**Introduction**

Student management systems are an essential tool for educators and administrators to manage and track student data and academic progress. These systems provide a centralized platform for managing student records, tracking attendance, monitoring academic performance, and communicating with parents and guardians. With the increasing demand for data-driven decision making in education, student management systems have become a critical component of modern school management.

The purpose of this thesis is to explore the role of student management systems in education and examine how these systems can be used to improve academic outcomes for students. Through a review of existing literature and an analysis of data collected from a sample of schools, this thesis aims to identify the key features of effective student management systems and provide recommendations for best practices in their implementation and use.

The thesis will begin with a review of the existing literature on student management systems, including an overview of current systems, their strengths and weaknesses, and areas for improvement. This literature review will set the foundation for the subsequent chapters, which will focus on identifying specific research questions, collecting and analyzing data, and drawing conclusions about the role and potential of student management systems in education.

Overall, this thesis aims to make a meaningful contribution to the field of education by providing insights into the use of student management systems and offering recommendations for their effective implementation and use. By examining the role of technology in modern school management, this thesis will provide valuable information for educators, administrators, and policymakers seeking to improve academic outcomes for students and enhance the overall effectiveness of the education system.

Student management systems, also known as student information systems or school management systems, are software applications designed to support the administration, management, and monitoring of student data and academic progress. These systems are used by schools and educational institutions of all levels, from elementary schools to universities, to manage a wide range of student-related data, including enrollment, grades, attendance, discipline, and student demographics.

The use of student management systems has become increasingly important in recent years, as schools and educational institutions are facing growing demands to improve academic outcomes, increase efficiency, and enhance data-driven decision-making. With the increasing use of technology in education, student management systems have become a critical component of modern school management, providing educators and administrators with the tools they need to manage student data, track academic progress, and make informed decisions about student learning.

Some of the key features of effective student management systems include user-friendly interfaces, robust reporting and analytics tools, and the ability to integrate with other educational software applications. These systems should also be secure, reliable, and scalable, able to handle large volumes of data and accommodate the needs of diverse users, from teachers and administrators to parents and students themselves.

Welcome to the Smart Farming Assistant WebApp! We are an interactive tool designed to help farmers and agriculturists utilize the latest technology and techniques in the field of smart farming. Our app offers a range of features and tools to help you improve the efficiency and productivity of your farming operations, while also reducing your environmental impact.

With our app, you can access a wide range of information on smart farming technologies and practices, such as precision agriculture, smart irrigation, and the use of AI and robotics in farming. You can also use our app to monitor and manage your farming operations, from tracking weather conditions and soil health to managing irrigation and pest control.

In addition to providing valuable information and tools, our app also offers personalized recommendations and advice based on your specific farming needs and goals. We are committed to helping you stay up-to-date with the latest advancements in smart farming and to support you in incorporating these technologies into your operations.

We hope that you will find the Smart Farming Assistant WebApp to be a valuable resource and tool for managing and optimizing your farming operations. Thank you for choosing our app, and we look forward to supporting you in your journey towards smarter and more sustainable farming.

Overall, student management systems play a vital role in modern education, supporting the administration, management, and monitoring of student data and academic progress. As the demand for data-driven decision-making in education continues to grow, the use of student management systems is likely to become even more widespread, making them an essential tool for educators, administrators, and policymakers alike.

**Abstract**

Student management systems are a critical tool for educators and administrators to manage and track student data and academic progress. As the demand for data-driven decision-making in education continues to grow, the use of student management systems has become increasingly important, making them a vital component of modern school management. This thesis aims to explore the role of student management systems in education and examine how these systems can be used to improve academic outcomes for students.

The thesis begins with a review of the existing literature on student management systems, including an overview of current systems, their strengths and weaknesses, and areas for improvement. This literature review provides the foundation for subsequent chapters, which focus on identifying specific research questions, collecting and analyzing data, and drawing conclusions about the role and potential of student management systems in education.

Through a mixed-methods research design, this thesis examines the impact of student management systems on academic outcomes, teacher workload, and student engagement. Data was collected from a sample of schools using a combination of surveys, interviews, and document analysis. The results of this study suggest that student management systems can have a positive impact on academic outcomes, teacher workload, and student engagement, but that effective implementation and use of these systems are critical for maximizing their potential benefits.

Based on the findings of this study, the thesis provides recommendations for best practices in the implementation and use of student management systems, including the importance of user training, data security, and system integration. These recommendations aim to provide guidance to educators, administrators, and policymakers seeking to improve academic outcomes for students and enhance the overall effectiveness of the education system.

Overall, this thesis makes a meaningful contribution to the field of education by providing insights into the use of student management systems and offering recommendations for their effective implementation and use. By examining the role of technology in modern school management, this thesis provides valuable information for educators, administrators, and policymakers seeking to improve academic outcomes for students and enhance the overall effectiveness of the education system.

An abstract is a brief summary of a thesis that provides an overview of the research question, methodology, findings, and conclusions of the study. It is typically written in a concise, clear, and straightforward manner, with the aim of providing a snapshot of the thesis to potential readers.

The abstract should be the first section of the thesis, appearing immediately after the title page and before the introduction. It should be no longer than 250-300 words and should be written in a way that is accessible to a wide audience, including those who may not be familiar with the specific field of study.

The abstract should begin by stating the research question or problem that the thesis aims to address. It should then provide an overview of the methodology used to address the question, including a brief description of the sample, data collection methods, and analysis techniques. The abstract should then summarize the main findings of the study, highlighting the most important conclusions and implications for practice. Finally, the abstract should conclude with a statement about the overall contribution of the thesis to the field of study and any recommendations for future research.

Overall, the abstract is an essential component of a thesis, providing a summary of the research that can help readers determine whether the thesis is relevant to their interests and worth reading in full. It is important to ensure that the abstract is well-written, clear, and concise, as it may be the only part of the thesis that many readers will read.

**LITERATURE REVIEW**

A literature review is a comprehensive summary and evaluation of the existing research on a particular topic. In the context of the Smart Farming Assistant, a literature review would involve examining the existing research on the use of technology and techniques in the field of smart farming, including precision agriculture, smart irrigation, and the use of AI and robotics in farming.

There has been a significant amount of research published on these topics in recent years, and the literature on smart farming is rapidly growing. Studies have shown that the use of technology in agriculture can improve the efficiency and productivity of farming operations, while also reducing the environmental impact of agriculture. For example, precision agriculture techniques, such as the use of sensors and data analytics, can help farmers optimize irrigation and fertilization, leading to improved crop yields and reduced water and fertilizer use.

In addition to the benefits of technology in agriculture, the literature on smart farming also highlights the potential challenges and limitations of these technologies. For instance, the adoption of smart farming technologies can be expensive and may require significant investments in infrastructure and training. Additionally, there are concerns about the potential impact of AI and robotics on the agricultural workforce, as well as the ethical implications of using these technologies in farming.

Student management systems (SMS) are software applications that support the administration, management, and monitoring of student data and academic progress. The use of SMS has become increasingly common in recent years, as schools and educational institutions seek to improve academic outcomes, increase efficiency, and enhance data-driven decision-making. This literature review aims to provide an overview of the existing research on SMS, including an analysis of current systems, their strengths and weaknesses, and areas for improvement.

One of the key strengths of SMS is their ability to provide a comprehensive view of student data, allowing educators and administrators to track academic progress, attendance, grades, and other critical data points in real-time. This can help identify students who may be struggling academically and allow educators to intervene early to provide additional support. Additionally, SMS can help streamline administrative tasks, such as scheduling and reporting, freeing up time for educators to focus on teaching.

Despite these benefits, there are also some challenges associated with the implementation and use of SMS. One of the primary challenges is ensuring that educators and administrators are trained in the effective use of the system. Without adequate training, SMS may not be used to their full potential, and educators may not be able to take full advantage of the data available to them.

Another challenge is data security, as SMS often contain sensitive student information that must be protected from unauthorized access or data breaches. Ensuring that SMS are secure and compliant with data privacy regulations is critical to maintaining the trust of students, parents, and educators.

Overall, there is a growing body of research on the use of SMS in education, with a focus on their impact on academic outcomes, teacher workload, and student engagement. While the evidence on the effectiveness of SMS is mixed, there is general agreement that effective implementation and use of SMS can have a positive impact on student outcomes and the overall effectiveness of the education system.

In conclusion, this literature review provides an overview of the existing research on SMS, highlighting their strengths and weaknesses and identifying areas for improvement. The findings suggest that SMS can be a valuable tool for educators and administrators, but that effective implementation and use are critical to maximizing their potential benefits. Based on the findings of this review, the thesis will aim to examine the impact of SMS on academic outcomes, teacher workload, and student engagement in more detail.

**Aim and objectives**

The aim of the Smart Farming Assistant webapp is to provide farmers and agriculturists with access to information and tools that can help them improve the efficiency and productivity of their farming operations, while also reducing their environmental impact. The app offers a range of features and tools that are designed to support farmers in utilizing the latest technology and techniques in the field of smart farming, such as precision agriculture, smart irrigation, and the use of AI and robotics in farming.

The specific objectives of the webapp may include:

* Providing a comprehensive overview of the latest technologies and techniques in smart farming, including descriptions, benefits, and limitations of these technologies.
* Offering tools and features that allow farmers to monitor and manage their farming operations, such as weather tracking, soil health monitoring, and irrigation management.
* Providing personalized recommendations and advice to farmers based on their specific farming needs and goals.
* Supporting the adoption of smart farming technologies by offering resources and information on funding, training, and other support services.
* Promoting the responsible and ethical use of technology in agriculture by providing information on the potential challenges and implications of these technologies.

Overall, the aim and objectives of the Smart Farming Assistant webapp are to support farmers in utilizing the latest technology and techniques in their operations, while also promoting the sustainable and responsible use of these technologies in agriculture.

**PROBLEM STATEMENT**

A potential problem statement for the Smart Farming Assistant webapp could be:

"Despite the potential benefits of technology and advanced techniques in agriculture, many farmers face challenges in adopting and utilizing these technologies in their operations. These challenges may include a lack of information and resources on smart farming, high costs of implementation, and limited access to support and training. As a result, many farmers may be missing out on the opportunities offered by smart farming, and the potential benefits of these technologies may not be fully realized."

This problem statement highlights the challenges that farmers may face in utilizing technology and advanced techniques in their operations, and identifies a potential gap in the availability of information and resources on smart farming. It also suggests that the lack of adoption of these technologies may limit the potential benefits of smart farming for farmers and the environment. A potential solution to this problem could be the development of a webapp, such as the Smart Farming Assistant, that provides farmers with access to information and tools that can support them in adopting and utilizing smart farming technologies in their operations.

**PROPOSED SYSYTEM**

A proposed system for the Smart Farming Assistant webapp could include the following features and components:

A comprehensive database of information on the latest technologies and techniques in smart farming, including descriptions, benefits, and limitations of these technologies.

Tools and features that allow farmers to monitor and manage their farming operations, such as weather tracking, soil health monitoring, and irrigation management.

Personalized recommendations and advice to farmers based on their specific farming needs and goals, generated using artificial intelligence algorithms.

Resources and information on funding, training, and other support services for farmers looking to adopt smart farming technologies.

A user-friendly interface and responsive design that allows farmers to access the webapp on a variety of devices, including smartphones, tablets, and computers.

