Appendix

Figure 1

Front-end criteria

1. Website has an always visible top bar with the language switcher on the top right
2. Language switcher is composed of language flag and language name. On mobile the language name is displayed as a 2 letter country code, e.g. EN for English
3. Clicking on the language switcher opens a full screen menu with the language selector
4. On the left side of the top bar there's an arrow pointing back, this arrow not visible on the homepage
5. Language selector has a search box on top, below which it displays the first 12 languages in alphabetical order with their corresponding flag
6. Visitors can use the search box to find their language using full language name or language code
7. When a user selects a language, the language selector closes and the website is displayed in the language they selected. Right to left languages are displayed with right to left formatting
8. When the user leaves and comes back to the website at a later stage, the site remembers their language selection
9. Detect browser language and display content in this language

Figure 2

UI criteria

1. When there isn't a flag there is a blank space e.g. Aymara
2. While searching the selection will decrease e.g. Af for Afrikaans
3. The search icon isn't clickable, it's just for decoration

Figure 3

Back-end criteria

1. Store language selected in cookies/local-storage for **xxx amount of time**
2. Retrieve selected language from database if it exists
3. If language does not exist then translate it using the API and store it in the database
4. Store static content translations in database

Figure 4

Front-end

- React - Stateful front-end framework
- Material UI - Design system for building UI components
- Storybook - Tool for developing and testing UI components in isolation
- Axios - Library for making HTTP requests

Back-end

- Node.js - JavaScript runtime
- Express - Node.js web application framework
- PostgreSQL - Open-source relational database

Deployment

- Heroku - Cloud platform for easy deployment for web applications

Translation

- Language translation API
- Internationalisation framework

Figure 5

Front-end folder structure

```
├── public
└── src
    ├── App.js        # App entry point
    ├── Hooks         # Custom react hooks
    ├── api-calls     # API calls
    ├── components    # Components stored in their own directories
    ├── constants     # Constant variables to be reused throughout front-
end
    ├── context       # Global contexts
    ├── helpers       # Reusable helper functions
    ├── index.js      # Index file that renders App
    ├── pages         # Individual page components
    ├── stories       # Storybook stories
    ├── theme         # Theme files for global theme values, colour, font,
etc
    └── validation    # YUP validation for the front-end
```

Figure 6

Back-end folder structure

```
src
  │   index.js        # App entry point
  └───config          # Environment variables and configuration related
stuff
  └───constants       # Constant variables to be reused throughout back-
end
  └───api             # Express route controllers for all the endpoints of
the
  └───modules         # Split the startup process into modules
  └───database        # Database models/data/...
  └───services        # All the business logic is here
  └───test            # Tests for the backend
```

Figure 7

```
$ git merge-base <feature_branch> develop

// example
$ git merge-base update-git-ignore develop
// Output commit hash: 74bc13f12ecb132415ada0c77a0cc9c4de65df92
```

Figure 8

```
$ git diff <commit id>..<feature_branch> > feature-patch.patch

// example
$ git diff 74bc13f12ecb132415ada0c77a0cc9c4de65df92..feature-branch >
feature-patch.patch
```

Figure 9

```
$ git apply --whitespace=fix --reject feature-patch.patch
```

Figure 10

```
Components                                   # Folder where all components
are
  └── Language                               # Language components
      ├── Language.stories.js                # Stories of all language
components
      ├── LanguageBar                        # Toolbar at the top of every
page
      │   ├── LanguageBar.stories.js         # Storybook file for toolbar
      │   ├── index.js                       # Entry point for component
      │   └── style.js                       # Emotion styling for toolbar
      ├── LanguageSelector                   # Menu with languages and
search
      │   ├── LanguageSelector.stories.js  # Storybook file for the
component
      │   ├── index.js                       # Entry point for component
      │   └── style.js                       # Emotion styling for
languageSelector
      ├── index.js                           # Both toolbar and selector
combined
      └── style.js                           # Styling for toolbar and
selector
```

Figure 11

```
const Desktop = ({dir}) => {

 const LTR = (
        <div>Left to right language bar goes here</div>
        )

 const RTL = (
        <div>Right to left language bar goes here</div>
        )

 return dir === 'ltr' ? LTR : RTL;
}
```

