SOP OF IDN-UA COMPLIANCE FOR INDIAN GOVERNMENT WEBSITES

Support document to enable URLs from English to Multilingual Indian languages

MARCH 28, 2023

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING
C-DAC Innovation Park, Panchavati, Pashan, Pune - 411008

Table of Contents

Python Django Configuration

Prerequisites-

Installation of GNU gettext toolkit-

1. This GNU gettext toolkit is used to generate and manage a plain text file known as the message file. Using this we can easily mark strings for translation, both in Python code and in our templates. It makes use of the the message file ends with *.po* as its extension. Another file is generated for each language once the translation is done, which ends with the *.mo* extension. This is known as the compiled translation.

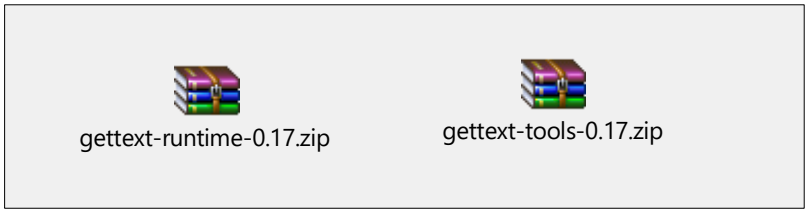
   We can install the gettext toolkit as given below,

   a. On linux, it's recommended to use Homebrew to install gettext toolkit:

   ```
   $ brew install gettext
   $ brew link --force gettext
   ```

   b. On windows –

   Download below files as packages.



   - Go to C Drive → Program Files
   - Create a folder (gettext-utils)
   - Open gettext-utils folder
   - Copy and paste above two zip files into gettext-utils folder
   - select both file together and extract
   - copy path of bin folder and add this to new path of environment variable→ Restart Computer

Django settings.py settings-

Django comes with some default internationalization settings in the *settings.py* file:

```
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True
```

a. The first setting is the `LANGUAGE_CODE`. By default, it's set to United States English (en-us). This is a locale-specific name. Let's update it to a generic name, English (en).

```
LANGUAGE_CODE = 'en'
```

b. Let's add some additional settings to complement the existing ones:

Add a LANGUAGES section having list of all the Indian languages in which you want your website to support.

```
from django.utils.translation import gettext_lazy as _

LANGUAGES = (
    ('en', _('English')),
    ('hi', _(Hindi)),
    ('mr', _('Marathi')),
)
```

c. Add `django.middleware.locale.LocaleMiddleware` to the `MIDDLEWARE` settings list. This middleware should come after the `SessionMiddleware` because the `LocaleMiddleware` needs to use the session data. It should also be placed before the `CommonMiddleware` because the `CommonMiddleware` needs the active language to resolve the URLs being requested. Hence, the order is very important.

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware', # new
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

d. Add a locale path directory in your application where message files will reside:

```
LOCALE_PATHS = [
    BASE_DIR / 'locale/',
]
```

You need to create the "locale" directory inside of your root project and add a new folder for each language.

```
locale
├── en
├── hi
└── mr
```

Django looks at the LOCALE_PATHS setting for translation files. Locale paths language codes that appear in order first have the highest precedence.

e. Open the terminal and run the following command from your project directory to create a *.po* message file for each language:

```
(env)$ django-admin makemessages --all --ignore=env
```

Currently, only the `LANGUAGES` from our *settings.py* file have been marked for translation. Therefore, for each `msgstr` under the "hi"(Hindi) and "mr"(Marathi) directories, edit *.po* files from your regular code editor and enter the Hindi or Marathi equivalent of the word manually, respectively.

f. Compile the messages by running the following command:

```
(env)$ django-admin compilemessages --ignore=env
```

A *.mo* compiled message file would be generated for each language.

```
locale
├── en
│   └── LC_MESSAGES
│       ├── django.mo
│       └── django.po
├── hi
│   └── LC_MESSAGES
│       ├── django.mo
│       └── django.po
└── mr
    └── LC_MESSAGES
        ├── django.mo
        └── django.po
```

Translation of template static data –

To translate templates static data, Django offers the {% trans %} and {% blocktrans %} template tags. You have to add {% load i18n %} at the top of the HTML file to use the translation templates tags.