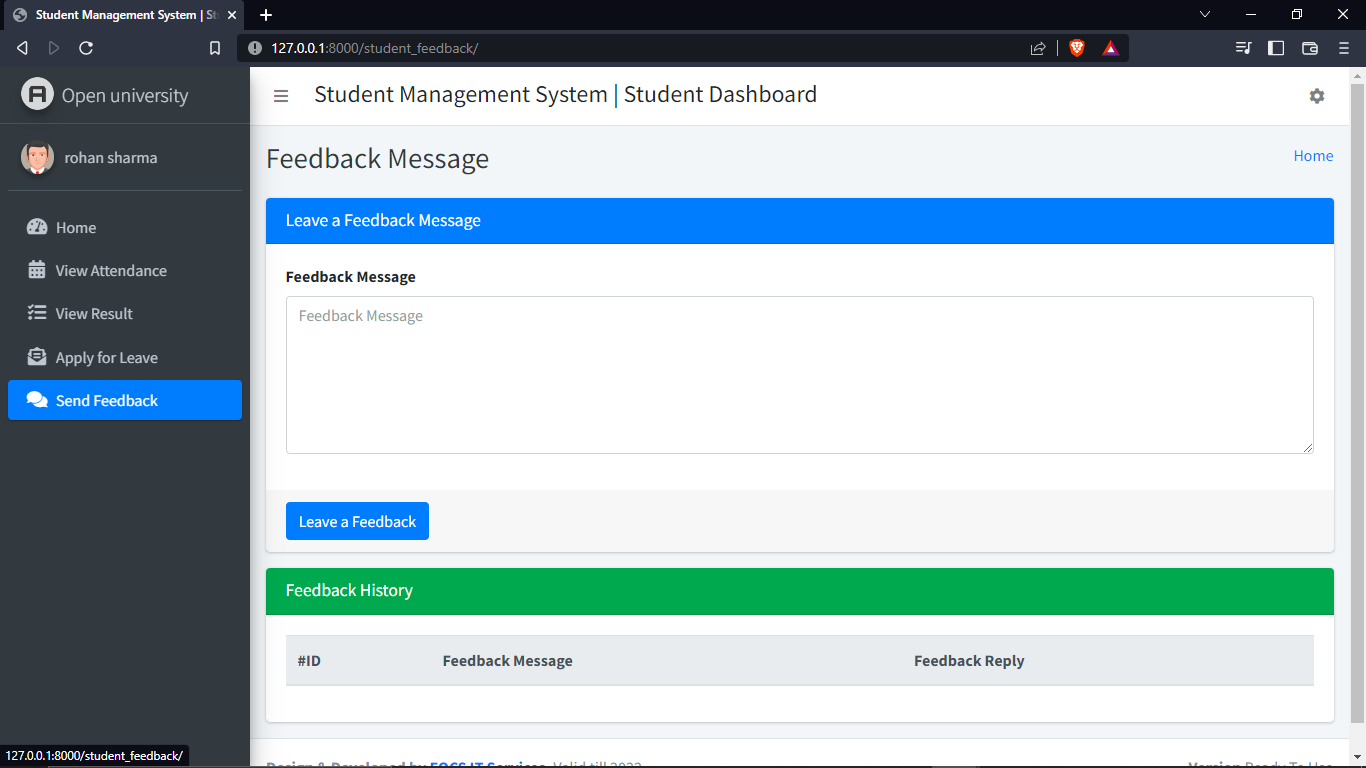
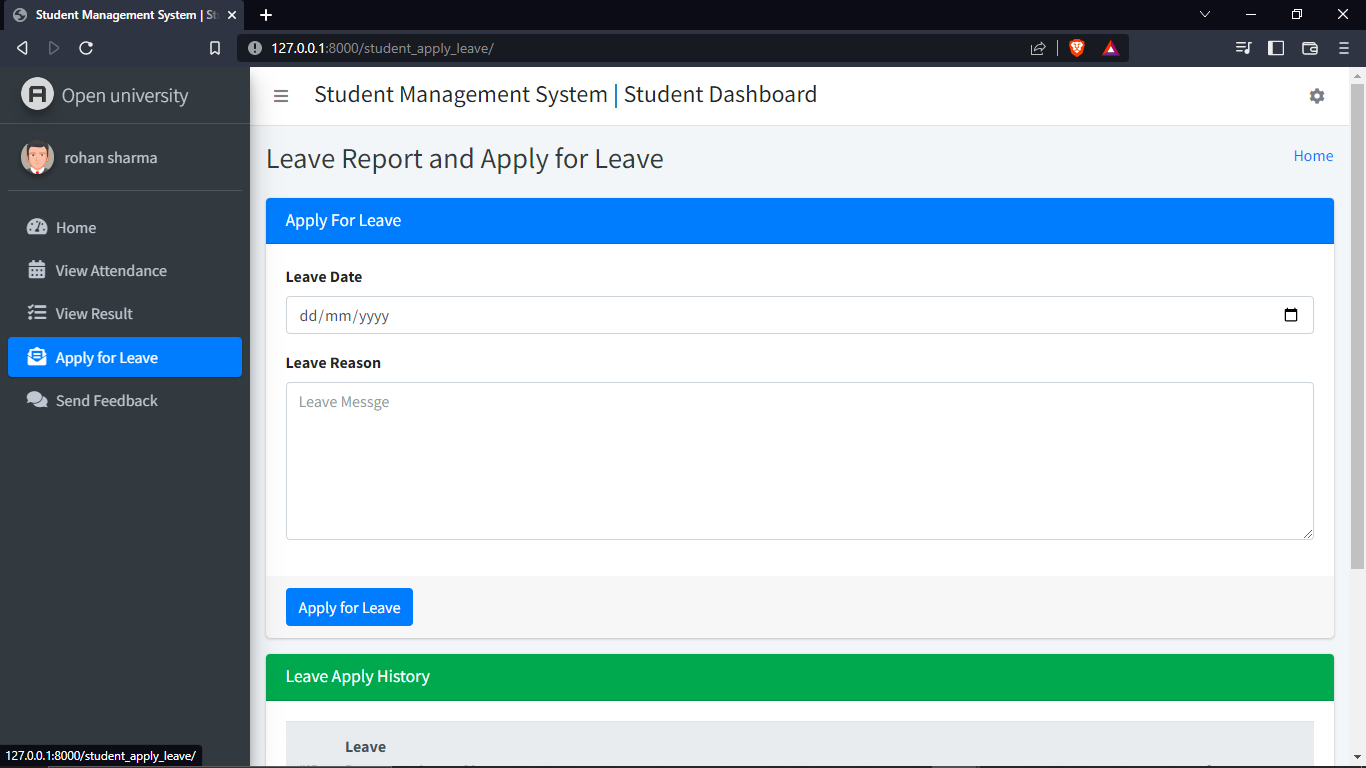
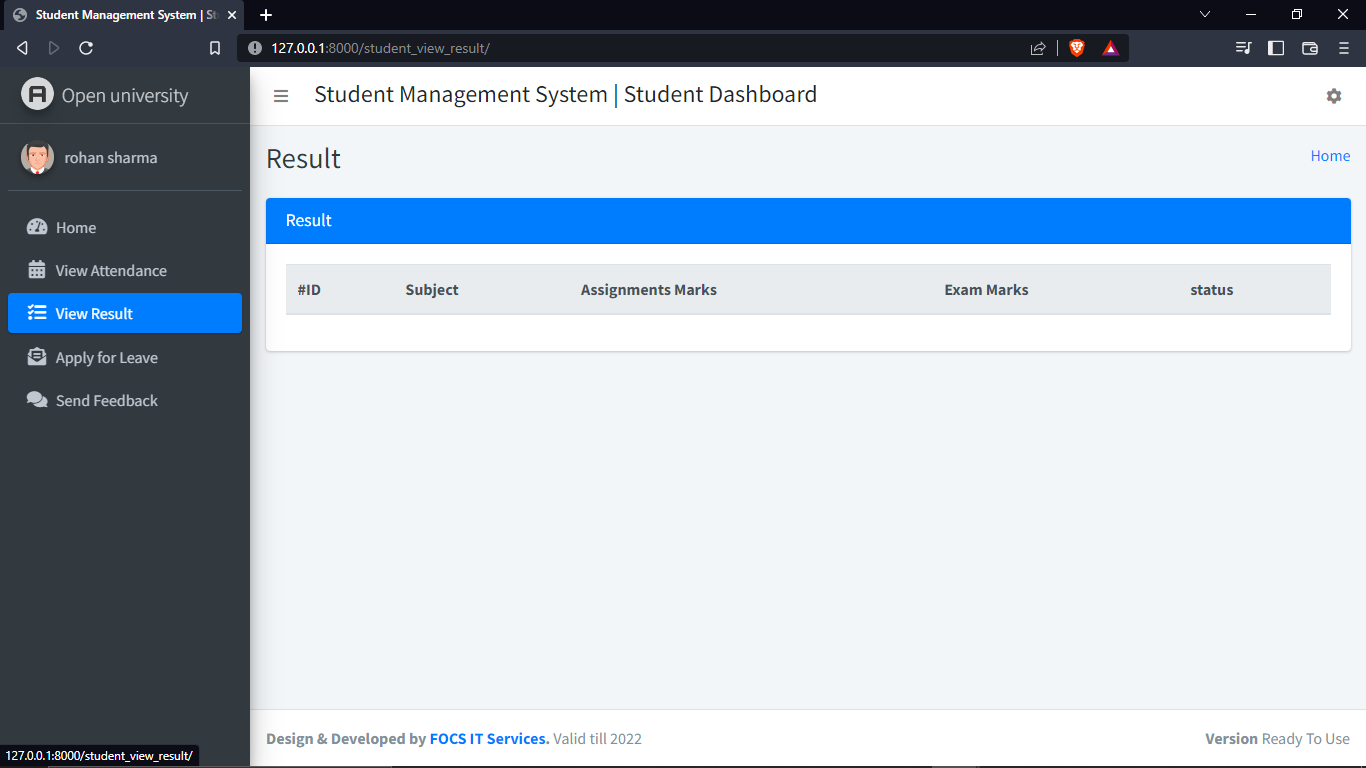
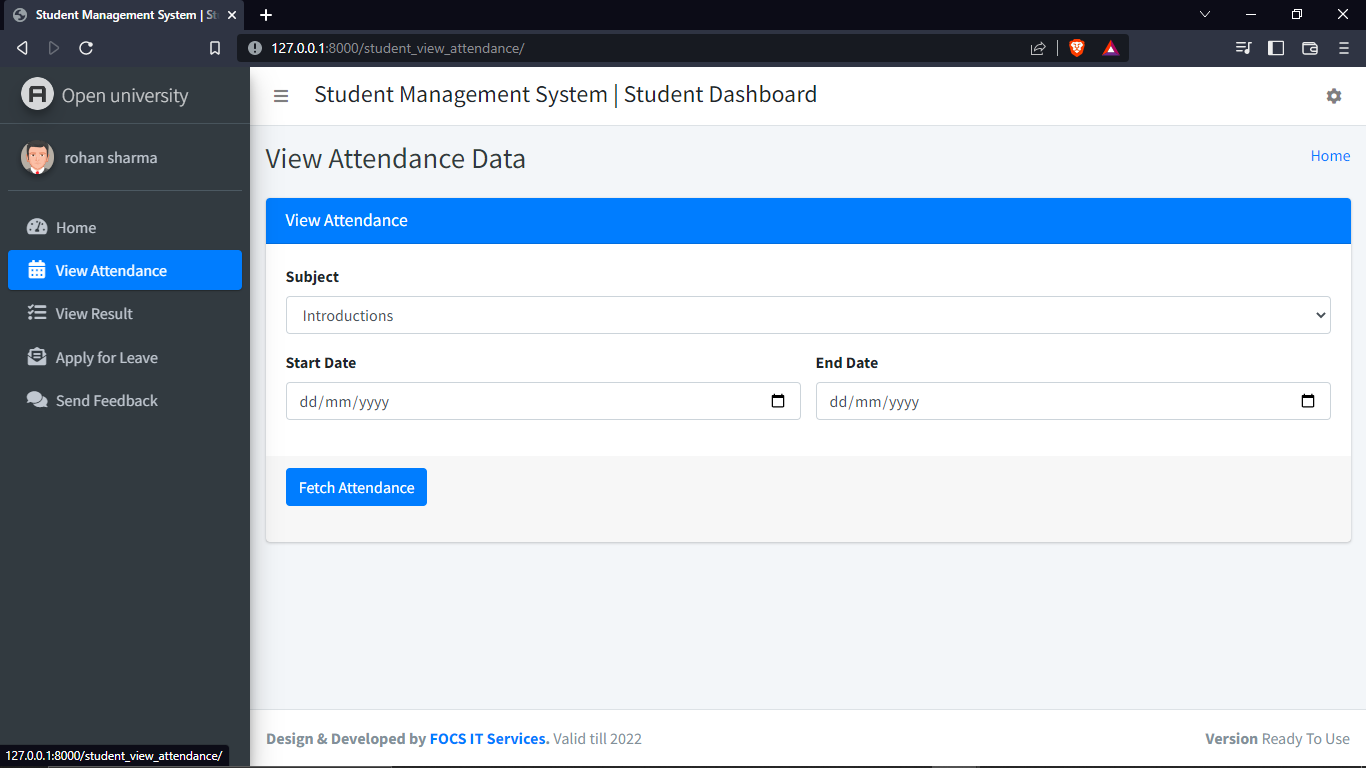
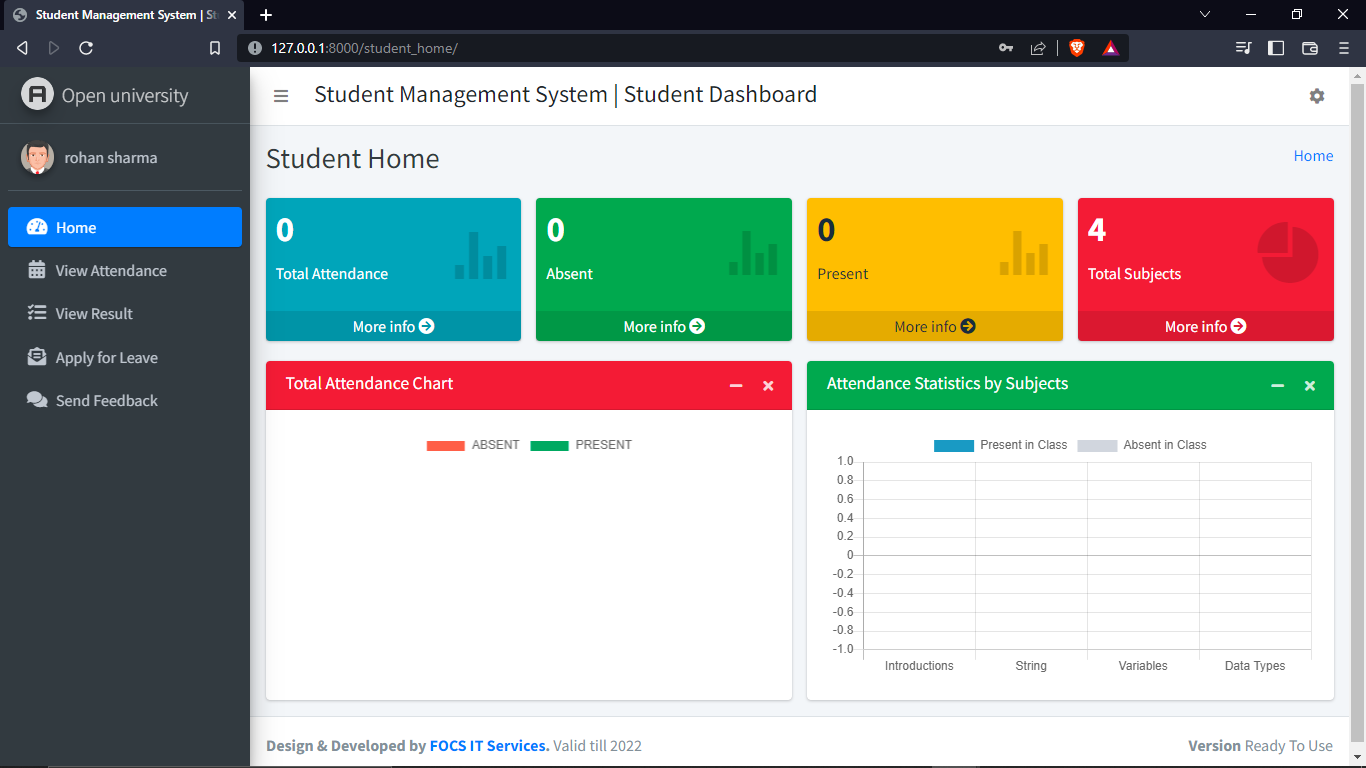
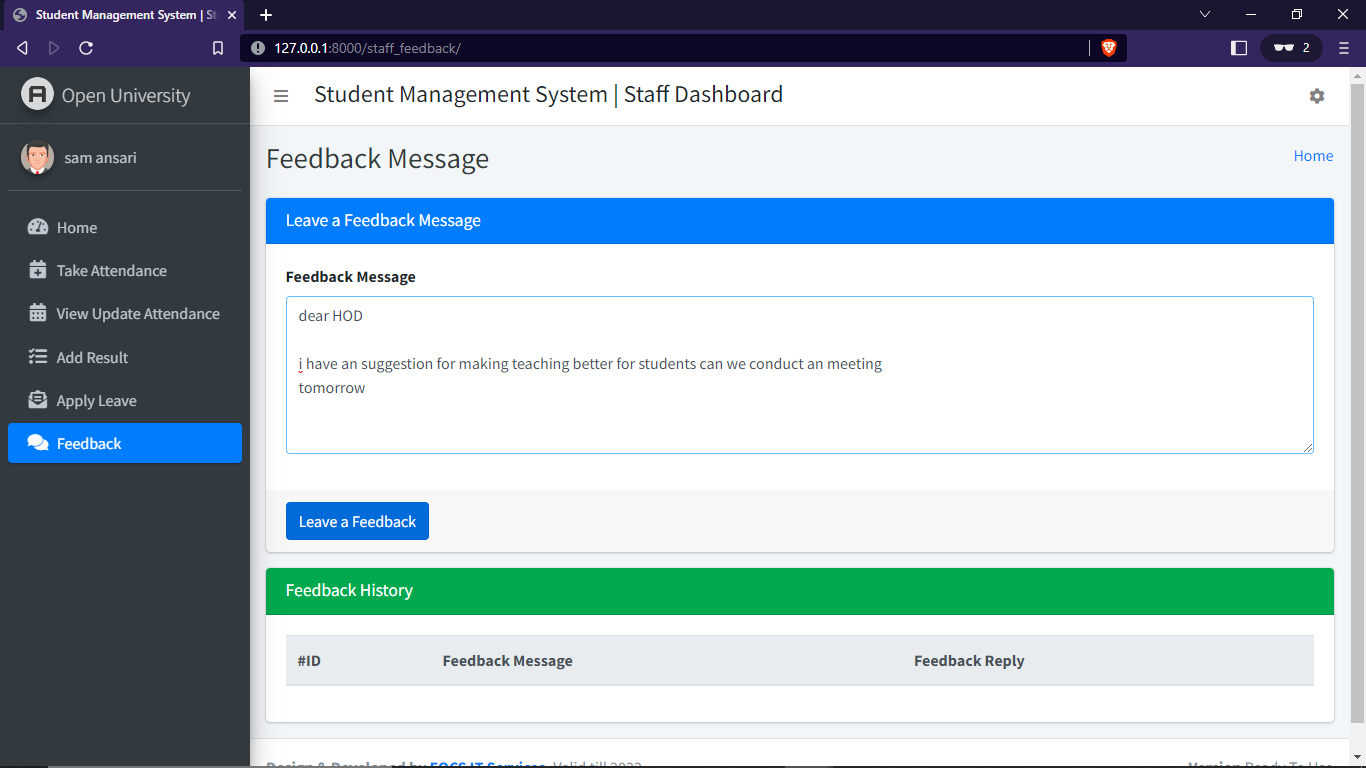
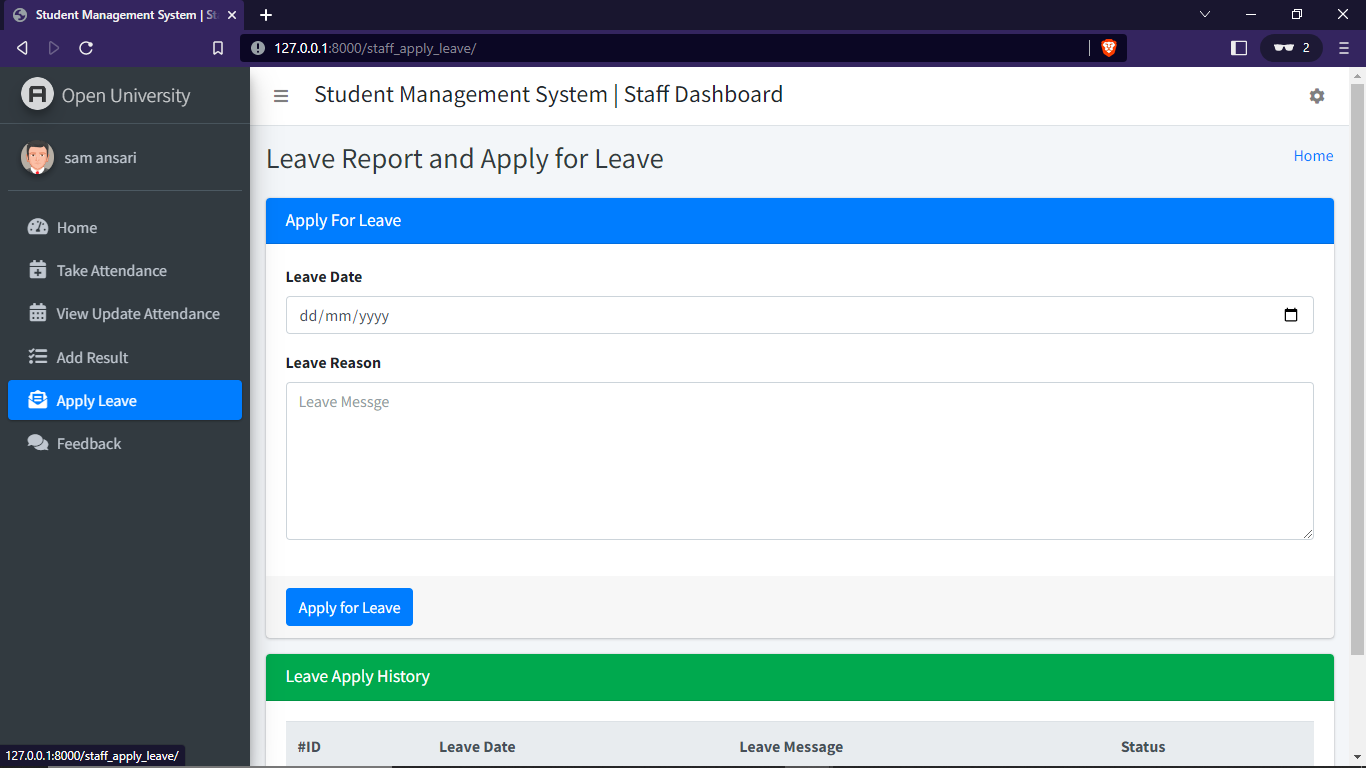
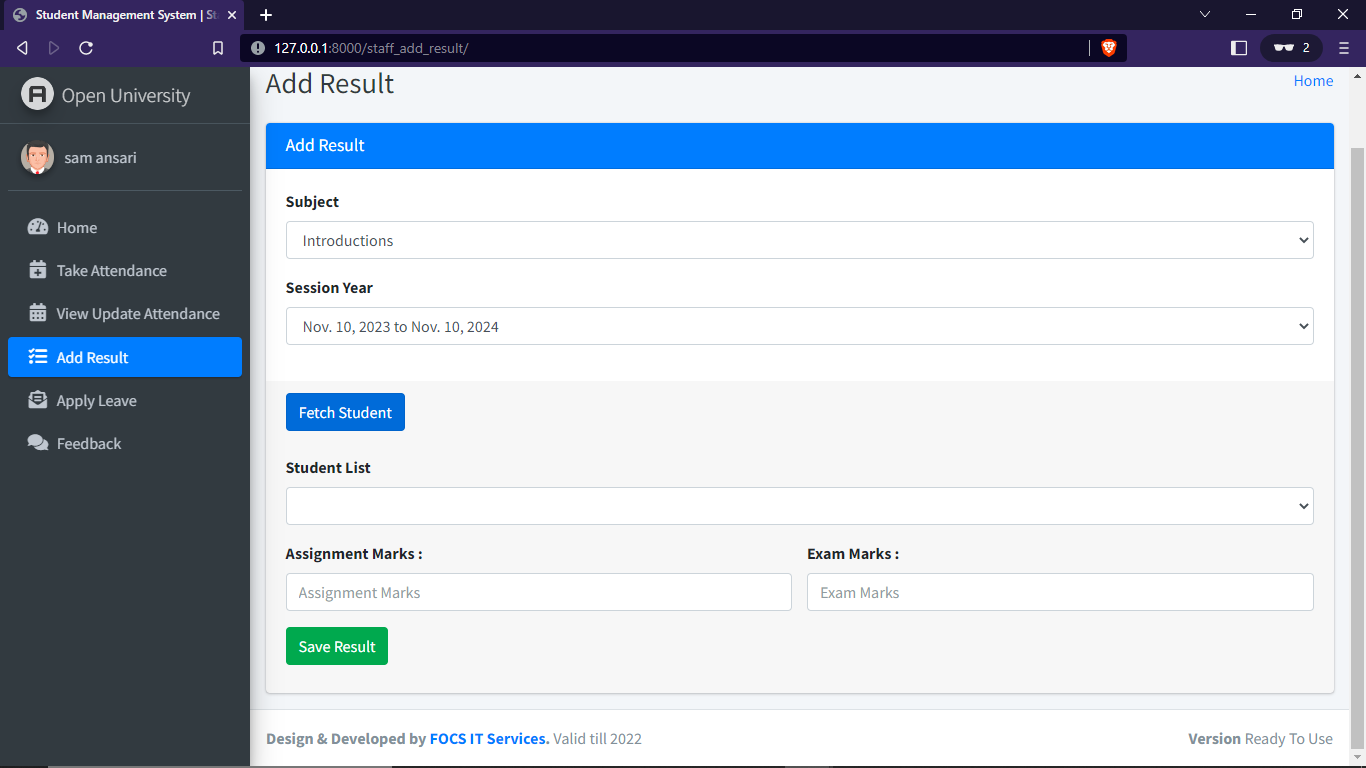
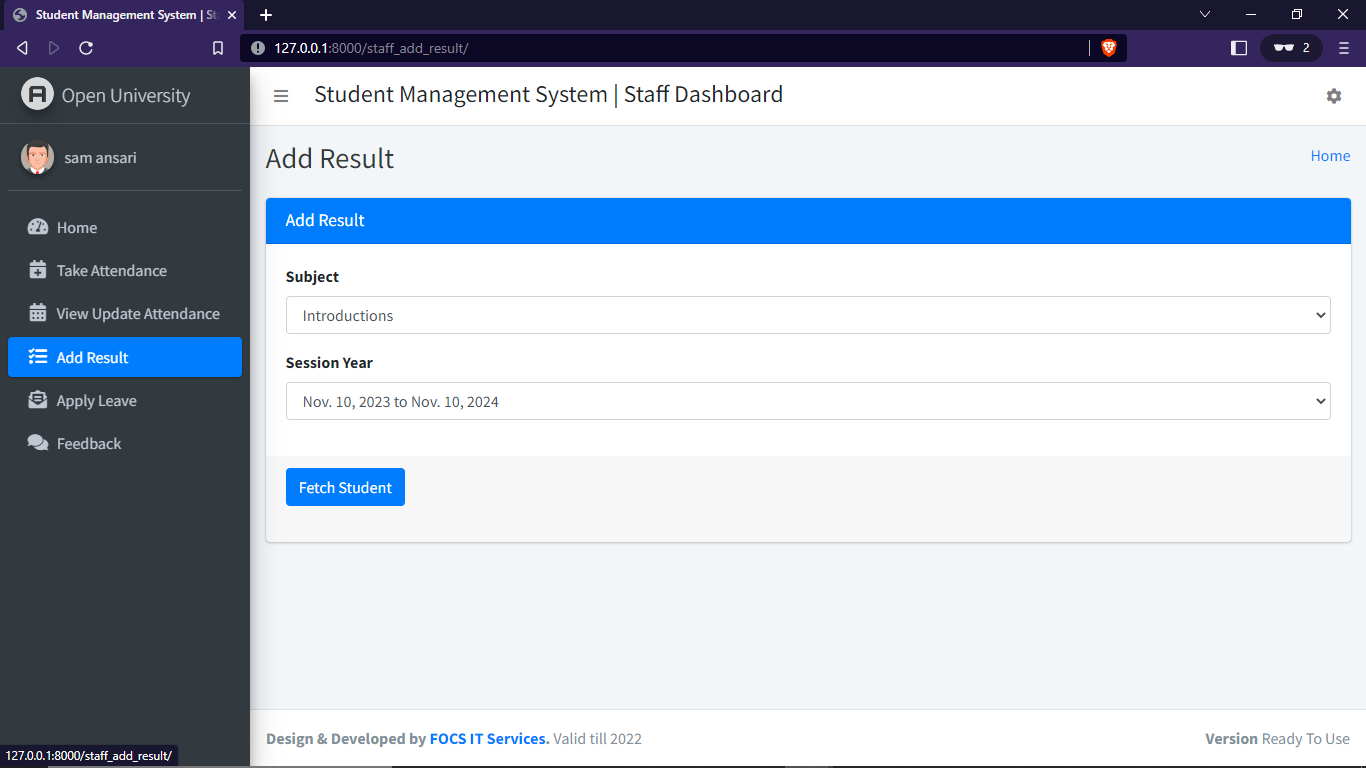
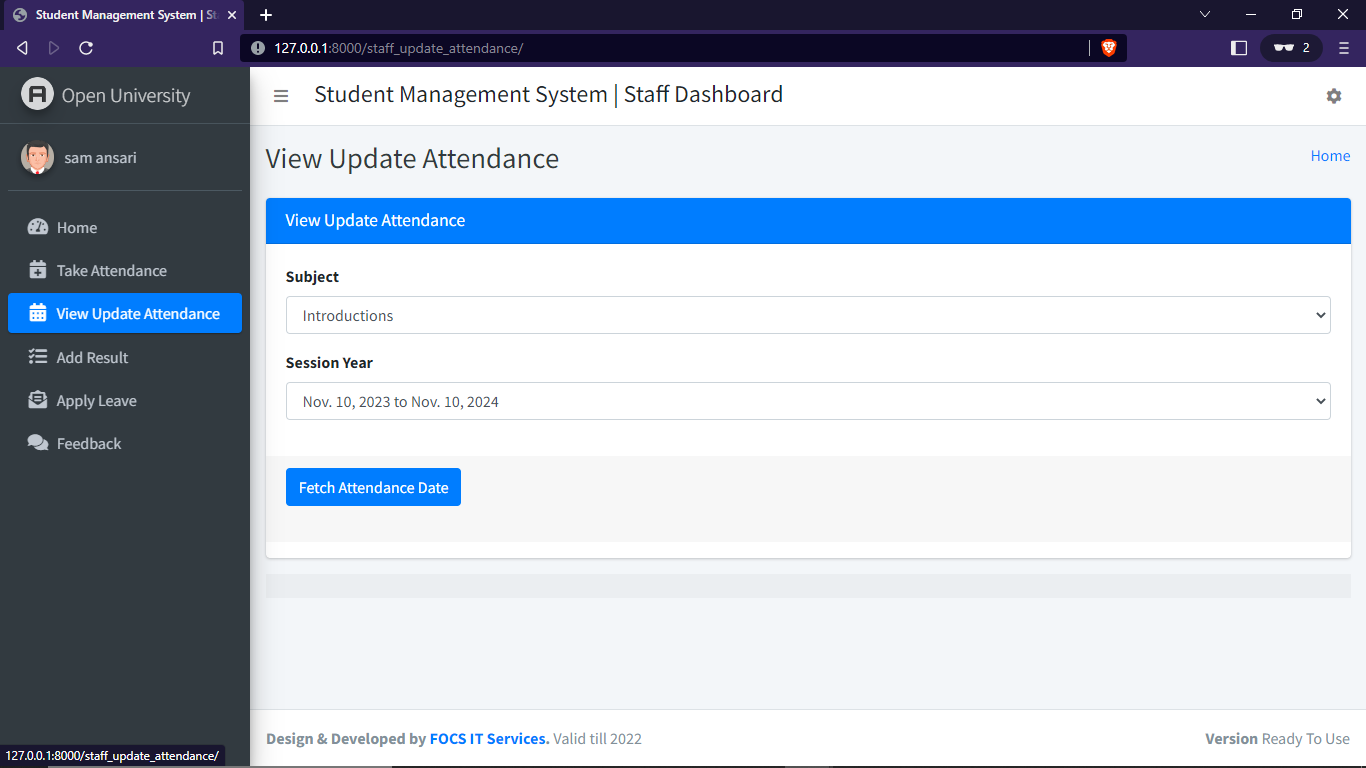
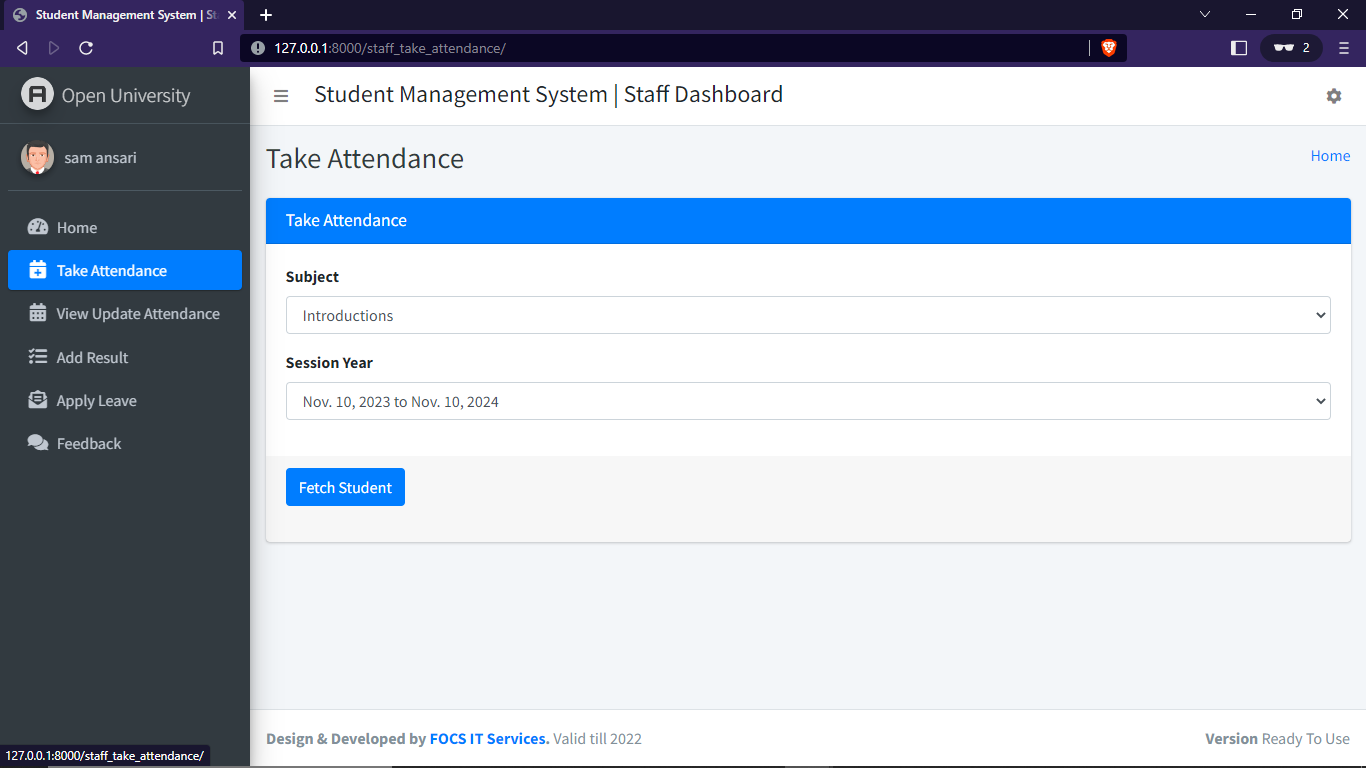
