

Section Handout #8

Problem Un: Deuxlingo

You're to leverage a single-endpoint API—one we've written for you—that knows how to translate from English to either Spanish, French, German, or any other language. The endpoint can be invoked using the following URL using **GET**:

```
https://web.stanford.edu/class/cs106ax/cgi-bin/translate.py
```

When invoking the `translate.py` endpoint, we need to be clear what English phrase should be translated **and** what the target language should be. These two items should be part of the query string as parameters named `source` and `to`, respectively. `source` can be any free English text whatsoever, and `to` can be any of the language codes documented at <https://tinyurl.com/deuxlingo>. This link, along with a starter version of the code needed to fully implement a solution to this problem are posted to the Sections page.

Here's the HTML file needed for the application. You can assume all CSS rules just do the right thing, and that all you need to manage is to add and remove the `invisible` class as you toggle between edit and translate mode.

```
<!DOCTYPE html>
<html>
  // head section omitted for brevity
  <body>
    <div class="main-content">
      <textarea id="textarea" class="text-region text-editable">/textarea>
      <div id="source-div" class="text-region text-frozen invisible">/div>
      <div id="target-div" class="text-region text-frozen invisible">/div>
      <div>
        <button id="translate-button" type="button">Translate</button>
        <button id="edit-button" type="button" class="invisible">Edit</button>
      </div>
    </div>
  </body>
</html>
```

Otherwise, you're to leverage the above API call to translate English phrases you enter into the textarea so the untranslated and translated strings show up in the neighboring `divs` with ids of `source-div` and `target-div`. You can hardcode in any target language you'd like that's supported, or you can extend the HTML to include an `<select>` tag with child `<option>` tags for each of the languages you'd like to support. If the `to` parameter is set to `"fr"`, then English is translated to French. Prefer Spanish, German, Japanese, or Bulgarian? Then go with `"es"`, `"de"`, `"ja"`, or `"bg"` instead! Try right-to-left languages (e.g. `"ar"` for Arabic, `"he"` for Hebrew) and change the CSS so that the translation `div` is right justified instead of left.

Problem Deux: Client-Side JavaScript

You're planning on launching a commercial version of a social network you've built, and you'd like to extend the current implementation to support video upload. Doing so requires you add two new API endpoints to the server.

- **POST** `api/upload`, which accepts as payload the contents of a video file and, on success, responds with a small JSON object containing nothing more than an `id` number forever associated with the video, and with:

```
{ "id": 26172831 }
```

The server responds as quickly as possible with the response, even though it may take several minutes to process the video and efficiently store it.

- **GET** `api/upload/<id>/status` assumes the embedded `id` number is a valid video upload id and responds with its own JSON object containing the same id and a `percent` field, as with:

```
{ "id": 26172831, "percent": 14 }
```

The above response suggests that the video with the stated id number is 14% processed, and that you should issue the same query again to get an updated progress percentage. You can assume that the percent value is always some integer between 0 and 100 inclusive.

Note that neither of the two endpoints require any query parameters (i.e., you never need to call `setParam` or `setParams`).

For this problem, you'd simply like to test both endpoints to see that they work properly. To do so, you're to implement the `testVideoUpload` function, which accepts the video data as a string and issues the **POST** request with the supplied video string as payload. Your success handler should print `"Video (id: 26172831) upload initiated."` to the console before calling `setTimeout` to prompt a **GET** request (to the second of the two new endpoints using the id number it just received) be sent five seconds later. The success handler for the **GET** request should `console.log` the video id and the percentage on a single line—structure the line as `"26172831: 14% processed."`—and schedule another **GET** request for the same id be sent five seconds later if the percentage is anything less than 100%. Each success handler will schedule the same **GET** request to be sent five seconds into the future until the server responds saying the video has been fully processed at 100%.

Assuming `video` is a variable bound to the video data—yes, as one very long string of data, a call to `testVideoUpload(video)`, via the execution of many success handlers five seconds apart, might print this over the course of approximately 30 seconds:

```
Video (id: 26172831) upload initiated.  
26172831: 14% processed.  
26172831: 28% processed.  
26172831: 51% processed.
```

```
26172831: 77% processed.  
26172831: 98% processed.  
26172831: 100% processed.
```

Of course, the id number will vary and will be dictated by the POST response. And the percentages can vary as well.

Present an implementation of your `testVideoUpload` function. Note that there's no client-side DOM manipulation in this problem. We're concerned primarily with your ability to use the `AsyncRequest` and `AsyncResponse` JavaScript classes discussed in lecture. Note that you'll need to implement several small functions to serve as success handlers and timer functions.

```
function testVideoUpload(video) {
```