A

Project Report

On

Alumni Management System

Submitted for partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

Ms. Medam Sai Sirisha	**16K81A05E7**
Mr. Kamatala Ashish	**16K81A05D5**
Mr. Siddharth Ramawat	**16K81A05G8**

UNDER THE GUIDANCE OF

Dr. Govinda Rajulu G.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

St. Martin's Engineering College

(Affiliated to JNTU, Hyderabad)

Dhulapally(V), Qutubullapur(M), Secunderabad

2019 – 2020

St. MARTIN'S ENGINEERING COLLEGE

An Autonomous Institute
Dhulapally, Secunderabad - 500 100
NAAC A+ Accredited

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the work embodies in this dissertation entitled '***Alumni Management System***' being submitted by **Ms. MEDAM SAI SIRISHA(16K81A05E7), Mr. KAMATALA ASHISH (16K81A05D5), Mr. SIDDHARTH RAMAWAT (16K81A0G8)** for partial fulfillment of the requirement for the award of '**Bachelor of Technology**' in **St. Martin's Engineering College, Dhulapally, Kompally, Secunderabad (T.S.)** during the academic year 2019-20 is a record of bonafide work, undertaken by him/her the supervision of the undersigned**.**

Supervisor

Dr. Govinda Rajulu G.

Head of the Department

Dr. P. Udayakumar

Principal

St. MARTIN'S ENGINEERING COLLEGE

An Autonomous Institute
Dhulapally, Secunderabad - 500 100
NAAC A+ Accredited

Department of Computer Science and Engineering

DECLARATION

We '**Ms. MEDAM SAI SIRISHA (16K81A05E7), Mr. KAMATALA ASHISH (16K81A05D5), Mr. SIDDHARTH RAMAWAT (16K81A05G8)**', are students of '**Bachelor of Technology'**, session: 2019 - 20**, St. Martin's Engineering college, Dhulapally, Kompally, Secunderabad, Telangana State**, hereby declare that the work presented in this Project Work entitled '**Alumni Management System'** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Ms. Medam Sai Sirisha	**16K81A05E7**
Mr. Kamatala Ashish	**16K81A05D5**
Mr. Siddharth Ramawat	**16K81A05G8**

Date:21.05.2020

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowded our efforts with success.

We extend our deep sense of gratitude to Principal, **Dr. P SANTOSH KUMAR PATRA**, St. Martin's Engineering College Dhulapally, for permitting us to undertake this project. We are also thankful to **Dr. P UDAYAKUMAR**, Head of the Department, Computer Science and Engineering, for his support and guidance throughout our project as well as Project Coordinator **Dr. GOVINDA RAJULU G.**, Professor, Computer Science and Engineering Department for his valuable support.

We would like to express our sincere gratitude and indebtedness to our project supervisor **Dr. GOVINDA RAJULU G.**, Professor, Computer Science and Engineering, St. Martin's Engineering College, Dhulapally, for his support and guidance throughout our project.

Finally, we express thanks to all those who have helped us successfully completing this project. Furthermore, we would like to thank our family and friends for their moral support and encouragement. We express thanks to all those who have helped us in successfully completing the project.

Ms. Medam Sai Sirisha	**16K81A05E7**
Mr. Kamatala Ashish	**16K81A05D5**
Mr. Siddharth Ramawat	**16K81A05G8**

ABSTRACT

Alumni management is one of the thrust areas considered focal to institutional development mostly in developing countries.

A strong alumni system plays an important role in reaping anonymous benefits for student-student networks as well as institution-student networks.

A major problem with some of the existing systems is the details of any student is exposed to anyone on the web. There is no Alumni Management System for most of the colleges in "Telangana State"(Tier - 3).

Our system proposes an easy and interactive management portal for creating networks among students as well as institutes. The portal allows currently enrolled students as well to create networks with graduates of the organization. The system validates the students enrolled to the organization based on their Registration Number collected from the Institute/Organization.

Our system makes it easier for users to register into the system, connect with the alumni of organization, easy creation of events like postings on jobs, events happening around the city etc. The users will be notified about the new events and the users can register for a particular event from their feed.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND DEFINITIONS

	ACRONYM	DEFINITION
01	API	Application Programming Interface
02	AVD	Android Virtual Device
03	CSS	Cascading Style Sheets
04	DB	Database
05	HTML	Hyper Text Markup Language
06	HTTP	Hyper Text Transfer Protocol
07	JSON	Java script Object Notation
09	MVT	Model View Template
09	PWA	Progressive Web Apps
10	SDK	Software Development Kit
11	SQL	Structured Query Language
12	UML	Unified Modelling Language
13	URL	Uniform Resource Locator

LIST OF TABLES

CHAPTER 1 - INTRODUCTION

1.1INTRODUCTION ABOUT DOMAIN

1.1.1Web Programming

Web programming, also known as web development, is the creation of dynamic web applications. Web development is the work involved in developing a website for the Internet (World Wide Web) or an intranet (a private network).Web development can range from developing a simple single static page of plain text to complex web-based internet applications (web apps), electronic businesses, and social network services. It includes aspects such as web design, web publishing, web programming, and database management.

Examples of web applications are social networking sites like Facebook or e-commerce sites like Amazon.

In fact, many argue it's the best form of coding for beginners to learn. It's easy to set up, you get instant results and there are a lot of online resources, videos and documentation of various technologies available.

People today are learning web development because they want to create a new startup or find a job in the industry. It's super easy to get started hence the best choice to begin development with. No matter whether you're looking for a career or just want to learn coding, learning how to develop for the web is for you. It's one of the smartest decisions you will ever make.

The two broad categories into which web development is divided into is front-end development (sometimes called as client side development) and back-end development (sometimes called as server side development).

Front end Development

Front end development mainly refers to the construction of web pages that are loaded when the user starts a web application, basically the User Interface, and how the web page responds and handles the events which are fired by the user by interacting with components on the webpage.

All HTML documents must start with a document type declaration: <!DOCTYPE html>.The HTML document itself begins with <html> and ends with </html>.The visible part of the HTML document is between <body> and </body>.

Javascript is used to define the behavior of the page. It is also used to make the page responsive. Javascript lets the users to enable dynamic loading of the content to be displayed on the webpage apart from static loading.

Responsive Web Design is used in all types of modern web development.

ECMAScript 5 (JavaScript 5) is supported in all modern browsers.

Popular Javascript libraries used these days are React.Js, React Native, Angular.Js, Angular 4,Vue.Js,W3.Js .

The frameworks used for creation of the responsive webpages are BootStrap , Material Design, W3.CSS.

Back end Development

Back-end development controls what goes on behind the scenes of a web application. A back-end often uses a database to generate the front-end.

Back-end scripts are written in many different coding languages and frameworks, such as:

1. PHP
2. Ruby on Rails
3. ASP.NET
4. Perl
5. Java
6. Node.js
7. Python

Back end developers are most focused on a site's responsiveness and speed. These languages are used to create dynamic sites which are different from static sites in that these types of websites store database information. Content on the site is constantly changing and updating. Examples of dynamic sites include Facebook, Twitter, and Google Maps.

Backend development languages handle the 'behind-the-scenes' functionality of web applications. It's code that connects the web to a database, manages user connections, and powers the web application itself. Backend development works in tandem with the front end to deliver the final product to the end user.

Backend programming can either be Object Oriented (OOP) or Functional.

The former is the technique that focuses on the creation of objects. With object-oriented programming, statements should be executed in a particular order. Popular OOP languages are Java, .NET, and Python. The latter is a technique that is more "action"-based. Functional programming uses declarative language, which means that statements can be executed in any order. It's commonly used for data science, and popular languages are SQL, F#, and R.

Languages can either be statically typed or dynamically typed. The former is more rigid, but better at catching errors, whereas the latter is more flexible but allows for variables to change types (which could account for unexpected errors).

Back end web infrastructure consists of specific things like:

1. Databases—data storage applications that provide websites with dynamic content updates (e.g. when you check your bank account balance from a bank website, the site accesses your account information from a database, causing your balance to update on your screen).
2. Server scripts—scripts are sets of instructions written in code that tell computer programs to "do something." Back end or "server side" scripts allow a website's servers (web hosting hardware where a site's images, videos, and other assets are stored) to respond to actions and commands from the front end (e.g. retrieving an image or a video from a server and displaying it on a user's screen).
3. APIs—APIs (dev speak for Application Programming Interface) are sets of routines, protocols, and tools that allow applications to communicate with each other. When you

share an article you just read to Facebook or Twitter with the click of a “share” button, it’s an API that allows that cross platform sharing to happen.

Back end developers build and maintain these server-side applications and tools, and in the process add a whole lot of function and utility to what users see on their computer, phone, or tablet screen. The below figure 1.1 shows an overview of web development with the way of communication between backend and frontend technologies.

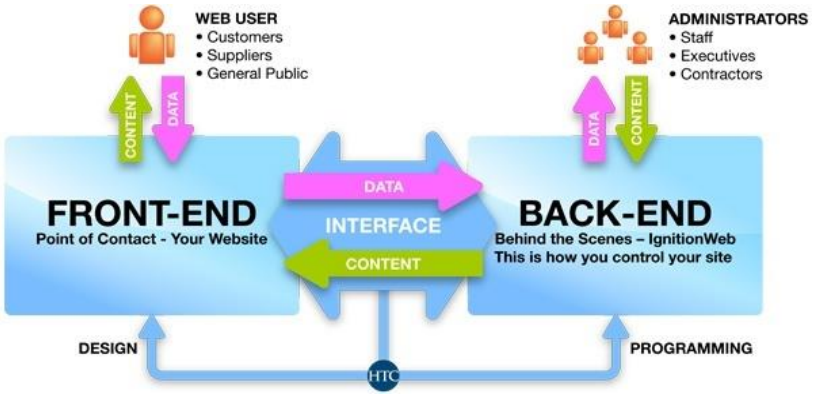

Fig 1.1Overview of Web Development

1.1.2 Android Development

Android software development is the process by which new applications are created for devices running the Android operating system. Google states that "Android apps can be written using Kotlin, Java, and C++ languages" using the Android software development kit (SDK), while using other languages is also possible. All non-JVM languages, such as Go, JavaScript, C, C++ or assembly, need the help of JVM language code, that may be supplied by tools, likely with restricted API support.

Mobile app development is the creation of software intended to run on mobile devices and optimized to take advantage of those product's unique features and hardware. Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code,

binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

There are two dominant platforms in the modern smartphone market. One is the iOS platform from Apple Inc. The iOS platform is the operating system that powers Apple's popular line of iPhone smartphones. The second is Android from Google. The Android operating system is used not only by Google devices but also by many other OEMs to built their own smartphones and other smart devices.

The types of mobile apps that developers create include native apps, hybrid apps and HTML5 apps.

The creation of mobile applications draws much of its roots from traditional software development. The end result, however, is software intended to utilize the unique features and hardware of mobile devices. Modern smartphones are equipped with Bluetooth, NFC, gyroscopic sensors, GPS, cameras and so much more. They can be used for virtual or augmented reality, barcode scanning and more. Mobile apps should utilize the full range of smartphone features, which is easier said than done.

With desktop PC software development, programmers must create an application that can operate on a minimum set of hardware. The same goes for mobile applications, though the hardware variances in this instance are much more minimal. At the same time, the hardware in smartphones and tablets doesn't quite match that in laptops and desktop computers, which means mobile apps must be designed to show optimal performance.

For example, a gaming app would be limited in its graphical elements because of the less-powerful graphics processors in mobile devices. With that said, cloud computing is making it easier than ever to accomplish mobile gaming. Popular games such as Fortnite , Hearthstone, and PUBG connect players across computers, phones and even consoles.

Mobile App Development Considerations:

Solving the issue of performance on any given device is ultimately dependent on developing an app natively on that device. This means designing the code specifically for the hardware on a particular device. In the instance of iOS devices, this proves quite easy, as mobile developers only need versions of the app for the iPhone and iPad to achieve universal usability. For Android devices, however, each smartphone and tablet runs on different hardware and varying versions of the operating system.

Web-based apps, on the other hand, don't depend on the device; they run off of a web browser, making them cheaper to develop and easier to access. The problem with web apps, however, is that their performance doesn't compare to that of a native app. For example, with web apps, you cannot use the phone's full features or send proper notifications, and they look less professional.

Types of Mobile Apps and Programming Languages:

Like desktop software, mobile apps are designed using a wide range of programming languages and frameworks. While the most popular operating systems, iOS and Android, have done an excellent job of standardizing the types of mobile app development available for programmers, apps can still vary. Here are some mobile app types:

1. Native apps. These are apps created for a specific platform (iOS or Android) using the software development tools and languages supported by those operating systems. iOS uses Xcode and Objective-C, whereas Android uses Eclipse and Java. Developers often prefer native apps because of their ability to utilize a device's full potential. With smart home devices becoming more ubiquitous, developers are creating unique applications that integrate things like wearables, IoT sensors and smart screens for personalized experiences. Of course, development for each platform is easier said than done and is a costly and time-consuming process that doesn't work for all businesses.
2. HTML5 apps. Based on the near-universal standards of web technologies – namely, HTML5, JavaScript and CSS – this type of mobile app takes a write-once-run-anywhere approach to mobile development. Apps developed in this framework are compatible with many platforms and require only minimal changes to ensure complete functionality in each

operating system. HTML5 apps can still send desktop notifications and trigger interactions through email and other avenues. Don't discount web apps' usability, but keep in mind that users are more likely to use a native app. A study from Oracle found that millennials spend 90% of their mobile time in apps, compared with 10% in web browsers.

3. Hybrid apps. These apps entail the creation of a container developed in the native system that makes it possible to embed an HTML5 app within it. This allows apps to make use of the diverse and unique elements of each native system. Before creating your own branded app, consider instead utilizing already existing apps for greater impact. For example, by using mobile-focused marketing on services such as Yelp, Facebook and Google Maps, you can drive traffic to both your website and brick-and-mortar location.
4. Cross-Platform Applications: Cross-platform native mobile applications can be written in variety of different programming languages and frameworks, but they are compiled into a native application running directly on the operating system of the device.
5. Hybrid-Web Applications: Hybrid mobile applications are built with standard web technologies - such as JavaScript, CSS, and HTML5 - and they are bundled as app installation packages. Contrary to the native apps, hybrid apps work on a 'web container' which provides a browser runtime and a bridge for native device APIs via Apache Cordova.
6. Progressive Web Applications: PWAs offer an alternative approach to traditional mobile app development by skipping app store delivery and app installations. PWAs are web applications that utilize a set of browser capabilities - such as working offline, running a background process, and adding a link to the device home screen - to provide an 'app like' user experience.

