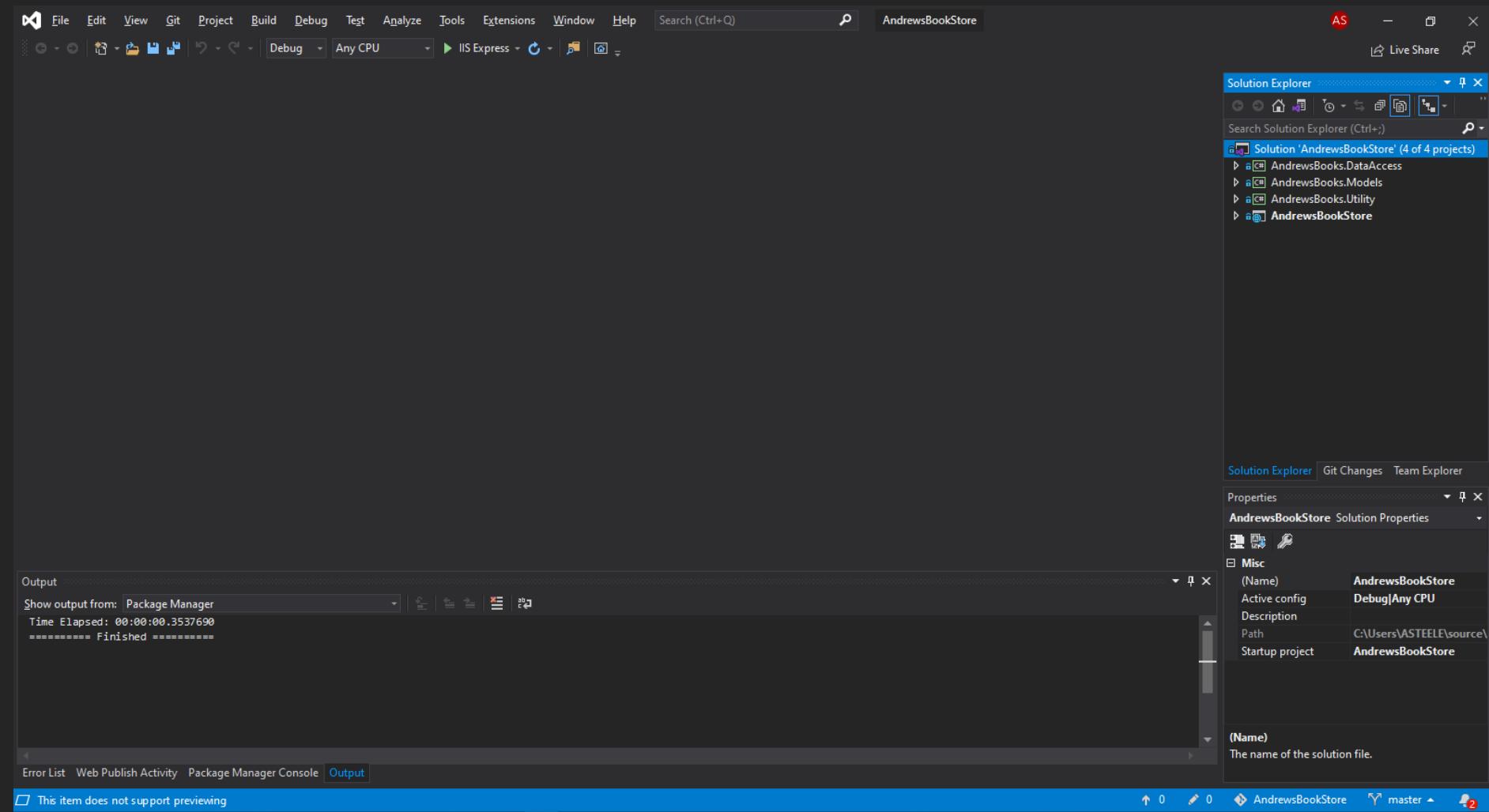
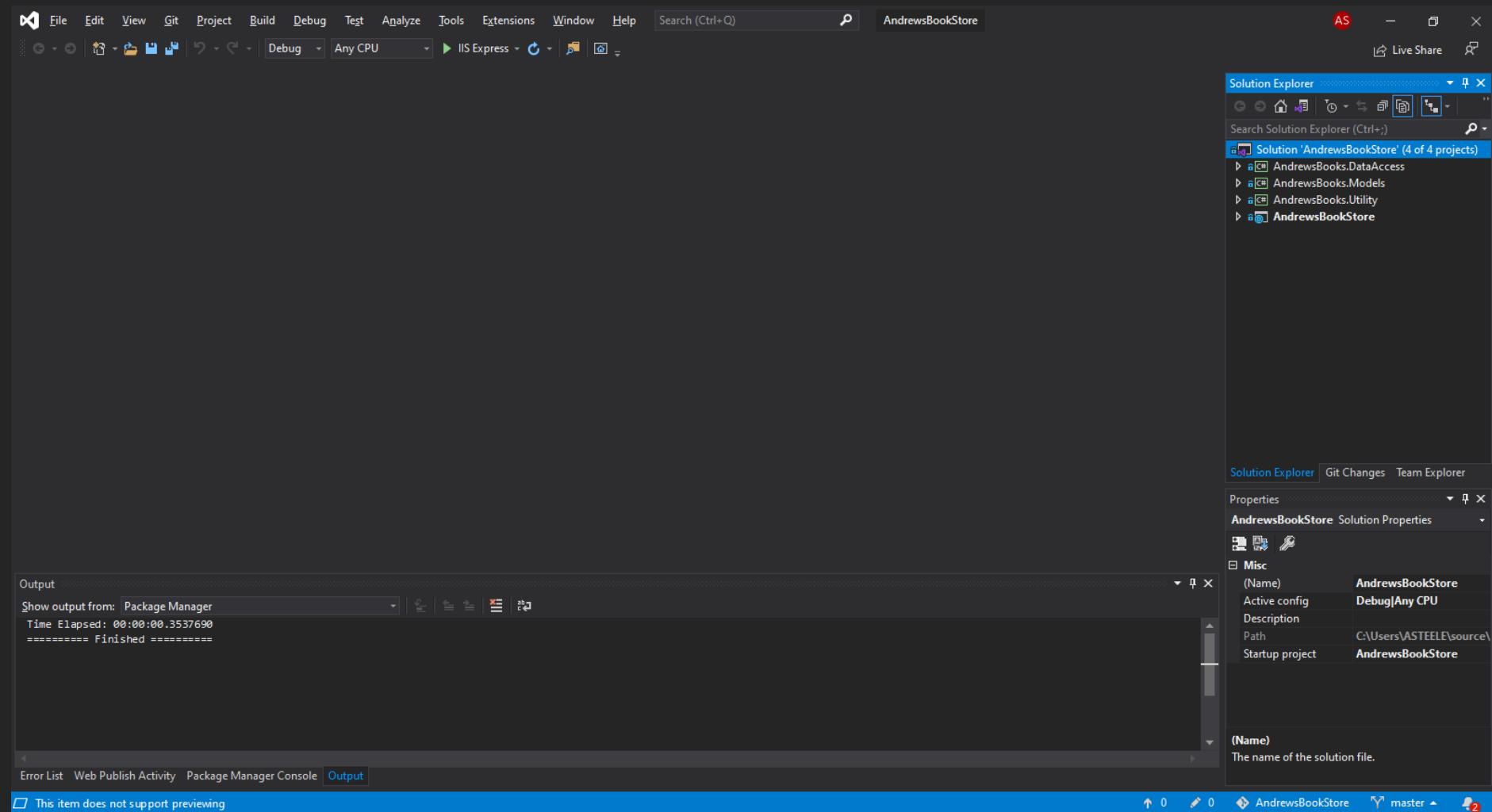


# Build the Project (Part 2)

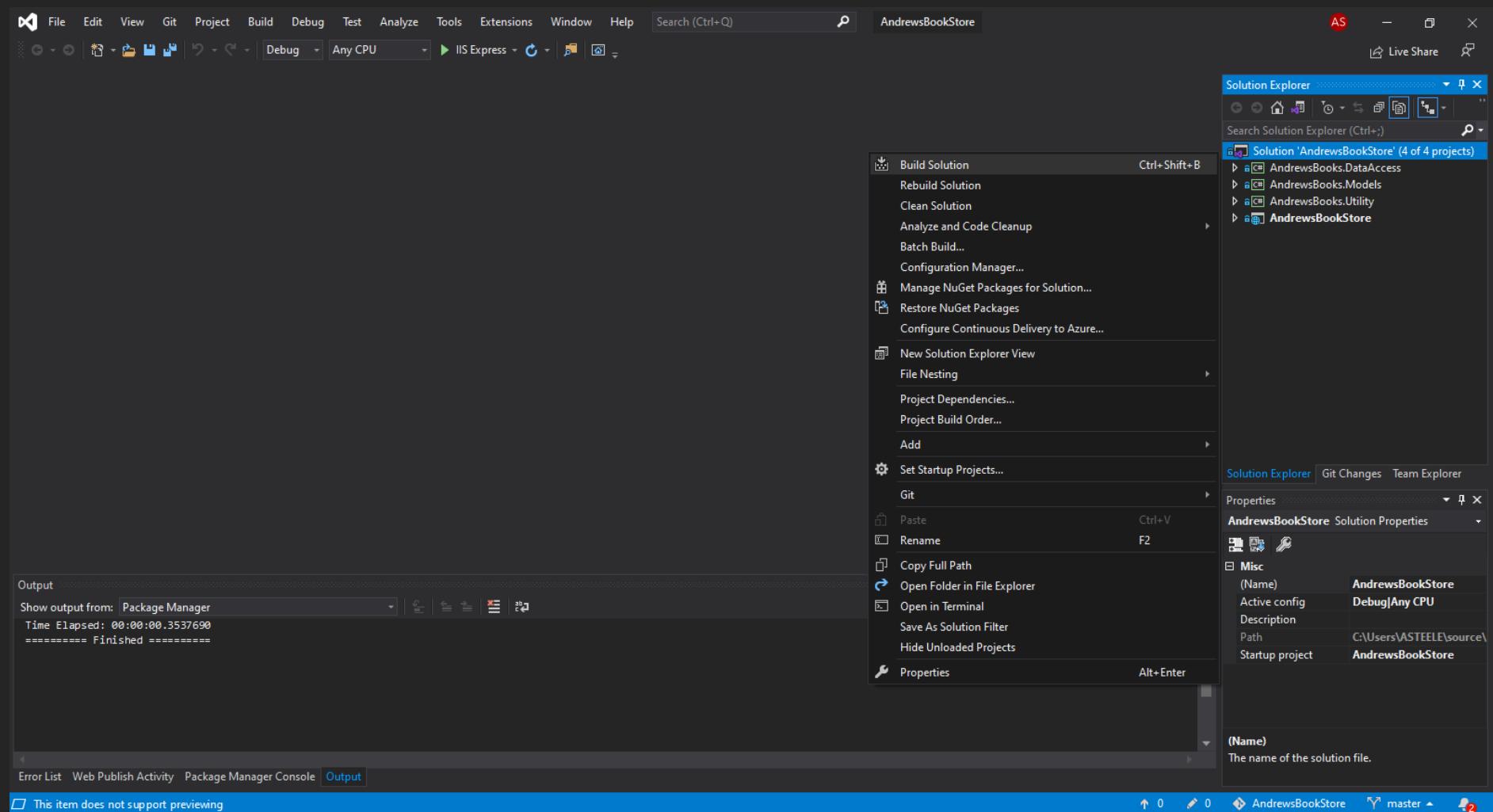
## 2.1 Create the DB



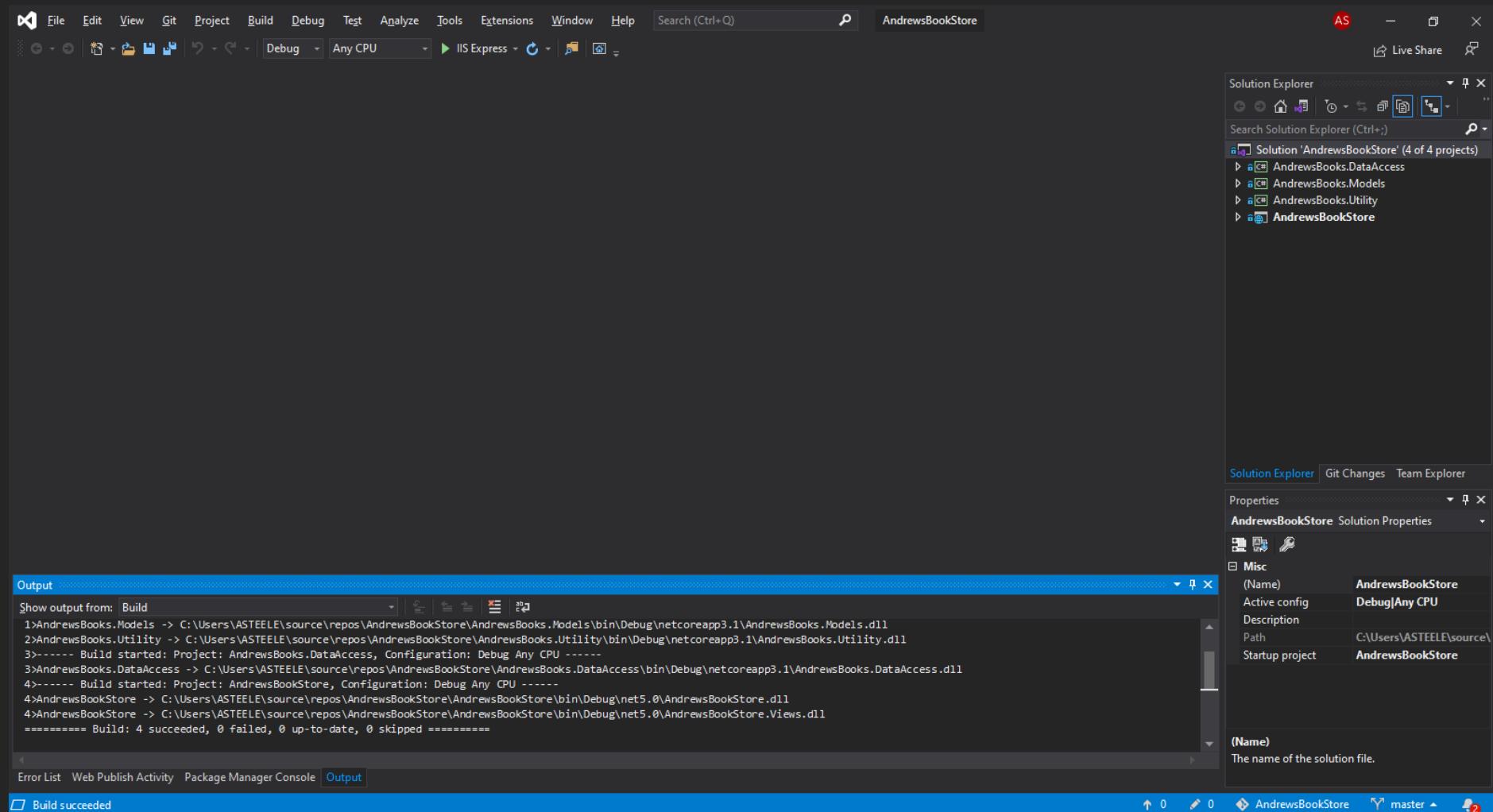
- To prepare for the next part, ‘build’ the application and confirm there are no errors



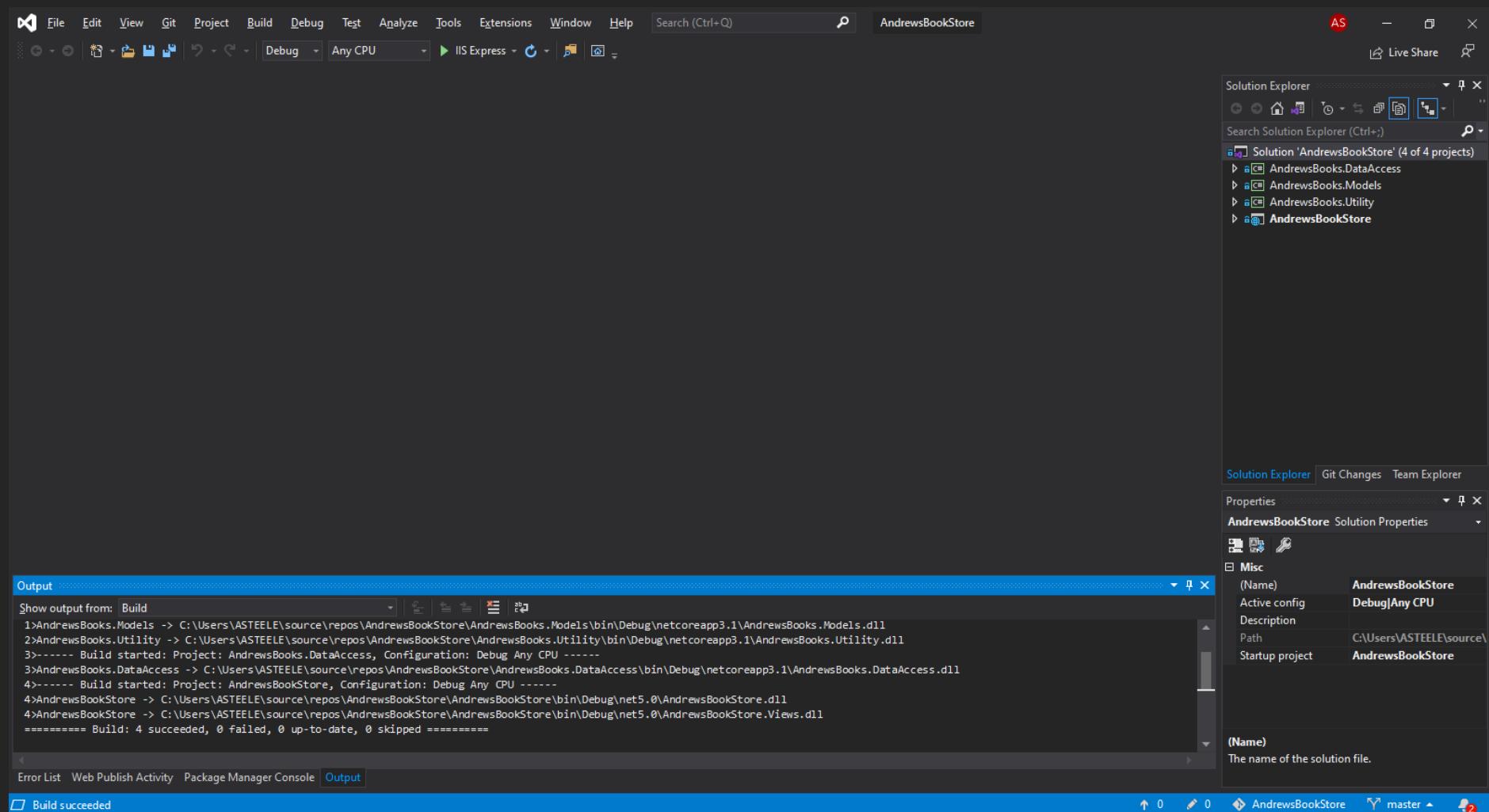
- To prepare for the next part, ‘build’ the application and confirm there are no errors



- 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

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** The current file being edited in the main editor window. It contains the following JSON configuration:

```

1  {
2    "ConnectionStrings": {
3      "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-AndrewsBookStore-4AA32BD9-959B-48DC-9E4F-00A4D9EBCA8C"
4    },
5    "Logging": {
6      "LogLevel": {
7        "Default": "Information",
8        "Microsoft": "Warning",
9        "Microsoft.Hosting.Lifetime": "Information"
10       }
11     },
12   "AllowedHosts": "*"
13 }
14

```

- Output Window:** Displays the build logs for the project. The output shows the following build steps and results:

```

1>AndrewsBooks.Models -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Models\bin\Debug\netcoreapp3.1\AndrewsBooks.Models.dll
2>AndrewsBooks.Utility -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Utility\bin\Debug\netcoreapp3.1\AndrewsBooks.Utility.dll
3>----- Build started: Project: AndrewsBooks.DataAccess, Configuration: Debug Any CPU -----
3>AndrewsBooks.DataAccess -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.DataAccess\bin\Debug\netcoreapp3.1\AndrewsBooks.DataAccess.dll
4>----- Build started: Project: AndrewsBookStore, Configuration: Debug Any CPU -----
4>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.dll
4>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.Views.dll
===== Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped ======

```

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

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** The current file being edited. The code is as follows:
 

```

1  {
2    "ConnectionStrings": {
3      "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-AndrewsBookStore-4AA32BD9-959B-48DC-9E4F-00A4D9EBCA8C"
4    },
5    "Logging": {
6      "LogLevel": {
7        "Default": "Information",
8        "Microsoft": "Warning",
9        "Microsoft.Hosting.Lifetime": "Information"
10       }
11    },
12    "AllowedHosts": "*"
13  }
14
      
```

 A red arrow points from the text "Connection string to the localdb that is automatically selected" to the highlighted connection string in the JSON.
- Output:** Shows the build logs for the project:
 

```

1>AndrewsBooks.Models -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Models\bin\Debug\netcoreapp3.1\AndrewsBooks.Models.dll
2>AndrewsBooks.Utility -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Utility\bin\Debug\netcoreapp3.1\AndrewsBooks.Utility.dll
3>----- Build started: Project: AndrewsBooks.DataAccess, Configuration: Debug Any CPU -----
3>AndrewsBooks.DataAccess -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.DataAccess\bin\Debug\netcoreapp3.1\AndrewsBooks.DataAccess.dll
4>----- Build started: Project: AndrewsBookStore, Configuration: Debug Any CPU -----
4>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.dll
4>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.Views.dll
===== Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
      
```
- Status Bar:** Shows the status "Ready".

- 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.

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** The current file being edited. The connection string 'DefaultConnection' is highlighted in yellow. A red arrow points from the text "Connection string to the localdb that is automatically selected" to this highlighted area.
- Output Window:** Displays the build logs for the project. It shows the build started for 'AndrewsBooks.DataAccess' and 'AndrewsBookStore'. The output ends with the message: "Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped".
- Status Bar:** Shows the status "Ready" and the repository information "AndrewsBookStore master 2".

```

1 {
2   "ConnectionStrings": {
3     "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-AndrewsBookStore-4AA32BD9-959B-48DC-"
4   },
5   "Logging": {
6     "LogLevel": {
7       "Default": "Information",
8       "Microsoft": "Warning",
9       "Microsoft.Hosting.Lifetime": "Information"
10    }
11  },
12  "AllowedHosts": "*"
13}
14
  
```

Connection string to the localdb  
that is automatically 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.

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** The current file being edited. It contains configuration settings for a database connection and logging. A red arrow points upwards from the bottom of the code editor towards the highlighted line: `Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"`.
- Output Window:** Displays the build logs for the project. The logs show the successful compilation of all four projects in the solution.

```

1<localdb>\\mssqllocaldb;Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"
2:
3": "Information"
4
5
6
7
8
9
10
11
12
13
14

146% No issues found
Output
Show output from: Build
1>AndrewsBooks.Models -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Models\bin\Debug\netcoreapp3.1\AndrewsBooks.Models.dll
2>AndrewsBooks.Utility -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Utility\bin\Debug\netcoreapp3.1\AndrewsBooks.Utility.dll
3>----- Build started: Project: AndrewsBooks.DataAccess, Configuration: Debug Any CPU -----
4>AndrewsBooks.DataAccess -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.DataAccess\bin\Debug\netcoreapp3.1\AndrewsBooks.DataAccess.dll
4>----- Build started: Project: AndrewsBookStore, Configuration: Debug Any CPU -----
5>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.dll
6>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.Views.dll
===== Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

```

- 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).

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** An open JSON file showing configuration settings. A red arrow points upwards from the 'Information' section towards the 'ConnectionString' entry, which is highlighted in blue.
- Output Window:** Displays the build logs for the project. The output shows the build process for four projects: AndrewsBooks.Models, AndrewsBooks.Utility, AndrewsBooks.DataAccess, and AndrewsBookStore. The logs indicate successful builds for all projects.

```

1>AndrewsBooks.Models -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Models\bin\Debug\netcoreapp3.1\AndrewsBooks.Models.dll
2>AndrewsBooks.Utility -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.Utility\bin\Debug\netcoreapp3.1\AndrewsBooks.Utility.dll
3>----- Build started: Project: AndrewsBooks.DataAccess, Configuration: Debug Any CPU -----
3>AndrewsBooks.DataAccess -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBooks.DataAccess\bin\Debug\netcoreapp3.1\AndrewsBooks.DataAccess.dll
4>----- Build started: Project: AndrewsBookStore, Configuration: Debug Any CPU -----
4>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.dll
4>AndrewsBookStore -> C:\Users\ASTEELE\source\repos\AndrewsBookStore\AndrewsBookStore\bin\Debug\net5.0\AndrewsBookStore.Views.dll
===== Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
  
```

- 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).

The screenshot shows the Visual Studio IDE interface with the following components:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** An open JSON file showing configuration settings, specifically the connection string 'AndrewsBookStore' which is set to '(localdb)\\mssqllocaldb;Database=AndrewsBookStore;Trusted\_Connection=True;MultipleActiveResultSets=true'.
- Package Manager Console:** A terminal window displaying the command 'PM> add-migration AddDefaultIdentityMigration'.

