TEST PLAN FOR

CASTR

Note that you can refine your testing plan as the project development goes. Keep the change log as follow:

ChangeLog

Version	Change Date	By	Description
1.0.0	Feb. 11, 2023	All Members	Initial Test Plan
1.0.1	Mar. 12, 2023	Dean Cruz	Add Dean's tests
1.0.2	Mar. 12, 2023	Da Tan	Add Da's tests

1 Introduction

1.1 Scope

This is our testing scope for Sprint 2:

1. User Account Management
    - Get JWT by login
    - Create user account (sign up)
    - Delete user account
    - Update username and password
    - Update user basic information
    - Update account preferences (i.e., notifications)

2. Create and Interact with Posts
    - Create, delete, and update a post
    - Create, delete, and update a comment
    - Like and unlike a post
    - Retrieve posts and comments
    - Share a post via a QR code
3. Subsciptions
    - Subscribe and unsubscribe to a user
    - Receive an email containing the posts of the users you subscribed to
    - Get the list of users you subscribed to
    - Get the list of users who subscribed to you

1.2 Roles and Responsibilities

Name	Net ID	GitHub username	Role
Weiyu Sun	sunw1	WeiyuSun	Back-end Developer, Back-end QA Analyst, Tester
Shahzaib Paracha	parachsa	ShahzaibParacha	Front-end Developer,
Da Tan	tand2	DaTanUmanitoba	Back-end Developer, Tester
Theo Gerwing	gerwing1	theogschool	Front-end Developer
Dean Cruz	cruzd	orangeboy55555	Back-end Developer, Front-end Developer, Tester

Role Details:

1. Front-end Developer
    - A developer working on the front-end (e.g., UI) of the application.
2. Back-end Developer
    - A developer working on the back-end (e.g., Logic and Database) of the application.
3. Tester
    - Responsible for writing tests for the application. The nature of the tests range from unit tests to acceptance and even load tests.
4. Back-end QA Analyst
    - Tasked with testing the quality of the back-end and back-end code before it gets merged to the develop branch.

2 Test Methodology

2.1 Test Levels

Core Feature: User Account Management

Unit Tests for Back-end:

1. User schema follows the requirements
    a. Schema requires **unique** username
        i. The length of username should >= 3 and <= 30
    b. Schema requires **unique** email
        i. The length of password should >= 5 and <= 50
    c. Schema requires password
        i. The length of password should > 8 and <= 20
2. Update a new username
3. Update a new password
4. Get Json web token
5. Request a signup
6. Delete user account
7. Update user information

Integration Tests for Back-end:

1. Send Post requests to /user/signup
2. Send Post requests to /user/login
3. Send GET requests to /user/profile
4. Send POST requests to /user/profile
5. Send GET requests to /user/delete_account
6. Send POST requests to /user/username
7. Send POST requests to /user/password

Acceptance Tests for Front-end:

1. Sign up and log in using the newly created account
2. Manage user profile (change affiliation, bio, username, and password)
3. Manage account preferences (enable or disable notifications)
4. Delete an account

Core Feature: Create and Interact with Posts

Unit Tests for Back-end:

1. Post schema follows the requirements.
    a. Schema requires the user_id.
    b. Schema only requires the user_id. Other attributes, though required, need not be initialized since they have default values.
    c. post_date attribute is a date.
    d. Default value of content is ‘ ‘.
2. Comment schema follows the requirements.
    a. Schema requires the user_id and post_id.
    b. Schema only requires the user_id and post_id.
    c. comment_date attribute is a date.

    d. Default value of content is ‘ ‘.
3. Like schema follows the requirements.
    a. Schema requires the user_id and post_id.
    b. Schema only requires the user_id and post_id.
4. Get all the posts.
5. Get pages of posts with the posts sorted by post_date in descending order from the (fake) database.
    a. Return the correct posts over varying page sizes and page indices.
    b. Return a number of posts that is less than the page size if the end of the database has been reached.
    c. Return nothing if the page being accessed does not exist.
6. Get a single post by id.
7. Get all the posts made by a user.
8. Get all the posts from the subscribed users.
9. Create a post.
10. Remove a post by id.
11. Remove all the posts made by a user.
12. Update the content of a post.
13. Count the number of posts made by a user.
14. Get all the comments of a given post.
15. Create a comment.
16. Get a single comment by id.
17. Remove a comment by id.
18. Remove all the comments of a post.
19. Update the content of a comment.
20. Get the number of likes a comment has.
21. Determine if a user has liked a post.
22. Like a post (i.e., add one to the number of likes a post has).
23. Unlike a post (i.e., subtract one from the number of likes a post has).

Integration Tests for Back-end:

1. Send a GET request to api/post/get_recent_posts
2. Send a GET request to api/post/get_user_posts
3. Send a POST request to api/post/update
4. Send a DELETE request to api/post/update
5. Send a POST request to api/post/create
6. Send a GET request to api/comment/getNumLikes
7. Send a GET request to api/comment/userLikedPost
8. Send a POST request to api/comment/likePost
9. Send a POST requestto api/comment/unlikePost
10. Send a POST request to api/comment/create
11. Send a GET request to api/comment/getCommentsFromPost

Acceptance Tests for Front-end:

1. Create a post
2. Interact with posts (like and comment)

3. Share a post via a QR code

Core Feature: Create and Interact with Subscription

Unit Tests for Back-end:

1. Post schema follows the requirements.
    a. Schema requires creator_id
    b. Schame requires audience_id
2. Test get user audiences
3. Test sending email notification to subscribers
4. Test get user following
5. Test update notification setting
6. Test subscribe new creator
7. Test cancem subscription
8. Test get subscription state
9. Test get subscription details

Integration Tests for Back-end:

1. Send POST requests to /user/subscription/followNewUser
2. Send GET requests to /user/subscription/getAudience
3. Send GET requests to /user/subscription/getFollowing
4. Send POST requests to /user/subscription/setNotification
5. Send GET requests to /user/subscription/cancel
6. Send GET reuquests to /user/subscription/isSubscribed
7. Send GET requests to /user/subscription/getSubscription

Acceptance Tests for Front-end:

1. Subscribe and unsubscribe to a user
2. Receive an email showing the newly created posts of users you are subscribed to

2.2 Test Completeness

Testing will be complete for Sprint 3 when:

- Each feature developed has at least 10 unit tests.
- There are at least 10 integration tests.
- Each user story has at least one manual acceptance test.
- All automated test cases executed successfully.
- All open bugs are fixed or will be fixed in the next release.

3 Resource & Environment Needs

3.1 Testing Tools

General Tools/Methods:

- Git Hub Actions (for automation of tests)
- Git Hub Issues (for bug and feature tracking)

Back-end:

- Sinon
- Mocha and Chai
- Postman (for manual testing of back-end routes)

3.2 Test Environment

The minimum **hardware** requirements for testing our application are:

- A computer with a stable Internet connection, and sufficient disk space and RAM.

The minimum **software** requirements for testing our application are:

- Node.js version 16.15.1 or above
- Docker
- Windows 8 or above, or Mac OS Ventura 13.0 or above

4 Terms/Acronyms

Make a mention of any terms or acronyms used in the project

TERM/ACRONYM	DEFINITION
API	Application Program Interface
AUT	Application Under Test
JSON	Javascript Object Notation
JWT	JSON Web Token
CI	Continuous Integration
CD	Continuous Deployment
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol