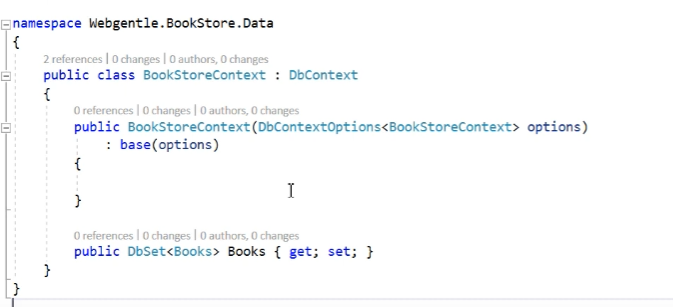
Code first approach:

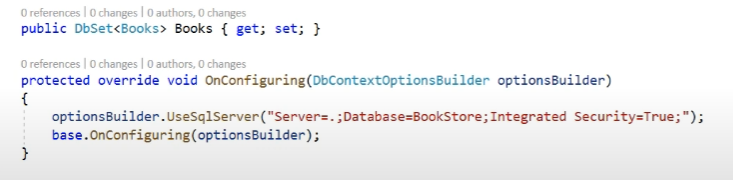
To keep the logic separated, we should use sperate folder for the DBContext. The class must extend the DbContext class. And we must pass the DbContextOption<> in the constructor and pass it to the base class.

As for the model we can use the model classes, but its better to create a new class inside the folder to keep everything separated.



There are two way to connect it to the database.

* First way is to configure it inside the class, then configure it on the startup file on ConfigureServices method.



* Second way is to define it inside the startup file. But if we do this, there is no need to use the configuration on the dbcontextclass.



But in real world application we get this connection string from the appsettings.json file. And that’s what we would do.

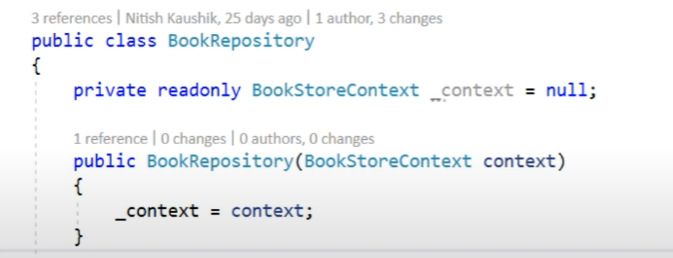
**Generating Database**

First we do command **Add-Migration [name here]** which will create the migration classes for us. To update the database with the class, we need to do command **update-database.** The Database will be created and it will automatically create a migration history table.

To update a model/add few more column to the table, we can define them in the model first. Then similar process, **add-migration name** followed by **update-database**

**Operating on Database**

After we create/edit our database and table we can use the dbcontext to operate on database. For that we need an instance of the dbcontext class. For that there is three ways, first is to create an instance when declaring the object, 2nd is assigning inside the constructor and 3rd is using dependency injection. Which is what I’ll use mostly.



Inserting the data:

