Team Members: Madhuroopa Irukulla, Aditya Gollapalli, Priyadarshini Shanmugasundaram Murugan, Priyanka Vysyaraju.
TP3 - Team Project - Detailed Design & Implementation
CPSC 5200-02 00-02 23WQ SOFTWARE ARCHITECTURE AND DESIGN

SUBMISSION DATE- MARCH 10, 2023

TABLE OF CONTENTS

INTRODUCTION

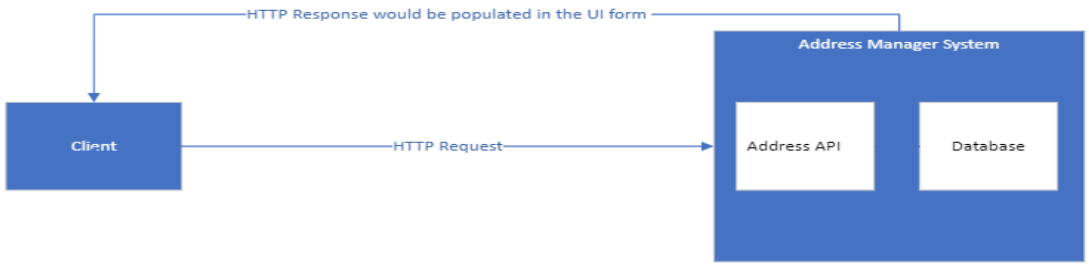
The common requirement for variety of applications such as E-Commerce platforms, Shipping/ delivery services, Location based services etc. It is to be maintaining and handling postal addresses. The requirement is to build a web-based interface that dynamically captures country specific address formats. The validation of the address given by the user, searching for the address and retrieving the results and displaying them, everything must be efficient and the response time must be less.

Overall, the problem statement requires us to design and implement a scalable, reliable, and efficient API that can provide accurate addresses based on input parameters such as postal code, state, country, or any combination of address specific details. This involves understanding the functional and nonfunctional requirements, selecting appropriate data sources to retrieve data and use it, designing an appropriate architecture and data model, and implementing the API using appropriate programming languages and frameworks.

ARCHITECTURAL OVERVIEW

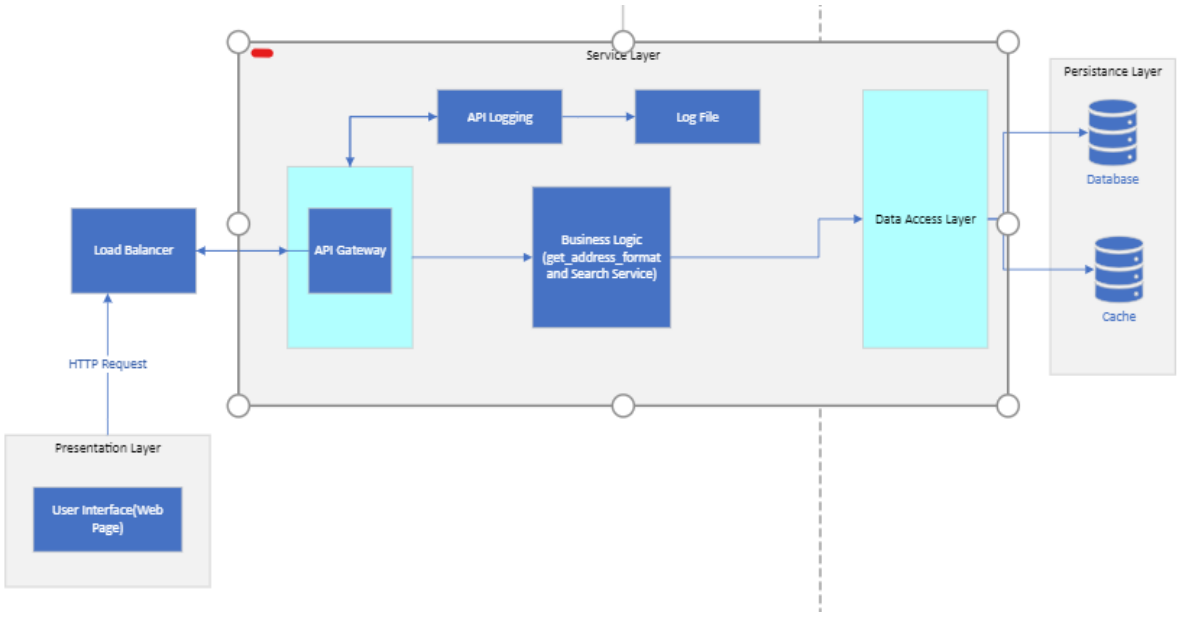
SYSTEM CONTEXT DIAGRAM

In this context diagram, the Client is the external entity that sends HTTP requests to the RESTful API, which then returns the street address data based on the input parameters (combination of any of the Postal Code, State, or address details). The client can provide the combination of input, For instance, Street Number and zip code or country, state and city and the API returns the corresponding street address data as output.



In this context diagram, the Client is the external entity that sends HTTP requests to the RESTful API, which then returns the street address data based on the input parameters (combination of any of the Postal Code, State, or address details). The client can provide the combination of input, For instance, Street Number and zip code or country, state and city and the API returns the corresponding street address data as output.

COMPONENT DIAGRAM



The above diagram is the high-level version of component diagram which show what all components are involved and how are they interacting with each other and data.

Components:

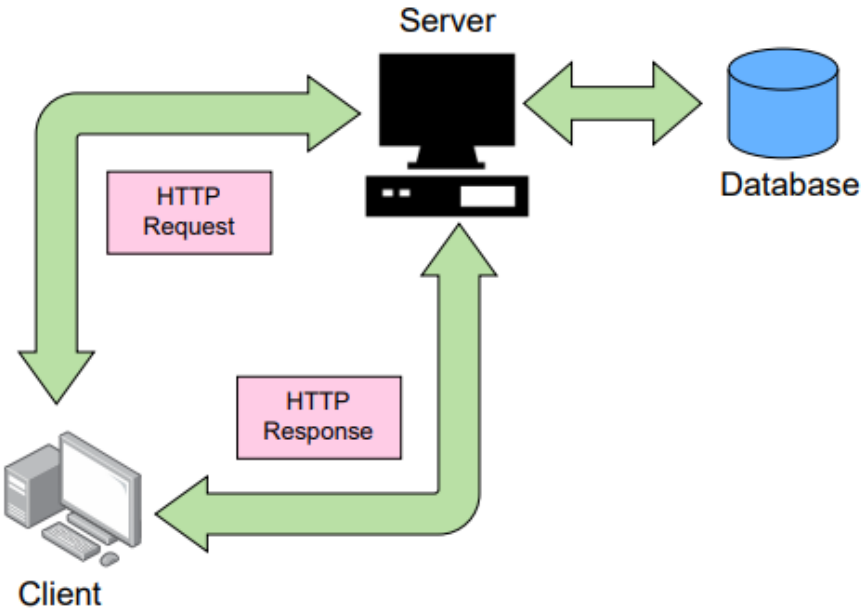
- **User Interface:** The user interface is designed to allow users to input the Address specific information they want to look up. It is designed to be intuitive and user-friendly.
- **Load Balancer:** A load balancer is a component that distributes incoming network traffic across multiple servers to ensure that no single server is overwhelmed with requests. This helps to improve the performance and availability of the application.
- **API Gateway:** It will handle the routing of requests to the appropriate microservice based on the endpoint and the HTTP method. For instance, it routes to Address Format service to get the format specific information of a particular country. It routes to Address Service to get the full addresses for the specified address parameters.
- **API Logging**: API logging is the process of capturing and storing information about requests and responses to an API. This includes details such as the request method, URL, headers, payload, response status code, and response body.
- **Business Logic:** This is where the search filter and the address_format method is built which is used to query the database to get the appropriate address.
- **Database:** Repository to store the address and format specific data.
- **Cache:** The code uses a cache to store search results. This will reduce the response time of a request.

