



# Flask – Role Based Access Control

Last Updated : 26 Mar, 2024

[Flask](#) is a micro-framework written in [Python](#). It is used to create web applications using Python. Role-based access control means certain users can access only certain pages. For instance, a normal visitor should not be able to

[Flask Templates](#) [Jinja2](#) [Flask-REST API](#) [Python SQLAlchemy](#) [Flask Bcrypt](#) [Flask Cookies](#) [Json](#) [Postman](#)

a Student can access one page, a Staff can access two, a Teacher can access three, and an Admin accesses four pages.

*Note: For storing users' details we are going to use flask-sqlalchemy and db-browser for performing database actions. You can find detailed tutorial [here](#).*

## Creating the Flask Application

**Step 1:** Create a Python [virtual environment](#).

To avoid any changes in the system environment, it is better to work in a virtual environment.


**Step 2:** Install the required libraries

```
pip install flask flask-security flask-wtf==1.0.1 flask_sqlalchemy  
email-validator
```

**Step 3:** Initialize the flask app.

Import Flask from the flask library and pass `__name__` to Flask. Store this in a variable.

Python3



```
1 # import Flask from flask
2 from flask import Flask
3 # pass current module (__name__) as argument
4 # this will initialize the instance
5
6 app = Flask(__name__)
```


**Step 4:** Configure some settings that are required for running the app.

To do this we use `app.config['_____']`. Using it we can set some important things without which the app might not work.

- **SQLALCHEMY\_DATABASE\_URI** is the path to the database. **SECRET\_KEY** is used for securely signing the session cookie.
- **SECURITY\_PASSWORD\_SALT** is only used if the password hash type is set to something other than plain text.
- **SECURITY\_REGISTERABLE** allows the application to accept new user registrations. **SECURITY\_SEND\_REGISTER\_EMAIL** specifies whether the registration email is sent.

Some of these are not used in our demo, but they are required to mention explicitly.

### Python3



```
1 # path to sqlite database
2 # this will create the db file in instance
3 # if database not present already
4 app.config['SQLALCHEMY_DATABASE_URI'] =
    "sqlite:///g4g.sqlite3"
5 # needed for session cookies
6 app.config['SECRET_KEY'] = 'MY_SECRET'
7 # hashes the password and then stores in the database
8 app.config['SECURITY_PASSWORD_SALT'] = "MY_SECRET"
9 # allows new registrations to application
10 app.config['SECURITY_REGISTERABLE'] = True
11 # to send automatic registration email to user
12 app.config['SECURITY_SEND_REGISTER_EMAIL'] = False
```

**Step 5:** Import SQLAlchemy for database

Because we are using SQLAlchemy for database operations, we need to import and initialize it into the app using `db.init_app(app)`. The `app_context()` keeps track of the application-level data during a request

### Python3



```
1 # import SQLAlchemy for database operations
2 # and store the instance in 'db'
3 from flask_sqlalchemy import SQLAlchemy
4 db = SQLAlchemy()
5 db.init_app(app)
6
7 # runs the app instance
8 app.app_context().push()
```

## Step 6: Create DB Models

For storing the user session information, the **flask-security** library is used. Here to store information of users **UserMixin** is used by importing from the library. Similarly, to store information about the roles of users, **RoleMixin** is used. Both are passed to the database tables' classes.

Here we have created a **user** table for users containing id, email, password, and active status. The **role** is a table that contains the roles created with id and role name. The **roles\_users** table contains the information about which user has what roles. It is dependent on the user and role table for `user_id` and `role_id`, therefore they are referenced from ForeignKeys.

Then we are creating all those tables using `db.create_all()` this will make sure that the tables are created in the database for the first time. Keeping or removing it afterward will not affect the app unless a change is made to the structure of the database code.

