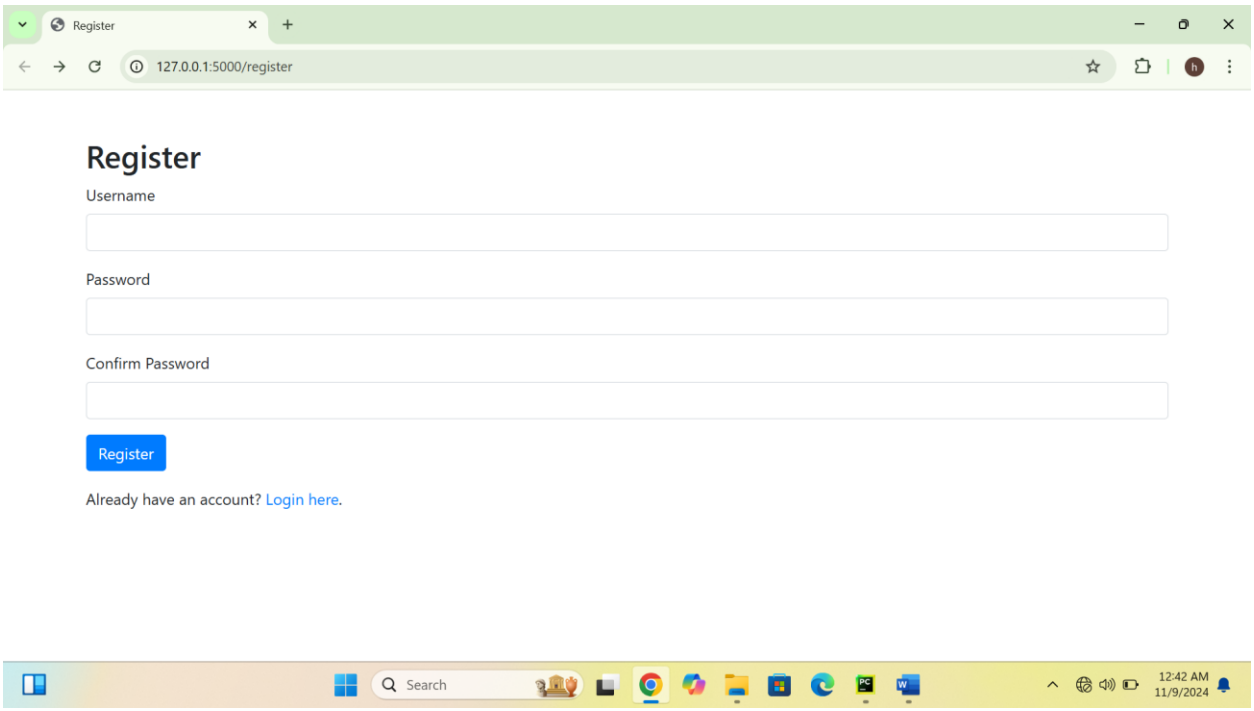
# Energy Usage and Carbon Footprint Tracker App Documentation

