# API Exploration Report

Hardik Chauhan (21EC39011)
IIT Kharagpur

March 2025

## 1 Introduction

The goal of this project was to extract all possible names from an autocomplete API hosted at "`http://35.200.185.69:8000`". Since no official documentation was provided, the approach involved systematic exploration to understand the API behavior, discover endpoints, and handle potential constraints like rate limiting.

## 2 API Exploration Process

### 2.1 Initial Testing

The starting point was the known endpoint:

- v1 Autocomplete API: `"/v1/autocomplete?query=<string>"`
- v2 Autocomplete API: `"/v2/autocomplete?query=<string>"`
- v3 Autocomplete API: `"/v3/autocomplete?query=<string>"`

***Initial queries tested how the API responded to different input strings. Early observations included:***

**1. v1 Autocomplete API:**

- A maximum of 10 names are returned per API call (if there are 'n' names with the given prefix, the API returns $\min(n, 10)$ names).
- Names consist of lowercase alphabets ('a' to 'z').
- Names are returned in lexicographical order (dictionary order).

**2. v2 Autocomplete API:**

- A maximum of 12 names are returned per API call (if there are 'n' names with the given prefix, the API returns $\min(n, 12)$ names).
- Names consist of lowercase alphabets ('a' to 'z') and numbers ('0' to '9').
- Names are returned in lexicographical order, with numbers taking precedence over alphabets.

**3. v3 Autocomplete API:**

- A maximum of 15 names are returned per API call (if there are 'n' names with the given prefix, the API returns $\min(n, 15)$ names).
- Names consist of lowercase alphabets ('a' to 'z'), numbers ('0' to '9'), and special characters (' ', '+', '-', '.').
- Names are returned in lexicographical order, with special characters having the highest precedence, followed by numbers, and then alphabets.

### 2.2 Systematic Query Generation

To ensure the complete extraction of names, an optimized recursive tree-based strategy was employed, adapting to different API versions and name structures:

- **Start with an initial set of characters:** For v1, lowercase alphabets ('a' to 'z'). For v2, lowercase alphabets and numbers ('a' to 'z', '0' to '9'). For v3, lowercase alphabets, numbers, and special characters ('a' to 'z', '0' to '9', ' ', '+', '-', '.').
- **Recursive Query Expansion:** For each prefix, retrieve the list of returned names. Extend the query by appending the next unseen character from each result. This ensures deeper traversal into longer names without redundant queries.
- **Handling Partial Results:** Since each API version limits the number of names per response (10, 12, and 15 for v1, v2, v3 respectively), the script ensures complete extraction by continuing to query with extended prefixes until no new names are returned.

This approach maximizes coverage, minimizes redundant requests, and adapts dynamically to variable name structures and lengths in different API versions.

## 3 Handling Rate Limiting

Several rate limiting behaviors were observed, including:

- Error 429: Too Many Requests error after a burst of rapid queries.
- A cooldown period is required before the API resumed responding.

### 3.1 Solution to Rate Limiting

The script implements the following rate limiting strategy:

- Automatically detect 429 responses.
- Back off with exponential delays (e.g., for v1: 16s - 32s - 64s..., for v2 and v3: 40s - 80s - 160s...).
- Log failed queries and retry them later.

## 4 Problems Faced and Their Solutions

### 4.1 Encoding Issues with Special Characters

**Problem:** In `v3`, certain characters like `' '` (space) and `'+'` needed to be encoded to be correctly interpreted by the system.
**Solution:** Implemented URL encoding to replace these characters with their respective encoded values before making requests.

### 4.2 Repetition in Output Names

**Problem:** Duplicate results in output lists due to repeated API calls.
**Solution:** Used an `early_list` to track and filter previously encountered outputs, ensuring only unique results were stored.

### 4.3 Rate Limiting Issues

**Problem:** Frequent requests to the API led to rate limiting, causing failures in fetching results.
**Solution:** Introduced exponential backoff with retries, pausing requests for increasing durations when a rate limit error was encountered.

## 5 Results

### 5.1 Total Number of Requests

After optimizations and rate handling, the total number of requests made:

**v1: 12,897**
**v2: 3,108**
**v3: 2,602**

### 5.2 Total Names Collected

The total number of unique names extracted:

**v1: 18,632**
**v2: 13,730**
**v3: 12,517**

## 6 Code

For code you can refer to the python notebooks in the repository (v1.ipynb, v2.ipynb, v3.ipynb). These notebooks are fully documented.

## 7 Conclusions and Learnings

The project demonstrated the importance of:

- Systematic exploration when dealing with undocumented APIs.
- Adapting to rate limiting through retry and backoff strategies.
- Balancing completeness and efficiency in data extraction.