ChatGPT

Implementing a US Address Validation & Standardization API

Overview of the Solution Approach

To validate and standardize free-form addresses, the backend will rely on a geocoding/validation service rather than writing a parser from scratch. The API (e.g. a Fastify TypeScript POST route) will accept a raw address string and use a third-party address API to obtain a **normalized, structured address**. The response will include separate fields for street name, street number, city, state, and ZIP code. We will also include a status indicator to flag whether the input address was found **valid**, was **corrected** (e.g. typos fixed or components added), or is **unverifiable**.

The service will be restricted to U.S. addresses, which simplifies the task. Many geocoding APIs allow filtering by country (for example, adding country code "US" in the query) to improve accuracy and avoid non-US results. The chosen solution must handle edge cases gracefully – for instance, **partial addresses** (missing components like ZIP or house number) and **minor misspellings/typos** – by attempting to infer or correct missing/wrong parts. If the address can be resolved to a single likely result, the API will return the cleaned address; if multiple candidates are possible, the system will apply criteria (explained below) to pick the best match or mark the result as ambiguous.

Using Google Maps APIs for Address Validation

One robust approach is to leverage Google's mapping APIs, which are known for their accuracy and rich data. Google offers two relevant APIs:

- **Google Geocoding API** – Given a free-form address, this API returns the geographic coordinates and the address broken into components (street number, route, locality, state, postal code, etc.) [1] [2]. We can use the returned `address_components` array to extract the structured fields (Google explicitly advises parsing these components rather than the one-line formatted address [1]). The Geocoding API is very good at interpreting messy input, often handling misspellings or incomplete addresses. It may return multiple results if the query is ambiguous, but typically the **best-matching result is first**. We can restrict results to the US by specifying a region or component filter (e.g. componentRestrictions with country=US). The Geocoding API also sets a `partial_match` flag when the result is not an exact full match for the input [3]. For example, if the user provides an incomplete address, Google might return a result with `partial_match: true` indicating it had to guess or could only match part of the input. We can use this to detect corrections or missing info (and label the address as “corrected” or “incomplete”). *Advantages:* Extremely high accuracy and tolerance for imperfect input (Google’s geocoder is known to handle typos and incomplete addresses well [4]). Also returns latitude/longitude if needed (nice-to-have). *Disadvantages:* Beyond the free tier, it is a paid service and comes with usage terms (e.g. results shouldn’t be stored long-term without displaying on a Google Map) [5]. For our expected volume (300–500 queries), Google’s free usage allowance (~40k lookups per month under the $200 monthly credit) is more than sufficient [6].

- **Google Maps *Address Validation API*** – This is a newer offering designed specifically for validating and formatting addresses (available for US and some countries). Unlike the basic Geocoding API which just converts an address to coordinates, the Address Validation API **parses and checks each component for correctness** [7]. It will attempt to **correct, complete, and standardize** the address, and it provides metadata on validation. For example, the API can fill in a missing ZIP code or correct a misspelled street name, and then return a fully standardized address (e.g. with proper USPS abbreviations) [8]. It also returns a verdict for each component (confirmed or not) and overall assessment. If the address is not fully confident, it may indicate which part is problematic or missing. Google's address validation can even integrate USPS's Coding Accuracy Support System (**CASS**) for U.S. addresses [9], meaning it uses official postal data for high accuracy. The response includes the cleaned address and can include geocoding (lat/long and an indication of precision) [10]. *Advantages:* Provides a clear indication of address validity and what changes were made. It's ideal for labeling addresses as **valid** (all components verified), **corrected** (was able to fix or complete the address), or **unverifiable** (not found or too ambiguous). *Disadvantages:* It requires Google Maps Platform usage (paid after the free credit) and is somewhat limited to certain countries (the US is fully supported, which meets our needs). For 30+ countries it works out of the box [11] [12], and the $200 monthly credit applies here as well. In practice, this is one of the most **reliable** solutions due to Google's extensive data and USPS integration.

Using Google's APIs in our Fastify route would involve calling the Google endpoint (either Geocoding or Address Validation) with the user-provided address. We'd then parse the JSON response to extract the structured address fields. For example, with the Geocoding API we'd look for components of type `"street_number"`, `"route"`, `"locality"`, `"administrative_area_level_1"` (state), and `"postal_code"` in the first result's `address_components`. With the Address Validation API, the response already breaks down the address and provides a standardized version. After parsing, we compare the returned normalized address to the input: if differences exist (e.g. different spelling, added ZIP code), we flag the result as **corrected**. If no result is returned or confidence is low (Google returns no candidates or only a very generic match), we flag it **unverifiable**. In borderline cases (multiple results), we can use Google's `partial_match` indicator or the fact that multiple results were returned to decide next steps (more on disambiguation below).

