UNIVERSITY OF LA VERNE
LA VERNE, CALIFORNIA
FALL 2020

SENIOR PROJECT

TITLE: FitU

A SENIOR PROJECT SUBMITTED TO:

THE FACULTY OF

COMPUTER SCIENCE AND COMPUTER ENGINEERING

IN CANDIDANCY FOR THE DEGREE OF

BACHELORS OF SCIENCE

COMPUTER SCIENCE AND COMPUTER ENGINEERING
WITH A CONCENTRATION IN INTERNET PROGRAMMING & SOFTWARE

BY

DAVID RAYGOZA

AFFILIATION:
The affiliation identifies the organization which the author conducted

UNIVERSITY PROJECT ADVISOR:
PROF. JOZEF GOETZ

Table of Contents

1. Introduction
2. Background/Literature Review
3. Analysis and Requirements
4. Design
5. Implementation
6. Test and Integration
7. Deployment
8. Conclusion
9. References
10. Appendices

The project REPORT, power point and **app zip file is uploaded at:** http://students.laverne.edu/~draygozada/CMPS499/

Google Drive Link for file download: https://drive.google.com/drive/folders/1l0QckKu4EAf483PrujelNEhUgXiM6oCJ?usp=sharing

Keep in mind, the app can only be ran using Xcode on a Mac. Xcode at times does not properly compile app unless it is opened on the original device it is being developed on, for security reasons. If you have trouble compiling it on Xcode please let me know and I can share it with you through iCloud.

Abstract:

The purpose of the project is to develop a mobile application where fitness people from all around the world can keep track of water intake, workouts, meal plans, and all other vital information all in one app. This uses a database to record the information of registered users, and their personal workout, and meal plans as well as set workout/meal data from the admin user. Non-registered and logged in users are unable to access any of the app features, but they can create an account, or verify their information to reset their forgotten password from a previous account. Registered users can simply login with their email and password to add, delete, view, and edit their own workouts/foods, and access all fitness information. Admins with full permissions are able to delete registered users as well as edit, add, and delete global workout/meal plan data from the database. Admins have access to manage foods, manage users, and manage workouts. The mobile application consists of a main page as well as other pages such as Home, Water, Diet, Exercise, Mindset, Sleep, Login, Forgot Password, Create Account, and a Manage page thats only available to a admins. The app is developed on Xcode using Swift, SwiftUI, SQLite, and Objective-C.

1. Introduction

Nowadays there's an app or everything, and there are some amazing fitness apps out there. However, they can sometime be difficult to find and overwhelming to put all of the information together in order to reach your fitness goals. My solution to this is an all-around fitness app called FitU (Fitness University) that will squash the initial fear of starting your fitness journey and even help experienced athletes improve by giving you everything you need all in one app. From water intake, to motivation, to personalized workout and meals plans. The project requires to develop a mobile application that allows users to list, insert, edit, and delete workouts/meals from their personalized plans The application will consist of multiple pages that will limit accessibility depending on if the user is signed in. Users logged in will have access to all public pages to view information, but will not have permissions to edit, delete, or add new workouts and foods that are not their own. The database driven app will implement the following pages: ViewController.swift(Main), HomeViewController.swift(Home), NewViewController.swift(water intake), DietViewController.swift, ExerciseViewController.swift, MindsetViewController.swift, and LoginViewController.swift. Finally, the project will be implemented using SwiftUI, Objective-C, and SQLite all on the new Xcode 12.

2. Background/Literature Review

FitU puts the power of fitness right into the palm of your hand. One can even learn how to live a well-rounded fitness lifestyle from every perspective. Use our water intake and dietary algorithms to find out how much water and food you should be putting into your body. Select from our database of foods and workouts to create your personalized workout and meal plans. Make sure to stay motivated by visiting our mindset page to get some tips and tricks on how to stay on track to living a healthier lifestyle.

3. Analysis and Requirements

- I chose to use Xcode/iOS development for my project because the market is huge. Yes around 80% of all users worldwide are android users, but yet iPhone apps continue to make almost double the money that android apps make. That's because while 80% of users across all ages worldwide use android, apps are mostly targeted towards younger demographics and here's where iOS wins – 83% of the younger generation here in the United States use iPhones. So this is where I decided it's either time to join in or get left behind.

The following is a list of the mobile interfaces and functional requirements for the proposed project. These include the **Mobile design best practices** for mobile app development and design. The app will be accessible and responsive on many devices such as smartphones and tablets.

1. App Design
    a. The app should conform to the Mobile App Best Practices Checklist **https://buildfire.com/top-mobile-development-practices/ & https://www.sitepoint.com/7-best-practices-designing-mobile-user-experience/**

2. App Navigation
    a. The navigation should be clear and easy to follow on every page
    b. Back button should be on every page (except main page)

3. Page Layout Design
    a. Easy access to all pages
    b. The app should appeal to the audience who would be viewing it
    c. The app should have a consistent header and minimal layout throughout each page
    d. Page should be viewable in tablet and smartphone resolutions with no horizontal scrolling.

e. The contrast between the background and content should be strong to ensure ease while reading
f. The app should have a logical alignment of content
g. The homepage of the app should load in a reasonable amount of time for a 56 kbps connection
h. There should be a good balance between content and “whitespace”

4. Color and Graphics
    a. The color of the app should be appealing to the eye, but should maintain a loose business feeling (no dark colors)
    b. Color should be used consistently throughout the entire app
    c. Color should not be used exclusively to convey meaning
    d. Graphics should not have a significant impact on the loading time of the app
    e. Alt text should be used on all images/videos

5. Content Presentation
    a. Information should be easy to find within the app
    b. Common fonts should be used (Times New Roman, Arial, Sans serif, etc.)
    c. Content organization should follow a consistent pattern
    d. Font sizes, font color, and font style should remain consistent

6. Accessibility
    a. The alt text should be used on any images or videos
    b. Captions should be used on any video content
    c. “Safe” colors should be used

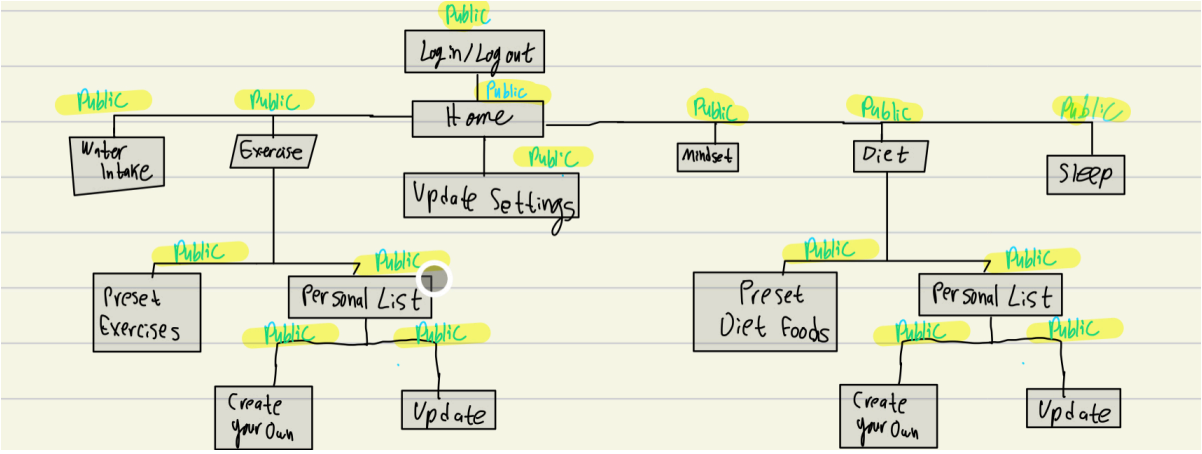
7. Special Requirements
    a. The app should be clean, simple, and easy to navigate/control for all users
    b. The app shall provide a way for the users to receive a newsletter from the app that notifies the users of new content

8. Mobile Best Practice Requirements

    a. Descriptive header(logo)
    b. Descriptive page title
    c. Descriptive alternate text for images/videos
    d. Provide “Back to Top” hyperlink
    e. Design “One App”, one code version for optimal display on multiple types of devices (iPads/iPhones)
    f. Optimize Layout, Navigation, Graphics, and Text for dynamic use

