Senior web developer coding challenge (v1.0, 2020-05-16)

Introduction

Thank you for your application. We would like to use a challenge to see how you think and solve problems. If you have any big issues with doing a challenge like this, please suggest a better way to understand your way of thinking.

Background

Our mobile app is available in 8 countries and languages (but because it's strongly dependent on local medical sources - not in English). An important part of the app are **tools**. Functionality of a tool can be to calculate risk of a disease, to calculate an intermediate score needed for drug dosing, to monitor disease progression etc. For an example, check a tool in the Italian web app: https://mediately.co/it/tools/SPESI (this one is in English, but most are fully translated).

Tools in mobile apps are not implemented natively, they are described using a declarative language - every tool has a specification in JSON (called **tool JSON spec**). The spec describes:

- UI elements (e.g. "yes/no picker" or "numeric input field between 30 - 300"),
- their behaviour (e.g. "If answer to question 3 is NO, hide questions 4 and 5"),
- calculation of result (e.g. "1 point for each YES answer, result is HIGH RISK if total is over 4 points").

This tool spec is then served to mobile apps through the API.

These tool specs need to be translated into all 8 languages. Automatic translation already exists (merging translations into tool spec template) but has proven to be error-prone. Since this is medical content, it's very important to have detailed control over every change on tool spec before it starts being served through the API. For this, we decided to use the familiar GitHub feature **Pull Requests**.

The following convention is used for tool spec filenames:

`<tool_name>.<language>.master.json`	"Master" spec (in a certain language) on which translations are based

`<tool_name>.<language>.json`	Translations of master into other languages

Task

You will create a simple Rails web app with one model and controller. No effort is required on frontend, just the most basic CRUD is enough.

You will be using interacting with two different services through their API or command-line tools:

- Lokalise, a web service for storing translations
- GitHub, a web service for versioning code and other text-based content

The only model in the app is **Tool** with fields:

- `name` [string]
- `language` [string] (2-letter language code)
- `json_spec` [json]

Controller for this model should implement two actions.

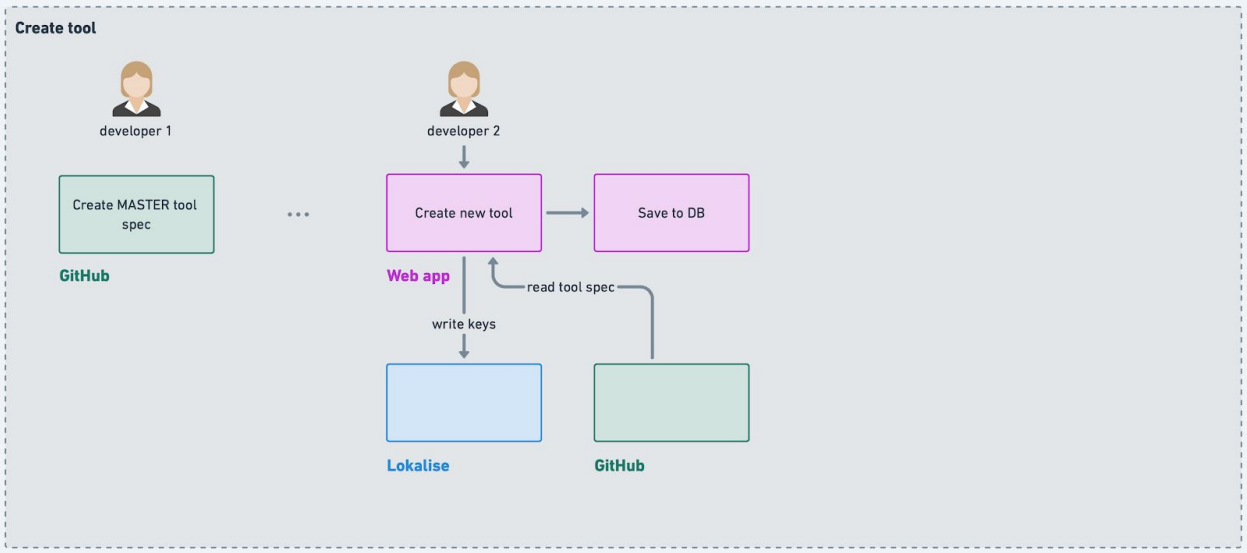
Environment

Upload this JSON tool spec to a repo on GitHub.