A red arrow points to the command 'add-migration AddDefaultIdentityMigration' in the Package Manager Console.

```

1
2
3   "ConnectionStrings": {
4     "AndrewsBookStore": "Data Source=(localdb)\\mssqllocaldb;Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"
5   }
6
7   "Logging": {
8     "LogLevel": {
9       "Default": "Information"
10    }
11  }
12
13
14

```

```

PM> add-migration AddDefaultIdentityMigration

```

- 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.

The screenshot shows the Visual Studio IDE interface with the following components visible:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** An open JSON file showing configuration settings. The connection string 'AndrewsBookStore' has been modified from 'Data Source=(localdb)\mssqllocaldb;Database=AndrewsBookStore;Trusted\_Connection=True;MultipleActiveResultSets=true' to 'Data Source=(localdb)\mssqllocaldb;Database=AndrewsBookStore;Trusted\_Connection=True;MultipleActiveResultSets=true'.
- Package Manager Console:** A terminal window showing the command 'PM> add-migration AddDefaultIdentityMigration'. A red arrow points to this command.
- Status Bar:** Shows the status 'Ready'.

- 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.

The screenshot shows the Microsoft Visual Studio interface with the following components:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** An open JSON file showing configuration settings, specifically the connection string 'AndrewsBookStore' which points to a localdb database.
- Package Manager Console:** A terminal window displaying the command `PM> add-migration AddDefaultIdentityMigration` and its output. The output includes a warning about Entity Framework tools version being older than the runtime, and instructions to change the target project or migration assembly.
- Properties:** A panel showing details for the 'AndrewsBookStore' project, including the file 'UserSecretsId'.

```

appsettings.json
Schema: https://json.schemastore.org/appsettings.json
1
2
3   "AndrewsBookStore": {
4     "ConnectionString": "(localdb)\\mssqllocaldb;Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"
5   }
6
7   "Logging": {
8     "LogLevel": {
9       "Default": "Information"
10      }
11    }
12
13  }

146 % No issues found
PM> add-migration AddDefaultIdentityMigration
Build started...
Build succeeded.
The Entity Framework tools version '5.0.2' is older than that of the runtime '5.0.11'. Update the tools for the latest features and bug fixes.
Your target project 'AndrewsBookStore' doesn't match your migrations assembly 'AndrewsBooks.DataAccess'. Either change your target project or change your migrations assembly.
Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connection, b => b.MigrationsAssembly("AndrewsBookStore")). By default, the migrations assembly is the assembly containing the DbContext.
Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.
PM>

```

- 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.

The screenshot shows the Visual Studio IDE interface with the following components:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- appsettings.json:** An open JSON file showing configuration settings, specifically the connection string 'AndrewsBookStore' which points to 'localdb'. The file also contains a 'Logging' section with a 'LogLevel' key set to 'Information'.
- Package Manager Console:** Displays the command 'PM> add-migration AddDefaultIdentityMigration' and its output: 'Build started...', 'Build succeeded.', and a warning message about Entity Framework tools version being older than the runtime. It also shows instructions to change the target project or migration assembly.
- Properties:** A sidebar showing the properties for the 'AndrewsBookStore' project, including the file 'UserSecretsId'.

```

1 1
2 2
3 3 (localdb)\mssqllocaldb;Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"
4 4
5 5
6 6
7 7
8 8
9 9 "Information"
10 10
11 11
12 12
13 13
  
```

```

PM> add-migration AddDefaultIdentityMigration
Build started...
Build succeeded.
The Entity Framework tools version '5.0.2' is older than that of the runtime '5.0.11'. Update the tools for the latest features and bug fixes.
Your target project 'AndrewsBookStore' doesn't match your migrations assembly 'AndrewsBooks.DataAccess'. Either change your target project or change your migrations assembly.
Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connection, b => b.MigrationsAssembly("AndrewsBookStore")). By default, the migrations assembly is the assembly containing the DbContext.
Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.
PM>
  
```

- 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.

The screenshot shows the Visual Studio IDE interface. In the center, the Package Manager Console window is open, displaying the command "PM> add-migration AddDefaultIdentityMigration" and its output: "Build started... Build succeeded." Below this, a warning message is shown in yellow: "The Entity Framework tools version '5.0.2' is older than that of the runtime '5.0.11'. Update the tools for the latest features and bug fixes." Further down, an error message in red states: "Your target project 'AndrewsBookStore' doesn't match your migrations assembly 'AndrewsBooks.DataAccess'. Either change your target project or change your migrations assembly." It provides instructions to either change the target project or the migrations assembly using DbContextOptionsBuilder.

```

appsettings.json
Schema: https://json.schemastore.org/appsettings.json
1
2
3 (localdb)\mssqllocaldb;Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"
4
5
6
7
8
9 : "Information"
10
11
12
13
146% No issues found
Package Manager Console
Package source: All Default project: AndrewsBookStore
PM> add-migration AddDefaultIdentityMigration
Build started...
Build succeeded.
The Entity Framework tools version '5.0.2' is older than that of the runtime '5.0.11'. Update the tools for the latest features and bug fixes.
Your target project 'AndrewsBookStore' doesn't match your migrations assembly 'AndrewsBooks.DataAccess'. Either change your target project or change your migrations assembly.
Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connection, b => b.MigrationsAssembly("AndrewsBookStore")). By default, the migrations assembly is the assembly containing the DbContext.
Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.
PM>

```

- 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.

The screenshot shows the Visual Studio IDE interface. In the center, the Package Manager Console window is open, displaying the command "PM> add-migration AddDefaultIdentityMigration" and its output: "Build started...", "Build succeeded.", and a warning message about Entity Framework tools version being older than the runtime. Above the console, the appsettings.json file is shown with a connection string configuration. To the right, the Solution Explorer shows a solution named 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The Properties and Git Changes tabs are also visible.

```

appsettings.json
Schema: https://json.schemastore.org/appsettings.json
1
2
3 (localdb)\mssqllocaldb;Database=AndrewsBookStore;Trusted_Connection=True;MultipleActiveResultSets=true"
4
5
6
7
8
9 "Information"
10
11
12
13
146% No issues found
Package Manager Console
Package source: All Default project: AndrewsBookStore
PM> add-migration AddDefaultIdentityMigration
Build started...
Build succeeded.
The Entity Framework tools version '5.0.2' is older than that of the runtime '5.0.11'. Update the tools for the latest features and bug fixes.
Your target project 'AndrewsBookStore' doesn't match your migrations assembly 'AndrewsBooks.DataAccess'. Either change your target project or change your migrations assembly.
Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connection, b => b.MigrationsAssembly("AndrewsBookStore")). By default, the migrations assembly is the assembly containing the DbContext.
Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.
PM>

```

- 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.
- 
- The screenshot shows the Visual Studio IDE interface. The main window displays the code for a migration class named 'AddDefaultIdentityMigration' in the 'AndrewsBooks.DataAccess' project. The code defines a table 'AspNetRoles' with columns for Id, Name, NormalizedName, and ConcurrencyStamp. The 'Package Manager Console' window at the bottom shows the command 'PM>' and the message 'Type 'get-help NuGet' to see all available NuGet commands.' The 'Solution Explorer' pane on the right lists four projects: AndrewsBooks.DataAccess, AndrewsBooks.Data, AndrewsBooks.Migrations, and AndrewsBookStore. The 'Properties' pane for the migration file is open, showing details like 'File Name' and 'Build Action'.
- ```

public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}

```

- 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.
- 
- The screenshot shows the Visual Studio IDE interface. The code editor displays a C# migration class named 'AddDefaultIdentityMigration' from the 'AndrewsBooks.DataAccess.Migrations' namespace. The class defines a 'Up' method that creates a 'AspNetRoles' table with columns for Id, Name, NormalizedName, and ConcurrencyStamp. The 'Package Manager Console' window at the bottom shows the NuGet Package Manager Host version 5.8.1.7021, with the command 'PM>' and the message 'Type 'get-help NuGet' to see all available NuGet commands.' The 'Solution Explorer' pane on the right lists four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The 'migrations' folder under AndrewsBooks.DataAccess contains a file named '2021102211202\_AddDefaultIdentityMigration.cs'. The 'Properties' pane for this file shows 'Build Action' set to 'C# compiler' and 'File Name' set to '2021102211202\_AddDefaultIdentityMigration.cs'. The status bar at the bottom indicates the solution name 'AndrewsBookStore' and the current branch 'master'.

Screenshot of Microsoft Visual Studio 2019 showing the code editor, Solution Explorer, Properties, and Package Manager Console.

**Code Editor:** The main window displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# code for a database migration. The code creates a table named `AspNetRoles` with columns for Id, Name, NormalizedName, and ConcurrencyStamp.

```
public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}
```

**Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the migration file.

**Properties:** The properties for the selected migration file (`2021102211202_AddDefaultIdentityMigration.cs`) are displayed, including Build Action (C# compiler), Copy to Output Direct, and File Name.

**Package Manager Console:** The console shows the version `5.8.1.7021` and the command prompt `PM>`.

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

The screenshot shows a Microsoft Visual Studio interface with the following components:

- Code Editor:** Displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# code for a database migration. The code defines a partial class `AddDefaultIdentityMigration` that inherits from `Migration`. It uses `MigrationBuilder` to create a table named `AspNetRoles` with columns for `Id`, `Name`, `Normalized Name`, and `ConcurrencyStamp`. A primary key constraint is added to the `Id` column.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the migration file.
- Properties:** Panel on the right showing build action details for the migration file.
- Package Manager Console:** Shows the console host version 5.8.1.7021 and a prompt for commands like 'get-help NuGet'.
- Status Bar:** Shows the repository name 'AndrewsBookStore' and branch 'master'.

```

public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}

```

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

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Code Editor:** Displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# code for an ASP.NET Identity migration. The code defines a partial class `AddDefaultIdentityMigration` that inherits from `Migration`. It uses `MigrationBuilder` to create a table named `AspNetRoles` with columns for `Id`, `Name`, `Normalized Name`, and `ConcurrencyStamp`. A primary key constraint is added to the `Id` column.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the migration file shown in the code editor.
- Properties:** The properties for the migration file are visible, including the `Build Action` set to `C# compiler` and the `File Name` set to `2021102211202_AddDefaultIdentityMigration.cs`.
- Package Manager Console:** Shows the console output for the Package Manager Host Version 5.8.1.7021. The message "Type 'get-help NuGet' to see all available NuGet commands." is displayed.
- Status Bar:** Shows the status bar with "Ready" and other build-related information.

- 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.

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** AndrewsBookStore
- Solution Explorer:** Shows the solution 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Properties Window:** The 'Advanced' section is open for the file '2021102211202\_AddDefaultIdentityMigration.cs'. It shows the 'File Name' as '2021102211202\_AddDefaultIdentityMigration.cs' and the 'Full Path' as 'C:\Users\ASTEELE\source\repos\AndrewsBookStore\Migrations\2021102211202\_AddDefaultIdentityMigration.cs'.
- Code Editor:** The file '2021102211202\_AddDefaultIdentityMigration.cs' is open, displaying C# code for creating an AspNetRoles table. The code defines a partial class 'AddDefaultIdentityMigration' that inherits from 'Migration'. It uses 'migrationBuilder.CreateTable' to define the table structure, including columns for Id, Name, NormalizedName, and ConcurrencyStamp. It also sets the PrimaryKey for the table.
- Package Manager Console:** The console shows the message 'Type 'get-help NuGet' to see all available NuGet commands.' and the prompt 'PM>'.
- Status Bar:** Shows the status 'Ready'.

```

public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}

```

- 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.

The screenshot shows a Microsoft Visual Studio interface with the following components:

- Code Editor:** Displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# code for an Entity Framework migration. The code creates a table named `AspNetRoles` with columns for Id, Name, NormalizedName, and ConcurrencyStamp, and adds a primary key constraint named `PK_AspNetRoles`.
- Solution Explorer:** Shows the solution structure for `'AndrewsBookStore' (4 of 4 projects)`, including `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder contains the generated migration file.
- Properties:** A floating window showing details for the selected migration file, including the build action as `C# compiler`.
- Package Manager Console:** Shows the command `PM>` and the message `Type 'get-help NuGet' to see all available NuGet commands.`
- Bottom Status Bar:** Shows the status `Ready` and other system information.

- 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.

The screenshot shows a Microsoft Visual Studio interface with the following components:

- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** Displays the file '20211102211202\_AddDefaultIdentityMigration.cs' containing C# code for creating an AspNetRoles table.
- Package Manager Console:** Shows the command 'PM> update-database' being run, followed by build logs: 'Build started...', 'Build succeeded.', and 'Done.'

```

public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}

```

- 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.

The screenshot shows the Microsoft Visual Studio interface with the following components:

- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** Displays the file '20211102211202\_AddDefaultIdentityMigration.cs' which contains C# code for creating an AspNetRoles table.
- Package Manager Console:** Shows the command 'PM> update-database' being run, followed by the output: 'Build started...', 'Build succeeded.', and 'Done.'.

```

public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}

```

- 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.

The screenshot shows the Visual Studio IDE interface with the following components:

- SQL Server Object Explorer:** On the left, it shows the database structure under "SQL Server" > "(localdb)\MSSQLLocalDB (SQL Server 13.0.4)". The "Tables" node under the "AndrewsBookStore" database is selected.
- Solution Explorer:** On the right, it shows the solution structure for "AndrewsBookStore" with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The central area displays the code for a migration named "20211102211202\_AddDefaultIdentityMigration.cs". The code defines a migration class "AddDefaultIdentityMigration" that creates tables for AspNetUsers, AspNetRoles, and AspNetUserTokens.

```

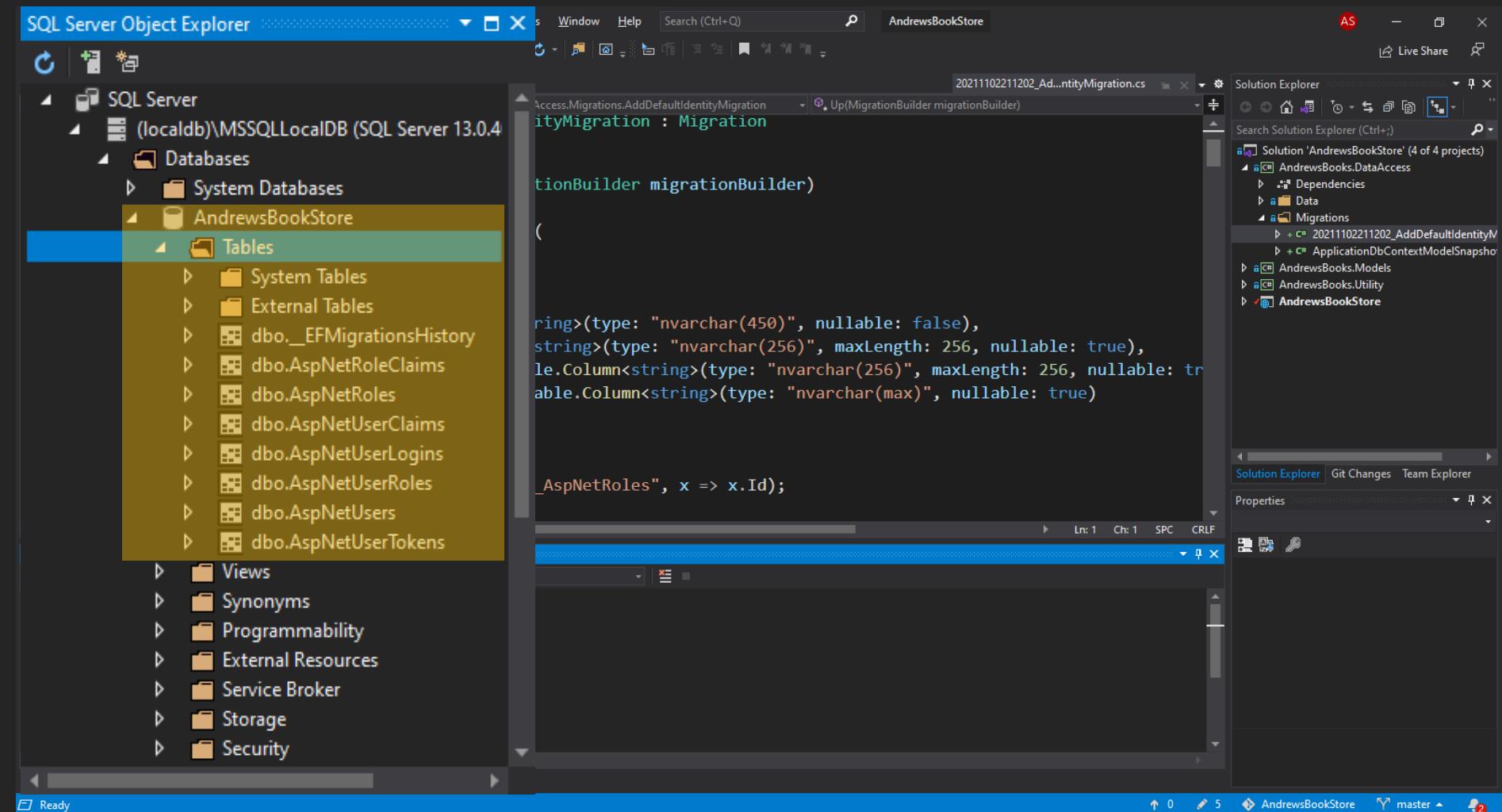
using Microsoft.EntityFrameworkCore.Migrations;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace AndrewsBooks.DataAccess.Migrations
{
    public partial class AddDefaultIdentityMigration : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AspNetUsers",
                columns: table =>
                {
                    table.Column<string>(type: "nvarchar(450)", nullable: false),
                    table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUsers", x => x.Id);
                    table.ForeignKey(
                        name: "FK_AspNetUsers_AspNetRoles",
                        column: x => x.Id,
                        principalTable: "AspNetRoles",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table =>
                {
                    table.Column<string>(type: "nvarchar(450)", nullable: false),
                    table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                });
            migrationBuilder.CreateTable(
                name: "AspNetUserTokens",
                columns: table =>
                {
                    table.Column<string>(type: "nvarchar(450)", nullable: false),
                    table.Column<string>(type: "nvarchar(450)", nullable: false),
                    table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserTokens", x => x.UserId);
                    table.ForeignKey(
                        name: "FK_AspNetUserTokens_AspNetUsers",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "AspNetUsers");
            migrationBuilder.DropTable(
                name: "AspNetRoles");
            migrationBuilder.DropTable(
                name: "AspNetUserTokens");
        }
    }
}

```

- 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.



The screenshot shows the Visual Studio IDE interface with the following components:

- SQL Server Object Explorer:** On the left, it shows the database structure under "SQL Server" > "(localdb)\MSSQLLocalDB (SQL Server 13.0.4)". The "Tables" node for the "AndrewsBookStore" database is selected, displaying various system and application tables like `System Tables`, `External Tables`, and several `dbo.AspNet\*` tables.
- Solution Explorer:** On the right, it shows a solution named "AndrewsBookStore" containing four projects: "AndrewsBooks.DataAccess", "AndrewsBooks.Models", "AndrewsBooks.Utility", and "AndrewsBookStore". The "Migrations" folder under "AndrewsBookStore" contains a file named "20211102211202\_AddDefaultIdentityMigration.cs".
- Code Editor:** The main window displays the C# code for the migration class:

```

public partial class AddDefaultIdentityMigration : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "AspNetRoles",
            columns: table =>
            {
                table.Column<string>(type: "nvarchar(450)", nullable: false),
                table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: constraint =>
            {
                constraint.PrimaryKey("PK_AspNetRoles", x => x.Id);
            }
        );
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "AspNetRoles");
    }
}

```

- 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

The screenshot shows the Visual Studio IDE interface with the following components:

- SQL Server Object Explorer:** On the left, it shows the database structure for the localdb\MSQLLocalDB instance. The **AndrewsBookStore** database is selected, revealing its **Tables**, **Views**, **Synonyms**, **Programmability**, **External Resources**, **Service Broker**, **Storage**, and **Security** objects.
- Solution Explorer:** On the right, it displays the solution structure for 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The central area shows the code for a migration named `20211102211202_AddDefaultIdentityMigration.cs`. The code defines a `Up(MigrationBuilder migrationBuilder)` method that adds a new column `NormalizedUserName` to the `AspNetRoles` table. The column is defined with a type of `nvarchar(450)`, `nullable: false`.

```

using Microsoft.EntityFrameworkCore.Migrations;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace Access.Migrations
{
    public partial class AddDefaultIdentityMigration : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<string>(type: "nvarchar(450)", nullable: false, maxLength: 256, defaultValue: "");
            migrationBuilder.AddColumn<string>(type: "nvarchar(256)", maxLength: 256, nullable: true);
            migrationBuilder.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true);
            migrationBuilder.Column<string>(type: "nvarchar(max)", nullable: true);

            migrationBuilder.CreateIndex(
                name: "IX_AspNetRoles_NormalizedUserName",
                table: "AspNetRoles",
                column: "NormalizedUserName");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropIndex(
                name: "IX_AspNetRoles_NormalizedUserName",
                table: "AspNetRoles");
        }
    }
}

```

Screenshot of Microsoft Visual Studio 2019 showing the code editor, Solution Explorer, Properties, and Package Manager Console.

**Code Editor:** The main window displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# code for a database migration. The code creates a table named `AspNetRoles` with columns for Id, Name, NormalizedName, and ConcurrencyStamp.

```
public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}
```

**Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the migration file.

**Properties:** The properties for the selected migration file (`2021102211202_AddDefaultIdentityMigration.cs`) are displayed, including Build Action (C# compiler), Copy to Output Direct, and File Name.

**Package Manager Console:** The console shows the version `5.8.1.7021` and the command prompt `PM>`.

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

The screenshot shows a Microsoft Visual Studio interface with the following components:

- Code Editor:** Displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# migration code for creating an `AspNetRoles` table.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the migration file.
- Properties:** Opened for the migration file, showing build actions and other properties.
- Package Manager Console:** Displays the command prompt interface for managing NuGet packages.
- Status Bar:** Shows the status "Ready" at the bottom left.

```

public partial class AddDefaultIdentityMigration : Migration
{
    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: true),
                ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_AspNetRoles", x => x.Id);
            });
    }
}

```

- 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

The screenshot shows a Microsoft Visual Studio interface with the following components:

- Code Editor:** Displays the file `2021102211202_AddDefaultIdentityMigration.cs` containing C# code for a database migration. The code defines a partial class `AddDefaultIdentityMigration` that inherits from `Migration`. It overrides the `Up` method to create a table named `AspNetRoles` with columns for `Id`, `Name`, `Normalized Name`, and `ConcurrencyStamp`.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the migration file.
- Properties:** A floating window showing properties for the selected migration file, including `Build Action: C# compiler`, `Copy to Output Direct`, and `Custom Tool`.
- Package Manager Console:** Shows the console output for the Package Manager Host Version 5.8.1.7021. It displays the message: "Type 'get-help NuGet' to see all available NuGet commands." and "PM>".

- 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

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** AndrewsBookStore
- Toolbars:** Standard (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help) and Search (Ctrl+Q).
- Code Editor:** The main window displays the `Category.cs` file from the `AndrewsBooks.Models` project. The code defines a `Category` class with properties `Id` and `Name`, annotated with `[Key]`, `[Display(Name="Category Name")]`, and `[Required]`.
- Solution Explorer:** Shows the solution structure with four projects: `AndrewsBookStore` (selected), `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, and `AndrewsBooks.Utility`. The `Category.cs` file is highlighted under `AndrewsBooks.Models`.
- Properties:** A panel on the right showing properties for the selected item.
- Task List:** A panel at the bottom showing tasks like "Ready".
- Status Bar:** Shows the current branch as "master".

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Text;
5
6  namespace AndrewsBooks.Models
7  {
8      public class Category
9      {
10          [Key]
11          public int Id { get; set; }
12
13          [Display(Name="Category Name")]
14          [Required]
15          [MaxLength(50)]
16          public string Name { get; set; }
17      }
18  }
19

```

- 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 screenshot shows a Visual Studio interface with the following details:

- Title Bar:** AndrewsBookStore
- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Solution Explorer:** Shows the solution structure: AndrewsBooks.Models.csproj (selected), AndrewsBooks.DataAccess, AndrewsBooks.Models, Dependencies, ViewModels, Category.cs, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The file Category.cs is open, displaying the following C# code:
 

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Text;
5
6  namespace AndrewsBooks.Models
7  {
8      public class Category
9      {
10          [Key]
11          public int Id { get; set; }
12
13          [Display(Name = "Category Name")]
14          [Required]
15          [MaxLength(50)]
16          public string Name { get; set; }
17      }
18  }
      
```
- Properties:** Shows the project properties for 'Misc'.
- Package Manager Console:** Shows the command 'Default project: AndrewsBookStore'.
- Status Bar:** Shows 'Ready'.

- 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 screenshot shows a Visual Studio interface with the following components:

- Code Editor:** Displays the `Category.cs` file from the `AndrewsBooks.Models` project. The code defines a `Category` class with an `Id` primary key and a `Name` property with validation attributes: `[Required]` and `[MaxLength(50)]`.
- Solution Explorer:** Shows the solution structure with four projects: `AndrewsBookStore`, `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, and `AndrewsBooks.Utility`. The `Category.cs` file is selected in the `AndrewsBooks.Models` project.
- Package Manager Console:** Shows the command `PM> add-migration AddCategoryToDb` entered in the console.
- Status Bar:** Shows the status "Ready" at the bottom left.

