DataModel, Technical Scope, Use Cases and Future Work

Unbabel Challenge

This webApp delivers the possibility of translate anything that the user writes in the input field. The page is composed by an Input field, a picklist , a submit button and a table with five columns. The user can use the app accessing to *localhost:8080/home*.

A request to the URI *https://api.unbabel.com/tapi/v2/language_pair/* will be made. After the API respondes with the representation of the resource, this will be mapped to a Dto (*Data transfer Object*). This Dto fields are putted in the context and sent to the engine to render the handlebars view. This fills the picklist with the languages supported by the Unbabel API and this page will be rendered:

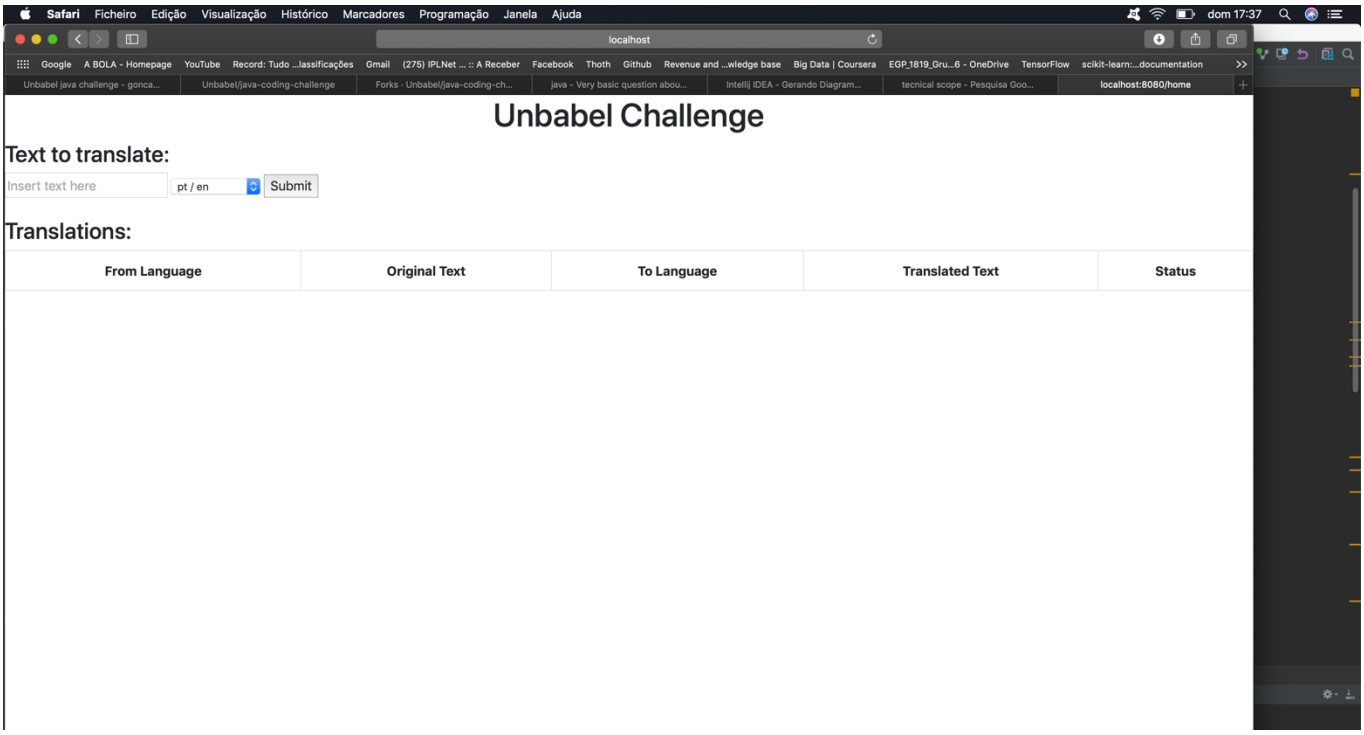


Figure 1: Home page

Then the user inserts the text and clicks in the submit button.

