

Software Design Specification

**PawHouse**

**Version: 1.0**

– Can Tho, January 2025 –

# Member

| # | Full name | Student ID | Account |
| --- | --- | --- | --- |
| 1 | Phan Thành Hưng | CE180265 | HungPT |
| 2 | Võ Chánh Huy | CE181002 | HuyVC |
| 3 | Nguyễn Huỳnh Chấn Hưng | CE180150 | HungNHC |
| 4 | Trương Phú Hưng | CE180618 | HungTP |
| 5 | Trần Hữu Tín | CE180262 | TinTH |
| 6 | Huỳnh Hoàng Huy | CE181847 | HuyHH |

# Revision History

# 

\*A - Added M - Modified D - Deleted

| **Version** | **Date** | **A,M,D** | **Assignee** | **Change Description** |
| --- | --- | --- | --- | --- |
| 0.1 | 12/02/2025 | A | HuyHH | Add Database Schema and Table Description. |
| 0.1 | 13/02/2025 | A | HungPT | Add Code Package, Package Des, and Use-Case for Guestsand Customers. |
| 0.1 | 13/02/2025 | A | HungNHC | Add Code Designs for Dashboard, Add Staff, Contact. |
| 0.1 | 13/02/2025 | A | TinTH | Add Code Designs for Delete Category, View Account, Edit/Add Product and Add to Cart. |
| 0.1 | 13/02/2025 | M | HuyVC | Modified Use-Case for Guests and Customers. |
| 0.1 | 13/02/2025 | A | HungTP | Add Use-Case for Admin and Staff. |
| 0.1 | 13/02/2025 | A | HungPT | Add Use-Case for Doctor and Customer. |
| 0.1 | 13/02/2025 | A | TinTH | Add Code Designs for Delete from Cart, Search Product, Checkout, Delete Product. |
| 0.2 | 15/02/2025 | A | HungPT | Add Code Designs for Log out, Delete Pet, Add Category, Register. |
| 0.2 | 15/02/2025 | A | HuyHH | Add Code Design for Add staff, Update staff profile, Disable/Enable staff, Update order status. |
| 0.2 | 15/02/2025 | M | HungNHC | Modified Use-Case for Guests, Customers, Doctor, Admin and Staffs |
| 0.2 | 15/02/2025 | A | TinTH | Code Designs for Delete from Cart, Search Product, Checkout, Delete Product. |
| 0.2 | 15/02/2025 | M | HuyVC | Modify all sequence and class diagram |
| 0.2 | 15/02/2025 | M | HungTP | Modify Login, register, forget password sequence diagram |
| 0.2 | 15/02/2025 | M | HungPT | Modify add new Pet |

# 

# 

# Contents

[**Member 2**](#_heading=h.oo9kfa6tvt4r)

[**Revision History 2**](#_heading=h.hns5yzahk86)

[**Contents 4**](#_heading=h.d67dwzdz7jwh)

[**I. Overview 11**](#_heading=h.30j0zll)

[1. Code Packages 11](#_heading=h.1fob9te)

[2. Package Descriptions 12](#_heading=h.tyjcwt)

[3. Database Design 14](#_heading=h.17dp8vu)

[3.1 Database schema 14](#_heading=h.3rdcrjn)

[3.2. Table Descriptions 15](#_heading=)

[**II. Use Case Diagram 20**](#_heading=h.cnfcxlbn3e7n)

[Figure 1.2.1: Use Case Guest 22](#_heading=h.p5jif8wfxwq9)

[Figure 1.2.2: Use Case Customer 23](#_heading=h.q17ltdwud56q)

[Figure 1.2.3: Use Case Staff 24](#_heading=h.2sx2895qkfe0)

[Figure 1.2.4: Use Case Doctor 25](#_heading=h.2crvx7c7kdvr)

[Figure 1.2.5: Use Case Admin 26](#_heading=h.ik7bagdhf0op)

[**III. Code Designs 27**](#_heading=h.cwi42yidzbdo)

[1. Login: 27](#_heading=h.2dtglux4nf8m)

[A.Class Diagram 27](#_heading=h.rs1ab3ldpcy3)

[B.Class Specifications 27](#_heading=h.xw59abw2mdxk)

[C.Sequence Diagram(s) 30](#_heading=h.bvwwk9slqk1z)

[D.Database Queries 30](#_heading=h.ramawwmw640k)

[2. Register: 30](#_heading=h.5eqyle3iq6od)

[A. Class Diagram 31](#_heading=h.me32td1xduqk)

[B. Class Specifications 31](#_heading=h.gmxoasdqccc1)

[C. Sequence Diagram(s) 33](#_heading=h.sz3o0q8ti1no)

[D. Database Queries 33](#_heading=h.56hq39ny2cpo)

[3. Forget: 33](#_heading=h.qhdu98gk7k3f)

[A. Class Diagram 34](#_heading=h.vlc5o5b5niof)

[B. Class Specifications 34](#_heading=h.d8p5dfeg1cdo)

[C. Sequence Diagram(s) 36](#_heading=h.lxry9i97yerj)

[D. Database Queries 36](#_heading=h.vldlswys0jx3)

[4. Edit customer’s profile: 37](#_heading=h.unenpx2upu8x)

[A. Class Diagram 37](#_heading=h.44wguil65y5k)

[B. Class Specifications 38](#_heading=h.b58ox5bkvnt)

[C. Sequence Diagram(s) 39](#_heading=h.yhmcsjilh6zx)

[D. Database Queries 39](#_heading=h.xc5r0xz3nibh)

[5. Change pass: 39](#_heading=h.6wn35tkhmxnd)

[A. Class Diagram 40](#_heading=h.gleh6f3hy9cp)

[B. Class Specifications 40](#_heading=h.dudf3jf4t5bn)

[C. Sequence Diagram(s) 42](#_heading=h.curhm8z4250o)

[D. Database Queries 42](#_heading=h.l2jdzo1z7cch)

[6. Login with Google: 42](#_heading=h.ld9njhymuu6t)

[A. Class Diagram 43](#_heading=h.q2jcl6cqkm6o)

[B. Class Specifications 43](#_heading=h.6vlawjx0k89a)

[C. Sequence Diagram(s) 45](#_heading=h.4rowv034itu8)

[D. Database Queries 45](#_heading=h.dsvt2t2aqn1r)

[7. Add service: 45](#_heading=h.rhx5fywkvn6t)

[A. Class Diagram 46](#_heading=h.vrlaayqtnh45)

[B. Class Specifications 46](#_heading=h.5j7icgtw8dqz)

[C. Sequence Diagram(s) 48](#_heading=h.uoc80fcrlpfg)

[D. Database Queries 48](#_heading=h.rt3egbcvs19e)

[8. Update service: 48](#_heading=h.8csa64ljaxbo)

[A. Class Diagram 49](#_heading=h.aczbdbypl46k)

[B. Class Specifications 49](#_heading=h.7ep89z5gzs8f)

[C. Sequence Diagram(s) 51](#_heading=h.22o1by8az1w5)

[D. Database Queries 51](#_heading=h.ul60y34hyo3z)

[9. Delete service: 51](#_heading=h.77ridei1ikna)

[A. Class Diagram 52](#_heading=h.c7vtjgvb1ryh)

[B. Class Specifications 52](#_heading=h.8796hssniv3a)

[C. Sequence Diagram(s) 54](#_heading=h.qmys2ftnx6w9)

[D. Database Queries 54](#_heading=h.9atjhw799aa9)

[10. Add to cart: 54](#_heading=h.ifn4kx7r0etu)

[A. Class Diagram 55](#_heading=h.wv94vx2u45t7)

[B. Class Specifications 55](#_heading=h.bdknwowfg64r)

[C. Sequence Diagram(s) 57](#_heading=h.cb9lxdrsgbml)

[D. Database Queries 57](#_heading=h.b1sbctc97sze)

[11. Payment: 57](#_heading=h.nugbz64qlyr2)

[A. Class Diagram 58](#_heading=h.up9wxilrf2o)

[B. Class Specifications 58](#_heading=h.pj3xkz9ma236)

[C. Sequence Diagram(s) 61](#_heading=h.lqn16rwobie1)

[D. Database Queries 61](#_heading=h.1z3gm3x9bc2u)

[12. Add product: 62](#_heading=h.82i43kxlb6s5)

[A. Class Diagram 62](#_heading=h.msaymmva3br)

[B. Class Specifications 62](#_heading=h.e6r61yeg7n67)

[C. Sequence Diagram(s) 64](#_heading=h.fwvsv2d77ubm)

[D. Database Queries 64](#_heading=h.7b6lrv59prgm)

[13. Edit Product: 64](#_heading=h.fkq92z7mhiq8)

[A. Class Diagram 65](#_heading=h.gykzhao7agkv)

[B. Class Specifications 65](#_heading=h.9rzghc7u93dp)

[C. Sequence Diagram(s) 67](#_heading=h.mybdyxs6rftn)

[D. Database Queries 67](#_heading=h.4ftccz2p3ia2)

[14. Delete Product: 67](#_heading=h.m63ij4smal37)

[A. Class Diagram 68](#_heading=h.hunrebm1k7o4)

[B. Class Specifications 68](#_heading=h.m2zwv7e9lhlv)

[C. Sequence Diagram(s) 70](#_heading=h.8b62cggdu23z)

[D. Database Queries 70](#_heading=h.o5nk8xw29cmy)

[15. Add Category: 70](#_heading=h.lslnco67vnss)

[A. Class Diagram 71](#_heading=h.2qgv7k6pkuel)

[B. Class Specifications 71](#_heading=h.kyp0raiihubk)

[C. Sequence Diagram(s) 73](#_heading=h.a6dl2x58itjf)

[D. Database Queries 73](#_heading=h.kqt5qzykuylm)

[16. Edit Category: 74](#_heading=h.7ctxhwg8rtmh)

[A. Class Diagram 74](#_heading=h.puccpkntbuwn)

[B. Class Specifications 74](#_heading=h.wgevkqg88bo3)

[C. Sequence Diagram(s) 77](#_heading=h.fx0nwjiwnxlp)

[D. Database Queries 77](#_heading=h.jax68j9g5f6b)

[17. Add Staff: 78](#_heading=h.wnc8p3ig7k7z)

[A. Class Diagram 78](#_heading=h.nhno13n4uwdn)

[B. Class Specifications 79](#_heading=h.x6sochgvtb68)

[C. Sequence Diagram(s) 80](#_heading=h.81mpseabvf1m)

[D. Database Queries 80](#_heading=h.qh6zjgokns84)

[18. Add Doctor: 80](#_heading=h.vym8nem8lm79)

[A. Class Diagram 81](#_heading=h.dk5gg5rkpdx1)

[B. Class Specifications 81](#_heading=h.ur1714aqu1us)

[C. Sequence Diagram(s) 83](#_heading=h.in3anog64fm5)

[D. Database Queries 83](#_heading=h.tes51f16vndp)

[19. Edit Doctor: 83](#_heading=h.pvlui2ncjxv0)

[A. Class Diagram 84](#_heading=h.33emjqg8l3id)

[B. Class Specifications 84](#_heading=h.hrd1dt5t6e4y)

[C. Sequence Diagram(s) 86](#_heading=h.bspufy4m6utv)

[D. Database Queries 86](#_heading=h.hwr4uvxhpeou)

[20. Delete Doctor: 86](#_heading=h.8wewd9t60qur)

[A. Class Diagram 87](#_heading=h.dvguvoptltq0)

[B. Class Specifications 87](#_heading=h.jvpreb9nnao4)

[C. Sequence Diagram(s) 88](#_heading=h.t16ncp3sirzv)

[D. Database Queries 89](#_heading=h.bg7wai2mt433)

[21. Update Staff: 89](#_heading=h.qyrhw8tchjoh)

[A. Class Diagram 90](#_heading=h.czm31qy6g0ga)

[B. Class Specifications 90](#_heading=h.ex0w9uhnpakw)

[C. Sequence Diagram(s) 92](#_heading=h.k2dxyeiqgkey)

[D. Database Queries 92](#_heading=h.tcb33abatjqv)

[22. Delete Staff: 93](#_heading=h.xu3zfzd6ymyo)

[A. Class Diagram 93](#_heading=h.wo8h5x932rae)

[B. Class Specifications 94](#_heading=h.8mqng69huru0)

[C. Sequence Diagram(s) 95](#_heading=h.egreswciiko5)

[D. Database Queries 95](#_heading=h.ps7thvy257x)

[23. Add pet 96](#_heading=h.w3jqkgwpf8xa)

[A. Class Diagram 96](#_heading=h.87e7pxdhjmp9)

[B. Class Specifications 96](#_heading=h.n5p3dqdoyiex)

[C. Sequence Diagram(s) 97](#_heading=h.ltgvlnb5pzn7)

[D. Database Queries 97](#_heading=h.mvnjrfh8bzpx)

[24. Update Pet: 97](#_heading=h.gk8oll9eminm)

[A. Class Diagram 98](#_heading=h.2fngjrx0ce0t)

[B. Class Specifications 98](#_heading=h.c2xds9nbjkuo)

[C. Sequence Diagram(s) 99](#_heading=h.ajv2mpugkxh9)

[D. Database Queries 100](#_heading=h.4va1w6dl5p0a)

[25. Delete Pet: 100](#_heading=h.whk9oa332zya)

[A. Class Diagram 101](#_heading=h.wk5o6sx81if4)

[B. Class Specifications 101](#_heading=h.oqi591o16hr7)

[C. Sequence Diagram(s) 103](#_heading=h.ba3f1ox7sznk)

[D. Database Queries 103](#_heading=h.e6qsec88vmp)

[26. Change order status: 103](#_heading=h.fziiyp96l0yz)

[A. Class Diagram 104](#_heading=h.tnwbwnjp28g1)

[B. Class Specifications 104](#_heading=h.po5i6a5i37t4)

[C. Sequence Diagram(s) 105](#_heading=h.yz61fltcqa8j)

[D. Database Queries 106](#_heading=h.6doiw15fg0p1)

[27. Adoption History: 106](#_heading=h.fz1rngu5k5w)

[A. Class Diagram 106](#_heading=h.diomztj1vd93)

[B. Class Specifications 107](#_heading=h.p3nwfzjf2qn6)

[C. Sequence Diagram(s) 108](#_heading=h.zgye771cie20)

[D. Database Queries 108](#_heading=h.1g9htmvasx8a)

[28. Edit product in cart: 109](#_heading=h.b7z981h5ttrc)

[A. Class Diagram 109](#_heading=h.ejzu8w4l3u6g)

[B. Class Specifications 109](#_heading=h.juhe1xfoxz6t)

[C. Sequence Diagram(s) 111](#_heading=h.9pkpw8tdtpb1)

[D. Database Queries 111](#_heading=h.j94urjzf58nx)

[29. Delete product in cart: 112](#_heading=h.eq1n4mlkala9)

[A. Class Diagram 112](#_heading=h.zfmrvx6tzw7i)

[B. Class Specifications 112](#_heading=h.x8htvmcrdvz2)

[C. Sequence Diagram(s) 114](#_heading=h.jbjjbwwzdsbq)

[D. Database Queries 114](#_heading=h.hqzeudxix3q)

[30. Add Order Product: 115](#_heading=h.yvfvozwd6xna)

[A. Class Diagram 115](#_heading=h.4jax01da6zww)

[B. Class Specifications 115](#_heading=h.5pyewlwgcva1)

[C. Sequence Diagram(s) 117](#_heading=h.z6j50c40ynz)

[D. Database Queries 117](#_heading=h.hu3xpv72t1cz)

[31. ConfirmOrder: 118](#_heading=h.y9fwtdsq75e5)

[A. Class Diagram 118](#_heading=h.65yhme5rge7i)

[B. Class Specifications 118](#_heading=h.73az6dr7ya62)

[C. Sequence Diagram(s) 120](#_heading=h.7u78gocmh98w)

[D. Database Queries 120](#_heading=h.nszqlmjdcndq)

[32. Manage Appointments: 121](#_heading=h.5sk0h0wbijtp)

[A. Class Diagram 121](#_heading=h.ux9uik8gjpmk)

[B. Class Specifications 121](#_heading=h.yvu1fef004pp)

[C. Sequence Diagram(s) 123](#_heading=h.odex0b25u18a)

[D. Database Queries 123](#_heading=h.hgw7lq806am5)

[33. Prescribe Medicine: 123](#_heading=h.rngdplh1gs1g)

[A. Class Diagram 124](#_heading=h.ikou57p96jbc)

[B. Class Specifications 124](#_heading=h.gv8ovdwavqxy)

[C. Sequence Diagram(s) 125](#_heading=h.1h43zsq3bvbz)

[D. Database Queries 125](#_heading=h.ayveca5myymx)

[34. Communicate with Patients: 126](#_heading=h.6kmslt97itpb)

[A. Class Diagram 126](#_heading=h.bly271j6pl4k)

[B. Class Specifications 126](#_heading=h.19vr03lx0n1t)

[C. Sequence Diagram(s) 126](#_heading=h.n0cccybyoxc1)

[D. Database Queries 126](#_heading=h.1rg6sc1x7z7c)

[35. Write Medical Reports: 127](#_heading=h.l60kyf845i3t)

[A. Class Diagram 127](#_heading=h.mkdjfnvnn5lt)

[B. Class Specifications 127](#_heading=h.45etq0neixvw)

[C. Sequence Diagram(s) 129](#_heading=h.pt3p2kkgs53w)

[D. Database Queries 129](#_heading=h.ltxs7r47jqdq)

[36. Add comment: 130](#_heading=h.xj6x7mgyyfoj)

[A. Class Diagram 130](#_heading=h.6ihe69tjrzm7)

[B. Class Specifications 131](#_heading=h.u2wosm4ycfpj)

[C. Sequence Diagram(s) 134](#_heading=h.9ynybydn4a6h)

[D. Database Queries 134](#_heading=h.v67s69wqj3ll)

[37. Delete comment: 135](#_heading=h.mmj9gtj589o0)

[A. Class Diagram 135](#_heading=h.ghccuigxq8fg)

[B. Class Specifications 135](#_heading=h.r5p3s574hoot)

[C. Sequence Diagram(s) 139](#_heading=h.yng0bnwhyzdj)

[D. Database Queries 139](#_heading=h.ax3a4bh6ijlt)

[38. Edit comment: 139](#_heading=h.33tm18use037)

[A. Class Diagram 140](#_heading=h.8v53e9t1aot8)

[B. Class Specifications 140](#_heading=h.m96hus1jzobp)

[C. Sequence Diagram(s) 144](#_heading=h.af0klmlu739e)

[D. Database Queries 144](#_heading=h.51p1epfc5g55)

# **I. Overview**

## 1. Code Packages

### 

### 

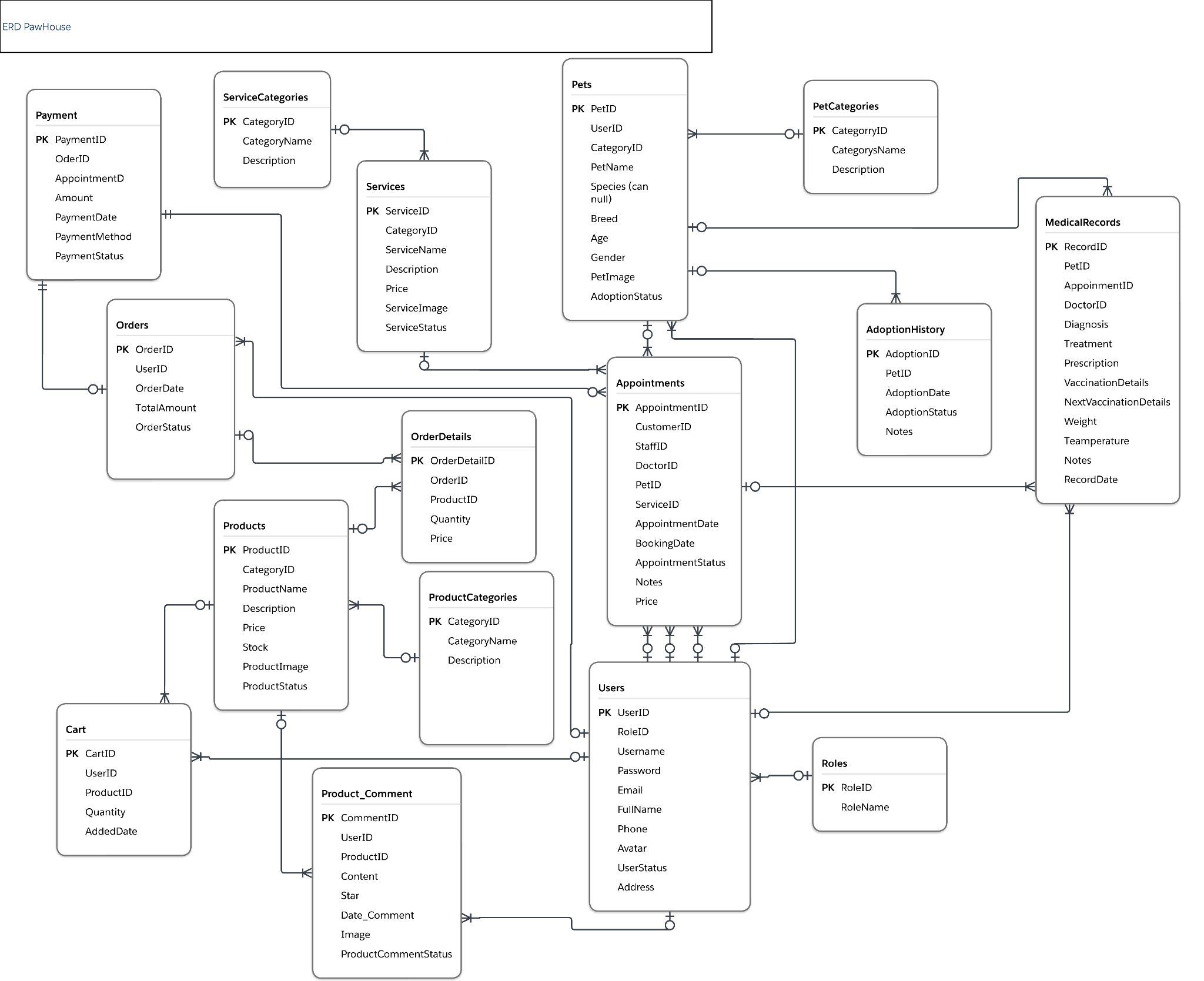
## 2. Package Descriptions

**Table 1: Package Description**

| **#** | **Package name** | **Description** |
| --- | --- | --- |
| 1 | view | This package include font-end source with: jsp page, css,  JavaScript, images, icon, font, bootstrap and jquery library. |
| 2 | controller | Includes Servlets with a responsibility to process, transfer the data  collected by user on web page(view) to DAO and Model in the  database to JSP pages. |
| 3 | DAO | Includes class ProductDAO, CartDAO, OrderDAO, UserDAO, and  etc with detailed functions that can interact directly with the  database for data processing to which the Controller can load data  for the user. |
| 4 | Ultils | Contains a class DBContext to connect with the database to load  and transfer data with DAO package by a connection with a  specified path (port, username, password, instance, database name)  is created. |
| 5 | constants | This package contains a class Iconstant.java file is a class that contains Google OAuth 2.0 constants used for the user authentication process with Google. It helps with easy maintenance, avoids code duplication, and enhances security (if used correctly). |
| 6 | Model | This package contains entities (classes) Product, Cart, Order,  OrderDetail, User, Account and Services, etc. That can be accessed  and used by other classes DAO. |

## 3. Database Design

### 3.1 Database schema



### 3.2. Table Descriptions

| **No** | **Table** | **Description** |
| --- | --- | --- |
| **01** | **Users** | **Users Information** - Primary keys: UserID (int) - Foreign keys: RoleID (int) - Unique: Username: (nvarchar(50)) - Password: nvarchar(100) - Email: nvarchar(100) - FullName: nvarchar(100) - Phone: nvarchar(20) - Avatar: nvarchar(255) - Status: bit  - Address: nvarchar(255) |
| **02** | **AdoptionHistory** | **AdoptionHistory Information**  - Primary keys: AdoptionID (int) - Foreign keys: PetID (int) - AdoptionDate: datetime - AdoptionStatus: nvarchar(20) - Notes: nvarchar(500) |
| **03** | **Appointments** | **Appointments Information** - Primary keys: AppointmentID (int) - Foreign keys: CustomerID (int) - Foreign keys: StaffID (int) - Foreign keys: DoctorID (int)  - Foreign keys: ServiceID (int)  - AppointmentDate: datetime - BookingDate: datetime - Price: decimal(18,2)  - AppointmentStatus: nvarchar(20) - Notes: nvarchar(500 ) |
| **04** | **Cart** | **Cart Information** - Primary keys: CartID (int) - Foreign keys: UserID (int) - Foreign keys: ProductID (int) - Quantity: int - AddedDate: datetime |
| **05** | **OrderDetails** | **OrderDetails Information** - Primary keys: OrderDetailID (int) - Foreign keys: OrderID (int) - Foreign keys: ProductID (int) - Quantity: int - Price: decimal(18,2) |
| **06** | **Orders** | **Orders Information** - Primary keys: OrderID (int) - Foreign keys: UserID (int) - OrderDate: datetime - TotalAmount:decimal(18,2) - OrderStatus: nvarchar(50) |
| **07** | **Pets** | **Pets Information** - Primary keys: PetID (int)  - Foreign keys: CategoryID (int)  - Foreign keys: UserID (int) - PetName: nvarchar(100)  - Species: nvarchar(50)  - Breed: nvarchar(50)  - Age: int  - Gender: nvarchar(10)  - PetImage: nvarchar(255)  - AdoptionStatus: nvarchar(50) |
| **08** | **Products** | **Products Information** - Primary keys: ProductID (int) - Foreign keys: CategoryID (int) - ProductName: nvarchar(100) - Description: nvarchar(255) - Price: decimal(18,2) - Stock: int - ProductImage: nvarchar(255)  - ProductStatus: bit |
| **09** | **Roles** | **Roles Information** - Primary keys: RoleID (int) - RoleName: nvarchar(50) |
| **10** | **Services** | **Services Information** - Primary keys: ServiceID (int)  - Foreign keys: CategoryID (int)  - ServiceName: nvarchar(100)  - Description: nvarchar(255)  - Price: decimal(18,2)  - ServiceImage: nvarchar(255)  - ServiceStatus: bit |
| **11** | **Service Categories** | **ServiceCategories Information** - Primary keys: CategoryID (int) - CategoryName: nvarchar(100) - Description: nvarchar(255) |
| **12** | **Payment** | **Payment Information** - Primary keys: PaymentID (int) - Foreign keys: OrderID (int) - Foreign keys: AppointmentID (int) - Amount: decimal(18,2) - PaymentDate: datetime - PaymentMethod: nvarchar(50) - PaymentStatus: nvarchar(50) |
| **13** | **Pet Categories** | **PetCategories Information** - Primary keys: CategoryID (int) - CategoryName: nvarchar(100) - Description: nvarchar(255) |
| **14** | **Product Categories** | **ProductCategories Information**  - Primary keys: CategoryID (int) - CategoryName: nvarchar(100) - Description: nvarchar(255) |
| **15** | **Medical Records** | **Medical Records Information**  - Primary keys: RecordID (int)  **-** Foreign keys: PetID (int)  **-** Foreign keys: AppointmentID (int)  **-** Foreign keys: DoctorID (int)  **-** Diagnosis: nvarchar(500)  - Treatment: nvarchar(500)  - Prescription: nvarchar(500)  - VaccinationDetails: nvarchar(500)  - NextVaccinationDate datetime  - Weight; decimal(5, 2)  - Temperature; decimal(4, 1)  - Notes; nvarchar(1000)  - RecordDate: datetime |
| **16** | **Product Comment** | **- Primary keys: CommentID** (int)  **-** Foreign keys: UserID (int)  **-** Foreign keys: ProductID (int)  - Star (int)  - Content: nvarchar(500)  - Date\_Comment: datetime  - Image: nvarchar(255)  - ProductCommentStatus: bit |

# 

# 

# 

# 

### 

### 

# 

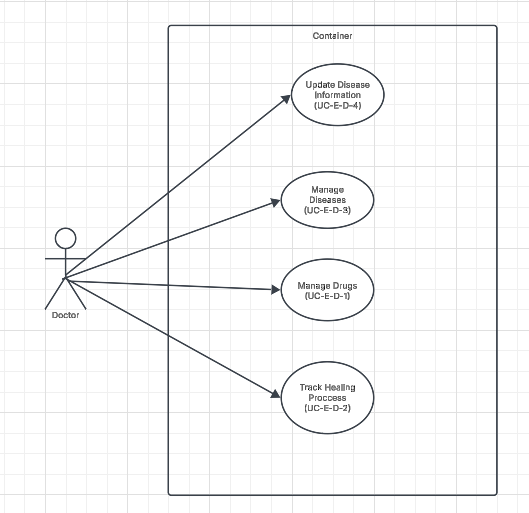
# **II. Use Case Diagram**

## 

## Figure 1.2.1: Use Case Guest

## Figure 1.2.2: Use Case Customer

## Figure 1.2.3: Use Case Staff



## Figure 1.2.4: Use Case Doctor

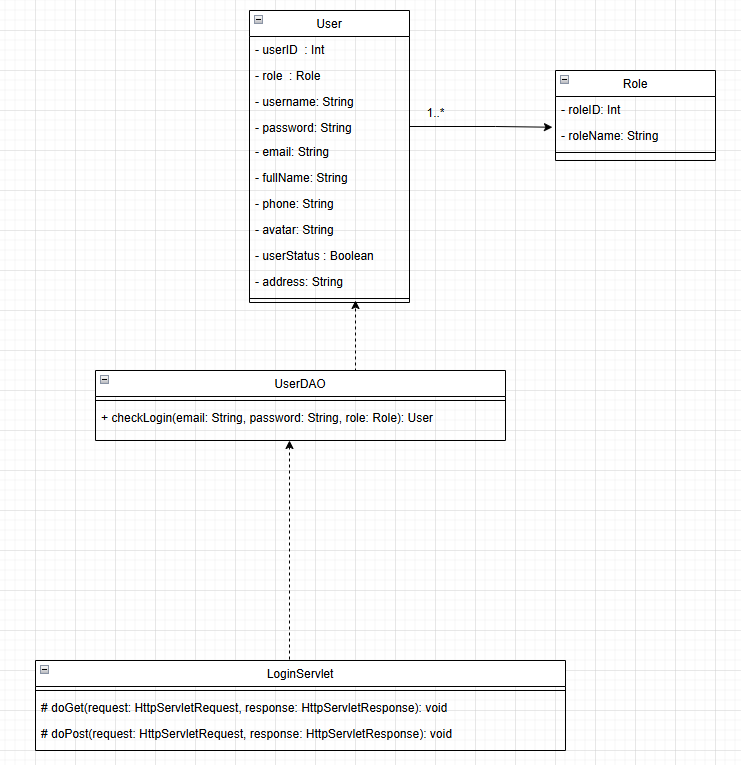


## Figure 1.2.5: Use Case Admin

# **III. Code Designs**

## 1. Login:

### A.Class Diagram



### B.Class Specifications

| # | Method | Description |
| --- | --- | --- |
| 1 | authenticate(String username, String password) | The method receives an authentication request to create an authentication object, authorize the user, and use JWT encryption to securely store user credentials. |

#### User repostitory

| # | Method | Description |
| --- | --- | --- |
| 1 | checkLogin(String username, String password) | The method checks whether the user exists and returns the corresponding result to determine the user's role and whether they can log in. |

#### EmployeeRepository

| # | Method | Description |
| --- | --- | --- |
| 1 | findEmployeeByUsername(String username) | The function takes a username parameter and uses JPA to look for the employee, returning an employee object or null if not found. |

#### CustomerRepostitory

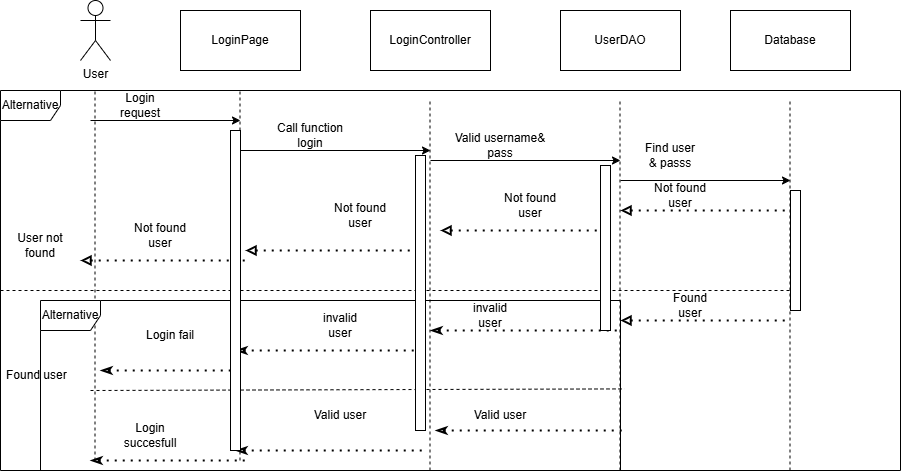
| # | Method | Description |
| --- | --- | --- |
| 1 | findCustomerByUsername(String username) | The function takes a username parameter and uses JPA to look for the customer, returning a customer object or null if not found. |

#### AdminRepository

| # | Method | Description |
| --- | --- | --- |
| 1 | findAdminByUsername(String username) | The function takes a username parameter and uses JPA to look for the admin, returning an admin object or null if not found. |

### 

### C.Sequence Diagram(s)



### D.Database Queries

SELECT \* FROM Users WHERE Email = ? AND Password = ?

## 

## 

## 

## 

## 

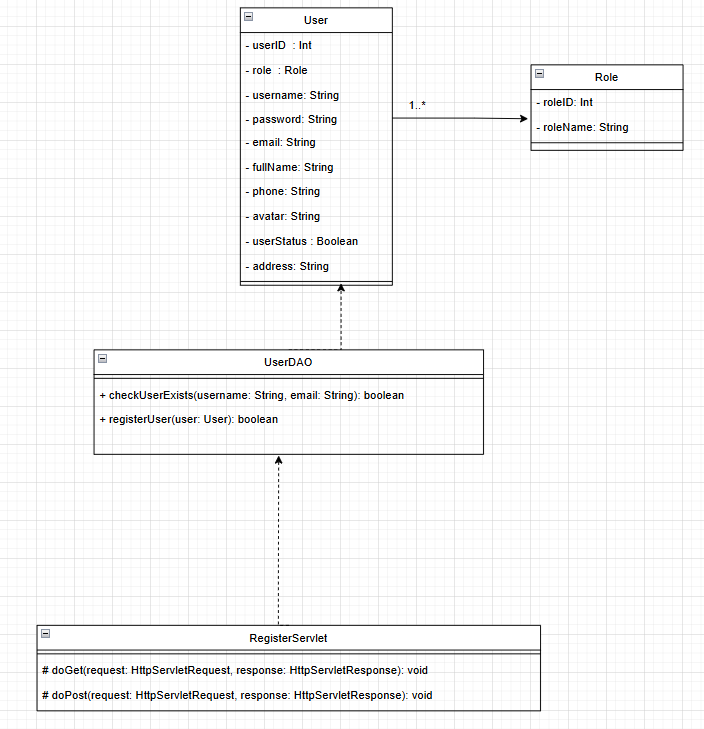
## 

## 

## 

## 2. Register:

### Class Diagram



### Class Specifications

# 

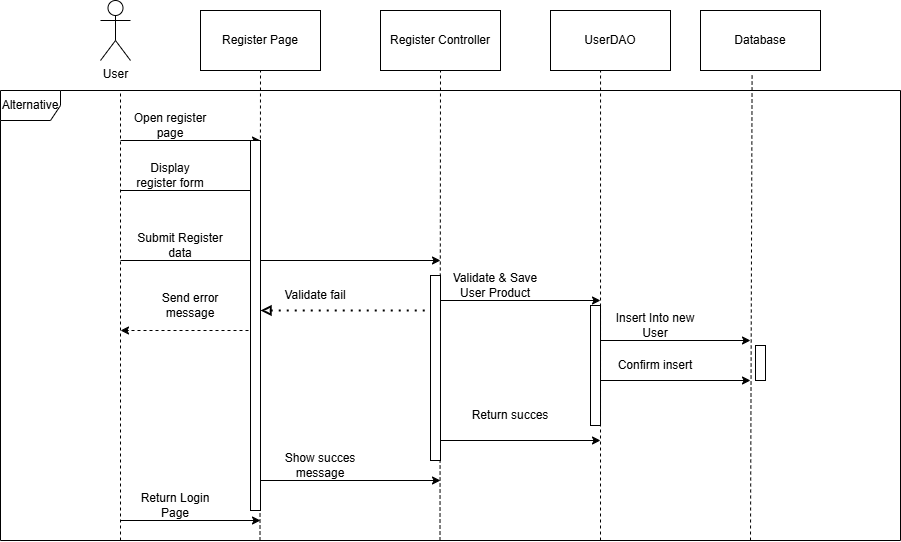
#### UserDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | checkUserExists(String username, String email) | Checks if a user with the given username or email already exists in the database, returning true if found. |
| 2 | registerUser(User user) | Registers a new user by inserting the user’s details into the database, returning true if successful. |

#### RegisterServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests by rendering the registration page for new users. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for processing user registration, validating data, and storing user details in the database. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO Users (Username, Password, Email, FullName, Phone, UserStatus, RoleID)

VALUES (?, ?, ?, ?, ?, ?, 2)

## 

## 

## 

## 

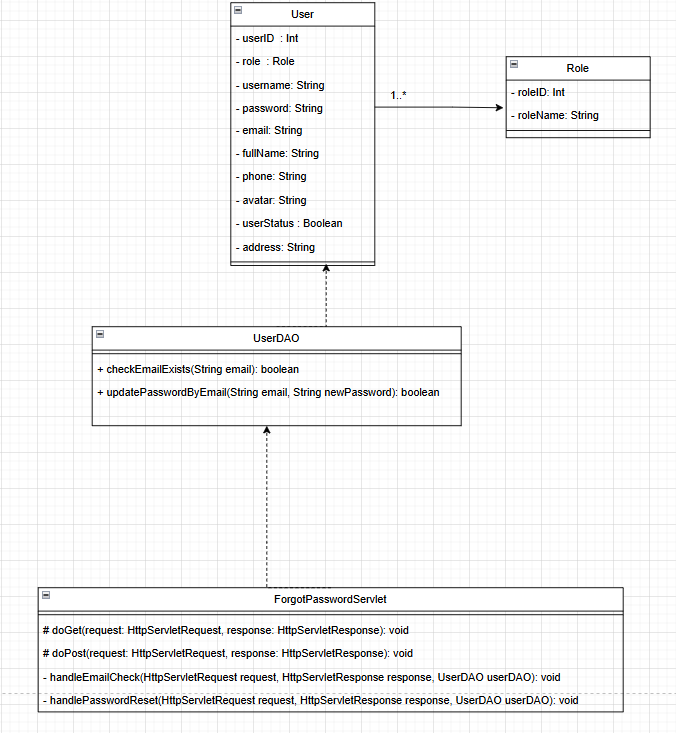
## 

## 

## 

## 3. Forget:

### Class Diagram



### Class Specifications

# 

#### UserDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | checkEmailExists(String email) | Checks if a user with the given email exists in the database, returning true if found. |
| 2 | updatePasswordByEmail(String email, String newPassword) | Updates the user's password based on the given email, returning true if successful. |

#### ForgotPasswordServlet

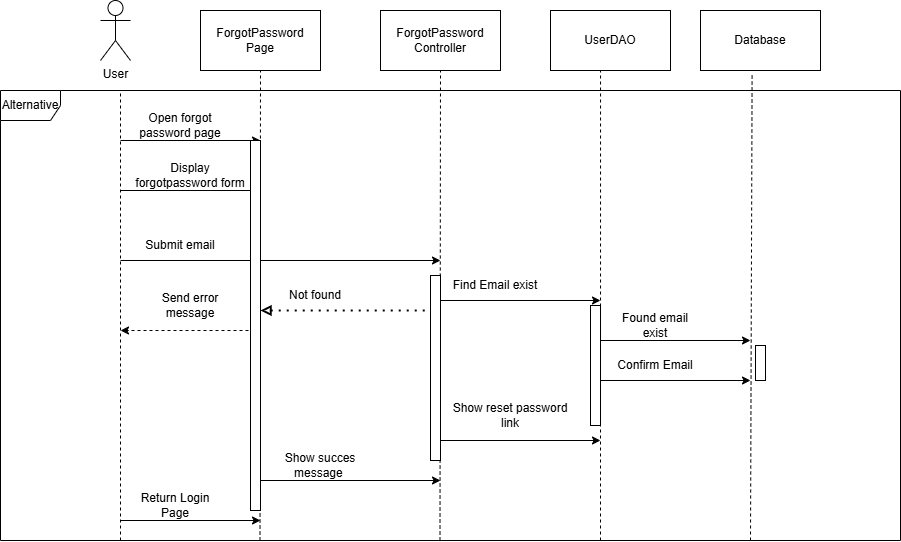
| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests by rendering the forgot password page. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests to process password reset requests, validate email, and send reset instructions. |
| 3 | handlePasswordReset(HttpServletRequest request, HttpServletResponse response, UserDAO userDAO) | Processes the password reset request, verifies user information, and updates the password if valid. |

## 

## 

## 

### Sequence Diagram(s)



### Database Queries

-UPDATE Users SET ResetToken = ?, ResetTokenExpiry = DATEADD(HOUR, 24, GETDATE()) WHERE Email = ?

-UPDATE Users SET Password = ?, ResetToken = NULL, ResetTokenExpiry = NULL "

+ "WHERE ResetToken = ? AND ResetTokenExpiry > GETDATE()

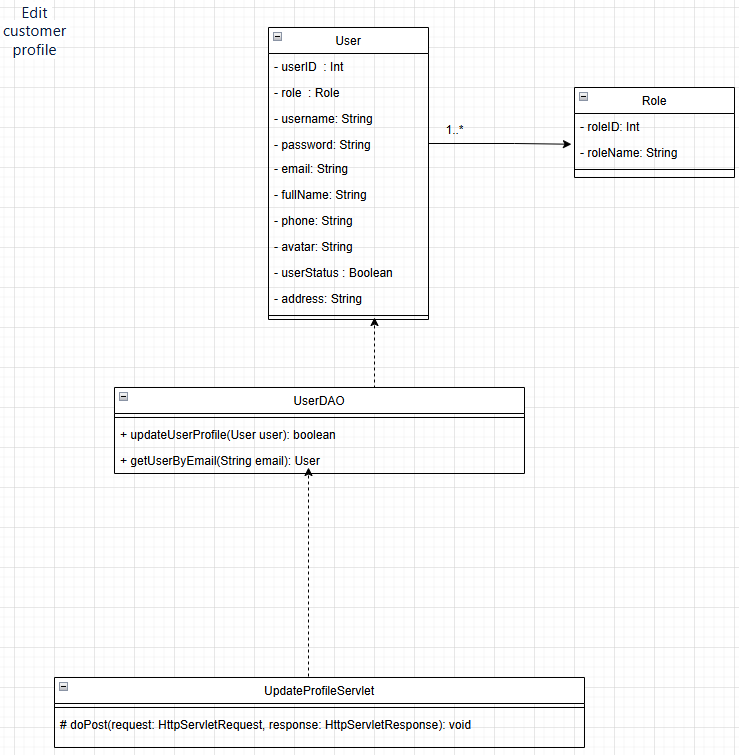
## 

## 

## 

## 4. Edit customer’s profile:

### Class Diagram



### Class Specifications

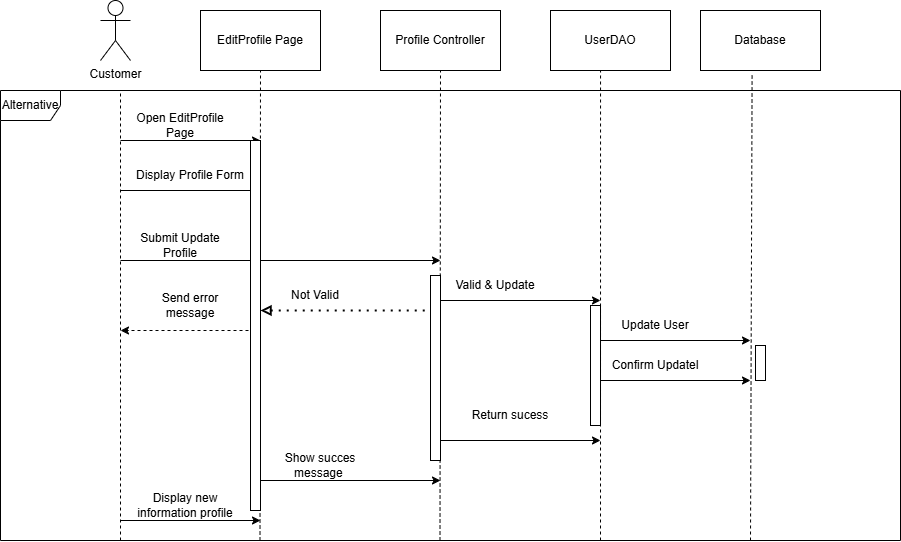
#### UserDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | updateUserProfile(User user) | Updates the user profile details (such as name, phone, address) and returns true if successful. |
| 2 | getUserByEmail(String email) | Retrieves the user details based on the given email, returning a User object if found. |

#### UpdateProfileServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating customer profile details. |

### Sequence Diagram(s)



### Database Queries

UPDATE Users SET Username = ?, Email = ?, FullName = ?, Phone = ?, Avatar = ? WHERE UserID = ?

## 

## 

## 

## 

## 

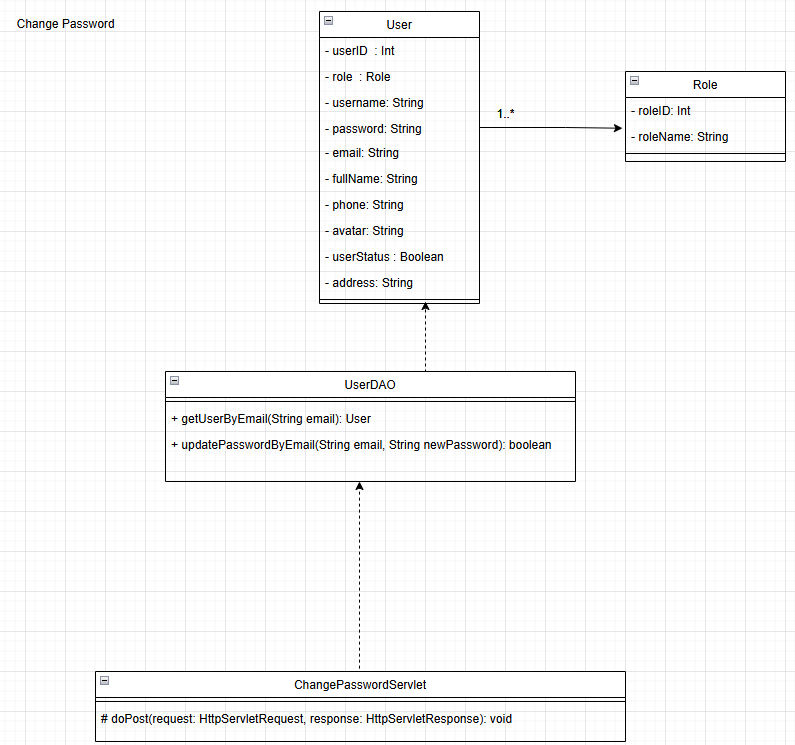
## 

## 

## 

## 5. Change pass:

### Class Diagram



### Class Specifications

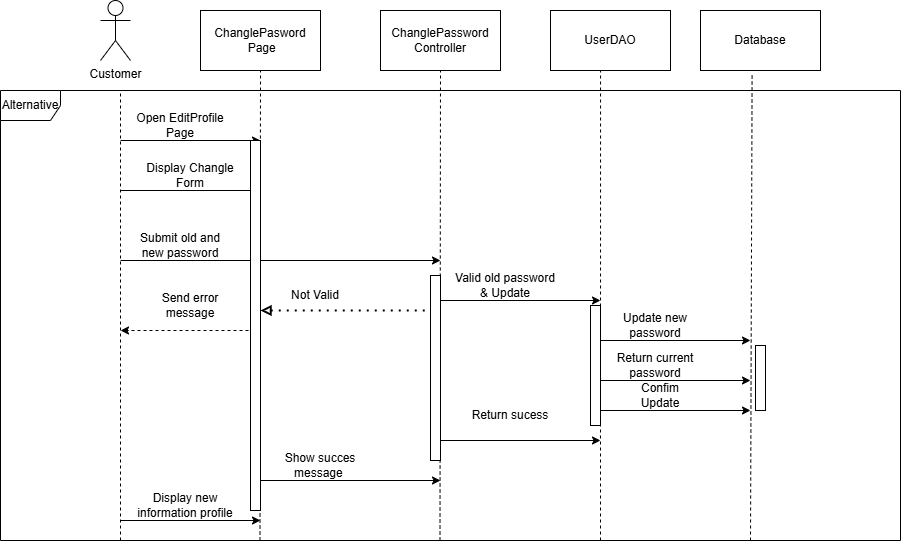
#### UserDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getUserByEmail(String email) | Retrieves the user details based on the given email, returning a User object if found. |
| 2 | updatePasswordByEmail(String email, String newPassword) | Updates the user's password based on the given email, returning true if successful. |

#### ChangePasswordServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating user password, validating current password, and applying changes. |

### Sequence Diagram(s)



### Database Queries

UPDATE Users SET Password = ? WHERE Email = ?

