

PetVet – A Social Platform

Rebof Katwal

Rebofkatwal7@gmail.com

Acknowledgement

I would like to take this opportunity to express my deepest gratitude to my family for their unwavering love and support throughout my academic journey. Their support has been constant, and I wouldn't be standing before all of you today if not for them. The inspiration for this Final Year Project came from my parents' and brothers' deep love for our pet and animals in general, which sparked the idea behind PetVet.

I would like to express my heartfelt gratitude to the FYP Support Team and Islington College for offering the platform and the chance to finish this project. Their unconditional support and the guidance that I have received enabled me to finish my job, and I am thankful for that.

A special note of appreciation goes to my supervisors, Mr. Ujjwal Adhikari and Mr. Suyog Man Singh, whose weekly support, consistent feedback, and constructive criticism helped me refine my work every step of the way. Their dedication, attention to detail, and constructive suggestions at times corrected my approach where needed and steered me onto the appropriate track. I would like to thank each of them earnestly for the experience of supervision I gained along the way.

I would also like to extend my gratitude to my peers at the university for not only participating in the testing of the project but also providing sincere comments and suggestions. Based on the opinions gained, I was able to identify bugs, improve the user interface, and design an efficient, user-friendly system. I highly appreciate the time, effort, and patience that went into this critical development process.

Once again, I would like to extend my sincere thanks to all the individuals who worked on this project and helped me achieve this scholarly distinction.

Abstract

This report discusses the design, development, and evaluation of PetVet, an academic project that seeks to enhance the manner in which pet care is digitally handled in Nepal. With growing awareness and interest in pet wellness, the demand for efficient digital solutions is on the rise. According to 2023 data, Nepal exported pet food worth Rs 2.26 billion in the first nine months of the fiscal year, with Rs 1.95 billion going to the United States alone. This trend reflects a significant growth in the pet care market and indicates increasing consumer investment in pet products and services.

The PetVet platform addresses this gap with functions such as the Reddit-like posting facility, community feature, real time chat system, management of appointments and reviews of vets. A moderation-capable admin interface ensures that the community remains safe and engaging. It was created to address problems such as invisibility of appointments, lack of good vet-patient communication, and no community-led participation toward the well-being of the pets.

This report has the usual academic format, with the introduction to the project, the current situation, and problem area, to the aim, objectives, and the report outline. Chapter one discusses the end-user requirements, technical descriptions, and the comparison of the intended solution to others. Development chapter covers how the methodology was selected, pre- and post-survey responses, the analysis of the requirements, and the methods of implementation. Test and analysis include the unit-level and systems-level, reflective analysis of the performance of the project. Conclusion covers the primary points learned, legal and ethical aspects, the benefits to the user, the constraints, and future possibilities of the development. This initiative highlights the increasing prominence of pet care in Nepal and illustrates how such a local, digital platform as PetVet can be one element of the changing world of animal welfare.

Table of Contents

Chapter 1: Introduction	1
1.1 Project Description	1
1.2 Current Scenario	2
1.2.1 Global Context.....	2
1.2.2 Nepal Context.....	3
1.3 Problem Domain	4
1.4 Project as a Solution	5
1.5 Aim and Objectives	6
1.5.1 Aim.....	6
1.5.2 Objectives	6
1.6 Structure of The Report.....	7
1.6.1 Background	7
1.6.2 Development	7
1.6.3 Testing and Analysis	7
1.6.4 Conclusion.....	7
Chapter 2: Background	8
2.1 About the End User	8
2.2 Understanding the Solution	9
2.3 Similar Projects.....	10
2.3.1 Vetstoria.....	10
2.3.2 Vetster.....	11
2.3.3 DaySmart Pet.....	12
2.3.4 PetDesk.....	13
2.4 Comparisons.....	14
Chapter 3: Development	15

3.1 Considered Methodologies	15
3.1.1 Scrum.....	15
3.1.2 Extreme Programming	15
3.1.3 Kanban.....	15
3.2 Selected Methodology	16
3.2.1 About Prototyping.....	16
3.2.2 Reasons for Choosing Prototyping.....	16
3.3 Phases of Methodology	17
3.4 Survey Results.....	19
3.4.1 Pre-Survey Results	19
3.4.2 Post-Survey Results.....	20
3.5 Software Requirement Specifications.....	21
1.1 Introduction	21
1.2 Overall Description.....	22
1.3 Functional Requirements	27
1.4 External Interfaces Requirements.....	36
3.6 Design	40
3.6.1 Initial Design.....	40
3.6.2 Initial Development.....	81
3.6.3 Prototype/Iteration 1	102
3.6.4 Prototype/Iteration 2.....	116
3.6.5 Prototype/Iteration 3.....	138
3.6.6 Prototype/Iteration 4.....	148
3.6.7 Final Prototype.....	162
3.7 Implementation	175
3.7.1 Trello	176
3.7.2 Github	190

3.7.3 Postgres Connection.....	191
3.7.4 Folder Structure	192
Chapter 4: Testing and Analysis	193
4.1 Test Plan	193
4.1.1 Unit Testing Test Plan.....	193
4.1.2 System Testing Test Plan	193
4.2 Unit Testing.....	194
4.2.1 Test: Auth1 Registration Functionality.....	194
4.2.2 Test: Auth2 Login Functionality	198
4.2.3 Test: Auth3 Admin Login Functionality	201
4.2.4 Test: Admin1 Vet Approval Functionality	204
4.2.5 Test: Admin2 Vet Decline Functionality.....	207
4.2.6 Test: Admin3 Content Moderation Functionality.....	211
4.2.7 Test: Admin4 Category Management Functionality.....	214
4.2.8 Test: Appt1 Add Schedule Functionality	218
4.2.9 Test: Appt2 Edit Schedule Functionality.....	221
4.2.10 Test: Appt3 Booking via Khalti Functionality	227
4.2.11 Test: Appt4 Booking via Store Credit Functionality	231
4.2.12 Test: Appt5 Accept Booking Functionality	236
4.2.13 Test: Appt6 Reject Booking Functionality.....	240
4.2.14 Test: Appt7 Mark as Complete Functionality.....	244
4.2.15 Test: Appt8 Booking Cancellation Functionality	249
4.2.16 Test: Appt9 Khalti Payment Functionality.....	252
4.2.17 Test: Appt10 Review Functionality	255
4.2.18 Test: Po1 Post Creation and Deletion Functionality	259
4.2.19 Test: Po2 Post Updating and Liking Functionality	263
4.2.20 Test: Po3 Liking, Deleting and Adding Comments Functionality	266

4.2.21 Test: Po4 Reply Creation and Deletion Functionality	270
4.2.22 Test: RTC1 Real-Time Chat Functionality.....	275
4.3 System Testing	279
4.3.1 System Test: Auth1 Pet Owner Registration Functionality	279
4.3.2 System Test: Auth2 Login and Logout Functionality	287
4.3.3 System Test: Auth3 Vet Registration Functionality	292
4.3.4 System Test: Appt1 Pet Addition, Deletion and Updating Functionality	305
4.3.5 System Test: Appt2 Vet Schedule Addition, Deletion and Updating Check	315
4.3.6 System Test: Appt3 Book Appointment via Khalti Flow.....	329
4.3.7 System Test: Appt4 Review Functionality	340
4.3.8 System Test: Appt5 Decline Appointment and Refunding.....	345
4.3.9 System Test: Appt6 Book Appointment via Store Credit and Cancellation .	350
4.3.10 System Test: Appt7 Book Appointment Time Slot Availability Checking ...	357
4.3.11 System Test: RTC1 Real Time Chat Functionality	362
4.3.12 System Test: Po1 Reply Creation and Deletion Functionality	366
4.3.13 System Test: Po2 Comment Creation and Deletion Functionality	375
4.3.14 System Test: Po3 Like and Unlike Functionality	380
4.3.15 System Test: Po4 Post Functionality.....	384
4.3.16 System Test: Admin1 Administrator Login Functionality	395
4.3.17 System Test: Admin2 Category Management Functionality.....	399
4.3.18 System Test: Admin3 Content Moderation Functionality.....	407
4.3.19 System Test: Utils1 Password Change Functionality	419
4.3.20 System Test: Utils2 Forgot Password Functionality	428
4.3.21 System Test: Utils3 Search Functionality	435
4.3.22 System Test: Utils4 User Information Update Functionality	439
4.3.23 System Test: Utils5 Contact Form and Email Notification Checking	448
4.4 Critical Analysis	467

Chapter 5: Conclusion.....	468
5.1 Legal, Social and Ethical Issues	469
5.1.1 Legal Issues	469
5.1.2 Social Issues	470
5.1.3 Ethical Issues	471
5.2 Advantages.....	472
5.3 Limitations	473
5.4 Future work.....	474
6. References.....	475
Chapter 7: Appendix.....	478
7.1 Appendix A: Pre-Survey	478
7.1.1 Pre-Survey Form.....	478
7.1.2 Sample of Filled Pre-Survey Forms	478
7.1.3 Pre-Survey Result	481
7.2 Appendix B: Post-Survey.....	494
7.2.1 Post-Survey Form	494
7.2.2 Sample of Filled Post-Survey Forms.....	494
7.2.3 Post-Survey Result	496
7.3 Appendix C: Unit Test Codes.....	505
7.3.1 Code of the Automation Script	505
7.4 Appendix D: Designs	591
7.4.1 Gantt Chart.....	591
7.4.2 Work Breakdown Structure	593
7.4.3 Entity Relation Diagram	594
7.4.4 Data Flow Diagram 0.....	596
7.4.5 Use Case	597
7.4.6 Activity Diagram	598

7.4.7 Sequence Diagram	599
7.4.8 Collaboration Diagram	601
7.4.9 Class Diagram.....	602
7.4.10 Architectural Choice	603
7.4.11 Draft Figma Design's Board	605
7.4.12 Final Figma Design's Board	606
7.6 Appendix F: User Feedback	607
7.6.1 User Feedback Form for Initial Development Results	607
7.6.2 User Feedback Form for Initial Development Sample Answer	612
7.6.3 User Feedback Form for Prototype 1 Results	613
7.6.4 User Feedback Form for Prototype 1 Sample Answer	618
7.6.5 User Feedback Form for Prototype 2 Results	619
7.6.6 User Feedback Form for Prototype 2 Sample Answer	626
7.6.7 User Feedback Form for Prototype 3 Results	627
7.6.8 User Feedback Form for Prototype 3 Sample Answer	632
7.6.9 User Feedback Form for Prototype 4 Results	633
7.6.10 User Feedback Form for Prototype 4 Sample Answer	636
7.6.11 User Feedback Form for Final Prototype Results	637
7.6.12 User Feedback Form for Final Prototype Sample Answer	642
7.7 Appendix G: Future Work	643
7.7.1 Reading for Future Work.....	643
7.8 Appendix H: Considered Methodologies.....	645
7.9 Appendix I: Research Work	647
7.9.1 Research 1	647
7.9.2 Research 2.....	649
7.9.3 Research 3.....	651
7.9.4 Research 4.....	653

7.9.5 Research 5.....	655
7.10 Appendix J: System Screenshot	656
7.11 Appendix K: Sample Codes	678
7.11.1 Blogs	678
7.11.2 Appointment.....	697
7.11.3 Chat	733
7.11.4 Admin Dashboard	757
7.11.5 Authentication	791

Table of Figures

Figure 1: Screenshot of Similar Application:Vetstoria	10
Figure 2: Screenshot of Similar Application:Vetster	11
Figure 3: Screenshot of Similar Application:DaySmartPet	12
Figure 4: Screenshot of Similar Application:PetDesk	13
Figure 5: Phases of prototyping model.....	18
Figure 6: System Perspective Diagram	22
Figure 7: Operating Environment	25
Figure 8: System Logo (PETVET)	41
Figure 9: Final Work Breakdown Structure.....	42
Figure 10: Final Gantt Chart.....	43
Figure 11: Final Entity Relationship Diagram	44
Figure 12: Final Class Diagram.....	45
Figure 13: Final Data Flow Diagram.....	46
Figure 14: Final Usecase Diagram	47
Figure 15: Wireframe for Login.....	48
Figure 16: Wireframe for Registration	49
Figure 17: Wireframe for Home	49
Figure 18: Wireframe for Pet profile	50
Figure 19: Wireframe for Add profile	50
Figure 20: Wireframe for Find Vets	51
Figure 21: Wireframe for Appointment lists	51
Figure 22: Wireframe for Inbox.....	52
Figure 23: Wireframe for Post details	52
Figure 24: Wireframe for User profile	53
Figure 25: Wireframe for Appointment details	53
Figure 26: Wireframe for Vet schedule.....	54
Figure 27: Wireframe for Booking	54
Figure 28: Wireframe for Landing.....	55
Figure 29: Wireframe for Landing-appointment.....	56
Figure 30: Wireframe for Landing-community	57
Figure 31: Wireframe for Landing-about-us.....	58

Figure 32: Wireframe for Landing-contact	59
Figure 33: Wireframe for Pending booking requests	60
Figure 34: Wireframe for Accepted Bookings.....	60
Figure 35: Wireframe for Add schedule.....	61
Figure 36: Wireframe for Admin Dashboard.....	61
Figure 37: Wireframe for User management	62
Figure 38: Wireframe for Category Management	62
Figure 39: Wireframe for Vet approvals.....	63
Figure 40: Wireframe for Post Management	63
Figure 41: Draft Design for Home	65
Figure 42: Draft Design for Post details.....	66
Figure 43: Draft Design for Find vets.....	67
Figure 44: Draft Design for Appointments list.....	68
Figure 45: Draft Design for Appointment details.....	69
Figure 46: Draft Design for Pending requests	70
Figure 47: Draft Design for Vet Schedule lists.....	71
Figure 48: Draft Design for Add schedule	72
Figure 49: Draft Design for Booking	73
Figure 50: Draft Design for Accepting bookings	74
Figure 51: Draft Design for Inbox	75
Figure 52: Draft Design for Admin Dashboard	76
Figure 53: Draft Design for Category management.....	77
Figure 54: Draft Design for Vet Approvals.....	78
Figure 55: Draft Design for User management.....	79
Figure 56: Draft Design for Post management	80
Figure 57: Main Design for Registration	81
Figure 58: Main Design for Complete your profile	82
Figure 59: Main Design for Waiting page	83
Figure 60: Main Design for OTP verification.....	83
Figure 61: Main Design for Login	84
Figure 62: Main Design for User profile.....	85
Figure 63: Main Design for Account settings.....	86
Figure 64: Main Design for Admin Login	87

Figure 65: Main Design for Vet approval	87
Figure 66: Flowchart for registration.....	88
Figure 67: Flowchart for login.....	89
Figure 68: Sequence for login	90
Figure 69: Sequence for registration	91
Figure 70: Activity diagram for vet registration	92
Figure 71: Activity diagram for pet owner registration	93
Figure 72: Activity diagram for login	94
Figure 73: Collaboration for diagram for vet registration	95
Figure 74: Collaboration diagram for pet owner registration.....	95
Figure 75: Collaboration diagram from login	95
Figure 76 Code Snippet – User Registration Logic in Initial Development	96
Figure 77 Code Snippet – User Authentication Logic in Initial Development.....	97
Figure 78 Code Snippet – Logout logic in Initial Development	97
Figure 79 Code Snippet – Logout Logic in Initial Development.....	98
Figure 80 Code Snippet – Waiting page Logic in Initial Development	98
Figure 81 Code Snippet – Verify OTP Logic in Initial Development	99
Figure 82: Main Design for Home	102
Figure 83: Main Design for Create modal.....	103
Figure 84: Main Design for Post details.....	104
Figure 85: Main Design for Edit post	105
Figure 86: Flowchart for Blog posts.....	106
Figure 87: Activity diagram for Blogs posts	107
Figure 88: Sequence diagram for Blog posts	108
Figure 89: Collaboration diagram for Blog posts	109
Figure 90 Code Snippet – Home page Logic in Prototype 1	110
Figure 91 Code Snippet – Liking Logic in Prototype 1	110
Figure 92 Code Snippet – Create post Logic in Prototype 1	111
Figure 93 Code Snippet – Post details page Logic in Prototype 1	112
Figure 94 Code Snippet – Delete post Logic in Prototype 1	113
Figure 95: Main Design for Add pet.....	116
Figure 96: Main Design for Pet profile	117
Figure 97: Main Design for Edit pet profile	118

Figure 98: Main Design for Find vets.....	119
Figure 99: Main Design for Vet's Schedule list	120
Figure 100: Main Design for Booking	121
Figure 101: Main Design for Appointment lists	122
Figure 102: Main Design for Appointment details.....	123
Figure 103: Main Design for Add schedule	124
Figure 104: Main Design for Edit schedule.....	125
Figure 105: Main Design for Pending requests	126
Figure 106: Main Design for Accepted bookings.....	127
Figure 107: Flowchart for appointment.....	128
Figure 108: Activity diagram for appointment.....	129
Figure 109: Sequence diagram for appointment	131
Figure 110: Collaboration for appointment	132
Figure 111 Code Snippet – Book appointment Logic in Prototype 2	133
Figure 112: Code Snippet – Cancel appointment Logic in Prototype 2	134
Figure 113: Code Snippet – Mark as complete Logic in Prototype 2.....	134
Figure 114: Code Snippet – Verify khalti payment Logic in Prototype 2.....	135
Figure 115: Main Design for Inbox	138
Figure 116: Flowchart for Chat system.....	139
Figure 117: Main Design for Chat system	140
Figure 118: Sequence diagram for chat system.....	141
Figure 119: Collaboration diagram for chat system.....	142
Figure 120: Code Snippet – Chat Notification Logic in Prototype 3.....	143
Figure 121: Code Snippet – Chat Logic in Prototype 3	144
Figure 122: Code Snippet – Chat page rendering Logic in Prototype 3	145
Figure 123: Main Design for Admin dashboard	148
Figure 124: Main Design for User management.....	149
Figure 125: Main Design for User Details.....	150
Figure 126: Main Design for Post management	151
Figure 127: Main Design for Admin post detail.....	151
Figure 128: Main Design for Category Management.....	152
Figure 129: Flowchart for admin dashboard.....	153
Figure 130: Activity diagram for admin dashboard	154

Figure 131: Sequence diagram for admin dashboard	155
Figure 132: Collaboration diagram for admin dashboard	156
Figure 133: Code Snippet – Admin required decorator Logic in Prototype 4.....	157
Figure 134: Code Snippet – Post management Logic in Prototype 4.....	157
Figure 135: Code Snippet – Approve vet Logic in Prototype 4.....	158
Figure 136: Code Snippet – Add category Logic in Prototype 4.....	159
Figure 137: Main Design for Error page	162
Figure 138: Main Design for Landing	163
Figure 139: Main Design for Landing-appointment	164
Figure 140: Main Design for Landing-community.....	165
Figure 141: Main Design for Landing-about-us	166
Figure 142: Main Design for Landing-contact.....	167
Figure 143: Main Design for Review	168
Figure 144: Code Snippet – Error Page Rendering Logic in Final Prototype	169
Figure 145: Code Snippet – Contact us Logic in Final Prototype	170
Figure 146: Code Snippet – Landing pages rendering Logic in Final Prototype.....	171
Figure 147: Code Snippet – Review Logic in Final Prototype	172
Figure 148: Entire Trello Board	176
Figure 149: Labels used in trello	176
Figure 150: Trello Initial Design Card	177
Figure 151: Trello Initial Development Card	178
Figure 152: Trello Initial Development Card 2	178
Figure 153: Trello Initial Development Card Checklist 1.....	179
Figure 154: Trello Initial Development Card Checklist 2.....	179
Figure 155: Trello Initial Development Card Checklist 3.....	180
Figure 156: Trello Prototype 1 Card	181
Figure 157: Trello Prototype 1 Card 2	181
Figure 158: Trello Prototype 1 Checkout 1	182
Figure 159: Trello Prototype 1 Checkout 2	182
Figure 160: Trello Prototype 1 Card Checklist 1	183
Figure 161: Trello Prototype 2 Card	184
Figure 162: Trello Prototype 2 Card 2	184
Figure 163: Trello Prototype 1 Card Checklist 1	185

Figure 164: Trello Prototype 2 Checklist 1	185
Figure 165: Trello Prototype 3 Card	186
Figure 166: Trello Prototype 3 Card 2	186
Figure 167: Trello Prototype 4 Card	187
Figure 168: Trello Prototype 4 Card 2	187
Figure 169: Trello Prototype 4 Card Checklist 1.....	188
Figure 170: Trello Final Prototype Card	189
Figure 171: Trello Final Prototype Card Checklist 1	189
Figure 172: Screenshot of GitHub Repository – PetVet Project.....	190
Figure 173: Snippet of Git Commit History for the PetVet Project	190
Figure 174: Screenshot of giving permission in Postgres.....	191
Figure 175: Folder Structure	192
Figure 176: Unit Test - Registration result.....	197
Figure 177: Unit Test - Login result	203
Figure 178: Unit Test - Vet approval result.....	206
Figure 179: Unit Test - Decline vet result	210
Figure 180: Unit Test - Content Moderation result	213
Figure 181: Unit Test - Category Management result.....	217
Figure 182: Unit Test - Add schedule result	220
Figure 183: Unit Test - Edit schedule result	226
Figure 184: Unit Test - Booking via khalti result	230
Figure 185: Unit Test - Booking via Store credit failed result.....	234
Figure 186: Unit Test - Booking via Store credit Solution	234
Figure 187: Unit Test - Booking via Store credit result	235
Figure 188: Unit Test - Accept booking result	239
Figure 189: Unit Test - Reject/decline booking result	243
Figure 190: Unit Test - Mark as Complete failed result	247
Figure 191: Unit Test - Mark as Complete Solution.....	248
Figure 192: Unit Test - Mark as Complete result.....	248
Figure 193: Unit Test - Booking Cancellation result	251
Figure 194: Unit Test - Khalti payment result	254
Figure 195: Unit Test - Review result	258
Figure 196: Unit Test - Post creation and Deletion result.....	262

Figure 197: Unit Test - Post Updating and Liking result	265
Figure 198: Unit Test - Like, Delete and Add Comments Results	269
Figure 199: Unit Test - Reply creation and deletion result.....	274
Figure 200: Unit Test - Real time chat result	278
Figure 201: Leaving the phone number field empty	281
Figure 202: Registering with correct credentials	282
Figure 203: Leaving the Pets owned field empty.....	282
Figure 204: Entering negative integer in the Pet owned field	283
Figure 205: Entering a valid data and Submitting the profile	284
Figure 206: Redirected to OTP Verification.....	284
Figure 207: Entering invalid OTP	285
Figure 208: Entering the correct OTP.....	285
Figure 209: Successful Pet Owner Registration	286
Figure 210: Entering incorrect credentials.....	288
Figure 211:Authentication error message displayed upon invalid login attempt.....	289
Figure 212: Entering correct credentials.....	289
Figure 213: Successfully Logged In	290
Figure 214: Clicking the logout button	290
Figure 215: Successfully Logs out	291
Figure 216: Leaving the username field empty	294
Figure 217: Mismatching the password fields	295
Figure 218: Entering the Correct fields.....	295
Figure 219: Leaving the years of experience field empty	296
Figure 220: Redirected to Waiting page	297
Figure 221: Admin dashboard for vet approval	298
Figure 222: Vet has been declined.....	299
Figure 223: Redirected to profile page again	300
Figure 224: Waiting for approval again.....	301
Figure 225: Admin views the profile	301
Figure 226:Admin approves of the Vet	302
Figure 227: Approved Vet has been listed in the dashboard.....	303
Figure 228: Now the Vet can access the home page	304
Figure 229: Add pet page.....	307

Figure 230: Leaving the Age, Color and Name field empty	308
Figure 231: Entering the complete information for the pet.....	309
Figure 232: Successfully adding the pet.....	310
Figure 233: Attempting to update the age field with negative integer	311
Figure 234: Successfully updated the pet profile.....	312
Figure 235: Clicking on the delete button	313
Figure 236: An alert box pops up	313
Figure 237: Redirected to add pet page after deletion	314
Figure 238: Viewing an old time slot.....	317
Figure 239: Adding a new time slot	318
Figure 240: Creating a time slot which overlaps with the old time slot	319
Figure 241: Overlapping could not be done	320
Figure 242: Adding a proper time slot	320
Figure 243: Time Slot has been added	321
Figure 244: Updating a time slot so it overlaps with another	322
Figure 245: Updating failed as overlapping couldn't be done	323
Figure 246: Making the Start time and End time invalid	324
Figure 247: Updating the time slot with a valid time	325
Figure 248: Updated the time Slot.....	326
Figure 249: Deleting the Time slot	327
Figure 250: Successfully deleted the time slot	328
Figure 251: Finding Vet page	331
Figure 252: Select a time slot.....	332
Figure 253: Click pay with khalti option	333
Figure 254: Redirected to khalit's payment page	334
Figure 255: Payment Successful.....	334
Figure 256: Appointment Details page after payment	335
Figure 257: Accepting the appointment request.....	335
Figure 258: Alert box after accepting.....	336
Figure 259: Redirected to accepted appointments page	337
Figure 260: Appointment status change after accepting	337
Figure 261: Mark as complete checkbox clicked.....	338
Figure 262: After mark as complete, review button appears	339

Figure 263: Redirected to review page after clicking the review button.....	342
Figure 264: Leaving the review field empty	342
Figure 265: Entering a proper review and submitting	343
Figure 266: Review button disappears after reviewing.....	343
Figure 267: Rating updated after review	344
Figure 268: Review added to the Vet's profile	344
Figure 269: Payment done through Khalti.....	347
Figure 270: After successful payment, shows a popup and starts redirecting to payment success page	347
Figure 271: User profile before decline	348
Figure 272: Decline with an alert box	348
Figure 273: After decline, the user is refunded.....	349
Figure 274: Pay with Store Credit option clicked.....	352
Figure 275: Successful payment redirects to appointment details page.....	353
Figure 276: After the payment via store credit, the store credit is 0	353
Figure 277: The Vet has received the request	354
Figure 278: Clicking the cancellation button.....	354
Figure 279: Cancellation with alert box	355
Figure 280: Status change to cancelled	356
Figure 281: Only 80% of the original amount refunded	356
Figure 282: Appointment confirmed of Pet owner Rebof.....	359
Figure 283: Astha Pet owner trying to book the same time slot	360
Figure 284: The appointment is completed of Pet owner Rebof.....	360
Figure 285: The time slot has opened up	361
Figure 286: Sending message to adminRebof as a Vet	363
Figure 287: Receiving message from the Vet	364
Figure 288: Vet receives message from adminRebof.....	364
Figure 289: Receiving seen status after vet sees the meesage	365
Figure 290: Selecting a post.....	368
Figure 291: Redirected to post details page.....	369
Figure 292: Leaaving the reply empty	370
Figure 293: Submitting a proper reply	371
Figure 294: Reply has been addded	372

Figure 295: Deleting the added reply	373
Figure 296: Reply has been deleted.....	374
Figure 297: Leaving the comment field empty.....	376
Figure 298: Added a valid comment.....	377
Figure 299: Deleting the comment	378
Figure 300: Comment has been deleted	379
Figure 301: A comment with 0 likes	381
Figure 302: Liking the comment	381
Figure 303: Unlike the comment	382
Figure 304: A post with zero likes	382
Figure 305: Like the post.....	383
Figure 306: Unlike the post	383
Figure 307: Clicking one "What's on your mind?"	386
Figure 308: Leaving the category field empty.....	386
Figure 309: Creating a proper post	387
Figure 310: Post has been added	388
Figure 311: Selecting the edit option.....	389
Figure 312: Leaving the title empty in edit mode.....	390
Figure 313: Updating the title	391
Figure 314: Successfully Updated the post title	392
Figure 315: Delete option has been clicked	393
Figure 316: Post has been deleted as newest post is different	394
Figure 317: Entering invalid admin credentials.....	396
Figure 318: Authentication Error message displayed	397
Figure 319: Entering correct credentials.....	397
Figure 320: Successfully logged into admin dashboard	398
Figure 321: Leaving the category title field empty	401
Figure 322: Entering the missing data.....	401
Figure 323: Added a new category	402
Figure 324: Showing the new category in the home page.....	402
Figure 325: Leaving category name field empty while updating.....	403
Figure 326: Updating the category name field.....	404
Figure 327: Updated the category name successfully	405

Figure 328: Deleting the category	405
Figure 329: Successfully deleted the category	406
Figure 330: The first post is showing in the home page before content moderation ...	409
Figure 331: The post was selected for deactivation	409
Figure 332: The post has been deactivated	410
Figure 333: The post disappears from the home page.....	411
Figure 334: The post can be seen in the user's profile while deactivated	412
Figure 335: Activating the post.....	413
Figure 336: The post has been activated	413
Figure 337: The post can be seen in the home page after activation.....	414
Figure 338: Deleting the Post.....	415
Figure 339: The post has been deleted.....	415
Figure 340: After admin deletes the post, it completely disappears	416
Figure 341: Admin deleting a reply.....	417
Figure 342: Reply was successfully deleted.....	417
Figure 343: Shubham's reply is nowhere to seen.....	418
Figure 344: Wrong current password	421
Figure 345: Entering the wrong current password.....	422
Figure 346: Entering a short new password	423
Figure 347: Error message displaying the password is short and common	424
Figure 348: Entering a valid new password and correct current password	425
Figure 349: The password was successfully updated	426
Figure 350: Logging in with the new changed password.....	427
Figure 351: Logged in with the new password	427
Figure 352: Entering the email for forgot password.....	430
Figure 353: Email will be sent to the email if it exists in the system	430
Figure 354: The link was sent to the email	431
Figure 355: Clicking the link will lead to a reset password page	431
Figure 356: The new password was made very small.....	432
Figure 357: Error message was displayed as the password was too small.....	432
Figure 358: Mismatching two passwords	433
Figure 359: Redirected to log in page	433
Figure 360: Error message will show if the link is clicked again	434

Figure 361: Clicking on the forgot password button from the login.....	434
Figure 362: Clicking the search.....	436
Figure 363: Searching for "Shubham"	436
Figure 364: Search result for "Shubham"	436
Figure 365: Searching for "Rebof"	437
Figure 366:Search result for "Rebof"	437
Figure 367: Searching for "Essential Grooming Tips"	438
Figure 368: Search result for "Essential Gromming Tips"	438
Figure 369: User profile page for updating	441
Figure 370: Edit mode is activated	442
Figure 371: Updated the fields	443
Figure 372: The fields are updated but need multiple refreshs for the change to show	444
Figure 373: HTML code before the solution	445
Figure 374: Newly Added field to the model.....	445
Figure 375: The added field used as the solution.....	445
Figure 376: Updating the profile picture again.....	446
Figure 377: Profile picture was immediately updated.....	447
Figure 378: Registering a pet owner user for otp email check	450
Figure 379: The otp was sent to the email	451
Figure 380: The otp was sent.....	453
Figure 381: Creating a new vet account for approval email notification check	453
Figure 382: The Vet was approved	454
Figure 383: The approval email was received.....	455
Figure 384: Booking the appointment with Khalti as Rebof	456
Figure 385: The appointment email notification is sent to the Vet	457
Figure 386: Booking the appointment by using Store Credit as Shubham	458
Figure 387: Email notification sent to the vet.....	459
Figure 388: Both booking appointment request received	460
Figure 389: Accepting Rebof's booking	461
Figure 390: Appointment confirmation sent to Rebof	462
Figure 391: Rejection email sent to Shubham.....	463
Figure 392: Contact form for email notification checking.....	464

Figure 393: Filling the contact form	465
Figure 394: Admin receives the email	466
Figure 395: Filled Presurvey Form Sample	478
Figure 396: Filled Presurvey Form Sample 2	479
Figure 397: Filled Presurvey Form Sample 3	480
Figure 398: Presurvey Form Question 2	481
Figure 399: Presurvey Form Question 3	481
Figure 400: Presurvey Form Question 4	482
Figure 401: Presurvey Form Question 5	482
Figure 402: Presurvey Form Question 6	483
Figure 403: Presurvey Form Question 7	483
Figure 404: Presurvey Form Question 8	484
Figure 405: Presurvey Form Question 9	484
Figure 406: Presurvey Form Question 10	485
Figure 407: Presurvey Form Question 11	485
Figure 408: Presurvey Form Question 12	486
Figure 409: Presurvey Form Question 13	486
Figure 410: Presurvey Form Question 14	487
Figure 411: Presurvey Form Question 15	487
Figure 412: Presurvey Form Question 16	488
Figure 413: Presurvey Form Question 17	488
Figure 414: Presurvey Form Question 18	489
Figure 415: Filled Post survey Form.....	494
Figure 416: Filled Post survey Form 2.....	495
Figure 417: Post survey Form Question 1.....	496
Figure 418: Post survey Form Question 3.....	496
Figure 419: Post survey Form Question 4.....	497
Figure 420: Post survey Form Question 5.....	497
Figure 421: Post survey Form Question 6.....	498
Figure 422: Post survey Form Question 7.....	498
Figure 423: Post survey Form Question 8.....	499
Figure 424: Post survey Form Question 9.....	499
Figure 425: Post survey Form Question 10.....	500

Figure 426: Post survey Form Question 11	500
Figure 427: Post survey Form Question 13.....	501
Figure 428: Post survey Form Question 14.....	501
Figure 429: Unit Test Auth1 Setup	505
Figure 430: Unit Test Auth1 Automation Script 1	505
Figure 431: Unit Test Auth1 Automation Script 2	506
Figure 432: Unit Test Auth1 Automation Script 3	506
Figure 433: Unit Test Auth1 Automation Script 4	507
Figure 434: Unit Test Auth1 Automation Script 5	507
Figure 435: Unit Test Auth2 Setup	508
Figure 436: Unit Test Auth2 Automation Script 1	508
Figure 437: Unit Test Auth2 Automation Script 2	508
Figure 438: Unit Test Auth2 Automation Script 3	509
Figure 439: Unit Test Auth2 Automation Script 4	509
Figure 440: Unit Test Auth2 Automation Script 5	509
Figure 441: Unit Test Auth3 Setup	510
Figure 442: Unit Test Auth3 Automation Script 1	510
Figure 443: Unit Test Auth3 Automation Script 2	511
Figure 444: Unit Test Auth3 Automation Script 3	511
Figure 445: Unit Test Auth3 Automation Script 4	511
Figure 446: Unit Test Auth3 Automation Script 5	512
Figure 447: Unit Test Admin1 Automation Setup	513
Figure 448: Unit Test Admin1 Automation Script 1	514
Figure 449 Unit Test Admin1 Automation Script 2	514
Figure 450 Unit Test Admin1 Automation Script 3	515
Figure 451 Unit Test Admin2 Setup	516
Figure 452 Unit Test Admin2 Automation Script 1	517
Figure 453 Unit Test Admin2 Automation Script 2	517
Figure 454 Unit Test Admin2 Automation Script 3	518
Figure 455 Unit Test Admin2 Automation Script 4	519
Figure 456 Unit Test Admin3 Setup	520
Figure 457 Unit Test Admin3 Setup 2	521
Figure 458 Unit Test Admin3 Automation Script 1	521

Figure 459 Unit Test Admin3 Automation Script 2	522
Figure 460 Unit Test Admin3 Automation Script 3	522
Figure 461 Unit Test Admin3 Automation Script 4	523
Figure 462 Unit Test Admin3 Automation Script 4	524
Figure 463 Unit Test Admin4 Setup	525
Figure 464 Unit Test Admin4 Script Automation 1	526
Figure 465 Unit Test Admin4 Script Automation 2	526
Figure 466 Unit Test Admin4 Script Automation 3	527
Figure 467 Unit Test Admin4 Script Automation 4	527
Figure 468 Unit Test Admin4 Script Automation 5	528
Figure 469 Unit Test Admin4 Script Automation 6	528
Figure 470 Unit Test Admin4 Script Automation 7	529
Figure 471 Unit Test Appt1 Setup	530
Figure 472 Unit Test Appt1 Automation Script 1	530
Figure 473 Unit Test Appt1 Automation Script 2	530
Figure 474 Unit Test Appt1 Automation Script 3	531
Figure 475 Unit Test Appt2 Setup	532
Figure 476 Unit Test Appt2 Automation Script 1	532
Figure 477 Unit Test Appt2 Automation Script 2	533
Figure 478 Unit Test Appt2 Automation Script 3	533
Figure 479 Unit Test Appt2 Automation Script 3	534
Figure 480 Unit Test Appt2 Automation Script 4	534
Figure 481 Unit Test Appt3 Setup	535
Figure 482 Unit Test Appt3 Automation Script 1	535
Figure 483 Unit Test Appt3 Automation Script 2	536
Figure 484 Unit Test Appt3 Automation Script 3	536
Figure 485 Unit Test Appt4 Setup	537
Figure 486 Unit Test Appt4 Automation Script 1	538
Figure 487 Unit Test Appt4 Automation Script 2	539
Figure 488 Unit Test Appt4 Automation Script 3	539
Figure 489 Unit Test Appt5 Setup	540
Figure 490 Unit Test Appt5 Setup 2	541
Figure 491 Unit Test Appt5 Setup 3	542

Figure 492 Unit Test Appt5 Automation Script 1	543
Figure 493 Unit Test Appt5 Automation Script 2	544
Figure 494 Unit Test Appt5 Automation Script 3	544
Figure 495 Unit Test Appt6 Setup	545
Figure 496 Unit Test Appt6 Setup 2	546
Figure 497 Unit Test Appt6 Automation Script 1	547
Figure 498 Unit Test Appt6 Automation Script 2	547
Figure 499 Unit Test Appt6 Automation Script 3	548
Figure 500 Unit Test Appt7 Setup	549
Figure 501 Unit Test Appt7 Setup	550
Figure 502 Unit Test Appt7 Automation Script 1	551
Figure 503 Unit Test Appt7 Automation Script 2	551
Figure 504 Unit Test Appt7 Automation Script 3	552
Figure 505 Unit Test Appt8 Setup	553
Figure 506 Unit Test Appt8 Setup	554
Figure 507 Unit Test Appt8 Automation Script 1	555
Figure 508 Unit Test Appt8 Automation Script 2	556
Figure 509 Unit Test Appt8 Automation Script 3	556
Figure 510 Unit Test Appt9 Setup	557
Figure 511 Unit Test Appt9 Setup	558
Figure 512 Unit Test Appt9 Automation Script 1	558
Figure 513Unit Test Appt9 Automation Script 2	559
Figure 514 Unit Test Appt9 Automation Script 3	559
Figure 515 Unit Test Appt9 Automation Script 4	560
Figure 516 Unit Test Appt9 Automation Script 5	560
Figure 517 Unit Test Appt9 Automation Script 6	561
Figure 518 Unit Test Appt10 Setup	562
Figure 519 Unit Test Appt10 Setup 2	563
Figure 520 Unit Test Appt10 Automation Script 1	563
Figure 521 Unit Test Appt10 Automation Script 2	564
Figure 522 Unit Test Appt10 Automation Script 3	564
Figure 523 Unit Test Appt10 Automation Script 4	565
Figure 524 Unit Test Appt10 Automation Script 5	565

Figure 525 Unit Test Po1 Setup	566
Figure 526 Unit Test Po1 Automation Script 1	567
Figure 527 Unit Test Po1 Automation Script 2	567
Figure 528 Unit Test Po1 Automation Script 3	568
Figure 529 Unit Test Po1 Automation Script 4	569
Figure 530 Unit Test Po1 Automation Script 5	569
Figure 531 Unit Test Po1 Automation Script 6	570
Figure 532 Unit Test Po2 Setup	571
Figure 533 Unit Test Po2 Automation Script 1	572
Figure 534 Unit Test Po2 Automation Script 2	572
Figure 535 Unit Test Po2 Automation Script 3	573
Figure 536 Unit Test Po2 Automation Script 4	574
Figure 537 Unit Test Po2 Automation Script 5	575
Figure 538 Unit Test Po3 Setup	576
Figure 539 Unit Test Po3 Automation Script 1	577
Figure 540 Unit Test Po3 Automation Script 2	577
Figure 541 Unit Test Po3 Automation Script 3	578
Figure 542 Unit Test Po3 Automation Script 4	578
Figure 543 Unit Test Po3 Automation Script 5	579
Figure 544 Unit Test Po3 Automation Script 6	579
Figure 545 Unit Test Po3 Automation Script 7	579
Figure 546 Unit Test Po4 Setup	580
Figure 547 Unit Test Po4 Automation Script 1	580
Figure 548 Unit Test Po4 Automation Script 2	581
Figure 549 Unit Test Po4 Automation Script 3	581
Figure 550 Unit Test Po4 Automation Script 4	582
Figure 551 Unit Test Po4 Automation Script 5	582
Figure 552 Unit Test Po4 Automation Script 6	582
Figure 553 Unit Test Po4 Automation Script 7	583
Figure 554 Unit Test RTC1 Setup	584
Figure 555 Unit Test RTC1 Automation Script 1	584
Figure 556 Unit Test RTC1 Automation Script 2	585
Figure 557 Unit Test RTC1 Automation Script 3	586

Figure 558 Unit Test RTC1 Automation Script 4	586
Figure 559 Unit Test RTC1 Automation Script 5	587
Figure 560 Unit Test RTC1 Automation Script 5	588
Figure 561 Unit Test RTC1 Automation Script 6	588
Figure 562 Unit Test RTC1 Automation Script 7	589
Figure 563 Unit Test RTC1 Automation Script 7	590
Figure 564 Old Gantt Chart.....	591
Figure 565 Old WBS	593
Figure 566 Old ERD	594
Figure 567 Old DFD0	596
Figure 568 Old Usecase.....	597
Figure 569 Old Activity Diagram for appointment.....	598
Figure 570 Old Sequence Diagram for appointment	599
Figure 571 Old Collaboration Diagram for appointment	601
Figure 572 Old Class Diagram	602
Figure 573 Draft Figma Design's Board	605
Figure 574 Main Figma Design's Board	606
Figure 575 User Feedback form: Initial Development Question 2	607
Figure 576 Feedback form: Initial Development Question 3.....	607
Figure 577 Feedback form: Initial Development Question 4.....	608
Figure 578 Feedback form: Initial Development Question 5.....	608
Figure 579 Feedback form: Initial Development Question 6.....	609
Figure 580 Feedback form: Initial Development Question 7.....	609
Figure 581 Feedback form: Initial Development Question 8.....	610
Figure 582: Initial Development Sample User Feedback Form	612
Figure 583 Feedback form: Prototype 1 Question 1	613
Figure 584 Feedback form: Prototype 1 Question 2.....	613
Figure 585 Feedback form: Prototype 1 Question 4.....	614
Figure 586 Feedback form: Prototype 1 Question 5.....	614
Figure 587 Feedback form: Prototype 1 Question 6.....	615
Figure 588 Feedback form: Prototype 1 Question 7.....	615
Figure 589 Feedback form: Prototype 1 Question 8.....	616
Figure 590 Prototype 1 Sample User Feedback form	618

Figure 591 Feedback form: Prototype 2 Question 2	619
Figure 592 Feedback form: Prototype 2 Question 3.....	619
Figure 593 Feedback form: Prototype 2 Question 4.....	620
Figure 594 Feedback form: Prototype 2 Question 5.....	620
Figure 595 Feedback form: Prototype 2 Question 6.....	621
Figure 596 Feedback form: Prototype 2 Question 7.....	621
Figure 597 Feedback form: Prototype 2 Question 8.....	622
Figure 598 Feedback form: Prototype 2 Question 9.....	622
Figure 599 Feedback form: Prototype 2 Question 1	623
Figure 600 Prototype 2 Filled User Feedback form.....	626
Figure 601 Feedback form: Prototype 3 Question 2.....	627
Figure 602 Feedback form: Prototype 3 Question 3.....	627
Figure 603 Feedback form: Prototype 3 Question 4.....	628
Figure 604 Feedback form: Prototype 3 Question 5.....	628
Figure 605 Feedback form: Prototype 3 Question 6.....	629
Figure 606 Feedback form: Prototype 3 Question 7.....	629
Figure 607 Prototype 3 Filled User Feedback Form.....	632
Figure 608 Feedback form: Prototype 4 Question 2.....	633
Figure 609 Feedback form: Prototype 4 Question 3.....	633
Figure 610 Feedback form: Prototype 4 Question 4.....	634
Figure 611 Feedback form: Prototype 4 Question 5.....	634
Figure 612 Prototype 4 Filled User Feedback Form.....	636
Figure 613 Feedback form: Final Prototype Question 2	637
Figure 614 Feedback form: Prototype 4 Question 3.....	637
Figure 615 Feedback form: Prototype 4 Question 4.....	638
Figure 616 Feedback form: Prototype 4 Question 5.....	638
Figure 617 Feedback form: Prototype 4 Question 6.....	639
Figure 618 Feedback form: Prototype 4 Question 7.....	639
Figure 619 Final Prototype Filled User Feedback Form	642
Figure 620: Screenshot of Research Paper 1	648
Figure 621: Screenshot of Research Paper 2	650
Figure 622: Screenshot of Research Paper 3	652
Figure 623: Screenshot of Research Paper 4	654

Figure 624: Screenshot of Research Paper 5 655

Table of Tables

Table 1: Comparison table of Similar Systems.....	14
Table 2 Reasons for Choosing Prototyping.....	17
Table 3 Functional Requirement: Registration:	28
Table 4 Functional Requirement: Login.....	30
Table 5 Functional Requirement: Appointment	32
Table 6 Functional Requirement: Blog Posts	33
Table 7 Functional Requirement: Real Time Chat	34
Table 8 Functional Requirement: Admin Dashboard.....	35
Table 9: Software Interfaces	36
Table 10: Performance Requirements.....	38
Table 11: Safety and Security Requirements	38
Table 12: Other Software Quality Attributes	39
Table 13:Unit Test - Registration.....	196
Table 14:Unit Test – Login	200
Table 15:Unit Test - Admin Login.....	203
Table 16:Unit Test - Vet Approval	206
Table 17:Unit Test - Vet Decline	209
Table 18:Unit Test - Content Moderation	213
Table 19:Unit Test - Category Management	217
Table 20:Unit Test - Add Schedule	220
Table 21:Unit Test - Edit Schedule.....	225
Table 22:Unit Test - Booking via Khalti	229
Table 23:Unit Test - Booking via Store Credit	234
Table 24:Unit Test - Accept Booking	239
Table 25:Unit Test - Reject Booking.....	243
Table 26:Unit Test - Mark as Complete.....	247
Table 27:Unit Test - Booking Cancellation	251
Table 28:Unit Test - Khalti Payment.....	253
Table 29:Unit Test - Review	257
Table 30:Unit Test - Post Creation and Deletion	261
Table 31:Unit Test - Post Updating and Liking	265

Table 32:Unit Test - Like, Delete and Add Comments	269
Table 33:Unit Test - Add and Delete Reply	273
Table 34:Unit Test - Real Time Chat.....	278
Table 35:System Test - Pet Owner Registration	281
Table 36:System Test - Login and Logout.....	288
Table 37:System Test - Vet Registration.....	294
Table 38:System Test - Pet Adding, Updating and Deleting	306
Table 39:System Test - Schedule Addition, Deletion and Updating	317
Table 40:System Test - Book Appointment via Khalti.....	331
Table 41:System Test - Review	341
Table 42:System Test - Decline Appointment	346
Table 43:System Test - Book Appointment via Store Credit and Cancellation.....	352
Table 44:System Test - Time Slot Availability Checking	358
Table 45:System Test - Real Time Chat	363
Table 46:System Test - Reply Addition and Deletion	367
Table 47:System Test - Comment Addition and Deletion.....	376
Table 48:System Test - Like and Unlike Toggle	381
Table 49:System Test - Post Addition, Deletion and Updating.....	385
Table 50:System Test - Administrator Login	396
Table 51:System Test - Category Management.....	400
Table 52:System Test - Content Moderation.....	408
Table 53: System Test - Password Change	420
Table 54:System Test - Forgot Password	429
Table 55:System Test - Search.....	435
Table 56:System Test - User Profile Update	440
Table 57:System Test - Email Notifications and Contact Form	450
Table 58:Reason for not choosing Considered Methodologies	646

Chapter 1: Introduction

1.1 Project Description

In recent years, the pet care industry growth has been astronomical with more and more pet owners looking for high quality but convenient care for their pets. Although advances have been made, challenges are still present, as getting the right vet arranging appointments appropriately and ensuring that pet owners and vets communicate effectively. Current systems definitely answer at least part of the problem, but more often than not real-time communication is missing, booking is too tedious and information sharing spread across different channels.

To address this void, PetVet was developed—a holistic platform to improve and streamline pet care service management. PetVet allows veterinarians to book and manage their appointments themselves while giving pet owners a simple-to-use scheduling system for their appointments. It is also integrated with a social post feature where users can post, comment, and like—allowing for socialization between pet owners. It supports a real-time chat feature for one-to-one communication between pet owners and vets, pre-empting both delay and confusion. For optimal administration, there is an admin dashboard that monitors interactions between users and content to keep the platform secure, efficient, and uniform for all users.

1.2 Current Scenario

1.2.1 Global Context

The change of pet ownership is even more obvious when we focus on America, from a 56% of homes with pets in 1988 to a predicted 66% by 2024, meaning that nearly 86.9 million American residences own at least one pet (Megna, 2024). As pets have come to be considered more like family, their company and emotional solace has become central in the work of many soldiers. Veterinarian and former President of the World Small Animal Veterinary Association, Dr Shane Ryan, highlights that the pet-keeping culture is evolving greatly and expects similar shifts in new markets like South-East Asia, South Asia, Central Asia, Sub-Saharan Africa or Eastern Europe (Heath For Animals, 2022).

Given these dynamic requirements and given the inconsistencies existing within available systems, this work sets off to the development of a robust website that will well bridge veterinarians and pet owners effectively. In effect, the essence is to make appointment schedules much easier to handle, facilitate real-time interactions, and nurture community involvement through blogging and discussion forums with far-enhanced experiences for both veterinarians and pet owners.

1.2.2 Nepal Context

Exports of pet food have developed a growth trend in Nepal, reaching Rs 2.26 billion in the first nine months of this fiscal year, up Rs 300 million compared to the same period in the previous fiscal year. A total amount of Rs 1.95 billion was exported to the United States of America alone-a major market-so far. It is indicative of the increased demand for pet products and, thus, growing awareness regarding pets and their care in Nepal. Hence, the market is likely to expand more in the days to come with an increasing preference of consumers for quality pet food. (Republica, 2023).

The demand for pet dogs, particularly those of the popular breeds like pugs, German Shepherds, and Labrador Retrievers, is growing day by day in the Kathmandu Valley. Currently, about 30,000 dogs are housed in Nepali kennel clubs, while businesses claim it shows growth of around 20 percent annually. Operators claim the market to be worth around Rs 72 million annually. Needless to say, with an increasing number of pet animals and their requirement for regular veterinary care, the need for an appointment system that would handle the bookings and hence communicate effectively with the vets develops (Kc, 2016).

1.3 Problem Domain

These connectivity options between veterinarians and owners are usually rife with inefficiencies, lack of transparency, and various communication options. It becomes hard, therefore, for pet owners to be able to search for qualified vets, book appointments successfully, and communicate with the said vets for both follow-up and emergency consultations. This leads to delayed medical attention, poor customer satisfaction, and, in general, a fragmented experience concerning pet care. While their appointment schemes are time-consuming, discussing every detail with a pet owner and advertising services while traditional systems are phone-based or through third-party apps which don't update in real-time nor offer flexible scheduling. There is no platform for either party to share knowledge or for pet owners to ask questions and build a community of likeminded people that are considering pets' care.

Analysis of five years of survey data points to a wide gap in communication between the veterinary team and their clients. In fact, officials with Partners for Healthy Pets said that data reveals many times pet owners don't really understand what members of the health care teams are communicating concerning the care of their pets. It was demonstrated that most dog and cat owners are not cognizant of what takes place during their animal's physical examination and of the value of the services rendered from a veterinarian. This gap is an opportunity lost for the veterinarian to ramp up communications and client education regarding the value of preventive health care examinations (AVMA.org, 2018).

1.4 Project as a Solution

The digital communication revolution brought with it instant feedbacks from firms and individuals. It solved problems in real-time and helped resolve many issues. Basically, most firms have now transformed to creating and nurturing loyal communities through such tools that allow instant two-way communication. This in turn normally indicates a higher return on investment, since through the use of digital communication, organizations will be able to reach almost any place in the world without having to set up a physical presence necessarily (DocPath, 2023).

All these merits are being tapped to the maximum in the proposed website i.e an all-encompassing channel of digital communication relating to veterinary care. Owners and veterinarians can easily interact over the portal by solving everyday issues with scheduling, communication, and community building. Pet owners can look for available vets, peek through a detailed profile, and get a booking done in real time. It would also be a hyper view for the veterinarians, including scheduling management tools, appointment acceptance or rejection, and interaction directly with the owners via an integrated chat system.

For community interaction, it will further go a long way by adding a blog to accompany the forum, in order to post the results and give room for questions within this interactive environment. It shall have an Admin panel in order to ensure that the shared content is appropriate and safe. That way, the website will be of great help and hence make life so easy for either party as it will attempt to bridge the gaps that exist in the current veterinary care systems.

1.5 Aim and Objectives

1.5.1 Aim

The primary aim of the project is to develop an interactive web-based platform that connects veterinarians and pet owners, enabling seamless management of appointments, real-time communication, and community engagement through blogs and forums.

1.5.2 Objectives

- Develop a user-friendly website that allows pet owners to search for veterinarians, view their profiles, and book appointments based on availability.
- Implement a secure and efficient appointment management system that allows vets to accept or decline appointments, set their availability, and manage their schedules.
- Create a real-time chat system for direct communication between vets and pet owners.
- Design and develop a blog and forum feature where both vets and pet owners can create posts, share articles, and interact with each other.
- Build an admin dashboard to manage user registrations, monitor activities, and moderate content.

1.6 Structure of The Report

1.6.1 Background

This section forms the foundation of the project by identifying end users and their needs, ensuring the solution is aligned with those needs. It includes research through journal reviews and analysis of existing systems, along with a comparison chart to justify design choices and highlight areas for improvement.

1.6.2 Development

The development section outlines the full system build process, from methodology selection to iterative prototyping. It covers UI design, coding, UML diagrams, testing, user feedback, and how survey results and the SRS guided system design and development.

1.6.3 Testing and Analysis

This section validates the system using Django's built-in testing framework for unit testing and black box testing for system-level checks. All major features and user interactions were tested to ensure reliability, correctness, and usability.

1.6.4 Conclusion

The project conclusion outlines the legal, social, and ethical considerations while highlighting key benefits like appointment management, real-time chat, and admin controls. It also identifies limitations in UI responsiveness and notifications. Based on feedback, future improvements include push notifications, file sharing in chat, and mobile app development.

Chapter 2: Background

2.1 About the End User

There are two primary user groups targeted by my website: veterinarians—commonly referred to as vets throughout the project—and pet owners, both of whom interact with the system based on their specific roles and needs. They are discussed below:

- Animal welfare is closely linked to global sustainable development goals and veterinarians and veterinary para professional (VPPs) are key stakeholders in animal welfare. The FAO affirms that a sustainable livestock system must raise the issue of animal welfare given that this influences human health, nutrition, environment and stable economy. Based on the fact that veterinarians are usually seen as trusted advocates for animal welfare, this belief is an easy for one to see and believe. Besides diagnosing and treating animal diseases, they are playing a pivotal role when it comes to managing pain, proper care and welfare practice. They are essential users of any system aimed at animal health and welfare because of their involvement and motivation (Doyle, et al., 2021).
- Today, the majority of the people in modern society are pet owners, about 67 percent of U.S. households have at least one pet, about 63 million with dogs and 42 million with cats. For these animals these are not just creatures we live with, they are our emotional, recreational and occasionally functional partners. However, this is explained using the concept of human–animal bond as a mutually advantageous relationship between humans and animals for their respective well being. Due to this fact, pet owners show great interest in the health and quality of life of their pets, being interested in receiving timely and reliable veterinary services and means of ensuring responsible pet care (Scoresby, et al., 2021).

2.2 Understanding the Solution

My project addresses the distinct needs of two primary user groups: veterinarians (vets) and pet owners, by offering a centralized, user-friendly platform that enhances efficiency, convenience, and communication for both parties.

1. For Pet Owners:

It proves to be challenging for pet owners to make vet appointments, especially outside of regular clinic hours. Using an appointment system over the Internet, the website offers:

- 24/7 Access: Allows clients to make appointments at their own convenient time, reducing the need to call during business hours.
- Improved Communication: Allows pet owners to better communicate with veterinary clinics, resulting in an enhanced overall client experience.
- Empowerment: Allows pet owners to take an active part in their pets' health care requirements.

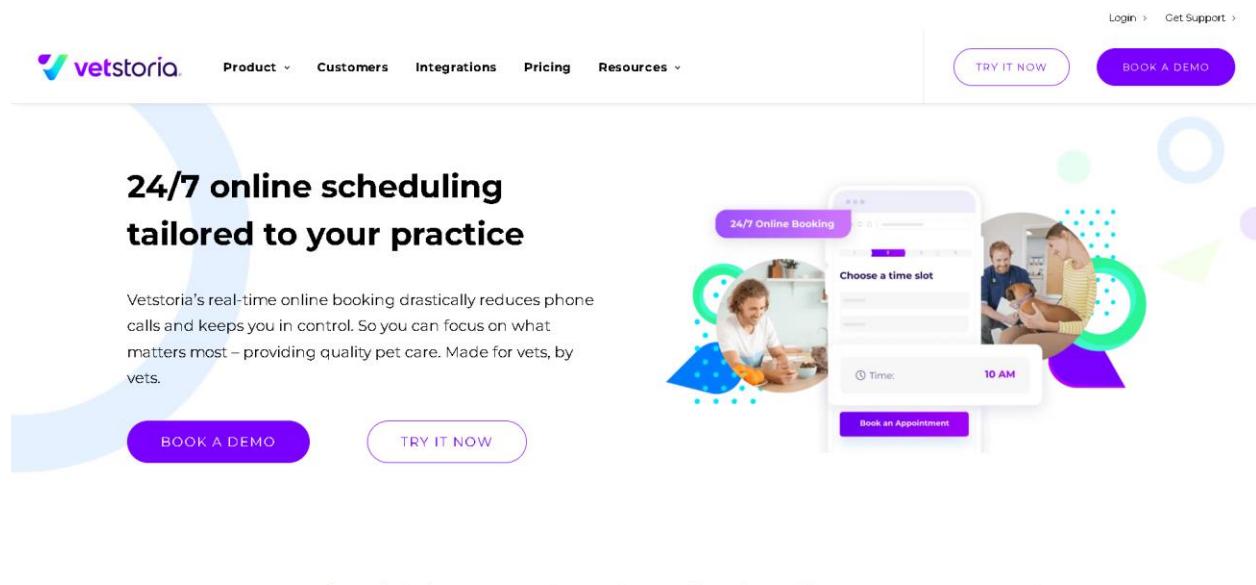
2. For Veterinarians Veterinary clinics can utilize the platform by:

- Operational efficiency: It enables appointment scheduling, minimizes administrator workload, and allows staff to dedicate more time to patient care.
- Flexible Scheduling: Vets can create, modify, or delete their available time slots.
- Appointment Management: Ability to accept, reject based on availability
- Blogging & Updates: Vets can post blogs or informational articles to engage with pet owners and provide useful health tips.

2.3 Similar Projects

2.3.1 Vetstoria

Vetstoria is a platform that allows practices to simplify how they manage their schedules by reducing phone calls and offering real-time online booking, fully customized for the unique needs of each practice. Founded by a couple of veterinarians, Vetstoria embodies years of veterinary experience combined with deep software knowledge in developing an online booking system that is fully integrated with PIMS in real time and customizable. It gives full control over the schedule to practices while offering convenience in the form of digital intervention to pet owners (Vetstoria, 2025).



Over **6000 practices** globally use Vetstoria to reduce calls and provide great pet owner experiences

Figure 1: Screenshot of Similar Application:Vetstoria

2.3.2 Vetster

Vetster is the world's largest marketplace for veterinary professionals, serving pet owners with over 6,000 practitioners every day. It avails the convenience of 24/7 booking of vet appointments, online prescriptions, refills, and diagnostic testing. A pet owner can gain access to a digital health history and have tailored treatment plans drawn up to suit his needs. Besides, Vetster has proprietary video and messaging for direct, seamless remote consultations between pet owners and veterinarians (Vetster, 2025).

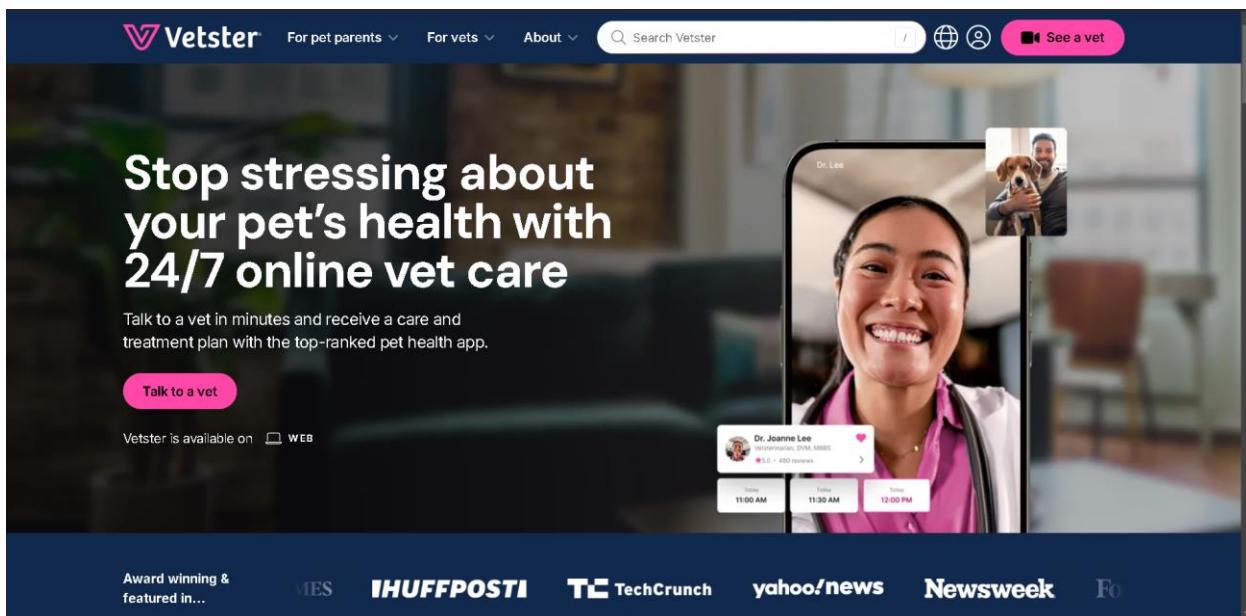


Figure 2: Screenshot of Similar Application:Vetster

2.3.3 DaySmart Pet

DaySmart Pet is a fully customizable online booking website for pet care businesses to be live 24*7 for appointment bookings. Automated text and email reminders minimize no-shows, save time, and increase efficiency. It's pretty easy to manage your payroll on the platform. It is pretty simple to set staff member salaries or commission settings, and payroll calculations are always available anywhere. Yet another facility offered by DaySmart Pet is its client management and their pets with ease and in minimum time; one can get easy access to clients' contact information, pets, and appointment history (DaySmart Pet, 2025).

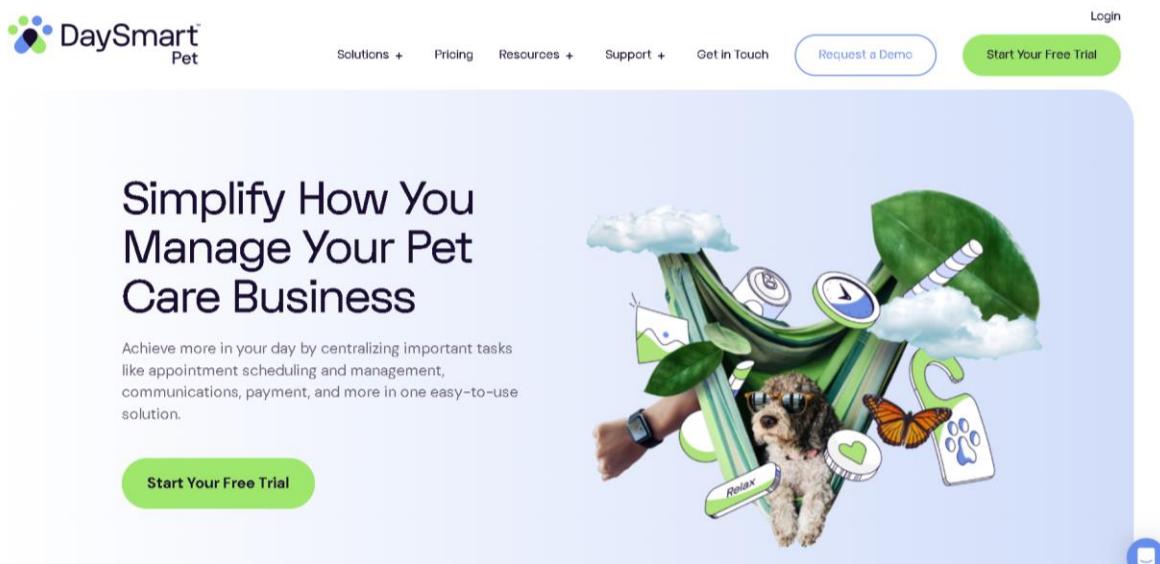


Figure 3: Screenshot of Similar Application:DaySmartPet

2.3.4 PetDesk

Modernize traditional veterinary practice management systems with new-school communication across app, text, and email messaging to communicate fluidly with clients. Once connected, PetDesk works in lockstep with your management system for an innovative approach to practice efficiency. Easy integration is set up quickly, and total customization is provided by the PetDesk team. What's more, PetDesk offers customized training with your staff and full-service support via live chat and phone to ensure you get the most from its platform (Pet Desk, 2025).

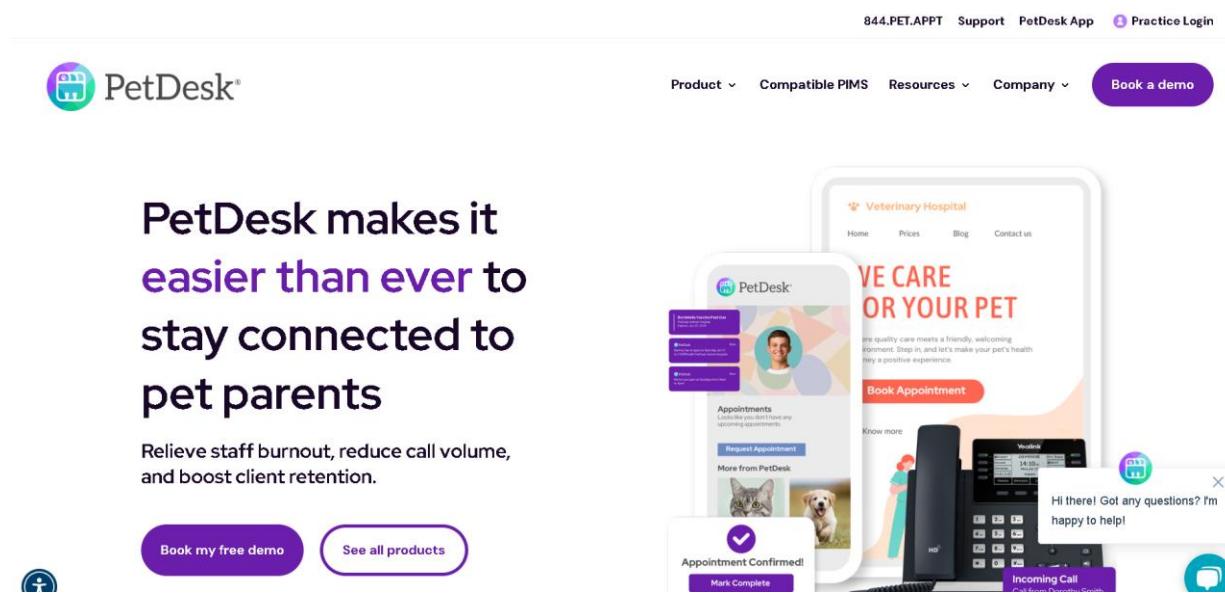


Figure 4: Screenshot of Similar Application:PetDesk

2.4 Comparisons

This chart provides an overview of various websites related to veterinarians, highlighting their features and drawing a comparison with the features offered by my website.

Features	Vetstoria	Vetster	DaySmart Pet	PetDesk	Appointy	Proposed System
Direct Video Calling	✗	✓	✗	✗	✗	✗
Client Profiles	✓	✓	✓	✓	✓	✓
Sync with Practice Management System	✓	✗	✗	✓	✗	✗
Training and Support	✗	✗	✗	✓	✓	✗
Admin Dashboard	✗	✗	✗	✗	✓	✓
Forum For Client	✗	✗	✗	✗	✗	✓
Flagging Inappropriate Content	✗	✗	✗	✗	✗	✓
Vet Verification	✗	✗	✗	✓	✗	✓
Digital Health History and Information	✓	✗	✗	✗	✗	✓
Online Prescriptions	✓	✓	✗	✗	✗	✗
Chat System	✓	✓	✓	✓	✓	✓

Table 1: Comparison table of Similar Systems

Chapter 3: Development

3.1 Considered Methodologies

The Agile methodologies considered within the selection of an optimum methodology for this project are Scrum, Extreme Programming, and Kanban.

3.1.1 Scrum

Scrum is indeed a very strong framework, correctly tailored for iterative development, placing significant emphasis on team collaboration and sprints. Projects with wide, unhandled, or even not-so-concrete requirements are good to go. Its flexibility and capabilities for reevaluation empower teams to flow along the changes. However, projects having stringent and concrete requirements may not fit in here (Carvalho & Mello, 2011).

3.1.2 Extreme Programming

XP emphasizes continuous code revision, often using pair programming to identify and resolve issues efficiently. It relies heavily on frequent testing, including unit tests and ongoing customer feedback, and uses short development iterations to deliver updates quickly (Fojtik, 2010). However, as I am the sole developer working on the VetPet web app, the collaborative aspects of XP, like pair programming and constant customer interaction, are not applicable to my project.

3.1.3 Kanban

Kanban is a management method aimed at visual representation of project workflows to teams so that they might understand how their efforts contribute to the overall process. This transparency keeps redundancy at a minimum, prevents wasted effort, and ensures continuous delivery. However, in my project, since tasks are handled individually, there is no need in my workflow to manage multiple tasks flowing through the Kanban system (Simplilearn, 2024).

The reasons for not selecting these methodologies are briefly discussed in the Appendix section: 7.8 Appendix H: Considered Methodologies

3.2 Selected Methodology

3.2.1 About Prototyping

The Prototype Model emphasizes building the software itself rather than putting extra effort into the documentation of it. Because of this, the software is released much earlier compared to other methods. There is major participation from the users in prototyping. Users can interact with the prototype built and give all the feedback and specifications required in detail. This can prevent lots of misunderstanding between the developers and the users because both parties will have a mutual understanding of what to expect if there is a prototype available that can be reviewed. For this reason, the final product is sure to meet the user's expectations concerning look and feel, functionality, and performance (Sabale & Dani, 2012).

3.2.2 Reasons for Choosing Prototyping

Justification	Possible Scenarios
Allows iterative development, making it easy to improve based on feedback after each cycle (Slideshare, 2023).	After the initial prototype of appointment booking, users might request the ability to upload medical records or pet history, which can be incorporated in the next development cycle.
Reduces risk by identifying and resolving potential issues early in the development cycle (Slideshare, 2023)	A security flaw, such as unauthorized access to profiles, might be discovered during the login module testing and resolved early.
Encourages early partial delivery for demonstrations or supervisor feedback.	A demo of the login and dashboard UI can be shown to the supervisor even before the full backend is ready.
Adapts well when all user requirements are not clearly known from the start.	New feature ideas like "liking comments" or "filtering vets by specialty" could emerge during early feedback.

Cost-effective and supports experimentation with design and implementation.	Different UI layouts for the appointment scheduler can be tested with users before finalizing the best one.
Promotes better alignment with user expectations and market demand.	Community posting features could be adjusted based on user feedback to make it more engaging.
Easy to manage scope changes without affecting already stable modules.	After OTP login is added, the rest of the application like appointment booking and review remains untouched.
Helps meet deadlines by delivering critical features first and gradually building the full system.	The system could go live with login, vet profiles, and appointment booking first, while reviews and payments are added later.