The {% trans %} template tag allows you to mark a literal for translation. Django simply executes the gettext function on the given text internally.

Note:

- The {% trans %} tag is useful for simple translation strings, but it can't handle content for translation that includes variables.
- The {% blocktrans %} template tag, on the other hand, allows you to mark content that includes literals and variables.

After inserting trans tag in your template run following command-

```
(env)$ django-admin makemessages --all --ignore=env
```

Update the msgstr translations in .po files inside respective language directory and then compile using below command:

```
(env)$ django-admin compilemessages --ignore=env
```

Translating Dynamic Data(Models with django-parler)-

django-parler is a third party library package, which is required to create a separate database table containing translation for each model. This table includes all of the translated fields. It also has a foreign key to link to the original object.

To install django-parler run the following command-

Pip install django-parler

a. Add it to your installed apps
b. Add the following code to your settings.py file:

```
PARLER_LANGUAGES = {
    None: (
        {'code': 'en',}, # English
        {'code': 'hi',}, # Hindi
        {'code': 'mr',}, # Marathi
    ),
    'default': {
        'fallbacks': ['en'],
        'hide_untranslated': False,
    }
}
```

c. Create model using TranslatableModel, TranslatedFields in models.py file :

```
from django.db import models
from parler.models import TranslatableModel, TranslatedFields


class Course(TranslatableModel):
    translations = TranslatedFields(
        title=models.CharField(max_length=90),
        description=models.TextField(),
        date=models.DateField(),
        price=models.DecimalField(max_digits=10, decimal_places=2),
    )

    def __str__(self):
        return self.title
```

Next, create the migrations using below commands:

```
(env)$ python manage.py makemigrations
```

Next, apply the migrations:

```
(env)$ python manage.py migrate
```

d. Edit *admin.py* like so:

```
from django.contrib import admin
from parler.admin import TranslatableAdmin

from .models import Course

admin.site.register(Course, TranslatableAdmin)
```

Run the server, and then navigate to http://127.0.0.1:8000/admin/ in your browser. Select one of the courses. For each course, a separate field for each language is now available.

To make website URL domain Universal Acceptance (UA) compliant-

List website domain URLS in all Indian Languages search for all domains-

1. Convert all unicodes of domain names into punycode and add to ALLOWED_HOSTS section of settings.py file as shown below-

```
ALLOWED_HOSTS = ['*','travela-example.com','travela-example.bharat','यात्रा-उदाहरण.भारत', 'xn----5td2fbm4cid4fzbcd3q.xn--h2brj9c','प्रवास-उदाहरण.भारत','xn----5td2flr0bd0b1an4ce1q.xn--h2brj9c']
```

1. Create a domain json file in app directory and add all punycode domain with language code as shown below-

```
[

    {
        "en": "http://travela-example.bharat:80"
    },
    {
        "hi": "http://xn --- 5td2fbm4cid4fzbcd3q.xn--h2brj9c:80"
    },
    {
        "mr": "http://xn --- 5td2flr0bd0b1an4ce1q.xn--h2brj9c:80"
    }

]
```

1. Now add a blank url for setting language code for requested domain name url in main urls.py file as shown below-

```
s.  path(('''), app_views.landingfunction, name='home'),
```

1. Write a function for above url for setting language code-
   This function first check requested domain, according to domain it will set language code and return home.html page.

```
    requested_domain_without_port=main_domain.split(':')[0]
    with open('app\domains.json', 'r',encoding="utf8") as j:
            dom = json.loads(j.read())
```

```
for content in range(len(contents)):
        for key, value in contents[content]['mainpath'].items():
                path_render=contents[content]['mainpath'][lang]

  response.set_cookie(settings.LANGUAGE_COOKIE_NAME, lang)
  return response
```

Language conversion on changing language using select drop down box-

1. Add all urls for all pages into urls.py file with gettext_lazy function for url name conversion as shown below.

```
from django.utils.translation import gettext_lazy as _
path(_('aboutus'), app_views.aboutus, name='aboutus'),
path(_('services'), app_views.services, name='services'),
```

