Project 1

Ruby Abutaleb

Andrew Gomez

Enrique Mendoza

Table of Contents

Articles

The Articles microservice allows the front end to manipulate articles by referencing certain routes. Using HTTP methods, Articles has various methods that allow creation, edits, deletion, and retrieval of articles or data from articles. The operations included in this microservice:

1. Posting a single article
    a. Route: '/article/new/<title>/<body>'
    b. HTTP Method: 'POST'
    c. Functionality: this python method first connects to the database and allocates a cursor to the connection. It then passes the article title and body to an internal variable. Calls get.request.authorization to retrieve username and password. checkAuth(username, password) is called to authenticate the user credentials before posting the article. If the credentials passed in are incorrect, a 409 error is thrown, if the credentials are correct, the variable with the new article info is inserted into the database and a 201 code is returned.

2. Retrieving a single article
    a. Route: '/article/<int: articleId>/title'
    b. HTTP Method: 'GET'
    c. Functionality: This method first connects to the database. Then it checks if the articleId passed in the route points to an existing article by searching the database. If not, error 404 is thrown, if it exists, the article title and body will be retrieved from the database and returned with an HTTP code 200.

3. Editing a single article and updating the timestamp
    a. Route: '/article/<int: articleId>/<title>/<body>'
    b. HTTP Method: 'PATCH'
    c. Functionality: After connecting to the database, this method passes the variables sent in the route, to a local variable. Calls get.request.authorization to retrieve username and password. Then the articleId is used to check if the article exists, if not, a 404 error is sent. If the article exists, the method runs checkAuth(username, password)

and then updates the article if the credentials are correct. If the credentials are incorrect, error code 409 is sent.

4. Deleting an existing article
    a. Route: '/article/<int:articleId>'
    b. HTTP Method: 'DELETE'
    c. Functionality: Once the method connects to the database, it checks if request.authorization exists, then checks the validity of the users credentials. The method then ensures an article of that articleId exists, returns 404 if it does not. If the article exists, it is deleted from the database.
5. Retrieving contents of n most recent articles
    a. Route: '/articles/<int:n>'
    b. HTTP Method: 'GET'
    c. Functionality: Once a connection to the database is made, the method retrieves the contents (title and body) of n most recent article from the database and returns it with an HTTP 200 code.
6. Retrieving metadata of n most recent articles
    a. Route: '/articles/info/<int:n>'
    b. HTTP Method: 'GET'
    c. Functionality: Once a connection to the database is made, the method retrieves the meta data (username, title, body, created date, and url) of n most recent article from the database and returns it with an HTTP 200 code.

Users

The Users microservice allows the front end to access the user database by referencing certain routes. Using HTTP methods, Users has various methods that allow creation of new user accounts, deletion of accounts, and updating account password. The operations included in this microservice:

1. Creating a new user
    a. Route: '/user/new'
    b. HTTP Method: 'POST'
    c. Functionality: this python method first connects to the database and allocates a cursor to the connection. It then passes the username, and password with a request.form call. A bcrypt function is called to hash the password which is then inserted with the username into the database. An HTTP 201 code is returned upon success.
2. Delete a user account
    a. Route: '/user'
    b. HTTP Method: 'DELETE'

c. Functionality: This method connects to the database, checks if request.authorization exists, and pulls the username and password from the http header. Upon succession, checkAuth is called to ensure the username/password pair exists in the database. Returns an error code, 409 if it is not found in the database. If the pair exists, the account is deleted from the User table in the schema.

3. Edit account password
    a. Route: '/user/edit'
    d. HTTP Method: 'PATCH'
    e. Functionality: This method connects to the database, checks if request.authorization exists, and pulls the username and password from the http header. Upon succession, checkAuth is called to ensure the username/password pair exists in the database. Returns an error code, 409 if it is not found in the database. If the pair exists, the new password is hashed and updated into the database.

Tag

The Tag microservice acts as REST service allowing another application or user to manipulate tags by referencing certain routes. Using HTTP methods, the service supports various creation, edits, deletion, and retrieval of tags or data from tags. The operations included in this microservice:

7. Posting an article and a tag with it
    a. Route: '/article/tag/<string:tag>'
    b. HTTP Method: 'POST'
    c. Required Form Attributes to include in the request:
        i. title: the title of the article to post
        ii. body: the body of the article to post
    d. Required Basic Auth Username and Password in Authorization header
    e. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. It then passes the authorization username and password, title, article body and tag to an internal variable. checkAuth(username, password) is called to authenticate the user credentials before posting the article. If the credentials passed in are incorrect, a 409 error is thrown, if the credentials are correct, the new article is inserted into the database and the new tag is inserted referencing the article. The articleId and the tag are returned with a 201 code.