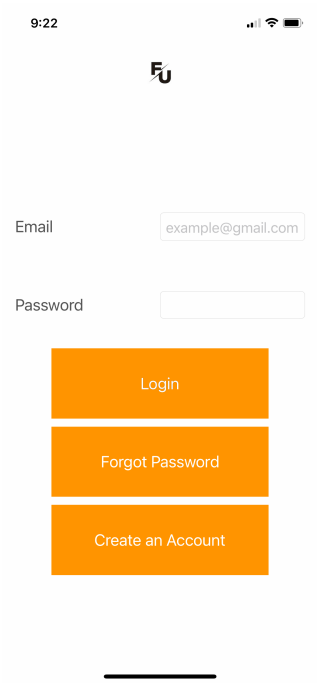
4. Design

1. Site Map

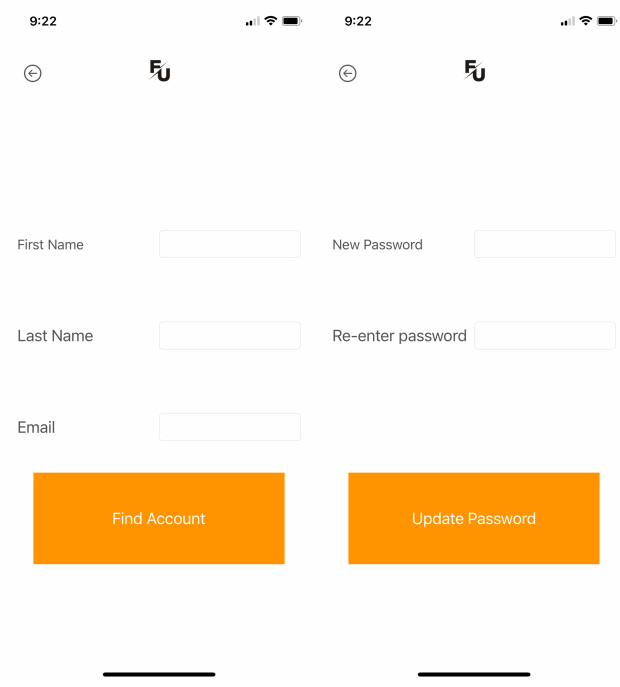


2. Interface Screenshots

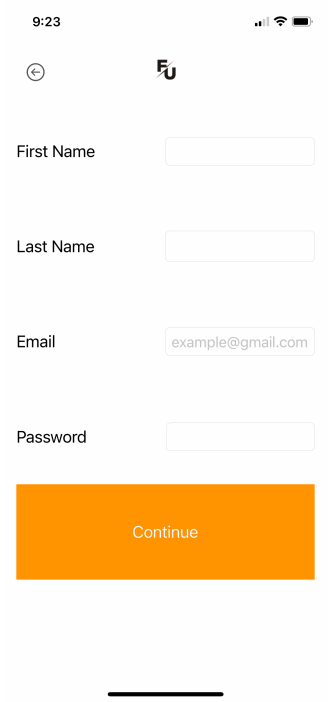
Login page

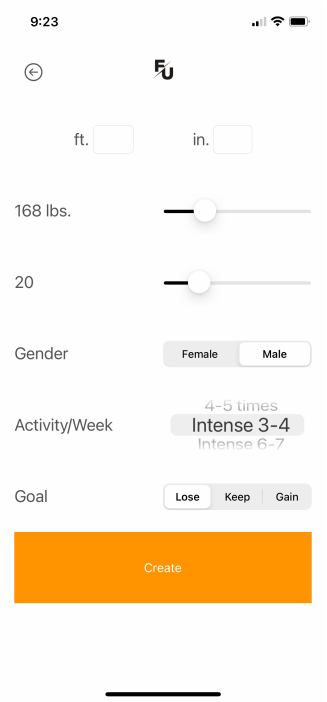


Forgot Password Pages

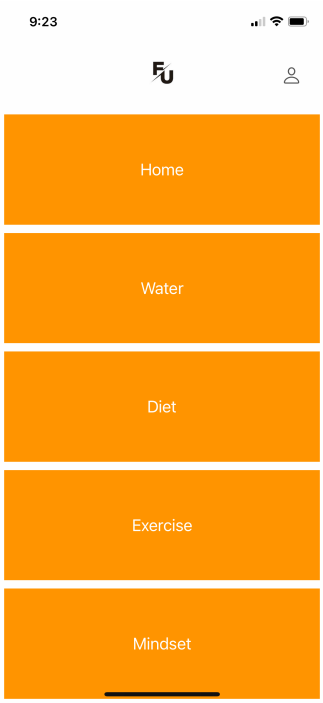


Create Account Pages

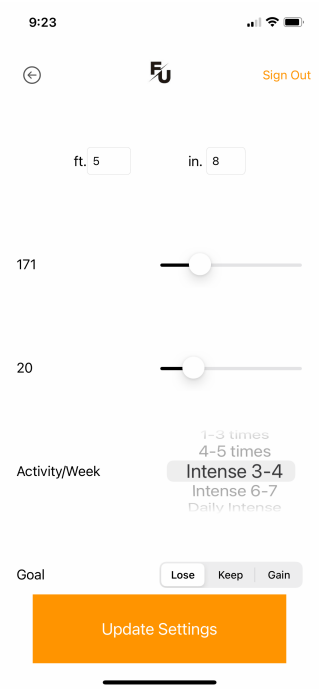




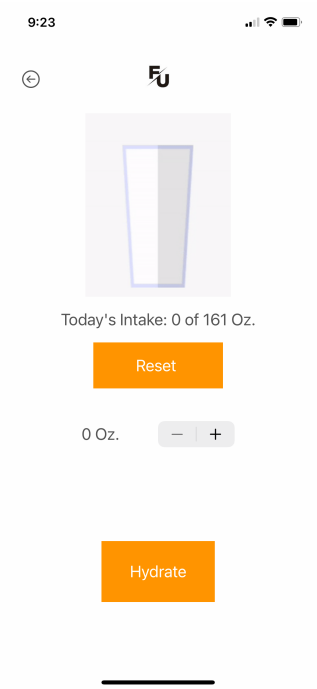
Main page



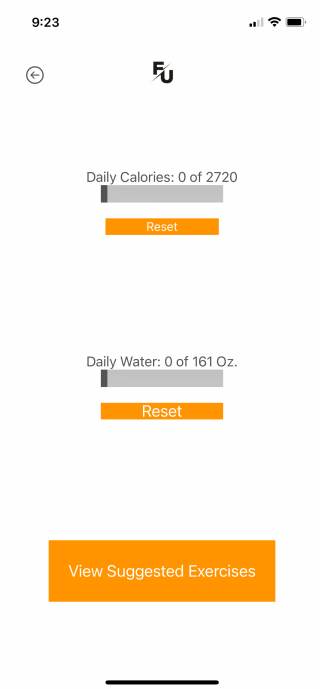
Update Profile Settings Page



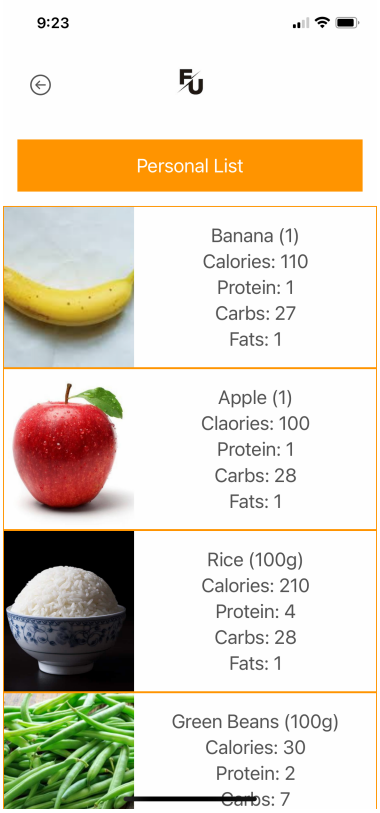
Water page



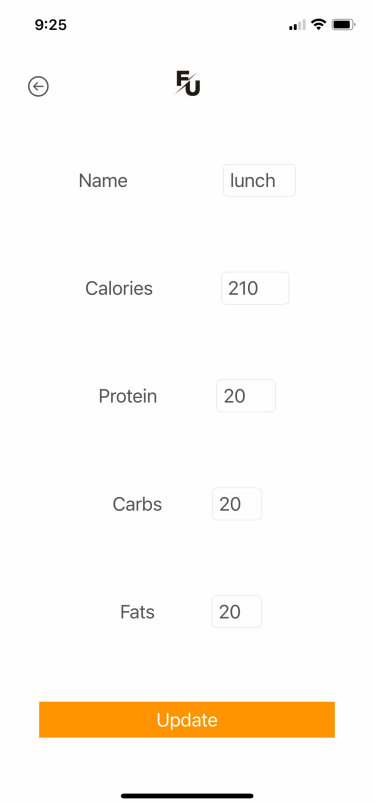
Home page



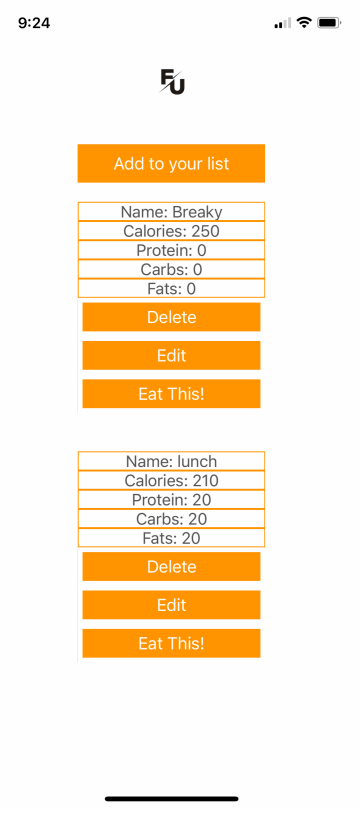
Diet page



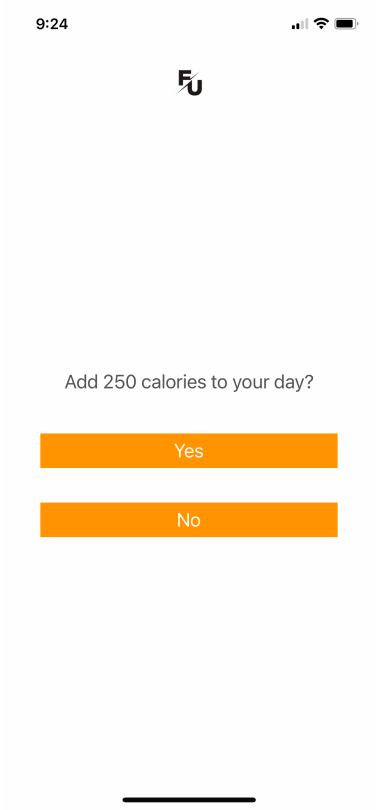
Edit/Create Food Page



Personal Food List Page

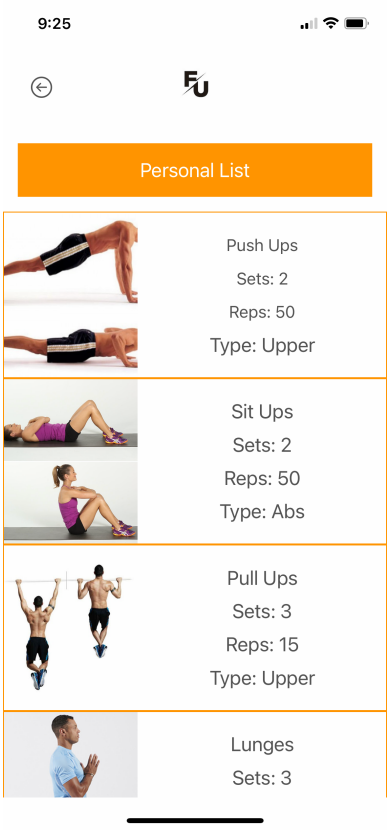


Eat Food Page



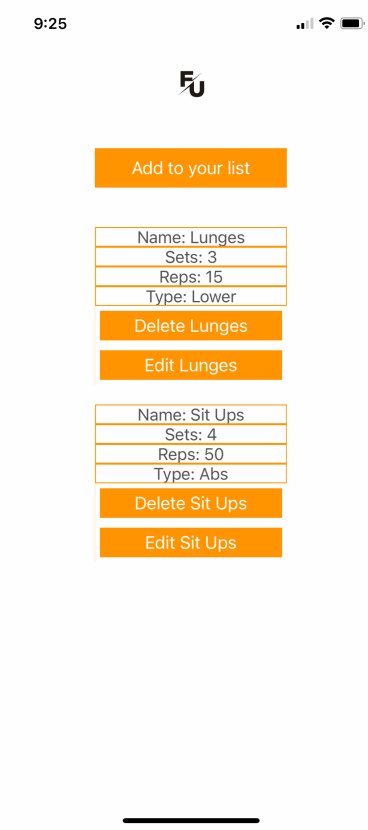
Delete Food/Exercise Page

Exercise page



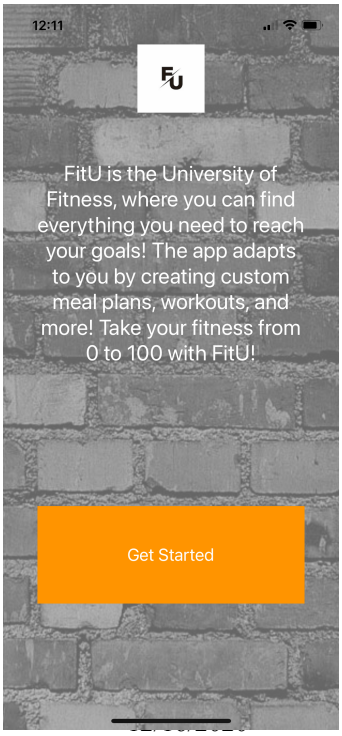
Edit/Create Exercise Page

Personal Exercise List Page



Mindset page

Splash Welcome Page



3. **Database Tables**
    a. Xcode does not have a feature to view database relationship tables. Either way a relationship table is not necessary because all tables are hosed locally on user devices so there's not need for table relationships or authentication.

Author Table

Column	Type
Id	INTEGER
fname	TEXT
lname	TEXT
username (email)	TEXT
password	TEXT
height	DOUBLE
weight	INTEGER
age	INTEGER
gender	TEXT
activity	TEXT
goal	TEXT

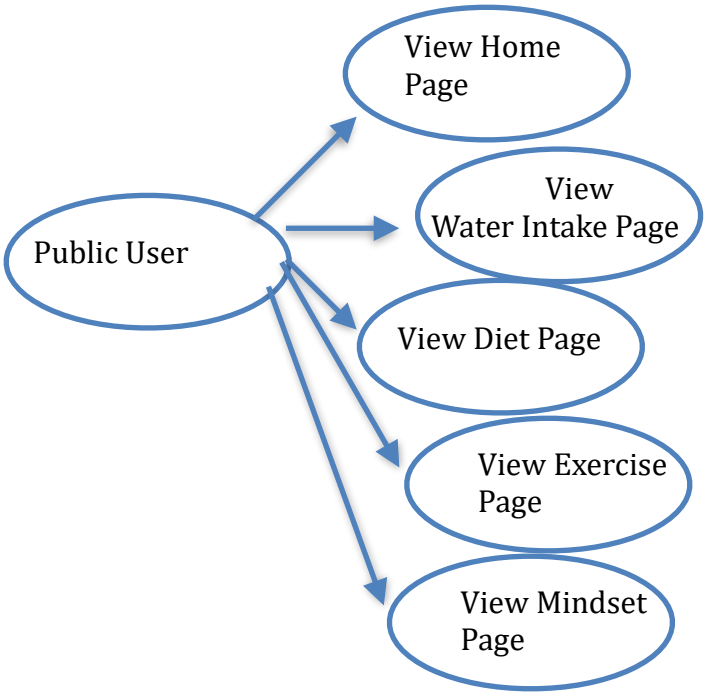
User Food Table

Column	Type
Id	INT
name	TEXT
calories	INTEGER
protein	INTEGER
carbs	INTEGER
fats	INTEGER

User Exercise Table

Column	Type
Id	
name	TEXT
sets	INTEGER
reps	INTEGER
type	TEXT

4. Use Case Diagram



5. Implementation

```
 let createAuthorsTable1 = "CREATE TABLE IF NOT EXISTS authorData3 (id
     INTEGER PRIMARY KEY AUTOINCREMENT,fname TEXT,lname TEXT,username
     TEXT,password TEXT, height DOUBLE, weight INTEGER, age INTEGER,
     gender TEXT, activity TEXT, goal TEXT);"
if sqlite3_exec(db, createAuthorsTable1, nil, nil, nil) != SQLITE_OK{