### Python3



```
1 # import UserMixin, RoleMixin
2 from flask_security import UserMixin, RoleMixin
3
4 # create table in database for assigning roles
```

```

roles_users = db.Table('roles_users',
6     db.Column('user_id', db.Integer(),
db.ForeignKey('user.id')),
7     db.Column('role_id', db.Integer(),
db.ForeignKey('role.id')))
8
9 # create table in database for storing users
10 class User(db.Model, UserMixin):
11     __tablename__ = 'user'
12     id = db.Column(db.Integer, autoincrement=True,
primary_key=True)
13     email = db.Column(db.String, unique=True)
14     password = db.Column(db.String(255), nullable=False,
server_default='')
15     active = db.Column(db.Boolean())
16     # backreferences the user_id from roles_users table
17     roles = db.relationship('Role', secondary=roles_users,
backref='roled')
18
19 # create table in database for storing roles
20 class Role(db.Model, RoleMixin):
21     __tablename__ = 'role'
22     id = db.Column(db.Integer(), primary_key=True)
23     name = db.Column(db.String(80), unique=True)
24
25 # creates all database tables
26 @app.before_first_request
27 def create_tables():
28     db.create_all()

```

Now, you have tables but you still don't have roles created. You have to create those roles(Admin, Teacher, Staff, Student). Keep in mind that the id for Admin role must be 1, for Teacher: 2, Staff: 3 and Student: 4.

Please create a new file in the same folder as app.py, call it create\_roles.py and add the below code. remember to execute this file post db creation.

### Python

```

1 # create_roles.py
2 from app import Role, User, db
3

```

```

def create_roles():
    5     admin = Role(id=1, name='Admin')
    6     teacher = Role(id=2, name='Teacher')
    7     staff = Role(id=3, name='Staff')
    8     student = Role(id=4, name='Student')
    9
    10    db.session.add(admin)
    11    db.session.add(teacher)
    12    db.session.add(staff)
    13    db.session.add(student)
    14
    15    db.session.commit()
    16    print("Roles created successfully!")
    17
    18    # Function calling will create 4 roles as planned!
    19    create_roles()

```

### Step 7: Define User and Role in Database

We need to pass this database information to flask\_security so as to make the connection between those. For that, we import *SQLAlchemySessionUserDatastore* and pass the table containing users and then the roles. This datastore is then passed to *Security* which binds the current instance of the app with the data. We also import *LoginManager* and *login\_manager* which will maintain the information for the active session. *login\_user* assigns the user as a current user for the session.

#### Python3



```

1  # import required libraries from flask_login and
   flask_security
2  from flask_login import LoginManager, login_manager,
   login_user
3  from flask_security import Security,
   SQLAlchemySessionUserDatastore
4
5  # load users, roles for a session
6  user_datastore = SQLAlchemySessionUserDatastore(db.session,
   User, Role)
7  security = Security(app, user_datastore)

```

## Step 8: Create a Home Route

The home page of our web app is at the '/' route. So, every time the '/' is routed, the code in index.html file will be rendered using a module render\_template. Below the @app.route() decorator, the function needs to be defined so, that code is executed from that function.

### Python3



```
1 # import the required libraries
2 from flask import render_template, redirect, url_for
3
4 # '/' URL is bound with index() function.
5 @app.route('/')
6 # defining function index which returns the rendered html
  code
7 # for our home page
8 def index():
9     return render_template("index.html")
```

### index.html

Below is the HTML code for index.html. Some logic is applied using the [Jinja2](#) templating engine which behaves similarly to python. The current\_user variable stores the information of the currently logged-in user. So, here {% if current\_user.is\_authenticated %} means if a user is logged in show that code: {{ current\_user.email }} is a variable containing the email of the current user. Because the user can have many roles so roles are a list, that's why a for loop is used. Otherwise, the code in {% else %} part is rendered, finally getting out of the if block with {% endif %}.

