How to Build a First Product, Project or Software Portfolio

[](https://hackernoon.com/@__r?source=post_header_lockup)

[\_](https://hackernoon.com/@__r)Follow

May 24, 2018



Credits: Tirachard Kumtanom

*I want to develop a polished product, and begin my software portfolio… where do I start?*

**P**rojects are a great way to improve your competency as a developer. The reason? They help you learn real engineering, gain credibility, and bring your ideas to life. In this short guide, I’ll provide some ideas on how to get started, move forward and build your first product or project in a portfolio.

Keep in mind as you go along:

*1. The more genuinely interested you are in working on an idea, the better it turns out.*

*2. In your career, you yourself are the product. That means the accumulation of all the things you do, write, say, reflect on and build.*

**tl;dr?**

Think of something that you personally would like to use.   
Decide to build a website or mobile app.   
Pick a software framework.   
Create a small feature plan with a deadline.   
Start building. Track feature completions.  
Finish a minimum product.   
Ship it! Reflect.   
Iterate and Keep Going!

**1. Starts with an Idea 💡**

So you need an idea?

Think of a market of one and ask yourself:

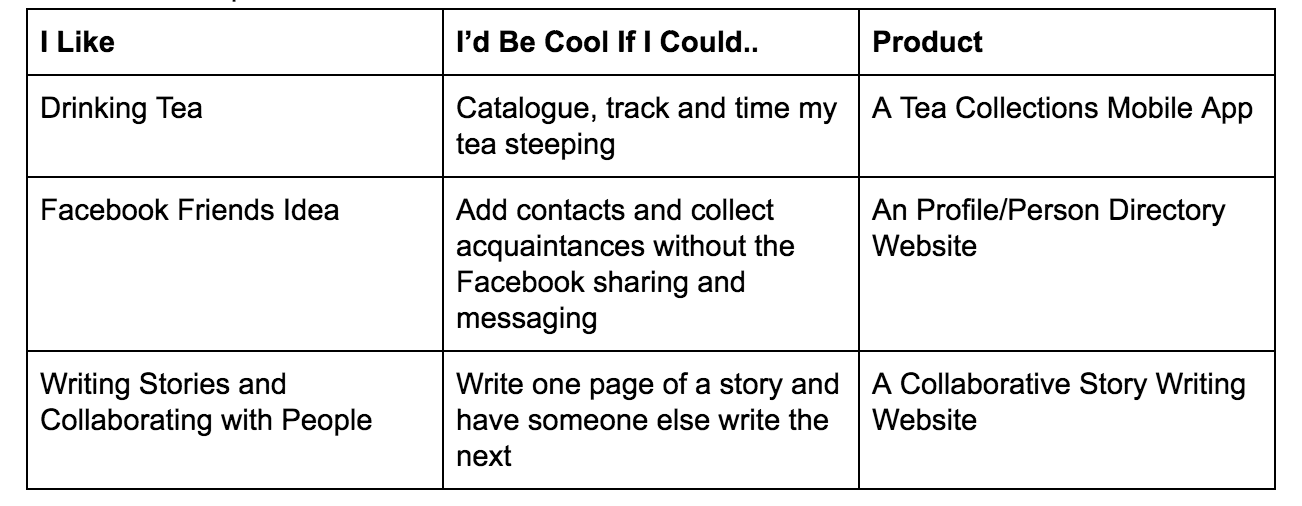
*What’s a tool or idea that I’ve always wanted for myself?*

The reason you think of yourself is that you’re accessible. You know your likes, dislikes and pain points on the daily. Secondly, the chances that there are people in the world similar to you is fairly high. Therefore, whatever you build is likely to find at a home or interest with least one other person.

Solving a real world problem (at any size) gives you something more readily ‘shippable’, which demonstrates an ability to take:

*Idea 💡 → Design ✏️ → Product 📱 → Ship 🛳 → ???*

Your initial idea doesn’t need to be grandiose, but it should be something that you personally think is interesting to use.