    print ("Error opening authors table")
 return
 }
```

```
let createfoodTable = "CREATE TABLE IF NOT EXISTS userfoodTable (id
    INTEGER PRIMARY KEY AUTOINCREMENT,name TEXT,calories INTEGER, protein
    INTEGER, carbs INTEGER, fats INTEGER);"
if sqlite3_exec(db, createfoodTable, nil, nil, nil) != SQLITE_OK{
    print ("Error opening user foods table")
 return
}
```

```
let createuserExerciseTable2 = "CREATE TABLE IF NOT EXISTS
    userExerciseTable2 (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT,
    sets INTEGER, reps INTEGER, type TEXT);"
if sqlite3_exec(db, createuserExerciseTable2, nil, nil, nil) != SQLITE_OK{
    print ("Error opening user exercise table")
 return
}
```

WARNING: Swift app development uses SQLITE which does not have a table viewer like phpmyadmin, so all database tables are viewed through the console outputs

```
Query Result for authorData3 table:

Id | fname  | lname | username | password | Height | Weight | Age | Gender | Activity | Goal
5 | David | Raygoza | draygoza2799@gmail.com | 123 | 172 | 170 | 20 | Male | 3 | Lose
6 | Dan | Baker | dan@gmail.com | pass | 167 | 202 | 22 | Female | 1 | Lose
7 | Kim | Hernandez | kim@yahoo.com | kim123 | 185 | 211 | 13 | Female | 1 | Keep
8 | John | Doe | johndoe@aol.com | jogndoe | 185 | 173 | 20 | Male | 3 | Gain
9 | Lesslie | Porter | porter@gmail.com | lporter | 154 | 112 | 16 | Female | 1 | Lose


Id | Name  | Calories | Protein | Carbs | Fats
64 | Apple | 100| 1| 1 | 1
60 | Chicken Breast | 200| 20| 10 | 5
61 | Green Beans | 100| 5| 2 | 2
63 | Talapia Fish | 150| 15| 5 | 5
62 | White Rice | 200| 4| 1 | 1


Query Result for User Exercises:

Id | Name  | Sets | Reps | Type
48 | Bench Press | 3| 8| 0
49 | Dumbbell Curls | 2| 25| 0
46 | Lunges | 3| 15| 1
45 | Push Ups | 2| 50| 0
47 | Sit Ups | 2| 50| 2
```

SQLite on XCODE is run locally on user device so there is no database to export, or way to view table as a tree form from command line

Graphical hierarchy of files and relationships between all files based on hyperlinks, buttons, and methods, and statements in each file.