## 

## 

## 

## 

## 

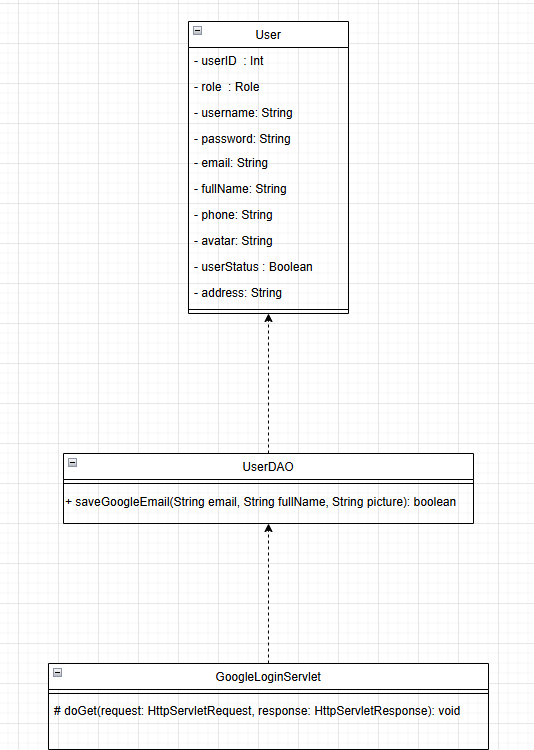
## 

## 

## 

## 6. Login with Google:

### Class Diagram



### Class Specifications

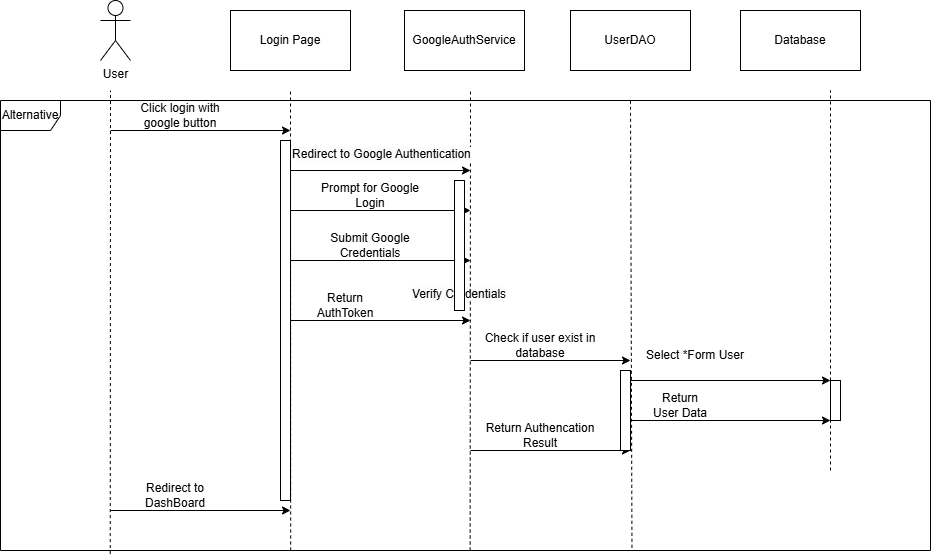
#### UserDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | saveGoogleEmail(String email, String fullName, String picture) | Stores Google account details (email, full name, and profile picture) in the database if the user does not exist, returning true if successful. |

#### GoogleLoginServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for Google OAuth login, retrieves user information from Google, and processes authentication. |

### Sequence Diagram(s)



### Database Queries

None

## 

## 

## 

## 

## 

## 

## 

## 7. Add service:

### Class Diagram



### Class Specifications

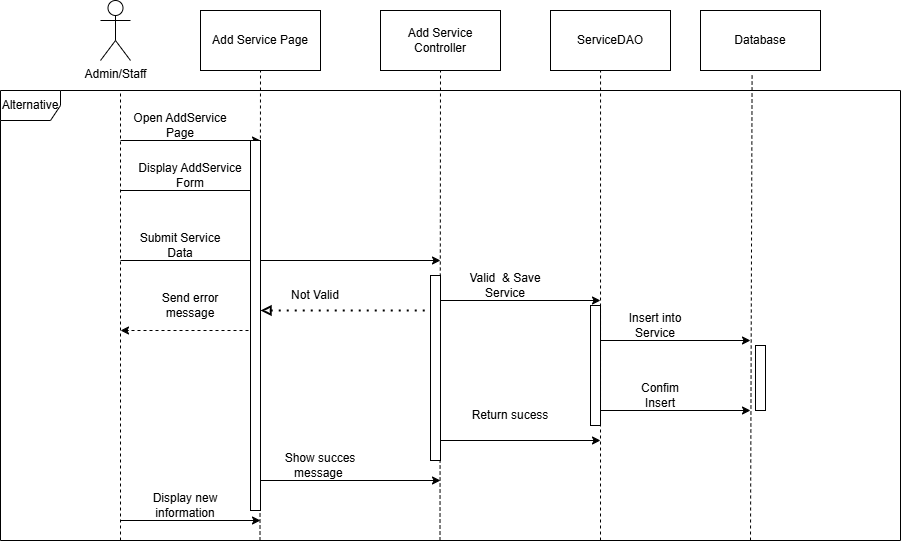
#### ServiceDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | insertService(Service service) | Inserts a new service into the database, storing service details such as name, category, description, price, and image. Returns true if successful. |

#### AddServiceServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding a new service, processes form data, and inserts the service into the database. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO Services (ServiceName, Description, Price, Duration, ServiceStatus) VALUES (?, ?, ?, ?, ?)

## 8. Update service:

### Class Diagram



### Class Specifications

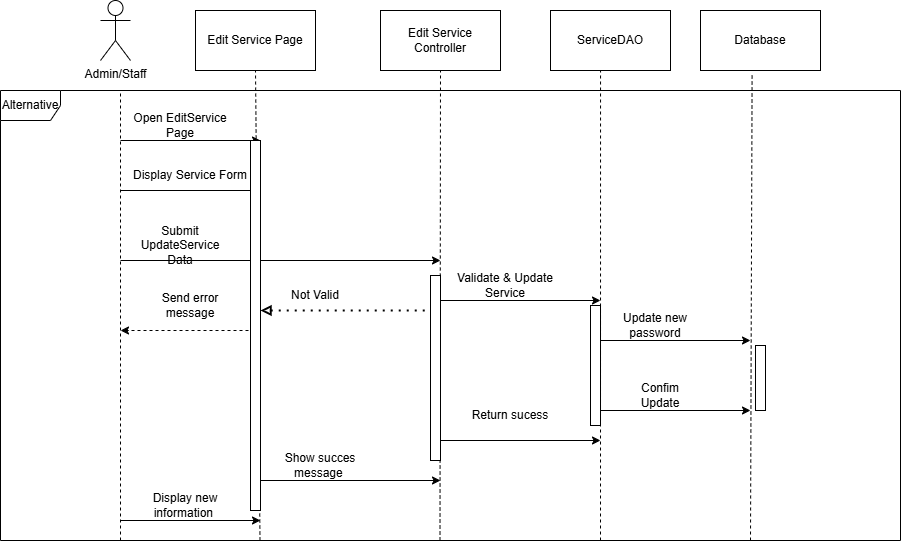
#### ServiceDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getServiceById(int serviceId) | Retrieves the service details based on the given service ID, returning a Service object if found. |
| 2 | updateServiceById(int serviceId, Service service) | Updates the service details in the database based on the given service ID, returning true if successful. |

#### AddServiceServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating an existing service, processes form data, and updates the service in the database. |

### Sequence Diagram(s)



### Database Queries

UPDATE Services SET ServiceName = ?, Description = ?, Price = ?, Duration = ?, ServiceStatus = ? WHERE ServiceID = ?

## 

## 

## 

## 

## 

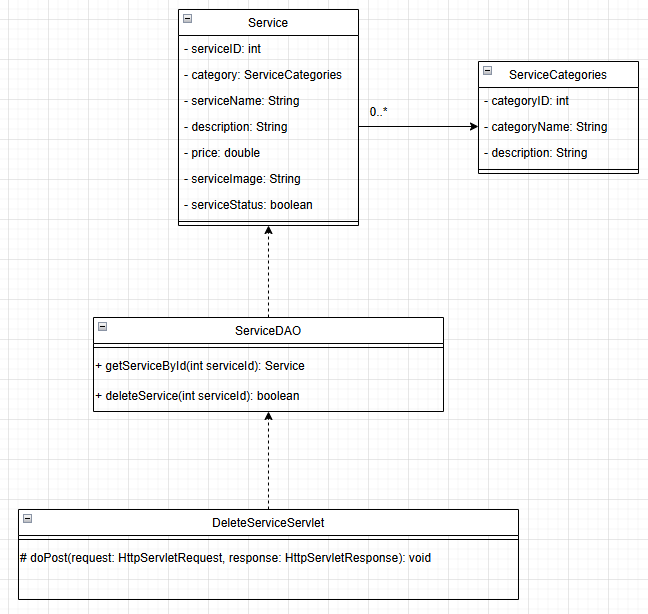
## 

## 

## 

## 9. Delete service:

### Class Diagram



### Class Specifications

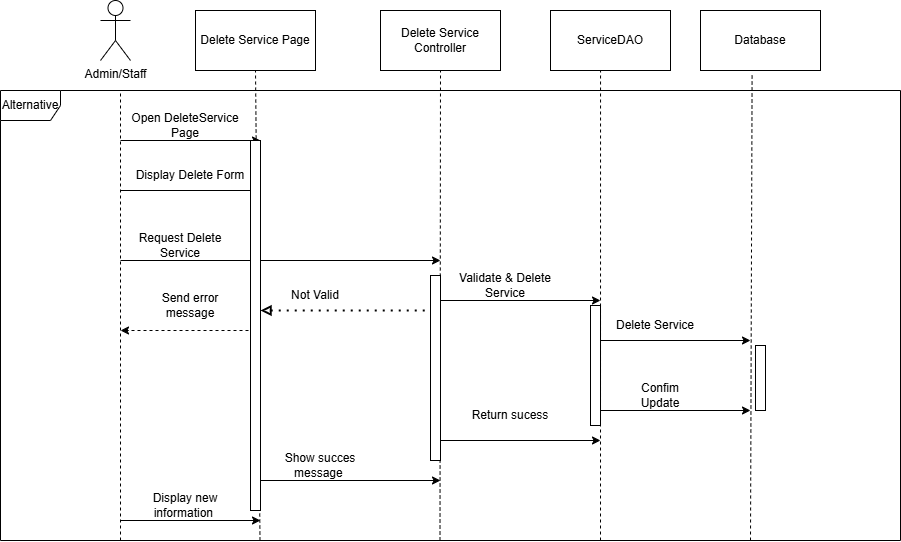
#### ServiceDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getServiceById(int serviceId) | Retrieves the service details based on the given service ID, returning a Service object if found. |
| 2 | deleteService(int serviceId) | Deletes the service from the database based on the given service ID, returning true if successful. |

#### DeleteServiceServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for deleting a service, retrieves service ID from request, and removes the service from the database. |

### Sequence Diagram(s)



### Database Queries

DELETE FROM Services WHERE ServiceID = ?

## 

## 

## 

## 

## 

## 

## 

## 

## 10. Add to cart:

### Class Diagram



### Class Specifications

#### CartDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | addToCart(int userID, int productID, int quantity) | Adds a product to the user's cart with the specified quantity, returning true if successful. |
| 2 | getCartByUser(int userID) | Retrieves all cart items associated with a specific user, returning a list of Cart objects. |

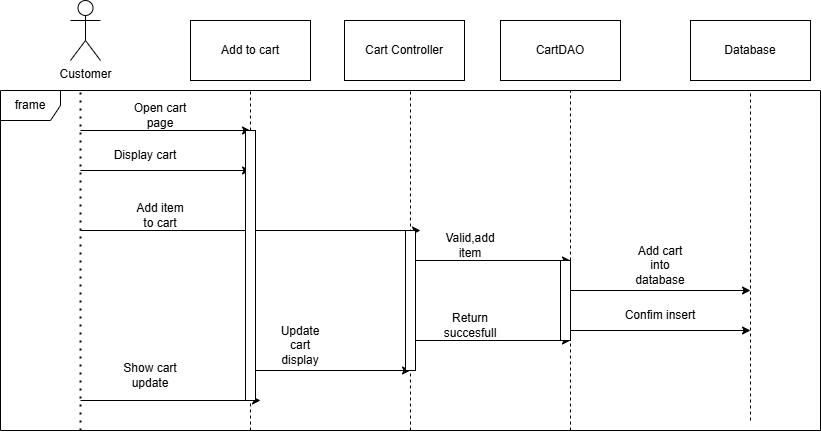
#### ProductDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getProductById(int productID) | Retrieves product details based on the given product ID, returning a Product object if found. |

#### CartServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for displaying the user's cart contents. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding a product to the cart, processing form data, and updating the cart in the database. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO Products(CategoryID, ProductName, Description, Price, Stock, ProductImage, ProductStatus) VALUES(?, ?, ?, ?, ?, ?, ?)

## 

## 

## 

## 

## 

## 

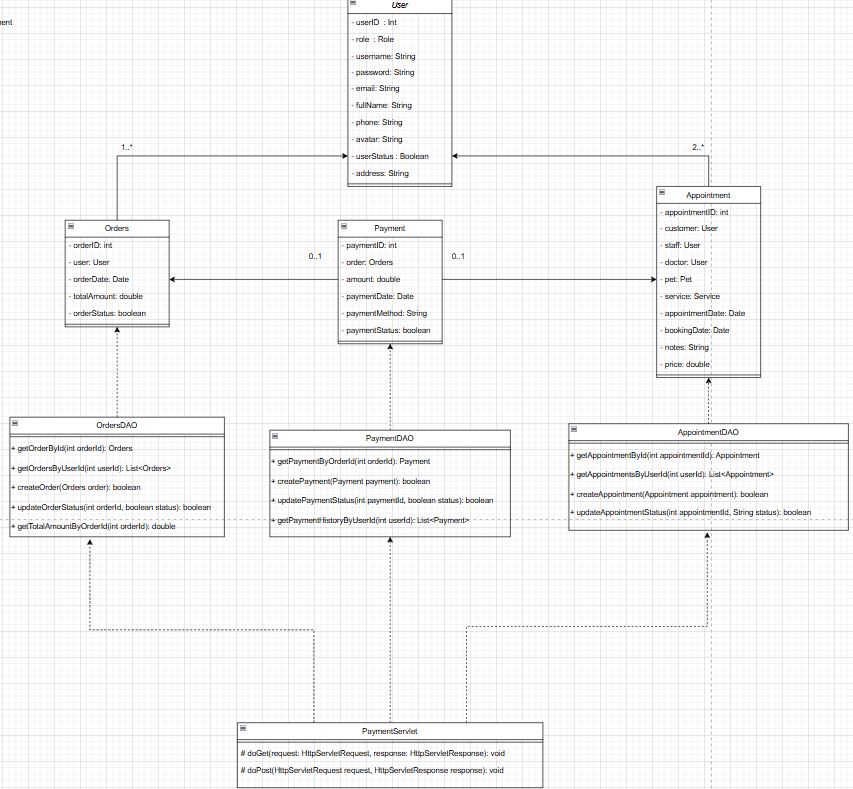
## 

## 

## 

## 11. Payment:

### Class Diagram



### Class Specifications

#### OrderDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getOrderById(int orderId) | Retrieves an order based on the order ID, returning an Orders object. |
| 2 | getOrdersByUserId(int userId) | Retrieves a list of orders made by a specific user. |
| 3 | createOrder(Orders order) | Creates a new order in the database, returning true if successful. |
| 4 | updateOrderStatus(int orderId, boolean status) | Updates the status of an order (e.g., completed or pending). |
| 5 | getTotalAmountByOrder(int orderId) | Calculates the total amount of a specific order. |

#### PaymentDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getPaymentByOrder(int orderId) | Retrieves a payment record linked to a specific order. |
| 2 | createPayment(Payment payment) | Creates a new payment record for an order. |
| 3 | updatePaymentStatus(int paymentId, boolean status) | Updates the status of a payment (e.g., successful or failed). |
| 4 | getPaymentHistoryByUserId(int userId) | Retrieves a list of past payments made by a user. |

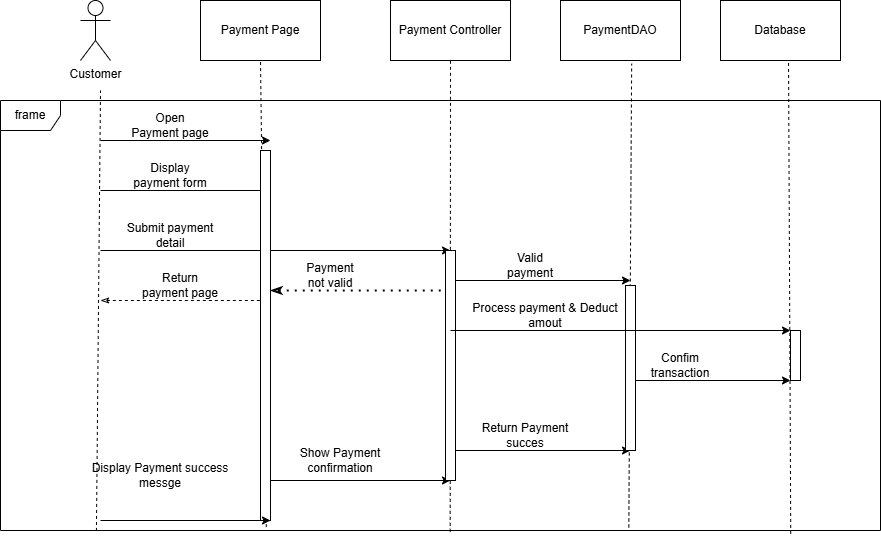
#### AppointmentDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAppointmentById(int appointmentId) | Retrieves an appointment based on the appointment ID. |
| 2 | getAppointmentsByUserId(int userId) | Retrieves a list of appointments booked by a specific user. |
| 3 | createAppointment(Appointment appointment) | Creates a new appointment record. |
| 4 | updateAppointmentStatus(int appointmentId, String status) | Updates the status of an appointment (e.g., confirmed, canceled). |

#### PaymentServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for processing payments. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving payment details. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO Payments (OrderID, PaymentMethod, Amount, PaymentDate, PaymentStatus) VALUES (?, ?, ?, GETDATE(), ?)

## 

## 

## 

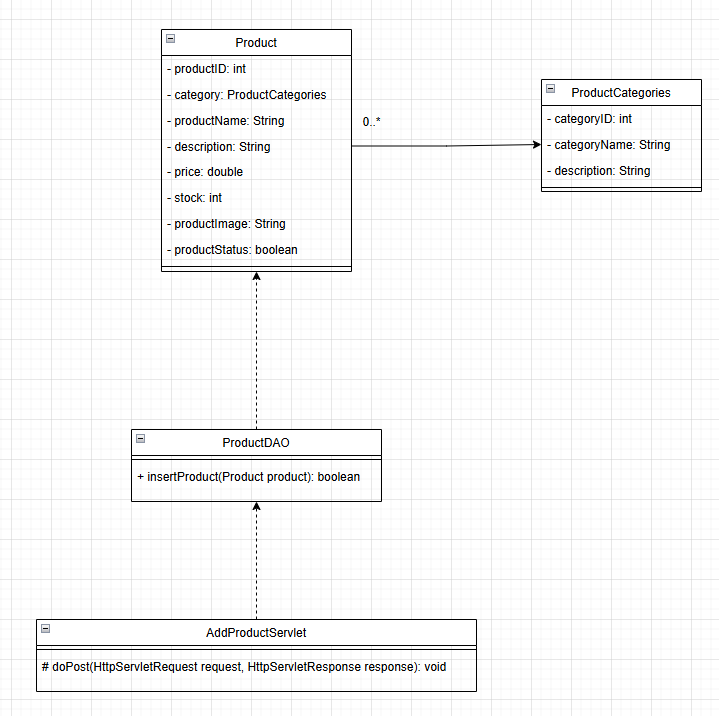
## 

## 

## 

## 12. Add product:

### Class Diagram



### Class Specifications

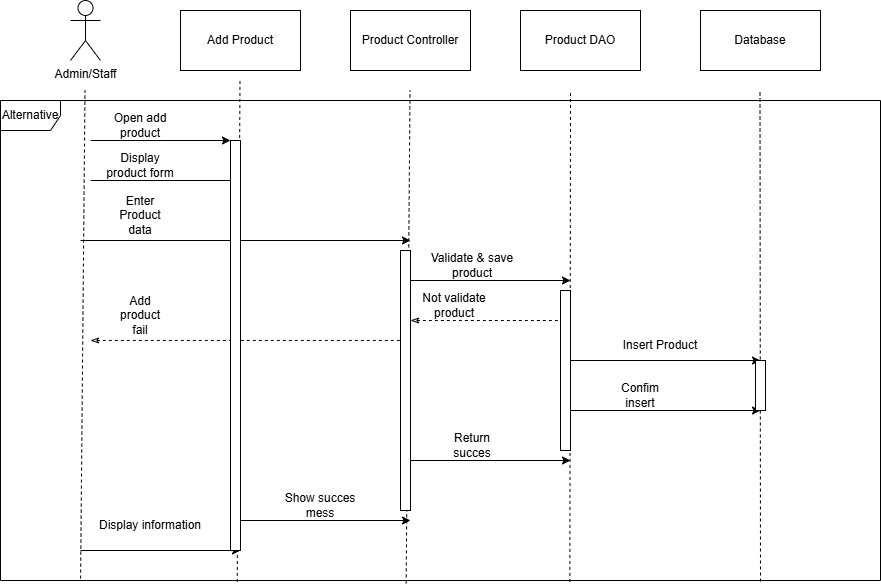
#### ProductDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | insertProduct(Product product) | Inserts a new product into the database, storing details such as name, category, description, price, stock, and image. Returns true if successful. |

#### AddProductServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding a new product, processes form data, and inserts the product into the database. |

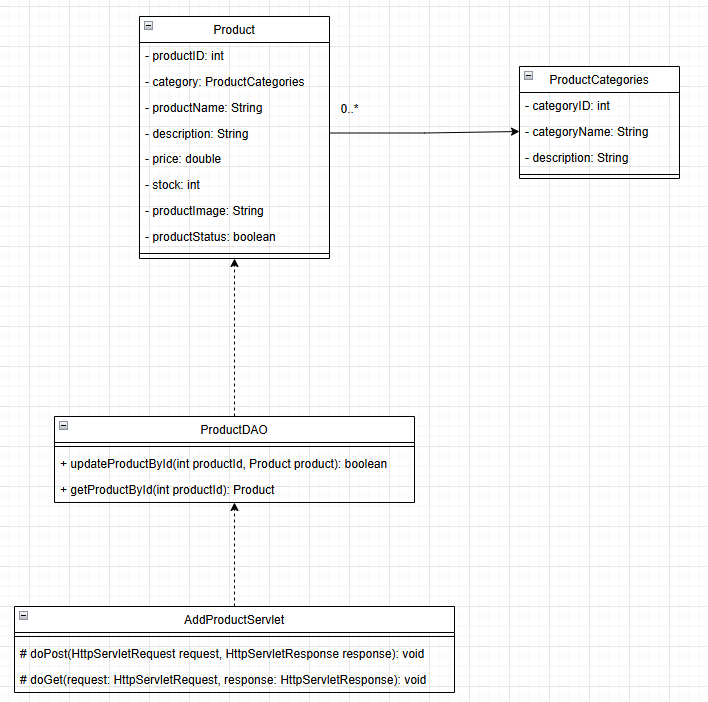
### Sequence Diagram(s)



### Database Queries

## 13. Edit Product:

### Class Diagram



### Class Specifications

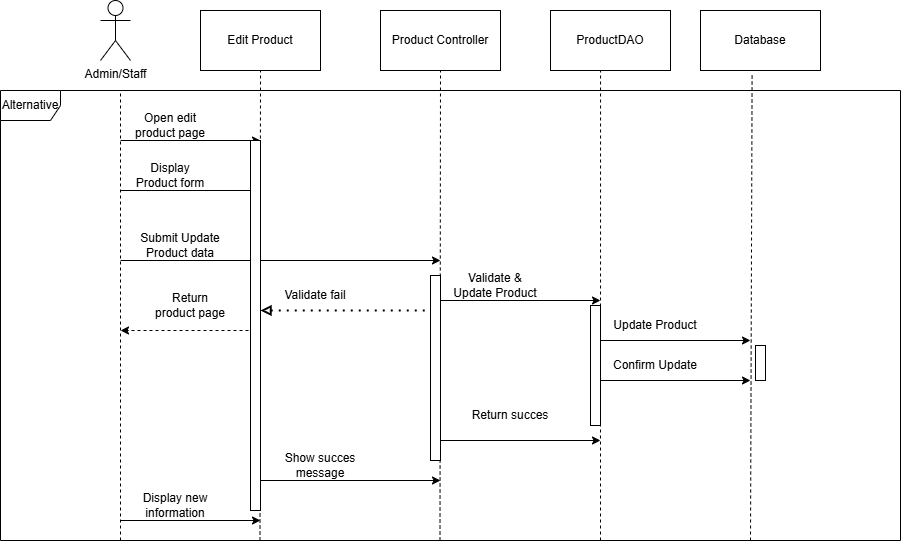
#### ProductDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | updateProductById(int productId, Product product) | Updates the product details in the database based on the given product ID, returning true if successful. |
| 2 | getProductById(int productId) | Retrieves product details based on the given product ID, returning a Product object if found. |

#### AddProductServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating an existing product, processes form data, and updates the product in the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying product details for editing. |

### Sequence Diagram(s)



### Database Queries

UPDATE Products SET CategoryID = ?, ProductName = ?, Description = ?, Price = ?, Stock = ?, ProductImage = ?, ProductStatus = ? WHERE ProductID = ?

