Dannielle Lewis – SWDV-691
Service Layer Design

WODBucket Service Layer Design

The service layer will perform the logic and validations of my application. This layer will ensure that the user has been verified with given credentials and that the requested data or queries exist for the user. If not, an error notification will alert and the application will roll back to the previous point if necessary. In reference to WODBucket, the user will first have to register to use the service. The service layer will validate if the user has previously registered with the email address. This layer will perform the CRUD actions of the database using either Flask or Django, request, redirect, render_template and HTTP Get and HTTP Post requests. If the user wants to display all previous workouts or add an exercise, this layer will perform this query and send the results to the view. RESTful principles will be used when designing this layer with the use of the HTTP verbs as represented in the code snippet below.

```
def username():
    username=request.form['username']
    if username == ' ':
        error="That's not a valid username."
        return redirect("/?error=" + error)
    return render_template('home.html', username=username)

def index():
    # if we have an error, make a <p> to display it
    error = request.args.get("error")

    # combine all the pieces to build the content of our response

    # build the response string

    return render_template('edit.html', error = error)
```

The service layer will include validation of credentials and sensitive information during login and registration such as username and password that will be hashed and salted when posting to the database during registration or when verifying passwords during login. For example, on the login page if the username entered does not exist in the model or database, a flash error will display letting the user know that the username or password is not correct. This will impact the service layer with added code and possible redirecting similar to the code snippet below.

```
def is_email(string):
    atsign_index = string.find('@')
    atsign_present = atsign_index >= 0
    if not atsign_present:
        return False
    else:
        domain_dot_index = string.find('.', atsign_index)
        domain_dot_present = domain_dot_index >= 0
        return domain_dot_present
```

Also, the “sessions” module will be used to handle logout or sign-out actions. When a user is logged in, that will represent an active session. When the user decides to logout, the session will end and redirect to the home page as represented below.

```
def logout():
    session.pop('logged_in', None)
    del session['user']
    return redirect("/home")
```

In addition, if a user gets to the data entry page and has not logged in, they will be directed to the home page with an option to login or register. This will impact my service layer with added code and possible redirecting.

There will be a number of endpoints required for the MVP:

https://www.wodbucket.com/login is a HTTP GET request as the login page that will display upon opening the app (if login has not been established previously). The purpose of this endpoint is to validate that the user is registered with the service and has a primary key in the database in order to connect and retrieve data. This GET request finds by the ID assigned to the user.

https://www.wodbucket.com/register is the registration page that requests to POST a new user to the database. This is required for all new users in order to establish a primary key for use of data in the model. This will bind a user and the id to exercises that the user creates by associating the primary key to the exercise id. This POST request adds a new user to the site.

https://www.wodbucket.com/user1/previous_session is where the user will have the previous session displayed upon successful login. This is the MVP of the project that provides the minimum product needed to accomplish the goals of the application. Here is where the user can either update the data to a current status of information or keep the data the same. This will update the database through the service layer. This GET requests queries the previous post by ID or by date and time.

http://www.wodbucket.com/user1/all_exercises is the RESTful endpoint that responds to the GET request from the interface with all of the exercises that ‘user1’ has added to the site and displays them. This GET request queries all exercises by user ID and exercise ID.

https://www.wodbucket.com/Error_404 is the result if any of the data is not validated (password is incorrect) or is in the wrong format (email doesn’t contain an “@” symbol).

As a result of the design feedback from peers, there are no additional endpoints required for the MVP. It has been decided that the authorization package will be handled by native code within the existing service layer, the error page will display a message on the current page and the sign-out will be handled by the sessions module which will redirect to the homepage as discussed earlier in this document.

The stretch features of generating WODS and connecting with other apps such as MyFitnessPal and MapMyRun will require more engagement of the service layer and there will be additional endpoints. This will be done by having users established as free version users or Pro users when they have logged in. The Pro users will have additional fields in the database which will represent the Pro options of

generating WODS and using API to pull data from other apps. Generating WODS will require a modification of the database schema and possibly an additional database collection that will include workout data available to respond to a pull request for a WOD.