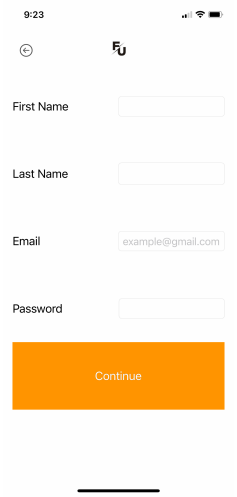
Directories	File Names with a short description	"Includes" Function Names
FitU	**Main page(ViewControllerA.swift): Gives full navigation to all public apps features.**	homeButton() waterButton() dietButton() exerciseButton() mindsetButton() showLoginController() viewDidLoad() mainDBInfo()
FitU	**Home Page(HomeViewController.swift): Allows user to view daily calorie/ water tracking determined by specified goals**	suggestedExercisesButton() waterReset() inputFitnessData2() calorieReset() getProgresses() viewDidLoad() mainDBInfo()
FitU	**Profile Page(ProfileViewController.swift): Allows user to update personal data in order to change app parameters.**	updateSettings() findFitnessDataIndexForProfile() inputRememberedEmailForProfile() inputAccountsForPorfile() signOutButton() viewDidLoad() mainDBInfo()
FitU	**Water intake Page(NewViewController.swift): Uses personal user data such as weight, and activity level to calculate suggested daily water intake. Also, allows users to manually enter water consumption for tracking**	intakeButton() inputFitnessDataForWater() getWaterPercentage() viewDidLoad() mainDBInfo()
FitU	**Diet Page(DietViewController.swift): Displays diet foods, and allows add food to their daily calories by clicking on them**	deleteinsertcalories() viewPersonalFoodListClick() openConfirmPage() makeClickableItems() viewDidLoad() mainDBInfo()

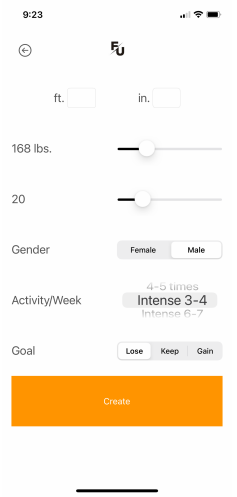
FitU	**Confirm Food Addition Page(ConfirmFoodAdditionViewController.swift): Allows user to confirm and insert food addition to their personal list**	inputCals() inputFitnessDataForConfirmFood() yesButton() noButton() viewDidLoad() mainDBInfo()
FitU	**Create Food Page(CreateFoodViewController.swift): Allows user to enter data to create a food for their personal list**	createFoodButton() viewDidLoad()mainDBInfo() mainDBInfo()
FitU	**User Foods Page(UserFoodsViewController.swift): Dynamically displays user's personal food list from database table**	deleteTaggerFromDB() insertTaggerIntoDB() inputUserFoods() createRecords() deleteAction() editAction() eatThisAction() Insertdelete() create() viewDidLoad() mainDBInfo()
FitU	**Update Food Page(UpdateFoodViewController.swift): Allows user to edit an existing food in their personal list**	viewDidLoad() inputTaggerIntoClass2() inputTempUserFoods2() updateButton() mainDBInfo()
FitU	**Confirm Food Deletion Page(ConfirmDeletionViewController.swift): Allows user to confirm they want to delete food from their personal list**	viewDidLoad() inputTaggerIntoClass2() yesButton() noButton()mainDBInfo()
FitU	**Exercise Page(ExerciseViewController.swift): Allows users to view exercises with personalized sets and reps data based on goals. Also, allows user to add exercise to their daily list by clicking on it**	viewDidLoad() inputRememberedEmailForExerciseVC() viewPersonalWorkoutList() inputAccountsForExerciseVC() openPersonalExerciseList() insertIntoUserList() makeClickableItems() mainDBInfo()
FitU	**Create Workout Page(CreateWorkoutViewController.swift): Allows user to create their own workouts for their personal list**	viewDidLoad() createWorkoutButton() mainDBInfo()

FitU	**User Exercise Page(userExerciseViewController.swift): Dynamically displays user's personal exercise list from database table**	mainDBInfo() viewDidLoad()
FitU	**Update Exercise Page(UpdateExerciseViewController.swift): Allows user to edit an existing exercise in their personal list**	viewDidLoad() exerciseUpdateButton() mainDBInfo()
FitU	**Confirm Exercise Deletion Page(ConfirmExerciseDeletionViewController.swift): Allows user to confirm they want to delete exercise from their personal list**	viewDidLoad() inputTaggerintoClass4() displayUserExercisePage() yesExerciseButton() noExerciseButton() mainDBInfo()
FitU	**Mindset PageMindset(ViewController.swift): gives users the tips/ motivation they need to stay one track of their fitness journey.**	viewDidLoad() findfitnessDataIndexForMindset() inputRememberedEmailForMindset() inputAccountsForMindset() getQuote() mainDBInfo()
FitU	**Welcome Splash Page(SplashViewController.swift): Splash page to welcome user to app**	viewDidLoad() getStartedButton() mainDBInfo()
FitU	**Login Page(LoginViewController.swift): Allows user to login, create new account, or reset forgotten password**	viewDidLoad() loggingIn() inputRememberedEmail() createAccountButton() forgotPasswordButton() loginButton() inputAccounts() findfitnessData2Index() mainDBInfo()
FitU	**Forgot Password Page(ForgotPasswordViewController.swift): Allows user to verify account info to reset password**	viewDidLoad() verifyAccountInfo() inputAccounts2() textValidatorEmail() findAccountButton() updatePasswordButton() mainDBInfo()
FitU	**Create Account Page(CreateAccountViewController.swift): Allows user to create new account and enter fitness data**	viewDidLoad() inputAccountsForCreation() findIfExists() textValidatorEmail() continueButton() createButtonClick() mainDBInfo()

```
324         if(((ft!.isInt2)==true) && ((inch!.isInt2)==true)){
325             print("No errors found")
326             feet = (ft as! NSString).doubleValue  ⚠ Forced cast from 'String?' to 'NSStrin...
327             inches = (inch as! NSString).doubleValue  ⚠ Forced cast from 'String?' to 'NS..
328             var weight = Int(floor(weightSliderO.value))  ⚠ Variable 'weight' was neve...
329             var age = Int(floor(ageSliderO.value))  ⚠ Variable 'age' was never mutated;...
330             var height = ((feet * 12) + inches) * 2.54  ⚠ Variable 'height' was never m...
331             if(genderClicker.selectedSegmentIndex == 0){
332                 gender = "Female"
333             }else if(genderClicker.selectedSegmentIndex == 1){
334                 gender = "Male"
335             }
336
337             activity = activityPicker.selectedRow(inComponent: 0).description
338             print("Activity: \(activity)")
339             if(goalClicker.selectedSegmentIndex == 0){
340                 goals = "Lose"
341             }else if(goalClicker.selectedSegmentIndex == 1){
342                 goals = "Keep"
343             }else if(goalClicker.selectedSegmentIndex == 2){
344                 goals = "Gain"
345             }
346             print("Goal: \(goals)")
347             var statement3: OpaquePointer?
348             let insertStatement4 = "INSERT into authorData3 (fname, lname,  2 ⚠
                    username, password, height, weight, age, gender, activity,
                    goal) values ('\(fn as! NSString)', '\(ln as! NSString)',
                    '\(email as! NSString)', '\(pass as!
                    NSString)','\(height)', '\(weight)', '\(age)', '\(gender
                    as! NSString)', '\(activity as! NSString)', '\(goals as!
                    NSString)');"
349             sqlite3_prepare_v2(db, insertStatement4, -1, &statement3, nil)
```

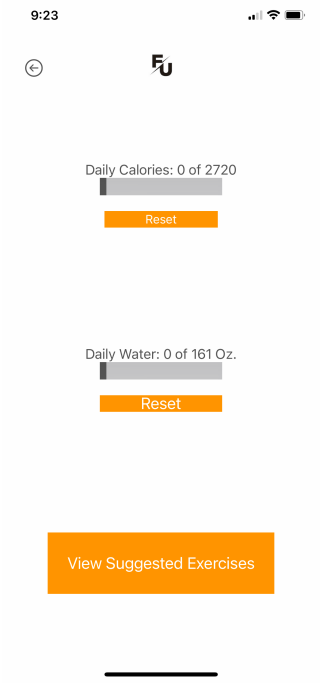
Function that allows user to create account, data is grabbed from text boxes, runs through error handling and is inserting into database tables





