Lifting Log

Software design specification

Joe Melito and Neil Kalanish 20OCT22

Quick recap

- We are building a mobile app for bodybuilders and powerlifters who want to track their workout’s without all of the “bloat” that other workout apps offer.
- The app will be built to run on iPhone, iPad and Mac.
- It will allow users to track workouts, create workouts and view history of past workouts.
- User will also have the ability to calculate their one rep max and to find their wilks number.
- Are app is unique because we are focusing solely on lifting weights and not concerned about any other sort of workout.

Architecture

- We are going to be following a “Smart” and “Dumb” component architecture
- “Smart” components will be components and services that perform calculations, database calls and data manipulation. (Backend services)
- “Dumb” components will be components that only display the information it is given and calls the smart components to do work for it. (Front end services)

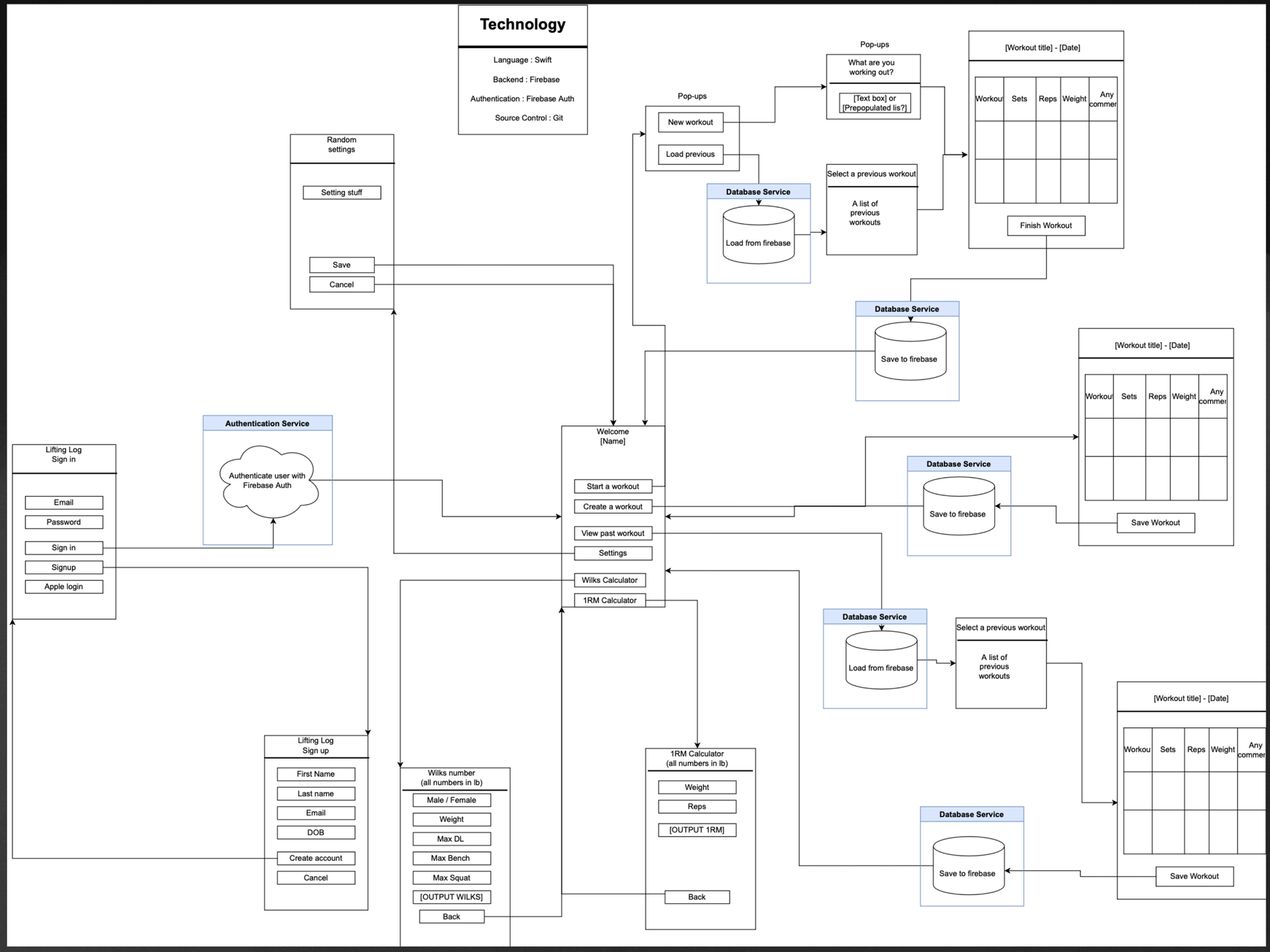
Why take this approach?

