

# IMS CONNECT DOCUMENTATION

- Introduction
  - Project Overview
  - Key Features
- Installation Instructions
  - Troubleshooting
- Configuration
  - Example settings.py snippet
- API Endpoints
  - User Endpoints
    - Example views.py snippet for UserViewSet
  - Idea Endpoints
    - Example models.py snippet for Idea model
  - Vote Endpoints
    - Example serializers.py snippet for VoteSerializer
  - Team Endpoints
  - Document Endpoints
  - Comment Endpoints
  - Notification Endpoints
  - Milestone Endpoints
  - Category Endpoints
  - Audit Log Endpoints
- View Logic
- Models
- Authentication
- Testing
- User Interfaces
  - Dashboard Pages
  - Idea Submission Page
  - My Ideas Page
  - Profile Page
  - Notifications Page
  - Settings Page
  - Audit Logs Page
  - Idea Review Page
  - Reports Page

- [Team Collaboration Page](#)
- [Milestones Page](#)

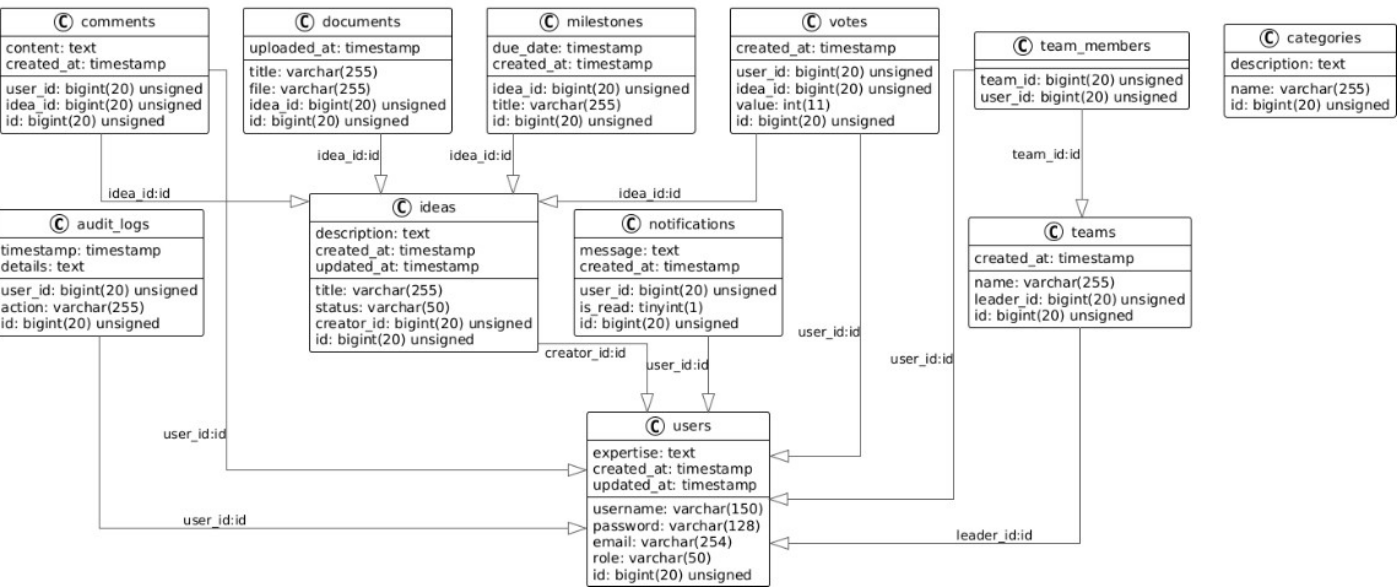
# Introduction

It is a very specialized project that aims to give users a space to collaborate and manage ideas within a software called `ims-connect-django-bootstrap-html-css-js-jquery` . This idea sharing application with a Django backend and a `bootstrap-html-css-js-jquery` frontend helps the users to create, vote and discuss ideas from one single platform. The first purpose is to increase efficiency among workers by improving means of communication and decision-making among the groups.

It will act as a record for developers and users of the application as it maps out how to install and set it up as well as features on how to use it. It contains information about a few important components and API interfaces, and gives an overview of the internals, making it simpler for new contributors to study and contribute to the project. This document covers all aspects required in order to configure, use, or deploy the application whether that is in a local environment or as part of a wider system.

# Project Overview

The `ims-connect-django-bootstrap` is a web application built on the top of Django web framework with bootstrap as a frontend. It is designed for sharing and idea implementation and growth giving users rights to post, vote and comment on ideas. The idea behind the application is to improve efficiency and the overall performance of the teams by having a tool where these ideas may be created and then assessed.



# Key Features

---

- Multifactor authentication and authorization by using JWT.
- Functions related to the creation, reading, updating and deleting, on ideas, votes, comments, documents, and teams notifications, new milestones, and categories, as well as, usage of audits logs.
- Real-time messages for the users' activity. Facilities for searching as well as the idea and for users.
- There is also ensured the concept of role-based working access control for different types of users.

## Installation Instructions

---

To set up the backend locally, follow these steps:

### 1. Clone the repository:

```
git clone https://github.com/yourusername/ims-connect-django-bootstrap-html-css-js-jquery .git cd
ims-connect-django-bootstrap/backend
```

### 2. Create a virtual environment and activate it:

```
python3 -m venv venv
source venv/bin/activate
```

### 3. Install the required packages:

```
pip install -r requirements.txt
```

### 4. Set up the database (make sure MySQL is running):

```
python manage.py migrate
```

### 5. Create a superuser to access the admin panel:

```
python manage.py createsuperuser
```

## 6. Run the development server:

```
python manage.py runserver
```

# Troubleshooting

---

- If you encounter issues with database connections, ensure that your MySQL server is running and that the credentials in your `.env` file are correct.
- If you face any package installation errors, ensure that you have the correct version of Python and pip installed.

# Configuration

---

Key settings from `settings.py`:

- **SECRET\_KEY**: Managed through environment variables for security. This key is used for cryptographic signing and should be kept secret in production.
- **DEBUG**: Set to `True` for development. In production, this should be set to `False` to prevent sensitive information from being exposed.
- **ALLOWED\_HOSTS**: Currently set to allow all hosts (`'*'`). This should be updated to include specific domains in production.
- **CORS\_ALLOWED\_ORIGINS**: Configured to allow requests from specific local development addresses. This is important for enabling cross-origin requests from the frontend.
- **DATABASES**: Uses MySQL as the database backend. Ensure that the database is created and the credentials are correct.

## Example `settings.py` snippet

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'rest_framework_simplejwt', 'corsheaders',  
    'users',  
    'ideas',  
    'votes',  
    'teams', 'documents',  
    'comments',  
    'notifications',  
    'milestones',  
    'categories',  
    'audit_logs',  
]
```

```

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    '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',
]

ROOT_URLCONF = 'backend.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'backend.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.environ.get('DB_NAME'),
        'USER': os.environ.get('DB_USER'),
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': os.environ.get('DB_HOST'),
        'PORT': os.environ.get('DB_PORT'),
    }
}

```

## API Endpoints

---

## User Endpoints

---

- **GET /users/**: List all users.

**Response** : A JSON array of user objects. It mostly helps the admin to manage the system users as all the registered users can be retrieved through this endpoint.

Example:

```
[
  {"id": 1, "username": "john_doe", "email": "john@example.com"},
  {"id": 2, "username": "jane_doe", "email": "jane@example.com"}
]
```

- ♦ **POST /users/**: Create a new user.

**Response** : JSON object with user details (username, password, etc.). This endpoint allows for the registration of new users into the system, facilitating user growth and engagement.

Example:

```
{"username": "new_user", "password": "securepassword"}
```

**Response**: The created user object, confirming successful registration.

## Example **views.py** snippet for UserViewSet

```
class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    def create(self, request):
        serializer = self.serializer_class(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
            status=status.HTTP_400_BAD_REQUEST)
```

## Idea Endpoints

- ♦ **GET /ideas/**: List all ideas.

**Response**: A JSON array of idea objects. This specifically focused endpoint involves a



complete collection of ideas submitted by users for easy tracking and assessment.

Example:

```
[
  {"id": 1, "title": "Idea 1", "description": "Description of Idea 1"},
  {"id": 2, "title": "Idea 2", "description": "Description of Idea 2"}
]
```

- **POST /ideas/**: Create a new idea.

**Request Body:** An object of JSON containing the details of the idea such as the title and description and so on. It is a post endpoint that enables users to post new ideas in the platform that will enhance innovation.

Example:

```
{"title": "New Idea", "description": "This is a new idea."}
```

- **Response:** The created idea object, confirming successful submission.

## Example `models.py` snippet for Idea model

```
from django.db import models

class Idea(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField()
    status = models.CharField(max_length=50, choices=[('draft', 'Draft'), ('submitted', 'Submitted'), ('approved', 'Approved')])
    creator = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

## Vote Endpoints

---

- ♦ **GET /votes/**: List all votes.

**Response:** A JSON array of vote objects. This particular endpoint serves to enable users view all the voting that takes place on ideas which increases transparency.

Example:

```
[
  {"id": 1, "user_id": 1, "idea_id": 1, "value": 1},
  {"id": 2, "user_id": 2, "idea_id": 1, "value": -1}
]
```

- ♦ **POST /votes/**: Create a new vote.

**Request Body:** JSON object that contains the additional information regarding the vote (user ID, idea ID and so on). This end point for example will enable users to vote on the ideas thus being able to proactively participate in the decision making process.

Example:

```
{"user_id": 1, "idea_id": 1, "value": 1}
```

- **Response:** The created vote object, confirming successful voting.

## Example **serializers.py** snippet for VoteSerializer

```
class VoteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Vote
        fields = ['id', 'user_id', 'idea_id', 'value']
```

## Team Endpoints

- ♦ **GET /teams/**: List all teams.

- **Response:** A JSON array of team objects. This endpoint helps with the management of all teams across the application and offers numerous relevant details.

and collaboration.

- **Example:**

```
[
  {"id": 1, "name": "Team Alpha"},
  {"id": 2, "name": "Team Beta"}
]
```

- ♦ **POST /teams/:** Create a new team.

- **Request Body:** JSON object with team details (name, leader ID, etc.). This endpoint allows users to form new teams, promoting collaboration on ideas.

- **Example:**

```
{"name": "New Team", "leader_id": 1}
```

- **Response:** The created team object, confirming successful team formation.

## Document Endpoints

---

- ♦ **GET /documents/:** List all documents.

- **Response:** A JSON array of document objects. This endpoint allows users to view all documents related to ideas, facilitating document management.

- **Example:**

```
[
  {"id": 1, "title": "Document 1", "file": "url_to_file"},
  {"id": 2, "title": "Document 2", "file": "url_to_file"}
]
```

- ♦ **POST /documents/:** Create a new document.

- **Request Body:** JSON object with document details (title, file, etc.). This endpoint allows users to upload new documents associated with their ideas.

