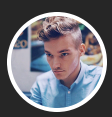
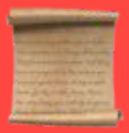
make

BOOTSTRAPPER'S HANDBOOK

building startups the indie way

by 

Product Hunt's 2x Maker of the Year + founder of Nomad List + Remote OK + Hoodmaps  Pieter Levels



Contents

TL;DR

Introduction

Because you have little time, here's the mega short TL;DR (too long; didn't read) summary of this book.

- ## 💡 Idea

  Get an **idea** from problems in your own life. If you don't have problems that are original enough, become a more original person. Don't build products that are solutions in search of a problem.

- ## 🛠️ Build

  Build your idea with the tools you already know. Don't spend a year learning some language you'll never use. Don't outsource building to other people, that's a competitive disadvantage. Build only the core functionality. The rest comes later.

- ## 🚀 Launch

  Launch early and multiple times. Launch to famous startups websites (like Product Hunt, Hacker News, The Next Web), mainstream websites (like Reddit) and mainstream press (like Forbes). But also remember to find where your specific audience hangs out on the internet and launch there. Launch in a friendly way, that means "here's something I made that might be useful for you", instead of acting like you're some big giant new startup coming to change the world.

- ## 🌱 Grow

  Grow organically. A great product that people really need which is better than the rest will pull people in. You don't need ads for that. Don't hire

people if there's no revenue yet. Don't hire many people if there's revenue either. Stay lean and fast. Do things yourself.

- 💰 **Monetize**

  Monetize by asking users for money. Don't sell their data. Don't put ads everywhere. Don't dilute your product. Be honest that you need money to build the product they love and they'll be fine paying for it.

- 🤖 **Automate**

  Automate by writing programs that do stuff that you do repeatedly. Only automate if it's worth the time saved. For stuff that's too hard to automate or not worth it, hire contractors. Let them work as autonomously as possible. Where possible let robots manage them (for example by giving them alerts when things happen in your product).

- 🤝 **Ethics**

  Not a chapter but important: **be ethical**, and don't cut corners on ethics. You'll be rewarded by not doing dodgy stuff like spamming, manipulating your users into doing stuff, growth hacking your search rankings or faking your social media, or abusing your power to compete unfairly if you're successful. If you make a good product, you don't need any of this. If you make mistakes, own up to them and say sorry. Be nice as a person and especially as a company. Karma always pays back in the end. Just being ethical and nice is a competitive advantage these days because most companies (and people) are not!

- 📝 **Homework**

  Homework: Each chapter ends with homework exercises that you can do. Instead of just reading, I'd like you to use this book as a handbook while actually building and shipping a product. It doesn't matter if it fails. But

you need to do something instead of just read! This is not startup porn! This is startup life.

Foreword

Thank you

The last years were a whirlwind of adventures while building all these products and being part of the startup ecosystem. I've met hundreds, maybe thousands of people while doing this. Writing this book is my final piece (for now 😛). And it wasn't possible without the help of a lot of people.

I'd like to thank every person who ever used my apps and websites, especially those who supported me financially by becoming a paying customer.

I'd like to thank all my followers on Twitter for supporting my work by giving me feedback, sharing it, and sticking with me through my ups and downs throughout this startup journey. Success sometimes makes you a dick (especially on Twitter!), and maybe I wasn't nice sometimes. So please forgive me and again, thank you.

I'd like to thank my family: my mom (Moeps), dad (Aap), **Jeroen Pixel** and Marijn for sticking with me and giving me true advice in times where it was impossible to get objective feedback anywhere else.

I'd like to thank **Youjin Do** for repeatedly asking me "when are you going to finish that f*cking book?" for 2 years, but mostly believing in me when so many did not.

I'd like to thank my co-shipping friends **Marc from BetaList**, **Lowen**, **Andrey**, **Oskar**, **John from Ghost**, **AJ from Carrd**, **Courtland from Indiehackers**, **Daniel my Server Guy™**, **Xiufen Silver**, **Yury the Critic™**, **Jelmer (aka Dutch Levels)**, **Amrith (aka the Shinbag)**, **Vlasti**, **Suska**, and **Felix from Fastlane**.

I'd like to thank everyone in **Work in Progress** chat for keeping me productive and forcing me to finally finish this book (which took me too long).

I'd like to thank UDL squad for their radical honesty, intellectual prowess and scholarly approach to reviewing my writing. And keeping me grounded as a human by being consistently unimpressed and laughing at my internet celebness.

I'd like to thank **Product Hunt**, and specifically **Ryan Hoover** and **Andreas Klinger** for always supporting me, giving great feedback and highlighting my startups repeatedly. And in a bigger way for creating this whole indie startup wave that I am benefiting from so much. It changed my life.

Why did I write a book?

I never wanted to write a book. I have to be honest to say I hardly read books myself. I think it takes a certain amount of hubris to put your thoughts in 200 pages and think you actually know something well enough that you should share it with people. But then all of you started buying it, so hey, let's do this!

I think it's stupid to read lots of books about doing something (in this case startups) and then believe you're actually learning something from it. Because most successful people I know learned mostly everything they know from practice. Just by doing things.

Books are also out-dated by definition. The moment I write this sentence and you read it (weeks, months or years later), I might have already changed my mind.

Then there's the entire survivorship bias, it assumes that what worked for me will work for you. But it probably won't. Because time has already changed and I will never be able to put on to paper all the variables that have attributed to things than went successful for me. I really don't like to give people false promises. Which is what most other books do. No, this book won't make you successful. That's all up to you.

But I do want to write a book

Even with all these things stacked against writing a book.

I want to write a book, if only because I see so much bullshit going around in the world of startups and tech. The media is presenting startups in the wrong way. People think they need to build billion dollar companies. They need to fly to San Francisco and build a "network" and get $10 million dollar investment from old rich guys. They need to hire 10x power developers and work them for 100-hour work weeks while feeding them pizza and soda. It will be great, they said.

But it won't. It'll probably suck. And you probably won't get rich. Because the odds of a venture capital (VC) funded startup are *by definition* stacked against you. Only 10% or less exit and that doesn't even tell you if the founders make good money. There's giant company exits where the founders barely made money. That's why I'm writing this. To show you, you might be able to do it differently.

The indie way of building a startup

What's the alternative? How about this: do things yourself and build a nice side project, that then can maybe turn into a bigger project, that then maybe becomes a company that makes you enough money to quit your day job and stop working for the man. Enough money to build up good savings, that if you invest well, will give you a nice early retirement. Your own company that can give you a little more freedom in your daily life, so you can spend it with friends, your family, your pets or just doing the things you love. Which in the best case is actually building an app, project, startup or company you love to work at/on/for!

I want to make bootstrapping great (again) #MBGA

This way of building a company is called "bootstrapped". Which means you're self-funded. You use the resources you have to get started. The odds of building a successful bootstrapped business are way higher than building a venture funded billion dollar company. Because the goal of a bootstrapped business is much more reachable. You don't need to do a billion dollars in revenue. You're already there if you can pay yourself enough to live from. Any money that comes extra is even better! You'll probably have less stress, be happier and be therefore a better friend, lover, partner or parent. Just....relax.

The coolest thing about bootstrapping it is that it doesn't exclude "going big" later. Venture capital investors LOVE to invest in companies that already have proven revenue. And that's literally what a bootstrapped business is. You'll be miles further than the person next to you pitching with just a PowerPoint deck. When you go for millions of dollars of VC investment on day one, it means you do exclude building a healthy simple business. Your company is now strapped to a rocket and you need to go big or explode. That's why I think bootstrapping is the better way to build a business now.

I really, truly, honestly want to see the mainstream startup narrative change into one where bootstrapping, revenue and actual profit is "cool". Writing a book on it with a proven framework people can apply, may help accelerate this change.

There's a personal legacy aspect here: if I can have a small influence in changing this, it feels good as a person. It's nice to change things for the better. And if it doesn't, well, thousands of people paid me money for this book, so it's a nice backup for me in case I go bankrupt.

Why should you listen to me?

Because I went from really scrappy side project to profitable company with users a few times now. Most times it failed miserably, but a few times it worked out for me.

At time of writing, my website **Remote OK** just became the most visited remote jobs board in the world with 1 million monthly visits. **Nomad List** is near that amount too, and ushered in a new era of digital nomads and remote work from **2014 onward**. They're both manually built by me, profitable with high margins (up to 90%), and highly automated. I was **Product Hunt's Maker of the Year twice**. I've launched my startups to Reddit's **frontpage twice**. I grew my projects together into **$50,000 monthly revenue** while **blogging** and **tweeting** about all the personal ups and (lots of) downs for the past years. Most of my other projects failed (some miserably), but I was able to get an idea to success a few times.

There is a good chance that there is strong **survivorship bias** at work here though. Remember that.

This book is my entire brain dump

I've been getting thousands of questions the last few years. I think if I started answering them I'd simply not get to working on my own projects anymore. That's why this book is the easiest knowledge transfer from me to you.

Literally every single thing I learned in the last few years building bootstrapped startups is in this book. It's my entire brain dumped on paper. It can be messy but it's everything I know. I hope it'll be something like giving back to the community and people will use it as guide in becoming indie makers and ship products. I've seen the drafts of this book already applied in hundreds of launched startups (because people will usually send me a message), which is super awesome. I'd love to see more. Having some positive influence on people's lives is a lot more interesting to me than more revenue, at this point.

This book is continuously updated

I'll be working on this book just like any of my startups. It's a continous project. I'll keep updating when I learn new things.

App, site, product, startup, business

You'll see these terms in the book used somewhat like synonyms. Because most of the theory in this book applies to all of these. Sites these days are like web apps, and apps are more like sites, together they are products and a few of these products make up a startup which in turn is a business. Generally, they're all the same thing.

You'll need persistence, and luck

You may need to try shipping 10 to 30 products for 1 to 3 years before you have anything that works. That's how this approach works. You build stuff and see what sticks. I don't know anybody who shipped one product and instantly became successful. It takes a long time to "get" it and even then it's a lot of luck and timing. If something doesn't seem to take off early on, it probably won't take off later, so make something new and try again.

As I, and this book practice radical honesty, there's a chance nothing you make will be successful. But by doing you'll have figured something new out, that might lead you to somewhere else, that will make you successful. Startups, and life, are about constantly pivoting when things don't work out. If you don't take action though, you can be sure nothing will ever happen. Stagnancy kills. So ship.

🚢 Always keep shipping.

Practice

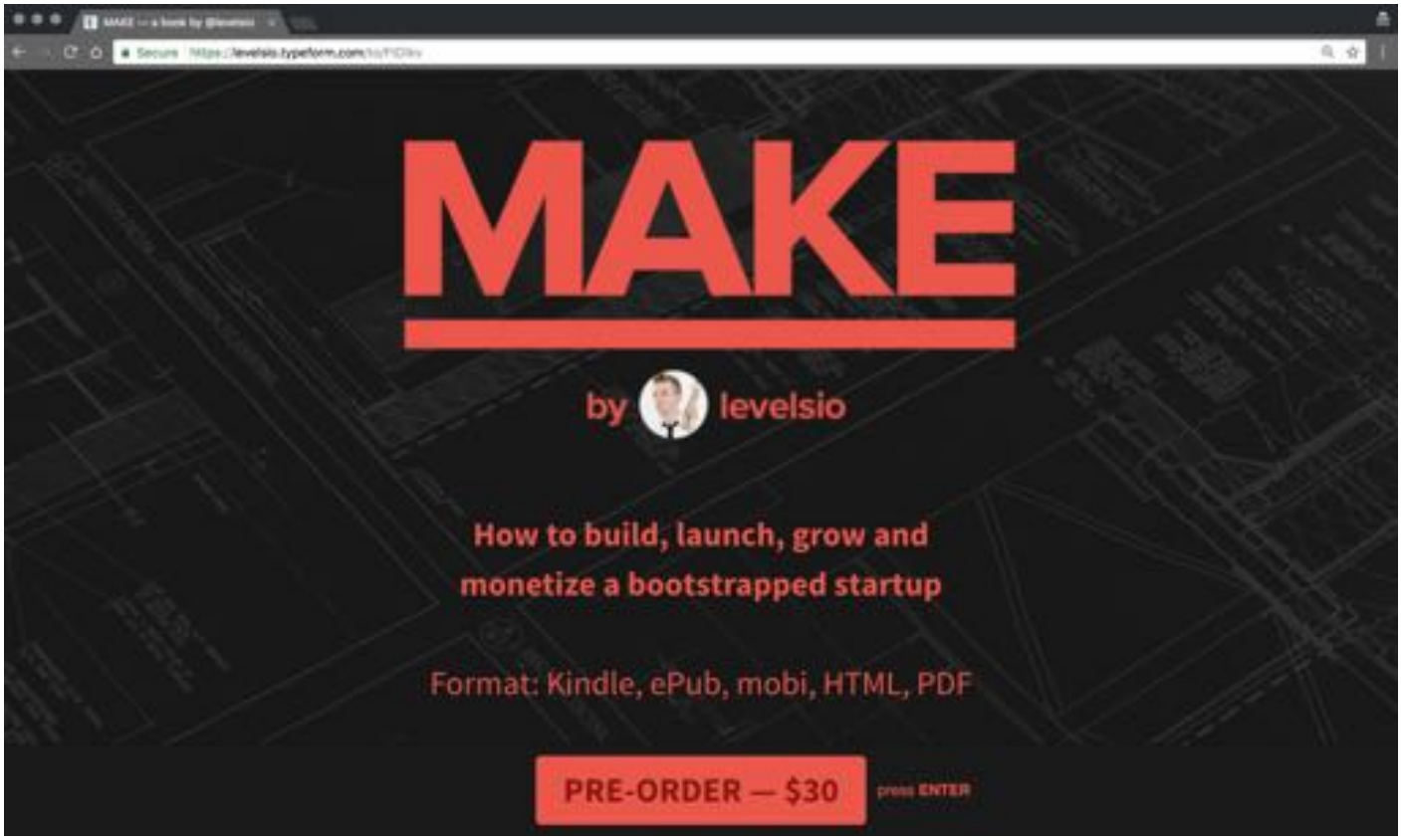
I want you to learn from actually shipping a product. This book is just ideas that might be wrong or right, and biased, but your own personal practical experience will be the thing (if anything) making you successful. Not this book! This book is just me pushing you to go sit on the bicycle. Now learn to ride it yourself. Practice is everything. Get your own style. And most importantly, ship.

This book

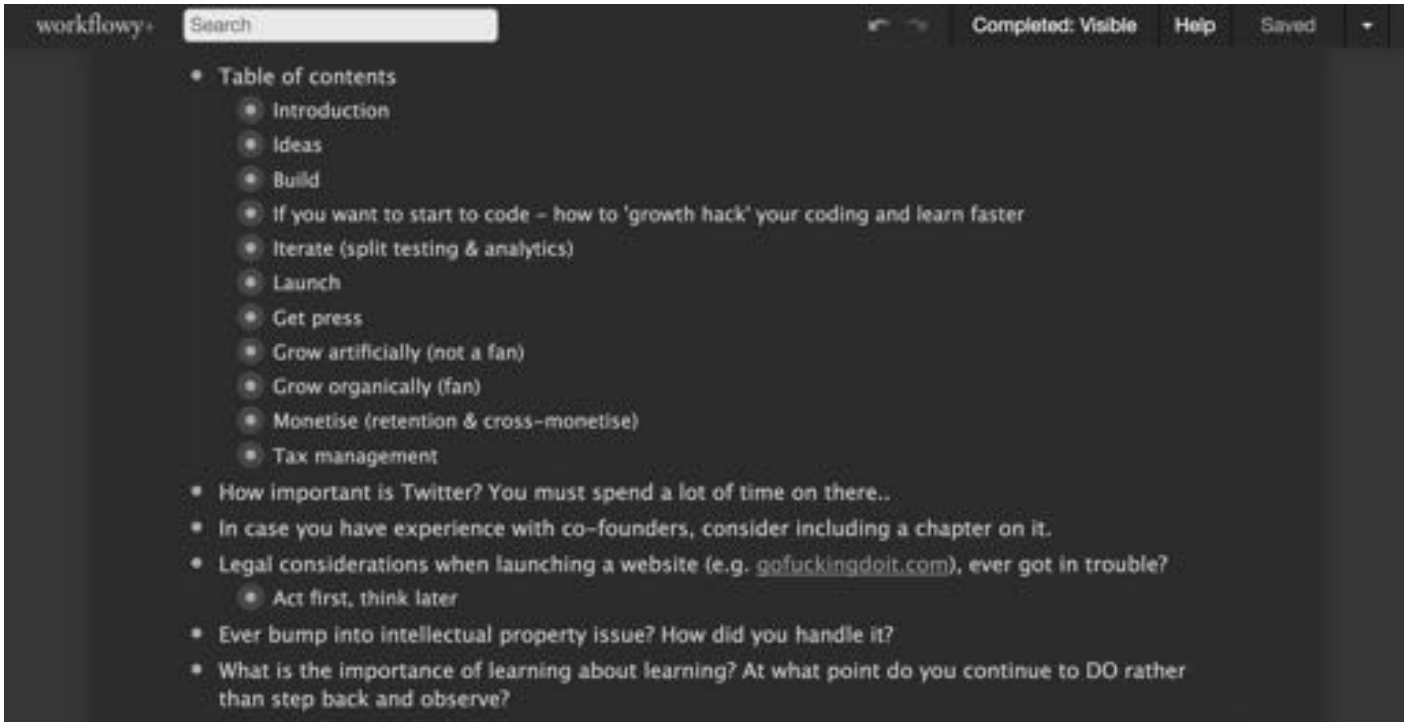
To avoid all those books with theories that are unproven, I felt on a very meta level, I wanted to write this book with the theory described in this book. To prove that if I could produce and sell this book in the ways described in this book, it'd somewhat prove the theories might work. So that's what I did.

Before even writing a single line on it I announced this book and opened it up for pre-orders:

The landing page was literally a **Typeform** telling that I wanted to write a book, but it didn't exist yet, and asking for $30 to support it.

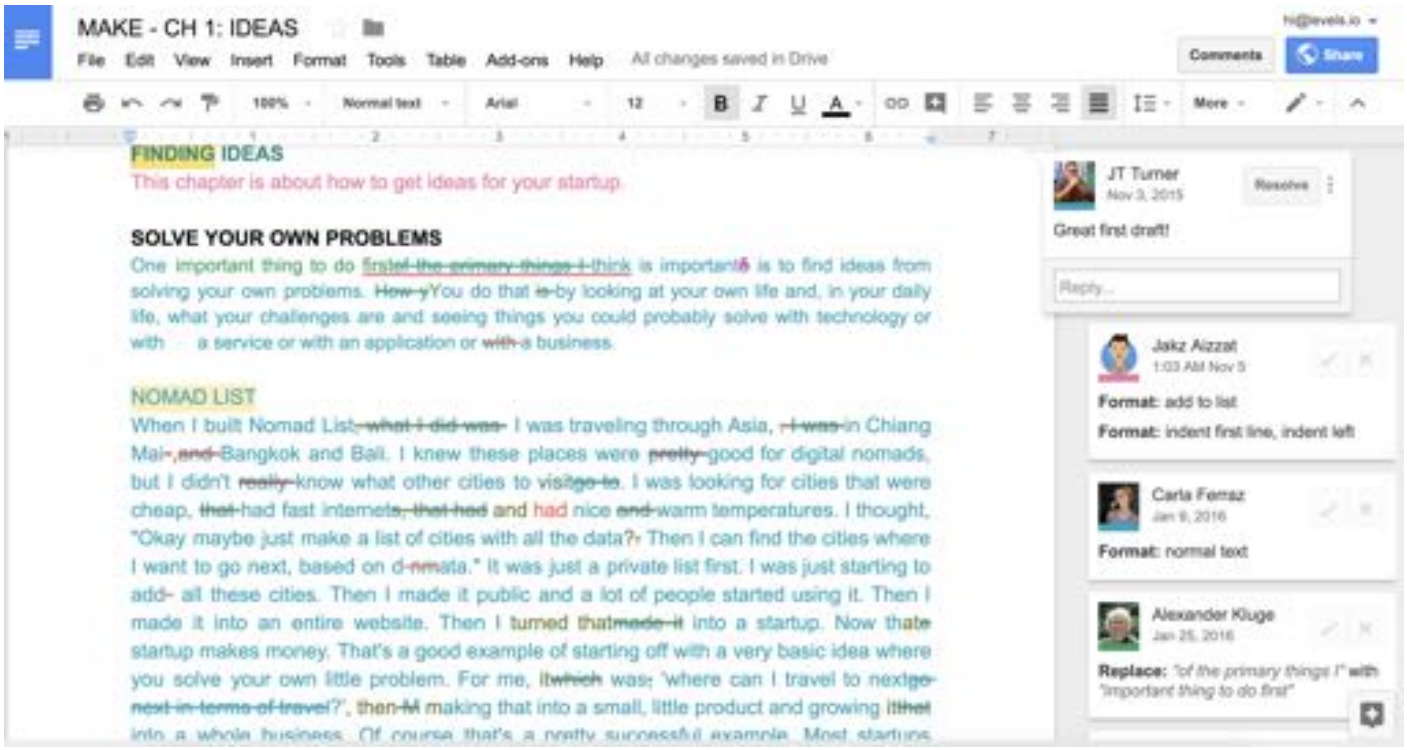


The only thing people received after paying $30 immediately was an empty **Workflowy** list where they could write what the book should be about specifically. That gave me immediate feedback from customers what they wanted me to make. Just like a startup.

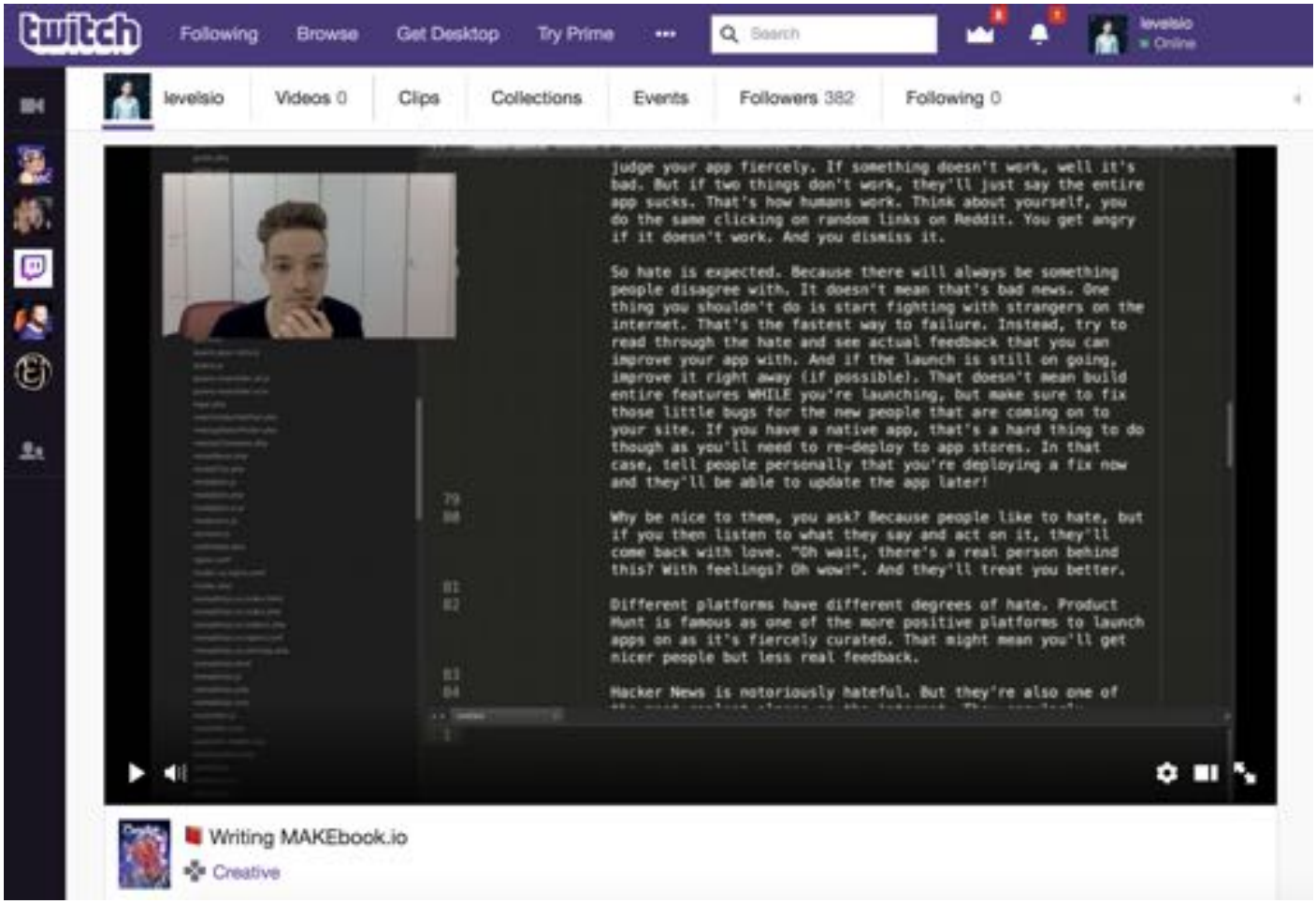


Thousands of people pre-ordered the book (quickly netting $50,000+ in revenue at $30 per pre-order) and there was thousands of items in the Workflowy list to write about. I went through it regularly and tried to re-order it to find patterns. I found I could divide up all questions people had in the different stages of startups. Ideas, building, launching, growing and monetizing. Everyone was at a different stage and needed different questions answered.

I then started writing the first chapter. I wanted to do this by "building in public" too. Or in this case, writing in public. I made a Google Docs text document and started writing.



But not before I started a live stream on Twitch of the writing process of course. I wanted to live stream most of the chapters. It helped me because, as you may know if you've ever written a book or thesis, the procrastination of writing can become incredible.



Every time I finished the draft of a chapter I sent it out to all the pre-order customers. And they could review it as a Google Doc again.

Here's the draft of the fourth chapter of the book! I know, a few days ago I wrote the third. But I'm on a creative super boost and I feel lots of stuff is finally getting out of my brain and pushed on to paper.

The previous chapter was about launching your ideas. Now I've written down everything I know about monetization. From how to find a model to monetize your app with, to when to start monetizing (e.g. before building your app or after?). Why should you monetize? And how do you monetize technically, which payment platforms should you use? How do you deal with refunds and chargebacks? And there's a secret part about how lots of people actually use venture capital as a way to monetize (without necessarily making any revenue).

This chapter is even longer than the last one, 24 pages and about 6.000 words.

It's a draft, and very rough (again!). On purpose. So I really welcome your feedback on it. I'm writing this WITH you, so your feedback is everything. Here's the Google Docs version, please add comments and everything you'd like to add (it's publicly commentable).

Chapter 4: Monetization
https://docs.google.com/document/d/ ic/edit

Chapter 3: Launch
https://docs.google.com/document/d/1 k/edit

Chapter 2: Build:
https://docs.google.com/document/d/1 60s/edit

Chapter 1: Ideas:
https://docs.google.com/document/d/ J/edit

Click View -> Mode -> Viewing to see the chapter without everyone else's edits!

Thanks!

Pieter Levels

Only the final process of writing this book (where I'm at now), was done by myself. Collecting all the content, cleaning it up, rewriting it and making edi-

torial decisions on what should be in it. But 90% of the process was out in the open.

And that's exactly how I have built and would build startups: launch early and build with/for your users.

Idea

Introduction

This first chapter is about how to get ideas for a startup.

Solve your own problems

The most important thing is to find ideas from solving your own problems. You do that by looking at your own life and observing what your daily challenges are. Then you see if you could make those challenges easier using technology. If you solve your own problem, it's very realistic that there's many more people like you who would also love their problem solved. And that's pretty much what a business is. Solving lots of people's problems in return for money.

In startups, this business can be in the form of an application, a website or even only just a physical service tied together with some technology.

Your problem might just be everyone else's

My most successful project is Nomad List. It's a platform for remote workers and travelers to find places to go to and meet other people when they're there.

When I built the first version of Nomad List, I was traveling through Asia. I was living in Chiang Mai, Bangkok, and Bali. I knew these places were good for digital nomads, but I didn't know what other cities to visit. I'd go to Singapore, and discover it was nice but also very expensive. I'd go to Vietnam and realize wow it's really cheap here, but the internet is unusably slow. I was looking for cities that were cheap to live, had fast internet, and warm temperatures. I thought "Okay, maybe just make a list of cities with that data?". It was just a private spreadsheet first. I added about 25 cities's temperature, internet speed and cost of living. Then I wanted to share it, I tweeted the URL.

But something was off. When I looked back, someone had hacked my spreadsheet. There were now about 75 cities and it wasn't just temperature, internet speed and cost of living. There was safety, best coworking space and climate added as columns. Instead of sharing it as "read-only", I had accidentally clicked "editable by everyone". Suddenly there were hundreds of people adding data about their cities, and the cities they'd been to. After a month, thousands of people had added data about over 250 cities.

This is a good example because I just wanted to solve my own problem, and it turned out it was thousands of other people's problem too. Your own problems are nice because often they're quite niche level, meaning you'll have enough other people with the same problem to be your user, but not enough that some giant company has already done it.

You are the greatest expert at your own problems

Even if it's not always successful, the concept of solving your own problems is a great way to find ideas that might be viable. A lot of people don't do this. When you try to solve problems that aren't even yours, like somebody else's, you can do that but you are not an expert in the problem area.

For example, I could make a health care app which registers patients and their health situation, which diseases and ailments they have, and where they are in the hospital system. But I'm not a doctor. I've only been to a hospital when I was sick. I have zero expertise on health care. I don't know the problems doctors have.

I can look at the entire industry from an outside perspective and think: "Okay, we need to solve this problem. There's a lot of money and opportunity here". But I still don't know anything about it. I'm just not an expert in the problem space. And until I become a doctor, I probably will never be.

You want to find ideas from your own problems because you'll be an expert on them immediately.

How well do you need to know a problem to solve it?

Somebody asked me, "How well do I need to know a problem to make something for it? How much can I learn along the way?".

I think you need to know a lot about the problem you're trying to solve. Once you have traction and your website, app or startup works, you can learn from user feedback and data you collect then. But that's useful once you're already running. It won't help you find ideas that are suitable for you to execute. First-hand experience with the problem you're solving is best.

And not just at the start. When your company grows, and you stop becoming the target audience, you face a big risk of not understanding the customer anymore. A good example is in hip-hop, a rapper comes from a low-income ghetto, gets success rapping about that world, then gets rich from the success, loses the inspiration of that world since he/she now lives in a giant mansion and their career is over.

Personally, I saw this happen with myself. For awhile, I took a break from traveling as a digital nomad and the product suffered. Now that I'm back on the road, I see where my site sucks. For example, it doesn't work well when I visited smaller towns where there's hardly any nomad activity. So I lowered the membership price of **Nomad List** and started focusing on just getting more people in. As that's more important short-term than making money. I wouldn't have realized this by staying home.

Problems are always changing, as are markets, as are people. If you are your target market, that's perfect. You just evolve with it. The founder of Boosted Boards, an electric skateboard, said:

"We just wanted to make skateboards for ourselves and there was no real good electric ones. We built it for ourselves. We knew exactly how we wanted it to

feel when driving. We knew what people wanted because we were the target market".

I'm starting to repeat myself, but that's because this is a repetitive theme with successful companies. And opposite: a repetitive theme with failing companies is founders who have no connection to, expertise on or passion for their industry.

Be more original, and your ideas will be original

There's an issue in itself with only solving your own problems. What if you're not as unique as you think? It takes only a slight glimpse at current startups ideas to see that everyone is making the same stuff.

Everybody's doing a to-do list app. Everybody's doing productivity apps. For a decade, people have been making apps to find your friends on a map. Everybody's doing some kind of photo sharing app. These are basic ideas that everyone has. And they're too obvious.

Everybody's doing them because everybody has the same problems. So how can you find problems that are actually unique and original? Well, become more unique and original yourself. I would have never built **Nomad List** if I didn't go traveling and working.

So, you need to do stuff that makes you explicitly very different. It will get you more unique ideas. That's super cool, because now you have two great attributes of an idea. It's not just unique, but it's also something you're an expert at since you've done it yourself. Even if you launch and get competitors later (like I did) because they see you're making money and it's a good market, you'll still be in a better position than them because you're real. You've done it. You're an expert in the problem you're solving.

I would be bad at making an app for doctors, but if I was a psychiatrist, I'd probably be able make a very good app for psychiatrists, since I'd know all about it. Part of the Lean Startup approach dictates "talk to customers to find

problems". That's nice and all, but then you're still working from an outside perspective. I'd say, just focus on your own problems.

Also, stop reading books to develop yourself or get ideas. You won't get them from there. Or if you do, there's lots of other people reading the same book probably. Get ideas from your life experience. Get outside. Become original. Do crazy stuff that you're scared off. Jump off cliffs (do it safely). Ask people you like out. Walk into random office buildings. Jump fences. Crash hotel pools. Whatever makes you different. Don't be so scared! Live.

The downside of solving only your own problems

There's a strong and somewhat valid criticism on the philosophy of solving only your own problems.

If you're a wealthy guy from a Western country, you're probably going to solve problems that make your already pretty good life better. You're not going to solve problems that women or people with other genders have, because you don't experience their problems. You're also probably not going to a developing country and solve the problematic garbage situation there. It's the privilege argument.

I get it. And I see the criticsm there, but then, it's hard for anyone to solve a problem outside the context of their own subculture, city, country, continent, region, income class and gender, because they're not experts on it. And maybe that's not as bad as we think. If we can democratize access to computers and the internet (as we're doing now), people anywhere can focus on solving THEIR own problems and reap the financial (and other) rewards from doing that. And isn't that much better than having a white guy do that and reap the rewards?

Always start from the problem, not the solution

A lot of companies start not from the problem but from the solution. This is one of the biggest mistakes you can make. Technology needs to solve a problem. If you make a solution for a problem that doesn't exist, it might look sexy technologically, but it won't get users.

When new technology becomes available people want to use it to build something. A great example is the endless amount of apps that have appeared since smartphones got GPS chips a decade ago. The first thought is, okay, let's use this to track each other. So you make a map with friends on it and where they are. This has been tried over and over and it's still a pretty terrible idea. I don't have a strong curiosity to know where my friends are and I don't necessarily want them to know where I am (due to privacy). The problem doesn't exist. When I meet up with friends, we simply say a place we're meeting up and we can find each other.

Now a good example where this technology is used to solve a problem would be Tinder. It does use your GPS location, to find people around you to date. That works because you don't want to match with people on the other side of the world. It solves a problem and the solution is enabled by available technology. The problem should always be first, not the technology, not the solution.

To get big, you have to start small

Niches are specific market segments that are shallow enough to easily access, with not many players in there. Niches are a great start because they're usually too small in economic value for big companies to attend to. They're also the perfect size to market a new company towards.

Niches go against the ridiculous "go big or go home" attitude that the rise of startups in mainstream media pushed from 2006 onward. But that attitude isn't realistic. Because most big companies started very small. If you start big from day one, it's the wrong way to go about it. People don't like niches because they're too small for people's ego's.

But niches are much more profitable than you'd think. If you have "just" 1,000 people paying you $83.33/month, that's $1,000,000 in revenue in one year!

How to make $1,000,000

Make a	$5,000 product for	200 people
Make a	$2,000 product for	500 people
Make a	$1,000 product for	1,000 people
Make a	$200 product for	5,000 people
Make a	$100 product for	10,000 people
Make a	$10 product for	100,000 people

How to make $1,000,000 w/ subscriptions	
500 people pay you	$167/month for 1 year
1,000 people pay you	$83/month for 1 year
2,000 people pay you	$42/month for 1 year
5,000 people pay you	$17/month for 1 year
10,000 people pay you	$9/month for 1 year

Start with a micro niche

Let's say you want to make booking software for hairdressers. That could be a niche. But how many hairdressers are there? Probably millions worldwide. Why not go even more niche? Booking software for hairdressers that focus on African hair. Now you're talking tens of thousands maybe. That's a good start. Let's say you captured only 10% of those 10,000+ hairdressers that focus on African hair, that could be 1,000 customers paying that $83/month making you a million-dollar bootstrapped company!

From micro niche to multi-niche

If you're at a party and someone asks you what you do and you answer you make booking software for African hairdressers, you migh think it's too basic, i.e. you're not "changing the world" with this. But you shouldn't. Because first of all, you're probably making good money off those 10,000 hairdressers pay-

ing $100/y (that's $1M/y in revenue already!). And if you've validated your startup in that niche, you can expand to other niches. What about Asian hair? White people hair? Step by step you move closer to the entire booking for hairdressers market. Why do it step by step? Because you can easily go smaller again if you fail. And try to expand in another direction. It's like a lightning shock in slow motion, it tries to find the path of lowest resistance. Your company should be doing the same thing.

From multi-niche to vertical integration

Now let's say you have the entire booking for hairdressers market, what about booking for nail salons? And tattoo shops? Oh now, you're doing restaurants. Now what else do all the shops need? A point-of-service payment platform. Okay you have all these shops as customers already, so you can easily sell this new service to them too (and best of all test is on them first). This is vertical integration, you try to go into the businesses that your current customers also use.

From vertical integration to becoming a platform

With the experience from payments for physical shops, you move to virtual payments too. With lots of luck, you'll be one of the biggest payment gateways in the world. You'll go into ecommerce with virtual storefronts for the physical shops of your customers.

You just became big by starting small!

See what happened here? You became big by starting small. Most people do the opposite and crash and burn. They want to build that giant payment platform that will change everything. But hardly anybody started with that. Facebook was a Hot Or Not app by Marc Zuckerberg. Apple was a personal computer build kit for amateur hackers. Microsoft was a tiny software agency that re-

sold MS-DOS from another developer to IBM. Google was a small academic experiment to search Stanford University's local intranet. Get it?

Stop. Thinking. Big.

Think small first! You'll be big at the end!

Your idea does not have to be earth-shattering

I don't think your idea should be earth- shattering. If you look at most big startups, their first idea was not earth- shattering at all, most of them. Think Uber. They started as an app where you can simply call a taxi. Right? Then it grew into an entire transport solution. The long goal is self-driving cars, that transport everything and to replace the entire transport and delivery industry.

All that started with a taxi hailing app. Your first idea does not have to be (and probably should not be) earth-shattering.

You start with something small. Don't think too big. Then slowly, you can get to the big part by extrapolating, scaling your idea to a bigger market, from a niche market, and to a bigger more abstract idea.

With Nomad List, I started with a list of cities with cost of living and internet speeds. Then I scaled that that up to a social platform for digital nomads. Now the long-term goal is an entire internet platform for the future of remote work. That means more tools for nomads, remote workers and businesses that embrace this future. That all started with a database of cities, that's not earth-shattering at all.

The more I talk with people in the startup world and they tell me like, "This idea is going to change the world. It's going to be a billion dollar company." Those are usually the ones that fail really, really bad. The ones that go really well, and I know from seeing friends around me, are the ones that are people are really modest and they just say, "Ah we just want to fix a small thing. Then okay we fix it. Okay. What's next? What are we going to do next." They do have

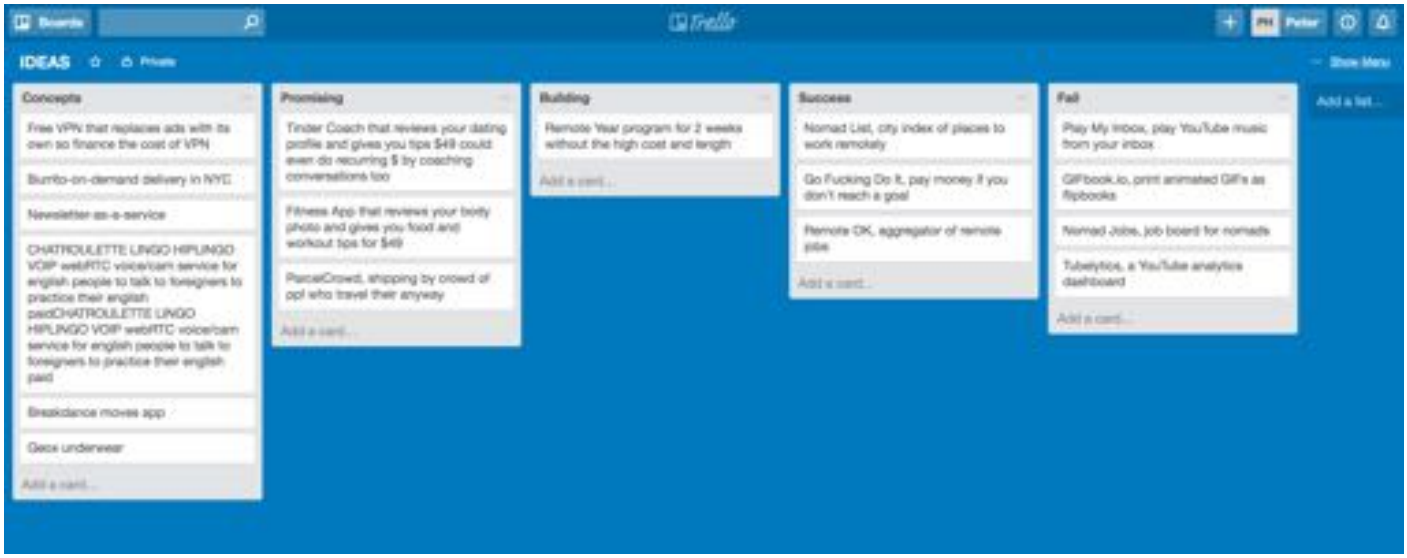
probably a long-term vision, but focus on the small stuff every step of the way because that's how you get big by focusing on small steps. If you can't even do the small steps right, how are you going to get to the big part? Right?

Don't focus on the idea that's earth-shattering. Just start with something basic. Virtual reality is an industry that will be earth-shattering, but you don't have to have an idea that's earth-shattering. You can start with a basic, little virtual reality game/app (like I'm doing now) and slowly add functionality until maybe it becomes like the next virtual social reality world where we're going to live our lives in.

You don't know what you're going to end up with. That's another point. You need constant feedback from your users in the markets to see what people want and what people use and whatnot. You can't just think of that. You can't think big immediately. You have to start small.

Create a list of ideas and keep track of them

A good start if you're looking for ideas is to keep track of any you might get. How you keep track of your ideas is up to you. I now use Workflowy and previously used Trello for it. I have a few different lists. The first is for "Concepts", that's rough ideas. Then if one of the ideas looks good, I move it to the "Promising" list. If I actually start executing them I put them in the "Building" list. Then if it's done I have a "Success" or "Failure" list. When it's a failure, I usually also write inside the card why it failed in a post-mortem that's one sentence long.



