

**Warehouse Output Manager**

**Software Design Specification**

**Team 4**

| **MSSV** | **Name** | **Account** |
| --- | --- | --- |
| QE170139 | Dinh Quoc Tien | TienDQ |
| QE170030 | Tran Minh Tuan | TuanTM |
| QE170154 | Nguyen Minh Tuan | TuanNM |
| QE170073 | Nguyen Tuan Hao | HaoNT |
| QE170233 | Nguyen Thanh Hau | HauNT |
| QE170162 | Diep Minh Hung | HungDM |
| QE170103 | Le Dong Tam | TamLD |

– QuyNhon, January 2024 –

Record of change

| Date | A\*  M, D | In charge | Change Description |
| --- | --- | --- | --- |
| 10/02/2024 | A | TienDQ | In the Create View of Note, when selecting the dropdown of |
| 01/03/2024 | M | TienDQ | Get the customer list directly in the database |
| 03/03/2024 | M | TienDQ | Get created name by database |
| 03/03/2024 | M | TienDQ | Change add phone by dropdown search |
| 04/03/2024 | M | TienDQ | Change not edit phone number |
| 05/03/2024 | A | TienDQ | Add Customer Rank |
| 13/03/2024 | A | TienDQ | Add download excel file |

\*A - Added M - Modified D - Deleted

**Table of Contents**

I. Overview.. 5

1. Code Packages. 5

2. Database Design. 6

a. Database Schema. 6

b. Table Description. 6

II. Code Designs. 8

1. Home Page 8

a. Class Diagram. 8

b. Class Specifications. 9

c. Sequence Diagram(s) 11

d. Database queries. 12

2. Warehouse Manager. 13

a. Class Diagram. 13

b. Class Specifications. 13

c. Sequence Diagram(s) 15

d. Database queries. 15

3. Manage Product 16

a. Class Diagram. 16

b. Class Specifications. 17

c. Sequence Diagram(s) 20

d. Database queries. 20

4. Manage Note Delivery Goods. 21

a. Class Diagram. 21

b. Class Specifications. 22

c. Sequence Diagram(s). 23

d. Database queries. 24

5 Manage Category. 24

a. Class Diagram.. 24

b. Class Specifications. 25

c. Sequence Diagram(s) 26

d. Database queries. 26

5 Manage Customer. 27

a. Class Diagram. 27

b. Class Specifications. 27

c. Sequence Diagram(s) 29

d. Database queries. 29

6 Login/Register

a. Class Diagram.

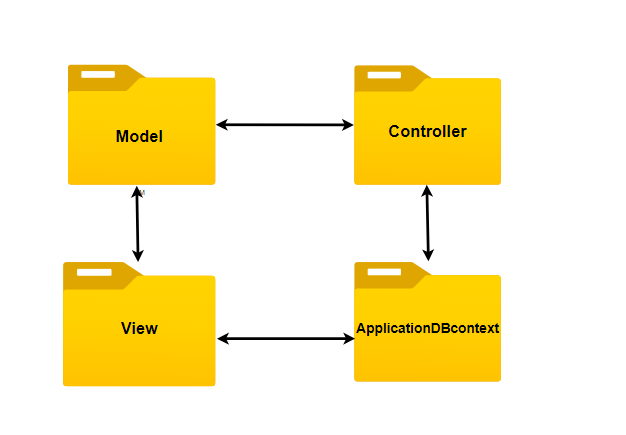
b. Class Specifications.

c. Sequence Diagram(s)

d. Database queries.

