2nd-Semester-Project

Generated by Doxygen 1.11.0

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 HttpWebshopCookie Namespace Reference

Namespaces

- namespace Services

5.2 HttpWebshopCookie.Services Namespace Reference

Classes

- class BasketService

  Service class for managing the shopping basket.
- class EmailService

  Service class for sending emails.
- class IdentityEmailSender

  Represents a class that sends emails using the identity service.
- class OrderService

  Service class for managing orders.
- class PostalCodeEntry
- class PostalCodeService
- class ProductService

  Service class for managing products.
- class SmtpSettings

  Configuration settings for SMTP.
- class TagService

  Service class for managing tags.

Chapter 6

Class Documentation

6.1 HttpWebshopCookie.Services.BasketService Class Reference

Service class for managing the shopping basket.

Public Member Functions

- BasketService (IHttpContextAccessor httpContextAccessor, ApplicationDbContext context, OrderService orderCreator)

  Initializes a new instance of the BasketService class.
- Basket GetOrCreateBasket ()

  Gets or creates the shopping basket for the current HTTP context.
- Basket GetOrCreateBasket (IHttpContextAccessor httpContextAccessor, ApplicationDbContext dbContext)

  Gets or creates the shopping basket for the specified HTTP context and database context.
- async Task AddToBasket (string productId)

  Adds a product to the shopping basket.
- async Task< bool > IsInBasket (string productId)

  Checks if a product is in the shopping basket.
- async Task< int?> GetQuantityInBasket (string productId)

  Gets the quantity of a product in the shopping basket.
- async Task< Dictionary< string, int > > GetAllQuantitiesInBasket ()

  Gets the quantities of all products in the shopping basket.
- async Task UpdateBasketItemQuantity (string productId, int newQuantity)

  Updates the quantity of a product in the shopping basket.
- async Task RemoveFromBasket (string productId)

  Removes a product from the shopping basket.
- async Task ClearBasket ()

  Clears the shopping basket.
- Order PlaceOrder (UserWrapper userWrapper)

  Places an order using the shopping basket.

6.1.1 Detailed Description

Service class for managing the shopping basket.

Definition at line 6 of file BasketService.cs.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 BasketService()

```
HttpWebshopCookie.Services.BasketService.BasketService (
            IHttpContextAccessor httpContextAccessor,
            ApplicationDbContext context,
            OrderService orderCreator)
```

Initializes a new instance of the BasketService class.

Parameters

httpContextAccessor	The HTTP context accessor.
context	The application database context.
orderCreator	The order service for creating orders.

Definition at line 18 of file BasketService.cs.

6.1.3 Member Function Documentation

6.1.3.1 AddToBasket()

```
async Task HttpWebshopCookie.Services.BasketService.AddToBasket (
            string productId)
```

Adds a product to the shopping basket.

Parameters

productId	The ID of the product to add.

Returns

A task representing the asynchronous operation.

Exceptions

InvalidOperationException	Thrown when the specified product is not found.

Definition at line 103 of file BasketService.cs.

6.1.3.2 ClearBasket()

```
async Task HttpWebshopCookie.Services.BasketService.ClearBasket ()
```

Clears the shopping basket.

Returns

A task representing the asynchronous operation.

Definition at line 238 of file BasketService.cs.

6.1.3.3 GetAllQuantitiesInBasket()

```
async Task< Dictionary< string, int > > HttpWebshopCookie.Services.BasketService.GetAll↩
QuantitiesInBasket ()
```

Gets the quantities of all products in the shopping basket.

Returns

A task representing the asynchronous operation. The task result contains a dictionary with the product IDs as keys and the quantities as values.

Definition at line 161 of file BasketService.cs.

6.1.3.4 GetOrCreateBasket() [1/2]

```
Basket HttpWebshopCookie.Services.BasketService.GetOrCreateBasket ()
```

Gets or creates the shopping basket for the current HTTP context.

Returns

The shopping basket.

Definition at line 29 of file BasketService.cs.

6.1.3.5 GetOrCreateBasket() [2/2]

```
Basket HttpWebshopCookie.Services.BasketService.GetOrCreateBasket (
            IHttpContextAccessor httpContextAccessor,
            ApplicationDbContext dbContext)
```

Gets or creates the shopping basket for the specified HTTP context and database context.

Parameters

httpContextAccessor	The HTTP context accessor.
dbContext	The application database context.

Returns

The shopping basket.

Definition at line 40 of file BasketService.cs.

6.1.3.6 GetQuantityInBasket()

```
async Task< int?> HttpWebshopCookie.Services.BasketService.GetQuantityInBasket (
            string productId)
```

Gets the quantity of a product in the shopping basket.

Parameters

productId	The ID of the product to get the quantity for.

Returns

A task representing the asynchronous operation. The task result contains the quantity of the product in the basket.

Definition at line 144 of file BasketService.cs.

6.1.3.7 IsInBasket()

```
async Task< bool > HttpWebshopCookie.Services.BasketService.IsInBasket (
            string productId)
```

Checks if a product is in the shopping basket.

Parameters

productId	The ID of the product to check.

Returns

A task representing the asynchronous operation. The task result contains a boolean indicating if the product is in the basket.

Definition at line 133 of file BasketService.cs.

6.1.3.8 PlaceOrder()

```
Order HttpWebshopCookie.Services.BasketService.PlaceOrder (
            UserWrapper userWrapper)
```

Places an order using the shopping basket.

Parameters

userWrapper	The user wrapper for the order.

Returns

The order placed.

Definition at line 279 of file BasketService.cs.

6.1.3.9 RemoveFromBasket()

```
async Task HttpWebshopCookie.Services.BasketService.RemoveFromBasket (
            string productId)
```

Removes a product from the shopping basket.

Parameters

productId	The ID of the product to remove.

Returns

A task representing the asynchronous operation.

Definition at line 211 of file BasketService.cs.

6.1.3.10 UpdateBasketItemQuantity()

```
async Task HttpWebshopCookie.Services.BasketService.UpdateBasketItemQuantity (
            string productId,
            int newQuantity)
```

Updates the quantity of a product in the shopping basket.

Parameters

productId	The ID of the product to update.
newQuantity	The new quantity of the product.

Returns

A task representing the asynchronous operation.

Exceptions

InvalidOperationException	Thrown when the specified product is not found in the basket.

Definition at line 184 of file BasketService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/BasketService.cs

6.2 HttpWebshopCookie.Services.EmailService Class Reference

Service class for sending emails.

Inheritance diagram for HttpWebshopCookie.Services.EmailService:

6.3 HttpWebshopCookie.Services.IdentityEmailSender Class Reference

Represents a class that sends emails using the identity service.

