Project 2

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. Articles microservice uses the articleDB database which is referenced by the creation of an engine, which points to that specific database through the bind key. The operations included in this microservice:

1. Posting a single article
    a. Route: '/article/new/<title>/<body>'
    b. HTTP Method: 'POST'
    c. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the Article database by using the bind key for this database. Then the method 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 api endpoint first connects to the sqlite database and calls an engine to connect to the Article database by using the bind key for this database. Then the method 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: This api endpoint first connects to the sqlite database and calls an engine to connect to the Article database by using the bind key for this database. Then the 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.
4. Deleting an existing article
    a. Route: ‘/article/<int:articleId>’
    b. HTTP Method: ‘DELETE’
    c. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the Article database by using the bind key for this database. Then 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: This api endpoint first connects to the sqlite database and calls an engine to connect to the Article database by using the bind key for this database. Then 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: This api endpoint first connects to the sqlite database and calls an engine to connect to the Article database by using the bind key for this database. Then 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. User microservice uses the usersDB database which is referenced by the creation of an engine, which points to that specific database through the bind key. The operations included in this microservice:

1. Creating a new user
    a. Route: ‘/user/new’

    b. HTTP Method: ‘POST’
    c. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the User database by using the bind key for this database. 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 api endpoint first connects to the sqlite database and calls an engine to connect to the User database by using the bind key for this database. If the pair exists, the account is deleted from the User table in the schema.
3. Edit account password
    a. Route: ‘/user/edit’
    b. HTTP Method: ‘PATCH’
    c. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the User database by using the bind key for this database. A session is then allocated to connect to the database. If the pair exists, the new password is hashed and updated into the database.
4. Authenticating User
    a. Route: ‘/user/auth’
    b. HTTP Method: ‘GET’
    c. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the User database by using the bind key for this database. The method then checks the credentials by running the checkAuth(username, password) function which uses the bcrypt library to check if the credentials match the credentials stored in the database without reading the given user credentials. But instead using Bcrypt function calls.

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. Tag microservice uses the tagDB database which is referenced by the creation of an engine, which points to that specific database through the bind key. 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. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the Tag database by using the bind key for this database. A session is then allocated to connect to the database. 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. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the Tag database by using the bind key for this database. A session is then allocated to connect to the database. 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. Functionality: This api endpoint first connects to the sqlite database and calls an engine to connect to the Tag database by using the bind key for this database. A session is then allocated to connect to the database. 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 calls an engine to connect to the Tag database by using the bind key for this database. A session is then allocated to connect to the database. 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 calls an engine to connect to the Tag database by using the bind key for this database. A session is then allocated to connect to the database. 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. Comments microservice uses the commentDB database which is referenced by the creation of an engine, which points to that specific database through the bind key. 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 calls an engine to connect to the Comment database by using the bind key for this database. A session is then allocated to connect to the database. It then passes the username and password and the comment to an internal variable, if these are not provided then you are considered an anonymous coward. 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 calls an engine to connect to the Comment database by using the bind key for this database. A session is then allocated to connect to the database. 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 calls an engine to connect to the Comment database by using the bind key for this database. A session is then allocated to connect to the database. 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 calls an engine to connect to the Comment database by using the bind key for this database. A session is then allocated to connect to the database. 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.

RSS Feed

The syndication microservices provides an RSS feed which list the ten most recent articles. Using HTTP Requests, the service is able to access one or more microservices.

1. Retrieve the 10 most recent articles
    a. Route: 'localhost:5400/articles/rss'
    b. HTTP Method: 'GET'
    c. Functionality: This api endpoint first makes a requests to the articles microservice. Once it has retrieve the articles, it will format each article into an rss item. Then, it will take all the items and turn into an RSS feed and return a list of items.

Database

The database was written using SQLAlchemy. The database configuration file creates 4 separate databases : User, Article, Tag, and Comment. The databases are created when the microservices init the DB. Each microservice only has access to the database that belongs to them, and the database is only allocated when used.

1. User Table stores:
    a. Default bind
    b. UserId integer primary key is auto-incremented
    c. Username string is unique
    d. Password string is stored as a hashed string using bcrypt
2. Article Table
    a. __bind_key__ "articles"
    b. username string - author of article
    c. artId integer primary key is auto-incremented
    d. created timestamp
    e. modified timestamp
    f. title string
    g. body string
3. Tag Table
    a. __bind_key__ "tags"
    b. tagId integer primary key is auto-incremented
    c. username string - author of article
    d. title string (of article)
    e. body string (of article)
    f. tag string
    g. artId integer
    h. created timestamp

4. Comment Table
    a. __bind_key__ “comments”
    b. commentId integer primary key is auto-incremented
    c. comment string
    d. artId integer is a foreign key that references the Article Table
    e. username string - author of article
    f. 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.

Nginx

Nginx acts as a reverse proxy load balancer that allows us to set up the “back-end for front-end” pattern. We set up this load balancer to hit /auth which proxies to the /user/auth route on every request. We first set up the 3 instances of each of the original four microservices by running the command “foreman start --formation "comments=3, articles=3, user=3, tag=3”. Then we added upstream servers to the ports of each of these apps and added them to the server blocks proxy_pass route. The way we set this up adds every microservice as a unique prefixed route to the api route on localhost of Nginx. An example would be that before to get the number of articles with a tag you would send an HTTP GET request at “http://localhost:5000/articles/tag/me” and now you would send an HTTP GET request at “http://localhost/tags/articles/tag/me”. Articles, comments and tags are the three prefixes for the microservices. The users microservice <u>does not need a prefix</u> because all of the routes start with user/ anyways. An example route for users from before would be an HTTP POST request at “http://localhost:5000/user/new” and now would be an HTTP POST request at “http://localhost/user/new”, so there is no prefix added.