February 1, 2019

FitHub

Design Doc

Team 5: Andy Plank, Jonathan Huber, Ray Truong, Collin Li, Trevor Hill, Brian Long

Index

- **Purpose** **2**
  - Functional Requirements
  - Non-Functional Requirement

- **Design Outline** **5**
  - Components
  - High-Level Overview

- **Design Issues** **7**
  - Functional Issues
  - Non-Functional Issues

- **Design Details** **10**
  - Class Diagram
  - Descriptions of Classes and Interaction between Classes
  - Sequence of Events
  - API Routes
  - UI Mockup

Purpose

Being fit is already hard enough, people should not have to create new workouts, remember them, and track their progress. FitHub allows users to log their progress, gives them suggestions for new exercises, and connects them with others so they can review their workouts. Many workout apps have features such as logging, workout suggestion, and social networking, but none combine all of them into one, easy to use, mobile app.

Our app seeks to satisfy this absence in the marketplace, providing a workout companion that provides all of the commonly desired features amongst users of other products into one elegant UI.

Functional Requirements

1. Users can register and maintain a user profile

As a user,

1. I would like to register for an account using Google.
2. I would like to log into my account.
3. I would like to view my profile.
4. I would like to edit my profile.
5. I would like to view personal record and total volume lifted statistics.
6. I would like to view the number of days of activity with a days-worked log in the style of GitHub's commit history.

2. Users can interact with and view previous workouts

As a user,

1. I would like to view a calendar of my previous workouts.
2. I would like to be able to view my workouts from any device that has FitHub installed.
3. I would like to select a day from the calendar to view the workouts for that day.
4. I would like to select a workout from a day and see more details about it.
5. I would like to be able to view all exercises within a previous workout
6. I would like to be able to view all weight and rep counts within a previous workout

3. Users can track their progress and log their workouts

As a user,

1. I would like to log a workout (custom or standard) on a day.

2. I would like to schedule a workout (custom or standard) for a day in the future.
3. I would like to track repetitions and weight for each set of an exercise, within a workout
4. I would like to receive a notification when I have “missed” my workout.

4. Users can create their own workouts and exercises

As a user,

1. I would like to create a custom workout by adding either standard or custom exercises.
2. I would like to create a custom exercise.
3. As a user, I would like to have workouts suggested to me based on muscle group and time.

5. Users can interact via an integrated social platform

As a user,

1. I would like to publicly post one of my workouts.
2. I would like to view the profile of someone who publicly posted a workout.
3. I would like to view publicly posted workouts as an Instagram style list.
4. I would like to filter the publicly posted workouts by muscle group and time.
5. I would like to sort the publicly posted workouts by rating and time.
6. I would like to rate a publicly posted workout.
7. I would like to comment on a publicly posted workout (if time allows).
8. I would like to save a publicly posted workout to my collection (if time allows).

6. Users can view caloric and nutritional information

As a user,

1. I would like to log my caloric and nutritional intake (if time allows).
2. I would like to view my previous caloric intake as a graph/chart (if time allows).
3. I would like to see a graph of my caloric intake (if time allows).

Non-Functional Requirements

1. Usability

One of the biggest complaints about workout apps is a lack of user-friendliness. Since our application is very feature-dense, we need to make general navigation

simple and self-explanatory. Users should be able to navigate without a tutorial or guide.

2. Scalability
Our app will store a lot of information in the user database. The database needs to be able to organize all of this information easily and traversed efficiently no matter how much data there is. We also need to be able to scale the database storage size up and down to handle a user base of one to a user base of one thousand.

3. Security
Sensitive user data, such as email and password, will be stored in a database. This database needs be protected from SQL injections and passwords need to be encrypted in case of a breach.

4. Performance
The front-end will have to request data from the back-end which will be communicating with the database. This amount of network communication will dramatically slow down the app which will provide a bad user experience. Since the user experience is important to us, we need to ensure that our app can display all content within one second of a user trying to access it.

Design Outline

Our project is a mobile application that allows users to create, log, and discover various workouts. Our implementation will be a simple Client-Server-Database model. Multiple iOS/Android React Native clients will request the server, the server will handle the request and query the database, and the database will hold all of the information.

Components

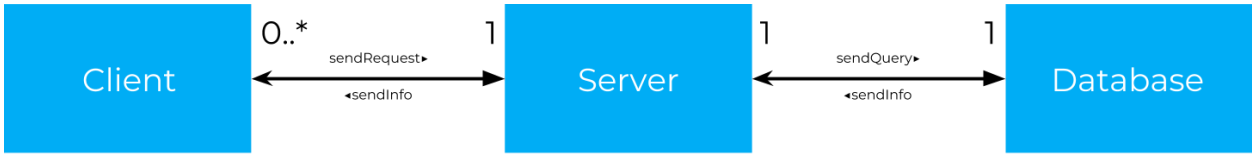
1. iOS/Android React Native Client (Client)
    a. The Client, running on multiple phones, will be the interface to our project.
    b. The Client will send data to, and receive data from, our Server by HTTP requests in JSON format.
    c. Data will be received from the Server, and then used in various parts of our application such as saved workouts.

2. Node.js Web Server with Express API (Server)
    a. The Server will allow users to send and receive information with requests to simple URLs.
    b. The Server will accept HTTP requests using REST URI conventions.
    c. The Server will act as a middleman for the Client and the Database by receiving a request, querying the Database, and then returning the data in JSON format.

3. MongoDB Non-relational Database (Database)
    a. The Database will hold all of the data used in our application.
    b. Typical data stored in the Database will be links between users, saved workouts, and statistics about the user.
    c. If scaling is needed, other parts of the application can be modified without losing data.

High-Level Overview

From left to right, the Client forms a many-to-one relationship with our Server in order to allow for multiple mobile clients contacting one web server API. When the Client sends a request to the Server, the Server will handle the routing, process the request, and then call a specific function in order to query the Database. When the Database receives a query, it handles the query within our Mongo database, and then returns the data to the Server. From there, the Server then handles the data and then returns it to the Client. Finally, the Client displays the information.



Design Issues

Functional Issues

1. Do users need to login to use the app, and if so, how?
   Option 1: Allow users to create a unique username and password
   Option 2: Allow creation of a login id via Google+, Facebook, etc
   Option 3: No account creation needed
   Decision: We chose to allow login through Google because we feel it would easier and in the end, more secure. Because none of us have experience in creating a secure login system, it would probably be in the best interest of the users to have a prebuilt login system from a large company.

