Documentation: Admission and Recruitment Portal

1. ***Configuration File Specifications.***

Django MVC framework has been used to build the admission portal. Both Model and View of the webapp (admission portal) is configurable and created/displayed based on the configuration parameters specified/set in the configuration file. However one needs to adhere to format described in this document, in order to pass right configuration parameters to build and deploy the webapp.

Following are the specifications towards writing correct configuration for the webapp.

1. At the top level, configuration file has five major components. Namely,
    a. **AdmissionInfo**.
    In this construct, one needs to compulsorily specify the following four attributes.
        i. *AdmissionDegree* - E.g. : “PhD”, “MTech” etc.
        ii. *AdmissionType* - E.g. : “Full Time” “Part Time” etc.
        iii. *AdmissionMonth* - E.g. : “January”
        iv. *AdmissionYear* - E.g : “2019”

    Note that the attributes mentioned above are compulsory as they appears in the home page of admission portal.

    b. **PersonalDetails**.
    In this construct we specify the fields required from the user as part of personal details. Every required field of personal details is specified as one *PersonalAttr* construct (Note that *PersonalAttr* is a sub-construct inside the top level *PersonalDetails* construct). Each *PersonalAttr* will form an independent column in database. Following are the details which need to be specified for every *PersonalAttr* (We make this more clear by giving example of *PersonalAttr* namely “Primary Address” or “Category”).
        i. *PersonalAttrName* - This will be used assign column name to the personalAttr in database. Try to keep this a single word (without spaces). E.g. “PrimaryAddressLine”.
        ii. *PersonalAttrLabel* - This is what would be displayed on UI/view. E.g. “Enter your Primary Address”
        iii. *[IsTypeString / IsTypeInteger / IsTypeFloat / IsTypeDate / IsTypeBoolean]* - Based on type of input expected only one of the five is set to True.
        E.g. Address being String type, only *IsTypeString* will be set to True.

iv. *PersonalAttrChoices* - The values specified here would be rendered to the user as options to choose from. The values should be specified as a list with elements separated by commas and each element inside a single quote. Otherwise specify as empty list.
E.g. For *PersonalAttr* a sample specification could be link - *{ "General", "OBC(creamy layer)", "OBC non-creamy layer", "SC", "ST" }*

v. *[StringConstraints / IntegerConstraints / FPConstraints / BooleanConstraints / DateConstraints]* - See the Constraint Specification section below.

vi. *MultipleEntries* - Set this to True if more than one entry is acceptable for the *PersonalAttr* under consideration.

vii. *IsOptional* - Set this to True if the *PersonalAttr* under consideration is optional.

c. **EducationalQualifications**.
In this construct we specify the fields required from the user as part of educational details. The top level component *EducationalQualifications* is composed of *EduAttrName* and one or more *EduAttr.*
EducationalQualifications = {EduAttrName}{EduAttr}+.
EduAttr defines the top level attributes like degree description, gate details, degree from iit etc. Whereas *SubAttr* constructs inside *EduAttr* are to query particular details about *EduAttr* it is enclosed within. Eg. *EduAttr "GateDetails"* can have *SubAttr* like *"GateScore", "GateRank"* etc.
Every *SubAttr* should be defined with following fields.

i. *SubAttrName* - Along with *EduAttrName t*his will be used assign column name to the *SubAttr* in database.
E.g. "*GateDetails_GateScore*".

ii. *SubAttrLabel* - Analogous to *PersonalAttrLabel* in *PersonalAttr.*

iii. *[IsTypeString / IsTypeInteger / IsTypeFloat / IsTypeDate / IsTypeBoolean]* - analogous to *Type field in PersonalAttr.*

iv.

v. *[StringConstraints / IntegerConstraints / FPConstraints / BooleanConstraints / DateConstraints]* - See the Constraint Specification section below.

vi. *IsOptional* - Set this to True if the *SubAttr* under consideration is optional.

d. **WorkExperience**.
In this construct we specify the fields required from the user as part of work experience. The top level component *WorkExperience* is composed of one or more *SubAttr,* and two other fields namely "*MultipleEntries*" and

"*IsOptional*" . *MultipleEntries* is set to True if more than one entry is acceptable for *WorkExperience*. *IsOptional* is set to True if having *WorkExperience* is optional.
Following are the details which need to be specified for every *SubAttr*

i. *SubAttrName* - This will be used assign column name to the *SubAttr* in database.
ii. *SubAttrLabel* - Analogous to *PersonalAttrLabel* in *PersonalAttr.*
iii. *[IsTypeString / IsTypeInteger / IsTypeFloat / IsTypeDate / IsTypeBoolean]* - analogous to *Type field in PersonalAttr.*
iv. *SubAttrChoices* - The values specified here would be rendered to the user as options to choose from. The values should be specified as a list with elements separated by commas and each element inside a single quote. Otherwise specify as empty list.
v. *[StringConstraints / IntegerConstraints / FPConstraints / BooleanConstraints / DateConstraints]* - See the Constraint Specification section below.
vi. *IsOptional* - Set this to True if the *SubAttr* under consideration is optional.

e. **Attachments**.
In this construct we specify the attachments required from applicant. The top level component *Attachments* is composed of one or more *Attachment.*
Following are the details which need to be specified for every *SubAttr*. In the database corresponding to every attachment we store a pointer to the location of file in file system.

i. AttachmentName: - This will be used in the database to assign column name to column storing the pointer to location of this attachment in file system.
ii. AttachmentLabel - This is what would be displayed on UI/view.
iii. AttachmentType - Only file type supported as of now
iv. MultipleEntries - Set this to True if more than one file is acceptable for the *Attachment* under consideration.
v. Is optional - Set this to True if the uploading the *Attachment* under consideration is optional.