- 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

The screenshot shows the Visual Studio IDE interface with the following components visible:

- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with projects like AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** Displays the 'Category.cs' file from the 'AndrewsBooks.Models' project. The code defines a 'Category' class with an Id primary key and a Name string property with validation attributes [Display(Name="Category Name")], [Required], and [MaxLength(50)].
- Package Manager Console:** Shows the command 'PM> add-migration AddCategoryToDb' entered in the console.
- Status Bar:** Shows the status 'Ready' at the bottom left.

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Text;
5
6  namespace AndrewsBooks.Models
7  {
8      public class Category
9      {
10          [Key]
11          public int Id { get; set; }
12
13          [Display(Name="Category Name")]
14          [Required]
15          [MaxLength(50)]
    
```

- 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

The screenshot shows a Visual Studio interface with the following components:

- Code Editor:** Displays the file `20211103005920_AddCategoryToDb.cs` containing C# code for a migration. The code defines a partial class `AddCategoryToDb` that inherits from `Migration`. It contains two methods: `Up` and `Down`, both of which take a `MigrationBuilder` parameter.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Migrations` folder under `AndrewsBooks.DataAccess` contains the generated migration files.
- Package Manager Console:** Shows the command `PM>` indicating the current context is the package manager console.

```

using Microsoft.EntityFrameworkCore.Migrations;

namespace AndrewsBooks.DataAccess.Migrations
{
    public partial class AddCategoryToDb : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
        }
    }
}

```

- 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

The screenshot shows a Visual Studio interface with the following details:

- Title Bar:** AndrewsBookStore
- Code Editor:** The main window displays the file `20211103005920_AddCategoryToDb.cs`. The code is as follows:

```

1  using Microsoft.EntityFrameworkCore.Migrations;
2
3  namespace AndrewsBooks.DataAccess.Migrations
4  {
5      public partial class AddCategoryToDb : Migration
6      {
7          protected override void Up(MigrationBuilder migrationBuilder)
8          {
9          }
10
11         protected override void Down(MigrationBuilder migrationBuilder)
12         {
13         }
14     }
15 }
16
17 }
```

- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`.
- Package Manager Console:** The bottom window is titled "Package Manager Console" and shows the command "PM>".

- 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

The screenshot shows a Visual Studio interface with the following components:

- Code Editor:** Displays the `ApplicationDbContext.cs` file from the `AndrewsBooks.DataAccess` project. The code defines a `public DbSet<Category> Categories { get; set; }`.
- Solution Explorer:** Shows the solution structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project contains `Category.cs` and `ApplicationDbContext.cs`. The `AndrewsBooks.Models` project contains `Category.cs`. The `AndrewsBooks.Utility` project is empty.
- Package Manager Console:** Shows the command `PM> add-migration AddCategoryToDb` being run.

- 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

The screenshot shows a Visual Studio interface with the following components:

- Code Editor:** Displays the `ApplicationDbContext.cs` file. A red arrow points to the first line of the `using` statements.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBookStore.DataAccess`, `AndrewsBooks.Models`, and `AndrewsBookStore`. The `Migration` folder under `AndrewsBooks.DataAccess` contains files like `20211102211202_AddDefaultIdentityM.cs` and `20211103005920_AddCategoryToDb.cs`.
- Package Manager Console:** Shows the command `PM> add-migration AddCategoryToDb` entered.

```

1  using AndrewsBooks.Models;
2  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBookStore.DataAccess.Data
9  {
10    public class ApplicationDbContext : IdentityDbContext
11    {
12      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
13        : base(options)
14      {
15      }
16      public DbSet<Category> Categories { get; set; }
17    }
18  }

```

- 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).

The screenshot shows a Visual Studio interface with the following components:

- Code Editor:** Displays the `ApplicationDbContext.cs` file. A red arrow points to the `using Microsoft.AspNetCore.Identity.EntityFrameworkCore;` line.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBookStore.DataAccess`, `AndrewsBooks.Models`, and `AndrewsBookStore`. The `Migration` folder contains files like `20211102211202_AddDefaultIdentityM.cs` and `20211103005920_AddCategoryToDb.cs`.
- Package Manager Console:** Shows the command `PM> add-migration AddCategoryToDb` being run.

```

1  using AndrewsBooks.Models;
2  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBookStore.DataAccess.Data
9  {
10    public class ApplicationDbContext : IdentityDbContext
11    {
12      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
13        : base(options)
14      {
15      }
16      public DbSet<Category> Categories { get; set; }
17    }
18  }

```

- 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

The screenshot shows a Visual Studio interface with the following components:

- Solution Explorer:** Shows the solution structure with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** Displays the `ApplicationDbContext.cs` file from the `AndrewsBooks.DataAccess` project. A red arrow points to the `using Microsoft.AspNetCore.Identity.EntityFrameworkCore;` line. The code defines the `ApplicationContext` class which inherits from `IdentityDbContext` and includes a `Categories` DbSet.
- Package Manager Console:** Shows the command `PM> add-migration AddCategoryToDb` entered in the 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
- 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

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** Displays the file '20211103011613\_AddCategoryToDb.cs' containing C# migration code for creating a 'Categories' table.
- Package Manager Console:** Shows the output of a build command, indicating a successful build of the 'AndrewsBooks.DataAccess' project.

```

using Microsoft.EntityFrameworkCore.Migrations;

namespace AndrewsBooks.DataAccess.Migrations
{
    public partial class AddCategoryToDb : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Categories",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Categories", x => x.Id);
                }
            );
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(name: "Categories");
        }
    }
}

```

# 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.

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** AndrewsBookStore
- Toolbars:** Standard (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help) and Search (Ctrl+Q).
- Status Bar:** AndrewsBookStore master ▾ Ready
- Solution Explorer:** Shows the solution structure with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Properties:** Standard properties window.
- Code Editor:** The file `ApplicationDbContext.cs` is open, showing the following code:

```
1  using AndrewsBooks.Models;
2  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBookStore.DataAccess.Data
9  {
10    public class ApplicationDbContext : IdentityDbContext
11    {
12      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
13      : base(options)
14      {
15      }
16      public DbSet<Category> Categories { get; set; }
17    }
18  }
```

The code editor includes standard navigation and search features like Find, Replace, and Go To Definition.

- 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.

The screenshot shows the Visual Studio IDE interface with the following details:

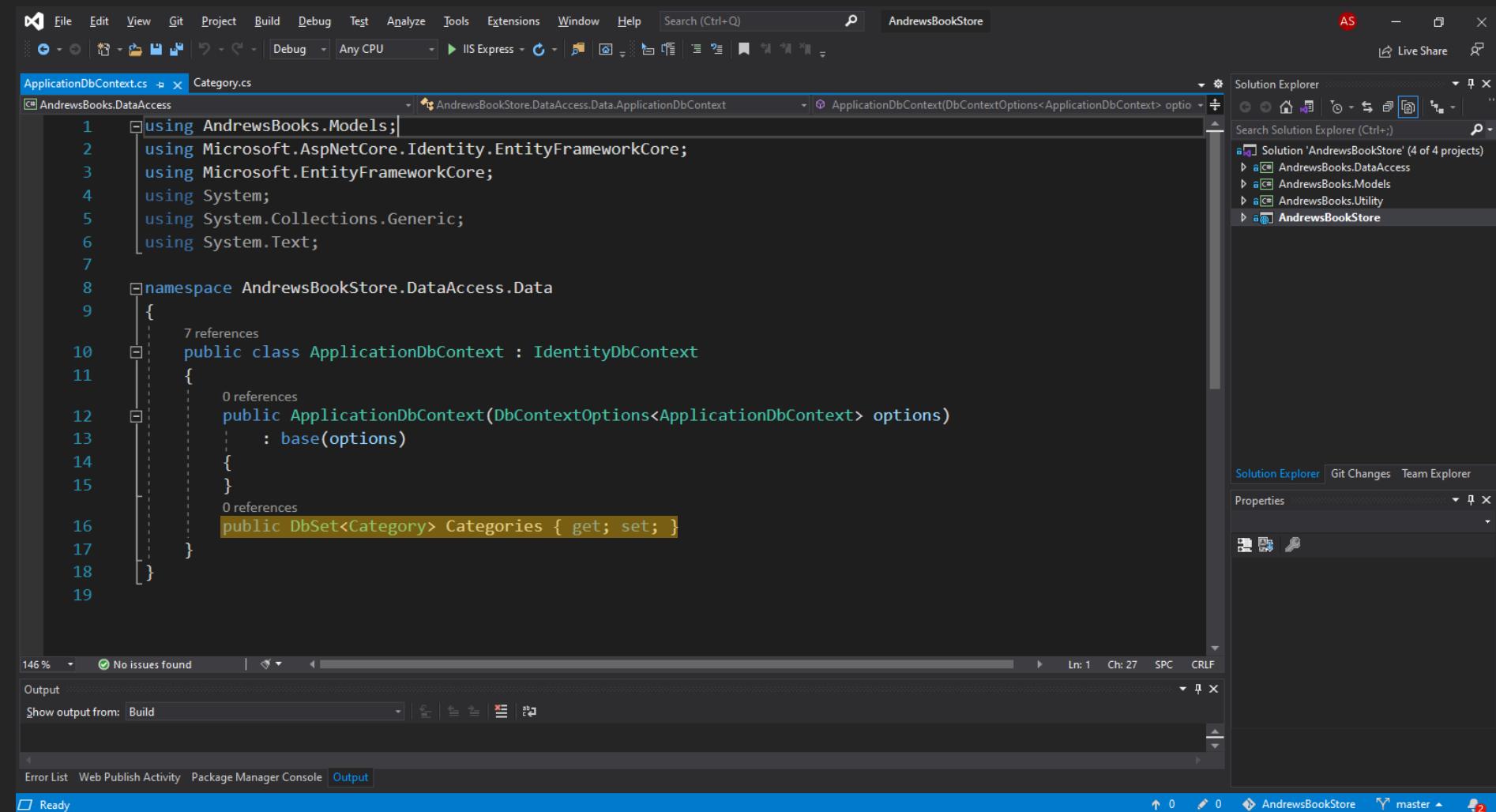
- Title Bar:** AndrewsBookStore
- Toolbars:** Standard (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help) and Search (Ctrl+Q).
- Status Bar:** AndrewsBookStore master ▾ Ready
- Solution Explorer:** Shows the solution structure with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Properties:** Shows icons for file, folder, and properties.
- Code Editor:** The file `ApplicationDbContext.cs` is open, showing the following code:

```
1  using AndrewsBooks.Models;
2  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBookStore.DataAccess.Data
9  {
10    public class ApplicationDbContext : IdentityDbContext
11    {
12      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
13      : base(options)
14      {
15      }
16      public DbSet<Category> Categories { get; set; }
17    }
18  }
```

The code editor includes a status bar at the bottom with zoom level (146%), issues found (No issues found), and line/column numbers (Ln: 1 Ch: 27 SPC CRLF). Below the code editor are the Output, Error List, Web Publish Activity, Package Manager Console, and Output tabs.

- 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.



The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** AndrewsBookStore
- Toolbars:** Standard toolbar with icons for file operations, search, and navigation.
- MenuStrip:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q).
- Toolbox:** Standard toolbox with icons for various development tools.
- Solution Explorer:** Shows the solution structure with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Properties:** Properties panel showing file properties.
- Code Editor:** The main window displays the `ApplicationDbContext.cs` file. The code defines the `ApplicationDbContext` class which inherits from `IdentityDbContext`. It includes a constructor that takes `DbContextOptions<ApplicationContext>` and a `Categories` DbSet property.

```
1  using AndrewsBooks.Models;
2  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBookStore.DataAccess.Data
9  {
10    public class ApplicationContext : IdentityDbContext
11    {
12      public ApplicationContext(DbContextOptions<ApplicationContext> options)
13      : base(options)
14      {
15      }
16
17      public DbSet<Category> Categories { get; set; }
18    }
19 }
```

- Status Bar:** Shows the current branch as "master".

- 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.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Relational" Version="5.0.11" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.11" />
  </ItemGroup>
  <ItemGroup>
    <Folder Include="Data\" />
    <Folder Include="Repository\IRepository\" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\AndrewsBooks.Models\AndrewsBooks.Models.csproj" />
    <ProjectReference Include="..\AndrewsBooks.Utility\AndrewsBooks.Utility.csproj" />
  </ItemGroup>
</Project>
```

No issues found

Show output from: Build

Error List Web Publish Activity Package Manager Console Output

This item does not support previewing

AndrewsBookStore

AS Live Share

Solution Explorer

Properties

IRepository Folder Properties

Misc

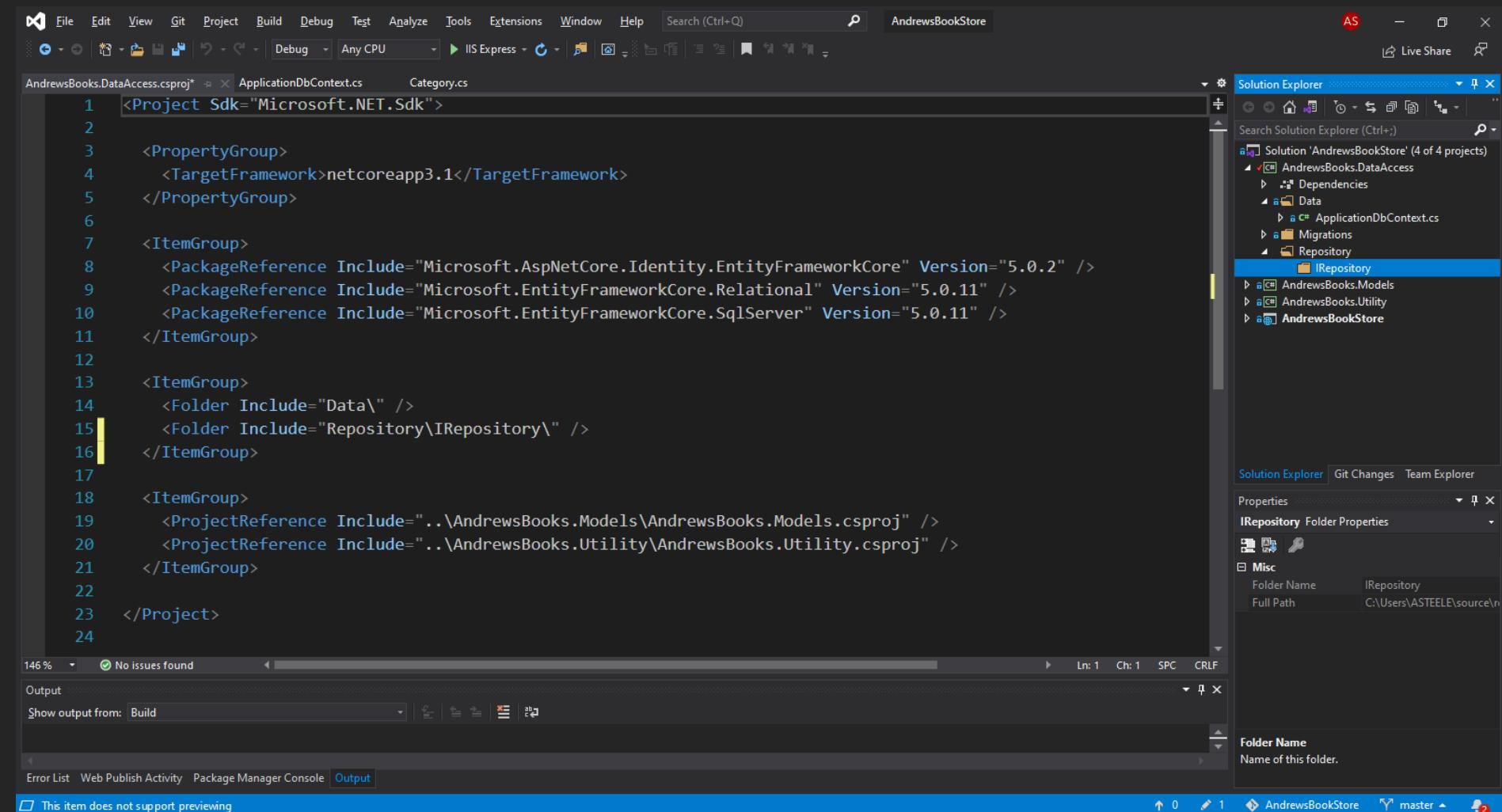
Folder Name IRepository

Full Path C:\Users\ASTEEL\source\repos\AndrewsBooks.DataAccess\AndrewsBooks.DataAccess\repository\IRepository

Folder Name Name of this folder.

- 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



The screenshot shows the Visual Studio IDE interface with the following details:

- Project Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The 'IRepository' folder under AndrewsBooks.DataAccess is selected.
- Code Editor:** Displays the contents of the 'AndrewsBooks.DataAccess.csproj' file, specifically the `<ItemGroup>` section for the 'Repository' folder.
- Status Bar:** Shows the message "This item does not support previewing".

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Relational" Version="5.0.11" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.11" />
  </ItemGroup>
  <ItemGroup>
    <Folder Include="Data\" />
    <Folder Include="Repository\IRepository\" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\AndrewsBooks.Models\AndrewsBooks.Models.csproj" />
    <ProjectReference Include="..\AndrewsBooks.Utility\AndrewsBooks.Utility.csproj" />
  </ItemGroup>
</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.
- 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).

The screenshot shows the Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the solution 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The AndrewsBooks.DataAccess project is currently selected.
- Code Editor:** Displays the contents of the `AndrewsBooks.DataAccess.csproj` file. The code defines a project targeting .NET Core 3.1 with Entity Framework Core dependencies. It includes references to `AndrewsBooks.Models` and `AndrewsBooks.Utility`. A folder named `IRepository` is created under `Repository`.
- Status Bar:** Shows the current branch is `master` with two pending changes.

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Relational" Version="5.0.11" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.11" />
  </ItemGroup>
  <ItemGroup>
    <Folder Include="Data\" />
    <Folder Include="Repository\IRepository\" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\AndrewsBooks.Models\AndrewsBooks.Models.csproj" />
    <ProjectReference Include="..\AndrewsBooks.Utility\AndrewsBooks.Utility.csproj" />
  </ItemGroup>
</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.
- 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).

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `IRepository.cs` in a dark-themed editor. The code defines a public interface `IRepository<T>` with two methods: `Get(int id)` and `GetAll(Expression<Func<T, bool> filter = null, Func<IQueryable<T>, IOrderedQueryable<T> orderBy = null, string includeProperties = null)`. The code uses several namespaces from the System and System.Linq families. The Solution Explorer on the right shows a solution named "AndrewsBookStore" with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The AndrewsBooks.DataAccess project contains a `Repository` folder which includes an `IRepository.cs` file. The Properties and Git Changes tabs are also visible in the Solution Explorer.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Linq.Expressions;
5 using System.Text;
6
7 namespace AndrewsBooks.DataAccess.Repository.IRepository
8 {
9     public interface IRepository<T> where T : class
10    {
11        T Get(int id);
12
13        IEnumerable<T> GetAll(
14            Expression<Func<T, bool> filter = null,
15            Func<IQueryable<T>, IOrderedQueryable<T>orderBy = null,
16            string includeProperties = null
17        );
18    }
19 }
```

- 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).

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q)
- Toolbars:** Standard, Debug, Task List, Solution Explorer, Properties, Output.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' (4 of 4 projects). Projects include AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. Under AndrewsBooks.DataAccess, there are Data, Migrations, and Repository folders, with IRepository.cs being the selected file.
- Code Editor:** The code for IRepository.cs is displayed:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Linq.Expressions;
5 using System.Text;
6
7 namespace AndrewsBooks.DataAccess.Repository.IRepository
8 {
9     public interface IRepository<T> where T : class
10    {
11        T Get(int id);
12
13        IEnumerable<T> GetAll(
14            Expression<Func<T, bool>> filter = null,
15            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
16            string includeProperties = null
17        );
18    }
19 }
```
- Status Bar:** 146%, No issues found, Ln: 17 Ch: 15 SPC CRLF.
- Output Tab:** Show output from: Build.
- Bottom Bar:** Error List, Web Publish Activity, Package Manager Console, Output (selected), Ready.

- 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).

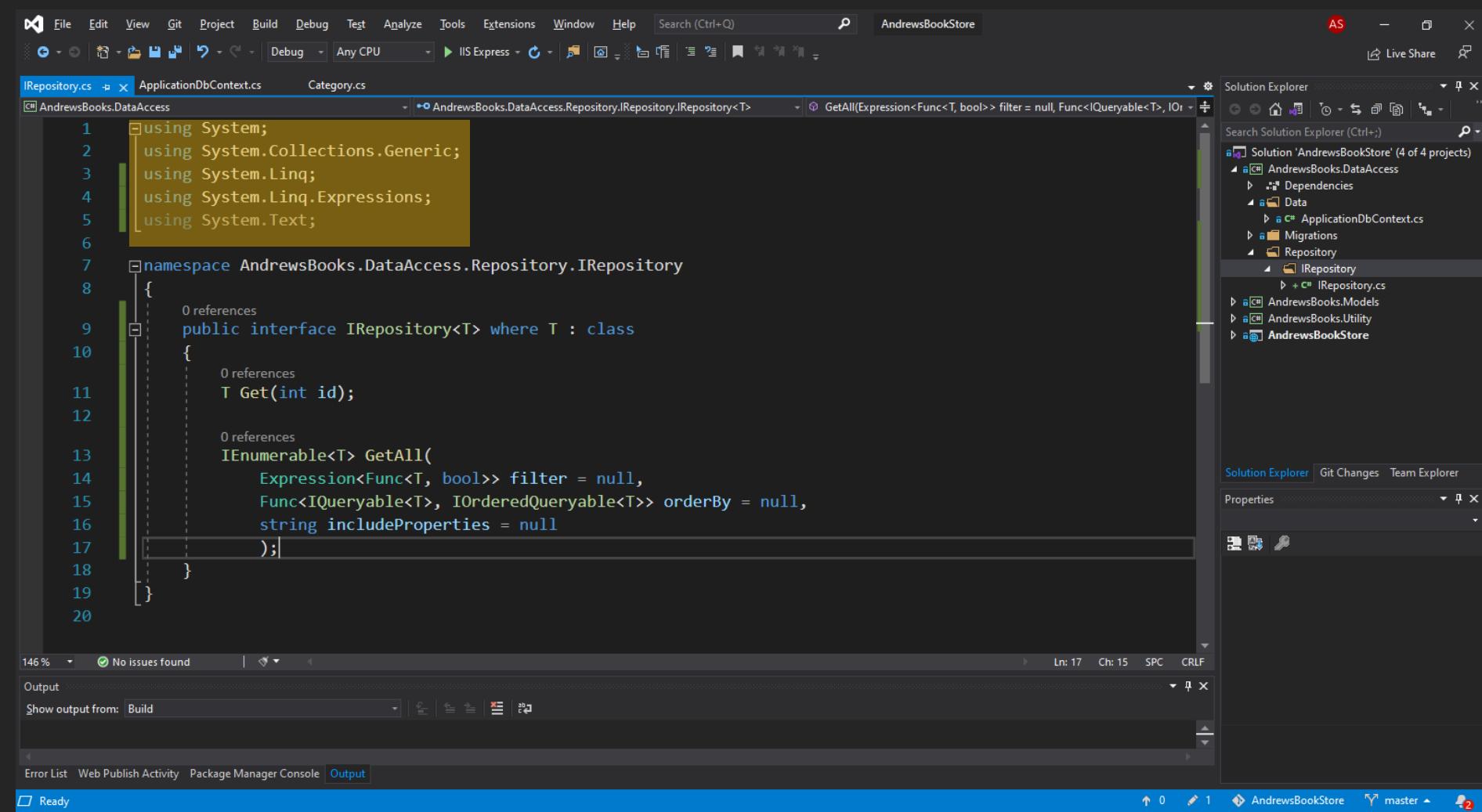
The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Toolbars:** Standard toolbar with icons for Open, Save, Print, etc.
- Search Bar:** Search (Ctrl+Q) at the top right.
- Project Name:** AndrewsBookStore
- Solution Explorer:** Shows the solution structure:
  - Solution 'AndrewsBookStore' (4 of 4 projects)
    - AndrewsBooks.DataAccess
      - Dependencies
      - Data
        - DbContext.cs
      - Migrations
      - Repository
        - IRepository.cs
    - AndrewsBooks.Models
    - AndrewsBooks.Utility
    - AndrewsBookStore
- Code Editor:** The IRepository.cs file is open, displaying the following C# code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Linq.Expressions;
5 using System.Text;
6
7 namespace AndrewsBooks.DataAccess.Repository.IRepository
8 {
9     public interface IRepository<T> where T : class
10    {
11        T Get(int id);
12
13        IEnumerable<T> GetAll(
14            Expression<Func<T, bool>> filter = null,
15            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
16            string includeProperties = null
17        );
18    }
19 }
20
```
- Status Bar:** Shows 146%, No issues found, Ln: 17 Ch: 15 SPC CRLF.
- Output Tab:** Shows Output, Show output from: Build.
- Bottom Bar:** Error List, Web Publish Activity, Package Manager Console, Output (highlighted), Ready.

- 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



