-cache' \
```

This lists all the URLs that have the tag `nature`

COMMENTS MICROSERVICE

comments.py

POST `/articles/<articleid>/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 set to Anonymous Coward.

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

Parameter	Description
`articleid`	Article uuid. Passed through URL.
`content`	The content of the comment. Passed through JSON

Quick Example

```
curl -X POST \
      'http://localhost:5002/articles/2/comments/new \
      -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 `/articles/<articleid>/comments/<commentid>`

Purpose
Delete an individual comment.

Description
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 Coward, the comment cannot be deleted.

Parameters
The following parameters are required to successfully delete a comment. It is passed through the **URL**.

Parameter	Description
`commentid`	Comment uuid
`articleid`	The uuid of the article the comment is in

Quick Example
```
curl -X DELETE \
      'http://localhost:5002/articles/5/comments/2' \
      -H 'Authorization: Basic aGVsbG9AZ21haWwuY29tOmVsbG8=' \
      -H 'Postman-Token: 64c00797-8f0f-4c90-8d69-0bf21283bcd0' \
      -H 'cache-control: no-cache'
```

This deletes the comment id `2` in the article with an id of `5`.

GET `/articles/<articleid>/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
`articleid`	Article uuid

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` `/articles/<articleid>/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. Comments are listed in reverse chronological order according to creation.

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

Parameter	Description
`articleid`	Article uuid
`amount`	Number of recent comments to list

Quick Example

```
curl -X GET \
      'http://localhost:5002/articles/5/comments?amount=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 `5`

RSS MICROSERVICE

rss.py

GET `/rss/recent`

Purpose
Gets the 10 most recent articles.

Description
Sends requests to the articles microservice to get the 10 most recent articles. Feed includes the article title, author, creation date, and URL.

Quick Example
```
curl http://localhost/rss/recent
```

Gets an RSS feed of the 10 most recent articles ordered by creation date.

GET `/rss/articles/<articleid>`

Purpose
Gets the full article and its data

Description
Sends requests to the articles microservice, comments microservice, and the tags microservice. Feed includes the article title, author, creation date, URL, article contents, categories, and the amount of comments.

Quick Example
```
curl http://localhost/rss/articles/5
```

Gets an RSS feed of the specified article.

GET `/rss/articles/<articleid>/comments`

Purpose
Gets the comments feed for a specified article.

Description
Sends requests to the comments microservice to get a list of all the comments in a specific article. Feed includes the article name, comment author, comment contents, and posting date.

Quick Example

```
curl http://localhost/rss/articles/5
```

Gets an RSS feed for the comments of the specified article.

NGINX

Purpose

Nginx is a Web Server that can be used as a reverse proxy and a load balancer. Here we will be using it to intercept requests from a browser and route it to the right server.

We will use it as a reverse proxy as well to send back responses from the server to the browser.

Load Balancing is setup to run 3 servers for each api on 3 different ports in case 1 or more server go down.

Configuration

Place `nginx-enabled` in `/etc/nginx/sites-enabled/` and start the Nginx service with the command `sudo service nginx restart`

Nginx is listening on port 80 therefore the client would not need to specify the port.

CUSTOM CLI COMMANDS

db.py

Assumptions

The following commands assume scyllaDB is configured and running along with the `db.py` file.

Database Initialization

The database can be initialized with the `init-db` flask CLI command. The schemas for the database can be found in the `cql` folder. It will create a keyspace called “blog” and use it for future queries, then create the tables for each microservice.

Quick Example

```
$ flask init-db
```

Initializes the database for all the microservices.

Loading Test Data

The database can be loaded with test data after it has been initialized. This is done with the `init-data` flask CLI command, followed by the name of the microservice. The schemas for the test data can be found in the `cql` folder.

Quick Example

```
$ flask init-data articles
```

Loads dummy data for the `articles` microservice.

Dropping the Database

The database can be dropped using the `drop-db` CLI command.

Quick Example

```
$ flask drop-db
```

TESTING

ScyllaDB

To test ScyllaDB we use a Docker container to run it in. The following commands were used to install Docker, enable the service, and allow the user to run Docker commands:

```
$ sudo apt install --yes docker.io
$ sudo service docker start
$ sudo usermod -aG docker $USER
```

Now we can run Docker and start a single instance of ScyllaDB while limiting the RAM to 1GB for systems that do not have sufficient RAM.