2. Create custom tag inside application directory-
   Add below lines to custom_tag.py file.

```
@register.tag(name='translate_url')
def do_translate_url(parser, token):
    return TranslatedURL(language)


class TranslatedURL(template.Node):
    def render(self, context):
        request_language = self.language with
        open('app\paths.json', 'r',encoding="utf8") as j:
            contents = json.loads(j.read())

        for content in range(len(contents)):
                for key, value in contents[content]['mainpath'].items():
                        path=contents[content]['mainpath'][request_language]
        returnpath=domain+path
        return returnpath
```

3. Add below drop down box code for language switch in html page-
   Note-First load custom tag as {% load custom_tag %} in html page

```
<li><a class="dropdown-item" href="{% translate_url en %}" hreflang="en">English</a></li>
<li><a class="dropdown-item" href="{% translate_url hi %}" hreflang="hi">Hindi-हिंदी</a></li>
<li><a class="dropdown-item" href="{% translate_url mr %}" hreflang="mr">Marathi-मराठी</a></li>
```

4. Add a url for multilingual path request into main urls.py file –

```
path("<path>", app_views.RenderPageWithPathAndLang,
name="RenderPageWithPathAndLang"),
```

5. Add a function for above url for multilingual path request into view file –

```
    with open('app\paths.json', 'r',encoding="utf8") as j:
        contents = json.loads(j.read())
    for content in range(len(contents)):
            for key, value in contents[content]['mainpath'].items():
                    path_render=value

      response.set_cookie(settings.LANGUAGE_COOKIE_NAME, lang)
      return response
```

Here you have completed Multilingual Python Django Project setup.

IDN domain-compliant examples

Some of the Indian government websites used Drupal as CMS and IDN compliant.

Government Website (.gov.in/.सरकार.भारत)

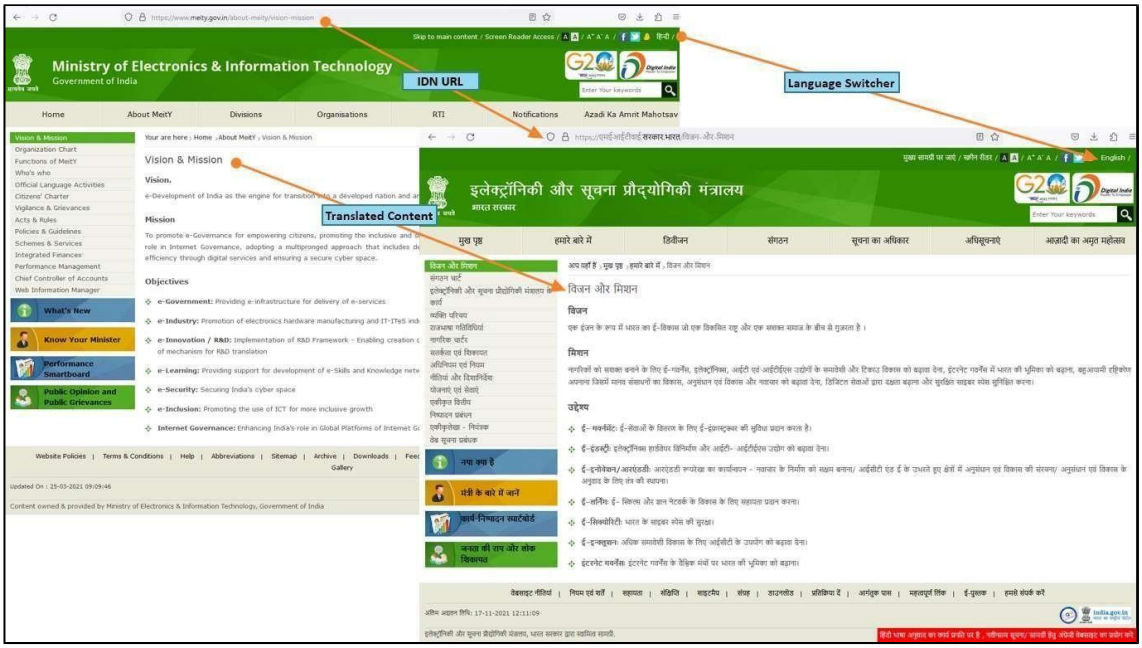


Figure 19: www.meity.gov.in

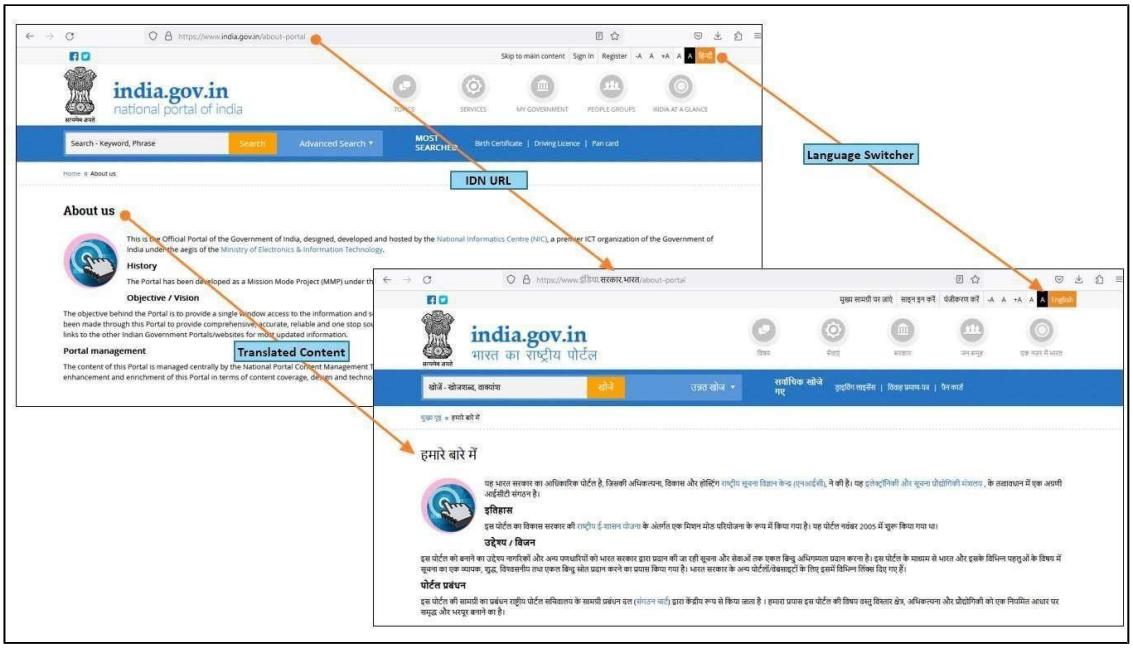


Figure 20: www.india.gov.in

Indian Domain Website (.IN/.भारत)

