Design Document - getFit

Repository: https://github.com/guttamc7/CS-490

Rishabh Mittal , Gurumukh Uttamchandani,
Neeraj Agrawal, Ishan Shah, and Jai Nalwa

TABLE OF CONTENTS

Purpose

Our getFit android application focuses on providing a platform through which users can create personalized workout schedules around their daily routines. Users can also find and connect with other nearby users with similar time schedules. Our app not only aims to connect and motivate fitness enthusiasts but also aims in creating a healthier lifestyle at colleges.

This document provides intricate details about getFit mobile application. It focuses on the implementation of the application by stating the priorities, design outline, design issues, design details and various other aspects.

General Priorities

- **Usability**
    - The usability is very important on the application side since the app will be used by non-technical users. The application's user interface needs to be user friendly. There should never be any hesitation of how to use the app especially when using key features such as creating workout schedules. Everything including advanced features such as geolocation should be easily accessible.

- **Reliability**
    - Reliability is a core focus for the front end application. A system crash on the server side should never result in lost data. This is especially important since the entire software is built upon the availability of that data. The client application should also be very stable to be user friendly.

- **Scalability**
    - The system should be scalable to accept increasing number of users. It should easily accumulate the number of users, which can change quickly. The server will be capable enough to store these ever increasing details and still ensure fluid communication with the clients. A system crash due to an increased number users is inadmissible.

- **Security**
    - All communication between the client application and the server needs to be encrypted. Even though most information sent between the application and the server will be public data, an acceptable cryptographic protocol has to be implemented to protect user data.

- **Performance**
    - In order to give the right shape to our idea and to truly exemplify the beauty of our app, it will be imperative to maintain top notch performance. This will be implemented by sending and receiving small chunks of data, that will be categorized to ensure minimal response time. Data that is frequently used will be stored on the client side, which would not require contacting the server and which shall enable it to run in offline mode as well. Hence, the application on the client side should consume minimal battery life and when not in use, it should not use any of the processor's power. Overall, having extremely efficient and robust algorithms and maintaining international coding standards will be our top priority.

- **Quality**
    - Outmost importance must be given to the quality of our application by checking for critical bugs/errors and test it properly so that the users don't have to face any problems.This is important because user experience is our top most priority and thats why everything has to run smoothly.

- **Extensibility**
    - We will be designing the code such that it can easily allow us to incorporate additional features into our app.

- **Platform**
    - As part of our first sprint, we will be developing getFit for the Android 5.0 framework. We do aim at making the app backward compatible which will be made extremely easy by making use of the appropriate libraries that are already provided on the Android SDK.

Note: We have replaced our initial design outline by replacing the server and database by Facebook Parse. Please refer below for more details.

Design Outline

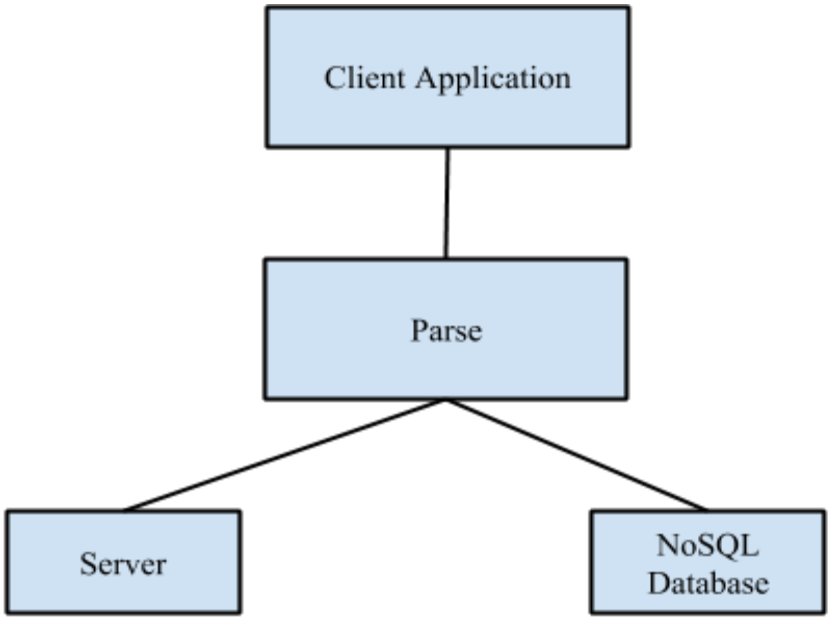


Diagram I: The client application communicates with a Parse Backend that in turn reads and writes data from a NoSQL database.

- Our system model consists of two subsystems: Client Application, Facebook Parse

- Client Application:
  - ❖ The client application will be Android devices used by the users which will communicate with the Parse backend. To connect the client application to the Parse backend, each application is given a unique application ID and client key.