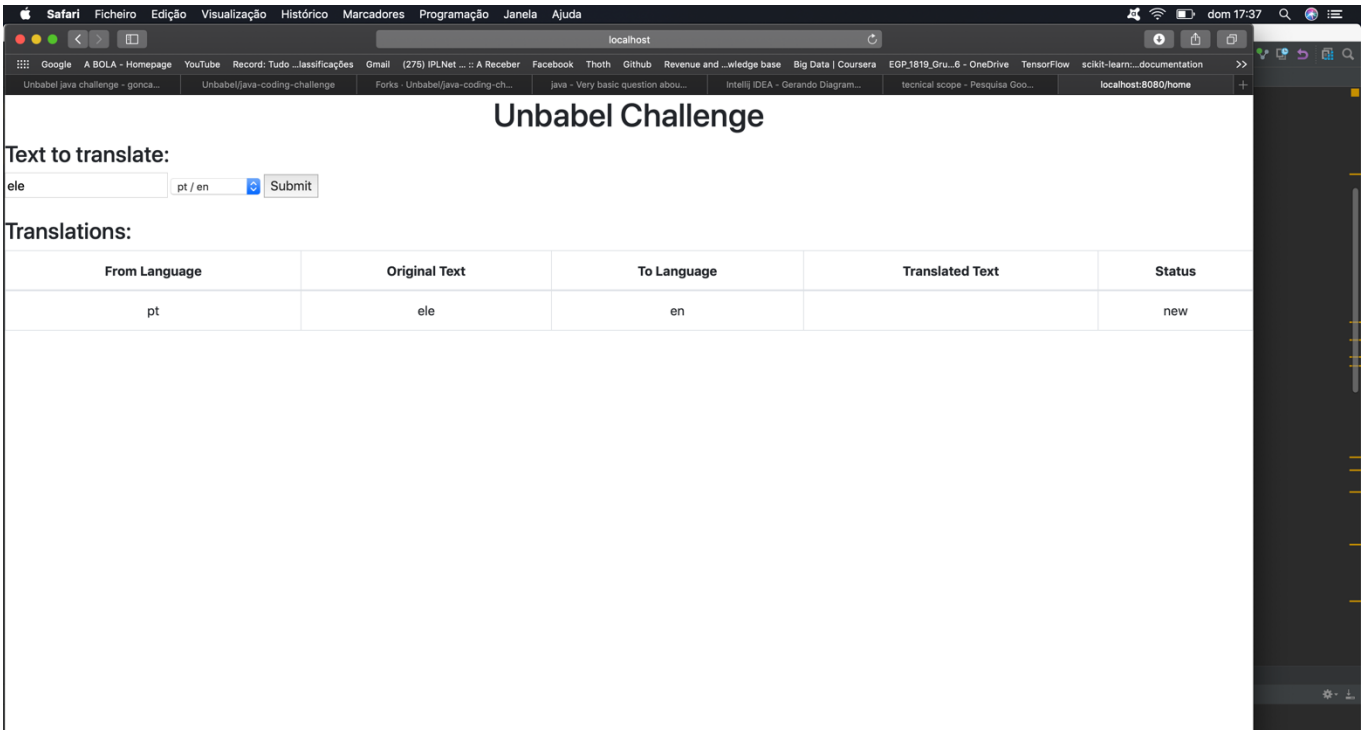


Figure 2: Translation request

This will make a request to the URL *https://api.unbabel.com/tapi/v2/mt_translation/* with the parameters: Source language: “pt”; Target language: “en”; Text to Translate: “ele” as a JSON object. After receiving the response, the fields of the dto will be putted in the context ans sent to the engine to render. Tis time the engine will not render the entire html page but will fild the table with a specific Id and append the data. After this row is appended a polling is triggered to check the translation status.

This polling send a request to: *https://api.unbabel.com/tapi/v2/translation/:u* . The “:u” is the id received from the first translation request.

