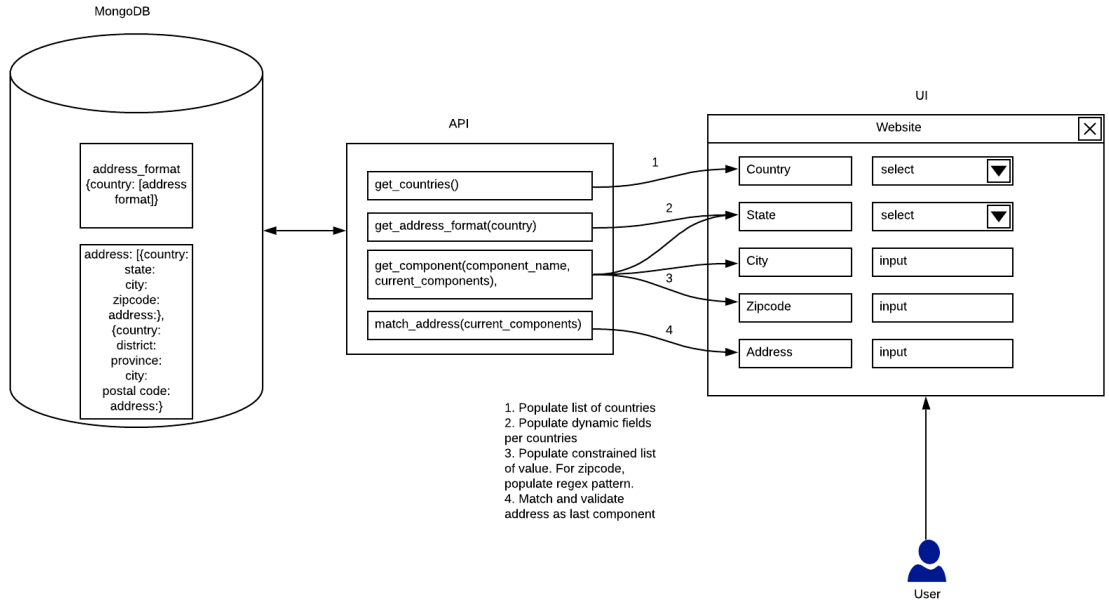
Name: Minh Nguyen, Nina Nguyen, Pongpichit Poonpiriyasup, Shaista Usman
CPSC 5200
Prelim Design Documentation

Design overview:
We are adopting **client-server style** for our software solution. On client particularly, we are using **event-based style** to wait and react to user's input or selection.

Given these styles, we are considering two design choices and patterns:

Option 1



API Design:

GET /getcountries

- Description: get a list of countries to pre-populate in country field. Client will have this list ready on load
- Parameters: None
- Response: 200 successful with JSON data; 404: not found

GET /getaddressformat

- Description: get a list of fields representing address parts per each country to send to client to populate dynamic address field
- Parameters: country

- Response: 200 successful with JSON data; 404: not found

POST /get_component

- Description: get a list of values/patterns to help validate current based on the parameters users select in the previous fields.
- Parameters: name of “lookup” field and list of values in previous fields (e.g. to populate city, send over “city” and list of selected values {“country”: “US”, “state”: “WA”})
- Response: 200 successful with JSON data; 404: not found

POST/match_address

- Description: validate the address text as last layer in hierarchy based on the parameters users have selected and types
- Parameters: list of values all fields
- Response: 200 successful; 404: not found

We organize data fields in the hierarchy of Country => State/Province/District => City/Village => Postal code => Address line.

Country, State/Province/District, City will be populated as a constrained list based on the user’s input of the previous fields. Postal code and address line fields will be free-form text with underlying validators.

As a user chooses Country, a request call is made to an API on server with the country as a parameter. The server will query Mongo Db on Country as key and return a list of valid inputs in State/Province/District field to send back to client to populate into State/Province/District. Same mechanism will be applied to populate the drop down list of valid cities in “City” field (given prior input of Country and State/Province/District).

For postal code, an API call will be made to server to query the regex pattern allowed (given prior input of Country, State/Province/District, and City). Using this regex pattern, client will impose constraint on input allowed in freeform text

For the address line field, an API call will be made to server to query to the exact match (given prior input of Country, State/Province/District, City, and Postal Code PLUS current input of address text). Server will return “valid” or “not valid” to the client as a response.