- Facebook Parse Backend:
  - ❖ We will be using Facebook Parse as a backend, which is an integration of a server and a database. This will allow the client applications to access and store the required information from the parse NoSQL database. Parse will automatically handle all the requests sent by the android client and respond accordingly. Data is stored and retrieved in the form of key-value pairs (these are also known as Parse Objects).

Android applications will send the respective requests to the Facebook Parse such as store user information / find workout buddy / Update workout schedules etc. Based on the request, Parse will retrieve the Parse objects and respond back to the client. However, mostly all the functional code will be written on the client side and Parse will simply act as an intermediary between the client and the database.

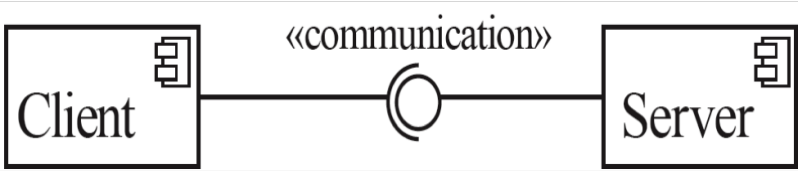
Design Issues

1. Functional Issues

1) Architecture Used

Option 1: Client-Server
Option 2: Pipe-and-Filter

Option Used: Client-Server



Reason

The client-server architecture works best for our case as the client which is the front end of our application will contact the Parse backend server which is a separate module working on the back-end to access data related to other users and fitness programs from the NoSQL database. The server will partition these requests received by various clients and will respond to these service requesters accordingly.

2) Client Design

Option 1: Thin Client
Option 2: Fat Client

Option Used: Fat Client

Reason

We will be using Facebook Parse which is an integration of both a server and a database. It will allow the Android applications to directly access and store the data on the Parse backend. Since most of the requests will be handled by the Android client, and be fulfilled by Parse, our client design will be using a Fat Client. Facebook Parse provides us with free hosting and moreover it greatly simplify the process of developing and deploying our application along with fulfilling

all the functional and nonfunctional requirements. Hence, we decided to use fat client design in accordance with Facebook Parse.

3) Platform Used

Option 1: iOS
Option 2: Android
Option 3: Windows Phone

Option Used: Android
Reason(s):
-User base: The Android operating system powers over 1 billion smartphones and tablets. This enables us to tap into an incredibly large market, which can lead us to making a greater impact with our application.

-Well-documented API: Android is an open-source software stack, which connects us as developers to a huge community. As this is the first time our team members are developing for mobile, we need a well-documented API to help us.

-Third party technologies; Android also gives us the freedom to use third-party tools for development, thus leaving a lot of room for innovation without affecting the usability aspect.

-Developing Restrictions: Developing for the iOS operating system requires the developer to use XCode, which is only available on the Mac OSX. Not all of our teammates own a Macintosh machine, therefore Android was the clear choice winner.

4) Server Hosting and Database Management

Option 1: Heroku
Option 2: Parse

Option Used: Parse
Reason(s):
-Cloud: Parse is a cloud application platform, which made it appealing to us as we can host our application on the cloud rather than host it from a physical Linux box.

-Speed, Security and Scalability: Parse offers one of the fastest instances for servers, and this is one of our major criteria, as we need our app to be very responsive. Security is one of our concerns, as we will handle information that could be personal to our users. As our application is centered on impact, scalability is a big issue and Parse offers a great backend to easily scale our application as it grows.

-Database Management System: Parse uses NoSQL for its database management and all of our teammates are comfortable with NoSQL. To access the database parse has an entity called ParseObject which can be used to extract and insert data into the database.

2. Non-Functional Issues

a. User Interaction Issues

1) Workout Plans

Option 1: Custom
Option 2: Pre-defined

Option Used: Custom(but offers pre-defined or base workout plans as well)
Reason:

getFit is inspired by making people workout. This is achieved by finding users nearby who are going to workout at the same time as the user. Users have the option to choose workout plans and our team decided that the best option is on letting the users create custom workout plans but there will also be a database filled with base workout plans (These base workout plans are based on level of intensity which the user can choose).

2) Response Time

Option 1: To cache data
Option 2: Not to cache data

Option Used: Cache most relevant data
Reason:

To maintain quick response time, there will be efficient search algorithms at use to help users quickly connect to other users. These connections will be parsed into the database with the aim of keeping data usage to a minimum. We will also cache the users most recent workout buddies and workout plans that they have been added, thus helping reduce response times.