Alternative Address Validation Services (Pros and Cons)

While Google is a strong option, it's important to consider other services. Here are several alternatives, including both paid enterprise-grade solutions and developer-friendly services with free tiers suitable for a proof-of-concept:

- **USPS Address API (USPS Web Tools)** – The United States Postal Service provides a free address verification API that returns USPS-formatted addresses and validates deliverability against their official database. This API will correct and standardize input to match USPS records (including adding ZIP+4 codes) if an exact match is found. *Advantages:* It's **official and very accurate** for checking deliverable mailing addresses in the US. It returns components like street, city, state, ZIP in a standardized format. It's free to use with registration. *Disadvantages:* USPS's API is not very tolerant of typos or partial input – you usually need a reasonably correct address (it may fail or not find close matches if, say, the street name is significantly misspelled or the city is wrong). It also has recently introduced strict rate limits: as of Jan 2026, free usage is capped to **60 requests per hour** [13], which can be a bottleneck even for moderate usage. In our case (300–500 total queries), this limit means we'd have to throttle requests (e.g. at most 1 per minute on average) or request USPS to increase the quota. Another downside is that USPS's API does **not provide latitude/longitude** coordinates – it's purely for address standardization.

- **SmartyStreets (now just "Smarty")** – Smarty provides a popular **US Address Validation API** that uses USPS data under the hood but with a more developer-friendly interface and looser throughput limits. It will verify addresses, correct them to USPS standard (including proper casing, abbreviations, ZIP+4), and can inform you if an address is residential or commercial and whether it's deliverable. *Advantages:* High accuracy (Smarty is USPS CASS-certified for address verification [14]), and the API is easy to use (JSON responses). It handles minor typos by using logical character substitutions and also can return candidates if an input is ambiguous. Smarty's service can also provide geocoding coordinates if needed, but note that precise rooftop geocoding is a separate service. The **free tier** allows **250 lookups per month** at no cost [15], which might cover half of our 500-address POC; higher volumes require a paid plan. *Disadvantages:* The free allotment is limited (250/month), so beyond that you'd need to pay or split calls across months. Also, Smarty's basic verification will only return addresses at ZIP+4 level accuracy (not exact building coordinates) in the free tier. Its terms of service may restrict permanently storing the validated data if using certain premium geocoding features [16] [15] (though keeping the standardized address text is generally fine, they mainly restrict proprietary coordinate data). Overall, Smarty is an excellent choice if you need USPS-level validation with a modern API.

- **Geocodio** – Geocodio is a cost-effective geocoding and address data service focusing on US and Canada. It performs forward geocoding (address to coordinates) and can also do **address component parsing and normalization**. One of its features is an optional **USPS delivery point validation** add-on: you can request that Geocodio return USPS mail deliverability info (like ZIP+4, carrier route, etc.) along with the geocoded address. *Advantages:* Generous free tier (2,500 lookups **per day** free during trial, which is effectively up to ~75k in a month if fully utilized) [17] [18], easily covering a 500-address test. Geocodio's pricing beyond that is very affordable ($0.50 per 1k lookups) [18]. It returns structured address components and coordinates, and it even provides an **accuracy score** for the geocode. The terms are developer-friendly – no restrictive caching rules (you can store results freely) [19] [20]. *Disadvantages:* Its address validation (USPS matching) is not CASS-certified (so it may not catch every nuance that USPS's own or Smarty's might) [14]. Also, Geocodio might not be as aggressive in correcting typos as Google or Azure's AI-backed engines; it often expects reasonably well-formed input or you need to use their autocomplete API for messy input. Nonetheless, for a reliable **low-cost alternative** that returns all required fields (street, number, city, state, zip) and even lat/long, Geocodio is a strong candidate.