2. When should we update user stats on the profile page? (e.g. period of time they haven't exercised)
   Option 1: Update values everytime they change
   Option 2: Update values in set intervals
   Decision: We chose to update values in set intervals rather everytime they change. By doing so, we lessen the amount of spam. This makes the UI look cleaner overall.

3. How should exercise activity be displayed on the profile page?
   Option 1: Display in a series of colored blocks similar to GitHub activity
   Option 2: Have a calendar and mark off each day the user worked out
   Decision: We chose to display exercise activity similar to that of GitHub. This would allow us to display a more broad overview of the user's workout commitment. For example, GitHub has blocks covering a whole year of commits, each with a different shade green. For our app, we will use shades of blue to describe the intensity of the workout that day.

4. What can the user search in the feed?
   Option 1: Users can search other users
   Option 2: Users can search for workouts
   Option 3: Users can search for both workouts and other users
   Decision: We chose to only display other users when they search and let workouts have their own dedicated page. This would clean up any cluttering in the search results and would allow us to categorize workouts on their own page.

5. How should an exercise's number of sets (with corresponding weight and repetitions) be structured within the UI?
    Option 1: Users can specify number of sets ahead of time, and then fill in weights and repetitions as they are completed into a provided chart-like structure.
    Option 2: Users can add new sets to a workout as they are performed, with each new set appearing below the previous.
    Decision: We chose to allow users to add sets as they are completed. This allows for less wasted space within the UI, and prevents the user from having to predetermine the number of sets they plan on performing ahead of time.

Non-Functional Issues

1. What type of database is appropriate to store our data?
    Option 1: A relational database (e.g. mySQL, Microsoft SQL, etc)
    Option 2: A NoSQL database (e.g. MongoDB)
    Decision: We decided on a NoSQL database, more specifically MongoDB. We chose this because a few of our members have a little bit of experience creating and hosting MongoDB databases on the Google Cloud Platform. We also feel that with our structure of creating workouts and saving other people's workouts, a non-relational database would be most efficient.

2. How are we going to host our backend services?
    Option 1: Google Cloud Platform
    Option 2: Amazon Web Services
    Decision: We chose to host our backend services using the Google Cloud Platform for a couple of reasons. First, a couple of our members have experience using the services that Google provides. Second, one member has credit to use on the platform, so we will be able to host everything for free, no matter the size of the application.

3. Which language is best suited to implement our backend services?
    Option 1: JavaScript
    Option 2: Java
    Option 3: Python

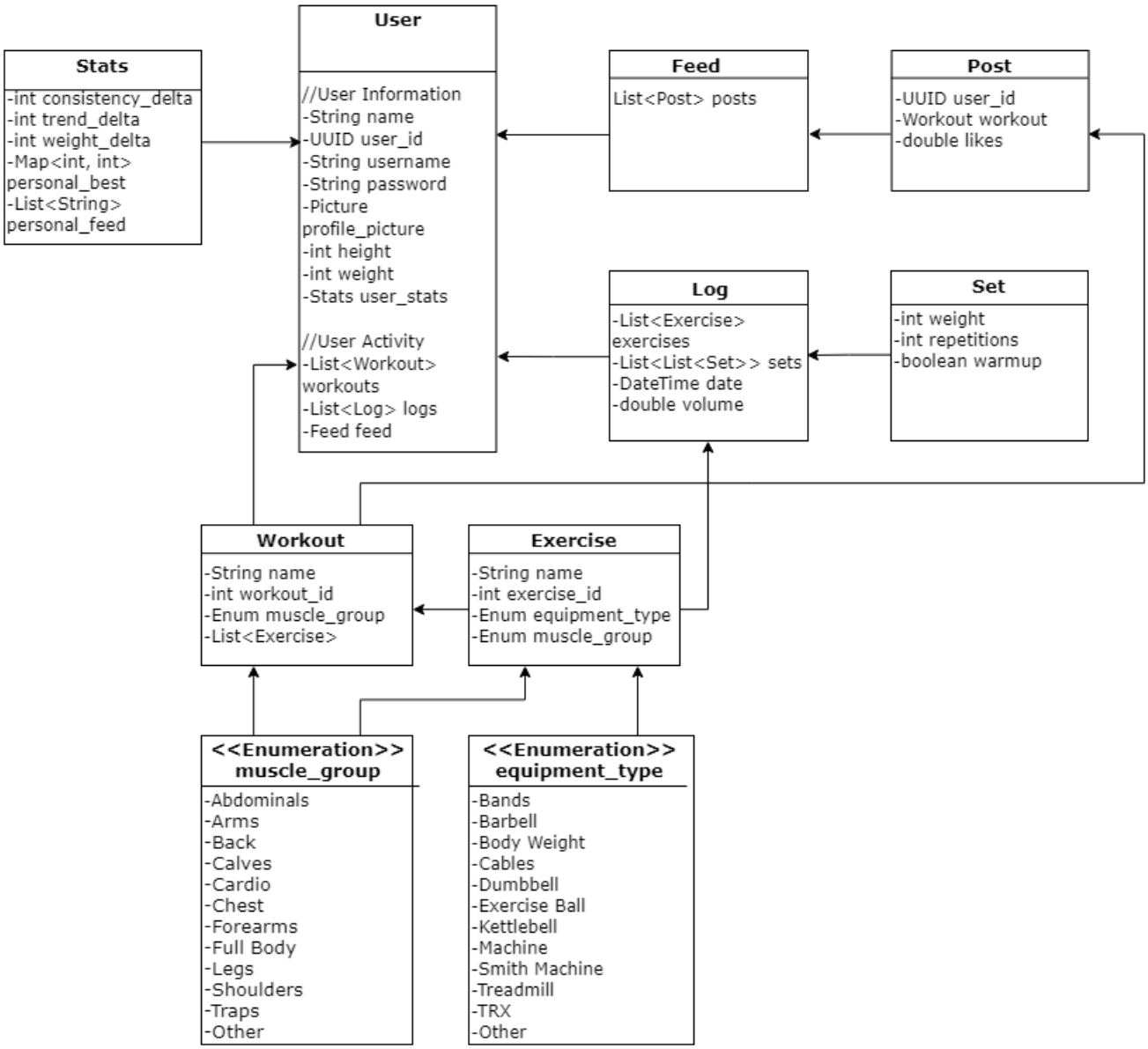
Decision: We chose to implement our backend using JavaScript since a majority of our group has experience implementing backend services in JavaScript. We plan on implementing our backend service using the Node.js environment, which has thousands modules that can be helpful.