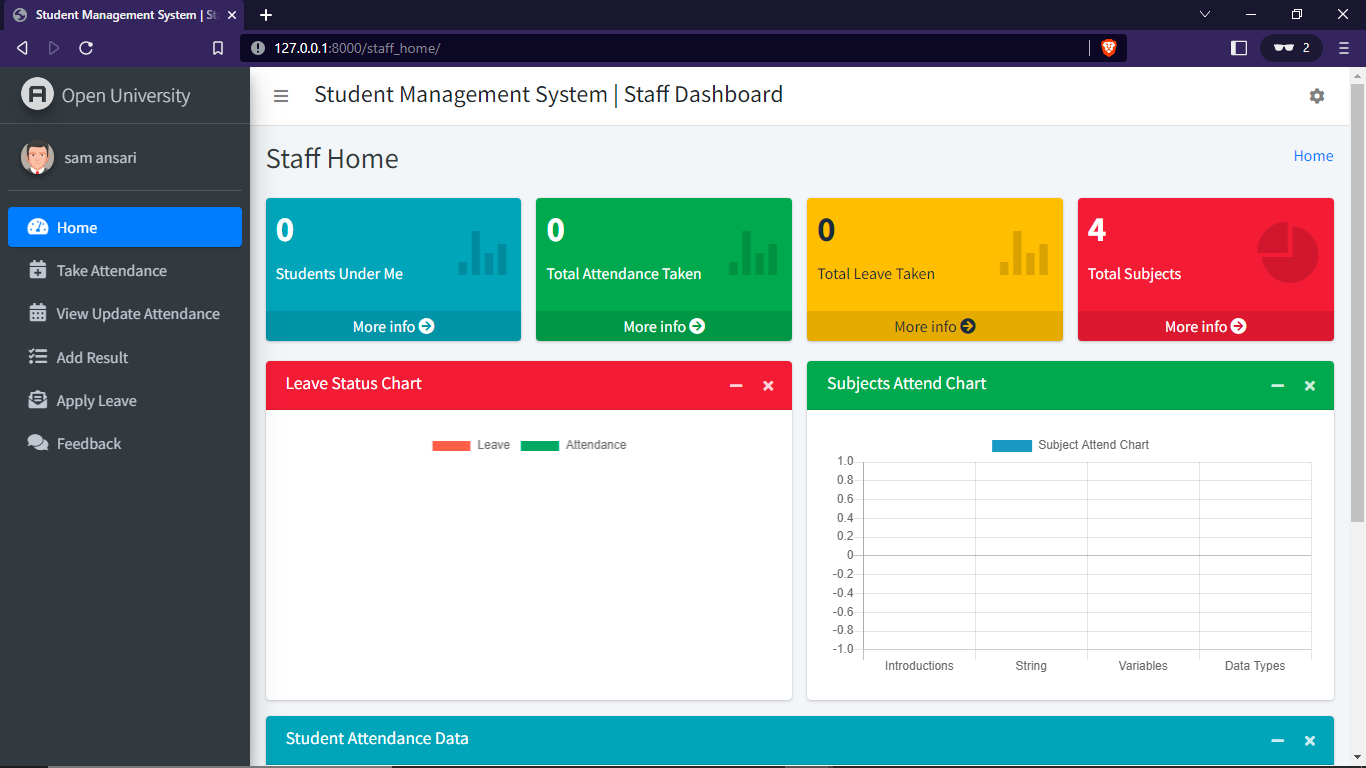
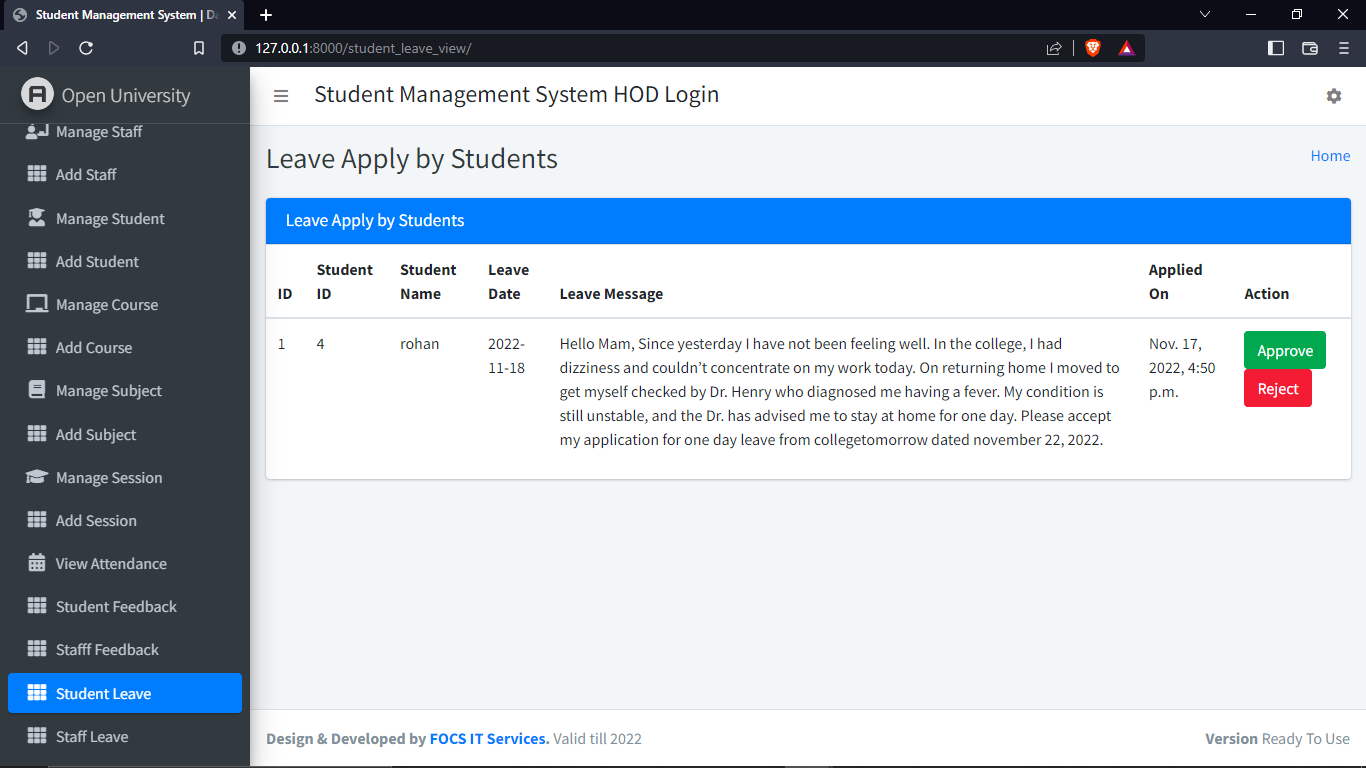
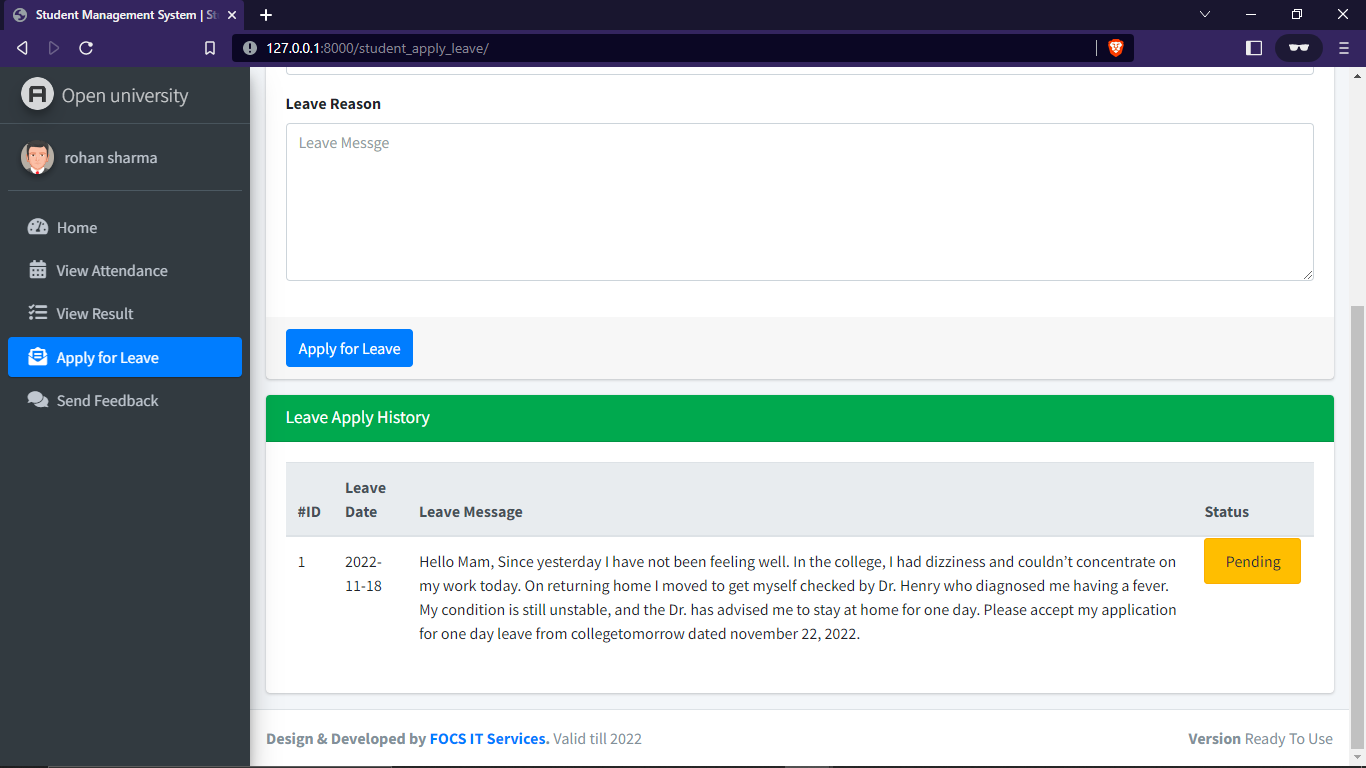
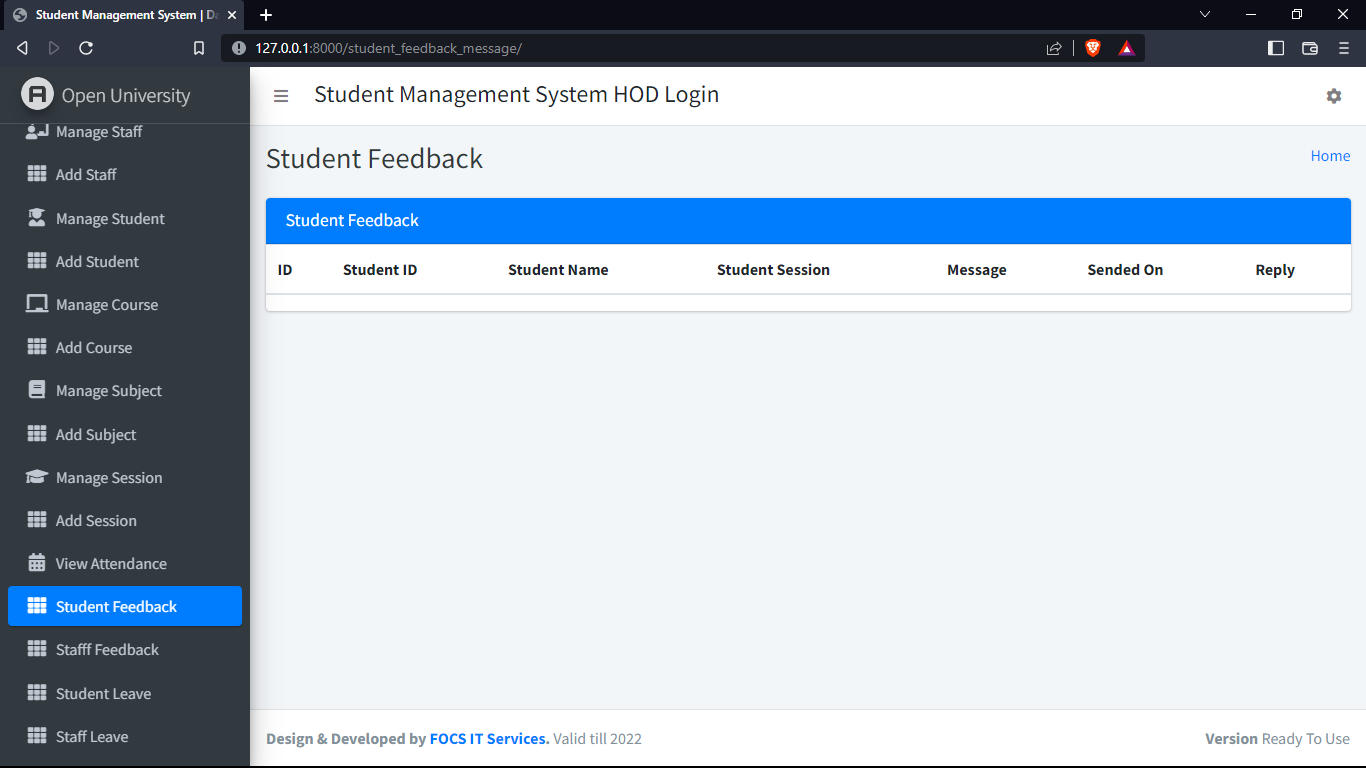
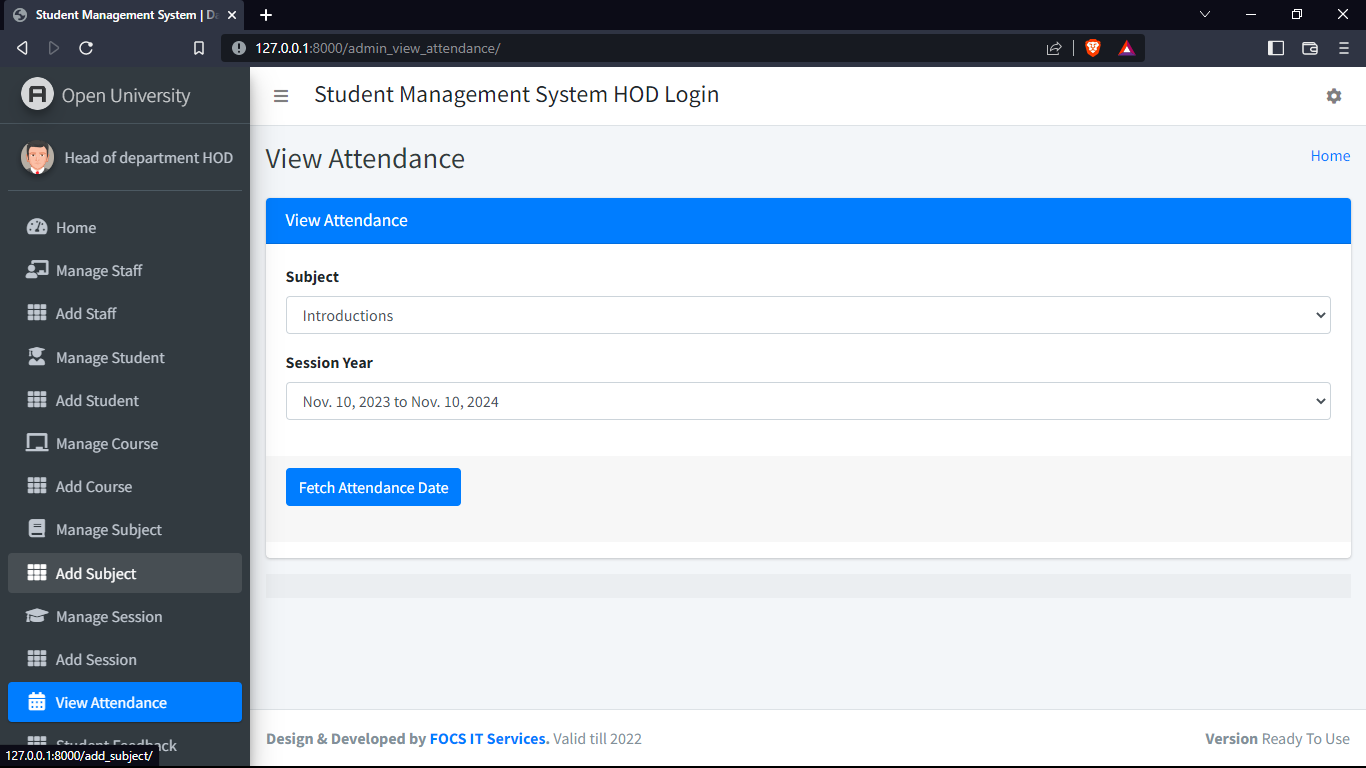
