Sprint 2: UI & Functionality Implementation

Identify & Define

Preface: This sprint will aim at implementing the UI with a backend framework so it can be hosted.

FUNCTIONAL	NONFUNCTIONAL
Website can be hosted Hosted on Python flask	Application security
Base functionalities -- Account creation, login, deletion	Aesthetics

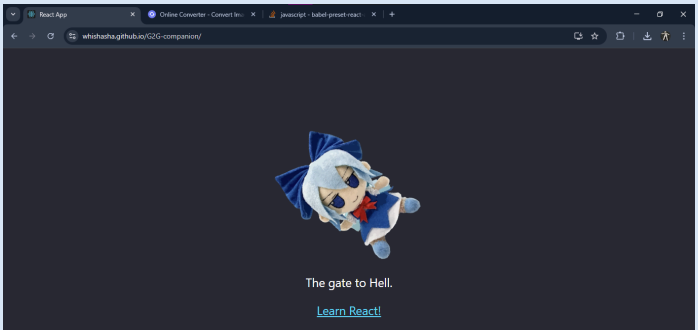
Specific Requirement	Justification
Create account	Be able to create user accounts
Delete account	Be able to delete user accounts
Change account details	Be able to change passwords, and classes taken
Login	Be able to login to the website for access to user pages

Research & Plan

Website Framework / Backend

A variety of frameworks were tested to see what could be used to run the code:

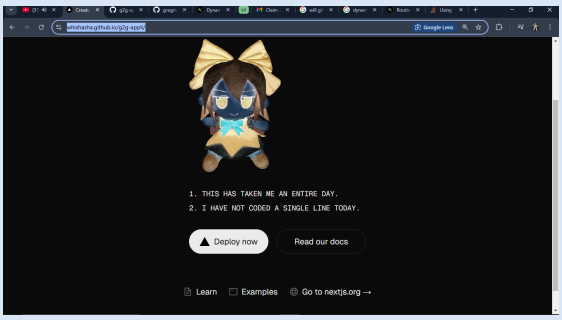
A total of 3 frameworks were attempted and evaluated for suitability.



Initially, **React** was used for its popularity in the industry, with a wide array of

documentation surrounding it since its inception in 2013.

However, due to the learning curve and the deprecated template for creating React apps (from *create-react-app*), this option was foregone.



Next, the **NEXT.JS** framework developed by Vercel in 2016 was tried. This was different from **React**, with Vercel offering hosting of NEXT.JS apps it seemed a viable next choice.
However, the learning curve for both of these new frameworks led me to resort to Python Flask, allowing me to use a familiar Python backend to kickstart the development process.

Python Flask does not require a template, offering valuable documentation in the form of a website:

https://flask.palletsprojects.com/en/stable/

This allowed me to quickly set up a webpage, with simple routing using their @app.route decorator.

```
@app.route("/")
def hello_world():
    return render_template('index.html')
```

An example of a webpage route using Flask

React		
PLUS	**MINUS**	**INTERESTING**
• Lots of documentation	• Learning curve • Outdated app template	• Extremely modular with React components

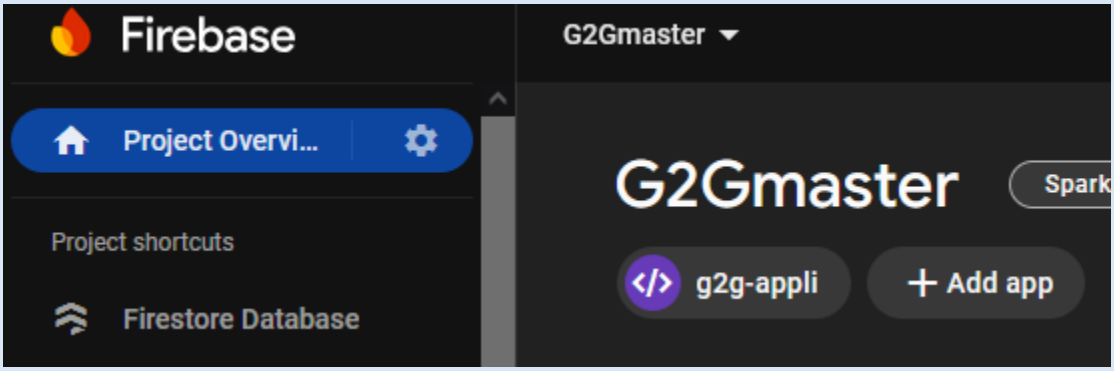
NEXT.JS		
PLUS	**MINUS**	**INTERESTING**
• Documentation • Website hosting is	• Learning curve	• Building blocks / root layouts

