CPSC 5200-02 23WQ
Team Project - Address Search Engine

Team 1

Name: Jen –Chieh Lu, Karan Doshi, Vidhi Rathod, Jay Singhvi

GitHub link: Software Architecture and Design - Team Project (github.com)

Problem statement:

A very common problem in enterprise software development is related to handling postal addresses. Nearly every country has a peculiar, way to structure a postal address. Design a web-based user interface (a form) that can capture a country-specific address format entered by an end user
The form must dynamically adjust & capture the address
Validation of the address formats and related data must occur in a reasonable time for the user i.e., < 75ms
The end user should be able to select data from appropriate user interface elements and not just type everything in free form
Where possible, default values and constrained lists should be presented to the user
A way for a user to search for a given address based on the country-specific format (name, partial address)
A way for a user to search across countries to find "matching" addresses and display them in the application
Multiple Country Search
An API callable via HTTP (curl or postman) because we might want to sell access to this application
Documentation for the API, preferably available alongside the API itself (think swagger / OpenAPI) should support at least 1000 concurrent requests (read, write, and search). Your "database" should be seeded with a large number of addresses of various formats spread across the countries on a per-capita basis (or approximately based on number of residents in each country).You are responsible for writing the code that seeds your database, and you can decide if you exercise your API to do so or if you want to "go around" your API and write the data in some other way.

Analysis on problem statement

Managing postal addresses may be a difficult and time-consuming operation for software development firms. Every country has its own address format, making it challenging to develop a uniform system for obtaining and verifying data. This issue may be solved by creating a web-based user interface that can dynamically gather country-specific address formats, validate the data, and offer end-users with suitable user interface components. Furthermore, the solution should allow users to search for a given address based on the country-specific format, as well as search across different nations for matching addresses. Also, the solution should handle at least 1000 concurrent requests and an HTTP API accessible through curl or postman. In this article, we will discuss in detail the analysis of this problem statement and the possible solutions.

Designing a Web-Based User Interface:
The first step in resolving the postal address problem is to create a web-based user interface that can record country-specific address forms. The user interface should be simple and straightforward to use, allowing end users to enter data without difficulty. The form should be structured in such a manner that the address format may be dynamically adjusted and captured dependent on the user's input.

Validation of Address Formats:
The next step is to check the address formats that end users have supplied. To guarantee a flawless user experience, the validation should take a fair amount of time, i.e. 75ms. The validation procedure should ensure that the address format is proper, including the number of characters, presence of special characters, and order of the address fields. Validation should additionally check for address correctness, including the existence of necessary fields like street name, house number, city, and zip code.

Appropriate User Interface Elements:
The user interface should give end-users with suitable user interface components, allowing them to pick data from limited lists and default values whenever possible. For example, based on the selected nation, the user interface may provide a drop-down list of cities and zip codes. This reduces the likelihood of mistakes and improves the overall user experience.

Search for a Given Address:
Users should be able to search for a specific address using the country-specific format, which includes the name and a partial address. The search tool should be swift and efficient, allowing end users to get the information they need quickly. The search feature should also be built to handle spelling mistakes and minor differences in address format.

Search Across Countries:
The solution should also allow users to search across nations for addresses that match. This capability is very valuable for multinational corporations with offices in various countries. The search tool should be built to handle diverse address formats and return correct results.

API Callable via HTTP:
The solution should have an API that can be accessed over HTTP, allowing users to utilize tools like curl or postman to access the application. The API should be well-documented, with all relevant information such as endpoint URLs, input parameters, and response formats included. The API documentation should be provided with the API, making it easy for developers to incorporate the application into their systems.

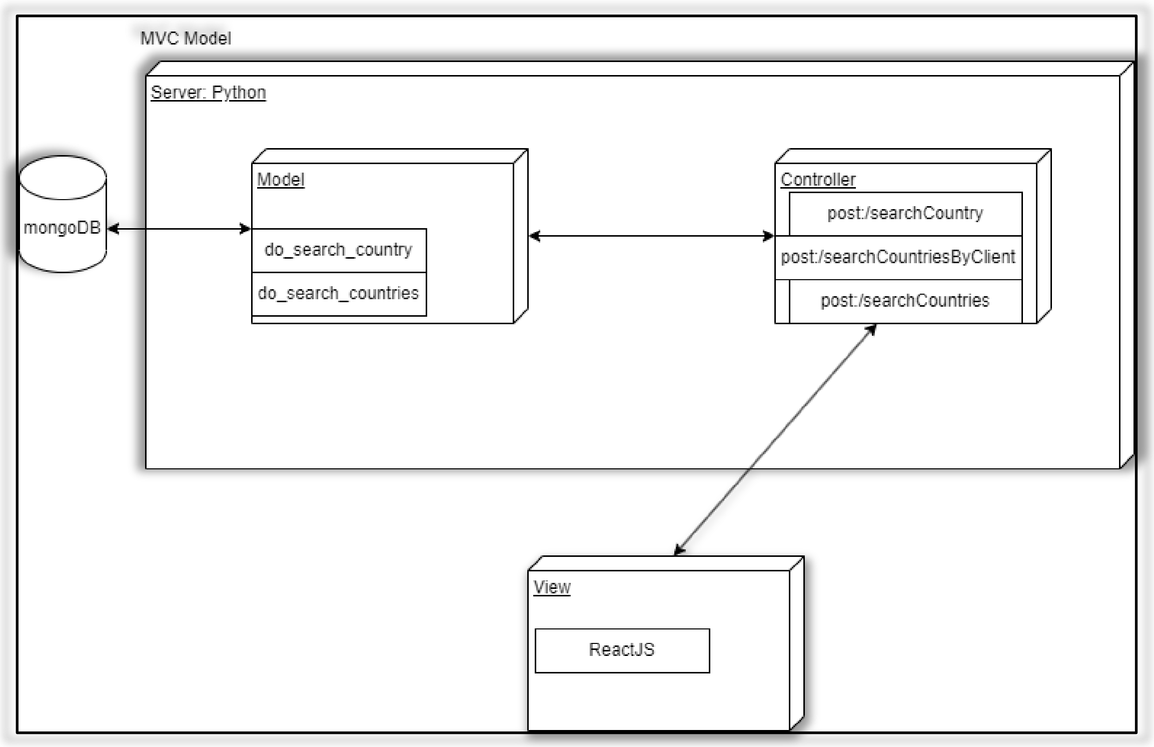
Concurrent Requests:
At least 1000 concurrent requests, including read, write, and search requests, should be supported by the solution. To do this, the program needs be built with scalable technologies like cloud computing or microservices. The database should also be constructed such that it can accommodate a huge number of requests without slowing down.

Seeding the Database:
The database should be seeded with a significant number of addresses in various formats distributed across nations on a per-capita or approximate basis based on the number of people in each country. The data can be seeded via the API or any other means the developer chooses. The database should also be constructed such that it can manage a big volume of data without slowing down.

Architecture Style

The architectural style we consider is MVC pattern, which stands for Model-View-Controller, is a software architecture pattern that separates an application data (model), user interface (view), and control logic (controller) into separate components. This separation of concerns has several advantages that make it an excellent choice for building websites:

1. Maintainability: By separating the different components, our team can work on each part independently without affecting the others. This makes it easier to update or modify individual components without affecting the entire application.
2. Scalability: As websites grow in complexity, it becomes more important to have a modular architecture that can scale with the application. With MVC, the separation of concerns makes it easier to add or remove features without having to rewrite the entire application.
3. Testability: With MVC, each component can be tested independently, which makes it easier to identify and fix issues. This also makes it easier to write automated tests, which can help catch issues early on.



Architecture Style Diagram

Analysis Model

Tech stack

REST API

REST APIs and HTTP APIs are both RESTful API products. REST APIs support more features than HTTP APIs, while HTTP APIs are designed with minimal features so that they can be offered at a lower price. Choose REST APIs if you need features such as API keys, per-client throttling, request validation, AWS WAF integration, or private API endpoints. Which makes REST API perfect for this project as we need to validate postal addresses too.

HTTPS Protocol

Hypertext transfer protocol secure (HTTPS) is the secure version of HTTP, which is the primary protocol used to send data between a web browser and a website. HTTPS is encrypted to increase security of data transfer. This is particularly important when users send sensitive data. HTTPS uses an encryption protocol to encrypt communications. The protocol is called Transport Layer Security (TLS), although formerly it was known as Secure Sockets Layer (SSL). This protocol secures communications by using an asymmetric public key infrastructure. i.e., private key and public key. HTTPS prevents websites from having their information broadcast in a way that is easily viewed by anyone snooping on the network. When information is sent over regular HTTP, the information is broken into packets of data that can be easily "sniffed" using free software. This makes communication over an unsecure medium, such as public Wi-Fi, highly vulnerable to interception. In fact, all communications that occur over HTTP occur in plain text, making them accessible to anyone with the correct tools, and vulnerable to on-path attacks.

JSON

It is a text-based way of representing JavaScript object literals, arrays, and scalar data. JSON is easy to read and write, while also easy for software to parse and generate. It is often used for serializing structured data and exchanging it over a network, typically between a server and web applications.

MONGO DB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is a cross-platform, document-oriented database that supplies high performance, high availability, and easy scalability. MongoDB works on the concept of collection and document. Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases. Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose. A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Service Design Patterns

Web Service API style

REST API Style – Validate the form data and search the information store in the database as per mentioned.

Client Service Interactions style

Request / Response – Client will send form as a request to the server and server provide the addresses as response.

Service Infrastructures

Service Interceptor – For the purpose of validating the form address values and searching the data and providing exception in case no data is found.

Service Implementation

Data-source Adapter – Allowing access to internal database to 3rd party clients

Request and Response Management

Data Transfer Object – Transfer of data from server to client based on queries raised by client through address request form.

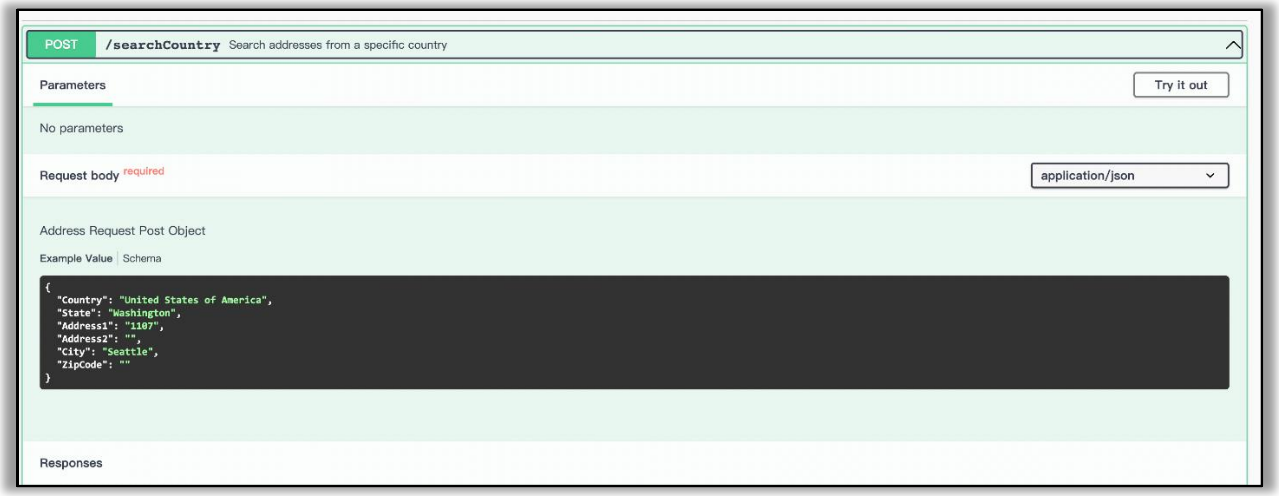
API definition

Little language

Post:{<searchCountry>}[{ **Country:"String"** **State:"String"** **Address1:"String"** **Address2:"String"** **City:"String"** **ZipCode:"String"** **}]**	Parameter: 1. API "searchCountry" is to search for addressed by a country which user choose. 2. Countries that Users can choose are USA, Mexico, Canada, Japan, India. 3. Partial address in Address1 and Address2 are allowed. 4. State, City and Zip Code are optional.
Post:{<searchCountries>}[{ **Country(Array)** **State:"String"** **Address1:"String"** **Address2: "String"** **City:"String"** **ZipCode:"String"** **}]**	Parameter: 1. API "searchCountries" is to search for addressed by multiple countries at same time 2. Countries that Users can search are USA, Mexico, Canada, Japan, India. 3. Paramater countries are Array type so that it can send multiple countries in a same request **4.** Partial address in Address1 and Address2 are allowed. **5.** State, City and Zip Code are optional.
Post:{<searchCountriesByClient>}[{ **Name:"String"** **Country(Array)** **State:"String"** **Address1:"String"** **Address2: "String"** **City:"String"** **ZipCode:"String"** **}]**	Parameter: 1. API "searchCountries" is to search for addressed in one or more countries by client's name 2. Client Name must be full name; partial is not allowed. 3. Countries that Users can search are USA, Mexico, Canada, Japan, India. 4. Parameter countries are Array type so that it can send multiple countries in a same request 5. Partial address in Address1 and Address2 are allowed. State, City and Zip Code are optional.

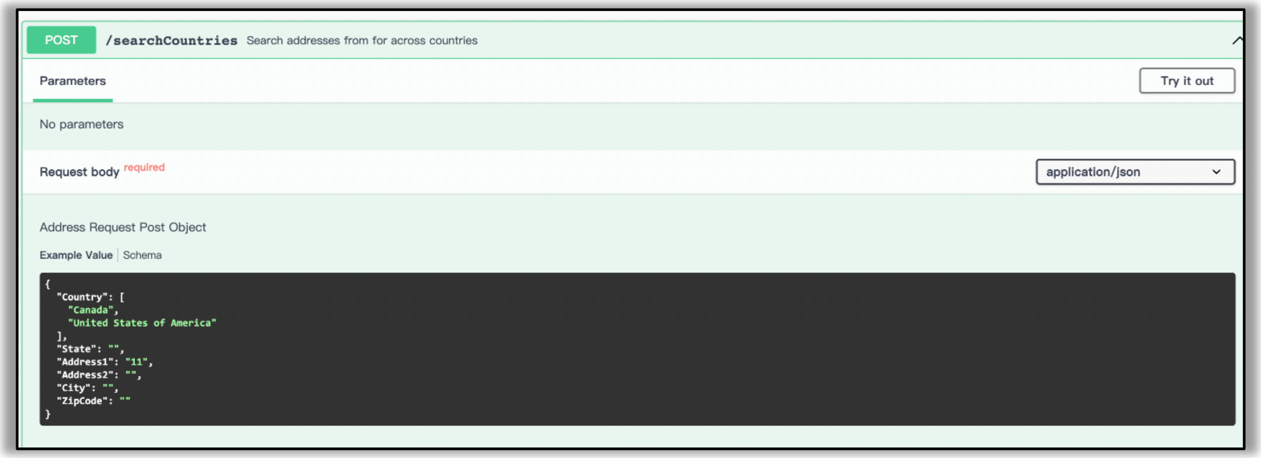
Swagger

URL: http://127.0.0.1:5000/swagger/
Features bellows are sample body messages in each API Calls:



```
{
  "Country": "United States of America",
  "State": "Washington",
  "Address1": "1107",
  "Address2": "",
  "City": "Seattle",
  "ZipCode": ""
}
```