Connector:

- MongoDB driver: A connector that allows the Flask application to communicate with the MongoDB database.

ARCHITECTURAL STYLE

RESTful API design style can be chosen for this system. REST is a widely used architectural style for web services due to its simplicity, scalability, and flexibility. It uses standard HTTP protocols for communication, which makes it easy to develop, test, and deploy web applications.



Moreover, RESTful APIs allow the separation of concerns between the client and server. The client can focus on the user interface and user experience, while the server can focus on data storage, processing, and security. This separation of concerns helps to keep the system modular, extensible, and maintainable over time.

The REST (Representational State Transfer) architecture is a popular choice for building web applications because it offers several benefits, including:

Scalability: REST is designed to be scalable, allowing resources to be distributed across multiple servers and enabling the creation of highly scalable and reliable systems.

Flexibility: REST allows developers to create APIs that can be accessed by a wide range of devices and applications, including web browsers, mobile devices, and IoT devices.

Simplicity: The REST architecture is based on a few simple principles, making it easy to understand and implement. This simplicity makes it easier for developers to create and maintain RESTful APIs.

Caching: REST allows responses to be cached, improving performance and reducing the load on servers.

Security: REST includes built-in security features such as SSL encryption and OAuth authentication.

Compatibility: REST is compatible with a wide range of programming languages and frameworks, making it easy to integrate with existing systems.

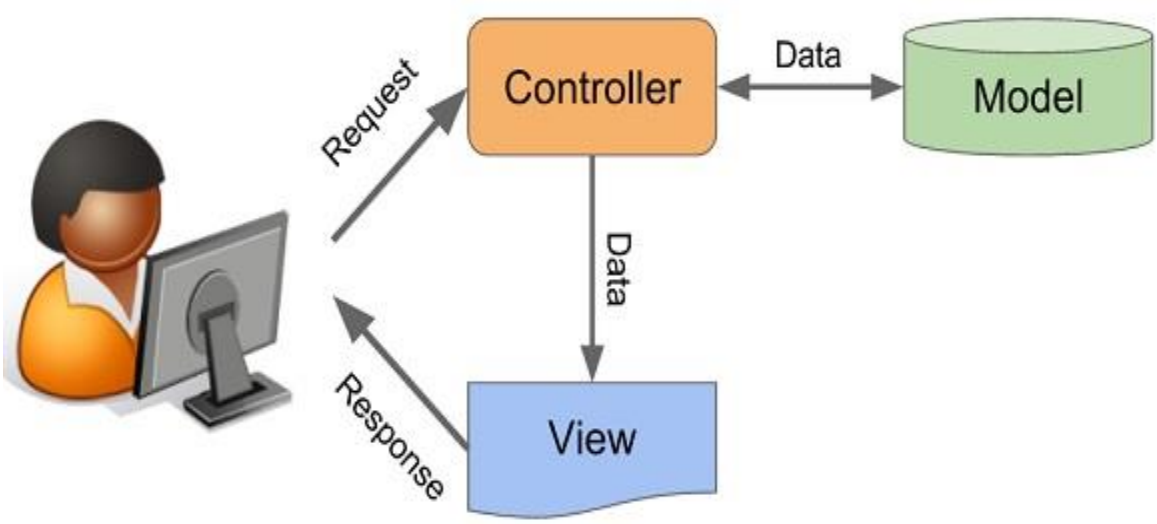
Overall, the REST architecture is a powerful and flexible approach to building web applications, offering a range of benefits that make it an ideal choice for many developers and organizations.

ARCHITECTURAL PATTERN

In our application we have used Model-view-controller architecture.

In general, Model–view–controller (MVC) is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.

Traditionally used for desktop graphical user interfaces (GUIs), this pattern became popular for designing web applications. Popular programming languages have MVC frameworks that facilitate the implementation of the pattern.



We have three components:

The view is responsible for rendering the UI elements and handling user interactions. This is represented by the HTML/CSS and JavaScript code that interacts with the DOM.

The model is responsible for managing the data and application logic. This is represented by the backend code that handles the API requests and responses, which is in routes.py.

The controller is responsible for managing the flow of data between the view and model components. This is represented by the JavaScript code that listens for user events and sends API requests based on user input, which is in Scripts.js

Overall, this follows the principles of MVC architecture by separating the UI from the business logic and providing a clear separation of concerns between the different components.

ASSUMPTIONS

- The input Parameters such as Street, country, state etc. are provided by the client in valid and well-formed format.
- We are limiting to just **5 countries** for our application.
- API should be secured; this can be done using authorization service for the user.
- API should be scalable and available, it should be able to handle large number of requests and should not face downtime, this can be implemented using load balancer.
- API should be efficient and deployable, For the API to be efficient we can use caching where we can store the frequently requested addresses, this will reduce the response time.
- Query optimizations should be done so that the results we get are accurate. For our API to be deployable, we may have to follow containerization such as using Docker can make the process of deployment easier and efficient.
- Choosing the cloud platforms that best fits the API requirement and host our API in the cloud (AWS, Azure, google cloud platform)

TRADE - OFF

One potential tradeoff in this code is the use of logging. While logging can be useful for debugging and monitoring application behavior, it can also add overhead and increase the complexity of the code. Additionally, logging sensitive data, such as request parameters, can be a security risk if not handled carefully.

TOOLS AND TECHNOLOGIES:

- **Web Framework:** FLASK
- **Backend:** Python
- **Frontend:** HTML, CSS, JAVASCRIPT
- **Database:** Mongo DB
- **Load Balancer:** Nginx
- **Load Testing Tool:** Apache JMeter

API SETUP

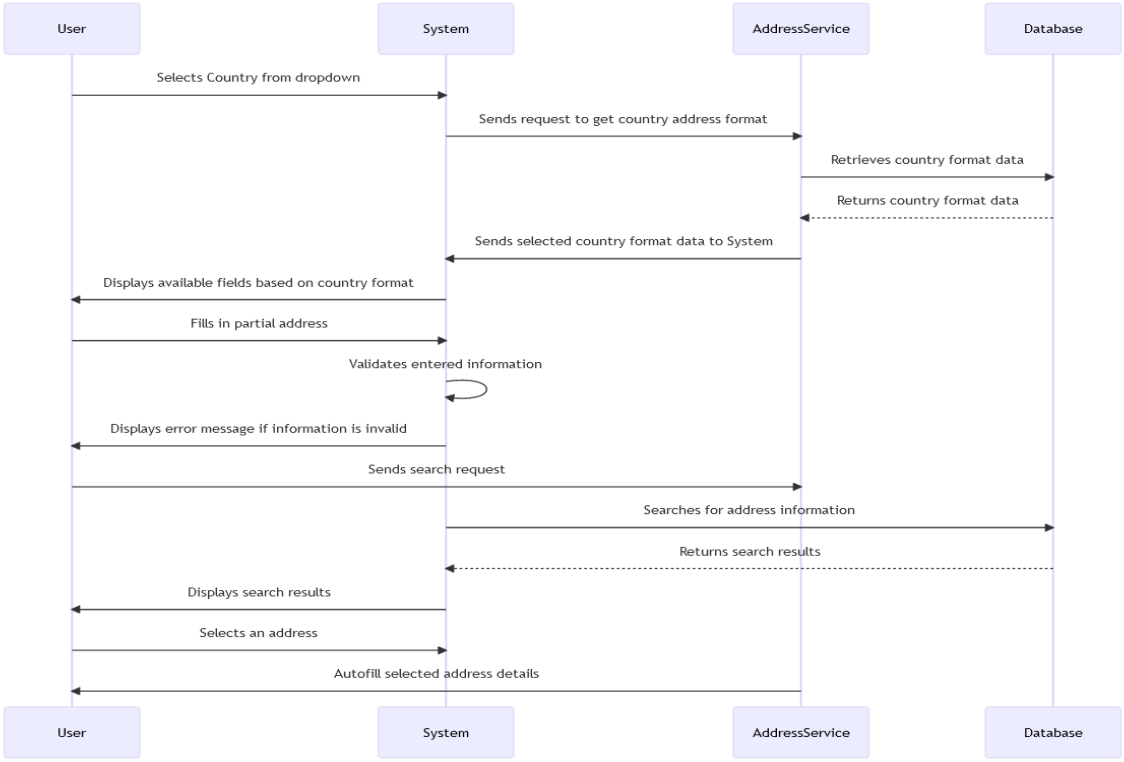
Flask Installation: pip install Flask.

