Team2000s First Iteration

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

Part 1:

Our GitHub repository can be found here: https://github.com/bjia56/coms4156

Part 2:

User Stories

Below are our user stories as they were from our Revised Project Proposal. Please note that we will only be implementing and unit testing 3 of our 5 user stories in this iteration, and will add the other two in the second iteration. The user stories we will be implementing in this iteration are **Blog Creation**, **Blog Editing**, and **Blog Deletion**.

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to delete blog posts that I no longer want my followers to be able to see, so that I can remove posts that might be outdated or misleading or otherwise no longer relevant.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.
- If I want immediate updates, 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 modify my notification preferences to determine the frequency of notifications.
- I should be able to unfollow blogs that I am no longer interested in reading.

* *Not implemented in the first iteration.*

<u>Acceptance testing plan</u>

In this first iteration our front end is not completely finished so we will not be able to use it as part of our acceptance testing. Instead our acceptance testing in this iteration involves using our API endpoints to test our user stories in a holistic fashion. Authentication is excluded for this first iteration in favor of focusing on functionality. Our testing will be carried out in Postman.

Here is a high level outline of our API, this overview is necessary to explain our acceptance testing:

/api/blogs: (will be implemented in first iteration)

```
    GET:
      summary: Get a list of blogs
      parameters:
        - name: author
          description: Fetch blogs by author uuid.
        - name: cursor
          description: Cursor token for paging the blogs returned.
        - name: limit
          description: Maximum number of results to return in a page.
     responses:
        '200':
            description: Success. Returns a list of blog ids.
    POST:
      summary: Create a new blog
      parameters: none
      responses:
        '201':
          description:
            A new blog was created successfully. Returns the blog
            uuid of the new blog.
        '401':
          description: Unauthenticated. The user is not logged in.
      requestBody: None
```

/api/blogs/{id}: (will be implemented in first iteration)

```
    GET:
      summary: Get blog details
      parameters:
        - name: id
          description: Blog id.
      responses:
          description: Success. Returns a blog object.

    PUT:
      summary: Update an existing blog
      description:
      parameters:
        - name: uuid
          description: Blog id.
      responses:
        '200':
          description: Success.
        '401':
          description: Unauthenticated. The user is not logged in.
        '403':
          description:
            Unauthorized. The user does not have permissions to
            modify the given blog.
        '404':
          description: Blog not found.
      requestBody:
            properties:
              title:
                description: Blog title
              contents:
                description: Blog contents in raw Markdown
    DELETE:
      summary: Delete an existing blog
      parameters:
        - name: id
          description: Blog uuid.
      responses:
        '200':
          description: Success.
        '401':
          description: Unauthenticated. The user is not logged in.
        '403':
          description:
            Unauthorized. The user does not have permissions to
            modify the given blog.
        '404':
          description: Blog not found.
```

/api/user: (will NOT be implemented in first iteration)

```
    GET:
      summary: Get user information
      parameters:
        - name: user
          description: User uuid.
      responses:
        '200':
          description: Success.

    PUT:
      summary: Update user information
      responses:
        '200':
          description: Success.
        '401':
          description: Unauthenticated. The user is not logged in.
      requestBody:
            properties:
              name:
                description: User name
              description:
                description: User description ("About Me")
              notificationPreference:
                description: Notification preference (TBD)
```

/api/follow: (will NOT be implemented in first iteration)

```
    POST:
      summary: Add a follow entry
      description: This can only be done by users who have logged in.
      parameters:
        - name: user
          description: User uuid.
      responses:
        '200':
          description: Success.
        '401':
          description: Unauthenticated. The user is not logged in.
      requestBody: None
    DELETE:
      summary: Delete a follow entry
      parameters:
        - name: user
          description: User uuid.
      responses:
        '200':
          description: Success.
        '401':
          description: Unauthenticated. The user is not logged in.
        '404':
          description:
            The current logged-in user is not following the target
            User.
```

/api/search: (will NOT be implemented in first iteration)

```
    GET:
      summary: Search for blogs
      parameters:
        - name: keyword
          description: Keyword(s) to search for
        - name: cursor
          description: Cursor token for paging the blogs returned.
        - name: limit
          description: Maximum number of results to return in a page.
      responses:
        '200':
          description: Success
```

For more detailed information about our API, please see:
https://bjia56.github.io/coms4156/openapi.html

Blog Post Creation User Story Acceptance Testing

Many of the conditions for satisfaction for this user story are related to actions that would occur on the front end. We will instead focus our testing on the responsibility of the backend, actually creating the blog, and the relevant endpoints to do that. Our actual results at each step are recorded in Italics.