### HTML



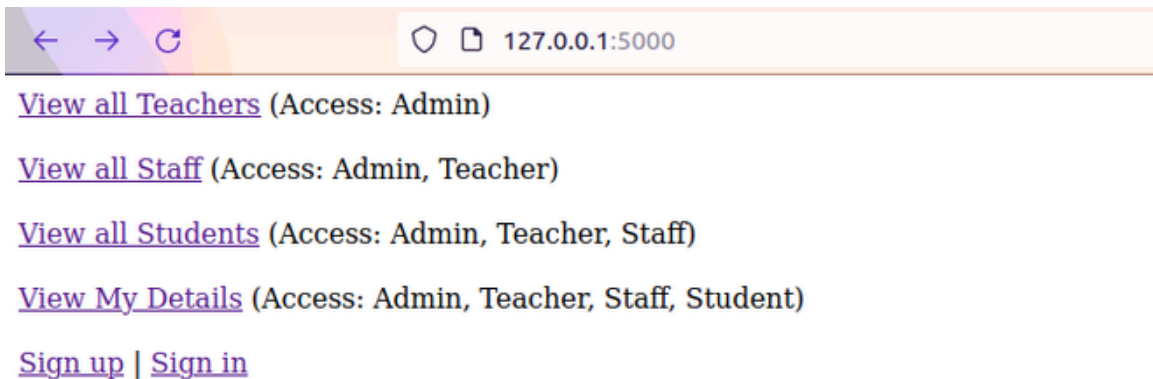
```
1 <!-- index.html -->
2
3 <!-- links to the pages -->
4 <a href="/teachers">View all Teachers</a> (Access: Admin)
  <br><br>
5 <a href="/staff">View all Staff</a> (Access: Admin, Teacher)
  <br><br>
```

```

    <a href="/students">View all Students</a> (Access: Admin,
    Teacher, Staff)<br><br>
7  <a href="/mydetails">View My Details</a> (Access: Admin,
    Teacher, Staff, Student)
8  <br><br>
9  <!-- Show only if user is logged in -->
10 {% if current_user.is_authenticated %}
11     <!-- Show current users email -->
12     <b>Current user</b>: {{current_user.email}}
13     <!-- Current users roles -->
14     | <b>Role</b>: {% for role in current_user.roles%}
15         {{role.name}}
16     {% endfor %} <br><br>
17     <!-- link for logging out -->
18     <a href="/logout">Logout</a>
19 <!-- Show if user is not logged in -->
20 {% else %}
21     <a href="/signup">Sign up</a> | <a href="/signin">Sign
    in</a>
22 {% endif %}
23 <br><br>

```

### Output:



### Step 9: Create Signup Route

If the user visits the '/signup' route the request is GET, so the else part will render this HTML page, and if the form is submitted the code in if the condition that is POST method is executed.

The data in the HTML form is requested [using the request module](#). First, we check if the user already exists in the database by querying for the user using the email provided and passing the msg according to that.

If not then we add the user and append the chosen role to the roles\_users DB table, for this we query for the role using the id that we will get from options in the radio button, this will return an object containing all the column attributes of that role, in this case, the id and name of the role. And then log the user in for the user's current session with **login\_user(user)**.

### Python3



```
1 # import 'request' to request data from html
2 from flask import request
3
4 # signup page
5 @app.route('/signup', methods=['GET', 'POST'])
6 def signup():
7     msg=""
8     # if the form is submitted
9     if request.method == 'POST':
10         # check if user already exists
11         user =
12         User.query.filter_by(email=request.form['email']).first()
13         msg=""
14         # if user already exists render the msg
15         if user:
16             msg="User already exist"
17             # render signup.html if user exists
18             return render_template('signup.html', msg=msg)
19
20         # if user doesn't exist
21
22         # store the user to database
23         user = User(email=request.form['email'], active=1,
24             password=request.form['password'])
25         # store the role
26         role =
27         Role.query.filter_by(id=request.form['options']).first()
28         user.roles.append(role)
29
30         # commit the changes to database
```



```

28         db.session.add(user)
29         db.session.commit()
30
31         # login the user to the app
32         # this user is current user
33         login_user(user)
34         # redirect to index page
35         return redirect(url_for('index'))
36
37     # case other than submitting form, like loading the page
    itself
38     else:
39         return render_template("signup.html", msg=msg)

```

### signup.html page:

Here in the form, the action is '#' which means after submitting the form the current page itself is loaded. The method in the form is *POST* because we are creating a new entry in the database. There are fields for email, password, and choosing a role with a radio button. In the radio button, the *value* should be different because that's the main differentiator of the chosen role.

