2022

Clinical Feedback Instructional Tool

CSC 400 PROJECT REPORT

DAVID CELOZZI, DEREK GONG, AND JEREMY CORPUZ

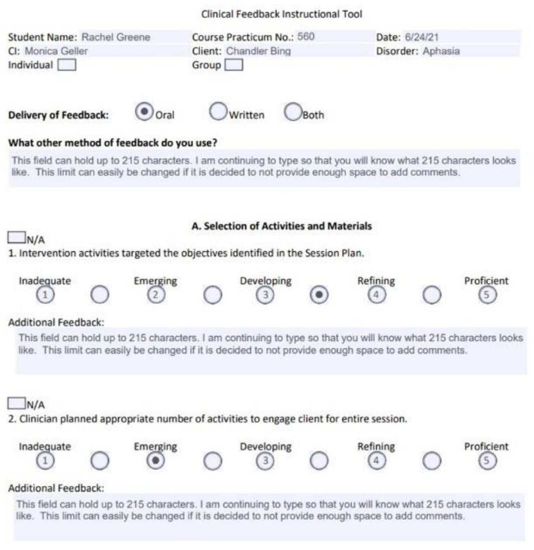
SUMMER 2022

The Communication Disorders Department at Southern Connecticut State University does not have an assessment application for clinicians and students. Senior and Graduate level students are required to diagnose and treat clients with various communication disorders as part of their degree curriculum. The clinical Instructors are currently performing assessments using pen and paper.

This application will provide the SCSU Communications Disorders Department a method to analyze, assess and evaluate graduate student clinicians' growth from the beginning to the end of a semester, across semesters and skill comparison across disorder types. Our vision of the application is to provide a comprehensive easy to use application that replicates the paper form to provides all users access to assessments and student information and data.

The application will contain levels or roles of different user types. These user types will have access to specific features or data within the application. Administrators will have the ability to create users and their roles. Instructors can maintain student evaluations, add and remove students and clients and view specific analytics related to each student. Students can also view their assessments and analytics along with Instructor comments. Communication Disorders Department faculty, Clinical Instructors, Director of Clinical Education, Clinic Manager and Clinical Field Coordinator will all be categorized as Instructors, and Graduate student clinicians will be categorized as Students. Any faculty member can be an Administrator. Graduate students can have multiple Clinical Instructors in a semester. Clinical Instructors will be able to view other Clinical Instructors reviews of the student clinicians.

Our solution is the Clinical Feedback Instructional Tool or CFIT. This application is a Web application developed using python and the Flask web framework, hosted through the Google Cloud Platform. The backend of the application is written in Python on the Flask web framework. The front end consists of frameworks such as bootstrap and is written using a combination of Javascript, CSS and HTML. The application features the ability to login to an Admin, Instructor or Student level, create a new student assessment for a specific client, view a student's completed assessments, see student analytics of past and current assessments. All of the data logged into this application will be both HIPPA and FRPA compliant and subsequently stored in a MySQL Database via SQLALchemy.



Clinical Feedback Instructional Tool

Student Name: Rachel Greene	Course Practicum No.: 560	Date: 6/24/21
CI: Monica Geller	Client: Chandler Bing	Disorder: Aphasia
Individual ☐	Group ☐	

Delivery of Feedback: [✓ Oral] Written Both

What other method of feedback do you use?

This field can hold up to 215 characters. I am continuing to type so that you will know what 215 characters looks like. This limit can easily be changed if it is decided to not provide enough space to add comments.

A. Selection of Activities and Materials

☐ N/A

1. Intervention activities targeted the objectives identified in the Session Plan.

Inadequate 1 ○ Emerging 2 ○ Developing 3 [✓ ○] Refining 4 ○ Proficient 5

Additional Feedback:

This field can hold up to 215 characters. I am continuing to type so that you will know what 215 characters looks like. This limit can easily be changed if it is decided to not provide enough space to add comments.

☐ N/A

2. Clinician planned appropriate number of activities to engage client for entire session.

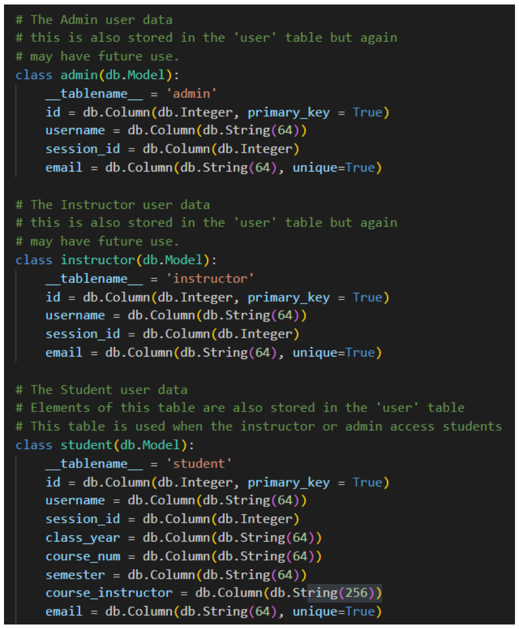
Inadequate 1 ○ [✓ Emerging 2] ○ Developing 3 ○ Refining 4 ○ Proficient 5

Additional Feedback:

This field can hold up to 215 characters. I am continuing to type so that you will know what 215 characters looks like. This limit can easily be changed if it is decided to not provide enough space to add comments.