```

IRepository.cs  ApplicationDbContext.cs  Category.cs
AndrewsBooks.DataAccess
  ↳ AndrewsBooks.DataAccess.Repository.IRepository<T>
    ↳ GetAll(Expression<Func<T, bool> filter = null, Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null, string includeProperties = null)
  ↳ AndrewsBooks.DataAccess.Repository.IRepository
    ↳ IRepository.cs
  ↳ AndrewsBooks.Models
  ↳ AndrewsBooks.Utility
  ↳ AndrewsBookStore

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository.IRepository
{
    public interface IRepository<T> where T : class
    {
        T Get(int id);

        IEnumerable<T> GetAll(
            Expression<Func<T, bool>> filter = null,
            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
            string includeProperties = null
        );
    }
}

```

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `IRepository.cs`. The code defines a public interface `IRepository<T>` with two methods: `Get(int id)` and `GetAll()`. The `GetAll()` method takes three parameters: `filter`, `orderBy`, and `includeProperties`. The code is highlighted in yellow. The Solution Explorer on the right shows the project structure with `AndrewsBooks.DataAccess` selected, containing `Dependencies`, `Data` (with `ApplicationDbContext.cs`), `Migrations`, `Repository` (with `IRepository` and `IRepository.cs`), `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`.

- 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

```

10     {
11         T Get(int id); // Retrieve a category from the database by id
12         // List of Categories based on requirements
13         0 references
14
15         IEnumerable<T> GetAll(
16             Expression<Func<T, bool>> filter = null,
17             Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
18             string includeProperties = null // useful for foreign key references
19             );
20         0 references
21
22         T GetFirstOrDefault(
23             Expression<Func<T, bool>> filter = null,
24             string includeProperties = null
25             );
26         0 references
27         void Add(T entity); // to add an entity
28         0 references
29         void Remove(int id); // to remove an object or category
30         0 references
31         void Remove(T entity); // another way to remove an object
32         0 references
33         void RemoveRange(IEnumerable<T> entity); // removes a complete range of entities
34     }
35 }
36 
```

The screenshot shows the Visual Studio IDE with the IRepository.cs file open in the code editor. The code defines a generic repository interface with methods for Get, GetAll, GetFirstOrDefault, Add, Remove, and RemoveRange. The RemoveRange method is highlighted with a yellow rectangle. The Solution Explorer on the right shows the project structure with AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore projects. The AndrewsBooks.DataAccess project contains the IRepository folder with the IRepository.cs file. The AndrewsBookStore project is currently selected.

- Implement the class that implements the repository

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the `ApplicationDbContext.cs` file from the `AndrewsBooks.DataAccess` project. The file contains the following C# code:

```
1  using AndrewsBooks.Models;
2  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBookStore.DataAccess.Data
9  {
10    public class ApplicationDbContext : IdentityDbContext
11    {
12      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
13      : base(options)
14      {
15      }
16      public DbSet<Category> Categories { get; set; }
17    }
18  }
```

The Solution Explorer on the right shows the solution structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBookStore` project is currently selected.

- Implement the class that implements the repository

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `Repository.cs` in the `AndrewsBooks.DataAccess` project. The code implements the `IRepository<T>` interface. A context menu is open over the code editor, specifically over the `GetFirstOrDefault` method. The menu is titled "Add" and includes options like "New Item...", "Existing Item...", "New Folder", "REST API Client...", "Class...", "New EditorConfig", "Run Tests", "Debug Tests", "Scope to This", "New Solution Explorer View", "Git", "Exclude From Project", "Cut", "Copy", "Delete", "Rename", "Copy Full Path", "Open Folder in File Explorer", "Open in Terminal", and "Properties". The "Properties" option is highlighted. The Solution Explorer on the right shows the project structure, including `AndrewsBooks.DataAccess`, `Dependencies`, `Data` (containing `ApplicationContext.cs`), `Migrations`, and `Repository` (containing `IRepository.cs`). The `Properties` tool window is also visible at the bottom.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Linq.Expressions;
5 using System.Text;
6
7 namespace AndrewsBooks.DataAccess.Repository.IRepository
8 {
9     public interface IRepository<T> where T : class
10    {
11        T Get(int id); // Retrieve a category from the database by id
12        // List of Categories based on requirements
13        IEnumerable<T> GetAll(
14            Expression<Func<T, bool>> filter = null,
15            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
16            string includeProperties = null // useful for foreign key references
17        );
18        T GetFirstOrDefault(
19            Expression<Func<T, bool>> filter = null,
20            string includeProperties = null
21    }
22 }
```

No issues found

Output

Show output from: Build

Error List Web Publish Activity Package Manager Console Output

This item does not support previewing

AS Live Share

Solution Explorer

Search Solution Explorer (Ctrl+F)

Solution 'AndrewsBookStore' (4 of 4 projects)

- AndrewsBooks.DataAccess
  - Dependencies
  - Data
    - ApplicationContext.cs
  - Migrations
  - Repository
    - IRepository.cs
- AndrewsBooks.Models
- AndrewsBooks.Utility
- AndrewsBookStore

Add

- New Item... Ctrl+Shift+A
- Existing Item... Shift+Alt+A
- New Folder
- REST API Client...
- Class... **Class...**
- New EditorConfig

Run Tests

Debug Tests

Scope to This

New Solution Explorer View

Git

Exclude From Project

Cut Ctrl+X

Copy Ctrl+C

Delete Del

Rename F2

Copy Full Path

Open Folder in File Explorer

Open in Terminal

Properties Alt+Enter

Properties

Misc

Folder Name Repository

Full Path C:\Users\ASTEEL\source\repos\AndrewsBookStore\AndrewsBooks.DataAccess\Repository\IRepository.cs

Folder Name Name of this folder.

Ln: 1 Ch: 1 SPC: CRLF

0 1 AndrewsBookStore master 2

- Implement the class that implements the repository
- Include the using statement
- View the potential fixes and 'implement interface'

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays a code editor with a C# file named `Repository.cs`. The code defines a public interface `IRepository<T>` with several methods: `Get(int id)`, `GetAll()`, and `GetFirstOrDefault()`. The `GetAll()` method includes comments about filtering and ordering. A context menu is open over the interface definition, showing options like "New Item...", "Existing Item...", "New Folder", "Class...", "Add", "Run Tests", "Debug Tests", "Scope to This", "New Solution Explorer View", "Git", "Exclude From Project", "Cut", "Copy", "Delete", "Rename", "Copy Full Path", "Open Folder in File Explorer", "Open in Terminal", and "Properties". The "Class..." option is highlighted. The Solution Explorer panel on the right shows the project structure for "AndrewsBookStore" with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project is expanded, showing its `Dependencies`, `Data` (containing `ApplicationContext.cs`), and `Migrations` folders. The `Repository` folder under `Data` is selected. The Properties panel on the right shows the `Repository` properties with the folder name set to "Repository" and the full path to "C:\Users\ASTEEL\source\repos\AndrewsBookStore\AndrewsBooks.DataAccess\Repository". The status bar at the bottom indicates the item does not support previewing.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository.IRepository
{
    public interface IRepository<T> where T : class
    {
        T Get(int id); // Retrieve a category from the database by id
        // List of Categories based on requirements
        IEnumerable<T> GetAll(
            Expression<Func<T, bool>> filter = null,
            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
            string includeProperties = null // useful for foreign key references
        );
        T GetFirstOrDefault(
            Expression<Func<T, bool>> filter = null,
            string includeProperties = null
        );
    }
}

```

- Implement the class that implements the repository
- Include the using statement
- View the potential fixes and 'implement interface'

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays a code editor with the file `Repository.cs` open. The code defines a generic class `Repository<T>` that implements the `IRepository<T>` interface. A tooltip from the Intellisense feature is visible, suggesting two options: `Implement interface` and `Implement all members explicitly`. The Solution Explorer on the right shows the project structure for 'AndrewsBookStore' with several sub-projects like `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, and `AndrewsBooks.Utility`.

```

using AndrewsBooks.DataAccess.Repository.IRepository;
using System;
using System.Collections.Generic;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository
{
    public class Repository<T> : IRepository<T> where T : class
    {
        // Implementation code follows
    }
}

// Error message in the status bar: CS0535 'Repository<T>' does not implement interface member 'IRepository<T>.Get(int)'

```

- 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)

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `Repository.cs` in the `AndrewsBooks.DataAccess` project. The code defines a generic class `Repository<T>` that implements the `IRepository<T>` interface. A tooltip from the Intellisense dropdown suggests implementing the interface explicitly. The Solution Explorer on the right shows the solution structure with projects like `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The Error List at the bottom indicates a warning about the class not implementing the `Get(int)` method.

```

using AndrewsBooks.DataAccess.Repository.IRepository;
using System;
using System.Collections.Generic;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository
{
    public class Repository<T> : IRepository<T> where T : class
    {
        public void Add(T entity)
        {
            throw new NotImplementedException();
        }

        public T Get(int id)
        {
            throw new NotImplementedException();
        }

        public IEnumerable<T> GetAll(Expression<Func<T, bool>> filter)
        {
            throw new NotImplementedException();
        }

        public T GetFirstOrDefault(Expression<Func<T, bool>> filter =
        {
            throw new NotImplementedException();
        }

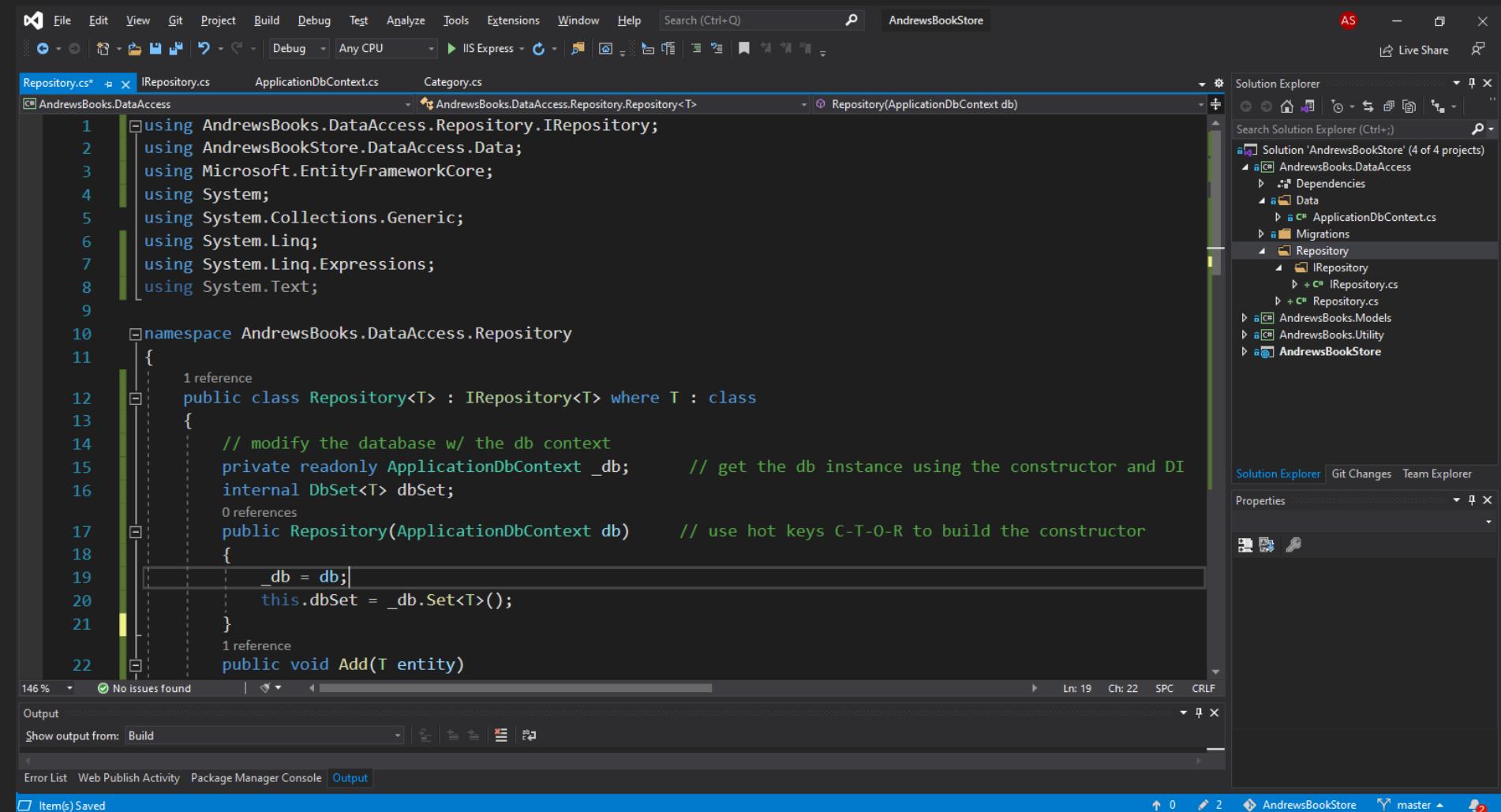
        public void Remove(int id)
        {
            throw new NotImplementedException();
        }

        public void Remove(T entity)
        {
            throw new NotImplementedException();
        }

        public void RemoveRange(IEnumerable<T> entity)
        {
            throw new NotImplementedException();
        }
    }
}

```

- 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)



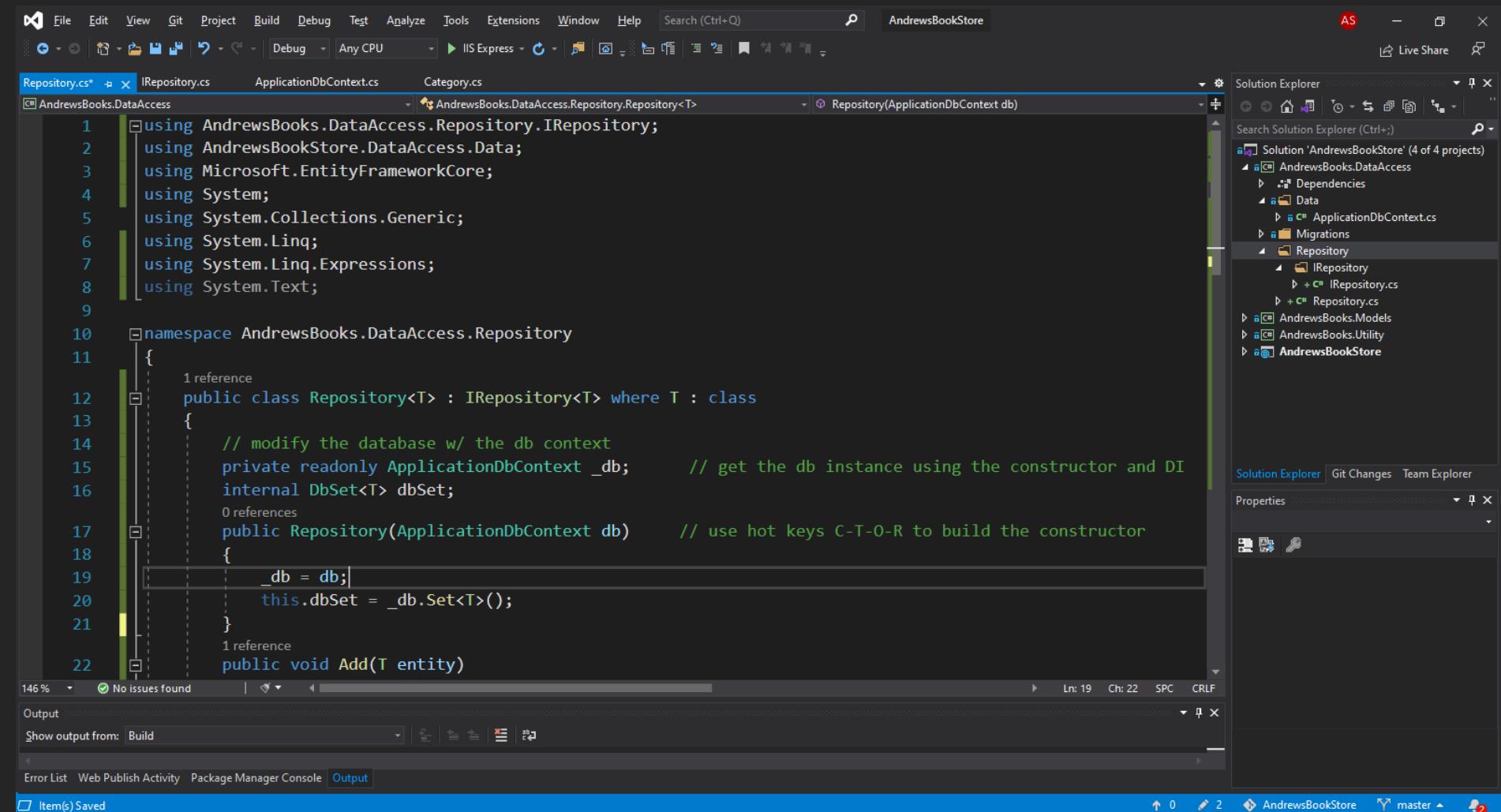
The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q).
- Toolbars:** Standard, Debug, Task List, Solution Explorer, Properties, Task List, Output.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The active file is `Repository.cs` under the `AndrewsBooks.DataAccess` project. The code implements the `IRepository<T>` interface using dependency injection.
- Properties Window:** Shows the properties for the selected item.
- Output Window:** Shows build output.
- Status Bar:** Item(s) Saved, Ln: 19 Ch: 22 SPC CRLF.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBookStore.DataAccess.Data;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Linq.Expressions;
8  using System.Text;
9
10 namespace AndrewsBooks.DataAccess.Repository
11 {
12     public class Repository<T> : IRepository<T> where T : class
13     {
14         // modify the database w/ the db context
15         private readonly ApplicationDbContext _db;           // get the db instance using the constructor and DI
16         internal DbSet<T> dbSet;
17
18         public Repository(ApplicationDbContext db)      // use hot keys C-T-O-R to build the constructor
19         {
20             _db = db;
21             this.dbSet = _db.Set<T>();
22         }
23
24         public void Add(T entity)
25     }
26
27 }
```

- 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



The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q).
- Toolbars:** Standard, Debug, Task List, Solution Explorer, Properties, Task List, Output.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The 'Repository' folder under AndrewsBooks.DataAccess is selected.
- Code Editor:** The active file is `Repository.cs` in the `AndrewsBooks.DataAccess` project. The code implements the  `IRepository<T>` interface using Entity Framework Core's `DbContext` and `DbSet`.
- Status Bar:** Shows the current branch is 'master' with 2 changes.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBookStore.DataAccess.Data;
3  using Microsoft.EntityFrameworkCore;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Linq.Expressions;
8  using System.Text;
9
10 namespace AndrewsBooks.DataAccess.Repository
11 {
12     public class Repository<T> : IRepository<T> where T : class
13     {
14         // modify the database w/ the db context
15         private readonly ApplicationDbContext _db;           // get the db instance using the constructor and DI
16         internal DbSet<T> dbSet;
17
18         public Repository(ApplicationDbContext db)        // use hot keys C-T-O-R to build the constructor
19         {
20             _db = db;
21             this.dbSet = _db.Set<T>();
22         }
23
24         public void Add(T entity)
25     }
26
27 }
```

- 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

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Code Editor:** Displays `IRepository.cs` from the `AndrewsBooks.DataAccess.Repository` project. The code defines an interface `ICategoryRepository` with a single method.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace AndrewsBooks.DataAccess.Repository.IRepository
6 {
7     interface ICategoryRepository
8     {
9     }
10}
```

- Solution Explorer:** Shows the solution structure for `AndrewsBookStore`, which includes four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Repository` folder under `AndrewsBooks.DataAccess` contains the `ICategoryRepository.cs` file.
- Properties:** Shows the properties for a selected item in the `Misc` folder.
- Output:** Shows the build output for the current project.

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

The screenshot shows a Visual Studio interface with the following components:

- Solution Explorer:** Shows the project structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Properties:** Shows the 'Misc' folder.
- Output:** Shows the 'Output' window with the 'Build' tab selected.
- Code Editor:** Displays the file `ICategoryRepository.cs` containing the following code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace AndrewsBooks.DataAccess.Repository.IRepository
6 {
7     0 references
8     interface ICategoryRepository
9     {
10 }
11
```

- 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)

The screenshot shows the Microsoft Visual Studio interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The AndrewsBooks.DataAccess project is expanded, showing its subfolders (Dependencies, Data, Migrations, Repository), interfaces (ICategoryRepository, IRepository), and classes (CategoryRepository, Repository).
- Code Editor:** The current file is 'ICategoryRepository.cs' under the 'AndrewsBooks.DataAccess.Repository.IRepository' namespace. The code defines an interface with a single method:
 

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace AndrewsBooks.DataAccess.Repository.IRepository
6 {
7     0 references
8     interface ICategoryRepository
9     {
10 }
11
      
```
- Output Window:** Shows 'No issues found'.
- Status Bar:** Displays 'Ready'.

- 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)

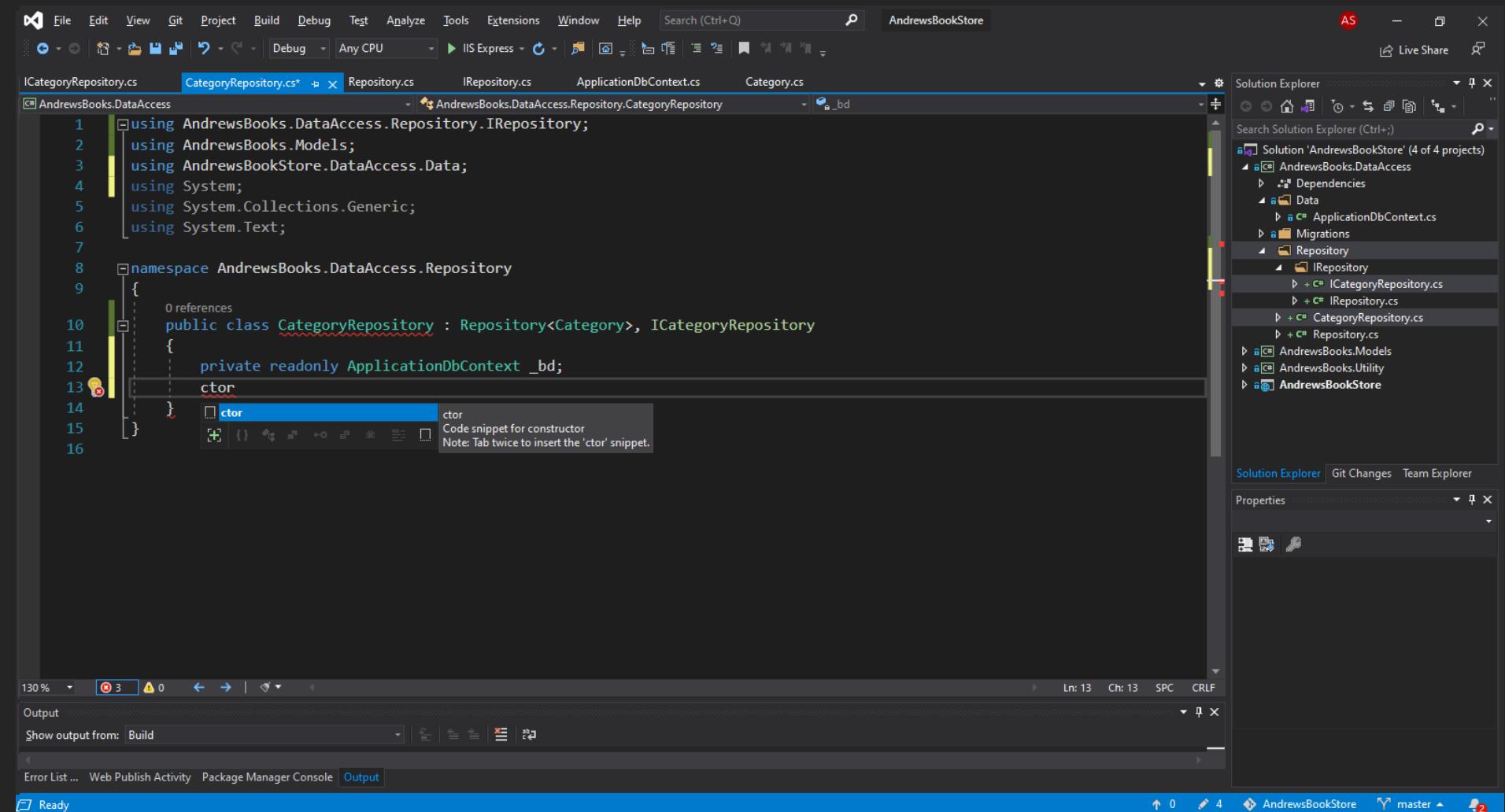
The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `CategoryRepository.cs` in the `AndrewsBooks.DataAccess` project. The code implements the `ICategoryRepository` interface. A tooltip for the constructor of `CategoryRepository` indicates a warning: `CS7036: There is no argument given that corresponds to the required formal parameter 'db' of 'Repository<Category>.Repository(ApplicationContext)'`. The Solution Explorer on the right shows the solution contains four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project includes `Repository.cs`, `ICategoryRepository.cs`, and `Category.cs`.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBooks.Models;
3  using System;
4  using System.Collections.Generic;
5  using System.Text;
6
7  namespace AndrewsBooks.DataAccess.Repository
8  {
9      public class CategoryRepository : Repository<Category>, ICategoryRepository
10     {
11     }
12 }
13

```

- 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)



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar says "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The toolbar has icons for file operations like Open, Save, and Build.

The Solution Explorer on the right shows a solution named "AndrewsBookStore" containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The AndrewsBooks.DataAccess project contains Data, Migrations, and Repository folders, with IRepository.cs, ICategoryRepository.cs, CategoryRepository.cs, and Repository.cs files.

The code editor window displays CategoryRepository.cs:

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBooks.Models;
3  using AndrewsBookStore.DataAccess.Data;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBooks.DataAccess.Repository
9  {
10    public class CategoryRepository : Repository<Category>, ICategoryRepository
11    {
12      private readonly ApplicationDbContext _bd;
13      ctor
14    }
15  }

```

A tooltip for the constructor ("ctor") is visible, stating: "Code snippet for constructor" and "Note: Tab twice to insert the 'ctor' snippet."

