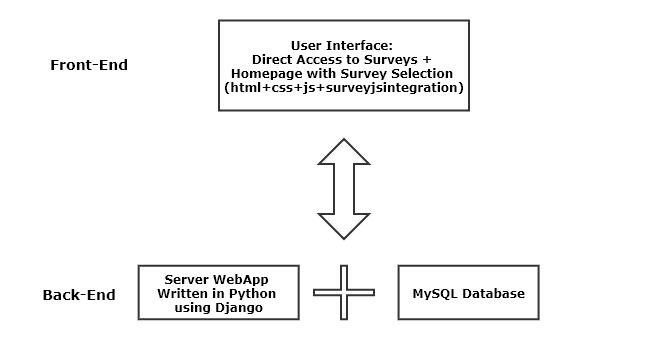
**Surveys WebApp**

**Introduction**

We want to write an application that will allow us to integrate the SurveyJS library while building surveys. In order to achieve this we will need to set up a server and to use and combine many programming languages. The structure will be the following:



The Users will access directly the surveys or through the Homepage where they will be redirected to the survey that they need to complete.

To edit the code, you can use any text editor you want. We used PyCharm as the Python IDE and Visual Studio Code to edit .css and .js files.

\*License Warning: the SurveyJS library is free to use and under the MIT License which means that we can integrate it in our project. What we can’t do, without buying it, is inserting their Survey Builder system in one of our webpages\*

**Specifications**

We want each survey to be accessed only by those who have a specific access code. This is an alternative to the authentication system, so users won’t have to go through a registration form to access surveys but will only need the key. Once a user logs in a specific survey we will grant him access for a set amount of time (a week) using a cookie stored on his system. Again, once completed, a survey will generate another cookie that will prevent a user from taking it more than one time.

**Setup Django**

The first thing we want to do is to set up our Django server that will be the starting point of our application. This are the steps to take to download it and start the project:

1. Download and install the latest version of Python (3.6.5 at the time of writing) <https://www.python.org/downloads/>
2. Install the latest version of pip <https://pip.pypa.io/en/latest/installing/>
3. Using the Command Prompt, write the following commands:
   1. mkvirtualenv env //creates a python virtual environment for our project
   2. workon env //use the virtual environment just created
   3. pip install Django //install Django in our environment
   4. django-admin startproject surveywebapp //sets up our Django project creating all the files/folders needed
   5. python manage.py runserver //with this command we have our Django application running on a local server

**General Settings and Database Setup**

By creating a Django project, we did set up the skeleton of our WebApp. We now have to set up the general settings modifying the files that were created in the process and then we will have to create as many Django applications as the number of surveys we want to have active on our website.

\*Warning: the following settings are suitable for the debug environment and might need to be changed in production\*

Within the setting.py file change/add the following sets of code:

Used to Change the time for each webpage of our project  
  
Sets the filepath for our static files like images, fonts and .css/.js files   
  
This is the basic configuration used to connect our application to the MySQL Server

TIME\_ZONE = 'Europe/Berlin'

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [  
 os.path.join(BASE\_DIR, "static"),  
]

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'surveytest',  
 'USER': 'root',  
 'PASSWORD': 'root',  
 'HOST': 'localhost', # Or an IP Address that your DB is hosted on  
 'PORT': '3306',  
 }  
}

At this point we want to download the MySQL server, configure it to work with Django and create the database that we will work on.

1. Download and install the server: <https://dev.mysql.com/downloads/windows/installer/8.0.html>
2. Download and install the Python connector: <https://dev.mysql.com/downloads/connector/python/>
3. Log in the MySQL Server with the admin user and create an empty database:  
   create database surveytest
4. Within the Command Prompt, execute the following commands:
   1. pip install Django mysqlclient //this will install the Django library necessary to connect to the MySQL server
   2. python manage.py makemigrations  
      python manage.py migrate //these two commands will set up the database with the Django tables that are necessary to work on it

At this point we have everything set up regarding database and Django configurations.

**Create the Application**

The way Django works is that you can have many applications per project. Each application can work independently from the others and have its own set of rules. To create the app, we want to work on write this command in the Command Prompt:

python manage.py startapp surveyweb

Now we need to tell the Django project that we added this new app by doing this:

1. In settings.py modify this set of code as follow:

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'websurvey',  
]

