FitDeck Report

Team Members:

Daphne Williams

Shane Trentham

Introduction:

This project aims to create a mobile fitness application that enables the user to create and manage their own workouts. They will have the ability to search through specific exercises of their choosing and organize them into a unique workout or several unique workouts. If they want none of that and just want something already made for them, they have that option as well and can choose from several preselected session lists. Additionally, they will be able to track their workout time and calories burned progress via weekly averages of these statistics and graphs that show this daily information from Sunday-Saturday. This program will attempt to consolidate fitness and workout information into one easy to use application. We came up with this idea when thinking about times when we created our own workouts; my problem being it was too tedious to have to search through lists of workouts on my own and either screenshotting or manually typing

my choices out into a note on my phone's note application. This program would be something I would be able to use personally for my own fitness needs.

Technology:

For this project, we decided on using the Visual Studio IDE with Xamarin Forms in the c-sharp language; we both are very familiar with coding in c-sharp for school projects and thought it would be easiest for the both of us when developing this application. We also thought that, since having experience with Visual Studio for Windows WPF projects, that when dealing with a potentially multiplatform application we would use something of similar functionality to what we were used to that would easily pair well with iOS apps for Mac developers. Since this was the case, we both ended up developing the program on our Mac computers and testing with our iPhone mobile devices. For a portion of our project, we used calls to an API software called ExRx. This API contains an extensive database of exercises which is an integral part of the program as it centers around using many exercises to create workouts. It would have been okay to had code in every single exercise and in some ways, it could have been less trouble to implement. However, since it would have been so much more time consuming and would need much more memory, it would have been detrimental to the development of this project; my computer is already having a hard enough time running with this project open and I

am positive if I manually entered tens or even hundreds of exercises it would have exploded. At any rate, we went with ExRx for the simple fact that not only was it an expansive database of hundreds of exercises, but it was a free service for students to use; once the both of us proved we were students, we were able to receive a crash-course in the API by a support representative and were given access to the API. We ended up using a ASP.NET CORE WebApi as the middleman between the client and database. I had to use Microsoft Azure to host the api because my Apartments network would block any incoming requests and I don't have access to the router to open a port so the only realistic way to be able to remote access was to use a 3rd party cloud service.

Design:

The application opens to a page with just two buttons on it: a login button and signup button; they both ultimately end up at the same place, the ProfileHomePage, with a few more detours for the signup button. If the signup button is pressed, the user will be sent to a new page for setting up a login for the future; this page, the SignupPage, will have text entries for the user to enter a username, email address, password, and a confirmation of the password. If the username, email, and password are a unique combination, per the call to the db table, and the passwords match, the user will be sent forward to finish creating

their profile information section, if not, they will receive an alert telling them to both make sure that all boxes have entries and that the passwords match. When they proceed, they will need to enter more information about themselves to create their User object and to save their information in the User table in the database. The User class utilizes the fields of corresponding entities for the user's information; the class contains the variables username, emailAddress, password, name, dOB, weight, and height in the form of four strings, a DateTime, an int, and a float respectively. When they reach the FinishSignupPage, this is where they will enter the other User information and this will instantiate the class and save it in the database and lead them to the main screen, the ProfileHomeScreen. If the login button is pressed, an event triggers and the page will navigate to a login page that will ask for the user's username and password in order to allow for them to login to the system via a button click. The client will send an HTTP Post request with a Json message containing the username and password used to login to the webApi. From the API it will use .NET CORE's SignInManager to check the database for a matching username and verify the password. If the user exists, then the API will return a User object that contains an authentication token. If they are unable to login, meaning their credentials do not match one existing in the system, they will remain on this screen but if they are successful, they will be navigated to the ProfileHomePage.