3) User Account

Option 1: Pre-existing Social networks
Option 2: App users

Option Used: Pre-existing social networks along with the option of registering with .edu email address
Reason:

Users will be asked to link their Facebook accounts or use their .edu email address when they first sign up to getFit. The application will parse the necessary data retrieved from the facebook account, and keep only what is required, such as the profile picture, friends, etc., thus keeping the data to a minimum. The reason we force users to use their existing social networks is so that they find it easier to find their friends more quickly, and also using the vast user base of an already existing network.

b. User Interface Issues

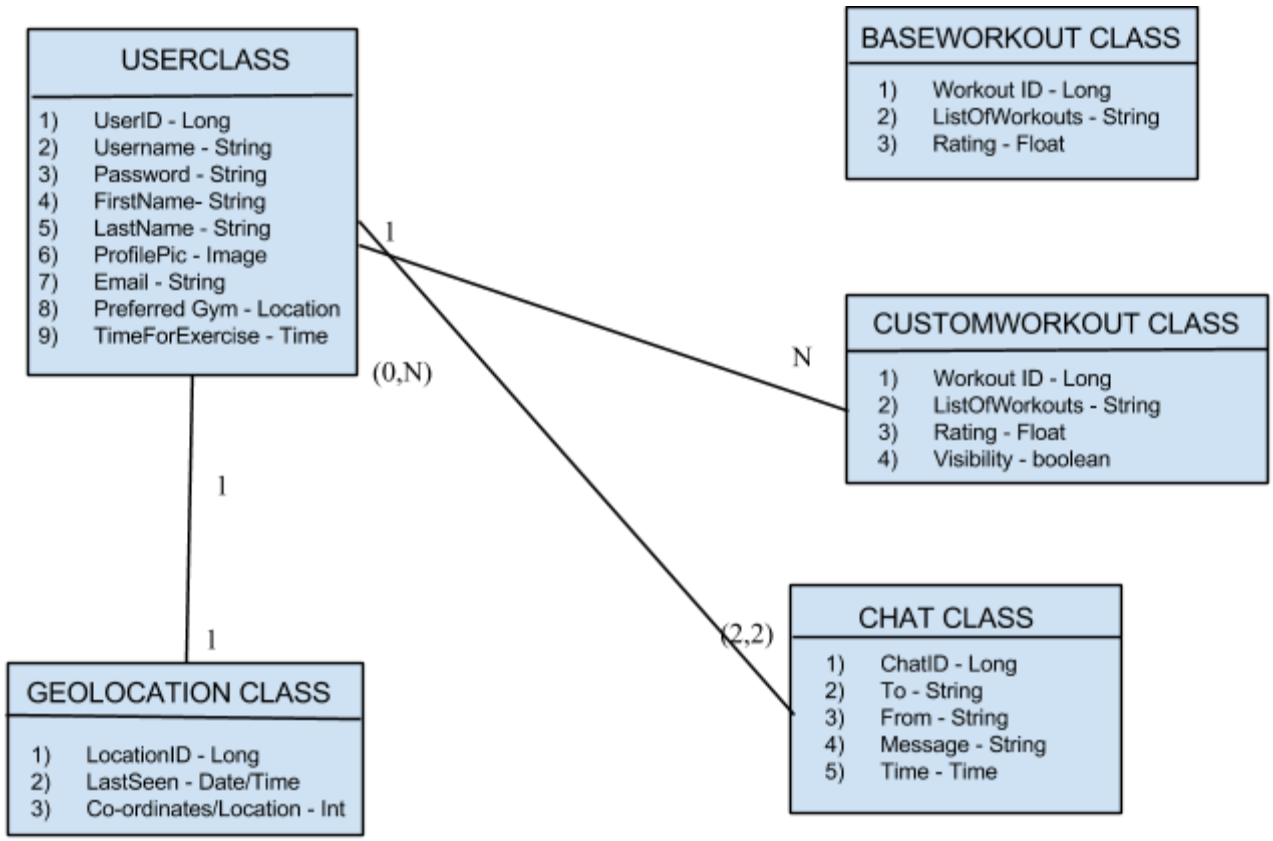
1) Color Scheme
Option 1: Warm colors
Option 2: Cool colors

Option Used: Warm
Reason

Our team decided to go with more warm color palette as we wanted our app to be as inviting as possible. Granted we can achieve this effect using cool colors such as blues and blacks, but the decision was unanimous amongst our team members.

Design Details

1. Class Diagram



2. Description of Classes and Models

We will be using the Parse NoSQL database for storing and retrieving the data. The Parse NoSQL database involves creating different classes for storing the information similar to SQL tables but it stores and retrieves them in name-value pairs (which are also known as Parse Objects) without having any primary/foreign keys. Each class will be having a long id which will uniquely identify different rows in the database.

UserClass

The user class will have a unique key which will be userID. The class will also contain the encrypted password for added security. The user will be adding his/her preferred gym location and the preferred time for workout so that its easier to find nearby workout buddies. The user

class will have an option to interact with custom workout class to create customized workouts, chat class to chat with other registered users and geolocation class to store his/her last seen location in the database.

