Navigating The Complex World Of Auto Insurance: A Vehicle Cost Analysis For Better Decision Making

A Project Report

Submitted by

Team ID : NM2023TMID16447

TEAM MEMBERS

A. MANI

L. NITHISH

R.S KRITHIK KRISHNA

TABLE OF CONTENT

1.1 INTRODUCTION

1 PROJECT OVERVIEW

The project "Auto Insurance of Vehicle" aims to develop a comprehensive auto insurance system that manages and automates various aspects of vehicle insurance. The system will provide a user-friendly interface for insurance companies, agents, and policyholders to streamline the insurance process and enhance overall efficiency.

objectives

- Improve Customer Experience Enhance the overall customer experience by providing a user-friendly interface, simplified processes, and easy access to policy information.
- This includes streamlining policy purchasing, quote generation, claims submission, and policy management.
- Increase Efficiency and Automation: Implement automation in various insurance processes to reduce manual efforts and increase operational efficiency.
- This involves automating policy generation, premium calculation, claims processing, and payment handling, thereby reducing processing time and improving accuracy.
- Enable insurance companies to efficiently manage policies, track policy changes, and ensure compliance with regulatory requirements.
- Strengthen Claims Processing: Improve the claims handling process by implementing a streamlined and transparent system.
- Integration and Scalability: Design the auto insurance system to seamlessly integrate with existing systems within insurance companies, such as CRM software and underwriting systems.

By achieving these objectives, the "Auto Insurance of Vehicle" project aims to enhance customer satisfaction, improve operational efficiency, reduce costs, minimize risks, and establish a competitive edge in the auto insurance market.

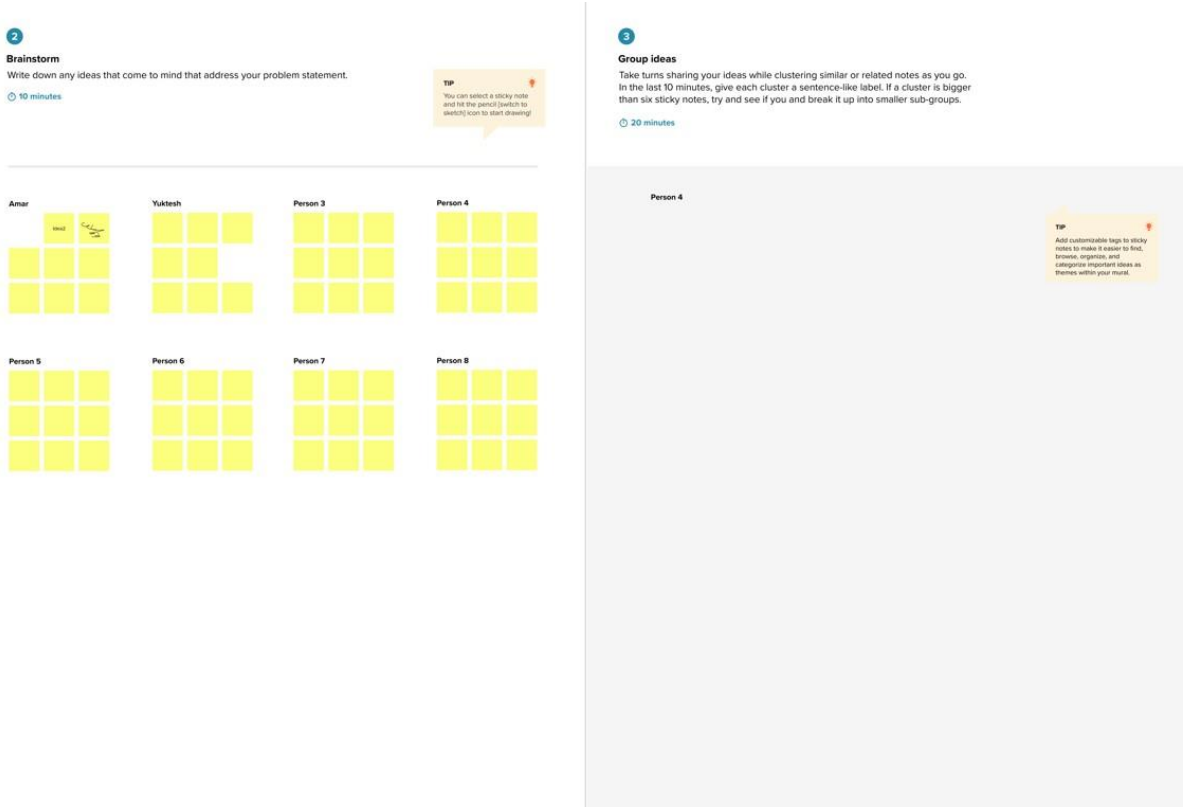
1.2 Purpose

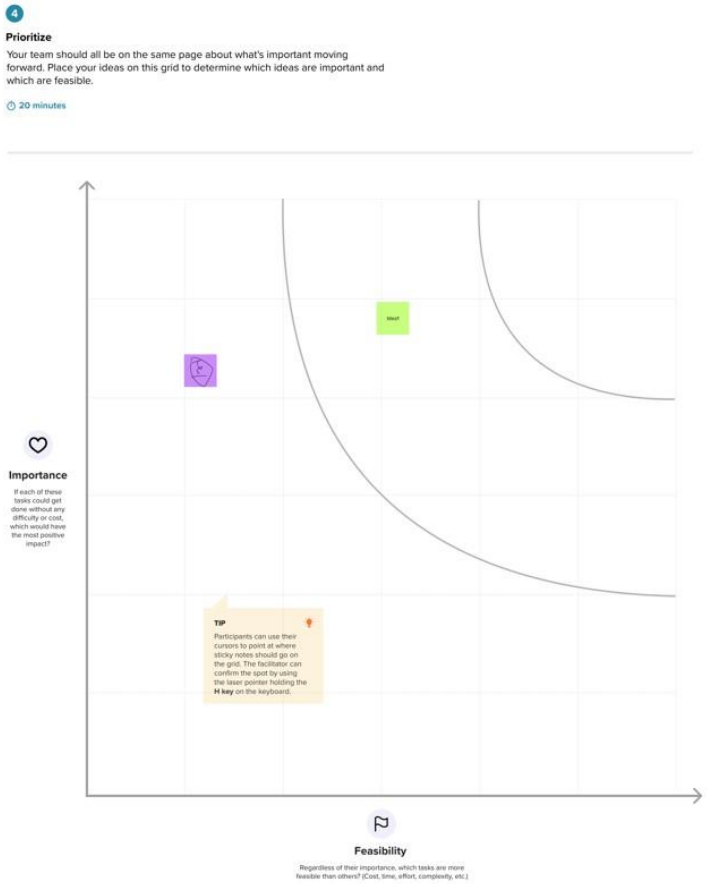
- Liability Coverage: This is the most basic and mandatory type of auto insurance. It covers the costs associated with injuries or property damage caused to others in an accident where you are at fault**.**
- Collision Coverage: This type of coverage helps pay for repairs or replacement of your own vehicle if it is damaged in a collision with another vehicle or object, regardless of fault.
- Comprehensive Coverage: Comprehensive coverage provides protection against non-collision incidents such as theft, vandalism, natural disasters, falling objects, fire, or hitting an animal.
- Personal Injury Protection (PIP): PIP coverage is designed to cover medical expenses, lost wages, and other related costs for you and your passengers, regardless of who is at fault in an accident.

2.1 Problem Statement Definition

High Premium Costs: Many drivers find auto insurance premiums to be expensive, making it difficult for them to afford adequate coverage. This issue can result in uninsured or underinsured motorists on the road, leading to potential financial and legal risks.Complex and Time-consuming Claims Process: The process of filing and settling insurance claims can be lengthy and cumbersome. The auto insurance industry is susceptible to fraudulent activities, such as staged accidents, false claims, and identity theft. This approach may result in inaccurate risk assessments and unfair premiums for certain drivers.

2.3 Ideation & Brainstorming





2.4 Proposed Solution:

1. **Usage-based insurance (UBI):** Implementing UBI programs that rely on telematics devices or smartphone apps to track driving behavior. By monitoring factors such as mileage, speed, and braking patterns, insurance premiums can be adjusted based on individual driving habits. Safe drivers would be rewarded with lower rates, while risky behavior could result in higher premiums. UBI encourages responsible driving and provides fairer pricing.
2. **Advanced driver assistance systems (ADAS)** integration: Collaborate with automobile manufacturers to promote the use of ADAS technologies such as lane departure warning, automatic emergency braking, and adaptive cruise control. Insurance companies could offer discounted rates for vehicles equipped with these safety features, as they help reduce the risk of accidents and lower claim frequencies.
3. **Improved fraud detection:** Develop sophisticated algorithms and AI models to detect insurance fraud more effectively. By analyzing data patterns and identifying suspicious claims, insurers can minimize losses due to fraudulent activities, ultimately reducing the overall cost of insurance and making premiums more affordable for honest policyholders.
4. **Enhanced customer experience**: Simplify the insurance process and make it more customer-friendly. This could involve streamlining the claims process, offering 24/7 customer support, and providing online tools for policy management. By focusing on customer satisfaction, insurers can build trust and loyalty, which could lead to more competitive rates.
5. **Collaboration with municipalities and road safety initiatives:** Work closely with local governments and organizations to promote road safety campaigns, driver education, and infrastructure improvements. By actively participating in efforts to reduce accidents and create safer driving environments,

insurance companies can contribute to lower claim rates and, in turn, lower insurance costs.

6. **Personalized risk assessment:** Utilize big data and predictive analytics to assess risk factors more accurately. By considering individual driving history, demographic data, and other relevant variables, insurers can tailor policies to reflect a person's specific risk profile. This personalized approach ensures fairer pricing based on individual circumstances rather than relying solely on general statistics.
7. **Partnerships with sharing economy platforms:** Develop specialized insurance products for ride-sharing services, car-sharing platforms, and other emerging sharing economy models. These tailored insurance options could provide coverage during commercial activities, helping to fill gaps in traditional personal auto insurance policies.

3. REQUIREMENT ANALYSIS

3.1 Functional Requirements :

Policy Management: The system should allow users to create, update, and manage auto insurance policies. This includes capturing policy details such as the insured vehicle's information, coverage limits, deductibles, and premium calculations.

Quote Generation: The system should provide a mechanism for generating insurance quotes based on user inputs. It should consider factors such as the driver's age, driving history, vehicle type, and other relevant risk factors to calculate accurate quotes.

Policy Underwriting: The system should include underwriting rules and logic to evaluate insurance applications and determine whether to approve or reject them. It should assess factors like the applicant's driving record, credit history, and other relevant data to make informed underwriting decisions.

Claims Management: The system should facilitate the submission, processing, and management of insurance claims. It should enable users to report accidents, upload supporting documents, track the progress of their claims, and facilitate communication with claims adjusters.

Premium Calculation: The system should be able to calculate insurance premiums based on various factors such as the insured vehicle's value, location, usage, and the driver's risk profile. It should take into account discounts, surcharges, and other pricing adjustments to provide accurate premium calculations.

3.2 Non-Functional Requirements :

Performance: The system should be responsive and provide quick response times for tasks such as generating quotes, processing policy changes, and handling claims. It should be able to handle a high volume of concurrent users without significant performance degradation.

Scalability: The system should be able to scale effectively to accommodate increasing user demands and growing data volumes. It should have the ability to handle a larger customer base, policy portfolio, and transactional load without sacrificing performance or reliability.

Reliability: The system should be highly reliable and available to ensure uninterrupted service for policyholders and other users. It should have mechanisms in place to handle system failures, recover data in case of incidents, and minimize downtime or service disruptions.

Security: The system should have robust security measures to protect sensitive data, such as personal information, financial details, and policy-related data. It should implement authentication, authorization, and encryption mechanisms to prevent unauthorized access, data breaches, and fraud.

Data Integrity and Privacy: The system should ensure the integrity and accuracy of data throughout its lifecycle. It should have controls to prevent data corruption, loss, or unauthorized modification. Additionally, it should comply with privacy regulations and protect customer data from unauthorized disclosure.

Usability: The system should be intuitive, user-friendly, and easy to navigate for both policyholders and insurance agents. It should have clear and well-designed user interfaces that allow users to perform tasks efficiently without unnecessary complexity or confusion.

Accessibility: The system should support accessibility standards to accommodate users with disabilities. It should provide features such as screen readers, keyboard navigation, and alternative text for visual elements to ensure equal access to information and functionality.