4. How can we update the feed to reflect current activity as the number of users increase?
   Option 1: Let the user manually refresh the page (e.g. pull down and let go)
   Option 2: Automatically refresh the feed after a set time interval
   Decision: We decided to let each user manually refresh the page, as that is the implementation that most modern social medias use. We also feel that it would be easier to implement.

5. What framework are we going to use to implement our application across multiple platforms?
   Option 1: React Native
   Option 2: Flutter
   Decision: We decided to go with React Native, as one of our members has experience with both React and React Native. Also Flutter requires you to write in Dart, which would require us all to spend a significant amount of time learning it.

Design Details

Class Diagram



Descriptions of Classes and Interaction between Classes

The classes represent the essentials of what a user needs to use our app. Using these classes, and their respective get/set methods, should make the interaction between the database and the user quick and easy.

User:

- Represents a unique user.
- Contains personal info about the user such as their name, weight, height, etc.
- Used as the main class to store stats and workouts of the user.
- Created when a person creates an account.
- Stats is a property in the user information section of the User class, and is updated whenever the user completes a workout.
- Workout is a property in the user activity section of the User class, containing all privately created workouts, as well as workouts saved from the public feed.
- Log is a property in the user activity section of the User class, and tracks the exercises the user did along with sets and weights.
- Feed is a property in the user activity section of the User class, containing workouts and/or users explicitly searched for by the user.

Exercise:

- Represents a specific exercise.
- Contains two identifiers, equipment_type and muscle_group, to classify exercise.
- Created when a user adds the exercise to their workout.

Workout:

- Represents a list of exercises.
- Contains a name, workout_id, and muscle_group to categorize the workout.

Set:

- Represents the amount of weight and number of repetitions performed within an exercise.
- Contains a boolean, initialized to false, if the user wants to classify the exercise as a warmup.

Log

- Represents what the user did for the day.
- Contains list of workouts, a list containing lists of sets, a DateTime object, and volume.

Post

- Represents a workout shared on the global feed.
- Contains a user id, specific workout, and amount of likes.

Feed

- Represents a list of posts.
- Elements within the list will be displayed on the global feed.

Stats

- Represent the stats of a user.
- Contains information about the user's improvement trends.
- Contains a mapping of the user's personal bests and a personal feed.

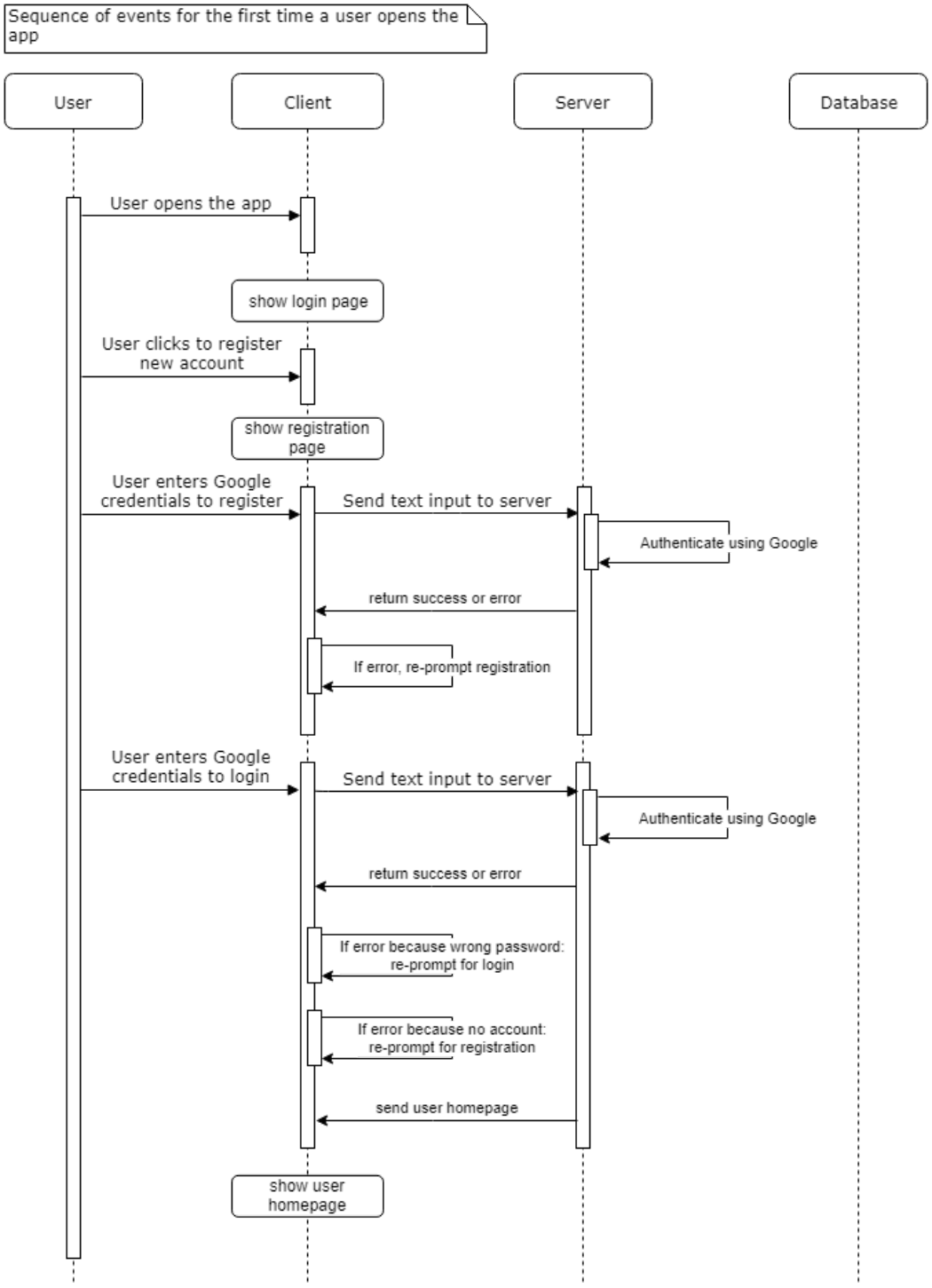
muscle_group

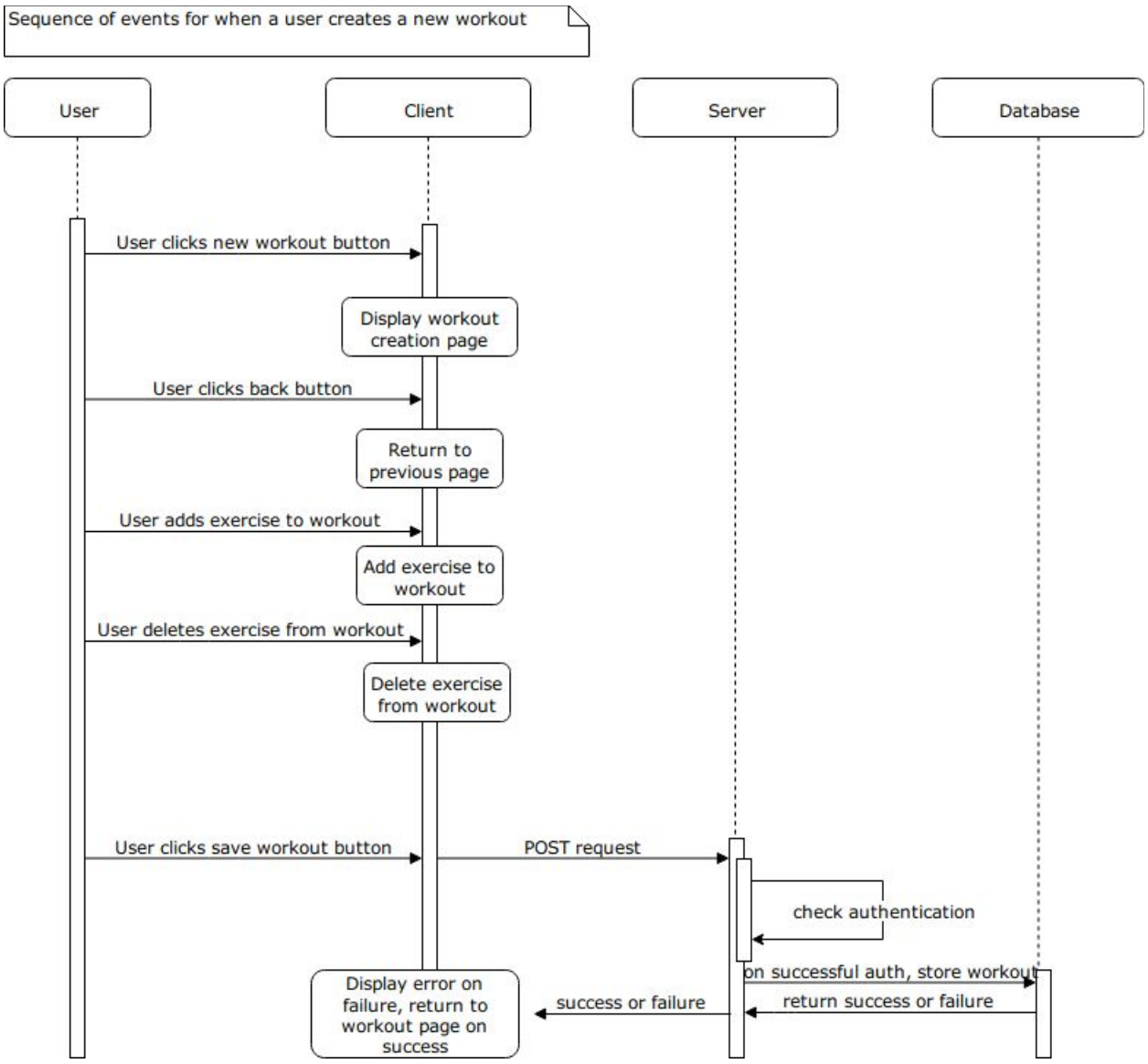
- Used to identify what muscle group the exercise or workout is training.
- Defined whenever an exercise or workout object is created by the user.