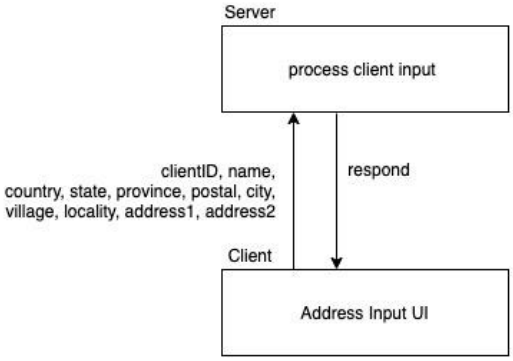
This design choice needs State and Memento **patterns** imposed on data fields on the hierarchy. Depending on the states of the upper chains, if these are in “selected” state, the immediate field below will be in valid state to accept input from users.

Trade-off:

- Multiple API calls as user input in each field vs. just one API call to validate all address components as one. May consider caching to address the number of calls
- Need to track state of the current address component given states of others in the upper chain
- The benefits however are immediate response on validity if one of the address components is incorrect

Option 2

Style: Client-Server



Record Address Procedure

1. Client receive user input via UI
2. Client send data to the server upon submission
3. Server validate the input data
4. If the input data is valid, then the server stores the data into the database
5. If the server stores the input data, then responds success. Otherwise, responds fail.

Search Address Procedure

1. Client receive user input via UI
2. Client send the query data to the server upon search submission
3. Server searches the database according to the query data.
4. Server responds with all the matched data according to the query

Pattern:

- Abstract Factory: The client provides the interface, whereas the server create the object (address)
- Facade: The server serve as a sub-system to the client
- Observer: One server can serve multiple clients.

Trade-off:

- Only one API call is made so latency won't be affected

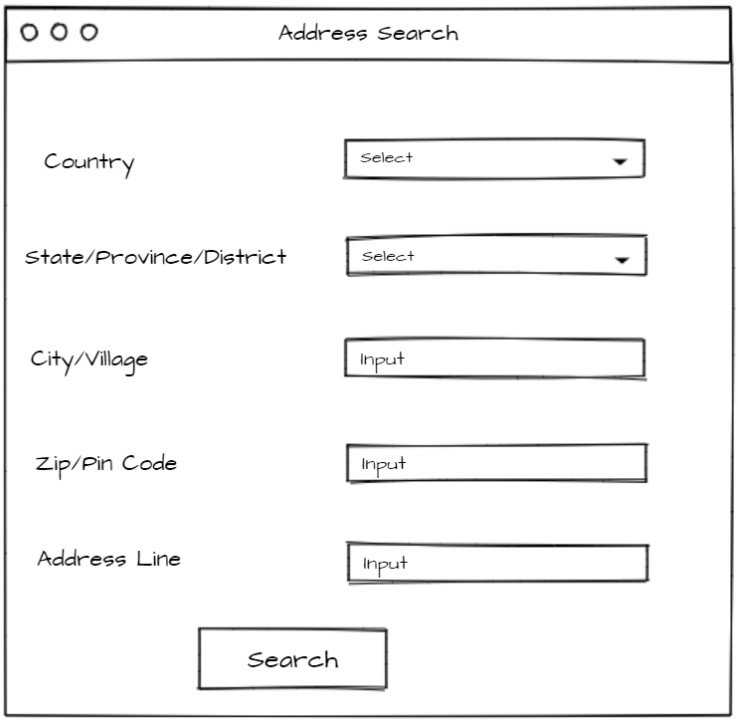
API design:

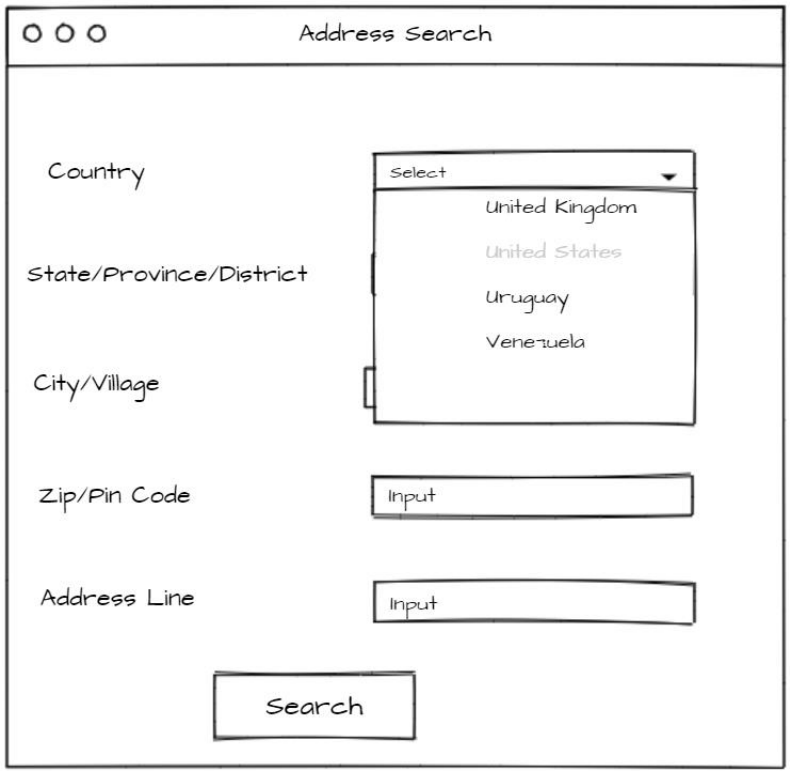
POST /record_address
- description: validate the input data format, and record the data onto the database if the format is valid
- parameters (body): { country: String, state: String, province: String, postal: String, city: String, village: String, locality: String, address1: String, address2: String }

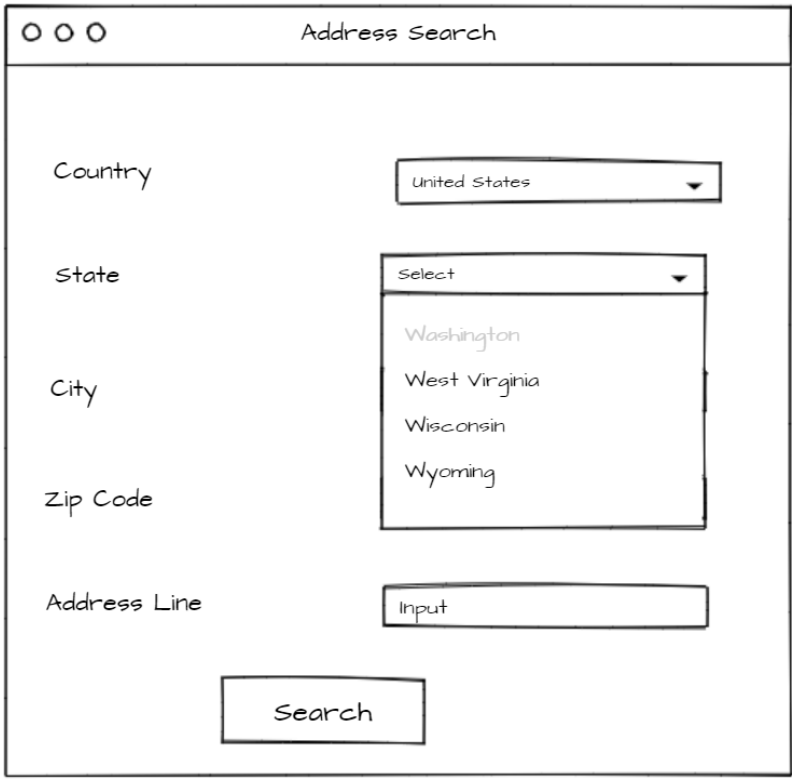
- response: 200; for a successful record.
  404; otherwise.