8. Add another tag to an existing article
    a. Route: '/article/<int:articleId>/tag/<string:tag>'
    b. HTTP Method: 'PUT'

    c. Required Basic Auth Username and Password in Authorization header
    d. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. It then passes the authorization username and password to an internal variable, if these are not provided the 401 error is returned. checkAuth(username, password) is called to authenticate the user credentials before posting the article. If the credentials passed in are incorrect, a 409 error is thrown, if the credentials are correct we verify that the articleId passed in the url exists. If it does not exist then we return the 404 error code and if it does we verify that the user is the author of the article and then insert the tag into the database referencing the article. We return the 200 success code and the json articleId back.
9. Delete one or more of the tags from the article
    a. Route: '/article/<int:articleId>/tag'
    b. HTTP Method: 'DELETE'
    c. Required Form Attributes to include in the request:
        i. tags: All of the tags you would like to add separated by commas
            1. Example Header: tag: me,socool,lit,superfly
    d. Required Basic Auth Username and Password in Authorization header
    e. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. It then passes the authorization username and password and the tag to an internal variable, if these are not provided the 401 error is returned. checkAuth(username, password) is called to authenticate the user credentials before deleting the tag. If the credentials passed in are incorrect, a 409 error is thrown, if the credentials are correct we verify that the articleId passed in the url exists. If it does not exist then we return the 404 error code and if it does we verify that the user is the author of the article and then delete the tags from the database referencing the article. We return the 200 success code and the json response back of the tag and True or False depending on if it was successfully deleted or not.

10. List all articles with a specific tag
    a. Route: '/articles/tag/<string:tag>'
    b. HTTP Method: 'GET'
    c. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. We verify that the tag passed in the url exists in at least one article. If it does not exist then we return the 404 error code and if it does we grab all of the article Id's from the tag table with the given tag. We return the 200 success code and the json response back of the article Id's. This artId then can be used to grab an article from the article microservice.

11. Retrieve the tags for an individual article
    a. Route: '/article/<int:articleId>/tags'
    b. HTTP Method: 'GET'
    c. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. We verify that the articleId passed in the url exists. If it does not exist then we return the 404 error code and if it does we grab all of the tags from the tag table with the given articleId. We return the 200 success code and the json response back of the articles tags.

Comments

The Comments microservice allows the front-end to manipulate comments by referencing certain microservice routes. Using HTTP methods, the service supports various creation, edits, deletion, and retrieval of comments or data from comments. The operations included in this microservice:

12. Post a new comment on an article
    a. Route: '/article/<int:articleId>/comment'
    b. HTTP Method: 'POST'
    c. Required Form Attributes to include in the request:
        i. comment: the string body of the comment
    d. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. It then passes the authorization username and password and the comment to an internal variable, if these are not provided then you are considered an anonymous coward. We also verify that the authorization is valid using checkAuth() and if the credentials are not correct we return the 409 error code. We verify that the articleId passed in the url exists. If it does not exist then we return the 404 error code and if it does we insert the comment into the comment table and grab its commentId. We return the 200 success code and the json response back of the articleId and the commentId.

13. Delete an individual comment
    a. Route: '/article/comment/<int:commentId>'
    b. HTTP Method: 'DELETE'
    c. Required Basic Auth Username and Password in Authorization header
    d. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. It then passes the authorization username and password to an internal variable, if these are not provided then you can not delete and a 401 error code is returned. We also verify that the authorization is valid using

checkAuth() and if the credentials are not correct we return the 409 error code. We verify that the commentId passed in the url exists. If it does not exist then we return the 404 error code and if it does we grab the author from the comment and verify that it matches your username. If it does not then we return the 409 error code. We then delete the comment from the comment table and return the 200 success code.

14. Retrieve the number of comments on a given article
    a. Route: '/article/<string:articleId>/comments'
    b. HTTP Method: 'GET'
    c. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. We verify that the articleId passed in the url exists. If it does not exist then we return the 404 error code and if it does we grab all of the comments from the comment table with the given articleId. We return the 200 success code and the json response back of the number of comments returned.

15. Retrieve the n most recent comments on a URL
    a. Route: '/article/<string:articleId>/comments/<int:n>'
    b. HTTP Method: 'GET'
    c. Functionality: This api endpoint first connects to the sqlite database and allocates a cursor to the connection. We verify that the articleId passed in the url exists. If it does not exist then we return the 404 error code and if it does we grab all of the comments up to n from the comment table with the given articleId. We return the 200 success code and the json response back of the articles comments formatted.

Database

The database was written using SQLite. The schema consists of tables: : User, Article, Tag, and Comment. The tables are connected to and by all microservices for multiple purposes including storing, retrieving, or authenticating data

1. User Table stores:
    a. UserId integer primary key is auto-incremented
    b. Username string is unique
    c. Password string is stored as a hashed string using bcrypt
2. Article Table
    a. artId integer primary key is auto-incremented
    b. userName is the foreign key that references the User table
    c. Created timestamp
    d. Modified timestamp

    e. Title string
    f. Body string
3. Tag Table
    a. tagId integer primary key is auto-incremented
    b. Tag string
    c. artId integer is a foreign key that references Article table
    d. Created timestamp
4. Comment Table
    a. commentId integer primary key is auto-incremented
    b. Comment string
    c. artId integer is a foreign key that references the Article Table
    d. Author String default to ‘anonymous coward’ if not given
    e. Created is a timestamp

Flask CLI

The Flask CLI allows you to add custom commands using the library click. I added the command init-db-command which can be run by typing ‘flask init-db-command’ in the terminal with either the tag.py or comments.py files set as FLASK_APP. This custom command drops the database and re-initializes it with three rows in each table. This command was immensely helpful during development and I can fully understand the need for this knowledge after learning how helpful it can be.