Figure 12

```
import { useTranslation } from 'react-i18next'
import { useMediaQuery } from 'react-responsive'

export const LanguageBar = () => {
        const { i18n } = useTranslation();
        const dir = i18n.dir()

        const isMobile = useMediaQuery({
          query: `(max-width: ${theme.breakpoints.mobile})`,
        });

        return isMobile ? <Mobile dir={dir} /> : <Desktop dir={dir} />;
}
```

Figure 13

```
import { useTranslation } from 'react-i18next'
import { useMediaQuery } from 'react-responsive'

const Desktop = ({dir}) => {
 const LTR = (
        <div>Left to right language bar goes here</div>
        )

 const RTL = (
        <div>Right to left language bar goes here</div>
        )

 return dir === 'ltr' ? LTR : RTL;
}

const Mobile = ({dir}) => {
 const LTR = (
        <div>Left to right language bar goes here</div>
        )

 const RTL = (
        <div>Right to left language bar goes here</div>
        )

 return dir === 'ltr' ? LTR : RTL;
}

export const LanguageBar = () => {
        const { i18n } = useTranslation();
        const dir = i18n.dir()

        const isMobile = useMediaQuery({
          query: `(max-width: ${theme.breakpoints.mobile})`,
        });

        return isMobile ? <Mobile dir={dir} /> : <Desktop dir={dir} />;
}
```

Figure 14

```
import styled from '@emotion/styled';
import theme from '../../../theme';

const commonStyle = `
  display: flex;  width: 100%;  align-items: center;  z-index: 2;
background-color: ${theme.colors.neutralSurface};
`;

const desktop = `
  height: ${theme.constants.translationBar.desktop.height};
  padding: ${theme.constants.translationBar.desktop.padding};
  justify-content: space-between;`;

const tablet = `
  height: ${theme.constants.translationBar.tablet.height};
  padding: ${theme.constants.translationBar.tablet.padding};
`;

export const TabletWrapperLTR = styled('div')`
  ${commonStyle};
  ${tablet};
  justify-content: ${({ showBack }) ⇒ (showBack ? 'space-between' :
'end')};
`;

export const TabletWrapperRTL = styled('div')`
  ${commonStyle};
  ${tablet};
  justify-content: ${({ showBack }) ⇒ (showBack ? 'space-between' :
'start')};
`;
```

Figure 15

```
export const languageCodes = {
  Afrikaans: 'af',
  Albanian: 'sq',
  Amharic: 'am',
  Arabic: 'ar',
  Armenian: 'hy',
  Azerbaijani: 'az',
  Bengali: 'bn',
  ...
```

Figure 16

```
  const languages = Object.entries(types.languageCodes)

// [
//   [ 'Afrikaans', 'af' ],
//   [ 'Albanian', 'sq' ],
//   [ 'Amharic', 'am' ],
//   [ 'Arabic', 'ar' ],
//   [ 'Armenian', 'hy' ],
//   ...
```

Figure 17

```
      <S.ButtonWrapper>
        {languages
          .map(([lng, code]) => {
            return (
              <S.Button onClick={() => changeLanguage({ lng })} key=
{code}>
                <TextWithIcon
                  text={lng}
                  iconProps={{
                    icon: FlagMap[code] !== undefined ? code : null,
                    pointer: true,
                    followLangDirection: false,
                  }}
                  {...props}
                />
              </S.Button>
            );
          })
          .slice(0, sliceTo)}
      </S.ButtonWrapper>
```

Figure 18

```
export const LanguageSelector = ({ hide, handleHide }) => {
  const [search, setSearch] = useState('');

  const languages = Object.entries(types.languageCodes).filter(
    ([lng, code]) => {
      return (
        code.toLowerCase().includes(search.toLowerCase()) ||
        lng.toLowerCase().includes(search.toLowerCase())
      );
    }
  );

...

  const Selector = (
    <S.Wrapper onClick={handleHide}>
        <BasicInput
          handleChange={(val) => setSearch(val)}
          label={common.section.changeLanguage.title}
          value={search}
          name="search-language"
          placeholder={common.section.changeLanguage.placeholder}
          suffix={<Icon icon="search" color="neutralMain" />}
        />

...
```

