Study Buddy

Test Plan

Andrea Abellera	abelleac@myumanitoba.ca
Katrina Dotzlaw	dotzlawk@myumanitoba.ca
Ryan Dotzlaw	dotzlawr@myumanitoba.ca
Millan David	davidm2@myumanitoba.ca
Elieser Capillar	capillae@myumanitoba.ca

1. Introduction

1.1 Scope

Feature 1: User Accounts

This feature will allow users to create accounts that will allow them to sign in and out of the Study Buddy to save their progress.

Feature 2: Track Study Time

This feature will allow users to track their study time and be presented with the progress of their time usage over a period of time.

1.2 Roles and Responsibilities

Name	Net ID	GitHub Username	Role
Andrea Abellera	abelleac	andreaabellera	Frontend QA Analyst
Katrina Dotzlaw	dotzlawk	kdotzlaw	Backend QA Analyst
Ryan Dotzlaw	dotzlawr	rdotzlaw	Backend QA Analyst
Millan David	davidm2	eliesercapillar	Frontend QA Analyst
Elieser Capillar	capillae	millandavid	Test Manager

2. Test Methodology

2.1 Test Levels

Feature 1: User Accounts

- 4 frontend unit tests
    - Validates user inputs are not blank
    - Validates that the password is at least 8 characters
- Unit testing on relevant Pinia store functions
- 3 backend database unit tests
    - **test_getUser():** Validate that specified user was retrieved from the database
    - **test_removeUser():** Validate that account created was correctly added to database
    - **test_createAccount():** Validate that a user was successfully removed from the database (for testing purposes)
- 3 backend API unit tests
    - **test_login():** Validate that a user is able to create a new session and log in to the server with valid credentials
    - **test_logout():** Validate that a user that is already logged in can log out
    - **test_newuser():** Validate that a new user is able to be made

Feature 2: Track Study Time

- 7 frontend unit tests
    - Test Timer class functions
        - Validates the time is correctly passed back
        - Validates the pause features works
        - Validates the timer can resume
- Unit testing on relevant Pinia store functions
- 7 backend database unit tests
    - **test_getClasses():** Validate that all classes for given user was retrieved from database for specified user
    - **test_getSingleClass():** validate that a single class was retrieved from the database for the specified user and class name
    - **test_ClassID():** Validate that the correct classID was retrieved when given a username and class name
    - **test_addClass():** Validate that class was correctly added to database for specified user using the required information provided

    - **test_removeClass():** Validate that class was correctly removed from database for specified user (for testing purposes)
    - **test_completeClass():** Validate that class is marked complete in the database for a specified user and class
    - **test_addStudyTime():** Validate that study time was updated in the database for the given class and specified user
- API unit tests
    - **test_allclasses()**: Validate that a logged in user is able to receive all the classes associated with their account
    - **test_class():** Validate that a logged in user is able to receive a single class associated with their account
    - **test_update_time():** Validate that a user is able to increase the total study time for an associated class

2.2 Test Completeness

- All unit test suites and included tests must pass for both frontend and backend on production release of the application.
- Tests should attain at least 75% coverage on each of frontend and backend functions or files.
- (For future sprints) All integration tests suites and acceptance tests must pass.

3. Resource and Environment Needs

3.1 Testing Tools

Automation Tools

- Jest
- UnitTest
- Github Actions

Requirements and Bug Tracking Tools

- GitHub Issues
- Jira

3.2 Test Environment

Frontend Testing Environment

- Node.js
- TypeScript

Backend Testing Environment

- Python
- Flask
- MSSQL
- Docker
- Ubuntu

4. Terms/Acronyms

Terms/Acronyms	Definition
MSSQL	Microsoft SQL
API	Application Program Interface