

**Durian Store**

**(Mobile Online Shop)**

**Software Design Specification**

*Hanoi, May 20*

[**I. Overview**](#_gjdgxs) **4**

[1. Code Packages](#) 4

[1.1. Package descriptions](#) 4

[2. Database Design](#) 5

[2.1. Database Schema](#_1fob9te) 5

[2.2. Table Description](#_3znysh7) 5

[**II. Code Designs**](#_2et92p0) **7**

[1.1. Access Page (Homepage, Login, Logout, Forgot password)](#_3dy6vkm) 7

[a. Class diagram:](#) 7

[b. Class specification:](#) 7

[c. Sequence Diagram(s)](#) 8

[d. Database queries](#) 10

[1.2. Product (List ProductVariant, Product Details, Search Product by Name, Filter Product)](#_1t3h5sf) 11

[a. Class diagram:](#) 11

[b. Class specification:](#) 11

[c. Sequence Diagram(s)](#) 13

[d. Database queries](#_nmv7zix0pg0i) 14

[1.3. Admin dashboard( Add Product , Add Product Variant, Update Product,](#_9xc9h3vayhyy) 16

[Update Product Variant, Delete Product, Delete Product Variant, Insert Product, Insert Product Variant)](#_ip1rtumtp3jo) 16

[a. Class diagram:](#_8y0vtn78c7o) 16

[b. Class specification:](#_q7d7hpffl7ff) 16

[c. Sequence Diagram(s):](#_4s94pqpnbouw) 19

[d. Database queries:](#_t6cm187lsyyk) 20

[1.4. Account profile: Account profile, Update profile, ChangePassword](#_vcezkudme70c) 22

[a. Class diagram:](#_1ymrir2b6mm) 22

[b. Class specification:](#_x6rvivsroedk) 22

[c. Sequence Diagram(s):](#_n332th8xqgv0) 24

[d. Database queries:](#_lufscnu1gn44) 24

[1.5. Sale dashboard( Add Sale, Update Sale, Delete Sale, Insert Sale)](#_8t46ml74m1ya) 26

[a. Class diagram:](#_yr19w36ce8ad) 26

[b. Class specification:](#_lqaprjda1b2b) 26

[c. Sequence Diagram(s):](#_2fo458tgnp24) 27

[d. Database queries:](#_wjus92gdnvv9) 27

[1.6. My Order List & My Order Detail](#_4e1ye3ucirf4) 28

[a. Class diagram](#_e52dkno8tcxs) 28

[b. Class specification](#_u4mr9j3nbgyr) 28

[c. Sequence diagram](#_hjqhbc1r1guu) 32

[d. Database queries](#_sor9d5r5qtm7) 32

[1.7. Cart (Add to Cart, View Cart, Cart Checkout)](#_gekcsi25fo01) 35

[a. Class diagram](#_qrhk7pnzsfnj) 35

[b. Class specification](#_vosfls51c4n7) 35

[c. Sequence diagram](#_n6mg7jtqhn6a) 39

[d. Database queries](#_4wby95lzu8tp) 39

# **I. Overview**

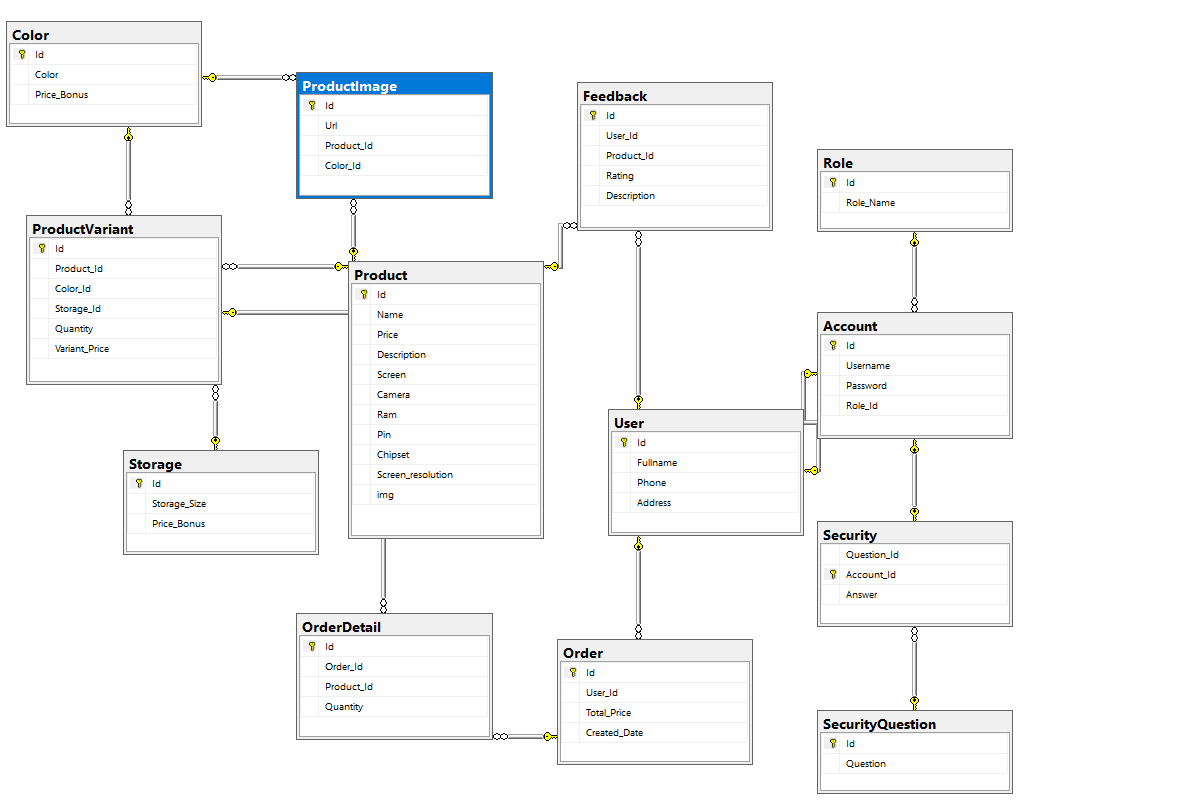
## **Code Packages**

### **Package descriptions**

| **No** | **Package** | **Description** |
| --- | --- | --- |
| 01 | Controller | This directory contains the files and controller classes responsible for handling requests and managing the flow of data between the user interface and logic. |
| 02 | Dao | This directory contains files and the DAO layer that is responsible for accessing and managing data in the database. |
| 03 | Utilities | This directory provides support functions for system software. |
| 04 | Model | This folder is used to store and manage files and resources related to the model. |

## **Database Design**

### **2.1. Database Schema**

****

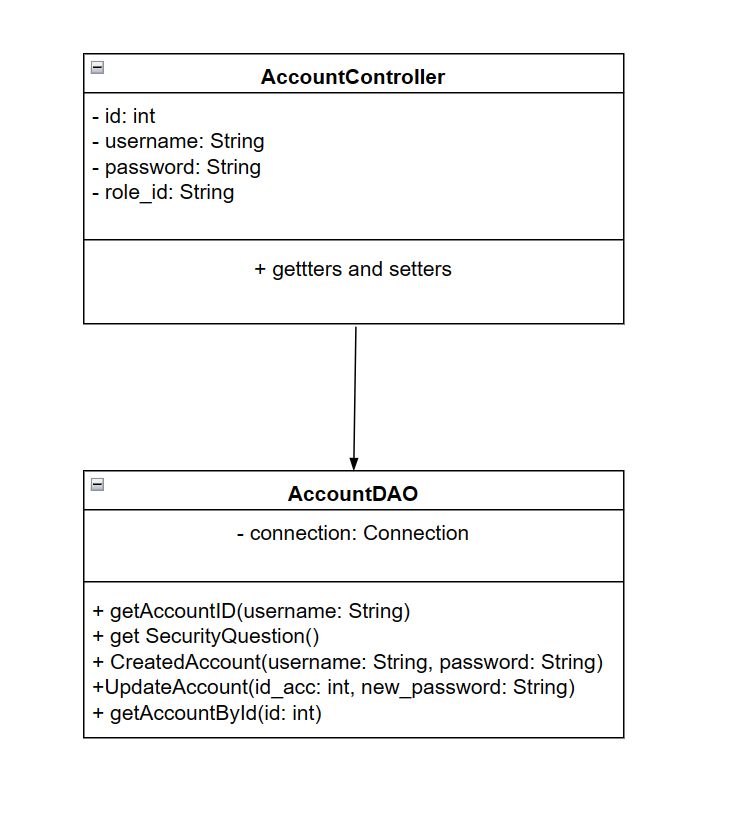
### **2.2. Table Description**

| **No** | **Table** | **Description** |
| --- | --- | --- |
| 1 | **Account** | *This table stores information about user accounts, including username ID, Password, Username and Role\_Id.* |
| 2 | **Order** | *It represents the orders placed by users, including information such as ID, User\_Id, Total\_Price and Created\_Date.* |
| 3 | **ProductImage** | *This table stores the images associated with each product. It may include fields such as product image ID, URL, Product\_Id and Color\_Id.* |
| 4 | **Color** | *This table stores product color information, including color ID, Color and Price\_Bonus.* |
| 5 | **Feedback** | *It stores user feedback or reviews about products or the overall shopping experience. It may include fields such as feedback ID, User\_Id, Product\_Id, Rating and Description.* |
| 6 | **User** | *It stores user information. It can include fields like ID, FullName, Phone, Address.* |
| 7 | **Security** | *It stores user account security answers. It can include fields like Qusetion\_id, Account\_ID, Answer* |
| 8 | **SecurityQuestion** | *It stores user account security questions. It can include fields like Qusetion\_id, Account\_ID, Answer.* |
| 9 | **OrderDetail** | *This table contains detailed information about each product in an order, including the Product ID, Quantity, Order\_ID.* |
| 10 | **Product** | *It stores information about individual products available in the Durian shop, including Name, Price, Description, Screen, Camera, Ram, Pin, Chipset, Img, Sceen\_resolution .* |
| 11 | **Storage** | *This table indicates to store memory information including components Strorage\_Size, Price\_Bonus.* |
| 12 | **Product Variant** | *This table holds product variations information including Products\_Id,Color\_Id, Storage\_Id, Quantity, Variant\_price.* |
| 13 | **Role** | *This table contains account role information including component Role\_Name* |