Table 2 Reasons for Choosing Prototyping

3.3 Phases of Methodology

The following are the phases of Evolutionary Prototyping methodology:

1. Requirements Gathering and Analysis

In this phase initial or incomplete or high level requirements are collected from the stakeholders. To understand user expectation, techniques such as interviews and questionnaires are used. This will be done in successive iterations, and these requirements will be refined (geeksforgeeks, 2024).

2. Quick Design

The initial requirements are used to create a preliminary design. Essential system features and interfaces are described and users are given a perception of the system's visual and functional idea, but without getting into full technical depth (geeksforgeeks, 2024).

3. Build Prototype

A working prototype of the system is developed based on the quick design. It is a scaled-down version of the actual system and may lack full functionality but includes enough features for user interaction and evaluation.

4. User Evaluation and Feedback

The prototype is demonstrated to stakeholders who interact with it to assess whether it meets their needs. Feedback is collected on both functionality and usability, identifying any changes or additional requirements (geeksforgeeks, 2024).

5. Refining the Prototype

Based on the feedback, the prototype is refined. Missing features are added, and existing components are adjusted to better match user expectations. This step may repeat several times as part of the iterative process.

6. User Acceptance Decision

Once the refined prototype meets user requirements, a formal acceptance decision is made. If accepted, the prototype serves as the basis for developing the final product (geeksforgeeks, 2024).

7. Implementation and Maintenance

The final system is built, thoroughly tested, and deployed. After deployment, the system is maintained regularly to fix bugs, optimize performance, and support evolving user needs.

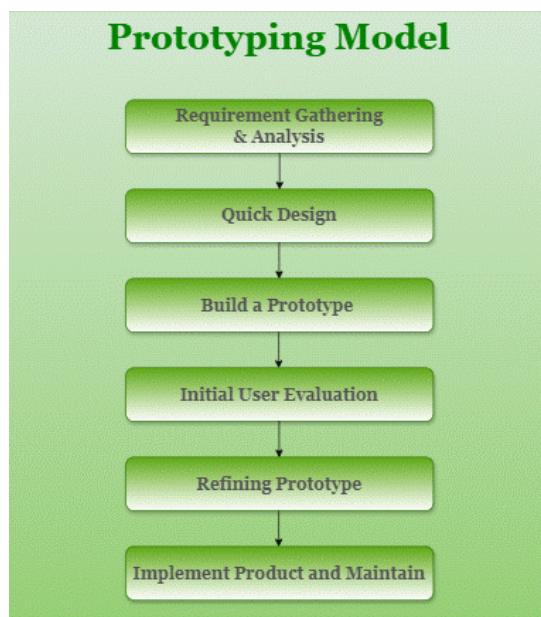


Figure 5: Phases of prototyping model

3.4 Survey Results

3.4.1 Pre-Survey Results

Before developing PetVet, a user survey was conducted to understand the target audience's needs, behavior, and expectations.

The findings from the pre survey show that there is a huge demand for an online platform like PetVet among the pet owners specifically for the services like booking vet appointments, checking real time vet availability and professional advice. Users largely trusted an online platform for pet healthcare, except that features like live chat and fast response were also very popular among them. It also showed a strong interest in a community-based forum where pet owners can talk with one another and discuss and back their experiences with one another. While respondents noted that the platform should suggest more features such as the sale of pet products, they emphasize that the platform should only offer its core veterinary services at first. At the same time, the user challenges posed by managing pet diets and dealing with an emotional attachment make it necessary for PetVet to provide practical tools that could ease them out. Overall PetVet feedback points to the potential for making PetVet a valuable resource for animal owners with opportunities to improve.

Below is the link that leads to the actual charts and a sample filled pre-survey form:

7.1.2 Sample of Filled Pre-Survey Forms

7.1.3 Pre-Survey Result

3.4.2 Post-Survey Results

The post feature update survey showed that users were very positive about PetVet's latest improvements. User experience was made better by real time chat feature while being able to appointment book and vet schedule effortlessly is what was appreciated. The payment and store credit system was used by fewer users, but those who did were mostly clear and fair. Active admin monitoring made most participants feel safer and most said that they would continue to use PetVet if it was released publicly. Most of the feature requests were for a mobile app version, video consultations, vaccination tracking, and push notifications. Overall, the feedback indicates that PetVet is meeting user needs and has potential to expand the service offering and make the platform more accessible and convenient.

Below is the link that leads to the actual charts and a sample filled post-survey form:

7.2.2 Sample of Filled Post-Survey Forms

7.2.3 Post-Survey Result

3.5 Software Requirement Specifications

1.1 Introduction

1.1.1 Purpose

The functional and non-functional requirements for a web application that promotes community involvement and makes it easier for pet owners and veterinarians to schedule appointments are described in this document. It seeks to give users, developers, and stakeholders a comprehensive grasp of the capabilities and constraints of the system.

1.1.2 Indented Audiences and Reading Suggestion

- **Developers:** To guide the implementation of the system.
- **Stakeholders:** To ensure the system meets business goals
- **Testers:** To validate the application against specified requirements.
- **End-users:** To understand the functionalities provided.

1.1.3 Project Scope

The web application serves as a platform that streamlines appointment scheduling and communication between pet owners and veterinarians. It addresses challenges such as managing available times and tracking appointment histories. Additionally, the platform includes features for community engagement, such as a chat system for better interaction and personalized advice, a blogging feature, and an admin dashboard for content moderation and platform maintenance.

1.2 Overall Description

1.2.1 System Perspective

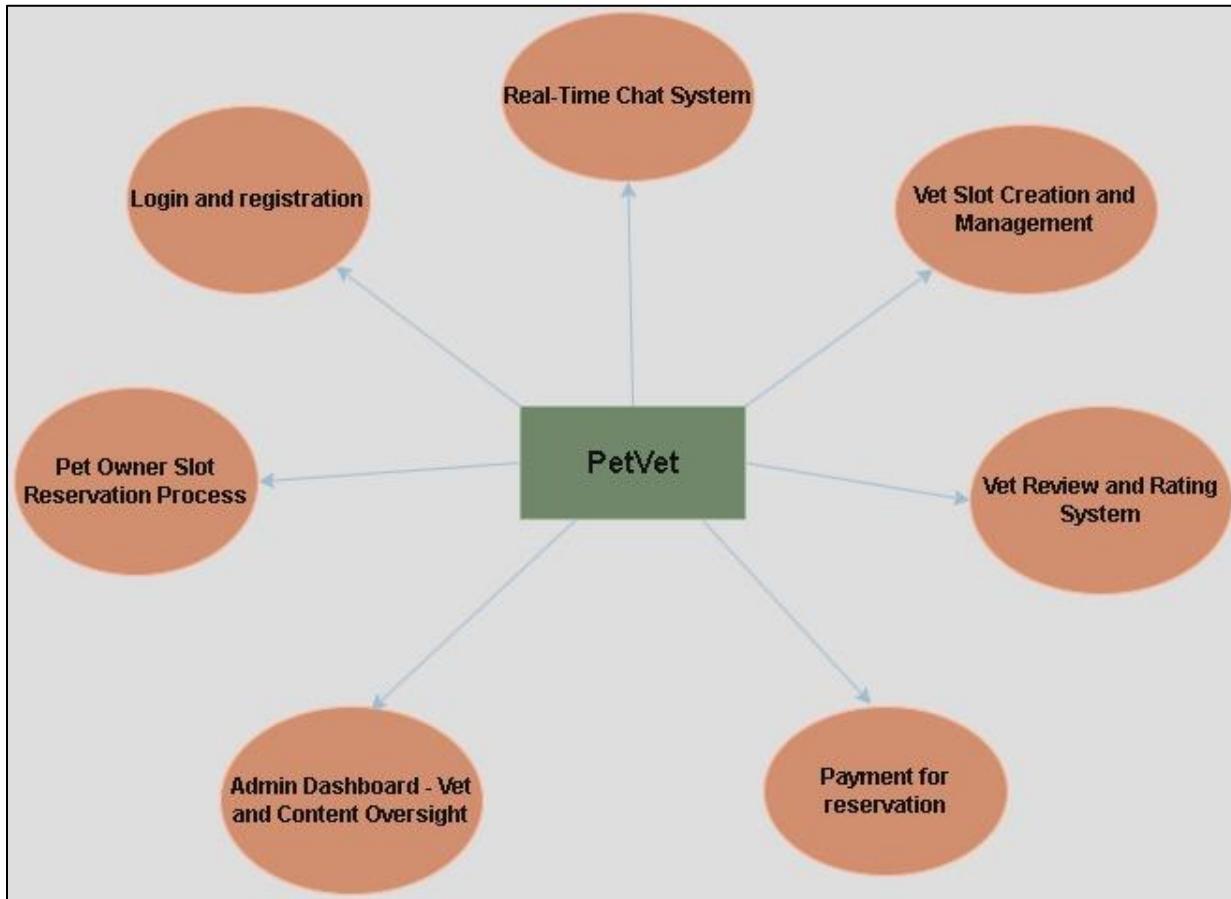


Figure 6: System Perspective Diagram

The Application provides an application-based interface to:

- **Developers:** To guide the implementation of the system.
- **Stakeholders:** To ensure the system meets business goals
- **Testers:** To validate the application against specified requirements.
- **End-users:** To understand the functionalities provided.

The system features are as following:

SF1: Appointment Schedule and Time Slot Creation

The system allows veterinarians to manage their availability and respond to appointment requests. Pet owners can add multiple pets with complete information and book appointments based on available slots. Payment is required during booking to ensure certainty. Veterinarians can view full pet profiles before confirming appointments. After the appointment, pet owners can leave a review, which appears on the veterinarian's user profile. Pet history is stored for better future care.

SF2: Blog Posts

Users can create posts that include one picture, allowing them to share information or experiences. They can comment, like, and reply to comments on posts, and posts can be sorted by categories for easier browsing and discovery. Additionally, users have the ability to update or delete their own posts, giving them full control over their content.

SF3: Real-Time Chat System

The system supports instant messaging between users, enabling real-time communication. It facilitates discussions on topics such as pet health, events, and other related matters. Messages have read and unread statuses to help users track their conversations easily.

SP4: Admin Dashboard

The system monitors and moderates all user-generated content to maintain a safe and credible platform. Admins approve veterinarian registrations, review user profiles, and organize content into categories for better management. Additionally, they can view platform statistics and user activity through charts and reports.

1.2.2 User class and characteristics

i. Pet Owner

A pet owner is a registered user who can fully interact with the platform's features. Pet owners are allowed to create and manage their profiles, add multiple pets with complete medical history, book appointments with veterinarians, and communicate through the messaging system. They can also engage in community activities such as posting, commenting, liking, and reviewing veterinarians after completed appointments. This user class benefits from the platform's full functionality, helping them maintain their pets' health while building a community with other pet enthusiasts.

ii. Veterinarians

A veterinarian is a specialized user who provides professional services through the platform. Vets are required to complete a profile and undergo admin approval before gaining full access. They can manage their appointment schedules, review detailed pet information before consultations, and participate in community discussions to share expertise. This user class represents the service providers of the platform, offering credible healthcare advice and appointment services to pet owners while maintaining professional engagement.

iii. Administrators

An administrator is a privileged user responsible for overseeing platform operations and maintaining system integrity. Admins monitor and moderate user-generated content, approve veterinarian registrations, manage platform categories, and review user activities. They have access to detailed platform statistics and charts, enabling them to make informed decisions about platform growth, user behavior, and compliance. This user class ensures that the platform remains credible, organized, and user-friendly for all participants.

1.2.3 Operating Environment

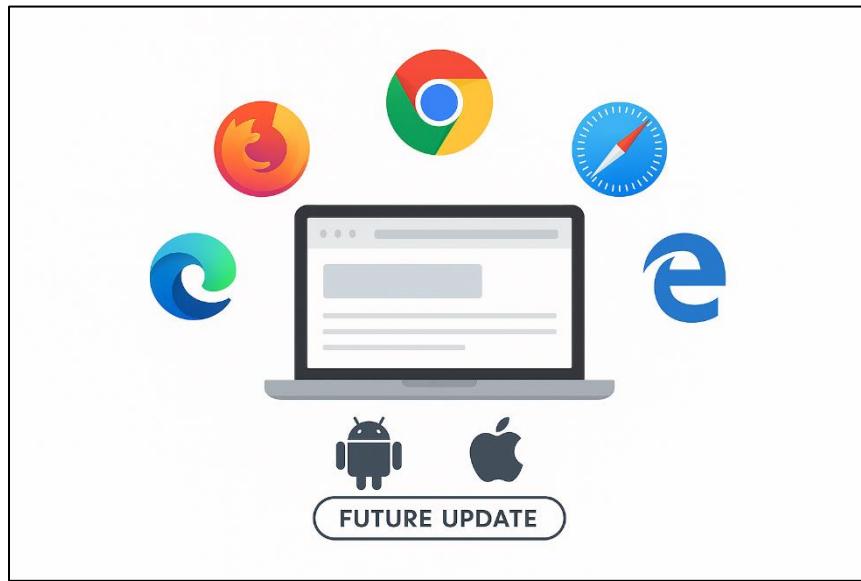


Figure 7: Operating Environment

OE1: The application is developed as a web based platform that works with the modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. Future updates will improve its functionality on the mobile platforms with adaptive web design or with dedicated native applications for Android and iOS devices.

1.2.4 Design and Implementation Constraints

- CO1:** The system must implement real-time features, particularly for the instant messaging (chat) functionality, ensuring smooth and immediate communication between users.
- CO2:** Pet owners must complete payment at the time of booking to confirm an appointment. Booking without payment is not permitted to maintain appointment certainty.
- CO3:** Older devices or browsers that do not support modern web technologies (such as WebSockets for real-time chat) may experience reduced functionality or be unable to use certain features of the application

1.2.5 Assumptions and Dependencies

- AS1:** Stable internet connectivity is essential for functionalities such as real-time chat, appointment bookings, and payment processing.
- AS2:** The system assumes that veterinarians will regularly update and maintain their appointment schedules to prevent booking conflicts.
- AS3:** Successful appointment confirmations depend on the availability and proper functioning of integrated payment systems.
- AS4:** Platform credibility and user safety rely on active administrative moderation of blogs, reviews

1.3 Functional Requirements

1.3.1 Registration

Req.ID	Requirement Description		Priority	Complexity
FR.01	Pet Owners can register by filling General Information, completing a detailed profile, verifying email through OTP, and then accessing the Community Page.		Most	High
System Requirement				
	SR.01	Pet Owners are required to select "Pet Owner" in the user type field during General Information registration.		
	SR.02	After submitting General Information, Pet Owners are redirected to a Complete Profile page to enter full details.		
	SR.03	An OTP is sent to the provided email address upon profile submission.		
	SR.04	Pet Owners must verify their account by entering the OTP on the verification page.		
	SR.05	After successful OTP verification, Pet Owners are redirected to the Community Page.		

FR.02	Veterinarians can register by filling General Information, completing a detailed profile and awaiting Admin Approval.	Most	High
System Requirement			
SR.06			Veterinarians are required to select "Veterinarian" in the user type field during General Information registration.
SR.07			After submitting General Information, Veterinarians are redirected to a Complete Profile page tailored for veterinary details.
SR.08			Veterinarians wait for Admin review and approval.
SR.09			If Admin declines the veterinarian's profile, the Veterinarian can edit the Complete Profile and resubmit for approval.
SR.10			Upon Admin approval, veterinarians are redirected to the Community Page.

Table 3 Functional Requirement: Registration:

1.3.2 Login

Req.ID	Requirement Description		Priority	Complexity										
FR.03	<p>Users (Pet Owners and Veterinarians) can access the login page, fill in their email and password, and upon successful authentication, are redirected to the Community Page.</p> <table border="1" data-bbox="349 599 1054 1326"> <thead> <tr> <th colspan="2" data-bbox="349 599 1054 671">System Requirement</th></tr> </thead> <tbody> <tr> <td data-bbox="349 671 535 811">SR.11</td><td data-bbox="535 671 1054 811">Users must access the Login Page from the application interface.</td></tr> <tr> <td data-bbox="349 811 535 950">SR.12</td><td data-bbox="535 811 1054 950">Users are required to input a valid email and password.</td></tr> <tr> <td data-bbox="349 950 535 1127">SR.13</td><td data-bbox="535 950 1054 1127">Upon successful authentication, users are redirected to the Community Page.</td></tr> <tr> <td data-bbox="349 1127 535 1326">SR.14</td><td data-bbox="535 1127 1054 1326">If credentials are invalid, an appropriate error message is displayed.</td></tr> </tbody> </table>		System Requirement		SR.11	Users must access the Login Page from the application interface.	SR.12	Users are required to input a valid email and password.	SR.13	Upon successful authentication, users are redirected to the Community Page.	SR.14	If credentials are invalid, an appropriate error message is displayed.	Most	Medium
System Requirement														
SR.11	Users must access the Login Page from the application interface.													
SR.12	Users are required to input a valid email and password.													
SR.13	Upon successful authentication, users are redirected to the Community Page.													
SR.14	If credentials are invalid, an appropriate error message is displayed.													
FR.04	<p>Administrators can access the Admin Login Page, fill in their email and password, and upon successful authentication, are redirected to the Admin Dashboard.</p> <table border="1" data-bbox="349 1571 1054 1761"> <thead> <tr> <th colspan="2" data-bbox="349 1571 1054 1643">System Requirement</th></tr> </thead> <tbody> <tr> <td data-bbox="349 1643 535 1761">SR.15</td><td data-bbox="535 1643 1054 1761">Admins must access the dedicated Admin Login Page.</td></tr> </tbody> </table>		System Requirement		SR.15	Admins must access the dedicated Admin Login Page.	Most	Medium						
System Requirement														
SR.15	Admins must access the dedicated Admin Login Page.													

	SR.16	Admins are required to input a valid email and password.	
	SR.17	Upon successful authentication, admins are redirected to the Admin Dashboard.	
	SR.18	If credentials are invalid, an appropriate error message is displayed.	

Table 4 Functional Requirement: Login

1.3.3 Appointment

Req.ID	Requirement Description		Priority	Complexity														
FR.05	<p>Pet Owners can search Veterinarians, book appointments, make payments, and leave reviews after completion.</p> <table border="1" data-bbox="344 551 1054 1638"> <thead> <tr> <th colspan="2" data-bbox="344 551 1054 614">System Requirement</th></tr> </thead> <tbody> <tr> <td data-bbox="344 614 523 751">SR.19</td><td data-bbox="523 614 1054 751">Pet Owners can search and filter Veterinarians</td></tr> <tr> <td data-bbox="344 751 523 889">SR.20</td><td data-bbox="523 751 1054 889">Pet Owners can view available time slots for each Veterinarian.</td></tr> <tr> <td data-bbox="344 889 523 1068">SR.21</td><td data-bbox="523 889 1054 1068">Pet Owners must provide a visit reason and select a pet from their pets.</td></tr> <tr> <td data-bbox="344 1068 523 1248">SR.22</td><td data-bbox="523 1068 1054 1248">Payment for the appointment must be made via Store Credit or Khalti at the time of booking.</td></tr> <tr> <td data-bbox="344 1248 523 1427">SR.23</td><td data-bbox="523 1248 1054 1427">Pet Owners are notified of appointment acceptance or rejection by Veterinarian.</td></tr> <tr> <td data-bbox="344 1427 523 1638">SR.24</td><td data-bbox="523 1427 1054 1638">After appointment completion, Pet Owners can submit a review of the Veterinarian.</td></tr> </tbody> </table>	System Requirement		SR.19	Pet Owners can search and filter Veterinarians	SR.20	Pet Owners can view available time slots for each Veterinarian.	SR.21	Pet Owners must provide a visit reason and select a pet from their pets.	SR.22	Payment for the appointment must be made via Store Credit or Khalti at the time of booking.	SR.23	Pet Owners are notified of appointment acceptance or rejection by Veterinarian.	SR.24	After appointment completion, Pet Owners can submit a review of the Veterinarian.	Most		High
System Requirement																		
SR.19	Pet Owners can search and filter Veterinarians																	
SR.20	Pet Owners can view available time slots for each Veterinarian.																	
SR.21	Pet Owners must provide a visit reason and select a pet from their pets.																	
SR.22	Payment for the appointment must be made via Store Credit or Khalti at the time of booking.																	
SR.23	Pet Owners are notified of appointment acceptance or rejection by Veterinarian.																	
SR.24	After appointment completion, Pet Owners can submit a review of the Veterinarian.																	

FR.06	Veterinarians can set availability, manage appointment requests, and mark appointments as completed or declined with refunds	Most	High
System Requirement			
SR.25 Veterinarians can create and manage their appointment schedules.			
SR.26 Veterinarians receive email notifications for appointment requests.			
SR.27 Veterinarians can accept or decline appointment requests.			
SR.28 Upon accepting, the appointment is scheduled and shown in the Veterinarian's schedule.			
SR.29 After completing the real-life appointment, Veterinarians can mark it as completed.			
SR.30 If an appointment is declined, Store Credit refund is automatically issued to the Pet Owner.			

Table 5 Functional Requirement: Appointment

1.3.4 Blog Posts

Req.ID	Requirement Description		Priority	Complexity														
FR.07	Pet Owners can search Veterinarians, book appointments, make payments, and leave reviews after completion. <table border="1" data-bbox="347 551 1054 1326"> <thead> <tr> <th colspan="2" data-bbox="347 551 1054 604">System Requirement</th></tr> </thead> <tbody> <tr> <td data-bbox="347 614 535 741">SR.31</td><td data-bbox="535 614 1054 741">Users can create a post with title, category, body, and one photo.</td></tr> <tr> <td data-bbox="347 751 535 920">SR.32</td><td data-bbox="535 751 1054 920">After successful post creation, users are redirected to the post detail page.</td></tr> <tr> <td data-bbox="347 931 535 984">SR.33</td><td data-bbox="535 931 1054 984">Users can like any post.</td></tr> <tr> <td data-bbox="347 994 535 1121">SR.34</td><td data-bbox="535 994 1054 1121">Users can comment on posts and reply to comments.</td></tr> <tr> <td data-bbox="347 1132 535 1258">SR.35</td><td data-bbox="535 1132 1054 1258">Users can delete their own comments and replies.</td></tr> <tr> <td data-bbox="347 1269 535 1396">SR.36</td><td data-bbox="535 1269 1054 1396">Users can edit and delete their own posts.</td></tr> </tbody> </table>		System Requirement		SR.31	Users can create a post with title, category, body, and one photo.	SR.32	After successful post creation, users are redirected to the post detail page.	SR.33	Users can like any post.	SR.34	Users can comment on posts and reply to comments.	SR.35	Users can delete their own comments and replies.	SR.36	Users can edit and delete their own posts.	Most	High
System Requirement																		
SR.31	Users can create a post with title, category, body, and one photo.																	
SR.32	After successful post creation, users are redirected to the post detail page.																	
SR.33	Users can like any post.																	
SR.34	Users can comment on posts and reply to comments.																	
SR.35	Users can delete their own comments and replies.																	
SR.36	Users can edit and delete their own posts.																	

Table 6 Functional Requirement: Blog Posts

1.3.5 Real Time Chat System

Req.ID	Requirement Description	Priority	Complexity														
FR.08	<p>Users can open a chat from the Inbox or a user's profile by clicking "Message." Messages are sent instantly with real-time delivery status (single tick for sent, double tick for read).</p> <table border="1" data-bbox="349 599 1054 1824"> <thead> <tr> <th colspan="2" data-bbox="349 599 1054 667">System Requirement</th></tr> </thead> <tbody> <tr> <td data-bbox="349 667 535 874">SR.37</td><td data-bbox="535 667 1054 874">Users can view a list of existing chats (Inbox) or initiate a new chat from another user's profile.</td></tr> <tr> <td data-bbox="349 874 535 1081">SR.38</td><td data-bbox="535 874 1054 1081">Clicking the "Message" button instantly opens a chat window with the selected user.</td></tr> <tr> <td data-bbox="349 1081 535 1241">SR.39</td><td data-bbox="535 1081 1054 1241">Messages are sent in real time without needing to refresh the page</td></tr> <tr> <td data-bbox="349 1241 535 1448">SR.40</td><td data-bbox="535 1241 1054 1448">Each message shows a single tick (✓) once sent and a double tick (✓✓) once read.</td></tr> <tr> <td data-bbox="349 1448 535 1634">SR.41</td><td data-bbox="535 1448 1054 1634">The chat window supports instant loading of message history when opened.</td></tr> <tr> <td data-bbox="349 1634 535 1824">SR.42</td><td data-bbox="535 1634 1054 1824">Users receive real-time notifications when new messages arrive.</td></tr> </tbody> </table>	System Requirement		SR.37	Users can view a list of existing chats (Inbox) or initiate a new chat from another user's profile.	SR.38	Clicking the "Message" button instantly opens a chat window with the selected user.	SR.39	Messages are sent in real time without needing to refresh the page	SR.40	Each message shows a single tick (✓) once sent and a double tick (✓✓) once read.	SR.41	The chat window supports instant loading of message history when opened.	SR.42	Users receive real-time notifications when new messages arrive.	Most	High
System Requirement																	
SR.37	Users can view a list of existing chats (Inbox) or initiate a new chat from another user's profile.																
SR.38	Clicking the "Message" button instantly opens a chat window with the selected user.																
SR.39	Messages are sent in real time without needing to refresh the page																
SR.40	Each message shows a single tick (✓) once sent and a double tick (✓✓) once read.																
SR.41	The chat window supports instant loading of message history when opened.																
SR.42	Users receive real-time notifications when new messages arrive.																

Table 7 Functional Requirement: Real Time Chat

1.3.6 Admin Dashboard

Req.ID	Requirement Description	Priority	Complexity												
FR.09	<p>Admins can log in to view dashboard stats, review user activity, manage posts and categories, and approve Veterinarian applications.</p> <table border="1" data-bbox="347 608 1054 1459"> <thead> <tr> <th colspan="2" data-bbox="347 608 1054 661">System Requirement</th></tr> </thead> <tbody> <tr> <td data-bbox="347 661 535 868">SR.43</td><td data-bbox="535 661 1054 868">Admins can view charts and statistics about users, posts, and categories on the dashboard.</td></tr> <tr> <td data-bbox="347 868 535 1051">SR.44</td><td data-bbox="535 868 1054 1051">Admins can view detailed user profiles, including their comment, post, and reply history.</td></tr> <tr> <td data-bbox="347 1051 535 1184">SR.45</td><td data-bbox="535 1051 1054 1184">Admins can delete or deactivate any user post.</td></tr> <tr> <td data-bbox="347 1184 535 1317">SR.46</td><td data-bbox="535 1184 1054 1317">Admins can create, edit, and delete categories.</td></tr> <tr> <td data-bbox="347 1317 535 1459">SR.41</td><td data-bbox="535 1317 1054 1459">Admins can review and accept Veterinarian applications.</td></tr> </tbody> </table>	System Requirement		SR.43	Admins can view charts and statistics about users, posts, and categories on the dashboard.	SR.44	Admins can view detailed user profiles, including their comment, post, and reply history.	SR.45	Admins can delete or deactivate any user post.	SR.46	Admins can create, edit, and delete categories.	SR.41	Admins can review and accept Veterinarian applications.	Most	High
System Requirement															
SR.43	Admins can view charts and statistics about users, posts, and categories on the dashboard.														
SR.44	Admins can view detailed user profiles, including their comment, post, and reply history.														
SR.45	Admins can delete or deactivate any user post.														
SR.46	Admins can create, edit, and delete categories.														
SR.41	Admins can review and accept Veterinarian applications.														

Table 8 Functional Requirement: Admin Dashboard

1.4 External Interfaces Requirements

1.4.1 User Interfaces

The application will feature:

- A responsive design for various devices.
- Easy navigation for appointment management, blogging, and chat features.

1.4.2 Hardware Interfaces

The system requires:

- A stable internet connection of at least 2 Mbps for optimal real-time performance.
- A modern web browser (Chrome, Firefox, Safari)
- Minimum device specifications: 2 GB RAM, dual-core processor recommended.

1.4.3 Software Interfaces

Frontend	HTML, CSS3, Javascript
Backend	Django
Database	Postgres
External APIs	Khalti Payment Gateway API for handling payment transactions
External Libraries	Google Fonts (Poppins) for typography Font Awesome (v6.4.0) for icons.

Table 9: Software Interfaces

1.4.4 Communication Interfaces

The system uses the following communication interfaces:

- WebSocket protocols for real-time communication.
- The frontend communicates with the Django backend over HTTPS using AJAX for asynchronous data updates.
- Payments are processed through Khalti Payment Gateway API via secure HTTPS requests.

1.4.5 Other Non-Function Requirements

1.4.5.1 Performance Requirements

Req.ID	Requirement Description	Priority	Complexity
PR.01	The application must load without crashing when opened.	Should	High
PR.02	The application should load content within 3 seconds.	Could	Normal
PR.03	Messages should be sent and received within 2 seconds for real-time communication.	Should	High

Table 10: Performance Requirements

1.4.5.2 Safety and Security Requirements

Req.ID	Requirement Description	Priority	Complexity
SR.01	Only verified veterinarians can manage appointment slots.	Should	Medium
SR.02	The application must not install any malware or spyware in the background without user knowledge.	Should	Medium
SR.03	The application should prompt users to set a strong password with a minimum of 8 characters, including both upper and lower case letters, numbers, and special characters.	Should	High

Table 11: Safety and Security Requirements

1.4.5.3 Other Software Quality Attributes

Req.ID	Requirement Description	Priority	Complexity
SQA.01	The application should be clean and easy to operate, with an intuitive user interface.	Should	High
SQA.02	The frontend (HTML, CSS, JavaScript) should be responsive and adapt seamlessly to different screen sizes and devices (e.g., desktop, tablet, mobile).	Should	High

Table 12: Other Software Quality Attributes

3.6 Design

3.6.1 Initial Design

Before prototyping, it was necessary to do an initial design process, which included creating wireframes and a rough Figma UI draft. This early stage allowed us to quickly establish a basic structure for the application as well as the visual direction pointing at how the layout will look like, what can it hold, and how it can be navigated to different screens. The designs were raw and required refinement, however they were a fundamental block that made the scrutiny of usability, reordering of screens and shaping the entire project all a possibility. I also designed Entity Relationship Diagram (ERD), Class Diagram, Use Case Diagram and Data Flow Diagram (DFD) for the whole system in addition to UI/UX design.

3.6.1.1 System Logo



Figure 8: System Logo (PETVET)

3.6.1.2 Work Breakdown Structure

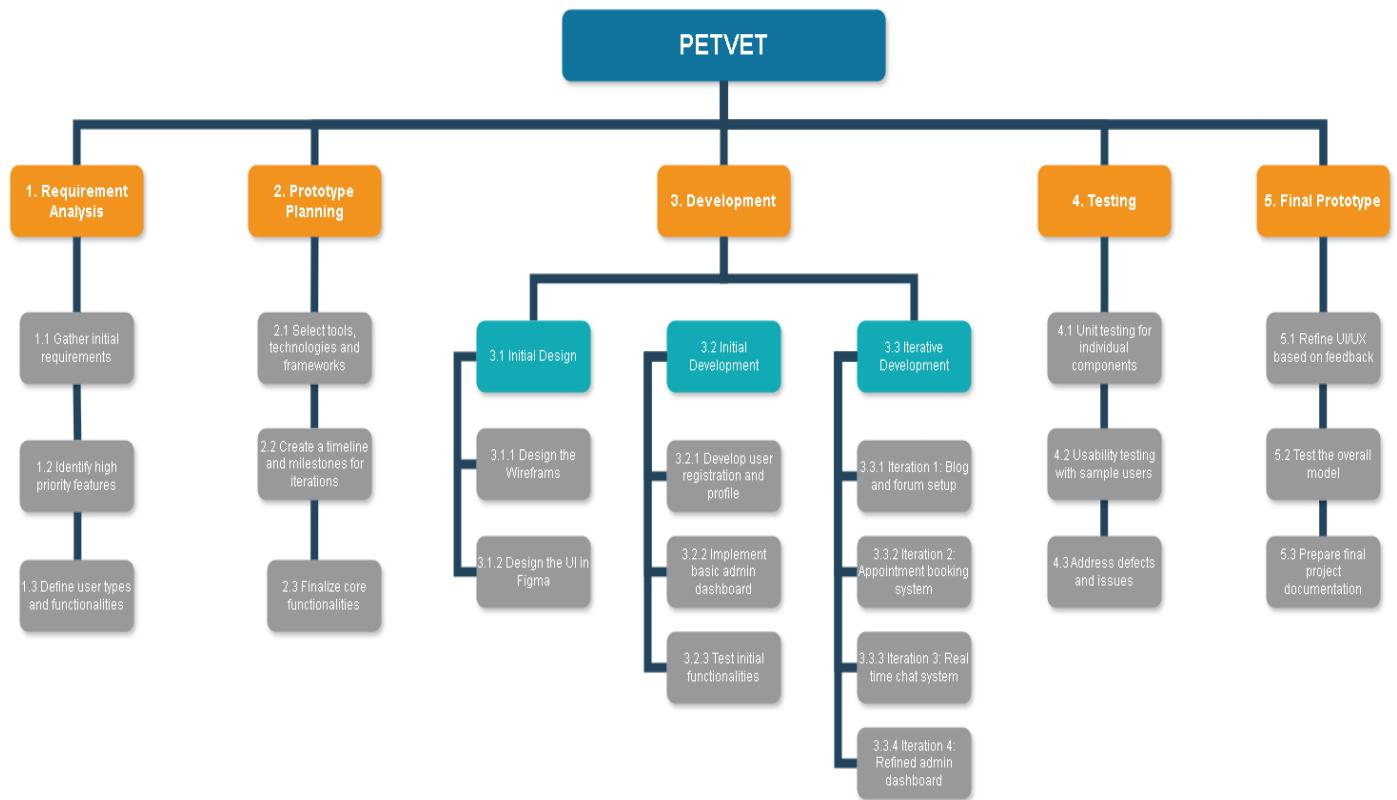


Figure 9: Final Work Breakdown Structure

Previous iteration of the Work breakdown structure: [7.4.2 Work Breakdown Structure](#)

3.6.1.3 Gantt Chart

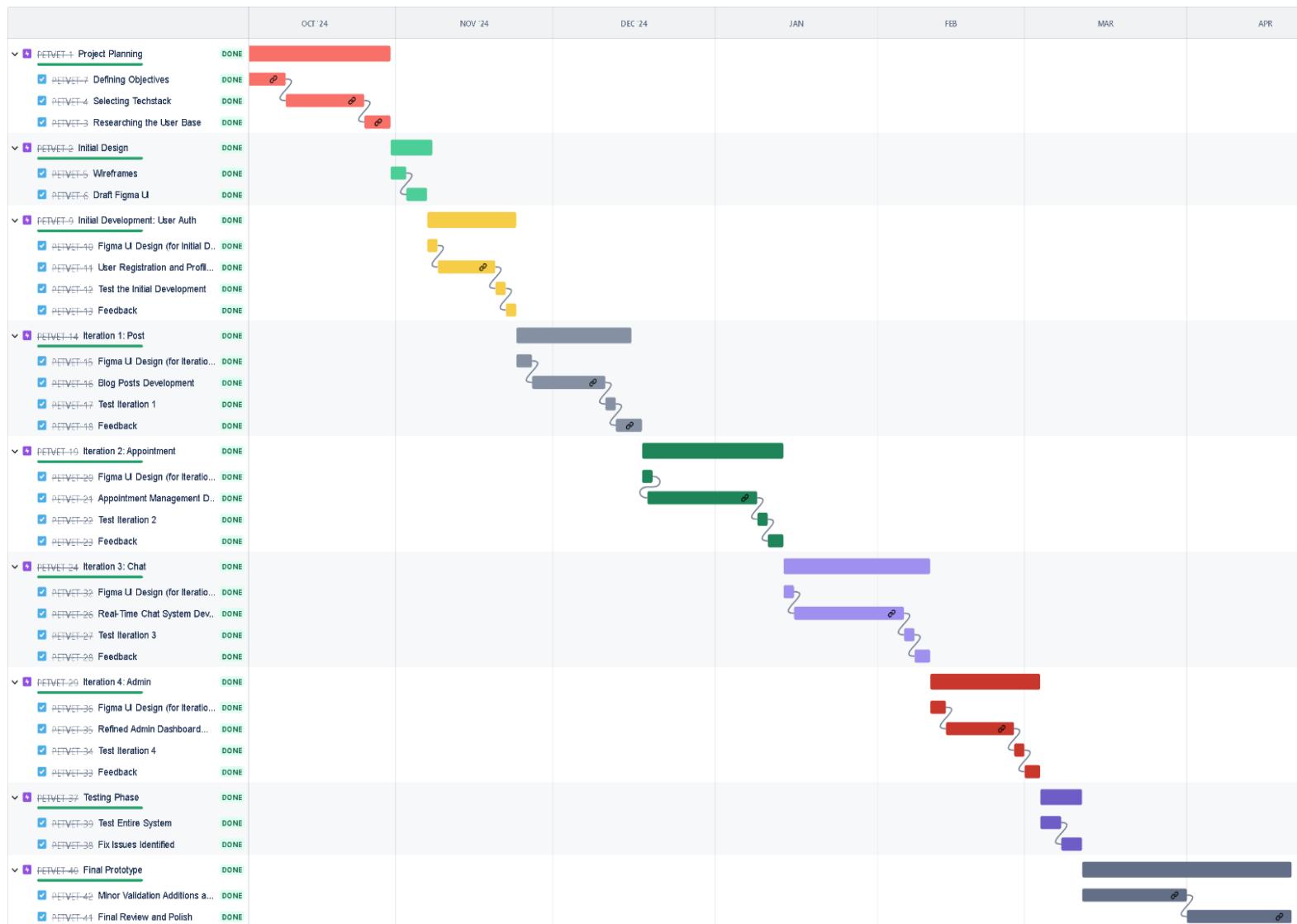


Figure 10: Final Gantt Chart

Previous iteration of the Gantt Chart: [7.4.1 Gantt Chart](#)

3.6.1.4 Entity Relationship Diagram

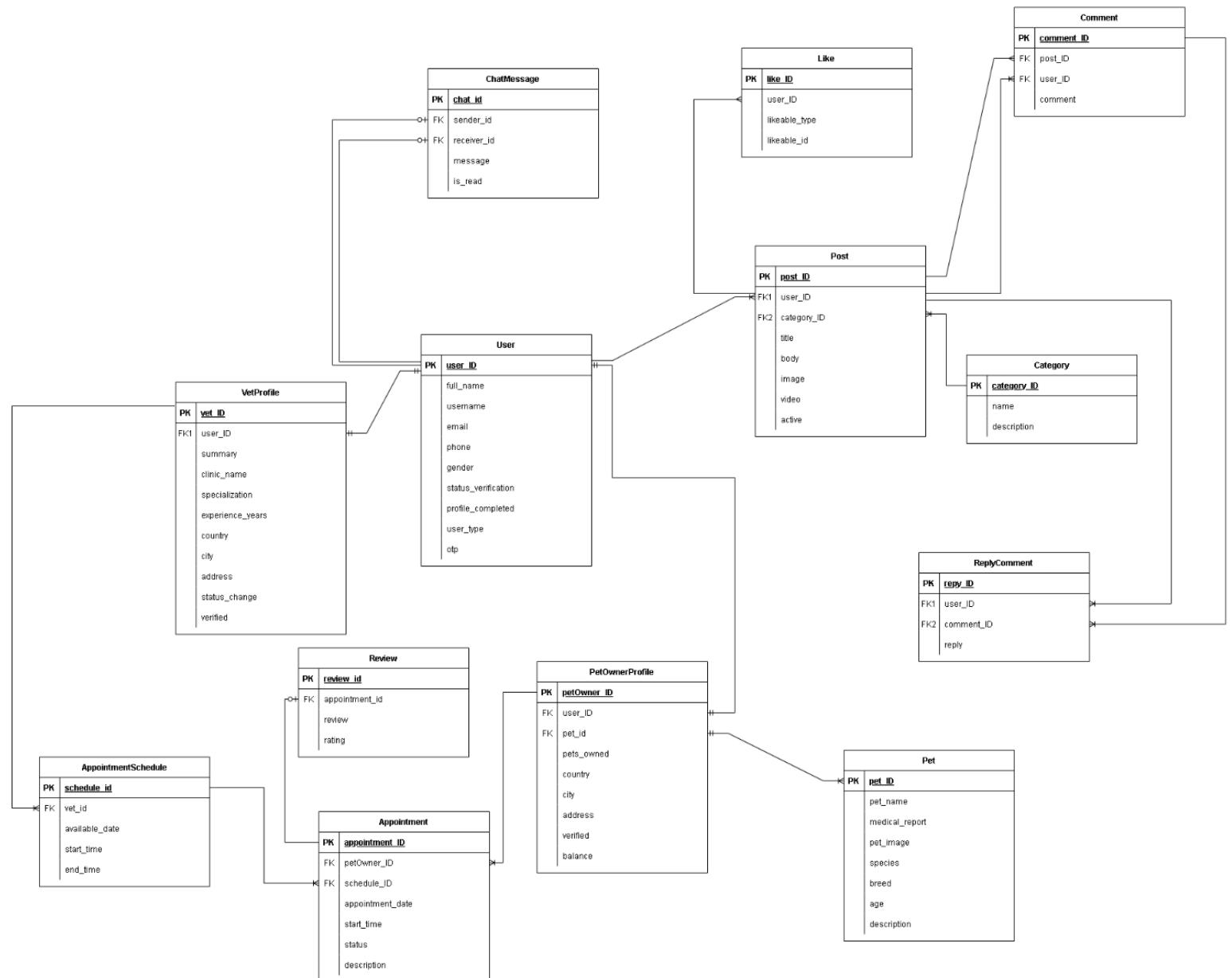


Figure 11: Final Entity Relationship Diagram

Previous iteration of the entity relationship diagram: [Entity Relation Diagram](#)

3.6.1.5 Class Diagram

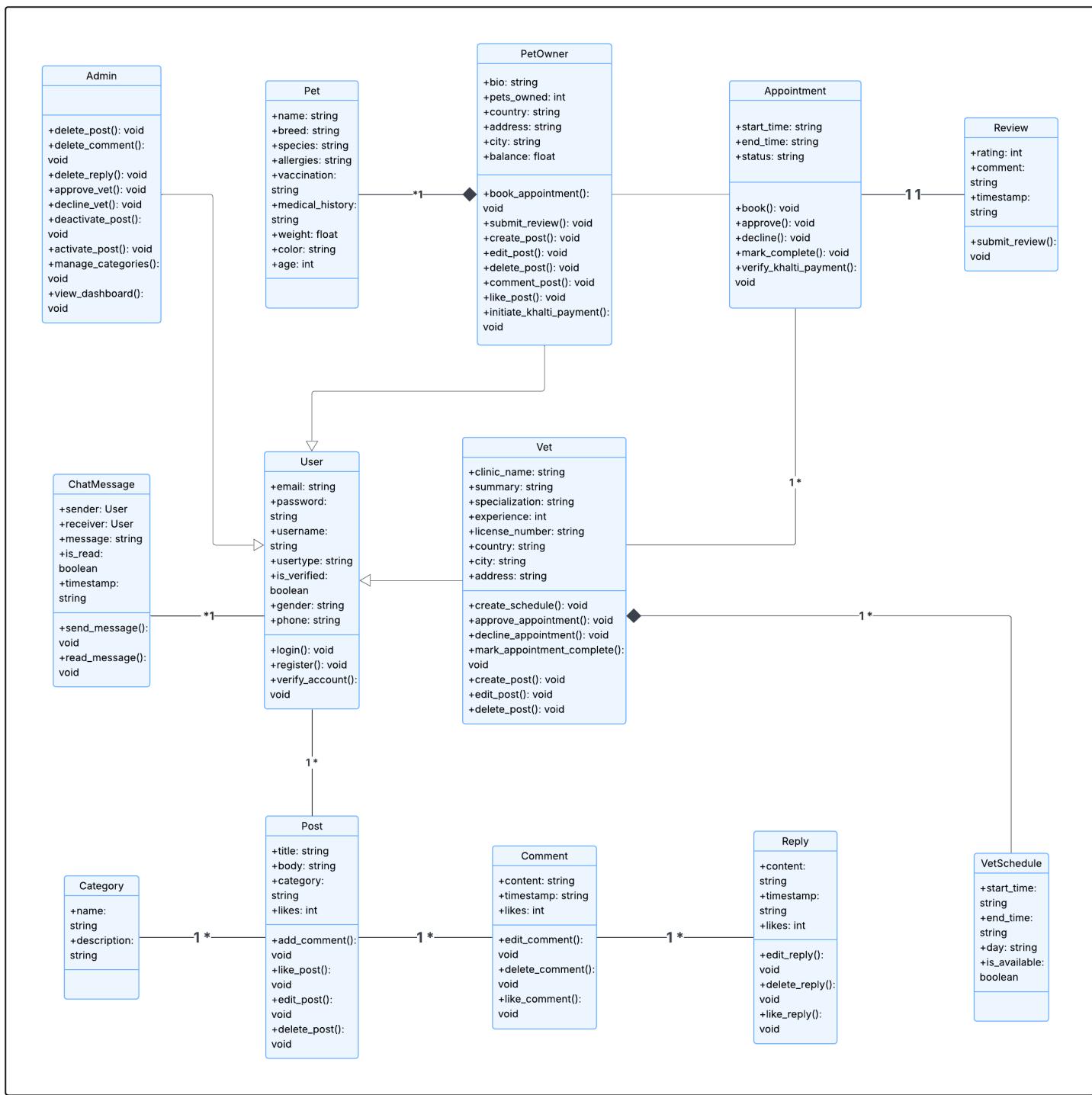


Figure 12: Final Class Diagram

Previous iteration of the class diagram: [7.4.9 Class Diagram](#)

3.6.1.6 Data Flow Diagram

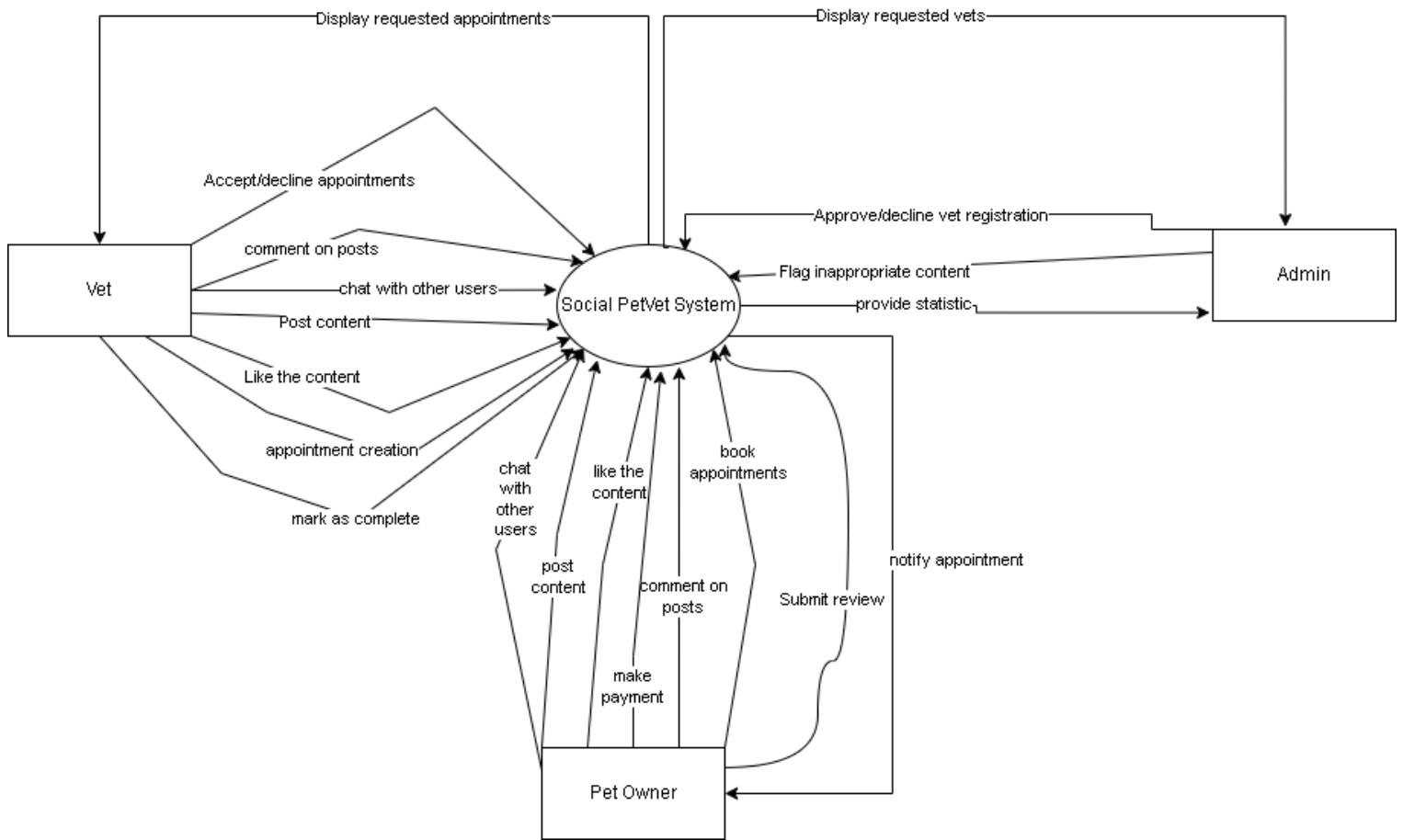


Figure 13: Final Data Flow Diagram

Previous iteration of data flow diagram: [Data Flow Diagram 0](#)

3.6.1.7 Usecase Diagram

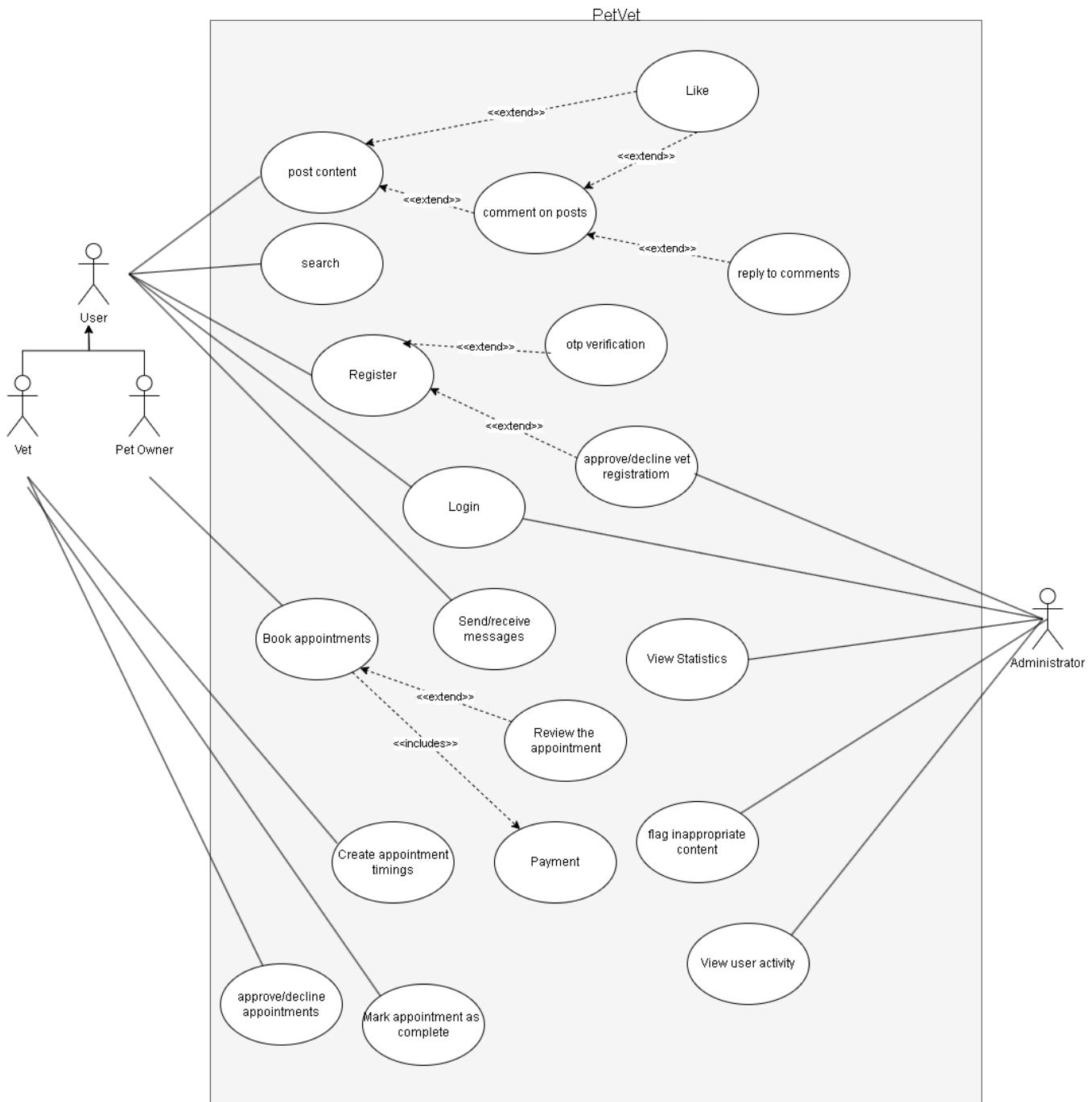


Figure 14: Final Usecase Diagram

Previous iteration of use case diagram: [7.4.5 Use Case](#)

3.6.1.1 Wireframes

Hand drawn blueprints of tangible wireframes are tangible wireframes that take abstract ideas and turn them into visible forms without the need of digital tools. Doing it on my own, working with wireframes, I could quickly see how the core screens would work, ideas could be refined without any technical distractions, and only have the focus on structure and flow (Sutipitakwong & Jamsri, 2020).

More interactivity and customization but also more time and more design experience, digital wireframes are created using Figma or Sketch.

Tangible wire frames worked in my early design phase: they were faster, more flexible, and helped solidify the vision before going on to digital prototyping.

