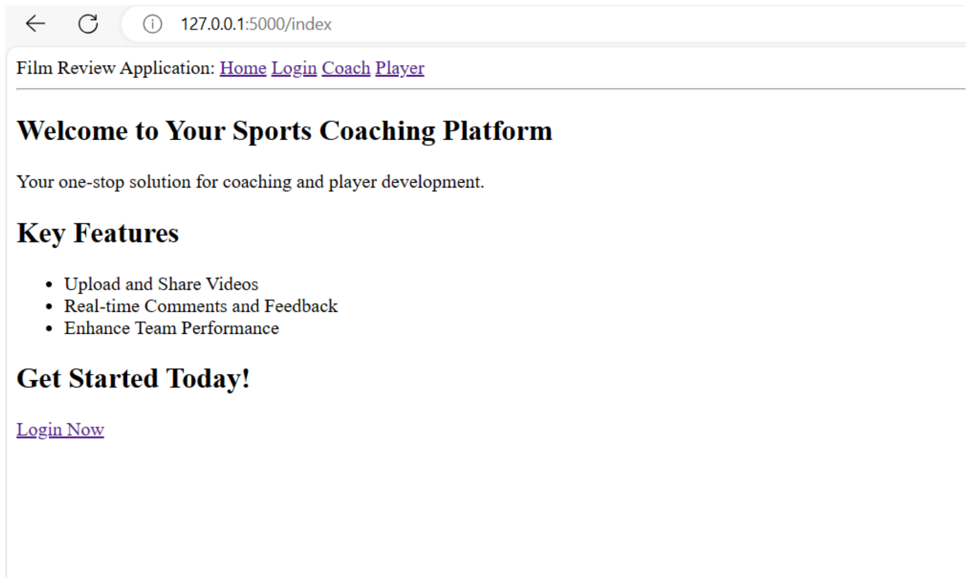
Group 3: Film Review Application
Milestone 2: Software Engineer
Connor Levinson

1. Below are screenshots of the API with stub responses(A-D), followed by screenshots of the python and flask code(E-I). After the screenshots is an explanation of the images shown.

A.


127.0.0.1:5000/index

Film Review Application: Home Login Coach Player

Welcome to Your Sports Coaching Platform

Your one-stop solution for coaching and player development.

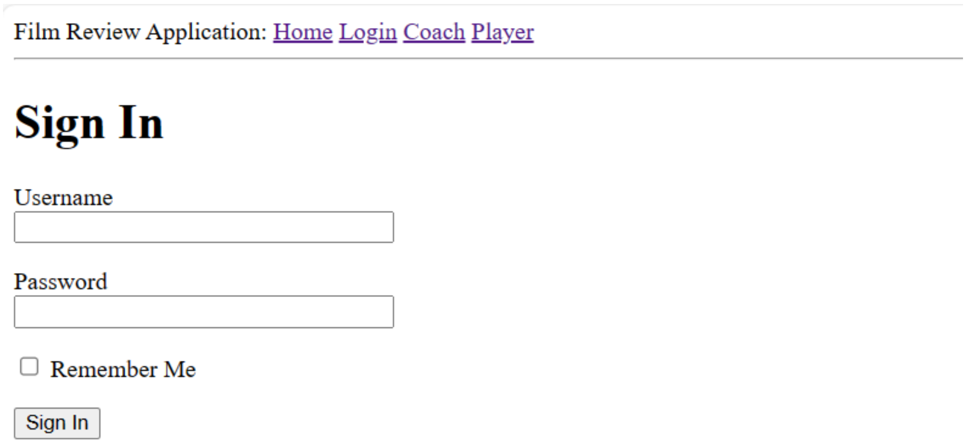
Key Features

- Upload and Share Videos
- Real-time Comments and Feedback
- Enhance Team Performance

Get Started Today!

Login Now

B.