The bottom status bar shows "Ready" and other build-related information.

- 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)

The screenshot shows the Microsoft Visual Studio interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q).
- Toolbox:** Standard icons for file operations.
- Current Project:** AndrewsBookStore (selected in the dropdown).
- Current File:** CategoryRepository.cs (highlighted in blue).
- Code Editor:**

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBooks.Models;
3  using AndrewsBookStore.DataAccess.Data;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBooks.DataAccess.Repository
9  {
10    public class CategoryRepository : Repository<Category>, ICategoryRepository
11    {
12      private readonly ApplicationDbContext _db;
13
14      public CategoryRepository(ApplicationDbContext db) : base(db)
15      {
16        _db = db;
17      }
18    }
19

```
- Solution Explorer:** Shows the project structure:
  - Solution 'AndrewsBookStore' (4 of 4 projects)
  - AndrewsBooks.DataAccess
    - Dependencies
    - Data
    - Migrations
    - Repository
      - IRepository
      - ICategoryRepository.cs
      - IRepository.cs
      - CategoryRepository.cs
      - IRepository.cs
  - AndrewsBooks.Models
  - AndrewsBooks.Utility
  - AndrewsBookStore
- Properties:** Standard Visual Studio property pages.
- Output:** Shows 'No issues found'.
- Bottom Status Bar:** Item(s) Saved, 0 errors, 4 warnings, master branch.

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Top Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q), AndrewsBookStore.
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' (4 of 4 projects). The 'Repository' project is selected, displaying its contents: Dependencies, Data, ApplicationDbContext.cs, Migrations, Repository, IRepository, ICategoryRepository.cs, IRepository.cs, CategoryRepository.cs, Repository.cs, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The file 'ICategoryRepository.cs' is open, showing the following code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace AndrewsBooks.DataAccess.Repository.IRepository
6 {
7     interface ICategoryRepository
8     {
9     }
10 }
```
- Status Bar:** 130%, No issues found, Ln: 1 Ch: 1 SPC CRLF.
- Output Tab:** Shows output from Build.
- Bottom Status Bar:** Ready, 0, 4, AndrewsBookStore, master, 2.

- Modify ICategoryRepository interface

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Code Editor:** The main window displays the `ICategoryRepository.cs` file from the `AndrewsBooks.DataAccess` project. The code defines an interface with a single method:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace AndrewsBooks.DataAccess.Repository.IRepository
6 {
7     interface ICategoryRepository
8     {
9     }
10}
```

- Solution Explorer:** Shows the solution structure with four projects:
  - `AndrewsBooks.DataAccess`: Contains `Dependencies`, `Data` (with `ApplicationContext.cs`), `Migrations`, `Repository` (containing `ICategoryRepository.cs`,  `IRepository.cs`, `CategoryRepository.cs`, and `Repository.cs`), `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`.
  - `AndrewsBooks.Models`
  - `AndrewsBooks.Utility`
  - `AndrewsBookStore`
- Properties:** A panel on the right showing properties for selected files.
- Output:** A bottom panel showing build output and other developer tools.

- Modify ICategoryRepository interface

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the file `ICategoryRepository.cs` from the project `AndrewsBooks.DataAccess`. The code defines a public interface `ICategoryRepository` that implements the `IRepository<Category>` interface. The interface contains one method, `Update(Category category)`. The Solution Explorer on the right shows the solution structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project is expanded to show its internal structure, including `Dependencies`, `Data`, `Migrations`, `Repository`, and  `IRepository`, with `ICategoryRepository.cs` highlighted.

```
1 using AndrewsBooks.Models;
2 using System;
3 using System.Collections.Generic;
4 using System.Text;
5
6 namespace AndrewsBooks.DataAccess.Repository.IRepository
7 {
8     public interface ICategoryRepository : IRepository<Category>
9     {
10        void Update(Category category);
11    }
12 }
```

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

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Code Editor:** The main window displays the `ICategoryRepository.cs` file from the `AndrewsBooks.DataAccess` project. The code defines a public interface `ICategoryRepository` that implements the `IRepository<Category>` interface. It includes a single method `Update(Category category)`.
- Solution Explorer:** Located on the right side, it shows the solution structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Repository` folder under `AndrewsBooks.DataAccess` contains the `ICategoryRepository.cs` file.
- Properties Window:** Located at the bottom right, it shows standard file properties like `Properties`, `File`, and `Image`.
- Output Window:** Located at the bottom left, it shows the output from the build process.
- Status Bar:** At the very bottom, it displays the current branch as `master` and has a red notification icon with the number `2`.

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

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- Title Bar:** AndrewsBookStore
- Toolbars:** Standard (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help) and Search (Ctrl+Q).
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBookStore.DataAccess.Data, and AndrewsBookStore.
- Code Editor:** The file `CategoryRepository.cs` is open, showing the implementation of the `ICategoryRepository` interface. The code includes imports for `AndrewsBooks.DataAccess.Repository.IRepository`, `AndrewsBooks.Models`, `AndrewsBookStore.DataAccess.Data`, `System`, `System.Collections.Generic`, and `System.Text`. It defines a `CategoryRepository` class that implements `ICategoryRepository` and inherits from `Repository<Category>`.
- Tooltips and Error Message:** A tooltip appears over the `ICategoryRepository` interface reference, stating: "••• interface AndrewsBooks.DataAccess.Repository.IRepository.ICategoryRepository CS0535: 'CategoryRepository' does not implement interface member 'ICategoryRepository.Update(Category)' Show potential fixes (Alt+Enter or Ctrl+.)".
- Status Bar:** Item(s) Saved, Ln: 15 Ch: 22 SPC CRLF.

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

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- Title Bar:** AndrewsBookStore
- Toolbars:** Standard (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help) and Search (Ctrl+Q).
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBookStore.DataAccess.Data, and AndrewsBookStore.
- Code Editor:** The file 'CategoryRepository.cs' is open in the editor. The code implements the `ICategoryRepository` interface. A tooltip from the IDE indicates a warning: CS0535: 'CategoryRepository' does not implement interface member 'ICategoryRepository.Update(Category)'.
- Status Bar:** Item(s) Saved, Ln: 15 Ch: 22 SPC CRLF.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBooks.Models;
3  using AndrewsBookStore.DataAccess.Data;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBooks.DataAccess.Repository
9  {
10    1 reference
11    public class CategoryRepository : Repository<Category>, ICategoryRepository
12    {
13      private readonly ApplicationDbContext _db;
14      public CategoryRepository(ApplicationDbContext db) : base(db)
15      {
16        _db = db;
17      }
18    }
19

```

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

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBooks.Models;
3  using AndrewsBookStore.DataAccess.Data;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace AndrewsBooks.DataAccess.Repository
9  {
10    public class CategoryRepository : Repository<Category>, ICategoryRepository
11    {
12      private readonly ApplicationDbContext _db;
13
14      public CategoryRepository(ApplicationDbContext db) : base(db)
15      {
16        _db = db;
17      }
18
19      public void Update(Category category)
20      {
21        throw new NotImplementedException();
22      }
23    }
24

```

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `CategoryRepository.cs` in the `AndrewsBooks.DataAccess` project. The code implements the `ICategoryRepository` interface. The interface definition is at the top, followed by the implementation which includes a constructor taking an `ApplicationDbContext` and an `Update` method that throws a `NotImplementedException`. The Solution Explorer on the right shows the project structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`.

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

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code editor with the file `CategoryRepository.cs` open. The code implements the `ICategoryRepository` interface, specifically the `Update` method. The Solution Explorer window on the right shows the project structure for 'AndrewsBookStore' with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project contains files like `ApplicationDbContext.cs`, `Category.cs`, `IRepository.cs`, and `Repository.cs`. The `Properties` window is also visible.

```

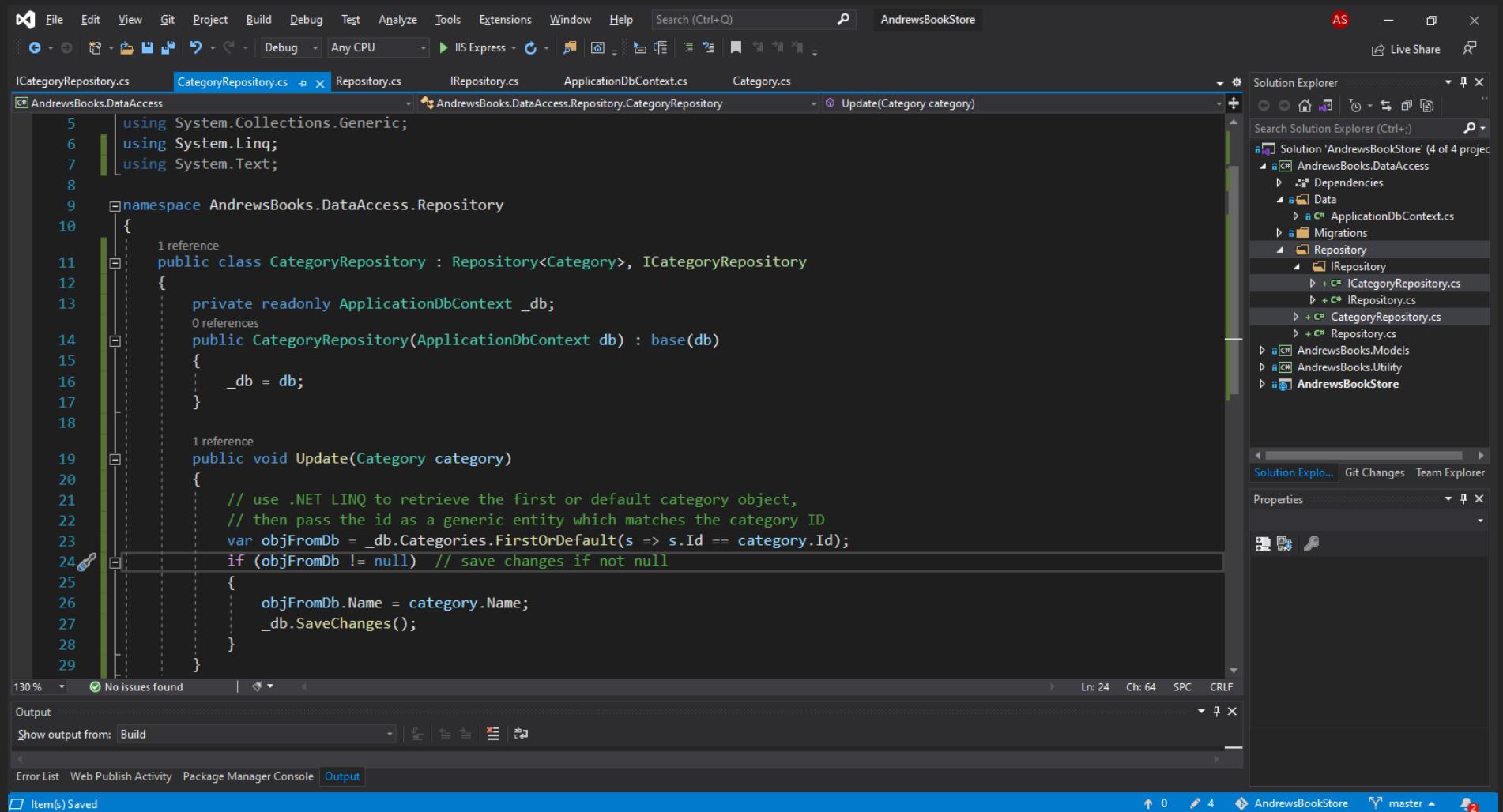
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository
{
    public class CategoryRepository : Repository<Category>, ICategoryRepository
    {
        private readonly ApplicationDbContext _db;
        public CategoryRepository(ApplicationDbContext db) : base(db)
        {
            _db = db;
        }

        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();
            }
        }
    }
}

```

- 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



The screenshot shows a Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows the project structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** Displays the file `CategoryRepository.cs` under the namespace `AndrewsBooks.DataAccess.Repository`. The code implements the `ICategoryRepository` interface and contains a `Update` method that uses LINQ to update a category in the database.
- Output Window:** Shows the build output from the 'Build' tab.

```

using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository
{
    public class CategoryRepository : Repository<Category>, ICategoryRepository
    {
        private readonly ApplicationDbContext _db;
        public CategoryRepository(ApplicationDbContext db) : base(db)
        {
            _db = db;
        }

        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();
            }
        }
    }
}

```

The screenshot shows a Microsoft Visual Studio interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Toolbars:** Standard, Debug, Task List, Solution Explorer, Properties, Task List, Output, Error List, Web Publish Activity, Package Manager Console, Output.
- Solution Explorer:** Shows the project structure:
  - Solution 'AndrewsBookStore' (4 of 4 projects)
  - AndrewsBooks.DataAccess
  - Dependencies
  - Data
  - Migrations
  - Repository
    - IRepository
    - ICategoryRepository.cs
    - IRepository.cs
    - CategoryRepository.cs
  - AndrewsBooks.Models
  - AndrewsBooks.Utility
  - AndrewsBookStore
- Code Editor:** Displays the `CategoryRepository.cs` file from the `AndrewsBooks.DataAccess` project. The code implements a repository pattern for managing categories in an `ApplicationContext`.

```
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository
{
    public class CategoryRepository : Repository<Category>, ICategoryRepository
    {
        private readonly ApplicationContext _db;
        public CategoryRepository(ApplicationContext db) : base(db)
        {
            _db = db;
        }

        public void Update(Category category)
        {
            var objFromDb = _db.Categories.FirstOrDefault(s => s.Id == category.Id);
            if (objFromDb != null) // save changes if not null
            {
                objFromDb.Name = category.Name;
                _db.SaveChanges();
            }
        }
    }
}
```
- Status Bar:** Item(s) Saved, Line 24, Column 64, SPC, CRLF.

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

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

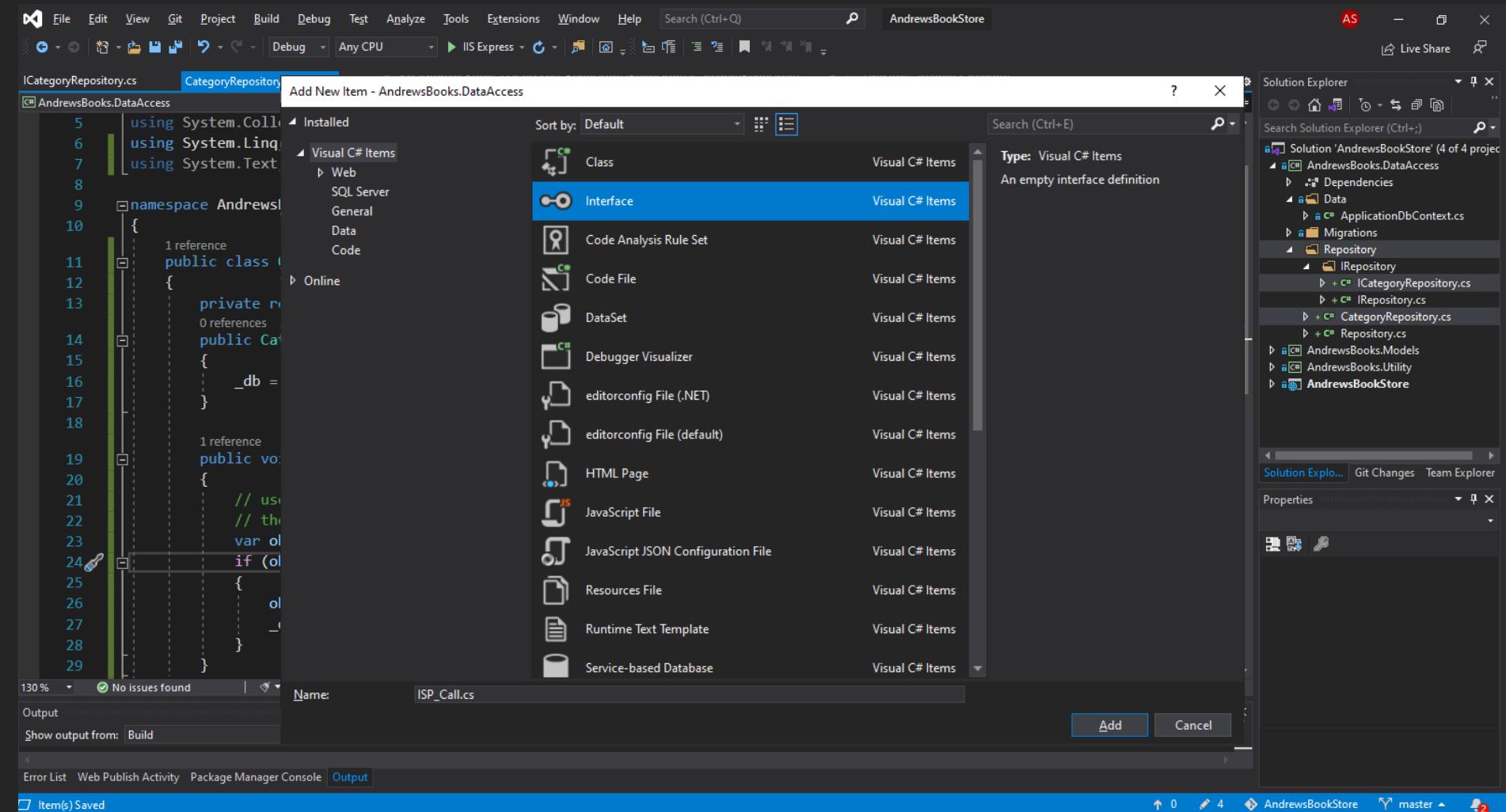
- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Toolbars:** Standard toolbar with icons for Open, Save, Print, etc.
- Code Editor:** The main window displays the `CategoryRepository.cs` file under the `AndrewsBooks.DataAccess` namespace. The code implements a repository for the `Category` entity using `ApplicationDbContext`.
- Solution Explorer:** Shows the project structure for 'AndrewsBookStore' (4 of 4 projects).
  - `AndrewsBooks.DataAccess`: Contains `Dependencies`, `Data` (with `ApplicationContext.cs`), `Migrations`, `Repository` (with `IRepository`, `ICategoryRepository.cs`, `IRepository.cs`, `CategoryRepository.cs`), and `Models`.
  - `AndrewsBooks.Models`
  - `AndrewsBooks.Utility`
  - `AndrewsBookStore`
- Properties Window:** Available on the right side of the interface.
- Status Bar:** Shows the current branch as `master` and other status indicators.

```
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository
{
    public class CategoryRepository : Repository<Category>, ICategoryRepository
    {
        private readonly ApplicationDbContext _db;
        public CategoryRepository(ApplicationDbContext db) : base(db)
        {
            _db = db;
        }

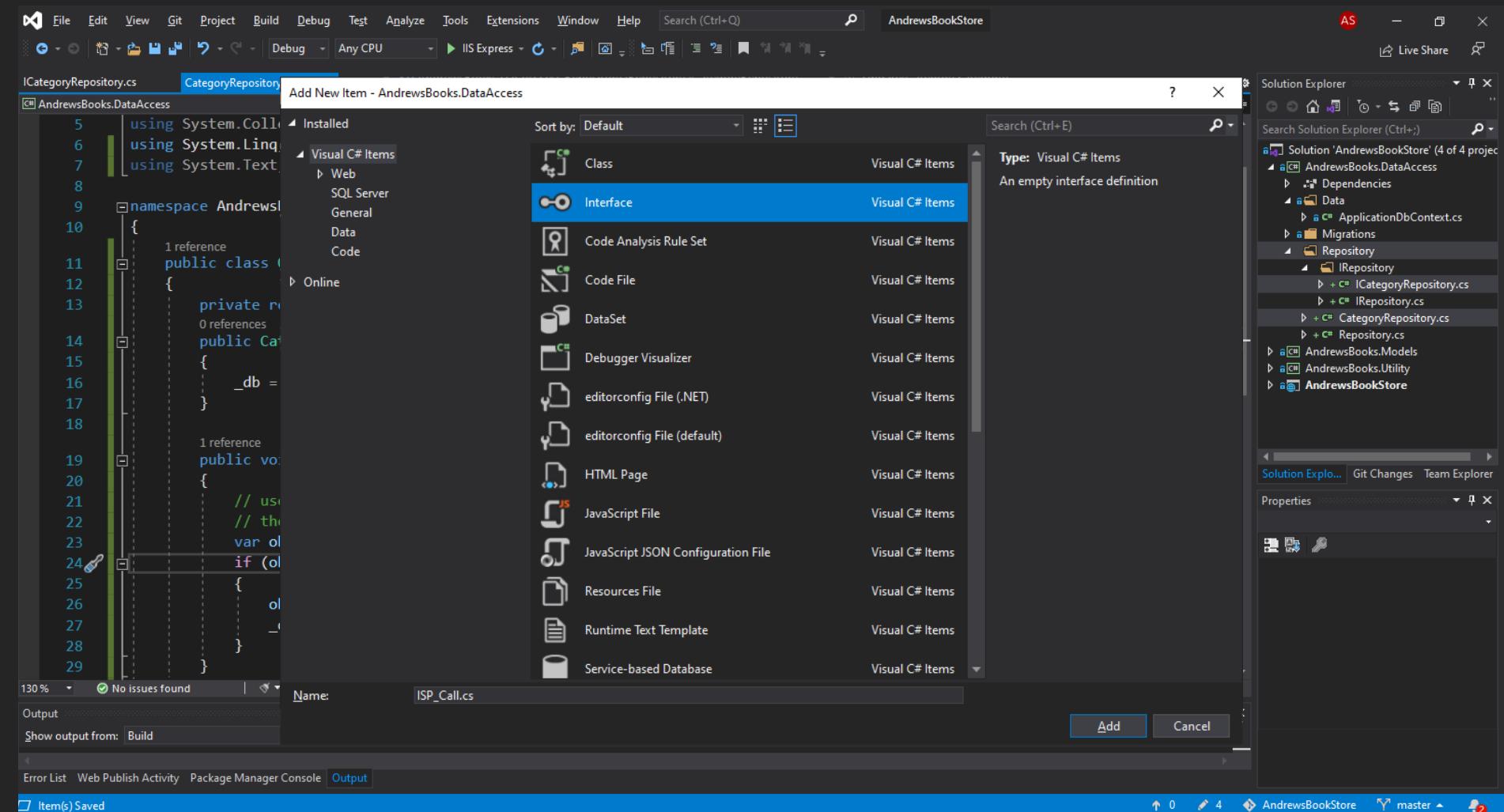
        public void Update(Category category)
        {
            // use .NET LINQ to retrieve the first or default category object,
            // then pass the id as a generic entity which matches the category ID
            var objFromDb = _db.Categories.FirstOrDefault(s => s.Id == category.Id);
            if (objFromDb != null) // save changes if not null
            {
                objFromDb.Name = category.Name;
                _db.SaveChanges();
            }
        }
    }
}
```

- 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

The screenshot shows a Visual Studio IDE window with the following details:

- Project:** AndrewsBookStore
- File:** ISP\_Call.cs
- Code Snippet:**

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository.IRepository
6  {
7      public interface ISP_Call : IDisposable
8      {
9          T Single<T>(string procedurename, DynamicParameters);
10 }
11 }
```
- Toolbox:** A context menu is open over the word "DynamicParameters" at line 9, showing options:
  - Generate class 'DynamicParameters' in new file
  - Generate class 'DynamicParameters'
  - Generate new type...
- Output:** Shows the message "Install package 'Dapper'".
- Errors:** CS0246: The type or namespace name 'DynamicParameters' could not be found. Are you missing a using directive or an assembly reference?
- Solution Explorer:** Shows the project structure with files like ICategoryRepository.cs, IRepository.cs, Category.cs, and ISP\_Call.cs.
- Task List:** Shows tasks related to Dapper installation.
- Status Bar:** Shows the current branch is master with two changes.