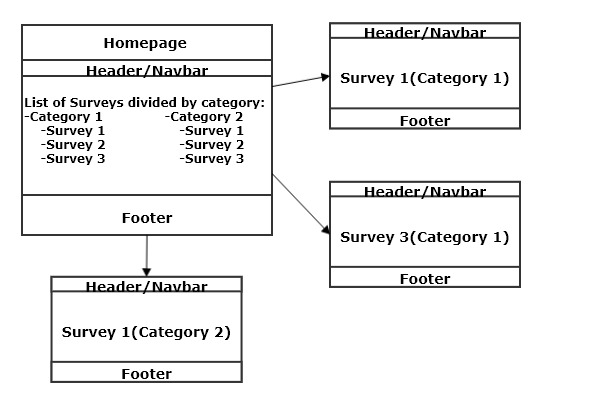
1. In the urls.py you want to have the following code:

from django.contrib import admin  
from django.urls import path  
from django.conf.urls import url, include  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 url(r'^', include('websurvey.urls')),  
]

The urls.py file is a configuration file that maps the website. Whenever a user accesses a specific webpage of our application the urls.py file gets the request and redirects it to the right view. The view has inside it the logic necessary to process that request and will eventually get back to the one who made the request sending data or a template. A template is an html file and is stores in a folder called templates.

**Survey Web Application**

We can start now developing the application itself. We can imagine that the website will be structured as follows:



As said before, users will be able to access surveys through a direct link or clicking on the specific link in the homepage. Watching the map, we see that all the pages have something in common: the header and the footer. In Django we can create a base template that is common to all the pages and that they will inherit the code from. The first thing to do is to create it.   
To do so modify the urls.py file of the app as follows:

from django.conf.urls import url  
from websurvey import views  
  
urlpatterns = [  
 url(r'^$', views.HomePageView.as\_view(), name='home'),  
]

This will tell Django that each time a user accesses the website without specifying a specific survey he will be redirected to the homepage view.

Now add this to the views.py file:

from django.shortcuts import render  
from django.views.generic import TemplateView   
  
# View for the Homepage  
  
class HomePageView(TemplateView):  
 def get(self, request, \*\*kwargs):  
 return render(request, 'Home.html', context=None)

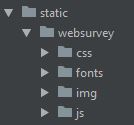
The view HomePageView will redirect users to the Home.html template that we will now create within the template folder.

The home.html will have the normal html structure. We only need to add Django template tags:

1. {% load static %} //first line of the html file
2. {% block title %} {% endblock title %} //inserted in the head section to modify the title
3. {% block css %} {% endblock css %} //inserted in the head section to add css
4. {% block content %} {% endblock content %} //inserted in the body
5. {% block js %} {% block js %} //inserted in the body as the last row to add js

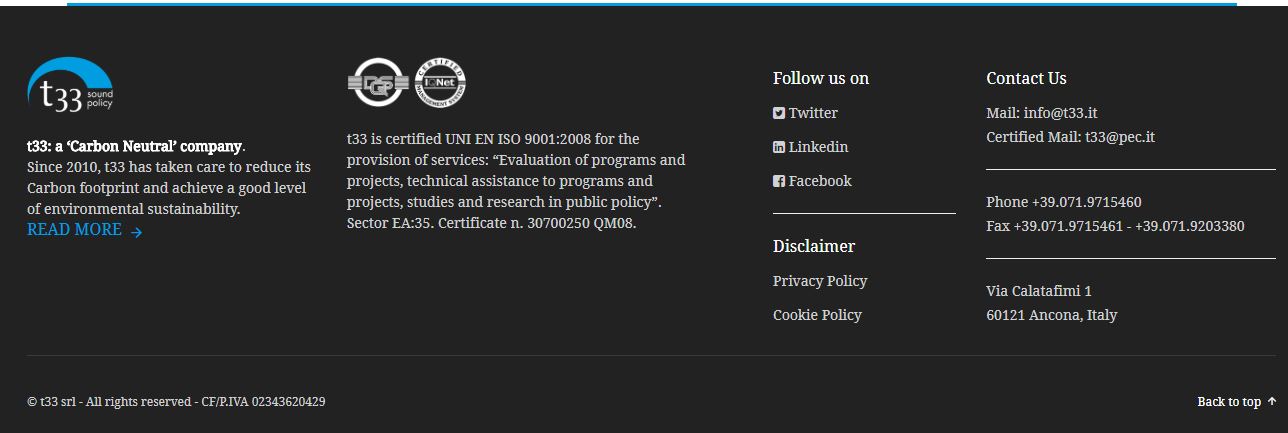
Everything that is found within these tags (except for the first one) will change in the other pages while the rest will be inherited.

We now want to create a folder that will include all the files that are not editable like images or .css files. We will call it static and will have the following structure:



At this point we can write the html/css/js code needed to create the header and the footer that will be common to every page on our website and put it outside of the Django template tags. We will end up with something like this:





The Active button in the navbar will lead the user to the section listing the active survey, while the Ended button will do the same for the ended ones.

Now, within the block content template tag we will insert the code to generate the homepage content and the lists of the surveys.