- This approach will allow us to consolidate all of our database calls and calculations to only one place.
- This will make it easier to update the calls and calculations as the application grows.
- It will allow us to make changes to the backend data structures without impacting the front end.
- Also the inverse is true, we can make changes to the UI and add views without effecting our backend services.

What tech are we using?

- We will be using the language Swift and the IDE Xcode.
  - This will allow the app to run natively on iPhone, iPad and macOS.
- For the database we are going to be using Firebase.
  - Firebase is no sql database created by Google.
  - It is a widely used DB that has a lot of support and references online.
- Lastly for authentication we will use Firebase’s built in authentication.
  - Firebase’s authentication allows your app to support users logging in with email, Facebook, Gmail, or Apple ID.

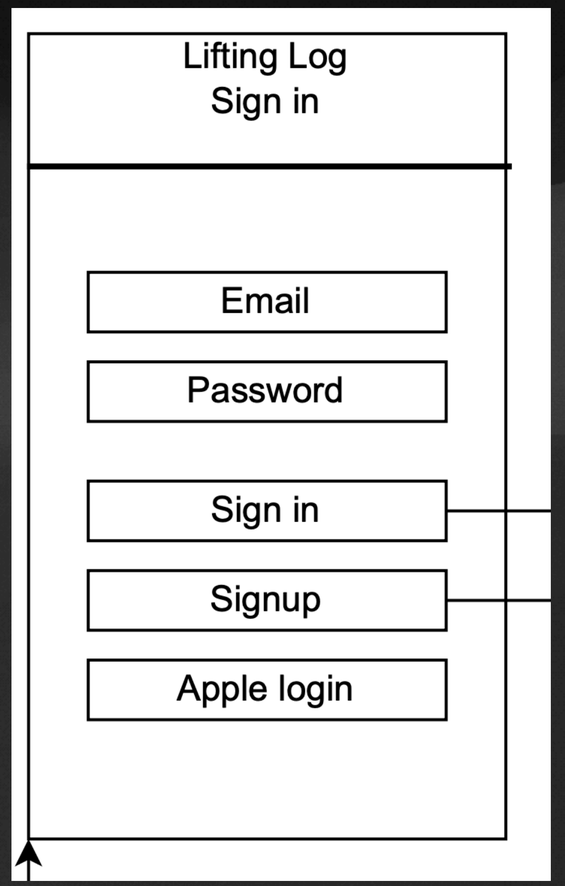
How does this all look?

