How your team will create the solution using your style / pattern choice

Our service uses RPC API(18) to run address queries on a remote server. The Client displays the address form, and will send in a request with address information. Then the server will process the request by searching for the address in a database and returning the results. The Client form will update based on the results of the RPC call.

The Client will communicate with the Service using Request/Response(54), as ideally the Service should not be a long operation. The API receives POST requests containing address data and extracts the address data. The address is then parsed into House Number and Street, and matched against the Metadata. The database is then queried using the address input, and any matches are packaged up into an appropriate response (valid/invalid), and sent back to the Client. If no matches exist, or if the appropriate format could not be detected from the input, then an error message is returned to the Client instead.

Matching of the address to the address pattern metadata will use the Request Mapper(109) pattern, as the address data will be received in different formats corresponding to the same thing - a house address - especially when the user is searching across countries. Request Mappers can be built that handle searching the same input address differently: 123 Main vs Main 123. One Mapper corresponding to [house number],[street] can query the database for results, while another Mapper can use [continental street],[house number]. Any results from the two Mappers can be combined and sent back to the Client for display.

The Parts:

Metadata description: Acts as an “address pattern” repository that houses appropriate address formats for all of the countries. Does not do anything by itself, but is used by the API to match a search request’s address pattern against all the countries patterns.

Client / dynamic form: Acts as a web-based “UI” for the API. Takes in user address input and transmits the input to the server to check for validity. Contains an Address field and a drop down menu for Country.

After the server responds, the front end updates the user form accordingly, showing either that the entered address was valid or not valid for a selected country, or displaying any errors that were encountered with the request.

Depending upon whether a Country has been selected, the system will use different search strategies, corresponding to the requirement that a user can search both across countries and within a country.

No Country: An address text field is the only field available.

Country selected: Selecting a country from the dropdown dynamically modifies the form to include other fields relevant to that country (State for US, Province for Canada for example), and constrains the search on the server side to just that country.

API – search: The main webservice program that is responsible for handling requests/responses. *The API receives POST requests containing address data and extracts the address data. The address is then parsed into House Number and Street, and matched against the Metadata. The database is then queried using the address input, and any matches are packaged up into an appropriate response (valid/invalid), and sent back to the Client. If no matches exist, or if the no appropriate format could be detected from the input, then an error is returned to the Client instead.*

Address Database: SQL(?) database that houses all of the addresses. Receives queries from the API service and responds with any matches.

Address Hydrator: Used to place a large number of addresses into the Database. A large collection of addresses in .csv format from each country will be used as the data source for the Hydrator. Rather than use the entire 14GB dataset, the Hydrator will sample each .csv file at a rate proportional to the size of the .csv file relative to other .csv files in each country. The Hydrator will then push the sampled addresses into the Address Database to be used later. Hydration is planned to occur once.