- 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

The screenshot shows the Visual Studio IDE with the following details:

- Project Structure:** The Solution Explorer shows a solution named 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The main window displays the file `ISP_Call.cs` which contains the following code:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository.IRepository
6  {
7      public interface ISP_Call : IDisposable
8      {
9          T Single<T>(string procedurename, DynamicParameters);
10 }
11 }
```
- Context Menu:** A context menu is open over the word `DynamicParameters` at line 9, showing options like "Generate class 'DynamicParameters' in new file", "Generate class 'DynamicParameters'", and "Generate new type...".
- Toolbars and Status Bar:** The status bar at the bottom indicates the file is "Ready".
- Output Window:** The Output window shows the message "CS0246 The type or namespace name 'DynamicP" followed by a tooltip for "DynamicParameters".
- Task List:** The Task List window shows the following items:
  - Find and install latest version of Dapper
  - Install with package manager...
  - using Dapper;
  - using System;
  - Find and install latest version of Dapper
  - Preview changes

- 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.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository.IRepository
6  {
7      public interface ISP_Call : IDisposable
8      {
9          T Single<T>(string procedurename, DynamicParameters);
10     }
11 }

```

Generate class 'DynamicParameters' in new file  
Generate class 'DynamicParameters'  
Generate new type...

Install package 'Dapper'  
Find and install latest version  
Install with package manager...

CS0246 The type or namespace name 'DynamicParameters' could not be found (are you missing a using directive or an assembly reference?)  
using Dapper;  
using System;

- 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.

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar "Search (Ctrl+Q)". The toolbar has icons for file operations like Open, Save, and Print, along with IIS Express and other development tools.

The code editor displays the file `ISP_Call.cs` which contains the following C# code:

```

1  using Dapper;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace AndrewsBooks.DataAccess.Repository.IRepository
7  {
8      public interface ISP_Call : IDisposable
9      {
10         // e.g. first column of first row in the result set
11         T Single<T>(string procedurename, DynamicParameters param = null);
12         // execute something to the database but not retrieve anything
13         void Execute(string procedurename, DynamicParameters param = null);
14         // retrieves the complete row or record
15         T OneRecord<T>(string procedurename, DynamicParameters param = null);
16         // get all of the rows
17         IEnumerable<T> List<T>(string procedurename, DynamicParameters param = null);
18         // stored procedure that returns two tables
19         Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedurename, DynamicParameters param = null);
20     }
21 }
22
23

```

The Solution Explorer on the right shows the project structure with four projects: `AndrewsBooks.DataAccess`, `Dependencies`, `Data`, and `AndrewsBookStore`. The `AndrewsBookStore` project contains files like `CategoryRepository.cs`, `Repository.cs`, `IRepository.cs`, `ApplicationDbContext.cs`, and `Category.cs`. The `Properties` and `Git Changes` tabs are also visible.

Solution Explorer

Search Solution Explorer (Ctrl+F)

ASP.NET Core

Live Share

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help

AndrewsBookStore

ISP\_Call.cs ICategoryRepository.cs CategoryRepository.cs Repository.cs IRepository.cs ApplicationDbContext.cs Category.cs

AndrewsBooks.DataAccess

```
1  using Dapper;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace AndrewsBooks.DataAccess.Repository.IRepository
7  {
8      public interface ISP_Call : IDisposable
9      {
10          // e.g. first column of first row in the result set
11          T Single<T>(string procedurename, DynamicParameters param = null);
12          // execute something to the database but not retrieve anything
13          void Execute(string procedurename, DynamicParameters param = null);
14          // retrieves the complete row or record
15          T OneRecord<T>(string procedurename, DynamicParameters param = null);
16          // get all of the rows
17          IEnumerable<T> List<T>(string procedurename, DynamicParameters param = null);
18          // stored procedure that returns two tables
19          Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedurename, DynamicParameters param = null);
20      }
21  }
```

130% No issues found

Output

Show output from: Build

Error List Web Publish Activity Package Manager Console Output

Item(s) Saved

AndrewsBookStore master 2

- Add a new class (SP\_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP\_Call interface

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. The toolbar contains icons for file operations like Open, Save, and Print.

The code editor displays the file `ISP_Call.cs` which implements the `ISP_Call` interface. The interface definition is as follows:

```

using Dapper;
using System;
using System.Collections.Generic;
using System.Text;

namespace AndrewsBooks.DataAccess.Repository.IRepository
{
    public interface ISP_Call : IDisposable
    {
        // e.g. first column of first row in the result set
        T Single<T>(string procedurename, DynamicParameters param = null);
        // execute something to the database but not retrieve anything
        void Execute(string procedurename, DynamicParameters param = null);
        // retrieves the complete row or record
        T OneRecord<T>(string procedurename, DynamicParameters param = null);
        // get all of the rows
        IEnumerable<T> List<T>(string procedurename, DynamicParameters param = null);
        // stored procedure that returns two tables
        Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedurename, DynamicParameters param = null);
    }
}

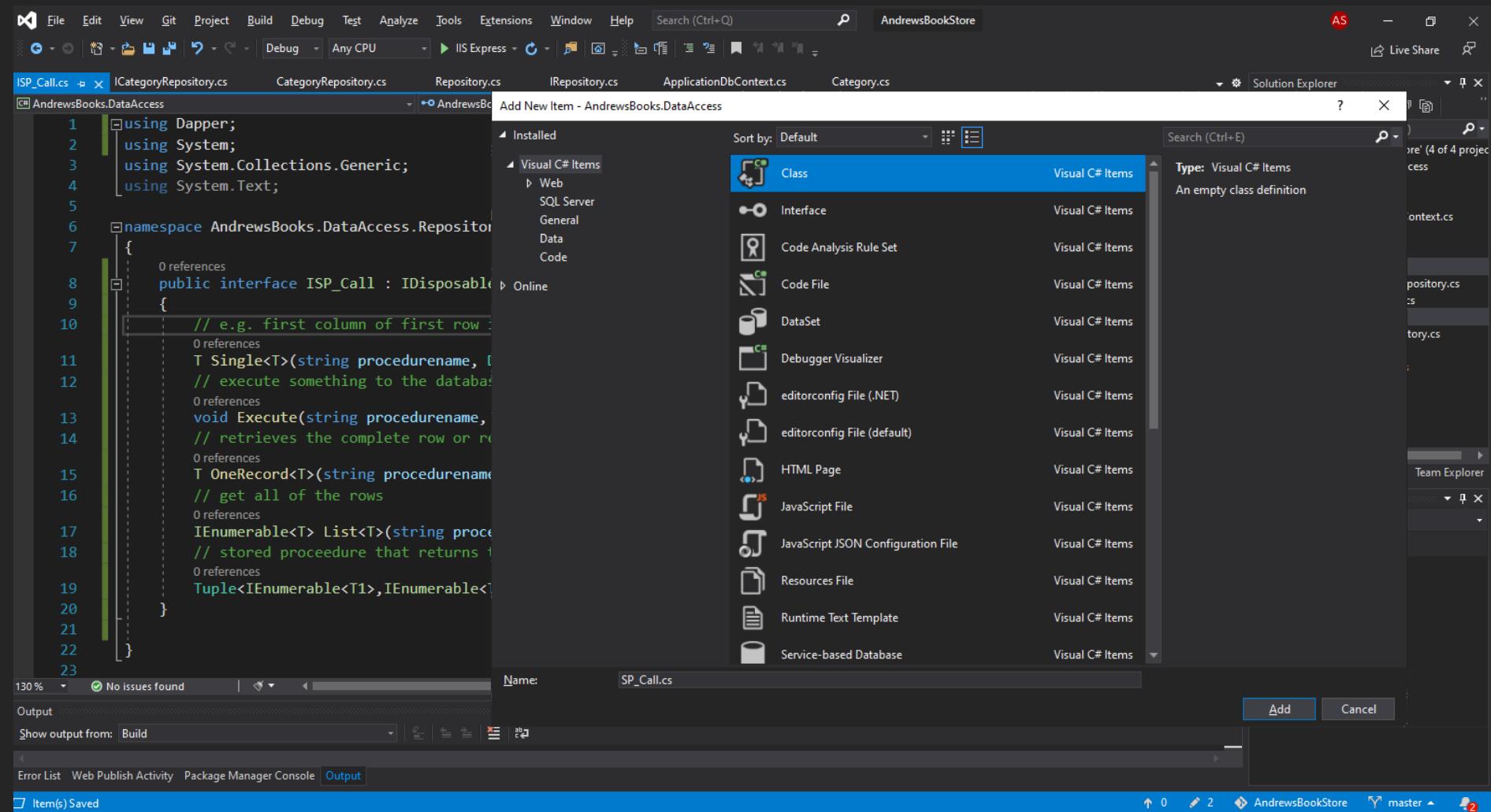
```

The Solution Explorer on the right shows the project structure:

- Solution 'AndrewsBookStore' (4 of 4 projects)
  - AndrewsBooks.DataAccess**
    - Dependencies
    - Data
      - ApplicationDbContext.cs
    - Migrations
    - Repository
      - ICategoryRepository.cs
      - IRepository.cs
      - ISP\_Call.cs**
      - CategoryRepository.cs
      - Repository.cs
  - AndrewsBooks.Models
  - AndrewsBooks.Utility
  - AndrewsBookStore

The Properties, Output, Error List, Web Publish Activity, Package Manager Console, and Live Share panes are also visible at the bottom of the 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

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `SP_Call.cs` in the `AndrewsBooks.DataAccess.Repository` namespace. The code implements the `ISP_Call` interface, which is highlighted with a red squiggly underline. A tooltip box is open over the `ISP_Call` interface reference, listing several CS0535 errors related to method implementations that have not been implemented yet. The Solution Explorer on the right shows the project structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project contains files like `ApplicationDbContext.cs`, `CategoryRepository.cs`, `Repository.cs`, and `ISP_Call.cs`.

```
1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace AndrewsBooks.DataAccess.Repository
7  {
8      public class SP_Call : ISP_Call
9      {
10  }
11 }
12 
```

CS0535: 'SP\_Call' does not implement interface member 'ISP\_Call.Single<T>(string, DynamicParameters)'  
CS0535: 'SP\_Call' does not implement interface member 'ISP\_Call.Execute(string, DynamicParameters)'  
CS0535: 'SP\_Call' does not implement interface member 'ISP\_Call.OneRecord<T>(string, DynamicParameters)'  
CS0535: 'SP\_Call' does not implement interface member 'ISP\_Call.List<T>(string, DynamicParameters)'  
CS0535: 'SP\_Call' does not implement interface member 'ISP\_Call.List<T1, T2>(string, DynamicParameters)'  
CS0535: 'SP\_Call' does not implement interface member 'IDisposable.Dispose()'

Show potential fixes (Alt+Enter or Ctrl+.)

- Add a new class (SP\_Call.cs) in the Repository folder, select the appropriate using statement and implement the ISP\_Call interface

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `SP_Call.cs` in the `AndrewsBooks.DataAccess.Repository` namespace. The code implements the `ISP_Call` interface, which is highlighted in blue. A tooltip from the IDE provides four options for implementing the interface: "Implement interface", "Implement interface with Dispose pattern", "Implement all members explicitly", and "Implement interface explicitly with Dispose pattern". A red error icon is present in the margin, indicating a CS0535 error: "'SP\_Call' does not implement interface member 'ISP\_CallSingle<T>(string, DynamicParameters)'". The Solution Explorer on the right shows the project structure, including `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and the `AndrewsBookStore` project itself. The `Repository` folder contains `IRepository.cs`, `ICategoryRepository.cs`, `ISP_Call.cs`, `CategoryRepository.cs`, `Repository.cs`, `IRepository.cs`, `ApplicationDbContext.cs`, and `Category.cs`.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace AndrewsBooks.DataAccess.Repository
7  {
8      public class SP_Call : ISP_Call
9      {
10  }
11 }

```

**CS0535** 'SP\_Call' does not implement interface member 'ISP\_CallSingle<T>(string, DynamicParameters)'

```

using AndrewsBooks.DataAccess.Repository.IRepository;
using Dapper;
using System;
Lines 9 to 10
{
    public void Dispose()
    {
        throw new NotImplementedException();
    }

    public void Execute(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public IEnumerable<T> List<T>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public T OneRecord<T>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public T Single<T>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }
}

```

- 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

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `SP_Call.cs` in the `AndrewsBooks.DataAccess.Repository` namespace. The code implements the `ISP_Call` interface, which is highlighted in blue. A tooltip from the IDE provides four options for implementing the interface: "Implement interface", "Implement interface with Dispose pattern", "Implement all members explicitly", and "Implement interface explicitly with Dispose pattern". A red error icon is visible next to the first line of the code, indicating a CS0535 error: "'SP\_Call' does not implement interface member 'ISP\_CallSingle<T>(string, DynamicParameters)'". The Solution Explorer on the right shows the project structure for 'AndrewsBookStore' with files like `ApplicationContext.cs`, `Category.cs`, and `CategoryRepository.cs`. The Output window at the bottom shows the message "Ready".

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace AndrewsBooks.DataAccess.Repository
7  {
8      public class SP_Call : ISP_Call
9      {
10  }
11 }

```

**CS0535** 'SP\_Call' does not implement interface member 'ISP\_CallSingle<T>(string, DynamicParameters)'

```

using AndrewsBooks.DataAccess.Repository.IRepository;
using Dapper;
using System;
Lines 9 to 10
{
    public void Dispose()
    {
        throw new NotImplementedException();
    }

    public void Execute(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public IEnumerable<T> List<T>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public T OneRecord<T>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }

    public T Single<T>(string procedurename, DynamicParameters param = null)
    {
        throw new NotImplementedException();
    }
}

```

- 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

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code editor with the file `SP_Call.cs` open. The code implements the `ISP_Call` interface, using `Dapper` and `Microsoft.EntityFrameworkCore` libraries. The `Solution Explorer` pane on the right shows the project structure, including files like `CategoryRepository.cs`, `Repository.cs`, and `ApplicationDbContext.cs`. The `Properties` pane is also visible.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBookStore.DataAccess.Data;
3  using Dapper;
4  using Microsoft.EntityFrameworkCore;
5  using System;
6  using System.Collections.Generic;
7  using System.Text;
8
9  namespace AndrewsBooks.DataAccess.Repository
10 {
11     public class SP_Call : ISP_Call
12     {
13         // access the database
14         private readonly ApplicationDbContext _db;
15         private static string ConnectionString = ""; // needed to called the stored procedures
16
17         // constructor to open a SQL connection
18         public SP_Call(ApplicationDbContext db)
19         {
20             _db = db;
21             ConnectionString = db.Database.GetConnectionString().ConnectionString;
22         }
23
24         // implements the ISP_Call interface
25         public void Dispose()
    }

```

- 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

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. The toolbar has icons for file operations like Open, Save, and Build.

The code editor displays the file `SP_Call.cs` with the following content:

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBookStore.DataAccess.Data;
3  using Dapper;
4  using Microsoft.EntityFrameworkCore;
5  using System;
6  using System.Collections.Generic;
7  using System.Text;
8
9  namespace AndrewsBooks.DataAccess.Repository
10 {
11     public class SP_Call : ISP_Call
12     {
13         // access the database
14         private readonly ApplicationDbContext _db;
15         private static string ConnectionString = ""; // needed to called the stored procedures
16
17         // constructor to open a SQL connection
18         public SP_Call(ApplicationDbContext db)
19         {
20             _db = db;
21             ConnectionString = db.Database.GetConnectionString().ConnectionString;
22         }
23
24         // implements the ISP_Call interface
25         public void Dispose()
    
```

The Solution Explorer on the right shows the project structure:

- Solution 'AndrewsBookStore' (4 of 4 projects)
  - AndrewsBooks.DataAccess
    - Dependencies
    - Data
      - ApplicationDbContext.cs
    - Migrations
    - Repository
      - IRepository
      - ICategoryRepository.cs
      - IRepository.cs
      - ISP\_Call.cs
      - CategoryRepository.cs
      - Repository.cs
      - SP\_Call.cs
  - AndrewsBooks.Models
  - AndrewsBooks.Utility
  - AndrewsBookStore

The Properties, Solution Explorer, Git Changes, and Team Explorer tabs are also visible at the bottom of the 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

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code editor with the file `SP_Call.cs` open. The code implements the `ISP_Call` interface, containing methods `Dispose()`, `Execute(string)`, and `IEnumerable<T> List<T>(string)`. The `Execute` method uses a `SqlConnection` to execute a stored procedure, while the `List` method uses `sqlCon.Query<T>`. The Solution Explorer on the right shows the project structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBooks.DataAccess` project contains files like `ApplicationDbContext.cs`, `CategoryRepository.cs`, `Repository.cs`, and `ISP_Call.cs`. The `AndrewsBookStore` project is the currently selected one. The status bar at the bottom indicates the file is saved.

```

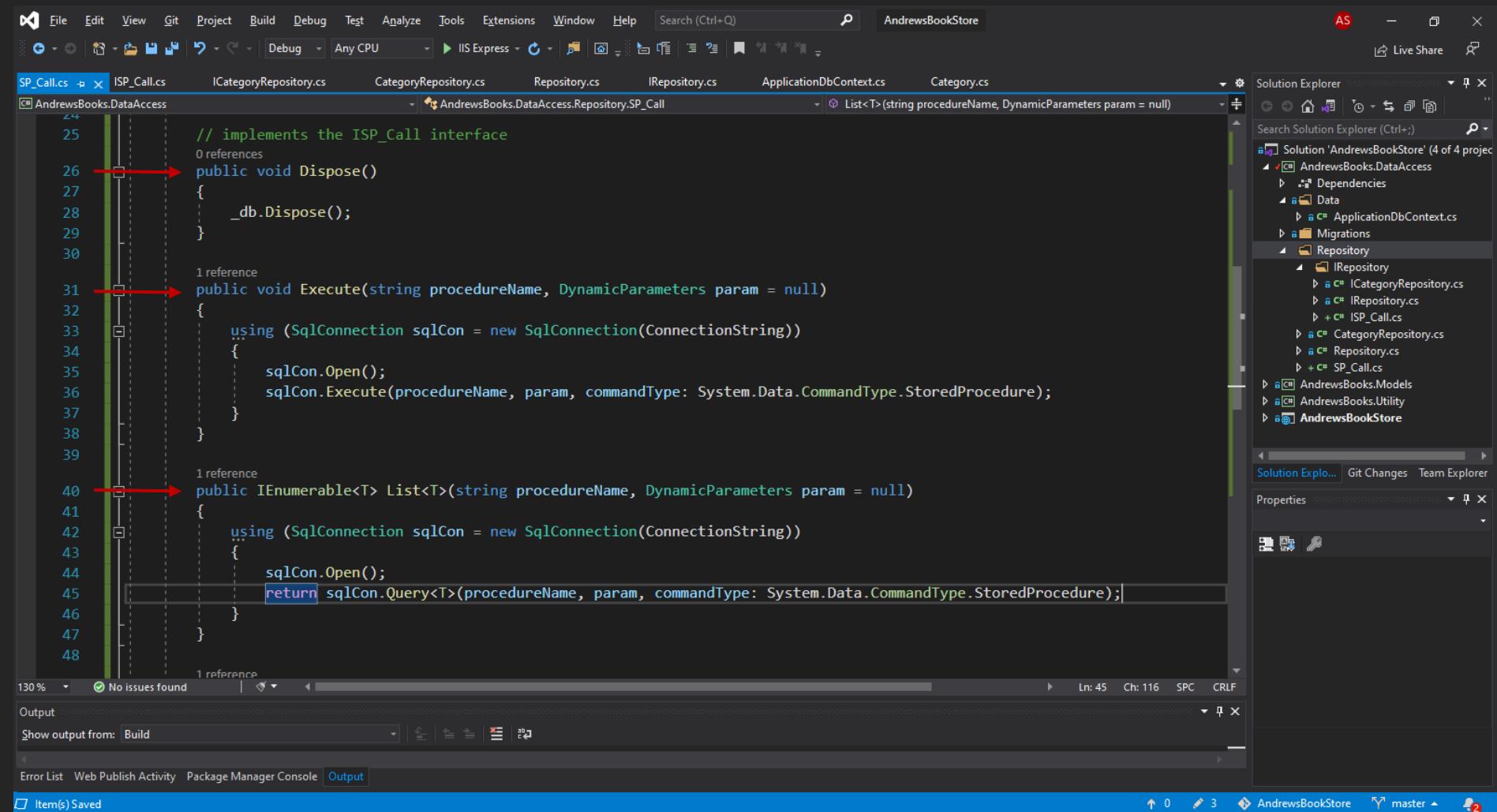
// implements the ISP_Call interface
public void Dispose()
{
    _db.Dispose();
}

public void Execute(string procedureName, DynamicParameters param = null)
{
    using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    {
        sqlCon.Open();
        sqlCon.Execute(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
    }
}

public IEnumerable<T> List<T>(string procedureName, DynamicParameters param = null)
{
    using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    {
        sqlCon.Open();
        return sqlCon.Query<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
    }
}

```

- 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



The screenshot shows the Microsoft Visual Studio IDE with the code editor open to the file `SP_Call.cs`. The code implements the `ISP_Call` interface. Three red arrows point to specific parts of the code:

- A red arrow points to the `Dispose()` method, which calls `_db.Dispose();`
- A red arrow points to the `Execute(string procedureName, DynamicParameters param = null)` method, which uses a `SqlConnection` to execute a stored procedure.
- A red arrow points to the `List<T>(string procedureName, DynamicParameters param = null)` method, which uses a `SqlConnection` to query the database.

```

// implements the ISP_Call interface
public void Dispose()
{
    _db.Dispose();
}

public void Execute(string procedureName, DynamicParameters param = null)
{
    using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    {
        sqlCon.Open();
        sqlCon.Execute(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
    }
}

public IEnumerable<T> List<T>(string procedureName, DynamicParameters param = null)
{
    using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    {
        sqlCon.Open();
        return sqlCon.Query<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
    }
}

```

The Solution Explorer on the right shows the project structure for `AndrewsBookStore`, including `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, and `AndrewsBooks.Utility`.

- 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

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The toolbar has icons for file operations like Open, Save, and Build.

The code editor displays the file `SP_Call.cs` from the `AndrewsBooks.DataAccess` namespace. The code implements the `ISP_Call` interface, which includes methods for executing stored procedures. The `List<T1, T2>` method returns a tuple of two lists, while the `OneRecord<T>` method throws a `NotImplementedException`.

```

    SP_Call.cs -> ISP_Call.cs      ICategoryRepository.cs      CategoryRepository.cs      Repository.cs      IRepository.cs      ApplicationDbContext.cs      Category.cs
    AndrewsBooks.DataAccess      <-- AndrewsBooks.DataAccess.Repository.SP_Call      <-- List<T1, T2>(string procedureName, DynamicParameters param = null)
    48
    49
    50     public Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedureName, DynamicParameters param = null)
    51     {
    52         using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    53         {
    54             sqlCon.Open();
    55             var result = SqlMapper.QueryMultiple(sqlCon, procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
    56             var item1 = result.Read<T1>().ToList(); // make sure to add using statement for LINQ
    57             var item2 = result.Read<T2>().ToList();
    58
    59             if(item1 != null && item2 != null)
    60             {
    61                 return new Tuple<IEnumerable<T1>, IEnumerable<T2>>(item1, item2);
    62             }
    63         }
    64
    65         return new Tuple<IEnumerable<T1>, IEnumerable<T2>>(new List<T1>(), new List<T2>());
    66     }
    67
    68     public T OneRecord<T>(string procedurename, DynamicParameters param = null)
    69     {
    70         throw new NotImplementedException();
    71     }
    72
  
```

The Solution Explorer on the right shows the project structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `AndrewsBookStore` project contains files like `ApplicationDbContext.cs`, `Category.cs`, `CategoryRepository.cs`, `IRepository.cs`, `ISP_Call.cs`, and `Repository.cs`.