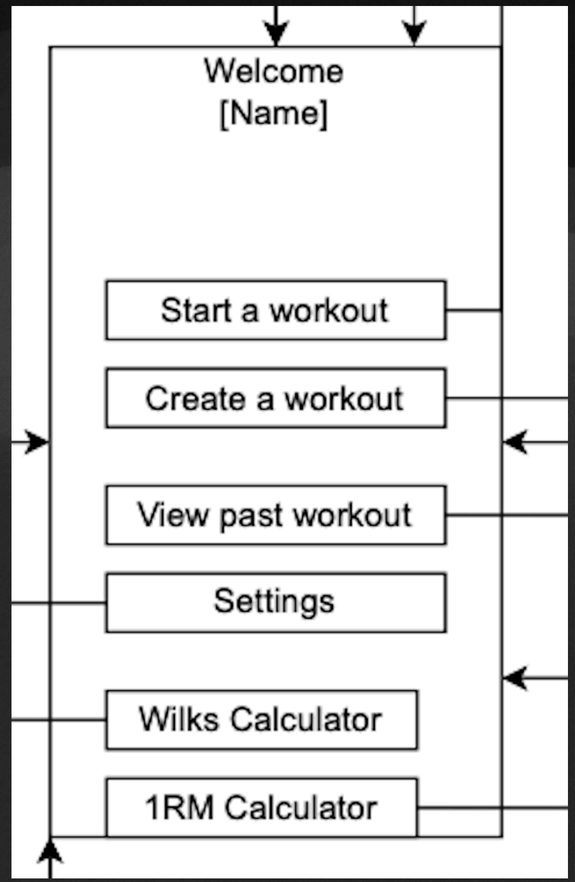


Components

Views

- All of the user interfaces (views) will be “dumb components”.
- The views will consist of buttons and textfields that will take in user data and call the “smart components”.
- For example when you type in your username and password then click “sign in” the view will call the authentication service.
- The view will never know if the information was correct only if it needs to display an error or advance to the welcome screen.