## 

## 

## 

## 

## 

## 

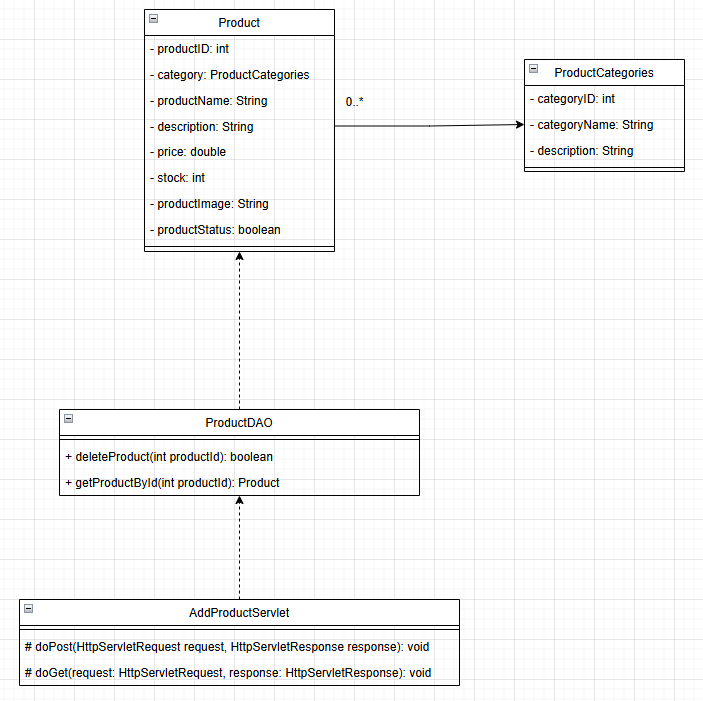
## 

## 

## 

## 14. Delete Product:

### Class Diagram



### Class Specifications

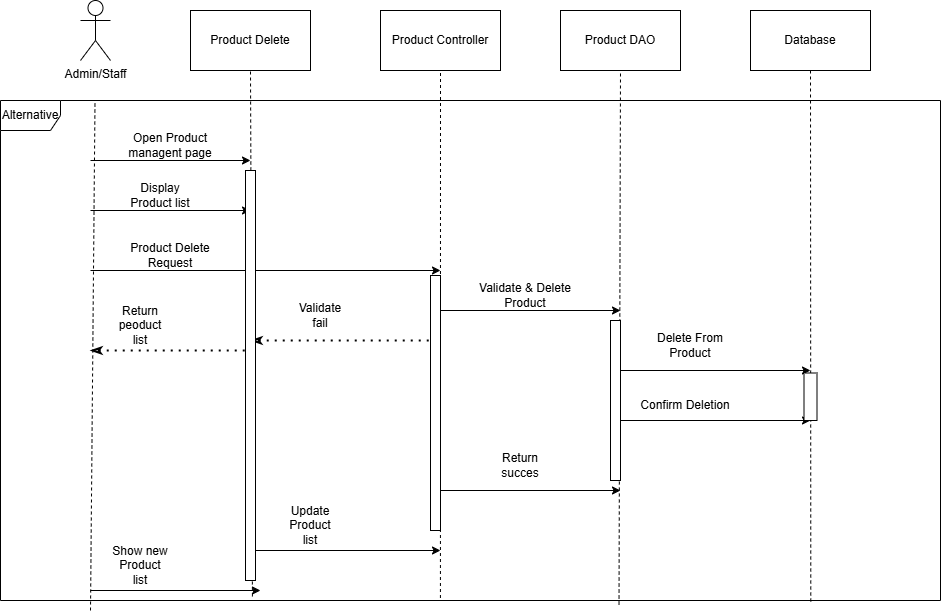
#### ProductDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | deleteProduct(int productId) | Deletes a product from the database based on the given product ID, returning true if successful. |
| 2 | getProductById(int productId) | Retrieves product details based on the given product ID, returning a Product object if found. |

#### AddProductServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for deleting a product, processes form data, and removes the product from the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying product details before deletion. |

### Sequence Diagram(s)



### Database Queries

DELETE FROM Products WHERE ProductID = ?

## 

## 

## 

## 

## 

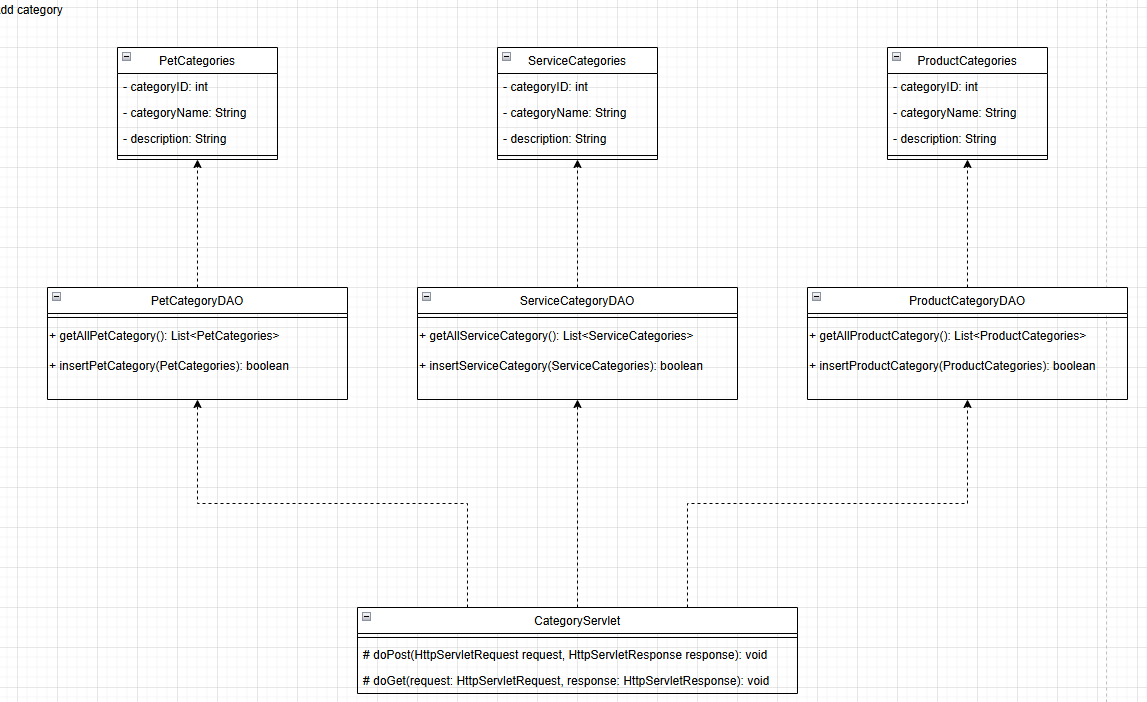
## 

## 

## 

## 15. Add Category:

### Class Diagram



### Class Specifications

#### PetCategoryDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllPetCategory() | Retrieves all pet categories from the database, returning a list of PetCategories objects. |
| 2 | insertPetCategory(PetCategories category) | Inserts a new pet category into the database, returning true if successful. |

#### ServiceCategoryDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllServiceCategory() | Retrieves all service categories from the database, returning a list of ServiceCategories objects. |
| 2 | insertServiceCategory(ServiceCategories category) | Inserts a new service category into the database, returning true if successful. |

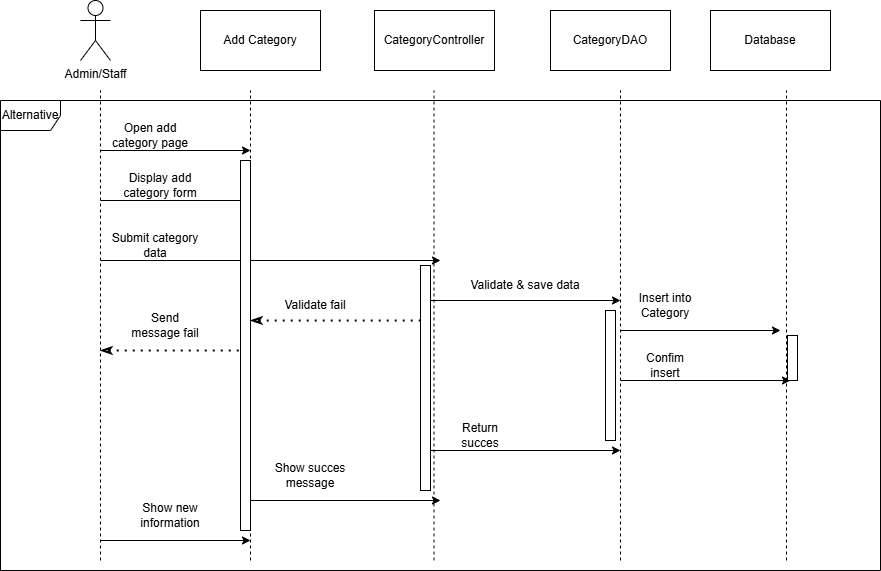
#### ProductCategoryDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllProductCategory() | Retrieves all product categories from the database, returning a list of ProductCategories objects. |
| 2 | insertProductCategory(ProductCategories category) | Inserts a new product category into the database, returning true if successful. |

#### CategoryServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding a new category, processing form data, and storing it in the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying category data. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO ProductCategories (CategoryName, Description) VALUES (?, ?)

## 

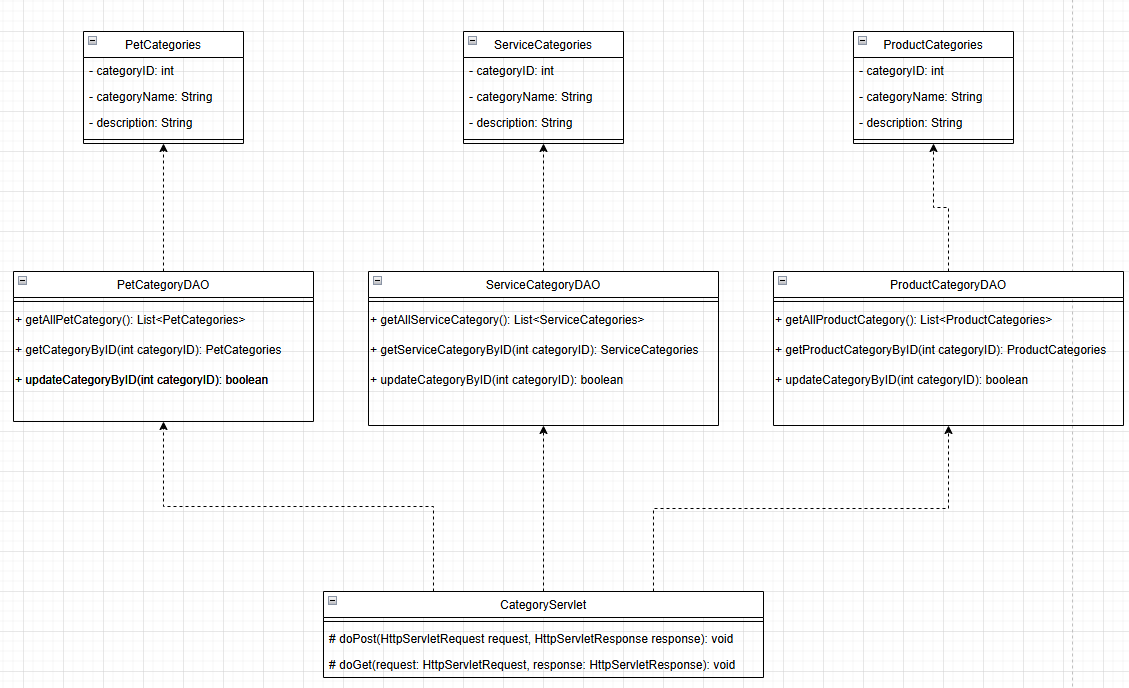
## 

## 

## 

## 16. Edit Category:

### Class Diagram



### Class Specifications

# 

#### PetCategoryDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllPetCategory() | Retrieves all pet categories from the database, returning a list of PetCategories objects. |
| 2 | getCategoryById(int categoryID) | Retrieves a specific pet category by its ID, returning a PetCategories object if found. |
| 3 | updateCategoryById(int categoryID) | Updates the details of a pet category based on the given ID, returning true if successful. |

#### ServiceCategoryDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllServiceCategory() | Retrieves all service categories from the database, returning a list of ServiceCategories objects. |
| 2 | getServiceCategoryById(int categoryID) | Retrieves a specific service category by its ID, returning a ServiceCategories object if found. |
| 3 | updateCategoryById(int categoryID) | Updates the details of a service category based on the given ID, returning true if successful. |

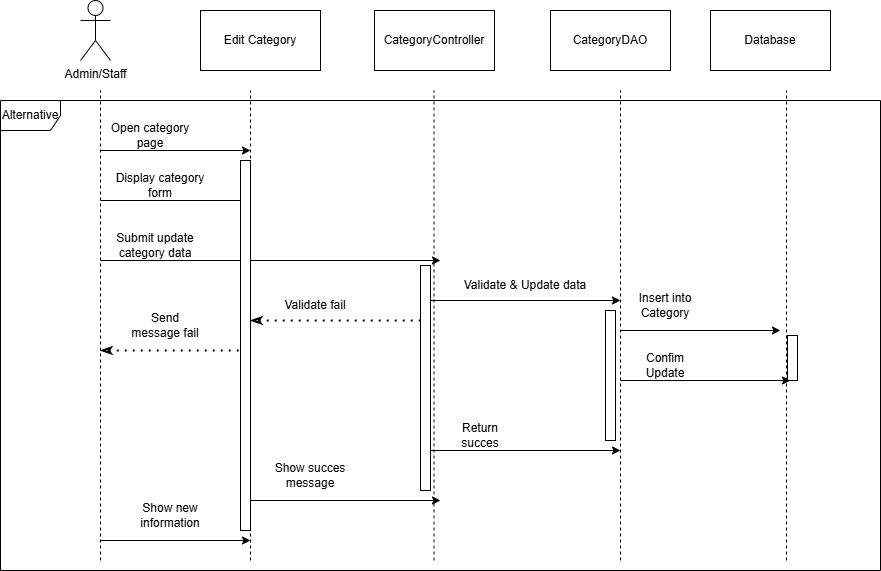
#### ProductCategoryDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllProductCategory() | Retrieves all product categories from the database, returning a list of ProductCategories objects. |
| 2 | getProductCategoryById(int categoryID) | Retrieves a specific product category by its ID, returning a ProductCategories object if found. |
| 3 | updateCategoryById(int categoryID) | Updates the details of a product category based on the given ID, returning true if successful. |

#### CategoryServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating category details, processes form data, and updates the category in the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying category data before updating. |

### Sequence Diagram(s)



### Database Queries

UPDATE ProductCategories SET CategoryName = ?, Description = ? WHERE CategoryID = ?

## 

## 

## 

## 

## 

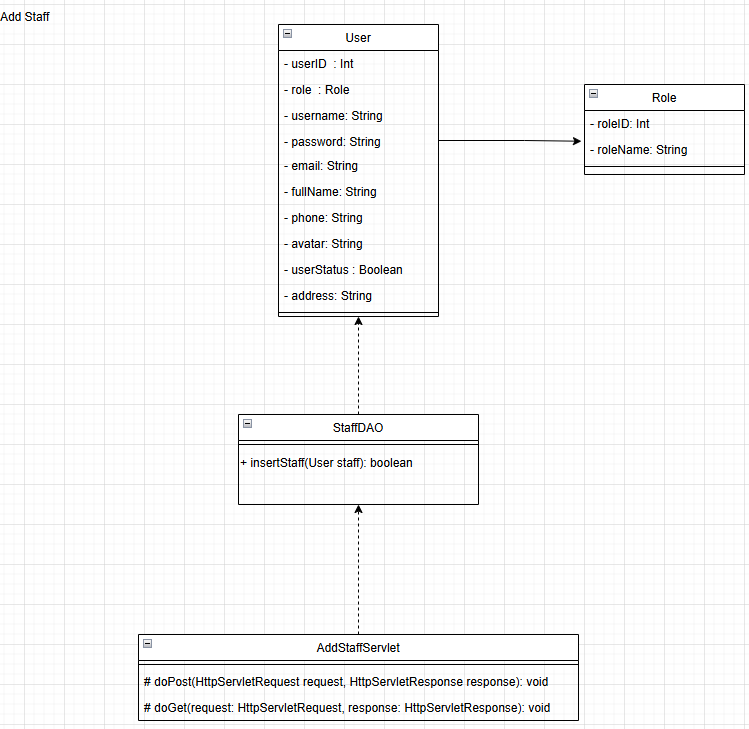
## 

## 

## 

## 17. Add Staff:

### Class Diagram



### Class Specifications

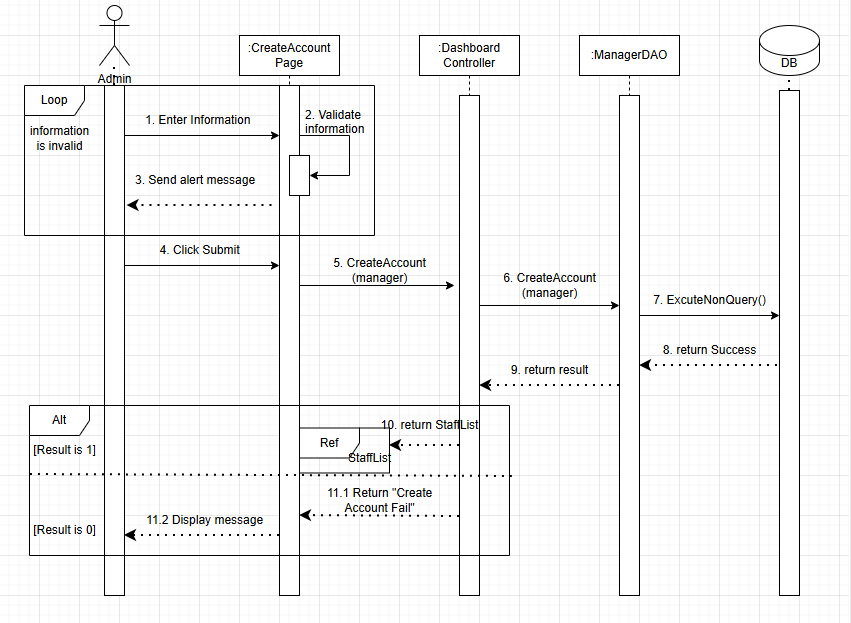
#### StaffDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | insertStaff(User staff) | Inserts a new staff member into the database, storing user details such as role, username, email, phone, and status. Returns true if successful. |

#### AddStaffServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding a new staff member, processes form data, and inserts the staff into the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying staff details before adding a new staff member. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO Users (Username, Password, Email, FullName, Phone, Avatar, UserStatus, RoleID) VALUES (?, ?, ?, ?, ?, ?, ?, 4)

## 

## 

## 

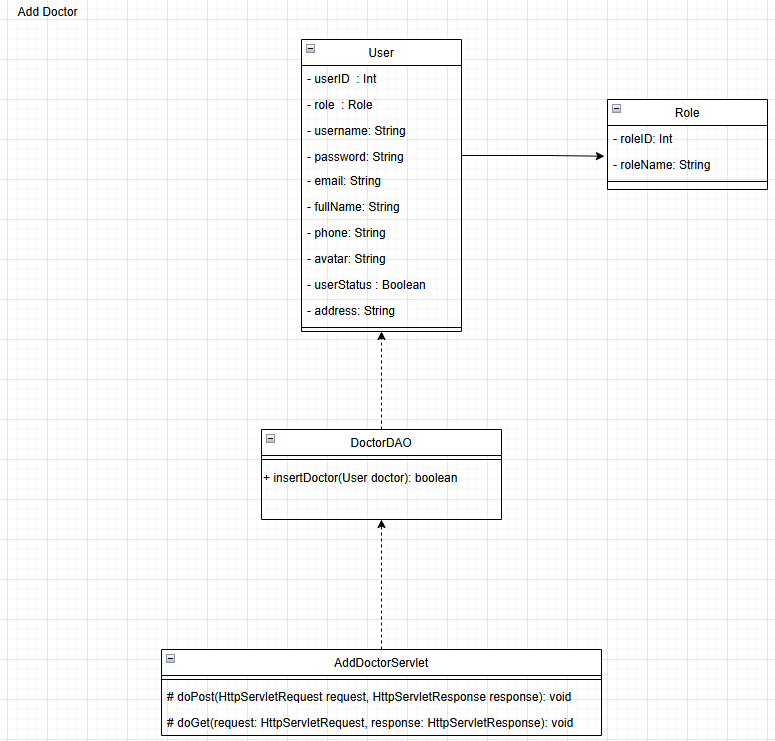
## 

## 

## 

## 18. Add Doctor:

### Class Diagram



### Class Specifications

## 

## 

## 

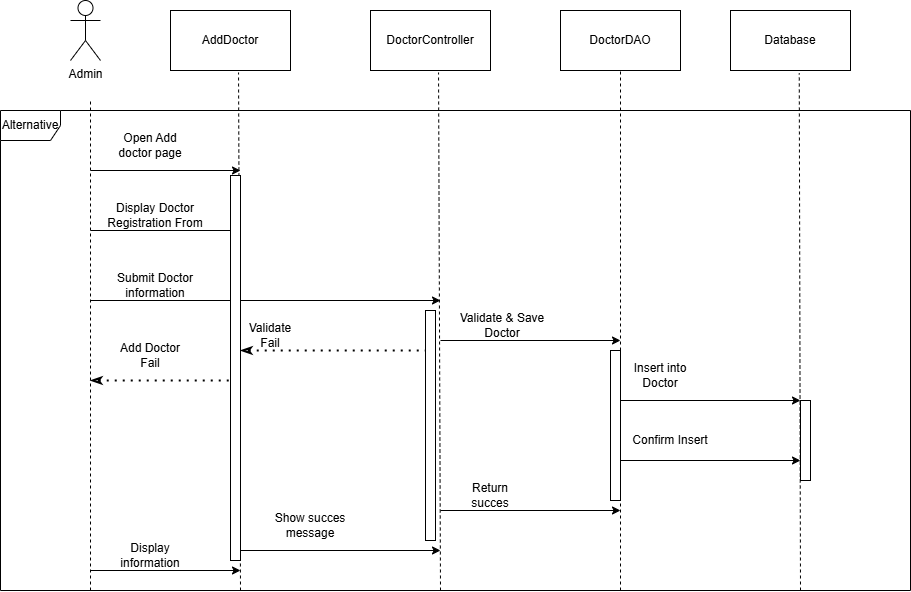
#### DoctorDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | insertDoctor(User doctor) | Inserts a new doctor into the database, storing user details such as role, username, email, phone, and status. Returns true if successful. |

#### AddDoctorServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding a new doctor, processes form data, and inserts the doctor into the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying doctor details before adding a new doctor. |

### Sequence Diagram(s)



### Database Queries

INSERT INTO Users (Username, Password, Email, FullName, Phone, Avatar, UserStatus, RoleID) VALUES (?, ?, ?, ?, ?, ?, ?, 3)

## 

## 

## 

## 

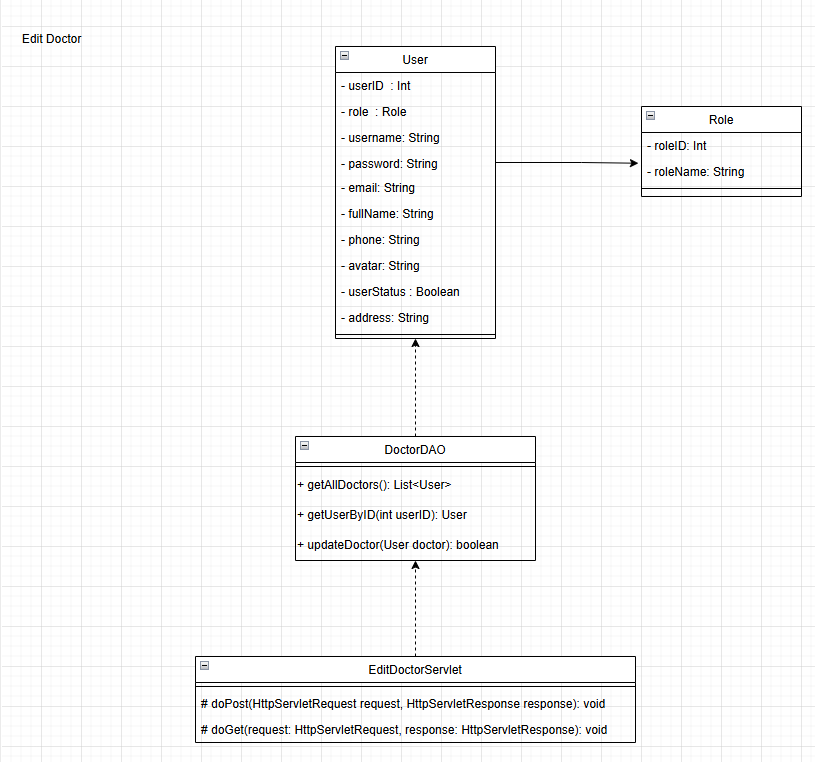
## 

## 

## 

## 19. Edit Doctor:

### Class Diagram



### Class Specifications

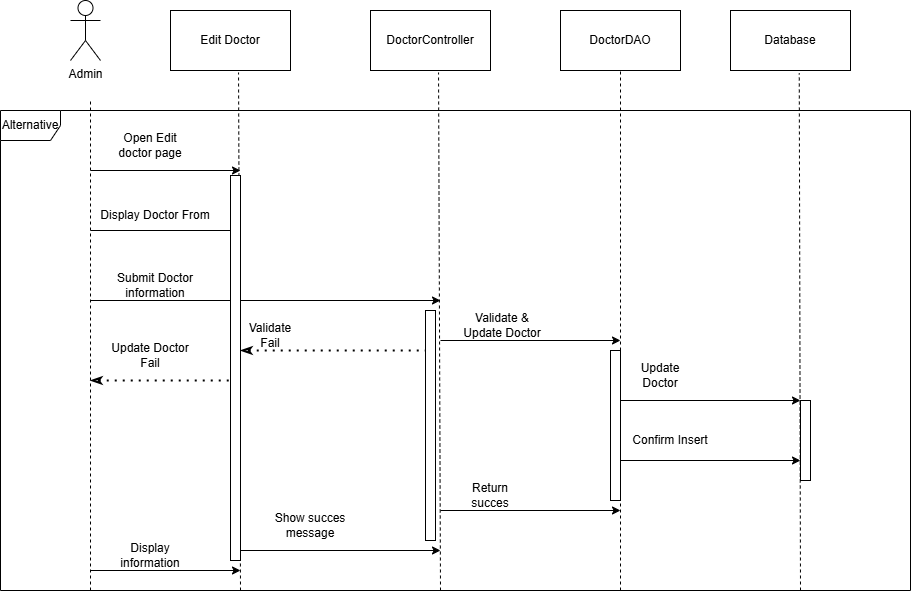
#### DoctorDAO

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllDoctors() | Retrieves all doctors from the database, returning a list of User objects. |
| 2 | getUserById(int userId) | Retrieves a doctor's details based on the given user ID, returning a User object if found. |
| 3 | updateDoctor(User doctor) | Updates the details of an existing doctor in the database, returning true if successful. |

#### EditDoctorServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating doctor details, processes form data, and updates the doctor's information in the database. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving and displaying doctor data before updating. |

### Sequence Diagram(s)



### Database Queries

UPDATE Users SET Username = ?, Email = ?, FullName = ?, Phone = ?, Avatar = ? WHERE UserID = ? AND RoleID = 3

## 

## 

## 

## 

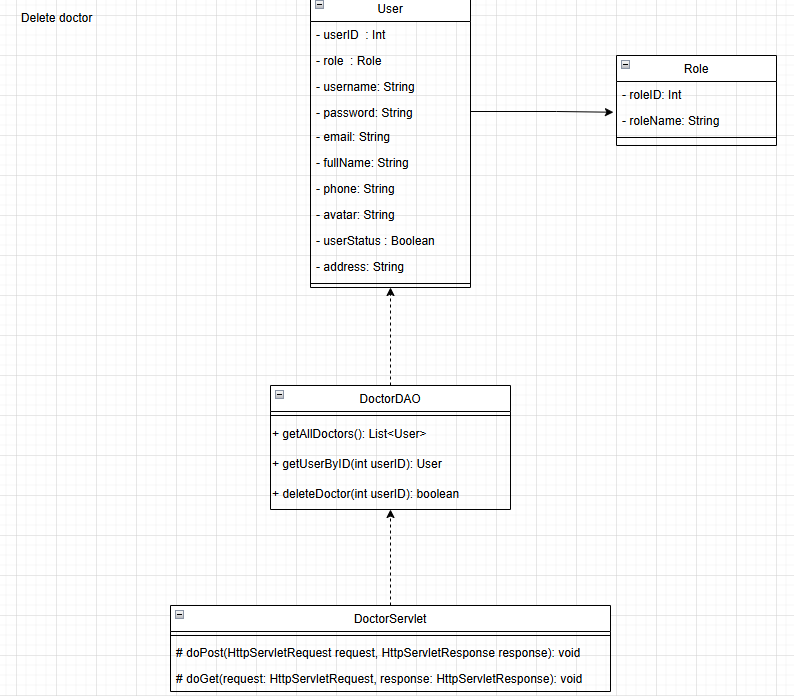
## 

## 

## 

## 20. Delete Doctor:

### Class Diagram



### Class Specifications

#### **DoctorDAO**

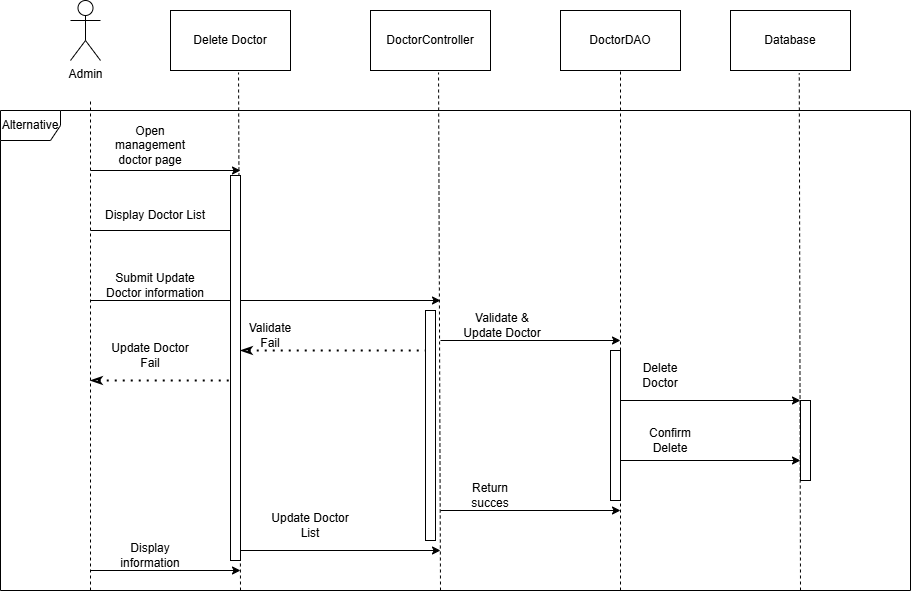
| # | Method | Description |
| --- | --- | --- |
| 1 | getAllDoctors() | Retrieves a list of all doctors in the system. |
| 2 | getUserByID(int userID) | Retrieves a doctor based on user ID. |
| 3 | deleteDoctor(int userID) | Deletes a doctor from the system based on user ID, returning true if successful. |

### 

#### **DoctorServlet**

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for deleting a doctor. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving doctor details before deletion. |

### Sequence Diagram(s)



### Database Queries

DELETE FROM Users WHERE UserID = ? AND RoleID = 3

## 

## 

## 

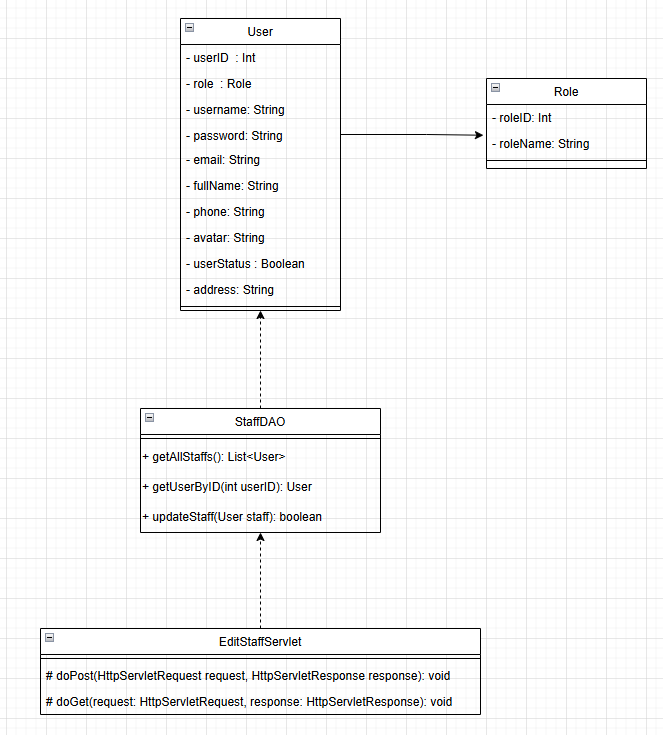
## 

## 

## 

## 21. Update Staff:

### Class Diagram



### Class Specifications

#### **StaffDAO**

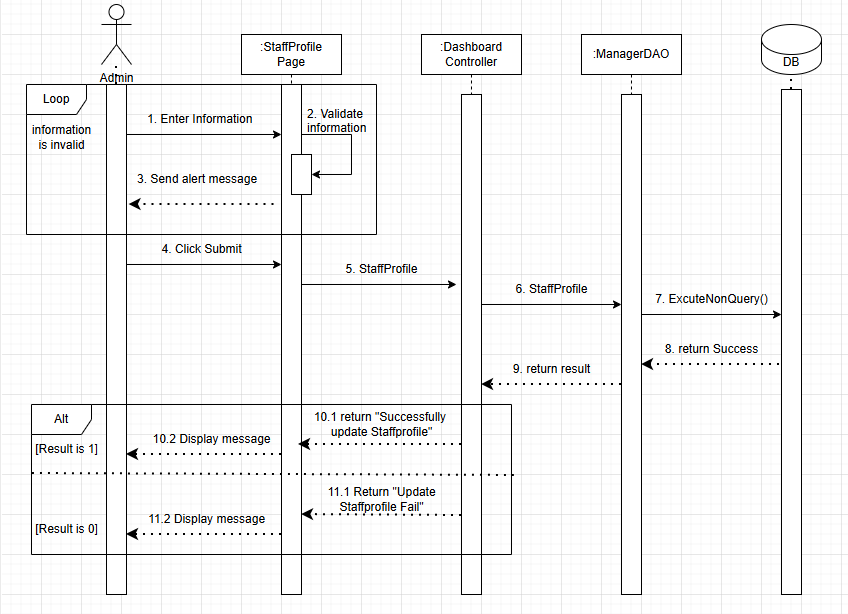
| # | Method | Desciption |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
| 1 | getAllStaffs() | Retrieves a list of all staff members in the system. |
| 2 | getUserByID(int userID) | Retrieves a staff member based on user ID. |
| 3 | updateStaff(User staff) | Updates the information of a staff member in the system. |

#### EditStaffServlet

| # | Method | Description |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating staff details. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving staff details before updating. |

### 

### Sequence Diagram(s)

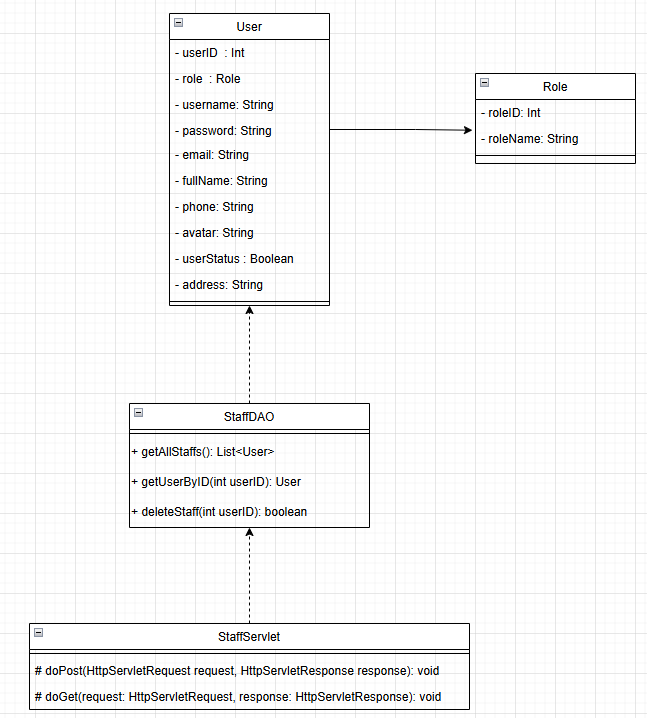


### Database Queries

UPDATE Users SET Username = ?, Email = ?, FullName = ?, Phone = ?, Avatar = ? WHERE UserID = ? AND RoleID = 4

## 22. Delete Staff:

### Class Diagram



### Class Specifications

#### **StaffDAO**

| Method | Desciption | Return Type |
| --- | --- | --- |
| getAllStaffs() | Retrieves a list of all staff members. | List<User> |
| getUserByID(int userID) | Finds a staff member by their userID. | User |
| deleteStaff(int userID) | Deletes a staff member based on userID. | boolean |

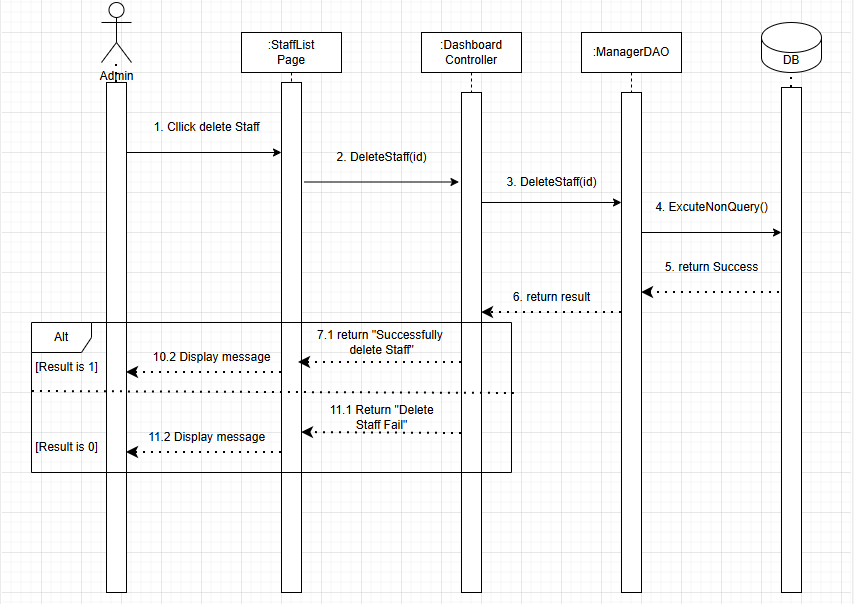
### 

#### StaffServlet

| Method | Description | Return Type |
| --- | --- | --- |
| doPost(HttpServletRequest request, HttpServletResponse response) | Handles the deletion of a staff member. | void |
| doGet(HttpServletRequest request, HttpServletResponse response) | Fetches staff data based on request. | void |

### 

### Sequence Diagram(s)



### Database Queries

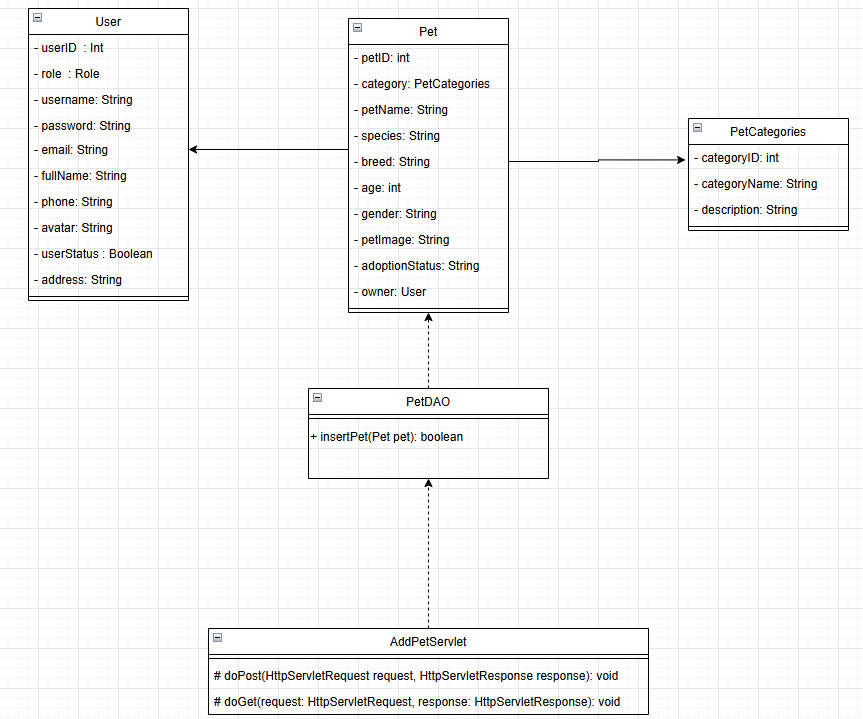
DELETE FROM Users WHERE UserID = ? AND RoleID = 4

## 

## 

## 23. Add pet

### Class Diagram

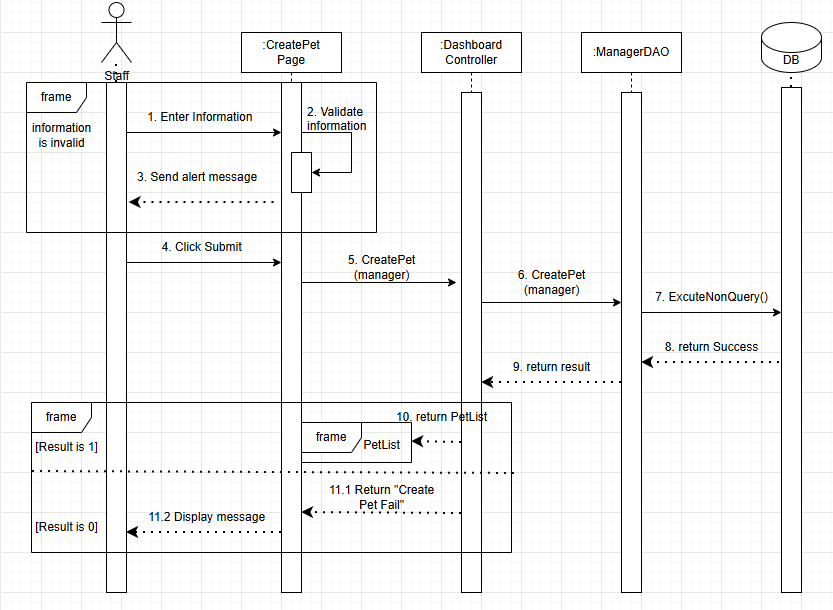


### Class Specifications

| Class Name | Method | Description |
| --- | --- | --- |
| PetDAO | insertPet(Pet pet) | Insert a new pet into the database and return a boolean result indicating success or failure. |
| AddPetServlet | doPost(HttpServletRequest request, HttpServletResponse response) | Handle POST requests to add a new pet to the system. |
| AddPetServlet | doGet(HttpServletRequest request, HttpServletResponse response) | Handle GET requests related to pet addition. |

### 

### Sequence Diagram(s)

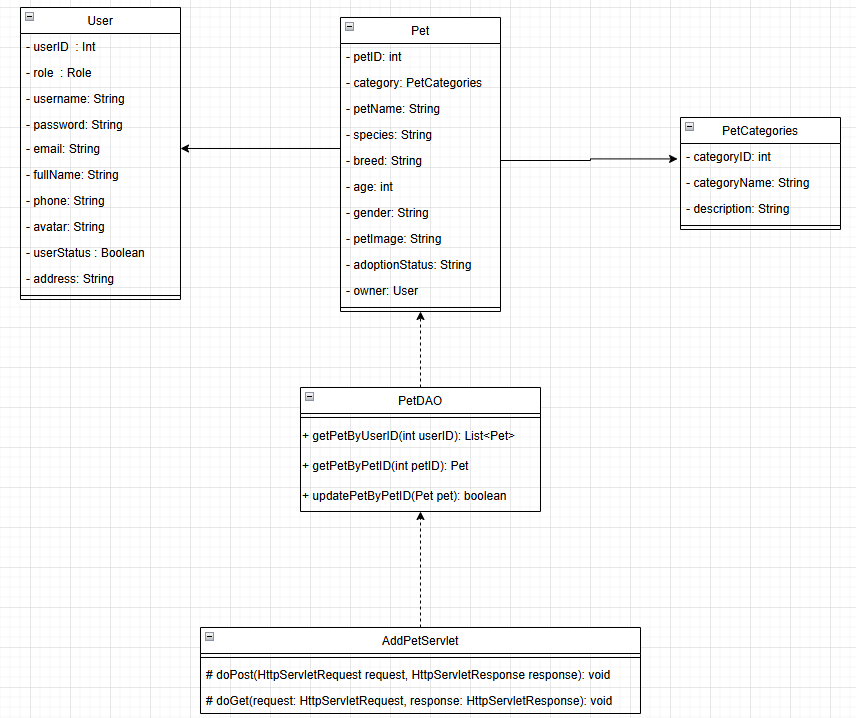


### Database Queries

INSERT INTO Pets (CategoryID, PetName, Species, Breed, Age, Gender, PetImage, AdoptionStatus, UserID) VALUES (?,?,?,?,?,?,?,?,?)