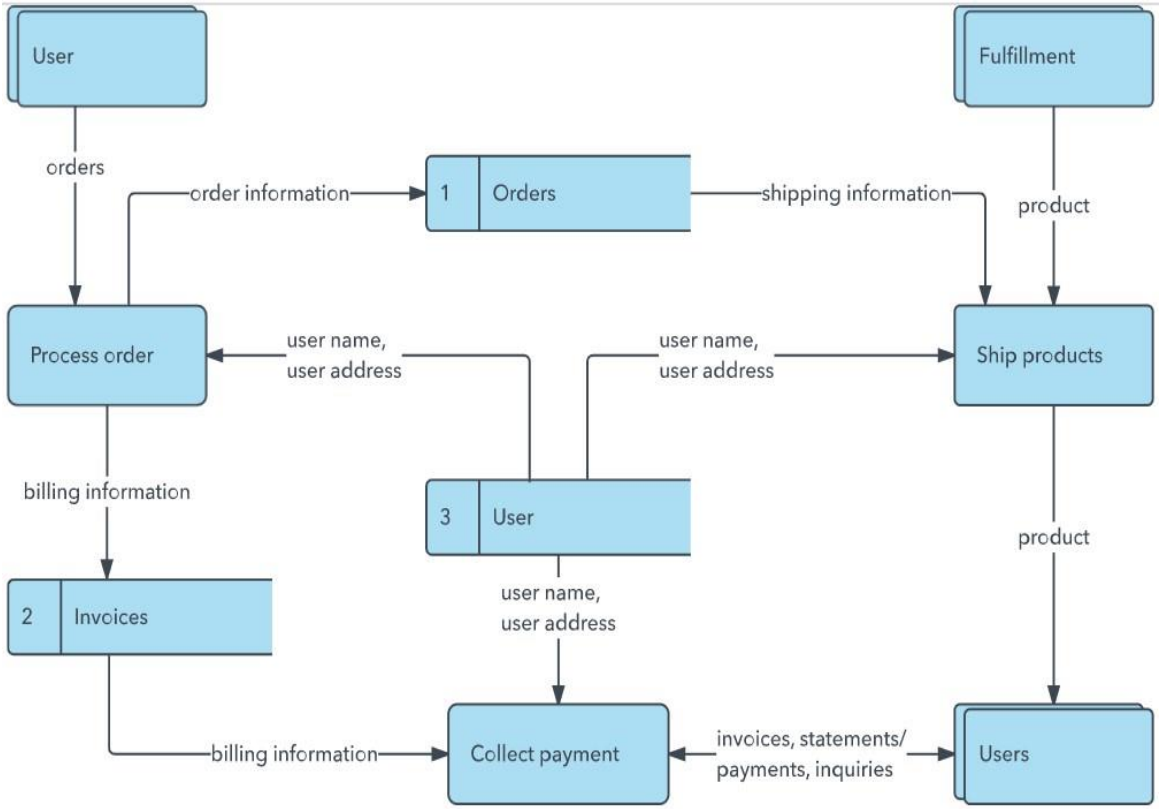
Interoperability: The system should be capable of integrating with external systems, such as third-party data providers, government databases, or claims management systems. It should support standard data formats and communication protocols to facilitate seamless data exchange and interoperability.

4.PROJECT DESIGN

4.1 Data Flow Diagrams

Data Flow Diagrams (DFDs) are a visual representation of how data moves within a system. They help to illustrate the flow of information, processes, and data stores. In the context of auto insurance, here's an example of a high-level DFD that represents the data flow within an auto insurance system:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

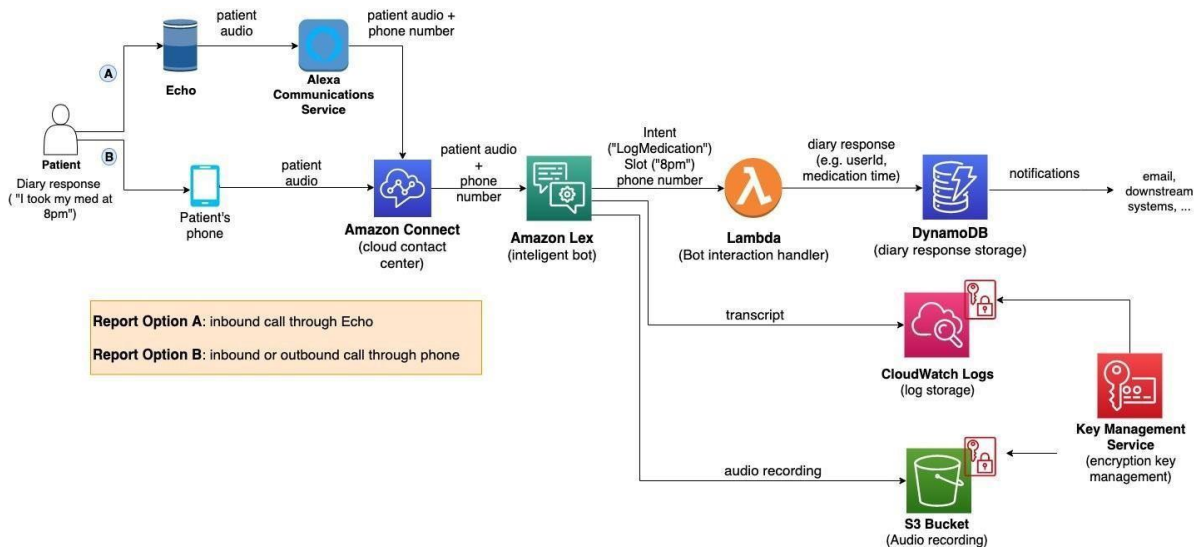


4.2 Solution and technical architecture :

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

Document Management: Create a centralized repository to manage claim-related documents, such as accident reports, repair estimates, and medical reports.



Policy Management:

User Interface: Develop a web or mobile application for customers to view policy details, make changes, and renew policies.

Policy Database: Maintain a database to store policy information, including customer details, coverage details, policy terms, and payment history.

Rating Engine: Implement a rating engine that calculates premiums based on various factors such as the customer's driving record, vehicle type, age, and location.

Integration with Data Providers: Integrate with external data providers to gather information on driving records, vehicle specifications, and historical claims data.

Claims Processing:Claim Submission: Provide a digital platform for policyholders to submit claims, including details of the incident, supporting documents, and photos.

Claims Workflow: Implement a workflow system to automate the claims process, including claim validation, assessment, and approval/rejection.

4.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Team Member
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Shivam
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Shivani
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Shivam
		USN-4	As a user, I can register for the application through Gmail		Medium	Shivam
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sandeep
	Dashboard					
Customer (Web user)						
Customer Care Executive						
Administrator						

5 . CODING & SOLUTIONING

5.1 Feature 1

Personalized Pricing: Telematics-based pricing allows auto insurance companies to offer personalized insurance rates to policyholders. Instead of relying solely on general demographics or historical data, the premiums are calculated based on the individual's driving behavior. Safe drivers with good habits can enjoy lower rates, while high-risk drivers may need to pay higher premiums.

Incentivizes Safe Driving: By incorporating telematics technology, auto insurance providers can encourage safe driving habits among policyholders. Knowing that their driving behavior is being monitored, drivers are more likely to adopt safer practices such as obeying speed limits, maintaining proper distances, and avoiding sudden accelerations or harsh braking. This can lead to a reduction in accidents and insurance claims, benefiting both the policyholder and the insurance company.

Risk Reduction: Telematics data provides insurers with valuable insights into individual driving habits and patterns. This information helps them identify high-risk drivers and develop risk mitigation strategies. By offering personalized feedback and suggestions for improvement, insurers can actively contribute to reducing accidents and promoting safer driving practices.

Usage-Based Insurance: Telematics-based pricing enables the implementation of usage-based insurance models. Instead of traditional fixed premiums, policyholders pay based on the actual usage of their vehicles. This can be particularly beneficial for occasional or low-mileage drivers who may be currently paying higher premiums than their actual risk warrants.

Transparency and Control: Telematics-based pricing offers transparency to policyholders by providing them with detailed information about their driving behavior and how it affects their insurance rates.

5.2 Feature 2

Prompt Emergency Assistance: Accurate and timely accident detection enables auto insurance companies to provide prompt emergency assistance to policyholders involved in accidents. This can be crucial in situations where immediate medical attention or rescue services are required. By leveraging technology to automatically detect accidents, insurers can reduce response times and potentially save lives.

Improved Claims Process: Accident detection technology can streamline the claims process for policyholders. By automatically capturing and transmitting accident data, such as location, time, and impact severity, insurers can expedite claim settlement and reduce the need for extensive manual documentation and investigations. This can result in faster claims resolution and enhanced customer satisfaction.

Enhanced Safety and Security: The presence of accident detection technology can contribute to increased safety and security on the road. Policyholders may feel more secure knowing that their insurance provider has mechanisms in place to promptly respond in the event of an accident. Additionally, accident detection systems can act as a deterrent, encouraging responsible driving behavior and reducing the likelihood of fraudulent claims.

Remote Monitoring and Support: Accident detection systems can enable remote monitoring of policyholders' vehicles, providing insights into their driving behavior and vehicle health. Insurance companies can use this data to offer proactive support and guidance, such as providing maintenance reminders, identifying potential vehicle issues, and promoting safe driving habits. This helps policyholders maintain their vehicles in optimal condition and reduces the risk of accidents due to mechanical failures.

Data-driven Insights: The data collected from accident detection systems can provide valuable insights to insurance companies. By analyzing accident patterns, locations, and contributing factors, insurers can identify high-risk areas and develop strategies to mitigate

accidents in those areas. This data-driven approach can contribute to overall road safety improvements and help insurers optimize their risk assessment and pricing models.

Considerations: Privacy and Data Security: Accident detection systems involve the collection and transmission of sensitive data, including location information and vehicle dynamics. Insurers must ensure strict adherence to data protection regulations and implement robust security measures to safeguard policyholders' privacy and prevent unauthorized access to their personal information.

Accuracy and Reliability: The accuracy and reliability of accident detection systems are crucial for their effectiveness. False positives or false negatives can impact the response time and efficiency of emergency services. Insurance companies should invest in reliable sensor technology and regularly test and validate their accident detection algorithms to ensure accurate and dependable performance.

Collaboration with Emergency Services: Seamless collaboration with emergency services is essential for the success of this feature. Insurance companies must establish partnerships or integration mechanisms with local emergency services to ensure effective communication and coordination in emergency situations. This may involve establishing protocols, sharing data, and conducting joint training sessions to optimize response times and outcomes.

User Adoption and Awareness: Policyholders need to be aware of the accident detection feature and understand its benefits. Insurance companies should educate their customers about the availability of this feature, its functionality, and the steps to enable it. Clear communication and incentivization can encourage policyholders to opt-in and actively use the feature.

Integration with Existing Systems: Insurance companies may need to integrate the accident detection system with their existing technology infrastructure, such as their claims processing and customer support systems.

6. RESULTS

6.1 Performance Metrics

Loss Ratio: The loss ratio is a key performance metric that represents the ratio of incurred losses (including claim payments and claim-related expenses) to earned premiums. It indicates the percentage of premium revenue that an insurer pays out in claims. A lower loss ratio generally signifies better underwriting and claims management.

Combined Ratio: The combined ratio is a broader metric that includes not only the loss ratio but also other expenses such as underwriting expenses and commissions. It is calculated by adding the loss ratio and the expense ratio. A combined ratio below 100% indicates an underwriting profit, while a ratio above 100% indicates an underwriting loss.

Expense Ratio: The expense ratio measures the percentage of underwriting expenses (such as administrative costs, commissions, and marketing expenses) to earned premiums. Lower expense ratios indicate greater efficiency in managing operational costs.

Loss Adjustment Expense (LAE) Ratio: The LAE ratio measures the ratio of claim-related expenses (such as legal fees, investigation costs, and adjuster fees) to earned premiums. It provides insights into an insurer's efficiency in managing claim costs.

Premium Growth Rate: Premium growth rate is the percentage change in written premiums over a specified period. Positive premium growth indicates an expanding customer base or successful marketing efforts, while negative growth may suggest declining market share or profitability challenges.

Customer Retention Rate: The customer retention rate measures the percentage of policyholders who renew their policies with the same insurance company.

7 . ADVANTAGES & DISADVANTAGES

ADVANTAGES :

- Damage or loss to insured vehicle
- Personal accident cover
- Large network of garages
- Third party liabilities
- No claim bonus
- Network garages
- Assured for medical claims
- Assured of repairs and replacement

DISADVANTAGES

- Coverage failures
- Time taking Process
- hassle-free claim settlement:
- Poor Customer Support
- No Claim Bonus
- Coverage against accident

8.CONCLUSION :

In conclusion, auto insurance is a crucial component of responsible car ownership. It provides financial protection against unexpected events such as accidents, theft, or damage to vehicles. Auto insurance not only safeguards the policyholder's interests but also helps protect other drivers and pedestrians by ensuring that the necessary resources are available to cover potential liabilities.

By having auto insurance, individuals can enjoy peace of mind knowing that they are financially protected in case of an accident. Insurance policies typically cover medical expenses, property damage, and legal liabilities resulting from accidents. This coverage

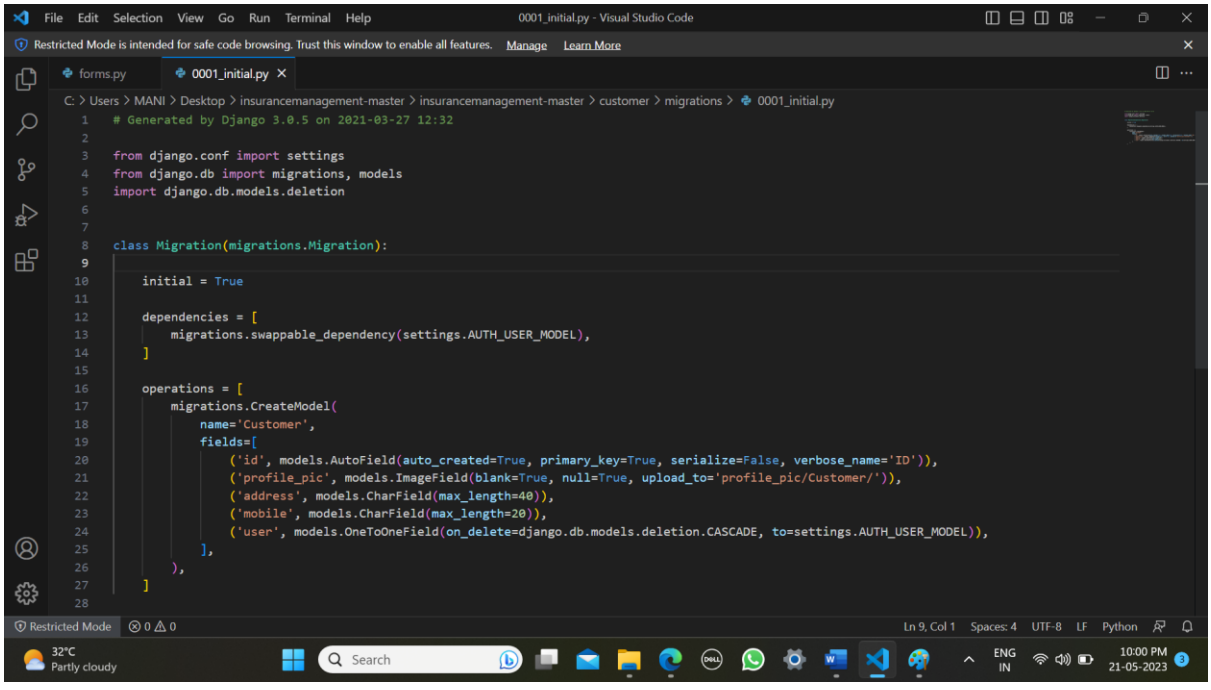
can be especially important in situations where substantial costs are involved, such as medical treatments, vehicle repairs, or legal proceedings.