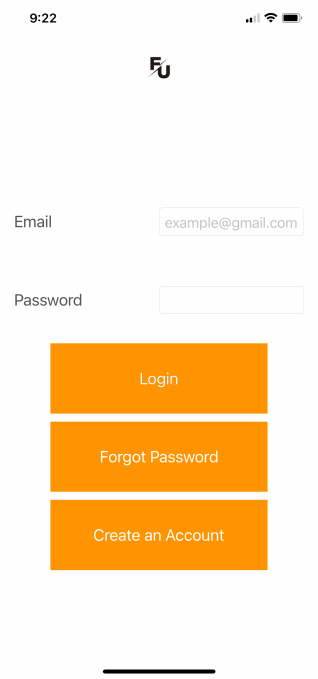
Function that grabs users fitness data, as well as daily calories/water intake from database tables and displays them in GUI by showing progress percentage

```
138 func getProgresses(){
139     inputfitnessData2(fd: fitnessData2Obj)
140     dailyCalorieLabel.text = "Daily Calories:
            \(String(fitnessData2Obj.yourBmr2[0])) of
            \(String(fitnessData2Obj.bmr2[0]))"
141     dailyWaterLabel.text = "Daily Water:
            \(String(fitnessData2Obj.yourWater2[0])) of
            \(String(fitnessData2Obj.water2[0])) Oz."
142     var top1 = Double(fitnessData2Obj.yourBmr2[0])  ⚠ Variable 'top1' was never mut...
143     var bottom1 = Double(fitnessData2Obj.bmr2[0])  ⚠ Variable 'bottom1' was never m..
144     var dailyCalorieProgress = Float(top1/bottom1)  ⚠ Variable 'dailyCalorieProgress..
145     var top2 = Double(fitnessData2Obj.yourWater2[0])  ⚠ Variable 'top2' was never...
146     var bottom2 = Double(fitnessData2Obj.water2[0])  ⚠ Variable 'bottom2' was neve...
147     var dailyWaterProgress = Float(top2/bottom2)  ⚠ Variable 'dailyWaterProgress' w...
148
149     if(dailyCalorieProgress <= 0.01 || Int(fitnessData2Obj.yourBmr2[0]) == 0){
150         calorieProgress.progress = 0.01
151     }else{
152         calorieProgress.progress = dailyCalorieProgress
153     }
154
155     if(dailyWaterProgress <= 0.01 || Int(fitnessData2Obj.yourWater2[0]) == 0){
156         waterProgress.progress = 0.01
157     }else{
158         waterProgress.progress = dailyWaterProgress
159     }
160 }
161
```



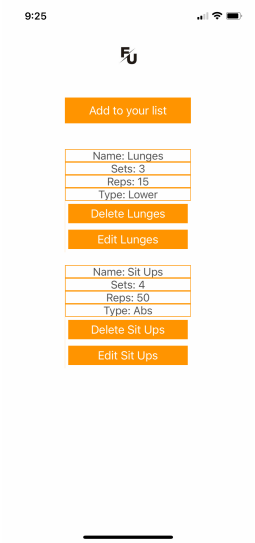
```
func loggingIn(people2: Person)->Bool{
    var k = 0
    print("Class Counter: \(people2.email2.count)")
    while(k < people2.email2.count){
        if(people2.email2[k] == loginEmailField.text && people2.password2[k]
            == LoginpasswordField.text){
            UserDefaults.standard.set(true, forKey: "isLoggedIn16")
            UserDefaults.standard.synchronize()

            //DELETE PERVIOUS data from table
            let deleteStatement1 = "Delete from rememberedEmail where id >=
                0;"
            var ds1: OpaquePointer?
            sqlite3_prepare_v2(db, deleteStatement1, -1, &ds1, nil)
            if sqlite3_step(ds1) == SQLITE_DONE{
                print("delete all")
            }else{
                print("not deleted")
            }
            sqlite3_finalize(ds1)

            //insert email into table
            var statement5: OpaquePointer?
            let insertStatement4 = "INSERT into rememberedEmail (email)
                values ('\(loginEmailField.text as! NSString)');"
            sqlite3_prepare_v2(db, insertStatement4, -1, &statement5, nil)
            if sqlite3_step(statement5) == SQLITE_DONE{
                print("inserted into rememberedEmail table")
            }else{
                print("not inserted into rememberedEmail table")
            }
            sqlite3_finalize(statement5)

            inputRememberedEmail(rm: rememberEmailObj)
            var index = findfitnessData2Index()  Variable 'index' was never mutate...
            var bmr:Double = 0.0
            var w = Double(people.weight2[index])  Variable 'w' was never mutated..
            var h = Double(people.height2[index])  Variable 'h' was never mutated...
            var a = Double(people.age2[index])  Variable 'a' was never mutated; co...
            var ac = people.activity2[index]  Variable 'ac' was never mutated; cons..
            bmr = (10 * (w * 0.453592)) + (6.26 * h) - (5 * a)

            let deleteStatement10 = "DELETE from fitnessData2 where id >= 0;"
            var ds10: OpaquePointer?
            sqlite3_prepare_v2(db, deleteStatement10, -1, &ds10, nil)
            if sqlite3_step(ds10) == SQLITE_DONE{
                print("delete all fitnessData2")
            }else{
                print("not deleted all fitnessData2")
            }
            sqlite3_finalize(ds10)

            var statement14: OpaquePointer?
            let insertStatement14 = "INSERT into fitnessData2 (bmr, water,
                yourBmr, yourWater) values ('\(bmr)', '\(water)', '0', '0');"
            sqlite3_prepare_v2(db, insertStatement14, -1, &statement14, nil)
            if sqlite3_step(statement14) == SQLITE_DONE{
                print("inserted into fitnessData2 table")
            }else{
                print("not inserted into fitnessData2 table")
            }
            return true
        }else{
            k = k+1
        }
    }
    return false
}
```

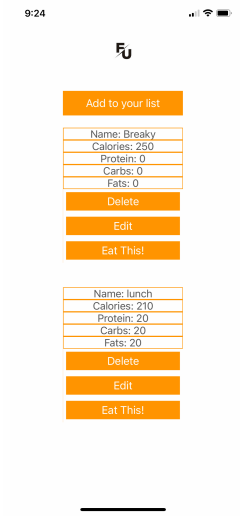
Function that determines if user login is failed or successful, and will then remember the users login status for future use, will store users email to remember their data, and store fitness data in a table for future use



```
167     func createRecords(uf: userFood){
168         var yPos =  CGFloat(80)
169         let xPos = CGFloat(self.view.frame.width)
170         var g = 0
171
172         createButton.setTitle("Add to your list", for: .normal)
173         createButton.backgroundColor = .systemOrange
174         createButton.addTarget(self,action: #selector(create),for: .touchUpInside)
175         createButton.frame = CGRect( x:(xPos/4), y:20, width:(xPos/2), height: 40)
176         subStack.isUserInteractionEnabled = true
177         subStack.addSubview(createButton)
178
179             while(g < uf.ID2.count){
180                 let nameLabel = UILabel()
181                 nameLabel.text = "Name: " + uf.name2[g]
182                 nameLabel.textAlignment = .center
183                 nameLabel.backgroundColor = .white
184                 nameLabel.textColor = .darkGray
185                 nameLabel.frame = CGRect( x:-xPos/4, y:yPos, width:(xPos/2),
                        height: 20)
186                 nameLabel.layer.borderWidth = 1
187                 nameLabel.layer.borderColor = UIColor.systemOrange.cgColor
188                 yPos += 20
189
190                 let caloriesLabel = UILabel()
191                 caloriesLabel.text = "Calories: \(uf.calories2[g])"
192                 caloriesLabel.textAlignment = .center
193                 caloriesLabel.backgroundColor = .white
194                 caloriesLabel.textColor = .darkGray
195                 caloriesLabel.layer.borderWidth = 1
196                 caloriesLabel.layer.borderColor = UIColor.systemOrange.cgColor
197                 caloriesLabel.frame = CGRect( x:-xPos/4, y:yPos, width:(xPos/2),
                        height: 20)
198                 yPos += 20
239         //creates dynamic buttons
240         yPos = CGFloat(180)
241         var t = -1
242         if(userFoodObj.ID2.count != 0){
243             for index in (userFoodObj.ID2){  Immutable value 'index' was never used; con...
244                 t+=1
245                 var oneBtn2: UIButton = {  Variable 'oneBtn2' was never mutated; conside...
246                     let myButton = UIButton()
247                     myButton.addTarget(self,action: #selector(deleteAction),for:
                            .touchUpInside)
248                     myButton.setTitle("Delete", for: .normal)
249                     myButton.setTitleColor(.red, for: .highlighted)
250                     myButton.frame = CGRect( x:(xPos/4), y:yPos, width:(xPos/2),
                            height: 40)
251                     myButton.tag = Int(userFoodObj.ID2[t])
252                     myButton.backgroundColor = .systemOrange
253                     myButton.layer.borderWidth = 5
254                     myButton.layer.borderColor = UIColor.white.cgColor
255                     yPos += 260
256                     myButton.isUserInteractionEnabled = true
257                     return myButton
258                 }()
259                 subStack.isUserInteractionEnabled = true
260                 subStack.addSubview(oneBtn2)
261             }
262         }
263
```

Function reads data from user exercise and user food tables and dynamically creates element (labels, buttons, etc.) to display the data in a GUI. These dynamic buttons allow user to interact with each data record by deleting, editing, or adding to their list.

