|  |  |
| --- | --- |
| logo (CMYK)-01 | MINISTRY OF EDUCATION & TRAINING  **HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY** |

**Course Project**

**[CMP184] – Systems Analysis And Design**

Major: **INFORMATION TECHNOLOGY**

**Supervisor:** Mr.Ngô Thanh Hùng

**Student’s name :**

|  |  |  |
| --- | --- | --- |
| Dương Anh Minh | ID: 2280601913 | Class: 22DTHQA2 |
| Đào Phước Hải | ID: | Class: 22DTHQA2 |

*Ho Chi Minh City, 2025*

Dev: Hi, thanks for taking the time to meet with me today. To start, can you share with me the general idea of ​​the website you want to build?

Customer: Hi. I just think that I need a website... an "online store" to sell my software keys. How can people come in to see the products and then buy them?

Dev: I understand. "Online store" is a great place to start. So let me try to visualize it with you. When customers visit, they will see a main page that displays all the products, right?

Customer: That's right! There should be pictures, product names, and prices for people to see. Oh, sometimes I have many types, if I can categorize them, for example "Windows keys", "Office keys", that would be great.

Dev: Great. So we have two important concepts: Product and Category. For customers to buy, do they need an account? Or can they buy without registering?

Customer: Well... I think we need an account. So that later I can know who bought my products. So there must be a place for people to register and log in.

Dev: That makes sense. When we have an account, we will have another object called User. After logging in, users will be able to review the history of the transactions they have made, right?

Customer: Yes! That is necessary. Sometimes they forget the key they bought, they need a place to review it. And... I want a shopping cart too. Like on Shopee, people can put many items in the cart and pay at once for convenience.

Dev: Yes. So we will have a Cart and an Order, in which each Order will record the transaction history. Now comes the important part, which is your management role. You said you want to manage your own products. Does this mean you need a separate area where you can add new products, change prices, or delete products that are no longer available?

Customer: That's right! I can't call you every time there's a new product, I need a tool for me to do it myself. Only I can do that.

Dev: Of course. This is where the concept of decentralization comes in. In our system, there will be at least two roles:

Customer: They are normal users, they can view products, manage their shopping carts, and make purchases.

Admin: You. You have all the rights of a customer, and you have special rights to manage products.

Customer: That makes sense. You organized my ideas so that they are easy to understand.

Dev: Yes, that's my job. Now, to ensure that our system works well and is easy to upgrade in the future, I will not build it arbitrarily.

Customer: Sounds professional and safe, so I trust your proposal. Just make sure the website runs well, is stable, and I can easily add new features later.

Dev: Don't worry. Based on what we discussed, the next step is to sketch out some simple diagrams to describe these functions and architecture. I will present them to you next time and confirm them again before we start building.

Customer: Great. Thank you, today's conversation was very productive.

# Use case description

Make Transaction

* Actor: Customer, Admin
* Goal: Enable user to buy key for the app
* Precondition:
  + Customer has to be logged in
* Postcondition:
  + A key is generated and given to the customer
  + The key and receipt are sent to the customer’s email
  + Transaction of customer is updated
* Basic Flow
  1. Customer choose product
  2. Customer enter credit card information
  3. After a successful transaction, System Key and receipt to the customer’s email
* Alternative Flow

2A. Credit card information is incorrect

3A. Key fails to generate and the customer is refunded

* Exceptions
  + Connect to server failed

Manage Profile

* Actor: Customer, Admin
* Goal: Let the user change their display name, change password, change profile pic, view transaction history and view current cart
* Precondition:
  + User must be logged in
  + User must not be banned
* Postcondition
  + User’s information is succesfully updated
* Basic Flow
  1. User enter user management system
  2. User view or make changes they want
  3. User update their information
* Alternative Flow

3A. The Account failed to update

* Exception
  + User is banned

Register and login

* Actor: Customer, Admin
* Goal: User can create an account, login and log out of the account
* Precondition:
  + Username or email must not be taken
  + Password must meet the minimum security requirement
* Postcondition:
  + Account is successfully created
* Basic Flow
  1. Customer goes into account registration page
  2. Customer enter required field
  3. Customer press create account
  4. Customer login their account in the login page
* Alternative Flow

3A. Customer’s username or email is taken

4A. Customer write wrong login information

* Exception
  + Sql database fail to connect or update

View Home Page

* Actor: Customer, Admin
* Goal: User can view all product, based on the filtered product
* Precondition:
  + User get into the site
* Post Condition:
  + User can browse items, or use all of the site’s feature
* Basic Flow:
  1. User enter the site’s domain into the web browser
  2. User browse site and use the site’s feature
* Alternative Flow:
  + User can’t access site cause the site is down (User enter the site’s domain into the web browser)

Search product

* Actor: Customer, Admin
* Goal: User can filter the home page’s item
* Precondition:
  + User must on the home page
* Postcondition:
  + The item on the home page is filtered based on the search bar
* Basic flow:
  1. User enter the condition (in string) into the search bar and confirm
  2. System the filtered items
* Alternative flow:

1A. There is no item that fit the filter

View Product

* Actor: Customer, Admin
* Goal: User can view product description before buying, make comment or rate the product
* Precondition
  + Customer is logged in if they want to comment ,rate or buy the product
* Postcondition:
  + Comment or rating the customer has made is updated
* Basic Flow
  1. User find the product they want
  2. User go into the product’s page
  3. User buy the item
  4. User comment or rate
* Alternative Flow

4A. User do not comment or rate or buy the product

* Exception
  + The product page is not available

Manage cart

* Actor: Customer, Admin
* Goal: User can make a list of item the user wants to purchase, then user can buy all the item in the cart at once
* Precondition:
  + User must be logged in
* Postcondition:
  + User is able store their order inside the cart
* Basic flow:
  1. User press add cart when viewing product
  2. User view all the items in the cart
* Alternative flow:

1A. Item failed to be added to cart

Check order history

* Actor: Customer, Admin
* Goal: User can check their order history and its detail
* Precondition:
  + User must be logged in
* Postcondition:
  + User can view all of their order and it’s detail
* Basic flow
  1. User goes into their product history
  2. User view their order history
  3. User goes into order’s detail
* Alternative flow

1A. Order history return empty

Add Product

* Actor: Admin
* Goal: User can add product to the database
* Precondition:
  + User must be admin
* Postcondition:
  + An item is added
* Basic flow:
  1. User Press add product in home page
  2. User enter product’s detail and add cover picture
  3. User confirm add product
* Alternative flow:

3A. User doesn’s add enough detail to the product

3B. Crucial information like product name is already in the database

3C. Product is not added due to server problem

Delete product

* Actor: Admin
* Goal: Allow User to delete product in the database
* Precondition:
  + User must be admin
* Postcondition
  + The item is deleted in the database
* Basic flow
  1. User find the product they want to delete in the home page
  2. User confirm deletion
* Alternative flow

2A. User is unable to delete the item due to server problem

Edit product

* Actor: Admin
* Goal: Allow user to edit already existed product
* Precondition:
  + User must be admin
* Postcondition:
  + The product is edited
* Basic flow
  1. User find the product
  2. User goes into detail edit page
  3. User edit product detail and confirm
* Alternative flow:

3A. User doesn’t change any information

3B. System edit confirmation fail due to server problem

@startuml

actor Customer

actor Admin

left to right direction

rectangle "Software Selling Application" {

(View Home Page)

(Search Product)

(View Product)

(Register and Log In)

(Manage Profile)

(Manage Cart)

(Check Order History)

(Make Transaction)

Customer --> (View Home Page)

Customer --> (Search Product)

Customer --> (View Product)

Customer --> (Register and Log In)

Customer --> (Manage Profile)

Customer --> (Manage Cart)

Customer --> (Check Order History)

(Manage Cart) <.. (Make Transaction) : <<extends>>

Admin --> (View Home Page)

Admin --> (Search Product)

Admin --> (View Product)

Admin --> (Manage Profile)

}

rectangle "Admin Actions" {

(Add Product)

(Edit Product)

(Delete Product)

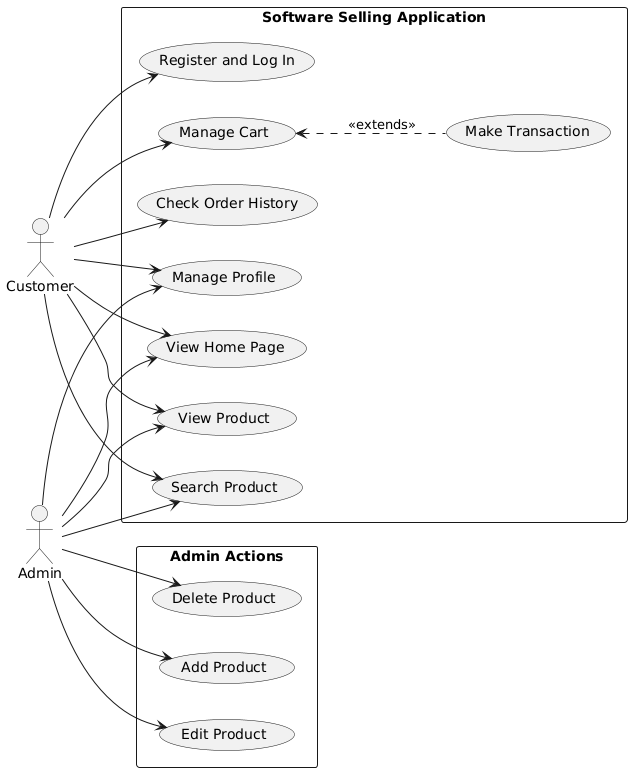
Admin --> (Add Product)

Admin --> (Edit Product)

Admin --> (Delete Product)

}

@enduml



]

# Class Diagram

|  |  |  |
| --- | --- | --- |
| Class | Responsibilities | Collaborators |
| Customer | Manage your account and authentication (register, log in, log out, update information), browse products and manage shopping cart, place orders, track orders and confirm receipts, make payments | Product, Cart, Order, Payment Gateway |
| User | A Superclass for customer and admin, contain name and id | User, Admin |
| Admin | Authenticate administrative rights, manage user accounts, manage product catalogs, process customer orders | Customer, Product, Order |
| Product | Provide detailed product information (name, price, product description), track status (inventory quantity) | Customer, Admin, Cart |
| Cart | Keeps user selected products, allows managing items (add/delete), calculates subtotals of items | Customer, Product, Order |
| Order | Record purchase details (user, product, total, shipping information), manage order status, process payments | Customer, Product, Cart, Admin, Payment Gateway |
| Payment Gateway | Process payment transactions, return payment status | Order, Customer |

@startuml

title Class Diagram - DuongAnhMinhDaoPhuocHai

Class User

{

-Name: string

-Id: string

-Role: string

}

Class Order

{

+OrderId: string

+DayBought: Date

+From: User

}

Class Cart

{

+ListOfProducts: List<Product>

}

Class ApplicationDbContext

{

-Users DbSet<User>

-Orders DbSet<Order>

-Products DbSet<Product>

-Categories DbSet<Category>

}

Class HomePageController

{

-db : ApplicationDbContext

+AddProduct(listOfProduct: Product)

+ManageCustomer(customer: Customer)

+DeleteProduct(listOfProduct: Product)

+EditProduct(product: Product)

+AddProductToCart(product: Product, categoryList List<Category>)

+MakePayment(gateway: PaymentGateway, user: User)

+SearchProduct(Name: string)

+ReturnProductView(id: int)

+UpdateProfile(user: User)

}

Class DbSet<T>

{

+GetList(): List<T>

+FindById(id: int)

+DeleteFromDB(item: T)