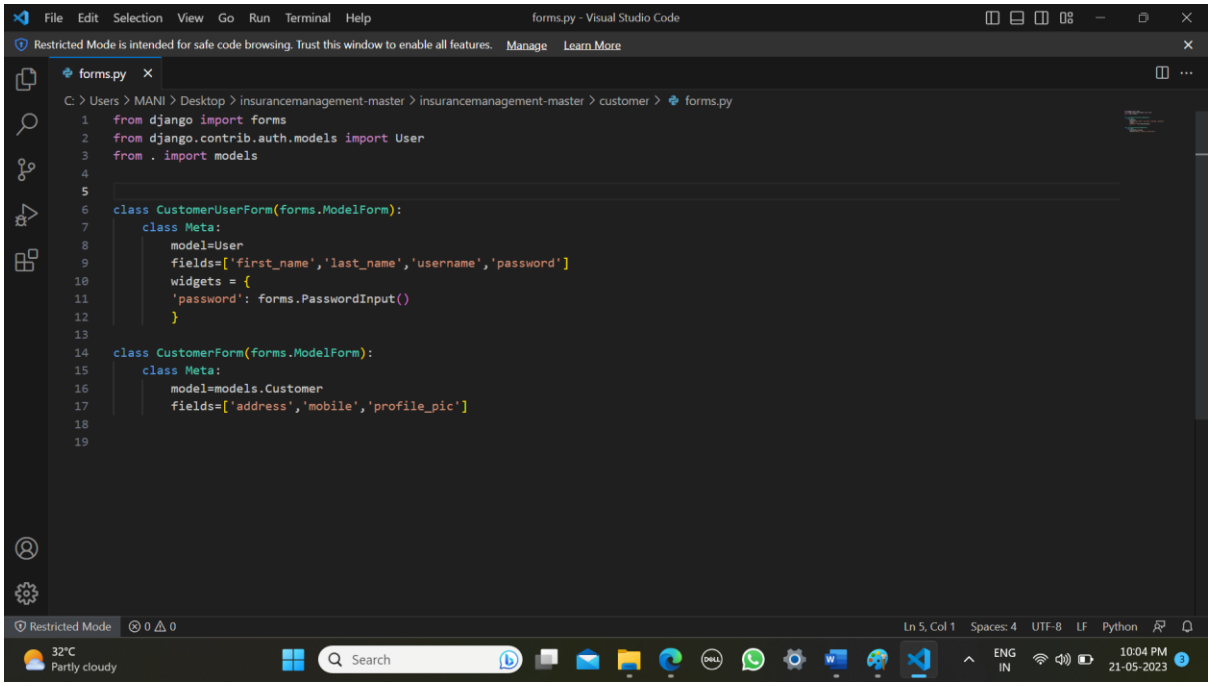
9.FUTURE SCOPE :

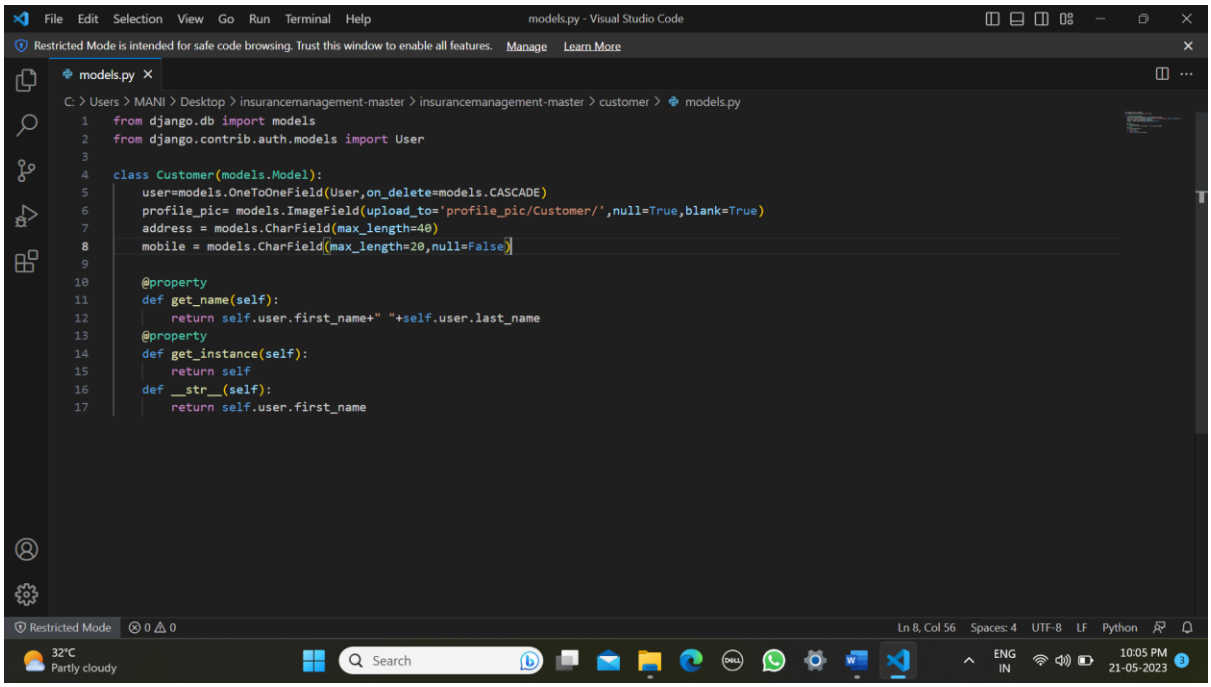
- Autonomous Vehicles
- Advanced Safety Features
- Big Data and Predictive Analytics
- Cybersecurity Insurance
- Integrated Mobility Solutions:
- Claims Automation and Customer Experience

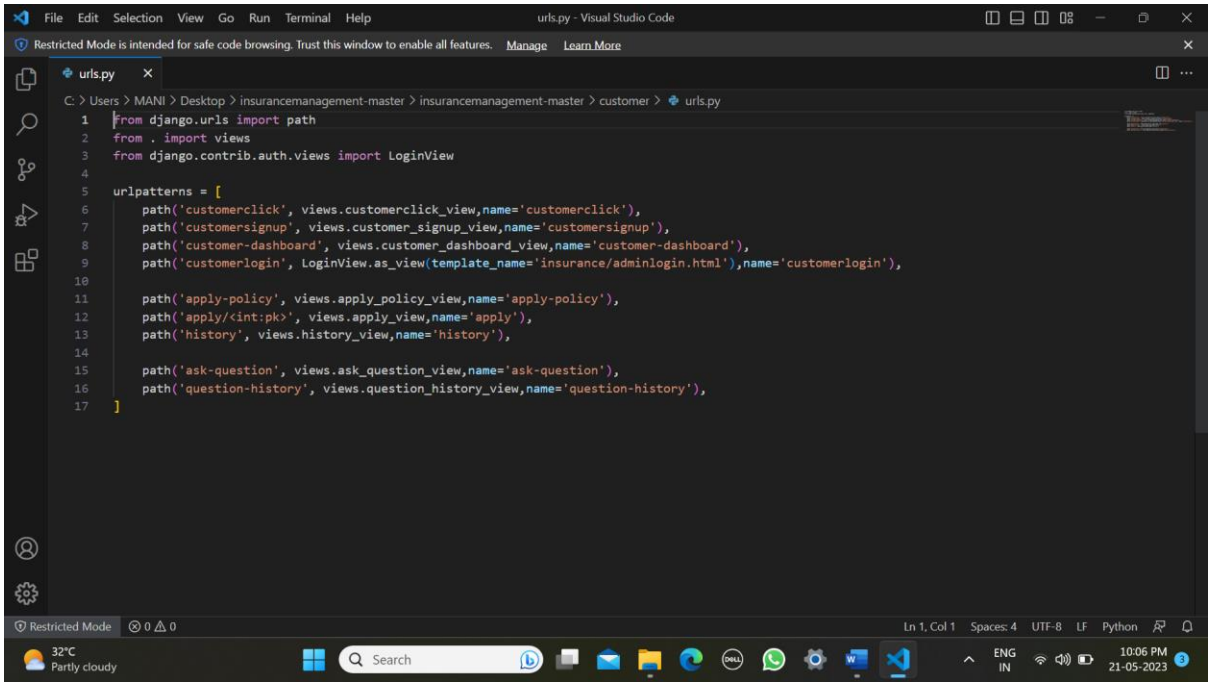
10.APPENDIX :

