**PCRAQ**

**Functionality**

1. Productivity\*
2. Clocking\* - (Facial recognition)
3. Reporting\* (Template)
4. Assessment\*, Training & Development Monitoring
5. Qualification\*
6. Key stroke\*
7. Messaging (Internal broadcast) & Chat
8. Generative AI, Machine Learning (ChartGPT)
9. **Hr component**
10. **Project Management component – Project tracking, Teams projects allocation, Budgeting functionality**
11. Asset Management
12. Risk Registry
13. Collaborative Platform (Developer Environment) (GitHub)
14. *Police*
15. *Health*
16. Data transfer (We Transfer)

**Human Resources Management (HRM) platform focusing on employee information and leave management,**

**Purpose:**

To manage and streamline employee-related data and leaves within an organization.

**Functions:**

1. Store employee information.
2. Record and manage leaves.

Let's begin with this simple prototype:

pythonCopy code

class Employee: def \_\_init\_\_(self, emp\_id, name, role): self.emp\_id = emp\_id self.name = name self.role = role self.leaves\_taken = 0 def request\_leave(self, days): self.leaves\_taken += days print(f"{self.name} has taken {days} days off. Total leaves taken: {self.leaves\_taken} days.") class HRM: def \_\_init\_\_(self): self.employees = {} def add\_employee(self, emp\_id, name, role): if emp\_id not in self.employees: new\_employee = Employee(emp\_id, name, role) self.employees[emp\_id] = new\_employee print(f"Employee {name} added successfully!") else: print(f"Employee ID {emp\_id} already exists!") def get\_employee(self, emp\_id): return self.employees.get(emp\_id, None) def list\_employees(self): for emp\_id, emp in self.employees.items(): print(f"ID: {emp\_id}, Name: {emp.name}, Role: {emp.role}, Leaves Taken: {emp.leaves\_taken} days") # Example Usage: hr\_system = HRM() hr\_system.add\_employee(1, "John Doe", "Software Engineer") hr\_system.add\_employee(2, "Jane Smith", "Data Scientist") employee = hr\_system.get\_employee(1) if employee: employee.request\_leave(5) hr\_system.list\_employees()

This code offers a basic framework for an HRM platform focusing on employee information and leave management. Important features such as authentication, persistent storage (e.g., using a database), and a user interface are omitted for simplicity. For a fully functional platform, you'd need to consider integrating with a database, adding security measures, and implementing a user-friendly interface.

**Project Management platform** using the agile methodology is a considerable task. Here's a high-level outline with a basic Python structure to get you started. This prototype will cover the concepts of projects, tasks (for project tracking), team allocation, and budgeting.

**Purpose:**

1. To manage and track projects using the agile methodology.
2. Assign teams to projects.
3. Budget projects and track expenses.

pythonCopy code

class Task: def \_\_init\_\_(self, title, description, status="To Do"): self.title = title self.description = description self.status = status # Can be "To Do", "In Progress", "Done" def update\_status(self, status): self.status = status class Project: def \_\_init\_\_(self, name, budget): self.name = name self.tasks = [] self.teams = [] self.budget = budget self.expenses = 0 def add\_task(self, title, description): task = Task(title, description) self.tasks.append(task) def allocate\_team(self, team): if team not in self.teams: self.teams.append(team) def add\_expense(self, amount): if self.expenses + amount <= self.budget: self.expenses += amount else: raise ValueError("Exceeds project budget!") class Team: def \_\_init\_\_(self, name): self.name = name self.members = [] def add\_member(self, member): self.members.append(member) class AgilePMPlatform: def \_\_init\_\_(self): self.projects = [] self.teams = [] def create\_project(self, name, budget): project = Project(name, budget) self.projects.append(project) def create\_team(self, name): team = Team(name) self.teams.append(team) def get\_project(self, name): return next((project for project in self.projects if project.name == name), None) def get\_team(self, name): return next((team for team in self.teams if team.name == name), None) def list\_projects(self): for project in self.projects: print(f"Project Name: {project.name}, Budget: ${project.budget}, Expenses: ${project.expenses}") def list\_teams(self): for team in self.teams: print(f"Team Name: {team.name}, Members: {', '.join(team.members)}") # Example Usage: platform = AgilePMPlatform() platform.create\_project("New Web App", 5000) platform.create\_team("Dev Team") project = platform.get\_project("New Web App") project.add\_task("Develop Frontend", "Frontend development using React") team = platform.get\_team("Dev Team") team.add\_member("John") team.add\_member("Jane") project.allocate\_team(team) project.add\_expense(1000) platform.list\_projects() platform.list\_teams()

This example provides a basic foundation for a project management platform. To transform this into a functional tool, you would need to add advanced features like user authentication, persistent storage with a database, a user interface, more complex task and project management features, and integrations with other tools. Consider using frameworks like Django or Flask for more extensive development.

**Creating a collaborative platform for software developers involves many components.** I'll outline a simplified version of such a platform using the Django framework, focusing on user registration, project creation, and collaboration on projects.

**Steps:**

1. **Setting Up the Django Project**:

First, you'll need to install Django:

Copy code

pip install django

Then create a new Django project and a new app:

bashCopy code

django-admin startproject collaborative\_platform cd collaborative\_platform django-admin startapp developers\_app

1. **Model Definitions**:

In **developers\_app/models.py**:

pythonCopy code

from django.db import models from django.contrib.auth.models import User class Project(models.Model): name = models.CharField(max\_length=255) description = models.TextField() created\_by = models.ForeignKey(User, related\_name='projects', on\_delete=models.CASCADE) collaborators = models.ManyToManyField(User, related\_name='collaborating\_projects') class Task(models.Model): title = models.CharField(max\_length=255) description = models.TextField() project = models.ForeignKey(Project, related\_name='tasks', on\_delete=models.CASCADE) assigned\_to = models.ForeignKey(User, related\_name='tasks', on\_delete=models.SET\_NULL, null=True)

1. **User Registration & Authentication**:

Django provides a built-in user authentication system. You'll need to set up URLs and views for registration and login.