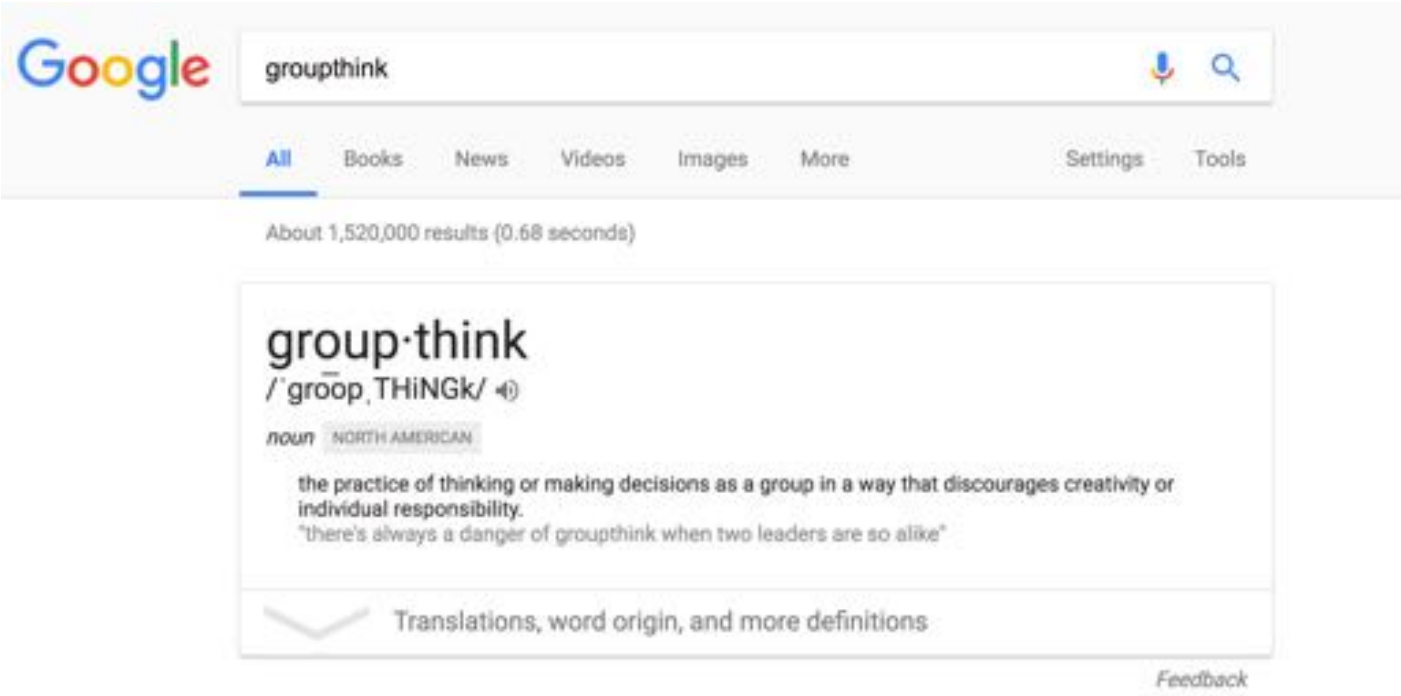
The first list of basic ideas has zero limits in how weird or crazy the ideas can be. This is on purpose. There shouldn't be any judgment on your first basic creative premise. It can evolve into something more practical later.

The thing with ideas is, at least with me, that they keep coming back in my brain. I'll get a basic hunch, then weeks later it comes back, and then months later it'll start to manifest in my mind. Then sometimes even 2 years later I'll finally start executing it. This is great because your mind is like a rice cooker for ideas. They need to get ready before you can put them out and build them.

This list of ideas doesn't have to be physical or virtual, it can also just be in your mind if that works for you. As long as you keep going back to ideas and see if they have evolved to become worthy of actual execution.

Should you make ideas alone or in a group?

I think collaborations can be very dangerous. Because if you work with somebody else in a team, there's a big tendency of group-think, where everyone starts hyping each other on the value of the idea. The prototype might only get mild validation from paying users, but you're working with this group and you're already so crazy about it that it doesn't really matter what users pay/do/say. That's very bad. It should only be about the users.



I've been in these rooms, I've seen it happen. "I"m telling you Joe, we've got something really good here". No, we don't care. It should be only about customers. You see a lot of startups go wrong because they have this group think in the beginning and it's actually not rational thinking. You're more rational on your own. Obviously, collaborations can work with idea generation too, but I think the most important part of idea generation is getting ideas yourself, then talking to people, customers, users to evolve them. Not talking to your teammates how great your team's idea is.

The worst is to be with people that just confirm what you already think. The best is to test your ideas as quickly as possible. Even asking other people for advice is kind of bullshit. You can't ask "will this idea work". You need to ask the market by building it! Nobody knows until you launch!

Don't be afraid to share your ideas

The most elementary mistake people still make is not sharing their ideas. No, people won't steal your idea if they like it. And even if they do, they probably can't execute it as well as you. And even if they do, you're not a snowflake! There's thousands or even millions of people with the same ideas as you. Stop

thinking you're so special! Ideas are a dime a dozen. Everything is about how you execute.

Idea	x	Execution	=	Business

Bad idea	=	0	No execution	=	$0
Good idea	=	10	Good execution	=	$100,000
Great idea	=	20	Great execution	=	$1,000,0000

> "To me, ideas are worth nothing unless executed. They are just a multiplier. Execution is worth millions. To make a business, you need to multiply the two. The most brilliant idea, with no execution, is worth $20. The most brilliant idea takes great execution to be worth $20,000,000."

— **Derek Sivers** in his famous essay **"Ideas are just a multiplier of execution"** (2005)

You shouldn't be scared of sharing your idea because execution gives the idea its details and specifics. That means 10 people with the same idea will execute it in 10 completely different ways.

The benefit of being able to share your ideas is that you'll be discussing them with everyone. Potential customers, vendors, suppliers, whoever. Everyone will have some input on it which you may or may not use as feedback. Again, the market remains the most important feedback though.

Not sharing your idea is stupid because it'll stay only in your head. You for sure won't be objective at judging it since you have something called "optimism bias" which is "the tendency of individuals to underestimate the likelihood they will experience adverse events", e.g. you think it'll definitely be very successful.

It doesn't matter if people say "that idea will never work" because they're not the validation. The user paying/using it is the validation, not other people judging your idea! The point of sharing your idea is thus not to get people loving it or hating it. The point is that you get your brain working outside of its comfort zone (of talking to itself) and you'll evolve your idea. You'll come up with adaptations of your idea, or entirely new ideas by talking about it.

Conclusion

To get ideas, try to find problems in your daily life. You're the foremost expert at problems you have, more than anyone else who doesn't have them. If you keep coming up with the same ideas as everyone else: try to make yourself a more original person by actively experiencing different things. Don't shy away from taboos and fringe ideas, that just mean you're ahead of the curve, they might become the next big thing. Don't think big, start thinking small first, then take it one step at a time, you'll become big by starting small. To avoid group-think and drama: work alone, especially early on. Share your ideas freely to get other people's input on them. Log every idea you have, filter them, and see which ones you can execute upon.

Resources mentioned

- Workflowy
- Trello
- Stripe
- Zapier
- Intercom
- Olark
- Typeform

Your homework

- Spend the next 7 days making a list of problems you have in your daily life, they can be small or big, try to find 3 ideas per day, so that you have at least 21 at the end of this week.

Build

Introduction

You made it to the second chapter. We'll now discuss how to build your idea.

In the previous chapter we looked at how to find ideas for your product. To avoid getting stuck in an infinite loop of brainstorm and bullshit, you want to start building as soon as possible. The faster you get it out, the faster you'll see if people want to use it, how they use it and what other features they want to see you build. Without shipping, it's difficult to get any idea of what they want. There are obvious exceptions here: Apple hardly uses any focus groups or user testing, they choose to ship the very best (according to them). That's fine, but you're not Apple (yet).

Build fast and minimal

While a decade ago development time was long and took lots of planning to get right, today we can go from idea to basic product in a matter of days.

We have cheap and easy to set up servers (virtual private servers or VPS, like **Linode** and **Digital Ocean**), and platforms as a service (PaaS, like **Heroku**) that take out all the difficulty of setting up your own server. Server operating systems like **Ubuntu** have become relatively simple to set up with tutorials readily being available online. The web server software itself has become fast and simple too: **NGINX** allows us to set up a basic default server that doesn't need much further configuration. Languages like Ruby, JavaScript, PHP and their frameworks like Ruby on Rails, Meteor, Laravel make it easy to build products by skipping the ground work.

> It's fine if you don't recognize a lot of the terms in the previous paragraph. This isn't a technical book. And the tools with which you make your product don't really matter in the beginning (and not even that much later either).

My point is, the speed of the development can be very fast now. And the zeitgeist of our time is transient too. Evertything moves fast now, and where in the past people would use an app for days or weeks, now you literally have seconds to make an impact, or they close/uninstall it. So you have to move fast to stay ahead. Another attribute of our zeitgeist is minimalism. Users finally accept minimal interfaces and minimal functionality, as long as an app does what it says well. There are single-purpose apps now that just do one small thing very well. Many of them have been quite successful.

Downsides of fast development

There are downsides to this culture of fast development. There are products these days which are still littered with bugs when shipped. They might not work well on all mobile devices. They're not tested. There's products that are launched and work great in the first week, but then their developers stop maintaining them and they break after a few months. The remaining users get annoyed when the app stops working. With development being fast, rough and dirty, so are the results.

The other downside is you don't have the time for sophistication and details. Code has a higher potential to become an epic mess of spaghetti because there's less architectural planning involved. But then again, you can always rewrite code right? At least you've shipped. Users don't care how your code looks.

Your enemy is perfection

The reasons you should launch fast are to avoid inaction due to perfection. You should start avoiding striving for perfection now and maybe later too. This applies to all phases of doing a startup (and maybe even life), but especially the start. Perfectionism is detrimental because 9 out of 10 times it doesn't make things better but just creates inaction. You'll have 50 meetings, 100 iterations of an idea or feature, just to get it right, when actually you should have just gotten it out of the door immediately and let people use it (and learned from them using it!). Perfectionism is necessary in smaller details of your service or startup, but not in the entire thing because then it will simply paralyze you with inaction and fear that what you do is not perfect. Nothing is perfect at the start. Things become perfect through lengthy iterations!

If you're in the early stages of a company, you want to get out stuff as fast as possible. Actually, overall I think in the entire stage, it will be useful to do that because users love new features. They love to test new stuff. They're actually pretty okay getting into bugs. They know development is difficult these days. I mean Gmail was in Beta for a decade. People are fine with errors, as long as they can use something new and as long as the errors get fixed, and if they have some way to tell you the feedback about the errors, for example. I think getting stuff out the door is the number one priority.

Avoiding perfectionism is a skill you have to develop. You need to learn to be fine with everything being not fine. Not everything will be perfect and high quality. Maybe your website will have the wrong images, the wrong background colors and the logo color doesn't match the branding text in the rest of the page. Is that important? Really? It's okay for a day. It needs to be fixed at some point, but it's not a priority. The product works. Prioritize perfectionism. Perfect what needs to be perfected now. The stuff that's low priority, don't put too much effort in it to perfect it.

You'll always have time to perfect stuff later. You can always iterate and develop it to be great later. Why not perfect it step by step? Make small improvements.

There's a limit though to this though. Don't make shitty products. With minimum viable products there's been a rise of people shipping products that look bad and hardly work. A first prototype should function really well. It's fine if there's some bugs on the side, but the core functionality should be operational. It should look at least OK, otherwise you simply turn people away. Minimum needs to be minimum good.

How viable should a minimum viable product (MVP) be?

The bad MVP's we're seeing recently are creating a counter movement against the lean product development trend. And that's understandable. A lean or minimum viable product doesn't mean it can be shit! It has to actually function, users have to be able to use it. Bugs are fine, but users should be able to report them instantly (like with a feedback chat box pop up). You'll also have to fix bugs fast or people will get annoyed (and you'll just grow more and more bugs). Making a minimum viable product means it has to be viable, it's not an excuse to be lazy and make something half-assed. So it's all about balance. Make something great that functions and it can be minimal. But make sure it works!

Build yourself or outsource

Many people ask me if they should build their product themselves, outsource it to other people or hire full-time employees to build it for them.

I have a pretty extreme opinion about this that goes against the grain of what most say. But I think I'm right because times are changing.

DIY vs. people who hire others to do it for them

I think you can get quite far by letting other people develop your product for you, but the problem is that in the long term you won't be able to have the same speed as a product maker with development skills.

To give you an example: my day usually starts with waking up, showering, having some coffee, seeing what bug on one of my sites is emailed/tweeted/shared with me today (someone sends me a bug report almost every day), and I'll see if I can fix it immediately. Then I'll see other stuff on my site that I don't like, and I'll just change it on the fly. All of this usually takes a few minutes. Those are just small tweaks every day, but over a year that's like 300 to 1,000 small tweaks which add up.

Now imagine if you outsource all of this, and you have to ask 1,000 times for a developer to tweak this and that. For me, a small tweak or bug fix will take just a few minutes because I can do it myself. But for you, each tweak takes a message to your developer, who then has to be working that day (and awake!) to get it fixed. He might have to do a meeting about it with his team. So it might take a few hours, or days, or weeks.

Now imagine if we have the same site and you and me are competing. In the time you've sent a bunch of emails to your developers, I've already pushed 5 bug fixes and added 2 new features to my app. Who's going to win here? Who's faster? Not you!

Now think about your users. They're now stuck with a bug for 3 days because your developer is hiking up a mountain in Peru for holiday. I fixed that bug while having my coffee in the morning in 5 minutes. What will your users like more? Your broken app or my working one?

It's not all roses though. Doing everything yourself in the long-term probably doesn't work. If you've proven a business model, it's probably smart to keep repeating it (by scaling it up), getting people to work FOR you, with you managing the operations instead of micromanaging and fixing tiny bugs. But that's not what this book is about. We're talking about the beginning. In that case, DIY always tops outsourcing for me.

DIY vs. big teams with VC money

I've seen countless real life examples of VC-funded companies spending loads on building giant development and design teams, or paying the same money to outsourced development and design agencies. There's lots of exceptions were it went right, but mostly in the beginning this just slows you down terribly.

There's been about 5 big direct **Nomad List** competitors come and go now that were VC-funded from 1 to 10 million dollars in funding with teams from 10 to 30 people that made the same site as me. But they all didn't go anywhere.

While working alone in my underwear on the side of my hotel bed with my MacBook and my coffee, I was able to outcompete million-dollar VC-funded teams of 30+ people in an office in San Francisco with Aeron chairs, oakwood meeting desks, $20,000 espresso machines, bean bangs and ping pong tables.

That's a really cool thing about the time we live in. It's a pretty fair race. You just need to make better shit than other people and it gets rewarded.

When starting up, you don't need a team if you have the skills yourself. You don't need startup capital if you have the skills yourself. And getting the skills yourself isn't that hard if you can search every question on Google. And in my opinion, having those skills sets you apart.

You don't even need to be great at it. I'm not great at programming or design. I'm pretty average. But because I can do it all well enough, I have an advantage over many people, teams and companies who are specialized.

My gripe with venture capital in terms of building products

My social media outrage might make you think differently, but I don't hate venture capital. It has its function and place. But I don't like to see (other people's) money wasted on bullshit. And there's plenty of VC money funding bullshit. The costs of building a startup (especially, especially in the beginning) are almost $0 now. That is, if you use your own current skills.

There's a problem with the current narrative for how you're supposed to build a startup:

- Get VC money from day one
- Hire too many developers, designers
- Rent an expensive office
- Have your team build something
- Buy an espresso machine
- Do team-building retreats
- Buy startup goodies like t-shirts and hats with your logo
- Get drunk in a jacuzzi to celebrate raising more money
- Oops, the product didn't get traction
- Sorry VC, we shut it down
- Bye money!

The more sustainable way to build a startup:

- Build something yourself
- See if it works

- No, build something else
- See if it works
- No, build another thng
- See if it works
- It works!
- Let's see if I can monetize it
- I can hire some people now with the money I'm making with it
- Now I have a team of a few people
- If we want, we can rent an office, or just save money and stay remote
- The business model seems to be proven because every time I spend $1 more, I get $1.50 in revenue, thus it's scalable.
- This means, if I get more money, I can spend more and get more profit theoretically
- Maybe we can borrow or get some angel investor or VC to fund our expansion in return for giving away some of our ownership, or use our cashflow for it
- We got more money now, we spend it on the right things, and now we're making even more profit, it worked!
- Now the product is really cool, people love it, and we're making lots of money, mission accomplished

- Let's see if I can sell the product because I want to do other stuff with my life (build a family, start a farm, raise kids)
- Okay, I sold it for $500,000 to $10,000,000, now I have financial independence in return for a few years of hard work and my investors are happy too!

This is relevant because venture capital money can destroy the build process before it's even begun. If approached wrong, raising money means you can skip the actual product validation (if people want the product) and finding a business model (because you don't need money to survive, you have funding). It lets people spend lots of money without having to see any (positive) results for it for a long time. That causes people and companies to be jaded. It's not natural for humans. The natural tension of having to survive is healthy and it makes people act in superhuman ways.

There's always the story of the son or daughter of rich parents who could never find a job until their parents cut all spending on them and suddenly they found a job (because they HAD to get a job to survive). I think the same applies to companies. If you don't need to find a business model (to survive), you're not really going to prioritize looking for it either.

Bootstrapping vs. venture capital

Bootstrapping has become an advantage. You keep your costs low, naturally. You get zero of the bullshit attached with VC money. You maintain full ownership over your product and its roadmap (where you want to go with it). You maintain full ownership over the equity so that when you sell it later, you'll get lots of money (instead of just a diluted 5% because VCs took the rest).

And in our times you don't really need a lot of money to build something yourself. A simple web server is $5/month. A code editor like Atom is free. To make iOS apps, XCode is free. An Apple developer license is $100/year. Not a lot compared to the money you can make back from it if you can get people to use your product and pay you hard dollars for it.

Bootstrapping has become a very viable option for most software-based start-up ideas. Whereas, venture capital has become tedious to get (schedule 6 months of meetings), limits your freedom (grow crazy big or die) and not really necessary at all. Did I mention how much less stressful bootstrapping is, yet? No, well it is!

The importance of keeping costs down

In the early stages of any project that's not funded, keeping your costs down is your biggest priority. And if you use your own skills to make the basics, that's a lot of costs saved. Developers these days are crazy expensive, due to high demand for them. Most dev's go for $50 to $250/hour. Just the MVP of any small app will already cost roughly 100 hours of work. So there you go, that's $5,000 to $25,000. A fully functioning app will cost 1000s of hours, from $50,000 and up.

That's a lot of money if you don't have a lot of money.

So, if you have access to a big money pile, then why not, hire some people to build it. But you'll still be competing with a lot of DIY-ers that don't have that cost, and you'll want to get your costs back at some point in the form of revenue. If your development costs are so much higher than the rest, you'll also need to make a lot more money than the rest. Possible, but hard. Especially in the beginning.

Now, the worst you can do is contact a developer and ask them to work for free in return for a 50% share of profit, while you get the other 50% because you came up with "the idea". It's become a startup trope and it's ridiculous. The market value of an idea without execution is $0. So either you get REAL skills

(an MBA != skills), or get money to pay developers. Paying with future equity is ridiculous, it's like paying with a $5 lottery ticket. Don't make a joke out of yourself! Nobody works for free anymore. You have to understand, as a developer you can now make $150,000/year as a starting salary in San Francisco. Do you really think anyone cares about your startup idea of a random person such as yourself? Not unless you have something to bring to the table, like money or skills. Some people used to add connections to that list, but I don't really believe networking is important in this age anymore. What's important is the product and to make it you need money or skills.

To build, should you learn to code?

Yes, I'd recommend you learn it. It's only getting easier now. Learning to code seems steep for newbies. But people approach it wrong. You can probably ride a bicycle, right? When you started "learning to ride a bicycle" did you think you'd be Lance Armstrong? No. And you probably aren't. You can just ride it, but you're not competing in world championships.

It's the same with learning to code. It doesn't mean you have to be great, or even good at it. Just know some bits so you can throw stuff together. When I code, every day I have to Google how to do stuff I don't know. Coding is continuous learning. You can ask any programmer and they'll answer the same. The good thing is, there's so much information on the internet nowadays. Almost every problem you face, someone else probably had before you.

If you're looking for ways to "learn to code", I'd say don't go for courses, bootcamps or mentors. They usually cost a lot of money and they don't teach you the core of coding: figuring it out yourself. That's the biggest skill. Fiddling for hours to days to get something working. Not giving up and keep trying. And then suddenly: EUREUKA!

If you want to learn to code, my advice is: try to build your idea with HTML and CSS and some JavaScript and see how far you get. Just Google every thing you don't know. Start with "how to make a HTML page". Then "how to make

text colored in HTML". Then "how to make a button in HTML". Etc. Keep searching. You'd be surprised how far you get. This is how I (and many others) have learnt to code. Figure it out for yourself.

If you really don't want to learn to code, read on as I'll discuss how to build a basic app without a single line of code later in this chapter.

Tools

Which tools should you use to build?

I'm a strong believer that right now you should use whatever tool works best for you. What tool do you already know? You've already worked with Ruby once? Was it fun? Use that. You've already worked with PHP? Then use that.

What you should definitely NOT do is listen to programming hipsters on the internet telling you which language is best.

Here's a little secret: The people discussing what programming language is best are not shipping products. The people who don't care what programming language they're using are shipping products. They'll use whatever tool they need, whenever they need it.

So use whatever is easiest for you to learn or work with. And then switch whenever you feel you've outgrown a language. But honestly, unless you're programming spaceships, it's pretty hard to outgrow a language. They're all based on the same principles of computing. You can build anything with most languages really.

- Facebook was built on **PHP**
- Twitter on **Ruby**
- Google on **Java**
- Reddit on **Python**

- Hacker News on **Arc**

The thing is, you're not building for the enterprise here. You're building a first product that might grow into something bigger and then turn into a startup company. You can always switch. Twitter switched from Ruby to Java after they kept going down. Twitter still exists. When Facebook became popular, it became overloaded with users, so they wrote their own engine (called HipHop) to speed up PHP. And they started writing critical parts that needed more speed in other languages.

The point is, it didn't stop them from being successful, so surely it won't matter for you.

What if you don't code? I'd recommend you to learn to code of course. But for you, I've written entire part on how to build without code, further down in this chapter.

My "light" stack

What I use personally grew out of my limitations. I wasn't educated in Computer Science or even programming and I simply used what I knew a little bit about. I didn't listen to everyone telling me what hip new framework (anything.JS etc.) or language (Ruby etc.) to program. I knew some PHP from making some WordPress sites before. And years before that I making my websites interactive with Perl, and it looked pretty similar.

Now the point is, don't go learn PHP. But use what you already know and see how far you get with it. And move to the next language or framework if you're starting to reach its limitations (which seems very hard with most modern languages).

The basic lite stack is a front-end (client) that is built with HTML, CSS and JavaScript. You then use the JavaScript to communicate with the server by making a web request. That request is received by your back-end (server). This back-end can run anything. I use PHP, but nowadays you can also run Java-

Script on the server (with Express or Node.JS for example). You separate the back-end (server) from the front-end (client) for security reasons because you don't want to let your entire user database be viewable by one user, right? The back-end code connects to your database (SQLite, MySQL or PostgreSQL are great).

SQLite specifically is great because it doesn't require you to install a lot, and when you make a database, it's just a file. It's very transportable. You can literally copy the database file from and to your server. There's misconceptions about SQLite that it'd be slow or not scalable enough. That's bullshit. In many cases, SQLite is now **faster than the filesystem** (!) itself.

There's a giant trend now to merge the client and server-side programming with frameworks like Meteor, React, Vue etc. And I support it. But at the time of writing this book, it simply is too obtruse, convoluted and complicated for beginner programmers. It's a rabbit hole really. We don't know where it'll go really: will we get more people using basic light stacks like this one (which still separate front-end and back-end) or will people all move to frameworks like React.JS? Just because a technology is newer, doesn't mean it's better though. My guess is we will probably merge front and back end at some point. But at time of writing this book, things are changing too fast to use it for me.

A good tip to choose a technology is its age. If a technology has been around for a decade or more, it's probably working well for people. PHP is over 2 decades old and JavaScript similar.

Does it matter what stack you use? Again, not really. Just use whatever works for you. In my case, that's this light stack. I keep it simple. Most companies will eventually built their own framework over time, and in a way that's what I did. But it was never my plan. You just streamline functionality that you keep repeating. Remember, you can always switch the stack later. It'll hurt but it's completely possible.

Why are people so obsessed with tools?

There's a recent trend of people becoming viciously obsessed with discussing tooling. What language do you, should you and will you use? Why all these tool discussions when it's not that important for shipping a product? I think people are obsessed with tools because it feels like they're actually doing something productive. Because when they figure out what tools they should use, they'll go learn that tool (or language) and build their product right?

That's the idea, but it's stupid because it never happens. They get stuck in this endless research. They'll learn a new language, then switch to the other one. Because this new language, tool or framework "may be a better fit" for the product, that is the product they still have to build and ship. Not any of these people ever finished what they wanted to make. And the problem is, every week a new framework shows up that promises to make your app or its development even faster and easier.

All of this stuff simply takes away from your goal, which is shipping a product and selling it to users and getting revenue from it. Who has this problem more than anybody? Software engineers. It helps to be a bit business-ey here, because business people always care most about revenue. And if a profitable company is your goal, you should too make that your first priority! Not the tools.

What about the people that finished and shipped? They mostly never cared in the first place about tools. They weren't discussing which tool to use. They just made something with whatever tool seemed good enough. And they switched it whenever their tool limited them and they needed something else. They're "toolset ambiguous".

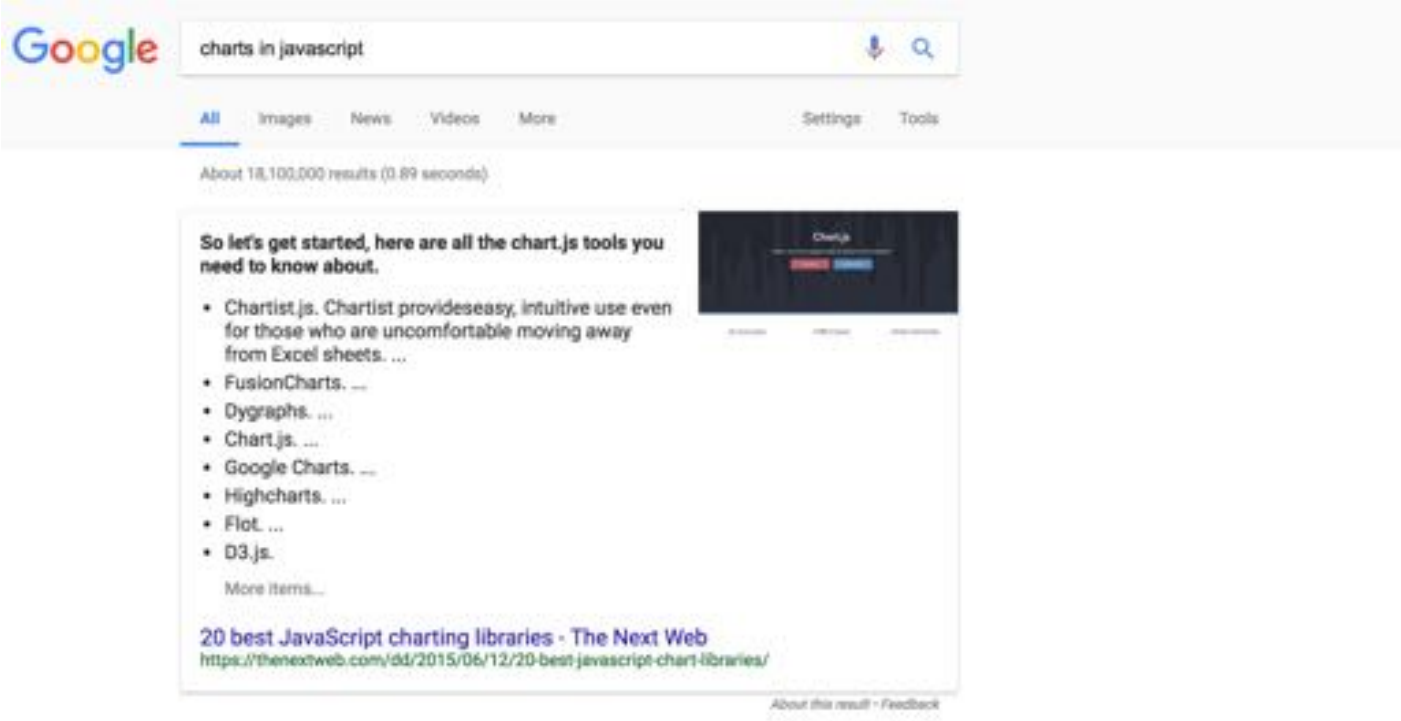
So, stop asking that question "what do you use to make that?" or "how did you make that?". It's an inherently stupid question. The question should be "why did you make that?". The philosophy behind something is way more important than how they made it. You can copy their tools, but you'll never be able to copy their WHY (which is what makes people and their products great).

How do you evaluate if a new tool is useful for you?

There has never been a time during which there has been as much innovation in programming languages as now (except in the post-war 20th century with the rise of computing itself.)

The center of programming is on the web now. There's new languages popping up every month and new frameworks every week. This makes it hard (especially for newbies) to judge what language or framework to invest time in to learn. You learn a new language, but it might already be passé by the time you finally understand it.

My approach is that I only learn what I directly need now. Let's say I'm building a new app where I need to make beautiful charts in HTMl quickly. I'll try it on my own first, but if it's too much work, I'll just Google **"charts in JavaScript"**:



I'll find D3.js, the dominant visualization framework for JavaScript and learn some basic stuff to build a chart.

The thing is, I only learn what I need when I need it. Instead of spending months on learning an ENTIRE language, framework or tool. I just learn the

bit that I need now. This is a much faster and leaner approach which will save you time and make you more productive. And actually ship your product.

I don't have so much of an inherent interest in programming that I'll go and learn stuff just for fun. I know many people do. If your goal is to ship, that might be a disadvantage, because you'll learn lots of stuff you don't necessarily need. And you'll be playing around with tools more than you'll be finishing your product.

Native vs. web

Web

A web app is a website that has the functionality that makes it an application. It can look the same as a native app, except that you access it through a web browser like Safari or Chrome (on your phone). You'll probably build it with HTML, CSS and JavaScript.

Native

A native app means that you program it in the language native to the device you use. And it'll be an actual app installed on the device. For an iPhone that means you'll build it for iOS and it'll use Objective-C or the new simpler language Swift. That's usually what people mean when they say "mobile apps".

User experience and development

In general, a native app's user experience feels faster than a web app. Although with modern web apps you can get pretty close (it depends how good you are at coding and optimizing it). In general, development of web apps is faster, but the user experience is slower, whereas development of mobile apps is slower, but the user experience is faster. A native app has the disadvantage that users have to actually install your app. That takes another user action. Wheras with a web app, all you have to do is click on a link from another website (like Facebook, Twitter or even The New York Times) and it'll open up. A web app is

nothing more than just a sophisticated website that is made to look like an application.

Updating native apps vs. web apps

When you make a change to a native app, you need to deploy it to Apple and Android's app stores. It needs to be reviewed by their staff and it might take days before your update is pushed into the store and weeks until all users download and update it. Instead, updating a web app literally means uploading the new version to your server, and every user that loads it will immediately load the new version. That means instant updates.

Hybrid web + native apps

There's famous hybrid apps that are half native and half website. Until recently, Uber used to be a hybrid. This meant it could launch special features on the map (like for Valentine's day, cars in the shape of hearts, as this was just some HTML they changed), without having to deploy an update to the app stores.

A more recent example of somewhat hybrid apps is React Native (which probably is outdated by the time of reading this), which is a framework that lets you write platform-independent code (for both iOS and Android) in a HTML, CSS and JS-like style to make native apps.

What are people using most?

There's some data that suggests people on mobile devices spend more time using native apps than web. That's probably true. But they'll spend it in major apps like Facebook, Instagram, YouTube, Twitter, WhatsApp and Messenger (remove from this list depending on whatever is popular in the time of reading this book).

These are core apps though. They're giant. You probably won't be able (or want) to compete with these apps. How about using them as your platform?

A future where your web app lives inside other native apps

What I mean sounds crazy but it's where I see this going. People will use your web app INSIDE these apps. What happens if you send a link to somebody on WhatsApp? They see a URL, they tap it, it opens INSIDE WhatsApp's web browser. It's a fully functional browser and your web app can run inside it. A web container inside some big company's native app is the real future platform for your web apps I think. Design for that use case. It's intrinsically viral as people will send your website's URL around and open it from there. They won't have to install anything.

The web and native will merge in the future

I believe that web apps and mobile apps are converging. You see web apps get more and more powerful and getting access to more functionality of the device that was usually limited to mobile apps (for example the iPhone's gyroscope is now accessible through web). Other examples are how recently (2016), iPhone's browser doesn't show entire URLs anymore, like "http://nomadlist.-com/amsterdam-netherlands", but instead shows the name of the site in the URL bar "nomadlist.com". That's obviously pointing towards seeing the web more as apps. In the lon gterm, there are a few obstacles to overcome for the two to converge. Honestly though, it's impossible to predict. Five yers ago, I would have though by now everything would be, but we still have users both on the web and inside native apps.

Learn both

If you have spare time, learn to do both web and native app development. It's a remarkable (and sought after) skill to be able to build both web apps and native apps. And of course, hybrid ones!

What are you able to build now?

This book is about getting something out as soon as possible. Therefore, you should choose the platform where you already have some skills. This way you can get something out as fast as possible. For most people, and definitely for me, that'll always be web. Programming (and shipping) native apps to mobile

devices and getting them into app stores is more work than just building a website, uploading it to a server and hooking it up to a domain name. Do what's fastest for you.

Other people will say I'm wrong there, and you should base it on how users will use your app. That's a good point. But I don't want to see you get stuck learning Objective-C and Swift to get an iPhone app out, or paying a mobile developer $25,000 to build it for you.

Remember: you can ALWAYS go native later. If people use your site already, you can bug them to install your native app later. This is annoying but it's possible. No platform choice has to be permanent.

Building with constraints

I'll now discuss how to build stuff if you're constrained. Having constraints seem like a disadvantage but you can turn them in an advantage. So many people look at the negatives in their current position, but in fact, most of these can be considered a positive advantage.

No money/investment

As mentioned previously, bootstrapping has become an advantage. You keep your costs low naturally. You get 0 of the bullshit attached with VC money. You maintain full ownership over your product and its roadmap (where you want to go with it). You maintain full ownership over the equity so that when you sell it later, you'll get lots of money (instead of just a diluted 5% because VCs took the rest).

And in our times you don't really need a lot of money to build something yourself. A simple web server is $5/m. A code editor like Sublime Text is $60. To make iOS apps, XCode is free. A developer license with Apple is $100/year. It's not a whole lot compared to the money you can make back from it if you get people to use your app and pay for it.

Bootstrapping has become a very viable option for most software-based start-up ideas.

No office

I obviously had to put this in here as I'm a big supporter of remote work. Not having to spend money on office rent is now an advantage. If you (and the people you work with) can work from anywhere, it also means you now have access to a worldwide pool of talent. Starting fully remote is much easier than switching to remote after you already have an entire set of office buildings and workers.

No coding skills

Not being able to code means you can use off-the-shelf tools to quickly prototype stuff without losing yourself in giant codebases. As you'll see later in this chapter, you can build an entire landing page, get data form users, process that data, charge money without writing code. This means that you can spin up lots of different MVPs and see which one sticks, without much effort. Although you're more limited in *what* you can make, you'll be shipping faster than people coding stuff for months.

Once one takes off, you can always learn to code on-the-fly, or get someone to help you build stuff out.

No connections

You're not a famous startup person with lots of connections in "the scene". Well, guess what. Those connections are mostly bullshit any way. And if you're "outside" the scene and not famous, it means you can act like an underdog. You'll be more indie than most and, guess what, people LOVE to support independent underdogs. You're fighting against big companies and people want to see you win, that is, until you become a big company yourself (the cycle of life). The "connections" people have after they get successful also put them in

a monoculture bubble, which you're not in (yet). You think more freely and that's better for creativity.

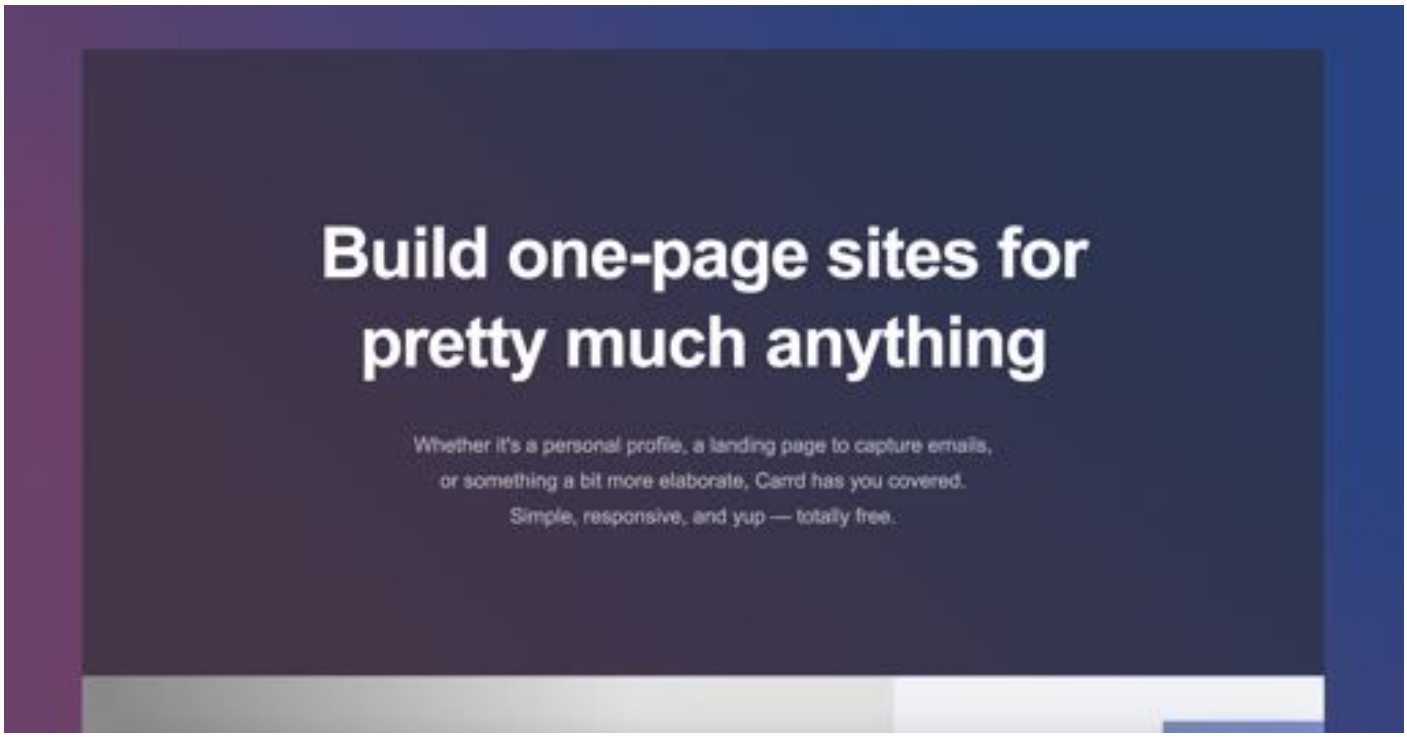
Building a startup without coding

What if you really don't want to learn to code? That's what this part is about. You can still do it. I'll show how you can build a basic prototype with off-the-shelf tools. You'll be able to make a landing page, let users enter data, manipulate and process it, charge them money, message them and add a task for your contractor (or you) to execute, and without writing a single line of code.

I'll discuss tools to use for each section and give some examples. These tools are obviously subject to change and might be out-of-date. If they are, the general concept remains. I'll give you some guidance. It's up to you to connect everything and execute. Be creative!

Building a landing page

To get your users in you need a landing page. Luckily these days they're easy to build with existing website builders that give you templates to customize.

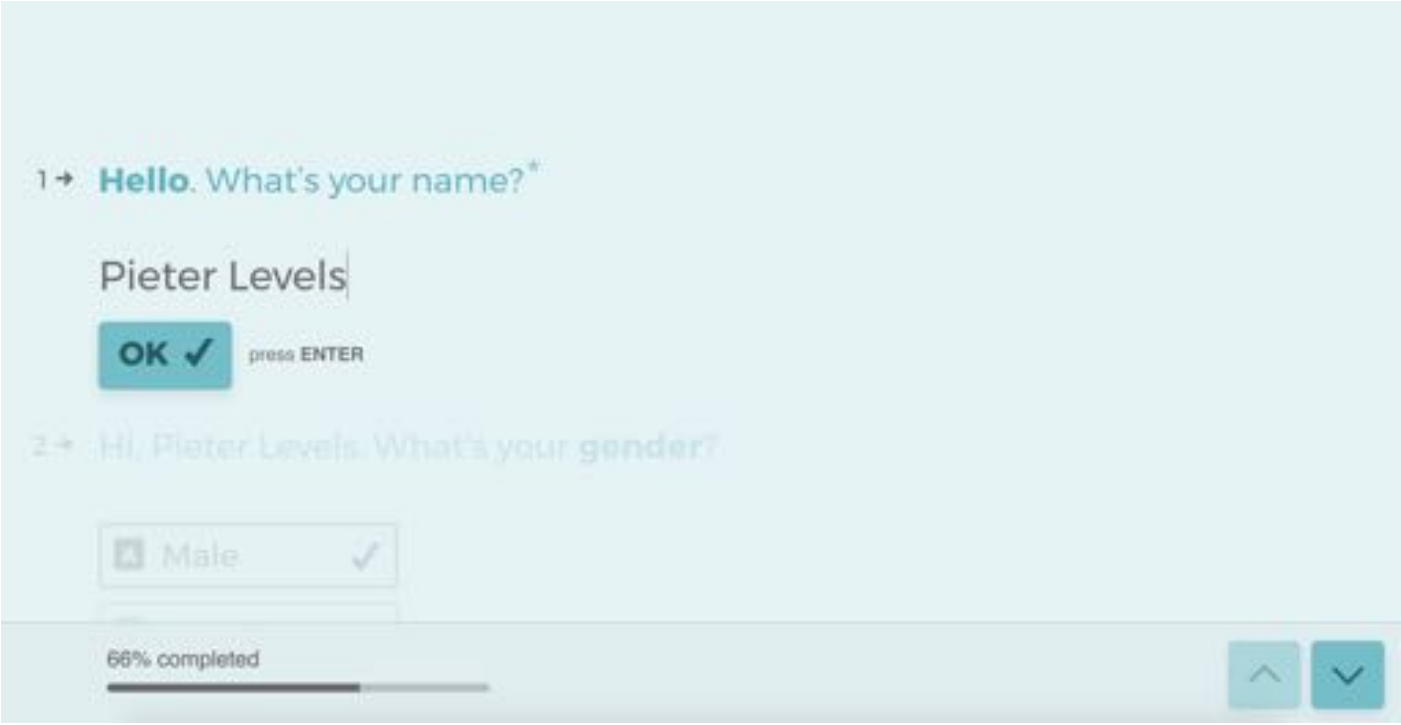


One of the most famous is **Squarespace**. A more recent indie website maker is **Carrd**. Others are **Tilda** and **Wix**. If you need a bit more freedom and the ability to add custom code later, try **WordPress**, it allows you to write PHP or JS to customize your website and add features later on easily.