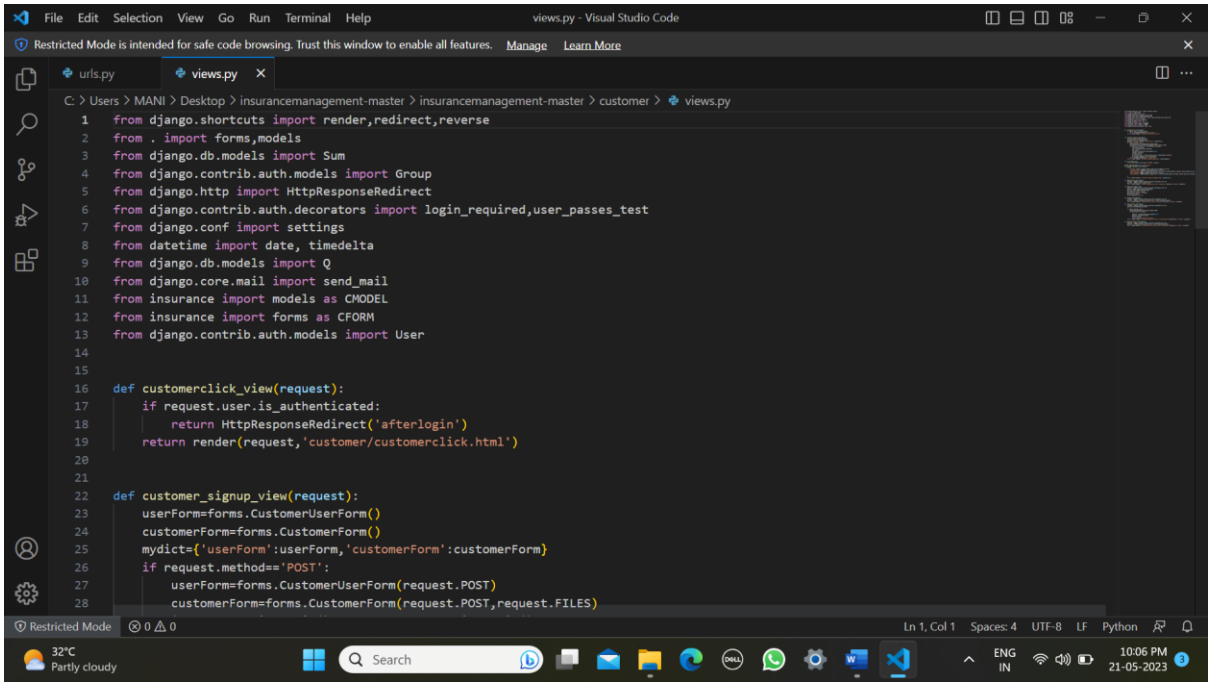
Source Code

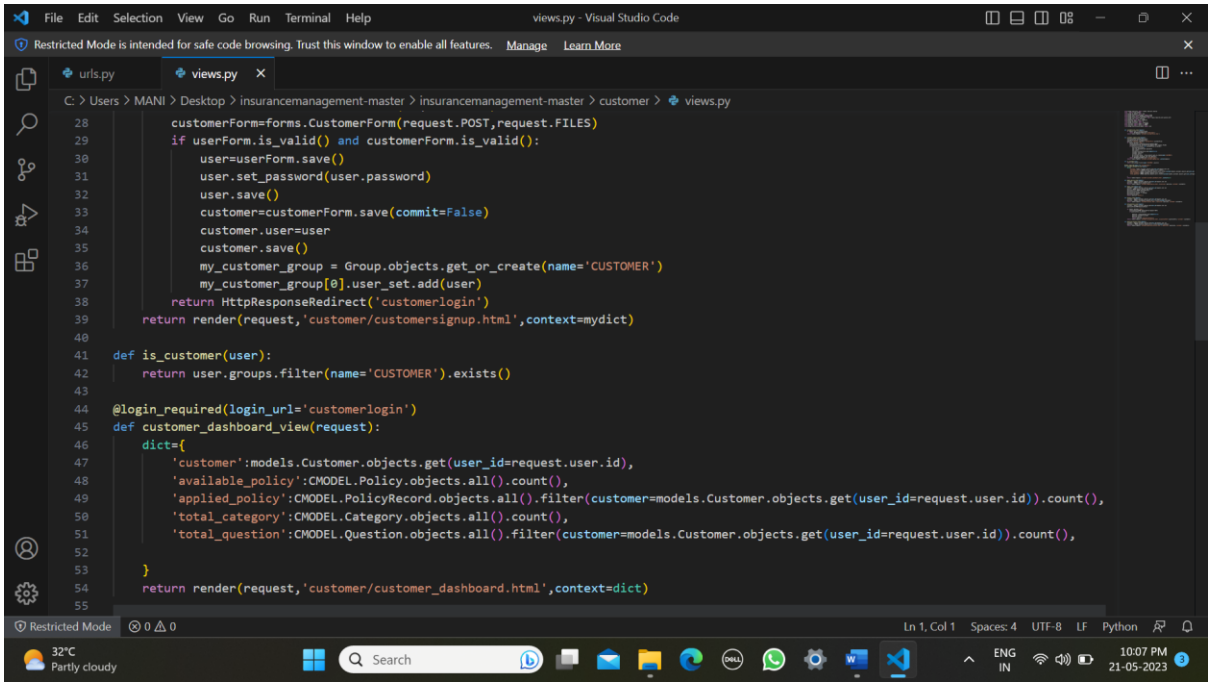



File Edit Selection View Go Run Terminal Help
forms.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
forms.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > customer > forms.py
from django import forms
from django.contrib.auth.models import User
from . import models
class CustomerUserForm(forms.ModelForm):
class Meta:
model=User
fields=['first_name','last_name','username','password']
widgets = {
'password': forms.PasswordInput()
}
class CustomerForm(forms.ModelForm):
class Meta:
model=models.Customer
fields=['address','mobile','profile_pic']
Restricted Mode
Ln 5, Col 1 Spaces: 4 UTF-8 LF Python
32°C
Partly cloudy
Search
ENG
IN
10:04 PM
21-05-2023

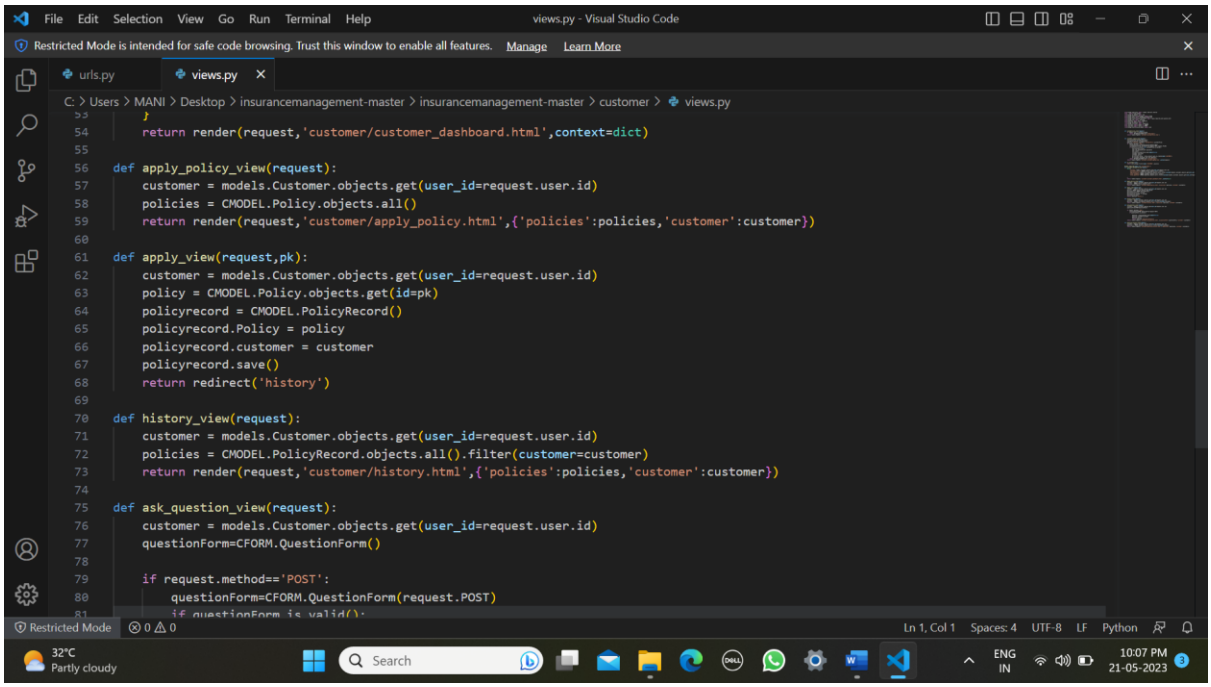

File Edit Selection View Go Run Terminal Help
models.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
models.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > customer > models.py
from django.db import models
from django.contrib.auth.models import User
class Customer(models.Model):
user=models.OneToOneField(User,on_delete=models.CASCADE)
profile_pic= models.ImageField(upload_to='profile_pic/Customer/',null=True,blank=True)
address = models.CharField(max_length=40)
mobile = models.CharField(max_length=20,null=False)
@property
def get_name(self):
return self.user.first_name+" "+self.user.last_name
@property
def get_instance(self):
return self
def __str__(self):
return self.user.first_name
Restricted Mode
Ln 8, Col 56 Spaces: 4 UTF-8 LF Python
32°C
Partly cloudy
Search
ENG
IN
10:05 PM
21-05-2023


File Edit Selection View Go Run Terminal Help
urls.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
urls.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > customer > urls.py
from django.urls import path
from . import views
from django.contrib.auth.views import LoginView
urlpatterns = [
path('customerclick', views.customerclick_view,name='customerclick'),
path('customersignup', views.customer_signup_view,name='customersignup'),
path('customer-dashboard', views.customer_dashboard_view,name='customer-dashboard'),
path('customerlogin', LoginView.as_view(template_name='insurance/adminlogin.html'),name='customerlogin'),
path('apply-policy', views.apply_policy_view,name='apply-policy'),
path('apply/<int:pk>', views.apply_view,name='apply'),
path('history', views.history_view,name='history'),
path('ask-question', views.ask_question_view,name='ask-question'),
path('question-history', views.question_history_view,name='question-history'),
]
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:06 PM 21-05-2023

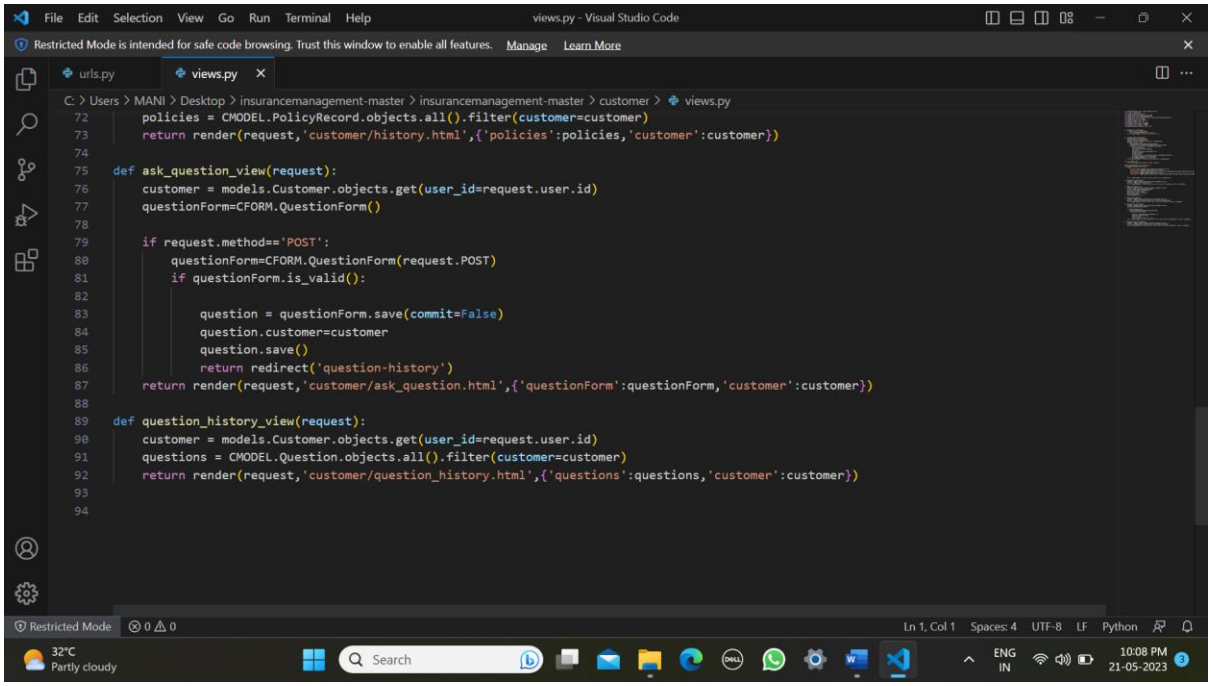

File Edit Selection View Go Run Terminal Help
views.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
urls.py
views.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > customer > views.py
from django.shortcuts import render,redirect,reverse
from . import forms,models
from django.db.models import Sum
from django.contrib.auth.models import Group
from django.http import HttpResponseRedirect
from django.contrib.auth.decorators import login_required,user_passes_test
from django.conf import settings
from datetime import date, timedelta
from django.db.models import Q
from django.core.mail import send_mail
from insurance import models as CMODEL
from insurance import forms as CFORM
from django.contrib.auth.models import User
def customerclick_view(request):
if request.user.is_authenticated:
return HttpResponseRedirect('afterlogin')
return render(request,'customer/customerclick.html')
def customer_signup_view(request):
userForm=forms.CustomerUserForm()
customerForm=forms.CustomerForm()
mydict={'userForm':userForm,'customerForm':customerForm}
if request.method=='POST':
userForm=forms.CustomerUserForm(request.POST)
customerForm=forms.CustomerForm(request.POST,request.FILES)
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:06 PM 21-05-2023

