Project Name: Access Recertification Web Application

Overview

The Access Recertification Web Application is a Python-based solution that automates the process of access recertification for users by role approver/owner for their role family. The application allows role owners to review the access rights of their role family members on a bi-annual/annual basis and make decisions on access approval or revocation. The application also includes a comment section for approvers to add any necessary comments, and sign off the review once completed. The application also triggers notifications to role owners/approvers once the review and sign-off process is complete.

Objectives

The objective of this project is to automate the manual process of access recertification that is currently managed in Excel sheets. The manual process is time-consuming and error-prone, and automating the process will improve efficiency and accuracy.

Features

The Access Recertification Web Application will include the following features:

Login page for role owners/approvers

Dashboard to view all assigned reviews

Review page for each assigned review

Ability to approve or revoke access

Comment section for approvers to add any necessary comments

Sign-off button for role owners to indicate that the review is complete

Notification system to notify role owners/approvers once the review and sign-off process is complete

To implement the Access Recertification Web Application with the features you listed, you will need to:

* Create a login page for role owners/approvers

This page should prompt users to enter their login credentials, which should be authenticated against a user database.

* Create a dashboard to view all assigned reviews

This dashboard should display a list of reviews assigned to each role owner/approver. It should include details such as the review name, review status, and due date.

* Create a review page for each assigned review

This page should display the details of the review, such as the user or group being reviewed, the permissions they have, and any other relevant information. It should also allow the approver to approve or revoke access and add any necessary comments.

* Implement the ability to approve or revoke access

The approver should be able to approve or revoke access from within the review page.

* Implement a comment section for approvers to add any necessary comments

The comment section should allow the approver to add comments regarding the review, the decision made, or any other relevant information.

* Implement a sign-off button for role owners to indicate that the review is complete

The sign-off button should be present on the review page and should allow the role owner to indicate that the review is complete.

* Implement a notification system to notify role owners/approvers once the review and sign-off process is complete

The notification system should send notifications to role owners/approvers once the review and sign-off process is complete.

Technologies Used

Python Django Web Framework

HTML, CSS, JavaScript

Bootstrap UI Framework

PostgreSQL Database

Django REST Framework for APIs

Workflow

The application will pull the access data from the IAM system and store it in the PostgreSQL database.

The application will send email notifications to role owners/approvers with a link to the review dashboard.

Role owners/approvers will login to the web application and view the list of assigned reviews on their dashboard.

Role owners/approvers will review the access rights of each user in their role family and make decisions on access approval or revocation.

Role approvers will add any necessary comments in the comment section.

Role owners will sign off on the review once completed.

The application will trigger a notification to the role owner/approver indicating that the review and sign-off process is complete.

User Roles

Role Owners: Responsible for reviewing the access rights of their role family members and signing off on the review.

Role Approvers: Responsible for reviewing the access rights of their role family members and making decisions on access approval or revocation.

Security Considerations

All user data will be stored in a secure database and access to the application will be authenticated and authorized.

Passwords will be hashed and salted before storing in the database.

The application will be hosted on a secure server with regular backups.

Conclusion

The Access Recertification Web Application will automate the manual process of access recertification, improve efficiency, and reduce errors. The application will be easy to use and will provide role owners/approvers with a simple and effective way to review access rights and make decisions.