# **II. Code Designs**

### **Access Page (Homepage, Login, Logout, Forgot password)**

#### ***Class diagram:***



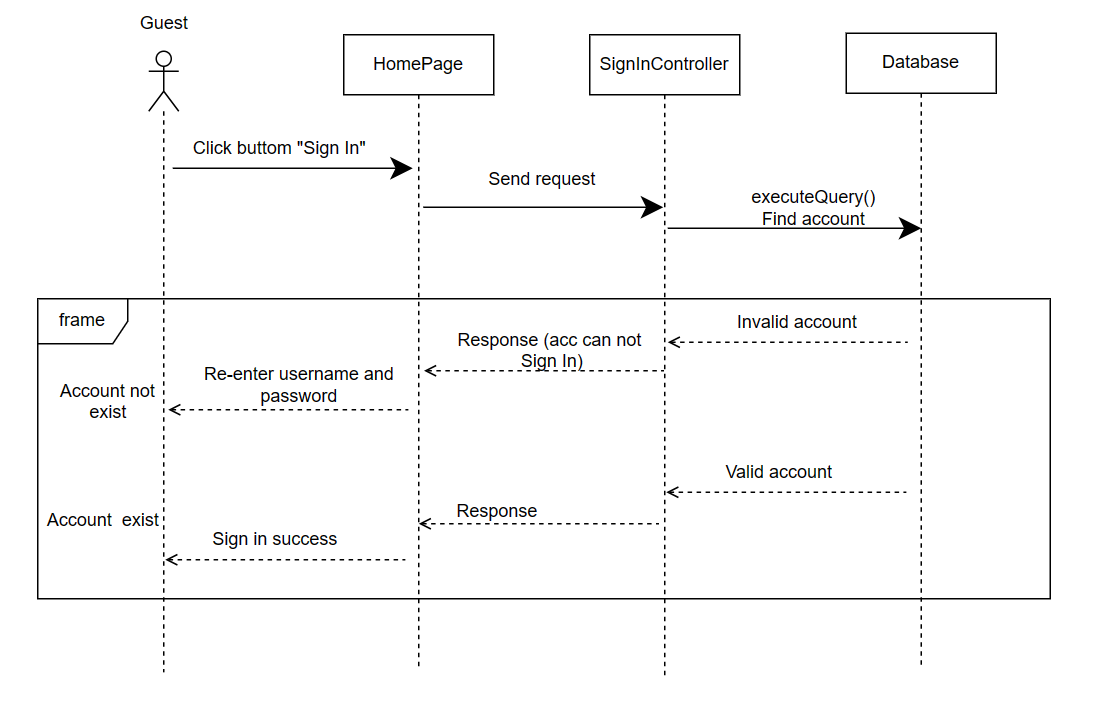
#### ***Class specification:***

* class AccountDAO

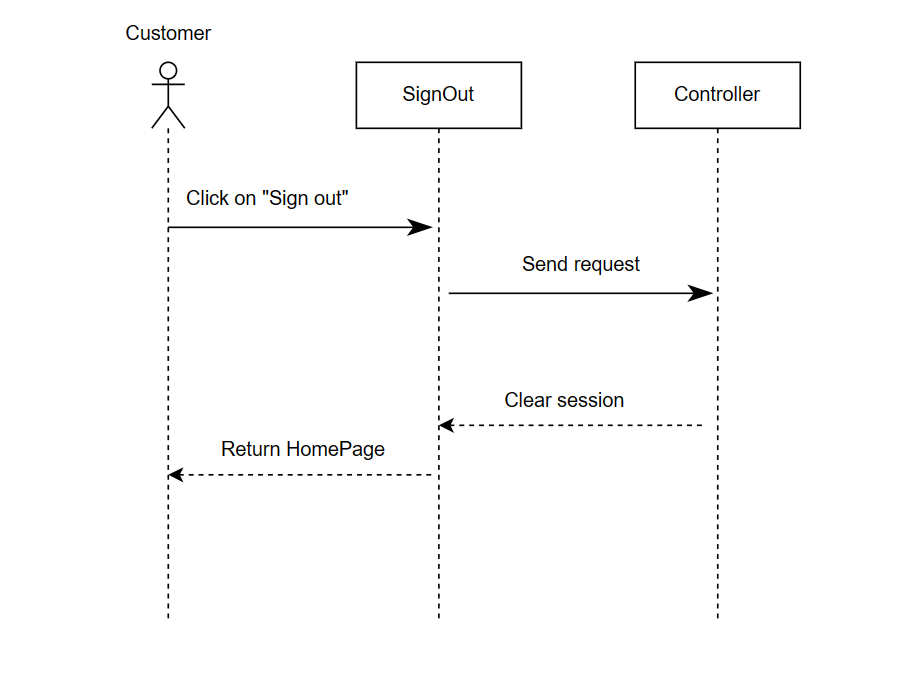
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAccountID(username: String) | This method to get information about the user account's ID from the database. It executes an SQL query to fetch all account records and stores them in a list of Account\_model objects. It returns the list of accounts. |
| 02 | get SecurityQuestion() | This method to get the set of security questions for the user account. It executes an SQL query to get all query records and stores them in a list, which is used to identify and verify user identity in necessary situations. The list of security questions is returned as an object: ArrayList<SecurityQuestion> |
| 03 | CreatedAccount(username: String, password: String) | This method is responsible for creating a new user account. It takes the username, password, role\_id. It executes an SQL insert statement to add account details to the database. |
| 04 | UpdateAccount(id\_acc: int, new\_password: String) | This method is responsible for updating the new password after the change of the user account. It executes an SQL insert statement to update the password back into the database. |
| 05 | getAccountById(id: int) | This method to get account information based on the provided account ID. It executes a query for the columns of the table "Account" in the database "[DURIAN\_SHOP]" provided that the ID of the account must match the parameter value. |

#### ***Sequence Diagram(s)***

* ***Sign in***:



* ***Sign out***:

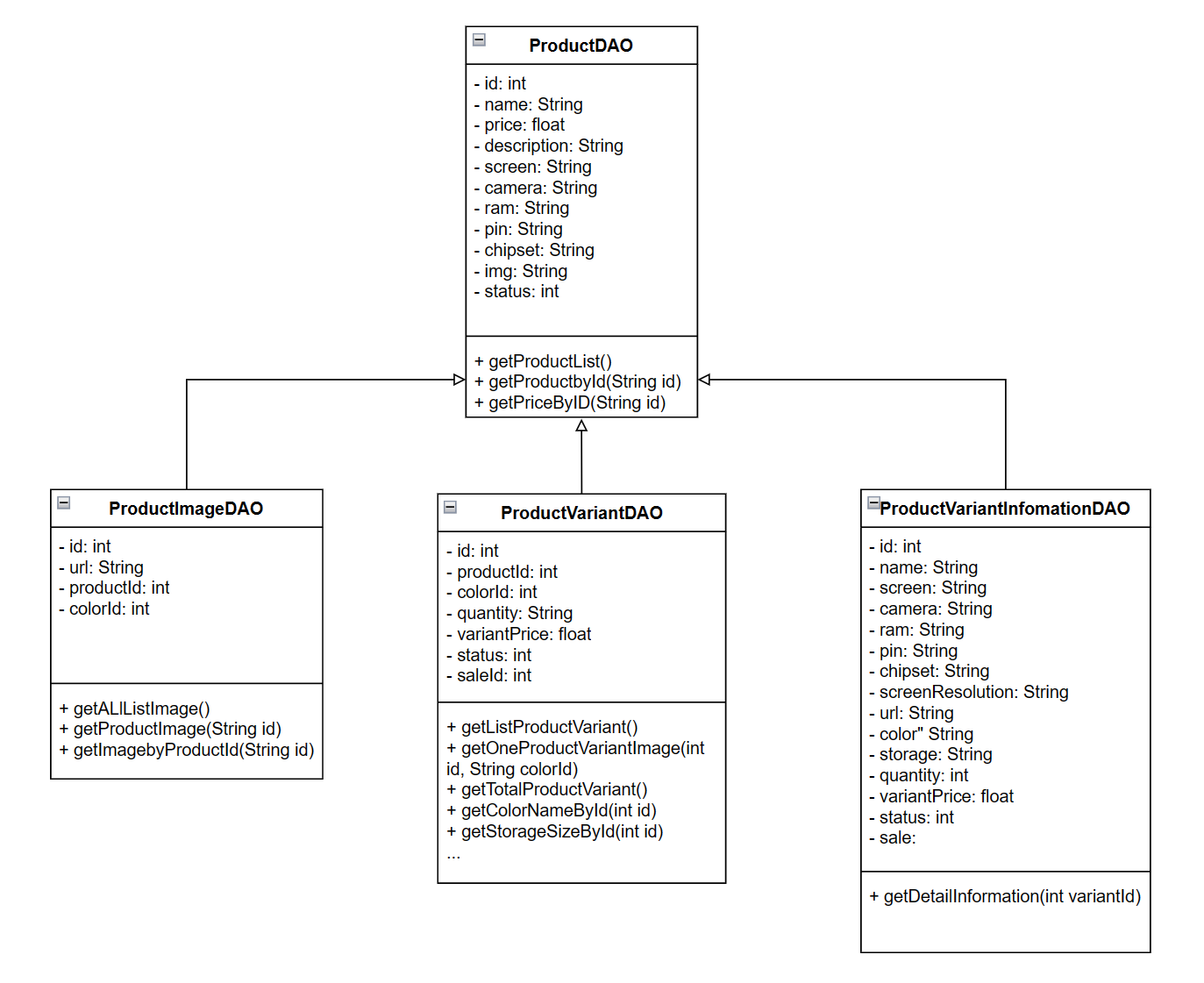


#### ***Database queries***

| Database queries | Detail |
| --- | --- |
| getAccountID(String userName) | Select Id from Account where Account.Username like ? |
| getSecurityQuestion() | select \* from SecurityQuestion |
| createAccount(String userName, String passWord) | Insert into [Account]([Username], [Password], [Role\_Id]) values ('" + userName + "', '" + passWord + "', 2) |
| updatePassword(int id\_acc, String new\_passWord) | UPDATE [Account] SET [Password] = ? WHERE Id = ? |
| getAccountById(int id) | select \* from [DURIAN\_SHOP].[dbo].[Account] "  + " Where [Id] = ? |

### **Product (List ProductVariant, Product Details, Search Product by Name, Filter Product)**

#### ***Class diagram:***



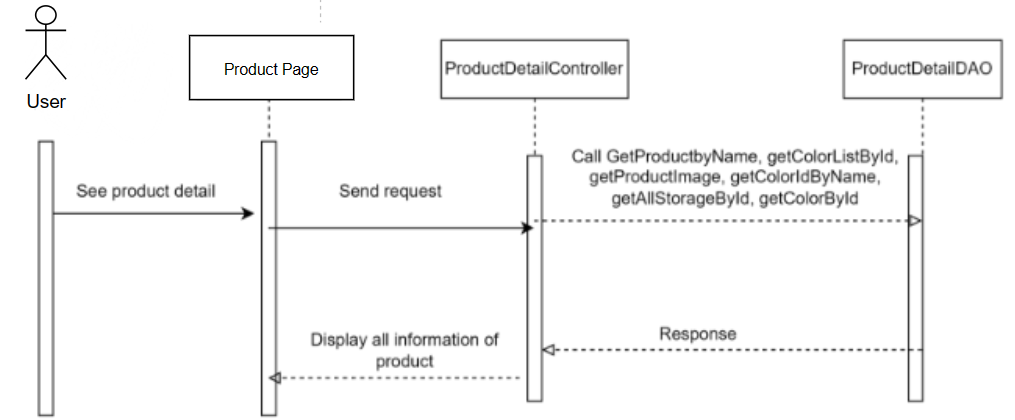
#### ***Class specification:***

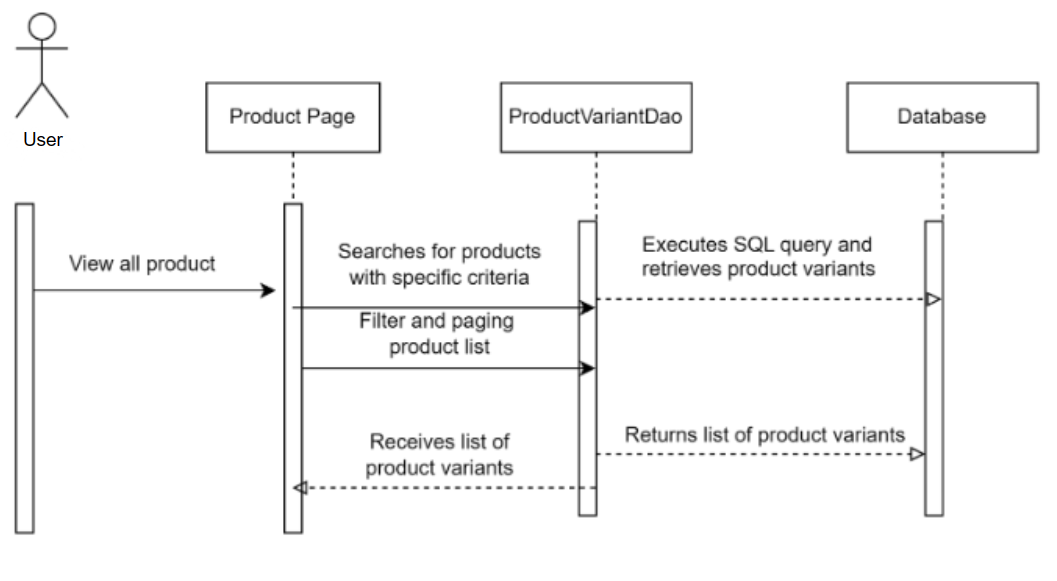
* Class ProductDetailDao:

| No | Method | Description |
| --- | --- | --- |
| 01 | getProductbyName | This method retrieves all products by name from the database. It executes an SQL query to select all product names from the "Products" instance. Then iterate over the result set and create ProductDetailDTO objects for each row. The values from the result set are assigned to the corresponding properties of the ProductDetailDTO object. Finally this method returns a list of ProductDetailDTO objects containing all products sorted by name. |
| 02 | getProductImage | This method retrieves all the productimg data from the database. It executes an SQL query to select all rows from the "ProductImage" table. Then iterate over the results and create a ProductDetailDTO object for each row. The values from the result set are assigned the corresponding properties of the ProductDetailDTO object. Finally, the method returns a list of images of the products |
| 03 | getColorIdByName | This method retrieves a category from the database based on a specific column name and input value. It takes the column name and input as parameters, executes an SQL query with a WHERE clause using the provided column name and input, creates aProductDetailDTO object for the retrieved row, sets the attributes of the object based on the retrieved data, and returns the object. |
| 04 | getAllStorageList | This method use to get all storage data from database . It executes all row and column form the storage table. The values from the result set are assigned the corresponding properties of the ProductDetailDTO object. Finally, the method returns a list of storage. |
| 05 | getStorageById | This method get all products from database with condition of id. It executes a SQL query to select all rows from the "Storage" table with the condition of id. It then iterates over the result set and creates Storage objects for each row. The values from the result set are assigned to the corresponding properties of the Storageobject. Finally, the method returns a list of Storage by Id. |
| 06 | getColorById | This method get all products from database with condition of id. It executes a SQL query to select all rows from the "Colors" table with the condition of id. It then iterates over the result set and creates Color objects for each row. The values from the result set are assigned to the corresponding properties of the Color object. Finally, the method returns a list of Color by Id. |
| 07 | getListProductVariant | This method retrieves all the products from the database. It executes a SQL query to select all rows from the "Product" table. It then iterates over the result set and creates ProductVariantDTO objects for each row. The values from the result set are assigned to the corresponding properties of the ProductVariantDTO object. Finally, the method returns a list of Product objects containing all the productvariants. |

#### ***Sequence Diagram(s)***

* View all product:



* View product detail :
* 

#### ***Database queries***

* getProductbyName: String query = "SELECT \* FROM product"

+ " Where product.name = ?";

* getProductImage:
* String query = "Select \* from ProductImage "

+ " Where productImage.product\_id = ? ";

if (!colorId.equals("")){

query+= " AND color\_id = " + colorId;

* getColorIdByName: String query = "Select id from color "

+ "Where color = ?";

* getAllStorageList: String query = "Select \* from Storage";
* getStoragebyId: String query = "SELECT \* FROM storage"

+ " Where id = ?";

* getColorbyId: String query = "SELECT \* FROM color"

+ " Where id = ?";

* getColorListById: String query = "Select distinct c.id, c.color, c.price\_bonus from ProductImage p join "

+ " Color c on p.Color\_Id = c.Id"

+ " Where p.Product\_id = ?";