- **Example:**

```
{"title": "New Document", "file": "url_to_file"}
```

- **Response:** The created document object, confirming successful upload.

## Comment Endpoints

---

- ♦ **GET /comments/:** List all comments.
  - **Response:** A JSON array of comment objects. This endpoint allows users to view all comments made on ideas, fostering discussion and feedback.
  - **Example:**

```
[  
  {"id": 1, "user_id": 1, "idea_id": 1, "content": "Great idea!"},  
  {"id": 2, "user_id": 2, "idea_id": 1, "content": "I agree!"}  
]
```

- ♦ **POST /comments/:** Create a new comment.
  - **Request Body:** JSON object with comment details (user ID, idea ID, content, etc.). This endpoint allows users to provide feedback on ideas through comments.
  - **Example:**

```
{"user_id": 1, "idea_id": 1, "content": "This is a comment."}
```

- **Response:** The created comment object, confirming successful posting.

## Notification Endpoints

---

- ♦ **GET /notifications/:** List all notifications.
  - **Response:** A JSON array of notification objects. This endpoint allows users to view all notifications related to their activities within the application.
  - **Example:**

```
[
  {"id": 1, "user_id": 1, "message": "You have a new comment.",
  "created_at": "2024-12-21T22:02:00Z"},
  {"id": 2, "user_id": 2, "message": "Your idea was approved.",
  "created_at": "2024-12-21T22:03:00Z"}
]
```

- ♦ **POST /notifications/**: Create a new notification.
  - **Request Body**: JSON object with notification details (user ID, message, etc.). This endpoint allows the system to send notifications to users about various activities.
  - **Example**:

```
{"user_id": 1, "message": "New vote on your idea."}
```

- **Response**: The created notification object, confirming successful notification.

## Milestone Endpoints

- ♦ **GET /milestones/**: List all milestones.
  - **Response**: A JSON array of milestone objects. This endpoint allows users to view all milestones associated with ideas, providing transparency in project timelines.
  - **Example**:

```
[
  {"id": 1, "idea_id": 1, "title": "Milestone 1", "due_date": "2024-12-31T00:00:00Z"},
  {"id": 2, "idea_id": 2, "title": "Milestone 2", "due_date": "2024-12-31T00:00:00Z"}
]
```

- ♦ **POST /milestones/**: Create a new milestone.
  - **Request Body**: JSON object with milestone details (idea ID, title, due date, etc.). This endpoint allows users to set milestones for their ideas, helping to track progress.

- **Example:**

```
{"idea_id": 1, "title": "New Milestone", "due_date": "2024-12-31T00:00:00Z"}
```

- **Response:** The created milestone object, confirming successful milestone creation.

## Category Endpoints

---

- ♦ **GET /categories/:** List all categories.

- **Response:** A JSON array of category objects. This endpoint allows users to view all categories available for organizing ideas.
- **Example:**

```
[  
  {"id": 1, "name": "Category 1"},  
  {"id": 2, "name": "Category 2"}  
]
```

- ♦ **POST /categories/:** Create a new category.

- **Request Body:** JSON object with category details (name, description, etc.). This endpoint allows users to define new categories for better organization of ideas.
- **Example:**

```
{"name": "New Category", "description": "Description of new category."}
```

- **Response:** The created category object, confirming successful category creation.

## Audit Log Endpoints

---

- ♦ **GET /audit-logs/:** List all audit logs.

- **Response:** A JSON array of audit log objects. This endpoint provides a record of all significant actions taken within the application, ensuring accountability and traceability.
- **Example:**

```
[
  {"id": 1, "user_id": 1, "action": "created idea", "timestamp":
"2024-12-21T22:00:00Z"},
  {"id": 2, "user_id": 2, "action": "voted", "timestamp": "2024-12-
21T22:01:00Z"}
]
```

- ♦ **POST /audit-logs/:** Create a new audit log.

- **Request Body:** JSON object with log details (user ID, action, etc.). This endpoint allows the system to record significant actions taken by users, enhancing transparency and accountability.
- **Example:**

```
{"user_id": 1, "action": "created idea", "details": "New idea
created."}
```

- **Response:** The created audit log object, confirming successful logging of the action.

## View Logic

Overview of viewsets and their methods:

- ♦ **UserViewSet:** Handles user-related actions such as profile retrieval and statistics.
  - **profile:** Retrieve user profile information.
  - **statistics:** Get user statistics.
  - **sync\_offline\_data:** Synchronize offline data.

```
class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    @action(detail=False, methods=['get'])
    def profile(self, request):
```

```

        user = request.user
        serializer = self.serializer_class(user)
        return Response(serializer.data)

    @action(detail=False, methods=['get'])
    def statistics(self, request):
        # Logic to retrieve user statistics
        pass

    @action(detail=True, methods=['post'])
    def sync_offline_data(self, request, pk=None):
        # Logic to synchronize offline data
        pass

```

- **IdeaViewSet**: Manages ideas, including creation, retrieval, and voting.
  - **my\_ideas**: Retrieve ideas created by the user.
  - **vote**: Vote on an idea.
  - **similar\_ideas**: Retrieve similar ideas.
  - **recommended\_collaborators**: Suggest collaborators for an idea.
  - **trending**: Get trending ideas.

```

class IdeaViewSet(viewsets.ModelViewSet):
    queryset = Idea.objects.all()
    serializer_class = IdeaSerializer

    @action(detail=False, methods=['get'])
    def my_ideas(self, request):
        user = request.user
        ideas = self.queryset.filter(creator=user)
        serializer = self.serializer_class(ideas,
        many=True) return Response(serializer.data)

    @action(detail=True, methods=['post'])
    def vote(self, request, pk=None):
        # Logic to handle voting on an idea
        pass

    @action(detail=True, methods=['get'])
    def similar_ideas(self, request, pk=None):
        # Logic to retrieve similar ideas
        pass

    @action(detail=True, methods=['get'])
    def recommended_collaborators(self, request, pk=None):
        # Logic to suggest collaborators
        pass

    @action(detail=False, methods=['get'])
    def trending(self, request):
        # Logic to get trending ideas
        pass

```



- ♦ **VoteViewSet**: Handles voting actions.

```
class VoteViewSet(viewsets.ModelViewSet):  
    queryset = Vote.objects.all()  
    serializer_class = VoteSerializer
```

- ♦ **TeamViewSet**: Manages teams and their members.
  - **add\_member**: Add a member to the team.
  - **remove\_member**: Remove a member from the team.

```
class TeamViewSet(viewsets.ModelViewSet):  
    queryset = Team.objects.all()  
    serializer_class = TeamSerializer  
  
    @action(detail=True, methods=['post'])  
    def add_member(self, request, pk=None):  
        # Logic to add a member to the team  
        pass  
  
    @action(detail=True, methods=['post'])  
    def remove_member(self, request, pk=None):  
        # Logic to remove a member from the team  
        pass
```

- ♦ **DocumentViewSet**: Handles document uploads related to ideas.

```
class DocumentViewSet(viewsets.ModelViewSet):  
    queryset = Document.objects.all()  
    serializer_class = DocumentSerializer
```

- ♦ **CommentViewSet**: Manages comments on ideas.

```
class CommentViewSet(viewsets.ModelViewSet):  
    queryset = Comment.objects.all()  
    serializer_class = CommentSerializer
```

- ♦ **NotificationViewSet**: Handles notifications for users.

```
class NotificationViewSet(viewsets.ModelViewSet):  
    queryset = Notification.objects.all()
```

```
serializer_class = NotificationSerializer
```

- ♦ **MilestoneViewSet**: Manages milestones related to ideas.

```
class MilestoneViewSet(viewsets.ModelViewSet):  
    queryset = Milestone.objects.all()  
    serializer_class = MilestoneSerializer
```

- ♦ **CategoryViewSet**: Handles categories for organizing ideas.

```
class CategoryViewSet(viewsets.ModelViewSet):  
    queryset = Category.objects.all()  
    serializer_class = CategorySerializer
```

- ♦ **AuditLogViewSet**: Manages audit logs for tracking actions.

```
class AuditLogViewSet(viewsets.ModelViewSet):  
    queryset = AuditLog.objects.all()  
    serializer_class = AuditLogSerializer
```

## Models

Overview of models:

- ♦ **User**: Represents application users with roles and expertise.

```
from django.contrib.auth.models import AbstractUser  
  
class User(AbstractUser):  
    role = models.CharField(max_length=50)  
    expertise = models.TextField()
```

- ♦ **Idea**: Represents ideas submitted by users, including fields for title, description, and status.

```
from django.db import models  
  
class Idea(models.Model):
```

```
title = models.CharField(max_length=255)
description = models.TextField()
status = models.CharField(max_length=50, choices=[('draft', 'Draft'),
('submitted', 'Submitted'), ('approved', 'Approved')])
creator = models.ForeignKey(User, on_delete=models.CASCADE)
created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)
```

- **Vote:** Represents votes cast by users on ideas, linking users and ideas.

```
class Vote(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    idea = models.ForeignKey(Idea, on_delete=models.CASCADE)
    value = models.IntegerField(choices=[(1, 'Upvote'), (-1, 'Downvote')])
    created_at = models.DateTimeField(auto_now_add=True)
```

- **Team:** Represents teams working on ideas, including team members and leaders.

```
class Team(models.Model):
    name = models.CharField(max_length=255)
    leader = models.ForeignKey(User, related_name='teams_lead',
on_delete=models.CASCADE)
    members = models.ManyToManyField(User, related_name='teams')
```

- **Document:** Represents documents related to ideas, including file uploads.

```
class Document(models.Model):
    title = models.CharField(max_length=255)
    file = models.FileField(upload_to='documents/')
    idea = models.ForeignKey(Idea, on_delete=models.CASCADE)
    uploaded_at = models.DateTimeField(auto_now_add=True)
```

- **Comment:** Represents comments on ideas, allowing for discussions.

```
class Comment(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    idea = models.ForeignKey(Idea, on_delete=models.CASCADE)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

- **Notification:** Represents notifications sent to users for various actions.

```
class Notification(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    is_read = models.BooleanField(default=False)
```

- **Milestone:** Represents milestones associated with ideas, tracking progress.

```
class Milestone(models.Model):
    idea = models.ForeignKey(Idea, on_delete=models.CASCADE)
    title = models.CharField(max_length=255)
    due_date = models.DateTimeField()
    created_at = models.DateTimeField(auto_now_add=True)
```

- **Category:** Represents categories for organizing ideas, allowing for better categorization.

```
class Category(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
```

- **AuditLog:** Represents logs of user actions for accountability and tracking.