Figure 19

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import LanguageDetector from 'i18next-browser-languagedetector';

i18n
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    ns: ['common'],
    defaultNS: 'common',
    fallbacklng: 'en',
    preload: true,
    supportedLngs: [
      'af',
      'sq',
      'am',
 .    ...
    ],
    load: 'languageOnly',
    react: {
      useSuspense: true,
    },
  });

export default i18n;
```

Figure 20

```
import { query } from '../connect';

const createCommon = async ({ content }) ⇒ {
  const sql = `INSERT INTO common (
    content
  ) VALUES (
    $1
  ) RETURNING *`;
  const res = await query(sql, [content]);
  return res.rows[0];
};

const addCommon = async () ⇒ {
  const common = await createCommon({
    content: {
      buttons: {
        readMore: 'Read more',
        seeAdvice: 'See advice',
        goBack: 'Go back',
        stuckTalkToSomeOne: 'Stuck? Talk to someone',
        accessibility: 'Accessibility',
        decreaseTextSize: '- Decrease text size',
        increaseTextSize: '+ Increase text size',
        seeMore: 'See more',
        seeLess: 'See less',
...
```

Figure 21

```
import getCommon from '../modules/translations/use-cases/get-common'
import { content } from '../constants/data-type'
import { Router } from 'express';

const router = Router();

test('Fetch common texts and compare with static object', async () ⇒ {
  const response = await getCommon({ lng: 'en' });
  const { content: _content } = response[0]
  expect(_content).toEqual(content);
});
```

Figure 22

```
import { query } from '../../../database';

const getCommon = async (lng) => {
  const sql = `
  SELECT
    common.id,
    COALESCE (common_i18n.content, common.content) AS content,
    common_i18n.language_code
  FROM common
  LEFT OUTER JOIN common_i18n
    ON common.id = common_i18n.common_id
   AND common_i18n.language_code = $2
  WHERE common.id = $1
`;
  const res = await query(sql, [1, lng]);
  return res.rows;
};

export { getCommon };
```

Figure 23

```
import * as Translation from '../model';
import translateContent from '../../../services/translation/translate-
content';

const getCommon = async ({ lng }) => {
  const common = await Translation.getCommon(lng);

  const commonT = await translateContent({
    lng,
    contentArray: common,
  });

  commonT.forEach((c) => {
    if (!c.isTranslated) {
      Translation.createCommonI18n({
        commonId: c.id,
        languageCode: c.languageCode,
        content: c.content,
      });
    }
  });

  return commonT;
};

export default getCommon;
```

Figure 24

```
import getCommon from '../modules/translations/use-cases/get-common'
import { content } from '../constants/data-type'
import { Router } from 'express';

import { translateText, translateJSON, translate } from
'./../services/translation/translation-api'

describe("translateText", () => {
  const text = "hello"
  const sourceLang = "en";

  it("should correctly translate a string into French", async () => {
    const translatedString = await translateText({
      text,
      targetLang: 'fr',
      sourceLang
    });

    expect(translatedString).toEqual('bonjour')
  });

  it("should correctly translate the string of the input into German",
async () => {
    const translatedObj = await translateText({
      text,
      targetLang: 'de',
      sourceLang
    });

    expect(translatedObj).toEqual('hallo')
  });

});

describe("translateJSON", () => {
  const obj = {
    key1: "Hello",
    key2: {
      subKey1: "World",
      subKey2: "Goodbye"
    }
  };
  const sourceLang = "en";

  it("should correctly translate an object of strings into French", async
() => {
    const targetLang = 'fr'
    const translatedObj = await translateJSON({
      obj,
      targetLang,
      sourceLang
    });

    expect(translatedObj[targetLang].key1).toEqual('Bonjour')
    expect(translatedObj[targetLang].key2.subKey1).toEqual('Monde')
    expect(translatedObj[targetLang].key2.subKey2).toEqual('Au revoir')
  });

  it("should correctly translate an object into German", async () => {
    const targetLang = 'de'
    const translatedObj = await translateJSON({
      obj,
      targetLang,
      sourceLang
    });

    expect(translatedObj[targetLang].key1).toEqual('Hallo')
    expect(translatedObj[targetLang].key2.subKey1).toEqual('Welt')
    expect(translatedObj[targetLang].key2.subKey2).toEqual('Auf
Wiedersehen')
  });