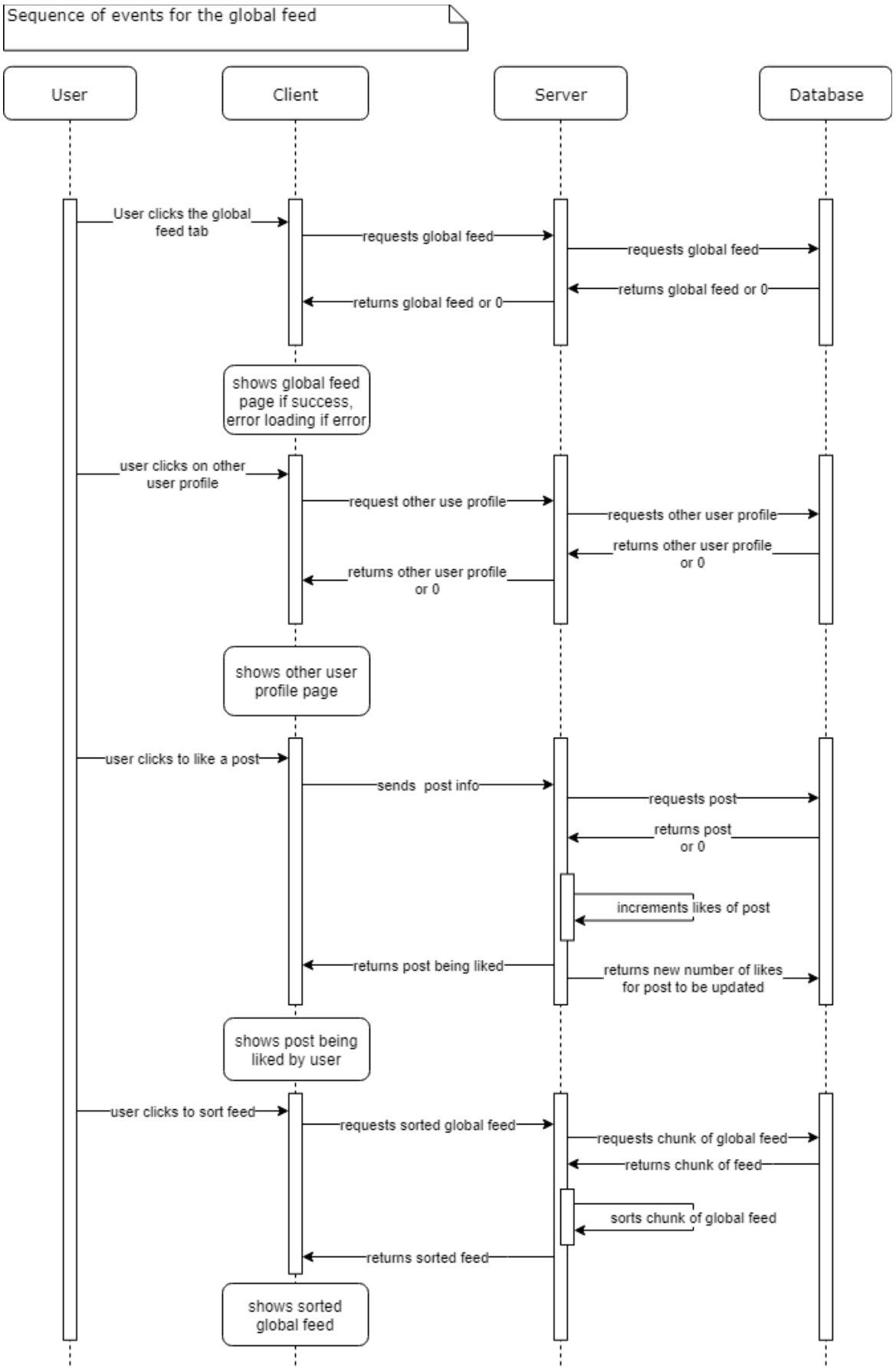
equipment_type

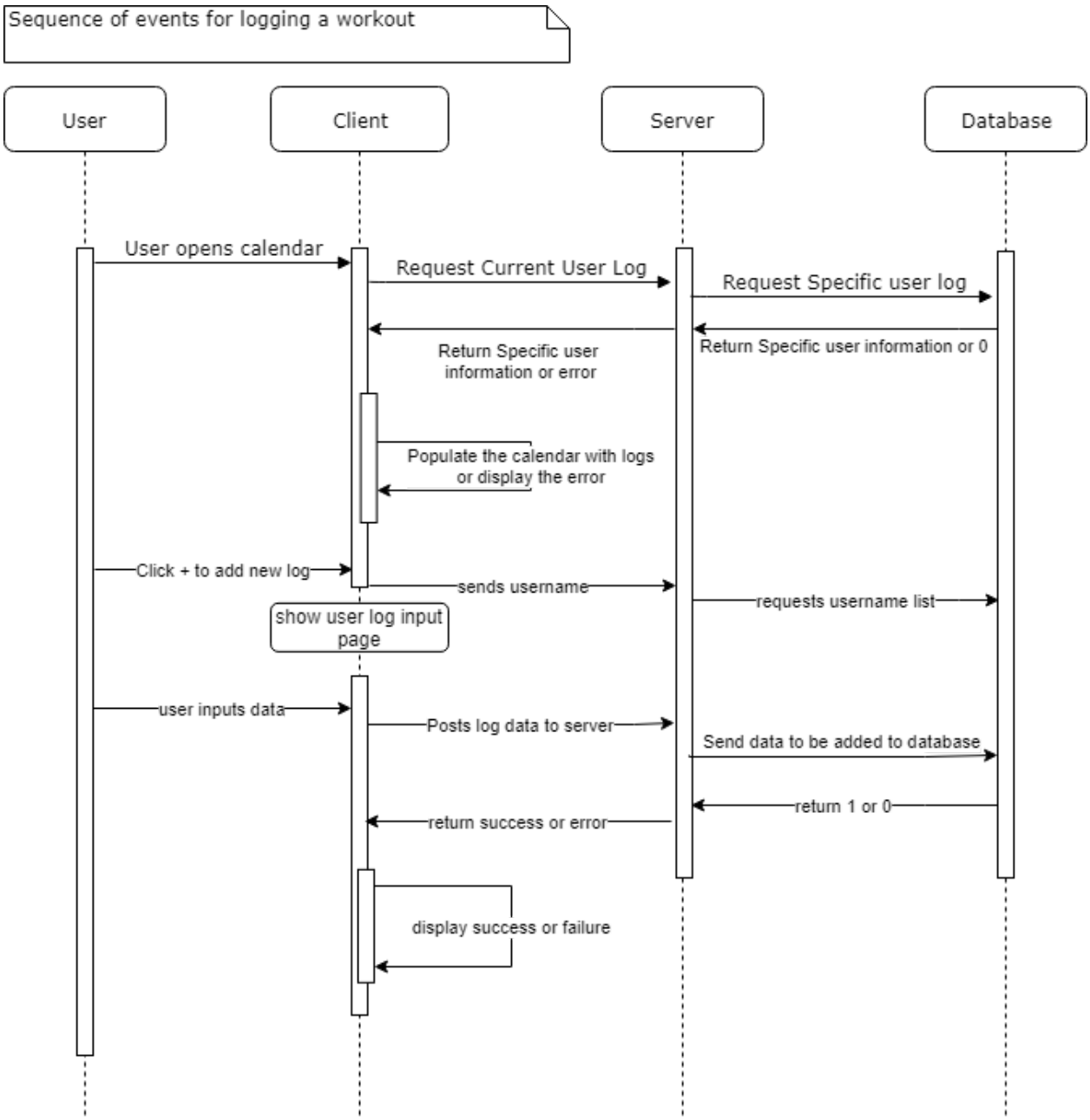
- Used to identify what equipment the exercise requires.
- Defined whenever a workout object is created.

Sequence of Events




Sequence of events for when a user creates a new workout
User
Client
Server
Database
User clicks new workout button
Display workout creation page
User clicks back button
Return to previous page
User adds exercise to workout
Add exercise to workout
User deletes exercise from workout
Delete exercise from workout
User clicks save workout button
POST request
check authentication
on successful auth, store workout
return success or failure
success or failure
Display error on failure, return to workout page on success