Inheritance diagram for HttpWebshopCookie.Services.IdentityEmailSender:

Collaboration diagram for HttpWebshopCookie.Services.IdentityEmailSender:

Public Member Functions

- IdentityEmailSender (IEmailService emailService)
  Initializes a new instance of the IdentityEmailSender class.
- async Task SendEmailAsync (string email, string subject, string htmlMessage)
  Sends an email asynchronously.

6.3.1 Detailed Description

Represents a class that sends emails using the identity service.

Definition at line 6 of file IdentityEmailSender.cs.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 IdentityEmailSender()

```
HttpWebshopCookie.Services.IdentityEmailSender.IdentityEmailSender (
            IEmailService emailService)
```

Initializes a new instance of the IdentityEmailSender class.

Parameters

emailService	The email service to use for sending emails.

Exceptions

ArgumentNullException	Thrown when the *emailService* is null.

Definition at line 15 of file IdentityEmailSender.cs.

6.3.3 Member Function Documentation

6.3.3.1 SendEmailAsync()

```
async Task HttpWebshopCookie.Services.IdentityEmailSender.SendEmailAsync (
            string email,
            string subject,
            string htmlMessage)
```

Sends an email asynchronously.

Parameters

email	The recipient email address.
subject	The email subject.
htmlMessage	The HTML content of the email.

Returns

A task representing the asynchronous operation.

Exceptions

ArgumentNullException	Thrown when the *email* , *subject* , or *htmlMessage* is null or empty.
InvalidOperationException	Thrown when there is an error sending the email.

Definition at line 29 of file IdentityEmailSender.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/IdentityEmailSender.cs

6.4 HttpWebshopCookie.Services.OrderService Class Reference

Service class for managing orders.

Public Member Functions

- OrderService (ApplicationDbContext context)

  Initializes a new instance of the OrderService class.
- Order GetOrder (string orderId)

  Retrieves an order by its ID.
- OrderItemView[ ] GetOrderItems (Order order)

  Retrieves the order items for a given order.
- string GetTotalPriceString (Order order)

  Retrieves the total price of an order as a formatted string.
- void UpdateOrderStatus (string orderId, OrderStatus newStatus)

  Updates the status of an order.
- void UpdateOrder (Order order)

  Updates an order.
- void DeleteOrder (string orderId)

  Deletes an order.
- List< UserWrapper > GetOrderInvolved (Order order)

  Retrieves the users involved in an order.
- Order CreateOrderFromBasket (Basket basket, UserWrapper userWrapper)

  Creates an order from a basket.

6.4.1 Detailed Description

Service class for managing orders.

Definition at line 6 of file OrderService.cs.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 OrderService()

```
HttpWebshopCookie.Services.OrderService.OrderService (
            ApplicationDbContext context)
```

Initializes a new instance of the OrderService class.

Parameters

context	The application database context.

Definition at line 14 of file OrderService.cs.

6.4.3 Member Function Documentation

6.4.3.1 CreateOrderFromBasket()

```
Order HttpWebshopCookie.Services.OrderService.CreateOrderFromBasket (
            Basket basket,
            UserWrapper userWrapper)
```

Creates an order from a basket.

Parameters

basket	The basket from which to create the order.
userWrapper	The user wrapper associated with the order.

Returns

The created order.

Exceptions

ArgumentNullException	Thrown when the basket or userWrapper is null.
ArgumentException	Thrown when an invalid user type is provided.

Definition at line 155 of file OrderService.cs.

6.4.3.2 DeleteOrder()

```
void HttpWebshopCookie.Services.OrderService.DeleteOrder (
            string orderId)
```

Deletes an order.

Parameters

order↩ Id	The ID of the order to delete.

Exceptions

InvalidOperationException	Thrown when the order is not found.

Definition at line 102 of file OrderService.cs.

6.4.3.3 GetOrder()

```
Order HttpWebshopCookie.Services.OrderService.GetOrder (
            string orderId)
```

Retrieves an order by its ID.

Parameters

order↩ Id	The ID of the order to retrieve.

Returns

The order with the specified ID.

Exceptions

InvalidOperationException	Thrown when the order is not found.

Definition at line 25 of file OrderService.cs.

6.4.3.4 GetOrderInvolved()

```
List< UserWrapper > HttpWebshopCookie.Services.OrderService.GetOrderInvolved (
            Order order)
```

Retrieves the users involved in an order.

Parameters

order	The order for which to retrieve the involved users.

Returns

A list of user wrappers representing the involved users.

Exceptions

ArgumentNullException	Thrown when the order is null.

Definition at line 120 of file OrderService.cs.

6.4.3.5 GetOrderItems()

```
OrderItemView[] HttpWebshopCookie.Services.OrderService.GetOrderItems (
            Order order)
```

Retrieves the order items for a given order.

Parameters

order	The order for which to retrieve the order items.

Returns

An array of order item views.

Exceptions

ArgumentNullException	Thrown when the order is null.

Definition at line 41 of file OrderService.cs.

6.4.3.6 GetTotalPriceString()

```
string HttpWebshopCookie.Services.OrderService.GetTotalPriceString (
            Order order)
```

Retrieves the total price of an order as a formatted string.

Parameters

order	The order for which to retrieve the total price.

Returns

The total price of the order as a formatted string.

Exceptions

ArgumentNullException	Thrown when the order is null.

Definition at line 59 of file OrderService.cs.

6.4.3.7 UpdateOrder()

```
void HttpWebshopCookie.Services.OrderService.UpdateOrder (
            Order order)
```

Updates an order.

Parameters

order	The order to update.

Exceptions

ArgumentNullException	Thrown when the order is null.

Definition at line 89 of file OrderService.cs.

6.4.3.8 UpdateOrderStatus()

```
void HttpWebshopCookie.Services.OrderService.UpdateOrderStatus (
            string orderId,
            OrderStatus newStatus)
```

Updates the status of an order.

Parameters

orderId	The ID of the order to update.
newStatus	The new status of the order.

Exceptions

InvalidOperationException	Thrown when the order is not found.

Definition at line 72 of file OrderService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/OrderService.cs

6.5 HttpWebshopCookie.Services.PostalCodeEntry Class Reference

Properties

- string? PostalCode `[get, set]`
- string? CityName `[get, set]`

6.5.1 Detailed Description

Definition at line 35 of file PostalCodeService.cs.

6.5.2 Property Documentation

6.5.2.1 CityName

```
string?  HttpWebshopCookie.Services.PostalCodeEntry.CityName  [get], [set]
```

Definition at line 38 of file PostalCodeService.cs.