## 24. Update Pet:

### Class Diagram



### Class Specifications

#### **PetDAO**

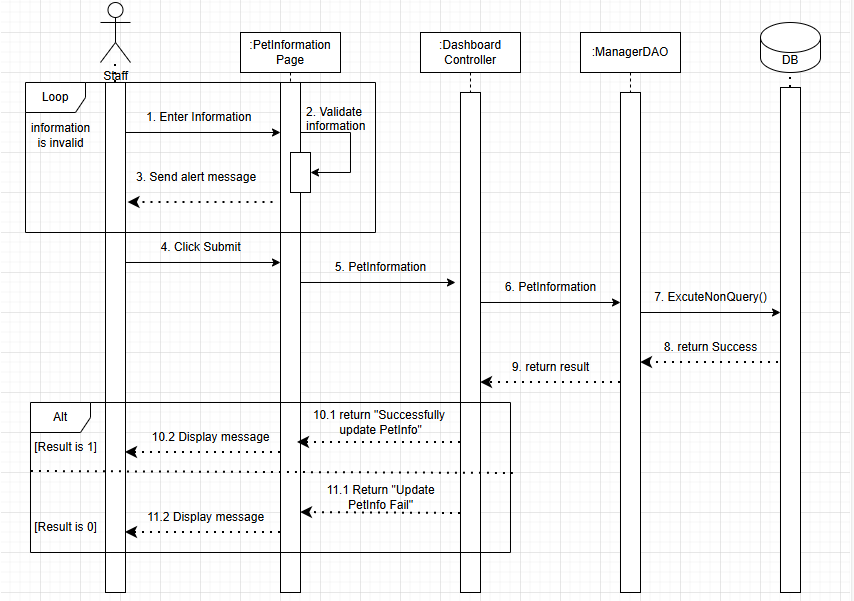
| # | Method | Description |
| --- | --- | --- |
| 1 | getPetByUserID(int userID): List<Pet> | Retrieves a list of pets owned by a user. |
| 2 | getPetByPetID(int petID): Pet | Retrieves a pet's details based on pet ID. |
| 3 | updatePetByPetID(Pet pet): boolean | Updates the information of a pet in the system. |

#### AddPetServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response): void | Handles POST requests for adding new pet details. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response): void | Handles GET requests for retrieving pet details before adding. |

### 

### Sequence Diagram(s)



### Database Queries

UPDATE Pets SET CategoryID=?, PetName=?, Species=?, Breed=?, Age=?, Gender=?, PetImage=?, AdoptionStatus=?, UserID=? WHERE PetID=?

## 

## 

## 

## 

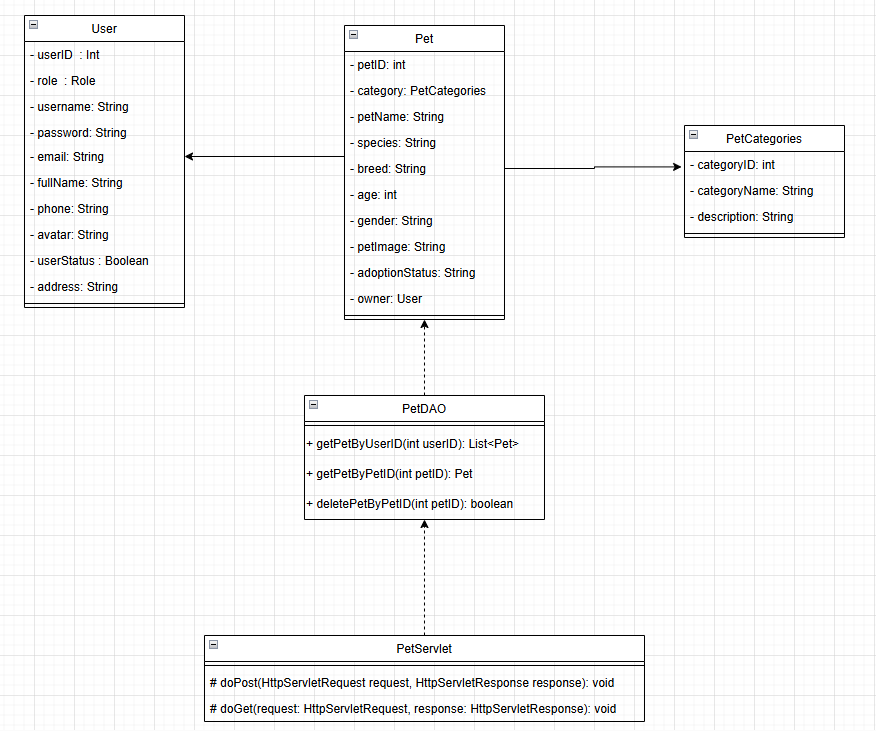
## 

## 

## 

## 25. Delete Pet:

### Class Diagram



### Class Specifications

#### PetDAO

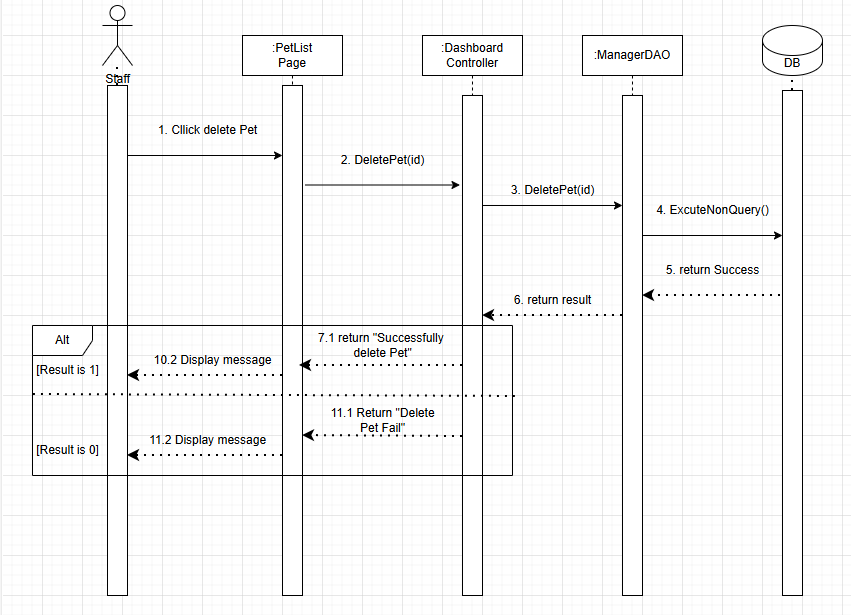
| # | Method | Description |
| --- | --- | --- |
| 1 | getPetByUserID(int userID) | Retrieves a list of pets owned by a specific user. |
| 2 | getPetByPetID(int petID) | Retrieves pet details based on pet ID. |
| 3 | deletePetByPetID(int petID) | Deletes a pet from the system based on pet ID. |

#### PetServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for deleting pet details. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving pet details before deletion. |

### 

### Sequence Diagram(s)



### Database Queries

DELETE FROM Pets WHERE PetID=?

## 

## 

## 

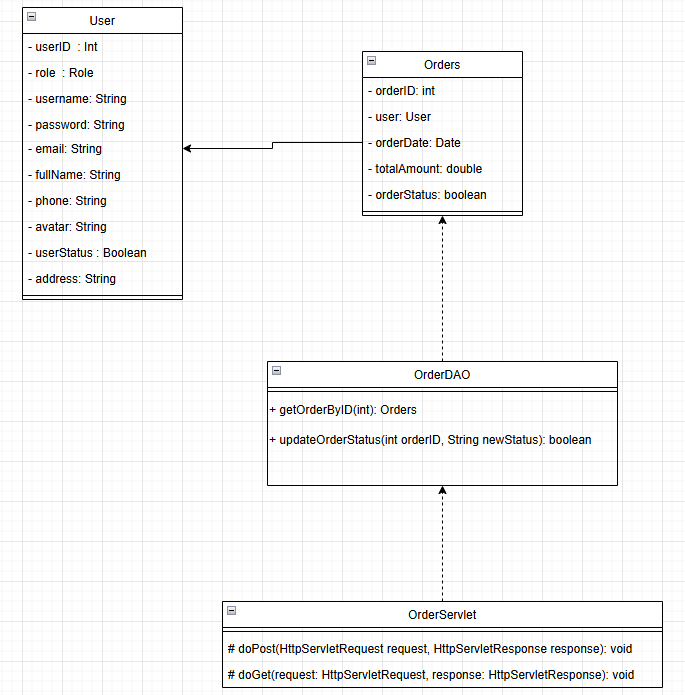
## 

## 

## 

## 26. Change order status:

### Class Diagram



### Class Specifications

#### OrderDAO

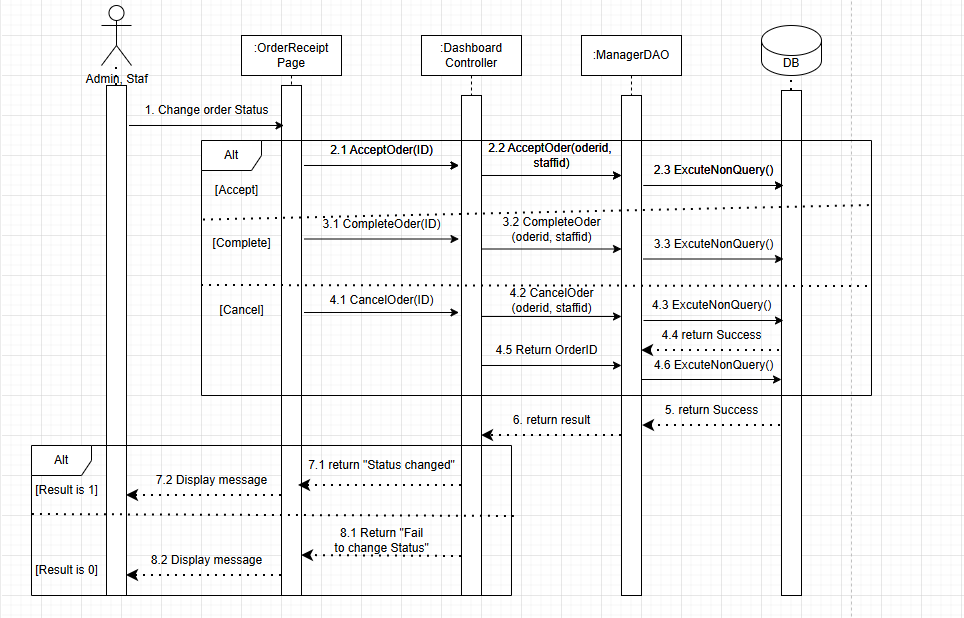
| # | Method | Description |
| --- | --- | --- |
| 1 | getOrderByID(int orderID) | Retrieves an order based on the given order ID. |
| 2 | updateOrderStatus(int orderID, String newStatus) | Updates the status of an order in the system. |

#### OrderServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests to update order details. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests to retrieve order details before updating. |

### 

### Sequence Diagram(s)



### Database Queries

None

## 

## 

## 

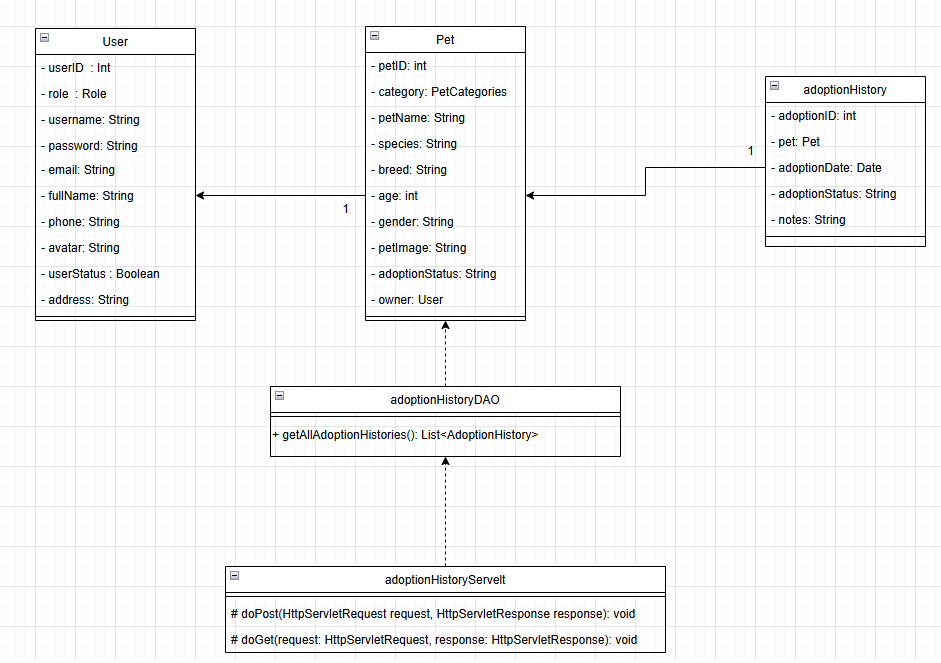
## 

## 

## 

## 27. Adoption History:

### Class Diagram



### Class Specifications

#### **AdoptionHistoryDAO**

| # | Method | Description |
| --- | --- | --- |
| 1 | getAllAdoptionHistories() | Retrieves a list of all pet adoption histories. |

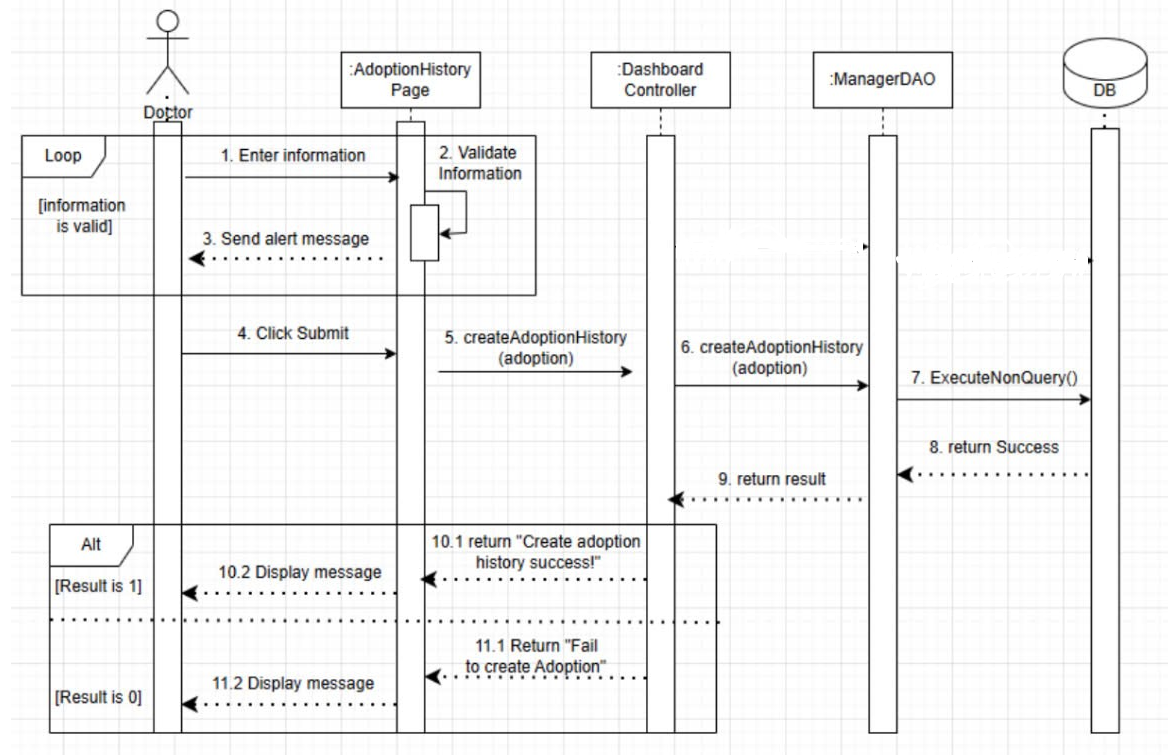
#### AdoptionHistoryServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for submitting adoption history. |
| 2 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving adoption history records. |

### 

### 

### Sequence Diagram(s)



### Database Queries

SELECT \* FROM AdoptionHistory WHERE UserID = ?

## 

## 

## 

## 

## 

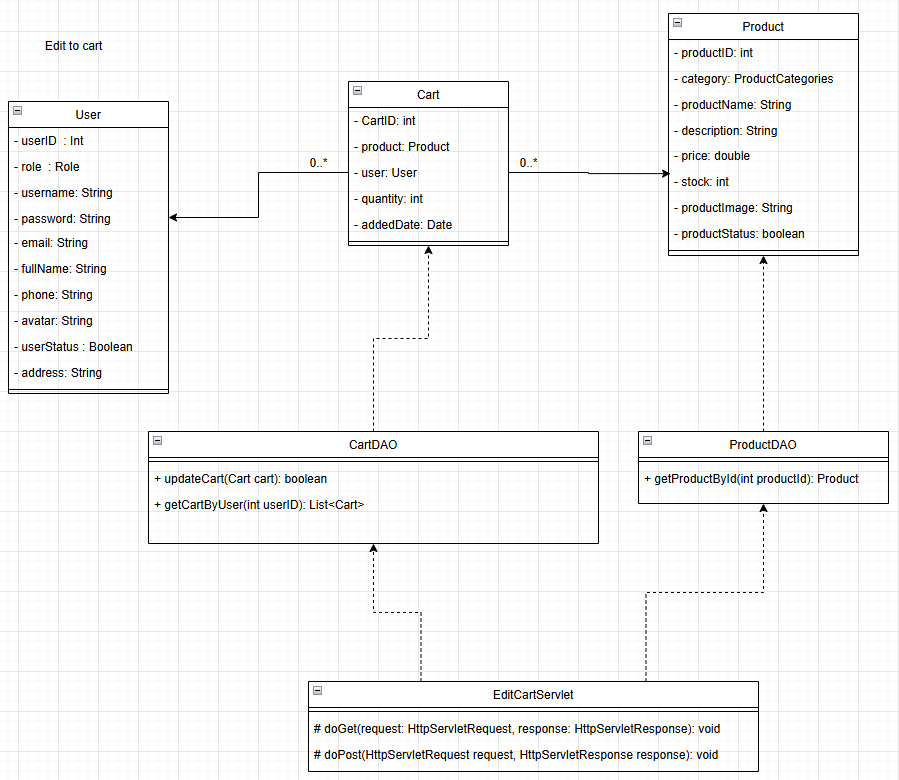
## 

## 

## 

## 28. Edit product in cart:

### Class Diagram



### Class Specifications

#### **CartDAO**

| # | Method | Description |
| --- | --- | --- |
| 1 | updateCart(Cart cart): boolean | Updates the quantity of a product in the cart. |
| 2 | getCartByUser(int userID): List<Cart> | Retrieves all cart items associated with a specific user. |

#### ProductDAO

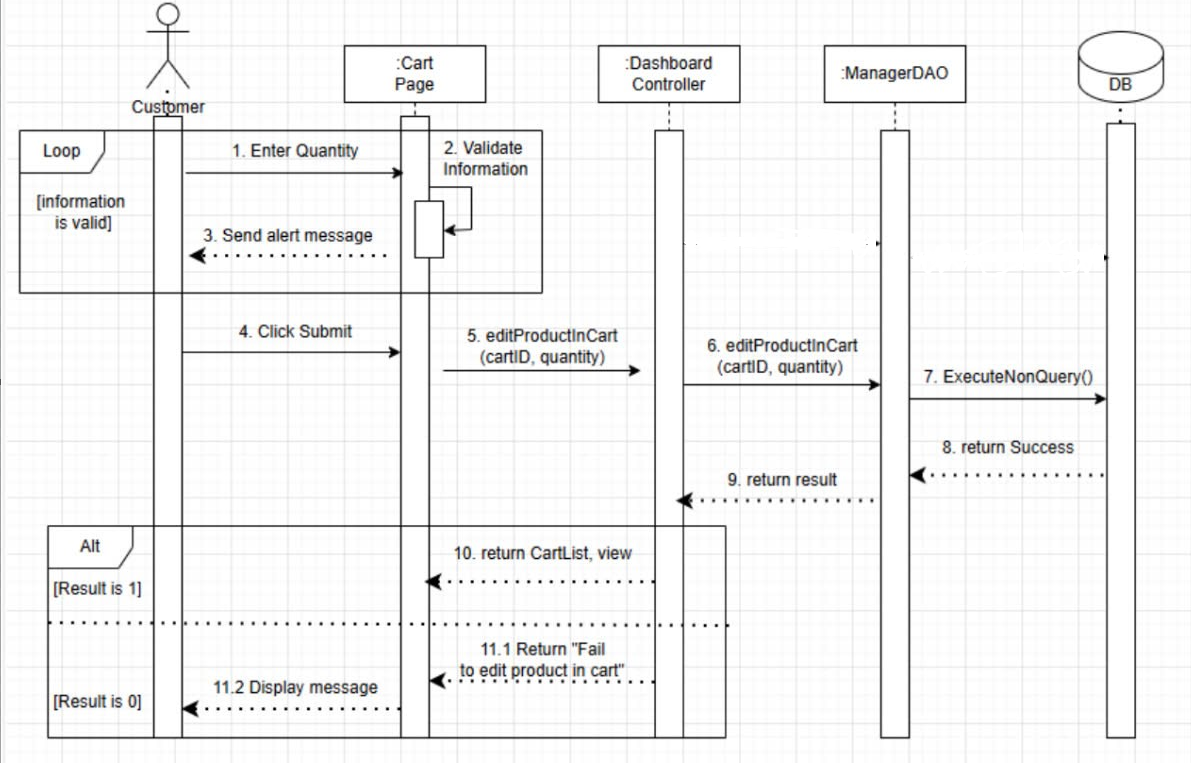
| # | Method | Description |
| --- | --- | --- |
| 1 | getProductByID(int productID): Product | Retrieves product details based on product ID. |

#### EditCartServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response): void | Handles GET requests for retrieving cart items before updating. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response): void | Handles POST requests for updating cart details. |

### 

### Sequence Diagram(s)



### Database Queries

None

## 

## 

## 

## 

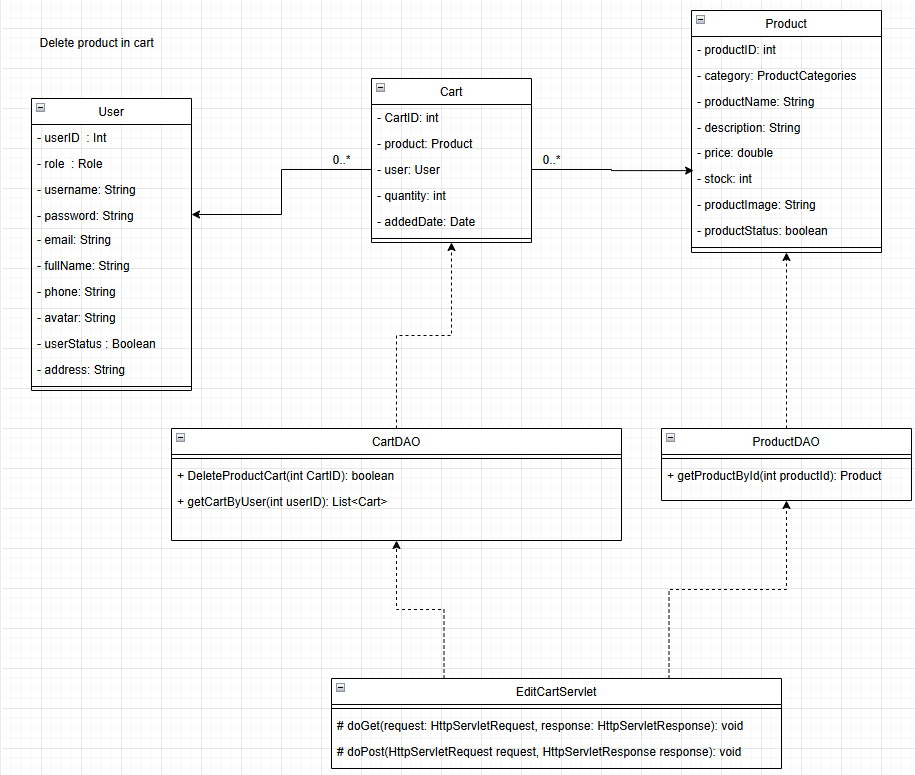
## 

## 

## 

## 29. Delete product in cart:

### Class Diagram



### Class Specifications

#### **CartDAO**

| # | Method | Description |
| --- | --- | --- |
| 1 | DeleteProductCart(int cartID): boolean | Deletes a specific product from the user's cart based on cart ID. |
| 2 | getCartByUser(int userID): List<Cart> | Retrieves all cart items associated with a specific user. |