BaseWorkoutClass

The BaseWorkout class will be uniquely identified by workoutID. This will contain all the basic,intermediate and advanced workout routines that can be used by all the users using our application. We will be manually putting in these routines so that the users can get a head start in finding a couple of exercises. This will also have a rating feature in which the users can rate these basic routines.

CustomWorkoutClass

The CustomWorkout class will be uniquely identified by workoutID. This class would contain list of custom workouts and their details posted by the registered users. It would also contain the rating for a particular workout schedule if made visible rated by other users. The class would only interact with the user class as each user will have an opportunity to create his/her custom workouts.

GeolocationClass

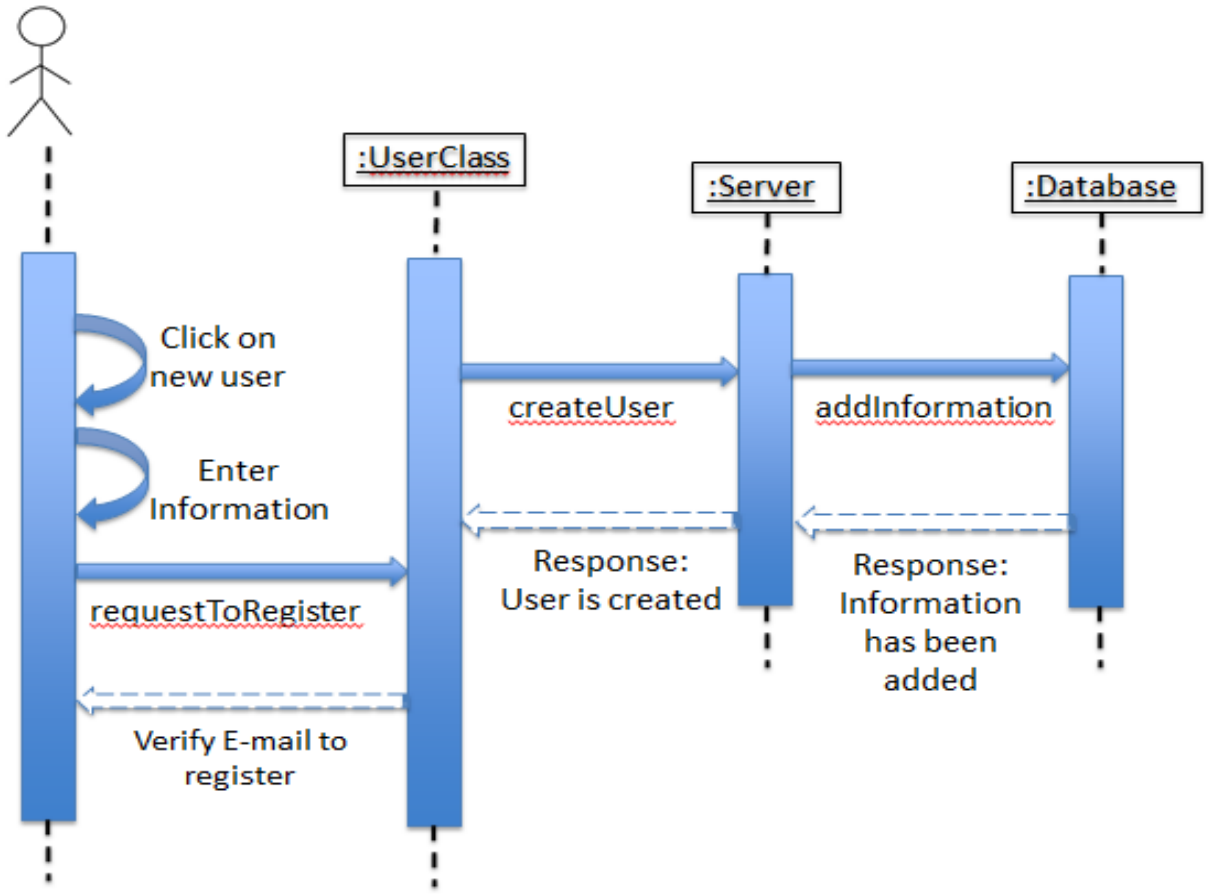
The geolocation class will be uniquely identified by locationID. This class will contain the last seen time and location of the users. This will be used when users are trying to find nearby users in order to find a gym buddy. The last seen time and location will be updated as per the users preference and availability.

ChatClass

The chat class will be uniquely identified by chatID. The chatClass will be used to store the chat messages between the users. Each chat message will contain the to/from, message and the time of the message so that it can easily be retrieved when the user want to see his/her chat history.

