# 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.

### Feature 3: Calendar

This feature will allow users to store any important dates and reminders and have it displayed in a scrollable list. This list will display the upcoming important dates and how close it is to that date.

### Feature 4: Grade Calculator

The grade tracker will allow users to enter and store their grades and also calculate their current grade in the course. An extension of this feature would be able to calculate grades a user would need to achieve on future assignments/tests/projects in order to obtain a certain letter grade.

## 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

- 13 frontend unit tests
    - Validate suite functions
        - Validates user inputs are not blank
        - Validates that the password is at least 8 characters
    - Pinia store functions
        - loginUser, logoutUser, toggleModal, setModal
    - Vue component mounts
        - App.vue, Header.vue, Login.vue, Register.vue, Dashboard.vue, Modal.vue, ModalManager.vue
- 5 backend database unit tests
    - **test_getUser():** Validate that specified user was retrieved from the database
    - **test_getUserFail():** Validate that users not in database cannot be retrieved
    - **test_removeUser():** Validate that account created was correctly added to database
    - **test_removeUserFail():** Validate that a user not present in the database cannot be removed
    - **test_createAccount():** Validate that a user was successfully removed from the database (for testing purposes)
- 5 backend API Integration 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
    - **test_login_wrongpass():** Validates that a inputting a valid username, but an invalid password will not allow a user to log in
    - **test_logout_fail():** Validate that the logout fails in the user isn't currently logged in
- 3 automated user acceptance tests
    - Create a new account
    - Login existing user
    - Logout existing user

## Feature 2: Track Study Time

- 11 frontend unit tests
    - Timer suite functions
        - Validates the time is correctly passed back
        - Validates the pause features works
        - Validates the timer can resume
    - Pinia store functions
        - setTimer, setPageName, setStudyTime, setStudyClass
    - Vue component mounts
        - CreateClass.vue, Class.vue, Accordion.vue, ClassCards.vue
- 14 backend database unit tests
    - **test_getClasses():** Validate that all classes for given user was retrieved from database for specified user
    - **test_getClassesNone():** Validate that None is returned if a user has no classes
    - **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_ClassId_AttrError():** Validate that None is returned if a class does not exist
    - **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
    - **test_editClassMeta():** Validates that all requested class metadata is correctly updated in the database
    - **test_addClassCode():** Validates that a class code was successfully added to class.
    - **test_editClassReqData_Name():** Validates that the required class field ‘class_Name’ is correctly updated in the database
    - **test_editClassReqData_Timeslot():** Validates that the required class field ‘timeslot’ is correctly updated in the database

    - **test_addClassBreakdown():** Validates that a grade breakdown was added to the specified class.
- 10 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
    - **test_class_fail():** Validate that a logged in user is unable to receive a class that doesn't exist on their account
    - **test_update_time_fail():** Validate that a user cannot update time for a class that doesn't exist
    - **test_newclass():** Validate that a user is able to create a new class for their account
    - **test_updatemeta():** Validate that a logged in user is able to modify the metadata for their class (grade breakdown, professor information, etc.)
    - **test_editclass():** Validate that a logged in user is able to change the important data for their class (class name, timeslot)
    - **test_deleteclass():** Validate that a logged in user is able to remove one of their classes
    - **test_completeclass():** Validate that a logged in user is able to mark a class as completed
- 5 automated user acceptance tests
    - Create a new class
    - Check new class workspace has been created from Dashboard
    - Update class information
    - Update study sessions and study time from Dashboard
    - Update study sessions and study time from Class pages

## Feature 3: Calendar

- 5 frontend unit tests
    - Filter suite functions
        - Validates test assignment ordering
            - Checks for correct and incorrect ordering
        - Validates assignment output length
            - Checks for correct and incorrect output length
    - Pinia store functions

        - updateReqSignal
    - Vue component mounts
        - AddRequirement.vue, RequirementCards.vue
- 13 backend database unit tests
    - **test_getEmptyTasks():** Validates that None is returned if a given user has no tasks for the associated class
    - **test_getSingleTask():** Validates that a single task is correctly retrieved from the database for the given user and class
    - **test_getTaskList():** Validates that all tasks for a specified user and class are correctly retrieved
    - **test_getTaskID():** Validates that the correct taskID is retrieved for the specified user, class, and task
    - **test_getTaskIDFail():** Validates that None is returned if a task does not exist for the specified user and class
    - **test_completeTask():** Validates that the correct task is completed when a grade is added when given a user, class, and taskName
    - **test_uncompleteTask():** Validates that a task reset to incomplete (ie grade reset to 0)
    - **test_getCompleteTasksForClassSingle():** Validates that a single completed task is correctly retrieved from the database for the specified user and class
    - **test_getCompleteTasksForClassMulti():** Validates that all completed tasks for the specified user and class are retrieved by the database
    - **test_removeTask():** Validates that a removed task no longer appears in the database
    - **test_addTask():** Validates that a new task is present in the database for the specified user and class
    - **test_editTask():** Validates that various updates made to a task are reflected in the database for the specified user and class
    - **test_getDeadlines():** Validates that deadlines for tasks are correctly retrieved from database
- 10 API integration tests
    - **test_task():** Validate that a user can get a specific task on their account
    - **test_task_fail1():** Validate that a user cannot get a specific task for a class that doesn't exist
    - **test_task_fail2():** Validate that a user cannot get a specific task that doesn't exist
    - **test_alltasks():** Validate that a user can get a list of all tasks associated with one of their classes

