SDEV2004 Software for Global Market 2

Localization

Lab Class Week 4

# What will I do in this lab class?

In this part of the lab, you are going to:

* Configure your FLASK\_APP variable so that you don’t need to keep changing it at environment level
* Setup a new Flask App
* Add some basic translation to this app for a single language
* Add an additional language

Contents

[Step 1: Set FLASK\_APP to a stable value 1](#_Toc127109577)

[1.a Create a file called main.py 1](#_Toc127109578)

[1.b Update FLASK\_APP 1](#_Toc127109579)

[1.c Change your sys.path in main 1](#_Toc127109580)

[1.d Install Flask Babel 2](#_Toc127109581)

[Step 2: Create a Flask app 3](#_Toc127109582)

[Step 3: Configure your app to use Flask Babel 4](#_Toc127109583)

[3.a Create an app configuration file 4](#_Toc127109584)

[3.b Create a Babel configuration file 4](#_Toc127109585)

[3.c Change your \_\_init\_\_.py 4](#_Toc127109586)

[Step 4: Implement the translation for German 5](#_Toc127109587)

[Step 4.a Mark some strings for translation 5](#_Toc127109588)

[Step 4.b: Extract the strings marked for translation 5](#_Toc127109589)

[Step 4.c: Generate a Message Catalog 5](#_Toc127109590)

[Step 4.d: Implement the translations 6](#_Toc127109591)

[Step 4.e: Compile the translation 6](#_Toc127109592)

[Step 4.f: Test your app 6](#_Toc127109593)

[Step 5: Mark another string and implement translation 7](#_Toc127109594)

[Step 6: Implement another translation 7](#_Toc127109595)

[Step 7: Implement another language 8](#_Toc127109596)

# Step 1: Set FLASK\_APP to a stable value

## 1.a Create a file called main.py

* This should be in the root folder of your project
* Include the following code:

import sys

# adding a folder to the system path

sys.path.insert(0, ‘the absolute path to the directory containing your app’)

#print(sys.path)

from app import app

* This code retrieves the system path PyCharm is aware off
* It then adds your directory Week3 to this path (if you called last weeks’ lab something else change it to reflect your naming)
* It then imports the flask app (from the path in this case the only directory that should include an app is Week3)
* The line print(sys.path) is commented out – it is included just in case you have problems and need to check what is on the path variable

## 1.b Update FLASK\_APP

* Locate the place where you set the value of FLASK\_APP
* (Settings/Preferences, Tools then Terminal then Environment variables)
* Set FLASK\_APP to main.py (you should copy the absolute path for this file and set FLASK\_APP to that value)
* RESTART PyCharm
* Try flask run – you should see the app we created in Week 3

## 1.c Change your sys.path in main

* Now if we want to run Week 2 for example – we simply change the sys.path.insert in the main.py to insert Week 2

E.g. sys.path.insert(0, '.\Week2')