```
class AuditLog(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    action = models.CharField(max_length=255)
    timestamp = models.DateTimeField(auto_now_add=True)
    details = models.TextField()
```

## Authentication

---

JWT authentication is implemented using `rest_framework_simplejwt`. Tokens are issued upon user login and can be refreshed using the refresh endpoint. This ensures secure access to the API.

## Testing

---

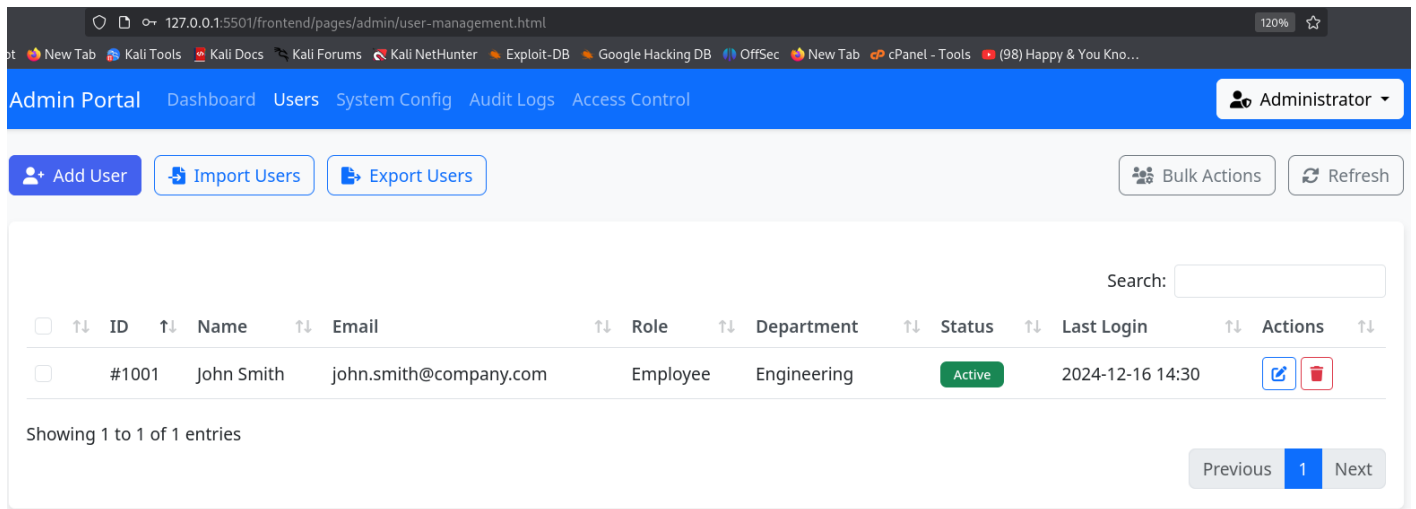
To run tests, use the following command:

```
python manage.py test
```

This will execute all test cases defined in the application, ensuring that all functionalities work as expected.

## User Interfaces

**File:** `frontend/pages/admin/user-management.html`

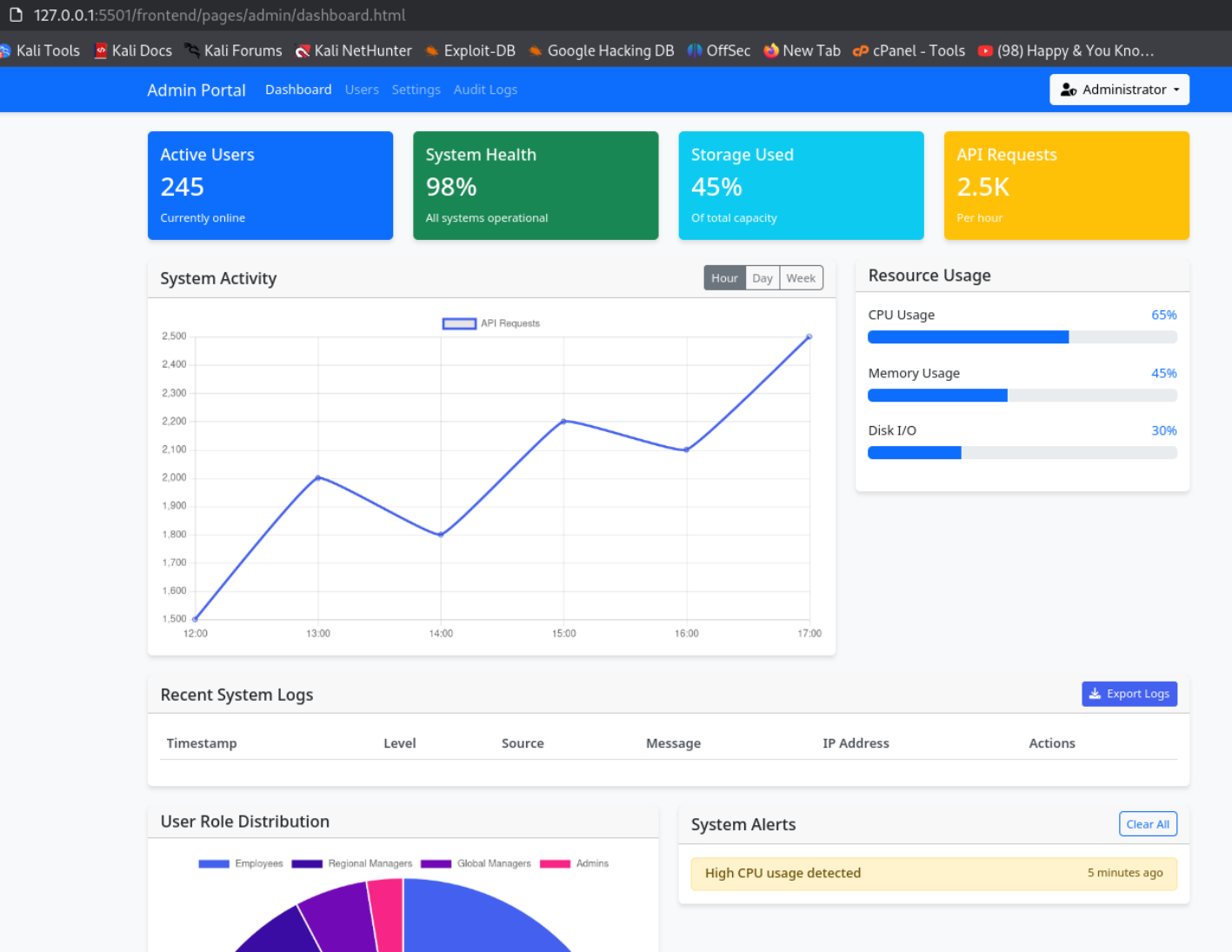


**Explanation:** This page is likely where administrators can manage user accounts, including adding, editing, or removing users. Screenshots here could show the user management interface, including forms for adding users or lists of existing users.

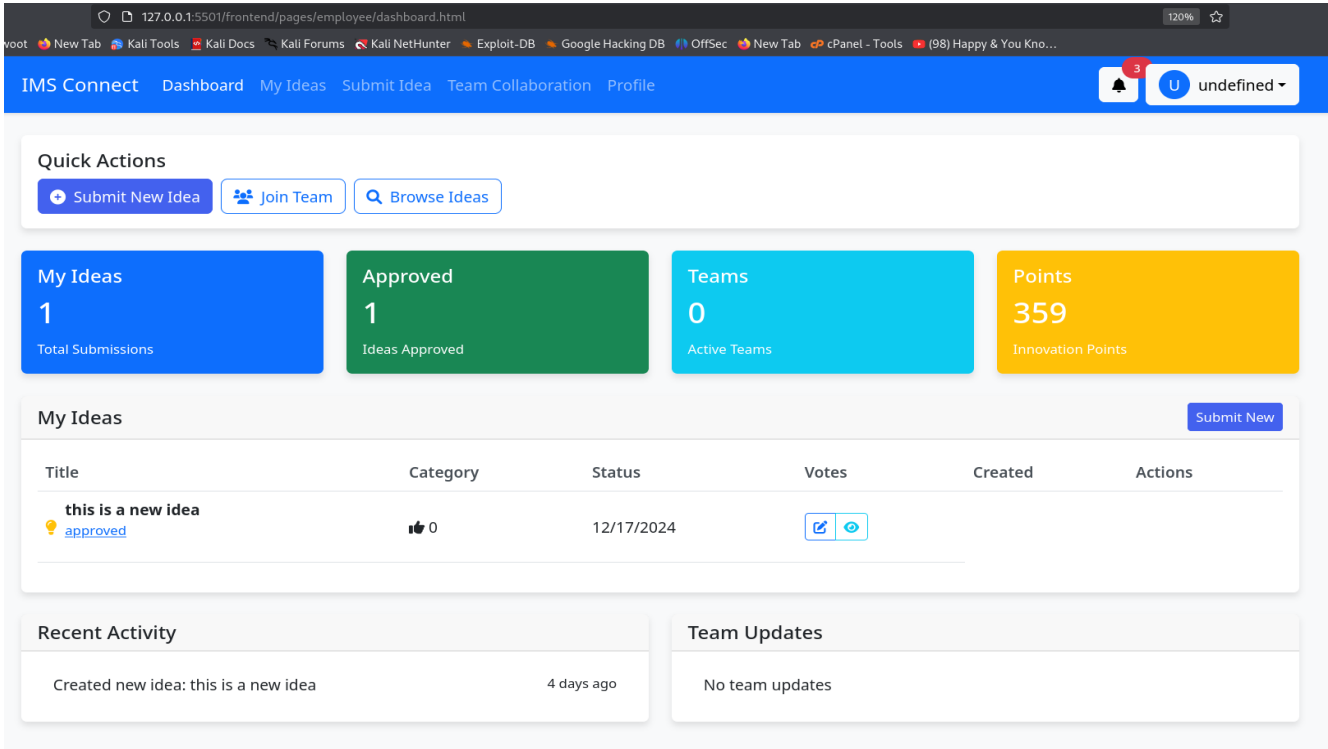
## Dashboard Pages

**Files:**

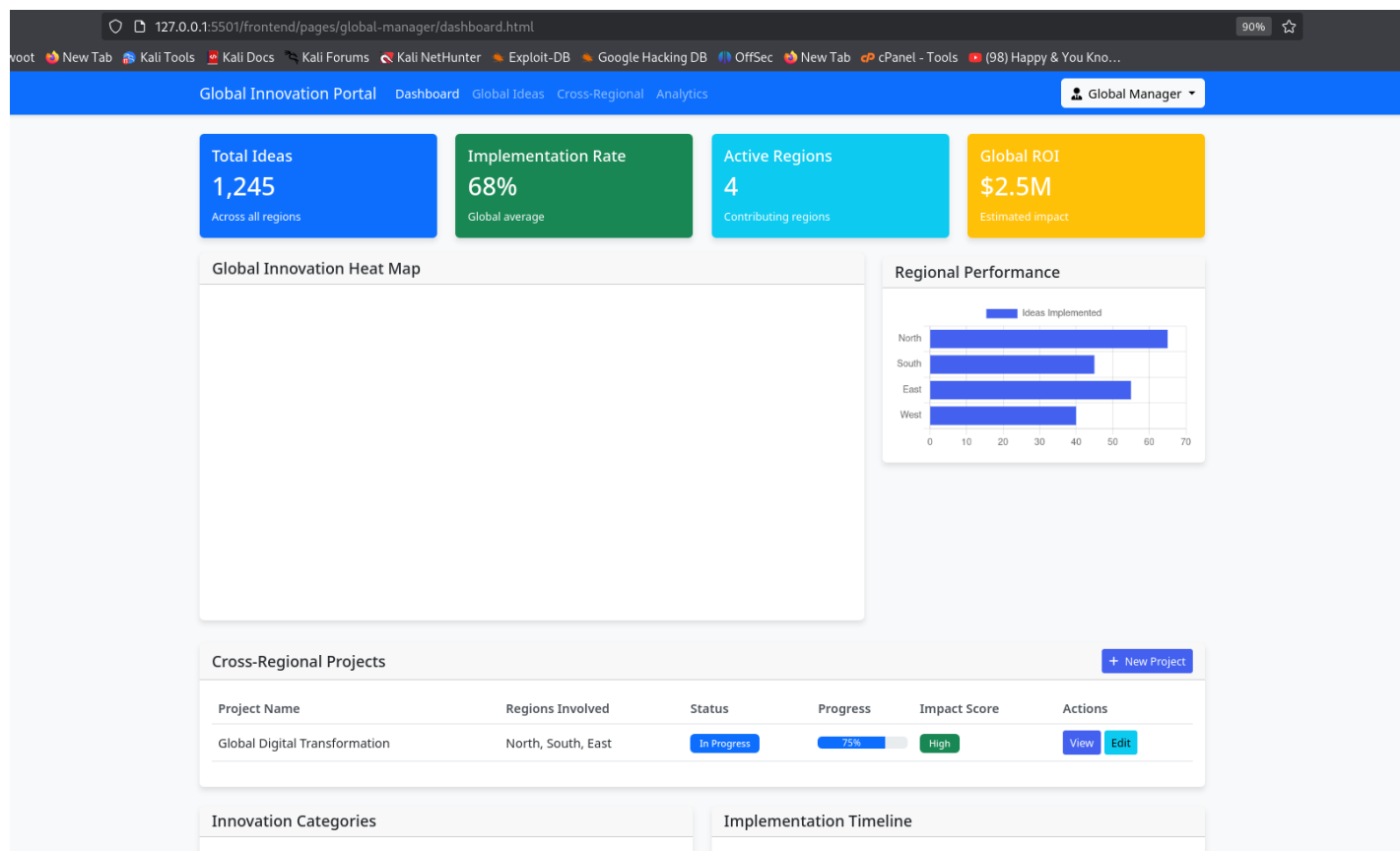
- `frontend/pages/admin/dashboard.html`



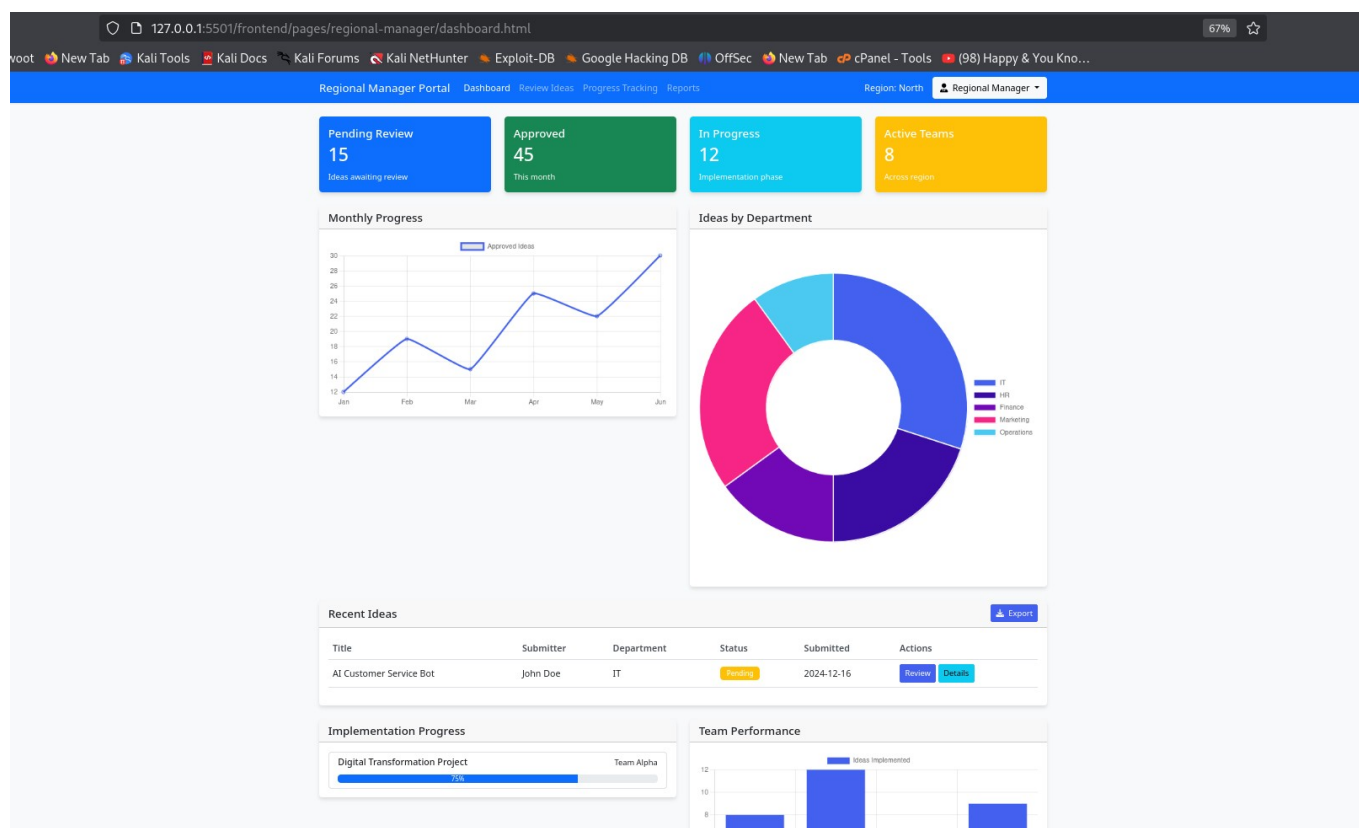
♦ frontend/pages/employee/dashboard.html



- frontend/pages/global-manager/dashboard.html



- frontend/pages/regional-manager/dashboard.html



**Explanation:** These dashboard pages provide an overview of key metrics and information relevant to different user roles. Screenshots could illustrate the dashboard

layout, key performance indicators, and any interactive elements.

# Idea Submission Page

File: frontend/pages/ideas/submit.html

127.0.0.1:5501/frontend/pages/ideas/submit.html90%☆

rootNew TabKali ToolsKali DocsKali ForumsKali NetHunterExploit-DBGoogle Hacking DBOffSecNew TabcPanel - Tools(98) Happy & You Kno...