|  |
| --- |
| Views.py |
| from django.shortcuts import render, redirect  from django.contrib.auth.decorators import login\_required  from django.contrib import messages  from .models import AccessReview  from .forms import ReviewForm  import csv  @login\_required  def dashboard(request):  """  Renders the dashboard for role owners/approvers to view assigned reviews.  """  reviews = AccessReview.objects.filter(role\_owner=request.user)  return render(request, 'dashboard.html', {'reviews': reviews})  @login\_required  def review(request, review\_id):  """  Renders the review page for a specific review.  """  review = AccessReview.objects.get(pk=review\_id)  if request.method == 'POST':  form = ReviewForm(request.POST, instance=review)  if form.is\_valid():  form.save()  messages.success(request, 'Review has been updated successfully.')  return redirect('dashboard')  else:  form = ReviewForm(instance=review)  return render(request, 'review.html', {'form': form, 'review': review})  @login\_required  def report(request, review\_id):  """  Generates a report for a specific review.  """  response = HttpResponse(content\_type='text/csv')  response['Content-Disposition'] = f'attachment; filename="review\_{review\_id}.csv"'  writer = csv.writer(response)  writer.writerow(['User', 'Access', 'Approval Status', 'Comments'])  review = AccessReview.objects.get(pk=review\_id)  for user in review.role\_family.all():  writer.writerow([user.username, user.access, user.approval\_status, user.comments])  return response  @login\_required  def history(request):  """  Renders the history page for role owners to view completed reviews.  """  reviews = AccessReview.objects.filter(role\_owner=request.user, sign\_off=True)  return render(request, 'history.html', {'reviews': reviews}) |
| Urls.py |
| from django.urls import path  from . import views  urlpatterns = [  path('', views.dashboard, name='dashboard'),  path('review/<int:review\_id>/', views.review, name='review'),  path('report/<int:review\_id>/', views.report, name='report'),  path('history/', views.history, name='history'),  ] |
| Dashboard.html |
| {% extends 'base.html' %}  {% block content %}  <h1>Access Recertification Dashboard</h1>  <p>Below is a list of access reviews assigned to you:</p>  <table>  <thead>  <tr>  <th>Review ID</th>  <th>Review Period</th>  <th>Status</th>  <th>Actions</th>  </tr>  </thead>  <tbody>  {% for review in reviews %}  <tr>  <td>{{ review.id }}</td>  <td>{{ review.review\_period }}</td>  <td>{{ review.status }}</td>  <td><a href="{% url 'review' review.id %}">Review</a></td>  </tr>  {% empty %}  <tr>  <td colspan="4">No access reviews assigned to you at this time.</td>  </tr>  {% endfor %}  </tbody>  </table>  {% endblock %} |
| Review.html |
| {% extends 'base.html' %}  {% block content %}  <h1>Access Review - {{ review.id }}</h1>  <p>Review period: {{ review.review\_period }}</ |