# **I. Overview**

## **1. Code Packages**

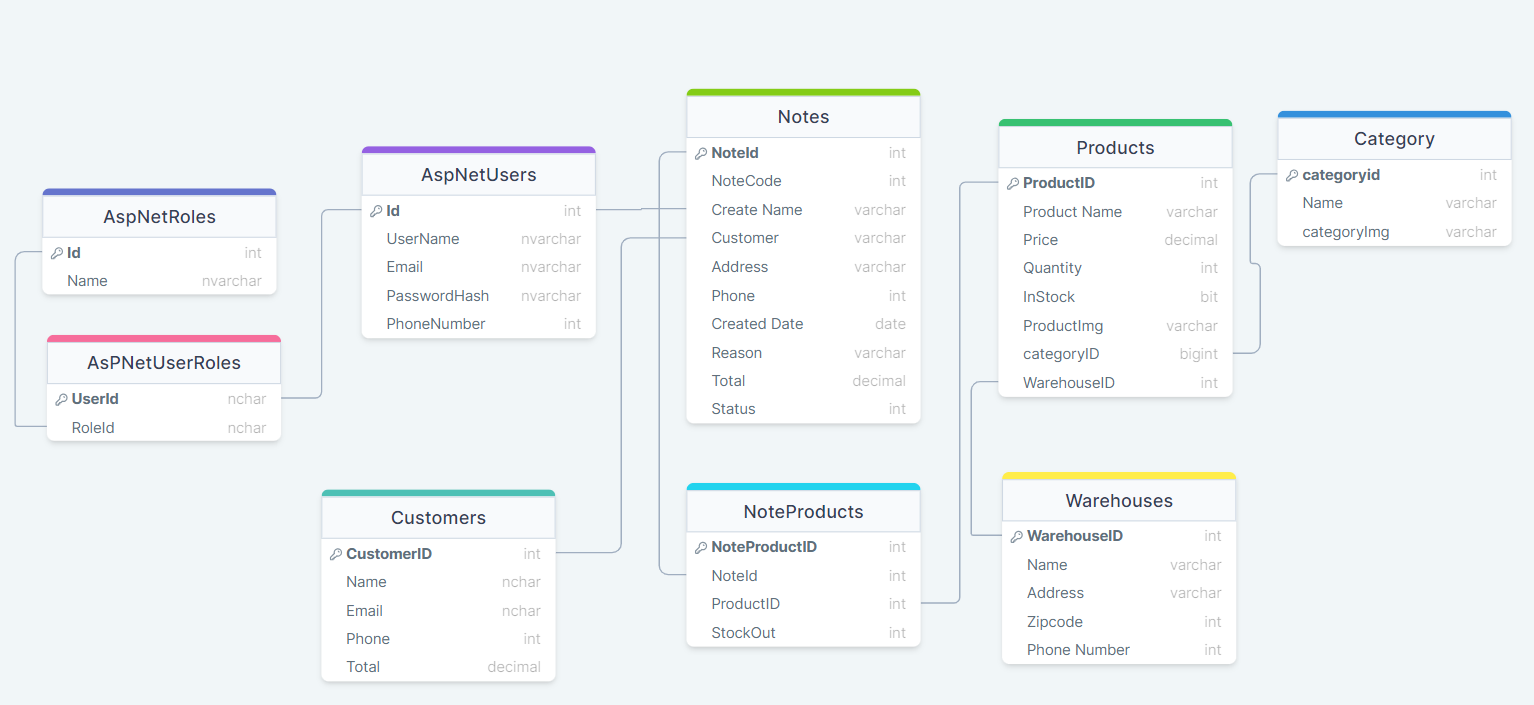


***Package descriptions***

| *No* | *Package* | *Description* |
| --- | --- | --- |
| *01* | *Model* | *Models represent the data and business logic in the application. They define the data structure of the product catalog, including the properties of that table* |
| *02* | *View* | *View is the user interface, where data is displayed to the end user. In a product catalog management web application, views can display a list of categories or detailed information about a specific category.* |
| *03* | *Controller* | *The controller is where the application logic is handled and directs the data flow between the model and view. In a product catalog management application, the controller may be responsible for retrieving data from the model and passing it to the view to display to the user.* |
| *04* | *ApplicationDbContext* | *ApplicationDbContext is a class created to manage connections to the database and perform data query operations. In ASP.NET MVC, this class is often used to interact with Entity Framework, helping to perform CRUD (Create, Read, Update, Delete) operations on the database.* |

## **2. Database Design**

### **a. Database Schema**



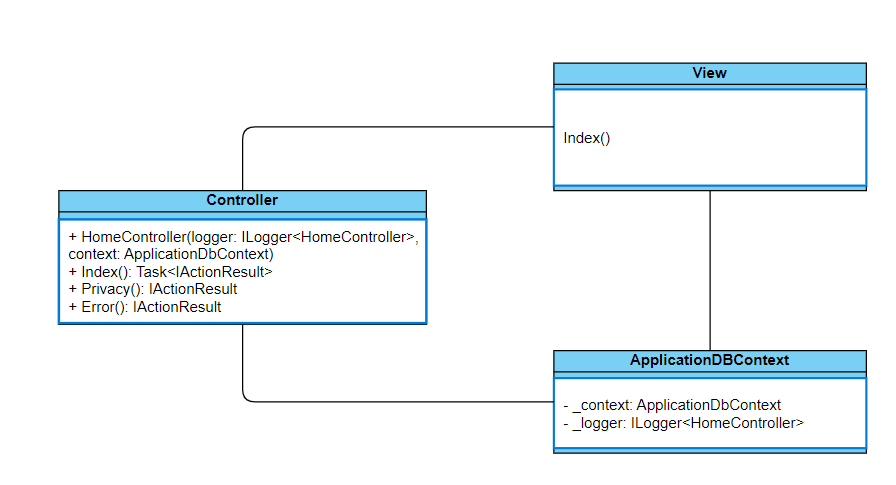
### **b. Table Description**

| **No** | **Table** | **Description** |
| --- | --- | --- |
| *01* | *AsPNetUserRoles* | *This table has a clustered primary key on these two columns. Additionally, there is a foreign key constraint between RoleId and the Id column of the AspNetRoles table.* |
| *02* | *Warehouse* | *This table has a clustered primary key on the WarehouseID column. This means that the data in the table will be sorted by the order of WarehouseID to optimize query performance.* |
| *03* | *Notes* | This table has a clustered primary key on the NoteId column. This means that the data in the table will be sorted by the order of NoteId to optimize query performance. |
| *04* | *Products* | This table has a clustered primary key on the ProductID column. This means that the data in the table will be sorted by the order of ProductID to optimize query performance. |
| *05* | *AspNetRoles* | This table has a clustered primary key on the Id column. This means that the data in the table will be sorted by the order of Id to optimize query performance. |
| *06* | *NoteProducts* | *This table has a clustered primary key on the NoteProductId column. This means that the data in the table will be sorted by the order of NoteProductId to optimize query performance.*   * *A constraint between the NoteId column of the NoteProducts table and the NoteId column of the Notes table.* * *A constraint between the ProductID column of the NoteProducts table and the ProductID column of the Products table.* |
| *07* | *Customers* | *This table has a clustered primary key on the CustomerID column. This means that the data in the table will be sorted by the order of CustomerID to optimize query performance.* |
| *08* | *Category* | *This table has a clustered primary key on the categoryId column. This means that the data in the table will be sorted by the order of categoryId to optimize query performance.* |
| *09* | *AspNetUsers* | *This table has a clustered primary key on the Id column. This means that the data in the table will be sorted by the order of Id to optimize query performance.* |

# **II. Code Designs**

## **1. Home Page**

### **a. Class Diagram**



### **b. Class Specifications**

*Home Controller*

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *AspNetRole* | These properties are fundamental components of the ASP.NET Identity system, which is responsible for managing user authentication and authorization within web applications. |
| *02* | *AspNetUser* | *Manage user information in the system with the help of AspNet* |
| *03* | doPost(HttpServletRequest, HttpServletResponse): void | *Call itself* |
| *04* | doGet(HttpServletRequest, HttpServletResponse): void | req.getRequestDispatcher |