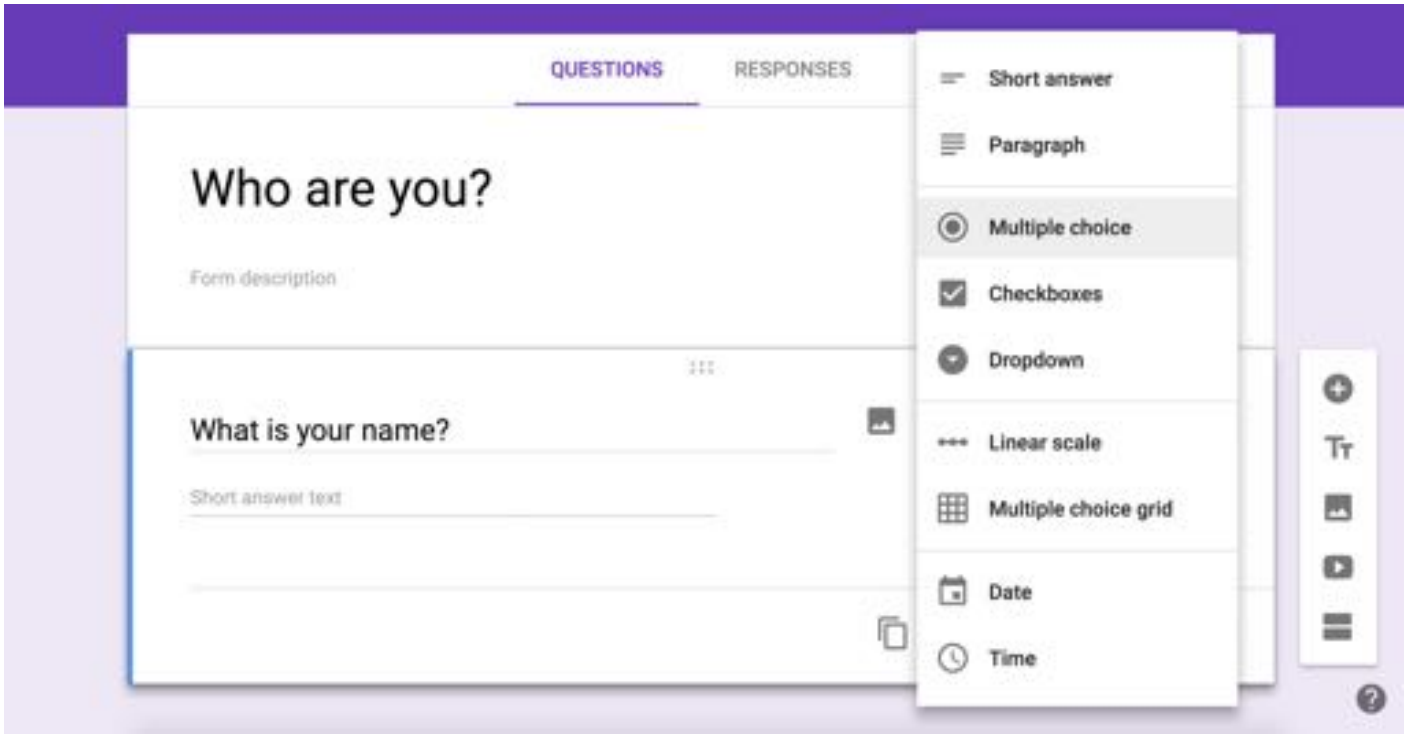
You'll want to use your landing page to explain your product or service. And from there lead them towards a so-called call to action (or CTA). What do you want form your users? Do you want to save their name and email? Do you want them to pay you money? Adding a big colored button in the center top of the page as a call-to-action will lead them click there. When they click, link them to the next part (which in most cases means, collecting data from your user).

Accepting user data entry

To get people to enter data, and then save it, you can use **Typeform** or **Google Forms**. Typeform has better forms, but Google Forms lets you directly put the data in a Google spreadsheet, which is great.

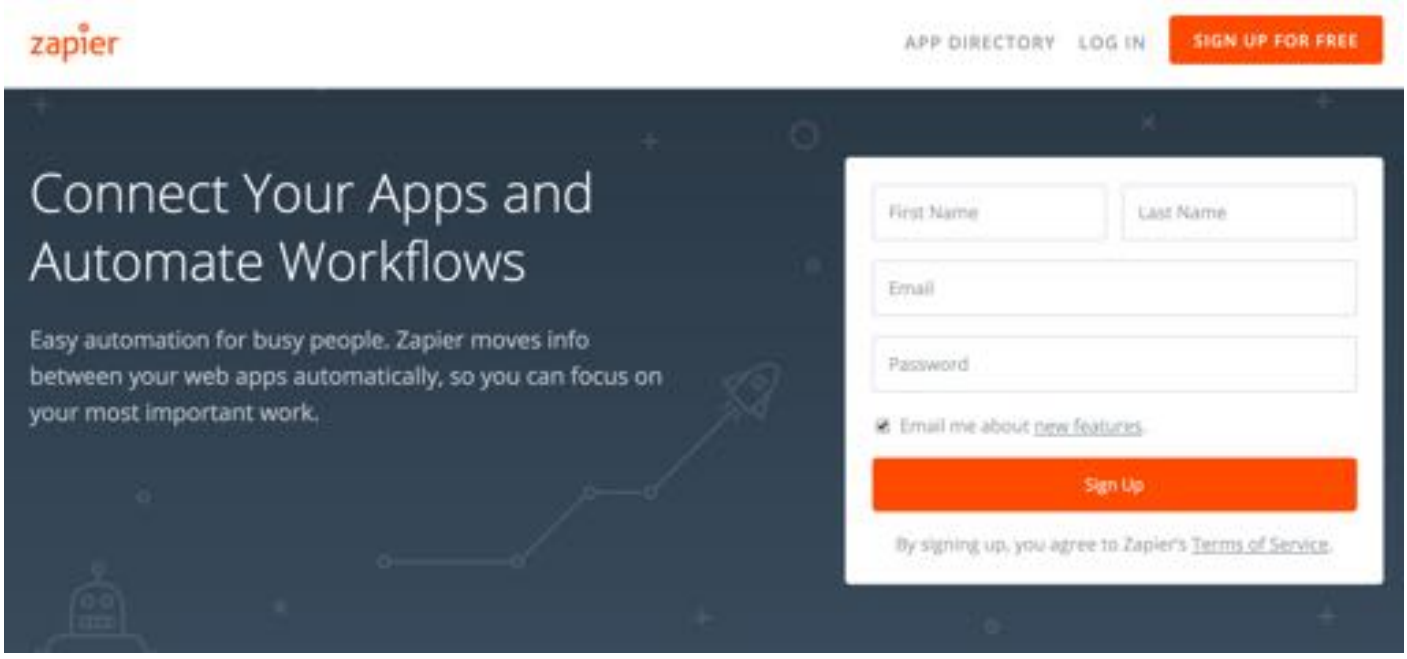


Google Forms is a bit less intuitive and more formal:



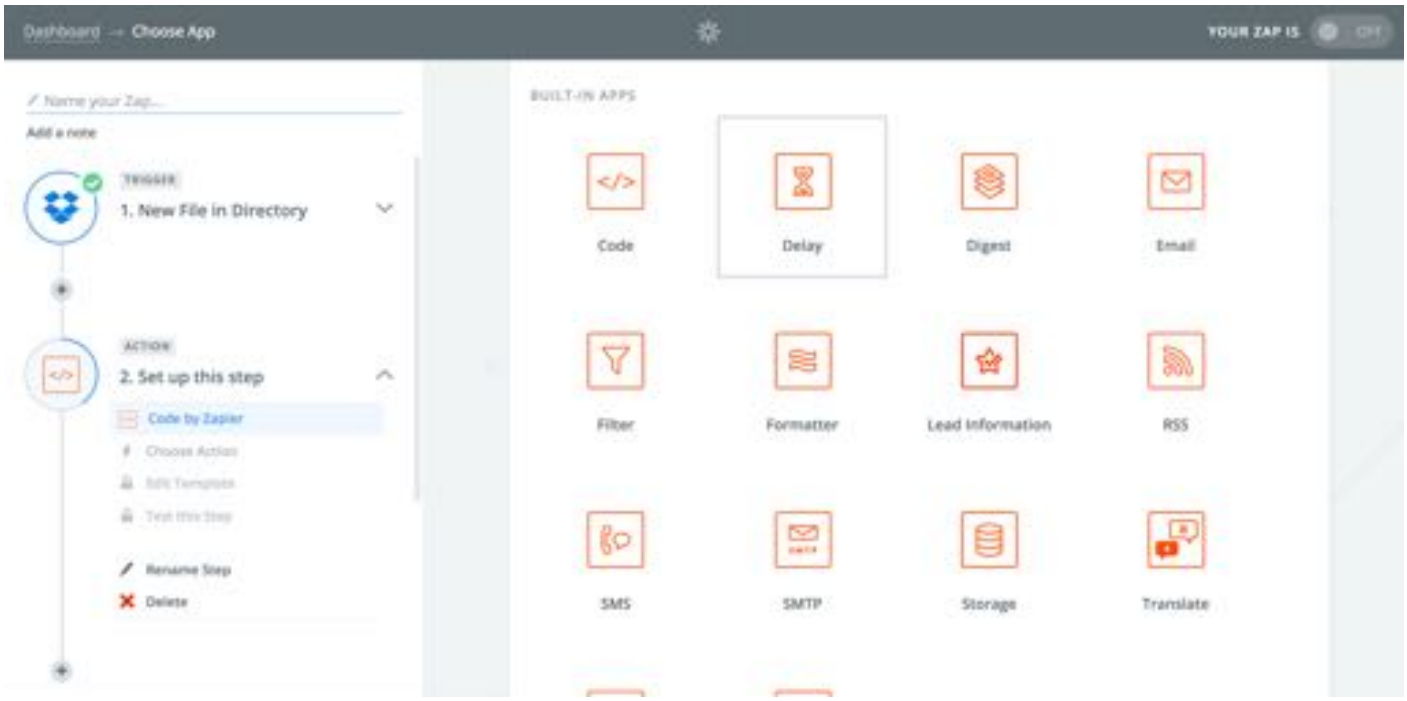
Processing & manipulating user data

After you have the user's data, you probably want to do something with it. Like save it, or process it and then save it, or process it and do something with as a next step. Here's where **Zapier** comes in. It's magic.



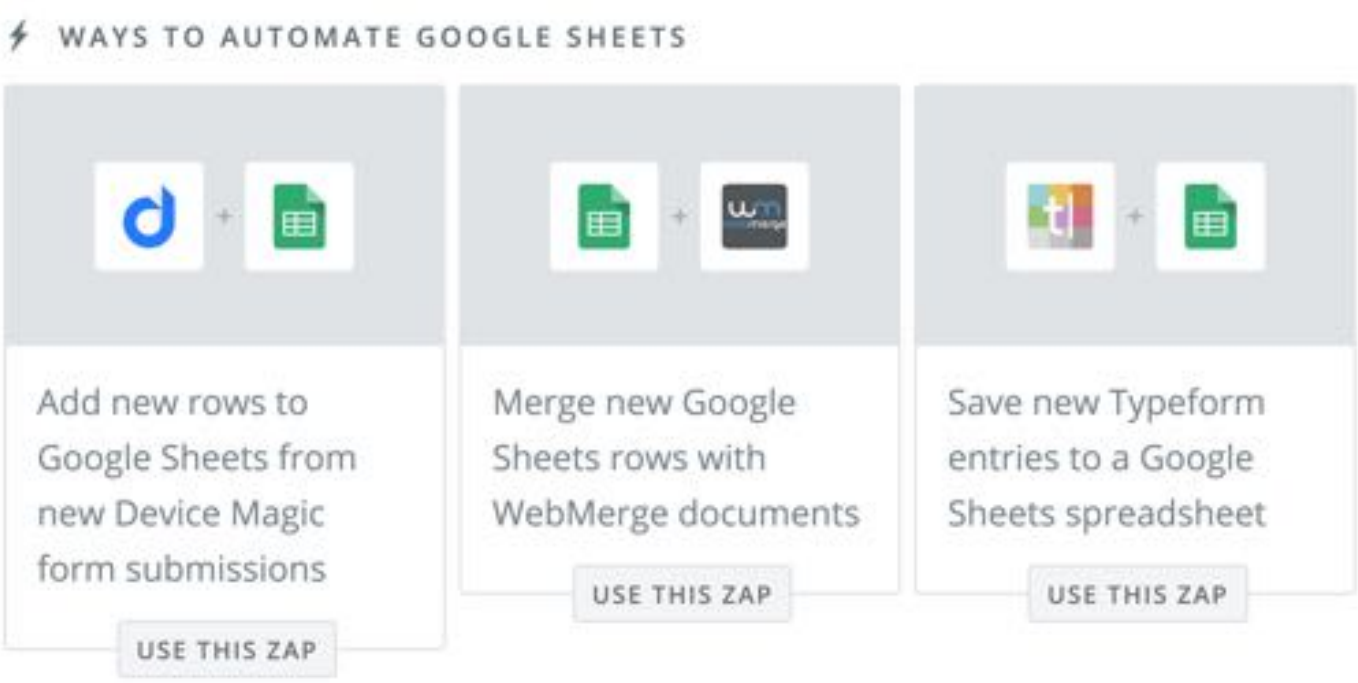
Zapier is a web app that lets you connect most web apps you know with others. It's like the glue in between. It can simply transfer data (or parts of data) it gets, like from a Google Sheet, a received email or a Stripe transaction, and

send it to another service. Or it can process and change the data in between, it even support basic JavaScript code:

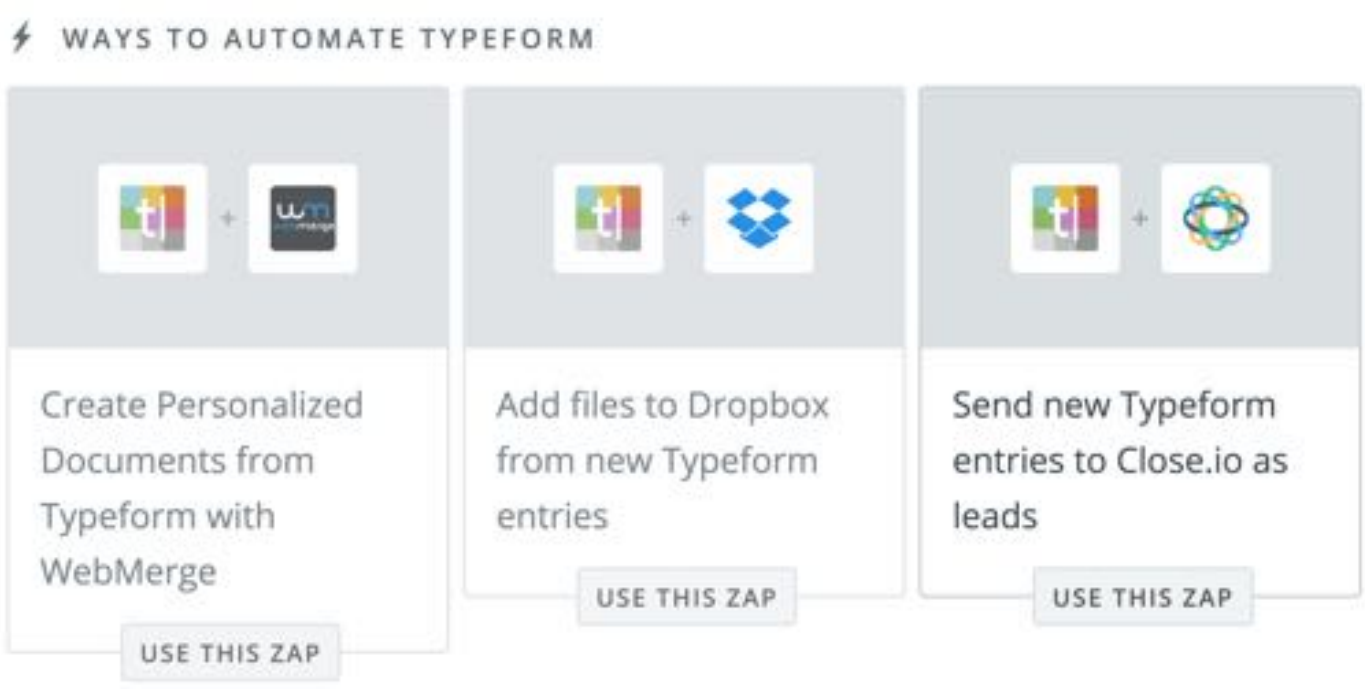


You can make your own flows to do whatever you want them to do. And they'll keep running perpetually. This is a lot like scheduled cron jobs you have on the server, but again, without coding yourself.

There's lots of pre-made flows (so-called "zaps"). Like taking data from or sending it to a Google Form:

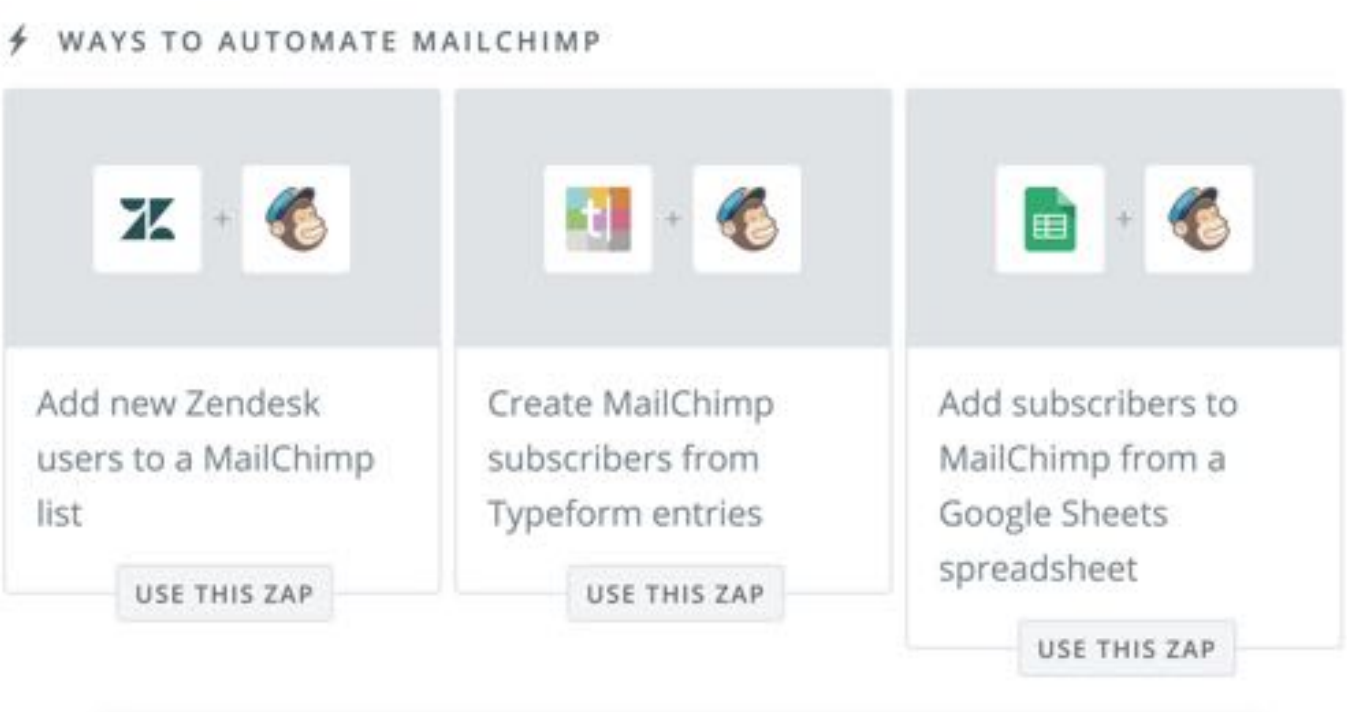


Or taking the data from Typeform and sending it to Dropbox:

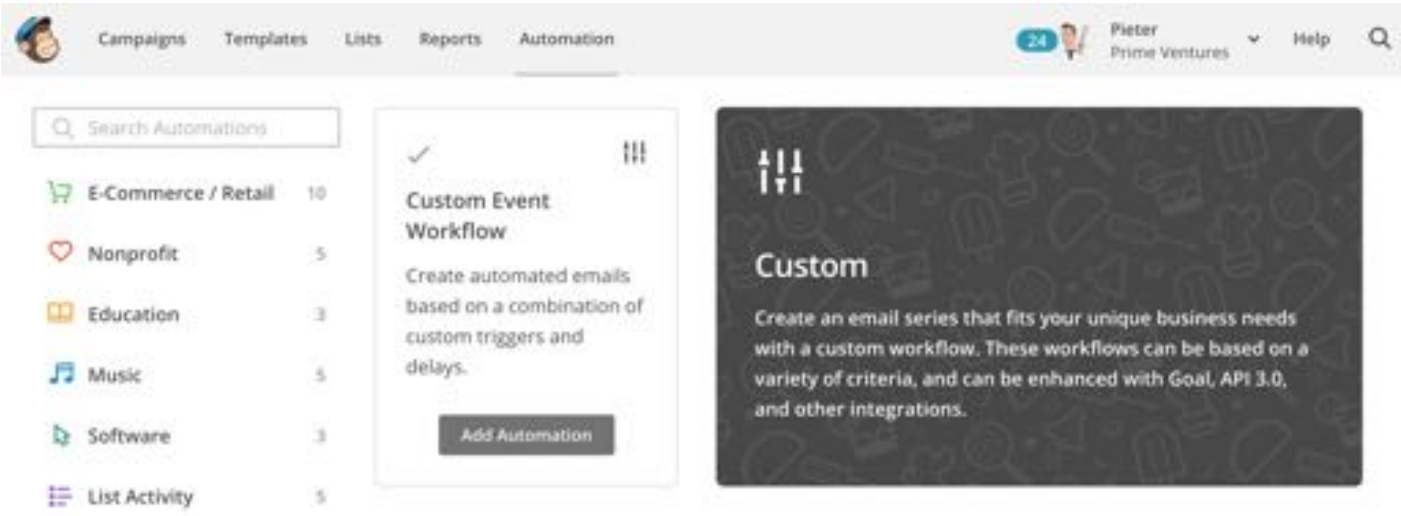


Contacting users

After you have processed the data, you might want to contact your users. Luckily, Zapier supports MailChimp, which means you can now automatically send emails:

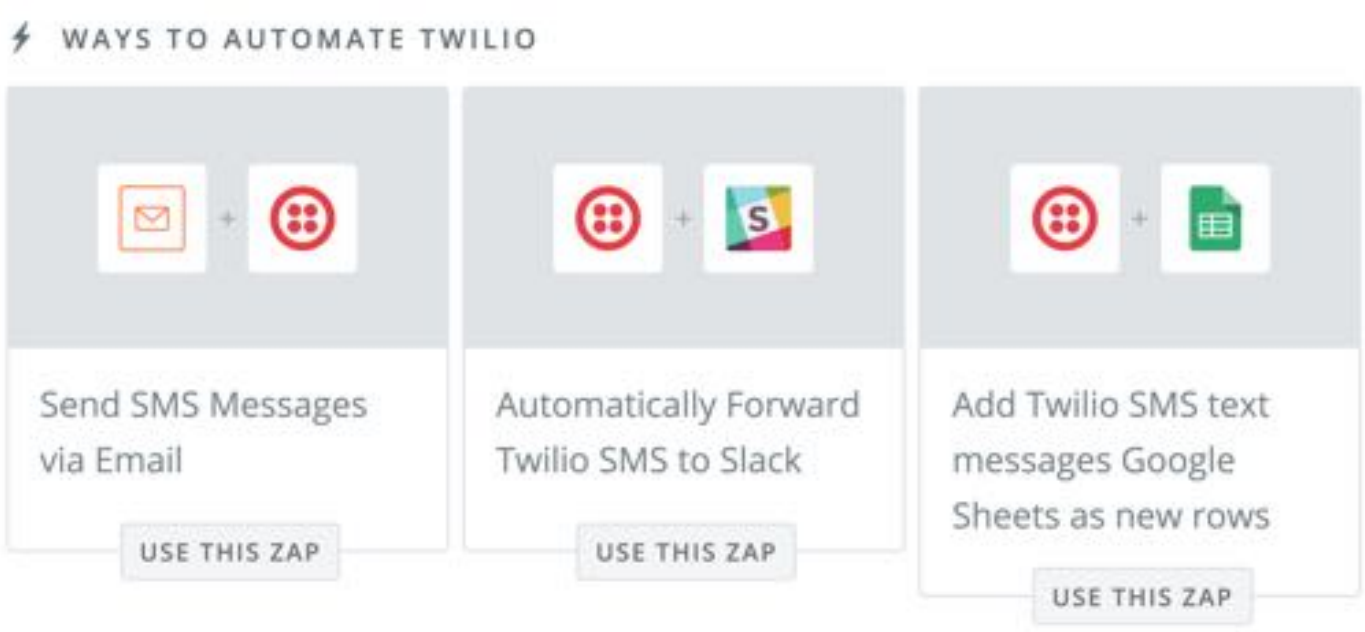


Or you can just give a nod to MailChimp to do something. In turn, MailChimp has advanced built-in automation too:



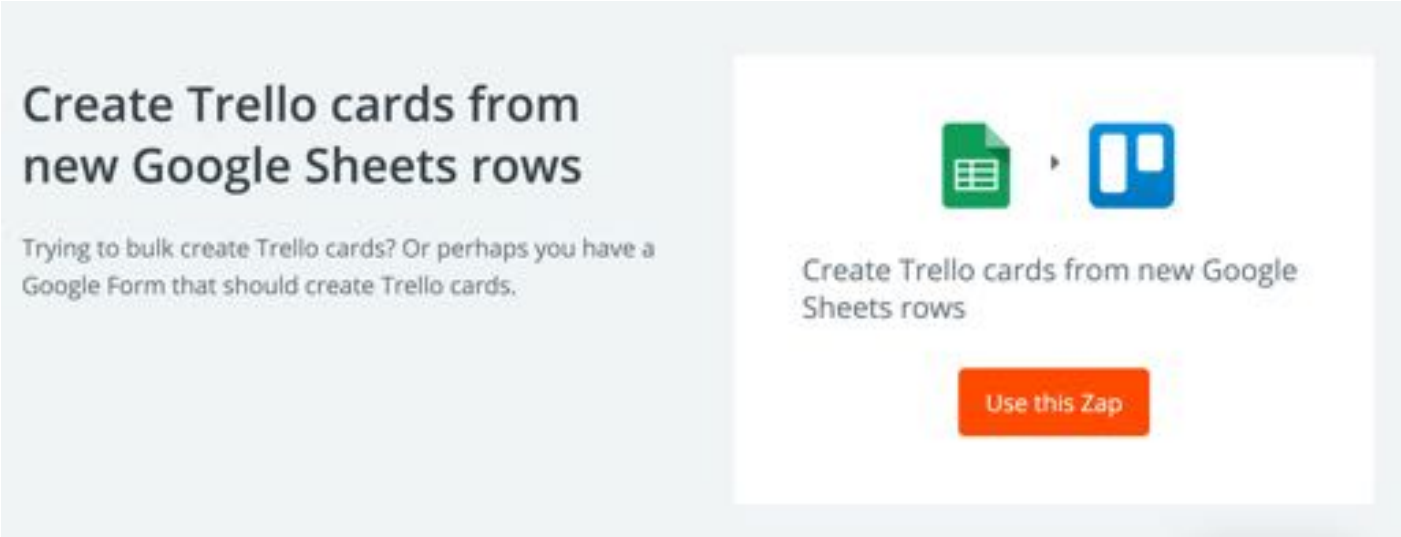
That means anything you'd want to do after a user lands on your site and enters data is possible. Like send them an email 14 days after with a link to another web app. Or send them an automated PayPal invoice within an hour of signing up.

It doesn't stop with email. You can send an SMS message or robot-voice phone call to your user with the telephone API service **Twilio**. Then you can even save what they respond on the phone. And in turn, send that to another web app!



Making tasks for contractors

What if you need a human to process some data or work with a user? Send it to productivity software Trello, where you add it as a to-do list item for your contractor to execute:

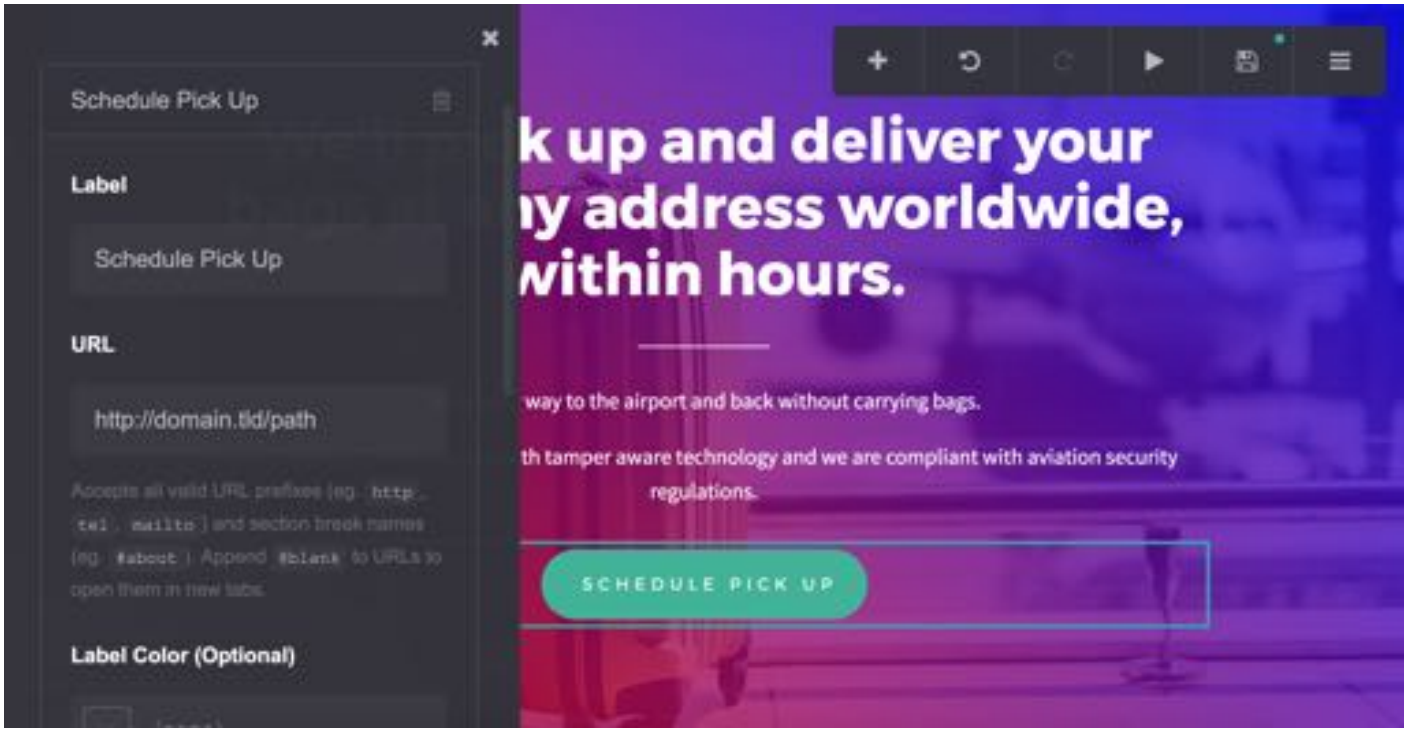


Charging users payments

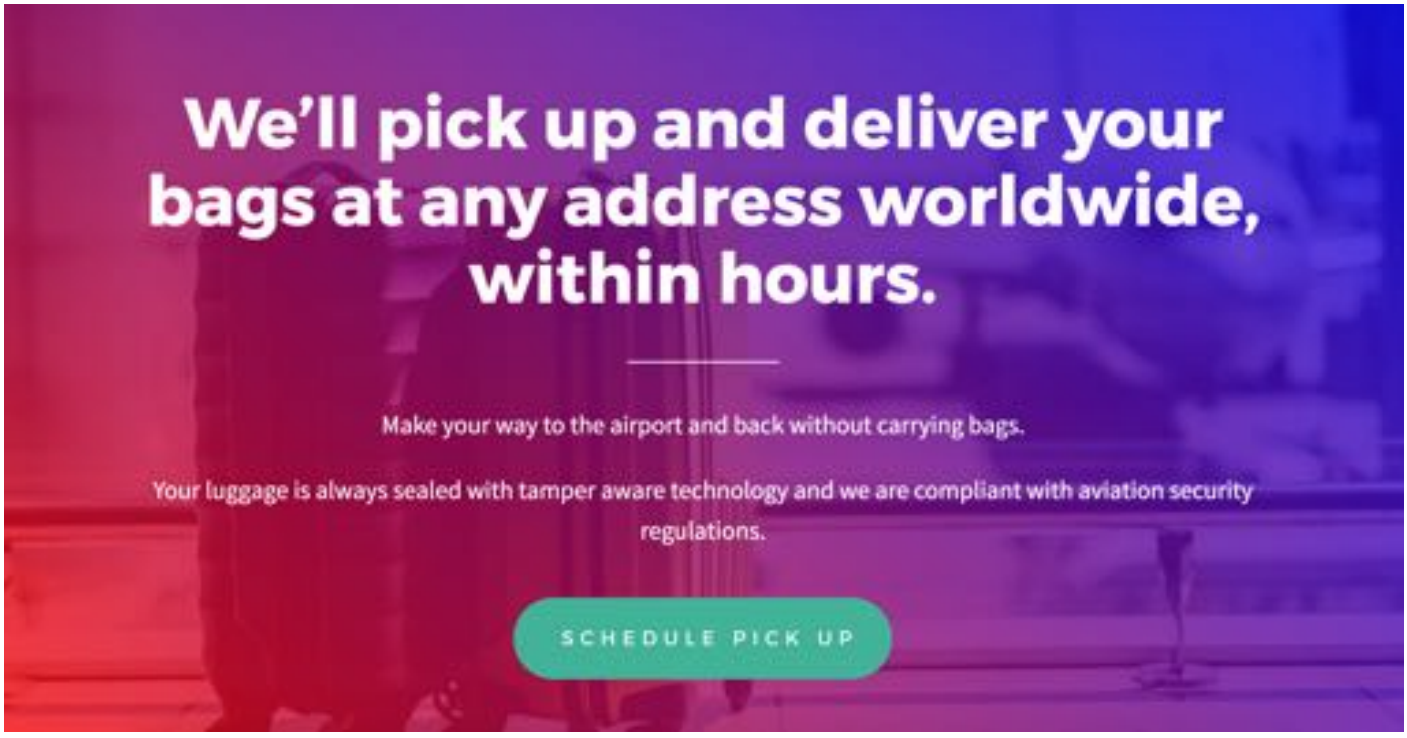
One of the most important parts is actually getting users to give you money. Until recently, this was reserved to people who could code payment logic together. Not anymore. Website builder **Carrd** supports **Stripe Checkout**, which means you simply connect your Stripe account and you can accept payments on your landing page.

Okay, let's try build something

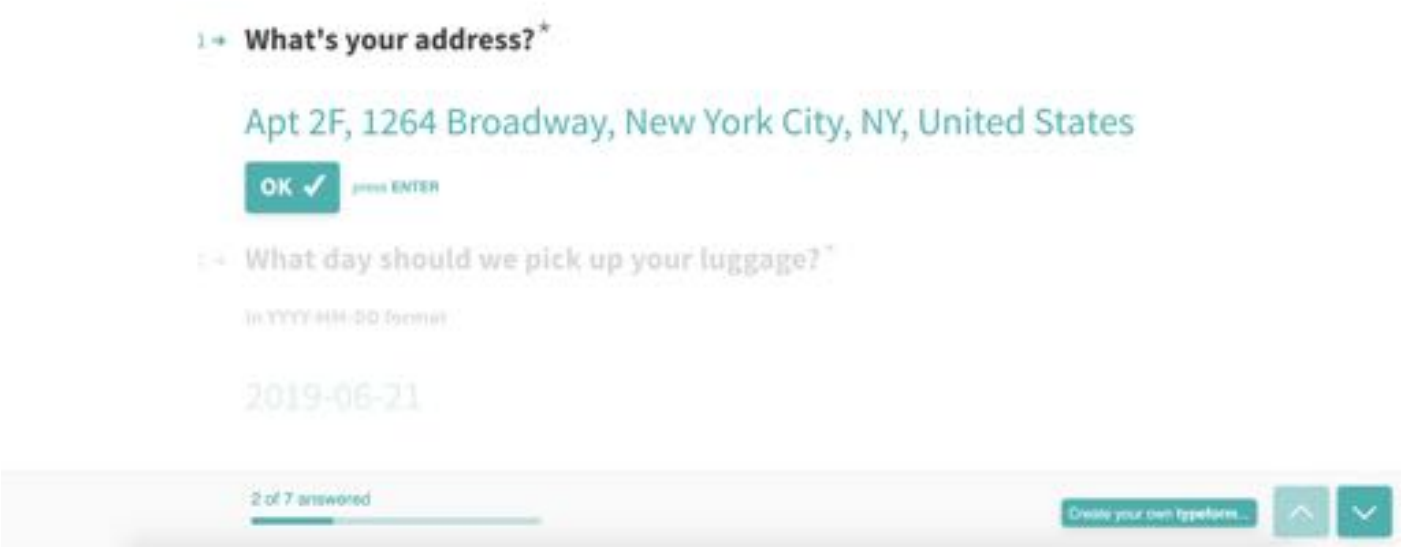
To prove to you this works, I want to build an MVP for a startup that picks up luggage anywhere in the world (where you are) and ships it to your destination. This way, you don't have to go to the airport carrying all your bags around. We won't write any code. Let's start making a landing in Carrd's editor:



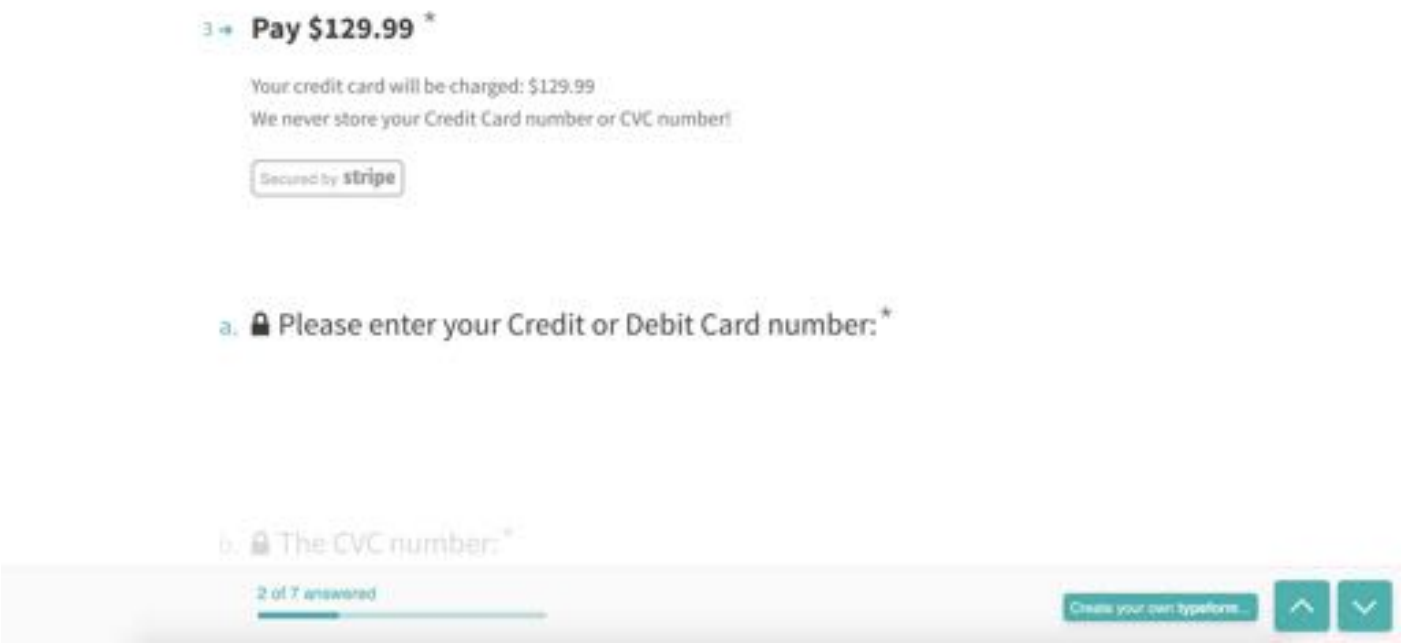
Here's how the final version looks:



Then when we make a new Typeform asking the user for their address, pick up date and we embed this Typeform into Carrd:

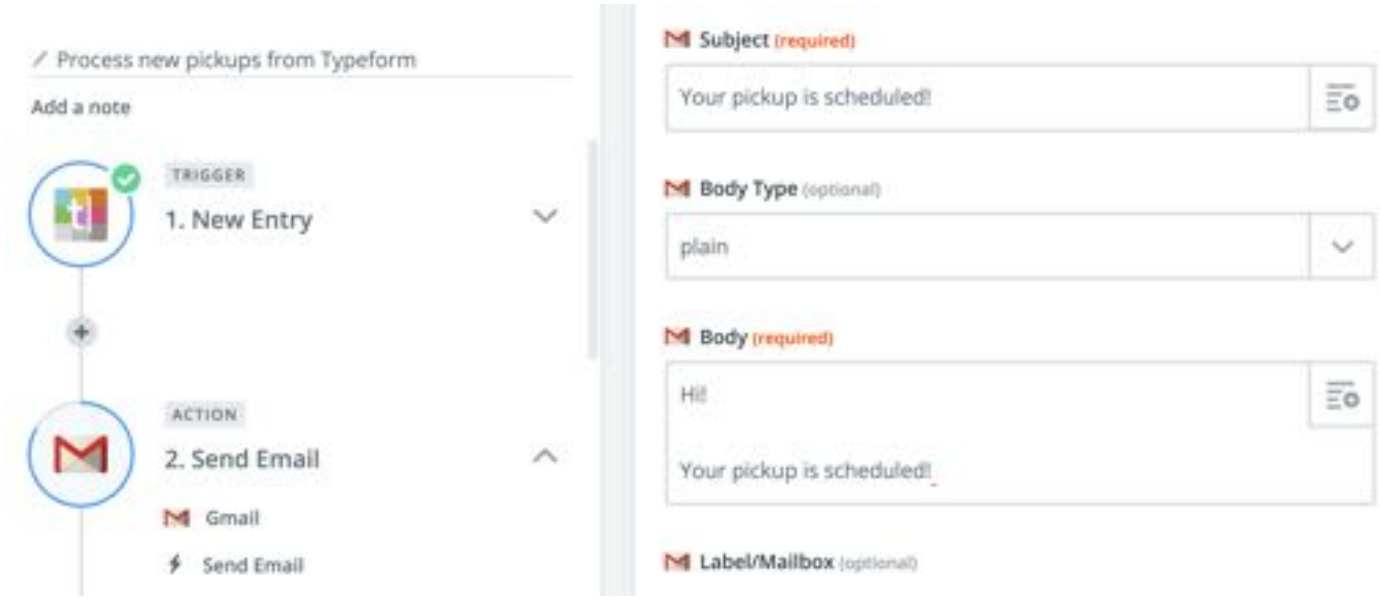


The good thing about Typeform is it also supports Stripe. So we can also charge the user's credit card from within (!) the form:

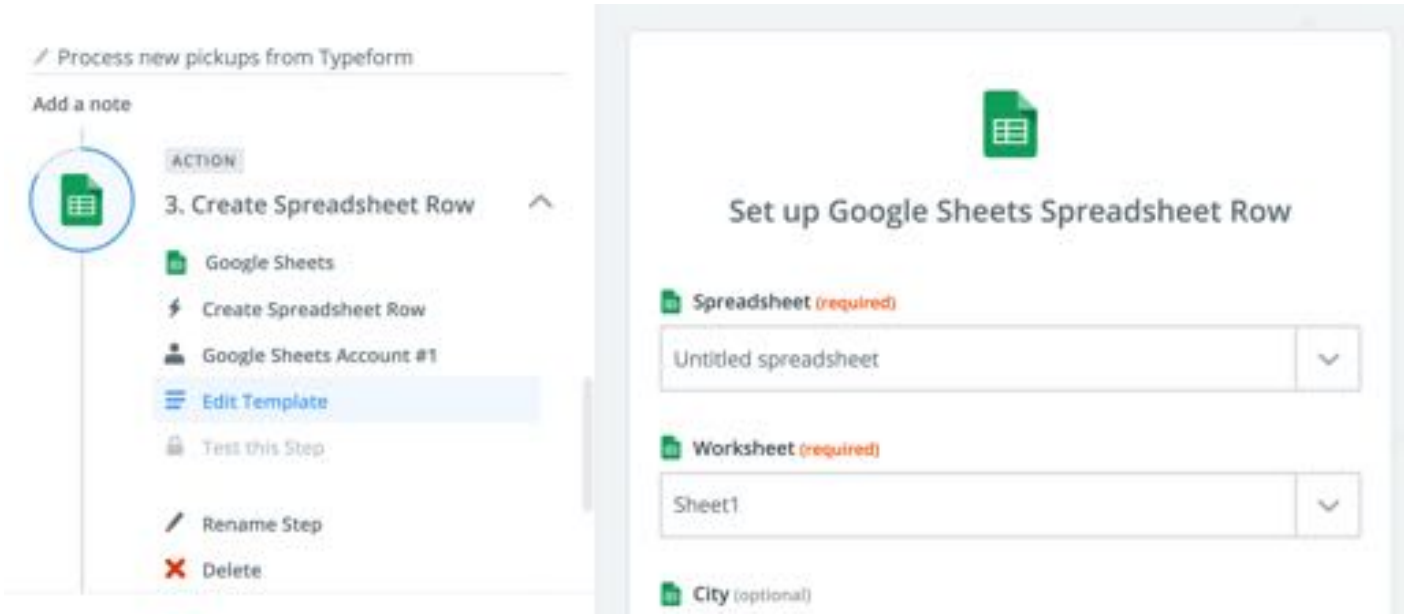


Now let's go into Zapier and make sure that after payment happens we do a few things. This Zapier zap only runs if a new paid entry hits the Typeform.