After reaching this screen, the two roads merge and the rest of the process throughout the application is the same. This ProfileHomePage outright loads the user's information to the screen, and they will see all of their already-entered health information; they will also see sections for the Daily Stats and Weekly Stats as well as a series of buttons pertaining to the workouts and the list view that will be populated with said workouts. The first button, which reads "Add Workouts" will display an action sheet that asks the user if they would like to create a workout, add a preselected workout, or cancel. Hitting cancel will just leave them on the current page but selecting either of the other two will send them to a new window. Selecting create a workout will bring the user to the CreateWorkoutPage which contains two columns and a series of pickers, some of which will populate based on the entry of the others. The first column is for selecting a workout based on muscle group; the first picker will allow the user to select a base muscle group, like Upper Arms or Thighs, and will populate the next picker that will give the specific muscle groups based on that first selection, Uppers Arms will trigger it to have the selections Triceps Brachii, Biceps Brachii, and Brachialis, and the third picker is already populated with apparatuses that can be used, bodyweight and dumbbells for example. The second column is for selecting based on the type of workout which is the first picker's options, the options being Weight Training, Plyometrics, Cardio and Conditioning, and Stretch; weight training will just defer

the user to picking based on a muscle group since it requires the same way to call the API, Plyometrics will populate the second picker with movements, vertical or lateral etc. and will add a third picker to define intensity of the movement, Cardio and Conditioning will add the cardio category selections to the picker, and stretch will add body parts. When the user has made selections, they will need to hit the find exercise button which will do one of two things, display an error message alert if the appropriate pickers are not complete or it will call the API. If the selections are not valid, meaning the exercises of that specific type either do not exist or are not available to use, an alert will display telling the user to make another selection. If the selections are valid, the ExRx API will be called once to authenticate the username and password of the ExRxDatabase class and the second will use the token, which was stored in the ExRxDatabaseResponseModel in response to the first call, received from the previous call and call the ExRx database for any exercises that contain those credentials that are tier-accessible to us; these calls are handled by the RestService class that contains variable fields to store the HttpClient and jsonSerializer, a constructor that initializes the client and base address of all the calls, and five methods that call the API based on specific credentials: the AuthenticationUserAsync that returns the bearer toke, the weightExercises that returns the list of weight training exercises based on the apparatus used and the specific muscle group given, the plyoExercises that returns

a list of plyometric exercises based on movement and intensity, stretchExercises that return a list of stretch exercises based on the given body part, cardioExercises that returns a list of cardio exercises based on the cardio subcategory, and getExerciseById that returns a list of exercises based on the ids passed to it. The exercises will populate a list view with their Exercise_Name, Instructions_Preparation, and Instructions_Execution visible to the user; these entries come from the class ExerciseResponseModel with takes the calls made by any of the previous methods and deserializes them into a list of exercises, titled Exercis, that contains over 30 fields which will be shown in the diagram following the design description. When an exercise is selected from the list, an even will trigger that will display an action sheet that asks if they would like to add the exercise to their workout; if they don't want to, they would just select cancel but if they would like to, they would select add exercise workout. This stores the exercise in the Exercise class with the Name, Type, and MuscleGroup, saves the exercise to the list and puts it in a separate list view where all added exercises will show. They will be able to continue this process of selecting and adding exercises to their hearts content and when finished, they will hit a button to publish the workout. This will cause a popup window that asks the user what day to save the workout on and to add a name to it; if either of these characteristics are violated (zero or more than one day or no name) an alert will display telling them to fix the issue. Once

done, they will hit another button to finally add the workout which fires off the event to store the workout in the Workout class in the fields List<Exercise> exercises, via the addExercise() method each time an exercise is added, and the Day and Name just chosen by the user; save workout to database; will bring them back to the ProfileHomePage; and add the workout to the listv iew on this page; this will also save the workout in the Workouts class via a List<Workout> workouts field utilizing the addWorkout() method.