Constraint Specifications:

Constraints should be specified as a dictionary with every key-value pair separated from other key-value pair by comma.

E.g. An *IntegerConstraints* could be specified as
{ DBType: "INTEGER" , LTE: "2019" , GTE: "2018"}

DBType is a must constrain for the type for the whichever of *[IsTypeString / IsTypeInteger / IsTypeFloat / IsTypeDate / IsTypeBoolean]* is set to True.
E.g. If *IsTypeString* is set to True, then *StringConstraints* must have DBType specified (with other constraints if present). Constraints for other types in this case would be NA.
Note that the DBType specified will be used to specify type of the attribute in database.

Following are the supported constraints for the data types:

1. StringConstraints:
    a. Supported DBType { DBType: "VARCHAR(max_length)" }
    b. Supported Regex: All regexes E.g. { REGEX: “^hello”}.
    c. Supported Validations: { ValidationType: "Email" }

2. IntegerConstraints:
    a. Supported DBType { DBType: "INTEGER" }, { DBType: "SMALLINT" }
    b. Supported Comparisons:
        i. { GT: "XYZ” } - Greater than. E.g. { GTE: "2019" }
        ii. { GTE "XYZ” } - Greater than Equal. E.g. { GT: "2019" }
        iii. { LT: "XYZ” } - Less than. E.g. { LT: "2019" }
        iv. { LTE: "XYZ” } - Less than Equal. E.g. { LTE: "2019" }
        v. { EQ: "XYZ” } - Equal. E.g. { EQ: "2019" }

3. FPConstraints:
    a. Supported DBType { DBType: "FLOAT(precision)" }
    b. Supported Comparisons:
        i. { GT: "XYZ” } - Greater than. E.g. { GT: "2019.5" }
        ii. { LT: "XYZ” } - Less than. E.g. { LT: "2019.5" }

4. BooleanConstraints:
    a. Supported DBType { DBType: "BOOLEAN" }
    b. Supported Comparisons:
        i. { EQ: "True/False” }

5. DateConstraints:
    a. Supported DBType { DBType: "DATE" }
    b. Supported Comparisons:
        i. { GT: "yyyy-mm-dd” } - Greater than. E.g. { GT: "2019-01-01" }
        ii. { LT: "yyyy-mm-dd” } - Less than. E.g. { LT: "2019-01-01" }

2. ***Steps to up a new webapp.***

a. Add the portal name to installed apps in settings.py.
   Eg.
```
INSTALLED_APPS = [
      ...
      'portalBase',
      'phDAdmissionPortal',
      ‘yourApp’
]
```

b. Include path of urls.py in new app to portal base
   Eg.
```
path('yourApp/', include('yourApp.urls')),
```

c. Add the following to urls.py of both portal directory under urlpatterns
   Eg.
```
path('yourApp/',yourApp.AdmissionDetails,name= "yourApp"),
```

d. Add import and pattern both in urls.py in portalBase directory
   Eg.
```
from yourApp import views as yourAppViews
```
   and
   path('yourApp/', include('yourApp.urls'), name = "yourApp"),

e. Add button to PrimaryPage in portal Base and point it to yourApp
   Eg.
```
...
<button id ="ApplyIdYourApp" name="ApplyYourApp"
type="submit" formaction="{% url 'yourApp' %}" value=True>
Apply For Your Portal (Month - Year) </button>
```

f. There are ways to create new apps using “`startapp`” command. However we suggest to take a simpler approach to create new app than developing from scratch is to just use a copy of phdAdmissionPortal as follow the following steps.
   i. cd djangoProject/
   ii. cp -r phDAdmissionPortal/ yourApp/
   iii. cd yourApp/
   iv. Replace in apps.py
```
class PhDAmissionportalConfig(AppConfig):
    name = 'phDAdmissionPortal'
```

With

```
class YourAppConfig(AppConfig):
    name = 'yourApp'
```

v. Replace in views.py
Eg.
`phDAdmissionPortal.models with yourApp.models`

vi. Modify createform.cfg according to requirements
Imp : Note that no two createForm.cfg should have all the attributes in AdmissionDetails same.

vii. cd back to djangoProject

viii. Run the three commands
   1. `python3 manage.py makemigrations`
   2. `python3 manage.py migrate`
   3. `python3 manage.py runserver`

ix. Go to http://127.0.0.1:8000/

x. If everything has gone right, your service/app/portal should be up :)

3. ***Dependencies.***

a. Django version - 2.1.7
b. Backend database: Postgres version - 9.5.14
c. Psycopg2 - version 2.8

4. ***Settings***

a. Create database named ‘mydatabase’ in postgres
   Eg.

```
Create database mydatabase;
```

b. Add this to settings.py
   Eg.

```
DATABASES = {
      'default': {
      'ENGINE': 'django.db.backends.postgresql',
      'NAME': 'mydatabase',
      'USER': 'postgres',
      'PASSWORD': 'password',
      'PORT': '5432',
      }
}
```

c. You might also want to see this.

---END---