}

Class ClientView

{

-user: User

+Register(loginManager: LoginManager)

+Login(loginManager: LoginManager)

+GetHomePageView(controller: HomePageController)

+GetProductView(controller: HomePageController)

+SearchProduct(controller: HomePageController)

+GetCartView(controller: HomePageController)

+AddToCart(controller: HomePageController)

+GetOrderHistoryView(controller: HomePageController)

+GetProfileView(controller: HomePageController)

+ConfirmProfileEdit(controller: HomePageController)

+MakeTransaction(controller: HomePageController)

+DeleteProduct(controlelr: HomePageController)

+AddProduct(controller: HomePageController)

+EditProduct(controller: HomePageController)

}

Class Product

{

-productName: string

-productId: string

-category: Category

-images: string

}

Class Category

{

-Category\_id: string

-Category\_Name: string

}

Class LoginManager

{

+Login() : User

+Register()

}

Class PaymentGateway

{

+ProcessPayment()

}

HomePageController ..> PaymentGateway

HomePageController ..> Order

HomePageController ..> User

HomePageController ..> Product

Cart <.. HomePageController

Product --o Category

HomePageController ..> Category

HomePageController o-- ApplicationDbContext

User --o ApplicationDbContext

Product --o ApplicationDbContext

Category --o ApplicationDbContext

Order --o ApplicationDbContext

LoginManager ..> User

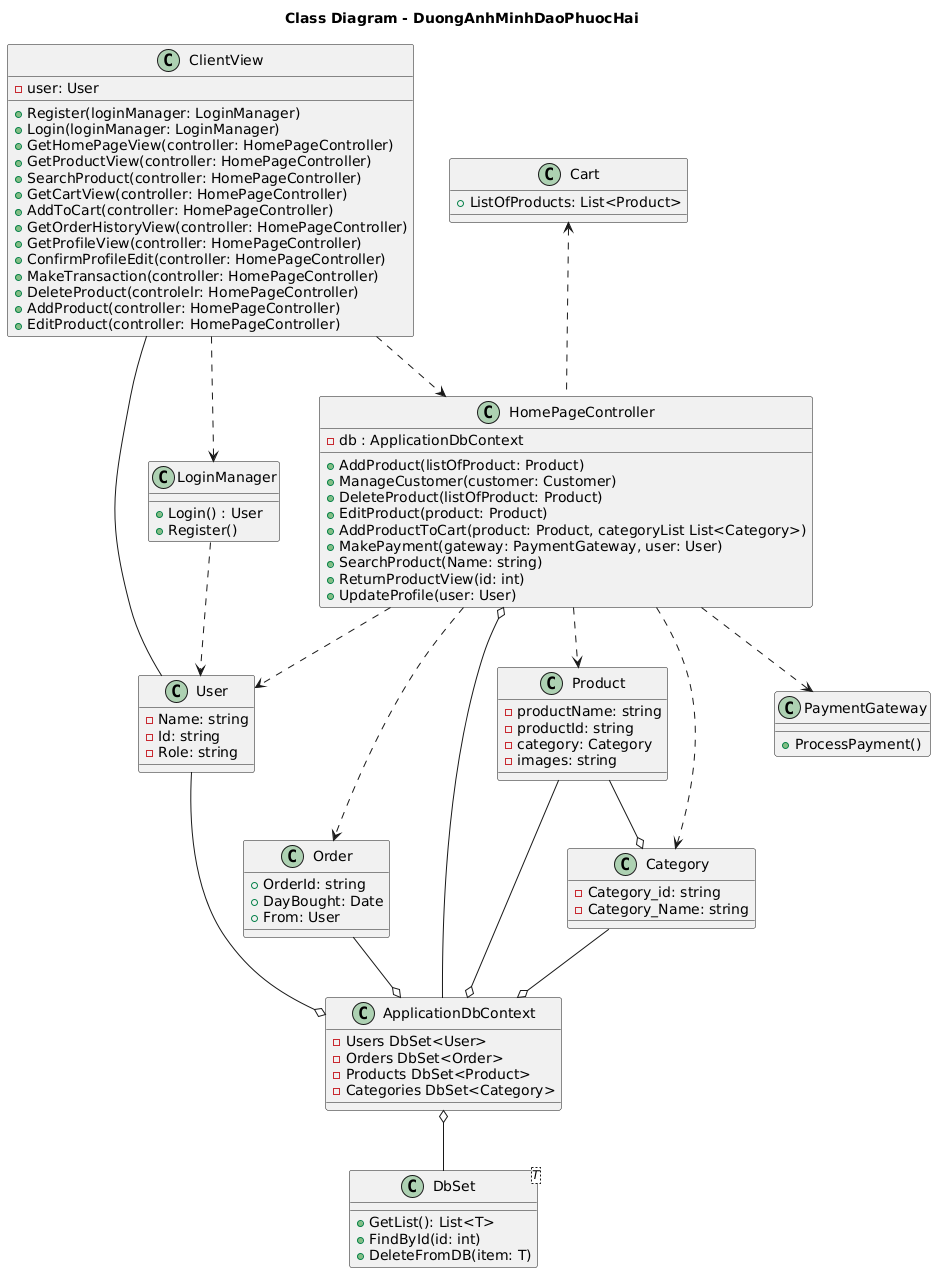
ApplicationDbContext o-- DbSet

ClientView ..> LoginManager

ClientView ..> HomePageController

ClientView -- User

@enduml



# Activity Diagram

Activity 1: Customer use the search product function

@startuml

title Activity Diagram - Customer Flow

start

:Customer accesses application;

:ClientView.GetHomePageView();

:HomePageController displays home page;

if (Customer wants to search?) then (yes)

:ClientView.SearchProduct();

:HomePageController.SearchProduct();

:Display search results;