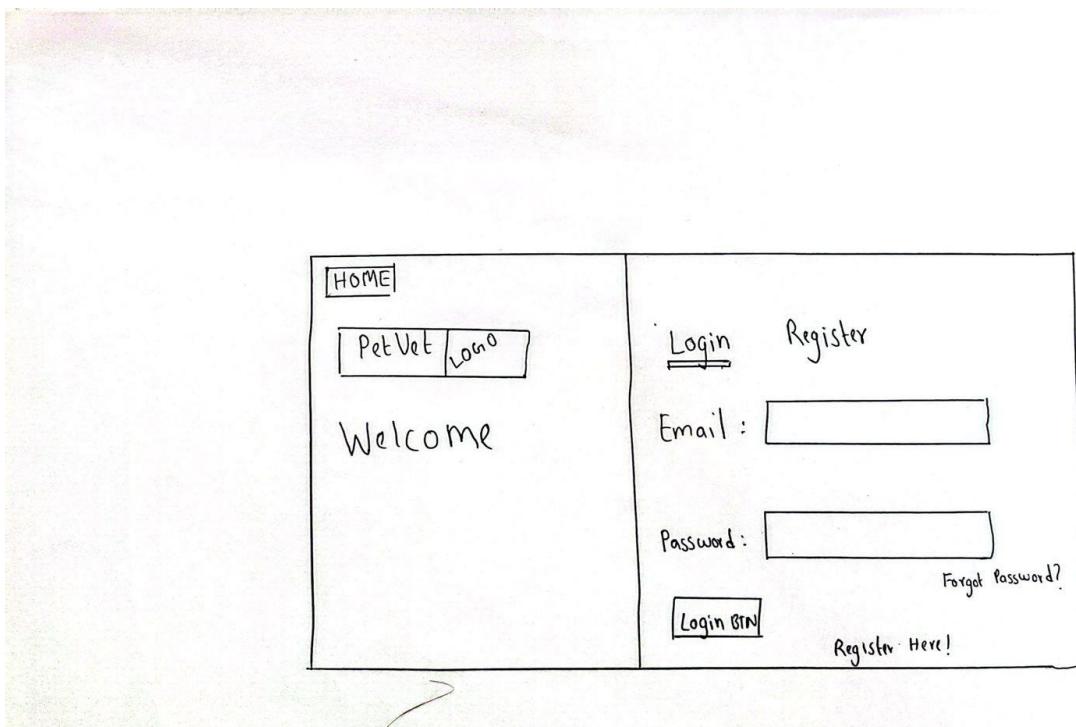


Figure 15: Wireframe for Login

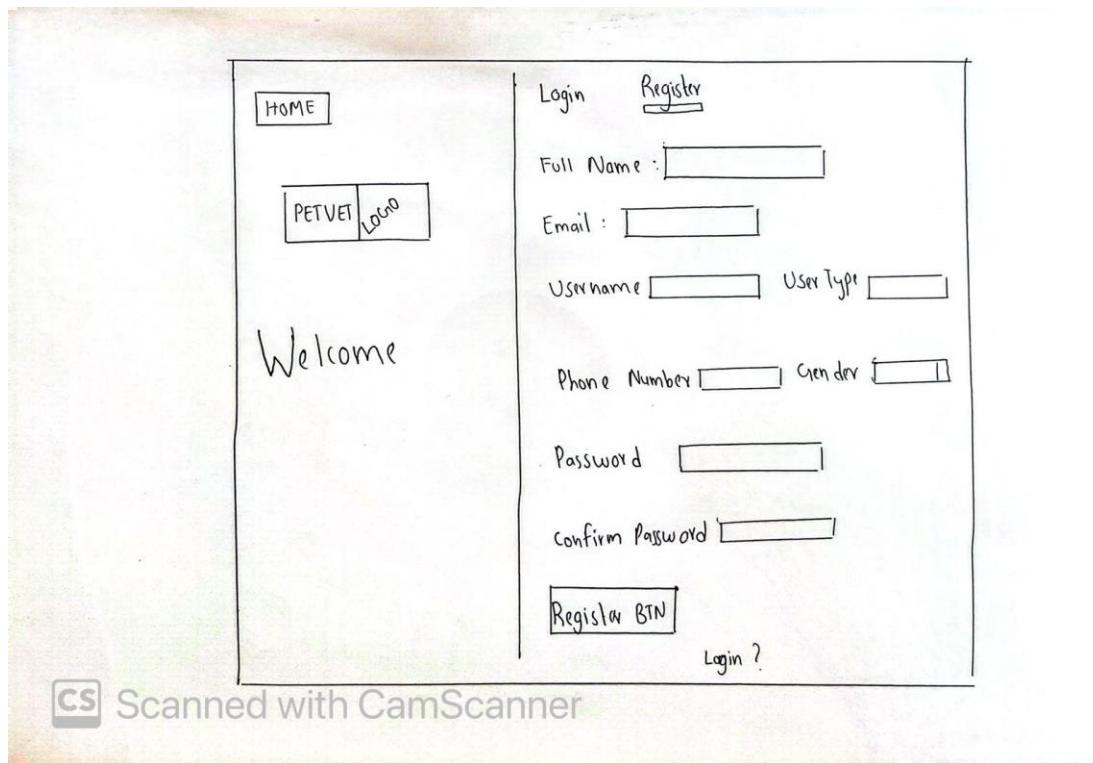


Figure 16: Wireframe for Registration

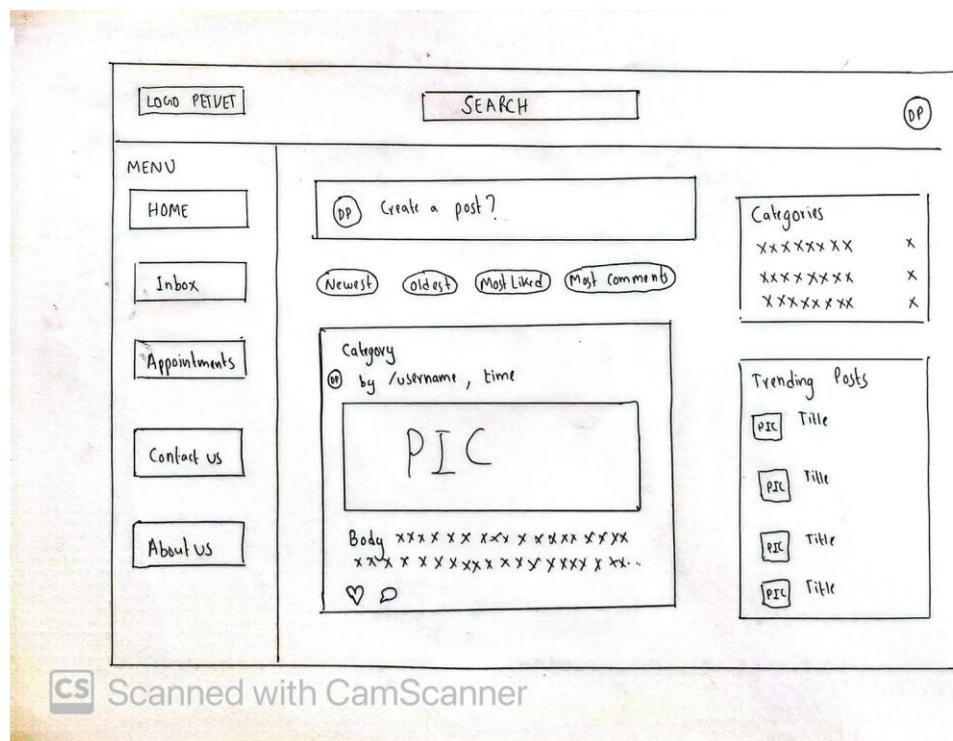


Figure 17: Wireframe for Home

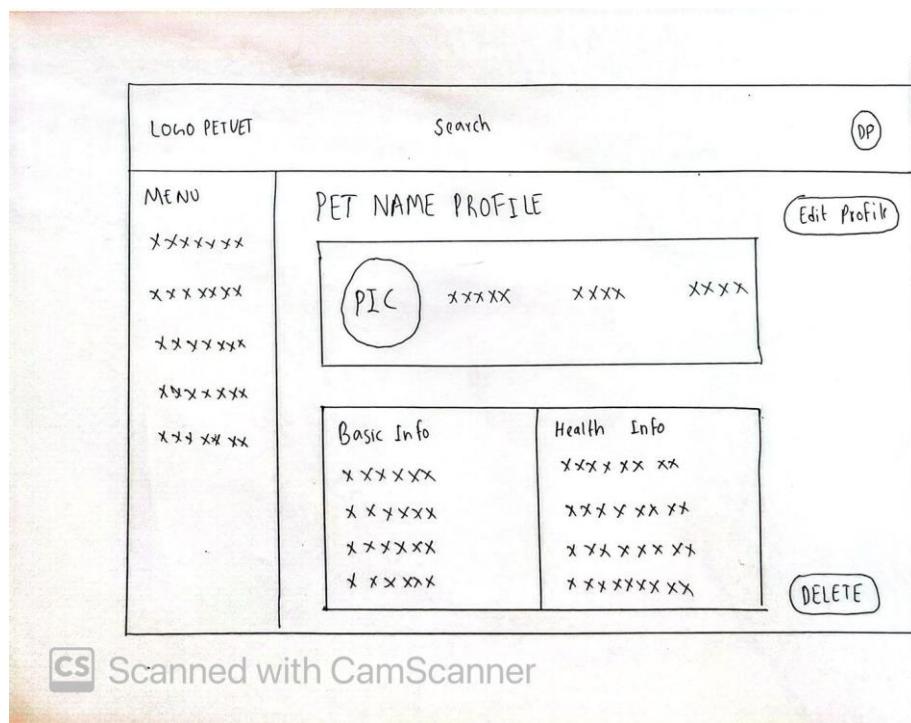


Figure 18: Wireframe for Pet profile

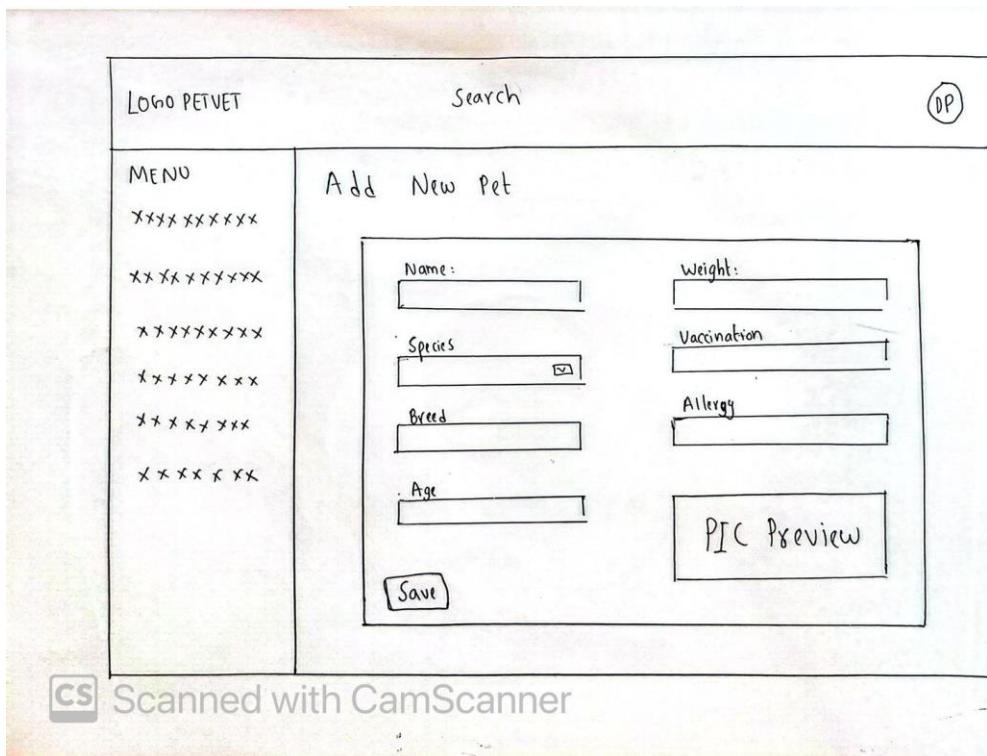


Figure 19: Wireframe for Add profile

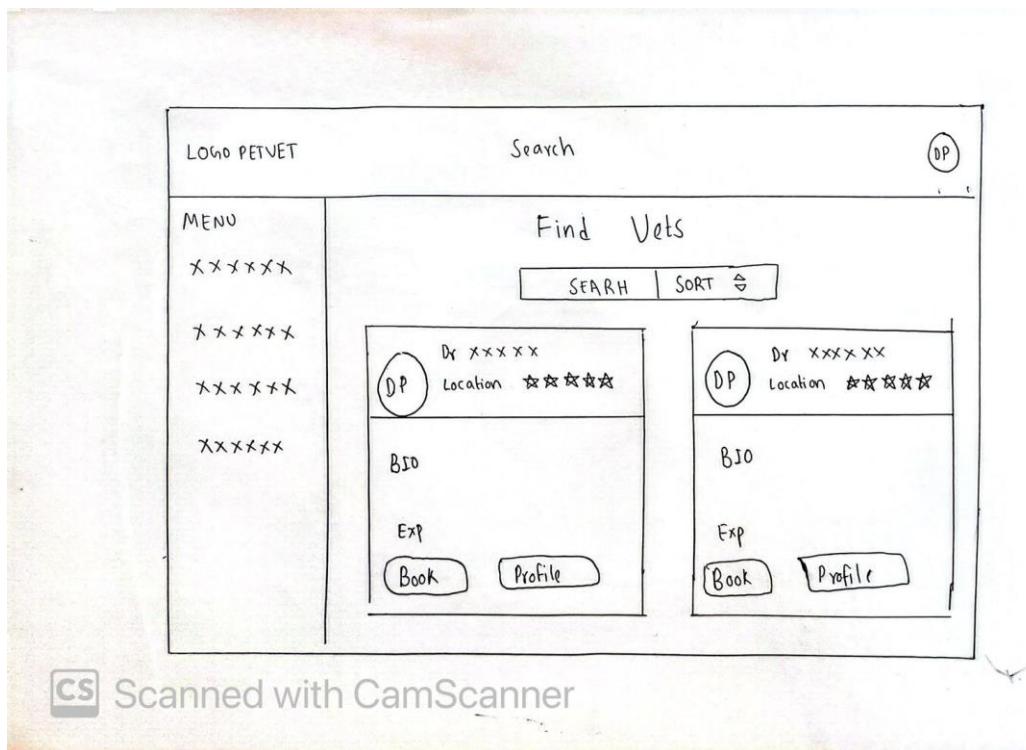


Figure 20: Wireframe for Find Vets

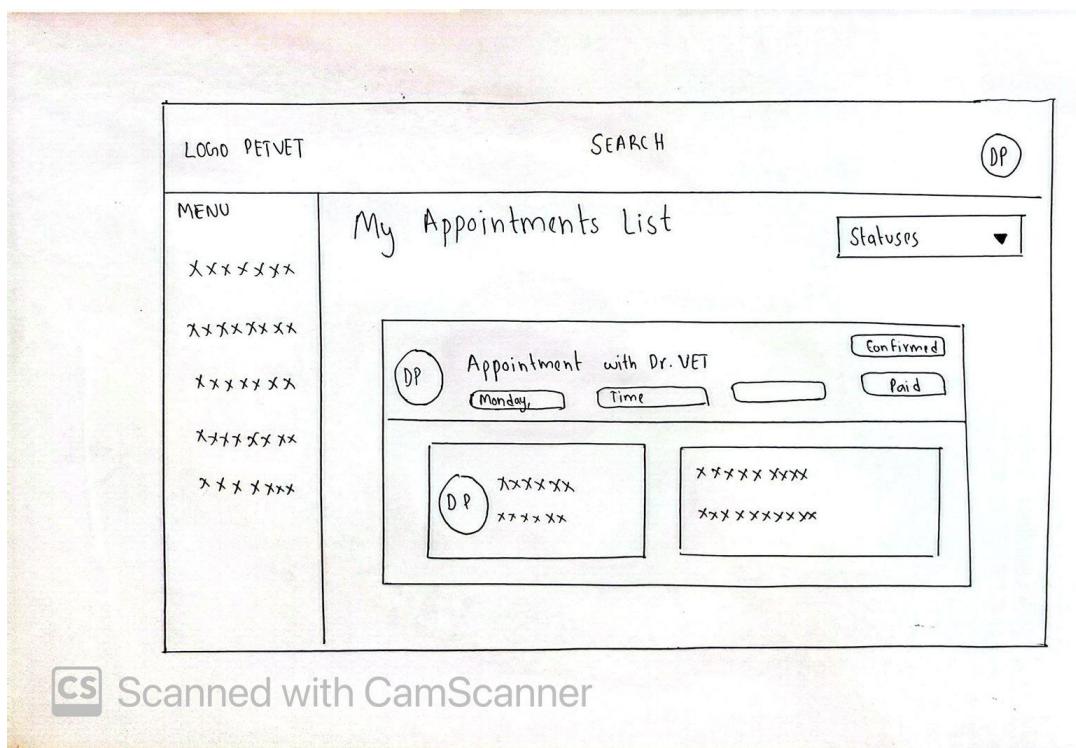


Figure 21: Wireframe for Appointment lists

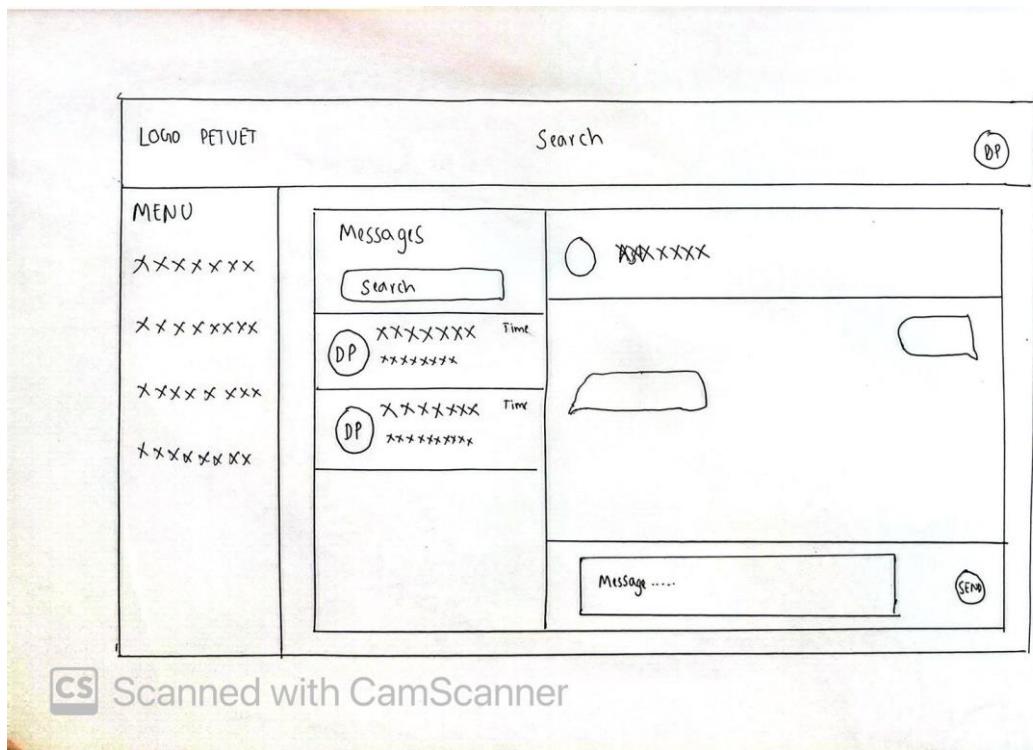


Figure 22: Wireframe for Inbox

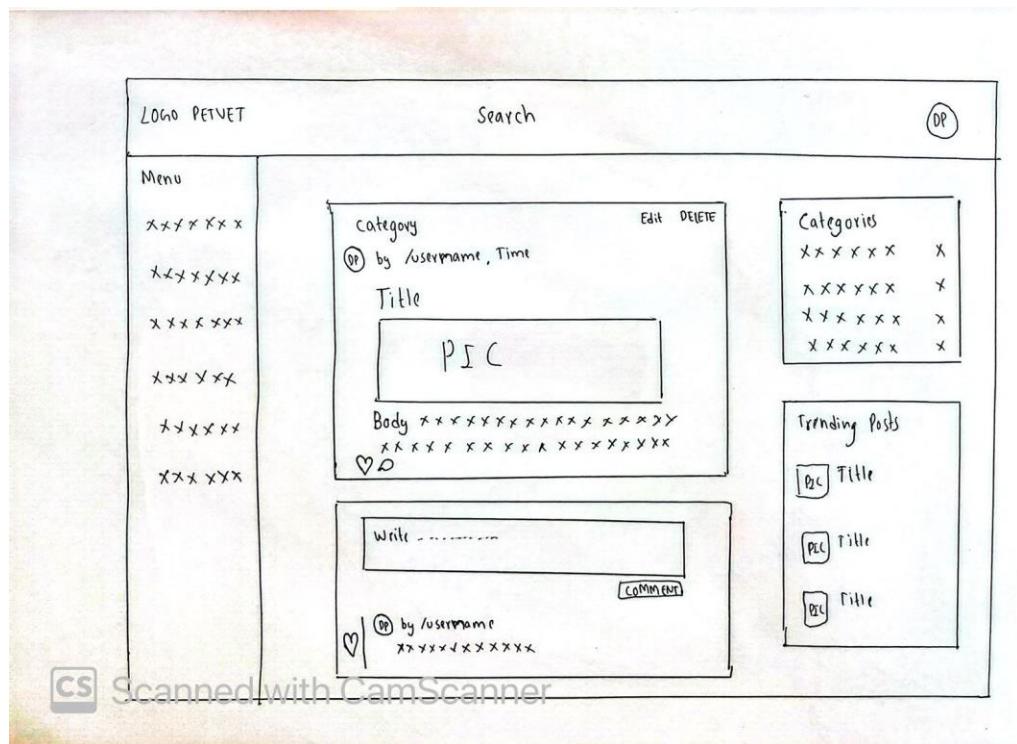


Figure 23: Wireframe for Post details

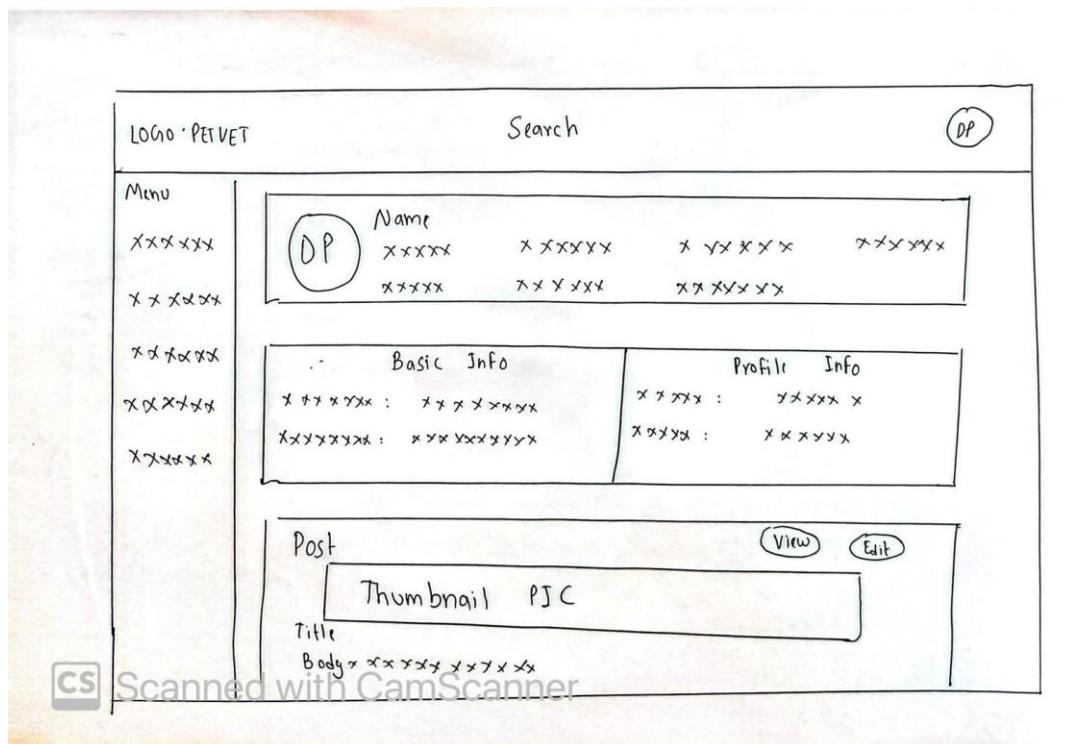


Figure 24: Wireframe for User profile

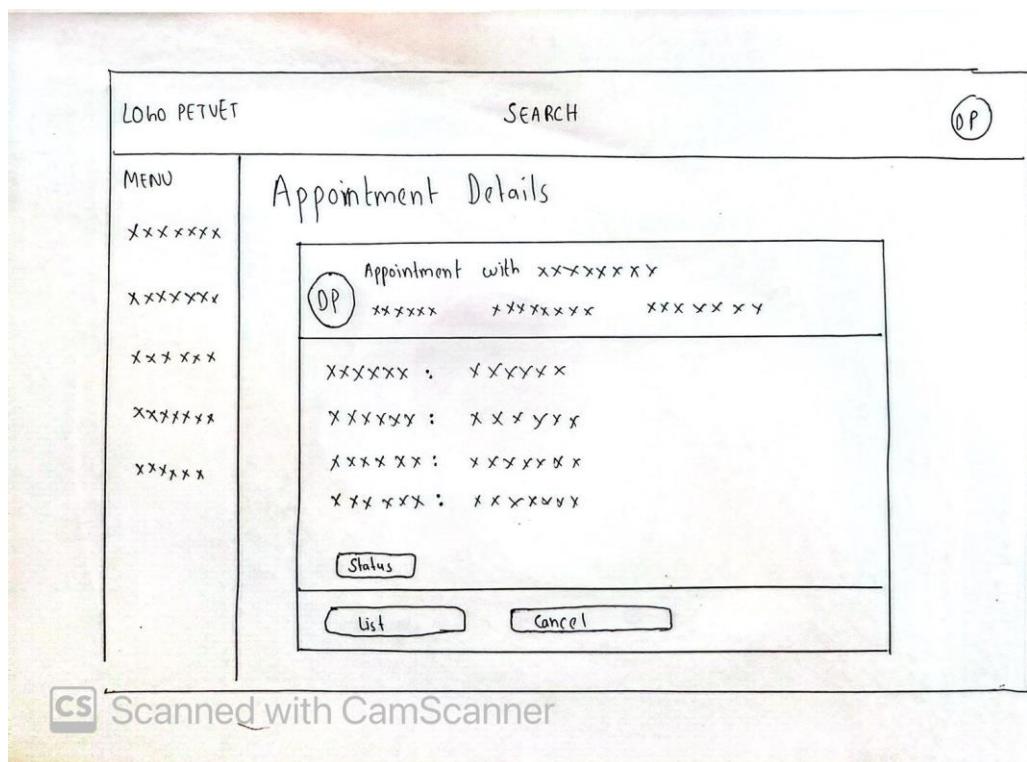


Figure 25: Wireframe for Appointment details

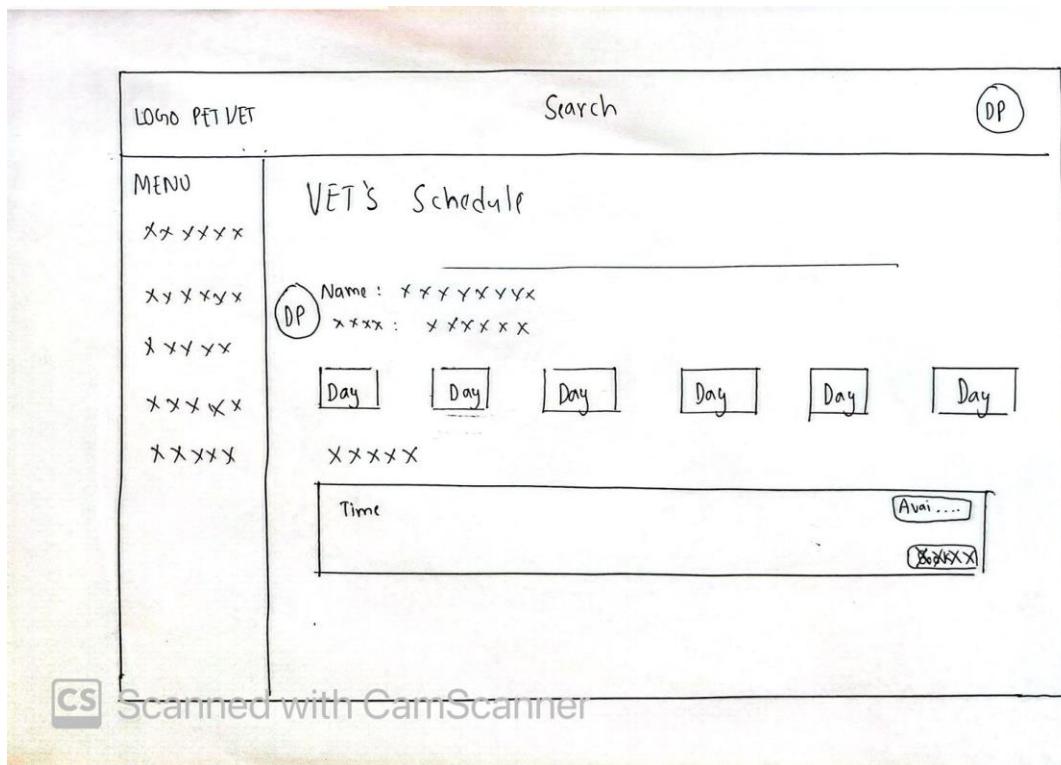


Figure 26: Wireframe for Vet schedule

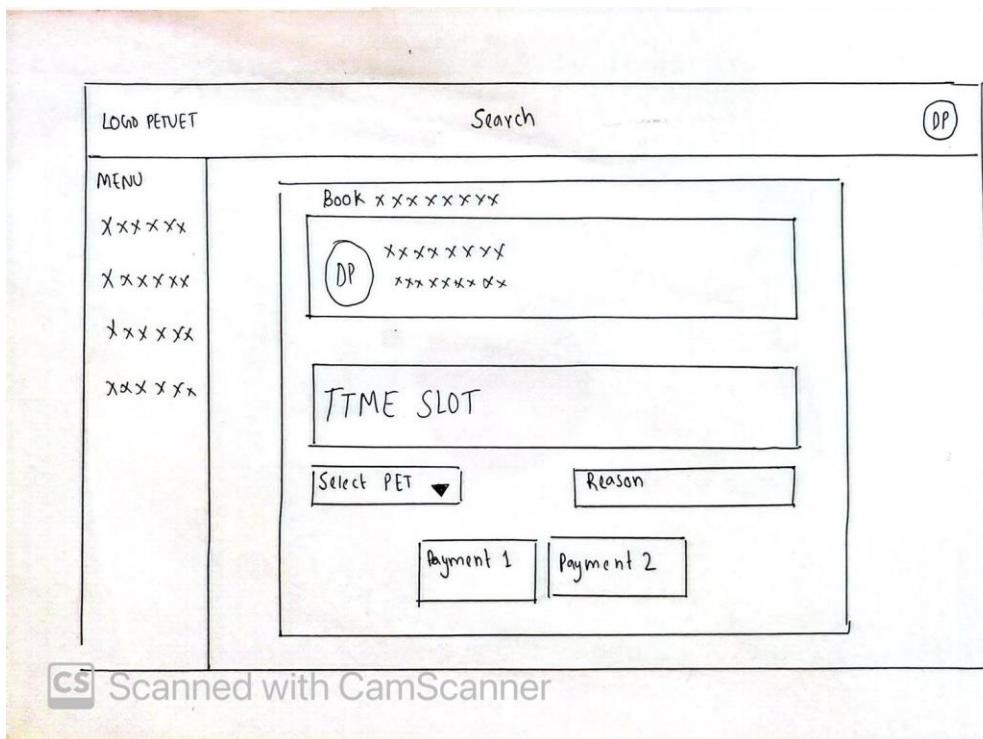


Figure 27: Wireframe for Booking

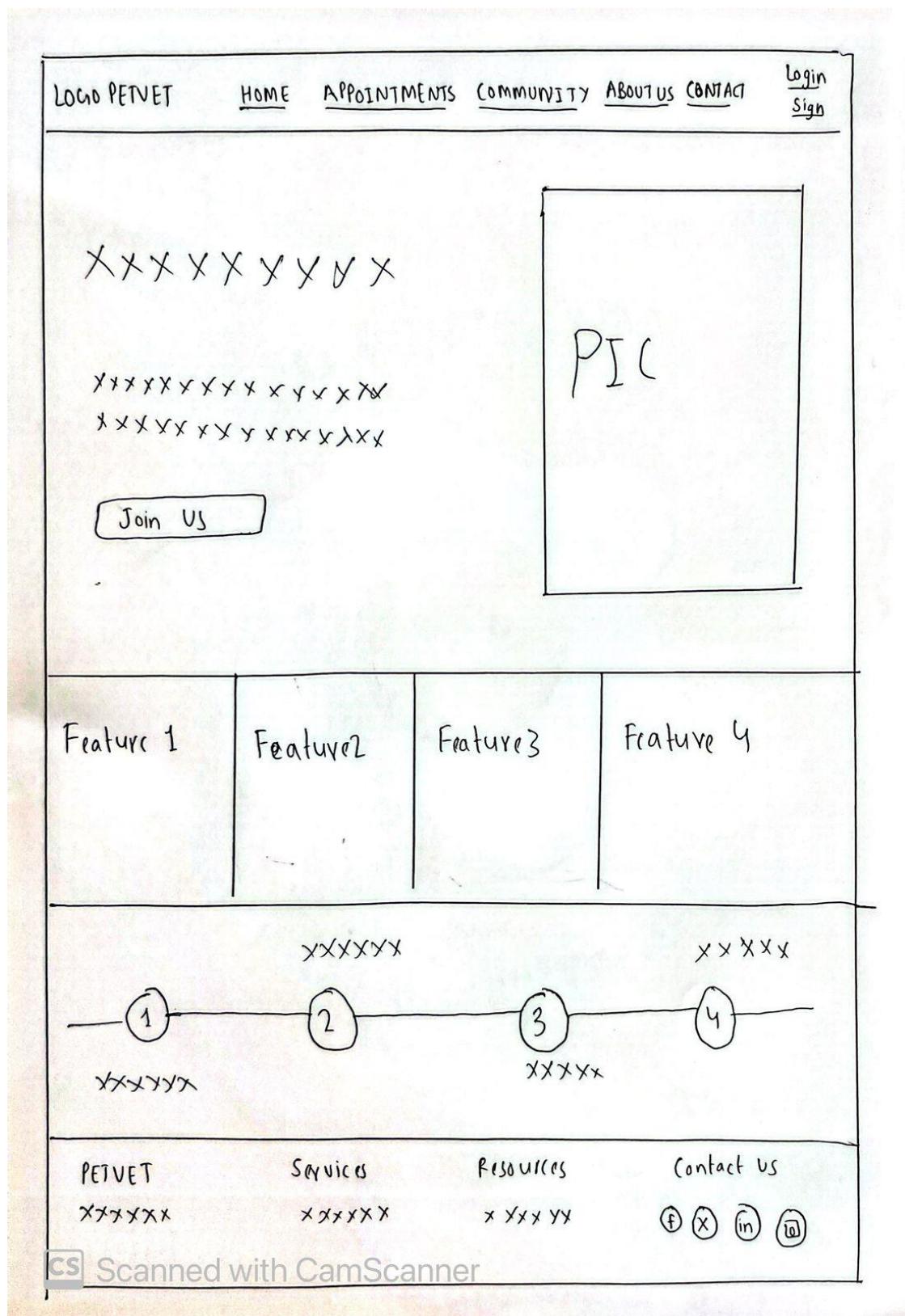


Figure 28: Wireframe for Landing

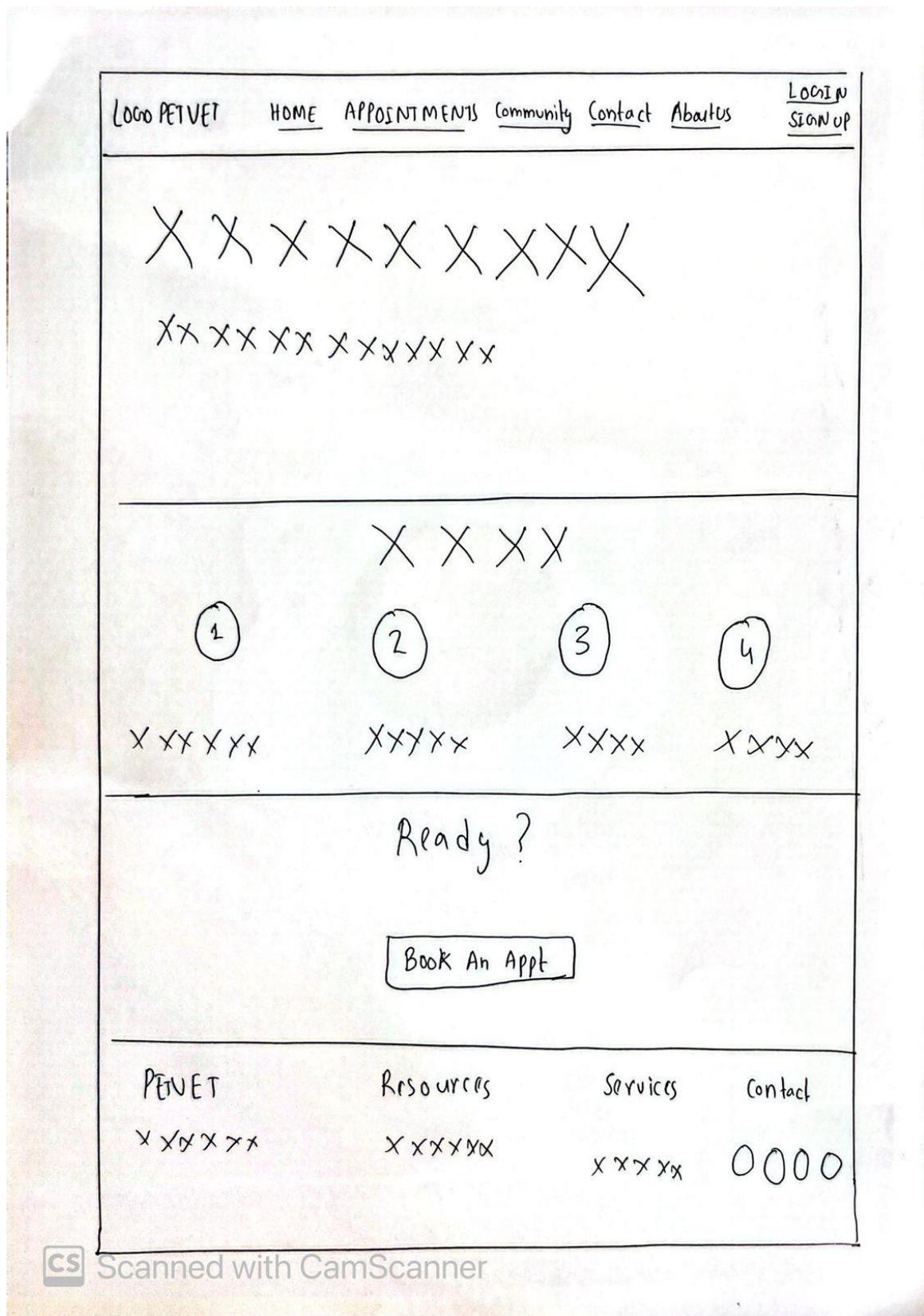


Figure 29: Wireframe for Landing-appointment

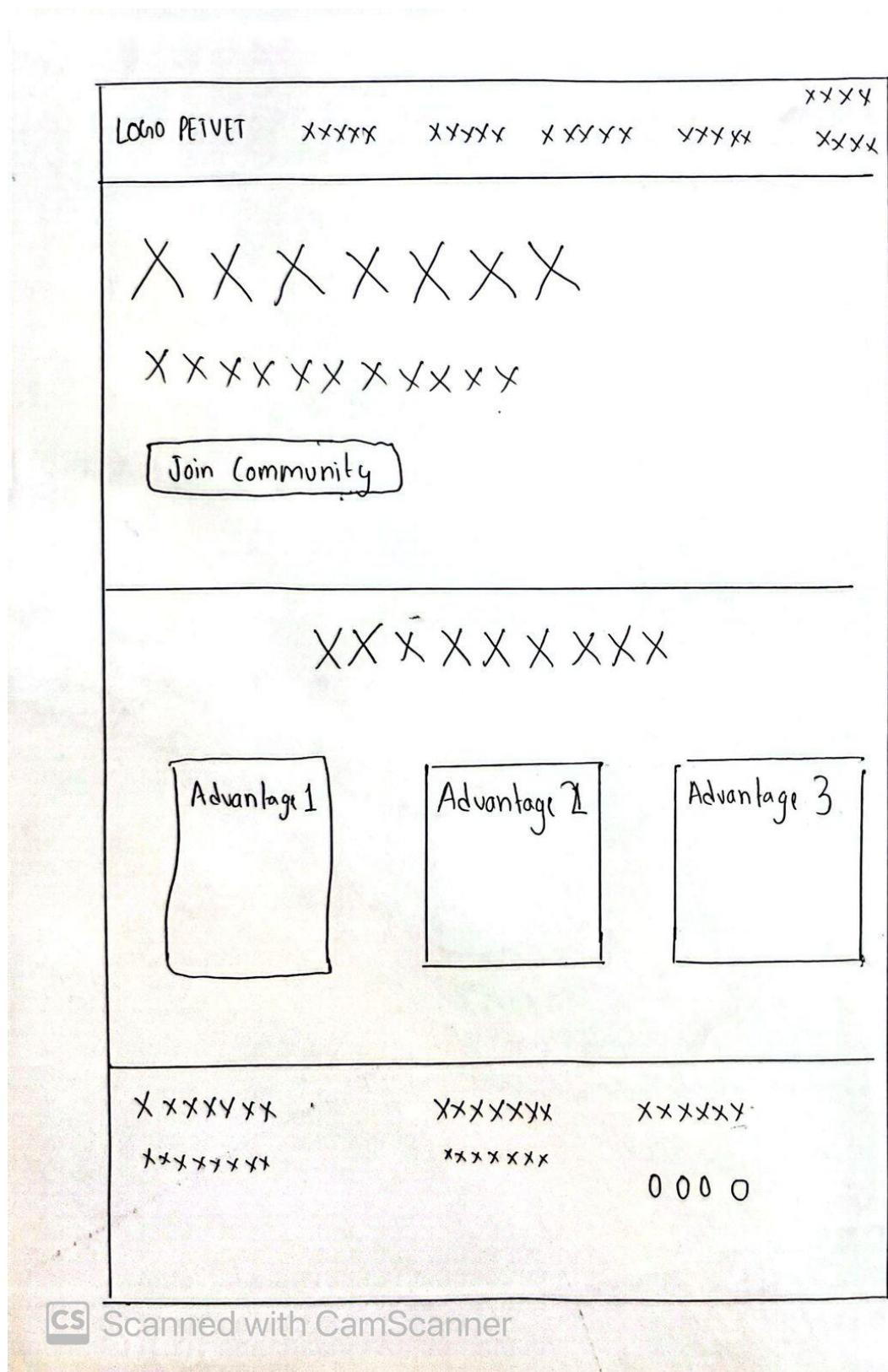
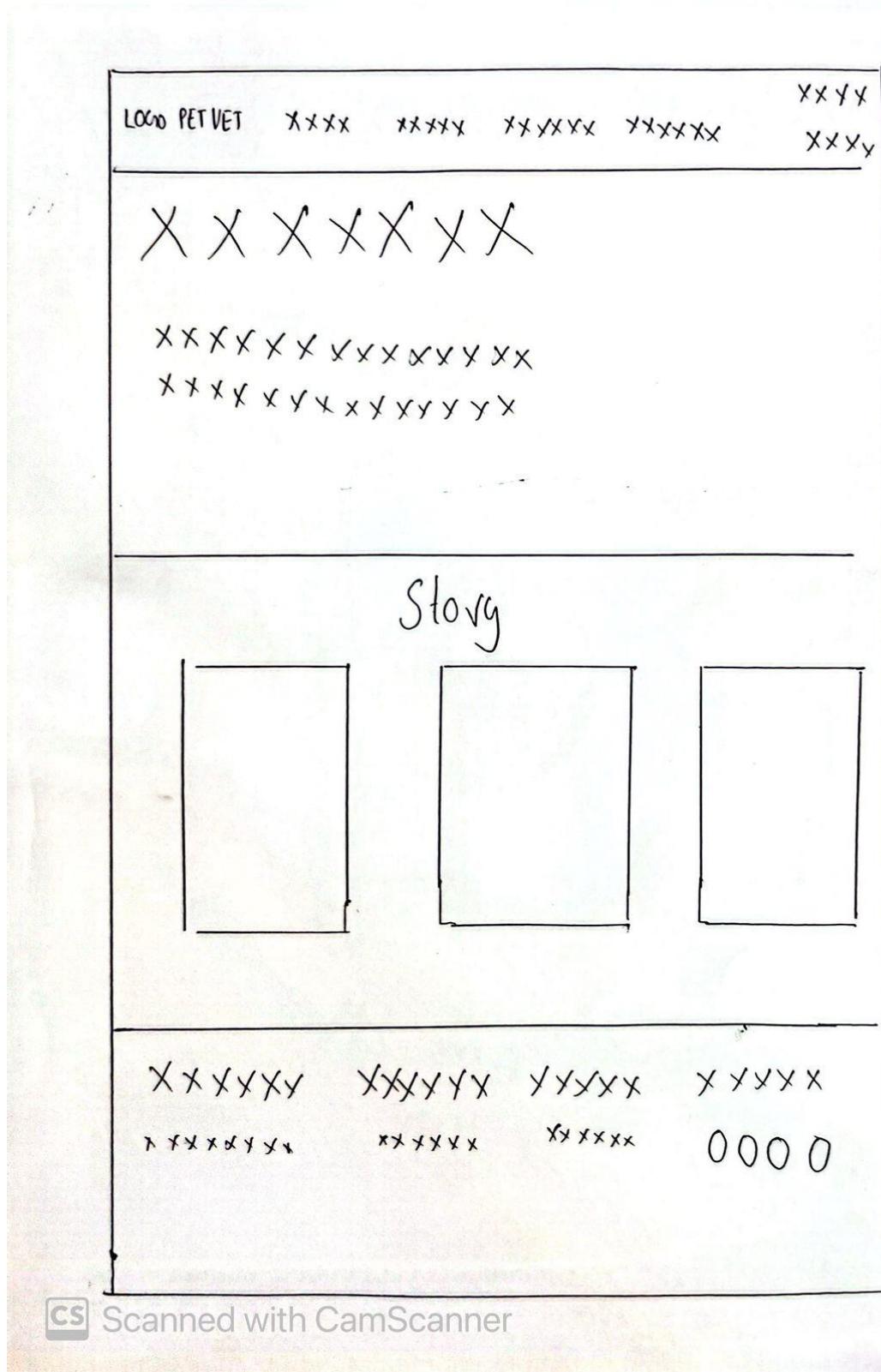


Figure 30: Wireframe for Landing-community



Scanned with CamScanner

Figure 31: Wireframe for Landing-about-us

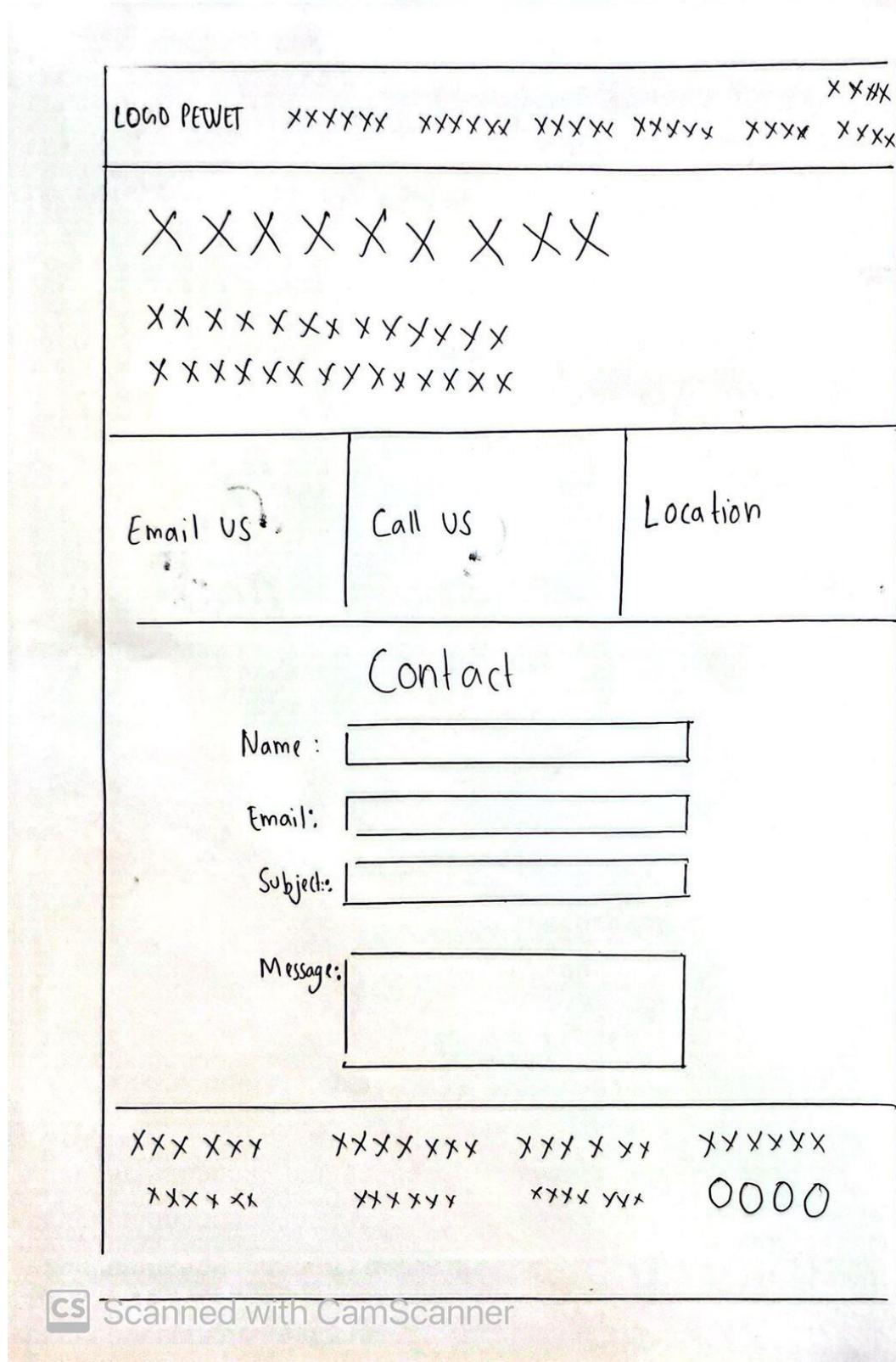


Figure 32: Wireframe for Landing-contact

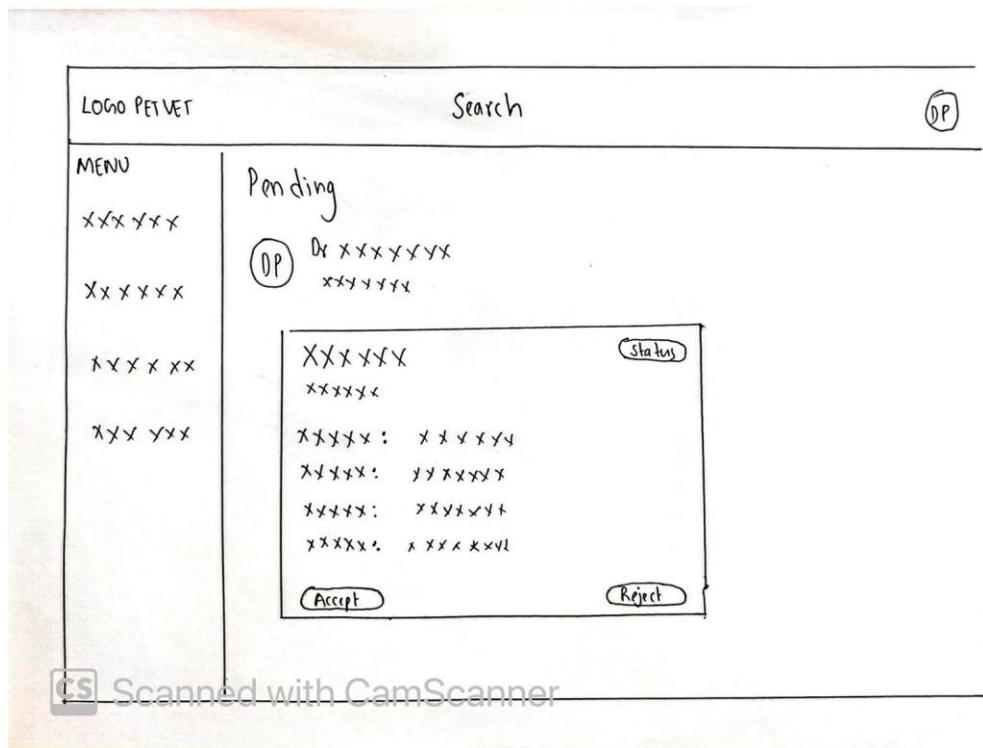


Figure 33: Wireframe for Pending booking requests

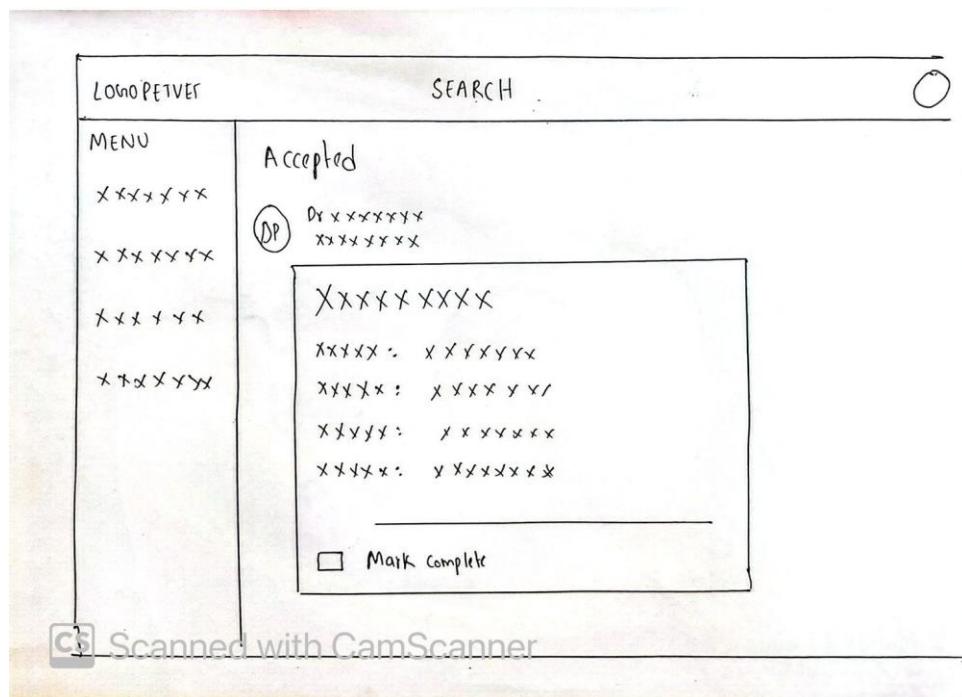


Figure 34: Wireframe for Accepted Bookings

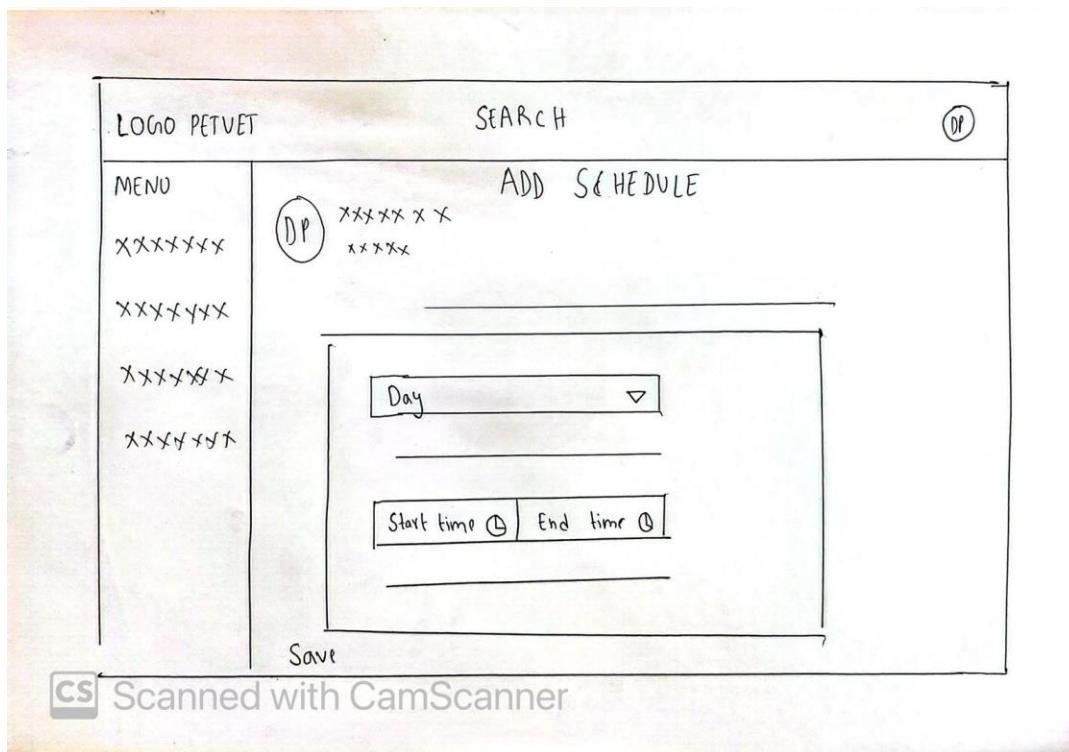


Figure 35: Wireframe for Add schedule

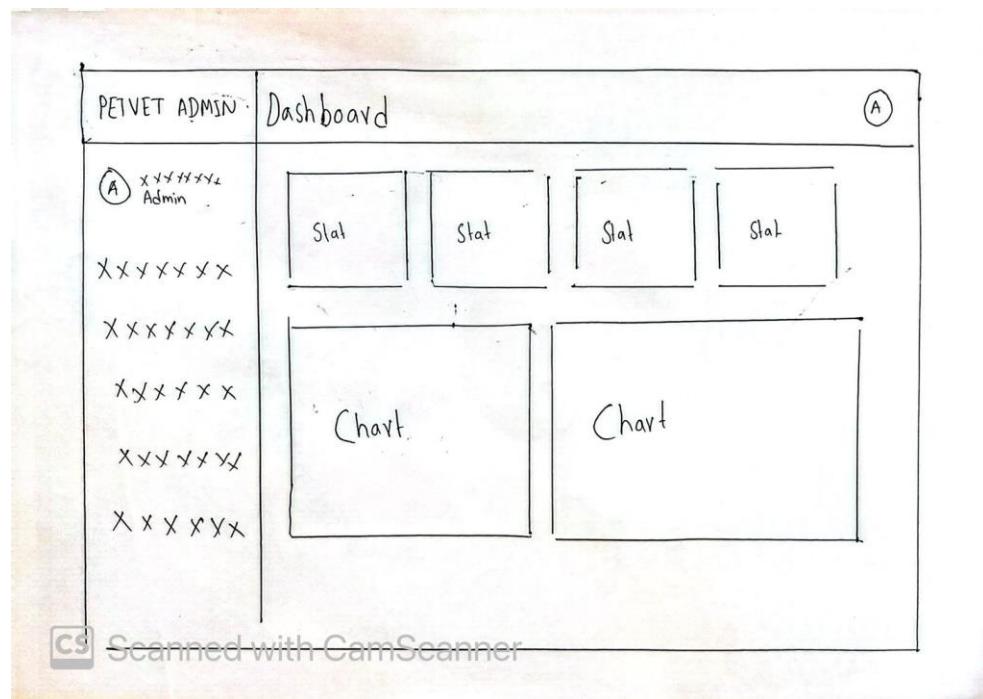


Figure 36: Wireframe for Admin Dashboard

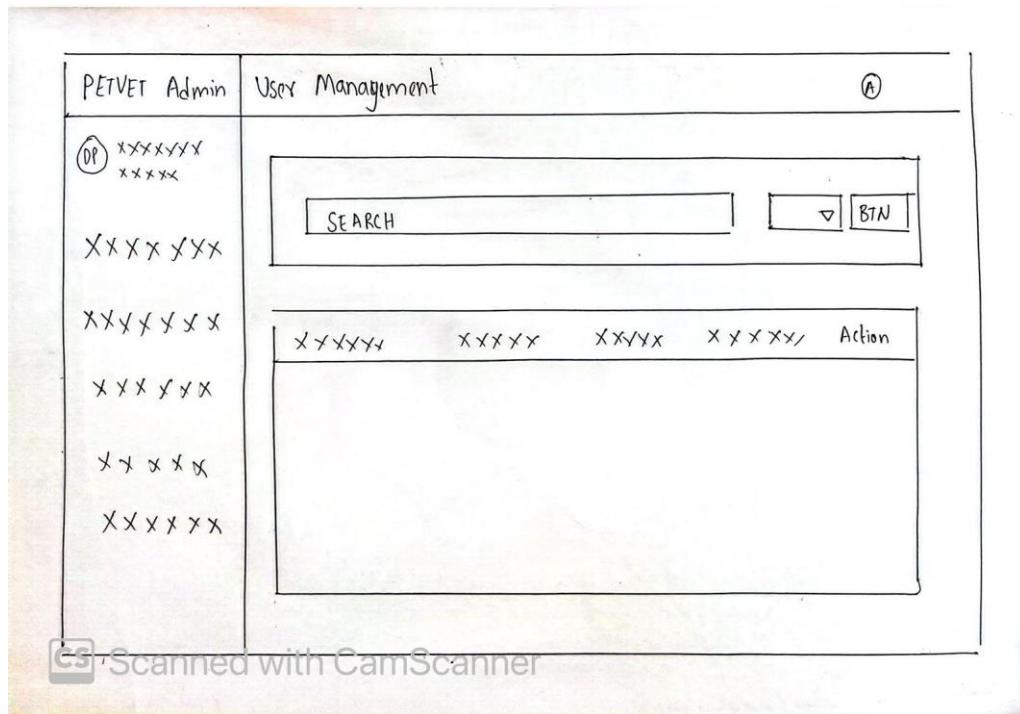


Figure 37: Wireframe for User management

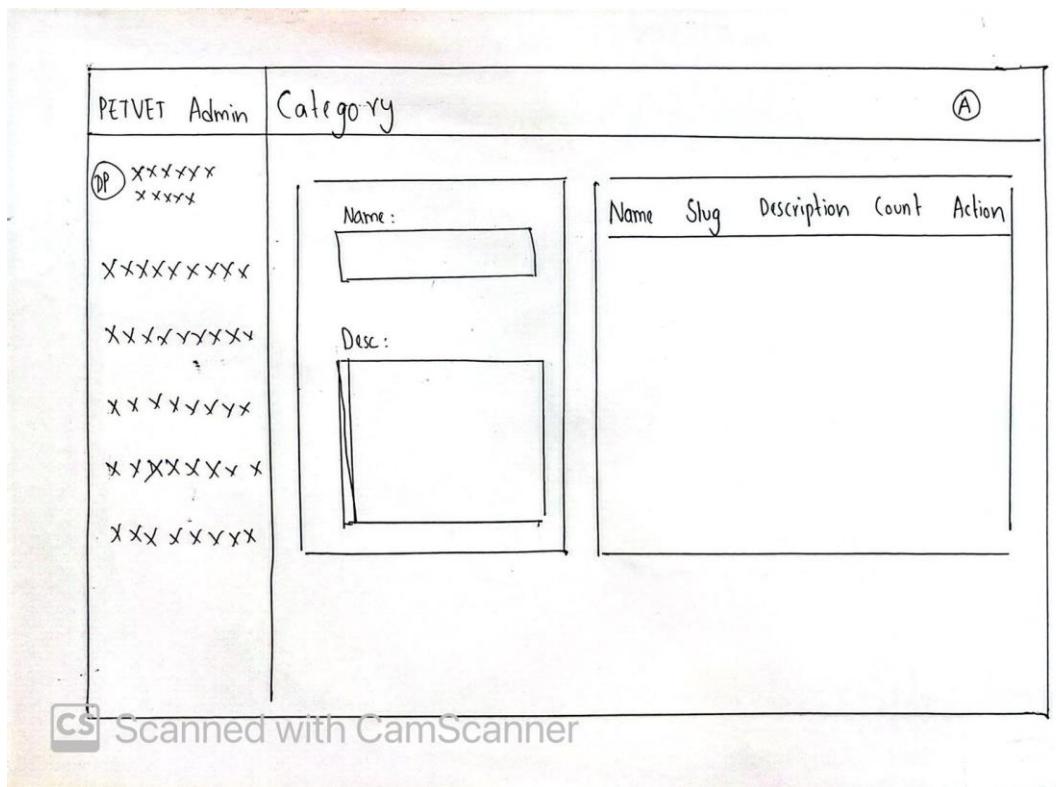


Figure 38: Wireframe for Category Management

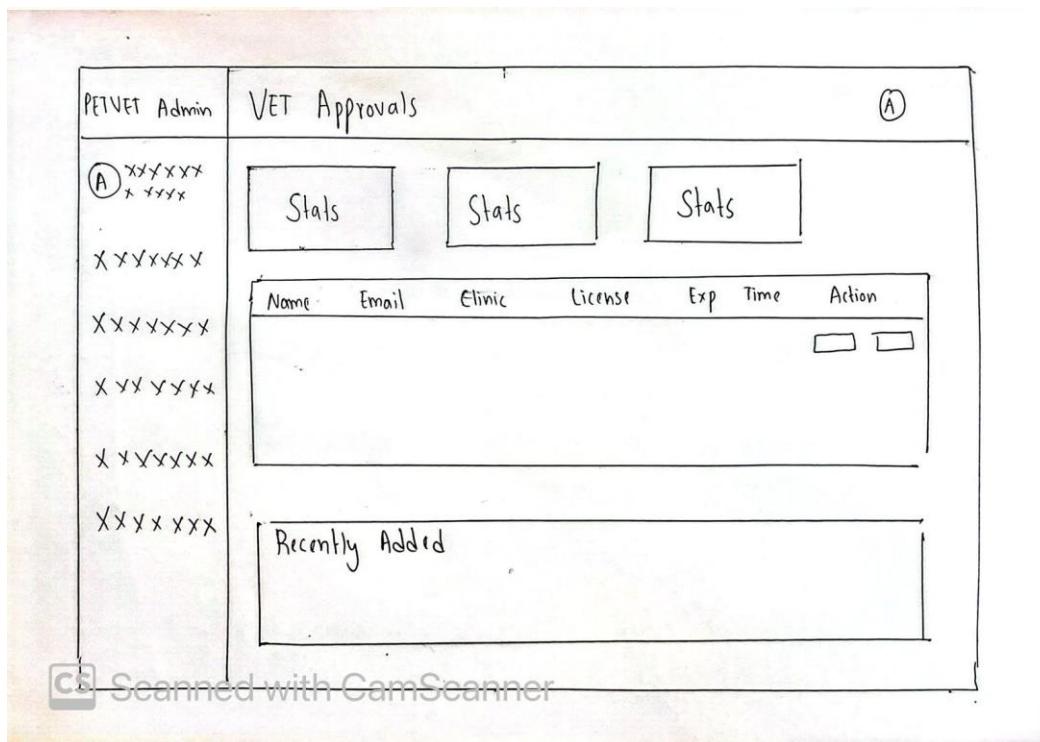


Figure 39: Wireframe for Vet approvals

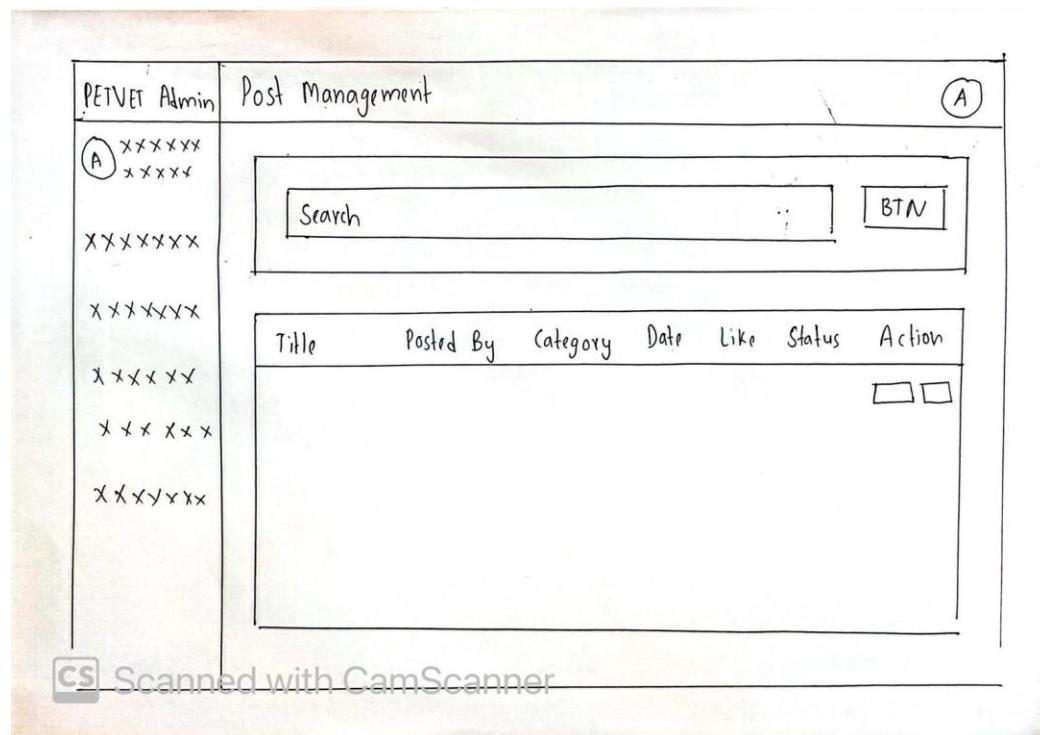


Figure 40: Wireframe for Post Management

3.6.1.2 Draft Figma Designs

To start development, I quickly drafted a design in Figma. This gave me a chance to see the core features, refine the color scheme and bring clarity to the vision set by the wireframes. It was a base for further design iterations and made sure that things were consistent and moving forward with development.

Note: The early Figma UI designs appeared stretched due to incorrect frame settings caused by limited experience, but they still provided a useful structural layout that was refined in later stages.

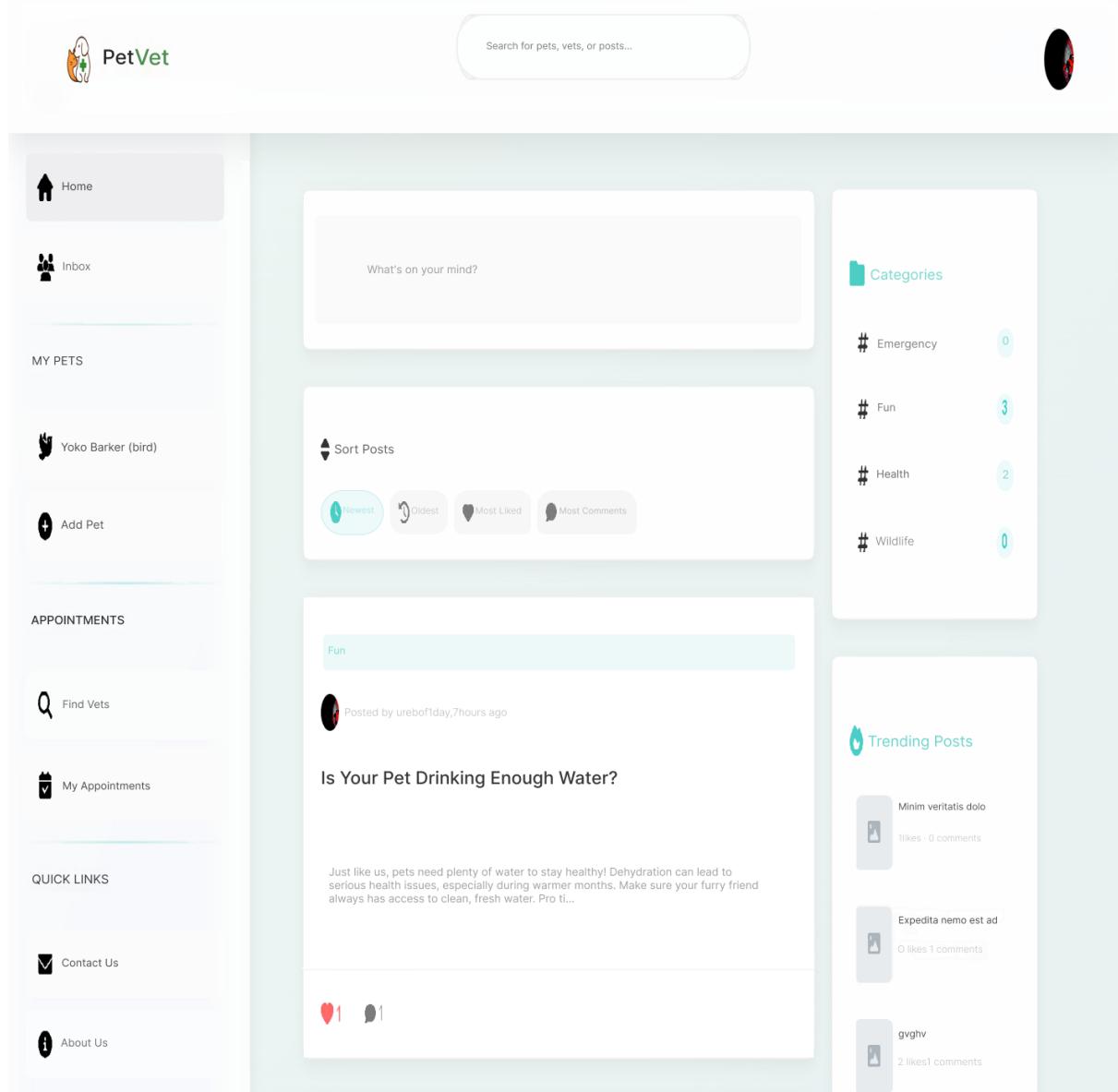


Figure 41: Draft Design for Home

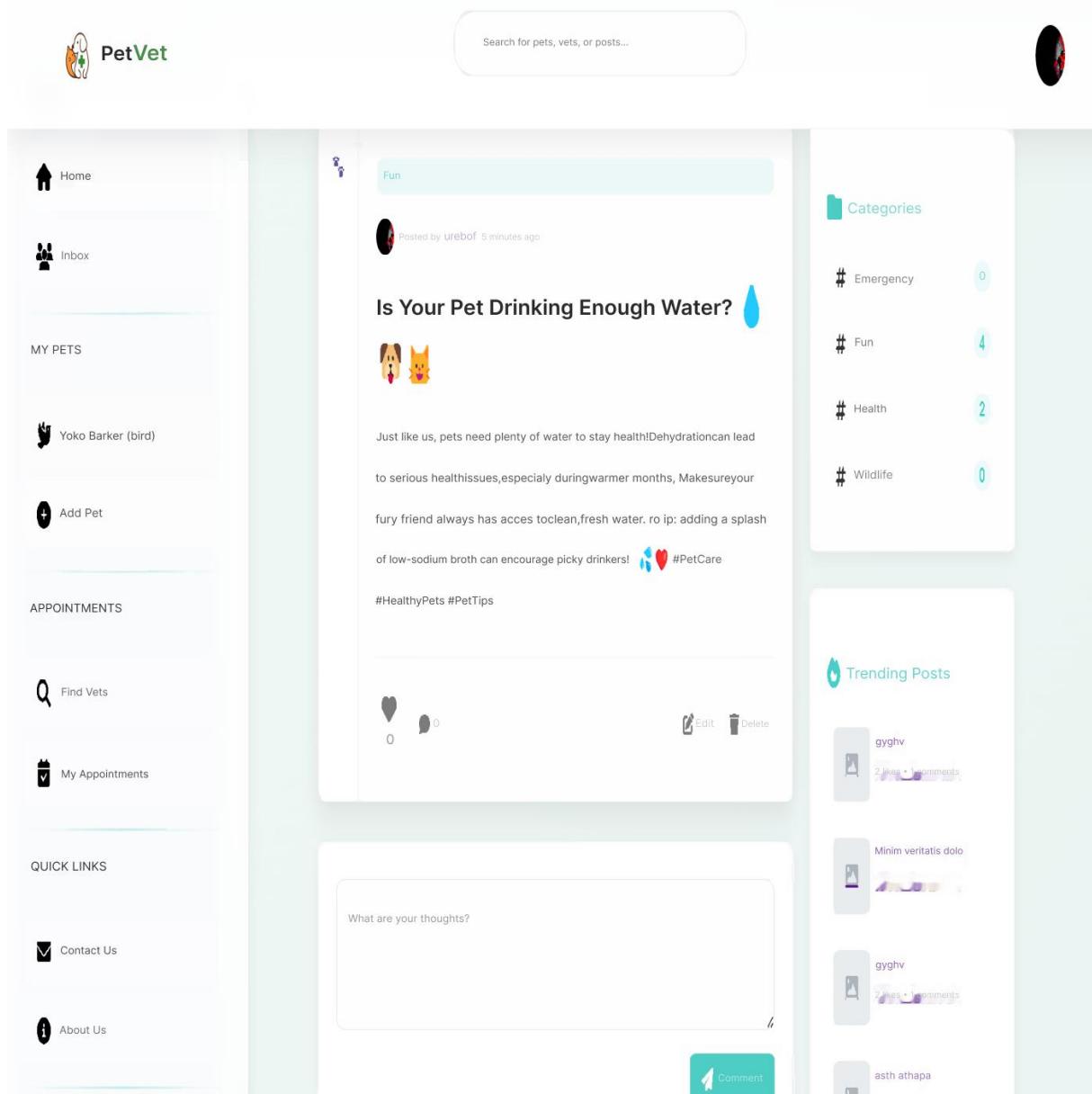


Figure 42: Draft Design for Post details

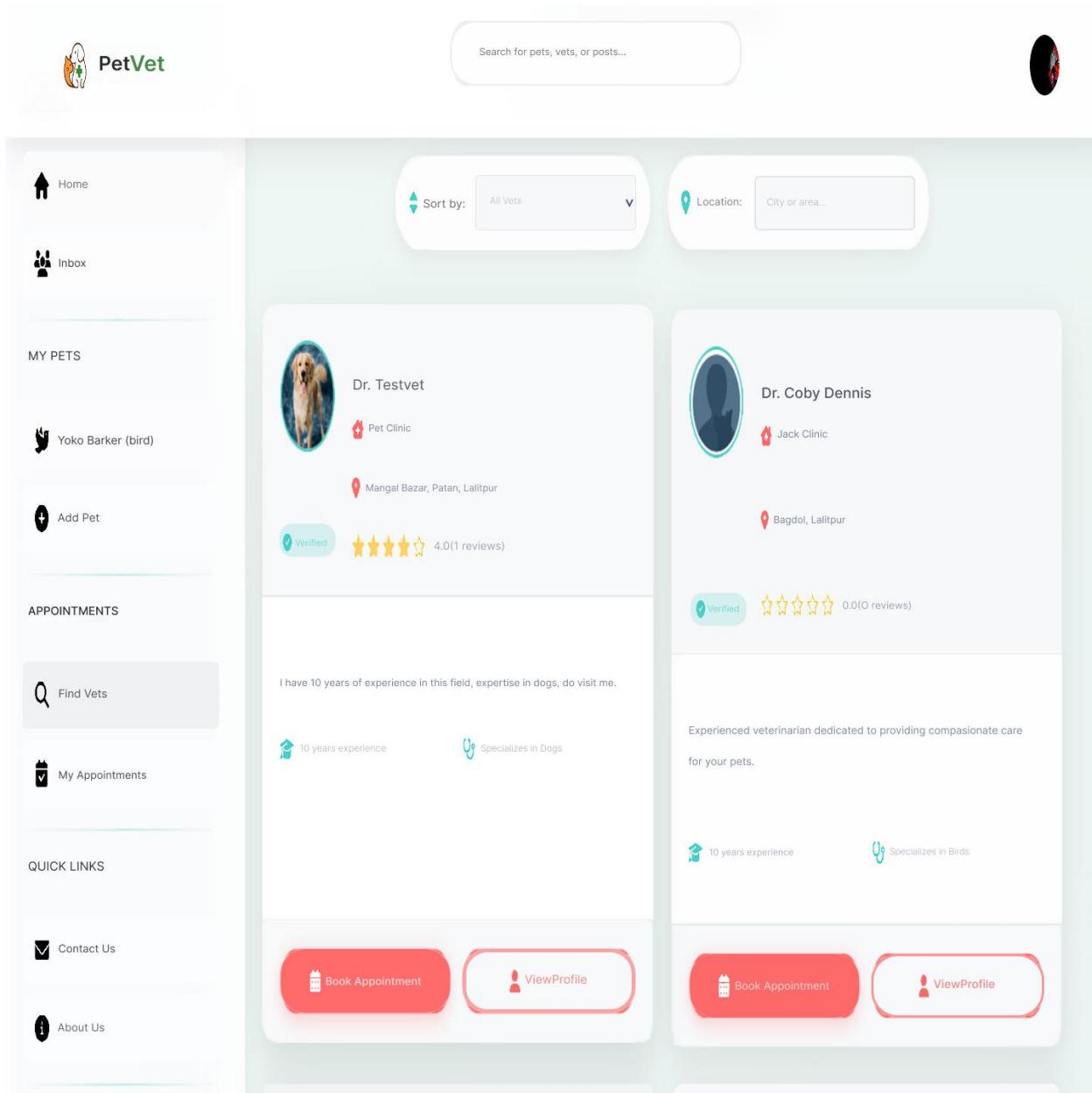


Figure 43: Draft Design for Find vets

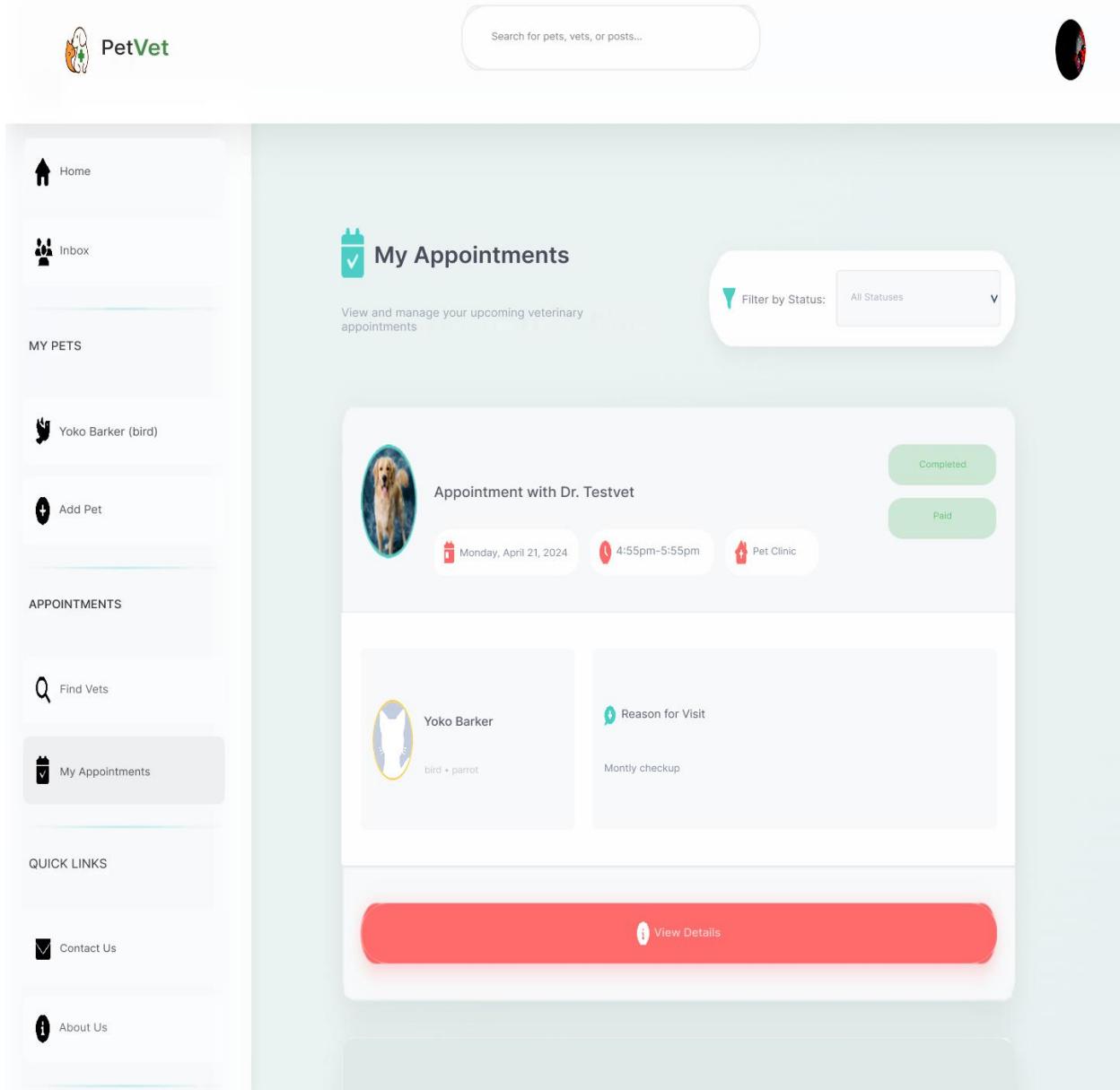


Figure 44: Draft Design for Appointments list

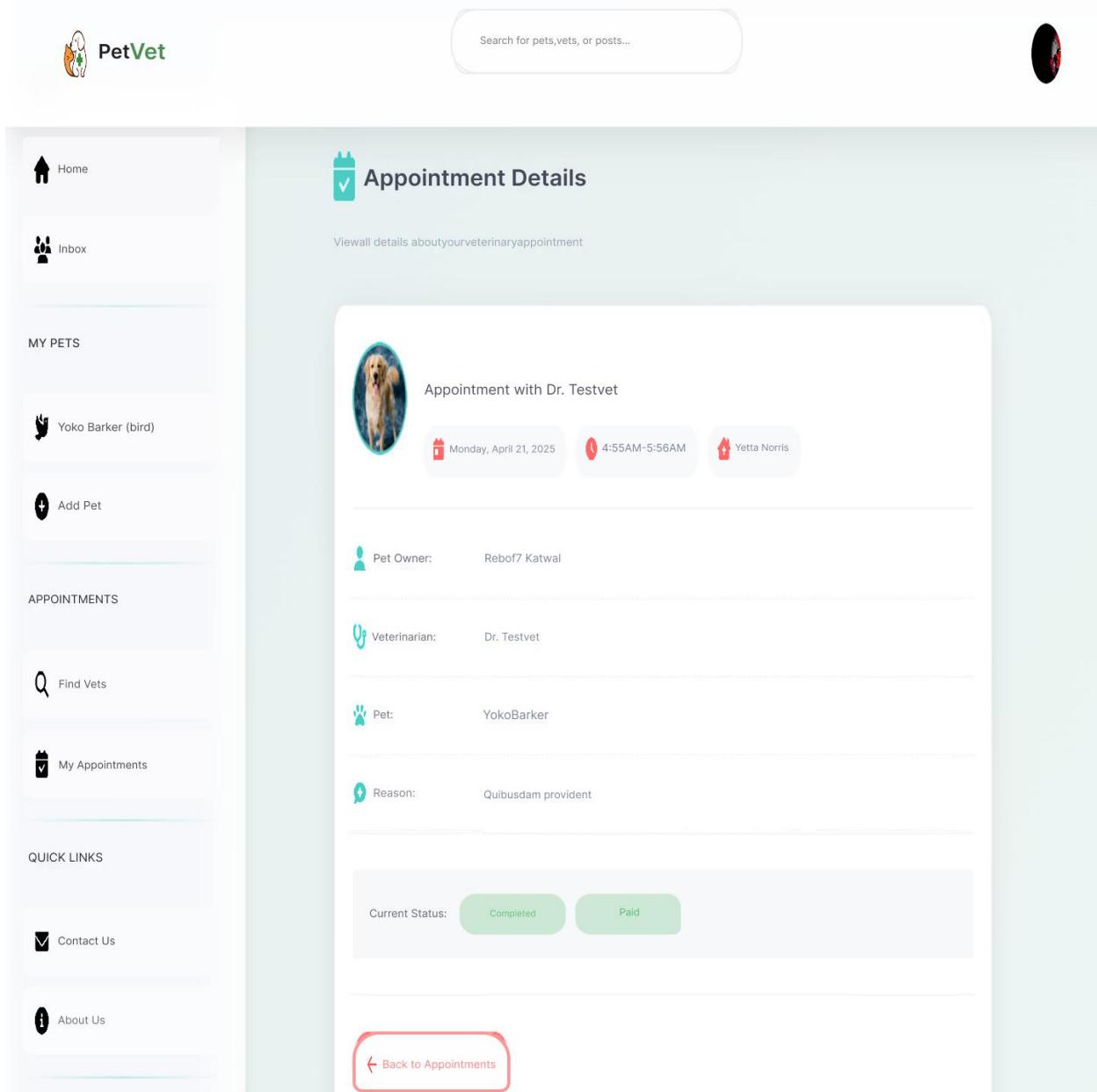


Figure 45: Draft Design for Appointment details

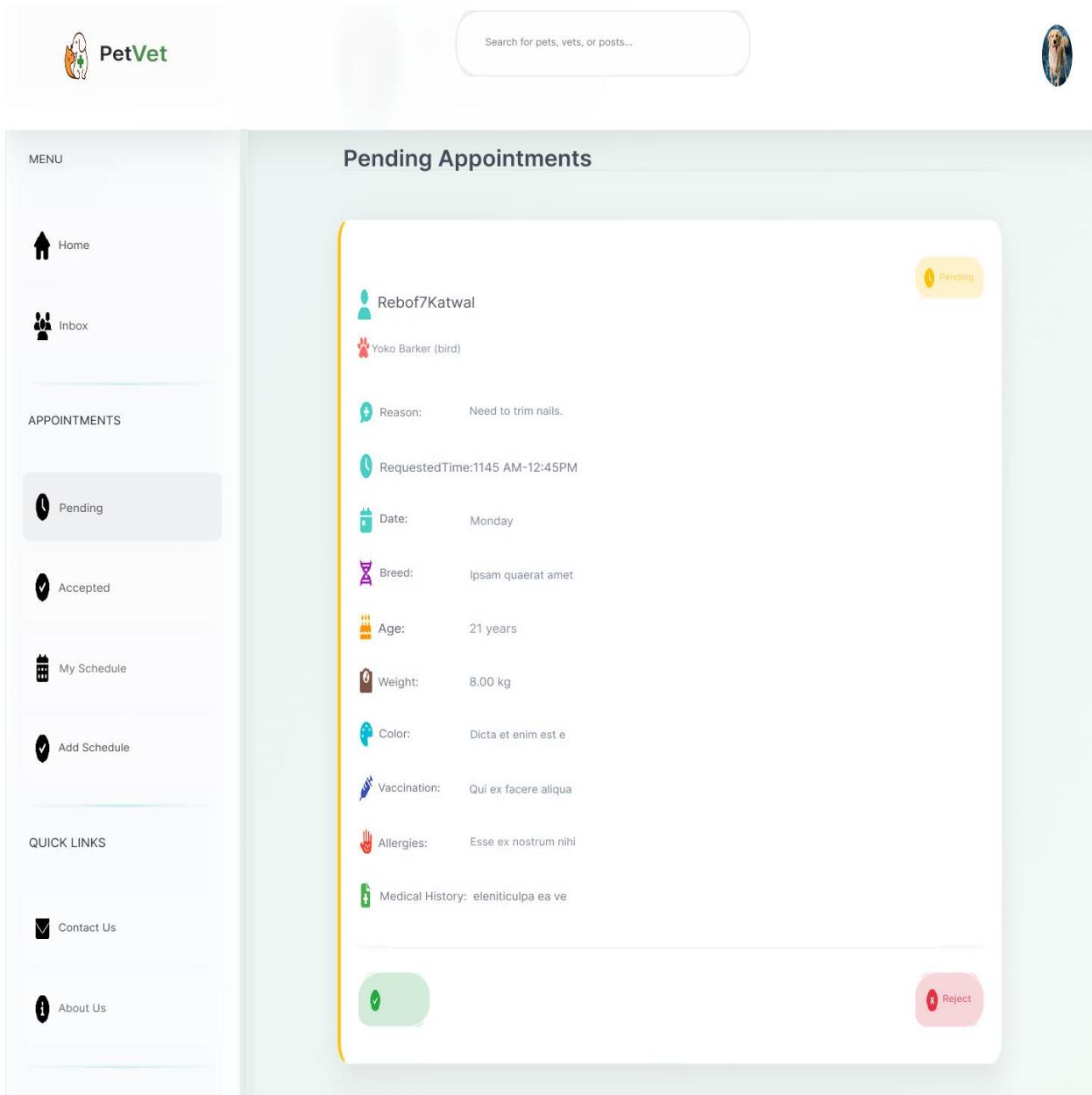


Figure 46: Draft Design for Pending requests

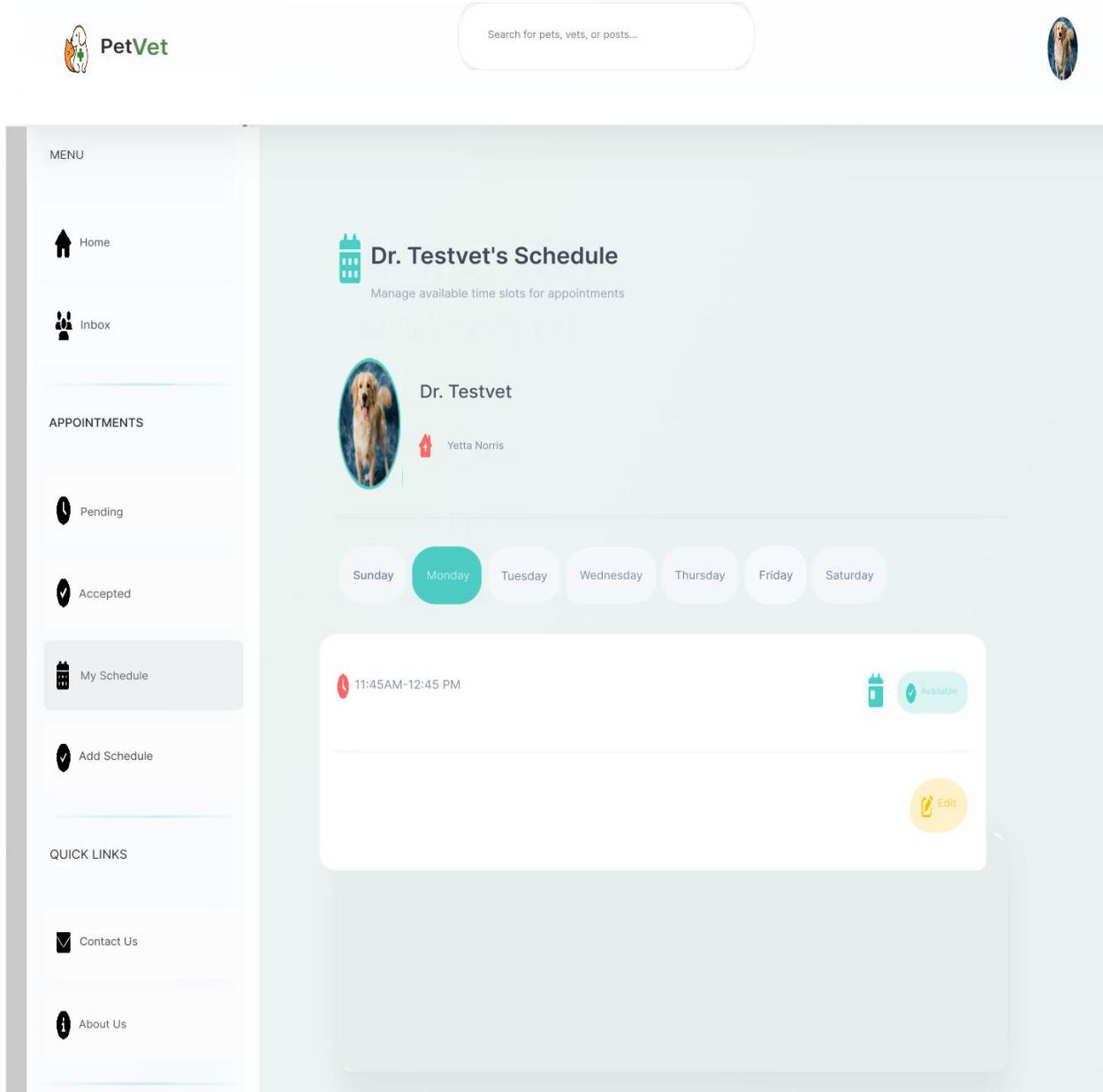


Figure 47: Draft Design for Vet Schedule lists

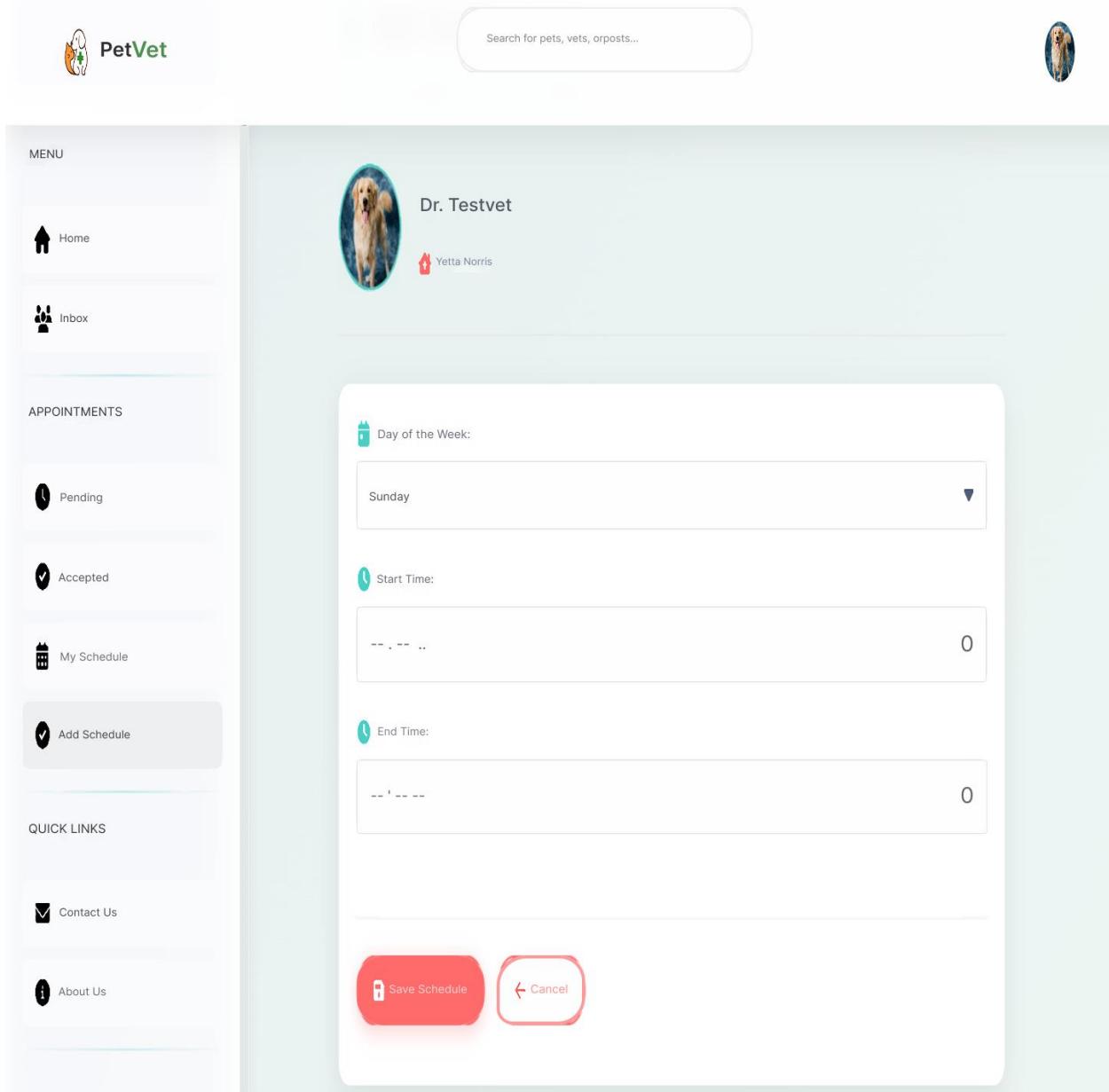


Figure 48: Draft Design for Add schedule

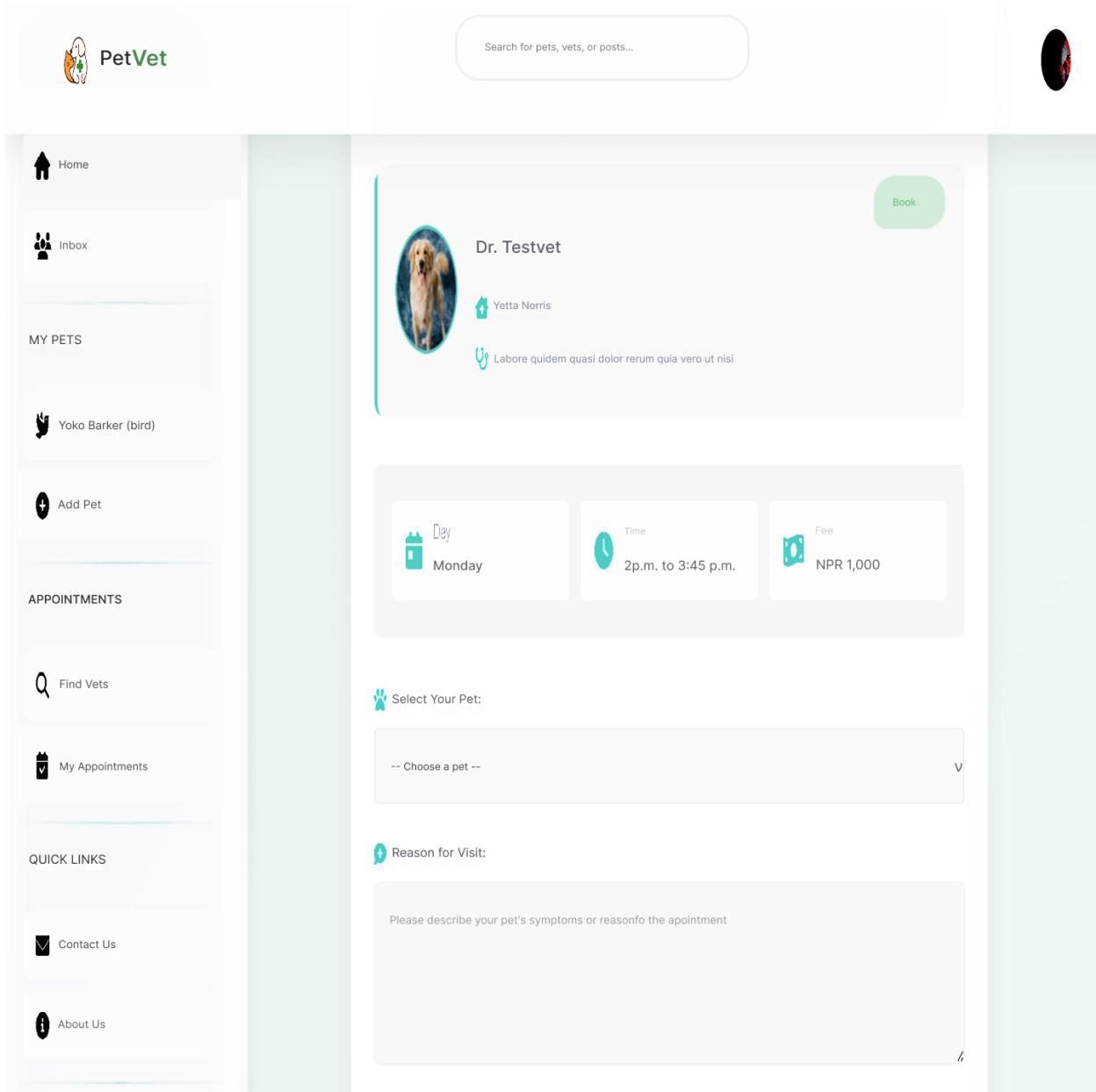


Figure 49: Draft Design for Booking

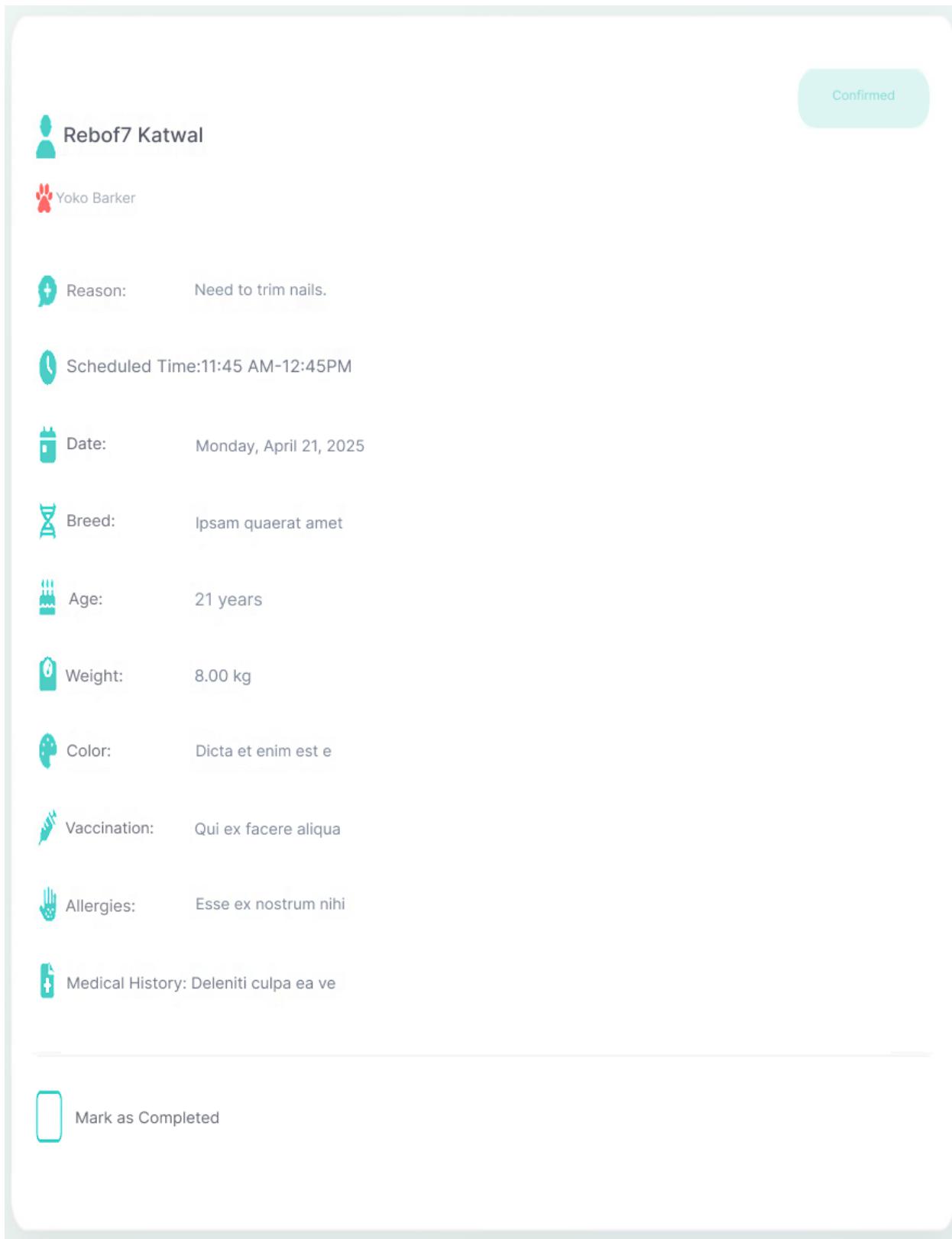


Figure 50: Draft Design for Accepting bookings

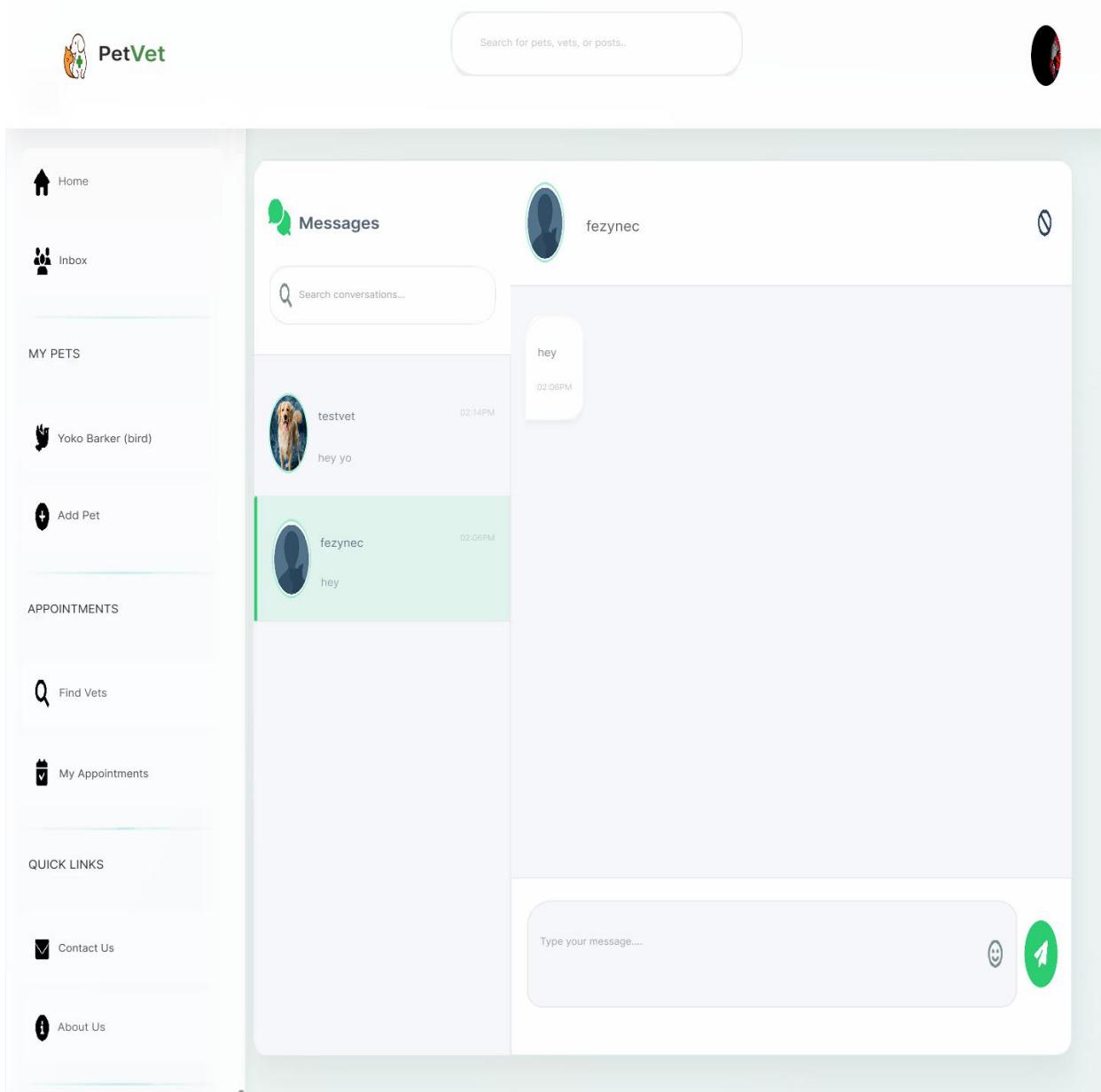


Figure 51: Draft Design for Inbox

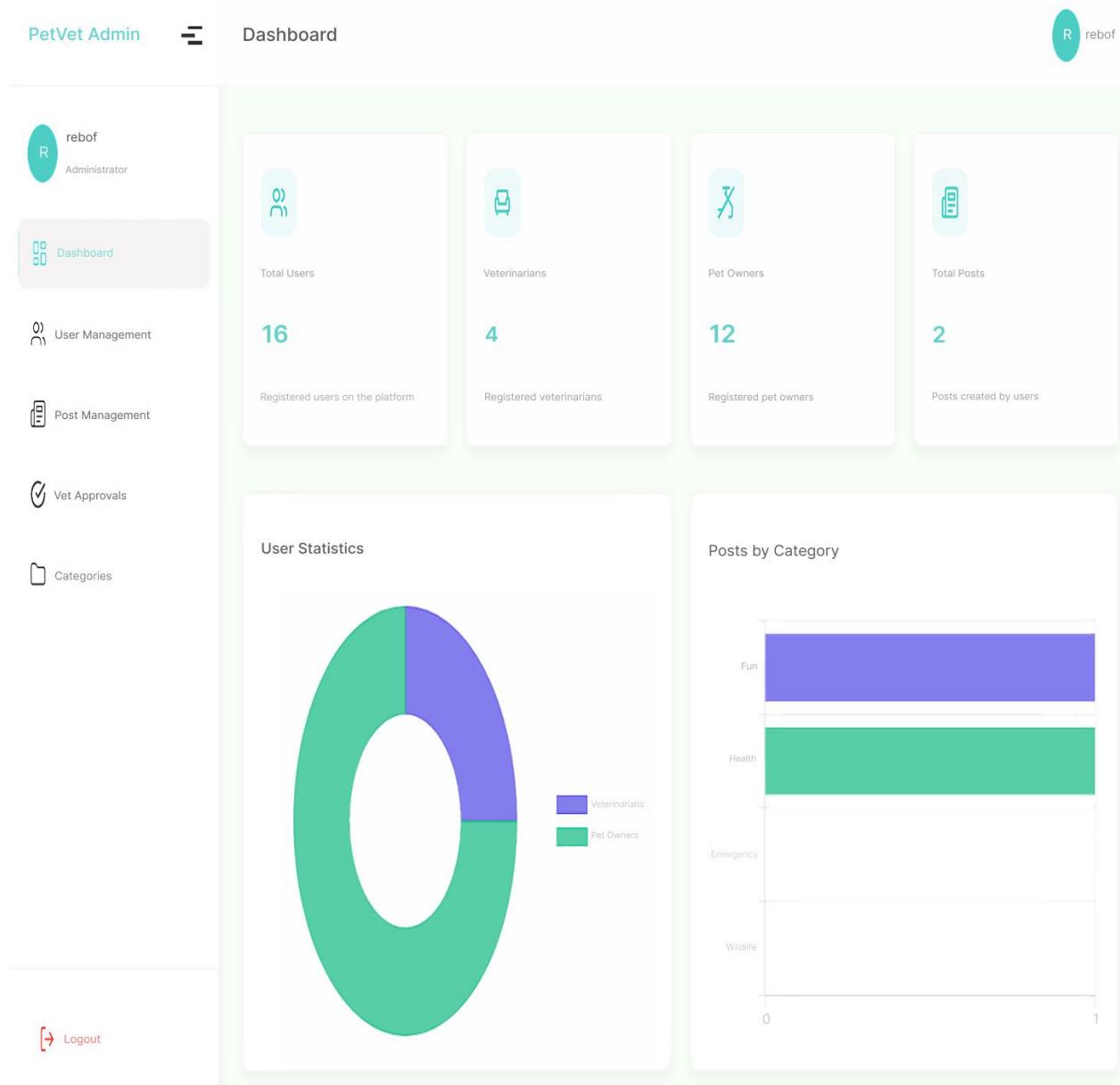


Figure 52: Draft Design for Admin Dashboard

The image shows a draft design for a category management application. It consists of two main sections: an 'Add New Category' form on the left and a 'Categories' list on the right.

Add New Category Form:

- Category Name:** An input field for entering the name of the new category.
- Description:** A large text area for describing the category.
- Add Category:** A teal-colored button at the bottom of the form.

Categories List:

Name	Slug	Description	Post Count	Actions
Emergency	emergency-en	test	0	<button>Edit</button> <button>Delete</button>
Wildlife	wildlife-nf	Wild animal	0	<button>Edit</button> <button>Delete</button>

At the top right of the interface, there is a user profile icon with the letter 'R' and the name 'rebof'.

Figure 53: Draft Design for Category management

The screenshot displays a user interface for 'Vet Approvals' under the 'PetVet Admin' section. The top navigation bar includes a profile icon for 'rebof' (Administrator), a search bar, and a 'Logout' button.

Sidebar Navigation:

- Dashboard
- User Management
- Post Management
- Vet Approvals** (highlighted with a pink background)
- Categories
- Logout

Main Content Area:

Pending Approvals

0 Vets waiting for approval

Recently Approved

1 Vets approved in the last 7 days

Recently Declined

0 Vets declined in the last 7 days

Pending Vet Approvals

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
No pending vet approvals.						

Recently Approved Vets

Name	Email	Clinic Name	License Number	Approved At
Coby Dennis	testpet34@gmail.com	Jack Guerrero	344	Apr 17, 2025

Recently Declined Vets

Name	Email	Clinic Name	License Number	Declined At
No recently declined vets.				

Figure 54: Draft Design for Vet Approvals

The screenshot displays a draft design for a 'User Management' feature within a 'PetVet Admin' application. The interface is divided into two main sections: a left sidebar and a right main content area.

Left Sidebar:

- PetVet Admin**: The title of the application.
- User Management**: The active section, indicated by a pink background.
- Dashboard**
- Post Management**
- Vet Approvals**
- Categories**
- Logout**: A red button at the bottom.

Right Main Content Area:

The main content area is titled 'User Management'. It features a header with 'Search Users' and 'User Type' dropdowns (set to 'All Users') and a 'Search' button. Below this is a table listing four user records:

Name	Username	Email	User Type	Joined	Status	Actions
Nicole Jo	wetugomeby	monofi@mailinator.com	Pet Owner	Apr 18, 2025	Verified	<button>View</button> <button>Delete</button>
Coby Dennis	fezynec	testpet34@gmail.com	Veterinarian	Apr 17, 2025	Verified	<button>View</button> <button>Delete</button>
Testvet	testvet	testvet@gmail.com	Veterinarian	Feb 12, 2025	Verified	<button>View</button> <button>Delete</button>
Rebof7 Katwal	rebof	rebofkatwal7@gmail.com	Pet Owner	Oct 11, 2024	Unverified	<button>View</button> <button>Delete</button>

Figure 55: Draft Design for User management

The image shows a draft design for a 'Post Management' application. The interface is divided into two main sections: a left sidebar and a right main content area.

Left Sidebar:

- PetVet Admin
- Administrator (rebof)
- Dashboard
- User Management
- Post Management** (highlighted with a pink background)
- Vet Approvals
- Categories
- Logout

Main Content Area:

Post Management

Search Posts

Category

All Categories ▾

Search

Title	Author	Category	Date	Likes	Status	Actions
5 Simple Tips to Keep Your Pet Healthy and Happy	wetugomeby	Health	Apr 18, 2025	2	Active	View Delete

Figure 56: Draft Design for Post management

3.6.2 Initial Development

The development team started by designing core system features that consisted of user authentication elements. The system developed registration options with login capabilities for vets and pet owners through which vets required admin inspection and pet owners needed verification through an OTP process. System development began with creating user profiles to sustain upcoming system developments.

3.6.2.1 Initial Development: Figma Design

It was based on the wireframes and draft UI color scheme so the layout and user experience emerged more clearer than before.

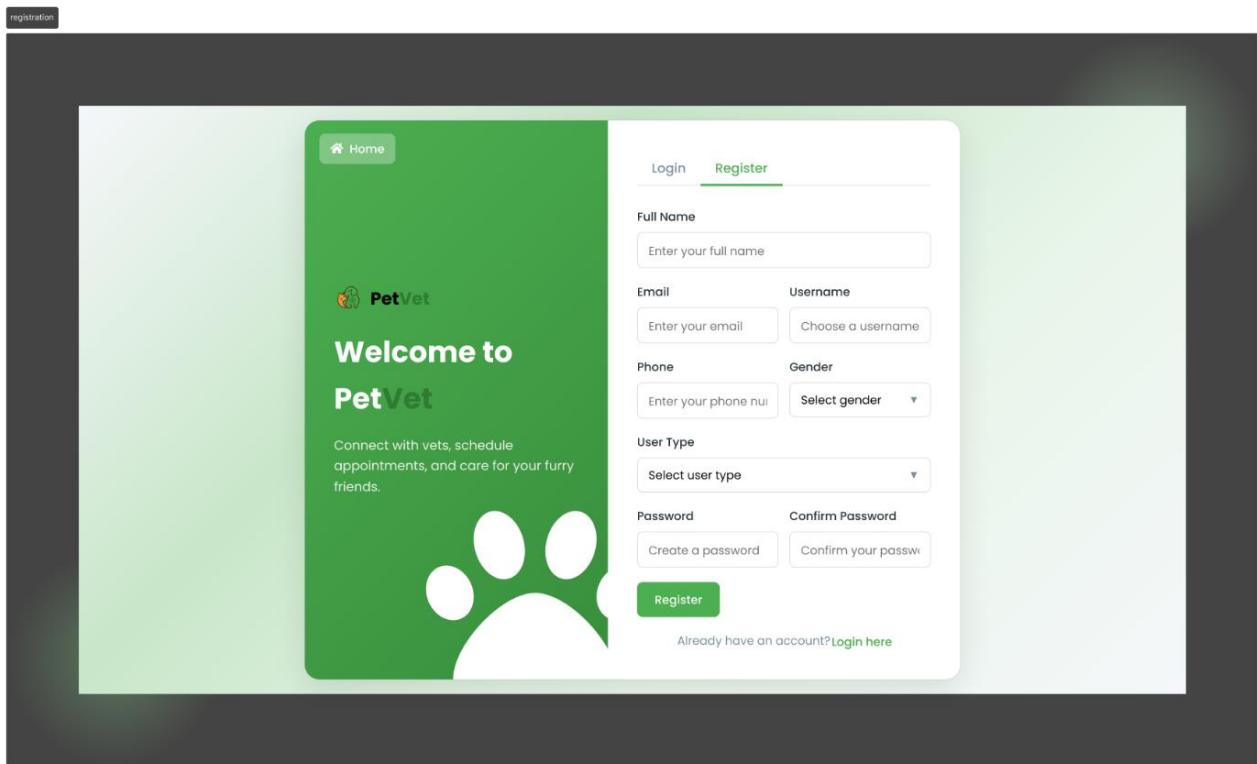
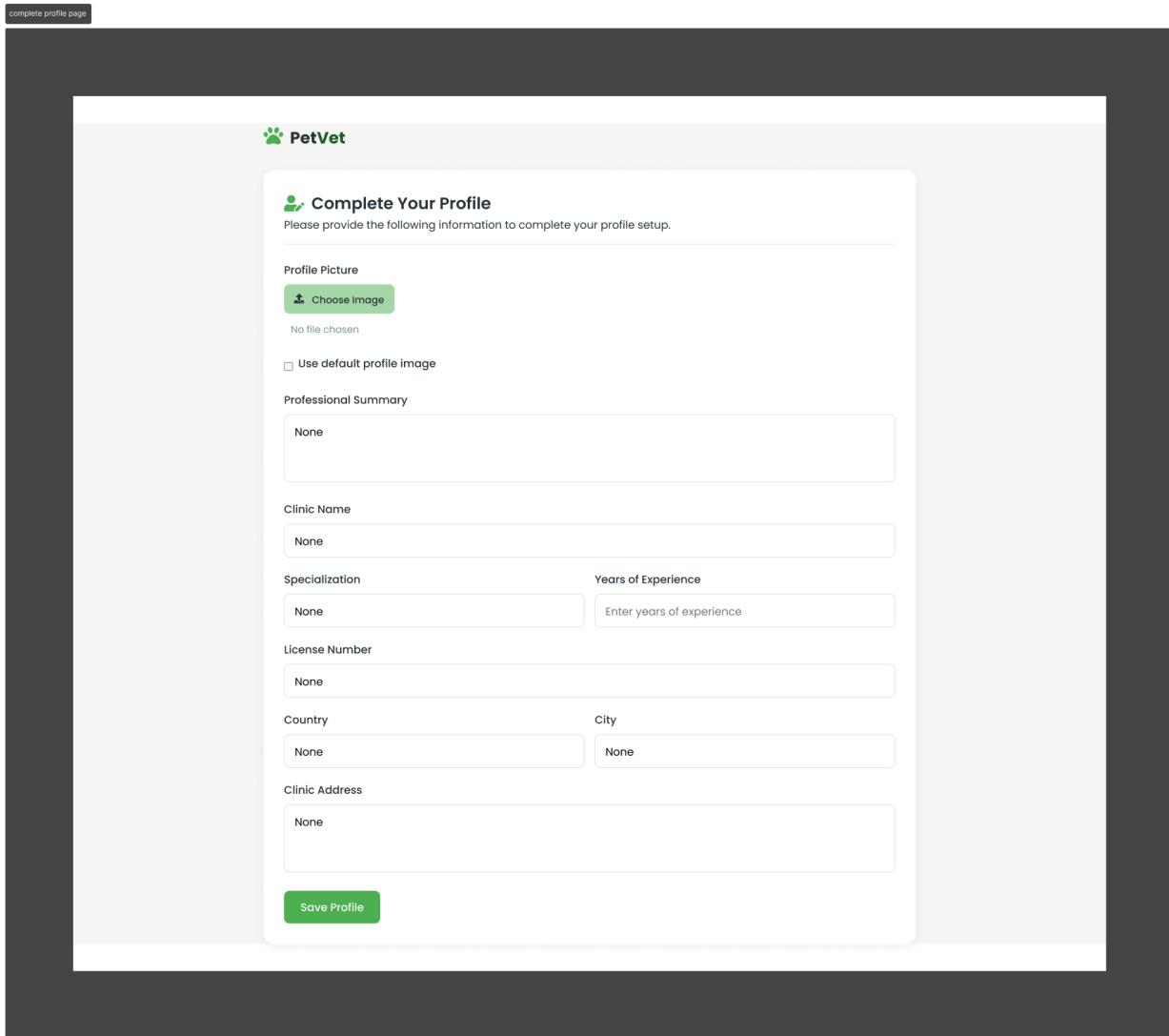


Figure 57: Main Design for Registration



The image shows a screenshot of a web application interface titled "PetVet". At the top left, there is a small button labeled "complete profile page". The main content area is titled "Complete Your Profile" with a sub-instruction "Please provide the following information to complete your profile setup." Below this, there are several input fields and options:

- Profile Picture:** A green button labeled "Choose Image" with the placeholder "No file chosen". Below it is a checkbox labeled "Use default profile image".
- Professional Summary:** A dropdown menu showing "None".
- Clinic Name:** A dropdown menu showing "None".
- Specialization:** A dropdown menu showing "None".
- Years of Experience:** A text input field with the placeholder "Enter years of experience".
- License Number:** A dropdown menu showing "None".
- Country:** A dropdown menu showing "None".
- City:** A dropdown menu showing "None".
- Clinic Address:** A dropdown menu showing "None".

At the bottom center of the form is a green "Save Profile" button.

Figure 58: Main Design for Complete your profile

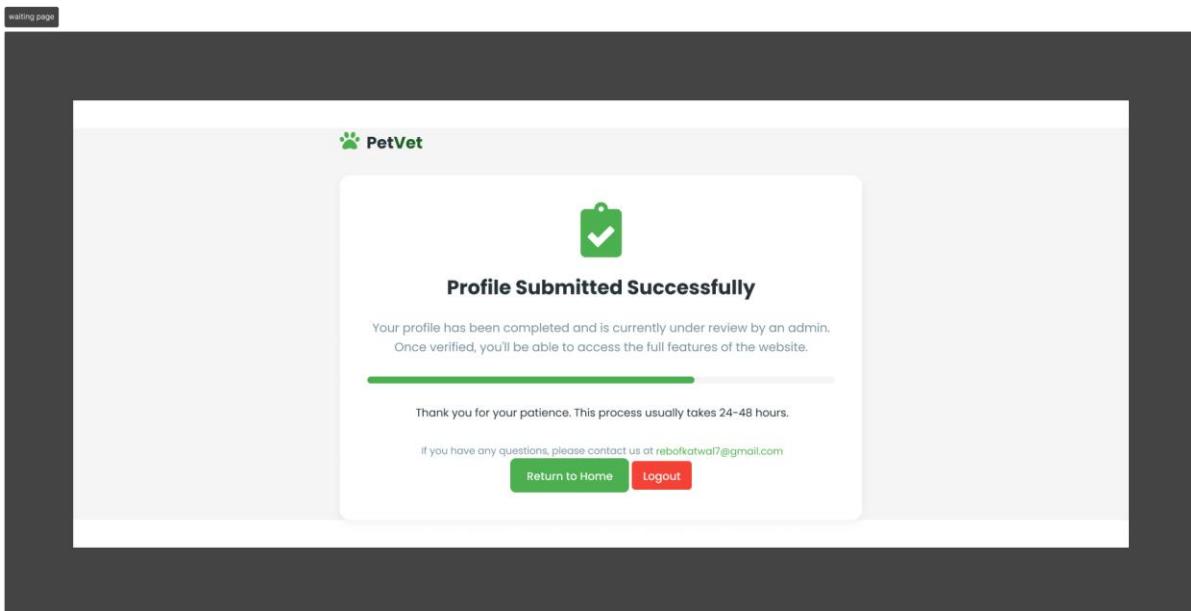


Figure 59: Main Design for Waiting page

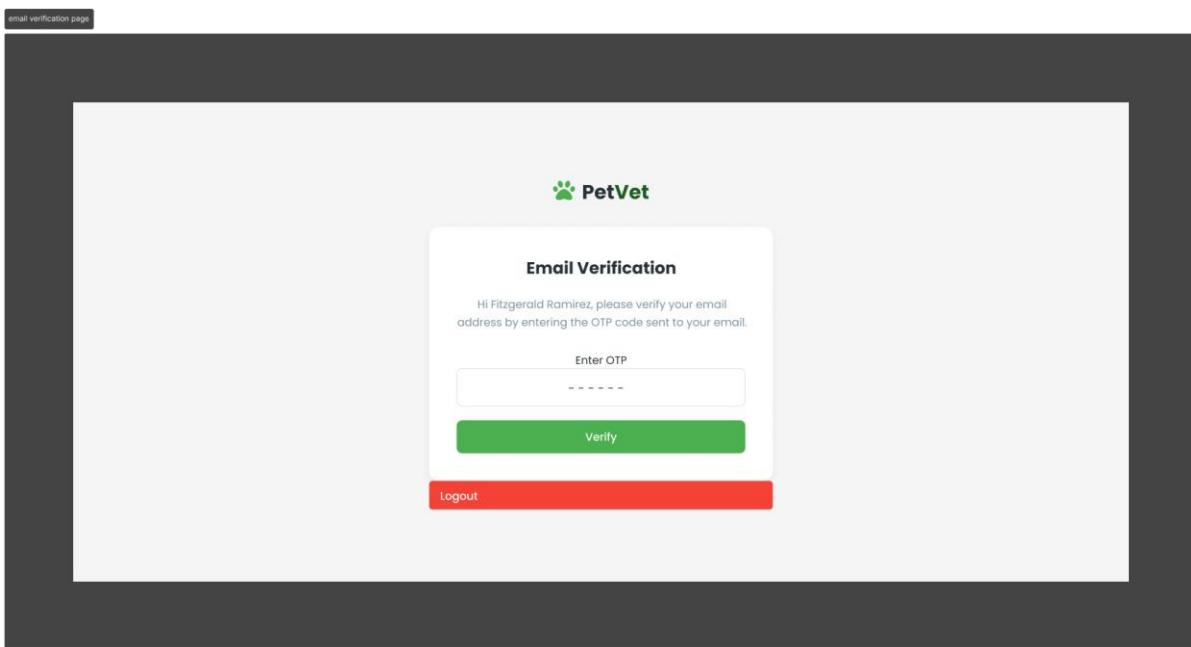


Figure 60: Main Design for OTP verification

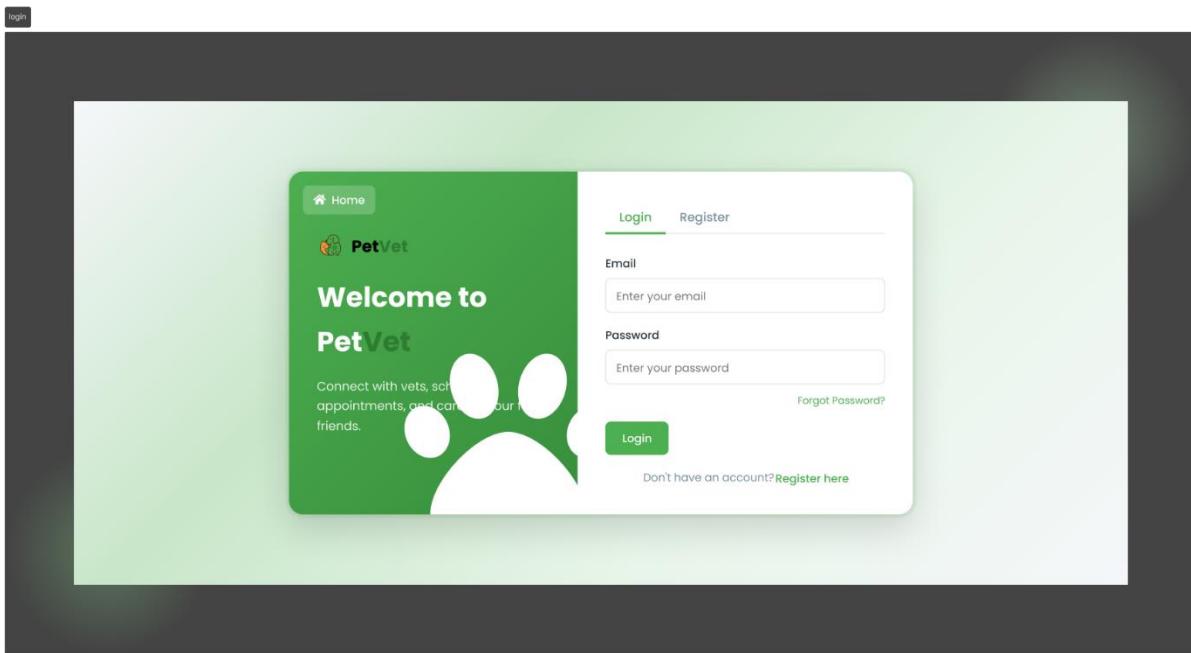


Figure 61: Main Design for Login

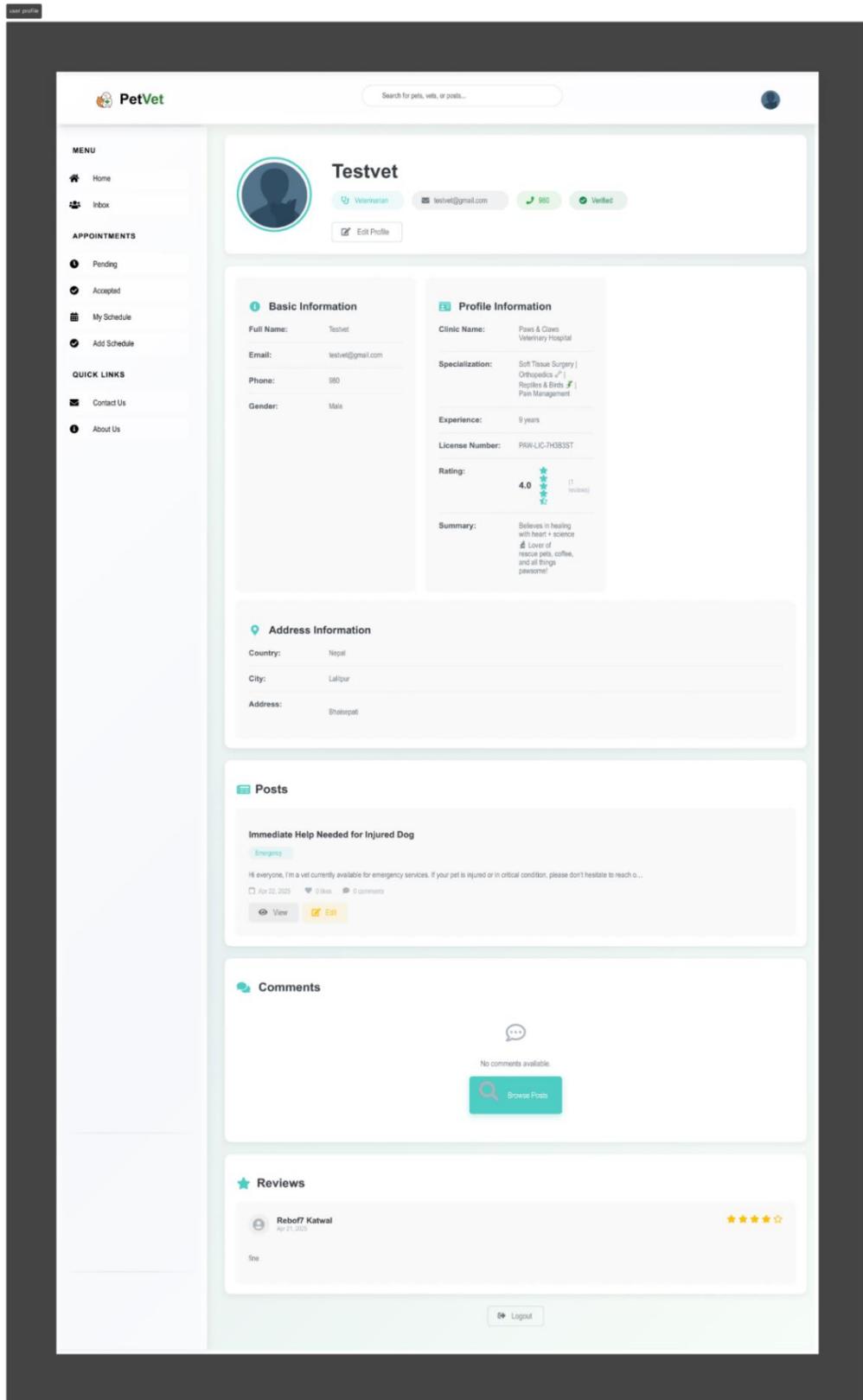


Figure 62: Main Design for User profile

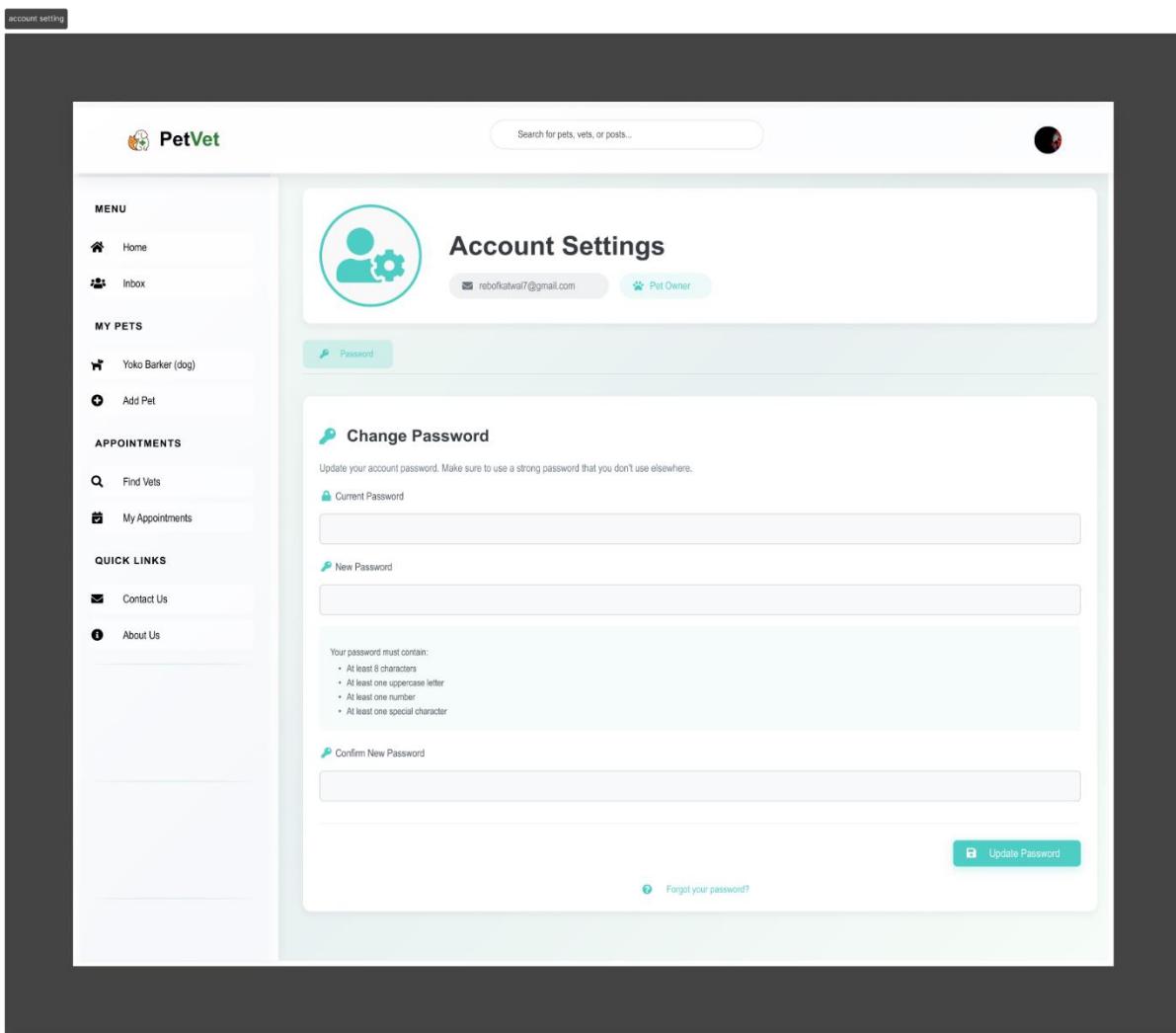


Figure 63: Main Design for Account settings

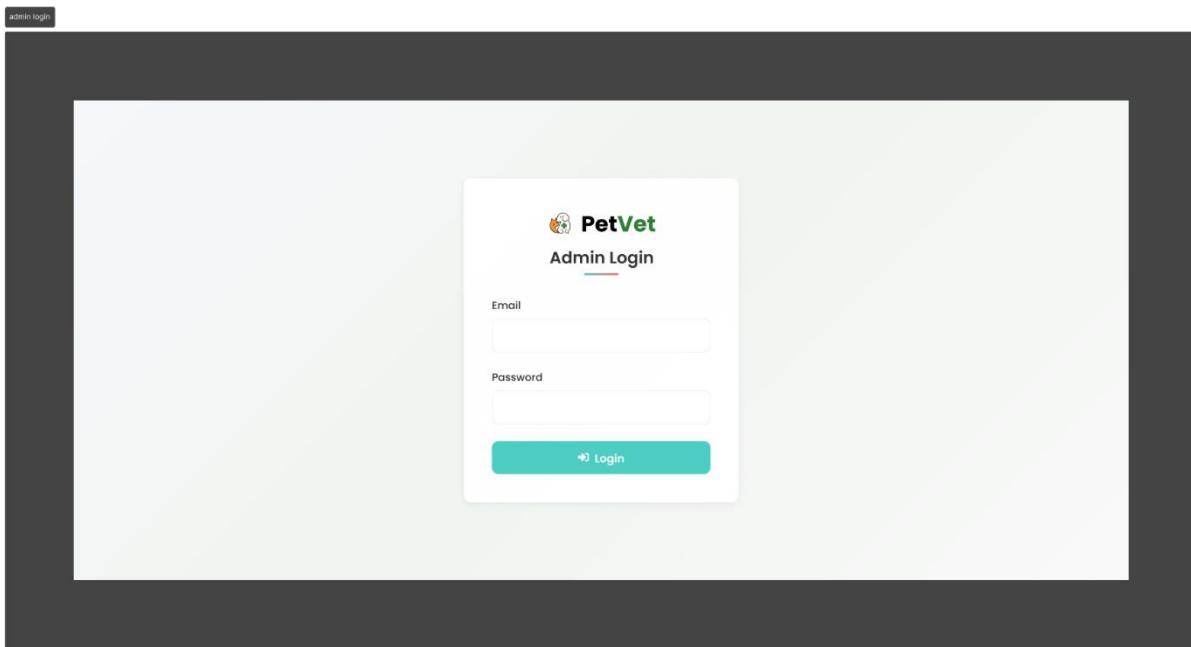


Figure 64: Main Design for Admin Login

The image shows the 'PetVet Admin' dashboard under the 'Vet Approvals' section. The left sidebar includes links for 'Dashboard', 'User Management', 'Post Management', 'Vet Approvals' (which is highlighted with a pink background), and 'Categories'. The main content area is titled 'Vet Approvals' and contains three summary boxes: 'Pending Approvals' (1 vet waiting for approval), 'Recently Approved' (1 vet approved in the last 7 days), and 'Recently Declined' (0 vets declined in the last 7 days). Below these is a table titled 'Pending Vet Approvals' showing details for Avishek Gautam. At the bottom, there are sections for 'Recently Approved Vets' (listing Coby Dennis) and 'Recently Declined Vets' (no data shown).

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
Avishek Gautam	avishekgautam6@gmail.com	None	None	0 years	Apr 22, 2025	<button>View</button> <button>Approve</button> <button>Decline</button>

Name	Email	Clinic Name	License Number	Approved At
Coby Dennis	testpet34@gmail.com	Paws & Whiskers Vet Clinic	NVC-NP-2024-89897	Apr 17, 2025

Figure 65: Main Design for Vet approval

3.6.2.2 Initial Development: Designs

Once the main Figma design was completed, I started designing and logic for registration and login. First, I made a basic flowchart to describe the process, then I made activity and sequence diagrams to describe the flow and validations. During this phase, I also worked on collaboration and defined the registration and login processes well.

