Developing a native iOS app

Making a cycling and running tracker

Willem L. Middelkoop

May 11, 2024

As a little side-project, squeezed between my normal work, I have been working on something of personal interest: a native workout tracking app for iOS. I wanted to make my smartwatch obsolete, instead using my phone to track workouts. How hard could it be to gather detailed sensor data using native Swift APIs?

Background

If you have been following my blog, this may not come as a surprise, but I love to ride my bike and have recently started to do some running. I have quite some experience with tracking my workouts, using Garmin, Strava, Omata, Wahoo, Biostrap, Whoop and various apps and sensors. I find health and fitness data useful to learn about your physical performance and to have some kind of accountability for yourself (e.g. motivating you to keep doing the workouts).

Nothing makes you feel alive like going outside for some physical action - I love working out!

The nerd in me loves the big data set, the many different available sensor types are a feast for my curiosity - yet it conflicts with my desire for minimalism and simplicity. Do you really need all the workout data? Isn't just listening to your body enough? I don't know, but I do appreciate a smartwatch's capability to (nearly) effortlessly track workouts in great detail. I like how Apple Watch can be used to track training-sessions, next to its capabilities as modern tool watch and smartphone replacement. While I understand most folks would wear their smartwatch and call it a day, I do like old fashioned mechanical watches like a Rolex, Tudor, Grand Seiko, or even something custom made. Although I appreciate both, I do gravitate towards mechanical watches when I have to choose between smartwatches and mechanical watches.

Idea for the app: Gran Fondo

What if there was a smartphone app that would collect all the workout data you want, without any of the hassle and clutter? I imagined it to be:

- **Simple**: it must get 'out of your way', designed to 'just work', without the need to fiddle with endless settings or options. Just press 'start' and begin your workout.
- **Flexible**: Use the app in different ways: mount or wear your phone in plain sight for realtime data, or put your phone in your pocket and have it work in the background.
- **Powerful**: Connect with heart rate, cycling cadence, power and speed sensors using Bluetooth; support multiple sensors, ideal if you use different bikes.

- **Apple Health friendly**: Save all data in Apple Health in maximum detail, including workout routes and individual power, cadence and speed measurements. This enables you to analyse recordings in any app that supports Apple Health, like Strava, WHOOP, HealthFit and many others.
- **Honest**: No ads or data selling. No accounts or registration. Your data should stay private, secure and on your device.

None of the existing smartphone sport and fitness apps tick all of the above. Most of them are needlessly complex or are somehow designed to take your data to serve the app builder or its platform. Could I do better?

Building the app

Knowing why and what I wanted, I set out to choose my preferred method of building the app.

Native vs Hybrid/web apps

For most work I do, I prefer to use open and free (as in libre) web technologies, like HTML, JavaScript and CSS - or server software released under a free software licence, such as GPL. For this particular usecase I needed to be extremely energy efficient and to be able to deeply integrate with my smartphone's operating system. The best way to do this for an iPhone is to fully embrace Apple's software development kit: making the app natively in Swift using SwiftUI and by following the technical and design guidelines as closely as possible.

AI assisted programming

Although I have my reservations, using AI for programming has all the attention lately. Some even claim that technologies like Copilot or ChatGPT would make programmers obsolete. I know my way around AI, as I have used it to generate code, as I teach folks how to use it, but I never used it to build a *complete* iOS app. I figured this would be a perfect opportunity to see how fit AI is for something more complex than a simple script snippet. Let's find out how obsolete I am as a programmer in this modern AI-age...

GPS and Bluetooth Low Energy Sensors

Having setup my Xcode environment, I set out to ask OpenAI's ChatGPT to give me some code to read GPS location data and to generate the necessary Swift classes to detect and connect with Bluetooth sensors. Boy, did I enter the proverbial rabbit hole here!

My bike with Bluetooth sensors connected to my MacBook running Xcode

The thing with AI-assisted programming is that, **YES** it works; but only **PARTIALLY**. It will give you working pieces, but they will not fit perfectly and do not necessarily work together. Seeing the code it generates is like hearing echos of a great song; you'll recognise it resembles something - but it is not quite the same as hearing the band play live music; let alone sing it yourself!