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *03* | *AspNetUserRole* | *Manage user roles in the system, each user logged in with a different role will have different actions in the system* |
| *04* | *HomeController* | *Includes warehouse management attributes: product, warehouse, delivery note based on user role* |
| *05* | *ProductController* | *includes the attributes of a product: ID, name, price, quantity, code of a product, managed and added, deleted, edited by users with the Keeper role* |
| *06* | *WarehouseController* | *Manage warehouses in the system, or can add, delete, and edit warehouses available in the system.* |
| *07* | *NoteController* | *Manage warehouse release notes, delivery notes will be created by the warehouse keeper and approved by the accountant and director to proceed with the release.* |

#### **AccountDBContext**

***Class Methods***

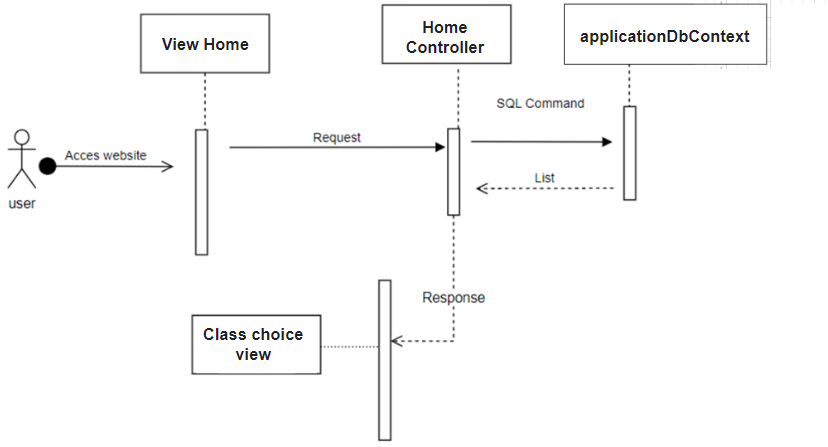
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updatePass(String username, String password) | Update password |
| 02 | insertRole(int role\_id, String username) | Insert role for account |
| 03 | updateUserRole(String username) | Update role |
| 04 | login(String username, String password) | User login |
| 05 | signup(String username, String password) | Sign up username and password |
| 06 | updateUserRole(String role) | Update role for user |

#### **DBContext**

***Class Methods***

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | DBContext() | To connect with database |
| 02 | main(String[]) | Call DBContext() |

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



### **d. Database Queries**

**Update Password User:** UPDATE [AspNetUsers]  
 SET [PasswordHash] = 'new\_password\_ WHERE [UserName] = 'your\_username';

**Insert Role User:** INSERT INTO [AspNetRoles] ([Id], [Name], [NormalizedName], [ConcurrencyStamp])  
 VALUES ('new\_role\_id', 'new\_role', 'NEW\_ROLE', 'new\_concurrency\_stamp');

**Update User Role:** UPDATE [AspNetUserRoles]  
 SET [RoleId] = 'new\_role\_id'   
 WHERE [UserId] = (SELECT [Id] FROM [AspNetUsers] WHERE [UserName] = 'your\_username');

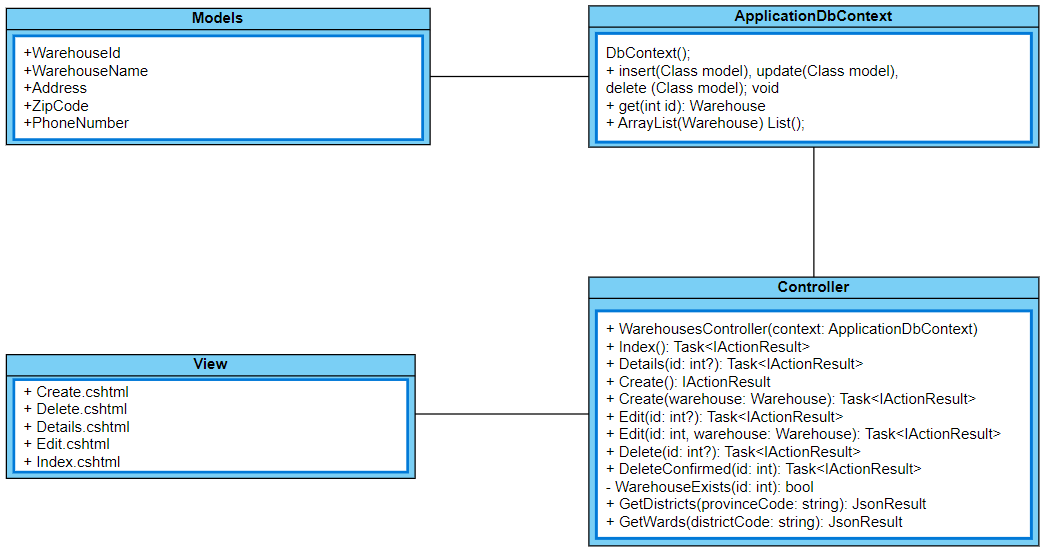
**Login (check login):** SELECT \*  
 FROM [AspNetUsers]  
 WHERE [UserName] = 'your\_username' AND [PasswordHash] = 'provided\_password\_hash';

**Signup (addUser):** INSERT INTO [AspNetUsers] ([Id], [UserName], [NormalizedUserName], [Email], [NormalizedEmail], [EmailConfirmed], [PasswordHash], [SecurityStamp], [ConcurrencyStamp], [PhoneNumber], [PhoneNumberConfirmed], [TwoFactorEnabled], [LockoutEnd], [LockoutEnabled], [AccessFailedCount])  
 VALUES ('new\_user\_id', 'new\_username', 'NEW\_USERNAME', 'new\_email@example.com', 'NEW\_EMAIL@EXAMPLE.COM', 1, 'new\_password\_hash', 'new\_security\_stamp', 'new\_concurrency\_stamp', NULL, 0, 0, NULL, 1, 0);