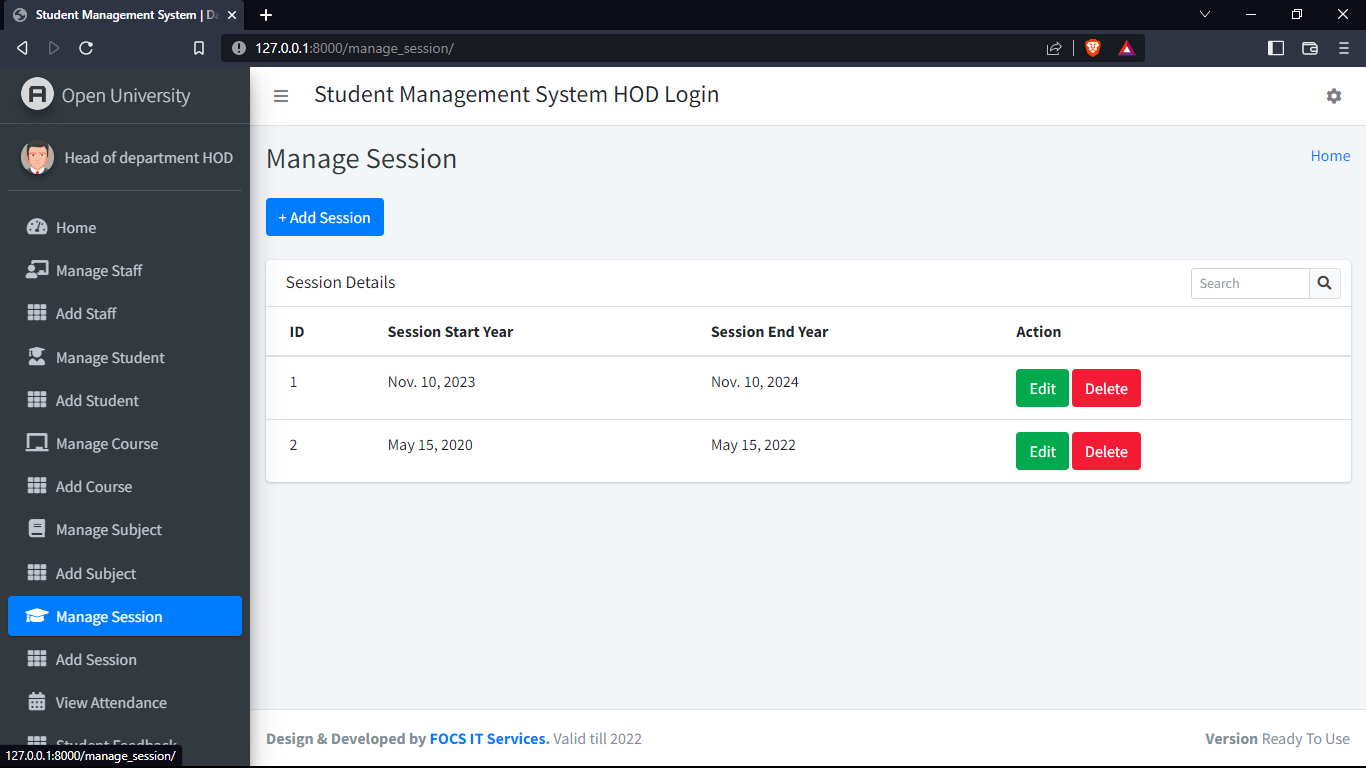
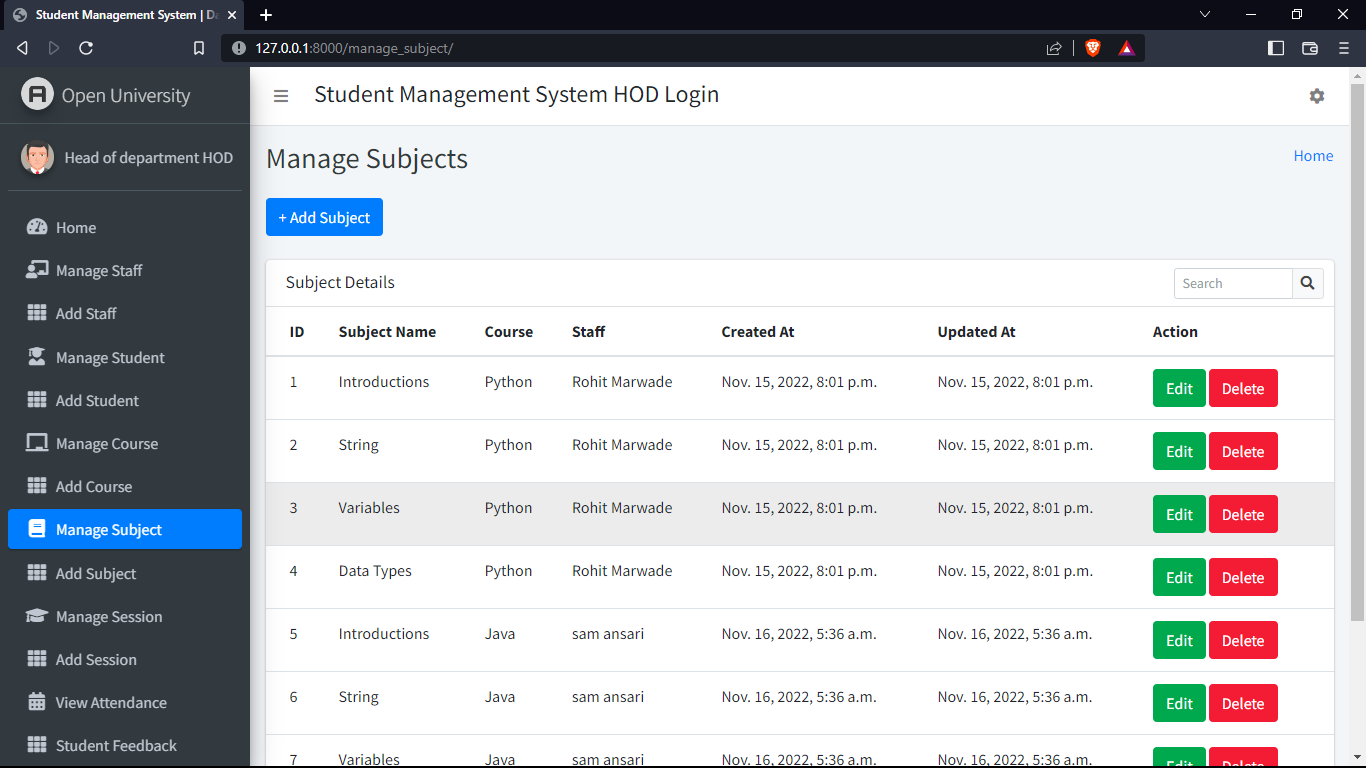
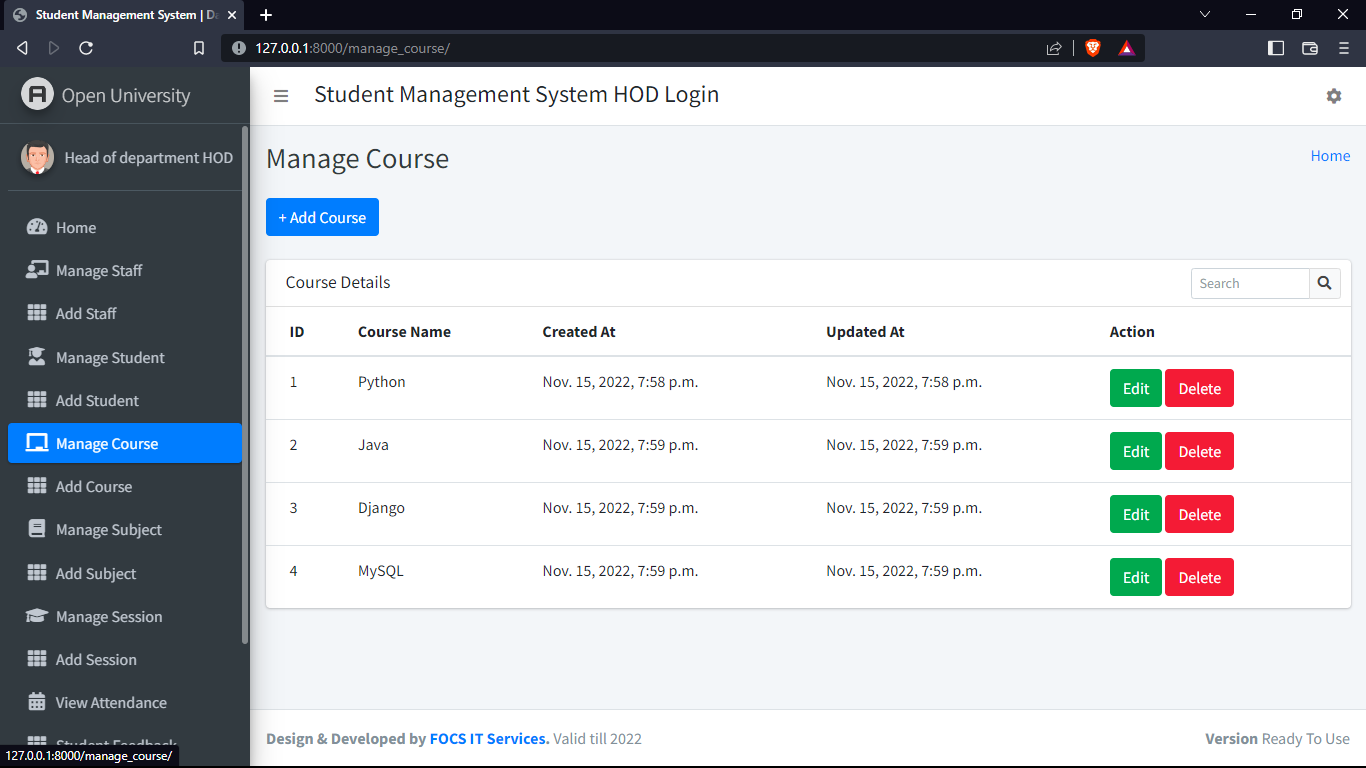
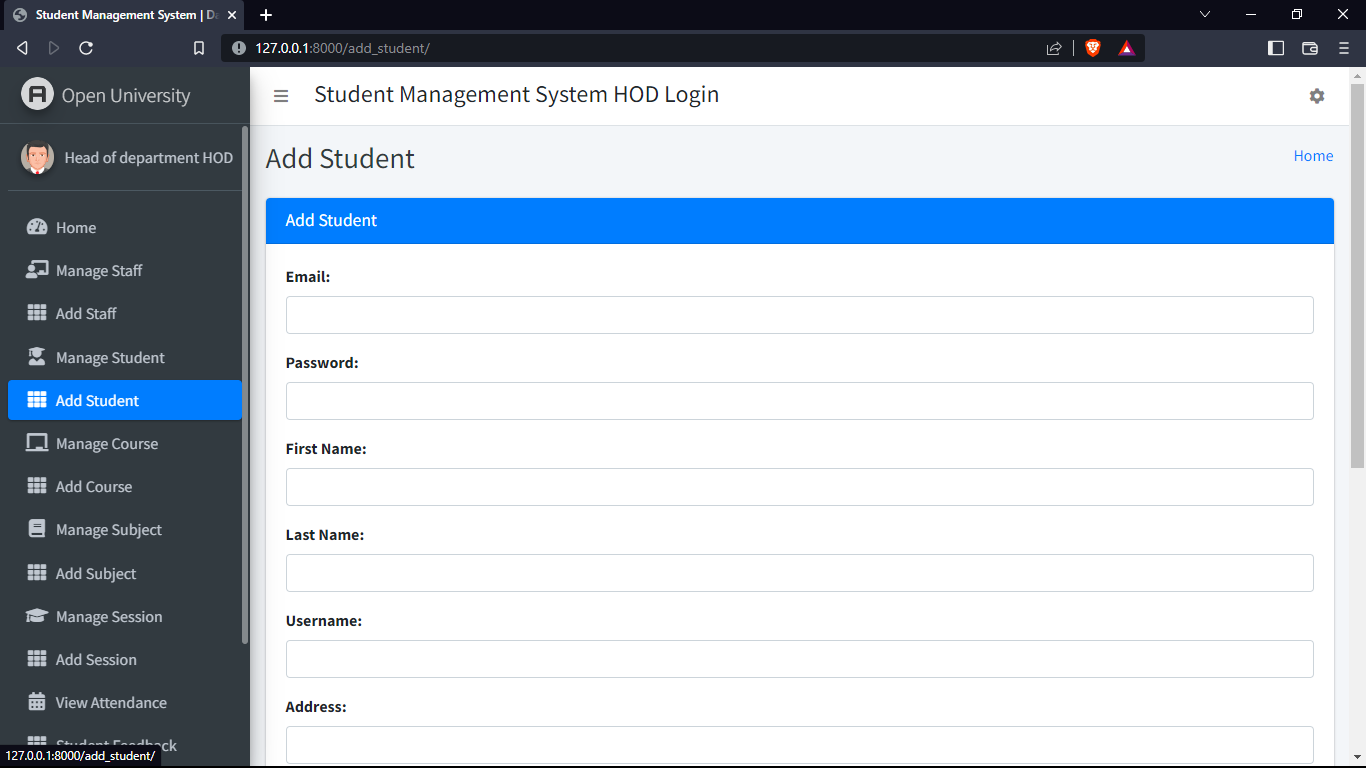
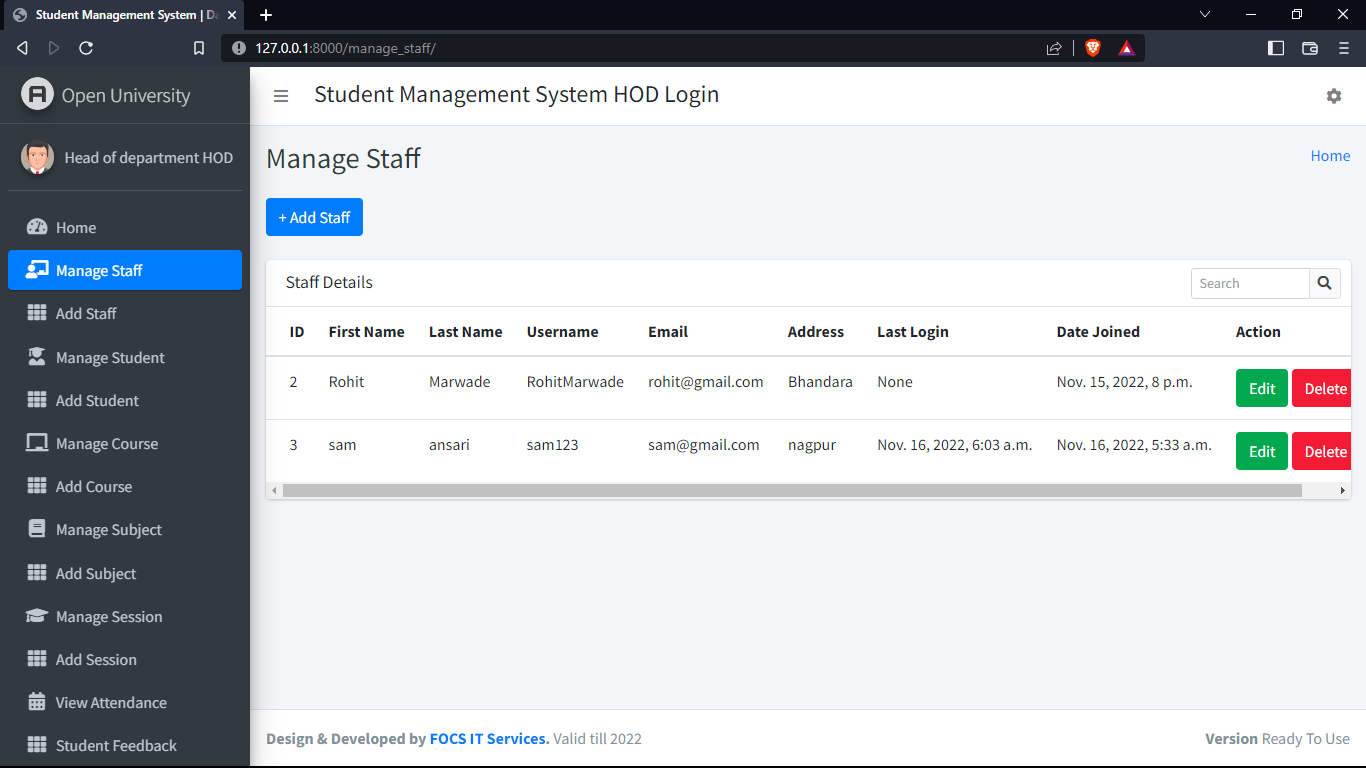
