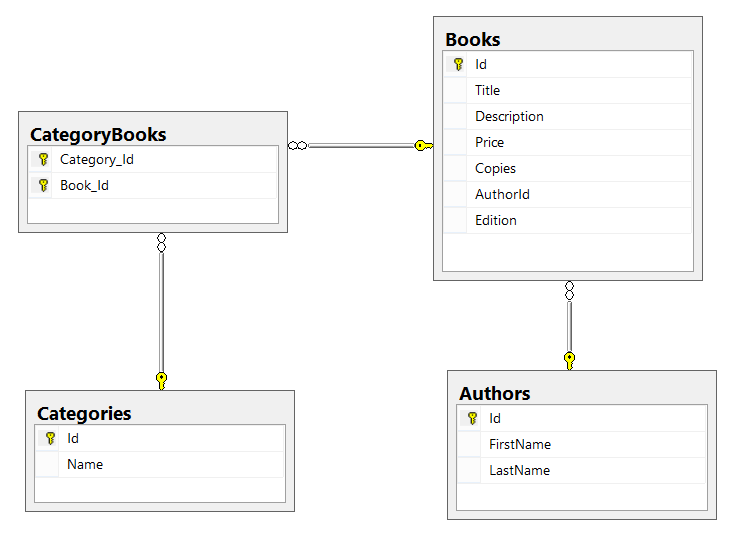
# Homework: ASP.NET Web API

This document defines the homework assignments from [the "Web Services and Cloud" Course @ Software University](https://softuni.bg/courses/web-services-and-cloud/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Book Shop Service

Create a REST Service with Web API for accessing a **book shop**. The shop should hold **books**, **authors** and **categories**.



Use the [**Book Shop Code First lab**](https://softuni.bg/downloads/svn/db-apps/July-2015/2.%20Entity-Framework-Code-First-Exercise-Book-Shop.zip) from the **Database Applications** course as a basis for your service. Check the **Individual Accounts** authenticationwhen creating your project.

### Endpoints

Define the following endpoints in your service project. Do **NOT** use scaffolding (auto-generating endpoints)!

|  |  |  |
| --- | --- | --- |
| **Http  Method** | **Endpoint** | **Description** |
| GET | /api/authors/{id} | Returns book by **id**. |
| POST | /api/authors | Creates a new author with **first name** and **last name** (mandatory). |
| GET | /api/authors/{id}/books | Gets books from author by id. Returns **all data about the book** + **category names**. |
| GET | /api/books/{id} | Gets data about a book by id. Returns **all data about the book** + **category names** + **author name** and **id**. |
| GET | /api/books?search={word} | Gets **top 10 books** which contain the given substring, sorted by **title** (ascending). Returns only the **title** and **id** of the books. |
| PUT | /api/books/{id} | Edits the book. Receives book **title**, **description**, **price**, **copies**, **edition**, **age restriction**, **release date** and **author id**. |
| DELETE | /api/books/{id} | Deletes the book. |
| POST | /api/books | Adds a new book with **title**, **description**, **price**, **copies**, **edition**, **age restriction**, **release date** and a string with space-separated **category names**. |
| GET | /api/categories | Gets all categories (**id** and **name**). |
| GET | /api/categories/{id} | Gets a category (**id** and **name**). |
| PUT | /api/categories/{id} | Edits a category **name**. Make sure no duplicates are created. |
| DELETE | /api/categories/{id} | Deletes a category. |
| POST | /api/categories | Adds a new category by **name**. Make sure no duplicates are created. |

Test the **endpoints** through [**Postman**](https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop).

### Requirements

* Separate your **models**, **data layer** and **services layer** into **separate** **projects**.
* Define separate controllers for **books**, **authors** and **categories**.
* Return proper **status codes** with the data.
* Use **binding models** (validated with attributes) for receiving data and **view models** for returning data.
* Enable **OData** for all **GET actions** where there's a collection returned.

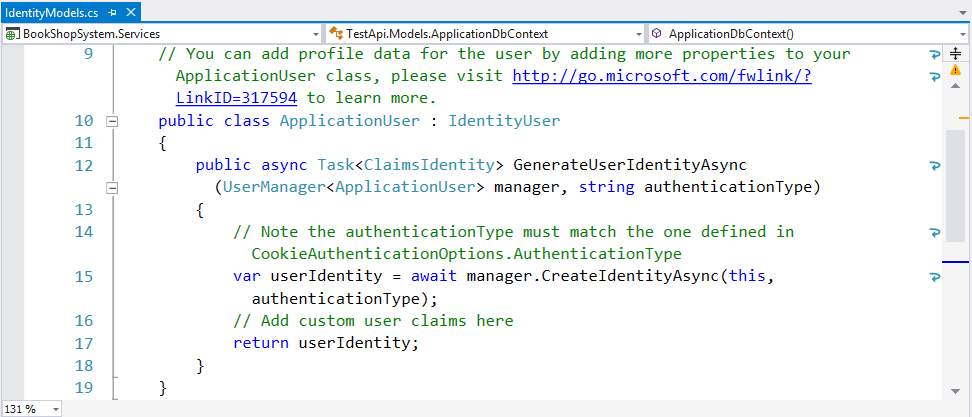
## Application Users

Implement two new models - **user** and **purchase**:

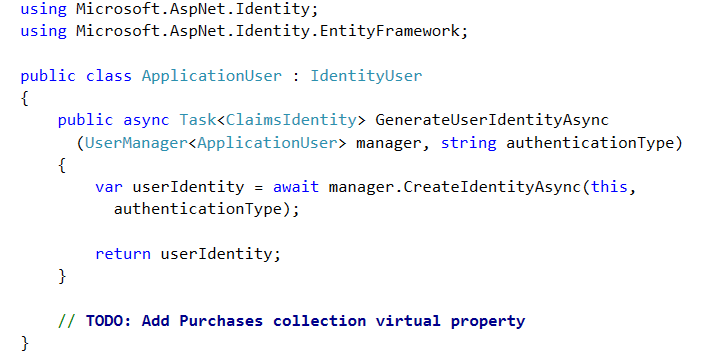
* **ApplicationUser -** has many **purchases**. It should extend (**inherit**) the default built-in **IdentityUser** class and add any extra properties. Users should be able to **login**/**register**/**logout** by using the standart identity API.

To integrate the ASP.NET identity API with your bookshop database:

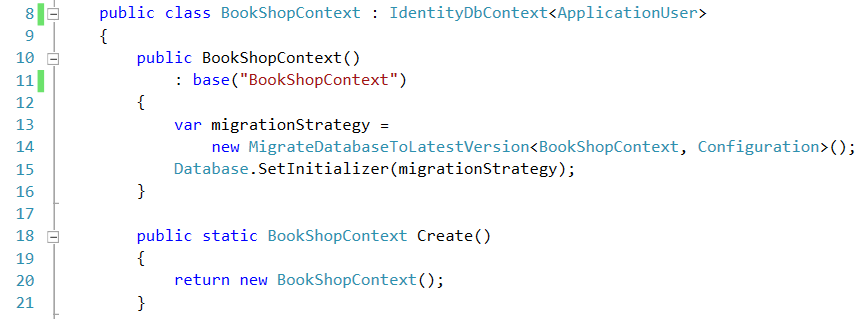
1. Delete the auto-generated **IdentityModels.cs** file



1. Install the **Microsoft.AspNet.Identity.EntityFramework** in your **BookShopSystem.Data** project (where your db context is located)
2. Create a new model **ApplicationUser** and inherit the **IdentityUser** class. That way **ApplicationUser** can have all default identity properties (e.g. **UserName**, **Email**, etc.). It should also hold a **GenerateUserIdentityAsync()** method (used internally by Web API).  
   Last but not least, add all **extra properties** the users should have.

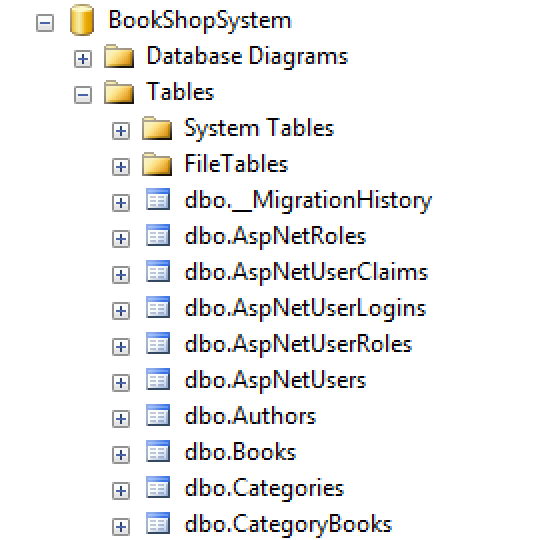


1. Now make changes to the **BookShopContext** class. It should inherit the **IdentityDbContext<T>** class, where **T** is the **ApplicationUser**. That way EF will automatically create user tables (with all their columns) in the database for us.  
   Finally, add a static **Create()** method which returns a new **BookShopContext** instance - Web API needs this.



We don't need to add **IDbSet<ApplicationUser>** in BookShopContext - **IdentityDbContext** already has one and because we inherited it, BookShopContext has it too.

1. Change broken references. In **IdentityConfig.cs** and **Startup.Auth.cs** change **ApplicationDbContext** references to **BookShopContext**. Import the **ApplicationUser** class wherever Web API cannot recognize it.
2. If everything is correct, the next time the service is run, the database should hold the following tables:



* **Purchase -** has **user**, **book, price, date of purchase** and **is-recalled**.

You might need to enable **Code First migrations**.

|  |  |  |
| --- | --- | --- |
| **Http  Method** | **Endpoint** | **Description** |
| PUT | /api/books/buy/{id} | Creates a **purchase** with the specified book and the currently logged user. Decrements the book's copies by 1. If there are no copies, return an appropriate status code. |
| PUT | /api/books/recall/{id} | Returns the book if **less than 30 days** have passed since the purchase. Increments the book count by 1 and sets the purchase to **recalled**. |
| GET | /api/user/{username}/purchases | Gets data about all **purchases** of the specified user, ordered by date of purchase. Returns **username** and for each purchase - **book title**, **purchase price**, **date of purchase** and whether it's **recalled or not**. |

## \* User Roles

Add an **Admin** **role**. Restrict access to all **POST**, **PUT** and **DELETE** routes only to users with **Admin** role (except buying and recalling books).   
Search online for ways to **authorize only certain roles**.

**Tip:** The database context should contain a **DbSet<Roles>**.

|  |  |  |
| --- | --- | --- |
| **Http  Method** | **Endpoint** | **Description** |
| PUT | /api/users/{username}/roles | Adds a new role to the user (passed in request body). |
| DELETE | /api/users/{username}/roles | Removes a role from the user (passed in request body). |

**Note:** Restrict the access to the above routes to users with **Admin role** only.

## \*\* Repository Pattern and Unit of Work

Implement the **repository pattern** and **unit of work** concepts.

Create an abstract unit of work **IBookShopData** with repositories for **books**, **authors** and **categories**.Each controller should receive the **unit of work** through **dependency injection** in the constructor. You might find the following article helpful:

* <http://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application>