Group 3: Jahlil Owens, Trishelle Leal, and Landon Strappazon

Written By Jahlil Owens, Test Engineer

Dr. Ruth Lamprecht

CMSCI 349 A

November 20, 2024

# Nutrition and Fitness tracking with Software Engineer

Within our Nutrition and fitness tracking project, both Test Engineers and Software Engineers have to work together to ensure a well-structured application that allows the purpose of users to track their nutrition and fitness throughout the week and reach their fitness goals. The application will allow the user to log their meals, track nutritional intake, and monitor their progress toward their dietary goals. The Software Engineer will help design, develop, build, and maintain the software of a project by following engineering principles along with software principles and goals. They will look for how to benefit users by providing structured code that is reliable, efficient, and stable in hopes of accommodating the user stories, requirements of the project, and stakeholders. After developing the major components of the nutrition tracking app we now develop the code for the API and how each user story works together. The first method of code for the API is the Login process. The login method is the entry point that allows users to access the project by adapting the login form method which takes the email and password variables and assigns them to the datarequired validator to make these two variables a requirement. If the data in the form is valid then the email is stored into the user's sessions and

then it takes the user to the dash otherwise it will prompt an error message of the email or password being invalid. It then calls the login API method to authorize the user based on a secure database.

```python
# running the process of each task including login, interface, logging meal and progress
# this validates the process of the user logging into their account
@app.route( rule: '/', methods=['GET', 'POST'])
def login():
    #get the form for the log in process
    form = LoginForm()
    if form.validate_on_submit():
        try:
            #validate the email that was provided
            validation = validate_email(form.email.data)
            email = validation.normalized
        # for now, we redirect to the dashboard but in the real app we would call the api login endpoint
            return redirect(url_for('dashboard'))
        #checks for any email validation errors that may occur
        except EmailNotValidError as check:
            print(str(check))
            print('Invalid email')
    return render_template( template_name_or_list: 'login.html', form=form)
```

```python
#Below is our API Routes
#login endpoint for the API
/api/login
@app.route( rule: '/api/login', methods=['POST'])
def login_api():
    email = request.form.get('email')
    password = request.form.get('password')
    #check to see if the email and password are provided
    if not email or not password:
        #return an error message indicating that this is a bad request based on the information provided
        return {'status': 'error', 'message': 'Email and password are required'}, 400
    #if the email and password are valid then we return a successful message.
    if email == 'test@example.com' and password == 'password':
        return {'status': 'success', 'user_id': 1}, 200
    else:
        #otherwise return an unauthorized message that the information that was inputted is invalid
        return {'status': 'error', 'message': 'Invalid credentials'}, 401
```

After the user is logged they now have access to the different user stories starting with logging a meal. The method log_meal() allows the user to input their food intake for the day or week depending on how much they want to input. These inputs for the meals include the food name, quantity of food item, and the type of meal. The name is a variable to give to each food

that is inputted in a string format. The quantity is how many of that food item is being inputted in an integer format using the IntegerField. The meal type variable is a SelectField type variable that provides a list of different options to help distinguish the different types of food items that go into the nutrition tracking.

```
# This assigns the variables of the food names, their quantities and the type of meal they coordinate
class MealLogForm(FlaskForm):  1 usage
    food_name = StringField( label: 'Food Name', validators=[DataRequired()])
    quantity = IntegerField( label: 'Quantity', validators=[DataRequired(), NumberRange(min=1)])
    meal_type = SelectField( label: 'Meal Type', choices=[
        ('breakfast', 'Breakfast'),
        ('lunch', 'Lunch'),
        ('dinner', 'Dinner'),
        ('snack', 'Snack')
    ])
    #we submit the log mean of the characteristics towards our options
    submit = SubmitField('Log Meal')
```

For our API stubs there's the login_api() and log_meal_api(). The login API is the endpoint that does the process of the user's authentication the purpose is to verify that the email and password in the request return a successful repose that is the user's ID if the input that was provided follows the rules of the credentials. If the input doesn't match the credentials of the email and password then we return an error message that says the credentials are invalid. In this section, there is hard code to help represent the process but in the real development of the application, we would use an actual database that may also utilize an external authentication

service to ensure that the credentials are valid or invalid.

```python
#Below is our API Routes
#login endpoint for the API
/api/login
@app.route( rule: '/api/login', methods=['POST'])
def login_api():
    email = request.form.get('email')
    password = request.form.get('password')
    #check to see if the email and password are provided
    if not email or not password:
        #return an error message indicating that this is a bad request based on the information provided
        return {'status': 'error', 'message': 'Email and password are required'}, 400
    #if the email and password are valid then we return a successful message.
    if email == 'test@example.com' and password == 'password':
        return {'status': 'success', 'user_id': 1}, 200
    else:
        #otherwise return an unauthorized message that the information that was inputted is invalid
        return {'status': 'error', 'message': 'Invalid credentials'}, 401
```

