Team2000s Second Iteration

Brett Jia (bbj2110)
Soorya Kumar (ssk2234)
Zixiong Liu (zl2683)
Mavis Athene U Chen (mu2288)

Part 1

Blog Post Creation: As a blog writer, I want to be able to create new blog posts in Markdown, so that I can create expressive and comprehensive posts that can better convey the educational content I want to convey from my blog.

My conditions of satisfaction are:

- When I am logged into my account I should easily be able to start creating a new blog post using a "Create" button or something of that sort.
- I should be able to create my blog post in Markdown that allows me to embed photos, include code snippets, and allow for other types of syntax formatting and highlighting common in other applications.
- The Markdown editor should be able to render a preview to show users what the post will look like.
- Once I have completed a blog post to my satisfaction, I should be able to post the blog post so that it is available to all of my followers to read.

Blog Post Editing: As a blog writer, I want to be able to edit blog posts that I have already posted to my followers in Markdown if needed so that I have the flexibility to make adjustments to my initial posts and am not locked in when I have made a mistake on initial posting.

My conditions of satisfaction are:

- I should be able to see a dashboard of all the posts I have ever made.
- I should be able to select a post on the dashboard that I want to edit.
- I should be able to use the same Markdown editor I used to create the Blog post to edit it.
- Once I press a button like "Post" or "Update", I should be able to publish my updated post in place of my original post. Any follower that then accesses that blog post should only see the updated post

Blog Post Deletion: As a blog writer, I want to be able to delete blog posts that I no longer want my followers to be able to see, so that I can remove posts that might be outdated or misleading or otherwise no longer relevant.

My conditions of satisfaction are:

- I should be able to see a dashboard of all the posts I have ever made.
- I should be able to select a post on the dashboard that I want to Delete.
- Once I click on a “Delete” button, that should delete my post from my dashboard, and the application as a whole
- My followers should not be able access my article anymore if I have deleted it. If they have a link to the article that I have deleted and try to use it (perhaps from a notification), then they should be routed to a page that tells them that the article is no longer available.

Blog Post Author Page: As a blog writer, I want to have an author page where my followers can read more about me and see the list of all the articles I have ever written so that potential followers can easily decide if my content is for them and whether or not they would like to follow along.

My conditions of satisfaction are:

- I should be able to see a dashboard of all the posts I have ever made. On this page I should have the option to edit my name as it appears to my followers.
- On my dashboard I should also have the option to set a About me field, so that I can describe myself and the type of content that I like to write for my followers and potential followers to see.
- When a follower or potential follower searches for me in the applications, or reads one of my blog posts and clicks on my name on that blog post, they should be directed to my author page
- The follower or potential follower should be able to read my “About me” and should be given the option to follow/subscribe to me if they are interested with the click of a “Follow” button.

Blog Subscription: As a blog reader, I want to be notified immediately when my favorite bloggers post a new blog entry so that I can be up to date with what the blog writers have to share.

My conditions of satisfaction are:

- Every time my favorite/followed bloggers submit a blog post, I want to receive a notification.
- For bloggers that I am not interested in, I do not want to receive a notification about their posts.
- I should be able to receive notifications within a minute of the blog being submitted.
- I should receive at most one notification per blog post and not multiple notifications for the same post.
- I should be able to unfollow blogs that I am no longer interested in reading.

Part 2

Our tests are located: https://github.com/bjia56/coms4156/tree/main/tests

Equivalence Classes for Back End Methods

apiFollowGET

The only input this function accepts is the implicit session cookie for the logged in user. It returns the followers and followees of the logged in user.

- Valid Equivalence Class - Session cookie includes a valid User uuid (this proves authentication)
    - Test case in apiFollowGet.test.js: `get all follow records with with multiple follow records returns a list of ids`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid (the user is not authenticated)
    - Test case in apiFollowGet.test.js: `get follow records when not logged in returns error`

apiFollowDELETE

The only input this function accepts (aside from an implicit session cookie for the logged in user) is the uuid of the user who is being followed in the existing follow relationship that is to be deleted.