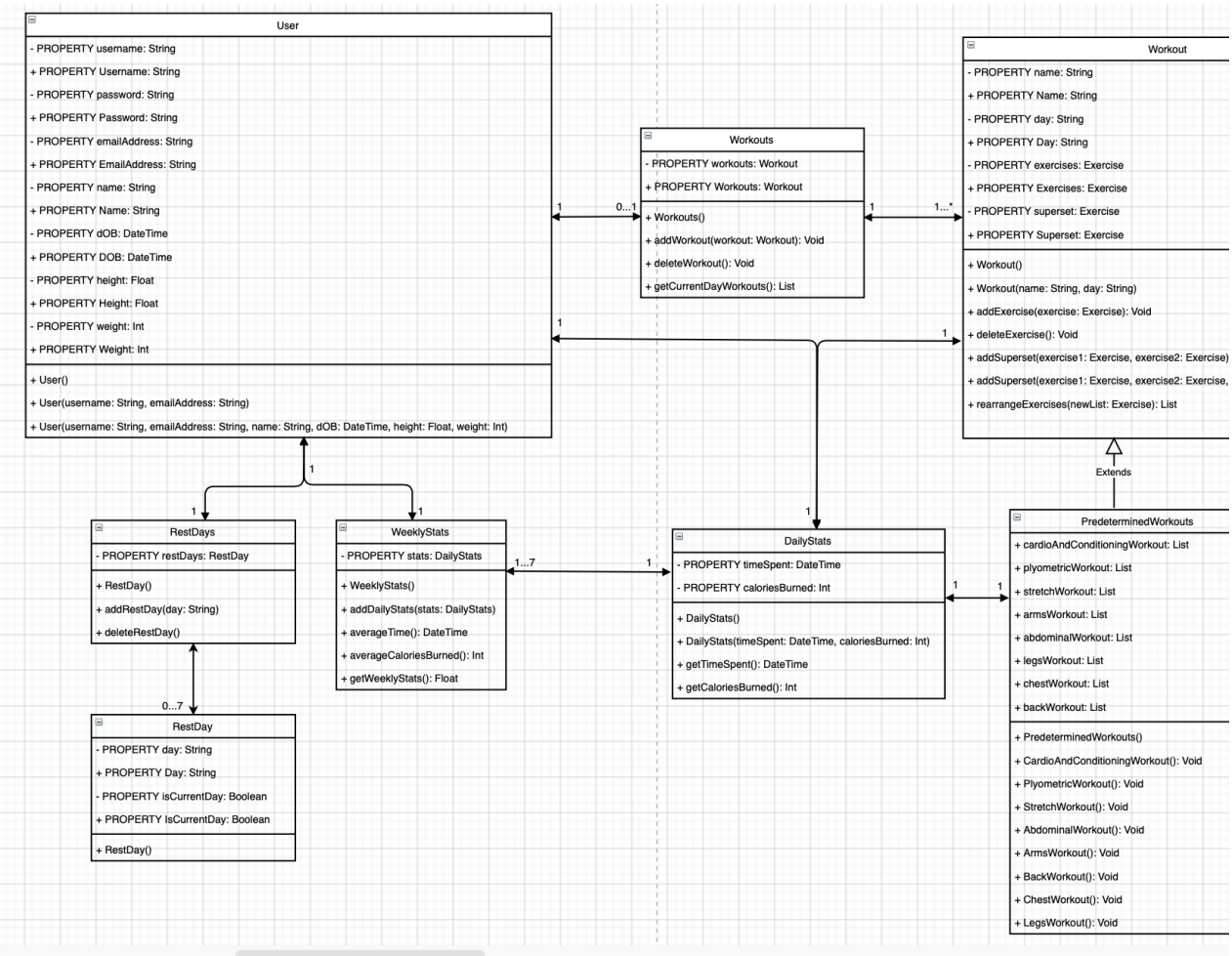
If the user wants to just add a preselected workout, they will just have to select the add workout button and choose the add Preselected Workout option. This will trigger a click event that sends them to the PredeterminedWorkoutPage where they will be able to select from the following already-created workouts that exist in the PreselectedWorkout class that extens the regular Workout class: Cardio and Conditioning, Plyometrics, Stretch, Abdominals, Arms, Back, Chest, and Legs. After making a selection and hitting the button to solidify their choice, a list view will show all of the exercises in that workout out along with how to set up and how to execute the exercise safely; these exercises are given to the user by way of calling the API's getExerciseById method for each Exercise_Id in the list. If they do not like this selection, they can simply choose again and once satisfied, they can choose to add the workout to their workouts saves the workout to database, brings them back to the ProfileHomePage, and adds the workout to the list view of

