MODUL 14

Flask Authentication & Authorization

Login • Registration • Session • Password Hashing • Role-Based Access Control

Level: Intermediate | Python Web Development

Deskripsi Modul

Modul ini membahas cara melindungi aplikasi web Flask dengan autentikasi dan otorisasi. Peserta mempelajari perbedaan identitas dan hak akses, registrasi, login, logout, pengelolaan sesi, penyimpanan password secara aman, pembatasan halaman berdasarkan status login, serta pengaturan akses berdasarkan peran (role). Implementasi menggunakan Flask-Login, Werkzeug password hashing, Flask-WTF untuk perlindungan CSRF, dan SQLite melalui Flask-SQLAlchemy.

Prasyarat dan Perangkat

- Memahami Python dasar, class, decorator, exception handling, dan virtual environment.
- Memahami route Flask, template Jinja2, form handling, SQLite, SQLAlchemy, dan CRUD (Modul 12–13).
- Python 3, editor kode, browser, dan terminal.

Instalasi

```
python -m venv .venv

# Windows PowerShell
.venv\\Scripts\\Activate.ps1

# macOS/Linux
source .venv/bin/activate

pip install Flask Flask-SQLAlchemy Flask-Login Flask-WTF email-validator

pip freeze > requirements.txt
```

Contoh kode mengikuti pola Flask-SQLAlchemy 3.x, Flask-Login, dan Flask-WTF. Pastikan versi dependensi yang digunakan konsisten dan diuji pada lingkungan pembelajaran.

1. Learning Objectives

Setelah menyelesaikan modul ini, peserta mampu:

- Menjelaskan perbedaan authentication (autentikasi) dan authorization (otorisasi).
- Menerapkan registrasi pengguna dan validasi input.
- Menyimpan password menggunakan hash, bukan plaintext.
- Mengimplementasikan login, logout, dan pengelolaan sesi dengan Flask-Login.
- Melindungi route agar hanya dapat diakses pengguna yang sudah login.
- Menerapkan role-based access control (RBAC) untuk membatasi fitur berdasarkan peran.
- Menggunakan CSRF protection pada form yang mengubah data.
- Menerapkan prinsip least privilege dan pengujian akses.
- Membangun aplikasi portal pengguna dengan role admin dan user.

2. Learning Materials

2.1 Authentication dan Authorization

Authentication menjawab pertanyaan “siapa pengguna ini?” dengan memverifikasi identitas, misalnya melalui email dan password. Authorization menjawab “apa yang boleh dilakukan pengguna ini?” setelah identitasnya diketahui.

Aspek	Authentication	Authorization
Tujuan	Memverifikasi identitas pengguna	Menentukan izin atau hak akses
Contoh	Login dengan email dan password	Admin boleh mengelola pengguna
Kapan	Saat pengguna membuktikan identitas	Saat pengguna mengakses resource/ fitur

Kegagalan	Login ditolak	Akses ditolak (403 atau redirect)

Login yang berhasil tidak otomatis berarti pengguna boleh mengakses semua fitur. Setiap route dan operasi penting tetap harus memeriksa hak akses.

2.2 Siklus Autentikasi Web

```
Pengguna mengirim email + password
              |
              v
      Flask menerima form
              |
              v
    Cari user berdasarkan email
              |
              v
    Verifikasi password hash
         /          \\
     Tidak cocok    Cocok
        |             |
   Login ditolak   login_user(user)
                      |
                      v
              Session/cookie
                      |
                      v
            Request berikutnya
                      |
                      v
            current_user tersedia
                      |
                      v
            Pemeriksaan hak akses
```

Cookie sesi biasanya menyimpan identifier atau data sesi yang ditandatangani, bukan password. Cookie harus dilindungi dengan konfigurasi keamanan yang tepat. Jangan menyimpan password atau data rahasia di dalam cookie.

2.3 Password Hashing

Password tidak boleh disimpan dalam bentuk teks biasa (plaintext). Aplikasi perlu menggunakan password hashing yang dirancang untuk password, dengan salt dan faktor kerja yang sesuai. Werkzeug menyediakan generate_password_hash() dan check_password_hash().

```
from werkzeug.security import generate_password_hash, check_password_hash

# Saat registrasi
password_hash = generate_password_hash(password)

# Saat login
if check_password_hash(user.password_hash, password):
    print("Password cocok")
else:
    print("Password tidak cocok")
```

- Simpan hasil hash, bukan password asli.
- Jangan membuat algoritma hashing sendiri atau menggunakan MD5/SHA-1 untuk password.
- Jangan pernah mencatat password ke log.
- Gunakan kebijakan password yang wajar dan mekanisme reset password yang aman.
- Hash bukan enkripsi: password asli tidak seharusnya dapat dipulihkan dari hash.

2.4 Flask-Login

Flask-Login mengelola status login pada sesi Flask. Fitur pentingnya antara lain LoginManager, UserMixin, login_user(), logout_user(), current_user, dan login_required. Flask-Login tidak menyediakan registrasi pengguna, penyimpanan password, atau sistem role secara otomatis; bagian tersebut perlu dibuat oleh aplikasi.

Komponen	Fungsi
LoginManager	Menghubungkan Flask-Login dengan aplikasi.
UserMixin	Menyediakan properti/metode dasar untuk objek user.
user_loader	Memuat pengguna dari ID sesi.
login_user()	Menandai pengguna sebagai login.
logout_user()	Mengakhiri sesi login pengguna.
current_user	Merepresentasikan pengguna pada request saat ini.
login_required	Melindungi route untuk pengguna yang login.

2.5 Session dan Cookie

Session mempertahankan status pengguna di antara request HTTP. Flask menggunakan SECRET_KEY untuk menandatangani cookie sesi. SECRET_KEY harus acak, rahasia, dan disimpan melalui environment variable atau secret manager, bukan ditulis sebagai nilai tetap pada source code.

- SESSION_COOKIE_HTTPONLY=True membantu mencegah JavaScript membaca cookie.
- SESSION_COOKIE_SECURE=True mengharuskan HTTPS; aktifkan pada deployment HTTPS, bukan pengujian HTTP lokal.
- SESSION_COOKIE_SAMESITE='Lax' membantu mengurangi sebagian risiko pengiriman cookie lintas situs.
- Atur masa berlaku sesi sesuai kebutuhan dan pertimbangkan logout pada perangkat bersama.
- Gunakan HTTPS untuk seluruh halaman autentikasi dan sesi.

2.6 Authorization dan Role-Based Access Control

Role-Based Access Control (RBAC) mengelompokkan izin berdasarkan peran. Contoh peran pada aplikasi pembelajaran adalah admin dan user. Admin dapat mengelola akun, sedangkan user hanya dapat mengakses dashboard dan profil sendiri.

Role	Hak akses contoh
Guest	Melihat halaman publik dan formulir login/registrasi.
User	Melihat dashboard dan mengakses profil miliknya.
Admin	Mengakses dashboard admin dan mengelola daftar pengguna.

Pemeriksaan role harus dilakukan di backend pada setiap route atau operasi sensitif. Menyembunyikan tombol admin di HTML saja bukan mekanisme keamanan. Untuk aplikasi lebih kompleks, gunakan permission granular dan kebijakan akses berbasis resource.

2.7 Decorator untuk Pembatasan Role

```
from functools import wraps
from flask import abort
from flask_login import current_user

def role_required(role):
    def decorator(view_function):
        @wraps(view_function)
        def wrapped_view(*args, **kwargs):
            if not current_user.is_authenticated:
                abort(401)
            if current_user.role != role:
                abort(403)
            return view_function(*args, **kwargs)
        return wrapped_view
    return decorator
```

Decorator di atas adalah contoh sederhana untuk satu role. Untuk beberapa role, ubah parameter menjadi daftar role yang diizinkan. Pastikan decorator dipasang bersama login_required dan urutannya dipahami; contoh implementasi lengkap tersedia pada bagian code examples.

2.8 CSRF Protection

Cross-Site Request Forgery (CSRF) adalah serangan yang dapat membuat browser pengguna mengirim request tanpa persetujuan yang semestinya. Form yang mengubah data, seperti registrasi, login, edit, dan delete, perlu perlindungan CSRF. Flask-WTF menyediakan CSRFProtect dan token pada form.

```
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect()

def create_app():
    app = Flask(__name__)
    app.config["SECRET_KEY"] = "ambil-dari-environment"
    csrf.init_app(app)
    return app
```

Pada contoh di bawah, token CSRF dikirim sebagai hidden field pada setiap form POST. Jangan menonaktifkan CSRF pada aplikasi produksi hanya untuk mengatasi error token.

2.9 Praktik Keamanan Tambahan

- Gunakan pesan login generik agar tidak membocorkan apakah email terdaftar.
- Batasi percobaan login dan pertimbangkan rate limiting untuk mengurangi brute force.
- Validasi input di server; validasi browser bukan pengganti validasi server.
- Lindungi halaman dan data milik pengguna dengan pemeriksaan ownership, bukan hanya status login.
- Jangan mengizinkan pengguna memilih role admin saat registrasi publik.
- Jangan gunakan debug=True pada production.
- Gunakan dependency yang diperbarui, logging aman, backup, dan konfigurasi secret yang benar.
- Pertimbangkan MFA untuk sistem dengan risiko tinggi.

3. Code Examples

3.1 Struktur Folder

```
flask_auth_app/
├── app.py
├── requirements.txt
├── instance/
│   └── auth.db
├── templates/
│   ├── base.html
│   ├── register.html
│   ├── login.html
│   ├── dashboard.html
│   ├── admin.html
│   └── profile.html
└── static/
    └── css/
        └── style.css
```

3.2 Backend: Model, Login Manager, dan Konfigurasi

```
# app.py
import os
from functools import wraps

from flask import (
    Flask, render_template, request, redirect,
    url_for, flash, abort
)
from flask_sqlalchemy import SQLAlchemy
from flask_login import (
```

```
    LoginManager, UserMixin, login_user, logout_user,
    login_required, current_user
)
from flask_wtf.csrf import CSRFProtect
from werkzeug.security import generate_password_hash, check_password_hash
from sqlalchemy.exc import IntegrityError

app = Flask(__name__)
app.config["SECRET_KEY"] = os.environ.get("FLASK_SECRET_KEY")
if not app.config["SECRET_KEY"]:
    raise RuntimeError("Set environment variable FLASK_SECRET_KEY")

app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///auth.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
app.config["SESSION_COOKIE_HTTPONLY"] = True
app.config["SESSION_COOKIE_SAMESITE"] = "Lax"
# Aktifkan True saat aplikasi berjalan di HTTPS:
app.config["SESSION_COOKIE_SECURE"] = os.environ.get("HTTPS_ONLY") == "1"

db = SQLAlchemy(app)
login_manager = LoginManager(app)
login_manager.login_view = "login"
login_manager.login_message = "Silakan login untuk melanjutkan."
csrf = CSRFProtect(app)

class User(UserMixin, db.Model):
    __tablename__ = "users"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False, index=True)
    password_hash = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(20), nullable=False, default="user")

@login_manager.user_loader
def load_user(user_id):
    try:
        return db.session.get(User, int(user_id))
    except (TypeError, ValueError):
        return None

def role_required(*allowed_roles):
    def decorator(view):
        @wraps(view)
        @login_required
        def wrapped(*args, **kwargs):
            if current_user.role not in allowed_roles:
                abort(403)
            return view(*args, **kwargs)
        return wrapped
    return decorator
```

Variabel FLASK_SECRET_KEY harus diatur sebelum aplikasi dijalankan. Contoh pembuatan secret key dapat menggunakan Python secrets.token_hex(32). Simpan nilainya di environment variable, jangan commit ke Git.

```
# Buat secret secara lokal (jalankan sekali di terminal Python)
import secrets
print(secrets.token_hex(32))

# Windows PowerShell untuk sesi terminal saat ini
$env:FLASK_SECRET_KEY = "isi-dengan-secret-acak"

# macOS/Linux
export FLASK_SECRET_KEY="isi-dengan-secret-acak"
```

3.3 Route Registrasi

```
@app.route("/register", methods=["GET", "POST"])
def register():
```

```
    if current_user.is_authenticated:
        return redirect(url_for("dashboard"))

    if request.method == "POST":
        name = request.form.get("name", "").strip()
        email = request.form.get("email", "").strip().lower()
        password = request.form.get("password", "")
        confirm = request.form.get("confirm_password", "")

        if not name or not email or not password:
            flash("Semua field wajib diisi.", "danger")
        elif len(name) > 100 or len(email) > 120:
            flash("Input melebihi batas panjang.", "danger")
        elif len(password) < 10:
            flash("Password minimal 10 karakter.", "danger")
        elif password != confirm:
            flash("Konfirmasi password tidak sama.", "danger")
        else:
            user = User(
                name=name,
                email=email,
                password_hash=generate_password_hash(password),
                role="user"  # role publik selalu user
            )
            try:
                db.session.add(user)
                db.session.commit()
                flash("Registrasi berhasil. Silakan login.", "success")
                return redirect(url_for("login"))
            except IntegrityError:
                db.session.rollback()
                flash("Registrasi tidak dapat diproses. Periksa data Anda.", "danger")

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

3.4 Route Login dan Logout

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if current_user.is_authenticated:
        return redirect(url_for("dashboard"))

    if request.method == "POST":
        email = request.form.get("email", "").strip().lower()
        password = request.form.get("password", "")
        user = db.session.execute(
            db.select(User).where(User.email == email)
        ).scalar_one_or_none()

        if user and check_password_hash(user.password_hash, password):
            login_user(user)
            next_page = request.args.get("next")

            # Hindari open redirect: jangan menerima URL eksternal.
            if next_page and next_page.startswith("/") and not next_page.startswith("//"):
                return redirect(next_page)
            return redirect(url_for("dashboard"))

        flash("Email atau password tidak valid.", "danger")

    return render_template("login.html")

@app.route("/logout", methods=["POST"])
@login_required
def logout():
    logout_user()
    flash("Anda telah logout.", "info")
    return redirect(url_for("login"))
```

Route logout menggunakan POST dan dilindungi CSRF. Redirect parameter next perlu divalidasi agar tidak menjadi open redirect. Pada aplikasi yang memiliki proxy atau domain khusus, gunakan validasi URL lokal yang lebih kuat.

3.5 Route Dashboard dan Admin

```
@app.route("/dashboard")
@login_required
def dashboard():
    return render_template("dashboard.html")

@app.route("/admin")
@role_required("admin")
def admin():
    users = db.session.execute(
        db.select(User).order_by(User.name)
    ).scalars().all()
    return render_template("admin.html", users=users)

@app.route("/profile")
@login_required
def profile():
    # current_user adalah pemilik profil yang sedang login
    return render_template("profile.html", user=current_user)

@app.errorhandler(403)
def forbidden(error):
    return render_template("403.html"), 403
```

Jangan membuat route publik yang memungkinkan pengguna mengubah role menjadi admin. Pembuatan admin harus dilakukan melalui proses administratif tepercaya, misalnya perintah CLI khusus atau pengaturan awal yang dikendalikan operator.