3.6.2.2.1 Initial Development: Flowchart

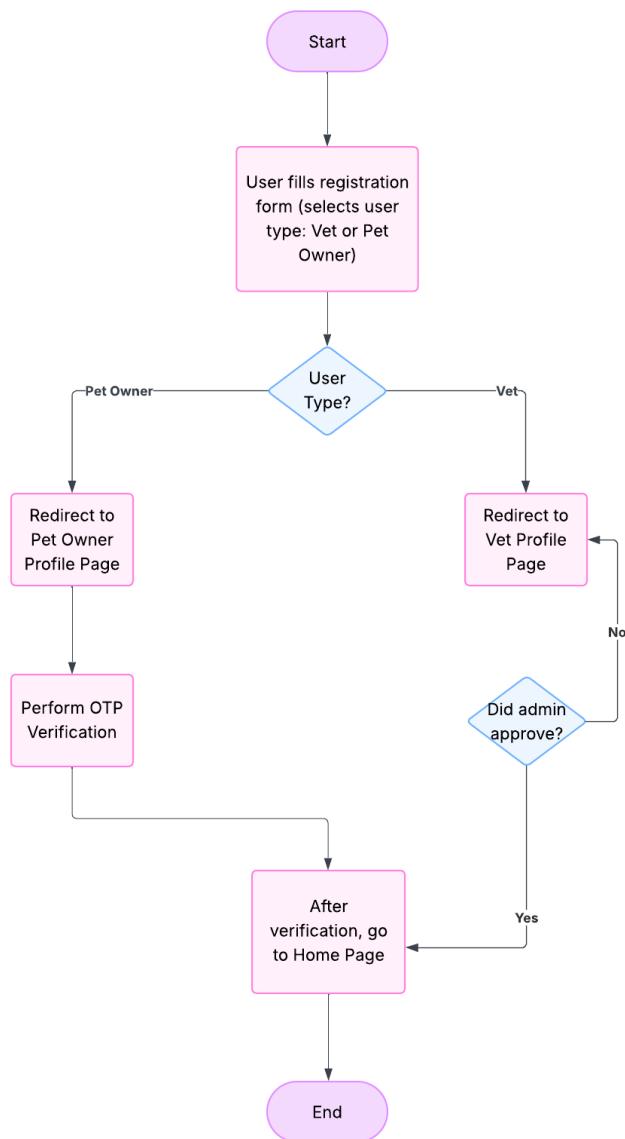


Figure 66: Flowchart for registration

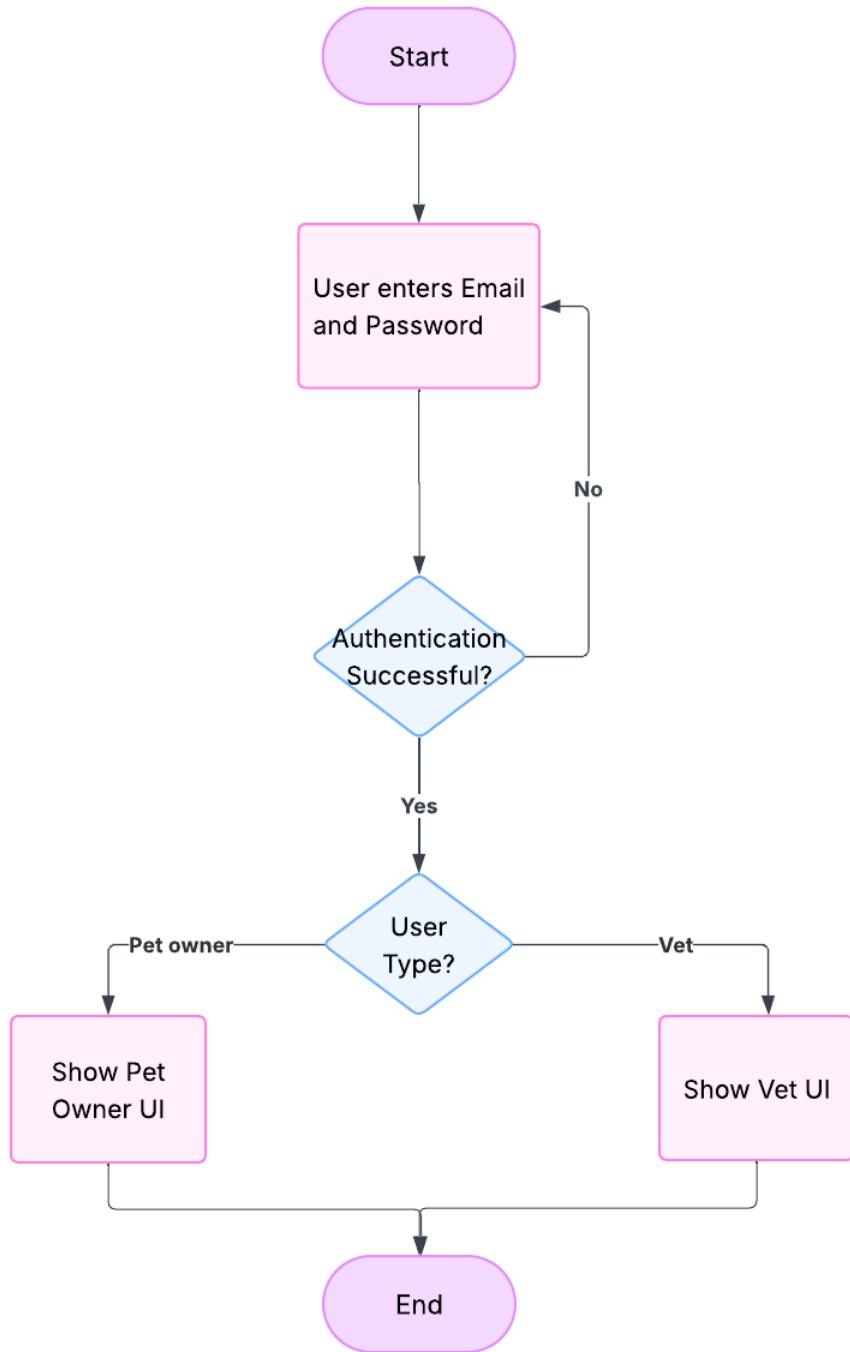


Figure 67: Flowchart for login

3.6.2.2.2 Initial Development: Sequence Diagrams

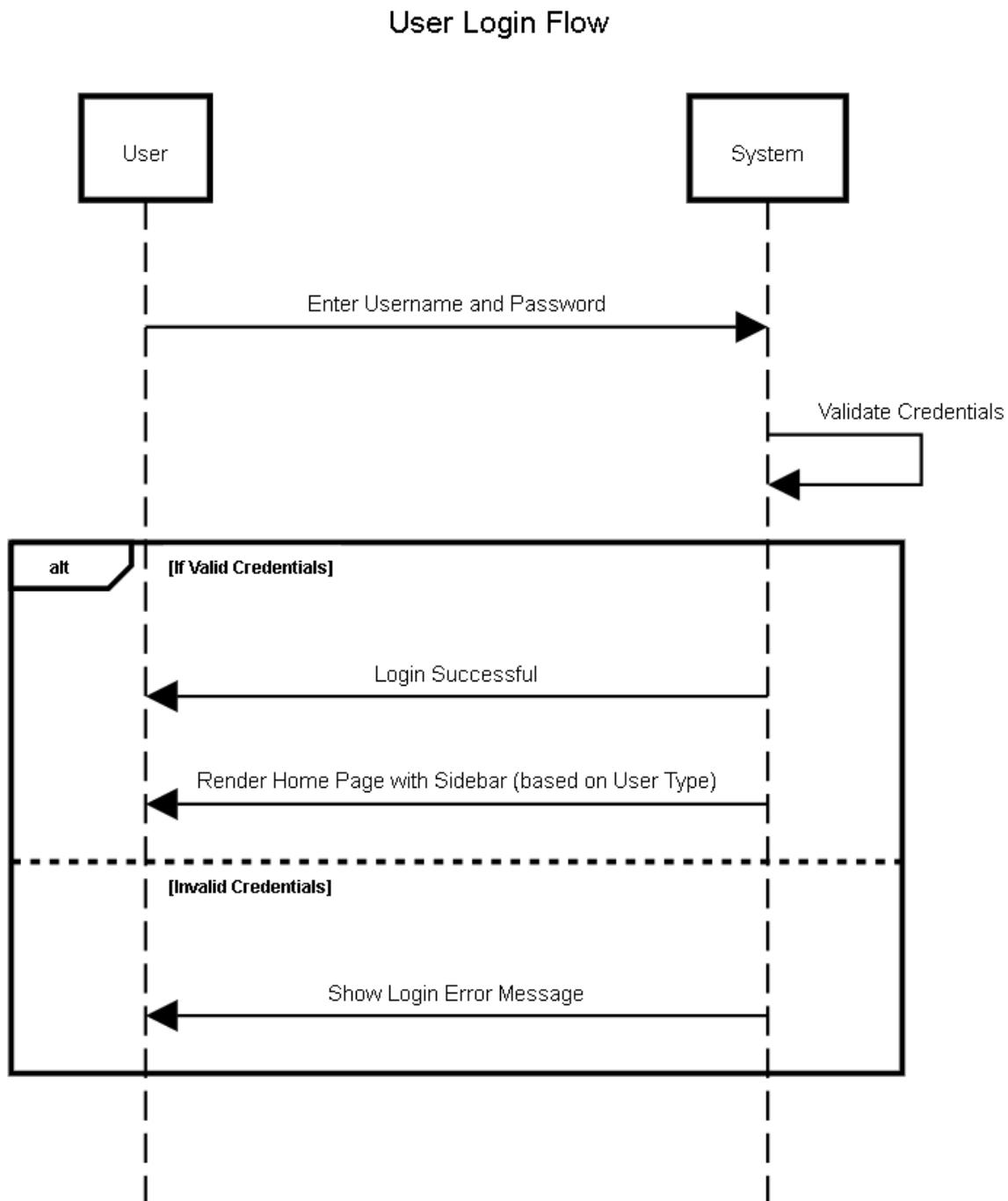


Figure 68: Sequence for login

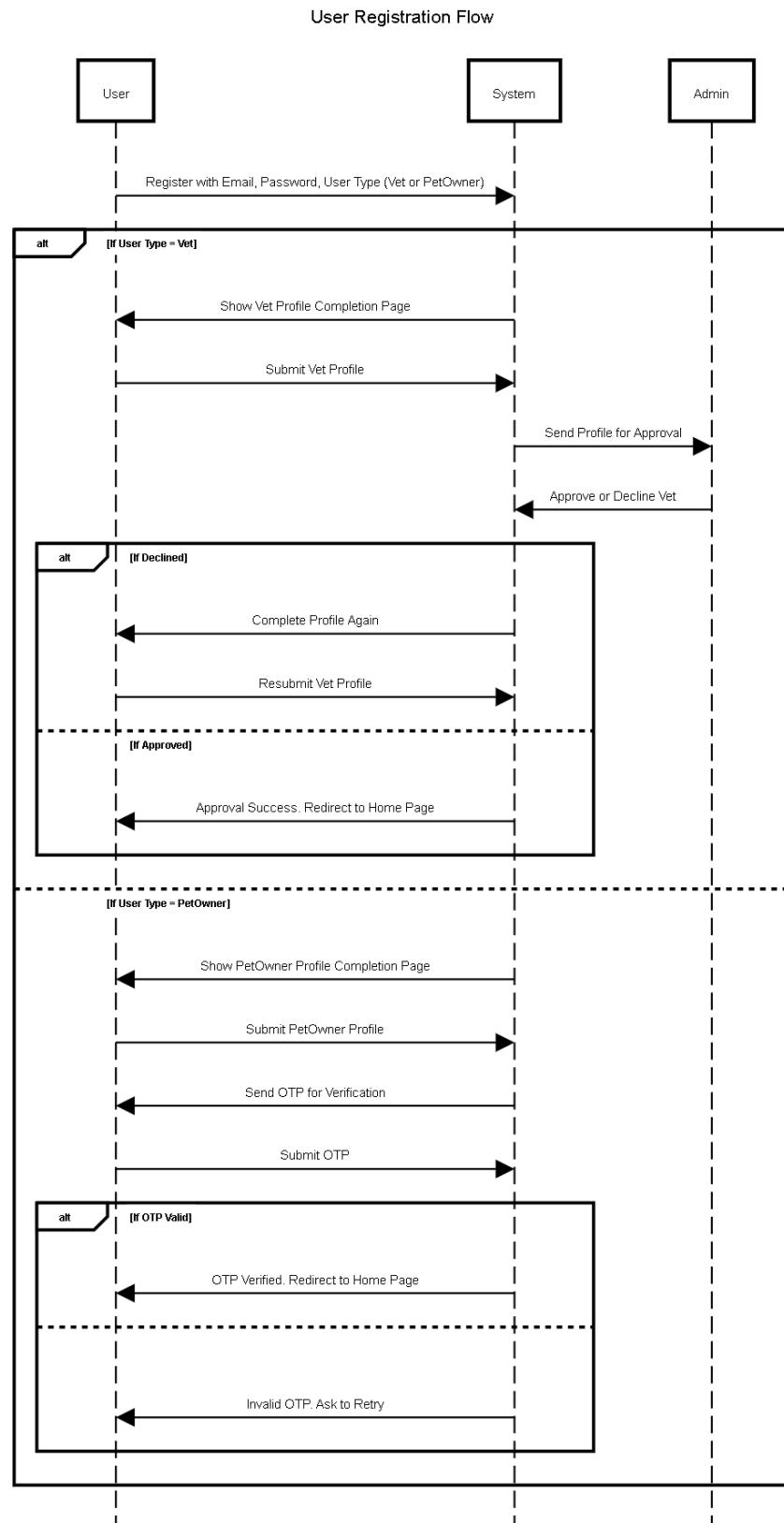


Figure 69: Sequence for registration

3.6.2.2.2 Initial Development: Sequence Diagrams

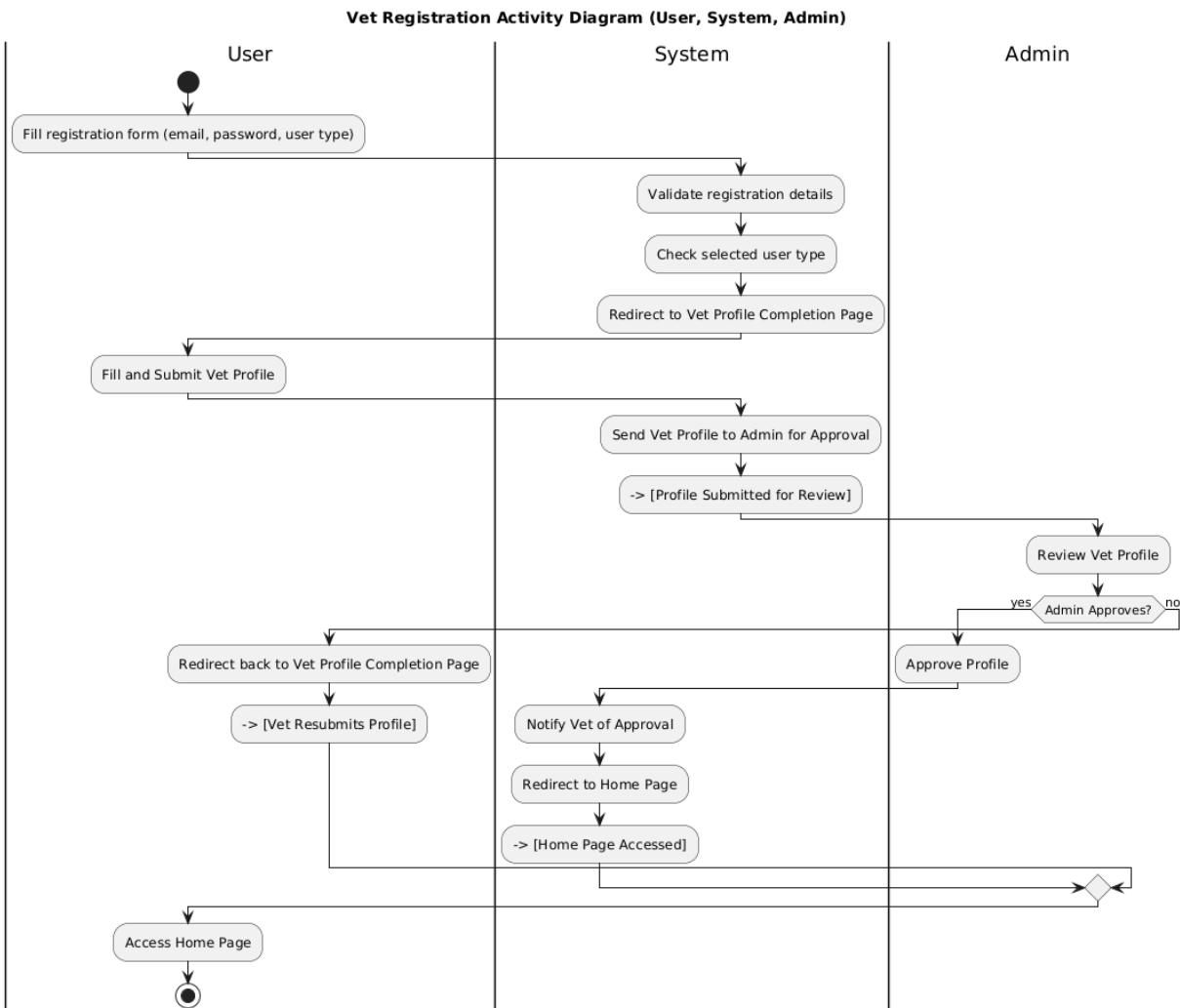


Figure 70: Activity diagram for vet registration

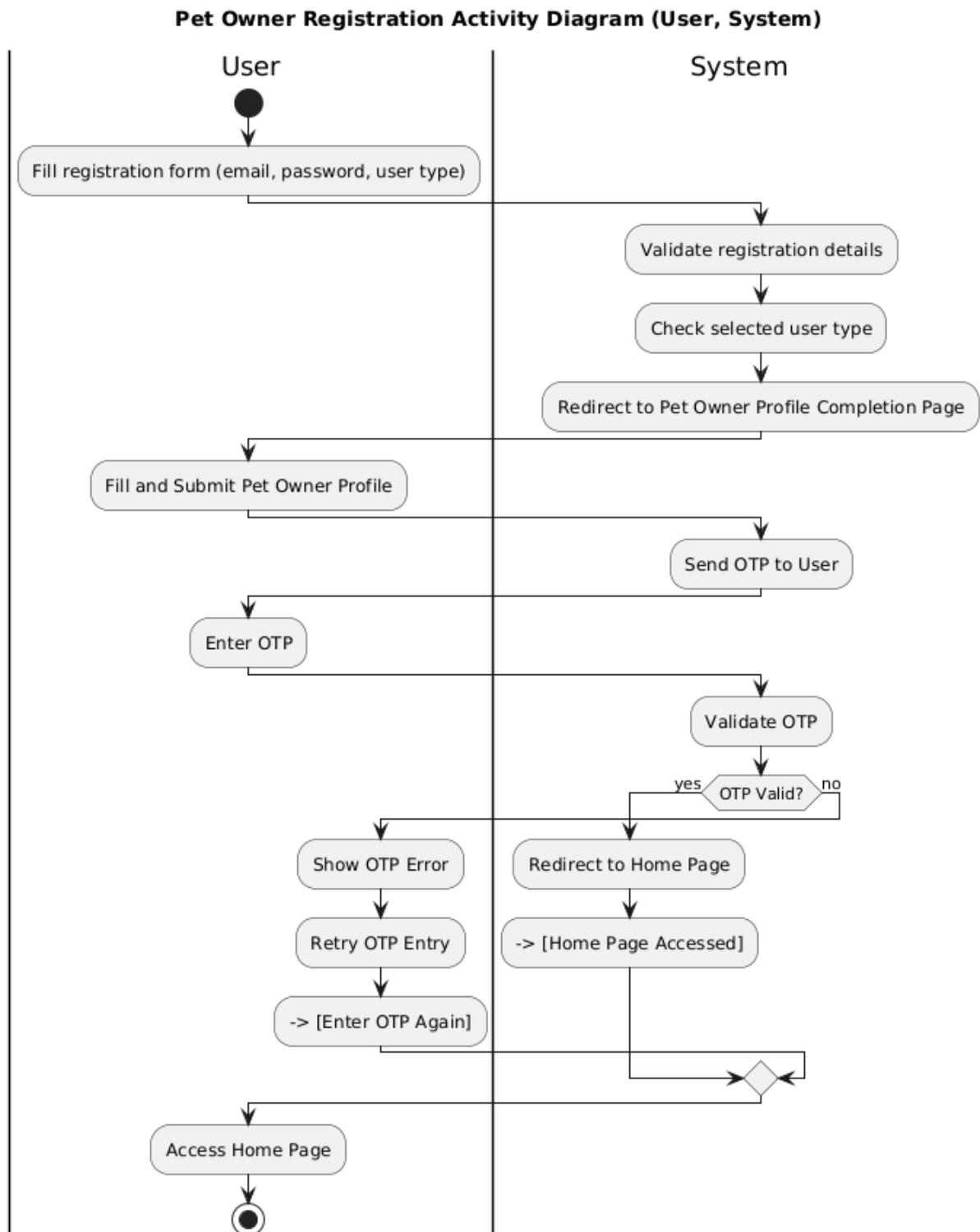


Figure 71: Activity diagram for pet owner registration

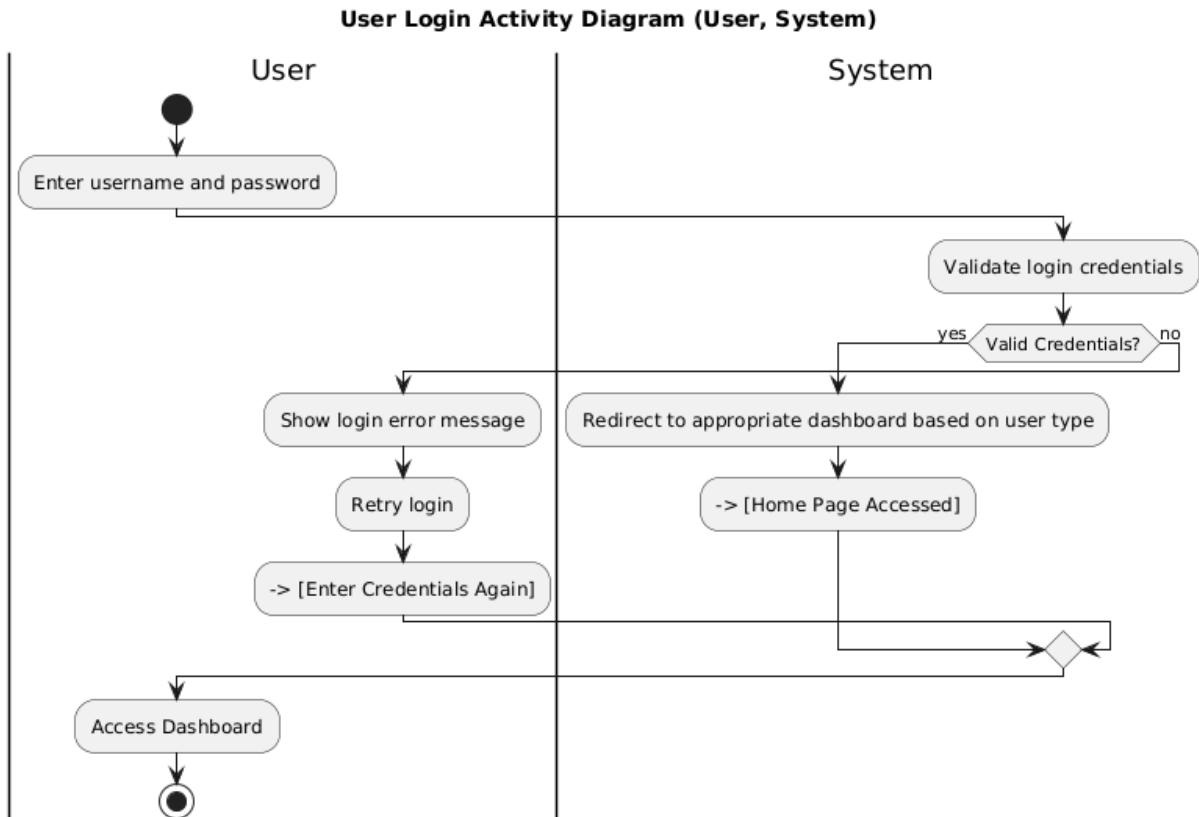


Figure 72: Activity diagram for login

3.6.2.2.4 Initial Development: Collaboration Diagram

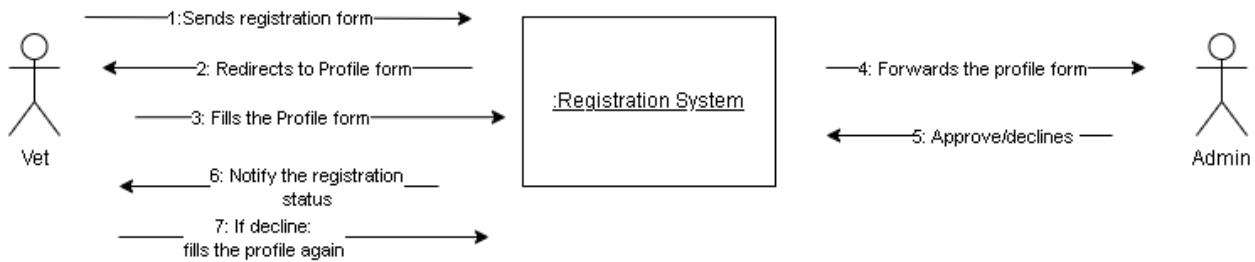


Figure 73: Collaboration for diagram for vet registration

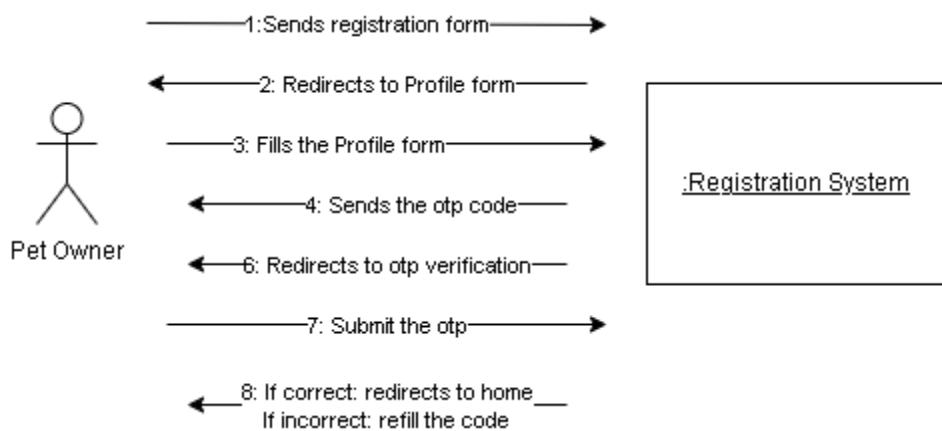


Figure 74: Collaboration diagram for pet owner registration

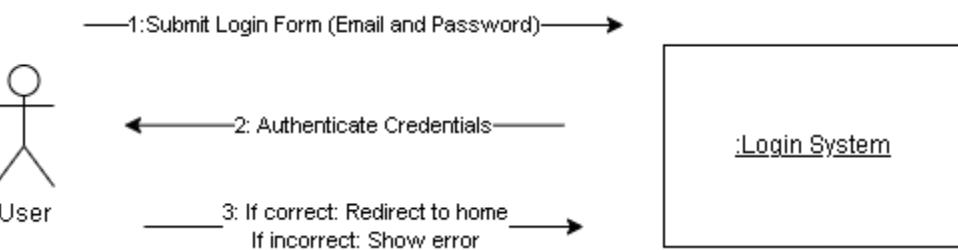


Figure 75: Collaboration diagram from login

3.6.2.3 Initial Development: Development

I have provided code snippets to illustrate the development process for key functionalities.

```
def Registerview(request):
    if request.user.is_authenticated:
        messages.warning(request, "You are already registered!")
        return redirect("coreFunctions:index")

    if request.method == "POST":
        # Extract data manually from request.POST
        full_name = request.POST.get("full_name")
        email = request.POST.get("email")
        username = request.POST.get("username")
        phone = request.POST.get("phone")
        gender = request.POST.get("gender")
        user_type = request.POST.get("user_type")
        password1 = request.POST.get("password1")
        password2 = request.POST.get("password2")
```

Figure 76 Code Snippet – User Registration Logic in Initial Development

```

def LoginView(request):
    if request.user.is_authenticated:
        return redirect("coreFunctions:index")

    if request.method == "POST":
        email = request.POST.get("email")
        password = request.POST.get("password")

        user = authenticate(request, email=email, password=password)

        if user is not None:
            # If authentication is successful
            login(request, user)

            # Redirect to complete-profile if the profile is incomplete
            if not user.profile_completed:
                # print("redirect bhayo")
                return redirect("complete-profile")

            return redirect("coreFunctions:index")
        else:
            # If authentication fails
            messages.error(request, "Invalid username or password")
            return redirect("authUser:loginUser")

    return render(request, "authUser/register.html")

```

Figure 77 Code Snippet – User Authentication Logic in Initial Development

```

def LogoutView(request):
    logout(request)
    messages.success(request, "You have successfully logged out.")
    return redirect("authUser:loginUser")

```

Figure 78 Code Snippet – Logout logic in Initial Development

```

def complete_profile(request):
    if not request.user.is_authenticated:
        messages.error(request, "You need to log in to complete your profile.")
        return redirect('authUser:loginUser')

    try:
        if request.user.user_type == 'vet':
            profile = VetProfile.objects.get(user=request.user)
            profile_type = 'vet'
        elif request.user.user_type == 'pet_owner':
            profile = PetOwnerProfile.objects.get(user=request.user)
            profile_type = 'pet_owner'
        else:
            messages.error(request, "Invalid user type.")
            return redirect('coreFunctions:index')
    except (VetProfile.DoesNotExist, PetOwnerProfile.DoesNotExist):
        messages.error(request, "Profile not found. Please contact support.")
        return redirect('coreFunctions:index')

```

Figure 79 Code Snippet – Logout Logic in Initial Development

```

def profile_verification_in_progress(request):
    # Check if the user is a vet and their profile is completed but not verified yet
    if request.user.user_type == 'vet' and request.user.profile_completed and not request.user.status_verification:
        return render(request, 'authUser/profile_verification_in_progress.html')
    elif request.user.user_type == 'pet_owner' and request.user.profile_completed and not request.user.status_verification:
        return render(request, 'authUser/verify_otp.html')
    # If the profile is verified, redirect to the feed page
    elif request.user.user_type == 'vet' and request.user.status_verification:
        return redirect("coreFunctions:index")
    else:
        # If the user is not a vet or verification is not needed
        return redirect("coreFunctions:index")

```

Figure 80 Code Snippet – Waiting page Logic in Initial Development

```
def verify_otp(request):
    # If user is already verified, redirect them
    if request.user.status_verification:
        return redirect("coreFunctions:index")

    if request.method == 'POST':
        entered_otp = request.POST.get('otp')

        if not entered_otp:
            messages.error(request, "Please enter the OTP")
            return render(request, 'authUser/verify_otp.html')

        # Compare the entered OTP
        if request.user.otp == entered_otp:
            request.user.status_verification = True
            request.user.otp = None # Clear OTP after verification
            request.user.save()
            messages.success(request, "Your email has been verified successfully!")
            return redirect("coreFunctions:index")
        else:
            messages.error(request, "Invalid OTP. Please try again.")

    # Handle GET request
    return render(request, 'authUser/verify_otp.html')
```

Figure 81 Code Snippet – Verify OTP Logic in Initial Development

3.6.2.4 Initial Development: Testing

As part of this prototype, testing was conducted on core authentication and vet management features, including registration, login, logout, vet approval, and vet decline. These tests are documented in the testing section under test numbers AUTH1, AUTH2, AUTH3, Admin1 and Admin2.

4.2.1 Test: Auth1 Registration Functionality

4.2.2 Test: Auth2 Login Functionality

4.2.3 Test: Auth3 Admin Login Functionality

4.2.4 Test: Admin1 Vet Approval Functionality

4.2.5 Test: Admin2 Vet Decline Functionality

3.6.2.5 Initial Development: User Feedback

The registration and onboarding process received positive feedback, with 83% of users finding it easy. Profile setup clarity scored 4.4, and 86% of pet owners had a smooth OTP verification experience. The vet approval page was rated 4.7, with 81% of users finding it clear. Overall, 86% of users were satisfied with the registration and onboarding flow, indicating a user-friendly system.

Below is the link that leads to the actual charts and a sample filled user feedback form:

[*7.6.1 User Feedback Form for Initial Development Results*](#)

[*7.6.2 User Feedback Form for Initial Development Sample Answer*](#)

3.6.3 Prototype/Iteration 1

Blogging capabilities formed the main focus of Prototype 1 development. It is vital to have the ability to create and interact with blog posts for enabling an informed and engaged community. This functionality promotes public understanding between proper and improper approaches associated with animal healthcare and veterinary services. Features for creating posts and editing them as well as features for commenting and replying and liking posts were implemented during this phase.

3.6.3.1 Prototype/Iteration 1: Figma Design

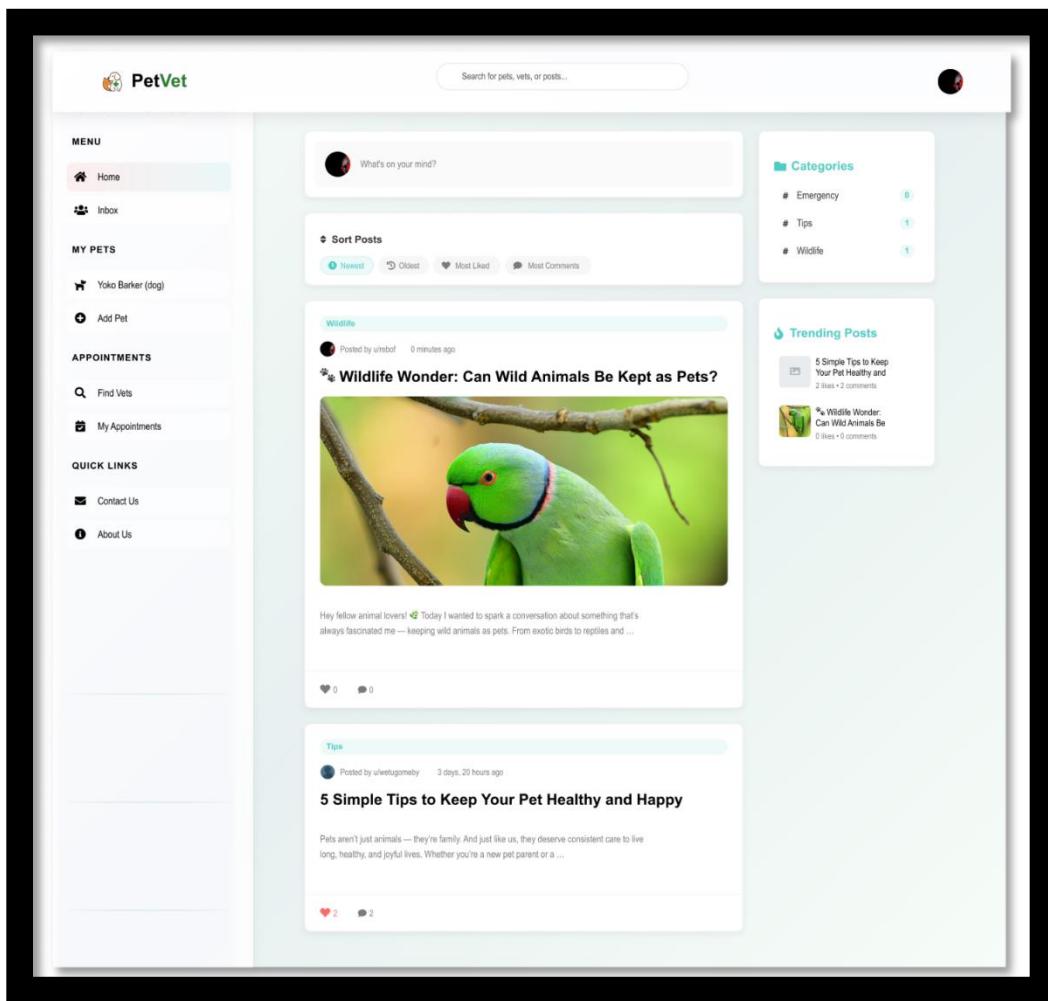


Figure 82: Main Design for Home

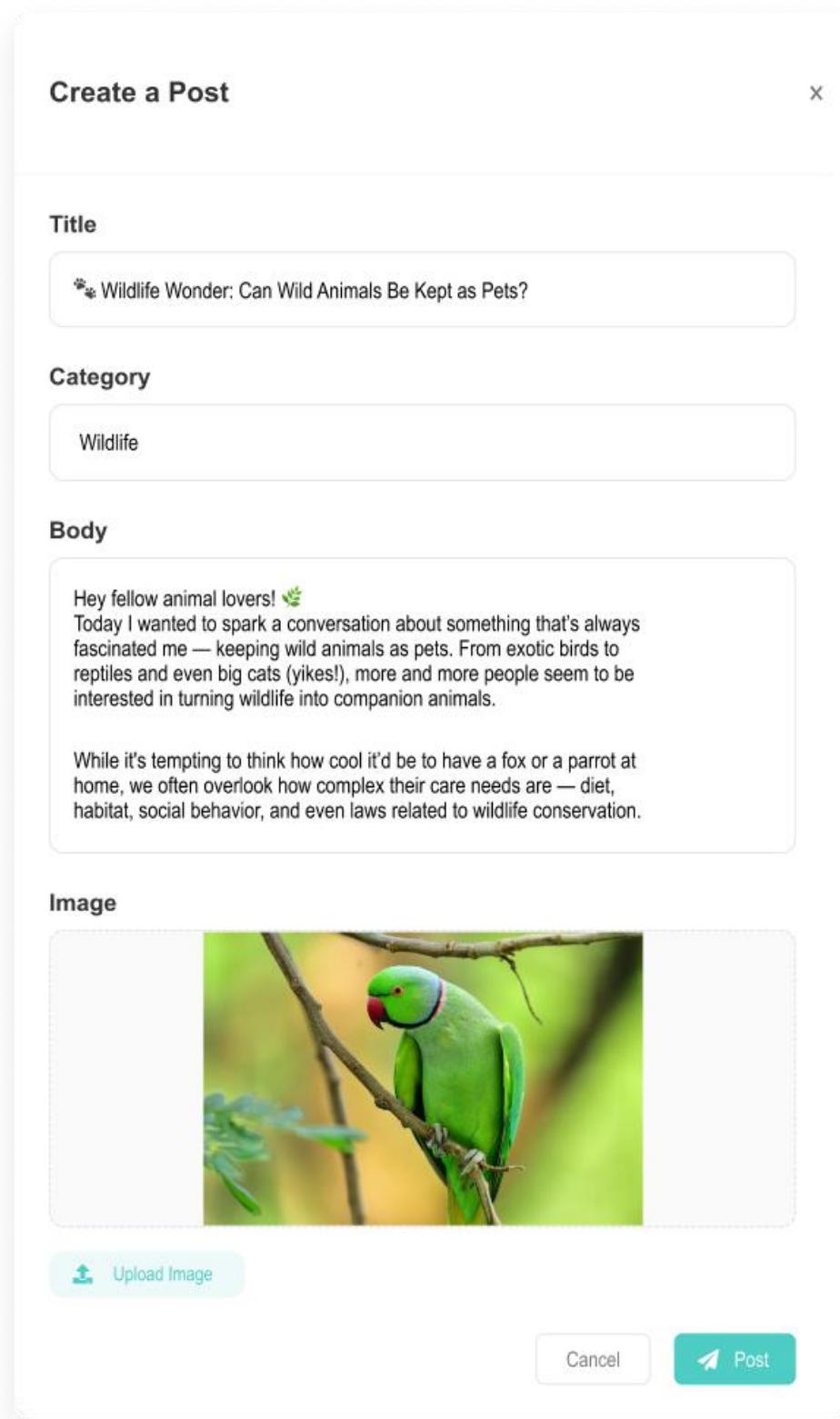


Figure 83: Main Design for Create modal

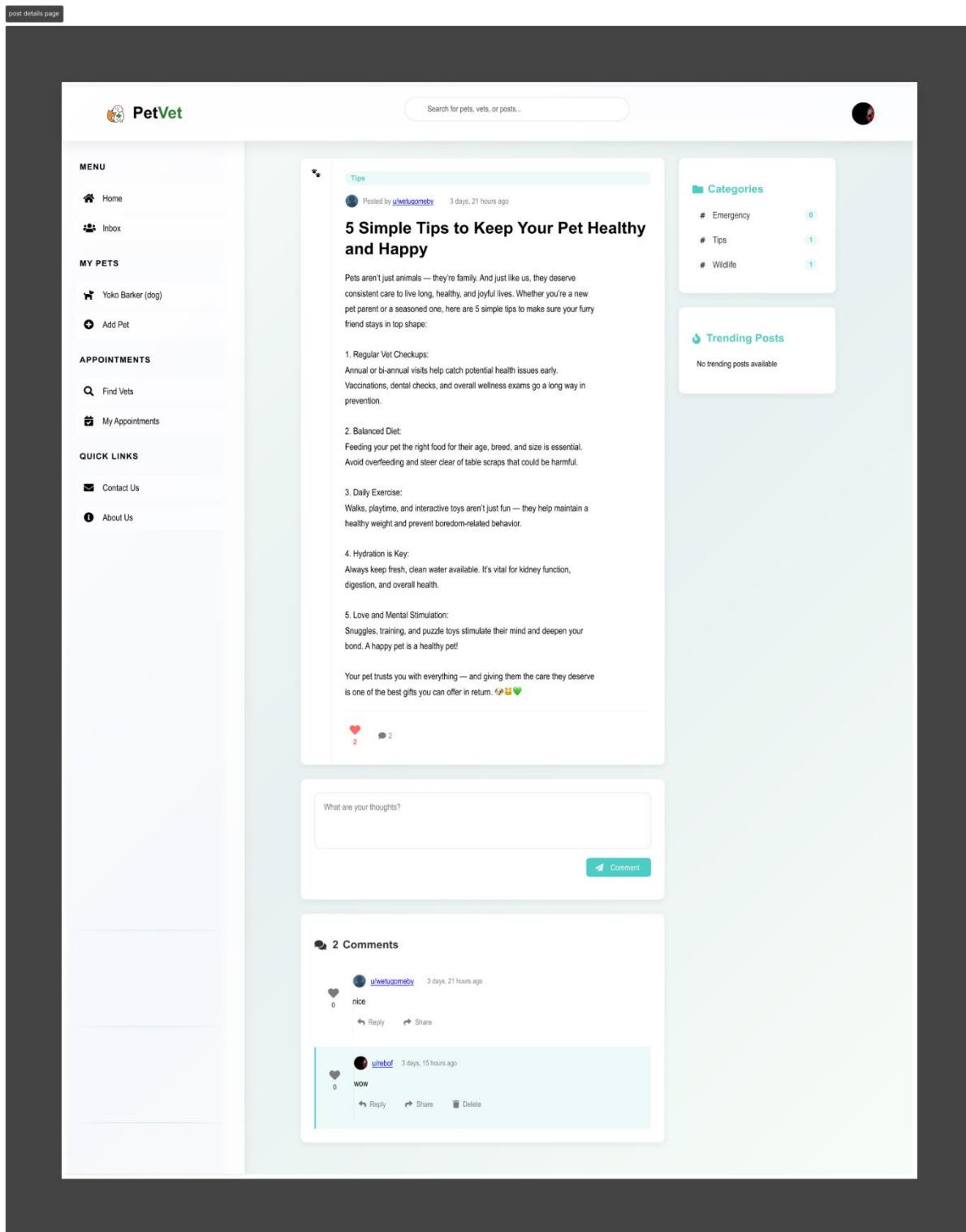


Figure 84: Main Design for Post details

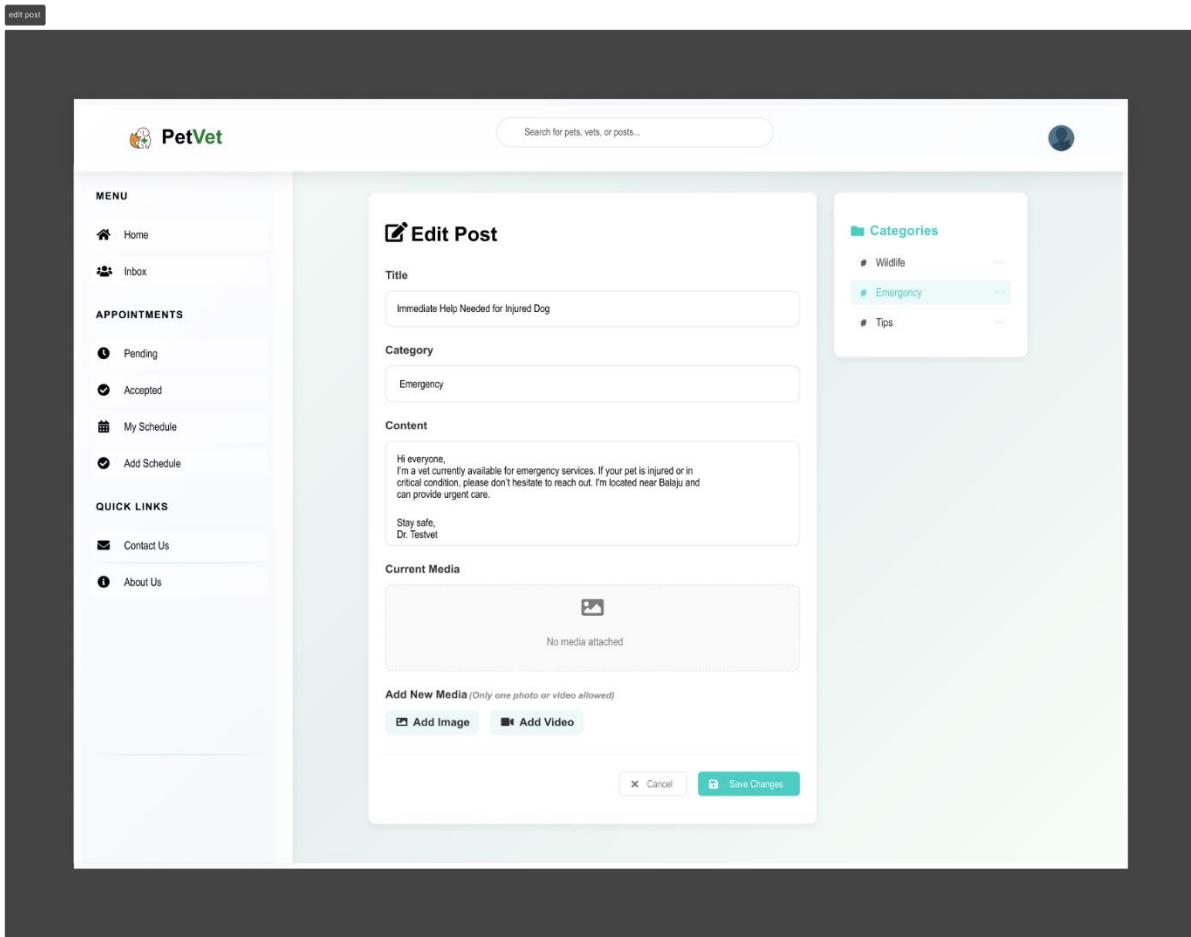


Figure 85: Main Design for Edit post

3.6.3.2 Prototype/Iteration 1: Designs

3.6.3.2.1 Prototype/Iteration 1: Flowchart Diagrams

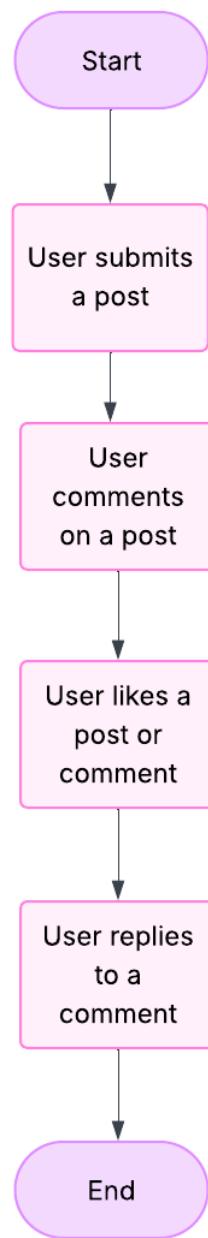


Figure 86: Flowchart for Blog posts

3.6.3.2.2 Prototype/Iteration 1: Activity Diagrams

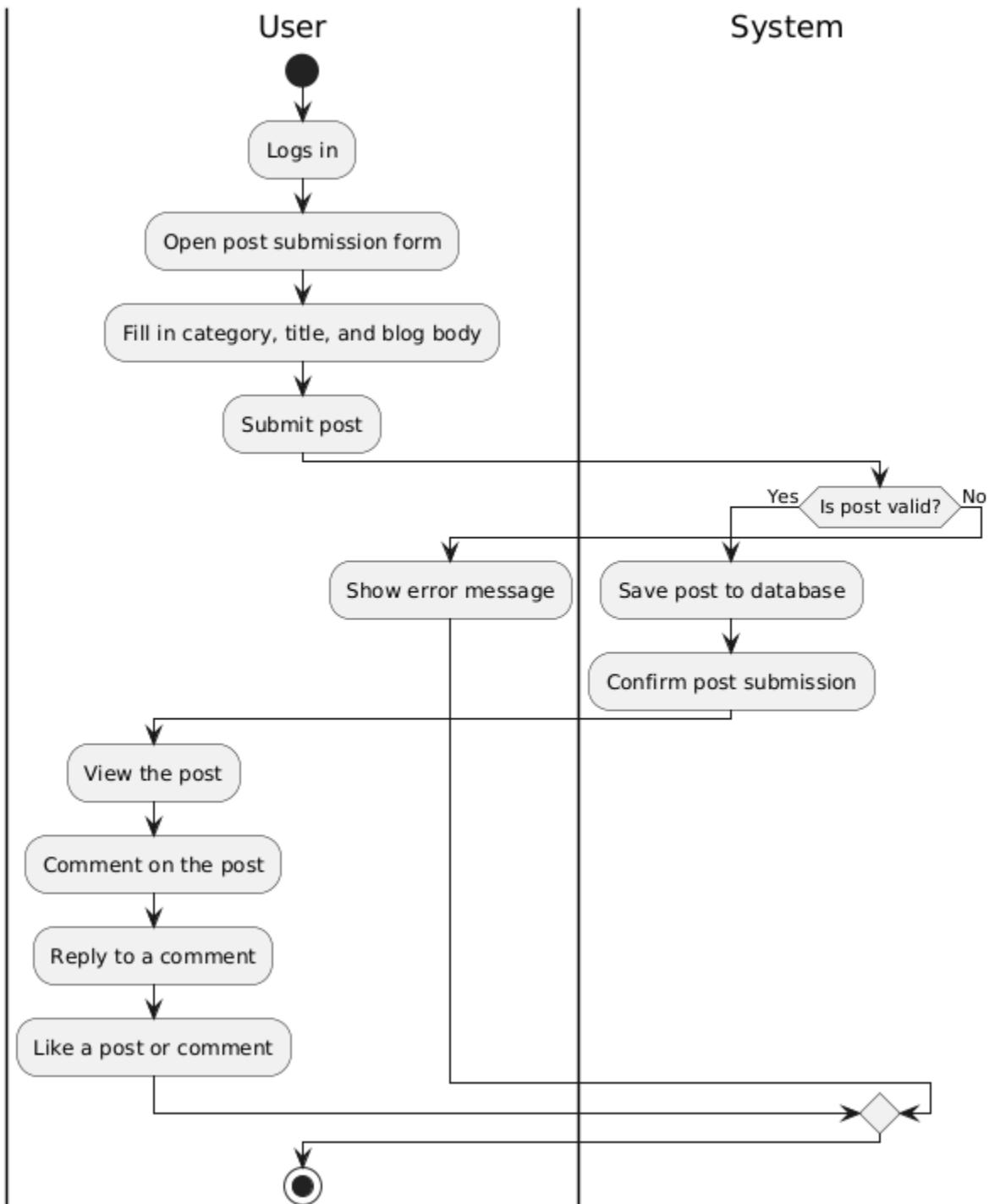


Figure 87: Activity diagram for Blogs posts

3.6.3.2.3 Prototype/Iteration 1: Sequence Diagrams

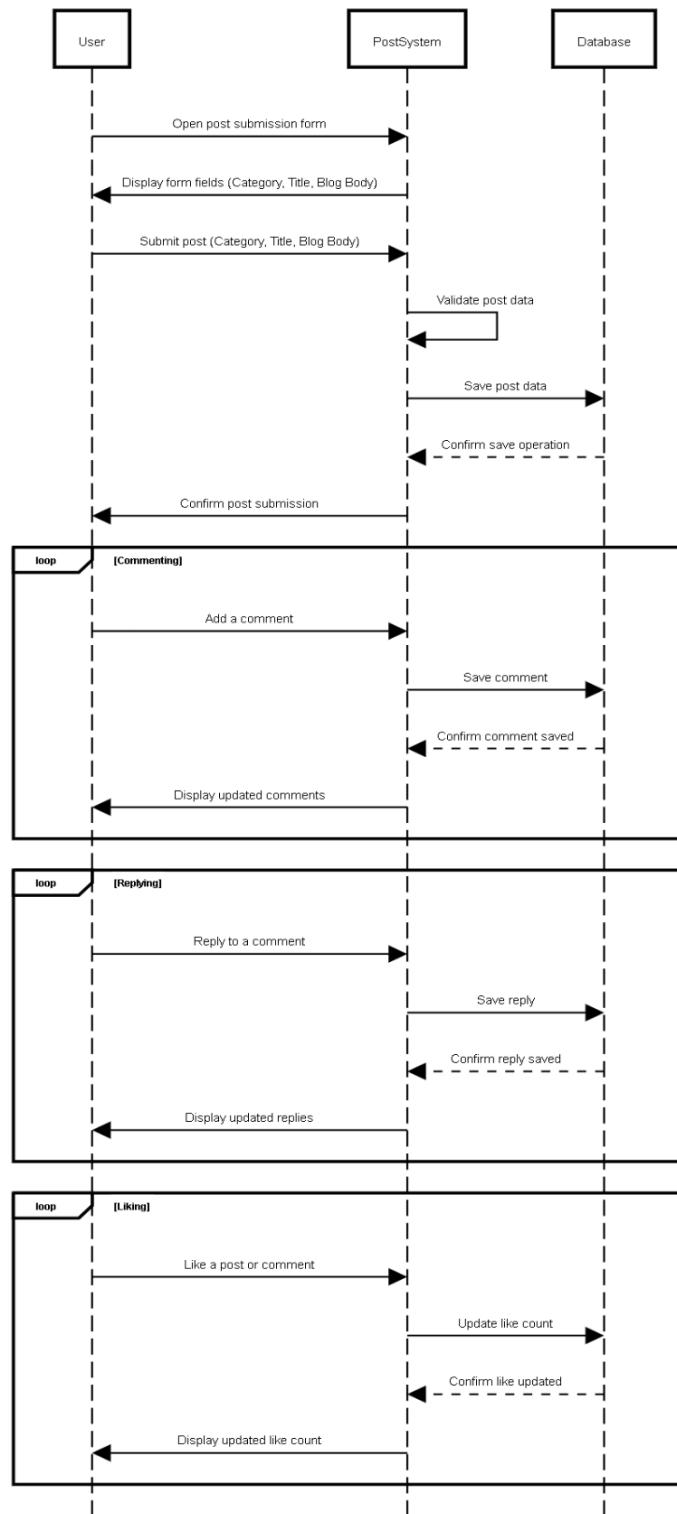


Figure 88: Sequence diagram for Blog posts

3.6.3.2.4 Prototype/Iteration 1: Collaboration Diagrams

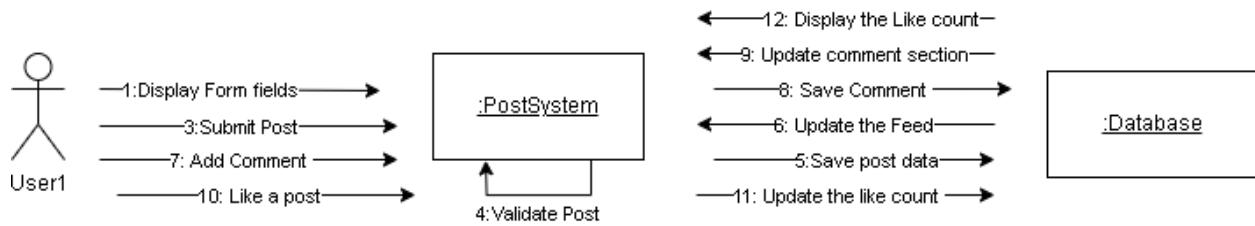


Figure 89: Collaboration diagram for Blog posts

3.6.3.3 Prototype/Iteration 1: Development

I have provided code snippets to illustrate the development process for key functionalities.

```
@login_required(login_url='authUser:register')
def index(request, category_slug=None):
    # Get sort parameter (default to 'newest')
    sort_by = request.GET.get('sort', 'newest')

    # Get date filter parameter
    date_filter = request.GET.get('date')

    # Start with all posts
    posts_query = Post.objects.filter(active=True)

    # Apply category filter if provided
    selected_category = None
    if category_slug:
        selected_category = get_object_or_404(Category, slug=category_slug)
        posts_query = posts_query.filter(category=selected_category)

    # Apply sorting
    if sort_by == 'likes':
        # Sort by most likes
        posts = posts_query.annotate(like_count=Count('likes')).order_by('-like_count')
    elif sort_by == 'comments':
        # Sort by most comments
        posts = posts_query.annotate(comment_count=Count('comments')).order_by('-comment_count')
    elif sort_by == 'oldest':
        # Sort by oldest first
        posts = posts_query.order_by('date')
    else:
        # Default: sort by newest
        posts = posts_query.order_by('-date')

    # Get all categories with post count
    categories = Category.objects.annotate(post_count=Count('post'))
```

Figure 90 Code Snippet – Home page Logic in Prototype 1

```
@login_required
def like_post(request, post_id):
    """Handle post liking via AJAX"""
    if request.method == 'POST':
        try:
            post = Post.objects.get(id=post_id)
            user = request.user

            # Toggle like
            if user in post.likes.all():
                post.likes.remove(user)
                liked = False
            else:
                post.likes.add(user)
                liked = True

            return JsonResponse({
                'status': 'success',
                'likes_count': post.likes.count(),
                'liked': liked
            })
        except Post.DoesNotExist:
            return JsonResponse({'status': 'error', 'message': 'Post not found'}, status=404)

    return JsonResponse({'status': 'error', 'message': 'Invalid request'}, status=400)
```

Figure 91 Code Snippet – Liking Logic in Prototype 1

```

def create_post(request):
    body = request.POST.get('body')
    category_id = request.POST.get('category')
    image = request.FILES.get('image')

    # Validate required fields
    if not title or not body or not category_id:
        messages.error(request, 'Please fill in all required fields')
        return redirect('coreFunctions:index')

    # Get category
    try:
        category = Category.objects.get(id=category_id)
    except Category.DoesNotExist:
        messages.error(request, 'Invalid category')
        return redirect(['coreFunctions:index'])

    # Create slug from title
    slug = slugify(title)
    # Check if slug already exists, if so, add a unique identifier
    if Post.objects.filter(slug=slug).exists():
        slug = f'{slug}-{shortuuid.uuid()[:6]}'

    # Create post
    post = Post.objects.create(
        title=title,
        body=body,
        category=category,
        user=request.user,
        slug=slug
    )
    # Add image if provided
    if image:
        post.image = image
        post.save()

    messages.success(request, 'Post created successfully')
    return redirect('coreFunctions:post-detail', slug=post.slug)

# If not POST, redirect to feed
return redirect('coreFunctions:index')

```

Figure 92 Code Snippet – Create post Logic in Prototype 1

```

    active=True
).annotate(
    like_count=Count('likes'),
    comment_count=Count('comments'),
    engagement_score=ExpressionWrapper(
        Count('likes') + Count('comments'),
        output_field=IntegerField()
    )
).order_by('-engagement_score')[ :10] # top 10 by total engagement

# Pick 5 random ones from top 10
trending_posts = sample(list(posts_with_engagement), min(5, len(posts_with_engagement)))

# Active comments and their active replies
comments = post.comments.filter(active=True).prefetch_related(
    models.Prefetch(
        'replies',
        queryset=ReplyComment.objects.filter(active=True),
        to_attr='active_replies'
    )
)

context = {
    'post': post,
    'categories': categories,
    'trending_posts': trending_posts,
    'comments': comments,
}
return render(request, 'coreFunctions/post_detail.html', context)

```

Figure 93 Code Snippet – Post details page Logic in Prototype 1

```
@login_required
def delete_post(request, post_id):
    if request.method != 'POST':
        return JsonResponse({
            'success': False,
            'error': 'Invalid request method'
        }, status=400, content_type='application/json')

    try:
        post = Post.objects.get(id=post_id)
        if post.user != request.user:
            return JsonResponse({
                'success': False,
                'error': 'Post not found or unauthorized'
            }, status=404, content_type='application/json')

        post.delete()
        return JsonResponse({'success': True}, content_type='application/json')

    except Post.DoesNotExist:
        return JsonResponse({
            'success': False,
            'error': 'Post not found'
        }, status=404, content_type='application/json')
```

Figure 94 Code Snippet – Delete post Logic in Prototype 1

3.6.3.4 Prototype/Iteration 1: Testing

In Prototype 1, the focus was on the blogging feature. Key functionalities such as liking, commenting, replying, creating a post, and editing were unit tested. These tests are documented in the testing section under the test numbers PO1, PO2, PO3, and PO4.

4.2.18 Test: Po1 Post Creation and Deletion Functionality

4.2.19 Test: Po2 Post Updating and Liking Functionality

4.2.20 Test: Po3 Liking, Deleting and Adding Comments Functionality

4.2.21 Test: Po4 Reply Creation and Deletion Functionality

3.6.3.5 Prototype/Iteration 1: User Feedback

The blogging feature on the PetVet platform received positive feedback, with users finding post creation, commenting, and interacting with posts easy and intuitive. The interface was praised for being clear and user-friendly, and the blogging feature was considered important and engaging. Suggestions for improvement included enhancing the UI, ensuring mobile responsiveness, and using high-quality images. Overall, users expressed strong satisfaction, with the feature being seen as a valuable addition to the platform.

Below is the link that leads to the actual charts and a sample filled user feedback form:

[7.6.3 User Feedback Form for Prototype 1 Results](#)

[7.6.4 User Feedback Form for Prototype 1 Sample Answer](#)

3.6.4 Prototype/Iteration 2

The development of the application's core functionalities was prototyped in Prototype 2. At this point the system was fully worked out and the appointment system was central. The structure was carefully put together to fulfill the distinct functionalities, namely one for the vets whilst the other for pet owners. The system was developed on both sides of the system: vet and pet owner, to perform their respective tasks like booking, accepting, rejecting, canceling appointments, managing schedules and sending notifications.

