# Update endpoint requests from Chris 11/5:

For more detailed examples, see the demo endpoints at: <https://github.com/Viva-Translate/viva-chrome-extension/blob/poc-phase1.2/background.js>

Working API:

## 1. Translate endpoint

Currently it takes in: { text, source\_language, target\_langauge }

**WHAT**: I would also like to provide an optional **glossary\_terms** array where I specify a set of glossary terms (in spanish, all lowercase, the exact terms that were returned from the glossary endpoint), which the api should substitute to their English glossary dictionary match before sending to deepL.

E.g. INPUT:

{

“text”: “Me quedo mirando la pantalla todo el día, pero no me importa la computadora.”,

“source\_language”: “es”,

“target\_language”: “en”,

“glossary\_terms”: [“pantalla”]

}

OUTPUT:

“Me quedo mirando la screen todo el día, pero no me importa la computadora.”

**WHY**: currently the glossary term endpoint already substitutes the english terms into the spanish source text, it does all matching terms at the same time. This does not allow the user to acknowledge certain industry terms while overriding others as the UI suggests.

## 2. Glossary (Remote Work) Endpoint (API\_URL + /glossary/remote-work)

Currently this endpoint takes in a spanish text string, substitutes glossary terms (putting the english terms matches in the spanish string), then returns the spanish string with the english substitutions. Notably, the endpoint makes ALL substitutions at once.

E.g. INPUT “Miro mi pantalla todo el día” ⇒ OUTPUT: “ Miro mi **screen** todo el día”

**WHAT**: The glossary endpoint should return the exact matches it found, and for each match return the english and spanish words. It should return the word in the exact same case as the original text input (i.e. if the word is capitalized in the original input, return it as capitalized in the output). Each object in the array should be in the form { original: string, translated: string}

E.g.

INPUT: “Me quedo mirando la pantalla todo el día, pero no me importa la computadora.”  
OUTPUT: [{ original: “pantalla”, translated: “computadora”} , {original: “computadora” , translated: “computer”}]

**WHY**: current endpoint does not return the exact spanish term which the user might wish to substitute, yet we want to display this on the front-end, so it makes sense to return it from the endpoint directly (since it should exist as an intermediate in the python code). Also the current’s text return makes all substitutions and does not allow the user to decide on each substitution.

## 3. Paraphrase Endpoint

Currently the paraphrase endpoint returns an array of strings representing possible paraphrasings of the input. On their own however, these strings are not too useful because we don’t know which of them would result in a higher quality estimation value than the original string. Instead of sending the un-vetted choices straight to the client, then expecting the client to make a QE request before continuing, the paraphrase endpoint should chain directly into the QE endpoint on the backend and only return a subset of choices (max 3) which have higher QE values than the original sentence.

**WHAT**: input/output will look very similar to existing paraphrase endpoint, except all output sentences will have been vetted on the backend before being returned. Additionally the api should truncate to the top 3 choices (since any more than that aren’t useful to the user).

## 4. Detect Language Endpoint

**WHAT**: take in email text, return a language code e.g. “en” or “es”. Note, the backend should probably truncate the input to the first 100ish characters to avoid using more deepL tokens than necessary.

**WHY:** this endpoint will allows us to detect the language of incoming emails, so that we don’t render the viva box for spanish incoming emails.

## 5. Changes to the QE endpoint (optional)

**WHAT:** not super urgent, but I would prefer we wait to return the value of the resolved quality estimation, instead of returning a task and having the client repeatedly query.

**WHY:** Less overhead on the server from processing the result endpoint repeatedly. There are also issues on the client side with sending lots of repeated requests, as occasionally the requests will fail, and the data on the server only seems to persist a limited time after being returned. Should be ok to send a request that takes a long time to resolve (15s) as long as the client is aware of it and does not time out.