Mongo DB : pip install Flask-Pymongo

Flask-Caching: pip install Flask-Caching

Flask-Swagger: pip install flask-swagger-ui

SEQUENCE DIAGRAM



COMMUNICATION PROTOCOL

We are using HTTP communication protocol. HTTP (Hypertext Transfer Protocol) is a communication protocol used for exchanging data between client and server over the internet or any network. It is the foundation of data communication for the World Wide Web.

The HTTP protocol uses a request-response model where the client sends a request message to the server, which then sends a response message back to the client. The request message contains a request method (e.g. GET, POST, PUT, DELETE), a URI (Uniform Resource Identifier), and headers. The response message contains a status code, headers, and a response body.

DATA STORAGE

Data is stored in a separate data store component as shown in the system context diagram above and in component diagram. The data store component contains information about street, postal code, city, state, country.

	A	B	C	D	E	F	G	H	I	J
1	recipient_name	street	city	distrcit	region	postal_co	state	country	country_c	state_code
2	Kai Le	10255 132	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
3	Robert Patel	13109 NE	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
4	Cameron Lo	13037 NE	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
5	Harper Castillo	13029 NE	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
6	Harper Dominguez	13021 NE	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
7	Ezra Vu	13013 NE	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
8	Jade Hu	13005 NE	KIRKLAND	KING		98033	WASHING	UNITED ST	US	WA
9	Miles Chang	17102 SE	BELLEVUE	KING		98006	WASHING	UNITED ST	US	WA
10	Gianna Holmes	17100 SE	BELLEVUE	KING		98006	WASHING	UNITED ST	US	WA

Data is stored in Mongo DB as 2 collections one is address_collection for storing addresses of different countries. The other collection is add_formats for storing different country formats with the field name and placeholders

Address_collection – document structure

```
1   _id: 640bdc35c7033d2a4597cf4b                    ObjectId
2   recipient_name: "Kai Le"                         String
3   street: "10255 132ND AVE NE"                     String
4   city: "KIRKLAND"                                 String
5   distrcit: "KING"                                 String
6   region: "    "                                   String
7   postal_code: "98033"                             String
8   state: "WASHINGTON"                              String
9   country: "UNITED STATES"                         String
10  country_code: "US  "                             String
11  state_code: "WA  "                               String
```

Address_formats- document structure

```
1   _id: ObjectId('640bdc75c7033d2a4597cf4c')        ObjectId
2   country_code: "US  "                             String
3 ⌄ address_format: Array                            Array
4   ⌄ 0: Object                                      Object
5     ⌄ recipient_name: Array                        Array
6         0: "Recipient Name"                        String
7         1: "Rachel Green"                          String
8   ⌄ 1: Object                                      Object
9     ⌄ street: Array                                Array
10        0: "Street Address"                        String
11        1: "901 12 ave"                            String
12  ⌄ 2: Object                                      Object
13    ⌄ city: Array                                  Array
14        0: "City"                                  String
15        1: "Seattle"                               String
16  ⌄ 3: Object                                      Object
17    ⌄ state: Array                                 Array
18        0: "State"                                 String
19  + 1: "Washington"                                String
20  ⌄ 4: Object                                      Object
21  + ⌄ postal_code: Array                           Array
22        0: "Postal Code"                           String
23        1: "98040"                                 String
```

Data moves between the client and the API server in the form of HTTP requests and responses, while data moves between the API server and the address database in the form of queries and results. The API server also performs validation on the input parameters to ensure they are valid and consistent with the API specifications.

API IMPLEMENTATION

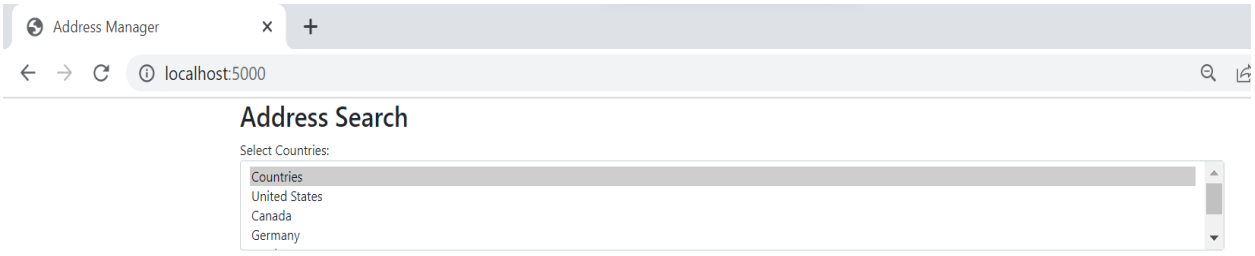
This is a Flask API implementation that provides endpoints for searching and formatting addresses. It uses MongoDB as the database to store address information and Flask-Cache to cache the search results.

The main functionalities of this application are:

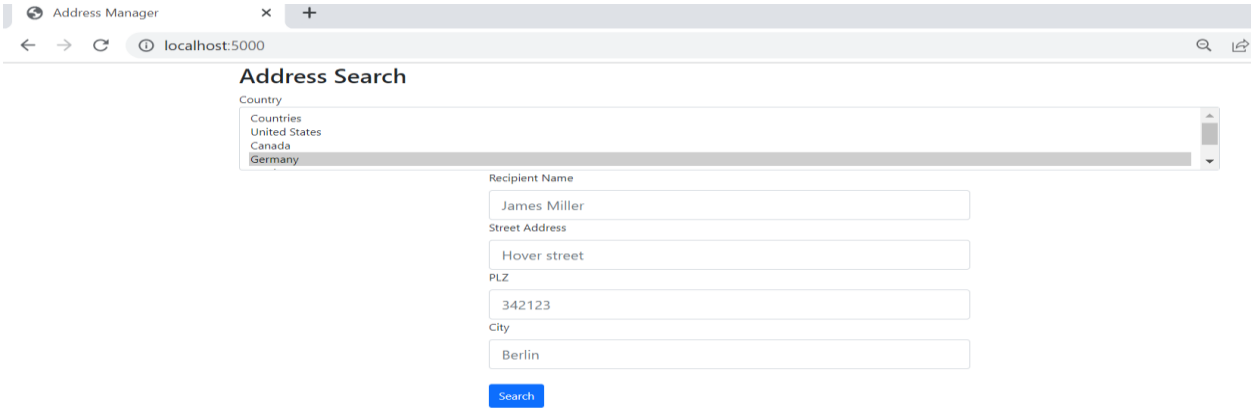
- The API logs requests and responses to a file using the RotatingFileHandler from Python's logging module. By logging events and data related to requests and responses, we can easily troubleshoot and identify the root cause of problems, and improve the overall quality of the application.
- The /address_API/search route handles the address search functionality. It retrieves search parameters from the request URL and builds a filter query to search for addresses in the MongoDB database. It then checks if the search has already been cached and retrieves the cached results if available. If not, it queries the database, caches the results, and returns the addresses in JSON format.
- Once the addresses are returned they are displayed in the table format, if any of the table row is chosen, the form is auto populated with that particular address.
- The /address_API/addressformat/<country_code> route returns the address format for the given country code by querying the MongoDB database. The User Interface is dynamically changed according to the chosen country and the format returned from this endpoint. However, if multiple countries are chosen a country agnostic form would be displayed.