IMS ConnectDashboardSubmit IdeaTeamsAnalyticsJohn Doe

Submit New Idea

Title






Enter a clear, concise title for your idea

Make it descriptive and engaging

Category

Select a category

Description

B I U     

Describe your idea in detail...

Provide detailed information about your idea

Tags

Add relevant tags

Add keywords to help others find your idea

Expected Impact

Select impact area

Select impact level

Implementation Timeline

Estimated timeline

Resource requirements

Attachments

Browse...

No files selected.

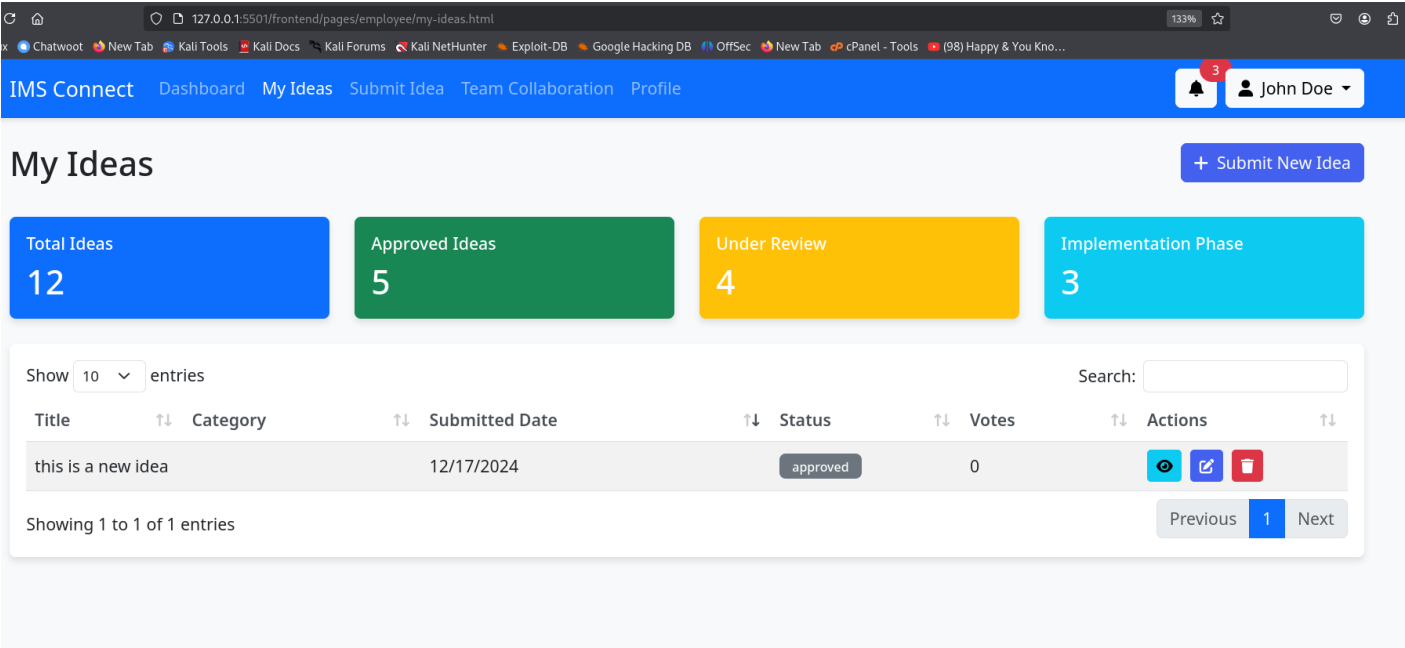
Upload relevant documents, images, or presentations (max 5 files, 10MB each)

**Explanation:** This page is where users can submit new ideas. Screenshots could show the submission form, including fields for title, description, and any other relevant inputs.



# My Ideas Page

File: frontend/pages/employee/my-ideas.html

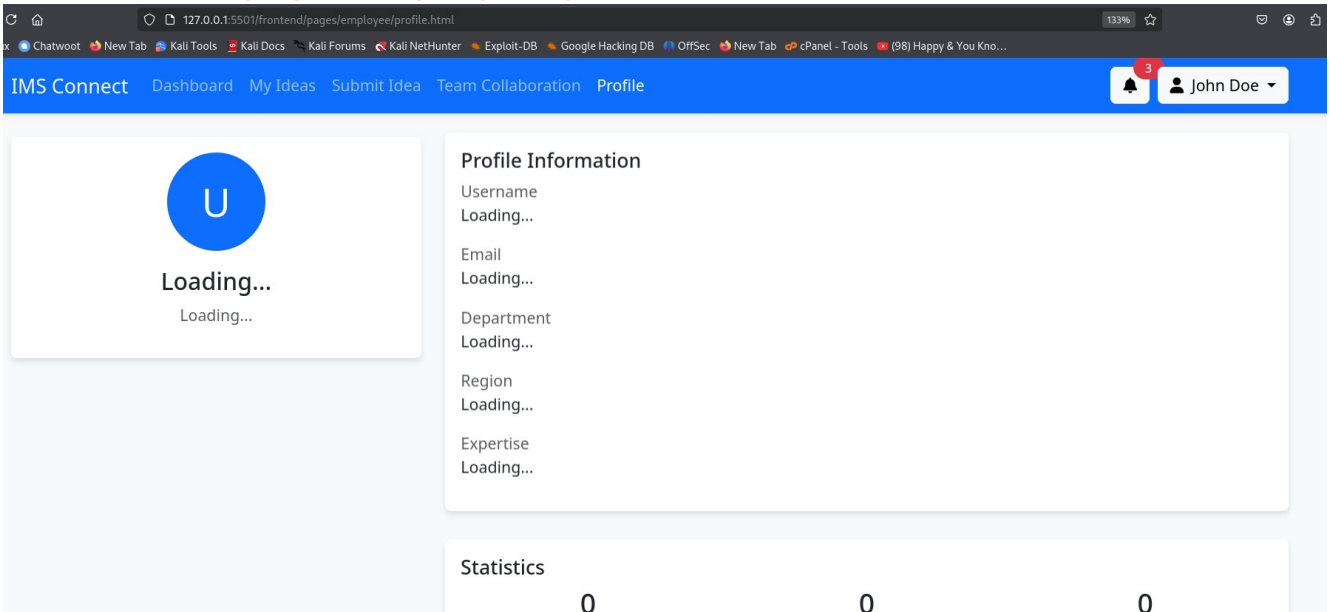


**Explanation:** This page likely displays the ideas submitted by the logged-in user. Screenshots could show the list of ideas, their statuses, and options to edit or delete them.

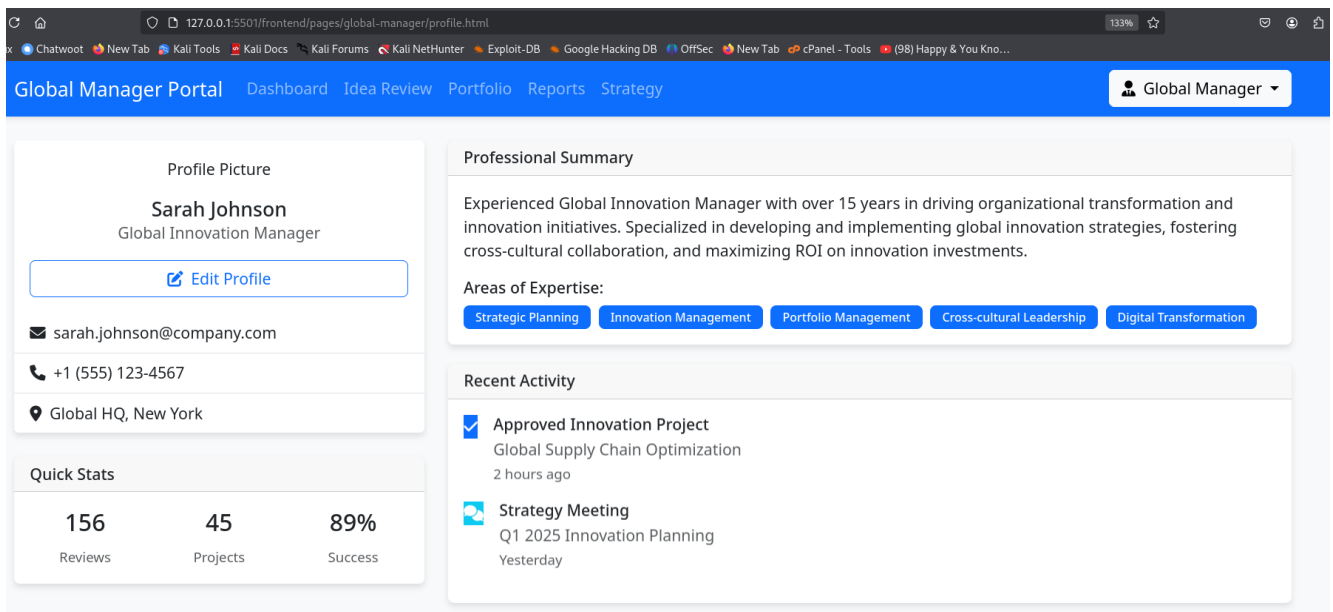
# Profile Page

Files:

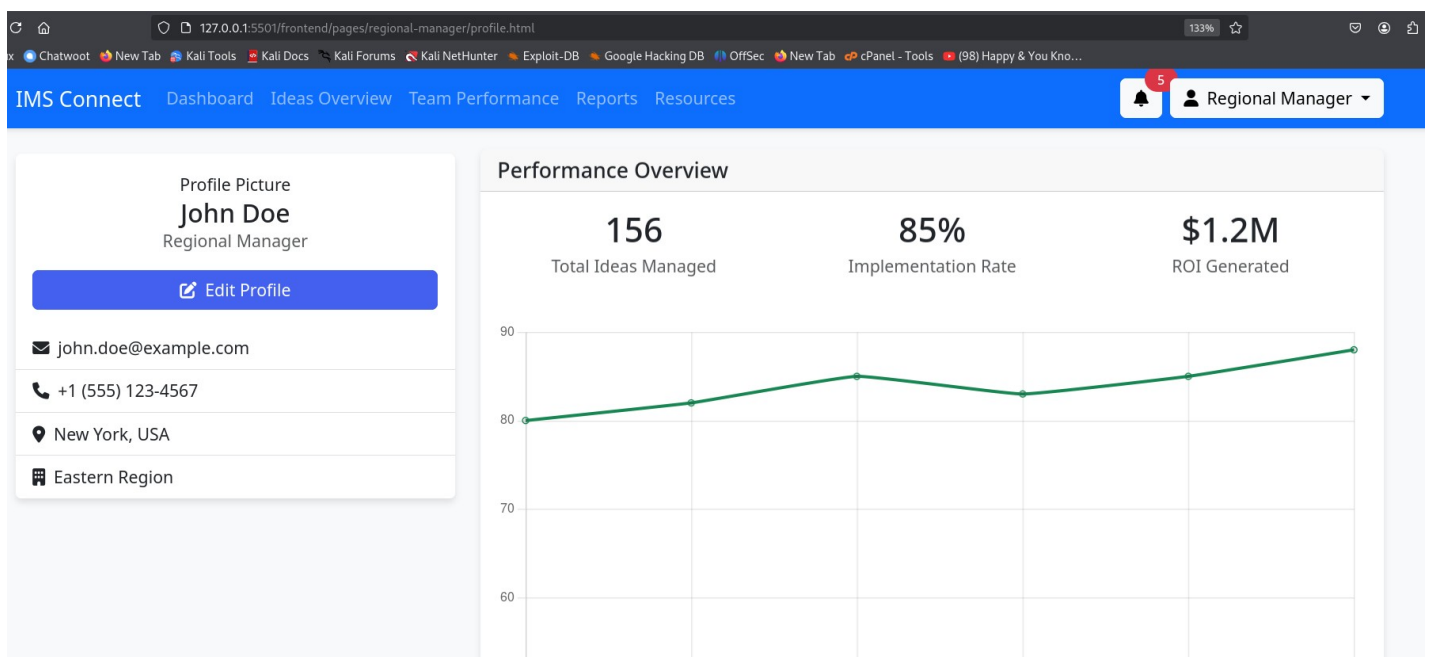
- frontend/pages/employee/profile.html



- frontend/pages/global-manager/profile.html



- frontend/pages/regional-manager/profile.html

