Company Count Application

Overview

Your task is to create a Web application using Django. The application will allow users to login and filter the database table using a form.

Once the user submits the form, display the count of records based on the applied filters.

Technical Requirements

1. **Project Setup**
   - **Framework:** The application should be built using the Django framework to leverage its performance and modern features. Name the application as **catalyst-count**.
   - **Programming Language:** Python 3.x
   - **Virtual Environment:** Use virtual environments to manage Python packages.

2. **Database Configuration**
   - **Database:** PostgreSQL. It should store user data and the uploaded CSV data.
   - **ORM:** Utilize Django ORM to facilitate database interactions.
   - **Company Data Model:** Download the test data set from here. Create a model schema (table) by as per the data set. Import this csv into a Postgres table.

3. **Environment and Repository**
   - **Environment Variables:** Securely manage environment variables using **django-environ**.
   - **Version Control:** Initialize a Git repository to maintain the project versioning. Host the project on GitHub or Bitbucket.

4. **Authentication**
   - **User Authentication:** Implement user session authentication using **django-all-auth**.

5. **File Upload and Processing**
   - **File Upload:** Implement a file upload mechanism capable of handling large files (up to 1GB) with a visual progress indicator.
   - **Background Processing:** Upload file and update the database asynchronously to prevent blocking the request-response cycle.
   - **Data Model:** Design and implement models to store & retrieve the CSV data efficiently in PostgreSQL.

6. **User Interface**

- **Template Engine:** Utilize Django template engine for front-end design to ensure a responsive and intuitive user interface. Feel free to use Bootstrap 4 to create your own UI.
- **Pages:** The application should include the following pages:
  1. **Login Page:** For user authentication.
  2. **Upload Data Page:** Allows users to upload CSV files.
  3. **Query Builder Page:** Enables users to filter the uploaded data and query & view the count of records matching the filters.
  4. **DRF API**: Querying should be done using an API. Use **DRF** to create this API.
  5. **User Management Page (Optional):** For viewing and managing user accounts.

7. **Documentation**
   - **README.md:** Include a README file in the repository with detailed setup instructions, environment configuration steps, and how to run tests.

8. **Testing (Optional)**
   - **Unit Testing:** Write unit tests for all functionalities, focusing on the file upload process, database operations, and query builder logic.

9. **Containerization using Docker (Optional)**
   - **DockerFile**: Package your application and all its dependencies together in the form of container.

Functional Requirements

10. **User Authentication**
    - Users must be able to register, log in, and log out.
    - Only authenticated users must be able to interact with application features.

11. **Data Upload and Management**
    - Users should be able to upload CSV files up to 1GB in size.
    - The system must provide feedback on the upload progress.
    - Once uploaded, the system processes the file in the background and updates the database with the new data.

12. **Data Interaction**
    - Users should be able to apply filters to the data through a query builder interface.
    - The application displays the count of records that match the applied filters.

13. **User Interface**
    - The interface should be user-friendly, responsive, and accessible on various devices and screen sizes.
    - Ensure secure and accessible forms for data entry and authentication.

Non-functional Requirements

14. **Performance**
    - The application should handle large file uploads and data processing efficiently without significant delays.
    - Optimize database queries to handle large datasets effectively.

15. **Scalability**
    - The system should be designed to accommodate an increasing amount of data and users.

16. **Security**
    - Adhere to best practices for web security to protect sensitive data and prevent common vulnerabilities.

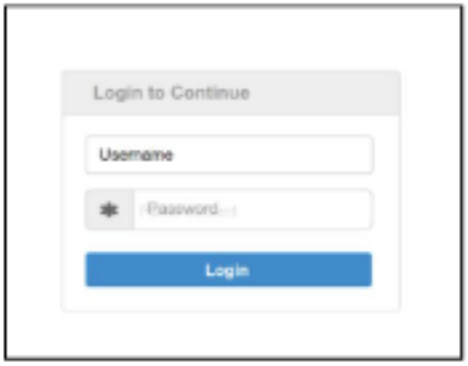
Deliverables

1. Source code hosted on a public Git repository on GitHub.
2. A functional web application ready to be deployed to a development server.
3. Documentation including setup instructions, environment variable configuration, and test execution details.
4. A suite of unit tests covering critical functionalities (Optional)

Sample UIs

The following are the approximate reference UIs for the task.

Application Login



Query Builder

Upload Data

Users