6.5.2.2 PostalCode

```
string?  HttpWebshopCookie.Services.PostalCodeEntry.PostalCode  [get], [set]
```

Definition at line 37 of file PostalCodeService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/PostalCodeService.cs

6.6 HttpWebshopCookie.Services.PostalCodeService Class Reference

Public Member Functions

- PostalCodeService ()
- string GetCityByPostalCode (string postalCode)

6.6.1 Detailed Description

Definition at line 5 of file PostalCodeService.cs.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 PostalCodeService()

```
HttpWebshopCookie.Services.PostalCodeService.PostalCodeService ()
```

Definition at line 10 of file PostalCodeService.cs.

6.6.3 Member Function Documentation

6.6.3.1 GetCityByPostalCode()

```
string HttpWebshopCookie.Services.PostalCodeService.GetCityByPostalCode (
            string postalCode)
```

Definition at line 25 of file PostalCodeService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/PostalCodeService.cs

6.7 HttpWebshopCookie.Services.ProductService Class Reference

Service class for managing products.

Public Member Functions

- ProductService (ApplicationDbContext context)
  Initializes a new instance of the ProductService class.
- List< Product > GetProducts ()
  Gets the list of all products.
- Product GetProductById (string id)
  Gets a product by its ID.
- void AddProduct (Product product)
  Adds a new product.
- void UpdateProduct (Product product)
  Updates an existing product.
- void DeleteProduct (string id)
  Deletes a product by its ID.

6.7.1 Detailed Description

Service class for managing products.

Definition at line 6 of file ProductService.cs.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 ProductService()

```
HttpWebshopCookie.Services.ProductService.ProductService (
            ApplicationDbContext context)
```

Initializes a new instance of the ProductService class.

Parameters

context	The application database context.

Exceptions

ArgumentNullException	Thrown when the context is null.

Definition at line 15 of file ProductService.cs.

6.7.3 Member Function Documentation

6.7.3.1 AddProduct()

```
void HttpWebshopCookie.Services.ProductService.AddProduct (
            Product product)
```

Adds a new product.

Parameters

product	The product to add.

Exceptions

ArgumentNullException	Thrown when the product is null.
InvalidOperationException	Thrown when unable to add the product.

Definition at line 61 of file ProductService.cs.

6.7.3.2 DeleteProduct()

```
void HttpWebshopCookie.Services.ProductService.DeleteProduct (
            string id)
```

Deletes a product by its ID.

Parameters

id	The ID of the product to delete.

Exceptions

ArgumentNullException	Thrown when the ID is null or empty.
InvalidOperationException	Thrown when unable to delete the product or product not found.

Definition at line 109 of file ProductService.cs.

6.7.3.3 GetProductById()

```
Product HttpWebshopCookie.Services.ProductService.GetProductById (
            string id)
```

Gets a product by its ID.

Parameters

id	The ID of the product.

Returns

The product with the specified ID.

Exceptions

ArgumentNullException	Thrown when the ID is null or empty.
InvalidOperationException	Thrown when the product is not found.

Definition at line 44 of file ProductService.cs.

6.7.3.4 GetProducts()

```
List< Product > HttpWebshopCookie.Services.ProductService.GetProducts ()
```

Gets the list of all products.

Returns

The list of products.

Exceptions

InvalidOperationException	Thrown when unable to retrieve products.

Definition at line 25 of file ProductService.cs.

6.7.3.5 UpdateProduct()

```
void HttpWebshopCookie.Services.ProductService.UpdateProduct (
            Product product)
```

Updates an existing product.

Parameters

product	The product to update.

Exceptions

ArgumentNullException	Thrown when the product is null.
InvalidOperationException	Thrown when unable to update the product.

Definition at line 85 of file ProductService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/ProductService.cs

6.8 HttpWebshopCookie.Services.SmtpSettings Class Reference

Configuration settings for SMTP.

Properties

- string? Server `[get, set]`
  Gets or sets the SMTP server.
- int Port `[get, set]`
  Gets or sets the SMTP port.
- string? SenderName `[get, set]`
  Gets or sets the sender's name.
- string? SenderEmail `[get, set]`
  Gets or sets the sender's email address.
- string? Username `[get, set]`
  Gets or sets the username for SMTP authentication.
- string? Password `[get, set]`
  Gets or sets the password for SMTP authentication.

6.8.1 Detailed Description

Configuration settings for SMTP.

Definition at line 113 of file EmailService.cs.

6.8.2 Property Documentation

6.8.2.1 Password

```
string? HttpWebshopCookie.Services.SmtpSettings.Password [get], [set]
```

Gets or sets the password for SMTP authentication.

Definition at line 143 of file EmailService.cs.

6.8.2.2 Port

```
int HttpWebshopCookie.Services.SmtpSettings.Port [get], [set]
```

Gets or sets the SMTP port.

Definition at line 123 of file EmailService.cs.

6.8.2.3 SenderEmail

```
string? HttpWebshopCookie.Services.SmtpSettings.SenderEmail [get], [set]
```

Gets or sets the sender's email address.

Definition at line 133 of file EmailService.cs.

6.8.2.4 SenderName

```
string?  HttpWebshopCookie.Services.SmtpSettings.SenderName  [get], [set]
```

Gets or sets the sender's name.

Definition at line 128 of file EmailService.cs.

6.8.2.5 Server

```
string?  HttpWebshopCookie.Services.SmtpSettings.Server  [get], [set]
```

Gets or sets the SMTP server.

Definition at line 118 of file EmailService.cs.

6.8.2.6 Username

```
string?  HttpWebshopCookie.Services.SmtpSettings.Username  [get], [set]
```

Gets or sets the username for SMTP authentication.

Definition at line 138 of file EmailService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/EmailService.cs

6.9 HttpWebshopCookie.Services.TagService Class Reference

Service class for managing tags.

Public Member Functions

- TagService (ApplicationDbContext context)
  Initializes a new instance of the TagService class.
- async Task< List< Tag > > GetTagsAsync ()
  Retrieves all tags asynchronously.
- async Task< Tag > GetTagByIdAsync (string id)
  Retrieves a tag by its ID asynchronously.
- async Task< Tag > CreateTagAsync (Tag tag)
  Creates a new tag asynchronously.
- async Task< Tag > UpdateTagAsync (Tag tag)
  Updates an existing tag asynchronously.
- async Task< Tag > DeleteTagAsync (string id)
  Deletes a tag by its ID asynchronously.
- async Task< List< string?> > GetOccasionsAsync ()
  Retrieves all unique occasions asynchronously.
- async Task< List< Tag > > GetTagsOrderedByOccasionAsync ()
  Retrieves all tags ordered by occasion, category, and subcategory asynchronously.
- async Task<(List< Tag >, int)> GetTagsOrderedByOccasionAsync (int pageNumber, int pageSize)
  Retrieves a paged list of tags ordered by occasion, category, and subcategory asynchronously.