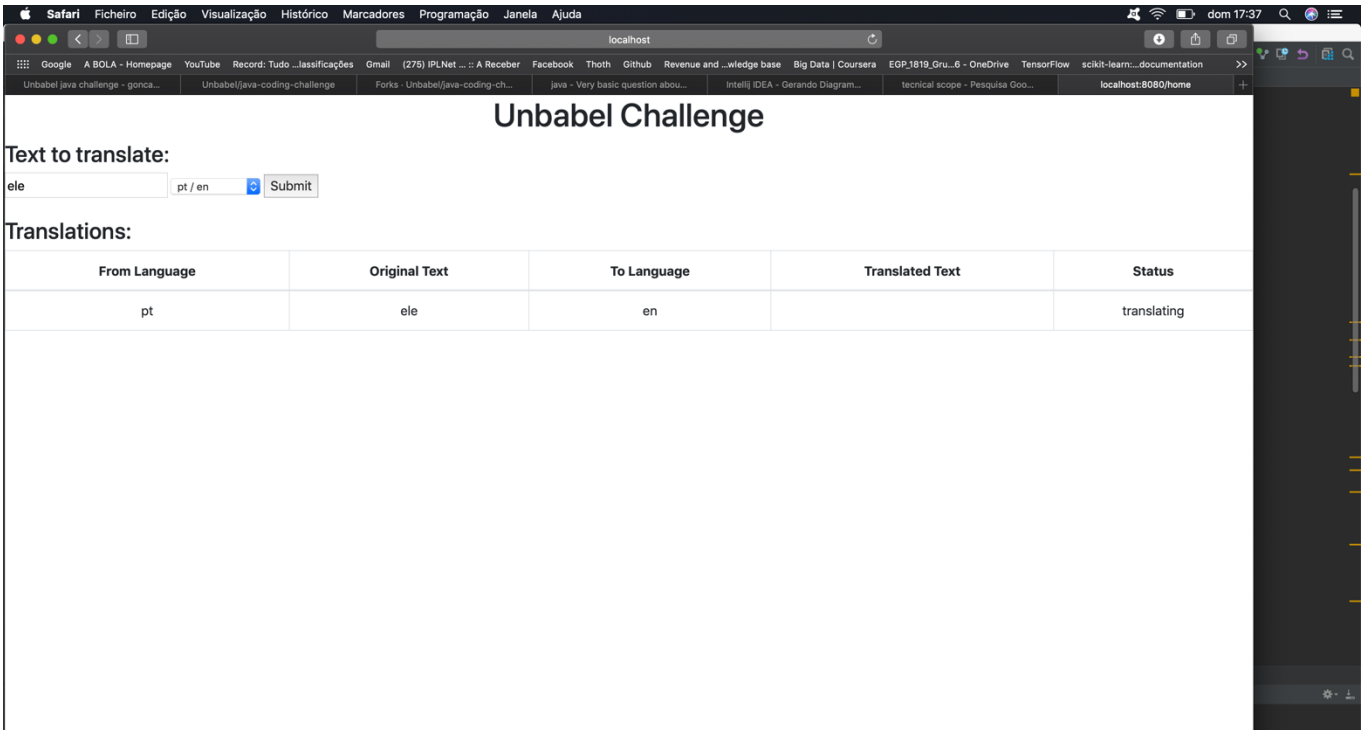


Figure 3: Translation status changed

If the translation status changes, only the “td” element of the table is changed instead of loading the entire page.