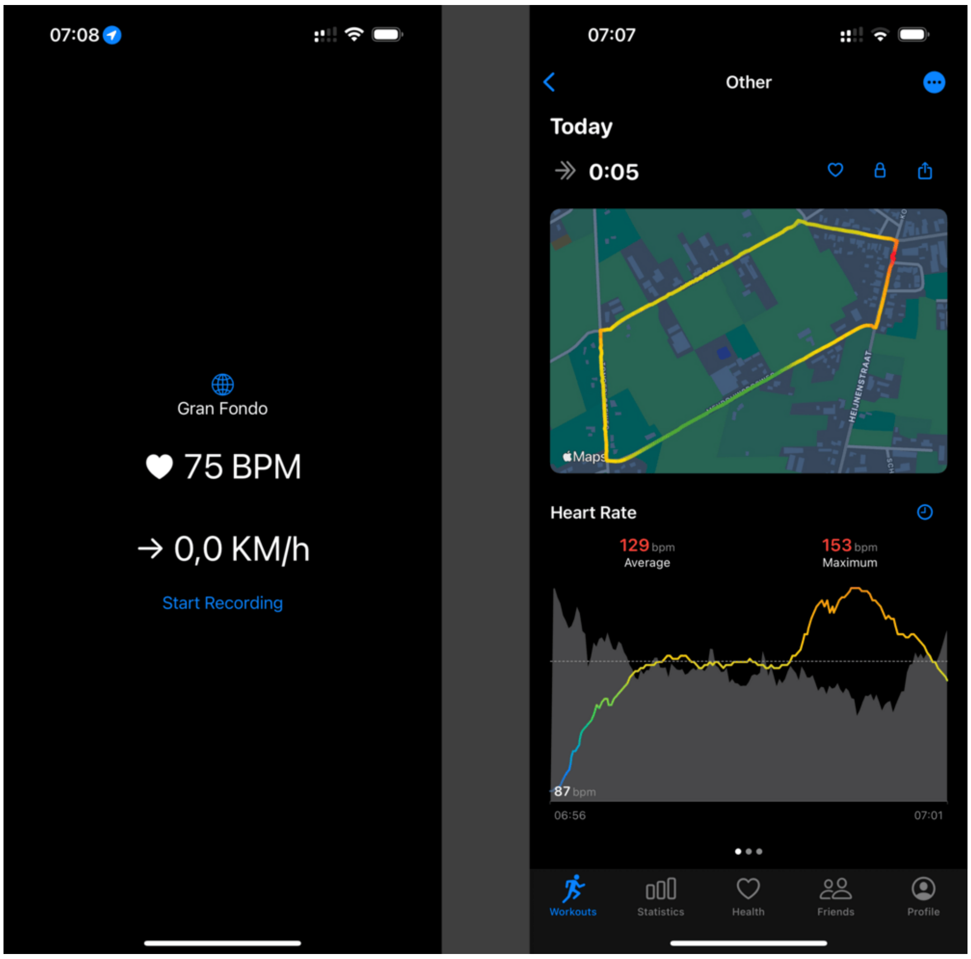
I took the pieces from the AI and set out to read the relevant documentation myself, this proved to be much more effective as it enabled me to actually understand what the mechanism should do. The AI helped me find the things I needed to learn. Tracking GPS location data was surprisingly easy using Apple's CoreLocation. Getting the Bluetooth sensor to connect to the app turned out to be much harder, as it involves many distinct steps:

- **Detecting Bluetooth devices**: scan the nearby area for any devices that are available and broadcasting
- **Connecting to a Bluetooth device**: making *and maintaining* a connection between the app and the device
- **Discovering Bluetooth device characteristics**: there is no such thing a simple Bluetooth device, you have to do your own 'discovery' of a device's capabilities and features; many devices broadcast multiple different signals for both sensor data, control mechanisms and operational parameters such as battery level.
- **Subscribing to characteristics**: Ultimately you subscribe to a data feed by tuning the Bluetooth radios to a certain Bluetooth characteristic.
- **Decoding data**: Given its energy efficient nature, you'll receive minimal data in raw bits; you'll need to decode data packets yourself; the hard part is that each sensor, by any vendor, has its own way of encoding data.