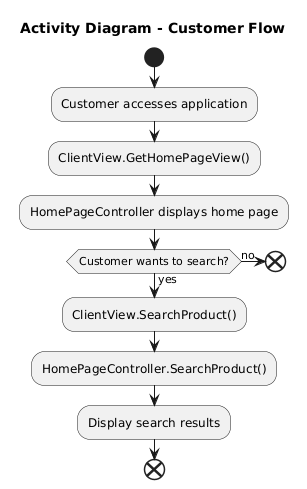
end

else (no)

end

endif

@enduml



## Activity 2: customer make transaction

@startuml

:Customer selects product;

:ClientView.GetProductView();

:HomePageController.ReturnProductView();

if (Customer logged in?) then (no)

:ClientView.Register() or Login();

:LoginManager processes authentication;

if (Authentication successful?) then (no)

:Display error message;

stop

else (yes)

endif

else (yes)

endif

if (Customer wants to add to cart?) then (yes)

:ClientView.AddToCart();

:HomePageController.AddProductToCart();

:Update Cart with Product;

if (Customer wants to make transaction?) then (yes)

:ClientView.MakeTransaction();

:HomePageController.MakePayment();

:PaymentGateway.ProcessPayment();

if (Payment successful?) then (yes)

:Create Order record;

:Clear Cart;

:Display confirmation;

else (no)

:Display payment error;

endif

else (no)

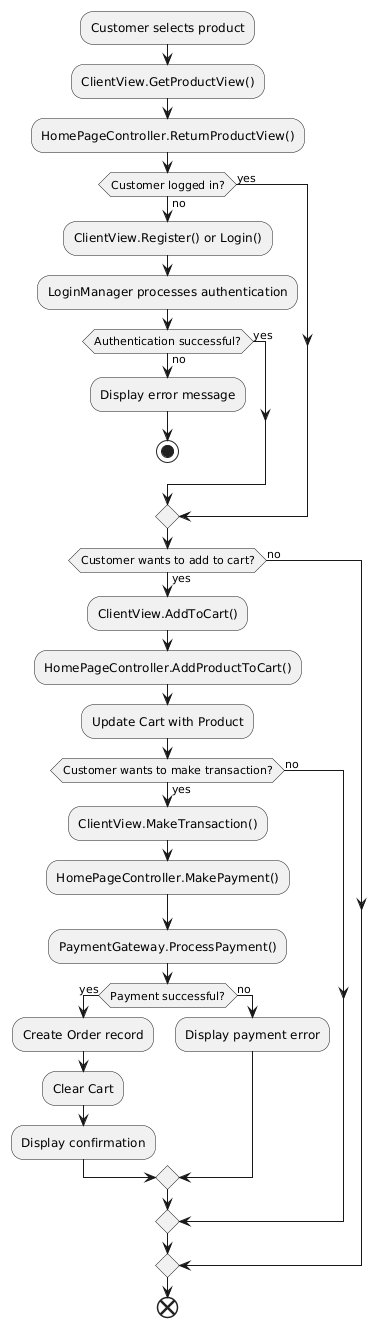
endif

else (no)

endif

end

@enduml



## Activity 3: Customer update profile

@startuml

if (Customer wants to view profile?) then (yes)

:ClientView.GetProfileView();

:Display user profile;

if (Customer wants to edit profile?) then (yes)

:ClientView.ConfirmProfileEdit();

:HomePageController.UpdateProfile();

:Update User information;

else (no)

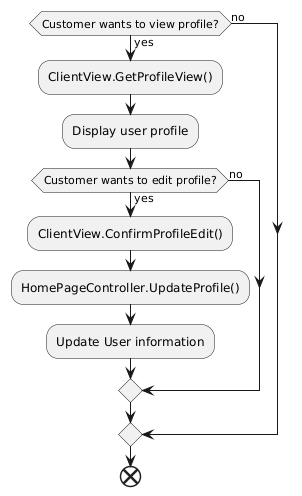
endif

else (no)

endif

end

@enduml



## Activity 4: Customer view order history

@startuml

if (Customer wants order history?) then (yes)

:ClientView.GetOrderHistoryView();

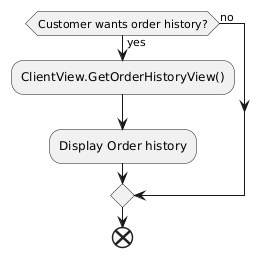
:Display Order history;

else (no)

endif

end

@enduml



## Activity 5: Admin manage products

@startuml

title Activity Diagram - Admin Flow

start

:Admin accesses application;

:ClientView.GetHomePageView();

:HomePageController displays home page;

if (Admin wants to manage products?) then (yes)

if (Action type?) then (Add)

:ClientView.AddProduct();

:HomePageController.AddProduct();

:Save Product to ApplicationDbContext;

elseif (Edit) then

:Select Product to edit;

:ClientView.EditProduct();

:HomePageController.EditProduct();

:Update Product in ApplicationDbContext;

elseif (Delete) then

:Select Product to delete;

:ClientView.DeleteProduct();

:HomePageController.DeleteProduct();

:Remove Product from ApplicationDbContext;

endif

:Display updated product list;

else (no)

endif

if (Admin wants to manage customers?) then (yes)

:HomePageController.ManageCustomer();

