You're right; my earlier suggestion wouldn't resolve the deadlock issue. Let's take a different approach to initialize the database without causing dea

Here's an updated strategy:

1. **Constructor**: We'll avoid any blocking calls within the constructor and use an asynchronous initialization method.
2. **Ensure the Table is Created**: We'll initialize the database once and ensure it completes before making any database calls.

Updated ExpenseService

Csharp Copy

```
public class ExpenseService : IExpenseService
{
    private readonly SQLiteAsyncConnection _connection;
    private readonly Task _initTask;

    public ExpenseService(ISqliteDb db)
    {
        _connection = db.GetConnection();
        _initTask = InitializeDatabaseAsync();
    }

    private async Task InitializeDatabaseAsync()
    {
        await _connection.CreateTableAsync<Expense>();
    }

    public async Task<IEnumerable<Expense>> GetExpensesAsync()
    {
        await _initTask; // Ensure the initialization is complete
        return await _connection.Table<Expense>().ToListAsync();
    }
}
```

Dependency Injection Setup

Csharp Copy

```
var builder = MauiApp.CreateBuilder();
builder.Services.AddScoped<IExpenseService, ExpenseService>();
builder.Services.AddScoped<ISqliteDb, LocalDbService>();
```

By using `_initTask`, we ensure that the initialization is awaited before any database operations are performed. This approach guarantees that the

This should help avoid the deadlock while ensuring the table is created properly. Give this a try and let me know if it resolves the issue! If further fin tuning is needed, we can tackle it together