A

Major Project Report

On

GENERATING CLOUD MONITORS FROM MODELS TO SECURE CLOUD

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

D.MOUNIKA(18C21A0507)

S.SAI KIRAN (187R5A0512)

M.SAI KUMAR GOUD(187R5A0502)

Under the Guidance of

DR.G.MADHUKAR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGC Act.1956, NIRF Rank Band 201-250

Kandlakoya (V), Medchal Road, Hyderabad-501401.

2018-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled **"GENERATING CLOUD MONITORS FROM MODELS TO SECURE CLOUD"** being submitted by **D.Mounika(18C21A0507), M.Sai Kumar Goud (187R5A0502), S.Sai Kiran(187R5A0512)** in partial fulfillment of the requirements for the award of the degree of Btech in Computer Science and Engineering of the Jawaharlal nehru technological University Hyderabad, during the year 2021-2022.

The results embodied in this thesis have not submitted to any other University or institute for the award of any degree or diploma.

DR. G. MADHUKAR
(Associate Professor, PH.D)
INTERNAL GUIDE

Dr. A. Raji Reddy
DIRECTOR

Dr. K. Srujan Raju
HOD

EXTERNAL EXAMINER

Submitted on viva voce Examination held on _____

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project. We take this opportunity to express my profound gratitude and deep regard to myguide

DR. G.Madhukar , Assistant. Professor,ph.d for her exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark. We also take this opportunity to express a deep sense of gratitude to Project Review Committee (PRC) **Dr. M. Varaprasad Rao, Mr. J. Narasimha Rao, Dr. T. S. Mastan Rao, Dr. Suwarna Gothane, Mr. A. Uday Kiran, Mr. A. Kiran Kumar, Mrs. G. Latha,** fortheir cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to the Head of the Department **Dr. K. Srujan Raju** for providing excellent infrastructure and a nice atmosphere for completing this project successfully.

We are obliged to our Director **Dr. A. Raji Reddy** for being cooperative throughout the course of this project. We would like to express our sincere gratitude to our Chairman Sri. **Ch. Gopal Reddy** for his encouragement throughout the course of thisproject

The guidance and support received from all the members of **CMR TECHNICAL CAMPUS** who contributed and who are contributing to this project, was vital for the success of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement without which this assignment would not be possible. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project.

D.Mounika (18C21A0507)

M.Sai Kumar Goud (187R5A0502)

S.Sai Kiran(187R5A0512)

ABSTRACT

Authorization is an important security concern in cloud computing environments. It aims at regulating an access of the users to system resources. A large number of resources associated with REST APIs typical in cloud makes an implementation of security requirements challenging and error-prone. To alleviate this problem, in this paper we propose an implementation of security cloud monitor. We rely on model-driven approach to represent the functional and security requirements. Models are then used to generate cloud monitors. The cloud monitors contain contracts used to automatically verify the implementation. We use Django web framework to implement cloud monitor and OpenStack to validate our implementation.

We present a cloud monitoring framework that supports a semi-automated approach to monitoring a private cloud implementation with respect to its conformance to the functional requirements and API access control policy. Our work uses UML (Unified Modeling Language) models with OCL (Object Constraint Language) to specify the behavioral interface with security constraints for the cloud implementation.

The behavioral interface of the REST API provides an information regarding the methods that can be invoked on it and pre- and post-conditions of the methods. In the current practice, the pre- and post-conditions are usually given as the textual descriptions associated with the API methods. In our work, we rely on the Design by Contract (DBC) framework, which allows us to define security and functional requirements as verifiable contracts.

LIST OF FIGURES/TABLES

LIST OF SCREENSHOTS

TABLE OF CONTENTS

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

This project is titled as “Generating cloud monitors from models to secure cloud”. This software provides facility a cloud monitoring framework that supports a semi-automated approach to monitoring a private cloud implementation with respect to its conformance to the functional requirements and API access control policy. Our work uses UML (Unified Modeling Language) models with OCL (Object Constraint Language) to specify the behavioral interface with security constraints for the cloud implementation.

1.2 PROJECT PURPOSE

The idea proposed here,A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system’s relationship to help user decision-making

1.3 PROJECT FEATURES

The main features of this project is it enables creating a (stateful) wrapper that emulates the usage scenarios and defines security-enriched behavioral contracts to monitor cloud. This also allows the security experts to observe the coverage of the security requirements during the testing phase.

2. SYSTEM ANALYSIS

2.SYSTEM ANALYSIS

SYSTEM ANALYSIS

System Analysis is the important phase in the system development process. The System is studied to the minute details and analysed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, “what must be done to solve the problem?” The system is viewed as a whole and the inputs to the system are identified.

Once analysis is completed the analyst has a firm understanding of what is to be done.

2.1 PROBLEM DEFINITION

Authorization is an important security concern in cloud computing environments. It aims at regulating an access of the users to system resources. A large number of resources associated with REST APIs typical in cloud makes an implementation of security requirements challenging and error-prone. To alleviate this problem, in this paper we propose an implementation of security cloud monitor. We rely on model-driven approach to represent the functional and security requirements. Models are then used to generate cloud monitors. The cloud monitors contain contracts used to automatically verify the implementation. We use Django web framework to implement cloud monitor and OpenStack to validate our implementation.

2.2 EXISTING SYSTEM

In many companies, private clouds are considered to be an important element of data center transformations. Private clouds are dedicated cloud environments created for the internal use by a single organization. Therefore, designing and developing secure private cloud environments for such a large number of users constitutes a major engineering challenge. Usually, cloud computing services offer REST APIs (REpresentational State Transfer Application Programming Interface) to their consumers. The REST architectural style exposes each piece of information

2.2.1 LIMITATIONS OF EXISTING SYSTEM

- Data breach and loss of critical data are among the top cloud security threats.
- The large number of URIs further complicates the task of the security experts, who should ensure that each URI, providing access to their system, is safeguarded to avoid data breaches or privilege escalation attacks.
- Since the source code of the Open Source clouds is often developed in a collaborative manner, it is a subject of frequent updates. The updates might introduce or remove a variety of features and hence, violate the security properties of the previous releases.

2.3 PROPOSED SYSTEM

We present a cloud monitoring framework that supports a semi-automated approach to monitoring a private cloud implementation with respect to its conformance to the functional requirements and API access control policy. Our work uses UML (Unified Modeling Language) models with OCL (Object Constraint Language) to specify the behavioral interface with security constraints for the cloud implementation. The behavioral interface of the REST API provides an information regarding the methods that can be invoked on it and pre- and post-conditions of the methods. In the current practice, the pre- and post-conditions are usually given as the textual descriptions associated with the API methods. In our work, we rely on the Design by Contract (DbC) framework, which allows us to define security and functional requirements as verifiable contracts.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

- Our methodology enables creating a (stateful) wrapper that emulates the usage scenarios and defines security-enriched behavioural contracts to monitor cloud.
- The proposed approach also facilitates the requirements traceability by ensuring the propagation of the security specifications into the code. This also allows the security experts to observe the coverage of the security requirements during the testing phase.

- The approach is implemented as a semi-automatic code generation tool in Django a Python web framework.

2.4 FEASIBILITY STUDY

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

2.4.1 ECONOMIC FEASIBILITY

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation:

- The costs conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already available, it give an indication of the system is economically possible for development.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 BEHAVIOURAL FEASIBILITY

This includes the following questions:

- Is there sufficient support for the users?
- Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives when developed and installed

2.5 HARDWARE & SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specifies the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

- Processor : i3 7th gen or higher RAM: 4 GB min.
- Free Space on Hard Disk : minimum 40 GB.

2.5.2 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software software requirements:

- Operating system: Windows 7
- Language : Python, Debugger and Emulator Any Browser (Particularly Chrome)
- IDE : Django

3. ARCHITECTURE

3.ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed for Generating cloud monitors from models to secure cloud

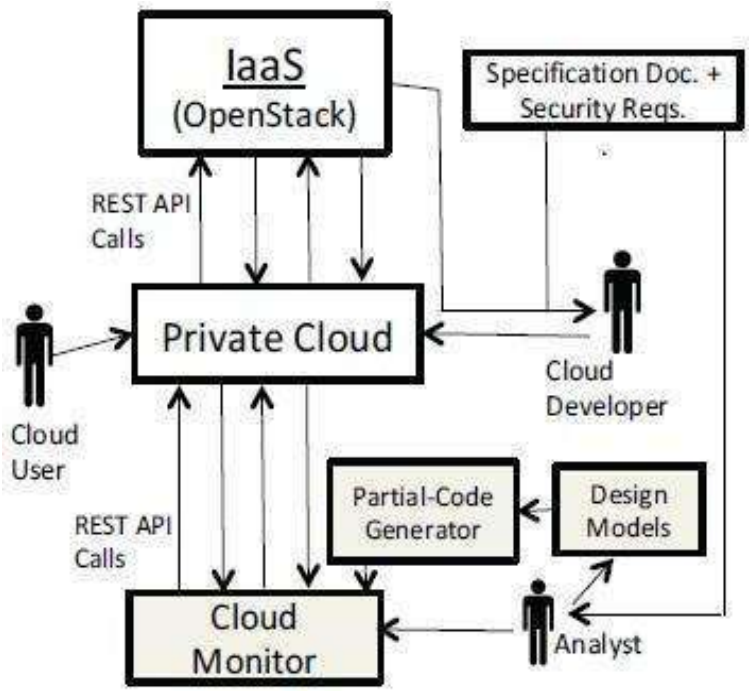


Fig no: 3.1

Fig: Project architecture of Generating cloud monitors from models to secure cloud

3.2 MODULE DESCRIPTION

User

It defines the access rights of the cloud users. A volume can be created, if the it has not exceeded its quota of the permitted volumes and a user Authorization is an important security concern in cloud computing environments. a POST request from the authorized user on the volumes resource would create a new volume. a DELETE request on the volume resource by an authorized user would delete the volume. if the user of the service is authorized to do so, and the volume is not attached to any instance. It aims at regulating an access of the users to system resources.

Cloud

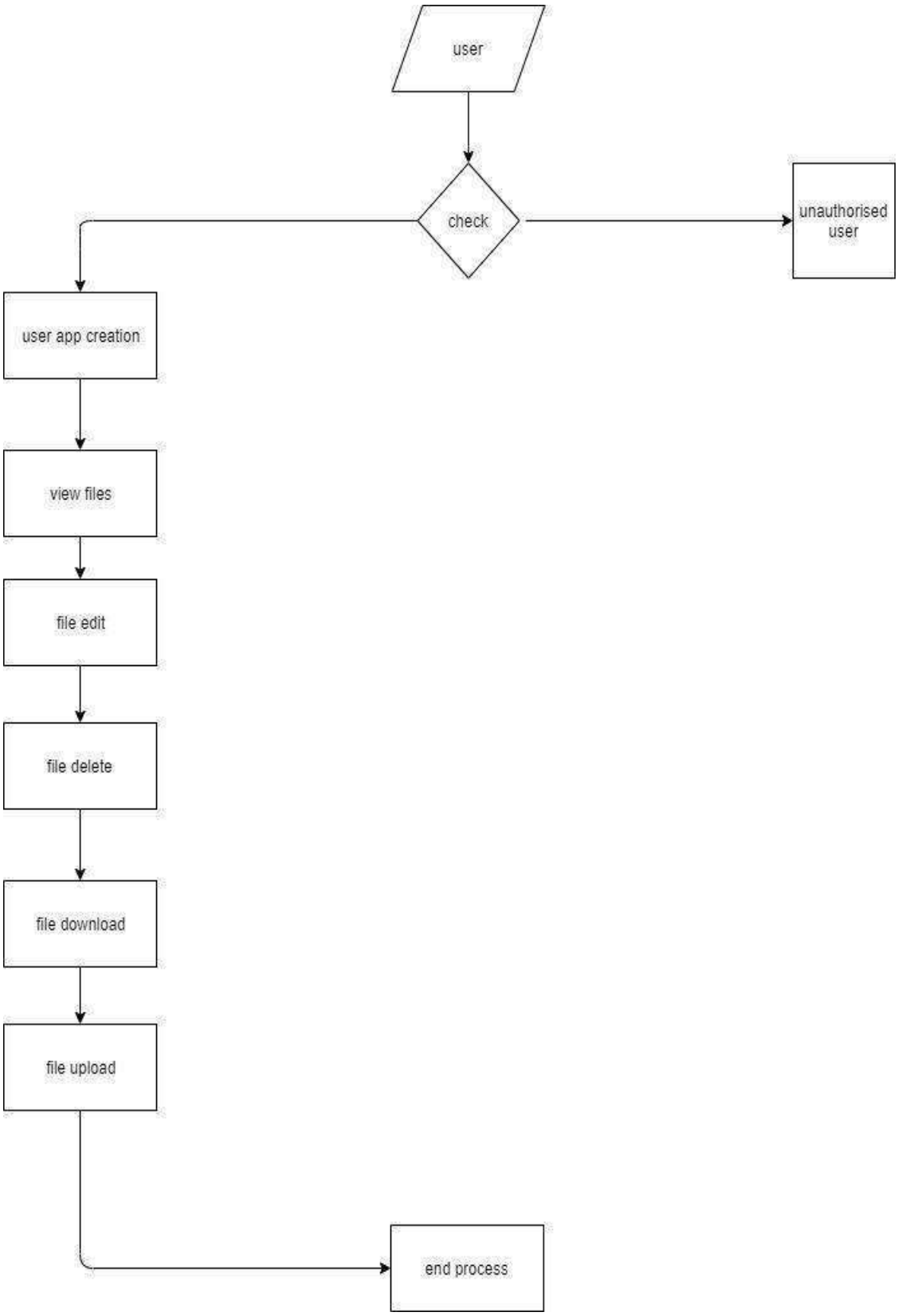
The cloud monitors contain contracts used to automatically verify the implementation. A cloud developer uses IaaS to develop a private cloud for her/his organization that would be used by different cloud users within the organization. In some cases, this private cloud may be implemented by a group of developers working collaboratively on different machines. We use Django web framework to implement cloud monitor and OpenStack to validate our implementation.