Overall, the implementation follows the typical request-response cycle of a web API, handling incoming requests, querying the database, and returning responses in JSON format. It also includes caching functionality to improve performance and logging to track API activity.

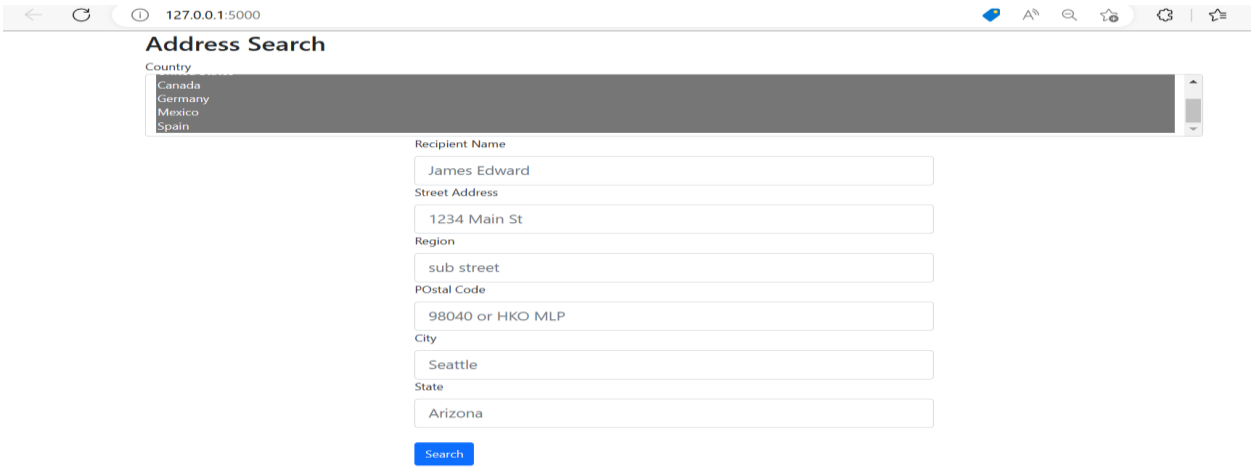
USER INTERFACE

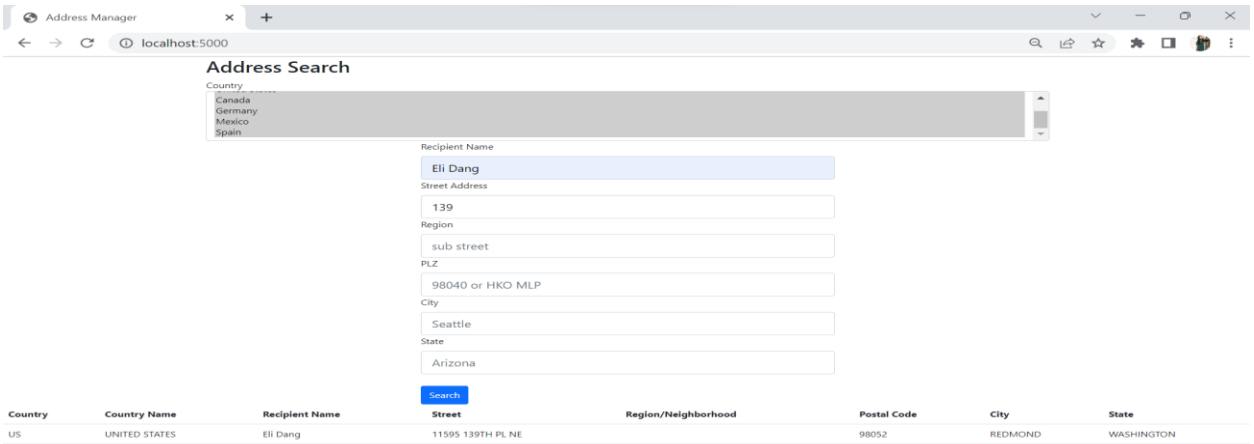


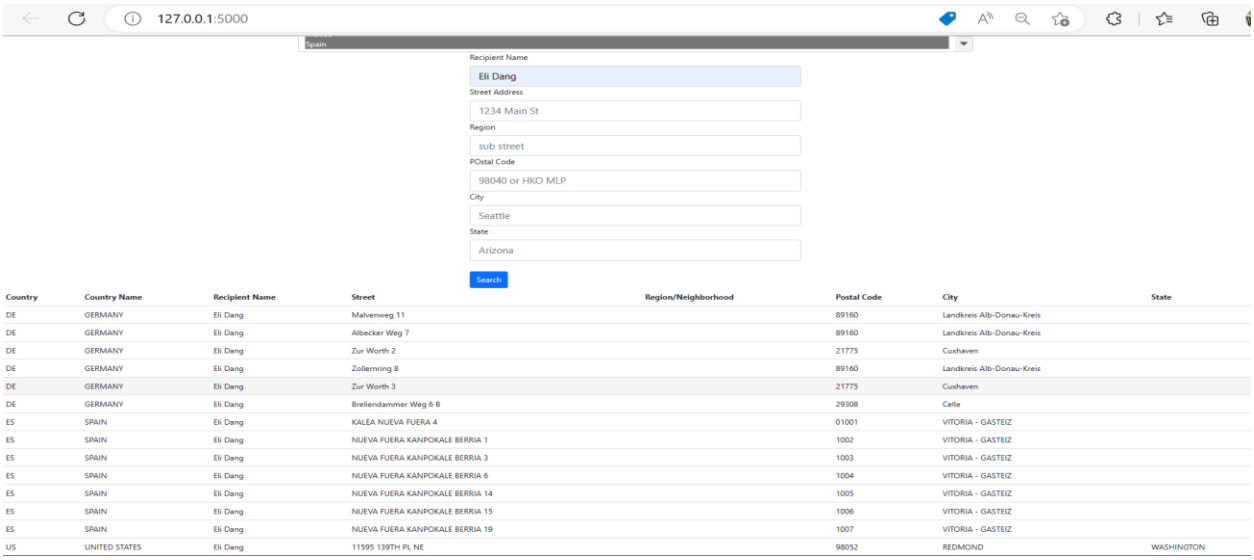
If Only one country Germany is chosen



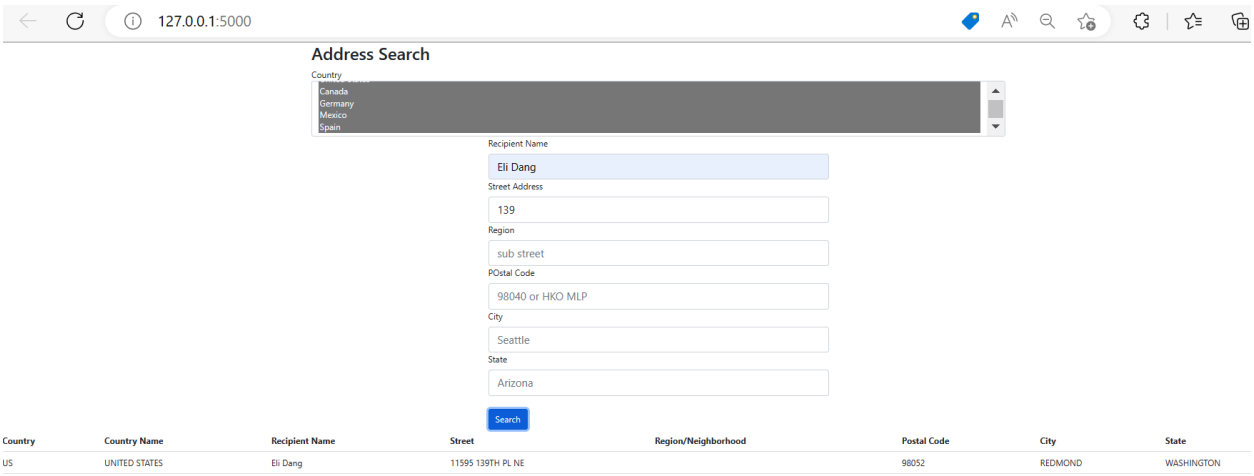
If Multiple countries are chosen Country Agnostic form is displayed







Country	Country Name	Recipient Name	Street	Region/Neighborhood	Postal Code	City	State
DE	GERMANY	Eli Dang	Malvenweg 11		89160	Landkreis Alb-Donau-Kreis	
DE	GERMANY	Eli Dang	Albecker Weg 7		89160	Landkreis Alb-Donau-Kreis	
DE	GERMANY	Eli Dang	Zur Worth 2		21775	Cuxhaven	
DE	GERMANY	Eli Dang	Zollernring 8		89160	Landkreis Alb-Donau-Kreis	
DE	GERMANY	Eli Dang	Zur Worth 3		21775	Cuxhaven	
DE	GERMANY	Eli Dang	Breliendammer Weg 6 B		29308	Celle	
ES	SPAIN	Eli Dang	KALEA NUEVA FUERA 4		01001	VITORIA - GASTEIZ	
ES	SPAIN	Eli Dang	NUEVA FUERA KANPOKALE BERRIA 1		1002	VITORIA - GASTEIZ	
ES	SPAIN	Eli Dang	NUEVA FUERA KANPOKALE BERRIA 3		1003	VITORIA - GASTEIZ	
ES	SPAIN	Eli Dang	NUEVA FUERA KANPOKALE BERRIA 6		1004	VITORIA - GASTEIZ	
ES	SPAIN	Eli Dang	NUEVA FUERA KANPOKALE BERRIA 14		1005	VITORIA - GASTEIZ	
ES	SPAIN	Eli Dang	NUEVA FUERA KANPOKALE BERRIA 15		1006	VITORIA - GASTEIZ	
ES	SPAIN	Eli Dang	NUEVA FUERA KANPOKALE BERRIA 19		1007	VITORIA - GASTEIZ	
US	UNITED STATES	Eli Dang	11595 139TH PL NE		98052	REDMOND	WASHINGTON



Country	Country Name	Recipient Name	Street	Region/Neighborhood	Postal Code	City	State
US	UNITED STATES	Eli Dang	11595 139TH PL NE		98052	REDMOND	WASHINGTON

CODE SNIPPETS

Below is the code snippets of the two end points logic one is the search function other is the address_format. In the Search function we have used Dictionary of filters which has Regular expressions for each field to be searched and I am using Mongo Db $IN parameter to filter with respect to countries. After creating the whole filter we are creating a Search Query using Mongo DB's "find" operation. This gives us the addresses with respect to the filter generated by giving the partial address.

```python
@main.route('/address_API/search', methods=['GET'])
##@cache.memoize(360)
def search_address():
    print("hello")
    current_app.logger.info('/address_API/search is called')
    country_code =  json.loads(request.args.get('country_code'))
    recipient_name = request.args.get('recipient_name')
    street = request.args.get('street')
    city = request.args.get('city')
    state = request.args.get('state')
    postal_code = request.args.get('postal_code')
    print(country_code)
    filters = {'country_code': {'$in': country_code}}

    if recipient_name:
        filters['recipient_name'] = {'$regex':  f'^{recipient_name}$', '$options': 'i'}
    if street:
        filters['street'] = {'$regex': f'.*{street}.*', '$options': 'i'}
    if city:
        filters['city'] = {'$regex': f'.*{city}.*', '$options': 'i'}
    if state:
        filters['state'] = {'$regex': f'.*{state}.*', '$options': 'i'}
    if postal_code:
        filters['postal_code'] = {'$regex': f'.*{postal_code}.*', '$options': 'i'}
    current_app.logger.info('Cache key: %s', str(request.args))
    addresses = cache.get(str(request.args))

    if addresses is None:
        current_app.logger.info('Cache miss: %s', str(request.args))
        addresses = list(mongodb.db.address_collection.find(filters))
        for address in addresses:
            address['_id'] = str(address['_id'])
        cache.set(str(request.args), addresses)

    if len(addresses) == 0:
        # return 404 Not Found status code and error message
        return jsonify({'error': 'No addresses found for the given criteria.'}), 404

    return jsonify(addresses), 200
```

```python
@main.route('/address_API/addressformat/<country_code>')
def get_address_format(country_code):
    current_app.logger.debug('/address_API/addressformat/<country_code>')
    addressformat = mongodb.db.address_formats.find_one({'country_code': country_code})
    print(f
    format=addressformat['address_format']
    print(format)

    return jsonify(format)
```

(variable) addressformat: Any

GITHUBLINK
https://github.com/madhuroopa/AddressManager

LITTLE LANGUAGE:

Example1: Scenario demonstrating address_format retrieval

API end point : http://{base_url}/address_API/address_format<country_code>

Http Request: http://127.0.0.1:5000/address_API/addressformat/MX

Response:

[{"recipient_name":["Recipient Name","James Miller"]},{"street":["Street Address","10-123 1/2 MAINSTREET"]},{"region":["Neighborhood","CENTRO"]},{"city":["City","TLALPAN"]},{"state":["Province","VERACRUZ"]},{"postal_code":["PostalCode","77520"]},{"country_codes":["Country",""]}]

Example1: Scenario demonstrating single country search

API end point for address search: http://{base_url}/address_API/search{}

HTTP Request: http://127.0.0.1:5000/address_API/search?recipient_name=Eli Dang&street=139&city=&state=&postal_code=&country_code=["US"]

Response: 200, [{

"_id": "63fde1a148d208dccfe0c00b",

"city": "REDMOND",

"country": "UNITED STATES",

"country_code": "US",

"distrcit": "KING",

"postal_code": "98052",

"recipient_name": "Eli Dang",

"region": "",

"state": "WASHINGTON",

"state_code": "WA",

"street": "11595 139TH PL NE"

}]

Example 2: Scenario demonstrating multiple country search

API end point for address search: http://{base_url}/address_API/search{}

Http Request : http://127.0.0.1:5000/address_API/search?recipient_name=Eli Dang&street=&city=&state=&postal_code=&country_code=["US","CA","DE","MX","ES"]

Response:

[{

"_id": "640a26df0ae6fbe6942db71e",

"city": "Landkreis Alb-Donau-Kreis",

"country": "GERMANY",

"country_code": "DE",

"distrcit": "",

"postal_code": "89160",
"recipient_name": "Eli Dang",
"region": "",
"state": "",
"state_code": "",
"street": "Malvenweg 11"
},
{
"_id": "640a26df0ae6fbe6942db723",
"city": "Landkreis Alb-Donau-Kreis",
"country": "GERMANY",
"country_code": "DE",
"distrcit": "",
"postal_code": "89160",
"recipient_name": "Eli Dang",
"region": "",
"state": "",
"state_code": "",
"street": "Albecker Weg 7"
}, ……multiple records up to 14]

Example 3: Addresses not found scenario

API end point for address search: http://{base_url}/address_API/search{}

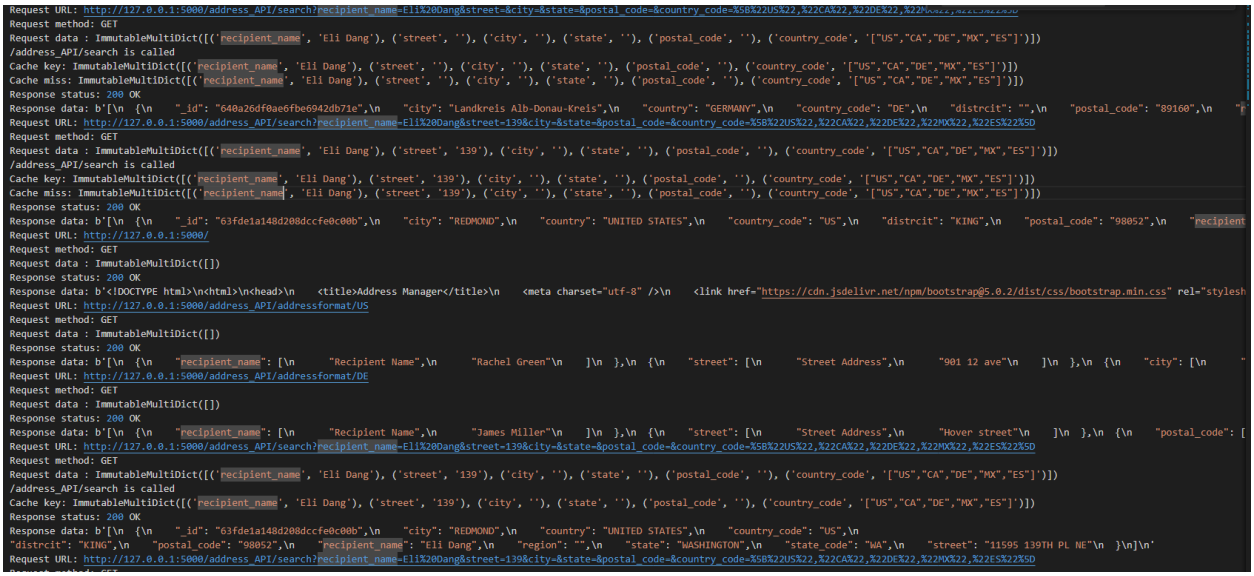
Http Request : http://127.0.0.1:5000/address_API/search?recipient_name=Eli &street=&city=&state=&postal_code=&country_code=["US","CA","DE","MX","ES"]

Response: 400, {

"error": "No addresses found for the given criteria."
}

API LOGGING

Log File:



LOAD BALANCER AND LOAD TESTING

LOAD BALANCER Tool: Nginx

LOAD TESTING Tool: Apache JMeter

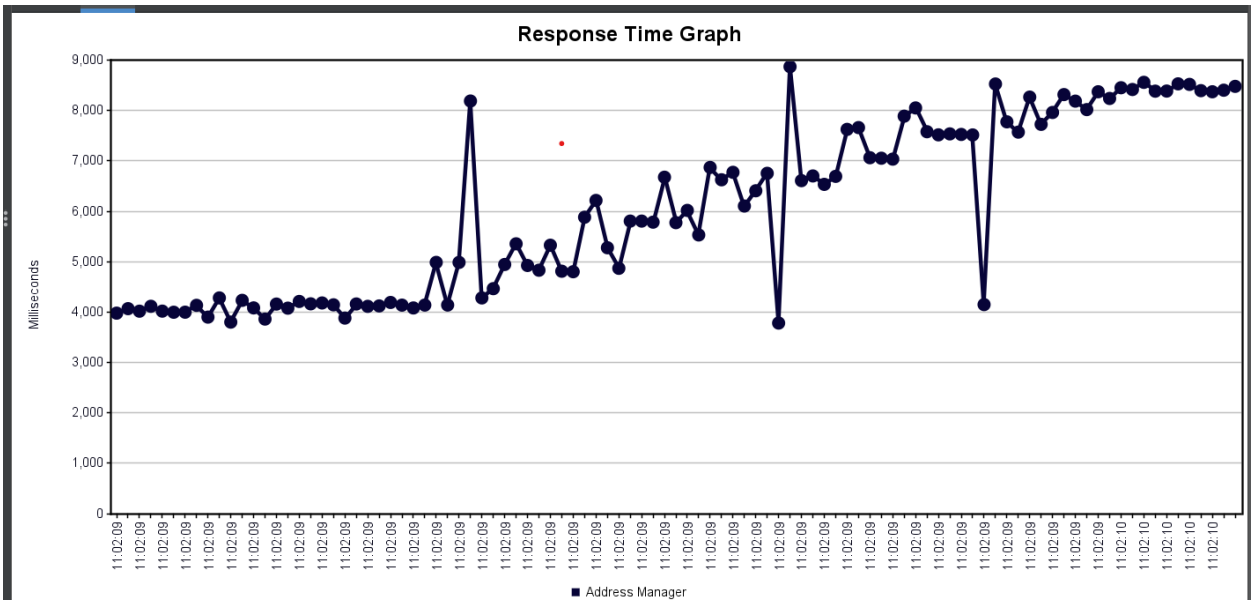
Apache JMeter is a popular open-source tool for load testing, performance testing, and functional testing of applications.

NGINX is a popular open-source web server, reverse proxy, and load balancer software that is widely used for serving static and dynamic content on the web. NGINX is known for its high performance, scalability, and low memory footprint. It can handle large amounts of traffic while consuming very few resources, making it ideal for high-traffic websites and web applications.

