Flask-User

Customizable User Account Management for Flask

By: Salman Butt

Purpose/Goal

- **Authenticate Users**
    - Handle Registrations and Email Confirmations
    - Change Usernames, Change Passwords, Forgotten Passwords

- Also offers Added Security, Increased Reliability, Role-Based Authorization, Internationalization, and Support for Multiple Emails per User

- Fully Customizable through customizable functions and email templates

Use Case/Application of Technology

- Customizable Login Pages
  - Login using Gmail Account
  - Login using Facebook Account

- Authorized access pages for members
  - Ex: Only members who pay can access a certain page

- Role-Based Authorization
  - Ex: only the admin can access a certain webpage
  - Ex: only users that fit a certain role can access a certain page

Getting Started - Installation

- Create a virtual environment for your project
- **pip install flask-user==0.6.15**
- **pip install flask-mail**

Getting Started – User DataModels

- Flask-User allows developers to store Authentication, Email and User information in one DataModel or across several DataModels

- Distinguishes between the following groups of user information:
    1. User Authentication information such as username and password
    2. User Email information such as email address and confirmed_at
    3. User information such as first_name and last_name
    4. User Role information

- Requires User Role information to be stored in a Role DataModel and a UserRole association table

- In my GitHub Link below, the "User_DataModels" folder defines the 4 different models you can use
    - Salman's GitHub Link: https://github.com/salman-butt24/technology_day_materials

Getting Started – Authorization

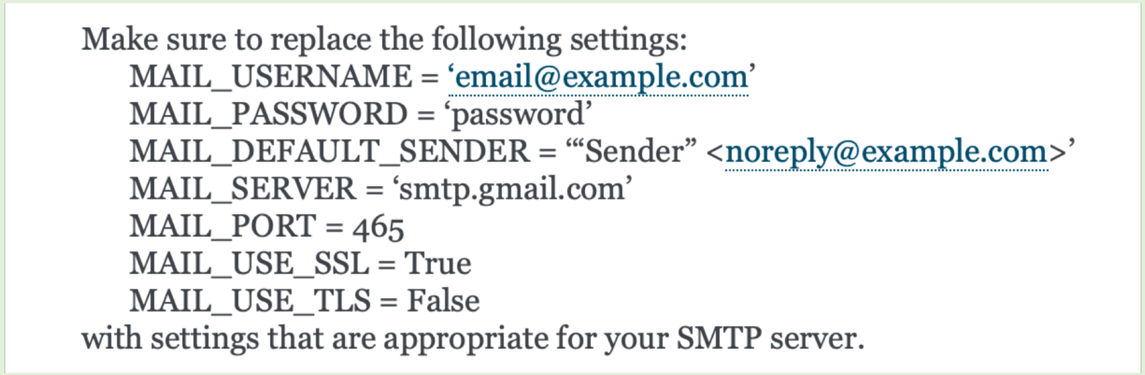
- The process of specifying and enforcing access rights of users to resources

- Flask-User offers role based authorization through the use of function decorators:
  - @login_required
  - @roles_required

- Required Tables; must have following models:
  - The usual User model with an additional ‘roles’ relationship field
  - A Role model with at least one string field called ‘name’
  - A UserRoles association model with a ‘user_id’ field and a ‘role_id’ field
  - Ex: Roles Based App

Application of Technology

- Basic App & Authorization App (Trivial)
  - Register with username and email
  - Email confirmation
  - Login with username or email, Logout
  - Protect pages from unauthorized access
  - Forgot password



Make sure to replace the following settings:

```
    MAIL_USERNAME = 'email@example.com'
    MAIL_PASSWORD = 'password'
    MAIL_DEFAULT_SENDER = '"Sender" <noreply@example.com>'
    MAIL_SERVER = 'smtp.gmail.com'
    MAIL_PORT = 465
    MAIL_USE_SSL = True
    MAIL_USE_TLS = False
```

with settings that are appropriate for your SMTP server.

- Roles Required App (Non-Trivial)
  - Create 'user007' user with 'secret' and 'agent' roles
  - Store user and roles

```
@roles_required('starving', ['artist', 'programmer'])
# Ensures that the user is ('starving' AND (an 'artist' OR a 'programmer'))
```

```
@roles_required(['funny', 'witty', 'hilarious'])
# Notice the usage of square brackets representing an array.
# Array arguments require at least ONE of these roles.
```

Recap Key Concepts

- Authenticate Users

- Create Login Pages
  - Users can register with email or usernames, or both
  - Forgot password, change username, change password
  - Secure password hashing
  - Logout

- Protect Pages from Unauthorized Access
  - Role-Based Authorization
  - Members, Admin, etc.

Resources for Further Reading

- **Flask-User GitHub**: https://github.com/lingthio/Flask-User.git
- **Documentation**: https://flask-user.readthedocs.io/en/v0.6/index.html
- **Using Flask-Login for User Management with Flask:** https://realpython.com/using-flask-login-for-user-management-with-flask/
- **Customization:** https://flask-user.readthedocs.io/en/v0.6/customization.html
- **User Authentication:** https://www.youtube.com/watch?v=BnBjhmspw4c

Q&A