API:searchCountry


POST **/searchCountries** Search addresses from for across countries

Parameters

Try it out

No parameters

Request body required

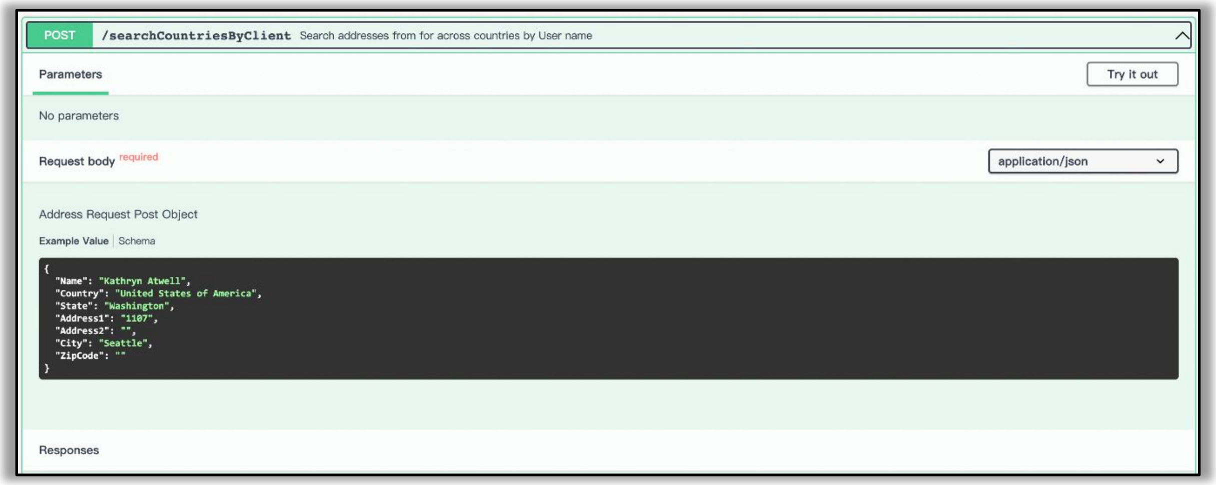
application/json

Address Request Post Object

Example Value | Schema

```
{
  "Country": [
    "Canada",
    "United States of America"
  ],
  "State": "",
  "Address1": "11",
  "Address2": "",
  "City": "",
  "ZipCode": ""
}
```

API:searchCountries


POST **/searchCountriesByClient** Search addresses from for across countries by User name

Parameters

Try it out

No parameters

Request body required

application/json

Address Request Post Object

Example Value | Schema

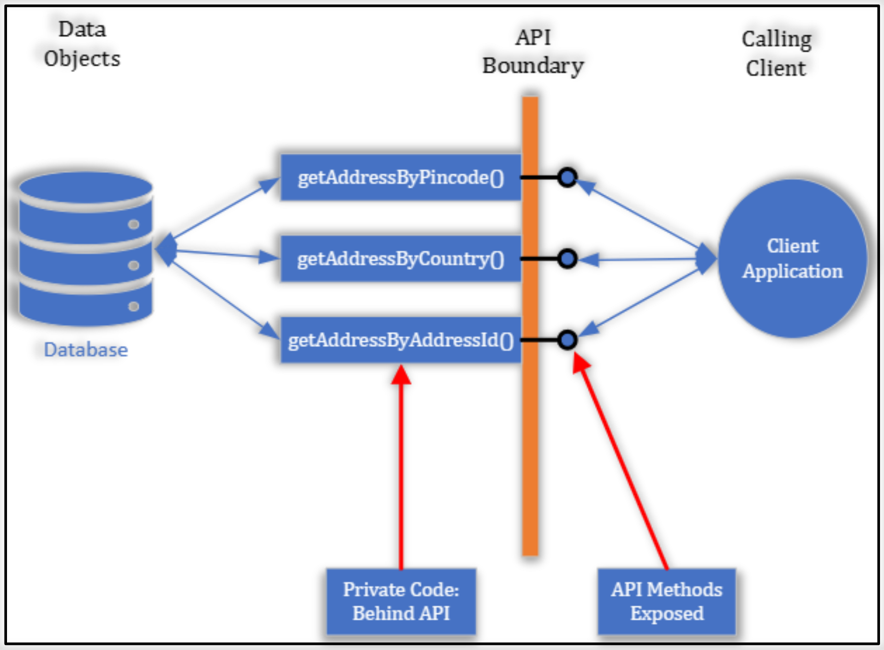
```
{
  "Name": "Kathryn Atwell",
  "Country": "United States of America",
  "State": "Washington",
  "Address1": "1107",
  "Address2": "",
  "City": "Seattle",
  "ZipCode": ""
}
```