6.9.1 Detailed Description

Service class for managing tags.

Definition at line 6 of file TagService.cs.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 TagService()

```
HttpWebshopCookie.Services.TagService.TagService (
            ApplicationDbContext context)
```

Initializes a new instance of the TagService class.

Parameters

context	The application database context.

Exceptions

ArgumentNullException	Thrown when the context is null.

Definition at line 15 of file TagService.cs.

6.9.3 Member Function Documentation

6.9.3.1 CreateTagAsync()

```
async Task< Tag > HttpWebshopCookie.Services.TagService.CreateTagAsync (
            Tag tag)
```

Creates a new tag asynchronously.

Parameters

tag	The tag to create.

Returns

A task representing the asynchronous operation. The task result contains the created tag.

Exceptions

ArgumentNullException	Thrown when the tag is null.
InvalidOperationException	Thrown when unable to create the tag.

Definition at line 62 of file TagService.cs.

6.9.3.2 DeleteTagAsync()

```
async Task< Tag > HttpWebshopCookie.Services.TagService.DeleteTagAsync (
            string id)
```

Deletes a tag by its ID asynchronously.

Parameters

id	The ID of the tag to delete.

Returns

A task representing the asynchronous operation. The task result contains the deleted tag.

Exceptions

ArgumentNullException	Thrown when the ID is null or empty.
KeyNotFoundException	Thrown when the tag with the specified ID is not found.
InvalidOperationException	Thrown when unable to delete the tag.

Definition at line 115 of file TagService.cs.

6.9.3.3 GetOccasionsAsync()

```
async Task< List< string?> > HttpWebshopCookie.Services.TagService.GetOccasionsAsync ()
```

Retrieves all unique occasions asynchronously.

Returns

A task representing the asynchronous operation. The task result contains a list of unique occasions.

Exceptions

InvalidOperationException	Thrown when unable to retrieve occasions.

Definition at line 145 of file TagService.cs.

6.9.3.4 GetTagByIdAsync()

```
async Task< Tag > HttpWebshopCookie.Services.TagService.GetTagByIdAsync (
            string id)
```

Retrieves a tag by its ID asynchronously.

Parameters

id	The ID of the tag.

Returns

A task representing the asynchronous operation. The task result contains the tag with the specified ID.

Exceptions

ArgumentNullException	Thrown when the ID is null or empty.
KeyNotFoundException	Thrown when the tag with the specified ID is not found.

Definition at line 44 of file TagService.cs.

6.9.3.5 GetTagsAsync()

```
async Task< List< Tag > > HttpWebshopCookie.Services.TagService.GetTagsAsync ()
```

Retrieves all tags asynchronously.

Returns

A task representing the asynchronous operation. The task result contains a list of tags.

Exceptions

InvalidOperationException	Thrown when unable to retrieve tags.

Definition at line 25 of file TagService.cs.

6.9.3.6 GetTagsOrderedByOccasionAsync() [1/2]

```
async Task< List< Tag > > HttpWebshopCookie.Services.TagService.GetTagsOrderedByOccasionAsync
()
```

Retrieves all tags ordered by occasion, category, and subcategory asynchronously.

Returns

A task representing the asynchronous operation. The task result contains a list of tags ordered by occasion, category, and subcategory.

Exceptions

InvalidOperationException	Thrown when unable to retrieve ordered tags.

Definition at line 166 of file TagService.cs.

6.9.3.7 GetTagsOrderedByOccasionAsync() [2/2]

```
async Task<(List< Tag >, int)> HttpWebshopCookie.Services.TagService.GetTagsOrderedByOccasion↩
Async (
            int pageNumber,
            int pageSize)
```

Retrieves a paged list of tags ordered by occasion, category, and subcategory asynchronously.

Parameters

pageNumber	The page number.
pageSize	The number of items per page.

Returns

A task representing the asynchronous operation. The task result contains a tuple with the paged list of tags and the total number of items.

Exceptions

InvalidOperationException	Thrown when unable to retrieve paged tags.

Definition at line 189 of file TagService.cs.

6.9.3.8 UpdateTagAsync()

```
async Task< Tag > HttpWebshopCookie.Services.TagService.UpdateTagAsync (
            Tag tag)
```

Updates an existing tag asynchronously.

Parameters

tag	The tag to update.

Returns

A task representing the asynchronous operation. The task result contains the updated tag.

Exceptions

ArgumentNullException	Thrown when the tag is null.
InvalidOperationException	Thrown when unable to update the tag.

Definition at line 88 of file TagService.cs.

The documentation for this class was generated from the following file:

- C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/TagService.cs

Chapter 7

File Documentation

7.1 C:/Users/Cal-l/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/BasketService.cs File Reference

Classes

- class HttpWebshopCookie.Services.BasketService

  Service class for managing the shopping basket.