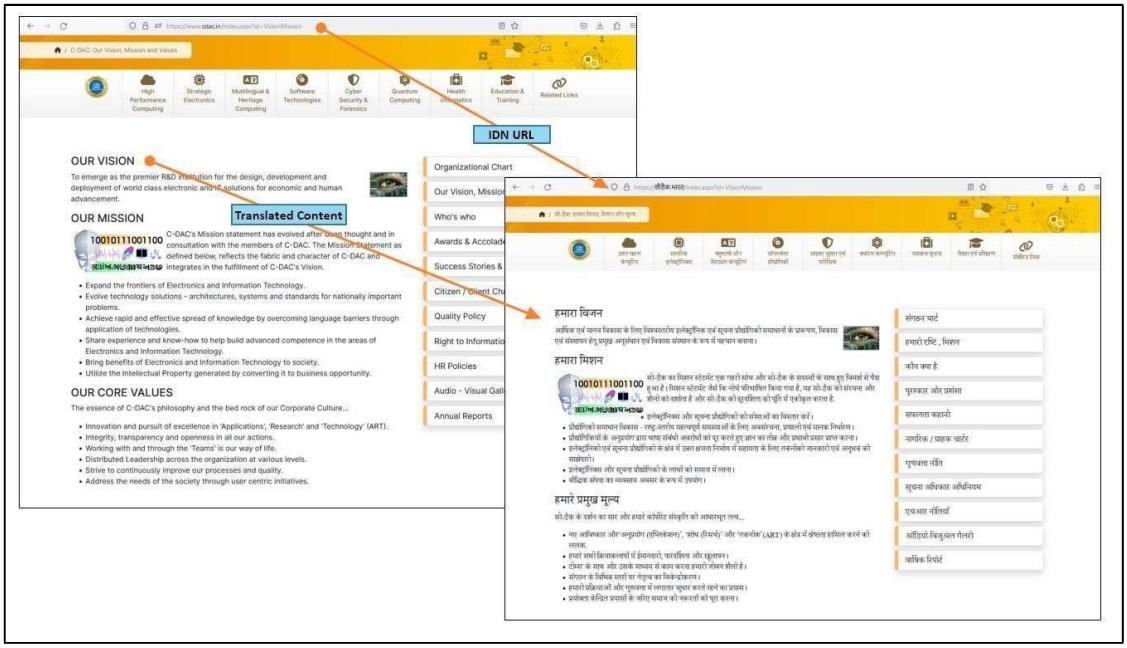


Figure 21: www.cdac.in

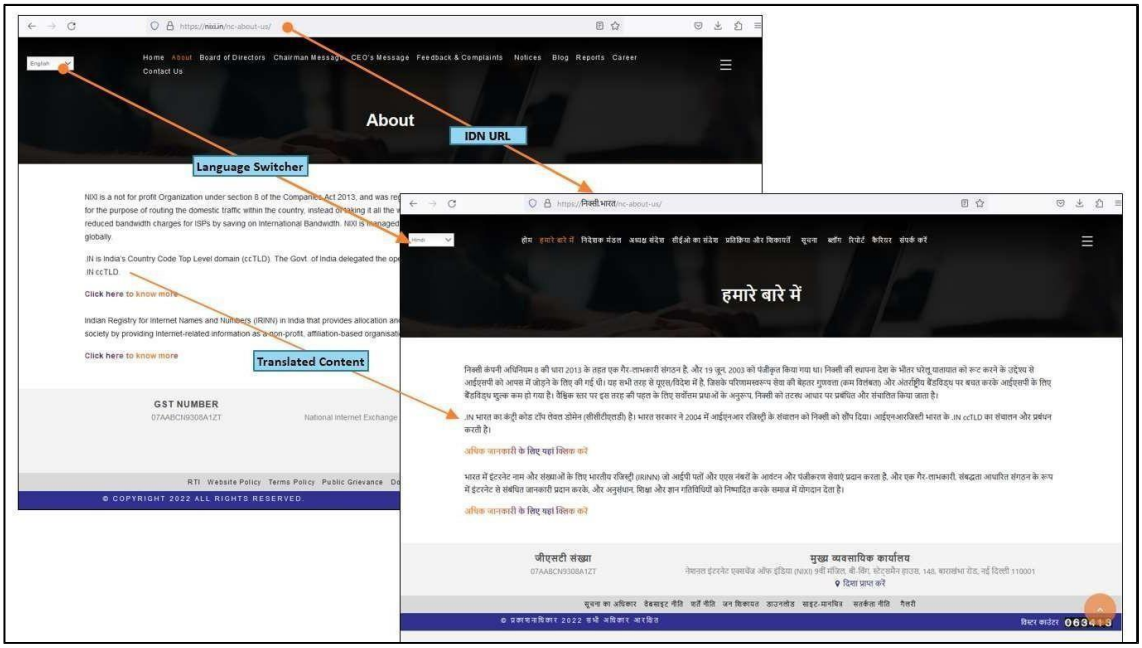


Figure 22: www.nixi.in

Thank You

Centre for Development of Advanced Computing

C-DAC Innovation Park,

Panchavati, Pashan,

Pune - 411 008, Maharashtra (India)