Take for instance a relatively 'simple' sensor like a cadence sensor that you put on your bike's pedal, it could be used to determine the revolutions per minute you make during a cycling session. How hard can this be? I figured that the official Bluetooth specification would provide me with all the needed details... NOT! A simple cadence sensor is surprisingly complex:

- **Combined Bluetooth profile for Cycling Speed and Cadence**: any cadence sensor **always** identifies as *both* speed *and* cadence sensor, you have to discover its capabilities 'on the fly'.
- **Non-similar data packets**: The contents of the actual data depends on the sensor's current abilities (e.g. it detected something), you'll have to interpret the data packet, bit for bit, by checking for possible flags indicating what the next data bit represents.
- **Non-trivial data**: Don't expect a clear cut value for ”cadence”, the sensor will give you a cumulative revolutions count **and** a rudimentary timestamp of the last detected change; you'll have to use these values to calculate the metric you're trying to measure.

The AI provided me with some sample code, but it proved not to work with my particular sensor. Using a Bluetooth debugging app, Bluefy, I was able to gain a better understanding of the data the sensor was transmitting. I found a piece of Java code on the official Bluetooth blog that gave me some clues on how to interpret data from the sensors. I adapted this piece of code to match various different sensor profiles and translated it into Swift code. This gave me an early working prototype.



Early prototype of the app working with Bluetooth heartrate sensor and its early dataset shown in another app that makes some pretty graphs (HealthFit)

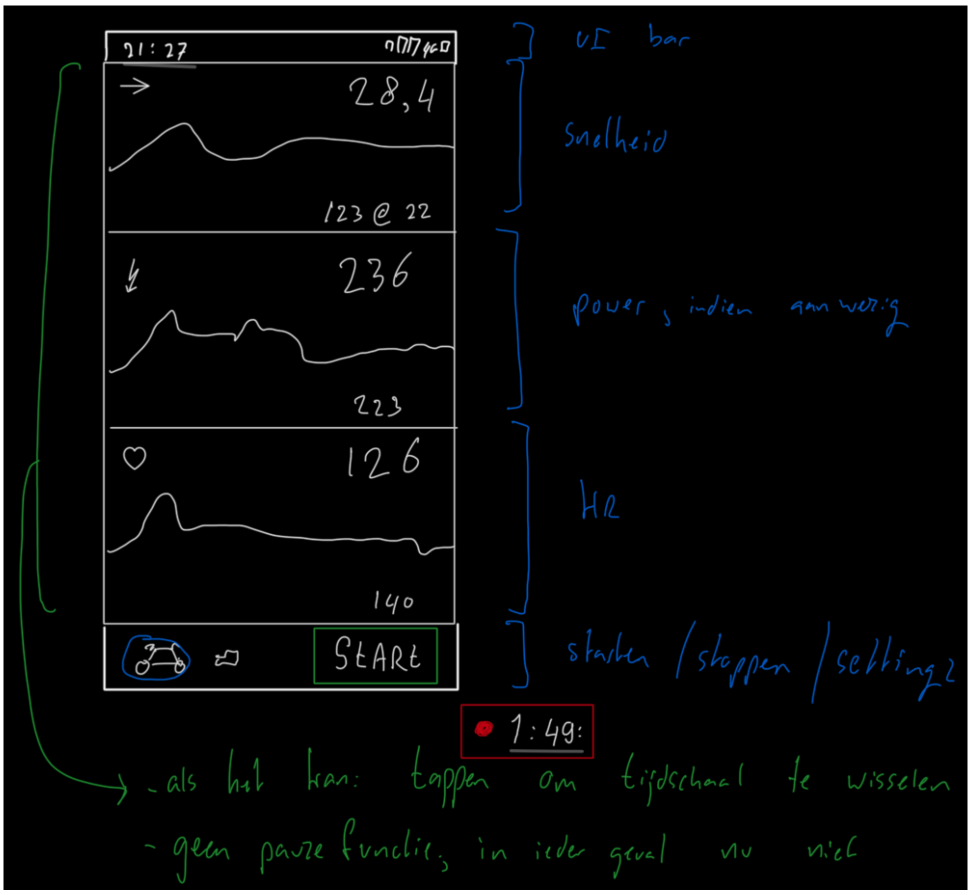
UI/UX Design

One of main reasons I like the Apple Watch workout app is how its user interface (UI) is designed to facilitate a pleasant user experience (UX). The app on the watch has fairly large buttons that you press to start a recording - *and that pretty much is it.* The app itself has few options, its design is opinionated but well thought through. It is a stark contrast with devices like Garmin that offer a ton of options for customisation. The trick is to strike a good balance between flexibility in usecases *and* having a default setup that works well.

