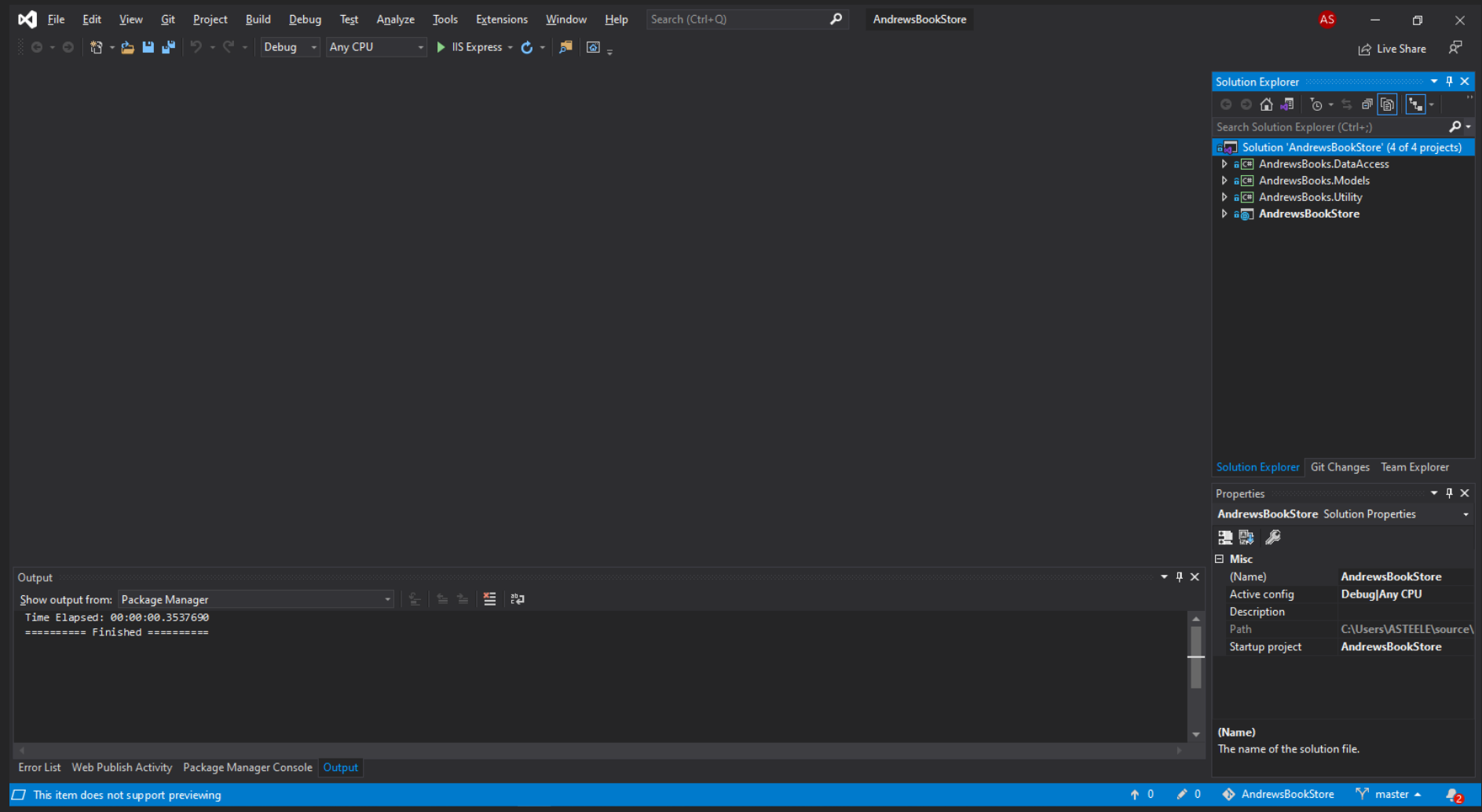
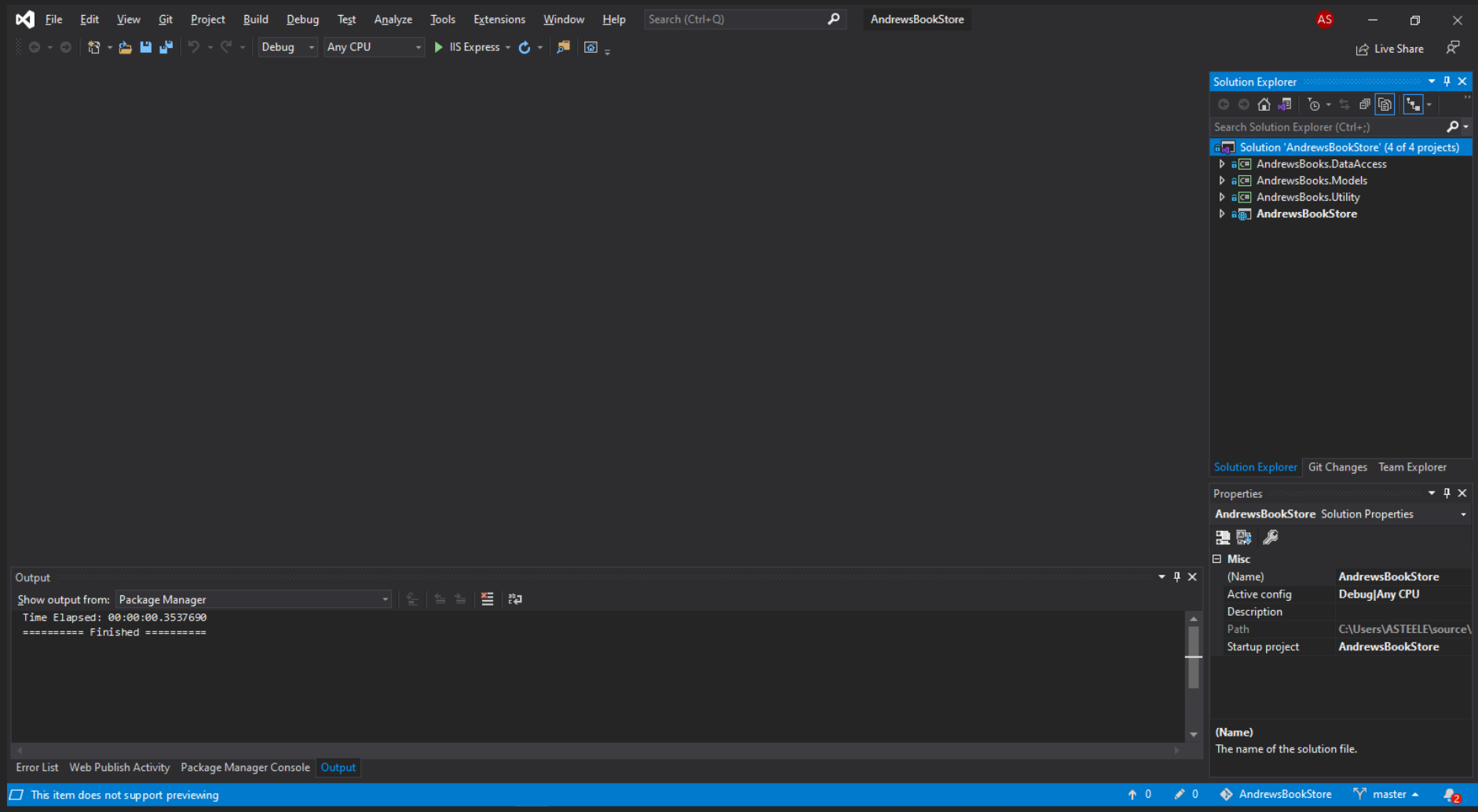
Build the Project (Part 2)

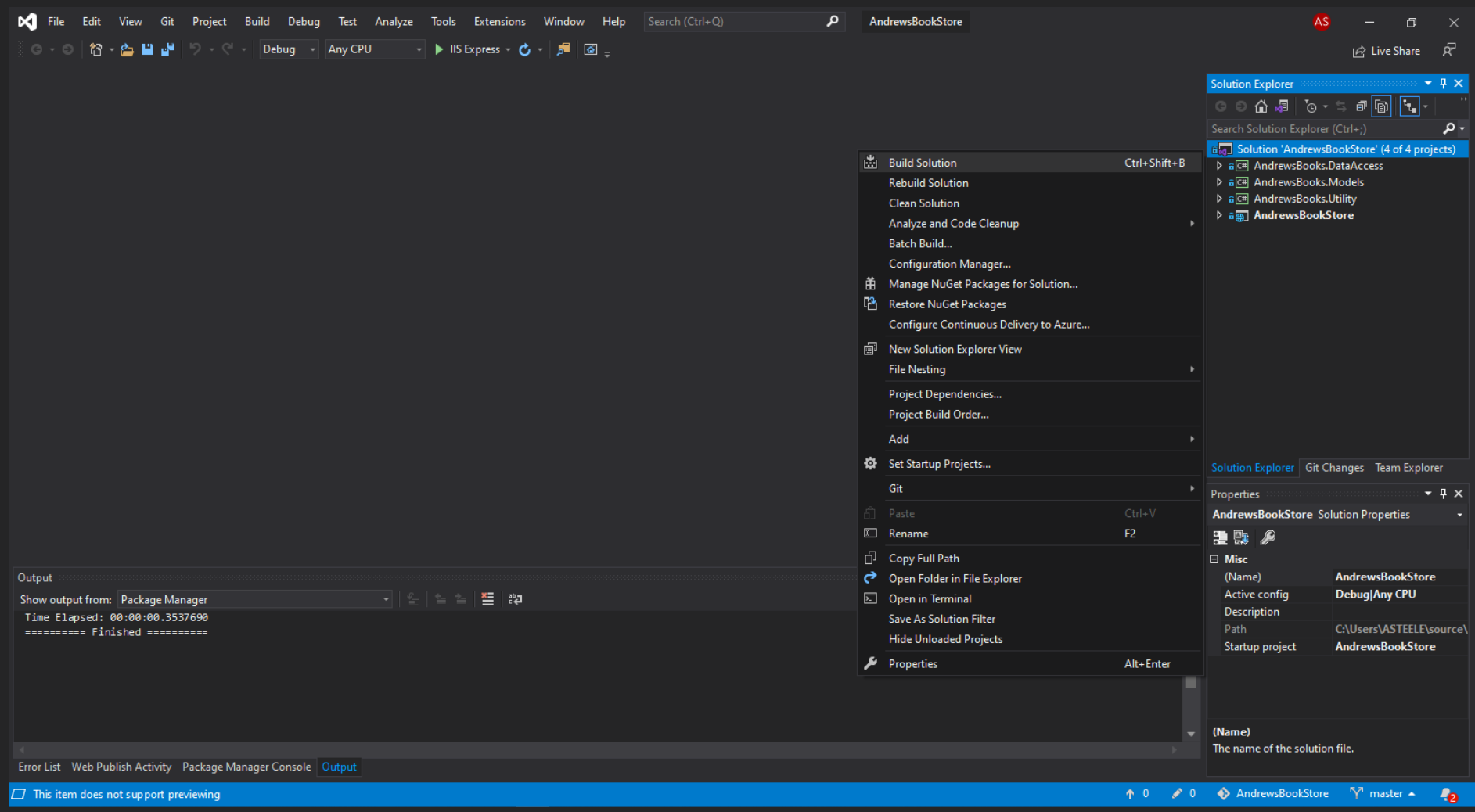
2.1 Create the DB


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
AndrewsBookStore
Debug
Any CPU
IIS Express
Live Share
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AndrewsBookStore' (4 of 4 projects)
AndrewsBooks.DataAccess
AndrewsBooks.Models
AndrewsBooks.Utility
AndrewsBookStore
Solution Explorer
Git Changes
Team Explorer
Properties
AndrewsBookStore Solution Properties
Misc
(Name) AndrewsBookStore
Active config Debug|Any CPU
Description
Path C:\Users\ASTEELE\source\
Startup project AndrewsBookStore
(Name)
The name of the solution file.
Output
Show output from: Package Manager
Time Elapsed: 00:00:00.3537690
========== Finished ==========
Error List
Web Publish Activity
Package Manager Console
Output
This item does not support previewing
0
0
AndrewsBookStore
master

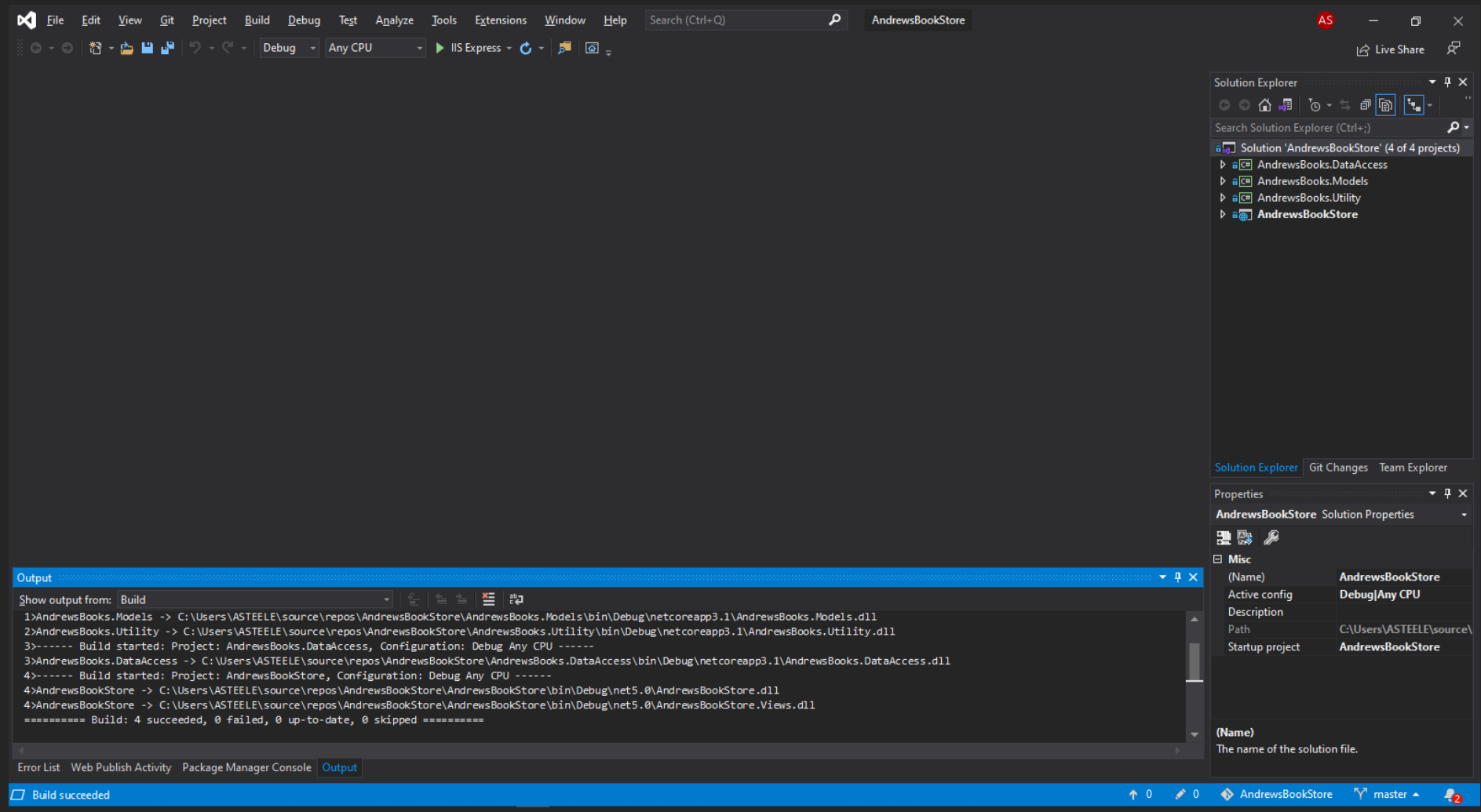
- To prepare for the next part, ‘build’ the application and confirm there are no errors



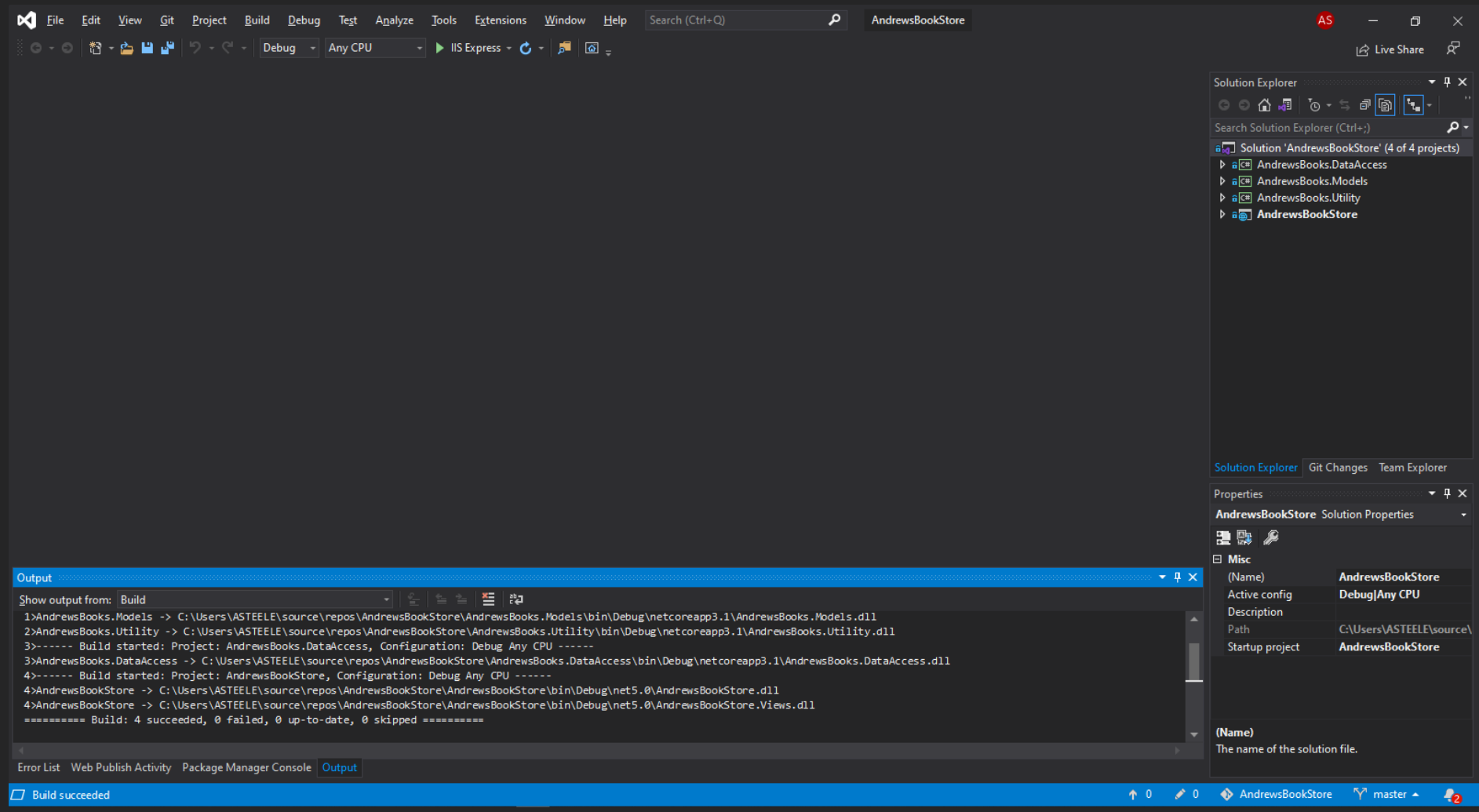
- To prepare for the next part, ‘build’ the application and confirm there are no errors



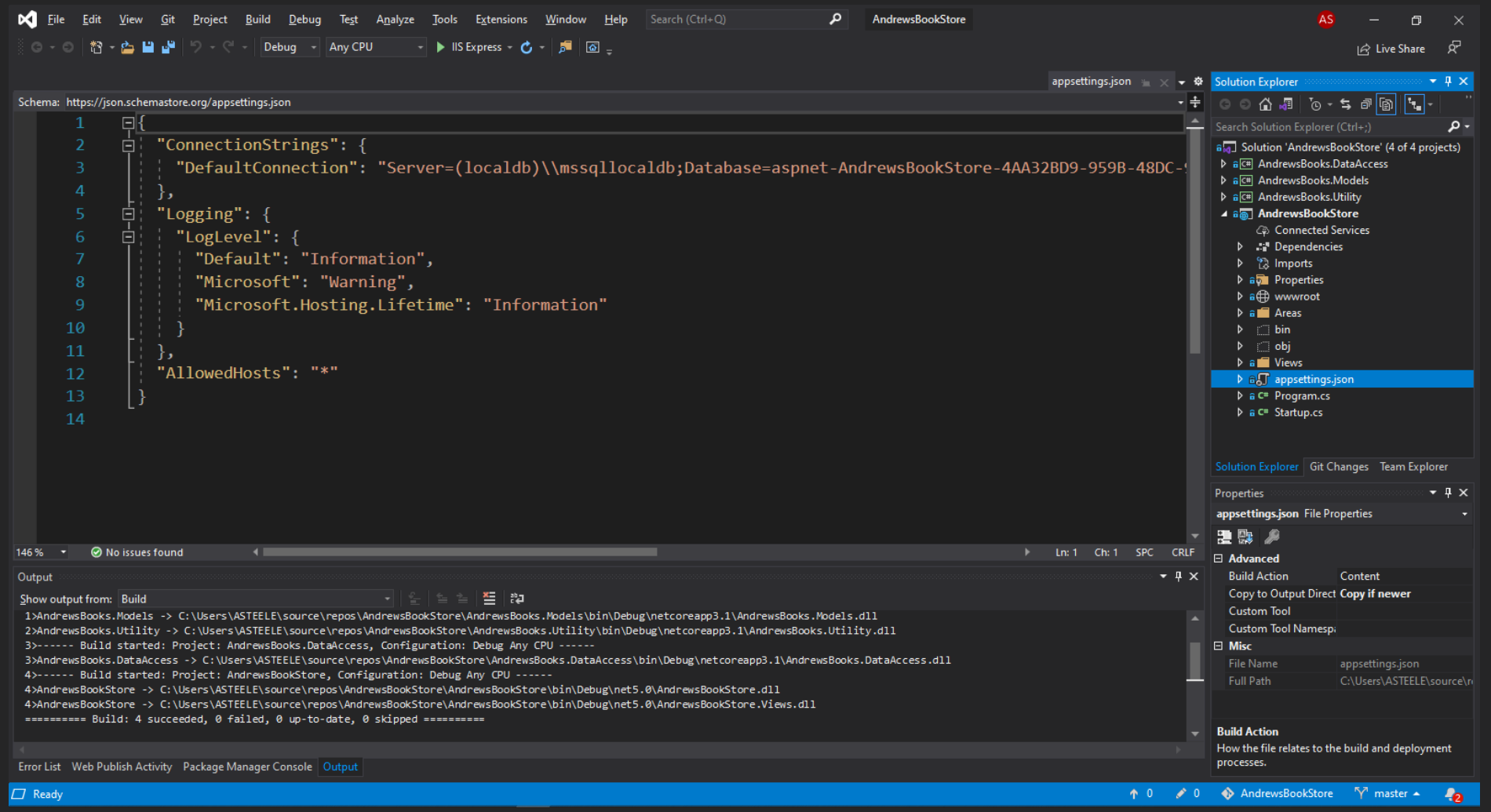
- To prepare for the next part, ‘build’ the application and confirm there are no errors



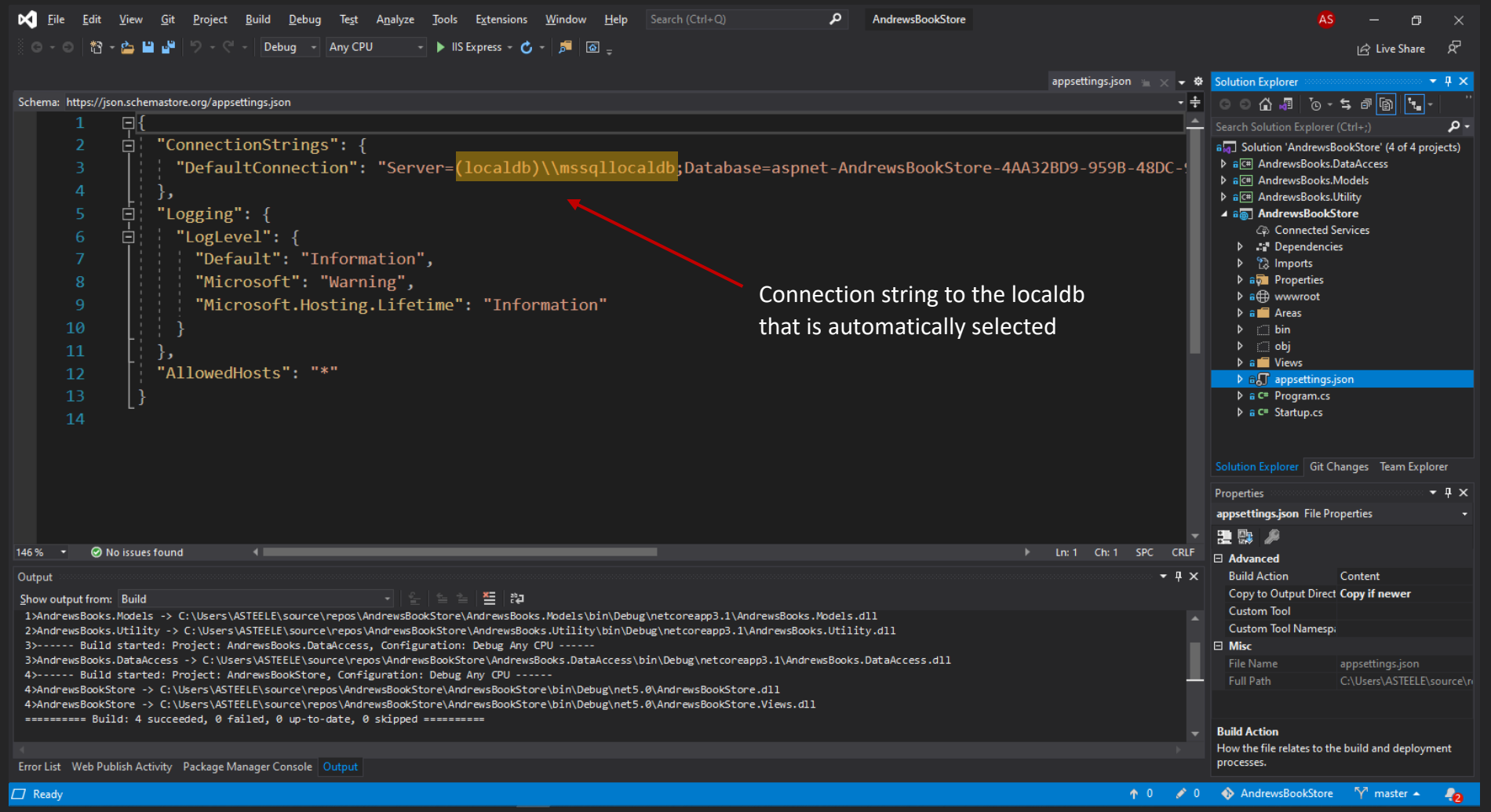
- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

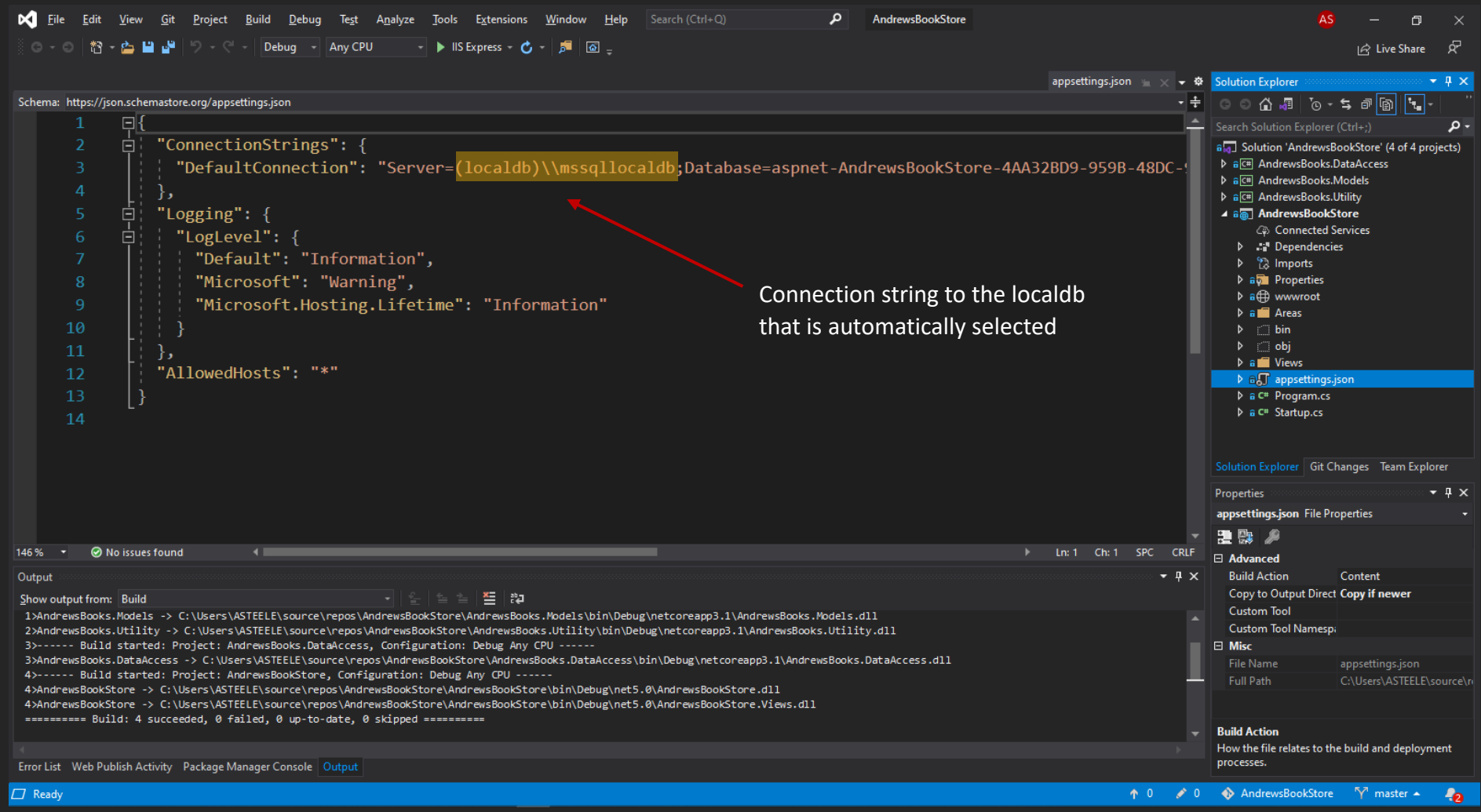


- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json



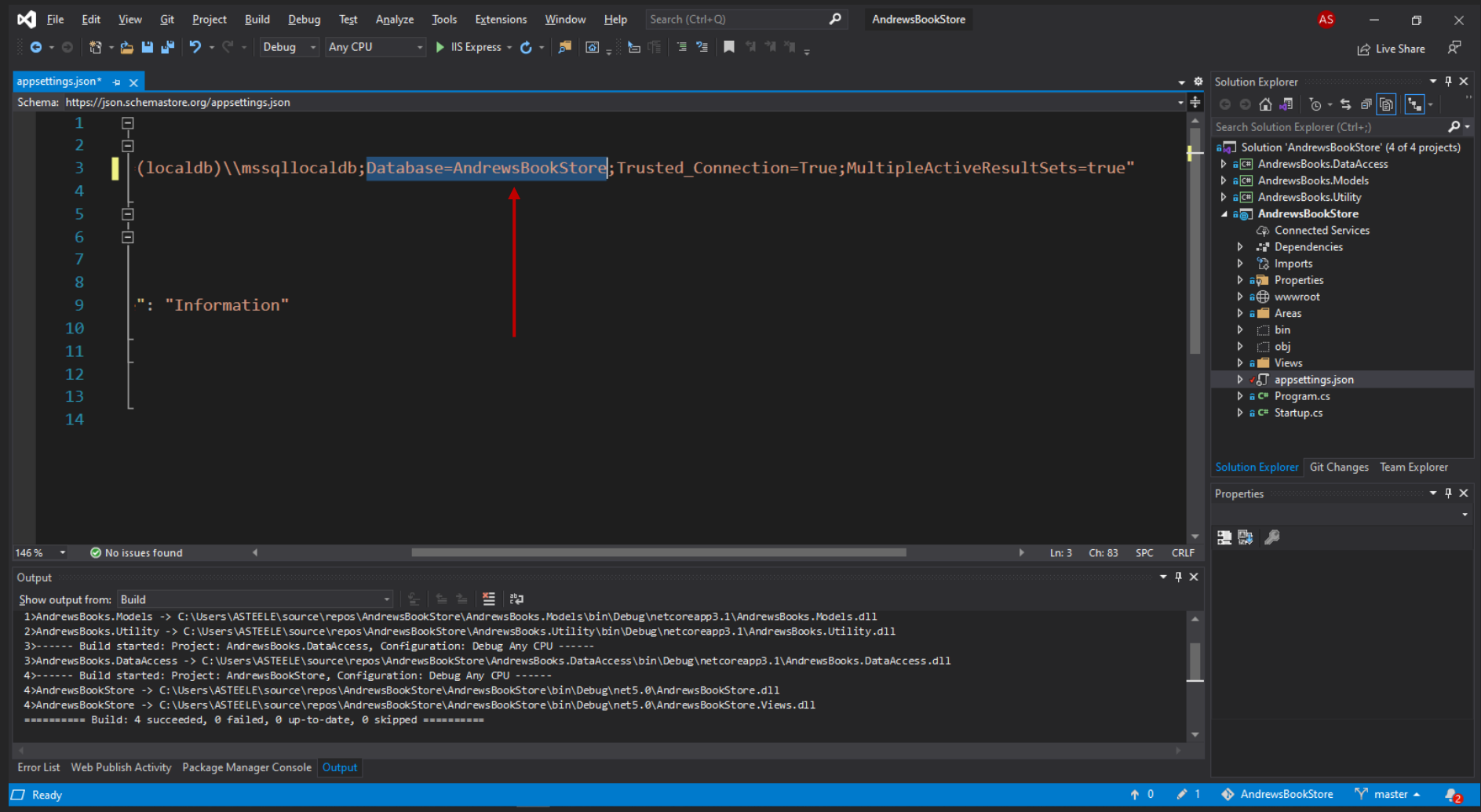
- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

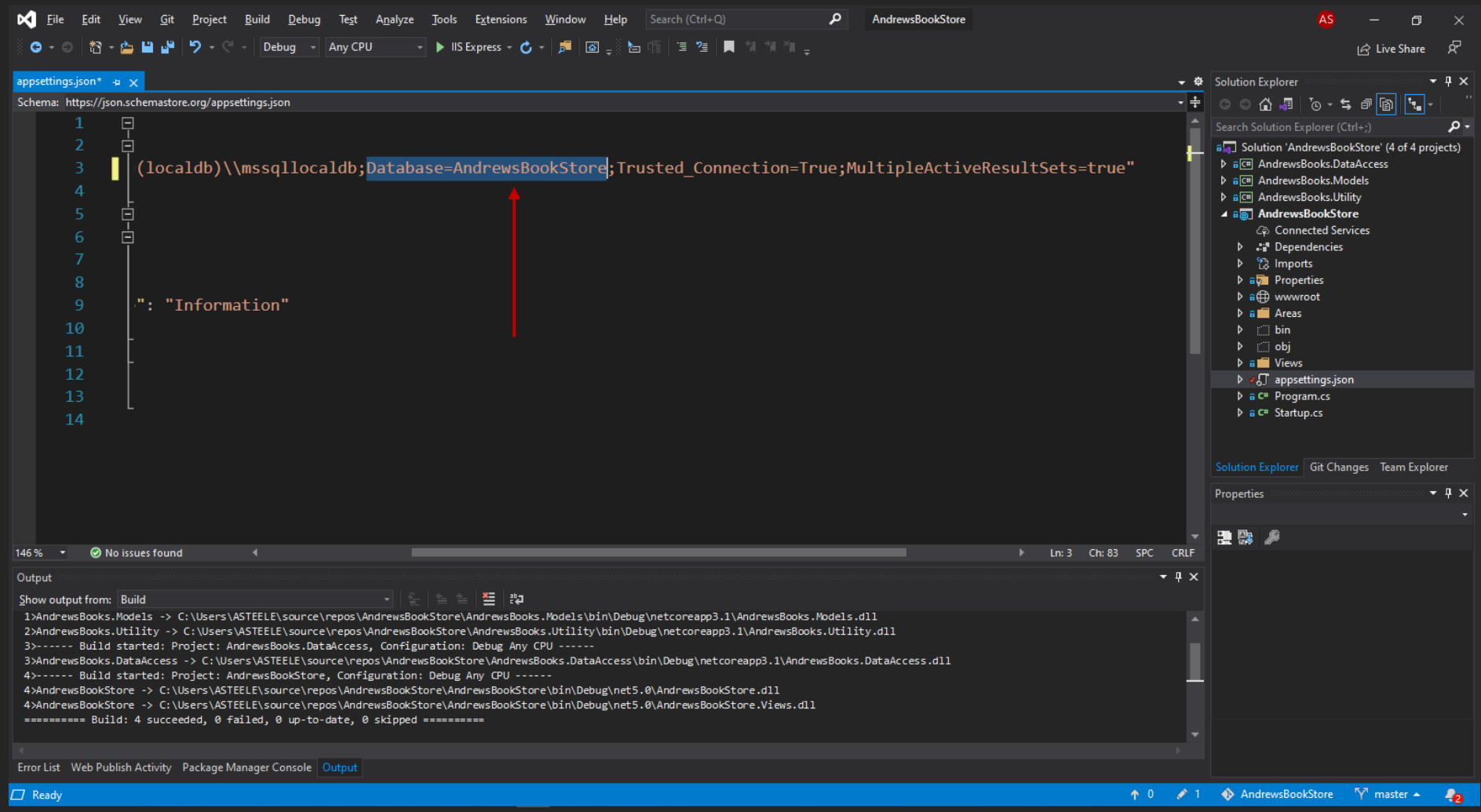
- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

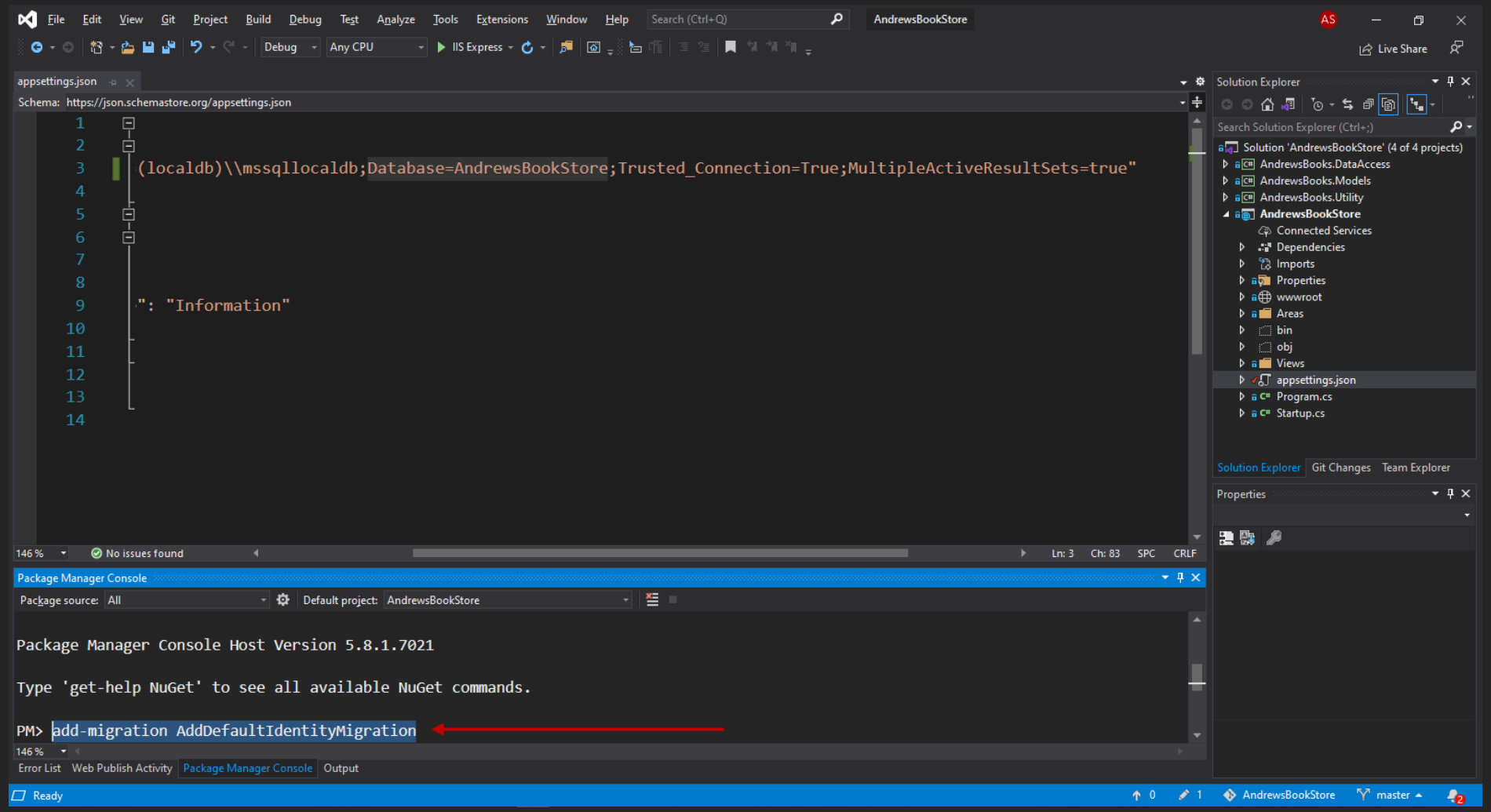
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

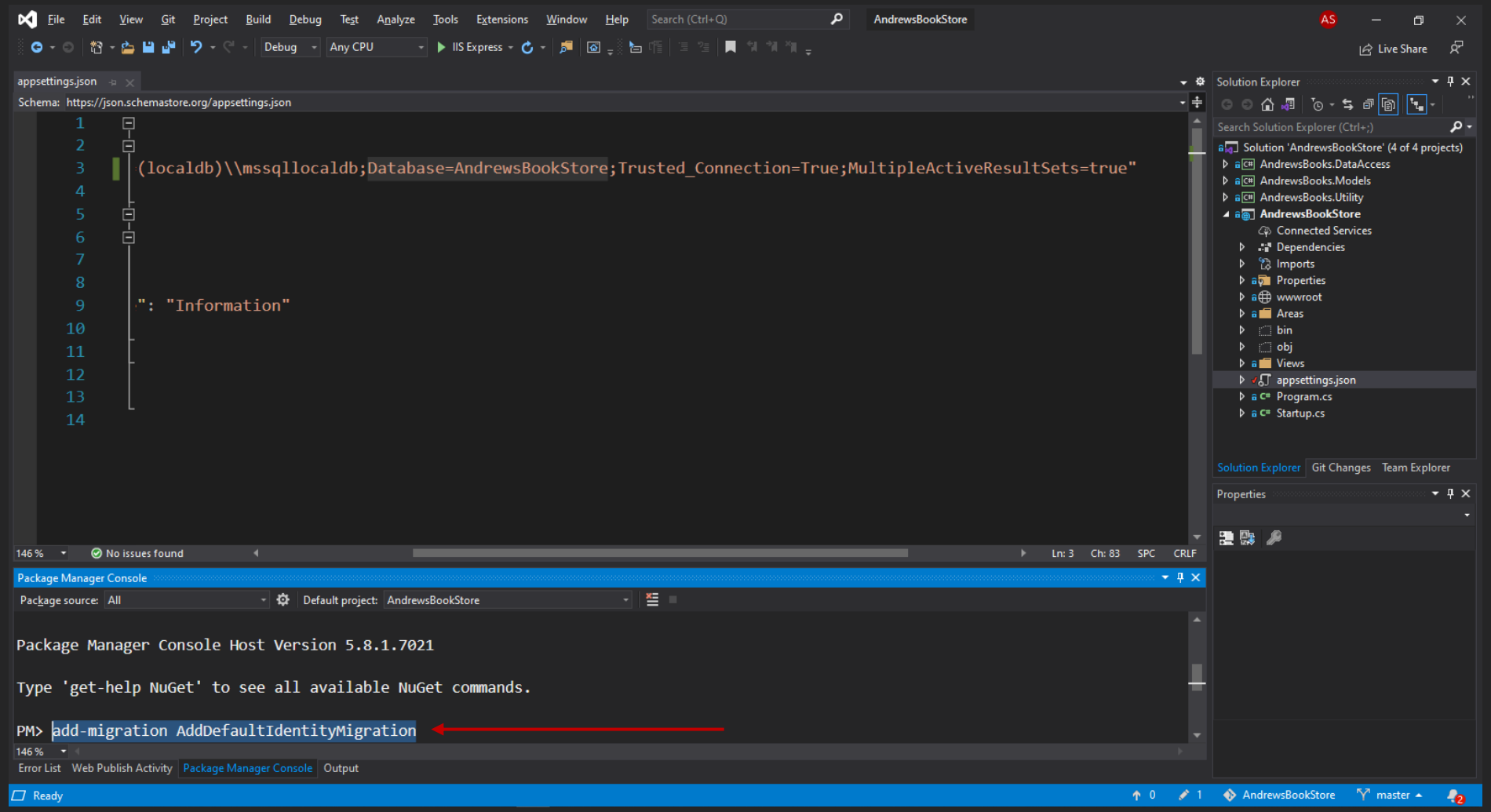
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

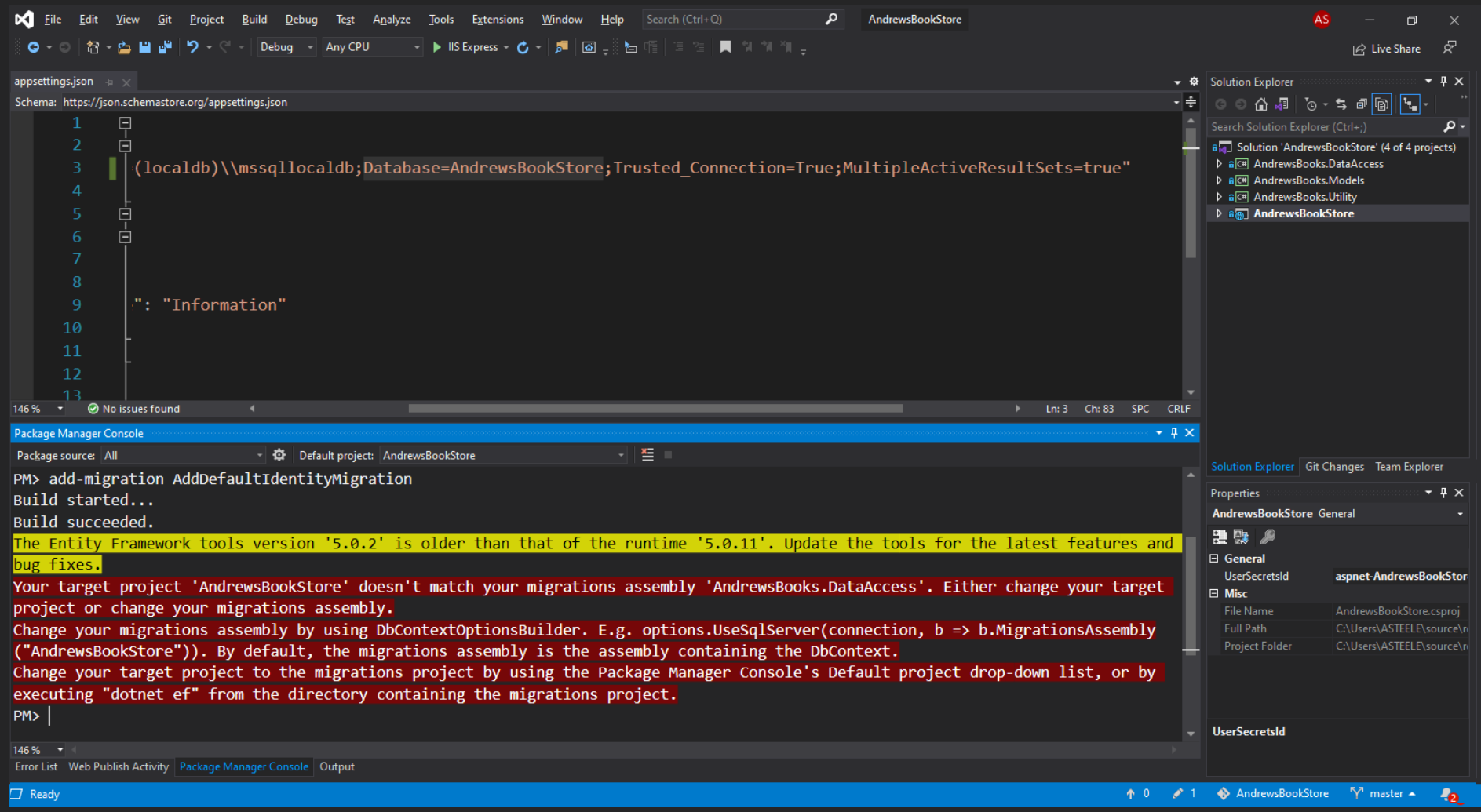
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

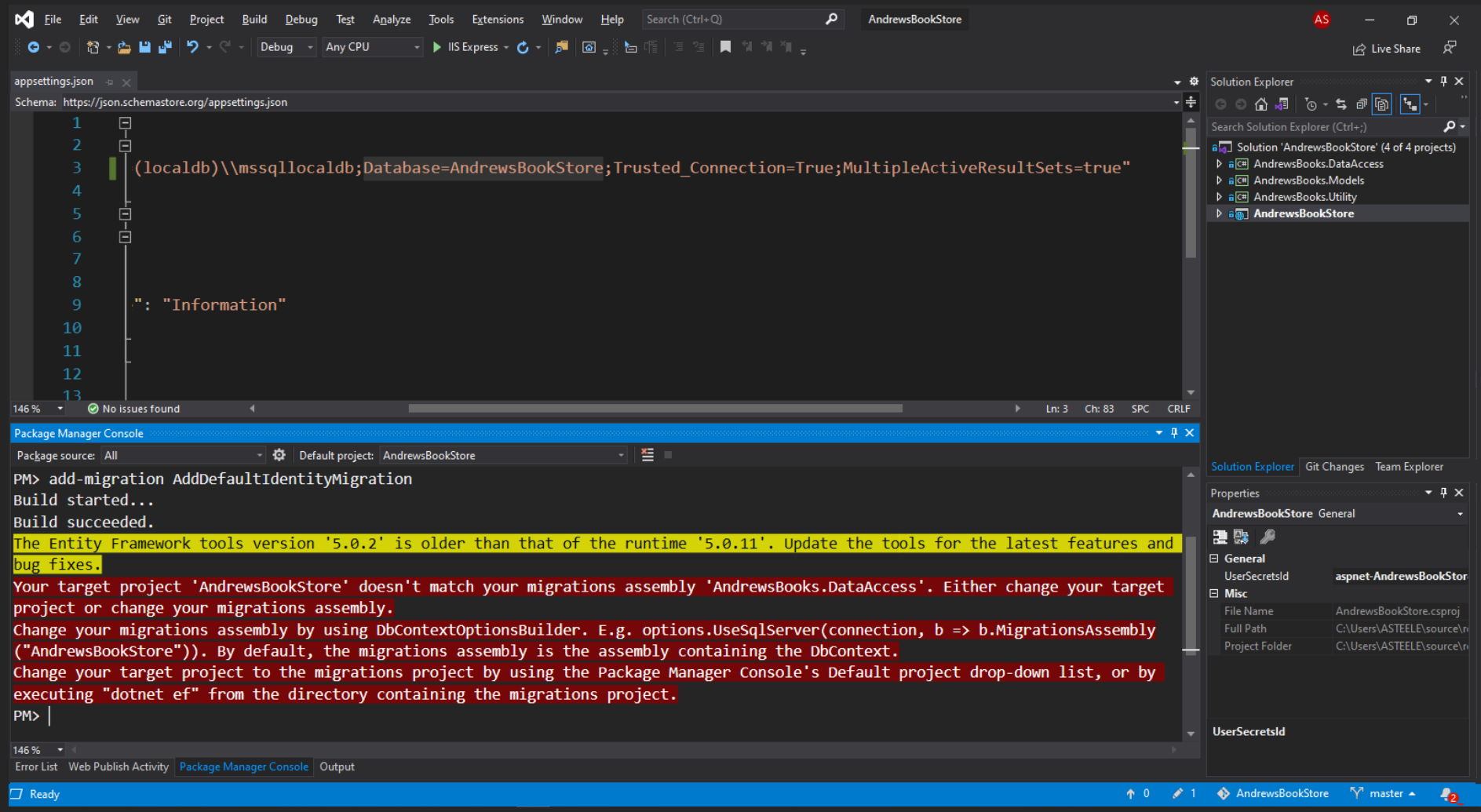
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