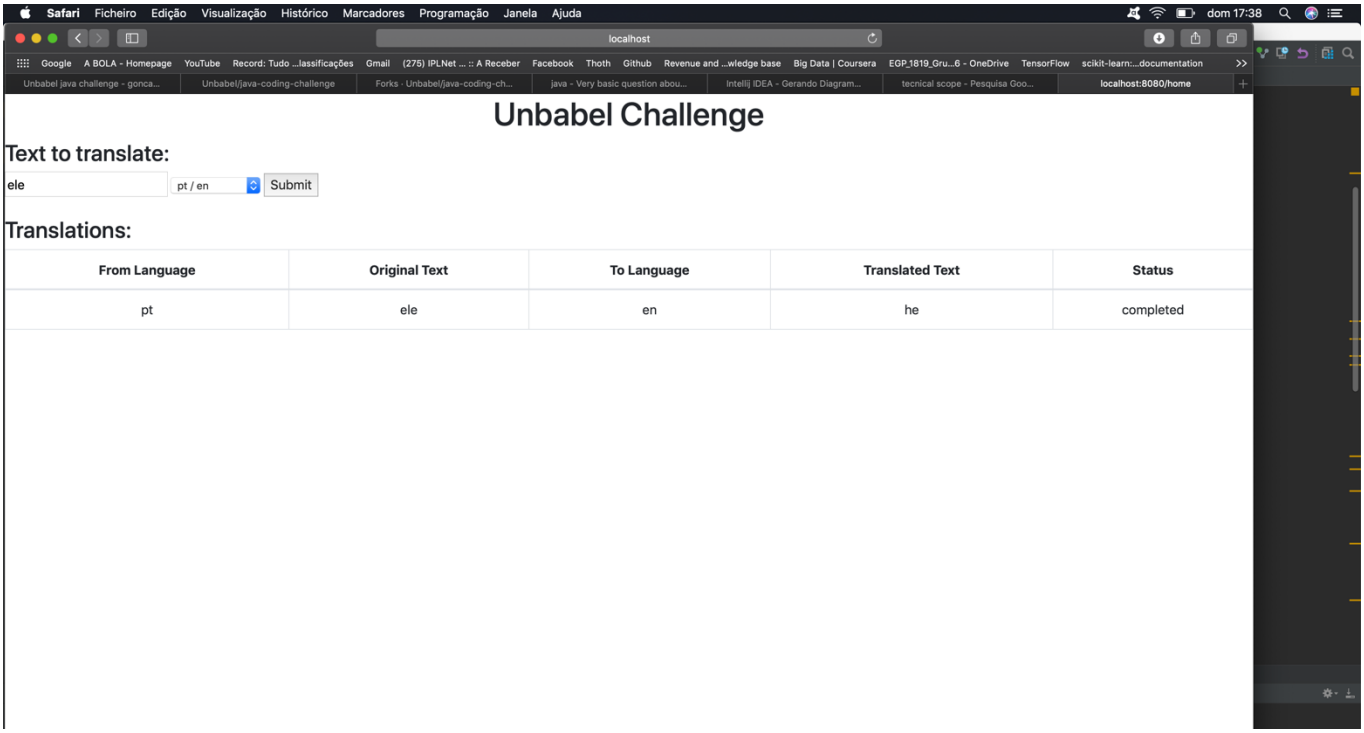


Figure 4: Translation completed

If the status received from Unbabel API is equal to “*completed*”. The two td elements are updated and the polling is stopped.

The user can even insert various translations at the same time and don’t have to wait for a translation to be completed to get other.