Software Development Kits:

Mobile app development requires access to software development kits (SDKs) that provide an environment through which programmers can design and test code in a simulated mobile environment. However, creating an app does not require full use of these kits. For example, developers can create mobile games using Unity and then use the Android SDK to ensure its deliverability on mobile devices. Developing apps for iOS requires a paid iOS developer license, whereas the Android SDK is freely available to users.

iOS (47%) and Android (52%) have similar mobile market shares, but developing for Apple is somewhat easier in that you don't need to worry about a wide range of devices from different manufacturers. Regardless of which operating system you choose, however, there are barriers to entry.

Mobile Application Development Services:

Mobile application development is changing constantly. Typically, every six months or so, a new version of an operating system rolls out with unique features that mobile apps can utilize. Developing for a specific version of the operating system, or even for a native operating system, usually requires developers to try multiple solutions to find the one that suits their development needs.

1.2 OBJECTIVE:

Our system proposes an easy and interactive management portal for creating networks among students as well as institutes. The portal allows currently enrolled students as well to create networks with graduates of the organization. The system validates the students enrolled to the organization based on their Registration Number collected from the Institute/Organization.

Our system makes it easier for users to register into the system, connect with the alumni of organization, easy creation of events like postings on jobs, events happening around the city etc. The users will be notified about the new events and the users can register for a particular event from their feed.

CHAPTER 2 - LITERATURE SURVEY& PROBLEM INDENTIFICATION

2.1 LITERATURE REVIEW

Alumni are the living examples and testimonials of any organization. It's because of the strong alumni network that leads to the recognition and fame of the college.

A constant and active involvement of the alumni with the institute proves to be very useful for the college as well as students.

Alumni Management System is used to connect with the students who have passed out of the college(ex-students).

An Online Alumni Tracking System is an example of web application which is under the information systems. It helps an academic institution in tracking its alumni. Also, it helps the alumni to communicate with the institution through the use of the internet. It also helps the alumni to get updated with the latest news and upcoming events of the institution. This application can easily be accessed through the use of the internet which will be very useful to the alumni because they can keep in touch with the institution even if they do not visit the school. This application can be very useful especially to those alumni who are now living abroad because they can still get connected with their fellowmen and the institution. This application is also useful because it can make transactions and process paperless.

Nowadays, computers have infiltrated all the aspects of our society. The computer is most likely one of the great technological mechanism for future change. It can now simply make our works easier and lighter. With this great thing it won't be more useful without the computer's software. Software is a generic term for organized collections of computer data and instructions, often broken into two major categories: system software that provides the basic non-task-specific functions of the computer, and application software which is used by users to accomplish specific tasks.

2.2 RELATED WORK

[2] The paper "Centralized Alumni System- A Prototype Proposal" proposes a system which is institution dependent. There are systems developed for students studying in any educational institute who are above the age of 15.

[1] “Online Alumni System “proposes a system which allows anyone on the web to access and search information about any student.

“Alumni Network Analysis“ is a system that evaluates educational institutes and ranks them based on the data collected about the network connections of alumni in the entrepreneurial and technological domain.

Features of an Alumni Information System

According to Webaloo (2007) an alumni builders systems features includes the following: Alumni class page- a directory of alumni names by class year with links to individual profiles and email addresses; Alumni Profile – is controlled by individual users and displays only the information that he/she wants to display; Alumni Search – allows the users to search by name, class, occupation, address, etc.; Alumni Forum – offers alumni a way to stay in touch with classmates and friends form other graduation years; Alumni Notes – allows classmates to communicate by posting on a notes page; Secure Log-in – use to block sensitive alumni information from other school constituents; Profile Change Report – allows the school to keep track of the personal information that alumni can update; and Missing Alumni Page – helps the school reconnect with graduates whose personal information is outdated.

Alumni Portal

According to Goodwin College (2012) the use of Alumni Portal will keep the alumni administrators updated and to keep in the know! The Alumni Portal has been activated so that you can: Update your contact information in order to receive important communication and invitations to events and programs from the Alumni Association; Notify us of a new job or job change, family additions or other news; Access College Central Network to search for jobs, post your resume, access informative career related documents, videos, podcasts and much more; Browse the events calendar for upcoming happenings; Search for classmates, students and faculty and staff; View course and grade history; and access your 1098-T.

Web-based Alumni Information System of West Negros University

The web-based Alumni Information System of West Negros University (2010) is capable of gathering information of all the alumni using the web application form. The system helps the administrator, alumni personnel and even the public relation office in maintaining the data and can easily send and display information for all concerns. The system is internet-based that can access all online portals and connect to all networks that enable to collect all the data as needed by the university particularly by the alumni office. The system is also a social network that provides space for chatting, forum blog and photo gallery functions.

On Student Information

In the study conducted by Agudera and Mendiola (2013) the implementation of Alumni Information System, the system will be secured, the process in having the student information will become faster, and accurate generating reports. The system will secure student files using the log-in log-out form for the unauthorized users. The developers design the interface so the users will understand the system easily and use the Microsoft Access for database to store important information. The system will help a school to be well-organized.

UMT Alumni Information System

The role of information system can't be ignored doing things faster, doing things better, and doing thinks smarter these all traits are possible just because of two words, Information system. Alumni information system is one of the examples of information system. To get contact with the old students and to provide the assistance to this old student for their future progress in all field of life and maintain the record of the students. Following core aims and objective can describe the real need of the AIS. The aims and objectives of UMT Alumni information system are to encourage alumni to maintain links with the University and with each other, in order: to promote more effectively the welfare and interests of the University and its alumni; to support the University's aims and objectives and uphold its reputation as an ambassadors of the University; to establish mutually beneficial relationship between the University and its alumni and to bind the alumni more closely together; to assist in developing financial and other resources for the University and the

Alumni Association; to develop linkages for mutual benefit (such as research) with other professional alumni bodies, and to remain the part of the university even after the study.

2.3 ISSUES IN EXISTING SYSTEM

1. The existing study stated shows that alumni information system contributes to a good relationship between them and the school.

2. The existing studies designs, and implementations get involved the web as well as the Online Alumni Information System however our system informs only the alumni of the college and enable them to interact with each other. It also provides career service opportunities to all the graduates of the college.

3. Existing system is a manual one in which users are maintaining documents paper work to store the information like colleges details, student details. It is very difficult to maintain historical data.

4. It is difficult to maintain important information in documents paper work. More manual hours need to generate required reports.

5. It is tedious to manage historical data which needs much space to keep all the previous years, ledgers, documents paper work.

CHAPTER 3- METHODOLOGY

3.1 MODEL VIEW TEMPLATE ARCHITECTURE

The MVT (Model View Template) is a software design pattern. It is a collection of three important components Model View and Template. The Model helps to handle database. It is a data access layer which handles the data.

The Template is a presentation layer which handles User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and renders a template.

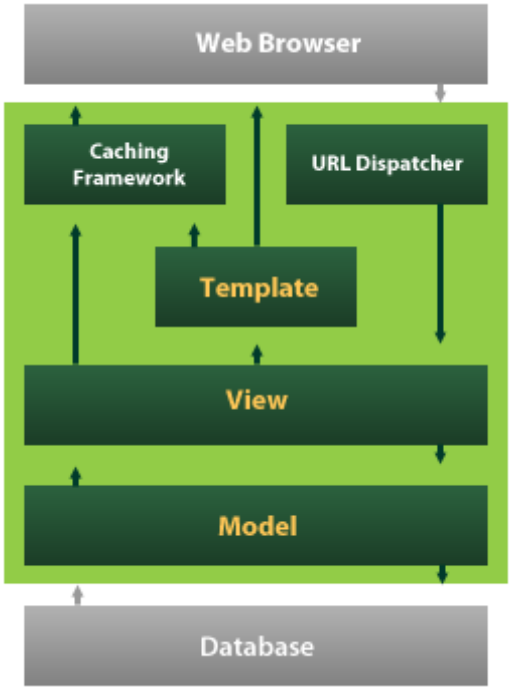

Fig 3.1 Model View Template Architecture

The above figure 3.1 shows the architecture of the Model View Template in a web browser.

3.2 DJANGO FRAMEWORK

Django is a Python's framework that enables developers to develop web applications.

It allows the developers to develop the apps , while Django does the setup of the project (major boiler plate code).The major features of Django framework which makes it the best choice for the development is:

1. Ridiculously fast: Developers can design the application and complete writing the code in no time.
2. Reassuringly secure: It helps developers avoid making common mistakes.
3. Highly scalable : Django provides flexibility to scale the users efficiently.
4. Open source : Django is open source and its documentation can be referred from the link :- https://docs.djangoproject.com/en/3.0/

Although Django follows MVC pattern but maintains it's own conventions. So, control is handled by the framework itself.

There is no separate controller and complete application is based on Model View and Template. That's why it is called MVT application.

The following figure 3.2 shows the MVT based control flow.

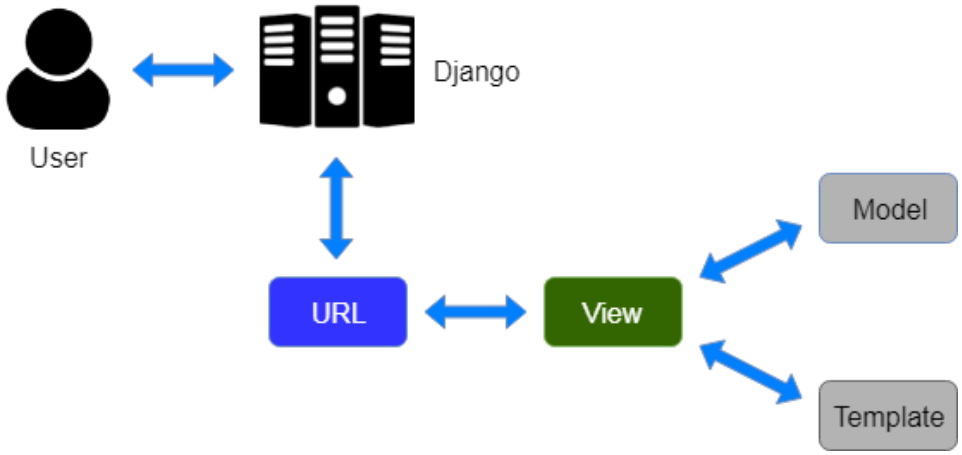

Fig 3.2 Control Flow in Django

Here, a user requests for a resource to the Django, Django works as a controller and check to the available resource in URL.

If URL maps, a view is called that interact with model and template, it renders a template.

Django responds back to the user and sends a template as a response.

3.2.1 Prerequisites to install Django:

To install Django 2.0 versions or above, make sure you have Python 3.0 or above installed into your system.

You can install Django using the command:

python -m pip install Django

3.2.2 Steps to create and run the project:

1. To create a new project in Django use the command:

   django-admin startproject <project_name>
2. Change the path to your project directory.
3. Then create a new app in your project use the command:

   python manage.py startapp <app_name>
4. To run the project

   python manage.py runserver
5. Configure the settings.py file in your folder in order to add database credentials.
6. In order to create database tables, edit the models.py file of the app.
7. Make sure you add your app's configuration into the INSTALLED_APPS list of settings.py
8. To add the new changes to the database , use the command

   python manage.py makemigrations
9. To make those changes permanent to the database, use the command

   python manage.py migrate
10. To register your models, use the following steps in admin.py of the app
    a. First import your models into admin.py
    b. Add this line : **admin.site.register(<model_name>)**

3.2.3 Overview of Django:

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed

based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files. The figure 3.3 shows the workflow among different files in Django.

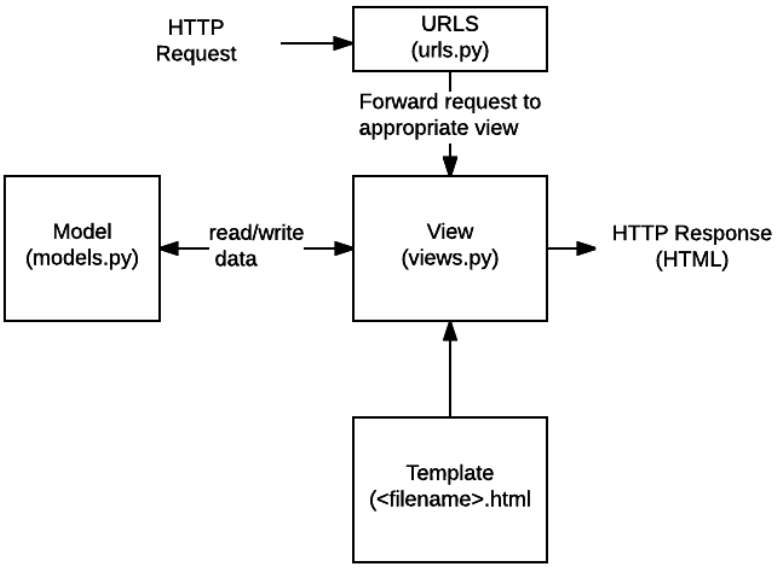

Fig 3.3 Django Workflow

- URLs: While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
- View: A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.

- Models: Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- Templates: A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

Just a few of the other things provided by Django include:

- Forms: HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
- User authentication and permissions: Django includes a robust user authentication and permission system that has been built with security in mind.
- Caching: Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
- Administration site: The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
- Serialising data: Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service (a website that purely serves data to be consumed by other applications or sites, and doesn't display anything itself), or when creating a website in which the client-side code handles all the rendering of data.

