Deskripsi Aplikasi: SmartUMKM

Aplikasi CRUD (Create, Read, Update, Delete) ini adalah aplikasi manajemen data yang dikembangkan untuk mendukung operasional bisnis SmartUMKM, sebuah software house yang berfokus pada solusi digital untuk UMKM (Usaha Mikro, Kecil, dan Menengah). Aplikasi ini dirancang untuk mempermudah manajemen data Client, Project, dan Invoice bagi para pelanggan dan tim internal.Aplikasi ini menawarkan fitur-fitur berikut:

1. **Manajemen Client:**
    - Menambah, mengelola, dan menghapus data klien seperti nama, jenis bisnis, alamat, nomor telepon, dan email.
    - Mempermudah pengelolaan data klien yang penting untuk proses proyek.
2. **Manajemen Project**:
    - Menambah, menampilkan, dan menghapus data proyek.
    - Pengelolaan proyek meliputi nama proyek, deskripsi, anggaran, dan status proyek (misalnya, "pending," "in progress," atau "completed").
3. **Manajemen Invoice**:
    - Menambah, mengelola, dan menghapus tagihan (invoice).
    - Mengelola nomor invoice, jumlah tagihan, dan status pembayaran (misalnya, "unpaid," "paid").

Tema Aplikasi SmartUMKM

Aplikasi ini mengusung tema **manajemen bisnis digital untuk UMKM**, dengan fokus pada kemudahan penggunaan dan fungsionalitas yang efisien. Desain aplikasi ini sederhana dan efektif, mengutamakan kepraktisan dalam setiap interaksi dengan pengguna. Berikut adalah beberapa elemen tema yang diterapkan:

1. **Desain Minimalis dan Bersih:**
   - Tampilan aplikasi menggunakan warna gelap (hitam) untuk latar belakang dan warna netral untuk tombol, menciptakan kontras yang nyaman untuk dilihat dan digunakan dalam waktu lama.
   - Desain UI yang sederhana, hanya menampilkan elemen-elemen yang diperlukan, seperti tombol navigasi, form input, dan daftar data yang dikelola.
2. **Navigasi yang Mudah:**
   - Menu utama memberikan akses mudah ke tiga kategori utama aplikasi: Clients, Projects, dan Invoices.
   - Pengguna dapat dengan cepat berpindah antar kategori untuk mengelola data mereka.
3. **Pengelolaan Data yang Terorganisir:**
   - Aplikasi ini menggunakan pendekatan CRUD untuk memudahkan pengguna dalam menambah, mengubah, menghapus, dan menampilkan data terkait klien, proyek, dan tagihan.
4. **UI yang Ramah Pengguna:**
   - Desain antarmuka yang jelas dan bersih memudahkan pengguna dari berbagai latar belakang untuk mengoperasikan aplikasi tanpa kesulitan.
   - Setiap kategori memiliki tampilan form untuk input data baru dan daftar data yang sudah ada, yang dapat dilihat atau dihapus dengan mudah.
5. **Mendukung Operasional Bisnis:**
   - Aplikasi ini membantu **SmartUMKM** dalam mengelola berbagai aspek bisnisnya, dari mengelola data klien dan proyek hingga memonitor status tagihan. Fungsionalitas ini mendukung efisiensi dan produktivitas dalam operasional perusahaan.

Penjelasan Teknis Implementasi Konsep OOP dalam Kode

1. **Contoh implementasi Encapsulation:**
   a. **Class BaseModel:**
      - Kelas BaseModel adalah kelas induk yang menyimpan logika dasar untuk operasi CRUD (Create, Read, Update, Delete) untuk data. Kelas ini memiliki metode **get_all**, **create**, dan **delete**, yang bertanggung jawab untuk mengakses dan mengubah data dalam database.
      - **Encapsulation** dilakukan dengan mendeklarasikan atribut seperti table_name yang diatur di dalam kelas dan tidak dapat diakses langsung dari luar, namun dapat diakses melalui metode kelas seperti create, get_all, dan delete

```python
class BaseModel:
    table_name = ""

    @classmethod
    def get_all(cls):
        conn, cursor = connect_db()
        cursor.execute(f"SELECT * FROM {cls.table_name}")
        rows = cursor.fetchall()
        conn.close()
        return rows

    @classmethod
    def create(cls, **kwargs):
        conn, cursor = connect_db()
        columns = ', '.join(kwargs.keys())
        values = ', '.join(f"'{v}'" for v in kwargs.values())
        cursor.execute(f"INSERT INTO {cls.table_name} ({columns}) VALUES ({values})")
        conn.commit()
        conn.close()

    @classmethod
    def delete(cls, id):
        conn, cursor = connect_db()
        cursor.execute(f"DELETE FROM {cls.table_name} WHERE id = {id}")
        conn.commit()
        conn.close()
```

b. Class Client, Project, dan Invoice:

- Kelas Client, Project, dan Invoice mewarisi metode dari kelas BaseModel. Setiap objek ini menyimpan data terkait (seperti nama, deskripsi, anggaran untuk Project) dan memiliki metode save() untuk menyimpan data ke dalam database.

```python
class Client(BaseModel):
    table_name = "clients"

    def __init__(self, name, business_type, address=None, phone=None, email=None):
        self.name = name
        self.business_type = business_type
        self.address = address
        self.phone = phone
        self.email = email

    def save(self):
        self.create(name=self.name, business_type=self.business_type, address=self.address, phone=self.phone, email=self.email)
```

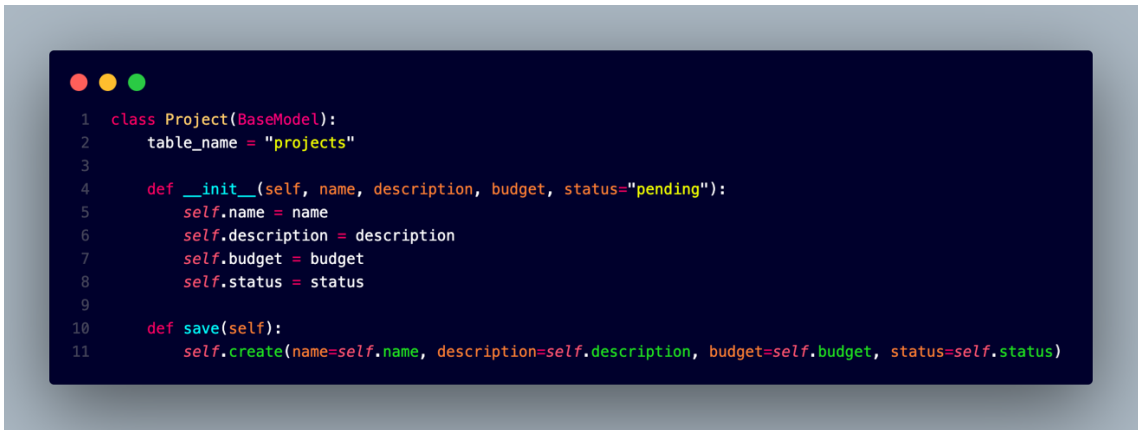
2. Contoh implementasi Inheritance:

a. Class BaseModel sebagai kelas induk:

- Kelas BaseModel menyimpan logika dasar untuk operasi CRUD yang dapat digunakan oleh semua objek lain (seperti Client, Project, dan Invoice) tanpa harus menduplikasi kode.

b. Class Client, Project, dan Invoice sebagai kelas turunan:

- Client, Project, dan Invoice mewarisi metode dari BaseModel. Setiap kelas ini memiliki metode save() untuk menyimpan data ke dalam database.

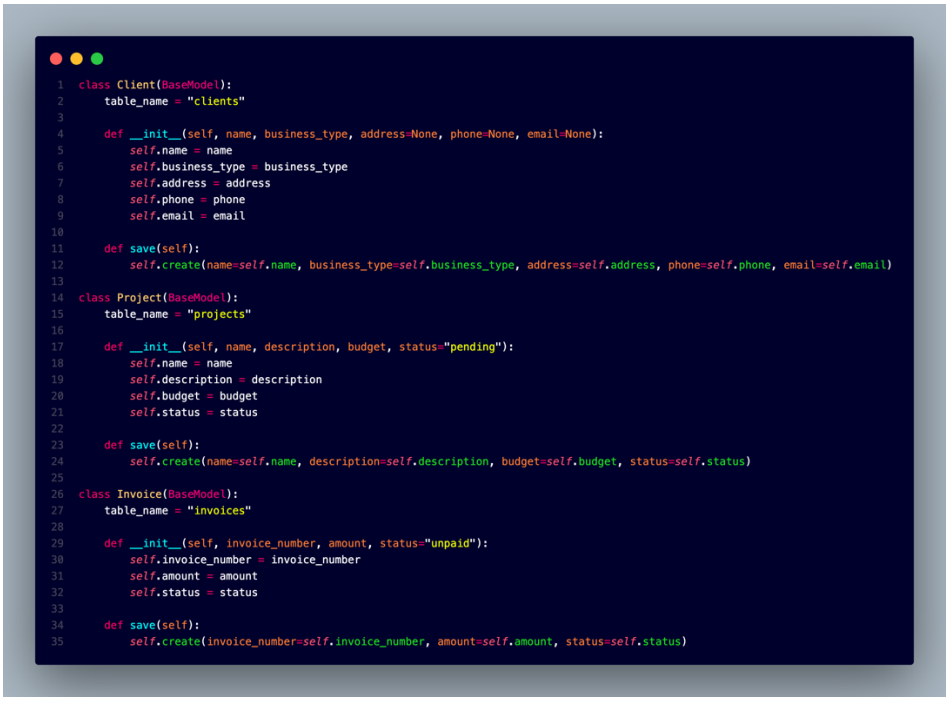


```python
class Project(BaseModel):
    table_name = "projects"

    def __init__(self, name, description, budget, status="pending"):
        self.name = name
        self.description = description
        self.budget = budget
        self.status = status

    def save(self):
        self.create(name=self.name, description=self.description, budget=self.budget, status=self.status)
```

3. **Contoh implementasi Polymorphism:**

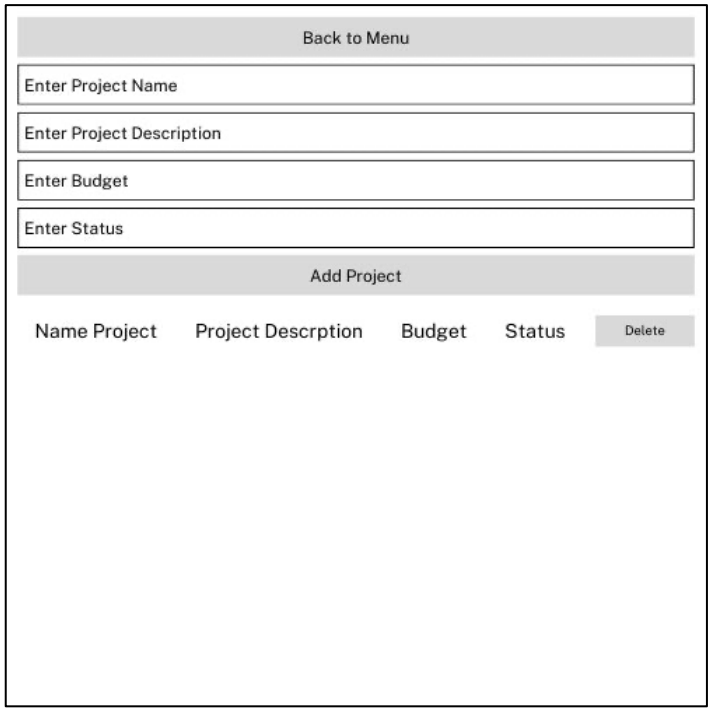
   a. **Polymorphism pada metode save():**

   - Semua kelas turunan (Client, Project, Invoice) memiliki metode save(), tetapi implementasinya berbeda, menyesuaikan dengan struktur data yang dimiliki oleh masing-masing kelas.

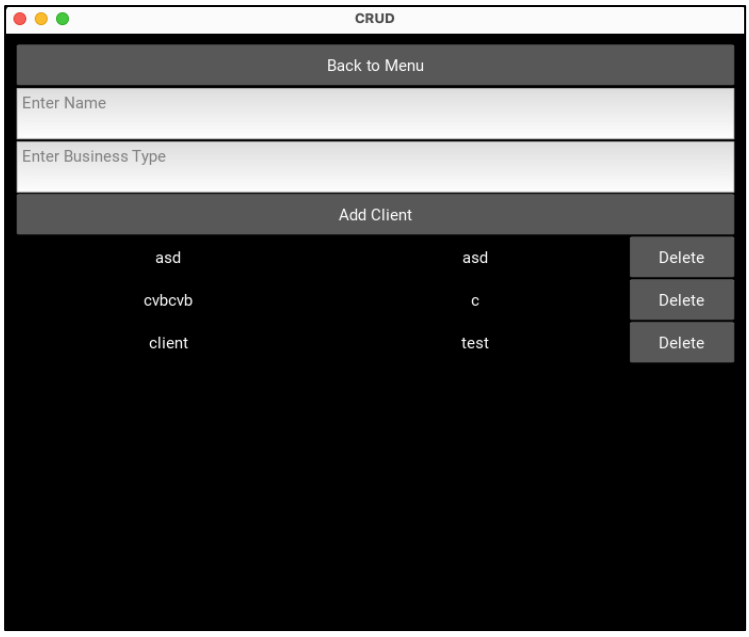


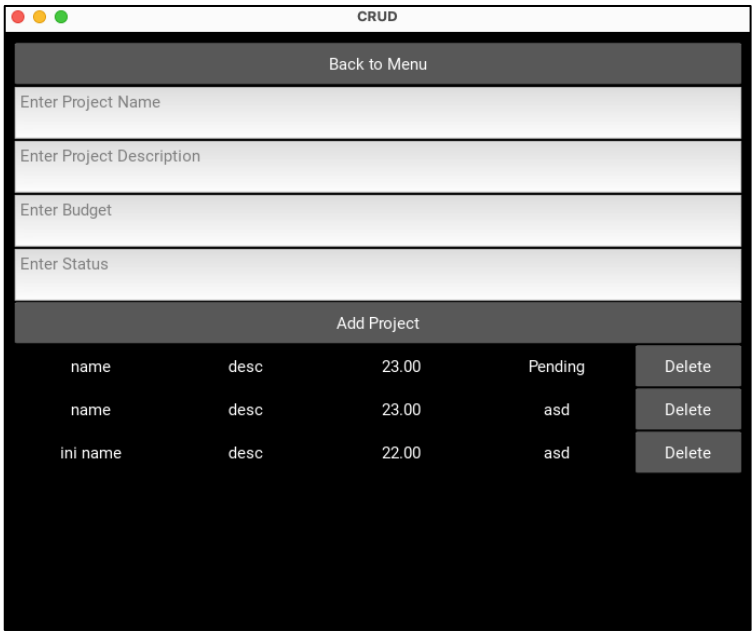
```python
class Client(BaseModel):
    table_name = "clients"

    def __init__(self, name, business_type, address=None, phone=None, email=None):
        self.name = name
        self.business_type = business_type
        self.address = address
        self.phone = phone
        self.email = email

    def save(self):
        self.create(name=self.name, business_type=self.business_type, address=self.address, phone=self.phone, email=self.email)

class Project(BaseModel):
    table_name = "projects"

    def __init__(self, name, description, budget, status="pending"):
        self.name = name
        self.description = description
        self.budget = budget
        self.status = status

    def save(self):
        self.create(name=self.name, description=self.description, budget=self.budget, status=self.status)

class Invoice(BaseModel):
    table_name = "invoices"

    def __init__(self, invoice_number, amount, status="unpaid"):
        self.invoice_number = invoice_number
        self.amount = amount
        self.status = status

    def save(self):
        self.create(invoice_number=self.invoice_number, amount=self.amount, status=self.status)
```

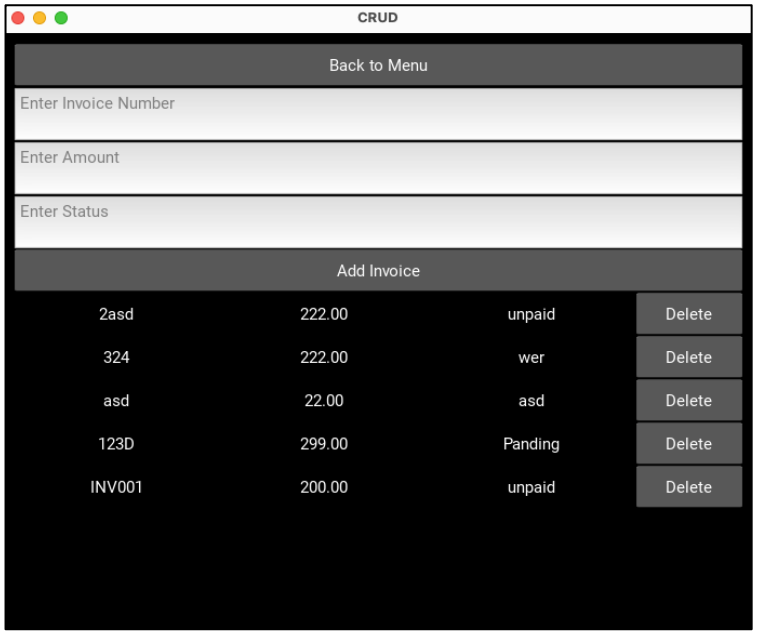
Wireframe Aplikasi



Screenshot Implementasi UI Di Kivy.







Kesimpulan

Proyek SmartUMKM adalah aplikasi manajemen untuk bisnis kecil dan menengah yang memungkinkan pengguna untuk mengelola klien, proyek, dan invoice. Aplikasi ini dibangun dengan menerapkan konsep Object-Oriented Programming (OOP), menggunakan Kivy untuk antarmuka pengguna dan SQLite untuk penyimpanan data. Dengan konsep Encapsulation, Inheritance, dan Polymorphism, aplikasi ini mudah dikembangkan, dan efisien dalam pengelolaan data. Melalui penerapan OOP, aplikasi ini tidak hanya terstruktur dengan baik, tetapi juga memungkinkan penambahan fitur dengan mudah di masa depan.

Refleksi Pembelajaran

Melalui pembuatan aplikasi ini, saya memperoleh pemahaman yang lebih dalam tentang penerapan OOP dalam pengembangan perangkat lunak, serta bagaimana membangun aplikasi yang terstruktur dan mudah dipelihara. Saya juga belajar tentang cara mengelola antarmuka pengguna dengan Kivy dan mengintegrasikan aplikasi dengan SQLite untuk pengelolaan data. Tantangan utama adalah memastikan validasi data dan antarmuka pengguna yang intuitif, namun pengalaman ini sangat berharga dalam mengasah keterampilan pemrograman dan pengembangan aplikasi berbasis data.