- Valid Equivalence Class - Combination of Input User uuid corresponds and session cookie user uuid corresponds to a valid Follow record in database
    - Test case in apiFollowDelete.test.js: `delete follow record with valid uuid`
- Invalid Equivalence Class - Combination of Input User uuid corresponds and session cookie user uuid does not correspond to a valid Follow record in database
    - Test case in apiFollowDelete.test.js: `delete follow record when there are no follow records returns reject error`
    -

Even though the session cookie is not passed by client code calling this API method, the session cookie still represents input into the API method and creates equivalence classes

- Valid Equivalence Class - Session cookie includes a valid User uuid (this proves authentication)
    - Test case in apiFollowDelete.test.js: `delete follow record with valid uuid`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid (the user is not authenticated)

    - Test case in apiFollowDelete.test.js: `delete follow record when not logged in returns error`

apiFollowPOST

The only input this function accepts (aside from an implicit session cookie for the logged in user) is the uuid of the user who is being followed in the new follow relationship.

- Valid Equivalence Class - Input User uuid corresponds to a valid User uuid in database
    - Test case in apiFollowPost.test.js: `create follow record with valid uuids`
- Invalid Equivalence Class - Input User uuid does not correspond to a valid User uuid in database
    - Test case in apiFollowPost.test.js: `create follow record when user not found returns error`
- Invalid Equivalence Class - Input User uuid corresponds to the same User uuid in as the session cookie of the logged in users (this is invalid because you cannot follow yourself)
    - Test case in apiFollowPost.test.js: `create follow record cannot follow self returns error`

Even though the session cookie is not passed by client code calling this API method, the session cookie still represents input into the API method and creates equivalence classes

- Valid Equivalence Class - Session cookie includes a valid User uuid (this proves authentication)
    - Test case in apiFollowPost.test.js: `create follow record with valid uuids`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid (the user is not authenticated)
    - Test case in apiFollowPost.test.js: `create follow record when not logged in returns error`

apiUserGET

The only input this function accepts (aside from an implicit session cookie for the logged in user) is the uuid of the user whose information should be retrieved

- Valid Equivalence Class - Input User uuid corresponds to a valid User uuid in database
    - Test case in apiUserGet.test.js: `valid user returns correct user object`
    - Test case in apiUserGet.test.js: `valid user description given returns correct user object`
    - Test case in apiUserGet.test.js: `phone returned when querying for self`

- Invalid Equivalence Class - Input User uuid does not correspond to a valid User uuid in database
    - Test case in apiUserGet.test.js: `invalid user returns reject error`
    - Test case in apiUserGet.test.js: `empty database returns reject error`
- Invalid Equivalence Class - Parameter is omitted
    - Test case in apiUserGet.test.js: `omitted parameter returns logged in user information`

Even though the session cookie is not passed by client code calling this API method, the session cookie still represents input into the API method and creates equivalence classes

- Valid Equivalence Class - Session cookie includes a valid User uuid (this proves authentication)
    - Test case in apiUserGet.test.js: `valid user returns correct user object`
    - Test case in apiUserGet.test.js: `valid user description given returns correct user object`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid (the user is not authenticated)
    - Test case in apiUserGet.test.js: `omitted parameter when not logged in returns error`

<u>apiUserPUT</u>

The only input this function accepts (aside from an implicit session cookie for the logged in user) is the body of the message that contains updated user information.

- Valid Equivalence Class - Session cookie includes a valid User uuid and body is a JSON-formatted message containing the optional arguments “name”, “description”, “notificationPreference”, and “phone”
    - Test case in apiUserPut.test.js: `update user name`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid
    - Test case in apiUserPut.test.js: `user not logged in should return error`

<u>apiBlogGET</u>

The only input this function accepts are the optional arguments “author”, representing the blog author’s uuid, and “limit”, the number of records to return.

