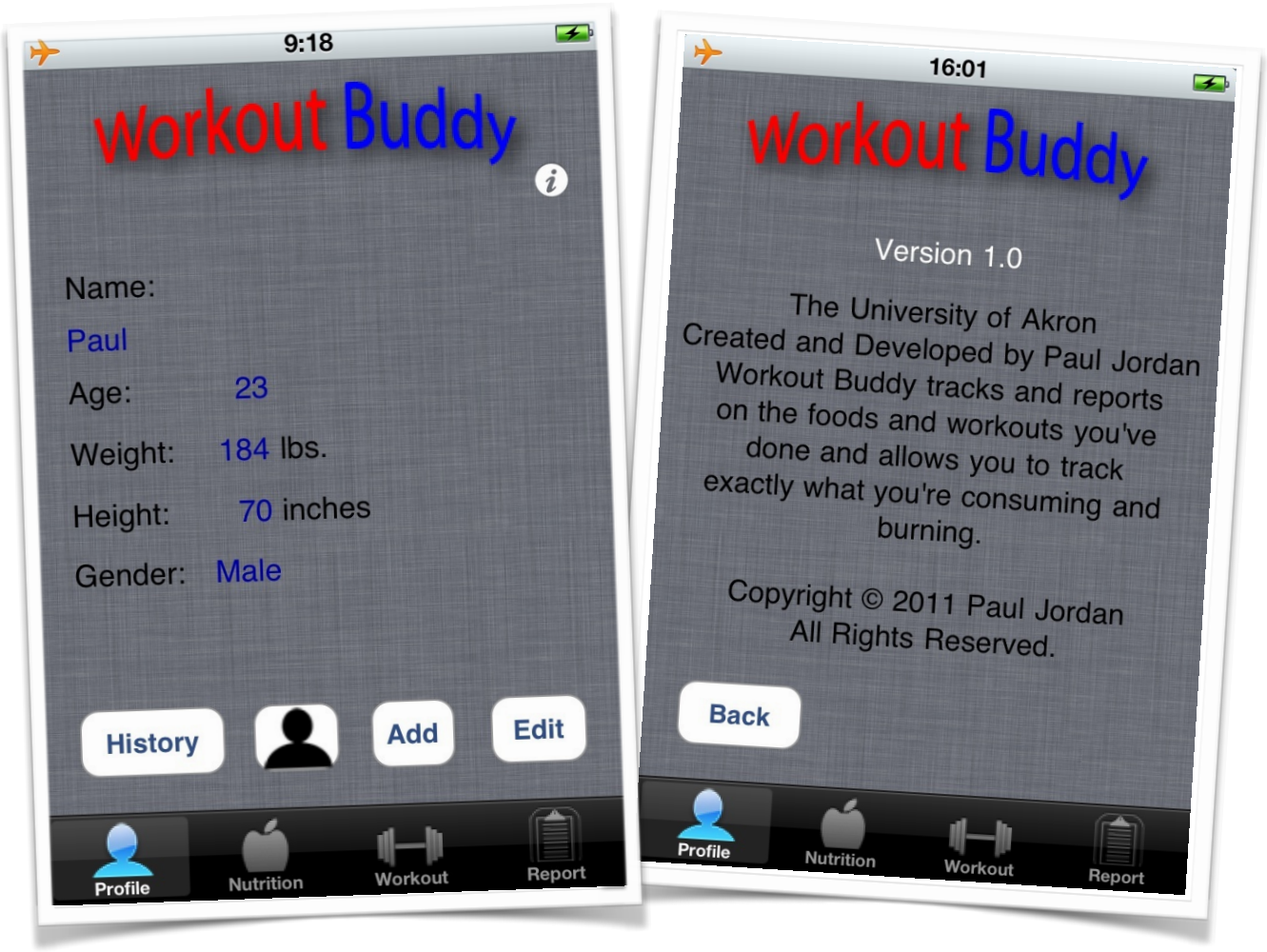
Workout Buddy

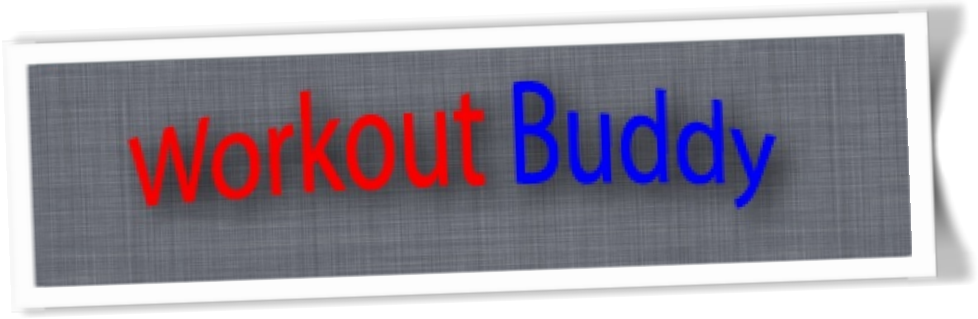
My first iOS application



Paul Jordan

April 26, 2011

Inception



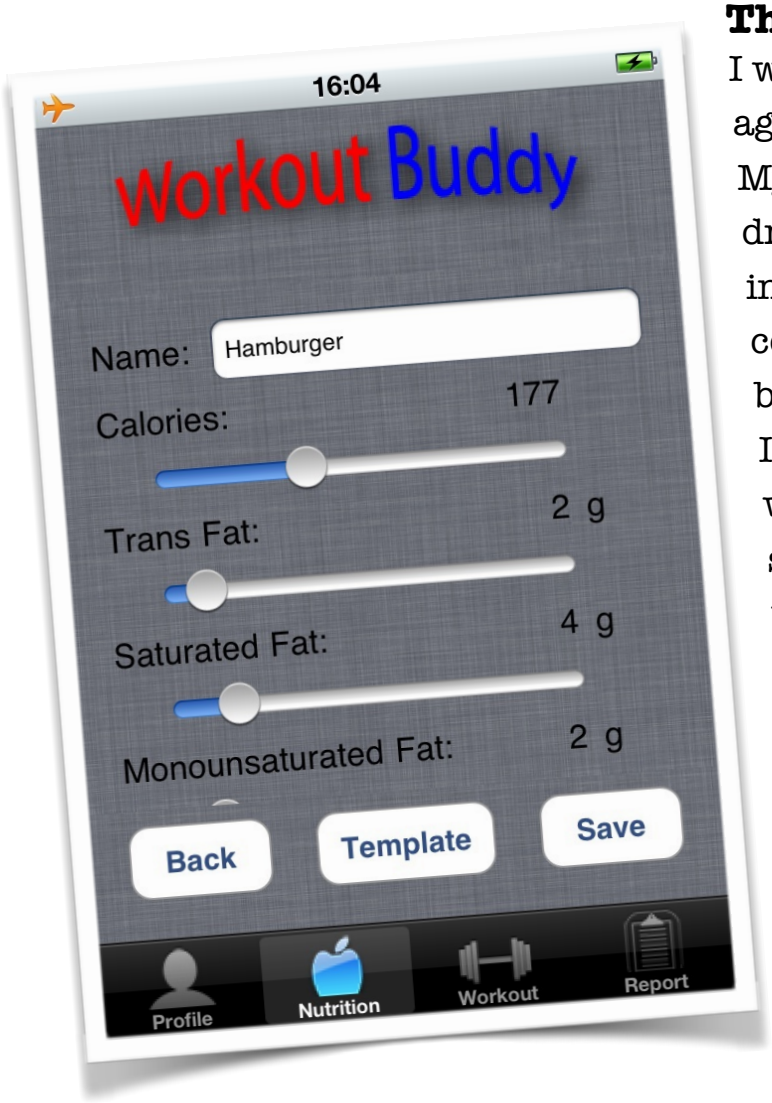
Why?

My idea for the application came from my enthusiasm for working out. I really enjoy working out and staying in shape, but it can be difficult to track all of the food that I eat, and workouts that I do. On top of that, even if I do write down everything I eat, then it becomes difficult adding up every category, and calculating percentages of daily values. This application allows a user to store the things that he or she eats often, and re-use them again later. It will then calculate percentages of daily values and report deficiencies in your diet.

Why iOS?

I chose to do an iOS app because of the portability. A lot of iPhone, and iPod users carry their devices on them for most of the day. It becomes really convenient to log your diet, and workouts on a device that you already carry. With the ability to add customized foods and workouts, you only need to enter information about the food once. Most people who attempt to track their diets fail because it's a lot of information to track. They end up only tracking calories, which is great, but there are many more nutrients, fats, vitamins, and minerals that your body needs.

Design



The design process

I went with an extremely agile approach to the design. My initial design was a hand drawn sketch of the user interface. I went through a couple of different designs before I found the one I used. I chose a tab bar interface which allows the user to switch between adding workouts, and foods, very easily. I designed all of the images myself in Adobe Illustrator. I chose a vector based graphics tool because I needed to be able to scale the images up and down. There are two resolutions supported in iOS. The retina display is much higher therefore I needed two sets of images. As for under the hood, I went with what would be easiest to code over what would be the most efficient because of time constraints, and my lack of experience with iOS, and objective-c.