Afterward, there's the log_meal_api which is the endpoint for allowing the user to log their meal. It takes the three variables from the log meal method and returns a successful message if the three inputs are valid. This also uses hard code to represent the process but in the real development process, it would interact with the databse and store the data of the new meal log and can potentially include the calories, macros, and more if the information is provided.

```python
@app.route( rule: '/api/log_meal', methods=['POST'])
def log_meal_api():
    user = session.get('user')
    #error check for invalid user
    if not user:
        return {'status': 'error', 'message': 'User does not exist'}, 401
    # Simulate logging a meal
    food_name = request.form.get('food_name')
    quantity = request.form.get('quantity')
    meal_type = request.form.get('meal_type')
    #error check for no food name, quantity and meal type
    if not (food_name and quantity and meal_type):
        return {'status': 'error', 'message': 'Food name, quantity and meal type are required'}, 400
    if user not in user_meals:
        user_meals[user] = []
    user_meals[user].append({'food_name': food_name, 'quantity': quantity, 'meal_type': meal_type})
    return {'status': 'success', 'message': f'Logged {quantity} {food_name} for {meal_type}'}
```

The last section is the goal which allows the user to input their target weight within a certain amount of weeks. The process checks to see if a user is logged into the session otherwise they are prompted with a message to log in. Afterward, it starts the process of validating a user's

input for their target weight and weeks. Once the information is inputted it stores these goals into the system allowing the user to see their goals in the progress page. The API for the goals checks for the validation of a user and to establish the connection to the goals being processed into the system.

```python
def goals():
    if 'user' not in session:
        flash( message: 'Please log in to set your goals.', category: 'warning')
        return redirect(url_for('login'))

    form = GoalForm()
    # allow the user to submit their new goals
    if form.validate_on_submit():
        user = session['user']
        target_weight = form.target_weight.data
        weeks = form.weeks.data
        # Stores the goals into the system
        user_goals[user] = {'target_weight': target_weight, 'weeks': weeks}
        flash( message: f"Goal set! Target Weight: {target_weight} lbs in {weeks} weeks.", category: "success")
        return redirect(url_for('dashboard'))

    return render_template( template_name_or_list: 'goals.html', form=form)

# API to retrieve user's goals
/api/goals
@app.route( rule: '/api/goals', methods=['GET'])
def get_goals_api():
    user = session.get('user')
    if not user:
        return {'status': 'error', 'message': 'User not logged in'}, 401

    goals = user_goals.get(user, None)
    if goals:
        return {'status': 'success', 'goals': goals}, 200
    else:
        return {'status': 'error', 'message': 'No goals set'}, 404
```

The user roles in our app are individuals who want to track their nutrition and dietary goals. This app can be used by those who want to lose weight, gain muscle, stay healthy, research, and more. It is also for administrators who want to have access to certain sections of the application that allow them to manage certain aspects.

The application uses different types of WTF controls and validators to make the data integrity stable such as StringField, PasswordField, IntegerField, SelectField, DataRequired,

Email, and NumberRange. StringField allows the input to be a text input which involves strings such as names, food names, information, and more. PasswordField accepts the same format as string but it will mask the input from the user to protect the information for security. IntegerFiled takes integer values of only whole numbers from the user. This allows the user to input the quantity of their meal. SelectField is a list of options that allows the user to choose when logging a meal. The DataRequired checks to see if the field has input otherwise we prompt an error message. The Email validator verifies if the input is a valid address that matches its email format. NumberRange allows us to specify the minimum or maximum value for the input from the user and in this class we emphasize that the value must be a minimum of 1. These controls and validations help keep the input mechanics valid for multiple data types while also ensuring that the information that is provided is within the correct format and fits the requirements of the project.

Coding standards and software engineering principles are important during the development process. Some aspects of this project are the naming conventions which are consistent names for variables and methods such as login, mealLog, dashboard, etc. This makes it easier to read the code structure and maintain the project. Throughout the code there is also error handling to ensure that the method works properly such as the try and except block in the login method to ensure that input information is valid. The code is formatted in a readable structure that involves having multiple comments for the documentation as a clear and concise explanation. We also have consistent spacing to help keep a professional layout. We use the passwordfield to help with the security stage of the application by ensuring that once the password is inputted correctly the password is masked so it's harder to access from others. During the development, we used multiple tests such as unit, integration, and more to ensure the

project worked properly. All of these tie to the principles of software engineering by establishing readability, maintainability, efficiency, reliability, and security. If the application does follow the majority of these principles then the application would not be a high-quality project that would help the stakeholders and users. Stakeholders should ensure these standards and principles are implemented because they help create a more safer and successful application for users when using the application for tracking their nutrients and dietary goals.

In conclusion, the code showcases the foundation of our nutrition tracking app with WTF forms and validations that help create stub API responses for tracking a user's nutrition. It follows the basic guidelines of software engineering principles to ensure that the application will be useful for those who want to look for a nutrition-tracking app that enhances the user's experience. By adhering to these standards with error handling, security measures, and test cases the application will be a reliable and secure application for those with a mindset to achieve their nutritional goals.