NEXT.JS		
supported by the same company which maintains the framework (Vercel)		

Flask		
PLUS	**MINUS**	**INTERESTING**
• Dynamic page routing • Familiarity	• Need to use different language for frontend v. backend	• Page templating using Jinja • Ability to use python in HTML using Jinja

Database Choices

Initially, Firebase was trialled for its wide usage in the industry and simple implementation. The Firestore version of Firebase was chosen for its long-term storage, scalability, and flexibility over the other option of Realtime Database.



An image showing the G2G firestore database

However, this option was switched from due to the subscription-based nature of Firebase, as well as its internet requirement. Due to the insensitive nature of the data being handled (first names, last names), and the relatively small scale of G2G Tutoring, a locally hosted SQL database was chosen instead for simplicity and cost-effectiveness.

Additionally, this promotes the response time of the website due to the SQL database being hosted locally as opposed to the cloud.

Firebase		
PLUS	**MINUS**	**INTERESTING**
• Cloud-based • Free	• Requires internet to function • Higher write/read counts to database require paid plans • Requires additional modules to be downloaded	• Node-based database structure

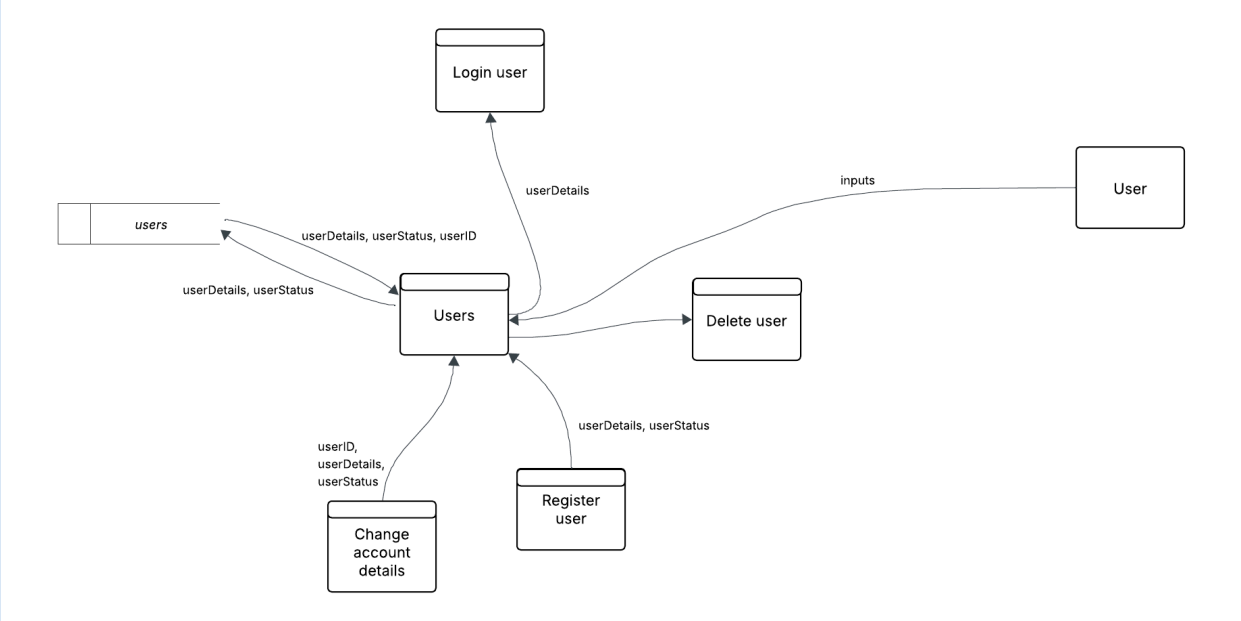
SQL (+sqlite3 module)		
PLUS	**MINUS**	**INTERESTING**
• Free • Locally-hosted • Sqlite3 module is built-in • Inputs are parameterised, granting safety	• suboptimal for storing files / images • Changing database fields is more troublesome than Firebase	• Database uses records