});
describe("translate", () => {
  const obj = {
    key1: "Hello",
    key2: {
      subKey1: "World",
      subKey2: "Goodbye"
    }
  };
  const sourceLang = "en";

  it("should correctly translate the object input into French", async ()
=> {
    const targetLang = 'fr'
    const translatedObj = await translate({
      id: 1,
      json: obj,
      target: [targetLang],
      source: sourceLang
    });

    expect(translatedObj.content.key1).toEqual('Bonjour')
    expect(translatedObj.content.key2.subKey1).toEqual('Monde')
    expect(translatedObj.content.key2.subKey2).toEqual('Au revoir')
  });

  it("should correctly translate the object input into German", async ()
=> {
    const targetLang = 'de'
    const translatedObj = await translate({
      id: 2,
      json: obj,
      target: [targetLang],
      source: sourceLang
    });

    expect(translatedObj.content.key1).toEqual('Hallo')
    expect(translatedObj.content.key2.subKey1).toEqual('Welt')
    expect(translatedObj.content.key2.subKey2).toEqual('Auf Wiedersehen')
  });
});
```

Figure 25

```javascript
import AWS from 'aws-sdk';
import config from '../../config';
import { Sentry } from '../error-handler';
import { removeNullsAndEmptyArraysAndObjects } from '../../helpers';

const { awsAccessKeyId, awsSecretAccessKey, awsRegion } = config.aws;

const awsConfig = {
  accessKeyId: awsAccessKeyId,
  secretAccessKey: awsSecretAccessKey,
  region: awsRegion,
};

const translateAWS = new AWS.Translate(awsConfig);

const translateText = async ({ text = '', sourceLang, targetLang }) => {
  if (!targetLang || !sourceLang) {
    throw new Error('Missing source or target lang');
  }
  const params = {
    SourceLanguageCode: sourceLang,
    TargetLanguageCode: targetLang,
    Text: text,
  };

  try {
    const translationData = await
translateAWS.translateText(params).promise();
    return translationData.TranslatedText;
  } catch (error) {
    throw new Error('translateText API error :>> ', error);
  }
};

const translateJSON = async ({
  obj,
  targetLang,
  sourceLang,
}) => {
  const response = {};

  for (const key in obj) {
    let word = '';
    try {
      if (typeof obj[key] === 'object') {
        word = await translateJSON({
          obj: obj[key],
          targetLang,
          sourceLang,
        });
      } else {
        word = await translateText({ text: obj[key], sourceLang,
targetLang });
      }
    } catch (error) {
      Sentry.captureException(error);
      word = '';
    }
    response[key] = word;
  }
  return response;
};

const translate = async ({ source, target, json, id }) => {
  if (!source || !target || !json || !id) {
    throw new Error('translation API missing params');
  }

  const value = await translateJSON({
    obj: removeNullsAndEmptyArraysAndObjects(json),
    targetLang: target[0],
    sourceLang: source,
  });

  if (value) {
    return { id, content: { ...value[target] }, languageCode: target[0] };
  }

  throw new Error('translation API error');
};

export { translate };
```

Figure 26

```
import { useTranslation } from 'react-i18next';
import { types } from '../constants';
const useLanguage = () => {
const { i18n } = useTranslation();
const { languages = 'en' } = i18n;

  const lng = languages
    .find((val) =>
      Object.keys(types.languageCodes).find(
        (key) => types.languageCodes[key] === val.slice(0, 2)
      )
    )
    .slice(0, 2);

  const lngUpperCase = lng.toUpperCase();

  const lngFull = Object.keys(types.languageCodes).find(
    (key) => types.languageCodes[key] === lng
  );

  const flag = lngFull?.charAt(0)?.toLowerCase() + lngFull?.slice(1);

  const dir = i18n.dir();

  const hasNamespace = (namespace) => {
    return i18n.hasResourceBundle(lng, namespace);
  };

  return { lng, lngUpperCase, lngFull, flag, dir, hasNamespace };
};

export default useLanguage;
```

Figure 27

```
import axios from 'axios';
import handleError from './format-error';

const TRANSLATION_BASE = 'translations';

const getCommon = async ({ options, lng }) ⇒ {
  try {
    const { data } = await axios.get(`${TRANSLATION_BASE}/common`, {
      params: { lng },
    });
    return { data };
  } catch (error) {
    const err = handleError(error, options);
    return { error: err };
  }
};