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.2 BasketService.cs

Go to the documentation of this file.

```
00001 namespace HttpWebshopCookie.Services;
00002
00006 public class BasketService
00007 {
00008     private readonly IHttpContextAccessor _httpContextAccessor;
00009     private readonly ApplicationDbContext _context;
00010     private readonly OrderService orderCreator;
00011
00018     public BasketService(IHttpContextAccessor httpContextAccessor, ApplicationDbContext context,
      OrderService orderCreator)
00019     {
00020         this.orderCreator = orderCreator;
00021         _httpContextAccessor = httpContextAccessor ?? throw new
      ArgumentNullException(nameof(httpContextAccessor));
00022         _context = context ?? throw new ArgumentNullException(nameof(context));
00023     }
00024
00029     public Basket GetOrCreateBasket()
00030     {
00031         return GetOrCreateBasket(_httpContextAccessor, _context);
00032     }
00033
00040     public Basket GetOrCreateBasket(IHttpContextAccessor httpContextAccessor, ApplicationDbContext
      dbContext)
00041     {
00042         return GetOrCreateBasketInternal(httpContextAccessor, dbContext);
00043     }
00044
```

```
00051     private Basket GetOrCreateBasketInternal(IHttpContextAccessor httpContextAccessor,
      ApplicationDbContext dbContext)
00052     {
00053         string? basketId = httpContextAccessor.HttpContext?.Request.Cookies["BasketId"];
00054         Basket? basket;
00055 
00056         if (string.IsNullOrEmpty(basketId))
00057         {
00058             basket = new Basket();
00059             dbContext.Baskets.Add(basket);
00060             dbContext.SaveChanges();
00061             StoreBasketIdInCookie(basket.Id, httpContextAccessor);
00062         }
00063         else
00064         {
00065             basket = dbContext.Baskets.Include(b => b.Items)
00066                                       .ThenInclude(i => i.ProductInBasket)
00067                                       .FirstOrDefault(b => b.Id == basketId);
00068             if (basket == null)
00069             {
00070                 basket = new Basket();
00071                 dbContext.Baskets.Add(basket);
00072                 dbContext.SaveChanges();
00073                 StoreBasketIdInCookie(basket.Id, httpContextAccessor);
00074             }
00075         }
00076 
00077         return basket;
00078     }
00079 
00085     private void StoreBasketIdInCookie(string basketId, IHttpContextAccessor httpContextAccessor)
00086     {
00087         var options = new CookieOptions
00088         {
00089             HttpOnly = true,
00090             Secure = true,
00091             SameSite = SameSiteMode.Strict,
00092             Expires = DateTime.UtcNow.AddDays(1)
00093         };
00094         httpContextAccessor.HttpContext?.Response.Cookies.Append("BasketId", basketId, options);
00095     }
00096 
00103     public async Task AddToBasket(string productId)
00104     {
00105         var basket = GetOrCreateBasket();
00106         var item = basket.Items.FirstOrDefault(i => i.ProductId == productId);
00107 
00108         if (item != null)
00109         {
00110             item.Quantity++;
00111         }
00112         else
00113         {
00114             Product? product = await _context.Products.FindAsync(productId) ?? throw new
      InvalidOperationException("Product not found.");
00115             item = new BasketItem
00116             {
00117                 ProductId = productId,
00118                 Quantity = 1,
00119                 ProductInBasket = product
00120             };
00121             basket.Items.Add(item);
00122         }
00123 
00124         await _context.SaveChangesAsync();
00125         await LogBasketActivity(basket.Id, productId, "Add", item?.Quantity);
00126     }
00127 
00133     public async Task<bool> IsInBasket(string productId)
00134     {
00135         var basket = GetOrCreateBasket();
00136         return await Task.FromResult(basket.Items.Any(i => i.ProductId == productId));
00137     }
00138 
00144     public async Task<int?> GetQuantityInBasket(string productId)
00145     {
00146         var basket = GetOrCreateBasket();
00147         var item = basket.Items.FirstOrDefault(i => i.ProductId == productId);
00148 
00149         if (item == null)
00150         {
00151             return 0;
00152         }
00153 
00154         return await Task.FromResult(item.Quantity);
00155     }
00156 
```

```
00161     public async Task<Dictionary<string, int» GetAllQuantitiesInBasket()
00162     {
00163         var basket = GetOrCreateBasket();
00164 
00165         if (basket.Id == null)
00166         {
00167             return new Dictionary<string, int>();
00168         }
00169 
00170         return await _context.BasketItems
00171             .Where(item => item.BasketId == basket.Id && item.ProductId != null)
00172             .Select(item => new { item.ProductId, item.Quantity })
00173             .Where(item => item.ProductId != null)
00174             .ToDictionaryAsync(item => item.ProductId!, item => item.Quantity ?? 0);
00175     }
00176 
00184     public async Task UpdateBasketItemQuantity(string productId, int newQuantity)
00185     {
00186         var basket = GetOrCreateBasket();
00187         var item = basket.Items.FirstOrDefault(i => i.ProductId == productId);
00188         if (item == null)
00189         {
00190             throw new InvalidOperationException("Product not in basket.");
00191         }
00192 
00193         if (newQuantity <= 0)
00194         {
00195             basket.Items.Remove(item);
00196         }
00197         else
00198         {
00199             item.Quantity = newQuantity;
00200         }
00201 
00202         await _context.SaveChangesAsync();
00203         await LogBasketActivity(basket.Id, productId, "Update", newQuantity);
00204     }
00205 
00211     public async Task RemoveFromBasket(string productId)
00212     {
00213         var basket = GetOrCreateBasket();
00214         var item = basket.Items.FirstOrDefault(i => i.ProductId == productId);
00215 
00216         if (item == null)
00217         {
00218             return;
00219         }
00220 
00221         if (item.Quantity > 1)
00222         {
00223             item.Quantity--;
00224         }
00225         else
00226         {
00227             basket.Items.Remove(item);
00228         }
00229 
00230         await _context.SaveChangesAsync();
00231         await LogBasketActivity(basket.Id, productId, "Remove", item?.Quantity);
00232     }
00233 
00238     public async Task ClearBasket()
00239     {
00240         var basket = GetOrCreateBasket();
00241         basket.Items.Clear();
00242         await _context.SaveChangesAsync();
00243         await LogBasketActivity(basket.Id, null, "ClearAll", 0);
00244     }
00245 
00254     private async Task LogBasketActivity(string basketId, string? productId, string activityType, int?
      quantityChanged)
00255     {
00256         var userId = _httpContextAccessor.HttpContext?.User.FindFirstValue(ClaimTypes.NameIdentifier);
00257         var sessionId = _httpContextAccessor.HttpContext?.Session.Id;
00258 
00259         var activity = new BasketActivity
00260         {
00261             BasketId = basketId,
00262             ProductId = productId,
00263             ActivityType = activityType,
00264             QuantityChanged = quantityChanged ?? 0,
00265             SessionId = sessionId,
00266             UserId = string.IsNullOrEmpty(userId) ? null : userId,
00267             IsRegisteredUser = !string.IsNullOrEmpty(userId)
00268         };
00269 
00270         _context.BasketActivities.Add(activity);
```

```
00271         await _context.SaveChangesAsync();
00272     }
00273
00279     public Order PlaceOrder(UserWrapper userWrapper)
00280     {
00281         var basket = GetOrCreateBasket();
00282         var order = orderCreator.CreateOrderFromBasket(basket, userWrapper);
00283
00284         basket.Items.Clear();
00285         LogBasketActivity(basket.Id, null, "Checkout", 0).Wait();
00286         _context.SaveChanges();
00287
00288         return order;
00289     }
00290 }
```

7.3 C:/Users/Cal-I/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/EmailService.cs File Reference

Classes

- class HttpWebshopCookie.Services.EmailService

  Service class for sending emails.
- class HttpWebshopCookie.Services.SmtpSettings

  Configuration settings for SMTP.

