Code Assessment Thoughts

Regarding my answer to Github/Peck's food truck challenge

This document goes over the thoughts and actions I went through while undertaking the food truck challenge under a time allotment of 3 hours. There will be discussion on obstacles I faced and my coding choices in the midst of my session. Included in this discussion will be tradeoffs and sacrifices I had to make to focus more on production-friendliness. Lastly, I will spin the yarn on how things could have proceeded differently if I knew I had more time. All this will be divided into two sections.

Undertaking the Challenge

Being the freshest language in my mind, Elixir became my choice with Phoenix holding the state and displaying the output of my application. Another way I could have approached this with Elixir was leaning more on OTP and using a GenServer to hold my state without Phoenix. However, this would mean that the console or files would become the vehicle for showing output. I felt both options for showing output were cumbersome compared to taking advantage of LiveView, where I wouldn't have to type multiple commands just to satisfy my whims on what food I really want. Plus coming from a background of HTML/CSS/JS, formatting on the web came easier than on the console or a rich text file. Having recently worked with LiveView in a production environment was the cherry on top.

Hydrate Truck Club was the title I picked I felt was different enough to show character while not implying too much in regards to its features. One can't go wrong with always having a liquid to drink unless the Wii is the goal; especially since classic food truck meals like Chicken and Lamb over rice or Burritos with fries inside tend to be on the saltier side. While San Francisco is more commuter friendly than others, my regretful experience in highway commuting has shined here. Helping to fight fatigue by ensuring drinks are an option (hopefully non-alcoholic) before going on the road feels a little better on the conscious.

One of my first choices in this time trial was not using Ecto with Elixir Phoenix. I did not feel a database would be needed nor tenable for a simple app. Adding metadata like having frequented a selected food truck or helping to ensure another layer of the application works on another computer didn't feel realistic within the short time frame. The effectiveness of the features it provides like queries or changesets feels quite dampened with only one input field. I haven't even mentioned testing yet. And so it was omitted as an option when I used the Phoenix Code Generator to generate my project.

Using `mix phx.new hydrate_truck_club` was definitely one of my biggest boons to saving time as I had a server ready to go in a few moments. A trade off here was leaving in the default "/" route within the router. I felt that creating a new live route to show my data with some instruction would be faster than trying to transform the non-live route and its contingent files. Even the new route had provided a bit of a challenge too, as the file I worked with was heex and not leex that my professional experience was based on. For a small form, I had to rely on older syntax I knew more than what heex also supported.

A general theme I had going on in my head while working on this was making sure what little I had worked without any sort of hacky-ness. I would only use pipelines like within` trucks.ex` if

I was sure that what it's working with is guaranteed. I would have liked a cleaner way to express the result within a filter lambda but spending more time on it felt like a luxury. I also forgot to cull some redundancy within the private function `fetch_trucks` of `club_live.ex`. The with version might be slightly less maintainable to read but I'm definitely open to feedback.

One sacrifice I made that may run counter to the assessment is not deploying the project for a production environment. This was an aspect of engineering I'm not well-versed in so I fell back to just uploading my project on github without doing anything special. Unless I had more time, I felt having to learn by looking it up would take too long.

I would have liked to add more unit and integration tests. The small form does not have test coverage and neither does the `club_live.ex` file. I only have most of the happy path and my honorable touch testing word.

Conditionally Hydrated with Time

Now if I was quenched with time, Ecto would become more attractive here. I'd be able to add meta fields like how many times has this truck been visited or revealing what the order was with a more robust form attached to a changeset for the struct Truck. Past orders would have to be preloaded of course but it'd be fun to see a smattering of random orders from trucks all over in a homepage of sorts.

One trade off left in was seeing multiples of the same truck name over and over in the filtered list page, only differing in addresses. It may be nicer if all of those same entries were collapsed together into one unit on the LiveView, showing all of the addresses it services. It may be necessary to pre-format the data such that all of the same multiples are grouped in subsets to facilitate. This would at least require extending the filtering logic and a special way of viewing it as a LiveComponent (probably stateless).

One feature I feel would benefit this application would be notifying website users if a food truck has changed any of its traits or if say was running a promotion. This would help showcase the strengths of LiveView because Phoenix comes with a PubSub feature and all the liveview process has to do is subscribe to the topic while mounting and handle the event. This requires a way to say MapDiff stored trucks and new truck data from within say a cronjob (could use Oban for instance) process.

While I do have a filter for only including trucks that have an “APPROVED” status, I would have liked to add a button set to include trucks with various statuses like “REJECTED”.

I’d definitely be able to add more tests like being able to write an integration test for filling in the form for filtering. I’d also be able to capture output from the json url so I wouldn’t have to request every single time I’d test or perhaps even part of some CI/CD pipeline.

Lastly, add a CSS Framework like Bulma to facilitate presentation; do a better thorough job of culling unneeded code, comments, or features that came with using the Phoenix Code Generator like swoosh; or learn how I can filter in the data api request since the Socrata Open Data API supports it.