Tutorial to add authentication and authorization to your Flask app (art gallery) using Flask-Login.

Step 1: Install Flask-Login

First, you need to install Flask-Login:

```
pip install Flask-Login
```

Step 2: Update the User Model

Ensure your User model implements the required methods for Flask-Login. Add necessary fields like password_hash for authentication.

```
from werkzeug.security import generate_password_hash,
check_password_hash

from flask_login import UserMixin


class User(UserMixin, db.Model):

    id = db.Column(db.Integer, primary_key=True)

    username = db.Column(db.String(150), unique=True, nullable=False)

    email = db.Column(db.String(150), unique=True, nullable=False)

    password_hash = db.Column(db.String(256), nullable=False)


    def set_password(self, password):

        self.password_hash = generate_password_hash(password)


    def check_password(self, password):

        return check_password_hash(self.password_hash, password)


    def json(self):

        return {
            'id': self.id,
            'username': self.username,
            'email': self.email
        }
```

Step 3: Initialize Flask-Login

```
from flask import Flask, redirect, url_for
from flask_login import LoginManager

app = Flask(__name__)
app.config.from_object(config.Config)

 Initialize Flask-Login
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'users.login'

 Initialize the database
Database.initialize(app)

 Register the blueprints
app.register_blueprint(artists_blueprint)
app.register_blueprint(artifacts_blueprint)
app.register_blueprint(users_blueprint)

app.config['artifact_repo'] =
artifacts_repository.ArtifactRepository()
app.config['artist_repo'] = artists_repository.ArtistRepository()
app.config['user_repo'] = users_repository.UserRepository()
```

```
 Create the tables
with app.app_context():
    db.create_all()

 Root route redirecting to /artists
@app.route('/')
def index():
    return redirect(url_for('artists.get_all_artists'))

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

if __name__ == '__main__':
    app.run(debug=True)
```

Step 4: Create Login and Logout Views

Add views for login and logout:

```
from flask import Blueprint, request, render_template, redirect,
url_for, flash
from flask_login import login_user, logout_user, login_required,
current_user
from models.user import User
from persistence.IUserDataAccess import IUserDataAccess

users_blueprint = Blueprint('users', __name__)

def get_user_repository() -> IUserDataAccess:
    return current_app.config['user_repo']
```

```python
@users_blueprint.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
            login_user(user)
            return redirect(url_for('index'))
        else:
            flash('Invalid username or password')

    return render_template('login.html')

@users_blueprint.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('users.login'))

@users_blueprint.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        repo = get_user_repository()

        if User.query.filter_by(username=username).first() is not
None:
            flash('Username already exists')
            return redirect(url_for('users.register'))

        if User.query.filter_by(email=email).first() is not None:
            flash('Email already registered')
            return redirect(url_for('users.register'))

        new_user = User(username=username, email=email)
        new_user.set_password(password)
        repo.create_user(new_user)

        flash('Registration successful! Please log in.')
        return redirect(url_for('users.login'))

    return render_template('register.html')
```

Step 5: Protect Routes with @login_required

Use @login_required to protect routes that require authentication:

```
from flask_login import login_required

@app.route('/protected')
@login_required
def protected():
    return 'This is a protected route.'
```

Step 6: Create Login and Register Templates

Create login.html and register.html templates:

login.html

```html
html
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form method="post">
        <label for="username">Username</label>
        <input type="text" id="username" name="username">
        <label for="password">Password</label>
        <input type="password" id="password" name="password">
        <button type="submit">Login</button>
    </form>
    <p>Don't have an account? <a href="{{ url_for('users.register')
}}">Register</a></p>
</body>
</html>
```

register.html

```html
html
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
```

```
</head>
<body>
    <h1>Register</h1>
    <form method="post">
        <label for="username">Username</label>
        <input type="text" id="username" name="username">
        <label for="email">Email</label>
        <input type="email" id="email" name="email">
        <label for="password">Password</label>
        <input type="password" id="password" name="password">
        <button type="submit">Register</button>
    </form>
    <p>Already have an account? <a href="{{ url_for('users.login')
}}">Login</a></p>
</body>
</html>
```

Step 7: Update User Repository

Update your UserRepository to handle the new user creation with hashed passwords:

```
from models.user import User
from utils.database import db
from persistence.IUserDataAccess import IUserDataAccess

class UserRepository(IUserDataAccess):
    def create_user(self, user: User):
        db.session.add(user)
        db.session.commit()

    def get_all_users(self):
        return User.query.all()

    def get_user_by_id(self, user_id: int):
        return User.query.get(user_id)

    def update_user(self, user: User):
        db.session.commit()

    def delete_user(self, user_id: int):
        user = User.query.get(user_id)
        if user:
            db.session.delete(user)
            db.session.commit()
            return True
        return False
```

Step 8: Test Your Application

Run your Flask app and navigate to the login and registration pages to test the authentication functionality.

```
flask run
```

Navigate to http://localhost:5000/login and http://localhost:5000/register to test login and registration.

Optional: Adding Role-Based Authorization

If you want to add role-based authorization, you can extend the User model with roles and implement checks in your routes.

User Model:

```
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    password_hash = db.Column(db.String(256), nullable=False)
    role = db.Column(db.String(50), nullable=False, default='user')

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

    def json(self):
        return {
            'id': self.id,
            'username': self.username,
            'email': self.email,
            'role': self.role
        }
```

```
Check Role in Routes:
python
from flask import abort

@app.route('/admin')
@login_required
def admin_route():
    if current_user.role != 'admin':
        abort(403)
    return 'This is an admin route.'
```

This completes the tutorial on adding authentication and authorization to your Flask art gallery app using Flask-Login.