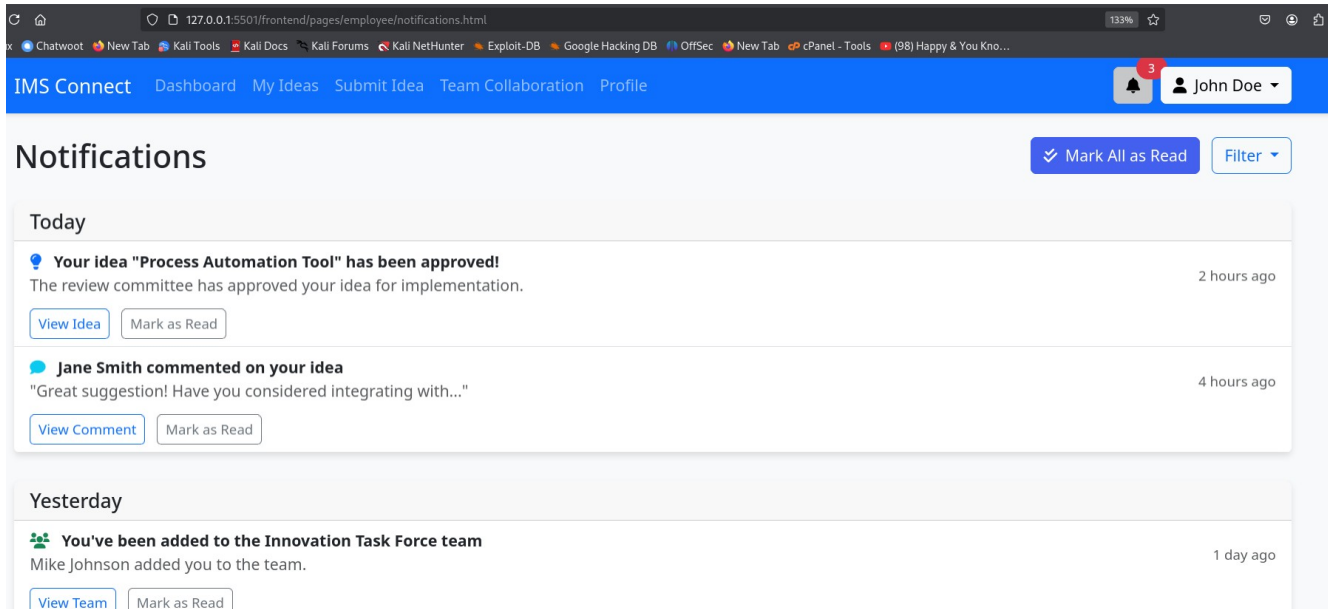


**Explanation:** These pages allow users to view and edit their profile information. Screenshots could show the profile details, including user roles and expertise.

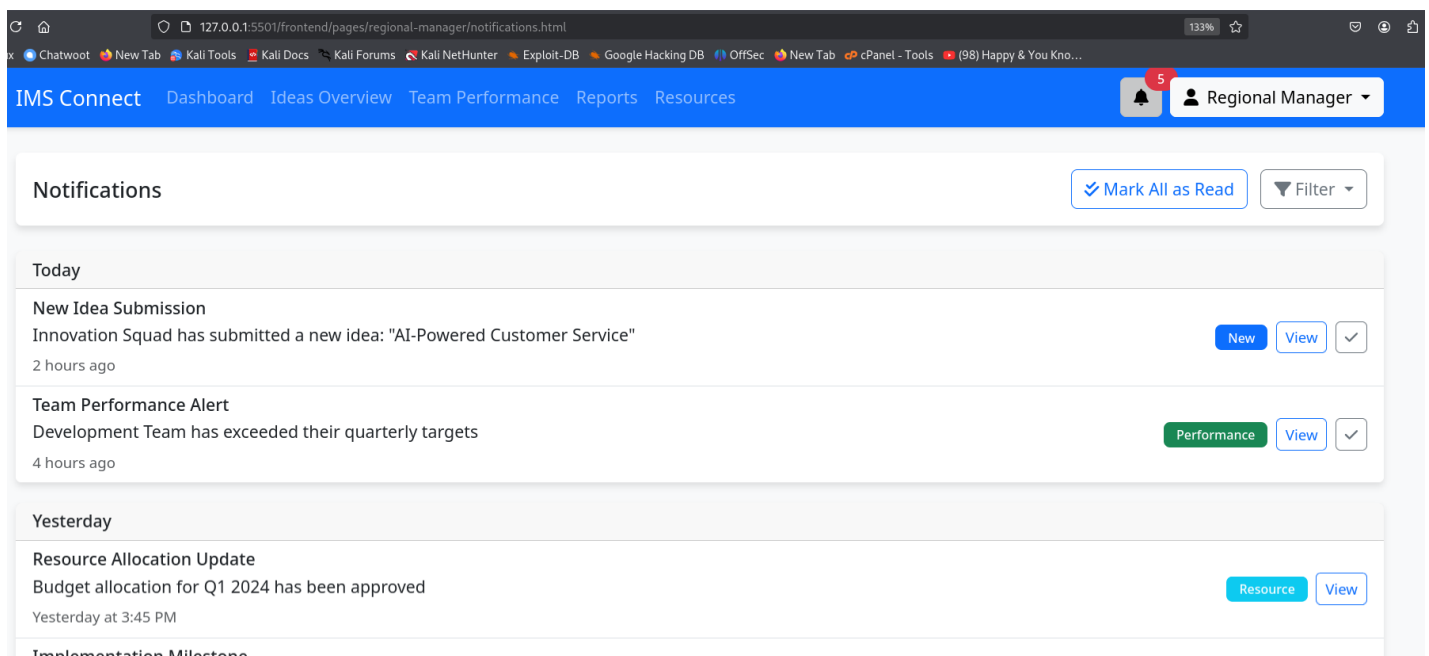
## Notifications Page

Files:

- frontend/pages/employee/notifications.html



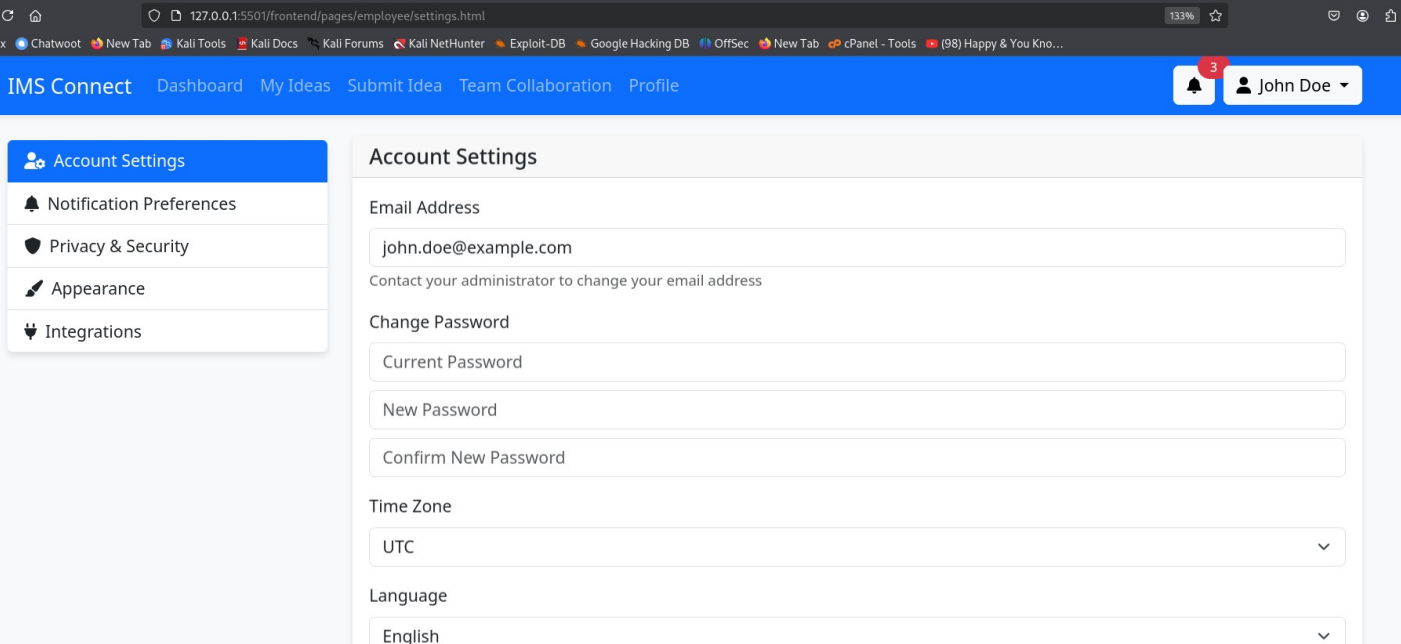
- `frontend/pages/regional-manager/notifications.html`



**Explanation:** These pages display notifications related to user activities. Screenshots could illustrate how notifications are presented and any actions users can take (e.g., marking as read).

## Settings Page

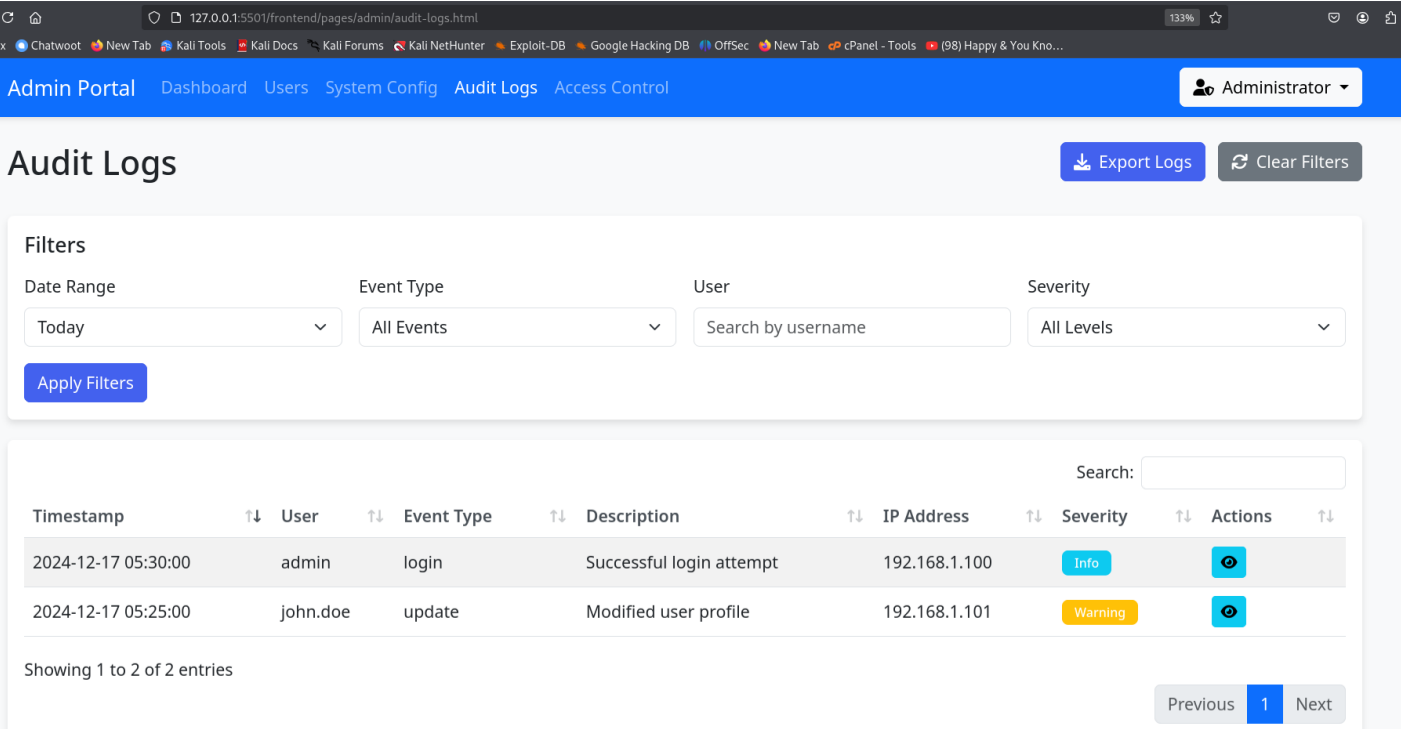
**File:** `frontend/pages/employee/settings.html`



**Explanation:** This page is where users can adjust their account settings. Screenshots could show the available settings options and any relevant forms.

## Audit Logs Page

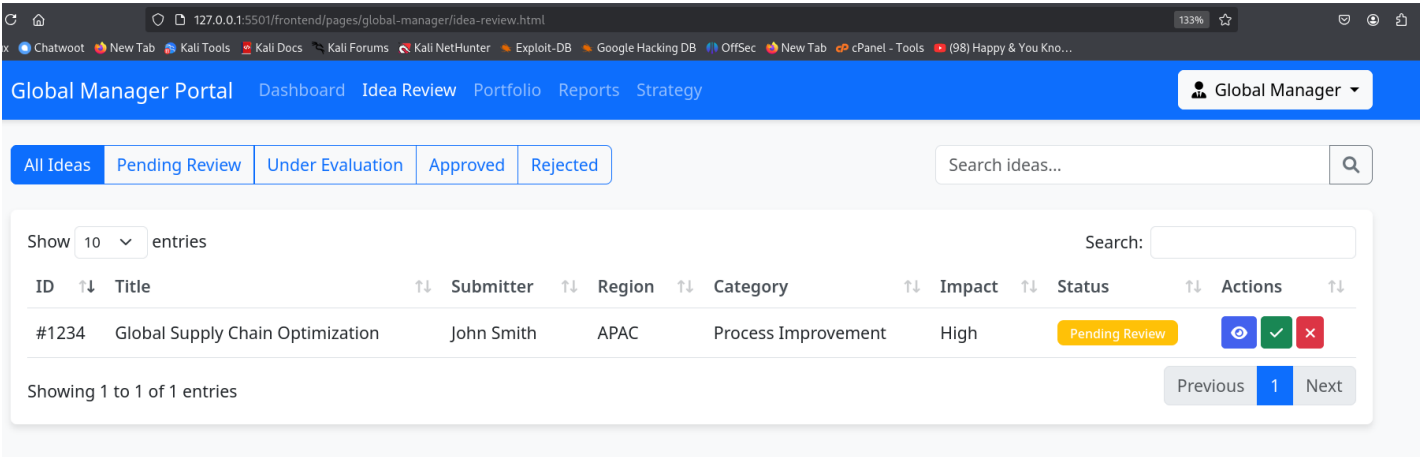
File: `frontend/pages/admin/audit-logs.html`



**Explanation:** This page likely displays logs of user actions for accountability. Screenshots could show the log entries and any filtering or searching capabilities.

## Idea Review Page

File: frontend/pages/global-manager/idea-review.html

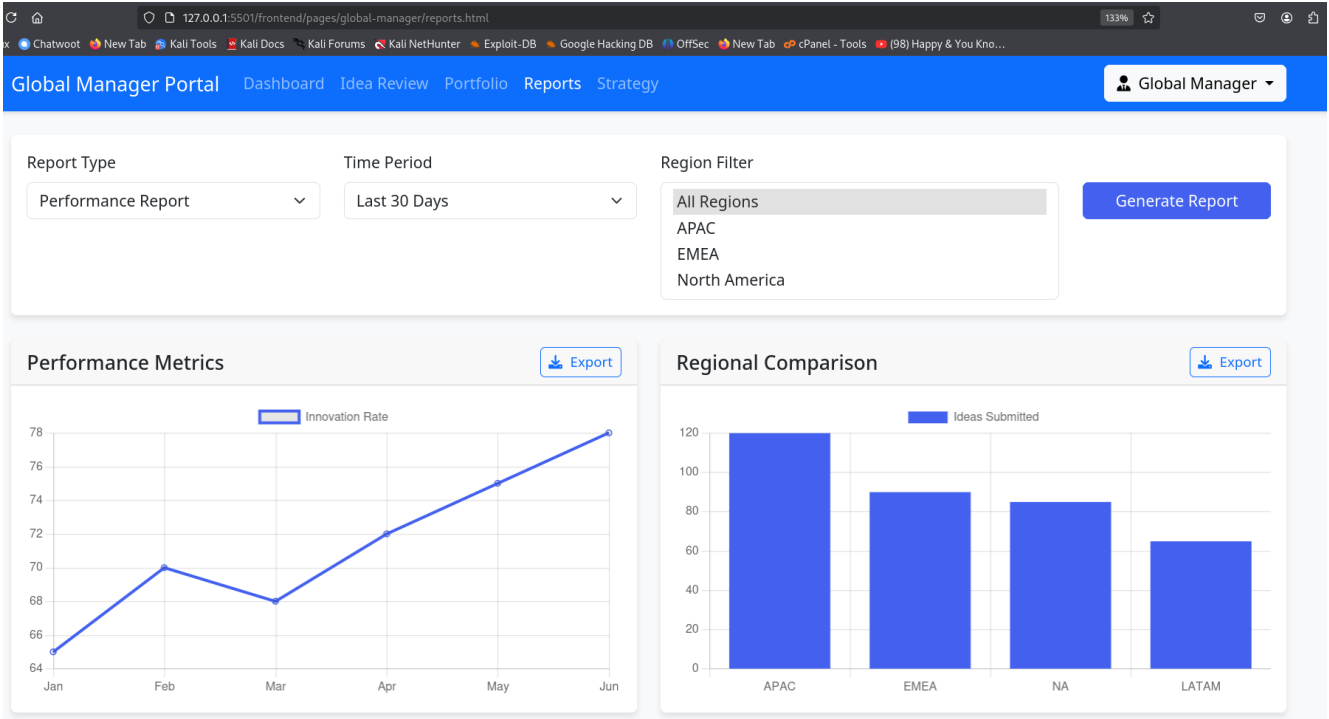