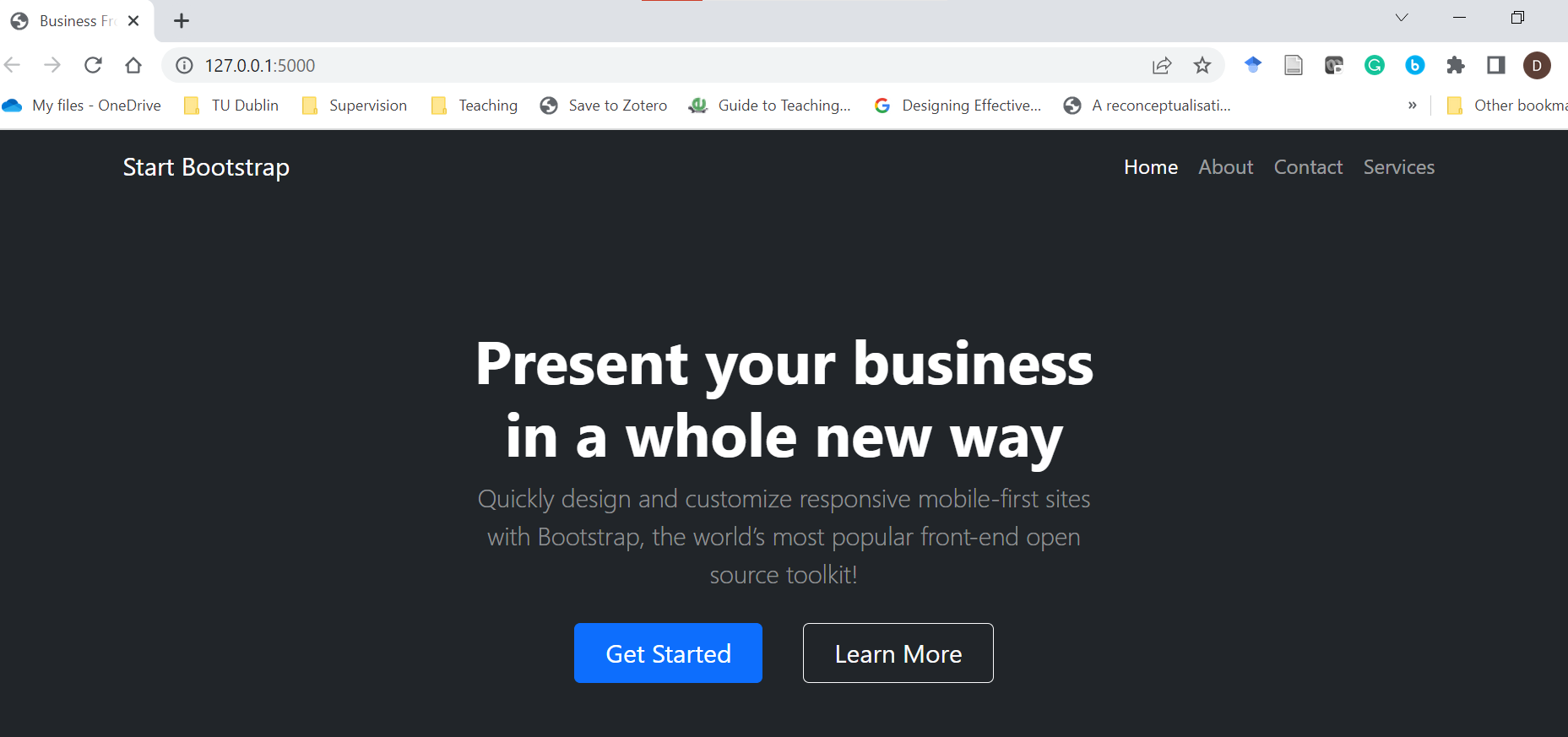
* We do not need to keep changing FLASK\_APP and restarting PyCharm
* Try it out

## 1.d Install Flask Babel

* Flask-Babel is an extension to Flask that allows you to add i18n and l10n support to any Flask application. It has builtin support for date formatting with timezone support as well as a very simple and friendly interface to gettext translations.
* Check to see if you have already installed it
  + pip list (in the Terminal window)
  + If you see flask-babel listed move to Step 2
    - If not pip install flask-babel

# Step 2: Create a Flask app

* Create a new **Directory** called **Week4**
* Create a New **Python** **Package** in this directory called **app**
  + Create two subfolders under app one called **static** andone called **templates**
* Download and unzip the zip file SDEV2004Week4.zip from Brightspace
* Copy the assets folder and all its content into the static folder
* Copy the css folder and all its content into the static folder
* Copy the index.html into the templates folder
* Note: this app uses a Bootstrap theme downloaded from <https://startbootstrap.com/themes>
  + This provided
    - A template
    - css, assets and js
* Create a routes.py file and create a route for the index file with a view that renders that template (index.html)
* Setup your \_\_init\_\_.py to run the flask app and import the routes you will define for the app
* Change your main.py to add Week4 to the path (instead of Week2/Week3)
* Test your app – you should see something like this (with the title usiness Frontpage - Start Bootstrap Template):



# Step 3: Configure your app to use Flask Babel

## 3.a Create an app configuration file

* In your app folder create a file called app.config
* In this file you
  + Set up the LANGUAGES dictionary where we list the languages we are going to support – in this case English and German.
  + Set up the directory where we will store the message catalogs (translations).
* Add the following lines:

LANGUAGES = {  
 *'en'*: *'English'*,  
 *'de'*: *'Deutsch'*}  
BABEL\_TRANSLATION\_DIRECTORIES = *'./translations'*

## 3.b Create a Babel configuration file

* In your app folder create a file called babel.cfg.
* Add the following lines:

[python: \*\*.py]  
[jinja2: templates/\*\*.html]

* These lines define the filename patterns for Python and Jinja2 template files respectively.
* These define the locations and types of file that will be searched for strings marked for translation.