Responses

API:searchCountriesByClient

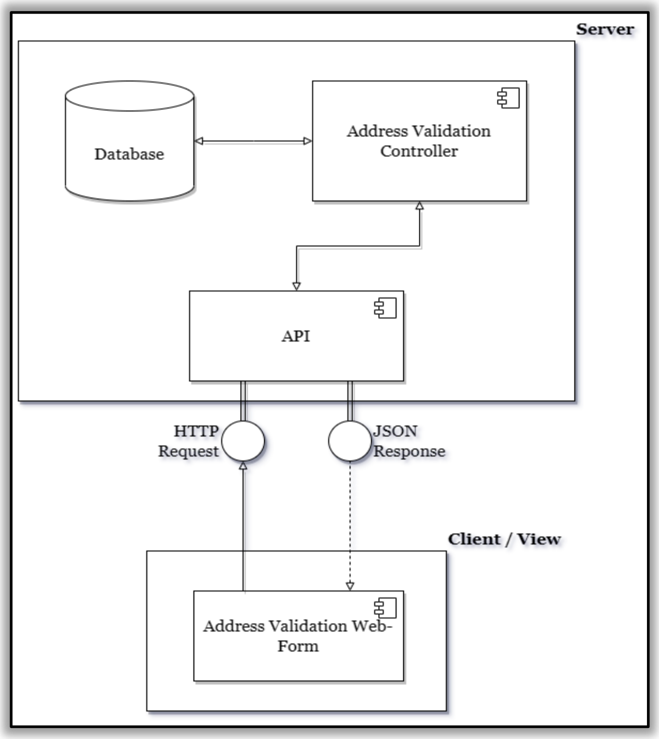
Diagrams

Context Diagram

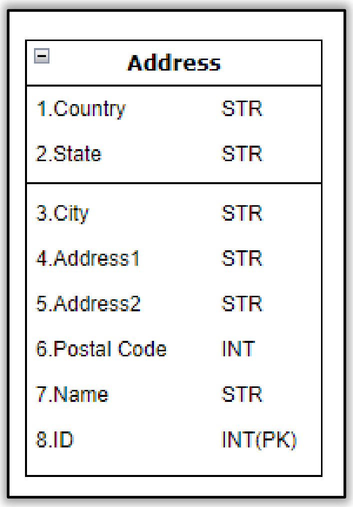


Component Diagram

a. Database – This will store all the valid addresses already available fed into the system.
b. Controller – Address Validator – Addresses provided by the API will be checked with reference to the addresses already available in the database for the validity of their formats
c. API – It will host all the methods available for searching which can be used by users for searching addresses in the system.
d. Client / View – Dynamic User Interface available for users while accessing the search functionalities of the system to browse over the addresses stored in the Database.
e. Request – Client System will send HTTPS requests to the API by calling the API Methods with form values passed as parameters.
f. Response - Once the validation check is performed by controller, API will send a JSON response back to the client system with results. Success message with search result list or Error message for failure of validation.

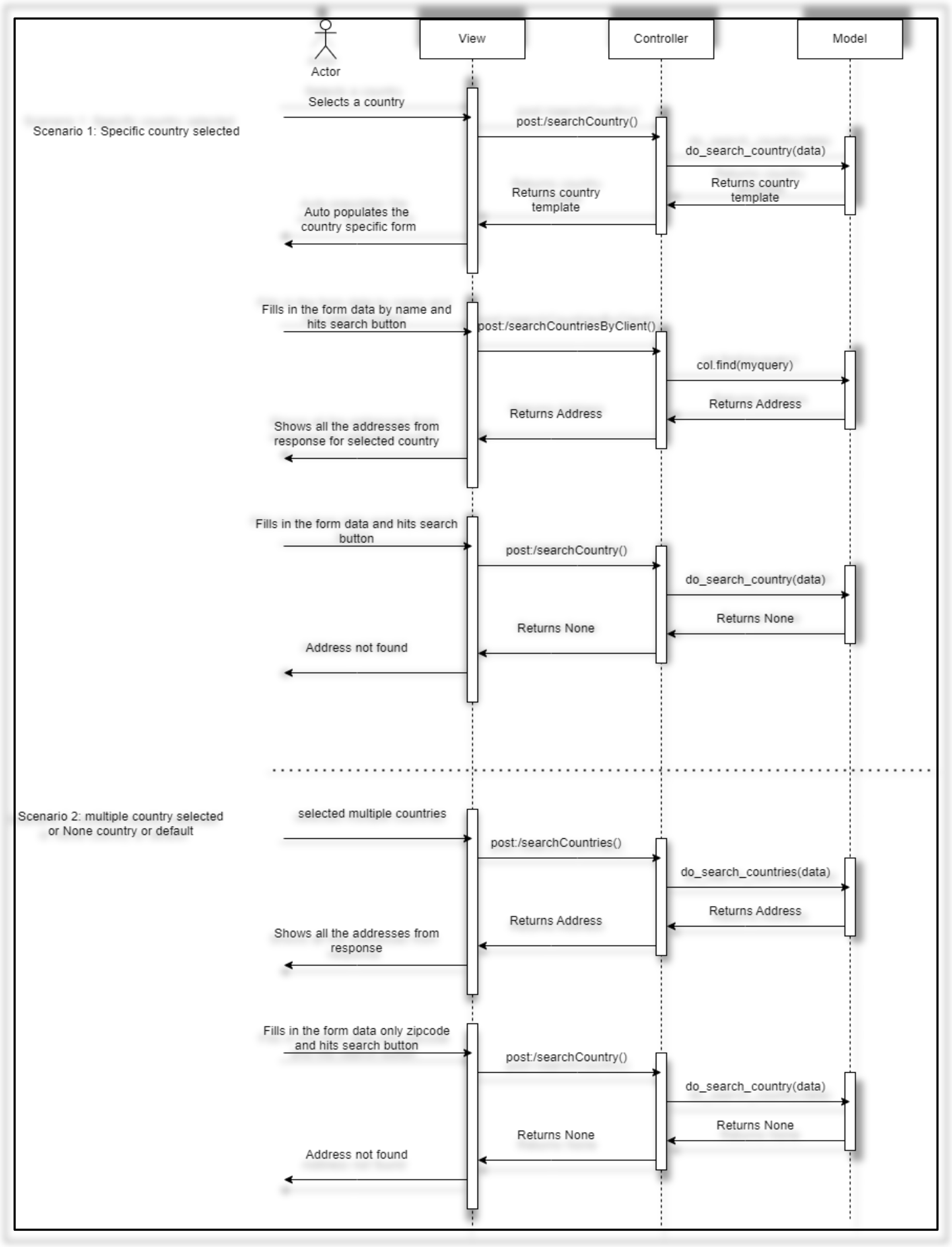


ER-diagram



Address	
1.Country	STR
2.State	STR
3.City	STR
4.Address1	STR
5.Address2	STR
6.Postal Code	INT
7.Name	STR
8.ID	INT(PK)