**Update User Role (Cập nhật vai trò của người dùng):** UPDATE [AspNetUserRoles]  
 SET [RoleId] = 'new\_role\_id'   
 WHERE [UserId] = (SELECT [Id] FROM [AspNetUsers] WHERE [UserName] = 'your\_username');

## **2. Manage Warehouse**

a. Class Diagram:



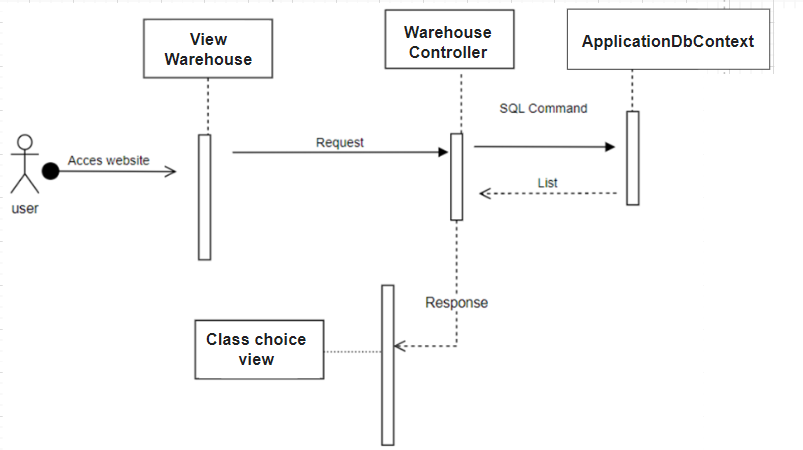
b. Class specification

Class method

- Home Controller:

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | doPost(HttpServletRequest, HttpServletResponse): void | Call itself |
| 02 | doGet(HttpServletRequest, HttpServletResponse): void | Get list classes and req.getRequestDispatcher("./warehouse.jsp").forward(req, resp); |
| 03 | public async Task<IActionResult> Index() | This method is likely responsible for handling requests to the index page of the controller. It might asynchronously perform some operations, such as fetching data from a database or calling other services, and then return an **IActionResult** to render the appropriate view or respond to the request in some other way. |
| 04 | public IActionResult Create() | This is the name of the action method. In this context, it likely refers to the action responsible for creating a new warehouse or handling the creation process, creation of a new warehouse. It expects to receive data related to the warehouse, including its ID, name, address, zip code, and phone number. |
| 05 | Task<IActionResult> Create([Bind("WarehouseId,WarehouseName,  Address,ZipCode,PhoneNumber")] | In ASP.NET Core, actions often return an **IActionResult** to represent the result of processing a request. This is a data annotation used for model binding. It specifies the properties that should be bound from the request data to the model object. In this case, it indicates that the **WarehouseId**, **WarehouseName**, **Address**, **ZipCode**, and **PhoneNumber** properties are allowed to be bound. Other properties of the model object will not be bound, providing protection against over-posting attacks. |
| 06 | Task<IActionResult> Edit(int? id) | This method signature indicates that the **Edit** action method accepts an optional **id** parameter, allowing clients to specify which resource they want to edit. The method would typically use this ID to retrieve the corresponding resource from the data store and then return a view or another **IActionResult** that allows users to edit the resource. |
| 07 | Task<IActionResult> Delete(int? id): doGet | This method prepares a Warehoues for deletion by displaying its details in a view, allowing the user to confirm deletion. |
| 08 | Task<IActionResult> DeleteConfirmed(int id) | This method permanently deletes a Warehouse from the database after the user confirms deletion. |

c. Sequence Diagram



d. Database queries

- Get all Warehouse:

SELECT \* FROM Warehouses

- Get details of Warehouse:

SELECT \* FROM Warehouses WHERE WarehouseId = @id

- Create new Warehouse:

INSERT INTO Warehouses (WarehouseId, WarehouseName, Address)

VALUES (@WarehouseId, @WarehouseName, @Address)

- Edit Warehouse:

UPDATE Warehouses

SET WarehouseName = @WarehouseName, Address = @Address

WHERE WarehouseId = @id

- Delete Warehouse:

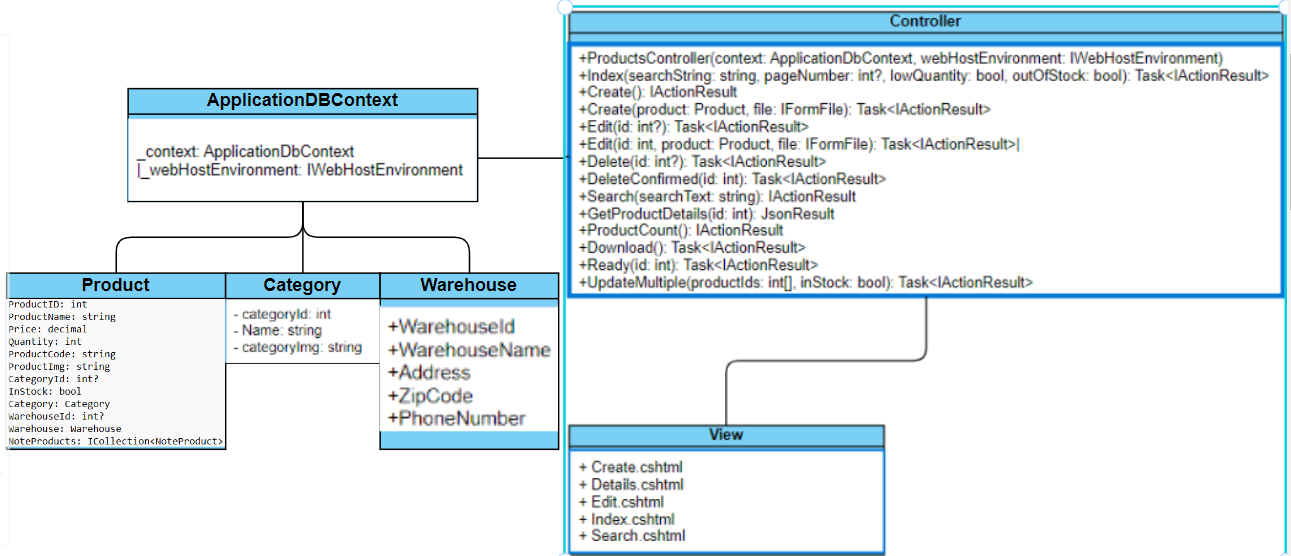
DELETE FROM Warehouses WHERE WarehouseId = @id