Registration/Login
Using SQL, the following table structure was defined to login a user:

Users Table Structure		
Field	**Data Type**	**Description**
ID	INTEGER	primary key used for identification
name	VARCHAR	user's display name
firstname	VARCHAR	user's first name
lastname	VARCHAR	user's last name

password	VARCHAR	user's encrypted password
is_tutor	BOOLEAN INTEGER	user's tutor status

Classes Table Structure		
Field	**Data Type**	**Description**
userID	INTEGER	User's ID
tutorID	INTEGER	ID of tutor in charge of user
is_english	BOOLEAN INTEGER	Checks if the user takes english
is_maths	BOOLEAN INTEGER	Check for if the user takes math

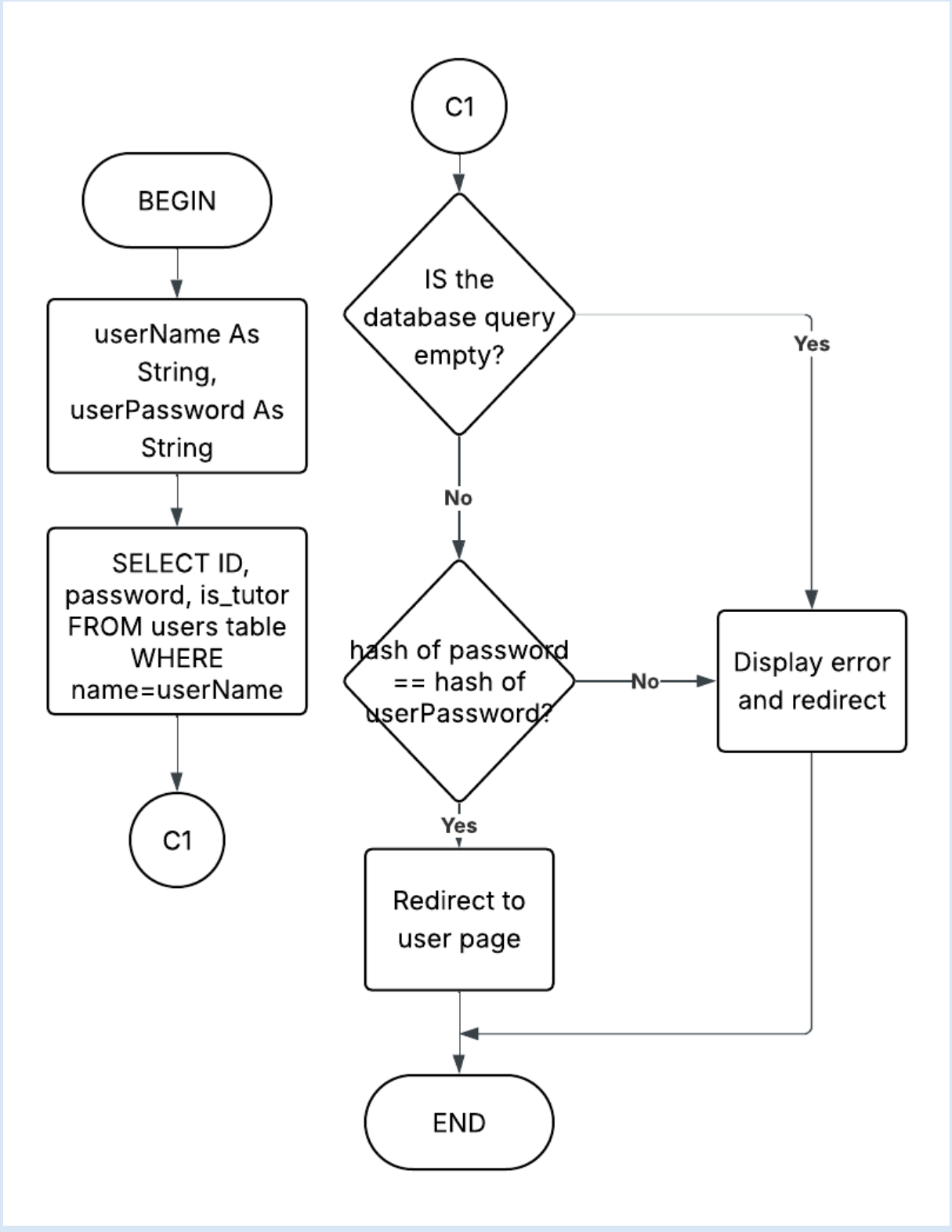


Produce & Implement

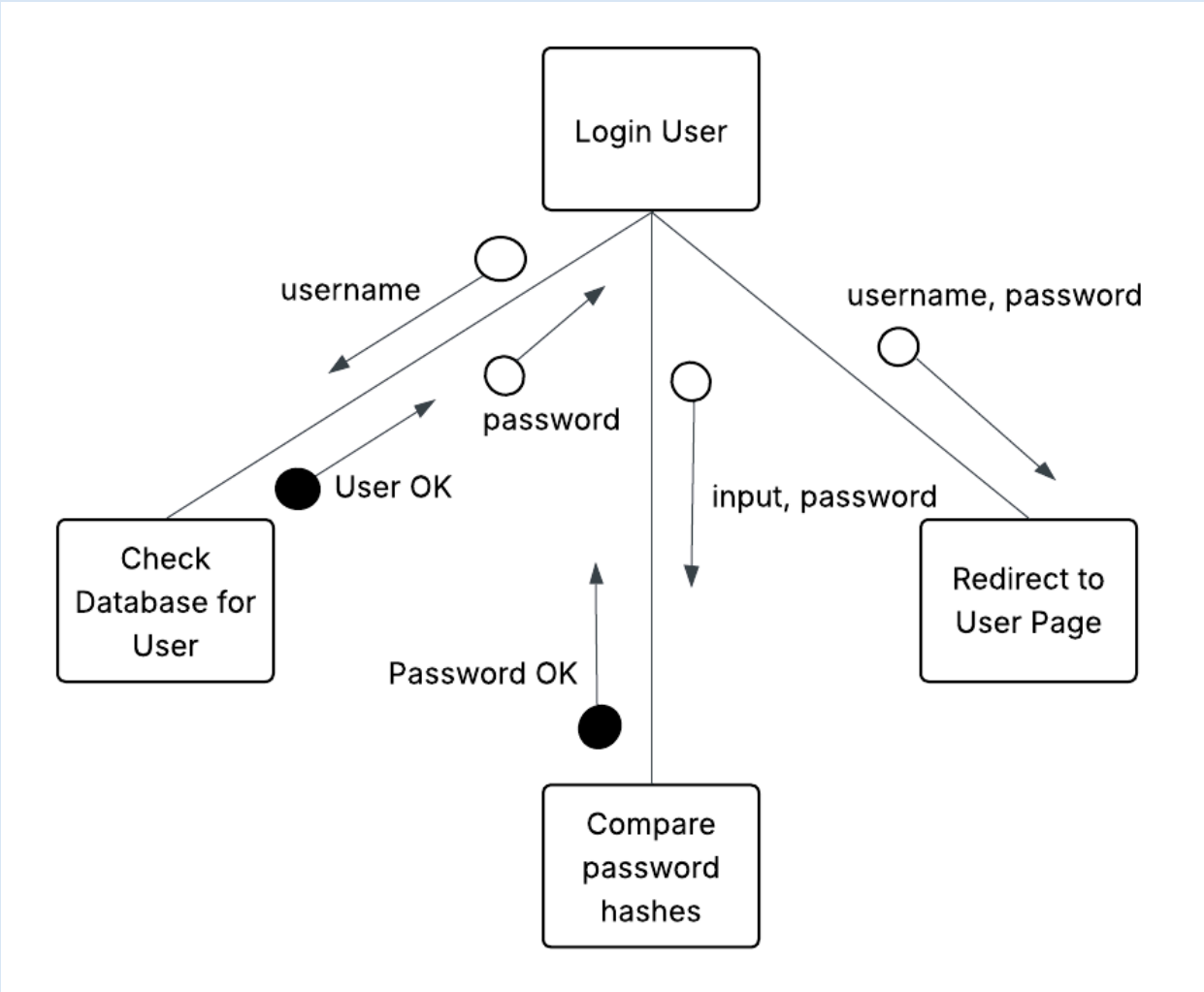
With these data structures in mind, the following algorithms were created:

Account Login

```python
@app.route("/login", methods = ['GET', 'POST'])
def login():
    print('User is accessing login page...')
    if request.method == "POST":
        name = request.form.get('name')
        password = request.form.get('password')
        con = sqlite3.connect('database.db')
        cur = con.cursor()

        userRecord = cur.execute('''SELECT ID, name, password, is_tutor FROM users WHERE name=?''', (name,)).fetchone()
        cur.close()
        con.close()
        if userRecord is None:
            flash('Invalid username')
            flash('Please try again')
            return redirect(request.url)
        if userRecord:
            passwordValidated = bcrypt.check_password_hash(userRecord[2], password)
            print(passwordValidated)
            if passwordValidated:
                #userRecord[0] = ID, [1] = username, [2] = encrypted password, [3] = is_tutor as a 1(TRUE) or 0 (FALSE)
                userID = str(userRecord[0])
                user = User(userID, userRecord[1], userRecord[2], userRecord[3])
                login_user(user) #from Flask_Login
                print('Successfully logged in')
                return redirect(url_for('user_home')) #add username as a parameter
            else:
                flash('Invalid password')
                flash('Please try again')
                return redirect(request.url)
    return render_template('login.html')
```


BEGIN
userName As String, userPassword As String
SELECT ID, password, is_tutor FROM users table WHERE name=userName
C1
C1
IS the database query empty?
Yes
No
hash of password == hash of userPassword?
No
Display error and redirect
Yes
Redirect to user page
END

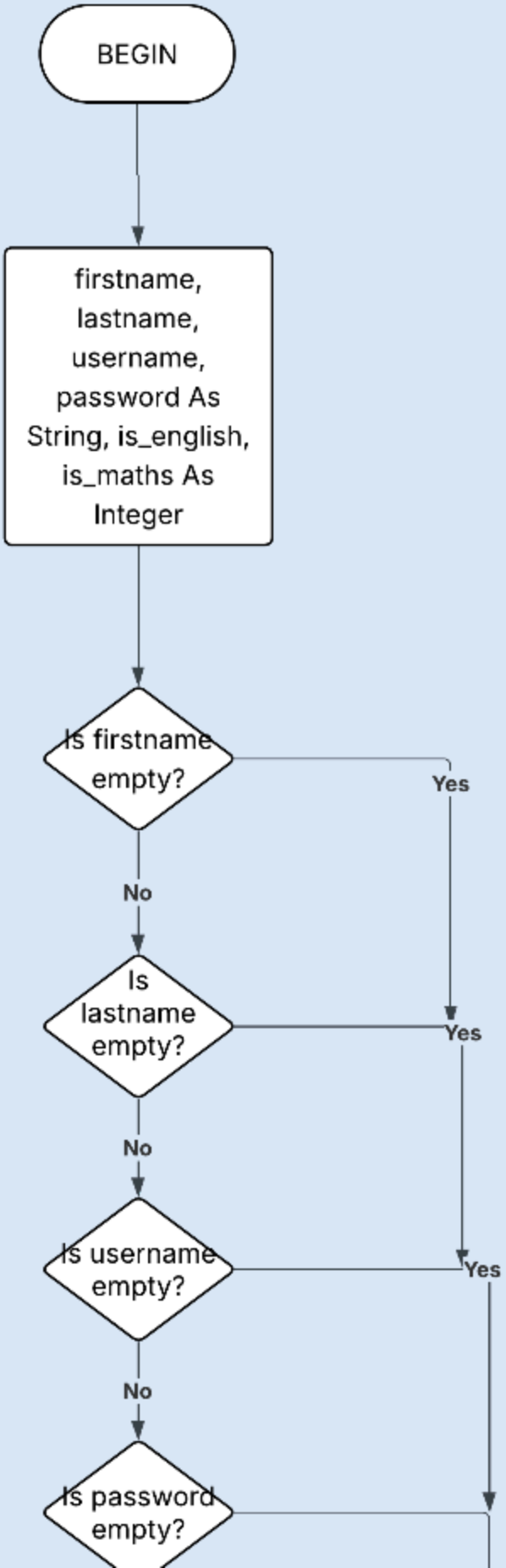
```
BEGIN
    userName As String
    userPassword As String
    SELECT ID, password, is_tutor FROM users table WHERE name=userName

    IF the database query is empty THEN
        IF password hash == userPassword hash THEN
            DISPLAY user page
        ELSE
            DISPLAY error
    ELSE
            DISPLAY error

END
```