- Valid Test Case - Create Initial Blog
    - Start with no blog records in the database
        - *We validate this by sending a GET to /api/blogs to ensure that no uuids are returned.*
    - Send a POST /api/blogs request (which takes no request parameters or request body)
        - *We receive back JSON with the field "uuid" set to 1, the first blog ID.*
    - Send a GET /api/blogs request
        - *We receive back JSON with the field "uuids" set to an array of [1].*
    - There should be 1 blog record returned. Take a note of the ID of the blog for the next step
        - *Our results match what was expected.*
- Valid Test Case - Create a Second blog to test Incrementation of Blog ID
    - Start after the Create Initial Blog Test Case
    - Send a POST /api/blogs request (which takes no request parameters or request body)
        - *We receive back JSON with the field "uuid" set to 2.*
    - Send a GET /api/blogs/{id} request with id = 1 + the ID of the blog created from the *Create Initial Blog* Test Case
        - *We receive back JSON representing the blog, including title, contents, rendered-contents, created/updated times, and basic author information.*
    - The result should be a blog record, that was created in this test, with the requested ID
        - *Our results match what was expected.*
- Invalid Test Case - Request Body Provided

    - As POST /api/blogs request does not take a request body the only invalid case is if a request body is provided, to test to see how the endpoint handles it.
    - Start after the Create a Second blog to test Incrementation of Blog ID Test Case
    - Send a POST /api/blogs request with a request body
        - *We send some text in the request body: "aaaaaaaaaaaaaaaaaaaa". The response is JSON with the field "uuid" set to 3.*
    - Send a GET /api/blogs request
        - *We receive back JSON with the field "uuids" set to an array of [3, 2, 1]. This is sorted by descending last-modified time.*
    - There should be a third blog record returned. This is because the API should just ignore any request body parameter.
        - *Our results match what was expected.*

<u>Blog Post Editing User Story Acceptance Testing</u>

Many of the conditions for satisfaction for this user story are related to actions that would occur on the front end. We will instead focus our testing on the responsibility of the backend, actually editing the blog, and the relevant endpoints to do that. Our actual results at each step are recorded in Italics.

- Valid Test Case - Edit a Blog
    - Start after the Request Body Provided Test Case
    - Send a GET /api/blogs request
        - *We receive back JSON with the field "uuids" set to an array of [3, 2, 1]. This is sorted by descending last-modified time.*
    - Choose one IDs of one of the returned blogs to edit
        - *For this test case, we will edit blog ID "2".*
    - Send a PUT /api/blogs/{id} request with the *title* and *contents* request body parameters filled in with any text.
        - *We send the request with body containing JSON with the field "title" set to "My Blog" and "contents" set to "# Some text". We receive back HTTP 200 with no response body.*
    - Send a GET /api/blogs/{id} request with the same ID used in the PUT message of the previous step
        - *We receive the blog contents object. The title is set to "My Blog", contents set to "# Some text", and the rendered blog set to "<h1>Some text</h1>\n".*
    - The *title* and *contents* properties of the returned blog object should match that of of the *title* and *contents* provided in the request body of the PUT /api/blogs/{id}
        - *Our results match what was expected.*
- Invalid Test Case - Attempt to Edit Blog that does not exist

    - Start after the Edit a Blog Test Case
    - Send a GET /api/blogs request
        - *We receive back JSON with the field "uuids" set to an array of [2, 3, 1]. This is sorted by descending last-modified time.*
    - Choose/create an ID that DOES NOT belong to any of the returned blogs, this will simulate trying to edit a blog that does not exist
        - *We use the blog ID "4".*
    - Send a PUT /api/blogs/{id} request with the *title* and *contents* request body parameters filled in with any text and the ID chosen that does not blog to any existing blog objects.
        - *We received back HTTP 404 with the contents "Blog not found".*
    - The PUT message should return an error "404 Blog Not Found"
        - *Our results match what was expected.*
    - Send a GET /api/blogs request, and verify that no new blog objects were created
        - *We receive back JSON with the field "uuids" set to an array of [2, 3, 1]. No new blogs were created.*

<u>Blog Post Deletion User Story Acceptance Testing</u>

Many of the conditions for satisfaction for this user story are related to actions that would occur on the front end. We will instead focus our testing on the responsibility of the backend, actually deleting the blog, and the relevant endpoints to do that. Our actual results at each step are recorded in Italics.