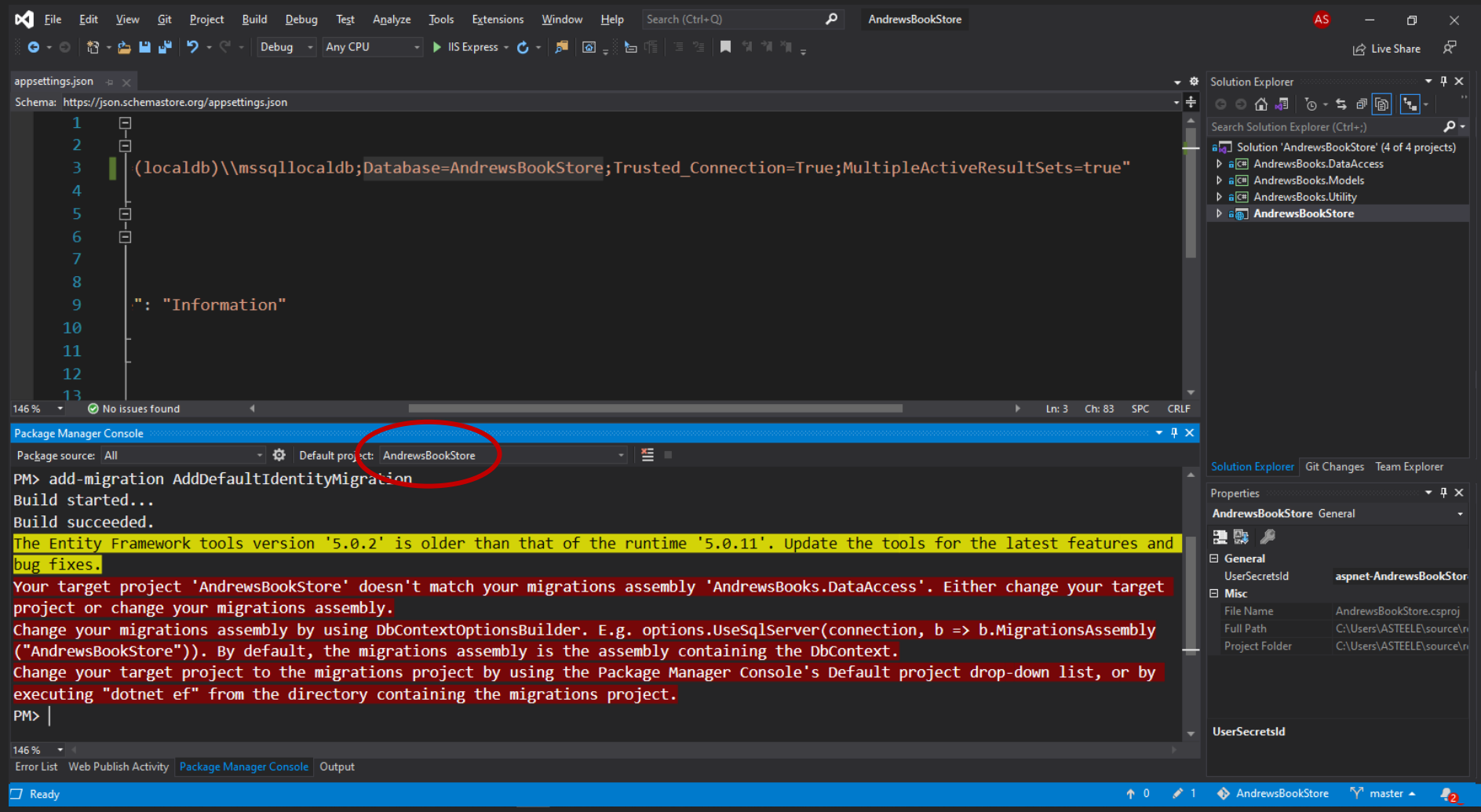
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.
- Change to the correct default project (.DataAccess) and run again.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

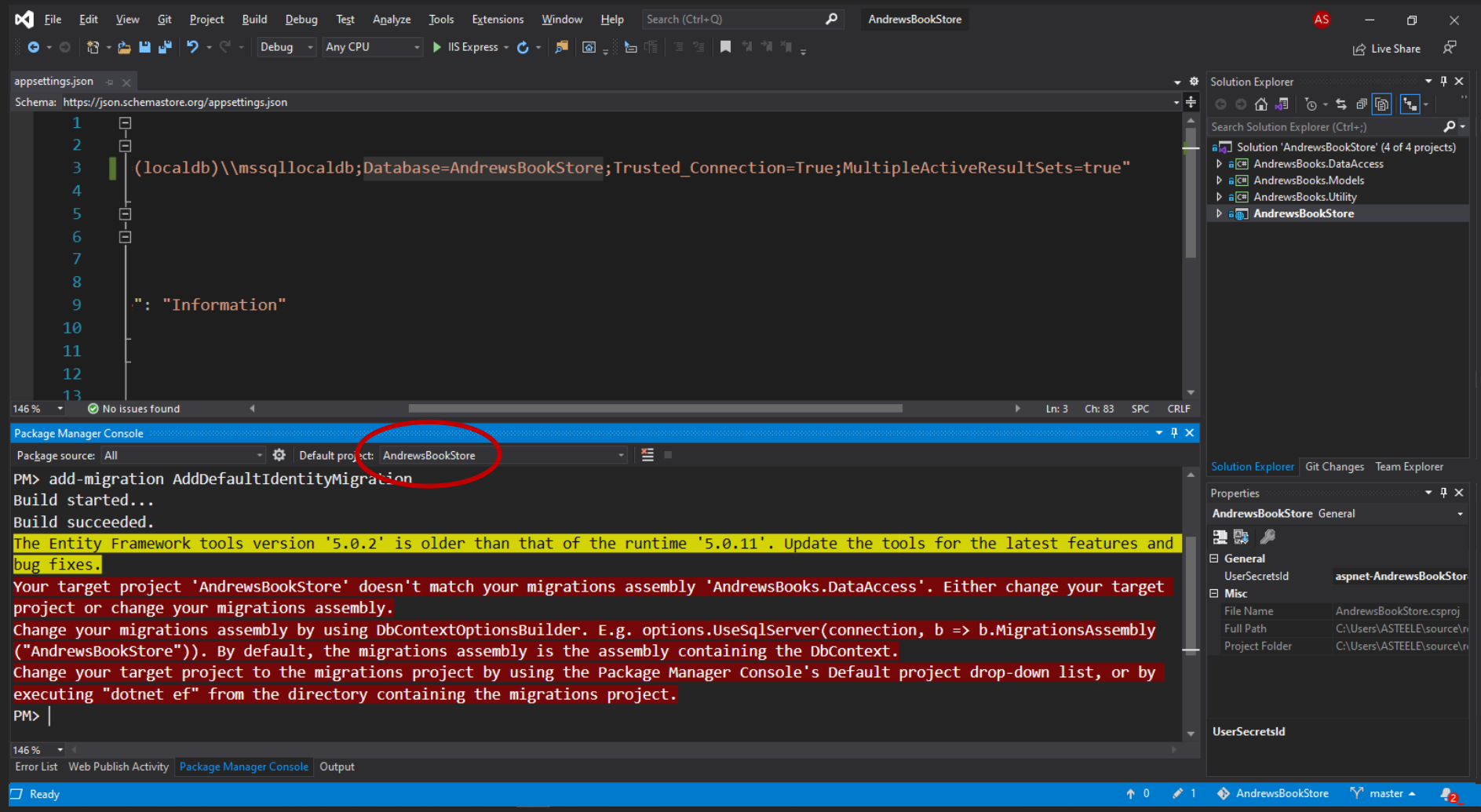
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.
- Change to the correct default project (.DataAccess) and run again.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

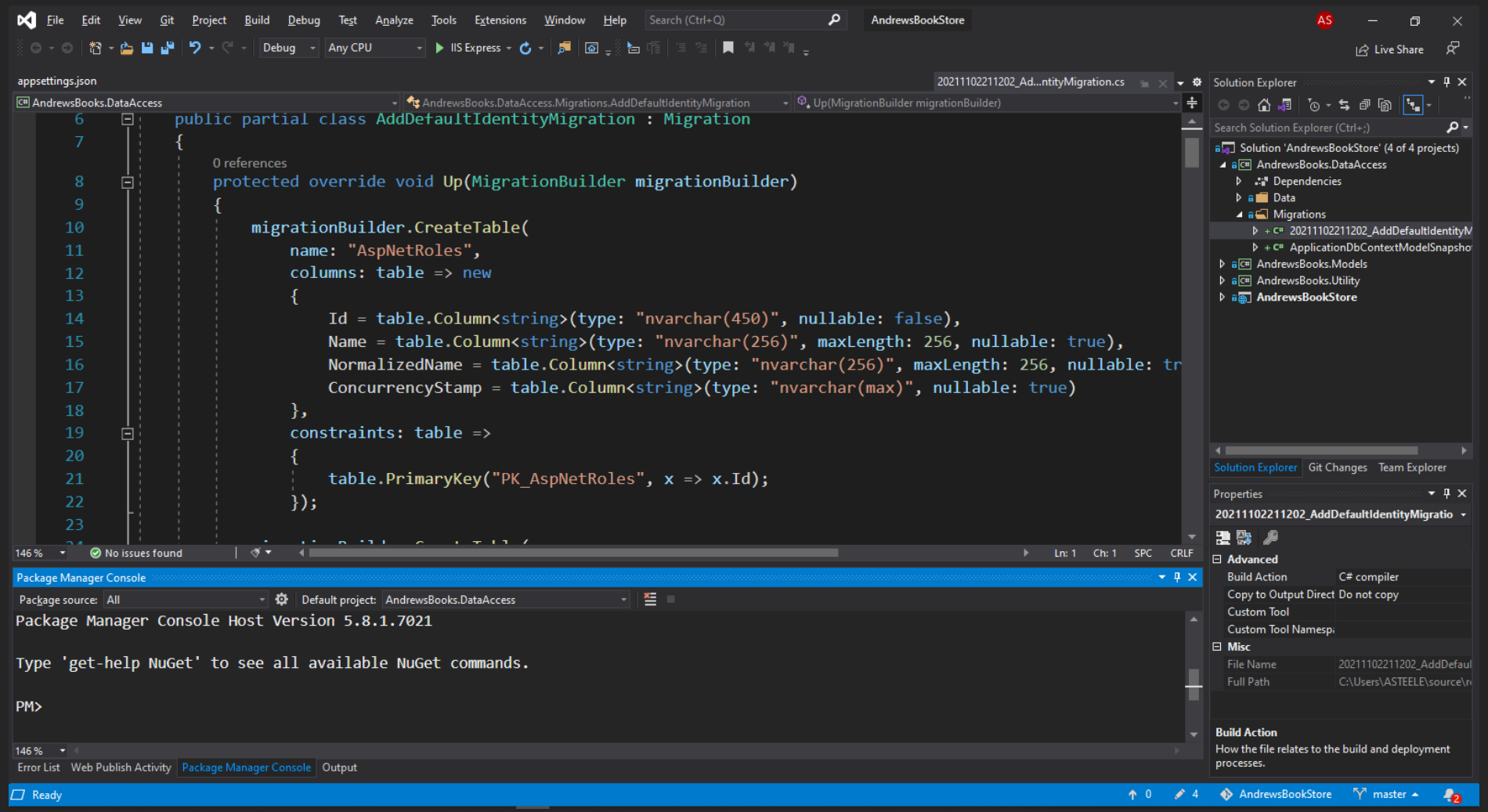
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.
- Change to the correct default project (.DataAccess) and run again.
- Add the new migration file name entry in the README.



- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

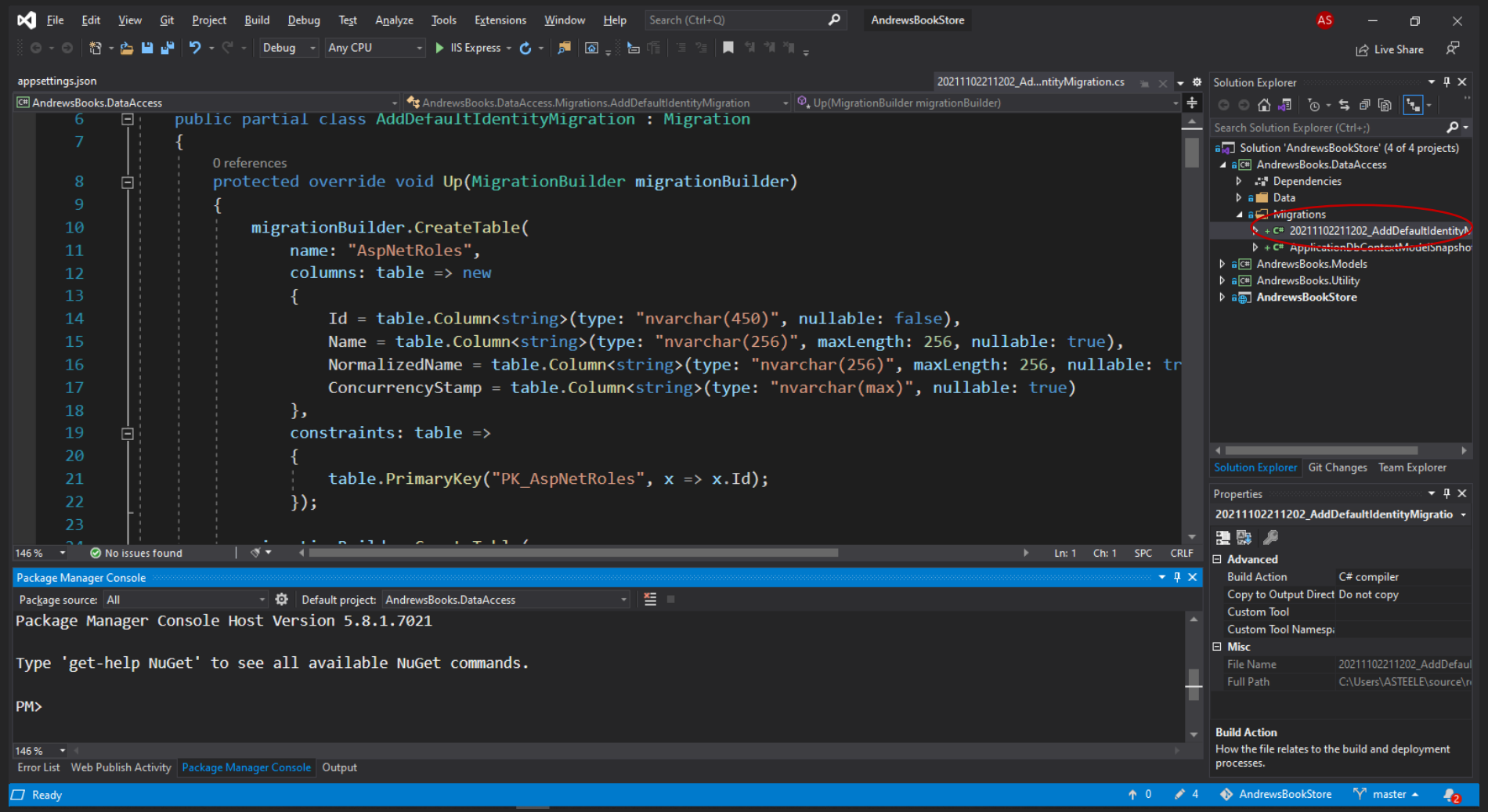
- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.
- Change to the correct default project (.DataAccess) and run again.
- Add the new migration file name entry in the README.

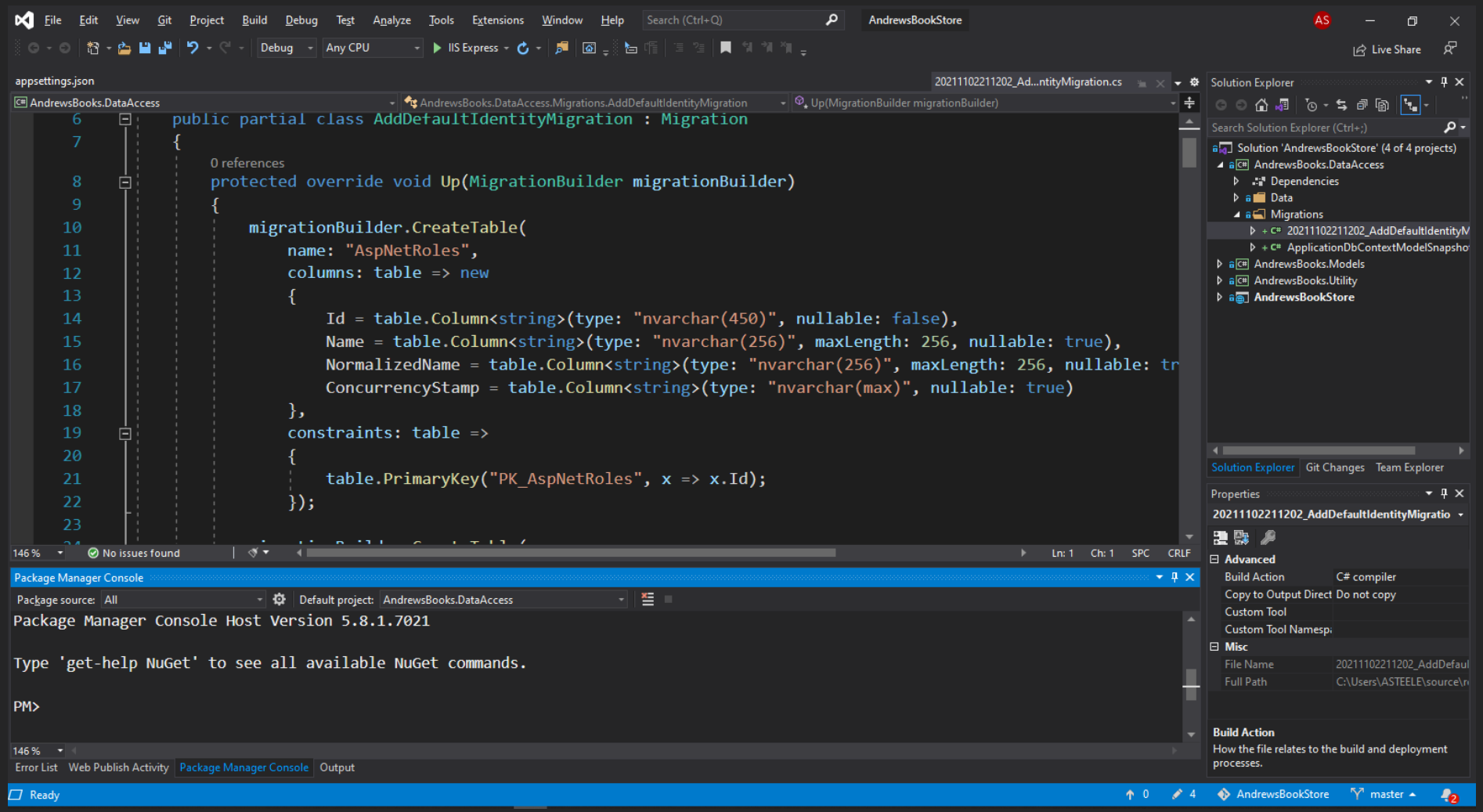


- To prepare for the next part, ‘build’ the application and confirm there are no errors
- Review appsettings.json

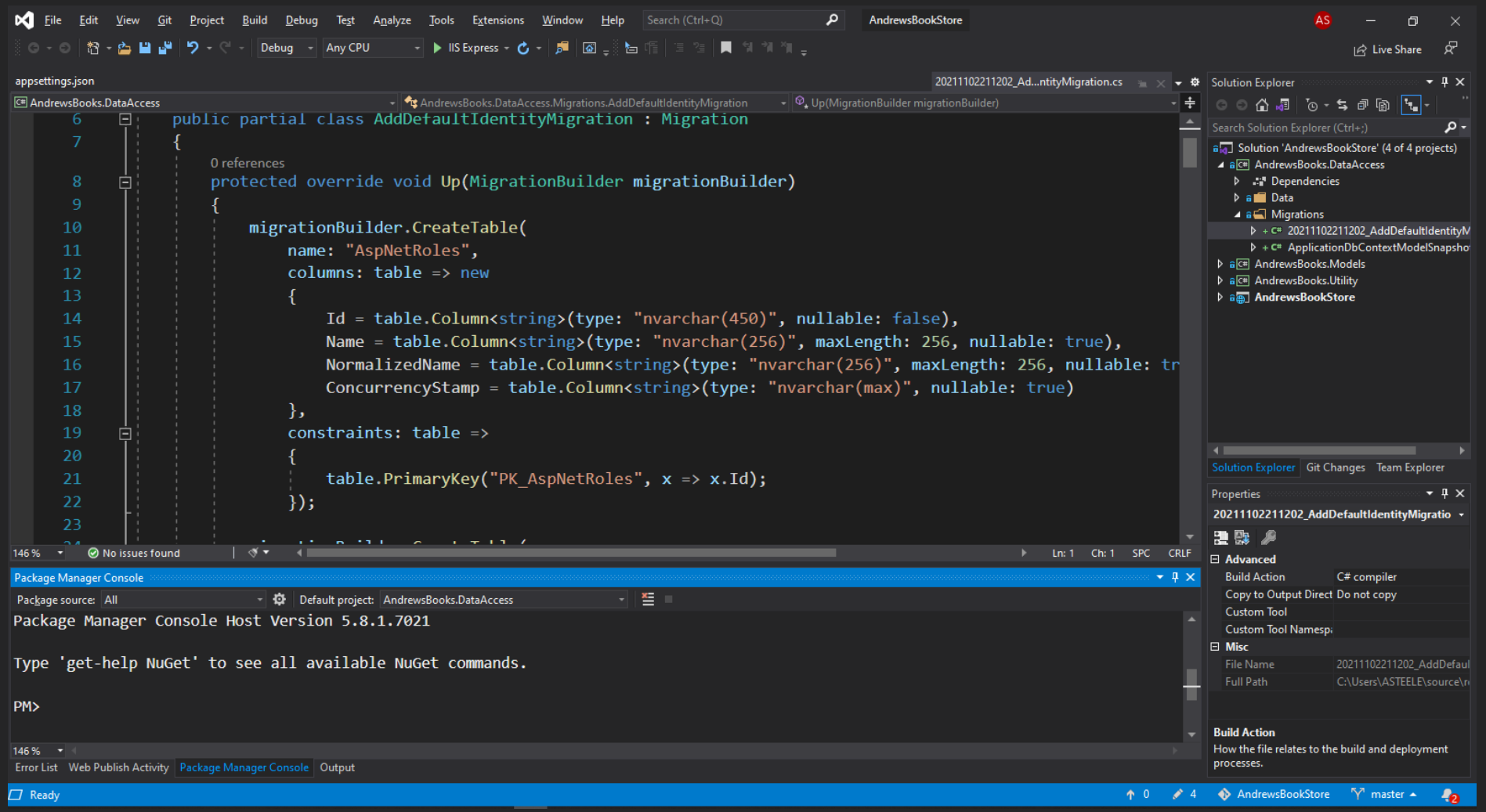
- Create the migrations (using code-first, where changes are “pushed” to the database), modify the database name and save.

- Use the NuGet Package Manager Console to add the migration (with a meaningful name).
- Note what happens if the wrong default project is selected.
- Change to the correct default project (.DataAccess) and run again.
- Add the new migration file name entry in the README.

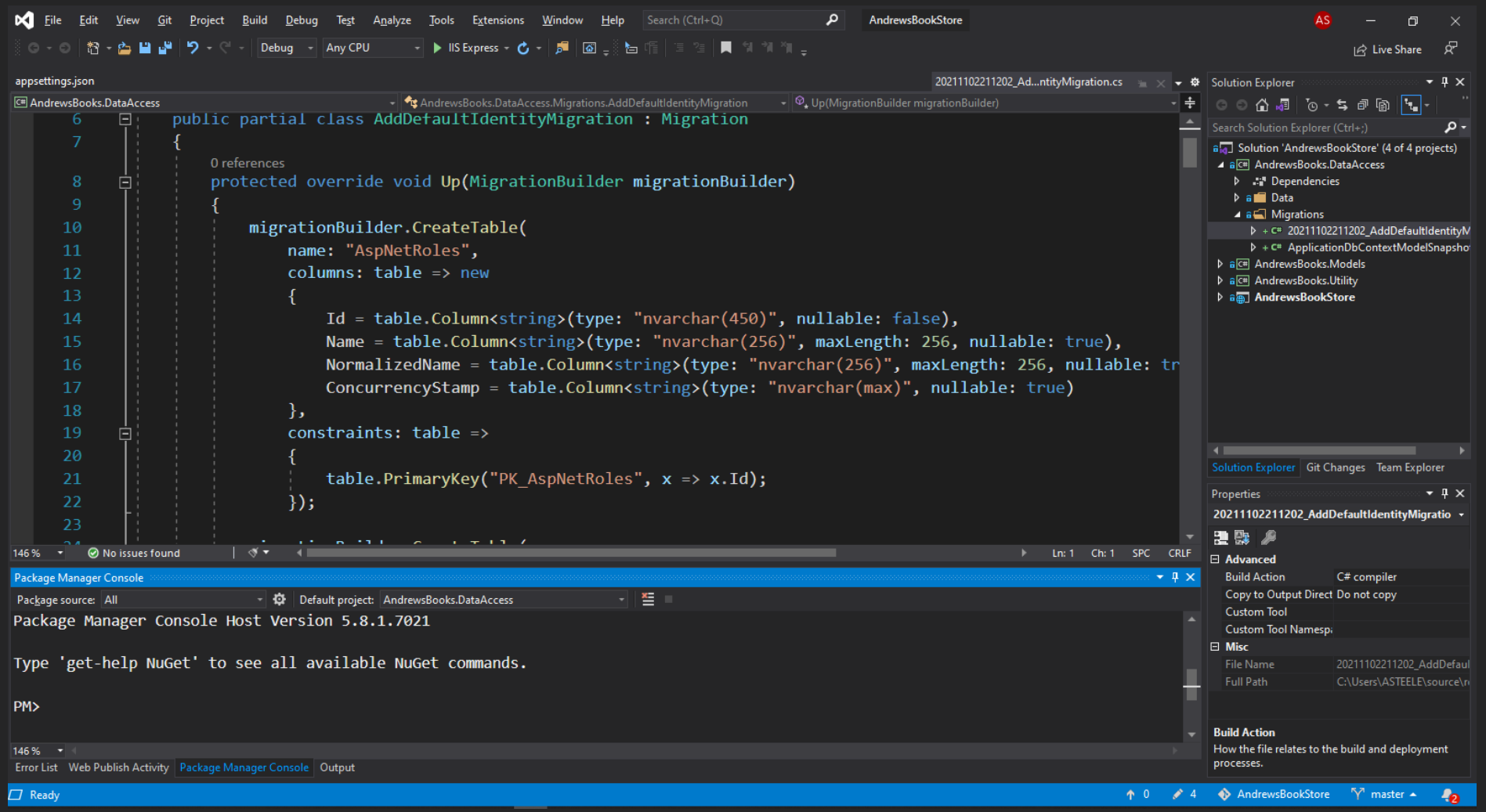



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
AndrewsBookStore
Debug
Any CPU
IIS Express
Live Share
appsettings.json
20211102211202_Ad...ntityMigration.cs
AndrewsBooks.DataAccess
AndrewsBooks.DataAccess.Migrations.AddDefaultIdentityMigration
Up(MigrationBuilder migrationBuilder)
public partial class AddDefaultIdentityMigration : Migration
{
0 references
protected override void Up(MigrationBuilder migrationBuilder)
{
migrationBuilder.CreateTable(
name: "AspNetRoles",
columns: table => new
{
Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
Name = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
NormalizedName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: tr
ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
},
constraints: table =>
{
table.PrimaryKey("PK_AspNetRoles", x => x.Id);
});
146 %
No issues found
Ln: 1 Ch: 1 SPC CRLF
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AndrewsBookStore' (4 of 4 projects)
AndrewsBooks.DataAccess
Dependencies
Data
Migrations
20211102211202_AddDefaultIdentityM
ApplicationDbContextModelSnapsho
AndrewsBooks.Models
AndrewsBooks.Utility
AndrewsBookStore
Solution Explorer
Git Changes
Team Explorer
Properties
20211102211202_AddDefaultIdentityMigratio
Advanced
Build Action
C# compiler
Copy to Output Direct
Do not copy
Custom Tool
Custom Tool Namespa
Misc
File Name
20211102211202_AddDefaul
Full Path
C:\Users\ASTEELE\source\r
Build Action
How the file relates to the build and deployment processes.
Package Manager Console
Package source: All
Default project: AndrewsBooks.DataAccess
Package Manager Console Host Version 5.8.1.7021
Type 'get-help NuGet' to see all available NuGet commands.
PM>
146 %
Error List
Web Publish Activity
Package Manager Console
Output
Ready
0
4
AndrewsBookStore
master

- Review the file for the SQL-like syntax, statements, columns and primary key constraints.

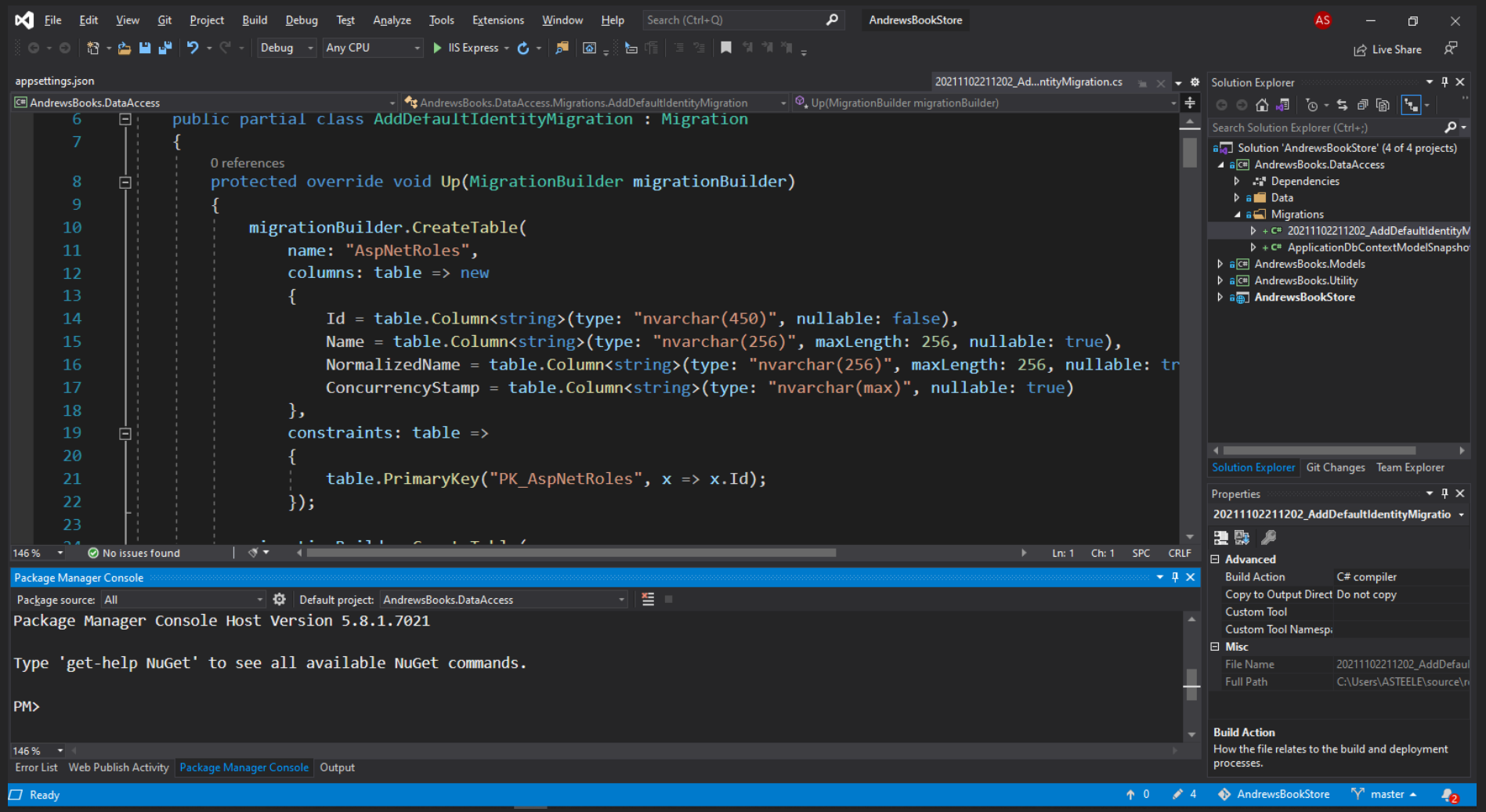


- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.



- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

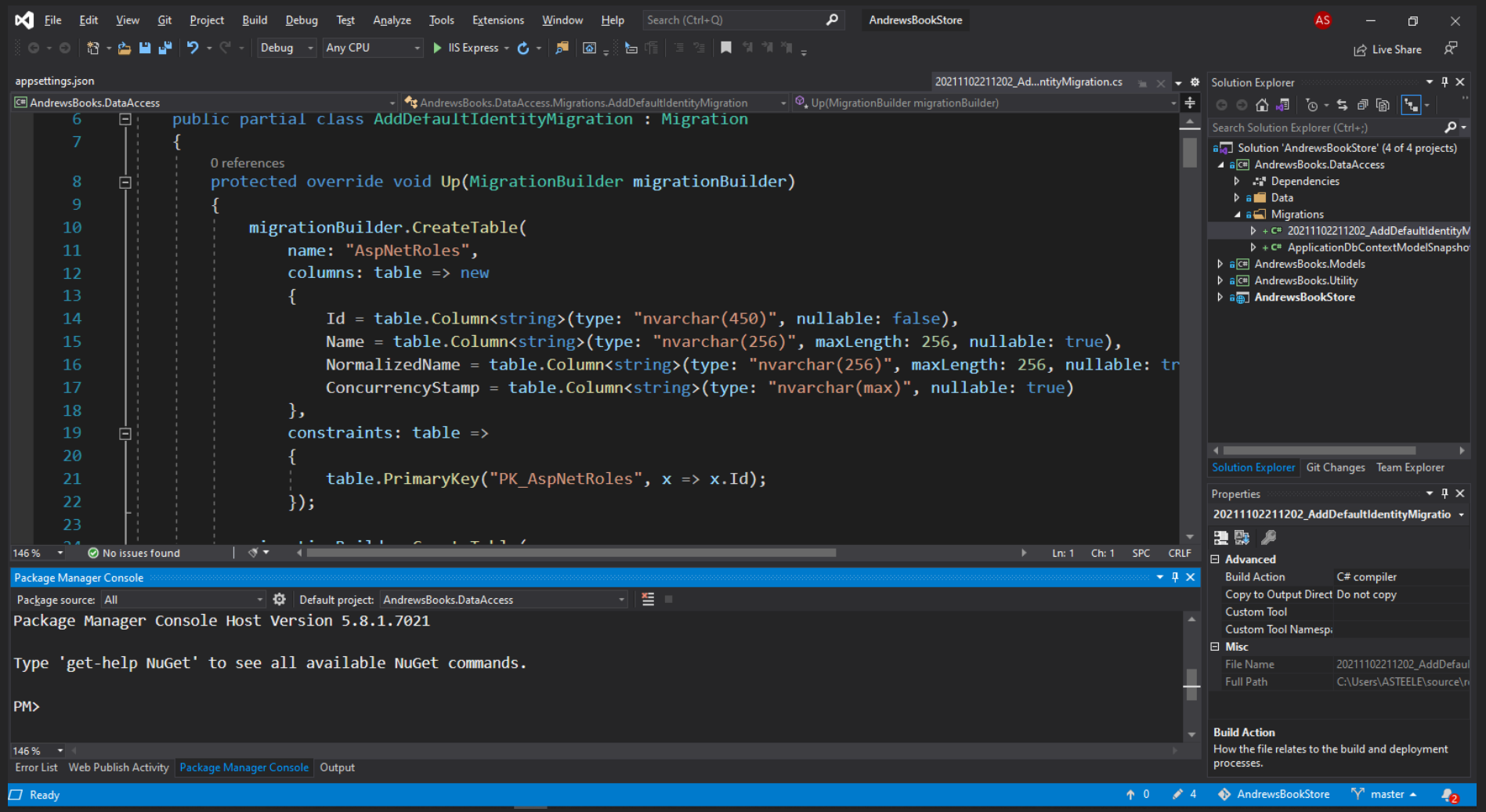
- Once the migration is complete, the database needs to be updated.



- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

- Once the migration is complete, the database needs to be updated.

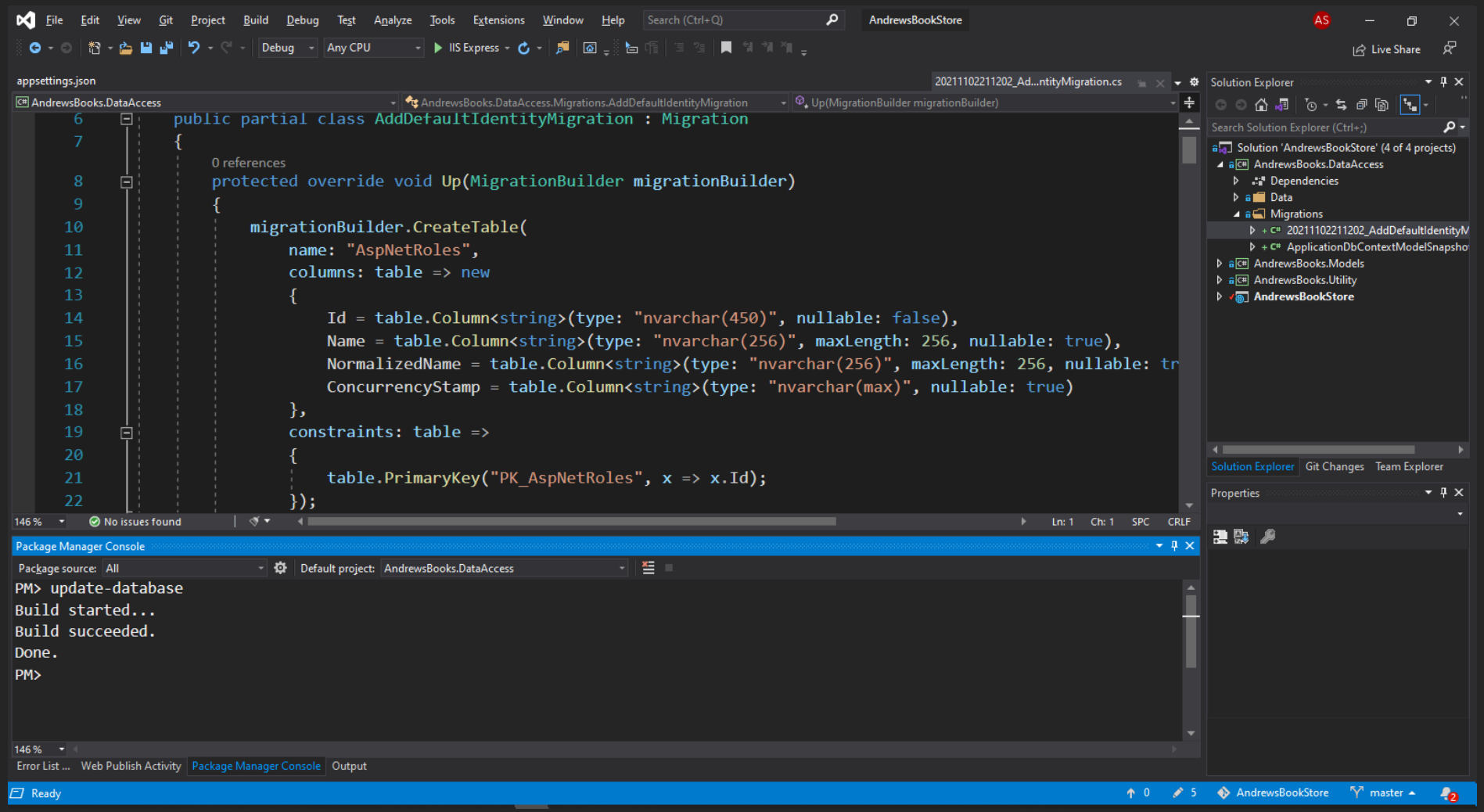
- In the PM console, update the database now.



- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

- Once the migration is complete, the database needs to be updated.

- In the PM console, update the database now.

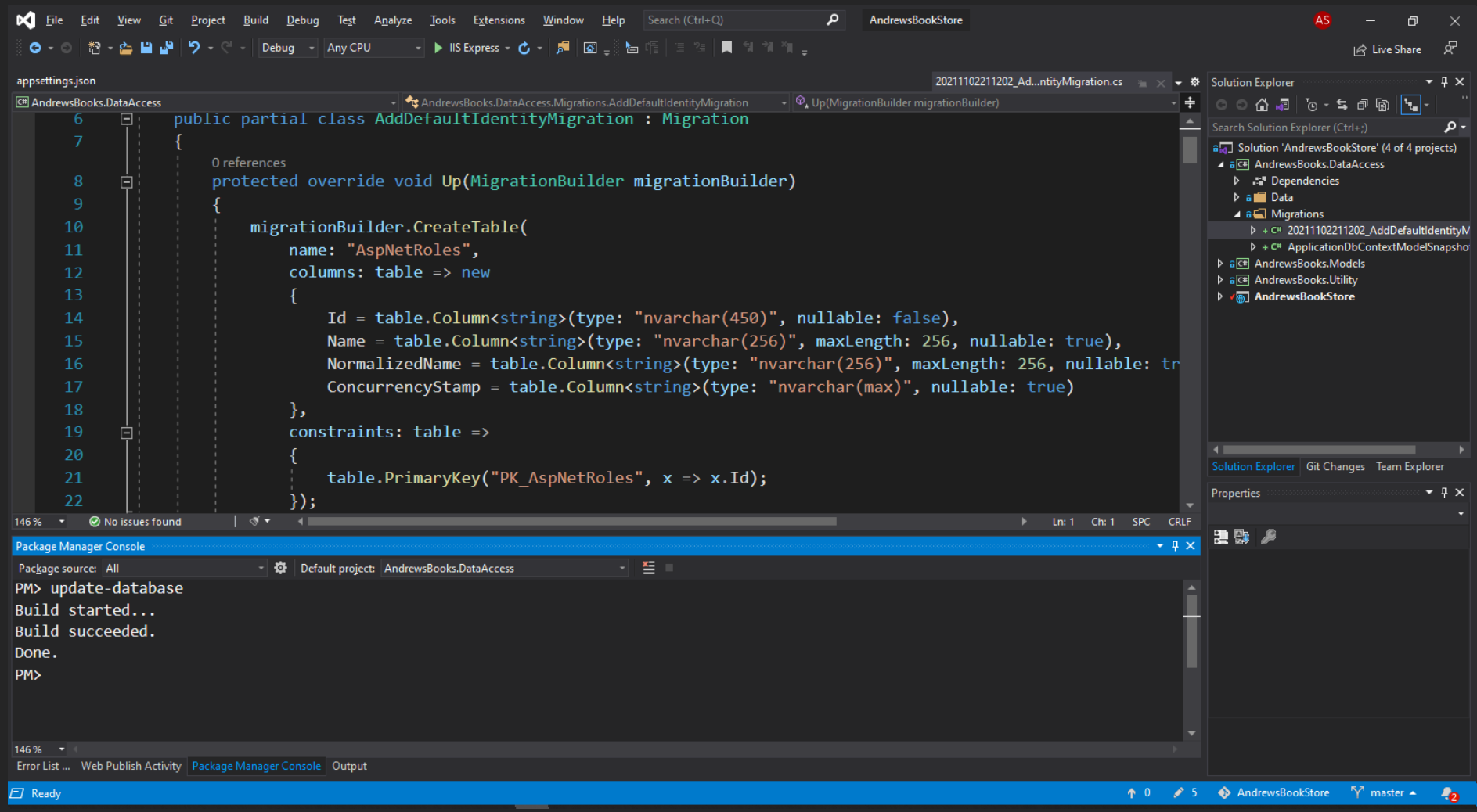


- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

- Once the migration is complete, the database needs to be updated.

- In the PM console, update the database now.

- Review the updated database in the SQL Server Object Explorer.

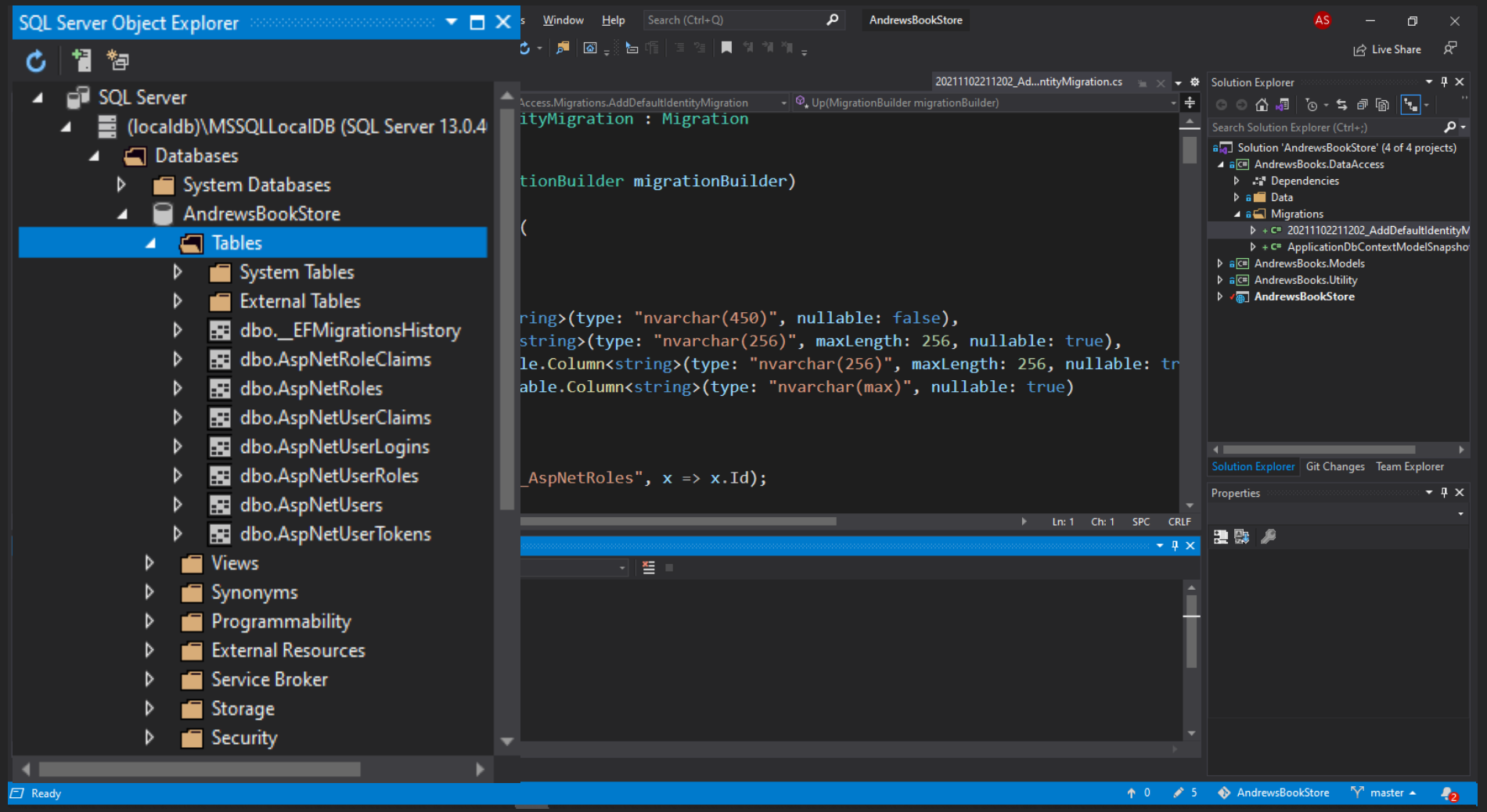


- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

- Once the migration is complete, the database needs to be updated.

- In the PM console, update the database now.

- Review the updated database in the SQL Server Object Explorer.

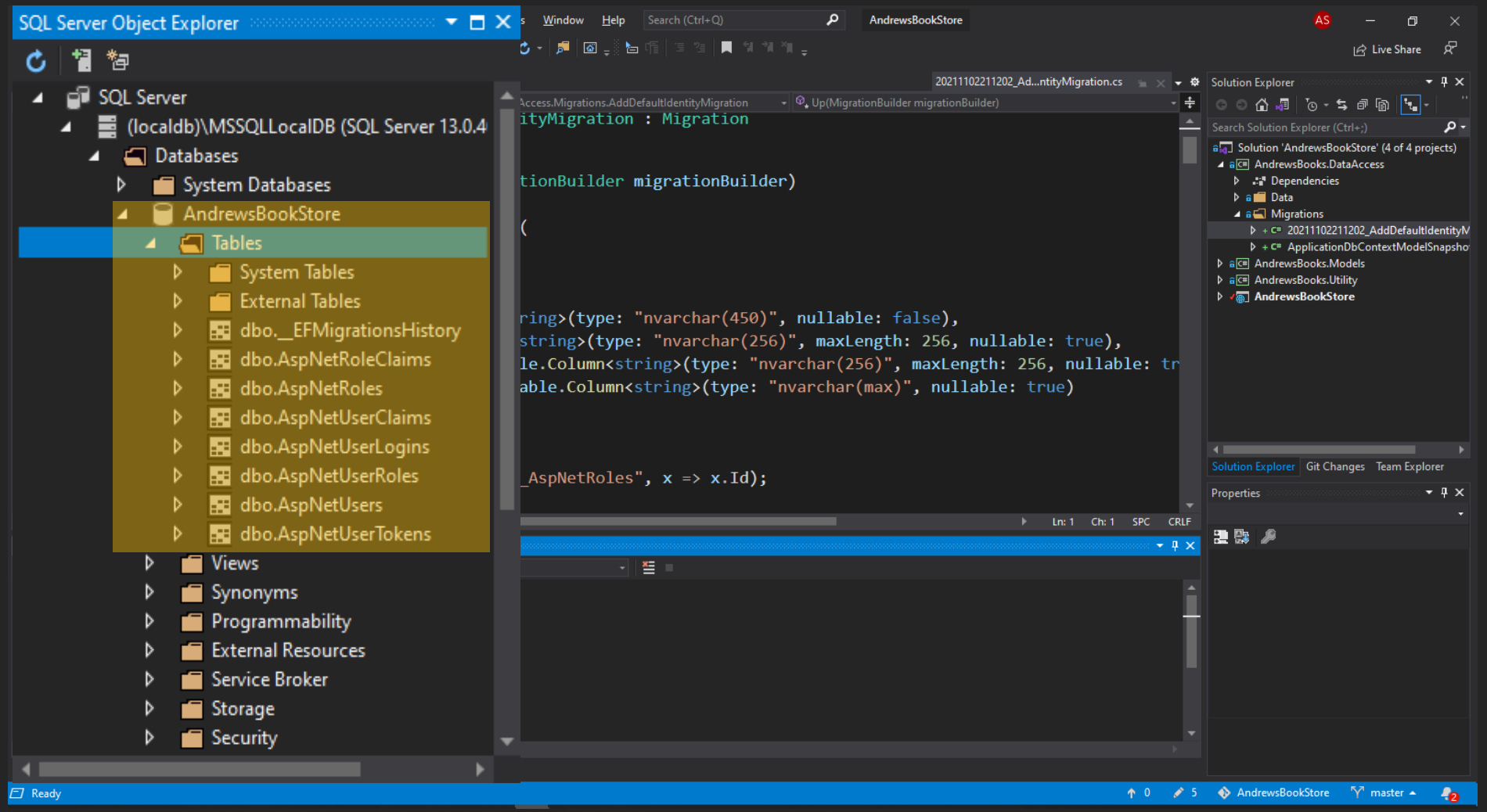


- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

- Once the migration is complete, the database needs to be updated.

- In the PM console, update the database now.

- Review the updated database in the SQL Server Object Explorer.