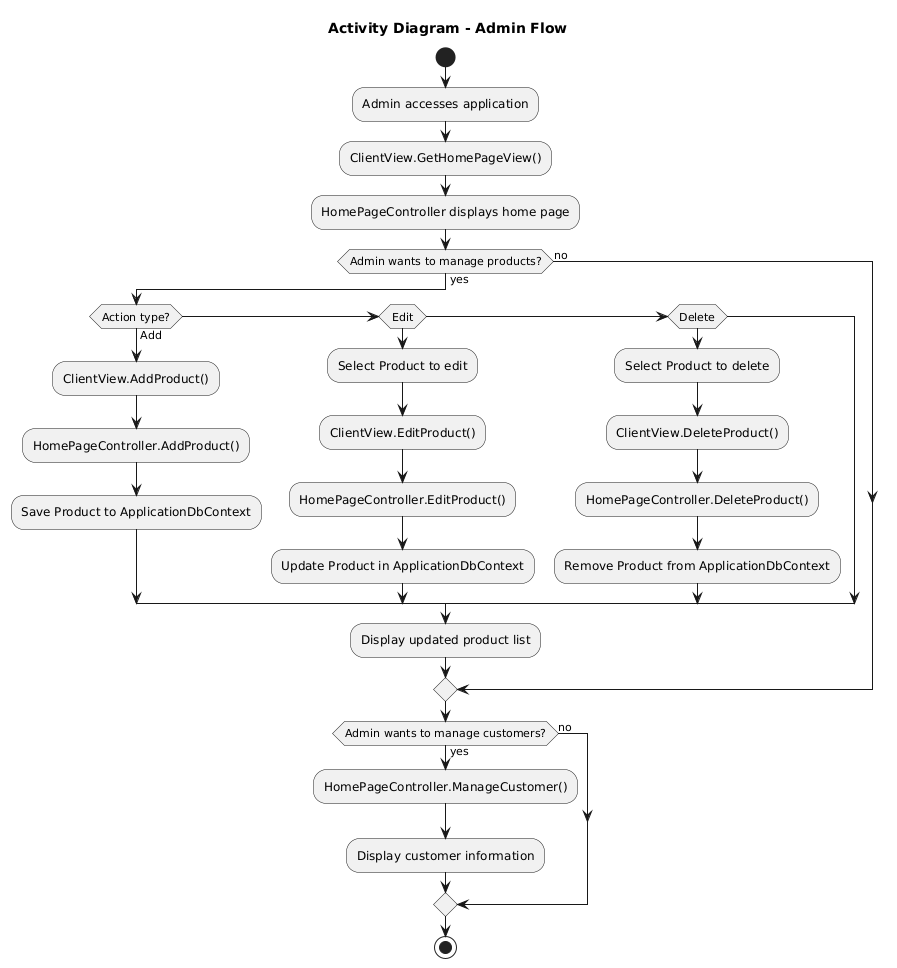
:Display customer information;

else (no)

endif

stop

@enduml



# Sequence diagram

## Sequence 1: Customer register and login

@startuml

title Sequence Diagram - Customer Registration and Login

actor Customer

participant ClientView

participant LoginManager

participant User

participant ApplicationDbContext

Customer -> ClientView: Access application

ClientView -> ClientView: Display registration/login form

alt Registration Process

Customer -> ClientView: Register()

ClientView -> LoginManager: Register()

LoginManager -> User: Create new User instance

LoginManager -> ApplicationDbContext: Save User to database

ApplicationDbContext --> LoginManager: Confirmation

LoginManager --> ClientView: Registration successful

ClientView --> Customer: Display success message

else Login Process

Customer -> ClientView: Login()

ClientView -> LoginManager: Login()

LoginManager -> ApplicationDbContext: Validate User credentials

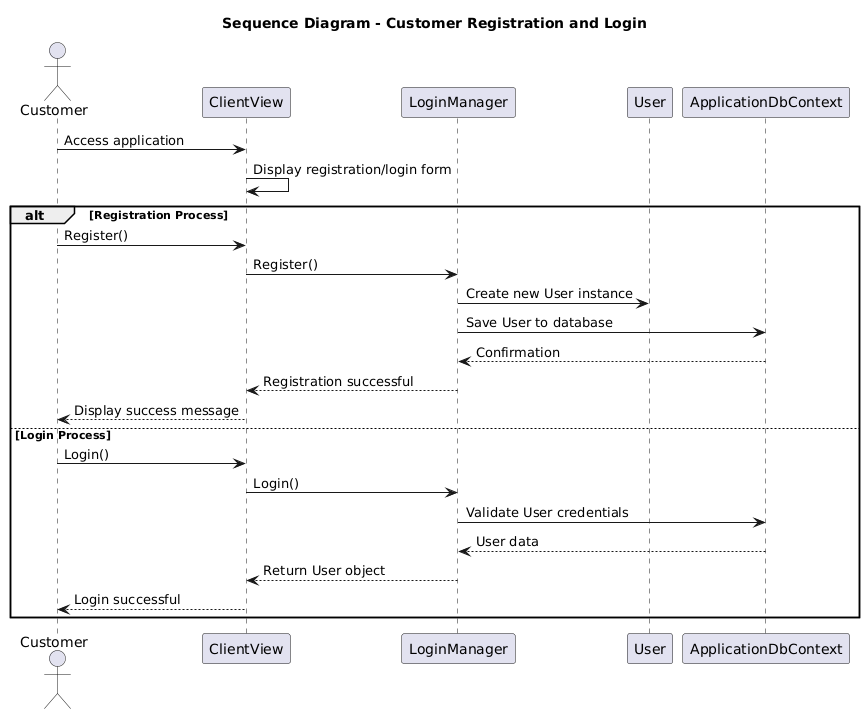
ApplicationDbContext --> LoginManager: User data

LoginManager --> ClientView: Return User object

ClientView --> Customer: Login successful

end

@enduml



## Sequence 2: Customer search product

@startuml

actor Customer

participant ClientView

participant HomePageController

participant ApplicationDbContext

participant "DbSet<Product>" as ProductDbSet

participant Product

participant Category

Customer -> ClientView: SearchProduct()

ClientView -> HomePageController: SearchProduct(Name)

HomePageController -> ApplicationDbContext: Access Products

ApplicationDbContext -> ProductDbSet: GetList()

ProductDbSet --> ApplicationDbContext: List<Product>

ApplicationDbContext --> HomePageController: Filtered products

HomePageController --> ClientView: Search results

ClientView --> Customer: Display products

Customer -> ClientView: Select product

ClientView -> HomePageController: ReturnProductView(id)