Figure 1. The paper version of the assessment

Implementation



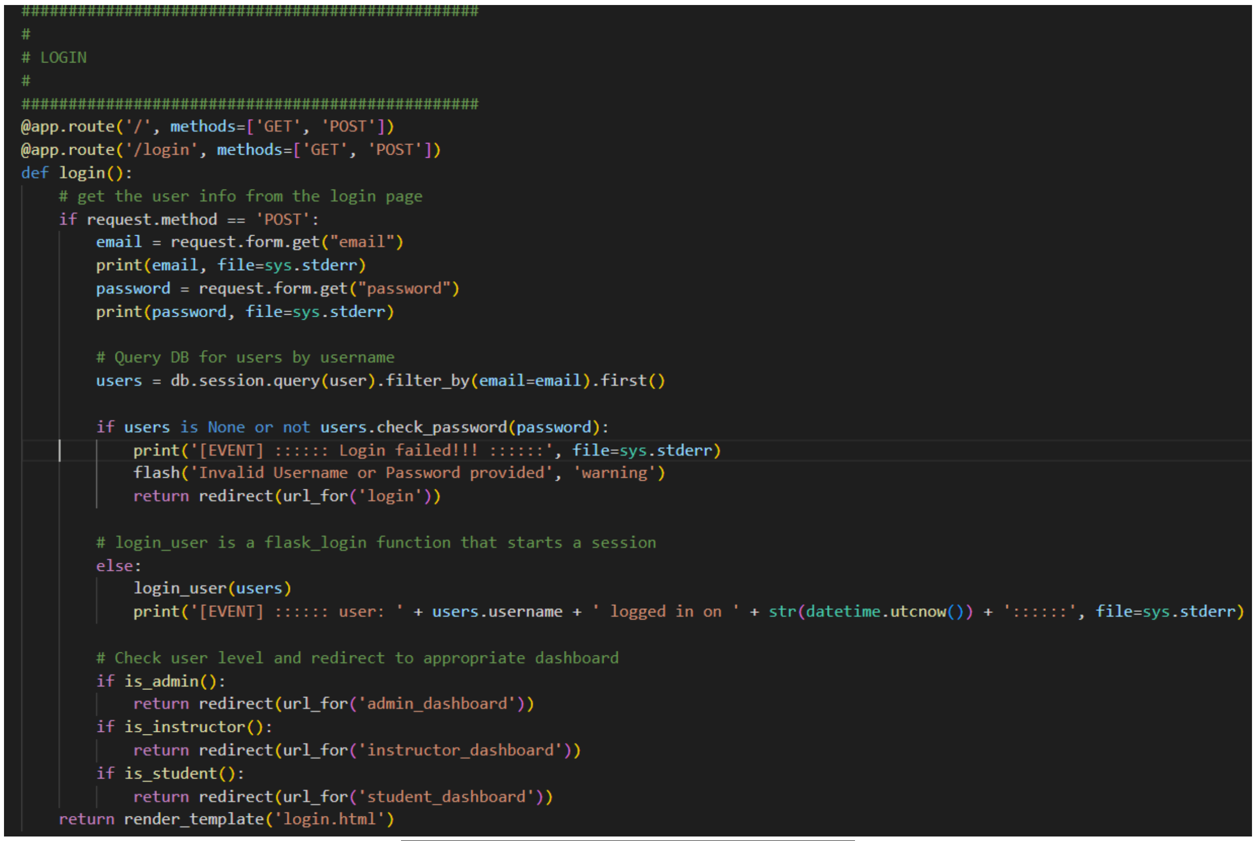
```python
# The Admin user data
# this is also stored in the 'user' table but again
# may have future use.
class admin(db.Model):
    __tablename__ = 'admin'
    id = db.Column(db.Integer, primary_key = True)
    username = db.Column(db.String(64))
    session_id = db.Column(db.Integer)
    email = db.Column(db.String(64), unique=True)

# The Instructor user data
# this is also stored in the 'user' table but again
# may have future use.
class instructor(db.Model):
    __tablename__ = 'instructor'
    id = db.Column(db.Integer, primary_key = True)
    username = db.Column(db.String(64))
    session_id = db.Column(db.Integer)
    email = db.Column(db.String(64), unique=True)

# The Student user data
# Elements of this table are also stored in the 'user' table
# This table is used when the instructor or admin access students
class student(db.Model):
    __tablename__ = 'student'
    id = db.Column(db.Integer, primary_key = True)
    username = db.Column(db.String(64))
    session_id = db.Column(db.Integer)
    class_year = db.Column(db.String(64))
    course_num = db.Column(db.String(64))
    semester = db.Column(db.String(64))
    course_instructor = db.Column(db.String(256))
    email = db.Column(db.String(64), unique=True)
```

Figure 2. Users database design in *models.py*

We have three main user roles of the application: admin, instructor, and student. This screenshot shows the setup of the user database table. The admin and instructor tables contain the user id, username, session_id, and email. The student table contains user id, session_id, class_year, course_num, semester, the course_instructor assigned to the student, and email. These extra pieces of information associated with the student help identify the student from the admin and instructor side and allow the student to be tied to an instructor and course. The id and email of each user type are unique, as these are the primary pieces of information distinguishing users. Users can have the same name or password, but not the same id. Some elements have been added to the Assessment tables to accommodate future feature development.



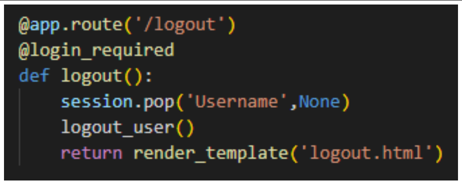
```
##################################################
#
# LOGIN
#
##################################################
@app.route('/', methods=['GET', 'POST'])
@app.route('/login', methods=['GET', 'POST'])
def login():
    # get the user info from the login page
    if request.method == 'POST':
        email = request.form.get("email")
        print(email, file=sys.stderr)
        password = request.form.get("password")
        print(password, file=sys.stderr)

        # Query DB for users by username
        users = db.session.query(user).filter_by(email=email).first()

        if users is None or not users.check_password(password):
            print('[EVENT] :::::: Login failed!!! ::::::', file=sys.stderr)
            flash('Invalid Username or Password provided', 'warning')
            return redirect(url_for('login'))

        # login_user is a flask_login function that starts a session
        else:
            login_user(users)
            print('[EVENT] :::::: user: ' + users.username + ' logged in on ' + str(datetime.utcnow()) + '::::::', file=sys.stderr)

        # Check user level and redirect to appropriate dashboard
        if is_admin():
            return redirect(url_for('admin_dashboard'))
        if is_instructor():
            return redirect(url_for('instructor_dashboard'))
        if is_student():
            return redirect(url_for('student_dashboard'))
    return render_template('login.html')
```