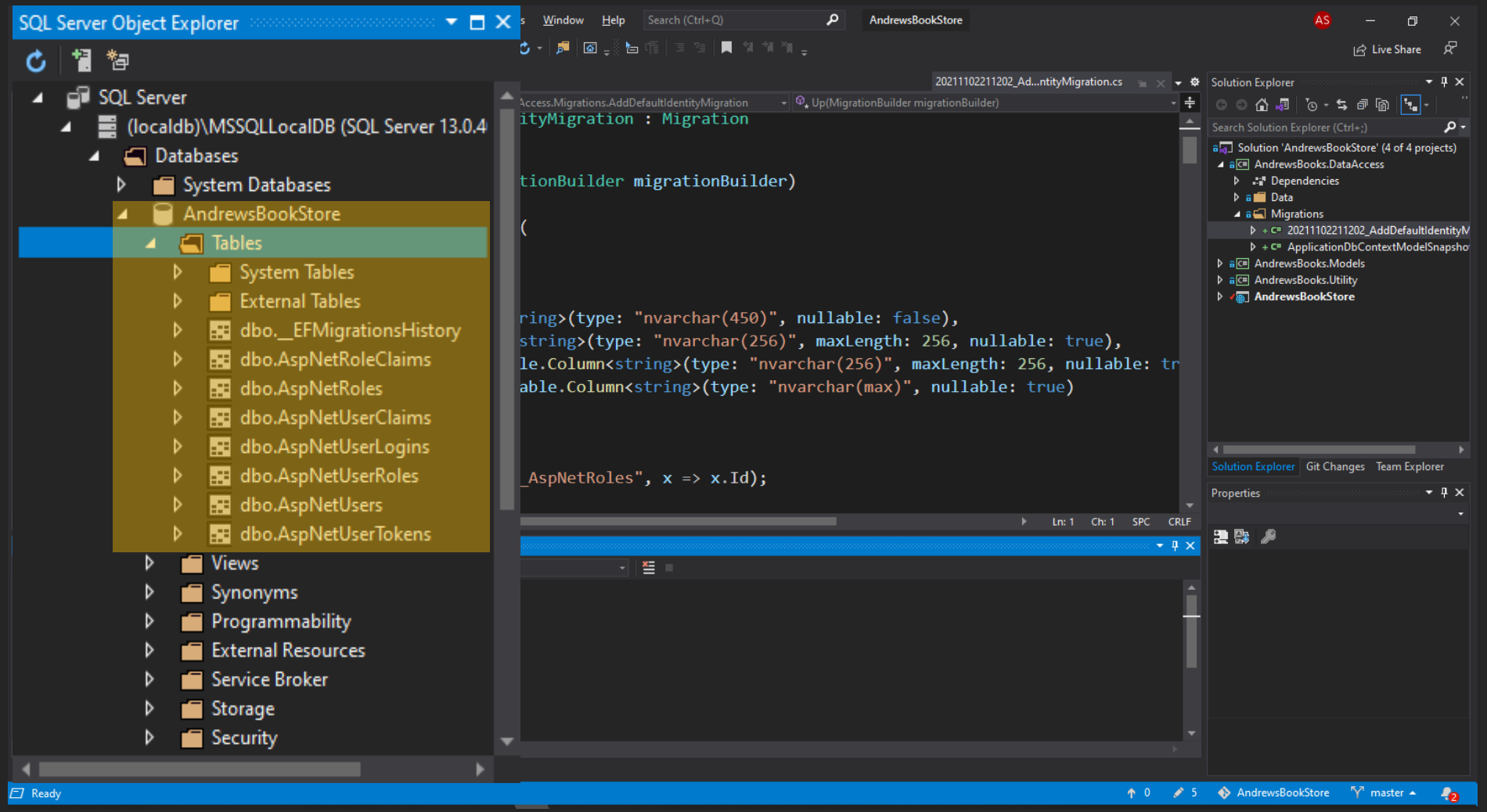


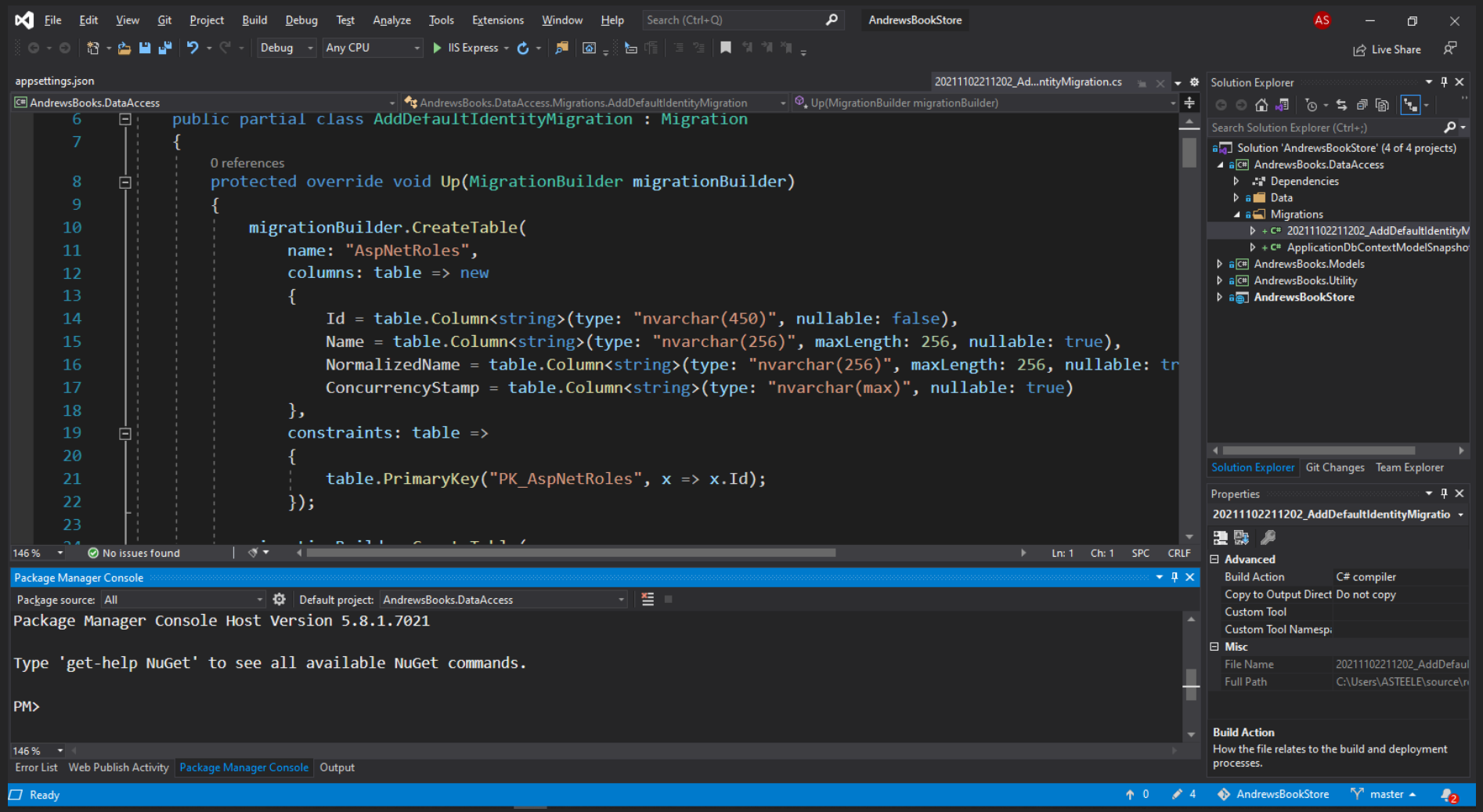
- Review the file for the SQL-like syntax, statements, columns and primary key constraints.
- Note tables related the ASP.NET Identity.

- Once the migration is complete, the database needs to be updated.

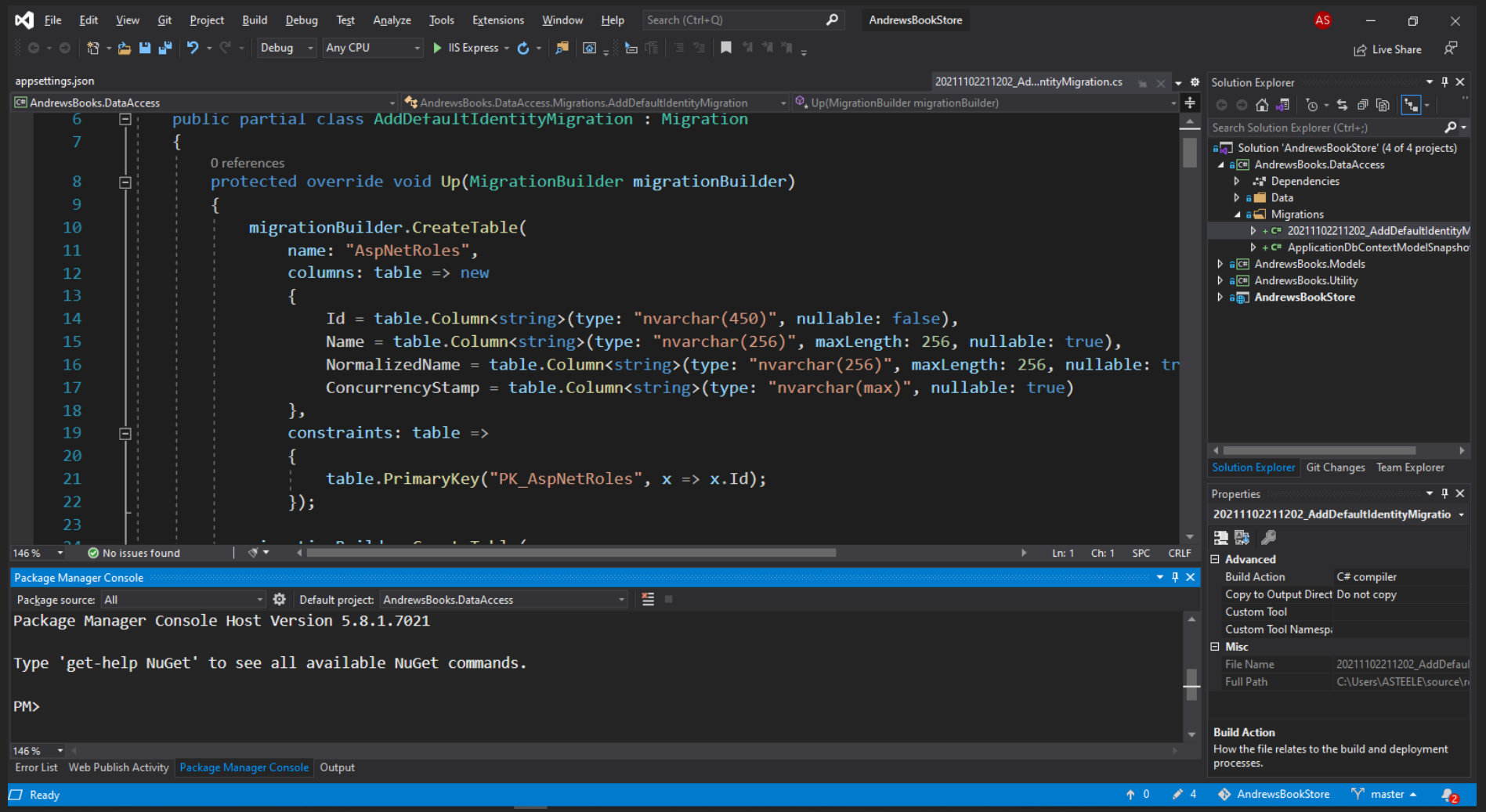
- In the PM console, update the database now.

- Review the updated database in the SQL Server Object Explorer.
- Check for errors, run the application

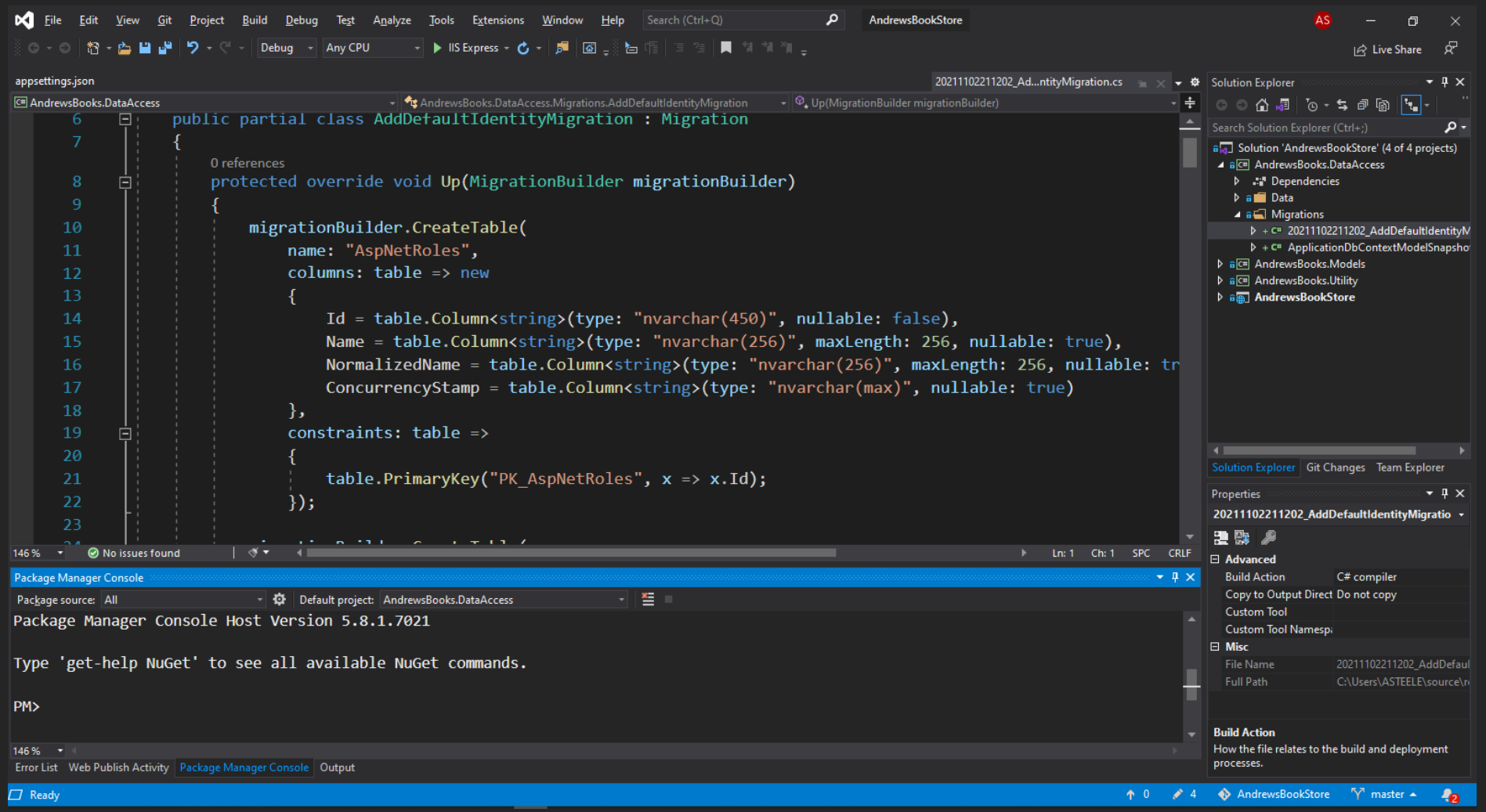



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
AndrewsBookStore
Debug
Any CPU
IIS Express
Live Share
appsettings.json
20211102211202_Ad...ntityMigration.cs
AndrewsBooks.DataAccess
AndrewsBooks.DataAccess.Migrations.AddDefaultIdentityMigration
Up(MigrationBuilder migrationBuilder)
public partial class AddDefaultIdentityMigration : Migration
{
0 references
protected override void Up(MigrationBuilder migrationBuilder)
{
migrationBuilder.CreateTable(
name: "AspNetRoles",
columns: table => new
{
Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
Name = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
NormalizedName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: tr
ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
},
constraints: table =>
{
table.PrimaryKey("PK_AspNetRoles", x => x.Id);
});
146 %
No issues found
Ln: 1 Ch: 1 SPC CRLF
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AndrewsBookStore' (4 of 4 projects)
AndrewsBooks.DataAccess
Dependencies
Data
Migrations
20211102211202_AddDefaultIdentityM
ApplicationDbContextModelSnapsho
AndrewsBooks.Models
AndrewsBooks.Utility
AndrewsBookStore
Solution Explorer
Git Changes
Team Explorer
Properties
20211102211202_AddDefaultIdentityMigratio
Advanced
Build Action
C# compiler
Copy to Output Direct
Do not copy
Custom Tool
Custom Tool Namespa
Misc
File Name
20211102211202_AddDefaul
Full Path
C:\Users\ASTEELE\source\r
Build Action
How the file relates to the build and deployment processes.
Package Manager Console
Package source: All
Default project: AndrewsBooks.DataAccess
Package Manager Console Host Version 5.8.1.7021
Type 'get-help NuGet' to see all available NuGet commands.
PM>
146 %
Error List
Web Publish Activity
Package Manager Console
Output
Ready
0
4
AndrewsBookStore
master

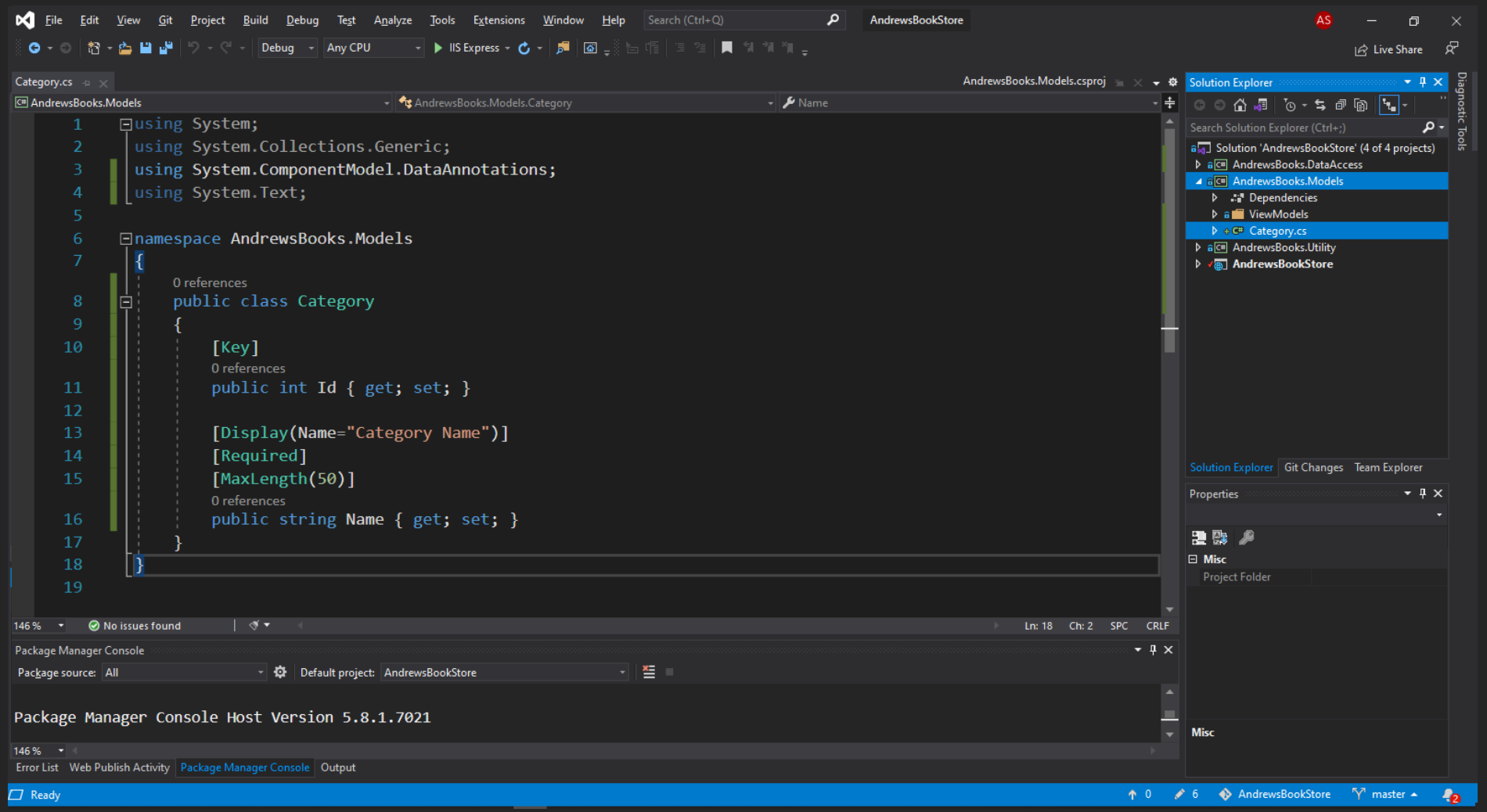
- Add a new table to the DB by creating a Category model and push it to the DB:



- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

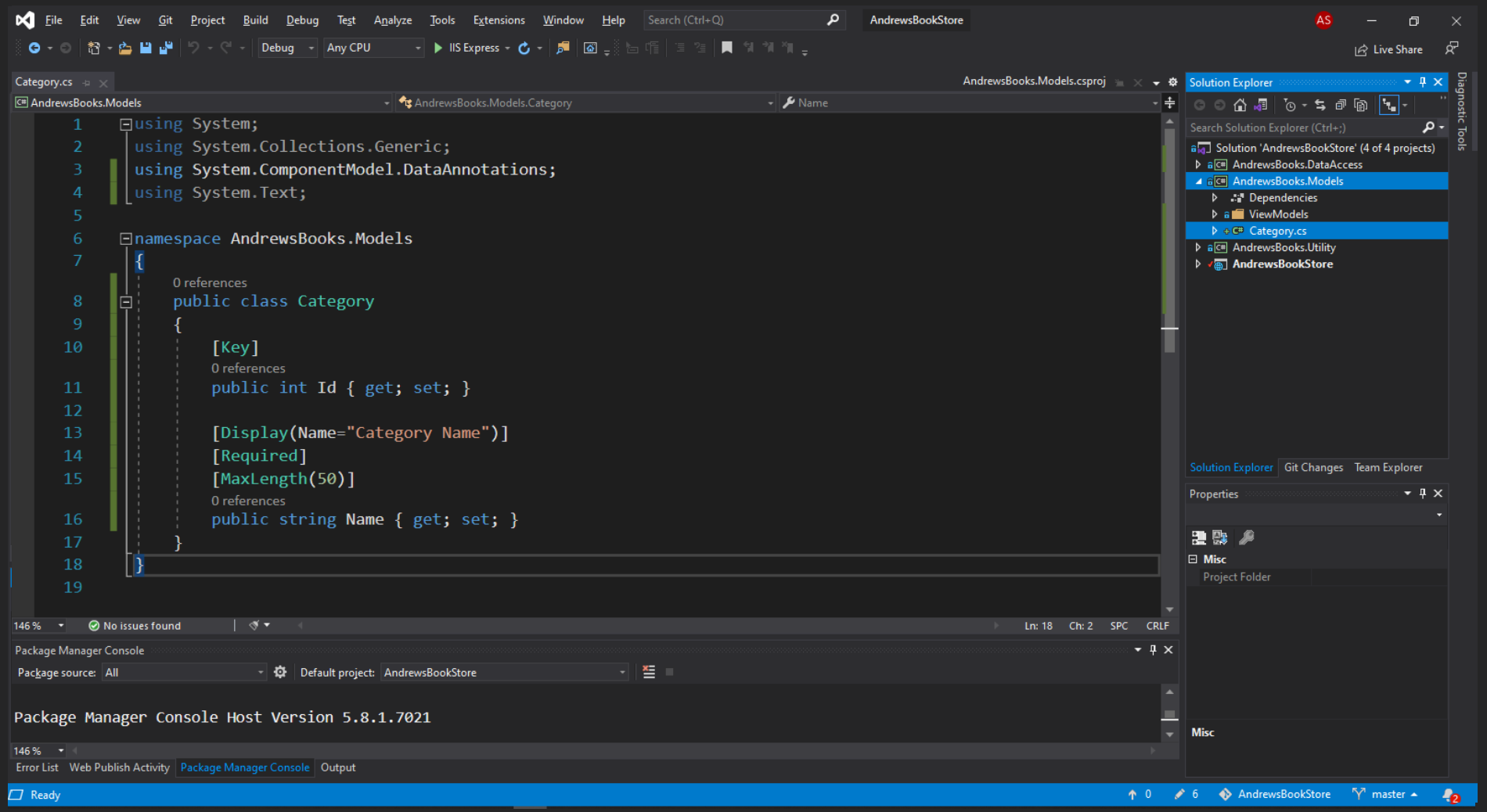


- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify



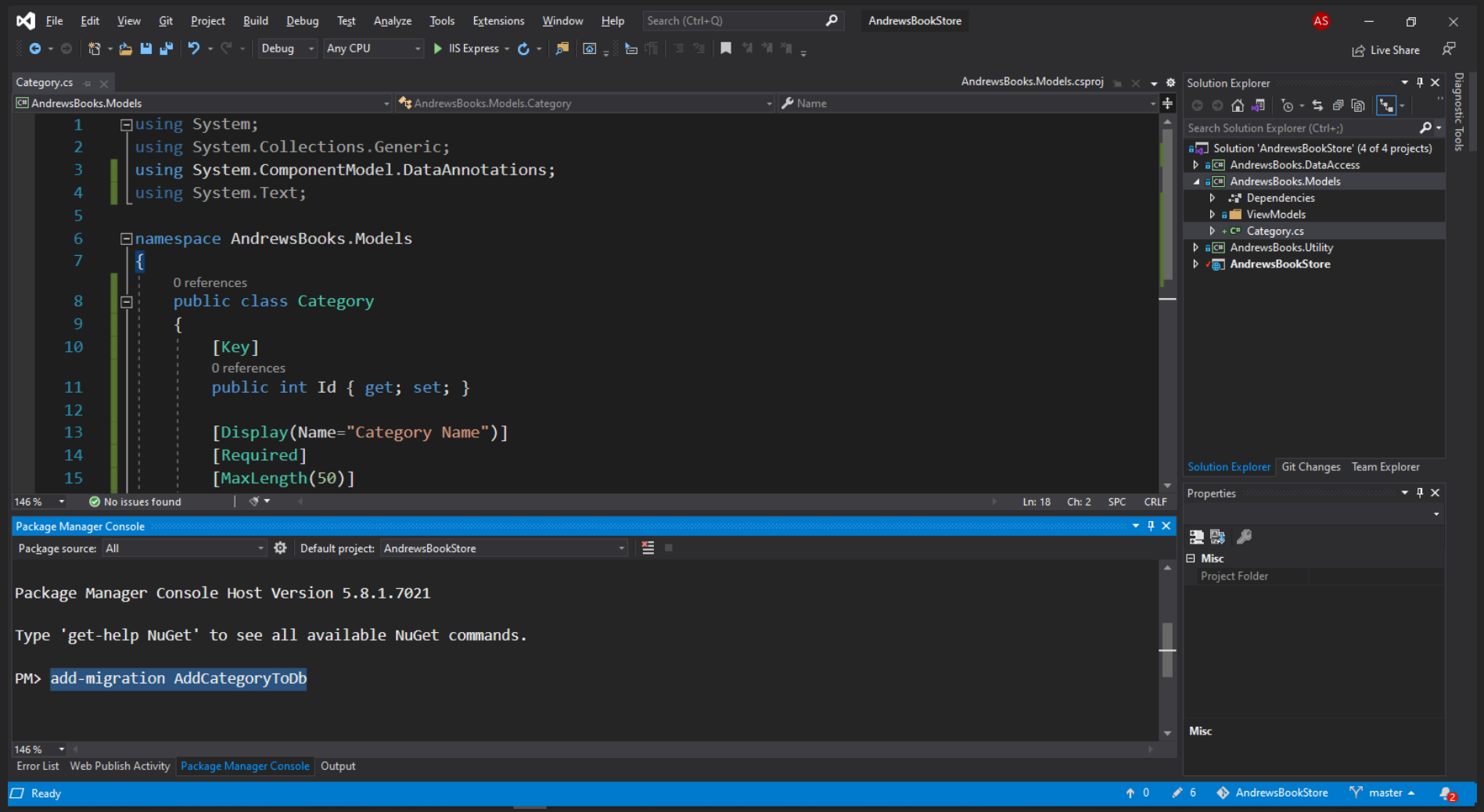
- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console



- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

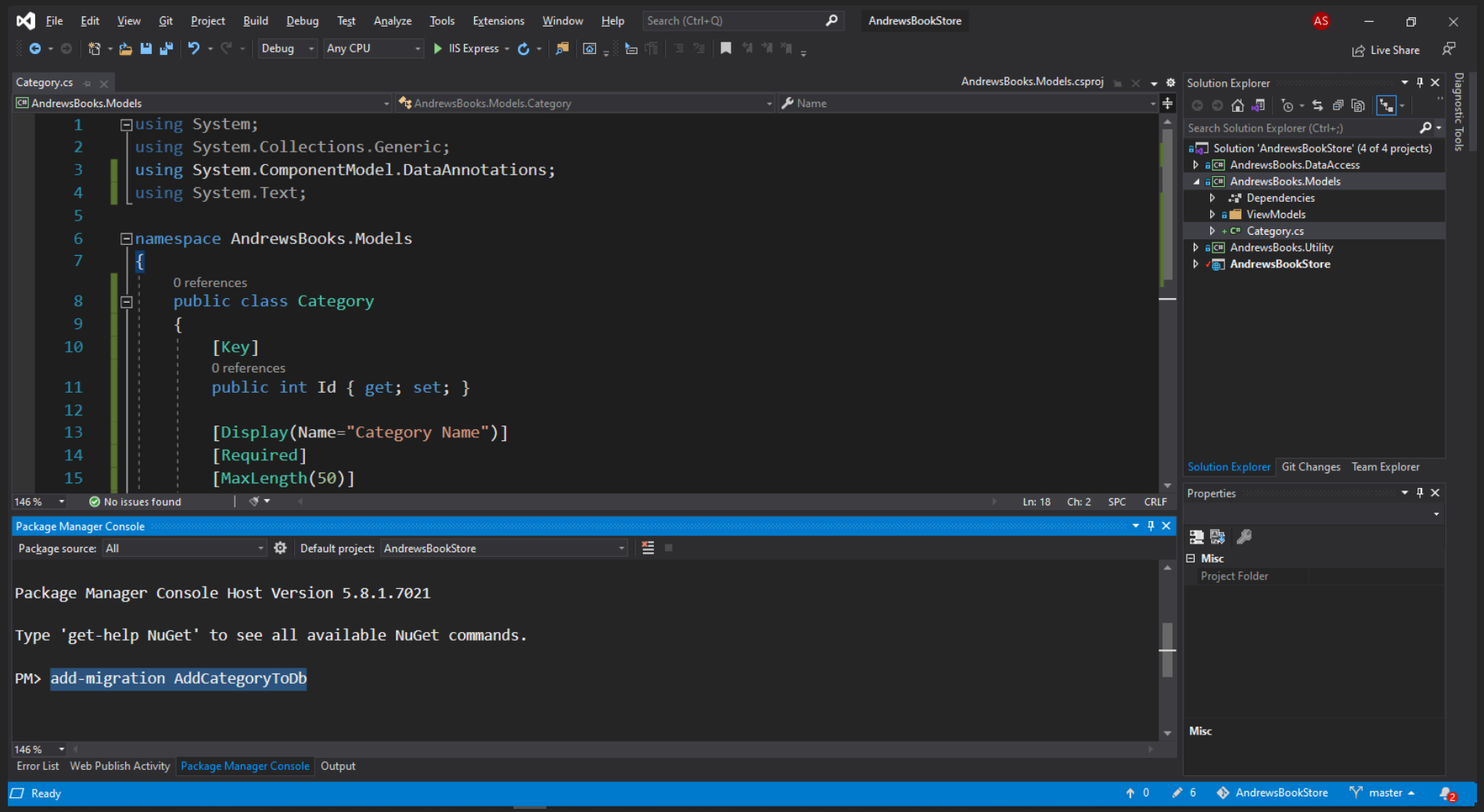
- Add the migration via the PM Console



- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

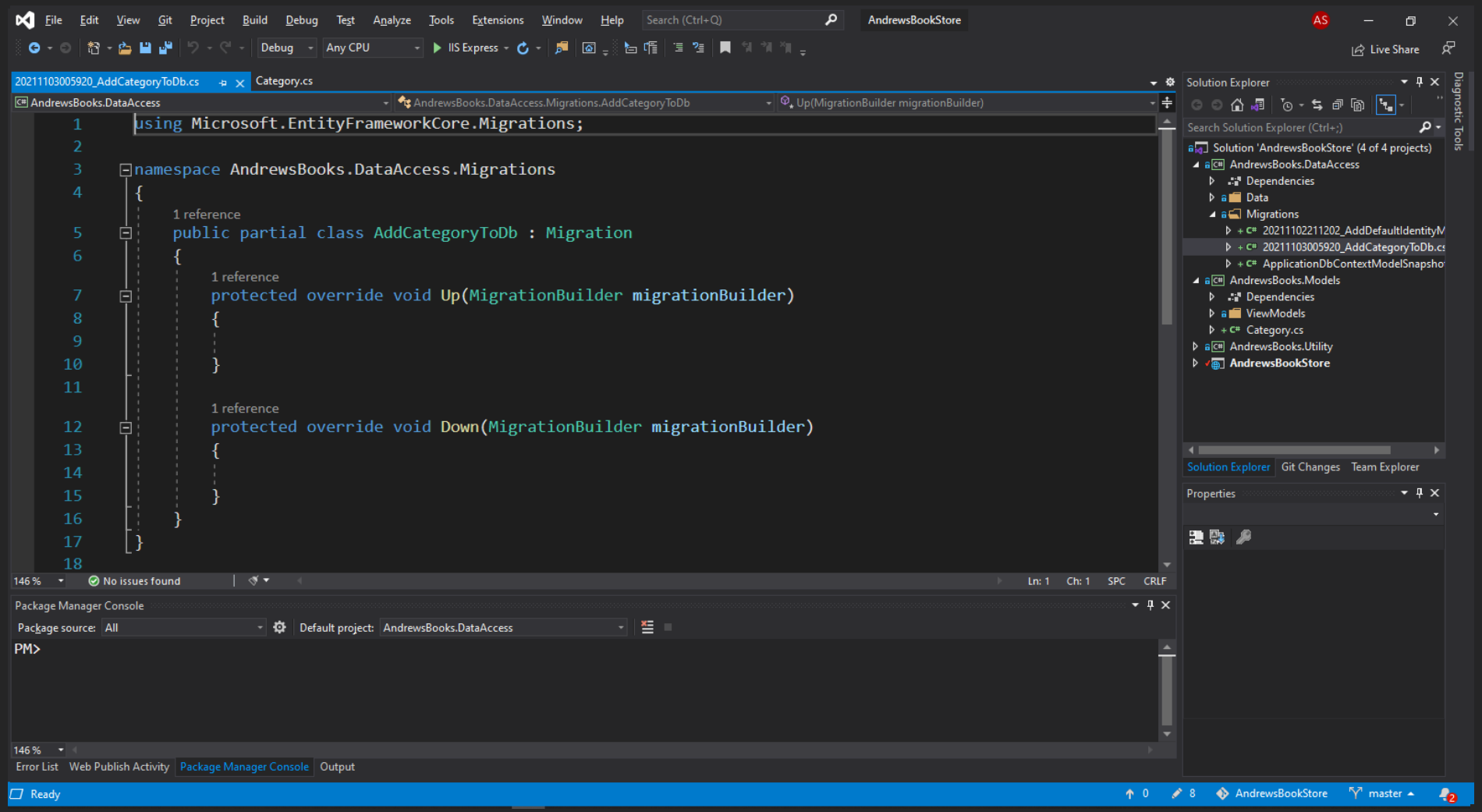
- The new migration file will be empty because it hasn't been added to the Application DB Context



- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

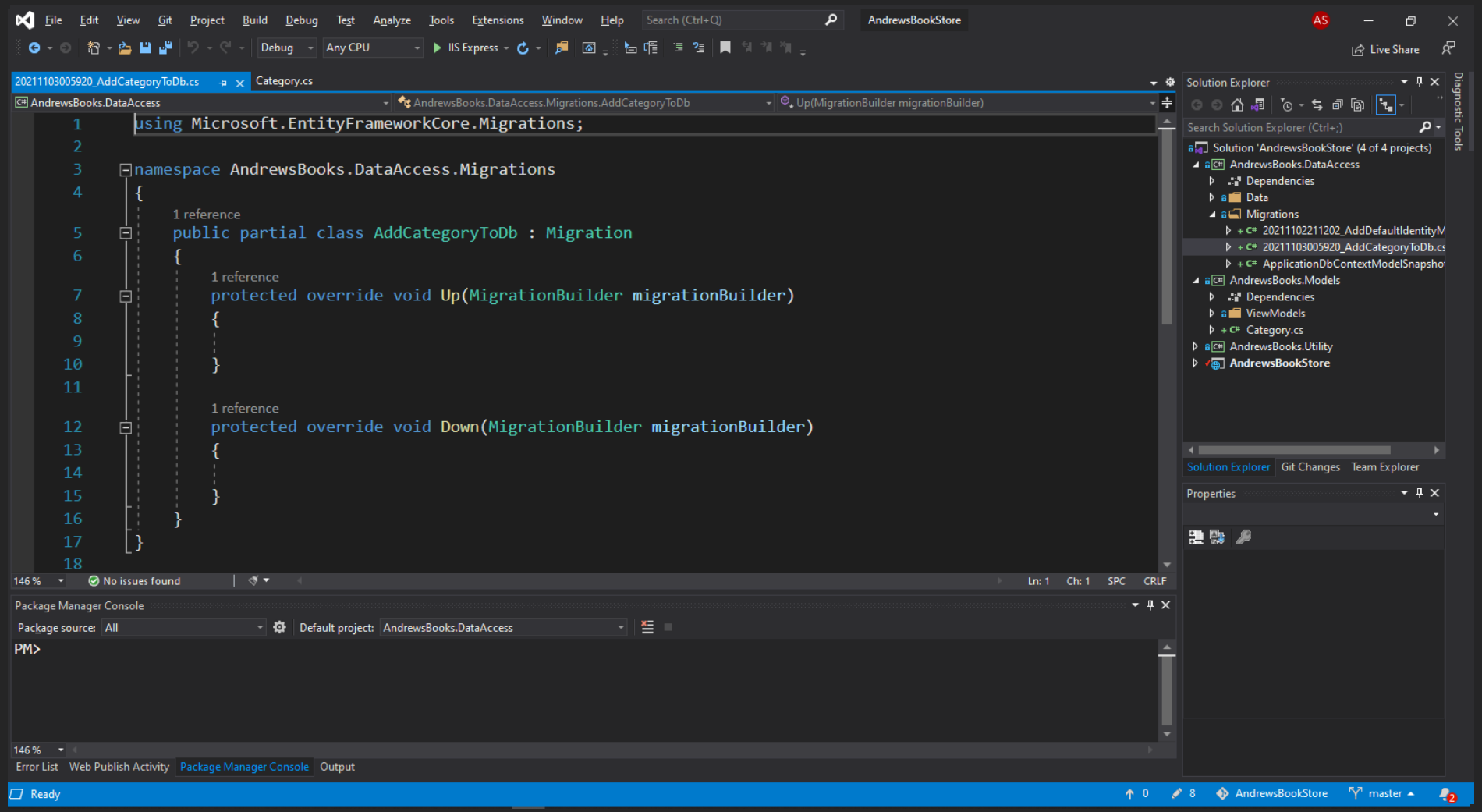


- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

- Update this and note the added using statement

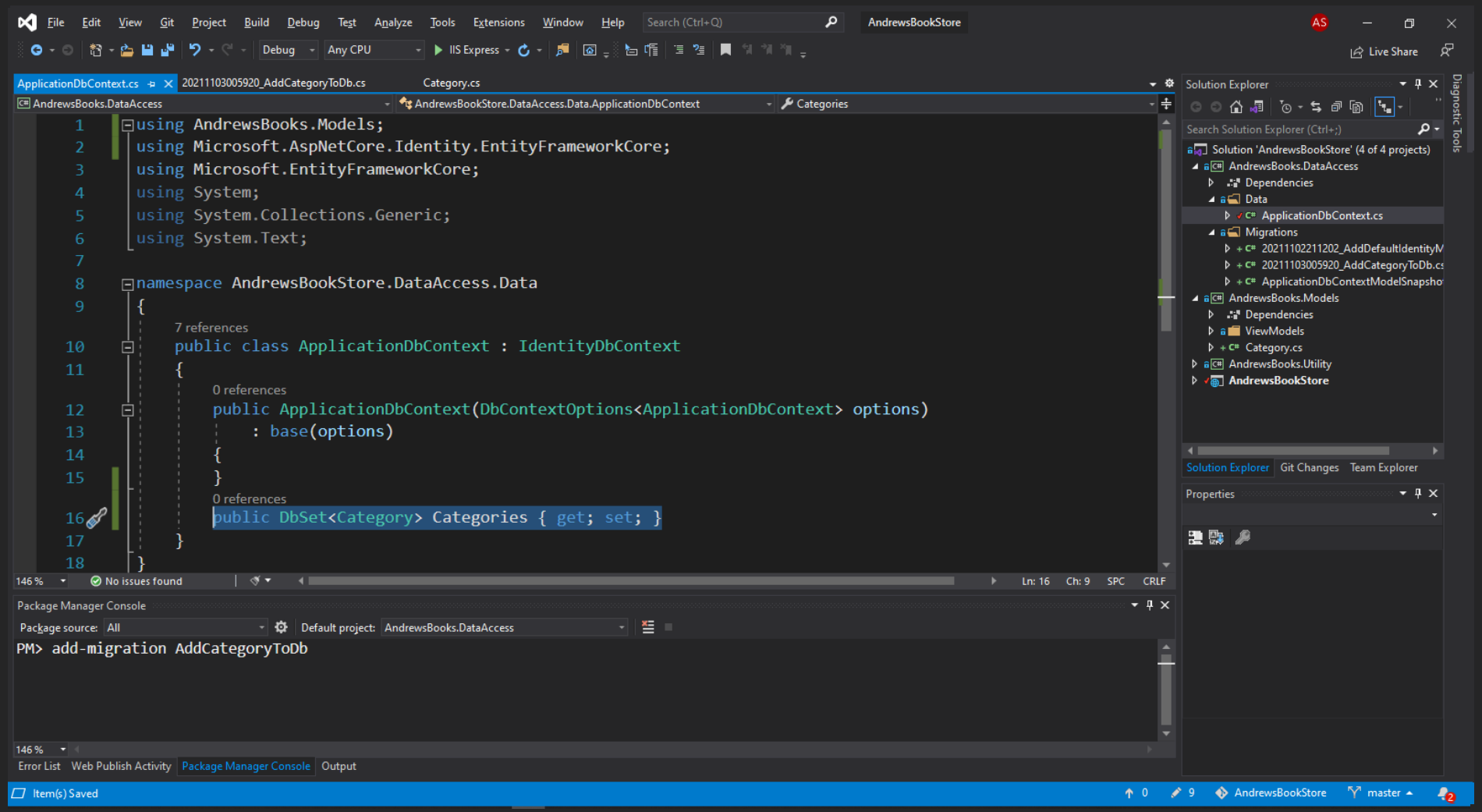


- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

- Update this and note the added using statement

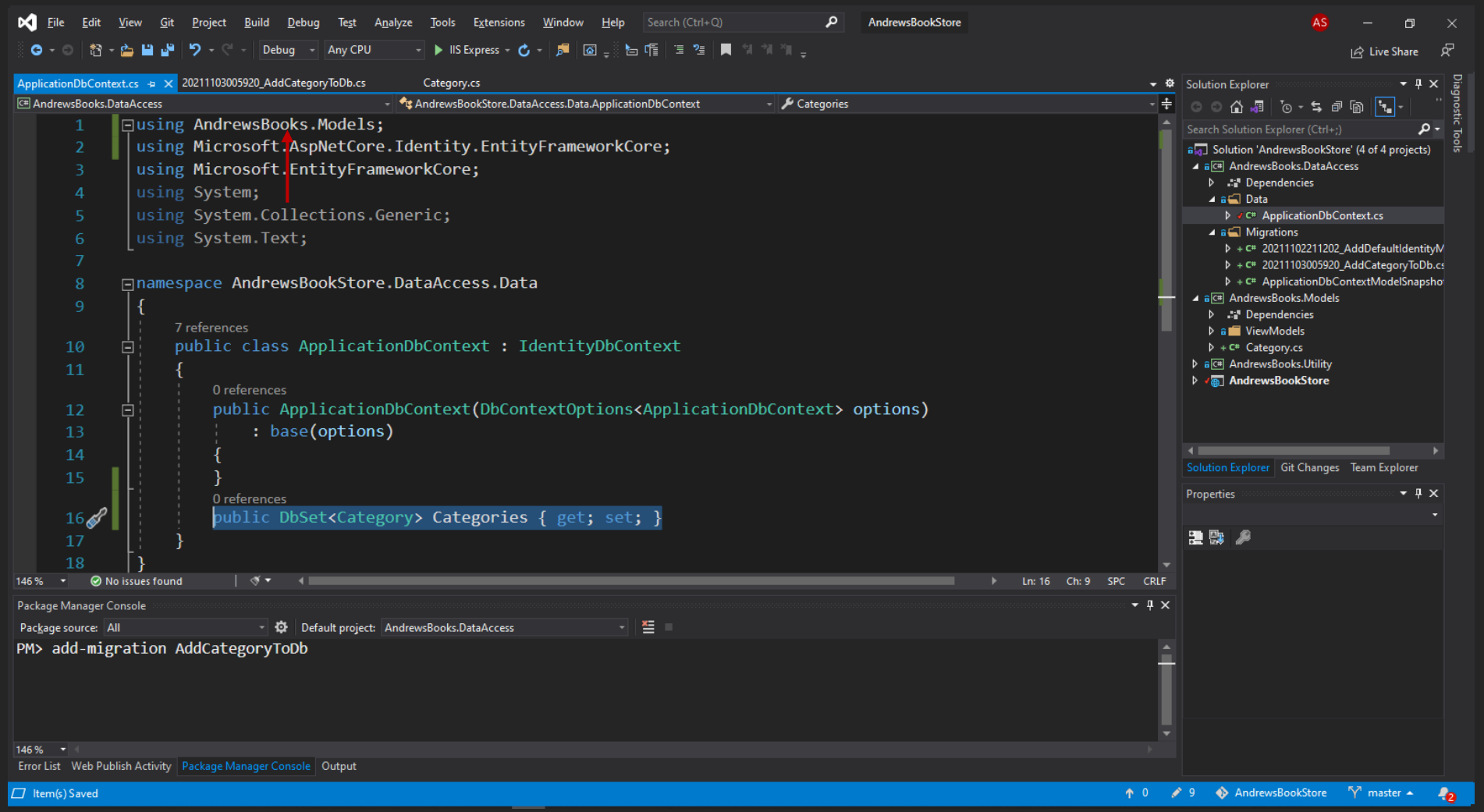


- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

- Update this and note the added using statement

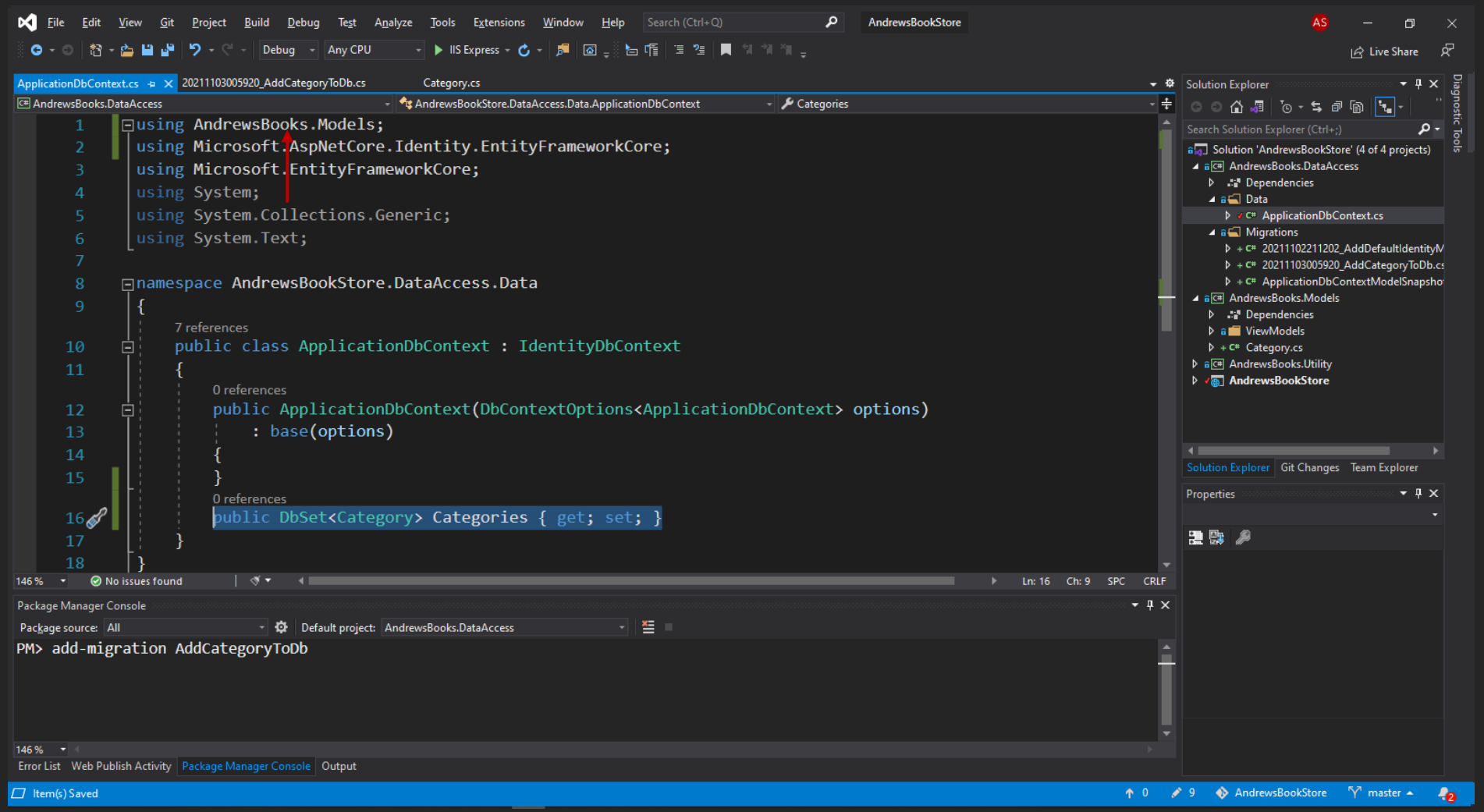


- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

- Update this and note the added using statement
- Re-run the add-migration and review the changes to the AddCategoryToDb (resolve the duplication error that will occur).



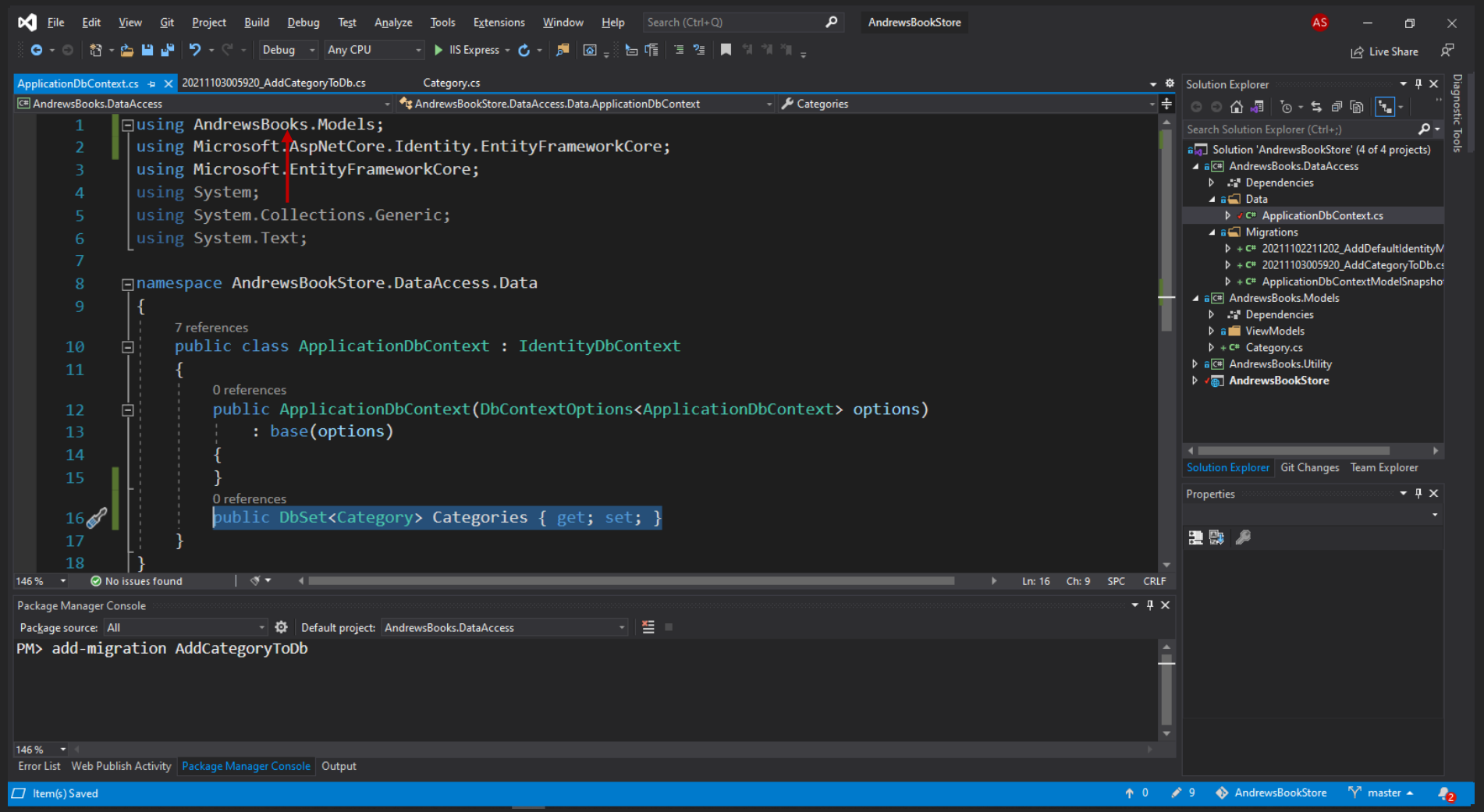
- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

- Update this and note the added using statement
- Re-run the add-migration and review the changes to the AddCategoryToDb (resolve the duplication error that will occur).

- Update the database, confirm the new Categories table via the SQL SOE and push commits to GitHub



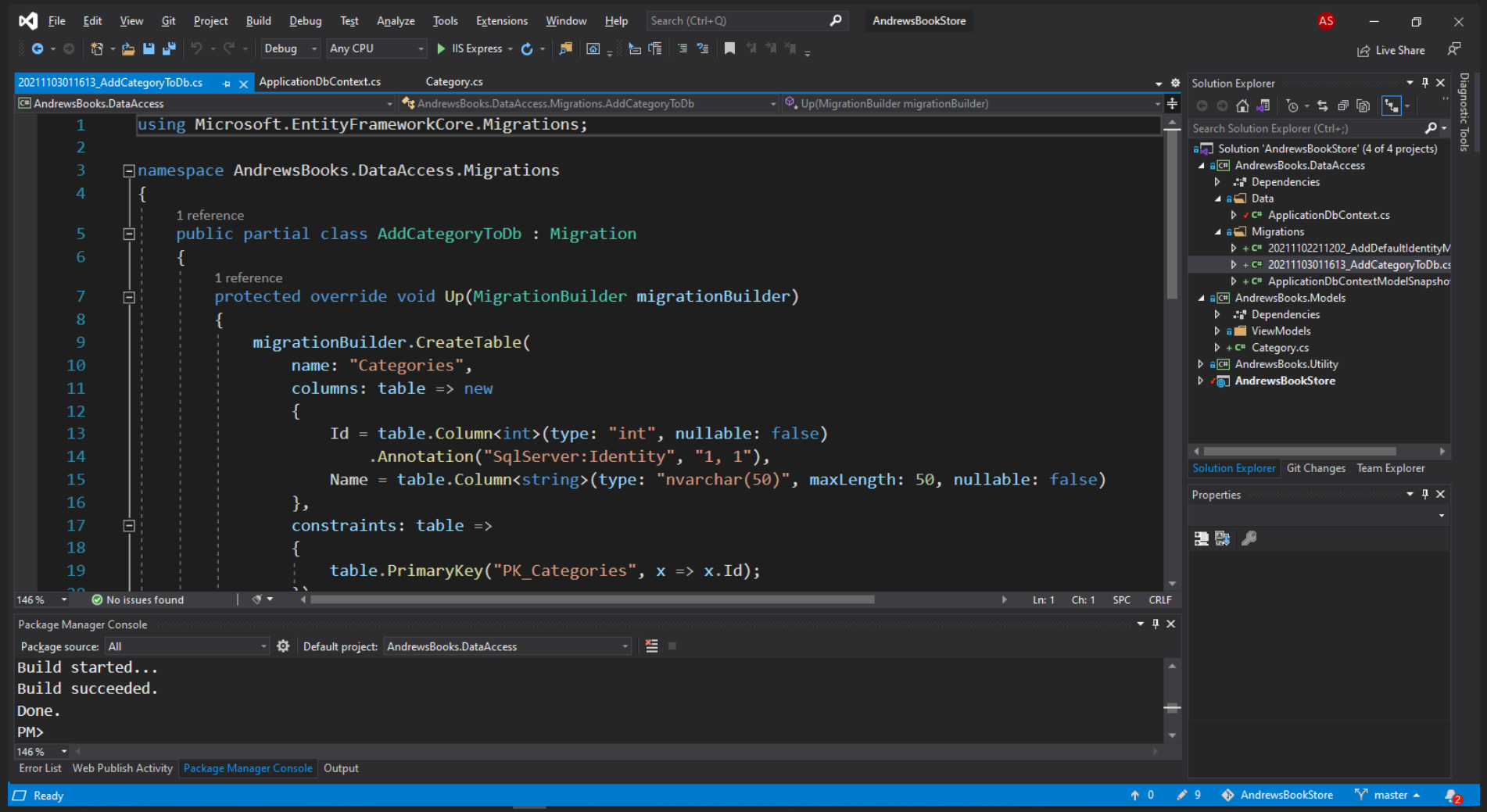
- Add a new table to the DB by creating a Category model and push it to the DB:
- Add a new class file to the .Models project and modify

- Add the migration via the PM Console

- The new migration file will be empty because it hasn't been added to the Application DB Context

- Update this and note the added using statement
- Re-run the add-migration and review the changes to the AddCategoryToDb (resolve the duplication error that will occur).