Also, an if condition is applied using Jinja2, {% if %} which checks that if a user is logged in, and if not {% else %} then only shows the form otherwise just shows the already logged-in message.

### HTML



```

1  <!-- signup.html -->
2
3  <h2>Sign up</h2>
4
5  <!-- Show only if user is logged in -->
6  {% if current_user.is_authenticated %}
7      You are already logged in.
8
9  <!-- Show if user is NOT logged in -->
10 {% else %}
11 {{ msg }}<br>
12
13 <!-- Form for signup -->

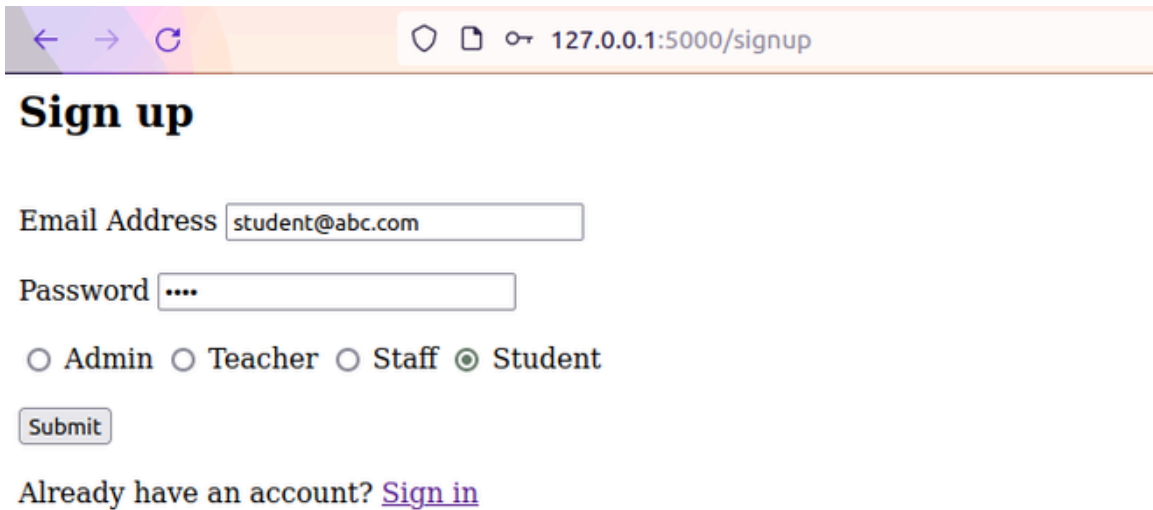
```

```

15         <label>Email Address </label>
16         <input type="text" name="email" required />
17     <br><br>
18         <label>Password </label>
19         <input type="password" name="password"
20     required/><br><br>
21         <!-- Options to choose role -->
22         <!-- Give the role ids in the value -->
23         <input type="radio" name="options"
24     id="option1" value=1 required> Admin </input>
25         <input type="radio" name="options" id="option2"
26     value=2> Teacher </input>
27         <input type="radio" name="options" id="option3"
28     value=3> Staff </input>
29         <input type="radio" name="options"
30     id="option3" value=4> Student </input><br>
31     <br>
32         <button type="submit">Submit</button><br><br>
33
34         <!-- Link for signin -->
35         <span>Already have an account?</span>
36         <a href="/signin">Sign in</a>
37     </form>
38 <!-- End the if block -->
39 {% endif %}

```

Output:



**Sign up**

Email Address

Password

☐ Admin ☐ Teacher ☐ Staff ☒ Student

Already have an account? [Sign in](#)

### Step 10: Create Signin Route

As you might have noticed we are using two methods GET, and POST. That is because we want to know if the user has just loaded the page (GET) or submitted the form (POST). Then we check if the user exists by querying the database. If the user exists then we see if the password matches. If both are validated the user is logged in using *login\_user(user)*. Otherwise, the msg is passed to HTML accordingly i.e., if the password is wrong msg is set to “Wrong password” and if the user doesn’t exist then the msg is set to “User doesn’t exist”.

