Components - dsouther

Based on MJ's updates and the assignment prompt, this system should focus on architectures driven by data schema. The core piece of this architecture is the address schema, or the "Metadata store" in the initial one-pager. This metadata store will have computable descriptions of valid address formats for known countries, and the other components will drive themselves from this data. The UI will use the metadata to choose the form fields and validation to provide the user. The server will have an API structured to fit the shapes in the metadata. The database will be optimized by having per-country tables, with subtly different schemas per country. All of these nuances will be driven at runtime by the per-country address format metadata.

Metadata

Metadata about address formats is static. Once created, it can be distributed from a caching system, or embedded directly in relevant binaries. At the top level, the metadata should be a map from country code to list of formats for the country. Formats will share some overlap. Formats must encode which pieces of data are necessary, which are optional, their layout, and any required tokens (especially newlines). This data can be captured in a form which names the expected tokens, and provides semantic understanding in the grammar. For example, Australia would have these four formats:

```
[Number:n+] [Street Name:str] [Street type:ALLEY|ARC|AVE...] '\n'
    [City:Sydney|Perth...] [Province:ACT|NSW|NT...] [Zip:nnnn]

[PO Box] '\n'
    [City:Sydney|Perth...] [Province:ACT|NSW|NT...] [Zip:nnnn] '\n'

[Number:n+] [Street Name:str] [Street type:ALLEY|ARC|AVE...]
    [City:Sydney|Perth...] [Province:ACT|NSW|NT...] [Zip:nnnn] '\n'
    AUSTRALIA

[PO Box] '\n'
    [City:Sydney|Perth...] [Province:ACT|NSW|NT...] [Zip:nnnn] '\n'
    AUSTRALIA
```

Alternatively, using a Backus form, the formats could be combined:

```
(
    [PO Box:str]
    |
    [Number:n+] [Street Name:str] [Street type:ALLEY|ARC|AVE...]
) '\n'
    [City:Sydney|Perth...] [Province:ACT|NSW|NT...] [Zip:nnnn] ('\n'
    AUSTRALIA)?
```

Because it will be used to inform runtime decisions, it must be stored in a computable format. The metadata should require little to no parsing outside the standard libraries, as it will be used in multiple runtime environments. There are several formats that fit this requirement, chiefly XML, JSON, and YAML. These files can be generated from the master format.

Client

With the description of address formats from the Metadata store, the UI will dynamically create form fields and validation elements. Each token in the address format is exposed to the user as an `\<input>` element, with placeholder taken from the token name. Validators are applied based on the type associated with the token. Tokens with a limited set of values will be presented as dropdowns. Literal newlines can also be used to lay out the form. Tokens can be given special styling instructions from the token, as well. For instance, `:nnnn` only needs to be wide enough to handle 4 digits.

When submitting the form, the client can structure the user's inputs into a message whose structure matches the expected tokens. This structure will be documented by the server's implementation, and should stay stable over time.

Database

Given the success of the OpenAddress data set, the database will use its schema as the basis for the table definitions. The OpenAddress data set stores address columns of `number`, `street`, `unit`, `city`, `district`, `region`, `postcode`. The OpenAddress data files are keyed per country. For this application, giving each country a dedicated table in the database might be overkill. TODO(TEAM): Discuss trade offs of various database table configurations.

Server

The server will have a copy of the metadata. Client requests that come in should have their messages structured according to one of the address formats. The server will match the message to the appropriate format, and then validate the message values against said format. Failing to find a format is itself a validation failure. After validating the message, the server must prepare a database request (insert or query) with the client's message. The server will normalize the request to the appropriate schema for data insertion, or will query the database' indices for find with partial data.

Hydrator

The hydrator is an ETL pipeline to pull data from known datasets into the appropriate database formats. At this time, the source dataset comes from OpenAddress. The complete OpenAddress dataset contains XXX million (?)

entries. To seed this application, the hydrator will choose Y million entries with a uniform sample.