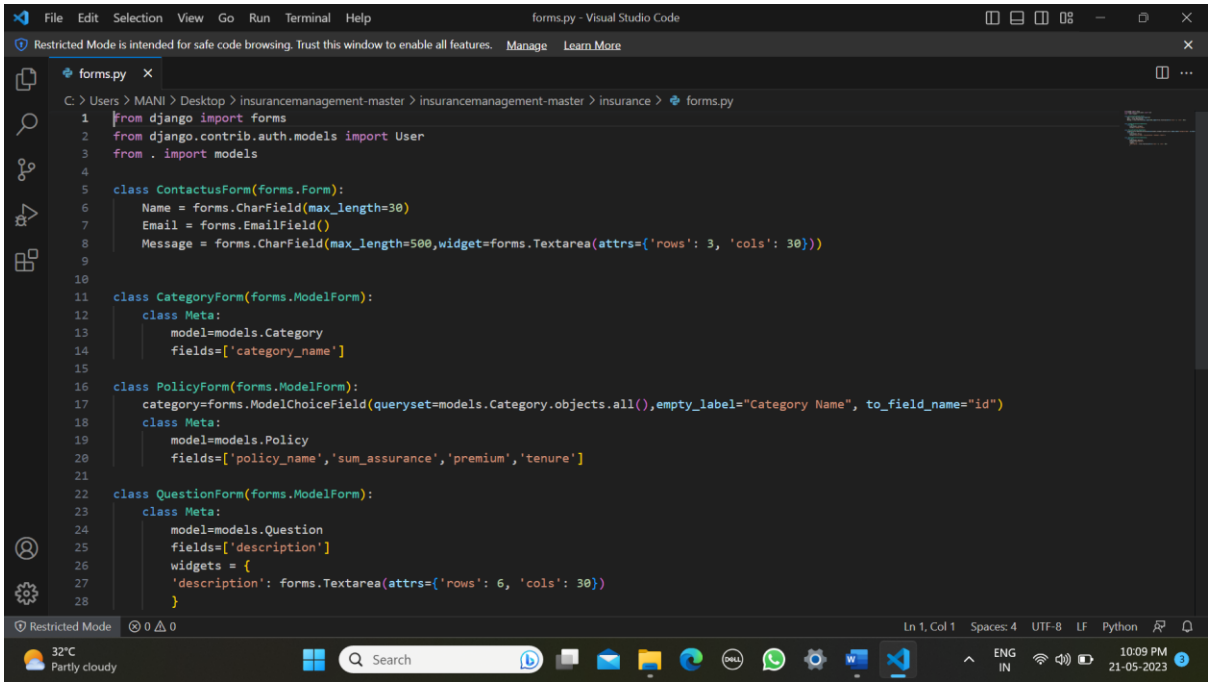


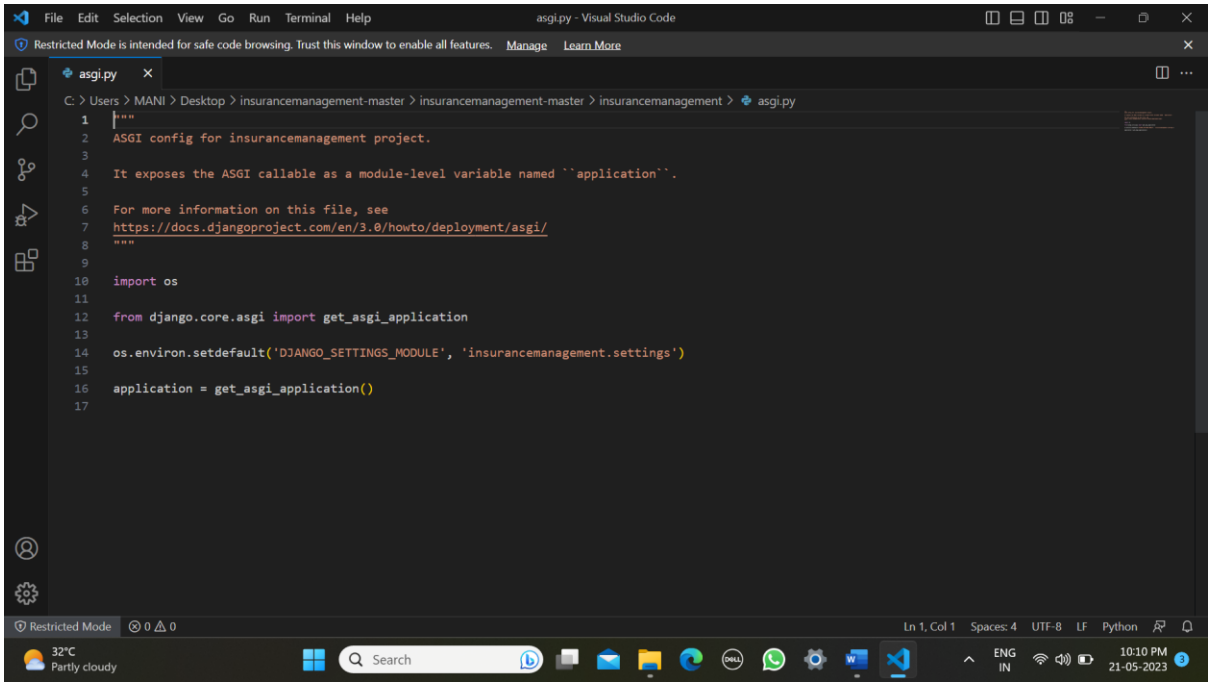
```
        customerForm=forms.CustomerForm(request.POST,request.FILES)
        if userForm.is_valid() and customerForm.is_valid():
            user=userForm.save()
            user.set_password(user.password)
            user.save()
            customer=customerForm.save(commit=False)
            customer.user=user
            customer.save()
            my_customer_group = Group.objects.get_or_create(name='CUSTOMER')
            my_customer_group[0].user_set.add(user)
        return HttpResponseRedirect('customerlogin')
    return render(request,'customer/customersignup.html',context=mydict)

def is_customer(user):
    return user.groups.filter(name='CUSTOMER').exists()

@login_required(login_url='customerlogin')
def customer_dashboard_view(request):
    dict={
        'customer':models.Customer.objects.get(user_id=request.user.id),
        'available_policy':CMODEL.Policy.objects.all().count(),
        'applied_policy':CMODEL.PolicyRecord.objects.all().filter(customer=models.Customer.objects.get(user_id=request.user.id)).count(),
        'total_category':CMODEL.Category.objects.all().count(),
        'total_question':CMODEL.Question.objects.all().filter(customer=models.Customer.objects.get(user_id=request.user.id)).count(),

    }
    return render(request,'customer/customer_dashboard.html',context=dict)
```

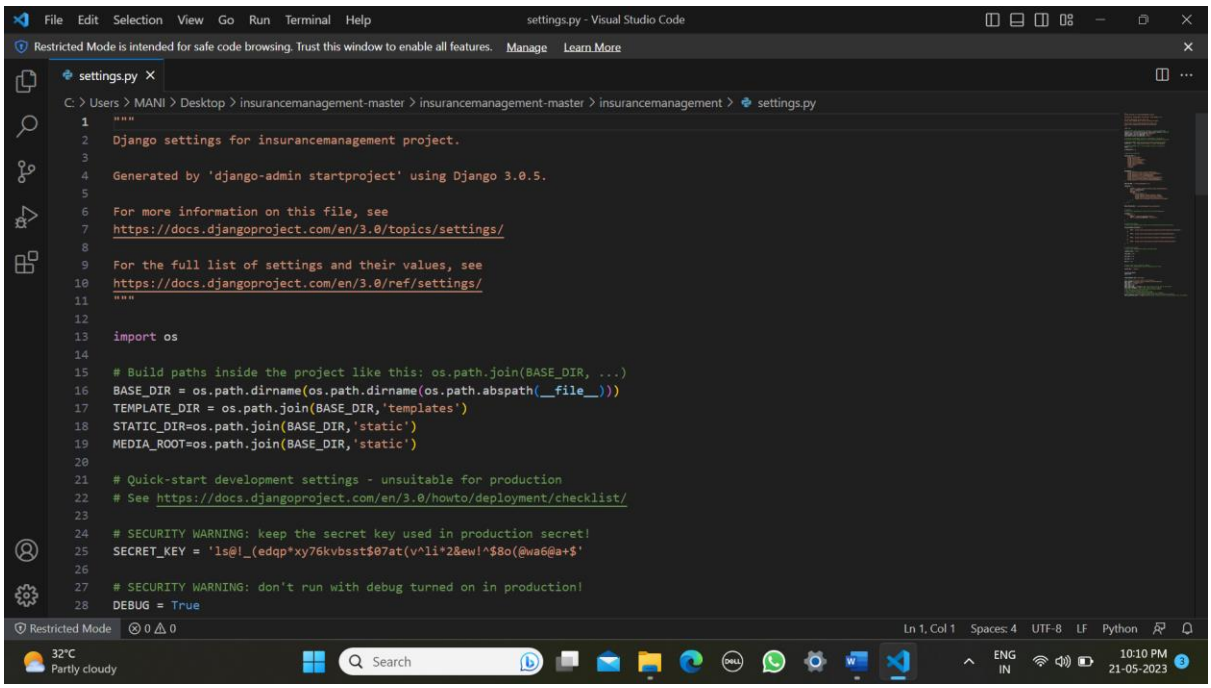


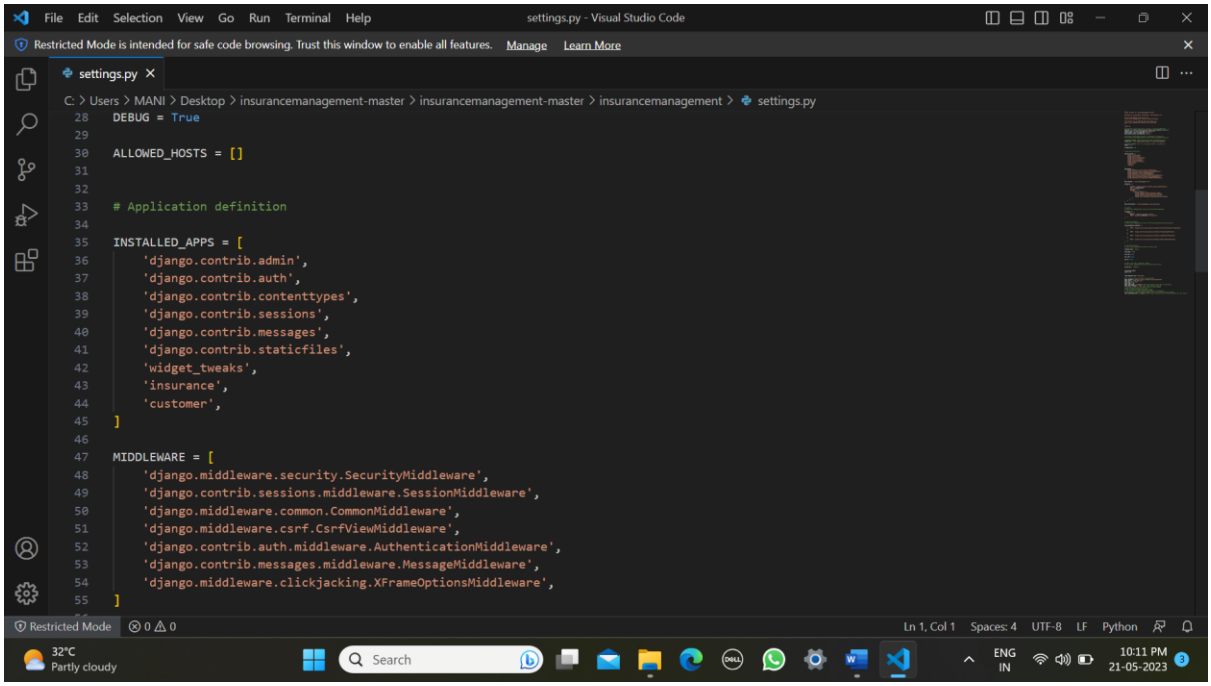
```
    return render(request,'customer/customer_dashboard.html',context=dict)

def apply_policy_view(request):
    customer = models.Customer.objects.get(user_id=request.user.id)
    policies = CMODEL.Policy.objects.all()
    return render(request,'customer/apply_policy.html',{'policies':policies,'customer':customer})

def apply_view(request,pk):
    customer = models.Customer.objects.get(user_id=request.user.id)
    policy = CMODEL.Policy.objects.get(id=pk)
    policyrecord = CMODEL.PolicyRecord()
    policyrecord.Policy = policy
    policyrecord.customer = customer
    policyrecord.save()
    return redirect('history')

def history_view(request):
    customer = models.Customer.objects.get(user_id=request.user.id)
    policies = CMODEL.PolicyRecord.objects.all().filter(customer=customer)
    return render(request,'customer/history.html',{'policies':policies,'customer':customer})

def ask_question_view(request):
    customer = models.Customer.objects.get(user_id=request.user.id)
    questionForm=CFORM.QuestionForm()

    if request.method=='POST':
        questionForm=CFORM.QuestionForm(request.POST)
```

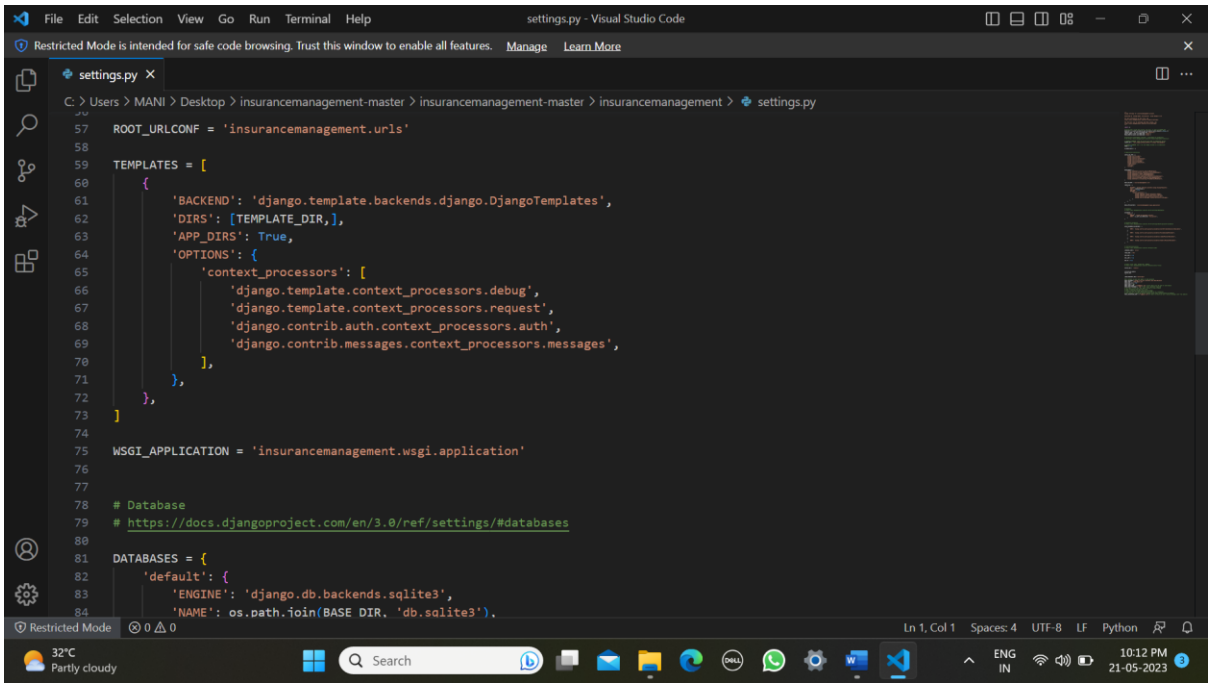

File Edit Selection View Go Run Terminal Help
views.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
urls.py
views.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > customer > views.py
policies = CMODEL.PolicyRecord.objects.all().filter(customer=customer)
return render(request,'customer/history.html',{'policies':policies,'customer':customer})
def ask_question_view(request):
customer = models.Customer.objects.get(user_id=request.user.id)
questionForm=CFORM.QuestionForm()
if request.method=='POST':
questionForm=CFORM.QuestionForm(request.POST)
if questionForm.is_valid():
question = questionForm.save(commit=False)
question.customer=customer
question.save()
return redirect('question-history')
return render(request,'customer/ask_question.html',{'questionForm':questionForm,'customer':customer})
def question_history_view(request):
customer = models.Customer.objects.get(user_id=request.user.id)
questions = CMODEL.Question.objects.all().filter(customer=customer)
return render(request,'customer/question_history.html',{'questions':questions,'customer':customer})
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:08 PM 21-05-2023