#### Python3

```
1 # signin page
2 @app.route('/signin', methods=['GET', 'POST'])
3 def signin():
4     msg=""
5     if request.method == 'POST':
6         # search user in database
7         user =
8         User.query.filter_by(email=request.form['email']).first()
9         # if exist check password
10        if user:
11            if user.password == request.form['password']:
12                # if password matches, login the user
13                login_user(user)
14                return redirect(url_for('index'))
```

```

        # if password doesn't match
15         else:
16             msg="Wrong password"
17
18         # if user does not exist
19         else:
20             msg="User doesn't exist"
21         return render_template('signin.html', msg=msg)
22
23     else:
24         return render_template("signin.html", msg=msg)

```

## signin.html

Similar to the signup page, check if a user is already logged in, if not then show the form asking for email and password. The form method should be POST. Ask in the form for, email and password. We can also show links for sign-up optionally.

### HTML



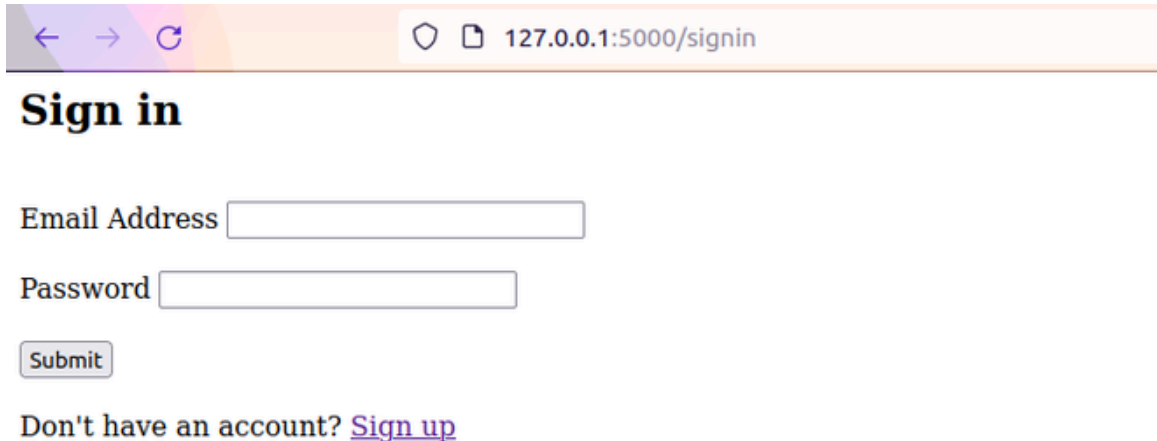
```

1  <!-- signin.html -->
2
3  <h2>Sign in</h2>
4
5  <!-- Show only if user is logged in -->
6  {% if current_user.is_authenticated %}
7      You are already logged in.
8
9  <!-- Show if user is NOT logged in -->
10 {% else %}
11 <!-- msg that was passed while rendering template -->
12 {{ msg }}<br>
13
14 <form action="#" method="POST" id="signin-form">
15     <label>Email Address </label>
16     <input type="text" name="email" required />
17     <br><br>
18     <label>Password </label>
19     <input type="password" name="password"
    required/><br><br>

```

```
21         <input class="btn btn-primary" type="submit"  
value="Submit"><br><br>  
22  
23         <span>Don't have an account?</span>  
24         <a href="/signup">Sign up</a>  
25     </form>  
26 {% endif %}
```

Output:



← → ↻ 127.0.0.1:5000/signin

## Sign in

Email Address

Password

Don't have an account? [Sign up](#)

### Step 11: Create a Teacher Route

We are passing the users with the role of Teacher to the HTML template. On the home page if we click any link then it will load the same page if the user is not signed in. If the user is signed in we want to give Role Based Access so that the user with the role:

- **Students** can access *View My Details* page.
- **Staff** can access *View My Details* and *View all Students* pages.
- **The teacher** can access *View My Details*, *View all Students*, and *View all Staff* pages.
- **Admin** can access *View My Details*, *View all Students*, *View all Staff*, and *View all Teachers* pages.

