**Knowledge Booster**

A Django based web application where kids can take the quizzes to test their skills in the respective subjects and can improve their skills by going through the resources provided.

The Knowledge Booster project has a home page where a user can login if he is already a user or else he has to register in order to access the application. After logging in the user can explore the application. The starting page of the application contains three test links for three different subjects.

**Subjects:** Mental Ability, English, General Knowledge.

We have selected the 7th grade for selecting the questions for the application.

The user can click on the test link to start the quiz. We provide the questions and the options from the database. The questions are randomly selected from the database. The user is given access to only three tests at present. Between the tests there should be a 24 hours gap. Only then the test link will be activated. In the twenty four hours gap a user can go through the resources provided. The test link within this 24 hours gap is changed to resources link per the particular subject. The resources are also provided from the database.

The questions are divided into three categories - easy, medium and difficult.

In the three tests per subject the level of the difficulty increases. The score of the user per each test in a subject is considered and the visualizations are provided for the scores. The user can go to the tests page or to the resources page after the scores are displayed.

A django project is divided into the main project and the applications. The main project is Scorer. The apps are home, starttest, instructions, quiz, resources. The HTML files are stored under the templates folder in django. The CSS and Javascript files are called static files stored under the static folder.

**Creating Virtual Environment:**

Django is created in a virtual environment. Instead of installing the Django software in the system it’s installed in the virtual environment and the server is run in the virtual environment. Open command prompt. Navigate to the project folder. Enter the following command.

>pip install virtualenvwrapper-win

Making the virtual environment

>mkvirtualenv project

Your virtual environment is created.

**Installing Django:**

Django is installed into the virtual environment created using pip.

(project) >pip install django

(project) >django-admin --version

**Creating a Project:**

Create a directory called projects. Create a project using the following django command in the given virtual environment.

>mkdir projects

>cd projects

>django-admin startproject scorer  
>cd scorer

**Structure of the project in django:**

The project in django has init.py, settings.py, urls.py, wsgi.py python files. The urls.py is used to configure the urls of the other apps. The user, database and configurations of the project is done in settings.py. Wsgi.py is used in deploying the application on the server.

**Running the server:**The django framework provides a light-weighted server present in the manage.py file.

The server is run in the virtual environment.

>python manage.py runserver

**Starting an app in Django:**The application in django is started using the following command.

>python manage.py startapp home

Thus the url file is created in the app. The url.py in the app is configured with the urls.py in the main project.

Example of the url pattern:

urlpatterns=[

path(‘/home’ ,views.home, name=’home’)

]  
The url to open the above app is localhost:8000/home. The django server port number is 8000. Views contains the functions to be done on opening the given url. The name of the function is home.

**Configuring the template folder:**Django consists of the html files in the template folder. Thus the template folder has to be configured in settings.py using the following code in the project.

'DIRS': [os.path.join(BASE\_DIR,'templates')]

The above line has to be added to the templates.

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR,'templates')],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

**Configuring the static files:**

The static files are placed under the static folder in django. So the static folder has to be configured in settings.py using the following code in the project.

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'static')

]

STATIC\_ROOT = os.path.join(BASE\_DIR, 'assets')

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

**Configuring the media files:**

The media files are also static files that are placed under the static folder in django. So the media folder has to be configured in settings.py using the following code in the project.

MEDIA\_URL = "/media/"

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media\_cdn')

This code has to be added under the static folder.

In the urls.py the urls of the static and media folder has to be added.

if settings.DEBUG:

urlpatterns+=static(settings.STATIC\_URL,document\_root=settings.STATIC\_ROOT)

urlpatterns+=static(settings.STATIC\_URL,document\_root=settings.MEDIA\_ROOT)

**Collecting static files:**

The static files are collected into the assets folder so that they can be used in any of the html files in the project. The following command has to be used to collect the static files.

>python manage.py collectstatic

**Database:**The database used is the postgresql. The username and password of the database has to be configured in the settings.py file of the project. The following code is used.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'scorer',

'USER': 'postgres',

'PASSWORD': '\*\*\*\*\*',

'HOST': 'localhost'

}

}

**Making migrations:**

As the django uses the ORM model, the models created can be directly changed into a table. Thus in order to migrate the database to the table migrations have to be made.

>python manage.py makemigrations

>python manage.py migrate

Thus the model fields are converted into the table fields in the database on migrations.

**Creating Superuser:**Admin user is created in order to access the database. The admin user is called super user. The superuser is created using the following command.

>python manage.py createsuperuser

Enter the name and password.