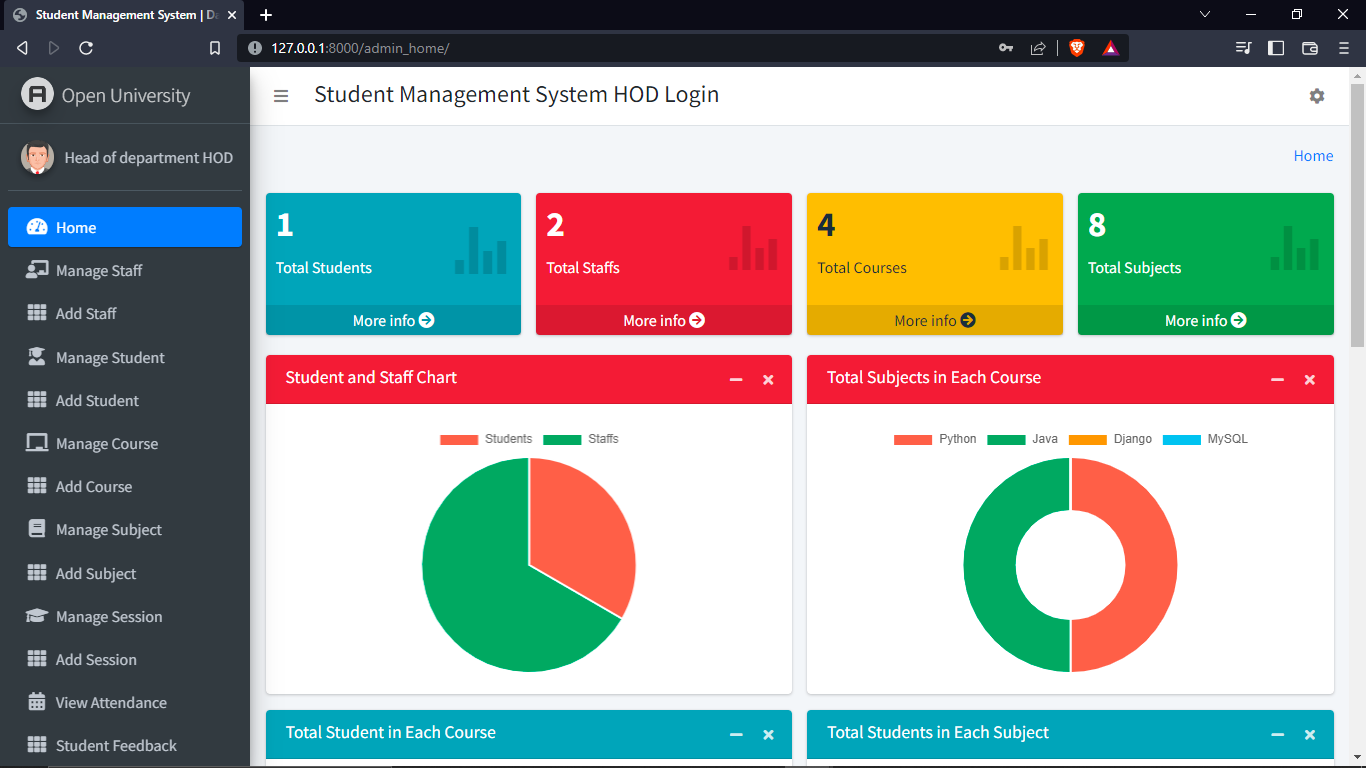
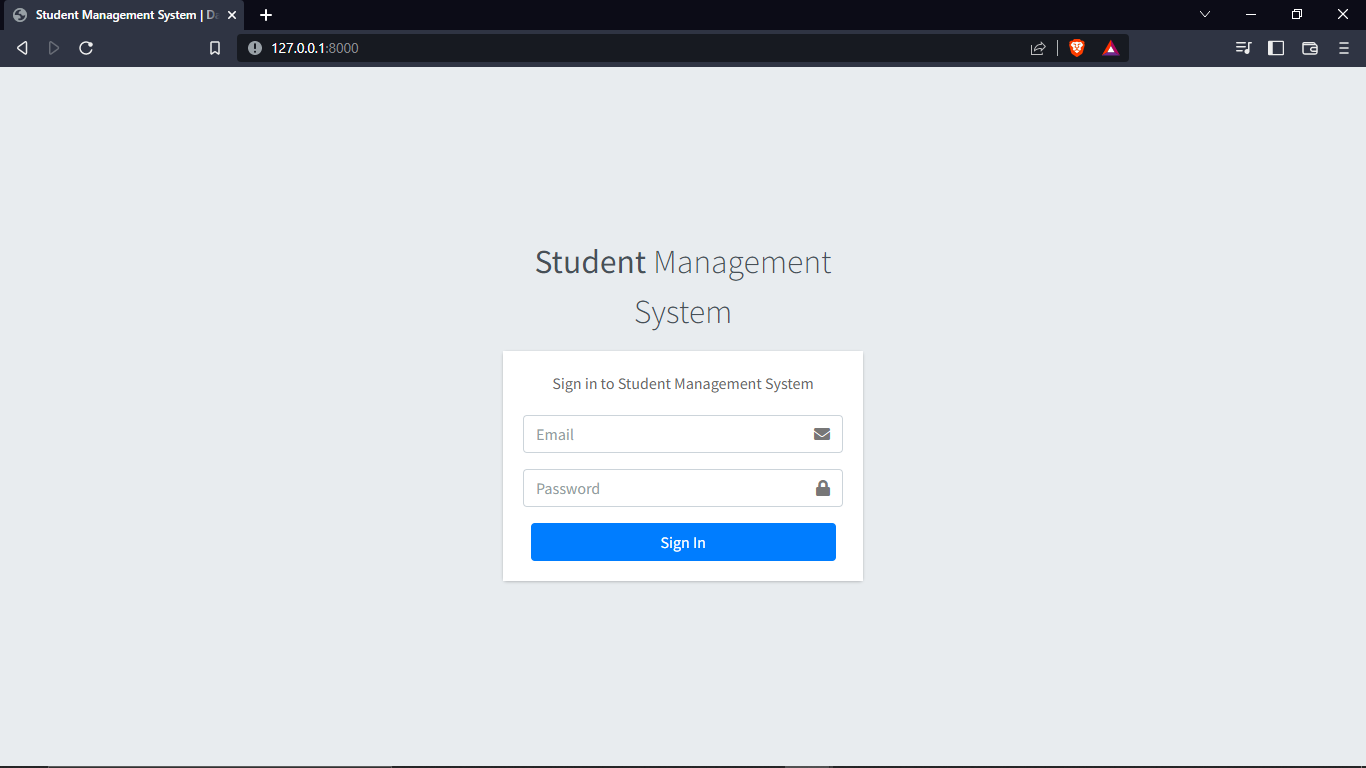
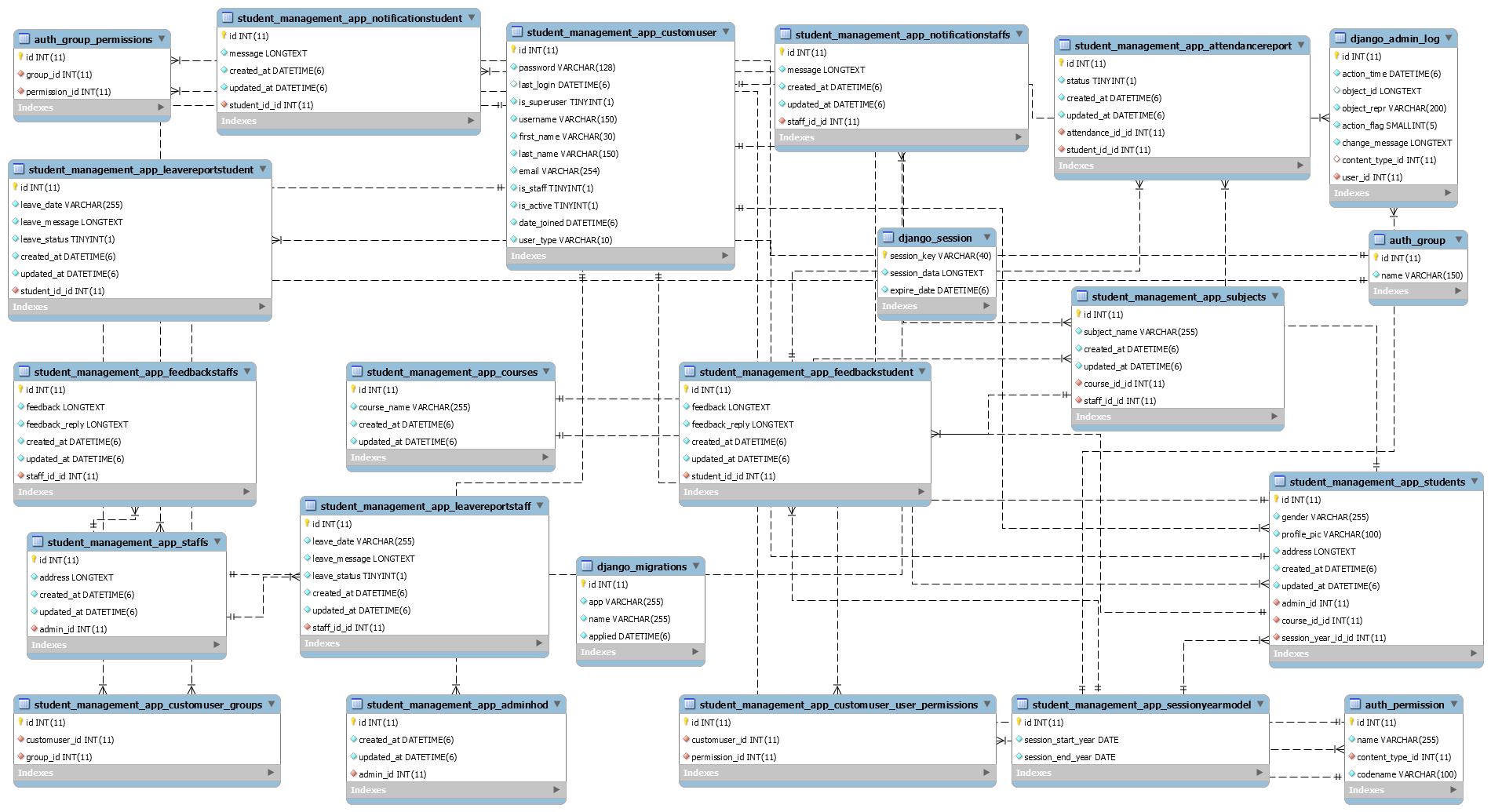
Secure data storage and management systems to ensure the privacy and security of farmers' data.