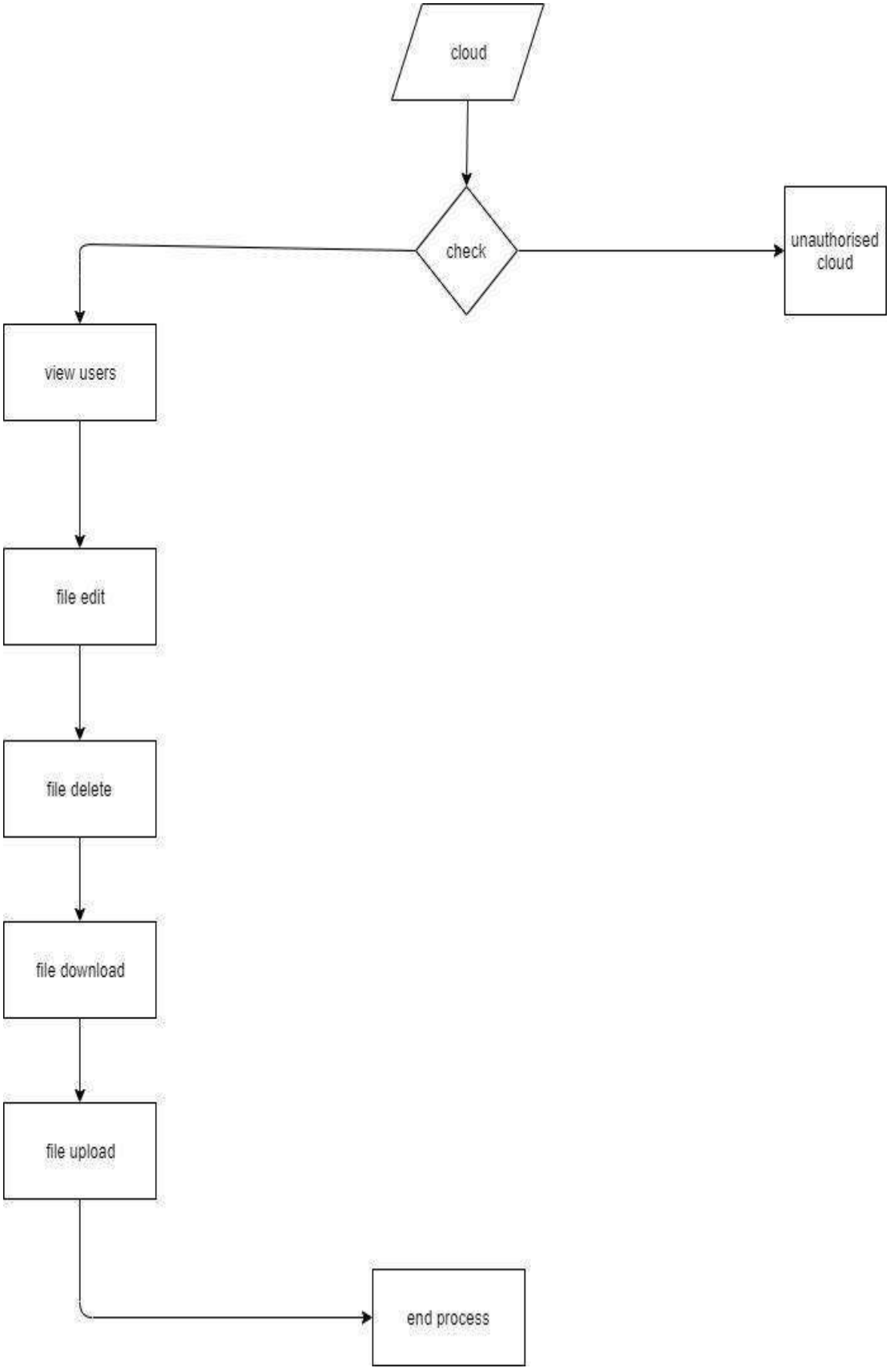
Admin

the cloud administrator using Keystone and users or usergroups are assigned the roles in these projects. It defines the access rights of the cloud users in the project. A volume can be created, if the project has not exceeded its quota of the permitted volumes and a user is authorized to create a volume in the project. Similarly, a volume can be deleted, if the user of the service is authorized to do so, and the volume is not attached to any instance, i.e., its status is not in-u

3.3 DATA FLOW DIAGRAM

1. to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and


user
check
unauthorised user
user app creation
view files
file edit
file delete
file download
file upload
end process


cloud
check
unauthorised cloud
view users
file edit
file delete
file download
file upload
end process

3.4 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

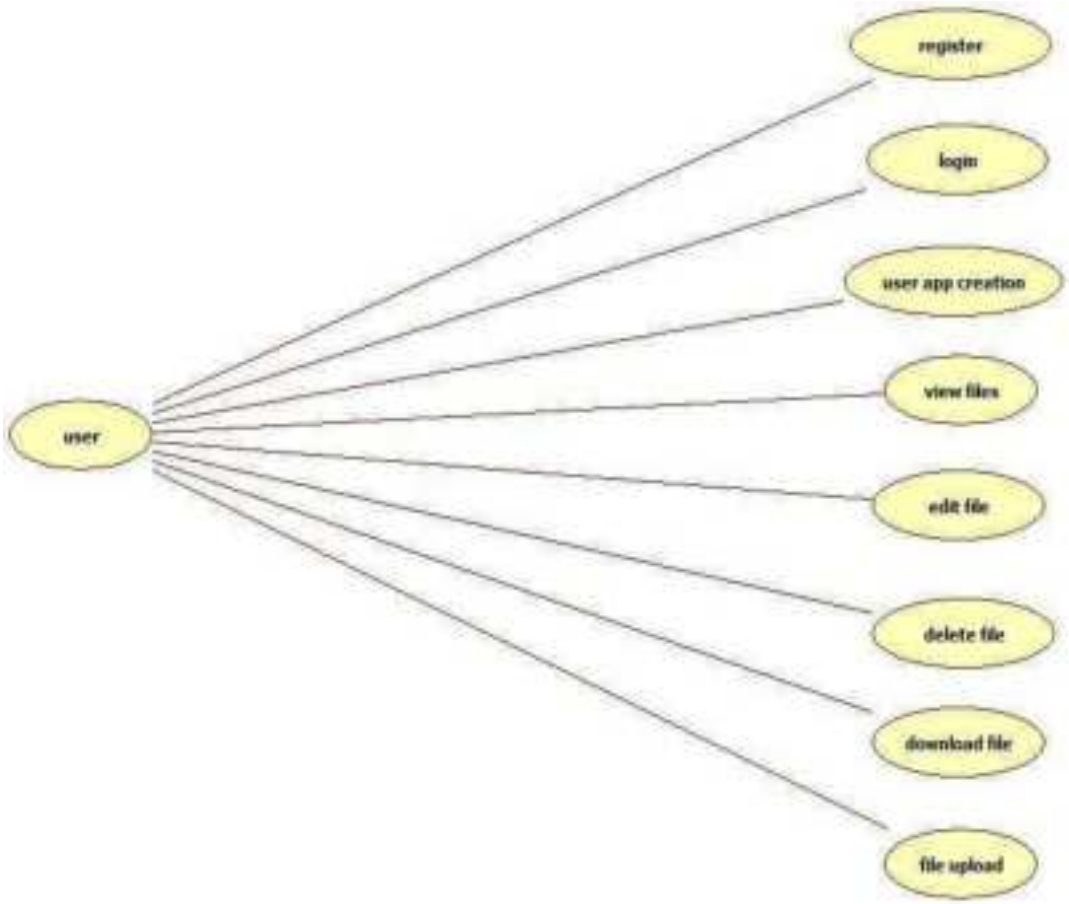


FIG NO: 3.4

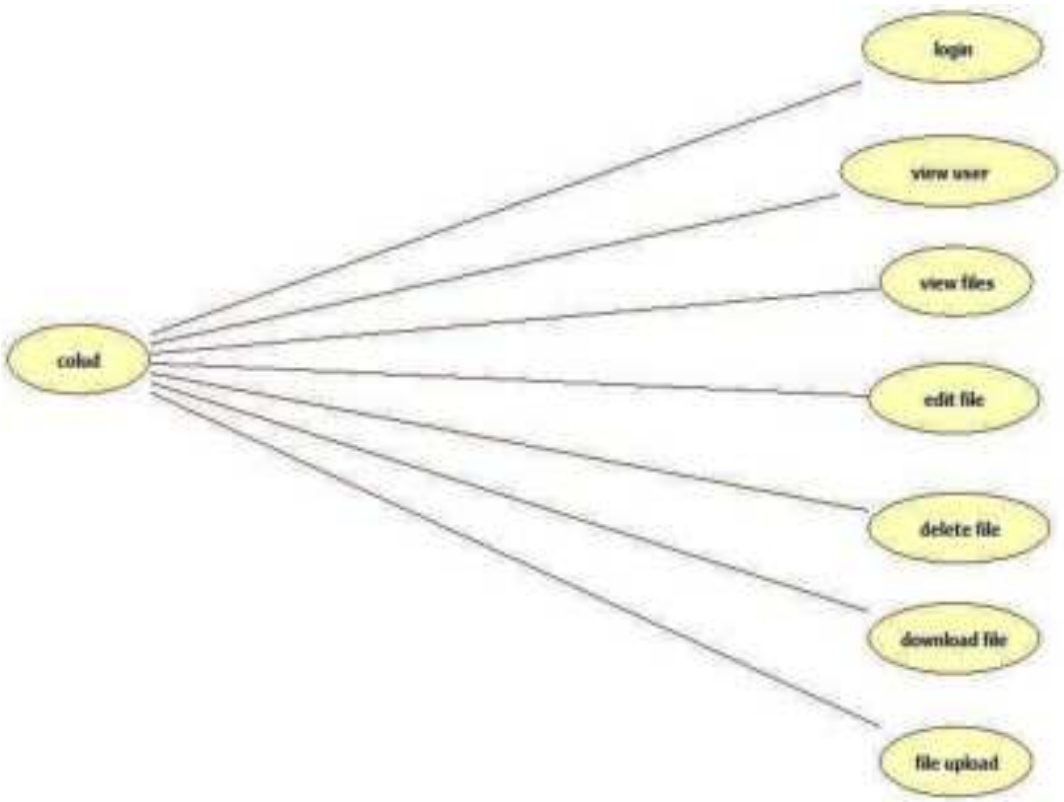


FIG NO:3.4

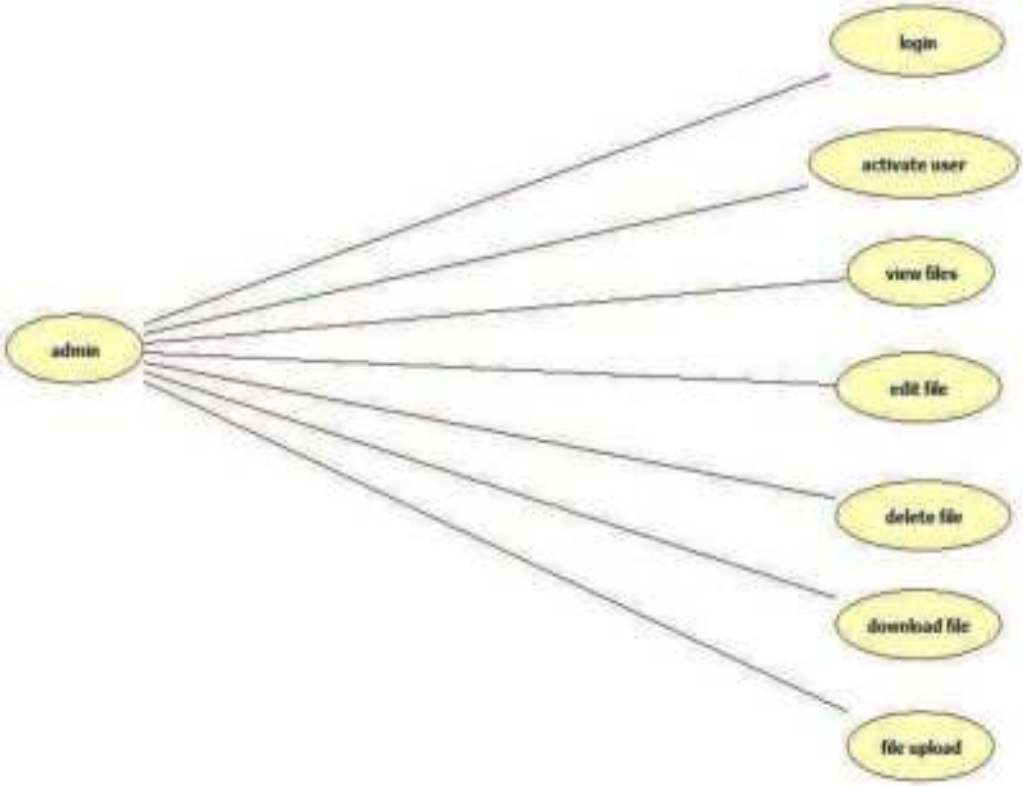
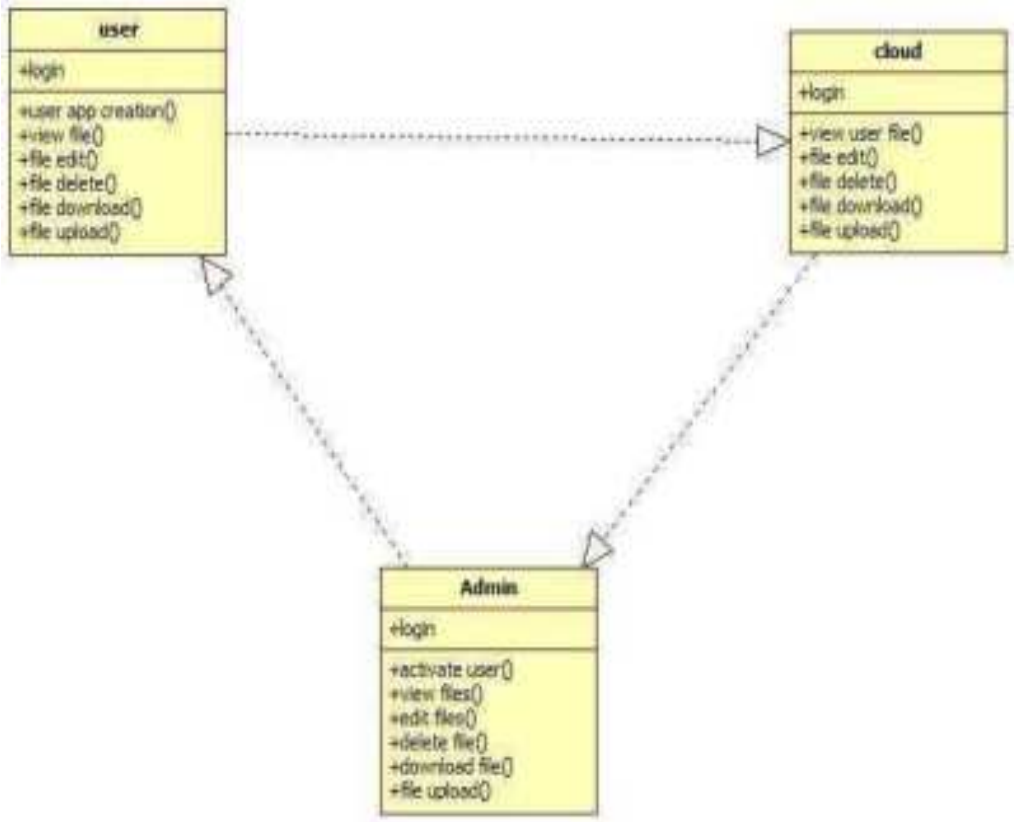