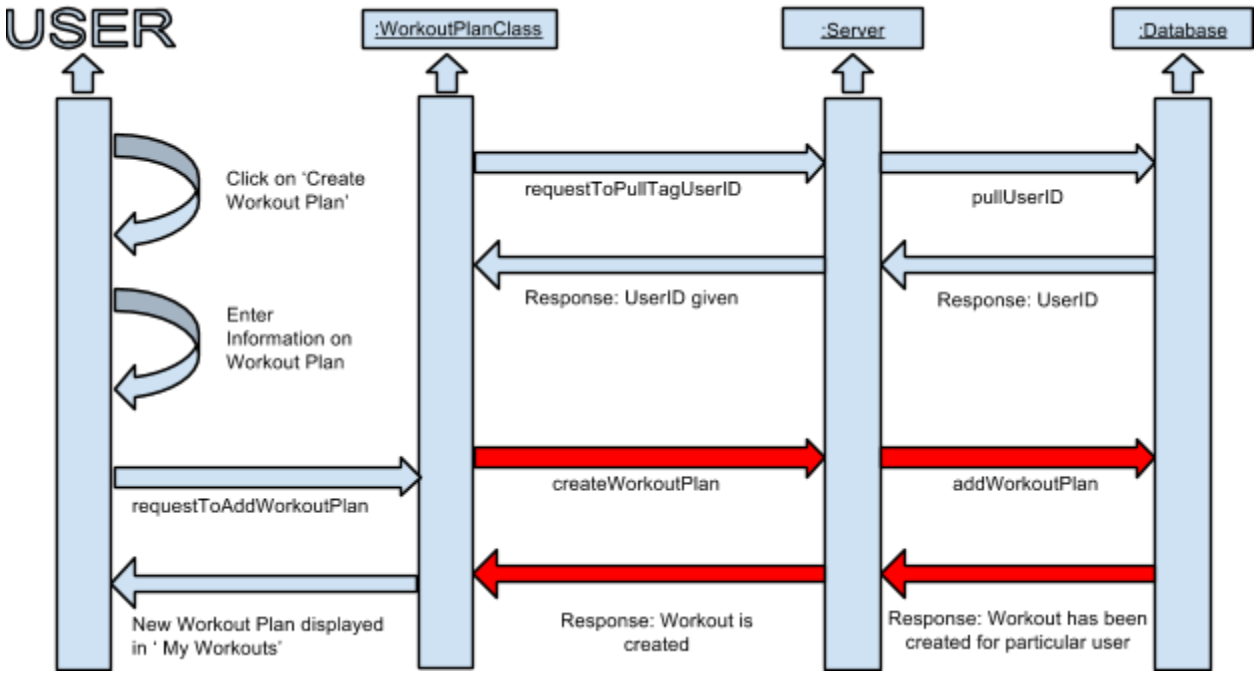
Sequence Diagrams

1) User – Application E-mail Registration Interaction:



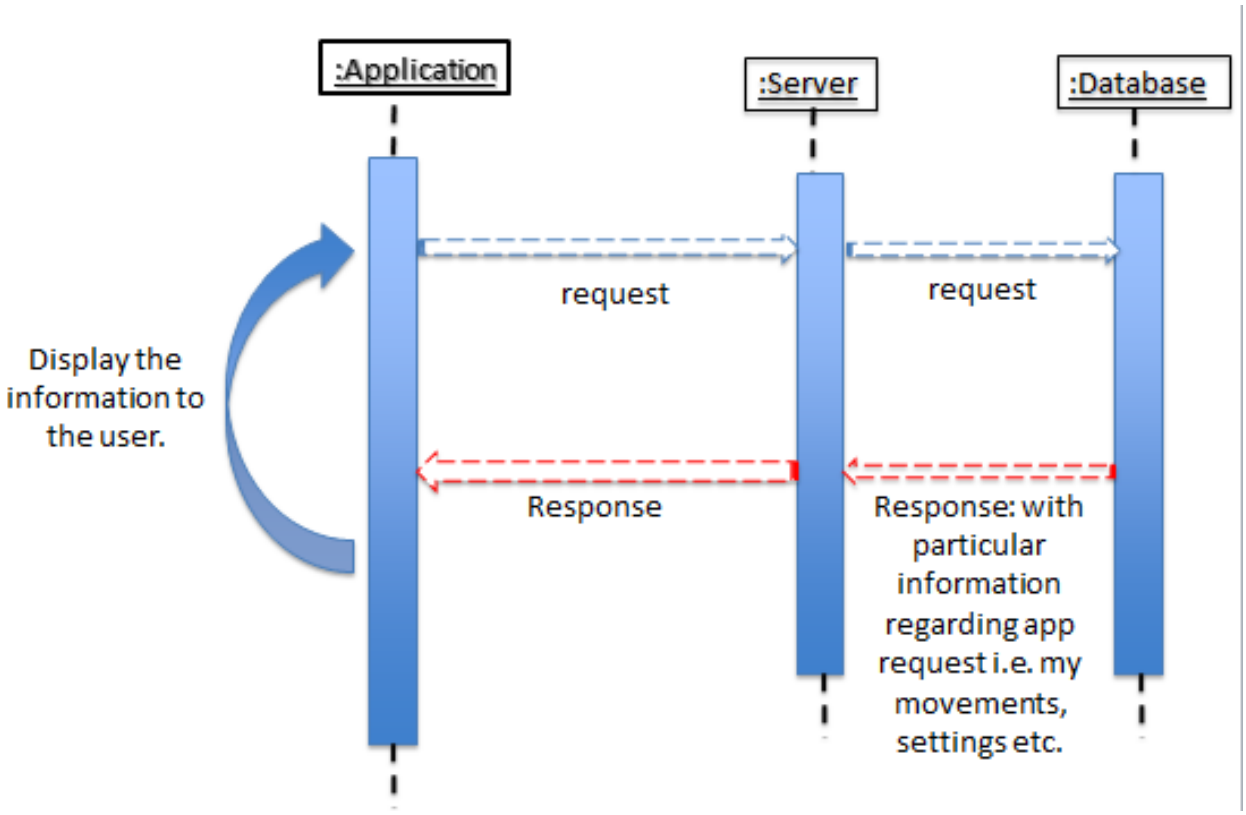
This diagram illustrates the process which takes place when the user wants to register for the application. The user fills out the information and sends a request to be registered. For every user, there is an object inside the UserClass which stores their information. The new user information is sent to the server and then the server adds the information to the database. The database responds back to the server with the response that the information has been added successfully. Then the server responds back to the UserClass with the response that the user has been created and then the user is redirected to the main user interface of the application.