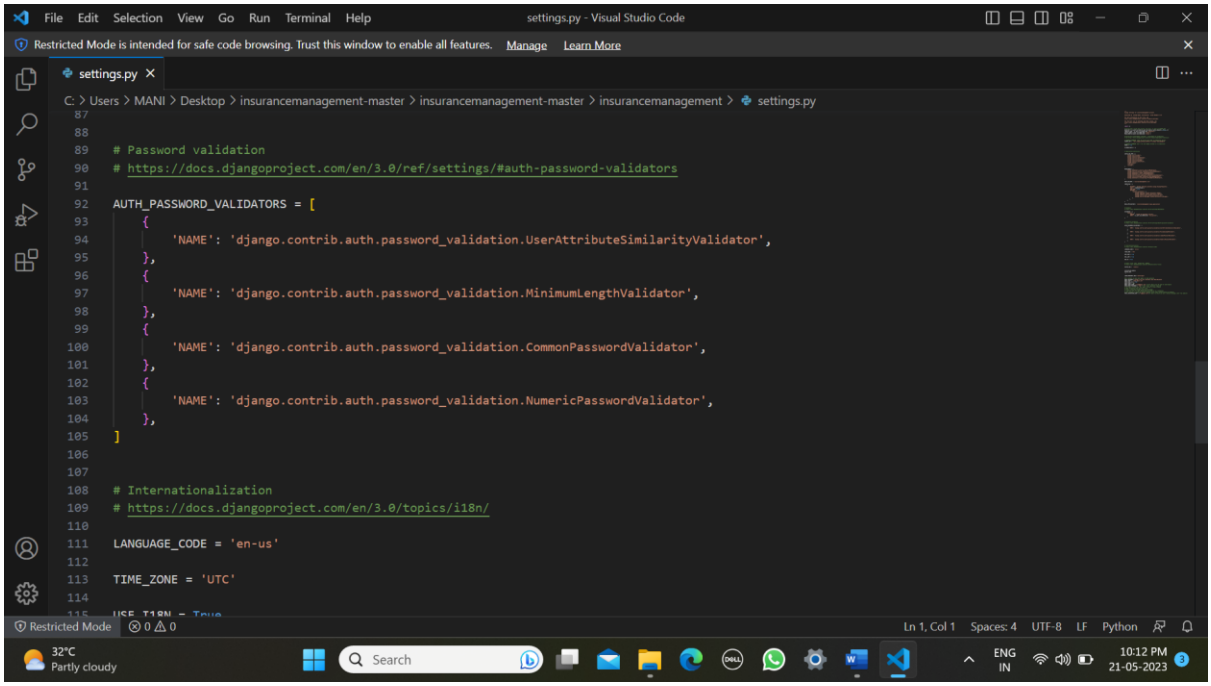

File Edit Selection View Go Run Terminal Help
forms.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
forms.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurance > forms.py
from django import forms
from django.contrib.auth.models import User
from . import models
class ContactusForm(forms.Form):
Name = forms.CharField(max_length=30)
Email = forms.EmailField()
Message = forms.CharField(max_length=500,widget=forms.Textarea(attrs={'rows': 3, 'cols': 30}))
class CategoryForm(forms.ModelForm):
class Meta:
model=models.Category
fields=['category_name']
class PolicyForm(forms.ModelForm):
category=forms.ModelChoiceField(queryset=models.Category.objects.all(),empty_label="Category Name", to_field_name="id")
class Meta:
model=models.Policy
fields=['policy_name','sum_assurance','premium','tenure']
class QuestionForm(forms.ModelForm):
class Meta:
model=models.Question
fields=['description']
widgets = {
'description': forms.Textarea(attrs={'rows': 6, 'cols': 30})
}
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:09 PM 21-05-2023

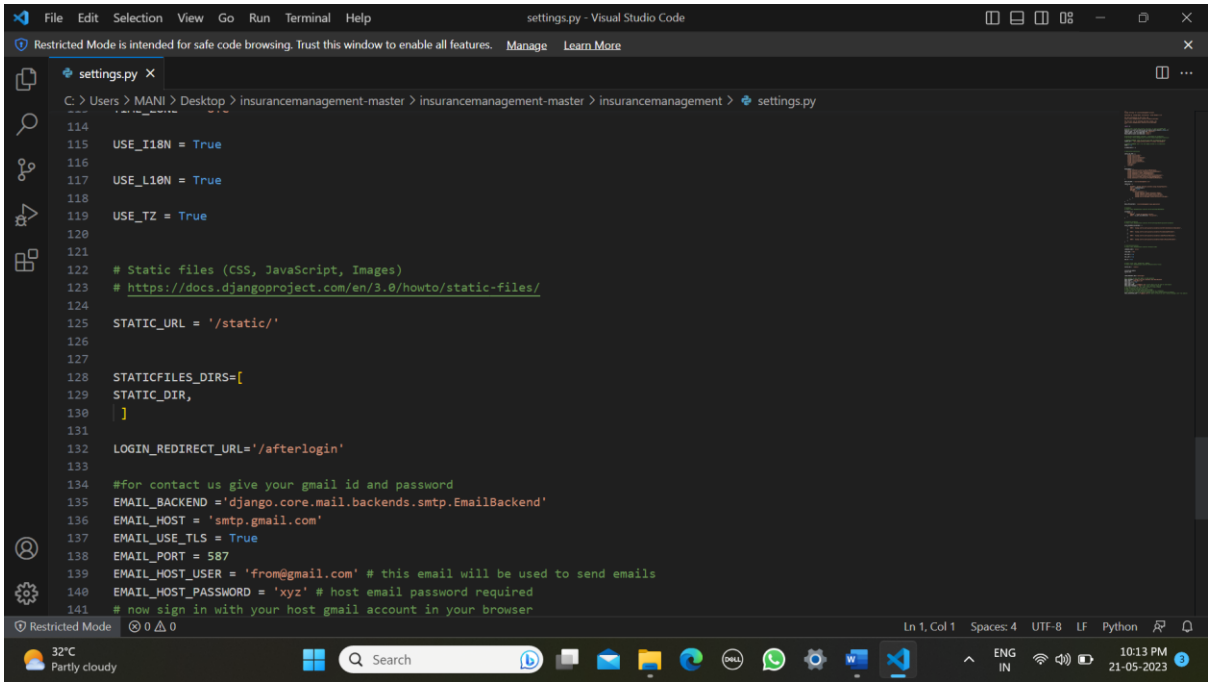

File Edit Selection View Go Run Terminal Help
asgi.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
asgi.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > asgi.py
"""
ASGI config for insurancemanagement project.
It exposes the ASGI callable as a module-level variable named ``application``.
For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/
"""
import os
from django.core.asgi import get_asgi_application
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'insurancemanagement.settings')
application = get_asgi_application()
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:10 PM 21-05-2023

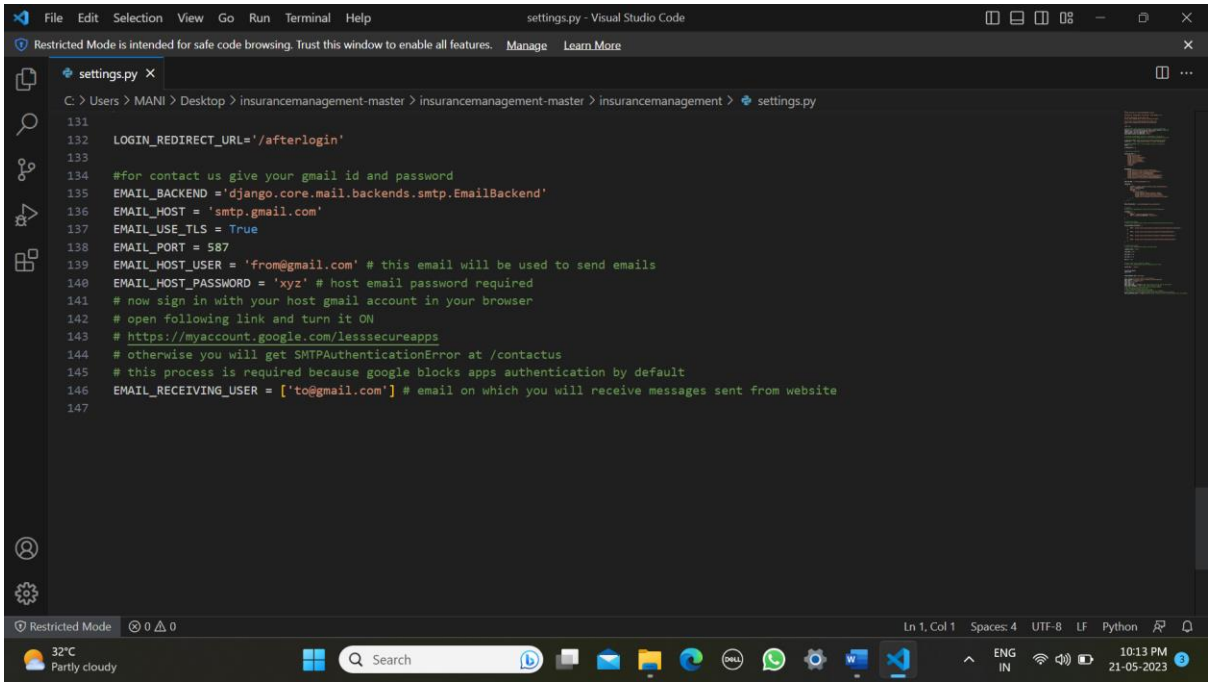

File Edit Selection View Go Run Terminal Help
settings.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
settings.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > settings.py
"""
Django settings for insurancemanagement project.
Generated by 'django-admin startproject' using Django 3.0.5.
For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""
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__)))
TEMPLATE_DIR = os.path.join(BASE_DIR,'templates')
STATIC_DIR=os.path.join(BASE_DIR,'static')
MEDIA_ROOT=os.path.join(BASE_DIR,'static')
# 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 = '1s@!_(edqp*xy76kvbsst$07at(v^li*2&ew!^$8o(@wa6@a+$'
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:10 PM 21-05-2023

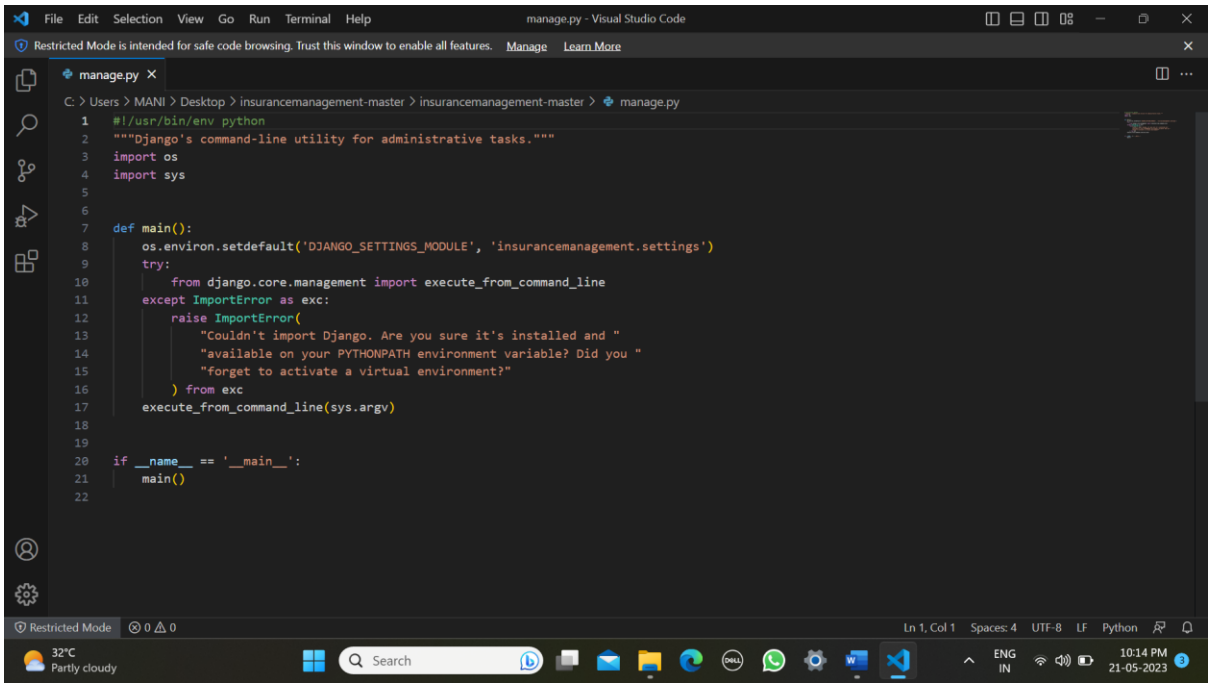

File Edit Selection View Go Run Terminal Help
settings.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
settings.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > settings.py
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',
'widget_tweaks',
'insurance',
'customer',
]
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',
]
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:11 PM 21-05-2023

