Product Build: Software for Secondaries

A little under a year ago, I helped a team of financiers begin working on facilitating secondaries in private tech companies: they would buy shares from early investors looking for liquidity, and sell them to later stage funds looking for exposure to startups. It's called a secondary because no new shares are issued: people who already own part of the company sell that part to other people.

They wanted a set of tools to help them manage this process, and tasked me as the Product Manager for the operation.

North Star and Vision

PRODUCT VISION

The core user need was to retrieve specific sets of information, and the pain point was the sheer difficulty of retrieving it. It's not easy to find the cap table (investors list) of private companies: that data is distributed among multiple platforms, data is often inaccurate or outdated, and there are very few APIs available for programmatic consumption.

Our vision was to shield users from the complicated process of cap table data collection, and make data retrieval as easy as inputting a company name. Throughout feedback and iterations we tried to keep this in mind as our goal.

We were essentially building an interface that collected messy data and made it easy to consume.

USER NEEDS

We had one major problem: our small set of end users was distinctly non-technical. They barely used email, and felt uncomfortable navigating a modern frontend UI. That's why I decided on building a text based interface (a Slack bot, to be more specific) instead of a full stack app. It's also easier to build, so that's nice.

Problems and Iterations

The initial MVP that I wanted to build hit some hiccups very early. Two out of the three data sources that we assumed we could rely on ended up not working out: Crunchbase's sales team had disappeared, and Pitchbook didn't offer the API we needed. So I started with a basic AngelList scraper that looked something like this:

```
#Parse past investors HTML
r = requests.get(investorsUrl)
c = r.content
content = json.loads(c)

#With the HTML in check, it's time to parse it and split it into the data structure we want
#The data format as is is an investor, followed by their roles on AngelList
#We need to separate each set into an investor and their roles so we can parse them

#Split all investors into a list with their AngelList roles
investorRoles = []
roles = []

for item in content[u'startup_roles/startup_profile']:
    soup = BeautifulSoup(item['html'], 'html.parser')
    para = soup.find_all('a')
    for name in para:
        if name.text == u'':
            investorRoles.append(roles)
            roles = []
        else:
            roles.append(name.text)
#Remove the initial blank space
investorRoles.remove(investorRoles[0])
```

There was a bunch more code involved because the data was so dirty, but you can get the idea. From here, I tried to improve data quality, make sure it worked every time, get it running on a server, and create the Slack interface. Here are some of the problems I ran into along the way:

DATA INTEGRITY

I failed to anticipate major data integrity issues that hampered product quality:

Problem	How we addressed it
AngelList ended up being the only reliable data source out there	Developed a relationship with an executive at AngelList, and adjusted expectations
Even once we got it, cap table data was often unreliable and outdated	Adjusted user expectations, and incorporated some human data cleaning
Users couldn't say for sure what data was or wasn't valuable to them	Created a structure to identify investor rounds, giving more information

These were the most frustrating and core problems we encountered. With the resources we had (just me as a coder!), it was difficult to keep the initial product vision in mind, and we realized that we would have needed some more beef on the engineering side to design the perfect API.

DATA-DRIVEN FEEDBACK

There wasn't much data available for this project, but we were able to creatively maneuver to one point: we got connected to someone with access to much cleaner (and proprietary) data.

> He told us that the data we were grabbing was around 75% accurate when compared to his proprietary, verified set.

After getting that feedback, I was able to effectively communicate our accuracy expectations to users, and set a benchmark for improvement.