2) User – Custom Workout Plan Creation Interaction:



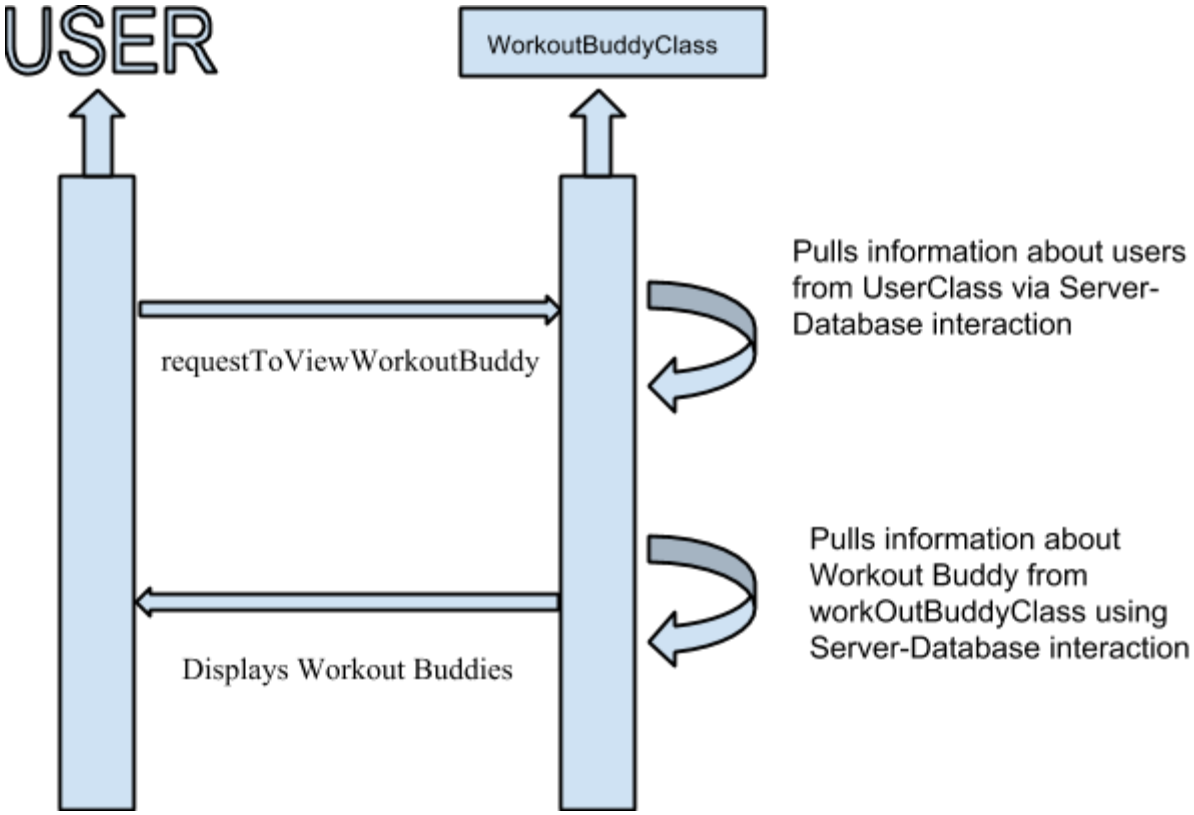
This diagram illustrates the process which takes place when the user wants to create a custom workout plan. The user clicks on the 'Create Workout Plan' button and adds the required information in the user interface. When the user clicks on save, the request is send to the Parse backend (this consists of the server and database), which saves the information and every workout is given is a unique workoutID. If the information is stored correctly then the user is displayed a popup which says that the workout has been stored and he/she is than redirected back to the main user interface page.