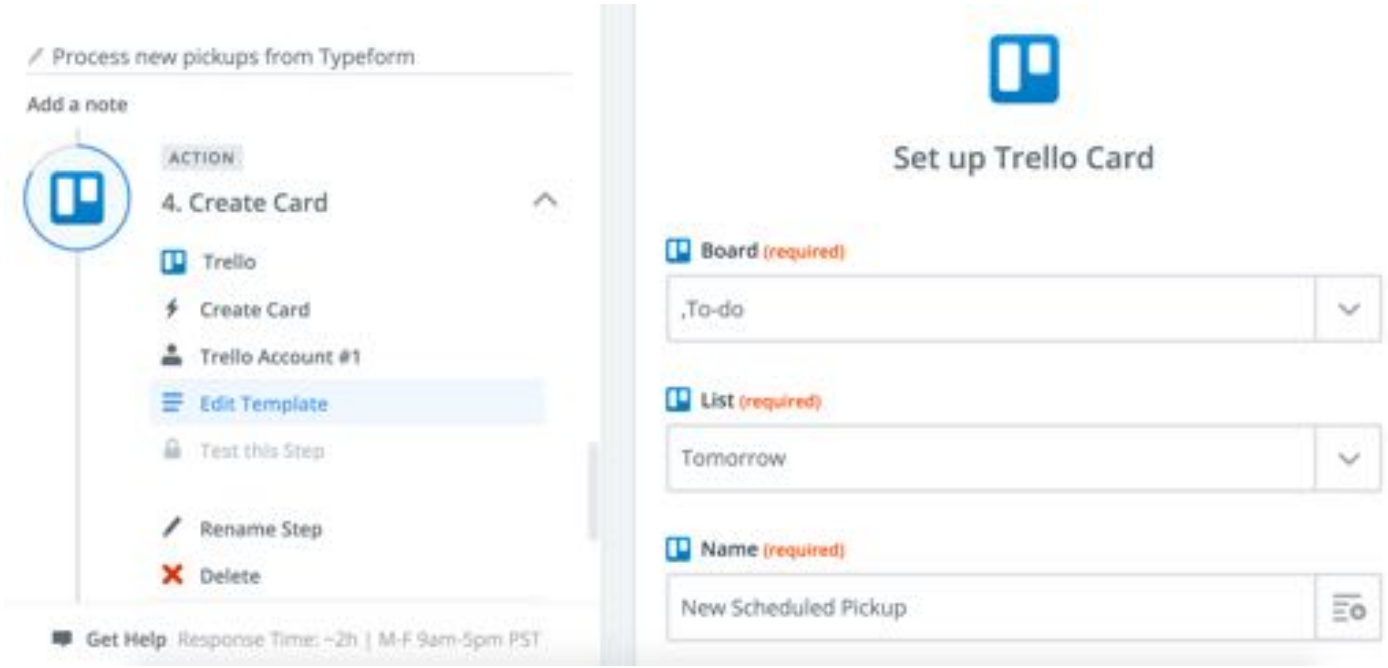
We automatically send an email to the customer using Gmail with information about the pickup:



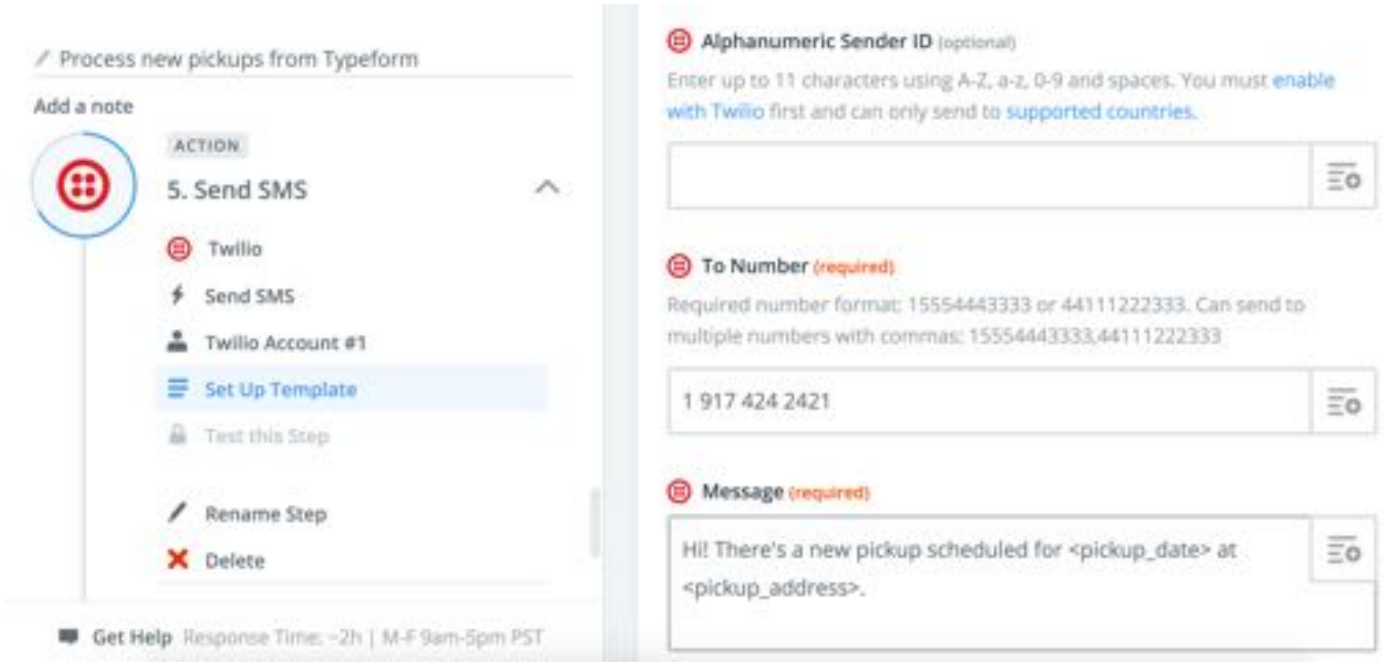
We automatically add the pickup order to our Google Sheet with active pick ups:



We automatically add a task into Trello for our pick up contractor to get the luggage:



We also automatically SMS our contractor an alert with Twilio that there's a new pick up ready to be fulfilled, with the date and address:



See how easy that was? We now have a fully functioning minimum viable product (and actually a basic startup) built. And it took about 30 minutes! You can go a lot more complicated from here. The possibilities are endless. And when you have validated your non-code MVP, you can start adding your own coded parts too to increase the complexity of your product. You can replace the "off-the-shelf" web apps with your own coded scripts.

Let's talk APIs

The entire previous section where we connected all these web apps without code is only possible because of the existence of so-called APIs. They're Application Program Interface. Which simply means apps that can communicate to each other in slightly human-readable data. Zapier's entire app is build around connecting different APIs together.

You don't need to use Zapier to use APIs though. If you can code, you can query APIs too and save their data, and in turn send it to other APIs.

One of the easiest ways to quickly build a prototype (if you can code) is to use thse third-party APIs. They can provide you with data (like cost of living or weather data), platforms to build on (like Facebook and Twitter) and services (like sending emails easily).

Why are they useful?

APIs are ways that computers and servers can share data in a computer readable format. It means that when I open https://nomadlist.com/amsterdam-netherlands a computer will literally read this:

```
<!doctype html><html class="nomadlist no-js pageType-cities user
units-metric "><head><!--<script type="text/javascript"
src="https://www.google.com/jsapi"></script>--><meta
charset="UTF-8" /><title>Nomad List - The Best Cities to Live
and Work Remotely</title><script
src="//use.typekit.net/zfz4znf.js"></script>
<script>try{Typekit.load();}catch(e){}</script><meta http-
equiv="Content-Language" content="en-us" /><meta name="google"
value="notranslate" /><meta name="viewport"
content="width=device-width, initial-scale=1.0, maximum-
scale=1.0, user-scalable=no"><meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1" /><meta name="apple-mobile-web-app-
capable" content="yes" /><meta name="apple-mobile-web-app-
status-bar-style" content="black" /><meta http-equiv="cleartype"
content="on" /><meta name="HandheldFriendly" content="true" />
<meta name="description" content="The best cities to live and
work remotely for digital nomads, based on cost of living,
internet speed, weather and other metrics. For startups that
work remotely and digital nomads." />
```

....etcetera (it goes on for pages). This stuff is nice to look at when the computer visualizes it and interprets the HTML code. But it's hard for the computer to get specific data from it. Because the layout is kinda made for humans.

But now, if I open the same page through an API at https://nomadlist.com/api/v2/amsterdam-netherlands (this URL will probably not work anymore when you read this book but you get the point), you and I, and even a computer, will be able to read this:

```
{
        "name":"Amsterdam",
        "country":"The Netherlands",
        "cost_of_living":"$2,500/m",
        "safety_score":4.3891,
        "nomad_score":4.2183,
        "fun_score":3.394,
        "cost_score":2.238
}
```


Now you can get the city name Amsterdam, the country The Netherlands and how fun it is there 3.394 (out of 5 stars) and display it on your own site. That's useful. But even more useful is if you combine multiple APIs together.

What if I want to find the most fun places from Nomad List, and get the live weather data from somewhere else? To see where it's fun and not raining?

Well, just a quick Google for "weather api" reveals there's a free weather API called **OpenWeatherMap.org** which can give you the current weather and forecast in a computer readable format, just like above. Let's try it: http://api.openweathermap.org/data/2.5/weather?q=Amsterdam

```
{
                "name":"Amsterdam",
                   "coord":
                       {
                           "lon":53,
                           "lat":4
                       },
                   "weather":
                       [
                           {
                               "main":"sunny",
                               "description":"clear sky and sunny"
                           }
                       ],
                   "main":
                       {
                           "temp_celsius":25.5,
                           "humidity":89,
                           "pressure":1013,
                           "temp_min":20.04,
                           "temp_max":26.04
                       },
                   "rain":
                       {
                           "3h":0
                       },
                   "clouds":
                       {
                           "all":5
                       },
               }
```

Yay, it's not raining in Amsterdam! And it's actually 25 degrees celsius! That means we can go outside and have some drinks :)

Now we can combine that weather API data, with the data from the **Nomad List** API and show it on our site. But we can go much much further.

With API services like **Twilio.com**, we can make an entire telephone voice and SMS service by just making calls to their API. I'll spare you the technical details as it's quite some work. But it's not difficult. It's simple and anyone can set it up.

Now start thinking of ideas. You can have a phone number you call that'll ask you some questions over voice, and even have the person answer them (yes Twilio has Speech-To-Text and Text-To-Speech). You can call that number to ask what's the city you should go now with the best weather and most fun.

The more APIs you start combining, the more fun it becomes. And because the heavy load is lifted by the APIs already, you are pretty much just connecting them together into new functionality. Which can be your product!

You can build a business on other people's API's

You can build entire businesses based on other company's API's. Many have done so before you. There's one big thing to remember though: if you become solely dependent on one company's API, you're in a bad place. Without even telling or asking you, they can shut down their API at any time. And that will immediately destroy your business.

More commonly, companies change how their APIs function whenever they want. That means you have to constantly monitor their API and see if it functions correctly. Whenever they change something (even something as small as changing the key of "nomad_cost" to "nomadcost"), it will break your app and you have to change your code.

Conclusion

When building, try to build fast and minimal. Instead of learning new stuff, use the tools you already know to build your idea. Make sure your MVP actually works and is not just a landing page that doesn't do anything. Lose your perfectionism, it'll never be perfect any way. Don't outsource, build it yourself. If you can't code, use off-the-shelf tools and connect APIs together to build it. Appreciate your constraints and limitations, they can be a giant advantage vs. people with lots of resources. Keep your costs low while building and later on. Build for the web first, you can go native mobile later. Don't build on an MVP too long, a good rule of thumb is to spend max. one month on it and launch.

Resources mentioned

- Linode
- Digital Ocean
- Heroku
- Ubuntu
- NGINX
- Olark
- Twilio
- Zapier
- WordPress.com

Your homework

- Rank the list of ideas you made previously by which you think are best.
- Now see which of those best ideas you can execute quickest with the tools and skillset you already have right now — that means without learning anything else now.
- Build the first prototype of your idea, it's minimal, but that doesn't mean not functional. It should do something, either it being a Typeform connected to Zapier or a WordPress landing page or your self-programmed web app or native app. It should have the core functionality working well to be useful for users.

Launch

Introduction

Congrats! You've built something. Now for the most important step, launching it to your future users.

What is a launch?

Many people have talked about it. Some have tried. Yet few have succeeded. It's the launch of your app. After getting your idea and putting all your effort into building it. This is the day.

You show your product to the world in the hope of getting people to use it (and pay for it).

But most people are overconfident. They think it's easy. It is relatively easy if you have a great product. But even then, a wrong launch can mean failure on day one.

That's why the launch is probably one of the most important parts of doing a startup.

In effect, what it means is getting your app in the hands of people so they start using it. And there are lots of ways to go about it. You can get people to know about your app through existing platforms like Product Hunt, Hacker News or Reddit. You can do outreach to press, so they'll write about it. You can try to get people with a big audience to use it.

There are many ways, and we'll discuss most of them in this chapter.

Why even launch?

The idea of having a first launch is to make a big splash, get lots of people to use your app, learn from their usage, fixing bugs, developeing new features, and hopefully getting them engaged so they stick around. You want to bring

lots of people in quickly, so that the app becomes active and so people start talking about it.

It's also a great press opportunity, a launch itself means you can get press from it. Even if it just means making your app publicly accessible by deploying it. If you don't launch it, nobody is going to show up and use it.

Launches can and mostly are arbitrary these days. With continuous development, you kinda launch every few hours, right? But you need to pick a certain day and time to make this into an event. Let's say you have lots of new updates and you feel like now it's almost like a new product. Then you can launch it and tell the world.

How fast should you launch?

In most cases, you should launch as fast as possible. Because you want to have people using it. Why? Because then you can figure out if they use it, how they're using it and if they don't, why not? You can find bugs you haven't found yourself. And you can get direct feedback from users to improve it.

Preparing your app for a launch

One of the most important things to do before launching is seeing if what you made is actually launch-ready.

Many people have the wrong idea about the term minimum viable product (or MVP). You can't just throw something together and call it an MVP and put it up. It needs to actually work.

An email sign up box *is not a product, it's an email sign up box*. Don't launch it. That's ridiculous. I see it too much and it's completely useless.

So make sure it has a function. It does something well.

Fix most bugs

Then make sure that most bugs are gone. Go through the flow of what a new user will go through many times. Act like different users. If you send out emails after a user signs up, see if they actually get it. Go through that flow countless times, until you're sure it actually works.

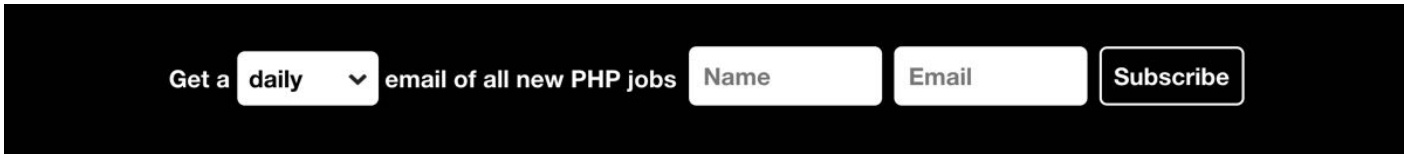
If you think all bugs are gone, make sure you get a few more people to check it out. It's easy to miss things because you're so deep in your own app's development.

Add an email box

One of the smartest things Product Hunt did, and many copied, was to add an email sign up box at the top of their page. They've since changed it. But it was smart as it gave them the ability to re-engage users.

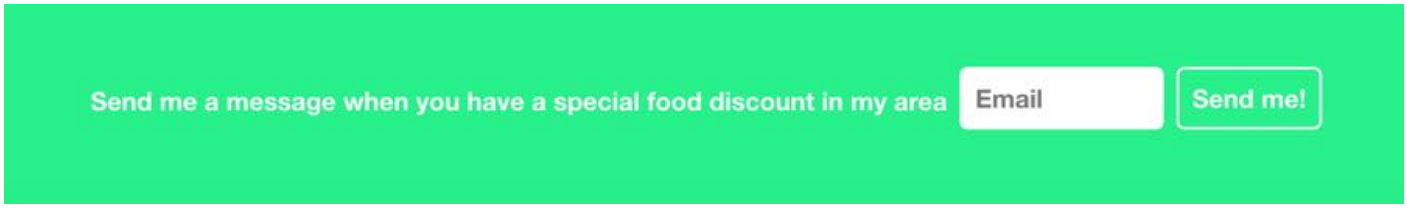
So how should it look? Adding just "subscribe to my newsletter" sucks! It doesn't show people value. The last thing I want if I go to your site is to subscribe to *another* newsletter. Instead, give me something useful and customized:

For example, if you have a jobs board and I'm browsing the PHP jobs:



That's actually useful to me. Instead of thinking about how useful getting their email address is to you, think about what would be useful to a new user so that you can capture their email address.

What if you have a food delivery service (now that's a startup idea nobody's done yet, /sarcasm!):



You can set up a free MailChimp account and start capturing addresses with their standard form. You can also add custom fields (like their city), so that you can email them later depending on their city (that's called *segmenting*).

Doing this is important as it gives you the ability to capture the audience that comes to your site on the day of your launch. And then get (some of) them to come back in a few weeks. This is so critical, because most sites see a giant drop of traffic after their launch ends. And that literally means, you've now got a app that nobody uses.

Add push notifications

This applies just as much to a native app as a web app. With a native app you have even more options though. You have push notifications which if you don't abuse them are incredibly valuable. You can literally push yourself to a user's most intimate device (their smart phone), straight on to their home screen.

Be extremely modest with push notifications though. If you overuse them, and they're not relevant, people will disable them.

To use my earlier example of the PHP jobs app, if I sign up to be notified about PHP jobs then you should send me a notification if there's a particularly good PHP job available or if there are X amount of PHP jobs available that you can show me grouped together each week. That way, you give me something useful and you're not spamming me.

Set up analytics

Something so common, but yet so many people forget, is setting up proper analytics.

The most easy and common platform is Google Analytics and it's good enough in most cases. Just embed their code. If your app is native, Google also offers a solution for that.

The most basic things that you want to see are how many people are using your app during launch (Google shows you that with their real-time tab), where they are coming from (referrers), and their flow inside your app.

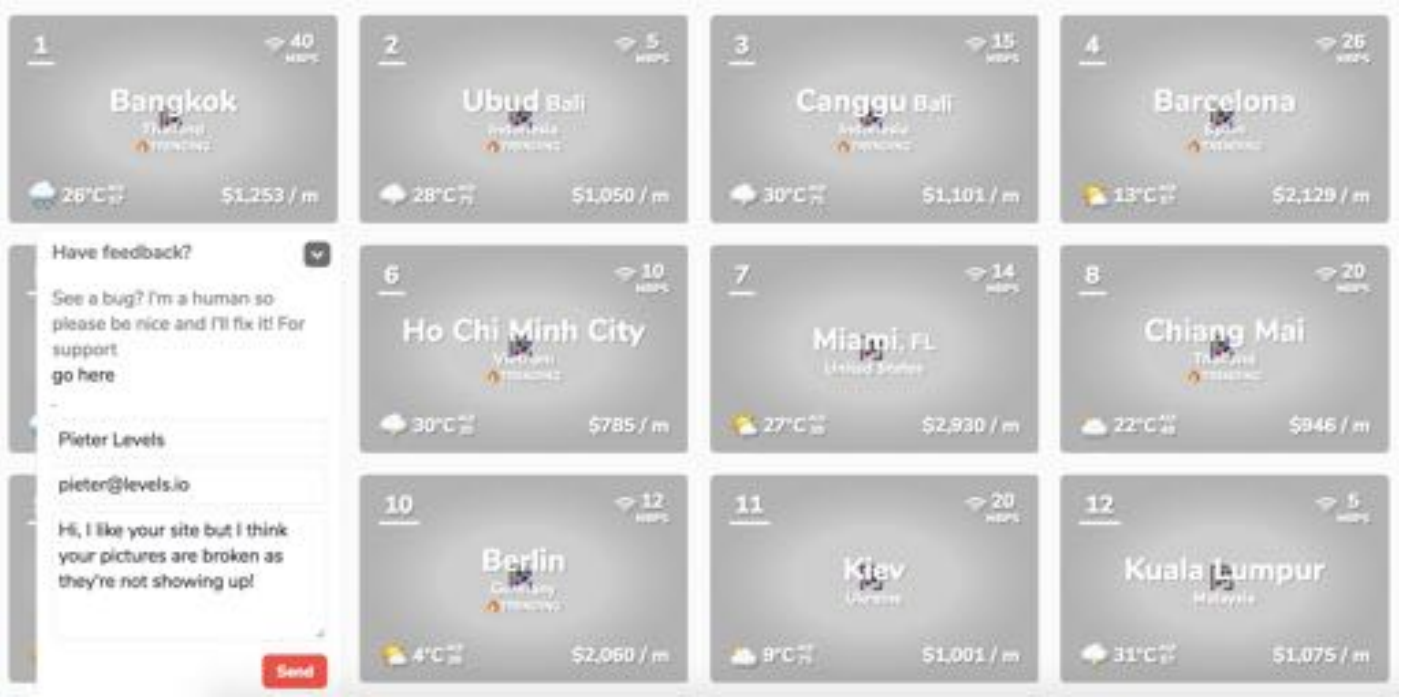
A good alternative is MixPanel, which lets you tie specific events (like when somebody taps a button) to your analytics. That means you can track more specifically what people are doing in your app. The problem is MixPanel is crazy expensive. A cheaper solution is Amplitude, which does exactly the same and in most cases is free.

All three of these work with native apps too.

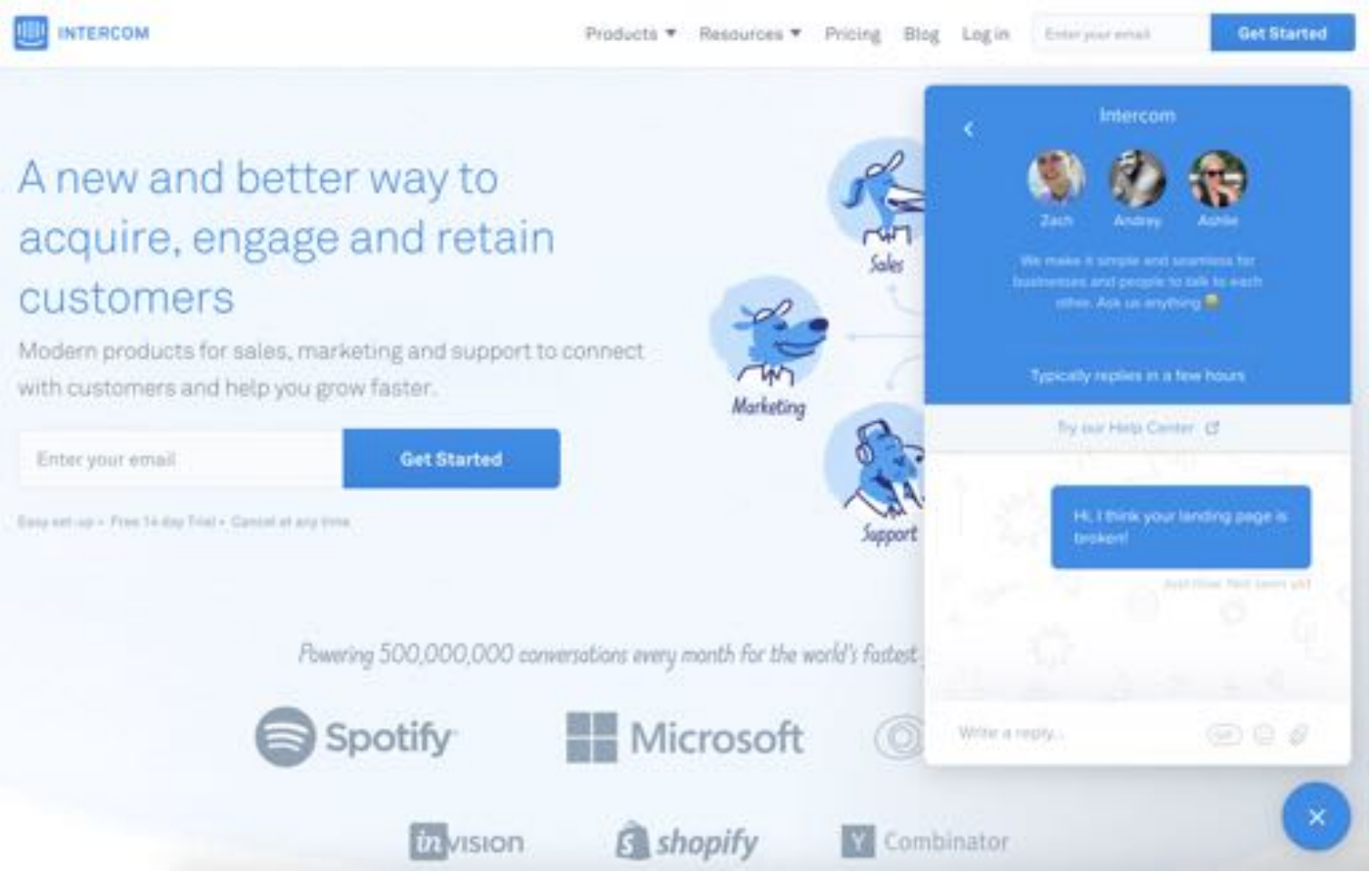
Another cool analytics app is called HotJar (https://hotjar.com). The ethics of this app might be questionable, as it lets you stream exactly what your users see. Especially in the early stages of your app, this is incredibly useful. You'll see exactly what people do, when they leave your page and sometimes even when they're confused and are randomly scrolling around your site.

Feedback box

This is super important and I rarely see people implement them. Set up a little feedback pop up in the bottom left or bottom right of your app. Especially when your app is very new, it's like a super power to your app's development to get direct user feedback.



There's many sites offering this. I used to use a free version of **Olark**, but their interface is getting extremely old. But the concept remains good.



A better option (but not free) is **Intercom**, which is quickly becoming the default for interaction with users on websites. They offer many products but one

of them adds a little chat box on the bottom right of your app, where you can chat with users directly. And if you're not there, people can leave a message.

I don't suggest being on-call at all. I'd suggest using it just so users can leave a message. The messages you'll get vary. I get a lot of very fun ones. Mostly it's people thanking for making a website and that I should add a new feature. Because they enter their email address, I am usually able to reply quickly. ANd many of these feedback messages turn into entire email threads full of feature discussions with random people from the internet! It's pretty amazing.

It's also very useful if you break something. I accidentally break my sites A LOT and many times the moment after I deploy the new version with a bug, I'll get a message within 10 minutes through my feedback box that I broke something. I wouldn't even know what to do if I didn't have that. It's become essential to my workflow even with startups I have running for years already. Things break, and you can't always test for them, so let your users be your 24/7 continuous testing army!

Where to launch?

Typical places to launch a startup

Traditionally, startups have launched in the tech subculture first. That works because tech people are early adopters and open to use new apps. They also care less about bugs than mainstream people will.

Platforms like Product Hunt, Hacker News and even Reddit are geeky but reach a huge mass of people. They're the most obvious places to launch new startups, products and apps.

Should you launch there though?

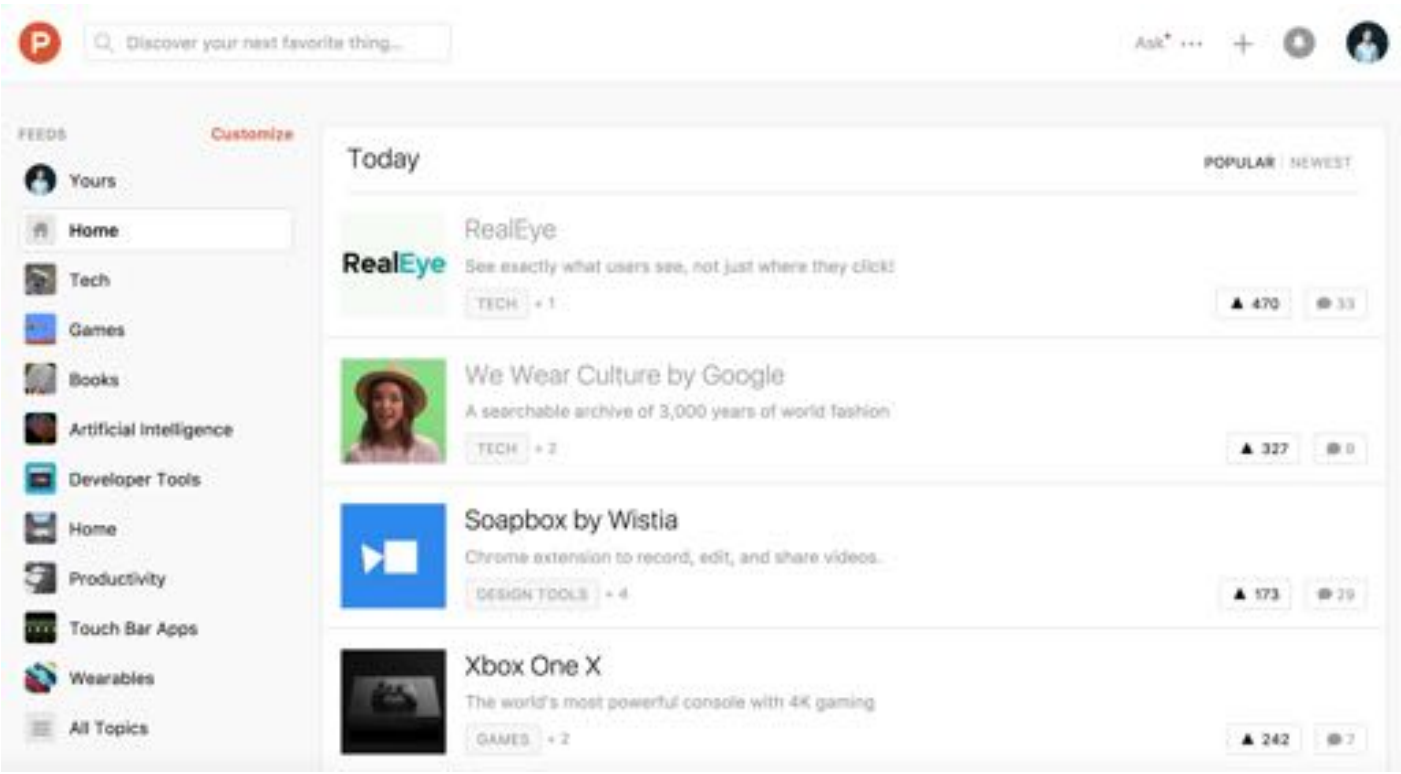
Many people say you should launch where your customers are. And that's a great point. If you make a food delivery service for pets. Are your then customers really on Hacker News? Or Product Hunt? Or Reddit? I guess, they

could be on all of those. But it'd be probably smart to find an place where pet owners hang out. That could mean advertising on Facebook targeting pet owners, or reaching out to online forums for pets. I have no idea. The point is, in that case you should target your audience specifically.

So there's a big argument to be made to not launch in the tech subculture. But then more recently this is changing though. Now that the whole word is starting to get smartphones, it seems tech has become mainstream. Everybody is obsessed with apps and startups. And in that sense, those startup platforms are actually a great place to launch. It really depends how you look at it.

Launching on Product Hunt

Product Hunt was founded in 2014 by **Ryan Hoover** as an email newsletter and has since become one of the most important platforms to launch a startup. Product Hunt is a daily leaderboard (kinda like Reddit) where people can submit and upvote new startups. In 2014, Product Hunt single-handedly instigated an entirely new cultural wave of indie makers, a wave of which I'm thankfully a part of. By letting anyone submit their app, it lets indie makers compete with giant VC- backed startups for attention.



It has a community of people that are startup fans, app makers and mostly just people curious to see the next big thing. There's a lot of early adopters that are literally on the site to discover and try new apps.

In general, being on top of Product Hunt will get you around 10,000 people visiting your site, with around 300 simultaneous users, with a percentage of those signing up. Producht Hunt traffic may convert less to paid users than normal traffic that arrives on your site by search. That's logical, because those users have a set goal. Instead, Product Hunt users are curious and mostly just visit to "take a peek".

Product Hunt's comments on products are generally very supportive and positive. But they're not radically honest (and painful) like Hacker News. You have to take them with a grain of salt.

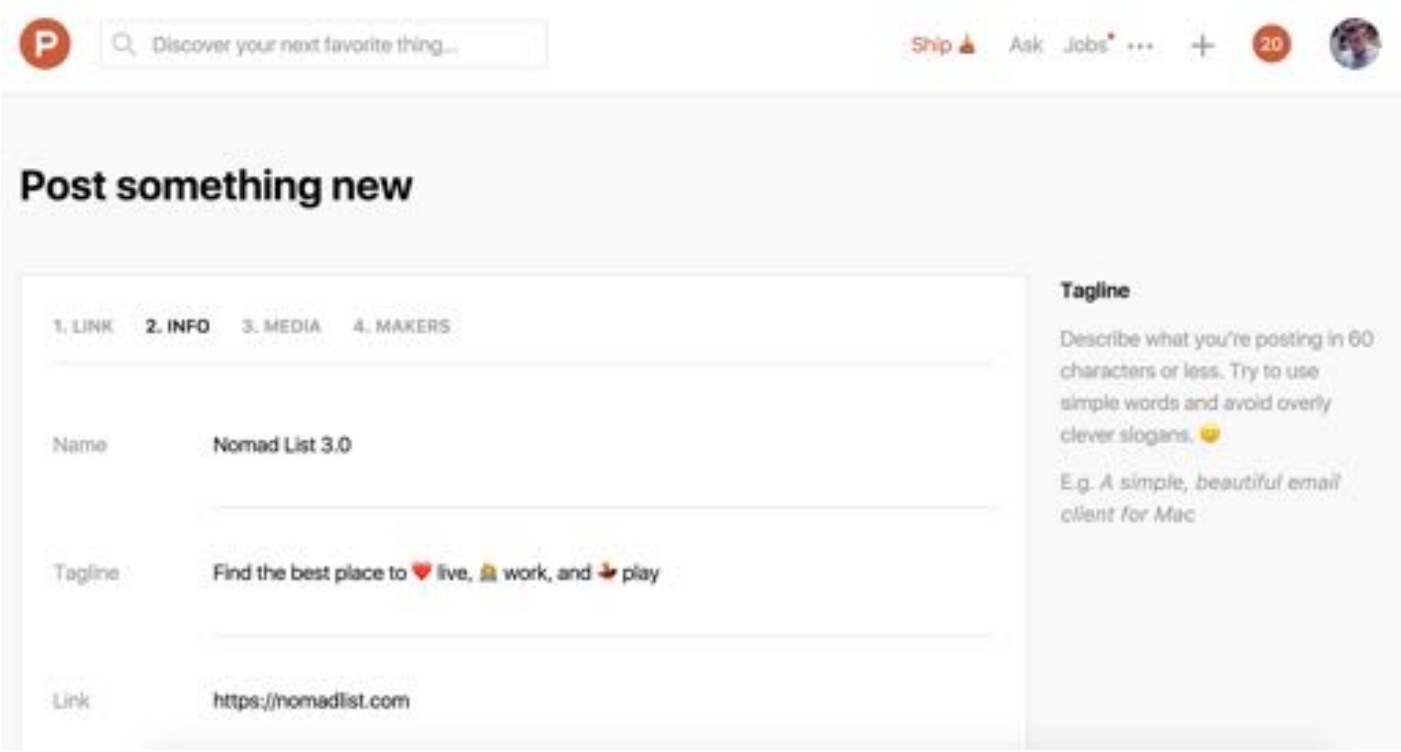
The most important selling point of Product Hunt for me is that it's packed with tech journalists trying to find their next story. Usually what happens just hours or a few days after a Product Hunt launch is that you'll see your app show up in articles in the tech press all around the world. That alone is a great thing and probably gets you another 50,000 people to visit it over the next few weeks, depending on the media outlet.

Submitting to Product Hunt

You only have one opportunity to submit to Product Hunt in a long time (you can't just keep submitting obviously). Therefore, you have to make sure you do things the right way. I've seen too many startups work for months on their product, to then absolutely fail at launching on Product Hunt by underestimating how hard it is.

The timing of a launch on Product Hunt is quite important. Product Hunt's timezone is San Francisco (PST timezone) and its ranking resets at midnight. That means you will need to launch at 00:00:01 or not much later. If you submit at 9pm PST when there's 3 hours left to vote, Product Hunt will take that into account somehow with their vote algorithm, but it still simply puts you at

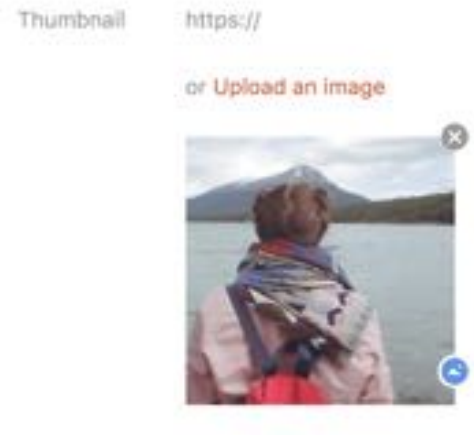
a disadvantage. Most people know by now to launch at midnight PST, so you won't be the only ones up there then. Due to timezones, I've had to stay up or wake up early several times just to submit to Product Hunt at the right time in PST.



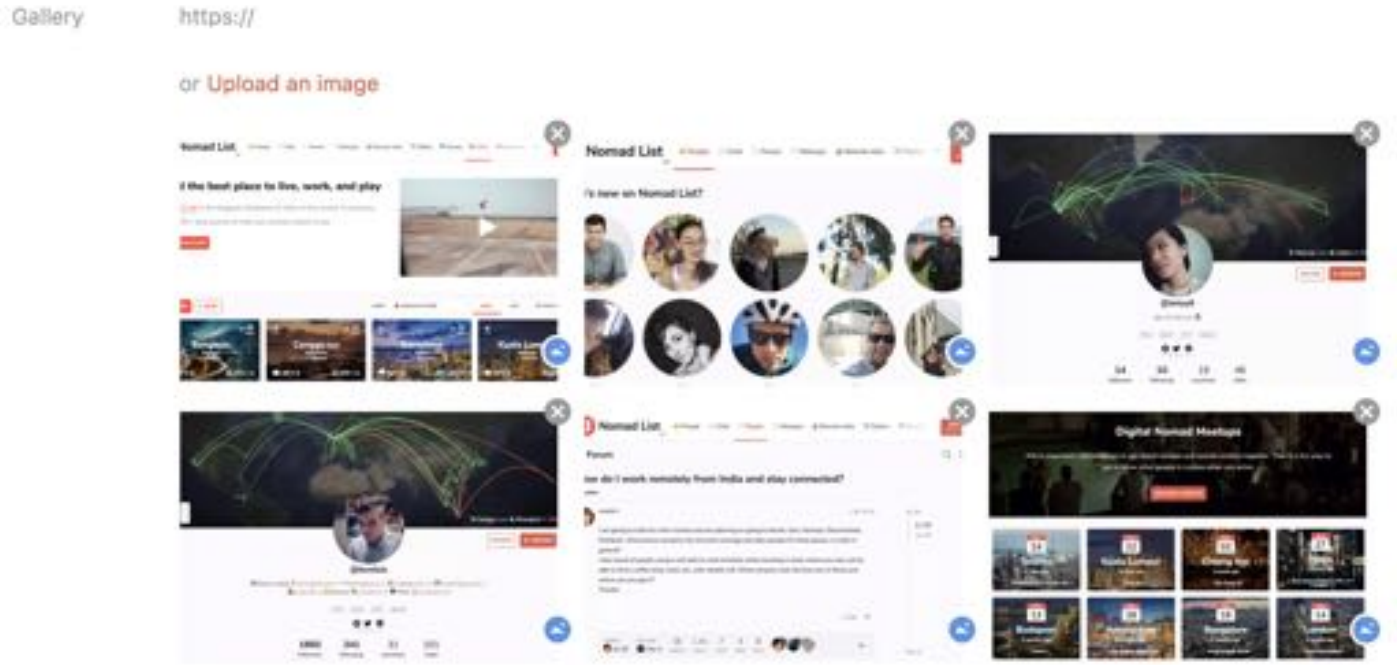
Set a proper name. If it's the first launch on Product Hunt, just use your app's name. If you already launched (like a year ago) and this is a big new version, use 2.0 or 3.0 etc. Be sure to inform Product Hunt's community managers about this, because sometimes they don't allow you to repost the same product.