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.4 EmailService.cs

Go to the documentation of this file.

```
00001 namespace HttpWebshopCookie.Services;
00002
00006 public class EmailService : IEmailService
00007 {
00008     private readonly SmtpSettings _smtpSettings;
00009
00014     public EmailService(IOptions<SmtpSettings> smtpSettings)
00015     {
00016         _smtpSettings = smtpSettings.Value;
00017     }
00018
00029     public async Task SendEmailAsync(string toEmail, string subject, string message)
00030     {
00031         if (string.IsNullOrEmpty(toEmail))
00032         {
00033             throw new ArgumentNullException(nameof(toEmail), "Recipient email address cannot be null
      or empty.");
00034         }
00035
00036         if (string.IsNullOrEmpty(subject))
00037         {
00038             throw new ArgumentNullException(nameof(subject), "Email subject cannot be null or
      empty.");
00039         }
00040
00041         if (string.IsNullOrEmpty(message))
00042         {
00043             throw new ArgumentNullException(nameof(message), "Email message cannot be null or
      empty.");
00044         }
00045
00046         var emailMessage = new MimeMessage();
```

```
00047         emailMessage.From.Add(new MailboxAddress(_smtpSettings.SenderName,
      _smtpSettings.SenderEmail));
00048         emailMessage.To.Add(new MailboxAddress("", toEmail));
00049         emailMessage.Subject = subject;
00050         emailMessage.Body = new TextPart(MimeKit.Text.TextFormat.Html) { Text = message };
00051 
00052         using (var client = new SmtpClient())
00053         {
00054             try
00055             {
00056                 await client.ConnectAsync(_smtpSettings.Server, _smtpSettings.Port,
      MailKit.Security.SecureSocketOptions.None);
00057                 await client.SendAsync(emailMessage);
00058                 await client.DisconnectAsync(true);
00059             }
00060             catch (SmtpCommandException ex)
00061             {
00062                 // Handle command errors (e.g., invalid recipient address)
00063                 throw new InvalidOperationException($"SMTP command error: {ex.Message}", ex);
00064             }
00065             catch (SmtpProtocolException ex)
00066             {
00067                 // Handle protocol errors (e.g., unexpected server response)
00068                 throw new InvalidOperationException($"SMTP protocol error: {ex.Message}", ex);
00069             }
00070         }
00071     }
00072 
00081     public async Task SendMimeMessageAsync(MimeMessage message)
00082     {
00083         if (message == null)
00084         {
00085             throw new ArgumentNullException(nameof(message), "MIME message cannot be null.");
00086         }
00087 
00088         using (var client = new SmtpClient())
00089         {
00090             try
00091             {
00092                 await client.ConnectAsync(_smtpSettings.Server, _smtpSettings.Port,
      MailKit.Security.SecureSocketOptions.None);
00093                 await client.SendAsync(message);
00094                 await client.DisconnectAsync(true);
00095             }
00096             catch (SmtpCommandException ex)
00097             {
00098                 // Handle command errors (e.g., invalid recipient address)
00099                 throw new InvalidOperationException($"SMTP command error: {ex.Message}", ex);
00100             }
00101             catch (SmtpProtocolException ex)
00102             {
00103                 // Handle protocol errors (e.g., unexpected server response)
00104                 throw new InvalidOperationException($"SMTP protocol error: {ex.Message}", ex);
00105             }
00106         }
00107     }
00108 }
00109 
00113 public class SmtpSettings
00114 {
00118     public string? Server { get; set; }
00119 
00123     public int Port { get; set; }
00124 
00128     public string? SenderName { get; set; }
00129 
00133     public string? SenderEmail { get; set; }
00134 
00138     public string? Username { get; set; }
00139 
00143     public string? Password { get; set; }
00144 }
```

7.5 C:/Users/Cal-I/Documents/GitHub/HttpWebshopCookie/Http↩ WebshopCookie/Services/IdentityEmailSender.cs File Reference

Classes

- class HttpWebshopCookie.Services.IdentityEmailSender

  Represents a class that sends emails using the identity service.

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.6 IdentityEmailSender.cs

Go to the documentation of this file.

```
00001 namespace HttpWebshopCookie.Services
00002 {
00006     public class IdentityEmailSender : IEmailSender
00007     {
00008         private readonly IEmailService _emailService;
00009 
00015         public IdentityEmailSender(IEmailService emailService)
00016         {
00017             _emailService = emailService ?? throw new ArgumentNullException(nameof(emailService),
      "Email service cannot be null.");
00018         }
00019 
00029         public async Task SendEmailAsync(string email, string subject, string htmlMessage)
00030         {
00031             if (string.IsNullOrEmpty(email))
00032             {
00033                 throw new ArgumentNullException(nameof(email), "Recipient email address cannot be null
      or empty.");
00034             }
00035 
00036             if (string.IsNullOrEmpty(subject))
00037             {
00038                 throw new ArgumentNullException(nameof(subject), "Email subject cannot be null or
      empty.");
00039             }
00040 
00041             if (string.IsNullOrEmpty(htmlMessage))
00042             {
00043                 throw new ArgumentNullException(nameof(htmlMessage), "Email message cannot be null or
      empty.");
00044             }
00045 
00046             try
00047             {
00048                 await _emailService.SendEmailAsync(email, subject, htmlMessage);
00049                 return;
00050             }
00051             catch (Exception ex)
00052             {
00053                 throw new InvalidOperationException($"An error occurred while sending the email:
      {ex.Message}", ex);
00054             }
00055         }
00056     }
00057 }
```

7.7 C:/Users/Cal-I/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/OrderService.cs File Reference

Classes

- class HttpWebshopCookie.Services.OrderService

  Service class for managing orders.