#### **ProductDAO**

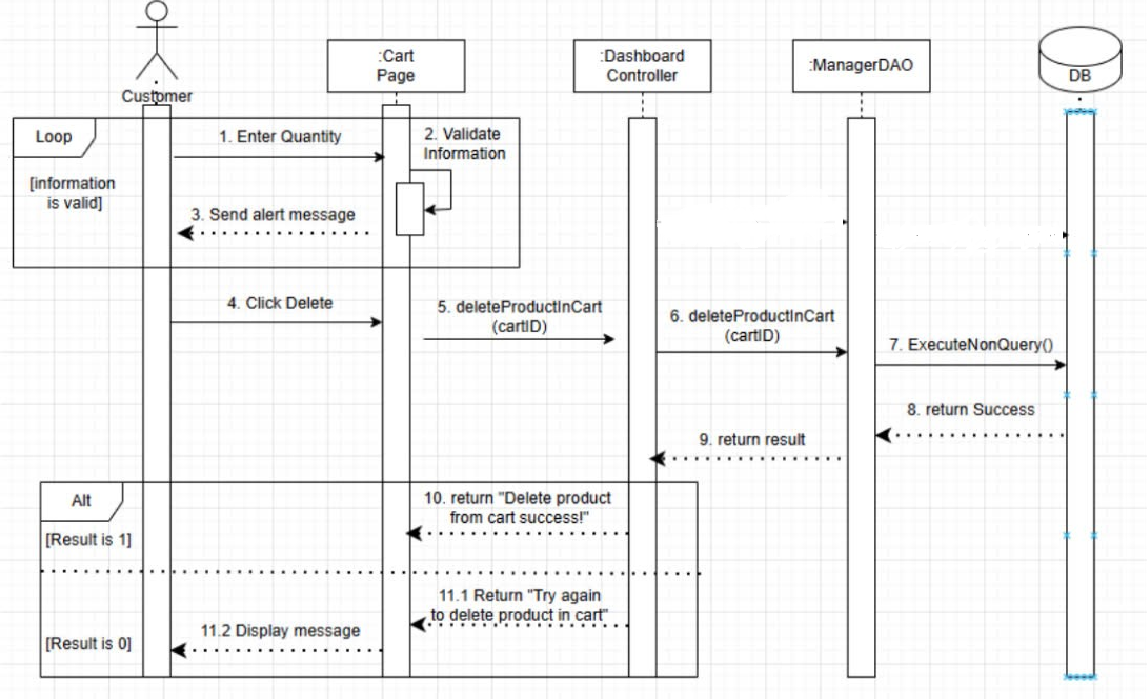
| # | Method | Description |
| --- | --- | --- |
| 1 | getProductByID(int productID): Product | Retrieves product details based on product ID. |

#### EditCartServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response): void | Handles GET requests for retrieving cart items before updating. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response): void | Handles POST requests for updating cart details. |

### 

### Sequence Diagram(s)



### Database Queries

None

## 

## 

## 

## 

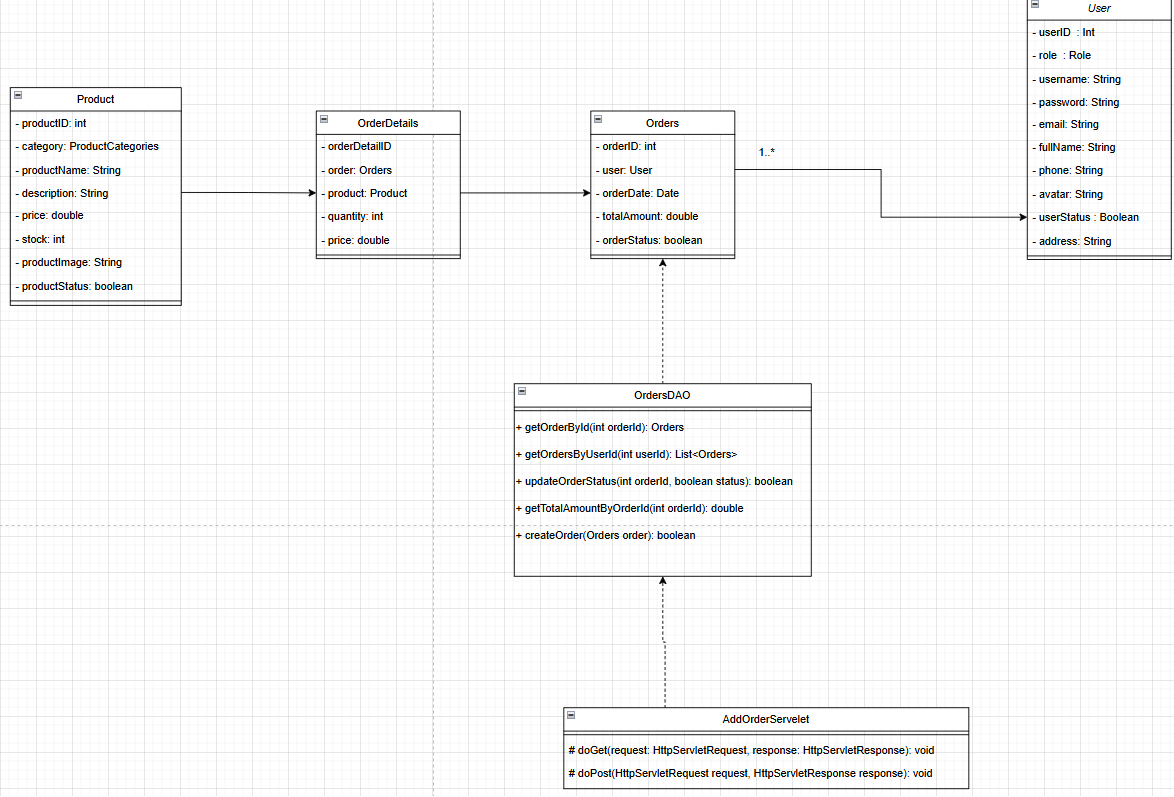
## 

## 

## 

## 30. Add Order Product:

### Class Diagram



### Class Specifications

#### **OrdersDAO**

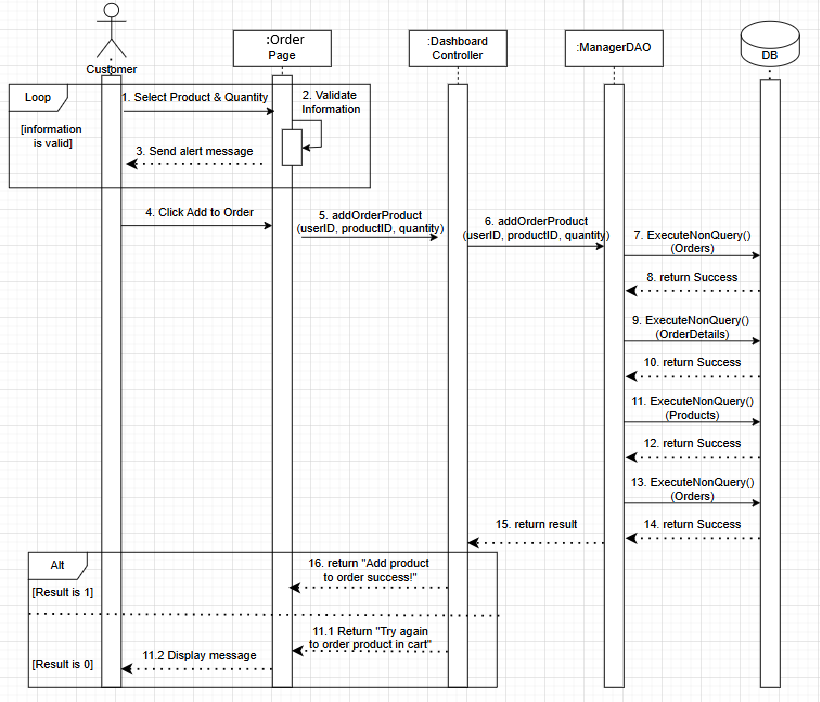
| # | Method | Description |
| --- | --- | --- |
| 1 | getOrderByID(int orderID): Orders | Retrieves order details based on order ID. |
| 2 | getOrdersByUser(int userID): List<Orders> | Retrieves all orders associated with a specific user. |
| 3 | updateOrderStatus(int orderID, boolean status): boolean | Updates the status of an order. |
| 4 | getTotalAmountByOrderID(int orderID): double | Retrieves the total amount of a specific order. |
| 5 | createOrder(Orders order): boolean | Creates a new order in the system. |

#### **AddOrderServlet**

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response): void | Handles GET requests for retrieving order details before processing. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response): void | Handles POST requests for adding new orders. |

### 

### Sequence Diagram(s)



### Database Queries

INSERT INTO Orders (UserID, OrderDate, TotalAmount, OrderStatus) VALUES (?, GETDATE(), ?, 'Pending')

## 

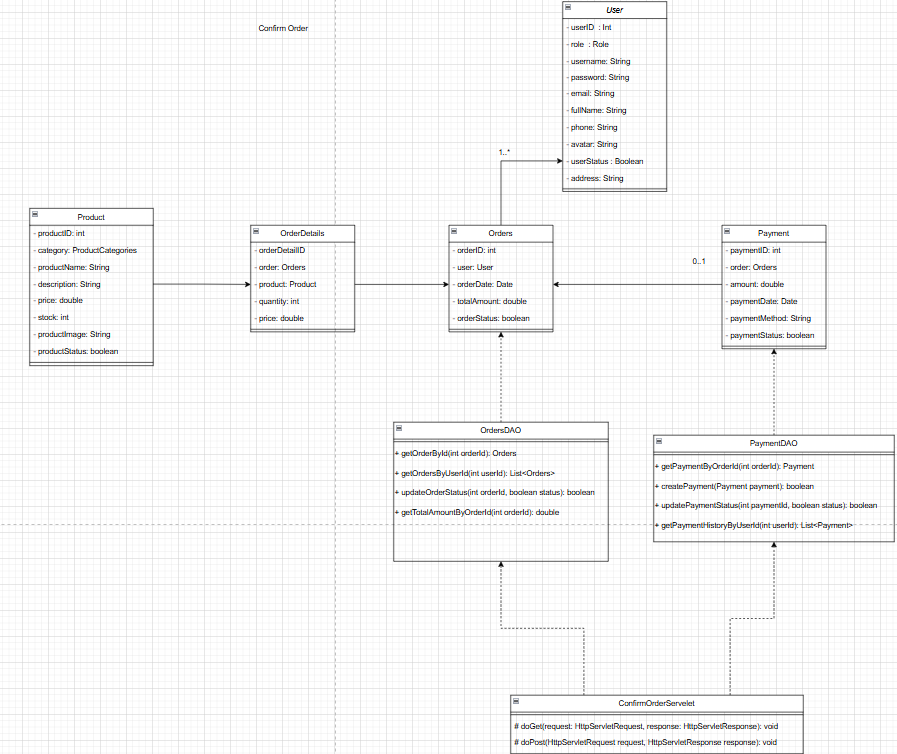
## 

## 

## 

## 31. ConfirmOrder:

### Class Diagram



### Class Specifications

#### **OrdersDAO**

| # | Method | Description |
| --- | --- | --- |
| 1 | getOrderByID(int orderID): Orders | Retrieves order details based on order ID. |
| 2 | getOrdersByUser(int userID): List<Orders> | Retrieves all orders associated with a specific user. |
| 3 | updateOrderStatus(int orderID, boolean status): boolean | Updates the status of an order. |
| 4 | getTotalAmountByOrderID(int orderID): double | Retrieves the total amount of a specific order. |

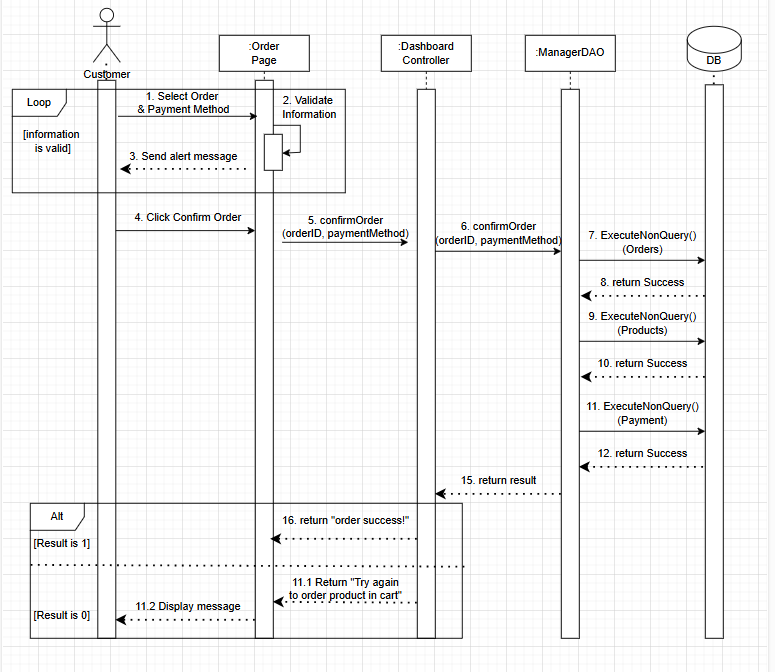
#### **PaymentDAO**

| # | Method | Description |
| --- | --- | --- |
| 1 | getPaymentByOrderID(int orderID): Payment | Retrieves payment details associated with an order. |
| 2 | createPayment(Payment payment): boolean | Creates a new payment record in the system. |
| 3 | updatePaymentStatus(int paymentID, boolean status): boolean | Updates the payment status of a specific payment. |
| 4 | getPaymentHistoryByUserID(int userID): List<Payment> | Retrieves the payment history of a specific user. |

#### **ConfirmOrderServlet**

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response): void | Handles GET requests for retrieving order details before confirmation. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response): void | Handles POST requests for confirming an order. |

### Sequence Diagram(s)



### Database Queries

UPDATE Orders SET OrderStatus = ? WHERE OrderID = ?

UPDATE Orders SET OrderStatus = 'Confirmed' WHERE OrderID = ?

## 

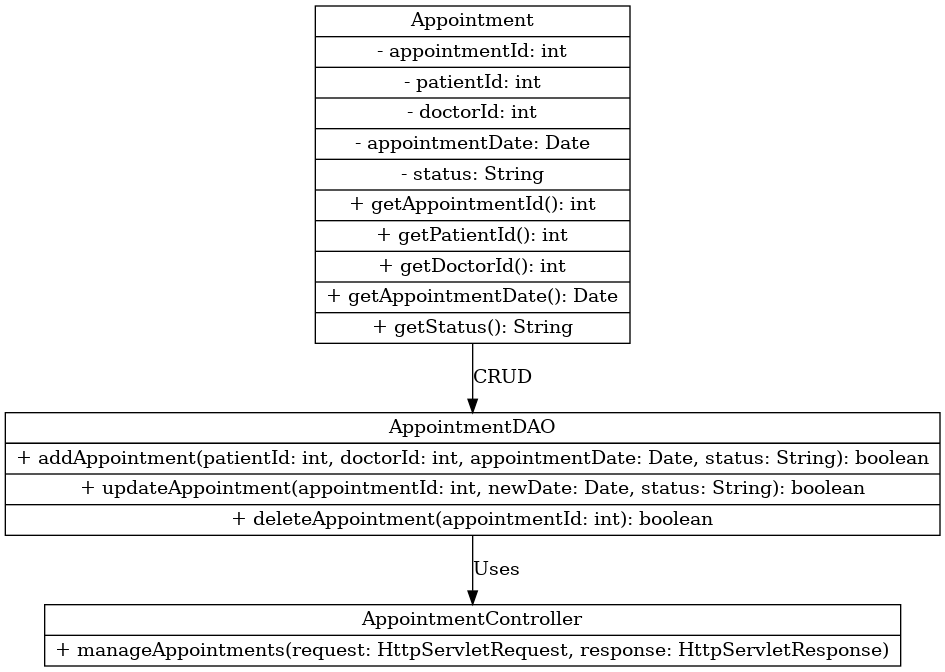
## 

## 

## 

## 32. Manage Appointments:

### Class Diagram



### Class Specifications

#### **Appointment**

| Attribute | Type | Description |
| --- | --- | --- |
| appointmentId | int | Unique identifier for the appointment. |
| patientId | int | The ID of the patient associated with the appointment. |
| doctorId | int | The ID of the doctor assigned to the appointment. |
| appointmentDate | Date | The scheduled date of the appointment. |
| status | String | The status of the appointment (e.g., scheduled, completed, canceled). |

#### **Methods**

| # | Method | Discription |
| --- | --- | --- |
| 1 | getAppointmentId() : int | Retrieves the ID of the appointment. |
| 2 | getPatientId() : int | Retrieves the ID of the patient. |
| 3 | getDoctorId() : int | Retrieves the ID of the doctor. |
| 4 | getAppointmentDate() : Date | Retrieves the date of the appointment. |
| 5 | getStatus() : String | Retrieves the status of the appointment. |

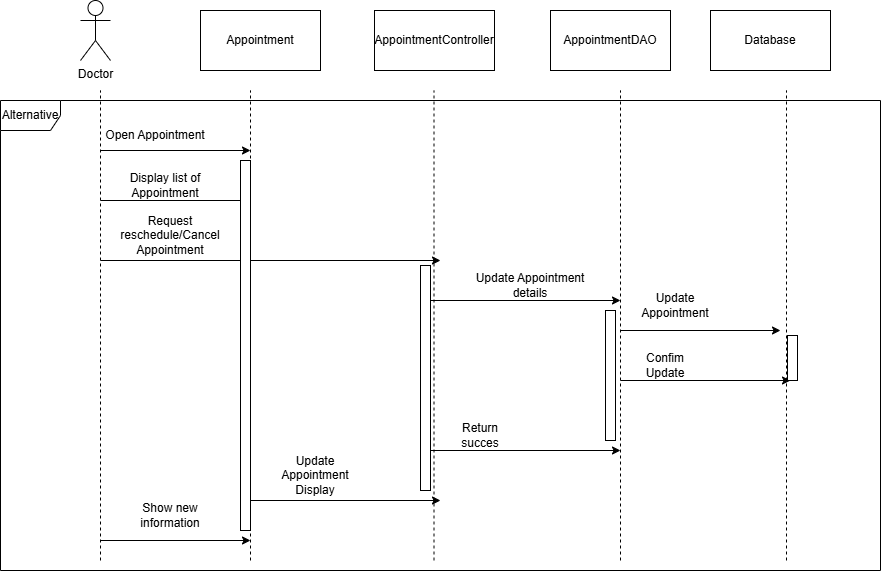
#### **AppointmentDAO**

| # | Method | Description |
| --- | --- | --- |
| 1 | addAppointment(int patientId, int doctorId, Date appointmentDate, String status) : boolean | Adds a new appointment to the system. |
| 2 | updateAppointment(int appointmentId, Date newDate, String status) : boolean | Updates an existing appointment's date and status. |
| 3 | deleteAppointment(int appointmentId) : boolean | Deletes an appointment from the system. |

#### **AppointmentController**

| # | Method | Description |
| --- | --- | --- |
| 1 | manageAppointments(HttpServletRequest request, HttpServletResponse response) | Handles appointment-related requests (e.g., create, update, delete). |

### Sequence Diagram(s)

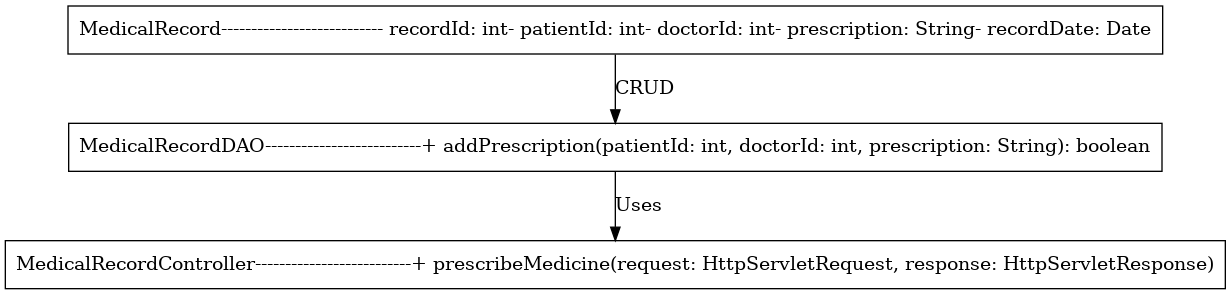


### Database Queries

SELECT \* FROM Appointments

## 33. Prescribe Medicine:

### Class Diagram



### Class Specifications

#### **MedicalRecord**

| Attribute | Type | Description |
| --- | --- | --- |
| recordId | int | Unique identifier for the medical record. |
| patientId | int | The ID of the patient associated with the medical record. |
| doctorId | int | The ID of the doctor who prescribed the treatment. |
| prescription | String | The prescription details provided by the doctor. |
| recordDate | Date | The date when the medical record was created. |

#### **MedicalRecordDAO**

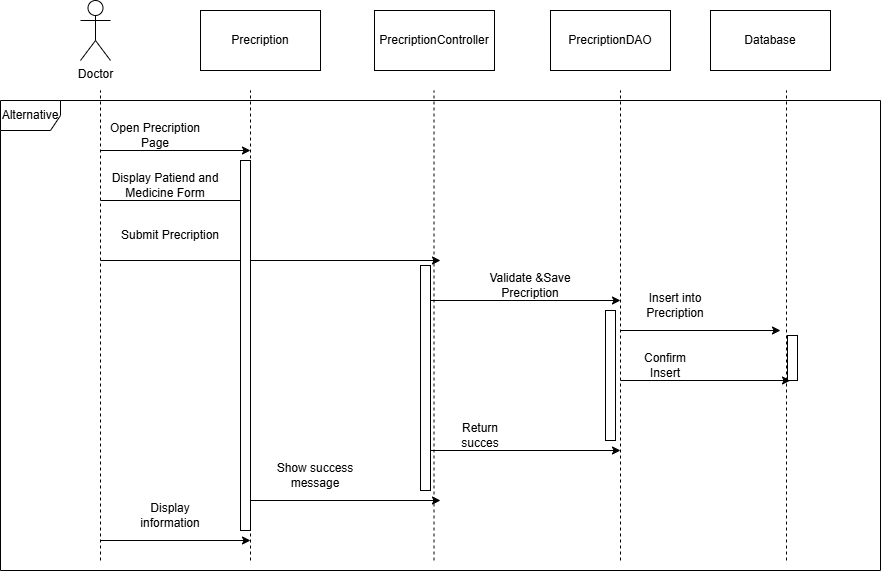
| # | Method | Description |
| --- | --- | --- |
| 1 | addPrescription(int patientId, int doctorId, String prescription) : boolean | Adds a prescription to a patient’s medical record. |

#### **MedicalRecordController**

| # | Method | Description |
| --- | --- | --- |
| 1 | prescribeMedicine(HttpServletRequest request, HttpServletResponse response) | Handles prescription-related requests from doctors. |

### 

### Sequence Diagram(s)



### Database Queries

INSERT INTO Prescriptions (AppointmentID, MedicineName, Dosage, Instructions) VALUES (?, ?, ?, ?)