3.6 Template Form dengan CSRF

```
<!-- templates/base.html -->
<!doctype html>
<html lang="id">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{% block title %}Portal Pengguna{% endblock %}</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
        rel="stylesheet">
</head>
<body class="bg-light">
<nav class="navbar navbar-expand navbar-dark bg-primary">
  <div class="container">
    <a class="navbar-brand" href="{{ url_for('dashboard') }}">Portal</a>
    {% if current_user.is_authenticated %}
      <span class="navbar-text text-white me-3">{{ current_user.name }}</span>
      <form method="post" action="{{ url_for('logout') }}">
        <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
        <button class="btn btn-outline-light btn-sm">Logout</button>
      </form>
    {% endif %}
  </div>
</nav>
<main class="container py-4">
  {% with messages = get_flashed_messages(with_categories=true) %}
    {% for category, message in messages %}
      <div class="alert alert-{{ category }}">{{ message }}</div>
    {% endfor %}
  {% endwith %}
  {% block content %}{% endblock %}
</main>
</body>
</html>
```

```
<!-- templates/register.html -->
{% extends "base.html" %}
{% block title %}Registrasi{% endblock %}
{% block content %}
<h1 class="h3">Buat Akun</h1>
<form method="post" class="card card-body">
  <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
  <label class="form-label" for="name">Nama</label>
  <input class="form-control mb-3" id="name" name="name" required maxlength="100">

  <label class="form-label" for="email">Email</label>
  <input class="form-control mb-3" id="email" name="email" type="email" required>

  <label class="form-label" for="password">Password (minimal 10 karakter)</label>
  <input class="form-control mb-3" id="password" name="password" type="password" required>

  <label class="form-label" for="confirm_password">Konfirmasi password</label>
  <input class="form-control mb-3" id="confirm_password" name="confirm_password"
         type="password" required>

  <button class="btn btn-primary">Daftar</button>
</form>
{% endblock %}
<!-- templates/login.html -->
{% extends "base.html" %}
{% block title %}Login{% endblock %}
{% block content %}
<h1 class="h3">Login</h1>
<form method="post" class="card card-body">
  <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
  <label class="form-label" for="email">Email</label>
  <input class="form-control mb-3" id="email" name="email" type="email" required>

  <label class="form-label" for="password">Password</label>
  <input class="form-control mb-3" id="password" name="password"
         type="password" required>

  <button class="btn btn-primary">Login</button>
  <p class="mt-3 mb-0">Belum punya akun? <a href="{{ url_for('register') }}">Registrasi</a></p>
</form>
{% endblock %}
<!-- templates/dashboard.html -->
{% extends "base.html" %}
{% block title %}Dashboard{% endblock %}
{% block content %}
<h1 class="h3">Dashboard</h1>
<p>Selamat datang, {{ current_user.name }}.</p>
<p>Role: <strong>{{ current_user.role }}</strong></p>
<a class="btn btn-outline-primary" href="{{ url_for('profile') }}">Profil Saya</a>
{% if current_user.role == "admin" %}
  <a class="btn btn-danger" href="{{ url_for('admin') }}">Admin Panel</a>
{% endif %}
{% endblock %}
```

Template boleh menyembunyikan tautan yang tidak relevan untuk pengalaman pengguna, tetapi keamanan tetap bergantung pada pemeriksaan role di backend.

3.7 Membuat Admin Awal Secara Terkendali

```
# Jalankan sekali dari Python shell atau script administratif tepercaya.
from app import app, db, User
from werkzeug.security import generate_password_hash

with app.app_context():
    db.create_all()
    admin_user = User(
        name="Administrator",
        email="admin@example.com",
```

```
        password_hash=generate_password_hash("GANTI-PASSWORD-SEGERA"),
        role="admin"
    )
    db.session.add(admin_user)
    db.session.commit()
```

Ini hanya contoh inisialisasi lokal. Jangan menyimpan password admin contoh di repository atau menggunakan password yang sama pada deployment. Buat admin melalui proses bootstrap aman, lalu minta penggantian kredensial awal.

3.8 Menjalankan Aplikasi

```
# Atur FLASK_SECRET_KEY terlebih dahulu, lalu:
python app.py

# Buka:
http://127.0.0.1:5000/register
http://127.0.0.1:5000/login
http://127.0.0.1:5000/dashboard
http://127.0.0.1:5000/admin
```

Untuk menjalankan contoh lengkap, tambahkan route pembuatan tabel pada konteks aplikasi sebelum app.run(), atau jalankan db.create_all() melalui script inisialisasi. Pada production, gunakan server WSGI dan nonaktifkan debug mode.

4. Hands-on Exercises

Latihan 1 — Password hashing

1. Buat script Python untuk menerima password dan menghasilkan hash dengan generate_password_hash().
2. Simpan hash ke variabel dan uji password benar serta salah dengan check_password_hash().
3. Bandingkan dua hash dari password yang sama dan diskusikan penggunaan salt.

Kriteria hasil: Password tidak disimpan sebagai plaintext dan verifikasi berjalan.

Latihan 2 — Registrasi pengguna

4. Buat model User dengan id, name, email, password_hash, dan role.
5. Buat route registrasi dan form HTML.
6. Validasi field wajib, format email, panjang password, dan konfirmasi password.
7. Tangani email duplikat menggunakan IntegrityError dan rollback.

Kriteria hasil: Pengguna dapat mendaftar dan data tersimpan dengan hash password.

Latihan 3 — Login, logout, dan session

8. Konfigurasikan LoginManager dan user_loader.
9. Buat route login dan logout.
10. Lindungi dashboard menggunakan login_required.
11. Uji akses dashboard sebagai guest, user login, dan setelah logout.

Kriteria hasil: Hanya pengguna yang login dapat membuka dashboard.