- **Microsoft Bing Maps / Azure Maps** – Microsoft's geocoding services are another solid alternative. Bing Maps' legacy Geocoding API is being succeeded by Azure Maps Search service (which uses the **TomTom** data under the covers). These services take free-form addresses and return structured address details plus coordinates. *Advantages:* Good accuracy for US addresses and built-in tolerance for typos/incomplete queries – for example, Azure's geocoder is *"very tolerant of typos and incomplete addresses"* [4]. You can restrict by country (using `countryRegion=US` in Bing or `countrySet=US` in Azure) to focus on U.S. results. The response includes a confidence score and a classification of the match. Bing/Azure will also return multiple candidates for ambiguous input (with scores), which you can use to pick the best. One big benefit is the **free usage quota**: Bing Maps offers about **10,000 free geocoding requests per month** (125k/year) [21] , and Azure Maps has a free tier as well if you have an Azure account. That easily covers our needs. These services also allow reasonably high throughput (up to ~50 requests/sec in Bing's case) [22] . *Disadvantages:* The quality is very high, though possibly a notch below Google in some edge cases (since Google's address data is extremely comprehensive). Also, like Google, the terms require using the data with their mapping platform for permanent storage (though enforcement is lax for small use). In practice, Bing/Azure is a reliable and cost-effective choice for address standardization, providing all the needed fields (`addressLine`, `locality`, `adminDistrict` for state, `postalCode`, etc. in the JSON response) [23] [24] , along with a confidence indicator.

- **HERE Maps Geocoding API** – HERE is another major provider with a strong geocoder. It returns detailed address components and has good coverage in the US and internationally. *Advantages:* Very generous free tier (**250k transactions/month** on the Freemium plan) [25] . The API can handle forward geocoding with fuzzy matching and will return a confidence or match quality metric. HERE's data is known to be accurate (they have a history of Navteq data for North America). The output includes latitude/longitude and address elements similar to Google's. *Disadvantages:* Like other pure geocoders, it doesn't explicitly tell you "this address is deliverable" – it simply gives the best match. Also, storing results long-term has restrictions unless you get an enterprise license (standard plan expects you to just use the results on the fly). For a backend service that just validates on request (no massive caching), this is not a big issue. HERE could be a good alternative if you want a high free allowance and solid data without using Google.

- **Mapbox Geocoding / OpenStreetMap-based APIs** – Mapbox's API and open-source options like **OpenCage**, **LocationIQ**, or **Nominatim** use OpenStreetMap (OSM) data. They can also convert a free-form address to structured data. *Advantages:* These often have **free tiers suitable for prototypes** – e.g. Mapbox offers up to 100k requests/month for free [26] , OpenCage allows 2,500/day for free [27] , etc. They return structured components (OSM has a well-defined address format). They are also quite flexible about storing results (especially OpenCage or direct OSM Nominatim, which have very open licenses) [28] . *Disadvantages:* The accuracy of OSM data for addresses can vary. In well-mapped urban areas, it's good; but in rural areas or smaller towns, the address might not be present or up to date in OSM. These services may struggle with **postal address validation** in the sense of knowing deliverability – they simply reflect what's in the map data. They also might not normalize to USPS mailing standards (for example, they might return "Road" vs "Rd" or miss a ZIP+4). Nonetheless, for a free proof-of-concept that needs basic geocoding, they are viable. For production where **reliability** and correctness are paramount, one of the above APIs backed by official or commercial datasets would be preferable.

- **Enterprise Address Verification Services** – There are specialized tools like *Melissa Data*, *Loqate (GBG)*, *Experian*, *PostGrid*, and others that provide address validation as a service. These often use postal databases (USPS and international) and offer advanced features (e.g. suggest alternatives if an address is not found, batch cleaning, etc.). *Advantages:* Very high accuracy and often coverage of many countries. They can handle tricky cases and often come with support/consulting for data quality. Some have nice extras like identifying dormancy (if an address was recently vacated) or providing metadata (county FIPS code, etc.). *Disadvantages:* These services are usually **paid** products with no (or very limited) free tiers. They might be overkill for a small project. Integration is straightforward via API, but you'll need to manage API keys, costs, and possibly contracts. For completeness, if the project later scales or needs international address validation, they are worth considering. For example, **Loqate** has both on-premise and cloud options and a free trial [29] [30], and **Melissa** offers a certain number of free credits for testing. In our scenario (U.S. only, low volume), these might not be necessary, but they are reliable alternatives to keep in mind.

