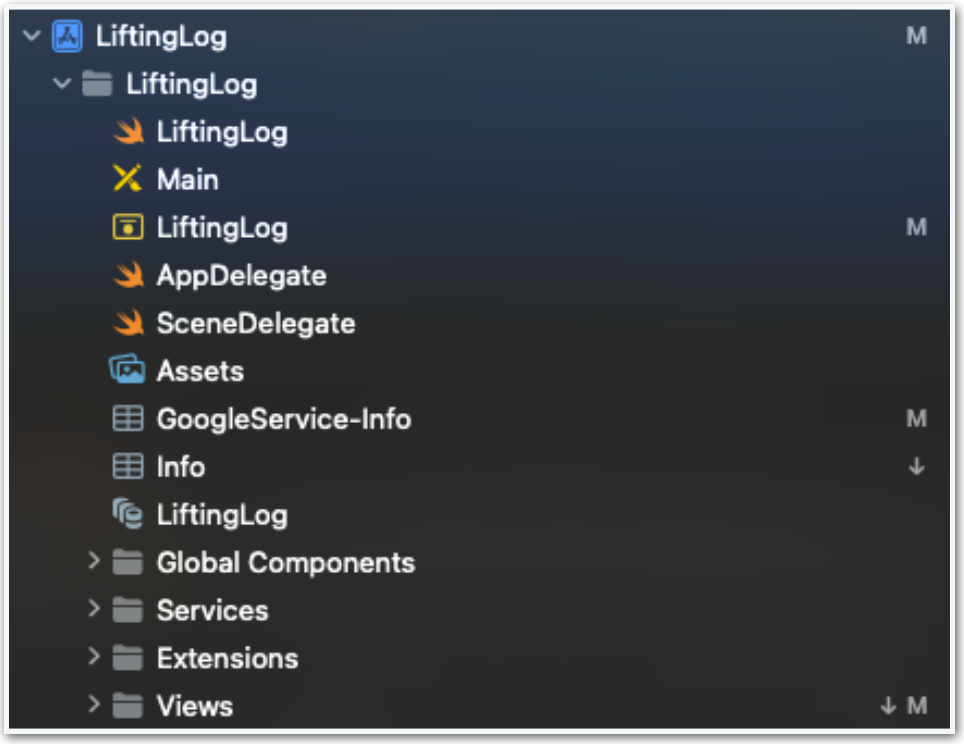
Pump Path Developer Manual

Neil Kalanish and Joe Melito

Table of Contents

About the App



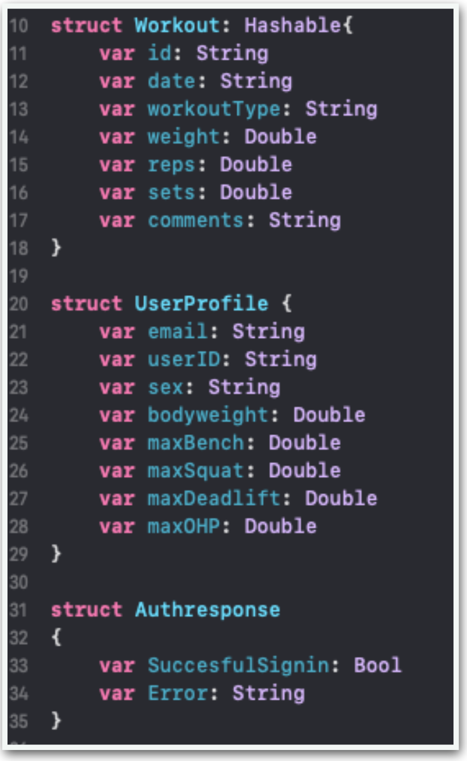
The file structure is divided into four different folders: Global Components, Services, Views, and Extensions. The Global Components section contains all the variables used throughout the app. Services handle all the “work” being done by the app, from database calls to authentication and calculations. Most troubleshooting will be done in the Services section. Extensions manage all the user prompts and notifications. Lastly, Views comprise all the UI views that the user sees and interacts with.

Pump Path uses two different backend services. The Firebase Authentication service is used for creating user profiles, resetting passwords, and authenticating the user when they log in. Pump Path also supports Apple ID login, and all related processes are managed by Apple.