HomePageController -> ApplicationDbContext: Access Products

ApplicationDbContext -> ProductDbSet: FindById(id)

ProductDbSet -> Product: Get product details

Product -> Category: Get category info

Category --> Product: Category details

Product --> ProductDbSet: Product with category

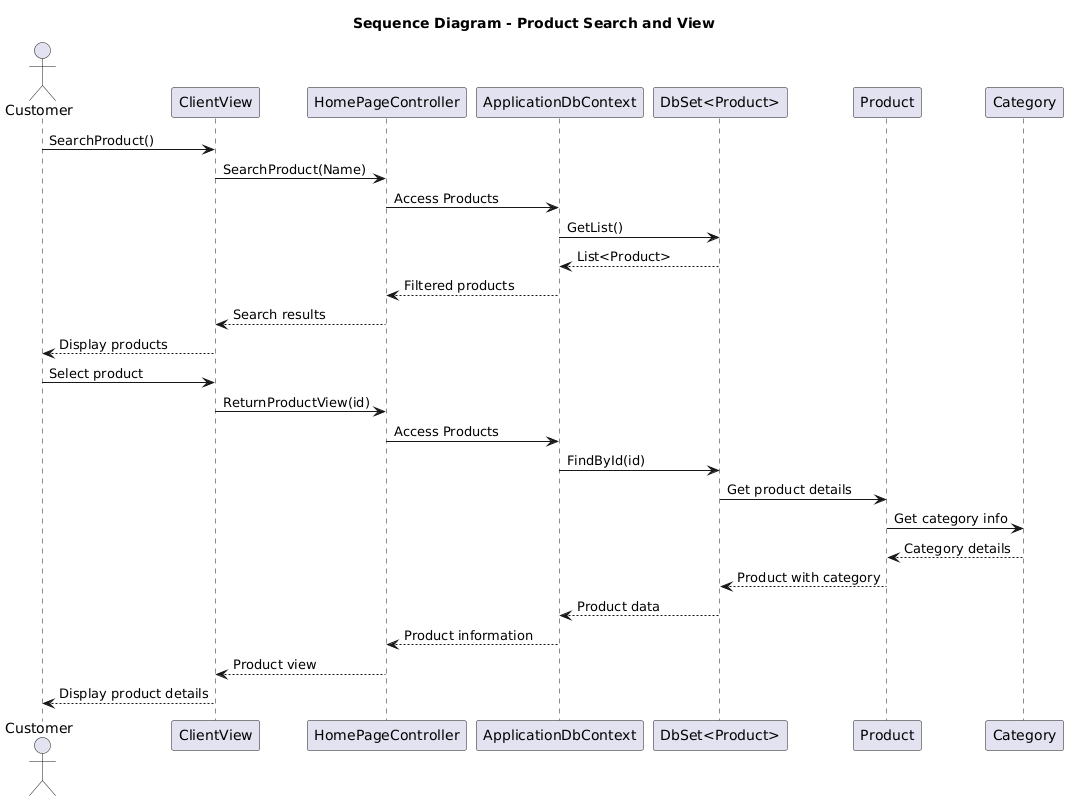
ProductDbSet --> ApplicationDbContext: Product data

ApplicationDbContext --> HomePageController: Product information

HomePageController --> ClientView: Product view

ClientView --> Customer: Display product details

@enduml



## Sequence 3: Customer make payment

@startuml

title Sequence Diagram - Add to Cart and Make Transaction

actor Customer

participant ClientView

participant HomePageController

participant Cart

participant Product

participant PaymentGateway

participant Order

participant ApplicationDbContext

participant User

Customer -> ClientView: AddToCart()

ClientView -> HomePageController: AddProductToCart(product, categoryList)

HomePageController -> Cart: Add Product to ListOfProducts

Cart --> HomePageController: Confirmation

HomePageController --> ClientView: Product added to cart

ClientView --> Customer: Display cart updated

Customer -> ClientView: MakeTransaction()

ClientView -> HomePageController: MakeTransaction()

HomePageController -> Cart: Get ListOfProducts

Cart --> HomePageController: List<Product>

HomePageController -> PaymentGateway: ProcessPayment()