FIG NO: 3.4

3.5 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



3.6 SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

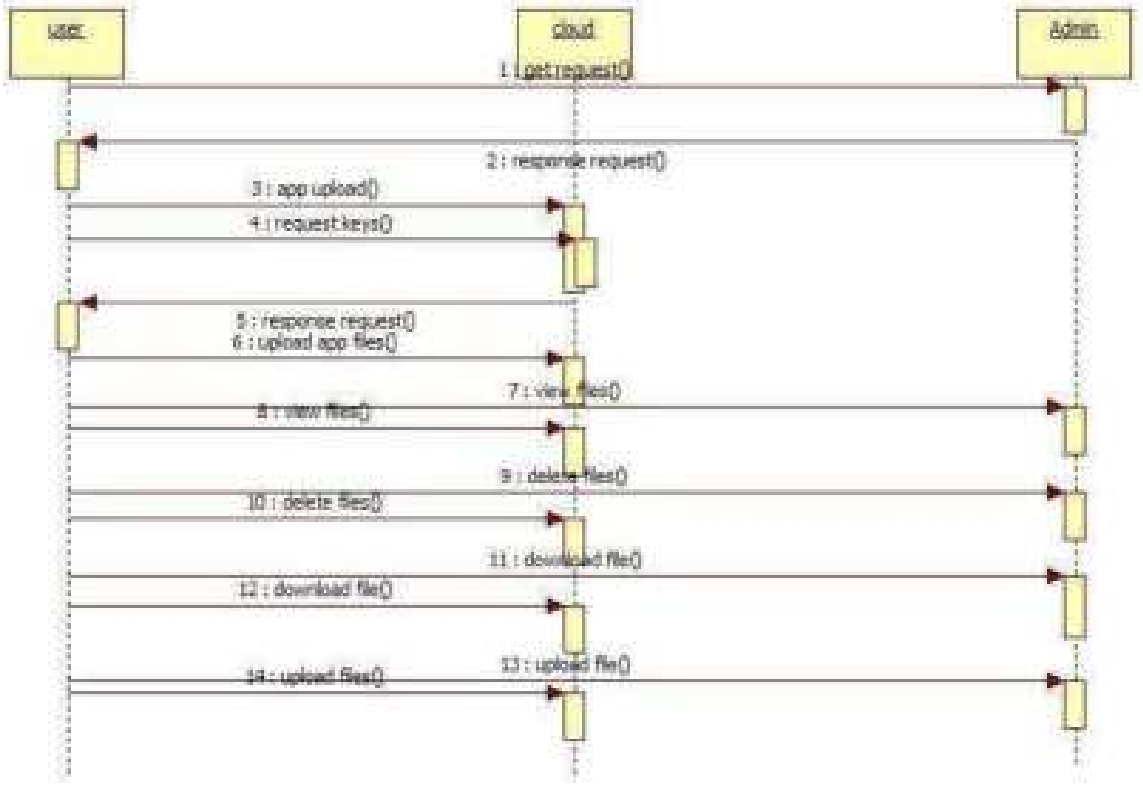


Fig no: 3.6 Sequence Diagram

3.4 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

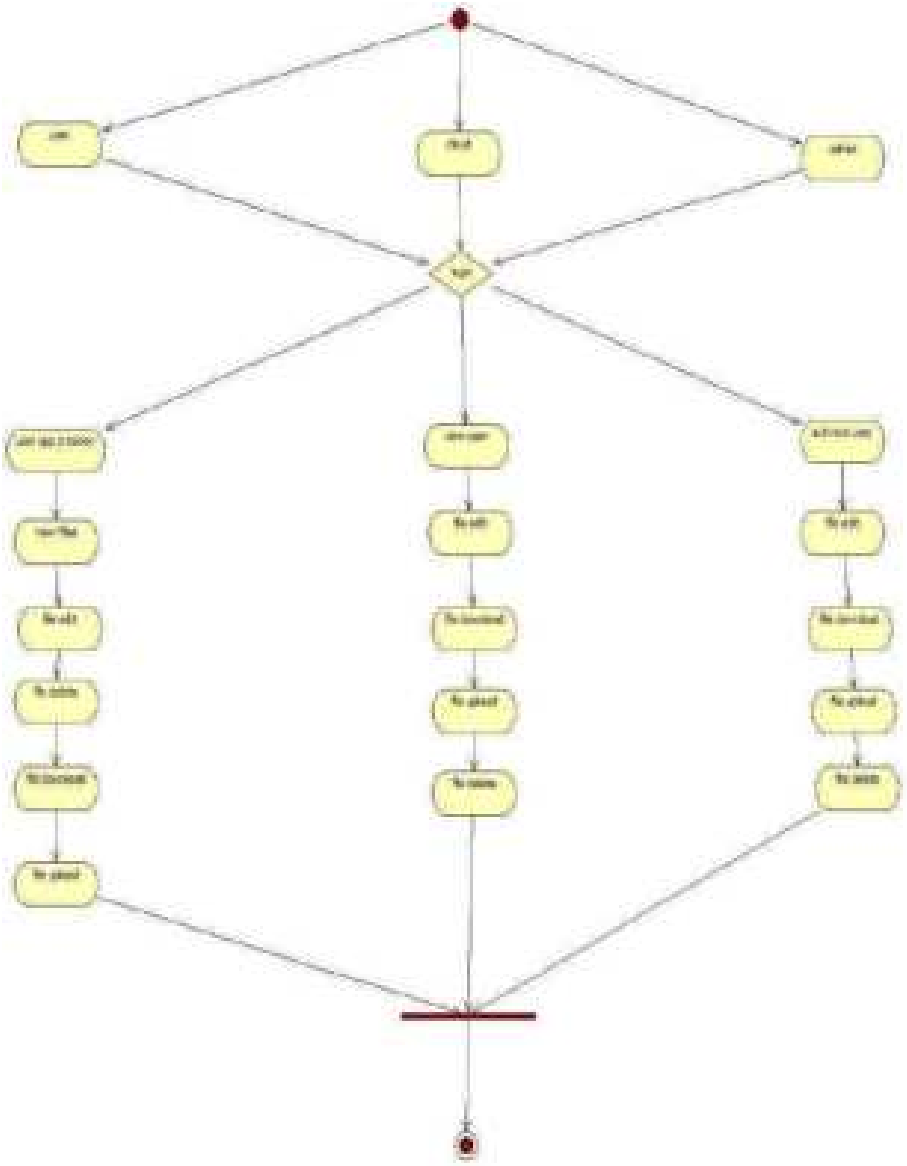


Fig no: 3.4 Use case Diagram

4. IMPLEMENTATION

4. IMPLEMENTATION

4.1 SAMPLE CODE

```
from django.contrib import admin
from django.urls import path from
django.conf import settings from
django.conf.urls.static import static from
users import views
from users.views import
index,userlogin,adminlogin,cloudlogin,userregister,storeregistration,logout,userlogincheck,us
ercreateapp,appcreaterequest,useruploadfile,snippet_detail
from admins.views import adminlogincheck,adminactivateusers,activatewaitedusers
from clouds.views import activateuserapp,cloudlogincheck,clouduserappactivations
from .views import resturl,downloadfile,deletefile,uploadfile
        urlpatterns = [ path('admin/', admin.site.urls),
path('',index,name='index'), path(r'accounts',
views.AccountAPIView.as_view(), name='account-list'), path(r'contacts',
views.ContactAPIView.as_view(), name='contact-list'), path(r'activities',
views.ActivityAPIView.as_view(), name='activity-list'),
        path(r'activitystatuses', views.ActivityStatusAPIView.as_view(), name='activity-
statuslist'),
        path(r'contactsources', views.ContactSourceAPIView.as_view(), name='contact-
sourcelist'),
        path(r'contactstatuses', views.ContactStatusAPIView.as_view(),
name='contact-status-list'), path(r'logout',logout,name='logout'),

        path(r'adminlogincheck',adminlogincheck,name='adminlogincheck'),
path(r'adminactivateusers',adminactivateusers,name='adminactivateusers'),
path(r'activatewaitedusers/<id>/$',activatewaitedusers,name='activatewaitedusers'),
```

```
        path(r'userlogin',userlogin,name='userlogin'), path(r'adminlogin',
adminlogin, name='adminlogin'), path(r'cloudlogin', cloudlogin,
name='cloudlogin'), path(r'userregister', userregister, name='userregister'),
path(r'storeregistration',storeregistration,name='storeregistration'),
path(r'userlogincheck', userlogincheck, name='userlogincheck'),
path(r'usercreateapp',usercreateapp,name='usercreateapp'),
path(r'appcreaterequest',appcreaterequest,name='appcreaterequest'),
path(r'useruploadfile/<appname>/$',useruploadfile,name='useruploadfile'),
path(r'^snippet_detail/$',snippet_detail,name='snippet_detail'),
        path(r'resturl/<id>',resturl,name='resturl'),
path(r'downloadfile/<id>',downloadfile,name='downloadfile'),
path(r'deletefile/<id>',deletefile,name='deletefile'),
path(r'uploadfile',uploadfile,name='uploadfile'),



          path(r'activateuserapp',activateuserapp,name='activateuserapp'),
path(r'cloudlogincheck',cloudlogincheck,name='cloudlogincheck'),
          path(r'clouduserappactivations/<appname>/$',clouduserappactivations,
name='clouduserappactivations'),
        ]
        if settings.DEBUG:
          urlpatterns+=static(settings.MEDIA_URL,document_root=settings.MEDIA_ROOT)
```

Cloud Side views.py

```
        from django.shortcuts import render,HttpResponse
        from rest_framework.views import APIView
        from rest_framework.decorators import api_view
        from rest_framework import generics
```

```
        from users.models import UserFileModel import os
         from django.conf import settings
        from django.contrib import messages
         from rest_framework import status
        from rest_framework.response import Response import
        os

        from django.http import HttpResponse, Http404
from users.models import UserAppCreatModel
        @api_view(['GET', 'PUT', 'DELETE','POST'])
        def      resturl(request,id):
role    =    request.session['role']
print('ROle  is  ',role)            if
request.method == 'GET':          if
role=='user':
                dict = {}
                data = UserFileModel.objects.get(id=id)
                filepath = data.userfile
file = str(filepath).split("/")
                rd=open(os.path.join(settings.MEDIA_ROOT+'/media/',
file[1]),'r',encoding='UTF-8',errors='ignore')
            filedata                                =                                rd.read()
dict.update({'id':id,'filename':file[1],'seckey':data.secretkey,'fdata':filedata})
return render(request,'users/editfilesdata.html',dict)      elif role=='admin':
                print('Admin resturl works fine')
                dict = {}
```

```
                data = UserFileModel.objects.get(id=id)
                filepath = data.userfile
        file = str(filepath).split("/")
                rd = open(os.path.join(settings.MEDIA_ROOT + '/media/', file[1]), 'r', encoding='UTF-8', errors='ignore')
                filedata = rd.read()        dict.update({'id': id, 'filename': file[1], 'seckey': data.secretkey, 'fdata': filedata})        return render(request, 'admin/admineditfilesdata.html', dict)
            elif role=='cloud':
                return Response(status=status.HTTP_405_METHOD_NOT_ALLOWED)
            else:
                print("Invalid URL")
elif request.method =='POST':
            fileid = request.POST.get('fileid')
filename = request.POST.get('filename')
filedata = request.POST.get('filedata')        with open(settings.MEDIA_ROOT+'/media'+'/'+filename, 'w+', encoding='UTF-8') as f:
f.write(filedata)
            return Response(status=status.HTTP_200_OK)
print('POST Request Executed')        print('User ID ',role,'File ID ',id)
            return HttpResponse('Am work fine')
        def downloadfile(request,id):
            data = UserFileModel.objects.get(id=id)
filepath = data.userfile
```

```
            # x1 = os.path.join(settings.MEDIA_ROOT+"//"+filepath)
          #   print('X1   path   =   ',filepath)
fppath = str(filepath).split("/")
            file_path = os.path.join(settings.MEDIA_ROOT+'/media/', fppath[1])
            if os.path.exists(file_path):
               with open(file_path, 'rb') as fh:
                  response   =   HttpResponse(fh.read(),   content_type="application/vnd.ms-
excel")                          response['Content-Disposition']   =   'inline;   filename='   +
os.path.basename(file_path)
                  return response
raise Http404
         @api_view(('GET',))
         def  deletefile(request,id):
            role                          =
request.session['role']       if  role ==
'user':
               data = UserFileModel.objects.get(id=id)
               data.delete()
               ##filepath = data.userfile
               #fpath = filepath #settings.MEDIA_ROOT+'/'+filepath
               #print('Removing FIle path is ',fpath)
               #os.remove(fpath)                          return
Response(status=status.HTTP_200_OK)
            elif role =='admin':
               return Response(status = status.HTTP_405_METHOD_NOT_ALLOWED)
```

```
            elif role =='cloud':                data =
UserFileModel.objects.get(id=id)
                data.delete()
                return Response(status = status.HTTP_200_OK)
        @api_view(('GET',))
        def uploadfile(request):
        role = request.session['role']
if role =='user':
                usremail = request.session['email']                dict =
UserAppCreatModel.objects.filter(email=usremail)                return
render(request,'users/uploadfile.html',{'objects':dict})          elif role
=='admin':
                return Response(status = status.HTTP_405_METHOD_NOT_ALLOWED)
            elif role =='cloud':
                return                          Response(status                          =
status.HTTP_405_METHOD_NOT_ALLOWED)
```

Cloud Side <u>models.py</u>

```
        from django.shortcuts import render,HttpResponse
        from django.contrib import messages
        from users.models import UserAppCreatModel
        import string
        import random
        # Create your views here.
        def cloudlogincheck(request):          if
request.method == "POST":              usid =
```

```
request.POST.get('name')                pswd =
request.POST.get('password')
        print("User ID is = ", usid)        if usid
== 'cloud' and pswd == 'cloud':
                request.session['role']         =         'cloud'
return render(request, 'clouds/cloudhome.html')
            else:
                messages.success(request, 'Invalid Login Details')
return render(request,'cloudlogin.html',{})
        def activateuserapp(request):
            dict    =    UserAppCreatModel.objects.all()                return
render(request,'clouds/userappactivation.html',{'objects':dict})
        def clouduserappactivations(request,appname):
            accessKey = genAccessToken(10)
            secretKey = genSecretKey(32)     print('App Name = ', appname,' Access Key
',accessKey,' Secret Key ',secretKey)
        UserAppCreatModel.objects.filter(appname=appname).update(accesskey=accessKey,sec
retk  ey=secretKey)        dict  =  UserAppCreatModel.objects.all()        return render(request,
'clouds/userappactivation.html', {'objects': dict})
        def genAccessToken(stringLength=10):
            letters     =     string.ascii_lowercase                return
''.join(random.choice(letters) for i in range(stringLength))
        def genSecretKey(stringLength=32):
            """Generate a random string of letters and digits """ lettersAndDigits =
string.ascii_letters + string.digits     return  ''.join(random.choice(lettersAndDigits) for i
in range(stringLength))
```