Use these credentials for Lokalise

- Project ID (`webdev_challenge_4`): `296344285ec22d64085400.99297850`
- API key: `79f58e3b769ce941f75bab97d7fb17f118cf5524`

Action 1: Create tool



1. When a tool is created, fetch the JSON tool spec from GitHub using a name convention. The code should try to fetch any file matching "`BMI.en*.json`" from GitHub (because the file for a specific language could be either master or a translation).

E.g. creating a tool called "BMI" with language "en" should fetch the tool spec for BMI from Environment (the one you uploaded to GitHub).

2. Save this file to the model (field "`json_spec`").

3. From this JSON, create translation keys from all translatable nodes. Write keys and their values (translations in a specific language) to Lokalise using their API.

Translatable nodes are all nodes with string values. To create translations keys, start with tool name and join (concatenate) nodes in the path with an underscore. E.g. from this JSON:

```
{
  "language": "EN",
  "result": {
    "resultRanges": {
      "bmi": {
        "title": "BMI",
        "shouldShowPoints": true,
        "pointsFrom": -1.0,
        "activityLevel": "gray",
      }
    }
  }
```

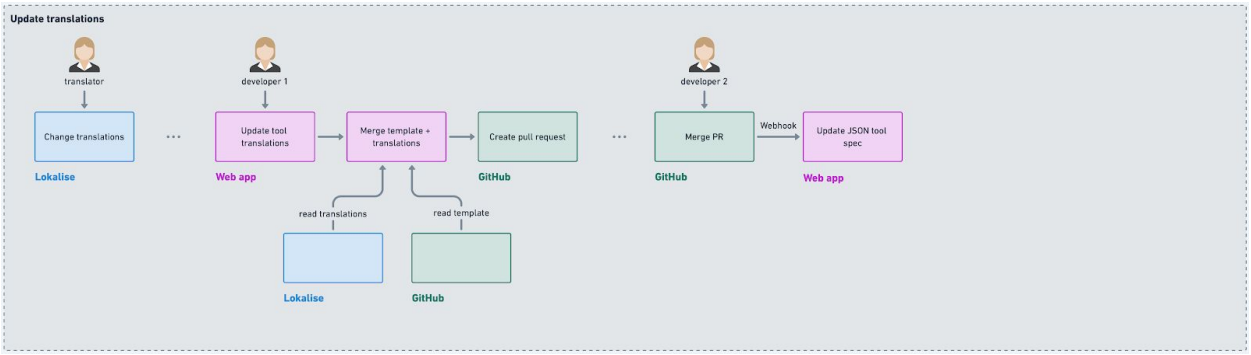
```
}
```

you would create these keys:

```
BMI_language
BMI_result_resultRanges_bmi_title
BMI_result_resultRanges_bmi_activityLevel
```

(You skip nodes "`shouldShowPoints`" and "`pointsFrom`" because their values are not strings so they don't need translation.)

Action 2: Update translations



(This happens after someone has updated translations on Lokalise.)

- User clicks "Update translations" on a tool (e.g. language "`it`")
- Fetch translations from Lokalise using API
- Fetch the master JSON from GitHub using API
    - e.g. for tool BMI, find file "`BMI.*.master.json`"
- Replace values in JSON with translations
- Instead of writing this JSON directly to model, write to file (e.g. "`BMI.it.json`") and create a pull request to merge into master

- *(... another user merges the pull request)*
- *(... a webhook fires, calling an endpoint in our web app)*

- Implement this endpoint to respond to GitHub's webhook(*) and write the new JSON to model (field "`json_spec`").

(*) To develop and test the GitHub webhook, please use **ngrok** or one of similar services on the web to expose localhost server publicly.