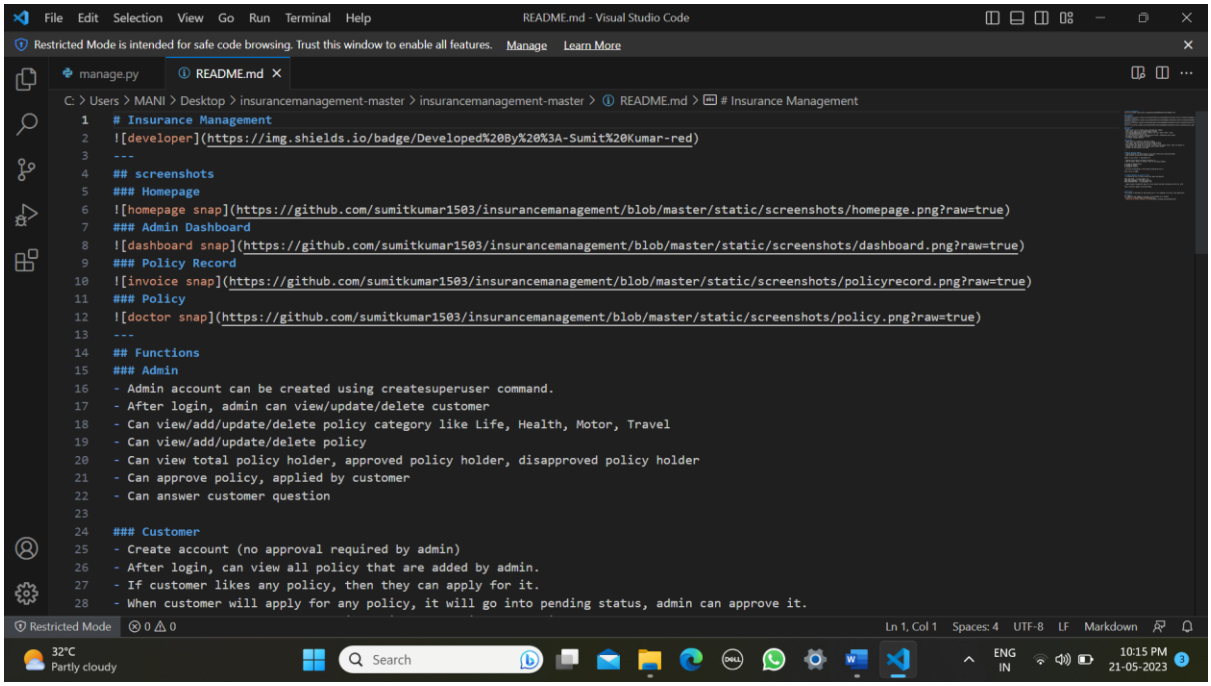

File Edit Selection View Go Run Terminal Help
settings.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
settings.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > settings.py
ROOT_URLCONF = 'insurancemanagement.urls'
TEMPLATES = [
{
'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': [TEMPLATE_DIR,],
'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 = 'insurancemanagement.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'),
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:12 PM 21-05-2023


File Edit Selection View Go Run Terminal Help
settings.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
settings.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > settings.py
# 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'
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:12 PM 21-05-2023

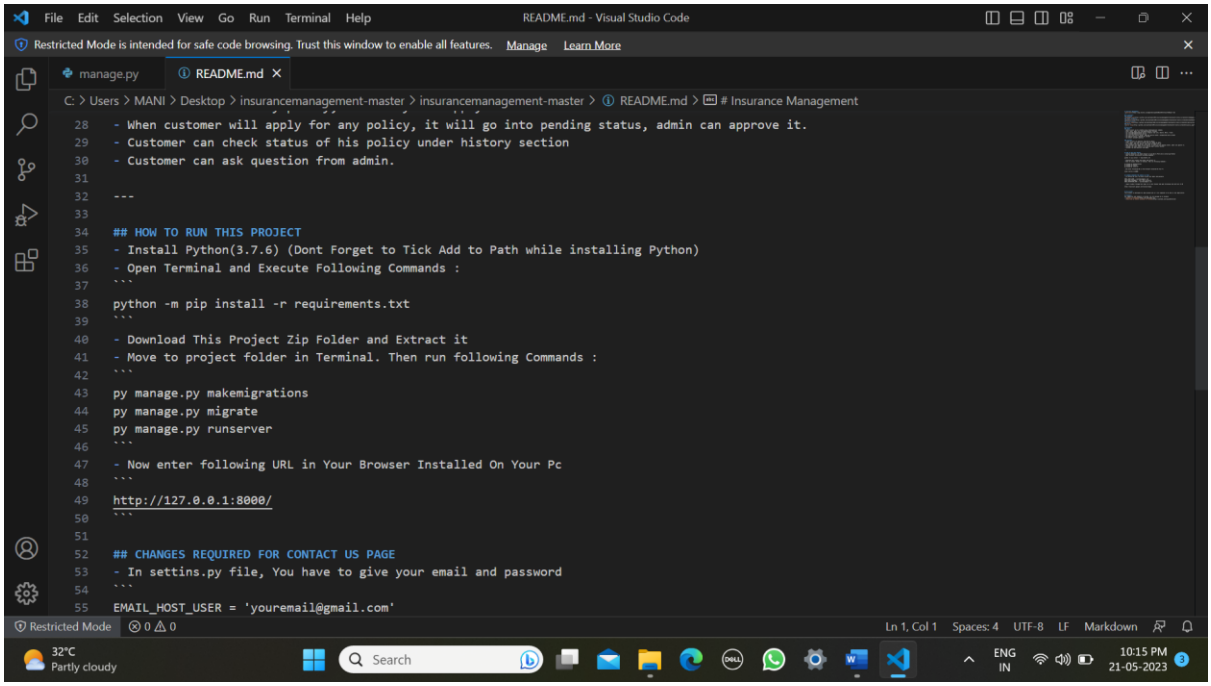

File Edit Selection View Go Run Terminal Help
settings.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
settings.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > settings.py
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/'
STATICFILES_DIRS=[
STATIC_DIR,
]
LOGIN_REDIRECT_URL='/afterlogin'
#for contact us give your gmail id and password
EMAIL_BACKEND ='django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = 'from@gmail.com' # this email will be used to send emails
EMAIL_HOST_PASSWORD = 'xyz' # host email password required
# now sign in with your host gmail account in your browser
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:13 PM 21-05-2023


File Edit Selection View Go Run Terminal Help
settings.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
settings.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurancemanagement > settings.py
131
132 LOGIN_REDIRECT_URL='/afterlogin'
133
134 #for contact us give your gmail id and password
135 EMAIL_BACKEND ='django.core.mail.backends.smtp.EmailBackend'
136 EMAIL_HOST = 'smtp.gmail.com'
137 EMAIL_USE_TLS = True
138 EMAIL_PORT = 587
139 EMAIL_HOST_USER = 'from@gmail.com' # this email will be used to send emails
140 EMAIL_HOST_PASSWORD = 'xyz' # host email password required
141 # now sign in with your host gmail account in your browser
142 # open following link and turn it ON
143 # https://myaccount.google.com/lesssecureapps
144 # otherwise you will get SMTPAuthenticationError at /contactus
145 # this process is required because google blocks apps authentication by default
146 EMAIL_RECEIVING_USER = ['to@gmail.com'] # email on which you will receive messages sent from website
147
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:13 PM 21-05-2023

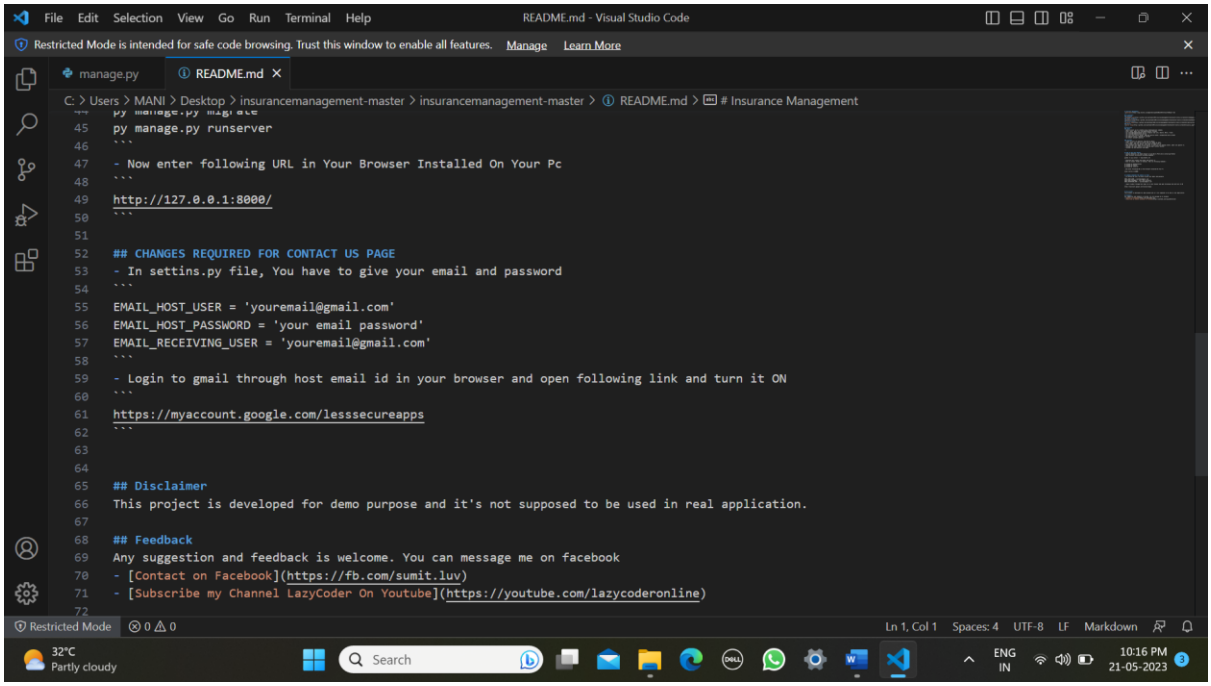

File Edit Selection View Go Run Terminal Help
manage.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
manage.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > manage.py
1 #!/usr/bin/env python
2 """Django's command-line utility for administrative tasks."""
3 import os
4 import sys
5
6
7 def main():
8 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'insurancemanagement.settings')
9 try:
10 from django.core.management import execute_from_command_line
11 except ImportError as exc:
12 raise ImportError(
13 "Couldn't import Django. Are you sure it's installed and "
14 "available on your PYTHONPATH environment variable? Did you "
15 "forget to activate a virtual environment?"
16) from exc
17 execute_from_command_line(sys.argv)
18
19
20 if __name__ == '__main__':
21 main()
22
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:14 PM 21-05-2023

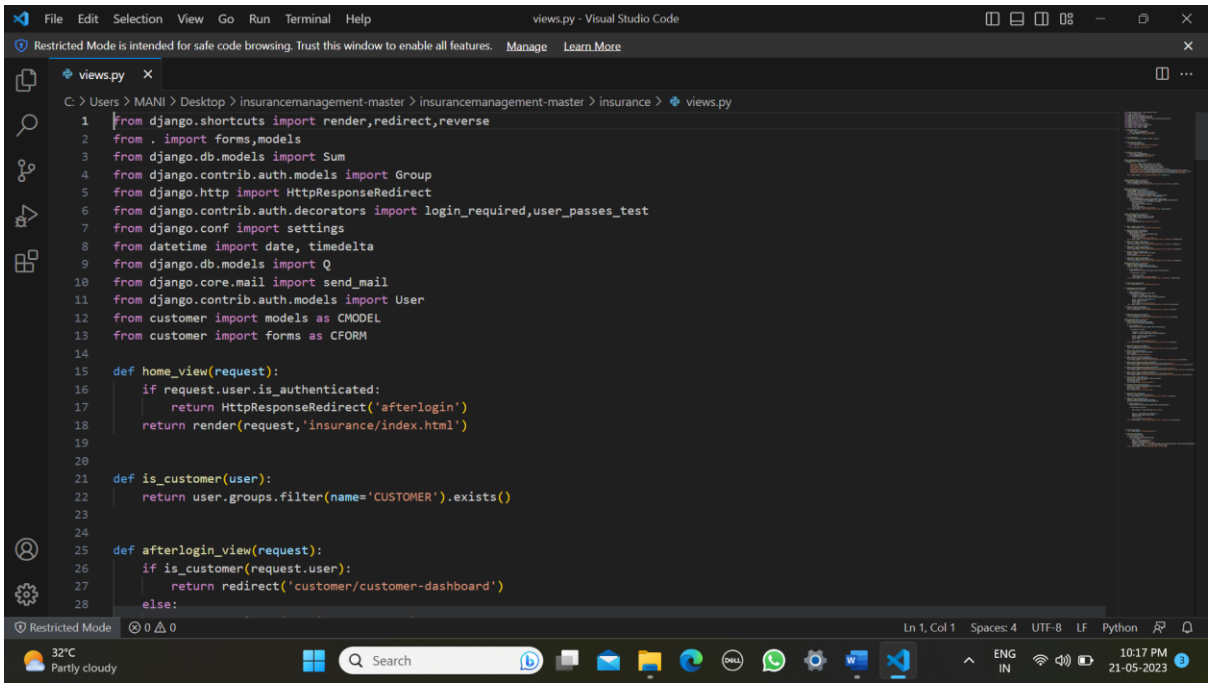

File Edit Selection View Go Run Terminal Help
README.md - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
manage.py
README.md
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > README.md > # Insurance Management
Insurance Management
![developer](https://img.shields.io/badge/Developed%20By%20%3A-Sumit%20Kumar-red)
---
screenshots
Homepage
![homepage snap](https://github.com/sumitkumar1503/insurancemanagement/blob/master/static/screenshots/homepage.png?raw=true)
Admin Dashboard
![dashboard snap](https://github.com/sumitkumar1503/insurancemanagement/blob/master/static/screenshots/dashboard.png?raw=true)
Policy Record
![invoice snap](https://github.com/sumitkumar1503/insurancemanagement/blob/master/static/screenshots/policyrecord.png?raw=true)
Policy
![doctor snap](https://github.com/sumitkumar1503/insurancemanagement/blob/master/static/screenshots/policy.png?raw=true)