export { getCommon };
```

Figure 28

```
import { useState, useEffect, createContext, useContext } from 'react';
import { Translations } from '../api-calls';
import { useTranslation } from 'react-i18next';
import { useLanguage } from '../helpers';
import { Outlet } from 'react-router-dom';

export const CommonContextData = createContext(null);

const CommonLogic = ({ children }) => {
  const { i18n } = useTranslation();
  const { lng } = useLanguage();
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchCommon = async () => {
      const { data, error } = await Translations.getCommon({
        lng,
      });
      const common = data?.[0]?.content;
      if (error) {
        // message.error('Something went wrong, please try again later');
      } else {
        i18n.addResourceBundle(lng, 'common', {
          common,
        });
        setData(common);
      }
    };
    fetchCommon();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [lng]);

  return (
    <CommonContextData.Provider value={{ data }}>
      {children}
    </CommonContextData.Provider>
  );
};

const useCommon = () => {
  const data = useContext(CommonContextData);
  return data;
};

const CommonProvider = () => {
  return (
    <CommonLogic>
      <Outlet />
    </CommonLogic>
  );
};

export { CommonProvider, useCommon };
```

Figure 29

```
const CommonLogic = ({ children }) => {
  const { i18n } = useTranslation();
  const { lng } = useLanguage();
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchCommon = async () => {
      const { data, error } = await Translations.getCommon({
        lng,
      });
      const common = data[0].content;
      if (error) {
                console.error(error)
      } else {
        i18n.addResourceBundle(lng, 'common', {
          common,
        });
        setData(common);
      }
    };
    fetchCommon();
  }, [lng]);
...
```

Figure 30

```
import { test, expect } from '@playwright/test';

test('Translating the page into French from English', async ({ page }) ⇒
{
  // Visit page
  await page.goto('http://localhost:3000/');

  // Check H1 is in English
  let pageH1 = page.locator('.css-14bizhd')
  await expect(pageH1).toHaveText(/Cost of Living Helper/);

  // Click the language selector
  await page.getByText('English').click();

  // Check menu has loaded
  const changeLanguageTitle = page.locator('.css-n79gqg')
  await expect(changeLanguageTitle).toHaveText(/Change language/);

  // Click French
  await page.getByText('French').click({ button: 'left' });

  // Check H1 is in French
  pageH1 = page.locator('.css-14bizhd')
  await expect(pageH1).toHaveText(/Aide au coût de la vie/);
});
```

Figure 31

```
uncaught exception detected error: there is no unique or exclusion
constraint matching the ON CONFLICT specification
```

Figure 32

```
CREATE UNIQUE INDEX
  common_id_language_code ON common_i18n
  (common_id, language_code);
```

Figure 33

```
const lng = languages
        .find((val) =>
            Object.keys(types.languageCodes).find(
            (key) => types.languageCodes[key] === val.slice(0, 2)
            )
        )
        .slice(0, 2);
```

Glossary

- **API**: Application Programming Interface, a set of protocols and tools for building software and applications
- **AWS**: Amazon Web Services, a collection of remote computing services
- **Beekeper Studio:** an SQL editor and database manager
- **CMS**: Content Management System
- **Controller**: A set of functions that handle the logic for a specific route or set of routes
- **ESLint**: Checks for code patterns that may indicate errors
- **GitHub**: Web-based platform for version control and collaboration
- **Jest**: A JavaScript testing framework
- **JSON**: JavaScript Object Notation, a lightweight data-interchange format
- **Postgres**: A free and open-source relational database management system
- **Playwright**: A Node.js library to automate web browsers for end-to-end testing and web scraping
- **Prettier**: Automatically formats code according to set rules
- **Query**: Specific request to a database to retrieve or modify data
- **React context**: Allows passing data through a component tree without having to pass it down manually at every level or make multiple requests
- **React hook**: A hook differs from a function because it uses state

- **Route**: Define which function(s) should be called in response to a request matching a specific path
- **Services folder**: The folder contains functions for tasks not related to handling HTTP requests, such as interacting with a database, making API calls, and complex calculations
- **Use-case**: Business logic that your application need to handle