The second backend service used by Pump Path is CloudKit. CloudKit serves as the database that stores all user profiles, workouts, and user IDs. The application uses the user ID to store, retrieve, update, and delete workouts. If the user logs in with an Apple ID, we are given the user ID every time they log in. However, if the user logs in using Firebase, we find the user ID based on their email.

Global Components

AllStructures



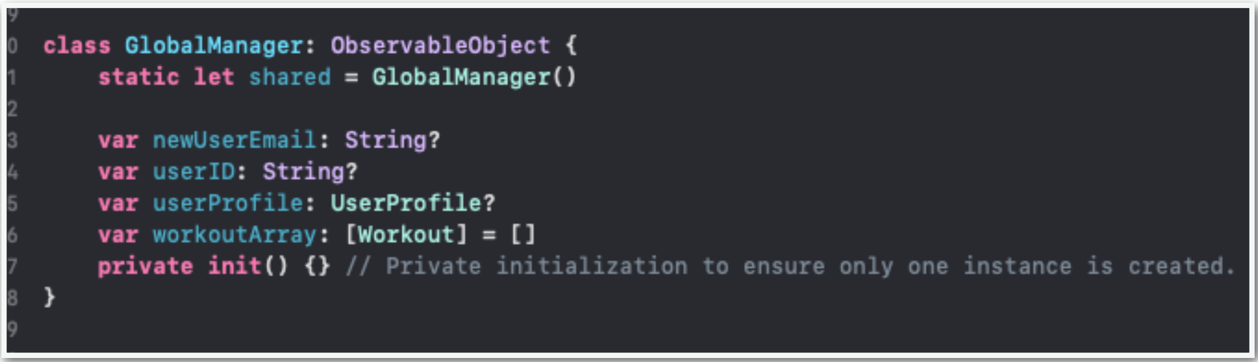
```
struct Workout: Hashable{
    var id: String
    var date: String
    var workoutType: String
    var weight: Double
    var reps: Double
    var sets: Double
    var comments: String
}

struct UserProfile {
    var email: String
    var userID: String
    var sex: String
    var bodyweight: Double
    var maxBench: Double
    var maxSquat: Double
    var maxDeadlift: Double
    var maxOHP: Double
}

struct Authresponse
{
    var SuccesfulSignin: Bool
    var Error: String
}
```

The “AllStructures” file contains three different structures that are used throughout the app. The first one is “Workout,” which includes details such as workouts, date, workout type, weight, reps, sets, and comments. Whenever a user saves a workout, it is sent to a function to transform the data into this structure. The ID links the workout to the user.

The second structure is “UserProfile,” which holds the user's email, user ID, security information, body weight, maximum bench press, maximum squat, maximum deadlift, and maximum overhead press (OHP). This information is required for the app, so when a new user signs up, they are immediately prompted to input this information to set up the app as soon as they log in. When the user logs in, this information is immediately retrieved from the database and stored in the app to allow for quick updates. All of this information can be updated on the user profile screen.

Lastly, the “AuthResponse” structure is used when logging in with a username and password. When the Firebase authentication services check the username and password, it returns these values—either true for a successful login or false along with an error message to inform the user of what happened.

GlobalVariables



```swift
class GlobalManager: ObservableObject {
    static let shared = GlobalManager()

    var newUserEmail: String?
    var userID: String?
    var userProfile: UserProfile?
    var workoutArray: [Workout] = []
    private init() {} // Private initialization to ensure only one instance is created.
}
```

The GlobalVariables folder contains all the variables used throughout the app. When users log in, we immediately query the database for their profile and all their workouts. All workouts are added to the workoutArray, and their user profile is updated. Additionally, we store the userId in a separate string in case we need it before the user profile is fully developed.