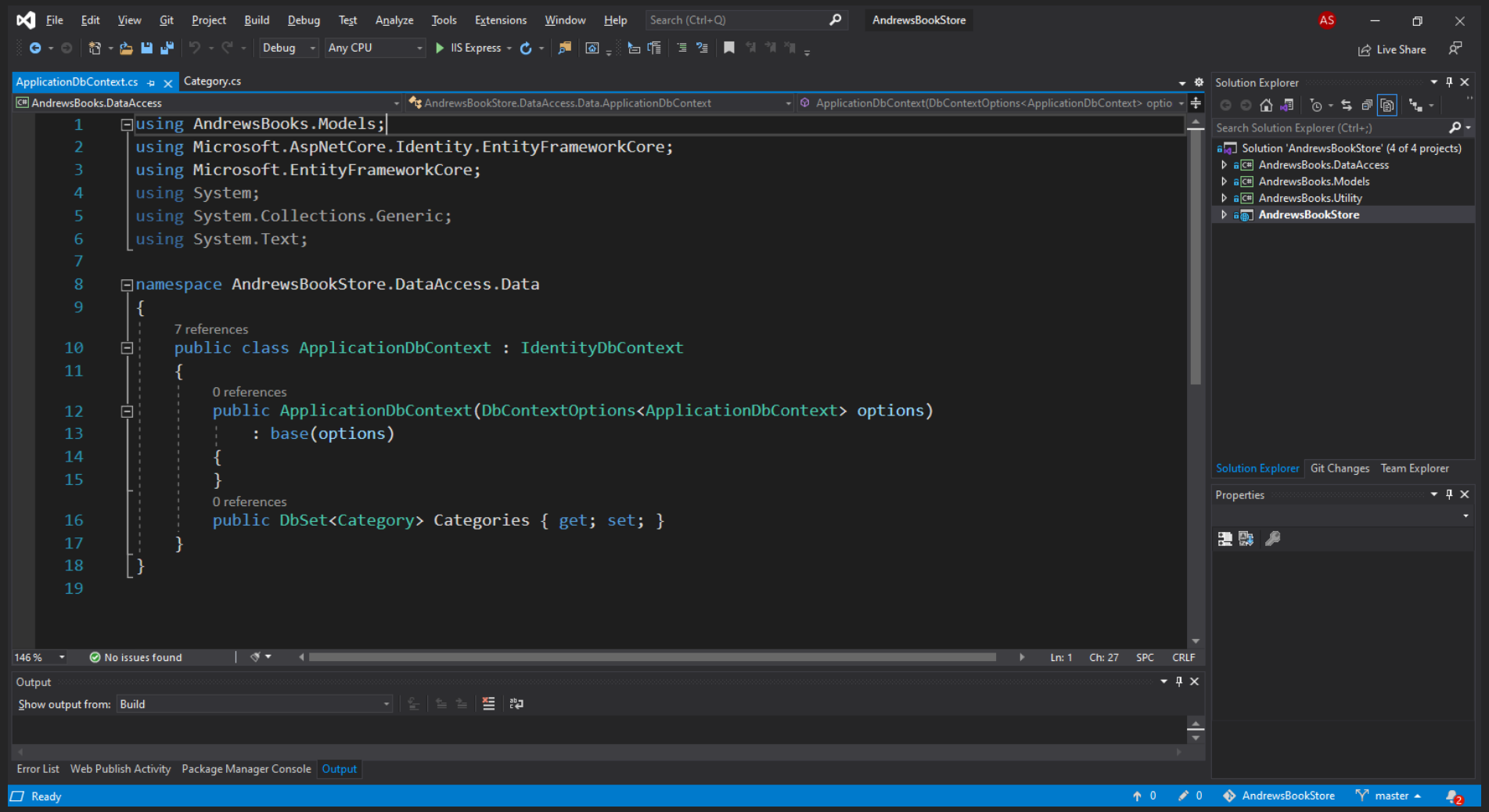
- Update the database, confirm the new Categories table via the SQL SOE and push commits to GitHub



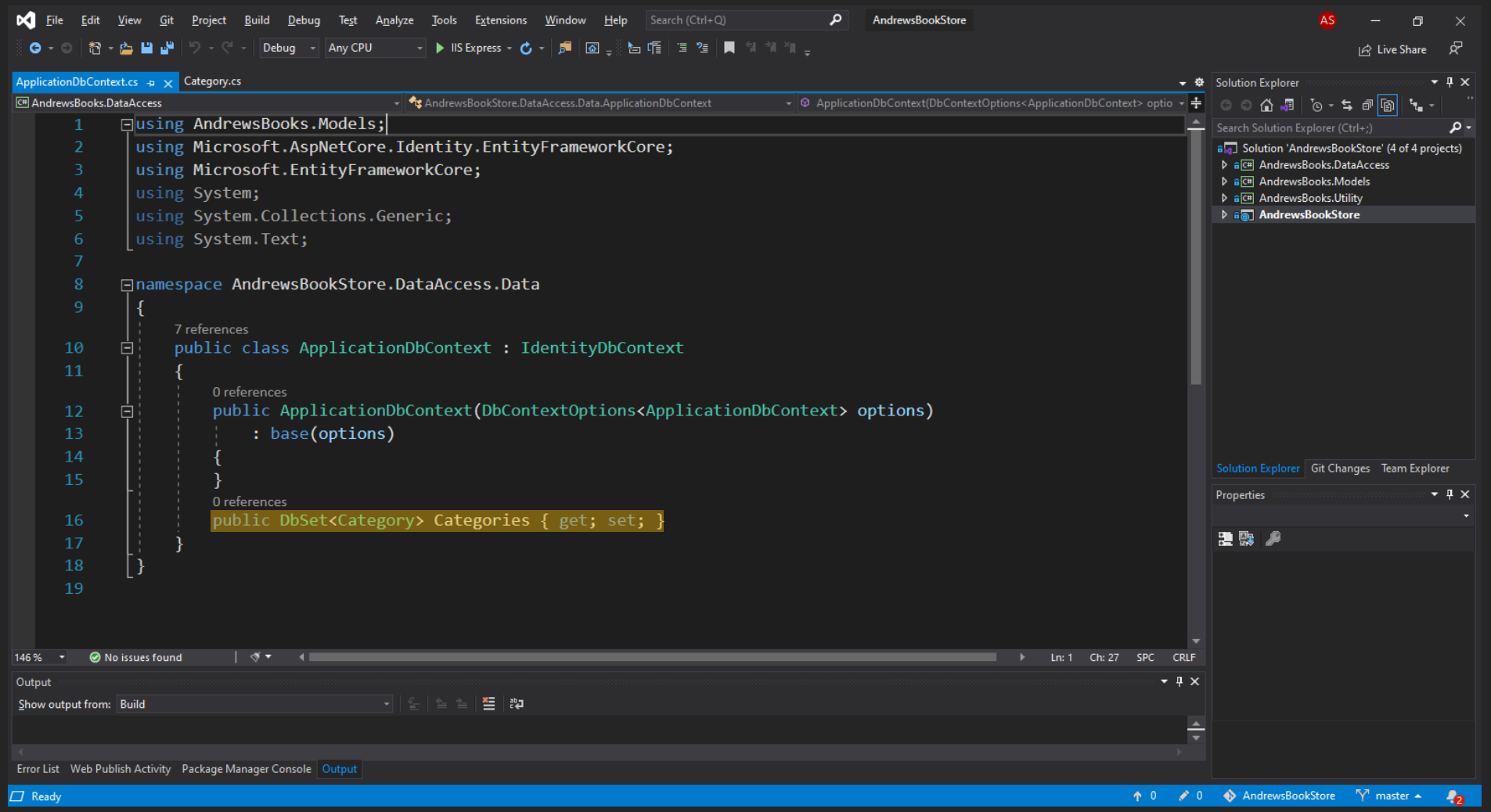
Build the Project (Part 2)

2.2 Repository

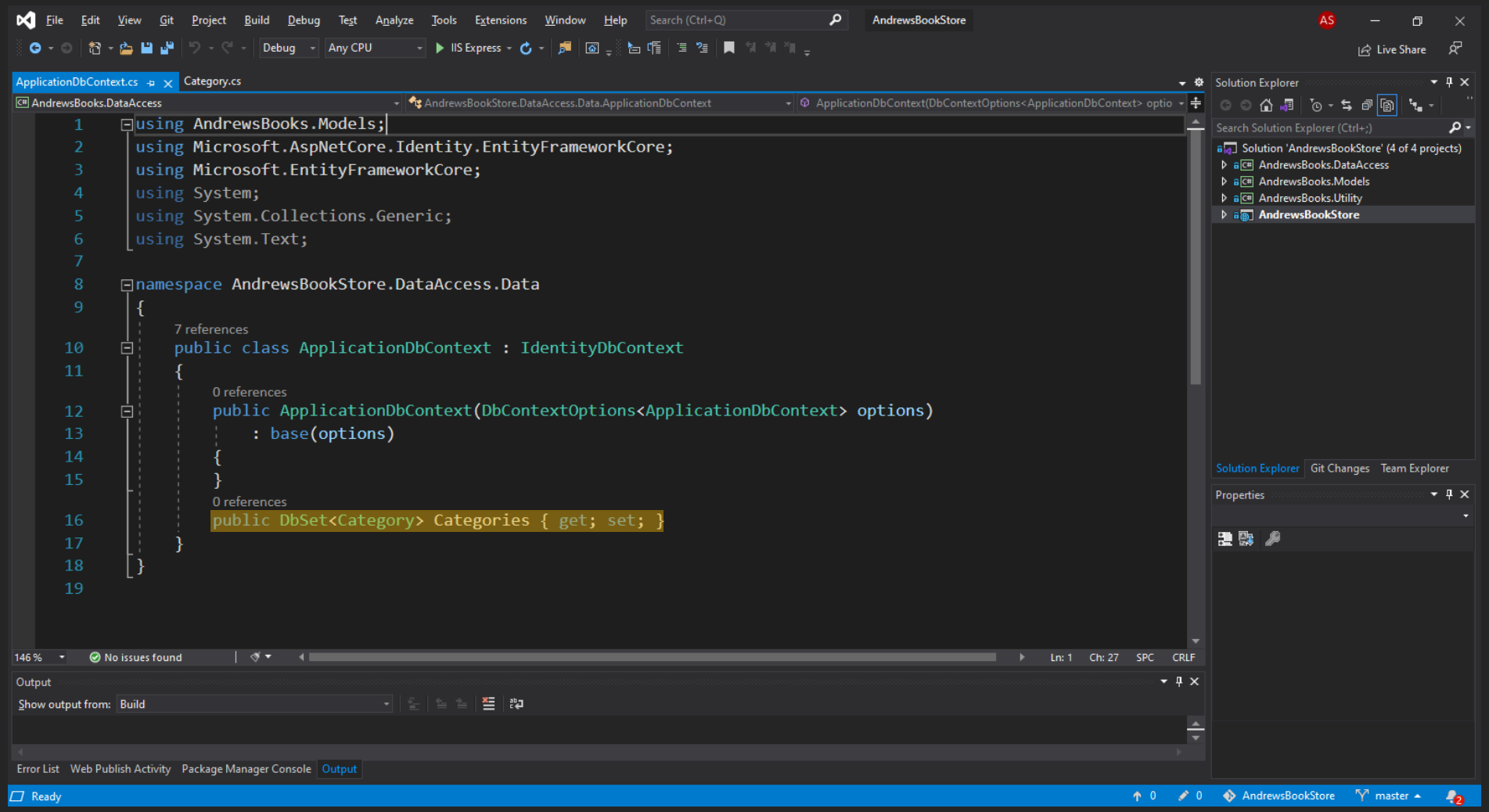
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.



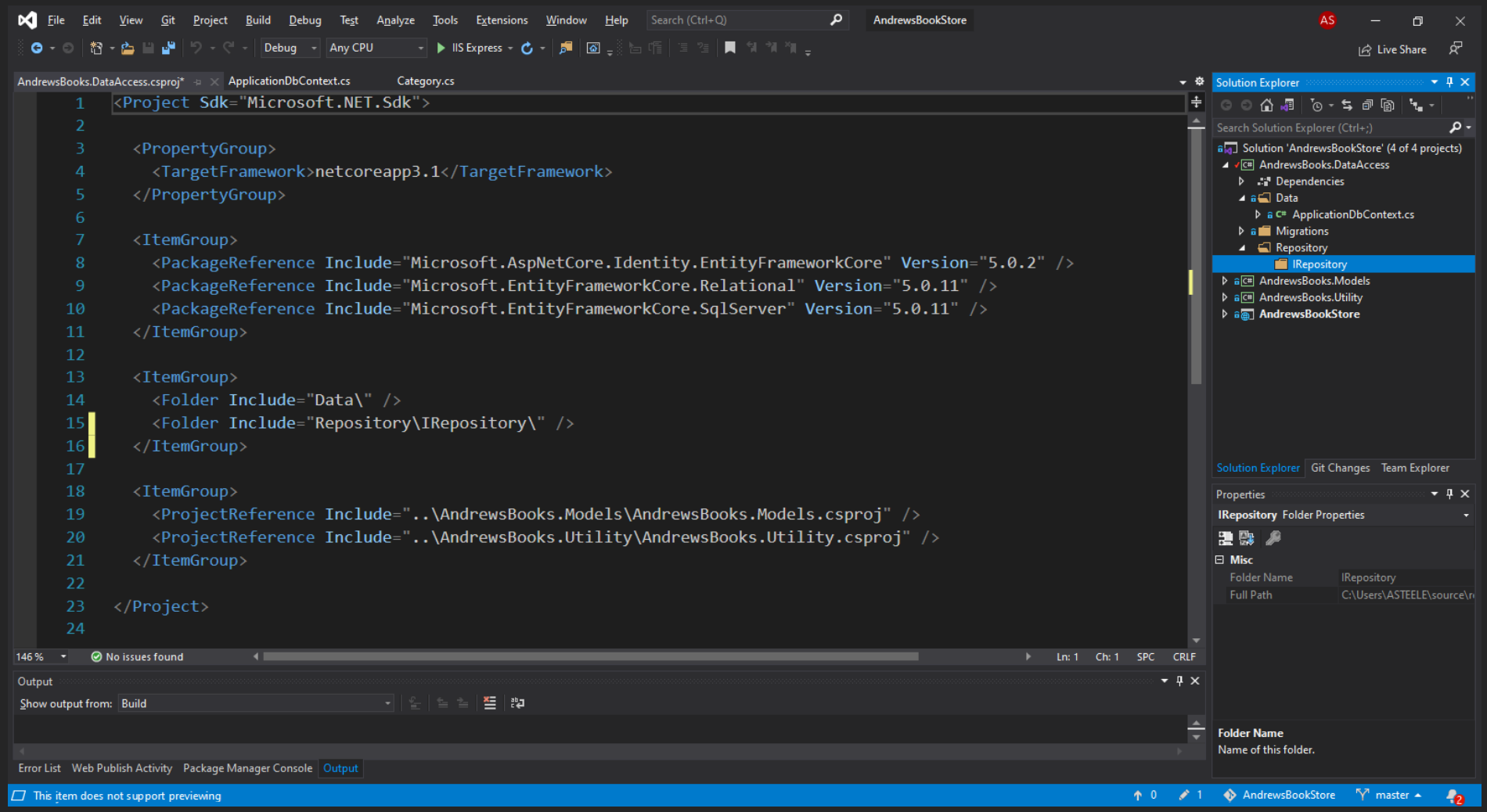
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.



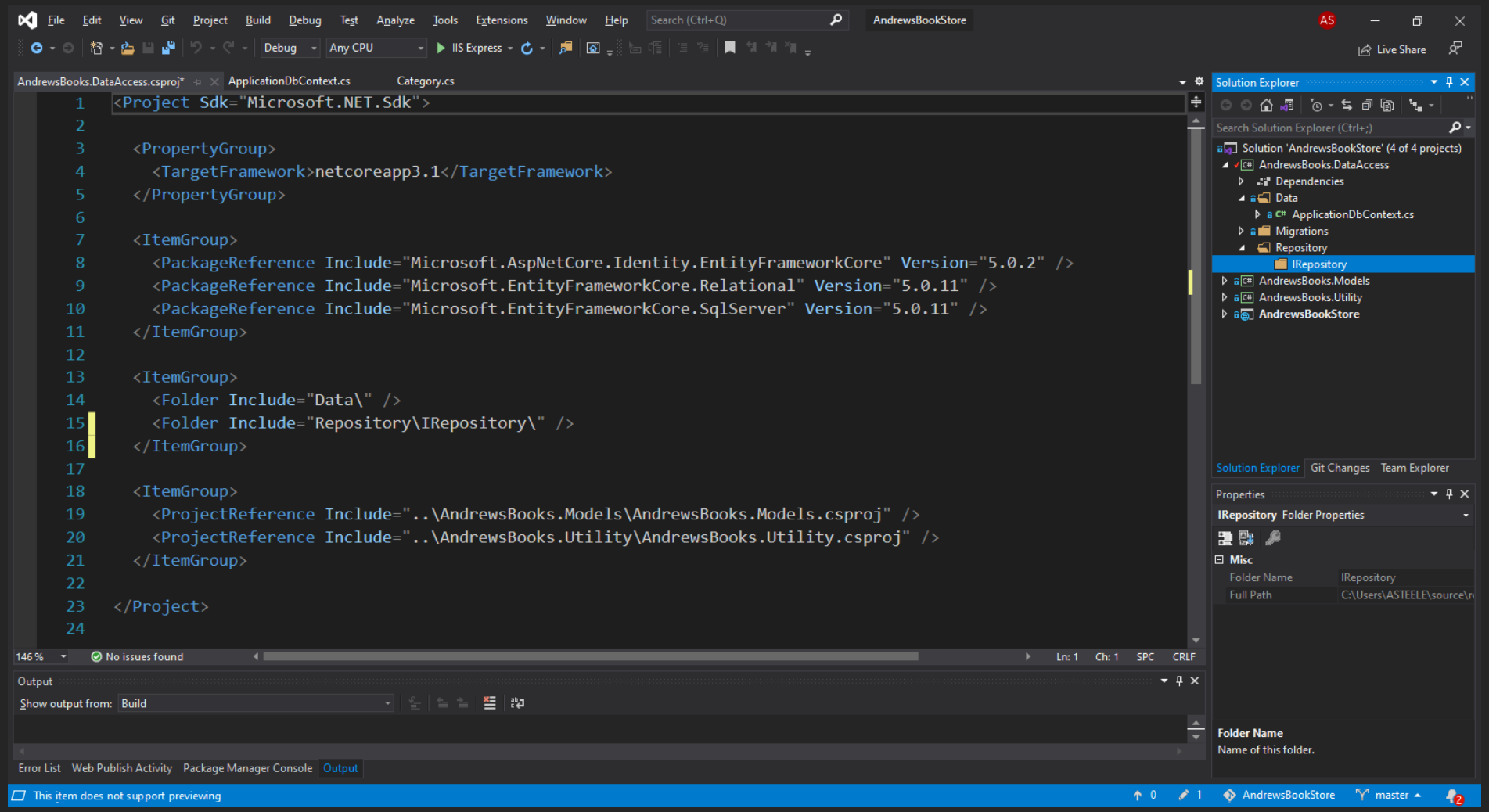
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.



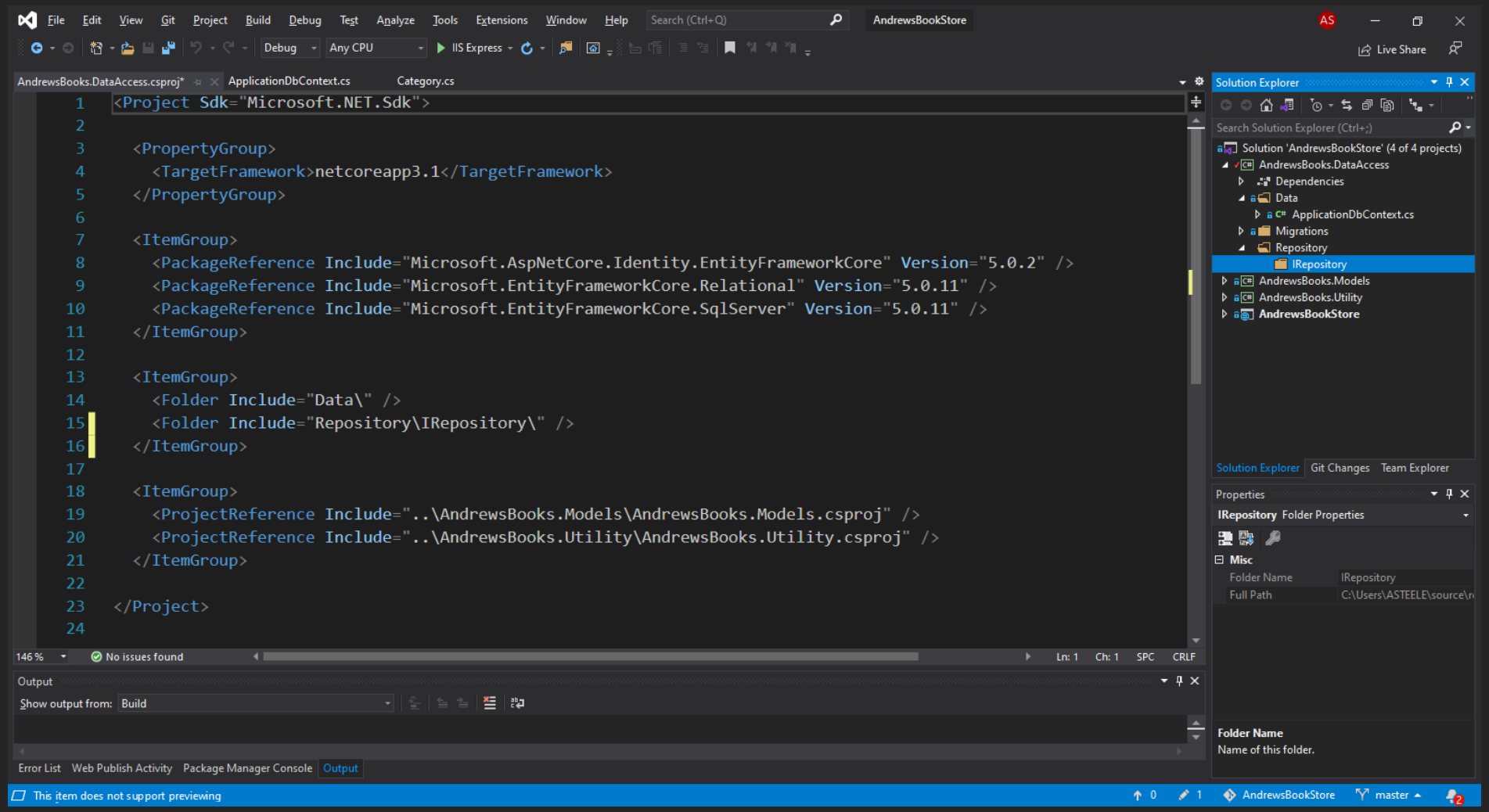
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.



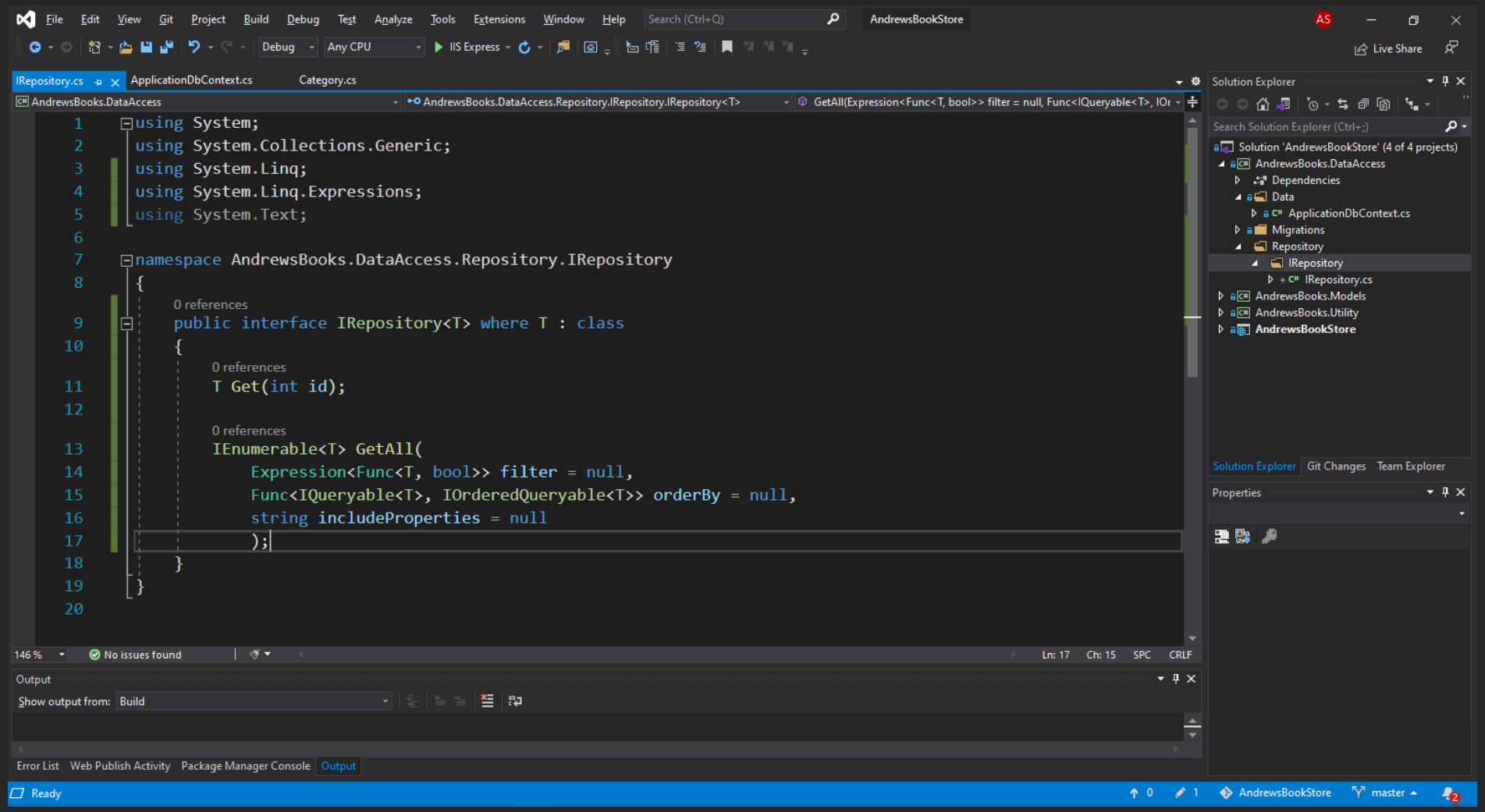
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs



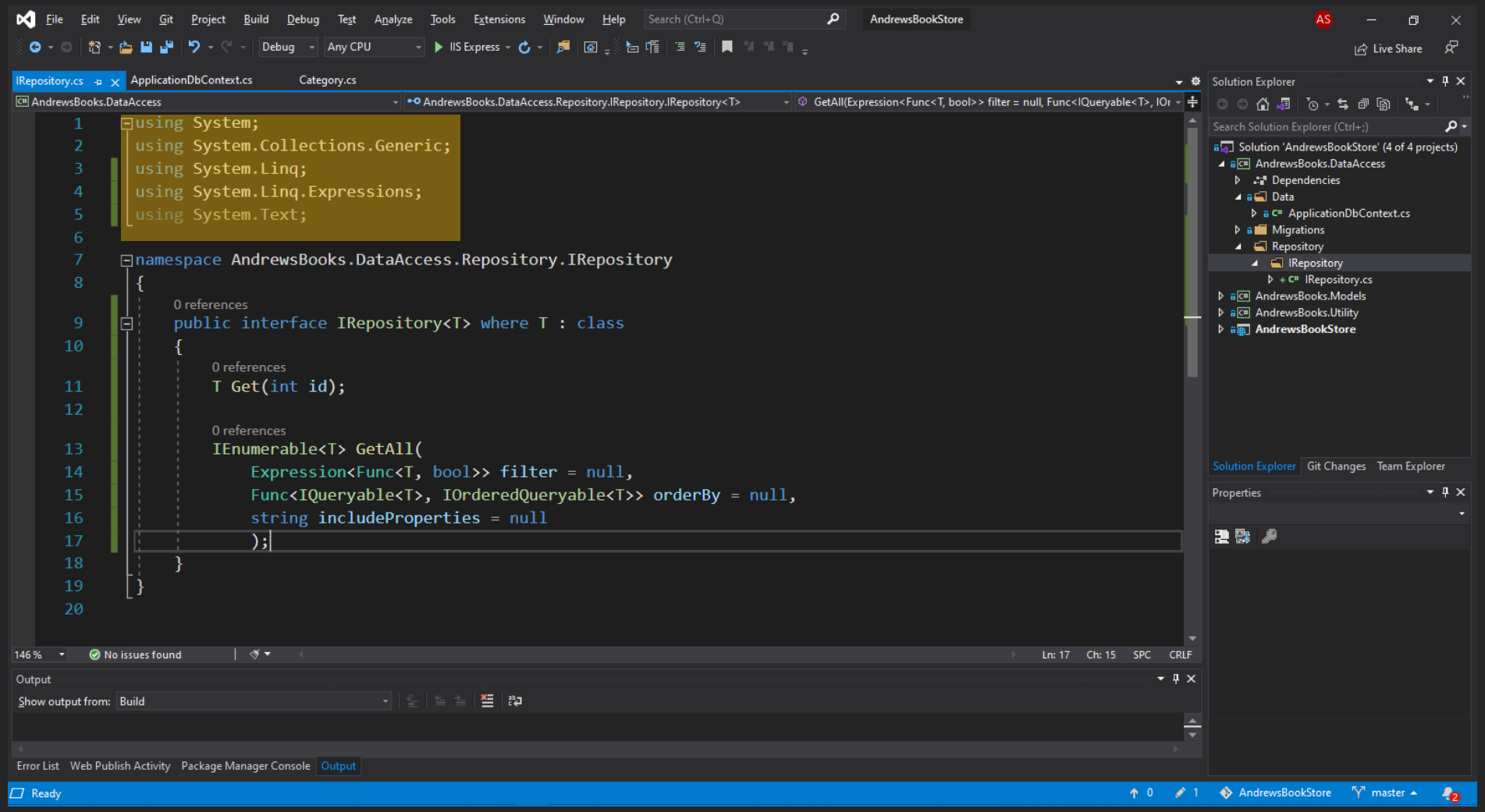
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs
- Modify so it can be used on the Category class to do all the CRUD operations (note using statements).



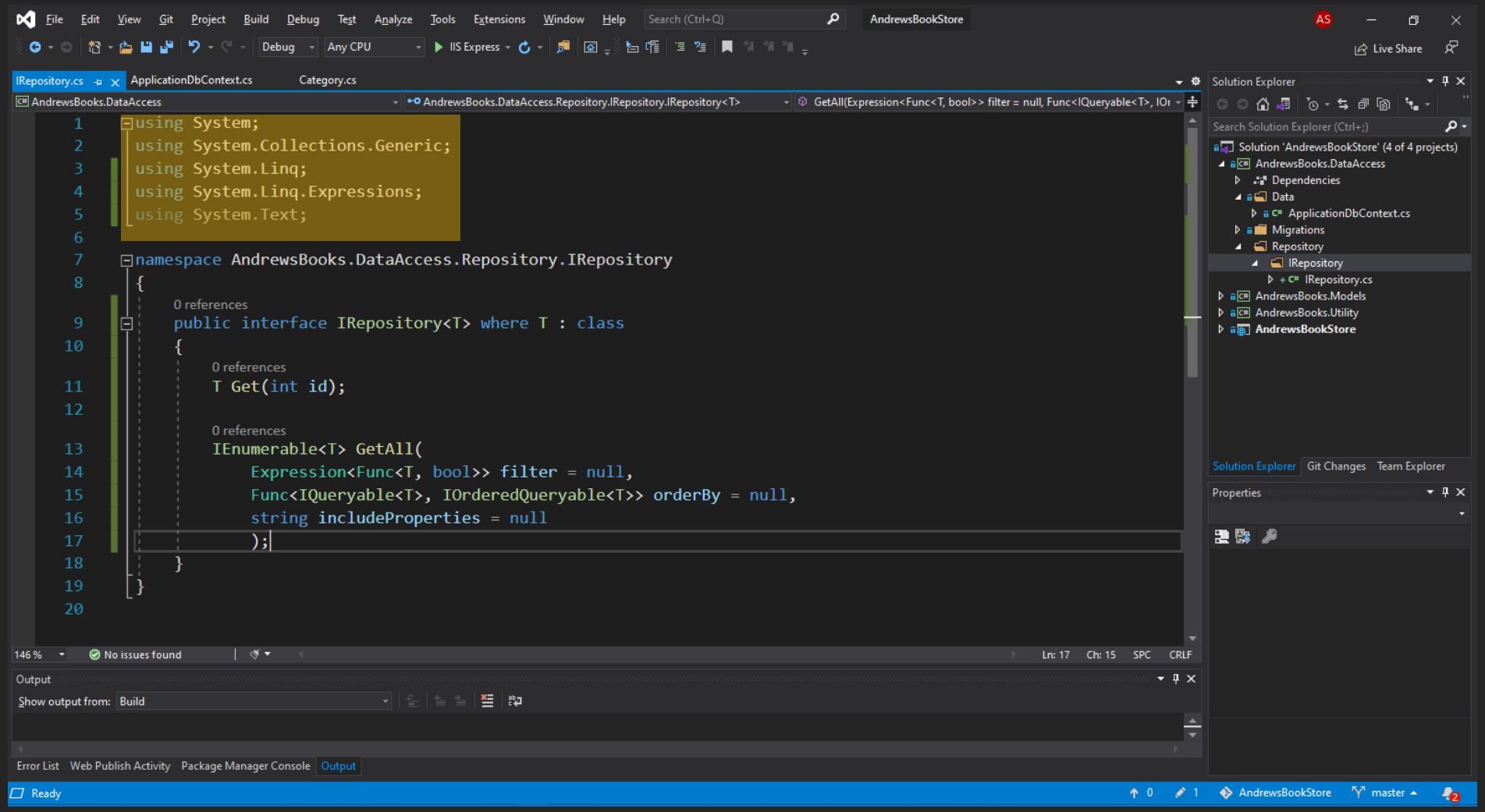
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs
- Modify so it can be used on the Category class to do all the CRUD operations (note using statements).



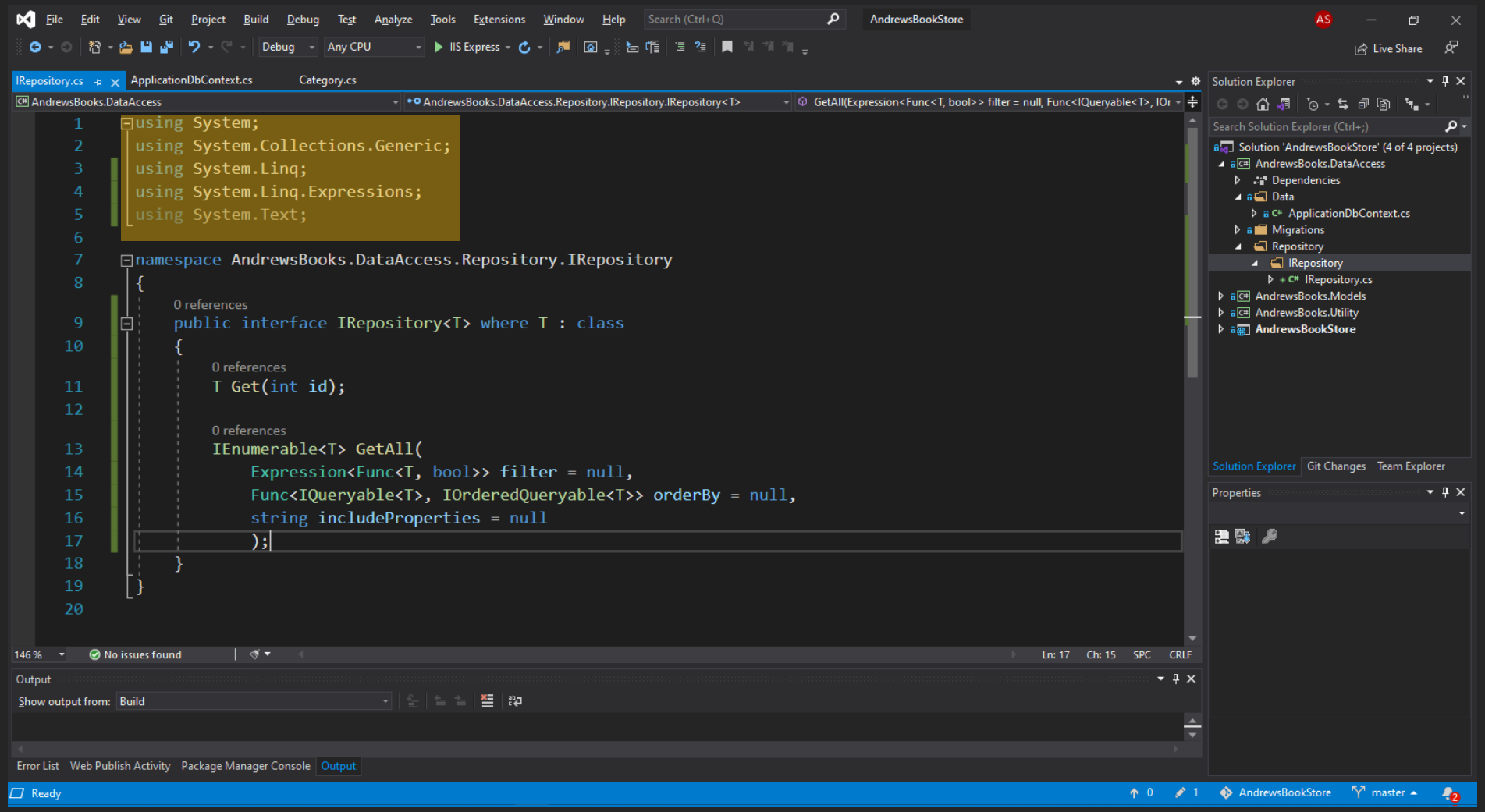
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs
- Modify so it can be used on the Category class to do all the CRUD operations (note using statements).



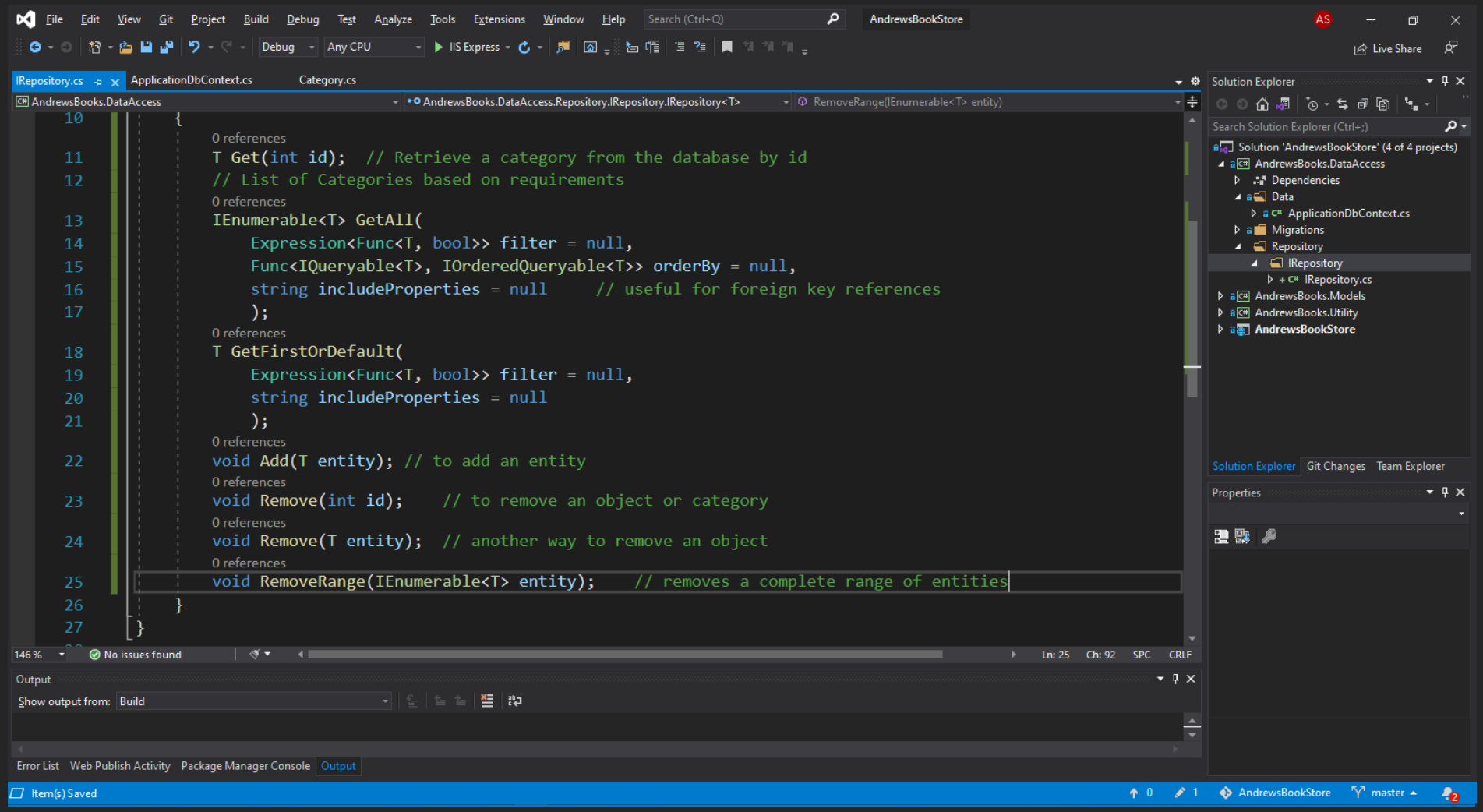
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs
- Modify so it can be used on the Category class to do all the CRUD operations (note using statements).



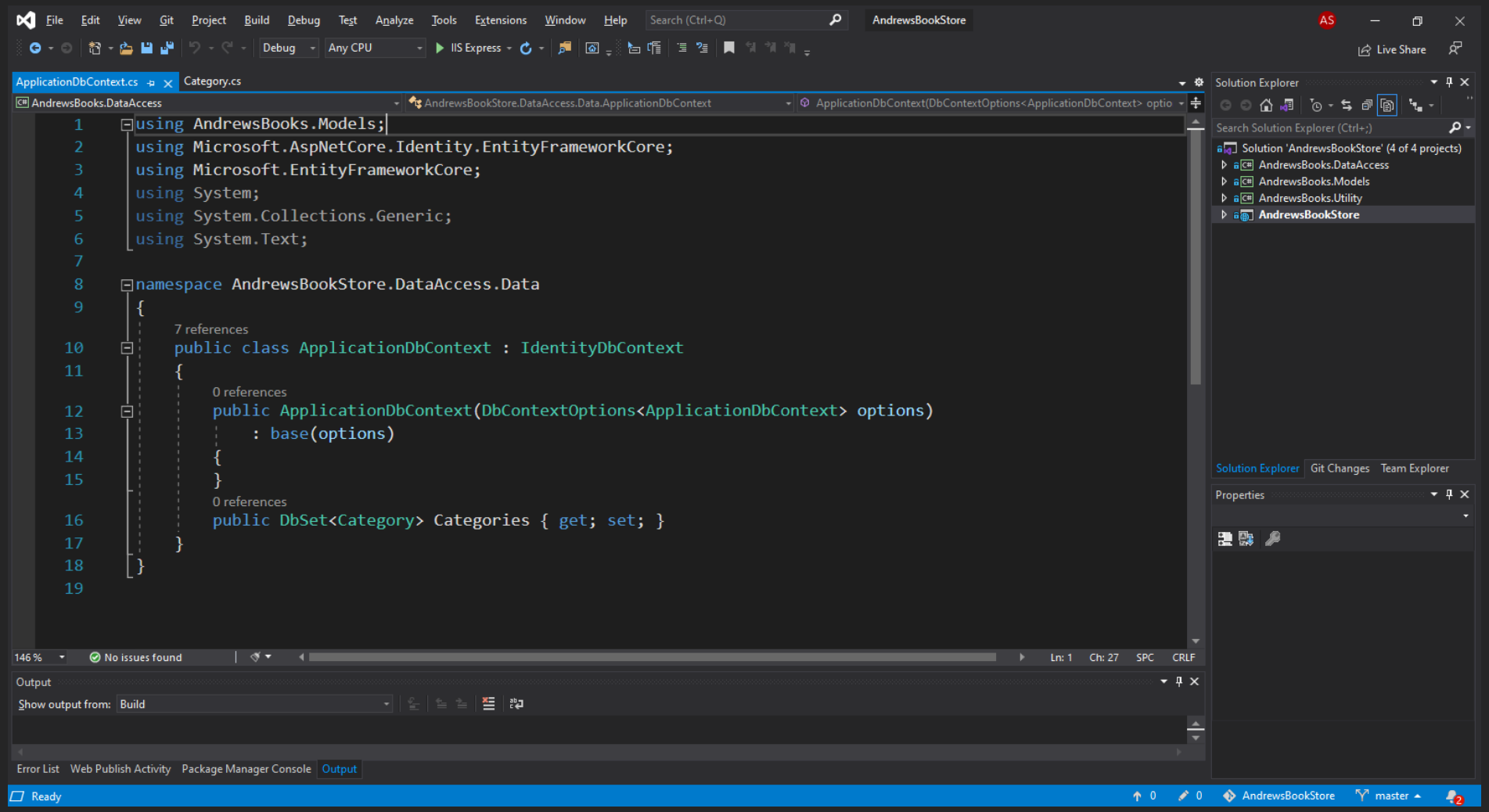
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs
- Modify so it can be used on the Category class to do all the CRUD operations (note using statements).
- Methods to use for CRUD:
  - Get item from the DB
  - List of Categories
  - Add objects
  - Remove objects



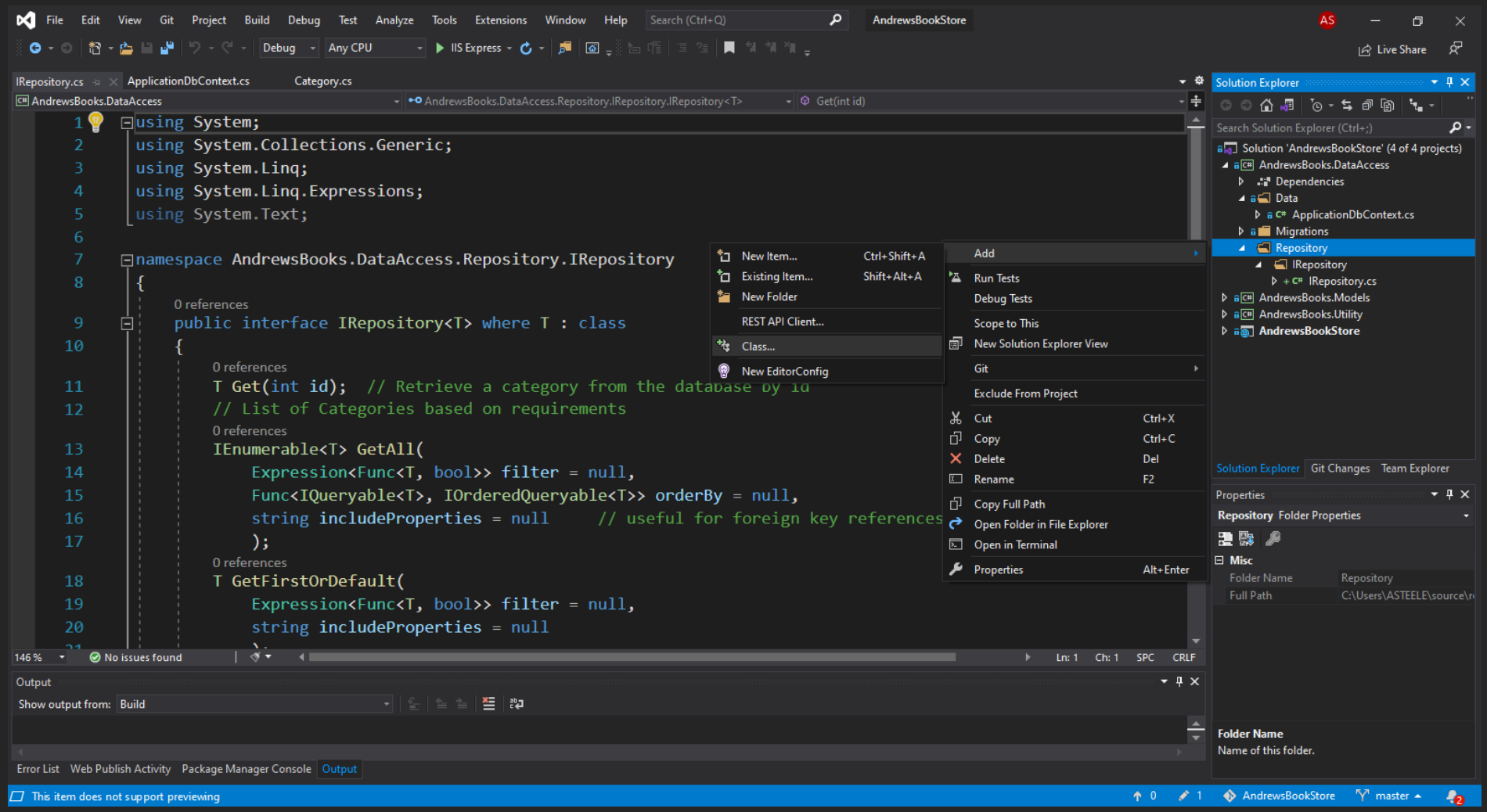
- After creating the Category.cs and adding it to the ApplicationDbContext it's time to implement the Repository, a generic way of accessing common functionality (e.g. getting a record), in the .DataAccess project.
- Add a new folder name it 'Repository' (for class implementations of interfaces) and add an IRepository (for the interfaces) folder inside it.
- Add a new item of type interface to the folder and name it IRepository.cs
- Modify so it can be used on the Category class to do all the CRUD operations (note using statements).
- Methods to use for CRUD:
  - Get item from the DB
  - List of Categories
  - Add objects
  - Remove objects