USER EXPERIENCE

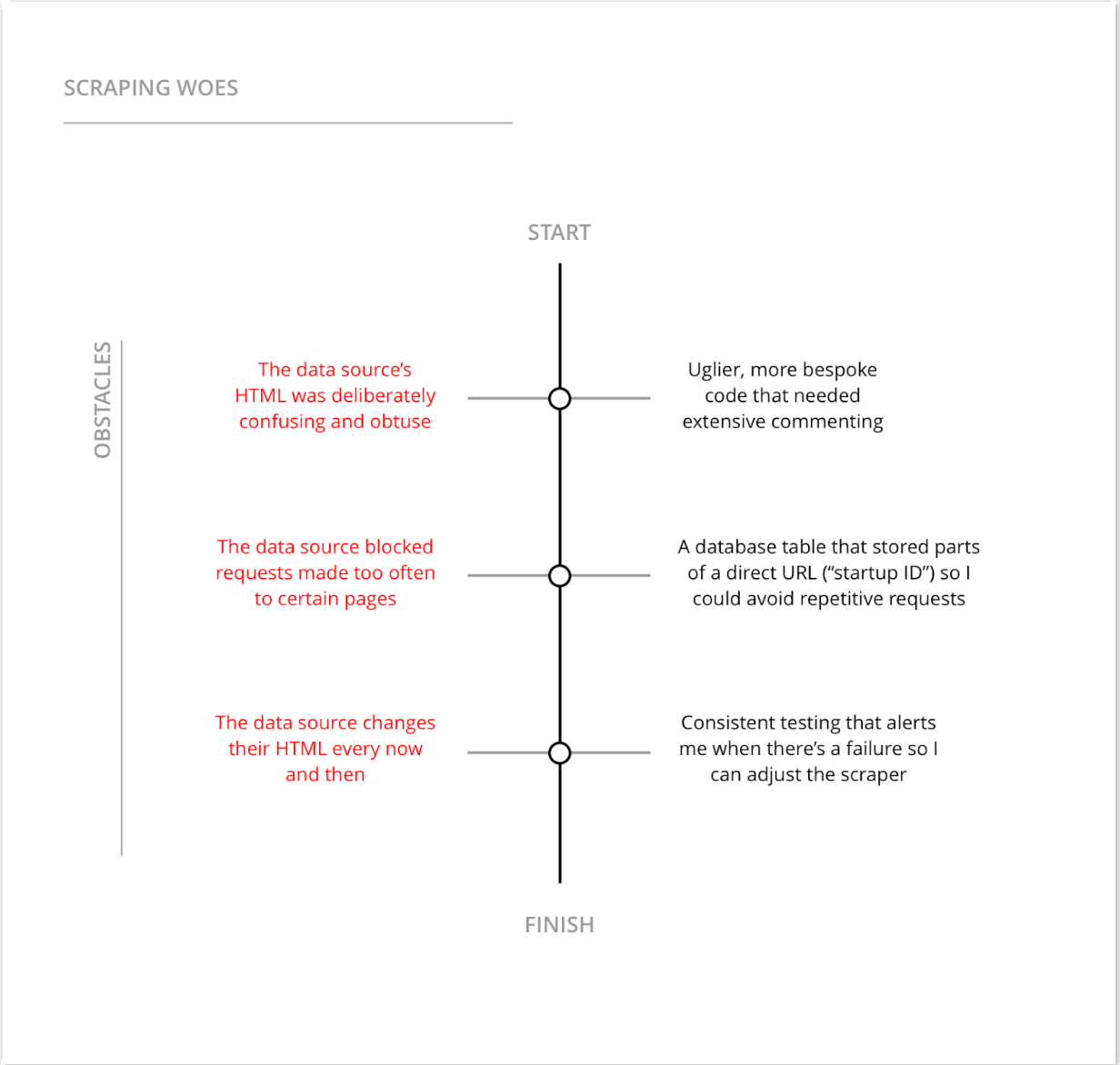
My original vision for what the product should look like made a few incorrect assumptions about how users would want to consume the service.

Problem	**How we addressed it**
Text as an input was not intuitive to users	Better set of onboarding messages and copy
Text had very tight input requirements	Very specific instructions and explanations as to why this was the case

These problems ended up being relatively innocuous, but I can see that if we ever wanted to scale this product, they might resurface.

BUILD PROBLEMS

As with any service that relies on web scraping, I ran into a ton of issues in the consistency department. AngelList employed a few strategies to prevent scraping, but I was able to navigate around them:



After adding in the Slack interface, I spun up an AWS EC2 instance and kept the bot running with Nohup. Here are some of the functions we ended up with:

```
#Function to check if the startupID is in our database
def checkSqlID(company):
    #Connect to DB
    engine = create_engine("mysql://INFO_HERE")
    con = engine.connect()
    ...
```

```
#Function to use requests to scrape a company page
def requestsURL(company):

    #Set url
    url = "https://angel.co/" + company.encode('utf-8').strip()
    ...
```

```
#Function to scrape investors
def scrapeInvestors(company):

    #See if the startup id is already in our database
    idNum = checkSqlID(company)
    ...
```

Successes and Failures

Overall, our small set of users ended up very satisfied, but hungry for more. The final feedback (in addition to the cycles above) was that this was a large step forward for a relatively unsophisticated industry, but the production and data constraints I needed to deal with severely hampered our ability to achieve our original vision.

In the end, we were able to create the right interface and delivery but weren't able to get the data and content that we had hoped for.

The project ended early for personal reasons, but we were able to spec out the next phase of improvements:

- A clear, user-defined schema for relevant data that can be applied as a filter to a raw set scraped from the web
- Ways of grabbing alternative data that could help augment our main set (for example, fund descriptions from their websites that could help classify what stage they invest in)
- Data verification through some external API or service
- Storing entities (people, funds) in a graph structure and ultimately predicting who's most likely to participate in secondaries

We never got a chance to work on any of these iterations, but hopefully I'll be able to get back to it some day!