User side Views.py from django.db

import models from django.contrib.auth.models

import User import os

```
INDCHOICES =
  ( ('FINANCE',
  'FINANCE'),
  ('HEALTHCARE', 'HEALTHCARE'),
  ('INSURANCE', 'INSURANCE'),
  ('LEGAL', 'LEGAL'),
  ('MANUFACTURING', 'MANUFACTURING'),
  ('PUBLISHING', 'PUBLISHING'),
  ('REAL ESTATE', 'REAL ESTATE'),
  ('SOFTWARE', 'SOFTWARE'),
 )
 class Account(models.Model):
      name = models.CharField("Name of Account", "name", max_length=64) email = models.EmailField(blank = True, null = True) phone = models.CharField(max_length=20, blank = True, null = True)
      industry = models.CharField("Industry Type", max_length=255, choices=INDCHOICES, blank=True, null=True) website = models.URLField("Website", blank=True, null=True)
    description = models.TextField(blank=True, null=True)
      createdBy = models.ForeignKey(User, related_name='account_created_by', on_delete=models.CASCADE)
      createdAt = models.DateTimeField("Created At", auto_now_add=True) isActive = models.BooleanField(default=False)
      def__str__(self): return self.name
    class ContactSource(models.Model):
      CMRTC
```

```
status = models.CharField("Contact Source", max_length=20)

        def  _str__(self):

return  self.status  class

ContactStatus(models.Model):

        status = models.CharField("Contact Status", max_length=20)

        def  __str_(self):

return self.status

    class Contact(models.Model):

        first_name = models.CharField("First name", max_length=255, blank =

True, null = True)  last_name = models.CharField("Last name",

max_length=255, blank = True, null = True)

        account = models.ForeignKey(Account,
related_name='lead_account_contacts',  on_delete=models.CASCADE,
blank=True, null=True)

    email = models.EmailField()  phone =

models.CharField(max_length=20, blank = True, null = True) address =

models.TextField(blank=True, null=True)  description =

models.TextField(blank=True, null=True)

        createdBy = models.ForeignKey(User, related_name='contact_created_by',
on_delete=models.CASCADE)

        createdAt = models.DateTimeField("Created At", auto_now_add=True)

isActive = models.BooleanField(default=False)

        def_str_(self):

          return self.first_name

    class ActivityStatus(models.Model):

        status = models.CharField("Activity Status", max_length=20)

        def  __str_(self):

return self.status
```

```
class Activity(models.Model):
    description = models.TextField(blank=True, null=True)          createdAt =
models.DateTimeField("Created At", auto_now_add=True)
    contact = models.ForeignKey(Contact, on_delete=models.CASCADE, null=True)
    def str (self):
        return self.description
class CloudUsersModel(models.Model):
    id = models.AutoField(primary_key=True)                name =
models.CharField(max_length=200)                          email =
models.CharField(max_length=100,unique=True)              password =
models.CharField(max_length=100)                          mobile =
models.CharField(max_length=100)                          address =
models.TextField(max_length=100)                          city =
models.CharField(max_length=100)                          state =
models.CharField(max_length=100)                          status =
models.CharField(max_length=100,default='waiting')
    def __str__(self):            return
self.email
    class Meta:
        db_table = "registrations"
class UserAppCreatModel(models.Model):              id =
models.AutoField(primary_key=True)                        name =
models.CharField(max_length=200)                          email =
models.CharField(max_length=200)                          appname =
models.CharField(max_length=200,unique=True)      accesskey =
models.CharField(max_length=200,default='waiting')  secretkey =
models.CharField(max_length=200,default='waiting')
```

```
        def
__str__(self):          return
self.appname            class
Meta:
            db_table = "userapps"
        class UserFileModel(models.Model):
            id = models.AutoField(primary_key=True)          name                =
models.CharField(max_length=200)                  email                =
models.CharField(max_length=200)              appname              =
models.CharField(max_length=200)           accesskey            =
models.CharField(max_length=200)           secretkey            =
models.CharField(max_length=200)           filename             =
models.CharField(max_length=200)             userfile             =
models.FileField(upload_to='media/')
        def_str_(self):
            return
os.path.basename(self.userfile.name)    class Meta:
            db_table = "userfiles"

        def delete(self, *args, **kwargs):
self.userfile.delete()      super().delete(*args,
**kwargs)
```

forms.py

```
        from django import forms from .models import
CloudUsersModel,UserFileModel                class
CloudUserFrom(forms.ModelForm):
        name        =        forms.CharField(widget=forms.TextInput(attrs={'size':50,'class':
'special'}), required=True,max_length=100 )
        email       =       forms.CharField(widget=forms.TextInput(attrs={'size':50}),
required=True, max_length=100)
```

```
        password = forms.CharField(widget=forms.PasswordInput(attrs={'size':50}),
required=True,max_length=100)

      mobile = forms.CharField(widget=forms.TextInput(attrs={'size':50}),
required=True,max_length=100)

        address = forms.CharField(widget=forms.Textarea(attrs={'rows':
3,      'cols': 52}), required=True,max_length=250)

        city = forms.CharField(widget=forms.TextInput(attrs={'size':50}),
required=True,max_length=100)

        state = forms.CharField(widget=forms.TextInput(attrs={'size':50}),
required=True,max_length=100)      status = forms.CharField(widget=forms.HiddenInput(),
initial='waiting', max_length=100)


        class Meta():

          model = CloudUsersModel

          fields=['name','email','password','mobile','address','city','state','status']

      class UserFileForm(forms.ModelForm):

        name = forms.CharField(max_length=100 )

        #email = forms.CharField(max_length=200)

        #appname = forms.CharField(max_length=200)

        #accesskey = forms.CharField(max_length=200)

        #secretkey = forms.CharField(max_length=200)

        #filename = forms.CharField(max_length=200)

        #userfile =
forms.FileField()   class Meta():

          model = UserFileModel

          fields = '_all_'
```

admin.py

```
from django.contrib import admin

from users.models import

Contact,Account,ContactSource,ContactStatus,ActivityStatus,Activity,CloudU
sersModel,Use rFileModel

# Register your models here.


admin.site.register(Contact)
admin.site.register(Account)
admin.site.register(ContactSource)
admin.site.register(ContactStatus)
admin.site.register(ActivityStatus)
admin.site.register(Activity)
admin.site.register(CloudUsersModel)
admin.site.register(UserFileModel)
```

user side views.py

```
from django.shortcuts import
render,HttpResponseRedirect,HttpResponse from
rest_framework import generics import json

from .models import Account, Activity, ActivityStatus,
Contact, ContactSource, ContactStatus

from .serializers import AccountSerializer, ActivitySerializer, ActivityStatusSerializer,
ContactSerializer, ContactSourceSerializer, ContactStatusSerializer,UserFileModelSerializer
from rest_framework.decorators import api_view # Create your views here.

from rest_framework import
generics from django.http import
JsonResponse

from .models import Account, Activity, ActivityStatus, Contact,
ContactSource, ContactStatus
```

```
from .serializers import AccountSerializer, ActivitySerializer, ActivityStatusSerializer, ContactSerializer, ContactSourceSerializer, ContactStatusSerializer

from rest_framework.renderers import TemplateHTMLRenderer from rest_framework.response import Response from rest_framework.views import APIView from rest_framework import serializers


from .forms import CloudUserFrom,UserFileForm from .models import CloudUsersModel from django.contrib import messages from rest_framework import status from .models import CloudUsersModel,UserAppCreatModel,UserFileModel #@api_view(['GET', 'POST']) class AccountAPIView(generics.ListCreateAPIView):
    queryset = Account.objects.all()
serializer_class = AccountSerializer
renderer_classes = [TemplateHTMLRenderer]
    template_name = 'profile_list.html'


    def get(self, request):
        queryset = Account.objects.all()
return Response({'profiles': queryset})


    def post(self, request, pk):
        print('AM going to Execute atleast once in my life') profile = get_object_or_404(Account, pk=pk) serializer = AccountSerializer(profile, data=request.data)
        if not serializer.is_valid():
```

```
                return Response({'serializer': serializer, 'profile':
Account})        serializer.save()        return redirect('account-list')

        class ActivityAPIView(generics.ListCreateAPIView):
            queryset        =        Activity.objects.all()
serializer_class = ActivitySerializer

        class ActivityStatusAPIView(generics.ListCreateAPIView):
            queryset = ActivityStatus.objects.all()
            serializer_class = ActivitySerializer

        class ContactAPIView(generics.ListCreateAPIView):
            queryset        =        Contact.objects.all()
serializer_class = ContactSerializer

        class ContactStatusAPIView(generics.ListCreateAPIView):
            queryset        =        ContactStatus.objects.all()
serializer_class = ContactSerializer

        class ContactSourceAPIView(generics.ListCreateAPIView):
            queryset        =        ContactSource.objects.all()
serializer_class = ContactSourceSerializer
        def index(request):
            return
render(request,'base.html',{})              def
userlogin(request):
```

```
        return
render(request,'userlogin.html',{})          def
adminlogin(request):
        return
render(request,'adminlogin.html',{})          def
cloudlogin(request):
        return
render(request,'cloudlogin.html',{})          def
userregister(request):
        return render(request,'userregister.html',{})


    @api_view(['GET', 'POST'])
    def
storeregistration(request):    if
request.method == 'POST':
            form = CloudUserFrom(request.POST)
            if form.is_valid():
               try:
                  rslt = form.save()              print("Form Result Status ",
rslt)                 messages.success(request, 'You have been successfully
registered')
               except:
                  messages.success(request,    'Email    Already    Registerd')
return render(request, 'userregister.html',{})
    else:
print("Invalid form")    else:
```

```
            form = CloudUserFrom()              return
render(request, 'userregister.html', {'form': form})


        def
userlogincheck(request):        if
request.method == "POST":
            email    =    request.POST.get('cf-
email')            pswd = request.POST.get('cf-
password')       print("Email = ", email)
            try:
               check = CloudUsersModel.objects.get(email=email, password=pswd)
               request.session['id']            =            check.id
request.session['loggeduser']              =              check.name
request.session['email']                   =                   check.email
request.session['role']='user'          status = check.status          if
status == "activated":
                  print("User  id  At",  check.id,  status)
return render(request, 'users/userpage.html', {})
               else:
                  messages.success(request, 'Your Account Not at activated')
return render(request, 'userlogin.html')


            return                 render(request,
'userlogin.html', {})       except:          pass
          messages.success(request, 'Invalid Email  id  and  password')       return
render(request, 'userlogin.html')
       @api_view(['GET', 'PUT', 'DELETE','POST'])
```

```
        def
snippet_detail(request):       role
= request.session['role']
            try:
               snippet = UserFileModel.objects.all()
               #print('Type is ',snippet.id)
            except CloudUsersModel.DoesNotExist:
                return
        Response(status=status.HTTP_404_NOT_FOUND)
            if request.method == 'GET':
               if role == 'user':
                  print('Get Method Works Fine')                usremail =
request.session['email']                                        queryset =
UserFileModel.objects.filter(email=usremail)              serializer_class =
UserFileModelSerializer              print('Return Type is ',serializer_class)
return          render(request,'users/uploadedfiles.html',{'objects':queryset})
elif role=='admin':
                  queryset = UserFileModel.objects.all()              serializer_class =
UserFileModelSerializer                                    return    render(request,
'admin/adminuploadedfiles.html', {'objects': queryset})
        elif role =='cloud':
                  queryset = UserFileModel.objects.all()              serializer_class =
UserFileModelSerializer                                    return    render(request,
'clouds/clouduploadedfiles.html', {'objects': queryset})


            elif request.method == 'PUT':
```

```
            print('PUT Method Works Fine')          serializer =
UserFileModelSerializer(snippet, data=request.data)
            if serializer.is_valid():
               serializer.save()
return Response(serializer.data)
            return                                  Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)          elif request.method ==
'DELETE':

            print('DELETE Method Works Fine')
            #snippet.delete()
            return Response(status=status.HTTP_204_NO_CONTENT)


        elif request.method=="POST":
            print('POST Method Works Fine')       form
= UserFileForm(request.POST, request.FILES)
            if      form.is_valid():
form.save()
            else:
               print('Invalid Form')
            return Response(status=status.HTTP_201_CREATED)


    def logout(request):
    request.session.modified = True
return render(request,'base.html',{})
    def usercreateapp(request):
```

```
        usremail = request.session['email']                dict =
UserAppCreatModel.objects.filter(email=usremail)              return
render(request,'users/userappcreations.html',{'objects':dict})
    def
appcreaterequest(request):      if
request.method=='POST':
        usrname = request.POST.get('usrname')
        usremail = request.POST.get('usremail')
        appname = request.POST.get('appname')
        accesskey = request.POST.get('accesskey')
secretkey = request.POST.get('secretkey')
        try:
    UserAppCreatModel.objects.create(name=usrname,email=usremail,appname=appnam
e)
            messages.success(request, 'Your App creation Request is Under Process')
        except:
            messages.success(request, 'App Name Already exist')
            pass
        print(usrname,usremail,appname,accesskey,secretkey)      dict      =
UserAppCreatModel.objects.filter(email=usremail                  return
render(request,'users/userappcreations.html',{'objects':dict})
    def useruploadfile(request,appname):
        check                                                =
UserAppCreatModel.objects.get(appname=appname)      acckey =
check.accesskey      secretkey = check.secretkey      dict =
{'appname':appname,'acckey':acckey,'secretkey':secretkey}
return render(request,'users/uploaddatatocloud.html',dict)
```

admin.py

```
        # Create your views here. def
adminlogincheck(request):                   if
request.method == "POST":          usid =
request.POST.get('name')           pswd =
request.POST.get('password')
              print("User ID is = ", usid)
if usid == 'admin' and pswd == 'admin':
                 request.session['role'] = 'admin'                    return render(request,
'admin/adminhome.html')
              else:
                 messages.success(request, 'Invalid Login Details')
return render(request,'adminlogin.html',{})


        def adminactivateusers(request):
           dict = CloudUsersModel.objects.all()          return
render(request,'admin/activateusers.html',{'objects':dict})


        def
activatewaitedusers(request,id):
if request.method == 'GET':
#uid = request.GET.get('uid')
      status = 'activated' print("PID
      =          ",          id,status)
      CloudUsersModel.objects.filt
      er(id=id).update(status=statu
      s)                  dict =
      CloudUsersModel.objects.all
```

```
()        return render(request,
'admin/activateusers.html',
{'objects': dict})
```

clouduploadfiles.html

```
{% extends 'clouds/cloudbase.html'%}
{% load static %}
{% block contents%}
<div id="qbootstrap-testimonial" class="qbootstrap-bg-section">
        <div class="container">
                <div class="row animate-box">
              <span><h1>Cloud Server View Uploaded Files of Users</h1></span>
                                <p>
<tr style="color: darkblue">
                                <th>S.No</th>
                                <th>Name</th>
                                <th>Email</th>
                                <th>App Name</th>
                                <th>File Name</th>
                                <th>Server path</th>
                                <th>Edit</th>
                                <th>Delete</th>
                                <th>Download</th>
                                <th>Upload New</th>
                            </tr>
                            {% for i in objects %}
```

```
                                    <tr style="color: RED">
                                        <td>{{forloop.counter}}</td>
                                        <td>{{i.name}}</td>
                                        <td>{{i.email}}</td>
                                        <td>{{i.appname}}</td>
                                        <td>{{i.filename}}</td>
                                            <td>{{i.userfile}}</td>
                                            <td><a class="btn btn-primary btn-md"
href="{%url
                                    </tr>
                                    {% endfor %}
                                </table>
                        </p>
                                </div>
                            </div>
    </div>
    {% endblock %}
```

