**PEMOGRAMAN BERORIENTASI OBJEK**

**LAPORAN TUGAS BESAR : SISTEM KASIR RESTORAN**

**BERBASIS OOP**

****

**Mata Kuliah : Pemograman Berorientasi Objek**

**Dosen : Prayitno, S.ST., M.T., Ph.D**

**Disusun oleh**

**Nama : Alfin Rozzaq Nirwana**

**NIM : 4.33.24.0.05**

**Kelas : TI-1A**

**PROGRAM STUDI D4-TEKNOLOGI REKAYASA KOMPUTER**

**JURUSAN TEKNIK ELEKTRO**

**POLITEKNIK NEGERI SEMARANG**

**2024/2025**

1. **Tujuan**

Aplikasi ini dikembangkan sebagai bagian dari penyelesaian tugas besar mata kuliah Pemrograman Berorientasi Objek. Melalui pengerjaan tugas ini, saya diharapkan mampu:

1. Merancang dan mengimplementasikan kelas-kelas (seperti KasirCRUD, Transaksi, dan model data Makanan, Minuman, Pelanggan) menggunakan prinsip-prinsip OOP untuk mengelola logika bisnis dan interaksi data secara terstruktur dan efisien.
2. Membangun backend RESTful API menggunakan framework **FastAPI** untuk menyediakan *endpoints* yang menangani operasi dasar CRUD (Create, Read, Update, Delete) pada data menu (makanan dan minuman) dan proses transaksi.
3. Merancang dan mengelola skema database relasional menggunakan **SQLite** yang diintegrasikan melalui **SQLAlchemy ORM**. Ini memungkinkan penyimpanan data entitas (menu, pelanggan, item pesanan) secara persisten dan terstruktur.
4. Mengembangkan antarmuka pengguna (frontend) yang interaktif dan terpisah menggunakan library **Streamlit**. Antarmuka ini memiliki dua peran utama: halaman admin untuk mengelola menu dan melihat riwayat transaksi, serta halaman kasir untuk membuat pesanan baru.
5. Mengintegrasikan frontend Streamlit dengan backend FastAPI, di mana frontend bertindak sebagai klien yang mengonsumsi API untuk menampilkan, menambah, mengubah, dan menghapus data, sehingga menciptakan arsitektur aplikasi *client-server* yang modular.
6. **Alat dan Bahan**

* **Perangkat Lunak :**
* Sistem Operasi Windows
* Python versi 3.13.2
* IDE atau Code Editor : Visual Studio Code
* Web Browser: Google Chrome, Microsoft Edge, atau Firefox untuk mengakses dan menguji aplikasi web.
* Terminal/Command Prompt: Untuk menjalankan server dan menginstal dependensi.
* Library Python :
* **FastAPI:** Framework modern dan berkinerja tinggi untuk membangun backend API.
* **Streamlit:** Library untuk membangun dan menyajikan antarmuka pengguna web interaktif dengan cepat.
* **SQLAlchemy:** Toolkit SQL dan Object-Relational Mapper (ORM) yang digunakan untuk berinteraksi dengan database SQLite dari dalam kode Python.
* **Uvicorn:** Server ASGI (Asynchronous Server Gateway Interface) yang digunakan untuk menjalankan aplikasi FastAPI.
* **Requests:** Library untuk membuat permintaan HTTP dari frontend (Streamlit) ke backend (FastAPI).
* **Perangkat Keras :**
* Komputer atau Laptop dengan spesifikasi yang memadai untuk menjalankan server backend FastAPI dan aplikasi frontend Streamlit secara bersamaan.
* **Bahan Pendukung :**
* Jobsheet 11 Mata Kuliah PBO.
* Materi perkuliahan terkait konsep Pemrograman Berorientasi Objek.
* Dokumentasi resmi dari library yang digunakan (FastAPI, Streamlit, SQLAlchemy).
* Koneksi internet (terutama untuk instalasi library).

1. **Dasar Teori**

Aplikasi Kasir Restoran ini dirancang sebagai sebuah sistem terdistribusi yang modern, memisahkan secara tegas antara logika bisnis di sisi server (backend) dengan antarmuka pengguna di sisi klien (frontend). Pendekatan ini, yang dibangun di atas fondasi kokoh dari paradigma *Object-Oriented Programming* (OOP), menghasilkan aplikasi yang tidak hanya fungsional tetapi juga modular, dapat diandalkan, dan mudah untuk dikembangkan lebih lanjut.

1. **Arsitektur Aplikasi: Decoupled Frontend & Backend**

Sistem ini mengimplementasikan arsitektur *client-server* yang sesungguhnya, di mana kedua komponen beroperasi secara independen dan berkomunikasi melalui protokol standar web (HTTP).

* **Backend (Server Side) dengan FastAPI:**

Inti dari seluruh sistem berada di backend yang dibangun menggunakan **FastAPI**. Komponen ini memiliki tanggung jawab tunggal: mengelola logika bisnis, memvalidasi data, dan berinteraksi secara aman dengan database. FastAPI mengekspos fungsionalitasnya melalui serangkaian *endpoint* RESTful API.

* 1. **Manajemen Menu:** Menyediakan endpoint seperti POST /makanan/, GET /makanan/, PUT /makanan/{id}, dan DELETE /makanan/{id} untuk operasi CRUD pada data makanan dan minuman.
  2. **Manajemen Transaksi:** Mengelola alur pesanan melalui endpoint seperti POST /pelanggan/ untuk membuat pesanan baru, PUT /pelanggan/{id} untuk memperbarui pesanan, dan POST /transaksi/selesai/ untuk menyelesaikan pembayaran.
  3. **Dependency Injection:** FastAPI menggunakan fitur *dependency injection* secara ekstensif, seperti pada db: Session = Depends(get\_db), untuk mengelola koneksi database secara efisien dan memastikan setiap permintaan API mendapatkan sesi database yang terisolasi.
* **Frontend (Client Side) dengan Streamlit:**

Antarmuka pengguna (UI) dibangun sepenuhnya menggunakan **Streamlit** dan berperan murni sebagai klien. Komponen ini tidak memiliki pengetahuan atau akses langsung ke database. Seluruh data yang ditampilkan dan dikirim berasal dari interaksi dengan backend API.

1. **Komunikasi via HTTP:** Frontend menggunakan library requests untuk berkomunikasi dengan backend. Sebagai contoh, saat admin menambahkan makanan baru, frontend.py akan mengeksekusi requests.post(f"{API\_URL}/makanan/", data=data, files=files) untuk mengirim informasi ke server FastAPI.
2. **Manajemen State:** Streamlit menggunakan st.session\_state untuk mengelola state di sisi klien, seperti siapa pengguna yang sedang login (st.session\_state.user), status admin (st.session\_state.is\_admin), dan isi keranjang belanja (st.session\_state.keranjang). Ini memungkinkan pengalaman pengguna yang interaktif dan persisten selama sesi mereka.
3. **Konsep OOP dalam Aplikasi**

Paradigma *Object-Oriented Programming* (OOP) adalah tulang punggung dari desain backend. OOP memungkinkan abstraksi entitas dunia nyata (seperti Makanan, Minuman, dan Transaksi) ke dalam objek perangkat lunak yang dapat dikelola.

* **Kelas Model Data (SQLAlchemy ORM di models.py)** Kelas-kelas seperti Makanan, Minuman, Pelanggan, dan PemesananItem berfungsi sebagai *blueprint* data. Kelas-kelas ini adalah representasi objek dari tabel-tabel di dalam database.
  + **Atribut:** Setiap kelas memiliki atribut yang merepresentasikan kolom pada tabel database. Contohnya, kelas Makanan memiliki atribut id, nama, harga, dan gambar.
  + **Object-Relational Mapping (ORM):** Ini adalah manifestasi utama dari prinsip **Abstraksi**. SQLAlchemy ORM menerjemahkan kelas-kelas dan objek Python ini menjadi perintah SQL yang dapat dimengerti oleh database. Pengembang dapat memanipulasi data menggunakan sintaks Python (misalnya, db.query(models.Pelanggan)) alih-alih menulis SQL mentah.
  + **Komposisi dan Relasi:** Prinsip **Komposisi** diwujudkan melalui relationship(). Pada kelas Pelanggan, items = relationship("PemesananItem", back\_populates="pelanggan") mendefinisikan bahwa sebuah objek Pelanggan "memiliki" (has-a) satu atau lebih objek PemesananItem. Ini secara elegan memodelkan hubungan *one-to-many* antara tabel pelanggan dan item pesanannya.
* **Kelas KasirCRUD (di crudController.py)** Kelas ini didedikasikan untuk mengelola logika CRUD dasar, memisahkan tanggung jawab manipulasi data dari logika API.
  + **Enkapsulasi:** Kelas ini membungkus (mengenkapsulasi) sesi database (self.db) beserta metode-metode yang beroperasi padanya (tambah\_makanan, update\_minuman, dll.). Detail implementasi, seperti self.db.add(), self.db.commit(), dan self.db.refresh(), tersembunyi dari luar, menyediakan *interface* yang bersih dan aman.
  + **Abstraksi:** Metode seperti update\_makanan menyediakan abstraksi tingkat tinggi. Metode ini menyederhanakan proses kompleks yang terdiri dari beberapa langkah: mencari data, memperbarui atribut, menghapus file gambar lama jika ada, dan menyimpan perubahan ke database. Semua ini dapat dipanggil dengan satu baris kode.
* **Kelas Transaksi (di kasir\_logic.py)** Kelas ini adalah orkestrator utama untuk semua logika yang terkait dengan proses transaksi pelanggan.
  + **Inheritance (Pewarisan):** class Transaksi(KasirCRUD): menunjukkan bahwa Transaksi mewarisi semua kemampuan CRUD dari kelas induknya. Ini adalah contoh klasik dari hubungan "is-a" yang memungkinkan penggunaan kembali kode dan perluasan fungsionalitas.
  + **Enkapsulasi & Pengelolaan State:** Kelas ini mengenkapsulasi seluruh alur kerja transaksi. Atribut-atributnya seperti nama\_pelanggan, list\_items, total\_keseluruhan, bayar, dan kembali merepresentasikan *state* (keadaan) dari sebuah transaksi pada satu waktu. Metode seperti tambah\_pesanan dan selesaikan\_transaksi memodifikasi state ini secara terkontrol.

1. **Penyimpanan Data dengan SQLAlchemy dan SQLite**

Persistensi data dijamin oleh database relasional yang kuat, yang dikelola melalui ORM.

* **Database dan Tabel:** Menggunakan **SQLite** yang disimpan dalam file tunggal kasir.db. Berdasarkan models.py, struktur tabelnya adalah sebagai berikut:
* **makanan**: Menyimpan id (Primary Key), nama, harga, dan gambar (nama file).
* **minuman**: Menyimpan id (Primary Key), nama, harga, dan gambar (nama file).
* **pelanggan**: Menyimpan data ringkasan transaksi, termasuk id (Primary Key), nama pelanggan, total\_harga, bayar, kembali, dan created\_at (timestamp).
* **pemesanan\_item**: Menyimpan detail setiap item yang dipesan. Tabel ini memiliki relasi *Foreign Key* (pelanggan\_id) ke tabel pelanggan.
* **Operasi Database Melalui ORM:** Alih-alih menggunakan SQL mentah, aplikasi ini memanfaatkan kekuatan SQLAlchemy ORM. Berikut adalah contoh pemetaan operasi SQL ke metode ORM yang digunakan dalam proyek:
  + **INSERT**: Direpresentasikan dengan membuat instance kelas model dan menambahkannya ke sesi: makanan = models.Makanan(...), self.db.add(makanan), self.db.commit().
  + **SELECT**: Dilakukan dengan metode query(): self.db.query(models.Makanan).all() untuk mengambil semua data, atau self.db.query(models.Minuman).filter(models.Minuman.id == id).first() untuk mengambil data spesifik berdasarkan ID.
  + **UPDATE**: Dilakukan dengan mengambil objek terlebih dahulu, mengubah atributnya (minuman.nama = nama), lalu melakukan commit pada sesi: self.db.commit().
  + **DELETE**: Dilakukan dengan mengambil objek yang akan dihapus, lalu memanggil self.db.delete(makanan) dan diakhiri dengan self.db.commit().

1. **Interface Web dengan Streamlit**

Aplikasi Manajemen Perusahaan ini dibangun menggunakan **Streamlit**, Aplikasi terbagi menjadi beberapa halaman (menu) dan mengandalkan berbagai widget input/output interaktif.

* **Widget Input:** Aplikasi memanfaatkan beragam widget input untuk interaktivitas pengguna:
* **st.text\_input:** Digunakan untuk memasukkan nama pengguna saat login, password admin, dan nama menu baru.
* **st.number\_input:** Untuk menentukan harga menu, jumlah pesanan di keranjang, dan nominal pembayaran.
* **st.file\_uploader:** Memungkinkan admin mengunggah gambar untuk menu makanan dan minuman.
* **st.button:** Tombol aksi utama seperti "Masuk", "Tambah ke Keranjang", "Selesaikan Transaksi", dan "Hapus" item.
* **st.expander:** Digunakan di halaman admin untuk menyembunyikan form edit dan hapus, membuat tampilan lebih rapi dan terorganisir.
* **Widget Output/Display:** Data dan informasi disajikan kepada pengguna melalui widget output:
* **st.image:** Menampilkan gambar menu makanan dan minuman yang diambil dari direktori statis backend.
* **st.success, st.error, st.warning:** Memberikan umpan balik visual yang jelas kepada pengguna, misalnya setelah transaksi berhasil atau jika input tidak valid.
* **st.tabs:** Mengorganisir halaman admin menjadi tiga bagian logis: "Riwayat Transaksi", "CRUD Makanan", dan "CRUD Minuman", yang meningkatkan usabilitas.
* **st.columns:** Digunakan untuk membuat tata letak yang responsif dan rapi, seperti menyandingkan gambar, nama menu, dan tombol tambah.

1. **Struktur Aplikasi**

struktur aplikasi dibagi menjadi beberapa halaman utama yang alurnya dikontrol oleh *session state* (st.session\_state.current\_page) berdasarkan status login dan aksi pengguna. Aplikasi ini memiliki dua alur utama: untuk **Admin** dan untuk **Pengguna/Kasir**.

* **Halaman Login (login\_page)**
  1. Merupakan halaman awal aplikasi tempat pengguna memasukkan identitas.
  2. Menggunakan st.text\_input untuk memasukkan nama. Jika nama yang dimasukkan adalah "admin", akan muncul input tambahan st.text\_input(type="password") untuk kata sandi.
  3. Tombol st.button ("Masuk" atau "Masuk sebagai Admin") akan mengarahkan pengguna ke halaman yang sesuai (Halaman Admin atau Halaman Pengguna).
* **Halaman Admin (admin\_page)**

1. Dasbor khusus yang hanya bisa diakses setelah login sebagai admin. Halaman ini menggunakan st.tabs untuk membagi fungsionalitas menjadi tiga bagian utama.
2. **Tab Riwayat Transaksi**:
   1. Menampilkan seluruh riwayat transaksi yang telah selesai.
   2. Setiap transaksi ditampilkan dengan nama pelanggan, total, bayar, dan kembali. Detail pesanan dapat dilihat menggunakan st.expander.
3. **Tab CRUD Makanan & Minuman**:
   1. Menyediakan fungsionalitas penuh untuk *Create, Read, Update, & Delete* (CRUD) pada menu.
   2. Setiap item yang ada ditampilkan dalam st.expander di mana admin bisa mengubah nama, harga, gambar, atau menghapusnya.
   3. Terdapat formulir terpisah untuk menambah menu baru, menggunakan st.text\_input, st.number\_input, dan st.file\_uploader.

* **Halaman Pengguna/Kasir (dashboard\_user\_page)**

1. Merupakan halaman utama bagi pengguna untuk membuat pesanan baru.
2. Menampilkan daftar menu makanan dan minuman yang diambil dari API, lengkap dengan gambar (st.image), nama, dan harga.
3. Setiap item memiliki tombol st.button "➕ Tambah" untuk memasukkan item ke dalam keranjang belanja (st.session\_state.keranjang).
4. Tata letak diatur secara rapi menggunakan st.columns.

* **Halaman Transaksi (transaksi\_page)**

1. Halaman *checkout* di mana pengguna memfinalisasi pesanan.
2. Menampilkan ringkasan pesanan dari keranjang. Pengguna dapat mengubah jumlah item menggunakan st.number\_input atau menghapusnya dengan tombol "❌".
3. Total belanja dihitung dan ditampilkan secara otomatis. Pengguna memasukkan nominal pembayaran melalui st.number\_input.
4. Tombol "Selesaikan Transaksi" mengirimkan data ke backend untuk diproses. Jika berhasil, halaman akan menampilkan ringkasan pembayaran (total, bayar, kembali) dan tombol untuk kembali ke halaman login.
5. **Langkah Pembuatan Aplikasi**

Proses pembuatan aplikasi ini dibagi menjadi beberapa langkah utama, dimulai dari persiapan database, pembangunan logika backend, pembuatan API, hingga pengembangan antarmuka pengguna (frontend).

**Langkah 1 : Setup Database dan Model Data**

Langkah pertama adalah mendefinisikan struktur dasar data dan bagaimana aplikasi akan terhubung ke database.

* + **File database.py:** File ini bertanggung jawab untuk mengonfigurasi koneksi ke database. Ini mendefinisikan DATABASE\_URL (lokasi file kasir.db), membuat engine SQLAlchemy, dan menyiapkan SessionLocal yang akan digunakan untuk setiap sesi interaksi dengan database.

|  |
| --- |
| 1. from sqlalchemy import create\_engine 2. from sqlalchemy.ext.declarative import declarative\_base 3. from sqlalchemy.orm import sessionmaker 4. DATABASE\_URL = "sqlite:///./kasir.db" 5. engine = create\_engine(DATABASE\_URL, connect\_args={"check\_same\_thread": False}) 6. SessionLocal = sessionmaker(bind=engine, autocommit=False, autoflush=False) 7. Base = declarative\_base() |

* + **File models.py:** File ini berisi cetak biru (blueprint) dari semua tabel yang akan ada di database. Setiap kelas (Makanan, Minuman, Pelanggan, PemesananItem) mewakili satu tabel dan atributnya adalah kolom-kolom tabel tersebut. SQLAlchemy akan menggunakan model-model ini untuk membuat tabel secara otomatis.

|  |
| --- |
| 1. from sqlalchemy import Column, Integer, String, ForeignKey, DateTime 2. from sqlalchemy.orm import relationship 3. from datetime import datetime 4. from app.database import Base 5. class Makanan(Base): 6. \_\_tablename\_\_ = "makanan" 7. id = Column(Integer, primary\_key=True, index=True) 8. nama = Column(String) 9. harga = Column(Integer) 10. gambar = Column(String) 11. class Minuman(Base): 12. \_\_tablename\_\_ = "minuman" 13. id = Column(Integer, primary\_key=True, index=True) 14. nama = Column(String) 15. harga = Column(Integer) 16. gambar = Column(String) 17. class Pelanggan(Base): 18. \_\_tablename\_\_ = "pelanggan" 19. id = Column(Integer, primary\_key=True, index=True) 20. nama = Column(String, nullable=False) 21. total\_harga = Column(Integer, default=0) 22. bayar = Column(Integer, default=0) 23. kembali = Column(Integer, default=0) 24. created\_at = Column(DateTime, default=datetime.now()) 25. items = relationship("PemesananItem", back\_populates="pelanggan") 26. class PemesananItem(Base): 27. \_\_tablename\_\_ = "pemesanan\_item" 28. id = Column(Integer, primary\_key=True, index=True) 29. pelanggan\_id = Column(Integer, ForeignKey("pelanggan.id"), nullable=False) 30. nama\_item = Column(String, nullable=False) 31. jumlah = Column(Integer, nullable=False) 32. harga\_satuan = Column(Integer, nullable=False) 33. total = Column(Integer, nullable=False) 34. pelanggan = relationship("Pelanggan", back\_populates="items") |

**Langkah 2 : Membangun Logika Bisnis Backend**

Setelah struktur data didefinisikan, langkah selanjutnya adalah membuat logika untuk mengelola data tersebut.

* + **File crudController.py:** Kelas KasirCRUD di dalam file ini berfungsi sebagai *controller* tingkat rendah yang menangani operasi dasar Create, Read, Update, dan Delete (CRUD) untuk data menu (makanan dan minuman). Ini memisahkan logika akses database dari logika bisnis utama.

|  |
| --- |
| 1. from sqlalchemy.orm import Session 2. import app.models as models 3. import os 4. class KasirCRUD: 5. def \_\_init\_\_(self, db: Session): 6. self.db = db 7. # === Makanan === 8. def tambah\_makanan(self, nama: str, harga: int, gambar: str): 9. makanan = models.Makanan(nama=nama, harga=harga, gambar=gambar) 10. self.db.add(makanan) 11. self.db.commit() 12. self.db.refresh(makanan) 13. return makanan 14. def semua\_makanan(self): 15. return self.db.query(models.Makanan).all() 16. def update\_makanan(self, id: int, nama: str, harga: int, gambar: str = None): 17. makanan = self.db.query(models.Makanan).filter(models.Makanan.id == id).first() 18. if makanan: 19. makanan.nama = nama 20. makanan.harga = harga 21. if gambar: 22. old\_path = f"app/static/images/makanan/{makanan.gambar}" 23. if os.path.exists(old\_path): 24. os.remove(old\_path) 25. makanan.gambar = gambar 26. self.db.commit() 27. return makanan 28. def hapus\_makanan(self, id: int): 29. makanan = self.db.query(models.Makanan).filter(models.Makanan.id == id).first() 30. if makanan: 31. self.db.delete(makanan) 32. self.db.commit() 33. # === Minuman === 34. def tambah\_minuman(self, nama: str, harga: int, gambar: str): 35. minuman = models.Minuman(nama=nama, harga=harga, gambar=gambar) 36. self.db.add(minuman) 37. self.db.commit() 38. self.db.refresh(minuman) 39. return minuman 40. def semua\_minuman(self): 41. return self.db.query(models.Minuman).all() 42. def update\_minuman(self, id: int, nama: str, harga: int, gambar: str = None): 43. minuman = self.db.query(models.Minuman).filter(models.Minuman.id == id).first() 44. if minuman: 45. minuman.nama = nama 46. minuman.harga = harga 47. if gambar: 48. old\_path = f"app/static/images/minuman/{minuman.gambar}" 49. if os.path.exists(old\_path): 50. os.remove(old\_path) 51. minuman.gambar = gambar 52. self.db.commit() 53. return minuman 54. def hapus\_minuman(self, id: int): 55. minuman = self.db.query(models.Minuman).filter(models.Minuman.id == id).first() 56. if minuman: 57. self.db.delete(minuman) 58. self.db.commit() |

* + **File kasir\_logic.py:** Kelas Transaksi pada file ini berisi logika bisnis tingkat tinggi. Kelas ini mengatur seluruh alur kerja sebuah transaksi, mulai dari menerima pesanan, menghitung total, hingga memfinalisasi pembayaran. Kelas ini mewarisi (inheritance) fungsionalitas dari KasirCRUD untuk berinteraksi dengan database.

|  |
| --- |
| 1. from app.crudController import KasirCRUD 2. from app.models import Pelanggan, PemesananItem 3. class Transaksi(KasirCRUD): 4. def \_\_init\_\_(self, db): 5. super().\_\_init\_\_(db) 6. self.pelanggan\_id = id 7. self.nama\_pelanggan = "" 8. self.list\_items = [] 9. self.list\_pemesanan = {} 10. self.total\_keseluruhan = 0 11. self.bayar = 0 12. self.kembali = 0 14. def tambah\_pesanan(self, nama: str, items: list[dict]): 15. total\_harga = 0 16. list\_objs = [] 17. for item in items: 18. jumlah = item["jumlah"] 19. harga = item["harga\_satuan"] 20. total = jumlah \* harga 21. total\_harga += total 23. item\_obj = PemesananItem( 24. nama\_item=item["nama\_item"], 25. jumlah=jumlah, 26. harga\_satuan=harga, 27. total=total 28. ) 29. list\_objs.append(item\_obj) 31. pelanggan = Pelanggan( 32. nama=nama, 33. total\_harga=total\_harga, 34. items=list\_objs 35. ) 36. self.nama\_pelanggan = nama 37. self.list\_items.append(list\_objs) 38. self.total\_keseluruhan = total\_harga 39. self.db.add(pelanggan) 40. self.db.commit() 41. self.db.refresh(pelanggan) 42. self.pelanggan\_id = pelanggan.id 43. def get\_semua\_pesanan(self): 44. return self.db.query(Pelanggan).all() 46. def update\_pesanan(self, pelanggan\_id: int, item\_updates: list[dict]): 47. pelanggan = self.db.query(Pelanggan).filter(Pelanggan.id == pelanggan\_id).first() 48. if not pelanggan: 49. return None 50. total\_baru = 0 51. for upd in item\_updates: 52. item = self.db.query(PemesananItem).filter( 53. PemesananItem.id == upd["id"], 54. PemesananItem.pelanggan\_id == pelanggan\_id 55. ).first() 56. if not item: 57. continue 58. if upd["jumlah"] == 0: 59. self.db.delete(item) 60. else: 61. item.jumlah = upd["jumlah"] 62. item.total = item.jumlah \* item.harga\_satuan 63. total\_baru += item.total 64. # Hitung ulang semua item yang masih ada 65. sisa\_items = self.db.query(PemesananItem).filter( 66. PemesananItem.pelanggan\_id == pelanggan\_id 67. ).all() 68. total\_baru = sum([i.total for i in sisa\_items]) 69. pelanggan.total\_harga = total\_baru 70. self.total\_keseluruhan = total\_baru 71. self.db.commit() 72. self.db.refresh(pelanggan) 73. return pelanggan 75. def selesaikan\_transaksi(self, bayar: int): 76. self.bayar = bayar 77. self.kembali = bayar - self.total\_keseluruhan 78. # Simpan juga ke DB 79. pelanggan = self.db.query(Pelanggan).filter(Pelanggan.id == self.pelanggan\_id).first() 80. if pelanggan: 81. pelanggan.bayar = self.bayar 82. pelanggan.kembali = self.kembali 83. self.db.commit() |

**Langkah 3 : Membangun API Server dengan FastAPI**

Dengan logika yang sudah siap, langkah ini adalah membuat *endpoint* API agar dunia luar (dalam hal ini, frontend) dapat berkomunikasi dengan logika backend.

* + - **File main.py:** Ini adalah file utama untuk server backend. Menggunakan **FastAPI**, file ini mendefinisikan semua alamat URL (misalnya, /makanan/, /transaksi/selesai/) dan menghubungkannya dengan fungsi atau metode yang sesuai dari KasirCRUD dan Transaksi. File ini juga bertanggung jawab untuk menangani unggahan file gambar.

|  |
| --- |
| 1. import os 2. import shutil 3. from typing import Optional 4. from fastapi import FastAPI, Depends, HTTPException, UploadFile, File, Form, Body 5. from fastapi.staticfiles import StaticFiles 6. from sqlalchemy.orm import Session 7. from app.database import SessionLocal, engine 8. import app.models as models 9. from app.crudController import KasirCRUD 10. from app.kasir\_logic import Transaksi 11. models.Base.metadata.create\_all(bind=engine) 12. app = FastAPI() 13. app.mount("/static", StaticFiles(directory="app/static"), name="static") 14. # Dependency untuk koneksi DB 15. def get\_db(): 16. db = SessionLocal() 17. try: 18. yield db 19. finally: 20. db.close() 21. ALLOWED\_EXTENSIONS = {".jpg", ".jpeg", ".png"} 22. # === ADMIN === 23. # === Makanan === 24. @app.post("/makanan/") 25. async def upload\_makanan(nama: str, harga: int, nama\_file: str, gambar: UploadFile = File(...), db: Session = Depends(get\_db)): 26. ext = os.path.splitext(gambar.filename)[1].lower() 27. if ext.lower() not in ALLOWED\_EXTENSIONS: 28. raise HTTPException(status\_code=400, detail="Format gambar tidak didukung.") 29. os.makedirs("app/static/images/makanan", exist\_ok=True) 30. filename = f"{nama\_file}{ext}" 31. path = f"app/static/images/makanan/{filename}" 32. with open(path, "wb") as buffer: 33. shutil.copyfileobj(gambar.file, buffer) 34. crud = KasirCRUD(db) 35. makanan = crud.tambah\_makanan(nama=nama, harga=harga, gambar=filename) 36. return {"message": "Makanan berhasil ditambahkan", "data": makanan} 37. @app.get("/makanan/") 38. def get\_makanan(db: Session = Depends(get\_db)): 39. crud = KasirCRUD(db) 40. return crud.semua\_makanan() 41. @app.put("/makanan/{id}") 42. async def update\_makanan(id: int, nama: str = Form(...), harga: int = Form(...), nama\_file: Optional[str] = Form(None), gambar: Optional[UploadFile] = File(None), db: Session = Depends(get\_db)): 43. filename = None 44. if gambar and nama\_file: 45. ext = os.path.splitext(gambar.filename)[1].lower() 46. if ext not in ALLOWED\_EXTENSIONS: 47. raise HTTPException(status\_code=400, detail="Format gambar tidak didukung.") 48. filename = f"{nama\_file}{ext}" 49. path = f"app/static/images/makanan/{filename}" 50. os.makedirs("app/static/images/makanan", exist\_ok=True) 51. with open(path, "wb") as buffer: 52. shutil.copyfileobj(gambar.file, buffer) 53. crud = KasirCRUD(db) 54. makanan = crud.update\_makanan(id=id, nama=nama, harga=harga, gambar=filename) 55. if not makanan: 56. raise HTTPException(status\_code=404, detail="Item tidak ditemukan") 57. return {"message": "Makanan berhasil diupdate", "data": makanan} 58. @app.delete("/makanan/{id}") 59. def delete\_makanan(id: int, db: Session = Depends(get\_db)): 60. crud = KasirCRUD(db) 61. crud.hapus\_makanan(id) 62. return {"message": "Makanan berhasil dihapus"} 63. # === Minuman === 64. @app.post("/minuman/") 65. async def upload\_minuman(nama: str, harga: int, nama\_file: str, gambar: UploadFile = File(...), db: Session = Depends(get\_db)): 66. ext = os.path.splitext(gambar.filename)[1].lower() 67. if ext.lower() not in ALLOWED\_EXTENSIONS: 68. raise HTTPException(status\_code=400, detail="Format gambar tidak didukung.") 69. os.makedirs("app/static/images/minuman", exist\_ok=True) 70. filename = f"{nama\_file}{ext}" 71. path = f"app/static/images/minuman/{filename}" 72. with open(path, "wb") as buffer: 73. shutil.copyfileobj(gambar.file, buffer) 74. crud = KasirCRUD(db) 75. minuman = crud.tambah\_minuman(nama=nama, harga=harga, gambar=filename) 76. return {"message": "Minuman berhasil ditambahkan", "data": minuman} 77. @app.get("/minuman/") 78. def get\_minuman(db: Session = Depends(get\_db)): 79. crud = KasirCRUD(db) 80. return crud.semua\_minuman() 81. @app.put("/minuman/{id}") 82. async def update\_minuman(id: int, nama: str = Form(...), harga: int = Form(...), nama\_file: Optional[str] = Form(None), gambar: Optional[UploadFile] = File(None), db: Session = Depends(get\_db)): 83. filename = None 84. if gambar and nama\_file: 85. ext = os.path.splitext(gambar.filename)[1].lower() 86. if ext not in ALLOWED\_EXTENSIONS: 87. raise HTTPException(status\_code=400, detail="Format gambar tidak didukung.") 88. filename = f"{nama\_file}{ext}" 89. path = f"app/static/images/minuman/{filename}" 90. os.makedirs("app/static/images/minuman", exist\_ok=True) 91. with open(path, "wb") as buffer: 92. shutil.copyfileobj(gambar.file, buffer) 93. crud = KasirCRUD(db) 94. minuman = crud.update\_minuman(id=id, nama=nama, harga=harga, gambar=filename) 95. if not minuman: 96. raise HTTPException(status\_code=404, detail="Item tidak ditemukan") 97. return {"message": "Minuman berhasil diupdate", "data": minuman} 98. @app.delete("/minuman/{id}") 99. def delete\_minuman(id: int, db: Session = Depends(get\_db)): 100. crud = KasirCRUD(db) 101. crud.hapus\_minuman(id) 102. return {"message": "Minuman berhasil dihapus"} 103. @app.get("/pelanggan/") 104. def get\_semua\_pelanggan(db: Session = Depends(get\_db)): 105. pelanggan\_list = db.query(models.Pelanggan).all() 106. hasil = [] 107. for p in pelanggan\_list: 108. hasil.append({ 109. "id": p.id, 110. "nama": p.nama, 111. "total\_harga": p.total\_harga, 112. "bayar": p.bayar, 113. "kembali": p.kembali, 114. "items": [ 115. { 116. "nama\_item": item.nama\_item, 117. "jumlah": item.jumlah, 118. "harga\_satuan": item.harga\_satuan, 119. "total": item.total 120. } 121. for item in p.items 122. ] 123. }) 124. return hasil 126. # === PELANGGAN === 127. @app.post("/pelanggan/") 128. def tambah\_pesanan(data: dict = Body(...), db: Session = Depends(get\_db)): 129. trx = Transaksi(db) 130. nama = data.get("nama") 131. items = data.get("items", []) 132. if not nama or not items: 133. raise HTTPException(status\_code=400, detail="Nama dan item harus diisi") 135. trx.tambah\_pesanan(nama=nama, items=items) 136. return { 137. "message": "Pesanan berhasil ditambahkan", 138. "id": trx.pelanggan\_id, 139. "total": trx.total\_keseluruhan 140. } 141. @app.put("/pelanggan/{id}") 142. def update\_pesanan(id: int, item\_updates: list[dict] = Body(...), db: Session = Depends(get\_db)): 143. trx = Transaksi(db) 144. updated = trx.update\_pesanan(pelanggan\_id=id, item\_updates=item\_updates) 145. if not updated: 146. raise HTTPException(status\_code=404, detail="Pelanggan tidak ditemukan") 148. return { 149. "message": "Pesanan berhasil diperbarui", 150. "total\_harga\_baru": trx.total\_keseluruhan 151. } 152. @app.post("/transaksi/selesai/") 153. def transaksi\_selesai(pelanggan\_id: int, bayar: int, db: Session = Depends(get\_db)): 154. trx = Transaksi(db) 155. trx.pelanggan\_id = pelanggan\_id 156. trx.total\_keseluruhan = trx.db.query(models.Pelanggan).filter(models.Pelanggan.id == pelanggan\_id).first().total\_harga 157. trx.nama\_pelanggan = trx.db.query(models.Pelanggan).filter(models.Pelanggan.id == pelanggan\_id).first().nama 158. trx.list\_items = [ 159. { 160. "nama\_item": item.nama\_item, 161. "jumlah": item.jumlah, 162. "harga\_satuan": item.harga\_satuan, 163. "total": item.total 164. } 165. for item in db.query(models.PemesananItem).filter(models.PemesananItem.pelanggan\_id == pelanggan\_id).all() 166. ] 167. trx.selesaikan\_transaksi(bayar=bayar) 168. return { 169. "message": "Transaksi berhasil", 170. "total": trx.total\_keseluruhan, 171. "bayar": trx.bayar, 172. "kembali": trx.kembali 173. } |

**Langkah 4 : Mengembangkan Antarmuka Pengguna (Frontend)**

Setelah backend dan API-nya siap, langkah terakhir adalah membangun antarmuka yang akan digunakan oleh pengguna.

* **File frontend.py:** File ini adalah aplikasi **Streamlit** yang sepenuhnya terpisah dari backend. Halaman ini berisi semua komponen visual, seperti tombol, tabel, dan form input. Untuk mendapatkan atau mengirim data, aplikasi frontend ini membuat permintaan HTTP ke API backend menggunakan library requests. Aplikasi ini mengelola alur halaman (login, admin, kasir, transaksi) menggunakan st.session\_state.

|  |
| --- |
| 1. import streamlit as st 2. import requests 3. API\_URL = "http://localhost:8000" 4. # Inisialisasi session state 5. if 'current\_page' not in st.session\_state: 6. st.session\_state.current\_page = 'login' 7. if 'user' not in st.session\_state: 8. st.session\_state.user = '' 9. if 'is\_admin' not in st.session\_state: 10. st.session\_state.is\_admin = False 11. if 'keranjang' not in st.session\_state: 12. st.session\_state.keranjang = [] 13. if 'last\_added' not in st.session\_state: 14. st.session\_state.last\_added = None 15. if 'transaksi\_selesai' not in st.session\_state: 16. st.session\_state.transaksi\_selesai = False 17. def login\_page(): 18. st.title("Kasir Muladi") 19. nama = st.text\_input("Masukkan Nama Anda") 20. if nama == "admin": 21. passwd = st.text\_input("Masukkan password", type="password") 22. if st.button("Masuk sebagai Admin"): 23. if passwd == "admin123":  # bisa diganti nanti autentikasi via API 24. st.session\_state.user = nama 25. st.session\_state.is\_admin = True 26. st.session\_state.current\_page = 'admin' 27. else: 28. st.error("Password salah!") 29. elif nama: 30. if st.button("Lanjutkan"): 31. st.session\_state.user = nama 32. st.session\_state.is\_admin = False 33. st.session\_state.current\_page = 'dashboard\_user' 34. def admin\_page(): 35. st.title("Admin Page") 36. col1, col2 = st.columns([4, 1]) 37. with col1: 38. st.write(f"Selamat datang, {st.session\_state.get('user', 'Admin')}!") 40. with col2: 41. # Tombol untuk kembali ke halaman login 42. if st.button("⬅️ Kembali", use\_container\_width=True): 43. # Hapus state user saat kembali ke login untuk sesi yang bersih 44. if 'user' in st.session\_state: 45. del st.session\_state['user'] 46. st.session\_state.current\_page = "login" 47. st.rerun() 48. tab1, tab2, tab3 = st.tabs(["📄 Riwayat Transaksi", "🍽️ CRUD Makanan", "🥤 CRUD Minuman"]) 49. # ========== RIWAYAT ========== 50. with tab1: 51. st.header("Riwayat Transaksi Pelanggan") 52. res = requests.get(f"{API\_URL}/pelanggan/") 53. if res.ok: 54. data = res.json() 55. for pelanggan in data: 56. st.subheader(f"{pelanggan['nama']} (ID: {pelanggan['id']})") 57. st.write("Total:", f"Rp {pelanggan['total\_harga']:,}".replace(",", ".")) 58. st.write("Bayar:", f"Rp {pelanggan['bayar']:,}".replace(",", ".")) 59. st.write("Kembali:", f"Rp {pelanggan['kembali']:,}".replace(",", ".")) 60. if pelanggan["items"]: 61. with st.expander("Lihat Pesanan"): 62. for item in pelanggan["items"]: 63. st.write(f"- {item['nama\_item']} × {item['jumlah']} = Rp {item['total']:,}".replace(",", ".")) 64. else: 65. st.error("Gagal memuat data pelanggan") 66. # ========== CRUD MAKANAN ========== 67. with tab2: 68. st.header("Kelola Menu Makanan") 69. makanan = requests.get(f"{API\_URL}/makanan/").json() 70. for item in makanan: 71. with st.expander(f"{item['nama']} (Rp {item['harga']:,})".replace(",", ".")): 72. new\_nama = st.text\_input(f"Nama Baru - {item['id']}", value=item["nama"], key=f"nama\_mkn\_{item['id']}") 73. new\_harga = st.number\_input(f"Harga Baru - {item['id']}", value=item["harga"], key=f"harga\_mkn\_{item['id']}") 74. new\_file\_name = st.text\_input(f"Nama File Gambar - {item['id']}", key=f"file\_mkn\_{item['id']}") 75. new\_gambar = st.file\_uploader("Gambar Baru", key=f"img\_mkn\_{item['id']}", type=["jpg", "jpeg", "png"]) 76. if st.button("💾 Simpan Perubahan", key=f"edit\_mkn\_{item['id']}"): 77. files = {"gambar": new\_gambar} if new\_gambar else None 78. data = { 79. "nama": new\_nama, 80. "harga": new\_harga, 81. "nama\_file": new\_file\_name or item["gambar"].split(".")[0] 82. } 83. r = requests.put(f"{API\_URL}/makanan/{item['id']}", data=data, files=files) 84. if r.ok: 85. st.success("Berhasil diperbarui!") 86. st.rerun() 87. if st.button("🗑️ Hapus", key=f"del\_mkn\_{item['id']}"): 88. r = requests.delete(f"{API\_URL}/makanan/{item['id']}") 89. if r.ok: 90. st.success("Berhasil dihapus!") 91. st.rerun() 92. st.subheader("➕ Tambah Makanan Baru") 93. nama\_baru = st.text\_input("Nama Makanan", key="tambah\_makanan\_nama") 94. harga\_baru = st.number\_input("Harga", min\_value=0, step=500, key="tambah\_makanan\_harga") 95. nama\_file\_baru = st.text\_input("Nama File Gambar (tanpa ekstensi)", key="tambah\_makanan\_file") 96. gambar\_baru = st.file\_uploader("Upload Gambar", type=["jpg", "jpeg", "png"], key="tambah\_makanan\_gambar") 97. if st.button("Tambah Makanan"): 98. if not gambar\_baru or not nama\_file\_baru: 99. st.warning("Mohon isi nama file dan upload gambar.") 100. else: 101. files = {"gambar": gambar\_baru} 102. data = { 103. "nama": nama\_baru, 104. "harga": harga\_baru, 105. "nama\_file": nama\_file\_baru 106. } 107. res = requests.post(f"{API\_URL}/makanan/", data=data, files=files) 108. if res.ok: 109. st.success("Berhasil ditambahkan!") 110. st.rerun() 111. # ========== CRUD MINUMAN ========== 112. with tab3: 113. st.header("Kelola Menu Minuman") 114. minuman = requests.get(f"{API\_URL}/minuman/").json() 115. for item in minuman: 116. with st.expander(f"{item['nama']} (Rp {item['harga']:,})".replace(",", ".")): 117. new\_nama = st.text\_input(f"Nama Baru - {item['id']}", value=item["nama"], key=f"nama\_mnm\_{item['id']}") 118. new\_harga = st.number\_input(f"Harga Baru - {item['id']}", value=item["harga"], key=f"harga\_mnm\_{item['id']}") 119. new\_file\_name = st.text\_input(f"Nama File Gambar - {item['id']}", key=f"file\_mnm\_{item['id']}") 120. new\_gambar = st.file\_uploader("Gambar Baru", key=f"img\_mnm\_{item['id']}", type=["jpg", "jpeg", "png"]) 121. if st.button("💾 Simpan Perubahan", key=f"edit\_mnm\_{item['id']}"): 122. files = {"gambar": new\_gambar} if new\_gambar else None 123. data = { 124. "nama": new\_nama, 125. "harga": new\_harga, 126. "nama\_file": new\_file\_name or item["gambar"].split(".")[0] 127. } 128. r = requests.put(f"{API\_URL}/minuman/{item['id']}", data=data, files=files) 129. if r.ok: 130. st.success("Berhasil diperbarui!") 131. st.rerun() 132. if st.button("🗑️ Hapus", key=f"del\_mnm\_{item['id']}"): 133. r = requests.delete(f"{API\_URL}/minuman/{item['id']}") 134. if r.ok: 135. st.success("Berhasil dihapus!") 136. st.rerun() 137. st.subheader("➕ Tambah Minuman Baru") 138. nama\_baru = st.text\_input("Nama Minuman", key="tambah\_minuman\_nama") 139. harga\_baru = st.number\_input("Harga", min\_value=0, step=500, key="tambah\_minuman\_harga") 140. nama\_file\_baru = st.text\_input("Nama File Gambar (tanpa ekstensi)", key="tambah\_minuman\_file") 141. gambar\_baru = st.file\_uploader("Upload Gambar", type=["jpg", "jpeg", "png"], key="tambah\_minuman\_gambar") 142. if st.button("Tambah Minuman"): 143. if not gambar\_baru or not nama\_file\_baru: 144. st.warning("Mohon isi nama file dan upload gambar.") 145. else: 146. files = {"gambar": gambar\_baru} 147. data = { 148. "nama": nama\_baru, 149. "harga": harga\_baru, 150. "nama\_file": nama\_file\_baru 151. } 152. res = requests.post(f"{API\_URL}/minuman/", data=data, files=files) 153. if res.ok: 154. st.success("Berhasil ditambahkan!") 155. st.rerun() 156. def dashboard\_user\_page(): 157. st.title(f"Halo, {st.session\_state.user}👋. Pesen Apa Hari Ini?") 158. if st.session\_state.last\_added: 159. st.success(f"✅ {st.session\_state.last\_added} berhasil ditambahkan ke keranjang!") 160. st.session\_state.last\_added = None 161. st.header("Menu Makanan") 162. makanan\_res = requests.get(f"{API\_URL}/makanan/") 163. if makanan\_res.ok: 164. makanan\_list = makanan\_res.json() 165. for item in makanan\_list: 166. col1, col2, col3 = st.columns([1, 3, 1]) 167. with col1: 168. st.image(f"{API\_URL}/static/images/makanan/{item['gambar']}", width=80) 169. with col2: 170. st.subheader(item["nama"]) 171. st.write(f"Rp {item['harga']:,}".replace(",", ".")) 172. with col3: 173. if st.button(f"➕ Tambah", key=f"add\_makanan\_{item['id']}"): 174. st.session\_state.keranjang.append({ 175. "nama\_item": item["nama"], 176. "jumlah": 1, 177. "harga\_satuan": item["harga"] 178. }) 179. st.session\_state.last\_added = item["nama"] 180. st.rerun() 181. st.header("Menu Minuman") 182. minuman\_res = requests.get(f"{API\_URL}/minuman/") 183. if minuman\_res.ok: 184. minuman\_list = minuman\_res.json() 185. for item in minuman\_list: 186. col1, col2, col3 = st.columns([1, 3, 1]) 187. with col1: 188. st.image(f"{API\_URL}/static/images/minuman/{item['gambar']}", width=80) 189. with col2: 190. st.subheader(item["nama"]) 191. st.write(f"Rp {item['harga']:,}".replace(",", ".")) 192. with col3: 193. if st.button(f"➕ Tambah", key=f"add\_minuman\_{item['id']}"): 194. st.session\_state.keranjang.append({ 195. "nama\_item": item["nama"], 196. "jumlah": 1, 197. "harga\_satuan": item["harga"] 198. }) 199. st.session\_state.last\_added = item["nama"] 200. st.rerun() 201. if st.button("Lanjut ke Transaksi"): 202. st.session\_state.current\_page = 'transaksi' 204. def transaksi\_page(): 205. import requests 206. st.title("Halaman Transaksi") 207. if "keranjang" not in st.session\_state: 208. st.session\_state.keranjang = [] 209. if "transaksi\_selesai" not in st.session\_state: 210. st.session\_state.transaksi\_selesai = False 211. if "user" not in st.session\_state: 212. st.session\_state.user = '' 213. if st.session\_state.transaksi\_selesai: 214. st.success("Transaksi berhasil!") 216. total = st.session\_state.get("total\_harga", 0) 217. bayar = st.session\_state.get("bayar", 0) 218. kembali = st.session\_state.get("kembali", 0) 219. st.write(f"Total Belanja: Rp {total:,}".replace(",", ".")) 220. st.write(f"Jumlah Bayar: Rp {bayar:,}".replace(",", ".")) 221. st.write(f"Kembalian: Rp {kembali:,}".replace(",", ".")) 223. if st.button("✅ Selesai"): 224. for key in ["keranjang", "total\_harga", "pelanggan\_id", "transaksi\_selesai", "bayar", "kembali", "user"]: 225. if key in st.session\_state: 226. del st.session\_state[key] 228. st.session\_state.current\_page = "login" 229. st.rerun() 230. return 231. # --- Bagian 2: Proses Transaksi --- 232. keranjang = st.session\_state.keranjang 233. if not keranjang: 234. st.warning("Keranjang belanja masih kosong.") 235. if st.button("⬅️ Kembali ke Menu"): 236. st.session\_state.current\_page = 'dashboard\_user' 237. st.rerun() 238. return 239. st.write("### Ringkasan Pesanan:") 241. for idx, item in enumerate(keranjang): 242. col1, col2, col3, col4 = st.columns([3, 2, 2, 1]) 243. with col1: 244. st.write(item["nama\_item"]) 245. with col2: 246. new\_jumlah = st.number\_input("Jumlah", min\_value=1, value=item["jumlah"], key=f"jumlah\_{idx}") 247. st.session\_state.keranjang[idx]["jumlah"] = new\_jumlah 248. with col3: 249. st.write(f"Rp {item['harga\_satuan']:,}".replace(",", ".")) 250. with col4: 251. if st.button("❌", key=f"hapus\_{idx}"): 252. st.session\_state.keranjang.pop(idx) 253. st.rerun() 254. total = sum(item["jumlah"] \* item["harga\_satuan"] for item in st.session\_state.keranjang) 255. st.session\_state.total\_harga = total 256. st.write(f"### Total Belanja: Rp {total:,}".replace(",", ".")) 257. bayar = st.number\_input("Nominal Bayar", min\_value=0, step=1000, format="%d") 258. col\_selesai, col\_kembali = st.columns(2) 260. with col\_kembali: 261. if st.button("⬅️ Kembali ke Menu", use\_container\_width=True): 262. st.session\_state.current\_page = 'dashboard\_user' 263. st.rerun() 264. with col\_selesai: 265. if st.button("Selesaikan Transaksi", use\_container\_width=True): 266. items = [item for item in st.session\_state.keranjang if item["jumlah"] > 0] 267. if not items: 268. st.error("Tidak ada item valid dalam keranjang.") 269. return 270. if bayar < total: 271. st.error("Nominal bayar kurang dari total belanja.") 272. return 273. try: 274. res = requests.post(f"{API\_URL}/pelanggan/", json={ 275. "nama": st.session\_state.user, 276. "items": items 277. }) 278. res.raise\_for\_status() 279. pelanggan\_id = res.json()["id"] 280. st.session\_state.pelanggan\_id = pelanggan\_id 281. bayar\_res = requests.post(f"{API\_URL}/transaksi/selesai/", params={ 282. "pelanggan\_id": pelanggan\_id, 283. "bayar": bayar 284. }) 285. bayar\_res.raise\_for\_status() 286. kembali = bayar - total 287. st.session\_state.bayar = bayar 288. st.session\_state.kembali = kembali 289. st.session\_state.transaksi\_selesai = True 290. st.rerun() 291. except requests.exceptions.RequestException as e: 292. st.error(f"Gagal terhubung ke server: {e}") 293. except Exception as e: 294. st.error(f"Terjadi kesalahan: {e}") 295. # Routing antar halaman 296. if st.session\_state.current\_page == 'login': 297. login\_page() 298. elif st.session\_state.current\_page == 'admin': 299. admin\_page() 300. elif st.session\_state.current\_page == 'dashboard\_user': 301. dashboard\_user\_page() 302. elif st.session\_state.current\_page == 'transaksi': 303. transaksi\_page() |

**Langkah 5 : Uji Coba Aplikasi**

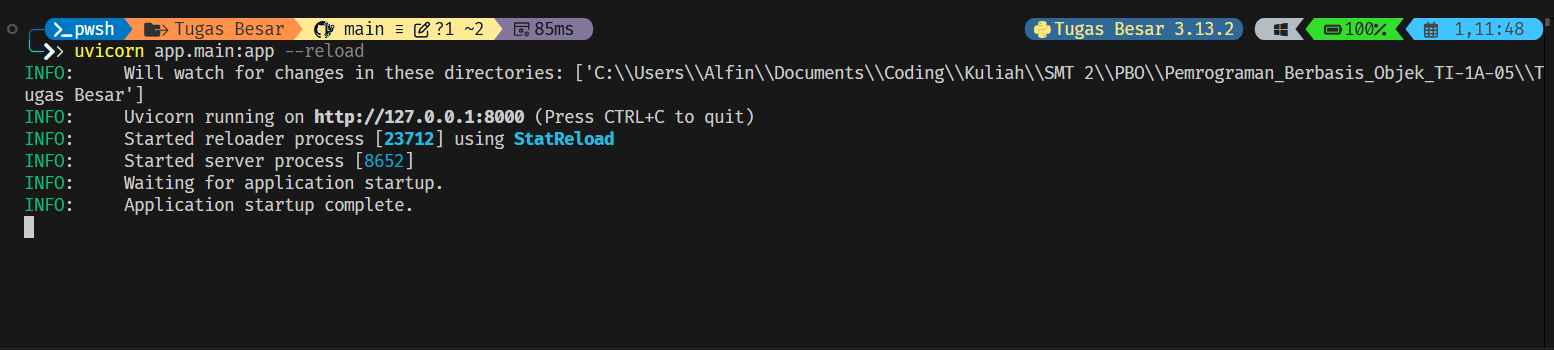
Karena arsitektur aplikasi terdiri dari dua bagian yang berjalan terpisah (backend dan frontend), keduanya harus dijalankan secara bersamaan.