* getListProductVariant: String query = "select \* from productVariant"

+ " where 1=1";

* getOneProductVariantImage: String query = "Select top(1) \* from ProductImage "

+ " Where productImage.product\_id = ? ";

if (!colorId.equals("")) {

query += " AND color\_id = " + colorId;

}

* getTotalProductVariant: String query = "select count(\*) from productVariant"

+ " where 1=1";

* SortProduct: int min = range.getMin();

int max = range.getMax();

query += "AND (variant\_price BETWEEN "+min +" AND "+max+") ";

* getStorageSizeById: String query = "Select [Storage\_Size] from [DURIAN\_SHOP].[dbo].[Storage] "

+ "Where id = ?";

* getListColor: String query = "SELECT \* FROM color";
* getListStorage: String query = "SELECT \* FROM storage";
* getListIdProductByName: String query = "Select id from product "

+ "where name like '%"+ search+ "%'";

* filterColor: if (!filterColor.isEmpty()) {

query += " and ";

if (filterColor.size() == 1){

query+= "color\_id =" + filterColor.get(0);

}

if (filterColor.size()>1){

query+= "(";

for (int i = 0; i < filterColor.size(); i++) {

if (i <=filterColor.size() - 2){

query+="color\_id = " + filterColor.get(i) + " or ";

}if (i == filterColor.size() -1){

query+="color\_id = " + filterColor.get(i) + ") ";

}

}

}

}

* filterStorage: if (!filterStorage.isEmpty()) {

query += " and ";

if (filterStorage.size() == 1){

query+= "storage\_id =" + filterStorage.get(0);

}

if (filterStorage.size()>1){

query+= "(";

for (int i = 0; i < filterStorage.size(); i++) {

if (i <=filterStorage.size() - 2){

query+="storage\_id = " + filterStorage.get(i) + " or ";

}

if (i == filterStorage.size() -1){

query+="storage\_id = " + filterStorage.get(i) + ") ";

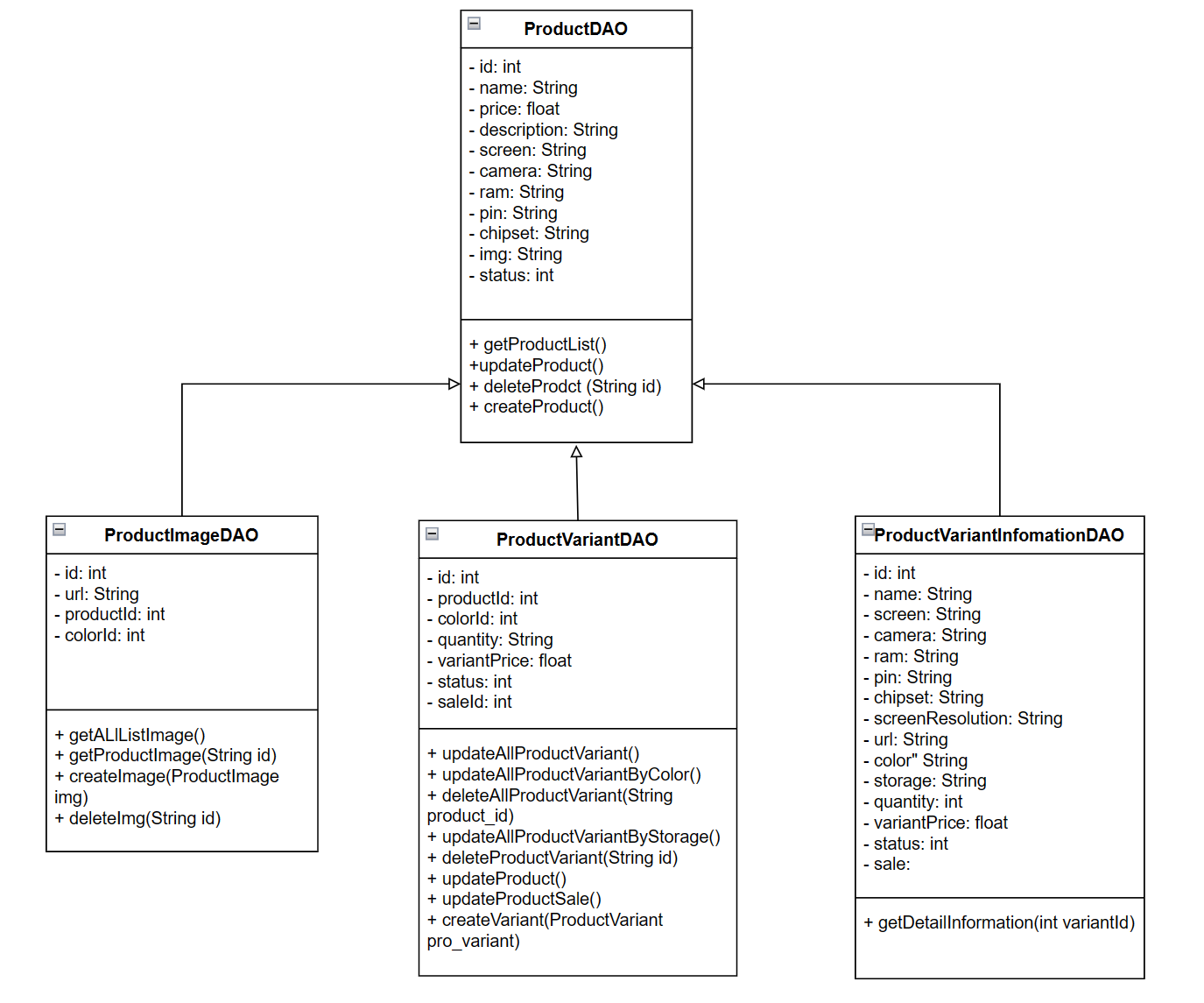
}

} } }

### **Admin dashboard( Add Product , Add Product Variant, Update Product,**

### **Update Product Variant, Delete Product, Delete Product Variant, Insert Product, Insert Product Variant)**

#### ***Class diagram:***



#### ***Class specification:***

* Class ProductDao:

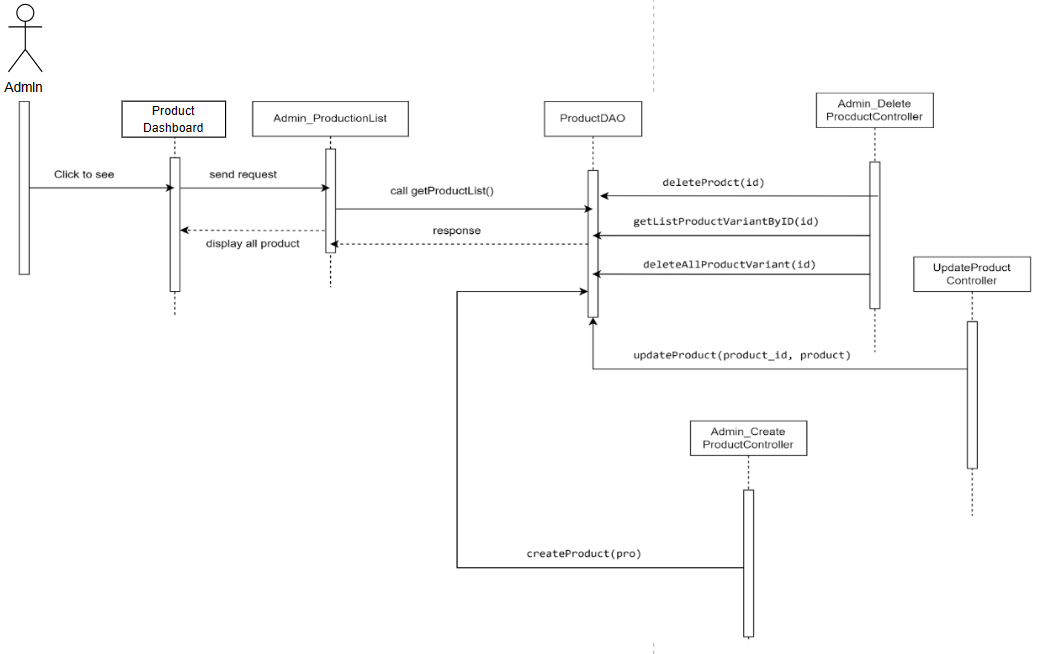
| No | Method | Description |
| --- | --- | --- |
| 01 | getProductList() | This method retrieves all the products from the database. It executes a SQL query to select all rows from the "Product" table. It then iterates over the result set and creates ProductDTO objects for each row. The values from the result set are assigned to the corresponding properties of the ProductDTO object. Finally, the method returns a list of Product objects containing all the products. |
| 02 | getHashMapProduct() | This method gets the product id and name from the database. It executes an SQL query to get the name and id data from the "Products" table. |
| 03 | getProductByID(int id) | This method get all products from database with condition of id. It executes a SQL query to select all rows from the "Products" table with the condition of id. It then iterates over the result set and creates Product objects for each row. The values from the result set are assigned to the corresponding properties of the Product object. Finally, the method returns a list of Product objects containing all the products. |
| 04 | updateProduct(int id, Product pro) | This method is used to add a new product to the database.You need to add the code to execute the SQL query that inserts a new product into the "Product" table. You can use the provided method parameters (id, name, price, description, screen, camera, ram, pin, chipset, screenResolution, img) to set the values for the new product. |
| 05 | deleteProdct (String id) | This method is used to delete a product from the database. You need more code to execute the SQL query that deletes a product with the id you want with the "Products" table. |
| 06 | createProduct(Product product) | This method is used to add a new Product from the database. It executes an SQL query to add a new product to the "Product" table. Then it creates Products objects for each row. The values from the result set are assigned to the corresponding properties of the Products object. |

* Class ProductVariantDao:

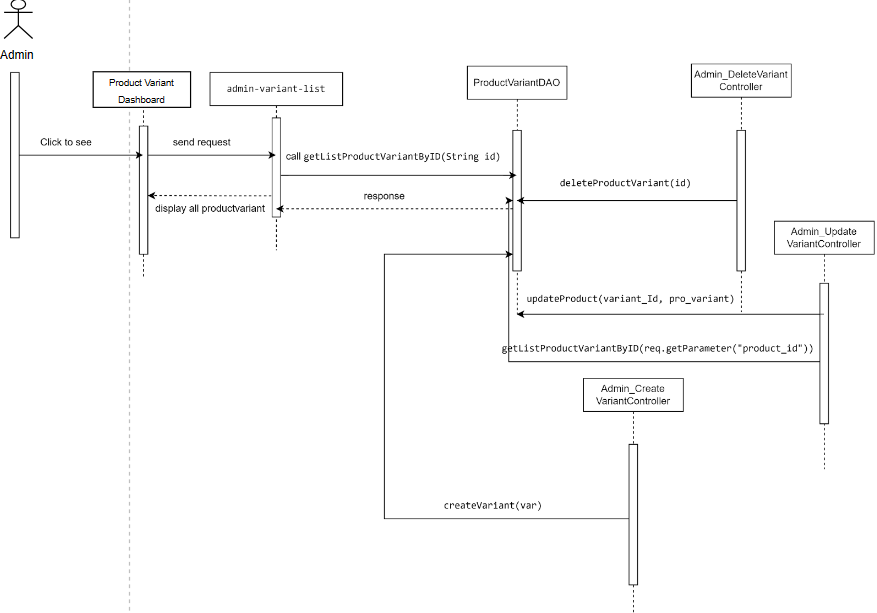
| No | Method | Description |
| --- | --- | --- |
| 01 | getProductVariantByID(int id) | This method get all ProductsVariants from database with condition of id. It executes a SQL query to select all rows from the "ProductsVariant" table with the condition of id. It then iterates over the result set and creates ProductsVariant objects for each row. The values from the result set are assigned to the corresponding properties of the ProductsVariant object. Finally, the method returns a list of ProductsVariant objects of all products that have been filtered by the condition. |
| 02 | getColdProductVariant() | This method is used to get every product in the highest quantity. It executes an SQL query to select all the rows from the "ProductVariant" table provided that Quantity equals the greatest. then iterate over the result set and create ProductVariant objects for each row. |
| 03 | getListProductVariantByID(String id) | This method retrieves all Product Variations from the database as long as product\_id is given. It executes an SQL query to select all rows from the "ProductsVariant" table provided that product\_id is given. It then iterates over the result set and creates ProductsVariant objects for each row. The values from the result set are assigned to the corresponding properties of the ProductsVariant object. Finally, the method returns a list of ProductsVariant objects of all products that have been filtered by the condition. |
| 04 | deleteAllProductVariant(String product\_id) | This method is used to delete a ProductsVariant from the database. You need more code to execute the SQL query that deletes a product with the id you want with the "ProductsVariant" table. |
| 05 | deleteProductVariant(String id) | This method is used to delete a ProductsVariant from the database. You need more code to execute the SQL query that deletes a product with the id you want with the "ProductsVariant" table. |
| 06 | updateProduct(int variant\_Id, ProductVariant pro\_variant) | This method is used to add a new ProductsVariant to the database. You need to add the code to execute the SQL query that inserts a new product into the "ProductsVariant" table. You can use the provided method parameters (id, color\_id, storage\_id, quantity, variant\_price) to set the values for the new ProductsVariant. |
| 07 | createVariant(ProductVariant pro\_variant) | This method is used to add a new Product Variant from the database. It executes an SQL query to add a new product to the "ProductsVariant" table. Then it creates ProductsVariant objects for each row. The values from the result set are assigned to the corresponding properties of the ProductsVariant object. |