Film Review Application: Home Login Coach Player

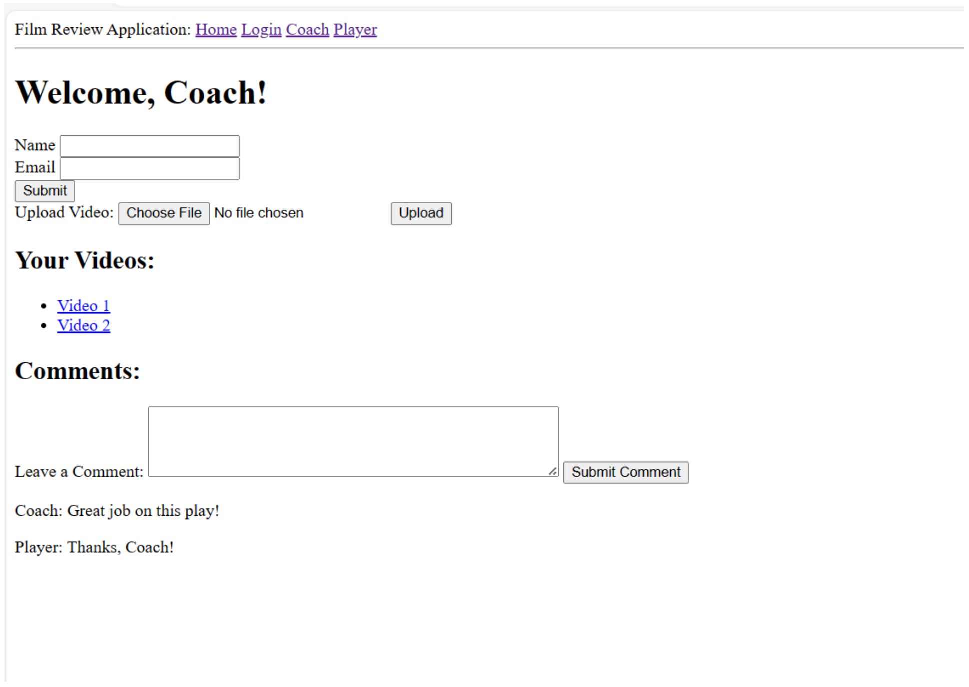
Sign In

Username

Password

Remember Me

C.



Film Review Application: Home Login Coach Player

Welcome, Coach!

Name

Email

Upload Video: No file chosen

Your Videos:

- Video 1
- Video 2

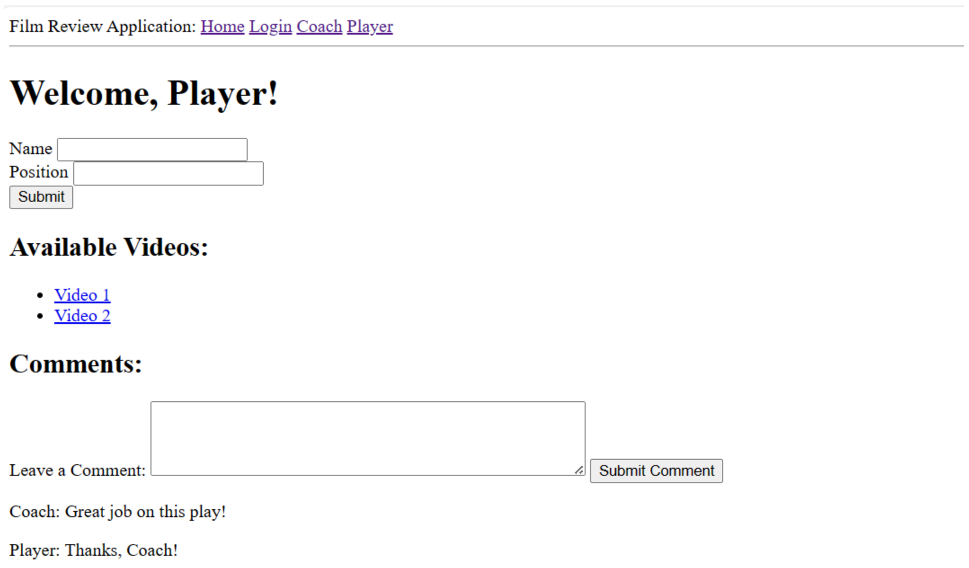
Comments:

Leave a Comment:

Coach: Great job on this play!

Player: Thanks, Coach!

D.



Film Review Application: Home Login Coach Player

Welcome, Player!