Handling Edge Cases and Selecting the Best Match

Partial or Incomplete Addresses: If a user provides an address missing some components (e.g. no street number, or no ZIP code), the chosen API should attempt to infer the missing pieces. For instance, Google's Address Validation API will *"provide the complete address as determined by the API, or indicate where information is missing"* [31]. In practice, this means it might return a result saying "123 Main St, Springfield, ??" with a flag that state is needed, or suggest a city if it was missing. Other APIs like Azure Maps will still return a result for a partial query (they might geocode to the center of a street or city if that's all that was given). Our implementation should detect that the result isn't fully specific. For example, if the user only entered a street name and city, the API might return a location for that street (without a house number). In such cases, we would not mark it "valid" because the input is incomplete – we might mark it as **unverifiable or incomplete** and possibly include a note that more detail is required. The key is that the **status** field in our response should reflect this: a partial match from the geocoding service would be indicated as not a full valid address. (For Google Geocoding, `partial_match: true` on the result is an immediate signal that the address was incomplete or only partially matched [3].)

Typos and Corrections: Modern geocoders use fuzzy matching to handle typos. For example, Azure (TomTom) can interpret minor spelling mistakes in street or city names [4], and Google is well-known for this as well. If the API corrects a typo (say, "1600 Ampitheater Pkwy" → "1600 Amphitheatre Pkwy"), the returned address will have the correct spelling. Our service can compare the normalized output to the original input. If they differ (beyond just trivial formatting), we flag the result as **corrected**. We will output the corrected, standardized address to the user, and possibly include a flag like `{"status": "corrected", "notes": "Corrected spelling of street name"}`. This way, the client knows the address we returned is valid but was slightly different from what they entered.

Multiple Candidate Matches: If the API returns multiple possible locations (which can happen if the input is ambiguous, e.g. "100 Washington, Springfield" – there could be many Springfields), we need criteria to automatically select the best option. Here are a few strategies we will employ:

- **Confidence/Score:** Many APIs provide a confidence metric. For example, Bing's response includes a `<Confidence>` field with values like High, Medium, Low [32]. TomTom/Azure provides a similar score, and Google's Address Validation might provide a component-level confirmation status. We will prefer the result with the highest confidence score. If one result is "High" confidence and the others are "Medium" or lower, choose the high confidence match [32]. Similarly, if using Google Geocoding, we generally trust the first result if the API sorted it highest, unless `partial_match` is true.

- **Match Quality / Granularity:** We should check the level of the returned result. For instance, Google and others tag results with types: if one result is a precise **street address** and another is just a **zipcode or city**, we pick the precise address. In Google's results, an exact address match will have types like `"street_address"` or `"premise"`, whereas a broader result might be `"locality"` (city) or `"route"` (street without number). We will prefer results that include a street number and full address details over those that don't. If the top result from Google has `partial_match: true` and only matched the street or city, but a second result perhaps had a full match, we'd examine that. However, Google usually ranks exact matches first. In the rare case it doesn't, we could implement logic to scan the results for one that has all the components (street, number, city, state, zip) present.

- **Geographic Filter:** Since we are focusing on the US, we will have the country restricted which helps a lot. If ambiguity remains (e.g., the same street address exists in multiple states), we might use additional context if available (none is mentioned, but in an interactive setting one could use the user's IP or a preferred state). In a pure API context, if multiple states/cities come up with the same address, it's safest to mark it unverifiable or return a list for the client to choose. For our design, we could decide to return the first result but include `status: "ambiguous"` and maybe an array of alternative suggestions. However, since the prompt suggests selecting the "best suitable option" automatically, we will implement a reasonable heuristic (highest confidence + most complete match as described above) and return that single address.

- **No Good Match:** If none of the returned candidates have high confidence or everything is a very generic match (e.g., the input was too vague), our service will gracefully handle this by returning an `unverifiable` status. For example, if someone enters "123, USA" – there's no way to validate that into a proper address. The API might return a best guess (maybe "USA" as a country center). We'd detect that important components are missing and respond with a failure status, indicating we could not verify the address.

In summary, the logic is: use the external service's best match if it is sufficiently confident and complete; otherwise, either ask for clarification or mark the address as not verified. Since this is a backend service without user interaction, our approach is to be conservative in marking an address "valid." We only do so if we get a single clear match. If we get a corrected address (like fixed spelling or added ZIP) from the service, we mark it "corrected but valid." If we get multiple possibilities with no clear winner, we mark as "unverifiable/ambiguous." This information will be reflected in the API's JSON output. For instance, a successful response might look like:

```
{
  "street": "1600 Amphitheatre Pkwy",
  "number": "1600",
  "city": "Mountain View",
  "state": "CA",
  "zip": "94043",
  "status": "valid",
  "latitude": 37.422408,
  "longitude": -122.085609
}
```

If the input had a minor typo that was fixed, `status` might be "corrected" and we could include an extra field like `"originalInput": "1600 Ampitheater Parkway, Mountain View, CA"` for reference. And if no valid address is found, `status` would be "unverifiable" with perhaps a `"message": "Address could not be validated"`.