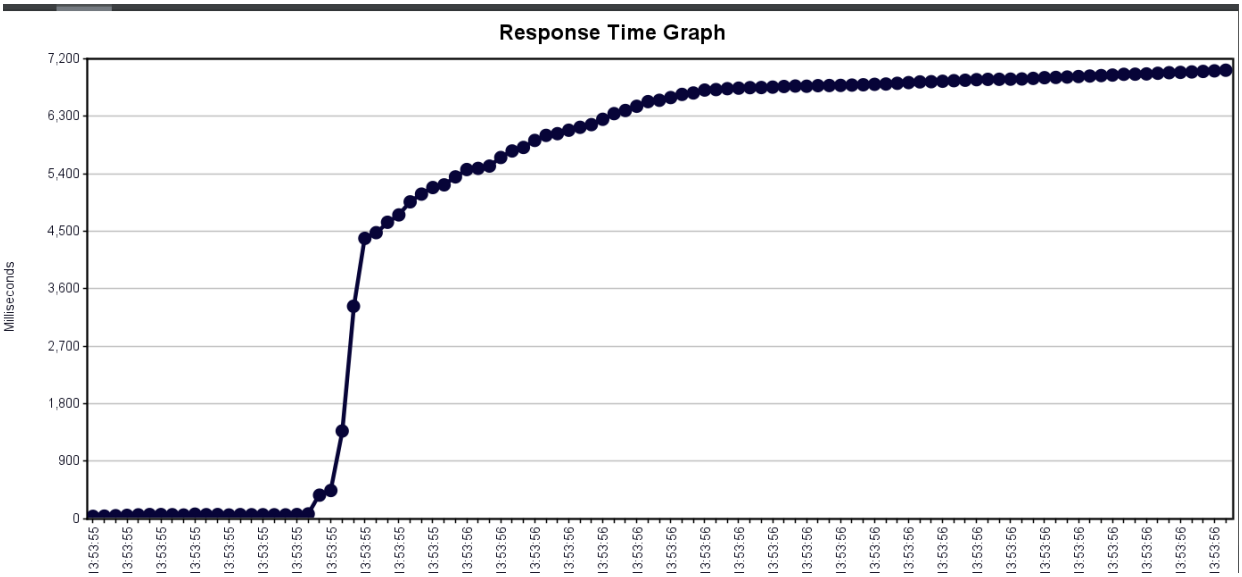
We setup a Docker-Compose.yaml file, Docker File and Nginx.confg file. There are three services defined one is the addressmanager-app service which is bind to the application implementation, the second one is the db service which is responsible for running MongoDB instance. The last service is the nginx service, this runs an NGINX reverse proxy in a Docker container. It listens on port 80 and forwards incoming HTTP requests to the addressmanager-app service.

In conclusion, The application is run as a set of Docker containers, with NGINX acting as a reverse proxy (creating about 5 servers) to route incoming requests to the appropriate server.

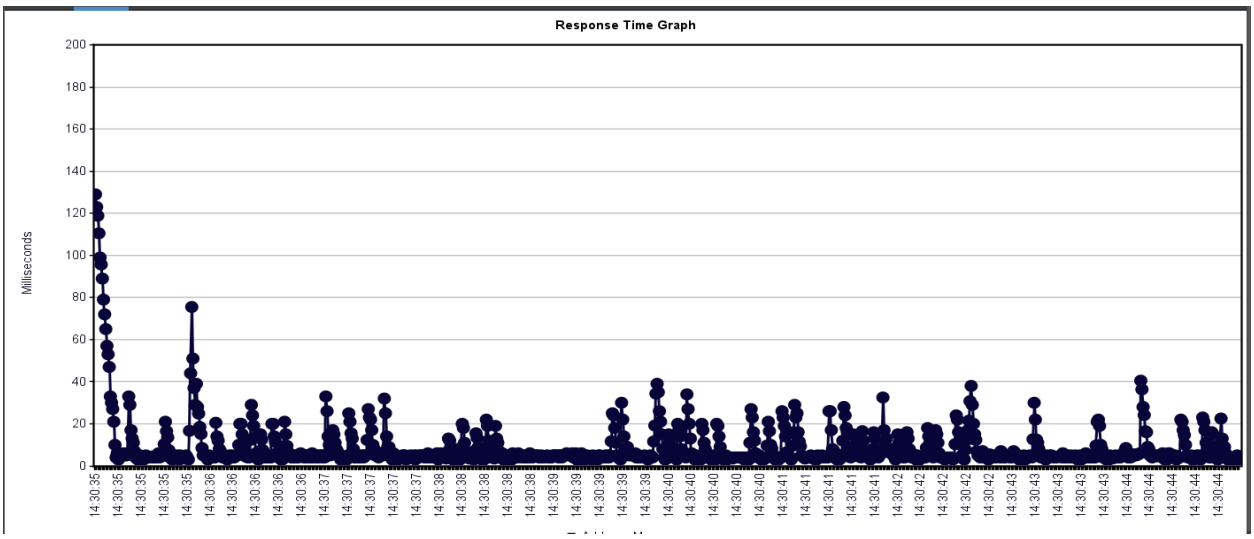
Response Time of an API without Cache implementation running on the local flask server for 100 requests:



Response Time of an API with Cache implementation running on the local flask server for 100 requests:



Response Time of an API with cache implementation running on Nginx server but only on one server instance for 1000 requests

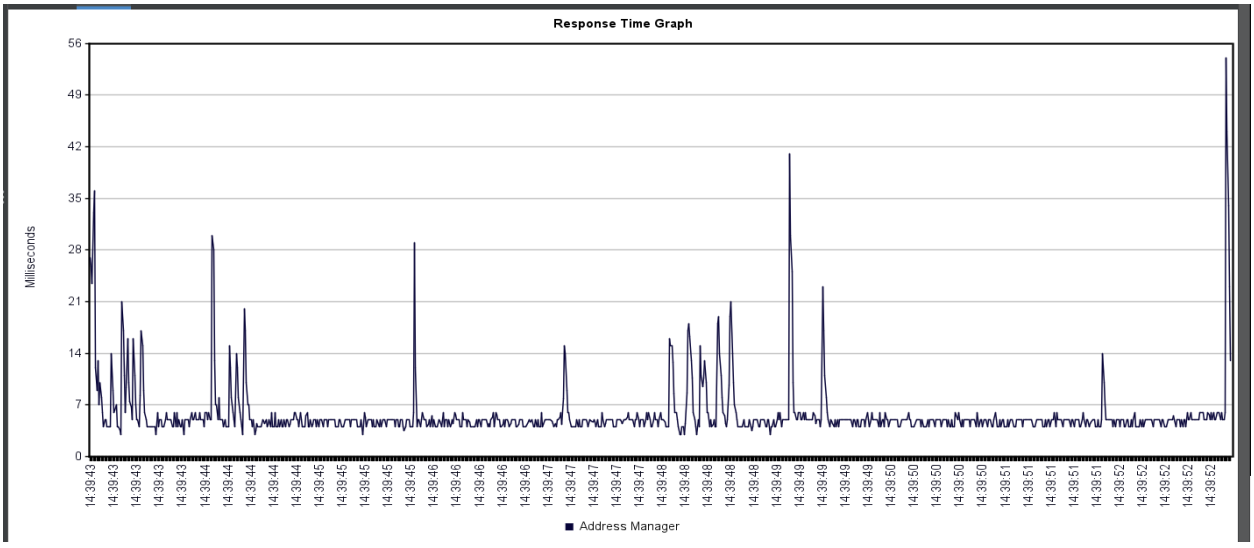


Summary of Response stats :

Here the min time is 3 ms and max time is 129 ms

A	B	C	D	E	F	G	H	I	J	K
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughpı	Received K	Sent KB/se	Avg. Bytes
Address M	1000	8	3	129	12.72	0.00%	99.99	340.1	25.78	3483
TOTAL	1000	8	3	129	12.72	0.00%	99.99	340.1	25.78	3483

Response Time of an API with cache implementation running on Nginx server but using 5 server instance to balance the load for 1000 requests



Summary of response stats:

Here as there are 5 servers the load is balanced and the min response time and max response time is also decreased to 3 ms and 54 ms with an average of 5ms.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp	Received K	Sent KB/se	Avg. Bytes
Address M	1000	5	3	54	4.15	0.00%	99.99	340.1	25.78	3483
TOTAL	1000	5	3	54	4.15	0.00%	99.99	340.1	25.78	3483

NON FUNCTIONAL REQUIREMENTS

Performance: As the application is using caching which improves application performance by storing frequently accessed data in memory or disk to reduce the number of database or API calls. Nginx is a high-performance web server and reverse proxy that can handle a large number of concurrent connections. By using Nginx as a load balancer, the application can achieve better performance and lower latency.