#### ***Sequence Diagram(s):***

* Class ProductDAO:

****

* Class ProductVariantDao:



#### ***Database queries:***

* Class ProductDao:
* getProductList(): "SELECT \* FROM product"
* getHashMapProduct(): "select Id, [Name] from [Product]"
* getProductByID(int id): "SELECT \* FROM product where id = "
* updateProduct(int id, Product pro): "UPDATE [Product] SET Name = '" + pro.getName() + "', Price = " + pro.getPrice() +

", Description = '" + pro.getDescription() +

"', Screen = '" + pro.getScreen() +

"', Camera = '" + pro.getCamera() +

"', Ram = '" + pro.getRam() +

"', Pin = '" + pro.getPin() +

"', Chipset = '" + pro.getChipset() +

"', Screen\_resolution = '" +pro.getScreenResolution() +

"', img = '" + pro.getImg() +

"' where id = " + id;

* deleteProdct(String id): "Delete FROM [Product] WHERE Id LIKE"
* createProduct(Product product): "INSERT INTO [Product]([Name], [Price], [Description], [Screen], [Camera], [Ram], [Pin], [Chipset], [Screen\_resolution], [img])" +

"values(' " + product.getName() + "', " + product.getPrice() + ", '" + product.getDescription() + "', '" + product.getScreen() +

"', '" + product.getCamera() + "', '" + product.getRam() + "', '" + product.getPin() + "', '" + product.getChipset() + "', '" +

product.getScreenResolution() + "', '" + product.getImg() + "')"

* Class ProductVariantDao:
* getProductVariantByID(int id): "SELECT \* FROM [ProductVariant] WHERE id LIKE "
* getColdProductVariant(): "select \* from [ProductVariant] where Quantity = (select MAX(Quantity) from [ProductVariant]) "
* getListProductVariantByID(String id): "SELECT \* FROM [ProductVariant] WHERE Product\_Id LIKE "
* deleteAllProductVariant(String product\_id): "Delete FROM [ProductVariant] WHERE Product\_Id LIKE "
* deleteProductVariant(String id): "Delete FROM [ProductVariant] WHERE Id LIKE "
* updateProduct(int variant\_Id, ProductVariant pro\_variant):

"UPDATE [ProductVariant] SET Color\_Id = '" + pro\_variant.getColor\_Id() +

"', Storage\_Id = '" + pro\_variant.getStorage\_Id() +

"', Quantity = '" + pro\_variant.getQuantity() +

"', Variant\_Price = '" + pro\_variant.getVariant\_Price() +

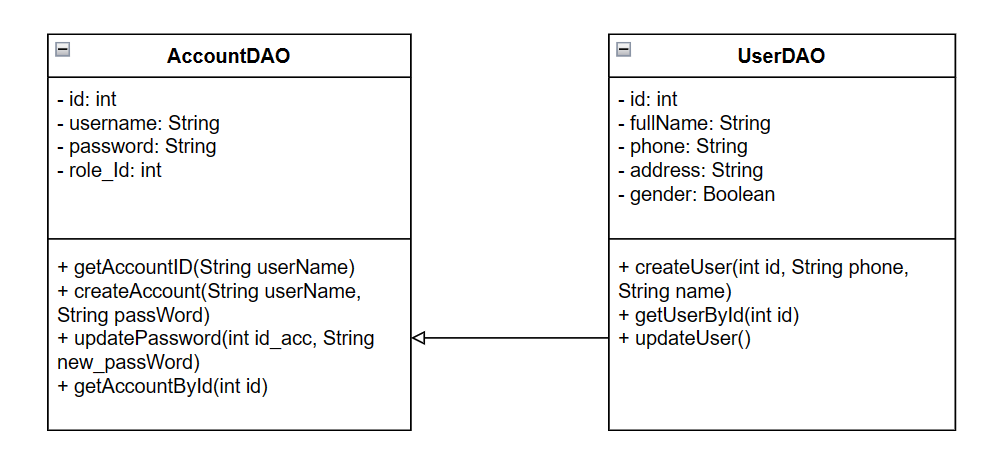
"' where id = " + variant\_Id;

* createVariant(ProductVariant pro\_variant): "INSERT INTO [ProductVariant]([Product\_Id],[Color\_Id],[Storage\_Id],[Quantity],[Variant\_Price])"

### 

### **Account profile: Account profile, Update profile, ChangePassword**

#### ***Class diagram:***



#### ***Class specification:***

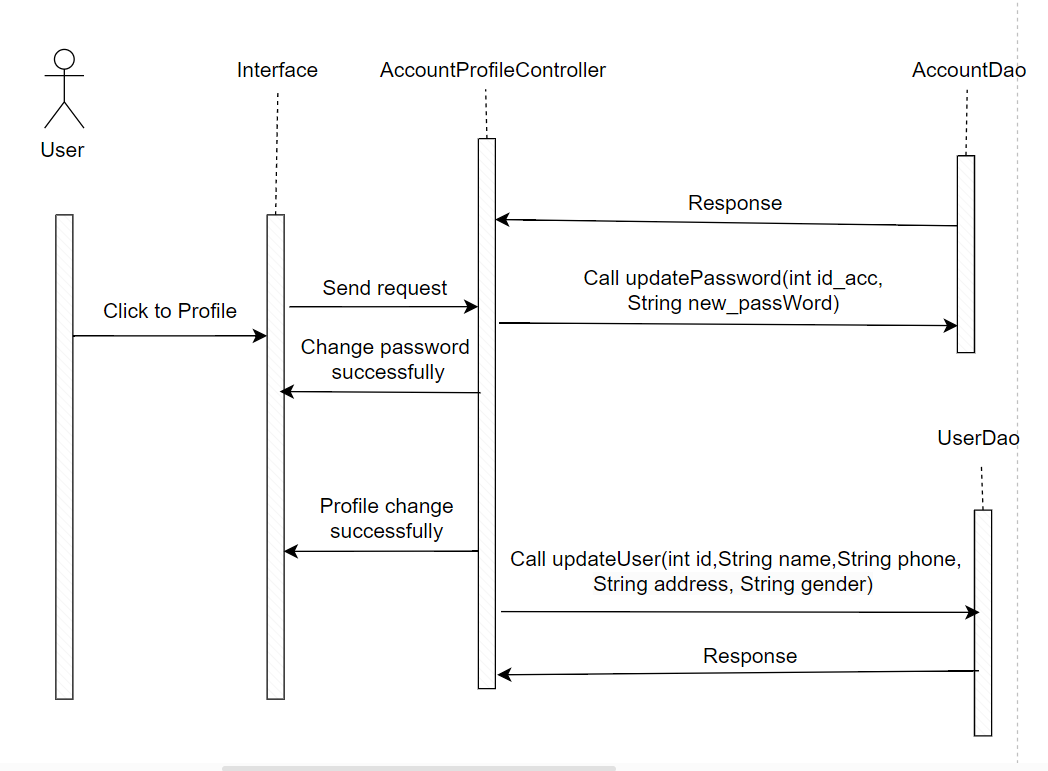
* class UserDao :

| No | Method | Description |
| --- | --- | --- |
| 01 | updateUser | The updateUser method is responsible for updating the user information in the system based on the input parameters. Search for the user with the ID identifier in the system and update their information based on the name, phone parameters. , address , and gender |
| 02 | getAccountById | The getAccountById method is responsible for retrieving the account information from the system based on the identifier (id) provided. use the identifier (id) to search and retrieve the corresponding account information in the system and other information related to the account. |

* class AccountDao:

| No | Method | Description |
| --- | --- | --- |
| 01 | getUserById | The getUserById method is responsible for retrieving user information from the system based on the provided identifier (id). The getUserById method will use the id identifier to search and retrieve the corresponding user information in the system and other information related to the user. |
| 02 | updatePassword | TThe updatePassword method is responsible for updating the password for the account in the system based on the input parameters. The method will look for the account with the identifier (id\_acc) in the system and update the password of that account with the password. The new password is provided in the (new\_password) parameter. After performing the updatePassword method, the account's password will be changed to the new password you provided. |
| 03 | getAccountID | The getAccountID method is responsible for retrieving the account identifier from the system based on the username (userName) provided. The getAccountID method will use the username userName to search and retrieve the corresponding account identifier in the system. |

#### ***Sequence Diagram(s):***



#### ***Database queries:***

* + - 1. Class AccountDAO:

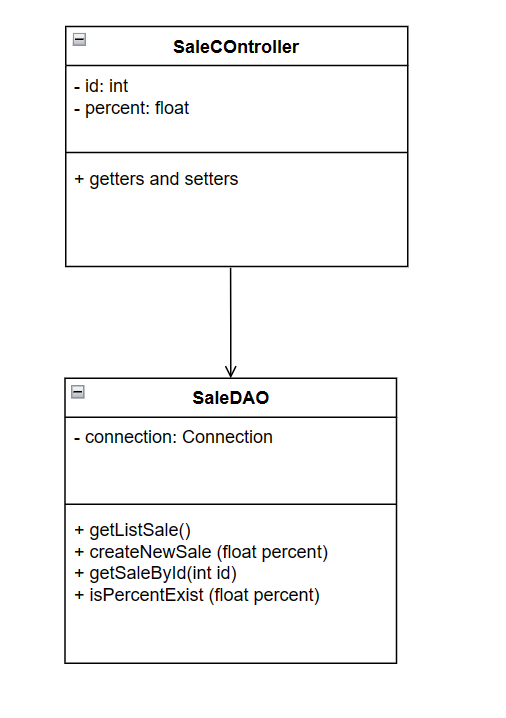
| Database queries | Detail |
| --- | --- |
| getAccountID(String userName) | "Select Id from Account where Account.Username like ?" |
| getAccountById(int id) | "select \* from [DURIAN\_SHOP].[dbo].[Account] "  + " Where [Id] = ?"; |
| updatePassword(int id\_acc, String new\_passWord) | "UPDATE [Account] SET [Password] = ? WHERE Id = ?" |

* + - 1. Class UserDAO:

| Database queries | Detail |
| --- | --- |
| getUserById(int id) | "Select \* from [DURIAN\_SHOP].[dbo].[User]"  + " Where [id] = ?"; |
| getAccountById(int id) | "select \* from [DURIAN\_SHOP].[dbo].[Account] "  + " Where [Id] = ?"; |
| updateUser(int id,String name, String phone, String address, String gender) | "UPDATE [DURIAN\_SHOP].[dbo].[User] " + " SET fullName = ?, phone = ?, address = ?"; |

### **Sale dashboard( Add Sale, Update Sale, Delete Sale, Insert Sale)**

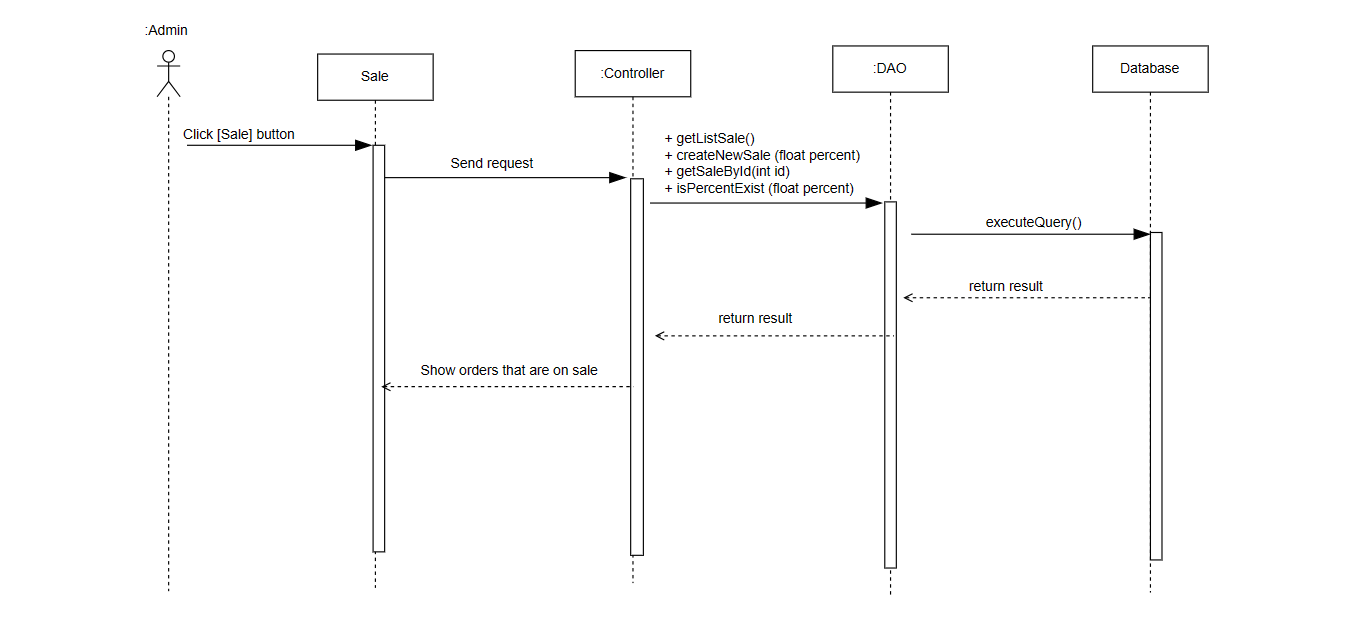
#### **a.** C**lass diagram:**

******

#### ***b. Class specification:***

| No | Method | Description |
| --- | --- | --- |
| 01 | getSaleList() | This method retrieves all the products from the database. It executes a SQL query to select all rows from the "Sale" table. It then iterates over the result set and creates SaleDTO objects for each row. The values from the result set are assigned to the corresponding properties of the SaletDTO object. Finally, the method returns a list of Product objects containing all the products. |
| 02 | getSaleByID(int id) | This method get all products from database with condition of id. It executes a SQL query to select all rows from the "Sale" table with the condition of id. It then iterates over the result set and creates Sale objects for each row. The values from the result set are assigned to the corresponding properties of the Sale object. Finally, the method returns a list of Sale objects containing all the products. |
| 03 | updateSale(int id, Product pro) | This method is used to add a new product to the database.You need to add the code to execute the SQL query that inserts a new product into the "Product" table. You can use the provided method parameters (id, name, price, description, screen, camera, ram, pin, chipset, screenResolution, img) to set the values for the new product. |
| 04 | deleteSale(String id) | This method is used to delete a product from the database. You need more code to execute the SQL query that deletes a product with the id you want with the "Products" table. |
| 05 | createSale(Product product) | This method is used to add a new Product from the database. It executes an SQL query to add a new product to the "Product" table. Then it creates Products objects for each row. The values from the result set are assigned to the corresponding properties of the Products object. |

