**Developer Documentation for Custodian IAM Project**

Project By Mohamed Khalil Brik

Contact [Mohamedkhalil.brik@aucegypt.edu](mailto:Mohamedkhalil.brik@aucegypt.edu) for any assistance.

**Overview**

The **Custodian Project** is a Django web application that allows users to access employee data from various Excel sheets, particularly related to role-based access control (RBAC), HR data, and daily activities. The project is designed to present this information through a web dashboard.

**Prerequisites**

* Python 3.8+
* Django 5.1+
* Pandas 1.4+
* Excel files required for data

**Project Structure**

custodian/

├── custodian/

│ ├── \_\_init\_\_.py

│ ├── asgi.py

│ ├── settings.py

│ ├── urls.py

│ ├── wsgi.py

├── dashboard/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── models.py

│ ├── tests.py

│ ├── views.py

├── Resources/

│ ├── groups/

│ │ ├── Cards\_Application\_groups.xlsx

│ │ ├── Core\_Banking\_Application\_groups.xlsx

│ │ ├── Signature\_groups.xlsx

│ ├── Daily Sheet Sample.xlsx

│ ├── HR Sheet - Dummy Data.xlsx

│ ├── PATHS.xlsx

│ ├── RBAC Sample.xlsx

├── static/

│ ├── dashboard/

│ │ ├── js/

│ │ ├── script.js

├── templates/

│ ├── dashboard/

│ ├── index.html

├── venv/ # Virtual environment folder

├── db.sqlite3 # SQLite database file

└── manage.py

**File Descriptions**

* urls.py: Contains URL routing for the application, mapping the root URL (/) to the handler view and /groups/ to the groups\_handler.
* views.py: Implements the main logic for handling requests and serving dynamic data to the dashboard.

**Data Files**

* **RBAC Sample.xlsx**: Stores RBAC data for employees, mapping functional titles to applications they can access.
* **HR Sheet - Dummy Data.xlsx**: Stores HR data with employee information such as ID, name, location, manager, and access details.
* **Daily Sheet Sample.xlsx**: Contains records of daily activities for employees, including recent actions.

**Setup Instructions**

1. **Install Dependencies** Ensure you have installed the required packages:
2. **Configure the Database** You will be using Excel sheets for data storage. Ensure the Excel files (RBAC Sample.xlsx, HR Sheet - Dummy Data.xlsx, Daily Sheet Sample.xlsx) are correctly placed in the Resources/ folder.
3. **Run Migrations** Since this project doesn’t use a database, migrations are not necessary. Skip this step unless you add models.
4. **Run the Server** Start the Django development server:

* python manage.py runserver
* Visit http://127.0.0.1:8000/ to view the dashboard.

**Function Descriptions**

**handler(request)**

* **Description**: The main view handler retrieves the employee's information based on the query passed through the GET request. It fetches data from HR Sheet - Dummy Data.xlsx and enriches the data with RBAC and recent activities.
* **Input**: The function expects a query parameter q containing the employee's ID.
* **Output**: Renders dashboard/index.html with the following context:
  + id: Employee's ID
  + name: Employee's Name
  + state: Employee's status
  + title: Functional Title
  + sector, division, location: Employee's work details
  + manager, managerID: Information about the line manager
  + apps: Applications accessible by the employee
  + RBAC: Access permissions for various applications based on RBAC
  + recent: List of recent activities for the employee

**groups\_handler(request)**

* **Description**: Fetches and displays the employee's groups based on the selected application. It reads data from the Excel file {app\_name}\_groups.xlsx.
* **Input**: The function expects a query parameter app\_name for the application name.
* **Output**: Renders dashboard/index.html with groups containing the employee’s group memberships.

**get\_application(df)**

* **Description**: Extracts the list of applications an employee has access to from the last row of their HR data.
* **Input**: DataFrame row containing the employee's HR data.
* **Output**: List of accessible applications.

**RBAC\_apps(title)**

* **Description**: Retrieves the RBAC information for the employee based on their functional title.
* **Input**: Employee's functional title.
* **Output**: Dictionary containing the employee’s RBAC access permissions.

**daily\_data(id)**

* **Description**: Fetches the most recent activities of an employee from Daily Sheet Sample.xlsx.
* **Input**: Employee's ID.
* **Output**: Dictionary containing the top 10 recent activities.

**URL Routing**

* /: Root endpoint, which triggers the handler view. Requires a query parameter q for the employee ID to return data.
* /groups/: Fetches the employee’s group memberships based on the application name provided in the query string as app\_name.

**Dependencies**

* **Django**: Used for creating views, handling requests, and rendering HTML templates.
* **Pandas**: Used for reading Excel files and processing the data.

**Notes**

* **Global State**: The project relies on a global Recent\_Context dictionary to cache the most recent context data. Ensure that you consider potential race conditions or concurrent request issues if extending this project further.
* **File Paths**: Paths to Excel files are hardcoded. Consider using Django settings for environment-specific configurations to make paths dynamic.

**Important Notes:**

* The files inside the resources -> groups folder should have the name of the apps same as the HR sheet names.

**Improvements for Future Versions**

1. **Database Integration**: Currently, the project relies heavily on Excel files for storing data. Integrating with a database would improve performance and scalability.
2. **Error Handling**: Improve error handling to deal with invalid query inputs, missing files, or unexpected data in the Excel sheets.
3. **Environment Configuration**: Use environment variables or Django settings for managing paths to resources and other sensitive configuration data.
4. **Unit Testing**: Add unit tests to ensure all views and helper functions work as expected.