Urls.py

A URL mapper is typically stored in a file named urls.py. In the example below, the mapper (urlpatterns) defines a list of mappings between routes (specific URL patterns) and corresponding view functions. If an HTTP Request is received that has a URL matching a specified pattern then the associated view function will be called and passed the request.

```
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

The urlpatterns object is a list of path() and/or re_path() functions.

The first argument to both methods is a route (pattern) that will be matched. The path() method uses angle brackets to define parts of a URL that will be captured and passed through to the view function as named arguments. The re_path() function uses a flexible pattern matching approach known as a regular expression.

The second argument is another function that will be called when the pattern is matched.

Models.py :

Django web applications manage and query data through Python objects referred to as models. Models define the structure of stored data, including the field types and possibly also their maximum size, default values, selection list options, help text for documentation, label text for forms, etc. The definition of the model is independent of the underlying database — you can choose one of several as part of your project settings. Once you've chosen what database you want to use, you don't need to talk to it directly at all — you just write your model structure and other code, and Django handles all the dirty work of communicating with the database for you.

Views.py:

The Django model provides a simple query API for searching the database. This can match against a number of fields at a time using different criteria (e.g. exact, case-insensitive, greater than, etc.), and can support complex statements (for example, you can specify a search on U11 teams that have a team name that starts with "Fr" or ends with "al").

HTML Templates:

Template systems allow you to specify the structure of an output document, using placeholders for data that will be filled in when a page is generated. Templates are often used to create HTML, but can also create other types of document. Django supports both its native templating system and another popular Python library called Jinja2 out of the box (it can also be made to support other systems if needed).

3.2.4 Django Working Process:

The simplest way to look at Django is to break it down into its component parts. First off, there's a models.py file which defines your data model by extrapolating your single lines of code into full database tables and adding a pre-built (totally optional) administration section to manage content.

The next element is the urls.py file which uses regular expressions to capture URL patterns for processing.

The actual processing happens in your views which, if you haven't seen the pattern yet, live in views.py. This is really the meat of Django, since views are where you grab the data you're presenting to the visitor.

Here's what happens when a visitor lands on your Django page:

1. First, Django consults the various URL patterns you've created and uses the information to retrieve a view.
2. The view then processes the request, querying your database if necessary.
3. The view passes the requested information on to your template.
4. The template then renders the data in a layout you've created and displays the page.

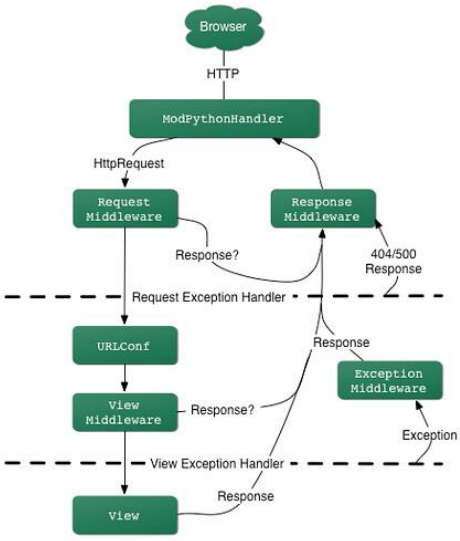

Fig 3.4 Django HTTP request handling mechanism

The figure 3.4 depicts the HTTP request - response handling mechanism to resolve the requests.

3.3 ANDROID

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets.

Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008. The current stable version is Android 10, released on September 3, 2019. The core Android source code is known as Android Open Source Project (AOSP), which is primarily licensed under the Apache License. This has allowed variants of Android to be developed on a range of other electronics, such as game consoles, digital cameras, PCs and others, each with a specialized user interface. Some well known derivatives include Android TV for televisions and Wear OS for wearables, both developed by Google.

3.3.1 Android SDK

The Android Software Development Kit (SDK) is a crucial part of Android development for beginners to come to grips with. It's a selection of files bundled together that you will need to begin creating Android apps. It consists of tools like the virtual device manager (emulator) and ADB bridge, as well as a library of additional code for making Java programs work with the Android platform.

3.3.2 Install Android SDK

The SDK is now included with Android Studio. Android development for beginners is getting easier and easier and this relatively recent change means you now only need to go through a single installation to get your development environment up and running. There's even an open Java Development Kit (JDK) included, so you no longer need to separately install the latest version separately. Setting up Android development for beginners has become a relatively streamlined process. It involves a few rather large files

3.3.3 Android WebView

WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView makes turns your application to a web application.

The figure 3.5 depicts the web view structure of Android.

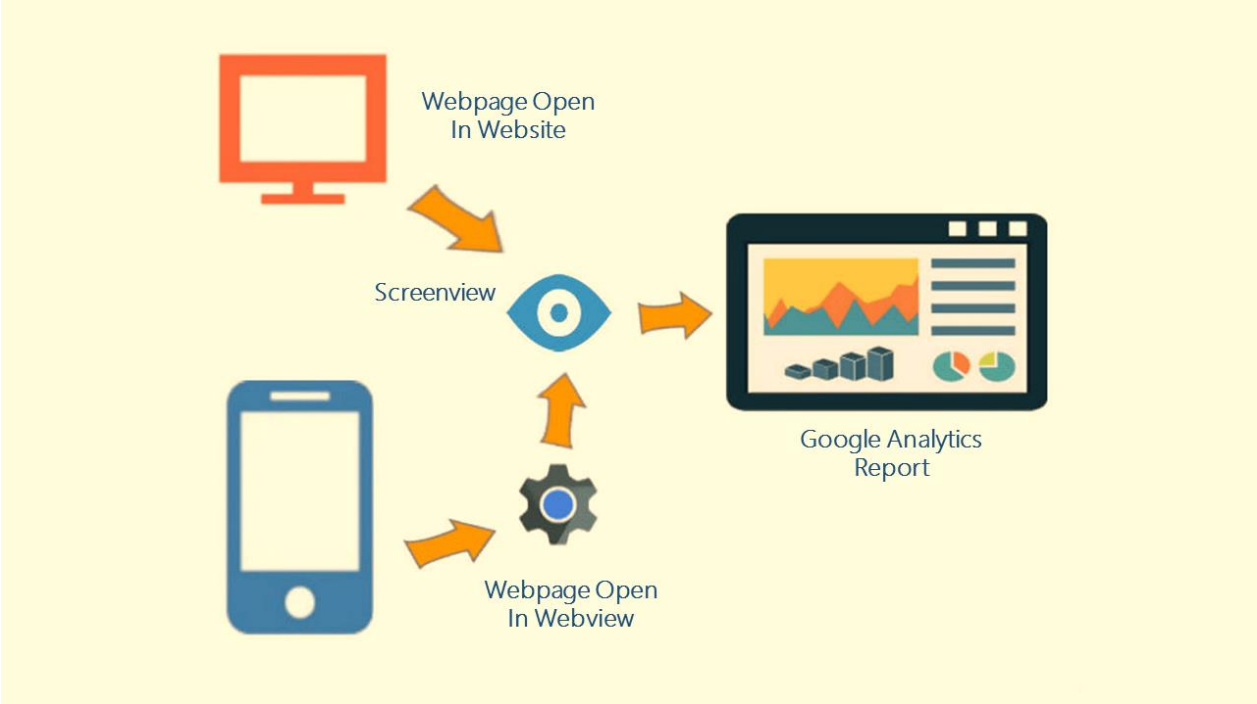

Fig 3.5 WebView Structure

3.3.4 Steps to run an application

Set up your device as follows:

1. Connect your device to your development machine with a USB cable. If you developed on Windows, you might need to install the appropriate USB driver for your device.
2. Perform the following steps to enable USB debugging in the Developer options window:
    a. Open the Settings app.
    b. If your device uses Android v8.0 or higher, select System. Otherwise, proceed to the next step.
    c. Scroll to the bottom and select About phone.

d. Scroll to the bottom and tap Build number seven times.
e. Return to the previous screen, scroll to the bottom, and tap Developer options.
f. In the Developer options window, scroll down to find and enable USB debugging.

Run the app on your device as follows:

1. In Android Studio, select your app from the run/debug configurations drop-down menu in the toolbar.
2. In the toolbar, select the device that you want to run your app on from the target device drop-down menu.

Run the app on an emulator as follows:

1. In Android Studio, create an Android Virtual Device (AVD) that the emulator can use to install and run your app.
2. In the toolbar, select your app from the run/debug configurations drop-down menu.
3. From the target device drop-down menu, select the AVD that you want to run your app on.
4. Click Run.

Android Studio installs the app on the AVD and starts the emulator. You now see "Hello, World!" displayed in the app.

CHAPTER 4-IMPLEMENTATION

4.1 PROPOSED SYSTEM

The goal of our application is to help university students to create networks and interact with alumnus of the university. The application lets the users to connect with alumnus by searching for them using name registration numbers.

The basic function of the application is to provide registered and authorized users an easy and interactive way to create events, post feedbacks about the curriculum/infrastructure etc., create and edit blogs, view feed and most importantly get referrals and create network with the alumnus. [3] The registered users can also participate in the events that are created by other users. The users get notified about the same.

4.1.1 Components:

1. Users component: This component is used to login or sign up to the system. The new users are authorized by the admin, once they are authorized they can access the system by logging in. The user's table is an extension to the default auth_user table provided by Django. The password is stored using SHA256 algorithm as a hash hence it is secure. After successful login the user is directed to dashboard where he can view all the recent blogs, posts, events posted by other users. The password is stored using different hashing algorithm by default to secure the user passwords.
2. Events component: The events can be created by any authorized user. Events such as technical events, recruitment events, fests, seminars, lectures etc. can be created. The users can view the percentage of available slots for a particular event. They can register for a particular event. Once they register for an event, they get notified about any updates from the event organizer about the event. Mailgun is being used to send mails to the registered users for a particular event. The users can contact the organizer for the event information on mail. All the past events that have been completed can be deleted by the admin or authorized user. The users can update the events after creation of the events.
3. Feedback component: The registered users can post feedback about curriculum, infrastructure, or general feedback. [9] The feedback received by all users can only be

viewed by the admin of the website, he can clear either entire feedback or can choose a particular date, the feedback posted till that date would be completely cleared. [3]
The admin can filter feedback based on the criteria.

4. Blog Component: The registered users can create blogs or post where they share technical knowledge or their interview experiences with other registered users on the system. The post can be deleted or updated the author of the post.

4.1.2 Other functionalities:

1. **Search:** The users can search other users by typing their name or registration number or department or job or company or location. The search results can be further filtered again based on users' requirement. The user can check the profile of other user that appeared in the search result.
2. **Profile:** Once the user has created the account, user can update their profile by visiting the profile section where they can update details like email id, image, job role, working location and company in which they are currently working.
3. **Blog:** Users can like or dislike posts on the feed.
4. **Poll Percentage:** Users can view the number of people registered for the event.
5. Authorized users can check the profile of the person giving feedback.
6. Users can check the profile of the organizer of the event.
7. The users can change their old passwords, there's a default PasswordChangeForm provided by default in Django that is being used to update the user's password. As soon as the password is updated the user will be logged out of current session.
8. Apart from these functionalities, there are various security filters and form validators available curbing the access to authorized pages and submission of forms respectively.

4.2 REQUIREMENTS

4.2.1 Hardware Requirements

RAM : 512 MB

Operating System : Windows/MAC/Linux

4.2.2 Software Development Tools

Frontend Framework : ReactJS

Backend Framework : Django

Database : PostgreSQL

Implementation Language : Python

IDE's: PyCharm Community , PgAdmin

Browsers : Chrome, Safari, Explorer

Android Versions : Above 8

Additional Requirements : Django version 2.5.4 or above ,PostgreSQL 9.5 or above, Python 3.x.y

4.3 UML DIAGRAMS

The Unified Modeling Language (UML) was created to forge a common, semantically and syntactically rich visual modeling language for the architecture, design, and implementation of complex software systems both structurally and behaviorally. UML has applications beyond software development, such as process flow in manufacturing.

It is analogous to the blueprints used in other fields, and consists of different types of diagrams. In the aggregate, UML diagrams describe the boundary, structure, and the behavior of the system and the objects within it.

The UML is popular among programmers, but isn't generally used by database developers. One reason is simply that the UML creators did not focus on databases. Despite this, the UML is effective for high-level conceptual data modeling, and it can be used in different types of UML diagrams.

Use Case Diagram

The figure 4.1 is the use case diagram for an authorized user and a normal user. The system can be used by authorized and authenticated users to perform various functions like search, insert, update ,delete etc.

Fig 4.1 Use Case Diagram

Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling.

The figure 4.2 below shows the class diagram of the system.

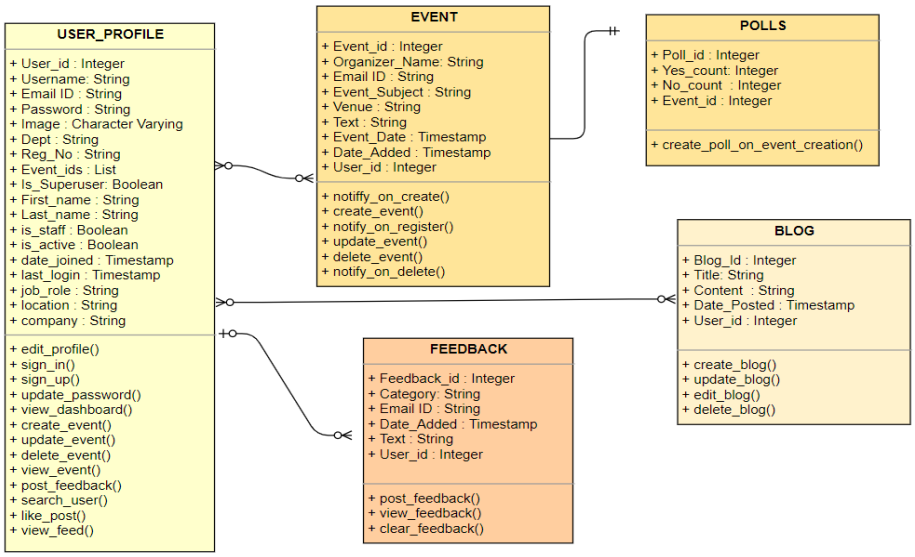

Fig 4.2 Class Diagram

Flow Diagrams

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. Flowcharts are used in designing and documenting simple processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help understand a process, and perhaps also find less-obvious features within the process, like flaws and bottlenecks. There are different types of flowcharts: each type has its own set of boxes and notations. The two most common types of boxes in a flowchart are:

- a processing step, usually called *activity*, and denoted as a rectangular box.
- a decision, usually denoted as a diamond.

The below figures show the flow diagram for various operations in the system for example figure 4.3 shows the create events, 4.4 view events etc.

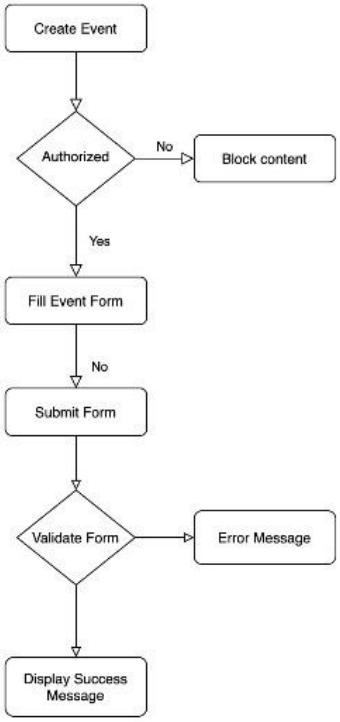

Fig 4.3 Create Events

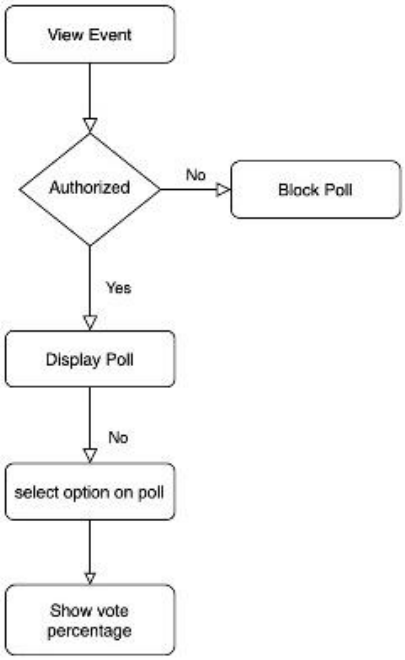

Fig 4.4 View Events

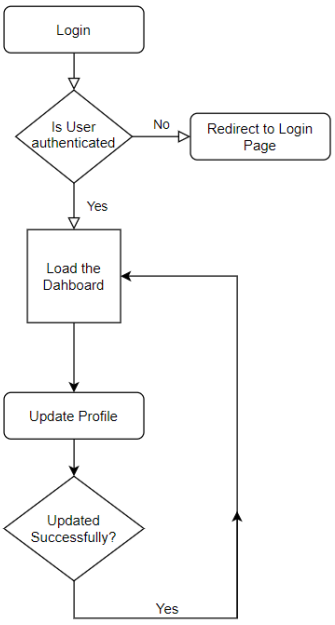

Fig 4.5 Update Profile

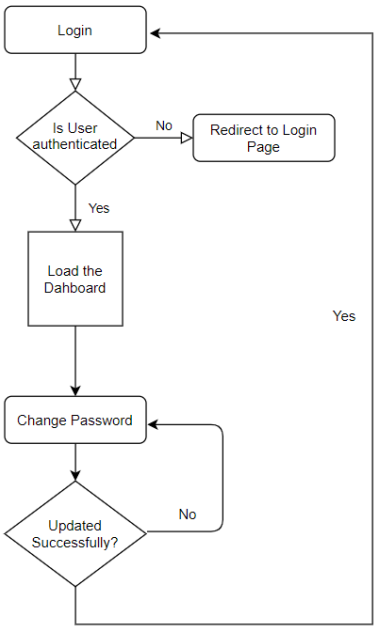

Fig 4.6 Change Password

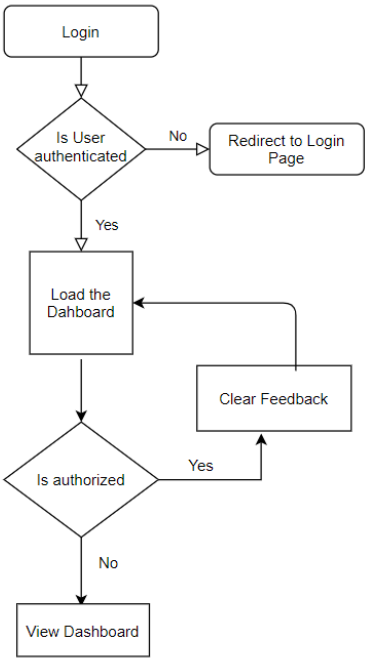

Fig 4.7 Clear feedback by Admin

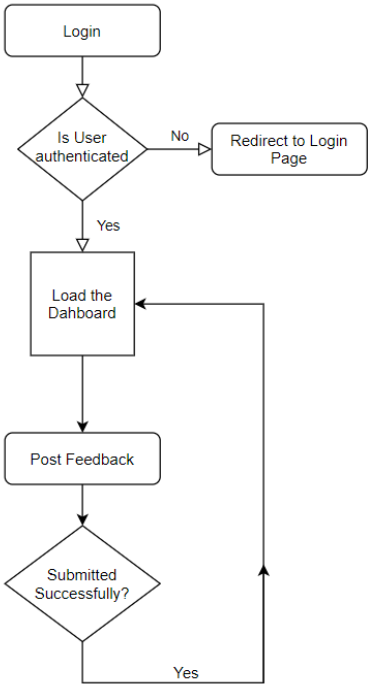

Fig 4.8 Post Feedback

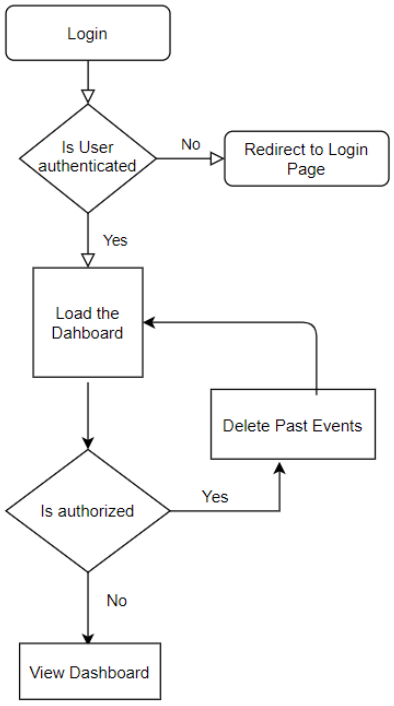

Fig 4.9 Delete Past Events

4.4 TABLES USED

- Default table created by Django to store necessary details about the users.

Table 4.1 Auth_user Table

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
id	integer			Yes	Yes
password	character varying	128		Yes	No
last_login	timestamp with time zone			No	No
is_superuser	boolean			Yes	No
username	character varying	150		Yes	No
first_name	character varying	30		Yes	No
last_name	character varying	150		Yes	No
email	character varying	254		Yes	No
is_staff	boolean			Yes	No
is_active	boolean			Yes	No
date_joined	timestamp with time zone			Yes	No

- Profile table created to store details such as registration number, department etc. about the users.

Table 4.2 Profile Table

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
id	integer			Yes	Yes
image	character varying	100		Yes	No
user_id	integer			Yes	No
dept	character varying	120		No	No
registration_number	character varying	12		No	No
event_ids	text			No	No
company	character varying	200		No	No
job_role	character varying	100		No	No
work_location	character varying	100		No	No

- Polls table to store the yes and no counts of an event.

Table 4.3 Polls Table

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
id	integer			Yes	Yes
yes_count	integer			Yes	No
no_count	integer			Yes	No
event_id_id	integer			Yes	No

- Events table to store details about an event created by users

Table 4.4 Events Table

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
id	integer			Yes	Yes
date_added	date			Yes	No
event_date	date			Yes	No
event_subject	character varying	40		Yes	No
organizer_name	character varying	50		Yes	No
text	character varying	400		Yes	No
email	character varying	254		Yes	No
venue	character varying	200		Yes	No
user_id	integer			Yes	No

- Table to store details about blogs created by user

Table 4.5 Posts/Blogs Table

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
id	integer			Yes	Yes
title	character varying	100		Yes	No
content	text			Yes	No
date_posted	timestamp with time zone			Yes	No
author_id	integer			Yes	No

- Feedback table to store details about feedback posted by users

Table 4.6 Feedback Table

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
id	integer			Yes	Yes
category	character varying	3		Yes	No
date_added	date			Yes	No
text	character varying	400		Yes	No
email	character varying	254		Yes	No
user_id	integer			Yes	No

4.5 SOURCE CODE

4.5.1 Users app

Models.py