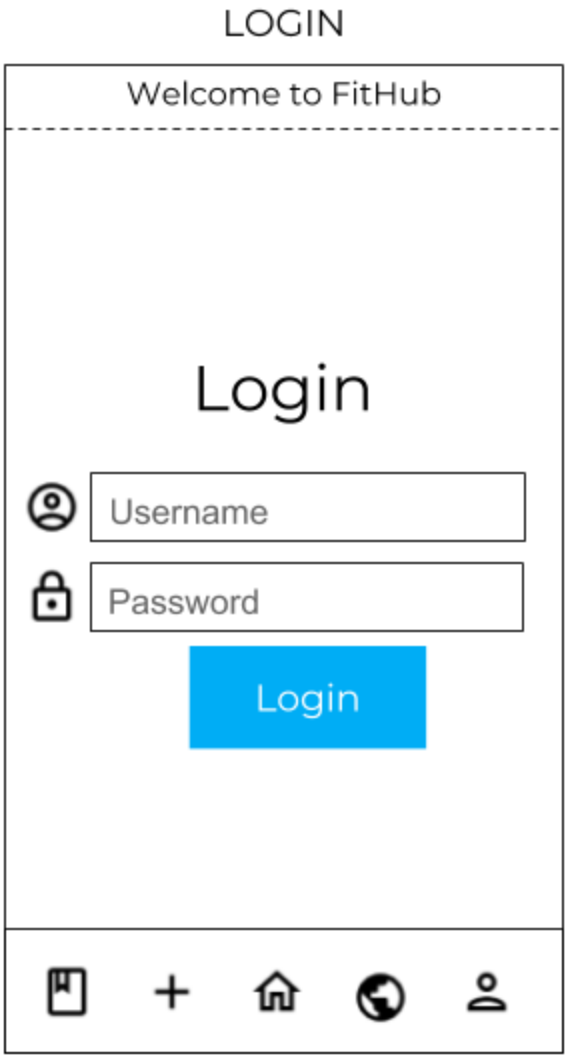

Sequence of events for the global feed
User
Client
Server
Database
User clicks the global feed tab
requests global feed
requests global feed
returns global feed or 0
returns global feed or 0
shows global feed page if success, error loading if error
user clicks on other user profile
request other use profile
requests other user profile
returns other user profile or 0
returns other user profile or 0
shows other user profile page
user clicks to like a post
sends post info
requests post
returns post or 0
increments likes of post
returns post being liked
returns new number of likes for post to be updated
shows post being liked by user
user clicks to sort feed
requests sorted global feed
requests chunk of global feed
returns chunk of feed
sorts chunk of global feed
returns sorted feed
shows sorted global feed


Sequence of events for logging a workout
User
Client
Server
Database
User opens calendar
Request Current User Log
Request Specific user log
Return Specific user information or 0
Return Specific user information or error
Populate the calendar with logs or display the error
Click + to add new log
sends username
requests username list
show user log input page
user inputs data
Posts log data to server
Send data to be added to database
return 1 or 0
return success or error
display success or failure

API Routes

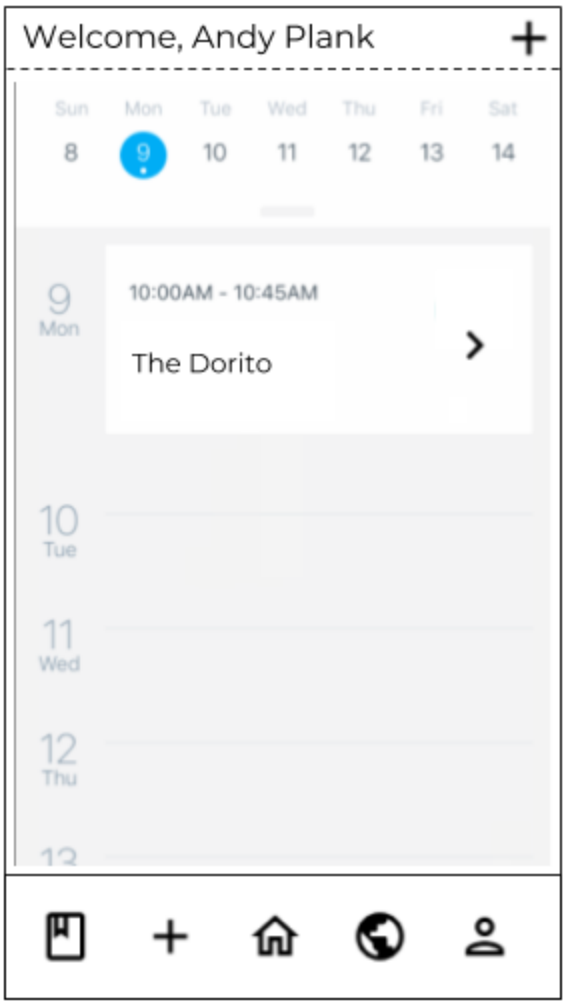
Purpose	API Route	Supported Requests
Logs in the user	/login	POST
Creates a user	/users/new	POST
Gets the workouts for a user	/users/:user/workouts	GET
Gets or sets the stats for a user	/users/:user/stats	GET, POST
Gets or sets the personal information for a user	/users/:user/details	GET, POST
Searches for a user	/users/search/:query	GET
Adds a workout for a user	/workouts/add	POST
Gets, edits, or deletes a workout	/workouts/:workout	GET, POST, DELETE
Gets the global feed	/workouts/global	GET
Likes a workout	/workouts/:workout/like	POST

UI Mockup

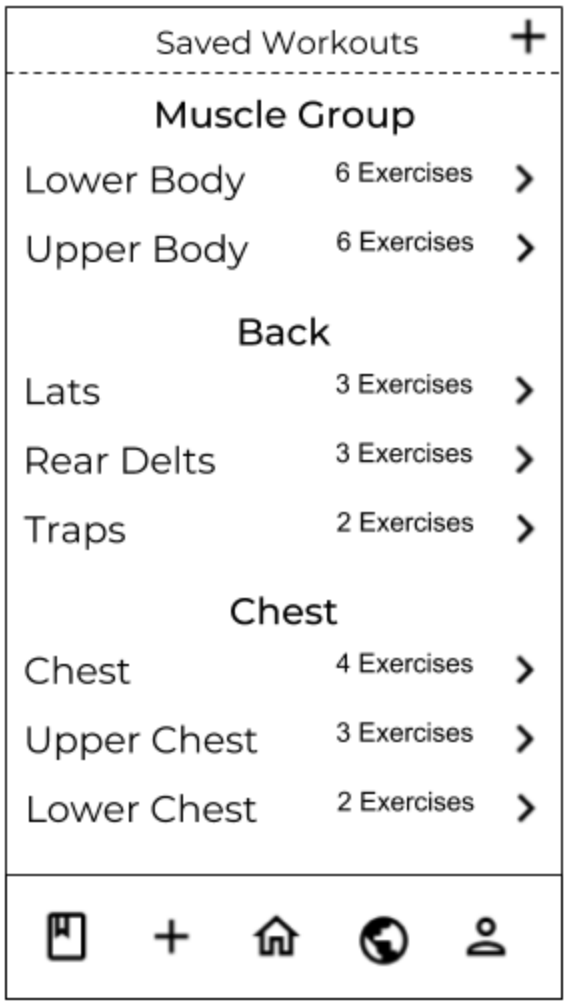


HOME PAGE (CALENDAR)

