Tracking Personal Finances using Python

Learn how to build a developer-friendly workflow to keep track of your money

Siddhant Goel

Tracking Personal Finances using Python

Siddhant Goel

ISBN-13: 978-3-9824543-2-0

The information contained within this book is strictly for educational purposes. Apply ideas contained in this book at your own risk. Your results are likely to differ than those of the author.

We do not guarantee that you will make any money using the methods and ideas in this book. Any examples in this book should not to be interpreted as a promise or guarantee of earnings.

We have made every effort to ensure the accuracy of the information in this book at time of publication. However, we do not guarantee that all of the information in this book is correct or up-to-date. Therefore, we disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from information in this book, negligence, or any other cause.

Introduction

I've been meaning to write this book for a long time now. In general, I'm the kind of person that obsesses about organizing and keeping track of things. The one area where this habit has given me the best return on investment by far is keeping track of personal finances.

About three years ago, I started working on setting up a personal finance system of my own. I didn't have lofty goals. All I was looking for was a way to keep an eye on the general state of things. I started looking into different alternatives and spent some time trying to set up a system to use myself. After some trials, I found something that fit my personal preferences. After spending some time with it, I started showing it to a few friends and presenting at local meetups and didn't get completely shot down. I took that as a sign that this is something that could be interesting to a few more people than just me. Since then, I've managed to bring this system to a point where handling my finances is something I would actually call fun.

So in the rest of this book, I'll describe what I use, the workflow and the tools I use to manage my finances.

I'll start with defining a few basic concepts which are crucial to understanding the rest of the text. After that bit of background is out of the way, I'll get to the "workflow" side of things and the different tools that support this workflow. I will also try my best to arm you with as much information as possible so that you don't run into the same walls as I did.

Background

About three years ago, I started thinking seriously about keeping track of personal finances. Looking back, I don't think there was any one defining moment that triggered that chain of thought. It was just the constant nagging feeling of not knowing where my money was. I didn't know what my spending patterns looked like, how much of my spending was recurring, how much of it was out of the blue, things like that.

These sorts of questions went unanswered, which made me increasingly uneasy.

At that time I had three bank accounts, each designated for a specific purpose. I had a personal account for my own use, one for shared use with my wife, and one dedicated to earnings from any side

projects. Similar to most people, I also had a few recurring expenses (rent, groceries, public transport, etc.). And all these expenses were split across the three different accounts. I also had taken out a small loan the year before and started investing small sums of money regularly into ETFs[1].

While the situation was not that straight-forward, at least to me, the problem was quite clear.

I was looking for a way to keep track of all my assets across all my bank accounts.

As it turns out, this is not a new problem at all. There are plenty of solutions available on the market which solve this issue. But none of the available solutions fit the criteria I was looking for except for one.

Requirements

From the beginning, I had a strict set of criteria that I wanted any potential solution to satisfy.

1. Self-hosted

With the advent of SaaS applications, everything moving to the cloud, there's a certain elegance in software that does not require any network connectivity to run.

To be clear, I don't think everything needs to be self-hosted. At times, there is a lot of value in letting someone else handle the infrastructure side of things. And for a lot of the applications we use on a day-to-day basis, the self-hosting attribute might indeed qualify as a nice-to-have. However, for the class of software that does anything with money, I personally consider self-hosting a necessity.

This becomes particularly important given the requirement of being able to track money *across all my bank accounts*. This, by definition, meant that whatever software I would end up using would have hosted all my financial data, historical data included. If you're anything like me, the thought of voluntarily putting all such data in one location where the hosting is controlled by someone else should make you uneasy.

A lot of the software solutions on the market have strict privacy policies that specify in a lot of detail what they're allowed to use the data for. While I generally appreciate this, I've found that such policies often tend to be a bit vague and open-ended. Besides, the language tends to be legalese which is a bit tricky for my developer mind to fully understand.

Given all these constraints, I consider self-hosting an absolute must-have when it comes to money-related software.

[1] https://en.wikipedia.org/wiki/Exchange-traded_fund

2. Open Source

I also wanted the software to be open-source.

Not necessarily because I wanted to read all the source code and verify everything for myself. But because when something is in the public domain, you can rely on people much smarter than yourself and trust that everything is going to be audited. Limiting my search to only open-source solutions meant that I could rely on other people much smarter than I am to have used and verified such solutions.

A second advantage of using something open-source is that if you run into bugs or want to have a new feature, you can consider building it yourself and contributing it upstream. That is simply not possible with closed-source and proprietary software. And as we'll later see, this requirement was very useful when working on my own system.

I would argue that open-source applications tend to fare well on longevity, as long as there's a community behind them. There are always going to be exceptions. But if there's a group of people behind a piece of software, there's a reasonable chance that if the primary maintainer(s) were to step down, someone else might take the lead.

3. Simple

The final requirement I had was that I wanted things to be simple.

One might argue that this requirement is subjective and that almost everyone will have a different definition of the word simple. My personal definition was based on three different criteria.

3.1 Ability to work with Plain Text

As a software developer, I've come to love and appreciate the power of plain text. The beauty of it lies in the fact that there's nothing special about it. It's literally just a bunch of plain characters in a text file. There are no formatting options, no bells, no whistles, just you and your content.

It's likely the simplest data format you can use, and that's what makes it so powerful. Given its power, I wanted to incorporate it in my workflow from the beginning.

3.2 Easy to understand and explain

> If you can't explain it simply, you don't understand it well enough.

Too often, we end up using applications or workflows that bring in their own complexity. Sometimes the complexity is just inherent to the problem[2]. But there certainly are cases where it could be avoided.

I wanted my final workflow to be easy to understand and explain. If I would not be able to explain it to someone else easily, how could I hope to continue using it long term?

3.3 Not too many moving parts

The developer in me loves software that does not have too many moving parts. If I have to deploy a system in production, and I have two choices - one with two components and the other one with three, I'm much more likely to go for the option with two.

Having been a part of the software industry for slightly more than a decade now, that is something that I'm deeply convinced of. All else being equal, the less moving parts something has, the easier it's going to be to maintain over time. At this point, you might consider this an oversimplification because there are so many other factors at play here. Generally speaking, I've found that the practice of sticking to systems with fewer moving parts has been helpful overall.

Working with money in software opens up the potential for things to go wrong. Another requirement was that whatever I ended up using should have the fewest moving parts possible.

[2] https://en.wikipedia.org/wiki/No_Silver_Bullet