



North South University

Department of Electrical and Computer Engineering

CSE327: Software Engineering

Semester: Fall 2025

Section: 06

CarHub: Car Reselling Platform

Submitted To:

AKM Iqtidar Newaz

Professor

NORTH SOUTH UNIVERSITY

Submitted by:

GROUP - 03

NAME	NSU ID
Samina Islam Mim	2311943042
Safwan Ismaun Amin	2311443642
Farhan Karim	2311790642
Safat Uddin	231196642

GitHub Repository: https://github.com/SafatUddin/CAR_Hub

1. PROJECT DESCRIPTION:

1.1 OVERVIEW

CarHub is an online platform for buying and selling cars that simplifies and speeds up the process compared to traditional methods like dealerships or newspaper ads. It connects buyers and sellers securely and conveniently, supporting new and used cars. Buyers can find vehicles based on their needs and budget, while sellers can reach a wider audience efficiently.

1.2 PROBLEM STATEMENT

The current car resale market faces major issues for both buyers and sellers.

- **Sellers** struggle to attract enough buyers, manage inquiries, and ensure safe payments.
- **Buyers** face problems verifying car conditions, tracking prices, and finding fair deals.

Existing platforms are limited, they lack advanced search, real-time notifications, secure transactions, and trust mechanisms. Moreover, poor verification often leads to fake listings and unreliable sellers.

1.3 SOLUTION AND KEY FEATURES

CarHub provides a marketplace by integrating advanced features that simplify the buying and selling process while ensuring security, transparency, and user satisfaction.

Advanced Search and Discovery

CarHub offers a powerful search system that allows users to combine multiple filters such as make, model, price range, year, mileage, and vehicle type. Users can refine their searches easily, save preferred criteria, and access tailored results that match their preferences.

Intelligent Price Alerts

Users can subscribe to notifications for specific car listings or price ranges. The system automatically tracks new listings and price changes, sending instant alerts when relevant opportunities arise ensuring buyers never miss the best deals.

Flexible Listing Creation

Sellers can quickly create and customize listings for different types of vehicles, each with relevant details such as warranty, ownership history.

1.4 TECHNICAL ARCHITECTURE AND DESIGN

CarHub is built on an architecture designed for scalability, reliability, and ease of maintenance. The system structure allows different components to work independently while maintaining seamless integration, enabling smooth updates and future feature expansion.

The platform emphasizes clear separation of concerns, efficient resource management, and reusable components. This ensures that new functionalities can be added with minimal disruption to existing systems, supporting long-term growth and adaptability as user needs evolve.

1.5 USER EXPERIENCE AND INTERFACE DESIGN

CarHub features a clean, responsive design for desktop and mobile users.

- Buyers can browse listings, manage alerts.
- Sellers access a dashboard to manage listings, and respond to inquiries.

The interface follows modern design principles—simple navigation, clear call-to-actions, and visual hierarchy for a smooth user journey.

1.6 SECURITY AND TRUST

CarHub prioritizes safety through:

- Secure password hashing.
- Protection against SQL injection.

An administrative moderation system ensures listing verification and user accountability, promoting trust and transparency across the platform.

2. END USERS:

2.1 PRIMARY USERS

Buyers: Individuals looking to purchase vehicles (new or used)

Sellers: Individual car owners looking to sell their vehicles or small-scale car dealers.

2.2 SECONDARY USERS

System Administrators: Admins to manage the data of Sellers and Buyers.

3. PROPOSED FUNCTIONAL REQUIREMENTS:

3.1 USER MANAGEMENT

a) User Registration

- Users can register as buyers or sellers with email and password.
- System validates unique email addresses.
- Users must verify email before full account access.

b) User Authentication

- Users can log in with email and password.
- System maintains secure sessions.
- Users can reset forgotten passwords.

c) User Profile Management

- Users can view and edit their profile information.
- Users can upload profile pictures.

3.2 CAR LISTING MANAGEMENT

a) Create Listings

- Sellers can create new car listings with detailed information (make, model, year, price, mileage, condition).
- System supports multiple car types: new cars or used cars.
- Sellers can upload images per listing.

b) Edit and Delete Listings

- Sellers can edit their active listings.
- Sellers can delete or mark listings as sold.

c) View Listings

- All users can browse available car listings
- Users can view detailed information about each car
- System displays seller contact information on listings

3.3 SEARCH AND FILTER

a) Search

- Users can search cars using specific criteria
- The system returns relevant results based on search query

b) Filter Options

- Users can filter search results by various attributes
- Users can sort results by price, year, mileage, or date posted

3.4 PRICE ALERT SYSTEM

a) Subscribe to get Alerts

- Buyers can subscribe to specific sellers/criteria for alerts.
- System sends alerts to users who are subscribed.
- Users can subscribe or unsubscribe.

b) Alert Notifications

- System automatically checks for matching listings when prices update.
- Users receive notifications when criteria are met

3.5 COMMUNICATION

a) Contact Seller

- Buyers can send inquiries to sellers through the platform.
- Users receive notifications for new messages

b) Notifications

- Users receive notifications for important events (new messages, price alerts, listing updates)
- System sends notifications for all critical actions

3.6 PAYMENT PROCESSING

a) Secure Payments

- Buyers can make payments through integrated payment gateway.
- System supports multiple payment methods (credit card, debit card).
- Platform processes payment and generates invoice.

b) Transaction Management

- System logs all transactions for audit purposes
- Users receive payment confirmation and receipt.
- System maintains transaction history for both buyers and sellers

3.7 ADMINISTRATIVE FUNCTIONS

a) Listing Moderation

- Admins can review and approve new listings
- Admins can flag or remove inappropriate listings

b) User Management

- Admins can view all registered users
- Admins can suspend or ban accounts

4. PROPOSED NON-FUNCTIONAL REQUIREMENTS:

4.1 PERFORMANCE

- System shall load pages within 2 seconds under normal conditions
- Search queries shall return results within 3 seconds
- System shall be scalable.

4.2 SECURITY

a) Authentication and Authorization

- System shall implement secure password hashing
- System shall use JWT tokens for API authentication
- Role-based access control shall restrict unauthorized access

b) Data Protection

- All sensitive data shall be encrypted.
- System shall implement SQL injection prevention.
- User personal information shall not be shared without consent

4.3 RELIABILITY

- System shall handle errors gracefully with user-friendly messages
- Critical errors shall be logged for debugging
- System shall recover from failures without data loss

4.4 USABILITY

- Interface should be intuitive, consistent, and responsive across all devices.
- Error messages should be clear and helpful.

4.5 MAINTAINABILITY

- System architecture should allow easy updates and feature extensions.
- Components shall be loosely coupled and highly cohesive

4.6 COMPATIBILITY

- System shall support latest versions of Chrome, Firefox, Safari, and Edge.
- System shall be compatible with PostgreSQL or whatever database we use.
- System shall integrate with standard payment gateways.

5. SYSTEM CONSTRAINTS:

a) Technical Constraints:

- Development using Python/Django framework
- PostgreSQL as primary database
- Cloud deployment.

b) Project Constraints:

- Project timeline: One month
- Budget limitations for third-party services
- Team size: 4 members

6. IMPLEMENTATION STATUS AND DESIGN VALIDATION

a) Functional Requirements

Requirement ID	Module	Key Features Implemented	Design Patterns Used	Implementation Status
FR 3.1	User Management	Registration with unique email, login via email/username, profile edit	Proxy (access checks)	Fully Implemented (<i>profile picture & password reset missing</i>)
FR 3.2	Car Listing Management	Create, edit, delete, mark sold, image upload, approval workflow	Factory, Proxy	Fully Implemented
FR 3.3	Search & Filter	Multi-criteria search, sorting, currency-based price filter	Strategy, Adapter	Fully Implemented
FR 3.4	Price Alert System	Follow cars, notify on price change	Observer	Partially Implemented (<i>no criteria-based alerts</i>)
FR 3.5	Communication	Email & WhatsApp contact, in-app notifications	Observer	Fully Implemented
FR 3.6	Payment Processing	Mock payment flow, transaction logs, invoices, order status	—	Half Implemented (<i>no real gateway</i>)
FR 3.7	Administrative Functions	Listing approval/rejection, user viewing	Proxy	Fully Implemented (<i>no ban/suspend</i>)

b) Non-Functional Requirements

NFR ID	Category	Implementation Summary	Status
NFR 4.1	Performance	Basic ORM queries, no caching or monitoring	Partially Implemented
NFR 4.2	Security	Password hashing, RBAC, ORM-based SQL protection	Partially Implemented (<i>no JWT, no encryption</i>)
NFR 4.3	Reliability	Graceful error handling, transactional integrity	Implemented
NFR 4.4	Usability	Responsive UI, consistent navigation, clear messages	Implemented
NFR 4.5	Maintainability	Modular architecture, extensive design pattern usage	Implemented
NFR 4.6	Compatibility	Modern browser support, SQLite/MySQL support	Partially Implemented

c) Design Patterns Summary

Pattern	Purpose	Used In
Factory	Create different car types dynamically	Car creation
Singleton	Single shared configuration instance	Database configuration
Observer	Notify users on price changes	Price alerts, notifications
Strategy	Flexible search logic	Search & filtering
Decorator	Add optional car features with cost	Car feature pricing
Proxy	Role-based access control	Listing & admin actions
Adapter	Currency conversion	Price display & filtering

c) Screenshots of code & UI

i) Functional Requirements

```
74  def signup(request):
75      # If user is already logged in, redirect to home
76      if request.user.is_authenticated:
77          return redirect('home')
78
79      if request.method == 'POST':
80          form = SignUpForm(request.POST)
81          if form.is_valid():
82              user = form.save()
83              login(request, user)
84              messages.success(request, "Registration successful!")
85              return redirect('home')
86          else:
87              form = SignUpForm()
88      return render(request, 'registration/signup.html', {'form': form})
```

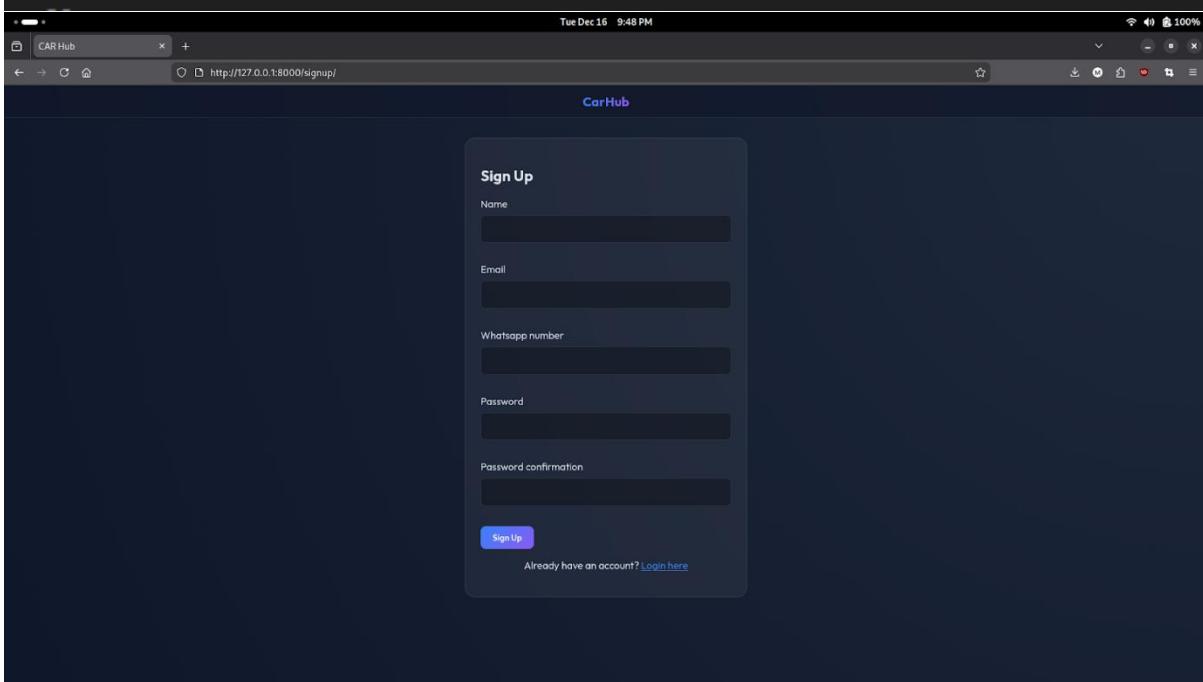


Fig.6.1.1: User Registration

```

25 def custom_login(request):
26     print("DEBUG: custom_login view called")
27     if request.user.is_authenticated:
28         print("DEBUG: User already authenticated, redirecting to home")
29         return redirect('home')
30
31     if request.method == 'POST':
32         print(["DEBUG: POST request received"])
33         from django.contrib.auth import authenticate, login as auth_login
34         username_or_email = request.POST.get('username')
35         password = request.POST.get('password')
36         print(f"DEBUG: Attempting login for: {username_or_email}")
37
38         # Try to authenticate with username first
39         user = authenticate(request, username=username_or_email, password=password)
40
41         # If authentication failed and input looks like an email, try to find user by email
42         if user is None and '@' in username_or_email:
43             print("DEBUG: Input looks like email, trying to find user by email")
44             try:
45                 from django.contrib.auth.models import User
46                 user_obj = User.objects.get(email=username_or_email)
47                 # Now try to authenticate with the username
48                 user = authenticate(request, username=user_obj.username, password=password)
49             except User.DoesNotExist:
50                 print("DEBUG: No user found with that email")
51                 user = None
52
53         if user is not None:
54             print("DEBUG: Authentication successful")
55             auth_login(request, user)
56             messages.success(request, f"Welcome back, {user.first_name or user.username}!")
57             print("DEBUG: Success message added")
58             return redirect('home')
59         else:
60             print("DEBUG: Authentication failed")
61             messages.error(request, "Invalid username/email or password. Please try again.")
62             print("DEBUG: Error message added")
63             return render(request, 'registration/login.html')
64
65     print("DEBUG: Rendering login form (GET request)")
66     return render(request, 'registration/login.html')
67