- 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

```

48 }
49 }
50 public Tuple<IEnumerable<T1>, IEnumerable<T2>> List<T1, T2>(string procedureName, DynamicParameters param = null)
51 {
52     using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
53     {
54         sqlCon.Open();
55         var result = SqlMapper.QueryMultiple(sqlCon, procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
56         var item1 = result.Read<T1>().ToList(); // make sure to add using statement for LINQ
57         var item2 = result.Read<T2>().ToList();
58
59         if(item1 != null && item2 != null)
60         {
61             return new Tuple<IEnumerable<T1>, IEnumerable<T2>>(item1, item2);
62         }
63     }
64
65     return new Tuple<IEnumerable<T1>, IEnumerable<T2>>(new List<T1>(), new List<T2>());
66 }
67
68 public T OneRecord<T>(string procedurename, DynamicParameters param = null)
69 {
70     throw new NotImplementedException();
71 }
72

```

No issues found

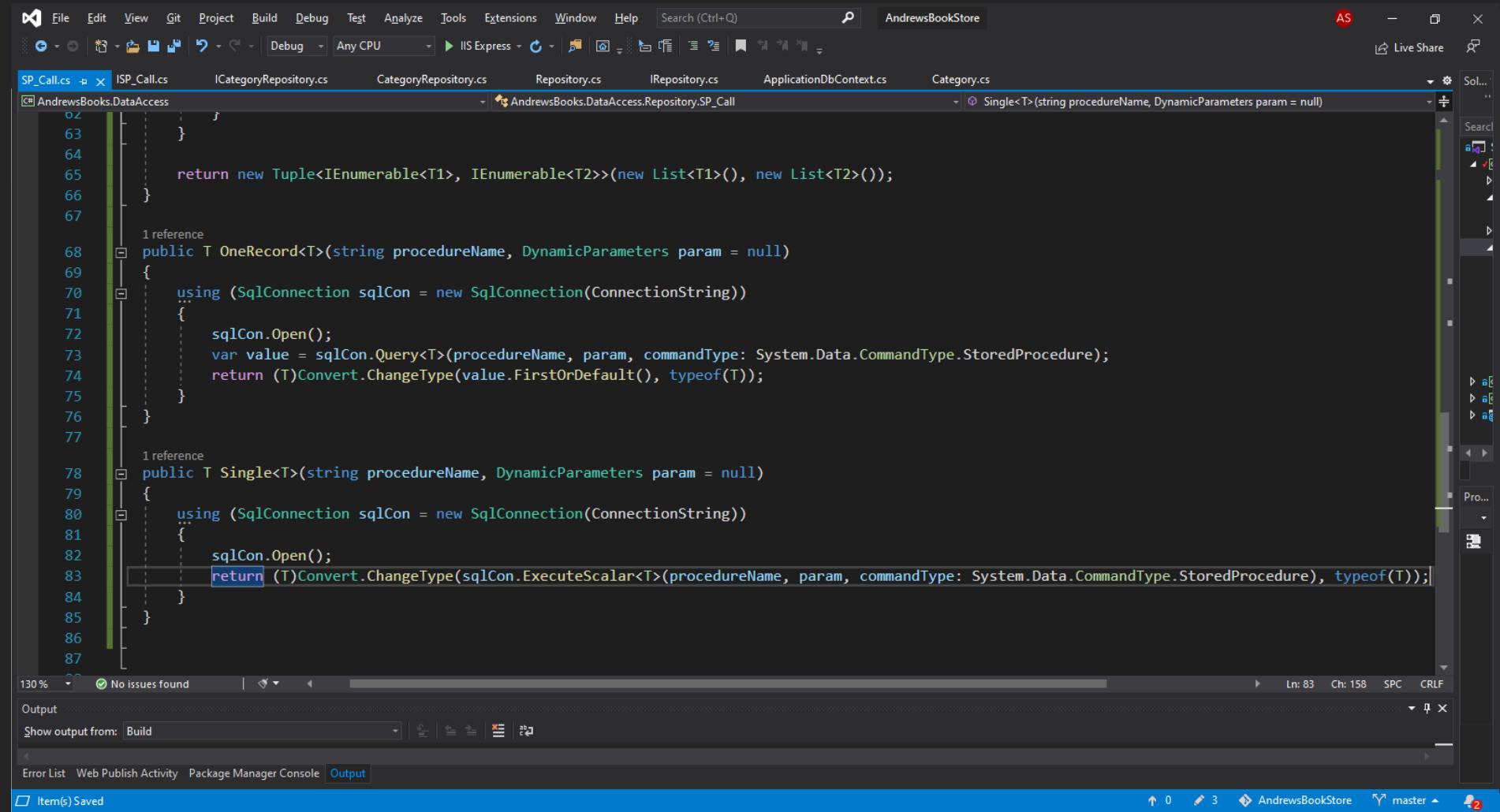
Show output from: Build

Error List Web Publish Activity Package Manager Console Output

Item(s) Saved

AndrewsBookStore

- 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



```

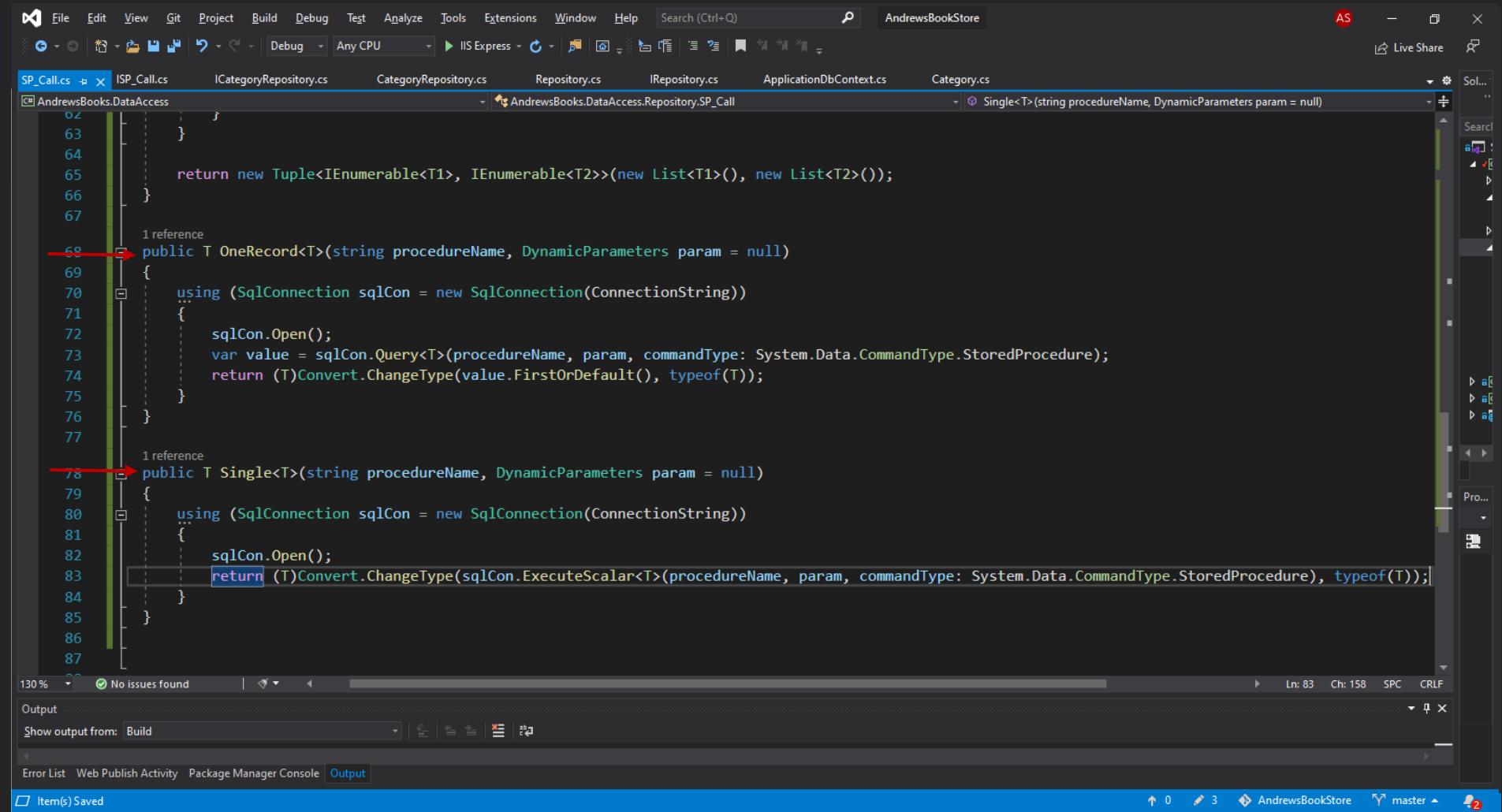
    SP_Call.cs  ISP_Call.cs  ICategoryRepository.cs  CategoryRepository.cs  Repository.cs  IRepository.cs  ApplicationDbContext.cs  Category.cs
    AndrewsBooks.DataAccess  ↳ AndrewsBooks.DataAccess.Repository.SP_Call
    1 reference
    public T OneRecord<T>(string procedureName, DynamicParameters param = null)
    {
        using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
        {
            sqlCon.Open();
            var value = sqlCon.Query<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
            return (T)Convert.ChangeType(value.FirstOrDefault(), typeof(T));
        }
    }

    1 reference
    public T Single<T>(string procedureName, DynamicParameters param = null)
    {
        using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
        {
            sqlCon.Open();
            return (T)Convert.ChangeType(sqlCon.ExecuteScalar<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure), typeof(T));
        }
    }

```

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The toolbar has icons for file operations like Open, Save, and Print. The solution explorer on the right shows files like SP\_Call.cs, ISP\_Call.cs, ICategoryRepository.cs, CategoryRepository.cs, Repository.cs, IRepository.cs, ApplicationDbContext.cs, and Category.cs. The code editor displays the SP\_Call.cs file, which contains C# code implementing the ISP\_Call interface. The code uses SqlConnection and SqlCommand to execute stored procedures. The status bar at the bottom shows "130 %", "No issues found", "Ln: 83 Ch: 158 SPC CRLF", and "Output". The output window shows "Show output from: Build". The bottom navigation bar includes Error List, Web Publish Activity, Package Manager Console, and Output, with Output being the active tab. A blue status bar at the very bottom indicates "Item(s) Saved".

- 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



The screenshot shows the Microsoft Visual Studio IDE with the code editor open to the `SP_Call.cs` file. The code implements the `ISP_Call` interface with two main methods: `OneRecord` and `Single`. Both methods use a `SqlConnection` to execute stored procedures and convert the results into the specified type using `Convert.ChangeType`.

```

    public T OneRecord<T>(string procedureName, DynamicParameters param = null)
    {
        using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
        {
            sqlCon.Open();
            var value = sqlCon.Query<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
            return (T)Convert.ChangeType(value.FirstOrDefault(), typeof(T));
        }
    }

    public T Single<T>(string procedureName, DynamicParameters param = null)
    {
        using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
        {
            sqlCon.Open();
            return (T)Convert.ChangeType(sqlCon.ExecuteScalar<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure), typeof(T));
        }
    }

```

- 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

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

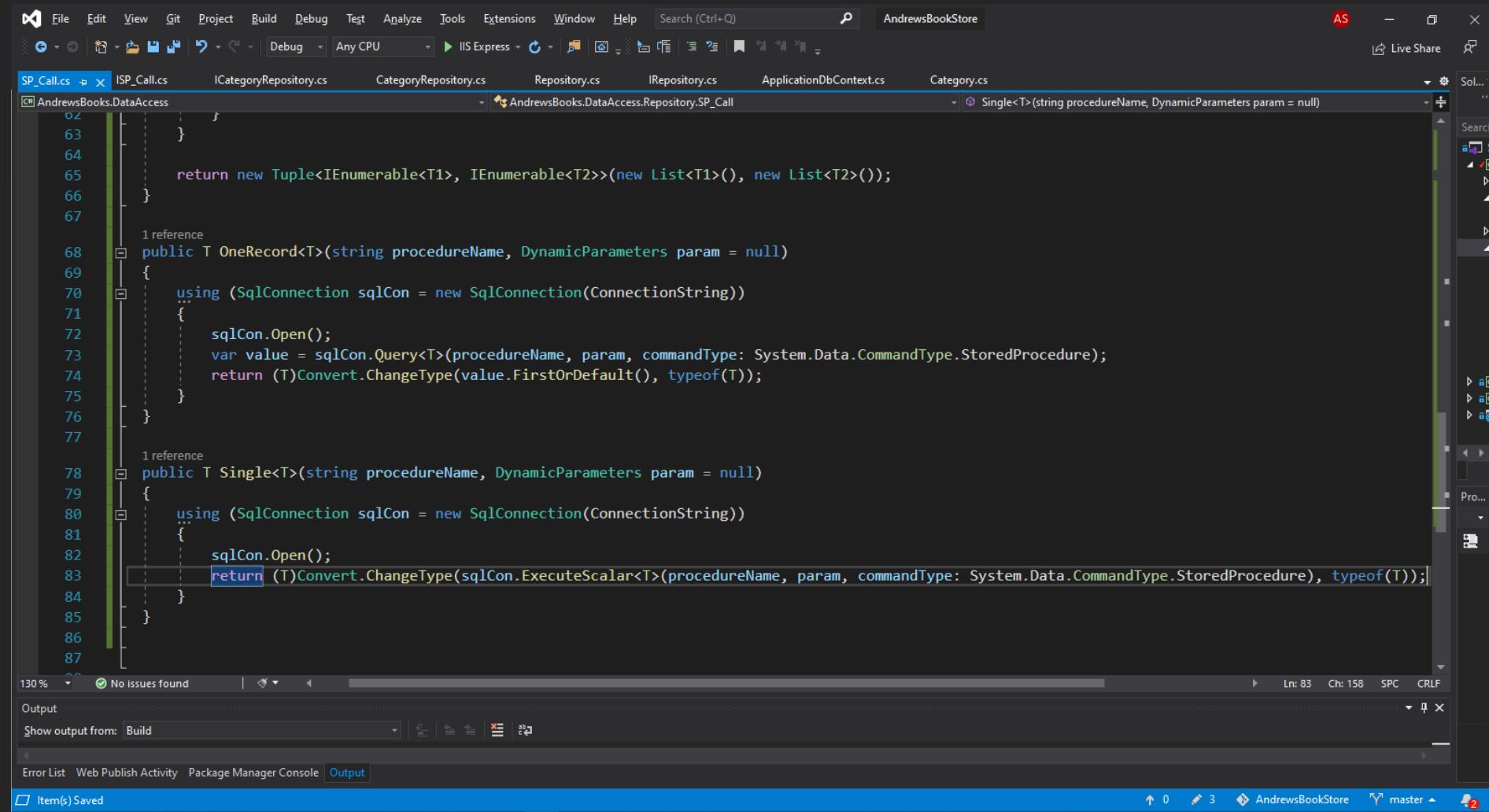
- Menu Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Toolbars:** Standard toolbar with icons for New, Open, Save, Print, etc.
- Project Explorer:** Shows files like SP\_Call.cs, ISP\_Call.cs, ICategoryRepository.cs, CategoryRepository.cs, Repository.cs, IRepository.cs, ApplicationDbContext.cs, and Category.cs.
- Code Editor:** The main window displays the `SP_Call.cs` file under the namespace `AndrewsBooks.DataAccess`. The code implements two methods: `OneRecord<T>` and `Single<T>`, both using `SqlCommand` to execute stored procedures.
- Annotations:** Two red arrows point to the method signatures at lines 68 and 78, highlighting them.
- Status Bar:** Shows zoom level (130%), issues found (0), line (Ln: 83), character (Ch: 158), and file endings (SPC, CRLF).
- Output Tab:** Shows "No issues found".
- Bottom Navigation:** Error List, Web Publish Activity, Package Manager Console, Output (selected), and Item(s) Saved.

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- Menu Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Toolbar:** Standard icons for file operations like Open, Save, Print, etc.
- Solution Explorer:** Shows the project structure with files like SP\_Call.cs, ISP\_Call.cs, ICategoryRepository.cs, CategoryRepository.cs, Repository.cs, IRepository.cs, ApplicationDbContext.cs, and Category.cs.
- Code Editor:** The main window displays the code for `SP_Call.cs`. The current method being edited is `Single<T>(string procedureName, DynamicParameters param = null)`. The code uses `ExecuteScalar<T>` to return a single record. A tooltip for `ExecuteScalar<T>` is visible, stating: "Single<T> (string procedureName, DynamicParameters param = null)".
- Status Bar:** Shows the zoom level (130%), search results ("No issues found"), line and character counts (Ln: 83 Ch: 158), and file endings (CRLF).
- Output Window:** Shows the output from the build process.
- Bottom Status Bar:** Shows the current branch name as "master".

\* Errata > The Interface file name is - IUnitOfWork

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



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. The toolbar has icons for file operations like Open, Save, and Build. The status bar at the bottom shows "130 %", "No issues found", "Ln: 83 Ch: 158 SPC CRLF", and a blue bar with "Item(s) Saved".

The code editor displays the `SP_Call.cs` file. The code implements two methods: `OneRecord<T>` and `Single<T>`. Both methods use a `SqlConnection` to execute stored procedures. The `OneRecord` method returns a tuple of two lists, while the `Single` method returns a single object of type `T`.

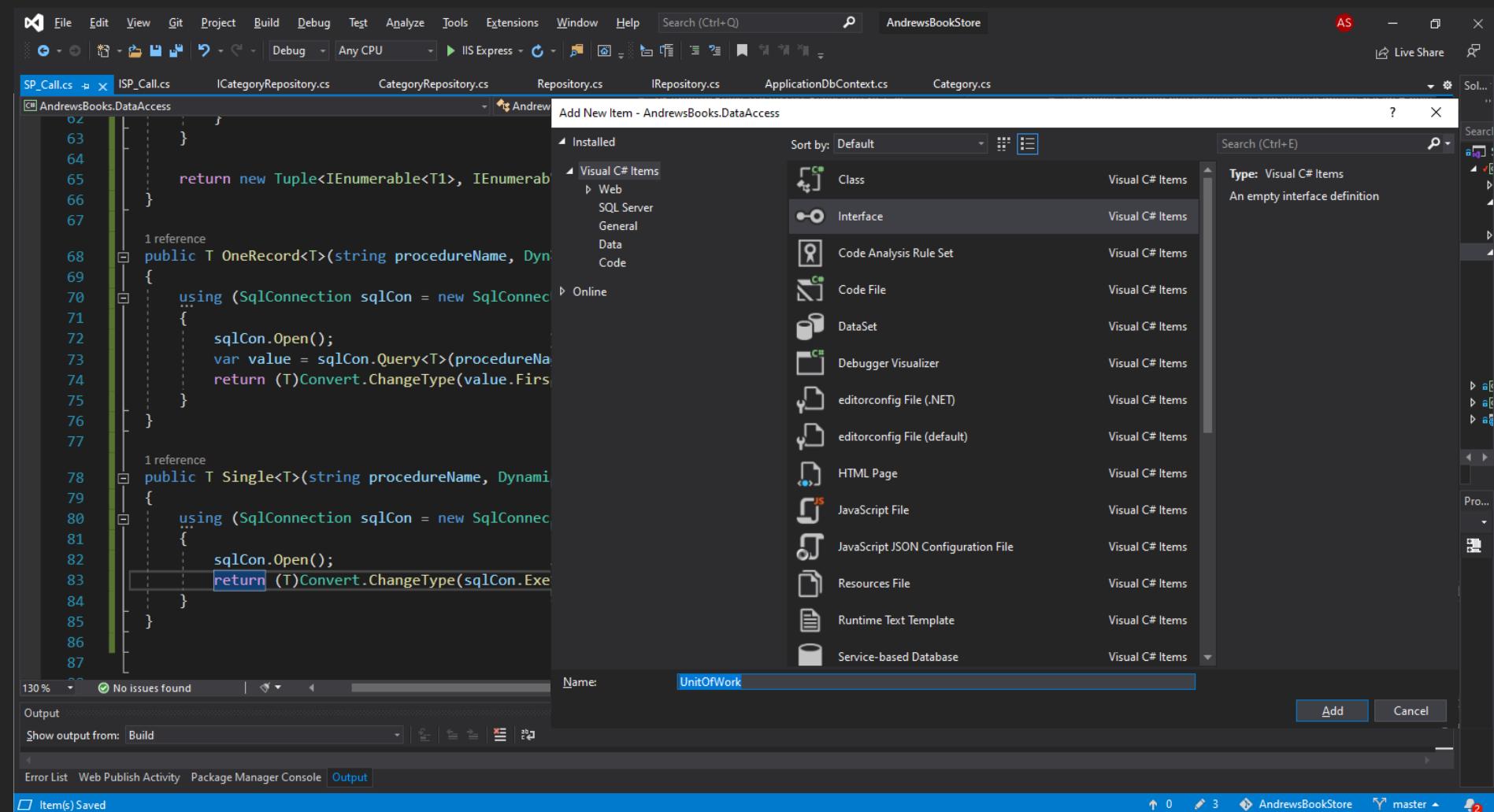
```
SP_Call.cs
...
    return new Tuple<IEnumerable<T1>, IEnumerable<T2>>(new List<T1>(), new List<T2>());
}

1 reference
public T OneRecord<T>(string procedureName, DynamicParameters param = null)
{
    using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    {
        sqlCon.Open();
        var value = sqlCon.Query<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure);
        return (T)Convert.ChangeType(value.FirstOrDefault(), typeof(T));
    }
}

1 reference
public T Single<T>(string procedureName, DynamicParameters param = null)
{
    using (SqlConnection sqlCon = new SqlConnection(ConnectionString))
    {
        sqlCon.Open();
        return (T)Convert.ChangeType(sqlCon.ExecuteScalar<T>(procedureName, param, commandType: System.Data.CommandType.StoredProcedure), typeof(T));
    }
}
```

\* 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.

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the file `UnitOfWork.cs` from the project `AndrewsBooks.DataAccess`. The code defines an interface `IUnitOfWork` within the namespace `AndrewsBooks.DataAccess.Repository.IRepository`. The Solution Explorer panel on the right shows the solution structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `IRepository` folder under `AndrewsBooks.DataAccess` contains the `IUnitOfWork.cs` file, which is currently selected. The Output and Error List panes at the bottom are visible, and the status bar at the bottom right indicates the repository name and branch.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace AndrewsBooks.DataAccess.Repository.IRepository
6 {
7     interface IUnitOfWork
8     {
9     }
10}
11
```

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

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

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the file `UnitOfWork.cs` with the following content:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository.IRepository
6  {
7      0 references
8      interface UnitOfWork : IDisposable
9      {
10         0 references
11         ICategoryRepository Category { get; }
12         0 references
13         ISP_Call SP_Call { get; }
14     }
15 }
```

The Solution Explorer on the right shows the project structure for 'AndrewsBookStore' (4 of 4 projects):

- AndrewsBooks.DataAccess
  - Dependencies
  - Data
    - ApplicationDbContext.cs
  - Migrations
  - Repository
    - ICategoryRepository.cs
    - IRepository.cs
    - ISP\_Call.cs
    - UnitOfWork.cs
  - CategoryRepository.cs
  - Repository.cs
  - SP\_Call.cs
- AndrewsBooks.Models
- AndrewsBooks.Utility
- AndrewsBookStore

The Properties window is also visible on the right side of the interface.

\* 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)

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search (Ctrl+Q) field. The toolbar contains various icons for file operations like Open, Save, and Build. The status bar at the bottom shows "130% No issues found" and "Ln: 10 Ch: 34 SPC CRLF".

The code editor window displays the `UnitOfWork.cs` file:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository.IRepository
6  {
7      interface IUnitOfWork : IDisposable
8      {
9          ICategoryRepository Category { get; }
10         ISP_Call SP_Call { get; }
11     }
12 }
13

```

The Solution Explorer window on the right shows the project structure for 'AndrewsBookStore' (4 of 4 projects). It includes `Dependencies`, `Data` (with `ApplicationDbContext.cs`), `Migrations`, `Repository` (with `IRepository`, `ICategoryRepository.cs`, `IRepository.cs`, `ISP_Call.cs`, and `UnitOfWork.cs`), `CategoryRepository.cs`, `Repository.cs`, `SP_Call.cs`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and the main `AndrewsBookStore` project.

\* 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)

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "AndrewsBookStore". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. The toolbar has icons for file operations like Open, Save, and Build.

The code editor displays `UnitOfWork.cs` from the `AndrewsBooks.DataAccess` project. The code defines a `UnitOfWork` class within the `AndrewsBooks.DataAccess.Repository` namespace:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository
6  {
7      class UnitOfWork
8      {
9      }
10 }
11

```

The Solution Explorer on the right shows the project structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `UnitOfWork.cs` file is selected in the `AndrewsBookStore` project. The Properties window shows the file properties for `UnitOfWork.cs`.

The Output window at the bottom indicates "No issues found".

\* 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)

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for `UnitOfWork.cs` in the `AndrewsBooks.DataAccess.Repository` namespace. The code defines a class `UnitOfWork` with a constructor and a method `GetRepository`. The Solution Explorer on the right shows the project structure, including the `IRepository` interface and its implementations like `ICategoryRepository.cs`, `CategoryRepository.cs`, etc. The Properties window on the far right shows the build action for `UnitOfWork.cs` is set to `C# compiler`.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace AndrewsBooks.DataAccess.Repository
6  {
7      0 references
8      class UnitOfWork
9      {
10  }
11

```

\* 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`)

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- Title Bar:** Shows "AndrewsBookStore" as the active project.
- Solution Explorer:** Displays the solution structure with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Toolbars:** Standard Visual Studio toolbars for file operations, search, and navigation.
- Code Editor:** The main window displays the `UnitOfWork.cs` file under the `AndrewsBooks.DataAccess` project. The code implements the `IUnitOfWork` interface with a constructor that injects an `ApplicationDbContext` and initializes repositories for `Category` and `SP_Call`.
- Status Bar:** Shows the current branch as "master" with a commit count of 2.