Apple Watch workout app shows large buttons to start a workout - no need for additional configuration

Using a mounting system like QuadLock, you can firmly attach your smartphone to your bike's steer. This means that the iPhone can replace your Garmin as bike computer. Modern iPhones have an always on display that can show information in an energy efficient way, provided you use the Apple software development kit that comes with various optimisations built-in. After many, many, kilometers with my Garmin bike computer, I settled on having a view with realtime data fields for speed, power and heartrate. What I like about the more advanced Garmin Edge devices is that they can plot a graph. This enables you to keep your eyes on the road when it matters, and glance at historic data at a (slightly) later stage.

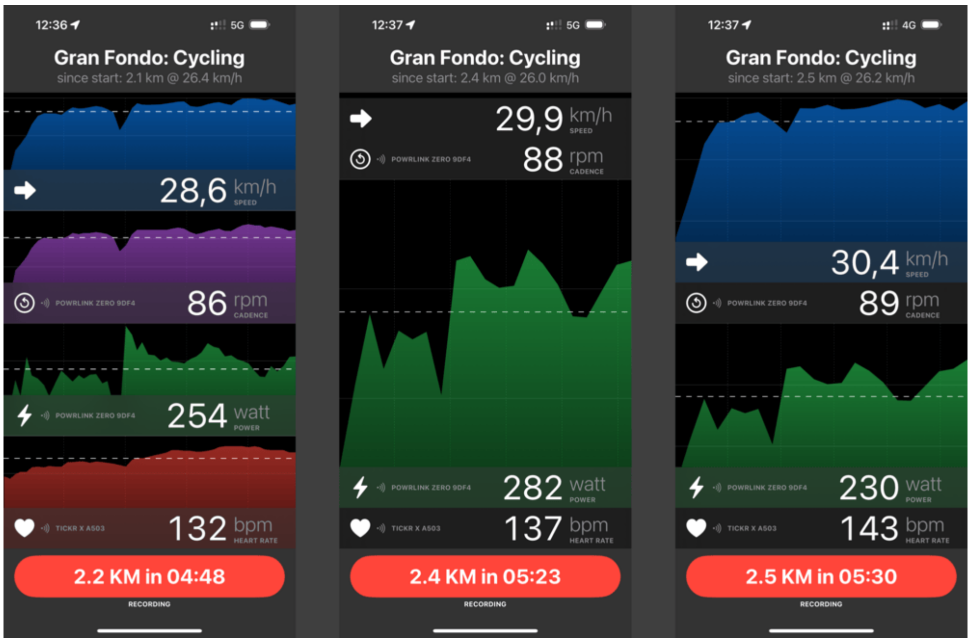
I used my Garmin Edge data layout as an inspiration when designing my app's layout when used as mounted bike computer


21:27
28,4
123 @ 22
236
223
126
140
START
1:49:
UI bar
Snelheid
power, indien aanwezig
HR
starten / stoppen / settings
- als het kan: tappen om tijdschaal te wisselen
- geen pauze functie, in ieder geval nu niet

Design sketch describing the desired layout

Testing the app on my bike using a QuadLock case and mount

To maximise energy efficiency, I decided to utilise Apple's own libraries for graphs: Swift Charts. One problem I encountered while developing this, is that measurements from sensors come in at different intervals and times. You need to create some kind of synchronisation mechanism to have data points matching common timestamps in order to compare and relate values. I designed this mechanism to synchronise UI updates, too: enabling all graphs and data points to be updated in one single refresh; optimising for CPU and graphics performance (and thus battery power). As a bonus, I built in support for different time scopes, enabling the graphs to show data for the last minute, 5 minutes, 15 minutes, hour or entire workout.