```
$ docker run --name scylla -d scylladb/scylla --smp 1 --memory
1G --overprovisioned 1 --developer-mode 1 --experimental 1
```

We are able to check if ScyllaDB is running using the `nodetool` command.

```
$ docker exec -it scylla nodetool status
```

Results:

Load Testing

We use Siege to do load testing to see the number of transactions made in 1 minute with 25 concurrent users. The first test will be done without HTTP caching and the second test will be done with HTTP caching.

In order to load test with Siege, all the APIs need to be up and running and Siege needs to know the links to the APIs which we specify in `urls.txt.` The flag `-l` is used to log the results.

```
$ siege -f /usr/local/etc/urls.txt -l
```

[NO HTTP Caching] Command Line Results:

```
Lifting the server siege...
Transactions:                  64089 hits
Availability:                 100.00 %
Elapsed time:                  59.12 secs
Data transferred:              27.70 MB
Response time:                  0.02 secs
Transaction rate:            1084.05 trans/sec
Throughput:                     0.47 MB/sec
Concurrency:                   24.86
Successful transactions:       64089
Failed transactions:               0
Longest transaction:            0.10
Shortest transaction:           0.00
```

[NO HTTP Caching] Log File Results:

Date & Time, Trans, Elap Time, Data Trans, Resp Time, Trans Rate, Throughput, Concurrent, OKAY, Failed
2019-05-12 23:19:58, 64089, 59.12, 27, 0.02, 1084.05, 0.46, 24.86, 64089, 0

[HTTP Caching] Command Line Results:

```
Lifting the server siege...
Transactions:                 104470 hits
Availability:                 100.00 %
Elapsed time:                  59.35 secs
Data transferred:              27.54 MB
Response time:                  0.01 secs
Transaction rate:            1765.05 trans/sec
Throughput:                     0.53 MB/sec
Concurrency:                   24.82
Successful transactions:      104470
Failed transactions:               0
Longest transaction:            0.07
Shortest transaction:           0.00
```

[HTTP Caching] Log File Results:

Date & Time, Trans, Elap Time, Data Trans, Resp Time, Trans Rate, Throughput, Concurrent, OKAY, Failed
2019-05-13 23:18:06, 104470, 59.35, 27, 0.01, 1765.20, 0.53, 24.81, 104470, 0

Caching makes a big difference considering the APIs were able to make 104470 transactions with caching versus 64089 transactions without caching in about a minute with 25 concurrent users.

Caching HTTP requests

Modifying the microservices to enable conditional responses

Last modified headers were added to all GET routes in the users.py and comment.py code and these values were filled with the values of the last modified date provided by the articles.db or comments.db. In article.py file, if the last-modified value was not present then it's values were placed by the date created of the article.

Once the header was added it was then tested against the If-modified-since value[if that header was provided]. If the value of the if modified since was younger than the last modified header the program would return a status code of 304. If it was not it would return the data as requested.

Caching HTTP responses on the client

In the rss.py file caching was applied by using CacheControl.

Enabling authentication request caching

Authentication request caching was causing some trouble. The current set up for it is placing the proxy cache key in the nginx.conf and the proxy cache value in the auth location block in default.conf

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 201 because the proper steps were followed to create a user.

Return:

201 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

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

400 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

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

Resources Used

https://pythonprogramming.net/password-hashing-flask-tutorial/

https://www.programcreek.com/python/example/51515/flask.Response

https://tecadmin.net/get-current-date-time-python/

GitHub Link

https://github.com/kirnehv/CPSC-476