- Check Warehouse exist:

SELECT COUNT(\*) FROM Warehouses WHERE WarehouseId = @id

## **3. Manage Product**

a. Class diagram



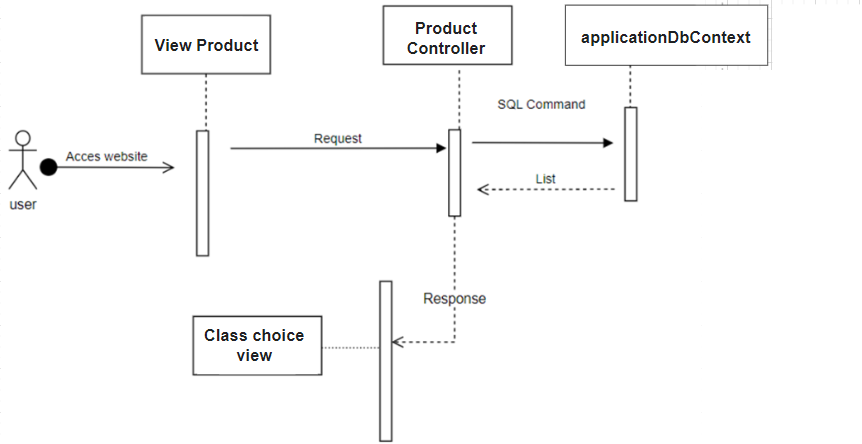
b. Class specification

Class method

- Product Controller:

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | IActionResult Index() | This code snippet defines an Index action method within a ProductsController class (assuming the class name is ProductsController). This action method is responsible for retrieving and displaying product data in a View |
| 02 | IActionResult Create(): doGet | This code snippet defines a Create action method within a controller (the name of the controller is not provided, but for clarity, let's assume it's ProductsController). |
| 03 | Task<IActionResult> Create([Bind("ProductID,ProductName,Price,Quantity,ProductCode,WarehouseId")] Product product): doPost | This code snippet defines a Create action method within a controller (again, the name isn't provided but for clarity let's assume it's ProductsController). This action method handles the form submission after a user tries to create a new product |
| 04 | Task<IActionResult> Edit(int? id): doGet | This code defines the Edit action method within a controller (assuming it's ProductsController). This action method handles the initial request to edit a specific product |
| 05 | Task<IActionResult> Edit(int id, [Bind("ProductID,ProductName,Price,Quantity,ProductCode,WarehouseId")] Product product):doPost | This code defines the Edit action method, likely within the ProductsController, handling the actual editing of a product in an ASP.NET MVC application. |
| 06 | Task<IActionResult> Details(int? id): doGet | This code defines a Details action method within a controller, likely ProductsController, handling displaying detailed information about a specific product |
| 07 | Task<IActionResult> Delete(int? id): doGet | This code snippet defines the Delete action method, likely within the ProductsController, responsible for handling the initial request to delete a product |
| 08 | Task<IActionResult> DeleteConfirmed(int id): doPost | This code handles the final deletion of a product from a database after the user has confirmed their intention. It's likely part of a larger product management system |
| 09 | ProductExists(int id) | Check that products exist in system |

c. Sequence diagram



d. Database queries

**Abstract method insert(Class model):**

INSERT INTO [dbo].[Classes] ([ClassName], [ProductID])

VALUES ('new\_class\_name', 'product\_id');

**Abstract method update(Class model):**

UPDATE [dbo].[Classes]

SET [ClassName] = 'updated\_class\_name'

WHERE [ClassID] = 'class\_id';

**Abstract method delete(Class model):**

DELETE FROM [dbo].[Classes]

WHERE [ClassID] = 'class\_id';

**Method list():**

SELECT \*

FROM [dbo].[Products];

**Method listClassbyProductId(int class\_id):**

SELECT \*

FROM [dbo].[Classes]

WHERE [ProductID] = 'product\_id';