Examples of projects.

Don’t try building another TODO app, or “Clone x in y language”, these types of projects are not inspirational and will likely be left unfinished.

**2. Designing the Idea ✏️**

For a lot of developers, it’s right to the code. However, learning the process outside of code can be as fascinating as the code it self.

Before heading off to the binary jungle, try put a designer hat on and ask questions like:

What is the core concept or task I am trying to accomplish with this product?  
Who would be using this? Why?   
What old systems or tools might this product be replacing?   
What kind of features would be important to someone using this?  
What might this look like?   
Does this make sense as a website, Mac app, mobile application?

This can be as free form or structured as you want. But I do recommend grabbing a pen and paper to:

1. Mind Mapping the core concept to identify deeper concepts 💡
2. Answers the questions literally. ✏️ to 📄
3. Doing 2–3 free form napkin sketches of the the product 🙇‍

**This work is throw away style**, it’s process is to distill your idea further and narrow down what you want to build. Don’t get obsessed with the details.

**3. Getting Technical 🤔**

Once you have a concrete idea and some sketches, it’s worth starting to think about the technical side.

**3.1 Websites 💻**

**Full Stack Web apps**are the types of websites we interact with everyday. A basic [**monolithic**](https://en.wikipedia.org/wiki/Monolithic_application) full stack app, in *very simple terms*, consists of 3 main components:

1. **Front End Interface**: An interface that allows users to manipulate the data, view pages and retrieve resources from the application.
2. **Backend Logic:** the logic or code that manipulates data and interacts with the database
3. **Database:**The place you store your data persistently

If you’re building a website it’s easiest to use a web framework.

A **framework** is a set of tools and software created by other developers to automate mundane tasks, reuse common logic (like database manipulation), and generally speed up the process of development for software projects.

For Full Stack apps some simple frameworks include:

**Try not to becoming obsessed with the “best” technology to use and focus on learning just one.**

**Aside:**Perhaps one of the only reasons to shy away from full stack web apps at first glance is the actual deployment process. In recent times platforms like Amazon’s AWS and Digital Ocean have given developers cloud computing tools which are powerful but intimidating.

**3.2 Mobile Apps 📱**

My personal bias is that mobile apps are more straightforward to get started with compared to web apps as a beginner. Most of the distribution is handled, and all APIs are included in the platform SDK. It might be worth while considering starting off your first project as a mobile app.

\*avoid React Native to start with it. It can get a little complicated quickly when things go wrong.

**If you’re having trouble deciding, just pick the platform you have a phone for and the tools to build on.**

**4. Building and Managing the Process 🙋‍♂️**

Woo! We’re off to building this thing. So why make a plan or process?

Well, when we create a plan we make our idea more concrete. Adding process makes the building phase more deliberate and thoughtful with deadlines we discipline ourselves to.

Additionally, when you plan out a few features, and create a system around it, you can easily pick up where you left off in the road map.