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.8 OrderService.cs

Go to the documentation of this file.

```
00001 namespace HttpWebshopCookie.Services
00002 {
00006     public class OrderService
00007     {
00008         private readonly ApplicationDbContext context;
00009 
00014         public OrderService(ApplicationDbContext context)
00015         {
00016             this.context = context ?? throw new ArgumentNullException(nameof(context), "Database
      context cannot be null.");
00017         }
00018 
00025         public Order GetOrder(string orderId)
00026         {
00027             var order = context.Orders
00028                 .Include(o => o.OrderItems)
00029                 .ThenInclude(oi => oi.ProductItem)
00030                 .SingleOrDefault(o => o.Id == orderId);
00031 
00032             return order ?? throw new InvalidOperationException("Order not found.");
00033         }
00034 
00041         public OrderItemView[] GetOrderItems(Order order)
00042         {
00043             if (order == null) throw new ArgumentNullException(nameof(order), "Order cannot be
      null.");
00044 
00045             return order.OrderItems.Select(oi => new OrderItemView
00046             {
00047                 ProductName = oi.ProductItem?.Name,
00048                 Quantity = oi.Quantity,
00049                 Price = oi.ProductItem?.Price
00050             }).ToArray();
00051         }
00052 
00059         public string GetTotalPriceString(Order order)
00060         {
00061             if (order == null) throw new ArgumentNullException(nameof(order), "Order cannot be
      null.");
00062 
00063             return order.TotalPrice.ToString("C2");
00064         }
00065 
00072         public void UpdateOrderStatus(string orderId, OrderStatus newStatus)
00073         {
00074             var order = context.Orders.Find(orderId);
00075             if (order == null)
00076             {
00077                 throw new InvalidOperationException("Order not found.");
00078             }
00079 
00080             order.Status = newStatus;
00081             context.SaveChanges();
00082         }
00083 
00089         public void UpdateOrder(Order order)
00090         {
00091             if (order == null) throw new ArgumentNullException(nameof(order), "Order cannot be
      null.");
00092 
00093             context.Orders.Update(order);
00094             context.SaveChanges();
00095         }
00096 
00102         public void DeleteOrder(string orderId)
00103         {
00104             var order = context.Orders.Find(orderId);
00105             if (order == null)
00106             {
00107                 throw new InvalidOperationException("Order not found.");
00108             }
00109 
00110             context.Orders.Remove(order);
00111             context.SaveChanges();
00112         }
00113 
00120         public List<UserWrapper> GetOrderInvolved(Order order)
00121         {
00122             if (order == null) throw new ArgumentNullException(nameof(order), "Order cannot be
      null.");
00123 
00124             List<UserWrapper> involvedEntities = new();
```

```
00125
00126             if (order.Guest != null)
00127             {
00128                 var guestWrapper = new UserWrapper(order.Guest);
00129                 involvedEntities.Add(guestWrapper);
00130             }
00131
00132             if (order.Customer != null)
00133             {
00134                 var customerWrapper = new UserWrapper(order.Customer);
00135                 involvedEntities.Add(customerWrapper);
00136             }
00137
00138             if (order.Employee != null)
00139             {
00140                 var employeeWrapper = new UserWrapper(order.Employee);
00141                 involvedEntities.Add(employeeWrapper);
00142             }
00143
00144             return involvedEntities;
00145         }
00146
00155         public Order CreateOrderFromBasket(Basket basket, UserWrapper userWrapper)
00156         {
00157             if (basket == null) throw new ArgumentNullException(nameof(basket), "Basket cannot be
      null.");
00158             if (userWrapper == null) throw new ArgumentNullException(nameof(userWrapper), "User
      wrapper cannot be null.");
00159
00160             var order = new Order
00161             {
00162                 OrderDate = DateTime.Now,
00163                 Status = OrderStatus.Pending,
00164             };
00165
00166             switch (userWrapper.GetUserType())
00167             {
00168                 case "Guest":
00169                     order.Guest = userWrapper.Guest;
00170                     order.GuestId = userWrapper.Id;
00171                     context.Addresses.Add(userWrapper.Guest!.Address!);
00172                     context.GuestUsers.Add(userWrapper.Guest!);
00173                     break;
00174                 case "Customer":
00175                     order.Customer = userWrapper.Customer;
00176                     order.CustomerId = userWrapper.Id;
00177                     break;
00178                 case "ApplicationUser":
00179                     order.Customer = userWrapper.Customer;
00180                     order.CustomerId = userWrapper.Id;
00181                     break;
00182                 case "Employee":
00183                     order.Employee = userWrapper.Employee;
00184                     order.EmployeeId = userWrapper.Id;
00185                     break;
00186                 default:
00187                     throw new ArgumentException("Valid user type must be provided");
00188             }
00189
00190             foreach (var basketItem in basket.Items)
00191             {
00192                 var orderItem = new OrderItem
00193                 {
00194                     ProductItem = basketItem.ProductInBasket,
00195                     ProductId = basketItem.ProductId!,
00196                     Quantity = basketItem.Quantity ?? 0,
00197                     UnitPrice = basketItem.ProductInBasket?.Price ?? 0
00198                 };
00199
00200                 order.OrderItems.Add(orderItem);
00201             }
00202
00203             context.Orders.Add(order);
00204             context.SaveChanges();
00205             return order;
00206         }
00207     }
00208 }
```

7.9 C:/Users/Cal-I/Documents/GitHub/HttpWebshopCookie/Http↩ WebshopCookie/Services/PostalCodeService.cs File Reference

Classes

- class HttpWebshopCookie.Services.PostalCodeService
- class HttpWebshopCookie.Services.PostalCodeEntry

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.10 PostalCodeService.cs

Go to the documentation of this file.

```
00001 using Newtonsoft.Json;
00002
00003 namespace HttpWebshopCookie.Services;
00004
00005 public class PostalCodeService
00006 {
00007     private readonly string jsonFilePath = Path.Combine("Data", "MockData", "postnummerfil.json");
00008     private readonly Dictionary<string, string> _postalCodeToCity;
00009
00010     public PostalCodeService()
00011     {
00012         string jsonData = File.ReadAllText(jsonFilePath);
00013         var postalCodes = JsonConvert.DeserializeObject<List<PostalCodeEntry»(jsonData) ?? [];
00014
00015         _postalCodeToCity = [];
00016         foreach (var entry in postalCodes)
00017         {
00018             if (entry.PostalCode != null && !_postalCodeToCity.ContainsKey(entry.PostalCode))
00019             {
00020                 _postalCodeToCity.Add(entry.PostalCode, entry.CityName ?? string.Empty);
00021             }
00022         }
00023     }
00024
00025     public string GetCityByPostalCode(string postalCode)
00026     {
00027         if (_postalCodeToCity.TryGetValue(postalCode, out var cityName))
00028         {
00029             return cityName;
00030         }
00031         return string.Empty;
00032     }
00033 }
00034
00035 public class PostalCodeEntry
00036 {
00037     public string? PostalCode { get; set; }
00038     public string? CityName { get; set; }
00039 }
```

7.11 C:/Users/Cal-I/Documents/GitHub/HttpWebshopCookie/Http↩ WebshopCookie/Services/ProductService.cs File Reference

Classes

- class HttpWebshopCookie.Services.ProductService

  Service class for managing products.