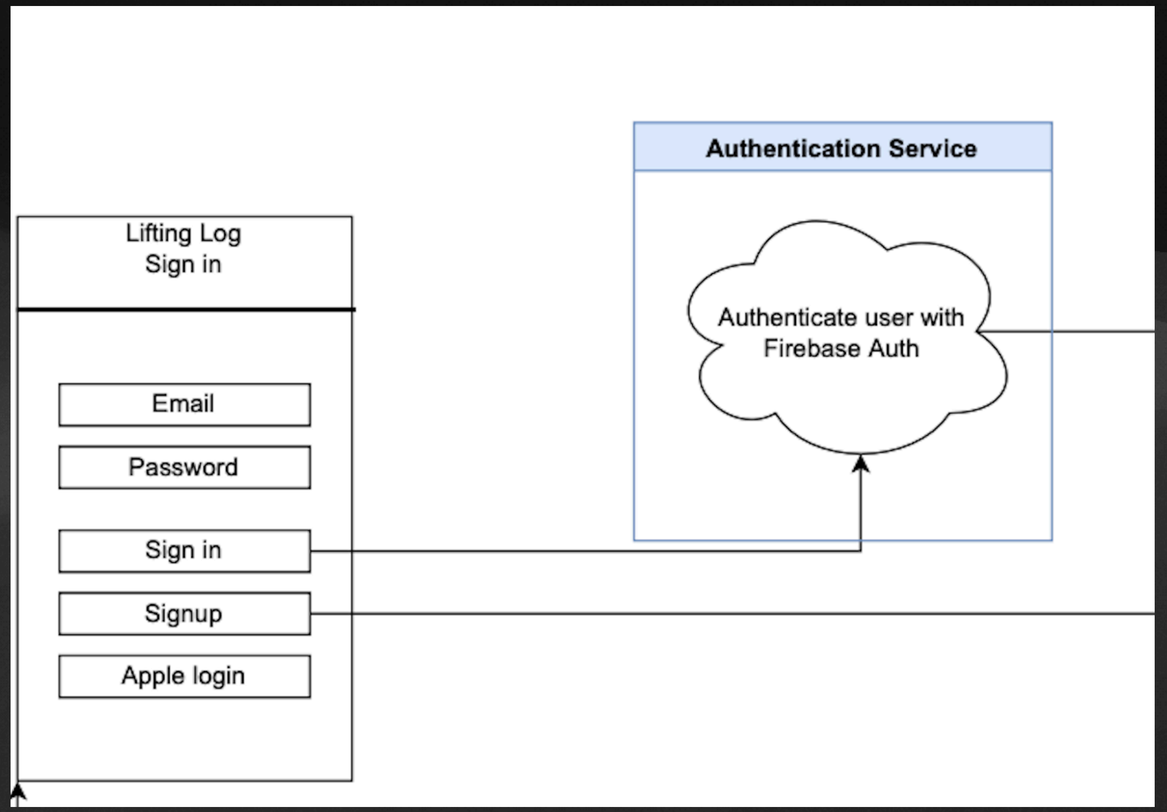




Components

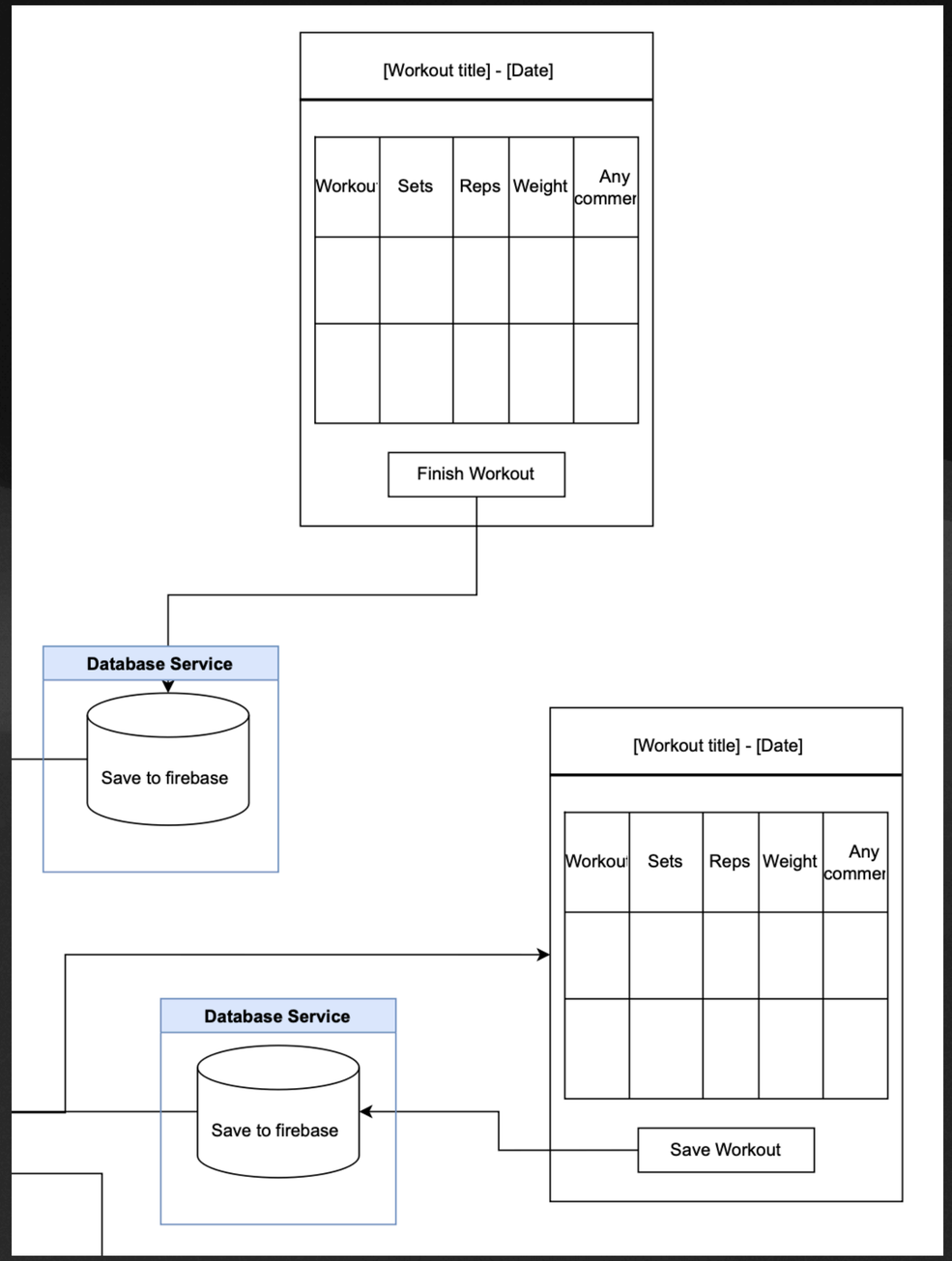
Authentication Service

- When the user clicks “Sign in” on the home screen the users information will be passed to the authentication service.
- The authentication service will validate the users credentials and handle all error handling.
- If the information is incorrect it will throw a error for the user to try again.
