Hi my dears punchers, lets create a solution on dotnet core, using visual studio 2019,

We are going to create a spence money management, to planning in what we are wasting our money.

We will start with a basic crud using dot net core, then we are going to advance to a better organization of the code, after that we are going to separate the logic, database and the core of the project, them we are going to create a secure API to consume our data, and finally a Xamarin APP, so now, lets begin.

First lets know something of the base of this project,

We are going to have something than its going to be WALLET, than will represent our separated entities than we are going to have, example,

Company ABC,

Personal Money,

Company XYZ

My Wife Money

Home Shared Money

And an account, than will emulate an accounting account, this will be  
BHD Credit Card USD

BHD Credit Card DOP

Checking Account BanReservas

Saving Account USD Popular,

Cash Money Germosen,

Cash Money ARA and so on

With this two tables we are going to start the major part of the tutorial, then at the finish we are going to introduce 8 more tables

Operation, Shop,Tags, Users, and more,

So, lets begin

On the MVC (model, view, Controller) we have a change of mind for some developers from the old school.

We have the Model, than represent our database table,

Views than represent or display (screen) information, I mean, what we show to the user, or what the user is seeing, and the controller, than process all the information.

The user do something on the view, and after an submit it send a model (or viewmodel) to the controller, and this is who make the iteration with the model, than is processed and updated, inserted, deleted, or listed to the database, so, lets create or first model

Something to have in mind, on the MVC patron, we have three ways to have a model, first is create our tables on the database and store to our application (Database first) another is create our models on the application and this will create our database (code first) the other one (model first, I never have understand it, so, you can google it,

On this tutorial we are going to use code first,

Our tables are going to be, just a class, and the fields of our tables are going to be properties

Now lets do magic :D

Lets create a controller, with the wizard, to have on simples steps, all the basic operations than we use to have on an application, Create Read Update and Delete (CRUD)

We have to introduce a DataContext, than will be our link or the communication, between our application and our database.

But, before do something we need to check one important thing

On the appsetting we need to configure our database connection, by default it create a local db connection, but I will use a SQL server database.

On this case we have a problem

Our database cant not be create it, if we don’t tell to the application what its going to create

On the manage nuggets packaged console, we can add files, nugget, or configuration to our project, so, lets create a migration file

Now lets update the database using the last migration file

So now lets run it

And now, lets create and relational the other table

We miss something

The account will depend from an wallet, I mean, a wallet can have multiples account on it,

So, we need to make a relationship

We don’t need to delete this, if we specify first, its better, because it will assume inmediatly the correct name of the table

After each creation of a table we need to run a migration

Lets see if everything its ok on our database

Im trying to write more than what the length than we specify but the validation don’t aceept it

That’s all we need to create the basic operations from tables on dot net core. In the next lesson we are going to make things better

The best practice use to say than we cant manipulate data directly on our controllers, because this is a little but insecure, so, let implement a patron called repository, than will be who its going to work or manipulate our data.

Contracts will be the interface than we need to implement eatch time than we want than controller make something with the database, and the interface it’s the responsible to localize the correct implementation

I mean,

The controller call the interface and the interface search the correct repository, maybe you will question, but why we didn’t call directly to the repository, well, we could, but, imagine than you want to test something specific on the future, you want to test a new way to create data, well, you don’t have to change all methods calls, you just create the new repository and tell to the interface (contract) where its going to search the implementation.

Lets see if everything continue ok

So, I need to create a contract and a repository to each model, but, you have to had something in mind, everytime than you need to copy something, you can obtimize it, so, lets see something

Those things are almost the same thing, so, lets combined to write and copy less on future tables

Lets see if nothing change

So now, lets see again if everything is ok

If you see, the wallet name, itsnt been showing

So, lets fix that

And that’s it

Lets fix it on the details too

Now, we have to do something, because, its not a good practice store the data members on the application, so, lets create a better structure

Database context will be our conection to the database

Domain will be our database tables structure

Models will be identically to our table structure, but, have something in mind, its not correct than you took your data models, I mean, this is the only part where its mandatory duplicate things,

Imagine than you need to change the database table, wallet, its not a int the id, now need to be an long, so, imagine than you have 50 application working than are using the int id, its stupid change those 50 applications if there are working well and those of them just consume data, so, for that reason you cant have directly your domain interacting with the app.

Common will store common classes and process

Services will store our services

Let’s make a trap here, to going testing

We are not going to use migrations on the client, so, remove that, maybe you will need to truncate the table too