```

From: views.py

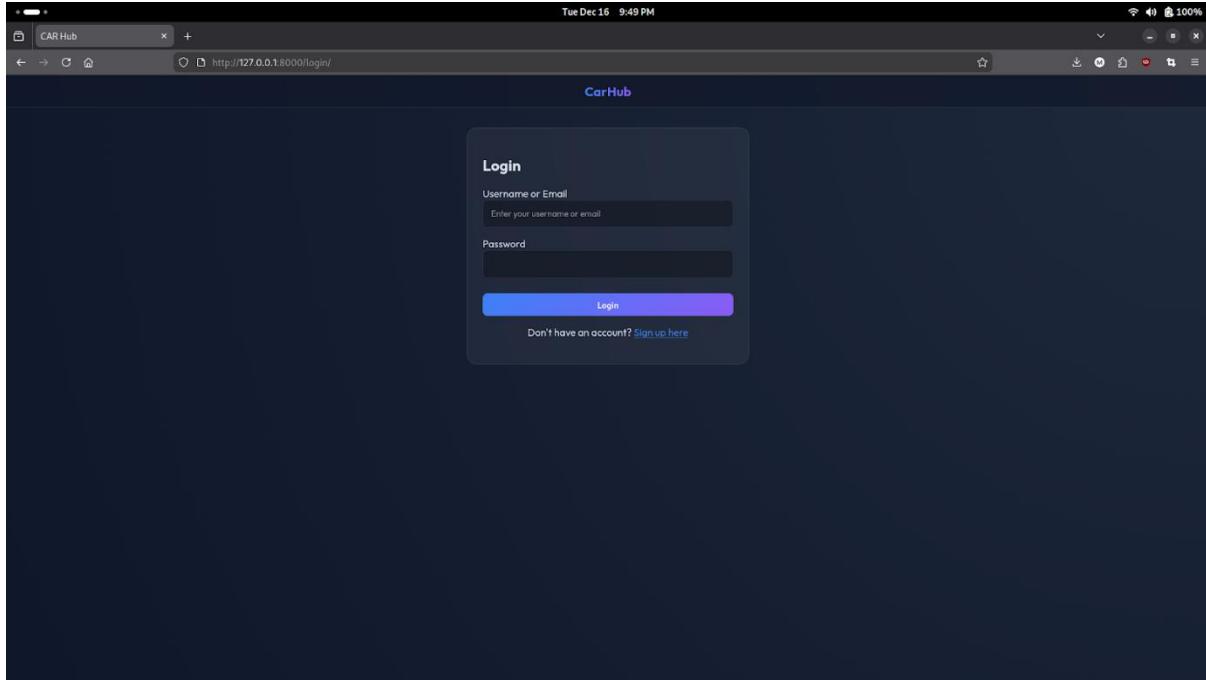


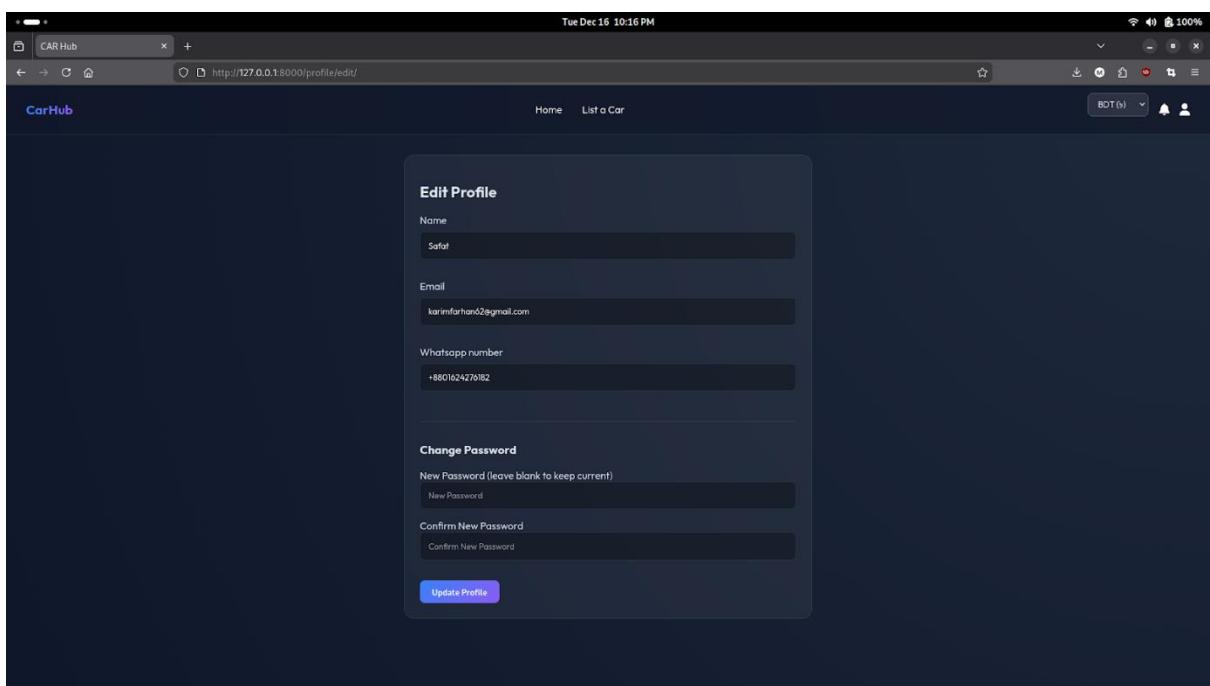
Fig.6.1.2: User Authentication

```
680     @login_required
681     def edit_profile(request):
682         if request.method == 'POST':
683             form = EditProfileForm(request.POST, instance=request.user)
684             if form.is_valid():
685                 user = form.save()
686
687             # Handle Password Change
688             new_password = request.POST.get('new_password')
689             confirm_password = request.POST.get('confirm_password')
690
691             if new_password:
692                 if new_password == confirm_password:
693                     user.set_password(new_password)
694                     user.save()
695                     login(request, user)
696                     messages.success(request, "Profile and password updated successfully!")
697                 else:
698                     messages.error(request, "Passwords do not match.")
699                     return render(request, 'cars/edit_profile.html', {'form': form})
700             else:
701                 messages.success(request, "Profile updated successfully!")
702
703             return redirect('profile')
704         else:
705             form = EditProfileForm(instance=request.user)
706
707     return render(request, 'cars/edit_profile.html', {'form': form})
```

From: views.py

```
6  <div class="card" style="max-width: 800px; margin: 0 auto;">
7    <h2>My Profile</h2>
8    <div style="margin-bottom: 2rem;">
9      <p><strong>Name:</strong> {{ user.first_name }}</p>
10     <p><strong>Email:</strong> {{ user.email }}</p>
11     {% if user.profile.whatsapp_number %}
12       <p><strong>WhatsApp:</strong> {{ user.profile.whatsapp_number }}</p>
13     {% endif %}
14     <p><strong>Date Joined:</strong> {{ user.date_joined|date:"F j, Y" }}</p>
15     <a href="{% url 'edit_profile' %}" class="btn btn-primary" style="margin-top:1rem;">Edit Profile</a>
16   </div>
```

From: profile.html



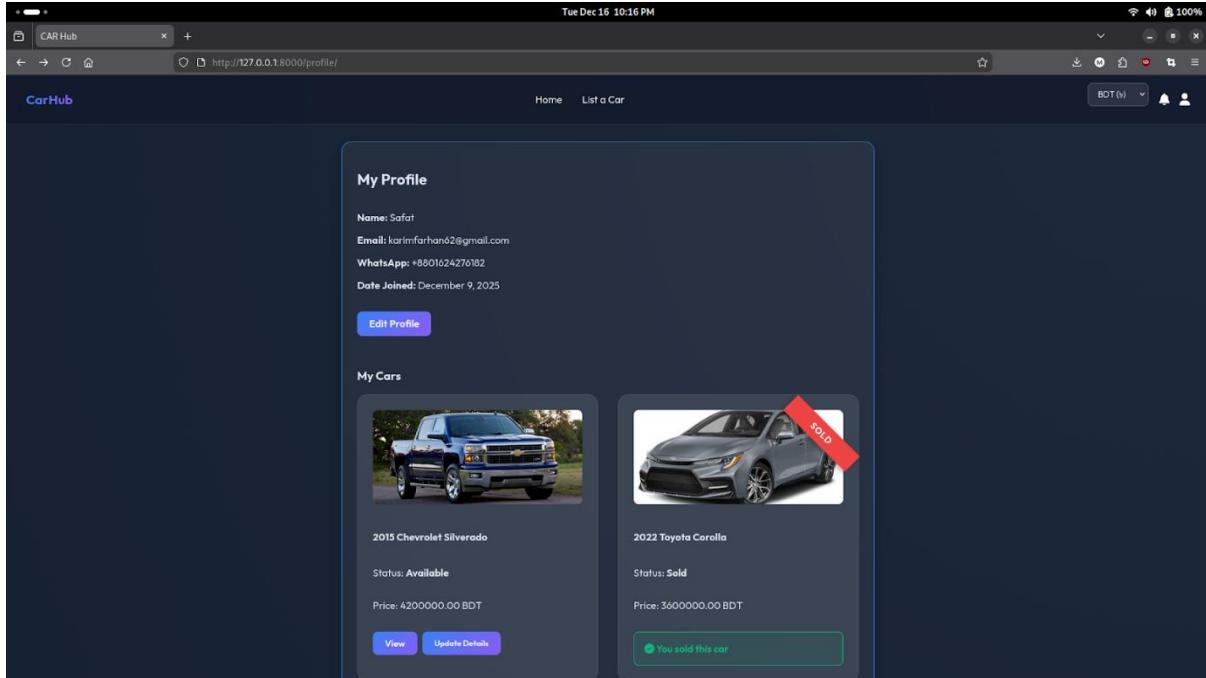


Fig.6.1.3: User Profile Management

```

382     @login_required
383     def create_car(request):
384         # Proxy Pattern check
385         proxy = CarAccessProxy(request.user)
386         allowed, msg = proxy.post_car({})
387         if not allowed:
388             messages.error(request, msg)
389             return redirect('home')
390
391         if request.method == 'POST':
392             make = request.POST.get('make')
393             model = request.POST.get('model')
394             year = request.POST.get('year')
395             price_input = float(request.POST.get('price'))
396             currency_input = request.POST.get('currency', 'BDT')
397             mileage = request.POST.get('mileage')
398             car_type = request.POST.get('car_type')
399             contact_email = request.POST.get('contact_email', '').strip()
400             contact_whatsapp = request.POST.get('contact_whatsapp', '').strip()
401             images = request.FILES.getlist('images')
402             registration_paper = request.FILES.get('registration_paper')
403
404             # Validate Registration Paper
405             if not registration_paper:
406                 messages.error(request, "Registration paper is required. Please upload the vehicle registration document.")
407                 return render(request, 'cars/create.html')
408
409             # Validate Contact Info
410             if not contact_email and not contact_whatsapp:
411                 messages.error(request, "You must provide at least one contact method (Email or WhatsApp).")
412                 return render(request, 'cars/create.html')
413
414             # Validate email format
415             if contact_email:
416                 from django.core.validators import validate_email
417                 from django.core.exceptions import ValidationError
418                 try:
419                     validate_email(contact_email)
420                 except ValidationError:
421                     messages.error(request, "Invalid email address format.")
422                     return render(request, 'cars/create.html')
423
424             # Check if domain has valid MX records
425             if not validate_email_domain(contact_email):
426                 messages.error(request, "This email domain does not exist or cannot receive emails. Please check your email address.")
427                 return render(request, 'cars/create.html')

```

```
    # Validate WhatsApp format
429    if contact_whatsapp:
430        is_valid, error_msg = validate_whatsapp_number(contact_whatsapp)
431        if not is_valid:
432            messages.error(request, error_msg)
433            return render(request, 'cars/create.html')
434
435    # Validate registration paper format
436    if not registration_paper.name.endswith('.pdf'):
437        messages.error(request, "Registration paper must be a PDF file.")
438        return render(request, 'cars/create.html')
439
440    # Convert to BDT
441    price_bdt = CurrencyAdapter.convert_to_bdt_static(price_input, currency_input)
442
443    if len(images) < 1:
444        messages.error(request, "You must upload at least 1 image.")
445        return render(request, 'cars/create.html')
446
447    if len(images) > 5:
448        messages.error(request, "You can upload a maximum of 5 images.")
449        return render(request, 'cars/create.html')
450
451    # Factory Pattern
452    factory = None
453    if car_type == 'sedan':
454        factory = SedanFactory()
455    elif car_type == 'suv':
456        factory = SUVFactory()
457    elif car_type == 'truck':
458        factory = TruckFactory()
459    elif car_type == 'coupe':
460        factory = CoupeFactory()
461
462    if factory:
463        car = factory.create_car(make, model, year, price_bdt, mileage, request.user)
464
465        # Save contact info
466        car.contact_email = contact_email
467        car.contact_whatsapp = contact_whatsapp
468
469        # Save registration paper (required)
470        car.registration_paper = registration_paper
471
472        car.save()
473
474
```

```

473     car.save()
474
475     # Save images
476     for image in images:
477         CarImage.objects.create(car=car, image=image)
478
479     messages.success(request, "Request for listing Car is sent successfully!")
480     return redirect('home')
481
482 import datetime
483 current_year = datetime.date.today().year
484 year_range = range(current_year, 1939, -1)
485
486     # Get user's contact info from profile
487 user_email = request.user.email
488 user_whatsapp = ''
489 if hasattr(request.user, 'profile'):
490     user_whatsapp = request.user.profile.whatsapp_number or ''
491
492 return render(request, 'cars/create.html', {
493     'year_range': year_range,
494     'user_email': user_email,
495     'user_whatsapp': user_whatsapp
496 })
497

```

From: views.py

The screenshot shows a web browser window titled 'CAR Hub'. The URL in the address bar is <http://127.0.0.1:8000/create/>. The page displays a 'List a Car' form. The form fields are as follows:

- Make:** (Input field)
- Model:** (Input field)
- Manufacture Year:** (Dropdown menu set to 2025)
- Price:** (Input field containing 100000, with a BDT suffix button)
- Mileage:** (Input field)
- Type:** (Dropdown menu set to Sedan)
- Contact information:**
 - A note: At least one contact method is required.
 - Contact Email:** (Input field containing karimforhan02@gmail.com)
 - WhatsApp Number:** (Input field containing +8801624276182)
- Registration Paper (Required PDF):** (Note at the bottom of the form)