The tagline is like a motto or slogan. Make sure you put some thought in to this. The most common problem with bad taglines is that they're too complex to understand for non-users of your product. What does your app do? It uses machine learning to make your photos better?

I see many people For example "An algorithmic application for machine learning applied to photos" may be accurate but is terrible for Product Hunt. Is it the first app to do this? Okay, so what about "The first machine learning photo editor". Then add some emojis: "The first 🤖machine learning 📷photo editor".



The thumbnail is as important. Product Hunt allows you to upload animated GIFs. This means you can even convert a video to a GIF (as a square box) and use that. Be sure it's only a few MB. A good app to capture videos to GIF I use is **GIPHY Capture**.



Make about 8 to 16 high-res screenshots of your app. Zoom in the browser on your website to make sure the screenshots are crisp. They're displayed scaled down on Product Hunt so it makes sense to zoom in a bit.

Show the core functionalities of your app in the screenshots. Make it eye catching.

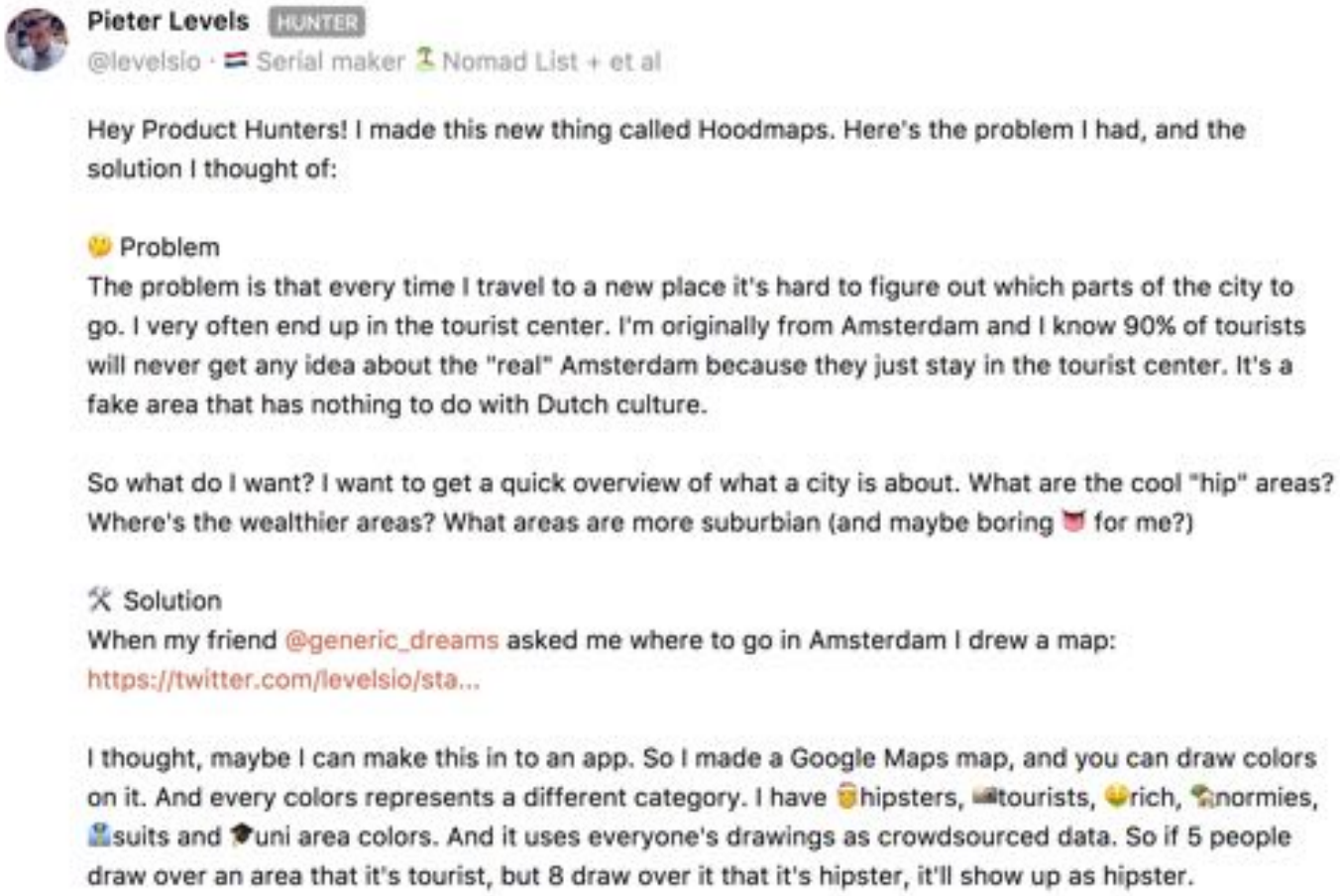
Product Hunt also allows you to add a video now. It's auto played (with muted audio) when your product page is opened on desktop. This is a great opportunity to make a short thematic promo or explainer video. Make it 30 seconds or shorter, because people don't have much time. Capture people's attention quickly and make sure the video gets across what your app is about.

When it's up there

I usually send out a tweet on Twitter, share it on Facebook and Instagram showing people that my new app is on Product Hunt. I explicitly do not ask them to upvote it. But they might do if they like it. And that helps getting up in the ranking.

If you have an email list, you can even send a message to them to tell you're on Product Hunt. Again, don't ask people to upvote. They'll do it if they like it any way.

I'll also immediately jump in to the Product Hunt comments and write an introduction post about who I am, why I made this app and what's my future plan with it:



Pieter Levels HUNTER
@levelsio · Serial maker Nomad List + et al

Hey Product Hunters! I made this new thing called Hoodmaps. Here's the problem I had, and the solution I thought of:

Problem
The problem is that every time I travel to a new place it's hard to figure out which parts of the city to go. I very often end up in the tourist center. I'm originally from Amsterdam and I know 90% of tourists will never get any idea about the "real" Amsterdam because they just stay in the tourist center. It's a fake area that has nothing to do with Dutch culture.

So what do I want? I want to get a quick overview of what a city is about. What are the cool "hip" areas? Where's the wealthier areas? What areas are more suburbian (and maybe boring for me?)

Solution
When my friend @generic_dreams asked me where to go in Amsterdam I drew a map:
https://twitter.com/levelsio/sta...

I thought, maybe I can make this in to an app. So I made a Google Maps map, and you can draw colors on it. And every colors represents a different category. I have hipsters, tourists, rich, normies, suits and uni area colors. And it uses everyone's drawings as crowdsourced data. So if 5 people draw over an area that it's tourist, but 8 draw over it that it's hipster, it'll show up as hipster.

I then stick around in the comment section to keep answering any questions people have. And see if people have feature suggestions.

It's important to be polite and humble when you're in the comments section. I see a lot of people pose, brag and market themselves when they're up there. And people don't like it. It looks sleazy. You're there to take people in as guests into your app and treat them well and hear their feedback and improve based on that. Not to sell!

Different platforms have different degrees of hate. Product Hunt is famous as one of the more positive platforms to launch apps on as it's fiercely curated. That might mean you'll get nicer people but less real feedback than e.g. Hacker News.

Launching on Hacker News

Hacker News is probably the hard "core" of the tech and startup scene.

Just like Product Hunt, it's a leaderboard. But it doesn't just contain startups. It also has blog posts about new technology, startups, funding rounds and venture capital.

Hacker News is open to anyone's submissions, but that makes it extremely competitive though. It also has lots of protections against spam, marketing, voting rings and even controversy. For example, if a post receives more comments than upvotes, it will quickly be flagged (automatically) and dissappear from the frontpage, that's the controversy filter at work. If Hacker News detects you're forwarding your post to your friends to upvote, it'll discount its votes and it'll just drop it to a few pages further. ATherefore, it takes quite some work and luck to get on Hacker News' frontpage.

Where Getting on Hacker News is already hard, staying there is a war in itself. Hacker News has agressive moderating, which means if a mod doesn't like your post, it'll simply discount the upvotes too and it'll fall off the frontpage. The whole goal of HN is to keep the content very specifically relevant to its audience and the current zeitgeist. You need to consider that. Many makers make things specifically for the Hacker News zeitgeist. For example, now that I'm writing this, there's a lot of pushback about Facebook's ruining of society and addicting its users to newsfeeds. Making an app around this that solves this will probably get high on Hacker News (if someone else didn't already)≥

As with Product Hunt, the title is very important here. Let's say we're making that food delivery app for pets. We call it Petsy.com. We submit it to HN like this:

> Petsy.com - The best food delivery for pets

It might work. But I think it probably won't.

A better way is to make it personal like:

> I made a site that lets you subscribe to food delivery for your pet

It's less marketing-y, and HN people will like it better.

An even better option is to use Hacker News's show-and-tell tab called "Show HN", like this:

> Show HN: I made a site that lets you subscribe to pet food delivery

This might have a good chance to stay on top. Because it's real, friendly, and authentic.

Now that you have submitted it, you need to give it a slight boost. This should be very slight though, because if you get too many people to upvote it too fast, it will be quickly be flagged as spam.

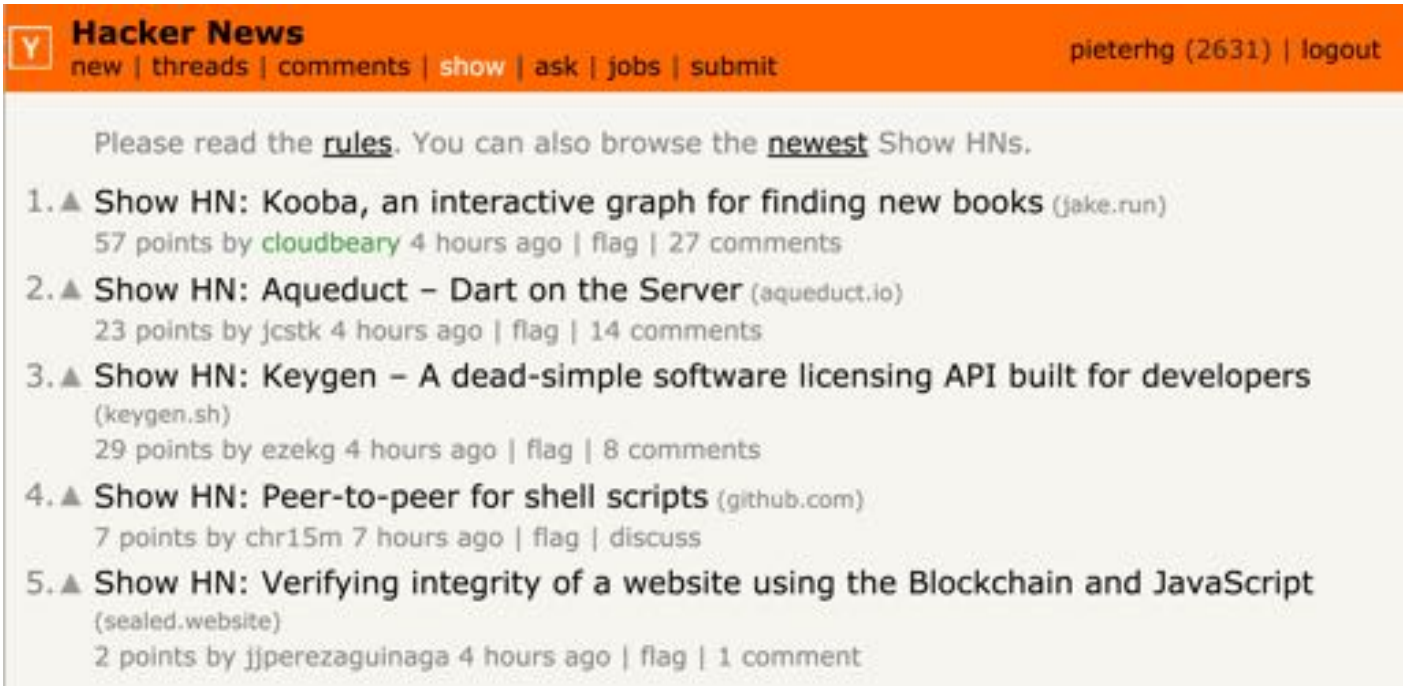
Your submission will be on the **new** page. It will be on there for about an hour. Getting 5 people spread over the first hour to upvote it, will usually already get it to the front page. From there, stay off it and just let the organic reach of being on the front page do its work.

Now if it doesn't go up more, it means Hacker News is probably just not interested in it. Right now. Luckily, Hacker News recently allows re-submissions. That means you can always try it again in a week. Try a different day. Or a different time. With a different title. Maybe a different landing page. But usually if it doesn't work the first time, it's simply not for Hacker News.

Be mindful that Hacker News is notoriously hateful. They can destroy your entire company verbally in the comment section. But they're also one of the most honest places on the internet. They regularly predict the downfall of huge companies years in advance because they see through the PR bullshit (like **they did predicting the fall of Groupon**). That means they'll see through your bullshit too. And many times they're right.

And many times they're wrong too. When **Dropbox** was posted by Drew Houston for the first time, it wasn't taken seriously by most of HN. It turned out to become a billion-dollar company.

Generally, Hacker News can get you around 50,000 to 100,000 visitors, and 1,000 simultaneous real-time visitors. That's about 5x to 10x as big as Product Hunt, so make sure your server can handle it.



Launching on Reddit

Reddit is cool because it's geeky but it has a big overlap with the mainstream.

Reddit is more mainstream. It's also a hive mind. That means if a few people hate it, everyone quickly jumps on and follows. But if some love it, it'll be the best site in the world for a day. According to them.

Nomad List was on **the first page of Reddit** (!) but was destroyed in the comments, as people doubted the validity of its data. This had to do with a subjective interpretation of what cost of living is and as it varied so much per person, it wasn't right for anybody. Still it went to the frontpage!

Hoodmaps also **got to Reddit's frontpage**, and people where a lot more positive. There was some hate about the socially sensitive nature of the site. But I resolved that by banning any sensitive words from appearing on the site.

Most sites can't take Reddit's traffic

The secret of getting on the frontpage of Reddit is to make sure your server can handle Reddit's insane traffic.

There's a reason most of Reddit's frontpage is image memes from **IMGUR**. Because most servers crash before they even reach the frontpage. If you're on the front page, you get from 50,000 to 500,000 visitors and 5,000 to 25,000 simultaneous users. That's gigantic amounts of traffic! To compare: **Nomad List** might get 50 simultaneous users at any point of they day.

My server crashed when it was on page 2, then I asked my friend @aikedejongste to jump on Skype and help me in the middle of the night. He logged into my server and together we optimized it to handle the traffic. TL;DR we made all dynamic (PHP-run) pages into static HTML files and as it was running on NGINX server, it went fine. With 5,000 people on there, it was usable. A small miracle. You can always make your site dynamic again when the traffic goes down. But static is definitely the way to prepare for this.

A quick and dirty way to make your dynamic pages static is to make a scheduled server job that does something like this:

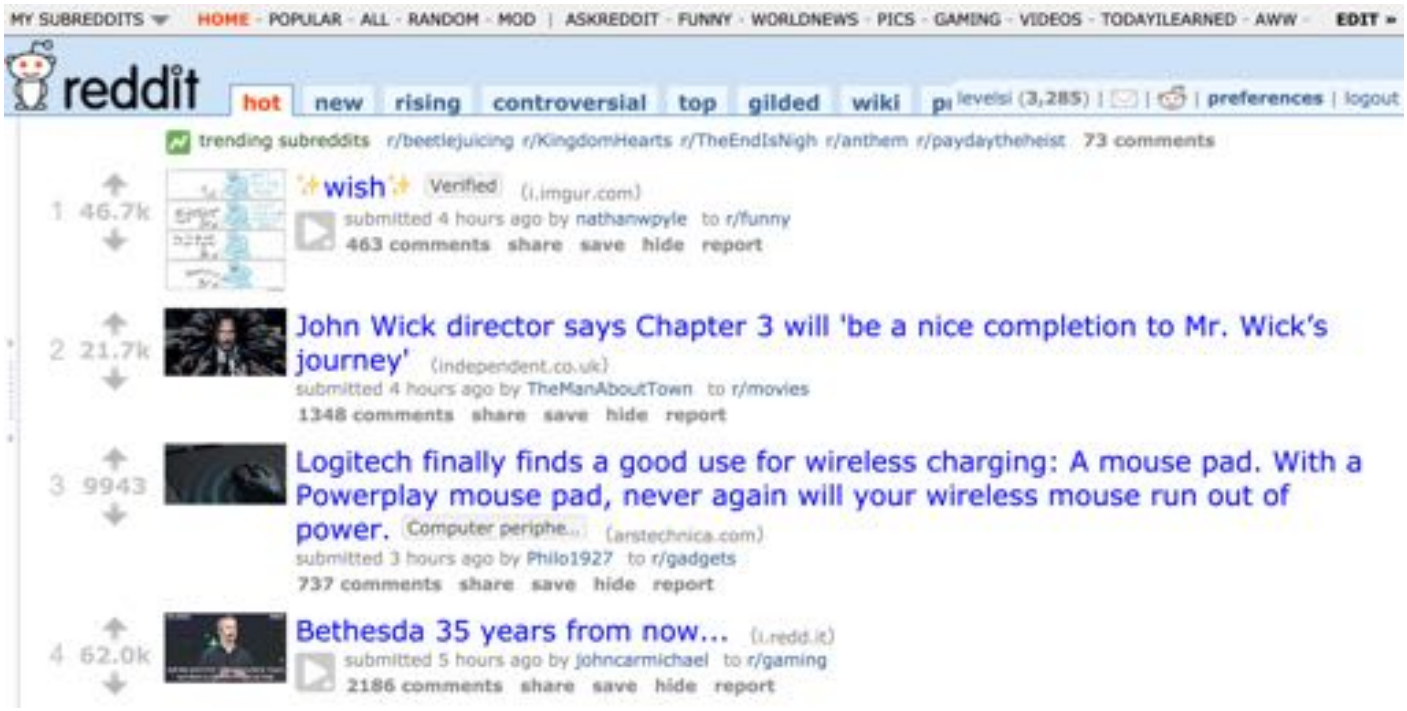
```
php index.php > index.html
```

If you're using PHP, this works. But you can probably do it with other languages too. Just make sure the index.html shows up on your site if people load

it and not the PHP file. It dramatically reduces load as now it's not a program operation but just a file read operation for your server!

Now you're saying, what if my app is dynamic (which it usually is). Well, in case of Nomad List, the filtering of cities is dynamic. But that's just one AJAX call to a very light API server script. You need way less power to let your server respond to API calls thaenthan generate an entire webpage. So you can keep that API running, just make the main page static.

You need to remember, most people bounce from your site anyway if you're on Reddit. About 20% will actually use it. That's why reducing the load of the initial page is important.



Pick the right subreddit

When submitting to Reddit, you need to pick a subreddit.

It's probably best to first submit it a subreddit that is particularly relevant the product you're launching. In my case, I'm building a food delivery app for pets, so I could go with:

> reddit.com/r/pets

Again, the title is very important:

> Hi /r/pets! I made a site that lets you subscribe to food delivery for your pet

This is a good way to get quick traction and see what people in your specific niche (pets) think about it.

A more general one to get quick feedback about your startup is:

> reddit.com/r/startups

How to get on the frontpage

Now here's what you've all been waiting for. How do you get on the frontpage?

Reddit's frontpage used to be based on a selection of subreddits that change regularly. But since 2016, all non-adult subreddits are included. Still, this means if your submission goes to the top of a high traffic subreddit, you have a good chance it will be in the top 100 posts of the entirety of Reddit too.

If you reach the top 100 of posts, you'll be somewhere on page 4 of the front-page. Here's where the real battle starts. You now have to climb from page 4 to page 1. Again, most sites fail here because they start going down because of the traffic already coming in now being too much. If you don't go down, it's now really about the collective "hivemind" of Reddit and if they like it or not. The scale of Reddit is so big if you're in the top 100, that it means you can't really affect it any longer from here. If it's great, it'll slowly get pushed to the front page. This might take a few hours, and more probable actually is that it doesn't happen.

A very common subreddit that people who make little functional apps/sites use is:

> reddit.com/r/internetisbeautiful

In my case, as **Nomad List** and **Hoodmaps** were a bit more data-based, I submitted both to:

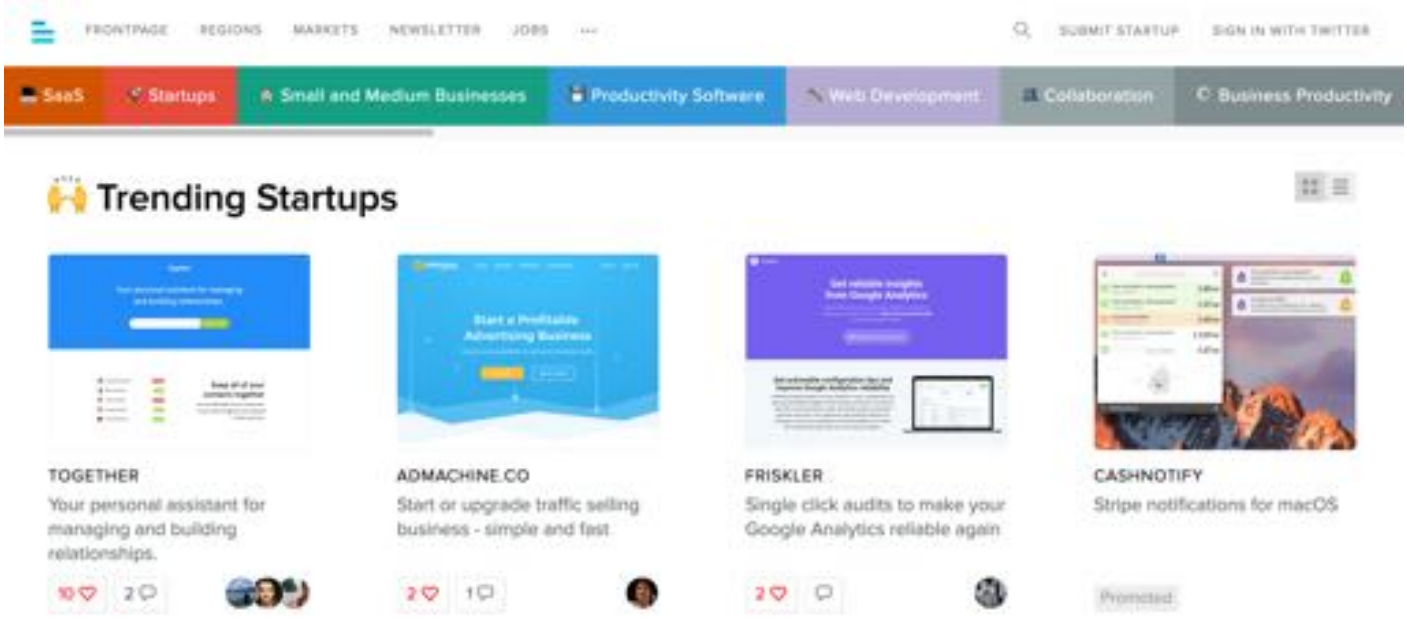
> reddit.com/r/dataisbeautiful

In terms of title, pick a title that's more general because you want to consider that it'll be seen on the frontpage:

> I made a site that lets you subscribe to food delivery for your pet

Submit it and see what happens! And as always, jump in the comments and participate!

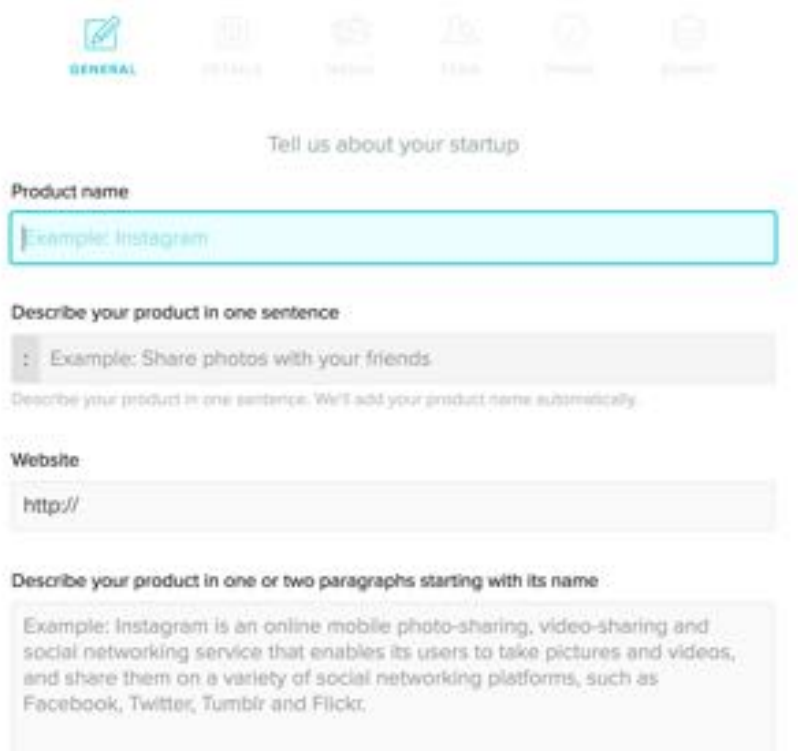
Launching on Beta List



BetaList is the internet's original startup directory. Launched in **2010**, by **Marc Köhlbrugge** to TechCrunch, it has since helped launch thousands of startups, including being the first launch platform for apps like Pinterest.

> "BetaList is the place for early adopters to discover upcoming and recently launched internet startups"

BetaList has a strict curation of what they accept, and unless you pay ~$129, you're put on a waiting list for launch which takes 2 months. Therefore, you probably have to pay to go live immediaetly. Unlike the name BetaList suggest, startups don't have to be in beta anymore to featured. They do have to be new though.



Generally, BetaList can get you around 500 - 1000 visitors with a percentage of those signing up. It's a very targeted tech audience though, and helpful to get the early users in before you launch on bigger platforms.

There will be hate

Your baby that you worked weeks or months on is now out in the world. And the world can be a pretty dark place. People will judge your app fiercely. If something doesn't work, well it's bad. But if two things don't work, they'll just say the entire app sucks. That's how humans work. Think about yourself, you

do the same clicking on random links on Reddit. You get angry if it doesn't work. And you dismiss it.

So hate is expected. Because there will always be something people disagree with. It doesn't mean that's bad news. One thing you shouldn't do is start fighting with strangers on the internet. That's the fastest way to failure. Instead, try to read through the hate and see actual feedback that you can improve your app with. And if the launch is ongoing improve it right away (if possible). That doesn't mean build entire features *while* you're launching, but make sure to fix those little bugs for the new people that are coming onto your site. If you have a native app, that's a hard thing to do though as you'll need to re-deploy to app stores. In that case, tell people personally that you're deploying a fix now and they'll be able to update the app later!

Why be nice to them, you ask? Because people like to hate, but if you then listen to what they say and act on it, they'll come back with love. "Oh wait, there's a real person behind this? With feelings? Oh wow!". And they'll treat you better.

Things not to do

Asking people to share/like/post your product

There's a difference between asking a few friends to give you a head start by upvoting something. A few! But it gets weird when you start asking everyone publicly to upvote, share and like your stuff.

I know people who keep asking me to tweet their stuff, and I can tell you I never did, I never will and I don't consider them my friends anymore. Because asking for this is fundamentally beyond the point. The idea is that great things rise up by being rewarded for being great. You can't force traction. Well, you can. For a little while. But all those people you forced to share your product won't be there next time, cause they don't like you anymore!

Buying fake upvotes, likes, followers

The same applies to buying fake virality. It won't work. It looks good on paper but it's not actual humans. And people are less and less impressed with seeing big numbers on somebody's social media account. More important is, again, what your app does.

So why doesn't this work

Well, it actually does work. For a while. You can create artificial traction. The problem is that it doesn't stick You'll have to keep adding more fake virality and the odds of it being picked up by actual REAL humans is quite small.

Because if you're app was good from the start, it wouldn't NEED any artificial following. It could push itself just by being great. Because if people like something, they'll share it themselves, they'll make it big. Especially if it's something unique and original that does something better than it has been done before.

Because we're sick and tired of fake stuff. This goes much deeper than just apps and sites. The whole Zeitgeist is now about people looking for authenticity, because everything around us is fake. We like people that are open, honest and real. Why? We are bombarded by bullshit marketing messages. Why do you think we're all installing ad blockers? We want purity. We want stuff to be authentic and organic! So what's the best way to get those people to follow you? Be real!

Be organic

So the answer is, be organic! Be real. Create traction by making a great product that is considerably better than the competition, easier to use and more original.

The big advantage by launching organically is that you will see if your app doesn't work. If it doesn't get traction, there's a good chance it's simply not good enough. The idea might suck. The interface might be bad. Maybe the idea isn't original enough. You'd never be able to find that out by faking it!

Telling your story

Blog

People might call this "content marketing", but I feel this is a sleazy term. It means that you're writing something with the goal of selling something. I don't like that. It should be real.

I started blogging years ago and pretty much the preface of **Nomad List** was me writing about traveling for 1.5 years. And randomly, because of that I had a small audience of people that knew me, and that just grew a lot and helped Nomad List's launch a lot.

Blogging in that way gives you the opportunity to build up an audience even before you launch.

I have a hunch though that text is becoming less important (lol writing this in a book). But I think it's true. Look at the rise of Instagram and Snapchat vs. Twitter and Facebook. People are getting too lazy to read lots of text. Probably only a few people will actually read this entire book. I wonder how many actually will read this sentence!

So there's a case to be made that telling your story should happen in a different, or at least, in more media than just text alone.

Press

Why press matters

The importance of press, is that it gets you lots of mainstream users outside of your niche bubble. Especially non-techy people that are hard to reach through platforms like Product Hunt.

Why press increasingly matters less

Just like I wrote before about blogging,. I think people read press less and less. Again that's a hunch but I think I'm right. The rise of Product Hunt is an exam-

ple of a platform that pretty much took a big bite out of the tech press pie by automating it.

Then there's the issues with the deep mistrust people have of press these days. Readers aren't sure if what they're reading is a paid advertorial or actually a real article about a company. Startup founders are scared to talk to journalists in fear of becoming part of clickbait hit pieces killing their careers.

You might need press to kickstart traction, but you definitely need it less and less when you have traction already. Especially in these times.

If you do use the press, make sure you control the conversation. You set the angle of the story. Check what type of articles journalists write before you talk to them. Fact check the articles they post about you or your product after and ask for corrections if necessary. Press can easily flip a story to become negative about you and your product in favor of page views. The press is not necessarily your friend, as much as they like to make it sound like it when they talk to you.

How to get press

Make a press list

The reason most people don't get press is because they simply contact the main email address of a press outlet like *tips@thenextweb.com* with their app's pitch and URL. Here's a little secret. Those email addresses are the industry's black holes. A lot of stuff goes in them. Not much comes out.

The way to get a journalist to write about you is to make it personal. How do you do that? Well. First figure out which journalist would actually be interested in your app and find it useful. Every journalist has their own style, taste and personality. And niche.

There's (tech) journalists who mostly write about cryptocurrencies, or only about travel apps, or only about bootstrapped companies. Find journalists that are relevant for your app by searching who writes about your competitors or apps in your industry. Just doing that, already gets you further than 99% of

people who don't take the time. Shooting 1 targeted email can result in more success than 1,000 untargeted emails to every single journalist out there!

If you know a journalist that already wrote about a previous app you made, that's a warm connection and it's *very* valuable. Keep them in the loop about your projects but don't bug them. They'll be happy to write about you again if they like what you're doing. Update them about significant stuff, and give them scoops, that means exclusive news you don't give anybody else. That's currency for journalists.

Based on all of this make a press list of journalist you want to approach. A fair estimate is that less than 5% will write about you. Even if you target it well. So to get 5 press features, you need to contact 100 journalists. That might even be a positive estimate! It might take hundreds of targeted (!) emails.

Submit.co
Where to get press coverage for your startup

Add a publication | Get Updates | Powered by BetaList

URL	Submit page	Twitter Username	Followers	Alexa Rank	Categories	Regions	Platforms
betalist.com	Submit	@BetaList	28,140	23,274	gaming, B2B, B2C, kids		iphone, ipad, android, mobile, web
producthunt.com		@producthunt	76,448	3,563			
news.ycombinator.com	Submit	@newsycombinator	101,958	1,644	B2B, B2C		iphone, ipad, android, mobile, web
thenextweb.com		@TheNextWeb	1,540,905	1,260			
blogs.wsj.com/tech-europe/		@wsj	6,400,366	309	B2B	Europe	
huffingtonpost.com		@HuffingtonPost	5,747,681	107			
mashable.com		@mashable	5,324,187	267			
techcrunch.com		@techcrunch	5,150,408	412			
wired.com		@wired	4,588,757	553			
lifehacker.com		@lifehacker	2,522,482	360			
readwrite.com		@RWW	1,626,408	8,409			

How do you find these journalists? Luckily here's a few directories to find contacts for journalists. One website to find tech journalists in specific is **Submit.co** lets you find a major amount of press outlets. From there you can go on to find specific journalists.

Put every journalist in a spreadsheet and personally contact them. Don't bug them! In the list, write down how far you are with them. For example,

"emailed?", then "replied?", then "will feature?" and keep track. Don't underestimate this. This takes work especially if you're new, nobody knows you and you don't know anybody either.

How to approach journalists

Journalists generally hate if you approach them through Twitter, Facebook or LinkedIn. They still use and read email, just not in the way you do. They skim through it very fast. The best way to get them to stick is to keep your approach concise and powerful.

I see so many people write lengthy emails about why their app is the best thing ever, but hardly anybody is able to argue why it's great. What's the value for people? And they're all lengthy. Journalists don't have time for that stuff!

Here’s an example of a bad email that is very typical:

> Subject: New startup Petsy.com
>
> Hi TechCrunch!
>
> We just launched our new startup Petsy. It's funded with a Series A round led by Lion Capital.
>
> We're growing like crazy, crushing it right now and we'd love to tell you more about our launch this week.
>
> Petsy connects pet owners and pet food. It works like this, you go to the app, then you select which food you want. Then you tap order. It then delivers the food to your house. We have about 5,000 people in our app and growing 40%. Thanks for covering us!
>
> -Rick CEO, Founder Petsy.com
>
> *This e-mail is intended for the addressee shown. It contains information that is confidential and protected from disclosure. Any review, dissemination or use of this transmission or its contents by persons or unauthorized employees of the intended organisations is strictly prohibited. The contents of this email do not necessarily represent the views or policies of Rick Andrews.*

Why is it bad?

- It's way, way, way too long, ain't nobody got time for this
- It gets to the point way too late
- Subject is vague
- It's not personally written towards the journalist

- It’s full of vague bragging instead of facts (e.g. “we’re growing like crazy”)
- It uses buzzwords like “crushing it” and jargon like “connecting pet owners”
- It’s full of ego and business lingo (like CEO, Founder, of what?)
- There is no clickable link to your startup
- It has a ridiculous disclaimer
- If you look at it from a distance, it’s just a block of text

This is not good. But this is what journalist get by default.

So how *should you* approach a journalist?

Most importantly: be concise. Keep your email to one or two sentences max. Here's a good pitch to Susan from TechCrunch, who you know is interested in food or pet industry startups:

> Subject: Food delivery startup for pets
>
> Hi Jody! I made a site that lets you subscribe to food delivery for your pet. Let me know if you need more info :)
>
> **https://petsy.com**

It's friendly, fast and clear. The journalist has the opportunity to check it out themselves without you selling it to them. And yes, if they like it, they'll reply and ask you for more info! That’s when you can give all your data and info.

Be sure to reply to them ASAP as they're impatient. They might be waiting to write an article on you and if you reply immediately, they can get on with it. If

you don’t they might scrap it and write about something else. They might reply:

> Nice! Do you have funding? Revenue? What’s the market size? Any screenshots I can use for the article?

Tell them quickly and concisely again and send them what they need immediately. Make it as easy as possible for them to write about it. Send them screenshots and press material that is already cropped perfectly, so their job is easy!

Make a journalist’s job easy and they’ll be happy to write about you!

And be nice!

Even better is to add a personal touch, if you know something personal about journalists that seems relevant to your app, tell them. If they have a puppy called Robert DeNiro (and talked publicly about it), you could ask:

> Subject: Food delivery startup for your puppy Robert de Niro
>
> Hi Jody! I made a site that lets you subscribe to food delivery for your pet. Maybe it'll save you some time feeding Robert DeNiro.
>
> **https://petsy.com**

Gaining controversy

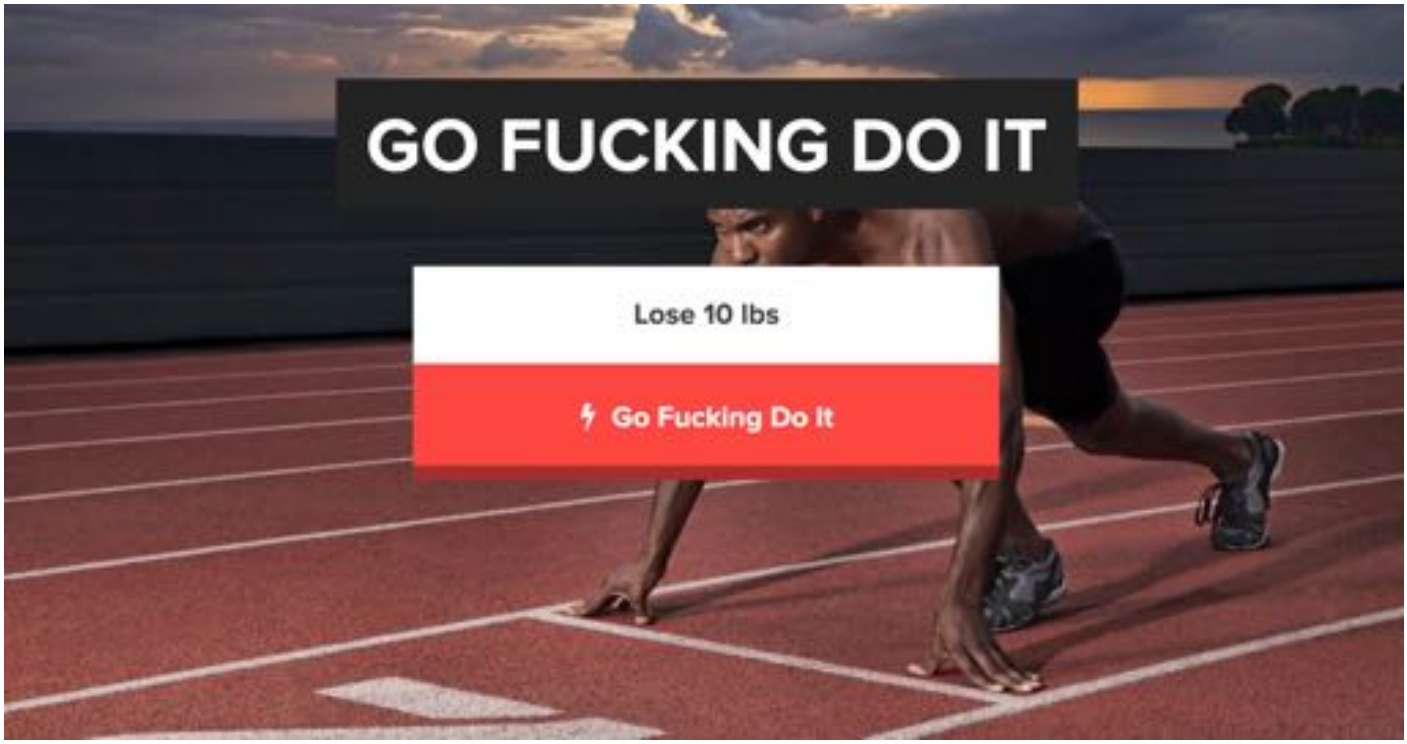
There's very few times where I'd say Trump is a good example to follow. But in case of marketing, he might give you some hints. Trump became president through controversy. And without making this book political, it's impossible to deny that didn't work. He became president.

The problem is that most people are not controversial. Because we're nice people. We don't like to shock. We're not raised like that. But if you want to get attention, a little bit of controversy will get you much further than the rest. And that's most certainly essential to getting press.

The "news" is literally called "news" because it writes about things that are "new". If you just build an adaptation of the same-old thing and tell the story about it in the same way as people did before. Well, they're not going to write about it! So it needs to be fresh. Either your product needs to be radically different or better (hint: most people think their product is, but it isn't), or how you tell the story about it needs to be different. Controversy is by definition different than the "norm".

I use controversy all the time. I wanted to make an app to get people to complete their goals. But when they don't make their goal by their deadline, it charges their credit card. First I thought, I should donate it to a charity. But then I thought, fuck it, let's keep it to myself! Then I had to find a name: I wanted to call it "just do it". But Nike owns that trademark. So I thought, "go do it". Okay, that's nice but a bit plain. What about "go fucking do it".

I asked friends, and they all said that I should remove the "fucking" part. And I agree, in most cases it's not even that it's offensive, it just looks cheap and cheesey. But I did, and I registered **GoFuckingDoIt.com** and launched it. It went viral all over the press and struck a right nerve. The two reasons were the name and the fact that I was very brutally honest that I would get the money if you failed your goal. Every single time I presented the idea and said that, the audience started laughing. So there's something there. It's cheeky, naughty, controversial.



The way I market this book is a bit controversial. I will diametrically oppose the old way of doing startups, namely raising funding through venture capitalists, attack it, and provide a solution: namely, do-it-yourself, ship lean. I make my slightly controversial message very simplified: "VC is bad, indie is good". People don't have a lot of time, so you need to get your point across. Even though "VC is bad, indie is good" is simplified, it *does* align with my core values. Sure enough, not *all* VCs are bad, and there's plenty of indie businesses that are also bad, but generally I agree with the statement. And it sells because it's simple and controversial.

Which press outlets

Tech and mainstream press is merging and has lots of overlap. Here's the biggest outlets right now in both. This list will probably be outdated quickly if this book is getting older, so take it with a grain of salt.

A quick way to find which press outlets to target right now is to check where your favorite apps or your competitors have gotten coverage. Also remember to find press outside of tech and mainstream, but in the niche of your product. If you're building a software-as-a-service app for hairdressers, be sure to reach out to the main industry magazine/website for hairdressers in America!