Conclusion and Recommendations

For a highly **reliable** US address validation service, using Google's Address Validation API would be a top choice given its strong accuracy and ability to explicitly handle corrections/missing info. It directly fulfills the requirement to indicate validity or necessary corrections in the address. The Google Geocoding API is a close alternative if we primarily want lat/long and components without the extra validation metadata (it's simpler and a bit more mature in terms of usage examples, but we'd have to infer validity from the results). If avoiding Google or seeking other options, a combination of a USPS-based service for validation and another for geocoding could work: for example, use SmartyStreets to validate and standardize the address (ensuring USPS-approved format) and then optionally use a geocoder to get coordinates if needed. This covers deliverability and structure nicely.

For a **proof-of-concept (POC)** or limited budget scenario, services like **Geocodio** or **Microsoft Azure Maps** offer generous free usage and would allow 300–500 lookups easily within their free tiers [21] [18]. These will return the necessary components (though you should note if they return all fields: e.g., Geocodio will return ZIP code and often ZIP+4 if USPS data is appended, whereas Azure will return just the 5-digit postal code by default). Using one of these free-tier options can save cost during development. Just keep in mind the trade-offs: e.g., Geocodio and Azure won't outright tell you "this address is invalid" – they'll just fail to find it or return a low confidence result, so your code has to interpret that.

In conclusion, the implementation will involve choosing one of these services, integrating its API call in the Fastify route, and then applying the above logic to handle edge cases. Google Maps Platform (Geocoding or Address Validation) is a robust default with high success in parsing messy input into standardized addresses [7]. USPS or Smarty can guarantee postal validity (useful if the use-case is mailing). The final decision may weigh **cost vs. accuracy**: Google's free allowance covers our usage and provides top-notch accuracy, so it stands out as a strong candidate. Regardless of the service, we will ensure the API's output structure is consistent (street, number, city, state, zip, plus maybe lat/long) and that we clearly indicate the address's validity status (valid, corrected, or unverifiable). This approach will fulfill the requirements and handle user input robustly, even when addresses aren't perfectly provided.

Sources:

- Google Maps Platform – Address Validation API overview [7] [31] (address parsing, correction, and component validation)
- Google Maps Geocoding documentation [1] [3] (structured components and partial match indicator)
- Azure Maps Address Search documentation [4] (tolerance for typos and incomplete addresses)
- Bing Maps Locations API example [23] [32] (sample output with formatted address, components, and confidence)
- Mapscaping Guide to Geocoding APIs [6] [21] [18] (free tier and pricing info for Google, Bing, Geocodio, etc.)
- Geocodio vs SmartyStreets comparison [15] [14] (free usage limits and USPS verification capability)
- SmartyStreets documentation and blog [13] (USPS API rate limit and reasons to use alternatives like Smarty)

[1] [2] [3] Geocoding Service | Maps JavaScript API | Google for Developers
https://developers.google.com/maps/documentation/javascript/geocoding

[4] Search - Get Search Address - REST API (Azure Maps) | Microsoft Learn
https://learn.microsoft.com/en-us/rest/api/maps/search/get-search-address?view=rest-maps-1.0

[5] [6] [18] [21] [22] [25] [26] [27] [28] Guide To Geocoding API Pricing - December 8, 2025
https://mapscaping.com/guide-to-geocoding-api-pricing/

[7] [8] [9] [10] [31] Address Validation API overview | Google for Developers
https://developers.google.com/maps/documentation/address-validation/overview

[11] [12] [29] [30] Choosing the Best Address Verification Tool - Byteplant Data Quality Blog
https://www.byteplant.com/blog/best-address-verification-tool/

[13] USPS API rate limit capped at 60 requests per hour: How to prepare and why many are choosing Smarty
https://www.smarty.com/blog/usps-api-rate-limit

[14] [15] [16] [17] [19] [20] Geocodio vs Smarty (SmartyStreets) Comparison - Geocodio
https://www.geocod.io/geocodio-vs-smartystreets-comparison/

[23] [24] [32] Find a Location by Address - Bing Maps | Microsoft Learn
https://learn.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address