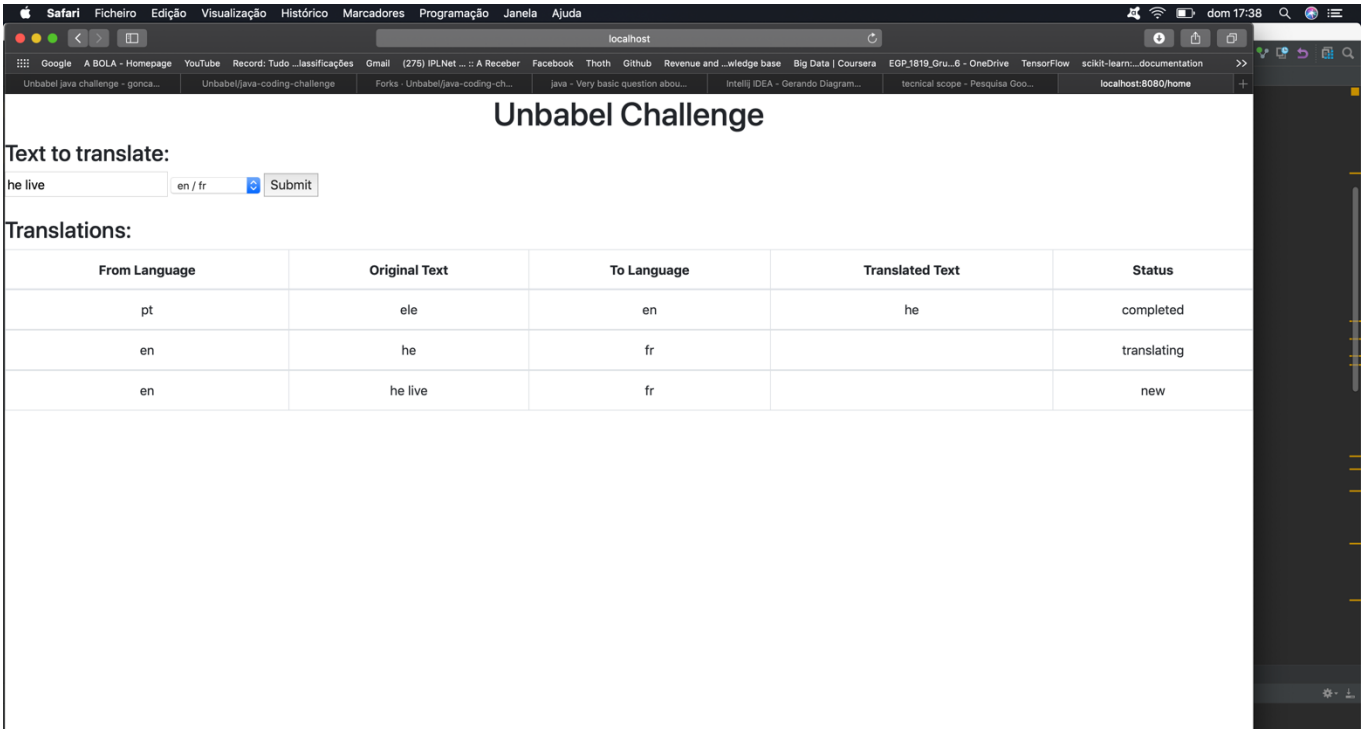


Figure 5: Various translations at the same time.

In the image above (5) we can see that the user inserted 3 translations. One is completed, the second one is translating and the third is new.

