**Brewery and Wholesale Management System Documentation**

**Overview**

Brewery and Wholesale Management System is a RESTful API intended for breweries, beers, wholesalers, and the transactions among them. Code architecture modularized under maintainability, scalability, and testability concerns; mostly it consists of the layer architecture: Controllers, Data, DTOs, Models, Repositories, Services.

**Architecture**

The architecture is organized into several layers, each with specific responsibilities. Here is an overview of each layer and their interactions:

1. **Controllers**
2. **DTOs (Data Transfer Objects)**
3. **Models**
4. **Repositories**
5. **Services**
6. **Data**

**1. Controllers**

**Responsibilities:**

* Handle HTTP requests and responses.
* Act as intermediaries between the client and the service layer.
* Validate input and return appropriate HTTP responses.

**Classes:**

* **BeersController**: Manages requests related to beers (e.g., list, add, delete).
* **WholesalerStocksController**: Manages requests related to wholesalers and quotes (e.g., update stock, generate quote).
* **QuoteController**: Manages requests related to quotes (e.g., generate quote).

**Interaction:**

* Controllers receive requests from the client.
* They call the respective methods in the Service layer.
* Return HTTP responses based on the outcome of Service layer operations.

**2. DTOs (Data Transfer Objects)**

**Responsibilities:**

* Transfer data between layers in a format that is easily consumable.
* Encapsulate data for API responses and requests.

**Classes:**

* **BeerDto**: Represents data related to a beer (e.g., ID, Name, AlcoholContent, Price).
* **WholesalerDto**: Represents data related to a wholesaler (e.g., ID, Name).
* **WholesalerStockDto**: Represents data related to a wholesaler stock (e.g., wholesaler ID, wholesaler Name, beer ID, Beer Name, Quantity).
* **QuoteRequestDto**: Represents data for generating a quote (e.g., list of beers and quantities).
* **QuoteDto**: Represents the response for a quote request (e.g., total amount, discount).

**Interaction:**

* DTOs are used by Controllers to receive and send data to/from the client.
* DTOs are passed to the Service layer for processing.

**3. Models**

**Responsibilities:**

* Define the structure of the database entities.
* Map to database tables and represent data in the system.

**Classes:**

* **Beer**: Represents a beer entity in the database.
* **Wholesaler**: Represents a wholesaler entity in the database.
* **Quote**: Represents the response for a quote request.
* **WholesalerStock**: Represents the response for a quote request.

**Interaction:**

* Models are used by Repositories to interact with the database.
* Services use Models to process and manipulate data.

**4. Repositories**

**Responsibilities:**

* Handle data access logic.
* Provide methods for CRUD operations on database entities.
* Abstract data access to ensure separation of concerns.

**Classes:**

* **BeerRepository**: Provides methods for accessing and modifying beer data in the database.
* **WholesalerStockRepository**: Provides methods for accessing and modifying wholesaler data and generating quotes.

**Interaction:**

* Repositories are called by Services to perform data operations.
* They interact directly with the database context to execute queries.

**5. Services**

**Responsibilities:**

* Contain business logic.
* Coordinate operations between different Repositories.
* Handle transactions and complex operations.

**Classes:**

* **BeerService**: Implements business logic related to beers (e.g., add, delete, list).
* **WholesalerStockService**: Implements business logic related to wholesalers (e.g., update stock) and implements business logic related to quote generation (e.g., apply discounts, validate orders).

**Interaction:**

* Services are called by Controllers to perform business operations.
* They utilize Repositories to interact with the database.

**6. Data**

**Responsibilities:**

* Manage database context and configurations.
* Define database schema and relationships.

**Classes:**

* **ApplicationDbContext**: Entity Framework Core context for managing database connections and sets.

**Interaction:**

* The DbContext is used by Repositories to perform data operations.
* It is configured in the application's startup class.

**Data Flow**

1. **Client Request**: The client sends an HTTP request to the API.
2. **Controller**: The request is handled by a Controller which validates the input.
3. **Service**: The Controller calls the appropriate method in the Service layer.
4. **Repository**: The Service layer interacts with the Repository to perform database operations.
5. **Data**: The Repository uses the DbContext to execute database queries.
6. **Response**: The Service layer processes the results and sends them back to the Controller.
7. **Client Response**: The Controller returns the HTTP response to the client.

**Example Scenarios**

1. **Listing Beers by Brewery**:
   * **Controller**: BeerController receives the request.
   * **Service**: Calls BeerService to retrieve the beers.
   * **Repository**: BeerRepository queries the database for beers.
   * **DTO**: Returns a list of BeerDto to the client.
2. **Adding a New Beer**:
   * **Controller**: BeerController processes the request.
   * **Service**: Calls BeerService to add a new beer.
   * **Repository**: BeerRepository inserts the new beer into the database.
   * **DTO**: Confirms the addition and returns a success response.

**Summary of Layer Interaction**

1. **Client Request** → **Controller**: Receives and processes the HTTP request.
2. **Controller** → **Service**: Calls the Service layer for business logic.
3. **Service** → **Repository**: Calls the Repository layer to perform database operations.
4. **Repository** → **Data**: Uses the DbContext to interact with the database.
5. **Data** → **Repository**: Returns data to the Repository layer.
6. **Repository** → **Service**: Returns data to the Service layer.
7. **Service** → **Controller**: Returns processed data to the Controller.
8. **Controller** → **Client**: Sends the final HTTP response back to the client.

N.B:

- **Models**: Define the structure of data as it is stored in the database and are used by the Data and Repository layers to perform database operations.

* **Data Layer**: Models are primarily used in the Data Layer to interact with the database. They define the structure of tables and relationships.
* **Repository Layer**: The Repository Layer uses models to perform CRUD operations and queries against the database.

- **DTOs**: Shape the data that is transferred between layers, particularly between the Service and Controller layers, and format the data for client consumption:

* **Controller Layer**: DTOs are used in the Controller Layer to receive input from the client and to format responses.
* **Service Layer**: DTOs are used to pass data between the Service Layer and the Controller Layer. They are often used to transform and validate data.
* **Repository Layer**: DTOs may also be used to shape the results returned by the Repository Layer, especially if complex queries are involved or if the data needs to be transformed before being returned to the Service Layer.