Reference Document for Insurance API

Overview

This document provides an overview of the Insurance API, including how to run the service and detailed information about its endpoints and functionality.

1. Running the Service

To run the Insurance API, follow these steps:

1. **Setup Environment:**
    - Ensure you have .NET SDK installed (version compatible with the project).
    - Install any required dependencies specified in the project file (.csproj).
2. **Build and Run:**
    - Navigate to the project directory.
    - Run the following commands in the terminal or command prompt:

dotnet restore

dotnet build

dotnet run

    - The service should be accessible at **http://localhost:5001** or another port configured in **launchSettings.json**.
3. **Testing:**
    - Use a tool like Postman or curl to test the API endpoints.
    - The integration tests are located in the Insurance.Api.IntegrationTests project and can be run using:

dotnet test

2. API Endpoints

2.1 InsuranceController

Base Route: /api/insurance

Endpoints:

1. **Calculate Insurance for a Single Product**
    - **Method:** POST
    - **Route:** /product

- **Description:** Calculates insurance for a single product based on its ID and other details.
- **Request Body:** InsuranceDto (e.g., { "ProductId": 123 })
- **Responses:**
  - 200 OK: Returns insurance details in InsuranceDto.
  - 400 Bad Request: Invalid product ID.
  - 404 Not Found: Product not found.

2. **Calculate Total Insurance for a Cart**
   - **Method:** POST
   - **Route:** /cart
   - **Description:** Calculates the total insurance value for a list of cart items.
   - **Request Body:** List<InsuranceDto> (e.g., [{ "ProductId": 123 }, { "ProductId": 456 }])
   - **Responses:**
     - 200 OK: Returns total insurance value in TotalInsuranceResponse.
     - 400 Bad Request: Cart is empty or insurance value is zero.

2.2 SurchargeController

Base Route: /api/surcharge

Endpoints:

1. **Upload Surcharge Rates**
   - **Method:** POST
   - **Route:** /upload
   - **Description:** Uploads surcharge rates for product types.
   - **Request Body:** List<ProductSurchargeDto> (e.g., [{ "ProductTypeId": 1, "Surcharge": 150 }])
   - **Responses:**
     - 200 OK: Surcharge rates successfully saved.
     - 400 Bad Request: Invalid surcharge data.

2. **Get Surcharge for a Product Type**

- **Method:** GET
- **Route:** /{productTypeId}
- **Description:** Retrieves the surcharge for a specific product type by its ID.
- **Route Parameters:** productTypeId (e.g., 1)
- **Responses:**
  - 200 OK: Returns surcharge details in ProductSurchargeDto.
  - 400 Bad Request: Product ID cannot be empty.

3. Models and DTOs

- **InsuranceDto:**
  - ProductId (int): The ID of the product.
  - InsuranceValue (decimal): The calculated insurance value for the product.
- **TotalInsuranceResponse:**
  - TotalInsurance (decimal): The total insurance value for the cart.
- **ProductSurchargeDto:**
  - ProductTypeId (int): The ID of the product type.
  - Surcharge (decimal): The surcharge amount for the product type.

4. Integration Tests

The integration tests validate the functionality of the API endpoints. They are located in the Insurance.Api.IntegrationTests project and cover scenarios for both insurance calculations and surcharge management.

- **InsuranceControllerTests:** Tests include verifying insurance calculations for single products and carts.
- **SurchargeControllerTests:** Tests include uploading and retrieving surcharge rates.

5. Helper Methods

- **IsValidProductId(int productId):** Checks if the product ID is valid (greater than 0).
- **FormatInvalidProductMessage(int productId):** Formats an error message for an invalid product ID.
- **FormatProductNotFoundMessage(int productId):** Formats an error message for a product not found.

- **IsCartEmpty(List<InsuranceDto> cartItems):** Checks if the cart is empty