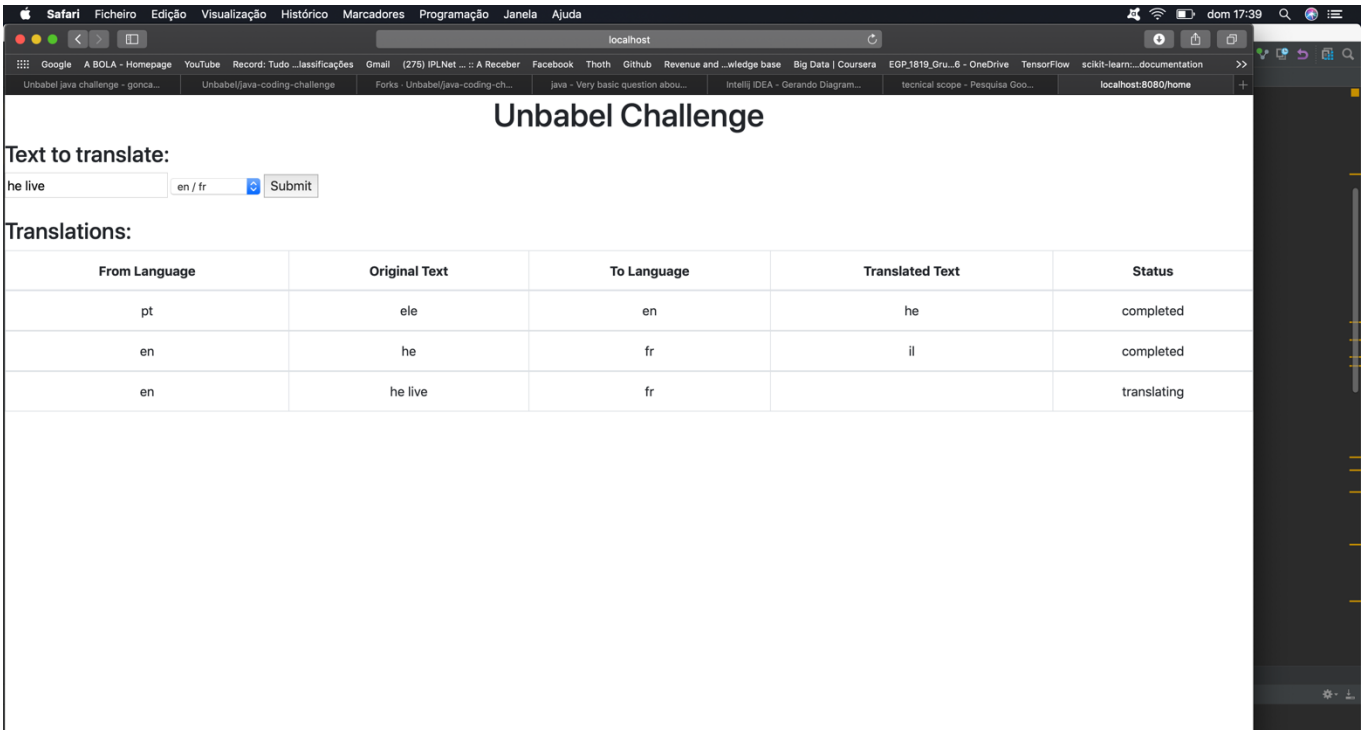


Figure 6: Second translation completed

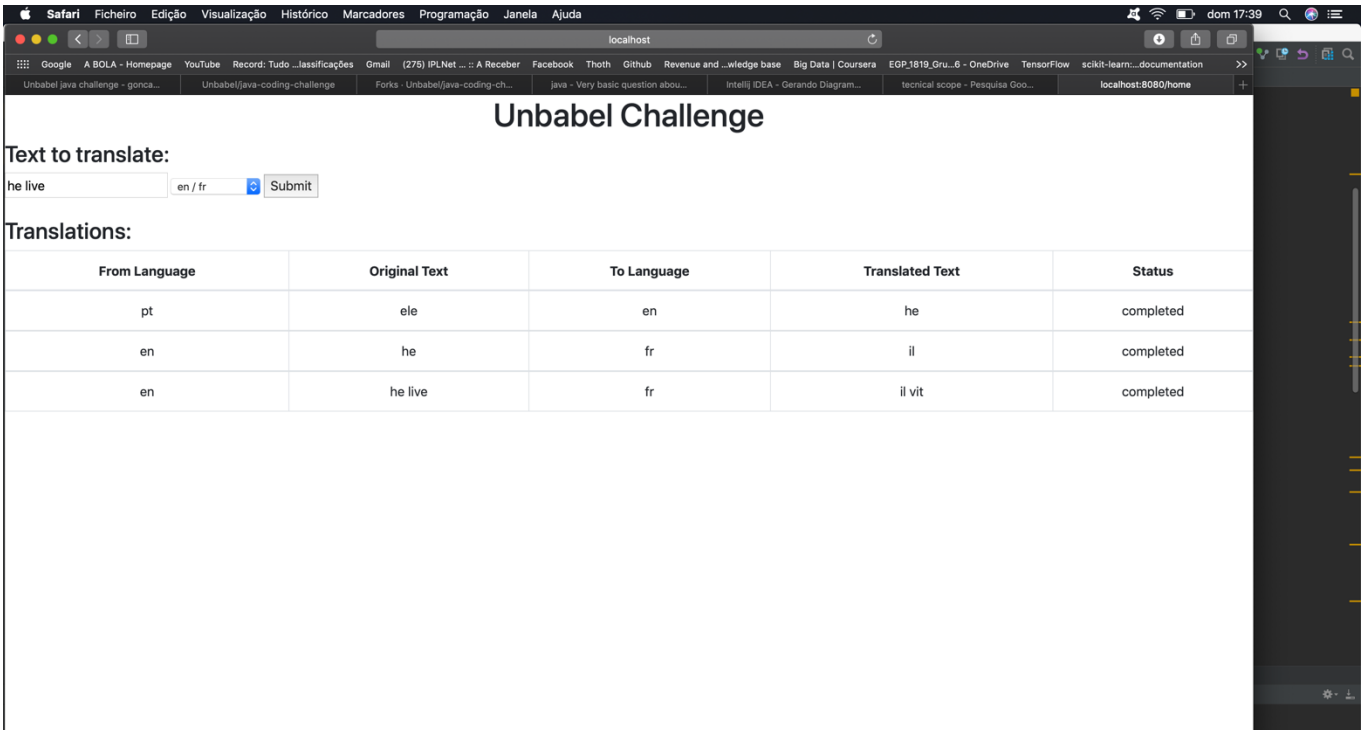
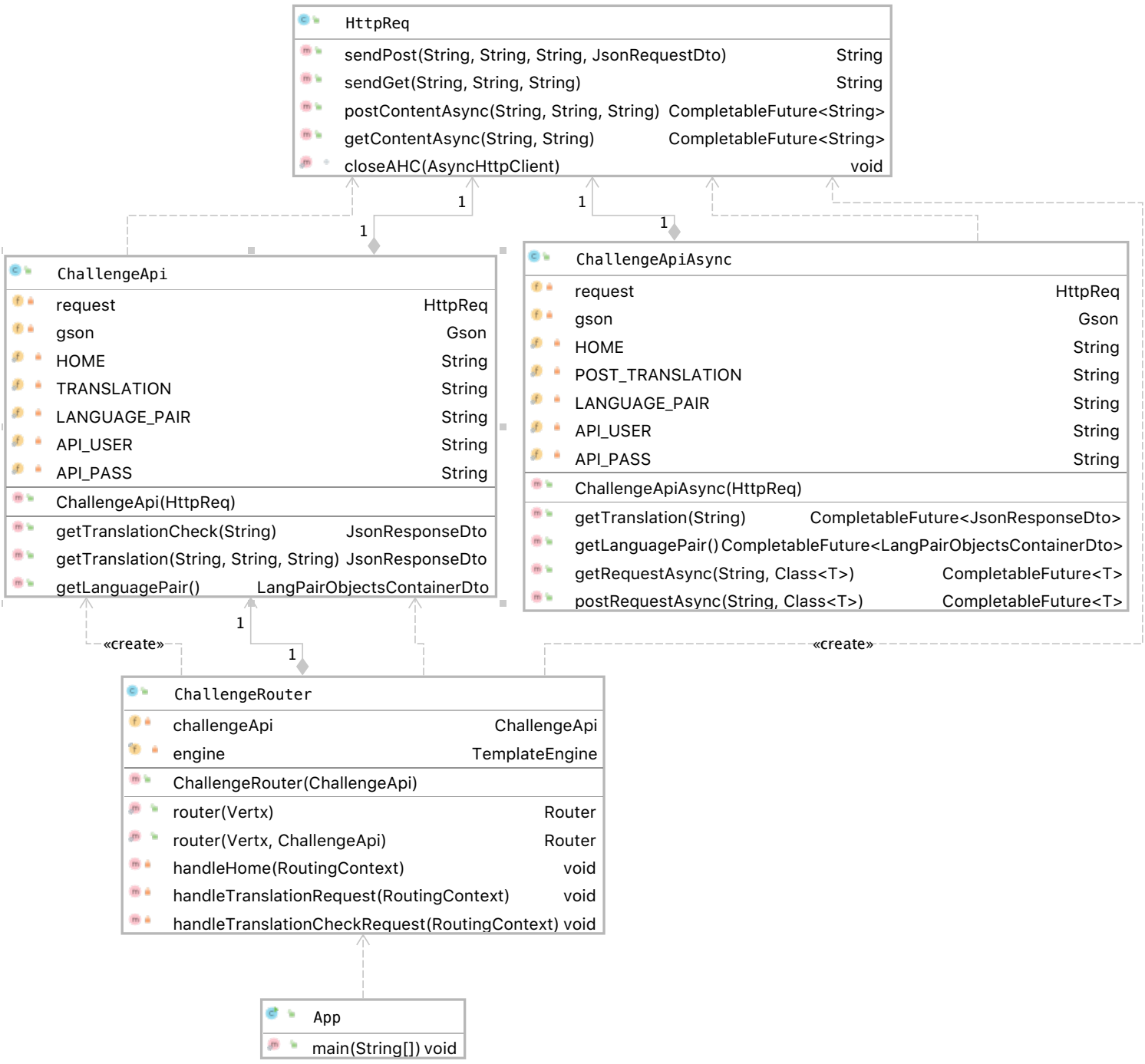