Figure 1 below shows an example of a GET request for all of the workouts that “user7” has POSTED previously.

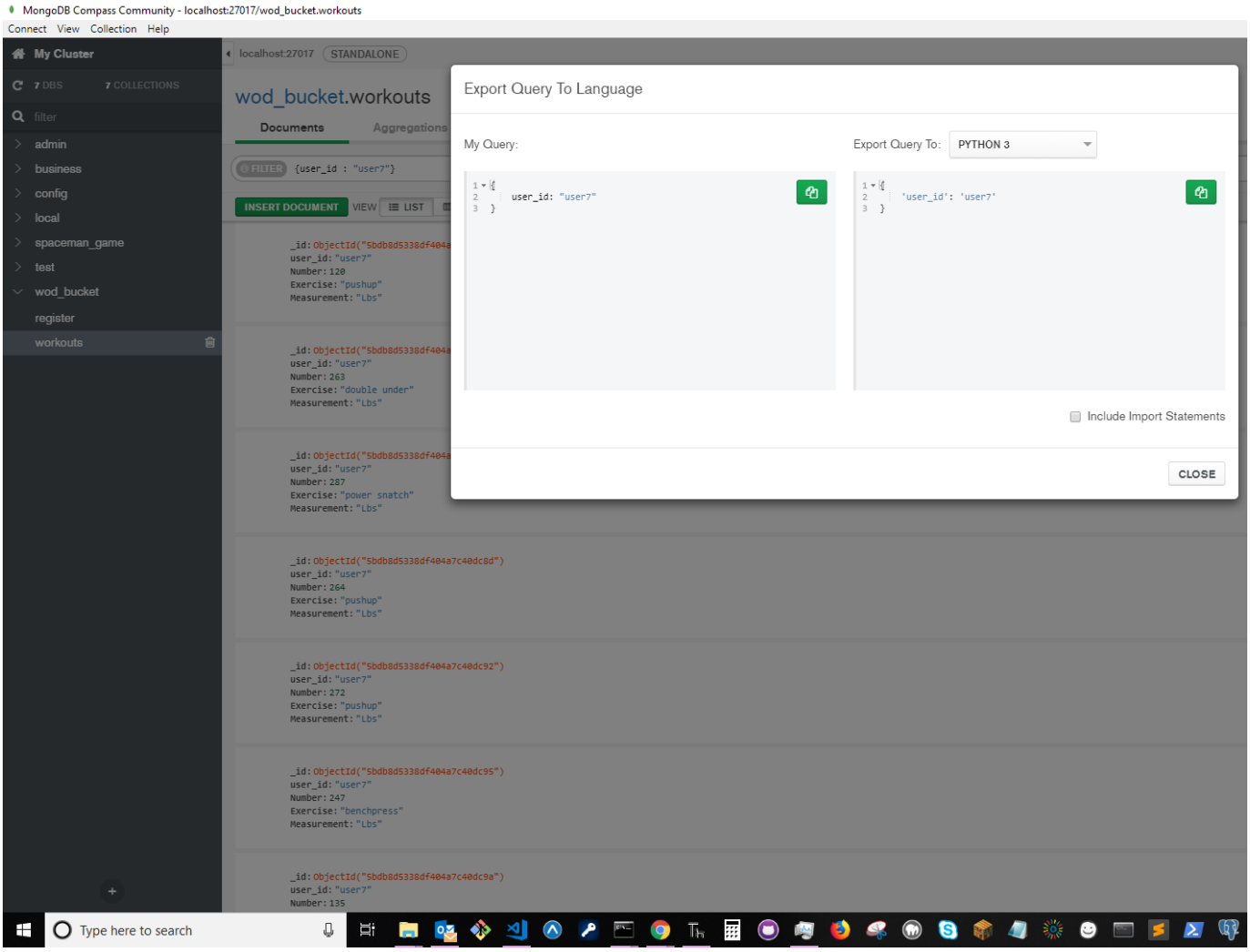

Figure 1 - Sample GET request

Dannielle Lewis – SWDV-691
Service Layer Design