Rust Project: Budget Manager

Main Project Structure
- main.rs: Entry point of the application
- budget.rs: Handles budgets
- transaction.rs: Manages transactions
- database.rs: SQLite connection setup
- ui.rs: Handles user interaction and CLI menus

Main Code

main.rs
```
mod budget;
mod transaction;
mod database;
mod ui;

use database::init_db;

fn main() {
    let conn = init_db().expect("Database connection failed");
    loop {
        match ui::menu() {
            0 => ui::add_budget(&conn),
            1 => ui::view_budgets(&conn),
```

```
            2 => ui::add_transaction(&conn),
            3 => ui::view_transactions(&conn),
            4 => break,
            _ => println!("Invalid option"),
        }
    }
}
```

budget.rs
```
use serde::{Deserialize, Serialize};

#[derive(Debug, Serialize, Deserialize)]
pub struct Budget {
    pub id: i32,
    pub name: String,
    pub total: f64,
    pub remaining: f64,
}

impl Budget {
    pub fn new(name: String, total: f64) -> Self {
        Self {
            id: 0,
            name,
            total,
```

```
            remaining: total,
        }
    }
}
```

transaction.rs
```
use serde::{Deserialize, Serialize};

#[derive(Debug, Serialize, Deserialize)]
pub struct Transaction {
    pub id: i32,
    pub budget_id: i32,
    pub description: String,
    pub amount: f64,
}

impl Transaction {
    pub fn new(budget_id: i32, description: String, amount: f64) -> Self
{
        Self {
            id: 0,
            budget_id,
            description,
            amount,
        }
```

```
    }
}
```

database.rs

```
use rusqlite::{Connection, Result};

pub fn init_db() -> Result<Connection> {
    let conn = Connection::open("budget_manager.db")?;
    conn.execute(
        "CREATE TABLE IF NOT EXISTS budgets (
            id INTEGER PRIMARY KEY,
            name TEXT NOT NULL,
            total REAL NOT NULL,
            remaining REAL NOT NULL
        )",
        [],
    )?;
    conn.execute(
        "CREATE TABLE IF NOT EXISTS transactions (
            id INTEGER PRIMARY KEY,
            budget_id INTEGER NOT NULL,
            description TEXT NOT NULL,
            amount REAL NOT NULL,
            FOREIGN KEY(budget_id) REFERENCES budgets(id)
        )",
```

```
        [],
    )?;
    Ok(conn)
}
```

ui.rs
```
use dialoguer::{Input, Select};
use rusqlite::Connection;
use crate::budget::Budget;
use crate::transaction::Transaction;

pub fn menu() -> usize {
    let options = vec![
        "Add a budget",
        "View budgets",
        "Add a transaction",
        "View transactions",
        "Exit",
    ];
    Select::new()
        .with_prompt("Choose an option")
        .items(&options)
        .interact()
        .unwrap()
}
```

```
pub fn add_budget(conn: &Connection) {
            let name: String = Input::new().with_prompt("Budget name").interact().unwrap();
                let total: f64 = Input::new().with_prompt("Total amount").interact().unwrap();
    let budget = Budget::new(name, total);
    conn.execute(
        "INSERT INTO budgets (name, total, remaining) VALUES (?1, ?2, ?3)",
        &[&budget.name, &budget.total, &budget.remaining],
    )
    .expect("Error adding budget");
    println!("Budget added successfully!");
}

pub fn view_budgets(conn: &Connection) {
     let mut stmt = conn.prepare("SELECT id, name, total, remaining FROM budgets").expect("Error");
    let budgets = stmt.query_map([], |row| {
        Ok(Budget {
            id: row.get(0)?,
            name: row.get(1)?,
            total: row.get(2)?,
            remaining: row.get(3)?,
        })
    }).expect("Error mapping data");
```

```
    for budget in budgets {
        let budget = budget.unwrap();
            println!("ID: {}, Name: {}, Total: {}, Remaining: {}", budget.id, budget.name, budget.total, budget.remaining);
    }
}
```

How to Run

1. Compile the project: `cargo build`
2. Run: `cargo run`