- Valid Equivalence Class - The “author” optional argument is not provided, and “limit” is any integer between 1 and 100 inclusive, or unspecified (defaults to 10)
    - Test case in apiBlogGet.test.js: `get all blogs given a author uuid returns a list of ids`

- Invalid Equivalence Class - The “limit” value is not within the range of 1 to 100 inclusive
    - Test case in invalidInput.test.js: `invalid GET /api/blogs limit 101`
    - Test case in invalidInput.test.js: `invalid GET /api/blogs limit 0`

apiBlogPOST

The only input this function accepts is an implicit session cookie for the logged in user.

- Valid Equivalence Class - Session cookie includes a valid User uuid
    - Test case in apiBlogPost.test.js: `create blog returns JSON with an integer uuid`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid
    - Test case in apiBlogPost.test.js: `post blogs rejects if unauthenticated`

apiBlogUuidDELETE

The only input this function accepts is an implicit session cookie for the logged in user and a query parameter of the blog uuid to delete.

- Valid Equivalence Class - Session cookie includes a valid User uuid and the Blog uuid is an existing blog authored by the user
    - Test case in apiBlogDelete.test.js: `delete blog with valid uuid`
    - Test case in apiBlogDelete.test.js: `delete 2 blog with valid uuid`
- Invalid Equivalence Class - Session cookie does not include a valid User uuid, or the Blog uuid does not exist, or the blog is authored by another user
    - Test case in apiBlogDelete.test.js: `delete blog when not logged in returns error`
    - Test case in apiBlogDelete.test.js: `delete another user’s blog returns error`
    - Test case in apiBlogDelete.test.js: `delete blog when there are no blogs returns reject error`
    - Test case in apiBlogDelete.test.js: `delete blog with invalid uuid returns reject error`

apiBlogUuidGET

The only input this function accepts is a query parameter of the blog uuid to fetch.

- Valid Equivalence Class - The Blog uuid is an existing blog
    - Test case in apiBlogGet.test.js: `valid blog uuid returns correct object`
    - Test case in apiBlogGet.test.js: `valid blog uuid with no content returns correct object`
- Invalid Equivalence Class - The Blog uuid is not a valid blog

    - Test case in apiBlogGet.test.js: `invalid blog uuid returns reject error`
    - Test case in apiBlogGet.test.js: `empty database returns reject error`

apiBlogUuidPUT

The only input this function accepts is a query parameter of the blog uuid to fetch, request body that is JSON-formatted and includes optional properties “title” and “contents” for the title and contents respectively, and the implicit session cookie for the logged-in user.

- Valid Equivalence Class - The Blog uuid is an existing blog and the session cookie includes a valid User uuid
    - Test case in apiBlogPut.test.js: `valid blog uuid and empty body returns correct object`
    - Test case in apiBlogPut.test.js: `valid blog uuid and empty title returns correct object`
    - Test case in apiBlogPut.test.js: `valid blog uuid and empty contents returns correct object`
    - Test case in apiBlogPut.test.js: `valid blog uuid and non empty body returns correct object`
- Invalid Equivalence Class - The Blog uuid is not a valid blog, or the session cookie contains an invalid user, or the Blog uuid is not owned by the logged-in user
    - Test case in apiBlogPut.test.js: `invalid blog uuid returns reject error`
    - Test case in apiBlogPut.test.js: `put blog without logging in returns reject error`
    - Test case in apiBlogPut.test.js: `put another user's blog returns reject error`

The only major function in our backend that contains any boundary conditions is the GET /api/blogs endpoint when provided with the “limit” parameter, as its range is [1, 100]. We tested the boundary outside to ensure that values outside this valid range are being rejected. All other endpoints have equivalence classes where the cases are binary (e.g. a uuid exists in the database or not, or a parameter is provided or not).

Part 3:

We were able to reach just over 70% branch coverage.

Coverage can be found in this GitHub Actions run, at the very end of stage “Test”: https://github.com/bjia56/coms4156/runs/1521570068?check_suite_focus=true

Part 4:

GitHub Actions: https://github.com/bjia56/coms4156/tree/main/.github/workflows
GitHub Actions workflow reports: https://github.com/bjia56/coms4156/actions