Fig.6.2.1 Create Listings

```

498     @login_required
499     def delete_car(request, car_id):
500         # Proxy Pattern
501         proxy = CarAccessProxy(request.user)
502         success, msg = proxy.delete_car(car_id)
503
504         if success:
505             messages.success(request, msg)
506         else:
507             messages.error(request, msg)
508
509         return redirect('home')
510
511     @login_required
512     def update_car(request, car_id):
513         car = get_object_or_404(Car, id=car_id)
514
515         if request.user != car.owner:
516             messages.error(request, "You are not authorized to edit this car.")
517             return redirect('car_detail', car_id=car.id)
518
519         # Prevent editing if car is sold
520         if car.status == 'sold':
521             messages.error(request, "You cannot edit a car that has been sold.")
522             return redirect('profile')
523
524         if request.method == 'POST':
525             # Get fields
526             make = request.POST.get('make')
527             model = request.POST.get('model')
528             year = request.POST.get('year')
529             price_input = float(request.POST.get('price'))
530             currency_input = request.POST.get('currency', 'BDT')
531             mileage = request.POST.get('mileage')
532             car_type = request.POST.get('car_type')
533             contact_email = request.POST.get('contact_email', '').strip()
534             contact_whatsapp = request.POST.get('contact_whatsapp', '').strip()
535             images = request.FILES.getlist('images')
536             registration_paper = request.FILES.get('registration_paper')
537
538             # Validate Contact Info
539             if not contact_email and not contact_whatsapp:
540                 messages.error(request, "You must provide at least one contact method (Email or WhatsApp).")
541                 import datetime
542                 current_year = datetime.date.today().year
543                 year_range = range(current_year, 1939, -1)
544                 return render(request, 'cars/update.html', {'car': car, 'year_range': year_range})
545
546             # Validate email format
547             if contact_email:
548                 from django.core.validators import validate_email
549                 from django.core.exceptions import ValidationError
550                 try:
551                     validate_email(contact_email)
552                 except ValidationError:
553                     messages.error(request, "Invalid email address format.")
554                     import datetime
555                     current_year = datetime.date.today().year
556                     year_range = range(current_year, 1939, -1)
557                     return render(request, 'cars/update.html', {'car': car, 'year_range': year_range})
558

```

```

558     # Check if domain has valid MX records
559     if not validate_email_domain(contact_email):
560         messages.error(request, "This email domain does not exist or cannot receive emails. Please check your email address.")
561         import datetime
562         current_year = datetime.date.today().year
563         year_range = range(current_year, 1939, -1)
564         return render(request, 'cars/update.html', {'car': car, 'year_range': year_range})
565
566
567     # Validate WhatsApp format
568     if contact_whatsapp:
569         is_valid, error_msg = validate_whatsapp_number(contact_whatsapp)
570         if not is_valid:
571             messages.error(request, error_msg)
572             import datetime
573             current_year = datetime.date.today().year
574             year_range = range(current_year, 1939, -1)
575             return render(request, 'cars/update.html', {'car': car, 'year_range': year_range})
576
577     # Convert to BDT
578     new_price_bdt = CurrencyAdapter.convert_to_bdt_static(price_input, currency_input)
579
580     # Check if price changed for notification
581     old_price = float(car.price)
582     price_changed = abs(old_price - new_price_bdt) > 1.0 # Float comparison
583
584     # Update fields
585     car.make = make
586     car.model = model
587     car.year = year
588     car.mileage = mileage
589     car.car_type = car_type
590     car.contact_email = contact_email
591     car.contact_whatsapp = contact_whatsapp
592
593     # Note: Registration paper cannot be changed after listing
594
595     # Save all fields except handle price change through Observer pattern
596     if not price_changed:
597         car.price = new_price_bdt
598
599     car.save()

599     car.save()

600
601     # Add new images
602     for image in images:
603         CarImage.objects.create(car=car, image=image)
604
605     # Notify followers if price changed (BEFORE updating price in DB)
606     if price_changed:
607         # Reload car from database to get the current saved price
608         car.refresh_from_db()
609
610         # Create subject with the car (car has old price from DB)
611         subject = CarPriceSubject(car)
612
613         # Attach all followers as observers (proper Observer pattern)
614         for follower in car.followers.all():
615             observer = UserObserver(follower)
616             subject.attach(observer)
617
618         # Change price and notify all attached observers
619         subject.change_price(new_price_bdt)
620
621         messages.success(request, "Car details updated successfully!")
622         return redirect('car_detail', car_id=car.id)
623
624     import datetime
625     current_year = datetime.date.today().year
626     year_range = range(current_year, 1939, -1)
627     return render(request, 'cars/update.html', {'car': car, 'year_range': year_range})
628

```

```

143     @login_required
144     def update_car_status(request, car_id):
145         car = get_object_or_404(Car, id=car_id)
146
147         if request.user != car.owner:
148             messages.error(request, "You are not authorized to perform this action.")
149             return redirect('home')
150
151         if request.method == 'POST':
152             new_status = request.POST.get('status')
153             if new_status in ['available', 'sold']:
154                 car.status = new_status
155                 car.save()
156                 messages.success(request, f"Car status updated to {new_status}.")
157
158         return redirect('car_detail', car_id=car.id)

```

From: views.py

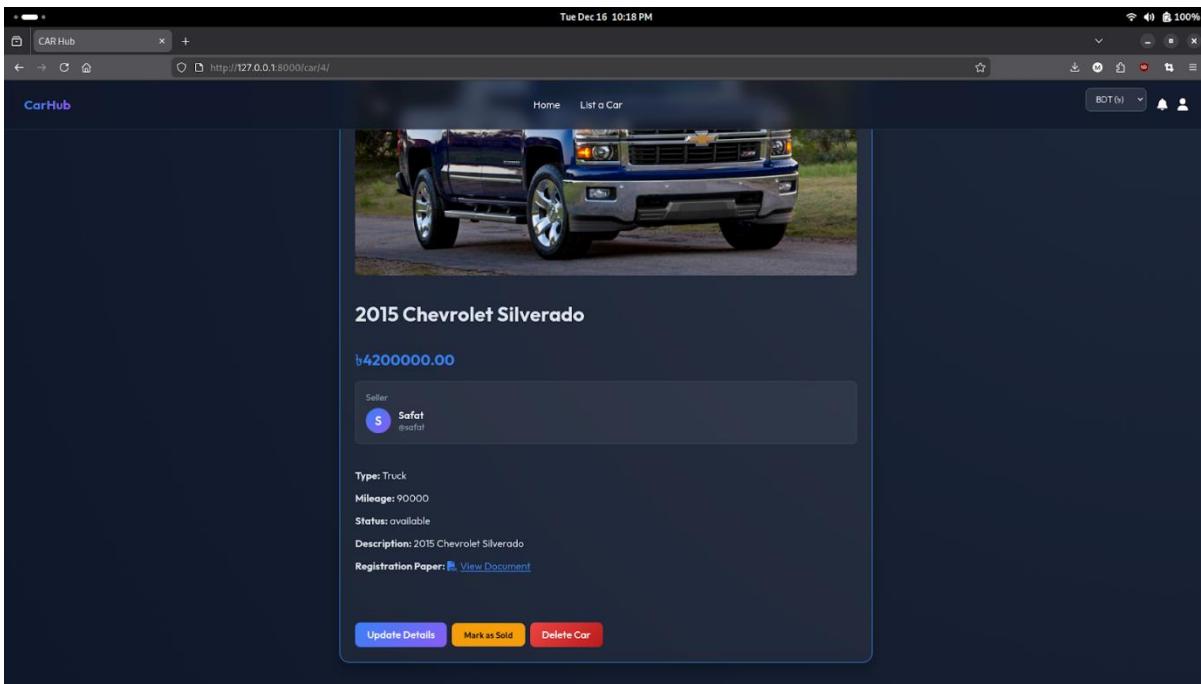


Fig.6.2.2: Edit and Delete Listings

```
221 @login_required
222 def home(request):
223     # Require authentication for home page
224     if not request.user.is_authenticated:
225         return redirect('welcome')
226
227     # Redirect admin to dashboard
228     if request.user.is_superuser:
229         return redirect('admin_dashboard')
230
231     # Singleton usage (just for demo)
232     db_config = DatabaseConfigManager().get_config()
233
234     # Only show approved cars
235     cars = Car.objects.filter(approval_status='approved')
236
237     # Strategy Pattern
238     search_type = request.GET.get('search_type')
239     query = request.GET.get('query')
240
241     from .patterns.strategy import ModelSearchStrategy
242
243     if search_type:
244         context = None
245         if search_type == 'price':
246             min_price = request.GET.get('min_price')
247             max_price = request.GET.get('max_price')
248             if min_price and max_price:
249                 try:
250                     # Get current currency
251                     currency = request.GET.get('currency', request.session.get('currency', 'BDT'))
252
253                     # Convert input prices (in selected currency) to BDT
254                     min_price_bdt = CurrencyAdapter.convert_to_bdt_static(float(min_price), currency)
255                     max_price_bdt = CurrencyAdapter.convert_to_bdt_static(float(max_price), currency)
256
257                     context = CarSearchContext(PriceSearchStrategy())
258                     cars = context.execute_search([min_price_bdt, max_price_bdt])
259                     # Filter to only approved cars
260                     cars = cars.filter(approval_status='approved')
261                 except ValueError:
262                     pass
263             elif search_type == 'brand':
264                 query = request.GET.get('query')
265                 if query:
266                     context = CarSearchContext(BrandSearchStrategy())
```

```
266     context = CarSearchContext(BrandSearchStrategy())
267     cars = context.execute_search(query)
268     # Filter to only approved cars
269     cars = cars.filter(approval_status='approved')
270 elif search_type == 'model':
271     query = request.GET.get('query')
272     if query:
273         context = CarSearchContext(ModelSearchStrategy())
274         cars = context.execute_search(query)
275         # Filter to only approved cars
276         cars = cars.filter(approval_status='approved')
277 elif search_type == 'mileage':
278     min_mileage = request.GET.get('min_mileage')
279     max_mileage = request.GET.get('max_mileage')
280     if min_mileage and max_mileage:
281         try:
282             context = CarSearchContext(MileageSearchStrategy())
283             cars = context.execute_search([int(min_mileage), int(max_mileage)])
284             # Filter to only approved cars
285             cars = cars.filter(approval_status='approved')
286         except ValueError:
287             pass
288 elif search_type == 'type':
289     query = request.GET.get('query')
290     if query:
291         context = CarSearchContext(TypeSearchStrategy())
292         cars = context.execute_search(query)
293         # Filter to only approved cars
294         cars = cars.filter(approval_status='approved')
295 elif search_type == 'year':
296     query = request.GET.get('query')
297     if query:
298         try:
299             # Single year selection
300             context = CarSearchContext(YearSearchStrategy())
301             cars = context.execute_search([int(query), int(query)])
302             # Filter to only approved cars
303             cars = cars.filter(approval_status='approved')
304         except ValueError:
305             pass
```

From: views.py

```

20 <div style="margin-bottom: 1.5rem; padding: 1rem; background: #rgba(255,255,255,0.05); border-radius: 0.5rem; border: 1px solid var(--glass-border);">
21   <p style="margin: 0; color: #94a3b8; font-size: 0.9rem;">Seller</p>
22   <a href="#${url}'seller_profile' car.owner.id ${}" style="display: flex; align-items: center; gap: 0.75rem; text-decoration: none; color: white; margin-top: 0.5rem;">
23     <div style="width: 40px; height: 40px; border-radius: 50%; background: linear-gradient(135deg, var(--primary-color), var(--secondary-color)); display: flex; align-items: center; justify-content: center; font-size: 0.8rem; font-weight: bold; color: white; font-family: inherit; text-align: center; border: 1px solid var(--primary-color);">
24       {{ car.owner.first_name.0|upper }}
25     </div>
26   <div>
27     <p style="margin: 0; font-weight: 600; font-size: 1rem;">{{ car.owner.first_name }}</p>
28     <p style="margin: 0; color: #94a3b8; font-size: 0.85rem;">@{{ car.owner.username }}</p>
29   </div>
30 </a>
31 </div>
32
33 <div style="margin-bottom: 2rem;">
34   <p><strong>Type:</strong> {{ car.get_car_type_display }}</p>
35   <p><strong>Mileage:</strong> {{ car.mileage }}</p>
36   <p><strong>Status:</strong> {{ car.status }}</p>
37   <p><strong>Description:</strong> {{ description }}</p>
38   <div style="display: flex; align-items: center; gap: 0.5rem;">
39     <strong>Registration Paper:</strong>
40     <a href="{{ car.registration_paper.url }}" target="_blank"
41       style="color: var(--primary-color); text-decoration: underline; display: inline-block; align-items: center; gap: 0.3rem;">
42       <i class="fa-solid fa-file-pdf"></i> View Document
43     </a>
44   </div>
45   <% endif %>
46 </div>
47
48 <% else %>
49   <div style="width:100%; margin-bottom:1rem;">
50     <h3 style="margin-bottom: 0.5rem; font-size: 1.2rem;">Seller Contact</h3>
51     <div style="display:flex; gap:1rem;">
52       <% if car.contact_email %>
53         <a href="mailto:{{ car.contact_email }}" class="btn"
54           style="background:white; color:black; display:flex; align-items:center; gap:0.5rem; font-weight:600;">
55           <i class="fa-solid fa-envelope"></i> Email Seller
56         </a>
57       <% endif %>
58       <% if car.contact_whatsapp %>
59         <a href="https://wa.me/{{ car.contact_whatsapp }}" target="_blank" class="btn"
60           style="background:white; color:black; display:flex; align-items:center; gap:0.5rem; font-weight:600;">
61           <i class="fa-brands fa-whatsapp"></i> WhatsApp
62         </a>
63       <% endif %>
64     </div>
65   </div>
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214