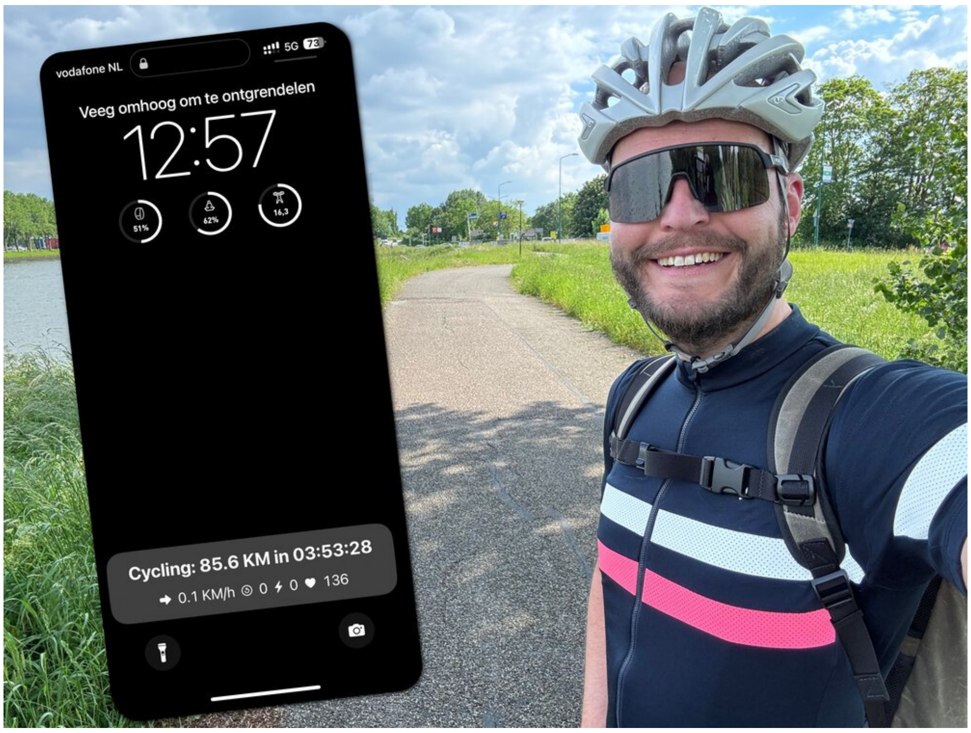


Different configurations of realtime data

The app automatically shows the available graphs based on connected sensors, alleviating the need for different setups when you have more than one bike. If you tap a graph it maximises or minimises. If you tap the data scope (e.g. ”last 5 minutes”), you can toggle between scopes. The app remembers your configuration, automatically setting you up correctly for the next workout. Because all the touch targets and buttons are fairly large, they work well when your hands are either sweaty or if you wear gloves.

Pocketability: Optimising for background usage and energy consumption

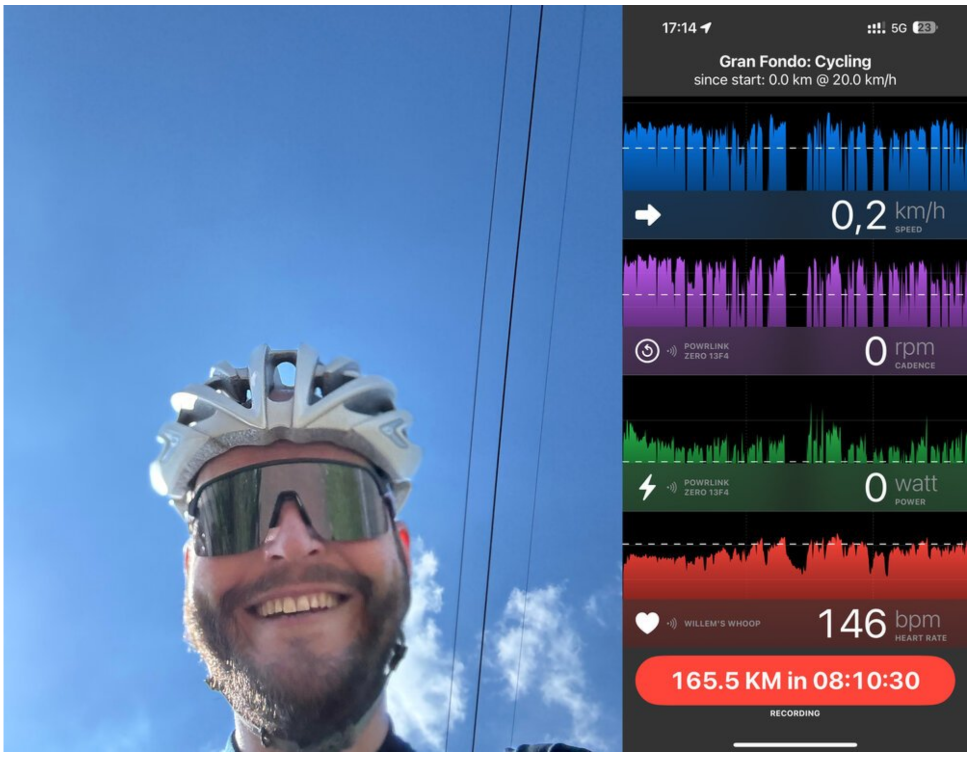
An alternative way I envisioned to use the app was to use it from my pocket, as background app. This would enable low profile data collection, where you start the app and put your phone away. I love the idea of having a 'complete dataset' for *all* my rides and runs, while not being required to have the app be center staged *all the time*. Many training runs and rides aren't *that* interesting.