```
from django.contrib.postgres.fields import ArrayField, IntegerRangeField

from django.db import models

from django.contrib.auth.models import User

from PIL import Image

# Create your models here.

CSE = 'COMPUTER SCIENCE AND ENGINEERING'

IT = 'INFORMATION TECHNOLOGY'

ECE = 'ELECTRONICS AND COMMUNICATION ENGINEERING'

EEE = 'ELECTRICAL AND ELECTRONIC ENGINEERING'

ME = 'MECHANICAL ENGINEERING'

CE = 'CIVIL ENGINEERING'
```

```
NONE = ''


class Profile(models.Model):
    """
    User objects have the following fields
    username
    first_name
    last_name
    email
    password
    event_id
    """
    DEPT_CHOICES = (
        (NONE,''),
        (CSE ,'COMPUTER SCIENCE AND ENGINEERING'),
        (IT,  'INFORMATION TECHNOLOGY'),
        (ECE , 'ELECTRONICS AND COMMUNICATION ENGINEERING'),
        (EEE , 'ELECTRICAL AND ELECTRONIC ENGINEERING'),
        (ME , 'MECHANICAL ENGINEERING'),
        (CE , 'CIVIL ENGINEERING'),
        )
```

```
user = models.OneToOneField(User, on_delete=models.CASCADE)
image = models.ImageField(default='default.jpg', upload_to='profile_pics')
dept = models.CharField(max_length=120,
                        choices=DEPT_CHOICES,
                        default=NONE,
                        null=True)
registration_number = models.CharField(max_length=12, blank=True, null=True)
job_role = models.CharField(max_length=100,blank=True,null=True)
work_location = models.CharField(max_length=100,blank=True,null=True)
company = models.CharField(max_length=200,blank=True,null=True)
event_ids = models.TextField(null=True)
def __str__(self):
    return f'{self.user.username} Profile'
# Resizing the image to a smaller size
def save(self, *args, **kwargs):
    super(Profile, self).save(*args, **kwargs)
    img = Image.open(self.image.path)
    if img.height > 300 or img.width > 300:
        output_size = (300, 300)
        img.thumbnail(output_size)
        img.save(self.image.path)
```

Views.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import render, redirect
from django.contrib.auth.models import User
from django.contrib.auth import login, authenticate
from .forms import UserRegisterForm,ProfileRegisterForm,
UserUpdateForm,ProfileUpdateForm
from django.contrib import messages
from django.contrib.auth import update_session_auth_hash
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth.decorators import login_required
from .models import Profile
# Create your views here.

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        profile_form = ProfileRegisterForm(request.POST)
        if form.is_valid() and profile_form.is_valid():
            try:
                user = form.save()
                userprofile = Profile.objects.update_or_create(
                    user=user,
                    dept = profile_form.cleaned_data['dept'],
                    registration_number = profile_form.cleaned_data['registration_number'].upper(),
                    job_role = profile_form.cleaned_data['job_role'],
                    work_location=profile_form.cleaned_data['work_location'],
                    company=profile_form.cleaned_data['company']
```

```
            )
            username = form.cleaned_data.get('username')
            raw_password = form.cleaned_data.get('password1')
            messages.success(request, f'Your account has been created successfully')
            user = authenticate(username=username, password=raw_password)
            login(request, user)
            return redirect('dash-home')
        except Exception:
            messages.warning(request,f'There was some issue')
            return HttpResponseRedirect('users/login.html')
    else:
        form = UserRegisterForm()
        profile_form = ProfileRegisterForm()
    return render(request, 'users/register.html', {'form': form,'profile_form':profile_form})

@login_required
def profile(request):
    if request.method == 'POST':
        u_form = UserUpdateForm(request.POST, instance=request.user)
        p_form = ProfileUpdateForm(request.POST,
                                   request.FILES,
                                   instance=request.user.profile)

        if u_form.is_valid() and p_form.is_valid():
            u_form.save()
            p_form.save()
            messages.success(request, f'Your account has been updated')
            return redirect('profile')
```

```
    else:
        u_form = UserUpdateForm(instance=request.user)
        p_form = ProfileUpdateForm(instance=request.user.profile)


    context = {
        'u_form': u_form,
        'p_form': p_form
    }
    return render(request, 'users/profile.html', context)

@login_required
def view_profile(request,*args,**kwargs):
    if request.method == 'GET':
        user = Profile.objects.get(id=kwargs['id'])
        return render(request, 'users/view_profile.html',context={'view_user':user, 'title':'User
Profile'})

    return render(request, 'users/view_profile.html',context={'title':'User Profile'})

@login_required
def update_password(request,*args,**kwargs):
    if request.method == 'POST':
        form = PasswordChangeForm(user=request.user,data = request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, form.user)
```

```
            messages.success(request,f'Password updated successfully')
            return HttpResponseRedirect('logout')
        else:
            messages.warning(request,f"Passwords doesn't pass the required criteria or doesn't match.Please check")
            return render(request,template_name='users/profile.html')
    else:
        password_form = PasswordChangeForm(request.user)
        context={
        'password_form':password_form,
        'title':'Change Password'
        }
    return render(request,'users/profile.html',context=context)
```

Forms.py

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm,PasswordChangeForm
from django.core.exceptions import ValidationError

from .models import Profile



CSE = 'COMPUTER SCIENCE AND ENGINEERING'
IT = 'INFORMATION TECHNOLOGY'
ECE = 'ELECTRONICS AND COMMUNICATION ENGINEERING'
EEE = 'ELECTRICAL AND ELECTRONIC ENGINEERING'
ME = 'MECHANICAL ENGINEERING'
CE = 'CIVIL ENGINEERING'
```

```
NONE = ''

DEPT_CHOICES = (
        (NONE,''),
        (CSE ,'COMPUTER SCIENCE AND ENGINEERING'),
        (IT,  'INFORMATION TECHNOLOGY'),
        (ECE , 'ELECTRONICS AND COMMUNICATION ENGINEERING'),
        (EEE , 'ELECTRICAL AND ELECTRONIC ENGINEERING'),
        (ME , 'MECHANICAL ENGINEERING'),
        (CE , 'CIVIL ENGINEERING'),
        )

class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()
    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']

class ProfileRegisterForm(forms.ModelForm):
    registration_number = forms.CharField(max_length=12)
    job_role = forms.CharField(max_length=100, required=False)
    work_location = forms.CharField(max_length=100, required=False)
    company = forms.CharField(max_length=200, required=False)
    class Meta:
        model = Profile
        fields = ['dept', 'registration_number', 'job_role','work_location','company']

class UserUpdateForm(forms.ModelForm):
```

```
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email']

    def clean_email(self):
        return self.cleaned_data['email'].lower()

class ProfileUpdateForm(forms.ModelForm):
    registration_number = forms.CharField(max_length=12, required=False)
    job_role = forms.CharField(max_length=100,required=False)
    work_location = forms.CharField(max_length=100, required=False)
    company = forms.CharField(max_length=200, required=False)

    class Meta:
        model = Profile
        fields = ['dept', 'registration_number', 'job_role','work_location','company','image']

    def clean_registration_number(self):
        return self.cleaned_data['registration_number'].upper()
```

Urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register, name='register'),
    path('view_user/<int:id>', views.view_profile, name='user_profile'),
```

```
    path('update_password',views.update_password,name='update_password')
]
```

4.5.2 Dash app

Models.py

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
# Create your models here.


class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    date_posted = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(User, on_delete=models.CASCADE)