## 

## 

## 

## 

## 

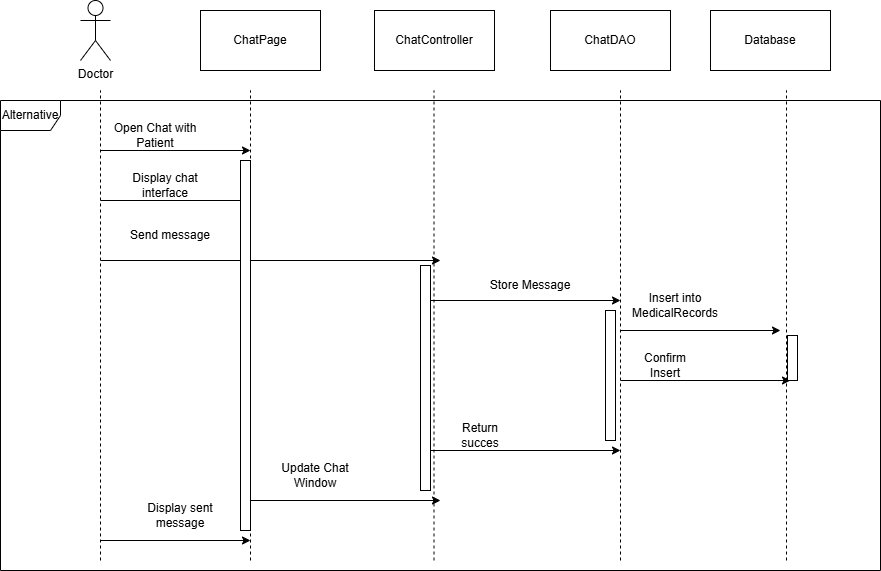
## 

## 34. **Communicate with Patients:**

### Class Diagram

### Class Specifications

### Sequence Diagram(s)

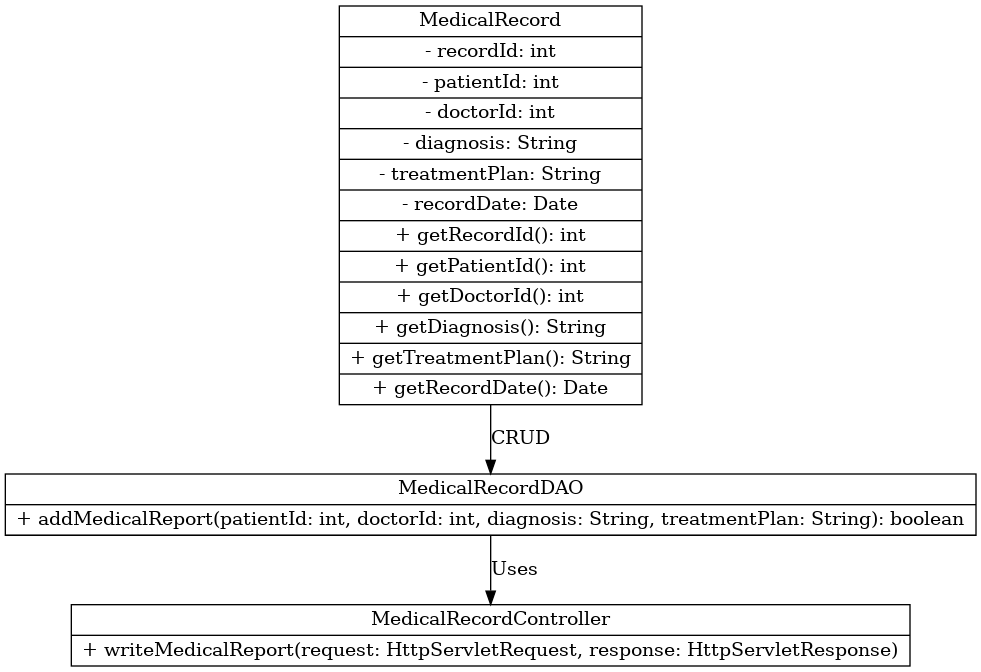


### Database Queries

None

## 35. Write Medical Reports:

### Class Diagram



### Class Specifications

#### **MedicalRecord**

| Attribute | Type | Description |
| --- | --- | --- |
| recordId | int | Unique identifier for the medical record. |
| patientId | int | The ID of the patient associated with the medical record. |
| doctorId | int | The ID of the doctor who created the medical report. |
| diagnosis | String | The diagnosis provided by the doctor. |
| treatmentPlan | String | The treatment plan recommended for the patient. |
| recordDate | Date | The date when the medical report was recorded. |

#### **Methods**

| Method | Return Type | Description |
| --- | --- | --- |
| getRecordId() | int | Retrieves the record ID. |
| getPatientId() | int | Retrieves the patient ID. |
| getDoctorId() | int | Retrieves the doctor ID. |
| getDiagnosis() | String | Retrieves the diagnosis details. |
| getTreatmentPlan() | String | Retrieves the treatment plan details. |
| getRecordDate() | Date | Retrieves the date of the medical report. |

#### MedicalRecordDAO

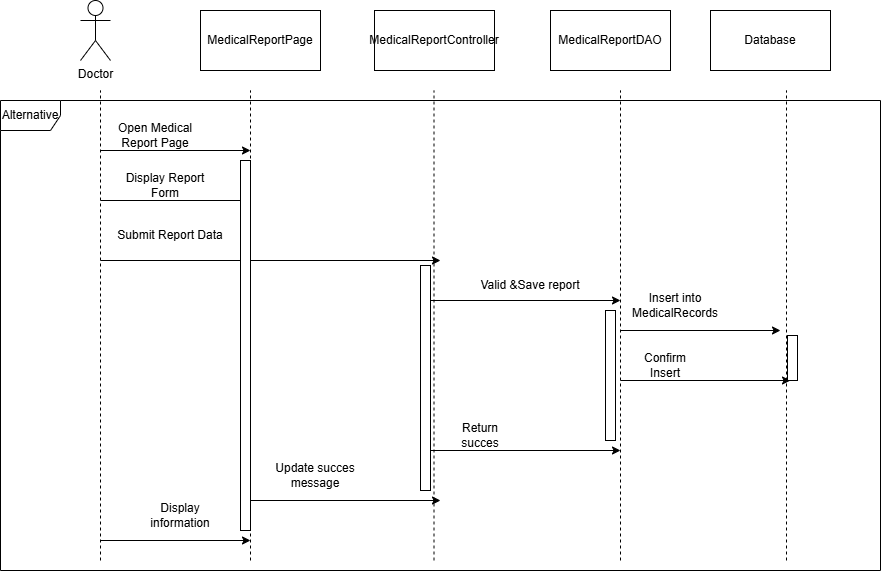
| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | addMedicalReport(int patientId, int doctorId, String diagnosis, String treatmentPlan) | boolean | Adds a new medical report to the system. |

#### **MedicalRecordController**

| # | Method | Description |
| --- | --- | --- |
| 1 | writeMedicalReport(HttpServletRequest request, HttpServletResponse response) | Handles HTTP requests related to creating medical reports. |

### 

### Sequence Diagram(s)



### Database Queries

INSERT INTO MedicalRecords (AppointmentID, Diagnosis, Treatment, Notes) VALUES (?, ?, ?, ?)

## 

## 

## 

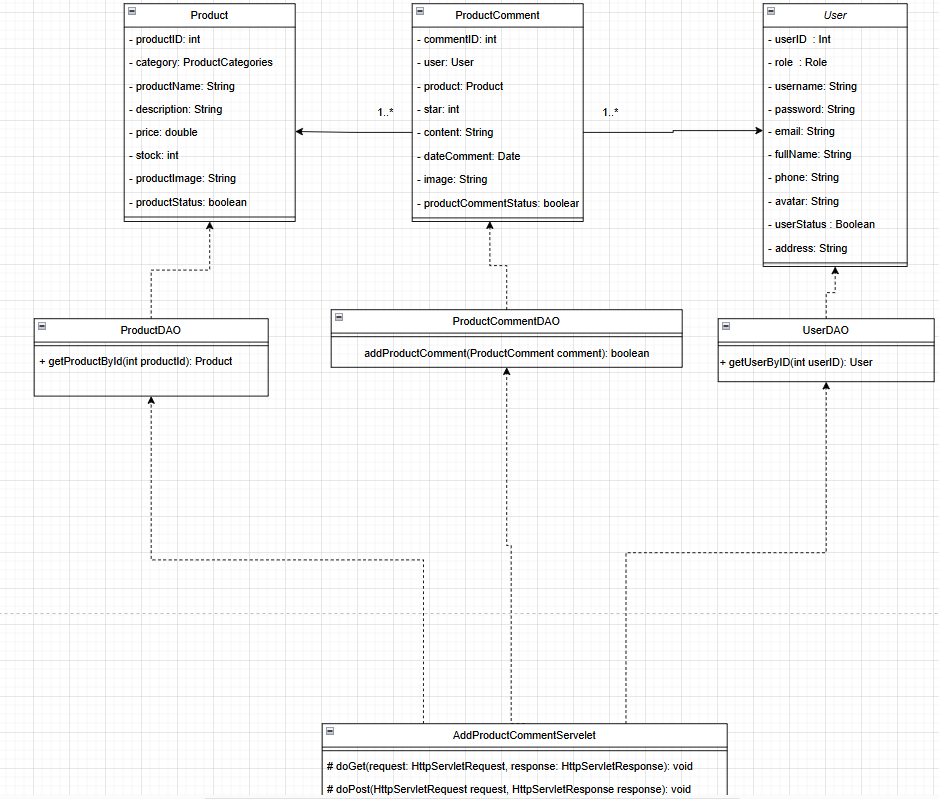
## 

## 

## 

## 36. Add comment:

### Class Diagram



### Class Specifications

#### Product

| Attribute | Type | Description |
| --- | --- | --- |
| productID | int | Unique identifier for the product. |
| category | ProductCategories | Category of the product. |
| productName | String | Name of the product. |
| description | String | Description of the product. |
| price | Double | Price of the product. |
| stock | int | Available stock of the product. |
| productImage | String | Image URL of the product. |
| productStatus | boolean | Status indicating if the product is available. |

#### ProductComment

| Attribute | Type | Description |
| --- | --- | --- |
| commentID | int | Unique identifier for the comment. |
| user | User | User who posted the comment. |
| product | Product | Product associated with the comment. |
| star | int | Star rating given by the user (1-5). |
| content | String | Text content of the comment. |
| dateComment | Date | The date when the comment was posted. |
| image | String | Optional image attached to the comment. |
| productCommentStatus | boolean | Status indicating if the comment is active. |

#### **User**

| Attribute | Type | Description |
| --- | --- | --- |
| userID | int | Unique identifier for the user. |
| role | Role | Role of the user (e.g., Admin, Customer). |
| username | String | Username of the user. |
| password | String | Encrypted password of the user. |
| email | String | Email address of the user. |
| fullName | String | Full name of the user. |
| phone | String | Phone number of the user. |
| avatar | String | Profile image URL of the user. |
| userStatus | boolean | Status indicating if the user account is active. |
| address | String | Address of the user. |

#### ProductDAO

| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | getProductById(int productID) | Product | Retrieves product details based on product ID. |

#### **ProductCommentDAO**

| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | addProductComment(ProductComment comment) | boolean | Adds a new product comment to the system. |

#### **UserDAO**

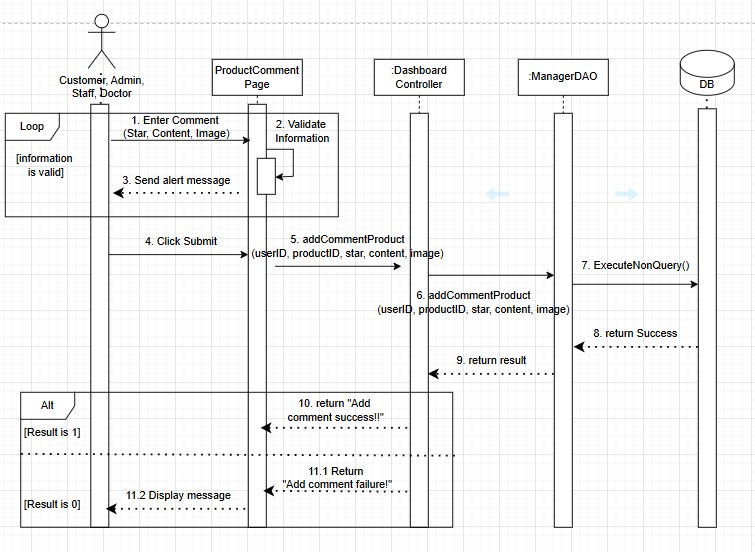
| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | getUserByID(int userID) | User | Retrieves user details based on user ID. |

#### **AddProductCommentServlet**

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving product comments. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for adding new product comments. |

### 

### Sequence Diagram(s)



### Database Queries

INSERT INTO ProductComments (ProductID, UserID, Comment, Rating, CommentDate) VALUES (?, ?, ?, ?, GETDATE())

## 

## 

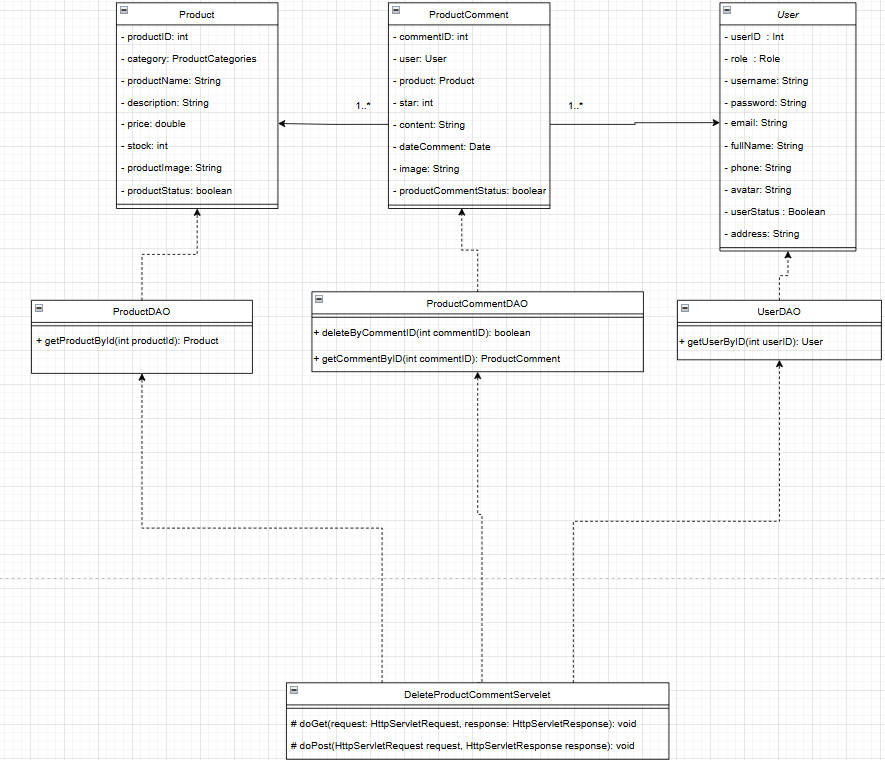
## 

## 

## 

## 37. Delete comment:

### Class Diagram



### Class Specifications

#### Product

| Attribute | Type | Description |
| --- | --- | --- |
| productID | int | Unique identifier for the product. |
| category | ProductCategories | Category of the product. |
| productName | String | Name of the product. |
| description | String | Description of the product. |
| price | double | Price of the product. |
| stock | int | Available stock of the product. |
| productImage | String | Image URL of the product. |
| productStatus | boolean | Status indicating if the product is available. |

#### **ProductComment**

| Attribute | Type | Description |
| --- | --- | --- |
| commentID | int | Unique identifier for the comment. |
| user | User | User who posted the comment. |
| product | Product | Product associated with the comment. |
| star | int | Star rating given by the user (1-5). |
| content | String | Text content of the comment. |
| dateComment | Date | The date when the comment was posted. |
| image | String | Optional image attached to the comment. |
| productCommentStatus | boolean | Status indicating if the comment is active. |

#### **User**

| Attribute | Type | Description |
| --- | --- | --- |
| userID | int | Unique identifier for the user. |
| role | Role | Role of the user (e.g., Admin, Customer). |
| username | String | Username of the user. |
| password | String | Encrypted password of the user. |
| email | String | Email address of the user. |
| fullname | String | Full name of the user. |
| phone | String | Phone number of the user. |
| avatar | String | Profile image URL of the user. |
| userStatus | boolean | Status indicating if the user account is active. |
| address | String | Address of the user. |

#### **ProductDAO**

| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | getProductById(int productID) | Product | Retrieves product details based on product ID. |

#### **ProductCommentDAO**

| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | deleteByCommentID(int commentID) | boolean | Deletes a product comment based on comment ID. |
| 2 | getCommentByID(int commentID) | ProductComment | Retrieves a comment by comment ID. |

#### UserDAO

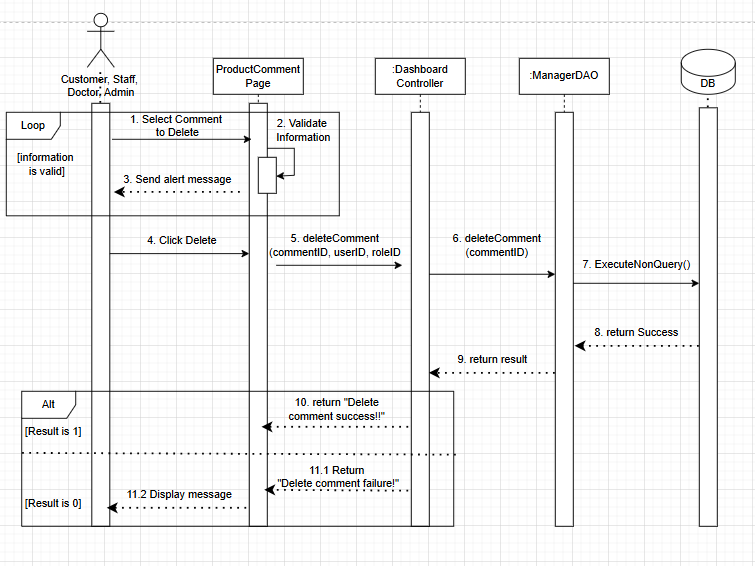
| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | getUserByID(int userID) | User | Retrieves user details based on user ID. |

#### **DeleteProductCommentServlet**

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving a product comment before deletion. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for deleting a product comment. |

### 

### Sequence Diagram(s)



### Database Queries

DELETE FROM ProductComments WHERE CommentID = ? AND UserID = ?

## 

## 

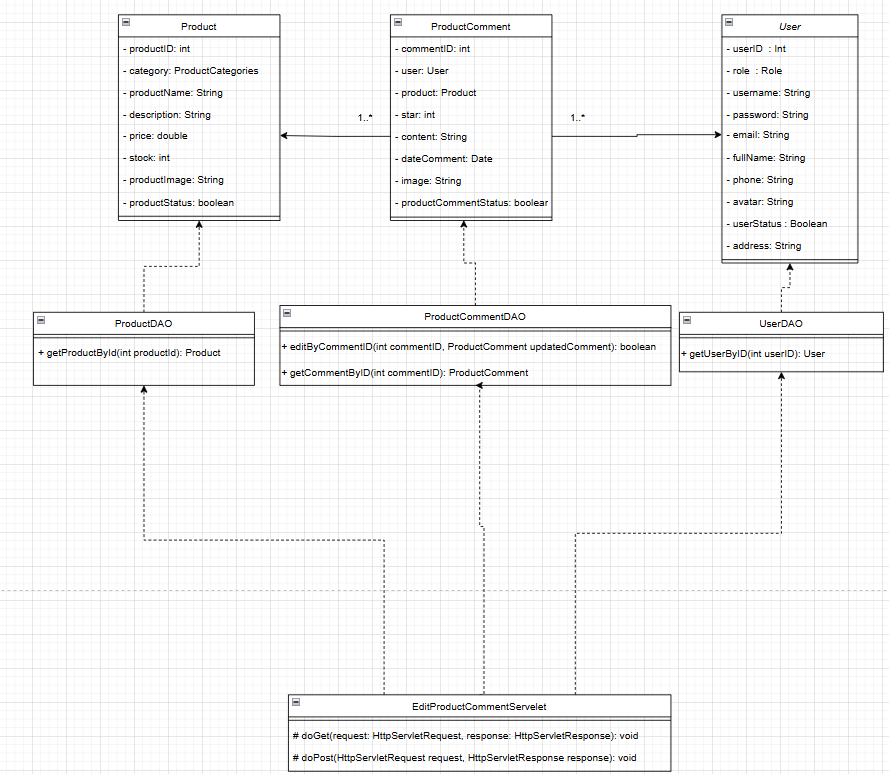
## 

## 

## 

## 38. Edit comment:

### Class Diagram



### Class Specifications

#### Product

| Attribute | Type | Description |
| --- | --- | --- |
| productID | int | Unique identifier for the product. |
| category | ProductCategories | Category of the product. |
| productName | String | Name of the product. |
| description | String | Description of the product. |
| price | double | Price of the product. |
| stock | int | Available stock of the product. |
| productImage | String | Image URL of the product. |
| productStatus | boolean | Status indicating if the product is available. |

#### ProductComment

| Attribute | Type | Description |
| --- | --- | --- |
| commentID | int | Unique identifier for the comment. |
| user | User | User who posted the comment. |
| product | Product | Product associated with the comment. |
| star | int | Star rating given by the user (1-5). |
| content | String | Text content of the comment. |
| dateComment | Date | The date when the comment was posted. |
| image | String | Optional image attached to the comment. |
| productCommentStatus | boolean | Status indicating if the comment is active. |

#### **User**

| Attribute | Type | Description |
| --- | --- | --- |
| userID | int | Unique identifier for the user. |
| role | Role | Role of the user (e.g., Admin, Customer). |
| username | String | Username of the user. |
| password | String | Encrypted password of the user. |
| email | String | Email address of the user. |
| fullName | String | Full name of the user. |
| phone | String | Phone number of the user. |
| avatar | String | Profile image URL of the user. |
| userStatus | boolean | Status indicating if the user account is active. |
| address | String | Address of the user. |

#### **ProductDAO**

| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | getProductById(int productID) | Product | Retrieves product details based on product ID. |

#### **ProductCommentDAO**

| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | editByCommentID(int commentID, ProductComment updatedComment) | boolean | Updates an existing product comment. |
| 2 | getCommentByID(int commentID) | ProductComment | Retrieves a comment by comment ID. |

#### **UserDAO**

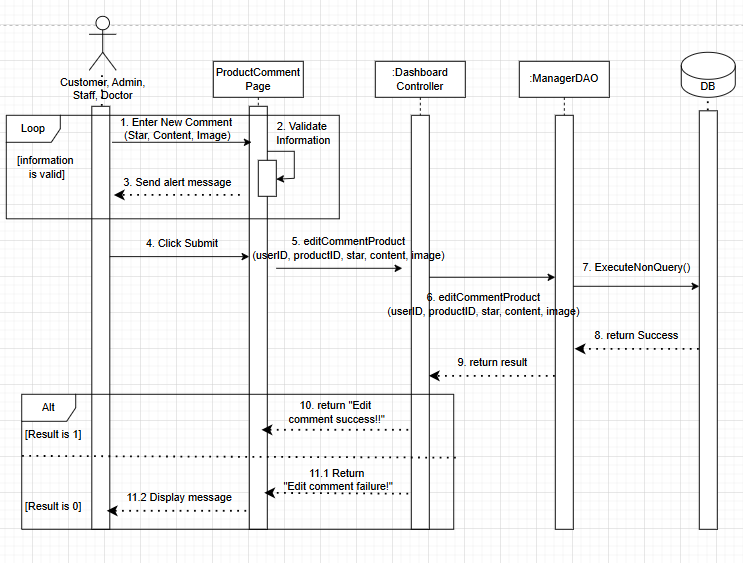
| # | Method | Return Type | Description |
| --- | --- | --- | --- |
| 1 | getUserByID(int userID) | User | Retrieves user details based on user ID. |

#### EditProductCommentServlet

| # | Method | Description |
| --- | --- | --- |
| 1 | doGet(HttpServletRequest request, HttpServletResponse response) | Handles GET requests for retrieving a product comment before updating. |
| 2 | doPost(HttpServletRequest request, HttpServletResponse response) | Handles POST requests for updating a product comment. |

### 

### Sequence Diagram(s)



### Database Queries

UPDATE ProductComments SET Comment = ?, Rating = ?, ModifiedDate = GETDATE() WHERE CommentID = ? AND UserID = ?