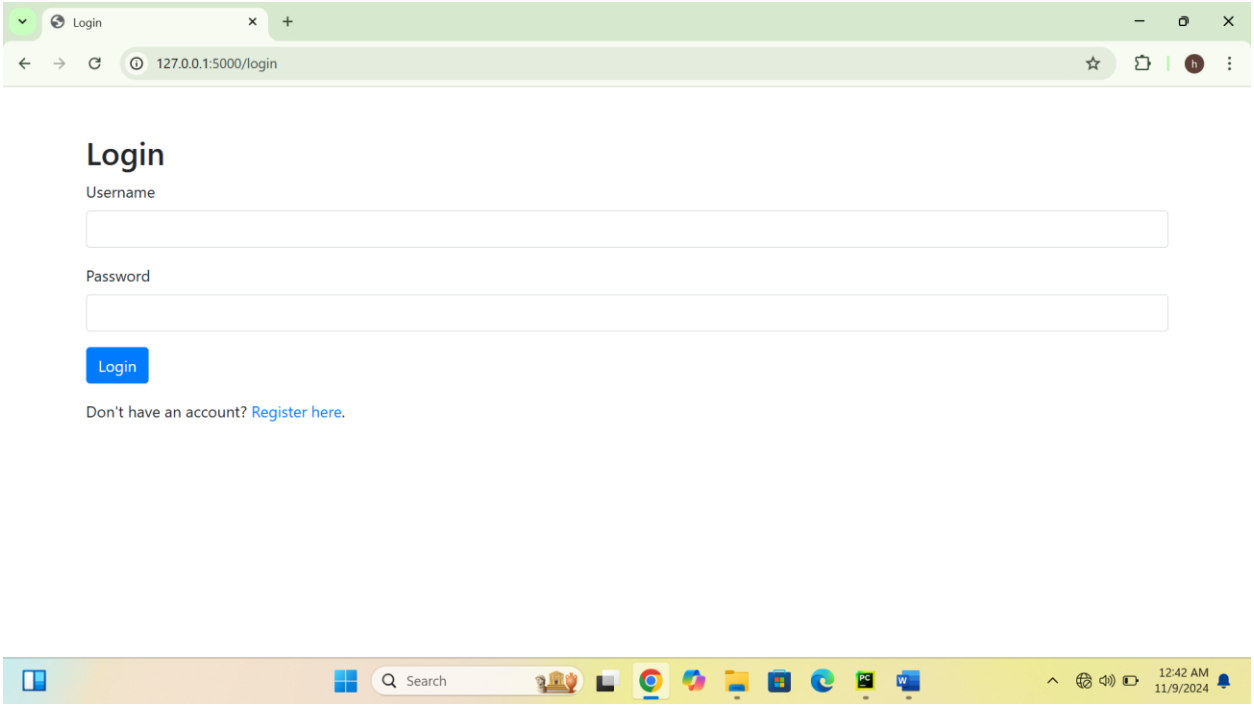
## Overview

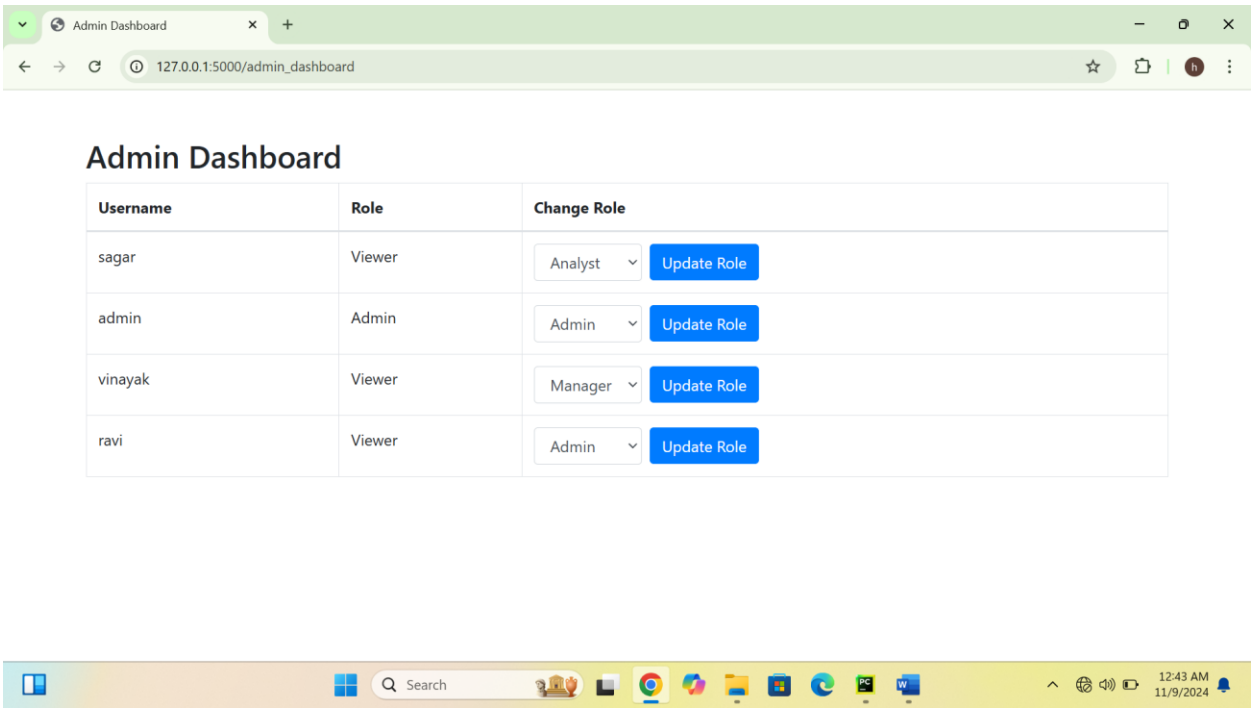
The **Energy Usage and Carbon Footprint Tracker** is a Flask-based application designed for monitoring energy consumption and carbon emissions. The app provides user authentication, role-based access, dynamic filtering, and database migration capabilities to facilitate energy data tracking and visualization.