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

Views.py

```
from django.shortcuts import render
from django.db.models import Q
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
from django.views.generic import (
    View,
    ListView,
    DetailView,
```

```
    CreateView,
    UpdateView,
    DeleteView
)
from .models import Post
from users.models import Profile
# Create your views here.


def home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'dash/home.html', context)


class PostListView(LoginRequiredMixin ,ListView):
    model = Post
    template_name = 'dash/home.html'  # <app>/<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']


class PostDetailView(LoginRequiredMixin ,DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
```

```
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)

class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post
    success_url = '/'

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
```

```
            return True
        return False


class SearchUserView(LoginRequiredMixin,View):
    def get(self,request,*args,**kwargs):


        if request.GET.get('query') != '':
            search = request.GET.get('query')
            users =
Profile.objects.filter(Q(user__username__icontains=search)|Q(job_role__iexact=search)|Q(work
_location__iexact=search)|Q(company__iexact=search)\
                                    |Q(dept__iexact =
search)|Q(registration_number__iexact=search)).order_by('-user_id__date_joined')
            context = {
                'title': 'Search User',
                'users': users,
                'query': search
            }
            return render(request,'dash/search_list.html', context)
        else:
            return render(request, 'dash/home.html', {'title':'Dash-Home'})


def about(request):
    return render(request, 'dash/about.html', { 'title' : 'About'})
```

Urls.py

```
from django.urls import path
from .views import (
    PostListView,
```

```
    PostDetailView,
    PostCreateView,
    PostUpdateView,
    PostDeleteView,
    SearchUserView
)
from . import views

urlpatterns = [
    path('', PostListView.as_view(), name='dash-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-create'),
    path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
    path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-delete'),
    path('about/', views.about, name='dash-about'),
    path('search/', SearchUserView.as_view(), name='search-users')
]
```

4.5.3 Events app

Models.py

```
from django.db import models
from users.models import Profile
import datetime

class Events(models.Model):
    user = models.ForeignKey(Profile, on_delete=models.CASCADE)
    date_added = models.DateField(auto_now=True)
    event_date = models.DateField(default=datetime.date.today)
    event_subject = models.CharField(max_length=40)
```

```
    organizer_name = models.CharField(max_length=50)
    text = models.CharField(max_length=400)
    email = models.EmailField()
    venue = models.CharField(max_length=200,default=0)

    def __str__(self):
        return self.event_subject

class Poll(models.Model):
    event_id = models.ForeignKey(Events, on_delete=models.CASCADE)
    yes_count = models.IntegerField(default=0)
    no_count = models.IntegerField(default=0)

    def __str__(self):
        return str(self.event_id)
```

Views.py

```
from django.shortcuts import render
from .forms import EventsCreation
from django.views import View
from django.views.generic import UpdateView
from django.contrib.auth.mixins import LoginRequiredMixin,UserPassesTestMixin
from users.models import Profile
from django.contrib.auth.models import User
from django.contrib import messages
from django.shortcuts import get_object_or_404
import requests
from datetime import datetime
```

```
from libs.mailgun import Mailgun
import json
from django.core.mail import EmailMultiAlternatives
from .models import Events, Poll

class events_creation(LoginRequiredMixin,View):
    def get(self, request, *args, **kwargs):
        initial = {'username': request.user.username}
        form = EventsCreation(instance=request.user, initial=initial)

        #for updating the event
        if kwargs:
            event = get_object_or_404(Events,id=kwargs['id'])
            form = EventsCreation(instance = event,initial = initial)
            return
render(request,template_name='create_event.html',context={'event_id':event.id,'form':form,'title':'
Event Form','user_logged':request.user})

        return render(request,template_name='create_event.html',context={'form': form, 'title':
'Event Form','user_logged':request.user})


def post(self, request, *args, **kwargs):
            initial = {'username': request.user.username}
            form = EventsCreation(request.POST, initial=initial)
            if form.is_valid():
                        #update_or_create should be done
                        form = form.save(commit = False)
                        form.user = get_object_or_404(Profile,user_id=request.user.id)
```

```
                    if request.POST.get('event_id'):
                            event = Events.objects.get(pk =
int(request.POST.get('event_id').strip('/')))
                            form = EventsCreation(data=request.POST,instance=event)
                            form.save()
                            messages.success(request, f'Event has been updated successfully')
                    else:
                            form.save()
                            messages.success(request, send_simple_message(form.id, None))
                            messages.success(request, f'Event created successfully')
                    return render(request,"succes.html", context={'title': 'Event Success'})

            return render(request, 'create_event.html', {'form': form, 'title': 'Event
Form','user_logged':request.user})

class view_events(View):
    def get(self, request):
        poll = Poll.objects.all()
        #taking current user id
        user_id = request.user.id
        #if the user is logged in we show polls for user to vote
        if user_id != None:
            user = Profile.objects.get(user_id=user_id)
            jsonDec = json.decoder.JSONDecoder()
            query_results = Events.objects.all().order_by('-date_added')

            #if the user has never voted in the poll before then we send false ids to the template
            #else we send the event ids which user has voted before as true for the template to show
the vote persentages
```

```
        if user.event_ids is None:
            ids = [False]*len(query_results)
        else:
            ids = [True if x.id in jsonDec.decode(user.event_ids) else False for x in query_results]

        query_results = zip(query_results,ids)
        context={'query_results':query_results,'user_logged':request.user,'poll_results':poll}
        return render(request, template_name="view_events.html",context=context)
    #this is the part where we won't send the poll info to the template to display
    else:
        query_results = Events.objects.all()
        ids = [False] * len(query_results)
        query_results = zip(query_results, ids)
        context = {'query_results': query_results, 'user_logged': request.user}
        return render(request, template_name="view_events.html", context=context)


def post(self, request):
    event_id = int(request.POST.get('event_id'))
    try:
        #check if the record has been already there for the poll with this current user or create one
        poll = Poll.objects.get_or_create(event_id=event_id)[0]
        user_id = request.user.id
        user = Profile.objects.get(user_id = user_id)

        #incrementing the yes count if poll result is 1 then increment yes count
        if(int(request.POST.get('result'))):
            poll.yes_count += 1
```

```
            send_simple_message(event_id, request.user)
        #and if it is 0 then increment no count
        else:
            poll.no_count += 1
        poll.save()
        #if user is voting for the first time then dump the event id in the user data as it is
        if user.event_ids is None:
            event_ids = list(map(int, str(request.POST.get('event_id'))))
            user.event_ids = json.dumps(event_ids)
        #append the new event_id to the existing event_ids in the user data
        else:
            jsonDec = json.decoder.JSONDecoder()
            ids = jsonDec.decode(user.event_ids)
            ids.append(event_id)
            user.event_ids = json.dumps(ids)
        user.save()
    except Exception:
        event = Events.objects.get(id = event_id)
        user_id = request.user.id
        user = Profile.objects.get(user_id=user_id)
        yes_count = 0
        no_count = 0
        if (int(request.POST.get('result'))):
            yes_count = 1
            send_simple_message(event_id, request.user)
        else:
            no_count = 1
        poll = Poll(event_id=event,yes_count=yes_count,no_count=no_count)
```

```
            poll.save()
            if user.event_ids is None:
                event_ids = list(map(int, str(request.POST.get('event_id'))))
                user.event_ids = json.dumps(event_ids)
            else:
                jsonDec = json.decoder.JSONDecoder()
                ids = jsonDec.decode(user.event_ids)
                ids.append(event_id)
                user.event_ids = json.dumps(ids)
            user.save()
        query_results = Events.objects.all()
        jsonDec = json.decoder.JSONDecoder()
        ids = [True if x.id in jsonDec.decode(user.event_ids) else False for x in query_results]
        query_results = zip(query_results, ids)
        poll = Poll.objects.all()
        context = {'query_results': query_results,'user_logged': request.user,'poll_results':poll}
        return render(request, template_name="view_events.html",context=context)


def send_simple_message(event_id,user_details):
    try:
        recievers = []
        #gather list of mails in the database
        for user in User.objects.all():
            recievers.append(user.email)
        details = Events.objects.get(id = event_id)
        #triggering this when event is created
        if user_details == None:
            Mailgun.send_mail(recievers, "Checkout this Event!","Checkout this Event!",
```

```
            "<p>Title: "+str(details.event_subject)+"<br>Event Date:
"+str(details.event_date)+"<br>Organizer name: "+ str(details.organizer_name)+"<br>Details:
"+str(details.text)+"<br> Venue: "+details.venue+"<br><br> please write to
kamatalaashish@gmail.com in case of any queries </p>")
        #triggering this when yes clikced on poll
        else:
            Mailgun.send_mail([user_details.email], "Thankyou For showing interest!", "Thankyou
For showing interest!",
                    "<p>Hi "+str(user_details.username)+",<br><br> Thankyou for registering for the
event - "+str(details.event_subject)+".The Event will be held on "+str(details.event_date)+" make
sure you are available.<br><br> Regards,<br> Team AMS <p>")
    except Exception:
        print("Something went wrong!")


class EventsDeleteView(View):
    def post(self,request,*args,**kwargs):
        current_date = datetime.now()
        past_events = Events.objects.filter(event_date__lt=current_date)
        past_events.delete()
        messages.warning(request, f'Past Events deleted successfully')
        return render(request,'succes.html',context={'title':'Delete Success'})
```

Forms.py

```
from django import forms
from .models import Events,Poll
import datetime

#date checker
def present_or_future_date(value):
```

```
    if value < datetime.date.today():
        raise forms.ValidationError("The date cannot be in the past!")
    return value

class EventsCreation(forms.ModelForm):
    text = forms.CharField(max_length=400, widget=forms.Textarea())
    event_date = forms.DateField(
            help_text="MM/DD/YYYY",validators=[present_or_future_date])
    venue = forms.CharField(empty_value='')
class Meta:

        model = Events
        exclude = ['username','user']


    def __init__(self,*args,**kwargs):
        super(EventsCreation, self).__init__(*args, **kwargs)
        instance = getattr(self, 'instance', None)
        if instance and instance.id:
            self.fields['email'].widget.attrs['readonly'] = True

class Polls(forms.ModelForm):
    class Meta:
        model = Poll
        include = ['event_id']
        exclude = ['username']
```

Urls.py

```
from django.urls import path
from .views import *
```

```
urlpatterns =[
    path(r'events/', events_creation.as_view(), name="events"),
    path(r'view_events/', view_events.as_view(), name ="view_events"),
    path(r'event_update/<int:id>',events_creation.as_view(),name='event-update'),
    path(r'delete/',EventsDeleteView.as_view(),name='event-delete'),
]
```

4.5.4 Feedback app

Models.py

```
from django.db import models
from users.models import Profile

# Create your models here.
INFRASTRCUTURE = 'INF'
GENERIC = 'GEN'
CURRICULUM = 'CUR'
class Feedback(models.Model):

    FEEDBACK_CHOICES = (
        (INFRASTRCUTURE,'Infrastructure'),
        (GENERIC,'Generic'),
        (CURRICULUM,'Curriculum'),
    )



    user = models.ForeignKey(Profile, on_delete=models.CASCADE)
    category = models.CharField(max_length=3,
                            choices = FEEDBACK_CHOICES,
                            default=GENERIC)
```

```
    date_added = models.DateField(auto_now_add=True)
    text = models.CharField(max_length=400)
    email= models.EmailField()

    def __str__(self):
        return self.user.user_id
```

Views.py

Insert_feedback.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.contrib import messages
from django.views import View
from django.shortcuts import render
from ..forms import FeedbackForm
from users.models import Profile
from django.contrib.auth.mixins import LoginRequiredMixin

class FeedbackView(LoginRequiredMixin,View):
    def get(self,request, *args, **kwargs):
        initial = {'username':request.user.username}
        form = FeedbackForm(instance=request.user, initial=initial, size = 1, maxChars=400)
        return render(request,
                      template_name='insert_feedback.html',
                      context={'form': form, 'title': 'Feedback Form'})

    def post(self, request, *args, **kwargs):
        initial = {'username':request.user.username}
```

```
        form = FeedbackForm(request.POST, initial=initial,size=1,maxChars=400)
        if form.is_valid():
            form = form.save(commit = False)
            form.user = Profile.objects.get(id=request.user.id)
            form.save()
            messages.success(request, f'Feedback has been submitted successfully')
            return render(request, "success.html", context={'title': 'Feedback Success'})

        return render(request, 'insert_feedback.html', {'form': form, 'title': 'Feedback Form'})
```

view_feedback.py

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from django.views import View
from django.shortcuts import render
from django.contrib.auth.mixins import LoginRequiredMixin
from ..models import Feedback

class ViewAllFeedback(LoginRequiredMixin, View):

    def get(self, request, *args, **kwargs):
        if request.user.is_superuser:
            if kwargs:
                feedback_entries = Feedback.objects.filter(category=kwargs['cat'])
                return render(request, template_name="view_feedback.html", context={'feedback':
feedback_entries, 'title': 'View Feedback'})

            feedback_entries = Feedback.objects.all()
```

```
        return render(request, template_name="view_feedback.html", context={'feedback':
feedback_entries, 'title': 'View Feedback'})

    return render(request, template_name="dash/stalker.html", context={'title': 'Alert'} )

class ClearFeedbackView(LoginRequiredMixin, View):
  def get(self,request):
    Feedback.objects.all().delete()
    feedback_entries = Feedback.objects.all()
    return render(request, template_name="view_feedback.html" , context={'title':'View
Feedback','feedback' :feedback_entries })

  def post(self,request):
    delete = Feedback.objects.all()
    delete.delete()
```

Forms.py

```
from django import forms

from ..models import Feedback
INFRASTRCUTURE = 'INF'
GENERIC = 'GEN'
CURRICULUM = 'CUR'

FEEDBACK_CHOICES = (
    (INFRASTRCUTURE,'Infrastructure'),
    (GENERIC,'Generic'),
    (CURRICULUM,'Curriculum'),
  )
```

```
class FeedbackForm(forms.ModelForm):
    username = forms.CharField(max_length=120, required=False)
    text = forms.CharField(max_length=400, widget=forms.Textarea())
    class Meta:
        model = Feedback
        include = ['username']
        exclude = ['user']

    def __init__(self, *args, **kwargs):
        size = kwargs.pop('size')
        maxChars = kwargs.pop('maxChars')
        super(FeedbackForm, self).__init__(*args, **kwargs)
self.fields['email'].widget.attrs['readonly'] = True
        self.fields['username'].widget.attrs['readonly'] = True
        self.fields['text'].widget.attrs['onkeypress'] = 'return
textCounter(this,this.form.counter,%d);'% maxChars
        self.fields['text'].widget.attrs['rows'] = size
        self.fields['text'].widget.attrs['cols'] ='40'
```