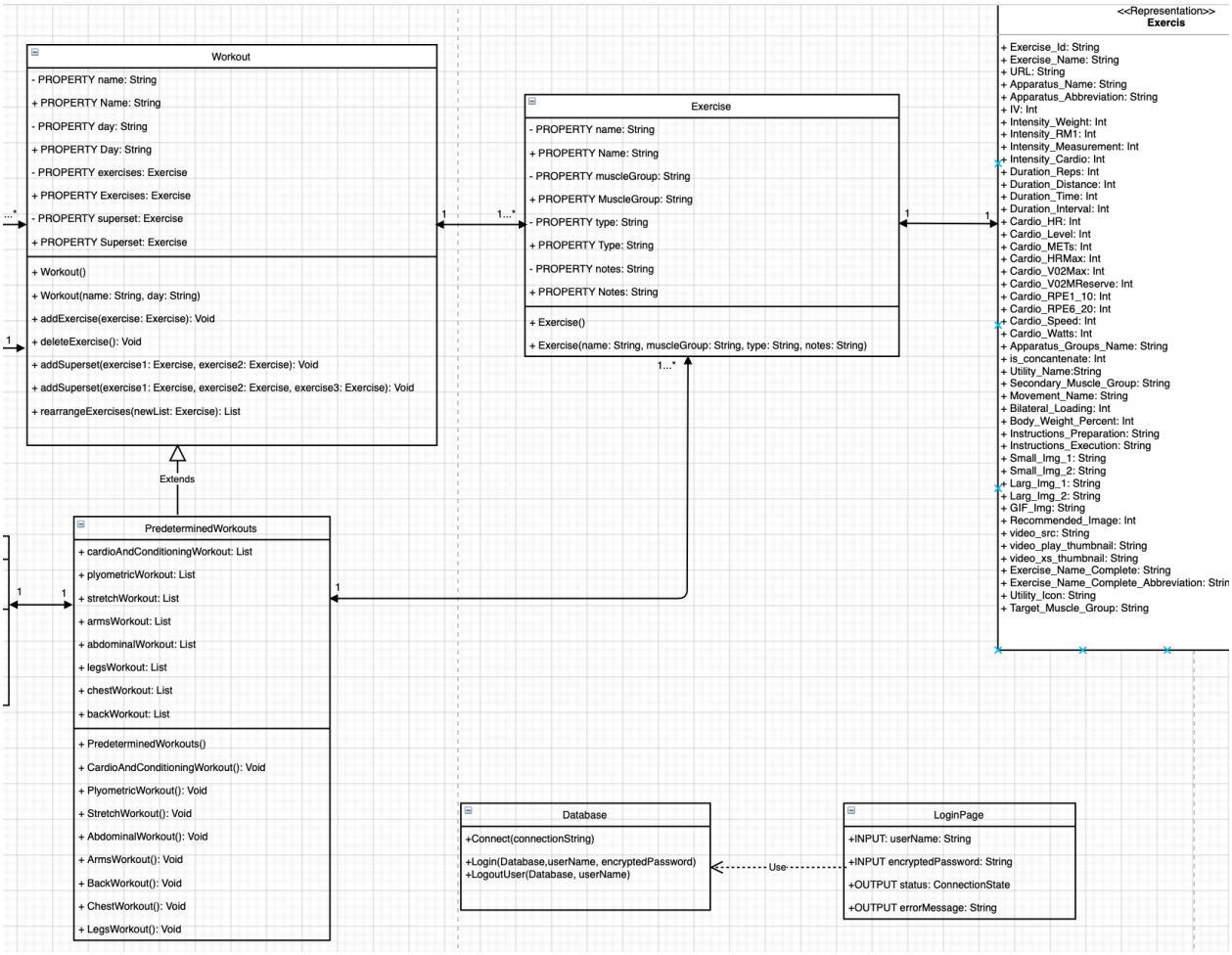
workouts on this ProfileHomePage, which stores the workout in the Workouts class list of workouts.