- If the information is correct it will tell the view to change to the home screen.



Components
Database Service

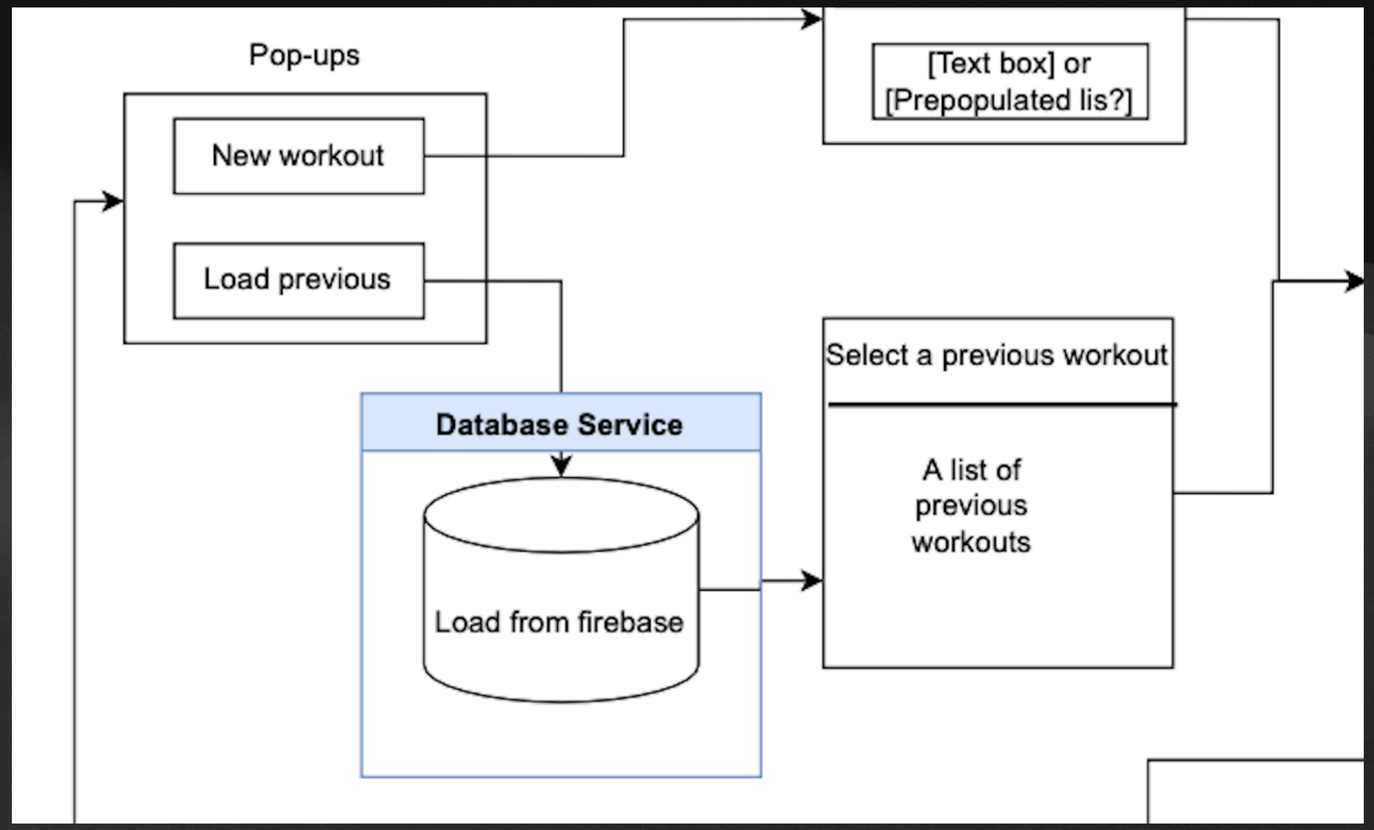
- When a user finishes a workout we will automatically save the workout to keep history.
- The user will also have the ability to create workouts and view them at a later time.
- The Database service is the smart component that will make all of the calls to the database.
    - It will structure the data into a format for Firebase to read.