- Implement the class that implements the repository

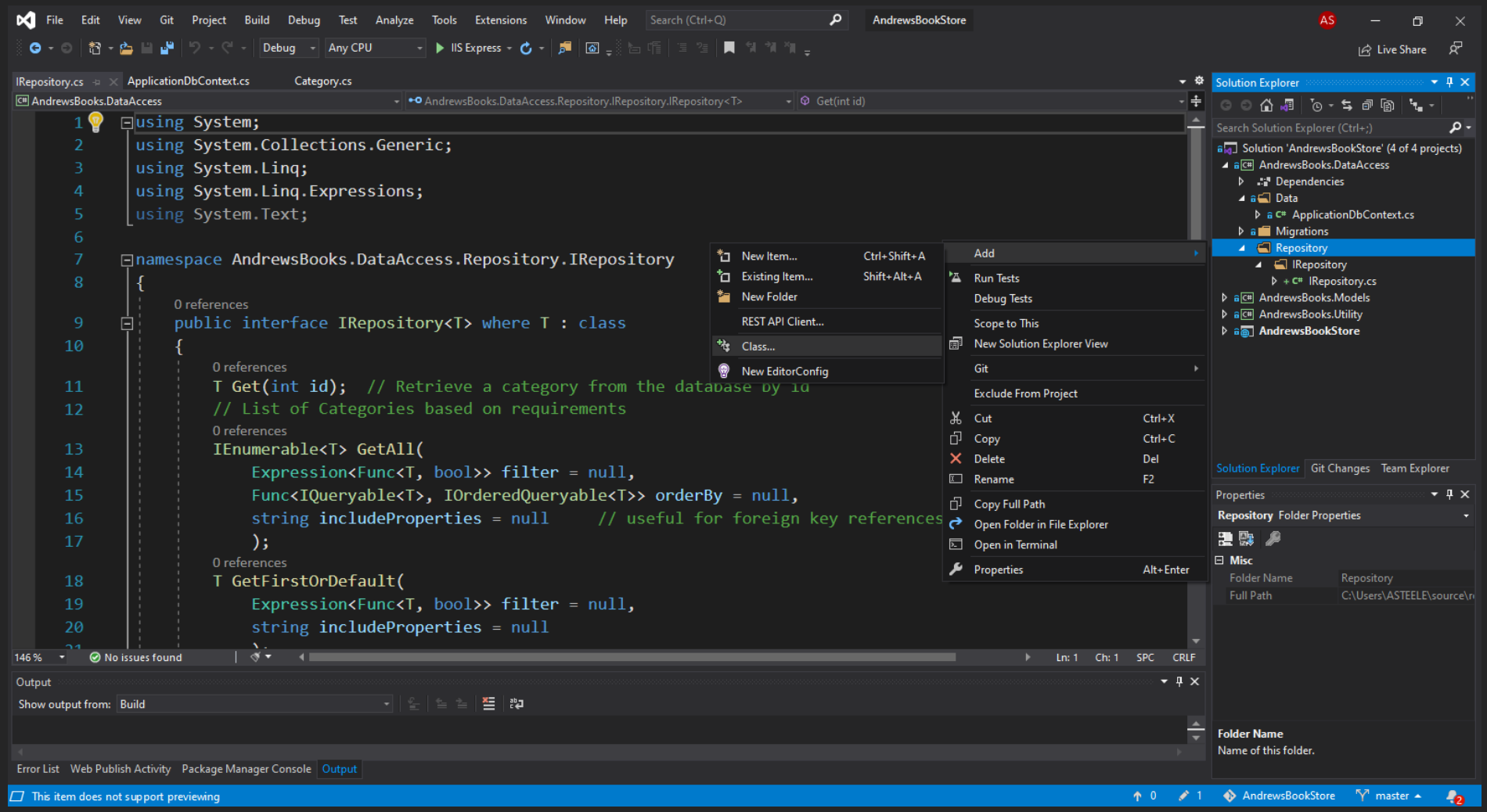


- Implement the class that implements the repository



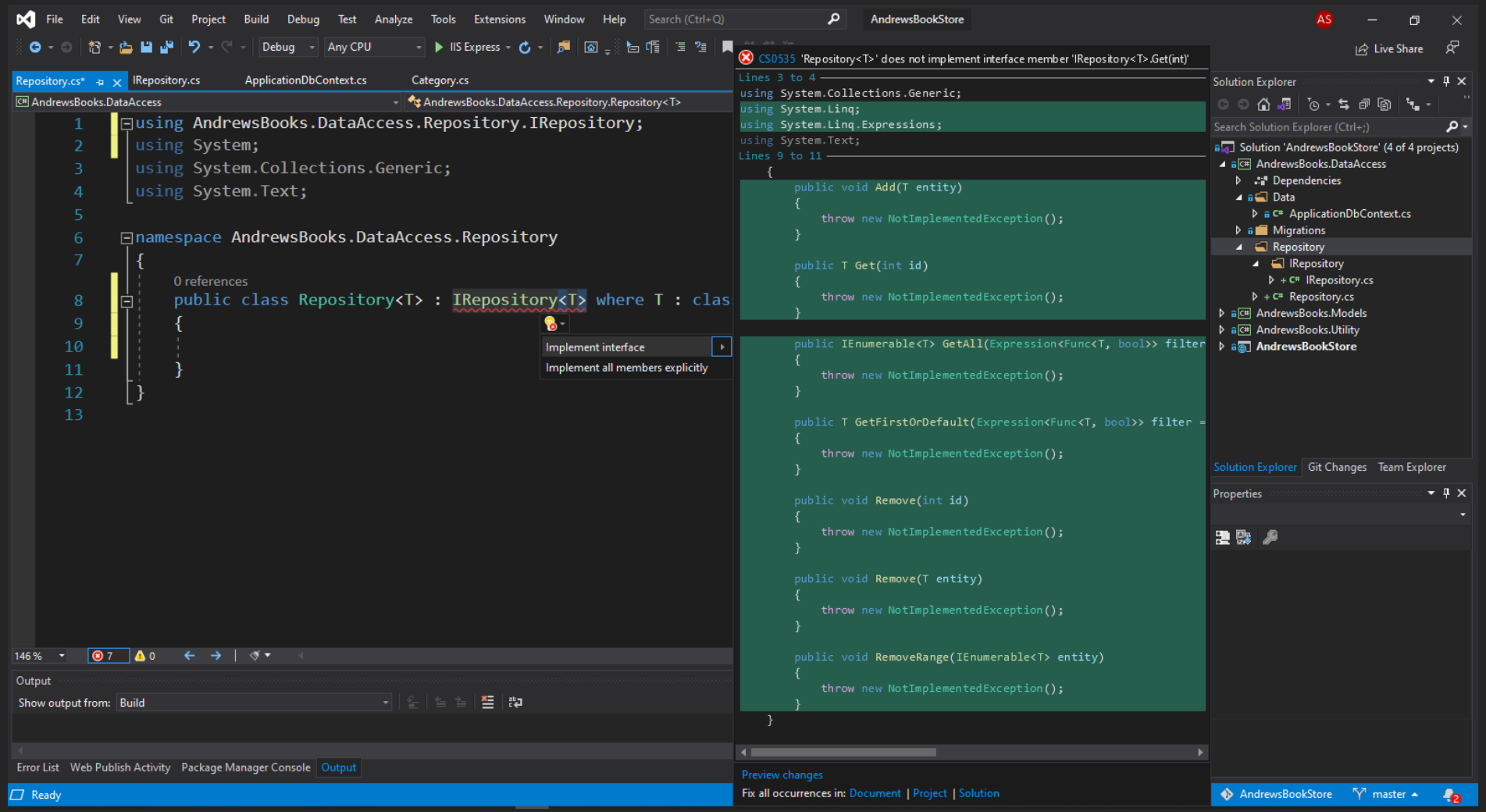
- Implement the class that implements the repository

- Include the using statement
- View the potential fixes and ‘implement interface’



- Implement the class that implements the repository

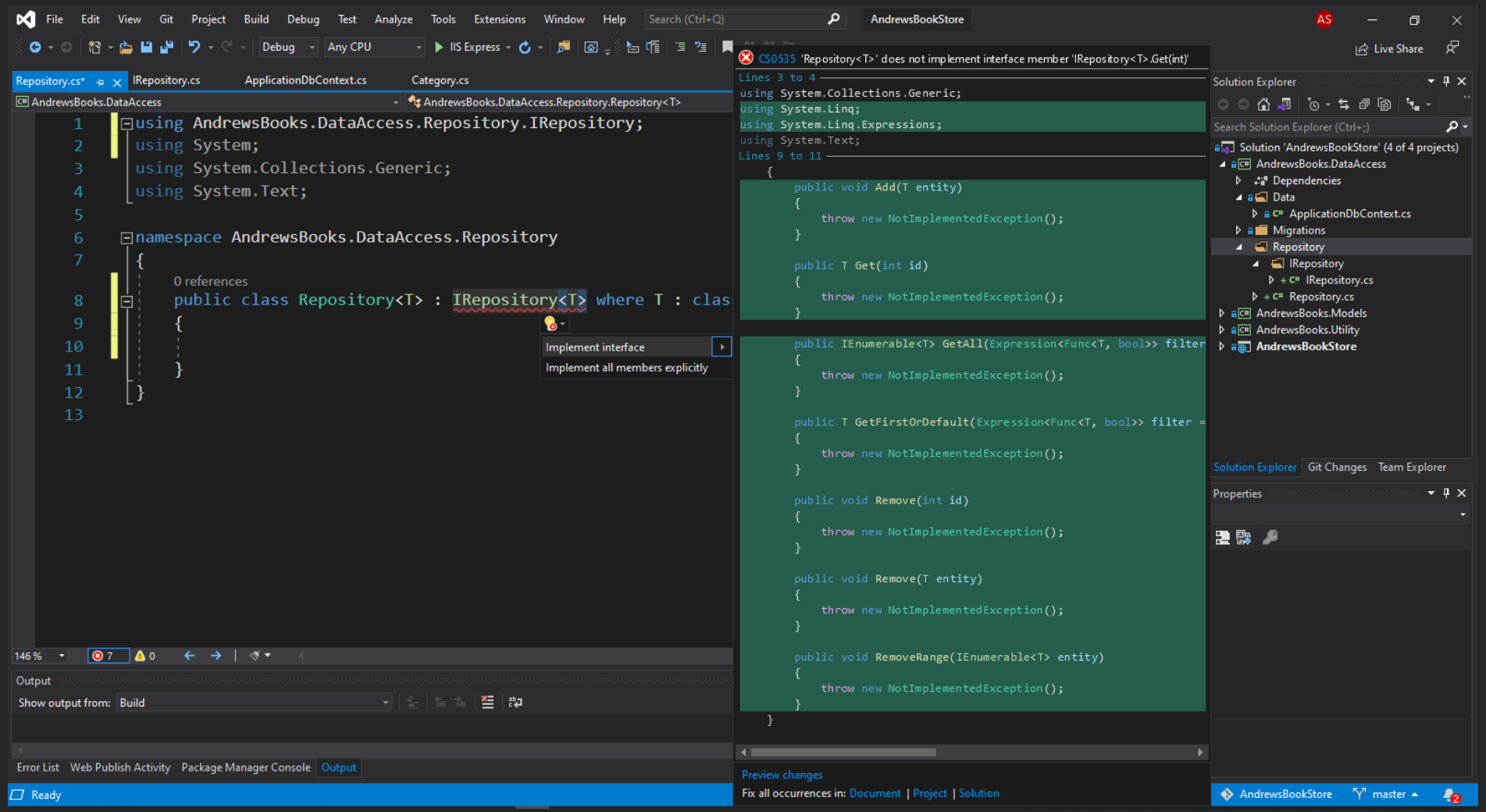
- Include the using statement
- View the potential fixes and ‘implement interface’



- Implement the class that implements the repository

- Include the using statement
- View the potential fixes and ‘implement interface’

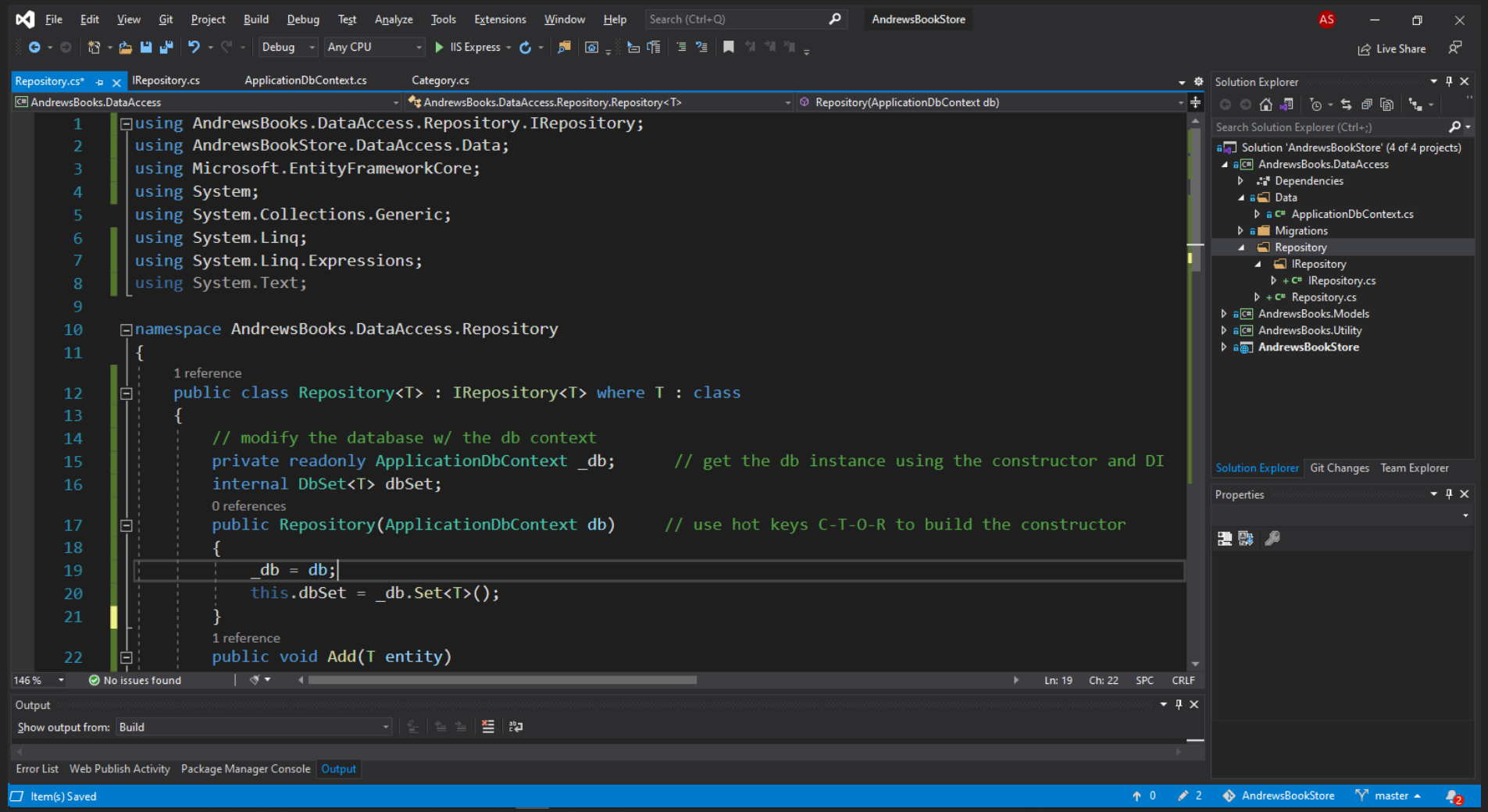
- Modify the code to create the constructors and dependency injection (DI)



- Implement the class that implements the repository

- Include the using statement
- View the potential fixes and ‘implement interface’

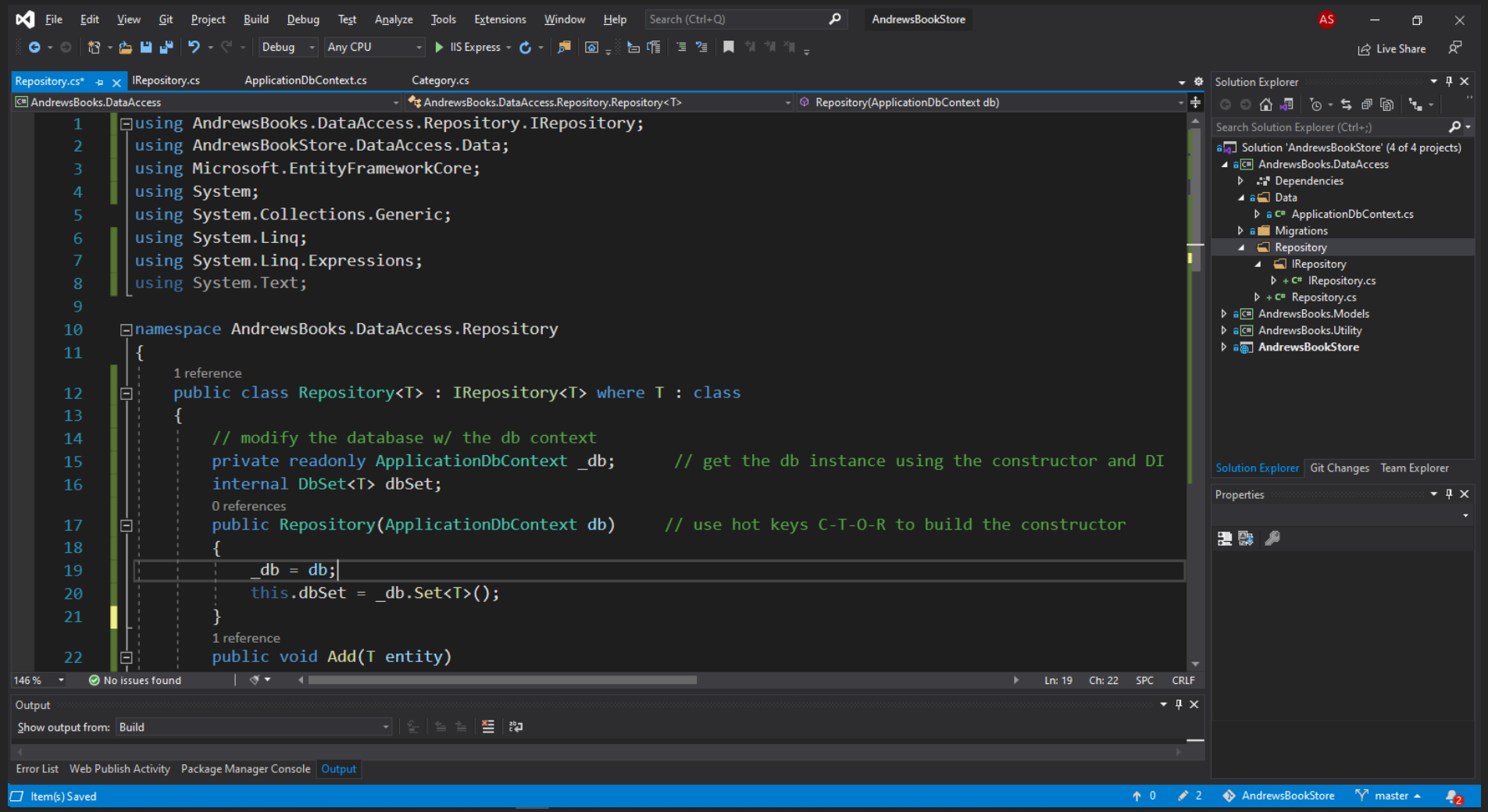
- Modify the code to create the constructors and dependency injection (DI)



- Implement the class that implements the repository

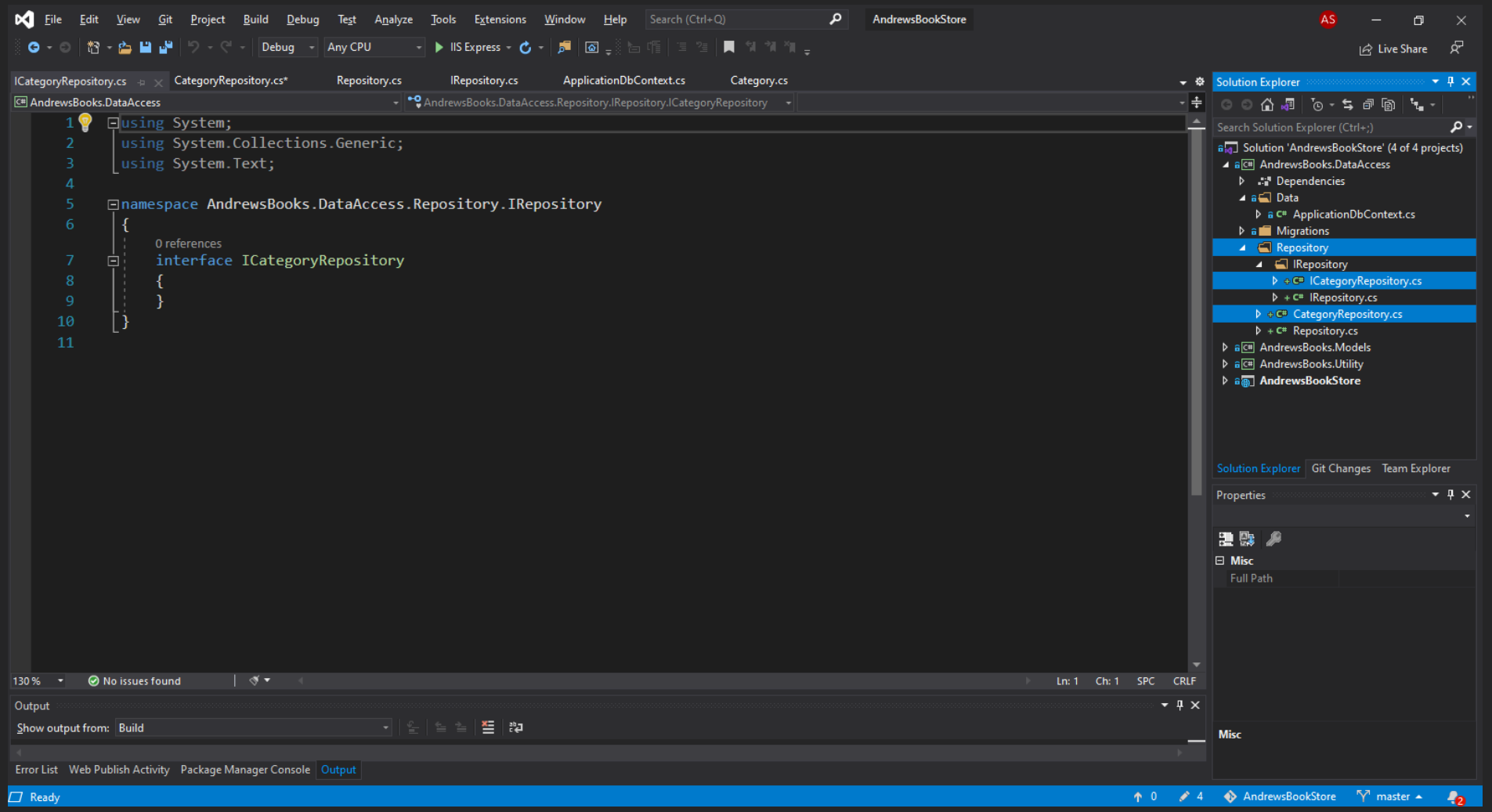
- Include the using statement
- View the potential fixes and ‘implement interface’

- Modify the code to create the constructors and dependency injection (DI)
  - *see* Assignment 2 folder

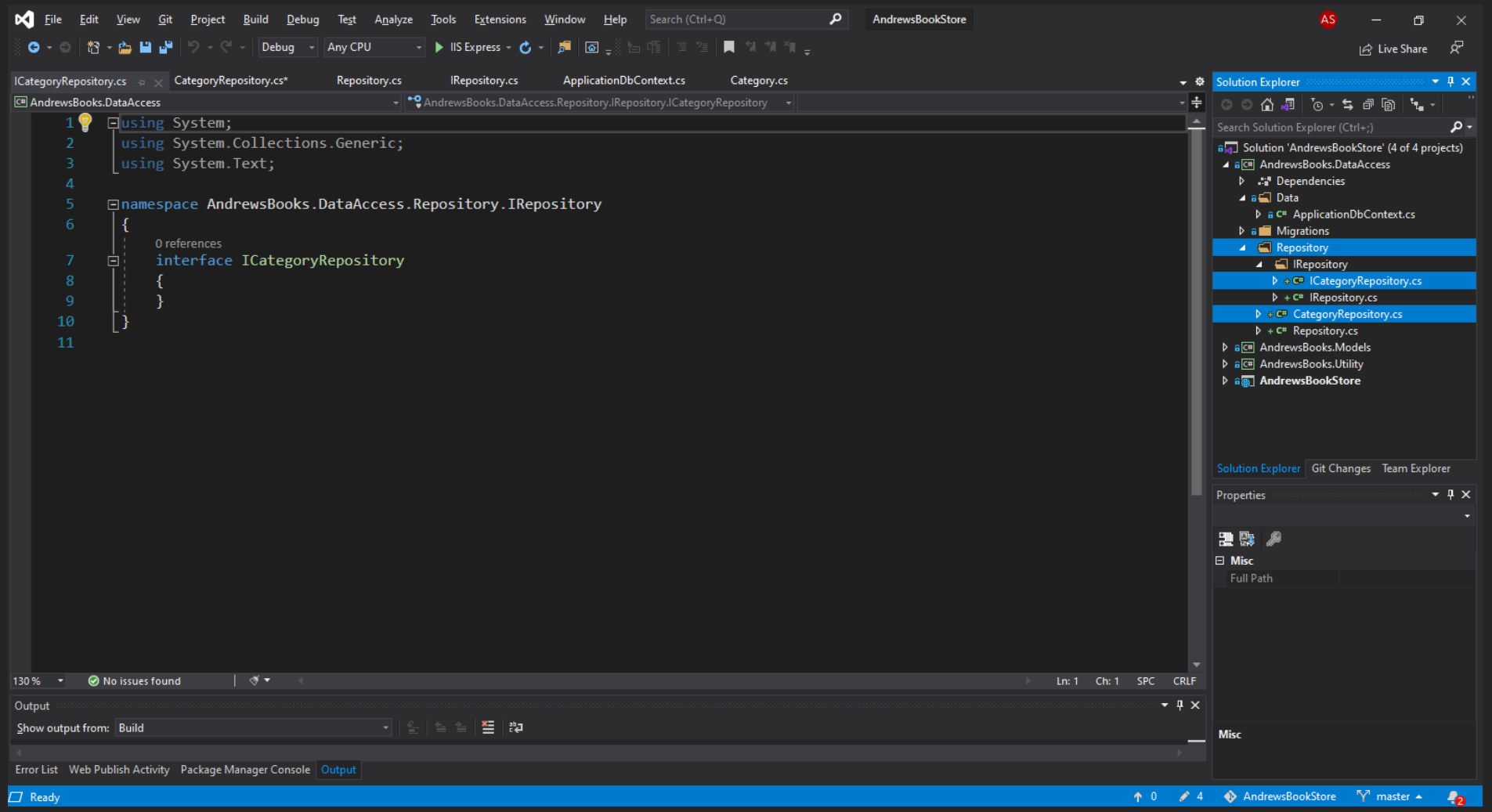


- Create individual repos for category and all potential models to be added in the future

- Create individual repos for category and all potential models to be added in the future

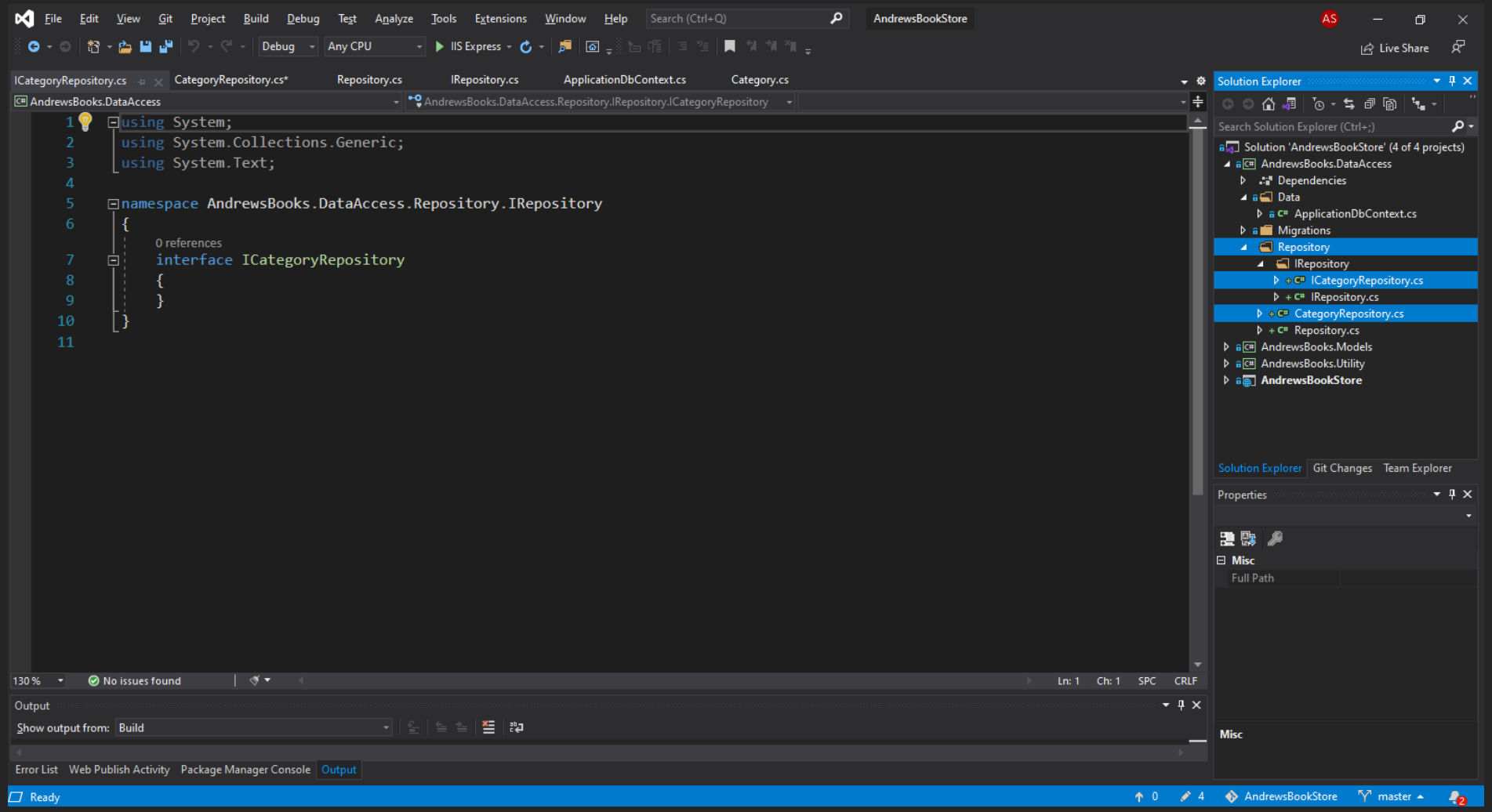


- Create individual repos for category and all potential models to be added in the future
  - CategoryRepository.cs
  - ICategoryRepository.cs



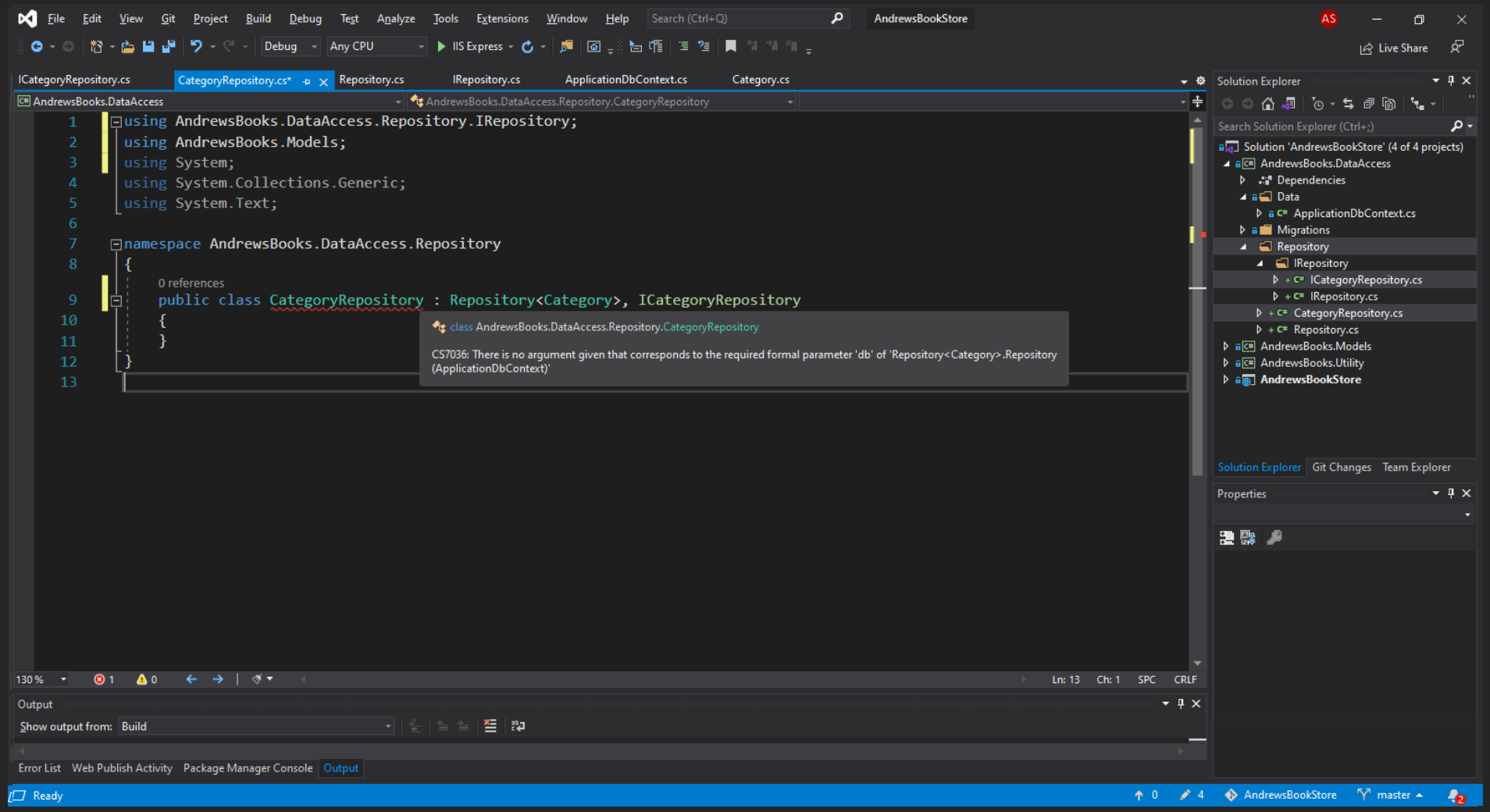
- Create individual repos for category and all potential models to be added in the future
  - CategoryRepository.cs
  - ICategoryRepository.cs

- Modify CategoryRepository (note the using statements and the message for formal parameters – review in Repository.cs the method pointing to the ApplicationDBContext)



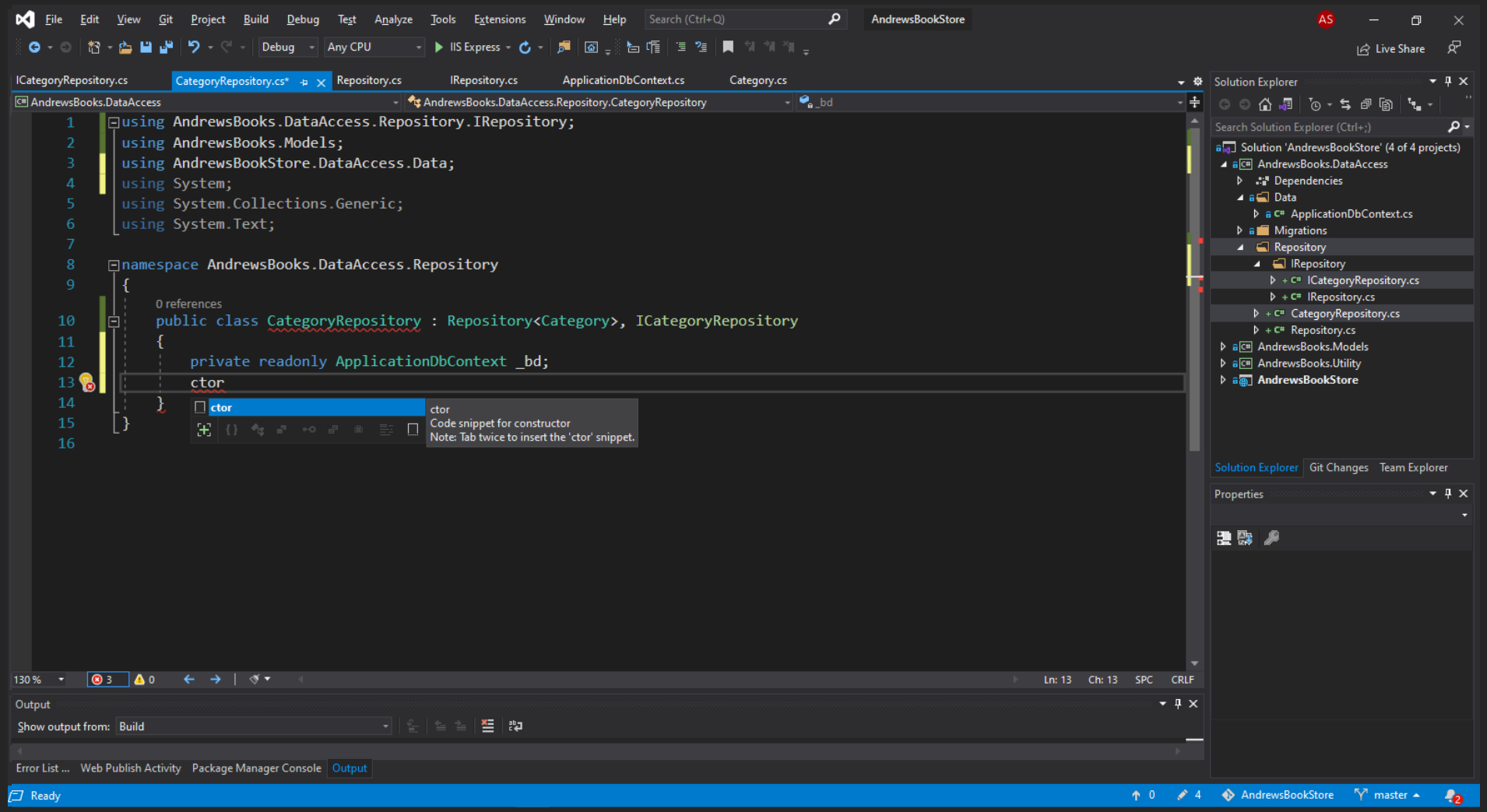
- Create individual repos for category and all potential models to be added in the future
  - CategoryRepository.cs
  - ICategoryRepository.cs

- Modify CategoryRepository (note the using statements and the message for formal parameters – review in Repository.cs the method pointing to the ApplicationDBContext)



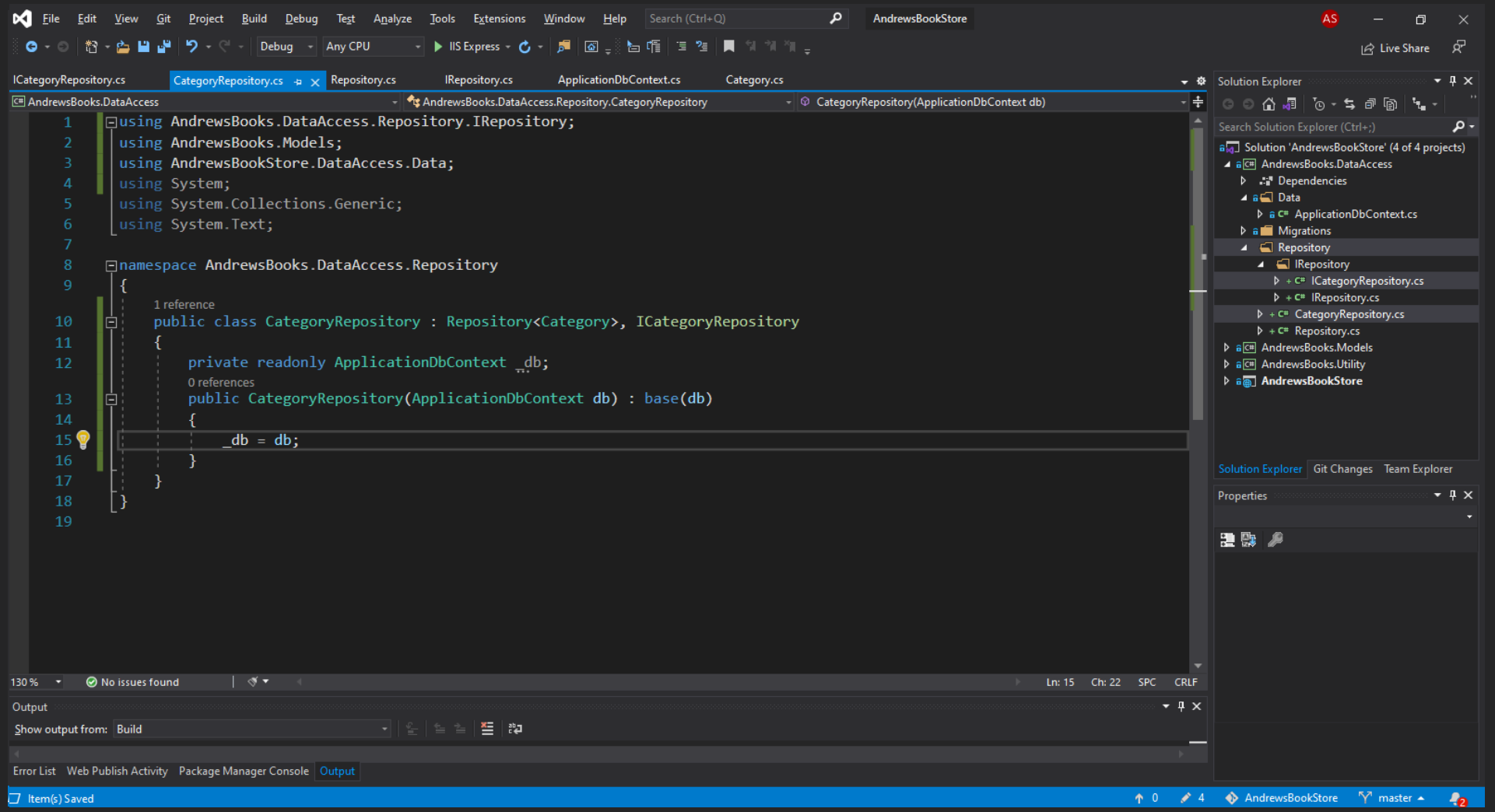
- Create individual repos for category and all potential models to be added in the future
  - CategoryRepository.cs
  - ICategoryRepository.cs

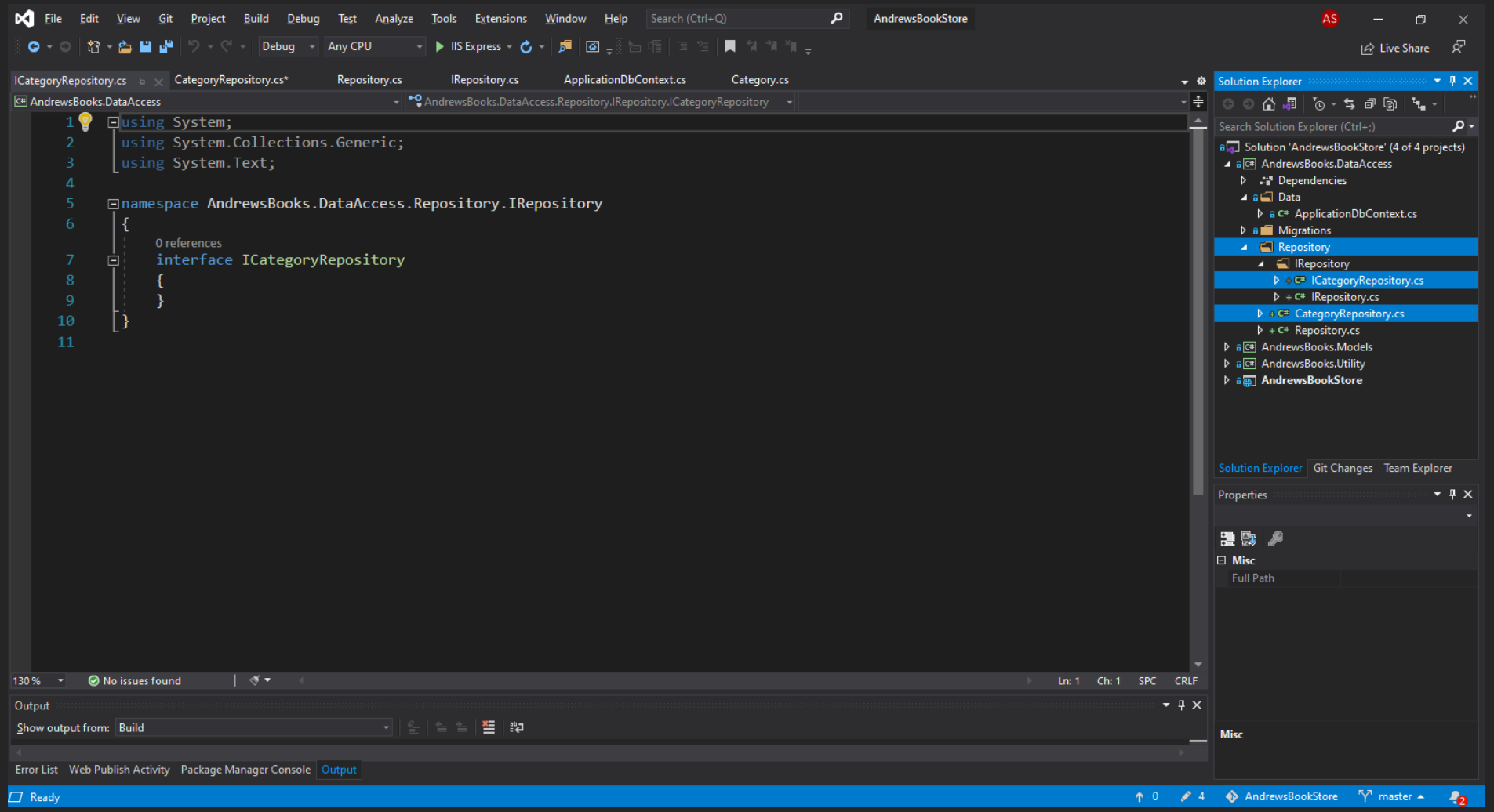
- Modify CategoryRepository (note the using statements and the message for formal parameters – review in Repository.cs the method pointing to the ApplicationDBContext)



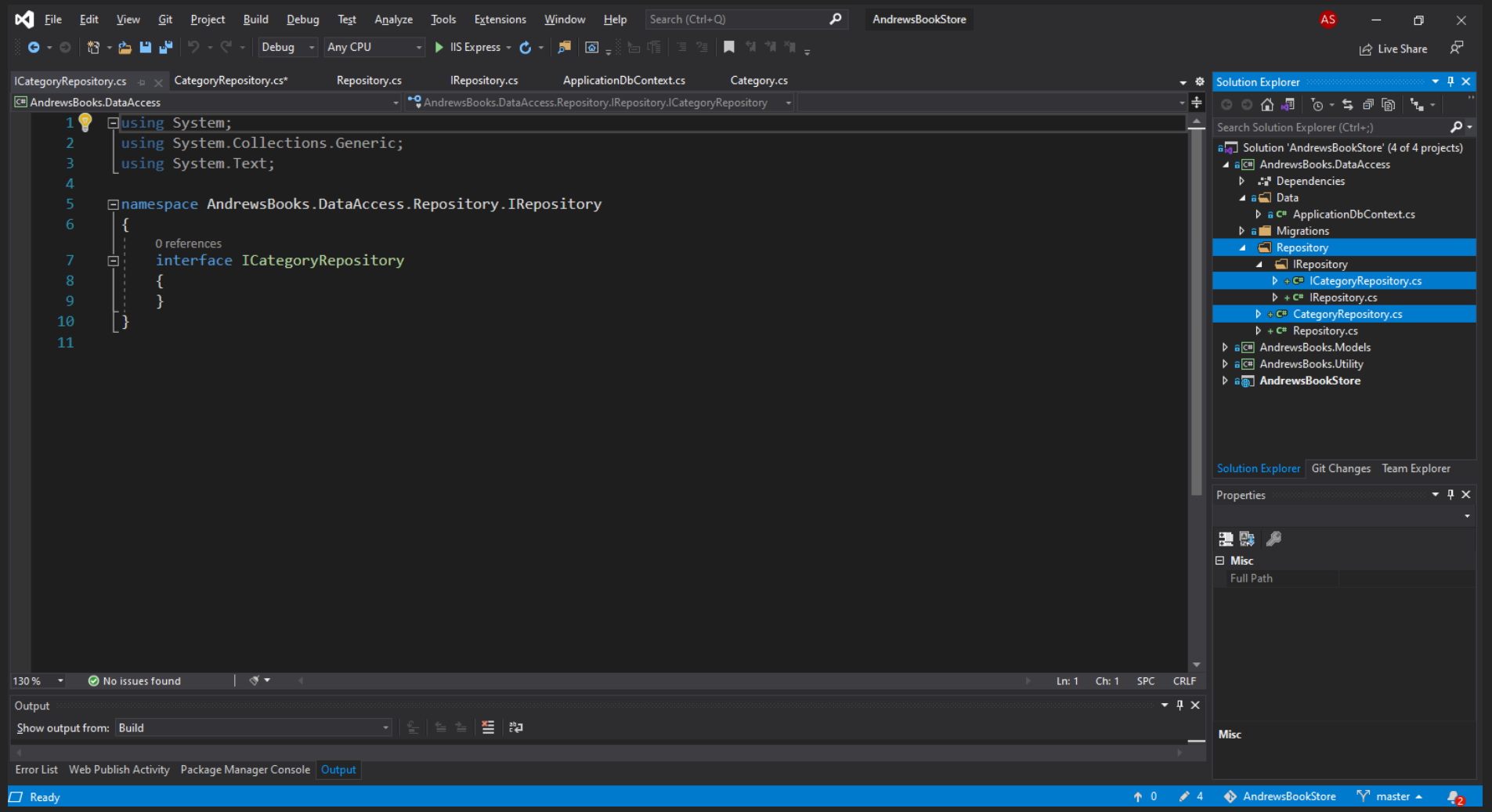
- Create individual repos for category and all potential models to be added in the future
  - CategoryRepository.cs
  - ICategoryRepository.cs

- Modify CategoryRepository (note the using statements and the message for formal parameters – review in Repository.cs the method pointing to the ApplicationDBContext)

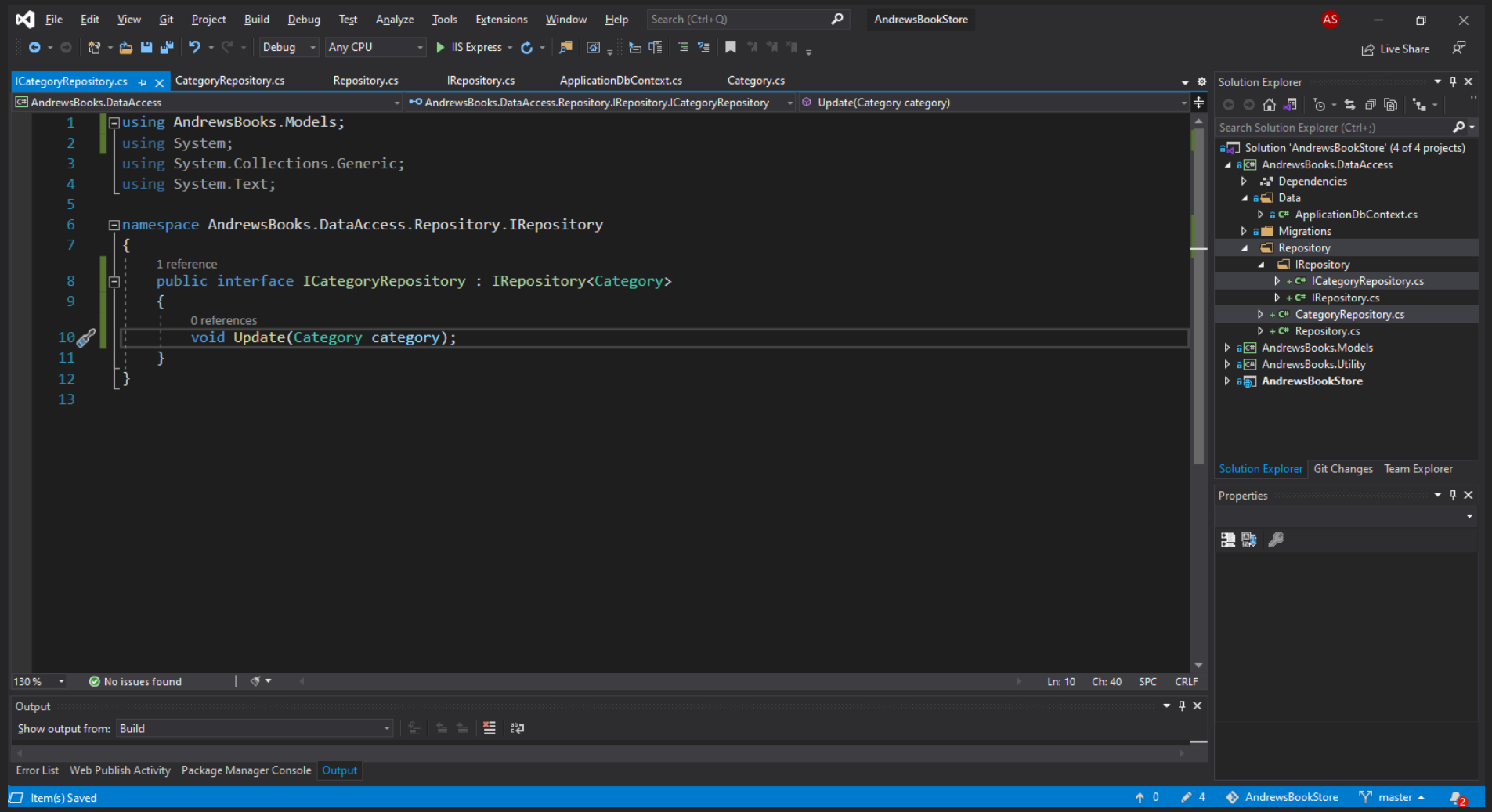



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
AndrewsBookStore
Debug
Any CPU
IIS Express
Live Share
ICategoryRepository.cs
CategoryRepository.cs*
Repository.cs
IRepository.cs
ApplicationDbContext.cs
Category.cs
AndrewsBooks.DataAccess
AndrewsBooks.DataAccess.Repository.IRepository.ICategoryRepository
using System;
using System.Collections.Generic;
using System.Text;
namespace AndrewsBooks.DataAccess.Repository.IRepository
{
0 references
interface ICategoryRepository
{
}
}
130 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF
Output
Show output from: Build
Error List
Web Publish Activity
Package Manager Console
Output
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AndrewsBookStore' (4 of 4 projects)
AndrewsBooks.DataAccess
Dependencies
Data
ApplicationDbContext.cs
Migrations
Repository
IRepository
ICategoryRepository.cs
IRepository.cs
CategoryRepository.cs
Repository.cs
AndrewsBooks.Models
AndrewsBooks.Utility
AndrewsBookStore
Solution Explorer
Git Changes
Team Explorer
Properties
Misc
Full Path
Misc
Ready
0
4
AndrewsBookStore
master