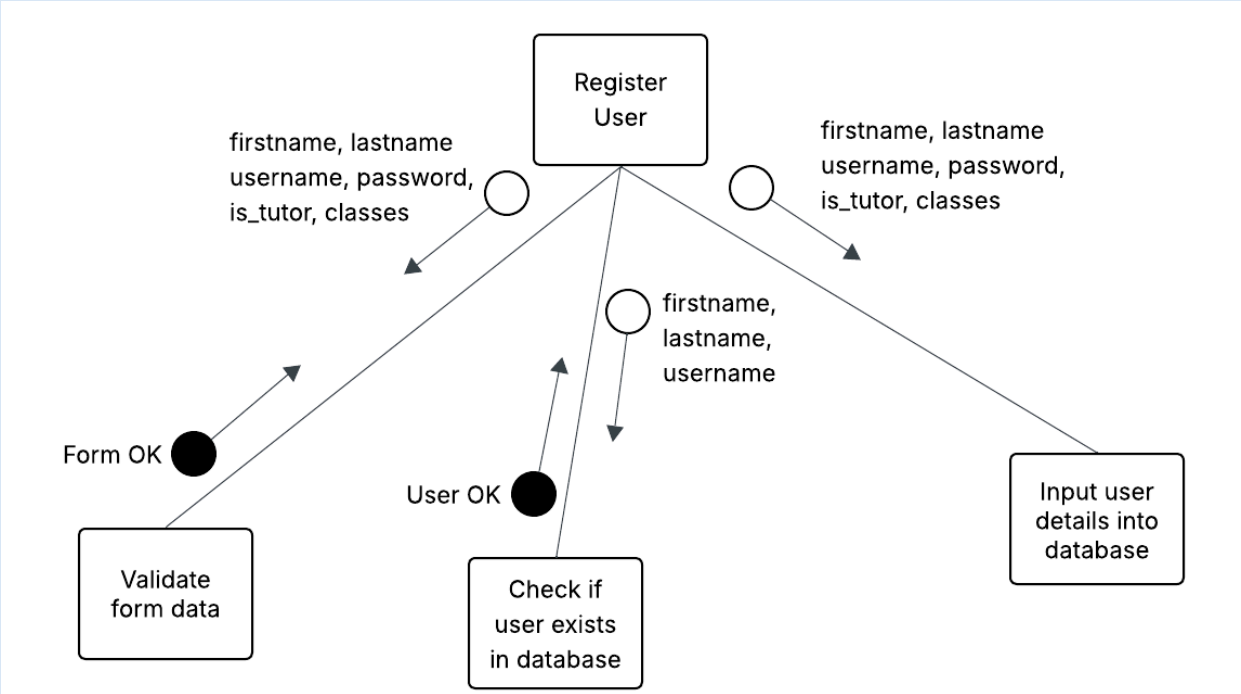


Account Registration

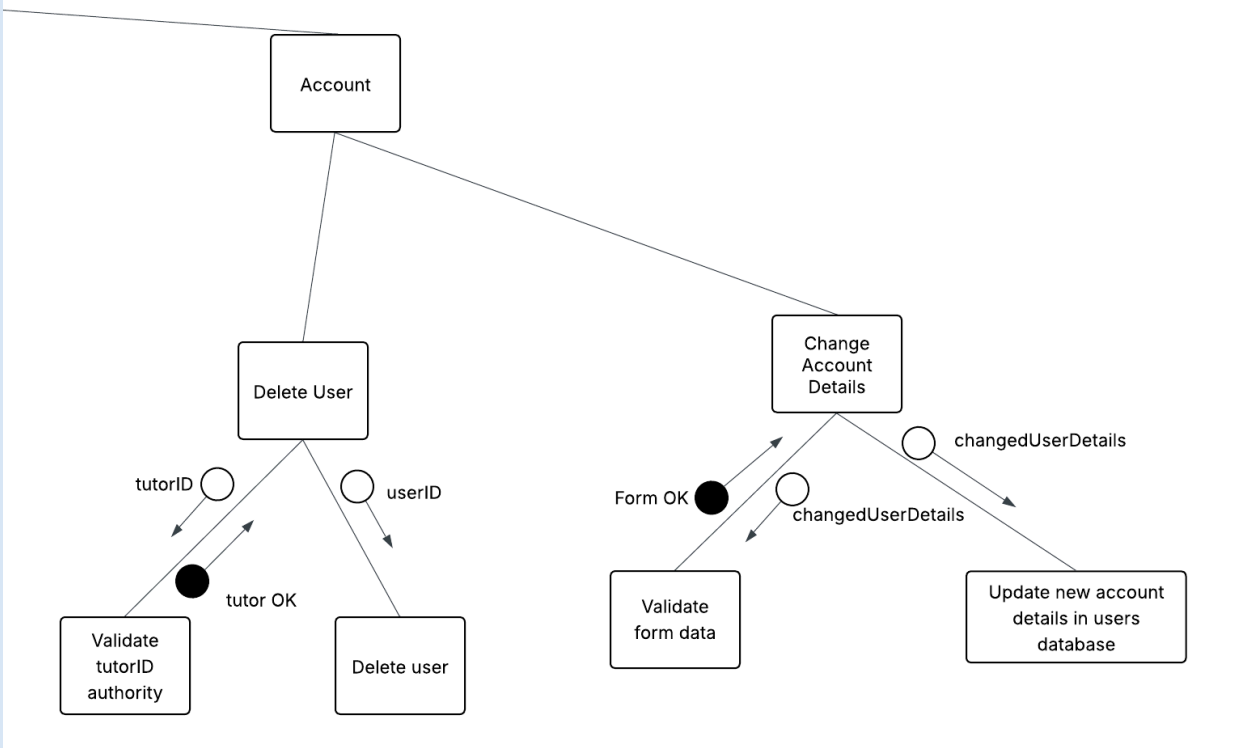
```
def register():
        try:
            firstname = str(request.form.get('firstname'))
            lastname = str(request.form.get('lastname'))

            username = str(request.form.get('username'))
            password = str(request.form.get('password'))

            is_english = int(request.form.get('is_english'))
            is_maths = int(request.form.get('is_maths'))

            is_english = assign_int_boolean(is_english) #Turns str value of checkbox to boolean integer (1: True, 2: False)
            is_maths = assign_int_boolean(is_maths)
        except TypeError as e:
            flash('Error. Invalid fields submitted')
            return redirect(request.url)

        if not firstname: #input validation
            print('Empty first name field!')
            return redirect(request.url)
        elif not lastname:
            print('Empty last name field!')
            return redirect(request.url)
        elif not username:
            print('Empty username')
            return redirect(request.url)
        elif not password:
            print('Invalid password')
            return redirect(request.url)
```