**Method get(int product\_id):**

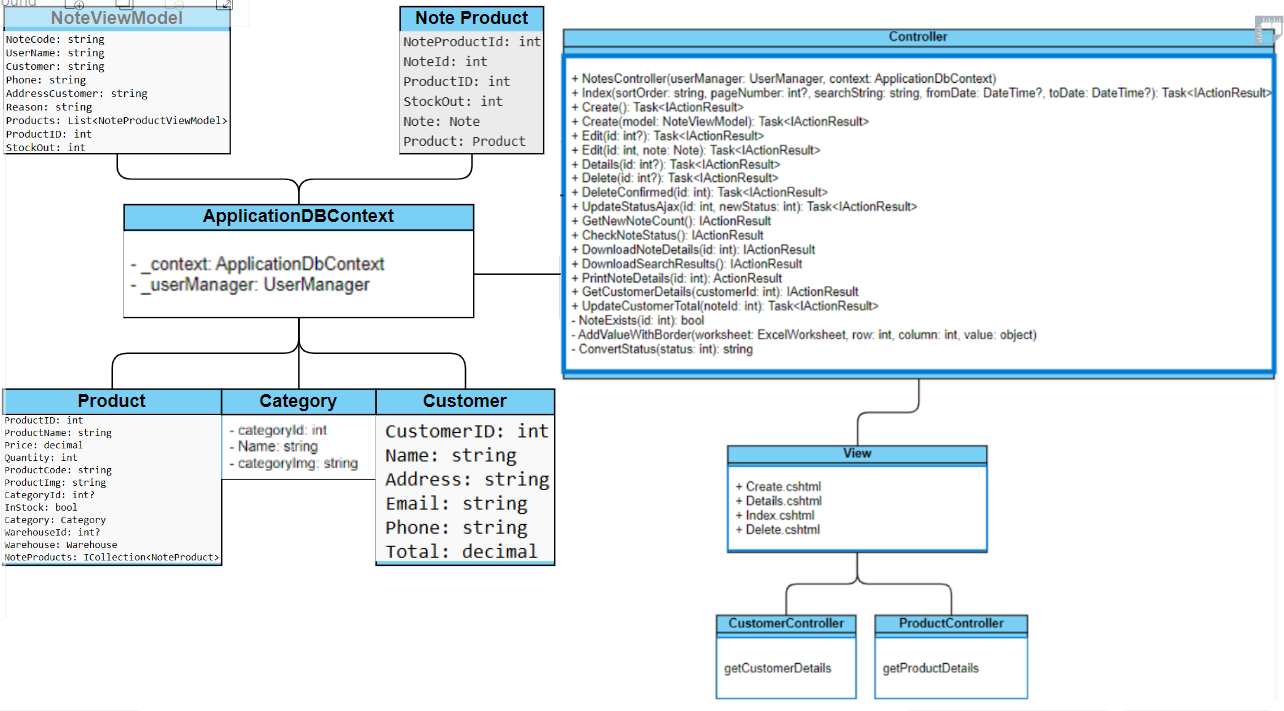
SELECT \*

FROM [dbo].[Products]

WHERE [ProductID] = 'product\_id';

**4. Manage Note Delivery Goods**

a. Class Diagram

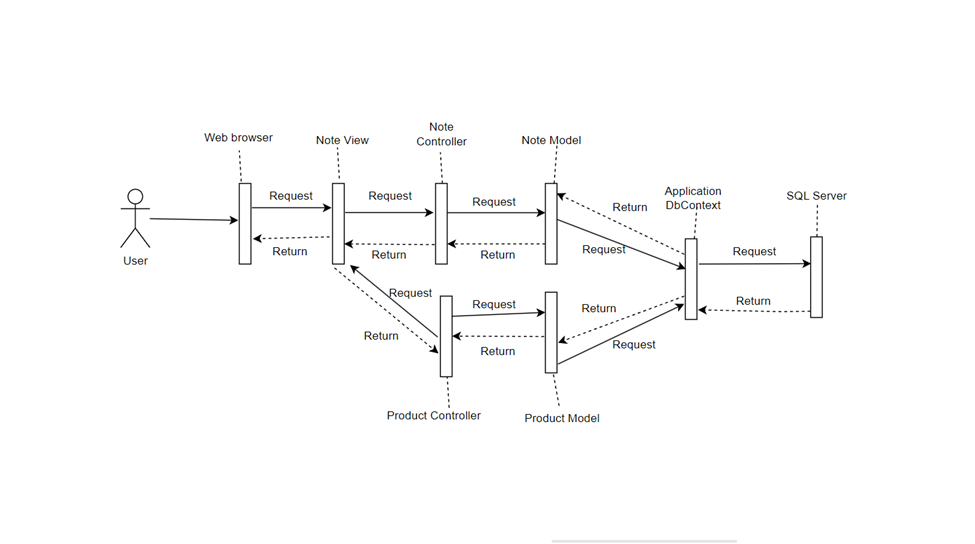


b. Class specification

Note Controller

| - **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | ActionResult Index( ) | **Gets all notes and shows them in a view.** |
| 02 | IActionResult Create():  do Get | Prepares the necessary data and renders a view for creating a Note Delivery Goods |
| 03 | IActionResult Create():  do Post | Creates a new note and associates it with selected products and their stock out, update default status of note is 1 . |
| 04 | Task<IActionResult> Details(int? id) | This method asynchronously retrieves a specific Note from the database and displays its details in a view. |
| 05 | Task<IActionResult> Delete(int? id): do Get | This method prepares a note for deletion by displaying its details in a view, allowing the user to confirm deletion. |
| 06 | Task<IActionResult> DeleteConfirmed(int id): do Post | This method permanently deletes a note from the database after the user confirms deletion. |
| 07 | Not Exists(int id) | This method checks if a note with the given id exists in the database. |
| 08 | IActionResult GetNewNoteCount() | This method retrieves the count of notes with a specific status (assumed to be "new") and returns it as a JSON response. |
| 09 | IActionResult CheckNoteStatus() | This method efficiently checks if there are any notes in the database with a specific status, returning a boolean response as JSON. |

c. Sequential diagram:



d. Database queries:

- Get all Note Delivery Goods

SELECT \* FROM Notes WHERE NoteId = @NoteId;

- Create new Note:

INSERT INTO Notes (NoteCode, CreateName, Customer, AddressCustomer, Reason, Status)

VALUES (@NoteCode, @CreateName, @Customer, @AddressCustomer, @Reason, @Status);

- Delete Note by Id:

DELETE FROM Notes WHERE NoteId = @NoteId;

- Update status of Note:

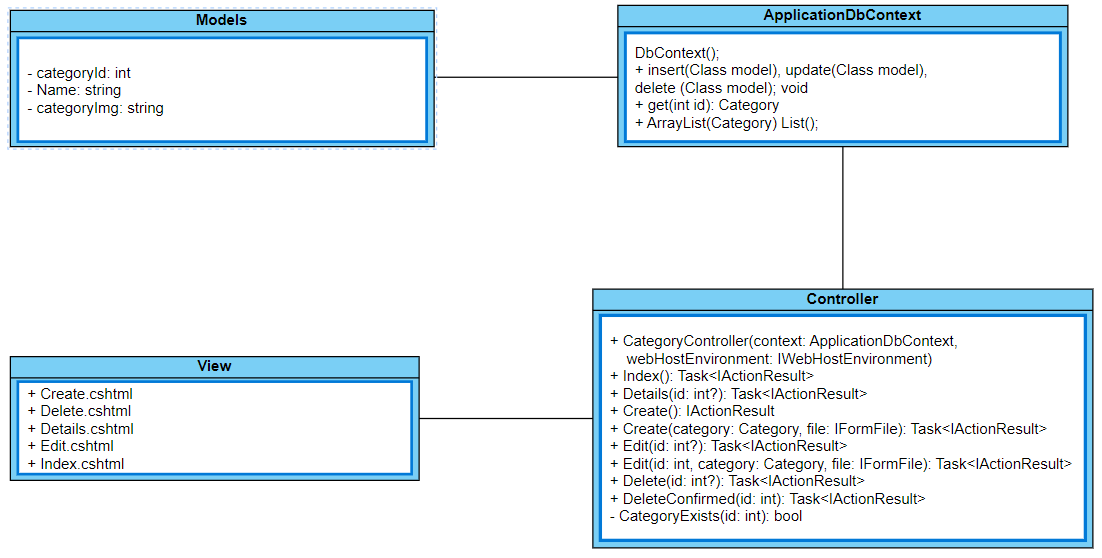
UPDATE Notes

SET Status = @NewStatus

WHERE NoteId = @NoteId;

## **5. Manage Category**

a. Class Diagram:



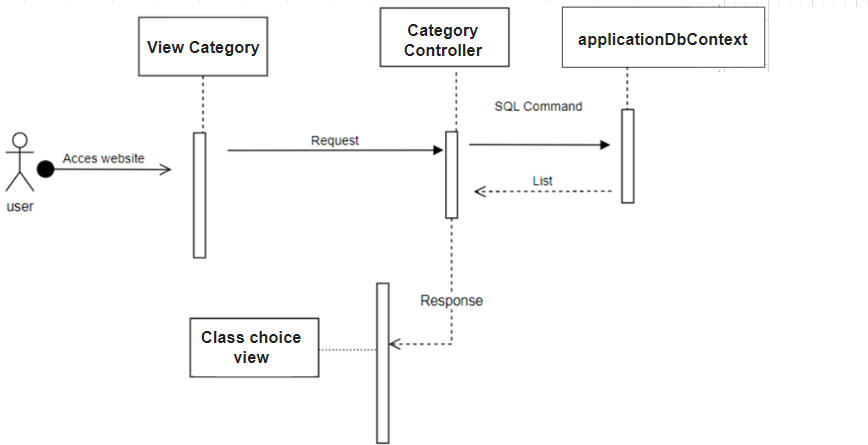
b. Class specification

Class method

- Category Controller:

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | public async Task<IActionResult> Edit(int? id) | The method signature public async Task<IActionResult> Edit(int? id) declares an action method named Edit within a controller class, likely in an ASP.NET Core application. |
| 02 | public async Task<IActionResult> Delete(int? id) | This method prepares a note for deletion by displaying its details in a view, allowing the user to confirm deletion. |
| 03 | public async Task<IActionResult> DeleteConfirmed(int id) | The method **public async Task<IActionResult> DeleteConfirmed(int id)** is an action method in an ASP.NET Core controller. It is used to confirm the deletion of a specific item based on a given ID. |
| 04 | public async Task<IActionResult> Create([Bind("CategoryId,Name")] Category category, [FromForm(Name = "file")] Microsoft.AspNetCore.Http.IFormFile file) | The method **public async Task<IActionResult> Create([Bind("CategoryId,Name")] Category category, [FromForm(Name = "file")] Microsoft.AspNetCore.Http.IFormFile file)** is an action method typically found in an ASP.NET Core controller. |
| 05 | public async Task<IActionResult> Edit(int id, [Bind("categoryId,Name,categoryImg")] Category category, [FromForm(Name = "file")] IFormFile file) | **public async Task<IActionResult> Edit(int id, ...)** - This is a public method named **Edit** that returns a **Task** of **IActionResult**. It's likely an asynchronous method (**async**), as indicated by the **async** keyword, which suggests it might perform I/O-bound operations. |

c. Sequence Diagram



d. Database queries.

Create: SELECT \* FROM Category WHERE categoryId = @id;

Detail: SELECT \*

FROM Category

WHERE categoryId = @categoryId;

Edit: DECLARE @categoryId INT

DECLARE @categoryName NVARCHAR(255)

DECLARE @categoryImg NVARCHAR(255)