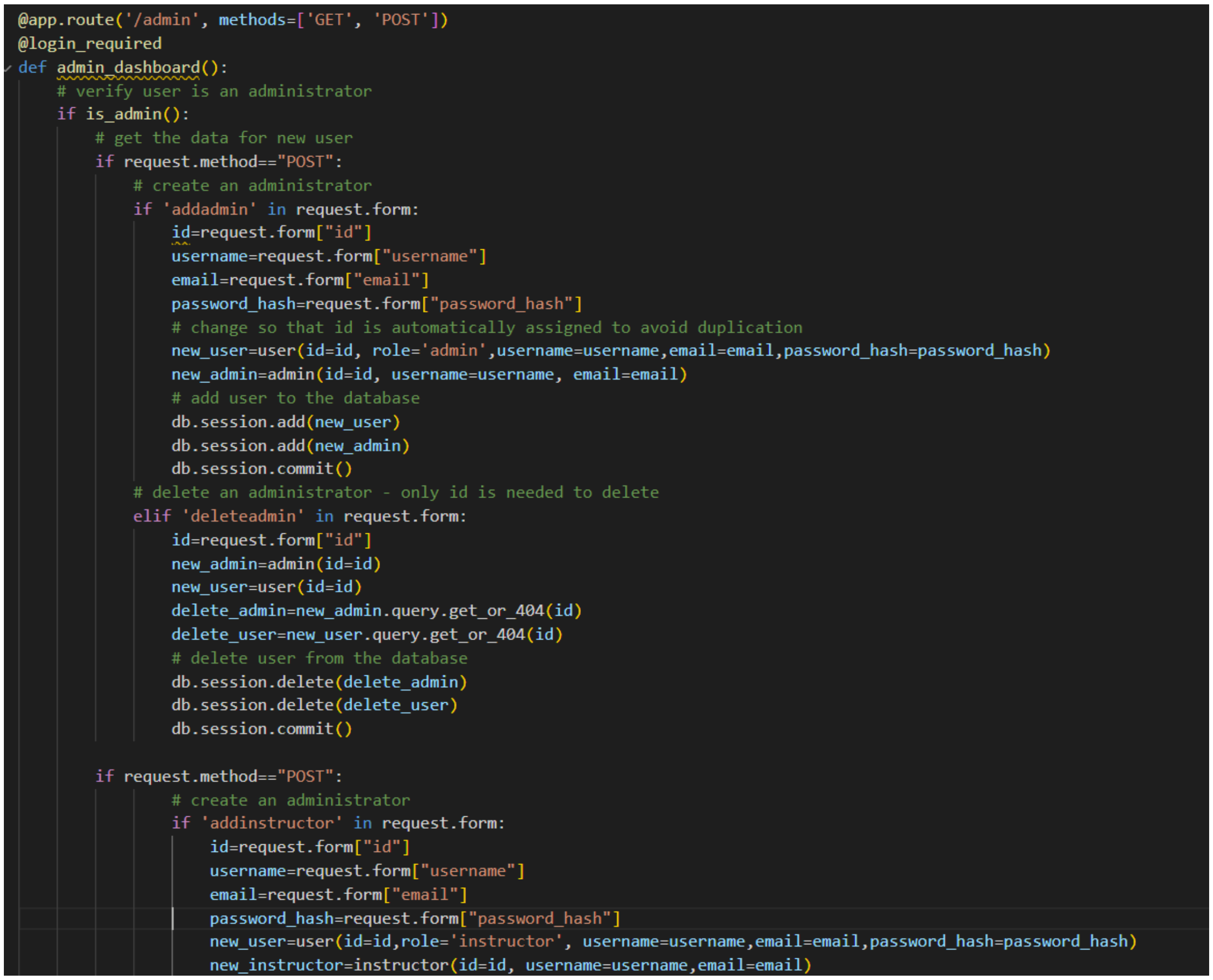


```
@app.route('/logout')
@login_required
def logout():
    session.pop('Username',None)
    logout_user()
    return render_template('logout.html')
```

Figure 3. Logging in and logging out in *routes.py*

Login_user(users) logs in a user to the system and authenticates the user. When logging into the system, the functions if is_admin(), if is_instructor(), and if is_student() checks the user role based on the id and email associated with the user, and only allows the user to view its assigned dashboard and perform their assigned duties. The admin can view the admin and instructor dashboard, the instructor can view the instructor dashboard, and the student can view the student dashboard.

Logout_user() logs out the user that is logged in using login_user(users). This causes the role to be returned as None, therefore when a user logs out, they cannot access any of the pages that require the user to be logged in.

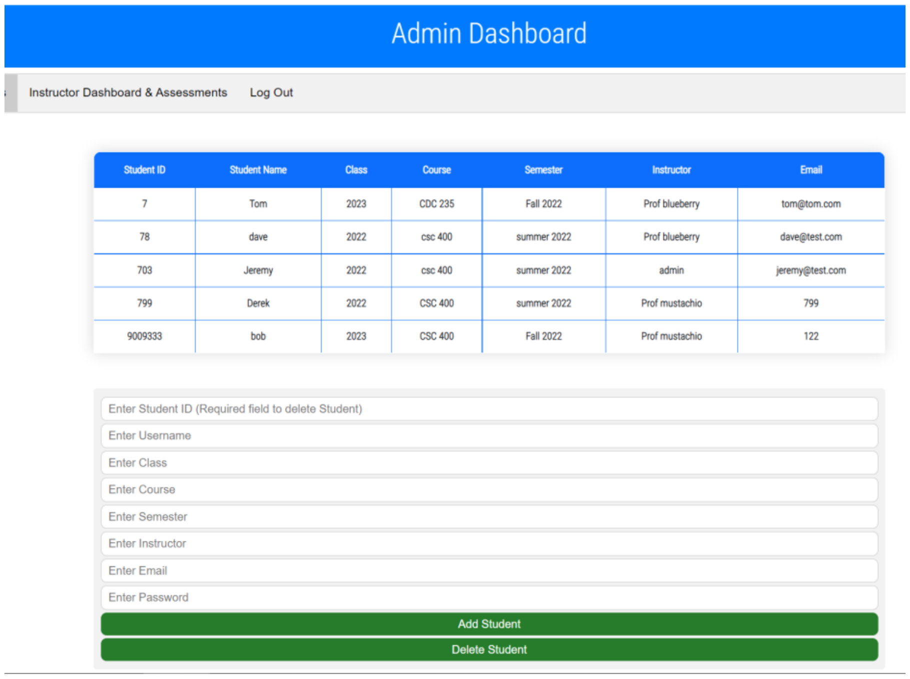


```
@app.route('/admin', methods=['GET', 'POST'])
@login_required
def admin_dashboard():
    # verify user is an administrator
    if is_admin():
        # get the data for new user
        if request.method=="POST":
            # create an administrator
            if 'addadmin' in request.form:
                id=request.form["id"]
                username=request.form["username"]
                email=request.form["email"]
                password_hash=request.form["password_hash"]
                # change so that id is automatically assigned to avoid duplication
                new_user=user(id=id, role='admin',username=username,email=email,password_hash=password_hash)
                new_admin=admin(id=id, username=username, email=email)
                # add user to the database
                db.session.add(new_user)
                db.session.add(new_admin)
                db.session.commit()
            # delete an administrator - only id is needed to delete
            elif 'deleteadmin' in request.form:
                id=request.form["id"]
                new_admin=admin(id=id)
                new_user=user(id=id)
                delete_admin=new_admin.query.get_or_404(id)
                delete_user=new_user.query.get_or_404(id)
                # delete user from the database
                db.session.delete(delete_admin)
                db.session.delete(delete_user)
                db.session.commit()

        if request.method=="POST":
                # create an administrator
                if 'addinstructor' in request.form:
                    id=request.form["id"]
                    username=request.form["username"]
                    email=request.form["email"]
                    password_hash=request.form["password_hash"]
                    new_user=user(id=id,role='instructor', username=username,email=email,password_hash=password_hash)
                    new_instructor=instructor(id=id, username=username,email=email)
```