Tech press

The main tech/business news outlets right now are:

- **The Next Web**
- **Forbes**
- **Lifehacker**
- **Quartz**
- **Mashable**
- **Fast Company**
- **Entrepreneur**
- **Business Insider**
- **WIRED**
- **Inc.**
- **Tech.co**

Mainstream press

The most important mainstream news outlets right now are:

- **HuffPost**
- **The New York Times**
- **Wall Street Journal**
- **The Guardian**
- **Washington Post**

- CNBC
- CNN
- The Telegraph
- Observer
- VICE
- Mirror
- Slate

Don't stick to one launch, keep launching

Make every feature a launch opportunity

A more recent strategy by many startups is to make every feature launch a special promotional opportunity. By adding new features and broadcasting them to press you can keep getting attention. Even if you've already launched ages ago, people like seeing improvements. And they'll write articles about it, resulting in traffic coming back to your site.

Side project marketing

Building up from that, many startups are now simply launching lots of mini apps with distinct functionality that before might have been a feature of their existing app. But by decoupling it from their main app, they can use it as an entire new launch. And that gives them the full benefit of a new product launch.

This is also called "side project marketing". On my self-confident days, I think I had something to do with it when I tried to launch **12 startups in 12 months**. It got people the idea that you don't need to ship perfect apps, you just need to ship a lot, and you'll get way more press than anyone else.

A famous example of this is Buffer. They launched Pablo, an online image editor to put text on photos for sharing on social media. More recently they launched Respond, a messaging solution for doing customer support on social media. You see the red line here. Their industry is social media, and they make lots of little apps now for it. They could have launched these as features of Buffer, but they specifically choose to not. This way they leverage getting press for each new product. And if they want, they can either spin off the product from their company entirely, or integrate it into their core product (like Buffer's main app).

Another famous example is **Crew**. They launched their side project **Unsplash**, a free stock photo site, which became even bigger than their main business-busines. Then they did it again by launching **HowMuchToMakeAnApp.com**, a calculator to see how much it costs to build an app. They got a crazy amount of press just by **doing these two mini apps**.

How to stay motivated working on one product

Keeping up the long-term motivation for a product is hard. And I think some people have difficulty keeping up motivation for an idea because they're not getting feedback from the market. You shouldn't work on an idea that isn't taking off. You can't just for months and years upon years work on something that doesn't give any feedback. It is important that you at least get people excited about your app or your idea and people using it. If nobody's using it, then you're going to lose motivation. For me, the motivation is intrinsic if people use it or if I make money with it. Then I know it works and I want to keep working on it.

If it doesn't motivate you, sell it or kill it

If your idea is actually successful and it becomes a real product, it becomes a company that actually makes money, then it's very normal to become bored with it. Maybe your long-term vision is not there anymore. You don't think it's going to get any bigger. In that case, sell it. Sell it to somebody or hire people

to run it for you or automate most of it so you don't have to focus on it anymore. Then you can focus on an idea that does motivate you because it's really important I think. For me, work has to be motivating for me. If I don't believe in what I'm making anymore, then I don't want to really do it anymore. That's when I lose the motivation. It's very important to have motivation for your idea. Make sure you just kill your product if it doesn't motivate you and if it doesn't get feedback from people.

How many ideas should I work on at a time?

I think my strategy is that I work on a lot of stuff at the same time because I want to not have all my eggs in one baskets. I think there's a higher chance of success if you try different stuff. Right now, I have two main projects. I have a third side project I work on. Then there's two others that are kind of small that might go somewhere in the next few months or few years. The most important ones obviously get most of the focus. They get like 80% of the focus. I also get pretty bored quickly, so switching projects works for me.

It depends on yourself though. How much time do you have to work on ideas? If you do have all the time in the world, then just work on different stuff. If you can't work on a few different ideas because you think it takes too much time, you're probably spending too much time per idea, too much time on executing something. You should make things more minimal.

Conclusion

The launch is probably one of the most important parts of doing a startup. Everyone underestimates its importance and difficulty. It can make an average product or break a great product.

You should launch early so you get feedback from users quickly. Launch on the typical tech sites like Product Hunt, Hacker News, then go for the mainstream on Reddit and go for your product's niche on specific niche platforms. Don't spam your launch, but do let people know you've launched, like your followers

and press outlets. Most importantly, a launch is not finite. If you'll have a product running, you'll keep launching repeatedly every few months or years when you have specific big new feature developments.

Nowadays, the launch is perpetual.

Resources mentioned

- MailChimp
- Google Analytics
- MixPanel
- Amplitude
- Hotjar
- Product Hunt
- Hacker News
- Reddit

Your homework

- Make a list of places you will launch your first product, these can be the typical ones like Product Hunt, Hacker News and Reddit, but make sure there's more niche platforms on there too.
- Write a title and description for each platform you will launch too. Personalize it for each platform depending on its audience. Make sure it doesn't look spammy.
- Do a final check that your product actually works. Can a new user immediately see what it is about and start using it? Go through the process multiple times to make sure every little bug is gone.
- Pick a day and time and launch!

Grow

Introduction

So you've launched your product. You've been on all the big sites, you got major press to write about you. But it's now been 2 weeks since launch day and you're (naturally) seeing traffic and usage drop off. What's going on?

Well, it's one thing to launch a product successfully. It's a whole different challenge to keep that momentum going and make your product grow in the future. It's somewhat of a mystical art. Everybody wants to succeed at it, and only some do. And those 'some' are only able to do itsometimes. If anybody knew exactly how to do it repeatedly, they'd be very rich.

I've only been able to do it a few times. So I'll tell you what I feel are important things to try, at least.

First to get a few things out of the door. Don't hire growth hackers. 99.99% of them are useless and are trying to sell a mystical art they often didn't even succeed at themselves as something you can just buy as a service. It doesn't work that way. If anything, growth should have been in your product from the start. The product has to be intrinsically viral nowadays. It needs to be something people really want or need. Like people staying up late using your app/site/product. People mentioning it to their friends for days. People need to be excited for it. You can't just get a growth hacker to buy that. Additionally, growth hackers usually have dark blackhat spammy techniques that might get you short-term results, but long-term you'll be punished by karma (or Google, or $insertOtherMegaCorpOrLawEnforcement).

Why is organic growth better?

So what *should* you do? Growth to me should be *mostly* natural and organic. Especially in the early stages (e.g. after launch) of your product. There's a few arguments I can make for this.

First off, it's free, you don't need to pay for ads or traffic.

Secondly, it gives you direct feedback if your product is good enough or not. If you'd be using non-organic ways of growth like ads, you'll see your product's usage grow. And you might think things are going well. But when you stop paying for those ads, those users are probably gone fast. This is actually a gigantic problem with funded startups. They can keep their big numbers up while they pay (sometimes 100%+) of their revenue in ads. But once the funding runs out, it all falls apart. So what happens if you don't do that? And you go for organic growth? You'll quickly see your product NOT take off, and you'll have to tweak stuff until it *does* take off.

Before you say it, I'll say it. Yes, you can combine the two. Obviously you can segment paid traffic and organic traffic and see who is doing what. That's probably smart. But personally I'm always for organic first. See if you can get your usage up yourself. It's cheaper and it gives better direct feedback if you're doing the right thing.

When you've got nice traction (depending on your app obviously, but I like 100,000+ MAU as a threshold), you might be able start scaling it up with paid traffic like ads. Why then? Because you've already proven your product works for people. Now you just need to get more of those people in to figure out that they want/need it.

I see a lot of people do the opposite. They start with paid traffic first, then end it once money runs out, and then they see the product fails. Because all the users they ever had were artificial. Don't be that person.

Especially...fake users

It's pretty crazy I even have to mention this, but I see this too often: people **buying fake users**, engagement, followers, likes etc. This is the fastest way to failure. If you didn't get the memo, people don't really care about followers/likes anymore. Actually only having a few, might give you the feel of being an "underdog" and they'll love to support your brand. Running bots on

social networks that favorite posts with certain keywords (try tweeting #growthhacking for fun!) is blatant and people will actually get annoyed by you and your company and you'll be worse off.

I tried it on Instagram to get a headstart for Nomad List, because I was late to get to the platform. And really, it doesn't result in any organic traction. It's better to come up with a strategy to build your brand on there for that specific platform than to just buy yourself into it. Again, it won't result in organic traction.

Trust me, if people like your product they'll use it and you'll get people engaging with you. There's no shortcuts though, this can take time. Years.

How to get organic growth

Growth can be divided into two by 1) either reaching people who don't know your product (new users) or 2) people coming back and using your product more (returning users). Reaching new people takes effort and means you literally have to reach out of your current environment to find them. Re-engaging previous users is a lot easier though. We'll now discuss both.

Get new users

Keep launching

The best way to get your first users is to launch the first version of your product fast. But a first launch will just be a single spike of new users for a week or so and then it drops off.

Nobody ever said you can't keep launching though. If you add a whole bunch of new features you should tell the world about it. And the best way is to just launch it as a new version of your product.

You can't do this every week, or every month, really. But, but every few months - why not??. Sure. It shows your product is evolving and getting better. A good

way is to put a version number on your site. You want to launch Product 2.0. And then 3.0 etc.

Spinning off

So many times when making a new feature I found out that feature was strong enough to be a product of its own. For example, I started adding coworking spaces to Nomad List. The thing is, there's lots of non-travelers that a giant index of coworking spaces would be useful to. So I spun it off as a separate app called Places To Work (**placestowork.co**). It runs well on mobile and lets people walk around while it uses the GPS API to see where they are and direct them to the closest places to work.

Spinning off stuff works well because it also gives you an entirely new launch to get traffic from. The product you spin off should be strong enough though to allow it to be its own product.

Tell stories to people & press

While you keep improving your product, it's also important to keep getting into press. Even after you already launched the first time. A way to do that is by telling stories.

Instead of just adding a feature to your product and ending there, every new feature can be a story. How is what you just added changing the vision or path of your product? With **Nomad List** I started with a list of cities. Then I added coworking spaces, hotels, and finally you could see where other travelers were at any moment. You can see that as just adding features, but you can also make it a narrative where you're slowly moving down the chain of scale to capture every little part of a traveler's experience. That narrative is interesting for press to write about. Just adding a feature isn't.

Stories can be more personal than just features though. Tell about your own journey and maybe struggles before or during the build up to your product. Before I had anything really successful, I had a YouTube channel for electronic music that made some money. That money made it possible for me to travel

and work somewhere else than home. That experience helped me make a site for those travelers. And that process of building a bootstrapped startup gave me the information I'm sharing with you now in this book.

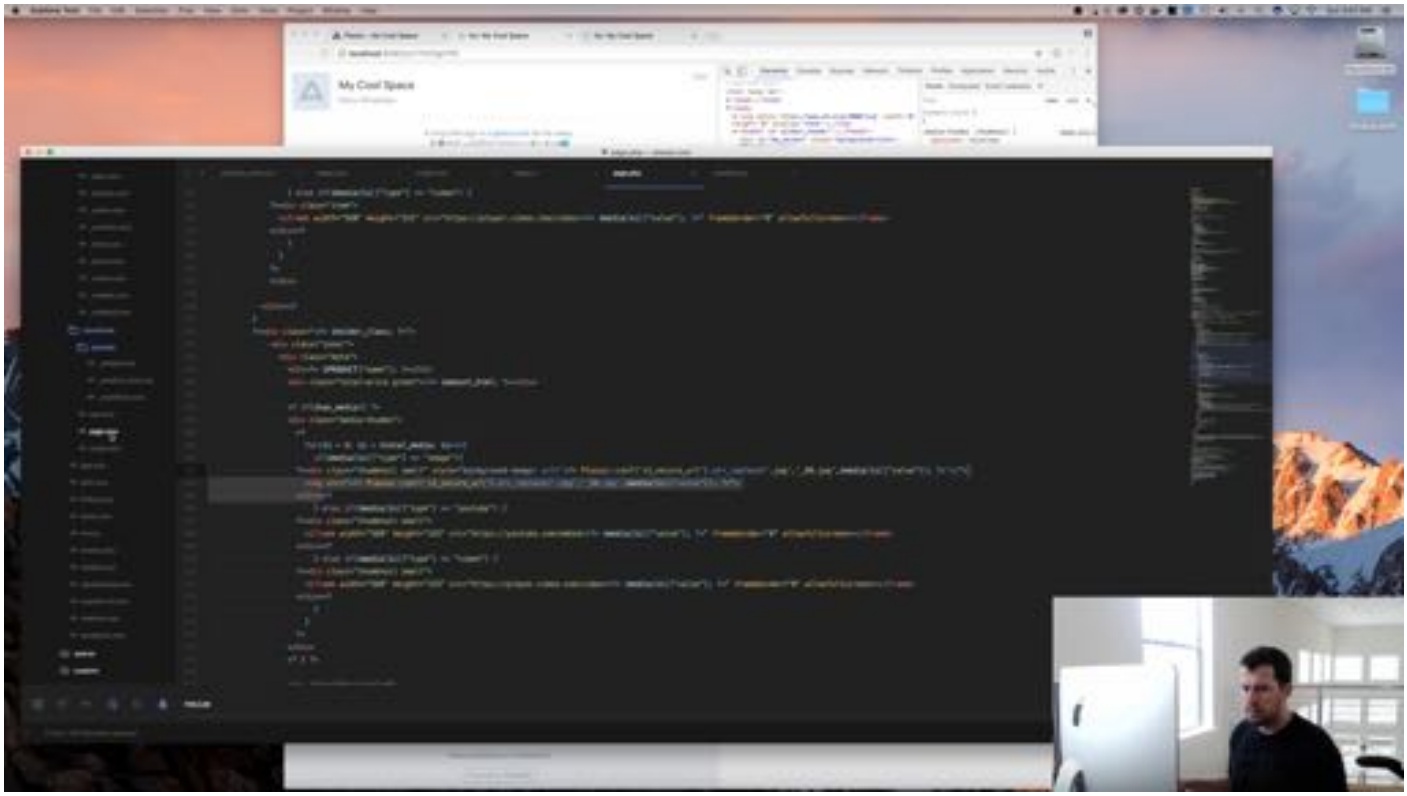
That's another narrative. And it's much more original to tell that than to just say "I build another internet company". You did but there was stuff before and after that that's more interesting to people you meet (that spread your story), and especially to press. Your story is unique in some ways, so figure out why it is. And tell that.

Build in public

One way of story telling is building in public. The last few years, one of the most effective strategies to get promotion has been to "build in public". It means to publicly blog, vlog or in any way communicate the entire story of building your product or startup from the start. You're completely transparent about how things are going, even if things don't go so well. It's about sharing the stuff you learn from trying to build a product.

One of the greatest examples of this strategy is **Buffer**. From the start they've shared every little step in the way to getting where they're now on their blog **Buffer Open**.

Another good example is **Drew Wilson** who **livestreamed** the building of his new product **Plasso** for a whole week. He didn't just stream his screen, he streamed himself sleeping, waking up and making coffee. It worked because it built attention around him and his product.



Why does being transparent work? Because people like to be part of success (or failure) stories. And with every little step, there's an opportunity to get attention and press by sharing it with the public. People become engaged with your product by reading about it and a share of those people may become actual users of your product.

In that respect, transparency is very meta in its nature. By sharing about the product, it usually makes the product itself do better.

You have an advantage right now (as of writing) because most companies are not transparent at all. It's a relatively new trend. They're not open about their progress. So for now, it'll bring you attention. There's a clause though: if the people who are interested in your progress are also potential users of your product then do it. It doesn't make sense otherwise because they'll never convert into users.

This goes further than startups. Right now we live in the age of transparency. If you show your vulnerabilities and mistakes it doesn't make you look weak, it makes you look like a human. And that works because your audience is hu-

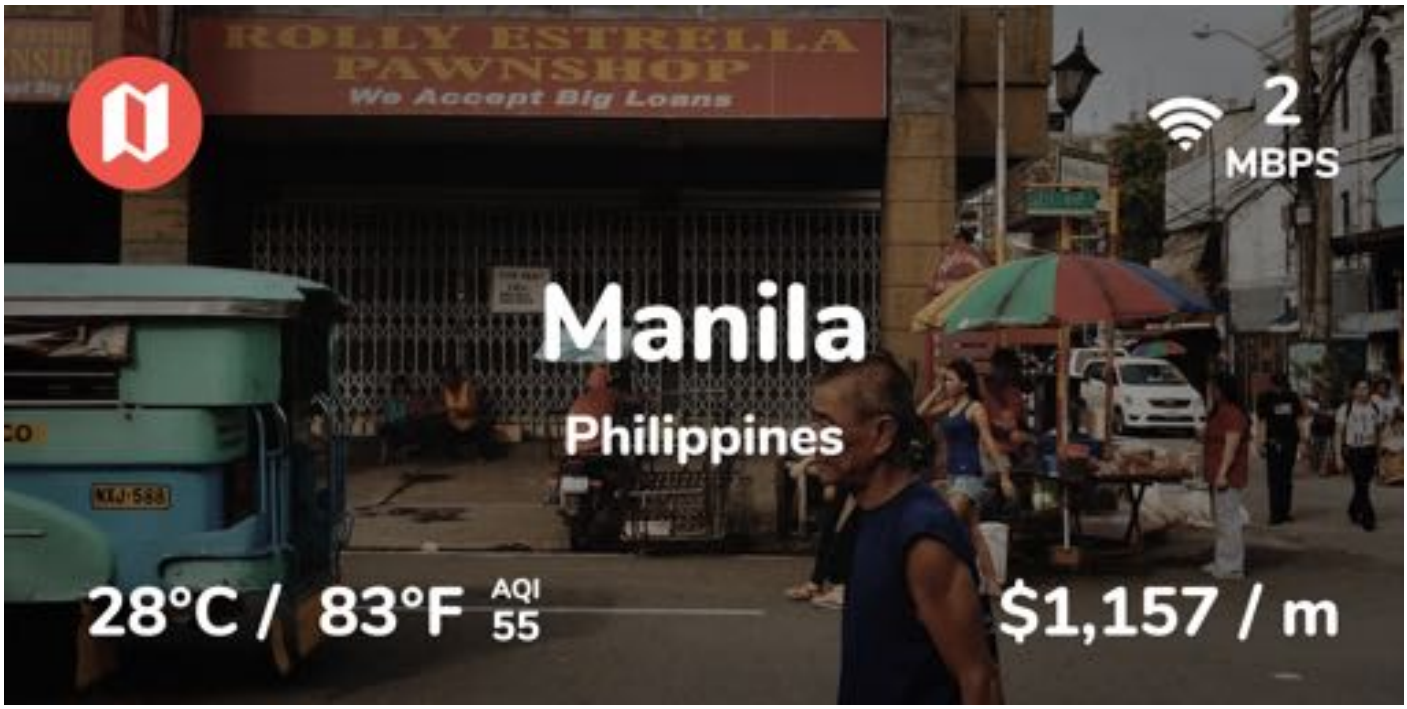
mans (okay, and cats!) and they'll recognize themselves in you or your company.

A good example of transparency is an app called Arq Backup, it automatically backs up your data to the cloud. In 2017, they discovered that one of their versions actually deleted your backups. This is the most terrible thing that could happen to backup software which has the goal of SAVING your backups. The maker of the app wrote a personal post describing what happened and apologizing. People were angry but their anger was less seeing how the maker handled it. Don't hide. Show your mistakes. Fix them. Behind every company is imperfect human. Your users know and appreciate this.

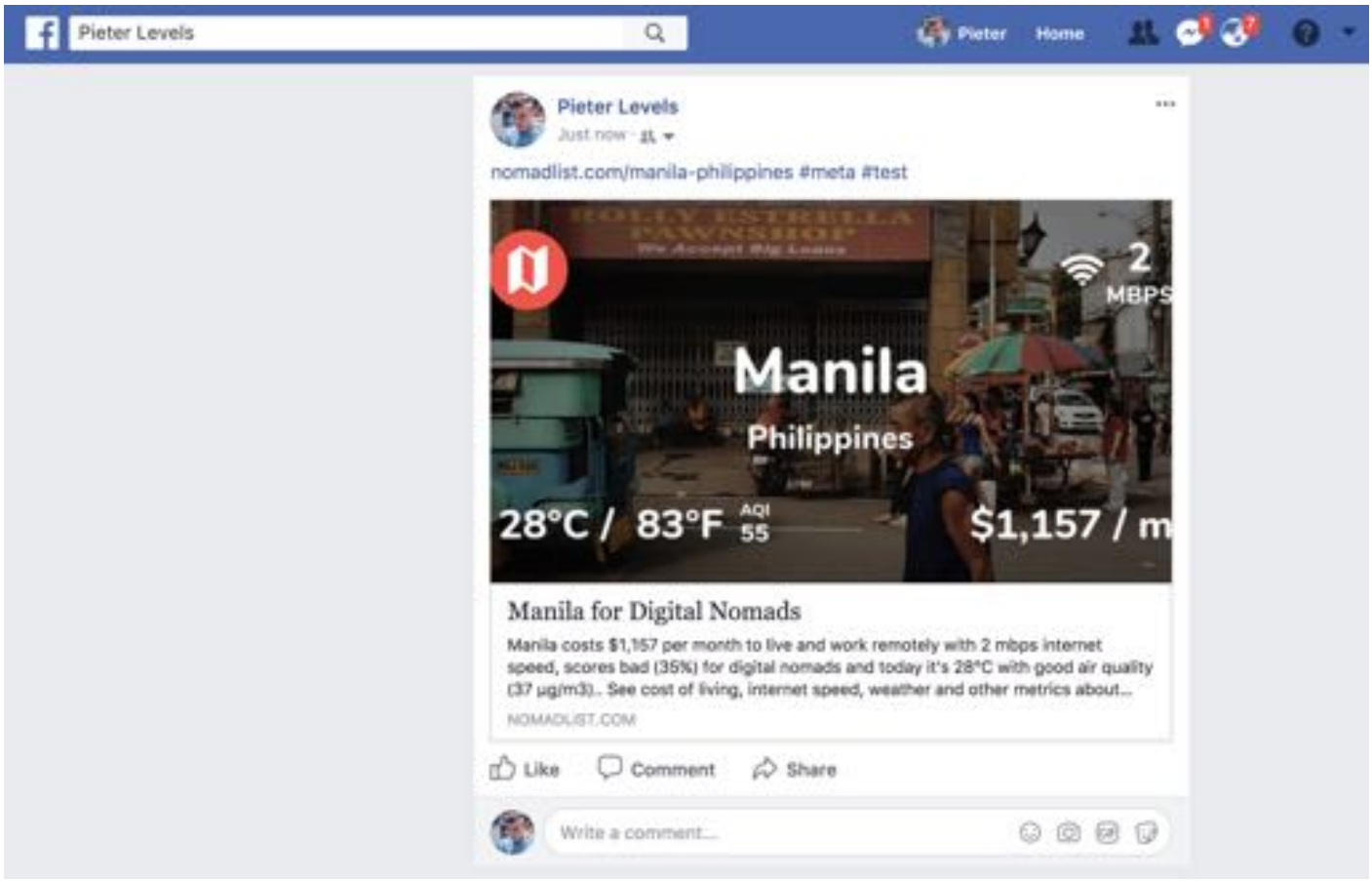
More on this in **"Telling your story"** .

Make people share easily

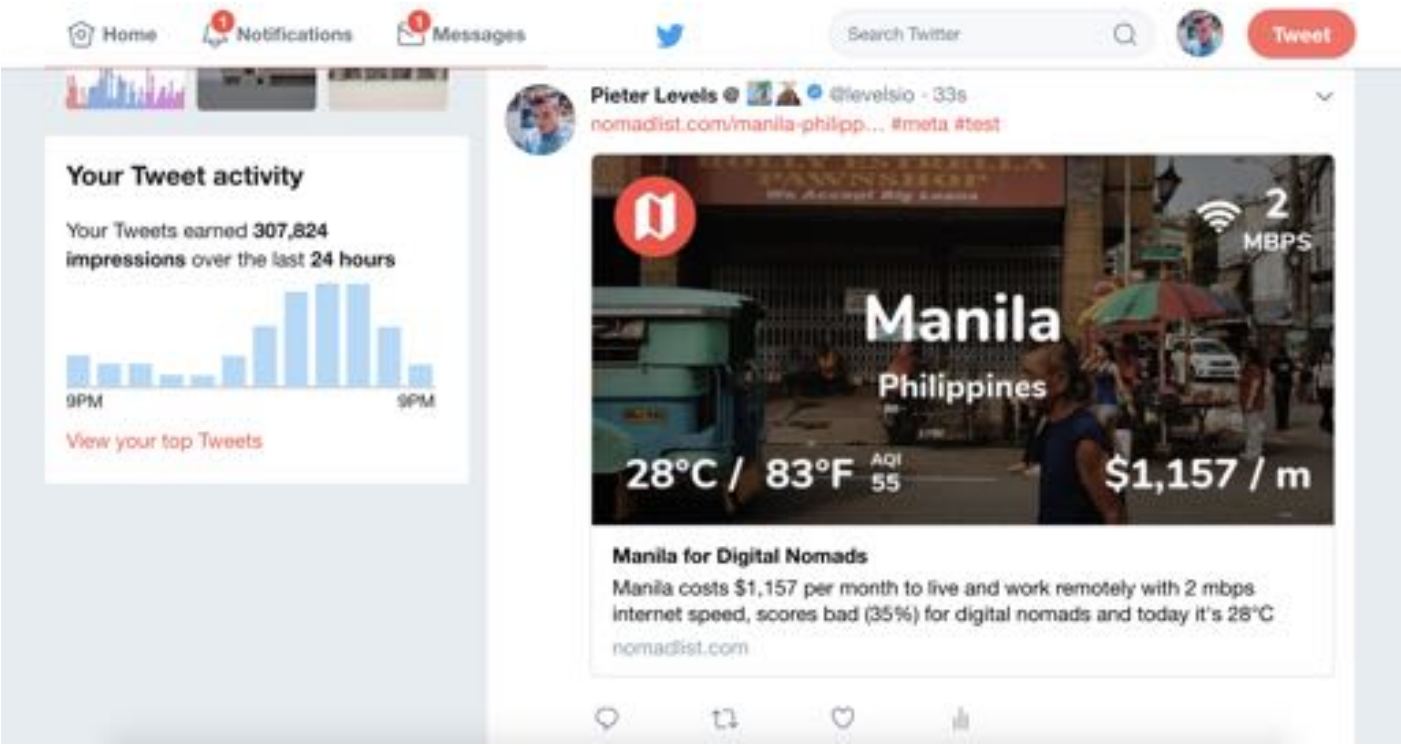
This is a very common thing so many people get consistently wrong or forget but it's so major. You can undoubtedly argue about how important social networks are these days, and I'm not a giant fan of Facebook either, but they still account for large parts (up to 30%) of all traffic sources on the web.



Facebook uses a set of <meta> tags in your HTML called Open Graph and they start with "og:". Make sure you fill them out for every page on your site and if you can just program them so they dynamically get generated. The most important tag is "og:media" and it lets you set a picture for each page on your site. It's very important to use this! A shared link with a picture on it gets so much more clicks than one without.

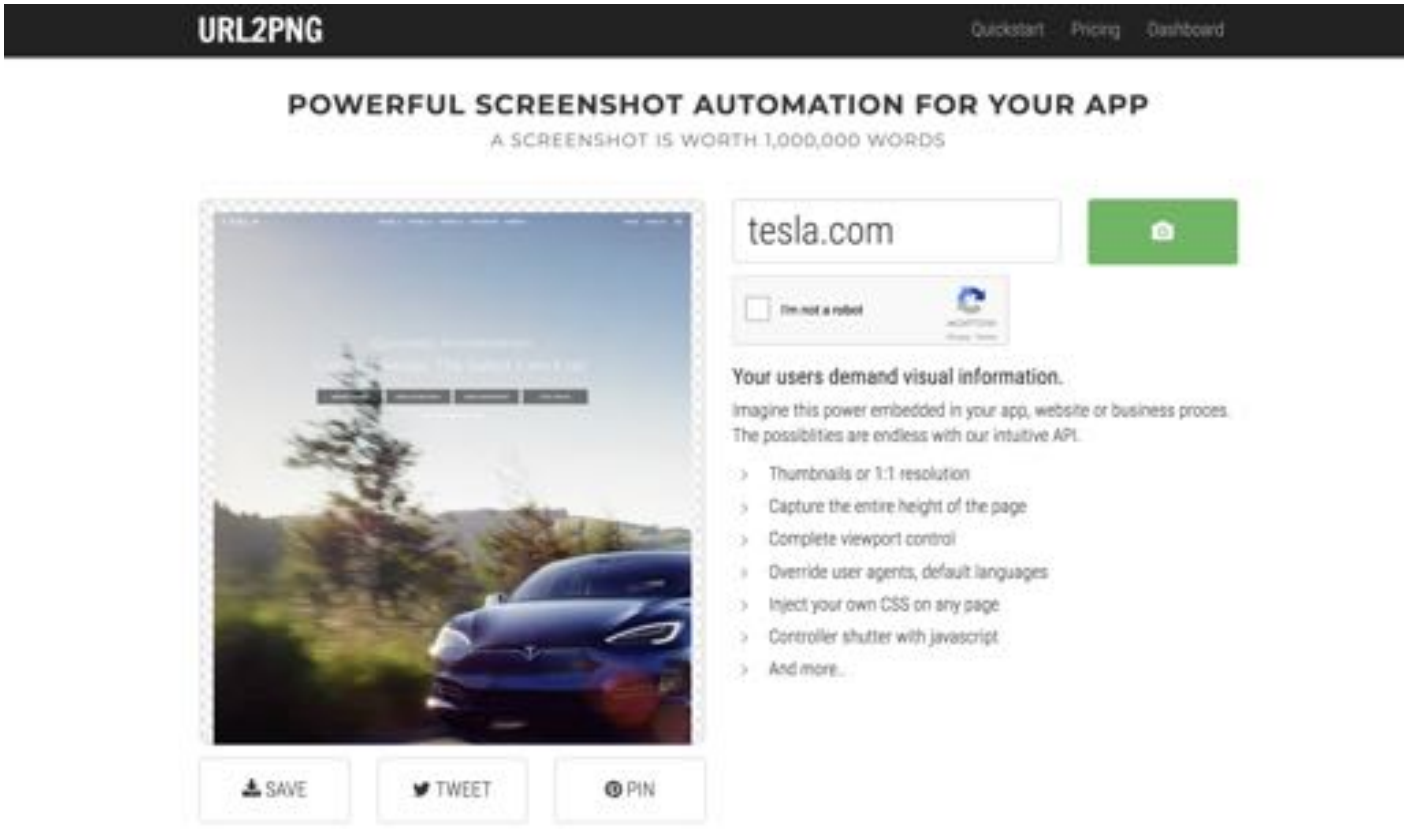


Twitter uses something similar called Twitter Cards. It also involves <meta> tags and when it works it'll show a big picture on the Twitter timeline when people share your URL, just like Facebook does.



Even if you're making a native iOS or Android app, it makes sense to let people share stuff and add some little tiny web part of to it to do just that. A good example of a big company doing that was Instagram, who never had a big web presence. They just had a page with the shared picture and a button "Install Instagram" back then. That was good enough to reach the user's goal of sharing the picture and the company's goal of growing the app's installs.

Here's where it gets even better. If you have some programming chops, you can make the image for each of your site dynamic. Product Hunt uses a service called **URL2PNG** that dynamically makes screenshots of each page on their site, so that if you share it you see a preview of the page. That's cool. But you can do better.



Make a page in HTML that displays the most important information about the page you want to share and is very visual. For example, if you have an app for marathon runners, and you want them to share specific running routes, why not make a page that shows the entire route on a map with thick fat lines and maybe some data like "25 km / 15 mi" written in the top right. Now you think, what do I do with yet another page? Well this isn't a page. You're going to screenshot this page and use that as the image for the original page users would want to share. That screenshotting can be automated with **URL2PNG**. Or if you want to do it yourself with a Headless Chrome script that screenshots URLs. It's a bit too extensive and technical (and subject to change) to go into this deeply. But Google everything I said and you'll figure out the technicalities of it. The goal is to make dynamically generated shareable images for every page on your site. You'll see lots more people come in from social media after you do this.

A great example is **Gyroscope**'s app on iOS. It lets you import lots of data from your fitness trackers and iPhone's Health Kit sensors. Then it shows you visualizations of that data in a really pretty format. For example, I can see a beau-

tiful map of my city with a path traced of where I walked that in the last few weeks. It's shareable to my friends and gets more people using Gyroscope.

Human-readable URLs and slugs

Let's talk about URLs. Do you have an interactive element to your site? For example, if you have a product comparison site, do you have lots of filters people can select? Make sure that after people select new filters you add data to the URL of your page so that they can share the *exact* state of your page when they shared it. Instead of

> http://compare.com/razorblades

Make it into:

> http://compare.com/razorblades?size=large&blades=4&color=blue

Or even better:

> http://compare.com/razorblades/large/4-blade-razors-in-blue

Make sure *every* interactive state of your app or product is shared in the URL or it'll confuse people.

More cool stuff you can do is letting people share specific stuff. For example, on your product comparison site you can suggest sentences to share like "The Gilette 4-blade razor is 6x more powerful than the Dollar Shave Club 2-blade". Something like that. Program your site to make dynamic sentences that explain the current state of your app the user is in, but in human language.

On Nomad List, if people select the filters "surfing", "fast internet", "near a beach" and "safe for women", my JavaScript code generates this URL:

> https://nomadlist.com/cities-with-fast-internet-safe-for-women-to-travel-alone-near-a-beach

When the page is opened, it will look like this:

It's not easy to code this, and it took me awhile, but it's important to make the URLs easy to understand and share for the user.

Launch an API

A good strategy to grow is to launch an API. This means you make your data (and sometimes features) on your site accessible to other startups. Well-known API's are Facebook's API that lets people "Login with Facebook" with-

out creating user accounts. Or Twitter's API that lets you tweet with a user's account from within your app.

Making an API means other startups can integrate your app's features and data into their app. In case of Nomad List, that might mean Booking.com shows a Nomad Score next to destinations on their website. This is good because it might mean more brand awareness and traffic for Nomad List, and for Booking.com it helps their user experience.

There's big risks involved with sharing data through an API though. In case of Nomad List, I launched my API that shared ALL public data from my site. Within 90 days, there were 6 identically looking clones of **Nomad List** in English, Russian and Chinese. A few of them launched on Product Hunt and Hacker News with sucecss and there was no mention of **Nomad List** at all. It helped them, it didn't help me. People launched apps to the App Store using my data, without any links back. Obviously, this wasn't the goal and it was now negatively affecting my company. So I disabled the API after that.

That's why you should probably think about your goal with making an API and how much data and features you want to make accessible. You can also require companies to register to use your API (and give them an API key) and cut them off if they abuse it.

If an API works well for you, it means increased brand awareness on other sites, so it can act as a funnel to get you more users. And it helps the company using your API with extra data or features. That's great for both parties!

Build with your users

Another big part of growth and people coming back is how to get ideas for more features your users want to have. As mentioned before, on every website or app I have a small feedback box. You can set this up with a lot of services like Olark or Intercom. What they do is they add a small chat box on the bottom left or bottom right of your web page. People can simply enter like, "Hey,

there's a bug here" or "I want this feature." This goes straight to your email. The cool thing is that actually people send pretty cool stuff. They send lengthy emails, messages, reports on features they would love to use.

People love doing that because they love co-creating a startup with you. The people that give you those ideas are usually the best users because they're experts at your product. Choosing what to build next, based on what they're saying, is amazing because many times, after you've already solved your own problem, you think that this is it. A lot of people use the app have extra information for you. I'm not saying you should build for other people. You should build for yourself, but they can give you different perspectives on your own problem space a little bit. That's super useful to think of making new features.

Measure how you stand up against your competitors

Now that you have a running product. You want to know how it's performing relatively against your potential competitors. A great app for this is called **SimilarWeb**. The data isn't that accurate for smaller sites, e.g. ~10,000 visits per month, but gets accurate around >100,000 visits per month. Even if the data is a bit different than your Google Analytics stats, relatively to other competitors you can see if you're doing better or worse over time. That's pretty valuable to make decisions on. Remember: traffic is a very limiting metric though. Your competitor might get more users and visits but still make less revenue than you. Sometimes you can deduct how much money they make from their traffic though. Sites that get 1,000,0000 visits/month often make $1 to $10 million per year if monetized well. Sites that get 100,000 visits/month often make $100k to $1M per year similarly. It's a very very rough rule but 1,000 visits usually means $1-$10 (if they monetize properly).

Conclusion

So, what's the best "hack" ever to get people to come back and grow a product?

Make a really useful and great product!

That's *way* more efficient than trying to get users to share stuff on social networks, emailing them or sending them push notifications to come back to your site or trying to get into press all the time.

If people really love your product, they'll come back to it. It'll be planted in their mind. They *need* it! So whatever you do, focus on making the product great first. Then think about growth later.

That doesn't mean you shouldn't optimize for growth with the techniques described here. But if the product sucks, you can try to hack growth as much as you want, but it won't turn into sustainable growth and just die down again

Growth is a multiplier for an already great product.

Resources mentioned

- Buffer Open
- Google Analytics
- Drew Wilson
- Social media share images

Your homework

- Analyze your analytics from your current launch. What have you learnt? Did many people who visited your site or installed your app kept using it? How was the fall of users after a few days?
- Analyze the user feedback you got from users directly (through a feedback box), but also in the comments sections of the platforms you launched it on. If you had press coverage, what did they think about it? Use this feedback to immediately improve your product.

💰 Monetize

Introduction

You now have a working product with users, but you're not making money yet. Together we'll try apply business models on different parts of your product to see what users will pay for.

Why is monetization so important?

It seems too obvious to even ask but these days, in the world of startups, it's actually seldom you see any company make any real money

After Hollywood movies, the startup narrative has become so incredibly jinxed that you now have millions of people following a dream of becoming a billionaire by programming away while eating pizza and sleeping in the office. Who cares about money right? We'll become billionaires SOME DAY!

But most people don't.

So that's why especially now, it's more important than ever to focus on monetization. Monetizing a startup early on is the best thing you can do for yourself, your friends, family, society, and pets.

Moreso, if you're never able to monetize your app, you'll never be able to turn it into a business. You'll always be stuck doing maintenance on it for years to come. You'll never be able to hire someone to take over the work so you can focus on the long-term strategy of your company. Wait, no, it won't even become a company if you don't charge users for it.

> "You aren't running a charity. You're running a business. If people won't give you money for your product, you have an existential crisis on your hands."
>
> — **Josh Pigford**, founder **Baremetrics**

Don't be afraid to charge money

One of the most common problems I see with makers these days is that they're so incredibly scared to charge any money for their apps.

They make something, launch it, get people who use and love their product and then are afraid to scare them away by charging for it.

The problem is, if you never charge for a product, it'll never become a real business, it'll remain a side project forever. That's fine. But you'll have to maintain it forever too. That costs time and money. And the whole point was to make a business, right? So start today and call 1-800-STARTUPS (just kidding).

Don't be afraid. People are happy to reward you for your work. And if your product really improves their experience of life even a little bit, people are happy to pay for it. But those people will only be a small percentage of all the people that are happy to use your product for free.

What I mean is, if you have 1,000 people using your product actively, don't expect more than 5% to pay for it. Probably fewer. So your paying customer base is fewer than 50 people.

Here's the challenge. A side project can be successful with 1,000 users, but it won't be a business with 1,000 free users. You need 1,000 paying users. That

means you'll need a side project that would have 20,000 potential free users. Think about this.

That means not every project is monetizable. And if that's your goal, and it's not reaching that scale. Then it's probably best to shut it down and try again. Don't waste your time!

Charge money, get hate

Here's another thing that you wouldn't expect you'd get, but if you start charging for a product, you'll get hate. Internet hate. A lot of it.

There's an interesting subculture on the internet of people who presume everything should be free, "because Facebook is free too, right?". These people are the problem.

People will go on Reddit and other internet forums and write about your app or site and simply attack it with posts like:

> **HOW DARE HE CHARGE FOR A SIMPLE FACEBOOK SCHEDULER APP**
> **by GrumpyCat2019**
>
> i can't believe what just happened. so anyway i was feeding my cat and then i was trying to find an app so i can schedule my social media posts. i really put too much time into scheduling this so i need an app.
>
> so i found this app MediaScheduler2000.com, okay so i sign up and what the hell! i have to pay $25/month for it? who does the maker of this app think he is? what a capitalist. he's just making easy money over the backs of others. this should be free!!!1 it's always these big companies trying to make money off of the little people!!!!!even gmail is free!!!!11
>
> DON'T SUPPORT THIS APP!!! THE MAKER IS EVIL!!!11oneone

You'll get this a lot. And it's fine. Don't lower your prices. Don't make it free.

These people are *not* your customers. These people will *never* pay for your app, not even if you lower the price. And it doesn't matter that they write about you in a bad way even if they get a response from others.

There's enough sane people who do have money to spend and who'd love to spend it on your app. They will not whine about your price on internet forums. Instead, they'll pay you. That's your customers.

Build with monetization in mind

This may limit your creativity, but I strongly suggest you build with monetization in mind early on. Especially if you don't have a lot of cash flow now and actually need a project to give you some revenue (to literally pay your bills).

If you do have lots of cash flow (and runway), you can experiment more. If you don't have monetization in mind it can often lead to more interesting projects, that might also have more interesting ways to monetize later on. It just won't be as clear in the beginning. That's fine if you don't need to make money quickly.

Here's the interesting thing though which you see with both entrepreneurs and artists alike. Often, their best work was when they were constrained by resources. Musicians usually make the best and most authentic songs when they were poor. Entrepreneurs made their best apps when they needed to make something successful and their was pressure to pay their bills.

Here's a test: how many entrepreneurs you know who became successful (and rich) and then made a second project that also became as successful. It's fairly rare. And it's because success changes people's life. They are no longer (financially) dependent on the success of a project. They already have had money and success. Often the "chip on their shoulder" of "I'm going to show the world I can do it" is gone. Also having money gives them TOO MANY resources. They'll overengineer a product and since they're now in a small minority of successful people, they're not the target customer anymore either.

That's why the position you're probably in now, where you're heavily bound by resources is a competitive advantage. You can much more easily compete with big entrepreneurs than you think. You're pure and real. Probably more than I am now. I'm jaded already. You're limited resources make you creative. It gives you a very clear goal and a strong focus.

Monetization is validation

In the last few years, we've seen monetization become not just more important for startups as venture capital money runs out. We've also seen it become a validation tool in itself. If people are willing to pay for a product early on, it gives it good odds to succeed later. Because one person paying for something

means you just need to find thousands of other people like that person and also make them pay for your product.

A famous example of validation by monetization is **Buffer**. They started with a landing page that described their product and they added a "Plans and Pricing" button. Once people clicked on there, they discovered there wasn't actually a product yet. They could add their email to get a message when the app would launch.

This was 2011, a long time ago. My own iteration of this idea is way more direct. Add a Buy button, if users click it, show an actual credit card modal where they enter their details. They click Pay. Then you show an alert "Sorry, we didn't charge your card but thanks for your interest. Our product isn't done yet, but we'll let you know when it is". This means the user actually was about to pay but they couldn't because your product wasn't ready yet.

If you log how many people actually complete this process, you can make a rough estimate of how many people would pay per month and how much revenue you'd have. And you know if it'd be worth to fully develop the product (or feature) in terms of how much it'd cost you and how much revenue it'd make.