PaymentGateway --> HomePageController: Payment result

alt Payment Successful

HomePageController -> Order: Create new Order

Order -> User: Set From field

HomePageController -> ApplicationDbContext: Save Order

ApplicationDbContext --> HomePageController: Order saved

HomePageController -> Cart: Clear ListOfProducts

HomePageController --> ClientView: Transaction successful

ClientView --> Customer: Display confirmation

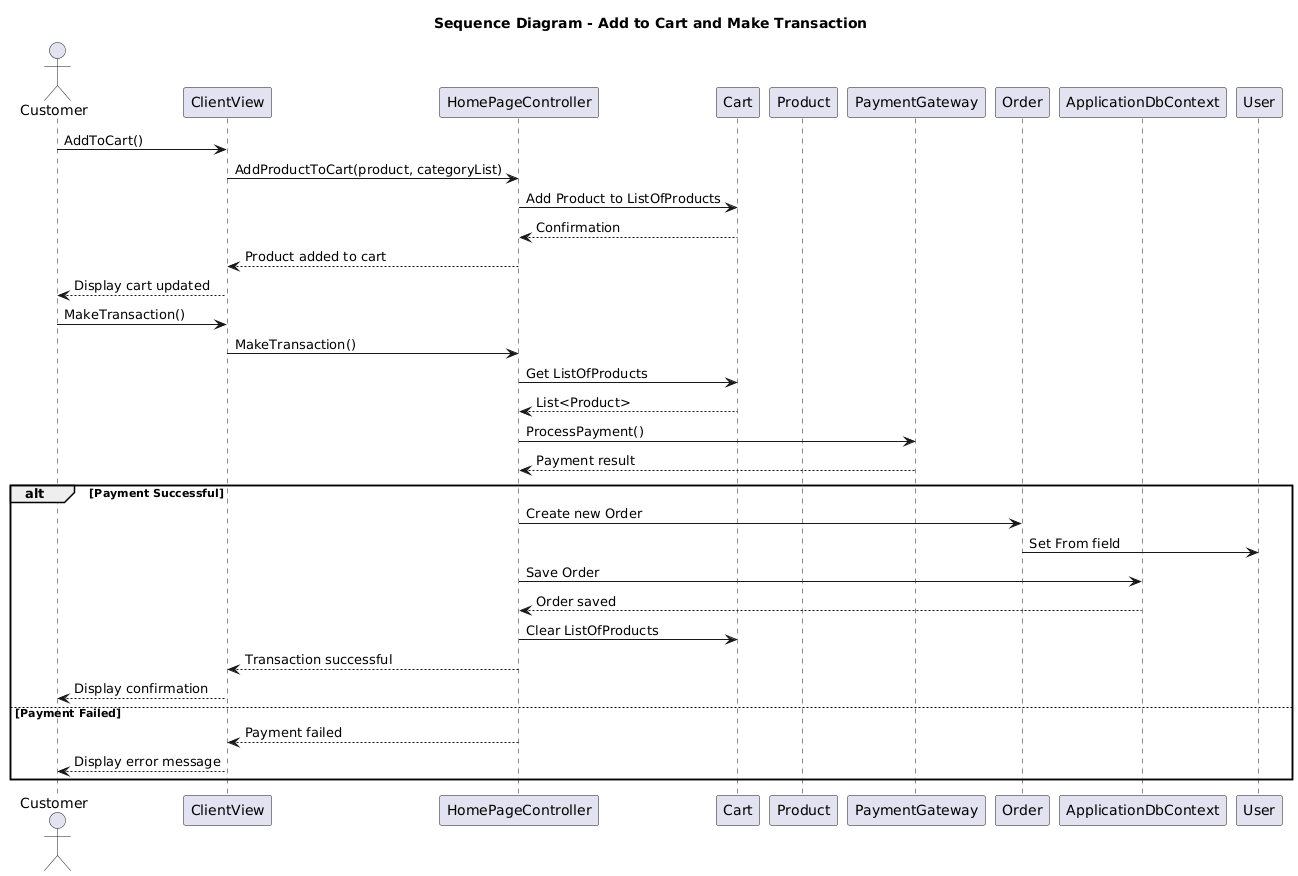
else Payment Failed

HomePageController --> ClientView: Payment failed

ClientView --> Customer: Display error message

end

@enduml



## Sequence 4: Admin manage product

@startuml

title Sequence Diagram - Admin Product Management

actor Admin

participant ClientView

participant HomePageController

participant ApplicationDbContext

participant "DbSet<Product>" as ProductDbSet

participant Product

participant Category

alt Add Product

Admin -> ClientView: AddProduct()

ClientView -> HomePageController: AddProduct(listOfProduct)

HomePageController -> Product: Create new Product instance

Product -> Category: Set category

HomePageController -> ApplicationDbContext: Access Products

ApplicationDbContext -> ProductDbSet: Add Product

ProductDbSet --> ApplicationDbContext: Product added

ApplicationDbContext --> HomePageController: Confirmation

HomePageController --> ClientView: Product added successfully

ClientView --> Admin: Display success message

else Edit Product

Admin -> ClientView: EditProduct()

ClientView -> HomePageController: EditProduct(product)

HomePageController -> ApplicationDbContext: Access Products

ApplicationDbContext -> ProductDbSet: FindById(productId)

ProductDbSet -> Product: Get existing product

Product --> ProductDbSet: Product data

HomePageController -> Product: Update product details

HomePageController -> ApplicationDbContext: Save changes

ApplicationDbContext --> HomePageController: Update confirmation

HomePageController --> ClientView: Product updated

ClientView --> Admin: Display success message

else Delete Product

Admin -> ClientView: DeleteProduct()

ClientView -> HomePageController: DeleteProduct(listOfProduct)

HomePageController -> ApplicationDbContext: Access Products

ApplicationDbContext -> ProductDbSet: DeleteFromDB(product)