**Explanation:** This page is likely for reviewing submitted ideas. Screenshots could show the review interface, including options for approval or feedback.

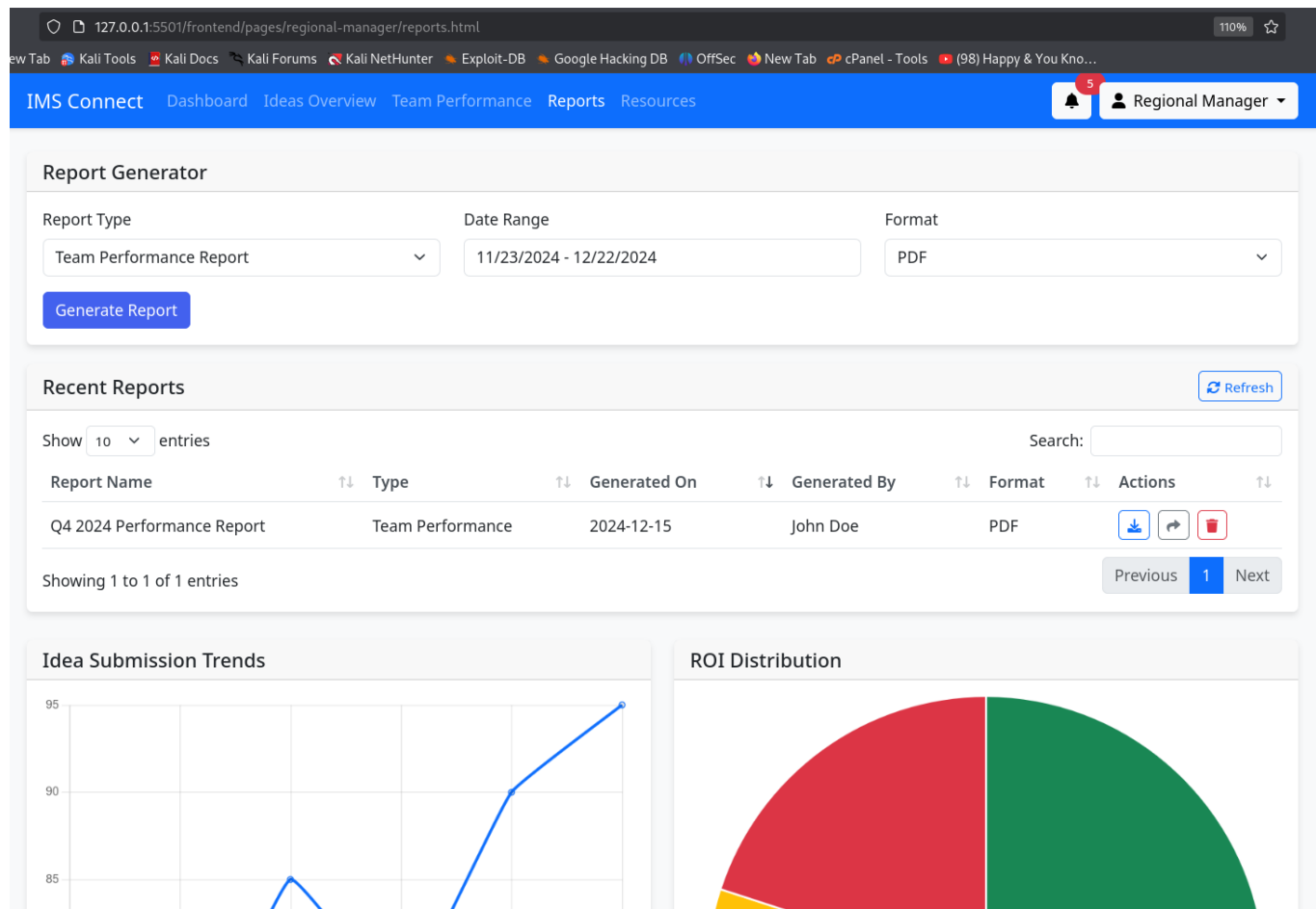
## Reports Page

Files:

- frontend/pages/global-manager/reports.html



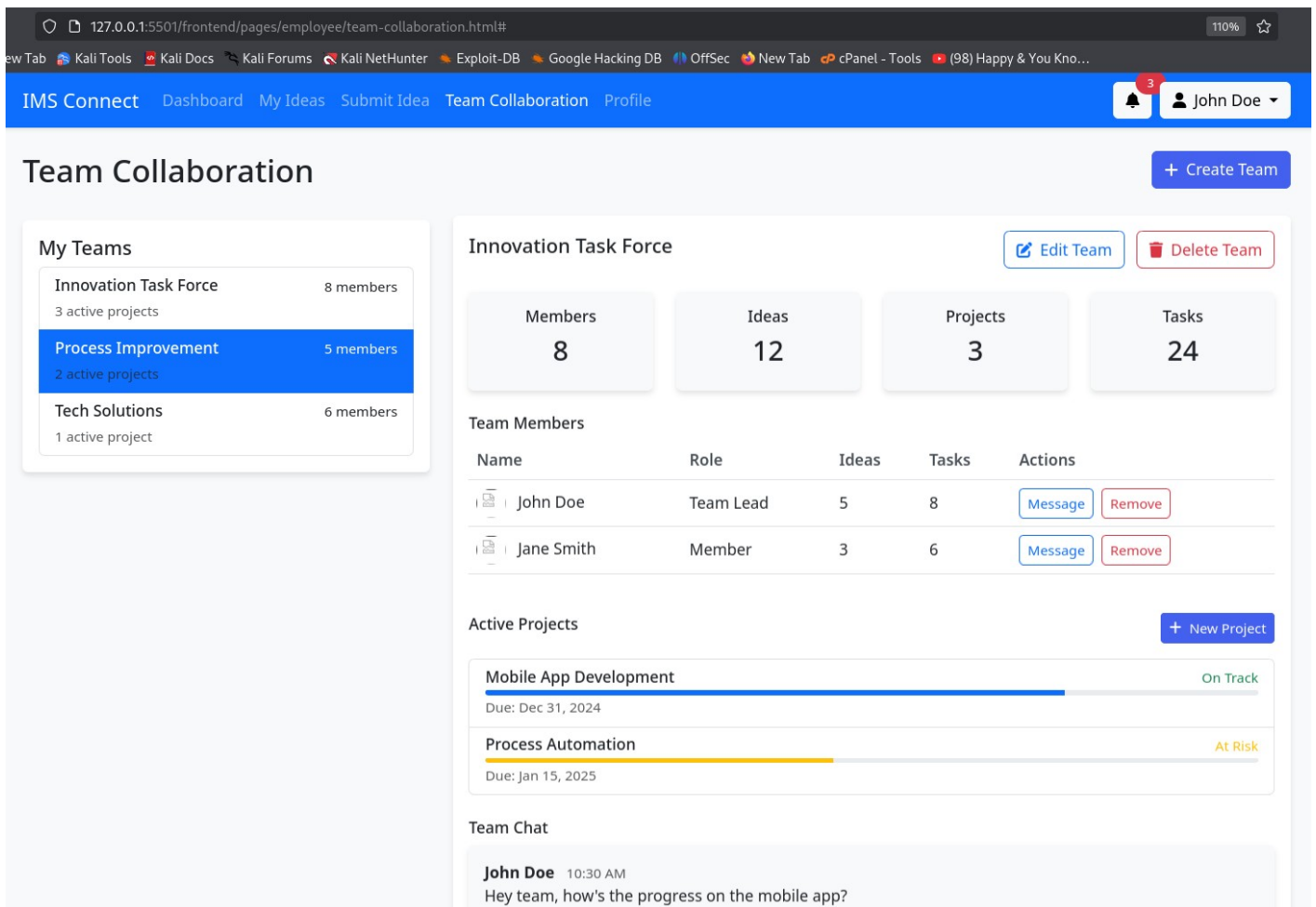
- frontend/pages/regional-manager/reports.html



**Explanation:** These pages likely provide detailed reports on various metrics. Screenshots could illustrate the types of reports available and how to access them.

## Team Collaboration Page

**File:** frontend/pages/employee/team-collaboration.html



**Explanation:** This page is where team members can collaborate on ideas. Screenshots could show collaboration tools, team member lists, and any shared documents.

## Home Page

# Innovate Together, Achieve More

IMS Connect empowers organizations to transform ideas into impact. Our platform streamlines innovation management, facilitates collaboration, and drives measurable results.

Innovation

Get Started

Learn More

500+  
Organizations

10k+  
Ideas Generated

95%  
Success Rate

FEATURES

## Everything You Need to Innovate

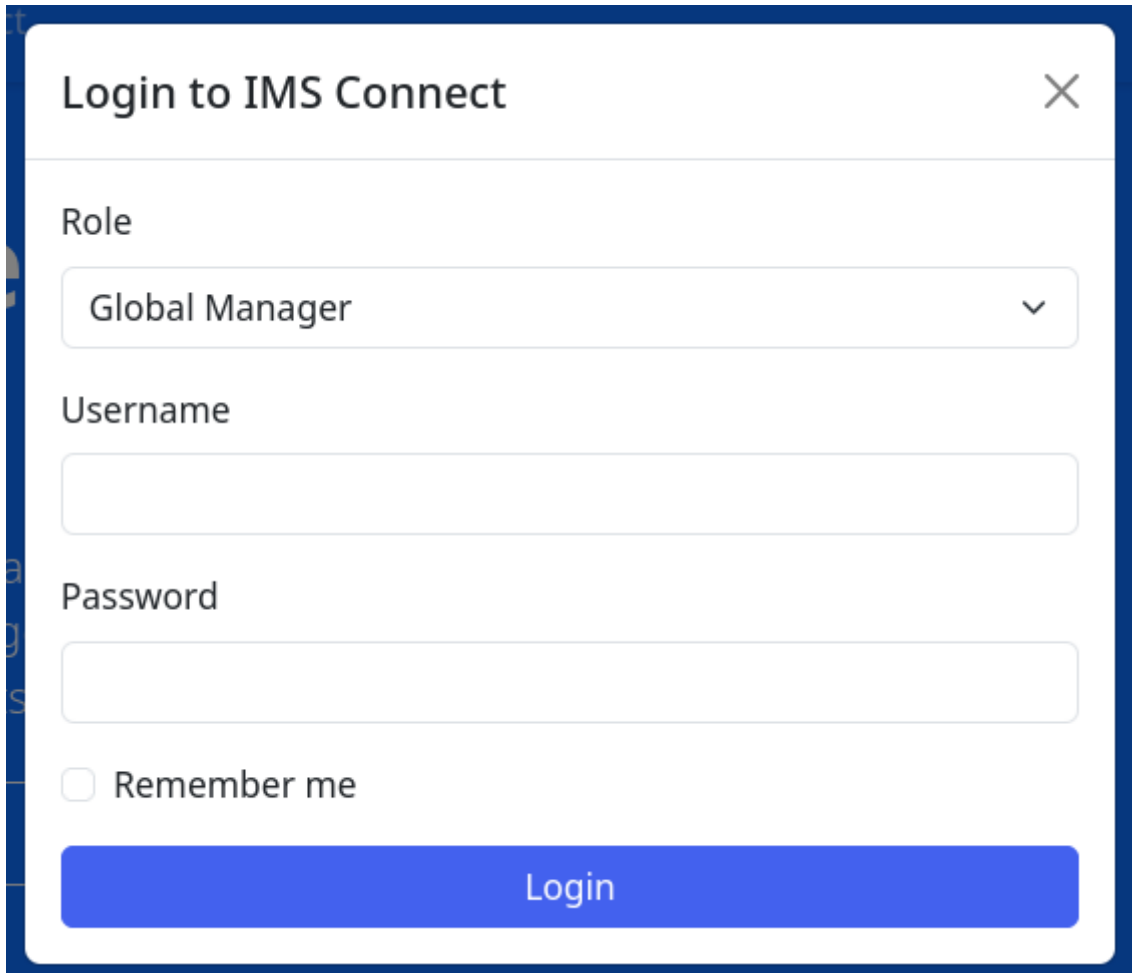
Our comprehensive platform provides all the tools necessary for successful innovation management.





This page is the entry point containing the login and registration forms for all the users and their respective redirects