Figure 4. Adding and viewing users in *routes.py*

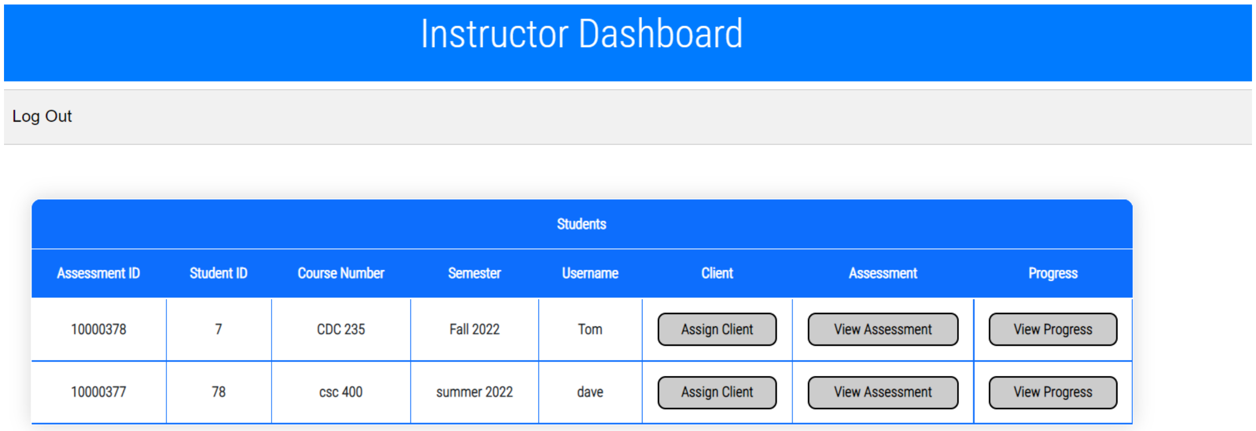
This piece of code demonstrates adding an admin. From the HTML page, it checks for the form with method == ”POST”. From there, all the information is retrieved and stored into the new_user and new_admin variable. New user adds the user to the user table in the database and new admin adds the user to the admin table. We need both tables as the user table contains the login information and the admin table contains the admin information. Return render_template(“admin.html”, admin=user.query.filter_by(role=’admin’).all() returns the value of all users where the role is equal to admin and displays it on the table. The same method is repeated to add and delete instructors and students and display all the current instructors and students. This code is mimicked in the instructor_dashboard for the creation and deletion of clients.



Student ID	Student Name	Class	Course	Semester	Instructor	Email
7	Tom	2023	CDC 235	Fall 2022	Prof blueberry	tom@tom.com
78	dave	2022	csc 400	summer 2022	Prof blueberry	dave@test.com
703	Jeremy	2022	csc 400	summer 2022	admin	jeremy@test.com
799	Derek	2022	CSC 400	summer 2022	Prof mustachio	799
9009333	bob	2023	CSC 400	Fall 2022	Prof mustachio	122

Figure 6. Student table for adding and deleting students

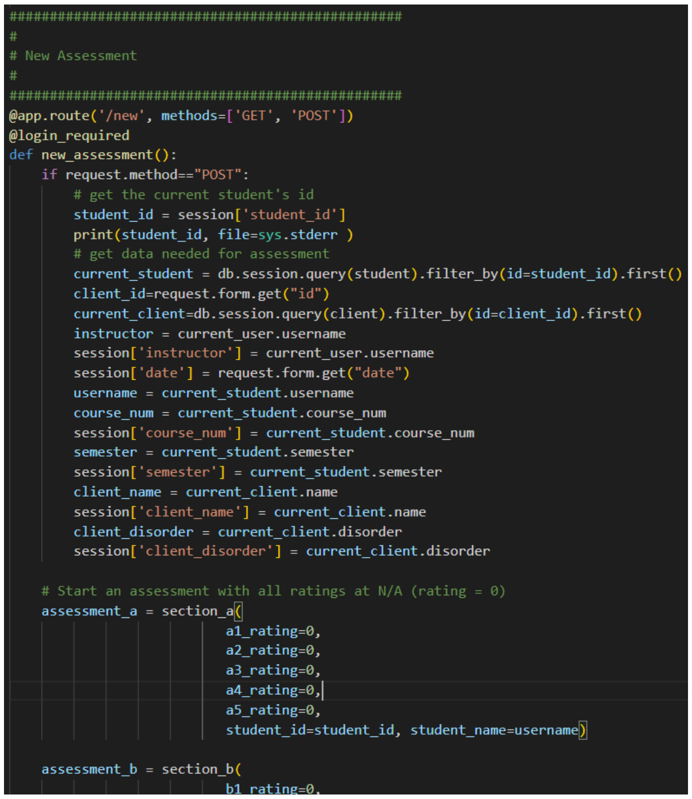
Creating a new assessment requires traversing a few functions. When an instructor wishes to create a new assessment for a student, a client needs to be assigned to that student and assessment. This is done via the instructor dashboard client table. This table contains the "Create Assessment" button which initiates a new assessment form for the instructor to assess that student and client.



Students							
Assessment ID	Student ID	Course Number	Semester	Username	Client	Assessment	Progress
10000378	7	CDC 235	Fall 2022	Tom	Assign Client	View Assessment	View Progress
10000377	78	csc 400	summer 2022	dave	Assign Client	View Assessment	View Progress

Figure 5. Assigning a client to the assessment

Next, the new_assessment function is called which gathers all data needed for the assessment and creates blank assessment with all ratings initially set to ’0’ or N/A. There are five assessment sections containing questions or items for the instructor to rate.