This type of validation is way better than just having a landing page without a product where you might get 10,000 visitors. You know nothing about the intention of these 10,000 people. They might not even like your planned product.

Business models

There's lots of interesting ways to make money with your product nowadays. I'll discuss the most popular ones now.

Limit features to paid users

One of the most common and obvious ways to monetize is limiting features for paying users only.

You make your app with every feature you want in it, launch it, then see what people use extensively and which features they value most. Figure that out with analytics or talking to users (or just intuition). Pick those features and test if you can make them for paid users only. Keep them available in your product's interface but once people click/tap them, show a popup/modal that reads:

> **To use this feature, please upgrade [UPGRADE]**

Then as soon as possible, push them to a payment window before they change their mind! Now you have a paying user.

This is powerful as the emotion involved here is, "Oh wow, I'd like to use that feature!" to "Aargh! I have to pay?" to "Okay, how much?" to "Hmm, $5 is not that much, okay whatever, it solves my problem!".

It's not the most fun emotion, but it works if you make the conversion fast and smooth. After they pay, immediately enable the feature and let them use it. Done!

Pay-per-feature

You can do this in many ways. You can let customers unlock lots of features at once. You can even let them become paying subscribers (more on that later in

this chapter). Or you can make each little unlockable feature a single payment. For example, $5 to use the sharing feature, $10 to add a world map. Whatever, really.

I actually think this "pay-per-feature" model has lots of potential and I'm surprised more people aren't trying it on the web. You see it in native iOS or Android apps a lot. Especially in games. You could do the same for web apps. Charge per feature that the users want to use and price accordingly. Either the most popular features could be the most expensive, or you could charge a lot for the features that are more custom and niche. Both models can work.

Ads

The online advertisement is in quite a difficult place. And it's put it there itself. Their ads are so bad, and so full of malware and viruses that we've all started to install ad blockers. And if we don't have ad blockers, well, we just don't really click on ads a lot anymore (again, because they're so shit). Ads these days are annoying, ugly and spammy. We can't even stand ads on television anymore so we'd rather pay Netflix $9.99 per month and watch that instead.

We can agree that traditional ads don't make any sense anymore in a time where we decide what we want to do, watch and listen to. We don't want to be interrupted by a voice that tells us to buy something we don't want or need. We're Generation Fight Club.

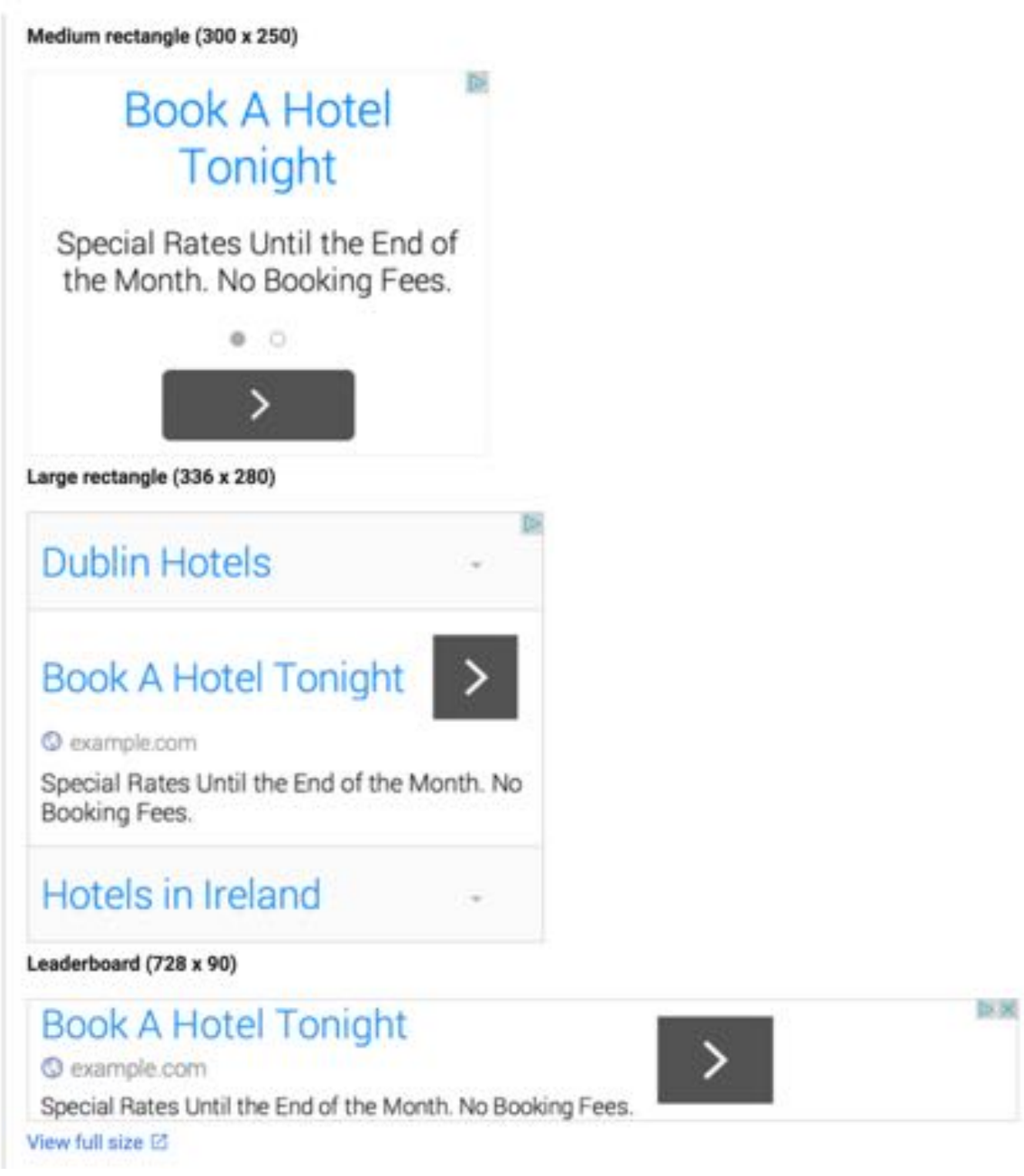
Is there any future for advertisements then if everybody is sick of them and blocks them anyway? Yes. They're called Native Ads.

Native Ads are ads that don't necessarily come from some big ad network like Google. You write, design and sell them yourself and customime them so they fit with your entire product's look, feel and, most importantly, objectives.

If you have a site for travelers like I do, and a bank comes up to give advertise with high-interest predatory loans, I'll immediately say no. Because it means

I'm making my users' life a lot worse. If there's a tool that travelers can use (and I would use), I'll happily let them advertise (if they pay me). So it's about having ads you'd like yourself, that are actually helpful, and that don't look and feel out of place. Joe Rogan's podcast is a good example of another product which maintains the same strategy. Joe only advertises with products he'd use himself, and he tells about them in his own way, without a script. That's native too.

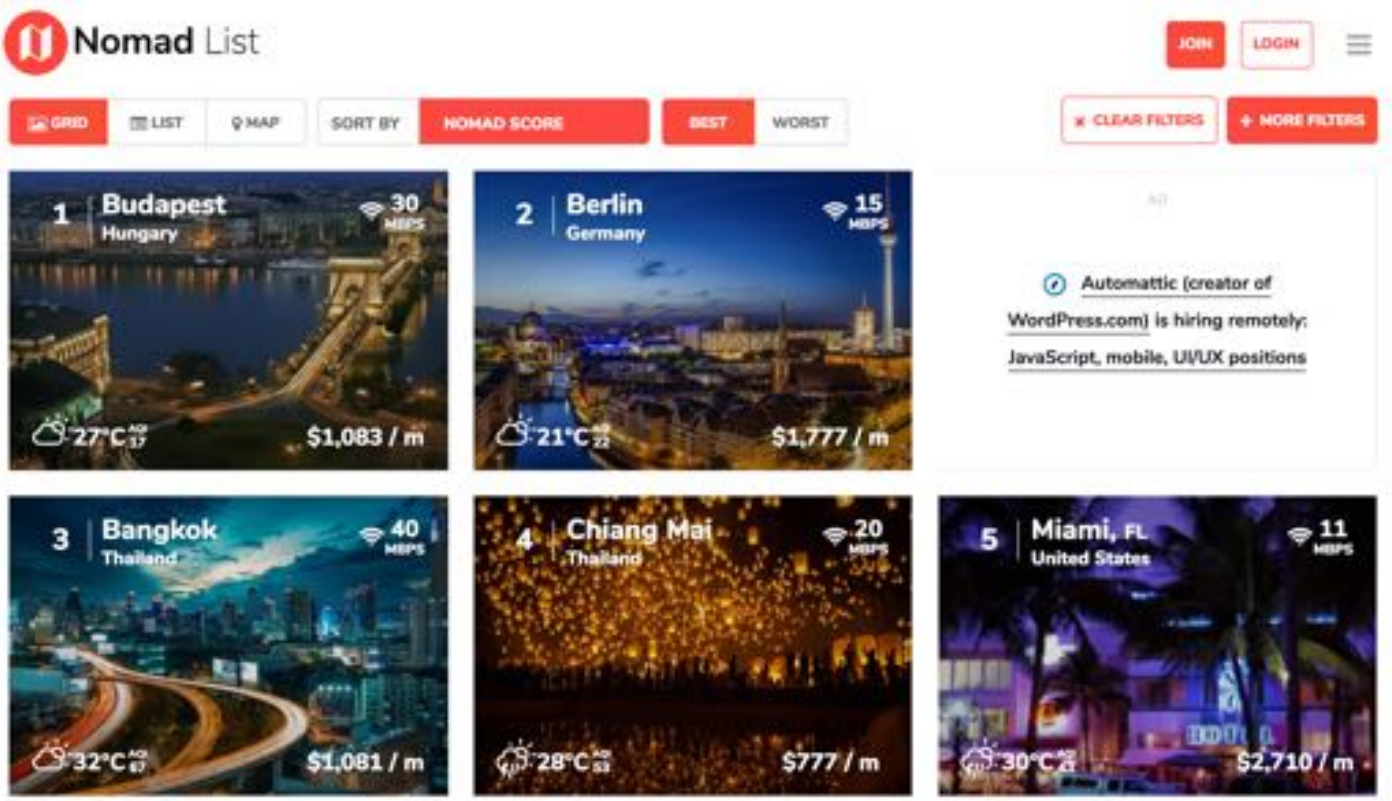
Do native ads pay well? It depends. For me, they pay a lot better than Google AdSense does. Let's say I'd install Google AdSense on **Nomad List** today. I'd get an ad like this:



Firstly, it looks ridiculous and spammy. Google pays about $1 for 1,000 views of this ad. With 500,000 views, I'd make $500 per month! That's not a lot for a site my size. Secondly, it's not extremely relevant. Sure it's a hotel booking ad, but how about something my users actually need? I think they already know how to book a hotel by now!

Instead of using Google Ads, how about I talk to a company that is really interested in travelers and they have a product that closely fits with my audience, such as a high-tech backpack. Or a VPN, so people can always connect to the internet securely? Or a laptop stand that travels could use to work remotely and not get back injuries? Or a remote work company like Automattic who recruit remote workers?

In that case, I could do a native ad (hint: it's the Automattic one).



There's a few things that are different here vs. the Google ad. This native ad is friendly, honest and simply designed. It fits the look of my site. The topic fits the audience of my site. And it pays 5x to 20x as much (e.g. $2,500 to $10,000 per month) as the Google ad because I make a custom deal with the company instead of getting resold through an ad network.

Guess what? My users are happier seeing native ad than an ugly looking Google ad that most of the time doesn't make any sense anyway. It makes my site look more classy, and I make more money!

Happier users, happier advertisers, higher conversion and you make more money. What about ad blockers? Well, since it's a native ad (it's custom), it's hard to block. Plus, because the ads are truly relevant to the audience, users won't despise the ads and want to block them, generally.

Native ads don't even have to look like ads. They come in endless shapes and sizes. IT simply means you connect your audience to a company that fits and where both your audience and the company gets a positive result from it.

Congratulations! You've just created value for everyone.

One important note: always specify a native ad is in fact an ad by writing on it "Sponsored" or "Ad". If you don't, you'll be breaking advertising laws in many countries!

Sponsorships

A popular model, especially with open source projects and small side projects, is sponsorships. You make something that many people find very useful and it aligns with a big company's mission. If you get some attention for your project, a company might ask you, or you can ask them, to sponsor you. This could be a fixed amount per month so you can cover the costs of keeping your product up and of course continuing to develop it. In return, a company might not require anything, but mostly they'd like to have their logo appear on your site or about page.

Which companies should you aim for here? Well, companies with lots of (venture capital) money in the bank that look to grow and get more brand awareness are a good target. If they don't reach out to you, send them a short email introducing your product and how it aligns with their mission. See if they'd like to sponsor it for $250 to $2,500/month. If you get 4 sponsors at

$250/month, you're already profitable enough to keep yourself alive (somewhat).

Patronage

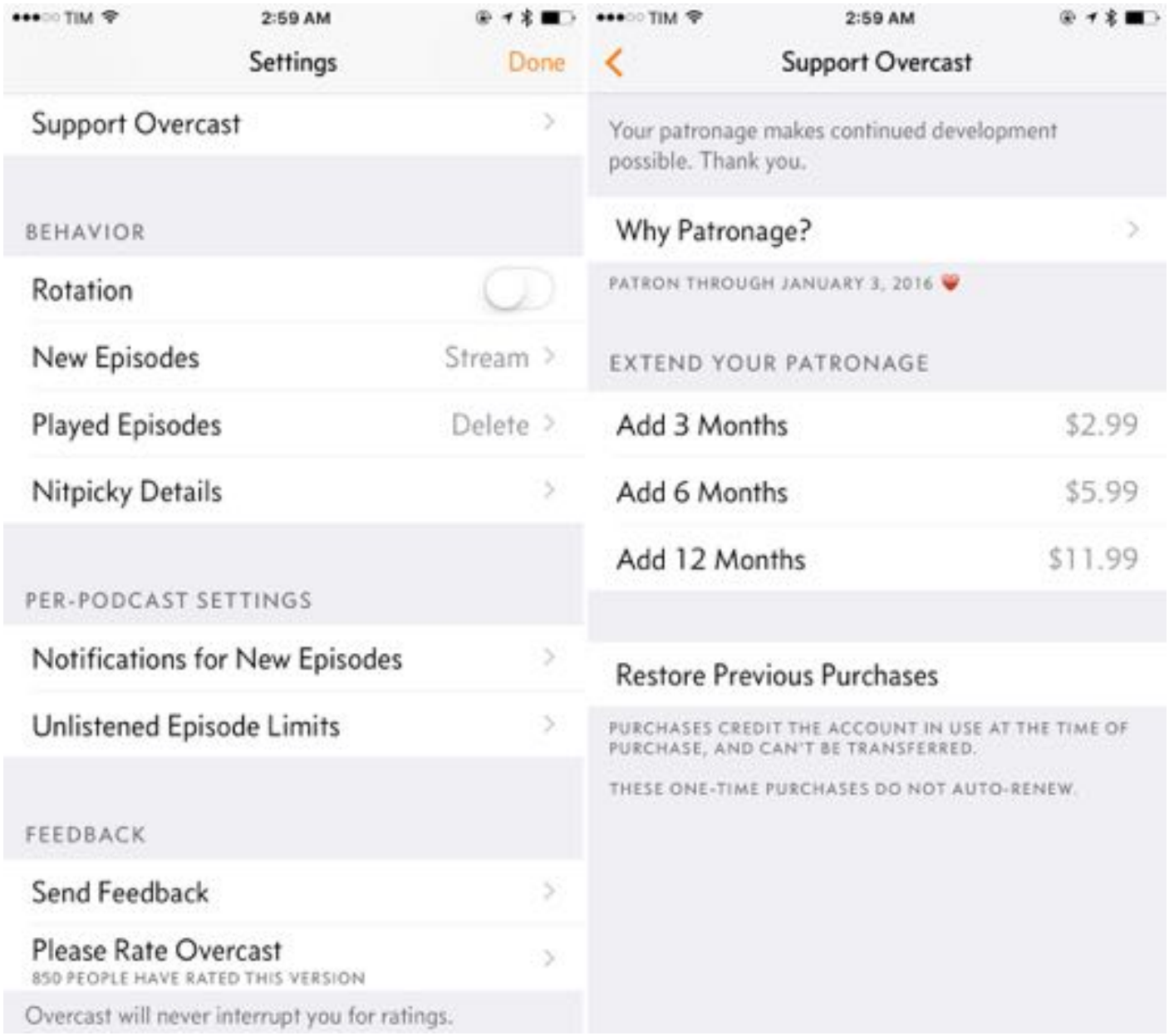
Patronage is a more recent and very interesting model that's only going to get bigger in the future.

Patronage originates in the Middle Ages when artists were paid by royalty to live in with ehm and paint their daily lives. They made portraits of them but also scenes and landscape paintings of important events. Pretty much they were the digital camera for kings back then. The good thing is, it allowed artists to get fed, get a roof and stay alive. On the side they could work on their passion of making real art. Kinda like a side project.

Fast forward a few centuries and patronage is back. In our case it means that if we're a creator, illustration, musician, artist or just a maker, you ask the people that like and use your work for money. Directly. Usually your work stays free, but you simply ask for monthly donations. Many people struggle with this because it comes across as begging, but these days it's a bit more classy than that. People are less afraid now to say "I want to work on something cool but I have bills to pay, so help me!".

There's a few platforms for patrons in different industries. The main platform is **Patreon** which is mostly focused on creatives but now branching out into journalists and even game and software makers.

We're now also starting to see patronage in apps itself. A good example is **Marco Arment** who made a podcasting app called **Overcast**. The app itself is free (and very good) but after you've used it for a while it'd ask you "Do you want to support Overcast's development?":

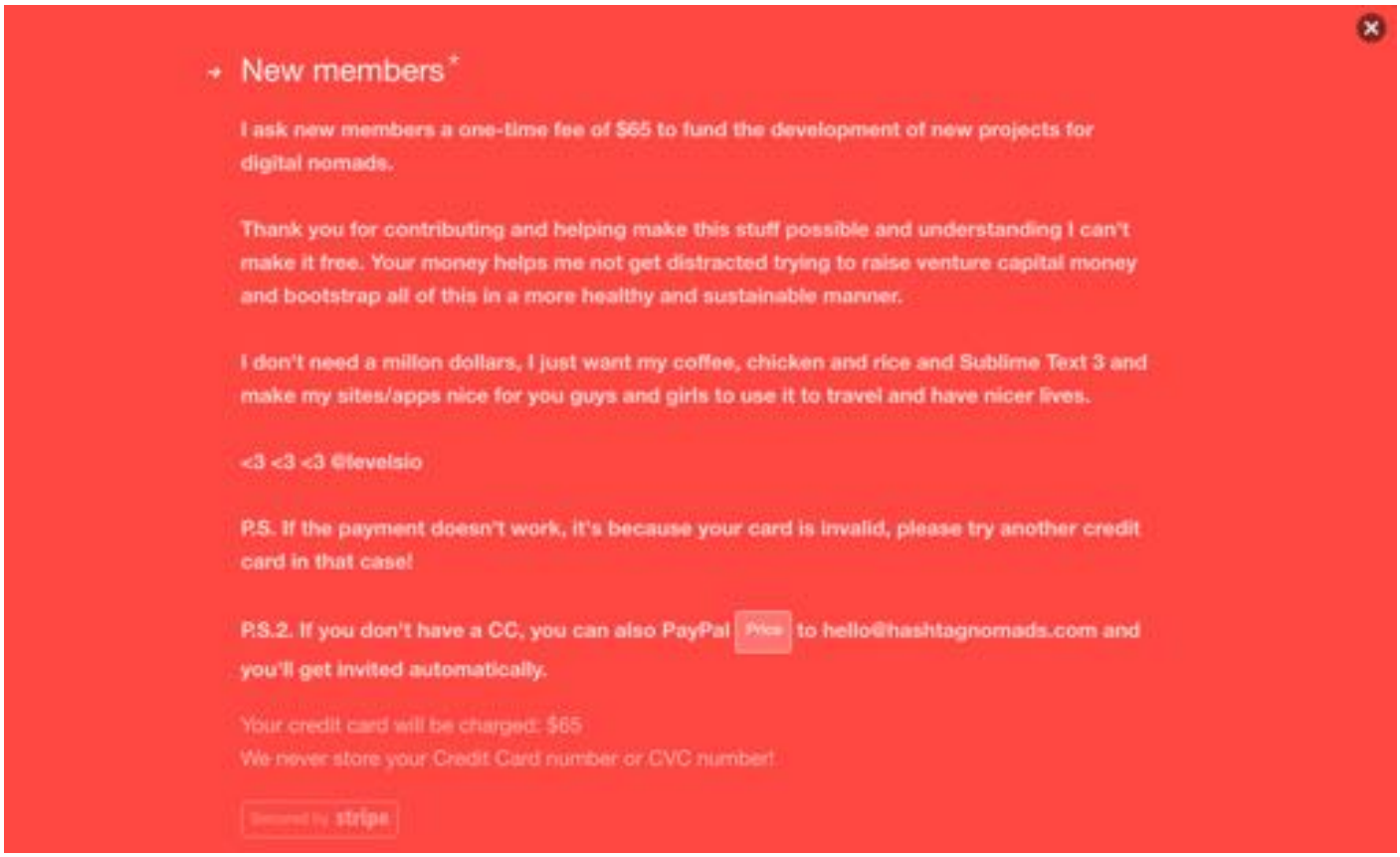


It gives you the option to pay several different amounts of money (either re-curring or not) through an in-app purchase. Strikingly, the in-app purchase doesn't unlock any new features. That's typical for patronage usually. It just means you're now a supporter of its development. The feeling itself is what you pay for.

This is incredible in so many ways. Just the fact that you don't need to some-how trick people into unlocking features, but can ask them straight up "Hey, do you like this thing I made? Support me making it please!" is magical. I don't think this would've worked a decade ago.

The issue is that asking users for money to support development goes against the idea most of us have been brought up with about entrepreneurship. You make a great product, there's demand for it, and people pay for it. If people don't pay, it probably means the product isn't good enough (as the market judged it and didn't want to pay for it, right?). With patronage, you make a great product, people use it for free, and those who feel a personal connection and are fans of it will pay you for it. This also means apps that are so niche and would never get enough audience to be profitable to develop suddenly become economical to build, as long as you can get enough people to just support its concept.

Personally, in the early days of Nomad List, I actually tried this model and it worked really well. I had a sign up page on **Typeform** and I'd ask:



People these days understand you are a human and need to pay your bills (like them). And many are probably increasingly attracted by an authentic request for money than a big marketing page with flashy buttons to unlock the full version of your app.

It was an honest and frank request for money. It worked and people appreciated it. It wasn't a big sales page (like I have now), it was just very basic. Patronage has its time and place and for me in the beginning this made sense. Now I feel I can make more money by in fact selling it as a big product. But patronage bootstrapped the first year of Nomad List! When it makes sense for you is up to you to judge. I'd try it for sure.

Subscription-based memberships

If you've heard about software-as-a-service (commonly abbreviated as SaaS), you know why subscription revenue has become such a big deal in recent years.

If you don't, it's the idea of offering your apps, not for sale, but for a subscription that keeps going. Like your television subscription. If people stop paying, they also can't use your app anymore. In return for payoing, you keep maintaining and updating the app and give them customer support.

Subscription (or recurring) revenue is somewhat of a holy grail for entrepreneurs. Let's see why. We assume we have 1,000 customers per year and they pay a single payment of $75, or in the second model, a subscription of $75/year.

	Single payment $75	Subscription $75/year
Year 1	$75,000	$75,000
Year 2	$75,000	$150,000
Year 3	$75,000	$225,000
Year 4	$75,000	$300,000
Year 5	$75,000	$375,000

What if you have a growing business and sales grow by 25% each year?

	Single payment $75	Subscription $75/year
Year 1	$75,000	$75,000
Year 2	$93,750	$168,750
Year 3	$117,187	$379,687
Year 4	$146,484	$854,296
Year 5	$183,105	$1,922,167

See how fast recurring revenue grows? Obviously, this is without people canceling a subscription (that's called churn). Let's be pedantic and add churn of 7% per year. That means 7% of users cancel their subscription yearly.

	Single payment $75	Subscription $75/year
Year 1	$75,000	$75,000
Year 2	$93,750	$163,500
Year 3	$117,187	$356,430
Year 4	$146,484	$777,017
Year 5	$183,105	$1,693,897

See? Even with a moderate churn, it'll still grow much faster than a single payment does. After 5 years of single payments, 25% annual user growth and 7% churn you'll be making $1,693,897 of revenue! Versus only $183,105 if you use single payments. The difference is $1,510,792. Recurring revenue is *very* powerful.

There's a very good reason why entrepreneurs like this. It's easy to sell a company like this. Acquiring companies love subscription revenue as it's highly predictable and relatively stable. If you know the growth rate and the amount of people canceling, you can evaluate the company easily and in turn, value it with a good price.

From the customer's perspective, subscriptions might not be that great though. People spend much more money generally. They often forget to cancel their subscriptions and then they renew for another month or year, meaning they overpay. It's hard to get that money back then. Actually, "forgotten subscriptions" are the elephant in the room of subscription-based businesses, with some assuming up to 50% of subscription revenue are by customers that don't even use the product anymore. Obviously, subscription business owners are the last to discuss this. It's their cash cow. But it's dark. A solution to this would be to specifically ask people to renew their subscription or not after you detect they're not actually using your product but are still paying for it. Nobody is doing this though that I know of yet. Obviously. It'd instantly make the subscription model more healthy for users, and still beneficial for businesess.

What should you offer as a subscription though? Well, anything really. Your app. A service. A forum. A chat group. A social network for a specific niche (like I did). Most apps people repeatedly use work with subscription revenue. You might see a drop in sales though when switching to subscription revenue because subscriptions require commitment. And not everyone wants to commit to another subscription in their lives (besides Netflix). This depends heavily on your audience. Personally, I have become a bit tired of signing up for another subscription.

A good middle ground (that I use) is to offer besides subscription, also a one-time-payment lifetime memberships for the predicted lifetime value (LTV) of a customer. That means if you know on average your users remain members for 2 years and you charge $100/year. Your lifetime membership will be $200. That means you'll make the same amount of money as with subscription, they just pay the entire amount up front.

Community model

Subscription payments work especially well with the business model of building a community. What's a community? It can be a discussion board, forum,

chat group, social network to physical meetups. I do all of these with Nomad List.

Communities are attractive business models because by default they're niche. You make a community for travelers, or motorbike enthusiasts, people who have horses, fans of Ariana Grande. Whatever. There's endless niches to serve. They're also attractive because if you become the dominant community in your niche, you have a monopoly and you can charge high prices. As with social networks, communities have the famous network effect. The more people you have in a community, the more valuable it becomes (and the more money you can charge). Each user usually has a profile like this:

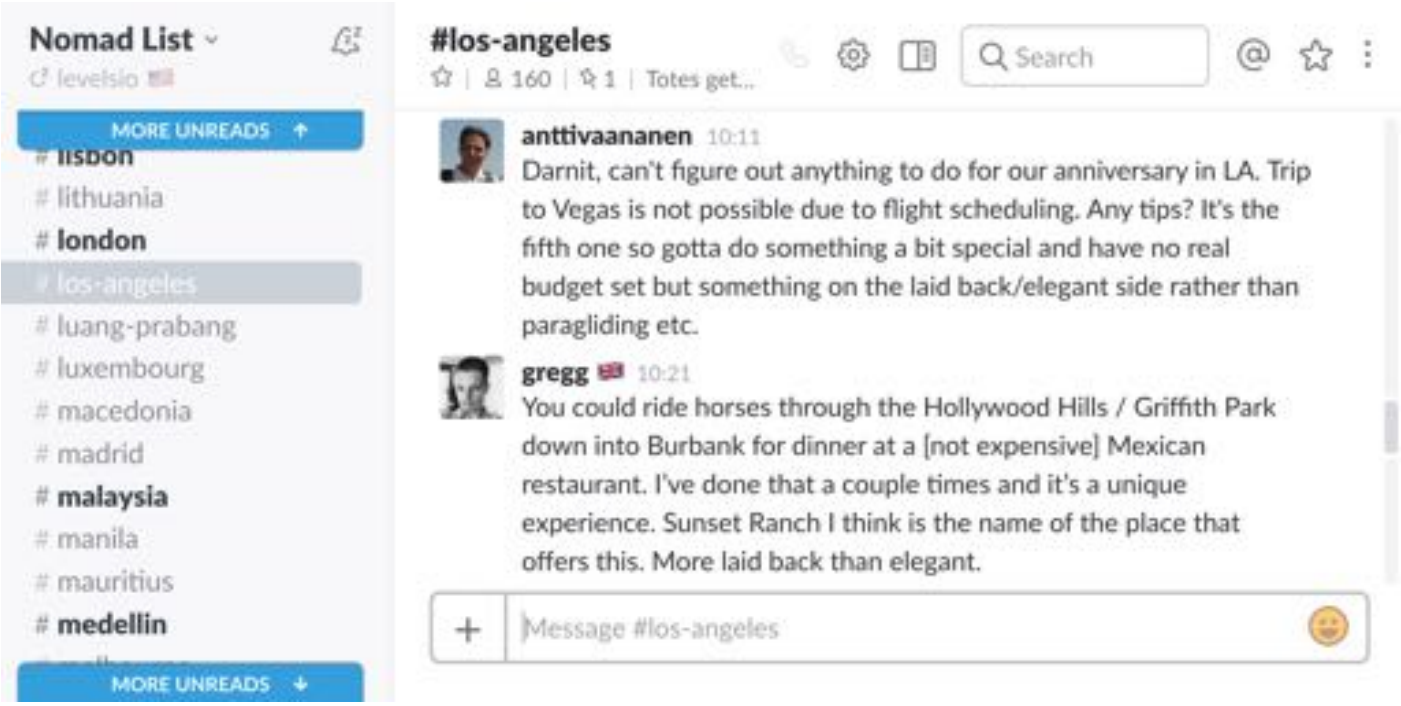
Operationally communities can be run quite lean because community software like forums is quite maintenance free, you might need a moderator but that's it. On the other hand, you can go as big as you want. I started with a chat, but now have a forum, a location-based social network and physical meetups all over the world. The model is also highly relevant to current time. People don't like to pay for content anymore, but they do like to pay for connections, e.g. communities. Tech makes people more lonely than ever, but it can also connect them to more people than ever. Your community can enable that!

The story of Nomad List's paid membership community

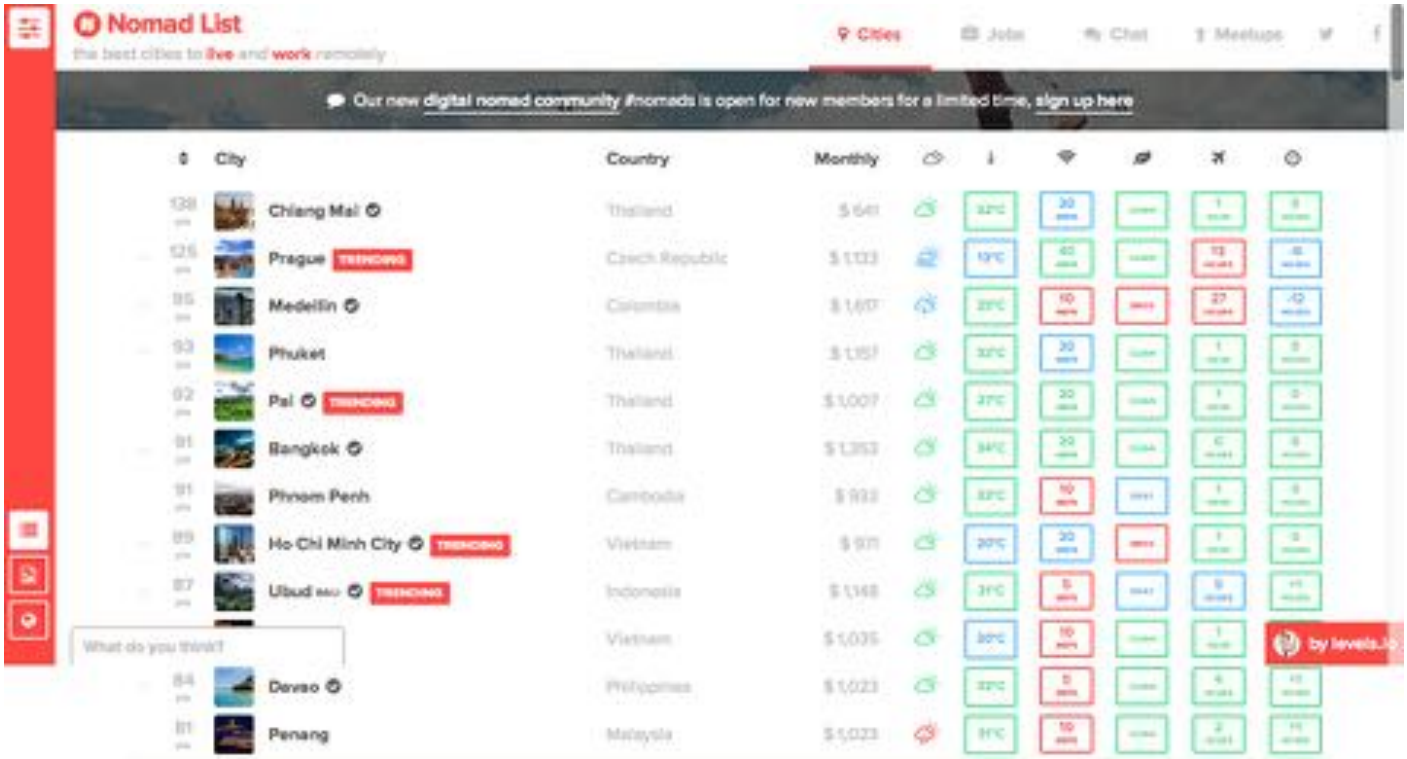
In my case I built a community around digital nomads and travelers. And I did it mostly accidentally.

I had launched **Nomad List** as a list of cities with some data on internet speed, cost of living and weather. It then went viral. But the site was very basic. I knew I was going to lose all these users because when you go viral, it only takes a few days or so and you're gone. I knew to make them stick I needed social features. But I couldn't code fast enough to build social features into my site (like signing up, logging in). I thought, "okay, let's just use other platforms for it, for now".

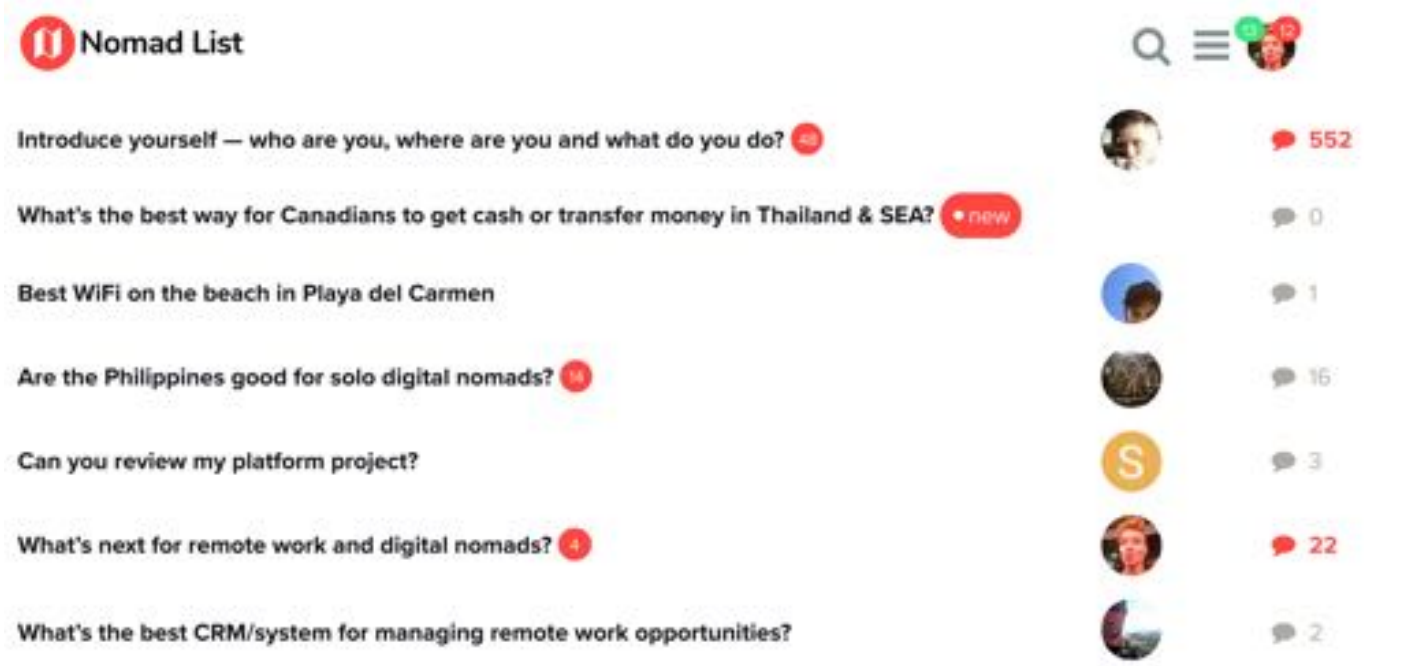
Back then, **Slack** was just taking off (~2014). I invited some of my friends to seed the community:



I also added a big banner to the top of the site so that people could sign up (for free back then):



A month later I added a discussion board (running on **Discourse**):



I could have never built a chat and forum myself this fast so it made sense to use off-the-shelf tools.

My trick had worked, the social stuff had made the site a bit more sticky and it was actually getting more popular! After a few months I started getting lots of spammers in. The quality of users decreased due to volume. So I thought, I'll

just start charging $1 to remove spammers. It worked, the spammers were gone but the quality of users was still not so high. I raised it to $5. Then $25. Every time I raised the price I'd see a drop of users for a week and then it'd go back up the same level of sign ups. I doubled it to $50. With 10 people signing up per day, suddenly I was making $500 per day, or $15,000 per month.

I had no idea what I just did. I wanted to stop spammers and increase the quality of user sign ups. I did that, but I also suddenly monetized my site and within days it became a profitable business!

Many people have followed this concept since and done the same.

You have a free basic app or site about a certain topic. People who visit your site probably have the purpose of taking action based on what's on your site. And because they're on the internet, they probably don't know a lot of people in your niche. You provide free content, attract people and then sell them a way to connect with people like them in your niche. By building a community around your site's niche you give them a valuable service, and you're able to monetize your audience instantly.

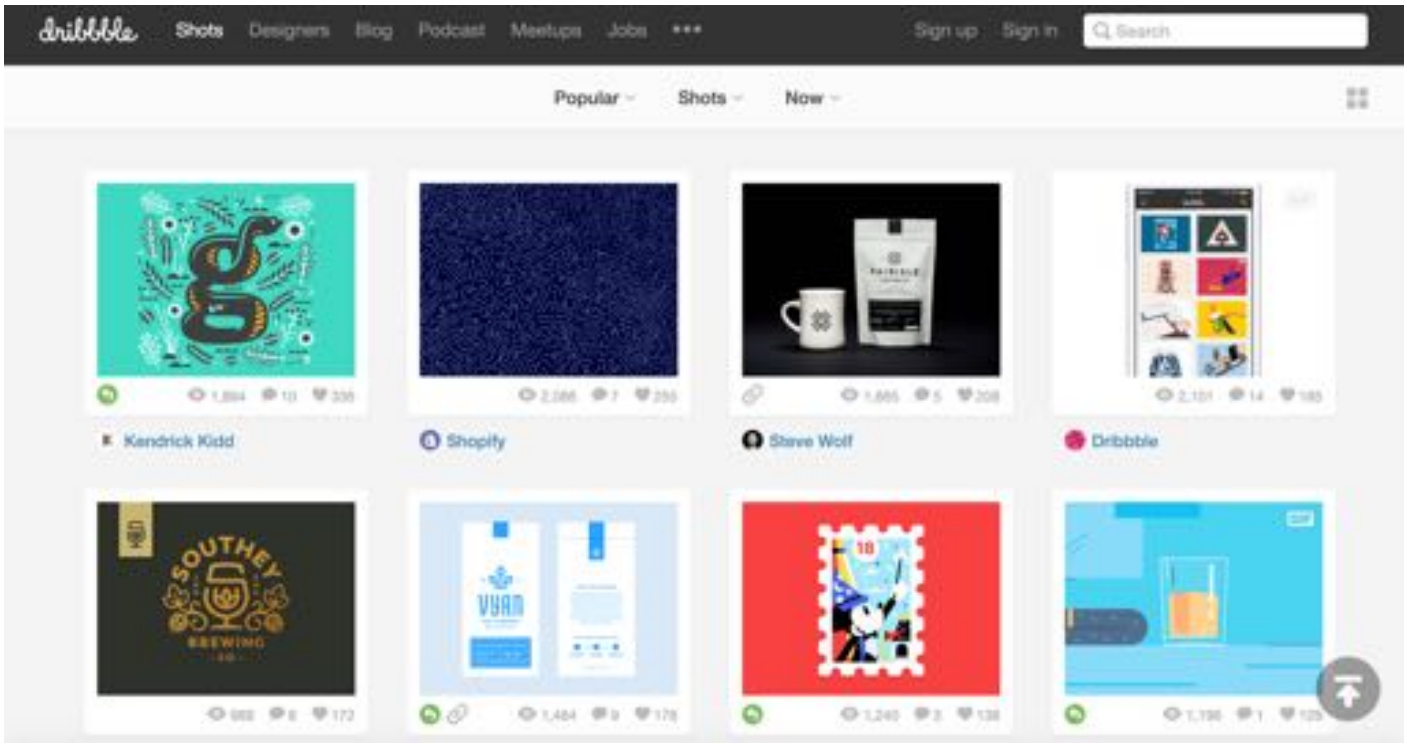
You can take this model and apply it to expats, illustrators, 3d modelers, coffee baristas, Apple product fans etc. I like this model because it's valuable for you (your free product gets monetized with a community) and for your users (they get connections out of it).

