```
public class Market
{
    private Dictionary<string, decimal> marketPrices;
    private readonly string ConnectionString;

    public Market(string ConnectionString)
    {
        this.ConnectionString = ConnectionString;
        marketPrices = new Dictionary<string, decimal>();
        LoadGoodsFromDatabase(); // Fetch goods from the database
    }

    private void LoadGoodsFromDatabase()
    {
        string sql = "SELECT GoodName, PurchasePrice FROM Goods;";

        using (SQLiteConnection conn = new
SQLiteConnection(DataBaseConfig.ConnectionString))
        {
            conn.Open();
            using (SQLiteCommand cmd = new SQLiteCommand(sql, conn))
            {
                using (SQLiteDataReader reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        string goodName = reader.GetString(0);
                        decimal purchasePrice = reader.GetDecimal(1);

                        if (!marketPrices.ContainsKey(goodName))
                        {
                            marketPrices[goodName] = purchasePrice;
                        }
                    }
                }
            }
        }
    }
    public int GetGoodId(string productName)
    {
        string sql = "SELECT Good_Id FROM Goods WHERE GoodName = @GoodName;";

        using (SQLiteConnection conn = new
SQLiteConnection(DataBaseConfig.ConnectionString))
        {
            conn.Open();
            using (SQLiteCommand cmd = new SQLiteCommand(sql, conn))
            {
```

```
            cmd.Parameters.AddWithValue("@GoodName", productName);
            object result = cmd.ExecuteScalar();

            if (result != null)
            {
                int goodId = Convert.ToInt32(result);
                //Console.WriteLine($"DEBUG: Found Good_Id {goodId} for product
'{productName}'.");
                return goodId;
            }
            else
            {
                Console.WriteLine($"ERROR: No Good_Id found for product name
'{productName}'.");
                return -1;
            }
        }
    }
    public decimal GetMarketPrice(string productName)
    {
        return Math.Round(marketPrices.ContainsKey(productName) ?
marketPrices[productName] : 0, 2);

    }
    public void UpdateMarketPrice()
    {
        Random random = new Random();
        var productNames = new List<string>(marketPrices.Keys);

        foreach (var productName in productNames)
        {
            // Generate a random price change between £0.01 and £0.50
            decimal priceChange = Math.Round((decimal)(random.NextDouble() * 0.49 + 0.01),
2);

            // Randomly decide whether to increase or decrease the price
            bool increase = random.Next(2) == 0; // 50% chance to increase or decrease

            if (increase)
            {
                marketPrices[productName] += priceChange;
            }
            else
            {
                marketPrices[productName] -= priceChange;

                // Ensure the price doesn't drop below £0.01
```

```
            if (marketPrices[productName] < 0.01m)
            {
                marketPrices[productName] = 0.01m;
                marketPrices[productName] = Math.Round(marketPrices[productName], 2);
            }
        }
    }
}
public int SimulateProductSales(Product product)
{
    decimal marketPrice = GetMarketPrice(product.Name);// Get the current marketprice of the product by its name
    if (marketPrice == 0) return 0;
    // Calculates price factor (higher selling price compared to market price means lower price factor)
    double priceFactor = (double)(marketPrice / product.SellingPrice);

    int maxSales = product.Quantity;// maximum amount of sales possible based on the amount of the product available
    int estimatedSales = (int)(maxSales * priceFactor);// Estimates sales based on the price factor and max quantity
    switch (product.Elasticity)// Changes the logic based on the elasticity of the product
    {
        case PEDtype.StrongElastic: // StrongElastic means the lower the price the higher the sales
            estimatedSales = (int)(estimatedSales * Math.Min(priceFactor, 1.8));// Maximum 80% increase
            break;
        case PEDtype.WeakElastic:// Weak elastic means there will still be more sales if the price is lower but not a massive amount more
            estimatedSales = (int)(estimatedSales * Math.Min(priceFactor, 1.2)); // Maximum 20% increase
            break;
        case PEDtype.WeakInelastic:// Weak inelastic means the price can be put slightly higher and sales will remain similiar but there will be a slight decrease
            estimatedSales = (int)(maxSales * 0.8); // Sets estimated to 80% of the maximum quantity, regardless of price
            break;
        case PEDtype.StrongInelastic: // Strong inelastic means price can be put higher and sales will remain generally unaffected
            estimatedSales = (int)((maxSales * 0.5));// Sets stimated sales to 50% of the maximum quantity
            break;
    }
    return Math.Min(estimatedSales, product.Quantity);
}
public Dictionary<string, decimal> GetGoodsByCategory(string category) // go through all products, check their storage type and returns the neccessarry ones
```

```
    {
        var availableGoods = new Dictionary<string, decimal>();

        foreach (var product in marketPrices)
        {
            if (GetStorageType(product.Key).ToString().ToLower() == category.ToLower())
            {
                availableGoods[product.Key] = product.Value;
            }
        }

        return availableGoods;
    }
    public StorageType GetStorageType(string productName) // defines what type of storage the good belongs in
    {
        switch (productName.ToLower())
        {
            case "milk":
            case "yoghurt":
            case "steak":
            case "chicken":
            case "bacon":
                return StorageType.Chilled;
            case "strawberries":
            case "carrots":
            case "bananas":
            case "cabbage":
            case "mangos":
                return StorageType.Fresh;
            case "magnums":
            case "cornettos":
            case "pizza":
            case "turkey":
            case "peas":
                return StorageType.Frozen;
            case "sweets":
            case "chocolate":
            case "crips":
            case "sandwich":
            case "wine":
                return StorageType.Regular;
            default:
                return StorageType.Regular;
        }
    }
    public bool IsElastic(string productName)
    {
```

```
        switch (productName.ToLower())
        {
            case "milk":
            case "carrots":
            case "bananas":
            case "cabbage":
            case "mangos":
            case "strawberries":
            case "peas":
                return false; // ineslastic
            case "yoghurt":
            case "steak":
            case "chicken":
            case "bacon":
            case "magnums":
            case "cornettos":
            case "pizza":
            case "turkey":
            case "sweets":
            case "chocolate":
            case "crips":
            case "sandwich":
            case "wine":
                return true; //elastic
            default: return false;
        }
    }
}
```