EXPERIMENT NO. 6

AIM : Implement authentication and user roles with JWT

Code :

> Users > Lenovo > Desktop > jwt-auth-app > models.py > ...

```python
from werkzeug.security import generate_password_hash, check_password_hash

class User:
    def __init__(self, id, username, password, role='user'):
        self.id = id
        self.username = username
        self.password_hash = generate_password_hash(password)
        self.role = role

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

# In-memory user store
users = []
user_id_counter = 1
```

OBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Users\Lenovo>

C: > Users > Lenovo > Desktop > TODO.md > # TODO: Implement JWT Authentication and User Roles

```markdown
# TODO: Implement JWT Authentication and User Roles

- [x] Create project directory `jwt-auth-app`
- [x] Create `requirements.txt` with necessary dependencies
- [x] Create `models.py` for User model with roles
- [x] Create `auth.py` for authentication routes (register, login)
- [x] Create `app.py` for main application setup
- [x] Create `main.py` for protected routes with role checks
- [x] Install dependencies using pip
- [x] Create Postman collection for API validation
- [x] Run the Flask application and test endpoints
```

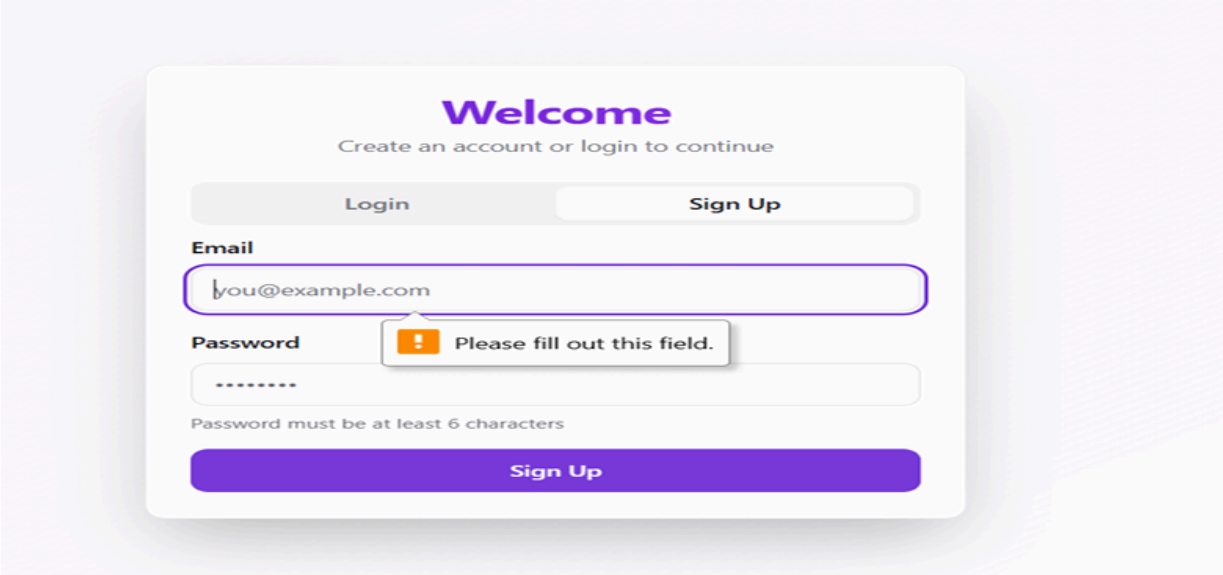
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\Lenovo>

```
C: > Users > Lenovo > Desktop > jwt-auth-app > app.py > ...
1   from flask import Flask
2   from flask_jwt_extended import JWTManager
3
4   app = Flask(__name__)
5   app.config['JWT_SECRET_KEY'] = 'super-secret-key'  # Change this in production
6
7   jwt = JWTManager(app)
8
9   from auth import auth_bp
10  from main import main_bp
11  app.register_blueprint(auth_bp, url_prefix='/auth')
12  app.register_blueprint(main_bp, url_prefix='/api')
13
14  if __name__ == '__main__':
15      app.run(debug=True)
16

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\Lenovo>
```

OUTPUT :



Conclusion

In conclusion, implementing authentication and user roles with JWT provides a secure, stateless solution for managing user access. JWTs ensure data integrity, scalability, and flexibility across platforms. By embedding user roles in the token, you can enforce fine-grained access control. While offering many advantages, it's essential to handle token security, expiration, and storage properly to prevent vulnerabilities. Overall, JWT is an efficient and scalable choice for modern authentication systems.