- Valid Test Case - Delete a Blog
    - Start after the Attempt to Edit Blog Test Case
    - Send a GET /api/blogs request
        - *We receive back JSON with the field "uuids" set to an array of [2, 3, 1]. This is sorted by descending last-modified time.*
    - Choose one IDs of one of the returned blogs to edit, and note how many blog objects are returned
        - *We use the blog ID "1".*
    - Send a DELETE /api/blogs/{id} request with the selected ID
        - *We receive back HTTP 200 with no response body.*
    - Send a GET /api/blogs request
        - *We receive back JSON with the field "uuids" set to an array of [2, 3]. This is sorted by descending last-modified time.*
    - In the result set from the second GET request there should be one fewer blog object than compared to the first GET request in this test
        - *Our results match what was expected.*
- Invalid Test Case - Attempt to Delete Blog that does not exist

- Start after the Delete a Blog Test Case
- Send a GET /api/blogs request
  - *We receive back JSON with the field "uuids" set to an array of [2, 3]. This is sorted by descending last-modified time.*
- Choose/create an ID that DOES NOT belong to any of the returned blogs, this will simulate trying to delete a blog that does not exist. Also note the total number of blog objects in the result set
  - *We use the blog ID "1" as it was already deleted.*
- Send a DELETE /api/blogs/{id} request and the ID chosen that does not blog to any existing blog objects.
  - *We receive back HTTP 404 with the response body "Blog not found".*
- The DELETE message should return an error "404 Blog Not Found"
  - *Our results match what was expected.*
- Send a GET /api/blogs request, and verify that no new blog objects were deleted, or in other words that there are the same number of blog objects as there were in the response of the first GET request in this test.
  - *We receive back JSON with the field "uuids" set to an array of [2, 3]. No blogs were deleted.*

We see from our testing and results that our APIs behave as expected. Bugs were not found during the acceptance testing phase as we were careful during our development process to test these endpoints to our agreed-upon API specifications.

We are not doing acceptance testing for the Blog Subscription or Blog Author Page user stories in this iteration because we have not implemented those User Stories yet.

Part 3:

Please see the following for links to our tests and tools:

- Link to test cases: https://github.com/bjia56/coms4156/tree/575021f04df54fef831f158373842a64caf89175/utils/tests/unit
- Link to build tool: https://github.com/bjia56/coms4156/blob/575021f04df54fef831f158373842a64caf89175/package.json#L8
- Link to automated testing tool: https://github.com/bjia56/coms4156/blob/575021f04df54fef831f158373842a64caf89175/package.json#L9
- Link to GitHub Action invoking testing tool: https://github.com/bjia56/coms4156/runs/1428275791?check_suite_focus=true

Part 4:

Please see the following for links to our style checker and bug finder:

- Link to style checker report:
    - Bad style output:
      https://github.com/bjia56/coms4156/blob/5e874620bfe38a0acf2e68a4c6380c20359f08ac/reports/bad_style.txt
    - Bad style in GitHub Actions:
      https://github.com/bjia56/coms4156/runs/1428248036?check_suite_focus=true
    - Good style output:
      https://github.com/bjia56/coms4156/blob/5e874620bfe38a0acf2e68a4c6380c20359f08ac/reports/good_style.txt
    - Good style in GitHub Actions:
      https://github.com/bjia56/coms4156/pull/23/checks?check_run_id=1428275830
- Link to bug finder report:
    - Buggy:
      https://lgtm.com/projects/g/bjia56/coms4156/rev/850632504e49391993e651b9e13a15ff5ce1de3e
    - Fixed:
      https://lgtm.com/projects/g/bjia56/coms4156/rev/0b7a906e5bec6b6c8bddb4cd4f87bf5890e322fd