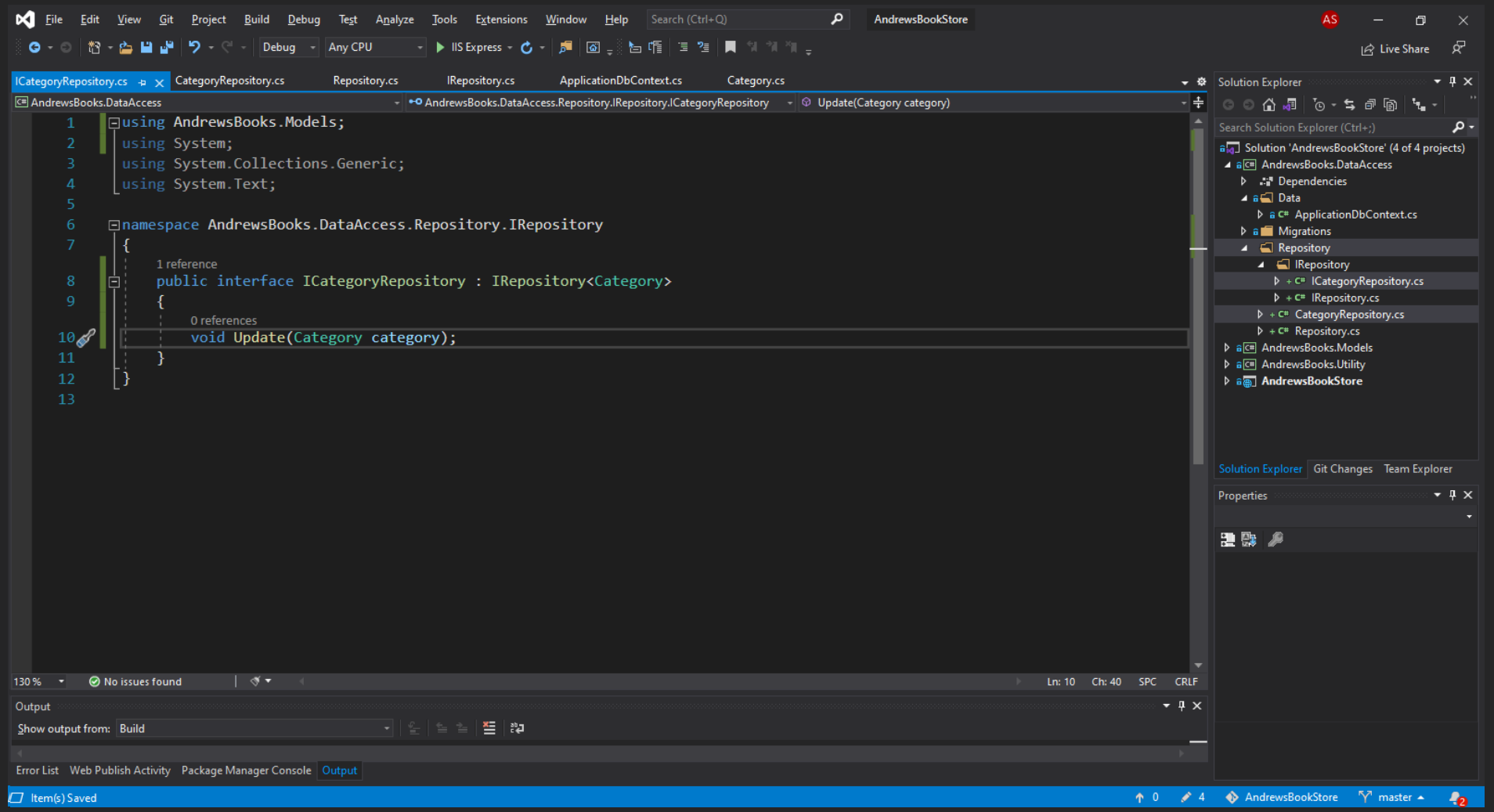
- Modify ICategoryRepository interface



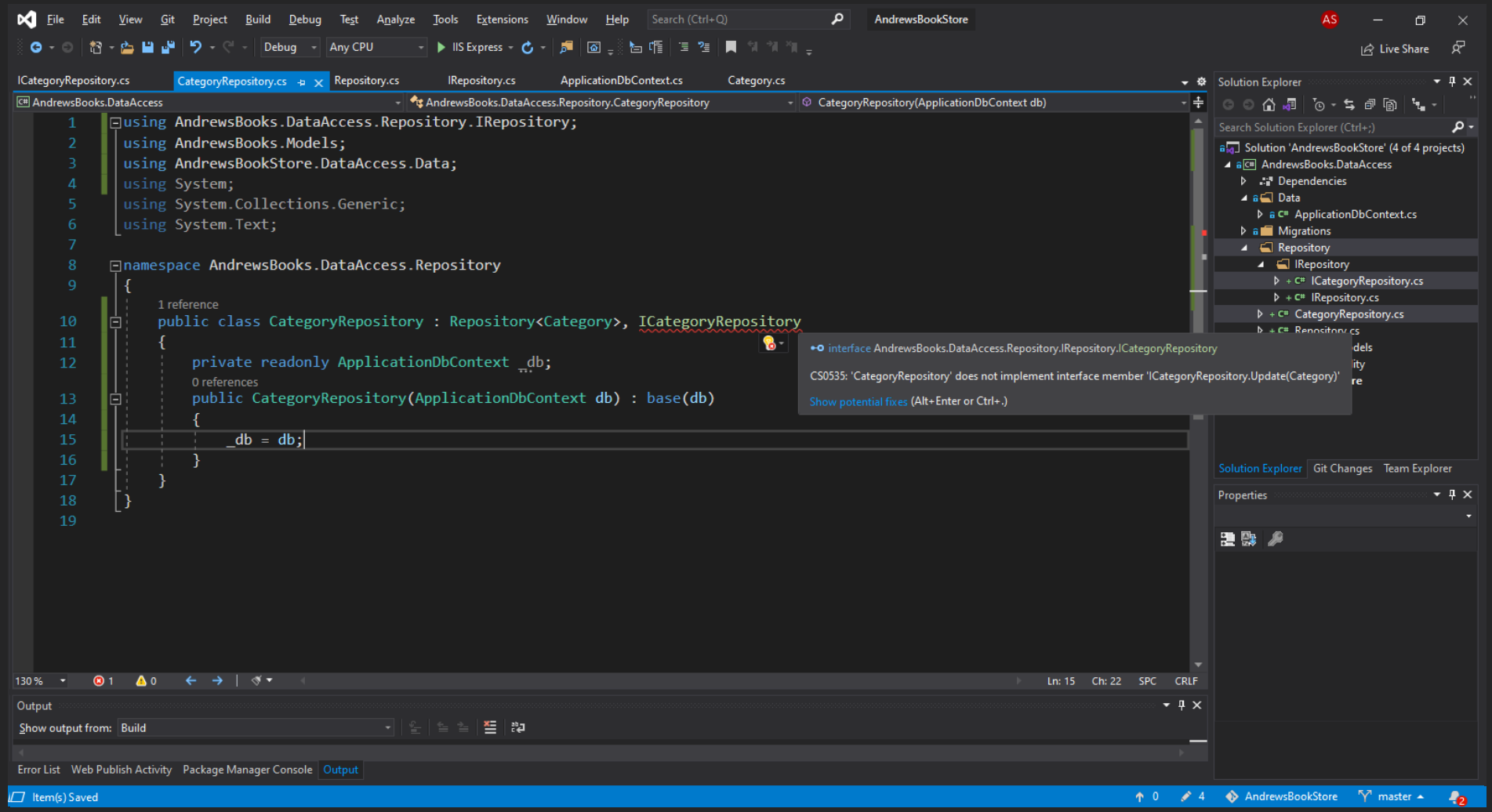
- Modify ICategoryRepository interface



- Modify ICategoryRepository interface
- Review and modify the error now in CategoryRepository.cs (implement the interface to update)

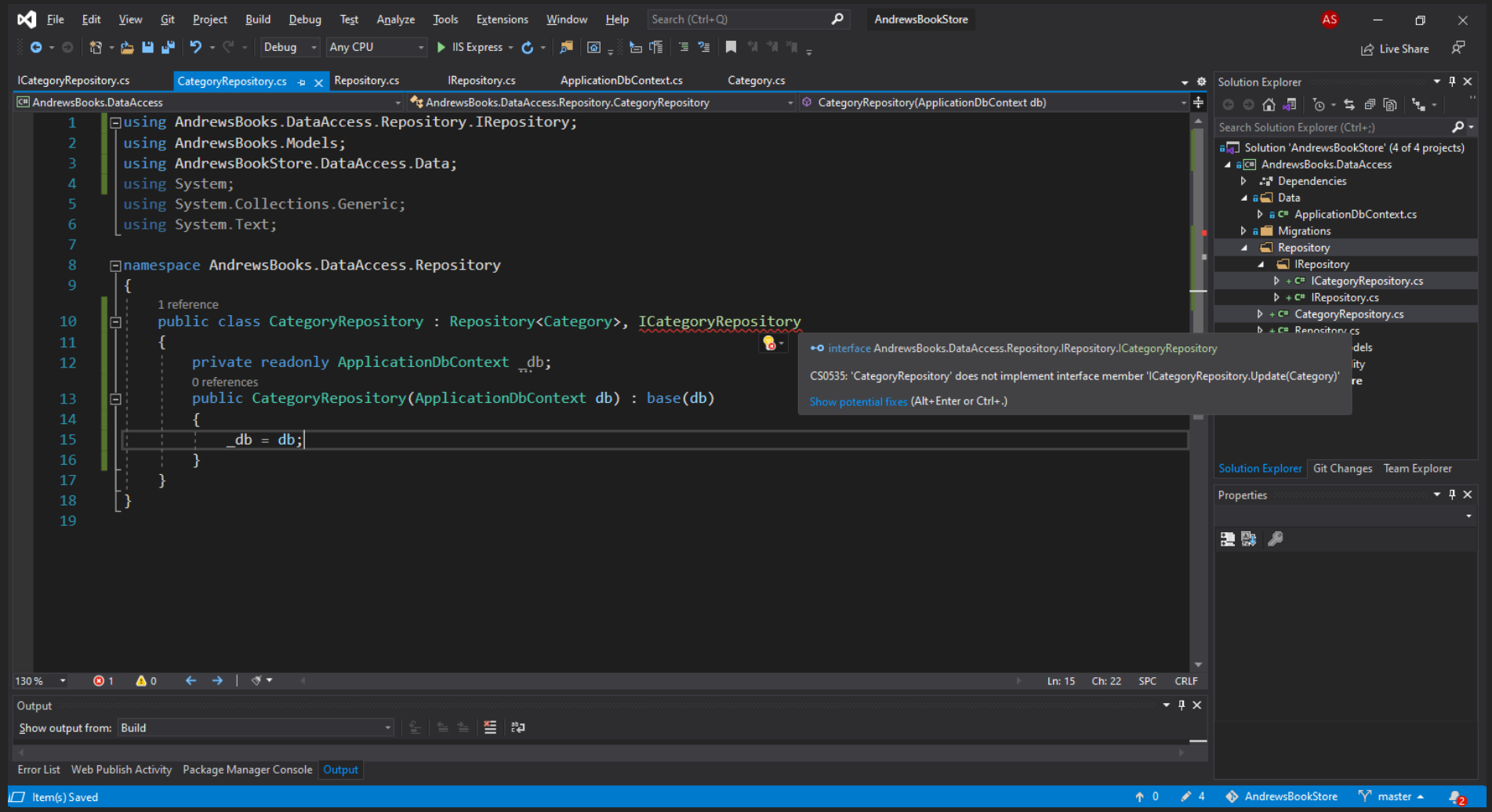


- Modify ICategoryRepository interface
- Review and modify the error now in CategoryRepository.cs (implement the interface to update)



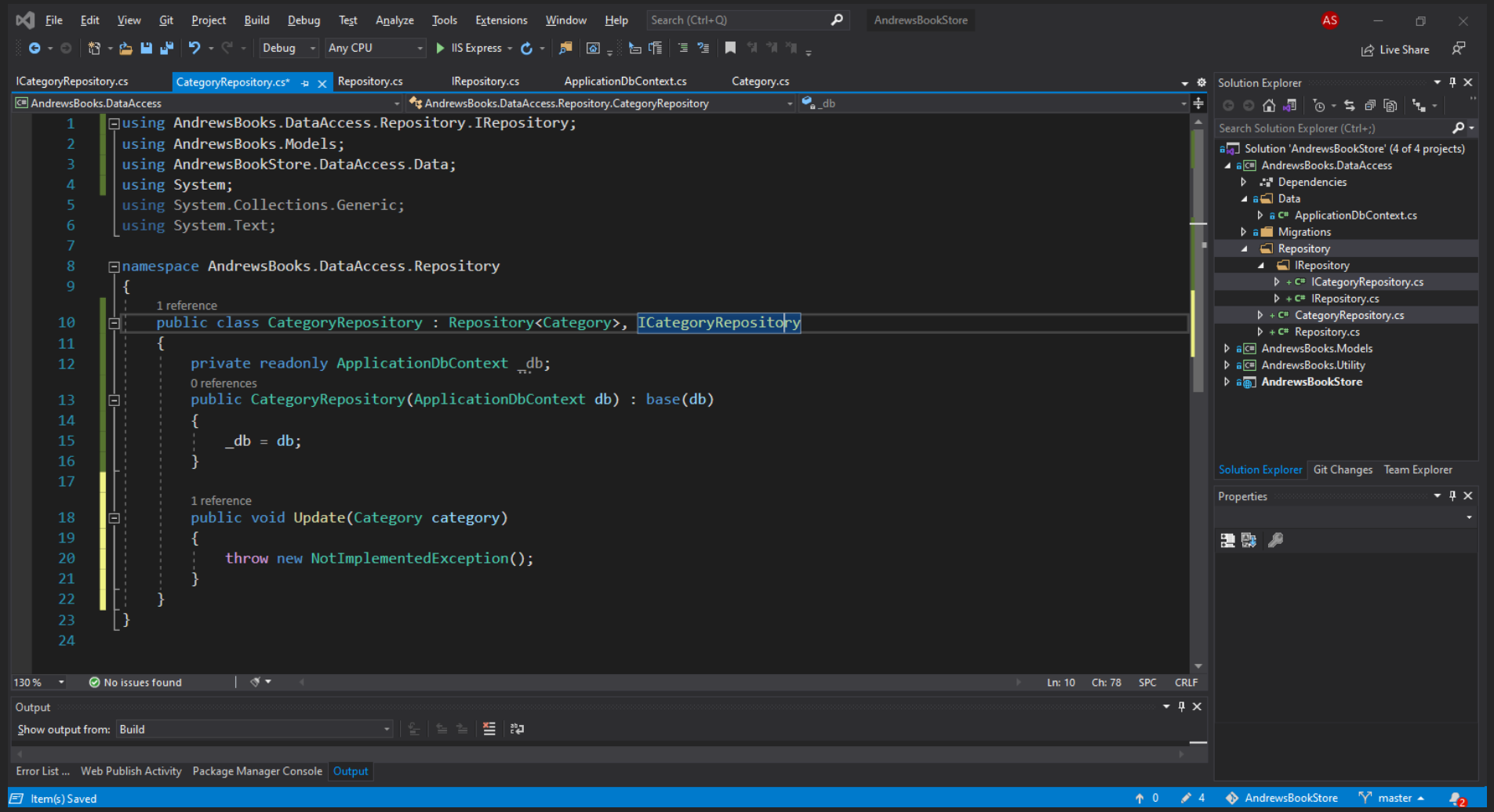
- Modify ICategoryRepository interface

- Review and modify the error now in CategoryRepository.cs (implement the interface to update)
- Complete the remaining modifications (note the comments)



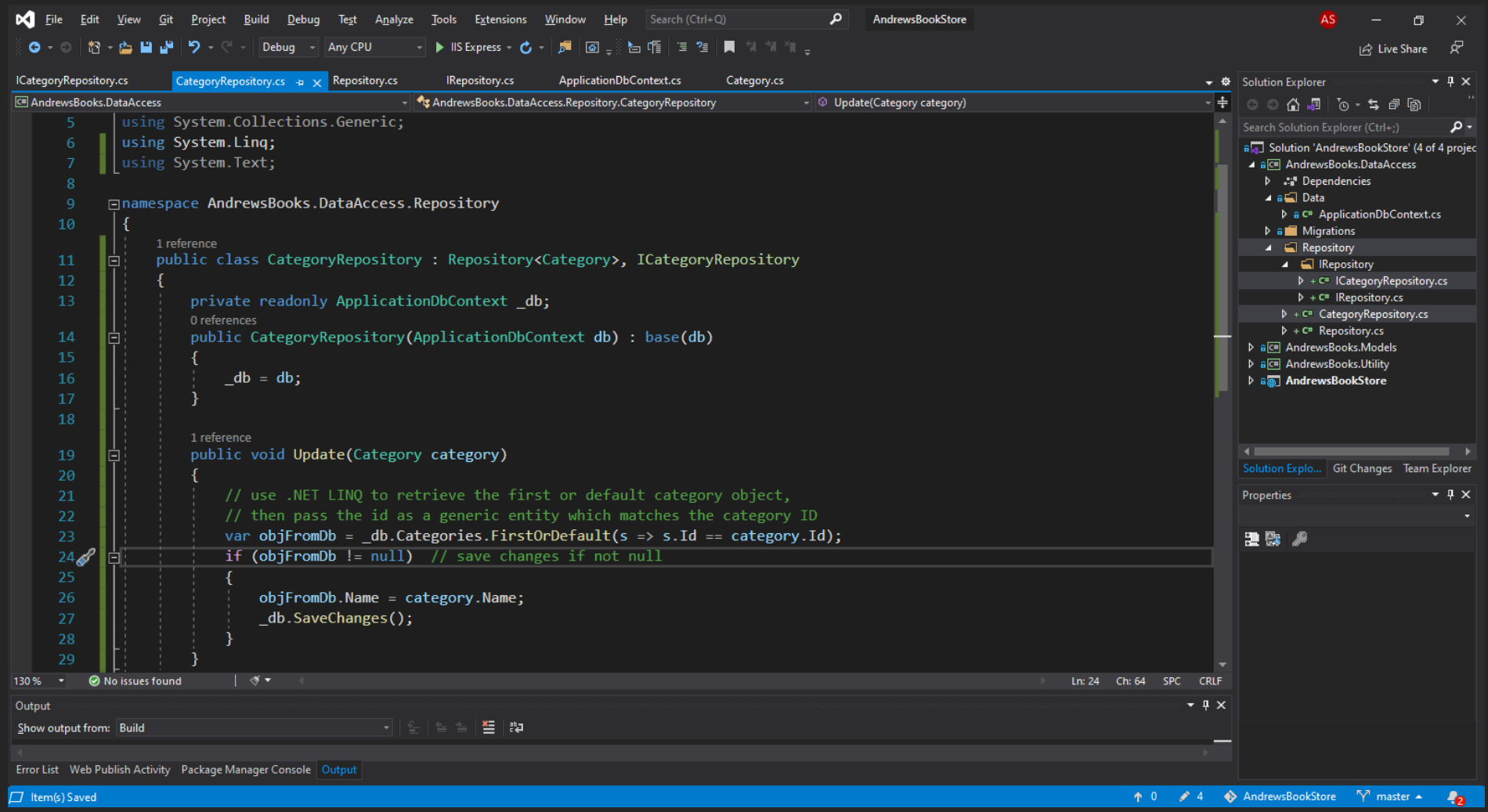
- Modify ICategoryRepository interface

- Review and modify the error now in CategoryRepository.cs (implement the interface to update)
- Complete the remaining modifications (note the comments)



- Modify ICategoryRepository interface

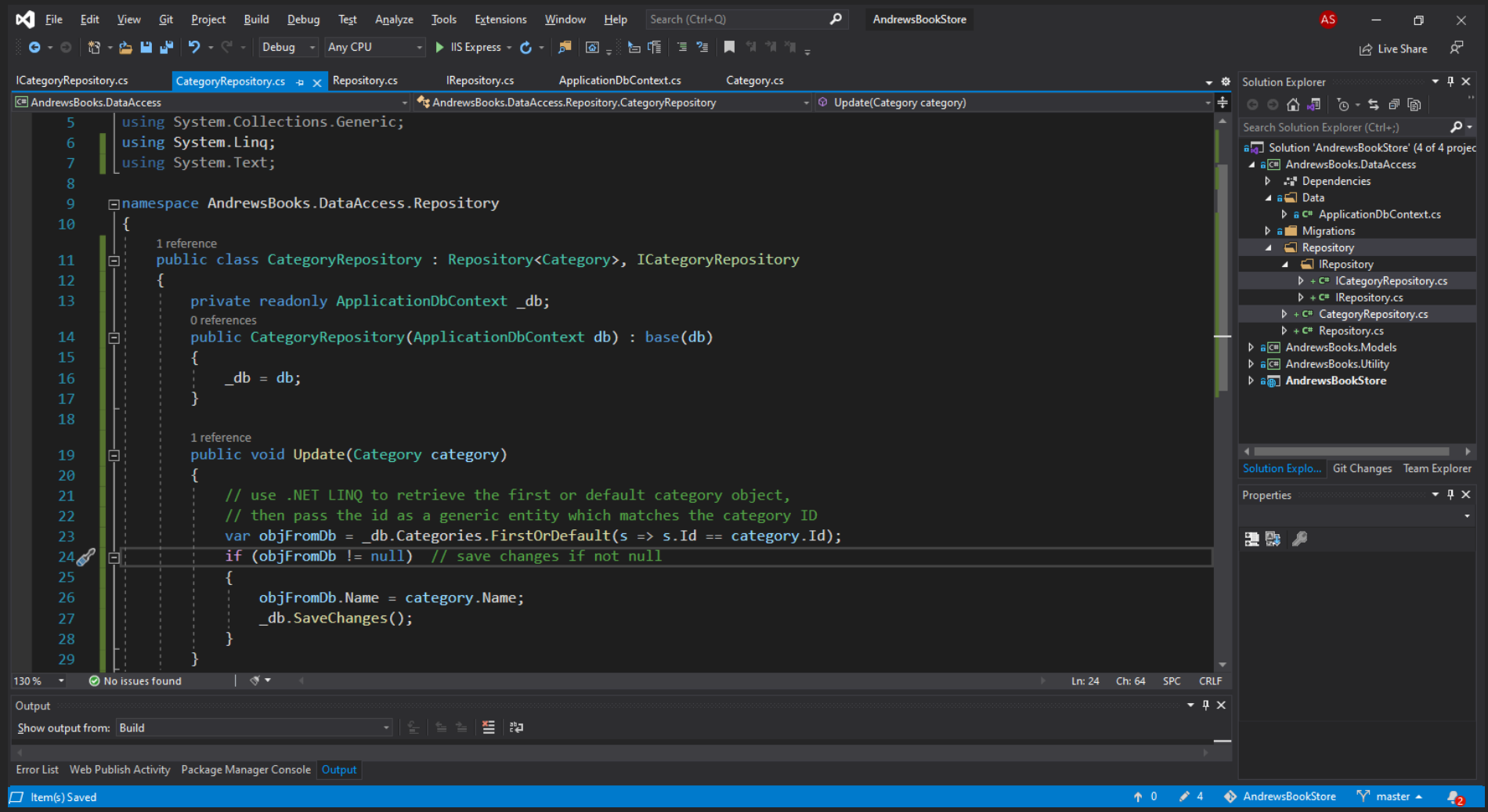
- Review and modify the error now in CategoryRepository.cs (implement the interface to update)
- Complete the remaining modifications (note the comments)

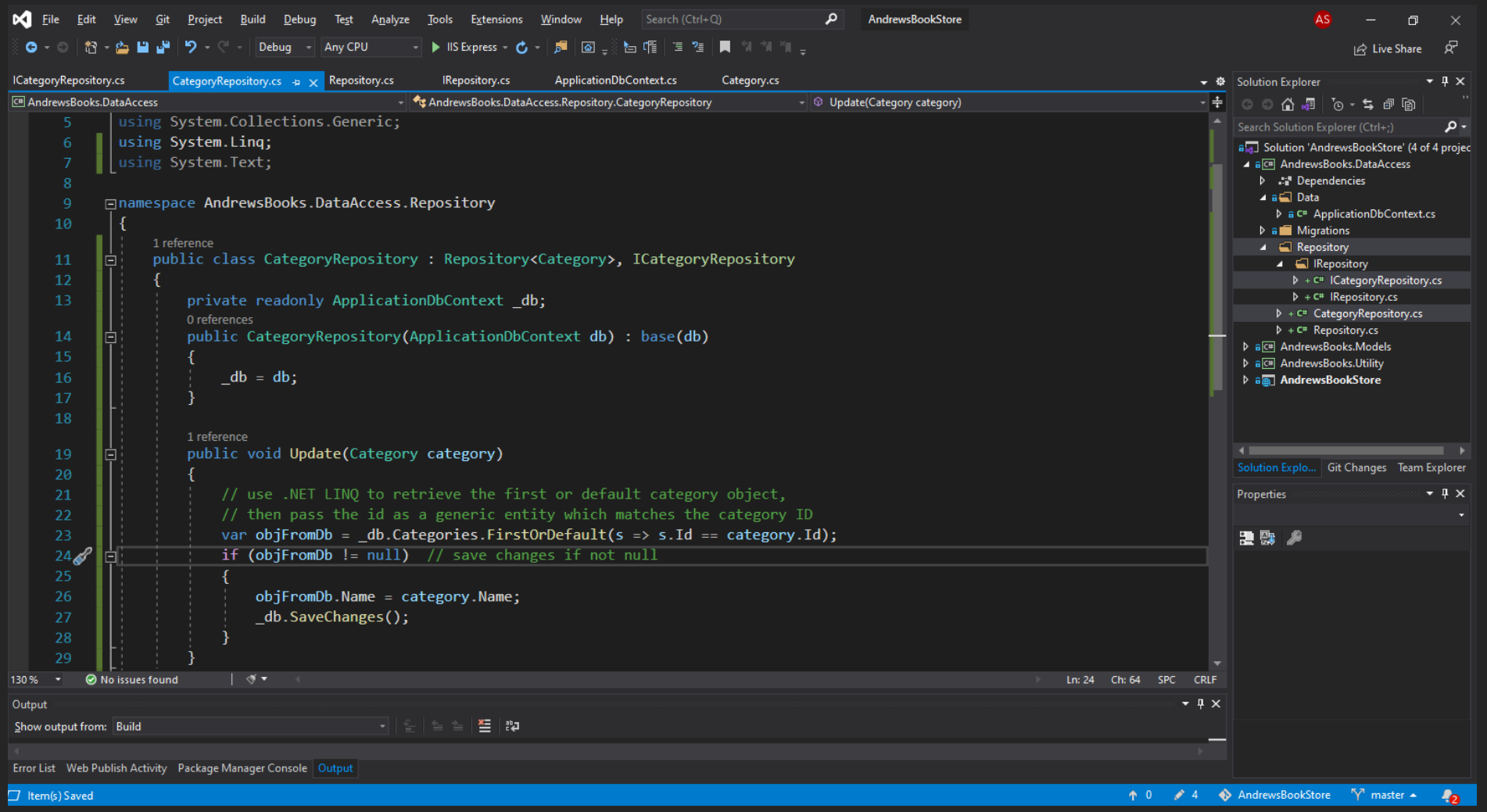


- Modify ICategoryRepository interface

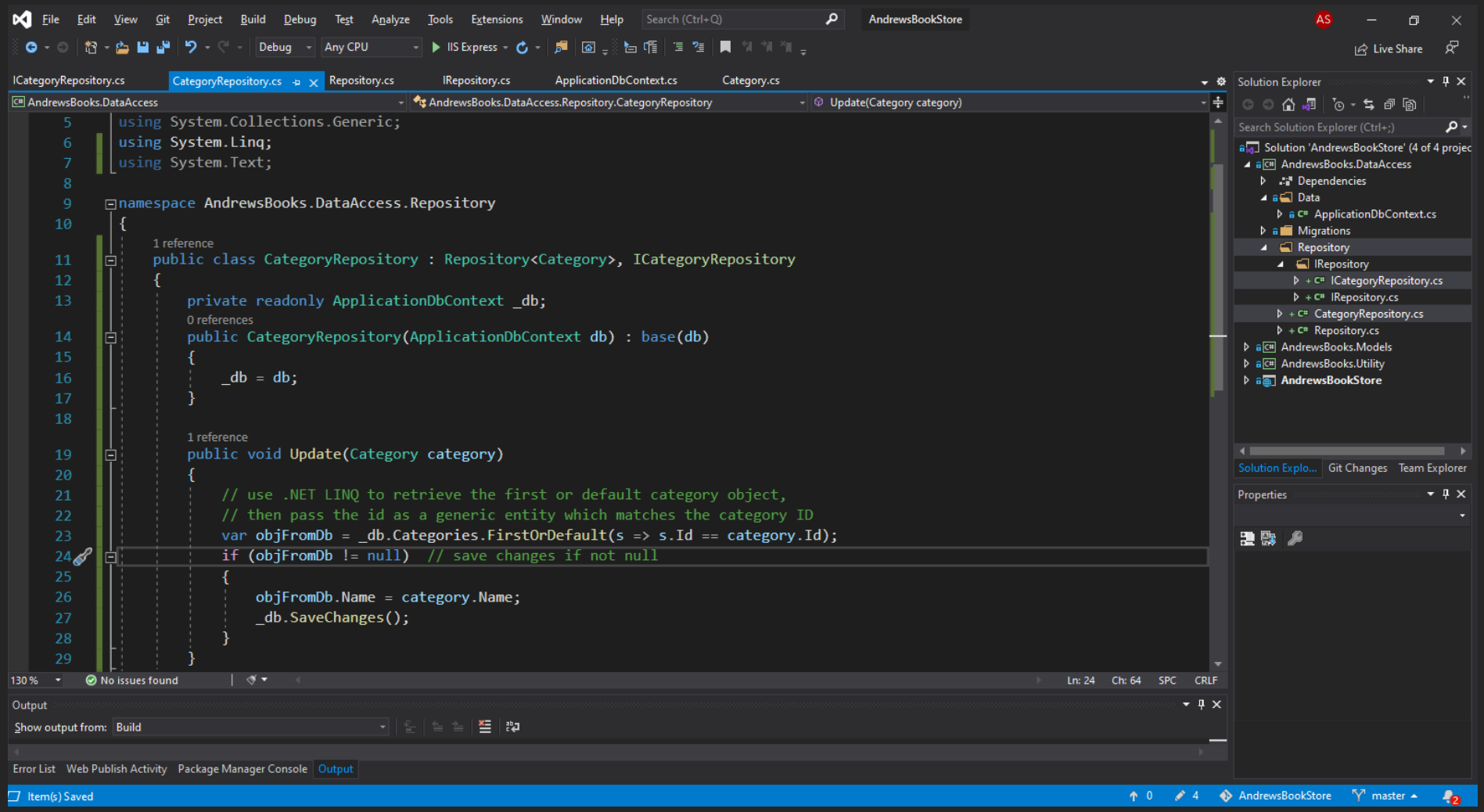
- Review and modify the error now in CategoryRepository.cs (implement the interface to update)
- Complete the remaining modifications (note the comments)

- Build, fix any error and push commits to GitHub

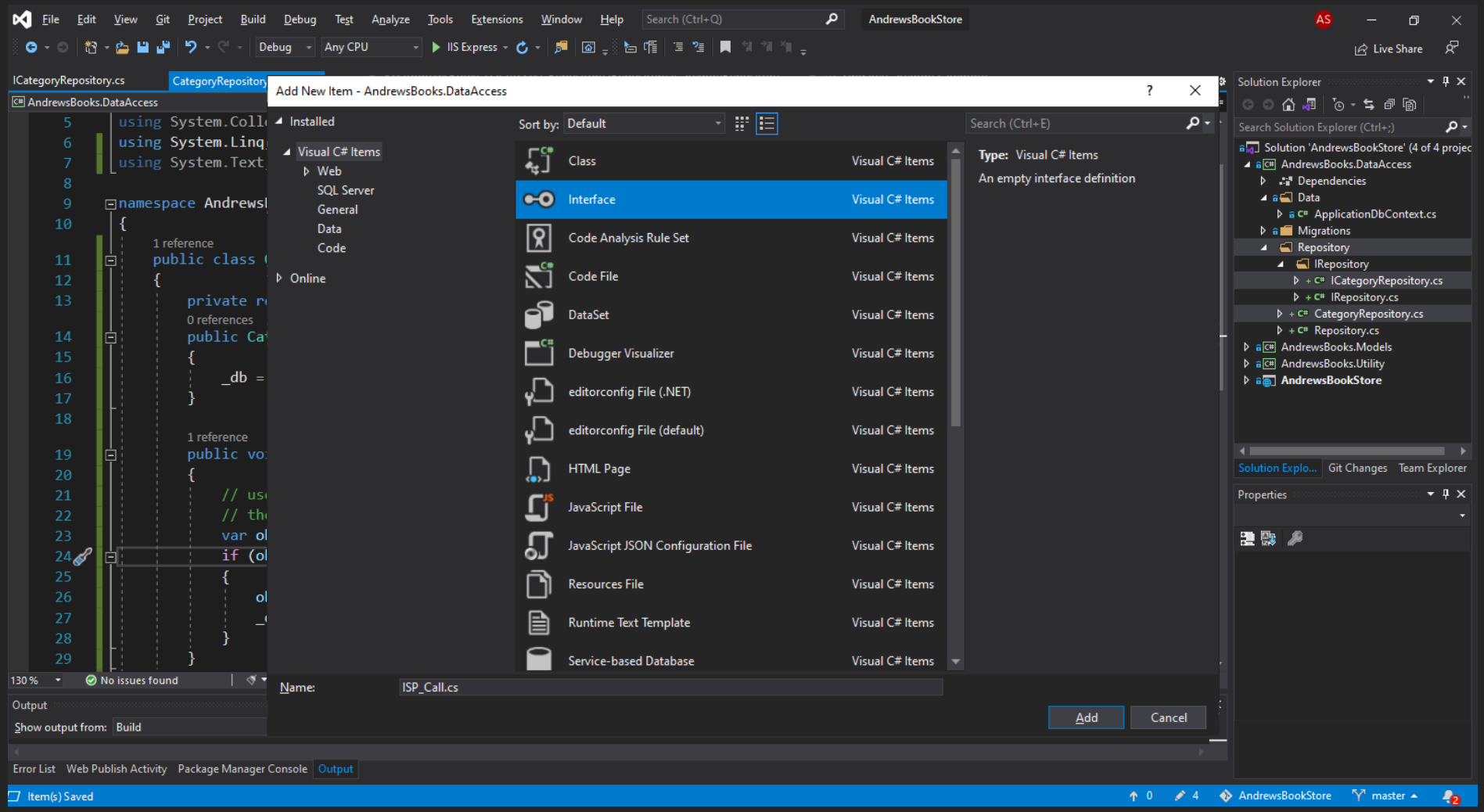



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
AndrewsBookStore
Debug
Any CPU
IIS Express
Live Share
ICategoryRepository.cs
CategoryRepository.cs
Repository.cs
IRepository.cs
ApplicationDbContext.cs
Category.cs
AndrewsBooks.DataAccess
AndrewsBooks.DataAccess.Repository.CategoryRepository
Update(Category category)
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace AndrewsBooks.DataAccess.Repository
{
1 reference
public class CategoryRepository : Repository<Category>, ICategoryRepository
{
private readonly ApplicationDbContext _db;
0 references
public CategoryRepository(ApplicationDbContext db) : base(db)
{
_db = db;
}
1 reference
public void Update(Category category)
{
// use .NET LINQ to retrieve the first or default category object,
// then pass the id as a generic entity which matches the category ID
var objFromDb = _db.Categories.FirstOrDefault(s => s.Id == category.Id);
if (objFromDb != null) // save changes if not null
{
objFromDb.Name = category.Name;
_db.SaveChanges();
}
}
130 %
No issues found
Ln: 24
Ch: 64
SPC
CRLF
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AndrewsBookStore' (4 of 4 projec
AndrewsBooks.DataAccess
Dependencies
Data
ApplicationDbContext.cs
Migrations
Repository
IRepository
ICategoryRepository.cs
IRepository.cs
CategoryRepository.cs
Repository.cs
AndrewsBooks.Models
AndrewsBooks.Utility
AndrewsBookStore
Solution Explo...
Git Changes
Team Explorer
Properties
Output
Show output from: Build
Error List
Web Publish Activity
Package Manager Console
Output
Item(s) Saved
0
4
AndrewsBookStore
master

- Implement a stored procedure repository and map multiple repositories in a Unit of Work

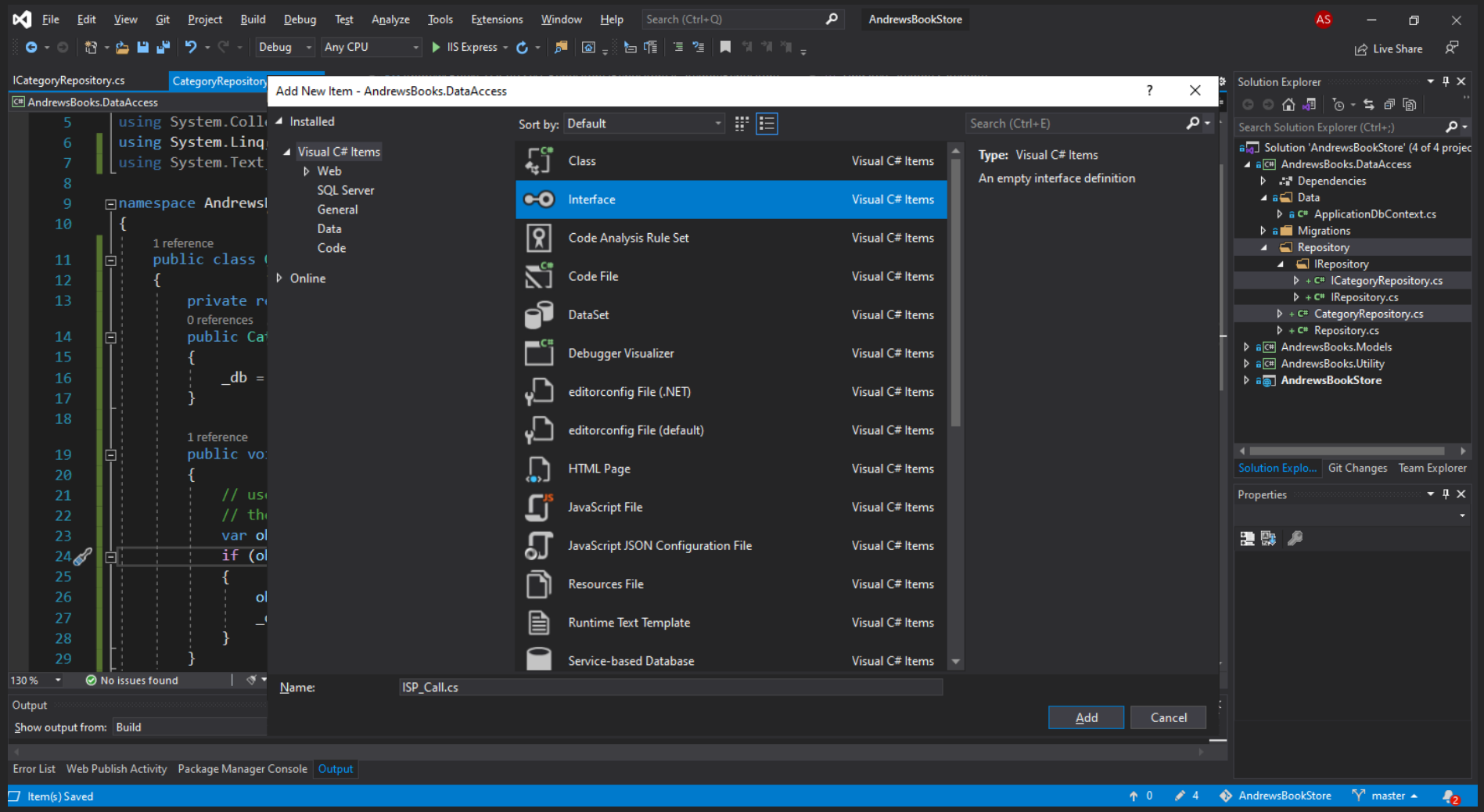


- Implement a stored procedure repository and map multiple repositories in a Unit of Work



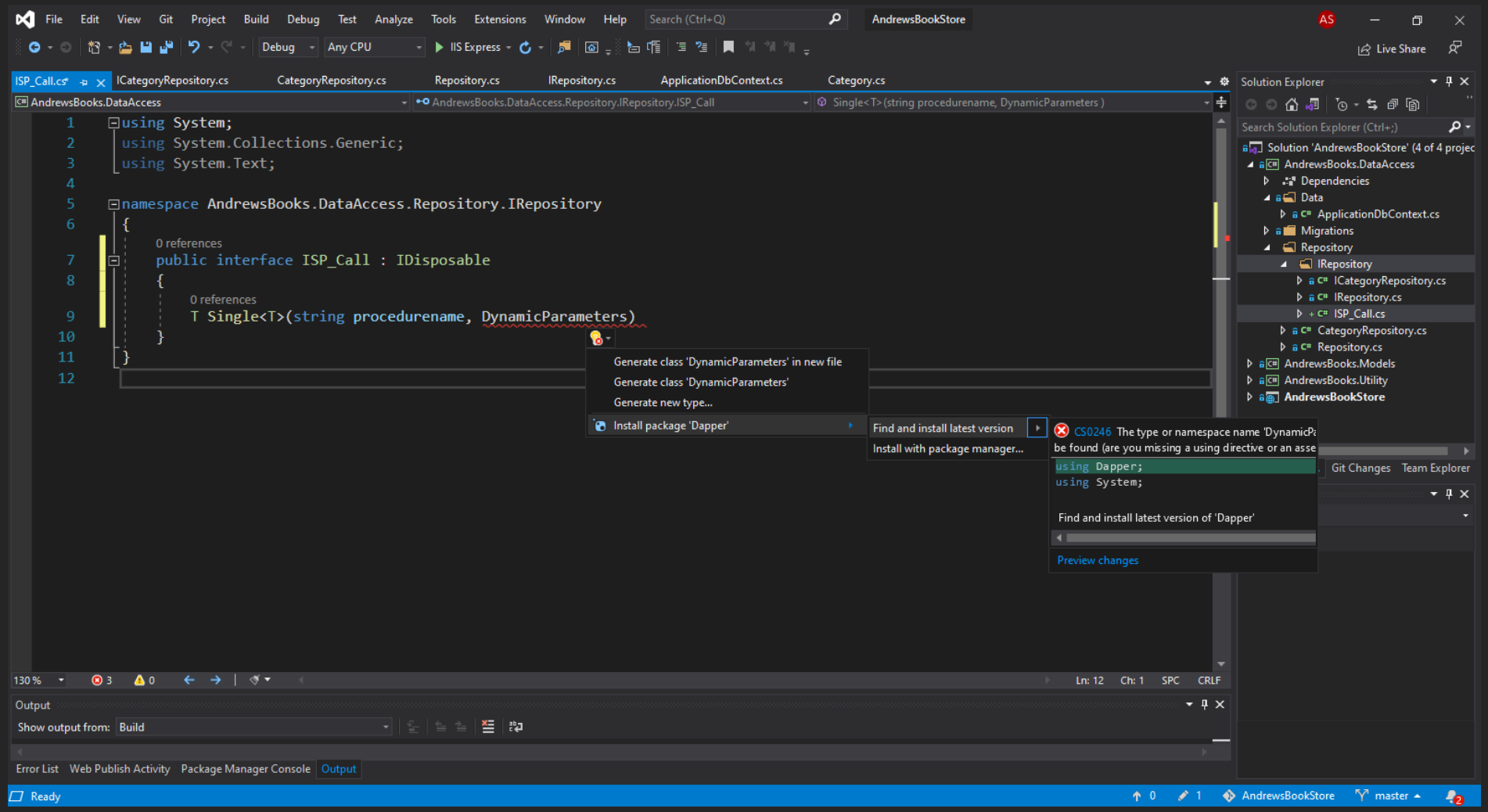
- Implement a stored procedure repository and map multiple repositories in a Unit of Work

- Add a new interface in the IRepository folder - ISP_Call.cs – that extends IDisposable



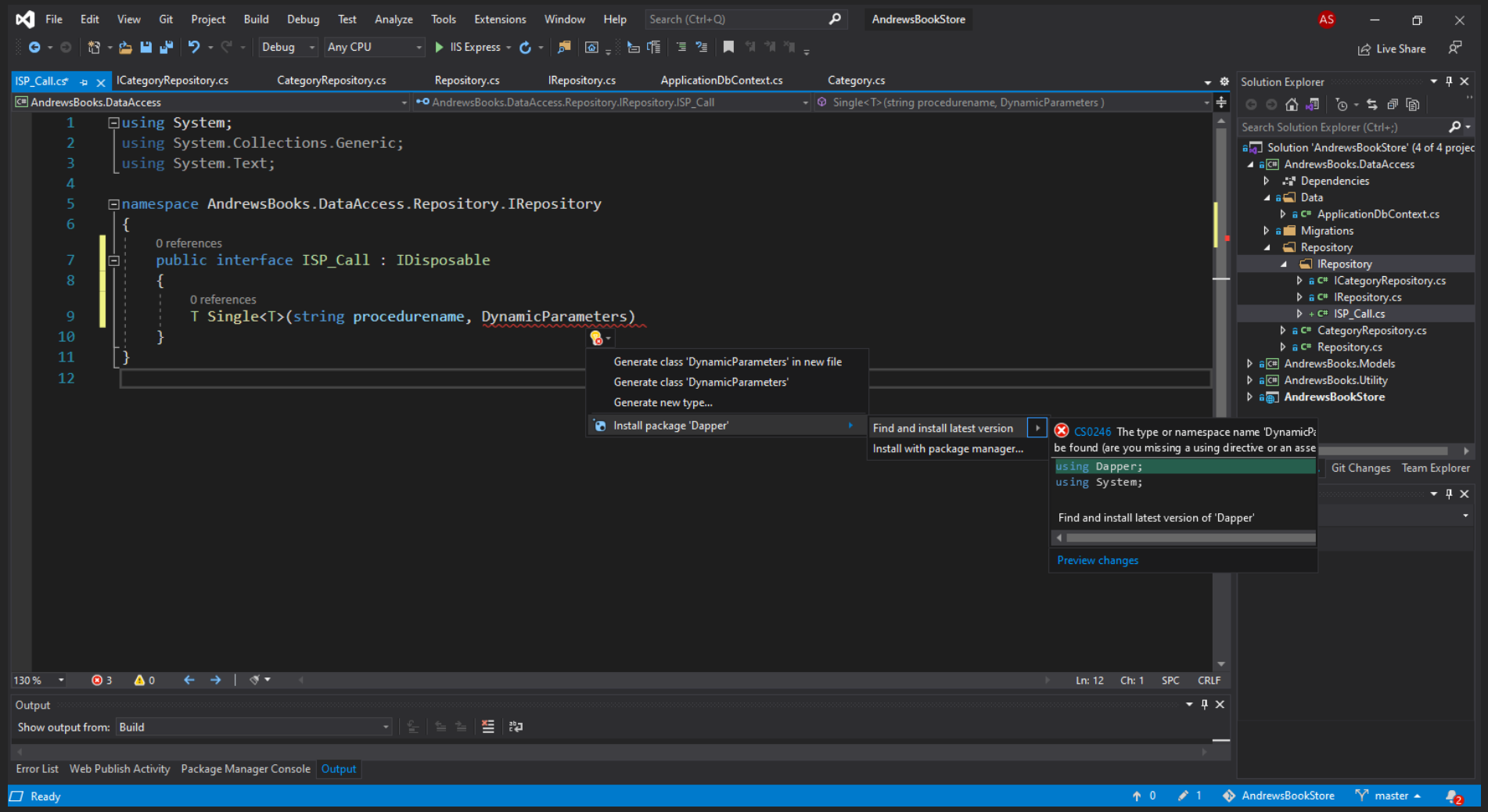
- Implement a stored procedure repository and map multiple repositories in a Unit of Work

- Add a new interface in the IRepository folder - ISP_Call.cs – that extends IDisposable



- Implement a stored procedure repository and map multiple repositories in a Unit of Work

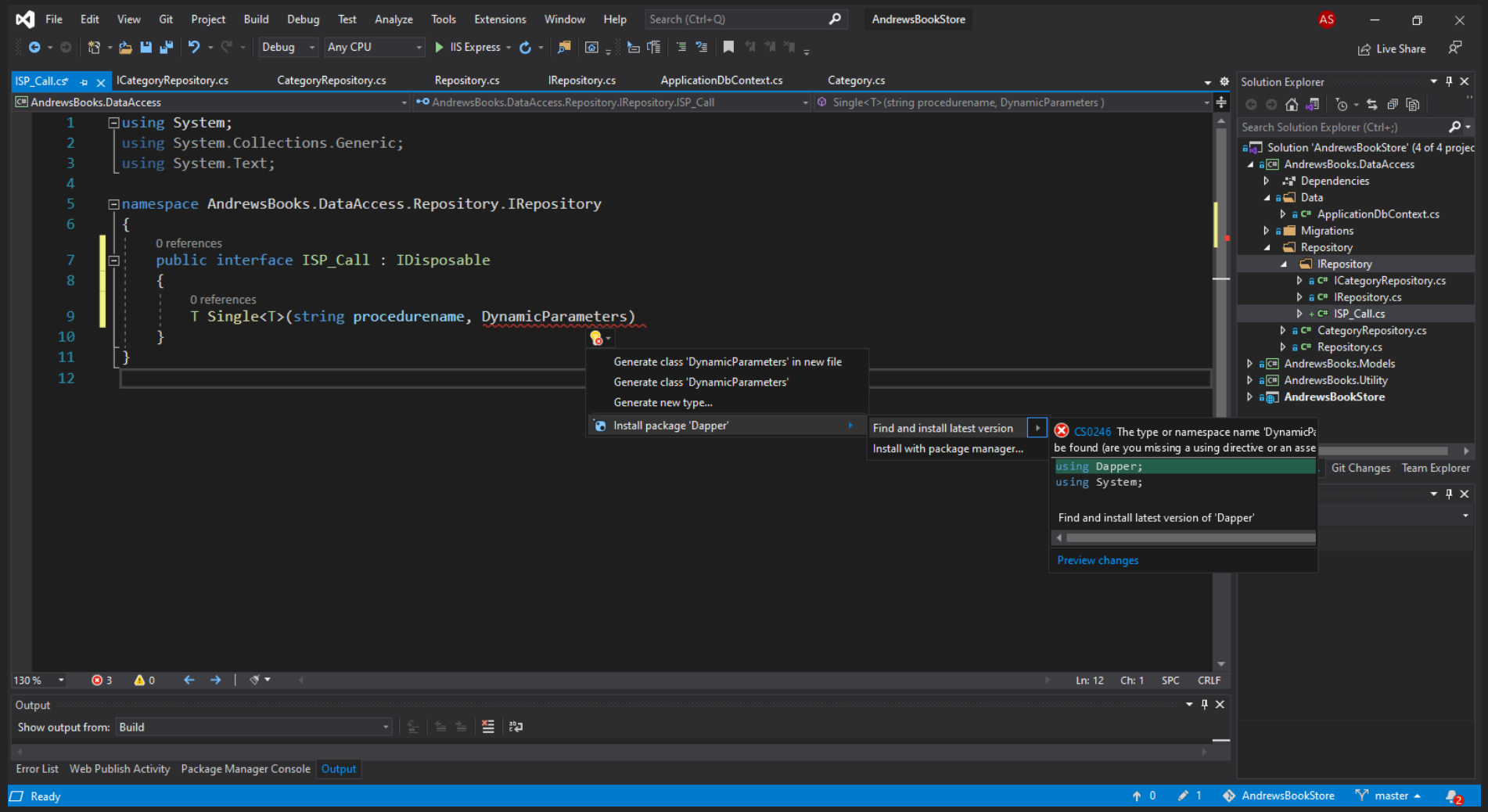
- Add a new interface in the IRepository folder - ISP_Call.cs – that extends IDisposable
- Include the methods shown (note the comments) and install the NuGet package for Dapper



- Implement a stored procedure repository and map multiple repositories in a Unit of Work

- Add a new interface in the IRepository folder - ISP_Call.cs – that extends IDisposable
- Include the methods shown (note the comments) and install the NuGet package for Dapper

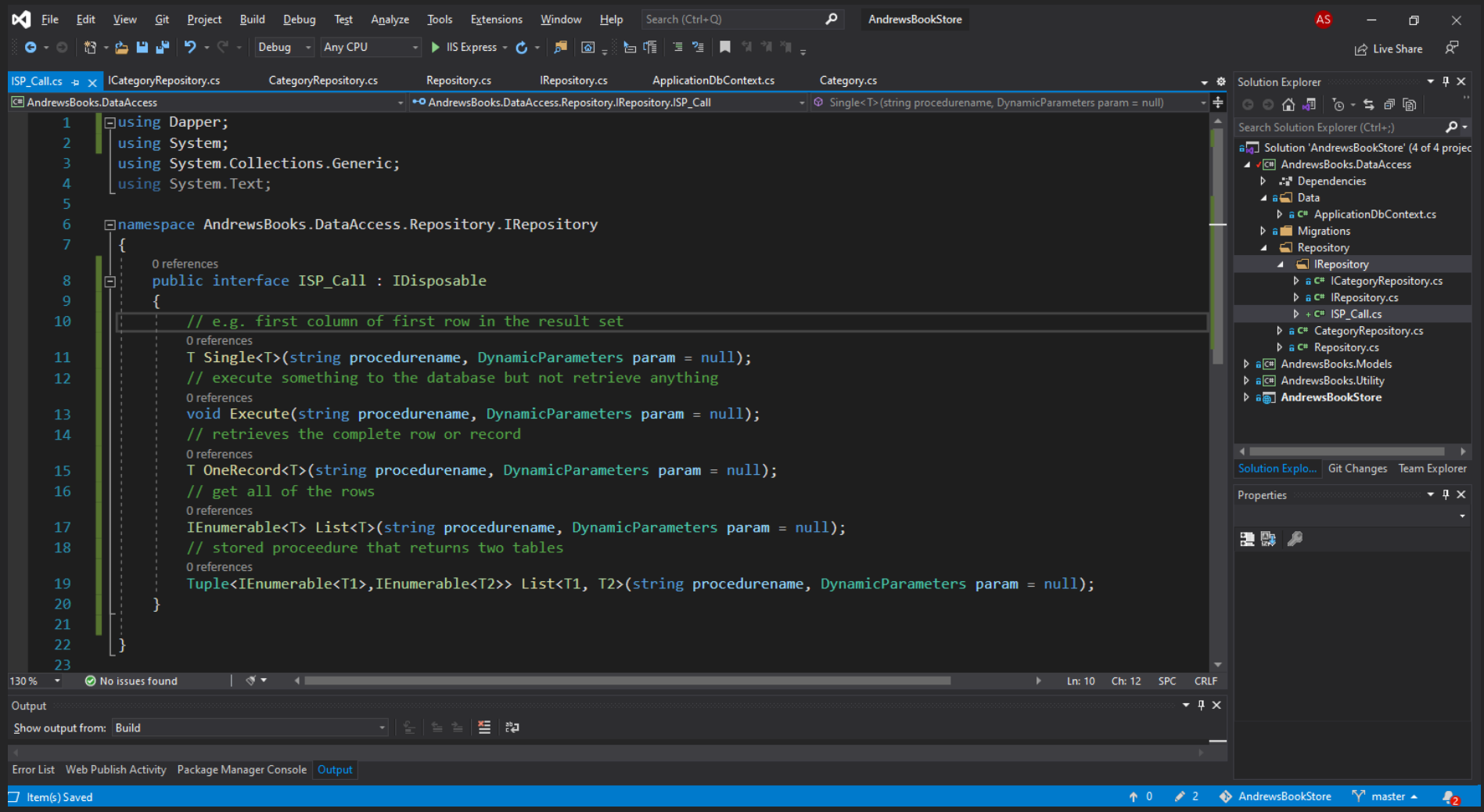
- Now implement this with a new class in the Repository folder.