An example code snippet of registration validating inputs


BEGIN
firstname, lastname, username, password As String, is_english, is_maths As Integer
Is firstname empty?
Yes
No
Is lastname empty?
Yes
No
Is username empty?
Yes
No
Is password empty?




```
BEGIN
    firstname, lastname, username, password As String
    is_english, is_maths As Integer


    IF firstname is not empty THEN
        IF lastname is not empty THEN
            IF username is not empty THEN
                IF password is not empty THEN
                    IF is_english AND is_math integers THEN
                        IF password is strong enough THEN
                            IF user record with same firstname, lastname, and username DOES NOT EXIST in users table?
                                INSERT firstname, lastname, username, and password into users table
                                INSERT is_english, is_maths into classes table
                                RETURN
    DISPLAY error
    RETURN

END
```



Data Dictionary				
Variable Name	**Type**	**Description**	**Validation**	**Scope**
User	Class	Defines what a user is and stores session data for Flask-Login	Has id, username, password, and is_tutor fields	Global
login_manager	Instance of class LoginManager()	Handles user sessions for logins	Is of class LoginManager() from the Flask-Login module	Global
app	Instance of class Flask()	The object used to set configurations for the website	Is of class Flask() from the Flask module	Global
csrf	Instance of class CSRFProtect(app)	Object used to handle CSRF tokens within	Is of class CSRFProtect	Global

Data Dictionary				
		forms in HTML templates		
current_user	Instance of class User()	Refers to the session's user, containing the parameters of the User class	Is of class User()	Global

Data Dictionary **Login**				
Variable Name	**Type**	**Description**	**Validation**	**Scope**
name	string	Name entered by user for attempted login	Maximum 100 characters	Local
password	string	Password entered by user for attempted login	Between 8 and 128 characters	Local
con	instance	Used to connect to the database	----	Local
cur	instance	Used to send SQL queries to the database	----	Local
userRecord	list	Stores user data fetched from the users database	Is a list containing valid data from users database	Local
passwordValidated	boolean	Checks if hashed passwords match	True/False	Local
userID	string	userID of registered user. Stored as a string for easy	Must be an integer	Local

Data Dictionary **Login**				
		concatenation		
user	instance	Instance of User() containing user data such as name and password	Is an object containing a user's parameters	Local

Data Dictionary **Register**				
Variable Name	**Type**	**Description**	**Validation**	**Scope**
firstname	string	Registree's first name	Between 1 and 100 characters	Local
lastname	string	Registree's last name	Between 1 and 100 characters	Local
username	string	Registree's chosen username	Between 1 and 100 characters	Local
password	string	Registree's chosen password	Between 8 and 128 characters	Local
is_english	string	Defines whether the registree takes English or not	Is a string	Local
is_maths	string	Defines whether the registree takes Maths or not	Is a string	Local
is_tutor	boolean integer	Defines whether the registree is a tutor or not	Is either 1 or 0	Local
passwordChec	tuple	Tuple	{integer, string}	Local

Data Dictionary **Register**				
k		containing: [0]: a boolean value of whether or not the password is strong enough, and [1]: an error message if it isn't		
encryptedpassword	string	Bcrypt encrypted user password	Is an unreadable hash	Local
usernameCheck	list	Checks if the user exists in the database already	Is either None or not	Local
registereduserID	list	Gets registered user's ID	Is not None (this is okay as the SQL table auto increments integer primary keys, which this is)	Local

Data Dictionary **users Table**				
Variable Name	**Type**	**Description**	**Validation**	**Scope**
ID	integer	Identifier of user	Must be an integer	Global
name	varchar(100)	Username of user	Must contain a maximum of 100 characters	Global
firstname	varchar(100)	First name of user	Must contain a maximum of 100 characters	Global

Data Dictionary **users Table**				
lastname	varchar(100)	Last name of user	Must contain a maximum of 100 characters	Global
password	TEXT	Encrypted password of user	Must contain characters	Global
is_tutor	boolean	Determines whether a user is a tutor or not	Must be 0 or 1	Global

Test & Evaluate