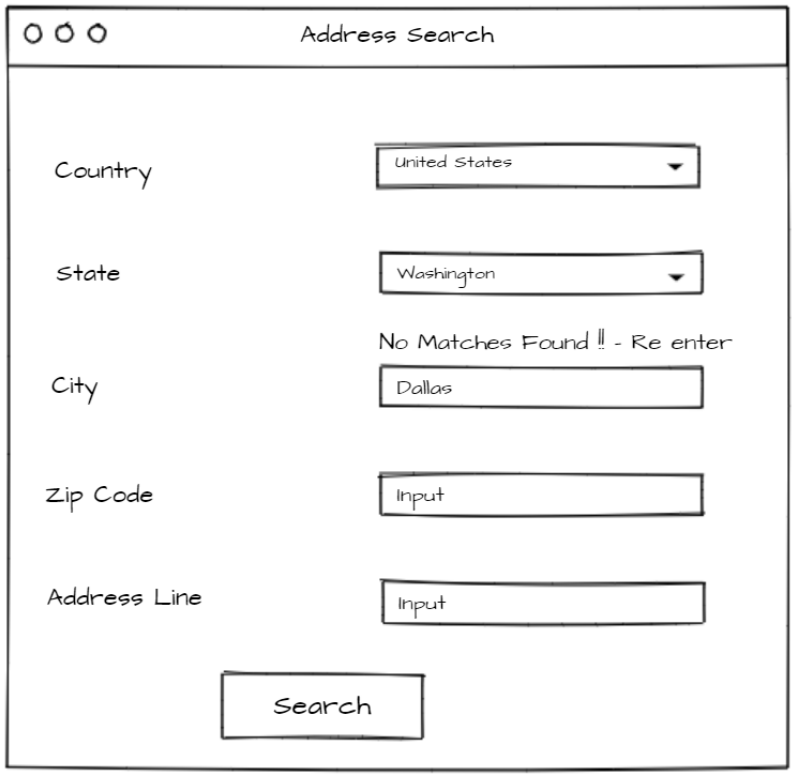
POST /search_address
- description: search the database according to the input query
- parameters (body): { country: String, state: String, province: String,
  postal: String, city: String, village: String,
  locality: String, address1: String, address2: String }
- response: 200; a list of matched addresses. An empty list otherwise.

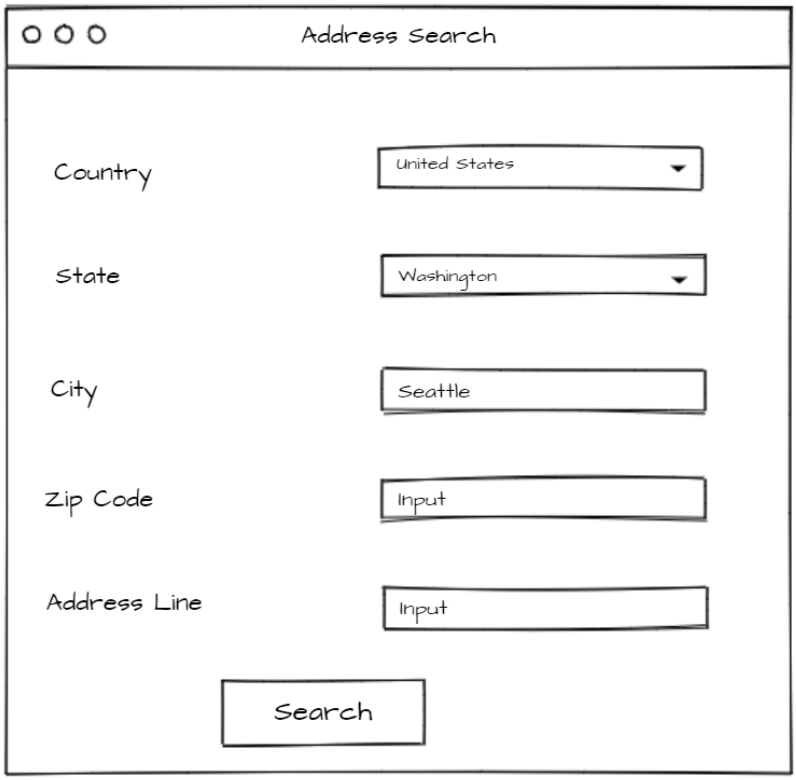
UI Mockups:

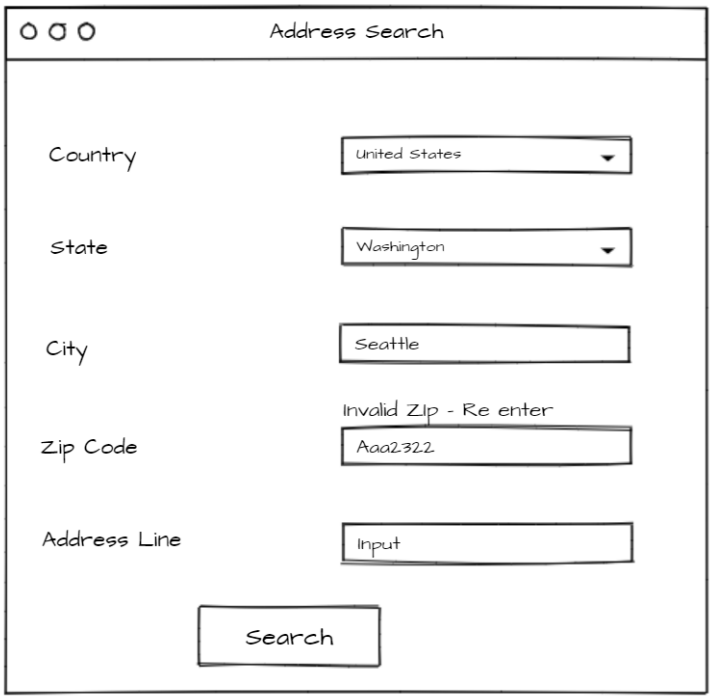


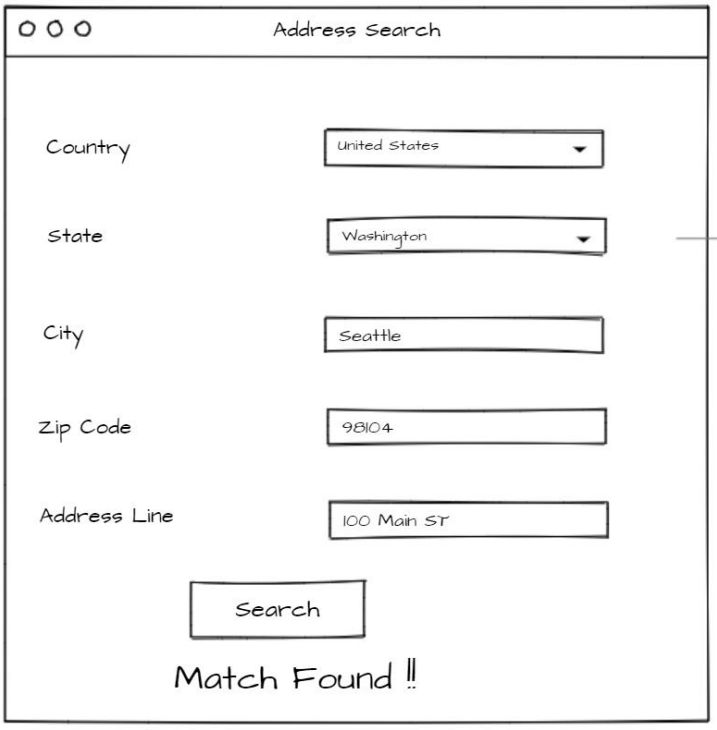

Address Search
Country
Select
United Kingdom
United States
Uruguay
Venezuela
State/Province/District
City/Village
Zip/Pin Code
Input
Address Line
Input
Search

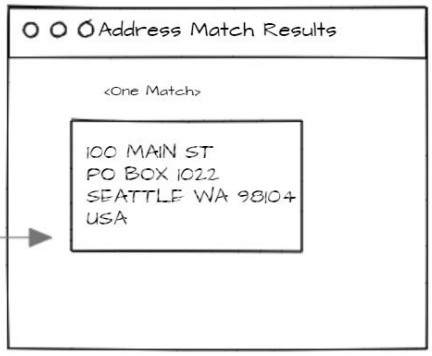

Address Search
Country
United States
State
Select
Washington
West Virginia
Wisconsin
Wyoming
City
Zip Code
Address Line
Input
Search


Address Search
Country
United States
State
Washington
No Matches Found !! - Re enter
City
Dallas
Zip Code
Input
Address Line
Input
Search


Address Search
Country
United States
State
Washington
City
Seattle
Zip Code
Input
Address Line
Input
Search


Address Search
Country
United States
State
Washington
City
Seattle
Invalid Zip - Re enter
Zip Code
Aaa2322
Address Line
Input
Search


Address Search
Country
United States
State
Washington
City
Seattle
Zip Code
98104
Address Line
100 Main ST
Search
Match Found !!


Address Match Results
<One Match>
100 MAIN ST
PO BOX 1022
SEATTLE WA 98104
USA