Latihan 4 — Role-Based Access Control

12. Tambahkan role user dan admin pada model.
13. Buat decorator role_required().
14. Lindungi route /admin agar hanya admin yang dapat mengakses.
15. Uji akses menggunakan dua akun dengan role berbeda.

Kriteria hasil: User biasa mendapat penolakan akses admin; admin dapat membuka panel.

Latihan 5 — CSRF dan pengujian keamanan dasar

16. Aktifkan CSRFProtect.
17. Tambahkan token CSRF pada seluruh form POST.
18. Uji submit form tanpa token dan amati penolakan.
19. Pastikan logout menggunakan POST, bukan GET.
20. Periksa bahwa registrasi publik tidak dapat menetapkan role admin.

Kriteria hasil: Form terlindungi CSRF dan tidak ada jalur publik untuk menaikkan privilege.

5. Mini Project — Secure User Portal

Bangun portal web dengan registrasi dan login pengguna, dashboard privat, profil pribadi, serta panel admin. Aplikasi menggunakan Flask, Flask-Login, SQLAlchemy, SQLite, Werkzeug password hashing, dan CSRF protection.

5.1 Aktor dan Hak Akses

Aktor	Hak akses
Guest	Melihat halaman publik, registrasi, dan login.
User	Login, melihat dashboard, melihat profil miliknya, logout.
Admin	Seluruh akses user serta melihat daftar akun pada panel admin.

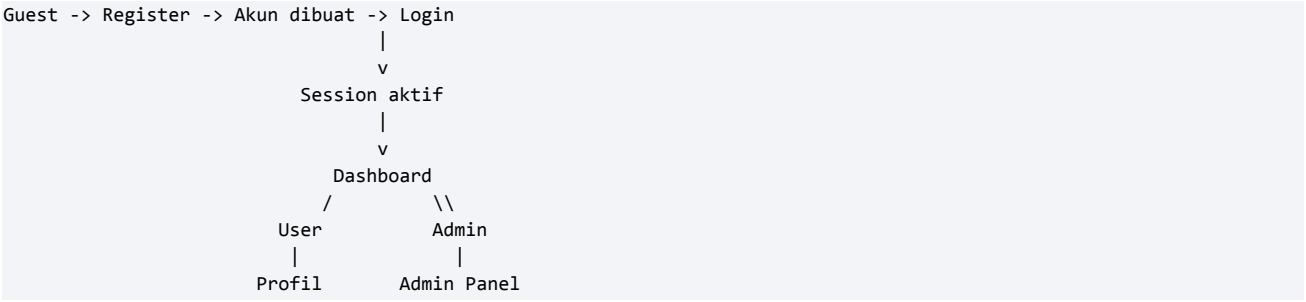
5.2 Kebutuhan Fungsional

- Registrasi dengan nama, email, password, dan konfirmasi password.
- Email pengguna harus unik.
- Password disimpan sebagai hash.
- Login dengan verifikasi password hash.
- Logout untuk mengakhiri sesi.
- Dashboard hanya dapat dibuka oleh pengguna login.
- Halaman profil hanya menampilkan data pengguna yang sedang login.
- Panel admin hanya dapat diakses role admin.
- Semua form POST dilindungi token CSRF.
- Pesan error tidak membocorkan apakah suatu email terdaftar.

5.3 Rancangan Database

Kolom	Tipe	Aturan
id	Integer	Primary key
name	String(100)	NOT NULL
email	String(120)	NOT NULL, UNIQUE, indexed
password_hash	String(255)	NOT NULL
role	String(20)	NOT NULL, default user

5.4 Alur Pengguna



```
                          \              /
                           ---- Logout ----
                                  |
                                  v
                            Session berakhir
```

5.5 Tahapan Pengerjaan

21. Buat virtual environment dan instal semua dependensi.
22. Siapkan konfigurasi Flask, secret key, database, LoginManager, dan CSRFProtect.
23. Buat model User dan inisialisasi database.
24. Implementasikan registrasi dengan password hashing dan validasi.
25. Implementasikan login, logout, user_loader, dan dashboard privat.
26. Buat decorator role_required dan panel admin.
27. Buat template dengan Bootstrap dan token CSRF pada semua form POST.
28. Buat akun admin melalui proses inisialisasi yang tepercaya.
29. Uji akses guest, user, dan admin; dokumentasikan hasil pengujian.