ProductDbSet --> ApplicationDbContext: Deletion confirmation

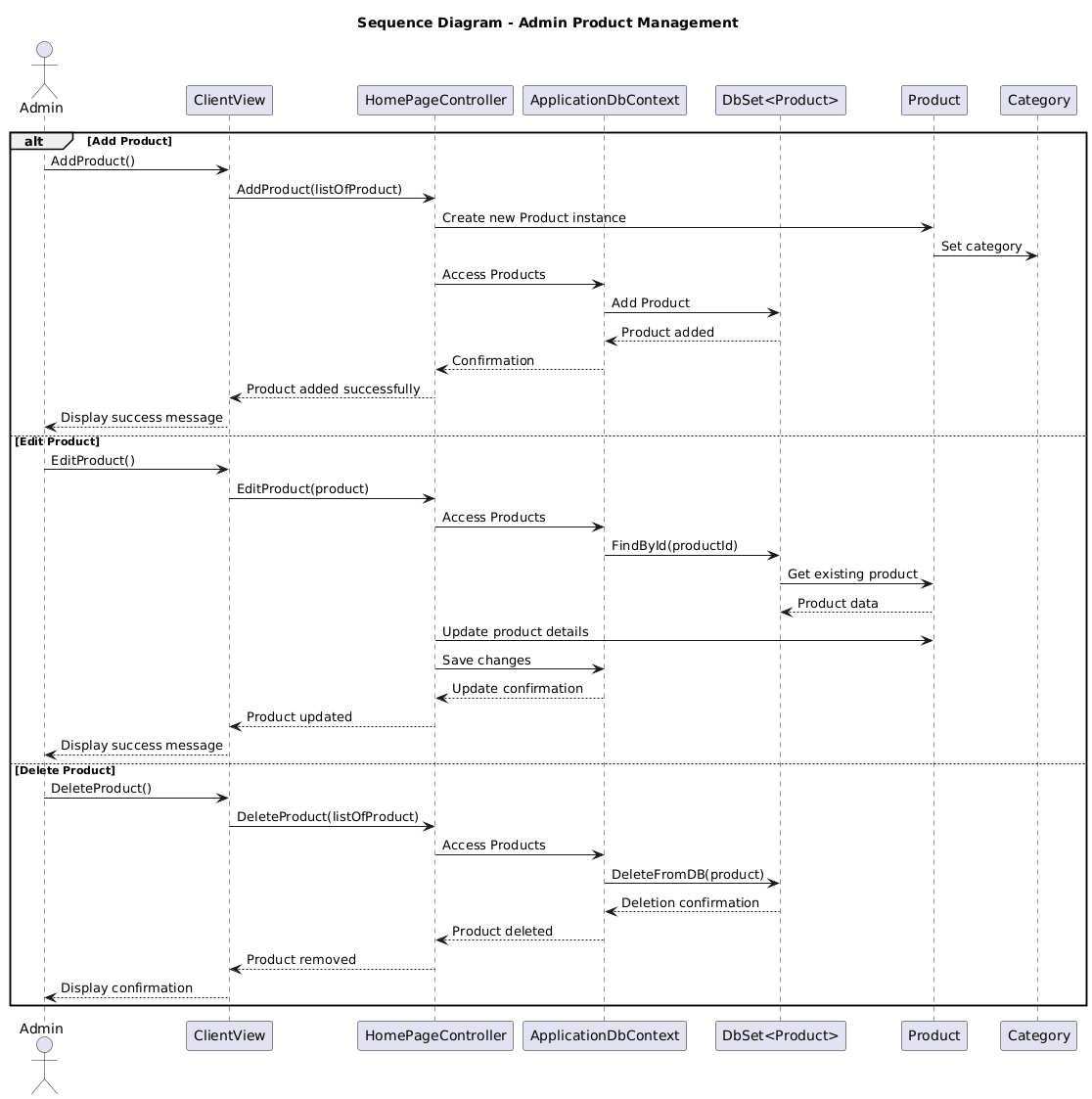
ApplicationDbContext --> HomePageController: Product deleted

HomePageController --> ClientView: Product removed

ClientView --> Admin: Display confirmation

end

@enduml



## Sequence 5: Customer manage their profile

@startuml

title Sequence Diagram - Profile Management

actor Customer

participant ClientView

participant HomePageController

participant User

participant ApplicationDbContext

Customer -> ClientView: GetProfileView()

ClientView -> HomePageController: Request profile data

HomePageController -> User: Get current user data

User --> HomePageController: User information

HomePageController --> ClientView: Profile data

ClientView --> Customer: Display profile

Customer -> ClientView: Edit profile

ClientView -> ClientView: Display edit form

Customer -> ClientView: ConfirmProfileEdit()

ClientView -> HomePageController: UpdateProfile(user)

HomePageController -> User: Update user properties (Name, etc.)

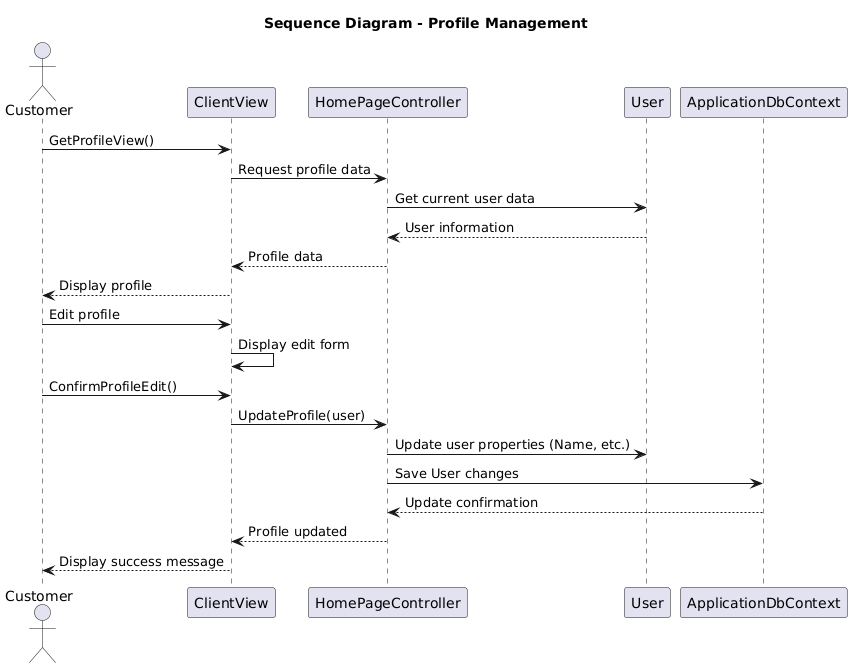
HomePageController -> ApplicationDbContext: Save User changes

ApplicationDbContext --> HomePageController: Update confirmation

HomePageController --> ClientView: Profile updated

ClientView --> Customer: Display success message

@enduml



## Sequence 6: Customer check order history

@startuml

title Sequence Diagram - Order History

actor Customer

participant ClientView

participant HomePageController

participant ApplicationDbContext

participant "DbSet<Order>" as OrderDbSet

participant Order

participant User

Customer -> ClientView: GetOrderHistoryView()

ClientView -> HomePageController: Request order history

HomePageController -> User: Get current user ID

HomePageController -> ApplicationDbContext: Access Orders

ApplicationDbContext -> OrderDbSet: GetList()

OrderDbSet -> Order: Filter by User (From field)

Order --> OrderDbSet: User's orders

OrderDbSet --> ApplicationDbContext: List<Order>

ApplicationDbContext --> HomePageController: Order history

HomePageController --> ClientView: Order data

ClientView --> Customer: Display order history

@enduml

