Implementation Document

for

DigiSchool

Prepared by

Group #: 19 **Group Name: Dementors**

GUGULOTH VARUN	180272	varunguguloth@gmail.com
SUMIT NAGLE	170732	sumit.nagle@gmail.com
GAURAV KUMAR	180265	phogat102135@gmail.com
MOTA JITENDRA	180434	jitendratherider23@gmail.com
AKASH DAYANAND CHAVAN	180054	akashchavan180054@gmail.com
DIKSHANT RAJ MEENA	180247	dkrajmeena27899@gmail.com
SHREYASI PRASAD	190824	student.shreyasi@gmail.com
YASHWANT TAILOR	180894	yasht88101@gmail.com
YEDDULA VINAY CHANDU	191181	chandu8042439@gmail.com
AMAN RAW	190108	amanraw4444@gmail.com

Course: CS253

Mentor TA: Shatroopa Saxena

Date: 03-03-2022

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft Type and Number	Full Name	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00

1 Implementation Details

Programming Languages,
Python as backend handling of web application
HTML/CSS for frontend page generation and styling.
JavaScript for responsive frontend pages.
Databases,
sqlite3 just for development database, in future we will use high-end db systems or can even use cloud.
Frameworks,
Front-end: bootstrap, django's template system
Back-end: Django framework (following MVT Model)

We used Django because it is easier to pick up than other frameworks. Django is easy and quick to learn, which helped some of our member who were totally unknown to django in the beggining. Moreover, Django is famously known for,

- *Reusability*
- *Pluggability of components*
- *Less code*
- *Low coupling*
- *Rapid development*

Which suited us, for our web application needs.

Talking about sqlite3, SQLite is well-known for its portability, dependability, and high performance in low-memory situations. Even if the system fails or there is a power outage, its transactions are ACID-compliant.
SQLite is a "serverless" database. Most relational database engines are implemented as a server process, with programmes communicating with the host server via interprocess communication. SQLite, on the other hand, allows any process that uses the database to read and write directly to the database disc file. This makes it easier to set up SQLite because it eliminates the need to configure a server process.
Moreover, programs that use the SQLite database don't need any setting; all they need is access to the disc.

Bootstrap aims to create a centralized development code that not only connects designers and developers, but also ensures consistent outcomes across platforms. It is based on grid system, which makes it easy to containerise sections of pages. where each individual container have their own independent screen section. Which also make it compatible for device of any size, thus eliminate the need of configuring web pages for each device size.

Django template system, comes in built with django framework, it is easier to use, and makes it very easy to make dynamic pages, but only using simple pythonic syntax in the html files itself.

2 Codebase

In the root folder,
Templates *directory is the main directory which contains all the html and static files, which provides the frontend (or to say UI) to the user.*

back_end functions *directory is the repository of all the extra work which is happening in the backend, such as, doing the data analysis (such as student's performance), validations of user's request inputs and so on.*

manage.py *is a Django internal python file, which manages the django projects.*

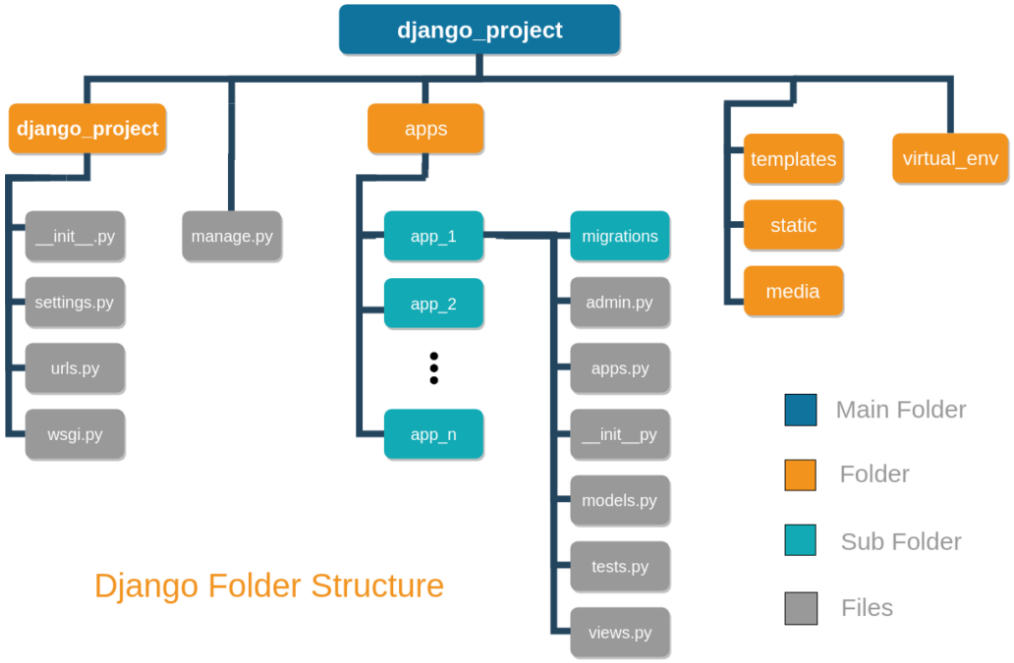
digischool *directory is the project's directory containing all the projects related things such as urlconf file (urls.py) , project's configurations (settings.py), web server configurations (wsgi.py asgi.py) and so on.*

In Django, we can define multiple applications, each doing their specific job where each app follows the MVT model. Along with that, they can inter-communicate with each other (via simple python import statements), in order to provide the whole desired functioning of our web-application.

Each app have some important python files,
admins.py file which makes the admin interacts with all the model/databases defined in that app.

views.py file, is the view functions files, which contains the logic of working that should happen when a certain web request comes, and what response should be sent.

models.py file, contains the django's ORM's way of defining table schema for databases involved in each app.



*And in our project, all the directories, ending with "**app**", is a Django app.*
***courseapp**: is responsible for handling all the courses being offered.*
***forumapp**: is responsible for handling discussion forums, across all the courses.*
***lectureapp** and **testapp**: are responsible for handling all the lectures and tests, across all the courses.*
***loginapp**: handling all the user login related things. Such as signup, login, contactus and the home page, as well as otp handling.*
***profileapp**: handles all the profile related things for all users, such as profile details and student's performance analysis.*

3 Completeness

Till Version 1 of this document:
***Login app** and **profile app** are fully functional. Allowing fully functional, signup of users, login of already signed up user as well as query in contact page. The home page (profile page) of all the users is fully visible and working. OTP handling is working fine.*

Backend work for validating user input is also done. This file is being utilised by all the apps which are taking inputs from the users.

Backend work for data analysis is in progress.

All the required databases (from all the apps) are fully structured and functional, and they are intercommunicating with each other successfully with no errors for now.

*backend for the **test app**, responsible for handling all the test related things, is done.*

Till Version 2 of this document:
All the app are now fully functioning, providing the features that we stated in SRS document and now we are progressing our way towards these future features.

Future features
We have decided to not include online timer based MCQ tests, because they are more prone to cheating. However as we progress, if we are capable of finding a way to overcome this issue, we may add this feature.

Currently we are storing the database on our server itself, however, as we expand, we will move our database to a cloud server, where through APIs, we will communicate with the data.

For now we are building simple data validation software from scratch in order to fulfill our small scale needs. But in future we will use libraries and APIs which will provide additional security.

Currently we are asking for lecture videos, thus reducing the available storage, and even our network bandwidth is high due to upload and download, so in future we will implement lectures uploading through youtube's API.

Appendix A - Group Log

<*Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to implement your software*>