## Key Features

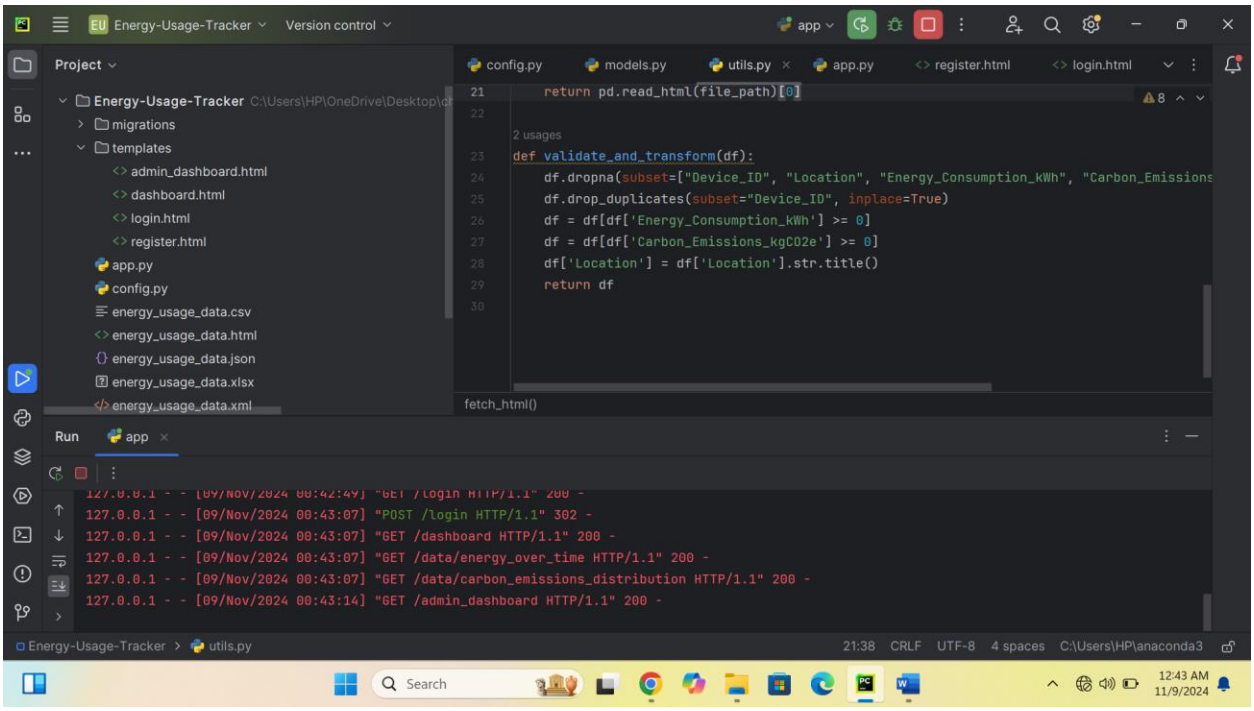
- **Role-Based Access Control**: Admins can manage user roles, with each role having different dashboard views and access levels.



