**Software Requirements and Design Document**

**For**

**Group <3>**

Version 3.0

**Authors**:

Luke B

# Overview (5 points)

The goal of the system is to be a hosted website acting as a student centric dual calendar and planner. It will be hosted on a website featuring several pages including a sign in page and the main page featuring the calendar/planner. The website’s will be made using HTML and CSS and will be connected to the backend using the Flask API, which will link to the backend database. The website is intended to be student centric displaying times for classes, upcoming homework assignments, study time for users, and more features. Along with this it will be connected to Canvas via the Canvas API to provide up to date information and get live updates.Functional Requirements (10 points)

1. Host a webpage for users to view a calendar/ planner. High.
2. Have a sign in page for users. High
3. Have a page for user settings. High.
4. Have a page for groups/messaging. High.
5. Have a database for storing user information. High.
6. Host webpage. High.
7. Host database. High.
8. Support scheduling/ adding events to calendar. High.

Following requirements were not achieved

1. Connect to the Canvas API to get users information for classes and upcoming assignments. High
2. Change view level of calendar, ie Day, Week, and Month level. Medium.
3. Have a button to automatically find and schedule study time. Medium.
4. Display important upcoming assignments. Medium.
5. Display recent messages on main page. Low.
6. Implement a texting notification service. Low.
7. Implement a notepad/journaling page. Low.
8. Import schedule from myFSU. Low.

Overall, despite missing many of our objectives we were able to achieve all of the High Priority requirements except for connecting to the Canvas API, to some extend which is good. Would have loved to get to more of the functional requirements we intended but this issue is touched on in the Progress Report Document.

# Non-functional Requirements (10 points)

1. Securely store user information on backend database.
2. Ensure webpage is fast and reliable.
3. Try to ensure all backend features such as saving user settings and importing data to and from database is quick and consistent.

These requirements could be said to have been reached as password were encrypted when entered into the database and while program was running website worked fine and accessing/storing in database was quick!

# Use Case Diagram (10 points)

For this project currently the only type of users are generic “Student Users”.

A current first iteration use case diagram is given below:

Diagram

Description automatically generated

Text Descriptions:

Sign In: The sign in Use case is used to give the user access to their specific account, which has their saved information and events. The only actor is the student user. The only precondition for signing in is being on the sign in page and having an existing account. The postconditions are that if the email/password were valid the user will be connected to their account and on the main webpage and if invalid the user will be alerted, they had an invalid username or password. The flow of events is essentially the same: on the sign in webpage the user enters their email into the email box, their password into the password box, and then clicks the sign in button. This then prompts the webpage to search the database for matching email/ password. If a matching combo is found, then the user is connected to their account in the database and is redirected to the main webpage with their calendar displayed. If no matching combo is found the user stays on the sign in page and an error message will appear on screen informing the user that they have an incorrect username or password.

Link to Canvas: The Link to Canvas use case is used to link a user’s account for the Student Companion to their Canvas account and then import their assignments from Canvas to our database. The preconditions are that the user has an existing account, is signed in, and has a canvas account. The postconditions are that the users Canvas sign-in information will be saved in the database and their Canvas assignments will be imported and added to our database and displayed on the calendar, along with user added events. The flow of events are as follows: assuming preconditions the user will select to connect to canvas, the webpage will prompt for their canvas ID and password, then the Python backend will use the Canvas API to interact with Canvas. If the given ID/Password are incorrect the operation will stop, and the user will be prompted that they have given an incorrect ID/ Password. If the information is correct the Canvas API will grab upcoming assignments and add them to our database, and the user’s Canvas information will be saved for future use, and then alert the user that the process was a success.

Change Calendar View: This use case will change the view level of the Calendar that is displayed on the main page. The preconditions are that the user is signed in and on the main page. The postconditions are that the calendar with display a different view level than is currently being displayed. Assuming preconditions the user will select the change view button and will be prompted with the different view levels available. Upon selecting the new view the webpage will use JavaScript to change the way the calendar is displayed based on which view level is selected and then refresh the calendar to display the new version.

