Part 8: File Upload for Product Images

Objective: Implement functionality to upload images for products, store them on the server, and link them via URL to product records.

Step 1: Create a Folder for Images

1. Create Directory: In your project's wwwroot folder, create an images directory manually for development. For production, we'll ensure this is created if it doesn't exist via code.  
   * In Visual Studio:
     + Right-click wwwroot
     + Add -> New Folder -> Name it images

Step 2: Update the Product Entity

Modify Product Model:

* Add ImageUrl to Models/Product.cs:

namespace AtirAPI.Models

{

public class Product

{

// ... existing properties ...

public string? ImageUrl { get; set; }

}

}

Database Migration:

* Run these commands in your terminal or command prompt:

dotnet ef migrations add AddImageUrlToProduct

dotnet ef database update

Step 3: Define or Update DTOs

Update DTOs:

* Add ImageUrl to DTOs/ProductDTO.cs:

public class ProductDTO

{

// ... existing properties ...

public string? ImageUrl { get; set; }

}

* Optionally, add to DTOs/ProductCreateDTO.cs if you want to accept image URLs at creation:

public class ProductCreateDTO

{

// ... existing properties ...

public string? ImageUrl { get; set; }

}

Step 4: Update AutoMapper Configuration

AutoMapper Profile:

* Ensure ImageUrl is mapped in Profiles/ProductProfile.cs:

public class ProductProfile : Profile

{

public ProductProfile()

{

CreateMap<ProductCreateDTO, Product>()

.ForMember(dest => dest.CategoryId, opt => opt.MapFrom(src => src.CategoryId))

.ForMember(dest => dest.Category, opt => opt.Ignore())

.ForMember(dest => dest.ImageUrl, opt => opt.MapFrom(src => src.ImageUrl));

CreateMap<Product, ProductDTO>();

CreateMap<Category, CategoryDTO>();

}

}

Step 5: Add File Upload Logic in ProductsController

Modify ProductsController:

* Add an endpoint for uploading images:

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Hosting;

using System.IO;

using AtirAPI.DTOs;

using Microsoft.AspNetCore.Mvc;

[HttpPost("{id}/upload-image")]

public async Task<IActionResult> UploadImage(int id, IFormFile file, [FromServices] IWebHostEnvironment hostEnvironment)

{

var product = await \_context.Products.FindAsync(id);

if (product == null)

{

return NotFound("Product not found.");

}

if (file == null || file.Length == 0)

{

return BadRequest("Invalid file.");

}

// Ensure the images directory exists

var imagePath = Path.Combine(hostEnvironment.WebRootPath, "images");

if (!Directory.Exists(imagePath))

{

Directory.CreateDirectory(imagePath);

}

// Validate file type

var allowedExtensions = new[] { ".jpg", ".jpeg", ".png", ".gif" };

var fileExtension = Path.GetExtension(file.FileName).ToLower();

if (!allowedExtensions.Contains(fileExtension))

{

return BadRequest("Invalid file type. Only JPG, JPEG, PNG, and GIF are allowed.");

}

// Restrict file size (example: 5MB)

const long maxFileSize = 5 \* 1024 \* 1024; // 5MB

if (file.Length > maxFileSize)

{

return BadRequest("File size exceeds the maximum limit of 5MB.");

}

// Generate a unique file name and save the file

var fileName = $"{Guid.NewGuid()}{fileExtension}";

var filePath = Path.Combine(imagePath, fileName);

using (var stream = new FileStream(filePath, FileMode.Create))

{

await file.CopyToAsync(stream);

}

// Cleanup old image if exists

if (!string.IsNullOrEmpty(product.ImageUrl))

{

var oldImagePath = Path.Combine(hostEnvironment.WebRootPath, product.ImageUrl.TrimStart('/'));

if (System.IO.File.Exists(oldImagePath))

{

System.IO.File.Delete(oldImagePath);

}

}

// Update the product's ImageUrl

product.ImageUrl = $"/images/{fileName}";

\_context.Products.Update(product);

await \_context.SaveChangesAsync();

return Ok(\_mapper.Map<ProductDTO>(product));

}

Step 6: Configure Static File Serving

In Program.cs, ensure static files are served:

app.UseStaticFiles();

Step 7: Test the File Upload Endpoint

Using Postman or Swagger:

* Endpoint: POST /api/Products/{id}/upload-image
* Body (select form-data):
  + Key: file
  + Type: File
  + Value: Select an image file
* Expected Response:

{

"id": <productId>,

"name": "...",

"description": "...",

"price": ...,

"stock": ...,

"categoryId": ...,

"category": {...},

"imageUrl": "/images/unique-file-name.jpg"

}

Step 8: Verify Image Serving

* Open the URL returned in imageUrl in your browser to check if the image is served correctly.

Step 9: Enhancements (Optional)

* Validation: You've already implemented file type and size validation.
* Cleanup: Included in the method for replacing images.
* Future Considerations: Consider cloud storage for scalability or image optimization for performance.