The app is set up in such a way that all information is read locally, which allows for quick responses and ensures that the user never has to endure a loading screen. All loading processes occur upon login, making the rest of the application's operation seamless. Whenever a user makes a change, such as adding a workout or updating their profile, we first update the variable locally so the user sees the change instantly. Then, we proceed to make the database call in the background. This approach keeps the database operations unnoticeable to users.Services

Services

CalculationService

The Calculation Service contains several functions that are used throughout the app. It includes two different functions to calculate the Wilks total for males and females, and it also has a calculator for determining the total weight lifted, measured in points and kilos. Lastly, at the bottom of the file, all the warm-ups for an exercise are calculated. Warm-up and lift calculations are in separate functions, making it easier to implement any mathematical changes required for any of the lifts.

DatabaseService

The Database Service handles all calls to the database. The first two functions are used for saving and loading a workout. Loading is only called once when the user first logs in. The "save workout" is triggered anytime a user enters a workout. The next function is for creating a new user. This function is activated when a new user logs in and is directed to the “UserProfileCreationView,” where we ask for all their lifts, body weight, and sex. This information is then saved in the CloudKit database.

The "load user profile" function is called whenever the user logs in to populate the global structure mentioned earlier. This happens as soon as the user logs in, ensuring that users don’t have to wait for loading when they go to the user profile screen. There is also a function to update the user profile, which is used whenever the user updates their profile and clicks the “Update Profile” button.

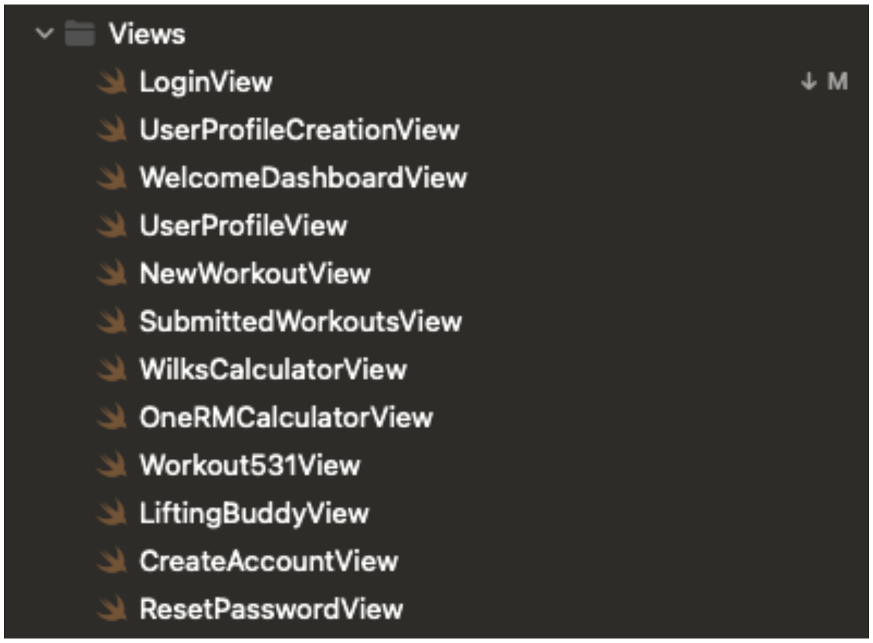
Lastly, the file includes a few helper functions that assist in formatting data or retrieving information. Among these are "create user profile," which formats the information, and "get current date," which provides the current date in a desired format so that a timestamp can be added when a user saves a workout.

AuthenticationService

The Authentication Service handles all calls to the Firebase Authentication service. This function provides a wrapper for the Firebase service. We have added some additional error handling and formatting of errors to prompt the user if there are any issues with creating an account or entering a password. Both CreateUser and LogUserIn return a boolean and an error message that is used to determine if the app should allow the user to continue.

The Authentication Service also includes several smaller helper functions such as retrieving the current user’s ID, signing the user out, and sending a reset password email. These functions are utilized throughout the app at appropriate places.

Views



Pump Path consists of twelve different views for the user to interact with. The primary view, "LoginView," does most of the work. It displays the initial welcome screen, makes most of the calls to the database service, and handles user authentication. When the user attempts to log in using "login" or Apple ID, the login function is activated. This function checks for a valid Apple ID, email, or if the user is attempting to log in with Firebase. If both text boxes for email and password are empty, then the user is logging in with Apple ID. If the email field is populated, it indicates a new user; if only the user ID is entered, it indicates an existing user. From there, we either direct the user to the new user creation screen or straight into the app.