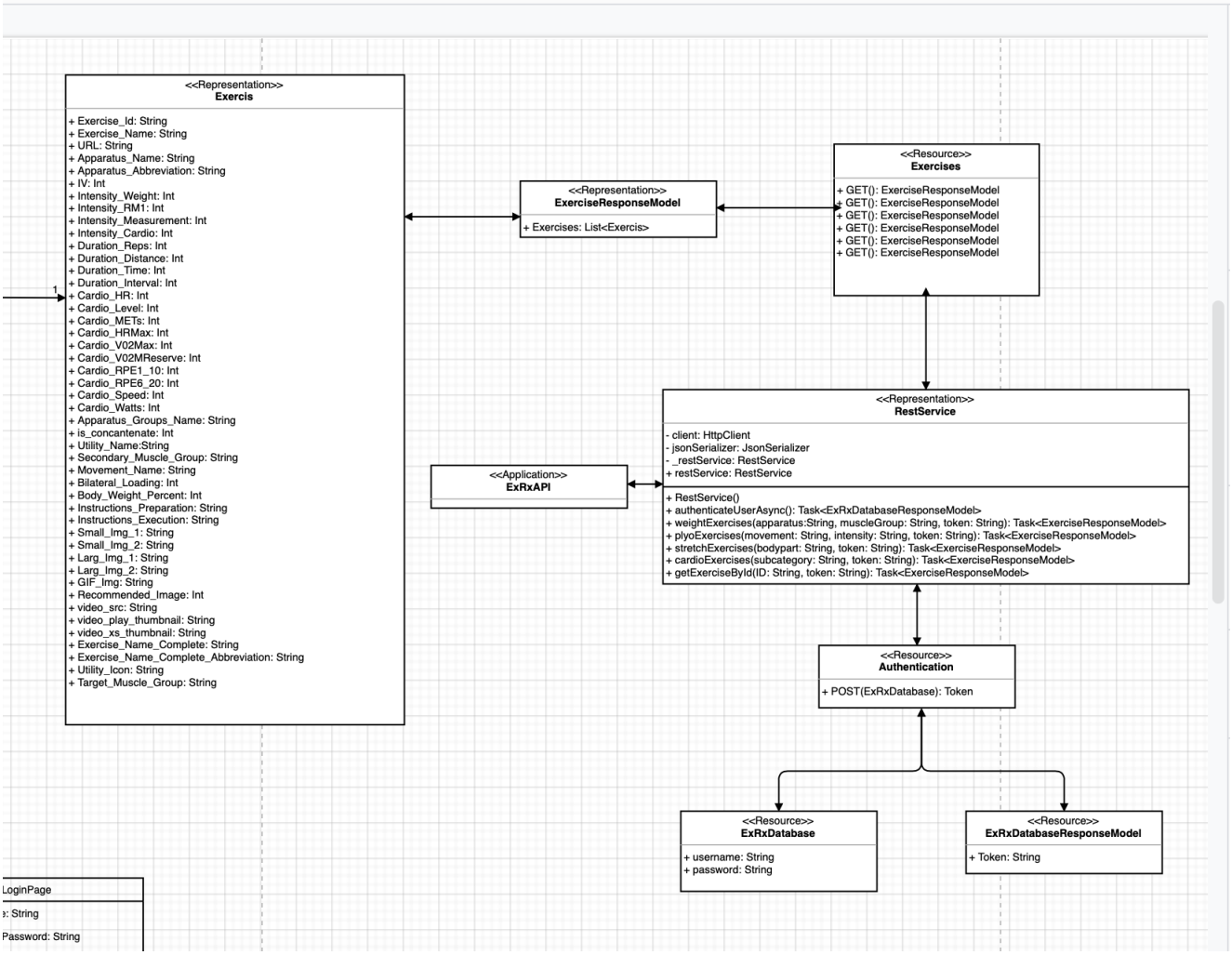
From this page, once they have added at least one workout, they can select the other buttons such as edit workout or start the workout. When the user chooses to edit the workout, they will first need to select that particular button which will pop up the picker and buttons to edit the workout; the picker is needed to choose the workout from the user's list of workouts and the buttons edit and delete workout exist to either to do just as they say. If the user chooses to edit, it will bring up a popup page similar to the CreateWorkoutPage where there are several pickers used to populate other pickers based on muscle group and exercise type and to call the database to retrieve the exercises of that muscle group or type of that apparatus of specific type respectfully. Just as with the CreateWorkoutPage, it will take that given information and put it to a listview and once they are finished and press the button that says so, they will see the pop up window close and notice they are back on the ProfileHomePage. If the user decides to delete a workout they would need to, again, press the edit workout button and select the workout from the picker and hit the delete workout button and this will remove the workout from the Workouts class list of Workouts. The can also choose to add rest days, which stem from the class RestDay that store a DayOfWeek Day and a boolean isCurrentDay(), where the user can set multiple values that will store each