We need to import, **roles\_accepted**: this will check the database for the role of the user and if it matches the specified roles then only the user is given access

to that page. The teacher's page can be accessed by Admin only using `@roles_accepted('Admin')`.

### Python3



```
1 # to implement role based access
2 # import roles_accepted from flask_security
3 from flask_security import roles_accepted
4
5 # for teachers page
6 @app.route('/teachers')
7 # only Admin can access the page
8 @roles_accepted('Admin')
9 def teachers():
10     teachers = []
11     # query for role Teacher that is role_id=2
12     role_teachers =
13     db.session.query(roles_users).filter_by(role_id=2)
14     # query for the users' details using user_id
15     for teacher in role_teachers:
16         user =
17         User.query.filter_by(id=teacher.user_id).first()
18         teachers.append(user)
19     # return the teachers list
20     return render_template("teachers.html",
21                             teachers=teachers)
```

### teachers.html

The *teachers* passed in the `render_template` is a list of objects, containing all the columns of the user table, so we're using [Python for loop](#) in jinja2 to show the elements in the list in HTML ordered list tag.

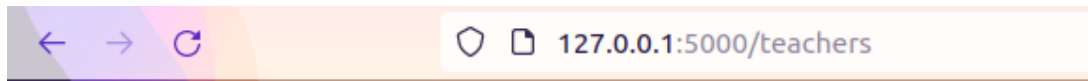
### HTML



```
1 <!-- teachers.html -->
2
3 <h3>Teachers</h3>
4
5 <!-- list that shows all teachers' email -->
6 <ol>
```

```
        {% for teacher in teachers %}
8      <li>
9        {{teacher.email}}
10     </li>
11   {% endfor %}
12 </ol>
```

Output:



## Teachers

1. teacher@abc.com
2. teacher1@abc.com
3. teacher3@any.in

### Step 12: Create staff, student, and mydetail Routes

Similarly, routes for other pages are created by adding the roles to the decorator `@roles_accepted()`.

#### Python3

```
1 # for staff page
2 @app.route('/staff')
3 # only Admin and Teacher can access the page
4 @roles_accepted('Admin', 'Teacher')
5 def staff():
6     staff = []
7     role_staff =
8     db.session.query(roles_users).filter_by(role_id=3)
9     for staf in role_staff:
10         user =
11         User.query.filter_by(id=staf.user_id).first()
12         staff.append(user)
13     return render_template("staff.html", staff=staff)
14
15 # for student page
16 @app.route('/students')
```

```

# only Admin, Teacher and Staff can access the page
16 @roles_accepted('Admin', 'Teacher', 'Staff')
17 def students():
18     students = []
19     role_students =
db.session.query(roles_users).filter_by(role_id=4)
20     for student in role_students:
21         user =
User.query.filter_by(id=student.user_id).first()
22         students.append(user)
23     return render_template("students.html",
students=students)
24
25 # for details page
26 @app.route('/mydetails')
27 # Admin, Teacher, Staff and Student can access the page
28 @roles_accepted('Admin', 'Teacher', 'Staff', 'Student')
29 def mydetails():
30     return render_template("mydetails.html")

```

## staff.html

Here, we are iterating all the staff and extracting their email IDs.

### HTML



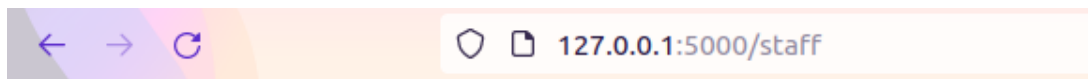
```

1  <!-- staff.html -->
2
3  <h3>Staff</h3>
4  <ol>
5  {% for staf in staff %}
6  <li>
7  {{staf.email}}
8  </li>
9  {% endfor %}
10 </ol>

```

## Output:





## Staff

1. staff@abc.com
2. staff1@abc.com
3. staff3@efg.afd

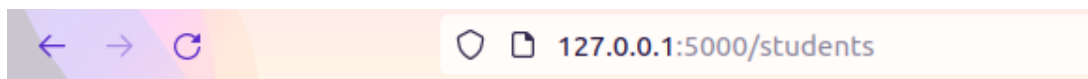
### student.html

Here, we are iterating all the students and extracting their email IDs.