#### ***c. Sequence Diagram(s):***

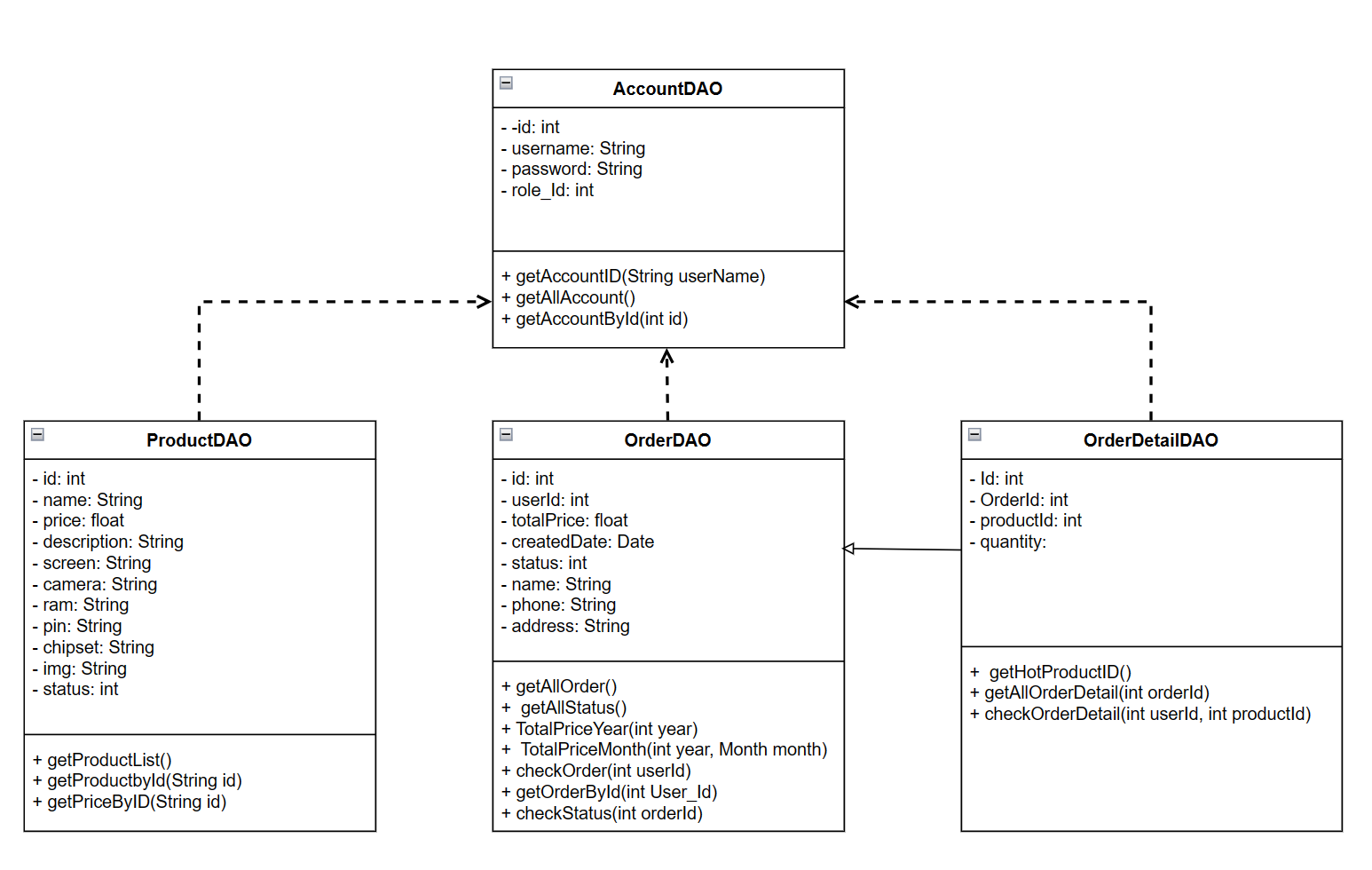


#### ***d. Database queries:***

| Database queries | Detail |
| --- | --- |
| getListSale() | select \* from [Sale] order by [percent] ASC |
| getSaleById(int id) | SELECT \* FROM sale"  + " Where id = ? |
| isPercentExist (float percent) | SELECT \* FROM sale"  + " Where [percent] = ? |
| createNewSale (float percent) | Insert Into [Sale]([percent])"  + " values ("+percent+") |
| getNewSale () | SELECT Top(1) [Id], [Percent] FROM sale"  + " ORDER BY [Id] DESC |

### **Order List & Order Status**

#### ***Class diagram***



#### ***Class specification***

* Class AccountDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAccountID(username: String) | This method to get information about the user account's ID from the database. It executes an SQL query to fetch all account records and stores them in a list of Account\_model objects. It returns the list of accounts. |
| 02 | get SecurityQuestion() | This method to get the set of security questions for the user account. It executes an SQL query to get all query records and stores them in a list, which is used to identify and verify user identity in necessary situations. The list of security questions is returned as an object: ArrayList<SecurityQuestion> |
| 03 | CreatedAccount(username: String, password: String) | This method is responsible for creating a new user account. It takes the username, password, role\_id. It executes an SQL insert statement to add account details to the database. |
| 04 | UpdateAccount(id\_acc: int, new\_password: String) | This method is responsible for updating the new password after the change of the user account. It executes an SQL insert statement to update the password back into the database. |
| 05 | getAccountById(id: int) | This method to get account information based on the provided account ID. It executes a query for the columns of the table "Account" in the database "[DURIAN\_SHOP]" provided that the ID of the account must match the parameter value. |

* Class OderDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllOrdergetAll() | The method takes a list of orders from the database, creates Order objects based on the obtained data and stores them in an ArrayList, and then returns a populated list. It executes SELECT query to get all columns from "Order" table where the condition applied is the year of field "Created\_Date" is 2023. |
| 02 | TotalPriceYear(int year) | The method gets the total value of orders for a particular year from the database by summing the Total\_Price values of the matching orders and returning the calculated total. It executes a SELECT query to get the Total\_Price column from the "Order" table, where the year of the Created\_Date field matches the year input parameter. |
| 03 | TotalPriceMonth(int year, Month month) | This method calculates the total value of orders for a particular month in a specified year from the database. It executes a SELECT query to get all the columns from the "Order" table, where the year of the Created\_Date field matches the year input parameter and the month of the Created\_Date field matches the month. |
| 04 | checkOrder(int userId) | This method checks for the existence of an order in the database based on a specific userId. It executes a SELECT query to get the Id column from the "Order" table, where the status column equals 1 and the User\_Id column matches the userId input value. |
| 05 | createOrder(int User\_id, double Total\_Price) | The method creates a new order in the database by inserting User\_id, Total\_Price, the current date and time, and a status value. It executes a query to insert values into the "Order" table. |
| 06 | getOrderById(int User\_Id) | The method retrieves an order from the database based on a specific User\_Id by querying the "Order" table. It returns the order as an Order object. It executes a SELECT query to get all columns from the "Order" table where the User\_Id column matches the User\_Id input value. |

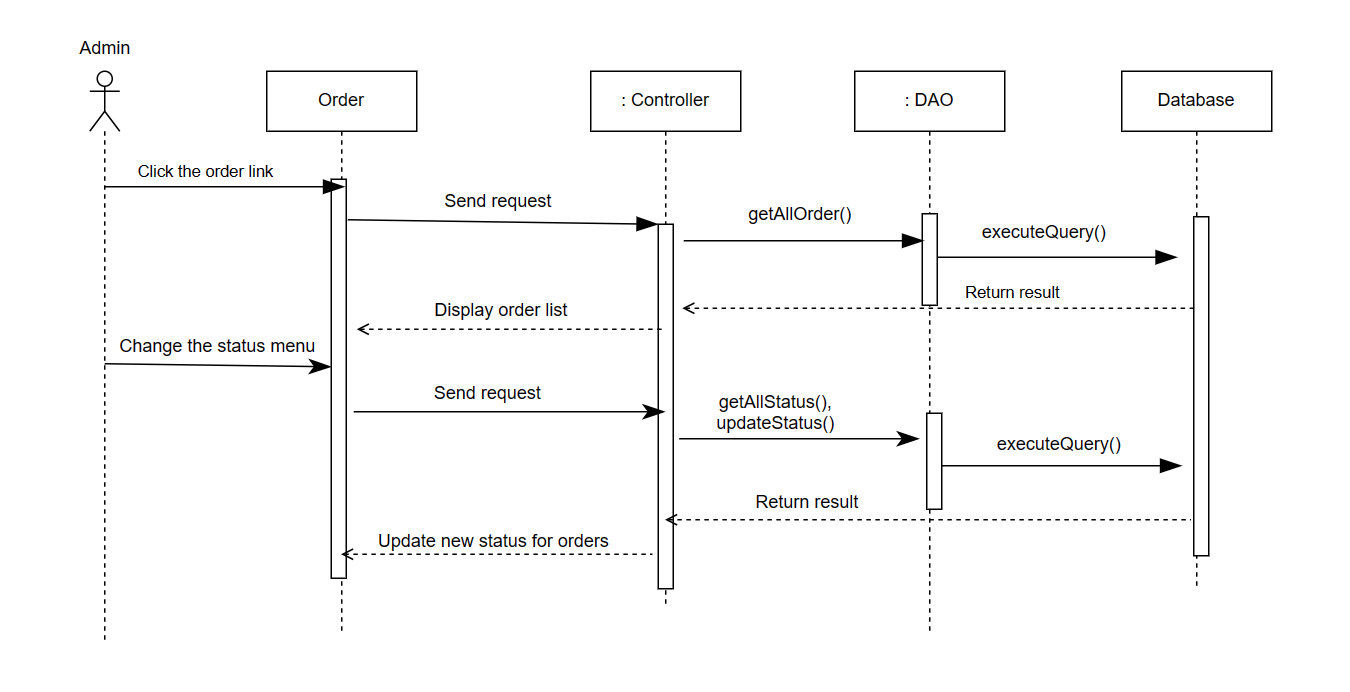
* Class OrderDetailDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getHotProductID() | The method retrieves the ID of the best-selling product from the OrderDetail table using an SQL query to determine the product with the highest sales volume. It executes a SELECT query to get the Product\_Id column from the OrderDetail table. |
| 02 | insertOrderDetail(int Order\_Id, int Product\_Id, int Quantity) | The method inserts order details into the database by providing the Order\_Id, Product\_Id, and Quantity values. Create an SQL query string to insert the value into the "OrderDetail" table. |
| 03 | getAllOrderDetail(int orderId) | The method gets all the order details for a particular orderId from the "OrderDetail" table and returns them as an ArrayList<OrderDetail>. It executes a SELECT query to get all columns from the "OrderDetail" table where the Order\_Id column matches the input orderId. |
| 04 | updateQuantity(int detailId, int quantity) | The method updates the quantity of an order item in the database by providing detailId and quantity values. An SQL query string is generated to update the Quantity column of the "OrderDetail" table based on the Id column. |
| 05 | deleteDetail(int detailId) | The method to delete an order item from the database based on the detailId provided. The SQL query string is generated to delete rows from the "OrderDetail" table where the Id column matches the input detailId. |

* Class ProductDAO

| No | Method | Description |
| --- | --- | --- |
| 01 | getProductList() | This method retrieves all the products from the database. It executes a SQL query to select all rows from the "Product" table. It then iterates over the result set and creates ProductDTO objects for each row. The values from the result set are assigned to the corresponding properties of the ProductDTO object. Finally, the method returns a list of Product objects containing all the products. |
| 02 | getHashMapProduct() | This method gets the product id and name from the database. It executes an SQL query to get the name and id data from the "Products" table. |
| 03 | getProductByID(int id) | This method get all products from database with condition of id. It executes a SQL query to select all rows from the "Products" table with the condition of id. It then iterates over the result set and creates Product objects for each row. The values from the result set are assigned to the corresponding properties of the Product object. Finally, the method returns a list of Product objects containing all the products. |
| 04 | updateProduct(int id, Product pro) | This method is used to add a new product to the database.You need to add the code to execute the SQL query that inserts a new product into the "Product" table. You can use the provided method parameters (id, name, price, description, screen, camera, ram, pin, chipset, screenResolution, img) to set the values for the new product. |
| 05 | deleteProdct (String id) | This method is used to delete a product from the database. You need more code to execute the SQL query that deletes a product with the id you want with the "Products" table. |
| 06 | createProduct(Product product) | This method is used to add a new Product from the database. It executes an SQL query to add a new product to the "Product" table. Then it creates Products objects for each row. The values from the result set are assigned to the corresponding properties of the Products object. |