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.12 ProductService.cs

Go to the documentation of this file.

```
00001 namespace HttpWebshopCookie.Services
00002 {
00006     public class ProductService
00007     {
00008         private readonly ApplicationDbContext _context;
00009
00015         public ProductService(ApplicationDbContext context)
00016         {
00017             _context = context ?? throw new ArgumentNullException(nameof(context), "Database context
      cannot be null.");
00018         }
00019
00025         public List<Product> GetProducts()
00026         {
00027             try
00028             {
00029                 return _context.Products.ToList();
00030             }
00031             catch (Exception ex)
00032             {
00033                 throw new InvalidOperationException("Unable to retrieve products.", ex);
00034             }
00035         }
00036
00044         public Product GetProductById(string id)
00045         {
00046             if (string.IsNullOrEmpty(id))
00047             {
00048                 throw new ArgumentNullException(nameof(id), "Product ID cannot be null or empty.");
00049             }
00050
00051             var product = _context.Products.FirstOrDefault(p => p.Id == id);
00052             return product ?? throw new InvalidOperationException("Product not found.");
00053         }
00054
00061         public void AddProduct(Product product)
00062         {
00063             if (product == null)
00064             {
00065                 throw new ArgumentNullException(nameof(product), "Product cannot be null.");
00066             }
00067
00068             try
00069             {
00070                 _context.Products.Add(product);
00071                 _context.SaveChanges();
00072             }
00073             catch (Exception ex)
00074             {
00075                 throw new InvalidOperationException("Unable to add product.", ex);
00076             }
00077         }
00078
00085         public void UpdateProduct(Product product)
00086         {
00087             if (product == null)
00088             {
00089                 throw new ArgumentNullException(nameof(product), "Product cannot be null.");
00090             }
00091
00092             try
00093             {
00094                 _context.Products.Update(product);
00095                 _context.SaveChanges();
00096             }
00097             catch (Exception ex)
00098             {
00099                 throw new InvalidOperationException("Unable to update product.", ex);
00100             }
00101         }
00102
00109         public void DeleteProduct(string id)
```

```
00110         {
00111             if (string.IsNullOrEmpty(id))
00112             {
00113                 throw new ArgumentNullException(nameof(id), "Product ID cannot be null or empty.");
00114             }
00115
00116             var product = _context.Products.FirstOrDefault(p => p.Id == id);
00117             if (product == null)
00118             {
00119                 throw new InvalidOperationException("Product not found.");
00120             }
00121
00122             try
00123             {
00124                 _context.Products.Remove(product);
00125                 _context.SaveChanges();
00126             }
00127             catch (Exception ex)
00128             {
00129                 throw new InvalidOperationException("Unable to delete product.", ex);
00130             }
00131         }
00132     }
00133 }
```

7.13 C:/Users/Cal-I/Documents/GitHub/HttpWebshopCookie/HttpWebshopCookie/Services/TagService.cs File Reference

Classes

- class HttpWebshopCookie.Services.TagService

  Service class for managing tags.

Namespaces

- namespace HttpWebshopCookie
- namespace HttpWebshopCookie.Services

7.14 TagService.cs

Go to the documentation of this file.

```
00001 namespace HttpWebshopCookie.Services;
00002
00006 public class TagService
00007 {
00008     private readonly ApplicationDbContext context;
00009
00015     public TagService(ApplicationDbContext context)
00016     {
00017         this.context = context ?? throw new ArgumentNullException(nameof(context), "Database context
      cannot be null.");
00018     }
00019
00025     public async Task<List<Tag>> GetTagsAsync()
00026     {
00027         try
00028         {
00029             return await context.Tags.ToListAsync();
00030         }
00031         catch (Exception ex)
00032         {
00033             throw new InvalidOperationException("Unable to retrieve tags.", ex);
00034         }
00035     }
00036
00044     public async Task<Tag> GetTagByIdAsync(string id)
00045     {
00046         if (string.IsNullOrEmpty(id))
```

```
00047         {
00048             throw new ArgumentNullException(nameof(id), "Tag ID cannot be null or empty.");
00049         }
00050 
00051         var tag = await context.Tags.FindAsync(id);
00052         return tag ?? throw new KeyNotFoundException("Tag not found.");
00053     }
00054 
00062     public async Task<Tag> CreateTagAsync(Tag tag)
00063     {
00064         if (tag == null)
00065         {
00066             throw new ArgumentNullException(nameof(tag), "Tag cannot be null.");
00067         }
00068 
00069         try
00070         {
00071             context.Tags.Add(tag);
00072             await context.SaveChangesAsync();
00073             return tag;
00074         }
00075         catch (Exception ex)
00076         {
00077             throw new InvalidOperationException("Unable to create tag.", ex);
00078         }
00079     }
00080 
00088     public async Task<Tag> UpdateTagAsync(Tag tag)
00089     {
00090         if (tag == null)
00091         {
00092             throw new ArgumentNullException(nameof(tag), "Tag cannot be null.");
00093         }
00094 
00095         try
00096         {
00097             context.Tags.Update(tag);
00098             await context.SaveChangesAsync();
00099             return tag;
00100         }
00101         catch (Exception ex)
00102         {
00103             throw new InvalidOperationException("Unable to update tag.", ex);
00104         }
00105     }
00106 
00115     public async Task<Tag> DeleteTagAsync(string id)
00116     {
00117         if (string.IsNullOrEmpty(id))
00118         {
00119             throw new ArgumentNullException(nameof(id), "Tag ID cannot be null or empty.");
00120         }
00121 
00122         var tag = await context.Tags.FindAsync(id);
00123         if (tag == null)
00124         {
00125             throw new KeyNotFoundException("Tag not found.");
00126         }
00127 
00128         try
00129         {
00130             context.Tags.Remove(tag);
00131             await context.SaveChangesAsync();
00132             return tag;
00133         }
00134         catch (Exception ex)
00135         {
00136             throw new InvalidOperationException("Unable to delete tag.", ex);
00137         }
00138     }
00139 
00145     public async Task<List<string?>> GetOccasionsAsync()
00146     {
00147         try
00148         {
00149             return await context.Tags
00150                 .Where(t => t.Occasion != string.Empty)
00151                 .Select(t => t.Occasion)
00152                 .Distinct()
00153                 .ToListAsync();
00154         }
00155         catch (Exception ex)
00156         {
00157             throw new InvalidOperationException("Unable to retrieve occasions.", ex);
00158         }
00159     }
00160 
```

```
00166     public async Task<List<Tag» GetTagsOrderedByOccasionAsync()
00167     {
00168         try
00169         {
00170             return await context.Tags
00171                                 .OrderBy(t => t.Occasion)
00172                                 .ThenBy(t => t.Category)
00173                                 .ThenBy(t => t.SubCategory)
00174                                 .ToListAsync();
00175         }
00176         catch (Exception ex)
00177         {
00178             throw new InvalidOperationException("Unable to retrieve ordered tags.", ex);
00179         }
00180     }
00181
00189     public async Task<(List<Tag>, int)> GetTagsOrderedByOccasionAsync(int pageNumber, int pageSize)
00190     {
00191         try
00192         {
00193             var query = context.Tags
00194                                 .OrderBy(t => t.Occasion)
00195                                 .ThenBy(t => t.Category)
00196                                 .ThenBy(t => t.SubCategory);
00197
00198             var totalItems = await query.CountAsync();
00199             var tags = await query.Skip((pageNumber - 1) * pageSize).Take(pageSize).ToListAsync();
00200
00201             return (tags, totalItems);
00202         }
00203         catch (Exception ex)
00204         {
00205             throw new InvalidOperationException("Unable to retrieve paged tags.", ex);
00206         }
00207     }
00208 }
```

Index