Job boards

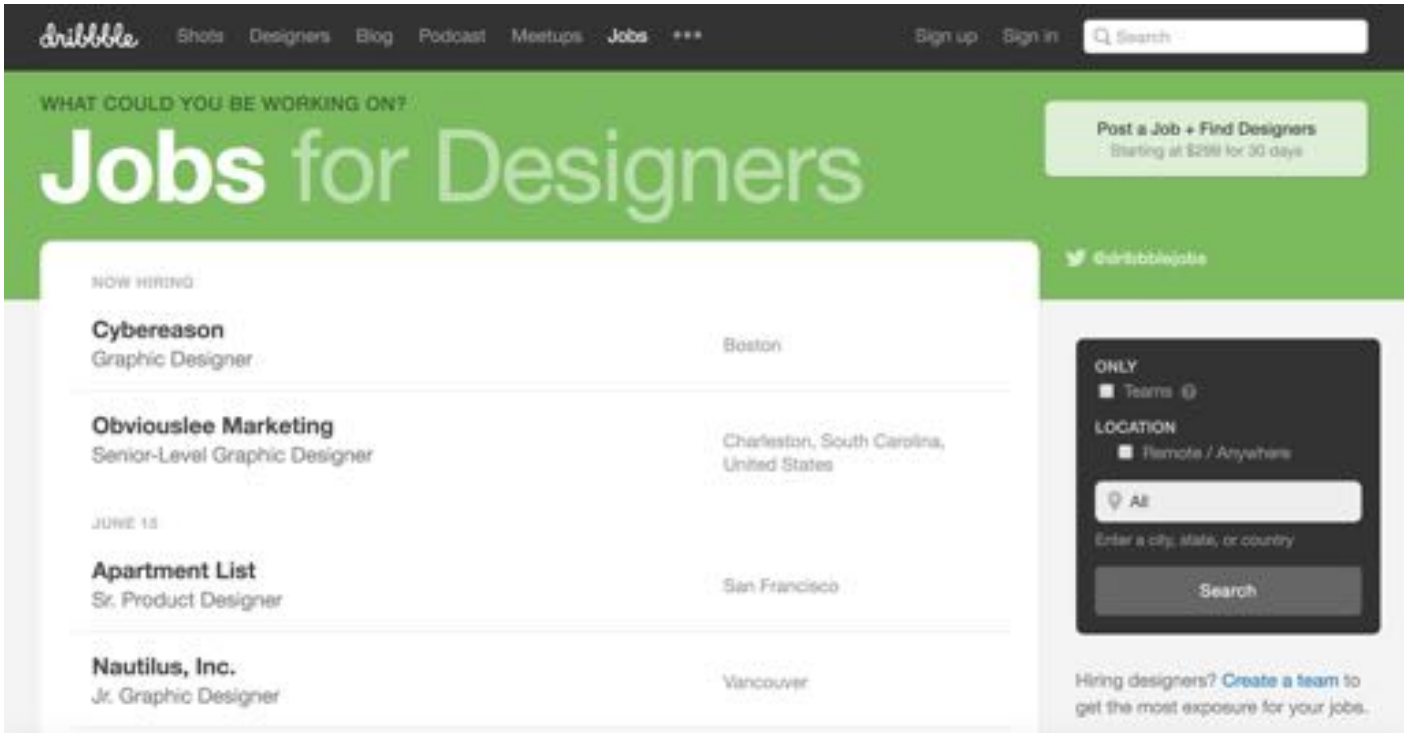
This is similar to the community model. You have a free site about a certain niche. Does that niche have its own industry with people working in it? Like designers, developers, baristas, expats, anything really. If so, one of your best ways to make money with it is to open a related job board besides it.

Job boards work because companies pay *a lot* to find the right talent to work for them. Especially tech companies. You charge companies to post a job, like $49 or $99, $199 or $299. If you get just 10 job posts at $299, you're making al-

most $3000/month. That's just 10 customers! Job boards are B2B so you need relatively few customers and high prices to become profitable.



A good example is **Dribbble**, which lets people share their creative design works in a show-and-tell-like platform. But they make zero money on this. All their money comes from their job board:



A job post costs $299 for 30 days on **Dribbble Jobs**. They get about 5 per day. That means they might be making $1,500/day or $45,000/month or $540,000/year.

Start to see the pattern? You build something that's free, which generates traffic, and then add something that's either valuable to your users or businesses that target your users and either one of them will pay for it. Which lets you keep your free site up, which in turn generates more traffic when you work on it.

Conditional payments

This is another interesting model that I tested myself.

With my app **Go Fucking Do It**, I asked people to set a goal, specify a deadline and enter their credit card details. If they did not reach their goal by the deadline, I charged their card (automatically).

This is interesting because users don't specifically pay for a service or good. They pay for failure. It uses money as a disciplinary tool.

I'm sure there are lots of ways you can evolve this model. One fitness app did this by adding GPS and charging you if you didn't show up at the gym every week.

So think of other ways how you can evolve this model. It simply means the user doesn't necessarily pay for a product or service but you use their money in a different more unique way.

Productizing an agency into a SaaS

This is not directly a monetization strategy but still important to discuss. Many startups started as creative agencies. That means they built a product for a single client, but then transformed that into a standalone product or service (usually a software a service or SaaS).

A great example is **Typeform**. It was started by **David Okuniev** who was working for a client at the time:

> Typeform was born out of a client project (..) [that] required us to build a lead generation form that would sit inside an exhibition space for a toilet company (..) we had to create something more inspiring to collect data than just a plain-old vanilla form, so we tried something completely different."

— David Okuniev, founder of Typeform on **Inbound.org**

An even bigger example is **Basecamp** (previously known as 37signals). They started as a simple web agency in 1999:

> "37signals started as a manifesto in 1999. We wanted to launch a web design firm that was focused on clean, fast, usable designs, and our manifesto was a series of statements covering our feelings about web design. In 1999 everyone else was elbowing for the loudest, brightest, most colorful, techiest "full service end-to-end" site. We went the opposite direction."

— Jonathan Kim on **Quora**

A decade later, they has so much experience working with clients that they started commonalities between the problems they were solving for them. They started building lots of small side projects to solve them. Most of them failed or were sold, but now their only product Basecamp remains. Some people now estimate their revenue at **$168 million**. That means a small web agency, productized their client's problems into a million dollar company.

An important side note here is to remember to make sure you own the rights to the work you make for a client, so that you can actually transform it into a

business later. Otherwise, they'll be the owners of your future million dollar company.

Learn from your competitors' business models

When you can't figure out any model that will let you make money, it's always a good idea to see how your competitors are making money then.

In general business theory, to compete with someone you either raise prices and sell a better product (called the premium strategy), or you lower prices, undercut them and sell an equal or lesser product but cheaper by reducing costs (called the low-cost strategy).

See how Android took 80% of the smartphone market with a low-cost product while Apple took the other 20% of the market with a premium product. Apple makes more money with only 20% of the market than Android smartphones with 80% of the market. Crazy? Yes! The point is, in most markets, there's room for both a low-cost and a premium product.

Keep experimenting with business models

One of the best things you can do is, and that I haven't seen many others do yet, is to continuously experiment with models to make money.

When launching a new product, I'll usually put a Buy button on it immediately. Just to see what happens. I'll make some part of the site exclusive to paying users, or offer a service to businesses to reach my users in some way.

And I'll keep putting those buttons everywhere. Because then I can see what people are willing to pay for. I'll test prices, raise them, lower them. You'll quickly see what works and what doesn't. And you can keep doing this perpetually to figure out any new monetization models that may appear.

When are you done monetizing?

This depends heavily on your objectives.

For many people, making just $2,000/month is a lot. I know it was a crazy lot of money when I started out. But you have to consider income tax, corporate tax, insurance and costs like that. It adds up. How much are you left with after costs and taxes? Maybe $1,200/month. Can you live off that? You'll probably need to make some more if you want to save money for the future too. A business is risky, it might work for a year and then stop working altogher because the market changes.

It depends on your objectives

It's also about how far you'd like to go. And your personal situation is as much a consideration as the potential of your business to go that far and scale up. Scaling up will take high resources from you too and will reduce the time you can spend with your lover, family and friends. It's always a trade off.

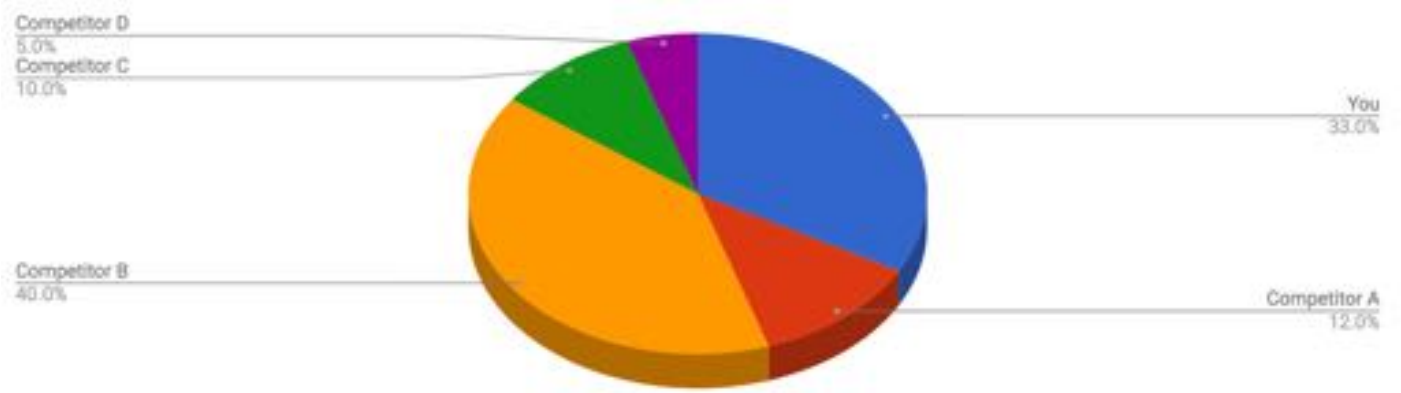
How big do you want to be?

Do you think your app should remain a side project, or do you think it should become a real business? Should it be a big company with 100s of employees? Do you want to manage that? Then you need a lot more revenue.

Look at your competitors. You're probably able to estimate how much money they're making roughly with some back-of-the-napkin math. Are they venture capital funded? If not, how many people do they employ? Make a calculation of how much it costs to employ those people. Add some overhead and you'll know a minimum of revenue they're making (if they're profitable, that is).

If you get the same amount of users or traffic as them, but you're making way less money, then there's obviously room for improvement in your monetization.

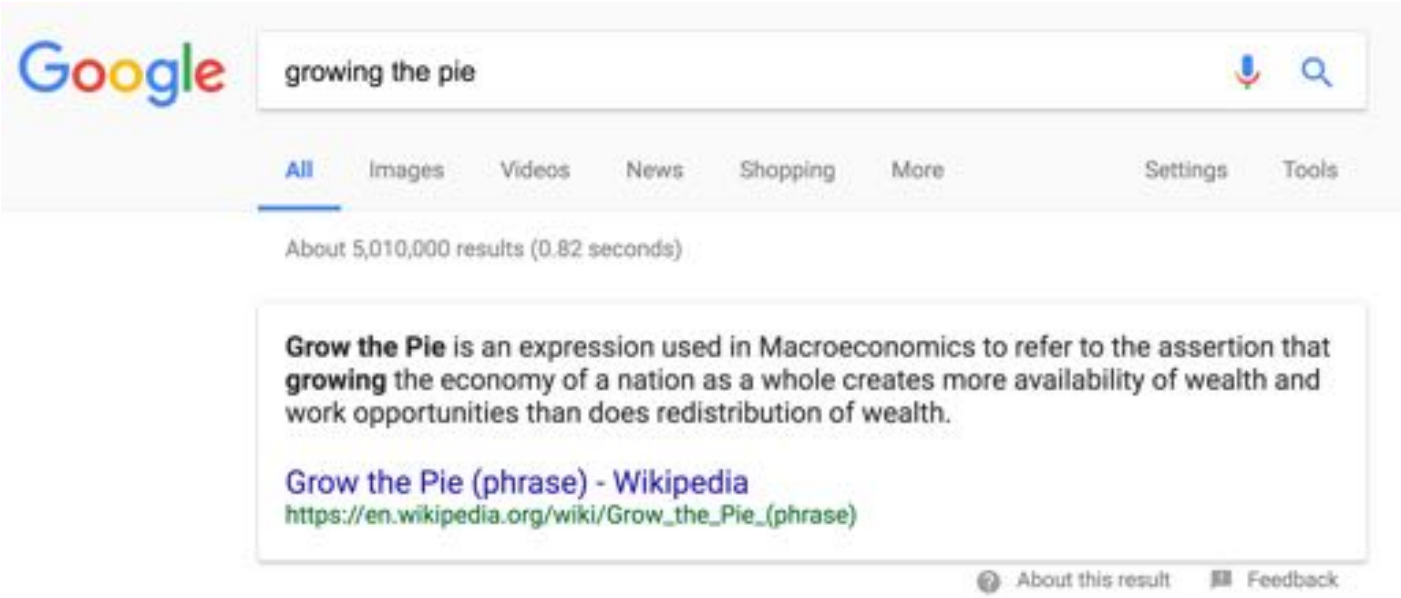
If you get less users or traffic than them, figure out where they're getting their users from. Google? Facebook? Referral traffic from certain websites? Map it out like a secret spy agent and see if you can compete with them in their own funnel.



Widen your market

If you think you've already captured most of the market and it's saturated, see how you can widen the market you sell to. Can you make your product broader and less niche? While **Nomad List** started as a site for digital nomads, now half the users are general travelers and expats. That market is 1000x bigger than the digital nomad market.

Grow the pie



Another way to get bigger is to literally grow the market you sell to. In macroeconomics we call this "growing the pie", but that's on a national level. It applies to an entire market too.

Most Airbnb users never rented vacation rentals before Airbnb arrived. Vacation rentals were a dusty old market for retirees. By promoting vacation

rentals for young hip people, suddenly everyone jumped in and started renting them. This didn't just help Airbnb, it helped their competitors like **HomeAway** too. They grew the entire market pie to be bigger by making people more aware of the industry, in turn, they were able to reach a lot more customers.

Payment platforms

To monetize a site, you'll need to accept payments from your users. Luckily, in the last few years there's been a lot of innovation in this industry. The dominant payment platforms (at time of writing) are Stripe, Braintree and PayPal. Braintree is now owned by PayPal.

It's always better to use a drop-in solution like these 3 than setting up some big merchant agreement with a payment gateway or bank for just your website. Even high-scale businesses like Uber and Airbnb use Braintree. Other companies like Lyft and Target use Stripe. And companies like Barnes & Noble or Abercrombie & Fitch use PayPal.

Stripe

Stripe revolutionized the online payment industry when it launched in 2011. For the firs time, it let people accept payments with an API that was simple

(unlike PayPal's). They gave websites a drop-in solution where you could just add some code and immediately accept payments. You didn't have to fax or post 12 contracts signing your life away to a merchant payment gateway company (how it used to go), it just worked. Developers loved Stripe (and still do).

Stripe is developer-first which means if you're building a website, app or whatever software, they'll have some pre-written code for you to build upon their API. One of their easiest products for newbies is called **Stripe Checkout** which lets you add a Pay button to your site which then opens a beautifully designed payment popup where people can enter their credit card and pay you. You then charge the user, get a signal back that payment went through and you can give the user access to your product.

Let's say you want to do subscription billing, and a user changes their subscription from $25/month to $100/year in the 12th day of the 3rd month they're a member. How do you calculate and pro-rate them? With Stripe you don't need to. They have it all worked out for you. You just tell them you sign a user up to a new subscription plan, then when they change it you tell Stripe, and they'll charge the right amounts. It's magic!

With Stripe, you don't even store credit cards anymore. Which makes you a target for hackers. They store them for you safely, all you save is a "token" which lets you charge them money that can only go to YOUR account (not the account of hackers). Instantly making it useless for hackers to hack you.

Depending on where you live, Stripe may or may not be available (yet). They're expanding worldwide though now. For remote workers it might be interesting to note they're available in Singapore, a popular place for people to register their companies when they're a nomad.

Braintree

Braintree

A PayPal Company

Around the same time that Stripe launched, it got a competitor called Braintree. It followed the same ideology. Make it simple and friendly for developers to implement their API. They are highly similar too.

The greatest benefit to use Braintree over Stripe is that Braintree is available in a lot more countries than Stripe. This is probably because they're now owned by PayPal which I think has banking licenses everywhere by now.

PayPal

PayPal is one of the oldest and most famous payment solutions on the web. And that's probably why it's despised by so many. Its technology is highly archaic, outdated, its API is badly documented and subject to unannounced

changes at any moment and until recently their site looked and functioned like an Egyptian tomb.

The great benefit of PayPal though is that is has very high conversion rates. That's because people usually don't have to enter their credit card information on your site but simply pay by logging into their PayPal account.

If you're North American, you think "why is this a big deal?". Well, most of the world doens't use credit cards like you do! For example, Dutch people use a weird customized debit card system called Cirrus. Latin Americans are culturally extremely afraid of using their credit card due to rampant fraud. And most Germans use cash or will pay by a bank wiring system called SOFORT. That's a lot of customers you won't reach.

PayPal enables those users to buy your product as they support most weird national payment systems in the world. And as people don't have to enter any payment details, it feels more secure to them. PayPal's buyer's protection usually favors buyers, which is bad for you, but great for users. All in, these factors increase the odds that people actually go through and pay on your site.

Recently, PayPal has started to rebuild their site, launched a modern app and are generally on a spree to quickly modernize after years of engineering stangancy. This is directly because of Stripe entering the market. To fend off Stripe, they acquired Stripe's biggest competitor: Braintree, which they're slowly integrating into PayPal too. That's all good news for us makers, as there'll be more payment platforms to choose from. They're slowly catching up again.

However, PayPal is not without its risks. Unlike Braintree and Stripe which automatically deposit your balance either monthly or weekly, PayPal by default keeps all money in your balance. And they've been known to shut down PayPal accounts with lots of cash in them when they suspect fraud. They'll lock down accounts for weeks or months. If you use PayPal, make sure you login at the end of every month, and manually deposit your balance to your bank account. Don't keep a high balance of money on PayPal, it's a business risk.

Local alternatives

At the time of writing, Stripe is available in America, Canada, Western Europe and Singapore. Braintree is available in a few more. PayPal is available in many more. But still, there's lots of countries where there's simply no international payment processor. In that case, you'll have to consider local alternatives. That means doing lots of Googling and research to figure out which is easiest (and cheapest) to set up. It will most certainly be more complicated and involve signing more paperwork than Stripe, Braintree or PayPal. But you have no choice.

Stripe Atlas

An alternative is to open a company in a locality that does support Stripe. Stripe itself introduced a service called **Atlas** which helps people from world-wide open a simple corporation in the United States and in turn lets them accept payments through that company. It might be worth investigating. Having a proper payment processor like Stripe will save you a lot of time in coding payment logic, refunding and dealing with credit card fraud.

Use a combination of platforms

Instead of picking Stripe, Braintree or PayPal, you can also use a combination of the services. On many of my sites I show a modal that asks users:

> [Pay by Credit Card]
>
> [Pay with PayPal]

When one of these buttons it's clicked, you link them to the appropriate platform. This gives your users more freedom to choose how to pay. And gives you a higher reach of potential sign ups.

When I added this, conversion went up about 40% from ~ $10,000 to $14,000 per month. Great! My bookkeeping become so difficult with PayPal that recently I switched solely to Stripe just to make my life simpler. That's the downside to it.

Use a Typeform to charge fast

What if you have an MVP and you don't have to time to set up Stripe, Braintree, PayPal or another platform's code on your site?

You don't need to! Use **Typeform**. I talked about Typeform in the chapters about building MVPs before, and it's very useful in this case too.

Typeform lets you connect your account to Stripe. That means you only need a Stripe account, you don't need to code anything. All you have to do is make a new form on Typeform, which asks the user for their name, email and credit card information.

If users enter and pay, Typeform handles everything and you'll directly see the money appear in your Stripe account.

Now if you have your MVP, you can simply add a Buy button that's actually a link to the Typeform form you build and you have instant monetization! If the monetization works (and your model is validated), you can always build your own payment integration with Stripe on your own site without the Typeform. But a Typeform is a good start when you don't have time or want to code something yourself. Especially if you're experimenting and validating what people would pay for.

How to deal with refunds?

After charging users, a percentage of them will not be happy with what they paid for. And it's usually a small, predictable and stable percentage.

The worst you can do when a customer asks for a refund is to not give it. You now have an angry customer. You don't want an angry customer, because they will talk about you on the internet. And they'll destroy your product's brand. That means you'll get less future customers.

When a customer asks for a refund, say sorry that it didn't work out (and mean it!). Then immediately refund them their money. Ask them why your product wasn't what they expected. This might be one of the most useful moments to ask for feedback as you might be able to avoid future refunds to other users. This is not a bad event, see a refund as a moment of critical feedback. It doesn't mean the customer will always be right, but it's important to listen. You'll learn more from your angry customers than your happy ones.

Another big reason to give a refund as quickly as possible is that these days users can request a chargeback to their bank (sometimes within their bank's app). This is the last thing you want. Chargebacks are payment disputes between you and the customer's bank. They give your company a bad reputation with your payment processor (like Stripe). And they cost money. A regular chargeback means the entire amount they paid is refunded and you pay a fine of $35 usually.

Now, as long as you provide a refund immediately, you just lose the money that they paid, but nothing more. The payment is already refunded and they can't initiate a chargeback. So it'll cost you less to refund! That's obviously why the banks have designed this system like this. It promotes refunds. Well, it's working!

How to deal with bookkeeping and tax?

If you've come as far that bookkeeping becomes a problem, you're probably doing well. Having thousands of transactions coming in from countries all over the world is challenging. Dealing with the right sales tax laws in states (like US) and countries (like Europe) is difficult.

I recommend if you're doing over $50,000/year revenue to pay up to $5,000 for a good accountant that understands technology, business and startups. Trust me, they're hard to find. Regardless of country, most accountants are old and behind. My advice is, approach 25 different accountants and make them do a quiz. Ask them what for example Stripe is, how to do sales tax payments in different countries and how they would help manage your thousands of annual transactions.

Not having an orderly bookkeeping can become a very big liability and burden if you ever get audited. In most countries fines are double or more of what you were supposed to pay if you didn't file properly. That can be a lot of money. Additionally, you'll be on a list at your governmental tax organization that you broke the rules before, so you'll probably be audited more now.

This is not a part of business you want to "save money" on. Pay a bookkeeper well! You need them to operate legally in the complicated environment that having an internet presence where you sell worldwide is.

Where possible, try to automate your side of the bookkeeping before it arrives at your accountant. That means, set up automated web hooks from your payment processor (like Stripe) to sync transactions automatically to a Google spreadsheet or database. Don't forget to also sync stuff like refunds. You want a ledger of all transactions that is always up to date. For expenses, you'll want to save every single invoice you pay to a Dropbox folder with the filename format

> ./Expenses/YYYY-MM-DD - Company - Description - USD x,xxx.PDF

Try to limit your income and expenses to as few mediums as possible to make your bookkeeping more simple. I only get money via Stripe now and I spend it via PayPal or my business bank account card. That means I only need to dump 3 data sources to get 100% coverage of my transactions. Remember, it's hard to get your bookkeeping perfectly accurate. You have an accountant to help you

with that and see if everything is done properly. Keeping your finances simple should be priority.

Conclusion

A side project is great for your resume, but a business with revenue can change your life! If it works out, it means you can quit your job. You can hire people (or robots). Get an office (or work remotely from anwyhere). It means you might be able to build a movement out of your vision. And that can all start with a small little project. Everyone starts small. If you grow or not often depends if you can make money and feed yourself (and the company).

Money is like rain on plant seeds. It gives it nutrition to grow. Money makes it possible to keep working on your project and pay your rent. If you don't make money, your project will stay small, and you'll be bound to stop working on it and quit.

If you'd like your project to become big, please, focus on monetization. Remember:

> An app without monetization is a charity
>
> An app with monetization is a business

Resources mentioned

- Patreon
- Overcast
- Slack
- Discourse
- Dribbble Jobs
- Typeform

Your homework

- From the business models mentioned in this chapter, make a list of which ones might work for your product.
- Pick one to start with and implement it. See if you're getting payments after a few days. Keep adjusting until the model works. If you're certain this model doesn't work for you product, ditch it and try a different one from here, repeat until you're generating revenue.

Automate

Introduction

If you've come this far, you've now build out an idea, launched it successfully, grew it to reasonable userbase that pays you money every month or year. We'll now see how we can remove you from the equation. How can we make this business run itself with just robots (and some contractors)?

How is automation relevant?

You have money coming in but probably still need to do maintenance because things randomly break (they always do!). You might also still be adding features every week. But at some point your app might be feature-complete for most users. It does most of what it should do (there's always more but let's set a limit). What's next then?

What's next is a personal decision as much as a business one. You probably have been working on this product now for months or probably years. A few years is a long time in a human life. We're more than our single business. You might want to start other ideas. You might want to invest more in your personal life: friends, family, relationships etc. That all makes a lot of sense.

Here's where many people would say: "go hire some people to work for you". Here's where I say: "avoid hiring, build robots". Hiring increasing the complexity of your product, business and life. Hiring a person means you need train and manage them and makes you liable for their income and in many countries a lot more than just that (e.g. health insurance). Humans are complex. They're also relatively slow. Robots can be simple. They're also very fast. Most of the regular stuff to maintain your product can be automated as a scheduled computer script (run by scheduled cron jobs on your server). Those scripts I like to call "robots". I have about 700 to 2,000 running depending on server load. They're such an important part of my business I list them on my team page:

You'd be surprised what robots can do. Here's a few things my robots do for me every second of the day:

- Find cities that are popular on **Nomad List** in certain days, then find a good Foursquare bar in the hip district and automatically organize meetups for them on a Thursday, Friday or Saturday and promote these meetups on Twitter, Facebook and Slack 1 month before, 14 days before and a day before.
- Find the weather, internet speed, air quality from 25 different sources for 1,000 cities every hour of the day
- Check if the server is completely up-to-date, if not, automatically update if possible.
- Notify me by SMS when anything breaks or doesn't work perfectly, like the server using too much CPU, disk space, or showing lots of errors.
- Go on Tinder (as a user), and see the most popular cities there every week and add them to the database so that people can filter cities on that on Nomad List.

- Browse 100+ job boards for jobs that can be done remotely, and parse the job position, description and company to see if it's a remote job or not, if it is, post it to **Remote OK** and share it on Twitter and Facebook and send it out to people subscribing to email alerts for that specific job tag (like JavaScript).

- Browse my city pages and screenshot part of them to make city thumbnails so that if people share a page on Twitter or Facebook it shows a picture of the city with data printed on it.

- Check every day if there are deadlines passed on GoFuckingDoIt.com and email the supervisor of the goal set if the person really passed their goal, give them a link to click if they didn't pass the goal which if pressed lets the robot charge their creditcard for money.

- More personal: Login to my smartphone telecom provider, downlaod the invoice, print is as PDF and email it to me. Send a copy to my spreadsheet robot who automatically enters it into my bookkeeping.

What's a robot really?

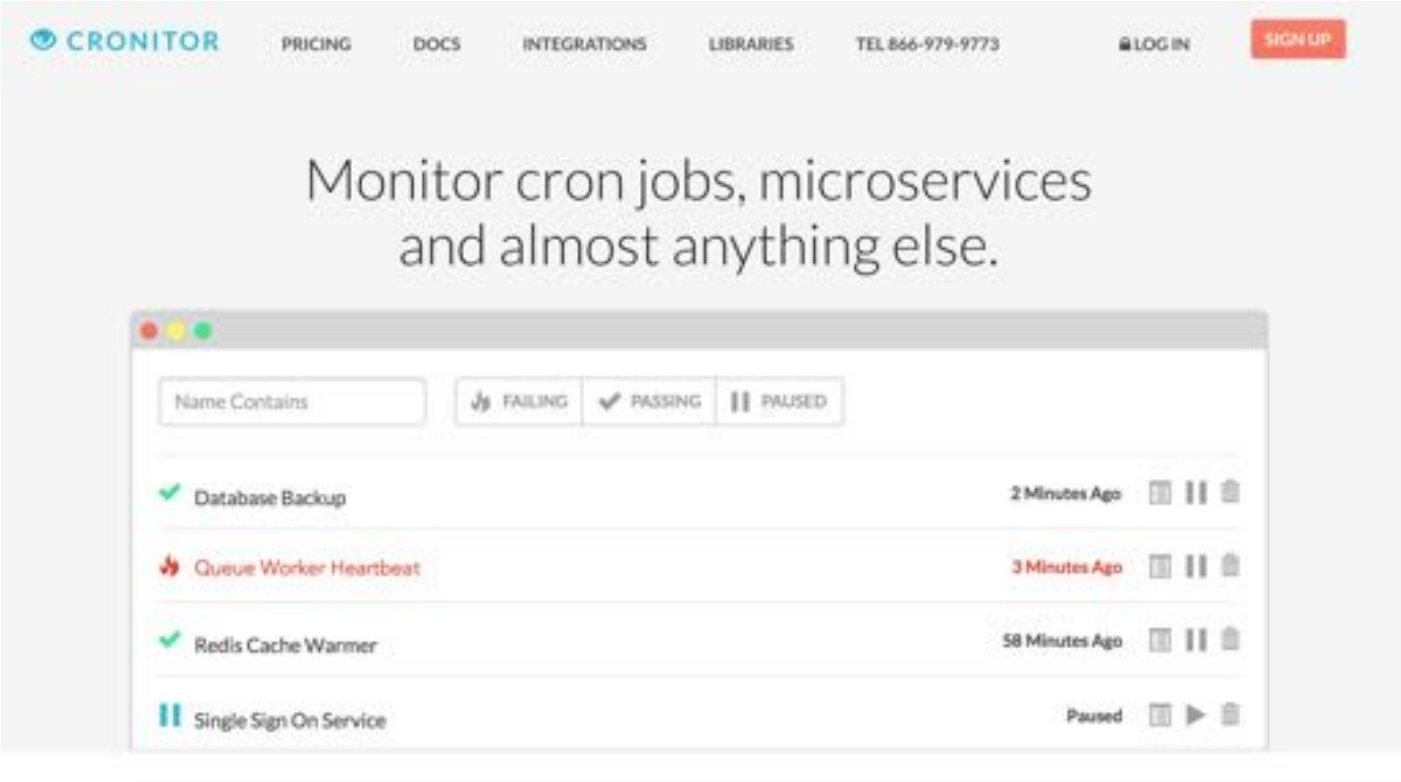
A robot for me simply means a task I used to do myself, that I wrote a script for, then scheduled to run every second, minute, hour, day or week. My robots are written usually as simple PHP scripts that are run in the shell, but I also sometimes use JavaScript on the server for it (with ExpressJS or PhantomJS). It doesn't really matter how you do it. If you can make a script that does something and saves you from doing that task, that's a robot. Here's my server dashboard (called cron) with some of my robots scheduled (the /srv/cronlogger is simply an app to log the result of the scheduled job, the stuff after php is actually the script being run):

```
*/10 * * * * /srv/cronlogger rinsefm-podcast "php -f /srv/http/etc.levels.io/rinsefmpodcast.php -- 'refreshCache=true'"
0 0 1 * *    /srv/cronlogger simyorobot "php -f /srv/workers/simyorobot/simyorobot.php"
@hourly      /srv/cronlogger gfdi-deadlinespassed    "php -f /srv/http/gofuckingdoit.com/workers/checkDeadlinesPassed.php"
@hourly      /srv/cronlogger kyfr-deadlinespassed    "php -f /srv/http/keepyourfuckingresolutions.com/workers/checkDeadlinesPassed.php"
@daily       /srv/cronlogger gfdi-deadlinesexpired   "php -f /srv/http/gofuckingdoit.com/workers/checkDeadlinesExpired.php"
@daily       /srv/cronlogger kyfr-deadlinesexpired   "php -f /srv/http/keepyourfuckingresolutions.com/workers/checkDeadlinesExpired.php"
@daily       /srv/cronlogger gfdi-staticpage         "php -f /srv/http/gofuckingdoit.com/app/index.php > /srv/http/gofuckingdoit.com/public/$

# REMOTEOK
@hourly sh /srv/http/remoteok.io/workers/sources/run.sh
@hourly         /srv/cronlogger remoteokio-staticpage       "php -f /srv/http/remoteok.io/app/index.php -- 'HTTP_HOST=remoteok.io' 'page=fro$
@daily          /srv/cronlogger remoteokio-retagjobs        "php -f /srv/http/remoteok.io/tools/retagAllJobs.php"
@daily          /srv/cronlogger remoteokio-downloadpics     "php -f /srv/http/remoteok.io/tools/downloadPicsForAllJobs.php"
30 * * * *      /srv/cronlogger remoteokio-queuemails       "php -f /srv/http/remoteok.io/workers/queueMails.php"
0 */6 * * *     /srv/cronlogger remoteokio-sendmails        "php -f /srv/http/remoteok.io/workers/sendMails.php"
*/4 * * * *     /srv/cronlogger remoteokio-twitterfirehose  "php -f /srv/http/remoteok.io/workers/twitterFirehose.php"

# NOMADLIST
@hourly phantomjs  --ignore-ssl-errors=true --ssl-protocol=tlsv1 /srv/http/nomadlist.com/workers/getApiKeyFromSlack.js
@hourly /srv/cronlogger nomadlist-setcurrentlocation php -f /srv/http/nomadlist.com/workers/setCurrentLocationForUsers.php"
@hourly /srv/cronlogger nomadlist-preparecitydata "php -f /srv/http/nomadlist.com/workers/prepareCityData.php"
@hourly /srv/cronlogger nomadlist-nomadslackuserdata "php -f /srv/http/nomadlist.com/workers/syncUserSlackData.php"
@hourly /srv/cronlogger nomadlist-postslacktrips "php -f /srv/http/nomadlist.com/workers/postTripsToSlack.php"
@hourly /srv/cronlogger nomadlist-airqualityaq "php -f /srv/http/nomadlist.com/sources/air_quality_openaq.php"
@hourly /srv/cronlogger nomadlist-scaleuserpics "php -f /srv/http/nomadlist.com/workers/scaleUserPics.php"
@hourly /srv/cronlogger nomadlist-tripstweet "php -f /srv/http/nomadlist.com/workers/postTripsToSocial.php"
@hourly /srv/cronlogger nomadlist-tripnotifications "php -f /srv/http/nomadlist.com/workers/emailTripNotifications.php"
@hourly /srv/cronlogger nomadlist-getfbmeetups "php -f /srv/http/nomadlist.com/workers/getMeetupsFromFacebook.php"
@hourly /srv/cronlogger nomadlist-exchangerates "php -f /srv/http/nomadlist.com/sources/exchangeRates.php"
@daily /srv/cronlogger nomadlist-joincache "php -f /srv/http/nomadlist.com/app/join.php -- 'refreshCache=true'"
@daily /srv/cronlogger nomadlist-synctwitterdata "php -f /srv/http/nomadlist.com/workers/syncUserTwitterData.php"
@daily /srv/cronlogger nomadlist-maketripmaps "php -f /srv/http/nomadlist.com/workers/makeTripMaps.php"
```

There's even services that will monitor if your scheduled cron jobs actually run like **Cronitor**:



Don't automate if it's not worth to automate it

As much as I believe robots are quite unlimited, how much effort you want to put in to make them advanced like humans is your decision based on how

much time you have. There's a famous trope of programmers automating everything but then programming the automation script takes longer than the time it saves. That's bad! You want to automate parts that take quite some time for a human, but don't take too much time to script.

Where do humans fit in here?

For the things that take too much time to program automation for, it might be useful to just get a human to do it. In my case, there were a few things I really couldn't automate away. I've made most of my customer support automatic: a user can sign up, cancel their account, get a refund, change their subscription plan, change their credit card etc. all with a self-help dashboard. But sometimes you have users falling in between the ship and the land (that's a Dutch expression). For these edge cases you want a human on standby to solve their issue. This is how a typical white hipster human looks in 2017:

This human needs basic skills like communicating with a customer, going into your users database to fix things and if they really can't resolve it contact you to solve the underlying problem. If you have enough stuff happening, you can also pay them a monthly lump sum like $2,500/m (that's what I do with my customer support) and just be done with it. I don't like to manage humans, so I like to work with people who are highly autonomous.

Another way is paying a human to be somewhat on standby based on when something happens and they have to deal with it and they log their hours and just invoice you through PayPal: that's what I do with my developer on stand-by. They get an alert when something doesn't work, and if they fix it first (before me), they invoice me for the hour worked (I pay $50/hour and up).

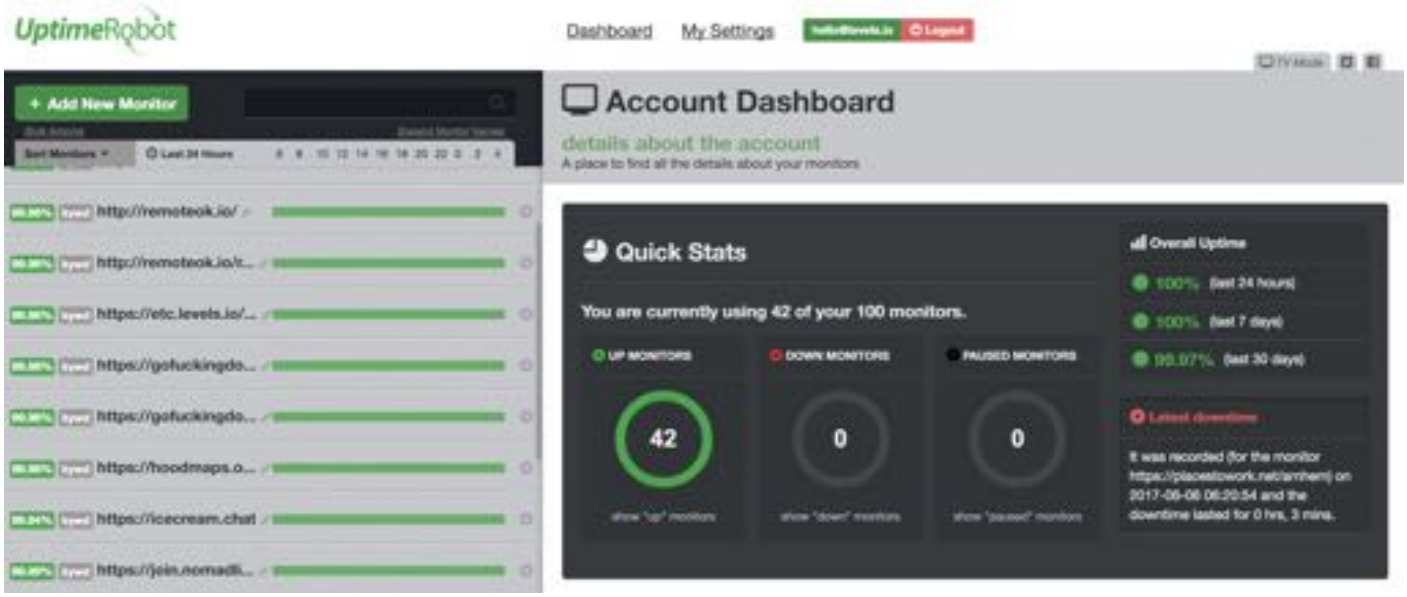
You're laughing about how I talk like humans as just modules in your business. But anyone who portrays they're not is lying. If the goal of your business is revenue, then a human equals a robot. You put energy/money in it, and it gives work as output. Easy. (the social implications of humans being used as modules and becoming unemployed by automation will require basic income which I'm a big proponent of, but that's a story for another book).

Where do you find humans? I usually hire my friends as temporary contractors, but you can also try sites like **Upwork**.

What if the robots can't fix things themselves. How do the robots ask the humans for help?

If you have humans on standby, how do you make sure the robots can figure out something is not right and call in human support troops? There's lots of ways. You can build your own robots that check if things are running properly. For example, a script that opens your website, checks for a certain keyword and if it's not there (which means the site is broken maybe?) emails you.

What if you want it to alert you by SMS? Twitter DM? Facebook? There's a fabulous app called **UptimeRobot.com** that does just that. Here's my dashboard there:



These are all little alerts that check something every minute or hour depending on importance. Like one alert visits the page on **Nomad List** for "safe cities in Amsterdam with good health care". I know that Amsterdam should probably be on there, so if Amsterdam is missing it means the filters are broken or something more severe is going on (like the page not loading).

So this is like passive income, right? No!

By fully automating the parts that are worth to automate and getting contractors to do the stuff that are not worth to or can't automate, you now have built a machine that just runs and pays out money without you having to work on it. People would call this the magic "passive income" but I think that's a ridiculous term. You probably had to work 3 years on a crazy hardcore level to get to the point where you can automate and not work and make money. Then the question is how long will that last? Maybe another 3 years? So it's more like the first 3 years you just worked as hard as someone working 6 years normally. That's not passive income, that's just compressed income. Passive income is a myth. You can minimze work a lot, but until you sell your company, there will probably always be at least monthly situations where you need to step in.

If you've came this far with your side project, which is now a real business, it means you're near to having an autonomous organization. With Bitcoin-type blockchain technology like **Ethereum**, we can build so-called DAO's (or De-

centralizd Autonomous Organizations). This is very new tech but the philosophical concept behind it is very interesting. You have companies that run virtually in the cloud, are owned by people that can trade stock in them and can get paid out dividends. Just like real publically traded stocks.

The "bus test"

Going completely decentralized is hard and I don't completely know how to that yet. But I do know the way towards an almost fully autonomous organization. You'll need a mix of robots and humans to run it. If this works it means you pass the "bus test":

> "A thought experiment which explores the impact of losing a person: If a particularly empowered individual in an organization is hit by a bus, will the organization suffer greatly? If yes, fail. If no, pass."

— **Urban Dictionary: "bus test"**

Ask your self, would you pass the bus test? Transforming your product into an autonomous organization means you will.

> Here's a rough set up to make your product into an autonomous organization:
>
> - Automate repetitive work with robot scripts
> - Hire a dev ops (or sys admin) person on a contract-basis (preferably hire 2 so you have a backup). They'll be the manager of the bots, kind of

- Set up alerts (like with Uptime Robot) so your devs are alerted if your product breaks
- Hire one part-time or full-time contractor who has access to your PayPal or bank and can manage and pay your devs (doesn't have to be technical necessarily). They're like the executive.
- Find one or two people that can be somewhat like "the board". They check if the executive runs your company as you would have wished.
- If you'd like, you can prepay balance to your hosting company (as I do with Linode) and domain name company (as I do with NameCheap), to prepay the costs for a few years. This will likely mean depositing $1,000's with them, but it means you're sure keeping your product alive isn't dependant on your credit cards existing.

Conclusion

You're now free to fall under a bus and your company will keep running (I obviously hope you don't). Even better, if you don't fall under a bus you now have a lot of free time to do other stuff! You've reached the holy grail of building a product: nearly passive income.

Nearly, because again, passive income doesn't really exist. There will always be small things. But luckily, you can get other people to manage most things for you. And in turn they can get bots to manage most things for them. Only in the most extreme cases, you're alerted for issues. That means you can focus on

spending more leisure time with your friends and family. Or making new projects and bankrolling them with the cashflow from this project.

The benefits of a clear mind are underestimated. Running a business is stressful. An entrepreneur's mind generally doesn't stop after work hours. New ideas, new solutions to challenges, they'll come up any time. That means they're not the nicest people to be with. But automation gives us a chance here to change that. Stepping away from the business when it's mature and having robots run it is one option. And it's an option I chose, and I think will be increasingly the norm in the future. Automation seems very, very obvious.

Resources mentioned

- Cron
- Uptime Robot
- Upwork
- Decentralized Autonomous Organizations
- Ethereum

Your homework

- From the business models mentioned in this chapter, make a list of which ones might work for your product.
- Pick one to start with and implement it. See if you're getting payments after a few days. Keep adjusting until the model works. If you're certain this model doesn't work for you product, ditch it and try a different one from here, repeat until you're generating revenue.