individual rest day in a RestDays list of rest days (List<RestDay> restDays) by using the addRestDay method. The user also can press start a workout which will popup a section that contains a picker for selecting a workout to start, a label to view the stopwatch timer, and three buttons: one to start the stopwatch, another to pause the stopwatch, and the final one to stop the time and end the workout. There additionally is a section of the page dedicated to DailyStats, a class that calculated caloriesBurned() in a single workout and intakes the length of a workout given by the start workout stopwatch, and WeeklyStats, a class that stores a list of DailyStats using addDailyStats(), List<DailyStats> dailyStats, and averaged the calories and time spent working out each week. However, these stats are not visible due to time constraints and were left out in the execution (see known bugs). Finally, at the very top of the ProfileHomePage, the user can select to logout; this will trigger an event to return to the first login/signup screen and will allow the user to log back in using their created username and password.

Class Diagram:




Workout
- PROPERTY name: String
+ PROPERTY Name: String
- PROPERTY day: String
+ PROPERTY Day: String
- PROPERTY exercises: Exercise
+ PROPERTY Exercises: Exercise
- PROPERTY superset: Exercise
+ PROPERTY Superset: Exercise
+ Workout()
+ Workout(name: String, day: String)
+ addExercise(exercise: Exercise): Void
+ deleteExercise(): Void
+ addSuperset(exercise1: Exercise, exercise2: Exercise): Void
+ addSuperset(exercise1: Exercise, exercise2: Exercise, exercise3: Exercise): Void
+ rearrangeExercises(newList: Exercise): List
Exercise
- PROPERTY name: String
+ PROPERTY Name: String
- PROPERTY muscleGroup: String
+ PROPERTY MuscleGroup: String
- PROPERTY type: String
+ PROPERTY Type: String
- PROPERTY notes: String
+ PROPERTY Notes: String
+ Exercise()
+ Exercise(name: String, muscleGroup: String, type: String, notes: String)
<<Representation>>
Exercis
+ Exercise_Id: String
+ Exercise_Name: String
+ URL: String
+ Apparatus_Name: String
+ Apparatus_Abbreviation: String
+ IV: Int
+ Intensity_Weight: Int
+ Intensity_RM1: Int
+ Intensity_Measurement: Int
+ Intensity_Cardio: Int
+ Duration_Reps: Int
+ Duration_Distance: Int
+ Duration_Time: Int
+ Duration_Interval: Int
+ Cardio_HR: Int
+ Cardio_Level: Int
+ Cardio_METs: Int
+ Cardio_HRMax: Int
+ Cardio_V02Max: Int
+ Cardio_V02MReserve: Int
+ Cardio_RPE1_10: Int
+ Cardio_RPE6_20: Int
+ Cardio_Speed: Int
+ Cardio_Watts: Int
+ Apparatus_Groups_Name: String
+ is_concantenate: Int
+ Utility_Name:String
+ Secondary_Muscle_Group: String
+ Movement_Name: String
+ Bilateral_Loading: Int
+ Body_Weight_Percent: Int
+ Instructions_Preparation: String
+ Instructions_Execution: String
+ Small_Img_1: String
+ Small_Img_2: String
+ Larg_Img_1: String
+ Larg_Img_2: String
+ GIF_Img: String
+ Recommended_Image: Int
+ video_src: String
+ video_play_thumbnail: String
+ video_xs_thumbnail: String
+ Exercise_Name_Complete: String
+ Exercise_Name_Complete_Abbreviation: Strin
+ Utility_Icon: String
+ Target_Muscle_Group: String
Extends
PredeterminedWorkouts
+ cardioAndConditioningWorkout: List
+ plyometricWorkout: List
+ stretchWorkout: List