Add Event: This use case is used to add events to the user’s calendar. The preconditions are that the user is signed in and on the main page. The post conditions will be that an extra event has been added to the calendar and database backend, and it is now displayed on whichever day it has been scheduled for. The flow of events are as follows: assuming preconditions the user will select the add event button on the main page. They will then be prompted to enter information about the event including length, start time, and date. Then the Python back end will enter this event into the database and then the calendar will be reloaded to include this new added event.

Schedule Study Time: This use case is used to automatically find, schedule, and add as an event to the user’s calendar a new event called “study time” based on the user’s current calendar. The preconditions are that the user is signed in and on the main page. The post conditions will be than an extra event called “Study Time” has been added to the calendar in a free time slot. The flow of events are as follows: assuming preconditions the user will select the schedule study time button on the main webpage. This will prompt the Python backend to go through the user’s schedule stored on the database and when an appropriate length of unobstructed time is found it will create a new event in this time called “Study Time” and add this as an event, similar to how the Add Event use case operates.

Form Group: This use case will be used to create groups for users to message and receive messages from other users. The preconditions are that the user is signed in and on the groups page. The post conditions will be that a new group is added to the groups page and the users account will be connected to the group. The flow of events are as follows: assuming preconditions, the user will select the form group button on the groups page. This will then prompt the user to enter the emails of other users they would like to be in the group and give the group a name. Once all emails have been entered the Python will create a new group adding all the emails entered, and the creator’s email, as group members (if an entered email does not exist, they will not be added) and the group will be created. This new group will then be displayed on the groups page and will be selectable to view messages in this group.

Send Message to Group: This use case is used to send messages in groups which were created in the previous use case. The preconditions are that the user is signed in, on the group page, and are a member in an existing group. The post conditions will be that a new message appears in the thread of group messages. The flow of events are as follows: assuming preconditions, the user will select an existing group on the groups page by left clicking the group. This will change the display of the portion of the screen to show the name of the selected group as well as display the existing messages in this group. There will be a text box at the bottom for the user to type a new message and upon hitting the send button the new group will be added to the message chai along with the email/username of the user who sent the message.

Change User Settings: This use case is used to change the settings of an existing user’s account. The preconditions are that the user is signed in and on the account settings page. The post conditions are that the settings page will display the new settings for the user and the new settings will be correctly stored, replacing old values, in the database. The flow of events are as follows: assuming preconditions, the user will have several textboxes to enter the new information. The user will then click the update information button and be prompted to enter their current password. The Python backend will then compare the entered password with the existing password, if they do not match the user will have the process cancelled and receive an error message. Otherwise, the new information will be entered into the database replacing old values and the page will reload with the new information being displayed.

# Class Diagram and/or Sequence Diagrams (15 points)

A sequence diagram works better for this project as we are not using a class structure, but what is more important is the order of events for each use case. Currently still in early stages so have not fully developed a sequence diagram. Most use cases would likely consist of a process like pushing a button on a webpage, the webpage interacts with the Flask API, which then either sends or requests data from the database.

Sequence Diagrams below:

Sign In:

A picture containing diagram

Description automatically generated

Link to Canvas:

Diagram

Description automatically generated

Change Calendar View:

Graphical user interface, text, application, email

Description automatically generated

Add Event:

Diagram

Description automatically generated

Schedule Study Time:

Diagram

Description automatically generated

Form Group:

Diagram

Description automatically generated

Send Message to Group:

Diagram

Description automatically generated

Change User Settings:

Diagram

Description automatically generated

# Operating Environment (5 points)

For increment three the webpage uses HTML, CSS, JavaScript, Flask API, SQLAlchemy for Database storage, and a few smaller libraries that are linked to Flask and SQLAlchemy, the full list is in the requirements.txt file. The project can be run on any device with a Python Interpreter and all of the requirements.txt libraries installed it runs with the command: python -m flask run.

We did not host the website as this is the end of the project (and this costs money) but if it was hosted on a webserver any user would have there data stored on the database and would not need to use Python to interact with anything. As this is the last increment the website is designed and intended to be viewed on a desktop computer, though a laptop can work as well but formatting may not line up completely as well as it would on a desktop, mobile is not supported.

# Assumptions and Dependencies (5 points)

As mentioned in part 6 for increment one the webpage is currently designed for being viewed on a laptop/desktop and would not look nice when viewed on a phone.

Along with this the user should have Python and the libraries necessary to run the project.