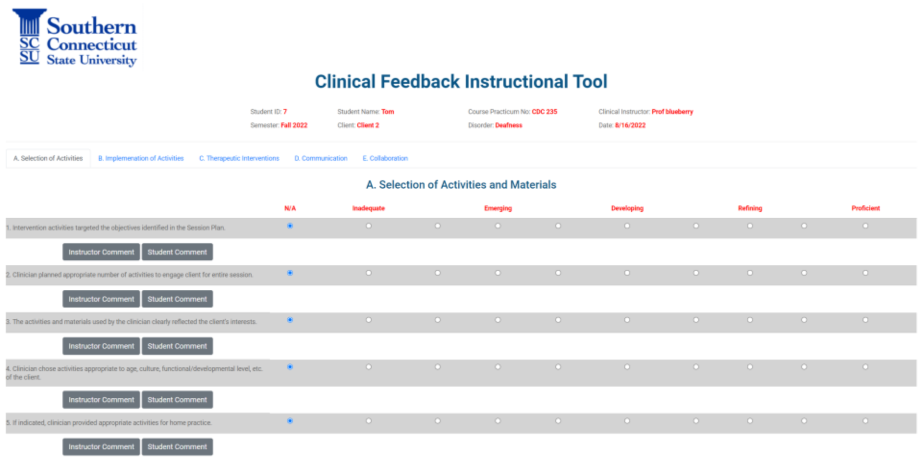
```
##################################################
#
# New Assessment
#
##################################################
@app.route('/new', methods=['GET', 'POST'])
@login_required
def new_assessment():
    if request.method=="POST":
        # get the current student's id
        student_id = session['student_id']
        print(student_id, file=sys.stderr )
        # get data needed for assessment
        current_student = db.session.query(student).filter_by(id=student_id).first()
        client_id=request.form.get("id")
        current_client=db.session.query(client).filter_by(id=client_id).first()
        instructor = current_user.username
        session['instructor'] = current_user.username
        session['date'] = request.form.get("date")
        username = current_student.username
        course_num = current_student.course_num
        session['course_num'] = current_student.course_num
        semester = current_student.semester
        session['semester'] = current_student.semester
        client_name = current_client.name
        session['client_name'] = current_client.name
        client_disorder = current_client.disorder
        session['client_disorder'] = current_client.disorder

    # Start an assessment with all ratings at N/A (rating = 0)
    assessment_a = section_a(
                           a1_rating=0,
                           a2_rating=0,
                           a3_rating=0,
                           a4_rating=0,
                           a5_rating=0,
                           student_id=student_id, student_name=username)

    assessment_b = section_b(
                           b1_rating=0,
```

Figure 6. Starting a new assessment

The rating scale is 1 to 5 including .5 point increments. The student is read from the student table and sent to the assessment.html page to be displayed on the top of the assessment form.


Southern Connecticut State University

Clinical Feedback Instructional Tool

Student ID: 7 Student Name: Tom Course Practicum No: CDC 235 Clinical Instructor: Prof blueberry
Semester: Fall 2022 Client: Client 2 Disorder: Deafness Date: 8/16/2022

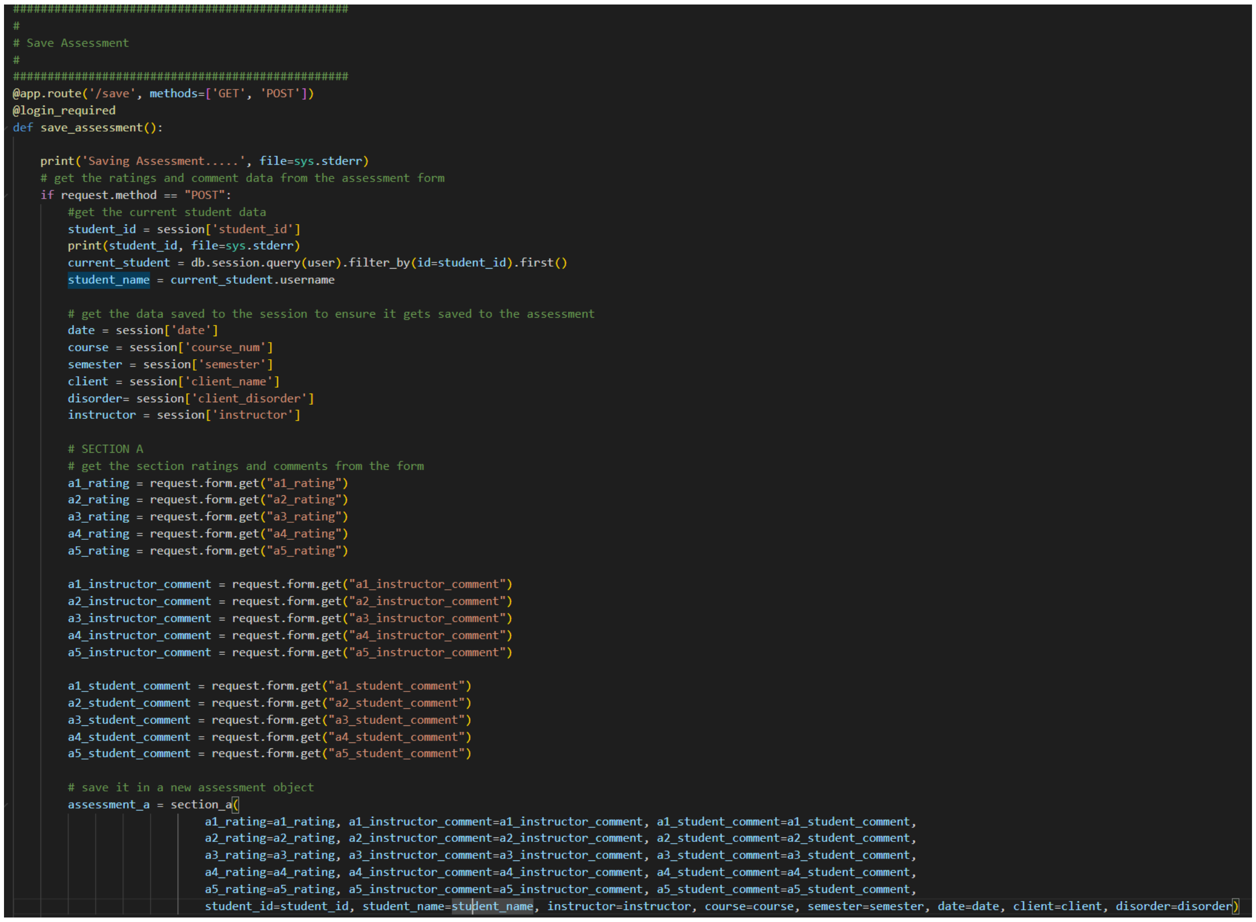
A. Selection of Activities
B. Implemenation of Activities
C. Therapeutic Interventions
D. Communication
E. Collaboration

A. Selection of Activities and Materials

	N/A	Inadequate		Emerging		Developing		Refining		Proficient
1. Intervention activities targeted the objectives identified in the Session Plan.	●	○	○	○	○	○	○	○	○	○
2. Clinician planned appropriate number of activities to engage client for entire session.	●	○	○	○	○	○	○	○	○	○
3. The activities and materials used by the clinician clearly reflected the client's interests.	●	○	○	○	○	○	○	○	○	○
4. Clinician chose activities appropriate to age, culture, functional/developmental level, etc. of the client.	●	○	○	○	○	○	○	○	○	○
5. If indicated, clinician provided appropriate activities for home practice.	●	○	○	○	○	○	○	○	○	○