#### HTML

```
1 <!-- students.html -->
2
3 <h3>Students</h3>
4 <ol>
5 {% for student in students %}
6 <li>
7 {{student.email}}
8 </li>
9 {% endfor %}
10 </ol>
```

Output:



## Students

1. student@abc.com
2. student1@abc.com
3. student3@abc.com

### mydetails.html

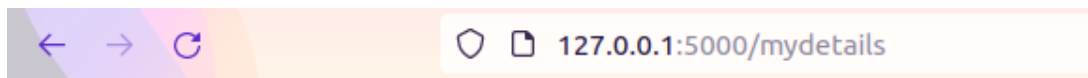
Similar to the index page, to show the role use a for loop from Jinja2, because a user can more than one role i.e., `current_user.roles` is a list of roles that were queried from the database.

### HTML



```
1 <!-- mydetails.html -->
2
3 <h3>My Details</h3><br>
4 <b>My email</b>: {{current_user.email}}
5 | <b>Role</b>: {% for role in current_user.roles%}
6                 {{role.name}}
7                 {% endfor %} <br><br>
```

Output:



## My Details

**My email:** admin@abc.com | **Role:** Admin

**Step 13:** Finally, Add code Initializer.

Here, debug is set to True. When in a development environment. It can be set to False when the app is ready for production.

### Python3



```
1 #for running the app
2 if __name__ == "__main__":
3     app.run(debug = True)
```

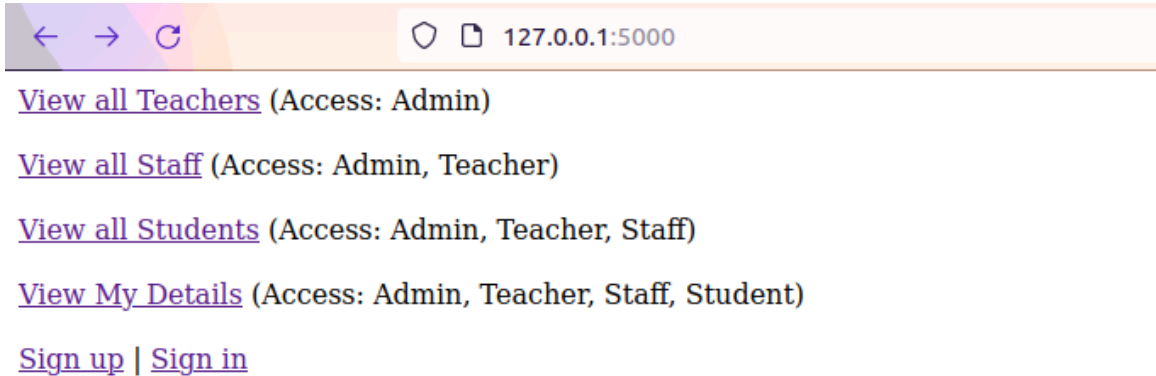
Now test your app by running the below command in the terminal.

```
python app.py
```

Go to:

`http://127.0.0.1:5000`

Output:



Looking to dive into the world of programming or sharpen your Python skills? Our [Master Python: Complete Beginner to Advanced Course](#) is your ultimate guide to becoming proficient in Python. This course covers everything you need to build a solid foundation from fundamental programming concepts to advanced techniques. With **hands-on projects**, real-world examples, and expert guidance, you'll gain the confidence to tackle complex **coding challenges**. Whether you're starting from scratch or aiming to enhance your skills, this course is the perfect fit. Enroll now and master Python, the language of the future!

P prasa...



59

## Previous Article

How to store username and password in Flask

## Next Article

How to use Flask-Session in Python Flask ?

## Similar Reads

### Documenting Flask Endpoint using Flask-Autodoc

Documentation of endpoints is an essential task in web development and being able to apply it in different frameworks is always a utility. This article discusses...