    - **test_newtask():** Validate that a user is able to create a new task for one of their classes
    - **test_complete_task():** Validate that a user is able to mark a task as complete by changing the task's 'grade' attribute
    - **test_complete_tasks_fail():** Validate that a user is unable to mark a non-existing task as complete
    - **test_edittask():** Validate that a user is able to modify the important data for one of their tasks (task name, weight, deadline)
    - **test_deletetask():** Validate that a user is able to remove a task
    - **test_donetasks():** Validate that a user is able to get all the graded tasks for one of their classes
- 6 automated user acceptance tests
    - View important dates from Dashboard
    - View current and elapsed tasks from each Class
    - Create new task
    - Edit a current task
    - Delete an existing task
    - Complete an existing task by assigning a grade

## Feature 4: Grade Calculator

- 2 frontend unit tests
    - Vue component mounts
        - GradeCalculator.vue
    - Pinia store functions
        - updateGradeSignal
- 1 backend database unit tests
    - **test_addClassBreakdown():** Validate that a given string of class grade breakdown is correctly added to the specified class via update method
- 3 API unit tests
    - **test_grade():** Validate that a user is able to get a correct letter grade for a class
    - **test_grade_notasks():** Validate that a user in unable to get a letter grade for a class that they have no graded tasks for
    - **test_updatemeta():** Validate that a logged in user is able to modify the metadata for their class (grade breakdown, professor information, etc.)
- 3 automated user acceptance tests
    - View estimated letter grades from all classes

    - Completed requirements from each class should have a letter grade assigned
    - Create grading scheme

### Future Features: Buddy Level System

- 10 frontend unit tests
    - Filter suite functions
        - Validates the vowel grade
            - Both for cases where it is correct and incorrect
        - Validates consonant grade
            - Both for cases where it is correct and incorrect
        - Validates if deadlines are included
    - Pinia store functions
        - updateSkin, updateBuddy
    - Vue component mounts
        - Buddy.vue, Corgi.vue, Bunny.vue, Settings.vue, Wallpaper.vue
- 0 backend database unit tests
- 0 API unit tests
- 4 automated user acceptance tests
    - Check for tailored Buddy Feed based on user study progress
    - Check for tailored Buddy Feed based on approaching deadlines
    - Customize UI with unlockables
    - Customize companion with unlockables

## 2.2 Test Completeness

- Tests should attain at least 75% coverage on each of frontend and backend functions or files.
- All unit test suites and API integration tests must pass for both frontend and backend on production release of the application.
- All Vue components must successfully mount at any run.
- All acceptance tests suites must pass on final release of the application.
- To pass the load tests, the application must be able to handle 1000 requests for 100 users concurrently.

# 3. Resource and Environment Needs

## 3.1 Testing Tools

### Automation Tools

- Jest
- Cypress
- UnitTest
- Github Actions

### Requirements and Bug Tracking Tools

- GitHub Issues
- Jira

### Load Testing

- Locust

## 3.2 Test Environment

### Frontend Testing Environment

- Node.js
- JavaScript
- TypeScript

### Backend Testing Environment

- Python
- Flask
- MSSQL
- Docker
- Ubuntu

## 3.3 Acceptance Test Instructions

1. Start the Docker daemon service
   ```
   $  sudo service docker start
   ```
2. Clone the StudyBuddy repository
   ```
   $  git clone https://github.com/kdotzlaw/StudyBuddy.git
   ```
3. Inside the repository root, create a Docker shared network

```
$  sudo docker network create app-network
```

4. Then, pull the repository's Docker image

   ```
   $  sudo docker-compose up
   ```

5. Check that the frontend, backend, and MSSQL containers are running

   ```
   $  sudo docker ps
   ```

6. Go to the frontend directory

   ```
   $  cd frontend
   ```

7. Run automated acceptance tests

   ```
   $  npm run usertest
   ```

| | Spec | | Tests | Passing | Failing | Pending | Skipped |
|---|---|---|---|---|---|---|---|
| √ | Authentication.cy.js | 00:16 | 3 | 3 | - | - | - |
| √ | CalendarTracking.cy.js | 00:50 | 6 | 6 | - | - | - |
| √ | ClassManagement.cy.js | 00:16 | 3 | 3 | - | - | - |
| √ | Customization.cy.js | 00:18 | 4 | 4 | - | - | - |
| √ | GradeTracking.cy.js | 00:14 | 3 | 3 | - | - | - |
| √ | StudyTracking.cy.js | 00:24 | 2 | 2 | - | - | - |
| √ | All specs passed! | 02:20 | 21 | 21 | - | - | - |

End-of-test report

# 3.4 Load Testing Instructions

## Local

1. `pip install locust`
2. Once in the StudyBuddy directory, `cd backend`
3. In the backend directory, run the production server: `python start.py`
4. In the backend directory, run locust: `python -m locust`
5. Visit: `http://localhost:8089`
6. Enter the desired amount of users, users started per second, and `localhost:5000`
7. Start swarm.

## Production

1. Start the Docker daemon service

   ```
   $  sudo service docker start
   ```

2. Clone the StudyBuddy repository

   ```
   $  git clone https://github.com/kdotzlaw/StudyBuddy.git
   ```

3. Inside the repository root, create a Docker shared network

   ```
   $  sudo docker network create app-network
   ```

4. Then, pull the repository's Docker image
   `$ sudo docker-compose up`
5. Check that the frontend, backend, and MSSQL containers are running
   `$ sudo docker ps`
6. Go to the backend directory
   `$ cd backend`
7. Run load test
   `$ python -m locust`
8. Visit: `http://localhost:8089`
9. Enter the desired amount of users, users started per second, and `localhost:5000`
10. Start swarm.

# 4. Terms/Acronyms

| Terms/Acronyms | Definition |
|---|---|
| MSSQL | Microsoft SQL |
| API | Application Program Interface |
| UI | User Interface |
| SQL | Structured Query Language |
| NPM | Node Package Manager |