Under the hood

I have an array of person classes that is read and written to "disk" before and after each access or change. It's not very efficient I know, but I needed to have a way for each UIViewController class to access

the data, and I didn't feel like I hade enough experience with SQL to store it in a database and access individual data members. Each view has a view controller class, and in order for that class to have access to the data I had to either create some sort of global array, or just read it in from disk (which in this case is flash memory) on each access. Each person class has a few describing data members like name, age, weight, height, and then an array of day objects. Each day object contains the date for which the object represents, an array of food objects, and an array of workouts. Each workout contains a double representing the rate at which calories are burned, and an integer representing the length of the workout. Each food object contains "doubles" representing each of the nutritional items listed on a nutritional facts label, and a string that represents the name of the food item.

The Learning Curve

Objective-C

Learning objective-c was my biggest hurdle. Thankfully I had plenty of resources due to the popularity of Apple's App Store. StackOverflow.com provided a lot of specific technical resources, while I used the Head First Development: iPhone development book for the initial development. My final and most helpful resource was on iTunes University. Stanford University offers an iOS development course at the undergraduate level, and for the passed two years, they recorded the lectures, and posted both the recordings, and the keynote slides on iTunes University for free download.

Objective-C is a very interesting language that offers a lot of really cool functionality. From my research, I've learned that objective-c is like small talk, a popular message passing language in Europe, mixed with C++. Objective-C doesn't make direct function calls, instead it passes a message to an instance of an object. If data is to be returned it's done the same way. This is cool, because it decreases dependencies. A class calling a method of another class doesn't necessarily have to know that the method or data member even exists. Another interesting feature of Objective-C is it's memory management. Like C++, the developer must manage his own memory, but unlike C++, if an object, or instance of a class isn't specifically retained, it will get released similar to Java. This creates problems if your trying to access something that has been released,

but is nice to not have to specifically release memory every time you allocate it.

Bugs

I learned a lot about debugging during this project. My first major bug was the applications inability to write to disk. Whenever the program terminated, all of the information stored would be lost. This was obviously very frustrating. As mentioned before, I opted to write all of my data to disk at once. Therefore, all of my data had to be serialized. In Objective-C, this means that all of my classes had to implement the NSCoding protocol so that the data could be encoded, and decoded for the write and read to and from disk. This wasn't the problem. The problem was that each iOS application is allocated it's own space in the file system, and can only access it's own directory. This is great for security. Inside that folder, each application has a documents directory used for storing data. The directory string I was using wasn't in the right format to write to disk. So whenever the application went to save to disk, it wouldn't return any errors, it would just act like it was written successfully. After reading multiple example programs I noticed the way they were telling the program where to write was just a little differently than the way I was doing it. After fixing my directory string, everything worked just fine.

My next major bug was in my memory allocation. As I mentioned earlier, my day class had two arrays, one workout, and one food. After trying to enter things into this array, I noticed that they weren't actually being saved. I thought it was in the way I was writing to disk, but the profiles were saving correctly. Next I thought maybe it was just because I was doing the user interface incorrectly, and the data

just wasn't being displayed, but I checked the array elsewhere, and noticed that it was empty. So upon reviewing my day class I noticed that both arrays were being initialized, but there was no memory allocation being done. Simple fix, but it took several hours to track that one down. That will definitely be the last time I forget to allocate memory.

Conclusion

What I learned

This was definitely a learning process. I learned a great deal about not only Objective-C, but also a lot about design, and higher level programming in general. Until this project, I had never done a lot of work with User Interfaces, and it really introduces a whole new aspect of design. There's a lot more thought that goes into how the user interacts with the program. For example, I had to use sliders for integer input to ensure that an actual integer would be returned. A developer must make sure the application flows well, and that the user won't struggle finding the functionality that he or she is looking for. I really enjoyed looking at this side of a program.

I learned a lot about memory management, and tracking down bugs. I had never really worked on this scale with a language that required memory management. It was very frustrating seeing Objective-C's version of a segmentation fault called EXC_BAD_ACCESS when I had no idea why. It made me appreciate Java's garbage collector a lot more.

Finally

Overall, I learned a lot about all aspects of working on a slightly larger project, and I had a lot of fun doing it. On top of everything, I now have an app for my iPod touch that does something that is very useful to me.