```

From: detail.html

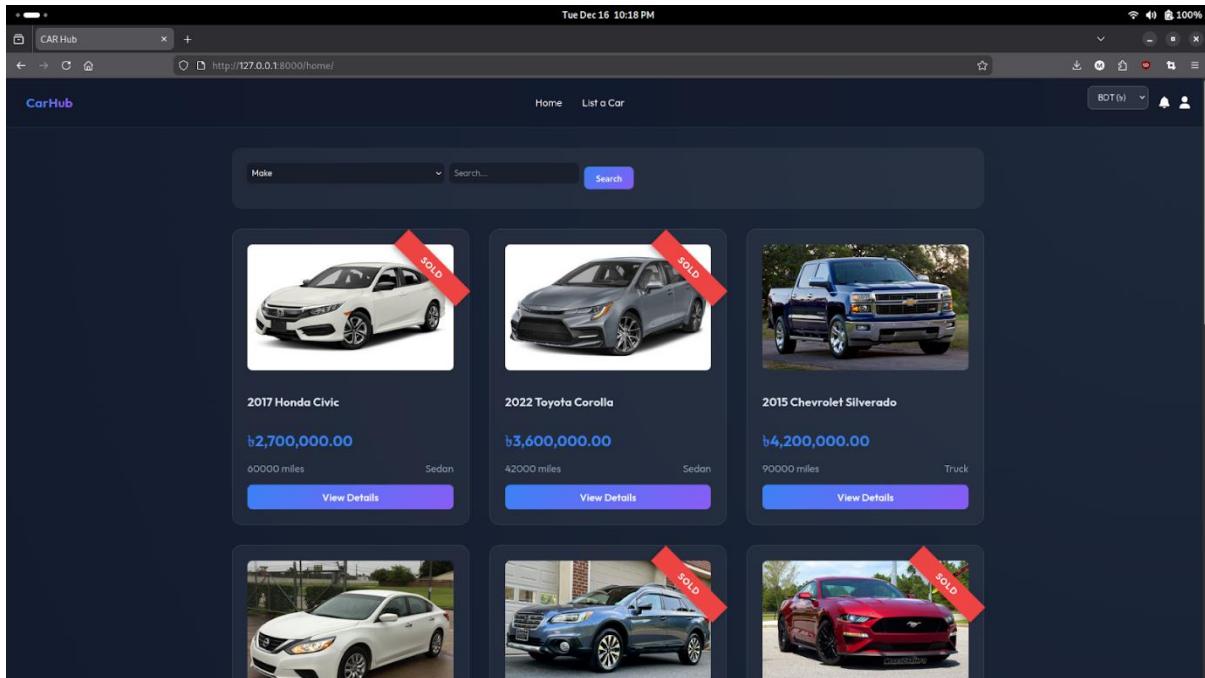


Fig.:6.2.3: View Listings

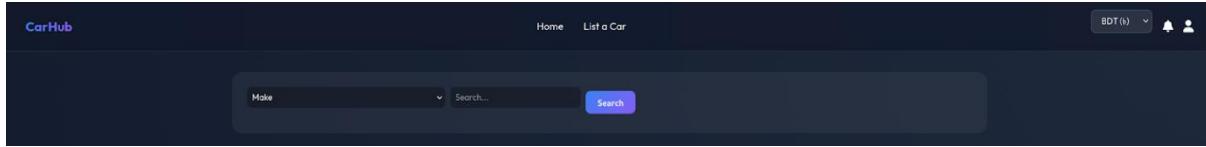


Fig.6.3.1: Search, Filter (code in views.py234-303, 246-257, 232)

```

753     @login_required
754     def follow_car(request, car_id):
755         car = get_object_or_404(Car, id=car_id)
756         if request.user in car.followers.all():
757             car.followers.remove(request.user)
758             messages.success(request, f"You have unfollowed this {car.make} {car.model}.")
759         else:
760             car.followers.add(request.user)
761             messages.success(request, f"You are now following this {car.make} {car.model}. You will be notified of updates.")
762         return redirect('car_detail', car_id=car.id)

```

From: views.py

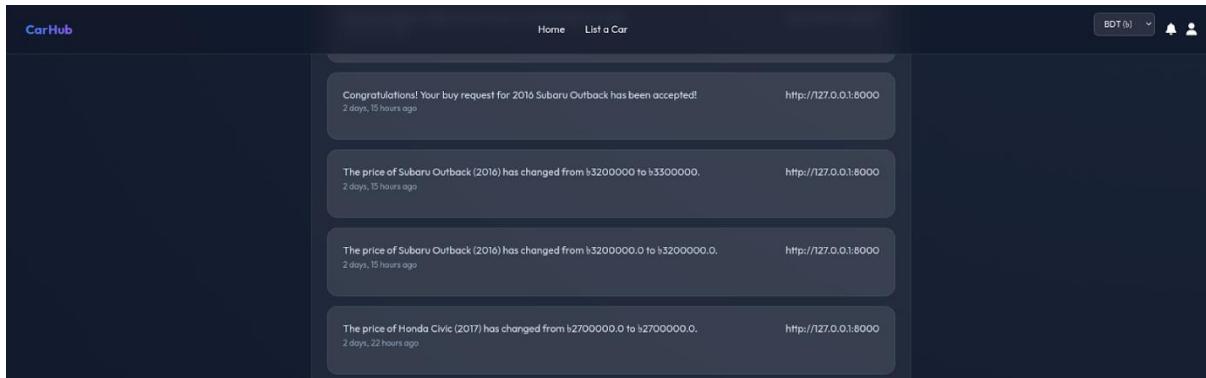


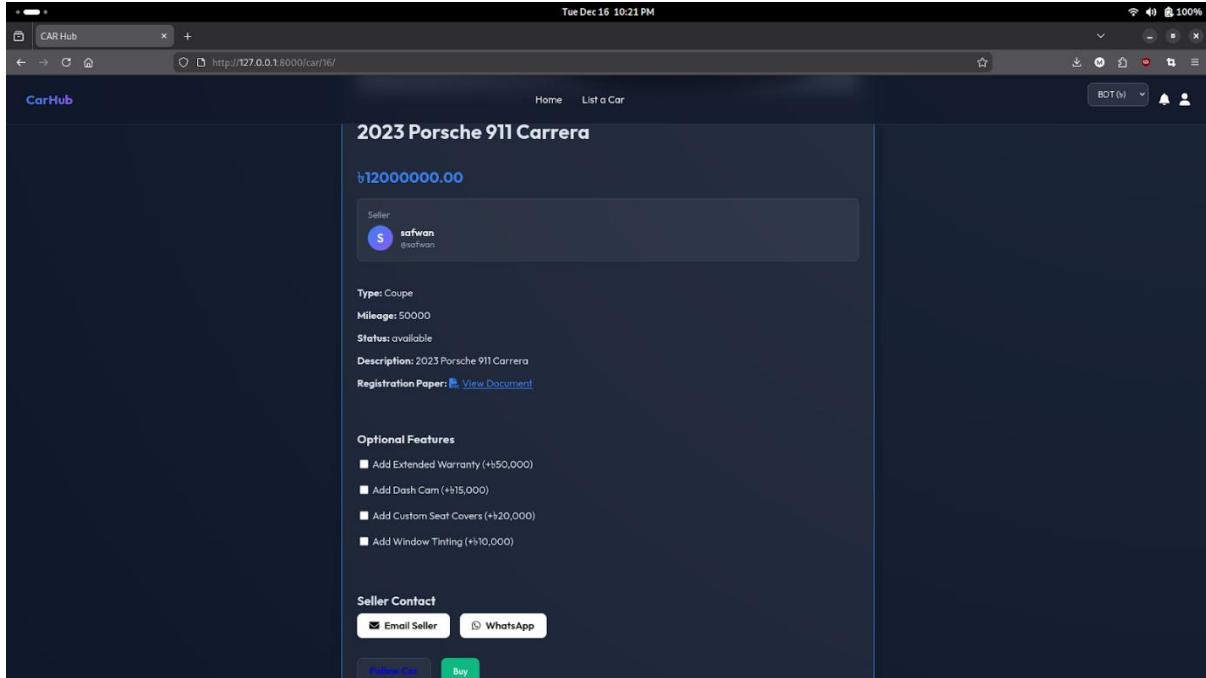
Fig: 6.4.1: Subscribe to Alerts

```

198     <div style="width:100%; margin-bottom:1rem;">
199         <h3 style="margin-bottom: 0.5rem; font-size: 1.2rem;">Seller Contact</h3>
200         <div style="display:flex; gap:1rem;">
201             {% if car.contact_email %}
202                 <a href="mailto:{{ car.contact_email }}" class="btn"
203                     style="background:white; color:black; display:flex; align-items:center; gap:0.5rem; font-weight:600;">
204                     <i class="fa-solid fa-envelope"></i> Email Seller
205                 </a>
206             {% endif %}
207             {% if car.contact_whatsapp %}
208                 <a href="https://wa.me/{{ car.contact_whatsapp }}" target="_blank" class="btn"
209                     style="background:white; color:black; display:flex; align-items:center; gap:0.5rem; font-weight:600;">
210                     <i class="fa-brands fa-whatsapp"></i> WhatsApp
211                 </a>
212             {% endif %}
213         </div>
214     </div>

```

From: detail.html



```
class Notification(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    message = models.TextField()
    is_read = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

132    # Notify Owner
133    Notification.objects.create(
134        user=car.owner,
135        message=f"New Buy Request: {request.user.username} wants to buy your {car.year} {car.make} {car.model}."
136    )
137
138    messages.success(request, "Buy request sent! Proceed to payment.")
139
140    # Redirect to profile page
141    return redirect('profile')
142
143 def notifications(request):
144     if not request.user.is_authenticated:
145         return redirect('login')
146     notifs = Notification.objects.filter(user=request.user).order_by('-created_at')
147     unread_count = notifs.filter(is_read=False).count()
148
149     # Get admin contact info
150     admin_user = User.objects.filter(is_superuser=True).first()
151     admin_email = admin_user.email if admin_user else None
152     admin_whatsapp = admin_user.profile.whatsapp_number if admin_user and hasattr(admin_user, 'profile') else None
153
154     return render(request, 'cars/notifications.html', {
155         'notifications': notifs,
156         'unread_count': unread_count,
157         'admin_email': admin_email,
158         'admin_whatsapp': admin_whatsapp
159     })
160
```