Name

Position

Available Videos:

- Video 1
- Video 2

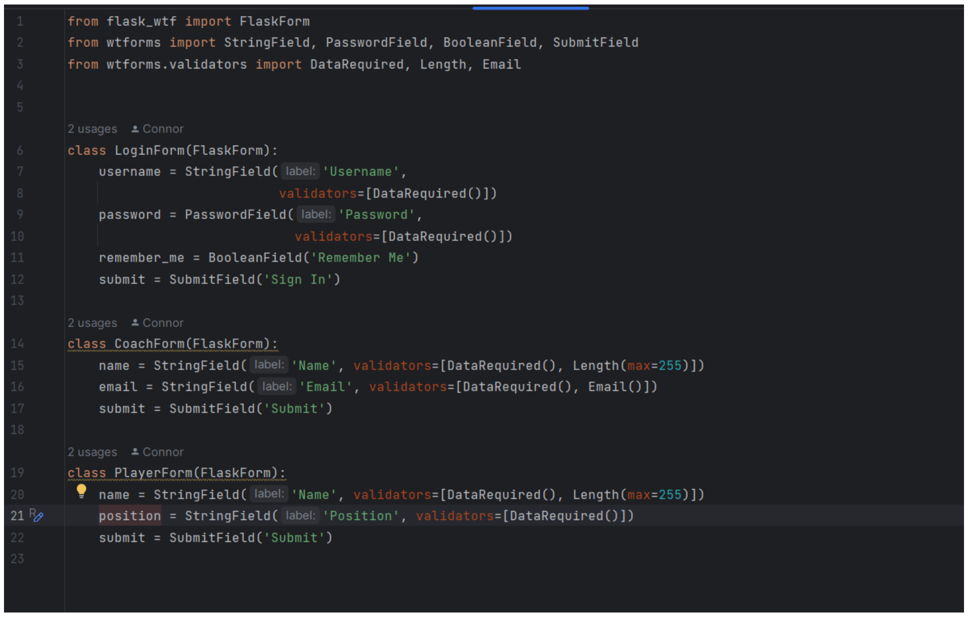
Comments:

Leave a Comment:

Coach: Great job on this play!

Player: Thanks, Coach!

E.



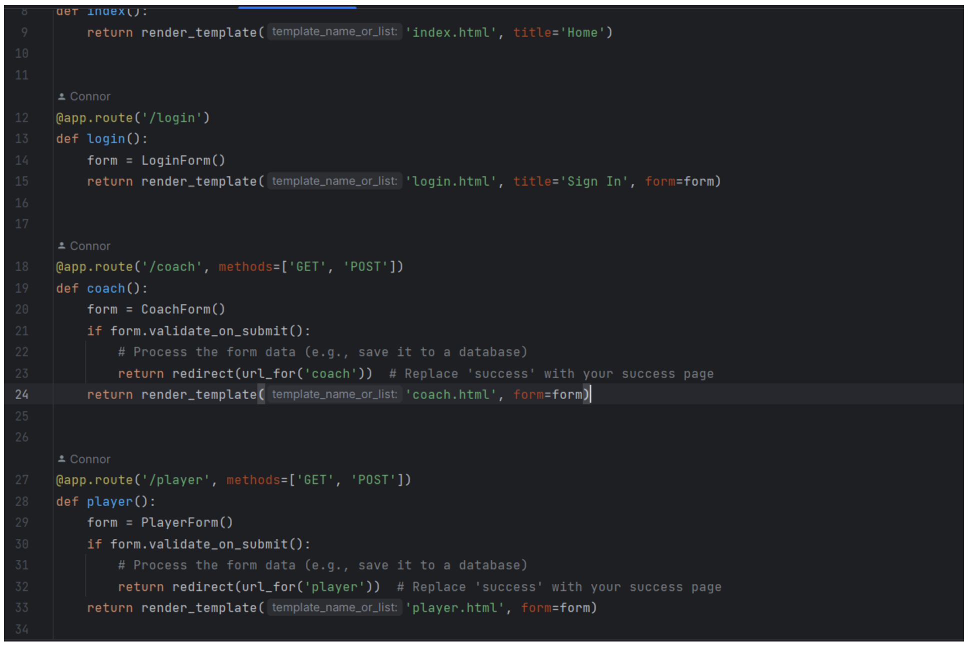
```python
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired, Length, Email


class LoginForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired()])
    password = PasswordField('Password',
                             validators=[DataRequired()])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Sign In')

class CoachForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired(), Length(max=255)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    submit = SubmitField('Submit')

class PlayerForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired(), Length(max=255)])
    position = StringField('Position', validators=[DataRequired()])
    submit = SubmitField('Submit')
```

F.



