**LAPORAN UJIAN AKHIR SEMESTER**

**PEMOGRAMAN ANALISIS DATA**



Disusun oleh:

## Nama : Sahril Nanda Putra

Nim/kelas : 225314056 / B

**PROGRAM STUDI INFORMATIKA FAKULTAS SAINS DAN TEKNOLOGI UNIVERSITAS SANATA DHARMA SEMESTER GASAL 2022/2023**

1. **Penjelasan**
2. Pengenalan aplikasi

Rent Ride merupakan aplikasi berbasis website yang memungkinkan penggunanya untuk melakukan pemesanan motor secara online, dengan desain aplikasi yang modern dan juga kenyamanan dalam melakukan navigasi saya yakin RentRide dapat menjadi sebuah platform yang bagus untuk dikembangkan lebih lanjut, RentRide sendiri memiliki beberapa fitur aplikasi diantaranya :

* **Registrasi dan juga Login**, pada aplikasi ini, kami memungkinkan pengguna agar dapat membuat akun baru dan masuk menjelajahi aplikasi kami, dan juga pengguna yang sudah terdaftar dapat langsung melakukan login untuk menjelajahi seluruh aplikasi kami.
* **Daftar dan filter motor**, pada aplikasi ini juga kami telah menyediakan fungsi untuk membangun daftar motor yang tersedia, dan juga kami telah menerapkan fitur filter pada halaman motor yang mana dnegan adanya hal ini saya kira dapat mempermudah pengguna dalam mencari dengan efisien
* **Detail motor**, Pengguna dapat melihat detail lengkap tentang motor tertentu, termasuk nama, deskripsi, harga, status, dan juga gambar, hal ini akan memantu para pengguna dalam menentukan pilihan dan mengambil keputusan motor mana yang ingin dia gunakan,
* **Form Pemesanan**, pengguna dapat mengisi informasi form pemesanan seperti nama, tujuan, alamat, metode pembayaran, jumlah hari pemesanan, dan juga opsi pengantaran
* **Pembayaran**, Pengguna dapat melakukan pembayaran menggunakan metode pembayaran yang dipilih dan memsukan nominal yang sesaui dengan harga total
* **Informasi Profile**, pengguna dapat melihat ringkasan pemesananya yang pernah dilakukan
* **Analisa pengguna,** pengguna diberikan informasi mengenai analisis penggunaan pengguna pada platform, seperti rata-rata durasi pemakaina, total pengeluaran pengguna, dan juga pemesanan terakhri yang pengguna lakukan
* **Logout,** fitur ini memungkinkan pengguna untuk keluar dari akun mereka.

1. Teknologi yang digunakan

* Flask: Framework Python yang digunakan untuk membangun aplikasi web.
* SQLite: Database relasional yang digunakan untuk menyimpan data pengguna, data motor, dan data pemesanan.
* Pandas: Library Python untuk analisis dan manipulasi data, digunakan untuk menganalisis data pemesanan.
* HTML/CSS: Bahasa pemrograman yang digunakan untuk mengatur tampilan dan gaya aplikasi web.
* Jinja2: Mesin template Python yang digunakan untuk menghubungkan data dari server Flask ke tampilan HTML.

1. Keuntungan penggunaan aplikasi

* Kemudahan Pemesanan: Pengguna dapat dengan mudah memesan motor secara online melalui aplikasi tanpa harus datang langsung ke tempat persewaan.
* Pilihan Motor yang Lengkap: Aplikasi menyediakan daftar motor yang lengkap dengan detail dan gambar, sehingga pengguna dapat memilih motor yang sesuai dengan kebutuhan mereka.
* Pengalaman Pengguna yang Baik: Antarmuka pengguna yang sederhana dan responsif membuat penggunaan aplikasi lebih nyaman dan mudah dipahami.
* Histori Pemesanan: Pengguna dapat melihat riwayat pemesanan mereka, sehingga memudahkan mereka untuk melacak dan mengelola pemesanan sebelumnya.

1. Tantangan membangun aplikasi

Aplikasi ini memiliki kompleksitas yang cukup bagi saya, sering kali saya mendapati kendala dalam membangun aplikasi ini, dan tantangan terberat yang sering saya hadapi adalah :

* Pengelolaan Database, konsistensi dalam pengguna database, dikarenakan dalam aplikasi ini setiap pengguna baru akan dibuatkan database sendiri agar memungkinkan data dari setiap pengguna berbeda-beda, dan saya sangat kesulitan pada hal ini, sering kali saya mendapati error saat aplikasi dijalankan
* Pengoptimalan Kinerja, memastikan aplikasi dapat berjalan dengan cepat dan juga responsif dalam pengiriman data yang diberikan.
* Pengujian dan Debugging, hal yang paling membosankan menurut saya, karena saya harus memastikan aplikasi dapat berjalan dengan baik, dan mencari bug yang tersembunyi dalam aplikasi yang mungkin saja dapat mempengaruhi aplikasi saat berjalan nantinya
* UI/UX Design, hal yang paling menangtang dan paling sulit adalah membangun konsitensi dalam pengembangan UI/UX yang mana pada tahapan ini saya harus membuat antarmuka yang konsisten dan membuat para pengguna dapat berinteraksi dengan mudah dan nyaman dalam berselancar dengan aplikasi ini.

1. **Struktur file dan aplikasi**

A picture containing screenshot, text

Description automatically generatedTerdapat struktur aplikasi dalam pengembangan aplikasi ini

1. **Folder static**, merupakan tempat penyimpanan file-file statis seperti gambar, file CSS, dan file JavaScript yang digunakan dalam aplikasi.
2. **Folder** **templates**, berisi file-template HTML yang digunakan untuk membuat tampilan halaman web aplikasi.
3. **Folder** **myenv**, adalah folder lingkungan virtual atau virtual environment yang digunakan dalam pengembangan aplikasi ini.
4. **File app.py,** merupakan file utama yang berisi kode-kode Python untuk mengatur logika aplikasi, seperti routing URL, pengolahan data, dan interaksi dengan database.
5. **File Motors.py dan User.py ,** merupakan file yang berisi perintah untuk membuat database baru dari motors dan database user
6. **file database seperti user.db, motor.db dan oreder\_.db ,** merupakanfile database SQLite yang menyimpan data pengguna, data motor, dan data pemesanan.
7. **Folder vsCode,** adalah folder yang berisi konfigurasi dan pengaturan untuk editor kode Visual Studio Code (VS Code).

1. **Code python dan penjelasan ringkas**

|  |  |
| --- | --- |
| App.py | penjelasan |
| from flask import Flask, render\_template, request, session, redirect, url\_for, g | Mengimport modul-modul dari libary flask yang sesuai kebutuhan untuk digunakan nantinya, libary yang diimport :  render template, yang berguna untuk merender template html. request yang berguna untuk mengirim dan menerima form.  session yang berguna utnuk mendapatkan dan menyimpan data sesi pengguna.  redirect dan url\_for yang berguna untuk mengarahkan pengguna ke halaman lain.  g yang berguna untuk menyimpan objek global yang dapat diakses di berbagai bagian aplikasi Flask. |
| import sqlite3  import pandas as pd  import os | Mengimport modul-modul dari libary sqlite3, pandas, dan os yang sesuai kebutuhan untuk digunakan nantinya.  sqlite3 digunakan untuk menghubungkan dan berinteraksi dengan database SQLite.  pandas digunakan untuk melakukan manipulasi dan analisis data, terutama dengan menggunakan struktur data DataFrame.  os digunakan untuk berinteraksi dengan sistem operasi, seperti mengakses path file, menghasilkan random string, dan lainnya |
| app = Flask(\_\_name\_\_) | Membuat instance objek Flask dengan nama app. \_\_name\_\_ digunakan sebagai argumen untuk menentukan root package aplikasi Flask. |
| def generate\_secret\_key():      return os.urandom(16).hex() | Mendefinisikan fungsi generate\_secret\_key() yang digunakan untuk menghasilkan secret key secara acak. Fungsi ini menggunakan os.urandom(16) untuk menghasilkan 16 byte data acak dan kemudian mengonversinya menjadi string heksadesimal menggunakan .hex() sebelum mengembalikan nilainya. Secret key ini digunakan untuk keperluan enkripsi dan keamanan dalam aplikasi Flask. |
| # Set secret key  app.secret\_key = generate\_secret\_key() | Mengatur secret key untuk aplikasi Flask dengan menggunakan fungsi generate\_secret\_key(). Nilai secret key yang dihasilkan oleh fungsi tersebut akan disimpan di atribut secret\_key dari objek app. Secret key ini digunakan untuk melakukan enkripsi data sesi dan keamanan lainnya dalam aplikasi Flask. |
| def generetader\_motor\_db():      conn = sqlite3.connect('motors.db')      cursor = conn.cursor()        # Membuat tabel motor      cursor.execute('''CREATE TABLE IF NOT EXISTS motor (                      id INTEGER PRIMARY KEY AUTOINCREMENT,                      nama TEXT NOT NULL,                      harga INTEGER NOT NULL,                      status TEXT NOT NULL,                      tipe TEXT NOT NULL,                      desc TEXT NOT NULL,                      gambar TEXT NOT NULL                  ''')      # Menambahkan data motor      motor\_data = [      ('Nmax', 150000, 'Tersedia', 'matic','Yamaha Nmax-155cc', 'nmax.png'),      ('Aerox', 180000, 'Tersedia', 'matic','Yamaha Aerox-155cc','aerox.png'),      ('Duke', 200000, 'Tersedia', 'kopling','KTM Duke-250cc','duke.png'),      ('CBR', 200000, 'Tersedia', 'kopling','Honda CBR-150cc','cbr.png'),      ('Wave', 50000, 'Tersedia', 'bebek','Honda Wave-120cc','wave.png'),      ('Revo', 60000, 'Tersedia', 'bebek','Honda Revo-110cc','revo.png'),      ]      cursor.executemany('INSERT INTO motor (nama, harga, status, tipe, desc, gambar) VALUES (?, ?, ?, ?, ?, ?)', motor\_data)      # Commit perubahan dan menutup koneksi      conn.commit()      conn.close() | Fungsi generetader\_motor\_db() digunakan untuk menghasilkan dan mengisi data dalam tabel "motor" dalam database "motors.db"  Fungsi ini awalnya akan Membuka koneksi ke database "motors.db" menggunakan modul sqlite3, lalu Membuat tabel "motor" jika belum ada selanjutnya Menambahkan data motor ke tabel "motor" menggunakan perintah cursor.executemany() lalu Melakukan commit perubahan ke database dan menutup koneksi.  Dengan menjalankan fungsi **generetader\_motor\_db()**, tabel "motor" akan terbentuk dengan skema yang telah ditentukan, dan data motor akan ditambahkan ke dalamnya. |
| def generated\_user\_db():      # Membuat koneksi ke database      conn = sqlite3.connect('users.db')      cursor = conn.cursor()      # Membuat tabel motor      cursor.execute('''CREATE TABLE IF NOT EXISTS users (                          id INTEGER PRIMARY KEY AUTOINCREMENT,                          email TEXT NOT NULL,                          username TEXT NOT NULL,                          password TEXT NOT NULL                      )''')      # Commit perubahan dan menutup koneksi      conn.commit()      conn.close() | Pada fungsi ini Fungsi **generated\_user\_db()** akan digunakan untuk menghasilkan dan menginisialisasi tabel "users" dalam database "users.db". Program akan Membuka koneksi ke database "users.db" menggunakan modul **sqlite3**. Dan membuat tabel "users" jika belum ada. Tabel ini memiliki beberapa kolom, yaitu "id" (sebagai primary key), "email", "username", dan "password". Lalu Melakukan commit perubahan ke database dan menutup koneksi. |
| # Mengecek file users.db  if not os.path.exists('users.db'):      print('membuat table user')       # Jika belum ada maka akan membuat table databse baru      generated\_user\_db()  # Mengecek file users.db  if not os.path.exists('motors.db'):      print('membuat table motor')      # Jika belum ada maka akan membuat table databse baru      generetader\_motor\_db() | melakukan pengecekan apakah file 'users.db' dan 'motors.db' sudah ada di direktori saat ini. Jika file-file tersebut tidak ada, maka akan dicetak pesan "membuat table user" atau "membuat table motor" tergantung pada file yang tidak ada. Setelah itu, fungsi **generated\_user\_db()** atau **generetader\_motor\_db()** akan dipanggil untuk membuat dan menginisialisasi tabel yang sesuai di database masing-masing. |
| def get\_user\_db():      if 'user\_db' not in g:          g.user\_db = sqlite3.connect('users.db')      return g.user\_db | mendapatkan koneksi ke database pengguna (**users.db**). Fungsi ini menggunakan objek **g** yang disediakan oleh Flask untuk menyimpan objek koneksi agar dapat digunakan kembali tanpa perlu membuat koneksi baru setiap kali fungsi ini dipanggil.  Pada fungsi ini, terdapat pengecekan apakah objek koneksi **user\_db** sudah ada dalam **g**. Jika belum ada, maka fungsi ini akan membuat koneksi baru ke database **users.db** menggunakan modul **sqlite3** dan menyimpannya dalam **g.user\_db**. Jika objek koneksi sudah ada sebelumnya, maka fungsi akan mengembalikan objek koneksi yang sudah ada tanpa membuat yang baru. |
| def get\_order\_db(user\_id):      if 'order\_db' not in g:          g.order\_db = sqlite3.connect(f'order\_{user\_id}.db')      return g.order\_db | Pada fungsi ini, terdapat pengecekan apakah objek koneksi **order\_db** dengan nama yang sesuai sudah ada dalam **g**. Jika belum ada, maka fungsi ini akan membuat koneksi baru ke database pemesanan yang terkait dengan pengguna tersebut menggunakan modul **sqlite3** dan menyimpannya dalam **g.order\_db**. Nama file database pemesanan didasarkan pada **user\_id**. Jika objek koneksi sudah ada sebelumnya, maka fungsi akan mengembalikan objek koneksi yang sudah ada tanpa membuat yang baru. |
| # Fungsi untuk membuat tabel pemesanan baru  def create\_order\_table(user\_id):      conn = get\_order\_db(user\_id)      cursor = conn.cursor()      cursor.execute('''CREATE TABLE IF NOT EXISTS orders (                          id INTEGER PRIMARY KEY AUTOINCREMENT,                          motor TEXT NOT NULL,                          alamat TEXT NOT NULL,                          haripemakaian INTEGER NOT NULL,                          total INTEGER NOT NULL                      )''')      conn.commit() | membuat tabel pemesanan baru dalam database pemesanan pengguna dengan user\_id yang diberikan. Pada fungsi ini, terlebih dahulu dilakukan pemanggilan fungsi **get\_order\_db(user\_id)** untuk mendapatkan objek koneksi ke database pemesanan pengguna. Setelah itu, dilakukan pembuatan objek cursor dari objek koneksi untuk menjalankan perintah SQL.  Perintah SQL yang dijalankan adalah **CREATE TABLE IF NOT EXISTS**, yang digunakan untuk membuat tabel baru dengan nama "orders" jika tabel tersebut belum ada. Setelah perintah SQL dieksekusi, dilakukan pemanggilan conn.commit() untuk menyimpan perubahan ke database. |
| def get\_motor\_connection():      conn = sqlite3.connect('motors.db')      conn.row\_factory = sqlite3.Row      return conn | **get\_motor\_connection()** digunakan untuk mendapatkan objek koneksi ke database motors.db yang berisi informasi mengenai motor-motor yang tersedia.  Pada fungsi ini, terlebih dahulu dilakukan pemanggilan **sqlite3.connect('motors.db')** untuk membuat objek koneksi ke database motors.db. Selanjutnya, **conn.row\_factory = sqlite3.Row** digunakan untuk mengatur row factory menjadi **sqlite3.Row**, sehingga hasil query yang diterima akan berupa objek **sqlite3.Row** yang memungkinkan kita mengakses kolom-kolom hasil query menggunakan nama kolom. |
| def get\_all\_motor():      conn = get\_motor\_connection()      cursor = conn.cursor()      cursor.execute('SELECT \* FROM motor')      motor = cursor.fetchall()      conn.close()      return motor | **get\_all\_motor()** digunakan untuk mendapatkan seluruh data motor dari database motors.db.  Pada fungsi ini, terlebih dahulu dipanggil get\_motor\_connection() untuk mendapatkan objek koneksi ke database motors.db. Selanjutnya, dibuat objek cursor menggunakan conn.cursor() untuk melakukan query pada database.  Query yang dilakukan adalah SELECT \* FROM motor, yang mengambil semua data dari tabel motor.  Setelah itu, cursor.fetchall() digunakan untuk mengambil seluruh baris hasil query sebagai list. Hasil ini kemudian disimpan dalam variabel motor. |
| def get\_current\_user():      user\_id = session.get('user\_id')      username = session.get('username')      if user\_id and username:          return {'id': user\_id, 'username': username}      return None | **get\_current\_user()** digunakan untuk mendapatkan informasi pengguna yang saat ini sedang login.  Pada fungsi ini, terlebih dahulu dilakukan pengambilan nilai user\_id dan username dari session menggunakan session.get('user\_id') dan session.get('username').  Selanjutnya, dilakukan pengecekan apakah user\_id dan username memiliki nilai (tidak None). Jika keduanya memiliki nilai, maka fungsi akan mengembalikan sebuah dictionary yang berisi informasi pengguna dengan kunci 'id' dan 'username'. Jika salah satu atau keduanya bernilai None, maka fungsi akan mengembalikan None. |
| def is\_logged\_in():      return 'user\_name' in session and session['user\_name'] != '' | Fungsi **is\_logged\_in()** digunakan untuk memeriksa apakah pengguna sedang dalam kondisi login.  Pada fungsi ini, dilakukan pengecekan terhadap session apakah terdapat kunci 'user\_name' dan nilai yang terkait tidak kosong (tidak sama dengan ''). Jika keduanya terpenuhi, berarti pengguna sedang dalam kondisi login dan fungsi mengembalikan nilai **True**. Jika salah satu atau keduanya tidak terpenuhi, fungsi mengembalikan nilai **False**. |
| def get\_orders\_data():      conn = get\_order\_db(session['user\_id'])      cursor = conn.cursor()      cursor.execute('SELECT \* FROM orders')      orders\_data = cursor.fetchall()      conn.close()      return orders\_data | **get\_orders\_data()** digunakan untuk mendapatkan data pemesanan pengguna yang sedang aktif.  Pada fungsi ini, terlebih dahulu dilakukan pengambilan koneksi ke database pemesanan pengguna menggunakan fungsi **get\_order\_db()** dengan parameter **session['user\_id']**. Selanjutnya, dilakukan eksekusi query untuk mengambil semua data pemesanan dari tabel "orders" menggunakan objek **cursor**. Hasil query kemudian diambil dengan menggunakan metode **fetchall()** dan disimpan dalam variabel **orders\_data**. Terakhir, koneksi ke database ditutup dan data pemesanan dikembalikan.  Fungsi ini berguna untuk mendapatkan data pemesanan pengguna yang dapat digunakan untuk ditampilkan atau diproses lebih lanjut dalam aplikasi. |
| def analyze\_orders():      orders\_data = get\_orders\_data()      if len(orders\_data) == 0:          return None, None, None      # Membuat DataFrame Pandas dari data pemesanan      df = pd.DataFrame(orders\_data, columns=['id', 'motor', 'alamat', 'haripemakaian', 'total'])      # Pemesanan Terakhir      order\_terakhir = df['motor'].iloc[-1]      avg\_pemakaian = df['haripemakaian'].mean()      ovr\_total =  df['total'].sum()      return order\_terakhir, avg\_pemakaian, ovr\_total | Fungsi **analyze\_orders()** digunakan untuk menganalisis data pemesanan yang ada.  Pada fungsi ini, pertama-tama data pemesanan diperoleh menggunakan fungsi get\_orders\_data(), dan disimpan dalam variabel orders\_data. Kemudian, dilakukan pengecekan apakah data pemesanan kosong atau tidak. Jika data kosong, fungsi ini akan mengembalikan None untuk semua nilai analisis.  Jika terdapat data pemesanan, langkah selanjutnya adalah membuat DataFrame Pandas dari data tersebut. Kolom-kolom DataFrame akan disesuaikan dengan nama kolom dalam tabel "orders" yaitu 'id', 'motor', 'alamat', 'haripemakaian', dan 'total'.  Setelah DataFrame dibuat, dilakukan analisis terhadap data pemesanan |
| @app.route('/')  def index():      return render\_template('index.html') | Fungsi index() merupakan route handler untuk URL utama ("/"). Ketika pengguna mengakses URL tersebut, fungsi ini akan dipanggil.  Di dalam fungsi index(), dilakukan rendering template HTML menggunakan render\_template(). Template yang digunakan adalah "index.html". Hasil rendering akan dikirimkan sebagai respons ke pengguna.  Dengan demikian, ketika pengguna mengakses URL utama, halaman "index.html" akan ditampilkan. |
| @app.route('/logout', methods=['POST'])  def logout():      # Hapus informasi pengguna dari sesi Flask      session.pop('user\_id', None)      session.pop('user\_name', None)      session.pop('user\_email', None)      return redirect(url\_for('index')) | Fungsi **logout()** adalah *route handler* untuk URL "/logout" dengan metode *POST*. Ketika pengguna melakukan permintaan *POST* ke URL tersebut, fungsi ini akan dipanggil.  Di dalam fungsi **logout()**, informasi pengguna (user\_id, user\_name, user\_email) dihapus dari sesi Flask menggunakan **session.pop()**. Setelah itu, pengguna akan diarahkan (redirect) kembali ke halaman utama menggunakan **redirect(url\_for('index'))**.  Dengan ini, ketika pengguna melakukan *logout* dengan mengirimkan permintaan *POST* ke URL "/logout", informasi pengguna akan dihapus dari sesi Flask dan pengguna akan diarahkan kembali ke halaman utama. |
| @app.route('/login')  def login():      return render\_template('login.html') | Fungsi **login()** adalah *route handler* untuk URL "/login". Ketika pengguna mengakses URL tersebut, fungsi ini akan dipanggil.  Di dalam fungsi **login()**, halaman template "login.html" akan dirender menggunakan fungsi **render\_template()**. Halaman template ini akan digunakan untuk menampilkan formulir login kepada pengguna.  Dengan ini, ketika pengguna mengakses URL "/login", halaman login akan ditampilkan. |
| @app.route('/login', methods=["POST"])  def login\_validation():      conn = get\_user\_connection()      cursor = conn.cursor()      username = request.form.get('username')      password = request.form.get('password')      cursor.execute('SELECT id, username, email FROM users WHERE username = ? AND password = ?', (username, password))      user = cursor.fetchone()      if user is not None:          # Login berhasil, simpan informasi pengguna ke sesi Flask          session['user\_id'] = user['id']          session['user\_name'] = user['username']          session['user\_email'] = user['email']          user\_id = user['id']          create\_order\_table(user\_id)          return redirect(url\_for('profile'))      else:          return render\_template('login.html', message="Username atau password salah.") | Fungsi **login\_validation()** adalah *route handler* untuk URL "/login" dengan metode POST. Ketika pengguna mengirimkan formulir login, fungsi ini akan dipanggil untuk memvalidasi informasi login yang diberikan.  Di dalam fungsi **login\_validation()**, terdapat beberapa langkah validasi:   1. Terhubung ke database pengguna menggunakan **get\_user\_connection()** untuk menjalankan query. 2. Mengambil username dan password yang dikirimkan dari formulir menggunakan **request.form.get()**. 3. Menjalankan query untuk memeriksa apakah terdapat pengguna dengan username dan password yang sesuai di dalam database. 4. Jika pengguna ditemukan, login berhasil. Informasi pengguna (ID, nama, email) akan disimpan dalam sesi Flask menggunakan **session**. Selain itu, fungsi **create\_order\_table()** akan dipanggil untuk membuat tabel pemesanan khusus untuk pengguna yang sedang login. 5. Jika pengguna tidak ditemukan, halaman login akan dirender kembali dengan pesan kesalahan menggunakan **render\_template()**.   Jika login berhasil, pengguna akan diarahkan ke halaman "profile" menggunakan fungsi **redirect()** dan **url\_for()**.  Jika login gagal, halaman login akan dirender kembali dengan pesan kesalahan yang ditampilkan kepada pengguna.  Ini adalah langkah-langkah untuk memvalidasi login dan mengelola informasi pengguna di dalam sesi Flask. |
| @app.route('/daftar')  def daftar():      return render\_template('daftar.html') | Fungsi daftar() adalah route handler untuk URL "/daftar". Ketika pengguna mengakses URL tersebut, fungsi ini akan dipanggil untuk merender halaman "daftar.html".  Fungsi ini mengembalikan hasil dari render\_template(), yang akan merender halaman HTML "daftar.html" dan mengirimkannya sebagai respons kepada pengguna. |
| @app.route('/profile')  def profile():      order\_terakhir, avg\_pemakaian, ovr\_total = analyze\_orders()      return render\_template('profile.html',                             order\_terakhir=order\_terakhir,                             avg\_pemakaian=avg\_pemakaian,                             ovr\_total=ovr\_total) | Fungsi **profile()** adalah *route handler* untuk URL "/profile". Ketika pengguna mengakses URL tersebut, fungsi ini akan dipanggil untuk merender halaman "profile.html".  Fungsi ini memanggil fungsi **analyze\_orders()** untuk mendapatkan informasi analisis pemesanan. Kemudian, hasil analisis tersebut akan dikirimkan sebagai argumen saat merender halaman "profile.html" menggunakan fungsi **render\_template()**. Halaman "profile.html" akan menerima argumen **order\_terakhir**, **avg\_pemakaian**, dan **ovr\_total** yang dapat digunakan untuk menampilkan informasi pemesanan pada halaman tersebut. |
| @app.route('/account', methods=['POST'])  def account():      conn = get\_user\_connection()      cursor = conn.cursor()      email = request.form.get('email')      username = request.form.get('username')      password = request.form.get('password')      cursor.execute('SELECT id FROM users WHERE email = ?', (email,))      existing\_user = cursor.fetchone()      cursor.execute('SELECT id FROM users WHERE username = ?', (username,))      existing\_user\_username = cursor.fetchone()      if existing\_user is not None:          return render\_template('daftar.html', message="Email sudah terdaftar. Silakan gunakan email lain.")      elif existing\_user\_username is not None:          return render\_template('daftar.html', message="Username sudah digunakan. Silakan pilih username lain.")      else:          # Tambahkan pengguna baru ke database          cursor.execute('INSERT INTO users (email, username, password) VALUES (?, ?, ?)', (email, username, password))          conn.commit()          # Dapatkan ID pengguna baru dan buat tabel pemesanan          cursor.execute('SELECT id FROM users WHERE email = ?', (email,))          user\_id = cursor.fetchone()['id']          create\_order\_table(user\_id)          conn.close()          return render\_template('login.html', message="Akun Berhasil Dibuat. Silakan Login.") | Fungsi **account()** adalah *route handler* untuk URL "/account" dengan metode POST. Fungsi ini akan dipanggil ketika pengguna mengirimkan formulir pendaftaran pada halaman "daftar.html".  Fungsi ini melakukan validasi untuk memastikan email dan username yang diinputkan tidak ada di database. Jika email atau username sudah terdaftar, maka halaman "daftar.html" akan dirender kembali dengan pesan kesalahan yang sesuai.  Jika email dan username belum terdaftar, pengguna baru akan ditambahkan ke database menggunakan perintah SQL INSERT. Kemudian, ID pengguna baru akan diambil dari database dan digunakan untuk membuat tabel pemesanan dengan fungsi **create\_order\_table()**. Setelah itu, halaman "login.html" akan dirender dengan pesan sukses bahwa akun berhasil dibuat.  Seluruh operasi database dilakukan menggunakan objek koneksi dan *cursor* yang diperoleh dari fungsi **get\_user\_connection()**. |
| @app.route('/motors', defaults={'tipe':None})  @app.route('/motors/<tipe>')  def motors(tipe):      all\_motor = get\_all\_motor()      if tipe:          filtered\_motor = [m for m in all\_motor if m['tipe'] == tipe]          return render\_template('koleksi.html', motor=filtered\_motor)      elif tipe:          filtered\_motor = [m for m in all\_motor if m['nama'] == tipe]          return render\_template('koleksi.html', motor=filtered\_motor)      else:          return render\_template('koleksi.html', motor=all\_motor) | Fungsi motors() adalah route handler untuk URL "/motors" dan "/motors/<tipe>". Fungsi ini menerima parameter opsional tipe yang dapat berisi jenis tipe motor.  Fungsi ini pertama-tama memanggil get\_all\_motor() untuk mendapatkan semua data motor dari database. Jika parameter tipe tidak ada, fungsi ini akan merender halaman "koleksi.html" dengan semua data motor yang diperoleh.  Jika parameter tipe ada, fungsi ini akan memfilter data motor berdasarkan jenis tipe yang diberikan. Jika ditemukan motor dengan tipe yang sesuai, data motor yang difilter akan dikirim ke halaman "koleksi.html" untuk dirender. |
| @app.route('/details', methods=['POST'])  def details():      motor\_nama = request.form.get('motor\_nama')      motor\_desc = request.form.get('motor\_desc')      motor\_harga = request.form.get('motor\_harga')      motor\_status = request.form.get('motor\_status')      motor\_gambar = request.form.get('motor\_gambar')      return render\_template('detail.html',motor\_nama=motor\_nama, motor\_harga=motor\_harga, motor\_desc=motor\_desc, motor\_status=motor\_status, motor\_gambar=motor\_gambar ) | Fungsi **details()** adalah *route handler* untuk URL "/details" dengan metode **POST**. Fungsi ini digunakan untuk menerima data motor yang dikirim melalui formulir pada halaman sebelumnya.  Fungsi ini mengambil nilai-nilai motor seperti **motor\_nama**, **motor\_desc**, **motor\_harga**, **motor\_status**, dan **motor\_gambar** dari permintaan formulir menggunakan **request.form.get()**. Nilai-nilai ini kemudian disimpan dalam variabel yang akan digunakan untuk merender halaman "detail.html".  Selanjutnya, fungsi ini merender halaman "detail.html" dan meneruskan nilai-nilai motor sebagai argumen untuk dirender di halaman tersebut.  Hal ini memungkinkan halaman "detail.html" untuk menampilkan informasi detail mengenai motor yang dipilih. |
| @app.route('/checkout', methods=["POST"])  def checkout():      if not is\_logged\_in():            return render\_template('index.html', message="Login Terlebih Dahulu sebelum memulai pemesanan")      motor\_nama = request.form.get('motor\_nama')      motor\_desc = request.form.get('motor\_desc')      motor\_harga = request.form.get('motor\_harga')      motor\_status = request.form.get('motor\_status')      motor\_gambar = request.form.get('motor\_gambar')      return render\_template('checkout.html', motor\_nama=motor\_nama, motor\_harga=motor\_harga, motor\_desc=motor\_desc, motor\_status=motor\_status, motor\_gambar=motor\_gambar ) | Fungsi **checkout()** adalah *route handler* untuk URL "/checkout" dengan metode **POST**. Fungsi ini digunakan untuk memproses pemesanan motor.  Pertama, fungsi ini memeriksa apakah pengguna telah login menggunakan fungsi **is\_logged\_in()**. Jika pengguna belum login, maka halaman "index.html" akan dirender kembali dengan pesan yang menunjukkan bahwa pengguna harus login terlebih dahulu sebelum memulai pemesanan.  Jika pengguna sudah login, fungsi ini akan mengambil nilai-nilai motor seperti **motor\_nama**, **motor\_desc**, **motor\_harga**, **motor\_status**, dan **motor\_gambar** dari permintaan formulir menggunakan **request.form.get()**. Nilai-nilai ini kemudian disimpan dalam variabel yang akan digunakan untuk merender halaman "checkout.html".  Selanjutnya, fungsi ini merender halaman "checkout.html" dan meneruskan nilai-nilai motor sebagai argumen untuk dirender di halaman tersebut. |
| @app.route('/payment', methods=['POST'])  def payment():      motor\_nama = request.form.get('motor\_nama')      motor\_desc = request.form.get('motor\_desc')      motor\_harga = int(request.form.get('motor\_harga'))      motor\_status = request.form.get('motor\_status')      motor\_gambar = request.form.get('motor\_gambar')      user\_nama = request.form.get('nama')      user\_alamat = request.form.get('alamat')      user\_hari = int(request.form.get('hari'))      user\_kota = request.form.get('kota')      user\_antar = 10000 if request.form.get('antar') == 'on' else 0      user\_pembayaran = request.form.get('pembayaran')      total = motor\_harga \* user\_hari + user\_antar      user\_id = session.get('user\_id')      # Menghubungkan ke database pemesanan pengguna      conn\_order = get\_order\_db(user\_id)      # Membuat tabel pemesanan baru jika belum ada      create\_order\_table(user\_id)      cursor\_order = conn\_order.cursor()      cursor\_order.execute('''INSERT INTO orders (motor, alamat, haripemakaian, total)                             VALUES (?, ?, ?, ?)''', (motor\_nama, user\_alamat, user\_hari, total))      conn\_order.commit()      return render\_template('payment.html', user\_antar=user\_antar,user\_nama=user\_nama, user\_alamat=user\_alamat, user\_hari=user\_hari,user\_kota=user\_kota,user\_pembayaran=user\_pembayaran, total=total ) | Fungsi **payment()** adalah *route handler* untuk URL "/payment" dengan metode **POST**. Fungsi ini digunakan untuk memproses pembayaran dari pemesanan motor.  Pertama, fungsi ini mengambil nilai-nilai motor seperti **motor\_nama**, **motor\_desc**, **motor\_harga**, **motor\_status**, dan **motor\_gambar** dari permintaan formulir menggunakan **request.form.get()**. Nilai-nilai ini kemudian disimpan dalam variabel yang akan digunakan untuk merender halaman "payment.html".  Selanjutnya, fungsi ini mengambil nilai-nilai pengguna seperti **user\_nama**, **user\_alamat**, **user\_hari**, **user\_kota**, **user\_antar**, dan **user\_pembayaran** dari permintaan formulir menggunakan **request.form.get()**. Beberapa nilai seperti **user\_hari** diubah menjadi tipe data integer menggunakan fungsi **int()**.  Selanjutnya, fungsi ini menghitung total pembayaran dengan mengalikan harga motor per hari dengan jumlah hari peminjaman dan menambahkan biaya antar jika dipilih.  Setelah itu, fungsi ini mendapatkan **user\_id** dari sesi Flask menggunakan **session.get('user\_id')**.  Kemudian, fungsi ini menghubungkan ke database pemesanan pengguna menggunakan **get\_order\_db(user\_id)** dan membuat tabel pemesanan baru jika belum ada menggunakan **create\_order\_table(user\_id)**.  Selanjutnya, fungsi ini memasukkan data pemesanan ke dalam tabel pemesanan menggunakan **INSERT INTO orders** dan **cursor\_order.execute()**. Data yang dimasukkan meliputi **motor\_nama**, **user\_alamat**, **user\_hari**, dan **total**. Setelah itu, perubahan di-commit ke database menggunakan **conn\_order.commit()**.  Terakhir, fungsi ini merender halaman "payment.html" dan meneruskan nilai-nilai yang diperlukan seperti **user\_antar**, **user\_nama**, **user\_alamat**, **user\_hari**, **user\_kota**, **user\_pembayaran**, dan **total** untuk dirender di halaman tersebut. |
| @app.route('/sukses', methods=['POST'])  def sukses():      nominal = int(request.form.get('nomnial'))      total = int(request.form.get('total'))      kembalian = nominal - total        return render\_template('sukses.html', kembalian=kembalian) | Fungsi ini digunakan untuk menangani halaman sukses pembayaran, di mana pengguna memasukkan nominal pembayaran dan total yang harus dibayarkan.  Pertama, fungsi ini mengambil nilai **nominal** dan **total** dari permintaan formulir menggunakan **request.form.get()**. Nilai-nilai tersebut diubah menjadi tipe data integer menggunakan fungsi **int()**.  Selanjutnya, fungsi ini menghitung kembalian dengan mengurangi nilai **total** dari **nominal**. Kembalian ini akan digunakan untuk merender halaman "sukses.html".  Terakhir, fungsi ini merender halaman "sukses.html" dan meneruskan nilai **kembalian** untuk dirender di halaman tersebut. |
| @app.route('/about')  def about():      return render\_template('about.html') | Pada fungsi **about()**, menggunakan **render\_template()** untuk merender halaman HTML "about.html". Setelah halaman dirender, halaman tersebut akan ditampilkan kepada pengguna yang mengakses URL "/about". |
| if \_\_name\_\_ == '\_\_main\_\_':      app.run(debug=True) | Kode **if \_\_name\_\_ == '\_\_main\_\_':** digunakan untuk memastikan bahwa server Flask hanya akan dijalankan ketika file ini langsung dieksekusi (bukan diimpor sebagai modul oleh file lain).  Ketika kita menjalankan file ini secara langsung, **\_\_name\_\_** akan memiliki nilai **\_\_main\_\_**, dan baris kode di dalam blok **if** akan dieksekusi. Namun, jika file ini diimpor sebagai modul oleh file lain, **\_\_name\_\_** akan memiliki nilai yang berbeda (nama modul), dan baris kode di dalam blok **if** akan diabaikan. |

1. **Screenshot code dan cara menjalankannya**

**A screenshot of a website

Description automatically generated with medium confidenceHome**

**A screenshot of a computer

Description automatically generated with medium confidenceLogin**

**A screenshot of a login form

Description automatically generated with low confidenceDaftar**

**Profile**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Motor Koleksi**

**A screenshot of a website

Description automatically generated with medium confidence**

A picture containing text, tire, wheel, vehicle

Description automatically generated**Details**

**Checkout  
A screenshot of a computer

Description automatically generated with medium confidence**

**Payment**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Sukses**

**A screenshot of a website

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with medium confidenceProfile degan analisa data**

**Cara menjalankan aplikasi**

**A black screen with white text

Description automatically generated with low confidence**Jalankan perintah untuk mengistall pandas dan juga flask

**A picture containing text, screenshot, font, graphics

Description automatically generated**membuat table motor

A picture containing text, screenshot, font

Description automatically generatedmenjalankan aplikasi

server akan berjalan pada port <http://127.0.0.1:5000>