Figure 7: Translations at the same time completed

Future work

For future work, the server should be done with an asynchronous processing. It is also a believe that an Exception class that extends from Exception should be implemented to deal with possible errors coming from the server


HttpReq
sendPost(String, String, String, JsonRequestDto) String
sendGet(String, String, String) String
postContentAsync(String, String, String) CompletableFuture<String>
getContentAsync(String, String) CompletableFuture<String>
closeAHC(AsyncHttpClient) void
1
1
1
1
ChallengeApi
request HttpReq
gson Gson
HOME String
TRANSLATION String
LANGUAGE_PAIR String
API_USER String
API_PASS String
ChallengeApi(HttpReq)
getTranslationCheck(String) JsonResponseDto
getTranslation(String, String, String) JsonResponseDto
getLanguagePair() LangPairObjectsContainerDto
ChallengeApiAsync
request HttpReq
gson Gson
HOME String
POST_TRANSLATION String
LANGUAGE_PAIR String
API_USER String
API_PASS String
ChallengeApiAsync(HttpReq)
getTranslation(String) CompletableFuture<JsonResponseDto>
getLanguagePair() CompletableFuture<LangPairObjectsContainerDto>
getRequestAsync(String, Class<T>) CompletableFuture<T>
postRequestAsync(String, Class<T>) CompletableFuture<T>
«create»
1
1
«create»
ChallengeRouter
challengeApi ChallengeApi
engine TemplateEngine
ChallengeRouter(ChallengeApi)
router(Vertx) Router
router(Vertx, ChallengeApi) Router
handleHome(RoutingContext) void
handleTranslationRequest(RoutingContext) void
handleTranslationCheckRequest(RoutingContext) void
App
main(String[]) void