```

1  using AndrewsBooks.DataAccess.Repository.IRepository;
2  using AndrewsBookStore.DataAccess.Data;
3  using System;
4  using System.Collections.Generic;
5  using System.Text;
6
7  namespace AndrewsBooks.DataAccess.Repository
8  {
9      public class UnitOfWork // make the method public to access the class
10     {
11         private readonly ApplicationDbContext _db; // the using statement
12
13         public UnitOfWork(ApplicationDbContext db) // constructor to use DI and inject in to the repositories
14         {
15             _db = db;
16             Category = new CategoryRepository(_db);
17             SP_Call = new SP_Call(_db);
18         }
19
20         public ICategoryRepository Category { get; private set; }
21         public ISP_Call SP_Call { get; private set; }
22     }
23 }
24

```

\* 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)

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Solution Explorer:** Shows a solution named 'AndrewsBookStore' containing four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore. The AndrewsBookStore project is currently selected.
- Toolbars:** Standard toolbar, Solution Explorer toolbar.
- Status Bar:** Item(s) Saved, Line 0, Column 5, AndrewsBookStore, master, 2.
- Code Editor:** The UnitOfWork.cs file is open, showing the following C# code:

```

1  // make the method public to access the class
2
3  public class UnitOfWork // make the method public to access the class
4  {
5      private readonly ApplicationDbContext _db; // the using statement
6
7      public UnitOfWork(ApplicationDbContext db) // constructor to use DI and inject in to the repositories
8      {
9          _db = db;
10         Category = new CategoryRepository(_db);
11         SP_Call = new SP_Call(_db);
12     }
13
14     public ICategoryRepository Category { get; private set; }
15
16     public ISP_Call SP_Call { get; private set; }
17
18     public void Dispose()
19     {
20         _db.Dispose();
21     }
22
23     public void Save() // all changes will be saved when the Save method is called at the 'parent' level
24     {
25         _db.SaveChanges();
26     }
27 }
28

```

- Output Window:** Shows 'No issues found'.
- Error List:** Empty.

\* 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)

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Solution Explorer:** Shows the solution structure for 'AndrewsBookStore' with four projects: AndrewsBooks.DataAccess, AndrewsBooks.Models, AndrewsBooks.Utility, and AndrewsBookStore.
- Code Editor:** The current file is `UnitOfWork.cs` located in the `AndrewsBooks.DataAccess` project. The code implements the `IUnitOfWork` interface.
- Code Content:**

```

1  public class UnitOfWork // make the method public to access the class
2  {
3      private readonly ApplicationDbContext _db; // the using statement
4
5      public UnitOfWork(ApplicationDbContext db) // constructor to use DI and inject in to the repositories
6      {
7          _db = db;
8          Category = new CategoryRepository(_db);
9          SP_Call = new SP_Call(_db);
10     }
11
12     public ICategoryRepository Category { get; private set; }
13
14     public ISP_Call SP_Call { get; private set; }
15
16     public void Dispose()
17     {
18         _db.Dispose();
19     }
20
21     public void Save() // all changes will be saved when the Save method is called at the 'parent' level
22     {
23         _db.SaveChanges();
24     }
25
26 }
27
28
29
30
31
32
33
34

```
- Status Bar:** Shows the current branch is 'master' with 2 untracked files.

\* 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.

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- MenuBar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Toolbars:** Standard toolbar with icons for New, Open, Save, Print, etc.
- Code Editor:** The main window displays the `UnitOfWork.cs` file under the `AndrewsBooks.DataAccess` project. The code implements the `IUnitOfWork` interface with methods `Save()` and `Dispose()`.
- Solution Explorer:** Shows the solution structure with four projects: `AndrewsBooks.DataAccess`, `AndrewsBooks.Models`, `AndrewsBooks.Utility`, and `AndrewsBookStore`. The `Repository` folder contains `IRepository.cs`, `CategoryRepository.cs`, `ISP_Call.cs`, and `UnitOfWork.cs`.
- Properties:** A panel on the right showing properties for selected items.
- Status Bar:** Shows the current branch as `master` with two pending changes.

```

public class UnitOfWork // make the method public to access the class
{
    private readonly ApplicationDbContext _db; // the using statement

    public UnitOfWork(ApplicationDbContext db) // constructor to use DI and inject in to the repositories
    {
        _db = db;
        Category = new CategoryRepository(_db);
        SP_Call = new SP_Call(_db);
    }

    public ICategoryRepository Category { get; private set; }
    public ISP_Call SP_Call { get; private set; }

    public void Dispose()
    {
        _db.Dispose();
    }

    public void Save() // all changes will be saved when the Save method is called at the 'parent' level
    {
        _db.SaveChanges();
    }
}

```

\* 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.

```

19     {
20         2 references
21         public class Startup
22         {
23             0 references
24             public Startup(IConfiguration configuration)
25             {
26                 Configuration = configuration;
27             }
28
29             2 references
29             public IConfiguration Configuration { get; }
30
31             // This method gets called by the runtime. Use this method to add services to the container.
32             0 references
33             public void ConfigureServices(IServiceCollection services)
34             {
35                 services.AddDbContext<ApplicationDbContext>(options =>
36                     options.UseSqlServer(
37                         Configuration.GetConnectionString("DefaultConnection")));
38
39                 services.AddDatabaseDeveloperPageExceptionFilter();
40
41                 // removed 'options => options.SignIn.RequireConfirmedAccount = true'
42                 services.AddDefaultIdentity<IdentityUser>()
43                     .AddEntityFrameworkStores<ApplicationDbContext>();
44                 services.AddScoped<IUnitOfWork, UnitOfWork>();
45                 services.AddControllersWithViews();
46
47                 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
48             }
49
50             // ...
51         }
52     }
53
54     // ...
55
56     // ...
57
58     // ...
59
59     // ...
60
61     // ...
62
62     // ...
63
63     // ...
64
64     // ...
65
65     // ...
66
66     // ...
67
67     // ...
68
68     // ...
69
69     // ...
70
70     // ...
71
71     // ...
72
72     // ...
73
73     // ...
74
74     // ...
75
75     // ...
76
76     // ...
77
77     // ...
78
78     // ...
79
79     // ...
80
80     // ...
81
81     // ...
82
82     // ...
83
83     // ...
84
84     // ...
85
85     // ...
86
86     // ...
87
87     // ...
88
88     // ...
89
89     // ...
90
90     // ...
91
91     // ...
92
92     // ...
93
93     // ...
94
94     // ...
95
95     // ...
96
96     // ...
97
97     // ...
98
98     // ...
99
99     // ...
100
100     // ...
101
101     // ...
102
102     // ...
103
103     // ...
104
104     // ...
105
105     // ...
106
106     // ...
107
107     // ...
108
108     // ...
109
109     // ...
110
110     // ...
111
111     // ...
112
112     // ...
113
113     // ...
114
114     // ...
115
115     // ...
116
116     // ...
117
117     // ...
118
118     // ...
119
119     // ...
120
120     // ...
121
121     // ...
122
122     // ...
123
123     // ...
124
124     // ...
125
125     // ...
126
126     // ...
127
127     // ...
128
128     // ...
129
129     // ...
130
130     // ...
131
131     // ...
132
132     // ...
133
133     // ...
134
134     // ...
135
135     // ...
136
136     // ...
137
137     // ...
138
138     // ...
139
139     // ...
140
140     // ...
141
141     // ...
142
142     // ...
143
143     // ...
144
144     // ...
145
145     // ...
146
146     // ...
147
147     // ...
148
148     // ...
149
149     // ...
150
150     // ...
151
151     // ...
152
152     // ...
153
153     // ...
154
154     // ...
155
155     // ...
156
156     // ...
157
157     // ...
158
158     // ...
159
159     // ...
160
160     // ...
161
161     // ...
162
162     // ...
163
163     // ...
164
164     // ...
165
165     // ...
166
166     // ...
167
167     // ...
168
168     // ...
169
169     // ...
170
170     // ...
171
171     // ...
172
172     // ...
173
173     // ...
174
174     // ...
175
175     // ...
176
176     // ...
177
177     // ...
178
178     // ...
179
179     // ...
180
180     // ...
181
181     // ...
182
182     // ...
183
183     // ...
184
184     // ...
185
185     // ...
186
186     // ...
187
187     // ...
188
188     // ...
189
189     // ...
190
190     // ...
191
191     // ...
192
192     // ...
193
193     // ...
194
194     // ...
195
195     // ...
196
196     // ...
197
197     // ...
198
198     // ...
199
199     // ...
200
200     // ...
201
201     // ...
202
202     // ...
203
203     // ...
204
204     // ...
205
205     // ...
206
206     // ...
207
207     // ...
208
208     // ...
209
209     // ...
210
210     // ...
211
211     // ...
212
212     // ...
213
213     // ...
214
214     // ...
215
215     // ...
216
216     // ...
217
217     // ...
218
218     // ...
219
219     // ...
220
220     // ...
221
221     // ...
222
222     // ...
223
223     // ...
224
224     // ...
225
225     // ...
226
226     // ...
227
227     // ...
228
228     // ...
229
229     // ...
230
230     // ...
231
231     // ...
232
232     // ...
233
233     // ...
234
234     // ...
235
235     // ...
236
236     // ...
237
237     // ...
238
238     // ...
239
239     // ...
240
240     // ...
241
241     // ...
242
242     // ...
243
243     // ...
244
244     // ...
245
245     // ...
246
246     // ...
247
247     // ...
248
248     // ...
249
249     // ...
250
250     // ...
251
251     // ...
252
252     // ...
253
253     // ...
254
254     // ...
255
255     // ...
256
256     // ...
257
257     // ...
258
258     // ...
259
259     // ...
260
260     // ...
261
261     // ...
262
262     // ...
263
263     // ...
264
264     // ...
265
265     // ...
266
266     // ...
267
267     // ...
268
268     // ...
269
269     // ...
270
270     // ...
271
271     // ...
272
272     // ...
273
273     // ...
274
274     // ...
275
275     // ...
276
276     // ...
277
277     // ...
278
278     // ...
279
279     // ...
280
280     // ...
281
281     // ...
282
282     // ...
283
283     // ...
284
284     // ...
285
285     // ...
286
286     // ...
287
287     // ...
288
288     // ...
289
289     // ...
290
290     // ...
291
291     // ...
292
292     // ...
293
293     // ...
294
294     // ...
295
295     // ...
296
296     // ...
297
297     // ...
298
298     // ...
299
299     // ...
300
300     // ...
301
301     // ...
302
302     // ...
303
303     // ...
304
304     // ...
305
305     // ...
306
306     // ...
307
307     // ...
308
308     // ...
309
309     // ...
310
310     // ...
311
311     // ...
312
312     // ...
313
313     // ...
314
314     // ...
315
315     // ...
316
316     // ...
317
317     // ...
318
318     // ...
319
319     // ...
320
320     // ...
321
321     // ...
322
322     // ...
323
323     // ...
324
324     // ...
325
325     // ...
326
326     // ...
327
327     // ...
328
328     // ...
329
329     // ...
330
330     // ...
331
331     // ...
332
332     // ...
333
333     // ...
334
334     // ...
335
335     // ...
336
336     // ...
337
337     // ...
338
338     // ...
339
339     // ...
340
340     // ...
341
341     // ...
342
342     // ...
343
343     // ...
344
344     // ...
345
345     // ...
346
346     // ...
347
347     // ...
348
348     // ...
349
349     // ...
350
350     // ...
351
351     // ...
352
352     // ...
353
353     // ...
354
354     // ...
355
355     // ...
356
356     // ...
357
357     // ...
358
358     // ...
359
359     // ...
360
360     // ...
361
361     // ...
362
362     // ...
363
363     // ...
364
364     // ...
365
365     // ...
366
366     // ...
367
367     // ...
368
368     // ...
369
369     // ...
370
370     // ...
371
371     // ...
372
372     // ...
373
373     // ...
374
374     // ...
375
375     // ...
376
376     // ...
377
377     // ...
378
378     // ...
379
379     // ...
380
380     // ...
381
381     // ...
382
382     // ...
383
383     // ...
384
384     // ...
385
385     // ...
386
386     // ...
387
387     // ...
388
388     // ...
389
389     // ...
390
390     // ...
391
391     // ...
392
392     // ...
393
393     // ...
394
394     // ...
395
395     // ...
396
396     // ...
397
397     // ...
398
398     // ...
399
399     // ...
400
400     // ...
401
401     // ...
402
402     // ...
403
403     // ...
404
404     // ...
405
405     // ...
406
406     // ...
407
407     // ...
408
408     // ...
409
409     // ...
410
410     // ...
411
411     // ...
412
412     // ...
413
413     // ...
414
414     // ...
415
415     // ...
416
416     // ...
417
417     // ...
418
418     // ...
419
419     // ...
420
420     // ...
421
421     // ...
422
422     // ...
423
423     // ...
424
424     // ...
425
425     // ...
426
426     // ...
427
427     // ...
428
428     // ...
429
429     // ...
430
430     // ...
431
431     // ...
432
432     // ...
433
433     // ...
434
434     // ...
435
435     // ...
436
436     // ...
437
437     // ...
438
438     // ...
439
439     // ...
440
440     // ...
441
441     // ...
442
442     // ...
443
443     // ...
444
444     // ...
445
445     // ...
446
446     // ...
447
447     // ...
448
448     // ...
449
449     // ...
450
450     // ...
451
451     // ...
452
452     // ...
453
453     // ...
454
454     // ...
455
455     // ...
456
456     // ...
457
457     // ...
458
458     // ...
459
459     // ...
460
460     // ...
461
461     // ...
462
462     // ...
463
463     // ...
464
464     // ...
465
465     // ...
466
466     // ...
467
467     // ...
468
468     // ...
469
469     // ...
470
470     // ...
471
471     // ...
472
472     // ...
473
473     // ...
474
474     // ...
475
475     // ...
476
476     // ...
477
477     // ...
478
478     // ...
479
479     // ...
480
480     // ...
481
481     // ...
482
482     // ...
483
483     // ...
484
484     // ...
485
485     // ...
486
486     // ...
487
487     // ...
488
488     // ...
489
489     // ...
490
490     // ...
491
491     // ...
492
492     // ...
493
493     // ...
494
494     // ...
495
495     // ...
496
496     // ...
497
497     // ...
498
498     // ...
499
499     // ...
500
500     // ...
501
501     // ...
502
502     // ...
503
503     // ...
504
504     // ...
505
505     // ...
506
506     // ...
507
507     // ...
508
508     // ...
509
509     // ...
510
510     // ...
511
511     // ...
512
512     // ...
513
513     // ...
514
514     // ...
515
515     // ...
516
516     // ...
517
517     // ...
518
518     // ...
519
519     // ...
520
520     // ...
521
521     // ...
522
522     // ...
523
523     // ...
524
524     // ...
525
525     // ...
526
526     // ...
527
527     // ...
528
528     // ...
529
529     // ...
530
530     // ...
531
531     // ...
532
532     // ...
533
533     // ...
534
534     // ...
535
535     // ...
536
536     // ...
537
537     // ...
538
538     // ...
539
539     // ...
540
540     // ...
541
541     // ...
542
542     // ...
543
543     // ...
544
544     // ...
545
545     // ...
546
546     // ...
547
547     // ...
548
548     // ...
549
549     // ...
550
550     // ...
551
551     // ...
552
552     // ...
553
553     // ...
554
554     // ...
555
555     // ...
556
556     // ...
557
557     // ...
558
558     // ...
559
559     // ...
560
560     // ...
561
561     // ...
562
562     // ...
563
563     // ...
564
564     // ...
565
565     // ...
566
566     // ...
567
567     // ...
568
568     // ...
569
569     // ...
570
570     // ...
571
571     // ...
572
572     // ...
573
573     // ...
574
574     // ...
575
575     // ...
576
576     // ...
577
577     // ...
578
578     // ...
579
579     // ...
580
580     // ...
581
581     // ...
582
582     // ...
583
583     // ...
584
584     // ...
585
585     // ...
586
586     // ...
587
587     // ...
588
588     // ...
589
589     // ...
590
590     // ...
591
591     // ...
592
592     // ...
593
593     // ...
594
594     // ...
595
595     // ...
596
596     // ...
597
597     // ...
598
598     // ...
599
599     // ...
600
600     // ...
601
601     // ...
602
602     // ...
603
603     // ...
604
604     // ...
605
605     // ...
606
606     // ...
607
607     // ...
608
608     // ...
609
609     // ...
610
610     // ...
611
611     // ...
612
612     // ...
613
613     // ...
614
614     // ...
615
615     // ...
616
616     // ...
617
617     // ...
618
618     // ...
619
619     // ...
620
620     // ...
621
621     // ...
622
622     // ...
623
623     // ...
624
624     // ...
625
625     // ...
626
626     // ...
627
627     // ...
628
628     // ...
629
629     // ...
630
630     // ...
631
631     // ...
632
632     // ...
633
633     // ...
634
634     // ...
635
635     // ...
636
636     // ...
637
637     // ...
638
638     // ...
639
639     // ...
640
640     // ...
641
641     // ...
642
642     // ...
643
643     // ...
644
644     // ...
645
645     // ...
646
646     // ...
647
647     // ...
648
648     // ...
649
649     // ...
650
650     // ...
651
651     // ...
652
652     // ...
653
653     // ...
654
654     // ...
655
655     // ...
656
656     // ...
657
657     // ...
658
658     // ...
659
659     // ...
660
660     // ...
661
661     // ...
662
662     // ...
663
663     // ...
664
664     // ...
665
665     // ...
666
666     // ...
667
667     // ...
668
668     // ...
669
669     // ...
670
670     // ...
671
671     // ...
672
672     // ...
673
673     // ...
674
674     // ...
675
675     // ...
676
676     // ...
677
677     // ...
678
678     // ...
679
679     // ...
680
680     // ...
681
681     // ...
682
682     // ...
683
683     // ...
684
684     // ...
685
685     // ...
686
686     // ...
687
687     // ...
688
688     // ...
689
689     // ...
690
690     // ...
691
691     // ...
692
692     // ...
693
693     // ...
694
694     // ...
695
695     // ...
696
696     // ...
697
697     // ...
698
698     // ...
699
699     // ...
700
700     // ...
701
701     // ...
702
702     // ...
703
703     // ...
704
704     // ...
705
705     // ...
706
706     // ...
707
707     // ...
708
708     // ...
709
709     // ...
710
710     // ...
711
711     // ...
712
712     // ...
713
713     // ...
714
714     // ...
715
715     // ...
716
716     // ...
717
717     // ...
718
718     // ...
719
719     // ...
720
720     // ...
721
721     // ...
722
722     // ...
723
723     // ...
724
724     // ...
725
725     // ...
726
726     // ...
727
727     // ...
728
728     // ...
729
729     // ...
730
730     // ...
731
731     // ...
732
732     // ...
733
733     // ...
734
734     // ...
735
735     // ...
736
736     // ...
737
737     // ...
738
738     // ...
739
739     // ...
740
740     // ...
741
741     // ...
742
742     // ...
743
743     // ...
744
744     // ...
745
745     // ...
746
746     // ...
747
747     // ...
748
748     // ...
749
749     // ...
750
750     // ...
751
751     // ...
752
752     // ...
753
753     // ...
754
754     // ...
755
755     // ...
756
756     // ...
757
757     // ...
758
758     // ...
759
759     // ...
760
760     // ...
761
761     // ...
762
762     // ...
763
763     // ...
764
764     // ...
765
765     // ...
766
766     // ...
767
767     // ...
768
768     // ...
769
769     // ...
770
770     // ...
771
771     // ...
772
772     // ...
773
773     // ...
774
774     // ...
775
775     // ...
776
776     // ...
777
777     // ...
778
778     // ...
779
779     // ...
780
780     // ...
781
781     // ...
782
782     // ...
783
783     // ...
784
784     // ...
785
785     // ...
786
786     // ...
787
787     // ...
788
788     // ...
789
789     // ...
790
790     // ...
791
791     // ...
792
792     // ...
793
793     // ...
794
794     // ...
795
795     // ...
796
796     // ...
797
797     // ...
798
798     // ...
799
799     // ...
800
800     // ...
801
801     // ...
802
802     // ...
803
803     // ...
804
804     // ...
805
805     // ...
806
806     // ...
807
807     // ...
808
808     // ...
809
809     // ...
810
810     // ...
811
811     // ...
812
812     // ...
813
813     // ...
814
814     // ...
815
815     // ...
816
816     // ...
817
817     // ...
818
818     // ...
819
819     // ...
820
820     // ...
821
821     // ...
822
822     // ...
823
823     // ...
824
824     // ...
825
825     // ...
826
826     // ...
827
827     // ...
828
828     // ...
829
829     // ...
830
830     // ...
831
831     // ...
832
832     // ...
833
833     // ...
834
834     // ...
835
835     // ...
836
836     // ...
837
837     // ...
838
838     // ...
839
839     // ...
840
840     // ...
841
841     // ...
842
842     // ...
843
843     // ...
844
844     // ...
845
845     // ...
846
846     // ...
847
847     // ...
848
848     // ...
849
849     // ...
850
850     // ...
851
851     // ...
852
852     // ...
853
853     // ...
854
854     // ...
855
855     // ...
856
856     // ...
857
857     // ...
858
858     // ...
859
859     // ...
860
860     // ...
861
861     // ...
862
862     // ...
863
863     // ...
864
864     // ...
865
865     // ...
866
866     // ...
867
867     // ...
868
868     // ...
869
869     // ...
870
870     // ...
871
871     // ...
872
872     // ...
873
873     // ...
874
874     // ...
875
875     // ...
876
876     // ...
877
877     // ...
878
878     // ...
879
879     // ...
880
880     // ...
881
881     // ...
882
882     // ...
883
883     // ...
884
884     // ...
885
885     // ...
886
886     // ...
887
887     // ...
888
888     // ...
889
889     // ...
890
890     // ...
891
891     // ...
892
892     // ...
893
893     // ...
894
894     // ...
895
895     // ...
896
896     // ...
897
897     // ...
898
898     // ...
899
899     // ...
900
900     // ...
901
901     // ...
902
902     // ...
903
903     // ...
904
904     // ...
905
905     // ...
906
906     // ...
907
907     // ...
908
908     // ...
909
909     // ...
910
910     // ...
911
911     // ...
912
912     // ...
913
913     // ...
914
914     // ...
915
915     // ...
916
916     // ...
917
917     // ...
918
918     // ...
919
919     // ...
920
920     // ...
921
921     // ...
922
922     // ...
923
923     // ...
924
924     // ...
925
925     // ...
926
926     // ...
927
927     // ...
928
928     // ...
929
929     // ...
930
930     // ...
931
931     // ...
932
932     // ...
933
933     // ...
934
934     // ...
935
935     // ...
936
936     // ...
937
937     // ...
938
938     // ...
939
939     // ...
940
940     // ...
941
941     // ...
942
942     // ...
943
943     // ...
944
944     // ...
945
945     // ...
946
946     // ...
947
947     // ...
948
948     // ...
949
949     // ...
950
950     // ...
951
951     // ...
952
952     // ...
953
953     // ...
954
954     // ...
955
955     // ...
956
956     // ...
957
957     // ...
958
958     // ...
959
959     // ...
960
960     // ...
961
961     // ...
962
962     // ...
963
963     // ...
964
964     // ...
965
965     // ...
966
966     // ...
967
967     // ...
968
968     // ...
969
969     // ...
970
970     // ...
971
971     // ...
972
972     // ...
973
973     // ...
974
974     // ...
975
975     // ...
976
976     // ...
977
977     // ...
978
978     // ...
979
979     // ...
980
980     // ...
981
981     // ...
982
982     // ...
983
983     // ...
984
984     // ...
985
985     // ...
986
986     // ...
987
987     // ...
988
988     // ...
989
989     // ...
990
990     // ...
991
991     // ...
992
992     // ...
993
993     // ...
994
994     // ...
995
995     // ...
996
996     // ...
997
997     // ...
998
998     // ...
999
999     // ...
1000
1000     // ...
1001
1001     // ...
1002
1002     // ...
1003
1003     // ...
1004
1004     // ...
1005
1005     // ...
1006
1006     // ...
1007
1007     // ...
1008
1008     // ...
1009
1009     // ...
1010
1010     // ...
1011
1011     // ...
1012
1012     // ...
1013
1013     // ...
1014
1014     // ...
1015
1015     // ...
1016
1016     // ...
1017
1017     // ...
1018
1018     // ...
1019
1019     // ...
1020
1020     // ...
1021
1021     // ...
1022
1022     // ...
1023
1023     // ...
1024
1024     // ...
1025
1025     // ...
1026
1026     // ...
1027
1027     // ...
1028
1028     // ...
1029
1029     // ...
1030
1030     // ...
1031
1031     // ...
1032
1032     // ...
1033
1033     // ...
1034
1034     // ...
1035
1035     // ...
1036
1036     // ...
1037
1037     // ...
1038
1038     // ...
1039
1039     // ...
1040
1040     // ...
1041
1041     // ...
1042
1042     // ...
1043
1043     // ...
1044
1044     // ...
1045
1045     // ...
1046
1046     // ...
1047
1047     // ...
1048
1048     // ...
1049
1049     // ...
1050
1050     // ...
1051
1051     // ...
1052
1052     // ...
1053
1053     // ...
1054
1054     // ...
1055
1055     // ...
1056
1056     // ...
1057
1057     // ...
1058
1058     // ...
1059
1059     // ...
1060
1060     // ...
1061
1061     // ...
1062
1062     // ...
1063
1063     // ...
1064
1064     // ...
1065
1065     // ...
1066
1066     // ...
1067
1067     // ...
1068
1068     // ...
1069
1069     // ...
1070
1070     // ...
1071
1071     // ...
1072
1072     // ...
1073
1073     // ...
1074
1074     // ...
1075
1075     // ...
1076
1076     // ...
1077
1077     // ...
1078
1078     // ...
1079
1079     // ...
1080
1080     // ...
1081
1081     // ...
1082
1082     // ...
1083
1083     // ...
1084
1084     // ...
1085
1085     // ...
1086
1086     // ...
1087
1087     // ...
1088
1088     // ...
1089
1089     // ...
1090
1090     // ...
1091
1091     // ...
1092
1092     // ...
1093
1093     // ...
1094
1094     // ...
1095
1095     // ...
1096
1096     // ...
1097
1097     // ...
1098
1098     // ...
1099
1099     // ...
1100
1100     // ...
1
```