|  |
| --- |
| View.py |
| # views.py  from django.shortcuts import render, redirect  from django.contrib.auth.decorators import login\_required  from django.contrib import messages  from django.http import HttpResponse  from django.template.loader import get\_template  from xhtml2pdf import pisa  from .models import Review, RoleOwner, RoleApprover  from .forms import ReviewForm  @login\_required  def dashboard(request):  # Get the logged in user  user = request.user    # Check if the user is a role owner or role approver  if RoleOwner.objects.filter(user=user).exists():  reviews = Review.objects.filter(role\_owner=user.roleowner)  elif RoleApprover.objects.filter(user=user).exists():  reviews = Review.objects.filter(role\_approver=user.roleapprover)  else:  # Redirect to homepage if user is not a role owner or role approver  return redirect('home')    return render(request, 'dashboard.html', {'reviews': reviews})  @login\_required  def review(request, review\_id):  # Get the review object  review = Review.objects.get(id=review\_id)    # Check if the logged in user is authorized to perform the review  if not (request.user == review.role\_owner.user or request.user == review.role\_approver.user):  # Redirect to homepage if user is not authorized  return redirect('home')    # Get the review form  form = ReviewForm(instance=review)    if request.method == 'POST':  form = ReviewForm(request.POST, instance=review)  if form.is\_valid():  form.save()  messages.success(request, 'Review saved successfully.')  return redirect('dashboard')    return render(request, 'review.html', {'form': form, 'review': review})  @login\_required  def report(request, review\_id):  # Get the review object  review = Review.objects.get(id=review\_id)    # Check if the logged in user is authorized to view the report  if not (request.user == review.role\_owner.user or request.user == review.role\_approver.user):  # Redirect to homepage if user is not authorized  return redirect('home')    # Generate PDF report  template\_path = 'report.html'  context = {'review': review}  response = HttpResponse(content\_type='application/pdf')  response['Content-Disposition'] = 'filename="report.pdf"'  template = get\_template(template\_path)  html = template.render(context)  pisa\_status = pisa.CreatePDF(html, dest=response)  if pisa\_status.err:  return HttpResponse('An error occurred while generating the PDF report.')  return response |
| # models.py  from django.db import models  from django.contrib.auth.models import User  class RoleOwner(models.Model):  user = models.OneToOneField(User, on\_delete=models.CASCADE)  role\_family = models.CharField(max\_length=100)  class RoleApprover(models.Model):  user = models.OneToOneField(User, on\_delete=models.CASCADE)  role\_family = models.CharField(max\_length=100)  class Review(models.Model):  role\_owner = models.ForeignKey(RoleOwner, on\_delete=models.CASCADE)  role\_approver = models.ForeignKey(RoleApprover, on\_delete=models.CASCADE)  user\_name = models.CharField(max\_length=100)  access = models.CharField(max\_length=100)  approved = models.BooleanField(default=False)  revoked = models.BooleanField |
|  |
| #Models.py  from django.db import models  from django.contrib.auth.models import User  class Role(models.Model):  name = models.CharField(max\_length=255)  description = models.TextField(blank=True)  def \_\_str\_\_(self):  return self.name  class UserAccess(models.Model):  user = models.ForeignKey(User, on\_delete=models.CASCADE)  role = models.ForeignKey(Role, on\_delete=models.CASCADE)  access = models.CharField(max\_length=255)  def \_\_str\_\_(self):  return f"{self.user.username} - {self.role.name}" |
| #VIEWS.PY  from django.contrib.auth.decorators import login\_required  from django.shortcuts import render, get\_object\_or\_404  from django.urls import reverse  from django.http import HttpResponseRedirect  from .models import Role, UserAccess  @login\_required  def dashboard(request):  """  Dashboard for role owners/approvers to view all assigned reviews.  """  roles = Role.objects.all()  assigned\_reviews = {}  for role in roles:  assigned\_reviews[role.name] = UserAccess.objects.filter(role=role)  return render(request, 'dashboard.html', {'assigned\_reviews': assigned\_reviews})  @login\_required  def review(request, user\_access\_id):  """  Review page for each assigned review.  """  user\_access = get\_object\_or\_404(UserAccess, pk=user\_access\_id)  if request.method == 'POST':  user\_access.access = request.POST['access']  user\_access.comments = request.POST['comments']  user\_access.approver = request.user  user\_access.save()  return HttpResponseRedirect(reverse('dashboard'))  return render(request, 'review.html', {'user\_access': user\_access})  @login\_required  def history(request):  """  History page to view old reviews completed by the user.  """  user\_reviews = UserAccess.objects.filter(user=request.user)  return render(request, 'history.html', {'user\_reviews': user\_reviews}) |
| #Urls.py  from django.urls import path  from . import views  urlpatterns = [  path('dashboard/', views.dashboard, name='dashboard'),  path('review/<int:user\_access\_id>/', views.review, name='review'),  path('history/', views.history, name='history'),  ] |
| dashboard.html:  {% extends 'base.html' %}  {% block content %}  <h1>Dashboard</h1>  {% for role, reviews in assigned\_reviews.items %}  <h3>{{ role }}</h3>  <table>  <thead>  <tr>  <th>Username</th>  <th>Access</th>  <th>Comments</th>  <th>Action</th>  </tr>  </thead>  <tbody>  {% for review in reviews %}  <tr>  <td>{{ review.user.username }}</td>  <td>{{ review.access }}</td>  <td>{{ review.comments }}</td>  <td><a href="{% url 'review' review.id %}">Review</a></td>  </tr>  {% endfor %}  </tbody>  </table>  {% empty %}  <p>No assigned reviews.</p>  {% endfor %}  {% endblock %} |
| review.html: |
| {% extends 'base.html' %}  {% block content %}  <h1>Review</h1>  <form method="post">  {% csrf\_token %}  <p><strong>Username:</strong> {{ user\_access.user.username }}</p>  <p><strong>Role:</strong> {{ user\_access.role.name }}</p>  <p><strong |