\*Warning: each link leading to a webpage within the Django WebApp needs to be written using the Django template tag {% url ‘PageName’ %}, while each static file need to use the tag {% static ‘PathToFile’ %} \*

We can now create the surveys. In this documentation we will go through the creation of one specific survey, the crossborder one, since the creation of the others is similar.

**Creation of a Survey**

Our first move will be to add the link to the new survey page in the urls.py file:

url(r'^crossborder/$', views.CrossBorderView.as\_view(), name='crossborder'),

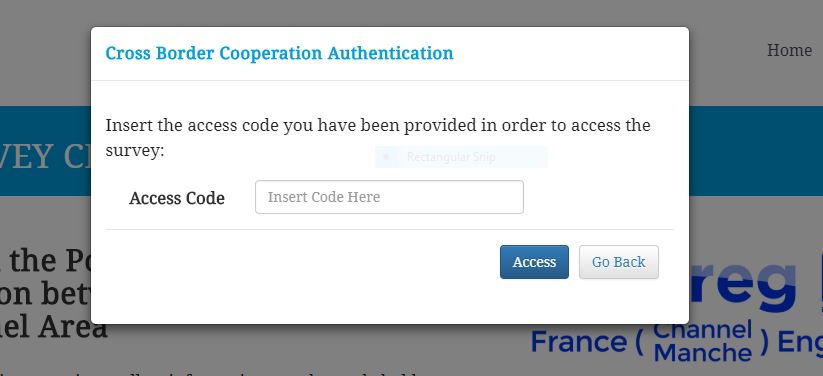
The next thing to do will be to add the view as we did with the homepage:

class CrossBorderView(TemplateView):   
 template\_name = "crossborder.html"

Now we can create the template crossborder.html and add as first line the template tag {% extends “Home.html” %} which will inherit from the home template everything that is outside the Django template tags.

As we stated into the specifications section, we don’t want to set up an authentication system, but we want users to access the survey by knowing an access code. The way we can do it is to make a pop up appear before the survey starts to allow users to enter the code.

We will end up with something like this:



Before we can proceed with the system that will check if the code is right, we need to create a table within the database to store all the keys for the relative surveys. In Django we create a table modifying the model.py file as follows

class surveyauth(models.Model):  
 key = models.CharField(max\_length = 20, primary\_key = True)  
 survey = models.CharField(max\_length = 255)

and then we do execute again the migrations commands that we used in the section where we integrated the database with Django. This will create a table with two fields, one containing the name of the survey and one the access key needed to be able to complete it.

Let’s now add one more line of code to the crossborder view:

obj = surveyauth.objects.get\_or\_create(key='crossborder2018', survey='Cross Border')

This will add the key into the database for our survey in case it’s the first time we open it up.

Now, coming back to the PopUp, we will send an Ajax POST request to the server when the user inserts the code and clicks on the Access button. This will activate another view on the server defined as follows:

# urls.py

url(r'^crosborder/check\_key/', views.check\_key, name='check'),

# views.py

def check\_key(request):  
  
 response\_data = {}  
  
 try:  
 AccessCode = surveyauth.objects.get(key=QueryDict(request.body).get('key'))  
 except surveyauth.DoesNotExist:  
 AccessCode = "fail"  
  
 if AccessCode == "fail":  
 response\_data['msg'] = 'Not Found'  
 else:  
 response\_data['msg'] = 'Found'  
  
 return HttpResponse (  
 json.dumps(response\_data),  
 content\_type='application/json'  
 )

The view will check if the key is present in the database and is correct and return a Found/Not Found message to the ajax. At this point, if the key is wrong the user will get an error message, otherwise he will access the survey and a cookie will be created. The cookie will grant access to the survey for 7 days so that the user will not have to insert the key every time he accesses it.

Now we can embed the survey that we created with the online Survey Builder and style it using our own css code.

The last operation to do is to save the results of the survey in the database, using again an Ajax request. Since the result is given in json, we need to integrate another library to save it in our MySQL database. In the Command Prompt write this command: pip install django-mysql.

Let’s now create the model in models.py:

from django\_mysql.models import JSONField

class surveysubmission(models.Model):  
 surveyJSON = JSONField()

And the relative view:

# urls.py

url(r'^crossborder/save/', views.save, name='save'),

# views.py  
  
def save(request):  
  
 response\_data = {}  
  
 s = surveysubmission(surveyJSON = QueryDict(request.body).get('survey'))  
 s.save()  
 response\_data['msg'] = 'Saved'  
  
 return HttpResponse (  
 json.dumps(response\_data),  
 content\_type='application/json'  
 )

At this point, if the Ajax request ended successfully, the result will be saved in the database, the user will be able to leave and a cookie will be created to prevent him from taking the survey again, otherwise he will get an error message and a button to start the survey again will appear.