3.6.4.1 Prototype/Iteration 2: Figma Design

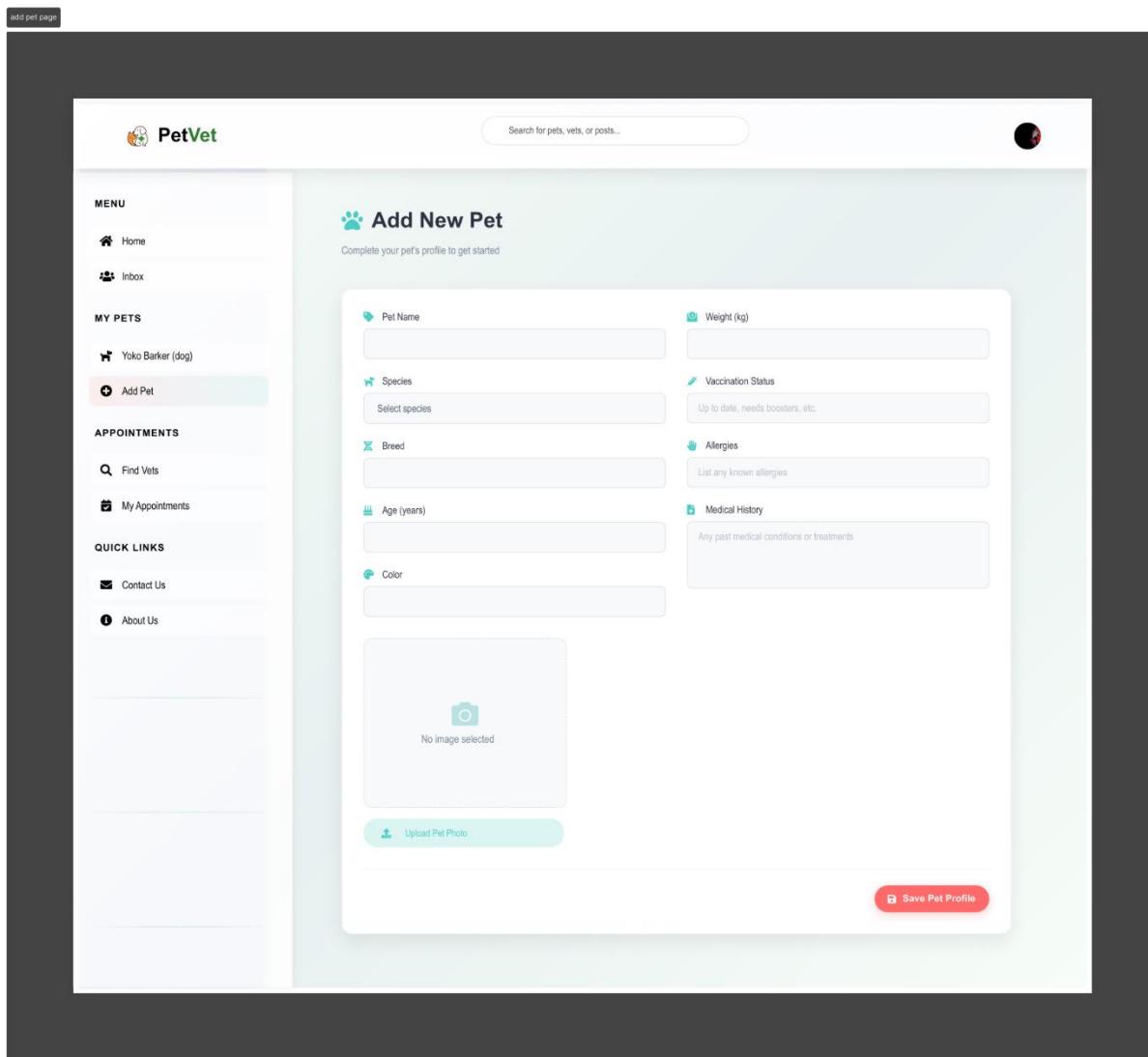


Figure 95: Main Design for Add pet

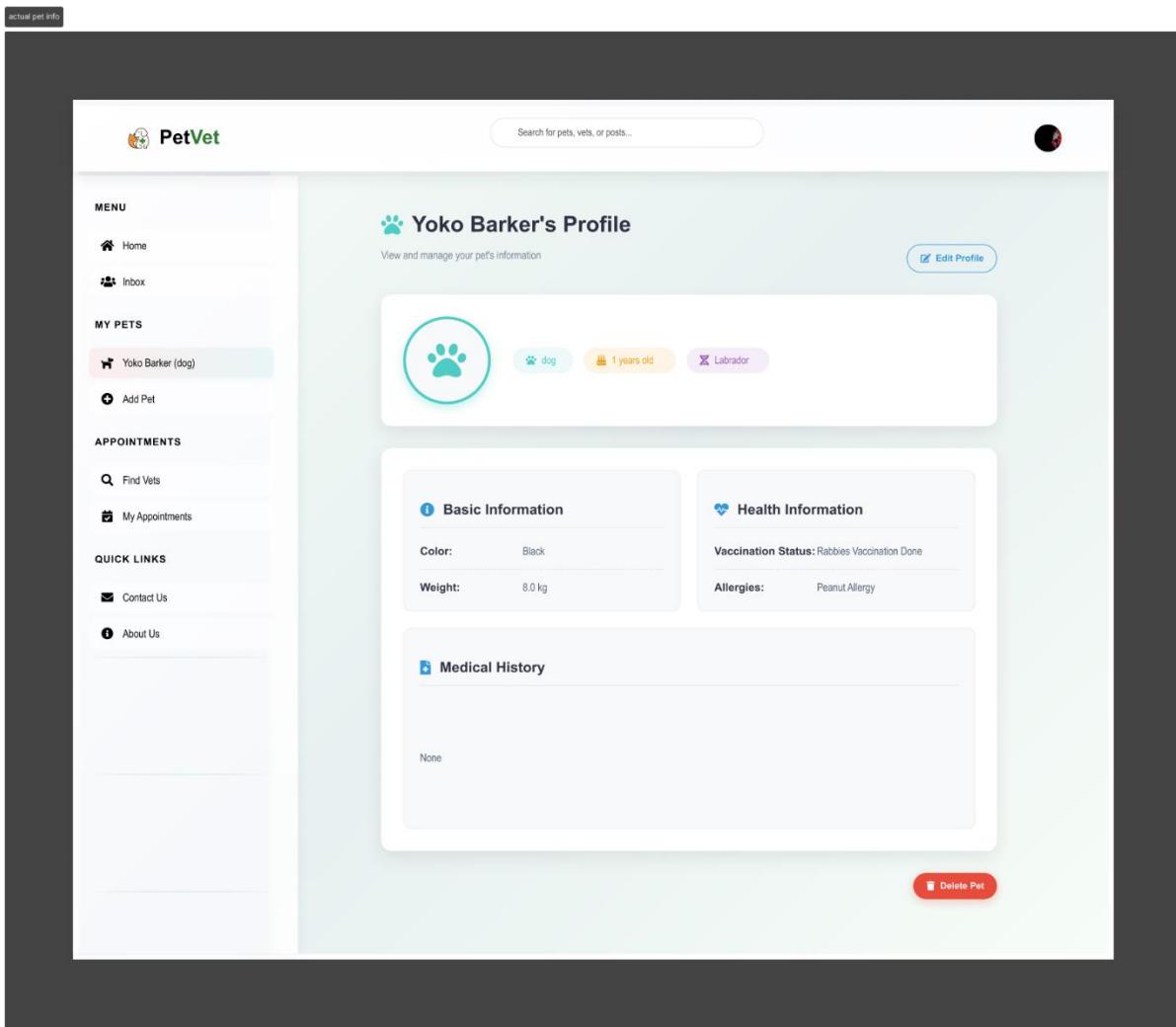


Figure 96: Main Design for Pet profile

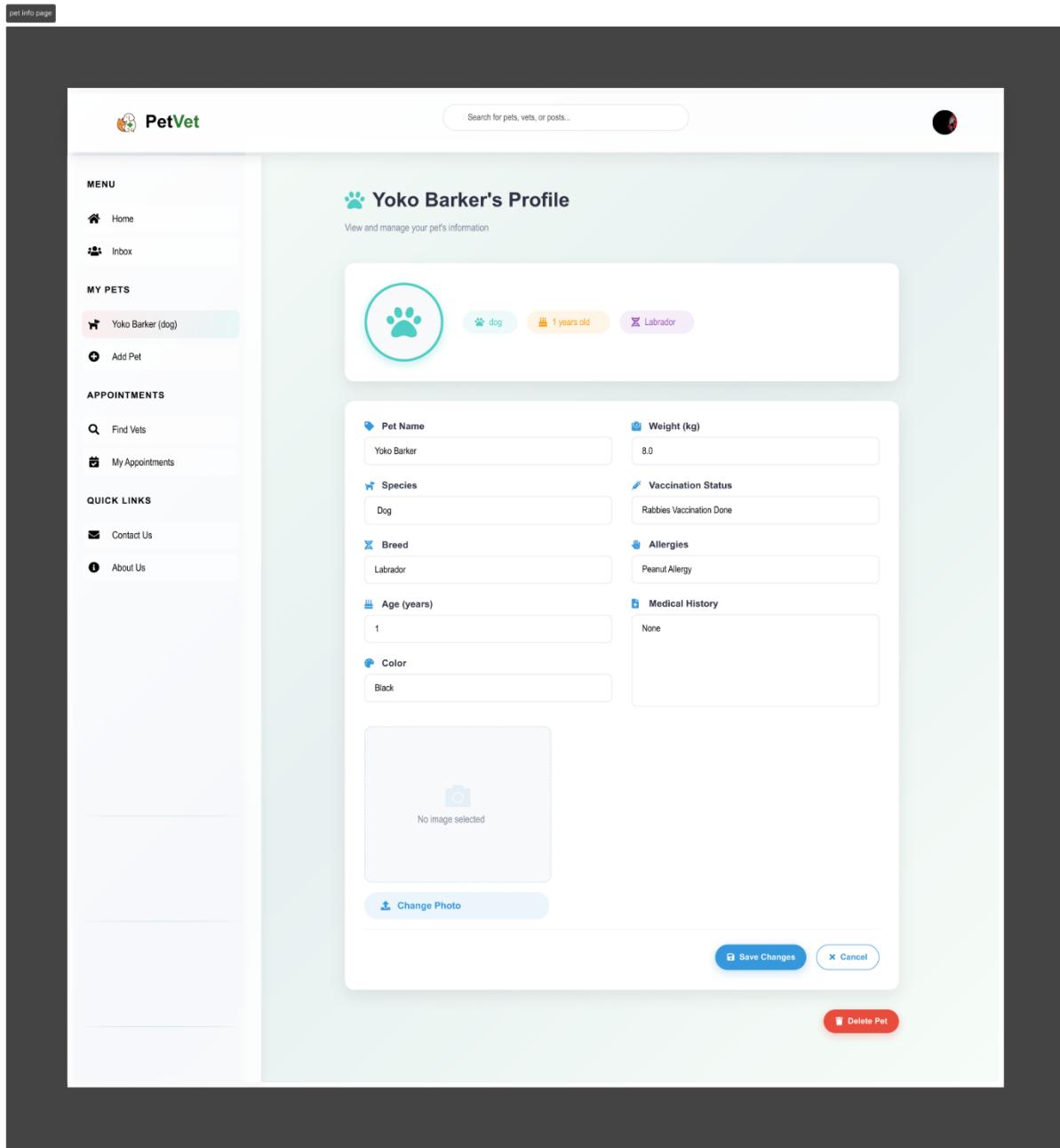


Figure 97: Main Design for Edit pet profile

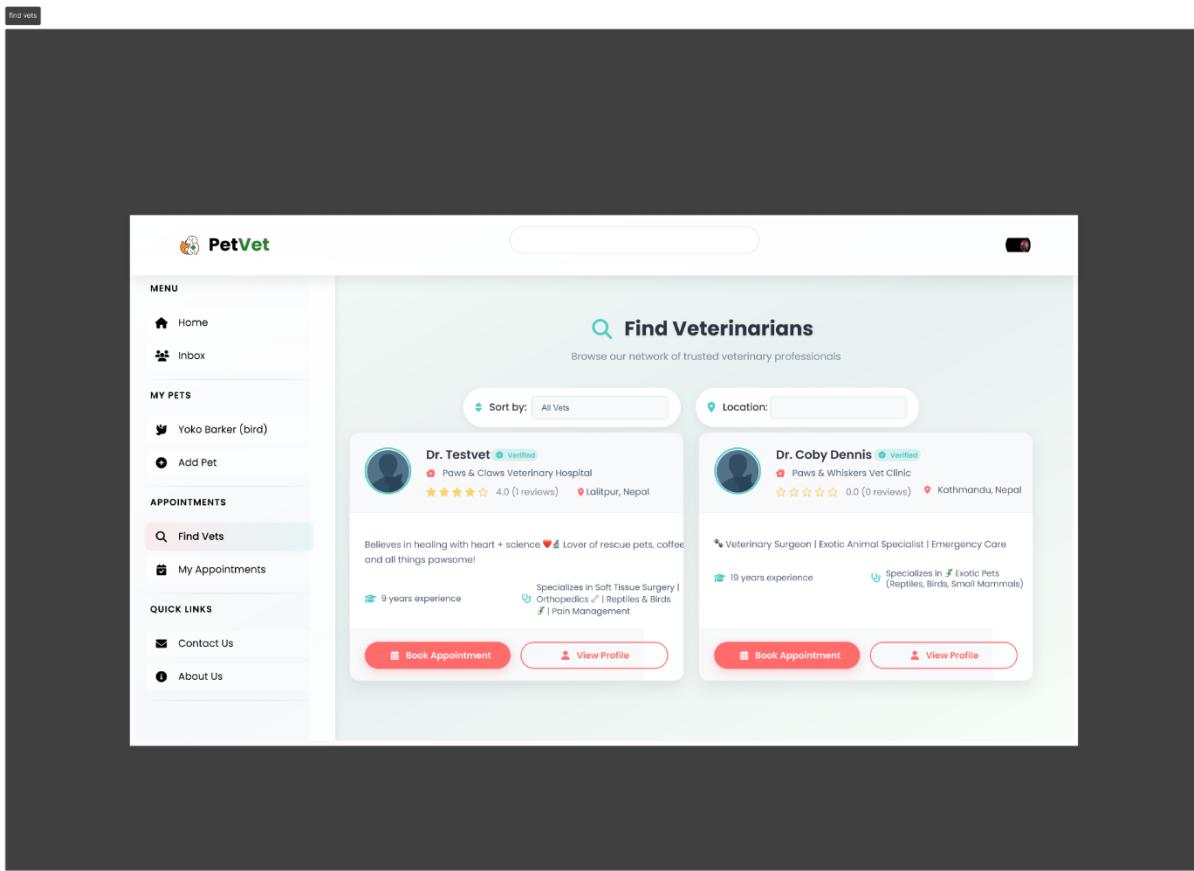


Figure 98: Main Design for Find vets

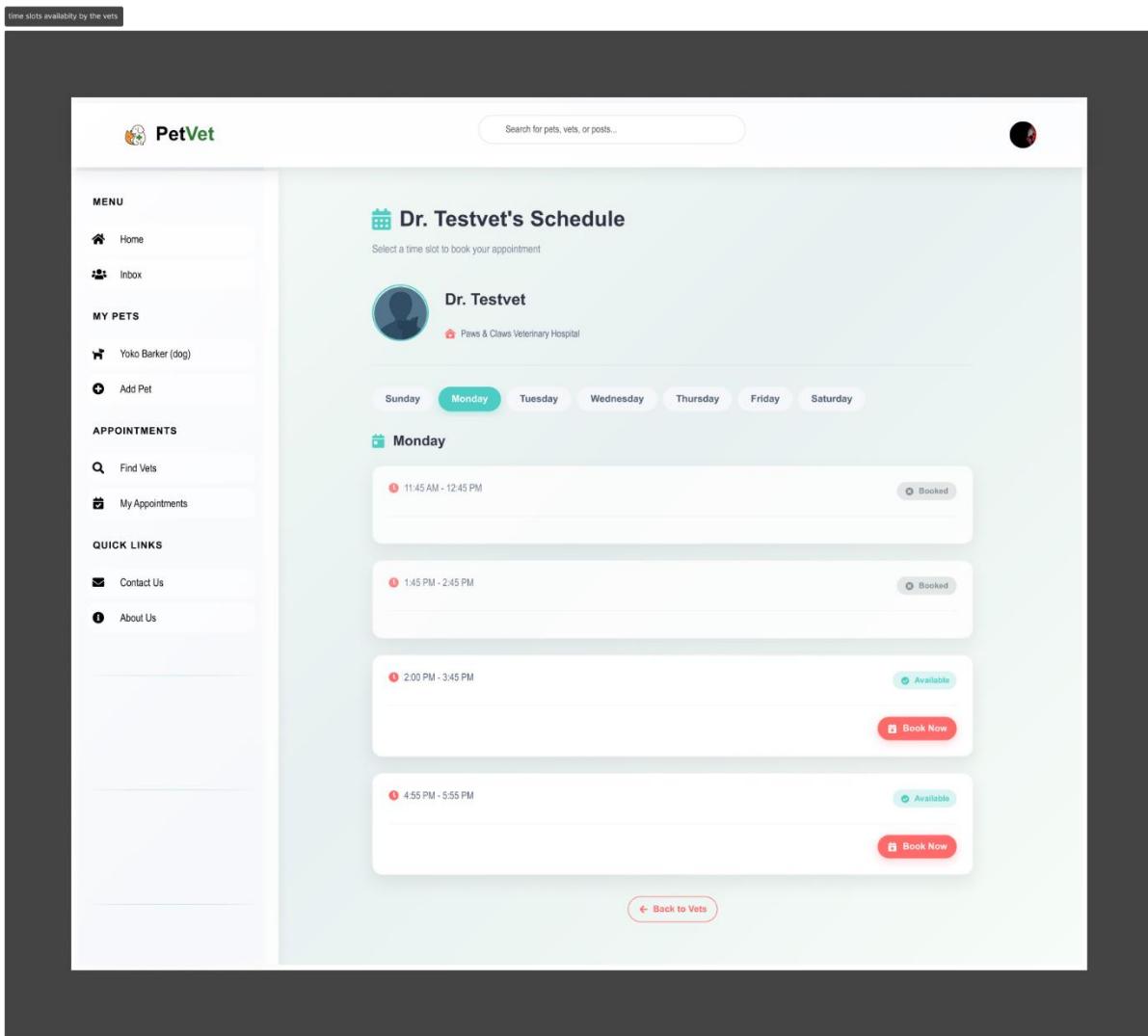


Figure 99: Main Design for Vet's Schedule list

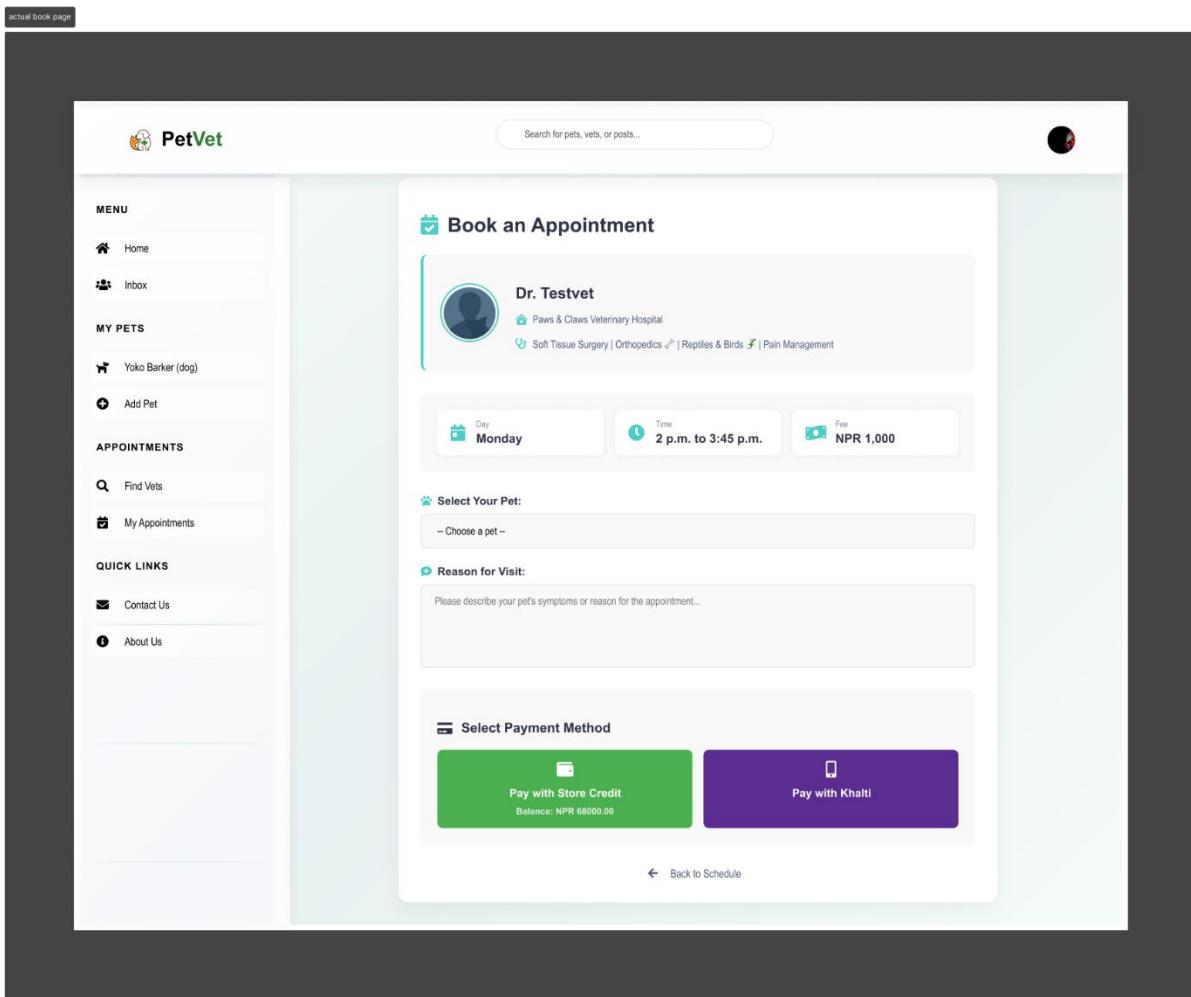


Figure 100: Main Design for Booking

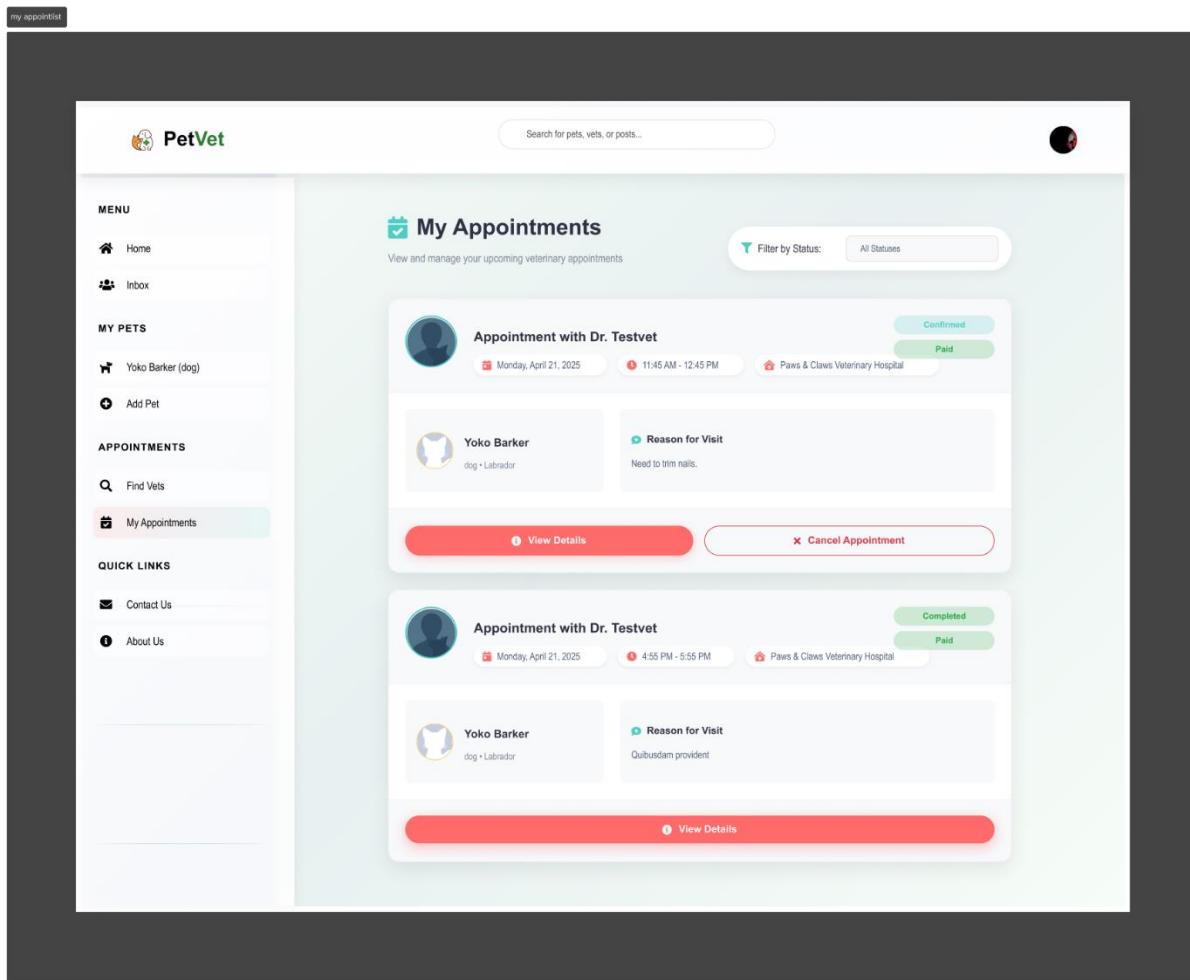


Figure 101: Main Design for Appointment lists

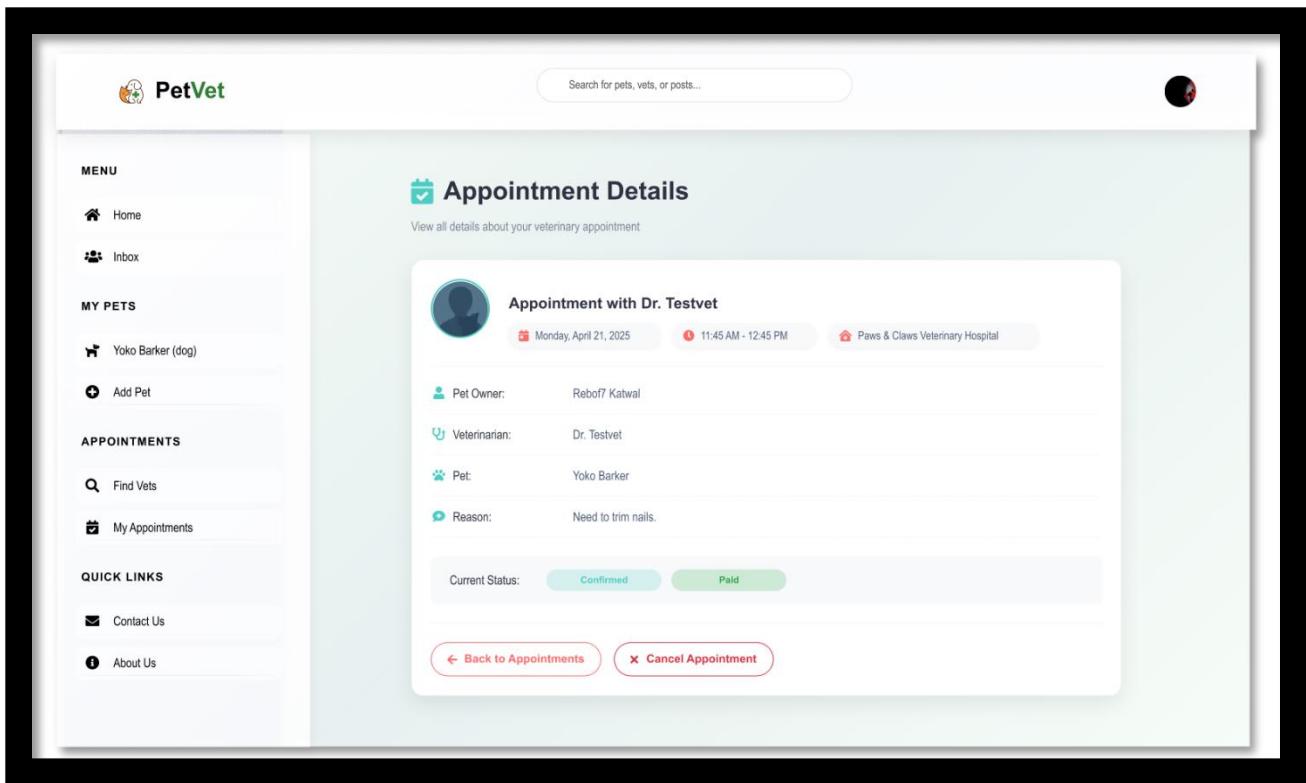


Figure 102: Main Design for Appointment details

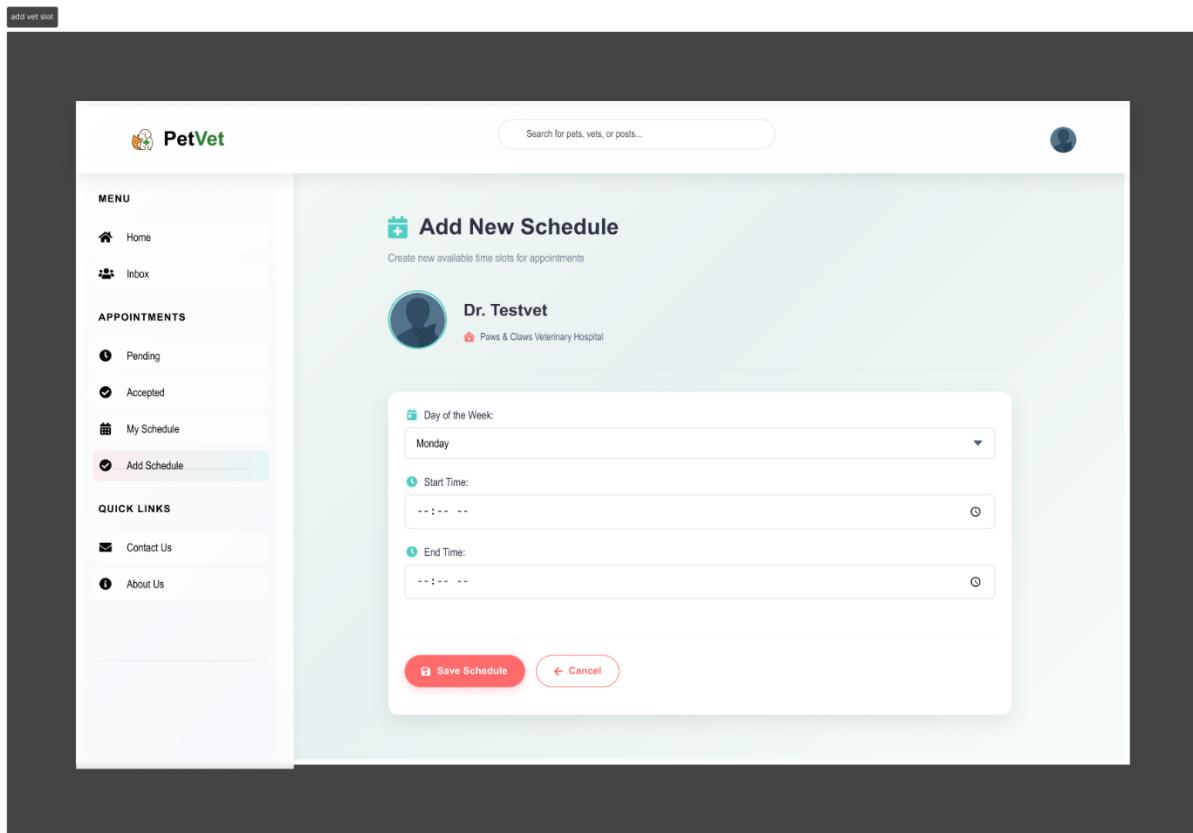


Figure 103: Main Design for Add schedule

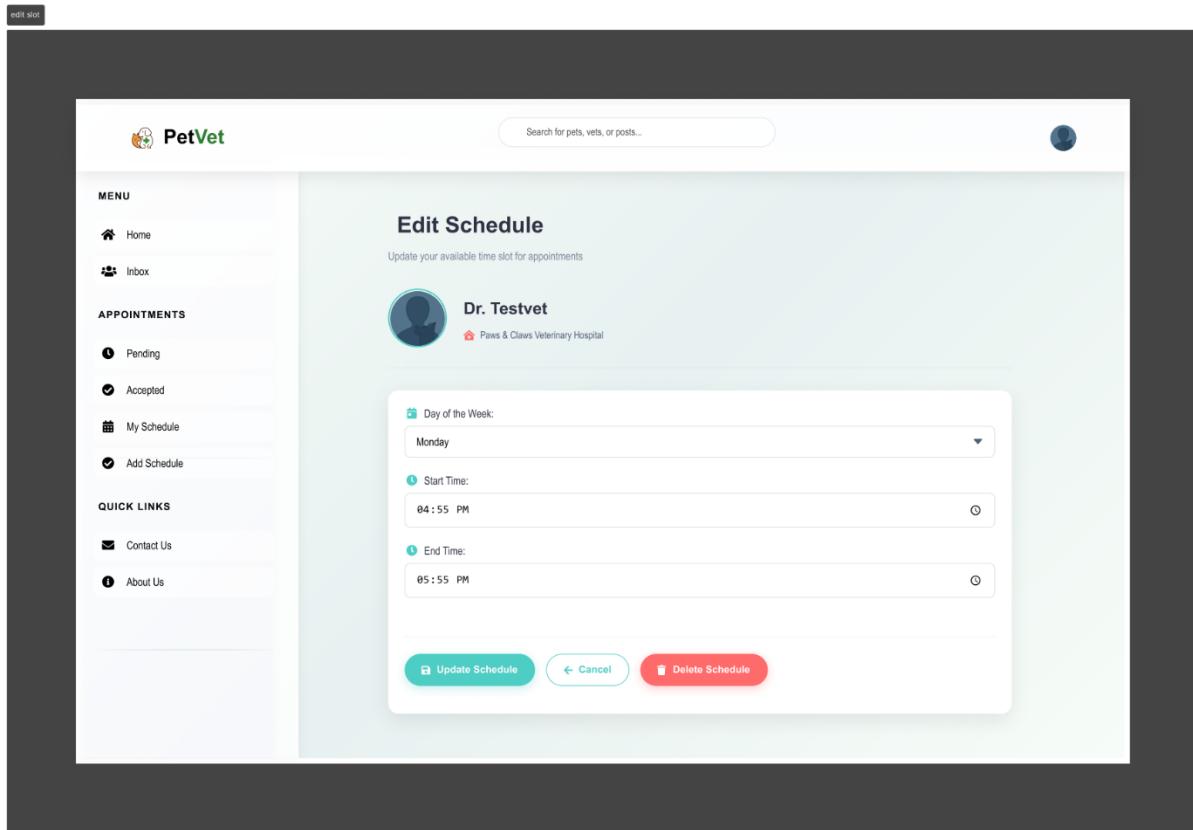


Figure 104: Main Design for Edit schedule

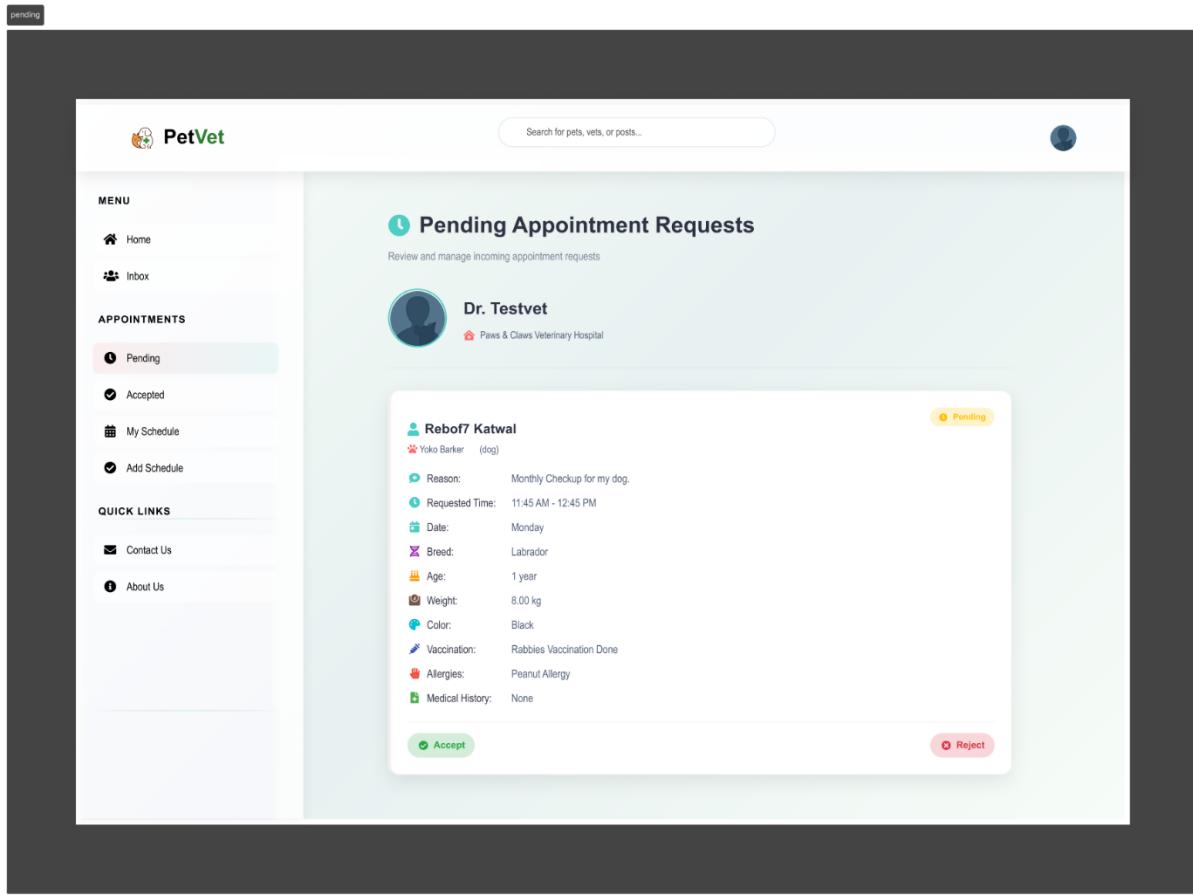


Figure 105: Main Design for Pending requests

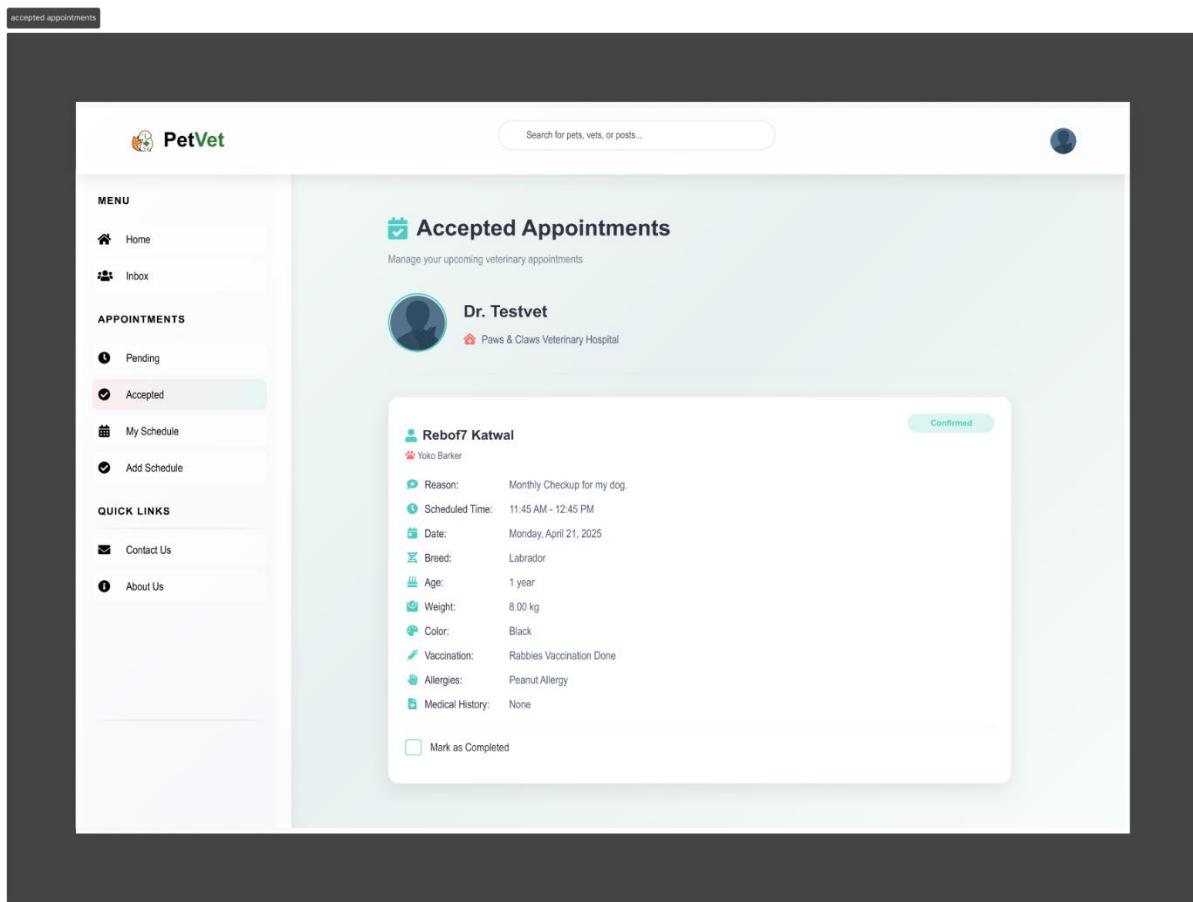


Figure 106: Main Design for Accepted bookings

3.6.4.2 Prototype/Iteration 2: Designs

3.6.4.2.1 Prototype/Iteration 2: Flowchart Diagrams

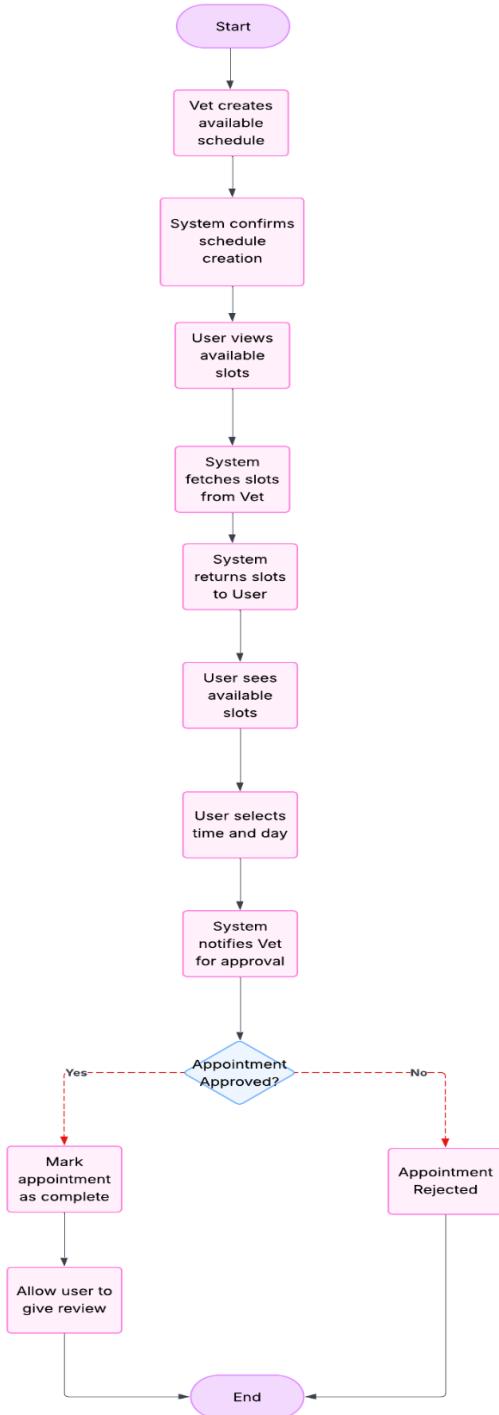


Figure 107: Flowchart for appointment

3.6.4.2.2 Prototype/Iteration 2: Activity Diagrams

Note: This is the updated version of the previous iteration of the activity diagram for the appointment system in the application. The earlier version can be found in the appendix for reference.

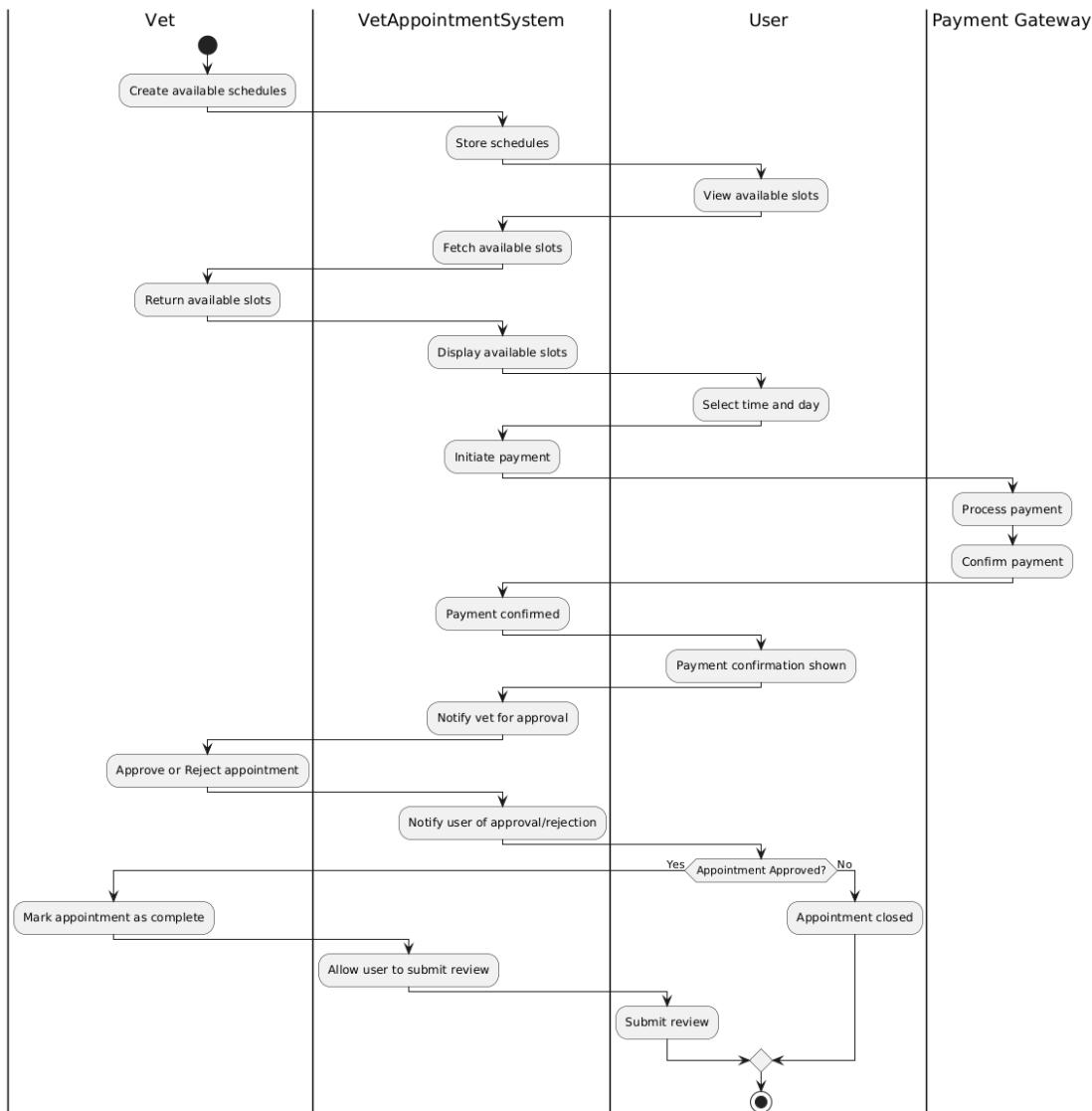


Figure 108: Activity diagram for appointment

Previous iteration of activity diagram for appointment: [7.4.6 Activity Diagram](#)

3.6.4.2.3 Prototype/Iteration 2: Sequence Diagrams

Note: This is the updated version of the previous iteration of the sequence diagram for the appointment system in the application. The earlier version can be found in the appendix for reference.

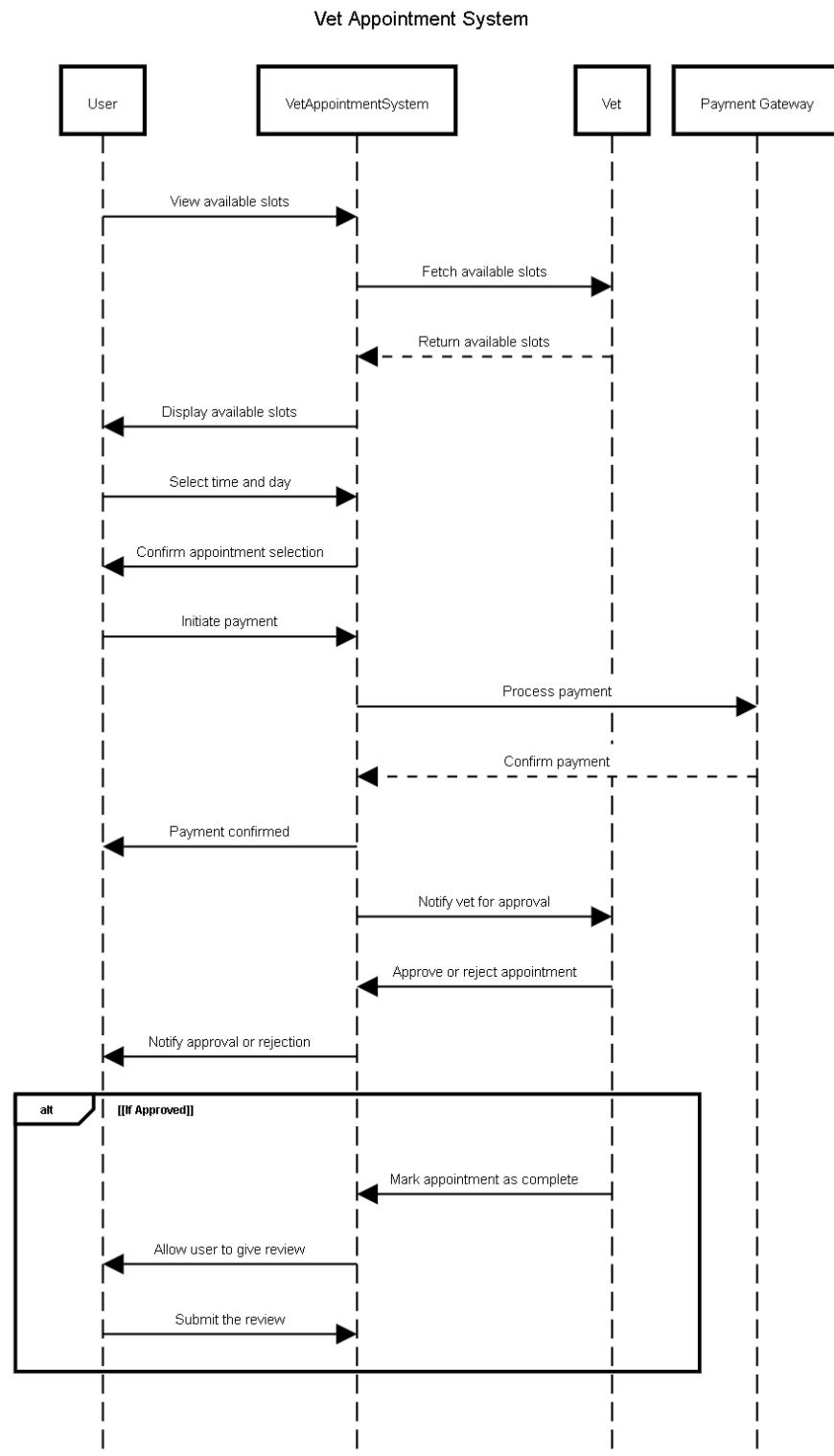


Figure 109: Sequence diagram for appointment

Previous iteration of sequence diagram for appointment: [7.4.7 Sequence Diagram](#)

3.6.4.2.4 Prototype/Iteration 2: Collaboration Diagrams

Note: This is the updated version of the previous iteration of the collaboration diagram for the appointment system in the application. The earlier version can be found in the appendix for reference.

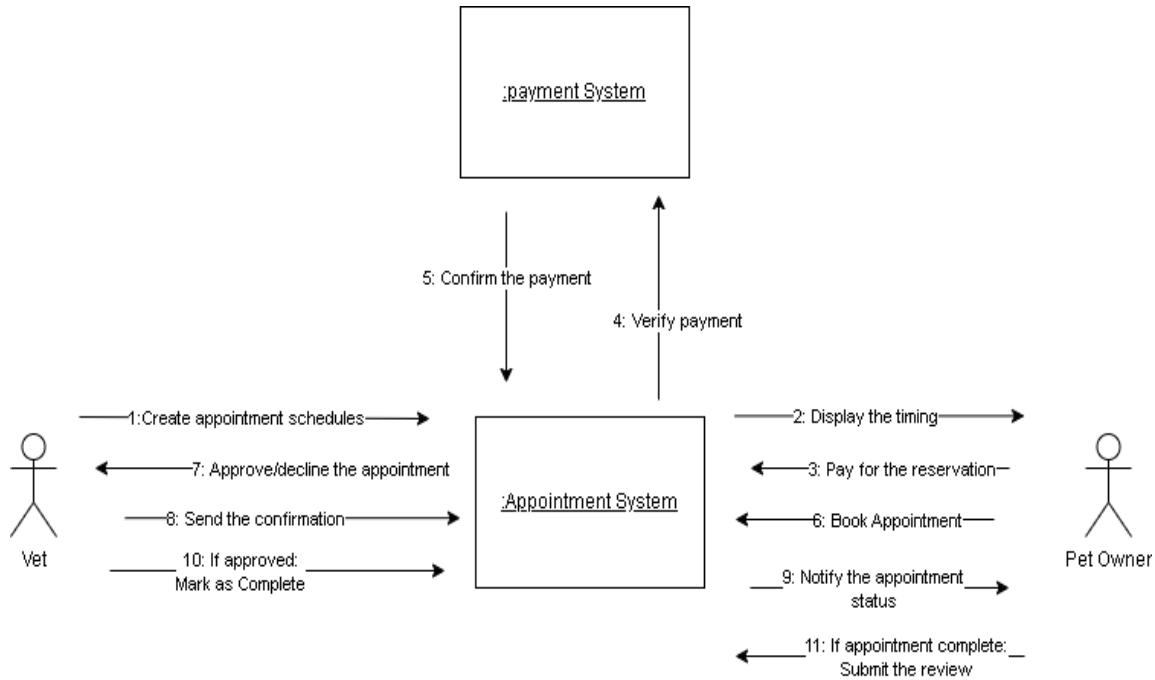


Figure 110: Collaboration for appointment

Previous iteration of collaboration diagram of appointment: [7.4.8 Collaboration Diagram](#)

3.6.4.3 Prototype/Iteration 2: Development

I have provided code snippets to illustrate the development process for key functionalities.

```
#test done
@login_required
def book_appointment(request, vet_id, schedule_id):
    vet = get_object_or_404(VetProfile, id=vet_id)
    schedule = get_object_or_404(VetSchedule, id=schedule_id)
    pet_owner = get_object_or_404(PetOwnerProfile, user=request.user)
    pets = pet_owner.pets.all() # Fetch all pets for the dropdown

    if request.method == "POST":
        pet_id = request.POST.get("pet")
        reason = request.POST.get("reason", "No reason provided")

        try:
            selected_pet = Pet.objects.get(id=pet_id, owner=pet_owner)
        except Pet.DoesNotExist:
            return render(request, "appointment/book_appt.html", {
                "vet": vet,
                "schedule": schedule,
                "pets": pets,
                "error": "Invalid pet selection."
            })

        appointment = Appointment.objects.create(
            pet_owner=pet_owner,
            vet=vet,
            schedule=schedule,
            pet=selected_pet,
            reason=reason,
            status="unpaid",
            payment_status='unpaid'
        )

        return redirect('appointment:initiate_khalti_payment', appointment_id=appointment.id)

    return render(request, "appointment/book_appt.html", {
        "vet": vet,
        "schedule": schedule,
        "pets": pets
    })
```

Figure 111 Code Snippet – Book appointment Logic in Prototype 2

```
#tested
@login_required
def cancel_appointment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)
    try:
        # Only process credit for paid appointments
        if appointment.payment_status == 'paid' and appointment.status in ['confirmed', 'paid_pending_approval']:
            # Calculate 80% refund (in paisa)
            refund_amount = int(appointment.amount_paid * 0.8)

            # Add credit to owner's account
            appointment.pet_owner.credit_balance += refund_amount
            appointment.pet_owner.save()

            # Update appointment
            appointment.status = 'cancelled'
            appointment.save()

            messages.success(request, f"Appointment cancelled. {refund_amount/100:.2f} NPR (80%) credited to your account.")
        else:
            # For unpaid appointments or invalid statuses
            appointment.status = 'cancelled'
            appointment.save()
            messages.warning(request, "Appointment cancelled (no refund applicable).")

    except Exception as e:
        messages.error(request, f"Error cancelling appointment: {str(e)}")
        return redirect('appointment:appointment_detail', appointment_id=appointment.id)

    return redirect('appointment:appointment_list')
```

Figure 112: Code Snippet – Cancel appointment Logic in Prototype 2

```
#tested
@login_required
def mark_completed(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Check permissions
    if not hasattr(appointment, 'vet') or not hasattr(appointment.vet, 'user') or \
       request.user != appointment.vet.user:
        messages.error(request, "You don't have permission to mark this appointment as completed.")
        return HttpResponseRedirect(reverse('appointment:veterinarian_accepted_appointments', args=[appointment.vet.id]))

    if request.method == "POST":
        appointment.status = "completed"
        appointment.save()
        appointment.schedule.available = True
        appointment.schedule.save()

    return redirect("appointment:veterinarian_accepted_appointments", vet_id=appointment.vet.id)
```

Figure 113: Code Snippet – Mark as complete Logic in Prototype 2

```

@login_required
@csrf_exempt
def verify_khalti_payment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)
    pidx = request.GET.get('pidx')

    if not pidx:
        return render(request, 'appointment/payment_error.html', {
            'error': 'Missing payment ID',
            'appointment': appointment
        })

    try:
        response = requests.post(
            f"{KHALTI_BASE_URL}/lookup/",
            json={"pidx": pidx},
            headers={
                "Authorization": KHALTI_SECRET_KEY,
                "Content-Type": "application/json"
            }
        )
        response.raise_for_status()
        data = response.json()

        if data.get('status') != 'Completed':
            appointment.status = 'unpaid'
            appointment.payment_status = 'failed'
            appointment.save()
            return redirect('appointment:payment_failed', appointment_id=appointment.id)

        # Payment successful - store details and await vet approval
        appointment.pidx = pidx
        appointment.amount_paid = data['total_amount']
        appointment.status = 'paid_pending_approval'
        appointment.payment_status = 'paid'
        appointment.save()

        # Notify vet
        send_vet_appointment_notification()
    
```

Figure 114: Code Snippet – Verify khalti payment Logic in Prototype 2

3.6.4.4 Prototype/Iteration 2: Testing

This is the key feature prototype, as it includes the core functionality of the application. Major features such as appointment cancellation, rejection, acceptance, vet scheduling (add/edit), email alerts, booking via Khalti, and store credit were all unit tested. These tests are documented in the testing section under the labels APP1 to APP9.

Test: Appt1 Add Schedule Functionality

4.2.9 Test: Appt2 Edit Schedule Functionality

4.2.10 Test: Appt3 Booking via Khalti Functionality

4.2.11 Test: Appt4 Booking via Store Credit Functionality

4.2.12 Test: Appt5 Accept Booking Functionality

4.2.13 Test: Appt6 Reject Booking Functionality

4.2.14 Test: Appt7 Mark as Complete Functionality

4.2.15 Test: Appt8 Booking Cancellation Functionality

4.2.16 Test: Appt9 Khalti Payment Functionality

3.6.4.5 Prototype/Iteration 2: User Feedback

The appointment and payment system received positive feedback from 30 users. All users successfully booked appointments, with veterinarians finding scheduling easy. The Khalti payment system was smooth and secure, with 96% of users supporting a credit system for canceled appointments. Overall satisfaction was high, with users appreciating the system's simplicity and efficiency. Suggestions included clear pricing and reminders.

Below is the link that leads to the actual charts and a sample filled user feedback form:

[7.6.5 User Feedback Form for Prototype 2 Results](#)

[7.6.6 User Feedback Form for Prototype 2 Sample Answer](#)

3.6.5 Prototype/Iteration 3

the real time chat system, which was introduced in prototype 3 was not directly part of the appointment system but was closely related to it. It was high in complexity, but small in scope. Having the ability to communicate with a vet instantly, to send notifications, chat with each other, and the persistence of the messages in database was made possible by this module with support for message routing.

3.6.5.1 Prototype/Iteration 3: Figma Design

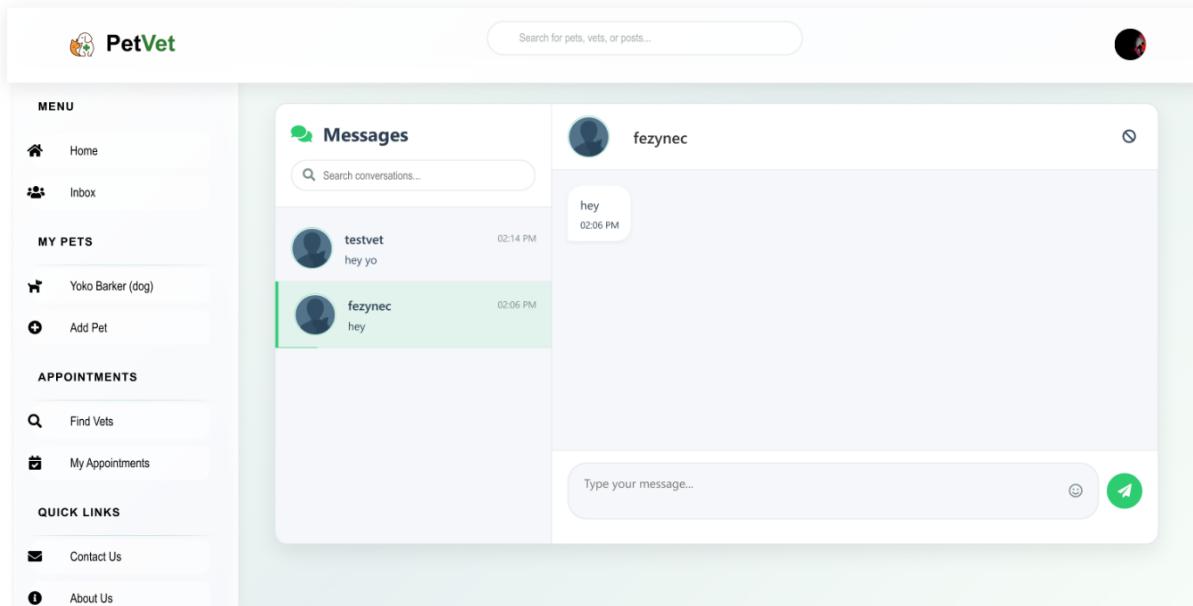


Figure 115: Main Design for Inbox

3.6.5.2 Prototype/Iteration 3: Designs

3.6.5.2.1 Prototype/Iteration 3: Flowchart

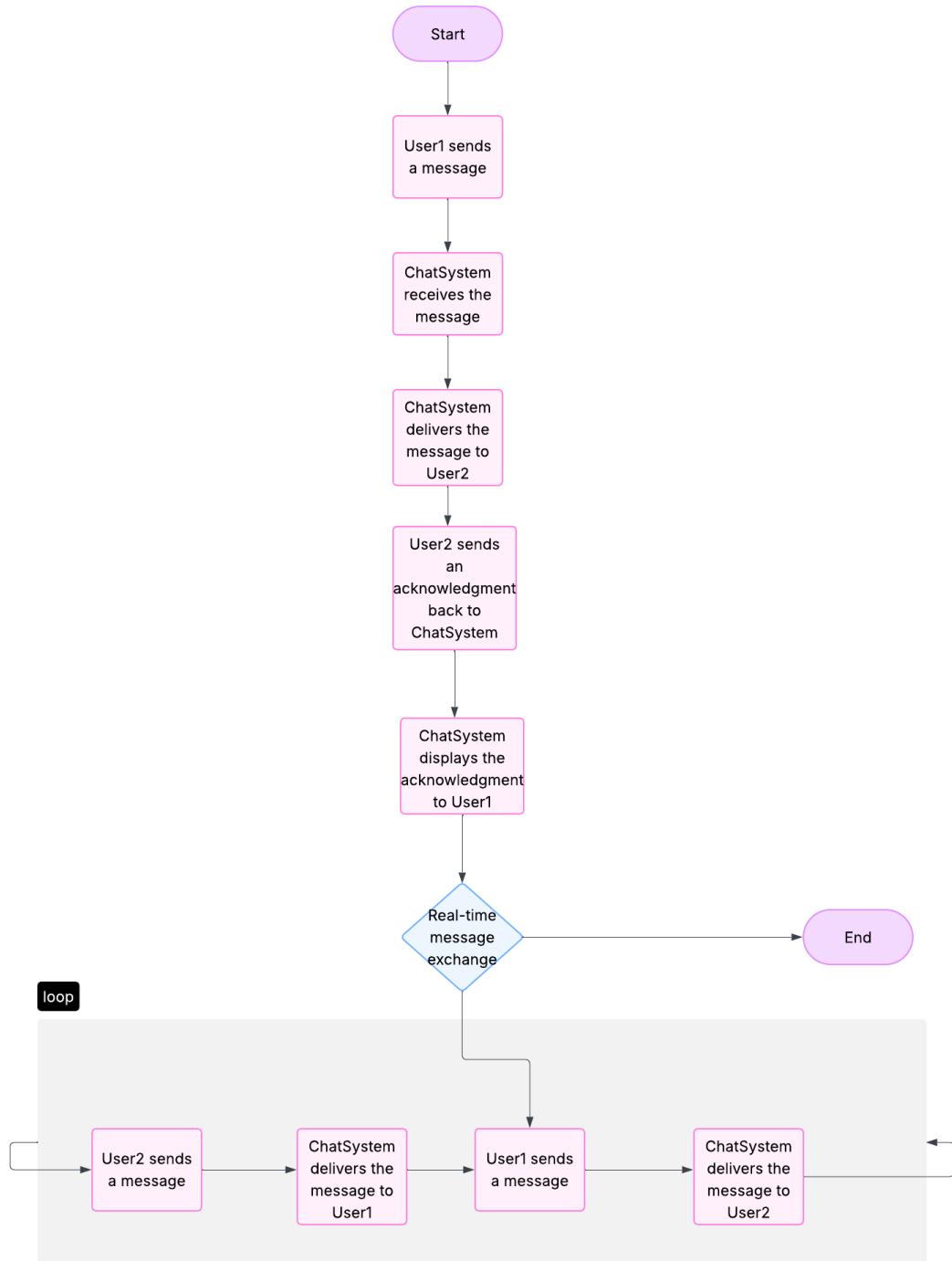
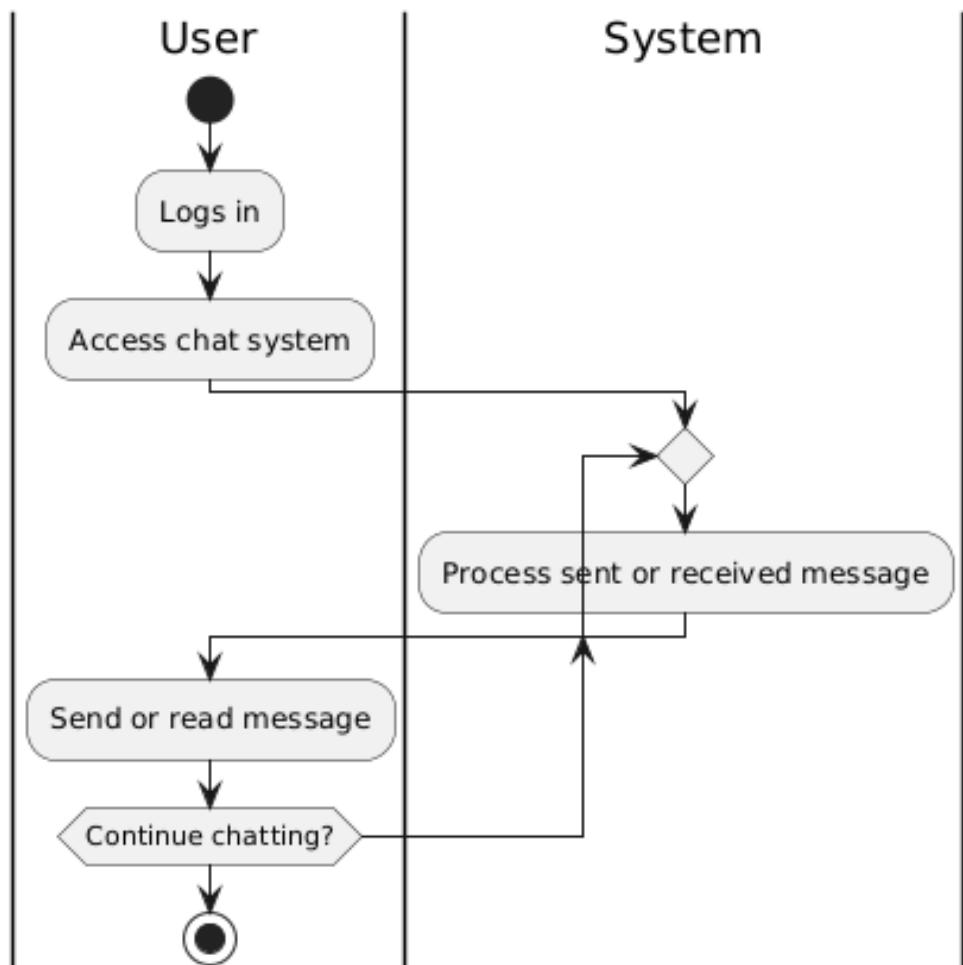
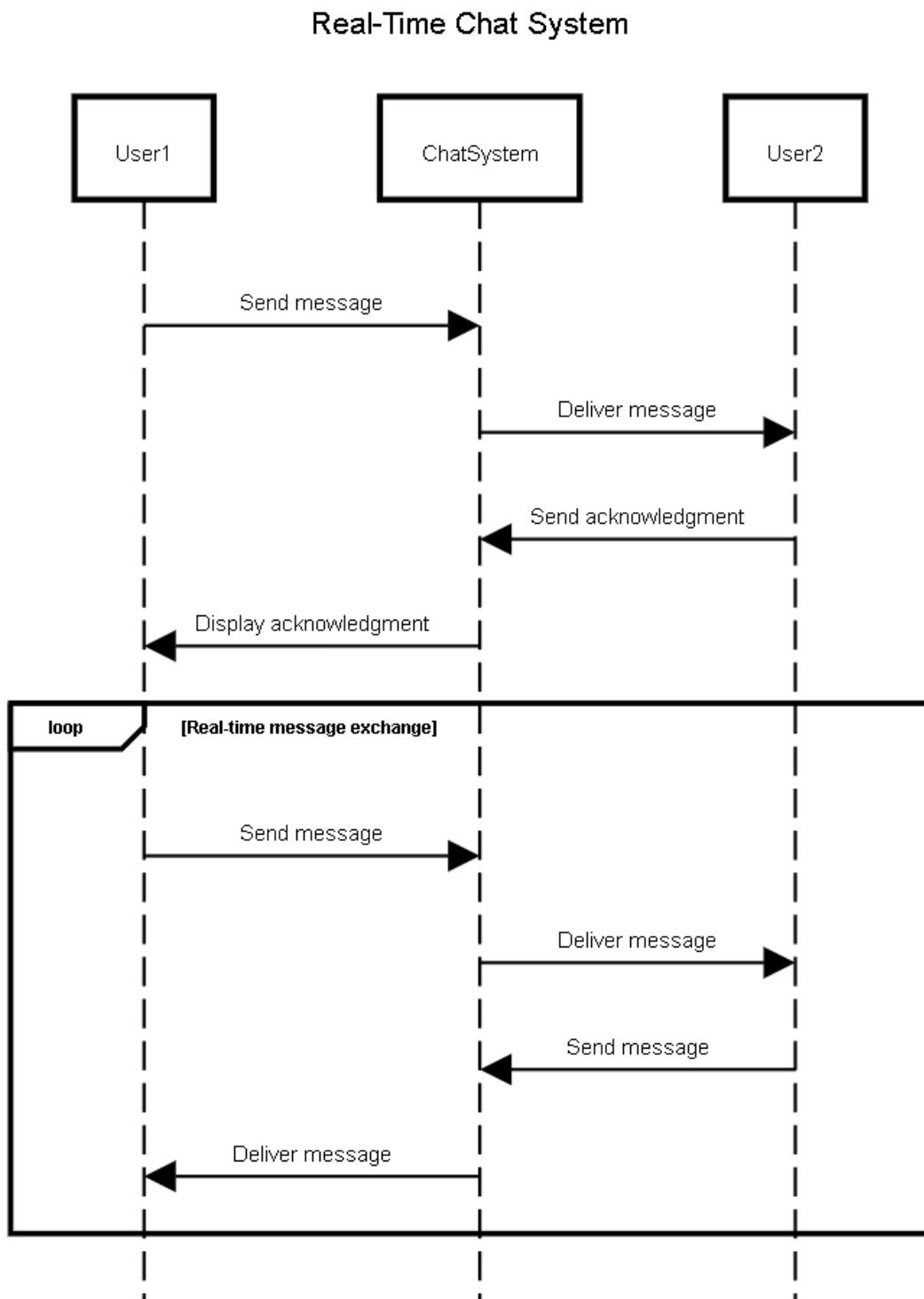


Figure 116: Flowchart for Chat system

3.6.5.2.2 Prototype/Iteration 3: Activity Diagram*Figure 117: Main Design for Chat system*

3.6.5.2.3 Prototype/Iteration 3: Sequence Diagram*Figure 118: Sequence diagram for chat system*

3.6.5.2.4 Prototype/Iteration 3: Collaboration Diagram

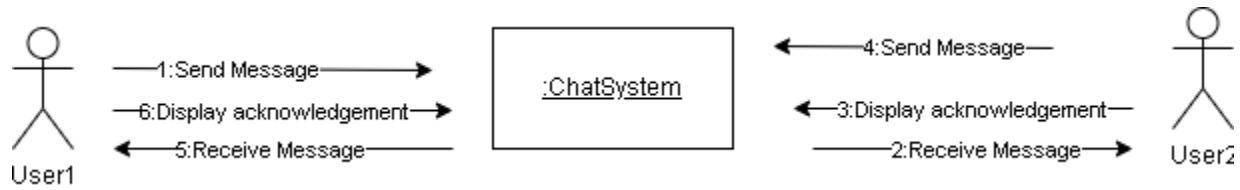


Figure 119: Collaboration diagram for chat system

3.6.5.3 Prototype/Iteration 3: Development

I have provided code snippets to illustrate the development process for key functionalities.

```
class NotificationConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.username = self.scope['url_route'][ 'kwargs' ][ 'username' ]
        self.room_group_name = f'notifications_{self.username}'

        # Join room group
        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.accept()

    async def disconnect(self, close_code):
        # Leave room group
        await self.channel_layer.group_discard(
            self.room_group_name,
            self.channel_name
        )

    async def send_notification(self, event):
        # Send notification to WebSocket
        await self.send(text_data=json.dumps({
            'type': 'new_message',
            'message': event[ 'message' ],
            'sender': event.get('sender'),
            'timestamp': event.get('timestamp'),
        }))
```

Figure 120: Code Snippet – Chat Notification Logic in Prototype 3

```

class ChatConsumer(WebSocketConsumer):
    def connect(self):
        self.room_name = self.scope['url_route']['kwargs']['room_name']
        self.room_group_name = 'chat_%s' % self.room_name

        async_to_sync(self.channel_layer.group_add)(
            self.room_group_name,
            self.channel_name
        )

        self.accept()

    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)(
            self.room_group_name,
            self.channel_name
        )

    def receive(self, text_data):
        data = json.loads(text_data)
        message = data.get('message')
        sender_username = data.get('sender')
        timestamp = data.get('timestamp', datetime.datetime.now().isoformat())

        try:
            # Get sender and receiver user objects
            sender = User.objects.get(username=sender_username)
            receiver = User.objects.get(username=data['receiver'])

            # Create and save the chat message
            chat_message = ChatMessage(

```

Figure 121: Code Snippet – Chat Logic in Prototype 3

```

@login_required
def inbox_details(request, username=None):
    sender = request.user
    receiver = None
    messages_detail = None

    if username:
        receiver = get_object_or_404(User, username=username)
        # Fetch conversation between sender and receiver, ordered by date
        messages_detail = ChatMessage.objects.filter(
            Q(sender=sender, receiver=receiver) | Q(sender=receiver, receiver=sender)
        ).order_by("date")
        # Mark received messages as read
        messages_detail.filter(receiver=sender).update(is_read=True)

    # Get recent conversations for sidebar
    recent_messages = ChatMessage.objects.filter(
        Q(sender=sender) | Q(receiver=sender)
    ).order_by("-date")

    # Organize conversations by partner, keeping the latest message
    conversation_partners = {}
    for message in recent_messages:
        if not message.sender or not message.receiver:
            continue
        partner = message.sender if message.sender != sender else message.receiver
        if partner.id not in conversation_partners:
            conversation_partners[partner.id] = message

    context = {
        "message_detail": messages_detail,
        "receiver": receiver,
        "message_list": conversation_partners.values(),
    }
    return render(request, 'chat/inbox_detail.html', context)

```

Figure 122: Code Snippet – Chat page rendering Logic in Prototype 3

3.6.5.4 Prototype/Iteration 3: Testing

In this prototype, only the real-time chat system was developed. The focus was on testing WebSocket functionality, notifications, and proper routing and saving of messages to the database. These were verified and documented under the test case labeled RTC1 in the testing section.

4.2.22 Test: RTC1 Real-Time Chat Functionality

3.6.5.5 Prototype/Iteration 3: User Feedback

The chat system prototype was well received by the 30 users who tested it. Starting a chat was easy for most, with 96% rating it positively. The message speed was also fast, with 93% rating it 4 or 5. The chat interface was found to be clear and simple, with 89% rating it positively. The feature was considered useful by 92% of users, and overall satisfaction was high, with 89% of users rating it 4 or 5. The system was praised for its smooth operation, responsiveness, and ease of use, with no major technical issues reported.

Below is the link that leads to the actual charts and a sample filled user feedback form:

[7.6.7 User Feedback Form for Prototype 3 Results](#)

[7.6.8 User Feedback Form for Prototype 3 Sample Answer](#)

3.6.6 Prototype/Iteration 4

Prototype 4 aimed to improve the user interface (UI) of the admin section, based on the already existing admin authentication and vet approval features. The functionalities added in this iteration were post management, category management and viewing user details. These improvements improved the overall admin experience by making it easier for the admin to regulate posts, categories, and user details easier and eventually aiding his workflow and views over the administration.

3.6.6.1 Prototype/Iteration 4: Figma Design

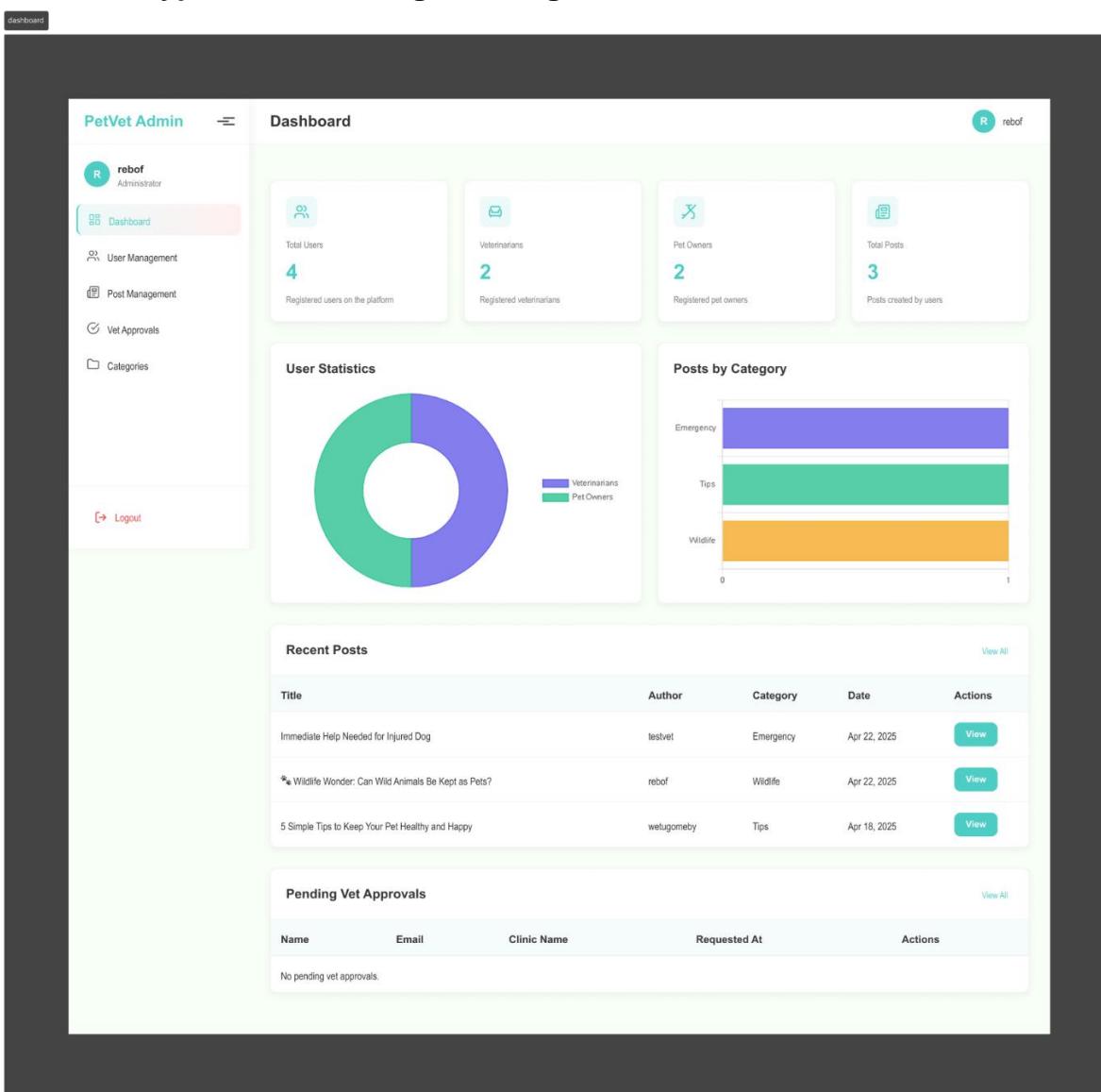


Figure 123: Main Design for Admin dashboard

The screenshot shows the 'User Management' section of the PetVet Admin application. On the left, there's a sidebar with a user profile for 'rebof' (Administrator), a 'Dashboard' link, and a 'User Management' link which is currently active and highlighted in pink. Other links include 'Post Management', 'Vet Approvals', and 'Categories'. At the bottom of the sidebar is a 'Logout' button. The main content area is titled 'User Management' and contains a search bar with 'Search by name, email, or username' and a dropdown for 'User Type' set to 'All Users', followed by a 'Search' button. Below this is a table listing four users:

Name	Username	Email	User Type	Joined	Status	Actions
Nicole Jo	welugomeby	monofli@mailinator.com	Pet Owner	Apr 18, 2025	Verified	<button>View</button>
Coby Dennis	fezynec	testpet34@gmail.com	Veterinarian	Apr 17, 2025	Verified	<button>View</button>
Testvet	testvet	testvet@gmail.com	Veterinarian	Feb 12, 2025	Verified	<button>View</button>
Rebof7 Katwal	rebof	rebofkatwal7@gmail.com	Pet Owner	Oct 11, 2024	Unverified	<button>View</button>

Figure 124: Main Design for User management

The screenshot shows the 'User Details' page within the PetVet Admin interface. The top navigation bar includes the title 'user details admin page' and the logo 'rebof'. The left sidebar contains links for 'Dashboard', 'User Management' (which is currently active), 'Post Management', 'Vet Approvals', and 'Categories'. A red 'Logout' button is located at the bottom of the sidebar.

The main content area displays the user profile for 'Nicole Jo', marked as a 'Pet Owner' and 'Verified'. The 'Account Information' section shows the following details:

Username	Email	Full Name	Phone
wetugomeby	monof@mailinator.com	Nicole Jo	+1 (126) 666-95

The 'Profile Information' section includes:

Bio	Pets Owned	Credit Balance	Country
Qui sint sint recusandae Praesentium consequatur cillum ipsum impedit recusandae Ut libero anim p	23	0	Illum atque ut nisi aliquam delectus at obcaecati eum facere rerum quis

Below this, there is a section for 'Address' with placeholder text: 'Sit suscipit earum alias quidem alias sunt est nisi consequatur cum enim ad eveniet quibusdam aut'.

The 'User Posts (1)' section lists one post:

Title	Category	Date	Views	Likes	Status	Actions
5 Simple Tips to Keep Your Pet Healthy and Happy	Tips	Apr 18, 2025	0	2	Active	<button>View</button>

Figure 125: Main Design for User Details

The screenshot shows the 'Post Management' section of the PetVet Admin interface. On the left is a sidebar with navigation links: Dashboard, User Management, Post Management (which is active and highlighted in pink), Vet Approvals, and Categories. At the bottom of the sidebar are 'Logout' and a small icon. The main area has a header 'Post Management' with a search bar and a 'Category' dropdown set to 'All Categories'. Below is a table with the following data:

Title	Author	Category	Date	Likes	Status	Actions
Immediate Help Needed for Injured Dog	testvet	Emergency	Apr 22, 2025	0	Active	<button>View</button> <button>Delete</button>
Wildlife Wonder: Can Wild Animals Be Kept as Pets?	rebof	Wildlife	Apr 22, 2025	0	Active	<button>View</button> <button>Delete</button>
5 Simple Tips to Keep Your Pet Healthy and Happy	wetugomeby	Tips	Apr 18, 2025	2	Active	<button>View</button> <button>Delete</button>

Figure 126: Main Design for Post management

The screenshot shows the 'View Post' section of the PetVet Admin interface. The sidebar is identical to Figure 126. The main area has a header 'View Post' and displays the following details for the post 'Immediate Help Needed for Injured Dog':

Immediate Help Needed for Injured Dog
 Posted by testvet on April 22, 2025 | Category: Emergency | Views: 0 | Likes: 0

Hi everyone,
 I'm a vet currently available for emergency services. If your pet is injured or in critical condition, please don't hesitate to reach out. I'm located near Balajji and can provide urgent care.
 Stay safe,
 Dr. Testvet

[Delete Post](#)

Comments (0)
 No comments yet.

Figure 127: Main Design for Admin post detail

The screenshot shows the 'Category Management' section of the PetVet Admin dashboard. On the left, there's a sidebar with links: Dashboard, User Management, Post Management, Vet Approvals, and Categories (which is highlighted with a pink background). The main area has a header 'Category Management'. On the right, there's a table titled 'Categories' showing three entries: Emergency, Tips, and Wildlife. Each entry includes columns for Name, Slug, Description, Post Count, and Actions (Edit and Delete buttons). To the left of the table is a form titled 'Add New Category' with fields for 'Category Name' and 'Description', and a 'Add Category' button.