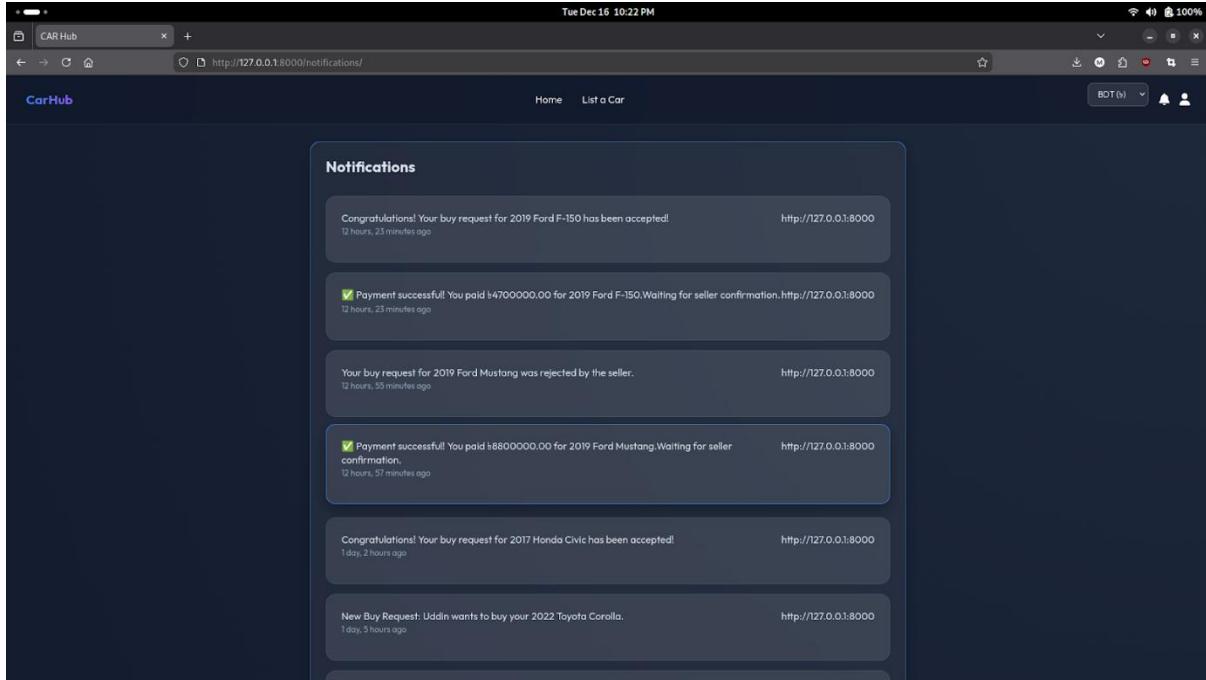


Fig.6.5.1: Contact Seller, Notifications

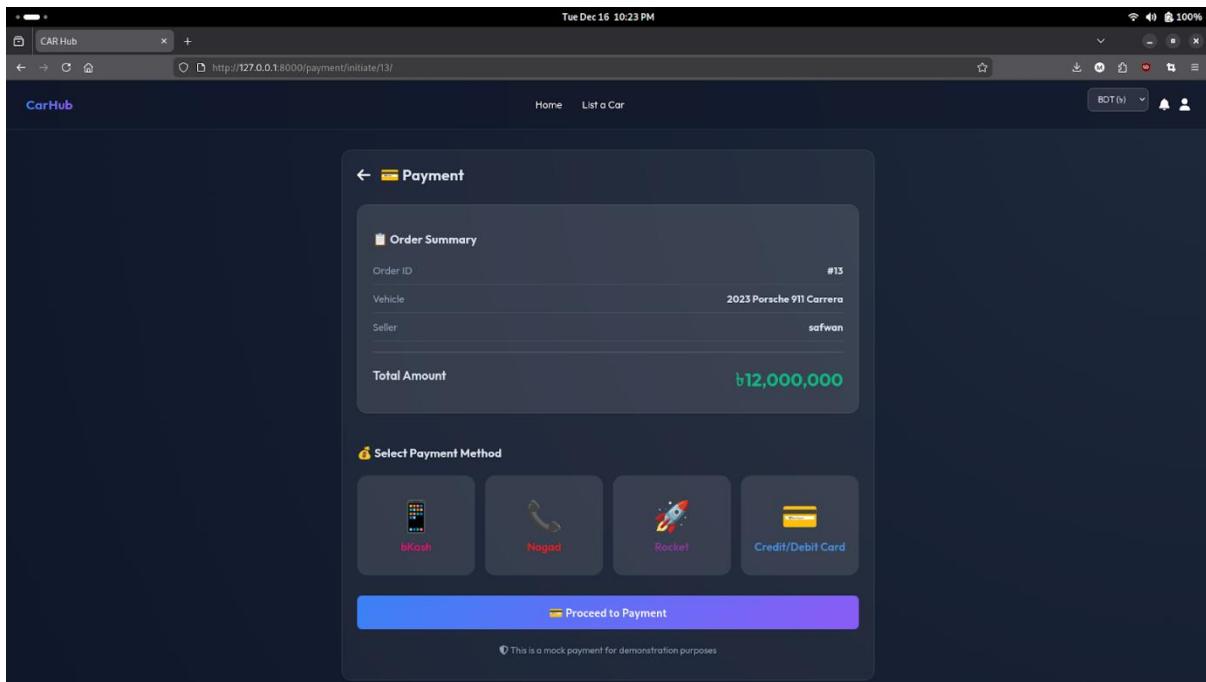


Fig.6.6.1: Payment Processing

```

764     @login_required
765     def admin_dashboard(request):
766         # Only allow superusers to access
767         if not request.user.is_superuser:
768             messages.error(request, "You do not have permission to access the admin dashboard.")
769             return redirect('home')
770
771         # Get statistics
772         from django.db.models import Count
773         total_users = User.objects.filter(is_superuser=False).count()
774         total_cars = Car.objects.count()
775         available_cars = Car.objects.filter(status='available').count()
776         sold_cars = Car.objects.filter(status='sold').count()
777         pending_orders = Order.objects.filter(status='pending').count()
778
779         # Get recent users with their car count (exclude superusers)
780         recent_users = User.objects.filter(is_superuser=False).annotate(car_count=Count('car')).order_by('-date_joined')[::10]
781         recent_users_data = [{"user": user, 'car_count': user.car_count} for user in recent_users]
782
783         # Get recent cars
784         recent_cars = Car.objects.all().order_by('-created_at')[::10]
785
786         # Get recent orders
787         recent_orders = Order.objects.all().order_by('-created_at')[::10]
788
789         return render(request, 'cars/admin_dashboard.html', {
790             'total_users': total_users,
791             'total_cars': total_cars,
792             'available_cars': available_cars,
793             'sold_cars': sold_cars,
794             'pending_orders': pending_orders,
795             'recent_users': recent_users_data,
796             'recent_cars': recent_cars,
797             'recent_orders': recent_orders,
798         })
799
800     @login_required
801     def approve_car(request, car_id):
802         proxy = CarAccessProxy(request.user)
803         success, msg = proxy.approve_car(car_id)
804
805         if success:
806             messages.success(request, msg)
807         else:
808             messages.error(request, msg)
809
810         return redirect('car_detail', car_id=car_id)
811
812     @login_required
813     def reject_car(request, car_id):
814         proxy = CarAccessProxy(request.user)
815         reason = request.POST.get('reason', '') if request.method == 'POST' else ''
816         success, msg = proxy.reject_car(car_id, reason)
817
818         if success:
819             messages.success(request, msg)
820         else:
821             messages.error(request, msg)
822
823         return redirect('admin_dashboard')
824
825         description = models.TextField(blank=True)
826         status = models.CharField(max_length=20, default='available') # available, sold
827         approval_status = models.CharField(max_length=20, default='pending') # pending, approved, rejected
828         owner = models.ForeignKey(User, on_delete=models.CASCADE)

```

```

770
771     # Get statistics
772     from django.db.models import Count
773     total_users = User.objects.filter(is_superuser=False).count()
774     total_cars = Car.objects.count()
775     available_cars = Car.objects.filter(status='available').count()
776     sold_cars = Car.objects.filter(status='sold').count()
777     pending_orders = Order.objects.filter(status='pending').count()
778

```

Tue Dec 16 10:24 PM

Welcome back, admin!

Admin Dashboard

7 Total Users	15 Total Listings	8 Available Cars	7 Sold Cars	1 Pending Orders
---------------	-------------------	------------------	-------------	------------------

Recent Users

Name	Email	Username	Joined	Listings
Iqtedar	farhankarim265@gmail.com	iqtedar	Dec 16, 2025	0
Farhan	farhan.karim03@northsouth.edu	farhan	Dec 16, 2025	1
safwan2	safwan.amin@northsouth.edu	safwan2	Dec 15, 2025	1
safwan	safwanismaun@gmail.com	safwan	Dec 15, 2025	7
Suraya	mekakashi17@gmail.com	suraya	Dec 15, 2025	2
Uddin	luciferthe77@gmail.com	Uddin	Dec 09, 2025	2
Safat	karimfarhan02@gmail.com	safat	Dec 09, 2025	2

Tue Dec 16 10:24 PM

Recent Car Listings

Car	Owner	Price (BDT)	Status	Decision	Listed	Actions
2019 Ford F-150	farhan	৳4700000	Sold	Approved	Dec 16, 2025	<button>View</button> <button>Delete</button>
2025 BMW M5	safwan2	৳12000000	Sold	Approved	Dec 15, 2025	<button>View</button> <button>Delete</button>
2003 Mitsubishi Lancer Evo	safwan	৳1800000	Available	Pending	Dec 15, 2025	<button>View</button> <button>Delete</button>
2023 Porsche 911 Carrera	safwan	৳12000000	Available	Approved	Dec 15, 2025	<button>View</button> <button>Delete</button>
2025 Chevrolet Corvette Stingray	safwan	৳8000000	Sold	Approved	Dec 15, 2025	<button>View</button> <button>Delete</button>
2025 Ford F-150	safwan	৳4800000	Available	Pending	Dec 15, 2025	<button>View</button> <button>Delete</button>
2019 Audi A4 30 TFSI	safwan	৳6900000	Available	Pending	Dec 15, 2025	<button>View</button> <button>Delete</button>
2025 Toyota Cross	safwan	৳570000	Available	Approved	Dec 15, 2025	<button>View</button> <button>Delete</button>
2024 Tesla Model 3	safwan	৳3600000	Available	Pending	Dec 15, 2025	<button>View</button> <button>Delete</button>
2019 Ford Mustang	Uddin	৳9000000	Sold	Approved	Dec 13, 2025	<button>View</button> <button>Delete</button>

The screenshot shows the Admin Dashboard for the CAR Hub application. At the top, there are two car listings: a 2024 Tesla Model S and a 2019 Ford Mustang. Below these are sections for 'Recent Orders' and 'Recent Orders'.

Order ID	Car	Buyer	Seller	Status	Additional Features	Date
#15	2023 Porsche 911 Carrera	safwan	safwan	Paid	-	Dec 16, 2025
#12	2019 Ford Mustang	iqtedar	Uddin	Completed	-	Dec 16, 2025
#11	2019 Ford F-150	safwan	farhan	Completed	-	Dec 16, 2025
#10	2019 Ford Mustang	safwan	Uddin	Cancelled	-	Dec 16, 2025
#9	2025 BMW M5	safwan	safwan2	Completed	-	Dec 15, 2025
#8	2023 Porsche 911 Carrera	safwan2	safwan	Pending	-	Dec 15, 2025
#7	2025 Toyota Cross	safwan2	safwan	Cancelled	-	Dec 15, 2025
#6	2017 Honda Civic	safwan	suraya	Completed	✓ Seat Covers ✓ Tinting	Dec 15, 2025
#5	2025 Chevrolet Corvette Stingray	safwan2	safwan	Completed	-	Dec 15, 2025
#4	2022 Toyota Corolla	Uddin	safwan	Completed	✓ Warranty ✓ Dashcam	Dec 15, 2025

Fig..6.7.1: Admin Dashboard

ii) Non-Functional Requirements

The screenshot shows the SQLite Viewer interface displaying the contents of the 'auth_user' table from the 'db.sqlite3' database. The table has columns: id, password, last_login, and a primary key column. The data shows 9 rows of user entries.

	id	password	last_login
1	1	pbkdf2_sha256\$120000\$IPKUKV2C3OG6tLQfbPwtz...	2025-12-16 16:24:16.78...
2	2	pbkdf2_sha256\$120000\$BcjHlyZqzBslOrrSwaTpR\$...	2025-12-16 16:25:28.83...
3	3	pbkdf2_sha256\$120000\$cAKDRvr3LW57eMDC122do...	2025-12-16 15:30:21.22...
4	4	pbkdf2_sha256\$120000\$81H8Mmsvq3DElt5SIzbE5...	2025-12-15 14:11:06.94...
5	5	pbkdf2_sha256\$120000\$8X4kvZfi30iJN8gCZaFJK\$...	2025-12-15 20:14:21.31...
6	6	pbkdf2_sha256\$120000\$amYPglCKppcyEF57DEauL...	2025-12-15 21:33:11.168...
7	7	pbkdf2_sha256\$120000\$doJxelRVFeIcluWizMn...	2025-12-15 21:31:35.58...
8	8	pbkdf2_sha256\$120000\$k9Ew7ZYOlEWg9fMSKQE...	2025-12-16 03:44:52.46...
9	9	pbkdf2_sha256\$120000\$ridBArcOxyOLYHfJJ\$...	2025-12-16 15:29:09.64...

Fig..6.8.1: Security

iii) Design Patterns

a) Factory Pattern

```
from abc import ABC, abstractmethod

# Car -> Product
from cars.models import Car

# Concrete Products inside models.py
# CAR_TYPES = (
#     ('sedan', 'Sedan'),
#     ('suv', 'SUV'),
#     ('truck', 'Truck'),
#     ('coupe', 'Coupe'),
# )

# Creator
class CarFactory(ABC):
    @abstractmethod
    def create_car(self, make, model, year, price, mileage,
owner):
        pass

# Concrete Creators
class SedanFactory(CarFactory):
    def create_car(self, make, model, year, price, mileage,
owner):
        return Car.objects.create(
            make=make, model=model, year=year, price=price,
            mileage=mileage, car_type='sedan', owner=owner
        )

class SUVFactory(CarFactory):
    def create_car(self, make, model, year, price, mileage,
owner):
        return Car.objects.create(
            make=make, model=model, year=year, price=price,
            mileage=mileage, car_type='suv', owner=owner
        )

class TruckFactory(CarFactory):
    def create_car(self, make, model, year, price, mileage,
owner):
```

```

        return Car.objects.create(
            make=make, model=model, year=year, price=price,
            mileage=mileage, car_type='truck', owner=owner
        )

class CoupeFactory(CarFactory):
    def create_car(self, make, model, year, price, mileage,
owner):
        return Car.objects.create(
            make=make, model=model, year=year, price=price,
            mileage=mileage, car_type='coupe', owner=owner
        )

```

b) Singleton Pattern

```

class DatabaseConfigManager:

    # Private static instance variable
    _instance = None

    def __new__(cls):
        """
        Private constructor - controls object creation.
        Only creates a new instance if one doesn't exist.
        """
        if cls._instance is None:
            cls._instance = super(DatabaseConfigManager,
cls).__new__(cls)
                # Initialize configuration only once
                cls._instance._initialize_config()
        return cls._instance

    def _initialize_config(self):
        """
        Private method to initialize configuration.
        Called only once when the instance is first created.
        """
        self.db_connection_string =
"mysql://root:password@localhost:3306/car_hub_db"
        self.max_connections = 100
        self.pool_size = 10
        self.timeout = 30

    @classmethod
    def getInstance(cls):
        """
        Public static method to get the singleton instance.
        """

```

```

This is the standard way to access the singleton.

"""

if cls._instance is None:
    cls._instance = cls()
return cls._instance

def get_connection_string(self):
    """Get the database connection string"""
    return self.db_connection_string

def get_config(self):
    """
    Get the complete database configuration.
    """
    return {
        "connection": self.db_connection_string,
        "max_connections": self.max_connections,
        "pool_size": self.pool_size,
        "timeout": self.timeout
    }

def update_max_connections(self, new_max):
    """Update maximum connections (affects all references)"""
    self.max_connections = new_max

```

c) Observer Pattern

```

from abc import ABC, abstractmethod
from cars.models import Notification

# Observer
class Observer(ABC):
    @abstractmethod
    def update(self, message):
        pass

# Concrete Observer
class UserObserver(Observer):
    def __init__(self, user):
        self.user = user

    def update(self, message):
        # Create a notification in DB

```

```
    Notification.objects.create(user=self.user,
message=message)

# Subject
class Subject(ABC):
    def __init__(self):
        self._observers = []

    def attach(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        try:
            self._observers.remove(observer)
        except ValueError:
            pass

    def notify(self, message):
        for observer in self._observers:
            observer.update(message)

class CarPriceSubject(Subject):
    def __init__(self, car):
        super().__init__()
        self.car = car

    def change_price(self, new_price):
        # Capture old price BEFORE saving
        old_price = self.car.price

        # Update and save the new price
        self.car.price = new_price
        self.car.save()

        # Notify all observers through the proper Observer
        # pattern mechanism
        message = f"The price of {self.car.make} {self.car.model}"
        f"({self.car.year}) has changed from {old_price:.0f} to"
        f"{new_price:.0f}."
        self.notify(message)
```

d) Strategy Pattern

```
from abc import ABC, abstractmethod
from cars.models import Car

class SearchStrategy(ABC):
    @abstractmethod
    def search(self, query):
        pass

class PriceSearchStrategy(SearchStrategy):
    def search(self, price_range):
        # price_range is expected to be a tuple or list:
        [min_price, max_price]
        min_price, max_price = price_range
        return Car.objects.filter(price__gte=min_price,
price__lte=max_price)

class BrandSearchStrategy(SearchStrategy):
    def search(self, brand_name):
        return Car.objects.filter(make__icontains=brand_name)

class ModelSearchStrategy(SearchStrategy):
    def search(self, model_name):
        return Car.objects.filter(model__icontains=model_name)

class MileageSearchStrategy(SearchStrategy):
    def search(self, mileage_range):
        # mileage_range is expected to be a tuple or list:
        [min_mileage, max_mileage]
        min_mileage, max_mileage = mileage_range
        return Car.objects.filter(mileage__gte=min_mileage,
mileage__lte=max_mileage)

class TypeSearchStrategy(SearchStrategy):
    def search(self, car_type):
        return Car.objects.filter(car_type__iexact=car_type)

class YearSearchStrategy(SearchStrategy):
    def search(self, year_range):
        # year_range is expected to be a tuple or list:
        [min_year, max_year]
        min_year, max_year = year_range
        return Car.objects.filter(year__gte=min_year,
year__lte=max_year)

class CarSearchContext:
```

```

    def __init__(self, strategy: SearchStrategy):
        self.strategy = strategy

    def set_strategy(self, strategy: SearchStrategy):
        self.strategy = strategy

    def execute_search(self, query):
        return self.strategy.search(query)