3) Server – Database Application Interaction:



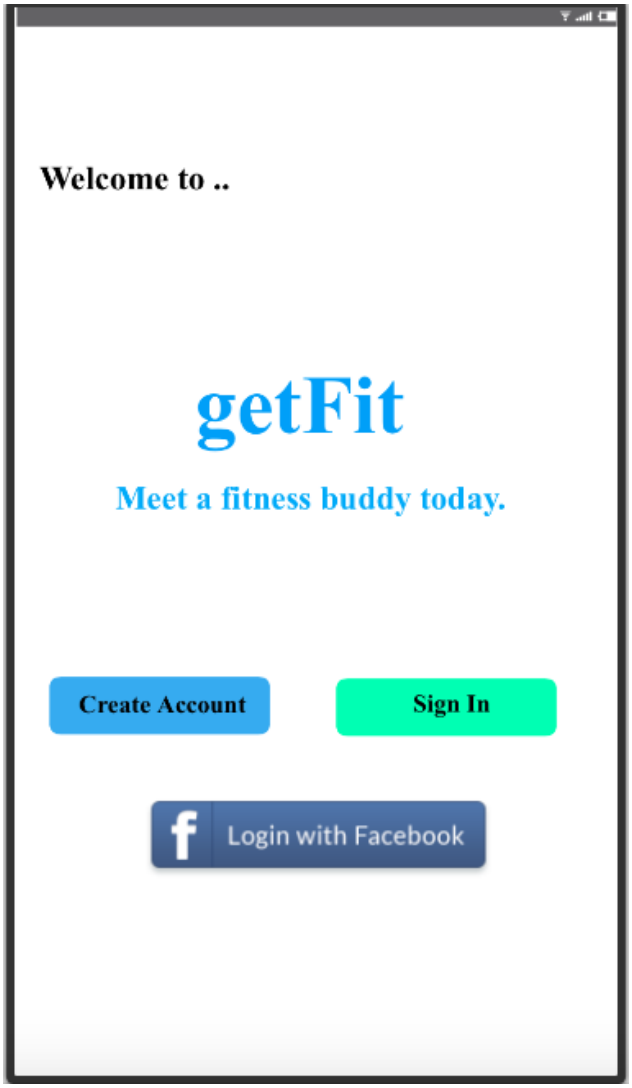
This diagram essentially describes the interaction between the server and the database for any given action that the user might make. For example, say that the user may want to view the users near by that are working out near you. The user taps on 'Find Gym Buddy'. The application sends a request to the server which in turn sends a request to the database. The database responds to the server with the required information of the list of users that have a similar workout schedule as the user. The server sends this information to the app and this displays the information to the user.

4) User – View workout buddies Interaction:



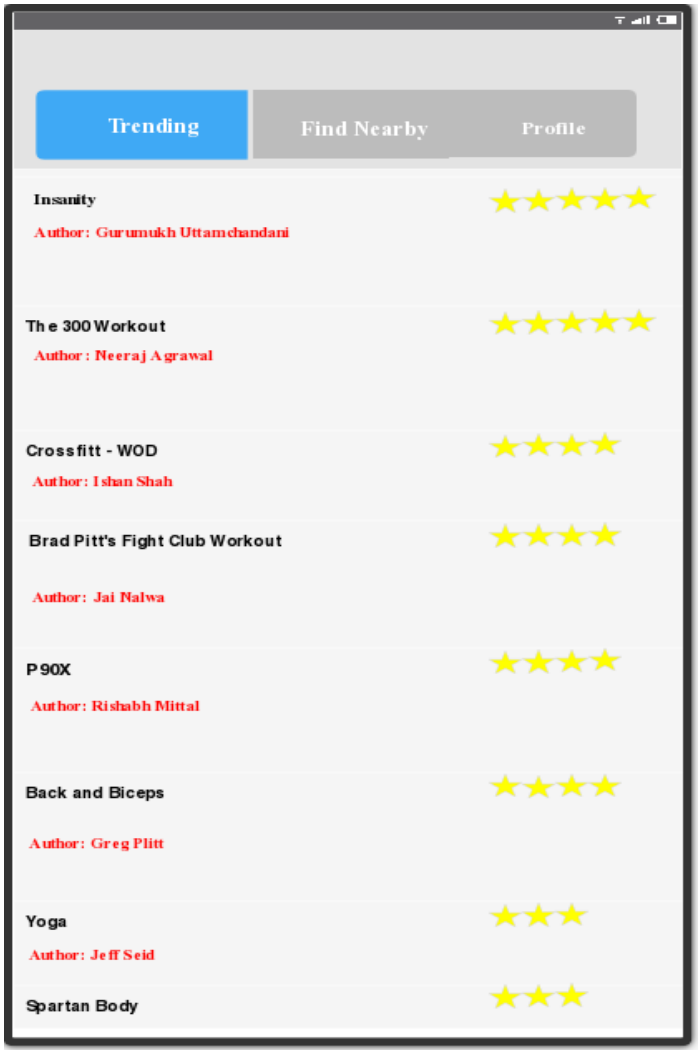
This diagram further explains the example presented in the description above. The user basically sends a request to view the workout buddies that have a similar workout schedule. The workout class responds by doing the actions shown in the diagram above and responds with the required information back to the user.