UPDATE Category

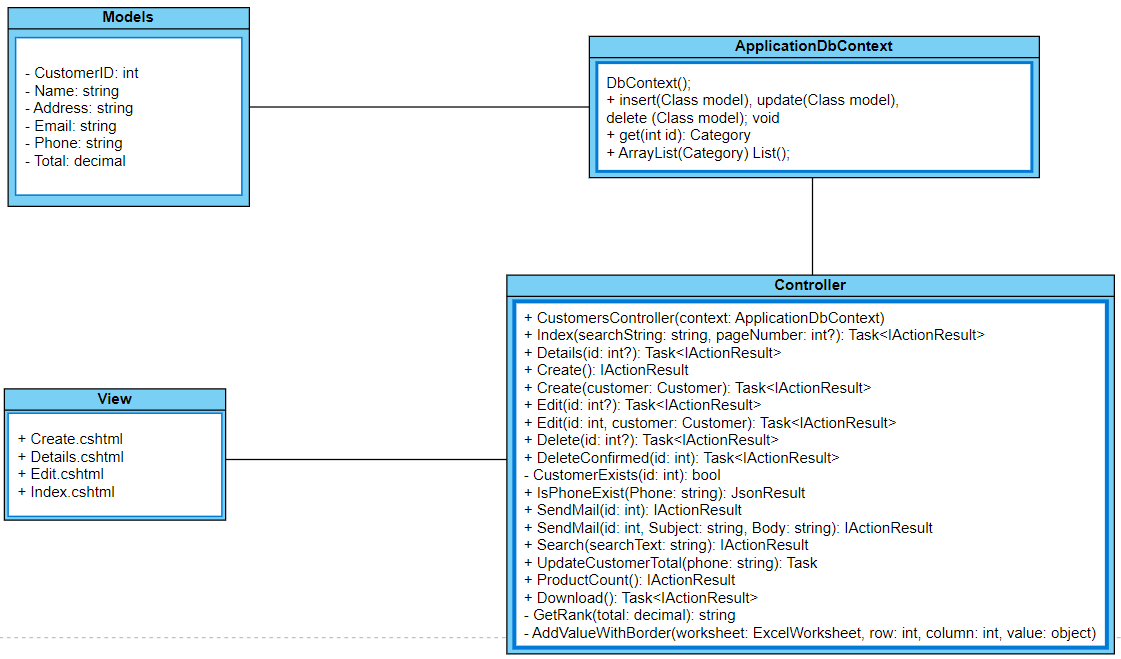
SET Name = @categoryName,

categoryImg = @categoryImg

WHERE categoryId = @categoryId;

## **6. Manage Customer**

a. Class Diagram:



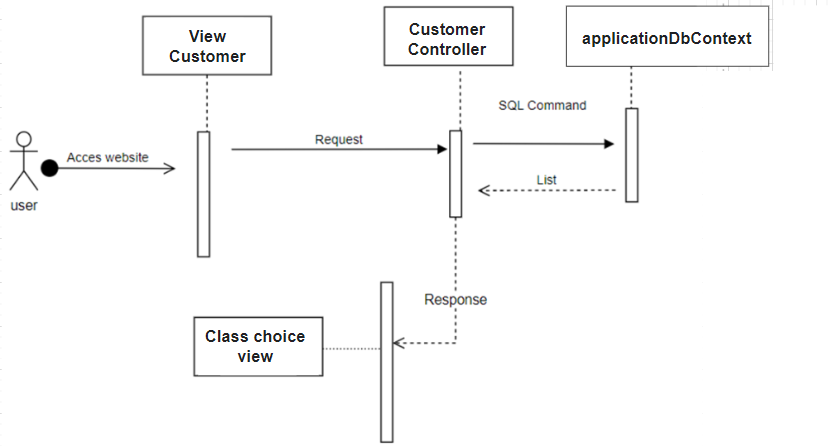
b. Class specification

Class method

- Home Controller:

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 1 | public async Task<IActionResult> Index | Gets all customer and shows them in a view. |
| 2 | I ActionResult Create():do Get | Prepares the necessary data and renders a view for creating a Customer |
| 3 | Task<IActionResult> Edit(int? id) | This method prepares a customer for edition by displaying its details in a view, allowing the user to confirm edit. |
| 4 | Task<IActionResult> Delete(int? id) :doGet | This method prepares a Customer for deletion by displaying its details in a view, allowing the user to confirm deletion. |
| 5 | Task<IActionResult> DeleteConfirmed(int id): doPost | This method permanently deletes a Customer from the database after the user confirms deletion. |
| 6 | public ActionResult SendMail(int id, string Subject, string Body) | this method seems to be designed to send an email with a specified subject and body to a recipient identified by the **id** parameter. However, it lacks implementation details such as the email address of the recipient, SMTP server configuration, and the actual logic for sending the email. These details would need to be included within the method or its surrounding context for it to function properly. |

c. Sequence Diagram



d. Database queries

Send mail: SELECT \* FROM Customers WHERE Id = @id;

Delete: SELECT \* FROM Customers WHERE CustomerID = @id;

UPDATE Customers

SET Name = @Name, Address = @Address, Email = @Email, Phone = @Phone

WHERE CustomerID = @CustomerID;

DECLARE @OldPhone NVARCHAR(255)

SELECT @OldPhone = Phone FROM Customers WHERE CustomerID = @CustomerID;

UPDATE Notes

SET Customer = (SELECT Name FROM Customers WHERE CustomerID = @CustomerID),

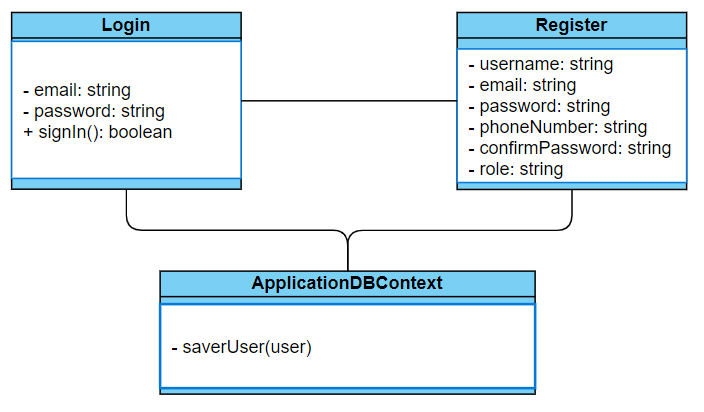
Phone = (SELECT Phone FROM Customers WHERE CustomerID = @CustomerID),

AddressCustomer = (SELECT Address FROM Customers WHERE CustomerID = @CustomerID)

WHERE Phone = @OldPhone;

## **7. Login/Register**

a. Class Diagram.



c. Sequence Diagram(s)

d. Database queries.

b. Class Specifications:

Login Attributes:

* bemail: string - The email address of the user.
* password: string - The password of the user.

Methods:

* signIn(): boolean - Attempts to sign in the user with the provided credentials. Returns true if successful, false otherwise.

Register Attributes:

* username: string - The desired username of the user.
* email: string - The email address of the user.
* password: string - The password of the user.
* phoneNumber: string - The phone number of the user.
* confirmPassword: string - The confirmation of the password.
* role: string - The role of the user. Methods: None specified.

c. Sequence Diagram(s) :

d. Database queries:

Login

* Query to check if the user exists with the provided email.
* Query to validate the password against the stored hash.

Register

* Query to insert a new user into the database with provided details.