Sequence Diagram


Actor
View
Controller
Model
Scenario 1: Specific country selected
Selects a country
post:/searchCountry()
do_search_country(data)
Returns country template
Returns country template
Auto populates the country specific form
Fills in the form data by name and hits search button
post:/searchCountriesByClient()
col.find(myquery)
Returns Address
Returns Address
Shows all the addresses from response for selected country
Fills in the form data and hits search button
post:/searchCountry()
do_search_country(data)
Returns None
Returns None
Address not found
Scenario 2: multiple country selected or None country or default
selected multiple countries
post:/searchCountries()
do_search_countries(data)
Returns Address
Returns Address
Shows all the addresses from response
Fills in the form data only zipcode and hits search button
post:/searchCountry()
do_search_country(data)
Returns None
Returns None
Address not found

A. When data is found

Verb : POST

uri : /searchCountry

In this we will search for the address based on the country

Body:

Request	Response
{ “country” : “USA”,	{ “country” : “USA”,

“Address1” : “901, 12th Ave”,	“Address1” : “901, 12th Ave”,
“Address2” : “Unit 900”,	“Address2” : “Unit 900”,
“City” : “Seattle”,	“City” : “Seattle”,
“State” : “WA”,	“State” : “WA”,
“Postal Code” : “98122”,	“Postal Code” : “98122”,
}	}

:

B. When multiple data are found
Verb : GET
uri : /addresses
In this we will search for the address based on the country

Request	Response
{	[{
“country” : “USA”,	“country” : “USA”,
“Address1” : “901, 12th Ave”,	“Address1” : “901, 12th Ave”,
“Address2” : “Unit 900”,	“Address2” : “Unit 900”,
“City” : “Seattle”,	“City” : “Seattle”,
“State” : “WA”,	“State” : “WA”,
“Postal Code” : “98122”,	“Postal Code” : “98122”,
}	},
	{
	“country” : “Canada”,
	“Address1” : “901, 12th Ave”,
	“Address2” : “Unit 900”,
	“City” : “Seattle”,
	“State” : “WA”,
	“Postal Code” : “98122”,
	}]

C. When data is not found
Verb : POST
uri : /addresses
In this we will search for the address based on the country

Request	Response
{	{
“country” : “USA”,	"status": 404,
“Address1” : “901, 12th Ave”,	"message": "No data found for the given query",
“Address2” : “Unit 900”,	"data": {}
“City” : “Seattle”,	}
“State” : “WA”,	
“Postal Code” : “98122”,	
}	

Non-Functional Requirements

1. Performance: Performance in our project defines how fast our system or a particular piece of the system responds to certain users’ request under a certain workload. This non-functional requirement metric explains how long a user needs to wait before the main desired operation

happens, fetching the data from the source given the overall number of users. With respect to our team project, to improve the performance of the system, we can use Database instead of a JSON to make sure that the response time is quick, and the output is received in proper time.

2. Availability: - System should always be available during working/office hours. And as this system is not specific to an organization, this system should be available always so that people from across globe able to access it throughout any time zone without any issues. Availability is important in a software system because it ensures that the system is consistently and reliably accessible to its users. High availability means the system can operate continuously without significant downtime or disruptions, which is crucial for mission-critical applications such as e-commerce, healthcare, finance, and emergency services. It also enhances user satisfaction and trust, which can lead to increased usage and revenue for the business.
3. Simplicity: Simplicity refers to the concept of designing a system that is easy to understand and easy to use and easy to maintain. It involves minimizing the unwanted complexity, reducing the number of components, and ensuring the system's behavior is easy to predict.
4. Scalability: Making system grow is the common goal in every project, a clever design architecture is a key to making a system easy to extend and scale. In our team project, we assume that there might be a lot of clients using our system to transfer addresses into formatted addresses as it is common for any paperwork in the company.

UI Design

On the search page (Fig. 1), users can select the country they want, and the country's formatted form will appear in(Fig. 2). States will be automatically fetched into the selection roll (Fig. 3). Users need to fill in Address1, Address2, City, and Postal Code and name, and then click on the "Submit" button to look for a matching address. For example, in Fig. 4, if users' type " Kathryn Atwell " in the country of USA and click on "submit," the form will be sent in JSON format to the service. Once the system receives the data back, it will display the results in a list format (Fig. 5). However, if the system cannot find any data in the database, it will show an empty data in the result page (Fig. 6).