```

e) Decorator Pattern

```

from abc import ABC, abstractmethod

# Component Interface
class CarComponent(ABC):
    @abstractmethod
    def get_price(self):
        pass

    @abstractmethod
    def get_description(self):
        pass

# Concrete Component
class BasicCar(CarComponent):
    def __init__(self, car_model):
        self.car = car_model

    def get_price(self):
        return float(self.car.price)

    def get_description(self):
        return f"{self.car.year} {self.car.make}\n{self.car.model}"

# Decorator
class CarDecorator(CarComponent):
    def __init__(self, car_component: CarComponent):
        self.car_component = car_component

    @abstractmethod
    def get_price(self):
        pass

    @abstractmethod
    def get_description(self):
        pass

```

```

# Concrete Decorators
class WarrantyDecorator(CarDecorator):
    def get_price(self):
        return self.car_component.get_price() + 50000.00

    def get_description(self):
        return self.car_component.get_description() + " +
Extended Warranty"

class DashCamDecorator(CarDecorator):
    def get_price(self):
        return self.car_component.get_price() + 15000.00

    def get_description(self):
        return self.car_component.get_description() + " + Dash
Cam"

class SeatCoversDecorator(CarDecorator):
    def get_price(self):
        return self.car_component.get_price() + 20000.00

    def get_description(self):
        return self.car_component.get_description() + " + Custom
Seat Covers"

class WindowTintingDecorator(CarDecorator):
    def get_price(self):
        return self.car_component.get_price() + 10000.00

    def get_description(self):
        return self.car_component.get_description() + " + Window
Tinting"

```

f) Proxy Pattern

```

from abc import ABC, abstractmethod
from cars.models import Car

class CarAccessInterface(ABC):
    @abstractmethod
    def delete_car(self, car_id):
        pass

    @abstractmethod
    def post_car(self, car_data):

```

```

    pass

    @abstractmethod
    def approve_car(self, car_id):
        pass

    @abstractmethod
    def reject_car(self, car_id, reason):
        pass

class RealCarService(CarAccessInterface):
    def delete_car(self, car_id):
        try:
            car = Car.objects.get(id=car_id)
            car.delete()
            return True, "Car deleted successfully."
        except Car.DoesNotExist:
            return False, "Car not found."

    def post_car(self, car_data):
        # Assuming car_data is a dictionary and we use the
        # factory or direct create
        # For simplicity, just creating here
        return True, "Car posted successfully."

    def approve_car(self, car_id):
        try:
            car = Car.objects.get(id=car_id)
            car.approval_status = 'approved'
            car.save()

            # Notify owner
            from cars.models import Notification
            Notification.objects.create(
                user=car.owner,
                message=f"Your listing '{car.year} {car.make} {car.model}' has been approved by admin and is now visible to buyers."
            )
            return True, "Car listing approved successfully."
        except Car.DoesNotExist:
            return False, "Car not found."

    def reject_car(self, car_id, reason=""):
        try:
            car = Car.objects.get(id=car_id)

```

```

        # Store car details before deletion
        car_year = car.year
        car_make = car.make
        car_model = car.model
        car_owner = car.owner

        # Build notification message
        from cars.models import Notification
        message = f"Your car listing '{car_year} {car_make}" \
{car_model}' has been rejected and removed by admin.\n\nReason:" \
{reason}"

        Notification.objects.create(
            user=car_owner,
            message=message
        )

        # Delete the car
        car.delete()
        return True, "Car listing rejected and seller notified."
    except Car.DoesNotExist:
        return False, "Car not found."


class CarAccessProxy(CarAccessInterface):
    def __init__(self, user):
        self.user = user
        self.real_service = RealCarService()

    def delete_car(self, car_id):
        if self.user.is_superuser:
            return self.real_service.delete_car(car_id)
        # Allow owner to delete their own car
        try:
            car = Car.objects.get(id=car_id)
            if self.user == car.owner:
                return self.real_service.delete_car(car_id)
        except Car.DoesNotExist:
            return False, "Car not found."
        return False, "Permission denied: You are not authorized to delete this car."

    def post_car(self, car_data):
        if not self.user.is_authenticated:
            return False, "Permission denied: You must be logged in to post a car."
        if self.user.is_superuser:

```

```

        return False, "Permission denied: Admin users cannot
list cars for sale."
    return self.real_service.post_car(car_data)

def approve_car(self, car_id):
    if not self.user.is_superuser:
        return False, "Permission denied: Only admin can
approve car listings."
    return self.real_service.approve_car(car_id)

def reject_car(self, car_id, reason=""):
    if not self.user.is_superuser:
        return False, "Permission denied: Only admin can
reject car listings."
    return self.real_service.reject_car(car_id, reason)

```

g) Adapter Pattern

```

# target interface
class CurrencyConverter:

    def convert_to_bdt(self, amount, currency): #to be
implemented by adapter
        """Convert any currency to BDT"""
        raise NotImplementedError #make sure derived class
implements this method

    def convert_from_bdt(self, amount_bdt, target_currency): #to
be implemented by adapter
        """Convert BDT to any currency"""
        raise NotImplementedError #make sure derived class
implements this method

    def get_currency_symbol(self, currency): #to be implemented
by adapter
        """Get the symbol for a currency"""
        raise NotImplementedError #make sure derived class
implements this method

# Adaptee - Third-party service with incompatible interface
class ThirdPartyCurrencyAPI:

    # Exchange rates: 1 unit of currency = X BDT
    EXCHANGE_RATES = {
        'BDT': 1.0,

```

```

        'USD': 120.0,      # 1 USD = 120 BDT
        'GBP': 150.0,      # 1 GBP = 150 BDT
        'EUR': 130.0,      # 1 EUR = 130 BDT
        'INR': 1.45         # 1 INR = 1.45 BDT
    }

CURRENCY_SYMBOLS = {
    'BDT': '৳',
    'USD': '$',
    'GBP': '£',
    'EUR': '€',
    'INR': '₹'
}

def get_exchange_rate(self, currency_code):
    """Third-party API method to get exchange rate"""
    return self.EXCHANGE_RATES.get(currency_code, 1.0)

def get_symbol_for_currency(self, currency_code):
    """Third-party API method to get currency symbol"""
    return self.CURRENCY_SYMBOLS.get(currency_code, '৳')

def multiply_by_rate(self, amount, rate):
    """Third-party API helper method"""
    return float(amount) * rate

def divide_by_rate(self, amount, rate):
    """Third-party API helper method"""
    return float(amount) / rate

# Adapter - Makes the adaptee compatible with the target
# interface
class CurrencyAdapter(CurrencyConverter):

    def __init__(self):
        # Composition: Adapter contains an instance of the
        # adaptee
        self._api = ThirdPartyCurrencyAPI()

    def convert_to_bdt(self, amount, currency):
        rate = self._api.get_exchange_rate(currency)
        return self._api.multiply_by_rate(amount, rate)

    def convert_from_bdt(self, amount_bdt, target_currency):
        rate = self._api.get_exchange_rate(target_currency)
        return self._api.divide_by_rate(amount_bdt, rate)

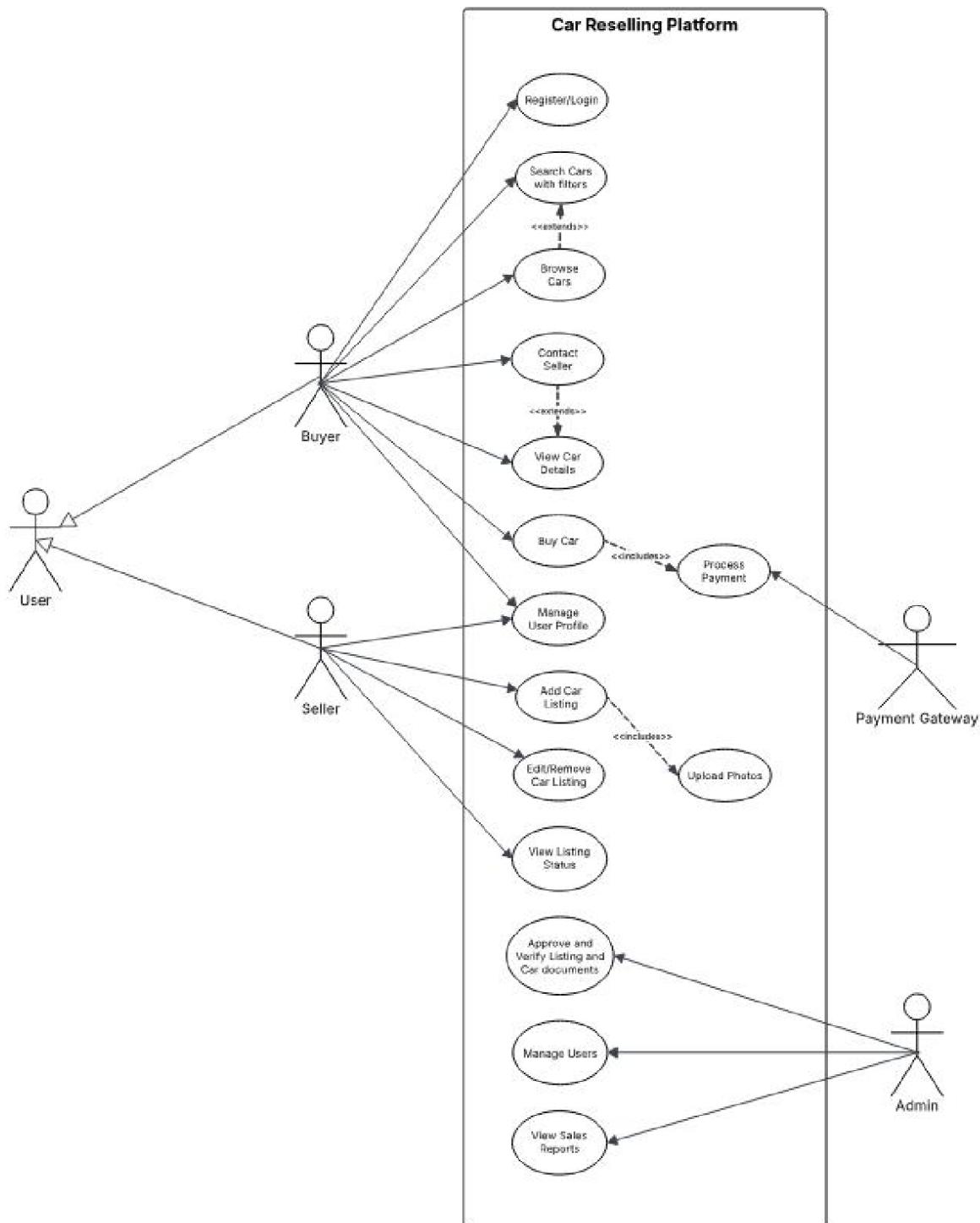
```

```
def get_currency_symbol(self, currency):
    return self._api.get_symbol_for_currency(currency)

# Convenience class methods for backward compatibility
@classmethod
def convert_to_bdt_static(cls, amount, from_currency):
    """Static helper method for quick conversions"""
    adapter = cls()
    return adapter.convert_to_bdt(amount, from_currency)

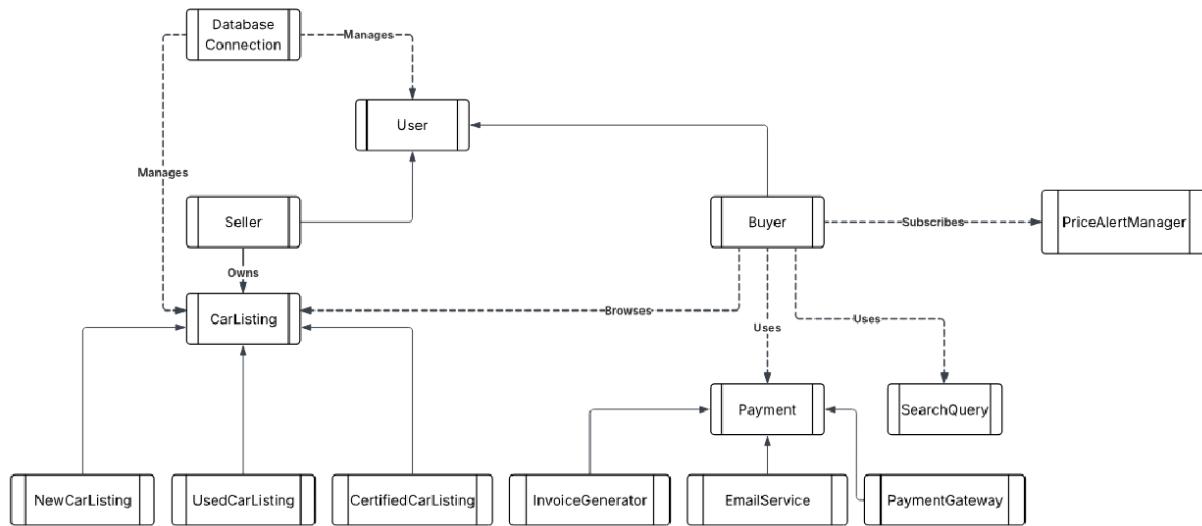
@classmethod
def get_symbol(cls, currency):
    """Static helper method to get currency symbol"""
    adapter = cls()
    return adapter.get_currency_symbol(currency)
```