Figure 7. Performing an assessment

Once an Instructor assigns a rating to each item in the assessment and potentially leaves a comment for a specific item, the assessment can be saved. This is done via the save_assessment function. This function saves all data relating to the student, client and instructor and saves it into the databse. Each section has its own database table but the common data in each section is mirrored in all 5 sections. The ratings, instructor and student comments are unique to each section_x table. This is shown in figure 9, the Entity Relationship diagram for the CFIT Application.



```
#################################################
#
# Save Assessment
#
#################################################
@app.route('/save', methods=['GET', 'POST'])
@login_required
def save_assessment():

    print('Saving Assessment.....', file=sys.stderr)
    # get the ratings and comment data from the assessment form
    if request.method == "POST":
        #get the current student data
        student_id = session['student_id']
        print(student_id, file=sys.stderr)
        current_student = db.session.query(user).filter_by(id=student_id).first()
        student_name = current_student.username

        # get the data saved to the session to ensure it gets saved to the assessment
        date = session['date']
        course = session['course_num']
        semester = session['semester']
        client = session['client_name']
        disorder= session['client_disorder']
        instructor = session['instructor']

        # SECTION A
        # get the section ratings and comments from the form
        a1_rating = request.form.get("a1_rating")
        a2_rating = request.form.get("a2_rating")
        a3_rating = request.form.get("a3_rating")
        a4_rating = request.form.get("a4_rating")
        a5_rating = request.form.get("a5_rating")

        a1_instructor_comment = request.form.get("a1_instructor_comment")
        a2_instructor_comment = request.form.get("a2_instructor_comment")
        a3_instructor_comment = request.form.get("a3_instructor_comment")
        a4_instructor_comment = request.form.get("a4_instructor_comment")
        a5_instructor_comment = request.form.get("a5_instructor_comment")

        a1_student_comment = request.form.get("a1_student_comment")
        a2_student_comment = request.form.get("a2_student_comment")
        a3_student_comment = request.form.get("a3_student_comment")
        a4_student_comment = request.form.get("a4_student_comment")
        a5_student_comment = request.form.get("a5_student_comment")

        # save it in a new assessment object
        assessment_a = section_a(
                            a1_rating=a1_rating, a1_instructor_comment=a1_instructor_comment, a1_student_comment=a1_student_comment,
                            a2_rating=a2_rating, a2_instructor_comment=a2_instructor_comment, a2_student_comment=a2_student_comment,
                            a3_rating=a3_rating, a3_instructor_comment=a3_instructor_comment, a3_student_comment=a3_student_comment,
                            a4_rating=a4_rating, a4_instructor_comment=a4_instructor_comment, a4_student_comment=a4_student_comment,
                            a5_rating=a5_rating, a5_instructor_comment=a5_instructor_comment, a5_student_comment=a5_student_comment,
                            student_id=student_id, student_name=student_name, instructor=instructor, course=course, semester=semester, date=date, client=client, disorder=disorder)
```

Figure 8. Saving the assessment data

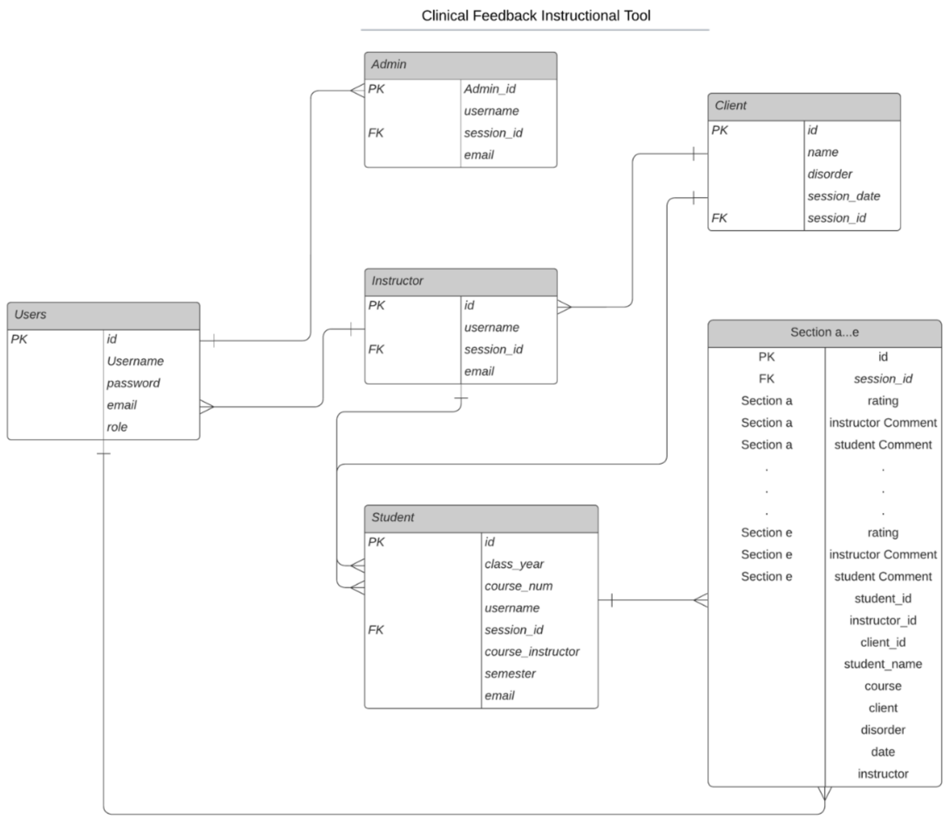


Figure 9. The application Entity Relationship Diagram

Installation and Deployment

The CFIT application is a streamlined and simple application and thus does not require an extensive list of technology or protocols to install and run. The application requires Python v3.x and Flask v2.1.2 including Flask-Bootstrap 3.x and Flask-SQLAlchemy v2.5. A mySQL database server is also required. Our application uses a mySQL database hosted on Google Cloud Platform. We used BeeKeeper studio as a front-end tool to view and edit our SQL database. Some display elements use prebuilt bootstrap CSS and some Fonts are from the Google Font Engine.