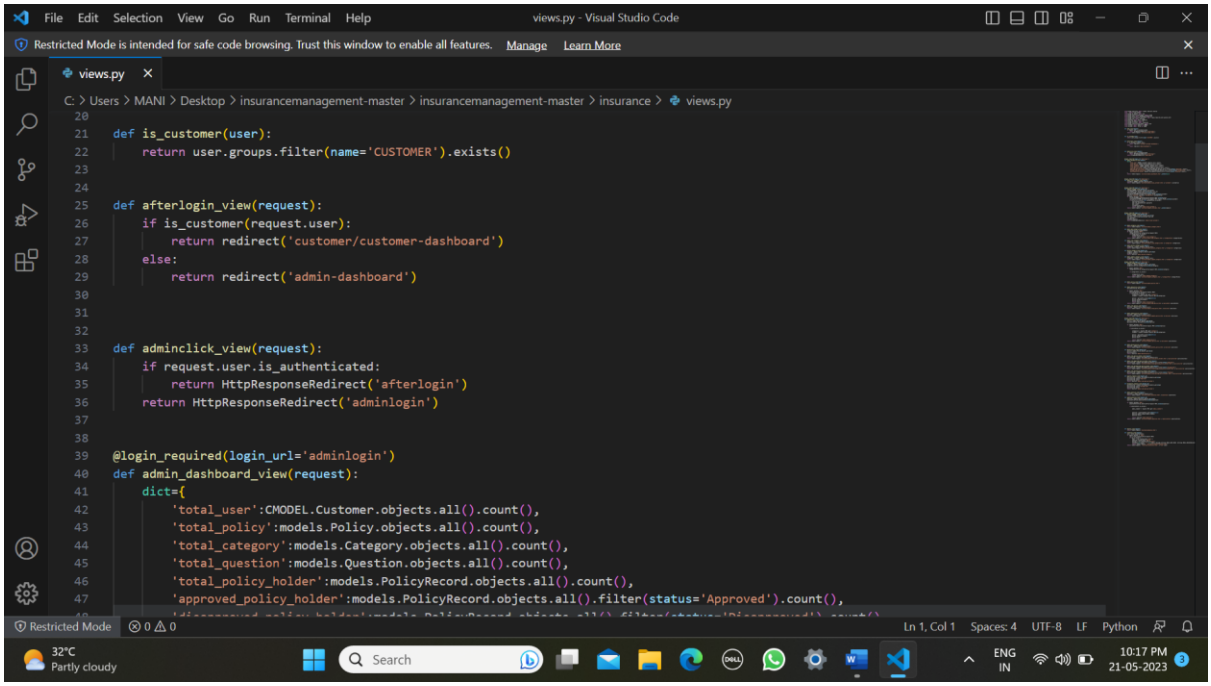
Functions
Admin
- Admin account can be created using createsuperuser command.
- After login, admin can view/update/delete customer
- Can view/add/update/delete policy category like Life, Health, Motor, Travel
- Can view/add/update/delete policy
- Can view total policy holder, approved policy holder, disapproved policy holder
- Can approve policy, applied by customer
- Can answer customer question
Customer
- Create account (no approval required by admin)
- After login, can view all policy that are added by admin.
- If customer likes any policy, then they can apply for it.
- When customer will apply for any policy, it will go into pending status, admin can approve it.
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Markdown
32°C Partly cloudy
Search
ENG IN
10:15 PM 21-05-2023

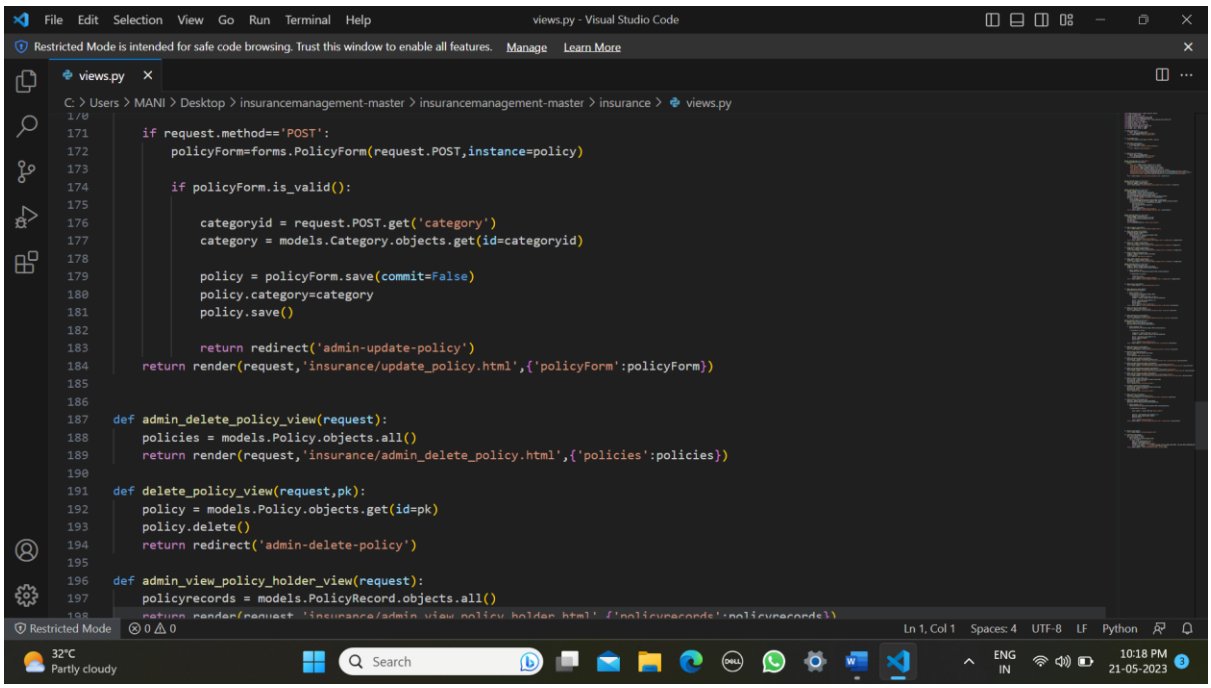

File Edit Selection View Go Run Terminal Help
README.md - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
manage.py
README.md
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > README.md > # Insurance Management
- When customer will apply for any policy, it will go into pending status, admin can approve it.
- Customer can check status of his policy under history section
- Customer can ask question from admin.

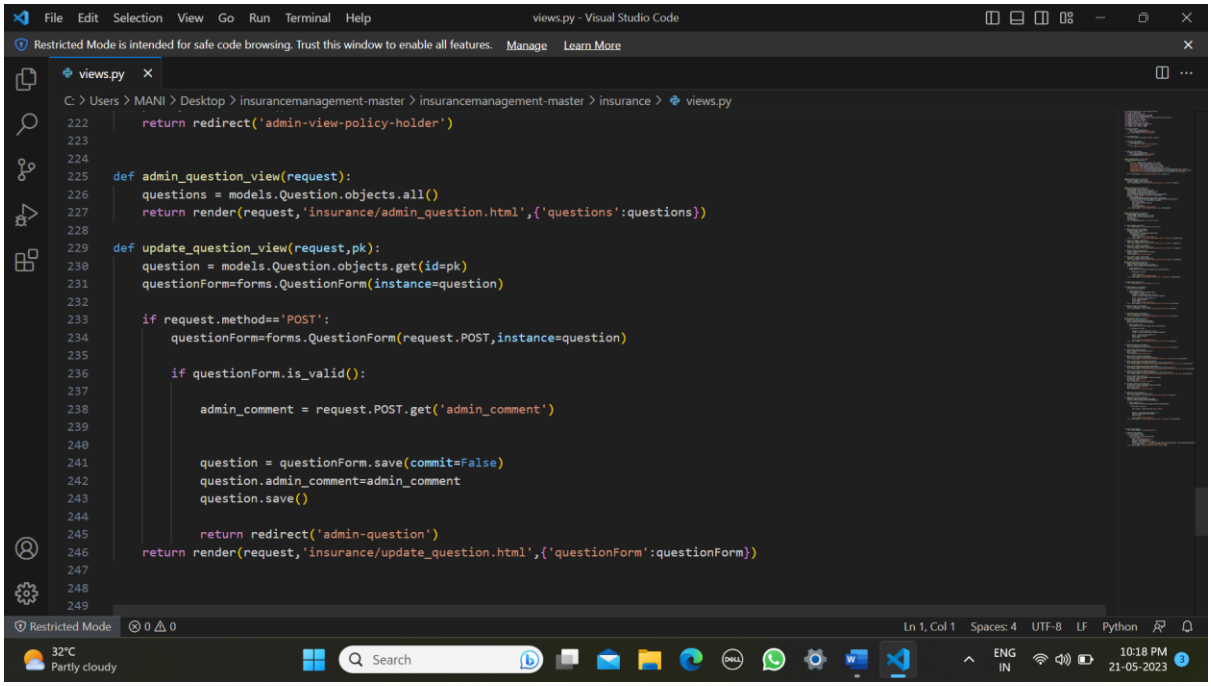
HOW TO RUN THIS PROJECT
- Install Python(3.7.6) (Dont Forget to Tick Add to Path while installing Python)
- Open Terminal and Execute Following Commands :
```
python -m pip install -r requirements.txt
```
- Download This Project Zip Folder and Extract it
- Move to project folder in Terminal. Then run following Commands :
```
py manage.py makemigrations
py manage.py migrate
py manage.py runserver
```
- Now enter following URL in Your Browser Installed On Your Pc
```
http://127.0.0.1:8000/
```
CHANGES REQUIRED FOR CONTACT US PAGE
- In settins.py file, You have to give your email and password
```
EMAIL_HOST_USER = 'youremail@gmail.com'
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Markdown
32°C Partly cloudy
Search
ENG IN
10:15 PM 21-05-2023
```


File Edit Selection View Go Run Terminal Help
README.md - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
manage.py
README.md
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > README.md > # Insurance Management
py manage.py runserver
- Now enter following URL in Your Browser Installed On Your Pc
http://127.0.0.1:8000/
CHANGES REQUIRED FOR CONTACT US PAGE
- In settins.py file, You have to give your email and password
EMAIL_HOST_USER = 'youremail@gmail.com'
EMAIL_HOST_PASSWORD = 'your email password'
EMAIL_RECEIVING_USER = 'youremail@gmail.com'
- Login to gmail through host email id in your browser and open following link and turn it ON
https://myaccount.google.com/lesssecureapps
Disclaimer
This project is developed for demo purpose and it's not supposed to be used in real application.
Feedback
Any suggestion and feedback is welcome. You can message me on facebook
- [Contact on Facebook](https://fb.com/sumit.luv)
- [Subscribe my Channel LazyCoder On Youtube](https://youtube.com/lazycoderonline)
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Markdown
32°C Partly cloudy
Search
ENG IN
10:16 PM 21-05-2023


File Edit Selection View Go Run Terminal Help
views.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
views.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurance > views.py
from django.shortcuts import render,redirect,reverse
from . import forms,models
from django.db.models import Sum
from django.contrib.auth.models import Group
from django.http import HttpResponseRedirect
from django.contrib.auth.decorators import login_required,user_passes_test
from django.conf import settings
from datetime import date, timedelta
from django.db.models import Q
from django.core.mail import send_mail
from django.contrib.auth.models import User
from customer import models as CMODEL
from customer import forms as CFORM
def home_view(request):
if request.user.is_authenticated:
return HttpResponseRedirect('afterlogin')
return render(request,'insurance/index.html')
def is_customer(user):
return user.groups.filter(name='CUSTOMER').exists()
def afterlogin_view(request):
if is_customer(request.user):
return redirect('customer/customer-dashboard')
else:
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:17 PM 21-05-2023


File Edit Selection View Go Run Terminal Help
views.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
views.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurance > views.py
def is_customer(user):
return user.groups.filter(name='CUSTOMER').exists()
def afterlogin_view(request):
if is_customer(request.user):
return redirect('customer/customer-dashboard')
else:
return redirect('admin-dashboard')
def adminclick_view(request):
if request.user.is_authenticated:
return HttpResponseRedirect('afterlogin')
return HttpResponseRedirect('adminlogin')
@login_required(login_url='adminlogin')
def admin_dashboard_view(request):
dict={
'total_user':CMODEL.Customer.objects.all().count(),
'total_policy':models.Policy.objects.all().count(),
'total_category':models.Category.objects.all().count(),
'total_question':models.Question.objects.all().count(),
'total_policy_holder':models.PolicyRecord.objects.all().count(),
'approved_policy_holder':models.PolicyRecord.objects.all().filter(status='Approved').count(),
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:17 PM 21-05-2023


File Edit Selection View Go Run Terminal Help
views.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
views.py
C: > Users > MANI > Desktop > insurancemanagement-master > insurancemanagement-master > insurance > views.py
if request.method=='POST':
policyForm=forms.PolicyForm(request.POST,instance=policy)
if policyForm.is_valid():
categoryid = request.POST.get('category')
category = models.Category.objects.get(id=categoryid)
policy = policyForm.save(commit=False)
policy.category=category
policy.save()
return redirect('admin-update-policy')
return render(request,'insurance/update_policy.html',{'policyForm':policyForm})
def admin_delete_policy_view(request):
policies = models.Policy.objects.all()
return render(request,'insurance/admin_delete_policy.html',{'policies':policies})
def delete_policy_view(request,pk):
policy = models.Policy.objects.get(id=pk)
policy.delete()
return redirect('admin-delete-policy')
def admin_view_policy_holder_view(request):
policyrecords = models.PolicyRecord.objects.all()
Restricted Mode
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python
32°C Partly cloudy
Search
ENG IN
10:18 PM 21-05-2023



GitHub & Project Video Demo Link

Github Link :

https://github.com/Majestic-Mani/Auto-insurance.git

Project Demo link:

https://screenrec.com/share/FDLBG8j7zE