#### ***Sequence diagram***



#### ***Database queries***

* Class AccountDao

| Database queries | Detail |
| --- | --- |
| getAccountID(String userName) | Select Id from Account where Account.Username like ? |
| getSecurityQuestion() | select \* from SecurityQuestion |
| createAccount(String userName, String passWord) | Insert into [Account]([Username], [Password], [Role\_Id]) values ('" + userName + "', '" + passWord + "', 2) |
| updatePassword(int id\_acc, String new\_passWord) | UPDATE [Account] SET [Password] = ? WHERE Id = ? |
| getAccountById(int id) | select \* from [DURIAN\_SHOP].[dbo].[Account] "  + " Where [Id] = ? |

* Class OrderDao

| Database queries | Detail |
| --- | --- |
| getAllOrder() | select \* from [Order] where YEAR(Created\_Date) = 2023 |
| TotalPriceYear(int year) | select Total\_Price from [Order] where YEAR(Created\_Date) =" + year |
| TotalPriceMonth(int year, Month month) | select \* from [Order] where YEAR(Created\_Date) =" + year + "and MONTH(Created\_Date) = " + month |
| checkOrder(int userId) | select Id from [Order] where [status] = 1 and [User\_Id] = " + userId |
| createOrder(int User\_id, double Total\_Price) | Insert into [DURIAN\_SHOP].[dbo].[Order]([User\_Id], [Total\_Price], [Created\_Date],[Status]) values (?,?,(SELECT GETDATE()),1) |
| getOrderById(int User\_Id) | select \* from [Order] where [Order].[User\_Id] = " + User\_Id |

* Class OrderDetailDao

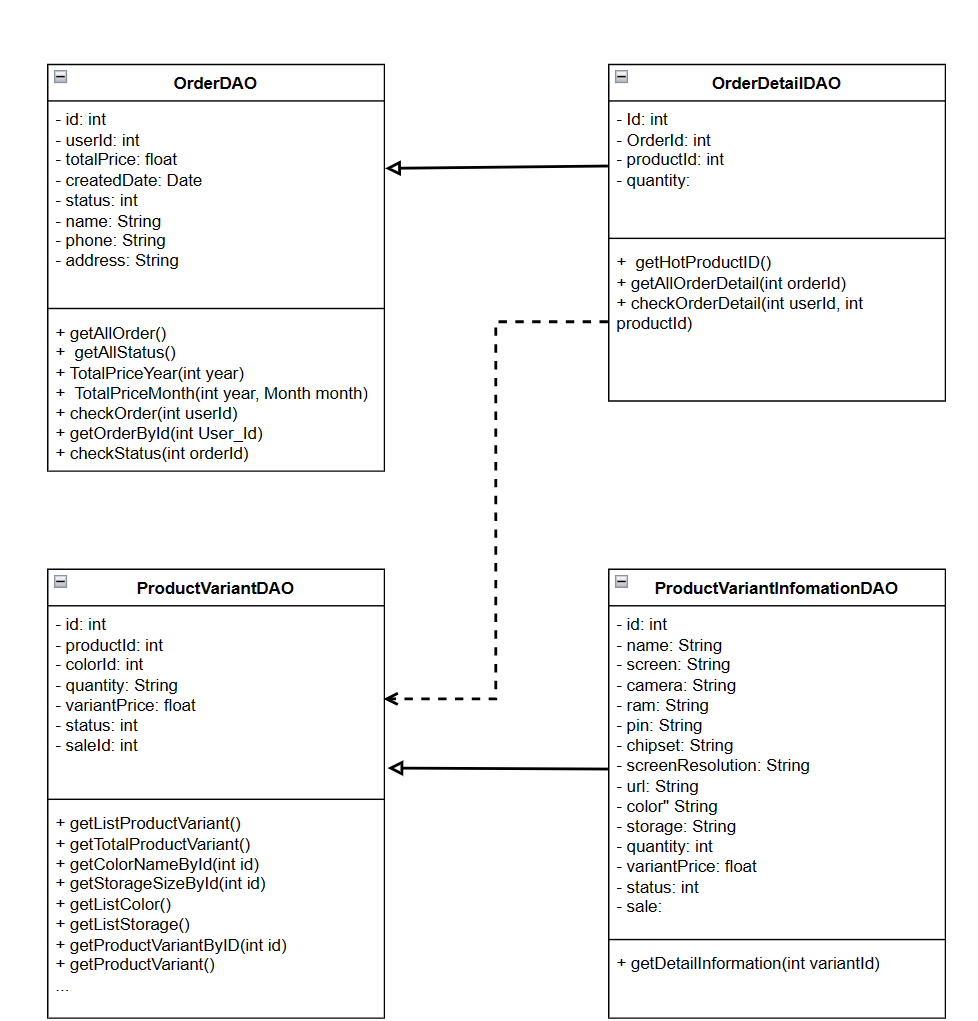
| Database queries | Detail |
| --- | --- |
| getHotProductID() | SELECT Product\_Id FROM OrderDetail GROUP BY Product\_Id\n"  + "HAVING SUM(Quantity) = ( SELECT MAX(TotalQuantity) FROM ( SELECT Product\_Id, SUM(Quantity) AS TotalQuantity FROM OrderDetail\n"  + "GROUP BY Product\_Id ) AS Subquery); |
| insertOrderDetail(int Order\_Id, int Product\_Id, int Quantity) | Insert into [DURIAN\_SHOP].[dbo].[OrderDetail]([Order\_Id], [Product\_Id], [Quantity]) values (?,?,?) |
| getAllOrderDetail(int orderId) | select \* from [OrderDetail] where [Order\_Id] = " + orderId; |
| updateQuantity(int detailId, int quantity) | UPDATE [OrderDetail] SET [Quantity] = ? WHERE Id = ? |
| deleteDetail(int detailId) | delete from [OrderDetail] where [Id] = ? |

* Class ProductDao

| Database queries | Detail |
| --- | --- |
| getProductList() | SELECT \* FROM product |
| getHashMapProduct() | select Id, [Name] from [Product] |
| getProductbyId(String id) | SELECT \* FROM product"  + " Where product.id = ? |
| getProductByID(int id) | SELECT \* FROM product where id = " + id |
| updateProduct(int id, Product pro) | UPDATE [Product] SET Name = '" + pro.getName() +  "', Price = " + pro.getPrice() +  ", Description = '" + pro.getDescription() +  "', Screen = '" + pro.getScreen() +  "', Camera = '" + pro.getCamera() +  "', Ram = '" + pro.getRam() +  "', Pin = '" + pro.getPin() +  "', Chipset = '" + pro.getChipset() +  "', Screen\_resolution = '" + pro.getScreenResolution() +  "', img = '" + pro.getImg() +  "' where id = " + id; |
| updateProduct(String id, String status) | UPDATE [Product] SET " +  "status = " + status +  " where id = " + id |
| deleteProdct (String id) | Delete FROM [Product] WHERE Id LIKE '" + id + "' |
| createProduct(Product product) | INSERT INTO [Product]([Name], [Price], [Description], [Screen], [Camera], [Ram], [Pin], [Chipset], [Screen\_resolution], [img])" +  "values(' " + product.getName() + "', " + product.getPrice() + ", '" + product.getDescription() + "', '" + product.getScreen() +  "', '" + product.getCamera() + "', '" + product.getRam() + "', '" + product.getPin() + "', '" + product.getChipset() + "', '" +  product.getScreenResolution() + "', '" + product.getImg() + "') |
| getPriceByID(String id) | select [Price] from [DURIAN\_SHOP].[dbo].[Product]"  + " Where product.id = ? |

### **Cart (Add to Cart, View Cart, Cart Checkout)**

#### ***Class diagram***



#### ***Class specification***

* Class OrderDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllOrdergetAll() | The method takes a list of orders from the database, creates Order objects based on the obtained data and stores them in an ArrayList, and then returns a populated list. It executes SELECT query to get all columns from "Order" table where the condition applied is the year of field "Created\_Date" is 2023. |
| 02 | TotalPriceYear(int year) | The method gets the total value of orders for a particular year from the database by summing the Total\_Price values of the matching orders and returning the calculated total. It executes a SELECT query to get the Total\_Price column from the "Order" table, where the year of the Created\_Date field matches the year input parameter. |
| 03 | TotalPriceMonth(int year, Month month) | This method calculates the total value of orders for a particular month in a specified year from the database. It executes a SELECT query to get all the columns from the "Order" table, where the year of the Created\_Date field matches the year input parameter and the month of the Created\_Date field matches the month. |
| 04 | checkOrder(int userId) | This method checks for the existence of an order in the database based on a specific userId. It executes a SELECT query to get the Id column from the "Order" table, where the status column equals 1 and the User\_Id column matches the userId input value. |
| 05 | createOrder(int User\_id, double Total\_Price) | The method creates a new order in the database by inserting User\_id, Total\_Price, the current date and time, and a status value. It executes a query to insert values into the "Order" table. |
| 06 | getOrderById(int User\_Id) | The method retrieves an order from the database based on a specific User\_Id by querying the "Order" table. It returns the order as an Order object. It executes a SELECT query to get all columns from the "Order" table where the User\_Id column matches the User\_Id input value. |

* Class OrderDetailDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getHotProductID() | The method retrieves the ID of the best-selling product from the OrderDetail table using an SQL query to determine the product with the highest sales volume. It executes a SELECT query to get the Product\_Id column from the OrderDetail table. |
| 02 | insertOrderDetail(int Order\_Id, int Product\_Id, int Quantity) | The method inserts order details into the database by providing the Order\_Id, Product\_Id, and Quantity values. Create an SQL query string to insert the value into the "OrderDetail" table. |
| 03 | getAllOrderDetail(int orderId) | The method gets all the order details for a particular orderId from the "OrderDetail" table and returns them as an ArrayList<OrderDetail>. It executes a SELECT query to get all columns from the "OrderDetail" table where the Order\_Id column matches the input orderId. |
| 04 | updateQuantity(int detailId, int quantity) | The method updates the quantity of an order item in the database by providing detailId and quantity values. An SQL query string is generated to update the Quantity column of the "OrderDetail" table based on the Id column. |
| 05 | deleteDetail(int detailId) | The method to delete an order item from the database based on the detailId provided. The SQL query string is generated to delete rows from the "OrderDetail" table where the Id column matches the input detailId. |

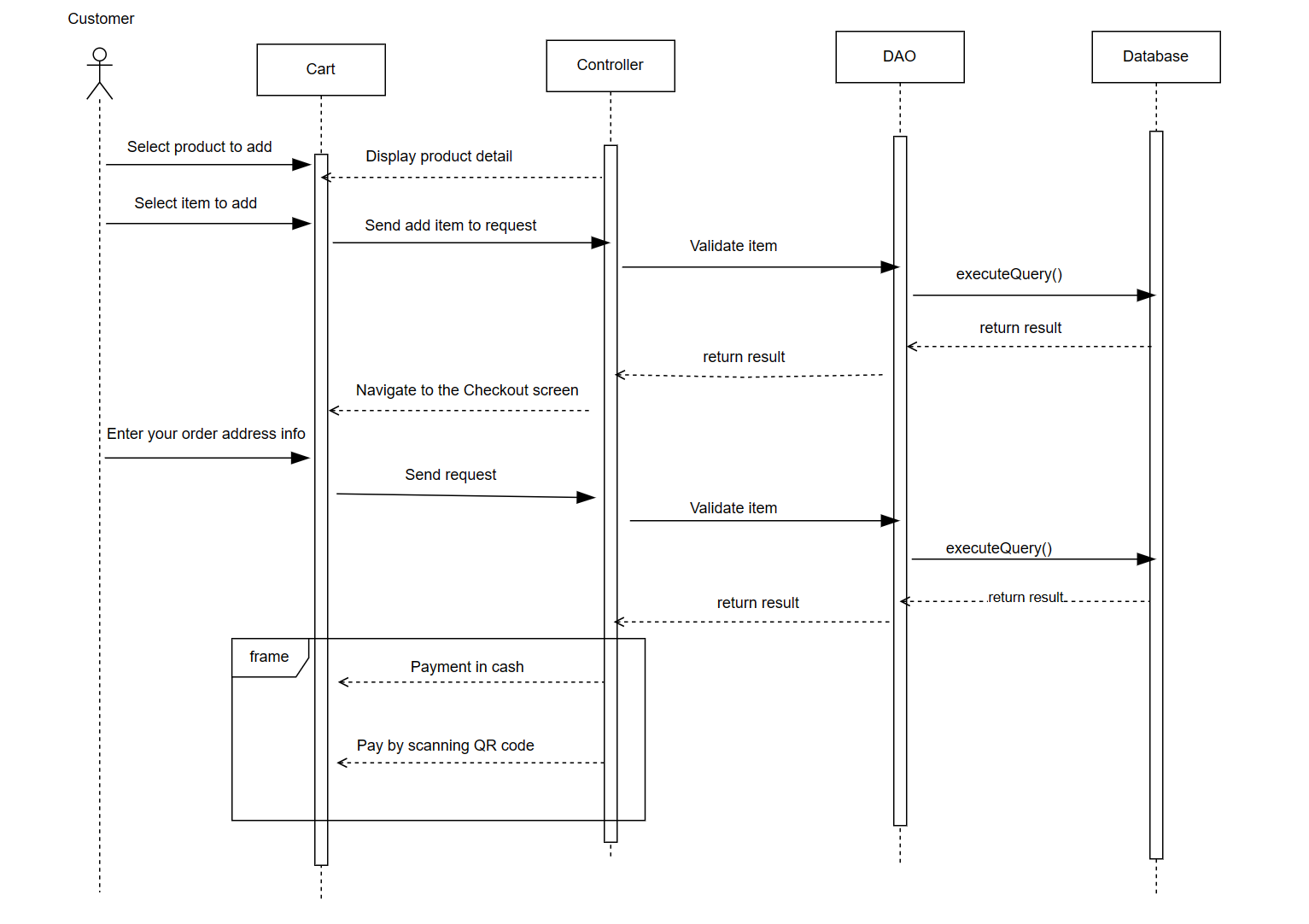
* Class ProductVariantDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getListProductVariant() | This method retrieves a list of product variants from the "productVariant" table based on various filters, search criteria, ordering, and pagination. The query initially selects all columns from the "productVariant" table and includes a where clause with the condition 1=1 to ensure subsequent filters are added correctly. |
| 02 | getOneProductVariantImage(int id, String colorId) | This method retrieves a single product variant image from the "ProductImage" table based on the provided product ID and color ID. The query selects the top 1 row from the "ProductImage" table and includes a where clause to filter based on the provided product ID (product\_id). |
| 03 | getTotalProductVariant() | This method retrieves the total number of product variations from the "productVariant" table based on multiple filters, search criteria, and filter types. It builds SQL query to filter based on ID colors. If there is only one color filter, it will add a simple condition (color\_id = filterColor.get(0)). If there are multiple color filters, it constructs a more complex condition than using algorithms "or" "and" to combine filters. |
| 04 | getColorNameById(int id) | This method retrieves the color name from the "color" table in the database based on the provided color ID. The SQL query string selects the "color" column from the "color" table, filtering the results based on the provided id. |
| 05 | getStorageSizeById(int id) | This method gets the memory size from the "Memory" table in the database based on the provided memory id. The SQL query string selects the "Storage\_Size" column from the "Storage" table in the "DURIAN\_SHOP" database, and filters the results based on the id provided. |
| 06 | getListColor() | This method takes a list of color objects from the "color" table in the database and returns them as a List<Color>. It executes SELECT statement to get all columns from "color" table. |
| 07 | getListStorage() | This method takes a list of storage objects from the "storage" table in the database and returns them as a List<Storage>. It executes SELECT statement to get all columns from "storage" table. |

* ClassVariantInformationDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getDetailInformation(int variantId | This method retrieves details about a product variant from the database. The SQL query string is generated to get specific columns from the "Product" and "ProductVariant" tables by combining them based on the Id column of the ProductVariant table. The query includes columns related to the product and its variant, such as name, screen, camera, RAM, etc., based on the variantId provided. |

#### ***Sequence diagram***



#### ***Database queries***

* Class OrderDao

| Database queries | Detail |
| --- | --- |
| getAllOrder() | select \* from [Order] where YEAR(Created\_Date) = 2023 |
| TotalPriceYear(int year) | select Total\_Price from [Order] where YEAR(Created\_Date) =" + year |
| TotalPriceMonth(int year, Month month) | select \* from [Order] where YEAR(Created\_Date) =" + year + "and MONTH(Created\_Date) = " + month |
| checkOrder(int userId) | select Id from [Order] where [status] = 1 and [User\_Id] = " + userId |
| createOrder(int User\_id, double Total\_Price) | Insert into [DURIAN\_SHOP].[dbo].[Order]([User\_Id], [Total\_Price], [Created\_Date],[Status]) values (?,?,(SELECT GETDATE()),1) |
| getOrderById(int User\_Id) | select \* from [Order] where [Order].[User\_Id] = " + User\_Id |

* Class OrderDetailDao

| Database queries | Detail |
| --- | --- |
| getHotProductID() | SELECT Product\_Id FROM OrderDetail GROUP BY Product\_Id\n"  + "HAVING SUM(Quantity) = ( SELECT MAX(TotalQuantity) FROM ( SELECT Product\_Id, SUM(Quantity) AS TotalQuantity FROM OrderDetail\n"  + "GROUP BY Product\_Id ) AS Subquery); |
| insertOrderDetail(int Order\_Id, int Product\_Id, int Quantity) | Insert into [DURIAN\_SHOP].[dbo].[OrderDetail]([Order\_Id], [Product\_Id], [Quantity]) values (?,?,?) |
| getAllOrderDetail(int orderId) | select \* from [OrderDetail] where [Order\_Id] = " + orderId; |
| updateQuantity(int detailId, int quantity) | UPDATE [OrderDetail] SET [Quantity] = ? WHERE Id = ? |
| deleteDetail(int detailId) | delete from [OrderDetail] where [Id] = ? |

* Class ProductVariantDao

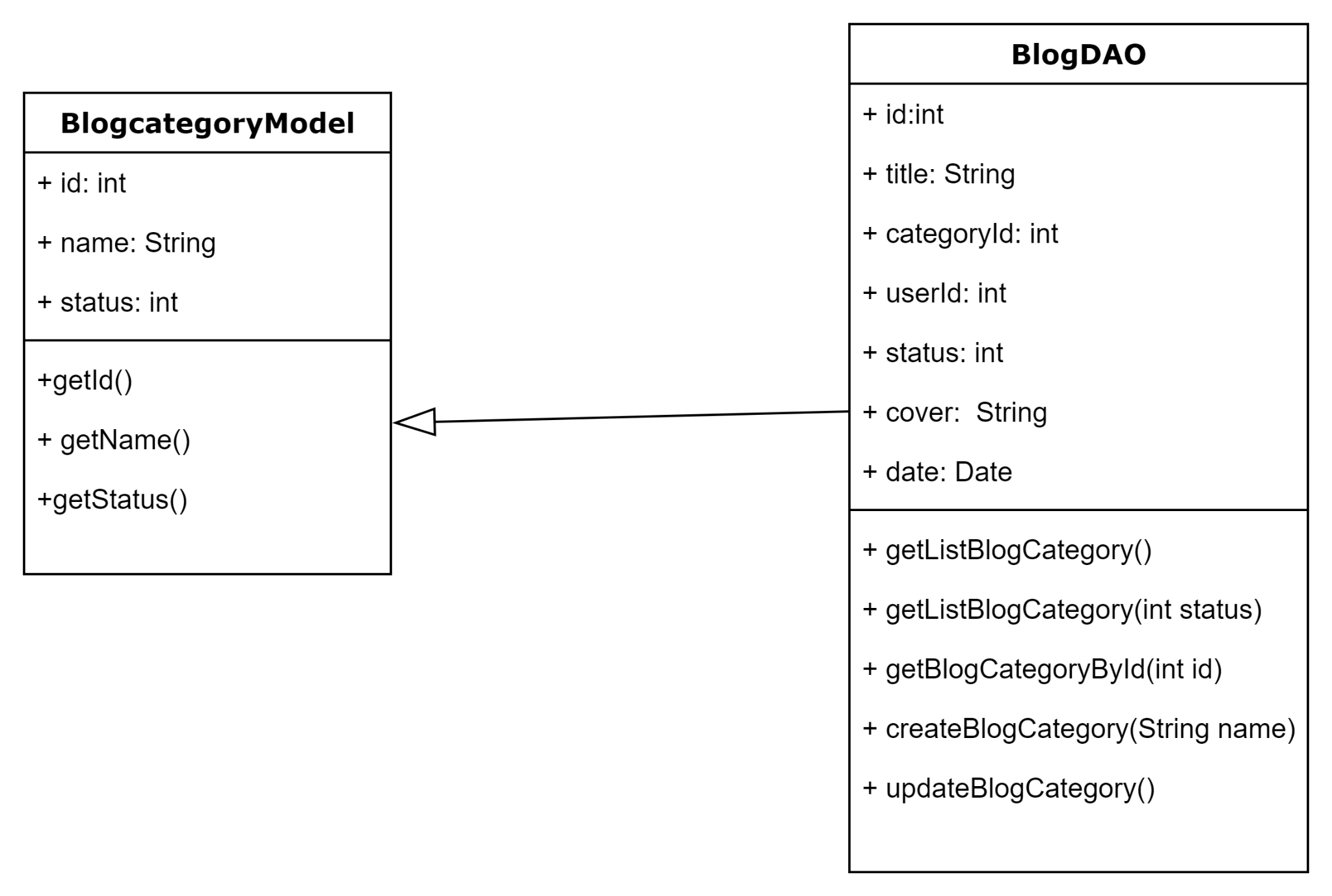
| Database queries | Detail |
| --- | --- |
| getListProductVariant() | select \* from productVariant"  + " where 1=1" |
| getOneProductVariantImage(int id, String colorId) | Select top(1) \* from ProductImage "  + " Where productImage.product\_id = ? |
| getTotalProductVariant() | select count(\*) from productVariant"  + " where 1=1 |
| getColorNameById(int id) | Select color from color "  + "Where id = ? |
| getStorageSizeById(int id) | Select [Storage\_Size] from [DURIAN\_SHOP].[dbo].[Storage] "  + "Where id = ? |
| getListColor() | SELECT \* FROM color |
| getListStorage() | SELECT \* FROM storage |

* Class ProductVariantInformationDao

| Database queries | Detail |
| --- | --- |
| getDetailInformation(int variantId) | select p.Id, p.[Name], p.[Screen], p.[Camera], p.Ram, p.Pin, p.Chipset, p.Screen\_resolution, p.img, v.Color\_Id, v.Storage\_Id, v.Quantity, v.Variant\_Price, v.[Status], v.Sale\_id\n" +  "from Product p join ProductVariant v on p.Id = v.Product\_Id where v.Id = " + variantId; |

### **Manage Blog Category (View Blog Category List, Update Blog Category, Add New Blog Category)**

#### Class diagram



#### Class specification

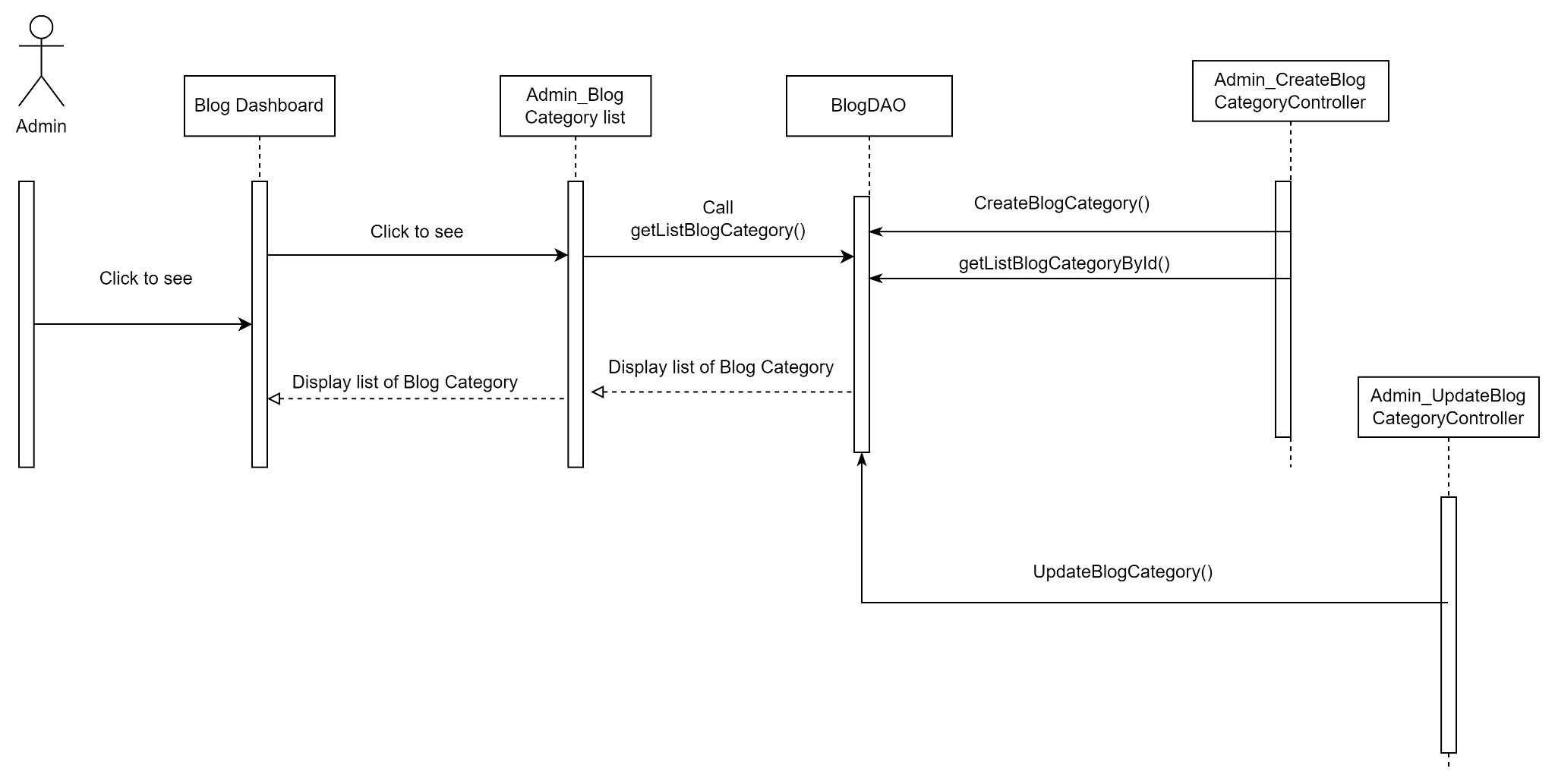
* BlogCategoryModel

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getId() | This method to get information about the BlogCategory's ID from the database. It executes an SQL query to fetch all BlogCategoryrecords and stores them in a list of BlogCategory objects. It returns the list of BlogCategories. |
| 02 | getName() | This method queries the "Blog Category" table in the database, then retrieves all the Names in the "Name" column of the table. |
| 03 | getStatus() | This method queries the "Blog Category" table in the database, then retrieves all Status of all Blog Category in the "Status" column of the table. |

* BlogDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | + getListBlogCategory() | This method retrieves all the BlogCategory from the database. It executes a SQL query to select all rows from the "BlogCategory" table. It then iterates over the result set and creates BlogCategoryModel objects for each row. The values from the result set are assigned to the corresponding properties of the BlogCategoryModel object. Finally, the method returns a list of BlogCategory objects containing all the BlogCategories. |
| 02 | + getBlogCategoryById(int id) | This method get all BlogCategory from database with condition of id. It executes a SQL query to select all rows from the "BlogCategory" table with the condition of id. It then iterates over the result set and creates BlogCategory objects for each row. The values from the result set are assigned to the corresponding properties of the BlogCategory object. Finally, the method returns a list of BlogCategory objects containing all theBlogCategories. |
| 03 | + createBlogCategory(String name) | This method is used to add a new BlogCategory to the database. It executes an SQL query to add a new BlogCategory to the "BlogCategory" table. Then it creates BlogCategory objects for each row. The values from the result set are assigned to the corresponding properties of the BlogCategory object. |
| 04 | + updateBlogCategory() | The updateBlogCategory method is responsible for updating the BlogCategory information in the system based on the input parameters. Search for the user with the ID identifier in the system and update their information |

#### Sequence diagram



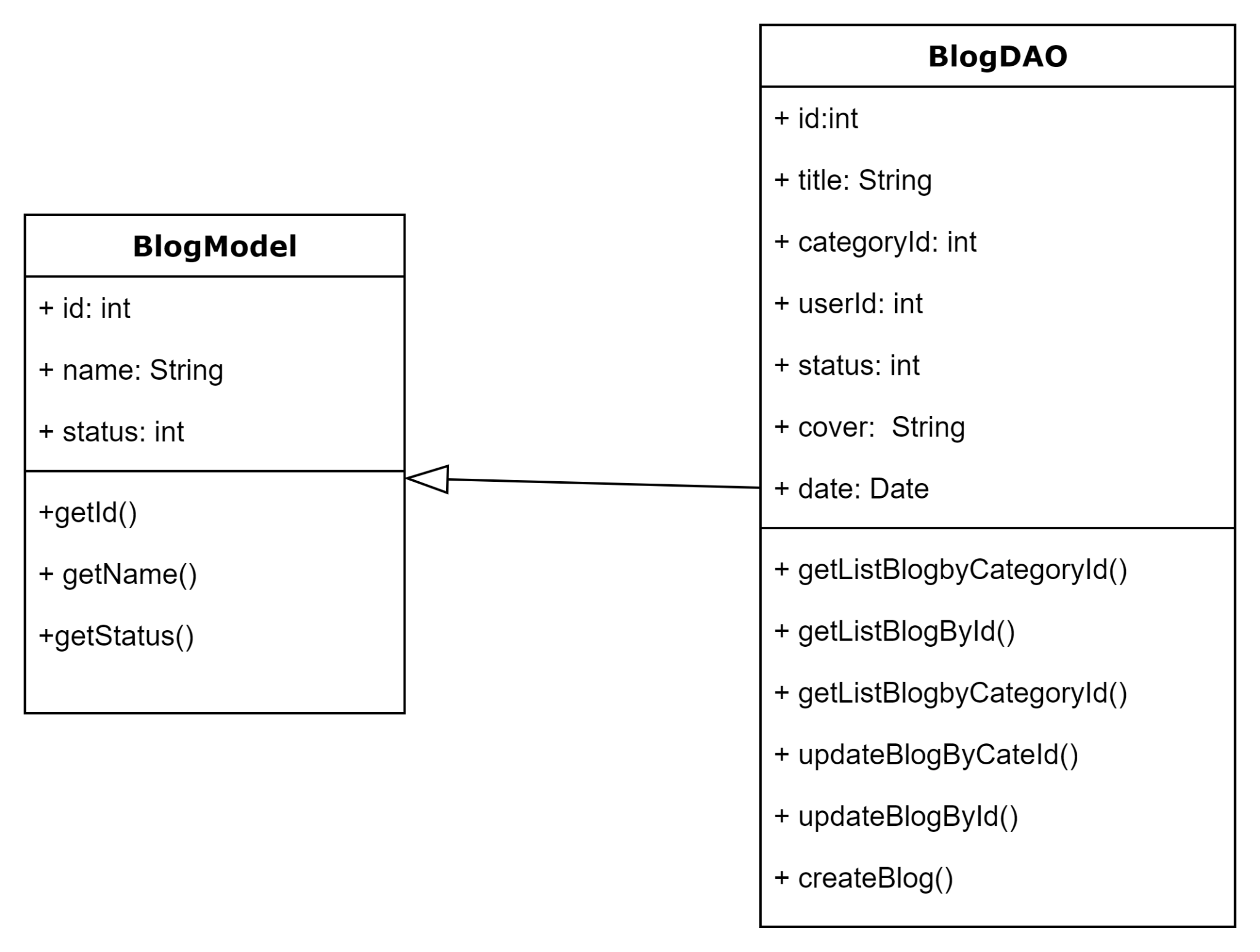
#### Database queries

* Class BlogDao

| Database queries | Detail |
| --- | --- |
| + getListBlogCategory() | String query = "Select \* from [DURIAN\_SHOP].[dbo].[BlogCategory]"; |
| + getBlogCategoryById(int id) | String query = "Select \* from [DURIAN\_SHOP].[dbo].[Blog] where categoryId = "+ id + " order by id desc"; |
| + createBlogCategory(String name) | String query = "Insert into [DURIAN\_SHOP].[dbo].[BlogCategory](categoryName, status) values (?, 1)"; |
| + updateBlogCategory() | String query = "UPDATE [BlogCategory] SET " +  "status = " + status;  if (!name.equals("")){  query+= ", categoryName = '" + name + "' ";  }  query += " where id = " + id; |

### **Manage Blog (View Blog List Of One Blog Category, Update Blog, Add New Blog, Hide Blog)**

#### Class diagram



#### Class specification

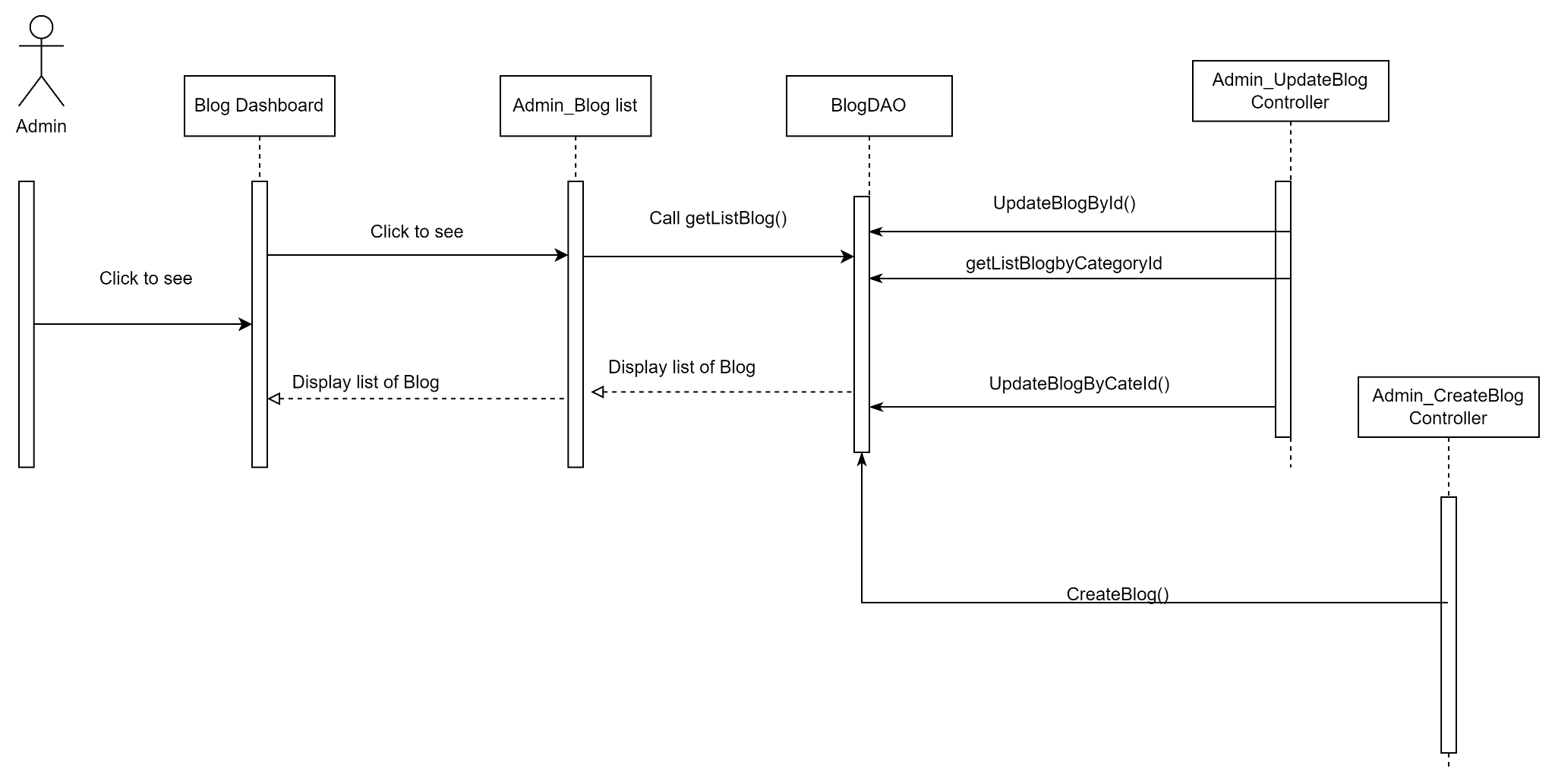
* BlogModel

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | +getId() | This method to get information about the Blog's ID from the database. It executes an SQL query to fetch all Blog records and stores them in a list of Blog objects. It returns the list of Blog. |
| 02 | + getName() | This method queries the "Blog" table in the database, then retrieves all the Names in the "Name" column of the table. |
| 03 | +getStatus() | This method queries the "Blog" table in the database, then retrieves all Status of all Blog in the "Status" column of the table. |

* BlogDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | + getListBlogbyCategoryId() | This method get all products from database with condition of Categoryid. It executes a SQL query to select all rows from the "BlogCategory" table with the condition of BlogCategory id. It then iterates over the result set and creates Blog objects for each row. The values from the result set are assigned to the corresponding properties of the Blog object. Finally, the method returns a list of Blog objects containing all the products. |
| 02 | + getListBlogById() | This method retrieves all the Blog from the database. It executes a SQL query to select all rows from the "Blog" table. It then iterates over the result set and creates BlogModel objects for each row. The values from the result set are assigned to the corresponding properties of the BlogModel object. Finally, the method returns a list of Blog objects containing all the Blogs. |
| 03 | + updateBlogByCateId() | The updateBlogByCateId method is responsible for retrieving BlogCategory information from the system based on the provided identifier (BlogCategoryid). The updateBlogByCateId() method will use the BlogCategoryid identifier to update the corresponding Blog information in the system and other information related to the Blog. |
| 04 | + updateBlogById() | The updateBlog method is responsible for updating the Blog information in the system based on the input parameters. Search for the user with the ID identifier in the system and update their information |
| 05 | + createBlog() | This method is used to add a new Blog to the database. It executes an SQL query to add a new Blog to the "Blog" table. Then it creates Blog objects for each row. The values from the result set are assigned to the corresponding properties of the Blog object. |

#### Sequence diagram



#### Database queries

* Class BlogDAO

| Database queries | Detail |
| --- | --- |
| + getListBlogbyCategoryId() | String query = "Select \* from [DURIAN\_SHOP].[dbo].[Blog] where categoryId = "+ id + " order by id desc"; |
| + getListBlogById() | String query = "Select \* from [DURIAN\_SHOP].[dbo].[Blog] where id = "+ id + " order by id desc"; |
| + updateBlogByCateId() | String query = "UPDATE [Blog] SET " +  "status = " + status;  if (!title.equals("")){  query+= ", title = '" + title + "' ";  }  if (catgoryId != 0){  query+= ", categoryId = " + catgoryId;  }  query += " where categoryId = " + id; |
| + updateBlogById() | String query = "UPDATE [Blog] SET " +  "status = " + status;  if (!title.equals("")){  query+= ", title = '" + title + "' ";  }  if (catgoryId != 0){  query+= ", categoryId = " + catgoryId;  }  if (!coverImg.equals("")){  query+= ", coverImg = '" + coverImg + "' ";  }  query += " where Id = " + id; |
| + createBlog() | String query = "Insert into [DURIAN\_SHOP].[dbo].[Blog](title, categoryId, userId, status, coverImg) values (?, ?, ?, 1, ?)"; |