In **developers\_app/urls.py**:

pythonCopy code

from django.urls import path from django.contrib.auth import views as auth\_views from . import views urlpatterns = [ path('register/', views.register, name='register'), path('login/', auth\_views.LoginView.as\_view(), name='login'), path('logout/', auth\_views.LogoutView.as\_view(), name='logout'), ]

In **developers\_app/views.py**:

pythonCopy code

from django.shortcuts import render, redirect from django.contrib.auth.forms import UserCreationForm from django.contrib.auth import login def register(request): if request.method == 'POST': form = UserCreationForm(request.POST) if form.is\_valid(): user = form.save() login(request, user) return redirect('project\_list') else: form = UserCreationForm() return render(request, 'registration/register.html', {'form': form})

You'll also need to create the template **registration/register.html** for the registration form.

1. **Setting Up Views & Templates for Project Collaboration**:

Add views for creating projects, listing projects, viewing project details, and collaborating.

Continue in **developers\_app/views.py**:

pythonCopy code

from .models import Project from .forms import ProjectForm def project\_list(request): projects = Project.objects.all() return render(request, 'projects/project\_list.html', {'projects': projects}) def project\_detail(request, project\_id): project = Project.objects.get(pk=project\_id) return render(request, 'projects/project\_detail.html', {'project': project}) def project\_create(request): if request.method == 'POST': form = ProjectForm(request.POST) if form.is\_valid(): project = form.save(commit=False) project.created\_by = request.user project.save() return redirect('project\_list') else: form = ProjectForm() return render(request, 'projects/project\_form.html', {'form': form})

Similarly, create appropriate templates for listing projects (**project\_list.html**), displaying project details (**project\_detail.html**), and the project creation form (**project\_form.html**).

1. **Run Migrations**:

Copy code

python manage.py makemigrations python manage.py migrate

1. **Start the Server**:

Copy code

python manage.py runserver

This example provides a foundation to build upon. For a fully functional collaborative platform, you'll need features like notifications, comments on tasks, file uploads, integrations with version control systems, and more. Consider using Django's third-party packages and plugins to enhance functionality and streamline development.

**Mentorship management platform** entails features such as user registration (for mentors and mentees), mentorship program creation, and mentor-mentee matching. Here's a basic outline to get started with such a platform using the Django framework:

**Steps:**

1. **Setting Up the Django Project**:

Install Django:

Copy code

pip install django

Create a new Django project and a new app:

bashCopy code

django-admin startproject mentorship\_platform cd mentorship\_platform django-admin startapp mentorship\_app

1. **Model Definitions**:

In **mentorship\_app/models.py**:

pythonCopy code

from django.db import models from django.contrib.auth.models import User class Profile(models.Model): user = models.OneToOneField(User, on\_delete=models.CASCADE) is\_mentor = models.BooleanField(default=False) bio = models.TextField(blank=True) expertise = models.TextField(blank=True) class Mentorship(models.Model): mentor = models.ForeignKey(Profile, related\_name='mentorships\_as\_mentor', on\_delete=models.CASCADE) mentee = models.ForeignKey(Profile, related\_name='mentorships\_as\_mentee', on\_delete=models.CASCADE) start\_date = models.DateField() end\_date = models.DateField() description = models.TextField(blank=True)

1. **User Registration & Authentication**:

Django provides a built-in user authentication system. Set up URLs and views for registration and login.

In **mentorship\_app/urls.py**:

pythonCopy code

from django.urls import path from django.contrib.auth import views as auth\_views from . import views urlpatterns = [ path('register/', views.register, name='register'), path('login/', auth\_views.LoginView.as\_view(), name='login'), path('logout/', auth\_views.LogoutView.as\_view(), name='logout'), ]

In **mentorship\_app/views.py**:

pythonCopy code

from django.shortcuts import render, redirect from django.contrib.auth.forms import UserCreationForm from django.contrib.auth import login def register(request): if request.method == 'POST': form = UserCreationForm(request.POST) if form.is\_valid(): user = form.save() login(request, user) return redirect('profile\_create') else: form = UserCreationForm() return render(request, 'registration/register.html', {'form': form})

Create the template **registration/register.html** for the registration form.

1. **Setting Up Views & Templates for Mentorship**:

Add views for creating profiles, listing mentors, viewing profiles, and creating mentorship relationships.

Extend **mentorship\_app/views.py**:

pythonCopy code

from .models import Profile, Mentorship from .forms import ProfileForm, MentorshipForm def profile\_create(request): # logic for creating profiles def mentor\_list(request): mentors = Profile.objects.filter(is\_mentor=True) return render(request, 'profiles/mentor\_list.html', {'mentors': mentors}) def profile\_detail(request, profile\_id): profile = Profile.objects.get(pk=profile\_id) return render(request, 'profiles/profile\_detail.html', {'profile': profile}) def create\_mentorship(request): # logic for creating mentorship relationships

Create appropriate templates for each of the above views.

1. **Run Migrations**:

Copy code

python manage.py makemigrations python manage.py migrate

1. **Start the Server**:

Copy code

python manage.py runserver

This provides the foundation for a mentorship platform. For a full-fledged application, consider features like mentorship requests, reviews, feedback, notifications, and more. You might also want to utilize Django's third-party packages and plugins to enhance functionality.

**Software As Service**