**ADVANTAGES**

The Smart Farming Assistant webapp may offer a number of advantages for farmers and agriculturists, including:

1. Improved efficiency and productivity of farming operations: The webapp provides access to a range of tools and features that can help farmers monitor and manage their operations, such as weather tracking, soil health monitoring, and irrigation management. This can help farmers make more informed decisions and optimize their farming practices,
2. Reduced environmental impact of agriculture: The webapp provides information on sustainable and environmentally-friendly farming techniques, such as precision agriculture and smart irrigation. By utilizing these techniques, farmers can reduce their use of water, fertilizer, and other resources, leading to a reduced environmental impact of their operations.
3. Access to personalized recommendations and advice: The webapp uses artificial intelligence algorithms to generate personalized recommendations and advice for farmers based on their specific needs and goals. This can help farmers stay up-to-date with the latest advances in smart farming and to incorporate these technologies into their operations.
4. Support for the adoption of smart farming technologies: The webapp offers a range of resources and information on funding, training, and other support services for farmers looking to adopt smart farming technologies. This can help farmers overcome barriers to adoption and to take full advantage of the opportunities offered by these technologies.
5. Promotion of the responsible and ethical use of technology in agriculture: The webapp provides information on the potential challenges and ethical implications of using technology in agriculture, and encourages the responsible and sustainable use of these technologies. This can help ensure that the benefits of smart farming are realized in a way that is safe and beneficial for all

**SnapShoots**



**Code**

Setting.py code

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = '(i#\*06f#keydy\_fh17bf=$0f6v)^wr^l7\*u4gq42m\*sztu#2\_m'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'student\_management\_app',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'student\_management\_app.LoginCheckMiddleWare.LoginCheckMiddleWare',

]

ROOT\_URLCONF = 'student\_management\_system.urls'

TEMPLATES = [

    {

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

        'DIRS': [],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

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

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

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

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

            ],

        },

    },

]

WSGI\_APPLICATION = 'student\_management\_system.wsgi.application'

# Database

# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    }

}

# Password validation

# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC\_URL = '/static/'

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

STATICFILES\_DIRS = [

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

]

MEDIA\_URL = '/media/'

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

#For Custom USER

AUTH\_USER\_MODEL = "student\_management\_app.CustomUser"

# Registering Custom Backend "EmailBackEnd"

AUTHENTICATION\_BACKENDS = ['student\_management\_app.EmailBackEnd.EmailBackEnd']

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

Urls.py code

from django.contrib import admin

from django.urls import path, include

from django.conf.urls.static import static

from student\_management\_system import settings

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('student\_management\_app.urls')),

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

Forms.py code

from django import forms

from django.forms import Form

from student\_management\_app.models import Courses, SessionYearModel

class DateInput(forms.DateInput):

    input\_type = "date"

class AddStudentForm(forms.Form):

    email = forms.EmailField(label="Email", max\_length=50, widget=forms.EmailInput(attrs={"class":"form-control"}))

    password = forms.CharField(label="Password", max\_length=50, widget=forms.PasswordInput(attrs={"class":"form-control"}))

    first\_name = forms.CharField(label="First Name", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    last\_name = forms.CharField(label="Last Name", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    username = forms.CharField(label="Username", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    address = forms.CharField(label="Address", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    #For Displaying Courses

    try:

        courses = Courses.objects.all()

        course\_list = []

        for course in courses:

            single\_course = (course.id, course.course\_name)

            course\_list.append(single\_course)

    except:

        course\_list = []

    #For Displaying Session Years

    try:

        session\_years = SessionYearModel.objects.all()

        session\_year\_list = []

        for session\_year in session\_years:

            single\_session\_year = (session\_year.id, str(session\_year.session\_start\_year)+" to "+str(session\_year.session\_end\_year))

            session\_year\_list.append(single\_session\_year)

    except:

        session\_year\_list = []

    gender\_list = (

        ('Male','Male'),

        ('Female','Female')

    )

    course\_id = forms.ChoiceField(label="Course", choices=course\_list, widget=forms.Select(attrs={"class":"form-control"}))

    gender = forms.ChoiceField(label="Gender", choices=gender\_list, widget=forms.Select(attrs={"class":"form-control"}))

    session\_year\_id = forms.ChoiceField(label="Session Year", choices=session\_year\_list, widget=forms.Select(attrs={"class":"form-control"}))

    # session\_start\_year = forms.DateField(label="Session Start", widget=DateInput(attrs={"class":"form-control"}))

    # session\_end\_year = forms.DateField(label="Session End", widget=DateInput(attrs={"class":"form-control"}))

    profile\_pic = forms.FileField(label="Profile Pic", required=False, widget=forms.FileInput(attrs={"class":"form-control"}))

class EditStudentForm(forms.Form):

    email = forms.EmailField(label="Email", max\_length=50, widget=forms.EmailInput(attrs={"class":"form-control"}))

    first\_name = forms.CharField(label="First Name", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    last\_name = forms.CharField(label="Last Name", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    username = forms.CharField(label="Username", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    address = forms.CharField(label="Address", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    #For Displaying Courses

    try:

        courses = Courses.objects.all()

        course\_list = []

        for course in courses:

            single\_course = (course.id, course.course\_name)

            course\_list.append(single\_course)

    except:

        course\_list = []

    #For Displaying Session Years

    try:

        session\_years = SessionYearModel.objects.all()

        session\_year\_list = []

        for session\_year in session\_years:

            single\_session\_year = (session\_year.id, str(session\_year.session\_start\_year)+" to "+str(session\_year.session\_end\_year))

            session\_year\_list.append(single\_session\_year)

    except:

        session\_year\_list = []

    gender\_list = (

        ('Male','Male'),

        ('Female','Female')

    )

    course\_id = forms.ChoiceField(label="Course", choices=course\_list, widget=forms.Select(attrs={"class":"form-control"}))

    gender = forms.ChoiceField(label="Gender", choices=gender\_list, widget=forms.Select(attrs={"class":"form-control"}))

    session\_year\_id = forms.ChoiceField(label="Session Year", choices=session\_year\_list, widget=forms.Select(attrs={"class":"form-control"}))

    # session\_start\_year = forms.DateField(label="Session Start", widget=DateInput(attrs={"class":"form-control"}))

    # session\_end\_year = forms.DateField(label="Session End", widget=DateInput(attrs={"class":"form-control"}))

    profile\_pic = forms.FileField(label="Profile Pic", required=False, widget=forms.FileInput(attrs={"class":"form-control"}))

forms.py

from django.utils.deprecation import MiddlewareMixin

from django.shortcuts import render, redirect

from django.urls import reverse

class LoginCheckMiddleWare(MiddlewareMixin):

    def process\_view(self, request, view\_func, view\_args, view\_kwargs):

        modulename = view\_func.\_\_module\_\_

        # print(modulename)

        user = request.user

        #Check whether the user is logged in or not

        if user.is\_authenticated:

            if user.user\_type == "1":

                if modulename == "student\_management\_app.HodViews":

                    pass

                elif modulename == "student\_management\_app.views" or modulename == "django.views.static":

                    pass

                else:

                    return redirect("admin\_home")

            elif user.user\_type == "2":

                if modulename == "student\_management\_app.StaffViews":

                    pass

                elif modulename == "student\_management\_app.views" or modulename == "django.views.static":

                    pass

                else:

                    return redirect("staff\_home")

            elif user.user\_type == "3":

                if modulename == "student\_management\_app.StudentViews":

                    pass

                elif modulename == "student\_management\_app.views" or modulename == "django.views.static":

                    pass

                else:

                    return redirect("student\_home")

            else:

                return redirect("login")

        else:

            if request.path == reverse("login") or request.path == reverse("doLogin"):

                pass

            else:

                return redirect("login")

views.py code

# from channels.auth import login, logout

from django.contrib.auth import authenticate, login, logout

from django.http import HttpResponseRedirect, HttpResponse

from django.shortcuts import render, redirect

from django.contrib import messages

from student\_management\_app.EmailBackEnd import EmailBackEnd

def home(request):

    return render(request, 'index.html')

def loginPage(request):

    return render(request, 'login.html')

def doLogin(request):

    if request.method != "POST":

        return HttpResponse("<h2>Method Not Allowed</h2>")

    else:

        user = EmailBackEnd.authenticate(request, username=request.POST.get('email'), password=request.POST.get('password'))

        if user != None:

            login(request, user)

            user\_type = user.user\_type

            #return HttpResponse("Email: "+request.POST.get('email')+ " Password: "+request.POST.get('password'))

            if user\_type == '1':

                return redirect('admin\_home')

            elif user\_type == '2':

                # return HttpResponse("Staff Login")

                return redirect('staff\_home')

            elif user\_type == '3':

                # return HttpResponse("Student Login")

                return redirect('student\_home')

            else:

                messages.error(request, "Invalid Login!")

                return redirect('login')

        else:

            messages.error(request, "Invalid Login Credentials!")