5.6 Skenario Pengujian

No.	Skenario	Aksi	Hasil yang Diharapkan
1	Registrasi valid	Daftar dengan email baru	Akun tersimpan; password berupa hash.
2	Email duplikat	Daftar dengan email terpakai	Registrasi ditolak dengan pesan aman.
3	Password salah	Login dengan password keliru	Login ditolak.
4	Dashboard guest	Akses /dashboard tanpa login	Dialihkan ke halaman login.
5	Dashboard user	Login sebagai user lalu buka dashboard	Dashboard dapat diakses.
6	Admin panel sebagai user	Akses /admin sebagai user	Akses ditolak (403).
7	Admin panel sebagai admin	Akses /admin sebagai admin	Panel admin dapat dibuka.
8	Logout	Kirim form logout	Sesi berakhir; dashboard tidak dapat diakses.
9	CSRF invalid	Kirim POST tanpa token	Request ditolak.
10	Privilege escalation	Coba mengirim role=admin saat registrasi	Role tetap user.

5.7 Rubrik Penilaian

Aspek	Bobot	Indikator
Model dan penyimpanan kredensial	20%	Model tepat, password di-hash, email unik.
Authentication	20%	Registrasi, login, logout, dan session berfungsi.
Authorization/RBAC	20%	Route privat dan admin terlindungi di backend.
Keamanan form dan konfigurasi	20%	CSRF, secret key, validasi, dan pesan aman.
UI dan usability	10%	Form dan dashboard mudah digunakan.
Pengujian dan dokumentasi	10%	Skenario guest/user/admin diuji dan dicatat.

Checklist Kompetensi

☐ Saya dapat membedakan authentication dan authorization.

☐ Saya dapat membuat hash password dan memverifikasinya.

☐ Saya dapat membuat registrasi pengguna.

☐ Saya dapat mengimplementasikan login, logout, dan session.

☐ Saya dapat melindungi route dengan login_required.

☐ Saya dapat menerapkan role-based access control.

☐ Saya dapat mengaktifkan CSRF protection pada form.

☐ Saya dapat mencegah pengguna publik menetapkan role admin.

☐ Saya dapat menguji akses guest, user, dan admin.

Glosarium

Istilah	Pengertian
Authentication	Proses memverifikasi identitas pengguna.
Authorization	Proses menentukan izin akses pengguna.
Password hashing	Mengubah password menjadi representasi hash satu arah untuk penyimpanan.
Session	Mekanisme mempertahankan status pengguna antar-request.
Cookie	Data kecil yang disimpan browser dan dikirim bersama request sesuai aturan cookie.
RBAC	Model kontrol akses berdasarkan peran pengguna.
Least privilege	Prinsip memberi hak akses minimum yang diperlukan.
CSRF	Serangan yang memanfaatkan sesi browser untuk mengirim request tanpa kehendak pengguna.
Decorator	Fungsi Python yang membungkus atau memodifikasi perilaku fungsi lain.
Privilege escalation	Kondisi ketika pengguna memperoleh hak akses lebih tinggi dari yang semestinya.

Referensi Belajar

- Flask Documentation — https://flask.palletsprojects.com/
- Flask-Login Documentation — https://flask-login.readthedocs.io/
- Flask-WTF Documentation — https://flask-wtf.readthedocs.io/
- Werkzeug Security Utilities — https://werkzeug.palletsprojects.com/en/stable/utils/
- Flask-SQLAlchemy Documentation — https://flask-sqlalchemy.palletsprojects.com/
- OWASP Authentication Cheat Sheet — https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- OWASP Authorization Cheat Sheet — https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Catatan Pengajar

Alokasi yang disarankan: 2 × 50 menit untuk teori dan demonstrasi, 2 × 50 menit untuk praktik autentikasi dan otorisasi, serta mini project sebagai tugas individu atau kelompok kecil. Penilaian sebaiknya mencakup demonstrasi login, pengujian akses berbasis role, pemeriksaan hash password, dan pengujian CSRF. Untuk materi lanjutan, peserta dapat menambahkan reset password melalui token sekali pakai, verifikasi email, rate limiting, multi-factor authentication, audit log, permission granular, dan deployment HTTPS.