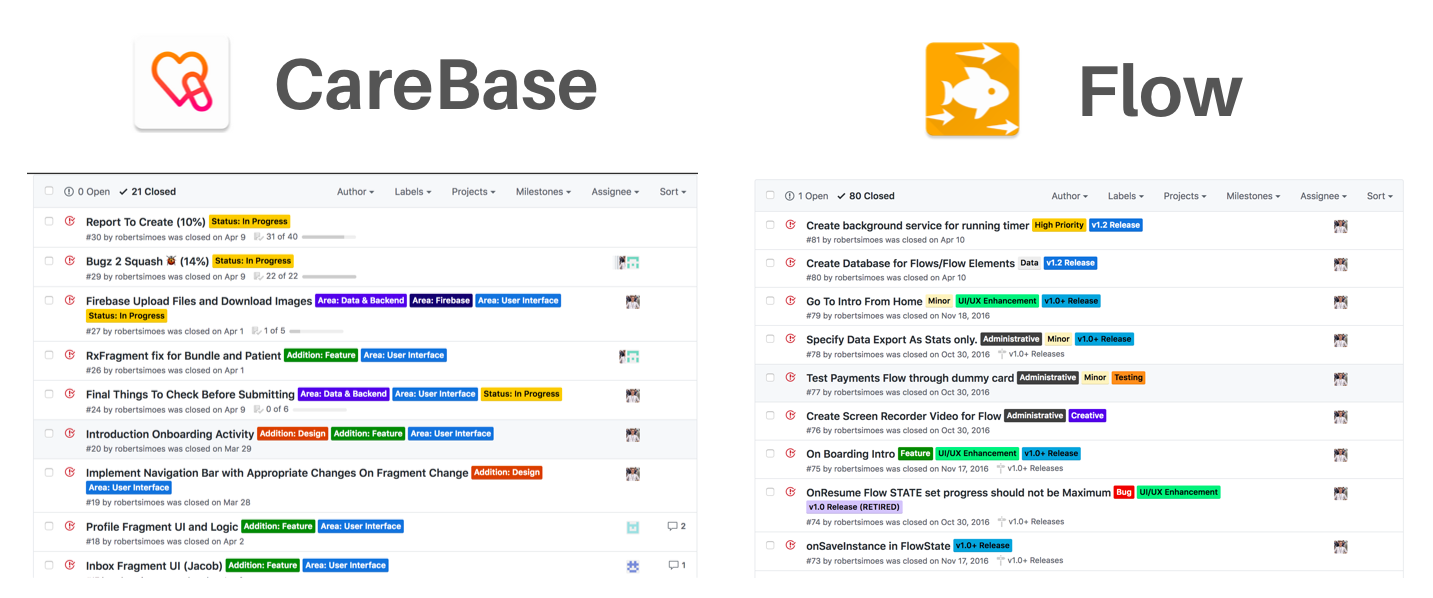
**Context penalties**are the price you pay to get “back into the zone” after stopping a task and coming back to it at a later time. These penalties are**huge**costs in software development, but can be mitigated with a plan or road map.

Finally, planning improves your accountability and awareness to keep yourself on track — which translates well to leading teams be it technical, design or product teams.

**4.1 Planning and Features**

I’d imagine this is certainly not most developers desire or forte, but this is the part of the process that I really enjoy, where you cast your vision and start making it come alive in feature ideas with tasks attached.

It can be as simple or complex as you’d like, but I recommend just using GitHub Issues to track features and Bugs. Below is two examples of projects I created and tracking the process with GitHub Issues.



Even creating two simple tags: **Feature** or **Bug**, and writing down the features you want to build before building, can save a lot of headaches.

**4.2 Setting Deadlines**

**Setting deadlines for the project is critical.** The reason is that without a deadline you can keep putting off shipping it out to the world. Believe me, it’s never “quite good enough” and there’s always “one more feature to add”, or better yet, eventually you “forget about the project” to work on something else.

Deadlines keep you accountable, and increase the pressure for you to learn and build. By formalizing a deadline, you can upgrade, downgrade and prioritize features, versions and bug fixes in order to hit the release. You’re deadline will be arbitrary, don’t worry. Just get a reasonable date sometime in the future and let that be your goal.

These are some rough estimates for starting out on a**solo** project:

Try to work in deadlines somewhere between 1 to 3 months to ship and publish a minimum build!

**4.3 Setting Criteria**

Make a set of criteria under which you would be *minimally satisfied* with the the product you produce and **will ship under**.

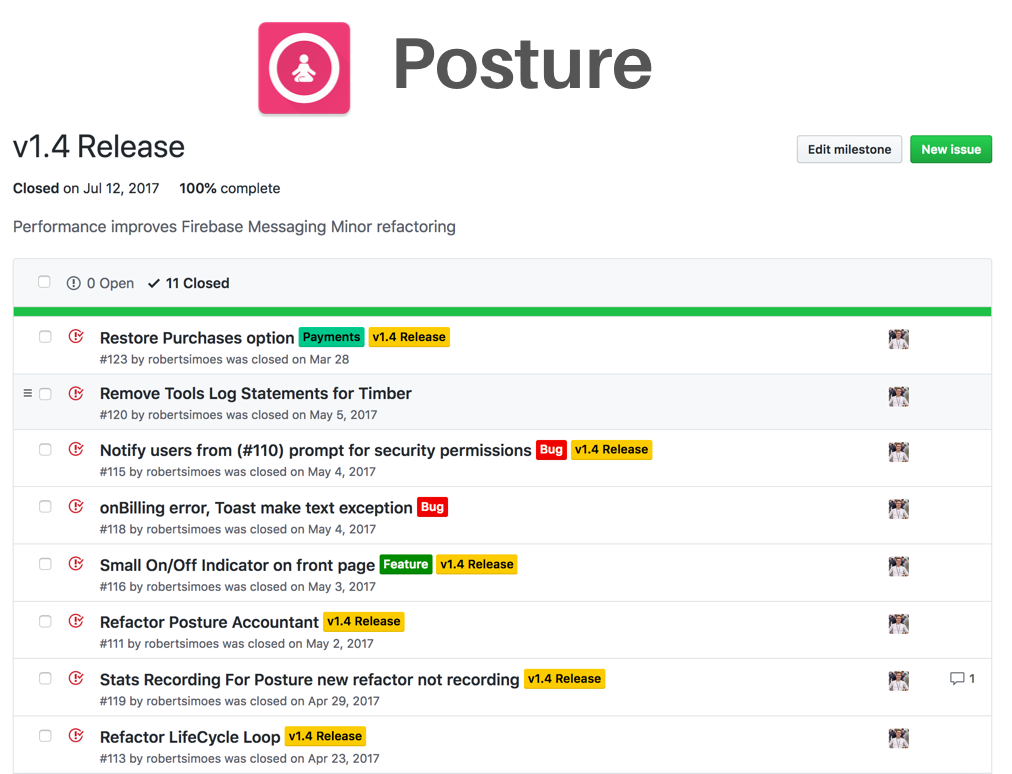
This could be a set of features, or a special feature milestone, under which a application will be considered a “release candidate”.

Example,

*Once we implement the database upgrade, and new profile design, we can ship it.*

**It’s critical to set a finite number of criteria** and not to let things keep creeping in. Otherwise you’ll delay release, testing, or worse, create potential upgrade and usability conflicts.

Here you can see a GitHub Milestone I used for a version 1.4 release I made for an app I made. You can see there is a deadline and set of issues and features I’ve tagged as criteria for this release.



Again this will be arbitrary but the point is to get used to the exercise, as you gain more experience you’ll improve and create your own process!

**How many features should I put in a milestone?**

5–20 depending on complexity

**5. Pre and Post Releasing Your Product**

***Shipping****is it art of publishing your project or product to the world wide web.*

Before shipping it’s important to understand the differences in how applications are released on the web versus mobile.

Web applications have a different set of procedures for deploying, releasing, and marketing compared to mobile apps. They don’t have a dedicated distribution channel, or rigorous standards to enforce. This offers flexibility but sometimes complication.

Mobile applications, on the other hand, tend to have more straight forward distribution channels such as the Apple App Store or Google Play store. This creates simplicity but more rigidity in the process.

Before shipping your product you should ensure that you test functionality of the app and squash any massive bugs. A minimum viable product will do, and get you going to keep building.

**5.1 Shipping Full Stack Web Apps**

Full Stack web apps have a slightly greater deal of deployment complexity. The main consideration is your infrastructure or hosting platform. Before things like Amazon Web Services (AWS), companies would buy their own servers or lease physical space from data centers.

However, nowadays services like [AWS](https://aws.amazon.com/), [Digital Ocean](http://www.digitalocean.com/?refcode=4acbf255c7e7), [Heroku](http://heroku.com/), [Firebase](http://firebase.google.com/) and others, act as Infrastructure as a Service (IaaS) or Backend as a Service (BaaS) platforms. These services are cloud computing technologies which offer developers the convenience to deploy and host their applications on dedicated resources for a **cost per time spent price**.