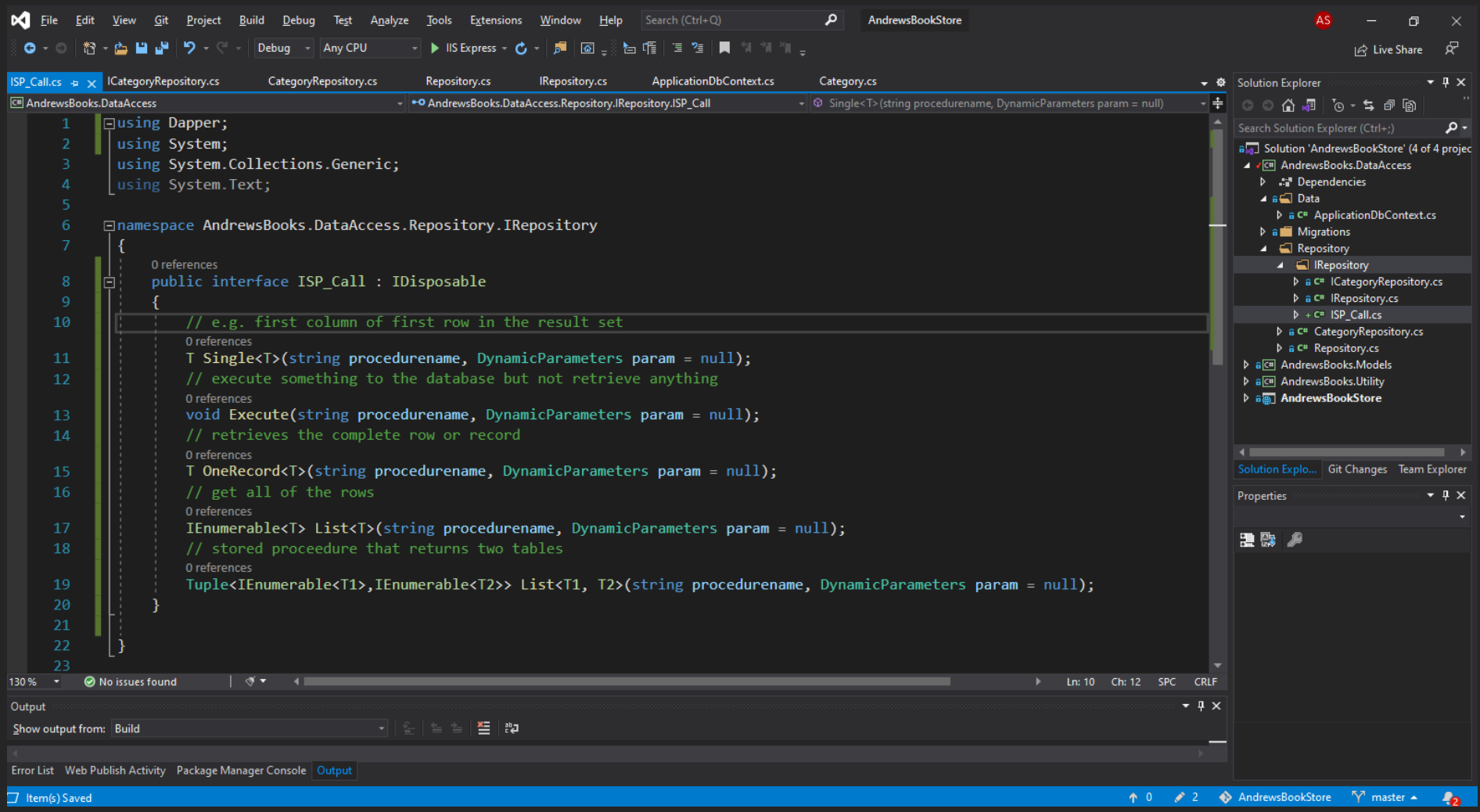


- Implement a stored procedure repository and map multiple repositories in a Unit of Work

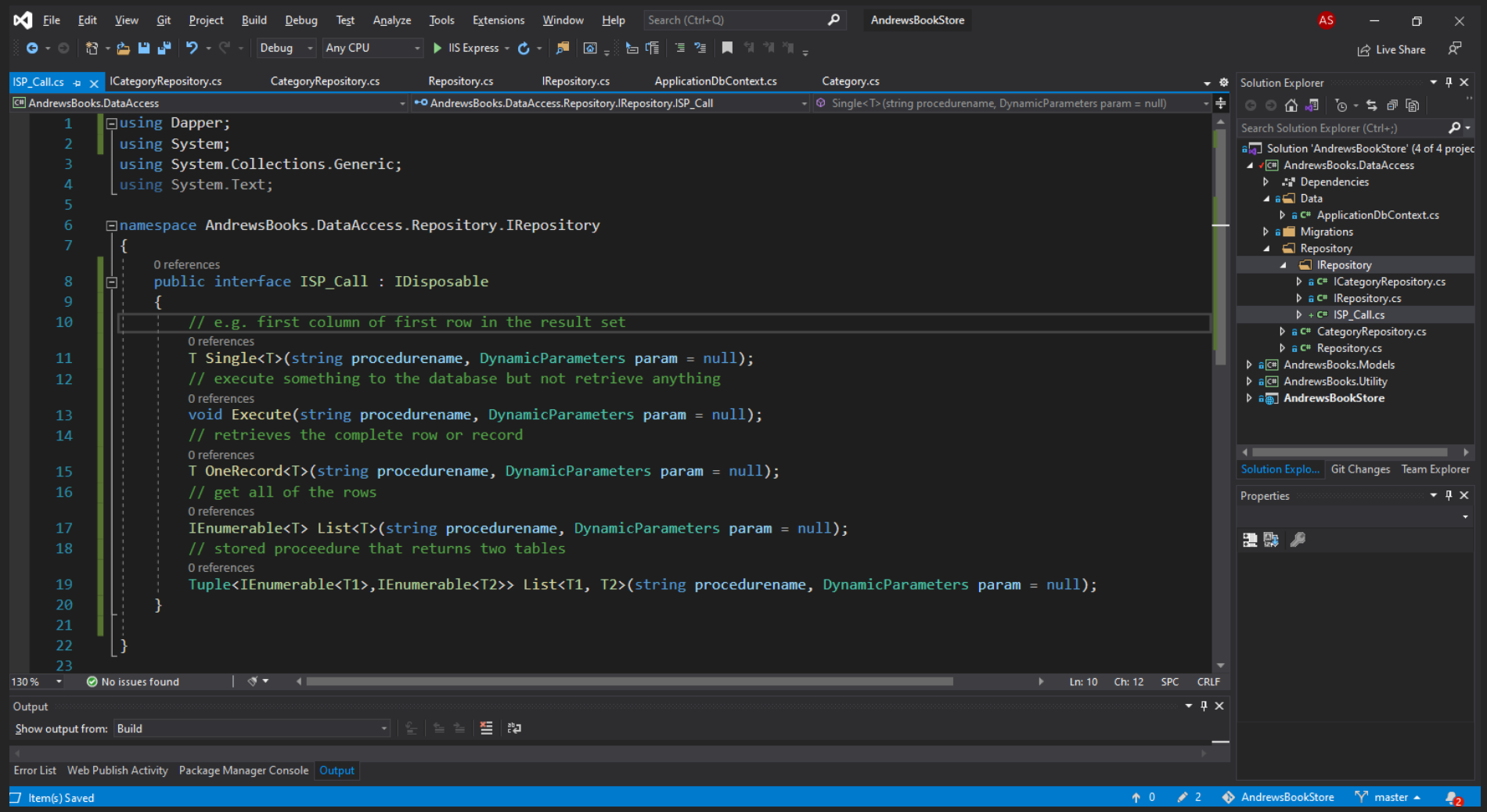
- Add a new interface in the IRepository folder - ISP_Call.cs – that extends IDisposable
- Include the methods shown (note the comments) and install the NuGet package for Dapper

- Now implement this with a new class in the Repository folder.

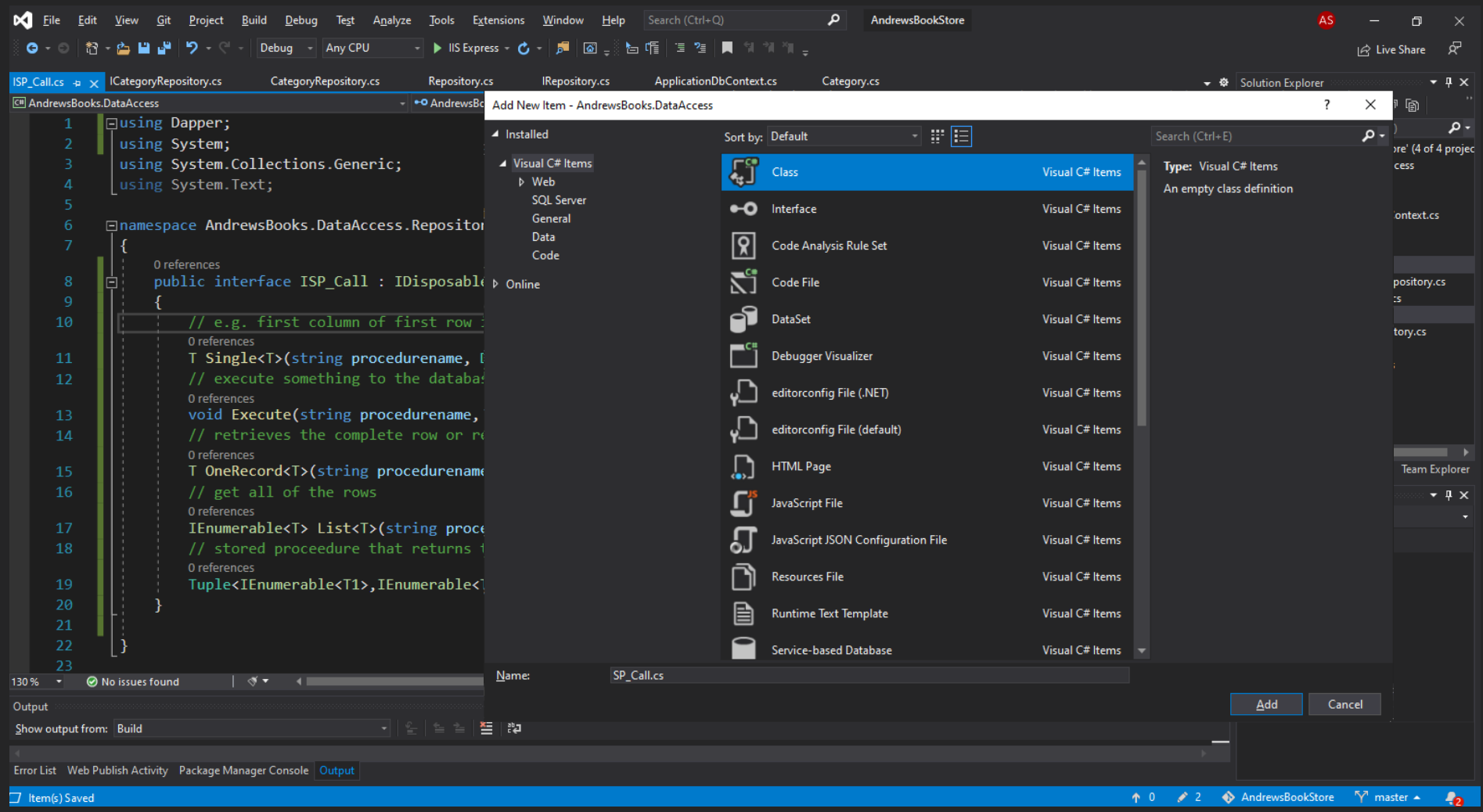



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
AndrewsBookStore
Debug
Any CPU
IIS Express
Live Share
ISP_Call.cs
ICategoryRepository.cs
CategoryRepository.cs
Repository.cs
IRepository.cs
ApplicationDbContext.cs
Category.cs
AndrewsBooks.DataAccess
AndrewsBooks.DataAccess.Repository.IRepository.ISP_Call
Single<T>(string procedurename, DynamicParameters param = null)
using Dapper;
using System;
using System.Collections.Generic;
using System.Text;
namespace AndrewsBooks.DataAccess.Repository.IRepository
{
0 references
public interface ISP_Call : IDisposable
{
// e.g. first column of first row in the result set
0 references
T Single<T>(string procedurename, DynamicParameters param = null);
// execute something to the database but not retrieve anything
0 references
void Execute(string procedurename, DynamicParameters param = null);
// retrieves the complete row or record
0 references
T OneRecord<T>(string procedurename, DynamicParameters param = null);
// get all of the rows
0 references
IEnumerable<T> List<T>(string procedurename, DynamicParameters param = null);
// stored proceedure that returns two tables
0 references
Tuple<IEnumerable<T1>,IEnumerable<T2>> List<T1, T2>(string procedurename, DynamicParameters param = null);
}
}
130 %
No issues found
Ln: 10
Ch: 12
SPC
CRLF
Output
Show output from: Build
Error List
Web Publish Activity
Package Manager Console
Output
Item(s) Saved
0
2
AndrewsBookStore
master
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AndrewsBookStore' (4 of 4 projec
AndrewsBooks.DataAccess
Dependencies
Data
ApplicationDbContext.cs
Migrations
Repository
IRepository
ICategoryRepository.cs
IRepository.cs
ISP_Call.cs
CategoryRepository.cs
Repository.cs
AndrewsBooks.Models
AndrewsBooks.Utility
AndrewsBookStore
Solution Explo...
Git Changes
Team Explorer
Properties

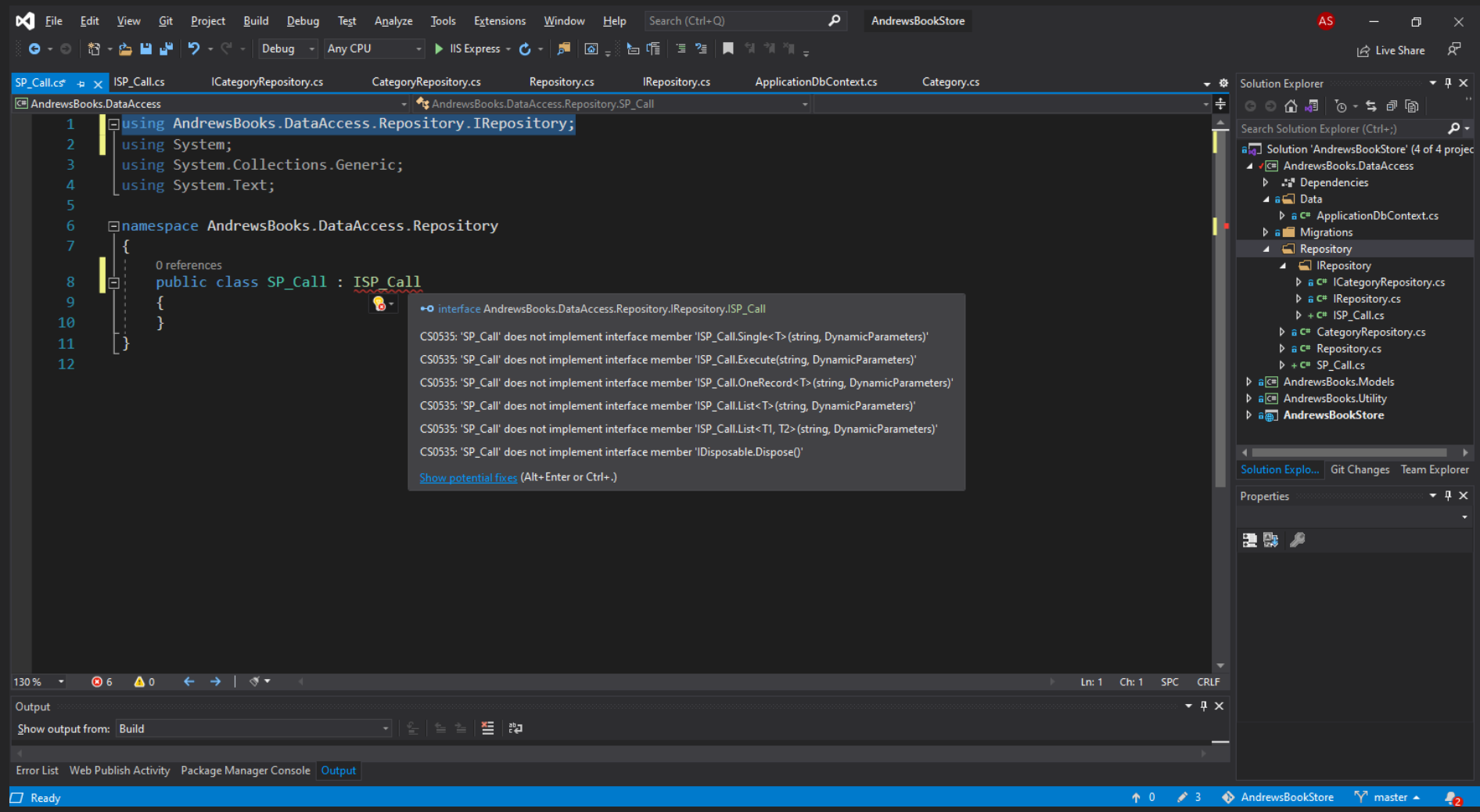
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface



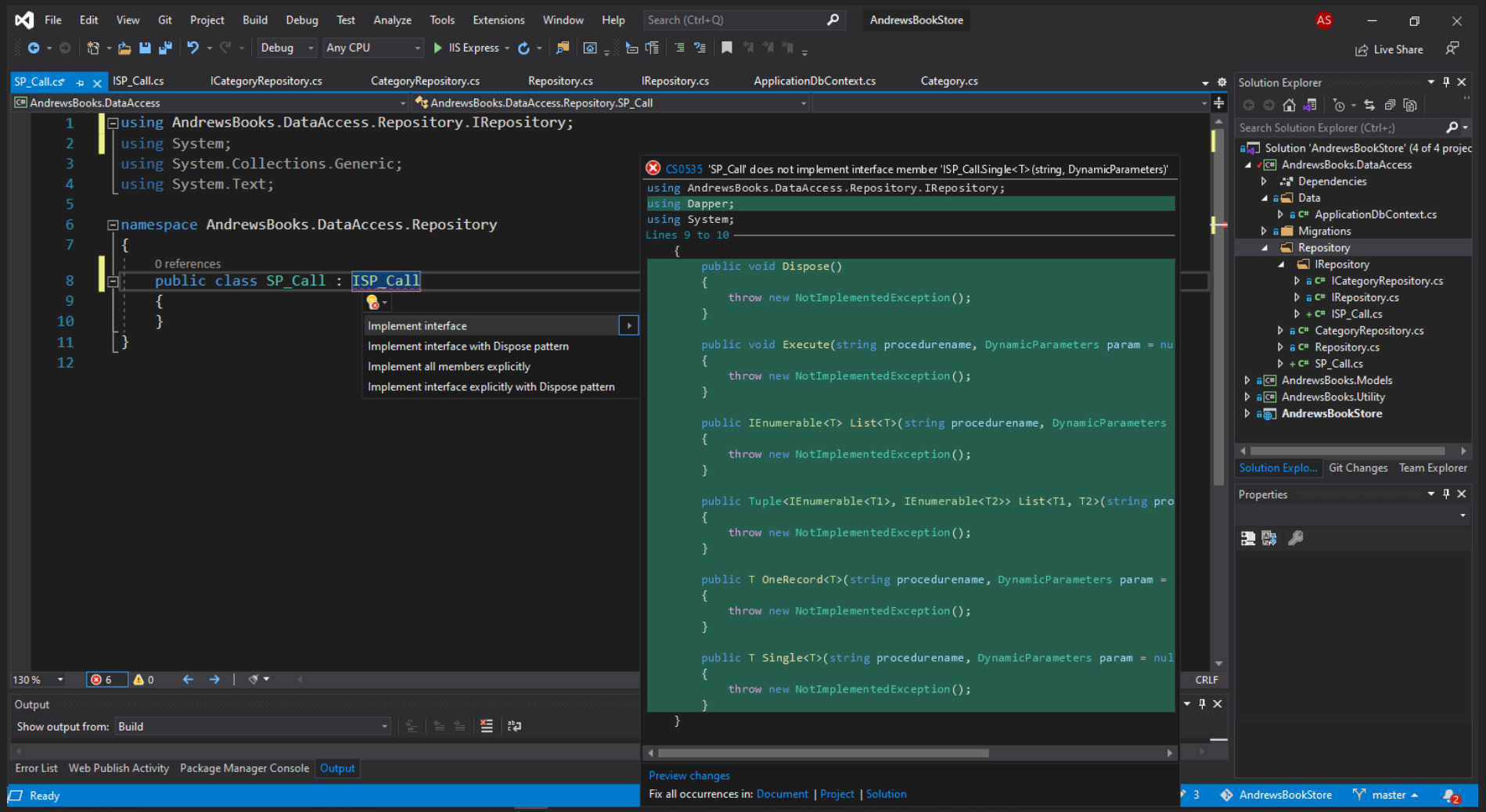
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface



- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

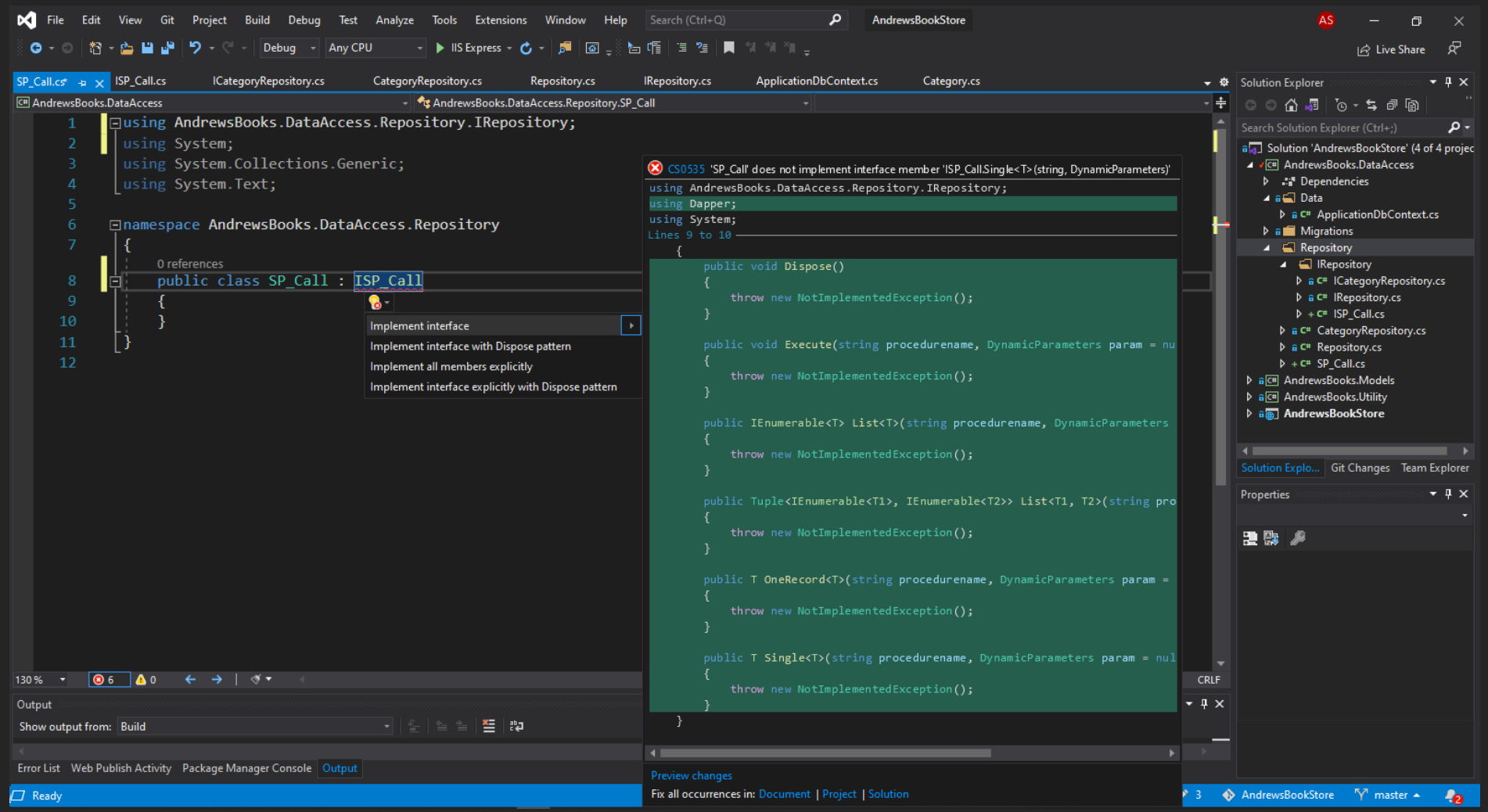


- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface



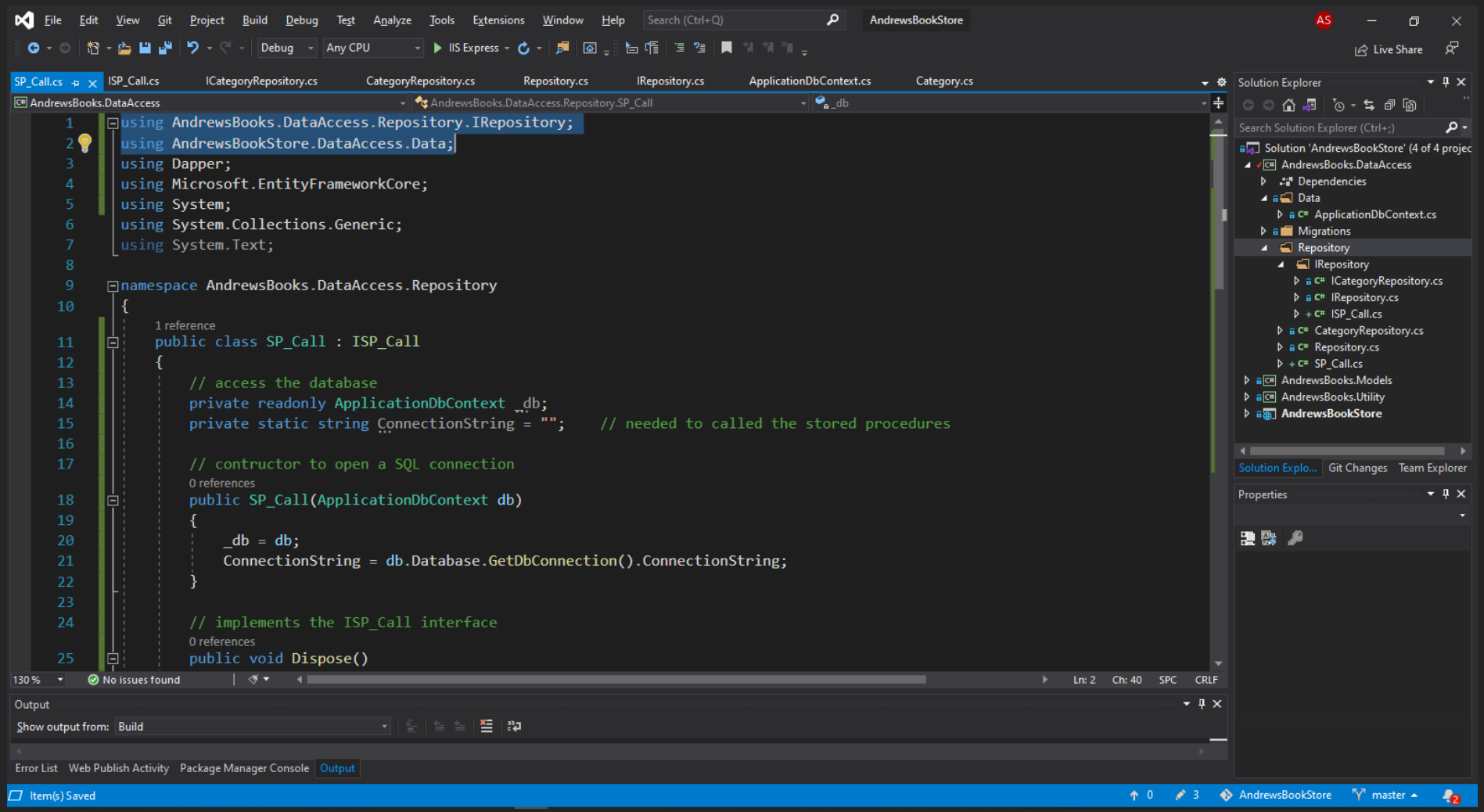
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements



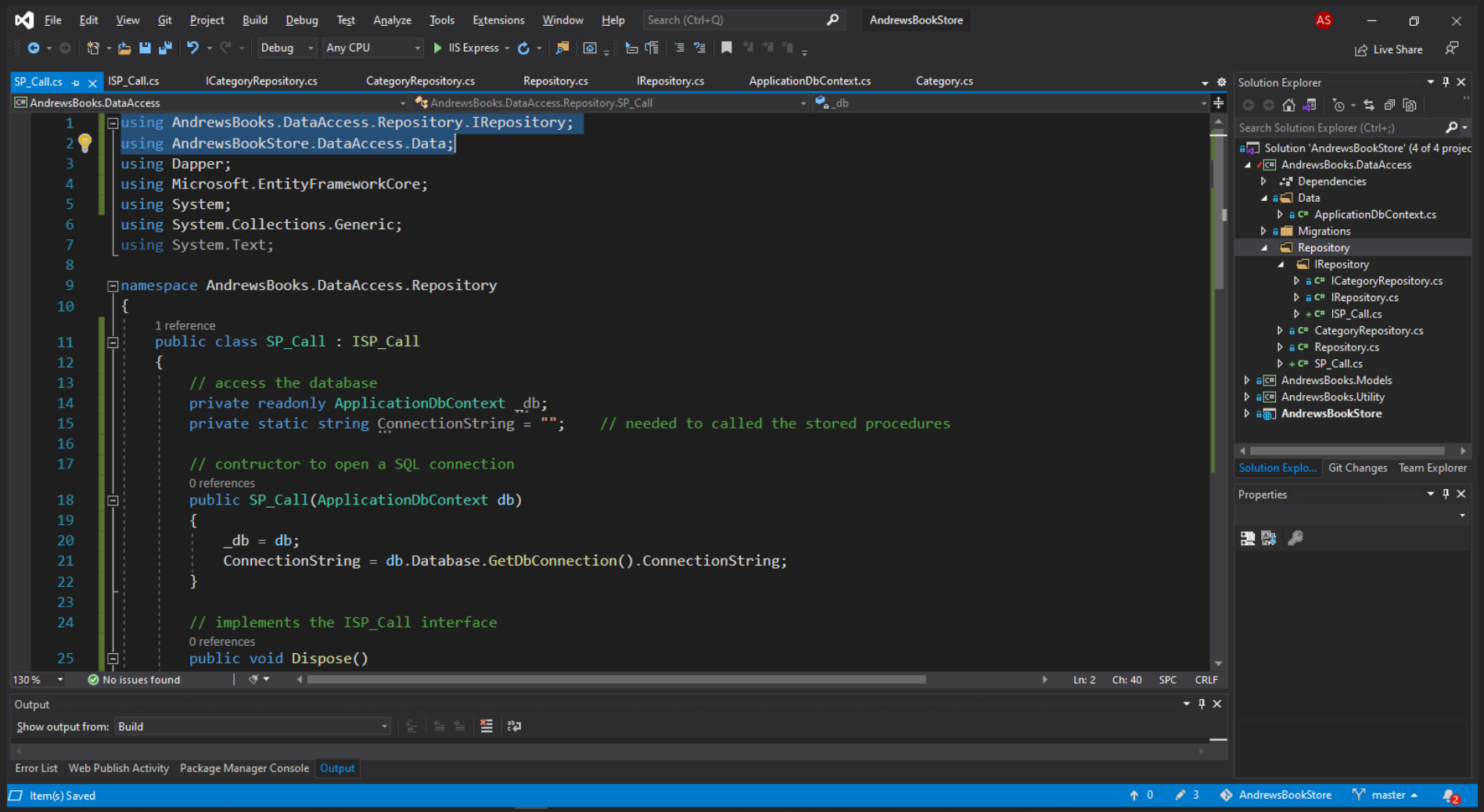
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements



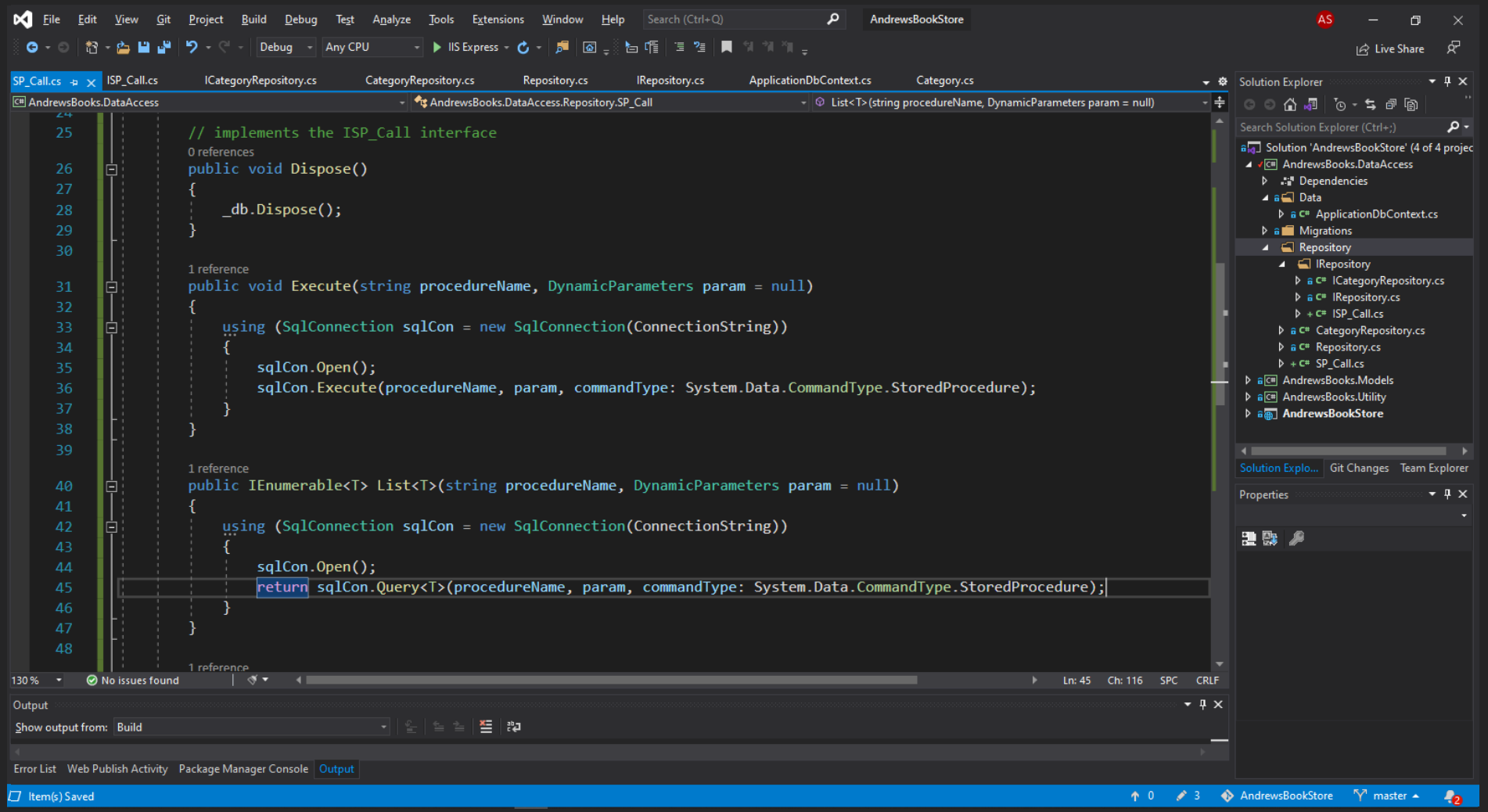
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface



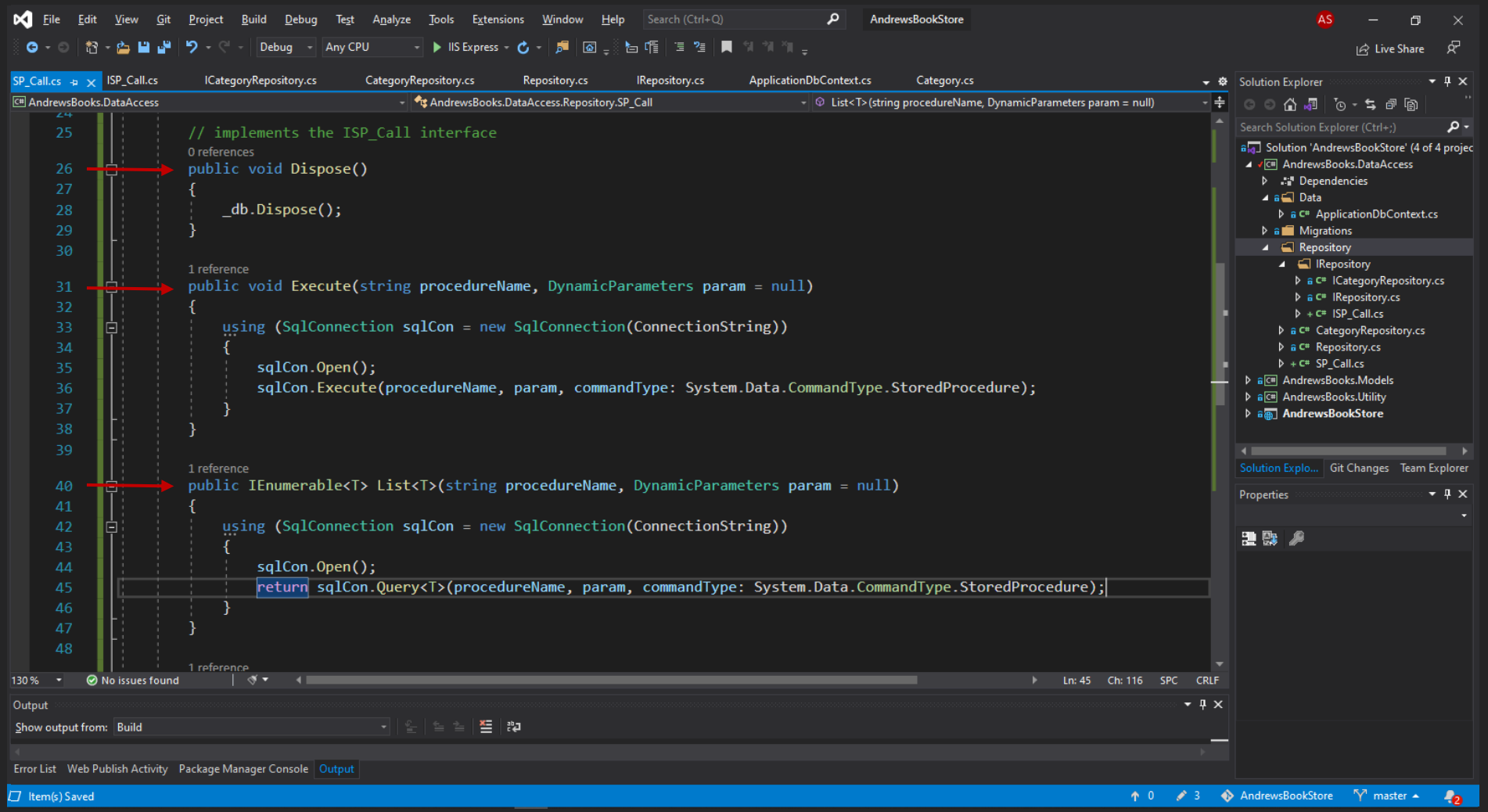
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface



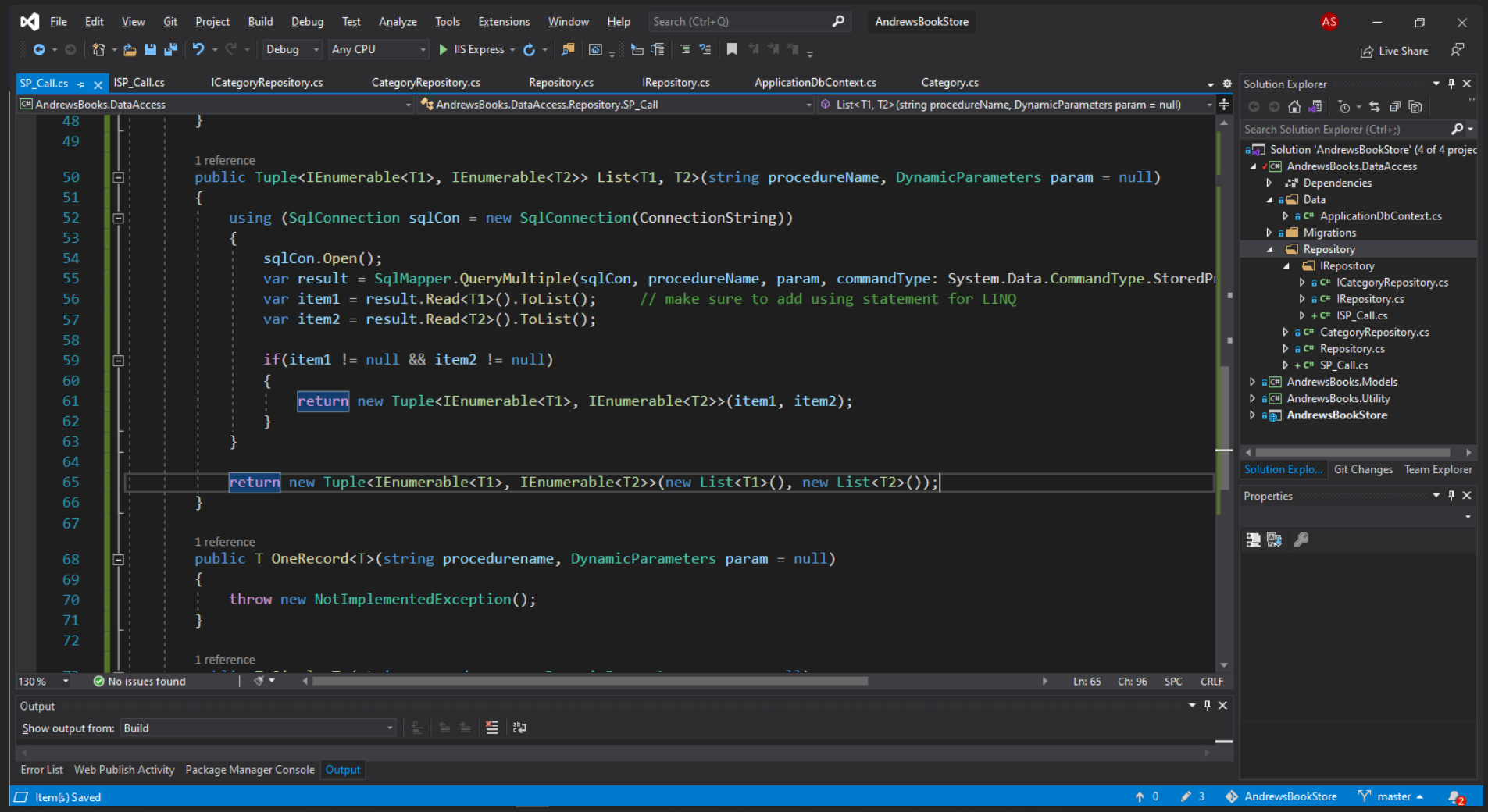
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface



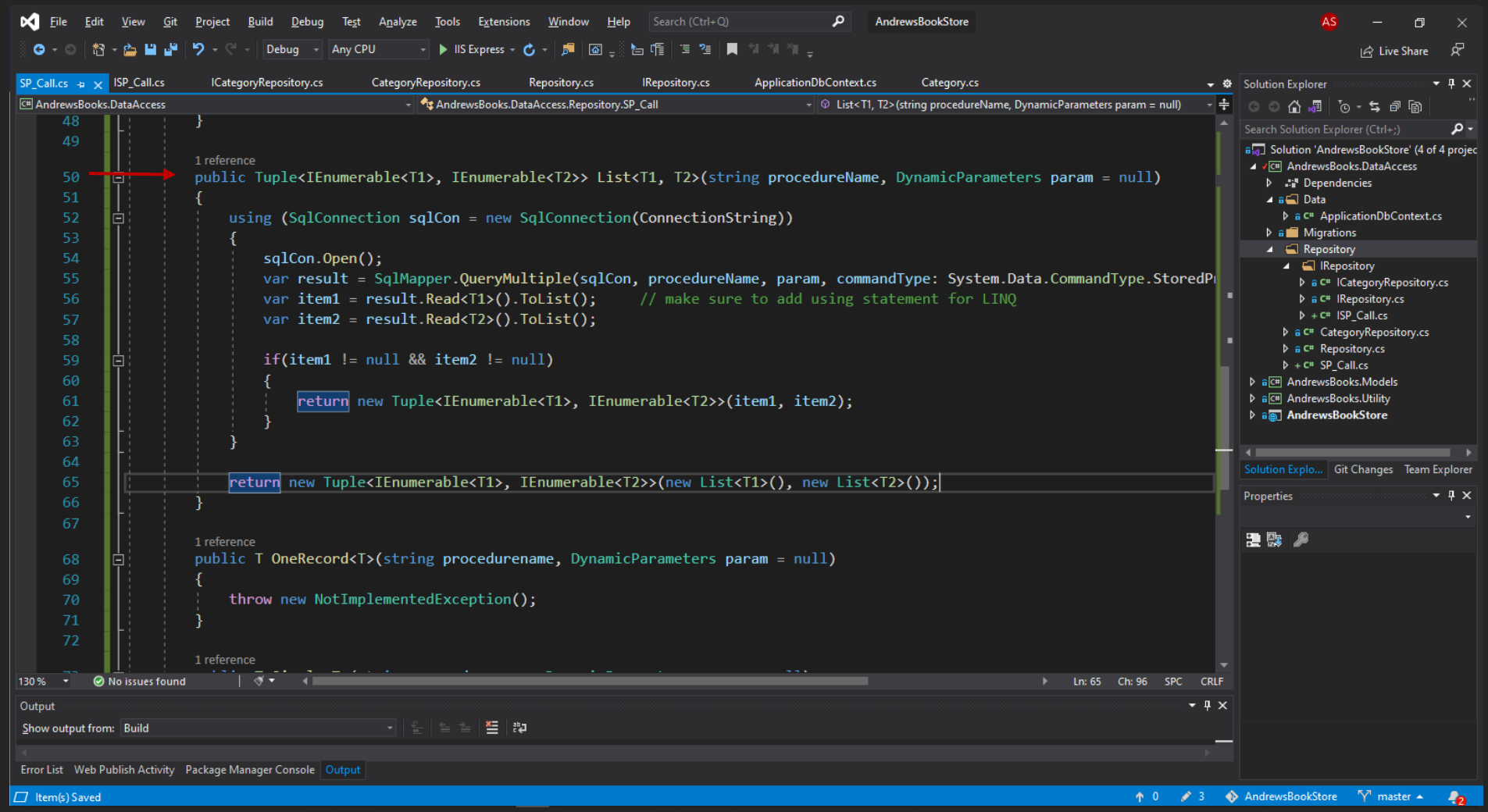
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface



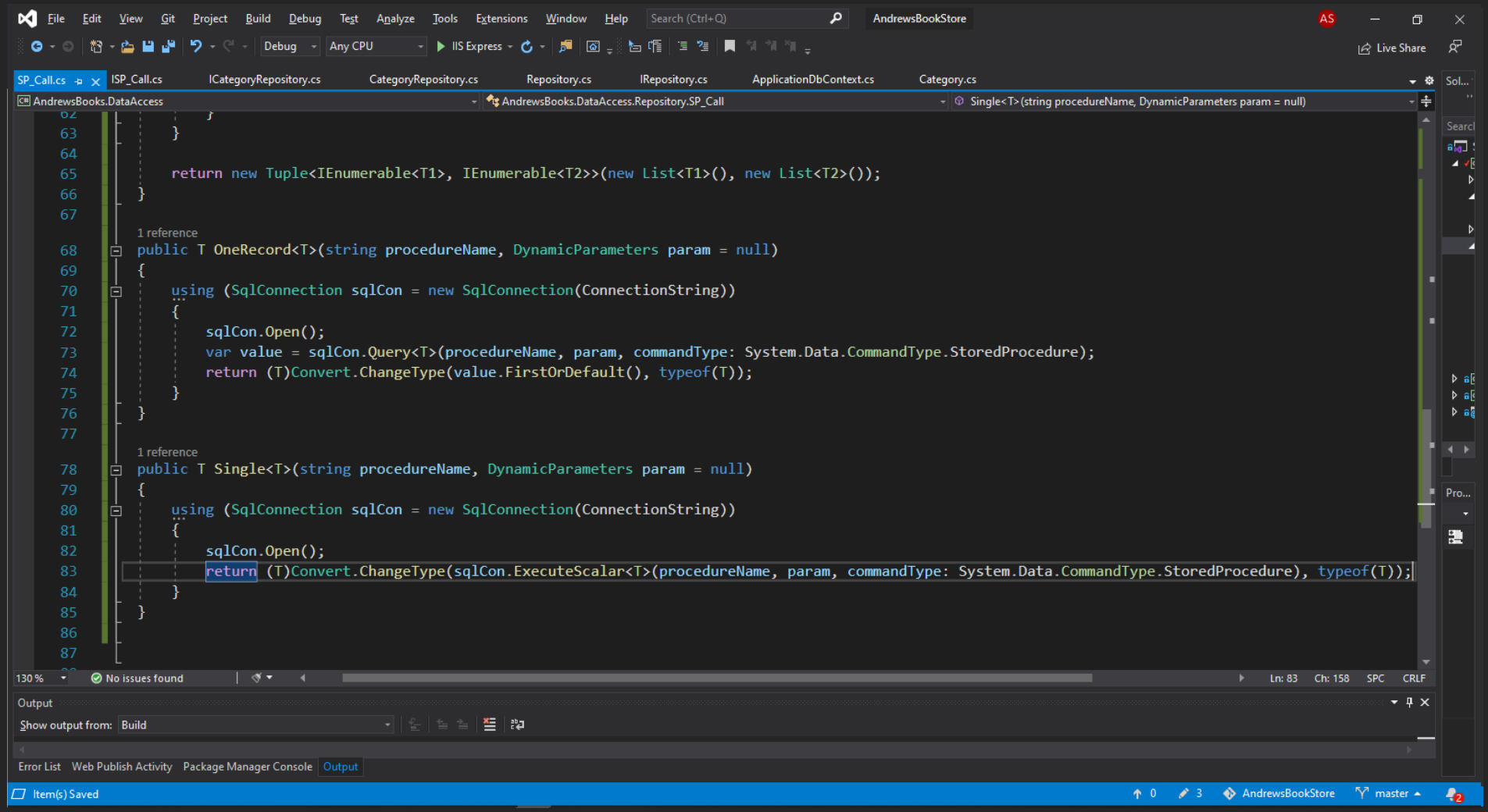
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface



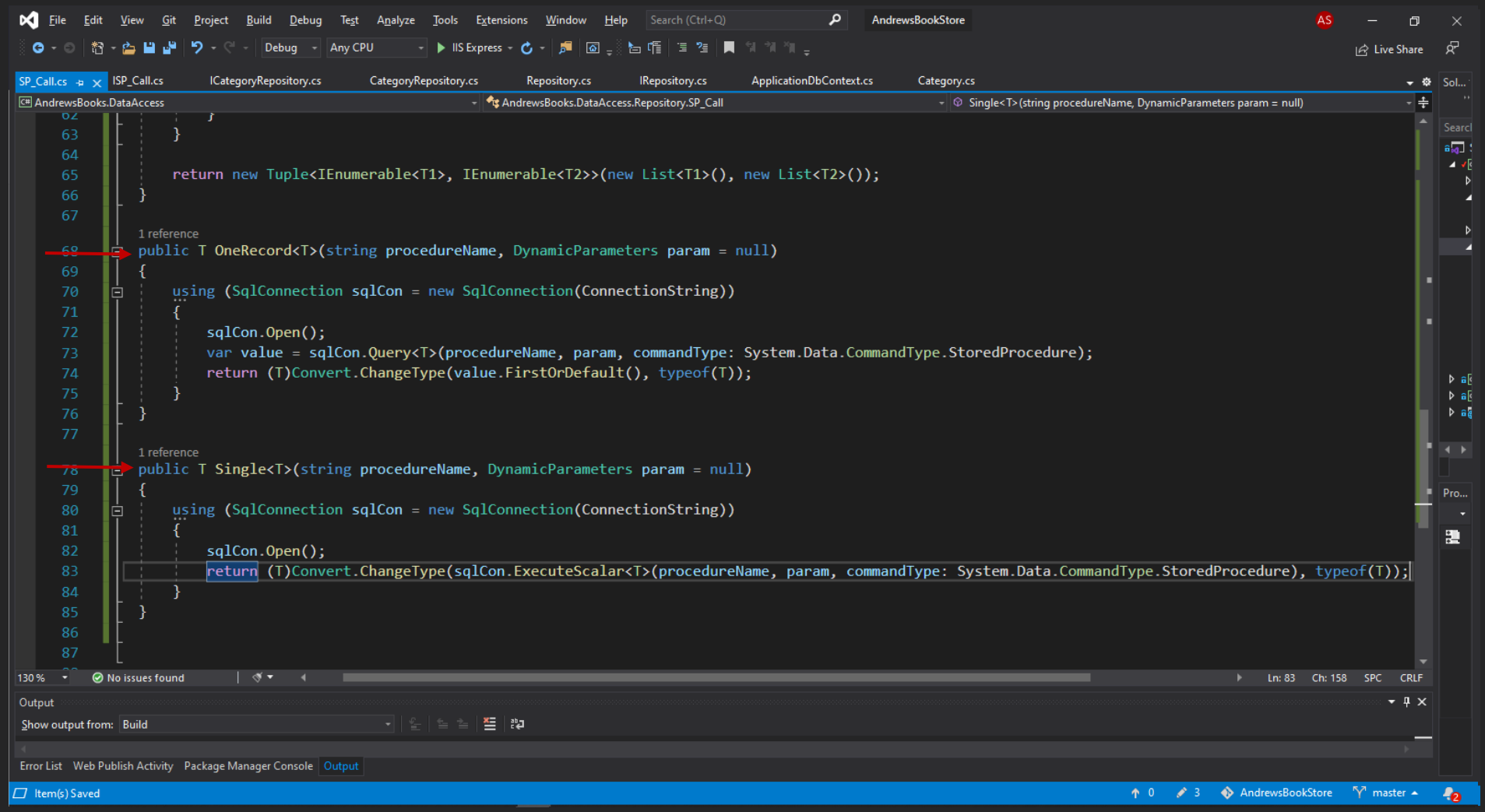
- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface



- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

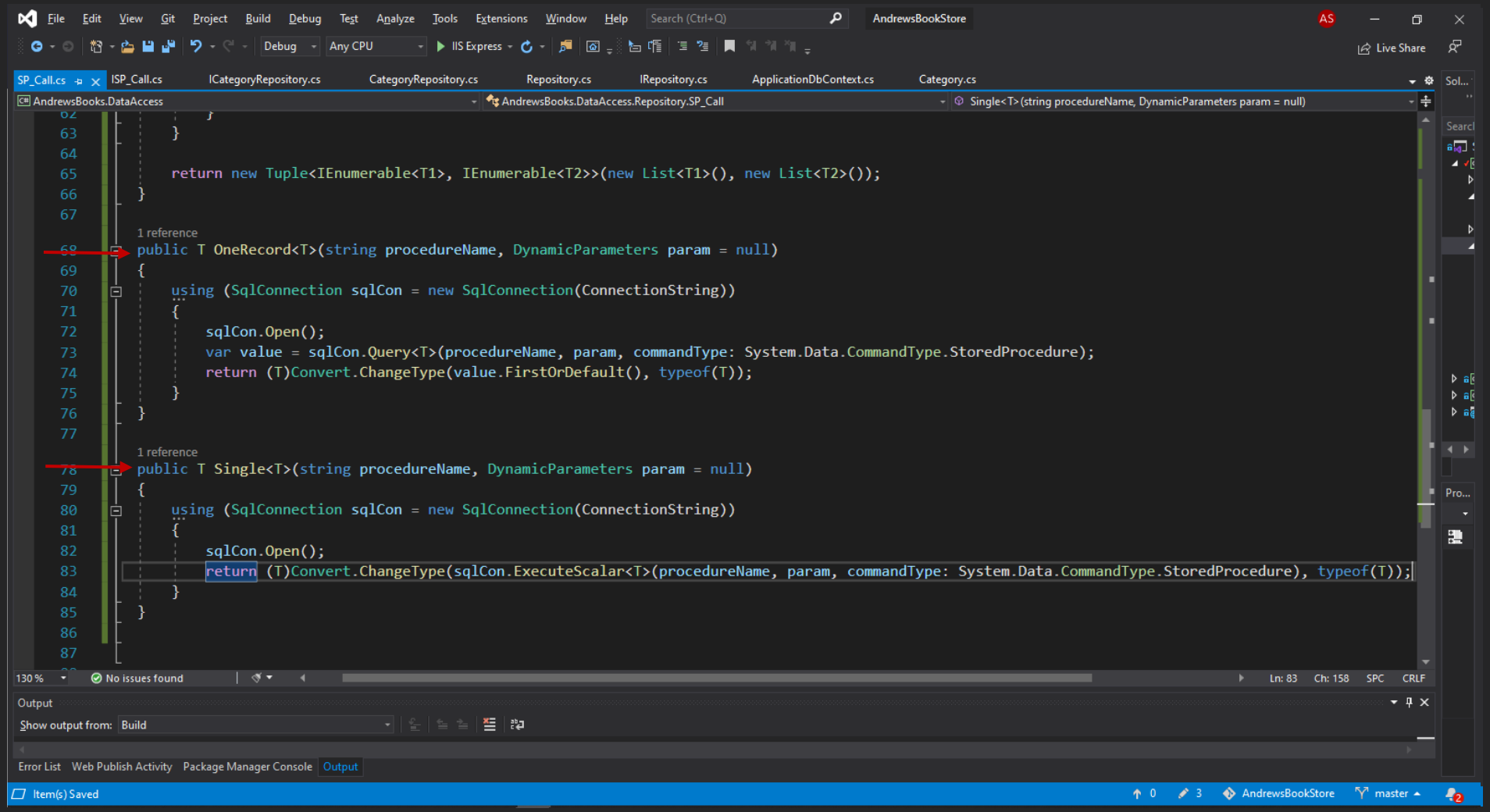
- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface

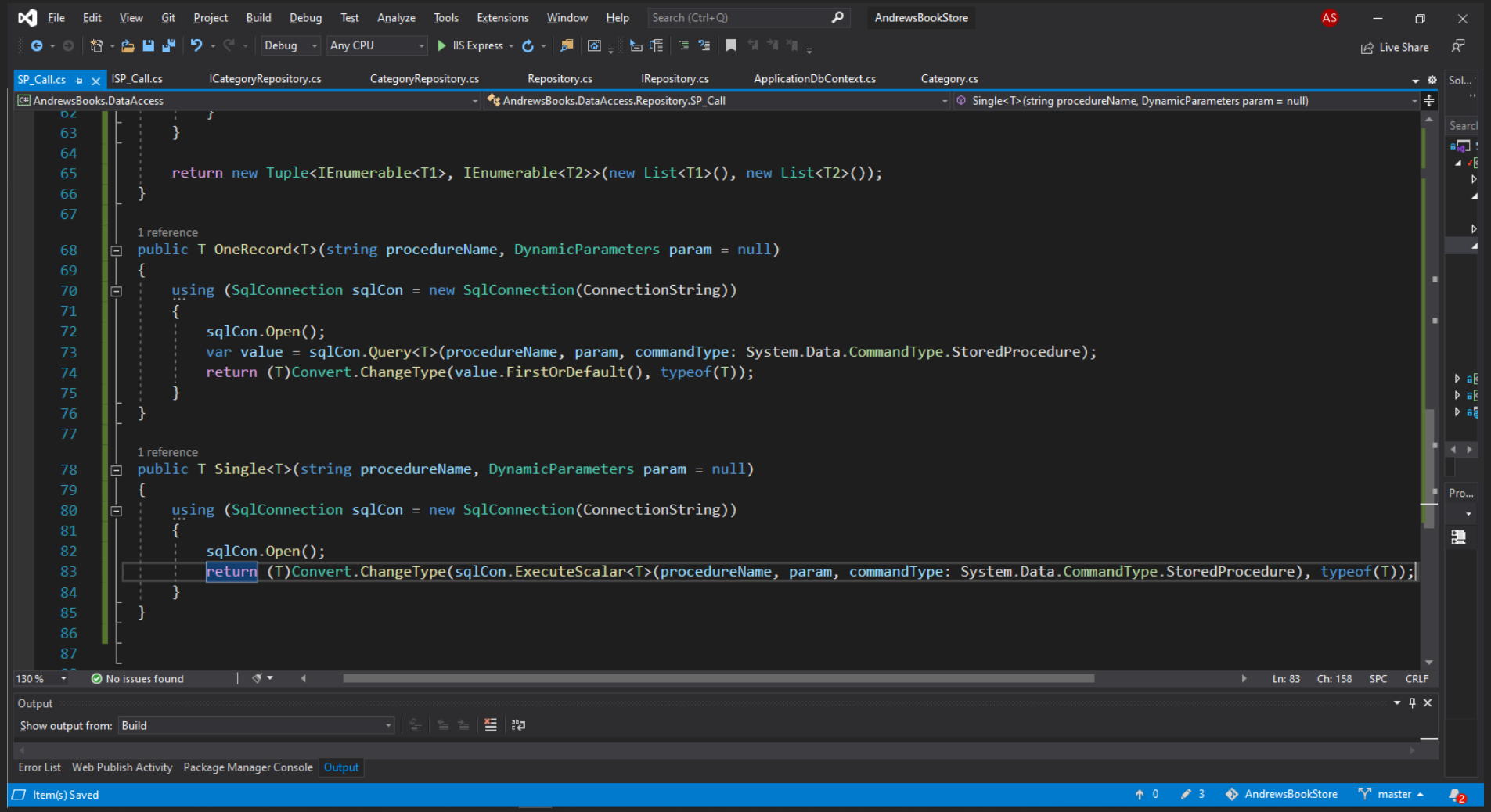


- Add a new class (SP_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP_Call interface

- Add a connection to the database, note the additional using statements
- Update the implementation of the ISP_Call interface

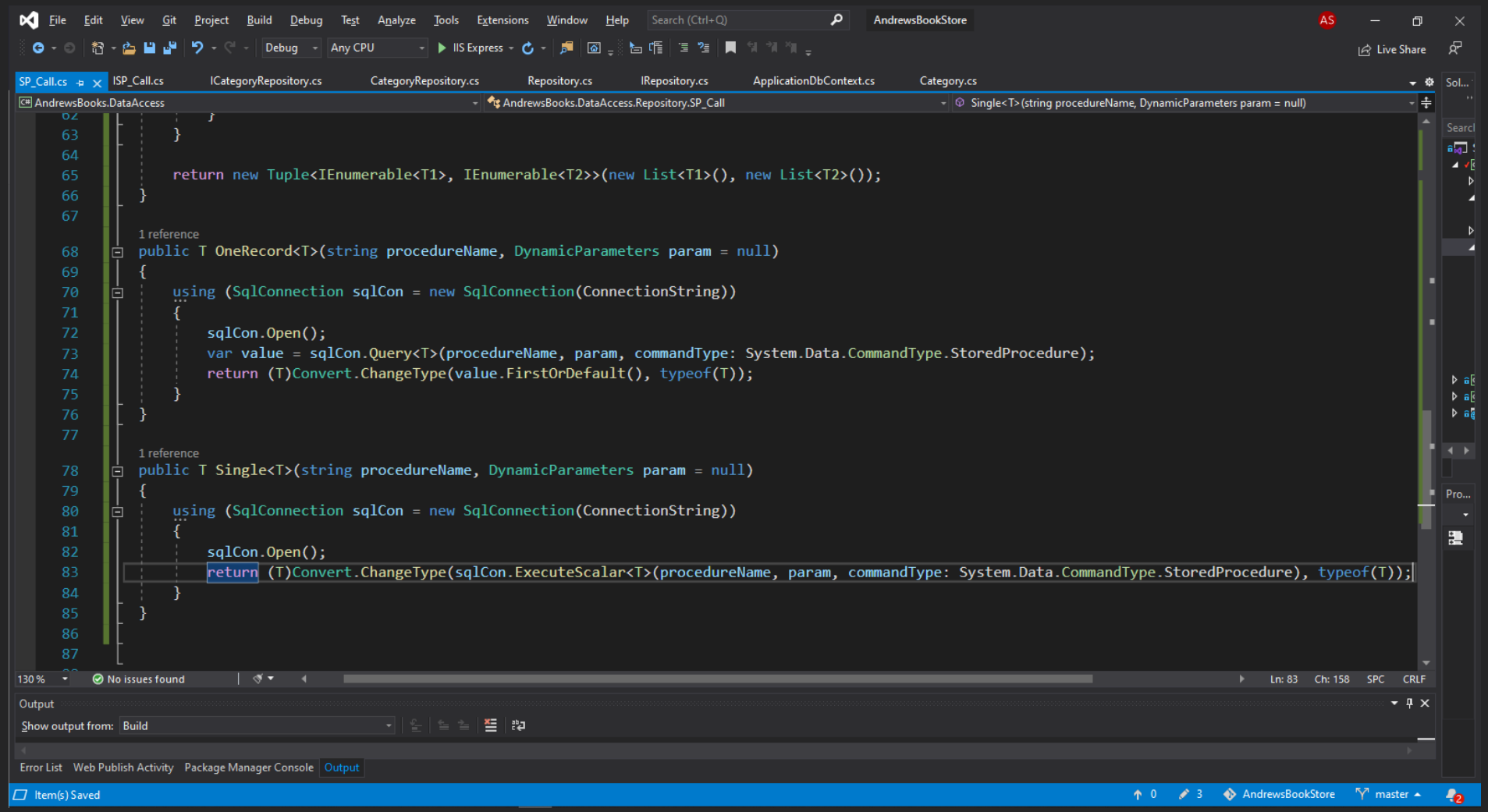
- Now add the wrapper for Unit of Work





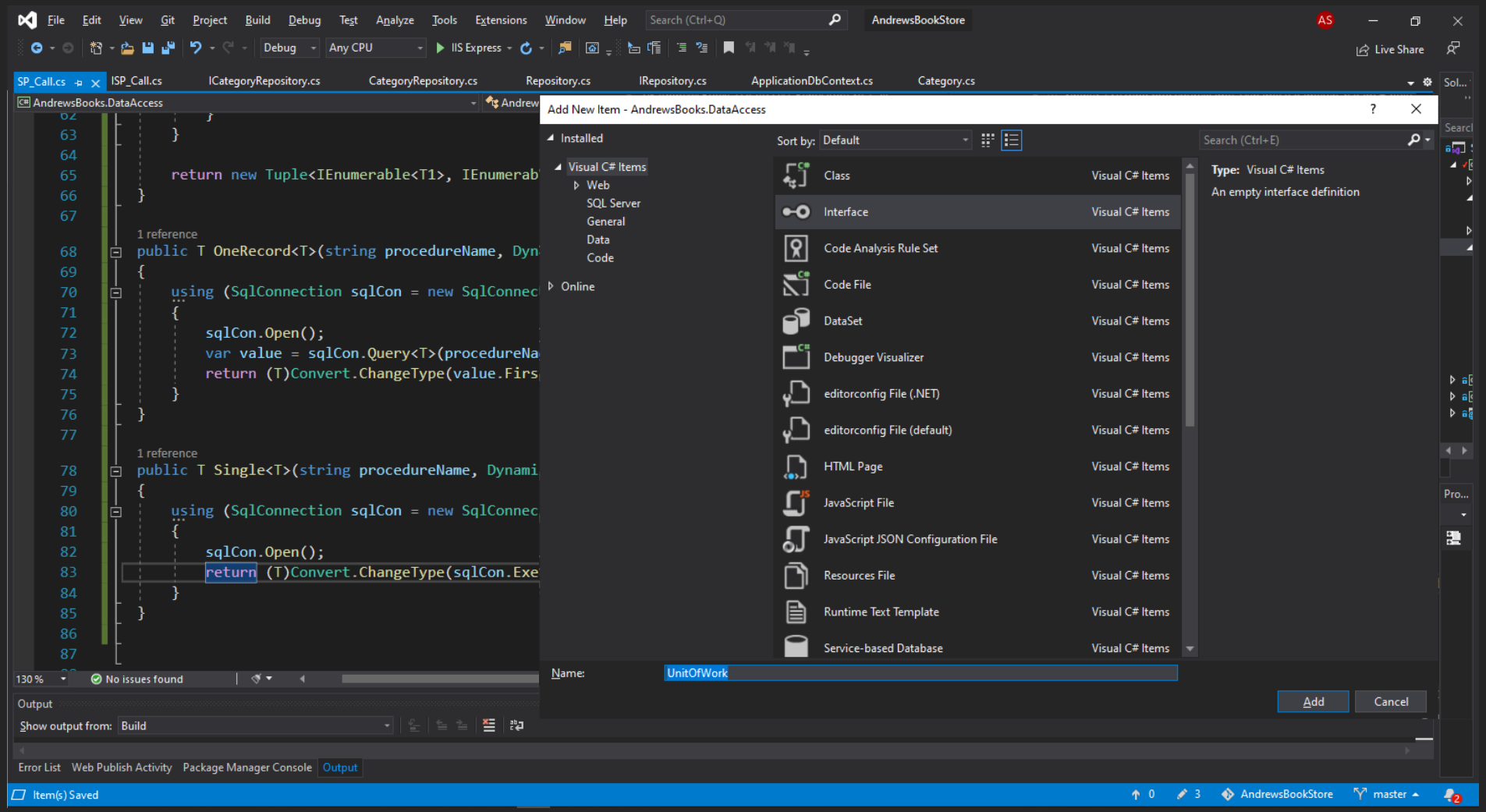
** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.



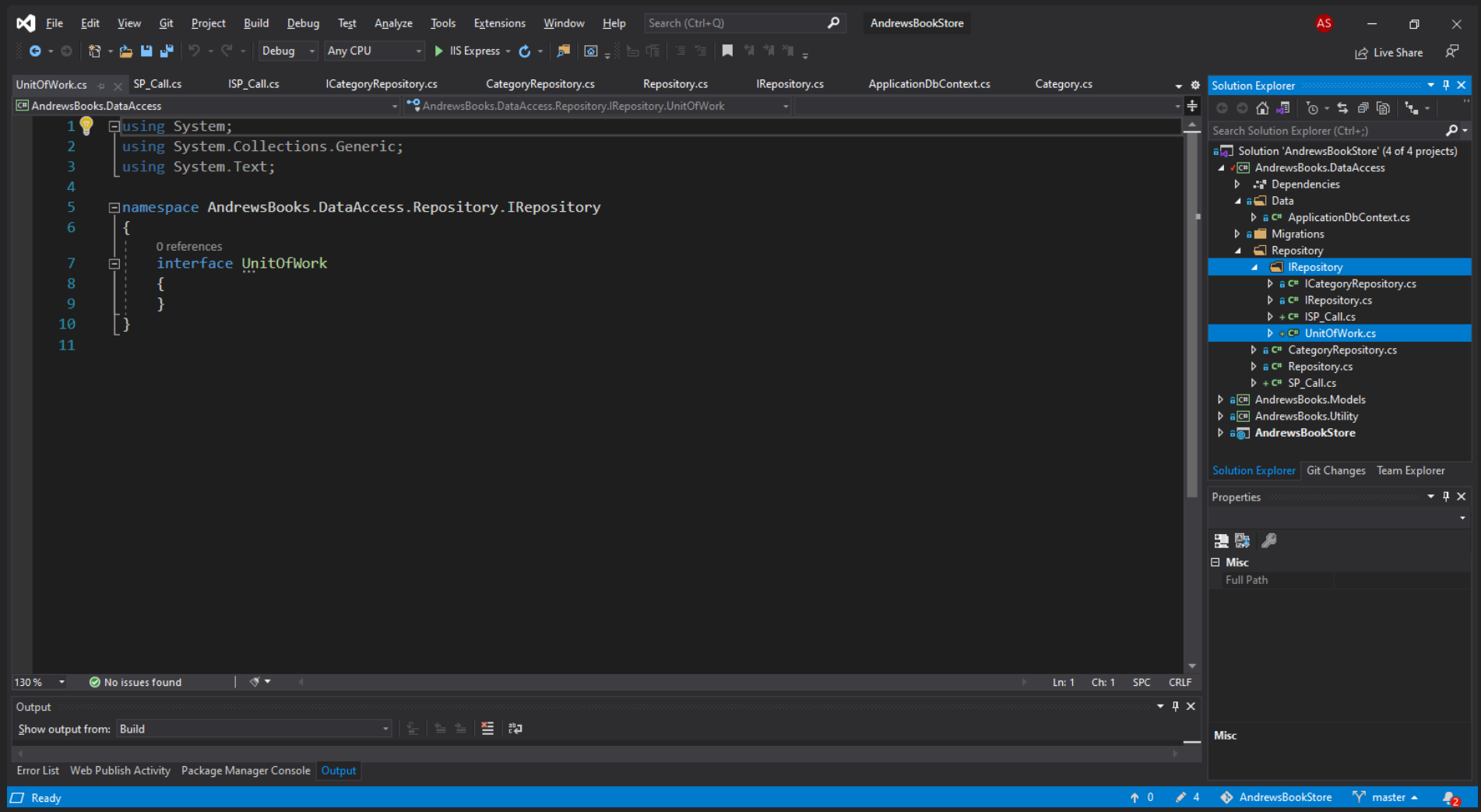
** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.



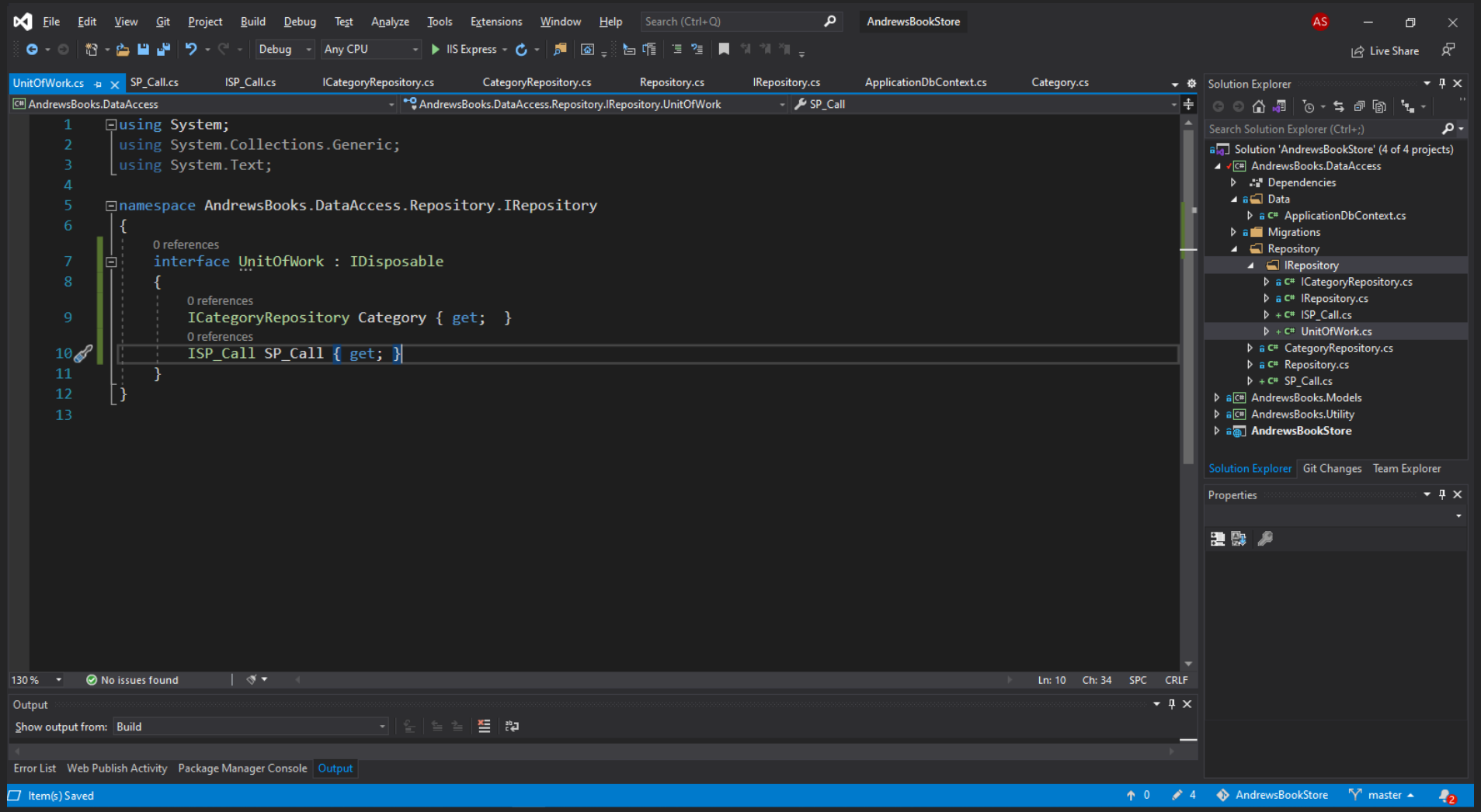
** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.



** Errata > The Interface file name is - IUnitOfWork*

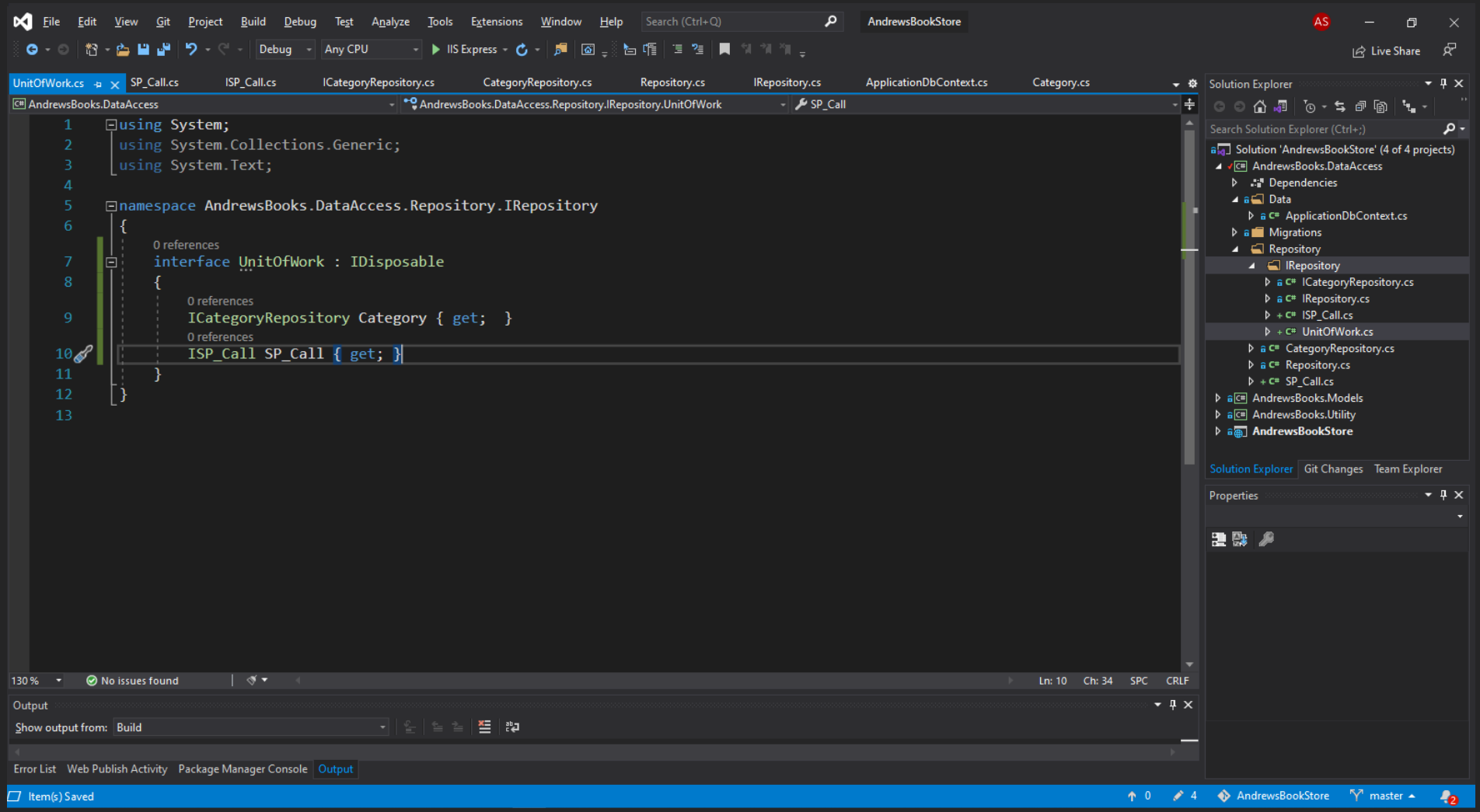
- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.



** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

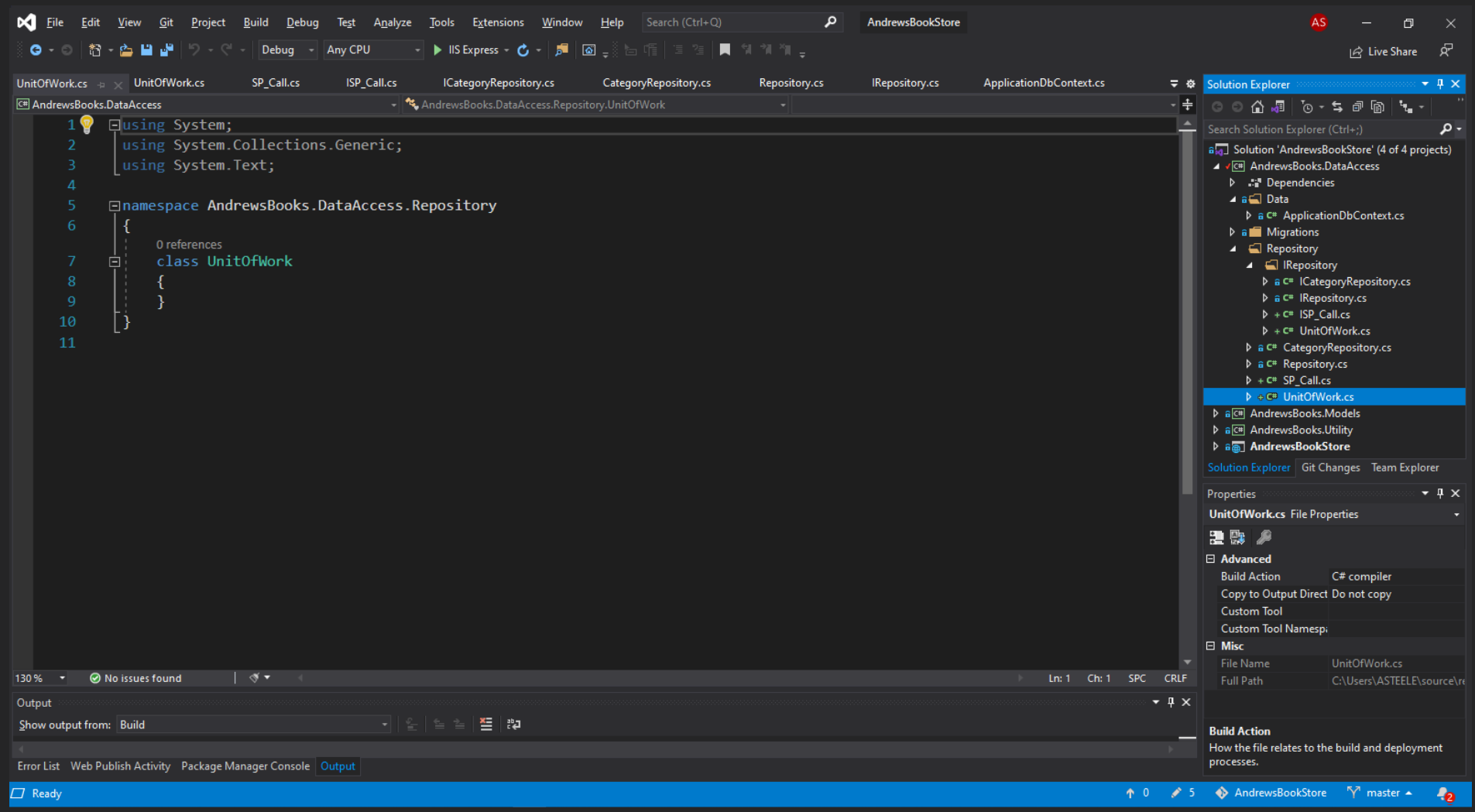
- Now implement this inside the UnitOfWork (Hint: Add a class)



** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

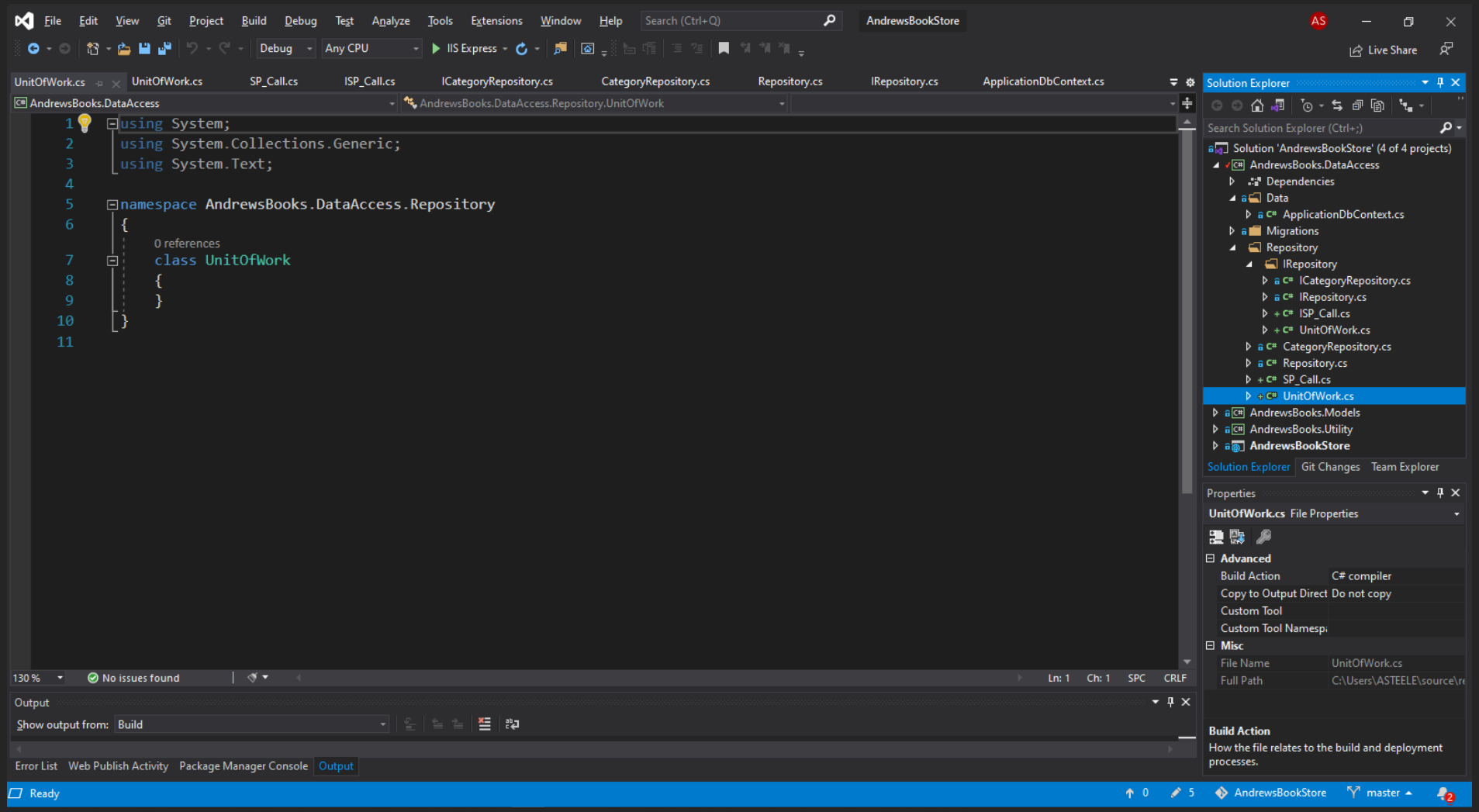
- Now implement this inside the UnitOfWork (Hint: Add a class)



** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

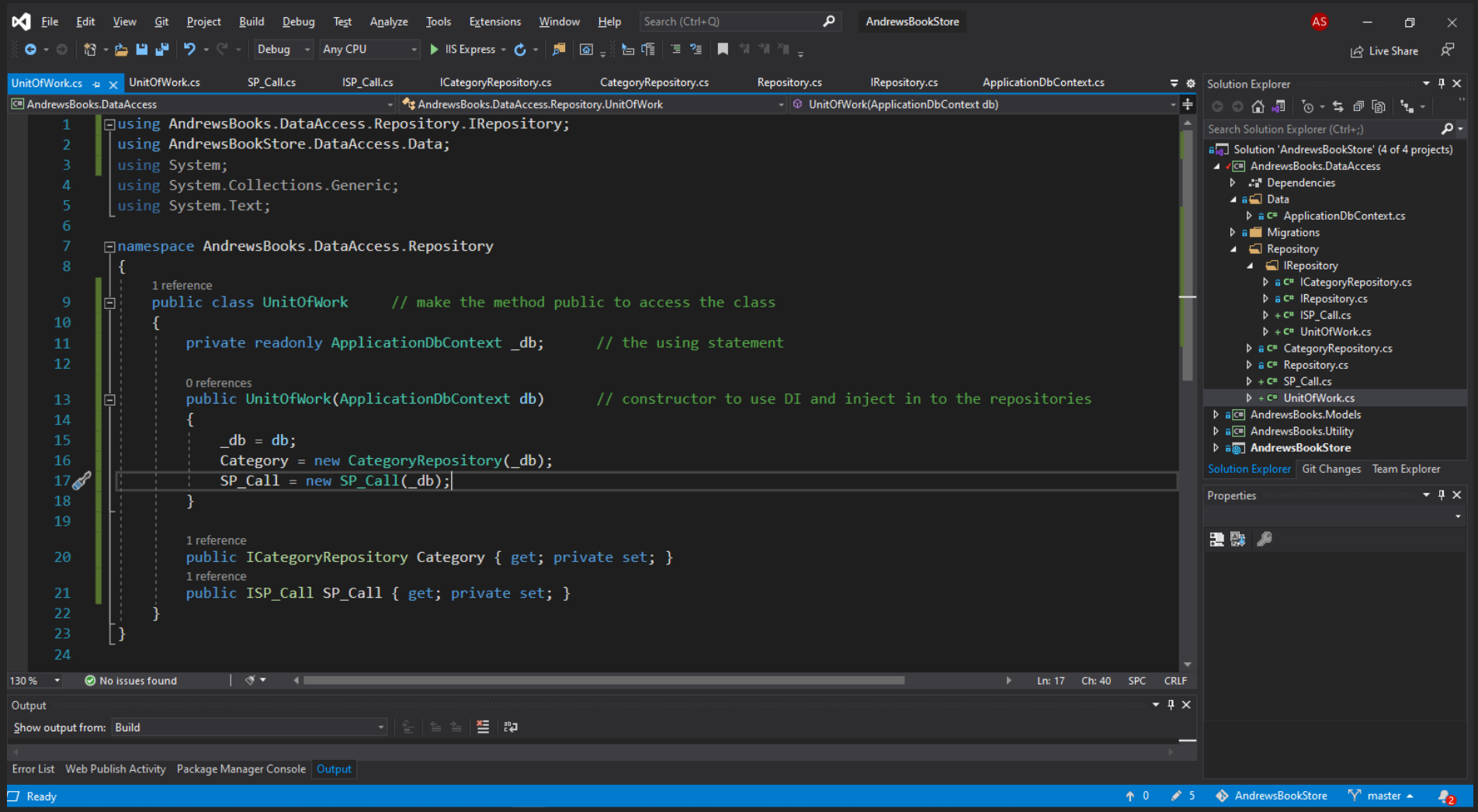
- Now implement this inside the UnitOfWork (Hint: Add a class)
- Modify the code (make sure the public class implements the interface - UnitOfWork : IUnitOfWork)



** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

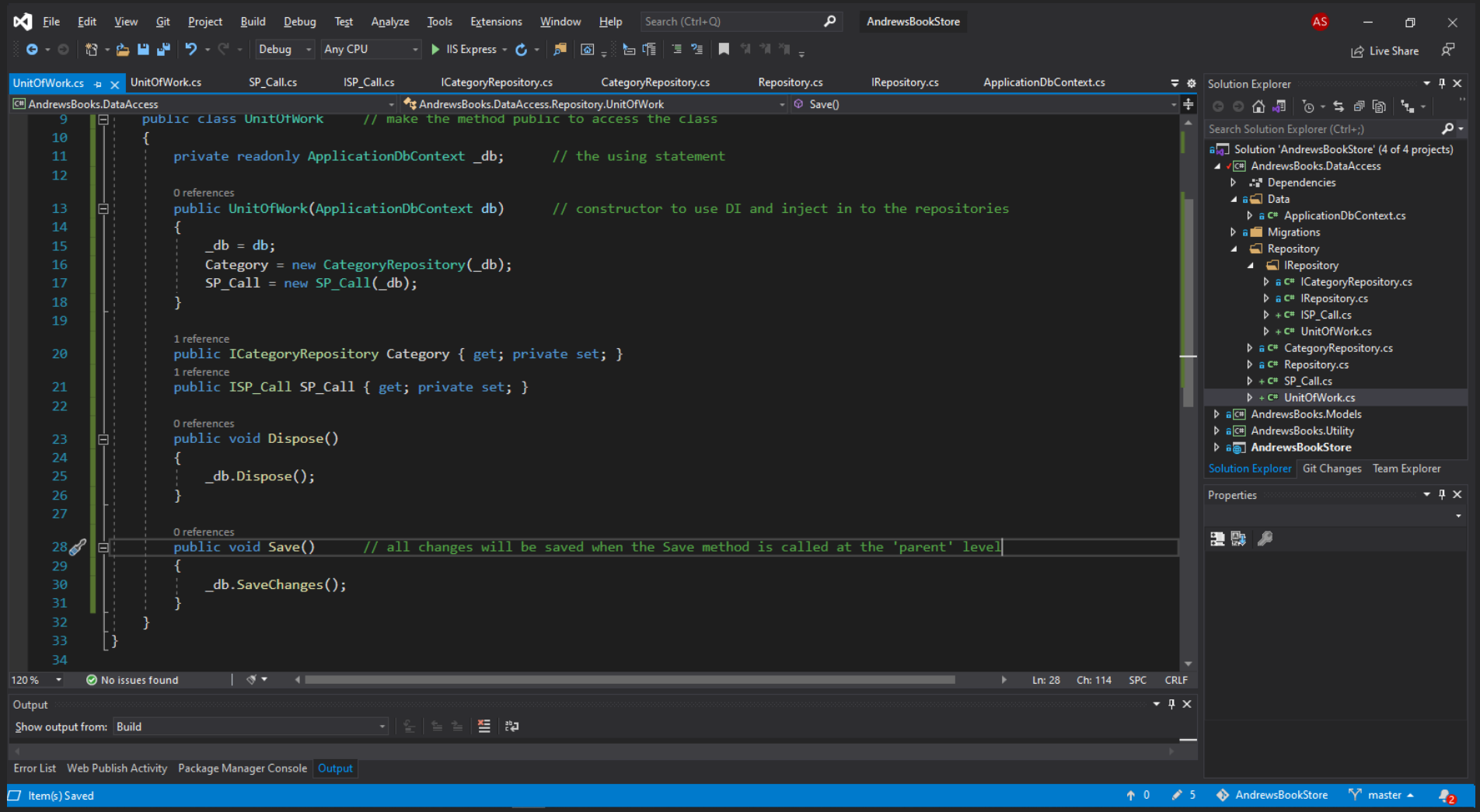
- Now implement this inside the UnitOfWork (Hint: Add a class)
- Modify the code (make sure the public class implements the interface - UnitOfWork : IUnitOfWork)



** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

- Now implement this inside the UnitOfWork (Hint: Add a class)
- Modify the code (make sure the public class implements the interface - UnitOfWork : IUnitOfWork)

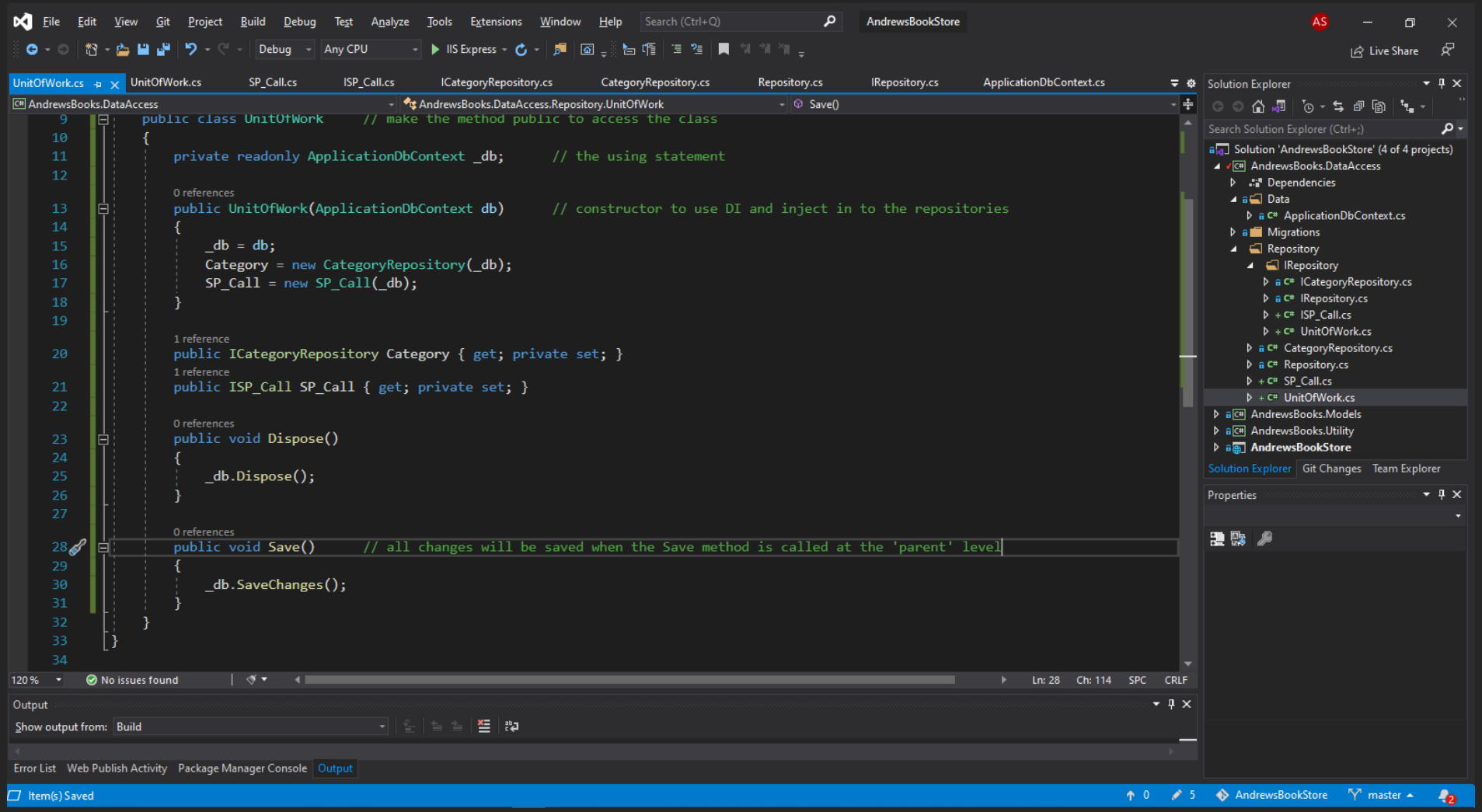


** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

- Now implement this inside the UnitOfWork (Hint: Add a class)
- Modify the code (make sure the public class implements the interface - UnitOfWork : IUnitOfWork)

- To make it accessible by the project, register it Startup.cs in the ConfigureServices method (don't forget the using statements)



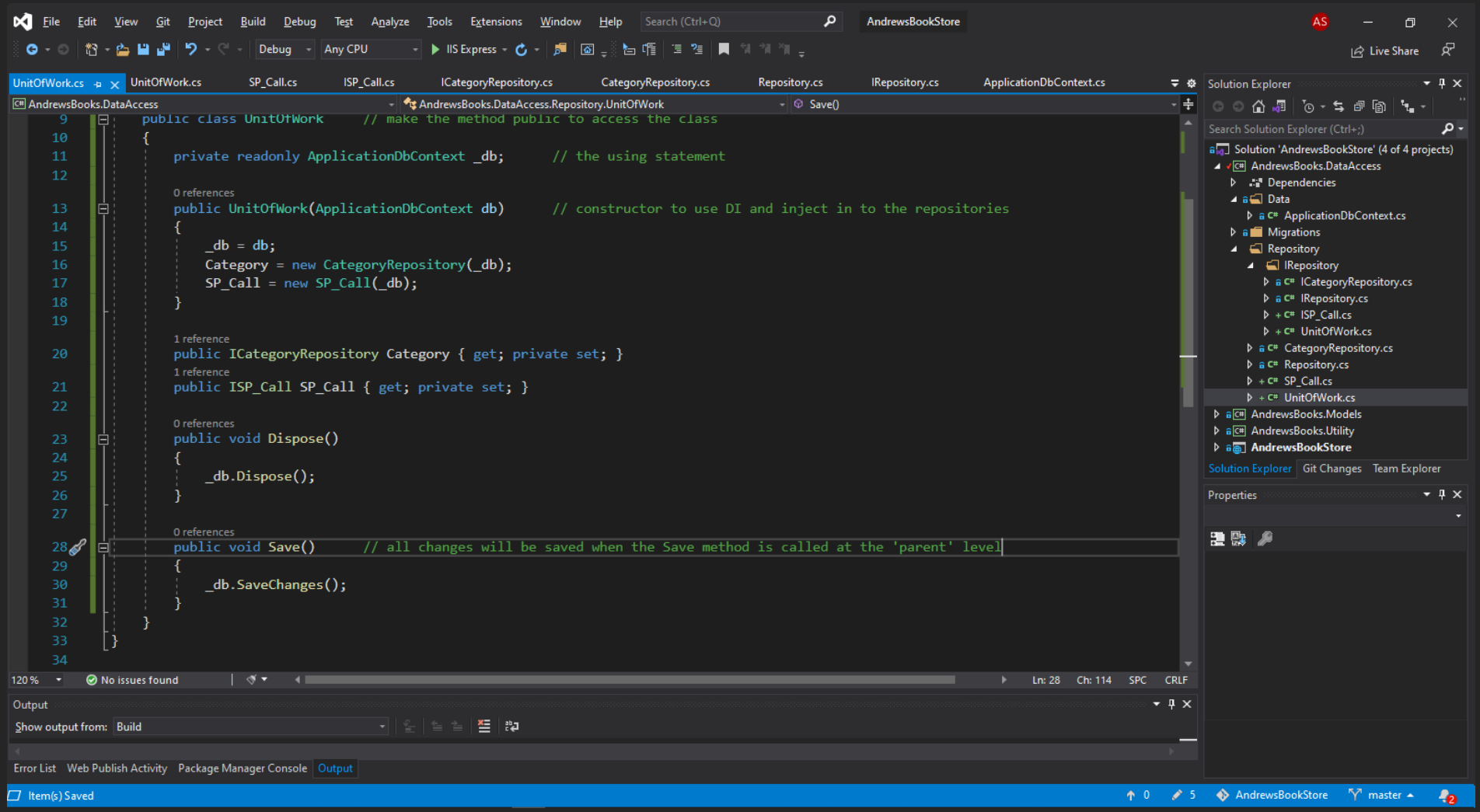
** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

- Now implement this inside the UnitOfWork (Hint: Add a class)
- Modify the code (make sure the public class implements the interface - UnitOfWork : IUnitOfWork)

- To make it accessible by the project, register it Startup.cs in the ConfigureServices method (don't forget the using statements)

- Build and correct any errors, push commits to GitHub.



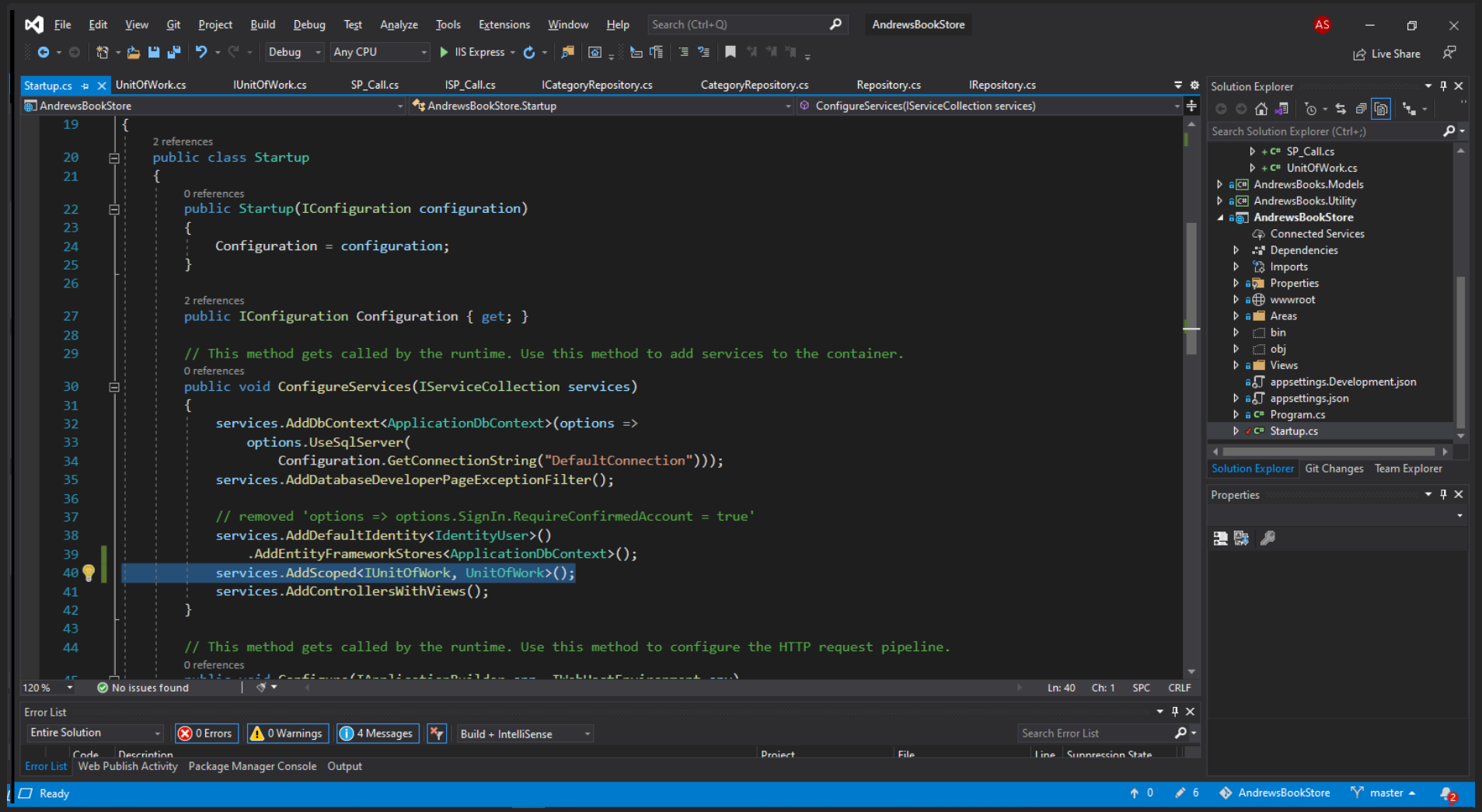
** Errata > The Interface file name is - IUnitOfWork*

- Add a new interface (IUnitOfWork) to the IRepository folder and update the code.

- Now implement this inside the UnitOfWork (Hint: Add a class)
- Modify the code (make sure the public class implements the interface - UnitOfWork : IUnitOfWork)

- To make it accessible by the project, register it Startup.cs in the ConfigureServices method (don't forget the using statements)

- Build and correct any errors, push commits to GitHub.



** Errata > The Interface file name is - IUnitOfWork*