Urls.py

```
from django.urls import path

from .views import *

urlpatterns =[
    path(r'feedback/', FeedbackView.as_view(), name="feedback_home"),
```

```
    path(r'view/', ViewAllFeedback.as_view(), name="view_feedback"),
    path(r'view/<str:cat>', ViewAllFeedback.as_view(), name="view_cat_feedback"),
    path(r'feedback/delete', ClearFeedbackView.as_view(), name="clear_feedback"),
]
```

4.5.5 Android SDK

MainActivity.java

```
package com.example.alumnimanaementsystem;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.os.Parcelable;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.webkit.ValueCallback;
import android.net.Uri;
import android.webkit.WebChromeClient;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Toast;

import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
```

```
import java.util.Date;

public class MainActivity extends AppCompatActivity {
    private WebView mywebview;
    private static final int INPUT_FILE_REQUEST_CODE = 1;
    private static final int FILECHOOSER_RESULTCODE = 1;
    private static final String TAG = MainActivity.class.getSimpleName();
    private ValueCallback<Uri> mUploadMessage;
    private Uri mCapturedImageURI = null;
    private ValueCallback<Uri[]> mFilePathCallback;
    private String mCameraPhotoPath;
    String url = "http://10.0.2.2:8000/";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mywebview = (WebView) findViewById(R.id.webView);
        WebSettings webSettings = mywebview.getSettings();
        webSettings.setJavaScriptEnabled(true);
        mywebview.getSettings().setAllowFileAccess(true);
        mywebview.setScrollbarFadingEnabled(false);
        mywebview.setWebViewClient(new WebViewClient(){
            @Override
            public void onPageFinished(WebView view, String url) {
                try{
                Thread.sleep(1000);
                }catch (InterruptedException e){
                    e.printStackTrace();
                }
```

```
            findViewById(R.id.splashScreen).setVisibility(View.GONE);
            mywebview.setVisibility(View.VISIBLE);
        }
    });
    mywebview.setWebChromeClient(new WebChromeClient() {

        @RequiresApi(api = Build.VERSION_CODES.JELLY_BEAN_MR2)
        public boolean onShowFileChooser(WebView view, ValueCallback<Uri[]> filePath,
WebChromeClient.FileChooserParams fileChooserParams) {
            // Double check that we don't have any existing callbacks
            if (mFilePathCallback != null) {
                mFilePathCallback.onReceiveValue(null);
            }
            mFilePathCallback = filePath;
            Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
                // Create the File where the photo should go
                File photoFile = null;
                try {
                    photoFile = createImageFile();
                    takePictureIntent.putExtra("PhotoPath", mCameraPhotoPath);
                } catch (IOException ex) {
                    // Error occurred while creating the File
                    Log.e(TAG, "Unable to create Image File", ex);
                }
                // Continue only if the File was successfully created
                if (photoFile != null) {
                    mCameraPhotoPath = "file:" + photoFile.getAbsolutePath();
                    takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                            Uri.fromFile(photoFile));
```

```
        } else {
            takePictureIntent = null;
        }
    }
    Intent contentSelectionIntent = new Intent(Intent.ACTION_GET_CONTENT);
    contentSelectionIntent.addCategory(Intent.CATEGORY_OPENABLE);
    contentSelectionIntent.setType("image/*");
    Intent[] intentArray;
    if (takePictureIntent != null) {
        intentArray = new Intent[]{takePictureIntent};
    } else {
        intentArray = new Intent[0];
    }
    Intent chooserIntent = new Intent(Intent.ACTION_CHOOSER);
    chooserIntent.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true);
    chooserIntent.putExtra(Intent.EXTRA_INTENT, contentSelectionIntent);
    chooserIntent.putExtra(Intent.EXTRA_TITLE, "Image Chooser");
    chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS, intentArray);
    startActivityForResult(chooserIntent, INPUT_FILE_REQUEST_CODE);
    return true;
}

// openFileChooser for Android 3.0+
public void openFileChooser(ValueCallback<Uri> uploadMsg, String acceptType) {
    mUploadMessage = uploadMsg;
    // Create AndroidExampleFolder at sdcard
    // Create AndroidExampleFolder at sdcard
    File imageStorageDir = new File(
            Environment.getExternalStoragePublicDirectory(
                    Environment.DIRECTORY_PICTURES)
```

```
            , "AndroidExampleFolder");
        if (!imageStorageDir.exists()) {
           // Create AndroidExampleFolder at sdcard
           imageStorageDir.mkdirs();
        }
        // Create camera captured image file path and name
        File file = new File(
              imageStorageDir + File.separator + "IMG_"
                    + String.valueOf(System.currentTimeMillis())
                    + ".jpg");
        mCapturedImageURI = Uri.fromFile(file);
        // Camera capture image intent
        final Intent captureIntent = new Intent(
              android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        captureIntent.putExtra(MediaStore.EXTRA_OUTPUT, mCapturedImageURI);
        Intent i = new Intent(Intent.ACTION_GET_CONTENT);
        i.addCategory(Intent.CATEGORY_OPENABLE);
        i.setType("image/*");
        // Create file chooser intent
        Intent chooserIntent = Intent.createChooser(i, "Image Chooser");
        // Set camera intent to file chooser
        chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS
              , new Parcelable[]{captureIntent});
        // On select image call onActivityResult method of activity
        startActivityForResult(chooserIntent, FILECHOOSER_RESULTCODE);
     }
  });
  mywebview.loadUrl(url);
}
@Override
```

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        if (requestCode != INPUT_FILE_REQUEST_CODE || mFilePathCallback == null) {
            super.onActivityResult(requestCode, resultCode, data);
            return;
        }
        Uri[] results = null;
        // Check that the response is a good one
        if (resultCode == RESULT_OK) {
            if (data == null) {
                // If there is not data, then we may have taken a photo
                if (mCameraPhotoPath != null) {
                    results = new Uri[]{Uri.parse(mCameraPhotoPath)};
                }
            } else {
                String dataString = data.getDataString();
                if (dataString != null) {
                    results = new Uri[]{Uri.parse(dataString)};
                }
            }
        }
        mFilePathCallback.onReceiveValue(results);
        mFilePathCallback = null;
    } else if (Build.VERSION.SDK_INT <= Build.VERSION_CODES.KITKAT) {
        if (requestCode != FILECHOOSER_RESULTCODE || mUploadMessage == null) {
            super.onActivityResult(requestCode, resultCode, data);
            return;
        }
        if (requestCode == FILECHOOSER_RESULTCODE) {
            if (null == this.mUploadMessage) {
```

```
            return;
        }
        Uri result = null;
        try {
            if (resultCode == RESULT_OK) {
                result = data == null ? mCapturedImageURI : data.getData();
            }
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(), "activity :" + e,
                    Toast.LENGTH_LONG).show();
        }
        mUploadMessage.onReceiveValue(result);
        mUploadMessage = null;
      }
   }
   return;
}

private File createImageFile() throws IOException {
   // Create an image file name
   String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
   String imageFileName = "JPEG_" + timeStamp + "_";
   File storageDir = Environment.getExternalStoragePublicDirectory(
         Environment.DIRECTORY_PICTURES);
   File imageFile = File.createTempFile(
         imageFileName,  /* prefix */
         ".jpg",         /* suffix */
         storageDir      /* directory */
   );
   return imageFile;
```

```
    }

    public void onBackPressed(){
        if(mywebview.canGoBack()){
            mywebview.goBack();
        }
        else {
            super.onBackPressed();
        }
    }
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

<ImageView
        android:id="@+id/splashScreen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/college_logo"
        android:background="@color/colorWhite"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:visibility="visible" />
<WebView
        android:id="@+id/webView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:visibility="invisible"/>



</androidx.constraintlayout.widget.ConstraintLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.alumnimanaementsystem">

<uses-permission android:name="android.permission.INTERNET" />

<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.NoActionBar">
<activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>

</manifest>
```

4.5.6 Libs

Mailgun.py

```
from typing import List
import os
from requests import Response, post


class MailgunException(Exception):
    def __init__(self, message: str):
        self.message = message


class Mailgun:

    FROM_EMAIL = "<do-not-reply@ams.com>"
```

```
    FROM_TITLE = f"Alumni Management Service <do-not-reply@{FROM_EMAIL}>"

    @classmethod
    def send_mail(cls, email: List[str], subject: str, text: str, html: str) -> Response:
        api_key = "key-4877531ece758788416250bd5229258a"
        domain =
"https://api.mailgun.net/v3/sandbox6e769479e7c84f8fa0efac40249c69f7.mailgun.org"

        if api_key is None:
            raise MailgunException("Failed to load Mailgun API_Key")
        if domain is None:
            raise MailgunException("Failed to load Mailgun Domain")

        response = post(
            f"{domain}/messages",
            auth=("api", api_key),
            data={"from": cls.FROM_EMAIL,
                  "to": email,
                  "subject": subject,
                  "text": text,
                  "html": html})

        if response.status_code != 200:
            print(response.status_code)
            raise MailgunException('An error occurred while sending email')