I got some magic in my pocket: The app tracking workouts from the background - during recordings it just shows a widget on the homescreen

By default, Apple's iOS operating system strictly limits the options apps have to run in the background. This maximises battery life and user privacy. If you follow Apple's guidelines and correctly ask the user for the required permissions, you can in fact design your app to work perfectly from the background. I really came to appreciate Apple's efforts on optimising the entire stack, enabling your app to be more energy efficient. Simply put, *"don't call us, we'll call you"* is what the operating system does: it takes control over *when* your app does its thing - enabling the OS to synchronise, combine and batch process different tasks on the available (energy efficient) cores in the Apple Silicon chips.

During one of my tests with a prototype version of the app, I was able to collect over 127 thousand (!!) data points during a 8 hour long distance ride on a single iPhone battery charge, with the screen enabled *all of the time.* If you play by Apple's rules, you really have *the power* (so to speak). Developing this app really took me somewhere, ha!



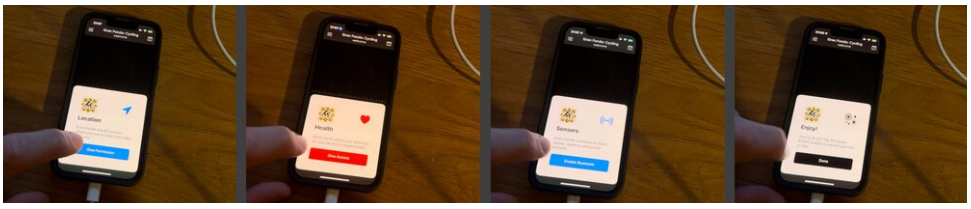
"Goin' the extra mile": 8 hours on my bike to test battery usage (note: there is still 23% power left!)

Real artists ship

The road to hell is paved with good intentions, they say... For this reason, I really like to finish my projects. For an app that means that I should have it published in the AppStore. Getting it ready for Apple's scrutiny, having it approved by the AppStore reviewers is an extra milestone. It is important, because it makes you think about the onboarding (of new users) and the earning model.

Onboarding

One particular challenging task when building an app is to guide folks that are new to your app on how to use it. A great user interface can help, but given the app's deep integration with Apple Health, Location data and Bluetooth sensors, there is an additional need to politely ask for permissions.