+ armsWorkout: List
+ abdominalWorkout: List
+ legsWorkout: List
+ chestWorkout: List
+ backWorkout: List
+ PredeterminedWorkouts()
+ CardioAndConditioningWorkout(): Void
+ PlyometricWorkout(): Void
+ StretchWorkout(): Void
+ AbdominalWorkout(): Void
+ ArmsWorkout(): Void
+ BackWorkout(): Void
+ ChestWorkout(): Void
+ LegsWorkout(): Void
Database
+Connect(connectionString)
+Login(Database,userName, encryptedPassword)
+LogoutUser(Database, userName)
Use
LoginPage
+INPUT: userName: String
+INPUT encryptedPassword: String
+OUTPUT status: ConnectionState
+OUTPUT errorMessage: String



How to Deploy FitDeck:

In order to deploy our software, one would need to at least have access to an iPhone simulator, if not an actual iPhone which would be more beneficial since simulators can be finicky and take a while to startup and load the project. If trying to build the software, the developer would need to probably use Visual Studio with xamarin forms and xamarin essentials just to make things easier although not entirely necessary. They would need to add access to json deserializers (we used a

NuGet packages Newtonsoft.Json), web API access (we used Microsoft.AspNet.WebApi.Client), and graph creating software (we attempted to use Syncfusion.Xamarin.SfChart).

Known Bugs:

There are several known bugs in our project as we both are aware of; with limited time and heavy course loads for this semester, it was hard to manage enough time to get everything into the project and as with all technology, there are bound to be bugs. The first main issue is no daily or weekly stats information or graphs; the graphs would not even load blank ones and the stats themselves fell by the wayside to make room for focusing on the essential parts of the project. Another bug is that when the user goes to select a workout to begin or a workout to edit, the pickers are populated with the type (i.e. FitDeck_CSCI4805.Workout) instead of the name of the workout; I found this especially strange since the code used to take the workout and populate the profile list view is one and the same. Another bug can be seen when the user adds their first workout and returns to the screen, everything is in place, but when they add another workout, only that open will appear and the other will be gone from view. Additionally, the program is known to crash when either the stopwatch for the start exercise is pressed or when the rest days are added. I am positive there are many more, such as the view being

skewed when certain buttons are clicked or when certain picker items are selected on account of using the absolute layout without anchors, and even more that we do not know about.

Future Work:

In the future, the main goal would be definitely to fix any and all known bugs, especially the once that greatly affect functionality and add a bit more graphic to the program in the form of images and video demonstrations of the exercises to make more user-friendly. I personally would like to add a portion of the programs that adds daily steps counted and a heart rate monitor to the daily and weekly stats and graphs. I would also like to implement additional preselected exercises to allow the user to have more options as well as add to the already-existing workouts.