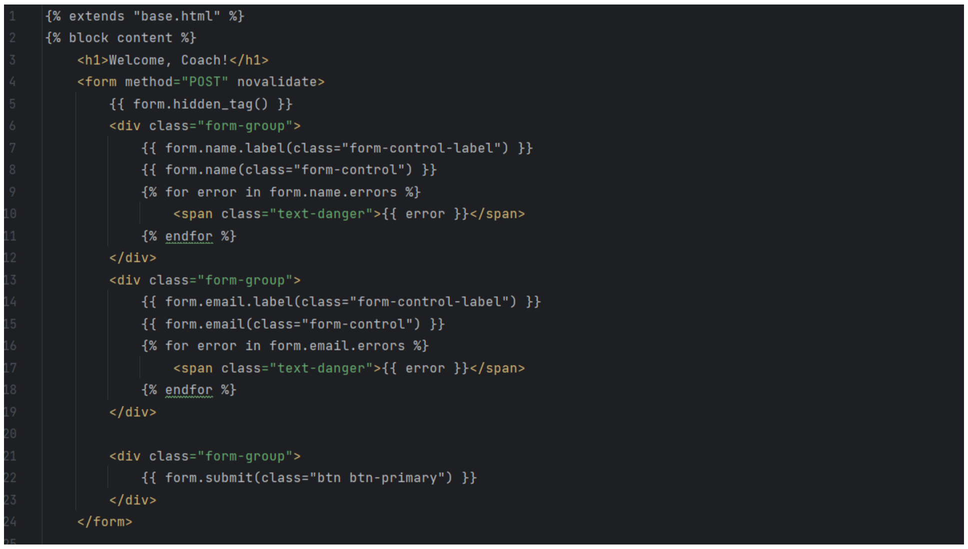
```python
def index():
    return render_template('index.html', title='Home')


@app.route('/login')
def login():
    form = LoginForm()
    return render_template('login.html', title='Sign In', form=form)


@app.route('/coach', methods=['GET', 'POST'])
def coach():
    form = CoachForm()
    if form.validate_on_submit():
        # Process the form data (e.g., save it to a database)
        return redirect(url_for('coach'))  # Replace 'success' with your success page
    return render_template('coach.html', form=form)


@app.route('/player', methods=['GET', 'POST'])
def player():
    form = PlayerForm()
    if form.validate_on_submit():
        # Process the form data (e.g., save it to a database)
        return redirect(url_for('player'))  # Replace 'success' with your success page
    return render_template('player.html', form=form)
```

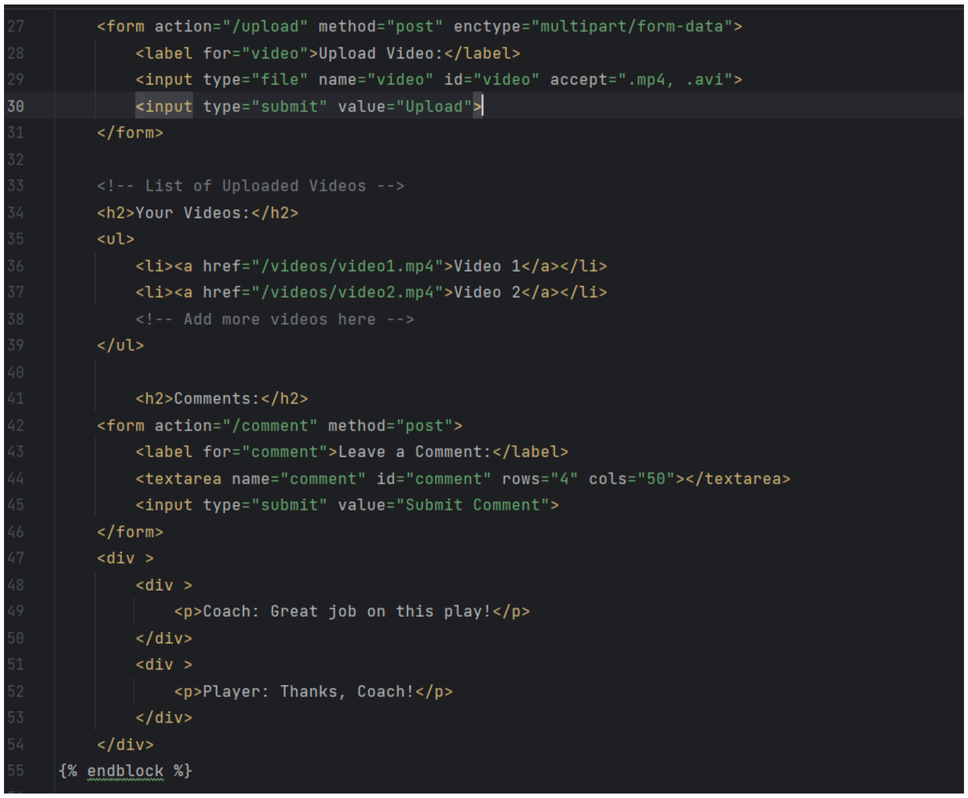
G.



```
{% extends "base.html" %}
{% block content %}
    <h1>Welcome, Coach!</h1>
    <form method="POST" novalidate>
        {{ form.hidden_tag() }}
        <div class="form-group">
            {{ form.name.label(class="form-control-label") }}
            {{ form.name(class="form-control") }}
            {% for error in form.name.errors %}
                <span class="text-danger">{{ error }}</span>
            {% endfor %}
        </div>
        <div class="form-group">
            {{ form.email.label(class="form-control-label") }}
            {{ form.email(class="form-control") }}
            {% for error in form.email.errors %}
                <span class="text-danger">{{ error }}</span>
            {% endfor %}
        </div>

        <div class="form-group">
            {{ form.submit(class="btn btn-primary") }}
        </div>
    </form>
```