Cloudbase.html

```
    {% extends 'clouds/cloudbase.html' %}
    {% load static %}
    {% block contents %}
        <div id="qbootstrap-counter" class="qbootstrap-counters"
style="background-image: url({% static 'images/img_bg_2.jpg'%});" data-
stellar-background-ratio="0.5">
```

```
<div class="overlay"></div>

<div class="container">

<div class="row">

<div class="col-md-10 col-md-offset-1">

<div class="row">

<div class="col-md-3 col-sm-6 <span class="icon"><i class="icongroup-outline"></i></span>

<span class="qbootstrap-counter jscounter" data-from="0" data-to="65535" data-speed="5000" data-refreshinterval="50"></span>

<span class="qbootstrap-counterlabel">Satisfied Customer</span>

</div>

<div class="col-md-3 col-sm-6 text-center animate-box">

<span class="icon"><i class="iconhome-outline"></i></span>

<span class="qbootstrap-counter jscounter" data-from="0" data-to="378" data-speed="5000" data-refreshinterval="50"></span>

<span class="qbootstrap-counterlabel">Cloud Hosts</span>

</div>

<div class="col-md-3 col-sm-6 text-center animate-box">

<span class="icon"><i class="icon-useradd-outline"></i></span>

<span class="qbootstrap-counter jscounter" data-from="0" data-to="400" data-speed="5000" data-refresh-
interval="50"></span>
```

```
                                        <span class="qbootstrap-counter-
label">Qualified
Professionals</span>
                                    </div>
                                    <div class="col-md-3 col-
                                    sm-6 text-center
animate-box">
                                        <span class="icon"><i
                                    class="icon-
point-of-interest-
outline"></i></span>
                                        <span class="qbootstrap-
                                    counter js-
        counter" data-from="0" data-to="30" data-speed="5000" data-
refresh-interval="50"></span>
            <span class="qbootstrap-counterlabel">Nodes</span>
                                    </div>
                                </div>
                            </div>
                        </div>
            </div>
    </div>
        {% endblock %}
```

Useruploadfile.html

```
        {% extends 'users/userbase.html'%}
        {% load static %}
        {% block contents%}
        <div id="qbootstrap-testimonial" class="qbootstrap-bg-section">
            <div class="container">
                    <div class="row animate-box">
```

```
                                    <p>
                                    <h1>Created Apps and upload file</h1>
                                    <table border="2px">
                                      <tr style="color: darkblue">
                                        <th>S.No</th>
                                        <th>Name</th>
                                        <th>Email</th>
                                        <th>App Name</th>
                                        <th>Access Key</th>
                                        <th>Token Key</th>
        <th>Upload Data</th>
                                      </tr>
                                      {% for i in objects %}
                                      <tr style="color: RED">
                                        <td>{{forloop.counter}}</td>
                                        <td>{{i.name}}</td>
                                        <td>{{i.email}}</td>
                                        <td>{{i.appname}}</td>
                                        <td>{{i.accesskey}}</td>
                                        <td>{{i.secretkey}}</td>
                                        {% if i.secretkey != 'waiting' %}
                                        <td><a     class="btn-link"
   href="{%      url      'useruploadfile'      i.appname      %}"
style="color:GREEN">Upload Files</a></td>
                                        {% else %}
                                        <td style="color:Yellow"> Key Not
Generated</td>
```

```
                                    {% endif %}
                                </tr>
                                {% endfor %}
                            </table>
                        </p>
                    </div>
                </div>
            </div>
        {% endblock %}


        {% load static %}
        {% block contents %}
        <aside id="qbootstrap-hero">
            <div class="flexslider">
                <ul class="slides">
                <li style="background-image: url({%static
        'images/cldadmin.webp'%});">
                                    <div
                                    class="overlay"
                                    ></div>
                                    <div
                                    class="containe
                                    r">
                                      <div
                                    class="row">
                                            <div class="col-md-
                                                                t
                                                                e
                                                                x
c                                                               t
e                                                               -
n
t
```

```
e
r
sl
i
d
e
r-
t
e
x
t"
>


                                                    <div class="sli
                        <h2>Cloud Server Login here</h2>

                    <form action="{%url 'cloudlogincheck'%}"
method="POST" class="contactform">

                                                {% csrf_token %}

                                            <div class="form-group">

                                                    <label  for="name"
   class="sronly">Cloud Name</label>

                                                    <input type="text"
   name="name" style="background-color : #d1d1d1;"  class="form-control"
id="name" placeholder="Enter Login Name">

                                            </div>

                                            <div class="form-group">

                                                    <label  for="email"
   class="sronly">Email</label>
```

Cloudlogin.html

```
        <input type="password" name="password"  style="background-
color : #d1d1d1;" class="form-control" id="email" placeholder="Enter
Password ">

                                            </div>

                                            <div class="form-group">
```

```
                                        <input    type="submit"    id="btn-
submit"
        class="btn btn-primary btn-send-message btn-md" value="Login">
                                        </div>


                                    </form>

                                   </div>

                                 </div>


                                    </div>

                                   </div>


                    </li>

                    </ul>
            </div>

  </aside>          This is the Index page                    {% endblock %}
```