The next view is "UserProfileCreationView." This view appears when the app identifies a new user logging in. It asks for the user's sex, bodyweight, and maximum lifts. After the user submits this information, the view saves it in the database, signs the user out, and returns them to the login screen. This view is shown only once and is accessible only by new users the first time they log in. Once they log in again, the app detects they are existing users and takes them to the welcome display.

"WelcomeDashboardView" is the main display and navigational hub of the app. All available options for the user are stored in the "welcomeDashOptions" array and are accessed whenever a user selects an option. Views can be added or removed by modifying the switch statement in the "destinationView" function.

"UserProfileView" displays all the user’s information and allows modifications to their profile. It shows their sex, bodyweight, and maximum lifts. The Wilks number and total weight lifted (in pounds and kilograms) are displayed at the bottom. These figures cannot be edited by the user and are updated in real time whenever the user updates their profile. The changes are not saved until the user clicks the “Update Profile” button, which triggers the “updateUserProfile” function to update the database.

"NewWorkoutView" offers a simple layout for entering a new workout. It requires details such as the title of the lift, weight, reps, and sets, with an optional comment. There is a cancel button and a submit button. All fields are mandatory except for the comment; omitting it prompts the user to enter one. Upon clicking “Submit Workout,” the “submitWorkout” function verifies all required fields are filled, sets up an instance of the database service, calls the “createWorkoutObject” to format the data correctly, and then saves it to the database. The workout is immediately available in the workout array for viewing without needing to query the database again.

"SubmittedWorkoutsView" displays the workouts in a card format but does not query the database as this is done at login, and any added workouts are appended to the array.

The "WilksCalculatorView" is deprecated; that information is now incorporated in the "UserProfileView."

"OneRMCalculatorView" allows lifters to quickly estimate their one-rep maximum based on their previous sets. The calculation is performed by clicking a button in the "calculateRM" function, which also verifies that the values entered are greater than one.

The 'Workout531View' is the meat and potatoes of the app. It displays all the workouts and warmups required for the 5/3/1 program. The struct 'Workout531View' holds all the arrays of information. Different workouts are calculated and assigned to variables, which then build out the array. Everything is laid out so that any math errors or necessary updates to the arrays are easy to identify. Then there is 'exerciseData,' which assigns all the arrays to the appropriate views. It involves a lot of manually coded information but is done intentionally for traceability and troubleshooting. Another important item in the view is the VStack in the body. The VStack contains all the labels that inform the user which rep set to follow. The switch statement determines which label to assign based on the week of the workout.

Currently, the 'LiftingBuddyView' section isn’t implemented, but it will be integrated into ChatGPT so users can request workout ideas or plans in real-time within the app, without needing to search the web. However, this feature has not been implemented yet.

The 'CreateAccountView' is where we ask the user for basic information to set up their account. The app requires users to have information in their profile to function, so when a new user signs up, we prompt them to immediately populate their profile.

Once the profile is created, we log them out and require them to log back in. Upon logging back in, the app will recognize the user has a profile and redirect them to the main screen.

The last view is the 'ResetPasswordView,' which only contains a text box for the user to enter their email and click submit. When the user submits, we call the Firebase Auth service to send an email with instructions on how to reset the password."

Extensions

GoogleService-Info File

The Google service file is what points the app to the Firebase service. This file is auto-generated by logging into your Firebase account, navigating to “AuthService,” and clicking “Generate Service Info.” Once generated, it needs to be added to the app, and the Firebase service should be included. You need to verify that the Bundle ID matches your app's Bundle ID and that the Project ID matches the one in Firebase. Once this file is set up and working, you shouldn’t need to change anything in it.

Signing & Capabilities

The “Signing and Capabilities” tab in Xcode contains all of the dependencies and build instructions for the apple. The “Bundle Identifier” is a unique name to identify the app. The Bundle identifier also identifies the iCloud container that all of the app information will stored in. The two must be identical otherwise the app won’t be able to reach the Database.

There is a tab. For “Frameworks, Libraries and Embedded Content” and that is an add-on the app uses. For PumpPath the only framework we use is “FirebaseAuth” to handle users who don’t have an Apple ID.