Packages needed to run the application:

```
sudo apt-get install python3-pip
sudo apt-get install micro
sudo pip3 install flask
sudo pip3 install flask-wtf flask-sqlalchemy
sudo pip3 install flask-mysql requests python-dotenv
sudo apt-get install zip
sudo pip3 install requests flask-mysql flask-wtf flask-sqlalchemy mysql-
connector-python
```

```
        sudo pip3 install flask_login werkzeug
```

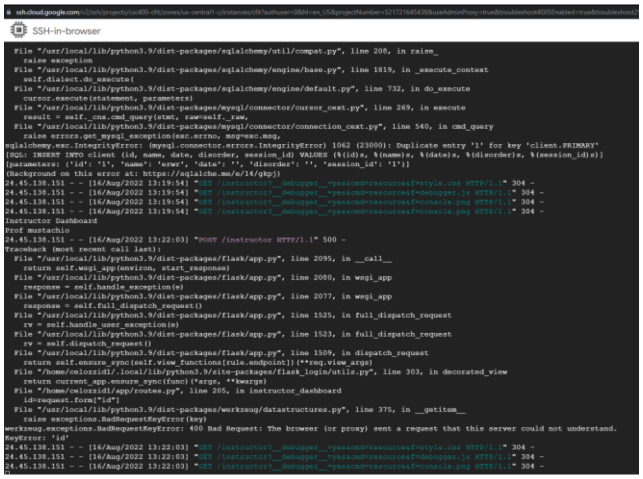
For deployment, a `Requirements.txt` containing all installed Python Packages and the Environment is available to easily mimic the installation and functionality of the application currently in use. The application deployment is currently in process and is dependent on client input.

State of Implementation

The web application is functional and at a point where it can be used by the client. designed has various features that were requested by the client that have been implemented. The first feature is login with the correct user role. This is fully functional. Logging in by your specific email and password will bring you to the correct landing page for your user type. The second feature is the different login home pages for each type of user (Administrator, Client Instructor, Student). As previously stated, this is fully functioning. The correct landing page will be retrieved by the user's email address which is directly associated with the user's role assigned to them when that user is created. The next feature is the ability for admins to add, and remove students, instructors, other admins, and clients. This feature functions properly. If an admin wishes to remove a user of any level, they can easily do so through the dashboard. The ability for admins and instructors to create, view and edit Clinical Feedback Instructional Tool forms functions as well. The instructor can do these actions through their dashboard. Students having the ability to view their assessments and instructor feedback is also functional. The main feature of this web application is the online version of their CFIT form, and this was fully implemented with their design specifications. Lastly the form can also track student ratings by showing their ratings average per evaluation section on the main assessment form page.

Testing and Evaluation

To test the implementation, we ran the application on the Virtual Machine (VM) in Google Cloud Platform (GCP) to verify our code works and to see the result on the browser. Due to the nature of the application, testing was done using verbose print(' ') statements within the code to output necessary information and associated variable data to the console. This helping in gathering information and data in real time while the application was running. We made sure to also add a print description to the data to ensure we know what we were investigating. We also made use of simple HTML to display data on test pages while developing to ensure data was being transmitted correctly and accurately before finalizing the application html pages. The most common issue was correctly retrieving a database value and displaying it in the correct format. Using these methods, we were able to step through and find the root of the issue. Each of the team members would run the application under various circumstances and attempt to emulate user scenarios, i.e. click all the buttons with various data to make sure the application runs in different case scenarios. We would also attempt to think outside the usual user scenario and perform tasks that are out of the ordinary such as navigating away using all links and browser functions, trying to click or overextend test. Using these methods allowed us to test the application in all the circumstances that may arise. For front-end development, we would run the HTML code file on either the VM in GCP or Visual Studio Code and go to inspect the page on the browser that the application is being run in. This allowed us to fix many of the issues that arose in the HTML or CSS styling of the code. While there are some issues that were not able to be resolved through this, we were able to get a closer look at what goes on behind the scenes in the application. By clicking a certain field in the application, we can view its properties and all the styling that affects the specified field. Then we can play with the code using various styles within the web browser, testing how each change would affect the look of the page. Inspecting the elements on a browser was extremely helpful in pinpointing the exact root of the issue in styling that we were trying to resolve, rather than changing styles individually in the code to see what changed in the application. Python and Flask were extremely informative in isolating errors with our code as well as details using the built in Traceback Interpreter.


ssh.cloud.google.com/v2/ssh/projects/csc400-cfit/zones/us-central1-c/instances/cfit?authuser=0&hl=en_US&projectNumber=521721645438&useAdminProxy=true&troubleshoot4005Enabled=true&troubleshoot2
SSH-in-browser

```
  File "/usr/local/lib/python3.9/dist-packages/sqlalchemy/util/compat.py", line 208, in raise_
    raise exception
  File "/usr/local/lib/python3.9/dist-packages/sqlalchemy/engine/base.py", line 1819, in _execute_context
    self.dialect.do_execute(
  File "/usr/local/lib/python3.9/dist-packages/sqlalchemy/engine/default.py", line 732, in do_execute
    cursor.execute(statement, parameters)
  File "/usr/local/lib/python3.9/dist-packages/mysql/connector/cursor_cext.py", line 269, in execute
    result = self._cnx.cmd_query(stmt, raw=self._raw,
  File "/usr/local/lib/python3.9/dist-packages/mysql/connector/connection_cext.py", line 540, in cmd_query
    raise errors.get_mysql_exception(exc.errno, msg=exc.msg,
sqlalchemy.exc.IntegrityError: (mysql.connector.errors.IntegrityError) 1062 (23000): Duplicate entry '1' for key 'client.PRIMARY'
[SQL: INSERT INTO client (id, name, date, disorder, session_id) VALUES (%(id)s, %(name)s, %(date)s, %(disorder)s, %(session_id)s)]
[parameters: {'id': '1', 'name': 'erwr', 'date': '', 'disorder': '', 'session_id': '1'}]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
24.45.138.151 - - [16/Aug/2022 13:19:54] "GET /instructor?__debugger__=yes&cmd=resource&f=style.css HTTP/1.1" 304 -
24.45.138.151 - - [16/Aug/2022 13:19:54] "GET /instructor?__debugger__=yes&cmd=resource&f=debugger.js HTTP/1.1" 304 -
24.45.138.151 - - [16/Aug/2022 13:19:54] "GET /instructor?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 304 -
24.45.138.151 - - [16/Aug/2022 13:19:54] "GET /instructor?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 304 -
Instructor Dashboard
Prof mustachio
24.45.138.151 - - [16/Aug/2022 13:22:03] "POST /instructor HTTP/1.1" 500 -
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/flask/app.py", line 2095, in __call__
    return self.wsgi_app(environ, start_response)
  File "/usr/local/lib/python3.9/dist-packages/flask/app.py", line 2080, in wsgi_app
    response = self.handle_exception(e)
  File "/usr/local/lib/python3.9/dist-packages/flask/app.py", line 2077, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.9/dist-packages/flask/app.py", line 1525, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.9/dist-packages/flask/app.py", line 1523, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.9/dist-packages/flask/app.py", line 1509, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
  File "/home/celozzid1/.local/lib/python3.9/site-packages/flask_login/utils.py", line 303, in decorated_view
    return current_app.ensure_sync(func)(*args, **kwargs)
  File "/home/celozzid1/app/routes.py", line 205, in instructor_dashboard
    id=request.form["id"]
  File "/usr/local/lib/python3.9/dist-packages/werkzeug/datastructures.py", line 375, in __getitem__
    raise exceptions.BadRequestKeyError(key)
werkzeug.exceptions.BadRequestKeyError: 400 Bad Request: The browser (or proxy) sent a request that this server could not understand.
KeyError: 'id'
24.45.138.151 - - [16/Aug/2022 13:22:03] "GET /instructor?__debugger__=yes&cmd=resource&f=style.css HTTP/1.1" 304 -
24.45.138.151 - - [16/Aug/2022 13:22:03] "GET /instructor?__debugger__=yes&cmd=resource&f=debugger.js HTTP/1.1" 304 -
24.45.138.151 - - [16/Aug/2022 13:22:03] "GET /instructor?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 304 -
```