5. SCREENSHOTS

5. SCREENSHOTS

Index page

Fig no: 5.1 Index page

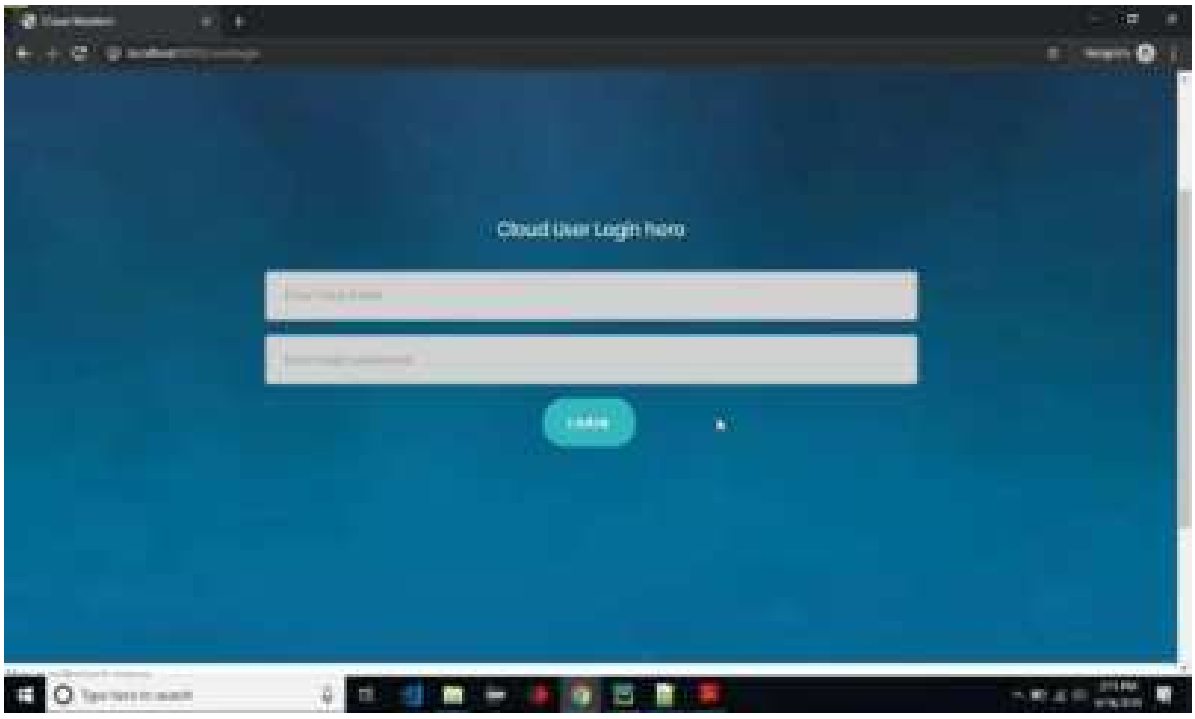


Fig no: 5.2 user login

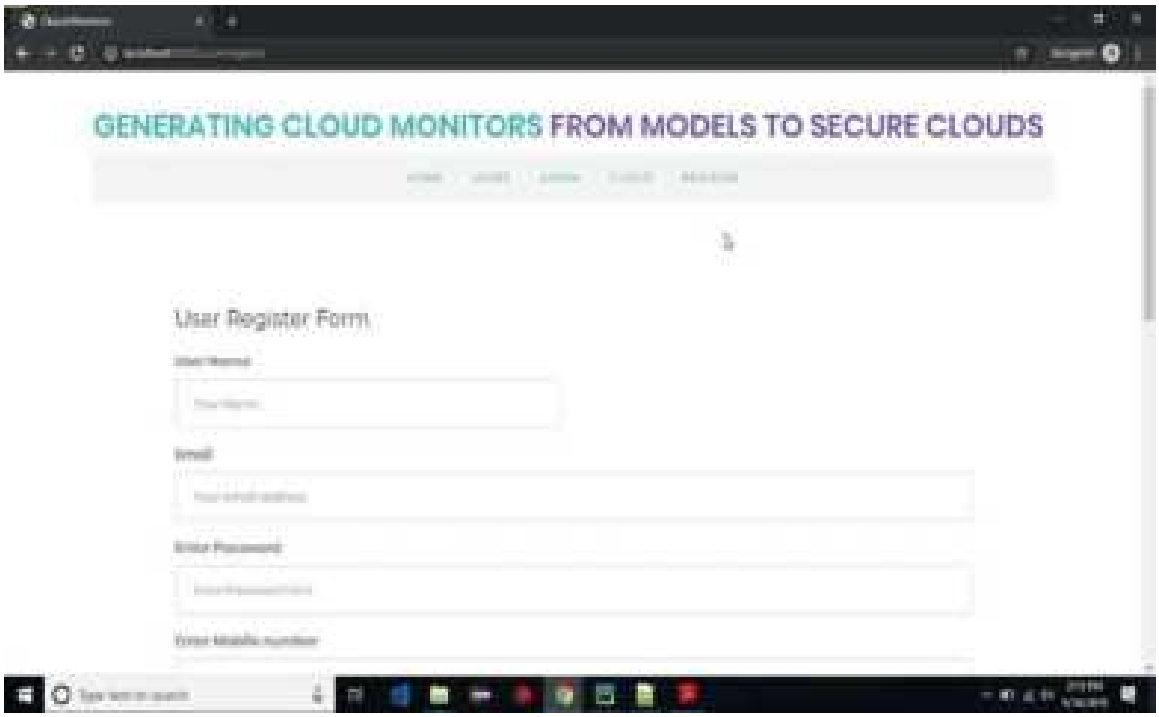


Fig no: 5.3 User register

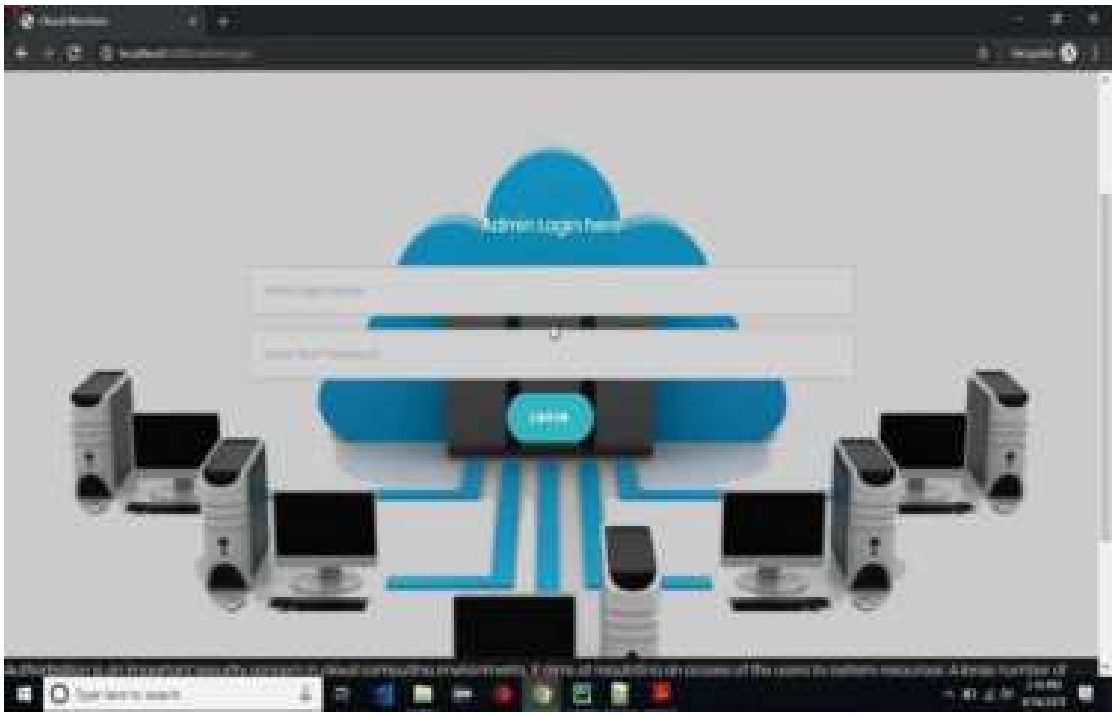


Fig no: 5.4 Admin login

Fig no: 5.5 Admin home page

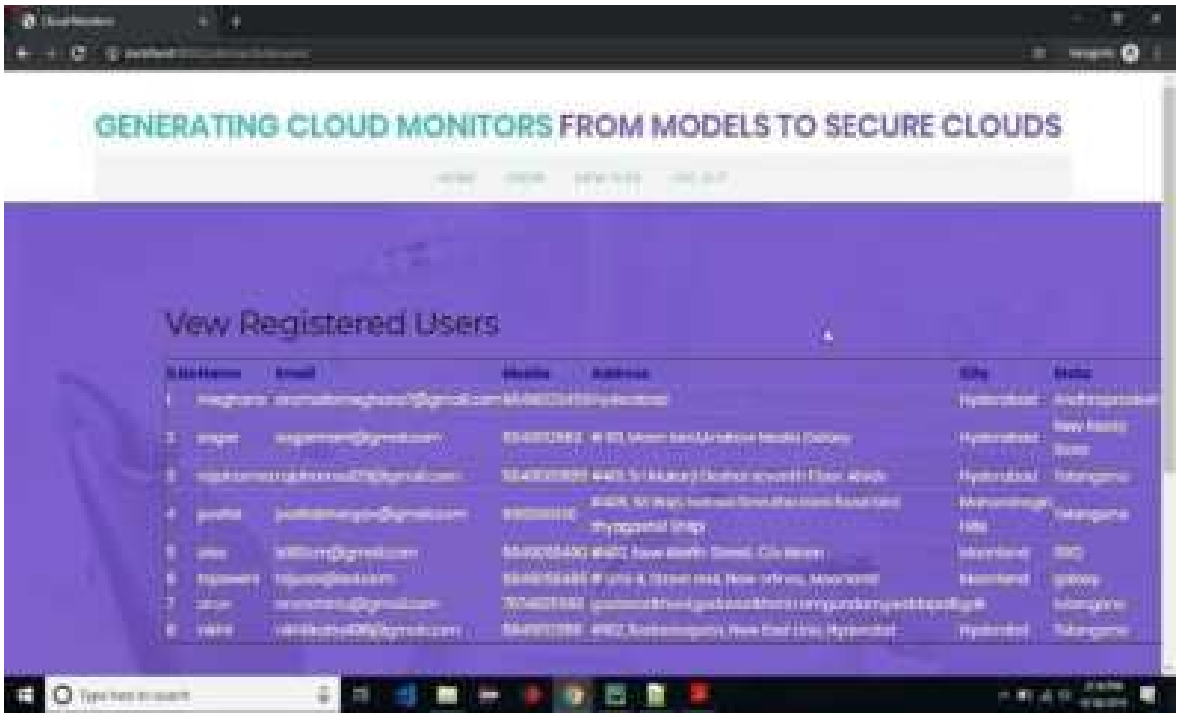


Fig no: 5.6 user app creation

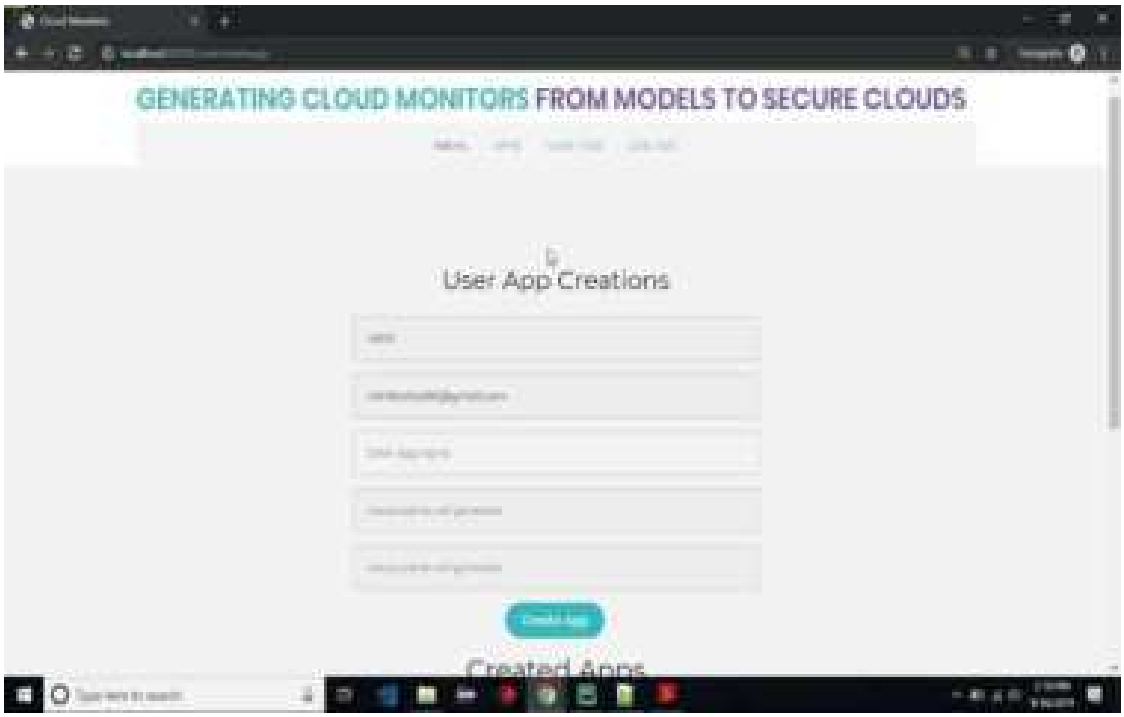


Fig no: 5.7 Django rest

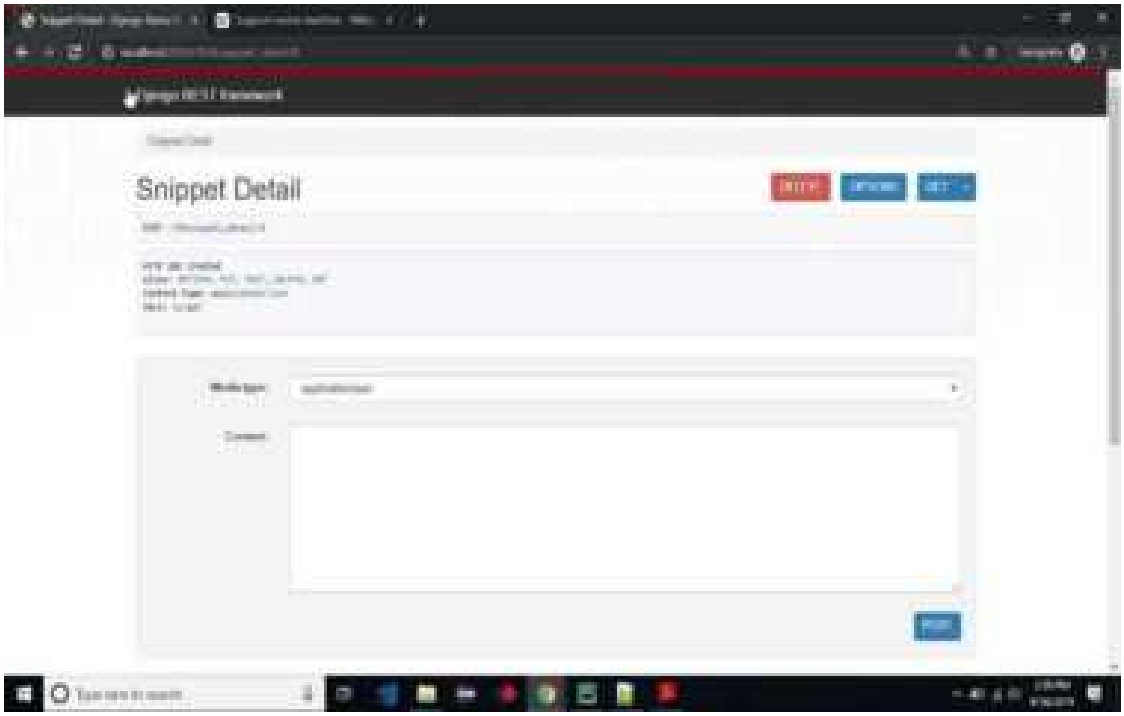


Fig no:5.8 User app check

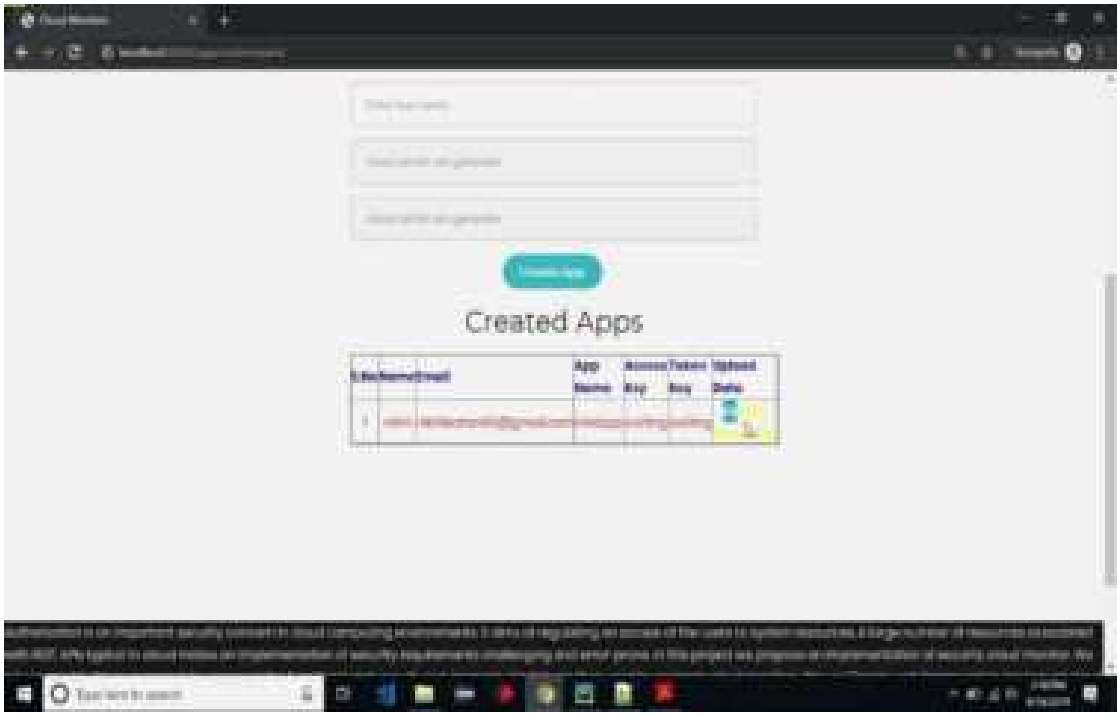


Fig no: 5.9 Cloud login

Fig no: 5.10 Cloud approve app

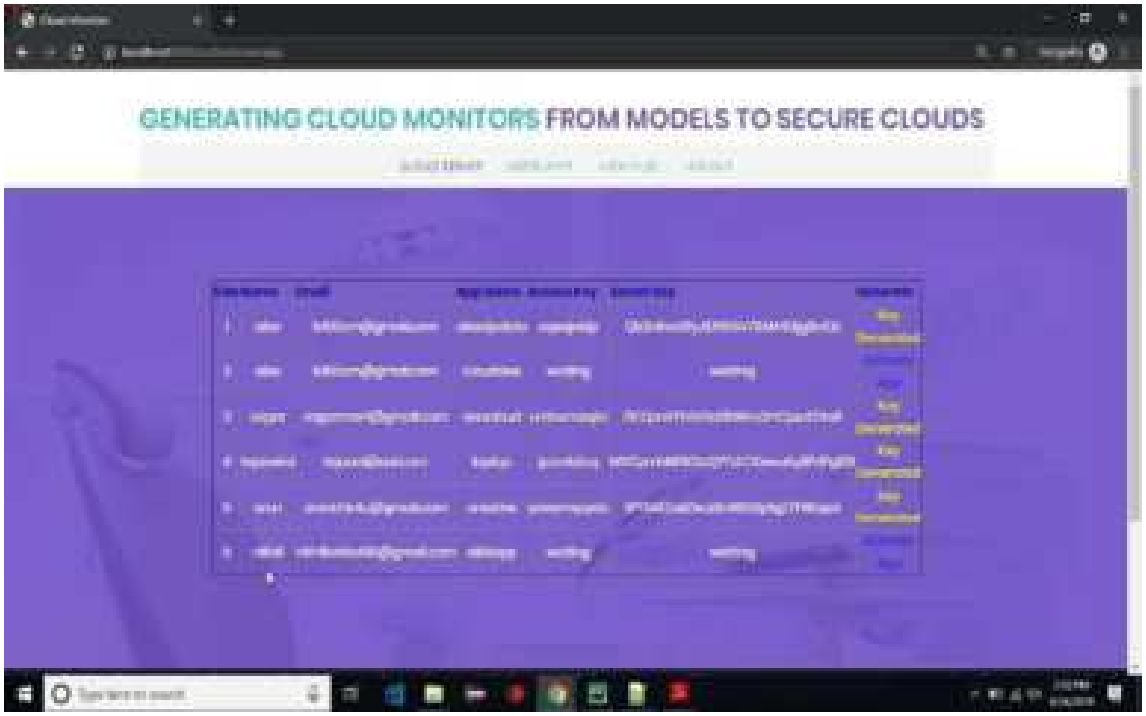


Fig no: 5.11 User Uploaded file

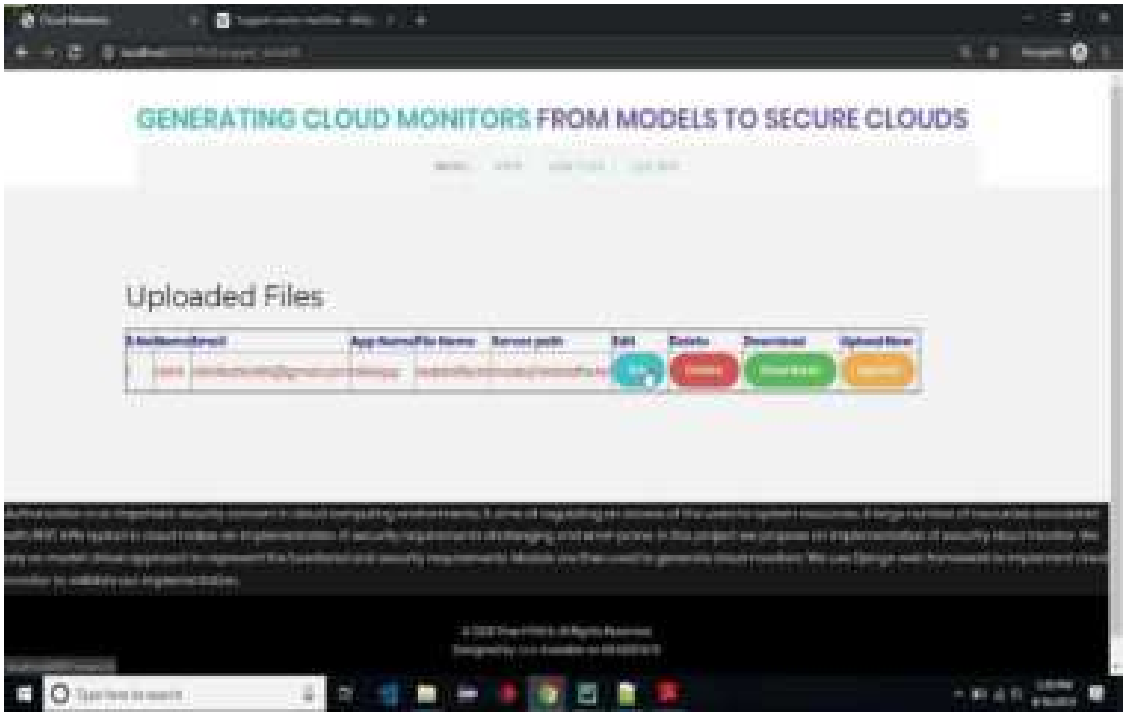
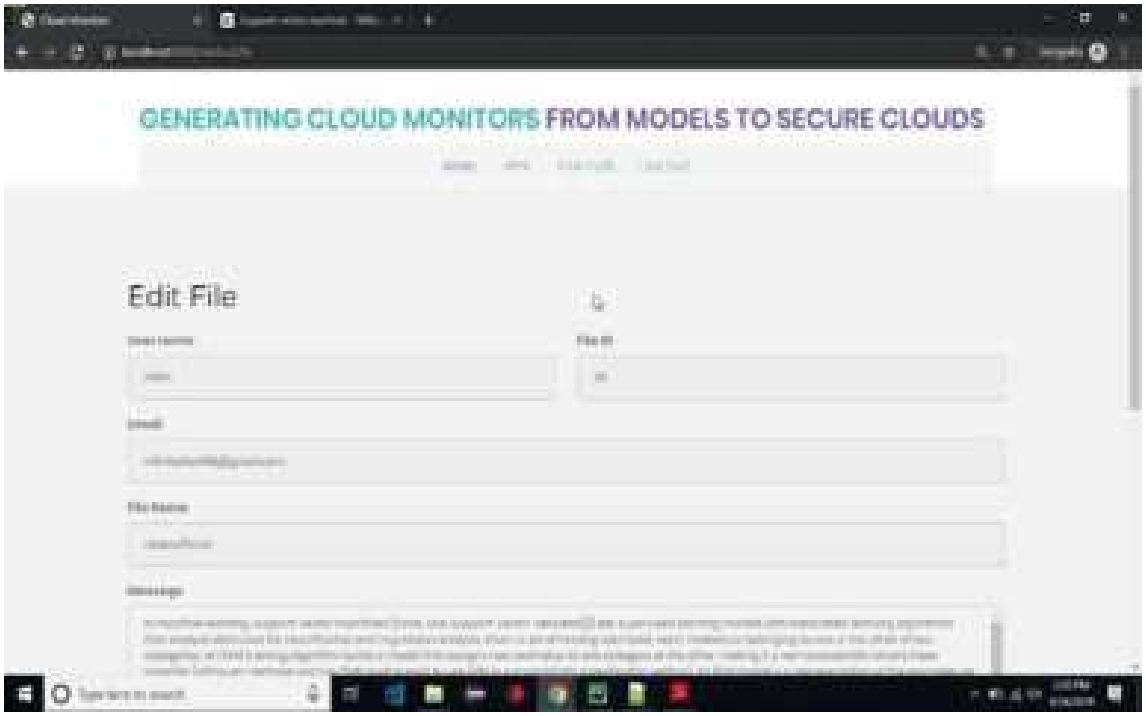
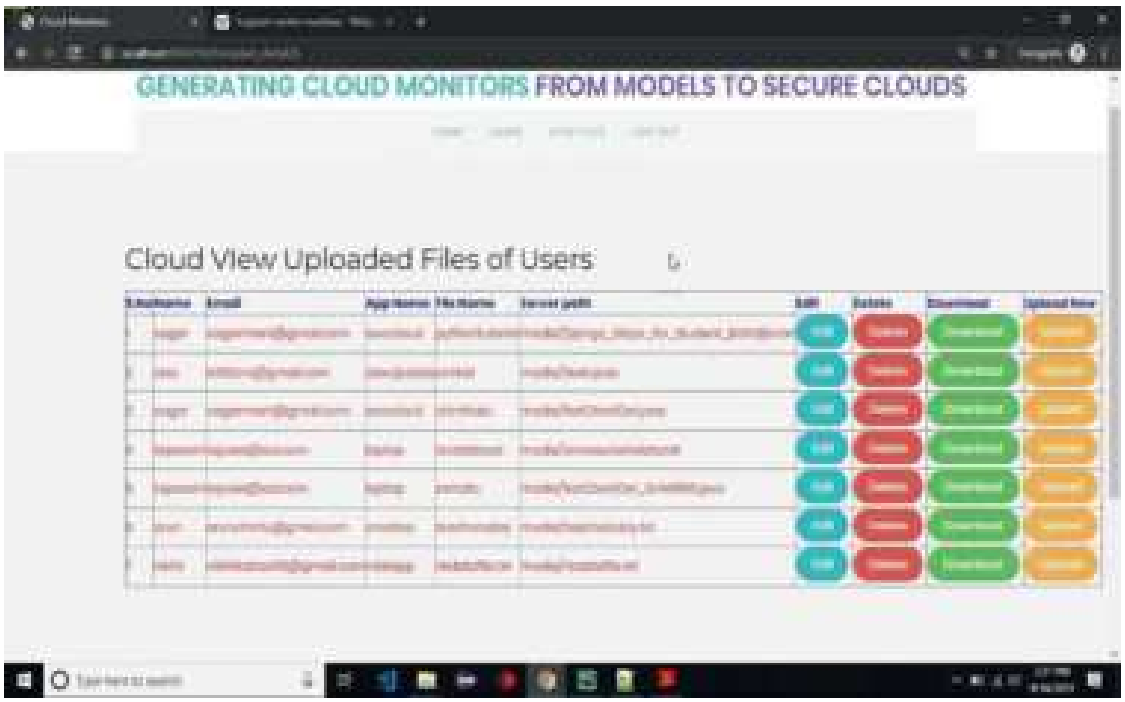


Fig no:5.12 Edit File


GENERATING CLOUD MONITORS FROM MODELS TO SECURE CLOUDS
Edit File


GENERATING CLOUD MONITORS FROM MODELS TO SECURE CLOUDS
Cloud View Uploaded Files of Users

6. TESTING

6. TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover very conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring thatthe Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components

6.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes.

6.3 TESTCASES

S.no	Test Case	Excepted Result	Result	Remarks(IF Fails)
1	USER REGISTERED	If USER registration successfully .	Pass	If USER is not registered.
2	USER LOGIN	If USER name and password is correct then it will getting valid page.	Pass	If USER name or password is not correct.
3	ADMIN	USER rights will be accepted here.	Pass	If USER are not registered.
4	USER upload files	Choose or select USER files	Pass	If USER is not select or SEND MESSAGES
5	cloud	USER app rights will be accepted here.	Pass	If USER app are not registered.
6	user	User can edit user uploaded file	Pass	If file is not available.
7	user	User can delete user uploaded file	Pass	If file is not available.
8	user	User can download user uploaded file	Pass	If file is not available.
9	user	User can upload user uploaded file	Pass	If file is not available.

7. CONCLUSION

7. CONCLUSION

In this paper, we have presented an approach and associated tool for monitoring security in cloud. We have relied on the model-driven approach to design APIs that exhibit REST interface features. The cloud monitors, generated from the models, enable an automated contract-based verification of correctness of functional and security requirements, which are implemented by a private cloud infrastructure. The proposed semi-automated approach aimed at helping the cloud developers and security experts to identify the security loopholes in the implementation by relying on modelling rather than manual code inspection or testing. It helps to spot the errors that might be exploited in data breaches or privilege escalation attacks. Since open source cloud frameworks usually undergo frequent changes, the automated nature of our approach allows the developers to relatively easily check whether functional and security requirements have been preserved in new releases.

8. REFERENCES

9. JOURNAL

Generating Cloud Monitors From Models To Secure Clouds

Dr. Madhukar G[1], Mounika D[2], Sai Kiran S[3], Sai Kumar Goud M[4]

[1]Assistant Professor, Dept. of Computer Science and Engineering, CMR Technical Campus, India
[2,3,4]B. Tech Student, Dept. of Computer Science and Engineering, CMR Technical Campus, India

----------**********----------

ABSTRACT

Our health care provider is also moving from paper records to electronic health records (EHRs) or could also be using EHRs already. EHRs permit providers to use information more effectively to enhance the standard and potency of your care, however EHRs won't modify the privacy protections or security safeguards that apply to your health info. This project focuses on developing secure cloud framework for evolving and accessing trustworthy computing services altogether levels of public cloud deployment model. Thus, eliminates each internal and external security threats. These ends up in achieving data confidentiality, data integrity, authentication and authorization, eliminating each active and passive attacks from cloud network surroundings. To develop a secure cloud framework for accessing sure computing and storage services altogether levels of public cloud preparation model.

Keywords: Cloud framework, Data security, Authentication, authorization, and data confidentiality.

I. INTRODUCTION

With the proliferation of data, it is a heavy burden for users to store huge amounts of data locally. Therefore, more and more organizations and individuals want to store their data in the cloud. However, data stored in the cloud can be corrupted or lost due to inevitable software failures, hardware failures, and human error in the cloud. Many remote data integrity check schemes have been proposed to ensure that data is stored correctly in the cloud. For remote data integrity verification schemes, the owner of data must first generate a signature for the block of data before uploading it to the cloud. These signatures are used to prove that the cloud owns these blocks of data during the integrity check phase. The data owner then uploads these data blocks to the cloud with their corresponding signatures.

Data stored in the cloud is often shared among multiple users of many cloud storage applications such as Google Drive, Dropbox, and iCloud. Data sharing as one of the most common features of cloud storage allows many users to share their data with others. However, this shared data stored in the cloud may contain sensitive information. For example, an Electronic Health Record stored and shared in the cloud typically contains patient sensitive information (patient name, phone number, ID number, etc.) and hospital sensitive information (hospital name, etc.).