Name	Slug	Description	Post Count	Actions
Emergency	emergency-en	Important	1	<button>Edit</button> <button>Delete</button>
Tips	tips-fk	Giving suggestion and health advices	1	<button>Edit</button> <button>Delete</button>
Wildlife	wildlife-rf	Wild animal	1	<button>Edit</button> <button>Delete</button>

Figure 128: Main Design for Category Management

3.6.6.2 Prototype/Iteration 4: Designs

3.6.6.2.1 Prototype/Iteration 4: Flowchart

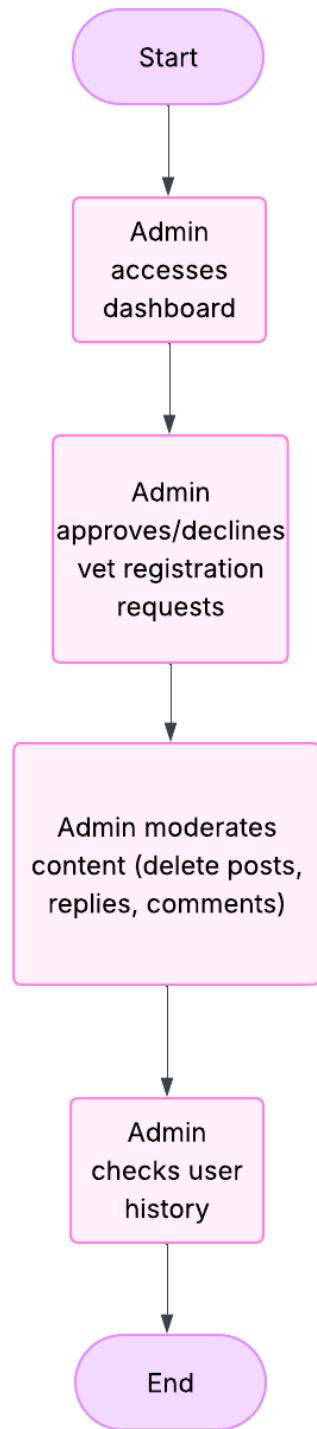


Figure 129: Flowchart for admin dashboard

3.6.6.2.2 Prototype/Iteration 4: Activity Diagram

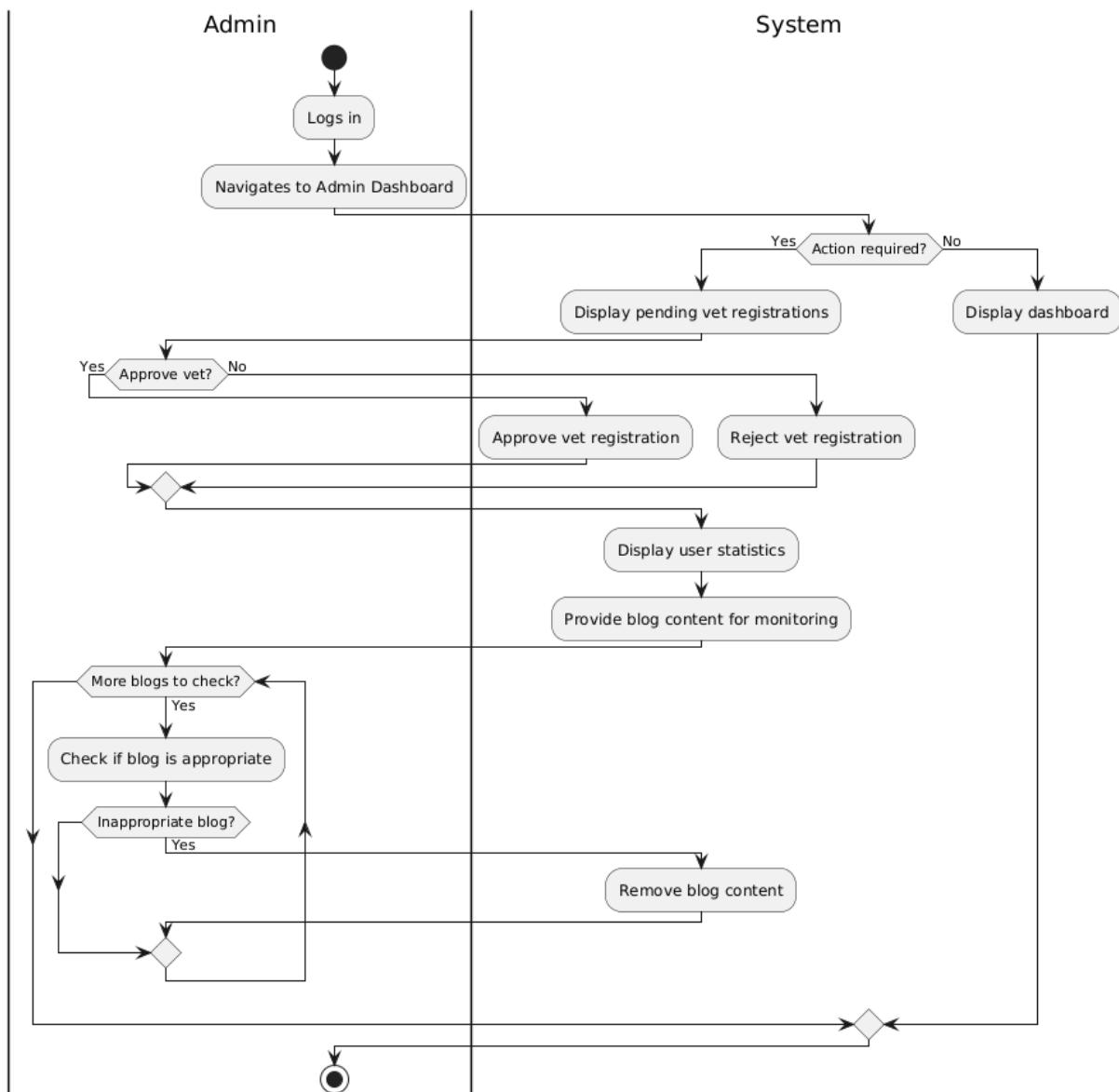


Figure 130: Activity diagram for admin dashboard

3.6.6.2.3 Prototype/Iteration 4: Sequence Diagram

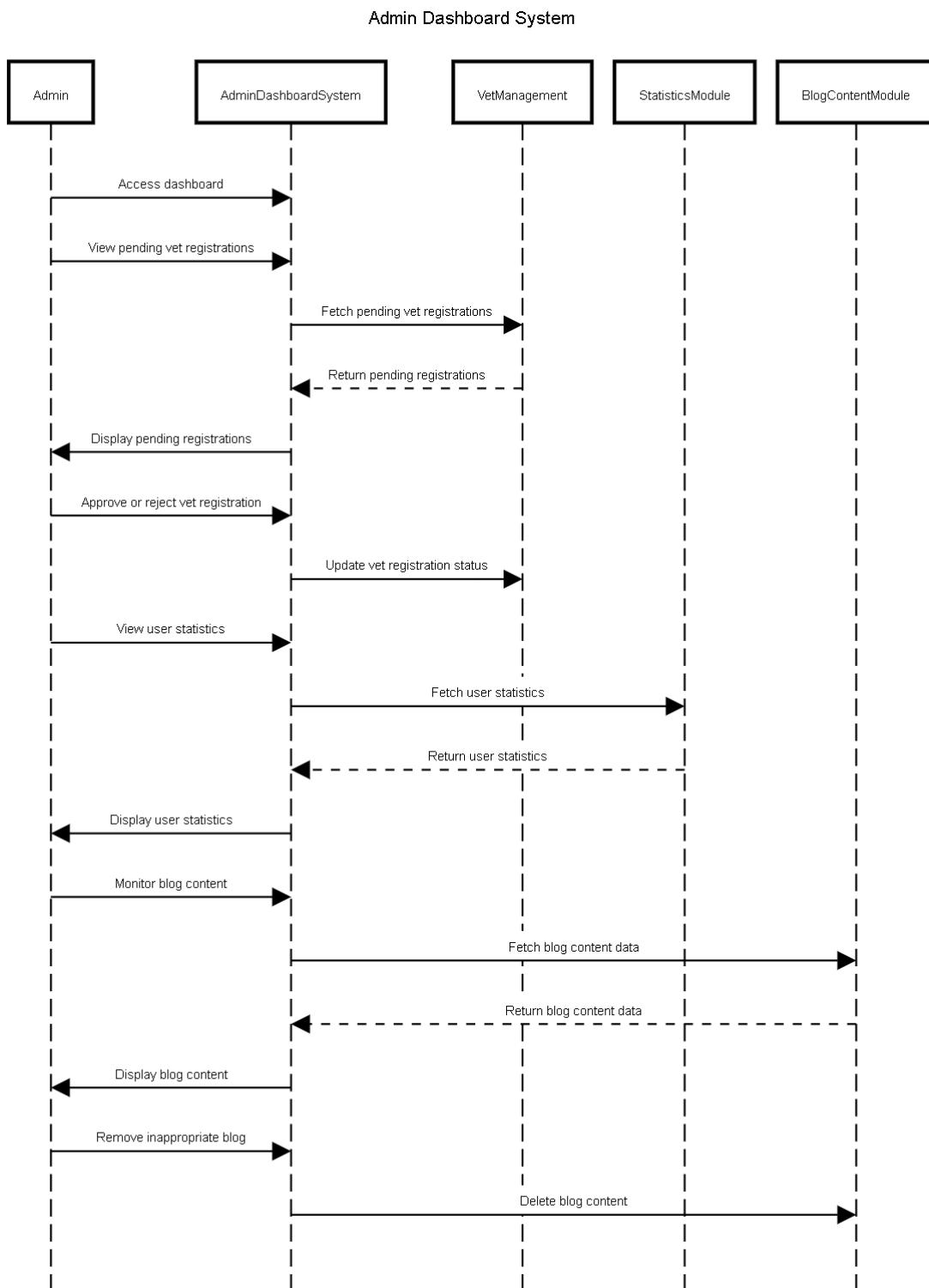


Figure 131: Sequence diagram for admin dashboard

3.6.6.2.4 Prototype/Iteration 4: Collaboration Diagram

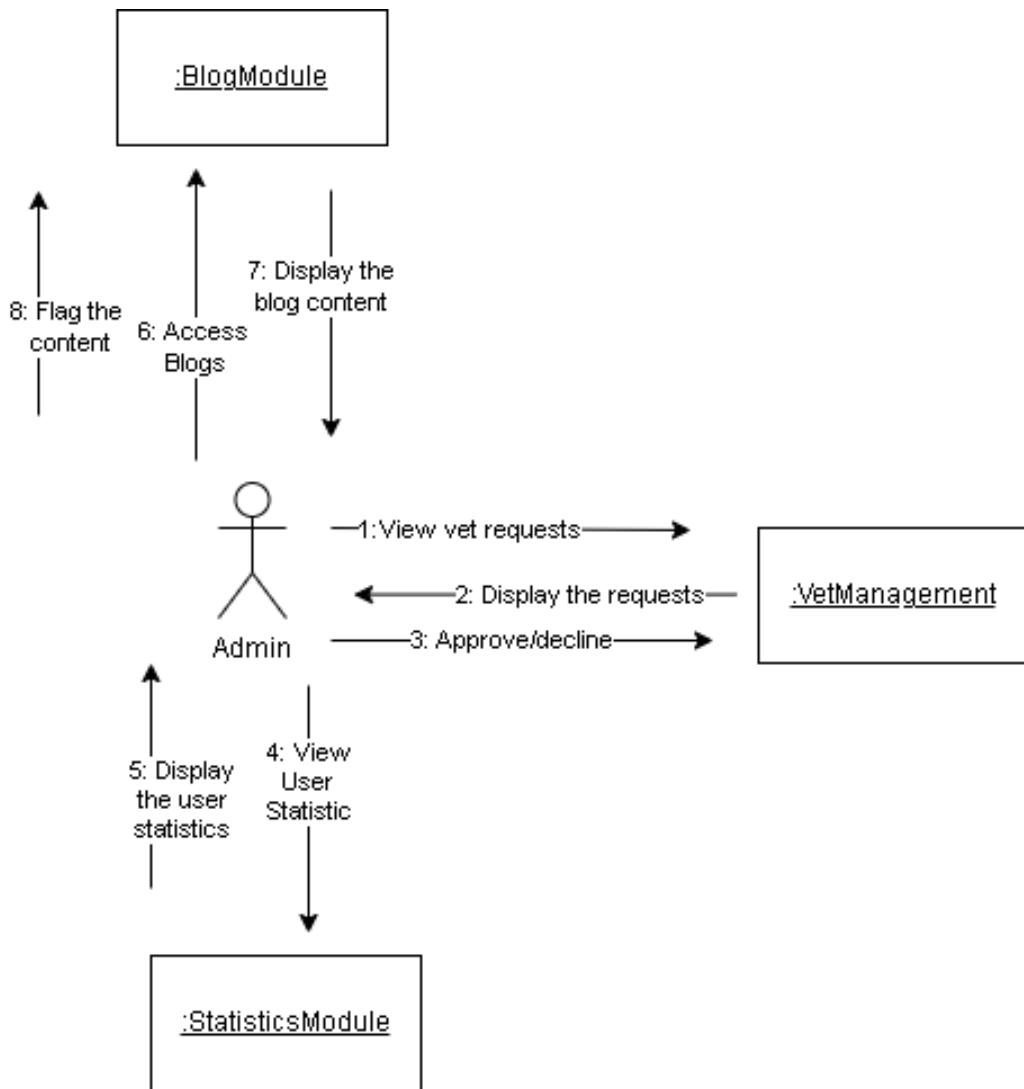


Figure 132: Collaboration diagram for admin dashboard

3.6.6.3 Prototype/Iteration 4: Development

I have provided code snippets to illustrate the development process for key functionalities.

```
from django.shortcuts import redirect
from django.contrib import messages

def admin_required(view_func):
    def wrapper(request, *args, **kwargs):
        if not request.user.is_authenticated or not request.user.is_staff:
            messages.error(request, "You don't have permission to access this page.")
            return redirect('django_admin:admin_login')
        return view_func(request, *args, **kwargs)
    return wrapper
```

Figure 133: Code Snippet – Admin required decorator Logic in Prototype 4

```
@admin_required
def post_management(request):
    search_query = request.GET.get('search', '')
    category_id = request.GET.get('category', '')

    # Filter posts based on search and category
    posts = Post.objects.all()

    if search_query:
        posts = posts.filter(
            Q(title__icontains=search_query) |
            Q(body__icontains=search_query)
        )

    if category_id:
        posts = posts.filter(category_id=category_id)

    # Get all categories for filter dropdown
    categories = Category.objects.all()

    # Paginate results
    paginator = Paginator(posts.order_by('-date'), 10)
    page = request.GET.get('page', 1)
    posts = paginator.get_page(page)

    return render(request, 'django_admin/post_management.html', {
        'posts': posts,
        'categories': categories,
        'admin_user': request.user,
        'active_page': 'posts'
    })
```

Figure 134: Code Snippet – Post management Logic in Prototype 4

```
#tested
@admin_required
def approve_vet(request, vet_id):
    vet_profile = get_object_or_404(VetProfile, id=vet_id)
    vet_user = vet_profile.user

    vet_user.status_verification = True
    vet_profile.verified = True
    vet_profile.status_change = now()

    vet_user.save()
    vet_profile.save()

    # Send email notification
    try:
        send_mail(
            'Your Veterinarian Account Has Been Approved',
            f'Dear {vet_user.full_name or vet_user.username},\n\nCongratulations! Your veterinarian account on PetVet has been
            settings.DEFAULT_FROM_EMAIL,
            [vet_user],
            fail_silently=False,
        )
    except Exception as e:
        messages.warning(request, f"Vet approved but email notification failed: {str(e)}")

    messages.success(request, f"Veterinarian '{vet_user.full_name or vet_user.username}' has been approved successfully.")
    return redirect('django_admin:vet_approvals')
```

Figure 135: Code Snippet – Approve vet Logic in Prototype 4

```

@admin_required
def add_category(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        description = request.POST.get('description')

        if Category.objects.filter(name=name).exists():
            messages.error(request, f"Category '{name}' already exists.")
        else:
            category = Category.objects.create(name=name, description=description)
            messages.success(request, f"Category '{name}' has been added successfully.")

    return redirect('django_admin:category_management')


@admin_required
def edit_category(request, category_id):
    category = get_object_or_404(Category, id=category_id)

    if request.method == 'POST':
        name = request.POST.get('name')
        description = request.POST.get('description')

        # Check if name already exists for another category
        if Category.objects.filter(name=name).exclude(id=category_id).exists():
            messages.error(request, f"Category '{name}' already exists.")
        else:
            category.name = name
            category.description = description
            category.save()
            messages.success(request, f"Category '{name}' has been updated successfully.")

    return redirect('django_admin:category_management')

```

Figure 136: Code Snippet – Add category Logic in Prototype 4

3.6.6.4 Prototype/Iteration 4: Testing

In this prototype, the admin dashboard was fully developed and refined. Previously, only vet approval and admin login existed. Now, features such as post management, user details page, and category management were implemented and tested. These are documented under the test cases ADMIN3 and ADMIN4 in the testing section.

4.2.6 Test: Admin3 Content Moderation Functionality

4.2.7 Test: Admin4 Category Management Functionality

3.6.6.5 Prototype/Iteration 4: User Feedback

The improved admin dashboard prototype was tested by 30 users, with most reporting a positive experience. Moderating posts and approving vets was rated as easy by 99% of users. The dashboard's organization and clarity were appreciated, with 96% rating it positively. The most useful features were vet approval (80%) and post moderation (77%). Overall, 96% of users expressed satisfaction, with half giving a rating of 5 (very satisfied). The feedback highlighted the dashboard's effectiveness in managing key tasks, with minor suggestions to simplify the layout further.

Below is the link that leads to the actual charts and a sample filled user feedback form:

[7.6.9 User Feedback Form for Prototype 4 Results](#)

[7.6.10 User Feedback Form for Prototype 4 Sample Answer](#)

3.6.7 Final Prototype

Prototype 5 was mainly used to test and improve the user interface (UI) according to user feedback. This iteration refined the whole design and user experience so as to have a more fluent interaction. The review functionality was also added so that users can give feedback on their experiences, this helped to increase the credibility and user engagement of the overall system. This prototype's updates made the platform more polished and better aligned with user needs.

3.6.6.1 Final Prototype: Figma Design

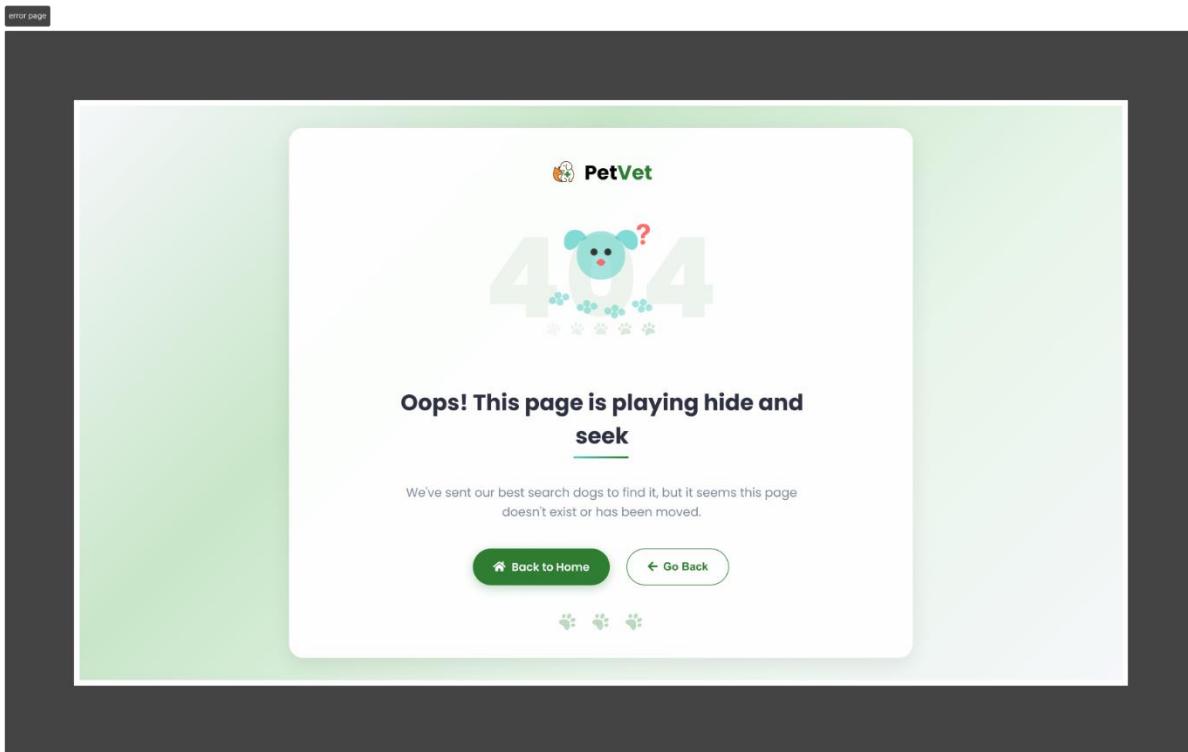


Figure 137: Main Design for Error page

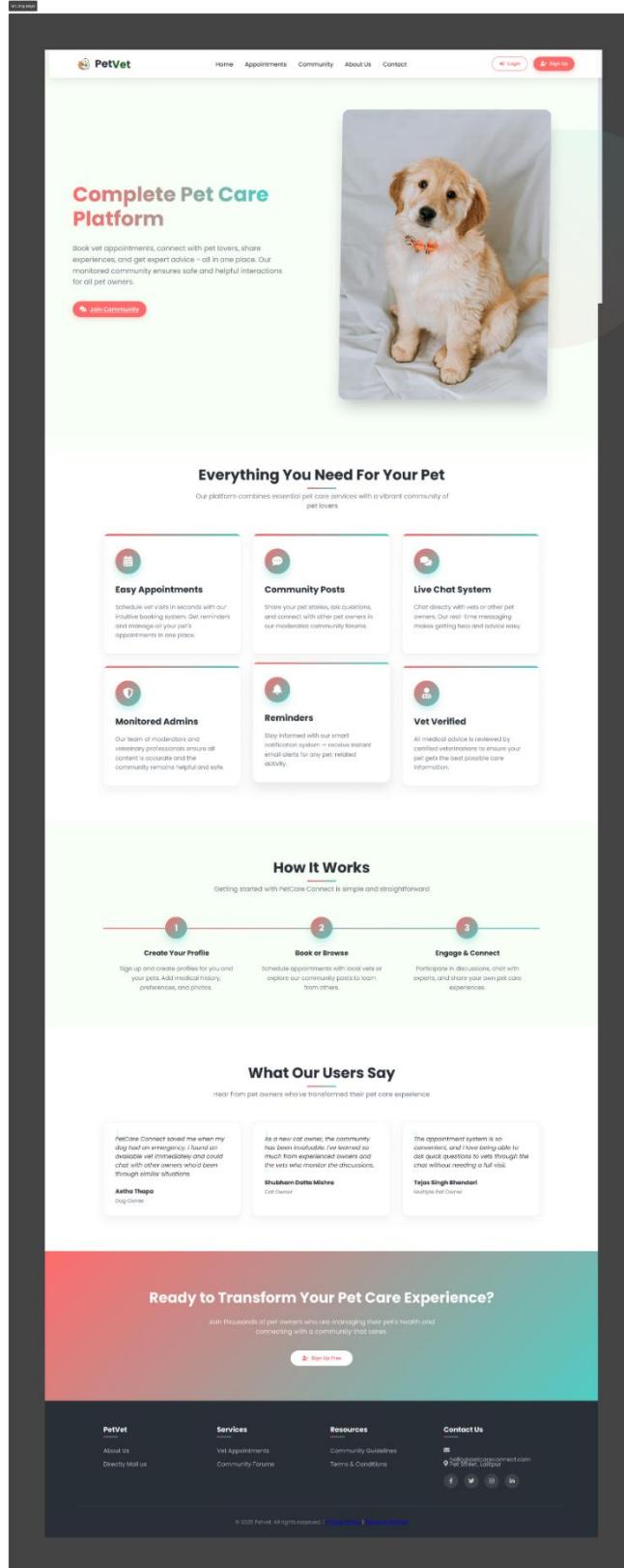


Figure 138: Main Design for Landing

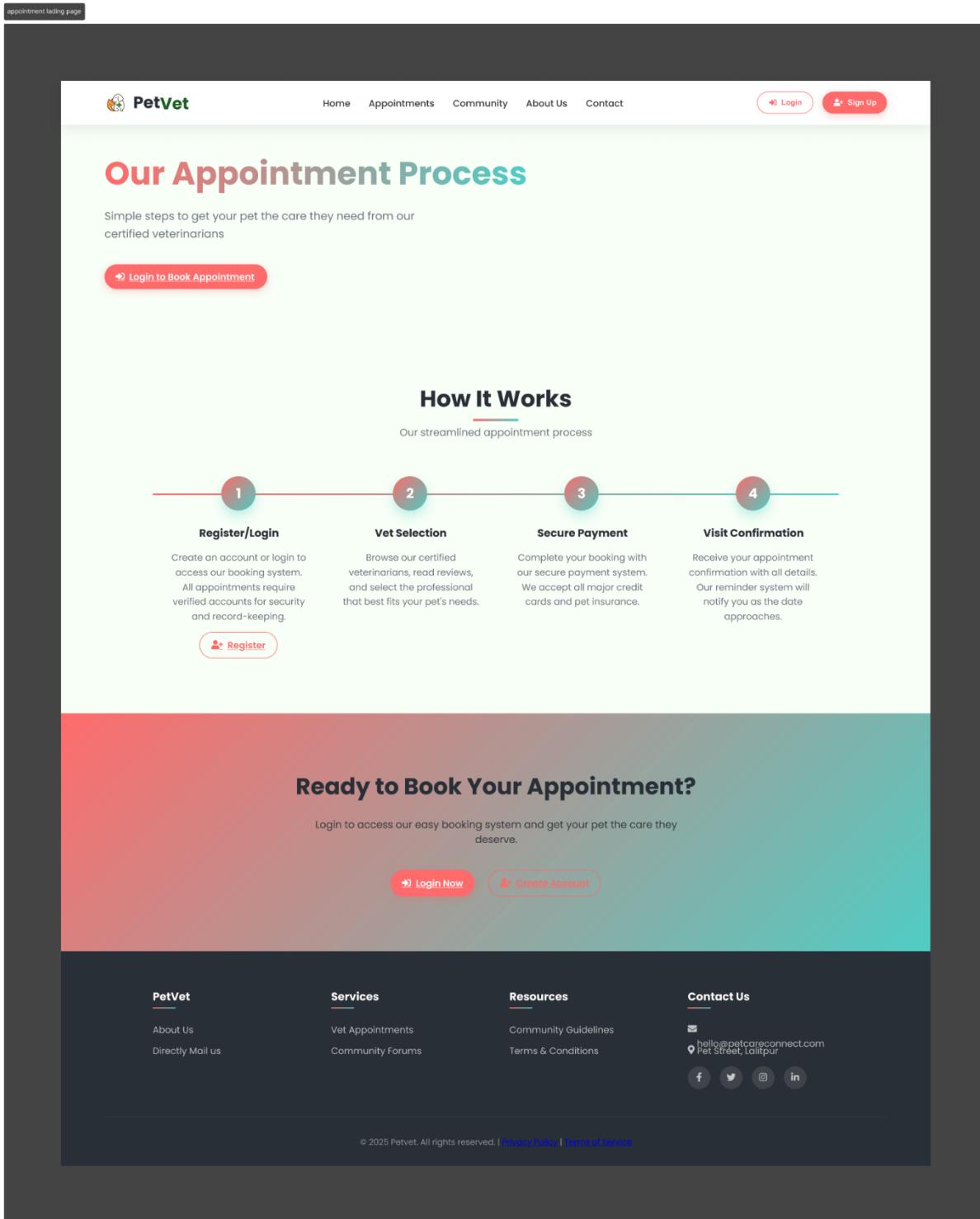


Figure 139: Main Design for Landing-appointment

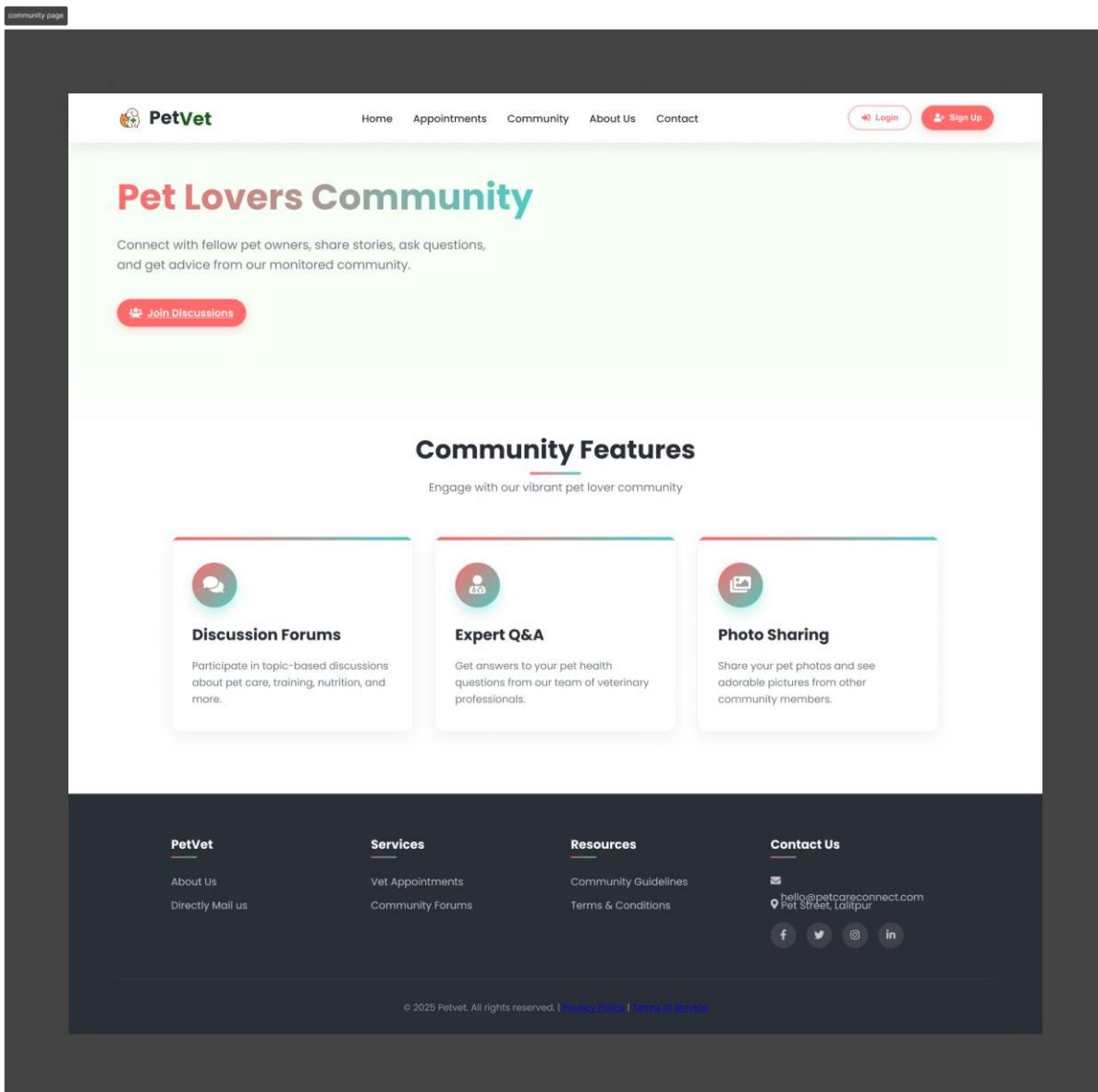


Figure 140: Main Design for Landing-community

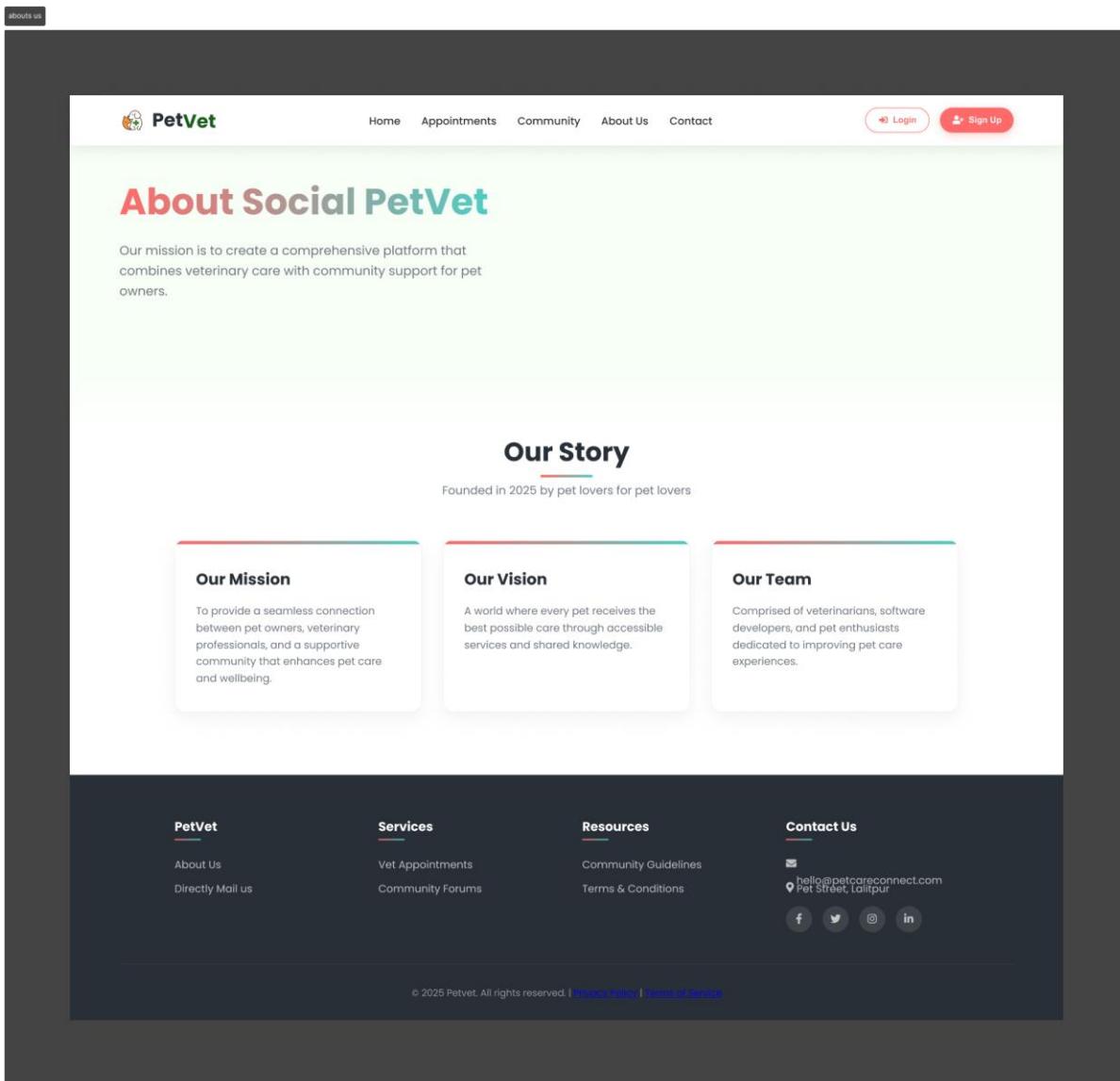


Figure 141: Main Design for Landing-about-us

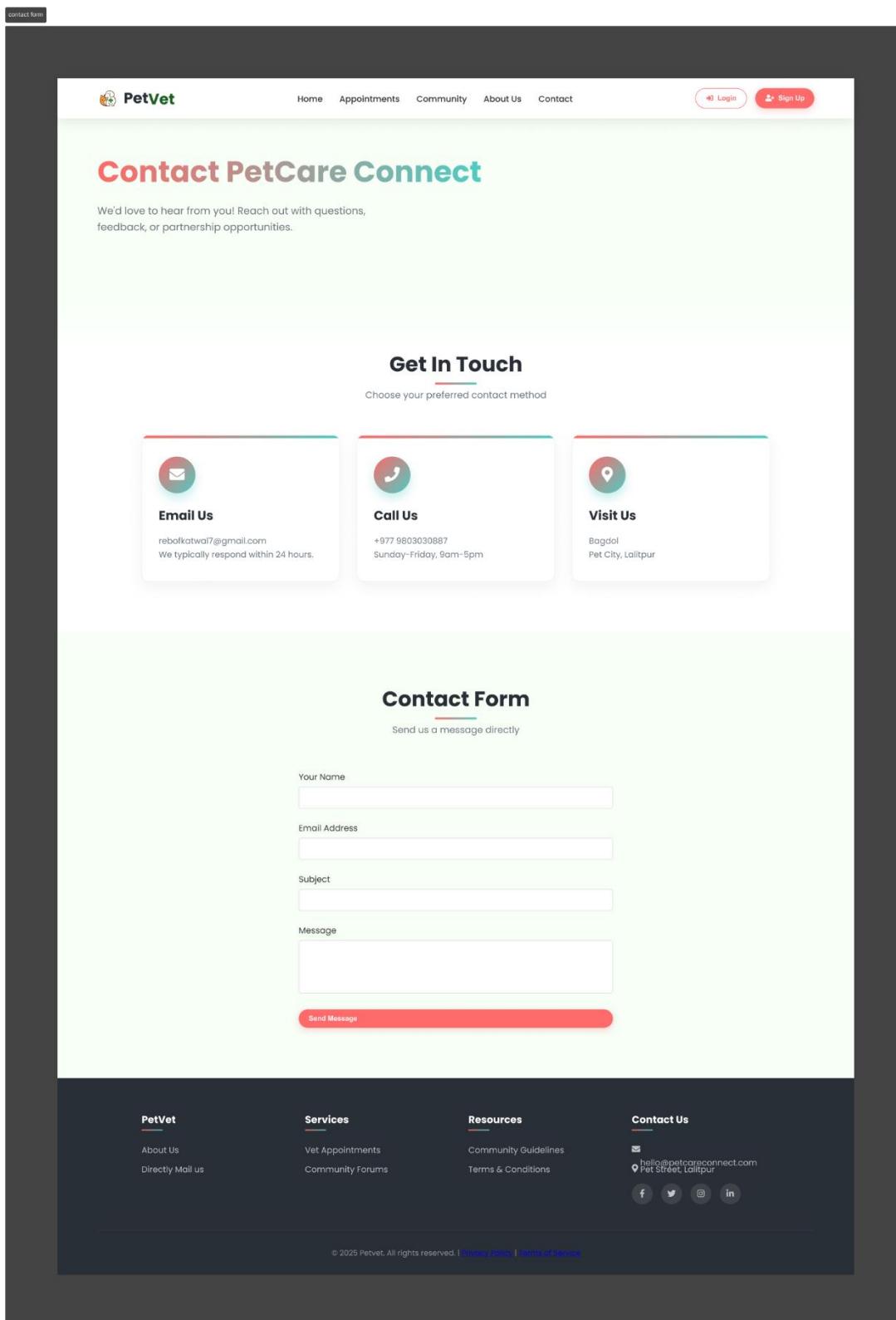


Figure 142: Main Design for Landing-contact

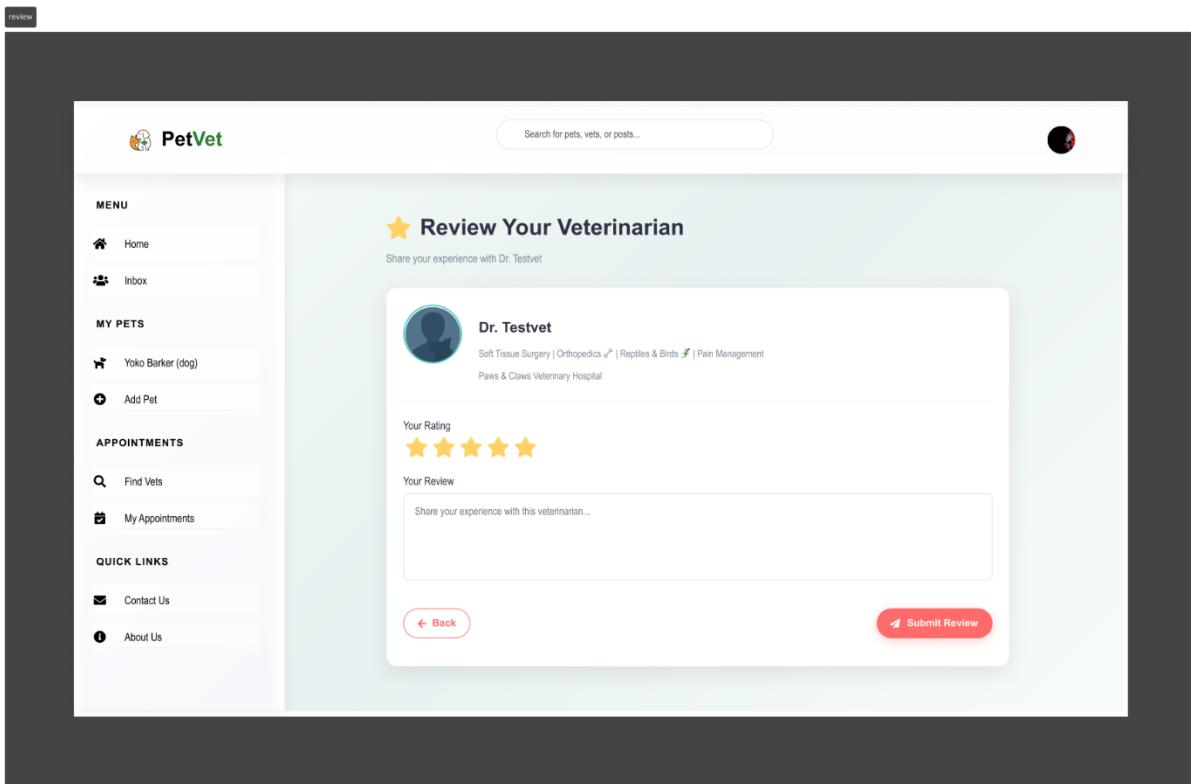


Figure 143: Main Design for Review

3.6.6.2 Final Prototype: Development

I have provided code snippets to illustrate the development process for key functionalities.

```
class CustomErrorMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        try:
            response = self.get_response(request)

            # You can handle 404 here if status_code is 404
            if response.status_code == 404:
                return render(request, 'errors/404.html', status=404)

        return response

    except Exception as e:
        # You can log the error or email the admin here
        if settings.DEBUG:
            print(traceback.format_exc())
        return render(request, 'errors/404.html', {'error': str(e)}, status=404)
```

Figure 144: Code Snippet – Error Page Rendering Logic in Final Prototype

```

def contact(request):
    if request.method == 'POST':
        # Get form data
        name = request.POST.get('name')
        email = request.POST.get('email')
        subject = request.POST.get('subject')
        message = request.POST.get('message')

        # Basic validation
        if not all([name, email, subject, message]):
            messages.error(request, "All fields are required!")
            return redirect('coreFunctions:contact')

        if '@' not in email:
            messages.error(request, "Please enter a valid email address")
            return redirect('corefunctions:contact')

    try:
        # Format the email content
        email_subject = f"Contact Form: {subject}"
        email_message = (
            f"Hello Admin,\n\n"
            f"You have received a new message through the contact form.\n\n"
            f"Name: {name}\n"
            f"Email: {email}\n"
            f"Message:\n{message}\n\n"
            f"Best regards,\n"
            f"The PetVet Website"
        )

        # Send email (consistent with your working OTP code)
        send_mail(
            email_subject,
            email_message,
            settings.DEFAULT_FROM_EMAIL, # From email

```

Figure 145: Code Snippet – Contact us Logic in Final Prototype

```
def home(request):
    return render(request, 'coreFunctions/landing.html')

def about(request):
    return render(request, 'coreFunctions/about.html')

def community(request):
    return render(request, 'coreFunctions/community.html')

def contact(request):
    return render(request, 'coreFunctions/contact.html')

def appointment(request):
    return render(request, 'coreFunctions/appointment.html')
```

Figure 146: Code Snippet – Landing pages rendering Logic in Final Prototype

```

@login_required
def review_vet(request, appointment_id):
    """View for reviewing a veterinarian after an appointment"""
    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Check if the user is authorized to review this appointment
    if request.user != appointment.pet_owner.user:
        return HttpResponseRedirect("Only the pet owner can review this appointment")

    # Check if the appointment is completed and paid
    if appointment.status != 'completed' or appointment.payment_status != 'paid':
        messages.error(request, "You can only review completed and paid appointments")
        return redirect('appointment:appointment_detail', appointment_id=appointment.id)

    # Check if the user has already reviewed this appointment
    existing_review = Review.objects.filter(
        vet=appointment.vet,
        reviewer=request.user,
        appointment=appointment
    ).first()

    if existing_review:
        messages.info(request, "You have already reviewed this appointment")
        return redirect('appointment:appointment_detail', appointment_id=appointment.id)

    if request.method == 'POST':
        # Process the review submission
        rating = int(request.POST.get('rating', 5))
        comment = request.POST.get('comment', '')

        # Create the review
        review = Review.objects.create(
            vet=appointment.vet,
            reviewer=request.user,
            rating=rating,
            comment=comment,
            appointment=appointment
        )
    
```

Figure 147: Code Snippet – Review Logic in Final Prototype

3.6.6.4 Final Prototype: Testing

In the final iteration, the review feature was also developed and unit tested. As this marked the completion of the full system, comprehensive system testing was conducted. The review feature's unit test is documented under APPT9, and the complete system was tested as outlined in the System Testing section.

4.2.17 Test: Appt10 Review Functionality

4.3 System Testing

3.6.6.5 Final Prototype: User Feedback

Below is the link that leads to the actual charts and a sample filled pre-survey form:

7.6.11 User Feedback Form for Final Prototype Results

7.6.12 User Feedback Form for Final Prototype Sample Answer

3.7 Implementation

The project was initialized using Django and version controlled using Git to start the implementation of the PetVet platform. A GitHub repository was made to store and track development progress. Trello was also used to manage tasks throughout the project. I made seven Trello cards per each phase of prototyping and designing and kept tasks unchecked until they were conquered with development.

The codebase was organized using (7.4.10 Architectural Choice) MVT architecture of Django. All the frontend files were kept under a single templates folder and the sub folders were created within the templates folder based on the app name to make things clear and modular. We created four separate Django apps based on key features: user authentication, appointments, community/forum and admin dashboard. The project was kept clean and maintainable by keeping each app in charge of its own models, views, forms and templates.

At first, the database used was SQLite for simplicity of setup during early development and testing. The database was switched from SQLite to PostgreSQL as the project matured for better scalability and performance. Visual Studio Code (VS Code) was used to develop and it was an efficient environment to write and debug the code. Changes were regularly committed and pushed to the GitHub repository with a clear history of changes. This structured approach helped in systematic, traceable and efficient development of the project.