Scalability: As the application is deployed behind a load balancer, it can scale horizontally by adding more instances of the application server. This ensures that the application can handle increasing traffic without performance degradation.

High Availability: By using a load balancer, the application can achieve high availability. In case of a failure of one of the application servers, the load balancer can redirect the traffic to the healthy servers, ensuring uninterrupted service.

Reliability: As the application is using logging, This can help to analyze and debug issues, identify bottlenecks, monitor application usage and make the API more reliable for future usage.

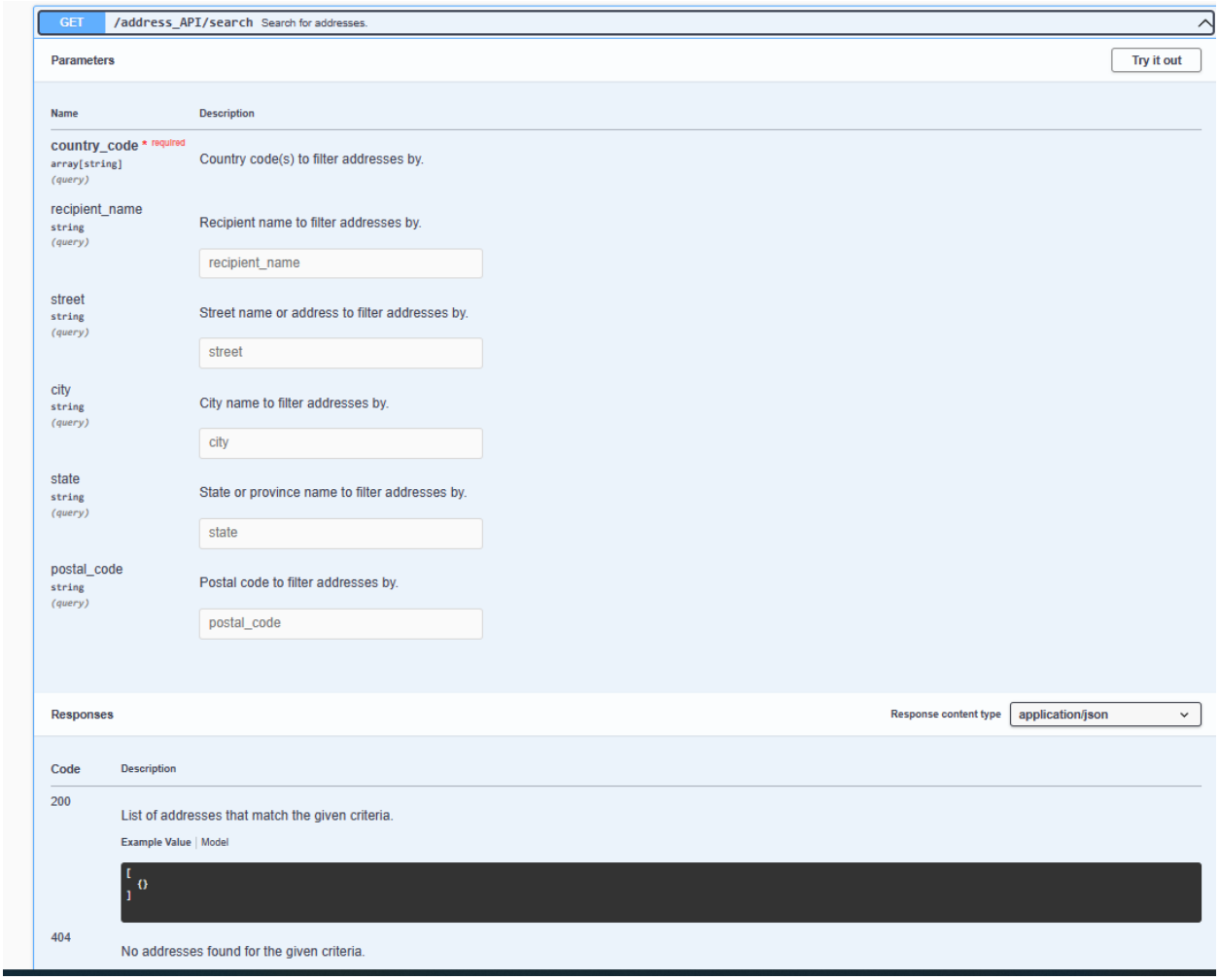
Usability: Overall using a well-defined user interface, API logging, caching and load balancing it will surely increase the usability of the API.

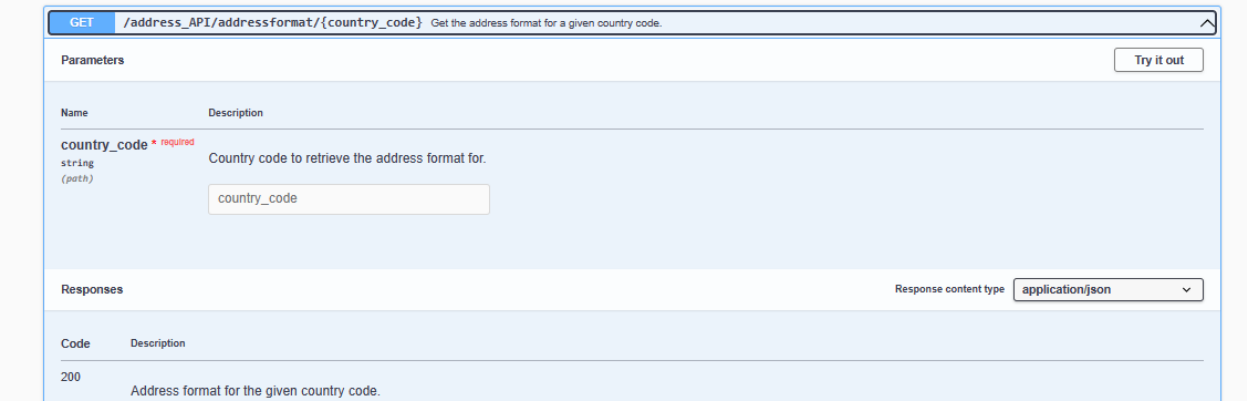
API DOCUMENTATION

API documentation is implemented using swagger

Swagger URL: http://127.0.0.1:5000/api/docs/


127.0.0.1:5000/api/docs/#/
Swagger
/static/swagger.json
Explore
Address API
1.0
/static/swagger.json
default
GET /address_API/search Search for addresses.
GET /address_API/addressformat/{country_code} Get the address format for a given country code.


GET /address_API/search Search for addresses.
Parameters
Try it out
Name
Description
country_code * required
array[string]
(query)
Country code(s) to filter addresses by.
recipient_name
string
(query)
Recipient name to filter addresses by.
recipient_name
street
string
(query)
Street name or address to filter addresses by.
street
city
string
(query)
City name to filter addresses by.
city
state
string
(query)
State or province name to filter addresses by.
state
postal_code
string
(query)
Postal code to filter addresses by.
postal_code
Responses
Response content type
application/json
Code
Description
200
List of addresses that match the given criteria.
Example Value | Model
[
{}
]
404
No addresses found for the given criteria.



REFERENCES

Pymongo. Flask. (n.d.). https://flaskpymongo.readthedocs.io

Logging. Logging - Flask Documentation (2.2.x). (n.d.).https://flask.palletsprojects.com/en/2.2.x/logging/

Caching. Flask. (n.d.). https://flask-caching.readthedocs.io/en/latest/

Beginner's Guide. nginx. (n.d.). http://nginx.org/en/docs/beginners_guide.html

Mongo DB. MongoDB Documentation. (n.d.). https://www.mongodb.com/docs/