If these EHRs are uploaded directly to the cloud and shared for research purposes, the patient and hospital sensitive information will inevitably be exposed to the cloud and researchers. It is also needed to ensure EHR integrity due to human error in the cloud or software / hardware failures.

Therefore, it is important to perform a remote dataintegrity check, provided that the sensitive information in the shared data is protected.

A potential way to solve this problem is to encrypt the entire shared file before sending it to the cloud, then generate the signatures used to verify the integrity of the encrypted file, and finally. The corresponding signatures of to this encrypted file and its cloud. This method allows only the data owner to decrypt this file, thus hiding sensitive information. However, the entire shared file will not be available to other users.

For example, encrypting the EHR of a patient with an infectious disease can protect the privacy of the patient and the hospital, but these encrypted EHRs cannot be effectively used by researchers. Distributing the decryption key to researchers seems to be a possible solution to the above problem. However, this method cannot be applied in real-world scenarios for the following reasons: First, the distribution of decryption keys requires a secure channel, which can be difficult to achieve. Moreover, when users upload their EHRs to the cloud, it seems very difficult to know which researchers will use their EHRs soon.

As a result, encrypting the entire shared file to hide sensitive information is impractical. As a result, figuring out how to do data exchange with sensitive information hidden in remote data integrity audits is crucial. Unfortunately, past studies have not investigated this issue.

II. LITERATURE REVIEW

A. Attribute-based Encryption

In a potential cryptographic primitive that increases the flexibility of access control schemes greatly. The computational complexities of ABE key-issuing and decryption are becoming too high due to the great expressiveness of ABE Policies. Despite the fact that existing Outsourced ABE systems can offload some expensive computational activities to a third party, the verifiability of the third-party results has yet to be addressed. To address the problem, we present a novel Safe Outsourced ABE system that allows for secure outsourced key issuance and decryption.

All access policy and attribute related operations in the key-issuing process or decryption are offloaded to a Key Generation Service Provider (KGSP) and a Decryption Service Provider (DSP), respectively, leaving only a constant number of simple operations for the attribute authority and eligible users to perform locally with our new method. Furthermore, it proposes an outsourced ABE structure for the first time, which delivers and directs the user back to the login action. To access the webpage, a registered user must first login. Validations are applied to all textboxes to ensure that the webpage functions properly.

Each of the recommended solutions has been proven secure and practical, just as the information in each textbox is required.

B. Patient Controlled Encryption

This examines the challenge of maintaining patient privacy with an electronic health record system. We argue that the security of such systems needs to be enforced through both encryption and access control. In addition, it advocates an approach that allows patients to generate and store encryption keys, protecting their privacy even if the host data center is compromised. The standard argument for such an approach is that encryption compromises the functionality of the system.

However, it does show that patients can share partial access with others and build an efficient system to search their records. It formalizes the requirements of a patient-driven cryptographic scheme and

performs multiple instantiations based on existing cryptographic primitives and protocols, each with a different set of properties.

C. Cross-Domain data sharing

Indistributed electronic health record system, Collaboration between organizations or domains takes place on an electronic health record (EHR) system from time to time for the need and quality of patient care. Careful design of the delegation mechanism is required as a component of cross-domain collaboration. Collaboration inevitably involves exchanging and sharing relevant patient data, which is considered to be very private and sensitive. The delegation mechanism grants permission to cooperating partners and limits access.

Patients will not embrace the EHR system until correct use and disclosure of their health data is ensured, which is difficult to achieve without cross-domain authentication and fine-grained access control. Furthermore, the granted rights should be revocable at any time during the collaboration. In this research, we present a secure EHR system based on cryptographic constructions that allows for secure sharing of critical patient data while preserving patient data privacy during collaboration. Our EHR system also includes advanced mechanisms for fine-grained access control and on-demand revocation, which are upgrades to the delegation mechanism's basic access control and the basic revocation mechanism's basic revocation, respectively. The recommended EHR system is shown to realize specific goals within the cross-domain delegation situation of interest.

D. Privacy-Preserving multi-Keyword Ranked

Data owners are driven to outsource their complicated data management systems from local locations to the commercial public cloud due to the development of cloud computing, which provides greater flexibility and cost savings. However, to ensure data protection, sensitive data must be encrypted first. It has been outsourced and the use of traditional plaintex keyword search-based data has been discontinued.