## login page

A login form titled "Login to IMS Connect" with a close button (X) in the top right corner. The form contains a "Role" dropdown menu with "Global Manager" selected, a "Username" text input field, a "Password" text input field, a "Remember me" checkbox, and a blue "Login" button at the bottom.

Login to IMS Connect

Role

Global Manager

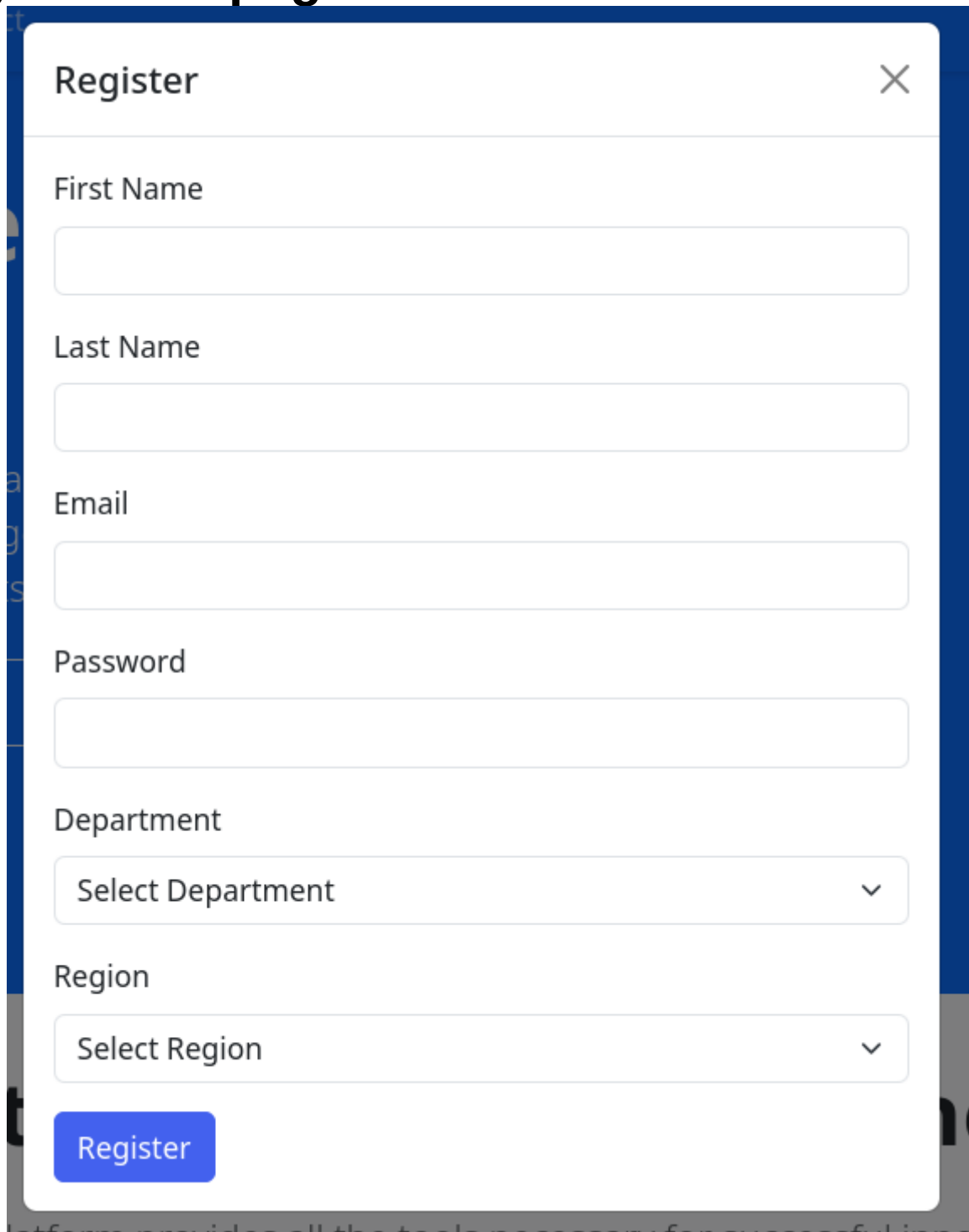
Username

Password

☐ Remember me

Login

## Registration page

A registration form modal with a white background and a blue border. It contains input fields for First Name, Last Name, Email, and Password. There are also dropdown menus for Department and Region. A blue Register button is at the bottom left, and a close button (X) is at the top right.

Register

First Name

Last Name

Email

Password

Department

Select Department

Region

Select Region

Register

## Role redirect section

# Choose Your Role

Access role-specific features and tools designed to meet your needs.



## Global Manager

Oversee global innovation initiatives and strategic decision-making.

Access Portal



## Regional Manager

Manage regional teams and coordinate innovation efforts.

Access Portal



## Employee

Submit ideas and collaborate with team members.

Access Portal



## Administrator

Manage system settings and user permissions.

Access Portal