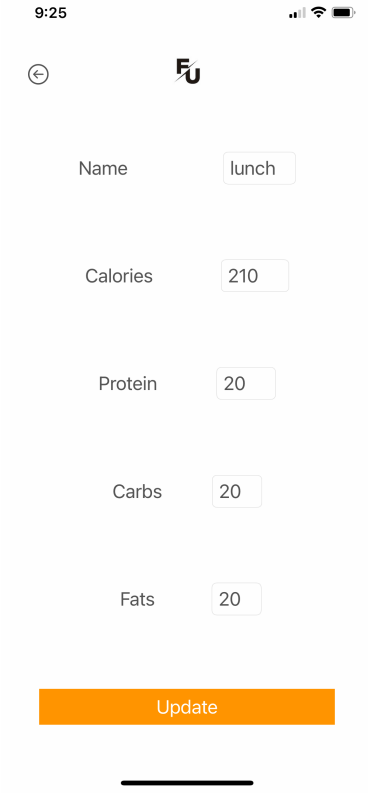




```
@IBAction func createWorkoutButton(_ sender: Any) {

    var name = nameLabel.text?.trimmingCharacters(in:
        .whitespacesAndNewlines)
    var sets2 = setsStepper0.value
    var reps2 = repsStepper0.value
    var type2 = typePicker.selectedRow(inComponent: 0).description
    print("Type: \(type2)")

    if(name?.isEmpty)!{
        print("Workout Name is empty")
        return
    }

    var statement4: OpaquePointer?
    let insertStatement4 = "INSERT into userExerciseTable2 (name,
        sets, reps, type) values ('\(name as! NSString)', '\(sets2)',
        '\(reps2)', '\(type2  as! NSString)');"
    sqlite3_prepare_v2(db, insertStatement4, -1, &statement4, nil)
    if sqlite3_step(statement4) == SQLITE_DONE{
        print("inserted into user exercise table")
    }else{
        print("not inserted into user exercise table")
    }

    sqlite3_finalize(statement4)
```

Variable 'sets2' was never mutated; consider chang...
Variable 'reps2' was never mutated; consider chan...
Variable...
2

Function grabs user enters data and runs through error handling to create an exercise for user personal list. This is done by inserting into database table that will be display on user exercise page. Same process is done when user creates a food.



- Software
    - Did you learn any computer languages?
        - In the process of creating this project I learned to program in SwiftUi (an extension of Objective-C) as well as SQLite
    - How long did it take you to learn?
        - The entire process took 2.5 months to learn. I was constantly learning and absorbing information throughout the project
    - What are the difficulties you have encountered?
        - The biggest challenge was keeping track of variables such as daily calories/water intake across different times and different pages. For example if user closes app, the app should bar able to remember their progress so far and re-display when user opens app again. To accomplish this I used algorithms to find daily calories/water intake and inserted into a database table in order for the app to remember this data no matter what.
        - I learned that the download of locally hosing the database tables from SQLite is that the only way to edit global app data is directly from Xcode rather than querying a server. This can be fixed by paying for a cloud database service such as Google SQL.
        - I also must pay a yearly membership fee to publish on the app store, so I want to make sure all of the fitness research is done before fully publishing
        - Dynamically creating elements based on table records was quite challenging because I had to find a way to give each dynamic element a tag for when the user want to edit, or delete that specific record. To accomplish this I set each dynamic tag equal to the ID of each specific table record

6. Test and Integration

The application code will be written in swift /objective-C, and will be validated using Xcode's built in debugger/validator. The display/outcome of the app will be tested on smartphone and tablet screens.

The application will:

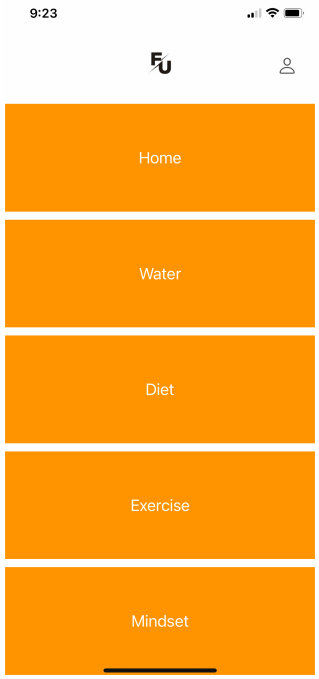
- Have user log in and create a new workout and food to add to their own plans. Users will enter in personal data to create personalized plans and show progress on home page (water/food intake).

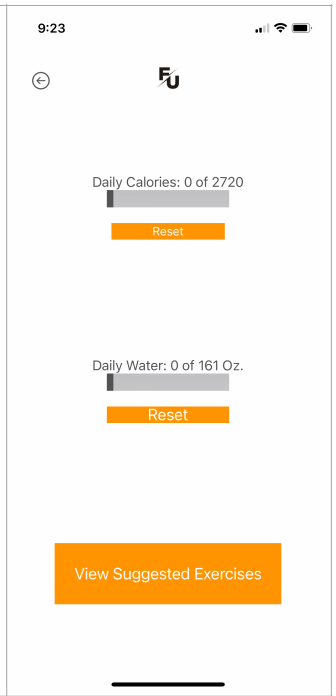
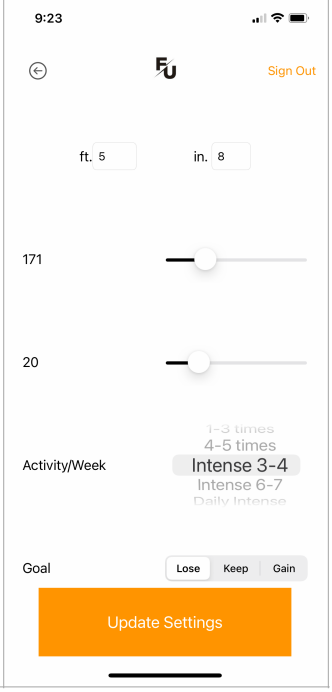
To test the final application the following steps were taken to test all features.

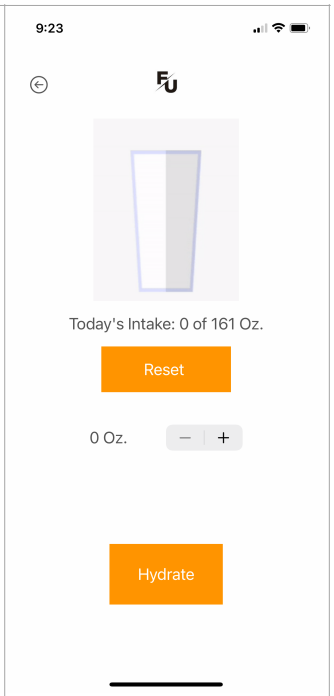
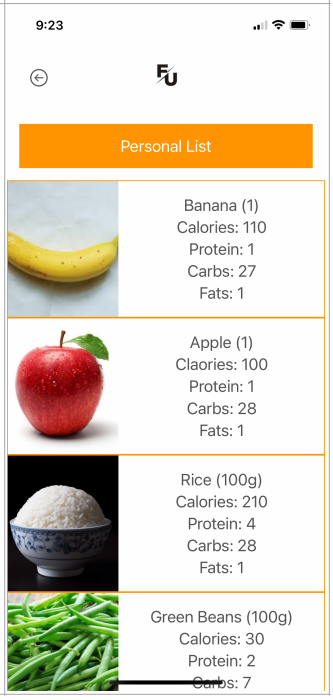
Test Case Instructions:

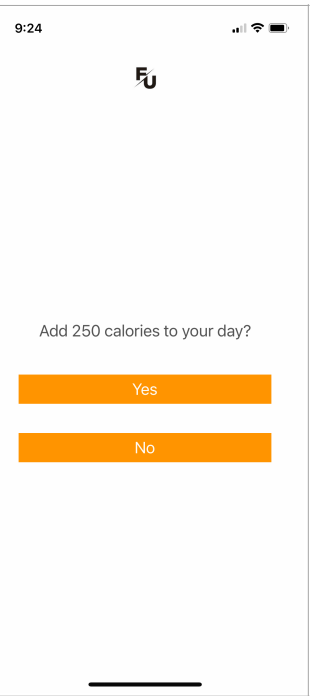
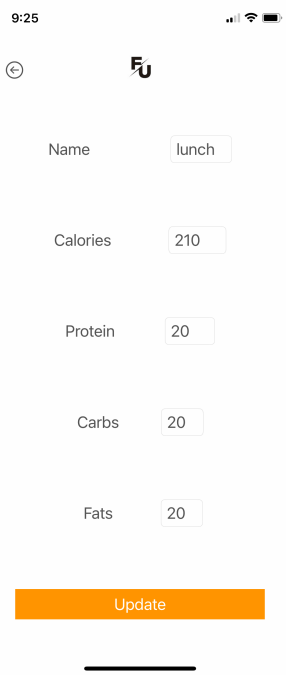
- Create user account (test error handling)
- Reset Password
- Login
- Update personal Settings
- Add to daily water intake
- Add to daily calories Intake
- Create own food for personal list (test error handling)
- Delete food from personal List
- Edit existing food from personal list (test error handling)
- Create own exercise for personal list (test error handling)
- Add exercise to list from suggested exercises page
- Delete exercise from personal list
- Edit existing exercise from personal list (test error handling)
- Test Mindset page timer function is changing quotes every 5 seconds

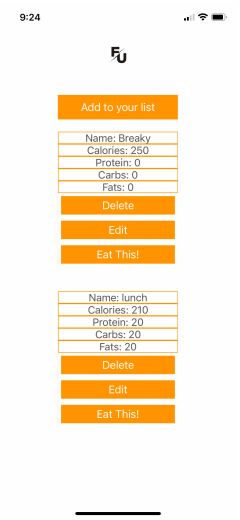
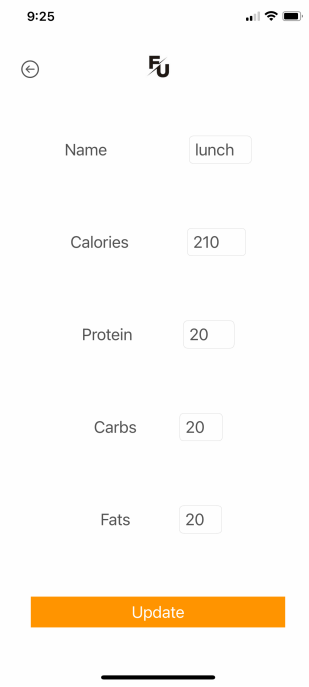
TABLE Test plan and results - Verification Matrix:

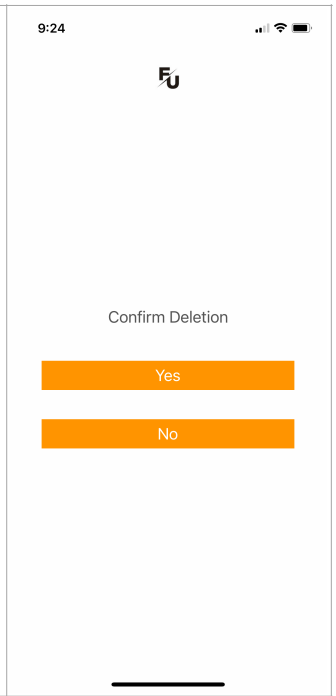
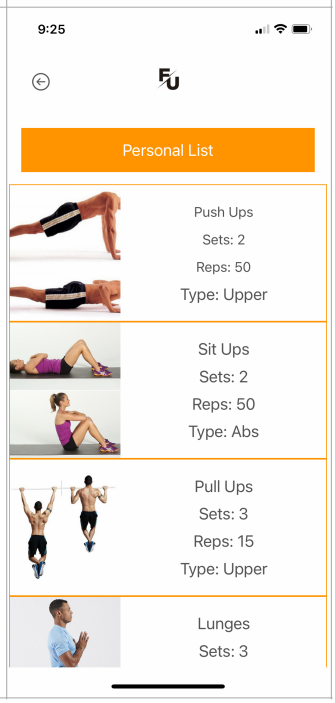
Page #/name – list all pages	**Test item** **- list hyperlinks, fields, buttons, videos, images,** **GUI controls, web pages** **Schema: select action => reaction Legend:** **=> land in**	**Validation of each app page using Xcode 12 built-in debugger/compiler**	**Outcome** in **Mobile Phone** **- iPhone 11 Pro**
Main Page	Profile Button => Takes you to profile page to update settings or sign out Home Button => Takes you to home page Water Button => Takes you to home page Diet Button => Takes you to home page Exercise Button => Takes you to home page Mindset Button => Takes you to home page	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

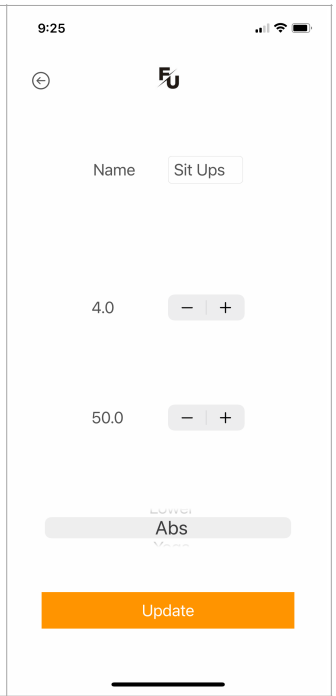
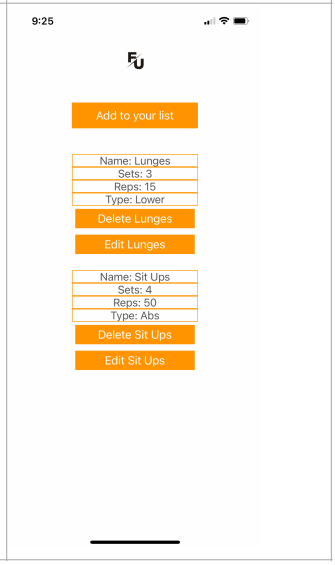
Home Page	Logo => Takes you to main page resetWaterButton=>Resets daily water intake resetsButton=>Resets daily calorie intake viewSuggestedExercisesButton=>Opens exercise page Displays all current progress	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Profile Page	Logo => Takes you to main page Allows user to edit all fitness data they entered when creating account All field work properly and the same as when creating account	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

Water Page	Logo => Takes you to main page ouncesCounter=>Grabs ounces value properly resetButton=>Resets daily water intake hydrateButton=>adds to water intake Accepts necessary inputs to display recommended daily water intake	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Diet Page	Accurately displays foods and necessary data such as calories, protein, carbs and fats Logo => Takes you to main page Each Food=>Allows suer to add corresponding food to their daily food intake PersonalListButton=>Allows user to view personal list Accurately displays workouts and necessary data such as sets, reps, and difficulty	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

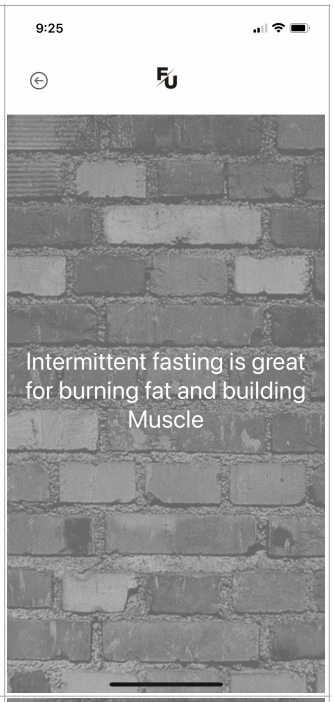
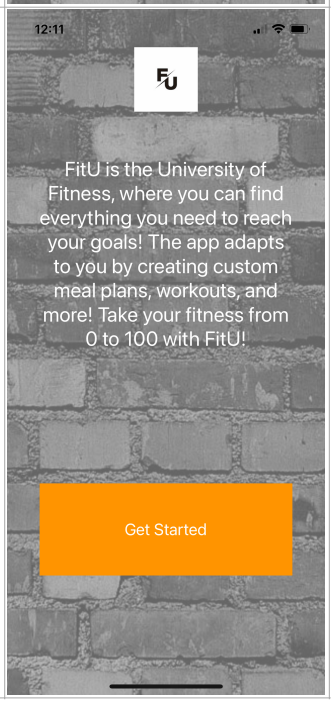
Confirm Food Addition Page	Logo => Takes you to main page yesButton=>Adds food to daily calorie intake noButton=>Returns user to diet page	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Create Food Page	Logo => Takes you to main page Name field => inputs user text properly Name field => inputs user text properly Calories field =>inputs user text properly Protein field =>inputs user text properly Carbs field =>inputs user text properly Fats field =>inputs user text properly CreateButton=>Properly creates record in table	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