H.

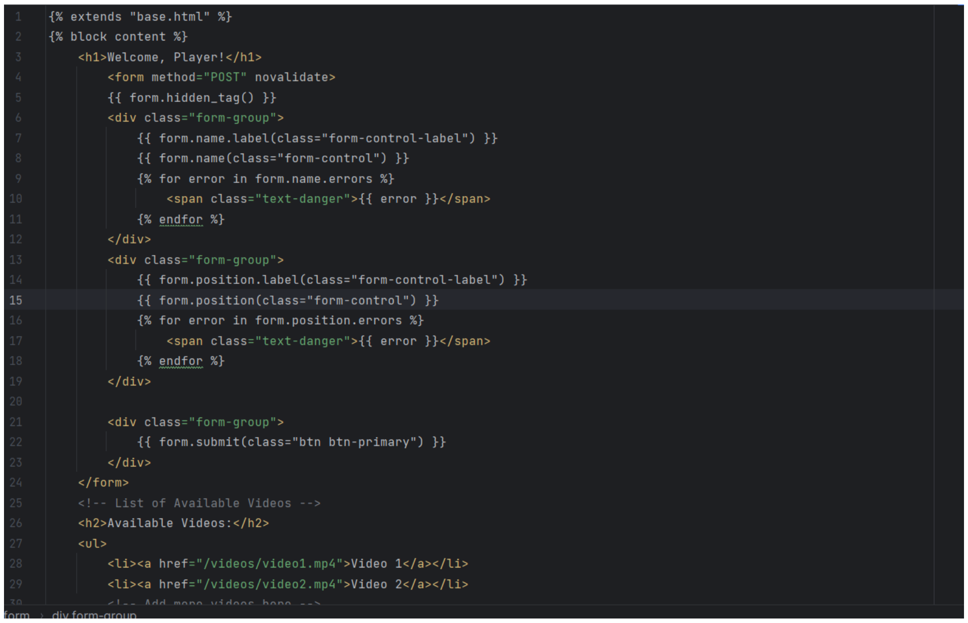


```
    <form action="/upload" method="post" enctype="multipart/form-data">
        <label for="video">Upload Video:</label>
        <input type="file" name="video" id="video" accept=".mp4, .avi">
        <input type="submit" value="Upload">
    </form>

    <!-- List of Uploaded Videos -->
    <h2>Your Videos:</h2>
    <ul>
        <li><a href="/videos/video1.mp4">Video 1</a></li>
        <li><a href="/videos/video2.mp4">Video 2</a></li>
        <!-- Add more videos here -->
    </ul>

        <h2>Comments:</h2>
    <form action="/comment" method="post">
        <label for="comment">Leave a Comment:</label>
        <textarea name="comment" id="comment" rows="4" cols="50"></textarea>
        <input type="submit" value="Submit Comment">
    </form>
    <div >
        <div >
            <p>Coach: Great job on this play!</p>
        </div>
        <div >
            <p>Player: Thanks, Coach!</p>
        </div>
    </div>
{% endblock %}
```

