Backend Views Structure – Django Application

This page explains how **Views** are structured and managed in the backend of the project. It includes the different view files created based on roles and common functionalities such as authentication and reusable actions.

Views File Structure

- Views are organized into separate files to maintain clarity and modularity:
  - `views.py`
  - `authentication_views.py`
  - `role_views/`
    - `agency_views.py`
    - `candidate_views.py`
    - `client_views.py`
    - `interviewer_views.py`
    - `recruiter_views.py`
    - `manager_views.py`
    - `celery_views.py`
  - `tasks.py`

Since the application supports multiple roles, views are grouped by **role-based responsibilities**. Developers can add or update views in any relevant file without needing to import each individually because all view modules are included in the `app/urls.py` .

Common Views

`app/views.py`

Contains basic and common views that can be accessed by two or more roles.

Modules included:

- Terms and Conditions management
- Blog module
- Ticket module for support

- View complete job posting
- View complete application
- Notifications module
- Invoice retrieval
- Pricing module (currently under development; future views should be added here or in a separate file)
- LinkedIn Integration for job and profile imports

`app/authentication_views.py`

Handles all **authentication-related views**:

- User login
- User signup
- Forgot password
- Token generation and refresh
- Change password
- Fetching user profile details

Role-Based Views

`app/role_views/agency_views.py`

Includes all views specifically designed for **agency-related operations**.
Detailed explanations are provided in code comments for each view.

`app/role_views/candidate_views.py`

Views related to **candidate functionality**, such as:

- Managing candidate profiles
- Viewing assigned job posts
- Submitting applications
- Updating interview availability
- Tracking recruitment status

`app/role_views/client_views.py`

Views designed for **client interactions**, including:

- Posting and managing job posts
- Reviewing applications
- Approving or rejecting candidate profiles
- Negotiating service terms
- Managing invoices and payments

`app/role_views/interviewer_views.py`

Handles interviewer-specific actions:

- Viewing and accepting interview invitations
- Managing interview schedules
- Submitting post-interview feedback and remarks
- Tracking assigned candidates for interviews

`app/role_views/recruiter_views.py`

Covers recruiter functionalities:

- Sourcing and recommending candidates
- Shortlisting profiles
- Coordinating with interviewers and managers
- Tracking assigned jobs and progress

`app/role_views/manager_views.py`

Manager-related views include:

- Approving job posts created by clients
- Managing recruiters and assigning job posts
- Reviewing negotiation requests
- Overseeing recruitment progress and reporting

app/role_views/celery_views.py

Defines all views related to **scheduled background tasks (Celery)**.
Tasks are static (not dynamically stored in the database).
Each view has a `process_*` prefix for clarity.

Types of views include:

- Sending invoices to clients after a candidate joins (delayed display based on service terms)
- Sending reminders:
  - Invoice payment reminders to clients
  - Approval reminders for job posts and negotiation requests to managers
  - Recruiter job assignment reminders
  - Candidate shortlist action reminders to clients
  - Interview scheduling reminders to recruiters
  - Post-interview feedback reminders to interviewers
  - Candidate selection reminders to clients
  - Job post acceptance reminders to candidates
  - Profile update reminders to candidates
  - Deadline reminders to organization managers and assigned recruiters
  - Joining status update reminders to clients

Celery Task Files

app/tasks.py

- Imports views from `celery_views`
- Registers and exposes them as **Celery shared tasks** for scheduling and execution.

RTMAS_BACKEND/celery.py

- Sets up Celery configuration.
- Defines and links all scheduled tasks.
- Specifies which views/tasks need to run periodically.

RTMAS_BACKEND/settings.py

- Includes the full Celery configuration.
- Defines:

    - Message broker setup (e.g., Redis/RabbitMQ)
    - Port connections
    - Task schedules and intervals
- Integrates Celery into the Django project settings.