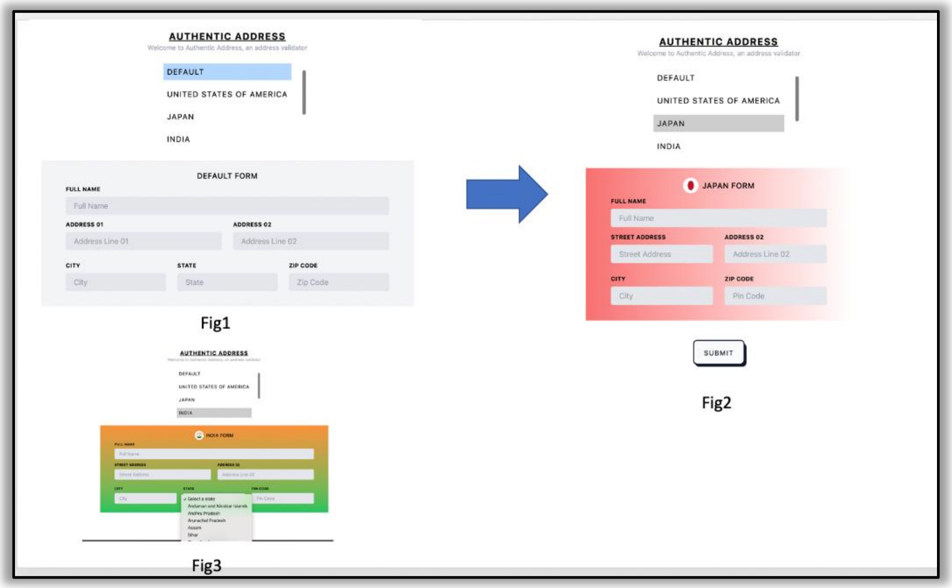
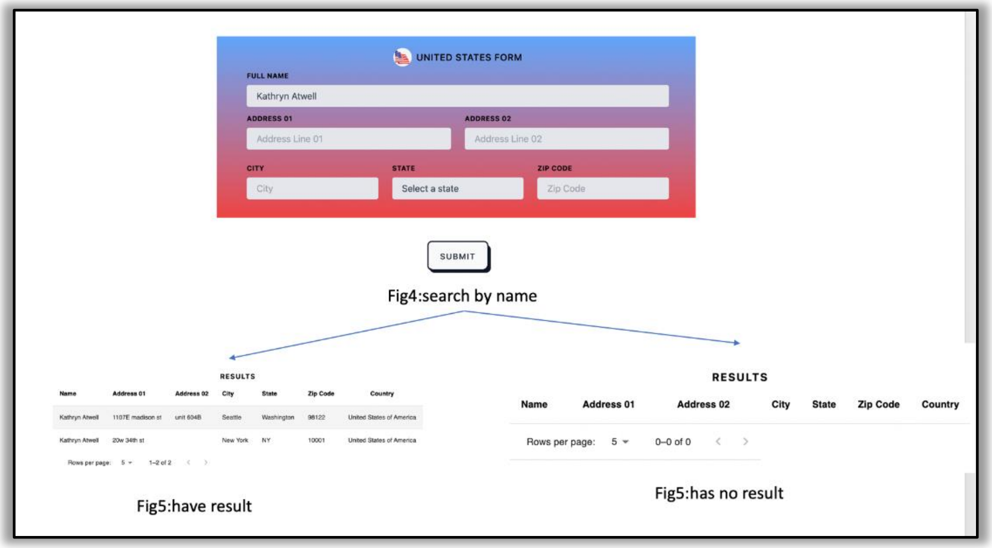


Fig1
Fig2
Fig3



UNITED STATES FORM

FULL NAME: Kathryn Atwell

ADDRESS 01: Address Line 01 | ADDRESS 02: Address Line 02

CITY: City | STATE: Select a state | ZIP CODE: Zip Code

Fig4:search by name

RESULTS

Name	Address 01	Address 02	City	State	Zip Code	Country
Kathryn Atwell	1107E madison st	unit 604B	Seattle	Washington	98122	United States of America
Kathryn Atwell	20w 34th st		New York	NY	10001	United States of America

Rows per page: 5 1–2 of 2

Fig5:have result

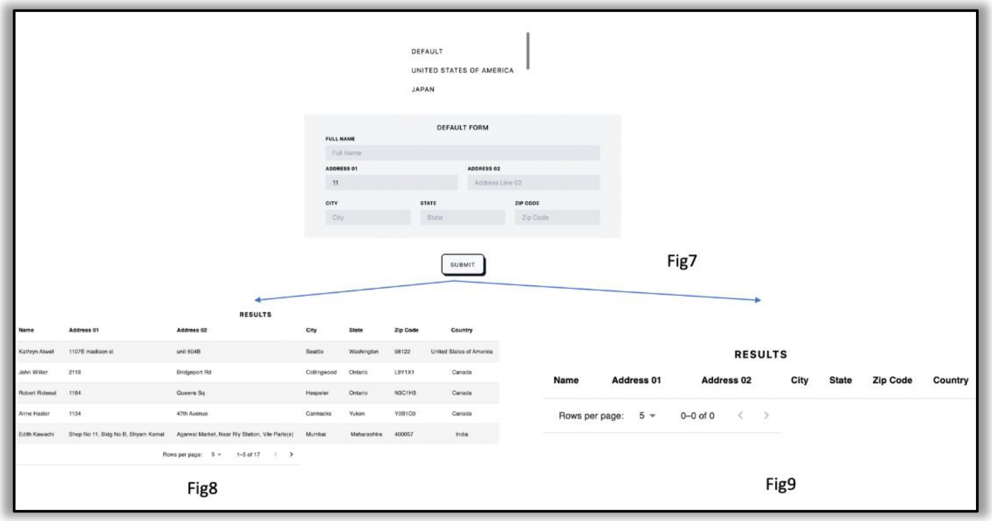
RESULTS

Name	Address 01	Address 02	City	State	Zip Code	Country

Rows per page: 5 0–0 of 0

Fig5:has no result

Another way to search for an address is to search across countries to find "matching" addresses. Users can type partial or complete address in the default form(fig.7). Since each country has a different form, the way we need to do this is to find the fields that are common across all countries and display the results in a list(fig.8). This will allow users to easily find the address they are looking for, regardless of the country. Also, if there is not any match address in the DB, then response no result page (Fig. 9)