Figure 10. Python traceback interpreted notification of a KeyError

The project, as a result, met most of the objectives that we set out. Although some issues were not able to be resolved through the various methods of debugging, the core features of the application are there and fully functional as a full-stack web application. Users can login to the system, then the system determines the user role based on the role stored in the database when creating a user, and then redirects to the designated page for the user. An admin can access the admin and instructor dashboard, to create and deleted users. An instructor can access the instructor dashboard, and a student can assess the student dashboard. Instructors can create assessments, apply a rating to each question, observe the students average rating and comment on individual items. Students can view assessments created by their assigned instructor and view and make comments on the form. This is the core functionality and main purpose of the application.

The goal was to create an application that was streamlined and simple to use without being overburdened, cluttered and bloated by overcomplicated features. We achieved that and the client is both delighted and satisfied with the application. Even though there are certain parts of the application that can be improved if we had more time to work on the project.

Lessons learned and reflection

Like any development team, we encountered many challenges during this project. The biggest issue we faces is that each of us in this team came from different technical and coding backgrounds and experience levels. We have different strengths and weaknesses and levels of experience. Many of the methods used in this project required extensive research and much trial and error. Tuning the HTML and

CSS to the application needs took a substantial amount of time. The entity relationships and tables in our database have been modified many times as during development as we recognized opportunities for simplicity and efficiency. For example, one of the team members (David) was very familiar with working with the web framework *Flask* while the other two (Derek and Jeremy) were not. One team member focused a lot on the front end of the web application while the other went back and forth working on both. Luckily our strengths and weaknesses complimented each other's and did not create issues during development.

This software development project did not seem challenging at first for the team. As development and implementation was in full swing, certain things that did not seem to pose a challenge at first, finally did. A lot of issues were ones that we thought would be a simple fix. We turned out to be wrong and most of these seemingly simple issues turned to be major time sinks. An example of this would be simply lining up the radio buttons on the form. We thought it would not be too hard to do but it took us a while to align them better. These time sinks overtime added up and took up a lot of the time that could have been used doing more important things such as working on the different main functionalities of the web application.

Another challenge we are faced since the project inception is planning and developing based on the future expansion and planned features of this project. As this project was planned to be developed by multiple teams over multiple semesters, designing based on future expansion and features was challenging, and with that, keeping our focus on the current feature implementation. One goal we had was to not burden future development teams with complete redesign or overhaul of the code in order to develop a new or planned feature.

Communication was one of the strong suits of the group. Group communication is crucial when it comes to software development. Everyone needs to be in sync or else it can get messy. Before the project began, we had already made a *Discord* server to communicate in. The duration of the project timeline we were constantly communicating ideas, issues, and solutions through this server. We would hop in a call and screen share to show code that we have been working on and give each other feedback. There was no point during this project where the server was inactive for long periods of time. The continuous stream of communication was a key factor in achieving the goals and objectives that were set.

Adaptation was another strength the group had. During every client meeting, we will ask the clients for feedback on how the web application looked, show them the core functionalities and ask them if the way we interpreted their original request was implemented the way they wanted to. As expected, they did constantly request changes throughout these meetings. Some of the requests were simple to fix and others were not. Some requests also had to be tempered because of the amount of work that needed to be done to implement it or the lack of knowledge in doing so. With this in mind, the group adapted to the given constraints and tried our best to give the client their needs for this web application. An example of an adaptation we made from the client's request was to autofill the information section on the top of the form. At first, we had input text boxes for the client instructor to fill in the information needed. The client gave us feedback saying it would be better to auto populate the information when creating a form. Our group adapted; did the necessary coding to retrieve that information from the database and got that functionality to work.

If we were able to do this project again there are a few things we would do differently. One thing we would do differently is to build it more modal from the beginning. The main component of this web application is the form. This form has several sections with various questions, radio buttons, text boxes etc. The way we have things set up right now, it can be very difficult and confusing traversing the code. Making slight code changes can be tedious and time consuming. Having it broken into parts can help alleviate time sinks, confusion, and cleaner/more efficient code.

Another thing we would have done differently was to use flask forms instead of HTML forms. Flask forms come with many advantages such as built-in form validation. Not only that but it would have made the project easier. The CFIT form has different sections where the layout and functionality are all the same. There would have been less code to handle. More JavaScript and Bootstrap may have assisted us with some of the functionality of the web application as well.

Version 2

Although the application is functional, the list of planned features and features the client requested is extensive. If the team did have an extra month or two to continue working on the CFIT application, we would implement the rest of the features the client requested. The highest priority for the client is the ability to track a student's progress over time. Student progress is currently only being tracked by viewing past assessments and manually observing the average rating per assessment section. The client would also like to display an overall rating metric column in the student list comprised of the aggregated total of all items in all sections of an assessment. The next step would be to implement a visual representation such as a graph or chart which shows the students' progression over time across semesters and years would supplement the tracking.

Another feature we would like to implement if we had the extra time is to be able to implement data sorting in the user tables. There would be multiple methods to sort based on user needs. The client also requested a search function allowing you to search through the table and get the result based on search criteria. This would likely be by searching for username, id, or search by semester depending on the page the user is on. This would help with organizing the data when the user tables inevitably get too large to organize as more users implement this application over future semesters.

The ability to add custom sections and associated assessment items is also on the future wish list as well as the ability to edit users. Currently, and edit consists of deletion and addition. Notifications via messages and visual indicators is also a planned feature. This pertains to modifying the color of a comment button on the assessment form indicating an instructor has left a comment for a student and highlighting the 'View Assessment' button when there is an assessment for that student.

Overall, we are extremely proud of the application and we have learned much throughout this course, the project and development cycle. We hope that this application eventually develops into the application the client has envisioned and is used by the SCSU Communication Disorders Department for many years to come.