UI Mock-ups



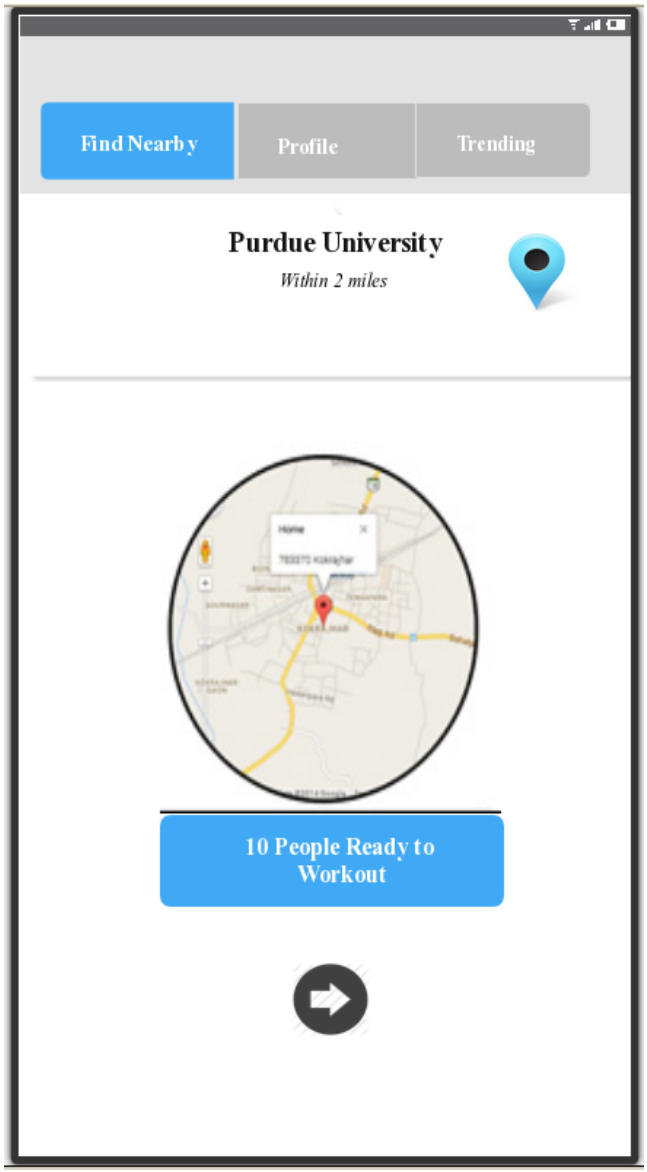
Welcome Screen

This screen should be pretty straightforward, it lets the user sign in to the system, or register if the user does not have an account yet (note that the server will have to verify the registration). The user will also have the option to sign in via facebook. The username entered will be a text field and the password field will be a secure text field in which the characters entered are displayed as black dots.



Trending

The Trending tab, displays the most highly rated workouts by the users. The user can change the tabs by swiping right or left. We chose to omit some of the description of the workouts as putting the entire information such as exercise, reps performed, etc. would create a crowded user interface to look at.



Find Nearby

The Find Nearby screen locates all the nearby users in the area. Once the user taps on the ‘Circle’, the client interacts with the server and finds all nearby users within a 5 mile radius. The user can then tap the button at the bottom, to see the list of people found.(Shown below)

Find Nearby Results

After the user presses the ‘Go’ button above, the client will connect to the server to get the list of all nearby fitness enthusiasts. Each row contains the User’s name, his/her profile picture, general information and when they were last active. Added information such as the nearby user’s workout information, his/her schedule and the option to chat will be available once the user clicks on a specific row. Like before, we chose to keep this information on a separate page, for a cleaner user interface.

Profile

The user can see and modify his profile information through this page. The user can choose to edit his profile, view his workouts and also view his schedule for the week. When the user taps ‘My Workouts’ he can view all the custom workouts he has created and also create a new one. Similarly by clicking on ‘View Schedule’ the user can view his workout schedule for the week and make changes if necessary. The screen also includes a sign out button at the top, which will redirect the user to the welcome page.