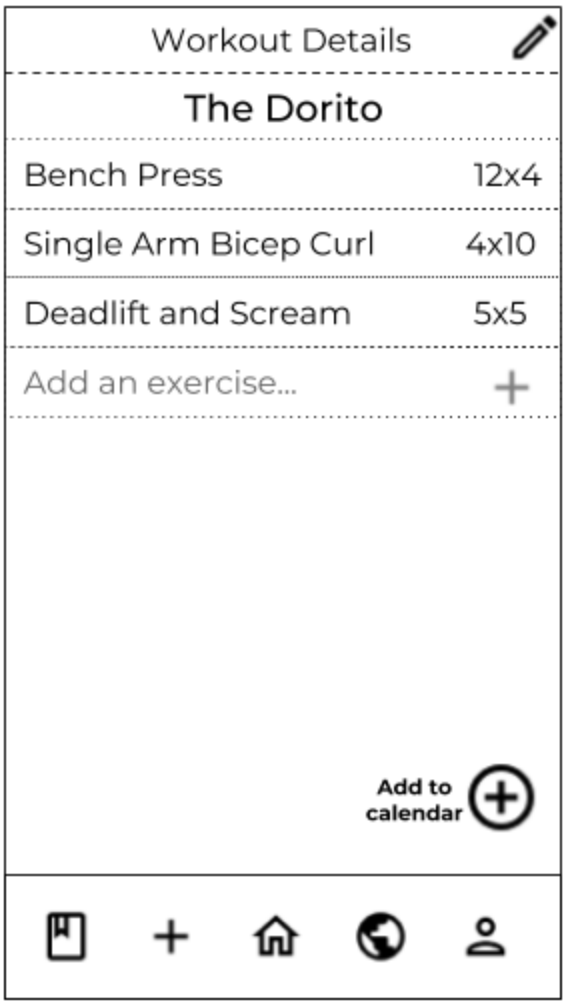
HOME PAGE (AGENDA)