I.



```
{% extends "base.html" %}
{% block content %}
    <h1>Welcome, Player!</h1>
        <form method="POST" novalidate>
        {{ form.hidden_tag() }}
        <div class="form-group">
            {{ form.name.label(class="form-control-label") }}
            {{ form.name(class="form-control") }}
            {% for error in form.name.errors %}
                <span class="text-danger">{{ error }}</span>
            {% endfor %}
        </div>
        <div class="form-group">
            {{ form.position.label(class="form-control-label") }}
            {{ form.position(class="form-control") }}
            {% for error in form.position.errors %}
                <span class="text-danger">{{ error }}</span>
            {% endfor %}
        </div>

        <div class="form-group">
            {{ form.submit(class="btn btn-primary") }}
        </div>
    </form>
    <!-- List of Available Videos -->
    <h2>Available Videos:</h2>
    <ul>
        <li><a href="/videos/video1.mp4">Video 1</a></li>
        <li><a href="/videos/video2.mp4">Video 2</a></li>
        <!-- Add more videos here -->
```

Explanation:

The above pictures are the current state of the film review application. The current state of the program is not fully intuitive or exactly what we intend for the final product, however it currently provides a nice base structure that can be easily exchanged once our final ideas are polished.

- Picture A is the home screen that has navigation to each view and allows the user to get to the login, while seeing our basic introduction. This would be seen by any user that opens our web application.
- Picture B is the view of the login page that allows any user to log into an account. The WTF control and validators are used to accept a user's username and password to log into their account. These make it helpful to distinguish between users and ensure that the user has an account. With the current implementation the user can log in, but it does not decipher between a coach and user role yet, it allows them to see both views. This is something that will be fixed in a future sprint.
- Picture C and D are the player and coach view to watch and upload film, while also being able to comment and view other comments. These currently use control and validators for the coach's name and email and the players name and position. These specific validators will not be in the final implementation, however we are currently unsure of what validators and controls to put for each view, so these are placeholders. These also hold stub responses for comments and videos that have been uploaded and posted.
- Picture E is our forms.py file that holds all the validators for the program, including the login validators, coach validators, and player validators. It imports the datarequired, email, and length validators that make it easier to validate that the control has data, that the email is valid, and that the names are not too long.
- Picture F is the routes.py that shows how each different page is loaded up and the information that is passed to the page to make it work and adapt to different users.
- Pictures G, H, and I are the html files for the coach and player views, this is where the controls and forms are located to take in the user's names, emails, and positions to be

validated. This is also where the stub videos and comments are produced to be place holders for future implementation to include live comments and videos.

Coding Standards

1. **Consistent File Pathing**

All files should follow a similar structure to what has been established. The config.py and filmReview.py are the only files explicitly in the Group3Project folder. All other files are under the app folder. Any HTML template should be in the templates folder. Each file name should be concise while declaring the purpose of the file to be understood by any developer. File pathing contributes to the software engineering concept of reliability because any team developer will know exactly where each file is located and where each new file should be located.

2. **Descriptive Variable and Function Names**

Any variable and function that is declared must have a name that can be easily read and understood for the use of the variable and/or function that is being created. All variables and functions should use camel case when being declared. This should be consistent in all files. Descriptive variable and function names contribute to clarity of the code, allowing code to be found when trying to understand why errors are occurring.

3. **Comprehensive Documentation**

Code files should include comments that explain confusing or unclear code. If code is basic and self-explanatory, comments are not needed. When code gets confusing and found in large chunks, comments must be used to explain the content. Documentation should be concise and less verbose, only including what is necessary to understand the code. Documentation is a software engineering concept of its own that contributes to all software engineering goals, because with adequate documentation, code can be easily understood and adapted.

4. **Version Control**

For any developer that is adjusting the program, they should create their own branch from the GitHub repository(Soorxa/Group3Project: Software engineering project creating a basketball hudl app. (github.com)). They will commit and push into their own branch. The developer should request a review and pull request from at least one other team member in order to get their code merged to the main branch. Version Control helps with the software engineering concepts because it allows easy error handling. If something goes wrong with the code, they can go back to the most recent working version of the program.

Why should stakeholders care?

Stakeholders should take great interest in the coding standards because the standards are the backbone for the project itself. For the most part, stakeholders will not fully understand all of the technical aspects of the project, but they can understand the standards that the technical side must follow. If the standards themselves are complete and strictly followed, the stakeholders can have faith that the code itself will be held to a high standard and not sloppy or lackluster.