## 3.c Change your \_\_init\_\_.py

* In your \_\_init\_\_.py add the following import statement

from flask\_babel import Babel

* After the line where you create the Flask app add the following lines:

# a function which at the moment returns a 2 letter string code for a locale

def get\_locale():  
 return 'en'

#Hook Babel into your app and pass the local returned by get\_locale to it

babel = Babel(app, locale\_selector=get\_locale)

# Step 4: Implement the translation for German

## Step 4.a Mark some strings for translation

* Edit index.html. Replace the title with:

**<title>{{\_('Business Frontpage - Start Bootstrap Template')}}</title>**

* This marks the string for translation.

Pybabel will look for text enclosed between \_()

We wrap that between {{}} to indicate that this is dynamic content.

## Step 4.b: Extract the strings marked for translation

* In your Terminal window make sure you are in your **app** folder
* Execute the following command:

**pybabel extract -F babel.cfg -k \_l -o messages.pot .**

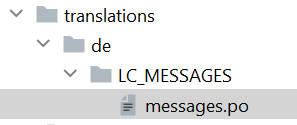
* The pybabel extract command reads the configuration file given in the -F option
* It then scans all code and template files in the directories that match the configured sources, starting from the directory given in the command (the current directory or . in this case).
* By default, pybabel will look for \_() as a text marker, but we may also use the lazy version, so you can import \_l() as well so use -k and \_l switches
* The -o option provides the name of the output file (messages.pot).

## Step 4.c: Generate a Message Catalog

* You now need to create a translation for each language that will be supported in addition to the base one (English)
* We are supporting German – we use the locale code de:

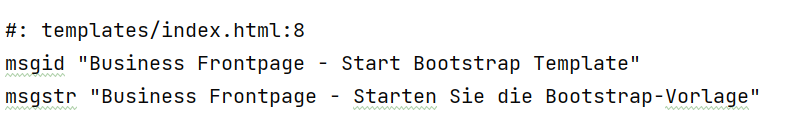
**pybabel init -i messages.pot -d translations -l de**

* This will result in the following folder and files being created in your app:



## Step 4.d: Implement the translations

* Edit the messages.po file
* Locate the msgid that has the text you marked for translation
* Translate the text using Google translate
* Insert the translation after the msgstr tag associated with the msgid

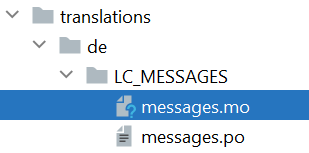


## Step 4.e: Compile the translation

* Execute the following command:

**pybabel compile -d translations**

* This will create a file called messages.mo in LC\_MESSAGES sub-folder of de:



## Step 4.f: Test your app

* In \_\_init\_\_.py change the locale returned in get\_locale to be 'de'
* Launch the app -you should see your translated string displayed

# Step 5: Mark another string and implement translation

* Locate the string in index.html

Present your business in a whole new way

* Replace it with:

**{{\_('Present your business in a whole new way')}}**

* Extract the marked strings

**pybabel extract -F babel.cfg -k \_l -o messages.pot .**

* Update the message catalogs for the languages supported

**pybabel update -i messages.pot -d translations**

* Implement the translation in message.po for de
* Recompile your translation

**pybabel compile -d translations**

* Test your app

# Step 6: Implement another translation

* Mark another string in index.html
* Extract the marked strings
* Update the message catalogs
* Implement the translation
* Recompile the message catalog for the new language
* Test your app

# Step 7: Implement another language

* Choose a language
* Locate the 2 character ISO code for that language (<https://www.sitepoint.com/iso-2-letter-language-codes/>)
* Edit app.config add the 2 char code to the list of languages e.g.

LANGUAGES = {  
 *'en'*: *'English'*,  
 *'de'*: *'Deutsch'*,  
 *'fr'*: *'French'*}

* Create a message catalog for it

pybabel init -i messages.pot -d translations -l your2charcode

e.g.

pybabel init -i messages.pot -d translations -l fr

* Edit the messages.po file for this locate in the translations folder
* Implement the translations
* Compile the translations for the new language

e.g.

pybabel compile -d translations

* Change get\_locale in \_\_init\_\_.py to return the 2 char language code
* Test your app

**END OF LAB**

**++++++++++++++++++++++++++++++++++++++++++**