4 min read

### How to use Flask-Session in Python Flask ?

Flask Session - Flask-Session is an extension for Flask that supports Server-side Session to your application. The Session is the time between the client logs in to...

4 min read

### How to Integrate Flask-Admin and Flask-Login

In order to merge the admin and login pages, we can utilize a short form or any other login method that only requires the username and password. This is know...

8 min read

### Minify HTML in Flask using Flask-Minify

Flask offers HTML rendering as output, it is usually desired that the output HTML should be concise and it serves the purpose as well. In this article, we would...

12 min read

### Flask URL Helper Function - Flask url\_for()

In this article, we are going to learn about the flask url\_for() function of the flask URL helper in Python. Flask is a straightforward, speedy, scalable library, used f...

11 min read

### Access the Query String in Flask Routes

In this article, we are going to see how to Access the Query String in Flask Routes in Python. The query string is the portion of the URL and Query Parameters are...

2 min read

### Class Based vs Function Based Views - Which One is Better to Use in Django?

Django...We all know the popularity of this python framework all over the world. This framework has made life easier for developers. It has become easier for...

7 min read

## Python Flask - ImmutableMultiDict

MultiDict is a sub-class of Dictionary that can contain multiple values for the same key, unlike normal Dictionaries. It is used because some form elements ha...

2 min read

## Handling 404 Error in Flask

Prerequisite: Creating simple application in Flask A 404 Error is showed whenever a page is not found. Maybe the owner changed its URL and forgot to...

4 min read

## Python | Using for loop in Flask

Prerequisite: HTML Basics, Python Basics, Flask It is not possible to write front-end course every time user make changes in his/her profile. We use a template...

3 min read

**Article Tags :**[Python](#)[Technical Scriptor](#)[Python Flask](#)[Technical Scriptor 2022](#)**Practice Tags :**[python](#)



Corporate & Communications Address:-  
A-143, 9th Floor, Sovereign Corporate  
Tower, Sector- 136, Noida, Uttar Pradesh  
(201305) | Registered Address:- K 061,  
Tower K, Gulshan Vivante Apartment,  
Sector 137, Noida, Gautam Buddh  
Nagar, Uttar Pradesh, 201305



## Company

About Us  
Legal  
In Media  
Contact Us  
Advertise with us  
GFG Corporate Solution  
Placement Training Program  
GeeksforGeeks Community

## DSA

Data Structures  
Algorithms  
DSA for Beginners  
Basic DSA Problems  
DSA Roadmap  
Top 100 DSA Interview Problems  
DSA Roadmap by Sandeep Jain  
All Cheat Sheets

## Web Technologies

HTML  
CSS  
JavaScript  
TypeScript  
ReactJS  
NextJS  
Bootstrap  
Web Design

## Computer Science

Operating Systems  
Computer Network  
Database Management System  
Software Engineering  
Digital Logic Design  
Engineering Maths  
Software Development  
Software Testing

## System Design

High Level Design  
Low Level Design  
UML Diagrams  
Interview Guide  
Design Patterns

## Languages

Python  
Java  
C++  
PHP  
GoLang  
SQL  
R Language  
Android Tutorial  
Tutorials Archive

## Data Science & ML

Data Science With Python  
Data Science For Beginner  
Machine Learning  
ML Maths  
Data Visualisation  
Pandas  
NumPy  
NLP  
Deep Learning

## Python Tutorial

Python Programming Examples  
Python Projects  
Python Tkinter  
Web Scraping  
OpenCV Tutorial  
Python Interview Question  
Django

## DevOps

Git  
Linux  
AWS  
Docker  
Kubernetes  
Azure  
GCP  
DevOps Roadmap

## Interview Preparation

Competitive Programming  
Top DS or Algo for CP  
Company-Wise Recruitment Process  
Company-Wise Preparation  
Aptitude Preparation

OOAD  
System Design Bootcamp  
Interview Questions

Puzzles

School Subjects

Mathematics  
Physics  
Chemistry  
Biology  
Social Science  
English Grammar  
Commerce  
World GK

GeeksforGeeks Videos

DSA  
Python  
Java  
C++  
Web Development  
Data Science  
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved