Individual Assessment #1 - Banking Application

Lucile Pelou

Student n°74526

Git repository : https://github.com/Clarisse78/epita-ca1-74526

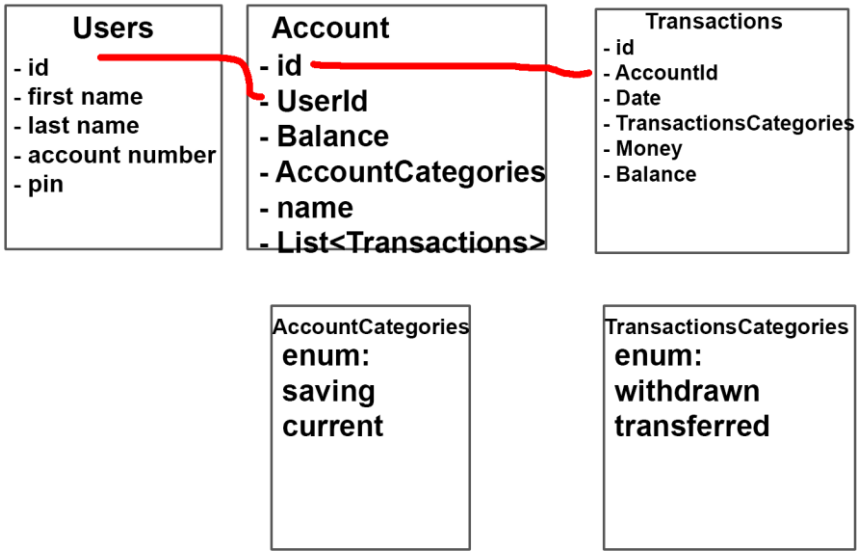
1) **First step: The structure**

I started the project by following tutorials on Youtube to better understand how to do a project of this type. During this working time, I created my first models, controllers and views. I did the basics with a class: AppUser, Transaction and BankAccount as well as enumerators for transaction types and bank account types.

For this, I created multiple folders:

- Models which contain all the models (AppUser, BankAccount and Transaction)
- Data with an Enum folder which contains all the enumerators

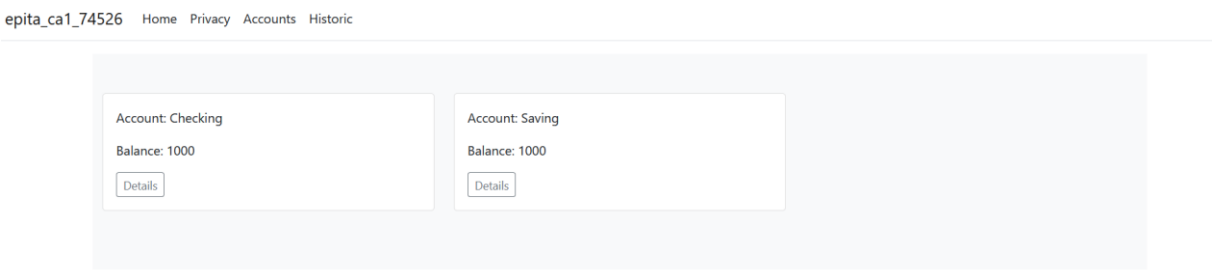
I also created the context of my database and a seed for it (which allowed me to test my application more simply) and I stored them in the Data folder. I also made diagrams to be able to have an idea of how to arrange the different classes, so this is how I saw the layout of the first 3 classes at the start:



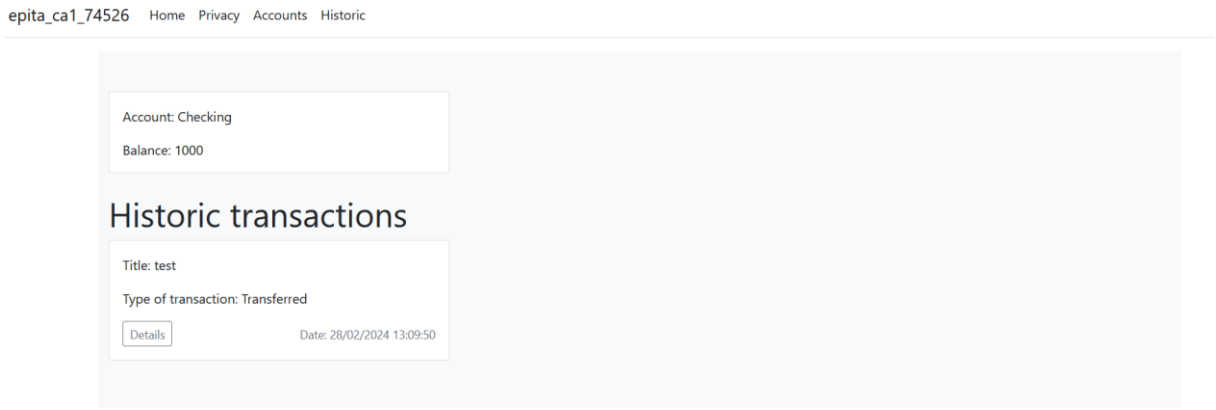
At the very beginning I did not implement having accounts (users and administrator). I first wanted to focus on the structure, i.e. creating bank accounts and displaying them, the same for transactions.

My application at this point looked like this:

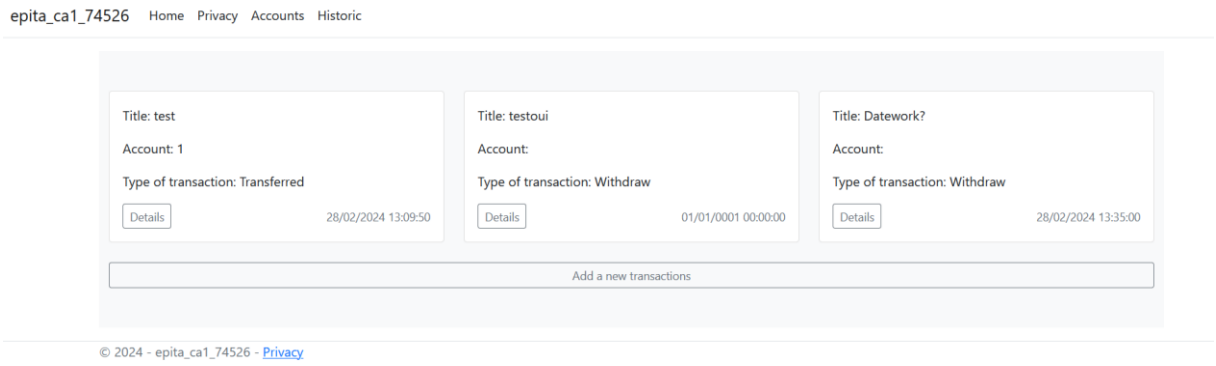
1) An accounts page where you could access to all the bank accounts created.



2) The detail button on the Accounts page led to more details on the account in question with a history of transactions for the chosen account.



3) The Historic page allowed you to see all transactions made regardless of the account concerned.



4) And finally, it was also possible to add a new transaction from the Historic page (but not yet linked to an account).

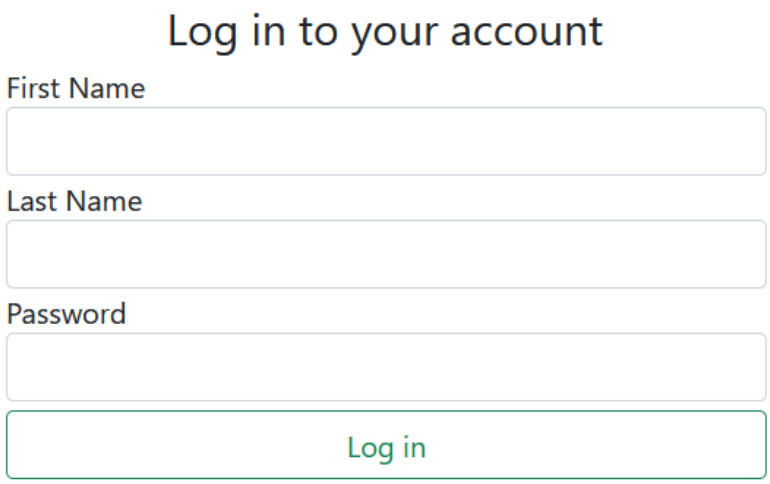
To facilitate the retrieval and addition of bank accounts and transactions I set up IAccountRepository and IBankAccountRepository interfaces stored in the interfaces folder with the repositories inheriting from these classes stored in the repository folder.

After doing that I had completed the first part of my application and I had laid the foundations for the future features. It was time to begin the second step which was the creation of user and administrator accounts.

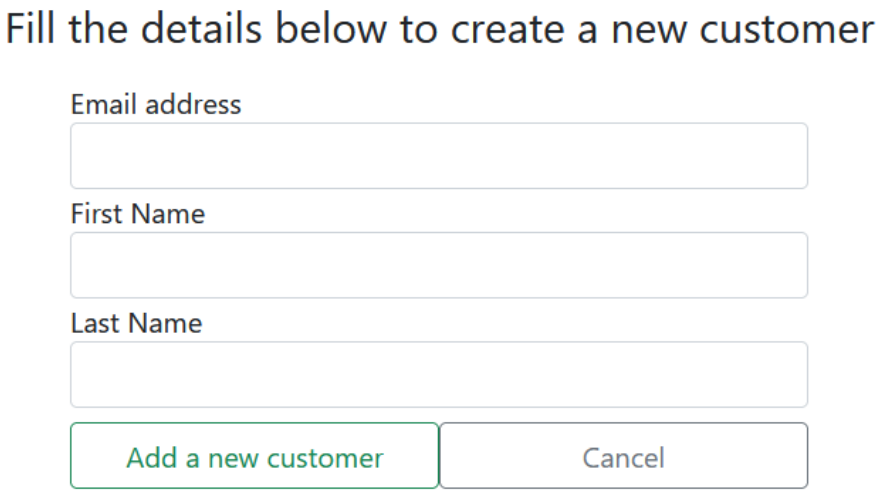
2) Second step: The Authentication (and connect all previous methods to this)

For this I used IdentityFramework. So, I created a UserRoles class that I placed in the data folder with two roles: User and Admin. I created two new “view” pages, login and register. While doing this, I discovered viewModels. At first, I didn't really understand their benefits but later I found them very practical and used them in almost all my future pages.

When I created my login page, I initially did not take into account the fact of being able to choose between customer and bank employee so this is what the page looked like at the start:

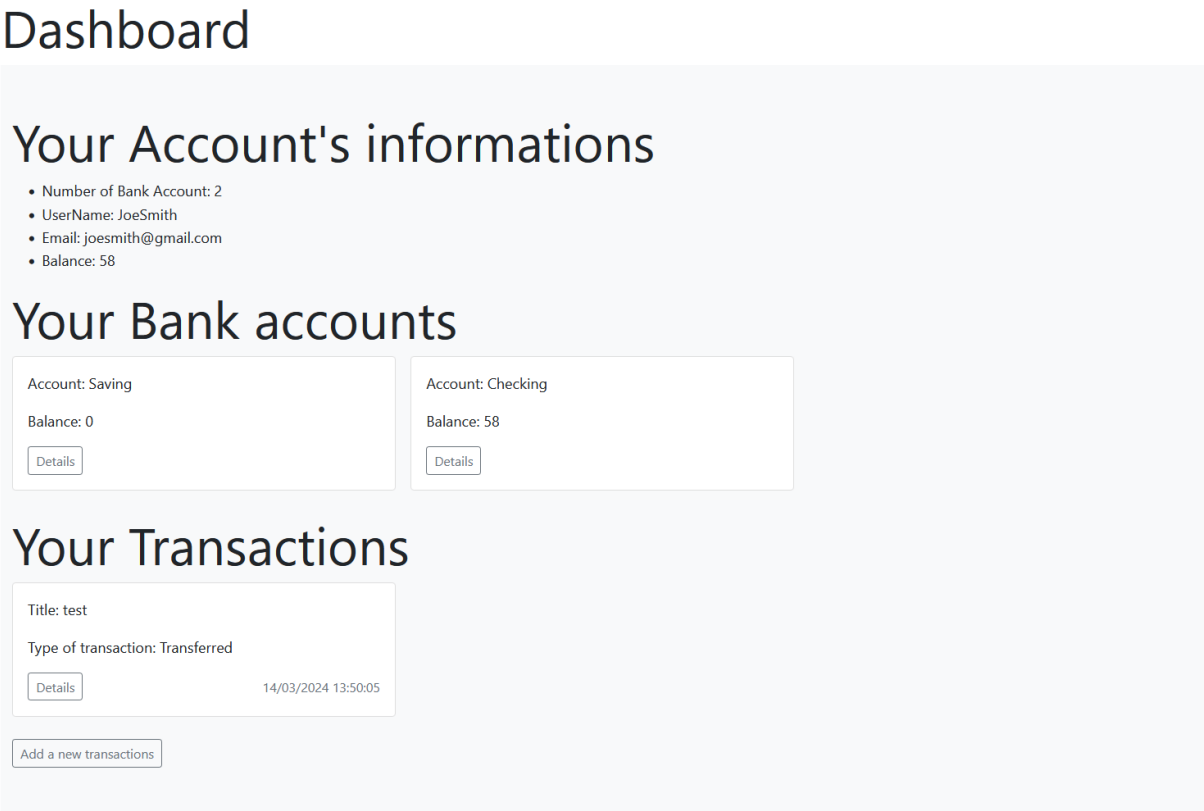


The register page allowed us to register a new account regardless of whether we were already logged in or not.

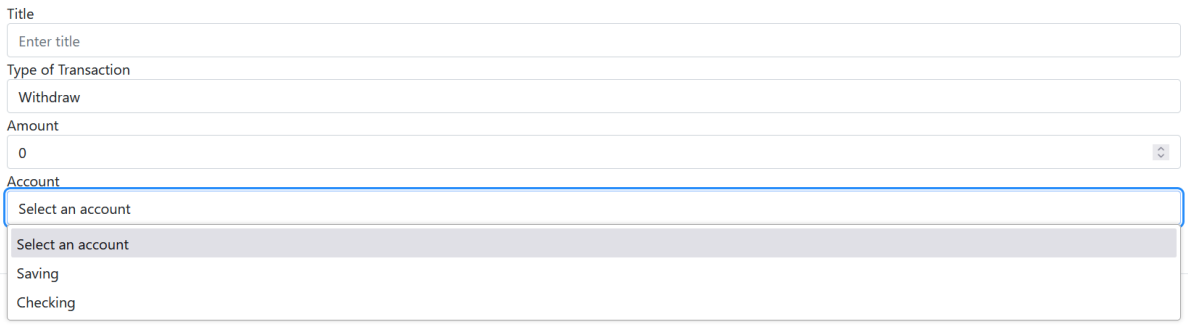


I decided to not create the pin with the bank account id but with the user id. This way the user can log in and see all their bank accounts and not log in to a specified bank account. I adapted the account number accordingly, so the checking account name ends with -C and the saving account name ends with -S. When a customer logs in, his PIN and two accounts are created for him.

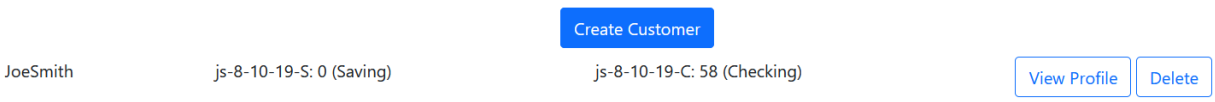
Once the connection logic was set up with the login page, I built a page called dashboard that allows customers to list all of their information. This page uses the same methods as previously with the addition of user ID information to retrieve only what concerns them.



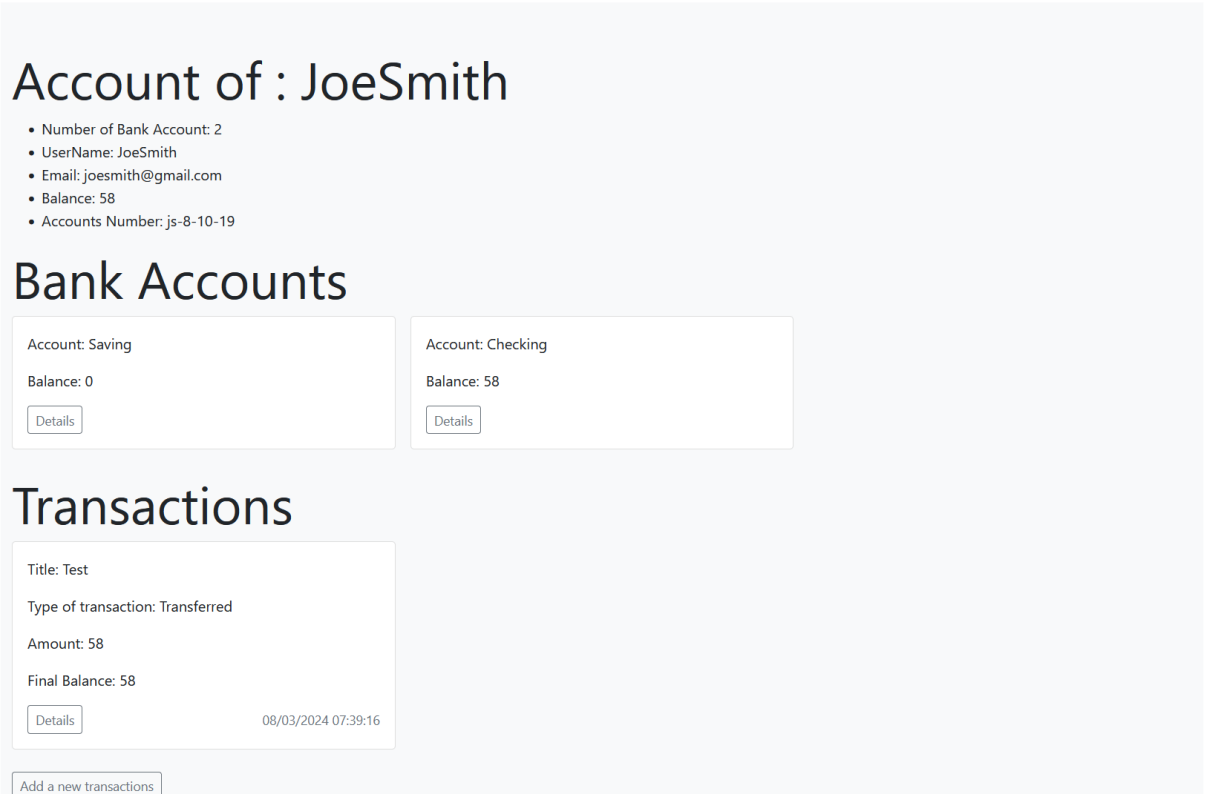
I linked this page with those made previously (details for transactions, details for bank accounts and add a new transaction). I also modified the page to add a new transaction so that the user can choose which of these accounts they want to make the transaction on.



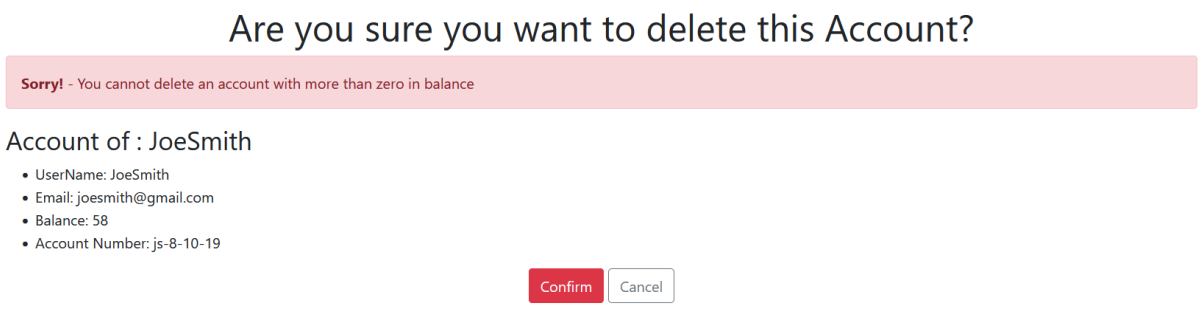
After finishing the customer view, I built the bank employee view. For this I made a User Lists page on which you can see the username and user accounts number with the respective balances. The bank employee can also access or delete each user's account.



By pressing view profile, you will get almost the same thing as the user gets in his dashboard:

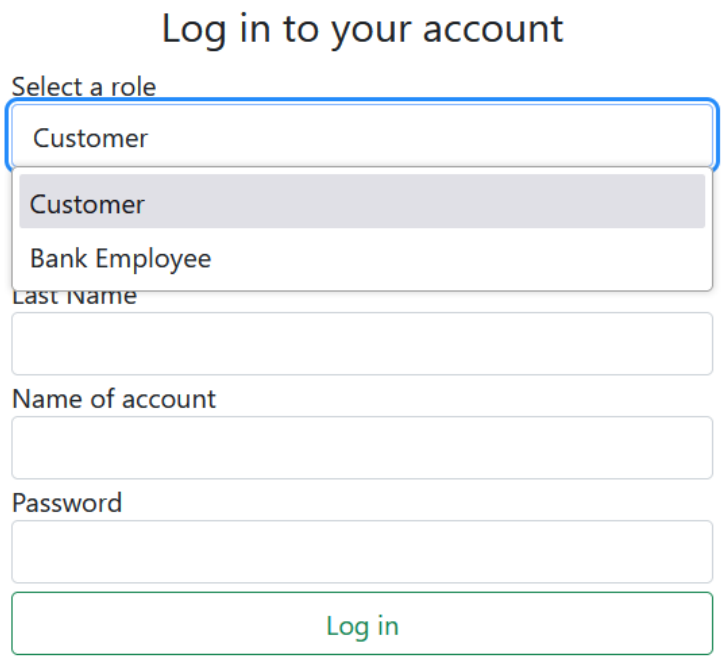


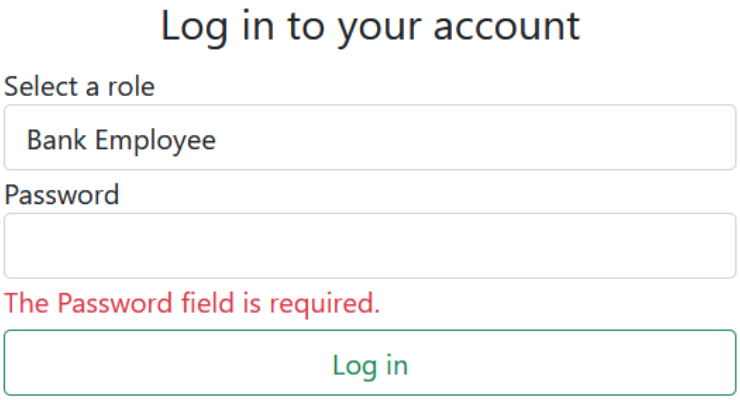
If bank employee presses delete and confirm, and the user concerned does not have zero currency then an error will be displayed:



To make it easier to set up these pages, as before, I created interfaces and repositories. Here one for the Users and one for the dashboard. ViewModels were also very useful for me to encapsulate different data and display everything I needed in the view.

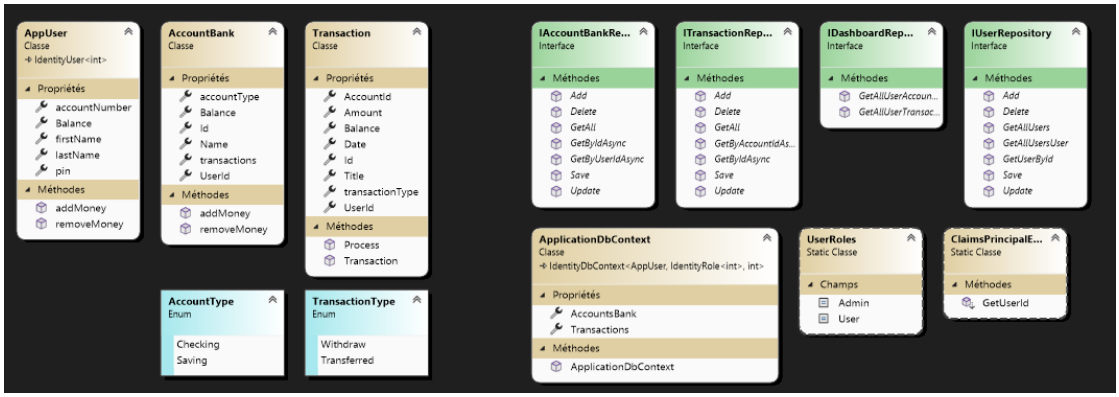
At this stage the application was almost finished! I added the possibility of choosing between bank employee or customer during login:





Finally, I added protections to prevent a non-connected user or a non-bank employee from being able to access pages that they should not. The application was therefore ready!

In the end, here is what the UML of my models, my interfaces and some other classes looks like:



Knowing that the AppUser user id and email is directly managed by IdentityUser and I also added the <int> specification so that the id is in integer.

In the end, I implemented a method to configure the database if it does not already exist. This way, everyone who has the project can launch it without problems!