Therefore, it is of utmost importance to enable encrypted search services for cloud data. Considering the largenumber of data consumers and documents in the cloud, it's vital to allow various keywords in the search request and return documents in the order associated with those keywords. Related work on searchable encryption focuses on a single keyword or Boolean search.

Keyword searches are rare, and the results are rarely sorted. This article is that the initial to outline and solve the tough drawback of privacy-preserving multi-keyword ranked search over encrypted information (MRSE). For such a safe cloud data consumption system, we create a set of stringent privacy standards. To capture the relevance of data documents to the search query, we use the efficient similarity measure of "coordinate matching," i.e., as many matches as possible, among several multi-keyword semantics.

We also use "Internal Product Similarity" to quantitatively assess such a measure of similarity. We provide a fundamental notion for the MRSE based on safe inner product computation, followed by two greatly improved MRSE schemes to meet various strict privacy criteria in two separate threat scenarios.We enhance these two techniques to allow more search semantics in order to improve the data search service's search experience. A detailed analysis is provided to study the privacy and efficiency assurance of the proposed system. Experiments on a real-world data set reveal that the proposed approaches do actually reduce compute and transmission overhead.

III. PROPOSED WORK

The proposed system is based on the secure hash algorithm and Advanced Encryption standard which is very popular in transmitting of messages in a secured way the data flow diagram is shown in the fig 1.1. The process is divided into four modules which employs various methods.

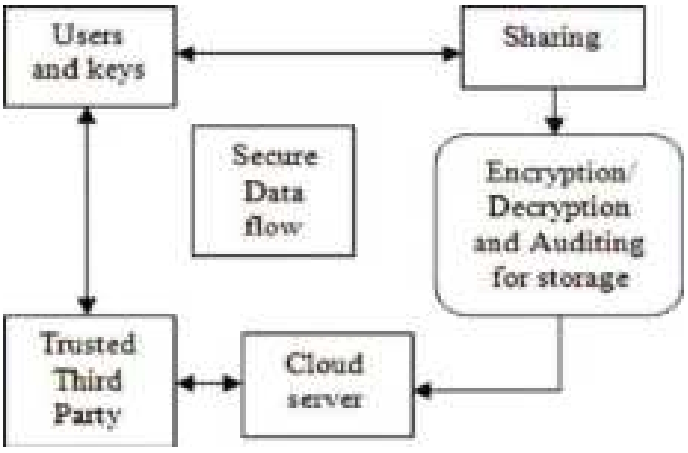


Fig 1.1 Data flow diagram

A. *Login Module*

When a visitor visits a website for the first time, this is the first activity that appears. In order to get onto the website, the user must supply a valid phone number and a password, which the user creates upon registering. If the user's information matches the data in the database table, the user is successful in logging into the website; otherwise, a notification of login failure is displayed, and the user must re-enter accurate information. Links to registration activities for new user registration are also provided.

B. *Registration Module*

New users who wish to access the website must first register before logging in. Click the Register button for your login activity to open the registration activity. New users register by entering their name, password, and contact number. The user must re-enter and confirm the password in the Confirm Password text box. When the user fills in all the text fields and clicks the Register button, the data is transferred to the database and the user is redirected to login activity again. Registered users must be logged into access the website. Validation is applied to all text fields to ensure that your website works properly. Like the information in each text field, the name, contact, password, or password confirmation text fields do not need to be blank during registration. If such text fields are empty, the app will print a message containing the information that should be included in each text field.

In addition, the data in the Password and Confirm Password fields must match for successful registration. Another validation is that the contact number must be a valid 10-digit number. If any of these validations are broken, registration will fail, and the user will have to re-register. When one of the fields on the page is left blank, a message will appear. If all the above information is correct, the user will be led to the login activity, which will allow them to log into the webpage.

C. *Creation of Storage*

After the data is uploaded to the cloud, the data owner has no control over the data. In this module, the original data is encoded into two different values. As shown in Figure 1.2, the data in each slice can be encrypted using various encryption algorithms and encryption keys before being stored in the cloud.

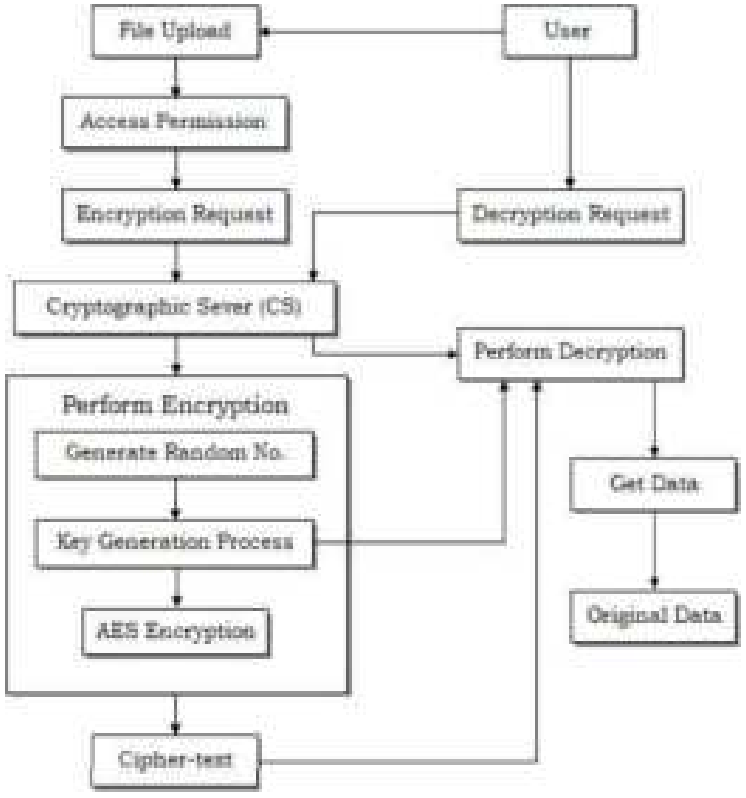


FIG 1.2 WORKFLOW OF A FILE UPLOADING IN CLOUD

D. Data Protection

This module uses techniques to store data properly and securely to avoid intruders and data attacks while reducing the cost and time required to store encrypted data in cloud storage increase.

IV. SYSTEM ARCHITECTURE

In the figure 1.3, the data owner sends the data which will be encrypted as cipher text from plain text. Further the data is divided into two slices as slice 1 and slice 2. Then the slice 2 is encrypted again and adds the slice 1. Finally, these values are stored in the cloud storage. In the receiver side it receives the data from the cloud as a cipher text and it decrypts the slice 1 and encrypted slice 2 data. Then the value of slice 2 is again decrypted to find the divided slices for the receiver. Then these values are added to find the encrypted text i.e., the original data.

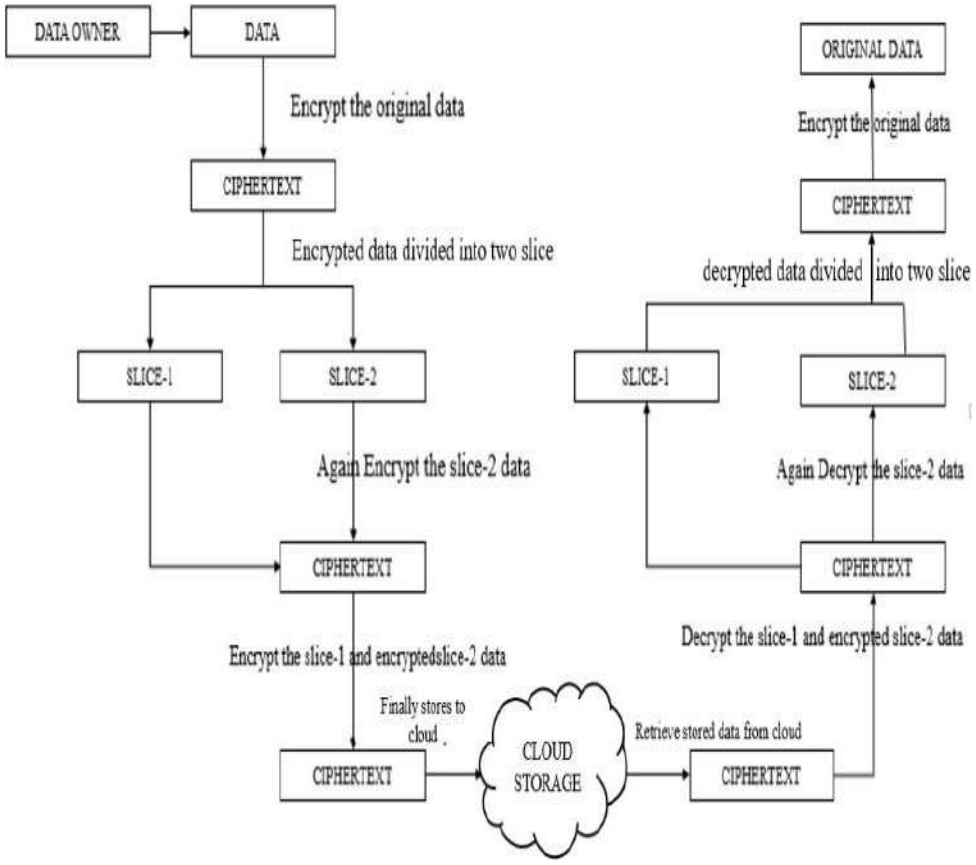


FIG 1.3 SYSTEM ARCHITECTURE

V. METHODOLOGY

A. *SHA Algorithm*

The Secure Hash Algorithms are a collection of cryptographic hash functions released by the National Institute of Standards and Technology (NIST) as a Federal Information Processing Standard (FIPS) in the United States. They include:

- SHA-0: A retronym for the original 160-bit hash function, which was published in 1993 under the name "SHA." It was superseded with the significantly altered version SHA-1 shortly after publication due to an unknown "major weakness."
- SHA-1: A 160-bit hash function that resembles the MD5 algorithm from the past. The National Security Agency (NSA) created this as part of the Digital Signature Algorithm. After SHA-1's cryptographic flaws were found, the standard was no longer authorized for most cryptographic applications after 2010.
- SHA-2:SHA-256 and SHA-512 are two hash methods that are similar but have differing block sizes. The word size of SHA-256 and SHA-512 is different; SHA-256 utilizes 32-bit bits and SHA-512 uses 64-bit words. Each standard also has shortened versions called as SHA-224, SHA-384, SHA-512/224, and SHA-512/256. The National Security Agency (NSA) was also responsible for these.
- SHA-3: After a public competition among non-NSA designers in 2012, a hash function formerly known as Keccak was chosen. It uses the same hash lengths as SHA-2 and has a different internal structure than the rest of the SHA family.

NIST has updated the Draft FIPS Publication 202, SHA3 Standards separately from the Secure Hash Standard (SHS).

B. *AES Algorithm*

The Advanced Encryption Standard (AES) could be a specification for the secret writing of electronic information established by the NIST in 2001. AES is wide used these days as it could be a abundant

stronger than DES and triple DES despite being tougher to implement. AES could be a block cipher. The key size are often 128/192/256 bits. Encrypts information in blocks of 128 bits each, meaning it takes 128 bits as input and outputs a 128-bit encrypted ciphertext.

AES depends on substitution-permutation network principle which implies it's performed employing a series of linked operations that involves exchange and shuffling input data.

AES performs operations on bytes of data instead of in bits. Since the block size is 128 bits, encryption Processes 128-bit (or 16 bytes) input data at the same time.

The number of rounds depends on the key length as follows 128 bit key – 10 rounds, 192 bit key – 12 rounds and 256 bit key – 14 rounds.Refer Fig1.4.

VI. CONCLUSION AND FUTURE SCOPE

A. *Conclusion*

These outcomes result in data confidentiality, data integrity, authentication, and authorization being achieved, as well as the elimination of both active and passive assaults in the cloud network environment. Develop a secure cloud framework for gaining access to trustworthy compute and storage services throughout the public cloud deployment paradigm at all levels.

B. *Future Scope*

To deliver trustworthy computing and storage services, it achieves high levels of security. Data integrity, confidentiality, authentication, and authorization are all provided. Both internal and external security dangers are eliminated. In a cloud network context, it protects against both active and passive assaults. Different levels of security are achieved in the cloud framework.

ACKNOWLEDGMENT

We thank CMR Technical Campus for supporting this paper titled with "Generating Cloud Monitors From Models To Secure Clouds ", which provided good facilities and support to accomplish our work. Sincerely thank to our Chairman, Director, Deans, Guide and Faculty Members for giving valuable suggestions and guidance in every aspect of our work.

REFERENCES

[1] C. Chu, S. Chow, W. Tzeng, J. Zhou, R. Deng, "Key-aggregate cryptosystem for scalable data sharing in cloud storage," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 2, pp.468-477, Feb. 2014.

[2] Y. Tong, J. Sun, S. Chow, P. Li, "Cloud-assisted mobile-access of health data with privacy and auditability", IEEE Journal of Biomedical and Health Informatics, vol. 18, no. 2, Mar. 2014.

[3] Z. Pervez, A. Khattak, S. Lee, Y. Lee, "SAPDS: Self-healing attribute-based privacy aware data sharing in cloud", The Journal of Supercomputing, vol. 62, no. 1, pp. 431´lC460, Oct. 2012.

[4] C. Fan, V. Huang, H. Rung, "Arbitrary-state attribute-based encryption with dynamic membership", IEEE Transactions on Computers, vol. 63, no. 8, pp. 1951-1961, Apr. 2013.

[5] X. Chen, J. Li, J. Weng, J. Ma, W. Lou, "Verifiable computation over large database with incremental updates" IEEE Transactions on Computers, vol. 65, no. 10: 3184-3195, Oct.2016. [11] C. Gao, Q. Cheng, X. Li, S. Xi

[6] N. Cao, C. Wang, M. Li, K. Ren, W. Lou, "Privacy-preserving multi key word ranked search over encrypted cloud data", IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 1, pp. 222-233, Nov. 2013.

[7] S. Seo, M. Nabeel, X. Ding, E. Bertino, "An efficient certificate less encryption for secure data sharing in public clouds", IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 9, pp. 2107-2119, Sept. 2014.

[8] L.A. Dunning, R. Kresman, "Privacy preserving data sharing with anonymous ID assignment", IEEE Transactions on Information Forensics and Security, vol. 8, no. 2, pp.402-413, Feb. 2013.

[9] X. Chen, X. Huang, J. Li, J. Ma, D. Wong, W. Lou, "New algorithms for secure outsourcing of large-scale systems of linear equations", IEEE Transactions on Information and Forensics Security, vol. 10, no. 1, pp. 69- 78, Jan. 2015.

[10] C. Fan, V. Huang, H. Rung, "Arbitrary-state attribute-based encryption with dynamic membership", IEEE Transactions on Computers, vol. 63, no. 8, pp. 1951-1961, Apr. 2013.