1. **Jalankan Server backend**

Buka terminal, arahkan ke direktori proyek Anda, dan jalankan perintah berikut untuk mengaktifkan server FastAPI.

uvicorn app.main:app --reload

app.main mengarah ke file main.py yang berada di folder app dan :app mengarah ke nama variabel yang menyimpan FastAPI()

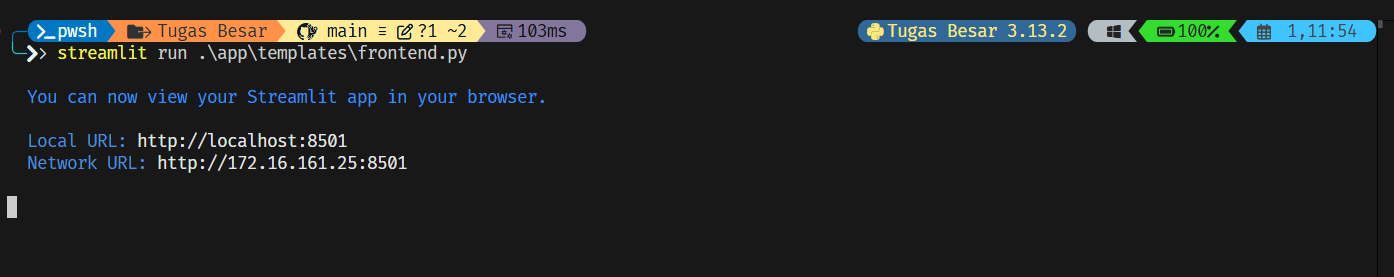


Setelah server backend dinyalakan, maka otomatis akan membuat file database dengan tipe .db untuk menyimpan data aplikasi.



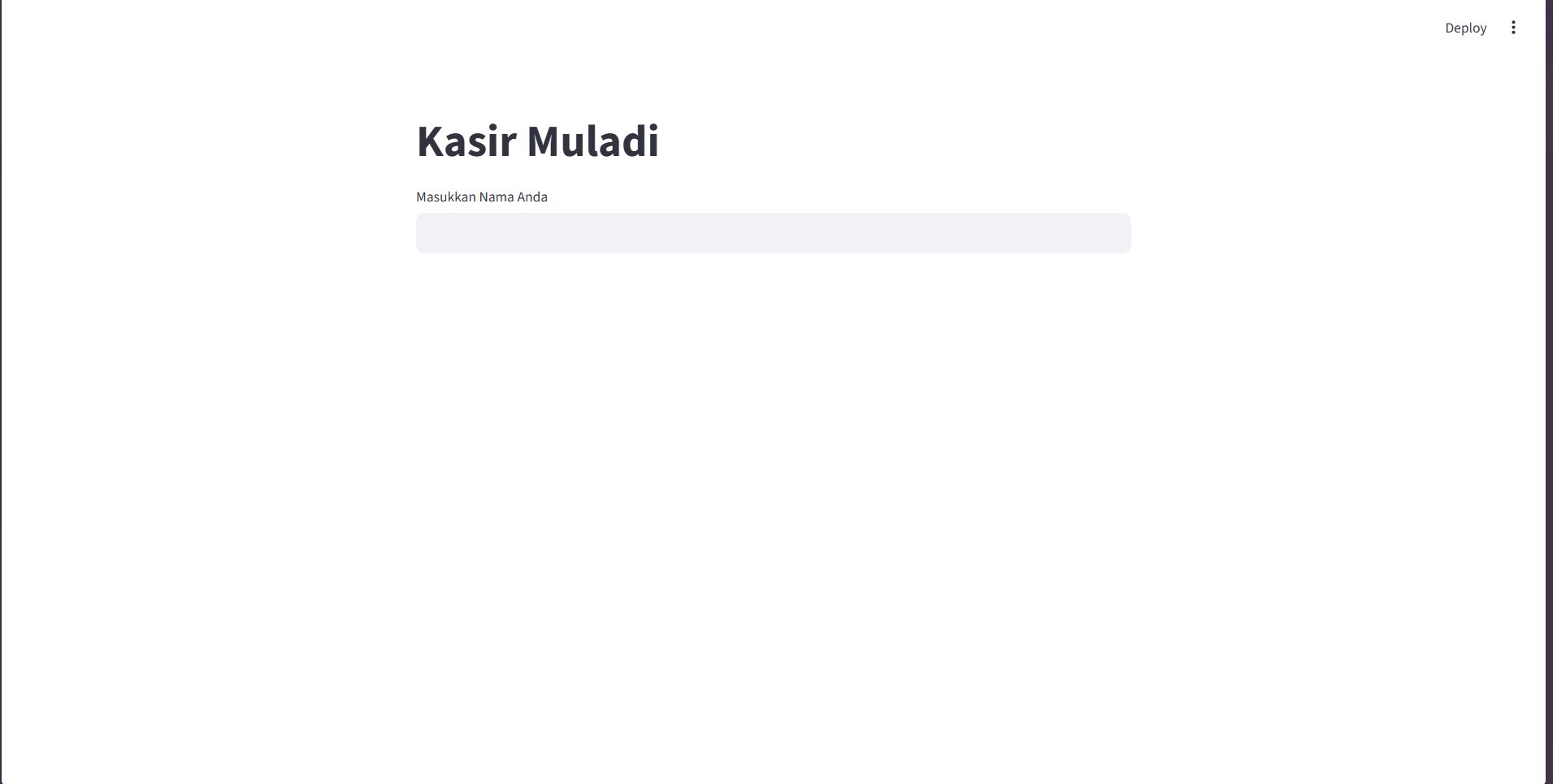
1. **Jalankan Aplikasi Frontend**

Buka terminal baru, arahkan ke direktori yang sama, dan jalankan perintah berikut untuk menampilkan antarmuka Streamlit.



.\app\templates\frontend.py mengacu pada letak file frontend.py yang berada di folder templates dan didalam folder app.

Setelah dijalankan maka website akan otomatis terbuka dibrowser.

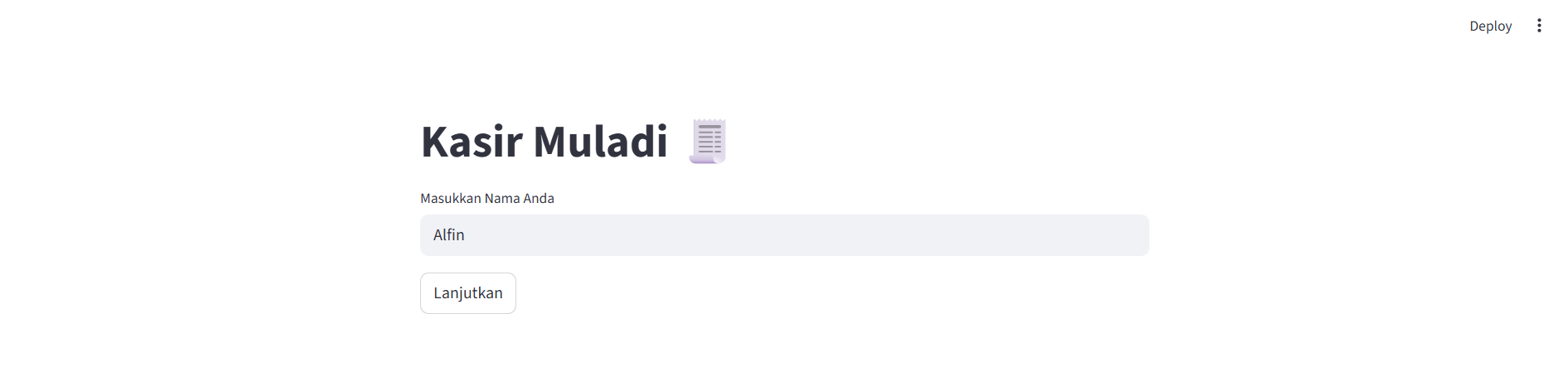


**Link Github :**

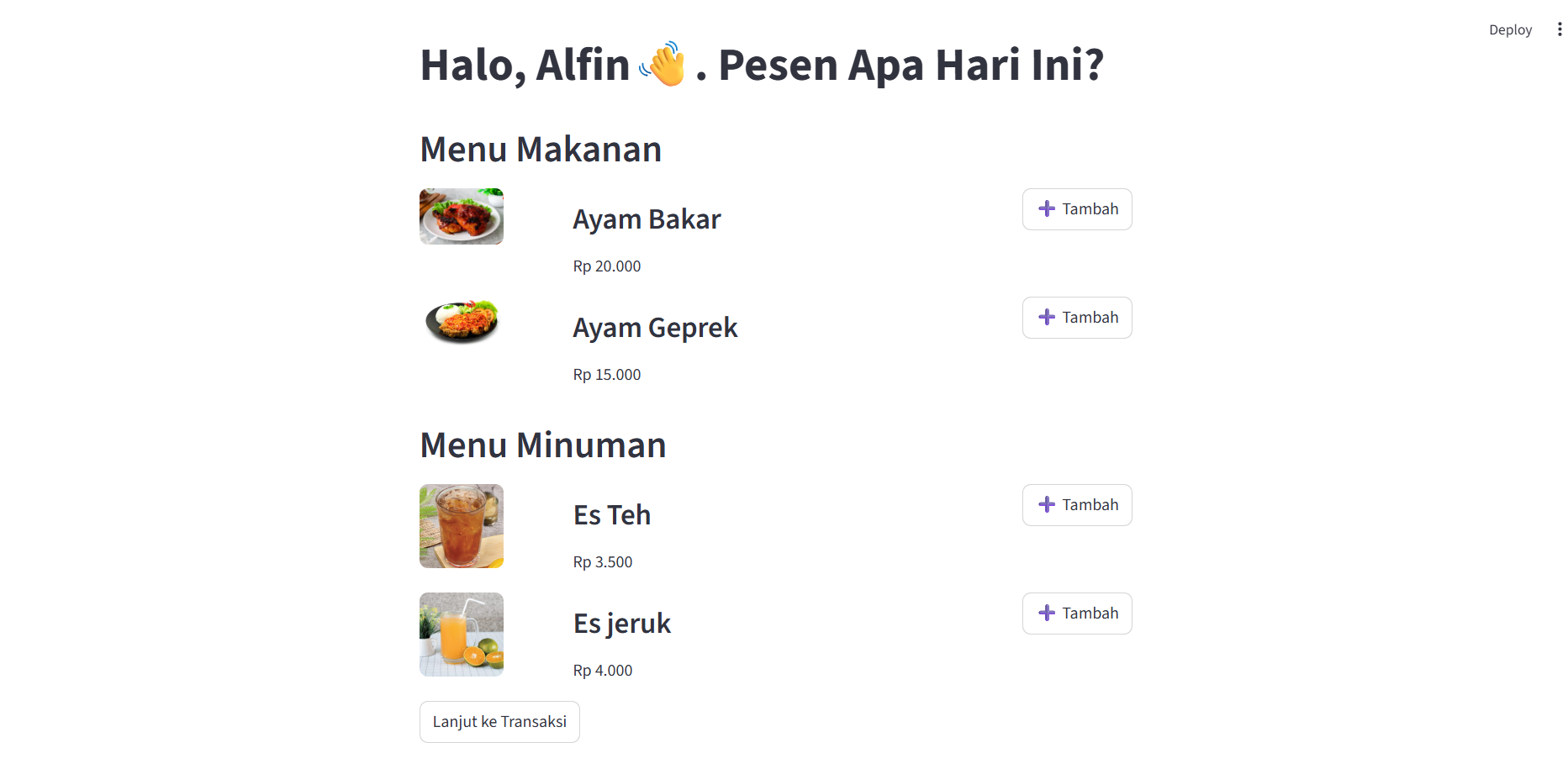
<https://github.com/Algaray02/Pemrograman_Berbasis_Objek_TI-1A-05/tree/main/Tugas%20Besar>

**Konsep Kerja Website**

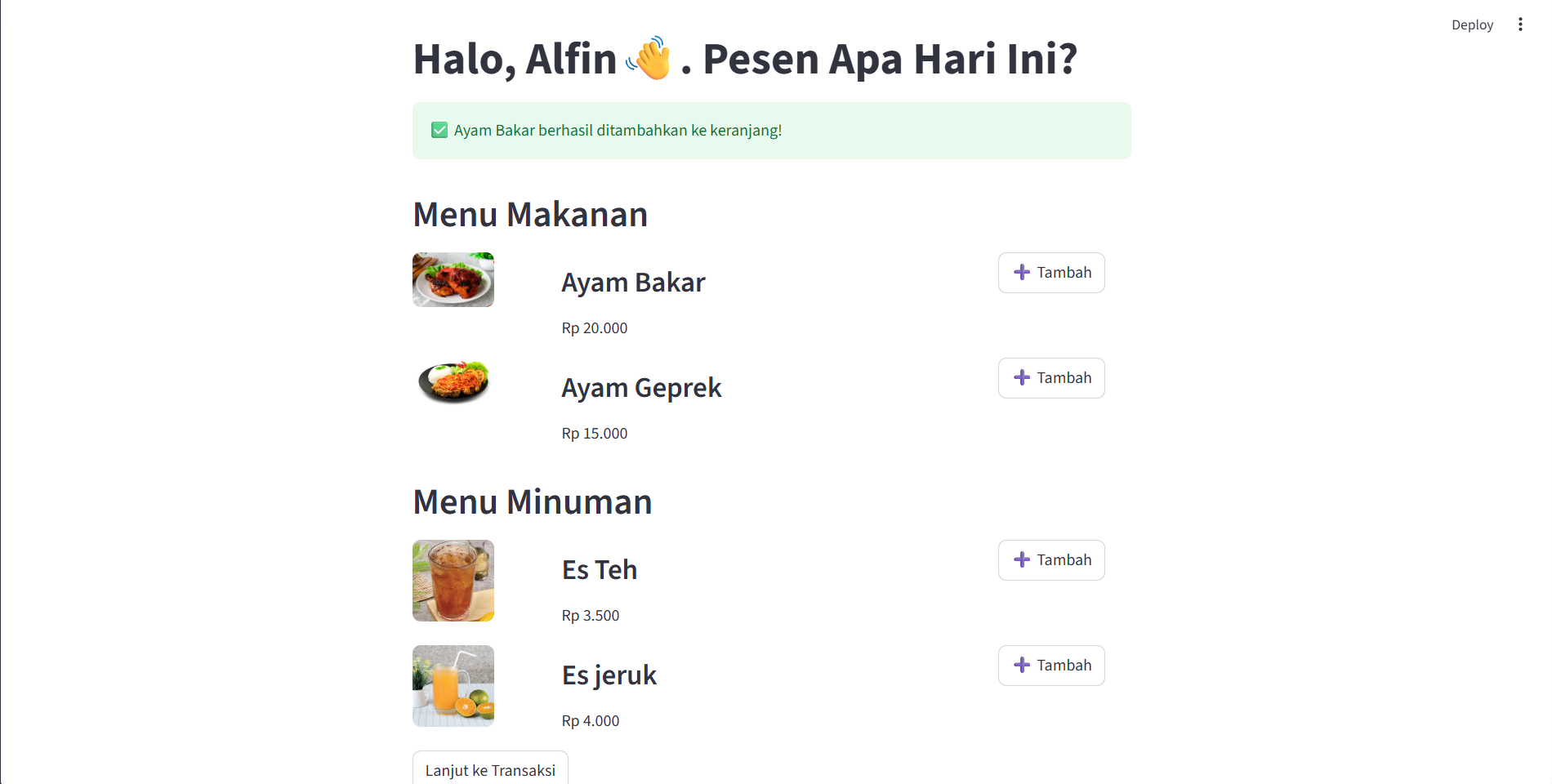
* **Halaman Awal**
* Masukkan Nama Pelanggan yang ingin memesan menu, selanjutnya klik “lanjutkan”



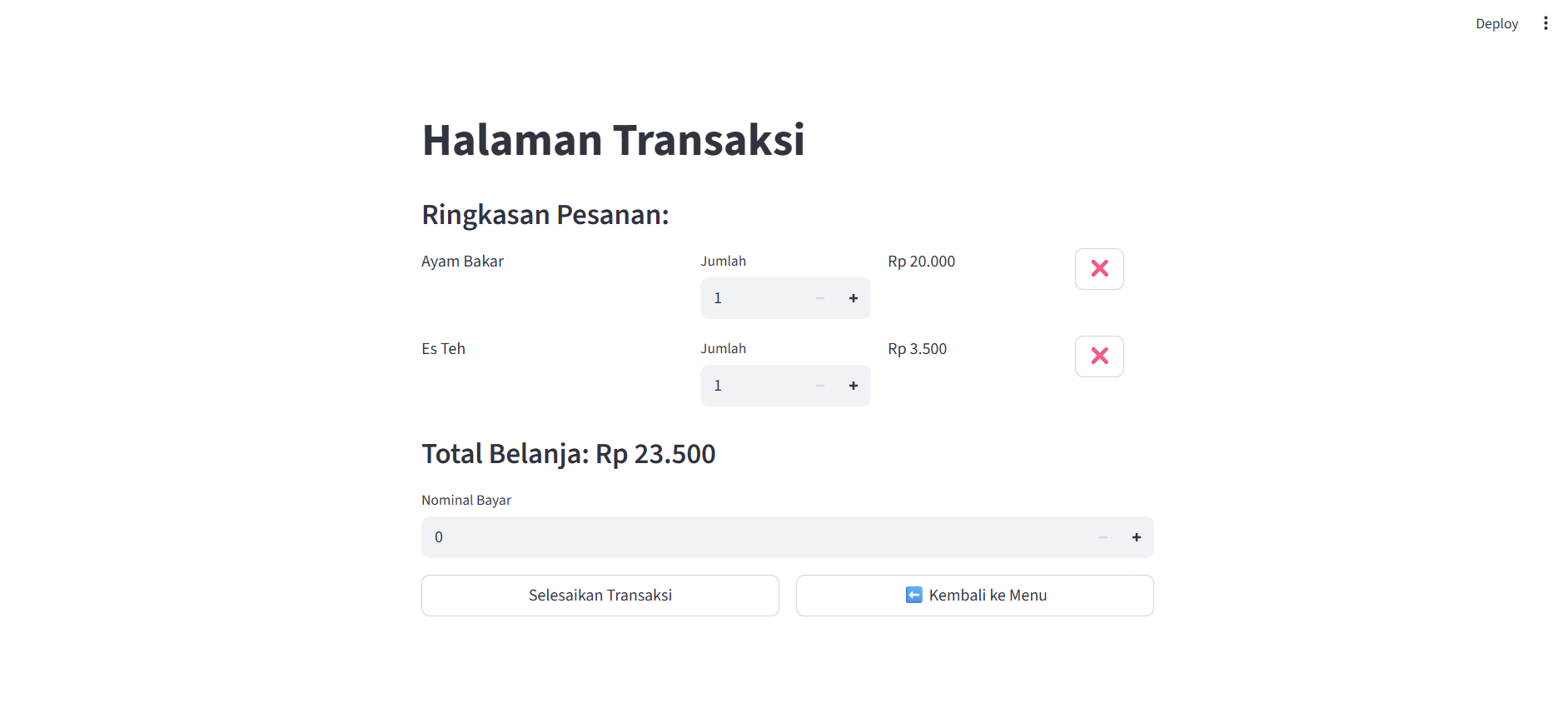
* **Halaman Dashboard Menu**



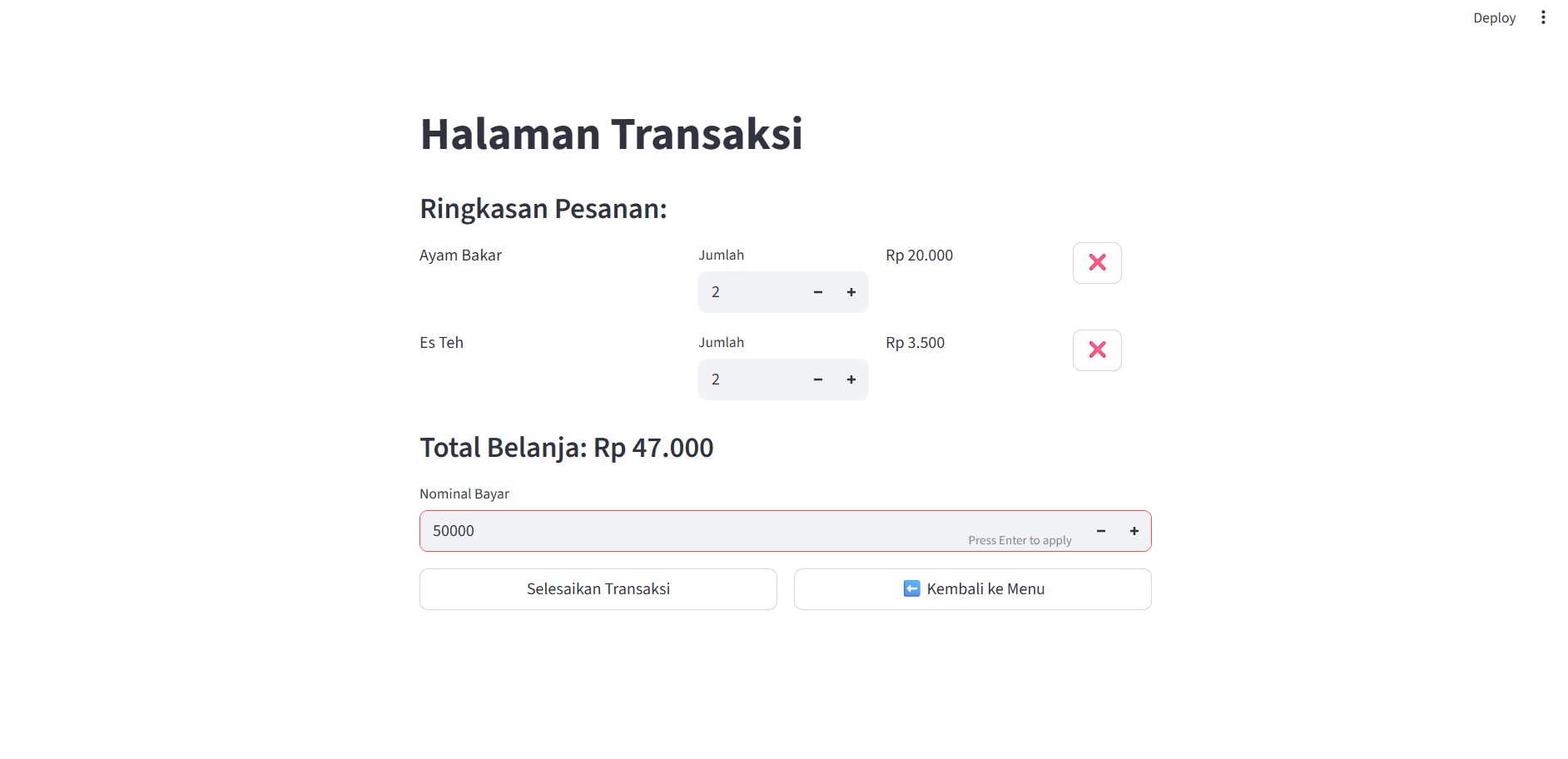
* Pada tampilan ini kita bisa menambahkan menu sesuka hati dengan cara klik tombol Tambah



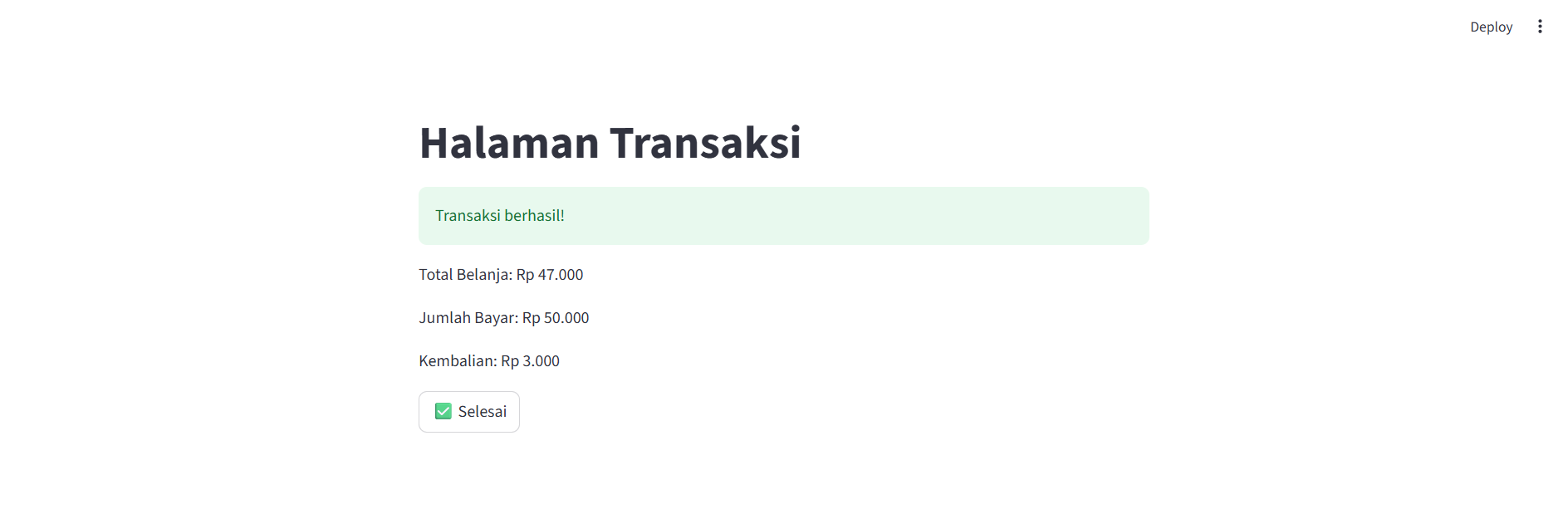
* Setelah selesai menambahkan menu selanjutnya klik “lanjut ke Transaksi”
* **Halaman Transaksi**



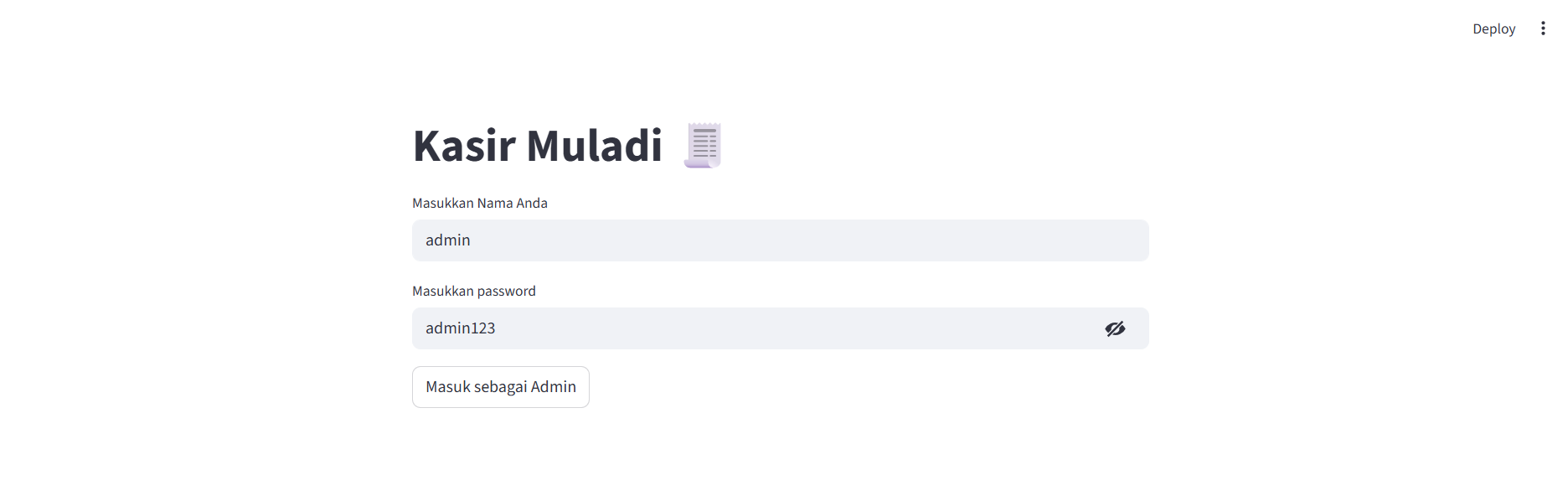
* Pelanggan bisa menambahkan, mengurangi, dan menghapus menu yang sudah ditambahkan pada halaman transaksi ini, setelah selesai masukkan nominal uang untuk membayar pesanan dan klik “Selesaikan Transaksi”



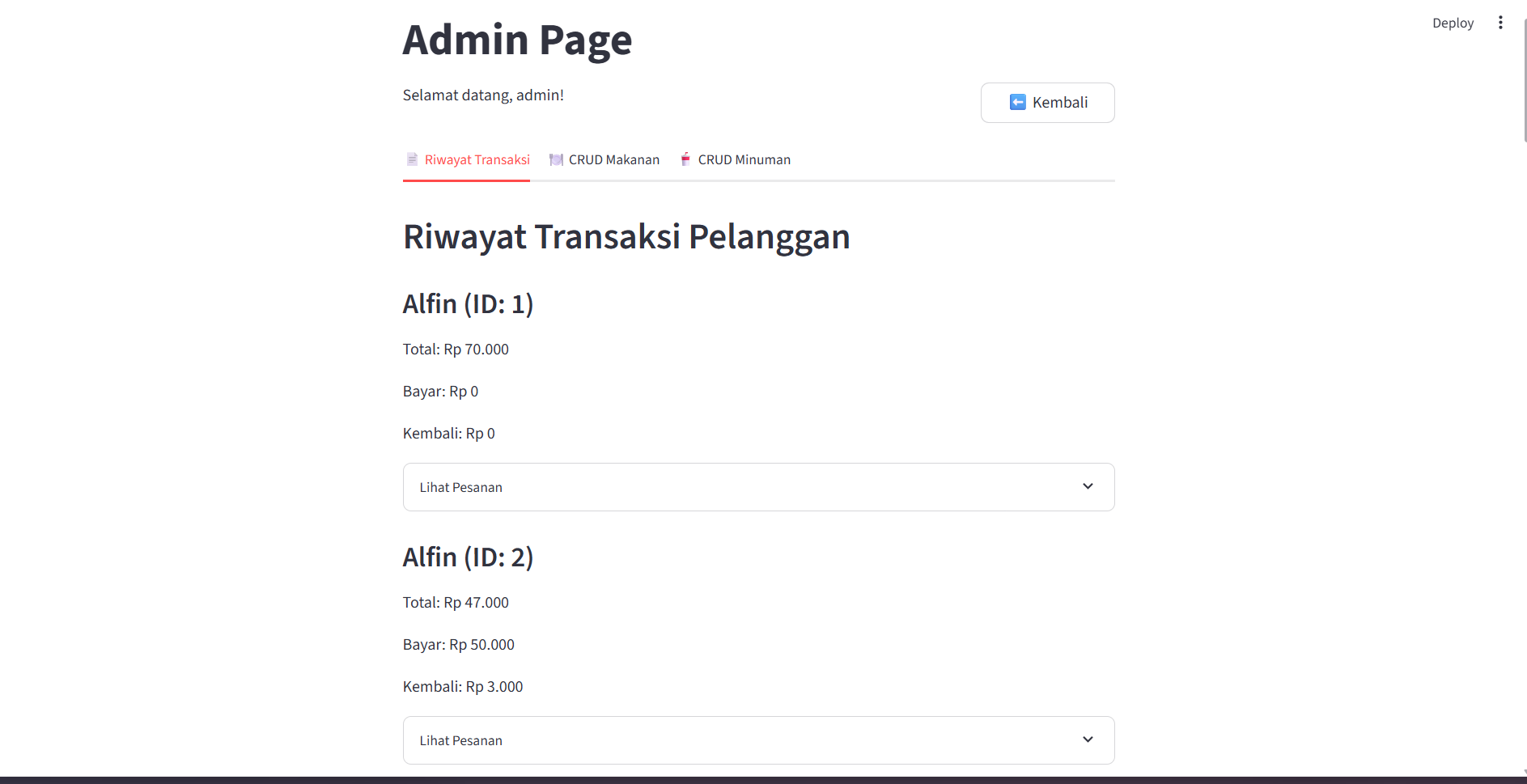
* Setelah selesai maka tampilan halaman akan menampilkan total belanja, jumlah bayar, dan Kembali

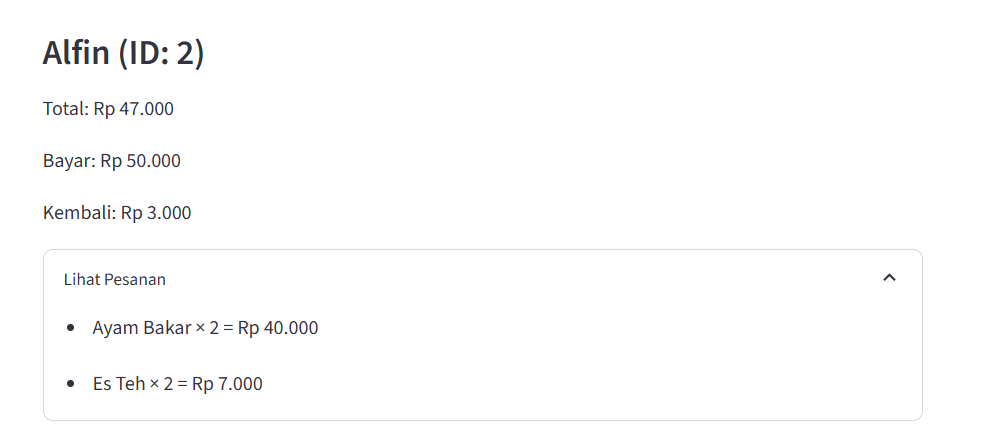


* **Admin Side**
* Apabila memasukkan nama “admin” maka akan diminta password, masukkan password “admin123”

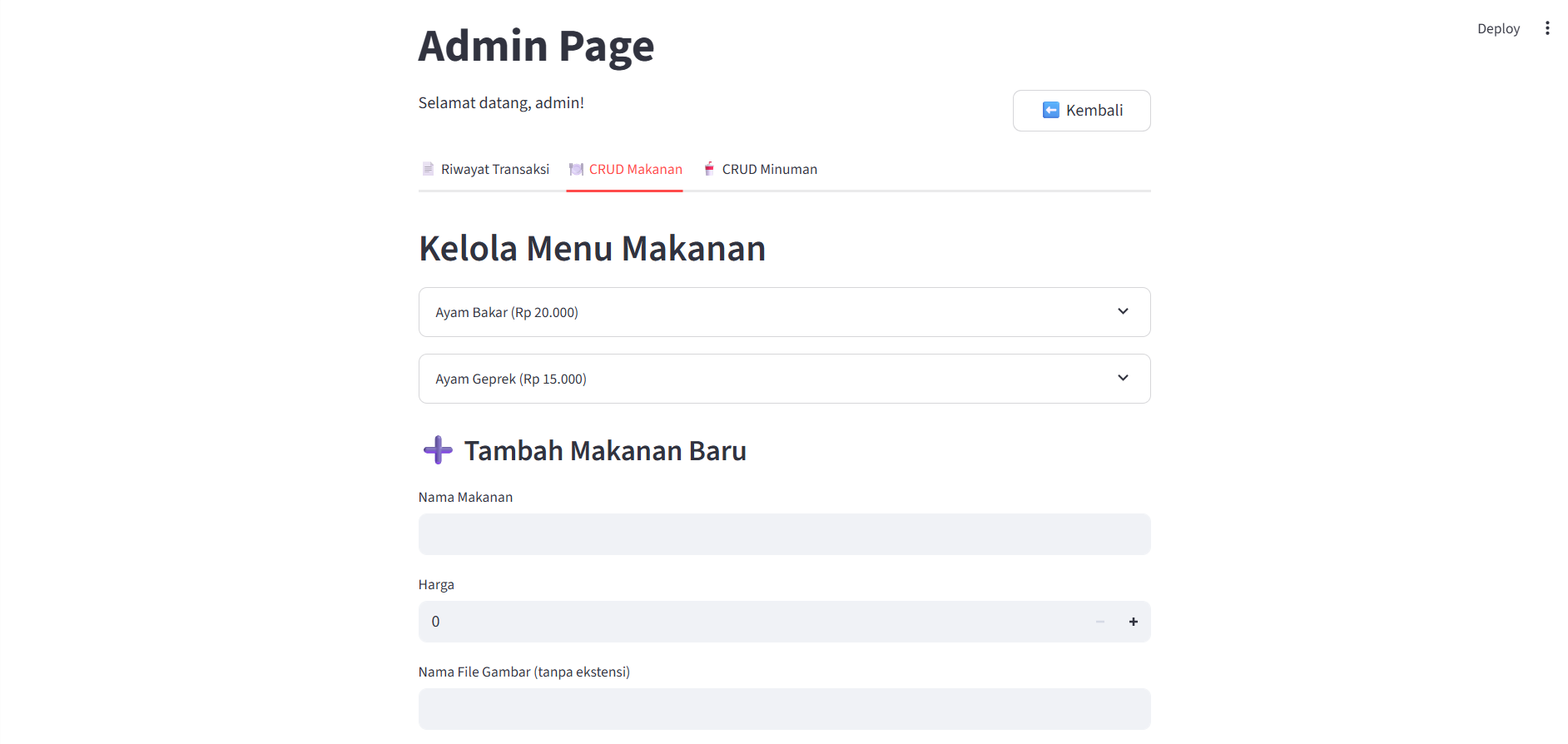
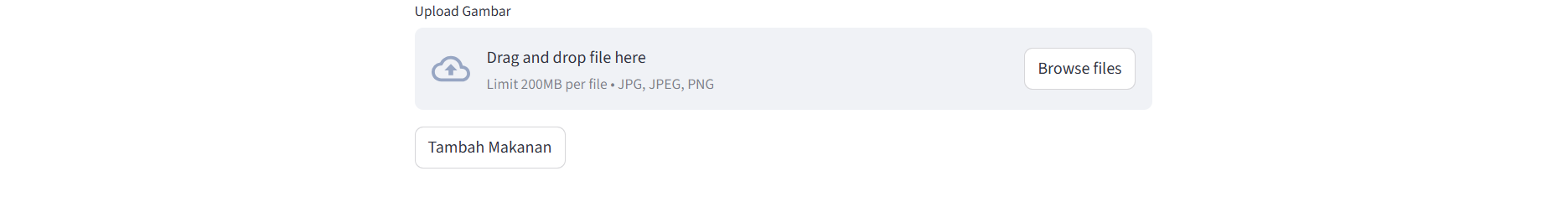


* **Admin Page – Riwayat Transaksi**

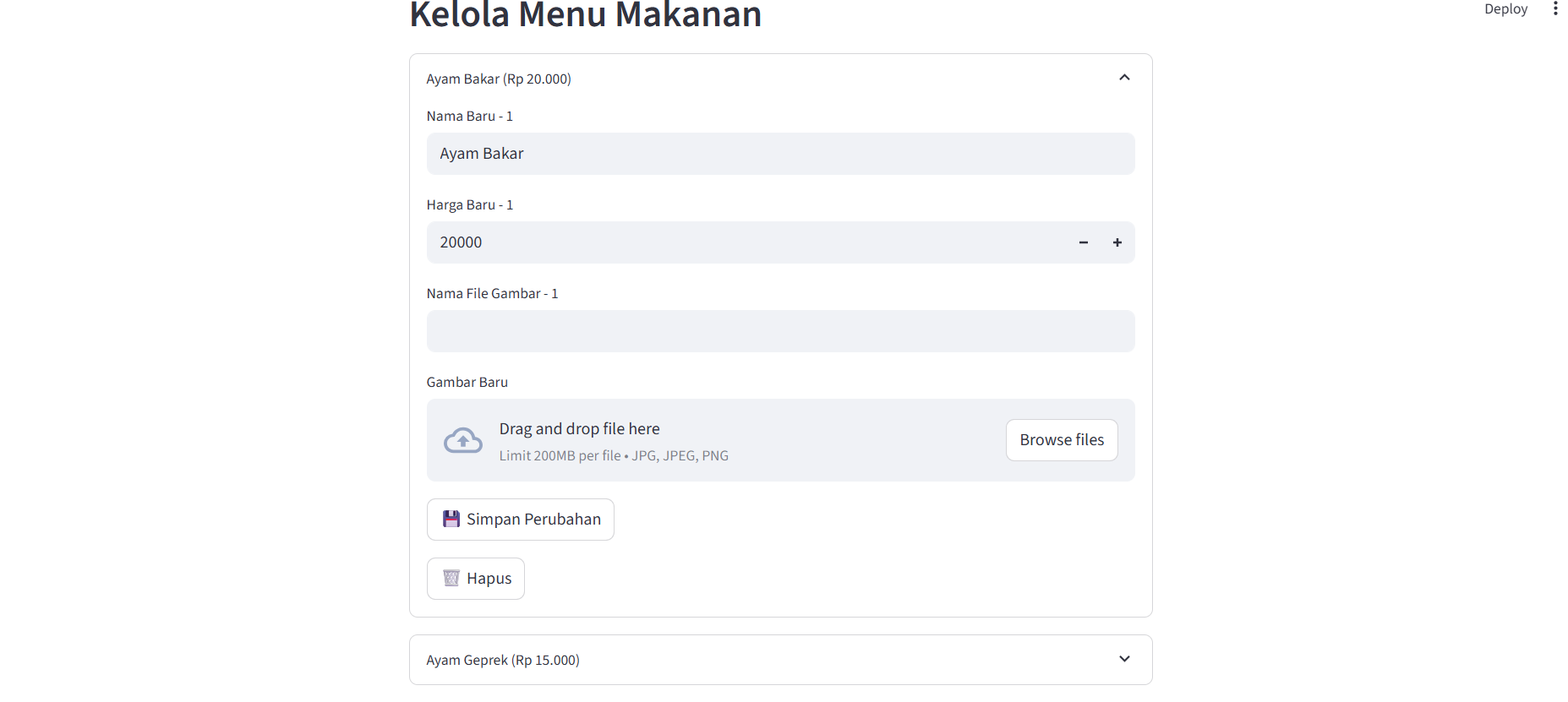


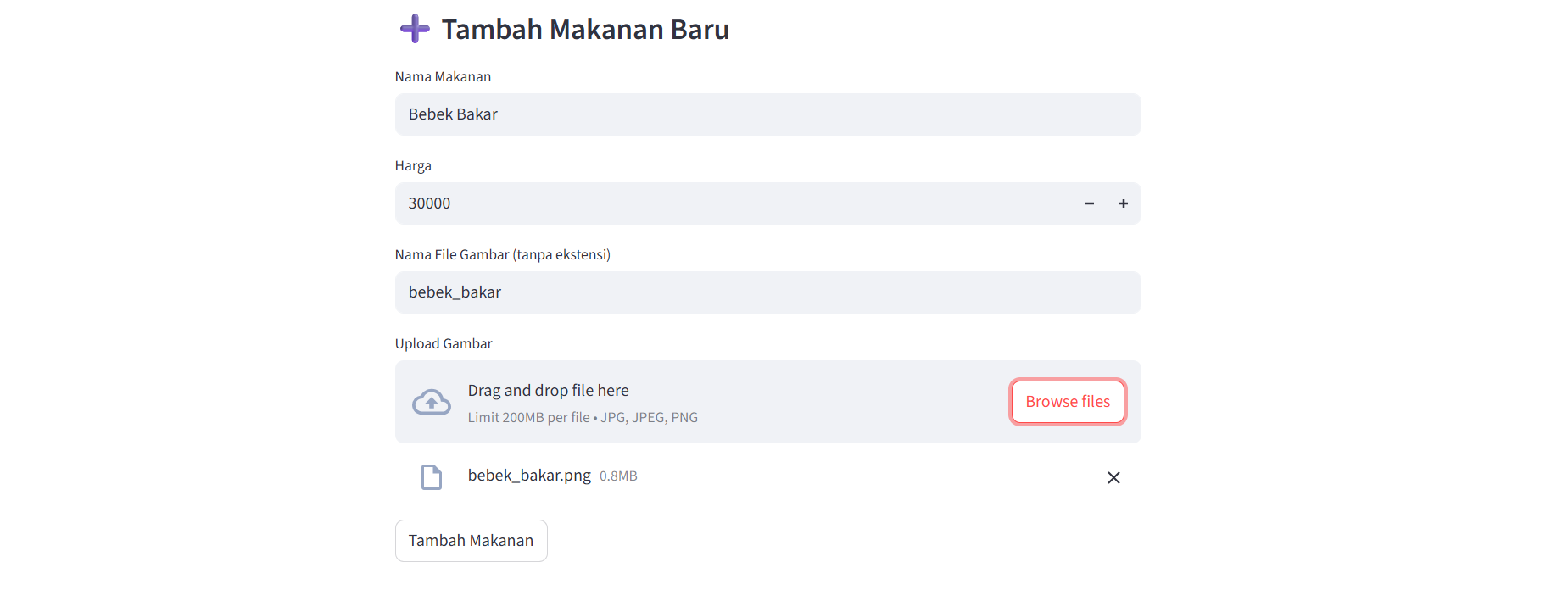


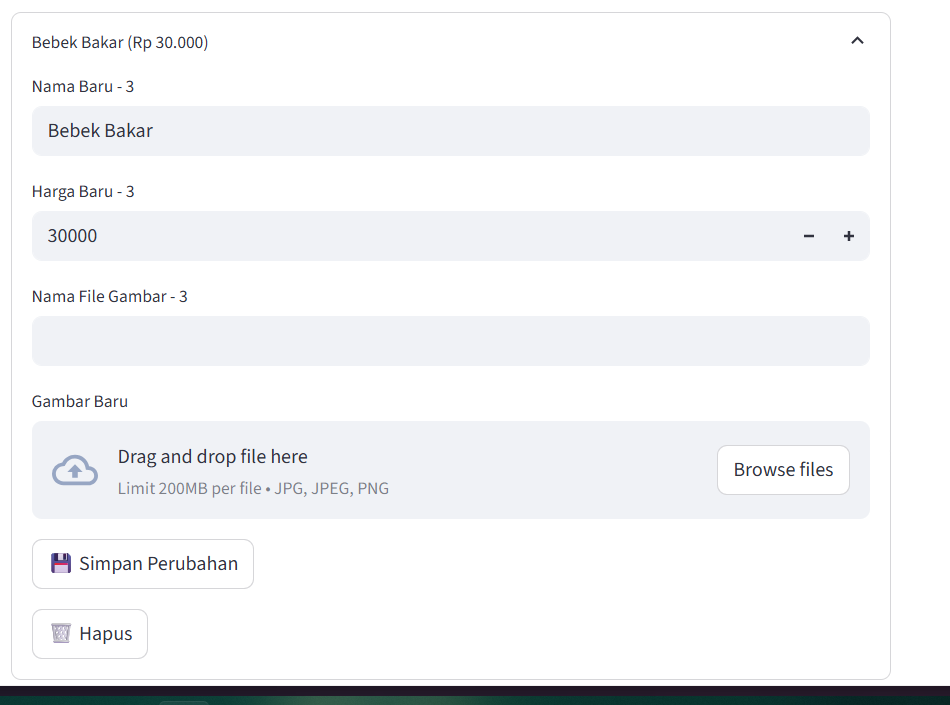
* Pada bagian ini admin bisa melihat Riwayat transaksi pelanggan secara detail termasuk pesanan yang dipilih
* **Admin Page – CRUD Makanan**
* Pada bagian ini admin bisa melakukan CRUD ke menu makanan pada aplikasi kasir

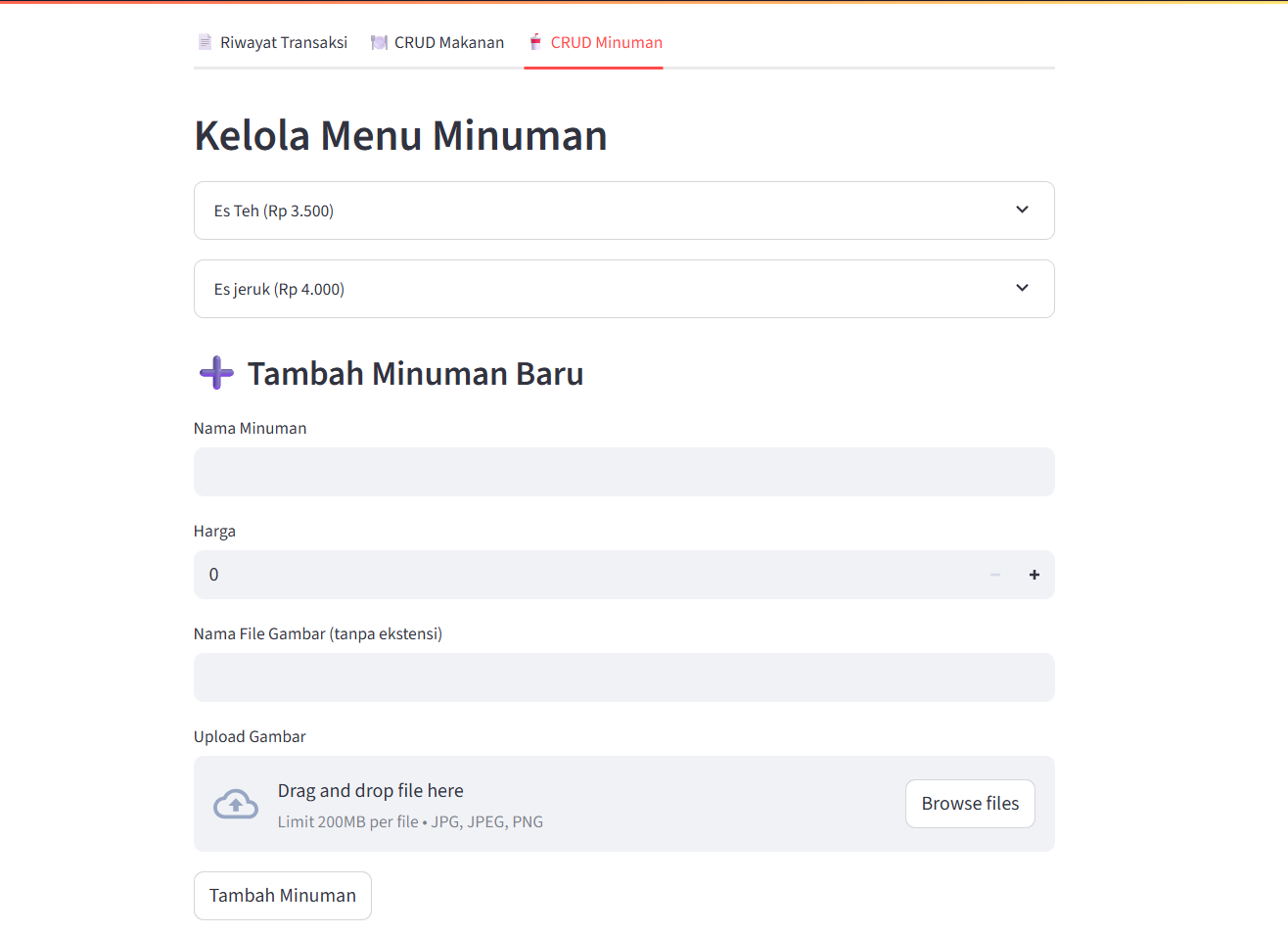
* Admin juga bisa mengedit menu yang sudah ada







* **Admin Page – CRUD Minuman**
* Pada bagian ini admin bisa melakukan CRUD ke menu minuman pada aplikasi kasir



* Admin juga bisa mengedit menu yang sudah ada

