MongoDb Design

For WODBucket, I will be using MongoDB as a database service. It will contain two tables; one for storing "user" objects and a second for storing "exercises" objects. It will have a one-to-many relationship entity. Conceptually, a user will have that capability to create more than one exercise with the exercises that tie to that specific user.

Table (Document) Users:

```
{
    "_id":{"$oid":"<<string>>",
    "FirstName":"<<string>>",
    "LastName":"<<string>>",
    "Email":"<<string>>",
    "Username":"<<string>>",
    "Password":"<<string>>",
    "Exercise_Ids":{
        "Exercise_1":{"$oid":"<<int>>"},
        "Exercise_2":{"$oid":"<<int>>"}
    }
}
```

Above represents the "user" table and associated fields. The "_id" is the primary key of the user. When a user registers, they will be given this unique ID. During registration, the database will use the entries given to populate the required fields with First Name, Last Name, Email, Username and Password. As exercises are created by this ID, the Exercise ID will be associated with the Primary key of this user. All fields will be string fields with the exception of the primary key and associated exercise object ids which are unique integers.

Table (Document) Exercises:

```
{
    "Exercise_1_id":{"$oid":"<<int>>"},
    "Exercise":"<<string>>",
    "Number":{"$numberInt":"<<int>>"},
    "Measurement":"<<string>>"
    },
 {
    "Exercise_2_id":{"$oid":"<<int>>"},
    "Exercise":"<<string>>",
    "Number":{"$numberInt":"<<int>>"},
    "Measurement":"<<string>>"
    }
}
```

Above represents the "exercise" table and associated files. The "Exercise_1_id" is the unique primary key that is assigned when an exercise is created and is associated with the primary key of the user that created it. In this way, a user will be have the ability to display all of the exercises that has been created and modify the exercise if necessary. Required fields are the exercise itself, the number and

measurement. The object ids are unique integers, the exercise and measurement is of the string data type and the number is an integer or float.

Example relationship strategy between tables (documents):

Above is an example of the fields with sample data that shows the relationships between the tables. Document_users shows the fields and the example data for the data types. In the Exercise_1 field of the users, the object id of the exercise_1 from the exercise table is shown as well as that of exercise_2. This represents the one-to-many relationship of the user to exercises. As shown, user “dfittest” has registered and added 2 exercises, the Deadlift and Swim to her profile. Upon registering, she enters simple data as strings such as her name, email, creates a username and password. Before adding this data to the database, verification (and hash and salt of the password) will be done through the controller. If the information is validated, the database will be updated with the new information. At this point, exercises can be added. The Deadlift measures in pounds as decided by the user and she lifted 190 during her last lift session. Her last swim, she swam 5 laps. Since measurement is a string, dfittest could have also entered “75 Meter Laps” in the measurements field and that would have been acceptable. So she swam 5, 75 Meter Laps during her last swim session.

Dannielle Lewis – SWDV 691
Document Database Design

The decision for the use of a document database as opposed to a relational database include a number of reasons. The data model is more flexible with the use of a document database. Changing the data model with the use of a relational database may be a complicated task depending on the amount of traffic and may be significant work for the developer. The use of a document database allows for dynamic modification of the schema without much impact to performance of the application. Growth and scalability is inherent with any successful application and a document database allows for this potential since they were designed with this focus in mind. In order to add attributes to a user, for example current weight, the database schema would have to be modified in a relational database. During production, it would be difficult to update all tables with associations with this field to maintain normalization. In a document database, only the key-value pair would need to be added for the new field. A relationship can be created by providing a reference in the document if necessary. Updates are done with ease, even in production.

As a result of the design feedback from peers, there are no additional updates needed to the database design for items required for the MVP. The authorization package, error handling and sign-out suggestions are all addressed in the service layer and will not require an update or a modification to the database. The remaining suggestions will be stretch features to be done once the MVP is accomplished.

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 and a modification of the database. This will be done by having users established as free version users or Pro users once 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. Integrating apps will use API to access data from other applications such as MyFitnessPal and MapMyRun. If the development team decides to store the data, this will force a modification of the database as well but since the data is stored in the other app, perhaps only an update of data or a pull of this data is necessary.