Components

Database Service

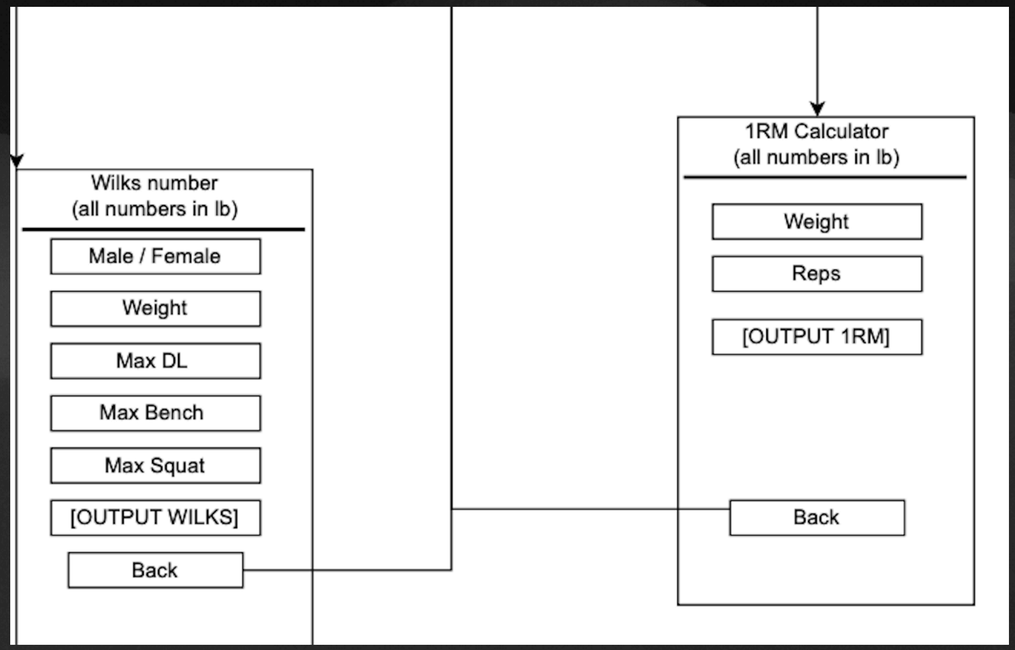
- The user will also be able to load workouts.
- The Database service will also handling calling the information from the database.
- The information returned from Firebase will be in JSON format.
- Database service will adapt the data into a structure that the view is expecting.



Components

Calculation Service

- We are going to offer a few calculators to help lifters in the gym.
- There will be a one rep max and a wilks number calculator.
- Both of these calculators will utilizer a calculation service.
- The views will pass the user information to the service and output the result.



Recap

- Using services to do all of our calculations and database calls will allow the app to be flexible.
  - We can change our backend without impacting our front end and vice versa. Such as switching from Google Firebase to Apple’s CloudKit.
- It keeps the code organized, if you need to make any changes you just have to look for the service that’s doing the work.
- It also streamlines the development:
  - Allowing one person to focus on creating and manipulating views. While the other builds out the backend services.
  - Easier to connect front end to backend.

Any questions?