This approach has simplified developers life greatly by the ability to scale up or down the resources (CPU, time, memory) needed on a regular basis. It’s a great exercise to go through deploying on any of these platforms.

I’ll defer to the beginner resources for guides:

Each platform has it’s pros and cons. AWS or Digital Ocean tend to be the go to choices.

**5.2 Shipping Mobile Apps**

Mobile applications are unique in that the distribution is handled almost entirely by the parent company who own the operating systems. Both distribution paths provide you the opportunity to see analytics, installs and update your applications accordingly.

**5.2.1 iOS**

For iOS your distribution will go mostly through Apple’s Developer program. You’ll need to [sign up as an Apple Developer](https://developer.apple.com/programs/)which **costs about $99/year**. Be aware Apple’s platform is **much stricter** with the quality of apps released.

You have to go through a set of tests and ensure your app is in line with their standards . These high standards make Apple’s app store a great place to launch or test, and developers also claim higher revenues from this distribution channel.

**5.2.2 Android**

For Android based applications you will most likely go through Google’s Play Store Developer Console. You will need to [sign up as a Google Developer](https://developer.android.com/distribute/console/)which **costs about $25 one time fee**. Google’s platform is less strict with the apps released, though it is getting better.

However, the less strict guidelines increase the noise of the platform and low quality apps immensely. It’s still a great place to go and getting better every week.

**5.2.3 Paying Developer Costs**

Some individuals I meet often complain and opt not to create expenses on things like cloud storage, or signing up to a developer program to build their project. I would caution against this. Just like planning before hand made the project more “real”, so too does adding a expenses. Now there is stakes on the line. There’s no need to go crazy, but consider any of the costs (if you can afford them), investments in the greater product you are working towards: yourself.

**5.3 Post Shipping**

After successfully releasing your application, you can pat yourself on the back ! Wicked job, but chances are no one knows about your app. 🤦‍♂️

If you want to get some feedback or users to test it, ask around or submit to product websites and forums:

**Product Hunt 🐱**

[Product Hunt  
Discover your next favorite thing Product Hunt surfaces the best new products, every day. It's a place for…www.producthunt.co](https://www.producthunt.com/)

**Hacker News 💻**

[Hacker News  
Three Charged in Fatal Kansas 'Swatting' Attacknews.ycombinator.com](https://news.ycombinator.com/)

**Reddit Startups 👏🏼**

[r/startups  
r/startups: The place to discuss startup problems and solutions. Startups are companies that are designed to grow and…www.reddit.com](https://www.reddit.com/r/startups)

**Side Projectors**

[SideProjectors | Marketplace to buy and sell side projects.  
SideProjectors is a friendly marketplace to sell and buy interesting side projects from other people. If you are a…www.sideprojectors.com](https://www.sideprojectors.com/project/home)

**Maker Base**

<http://makerbase.co/>

**6. Final Thoughts 🤔**

Building a polished product that you show the world is impressive and shows a lot of initiative, potential and ability. Anyone who has released something can tell you the process goes much beyond the code, but most people don’t get past that stage. Consider each project a learning opportunity and have a goal in mind to learn a new piece of technology or concept with each endeavor you begin.

Once you’ve release, get feedback and continue working on the product or iterating different designs or ideas. Perhaps you want to add payments, a new set of features or a complete overhaul on the design. Getting people’s feedback will allow you to improve the product and take it from a “just you” to many people kind of idea.

Finally, don’t get too frustrated. My first app is still on my GitHub, and I literally cringe at sight of that code and **how terrible I was.** Everyone is growing, learning and improving, and if you ever want to track your progress here’s a helpful heuristic:

*If you can look back on your code from the past few months and ask, “What the heck was I thinking?” then you know that you’ve made progress*