        else:
            print(response.status_code)
```

```
        return response


# Mailgun.send_mail(['kamatalaashish@gmail.com','siddharth_ramawat@yahoo.com'], "Hello",
"This is mailgun test email", "<p>This is a HTML Test</p>")
```

CHAPTER 5- RESULT ANALYSIS , CONCLUSION & FUTURE WORK

5.1 RESULTS

On hitting the website the login page is shown where the users can either login if they are already registered or create an account.

If registered users login into the system, they are initially redirected to the dashboard. The dashboard has options to choose to create blog, create event, search other users, update their profile, set new password or insert feedback.

The blogs/posts posted by users is visible on the feed of every other registered user on the system. The users can either update or delete the blogs if they are the authors of the particular post or blog.

The events posted by users is also visible on the view events page of every registered user. The users can choose to attend the event by clicking on the poll, if he hasn't already voted in the poll.

If the user who is logged in now is the organizer of the event he has the option of updating the event. The admin or authorized user has access to delete the past events which have event date less than the current date.

The Feedback option is visible only to the admin of the system. The admin can view all feedback given by various users. The data about the user posting feedback is not stored hence remains anonymous for the admin. The admin can further check feedback based on filters such as Infrastructure, Curriculum and General(which is by default taken for all feedbacks).[8]

If any user apart from admin tries to view the feedback the stalker alert page would be shown.

When a user searches for another user, he gets search results displayed. The searched user's profile can be checked by others. In order have the details hidden , users can choose to keep their data private by setting their preferences.

The user can change his password by clicking the settings button on the navigation bar, as soon as the password is successfully updated the user is logged out and has to login with new password.

The Android Application supports similar kind of functionalities. When the user opens the application the splash screen is displayed , he can login and signup to the application.

If the user tries to update his profile picture ,the file manager opens from where he can choose the picture. The screen automatically resizes based on the screen size.

The user can do all other operations similar to the what he can perform in website like search users, create/update events, view events, post feedback, create/update blog, delete blog etc.

5.1.1 Web Application Screenshots

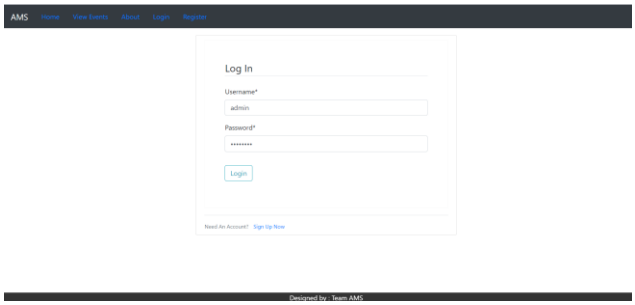

Fig 5.1 Login Page

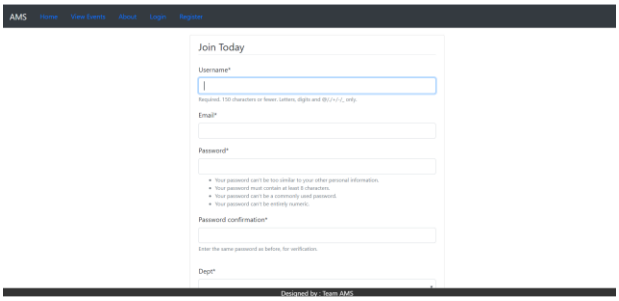

Fig 5.2 Signup Page

The figures 5.1 and 5.2 show the login and sign up page of the system. They are used to register to the system. On successful registration the user is directly logged in to the system. The login page confirms the authenticity of the users and logs user into the system. On successful login , the user can see the dashboard of the system where he has multiple options to choose from.

The user can either post feedback by selecting insert feedback as shown in figure 5.3

Or create events as shown in figure 5.4 where he can mention organizer name, venue ,location etc details. The user can create event on behalf of some other person. Registrants can contact the organizer by using the organizer's email id.

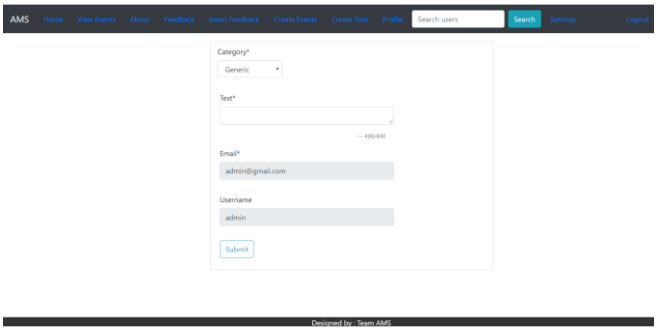

Fig 5.3 Post Feedback

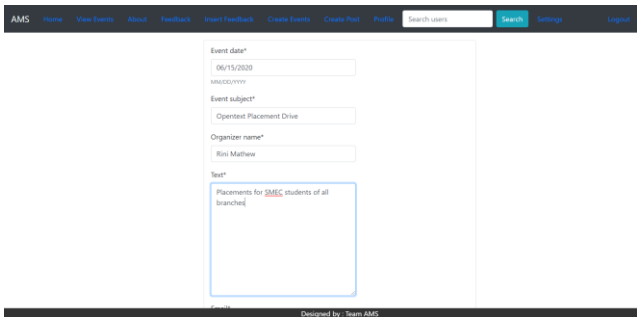

Fig 5.4 Create Events

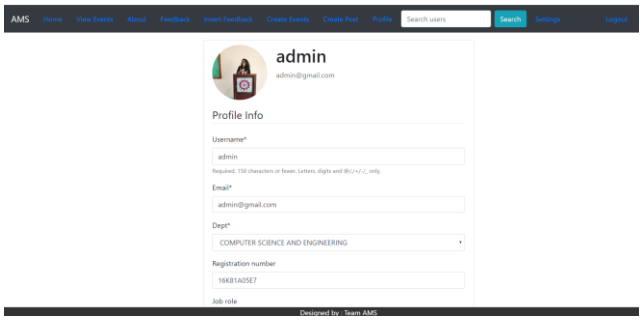

Fig 5.5 Profile Update

The update profile page can be used to update latest company details etc. by the user when he/she changes or switches to another company. They are optional fields. Even if the user gets a promotion he/she can update the job role in the profile update page as shown in figure 5.5 above. The profile picture can also be changed by the user from the same page.

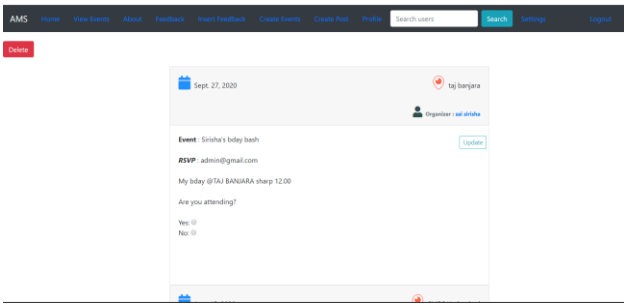

Fig 5.6 View/Update Events

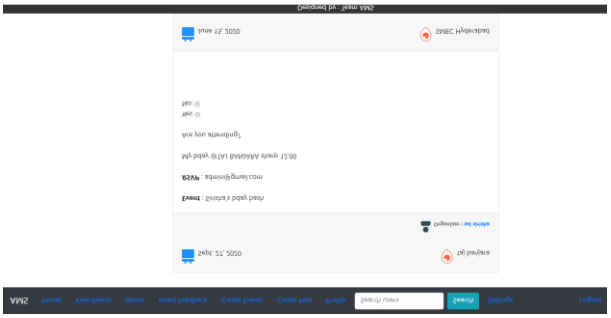

Fig 5.7 View Events

The events can be viewed by all viewers of the system, but can be updated or deleted by only the owner of the event. The admin has the access rights to delete all the past events of the system.

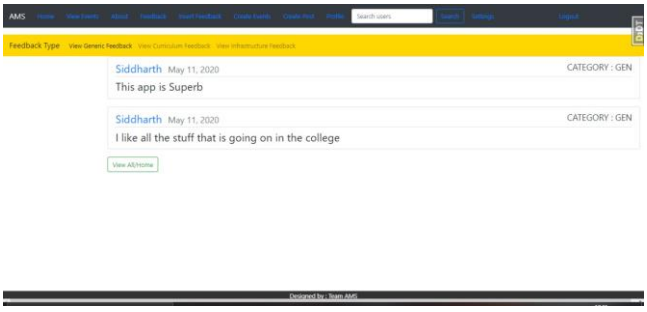

Fig 5.8 View and Delete Feedback by Admin

The feedback posted by the users of the system can only be seen by the authorized people of the organization by clicking on feedback option of navigation bar as shown in figure 5.8. The can segregate the feedback into three categories and also choose to delete the feedback. The system asks for confirmation before deleting entire feedback database. Once the delete option is clicked entire feedback table is emptied.

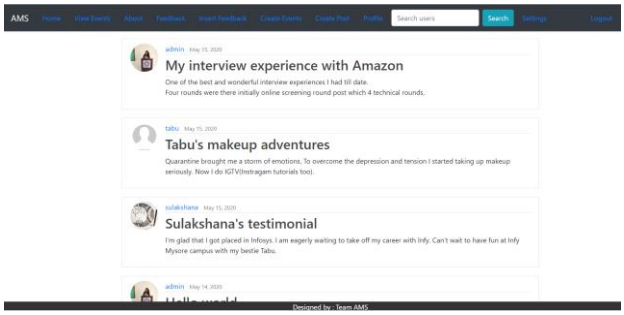

Fig 5.9 Dashboard – Home Page

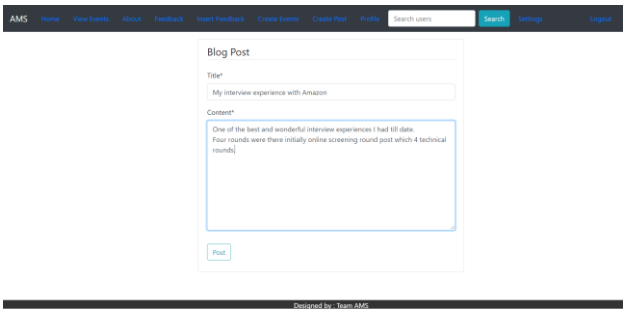

Fig 5.10 Create Blog/Post

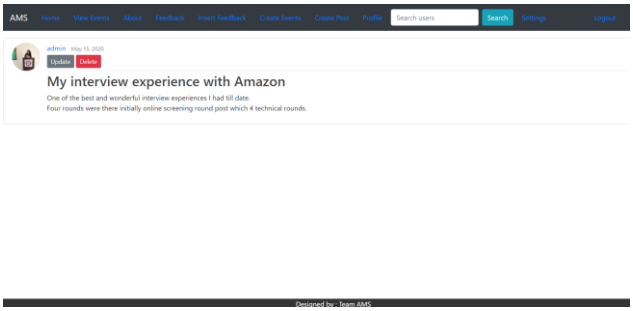

Fig 5.11 Blog/Posts Detail

The figure 5.10 above shows the form used to create blog. The author or the blog has the option to delete or update the blog by clicking on his blog. When he clicks on his blog the blog detail page appears which can be used to modify the blog.

The users can choose to update their password by choosing the settings option on the navigation bar as shown in figure 5.12.

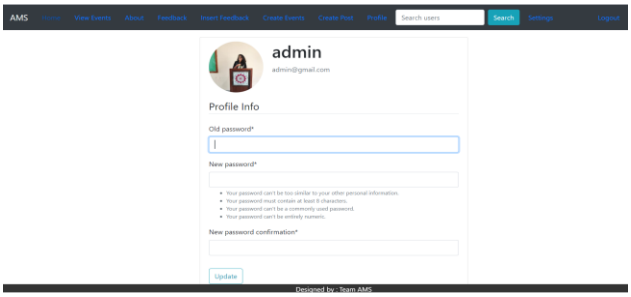

Fig 5.12 Change Password

5.1.2 Android Application Screenshots

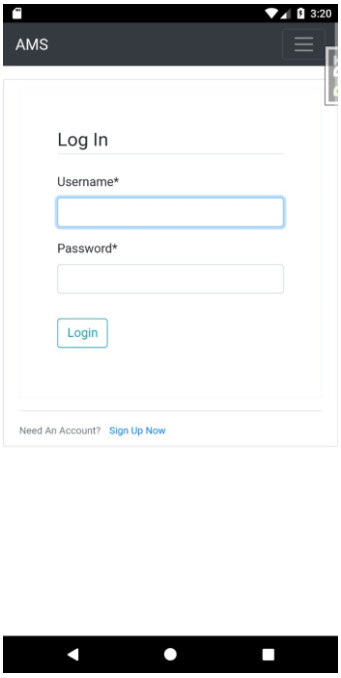

Fig 5.13 Login

Fig 5.14 Splash Screen

The figure 5.13 above shows the login screen of the Android application used to log in to system. As soon as the application is opened the splash screen appears on the application as shown in figure 5.14.

On successful login the dashboard appears on the screen as in figure 5.15 and the navigation bar changes.

The user can choose multiple options from the navigation bar as shown in figure 5.16

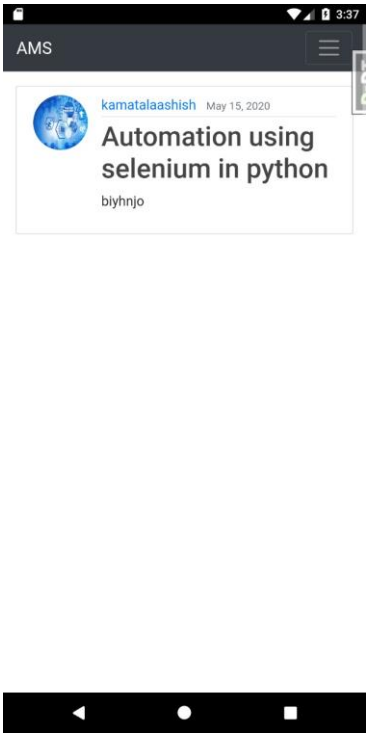

Fig 5.15 Dashboard

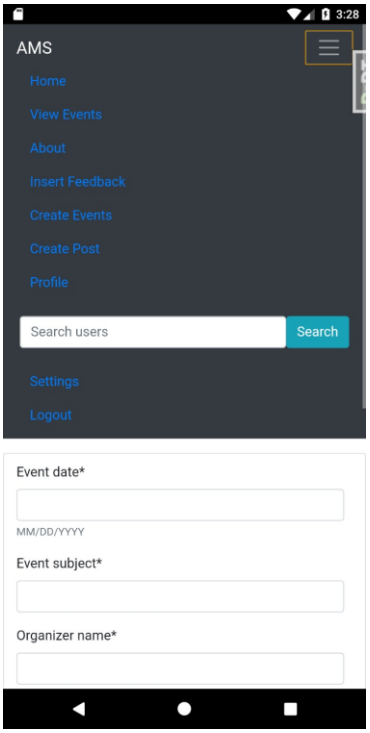

Fig 5.16 Search Users

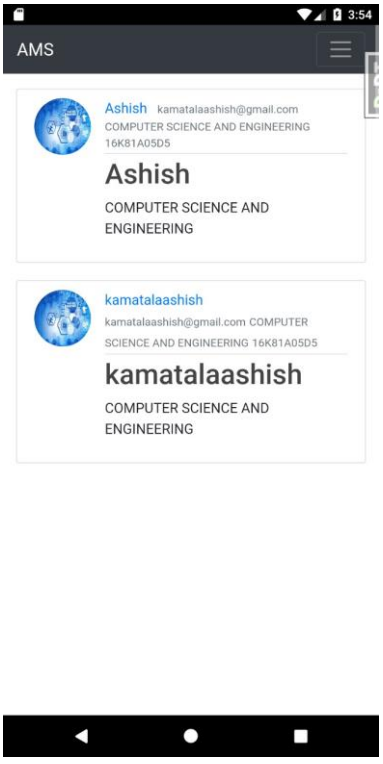

Fig 5.17 Search Results

Fig 5.18 Profile of Searched User

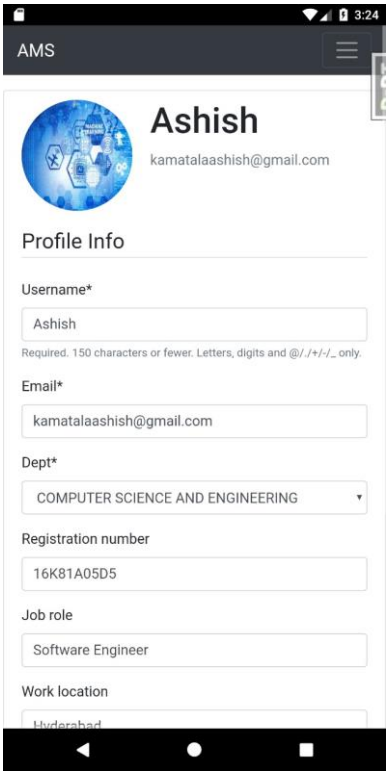

Fig 5.19 Update Profile

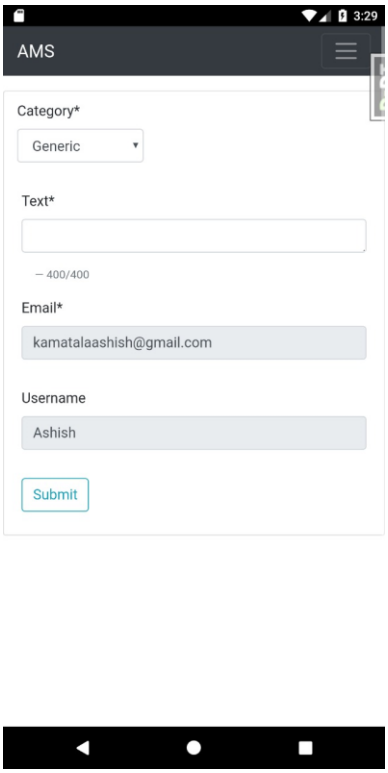

Fig 5.20 Post Feedback

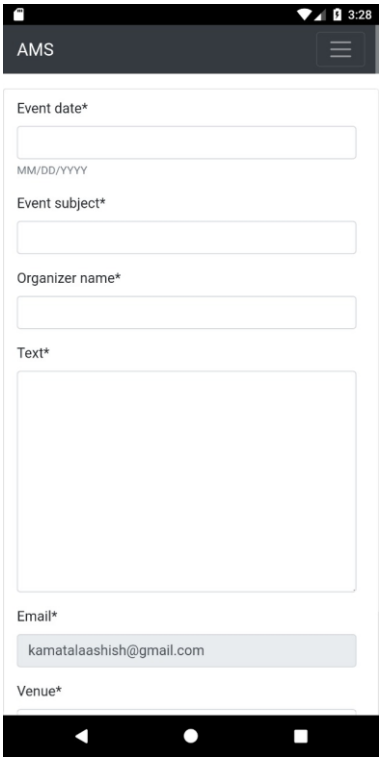

Fig 5.21 Create Event

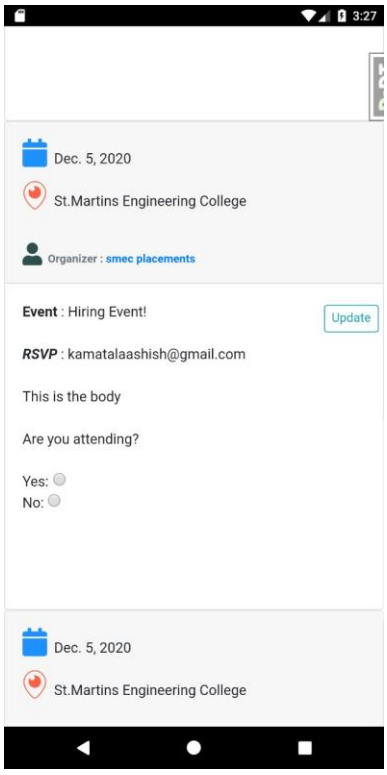

Fig 5.22 View/Update Event

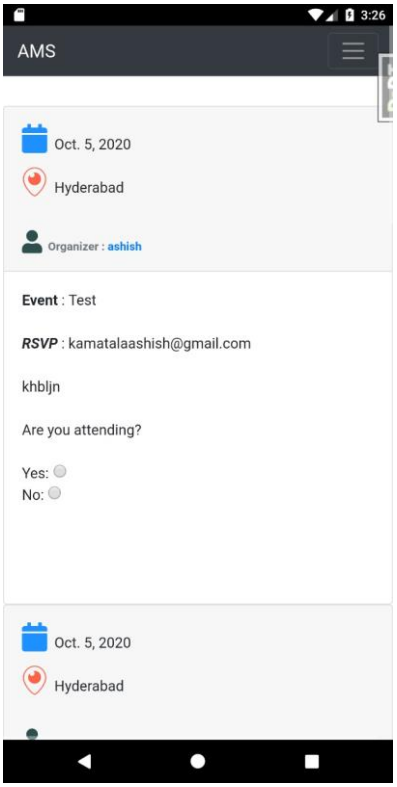

Fig 5.23 View Events

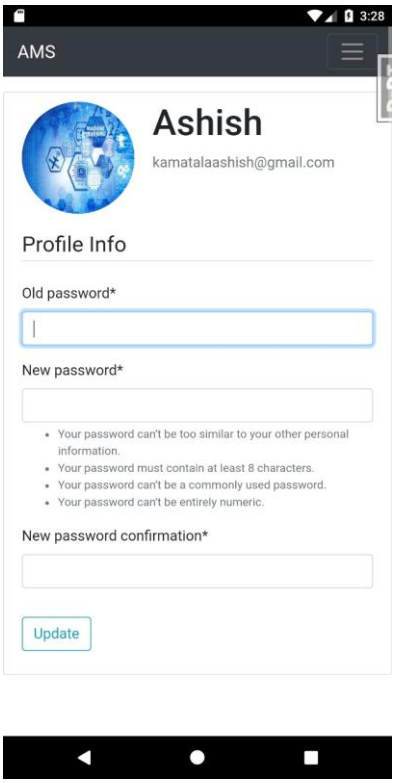

Fig 5.24 Change Password

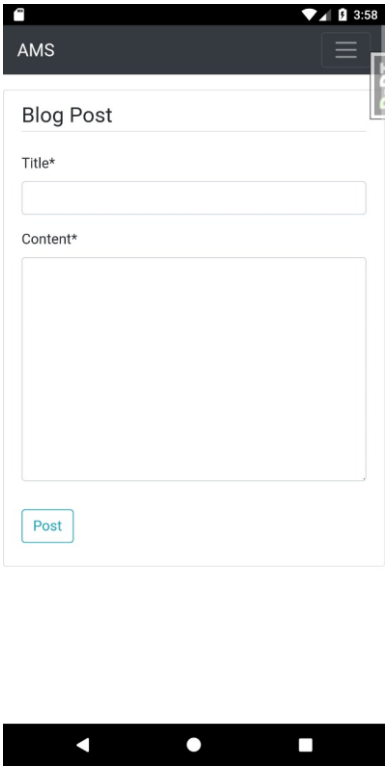

Fig 5.25 Create Blog

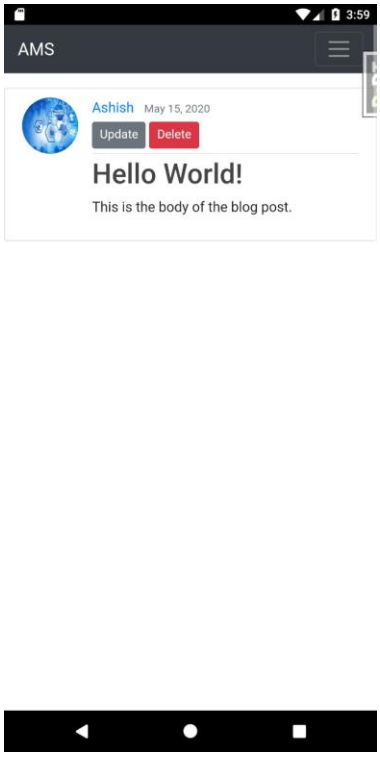

Fig 5.26 Blog Detail

5.2 CONCLUSIONS

The alumni management system that we propose is a centralized system for a university. Any alumnus of the university can register to the system. It removes the toil of hard work required in maintain records about the alumni. This system can be accessed from any website or mobile phone that is compatible. This system lets students currently enrolled in the university to connect with the alumni in an efficient manner. They can choose to connect to people based on their job profile, location, company working for etc.

5.3 CHALLENGES

- Updating and reminding users to update their profiles in timely manner
- Time commitment from leaders
- Informing new users about the events that are going to be held in future about which the notifications are already sent out.
- Real time chat and video conference among users and alumni.
- Providing a platform to ease the complete recruitment process online
- Resume filtering mechanism to process applications and find eligible candidates
- Getting users to attend the events
- Notifying attendee users on update or deletion of events

5.4 FUTURE WORK

The future work that can be done in the system:

1. Addition of chatting feature that allows users to communicate with each other.
2. Users can choose to hide or block some user in chatting.
3. The users can get a detailed summary of the alumni placed in big tech giants or with highest packages.
4. The results of recruitment events can be posted on the system.
5. The system can be configured to generate a resume by asking the users to fill up some columns or forms or based upon the profile.

6. They can connect via Linkedin profiles also. There can be an option provided for the users to communicate with each other using Linkedin to get referrals.
7. The system can be used to take online interviews by creating a video calling feature, that might help people to give referrals to the eligible candidates after conducting screening rounds.
8. We can give premium accounts for users so they can access advanced features like going live on system, webinars access etc.

REFERENCES

[1] N Rubens et al., "Alumni network analysis," in IEEE Global Engineering Education Conference (EDUCON), 2011, pp. 606- 611.

[2] Centralized Alumni Management System (CAMS) - A Prototype Proposal
IEEE, Dept. of CSE, Techno International Batanagar, Kolkata, India

[3] K Sivaprasad, "Making best use of alumni associations for holistic development of engineering institutes," in IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA), 2012, pp. 1-11.

[4] Almabay. [Online]. HYPERLINK "https://www.almabay.com/" https://www.almabay.com/

[5]AlmaConnect. [Online]. HYPERLINK "https://www.almaconnect.com/" https://www.almaconnect.com/

[6] Alumnet. [Online]. HYPERLINK "https://www.alumnet.co.za/" https://www.alumnet.co.za/

[7] almabase. [Online]. HYPERLINK "https://www.almabase.com/" https://www.almabase.com/

[8] J E Sharp, "Work in progress: alumni mentoring of engineers in a technical communication course," in 34th Annual Frontiers in Education, 2004, pp. F2F-11.

[9] J L Huff, W C Qakes, and C B Zoltowski, "Work in progress: Understanding professional competency formation in a service learning context from an alumni perspective," in 2012 Frontiers in Education Conference Proceedings, 2012, pp. 1-3.

[10] J E Sharp, "Work in Progress: Using Mock Telephone Interviews with Alumni to Teach Job Search Communication," in Frontiers in Education. 36th Annual Conference, 2006, pp. 7- 8.

[11] VIT Alumni System : https://www.vitaa.org/

APPENDIX

DJANGO-ENVIRON

django-environ allows you to use Twelve-factor methodology to configure your Django application with environment variables.

The idea of this package is to unify a lot of packages that make the same stuff: Take a string from os.environ, parse and cast it to some of useful python typed variables. To do that and to use the 12factor approach, some connection strings are expressed as url, so this package can parse it and return a urllib.parse.ParseResult. These strings from os.environ are loaded from a *.env* file and filled in os.environ with setdefault method, to avoid to overwrite the real environ. A similar approach is used in Two Scoops of Django book and explained in 12factor-django article.

Using django-environ you can stop to make a lot of unversioned settings_*.py to configure your app.

Use the command to install **django-environ**:

pip install django-environ

After installation create a ***.env*** file with contents such as:

DEBUG=TRUE

SECRET_KEY=*your_secret_key*

DATABASE_USER=*database_username*

DATABASE_PASSWORD=*database_password*

DATABASE_PORT=*database_port_number*

And use this with **settings.py**.

Example:

```
import environ

env = environ.Env(
    # set casting, default value
    SECRET_KEY=str
)
env_path = os.path.join(BASE_DIR, '.env')
```

```
# reading .env file
environ.Env.read_env('.env')

SECRET_KEY = env('SECRET_KEY')
DEBUG = env('DEBUG')
```

PSYCOPG2

Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection). It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make a large number of concurrent "INSERT"s or "UPDATE"s.

Psycopg 2 is mostly implemented in C as a libpq wrapper, resulting in being both efficient and secure. It features client-side and server-side cursors, asynchronous communication and notifications, COPY support. Many Python types are supported out-of-the-box and adapted to matching PostgreSQL data types; adaptation can be extended and customized thanks to a flexible objects adaptation system.

The current **psycopg2** implementation supports:

- Python version 2.7
- Python 3 versions from 3.4 to 3.8
- PostgreSQL server versions from 7.4 to 12
- PostgreSQL client library version from 9.1

Use the command to install **psycopg2**:

pip install pyscopg2

Changing the settings in **settings.py:**

By default, Django is configured to use SQLite as its backend. To use Postgres instead, "AMS/settings.py" needs to be updated:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.postgresql_psycopg2',
      'NAME': 'ams',  # name of the database
      'USER': env('DATABASE_USER'),  # create a user in postgres of this name
      'PASSWORD': env('DATABASE_PASSWORD'),  # set this as default password for the user created above
      'HOST': 'localhost',
      'PORT': env('DATABASE_PORT'),
   }
}
```

DJANGO-DEBUG-TOOLBAR

The Django Debug Toolbar is a configurable set of panels that display various debug information about the current request/response and when clicked, display more details about the panel's content.

To install **djangodebug toolbar** use the command:

pip install django-debug-toolbar

Next we need to configure project settings such as **AMS/settings.py** :

```
INSTALLED_APPS += [
'debug_toolbar',
]
MIDDLEWARE = [
#....
'debug_toolbar.middleware.DebugToolbarMiddleware',
#...
]
```

In some cases, it's also required to set **INTERNAL_IPS** in **settings.py**:

```
INTERNAL_IPS = [
    # ...
    '127.0.0.1'
    # ...
]
```

```
def show_toolbar(request):
    return True


DEBUG_TOOLBAR_CONFIG = {
    "SHOW_TOOLBAR_CALLBACK": show_toolbar,
}
```

Configuring the **urls.py** file to enable debug toolbar routing:

```
if settings.DEBUG and 'debug_toolbar' in settings.INSTALLED_APPS:
    import debug_toolbar
    urlpatterns += [
        url(r'^__debug__/', include(debug_toolbar.urls)),
    ]
```

Django-debug-toolbar in action as shown in figure A1.1 the SQL queries information can be obtained from the SQL header in the debug toolbar.

Similarly, we can obtain details about the GET,POST,PUT,DELETE requests etc. from the Request header of the Debug Toolbar as shown in figure A1.2.

Fig A1.1 SQL Queries information

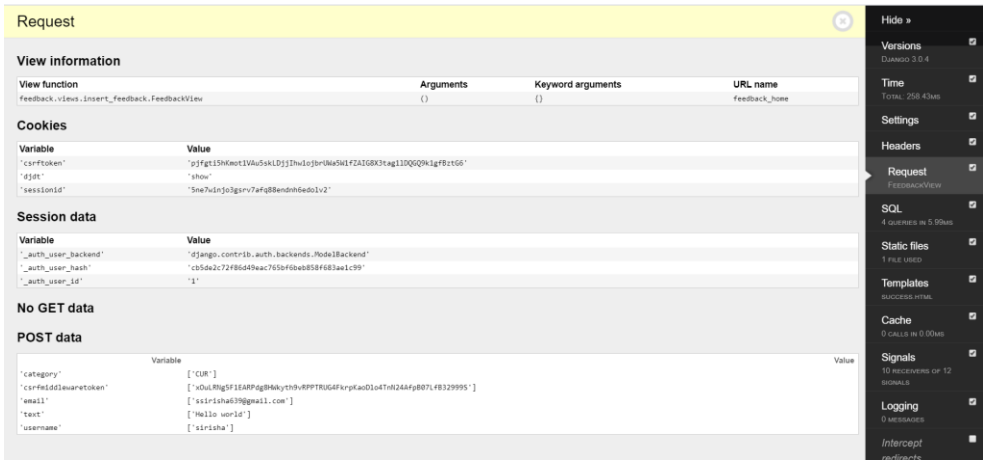

Fig A1.2 POST Data information

DJANGO REST FRAMEWORK

Django REST framework is a powerful and flexible toolkit for building Web APIs.

Rest framework requires Python 3.5 and above versions and Django 2.0 and above.

To install Django Rest Framework use the command:

pip install djangorestframework

Configuring rest_framework in **settings.py:**

```
INSTALLED_APPS = [
….
‘rest_framework’,
]
```

If you're intending to use the browsable API you'll probably also want to add REST framework's login and logout views. Add the following to your root **urls.py** file.

```
Urlpatterns = [
 ….
url(r’^api-auth/’,include(‘rest_framework.urls’))
]
```

DJANGO MATH FILTERS

Django-mathfilters is a Python 3 module that provides different simple math filters for Django. Django provides an add template filter, but no corresponding subtracting, multiplying or dividing filters.

To install Django Math Filters use the command:

pip install django-mathfilters

Configuring mathfilters in **settings.py:**

```
INSTALLED_APPS = [
….
```

'mathfilters',

]

If you wanted to try the operations in the templates then you have to use the below in **templates:**

{% load mathfilters %}

The Script provides the following:

1. sub – subtraction
2. mul – multiplication
3. div – division
4. intdiv – integer (floor) division
5. abs – absolute value
6. mod – modulo
7. addition – replacement for the add filter with support for float / decimal types

PILLOW

Pillow is a **Python** Imaging Library (PIL), which adds support for opening, manipulating, and saving images. The current version identifies and reads a large number of formats. Write support is intentionally restricted to the most commonly **used** interchange and presentation formats.

To install Pillow use the command:

pip install Pillow

To use Pillow for images simply import pillow in the specific module like below:

from PIL import Image

BIBLIOGRAPHY

1. Django documentation : https://docs.djangoproject.com/en/3.0/
2. React Js documentation : https://reactjs.org/docs/getting-started.html
3. Django Rest Framework documentation :https://www.django-rest-framework.org/
4. Django debug toolbar documentation : https://django-debug-toolbar.readthedocs.io/en/latest/
5. Psycopg2 documentation : https://pypi.org/project/psycopg2/
6. Django environ documentation : https://pypi.org/project/django-environ/
7. Mailgun API documentation : https://documentation.mailgun.com/en/latest/api-intro.html