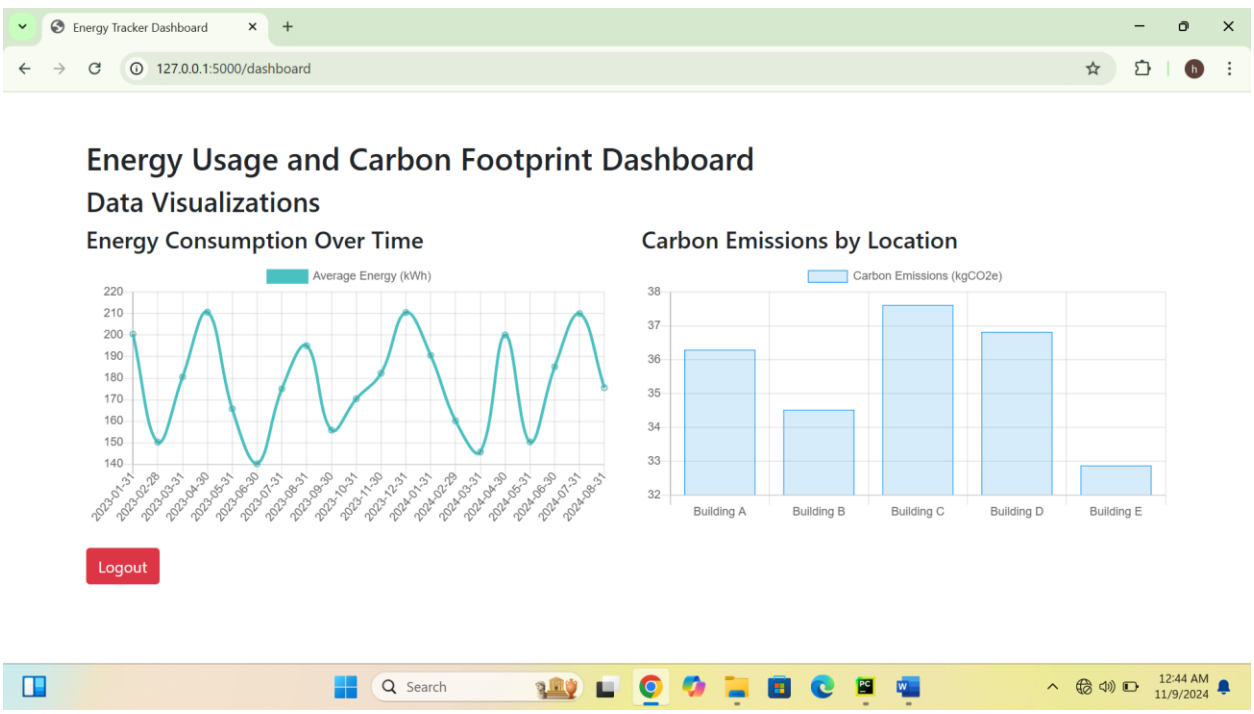


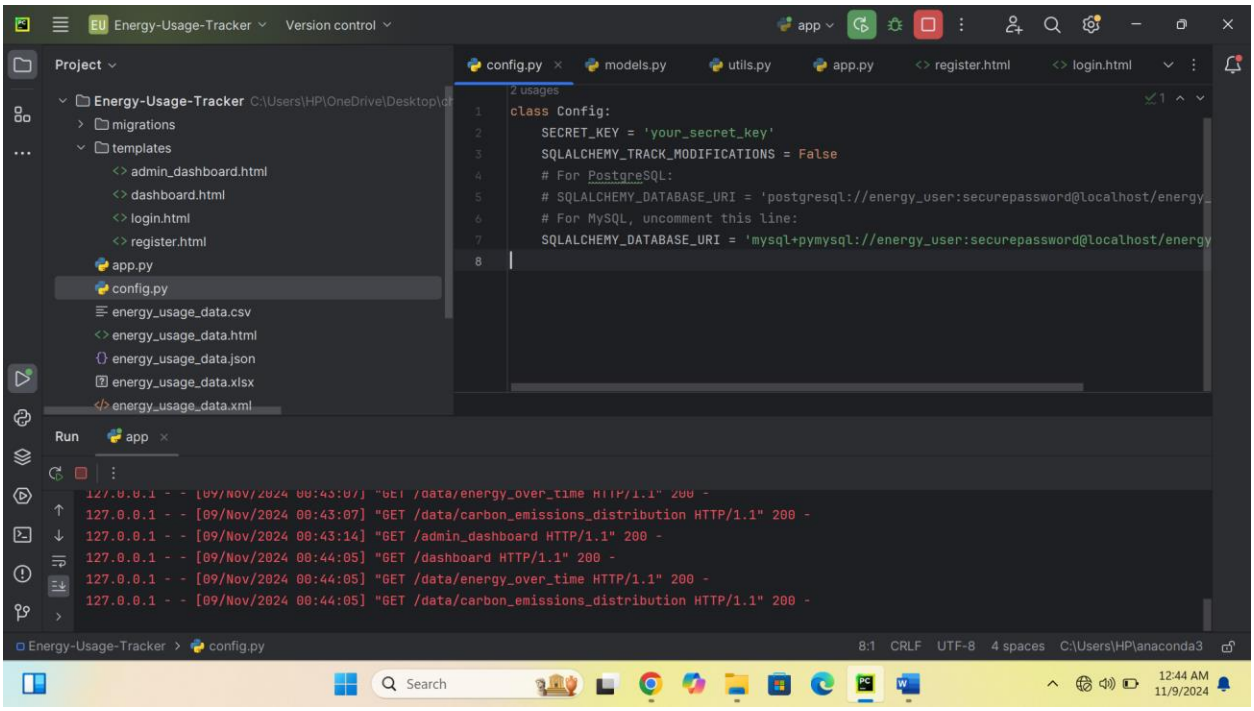

- **Data Fetching and Validation**: Supports data integration from multiple sources (JSON, XML, CSV, Excel, HTML) with validation and transformation before storing in the database.