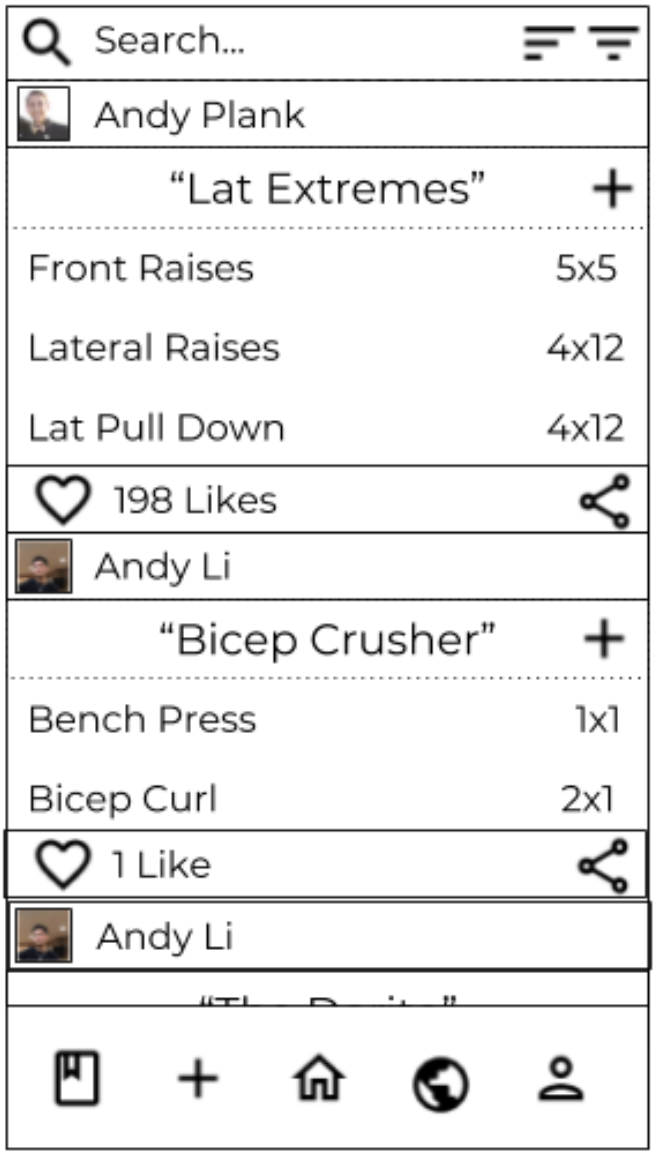
WORKOUTS PAGE



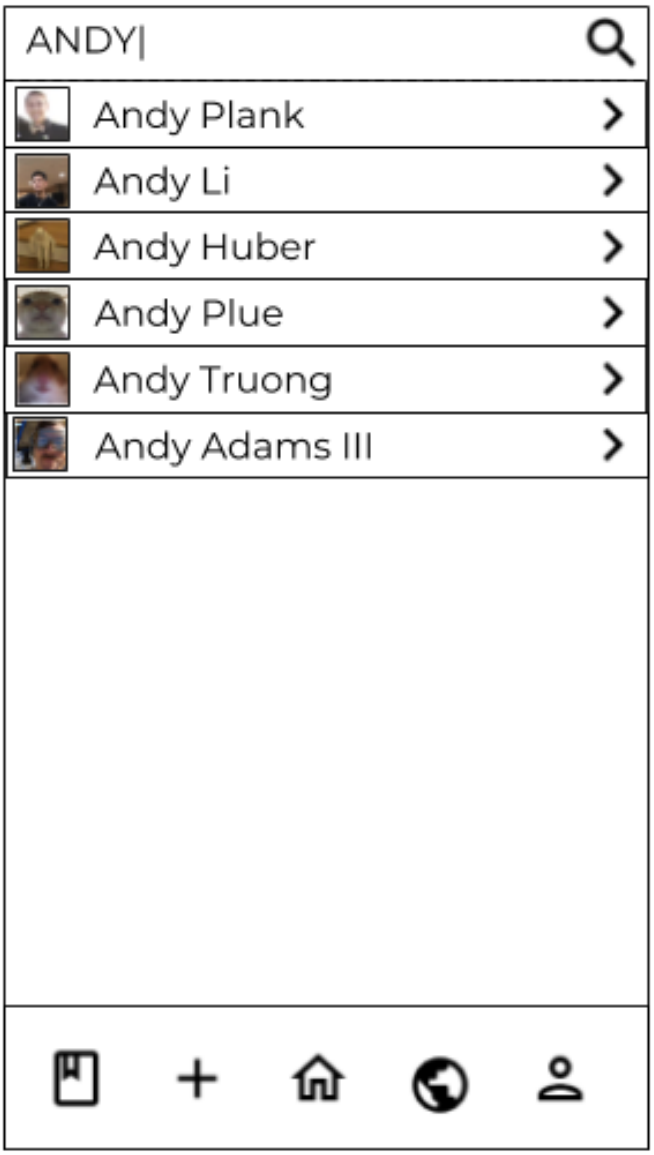
SINGLE WORKOUT PAGE



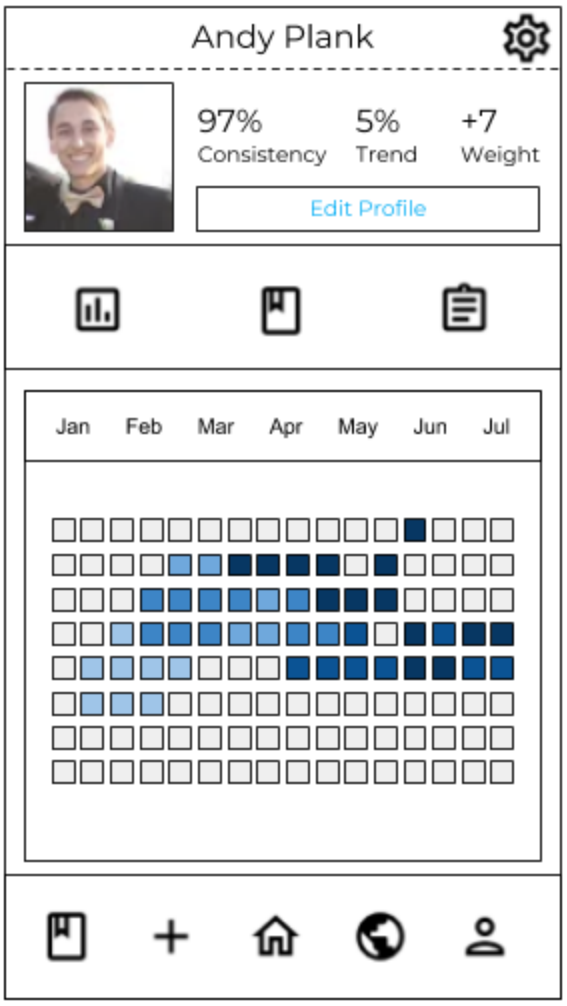
FEED



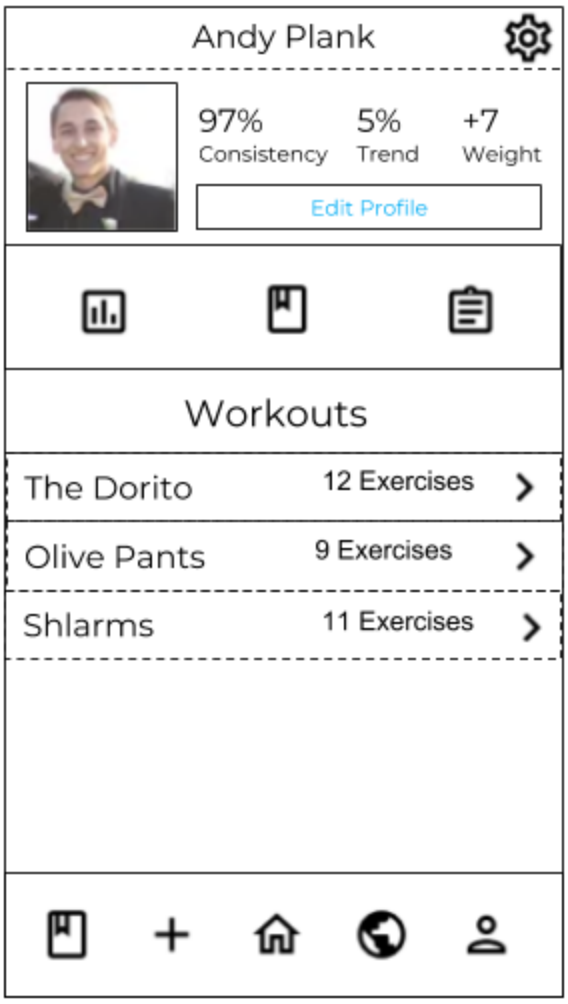
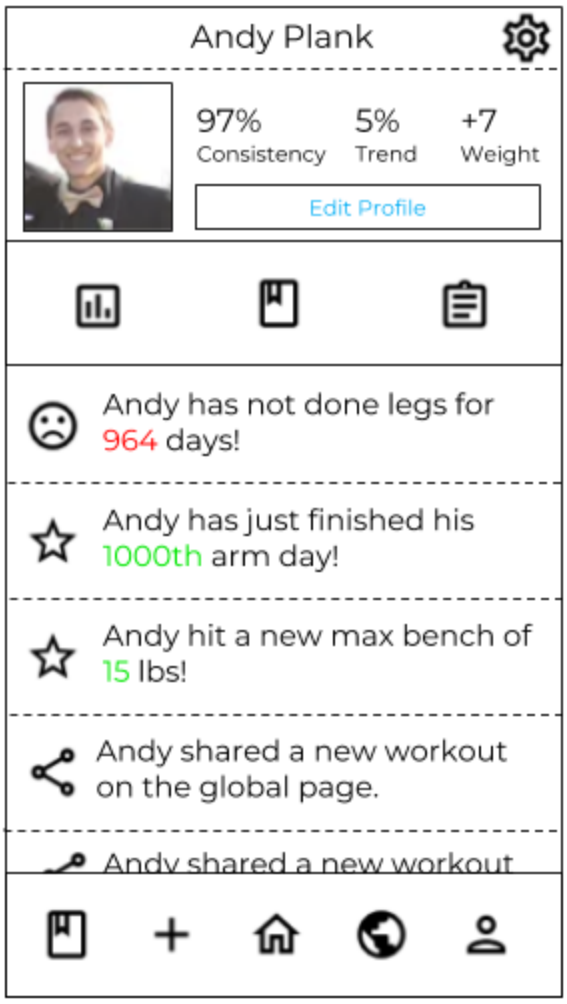
FEED (SEARCH)



PROFILE PAGE (DEFAULT)



PROFILE PAGE (WORKOUTS)



PROFILE PAGE (ACTIVITY)