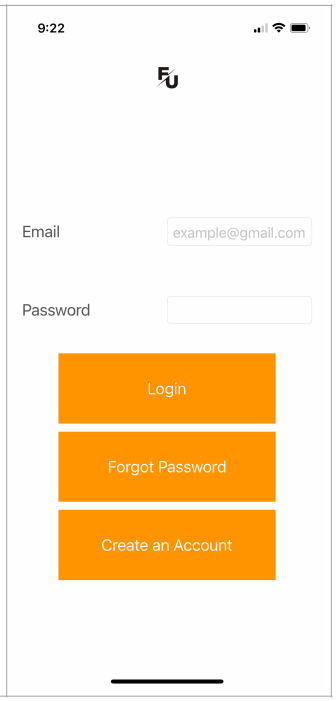
User Food Page	Displays all records properly Logo => Takes you to main page Delete Buttons=>Properly distinguishes record to delete Edit Buttons=>Properly distinguishes record to edit Eat Buttons=>Properly distinguishes record to add to daily calorie intake addButton=>Allows user to create food to add to their list	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Update Food Page	Properly displays pervious data and updates records Logo => Takes you to main page Name field => inputs user text properly Calories field =>inputs user text properly Protein field =>inputs user text properly Carbs field =>inputs user text properly Fats field =>inputs user text properly UpdateButton=>Properly updates record in table	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

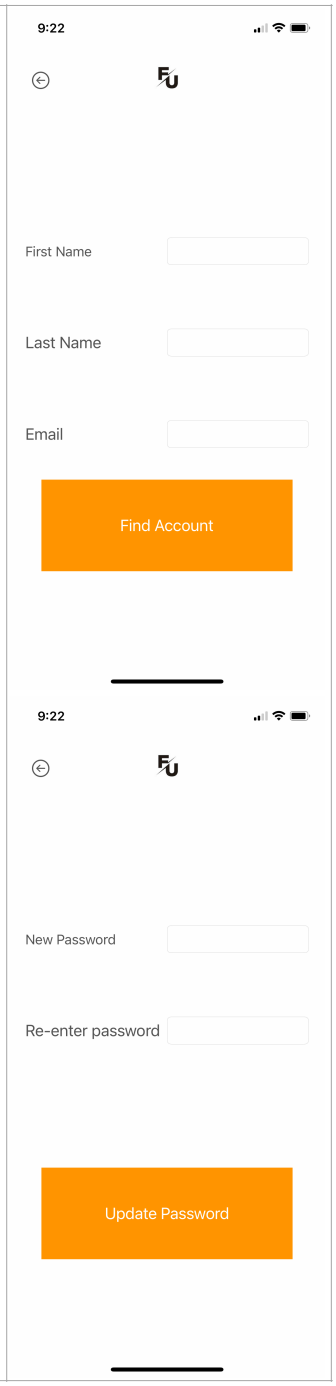
Confirm Food Deletion Page	Logo => Takes you to main page yesButton=>Deletes food from personal list noButton=>goes back to personal food page	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Exercise Page	Logo => Takes you to main page Each Exercise=>Allows suer to add corresponding exercise to their personal list PersonalListButton=>Allows user to view personal list Accurately displays workouts and necessary data such as sets, reps, and difficulty	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

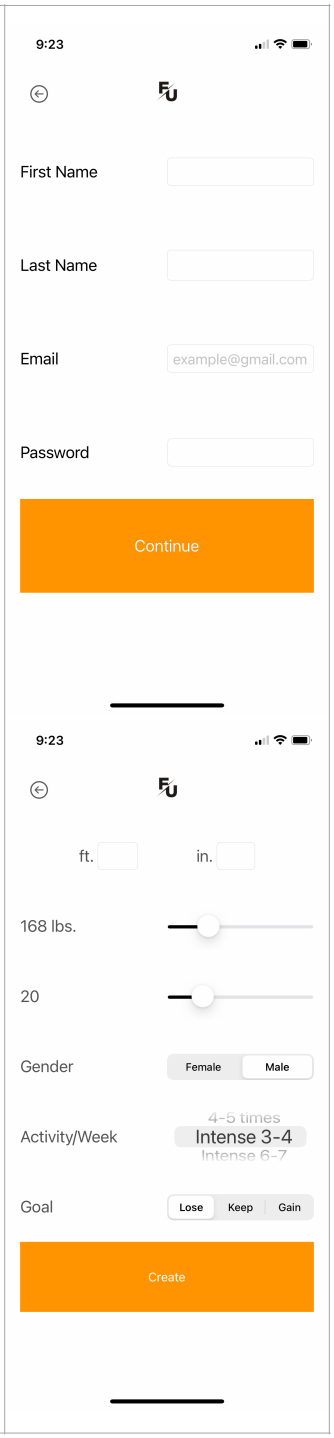
Create Exercise Page	Logo => Takes you to main page Name field => inputs user text properly Set Counter=>Grabs set value properly Rep Counter=>Grabs rep value properly Type selector=>Grabs exercise type properly CreateButton=>Properly creates record in table	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
User Exercise Page	Displays all records properly Logo => Takes you to main page Delete Buttons=>Properly distinguishes record to delete Edit Buttons=>Properly distinguishes record to edit addButton=>Allows user to create exercise to add to their list	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

Update Exercise Page	Properly displays pervious data and updates records Logo => Takes you to main page Name field => inputs user text properly Set Counter=>Grabs set value properly Rep Counter=>Grabs rep value properly Type selector=>Grabs exercise type properly UpdateButton=>Properly updates record in table	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Confirm Exercise Deletion Page	Logo => Takes you to main page yesButton=>Deletes exercise from personal list noButton=>goes back to personal exercise page	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

Mindset Page	Logo => Takes you to main page Displays randomly generated motivational quote	**Validation of each app page using Xcode 12 built-in debugger/compiler**	
Splash Page	Get Started Button => Takes you to login page	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

Login Page	Email Field => inputs user text properly Password Field => inputs user text properly Allows user user to log in or sign up	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

Forgot Password Page	Logo => Takes you to login page First Name Field => inputs user text properly Last Name Field => inputs user text properly Email Field => inputs user text properly Find Account Button=> Verifies user info and determines if user can reset password or not Password field 1 => inputs user text properly Password field 1 => if matches password matches field 1, then password is reset	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

Create Account Page	Logo => Takes you to login page First Name Field => inputs user text properly Last Name Field => inputs user text properly Email Field => inputs user text properly Password Field => inputs user text properly Continue Button=>If passes error handling will allow user to enter fitness data Height Fields => inputs user text properly Weight slider=> grabs user weight properly age slider=> grabs user age properly Gender button=> grabs user gender properly Weight slider=> grabs user weight properly Activity selector=> grabs user activity level properly Goal Button=> grabs user goal properly	**Validation of each app page using Xcode 12 built-in debugger/compiler**	

7. Deployment

- Connect to cloud database fro remote access to make entire app data dynamically accessible
- Pay for apple developer membership to publish on app store
- Market app to public to attract users
- Develop paid features to generate income from app
- Integrate ads to develop income from app
- Work on bug fixes and app updates/maintenance for the future

8. Conclusion

Discussion: iOS app development has now become a true passion of mine and I'm glad I was able to partake in such a project because a lot of the time we as students and developers are hesitant to start something because it is new or difficult, but this is exactly why I enjoyed this project so much, because I feel that I can more confidently overcome such obstacles in the future and attack new projects head on.

1. ***Summarize your project achievements.***
   - My app was able to successfully gather data across all aspects of fitness and put it together into a clean, simple design for the user.
   - In completing this project I learned how to use SwiftUI, and SQLite to make connections between the GUI and the actual code in order to create powerful app features
   - Next, some of the limitations of this project is that I have to pay a yearly subscription to publish the app and there's also no way to globally edit data unless you do it directly from Xcode or you connect to a cloud database such as google cloud, so hopefully in the future i can pay for that and really make everything dynamic.
   - In the process of creating this app I had to remember to keep my audience in mind, their interests, and what they value from an app like this in order for it to be successful.
   - I believe that the work I have done here is truly special and proves that I can hold my own in this industry. This project has taken my career the next level, and I'm confident that I will continue to grow and become a better programmer because of it.
2. One of the main things I learned in this long term project is to break things up into iced because all together it would just be way too overwhelming. I also learned that programming is universal across all languages, there's a certain skill set you must possess, and you have to understand that the first of everything you make will suck. In my case this was an app, but you can make a great app until you make your fist one.The first step you take is always the hardest in a journey like

this but it's one I'm glad I took. While this project is not fully published yet I'm confident that my idea can be special and loved by many worldwide.

3. If I was to do this project over again I would start my research a lot earlier. Researching how to program in Swift was a long process in order to get familiar with the syntax.
4. For future enhancement I would like to generate graphs and animations based on the user's tracked daily, create sleep features, and integrate a feature where users can connect with fitness trainers like myself through the app.
5. Course that helped me in the development of this project were: C#, Publishing o the Web, Data Structure, and Internet Apps

9. References

CodeWithChris. (2017, October 10). How to Make an App for Beginners (2018) - Lesson 1-11 [Video]. YouTube. https://youtu.be/YIZxSZJMU2Y

10. Appendices

- Appendix A: Final Report
- Appendix B: Project Code
- Appendix C: Presentation