DEFAULT FORM

FULL NAME: Full Name

ADDRESS 01: 11 | ADDRESS 02: Address Line 02

CITY: City | STATE: State | ZIP CODE: Zip Code

Fig7

RESULTS

Name	Address 01	Address 02	City	State	Zip Code	Country
Kathryn Atwell	1107E madison st	unit 604B	Seattle	Washington	98122	United States of America
John Willer	2116	Bridgeport Rd	Collingwood	Ontario	L9Y1X1	Canada
Robert Rideout	1184	Queens Sq	Hespeler	Ontario	N3C1H3	Canada
Anne Hasler	1134	47th Avenue	Carmacks	Yukon	Y0B1C0	Canada
Edith Kawachi	Shop No 11, Bldg No B, Shyam Kamal	Agarwal Market, Near Rly Station, Vile Parle(e)	Mumbai	Maharashtra	400057	India

Rows per page: 5 1–5 of 17

Fig8

RESULTS

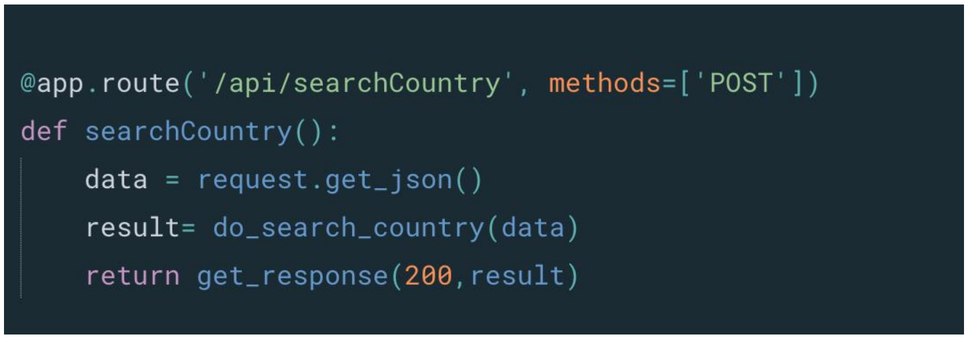
Name	Address 01	Address 02	City	State	Zip Code	Country

Rows per page: 5 0–0 of 0

Fig9

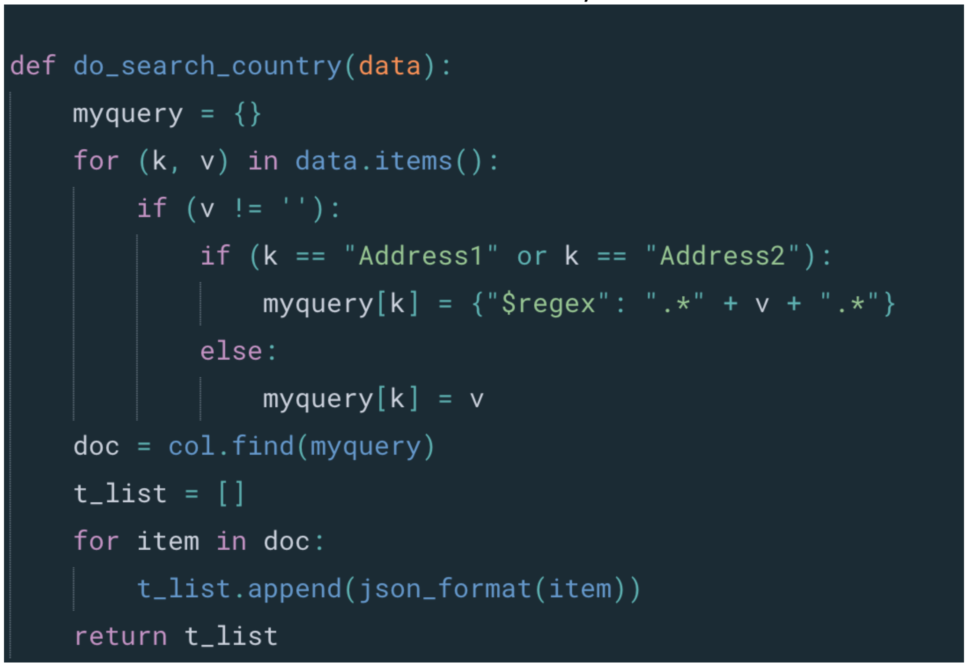
Code

Since we apply MVC pattern in the project, we see each API calls are controllers and call model which access MongoDB and extracted data and then send back to controller. Controller than response data to users. Take API “searchCountry” as an example, once the API gets a request from client, it calls model “do_search_country” to get data and then send response to client.



```
@app.route('/api/searchCountry', methods=['POST'])
def searchCountry():
    data = request.get_json()
    result= do_search_country(data)
    return get_response(200,result)
```

API:searchCountry



```
def do_search_country(data):
    myquery = {}
    for (k, v) in data.items():
        if (v != ''):
            if (k == "Address1" or k == "Address2"):
                myquery[k] = {"$regex": ".*" + v + ".*"}
            else:
                myquery[k] = v
    doc = col.find(myquery)
    t_list = []
    for item in doc:
        t_list.append(json_format(item))
    return t_list
```

Model: do_search_country

Future Scope/Summary/Conclusion:

1. Extending our database to host a greater number of addresses for existing countries
2. Extending our dataset to include more number of countries
3. Adding integration with Google Maps service to show the coordinates of the addresses on the map directly