            #return HttpResponseRedirect("/")

            return redirect('login')

def get\_user\_details(request):

    if request.user != None:

        return HttpResponse("User: "+request.user.email+" User Type: "+request.user.user\_type)

    else:

        return HttpResponse("Please Login First")

def logout\_user(request):

    logout(request)

    return HttpResponseRedirect('/')

apps urls.py code

from django.urls import path, include

from . import views

from .import HodViews, StaffViews, StudentViews

urlpatterns = [

    path('', views.loginPage, name="login"),

    # path('accounts/', include('django.contrib.auth.urls')),

    path('doLogin/', views.doLogin, name="doLogin"),

    path('get\_user\_details/', views.get\_user\_details, name="get\_user\_details"),

    path('logout\_user/', views.logout\_user, name="logout\_user"),

    path('admin\_home/', HodViews.admin\_home, name="admin\_home"),

    path('add\_staff/', HodViews.add\_staff, name="add\_staff"),

    path('add\_staff\_save/', HodViews.add\_staff\_save, name="add\_staff\_save"),

    path('manage\_staff/', HodViews.manage\_staff, name="manage\_staff"),

    path('edit\_staff/<staff\_id>/', HodViews.edit\_staff, name="edit\_staff"),

    path('edit\_staff\_save/', HodViews.edit\_staff\_save, name="edit\_staff\_save"),

    path('delete\_staff/<staff\_id>/', HodViews.delete\_staff, name="delete\_staff"),

    path('add\_course/', HodViews.add\_course, name="add\_course"),

    path('add\_course\_save/', HodViews.add\_course\_save, name="add\_course\_save"),

    path('manage\_course/', HodViews.manage\_course, name="manage\_course"),

    path('edit\_course/<course\_id>/', HodViews.edit\_course, name="edit\_course"),

    path('edit\_course\_save/', HodViews.edit\_course\_save, name="edit\_course\_save"),

    path('delete\_course/<course\_id>/', HodViews.delete\_course, name="delete\_course"),

    path('manage\_session/', HodViews.manage\_session, name="manage\_session"),

    path('add\_session/', HodViews.add\_session, name="add\_session"),

    path('add\_session\_save/', HodViews.add\_session\_save, name="add\_session\_save"),

    path('edit\_session/<session\_id>', HodViews.edit\_session, name="edit\_session"),

    path('edit\_session\_save/', HodViews.edit\_session\_save, name="edit\_session\_save"),

    path('delete\_session/<session\_id>/', HodViews.delete\_session, name="delete\_session"),

    path('add\_student/', HodViews.add\_student, name="add\_student"),

    path('add\_student\_save/', HodViews.add\_student\_save, name="add\_student\_save"),

    path('edit\_student/<student\_id>', HodViews.edit\_student, name="edit\_student"),

    path('edit\_student\_save/', HodViews.edit\_student\_save, name="edit\_student\_save"),

    path('manage\_student/', HodViews.manage\_student, name="manage\_student"),

    path('delete\_student/<student\_id>/', HodViews.delete\_student, name="delete\_student"),

    path('add\_subject/', HodViews.add\_subject, name="add\_subject"),

    path('add\_subject\_save/', HodViews.add\_subject\_save, name="add\_subject\_save"),

    path('manage\_subject/', HodViews.manage\_subject, name="manage\_subject"),

    path('edit\_subject/<subject\_id>/', HodViews.edit\_subject, name="edit\_subject"),

    path('edit\_subject\_save/', HodViews.edit\_subject\_save, name="edit\_subject\_save"),

    path('delete\_subject/<subject\_id>/', HodViews.delete\_subject, name="delete\_subject"),

    path('check\_email\_exist/', HodViews.check\_email\_exist, name="check\_email\_exist"),

    path('check\_username\_exist/', HodViews.check\_username\_exist, name="check\_username\_exist"),

    path('student\_feedback\_message/', HodViews.student\_feedback\_message, name="student\_feedback\_message"),

    path('student\_feedback\_message\_reply/', HodViews.student\_feedback\_message\_reply, name="student\_feedback\_message\_reply"),

    path('staff\_feedback\_message/', HodViews.staff\_feedback\_message, name="staff\_feedback\_message"),

    path('staff\_feedback\_message\_reply/', HodViews.staff\_feedback\_message\_reply, name="staff\_feedback\_message\_reply"),

    path('student\_leave\_view/', HodViews.student\_leave\_view, name="student\_leave\_view"),

    path('student\_leave\_approve/<leave\_id>/', HodViews.student\_leave\_approve, name="student\_leave\_approve"),

    path('student\_leave\_reject/<leave\_id>/', HodViews.student\_leave\_reject, name="student\_leave\_reject"),

    path('staff\_leave\_view/', HodViews.staff\_leave\_view, name="staff\_leave\_view"),

    path('staff\_leave\_approve/<leave\_id>/', HodViews.staff\_leave\_approve, name="staff\_leave\_approve"),

    path('staff\_leave\_reject/<leave\_id>/', HodViews.staff\_leave\_reject, name="staff\_leave\_reject"),

    path('admin\_view\_attendance/', HodViews.admin\_view\_attendance, name="admin\_view\_attendance"),

    path('admin\_get\_attendance\_dates/', HodViews.admin\_get\_attendance\_dates, name="admin\_get\_attendance\_dates"),

    path('admin\_get\_attendance\_student/', HodViews.admin\_get\_attendance\_student, name="admin\_get\_attendance\_student"),

    path('admin\_profile/', HodViews.admin\_profile, name="admin\_profile"),

    path('admin\_profile\_update/', HodViews.admin\_profile\_update, name="admin\_profile\_update"),

    # URLS for Staff

    path('staff\_home/', StaffViews.staff\_home, name="staff\_home"),

    path('staff\_take\_attendance/', StaffViews.staff\_take\_attendance, name="staff\_take\_attendance"),

    path('get\_students/', StaffViews.get\_students, name="get\_students"),

    path('save\_attendance\_data/', StaffViews.save\_attendance\_data, name="save\_attendance\_data"),

    path('staff\_update\_attendance/', StaffViews.staff\_update\_attendance, name="staff\_update\_attendance"),

    path('get\_attendance\_dates/', StaffViews.get\_attendance\_dates, name="get\_attendance\_dates"),

    path('get\_attendance\_student/', StaffViews.get\_attendance\_student, name="get\_attendance\_student"),

    path('update\_attendance\_data/', StaffViews.update\_attendance\_data, name="update\_attendance\_data"),

    path('staff\_apply\_leave/', StaffViews.staff\_apply\_leave, name="staff\_apply\_leave"),

    path('staff\_apply\_leave\_save/', StaffViews.staff\_apply\_leave\_save, name="staff\_apply\_leave\_save"),

    path('staff\_feedback/', StaffViews.staff\_feedback, name="staff\_feedback"),

    path('staff\_feedback\_save/', StaffViews.staff\_feedback\_save, name="staff\_feedback\_save"),

    path('staff\_profile/', StaffViews.staff\_profile, name="staff\_profile"),

    path('staff\_profile\_update/', StaffViews.staff\_profile\_update, name="staff\_profile\_update"),

    path('staff\_add\_result/', StaffViews.staff\_add\_result, name="staff\_add\_result"),

    path('staff\_add\_result\_save/', StaffViews.staff\_add\_result\_save, name="staff\_add\_result\_save"),

    # URSL for Student

    path('student\_home/', StudentViews.student\_home, name="student\_home"),

    path('student\_view\_attendance/', StudentViews.student\_view\_attendance, name="student\_view\_attendance"),

    path('student\_view\_attendance\_post/', StudentViews.student\_view\_attendance\_post, name="student\_view\_attendance\_post"),

    path('student\_apply\_leave/', StudentViews.student\_apply\_leave, name="student\_apply\_leave"),

    path('student\_apply\_leave\_save/', StudentViews.student\_apply\_leave\_save, name="student\_apply\_leave\_save"),

    path('student\_feedback/', StudentViews.student\_feedback, name="student\_feedback"),

    path('student\_feedback\_save/', StudentViews.student\_feedback\_save, name="student\_feedback\_save"),

    path('student\_profile/', StudentViews.student\_profile, name="student\_profile"),

    path('student\_profile\_update/', StudentViews.student\_profile\_update, name="student\_profile\_update"),

    path('student\_view\_result/', StudentViews.student\_view\_result, name="student\_view\_result"),

]

Models.py code

from django.contrib.auth.models import AbstractUser

from django.db import models

from django.db.models.signals import post\_save

from django.dispatch import receiver

class SessionYearModel(models.Model):

    id = models.AutoField(primary\_key=True)

    session\_start\_year = models.DateField()

    session\_end\_year = models.DateField()

    objects = models.Manager()

# Overriding the Default Django Auth User and adding One More Field (user\_type)

class CustomUser(AbstractUser):

    user\_type\_data = ((1, "HOD"), (2, "Staff"), (3, "Student"))

    user\_type = models.CharField(default=1, choices=user\_type\_data, max\_length=10)

class AdminHOD(models.Model):

    id = models.AutoField(primary\_key=True)

    admin = models.OneToOneField(CustomUser, on\_delete = models.CASCADE)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class Staffs(models.Model):

    id = models.AutoField(primary\_key=True)

    admin = models.OneToOneField(CustomUser, on\_delete = models.CASCADE)

    address = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class Courses(models.Model):

    id = models.AutoField(primary\_key=True)

    course\_name = models.CharField(max\_length=255)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

    # def \_\_str\_\_(self):

    #     return self.course\_name

class Subjects(models.Model):

    id =models.AutoField(primary\_key=True)

    subject\_name = models.CharField(max\_length=255)

    course\_id = models.ForeignKey(Courses, on\_delete=models.CASCADE, default=1) #need to give defauult course

    staff\_id = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class Students(models.Model):

    id = models.AutoField(primary\_key=True)

    admin = models.OneToOneField(CustomUser, on\_delete = models.CASCADE)

    gender = models.CharField(max\_length=50)

    profile\_pic = models.FileField()

    address = models.TextField()

    course\_id = models.ForeignKey(Courses, on\_delete=models.DO\_NOTHING, default=1)

    session\_year\_id = models.ForeignKey(SessionYearModel, on\_delete=models.CASCADE)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class Attendance(models.Model):

    # Subject Attendance

    id = models.AutoField(primary\_key=True)

    subject\_id = models.ForeignKey(Subjects, on\_delete=models.DO\_NOTHING)

    attendance\_date = models.DateField()

    session\_year\_id = models.ForeignKey(SessionYearModel, on\_delete=models.CASCADE)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class AttendanceReport(models.Model):

    # Individual Student Attendance

    id = models.AutoField(primary\_key=True)

    student\_id = models.ForeignKey(Students, on\_delete=models.DO\_NOTHING)

    attendance\_id = models.ForeignKey(Attendance, on\_delete=models.CASCADE)

    status = models.BooleanField(default=False)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class LeaveReportStudent(models.Model):

    id = models.AutoField(primary\_key=True)

    student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE)

    leave\_date = models.CharField(max\_length=255)

    leave\_message = models.TextField()

    leave\_status = models.IntegerField(default=0)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class LeaveReportStaff(models.Model):

    id = models.AutoField(primary\_key=True)

    staff\_id = models.ForeignKey(Staffs, on\_delete=models.CASCADE)

    leave\_date = models.CharField(max\_length=255)

    leave\_message = models.TextField()

    leave\_status = models.IntegerField(default=0)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class FeedBackStudent(models.Model):

    id = models.AutoField(primary\_key=True)

    student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE)

    feedback = models.TextField()

    feedback\_reply = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class FeedBackStaffs(models.Model):

    id = models.AutoField(primary\_key=True)

    staff\_id = models.ForeignKey(Staffs, on\_delete=models.CASCADE)

    feedback = models.TextField()

    feedback\_reply = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class NotificationStudent(models.Model):

    id = models.AutoField(primary\_key=True)

    student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE)

    message = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class NotificationStaffs(models.Model):

    id = models.AutoField(primary\_key=True)

    stafff\_id = models.ForeignKey(Staffs, on\_delete=models.CASCADE)

    message = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

class StudentResult(models.Model):

    id = models.AutoField(primary\_key=True)

    student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE)

    subject\_id = models.ForeignKey(Subjects, on\_delete=models.CASCADE)

    subject\_exam\_marks = models.FloatField(default=0)

    subject\_assignment\_marks = models.FloatField(default=0)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    objects = models.Manager()

#Creating Django Signals

# It's like trigger in database. It will run only when Data is Added in CustomUser model

@receiver(post\_save, sender=CustomUser)

# Now Creating a Function which will automatically insert data in HOD, Staff or Student

def create\_user\_profile(sender, instance, created, \*\*kwargs):

    # if Created is true (Means Data Inserted)

    if created:

        # Check the user\_type and insert the data in respective tables

        if instance.user\_type == 1:

            AdminHOD.objects.create(admin=instance)

        if instance.user\_type == 2:

            Staffs.objects.create(admin=instance)

        if instance.user\_type == 3:

            Students.objects.create(admin=instance, course\_id=Courses.objects.get(id=1), session\_year\_id=SessionYearModel.objects.get(id=1), address="", profile\_pic="", gender="")

@receiver(post\_save, sender=CustomUser)

def save\_user\_profile(sender, instance, \*\*kwargs):

    if instance.user\_type == 1:

        instance.adminhod.save()

    if instance.user\_type == 2:

        instance.staffs.save()

    if instance.user\_type == 3:

        instance.students.save()

manage.py code

#!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

    os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'student\_management\_system.settings')

    try:

        from django.core.management import execute\_from\_command\_line

    except ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

            "forget to activate a virtual environment?"

        ) from exc

    execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**REFERENCES**

SYSTEM ANALYSIS AND DESIGN”, Air Walk Publication, EliasAwath.

2. “SOFTWARE ENGINEERING CONCEPTS”, Tata Mc Graw Hill Publication

3. Robin a. Reynolds haertle,” oop with Microsoft c#.net and ASP.net step by step “, Microsoft

press publication, third edition, 2002.

5. David Scoppa,” Microsoft asp.net”, Microsoft press publications, second edition,2000.

6. https://www.asp.net/get-started

7. https://www.w3schools.com/asp/default.ASp

8. https://en.wikipedia.org/wiki/ASP.NET

9. Karuna Chandraul, Archana Singh, "An Agricultural Application Research on Cloud Computing", International Journal of Current Engineering and Technology, 2015.

10. V.C. Patil, K.A. Caadi, D.P. iradar, M. Rangasamy, Internet of things and cloud computing for agriculture, 2016.

11. Rupika Yadav, Jhalak Rathod, Vaishnavi Nair,

"Bigdata Meets Small Sensors In The Precision

Agriculture", International Journal of Computer

Applications, 2015.

12. Prasanth R, Rahul D, Sharan S C, Suresh P,

“IOT based Smart Irrigation System for

Precision Agriculture”, International Research

Journal of Engineering and Technology (IRJET),

e-ISSN: 2395-0056 Volume: 06 Issue: 03 | Mar

2019

**Submitted by:**

**Prof. Heena B. Kachhela**

**Head of Department**

**Department of Computer Science & Engineering**