- 

- 
- **Data Visualization**: Displays interactive charts for energy usage over time, emissions distribution, energy by device type, and more.




- **Database Migration**: Database schema changes are easily managed with Flask-Migrate for seamless updates and expansion.

- 


---

**Database Migrations**

This app uses **Flask-Migrate** (powered by Alembic) to manage database schema changes, making it easy to add, modify, or remove columns and tables without data disruption.

**Typical Migration Workflow:**

1. **Update Models**: Modify models in models.py for new columns or relationships.
2. **Create a New Migration**:

bash

Copy code

flask db migrate -m "Your migration message"

3. **Apply the Migration**:

bash

Copy code

flask db upgrade

4. **Rollback (if necessary)**:

bash

Copy code

flask db downgrade

**Switching Databases (PostgreSQL and MySQL):**

1. Update SQLALCHEMY_DATABASE_URI in config.py to the desired database URI.
2. Run flask db upgrade to apply migrations to the new database.

---

**Key Components**

**User Roles**

The app includes the following roles:

1. **Admin**: Full access, including user management.
2. **Manager**: Access to all visualizations.
3. **Analyst**: Limited access to specific charts and filters.
4. **Viewer**: Basic access to view visualizations only.

**Data Fetching, Validation, and Transformation**

The app fetches data from multiple file formats (JSON, XML, CSV, HTML, Excel) and validates it. Key validations include:

- Checking for required fields.
- Filtering data within specified value ranges.
- Formatting fields for consistency. These processes are defined in utils.py.

**Data Visualization**

Charts are rendered using **Chart.js** in dashboard.html. Data is fetched from API routes in the Flask backend. Available charts include:

- **Energy Usage Over Time** (Line Chart)
- **Carbon Emissions Distribution by Location** (Bar Chart)
- **Energy Consumption by Device** (Pie Chart)
- **Carbon Emissions Over Time** (Line Chart)

- **Energy Consumption vs. Carbon Emissions** (Scatter Plot)

---

**Routes and Functionality**

- **Authentication Routes**:
  - /register: Register new users.
  - /login: Login page for existing users.
  - /logout: Logs out the user.
- **Dashboard Routes**:
  - /dashboard: Displays role-based charts and filters.
  - /admin_dashboard: Allows admins to manage user roles.
- **Data API Routes**:
  - /data/energy_over_time: Returns energy consumption by date.
  - /data/carbon_emissions_distribution: Returns carbon emissions by location.
  - /data/energy_by_device: Returns energy usage by device type.
  - /data/carbon_emissions_over_time: Returns emissions data over time.
  - /data/energy_vs_emissions: Returns energy usage vs. carbon emissions.

---

**Sample Workflows**

**Admin Workflow**

1. **Create an Admin**: (Access /create_admin route once to set up an admin.)
2. **Login**: Login with admin credentials.
3. **Manage Users**: Access /admin_dashboard to view and modify user roles.
4. **View Dashboard**: Access /dashboard with full access to filters and visualizations.

**New User Workflow**

1. **Register**: Go to /register and create an account.
2. **Login**: Log in with the new account.

3. **Dashboard Access**: Access /dashboard to view visualizations, restricted to the assigned role.

---

**Additional Information**

**Error Handling**

1. **Database Errors**: SQLAlchemy's IntegrityError captures issues during data insertion (e.g., duplicate device IDs).
2. **Role-Based Access**: Unauthorized users attempting restricted actions are redirected to login.
3. **Data Validation**: Data from external sources is validated and transformed before being saved in the database.

**Adding New Features**

1. **Additional Roles**: Update models.py, admin_dashboard.html, and role conditions in routes.
2. **New Visualizations**: Define new API routes and add <canvas> elements and JavaScript logic in dashboard.html.
3. **Extended Filtering Options**: Add form fields in dashboard.html and update the backend API routes to handle new filters.