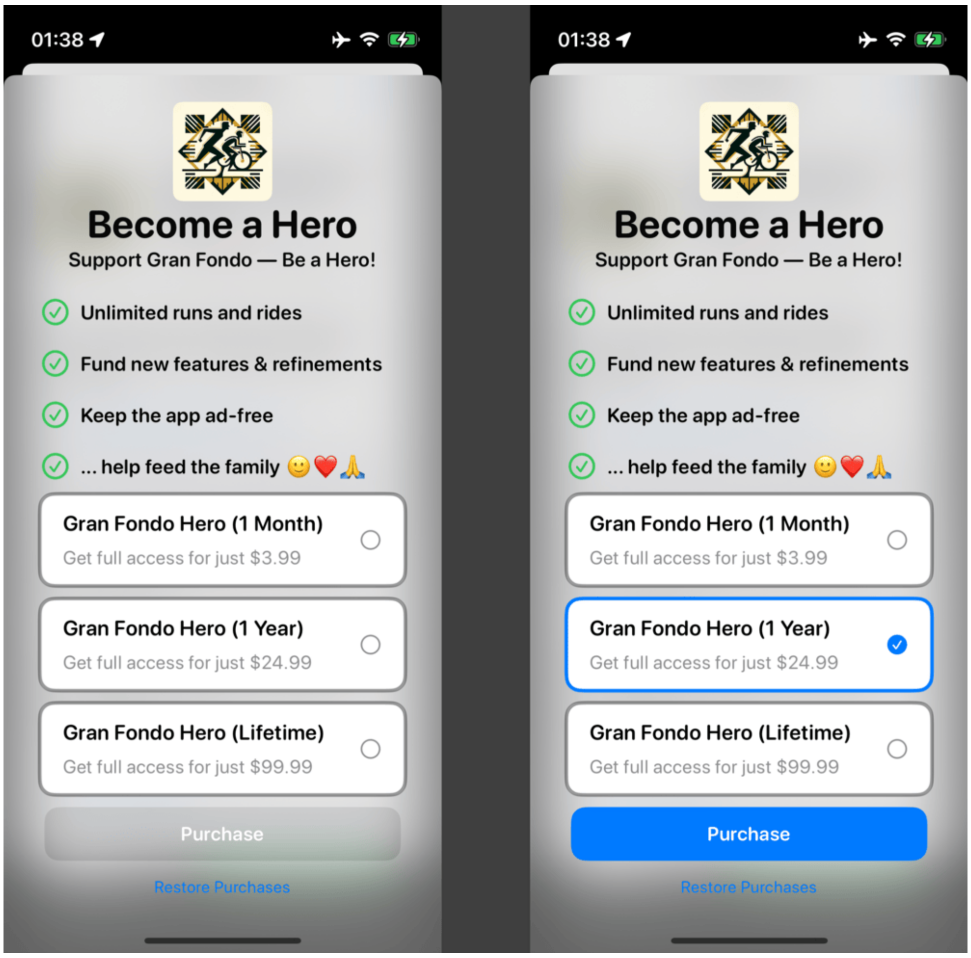


Screenshots of the app's onboarding process: asking for permissions and guiding new users

You will have to design the process so that it does not fail when people reject certain permissions. Apple is very strict in this: people have the right to reject. It is your job to make the app work with no or partial permissions. This is an extra challenge.

Earning model to fund development

While I cannot overestimate the importance of free software, as in libre, I cannot ignore that I have a family to feed. But, moreover, finding people willing to pay for your product can serve as an extra validation of your idea. Be open and honest about your earning model and it will help you fund your own project; don't rely on investors or venture capital, bootstrap your own development!


01:38

Become a Hero

Support Gran Fondo — Be a Hero!

- ✓ Unlimited runs and rides
- ✓ Fund new features & refinements
- ✓ Keep the app ad-free
- ✓ ... help feed the family 🙂❤️🙏

Gran Fondo Hero (1 Month) ○
Get full access for just $3.99

Gran Fondo Hero (1 Year) ○
Get full access for just $24.99

Gran Fondo Hero (Lifetime) ○
Get full access for just $99.99

Purchase

Restore Purchases

01:38

Become a Hero

Support Gran Fondo — Be a Hero!

- ✓ Unlimited runs and rides
- ✓ Fund new features & refinements
- ✓ Keep the app ad-free
- ✓ ... help feed the family 🙂❤️🙏

Gran Fondo Hero (1 Month) ○
Get full access for just $3.99

[✓ Gran Fondo Hero (1 Year)]
Get full access for just $24.99

Gran Fondo Hero (Lifetime) ○
Get full access for just $99.99

Purchase

Restore Purchases

Designing a paywall for my app: not my favourite ask from users, but it is a very important one nonetheless.

You do not need to pay in order to try the app. The payment screen only shows up when folks actually use the app (and it can always be dismissed). Why not try it yourself, you can find the app here.

Future

Eat your own dogfood, they say - I will continue to develop the app by actually using it myself. I don't expect this project to be a quick (monetary) win, but if it can be of practical use for myself, chances are that some other person may find it useful, too.

Not your average debugging session: Fietselfstedentocht 2024

One thing I like about this particular project is that it takes little effort to keep it going as I like working out. Debugging new releases of this app often involves going out for a running or cycling session. **Who ever said software development was a sedentary job?!**

Conclusion

A lot of folks have 'great ideas' - few persons actually go out to have them realised. This distinguishes the makers from the rest. I like to make things, as the process of doing so requires you to really get into the matter. You'll learn a great deal and gain experience that others won't. Heck, build it yourself, make it your own!