# Description

The project requires two fundamental entities: Accounts and Transactions.

Accounts support following operations:

1. Create: Add new account to the database
2. Edit: Edit account name and current balance
3. Delete: Instead of deleting an account, this marks the account as Inactive. This ensures that account information is available to the transactions table.

Transactions supports following operations:

1. Create a new transfer between accounts
2. Display all

The project was built using .NET Core 6. It uses Entity Framework with MVC. SQLite is used to simplify testing.

SQLite does not enforce strict data type for columns. In addition, SQLite does not support primary key - foregin key constraint by default. (PRAGMA foregin\_keys enables support this feature)

## Steps

1. Create new ASP.Net Core MVC project in Visual studio code.

dotnet new mvc -o Banking\_PK

1. Disable HTTPS mode by commenting app.UseHttpsRedirection(); in Program.cs
2. Add controller for Accounts (Controllers/AccountController.cs) and Transactions (Controllers/TransactionController.cs). Also add DbContext for managing the data. Add appropriate DbContext to builder.Services.AddDbContext(...)
3. Add Index, Details, Create, Edit, and Delete method in AccountController
4. Since transactions only support Create and Details, create those methods along with Index method in TransactionController.cs. *Note that default views and controller methods can be automatically created via commandline.*
5. Add model classes for Account and Transaction. In addition, create TransferViewModel for handling account transfers and create TransactionViewModel for rendering transaction details.
6. Populate views for each of the supported methods and link views in home page.
7. Setup data migration using steps listed in next section

## Data migration

[ASP.NET MVC Tutorial > Add a model](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-6.0&tabs=visual-studio-code) for more information on data migration steps.

Install/Update appropriate packages when using Visual Studio Code. Visual Studio provides interactive UI for these.

dotnet tool uninstall --global dotnet-aspnet-codegenerator  
dotnet tool install --global dotnet-aspnet-codegenerator  
dotnet tool uninstall --global dotnet-ef  
dotnet tool install --global dotnet-ef  
dotnet add package Microsoft.EntityFrameworkCore.Design  
dotnet add package Microsoft.EntityFrameworkCore.SQLite  
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design  
dotnet add package Microsoft.EntityFrameworkCore.SqlServer

**OPTIONAL:** Controller code can be auto generated using dotnet-aspnet-codegenerator:

dotnet-aspnet-codegenerator controller \  
 -name AccountController \  
 -m Account \  
 -dc BankingContext \  
 --relativeFolderPath Controllers \  
 --useDefaultLayout \  
 --referenceScriptLibraries -sqlite

-force to overwrite existing controller

Run the actual migration using

dotnet ef migrations add InitialCreate  
dotnet ef database update

## Business logic

### Accounts

Account model class adds Required attribute to various properties. It also adds DisplayName attribute to the properties. For Name, AllowEmptyStrings = false is set. No length limit (minimum/maximum) is set for string properties.

It might be advisable to set the length property to avoid issues with handling large text sizes.

Minimal changes are required for views associated with accounts.

### Transactions

* The Transaction class stores account ID for source and destination accounts.
* To display the list of transactions, TransactionViewModel class is defined. It contains source and destination account names instead of IDs. **Note that transactions associated with deleted accounts are listed as well.**
* To create a new transaction, the view needs to list all (active) accounts. A TransferViewModel class is added to hold this information. It also passes the user input back to the controller.
* All the business logic for creating a transaction is added in the controller.

## SQL schema

For SQL server, the schema can be created using following SQL statement

USE briley  
GO  
  
CREATE TABLE [ACCOUNT]  
(  
 Id INT NOT NULL IDENTITY PRIMARY KEY,  
 Name VARCHAR(50) NOT NULL,  
 Balance DECIMAL NOT NULL,  
 Active BINARY NOT NULL  
);  
  
CREATE TABLE [TRANSACTION]  
(  
 Id INT NOT NULL IDENTITY PRIMARY KEY,  
 Source INT NOT NULL  
 CONSTRAINT SRC\_FK REFERENCES ACCOUNT,  
 Destination INT NOT NULL  
 CONSTRAINT DEST\_FK REFERENCES ACCOUNT,  
 TransactionTime DATETIME NOT NULL,  
 SourceBalance DECIMAL NOT NULL,  
 DestinationBalance DECIMAL NOT NULL,  
 TransferAmount DECIMAL NOT NULL  
);  
  
ALTER TABLE [TRANSACTION] ADD CONSTRAINT CHK\_TRANSACTION\_AMOUNT  
 CHECK(TransferAmount >= 0  
 AND TransferAmount <= 10000  
 AND TransferAmount <= [TRANSACTION].SourceBalance);  
  
  
ALTER TABLE [TRANSACTION] ADD CONSTRAINT CHK\_ACCOUNT\_ID  
 CHECK(Source <> [TRANSACTION].Destination);  
GO