7. USE CASE DIAGRAM:

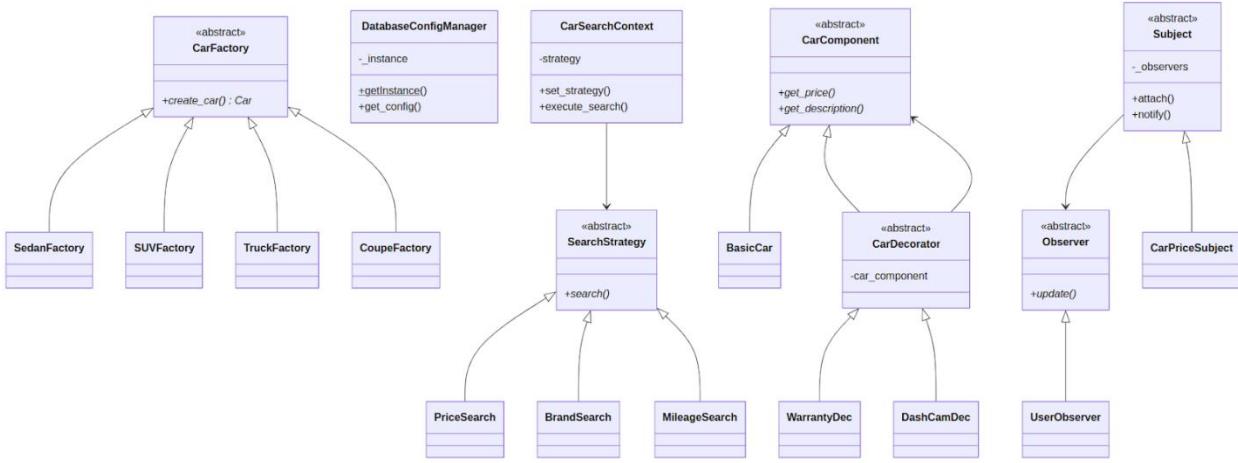


8. CLASS DIAGRAMS

a) Proposed class diagram:

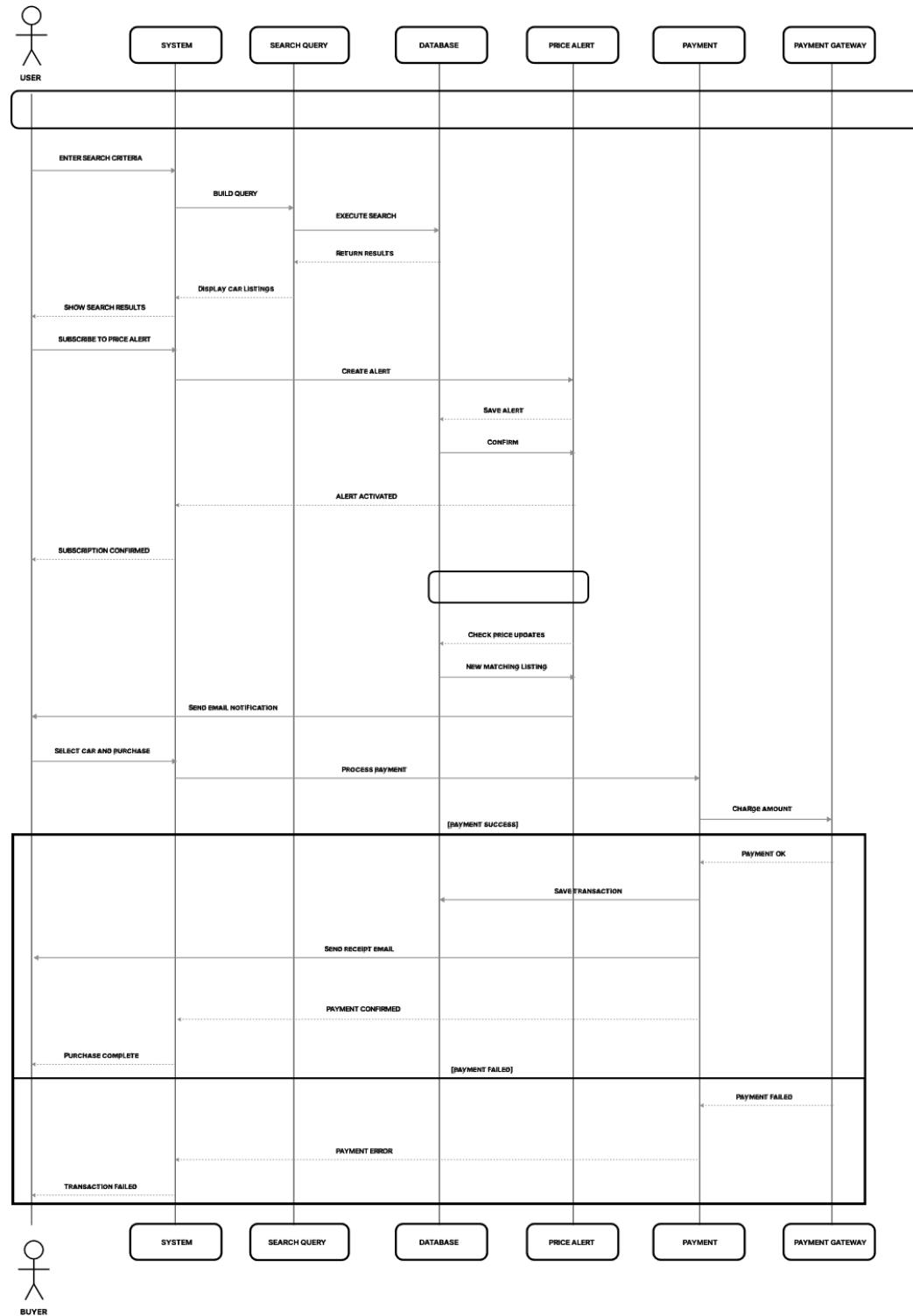


b) Final Class Diagram



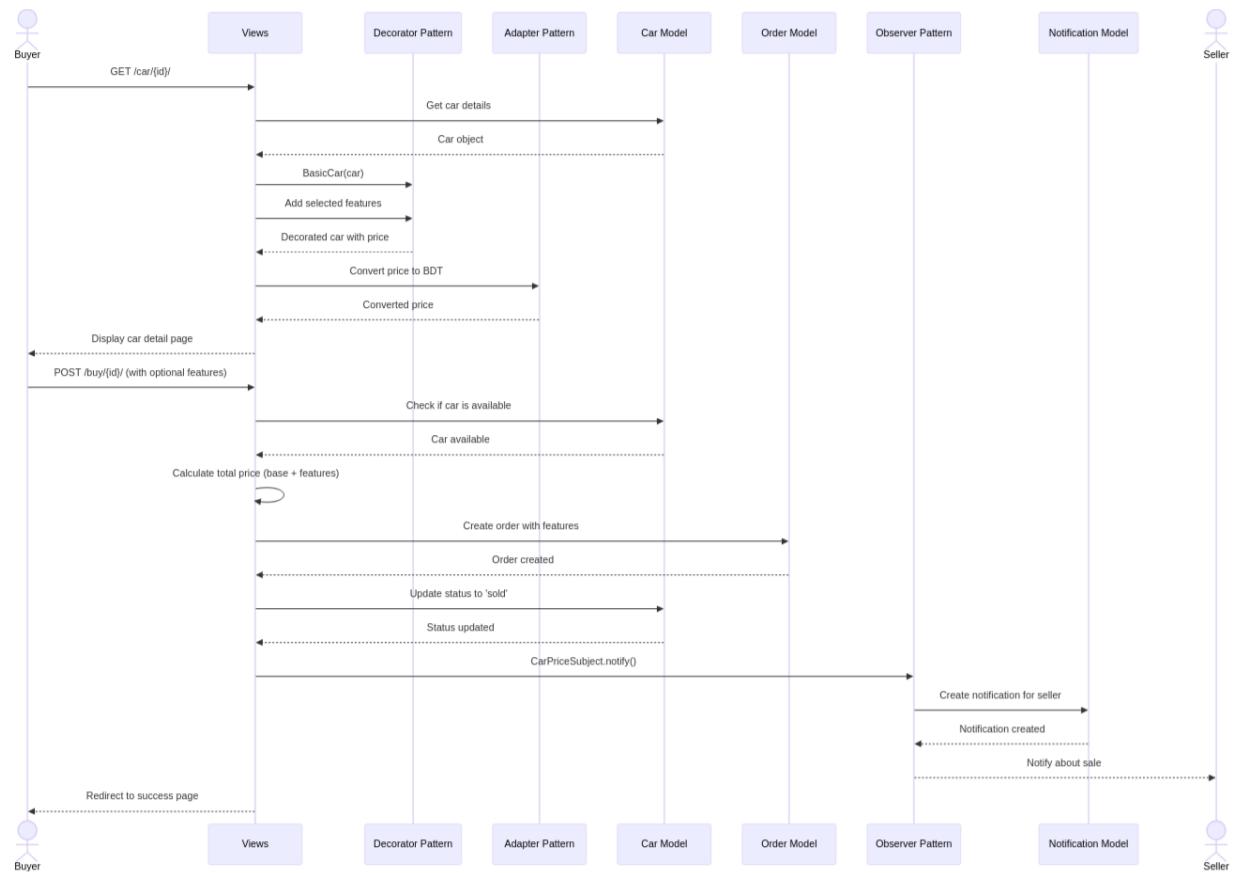
9. SEQUENCE DIAGRAM:

a) Proposed Sequence Diagram

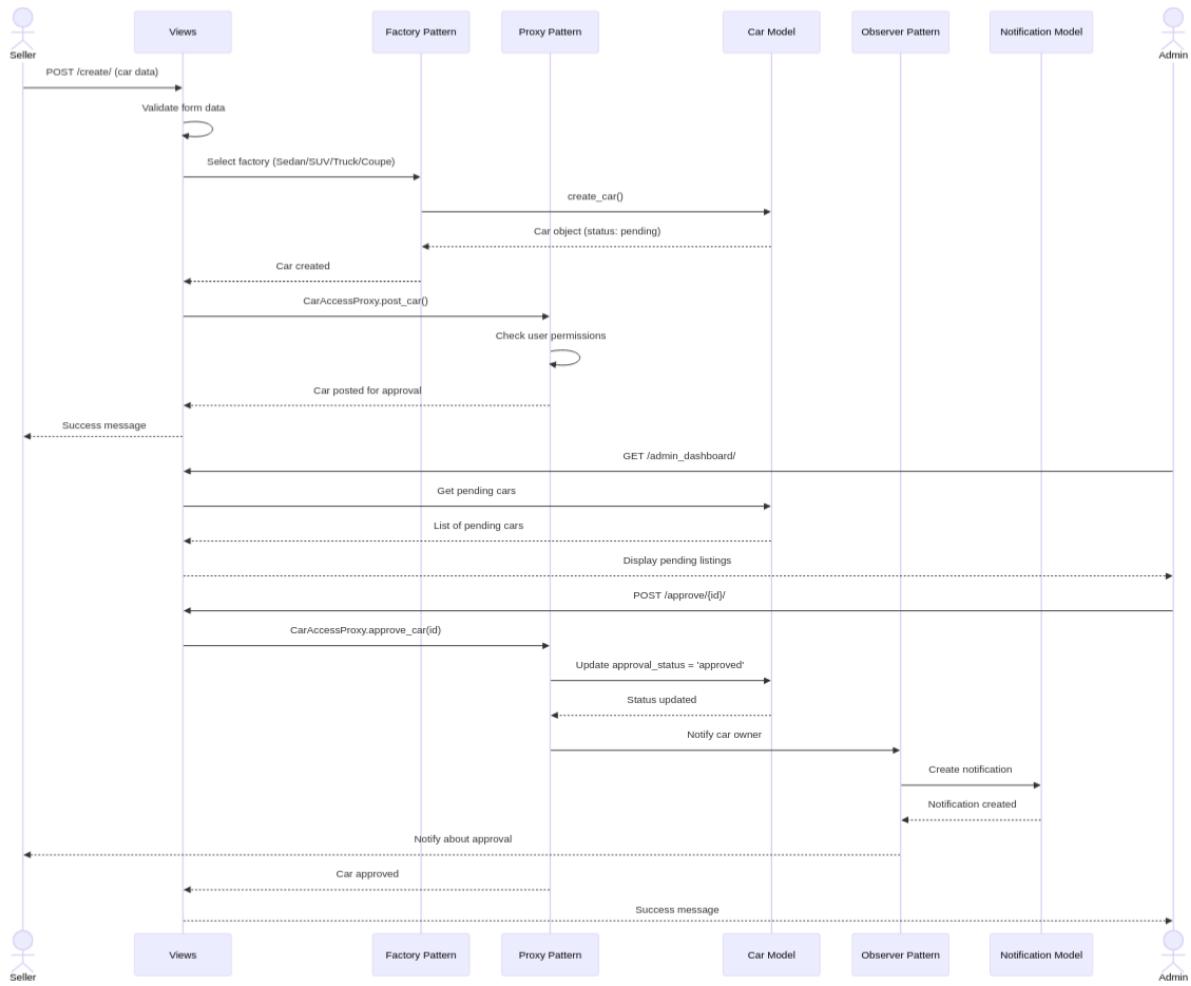


b) Final Sequence Diagram

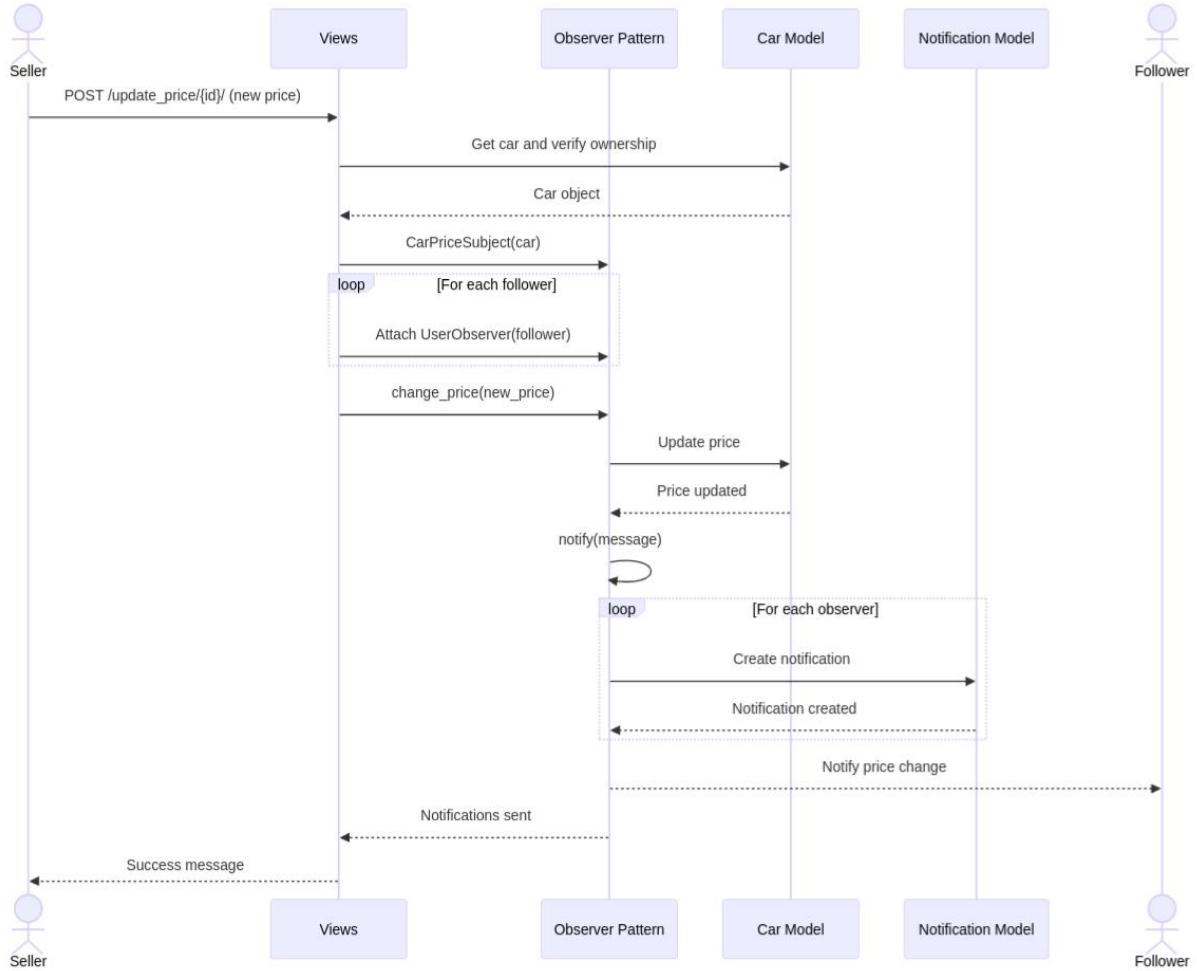
i) Car Purchase Flow:



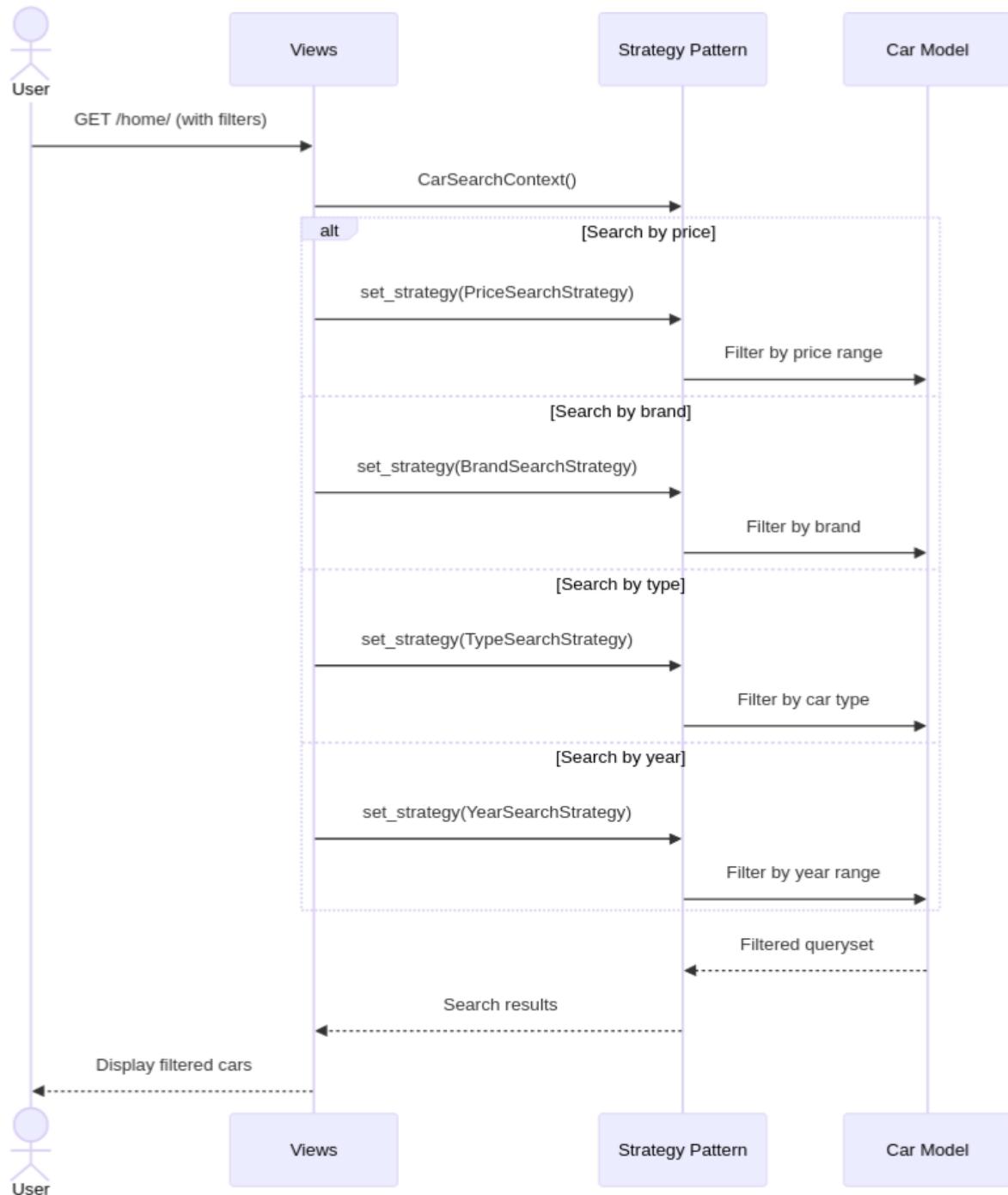
ii) Car Listing Creation Flow:



iii) Price Update and Notification Flow:

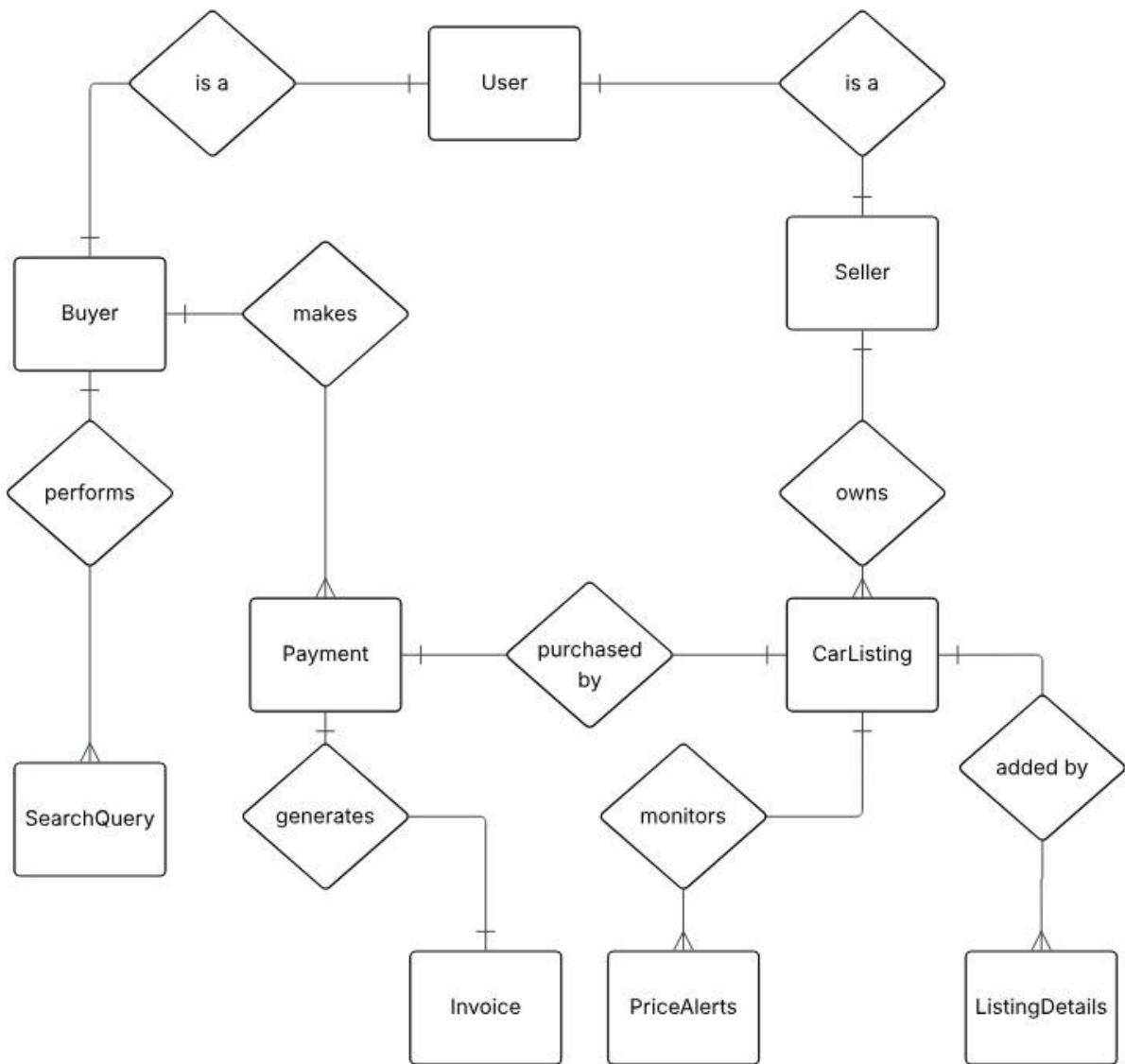


iv) Search and Filter Flow:

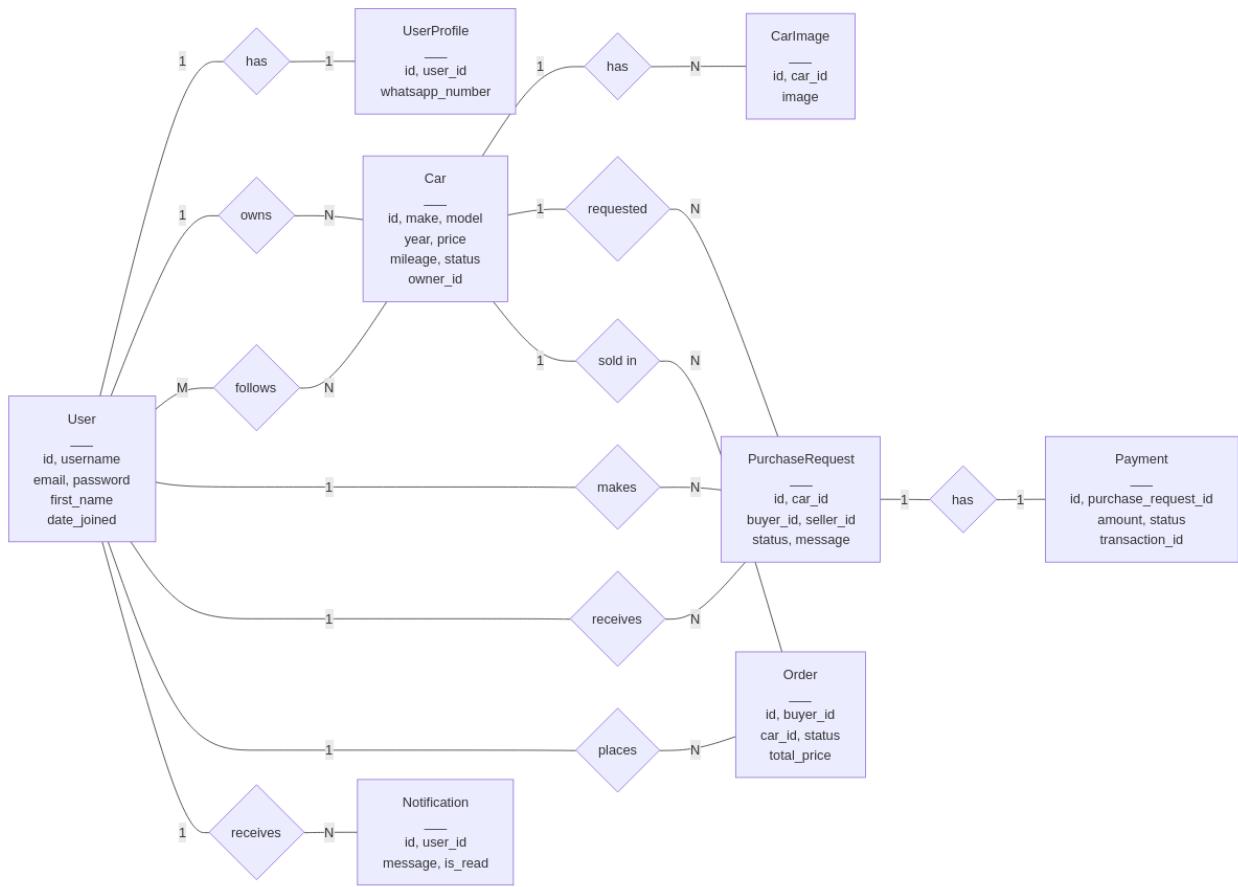


10. ENTITY RELATION DIAGRAM (ERD):

a) Proposed ERD



b) Final ERD:



END of the REPORT