Documentation for Product Fetch and Normalization Module

Step 1: Reading Documentation

Before starting the implementation, I thoroughly reviewed the documentation provided. The task was to create a module that fetches and normalizes product data from multiple sources, including a database and XML files. The products should include the following attributes:
- Title
- Description
- Image link
- Price

This module should ensure that data from multiple sources (like a database table or an XML file) is normalised into a common structure and displayed on a single result page.

Step 2: Creating the Contract

To ensure the system is flexible and extensible, I began by defining a contract, or interface, that would be implemented by any data source (e.g., database, XML) from which products would be fetched.

The contract interface, `ProductSourceInterface`, has two core methods:
- `fetchProducts`: Fetches the product data from the source.
- `normalizeProducts`: Normalizes the fetched product data into a standard format (including attributes like title, description, image_link, and price).

This interface ensures that each product source, regardless of the actual implementation, adheres to a consistent structure and is interchangeable.

```
interface ProductSourceInterface
{
    public function fetchProducts(): array;
    public function normalizeProducts(array $products): array;
}
```

Step 3: Implementing the Source Classes

I created the source classes that implement the `ProductSourceInterface`. These classes are responsible for fetching and normalizing product data from specific sources.

- **__DatabaseProductSource__**:
  - This class fetches product data from a database and normalizes it. The normalization process ensures that each product has the correct fields like `title`, `description`, `image_link`, and `price.

- **XMLProductSource**:
  - Similar to the database source, this class fetches product data from an XML file and normalizes it to the required format.

```
class DatabaseProductSource implements ProductSourceInterface
{
    public function fetchProducts(): array
    {
        // Code to fetch products from the database
    }

    public function normalizeProducts(array $products): array
    {
        // Normalize products to the desired structure
    }
}
```

Step 4: Implementing the Service Class

The **ProductService** class handles the business logic of fetching and normalizing products from multiple sources. It accepts one or more product sources, calls their `fetchProducts` and `normalizeProducts` methods, and aggregates the results.

Here's the class structure:

```
class ProductService
{
    private array $sources = [];

    public function addSource(ProductSourceInterface $source)
    {
        $this->sources[] = $source;
    }

    public function getProducts(): array
    {
        $products = [];
        foreach ($this->sources as $source) {
            $fetchedProducts = $source->fetchProducts();
            $normalizedProducts = $source->normalizeProducts($fetchedProducts);
            $products = array_merge($products, $normalizedProducts);
        }
```

```
        return $products;
    }
}
```

Step 5: Class Diagram

The final step in the task was to create a class diagram that visually represents the structure of the module. The class diagram includes the following components:

1. **ProductService**: This is the central class that manages the various product sources.
2. **ProductSourceInterface**: The interface that all product source classes must implement. It defines the contract for fetching and normalizing products.
3. **DatabaseProductSource** and **XMLProductSource**: These classes implement **ProductSourceInterface** and handle data fetching and normalization from their respective sources (database and XML).
4. **Product**: The product class represents the structure of the product with attributes like title, description, image_link, and price.

The diagram was created using **Draw.io** and is structured as follows:
- **ProductService** has an association with **ProductSourceInterface**, signifying that it uses sources that implement this interface.
- **DatabaseProductSource** and **XMLProductSource** both implement **ProductSourceInterface** using generalization arrows.
- **ProductService** works with **Product** objects, which are returned after the data is normalized.