3.7.1 Trello

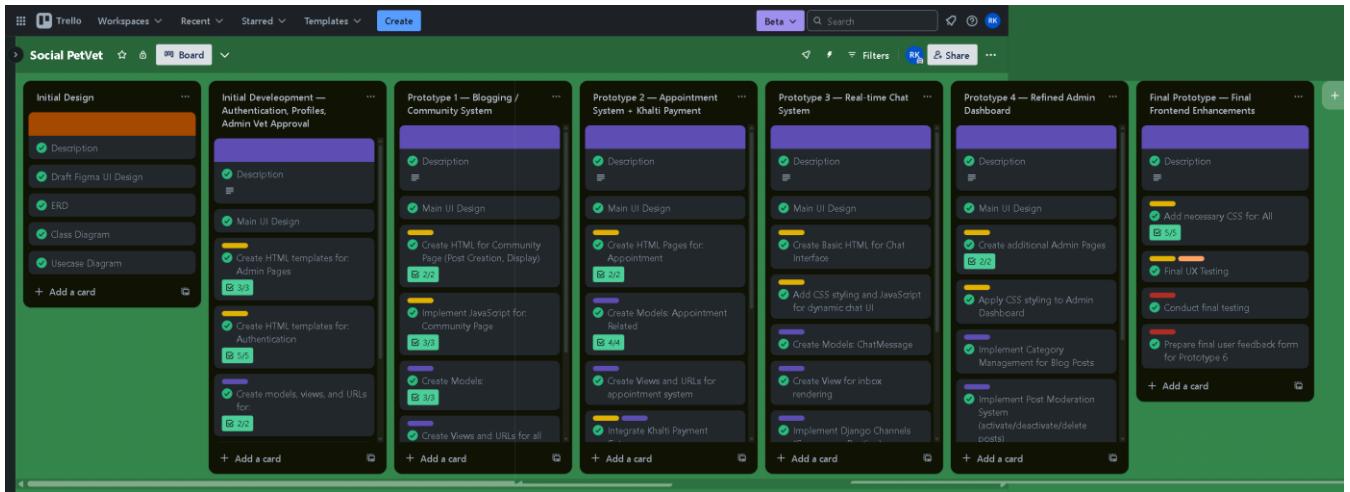


Figure 148: Entire Trello Board

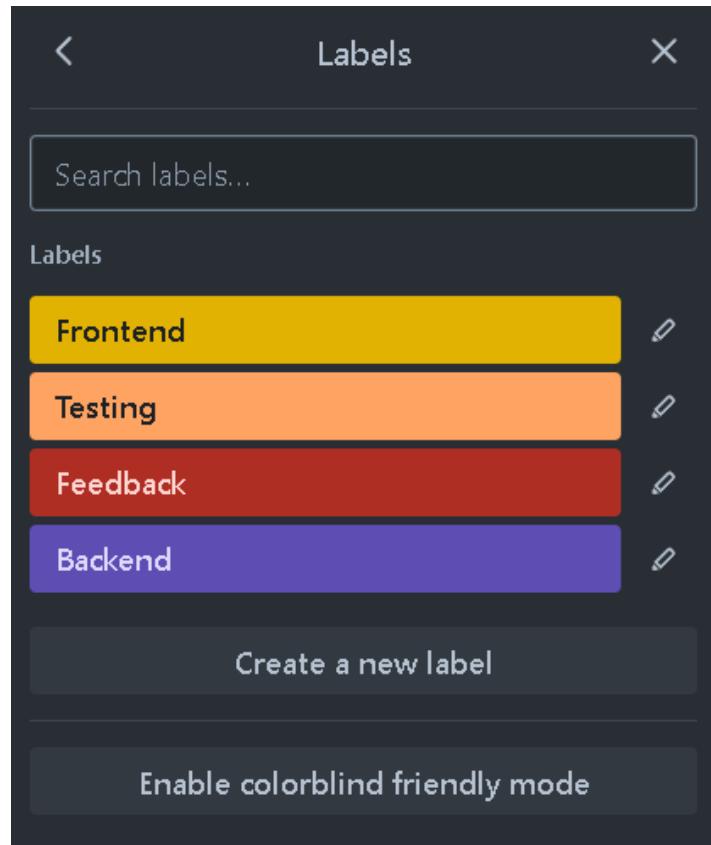


Figure 149: Labels used in trello

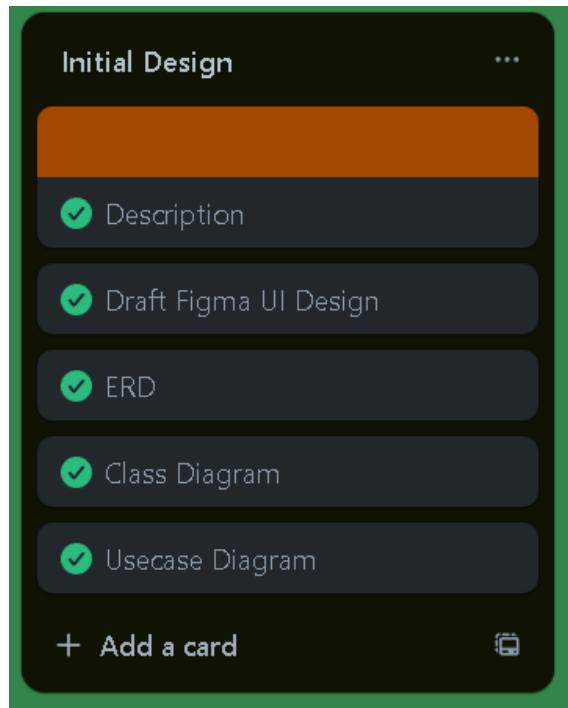


Figure 150: Trello Initial Design Card

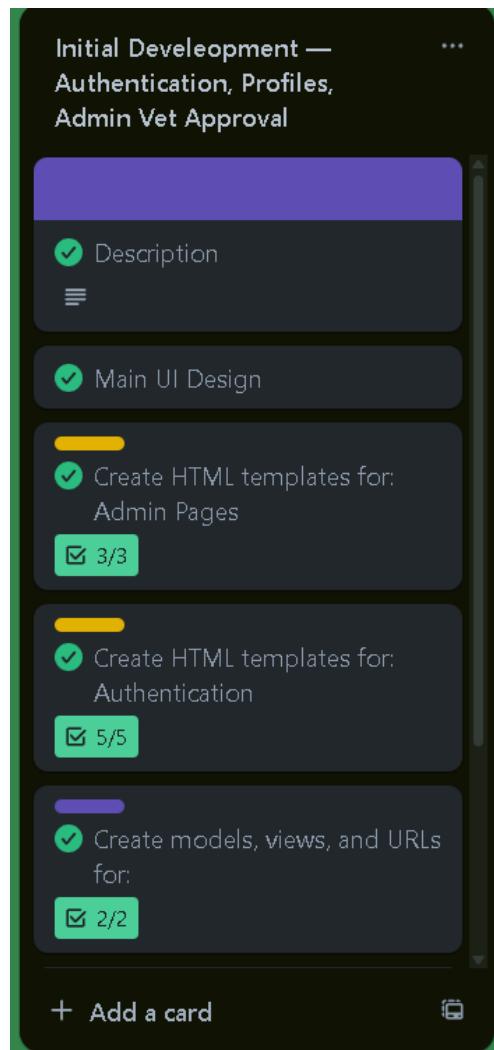


Figure 151: Trello Initial Development Card

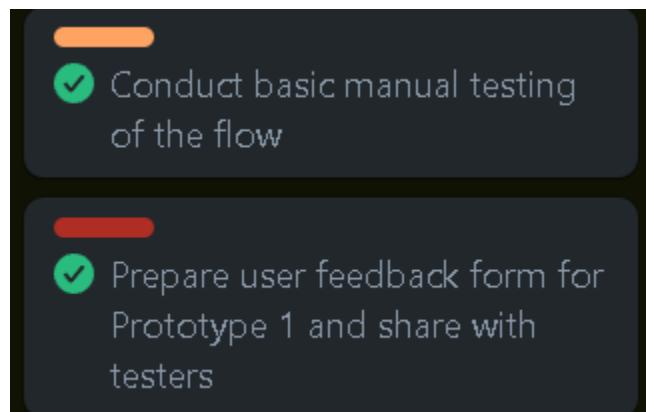


Figure 152: Trello Initial Development Card 2

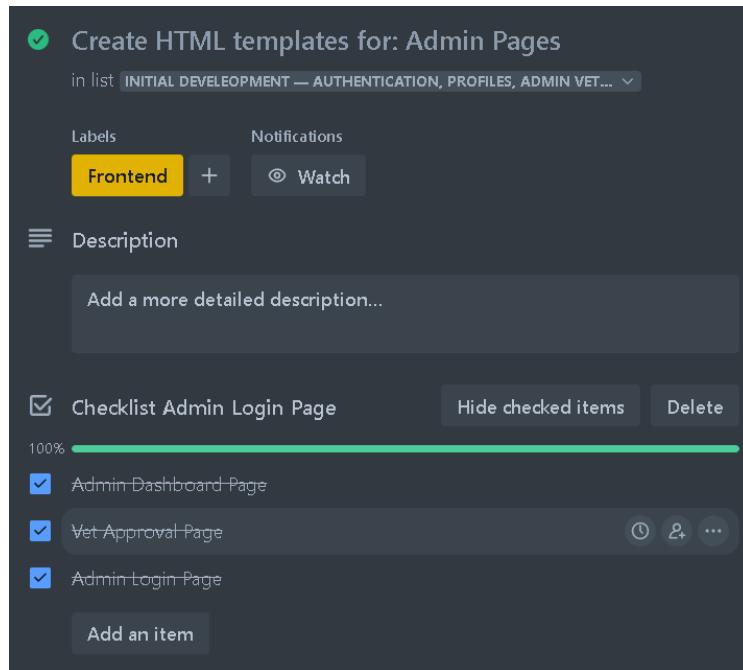


Figure 153: Trello Initial Development Card Checklist 1

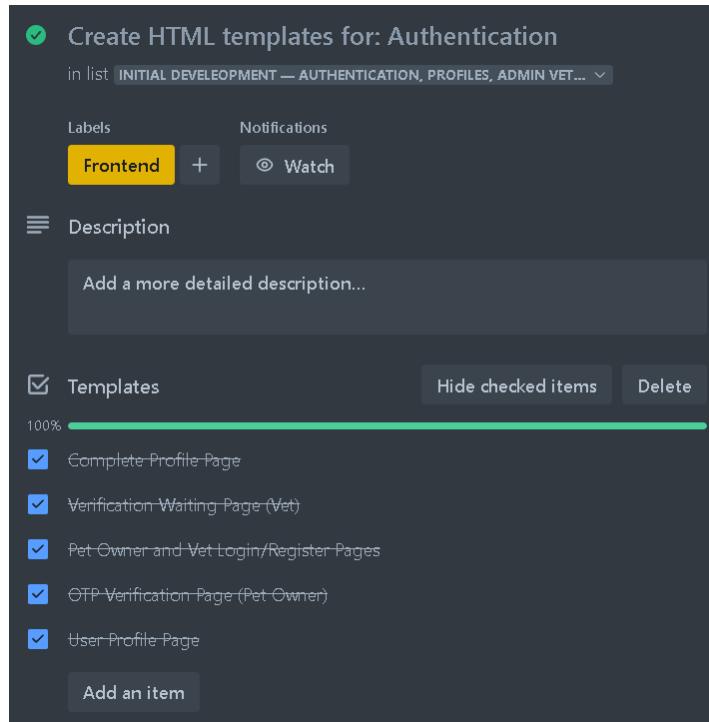


Figure 154: Trello Initial Development Card Checklist 2

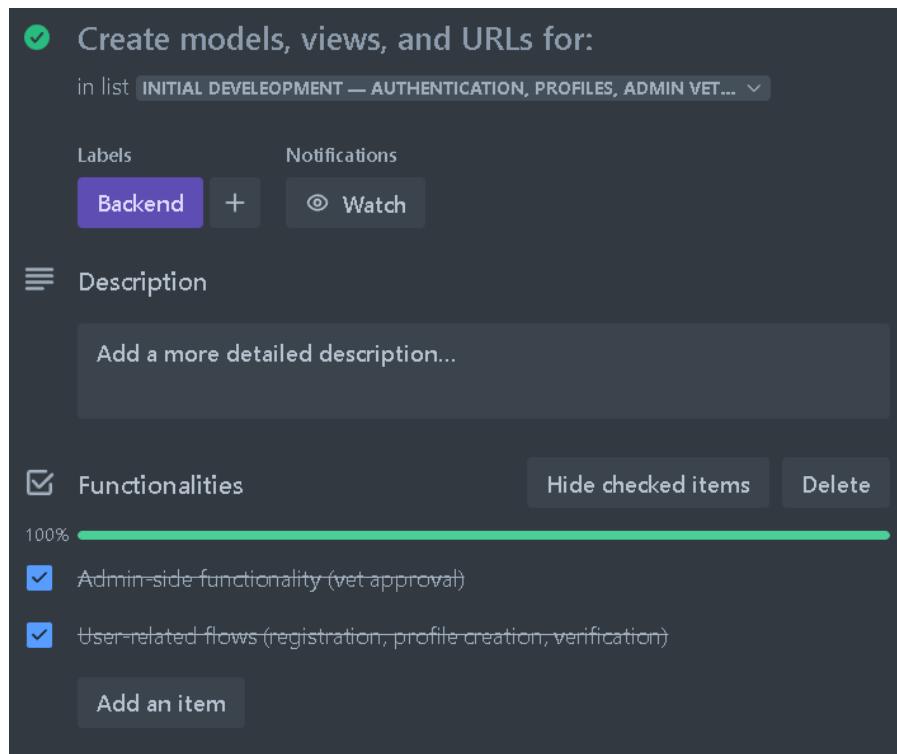


Figure 155: Trello Initial Development Card Checklist 3

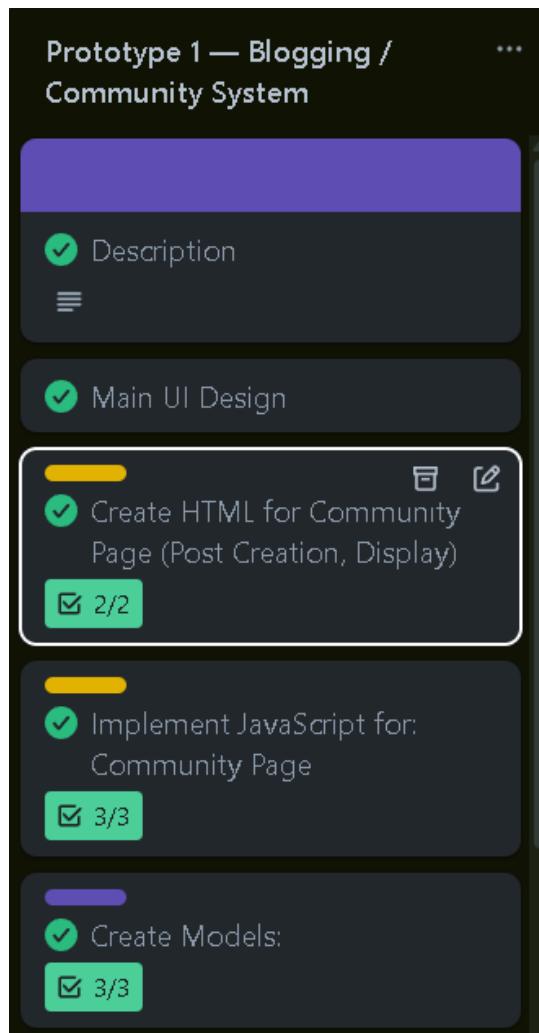


Figure 156: Trello Prototype 1 Card

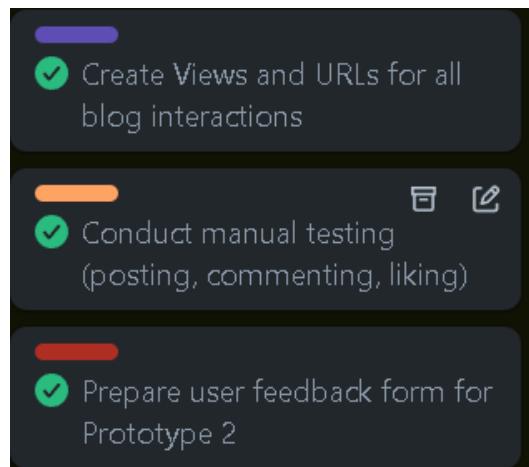


Figure 157: Trello Prototype 1 Card 2

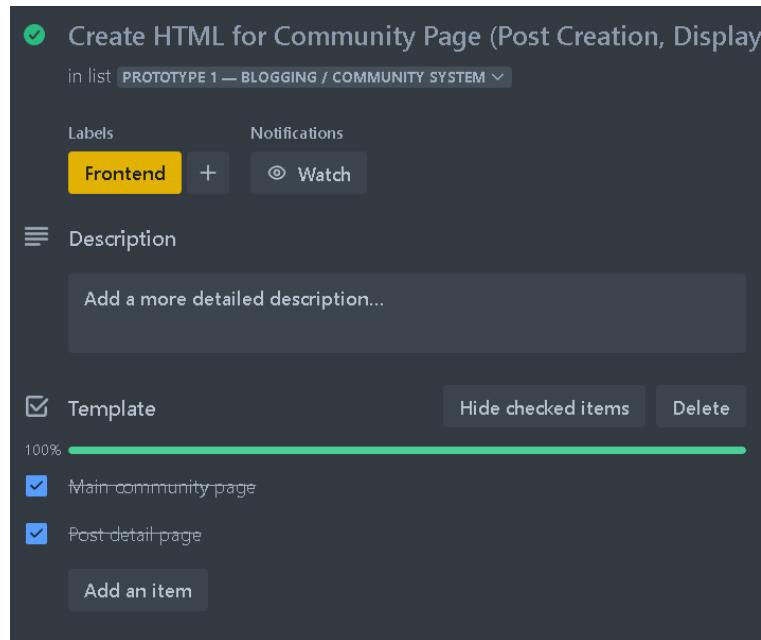


Figure 158: Trello Prototype 1 Checkout 1

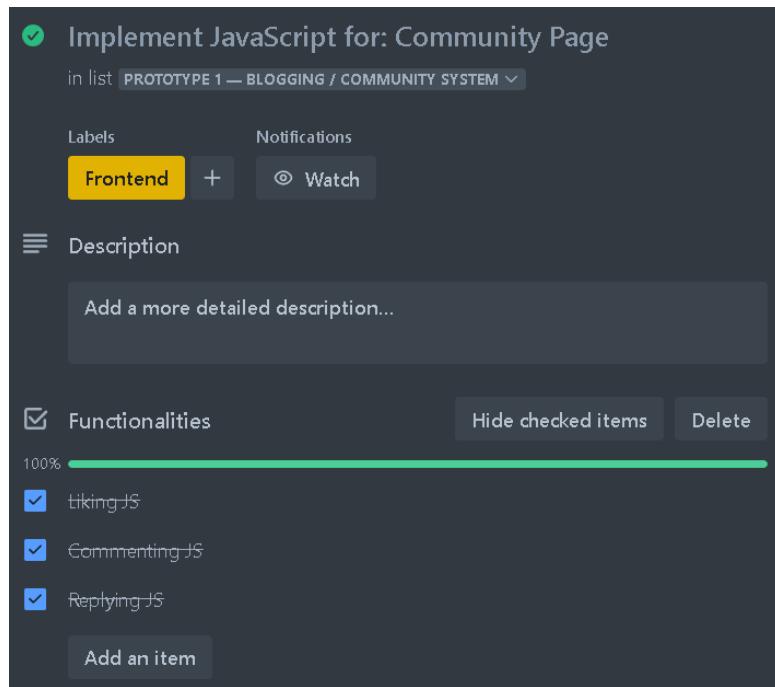


Figure 159: Trello Prototype 1 Checkout 2

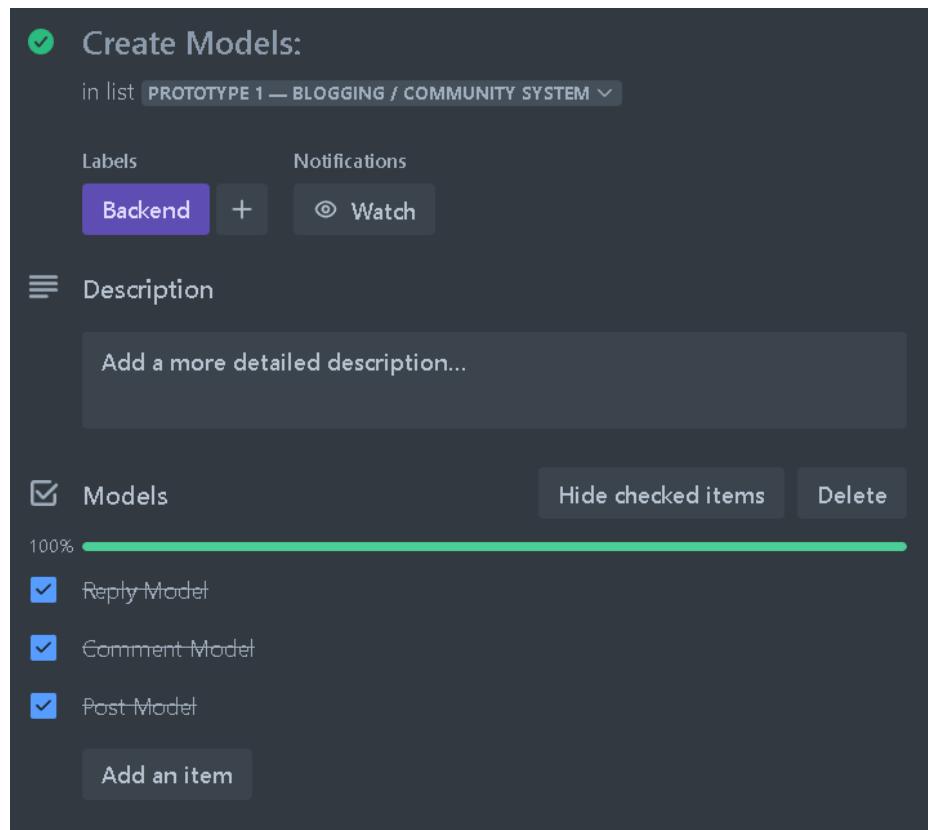


Figure 160: Trello Prototype 1 Card Checklist 1

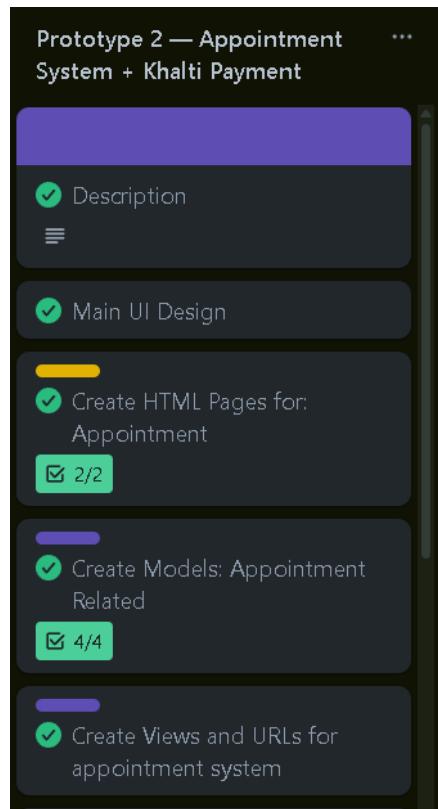


Figure 161: Trello Prototype 2 Card

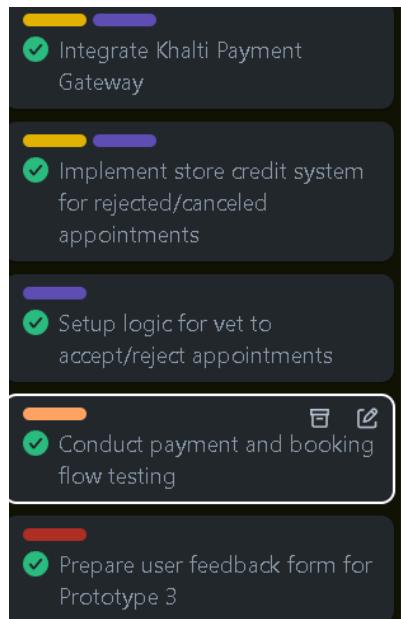


Figure 162: Trello Prototype 2 Card 2

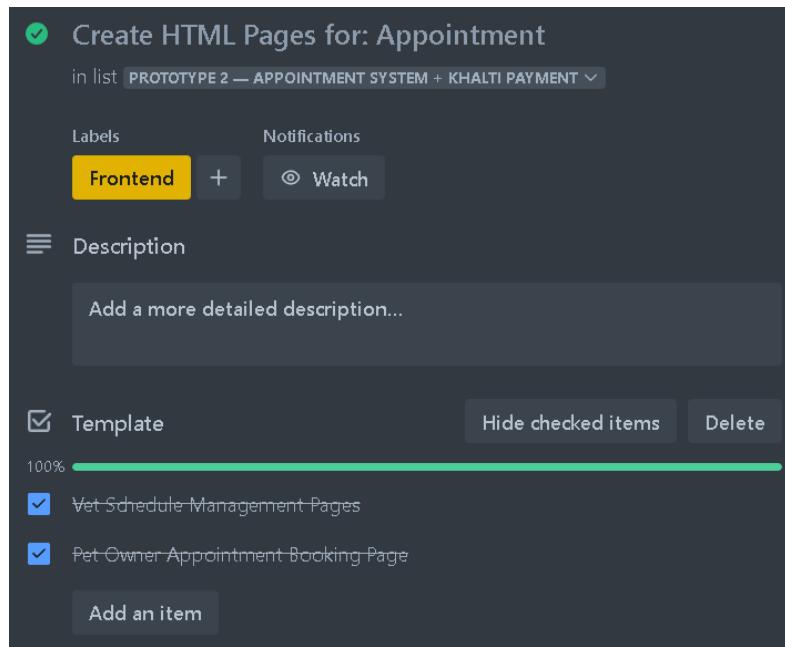


Figure 163: Trello Prototype 1 Card Checklist 1

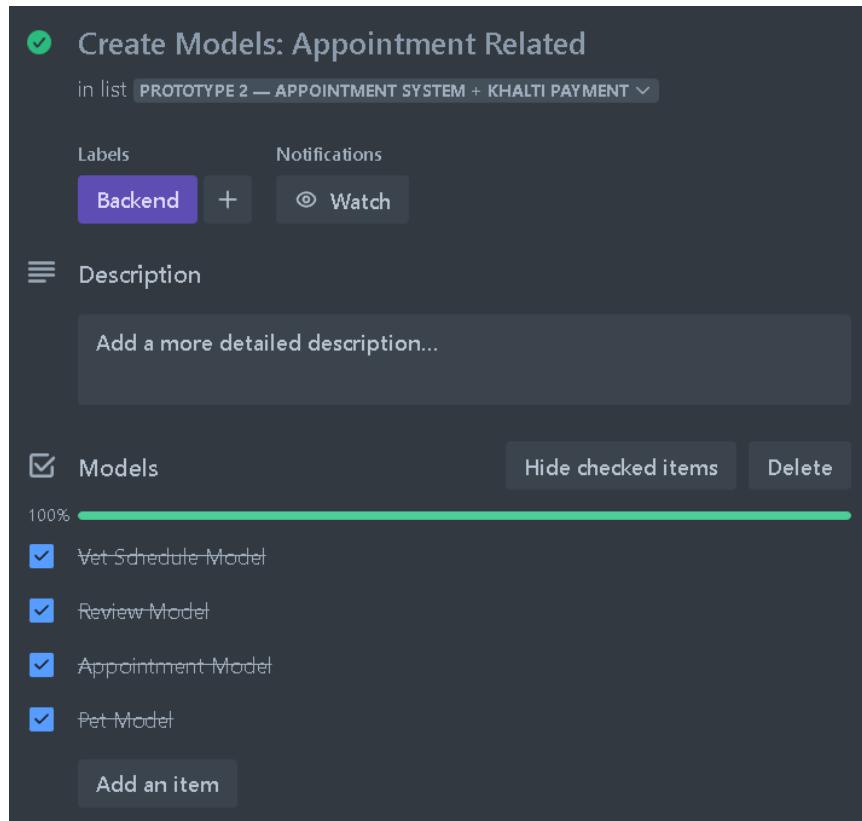


Figure 164: Trello Prototype 2 Checklist 1

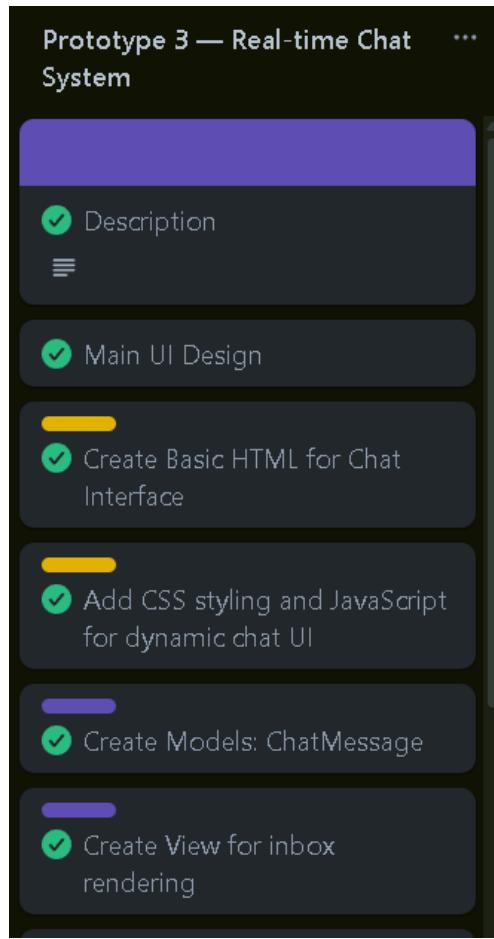


Figure 165: Trello Prototype 3 Card

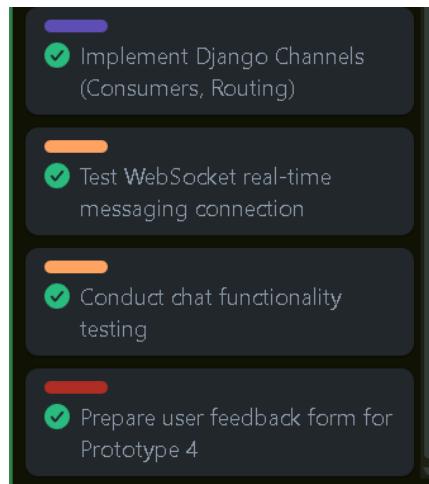


Figure 166: Trello Prototype 3 Card 2

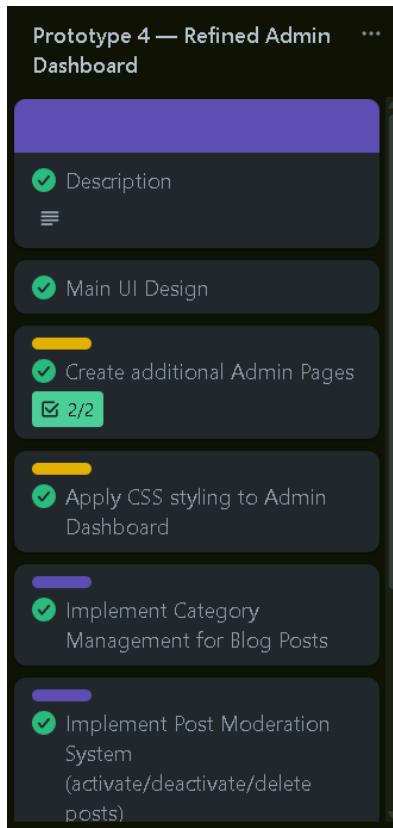


Figure 167: Trello Prototype 4 Card

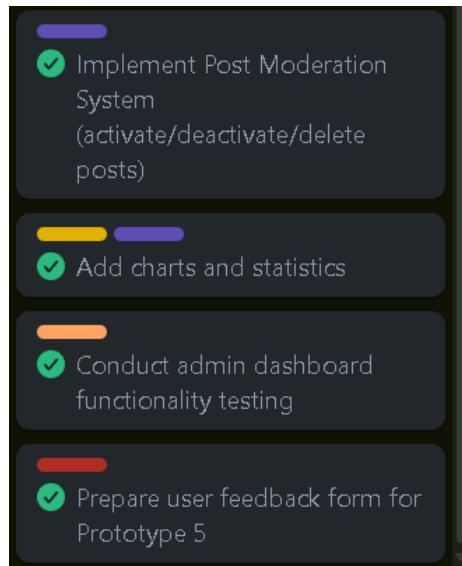


Figure 168: Trello Prototype 4 Card 2

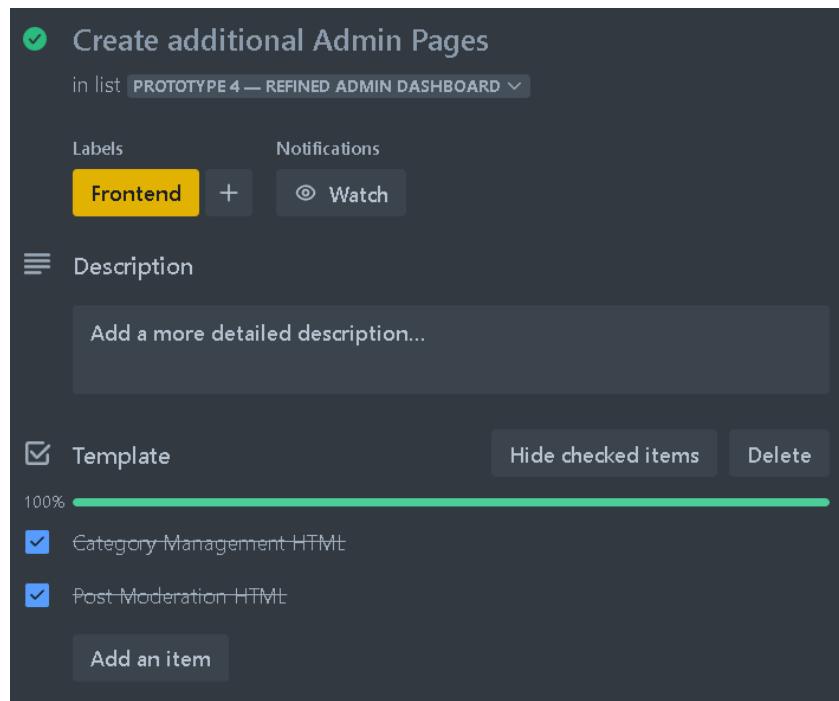


Figure 169: Trello Prototype 4 Card Checklist 1

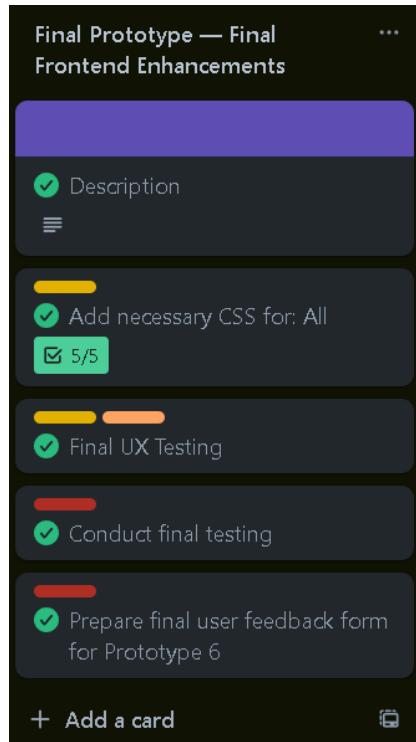


Figure 170: Trello Final Prototype Card

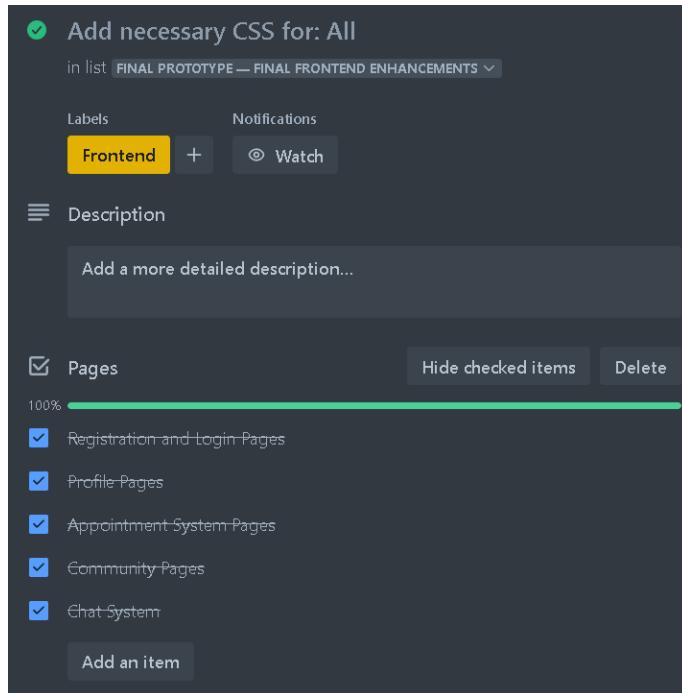


Figure 171: Trello Final Prototype Card Checklist 1

3.7.2 Github

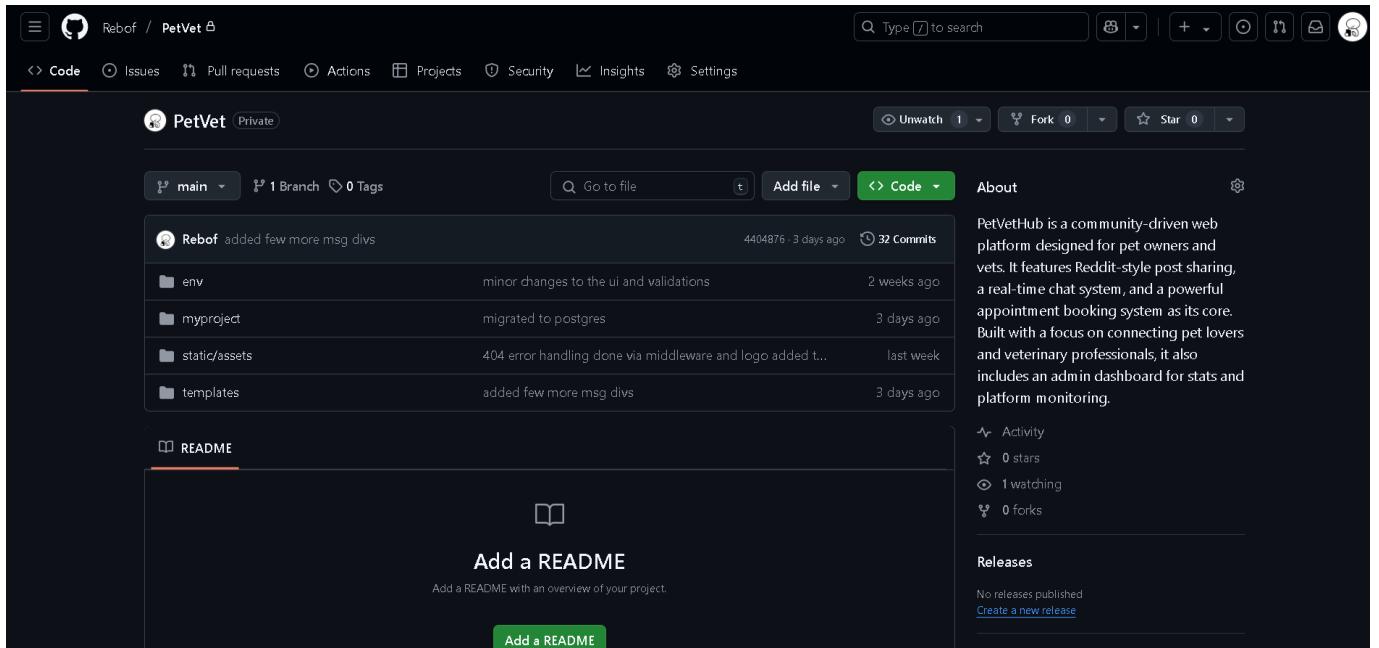


Figure 172: Screenshot of GitHub Repository – PetVet Project

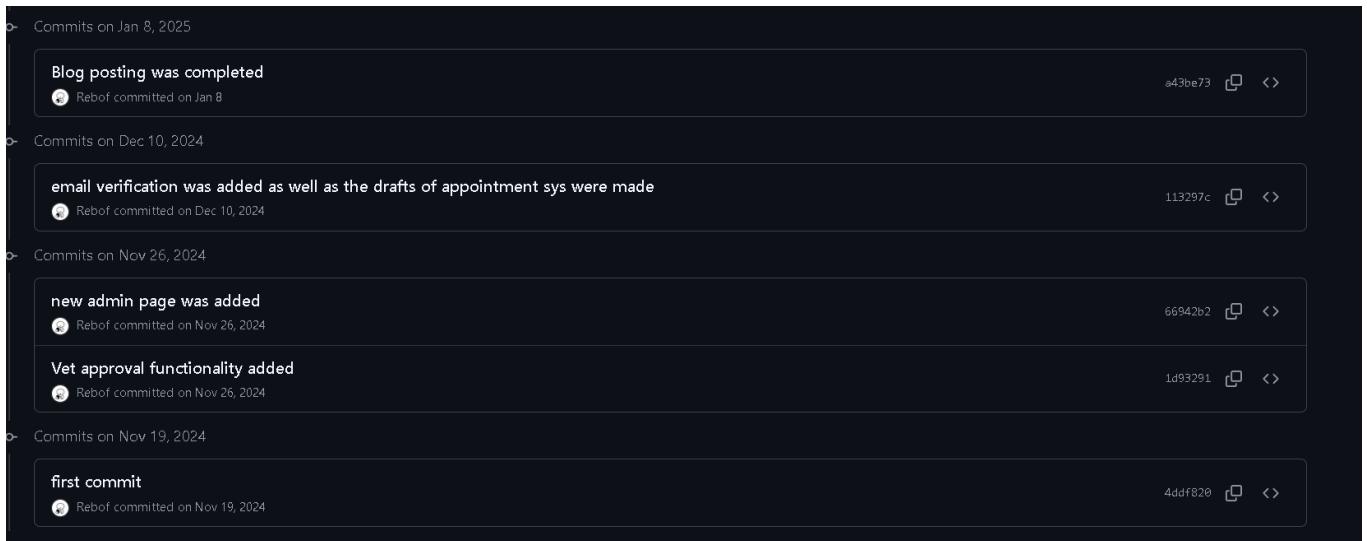


Figure 173: Snippet of Git Commit History for the PetVet Project

3.7.3 Postgres Connection

```
Server [localhost]:  
Database [postgres]:  
Port [5432]:  
Username [postgres]: postgres  
Password for user postgres:  
  
psql (17.4)  
WARNING: Console code page (437) differs from Windows code page (1252)  
          8-bit characters might not work correctly. See psql reference  
          page "Notes for Windows users" for details.  
Type "help" for help.  
  
postgres=#  
postgres=# CREATE DATABASE petvet_db;  
CREATE DATABASE  
postgres=# CREATE USER petvet_user WITH PASSWORD 'petvetpw';  
CREATE ROLE  
postgres=# ALTER ROLE petvet_user SET client_encoding TO 'utf8';  
ALTER ROLE  
postgres=# ALTER ROLE petvet_user SET default_transaction_isolation TO 'read committed';  
ALTER ROLE  
postgres=# ALTER ROLE petvet_user SET timezone TO 'Asia/Kathmandu';  
ALTER ROLE  
postgres=# GRANT ALL PRIVILEGES ON DATABASE petvet_db TO petvet_user;  
GRANT  
postgres=#
```

Figure 174: Screenshot of giving permission in Postgres

3.7.4 Folder Structure

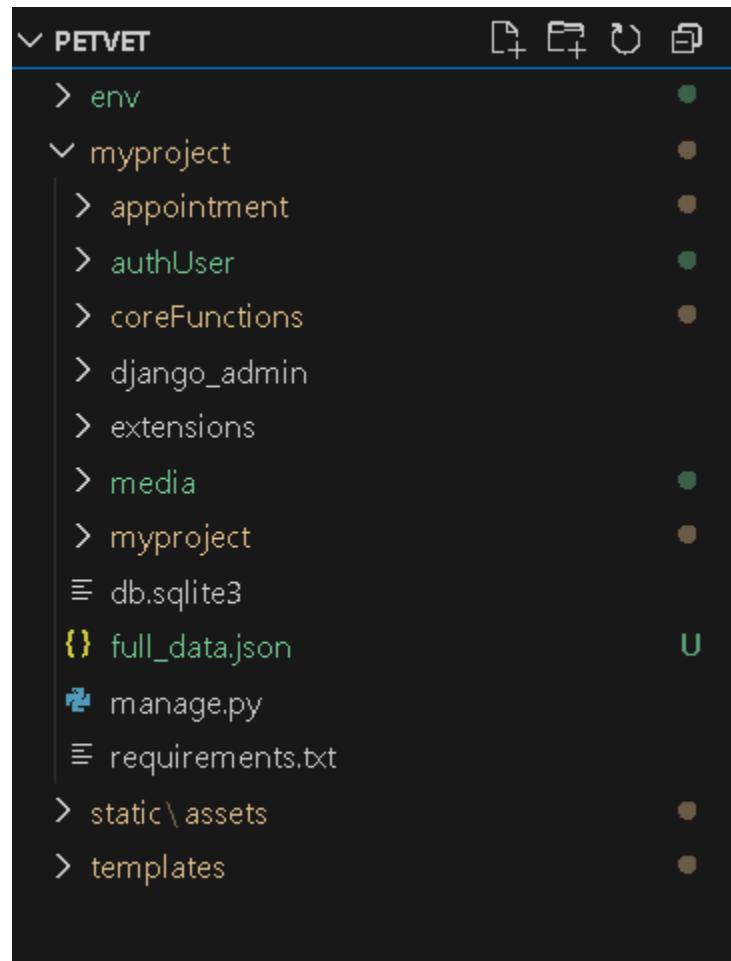


Figure 175: Folder Structure

Chapter 4: Testing and Analysis

4.1 Test Plan

4.1.1 Unit Testing Test Plan

Unit testing is the process of testing of the smallest individual component of the software system that is usually modules or functions to ensure that they fulfill their intended purpose. (Runeson, 2006).

After the completion of each prototype, unit testing will be done iteratively. Each development phase will have its related and core functions that are tested to ensure correctness and stability for that phase. The unit tests will be written and executed using Django's built in test.py framework and the TestCase class to test functionality, input validation, and expected outputs.

4.1.2 System Testing Test Plan

The testing phase that is performed on the fully integrated system to verify that it satisfies the specified requirements is system testing. It is followed by unit, component and integration testing. All the interconnected components that work together to give the desired functionality and features constitute a system. For the system to be effectively tested, it is important to know how the product works as a whole (Nirmala & Dhivya, 2018).

After the fourth prototype, the system will be tested and the final iteration before the complete prototype will be built. Testing will include the entire workflow of the application from user login and registration to access and operations in the admin dashboard.

4.2 Unit Testing

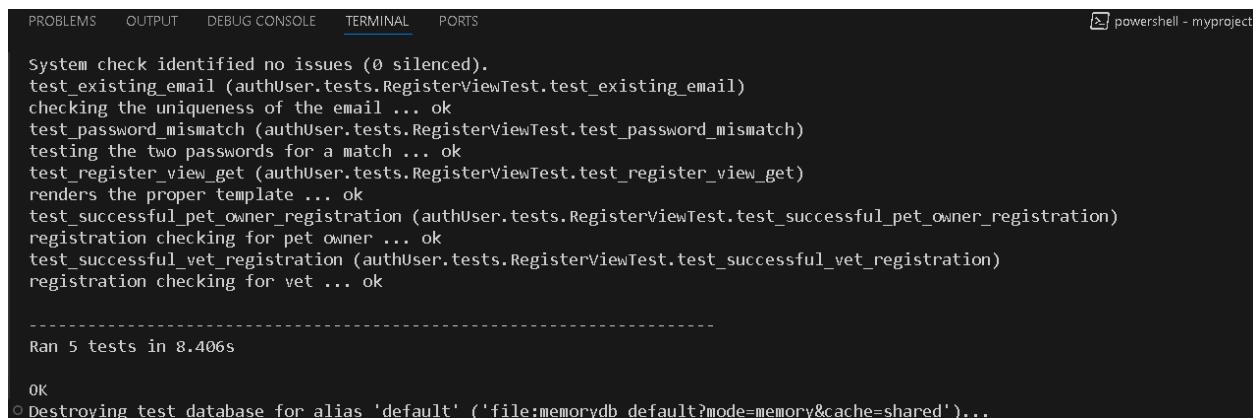
4.2.1 Test: Auth1 Registration Functionality

Test No.	Auth1
Objectives:	Test user registration functionality including validation, user creation, profile creation, and email notification
Test Cases:	<ol style="list-style-type: none"> 1. GET Request: Verify register page renders correctly 2. Pet Owner Registration: Verify user/profile creation + OTP email 3. Vet Registration: Verify vet profile creation 4. Password Mismatch: Validate error handling 5. Existing Email: Verify duplicate email check
Actions:	<ol style="list-style-type: none"> 1. GET Request: Sent GET request to register page 2. Pet Owner Registration: <ul style="list-style-type: none"> • Sample data: <pre> full_name: "Test User", email: "rebof@example.com", username: "testuser", phone: "1234567890", gender: "male", user_type: "pet_owner", password1: "testpassword123", </pre>

	<p>password2: "testpassword123"}</p> <ul style="list-style-type: none">• Sent POST request with data• Verified database records• Checked for OTP email to test@example.com <p>3. Vet Registration:</p> <ul style="list-style-type: none">• Sample data: {full_name: "Vet User", email: "votrebof@example.com", username: "testvet", phone: "9876543210", gender: "female", user_type: "vet", password1: "vetpassword123", password2: "vetpassword123"}• Sent POST request with data• Verified VetProfile creation <p>4. Password Mismatch:</p> <p>Sample data:</p> <p>{password1: " testpassword123", password2: " wrongpassword"}</p> <ul style="list-style-type: none">• Sent POST request• Verified error message
--	--

	<p>5. Existing Email:</p> <ul style="list-style-type: none"> • Pre-created user: email="existing@example.com" • Sample data: {email: "existing@example.com"} • Sent POST request • Verified duplicate error
Expected Results	<ul style="list-style-type: none"> • GET: Register template rendered successfully • Pet Owner: User + PetOwnerProfile created, OTP email sent, redirect to profile completion • Vet: User + VetProfile created, no OTP email • Validation: "Passwords do not match" error displayed • Existing Email: "Email already exists" error displayed
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 13:Unit Test - Registration



The screenshot shows a terminal window with the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject

System check identified no issues (0 silenced).
test_existing_email (authUser.tests.RegisterViewTest.test_existing_email)
    checking the uniqueness of the email ... ok
test_password_mismatch (authUser.tests.RegisterViewTest.test_password_mismatch)
    testing the two passwords for a match ... ok
test_register_view_get (authUser.tests.RegisterViewTest.test_register_view_get)
    renders the proper template ... ok
test_successful_pet_owner_registration (authUser.tests.RegisterViewTest.test_successful_pet_owner_registration)
    registration checking for pet owner ... ok
test_successful_vet_registration (authUser.tests.RegisterViewTest.test_successful_vet_registration)
    registration checking for vet ... ok

-----
Ran 5 tests in 8.406s

OK
	Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 176: Unit Test - Registration result

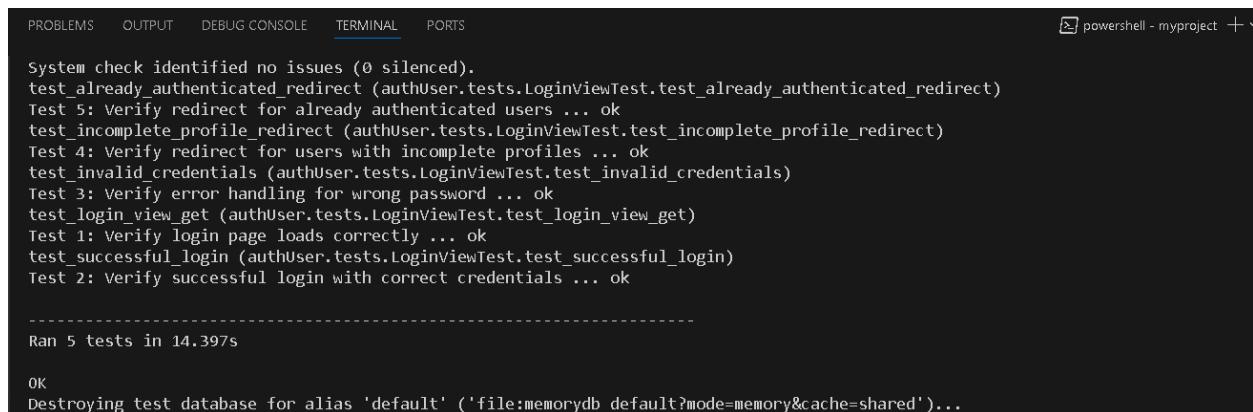
4.2.2 Test: Auth2 Login Functionality

Test No.	Auth2
Objectives:	Test user login functionality including authentication, redirects, and error handling
Test Cases:	<ol style="list-style-type: none"> 1. GET Request: Verify login page renders correctly 2. Successful Login: Verify authentication and redirect 3. Invalid Credentials: Verify error message 4. Incomplete Profile: Verify redirect to profile completion 5. Authenticated Redirect: Verify redirect for logged-in users
Actions:	<ol style="list-style-type: none"> 1. GET Request: Sent GET request to login page 2. Successful Login: <ul style="list-style-type: none"> • Sample data: {email: "rebof@gmail.com", password: "testkatwal123",} • Sent POST request with data • Verified redirect to homepage 3. Invalid Credentials:

	<ul style="list-style-type: none"> • Sample data: <code>{email: "rebof@gmail.com", password: "wrongpassword",}</code> • Sent POST request with data • Verified error message <p>4. Incomplete Profile:</p> <ul style="list-style-type: none"> • Pre-created user: <code>email="existing@example.com", profile_completed=False</code> • Sample data: <code>{email: "existing@example.com", Password: "testpass123"}</code> • Sent POST request • Verified redirect to profile completion <p>5. Authenticated Redirect:</p> <ul style="list-style-type: none"> • Logged in test user • Sent GET request to login page • Verified redirect to homepage
Expected Results	<ul style="list-style-type: none"> • GET: Login template rendered successfully • Successful Login: User authenticated, redirect to homepage

	<ul style="list-style-type: none"> • Invalid Credentials: "Invalid username or password" error displayed • Incomplete Profile: Redirect to profile completion page • Authenticated Redirect: Immediate redirect to homepage
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 14:Unit Test – Login



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject + v

System check identified no issues (0 silenced).
test_already_authenticated_redirect (authUser.tests.LoginViewTest.test_already_authenticated_redirect)
Test 5: verify redirect for already authenticated users ... ok
test_incomplete_profile_redirect (authUser.tests.LoginViewTest.test_incomplete_profile_redirect)
Test 4: verify redirect for users with incomplete profiles ... ok
test_invalid_credentials (authUser.tests.LoginViewTest.test_invalid_credentials)
Test 3: verify error handling for wrong password ... ok
test_login_view_get (authUser.tests.LoginViewTest.test_login_view_get)
Test 1: verify login page loads correctly ... ok
test_successful_login (authUser.tests.LoginViewTest.test_successful_login)
Test 2: verify successful login with correct credentials ... ok

-----
Ran 5 tests in 14.397s
OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

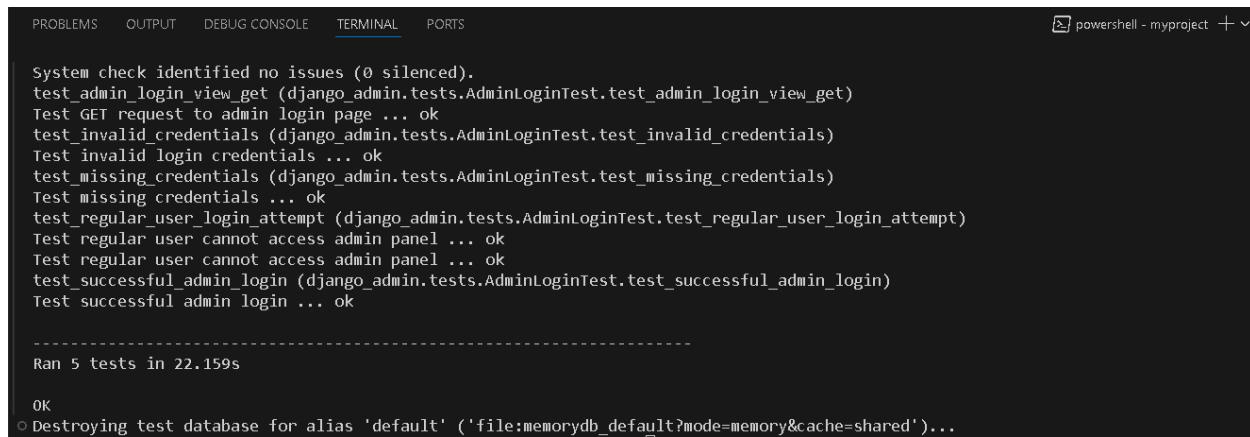
4.2.3 Test: Auth3 Admin Login Functionality

Test No.	Auth3
Objectives:	Test admin login functionality including authentication, redirects, and error handling
Test Cases:	<ol style="list-style-type: none"> 1. GET Request: Verify admin login page renders correctly 2. Successful Login: Verify authentication and redirect 3. Invalid Credentials: Verify error message 4. Missing Credentials: Verify error message 5. Regular User Login Attempt: Verify redirect for logged-in users
Actions:	<ol style="list-style-type: none"> 1. GET Request: Sent GET request to admin login page 2. Successful Login: <ul style="list-style-type: none"> • Sample data: {email: "admin@gmail.com", password: "admin123",} • Sent POST request with data

	<ul style="list-style-type: none">• Verified redirect to homepage <p>3. Invalid Credentials:</p> <ul style="list-style-type: none">• Sample data: <code>{email: "admin@gmail.com", password: "wrongpassword",}</code>• Sent POST request with data• Verified error message <p>4. Missing Credentials:</p> <ul style="list-style-type: none">• Sample data: <code>{email: " ", password: " ",}</code>• Sent POST request with data• Verified error message <p>5. Regular User Attempt:</p> <ul style="list-style-type: none">• Sample data: <code>{email: "user@gmail.com ", password: "user123 ",}</code>• Sent POST request with data
--	--

	<ul style="list-style-type: none"> Verified error message
Expected Results	<ul style="list-style-type: none"> Login page rendered on GET Admin successfully logged in and redirected to dashboard Invalid login shows proper error message Missing credentials show proper error message Regular user denied access to admin dashboard
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 15:Unit Test - Admin Login



```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
powershell - myproject + v

System check identified no issues (0 silenced).
test_admin_login_view_get (django_admin.tests.AdminLoginTest.test_admin_login_view_get)
Test GET request to admin login page ... ok
test_invalid_credentials (django_admin.tests.AdminLoginTest.test_invalid_credentials)
Test invalid login credentials ... ok
test_missing_credentials (django_admin.tests.AdminLoginTest.test_missing_credentials)
Test missing credentials ... ok
test_regular_user_login_attempt (django_admin.tests.AdminLoginTest.test_regular_user_login_attempt)
Test regular user cannot access admin panel ... ok
Test regular user cannot access admin panel ... ok
test_successful_admin_login (django_admin.tests.AdminLoginTest.test_successful_admin_login)
Test successful admin login ... ok

-----
Ran 5 tests in 22.159s
OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 177: Unit Test - Login result

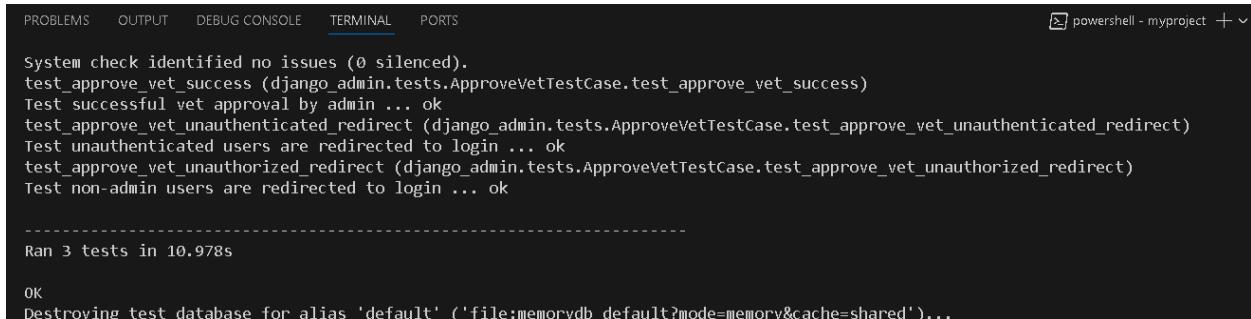
4.2.4 Test: Admin1 Vet Approval Functionality

Test No.	Admin1
Objectives:	Test vet approval functionality by admin, including successful approval, unauthorized access, unauthenticated access, and email failure handling.
Test Cases:	<ol style="list-style-type: none"> 1. Successful Vet Approval 2. Unauthorized Approval Attempt by Regular User 3. Unauthenticated Approval Attempt 4. Vet Approval with Email Failure
Actions:	<p>1. Successful Vet Approval:</p> <ul style="list-style-type: none"> • Sample data: Admin User with <code>is_staff</code>: True and Vet User with <code>status_verification=false</code> • Admin logs in • Sends POST request to <code>approve_vet</code> view • Verifies redirect to vet approvals list • Checks vet is marked as verified <p>2. Unauthorized Approval Attempt:</p>

	<ul style="list-style-type: none"> • Sample data: Regular User with is_staff=False and Vet User with status_verification=false • Regular user logs in • Sends POST request to approve_vet • Verifies redirection to admin login • Ensures vet is not approved <p>3. Unauthenticated Approval</p> <p>Attempt:</p> <ul style="list-style-type: none"> • Sends POST request to approve_vet without logging in • Verifies redirection to admin login • Ensures vet is not approved
Expected Results	<ul style="list-style-type: none"> • Vet is successfully approved by admin • Unauthorized users are redirected and cannot approve vets • Unauthenticated users are redirected to login

Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 16:Unit Test - Vet Approval



```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
[?] powershell - myproject + ▾

System check identified no issues (0 silenced).
test_approve_vet_success (django_admin.tests.ApproveVetTestCase.test_approve_vet_success)
Test successful vet approval by admin ... ok
test_approve_vet_unauthenticated_redirect (django_admin.tests.ApproveVetTestCase.test_approve_vet_unauthenticated_redirect)
Test unauthenticated users are redirected to login ... ok
test_approve_vet_unauthorized_redirect (django_admin.tests.ApproveVetTestCase.test_approve_vet_unauthorized_redirect)
Test non-admin users are redirected to login ... ok

-----
Ran 3 tests in 10.978s
OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 178: Unit Test - Vet approval result

4.2.5 Test: Admin2 Vet Decline Functionality

Test No.	Admin2
Objectives:	Test vet decline functionality by admin, including successful decline, access control for unauthorized users, unauthenticated access handling, and email notification on decline.
Test Cases:	<ol style="list-style-type: none"> 1. Successful Vet Decline 2. Unauthorized Decline Attempt by Regular User 3. Unauthenticated Decline Attempt 4. Vet Decline with Email Notification
Actions:	<p>1. Successful Vet Decline:</p> <ul style="list-style-type: none"> • Sample data: Admin User with is_staff: True and Vet User with status_verification=false • Admin logs in • Sends POST request to decline_vet view • Verifies redirect to vet approvals list

	<ul style="list-style-type: none">• Checks vet is marked as incomplete and status_change is set. <p>2. Unauthorized Decline Attempt:</p> <ul style="list-style-type: none">• Sample data: Regular User with is_staff=False and Vet User with status_verification=false• Regular user logs in• Sends POST request to decline_vet• Verifies redirection to admin login• Ensures no action has been taken <p>3. Unauthenticated Approval Attempt:</p> <ul style="list-style-type: none">• Sends POST request to decline_vet without logging in• Verifies redirection to admin login• Ensures vet is not declined <p>4. Vet Decline with Email Notification:</p>
--	---

	<ul style="list-style-type: none"> • Mocks send_email to simulate successful email sending • Admin logs in • Sends POST request to decline_vet • Asserts that an email containing the word "declined" was sent • Verifies vet's profile is marked incomplete • Checks for successful redirect to vet approvals
Expected Results	<ul style="list-style-type: none"> • Vet is declined by admin and marked appropriately • Unauthorized users are redirected and cannot decline vets • Unauthenticated users are redirected to login • Email is sent upon successful decline
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 17: Unit Test - Vet Decline

The screenshot shows a terminal window with the title "powershell - myproject". The window contains the output of a Django unit test. The test suite includes four tests: "test_decline_vet_email_notification", "test_decline_vet_success", "test_decline_vet_unauthenticated_redirect", and "test_decline_vet_unauthorized_redirect". All tests are marked as "ok". The test runner also reports that it ran 4 tests in 13.806 seconds and destroyed the test database for alias 'default'. The command used to run the test was "python manage.py test".

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject + [powershell - myproject]

System check identified no issues (0 silenced).
test_decline_vet_email_notification (django_admin.tests.DeclineVetTestCase.test_decline_vet_email_notification)
Test email notification is sent on successful decline ... ok
test_decline_vet_success (django_admin.tests.DeclineVetTestCase.test_decline_vet_success)
Test successful vet decline by admin ... ok
test_decline_vet_unauthenticated_redirect (django_admin.tests.DeclineVetTestCase.test_decline_vet_unauthenticated_redirect)
Test unauthenticated users are redirected to login ... ok
test_decline_vet_unauthorized_redirect (django_admin.tests.DeclineVetTestCase.test_decline_vet_unauthorized_redirect)
Test non-admin users are redirected to login ... ok

-----
Ran 4 tests in 13.806s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 179: Unit Test - Decline vet result

4.2.6 Test: Admin3 Content Moderation Functionality

Test No.	Admin3
Objectives:	Test admin's ability to delete user-generated content (posts, comments, replies), verify correct redirects, messaging, and enforce access control for unauthorized users.
Test Cases:	<ol style="list-style-type: none"> 1. Successful Post Deletion 2. Successful Comment Deletion 3. Successful Reply Deletion 4. Unauthorized Content Deletion Attempt
Actions:	<p>1. Successful Post Deletion:</p> <ul style="list-style-type: none"> • Admin logs in • Sends POST request to delete_post view with post ID • Verifies post is deleted from the database • Checks success message in response • Ensures redirection to post management page <p>2. Successful Comment Deletion:</p>

	<ul style="list-style-type: none">• Admin logs in• Sends POST request to delete_comment view• Verifies comment is deleted• Checks success message in response• Ensures redirection to post detail page <p>3. Successful Reply Deletion:</p> <ul style="list-style-type: none">• Admin logs in• Sends POST request to delete_reply view• Verifies reply is deleted• Checks success message in response• Ensures redirection to post detail page <p>4. Unauthorized Content Deletion Attempt:</p> <ul style="list-style-type: none">• Logs in with regular user• Sends POST request to delete post, comment, and reply
--	---

	<ul style="list-style-type: none"> • Verifies redirection to admin login for all attempts • Ensures post, comment, and reply remain in database
Expected Results	<ul style="list-style-type: none"> • Admin successfully deletes posts, comments, and replies with proper messaging • Non-admin users are redirected when attempting deletion • Content remains intact if deletion attempt is unauthorized
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 18: Unit Test - Content Moderation

```

System check identified no issues (0 silenced).
test_delete_comment_success (django_admin.tests.AdminDeleteContentTestCase.test_delete_comment_success)
Test successful comment deletion ... ok
test_delete_content_unauthorized (django_admin.tests.AdminDeleteContentTestCase.test_delete_content_unauthorized)
Test non-admin cannot delete content ... ok
test_delete_post_success (django_admin.tests.AdminDeleteContentTestCase.test_delete_post_success)
Test successful post deletion ... ok
test_delete_reply_success (django_admin.tests.AdminDeleteContentTestCase.test_delete_reply_success)
Test successful reply deletion ... ok

-----
Ran 4 tests in 14.082s
OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 180: Unit Test - Content Moderation result

4.2.7 Test: Admin4 Category Management Functionality

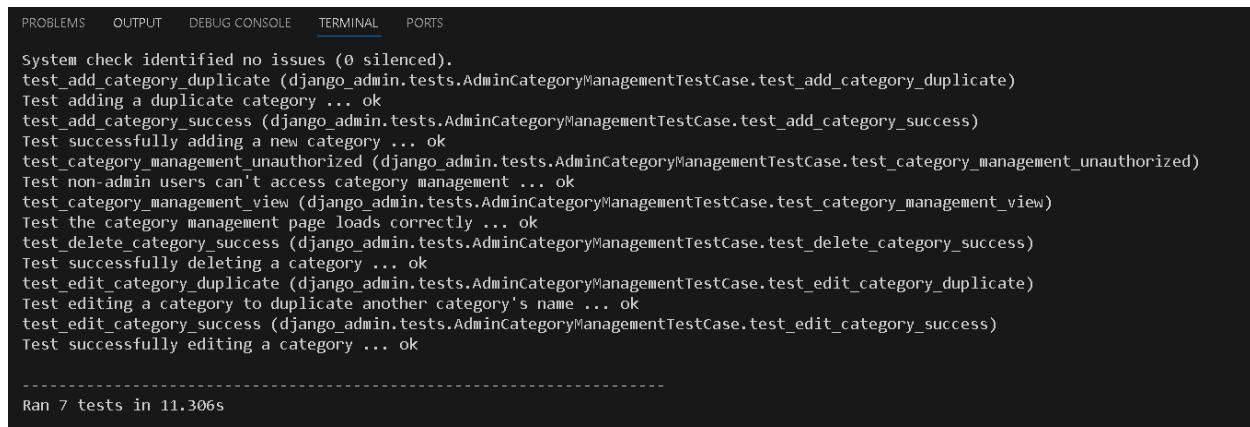
Test No.	Admin4
Objectives:	Ensure the admin can manage categories—view, add, edit, delete—while unauthorized users are blocked. Validate correct messaging and redirect behavior.
Test Cases:	<ol style="list-style-type: none"> 1. Load Category Management Page 2. Add New Category Successfully 3. Prevent Duplicate Category Creation 4. Edit Category Successfully 5. Prevent Duplicate Category Update 6. Delete Category and Update Posts 7. Restrict Category Management to Admin Only
Actions:	<p>1. Load Category Management Page:</p> <ul style="list-style-type: none"> • Sample Data: {Categories: Pets, Veterinary} • Admin logs in • Requests category management view • Asserts template used

	<ul style="list-style-type: none">• Confirms two categories appear with correct post count annotations <p>2. Add New Category Successfully:</p> <ul style="list-style-type: none">• Admin logs in• Submits POST with new category name/description• Verifies new category exists• Asserts success message• Checks redirect to category management page <p>3. Prevent Duplicate Category Creation:</p> <ul style="list-style-type: none">• Admin submits POST with a name that exists("Pets")• Ensures category count remains 1• Checks error message indicating duplication <p>4. Edit Category Successfully:</p> <ul style="list-style-type: none">• Admin edits existing category ("Pets" to "Updated Pets")• Verifies updated values
--	--

	<ul style="list-style-type: none">• Confirms success message• Checks proper redirect <p>5. Prevent Duplicate Category Update:</p> <ul style="list-style-type: none">• Admin tries to rename "Pets" to already existing "Veterinary"• Verifies name remains unchanged• Checks error message about duplicate category <p>6. Delete Category and Update Posts:</p> <ul style="list-style-type: none">• Admin deletes "Pets" category• Verifies deletion• Ensures associated post has category set to None• Confirms success message and redirect <p>7. Restrict Category Management to Admin Only:</p> <ul style="list-style-type: none">• Regular user logs in
--	--

	<ul style="list-style-type: none"> Attempts to access all category-related views Ensures each attempt redirects to admin login page
Expected Results	<ul style="list-style-type: none"> All category operations work correctly for admin users Duplicate name conflicts are handled gracefully Non-admins are restricted from accessing or modifying category data
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 19:Unit Test - Category Management



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

System check identified no issues (0 silenced).
test_add_category_duplicate (django_admin.tests.AdminCategoryManagementTestCase.test_add_category_duplicate)
Test adding a duplicate category ... ok
test_add_category_success (django_admin.tests.AdminCategoryManagementTestCase.test_add_category_success)
Test successfully adding a new category ... ok
test_category_management_unauthorized (django_admin.tests.AdminCategoryManagementTestCase.test_category_management_unauthorized)
Test non-admin users can't access category management ... ok
test_category_management_view (django_admin.tests.AdminCategoryManagementTestCase.test_category_management_view)
Test the category management page loads correctly ... ok
test_delete_category_success (django_admin.tests.AdminCategoryManagementTestCase.test_delete_category_success)
Test successfully deleting a category ... ok
test_edit_category_duplicate (django_admin.tests.AdminCategoryManagementTestCase.test_edit_category_duplicate)
Test editing a category to duplicate another category's name ... ok
test_edit_category_success (django_admin.tests.AdminCategoryManagementTestCase.test_edit_category_success)
Test successfully editing a category ... ok
-----
Ran 7 tests in 11.306s

```

Figure 181: Unit Test - Category Management result

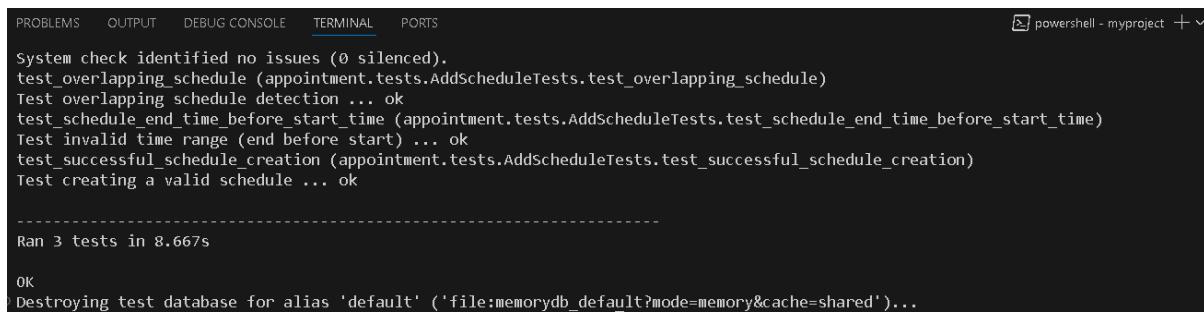
4.2.8 Test: Appt1 Add Schedule Functionality

Test No.	Appt1
Objectives:	Test adding a vet schedule, including valid schedule creation, invalid time range, and overlapping schedule detection.
Test Cases:	<ol style="list-style-type: none"> 1. Successful Schedule Creation: Verify a valid schedule can be created and saved correctly. 2. Schedule End Time Before Start Time: Verify error message when the end time is before the start time. 3. Overlapping Schedule: Verify error message when the new schedule overlaps with an existing one.
Actions:	<ol style="list-style-type: none"> 1. Successful Schedule Creation: <ul style="list-style-type: none"> • Sample data: {Day of the week: 'Monday', Start time: '10:00', End time: '12:00'} • Sent POST request with the above data to the schedule creation URL. • Verified that the response redirects to the vet schedule page.

	<ul style="list-style-type: none">• Checked that the new schedule was saved in the database. <p>2. Schedule End Timer Before Start Time:</p> <ul style="list-style-type: none">• Sample data: {Day of the week: 'Tuesday', Start time: '14:00', End time: '13:00'}• Sent POST request with the invalid time range data.• Verified that the response contains an error message: "End time must be after start time." <p>3. Overlapping Schedule:</p> <ul style="list-style-type: none">• Sample data: {Day of the week: 'Wednesday', Start time: '10:00', End time: '12:00'}• Created an initial schedule for Wednesday (9:00 to 11:00).
--	--

	<ul style="list-style-type: none"> • Sent POST request with the overlapping schedule data. • Verified that the response contains an error message: "The schedule overlaps with an existing one."
Expected Results	<ul style="list-style-type: none"> • Successful Schedule Creation: Redirects to the vet's schedule page, and the new schedule is saved in the database. • Schedule End Time Before Start Time: Error message displayed: "End time must be after start time." • Overlapping Schedule: Error message displayed: "The schedule overlaps with an existing one."
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 20:Unit Test - Add Schedule



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject + ✓

System check identified no issues (0 silenced).
test_overlapping_schedule (appointment.tests.AddScheduleTests.test_overlapping_schedule)
Test overlapping schedule detection ... ok
test_schedule_end_time_before_start_time (appointment.tests.AddScheduleTests.test_schedule_end_time_before_start_time)
Test invalid time range (end before start) ... ok
test_successful_schedule_creation (appointment.tests.AddScheduleTests.test_successful_schedule_creation)
Test creating a valid schedule ... ok

-----
Ran 3 tests in 8.667s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 182: Unit Test - Add schedule result

4.2.9 Test: Appt2 Edit Schedule Functionality

Test No.	Appt2
Objectives:	Test editing, updating, deleting vet schedules, and handling validation errors such as time range issues and schedule overlaps.
Test Cases:	<ol style="list-style-type: none"> 1. Edit Schedule GET Request: Verify that the edit schedule page renders correctly for a specific schedule. 2. Valid Schedule Update: Verify that a valid schedule update is processed successfully. 3. Invalid Time Range: Verify error message when attempting to update with an invalid time range (end time before start time). 4. Overlapping Schedule: Verify error message when attempting to update with a schedule that overlaps with an existing schedule. 5. Schedule Deletion: Verify that a schedule can be deleted successfully.
Actions:	<ol style="list-style-type: none"> 1. Edit Schedule GET Request: <ul style="list-style-type: none"> • Sample data:

	<p>{Schedule ID: ID of an existing schedule}</p> <ul style="list-style-type: none">• Sent GET request to the edit schedule page for a specific schedule.• Verified that the response status code is 200.• Checked that the correct template is used and the schedule object is passed in the context. <p>2. Valid Schedule Update:</p> <ul style="list-style-type: none">• Sample data:<p>{Day of the week: 'Tuesday', Start time: '10:00', End time: '15:00'}</p>• Sent POST request with valid data to update the schedule.• Verified that the response redirects to the vet schedule page.• Refreshed the schedule from the database and
--	--

	<p>checked if the updates were saved correctly.</p> <p>3. Invalid Time Range:</p> <ul style="list-style-type: none">• Sample data: {Day of the week: 'Wednesday', Start time: '14:00', End time: '12:00'}• Sent POST request with invalid time range (end time before start time).• Verified that the response contains an error message: "End time must be after start time."• Checked that the schedule was not updated and remained the same in the database. <p>4. Overlapping Schedule:</p> <ul style="list-style-type: none">• Sample data: {Day of the week: 'Thursday', Start time: '11:00', End time: '13:00'}
--	--

	<ul style="list-style-type: none">• Created a conflicting schedule for Thursday (10:00 to 12:00).• Sent POST request to update the schedule with overlapping times.• Verified that the response contains an error message: "The schedule overlaps with an existing one."• Checked that the schedule was not updated and remained the same in the database. <p>5. Schedule:</p> <ul style="list-style-type: none">• Sample data: {Schedule ID: ID of an existing schedule}• Sent POST request with the delete_schedule flag set to true.• Verified that the response redirects to the vet schedule page.
--	--

	<ul style="list-style-type: none"> Checked that the schedule was deleted from the database.
Expected Results	<ul style="list-style-type: none"> Edit Schedule GET Request: Page renders correctly with the schedule data in the context. Valid Schedule Update: The schedule is updated, and the response redirects to the vet schedule page. Invalid Time Range: Error message displayed: "End time must be after start time," and the schedule is not updated. Overlapping Schedule: Error message displayed: "The schedule overlaps with an existing one," and the schedule is not updated. Schedule Deletion: The schedule is deleted, and the response redirects to the vet schedule page.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 21:Unit Test - Edit Schedule

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

System check identified no issues (0 silenced).
test_edit_schedule_get (appointment.tests.EditScheduleTests.test_edit_schedule_get)
Test GET request to edit schedule page ... ok
test_invalid_time_range (appointment.tests.EditScheduleTests.test_invalid_time_range)
Test updating with end time before start time ... ok
test_overlapping_schedule (appointment.tests.EditScheduleTests.test_overlapping_schedule)
Test updating with overlapping time ... ok
test_schedule_deletion (appointment.tests.EditScheduleTests.test_schedule_deletion)
Test deleting a schedule ... ok
test_valid_schedule_update (appointment.tests.EditScheduleTests.test_valid_schedule_update)
Test updating schedule with valid data ... ok

-----
Ran 5 tests in 5.295s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 183: Unit Test - Edit schedule result

4.2.10 Test: Appt3 Booking via Khalti Functionality

Test No.	Appt3
Objectives:	<p>Test the appointment booking functionality, including displaying the booking page, successful booking, and error handling for missing pet selection.</p>
Test Cases:	<ol style="list-style-type: none"> 1. Book Appointment GET Request: Verify the correct rendering of the appointment booking page. 2. Successful Booking: Verify successful booking of an appointment and correct redirection to payment page. 3. Missing Pet Selection: Verify error handling when attempting to book an appointment without selecting a pet.
Actions:	<ol style="list-style-type: none"> 1. Book Appointment GET Request: <ul style="list-style-type: none"> • Sample data: {Vet profile: self.vet_profile, Schedule: self.schedule, Pet: self.pet} • Sent GET request to the appointment booking page for the selected vet and schedule.

	<ul style="list-style-type: none">• Verified that the response status code is 200.• Checked that the correct template is used and the context contains the correct vet, schedule, and pet data. <p>2. Successful Booking:</p> <ul style="list-style-type: none">• Sample data: {Pet ID: self.pet.id, Reason: 'Annual check-up'}• Sent POST request with valid data (pet and reason) to book the appointment.• Verified that the response status code is 302 (redirect).• Verified that the response redirects to the payment initiation URL.• Checked that an appointment was created in the database with the correct data (vet, pet owner, pet, and status 'unpaid'). <p>3. Missing Pet Selection:</p> <ul style="list-style-type: none">• Sample data:
--	--

	<p>{Reason: 'Checkup'}</p> <ul style="list-style-type: none"> • Sent POST request with the missing pet field. • Verified that the response status code is 200. • Checked that the error message "Please select a pet" is displayed. • Verified that no appointment was created in the database.
Expected Results	<ul style="list-style-type: none"> • Book Appointment GET Request: Page renders correctly with vet, schedule, and pet information in the context. • Successful Booking: Appointment is created, and the response redirects to the payment page. • Missing Pet Selection: Error message "Please select a pet" is displayed, and no appointment is created.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 22:Unit Test - Booking via Khalti

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

System check identified no issues (0 silenced).
test_book_appointment_get (appointment.tests.BookAppointmentTests.test_book_appointment_get)
Test GET request to book appointment page ... ok
test_missing_pet_selection (appointment.tests.BookAppointmentTests.test_missing_pet_selection)
Test booking without selecting a pet ... ok
test_successful_booking (appointment.tests.BookAppointmentTests.test_successful_booking)
Test successful appointment booking ... ok

-----
Ran 3 tests in 13.181s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 184: Unit Test - Booking via khalti result

4.2.11 Test: Appt4 Booking via Store Credit Functionality

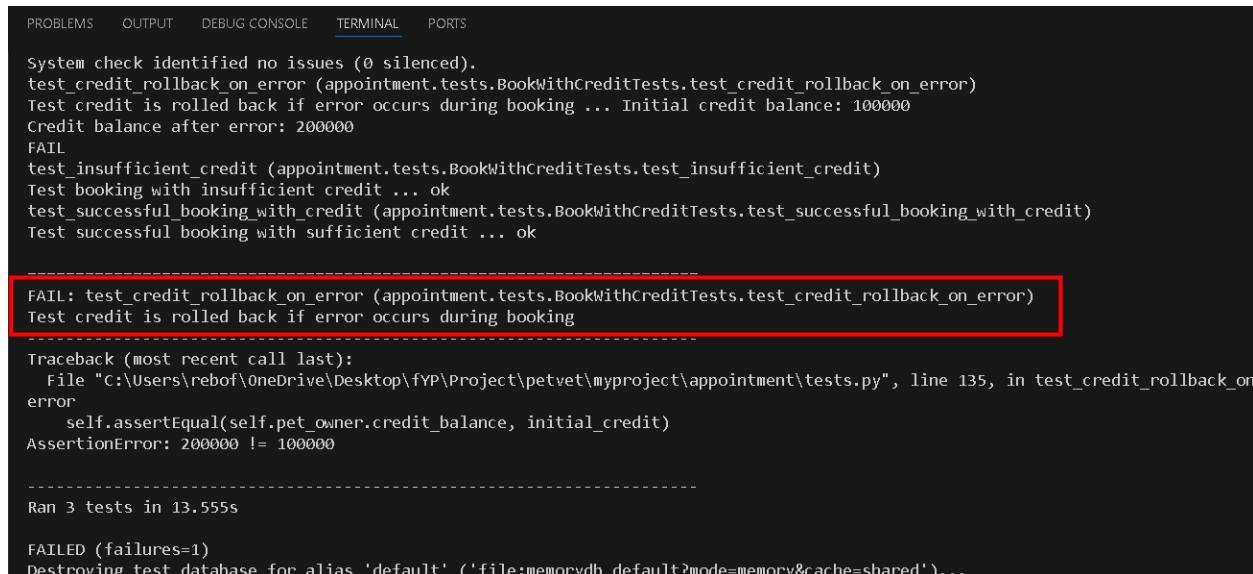
Test No.	Appt4
Objectives:	Test the appointment booking using store credit balance, including successful booking with sufficient credit, handling of insufficient credit, and ensuring credit is not deducted when an error occurs.
Test Cases:	<ol style="list-style-type: none"> 1. Successful Booking with Credit: Verify that an appointment is successfully booked using credit when the pet owner has enough balance. 2. Insufficient Credit: Verify that the booking fails and displays an error message when the pet owner's credit balance is insufficient. 3. Credit Rollback on Error: Ensure that if there's an error during booking (e.g., invalid pet ID), the credit balance is not deducted.
Actions:	<ol style="list-style-type: none"> 1. Successful Booking with Credit: <ul style="list-style-type: none"> • Sample Data: {Pet ID: self.pet.id, Reason: 'Annual checkup'} • Logged in as a pet owner with 1000 NPR credit. • Sent POST request to book_with_credit view.

	<ul style="list-style-type: none"> • Verified response status code is 200 and template used is appointment/appointment_detail.html. • Checked that an appointment was created with status paid_pending_approval, payment status paid, and amount paid = 1000 * 100. • Verified credit was deducted correctly from the pet owner. <p>2. Insufficient Credit:</p> <ul style="list-style-type: none"> • Sample Data: <pre>{Pet ID: self.pet.id, Reason: 'Checkup'}</pre> <ul style="list-style-type: none"> • Set pet owner's credit balance to 500 NPR. • Sent POST request to book_with_credit view. • Verified response status code is 200 and template used is appointment/book_appt.html. • Checked for error message "Insufficient credit balance". • Ensured no appointment was created in the database.
--	---

	<p>3. Credit Rollback on Error:</p> <ul style="list-style-type: none"> • Sample Data: <pre>{Pet ID: 'invalid', Reason: 'Checkup'}</pre> <ul style="list-style-type: none"> • Sent POST request with invalid pet ID.
Expected Results	<ul style="list-style-type: none"> • Successful Booking with Credit: Appointment created, credit deducted, and user redirected to appointment detail page. • Insufficient Credit: Error message displayed, no appointment created. • Credit Rollback on Error: Error handled gracefully, and credit balance remains unaffected.
Actual Results	1 st and 2 nd test case passed (ok in test output) but the 3 rd one showed an error. Initially, the credit was 10,000 paisa, but after the error occurred, the credit unexpectedly doubled.
Solution	<ul style="list-style-type: none"> • The test <code>test_credit_rollback_on_error</code> checks if credit remains unchanged when an error occurs during booking. • Django's transaction management auto-rolls back changes on exceptions, so this line is unnecessary in the view's function: <pre>self.pet_owner.credit_balance += 1000 * 100</pre>

	<ul style="list-style-type: none"> • Remove the manual rollback. Just handle the exception, and assert that the credit balance hasn't changed.
Conclusion	The test was successful

Table 23:Unit Test - Booking via Store Credit



```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

system check identified no issues (0 silenced).
test_credit_rollback_on_error (appointment.tests.BookWithCreditTests.test_credit_rollback_on_error)
Test credit is rolled back if error occurs during booking ... Initial credit balance: 100000
Credit balance after error: 200000
FAIL
test_insufficient_credit (appointment.tests.BookWithCreditTests.test_insufficient_credit)
Test booking with insufficient credit ... ok
test_successful_booking_with_credit (appointment.tests.BookWithCreditTests.test_successful_booking_with_credit)
Test successful booking with sufficient credit ... ok

-----
FAIL: test_credit_rollback_on_error (appointment.tests.BookWithCreditTests.test_credit_rollback_on_error)
Test credit is rolled back if error occurs during booking

Traceback (most recent call last):
  File "C:\Users\rebof\OneDrive\Desktop\fYP\Project\petvet\myproject\appointment\tests.py", line 135, in test_credit_rollback_on_error
    self.assertEqual(self.pet_owner.credit_balance, initial_credit)
AssertionError: 200000 != 100000

-----
Ran 3 tests in 13.555s

FAILED (failures=1)
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 185: Unit Test - Booking via Store credit failed result

In the views function, it was doubling the credit unknowingly. It was removed.



```

except Exception as e:
    # Revert credit deduction if error occurs
    pet_owner.credit_balance += 1000 * 100
    pet_owner.save()
    messages.error(request, f"Error booking appointment: {str(e)}")
    return redirect('appointment:book_appointment', vet_id=vet_id, schedule_id=schedule_id)

```

Figure 186: Unit Test - Booking via Store credit Solution

```
system check identified no issues (0 silenced).
test_credit_rollback_on_error (appointment.tests.BookWithCreditTests.test_credit_rollback_on_error)
Test credit is rolled back if error occurs during booking ... Initial credit balance: 100000
Credit balance after error: 100000
ok
test_insufficient_credit (appointment.tests.BookWithCreditTests.test_insufficient_credit)
Test booking with insufficient credit ... ok
test_successful_booking_with_credit (appointment.tests.BookWithCreditTests.test_successful_booking_with_credit)
Test successful booking with sufficient credit ... ok

-----
Ran 3 tests in 13.513s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 187: Unit Test - Booking via Store credit result

4.2.12 Test: Appt5 Accept Booking Functionality

Test No.	Appt5
Objectives:	<p>Test the vet's ability to accept appointments, ensuring that the selected appointment is confirmed, other conflicting appointments are rejected, schedule availability is updated, and that only the correct vet can perform the action.</p>
Test Cases:	<ol style="list-style-type: none"> 1. Accept Appointment Success: Verify that a vet can successfully accept an appointment, which results in confirmation of the selected appointment, rejection of other conflicting ones, and schedule update. 2. Accept Appointment by Wrong Vet: Ensure that a vet cannot accept appointments that are not assigned to them. 3. Accept Appointment Rejects Others: Confirm that accepting one appointment automatically rejects all other appointments linked to the same schedule.
Actions:	<ol style="list-style-type: none"> 1. Accept Appointment Success: <ul style="list-style-type: none"> • Sample Data: { Appointment ID: self.appointment.id}

	<ul style="list-style-type: none">• Created a GET request as the correct vet user.• Called the accept_appointment view with a valid appointment.• Verified:<ol style="list-style-type: none">i. Appointment status is updated to confirmed.ii. Other appointment linked to the same schedule is updated to rejected.iii. Schedule is marked as unavailable.iv. Notification function is triggered twice (confirmation and rejection).v. Redirects to vet_pending_appointments. <p>2. Accept Appointment by Wrong Vet:</p> <ul style="list-style-type: none">• Sample Data: {Appointment ID: self.appointment.id}• Created a GET request as a different vet user.
--	---

	<ul style="list-style-type: none"> • Called the accept_appointment view. • Verified response status code is 200 and template used is appointment/book_appt.html. • Checked for error message "Insufficient credit balance". • Ensured no appointment was created in the database. • Verified: <ul style="list-style-type: none"> i. User is redirected to the correct vet's pending appointments page. ii. Appointment status remains paid_pending_approval. <p>3. Accept Appointment Rejects Others:</p> <ul style="list-style-type: none"> • Sample Data: {Appointment ID: self.appointment.id} • Created another pending appointment with the same schedule. • Called the accept_appointment view as the correct vet. • Verified:
--	--

	<ul style="list-style-type: none"> i. The new appointment is rejected. ii. Notification for rejection is sent with the appropriate template.
Expected Results	<ul style="list-style-type: none"> • Accept Appointment Success: Appointment is confirmed, others rejected, schedule updated, and notifications sent. • Accept Appointment by Wrong Vet: Action fails with redirection; no changes to appointment status. • Accept Appointment Rejects Others: All conflicting appointments are rejected, and a rejection notification is sent.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 24:Unit Test - Accept Booking

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject + [ ] [x]

System check identified no issues (0 silenced).
test_accept_appointment_rejects_others (appointment.tests.AcceptAppointmentTestCase.test_accept_appointment_rejects_others) ... ok
test_accept_appointment_success (appointment.tests.AcceptAppointmentTestCase.test_accept_appointment_success) ... ok
test_accept_appointment_wrong_vet (appointment.tests.AcceptAppointmentTestCase.test_accept_appointment_wrong_vet) ... ok

-----
Ran 3 tests in 12.236s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 188: Unit Test - Accept booking result

4.2.13 Test: Appt6 Reject Booking Functionality

Test No.	Appt6
Objectives:	Ensure appointment rejection handles credit refunds, access control, and exception safety during refund processing.
Test Cases:	<ol style="list-style-type: none"> 1. Successful Rejection with Refund: Verify that a vet can reject a paid appointment, triggering a refund to the pet owner, updating appointment status, and sending a notification. 2. Rejection by Wrong Vet: Ensure that a vet who is not assigned to the appointment cannot reject it and is redirected without changes. 3. Rejection Error Handling: Simulate a failure in the credit refund process and ensure that the rejection is rolled back and the vet is redirected appropriately.
Actions:	<ol style="list-style-type: none"> 1. Successful Rejection with Refund: <ul style="list-style-type: none"> • Sample Data: {Appointment ID: self.paid_appointment.id} • Logged in as the correct vet. • Sent a GET request to reject_appointment view

	<ul style="list-style-type: none"> • Verified: <ul style="list-style-type: none"> i. Appointment status is updated to reject. ii. Credit is refunded to the owner's account iii. Notification function is called once with rejection template iv. Redirects to vet's pending appointment page 2. Rejection by Wrong Vet: <ul style="list-style-type: none"> • Sample Data: {Appointment ID: self.paid_appointment.id} • Logged in as a different vet • Called the accept_appointment view. • Sent GET request to reject_appointment • Verified: <ul style="list-style-type: none"> i. Appointment status remains paid_pending_approval ii. Redirects to the correct vet's pending appointment page
--	---

	<p>3. Rejection Error Handling:</p> <ul style="list-style-type: none"> • Sample Data: { Appointment ID: self.paid_appointment.id } • Mocked a failure in the credit update (override save method) • Logged in as the correct vet • Sent GET request to reject_appointment • Verified: <ul style="list-style-type: none"> i. Credit balance remains unchanged ii. Redirects to vet's pending appointment page iii. Appointment remains in the correct state (rolled back if needed)
Expected Results	<ul style="list-style-type: none"> • Successful Rejection with Refund: • Successful Rejection with Refund: Appointment is rejected, refund processed, notification sent, and user redirected. • Rejection by Wrong Vet: Rejection attempt is blocked, appointment unchanged, and user redirected.

	<ul style="list-style-type: none"> • Rejection Error Handling: No partial update occurs; credit remains unchanged, error is silently handled, and redirection is successful.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 25:Unit Test - Reject Booking

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject + v

System check identified no issues (0 silenced).
test_rejection_by_wrong_vet (appointment.tests.RejectAppointmentTestCase.test_rejection_by_wrong_vet)
Test that other vets can't reject the appointment ... ok
test_rejection_error_handling (appointment.tests.RejectAppointmentTestCase.test_rejection_error_handling)
Test error during rejection is properly handled ... ok
test_successful_rejection_with_refund (appointment.tests.RejectAppointmentTestCase.test_successful_rejection_with_refund)
Test rejecting a paid appointment credits the owner's account ... ok

-----
Ran 3 tests in 13.914s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 189: Unit Test - Reject/decline booking result

4.2.14 Test: Appt7 Mark as Complete Functionality

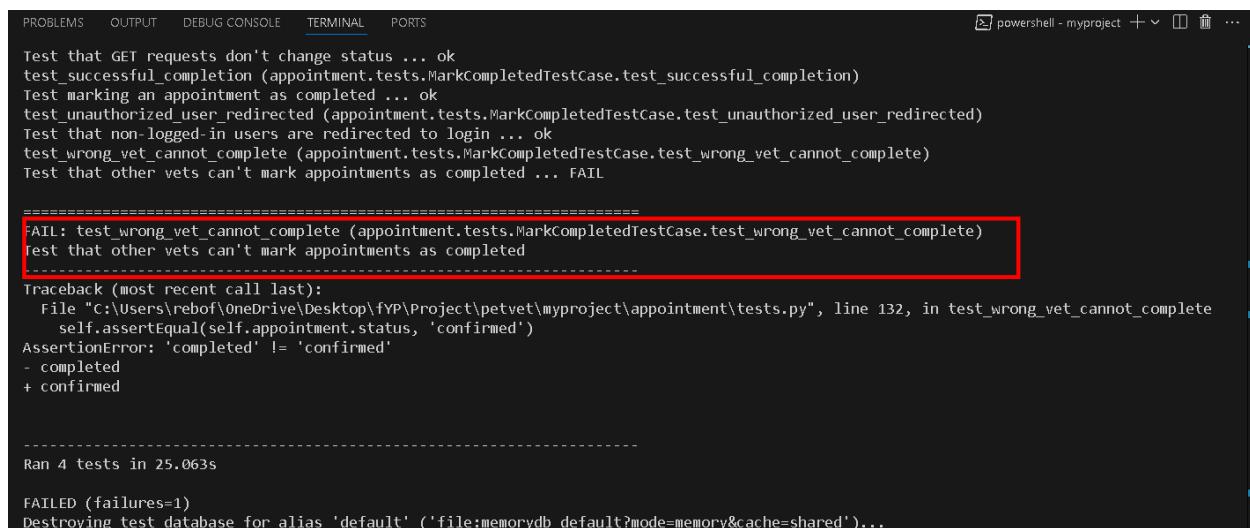
Test No.	Appt7
Objectives:	<p>Test the vet's ability to mark an appointment as completed, ensuring:</p> <ul style="list-style-type: none"> • The appointment status is correctly updated to completed • The associated schedule's availability is restored • Only the assigned vet can perform the action • Invalid requests (GET or unauthorized users) are handled properly
Test Cases:	<ol style="list-style-type: none"> 1. Successful Booking with Credit: Verify that an appointment is successfully booked using credit when the pet owner has enough balance. 2. GET Request Does Nothing: Ensure that a GET request to the completion endpoint does not alter appointment status or schedule. 3. Wrong Vet Cannot Complete: Confirm that only the correct vet can complete an appointment and others are redirected without changes. 4. Unauthorized User Redirected: Test that unauthenticated users attempting to

	complete appointments are redirected to the login page.
Actions:	<p>1. Successful Completion:</p> <ul style="list-style-type: none"> • Sample Data: {Appointment ID: self.appointment.id} • Logged in as the correct vet • Sent a POST request to mark_completed view • Verified: <ol style="list-style-type: none"> i. Appointment status updated to completed ii. Schedule availability set to True iii. Redirects to vet_accepted_appointments <p>2. GET Request Does Nothing:</p> <ul style="list-style-type: none"> • Logged in as the correct vet • Set pet owner's credit balance to 500 NPR. • Sent a GET request to the same view • Verified:

	<p>i. Appointment status remains confirmed</p> <p>ii. Response is a redirect (no state change)</p> <p>3. Wrong Vet Cannot Complete:</p> <ul style="list-style-type: none"> • Logged in as a different vet • Sent a POST request to mark_completed view <p>4. Unauthorized User Redirected:</p> <ul style="list-style-type: none"> • Sent POST request without logging in • Redirect response with URL leading to login page
Expected Results	<ul style="list-style-type: none"> • Successful Completion: Appointment marked as completed, schedule availability restored, and vet redirected. • GET Request Does Nothing: Appointment status remains unchanged; response is a redirect. • Wrong Vet Cannot Complete: No changes made; redirect occurs to original vet's accepted appointments page. • Unauthorized User Redirected: Redirect to login page with no changes made to the appointment.

Actual Results	1 st , 2 nd and 4 th test case passed (ok in test output) but the 3 rd one showed an error.
Solution	<ul style="list-style-type: none"> Previously, there were no permission checks to prevent other vets from accessing the rejection via a POST request. I've now added the necessary logic to handle this, which resolves the failed test case.
Conclusion	The test was successful

Table 26: Unit Test - Mark as Complete



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject + ... □

Test that GET requests don't change status ... ok
test_successful_completion (appointment.tests.MarkCompletedTestCase.test_successful_completion)
Test marking an appointment as completed ... ok
test_unauthorized_user_redirected (appointment.tests.MarkCompletedTestCase.test_unauthorized_user_redirected)
Test that non-logged-in users are redirected to login ... ok
test_wrong_vet_CANNOT_COMPLETE (appointment.tests.MarkCompletedTestCase.test_wrong_vet_CANNOT_COMPLETE)
Test that other vets can't mark appointments as completed ... FAIL

=====
FAIL: test_wrong_vet_CANNOT_COMPLETE (appointment.tests.MarkCompletedTestCase.test_wrong_vet_CANNOT_COMPLETE)
Test that other vets can't mark appointments as completed

Traceback (most recent call last):
  File "C:\Users\rebof\OneDrive\Desktop\fYP\Project\petvet\myproject\appointment\tests.py", line 132, in test_wrong_vet_CANNOT_COMPLETE
    self.assertEqual(self.appointment.status, 'confirmed')
AssertionError: 'completed' != 'confirmed'
- completed
+ confirmed

Ran 4 tests in 25.063s
FAILED (failures=1)
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 190: Unit Test - Mark as Complete failed result

The check permission was added, previously there was nothing to handle other user accessing the post request.

```
@login_required
def mark_completed(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Check permissions
    if not hasattr(appointment, 'vet') or not hasattr(appointment.vet, 'user') or \
        request.user != appointment.vet.user:
        messages.error(request, "You don't have permission to mark this appointment as completed.")
        return HttpResponseRedirect(reverse('appointment:veterinarian_appointments', args=[appointment.vet.id]))

    if request.method == "POST":
        appointment.status = "completed"
        appointment.save()
        appointment.schedule.available = True
        appointment.schedule.save()
```

Figure 191: Unit Test - Mark as Complete Solution

```
System check identified no issues (0 silenced).
test_get_request_does_nothing (appointment.tests.MarkCompletedTestCase.test_get_request_does_nothing)
Test that GET requests don't change status ... ok
test_successful_completion (appointment.tests.MarkCompletedTestCase.test_successful_completion)
Test marking an appointment as completed ... ok
test_unauthorized_user_redirected (appointment.tests.MarkCompletedTestCase.test_unauthorized_user_redirected)
Test that non-logged-in users are redirected to login ... ok
test_wrong_vet_CANNOT_COMPLETE (appointment.tests.MarkCompletedTestCase.test_wrong_vet_CANNOT_COMPLETE)
Test that other vets can't mark appointments as completed ... ok

-----
Ran 4 tests in 18.636s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 192: Unit Test - Mark as Complete result

4.2.15 Test: Appt8 Booking Cancellation Functionality

Test No.	Appt8
Objectives:	<p>Verify that pet owners can cancel their appointments and receive appropriate refunds, with conditions:</p> <ul style="list-style-type: none"> • 80% refund for paid appointments • Only POST requests perform cancellation • Appropriate success messages are displayed • Status updates and balance changes are correct
Test Cases:	<ol style="list-style-type: none"> 1. Cancel Paid & Confirmed Appointment: Tests cancellation with 80% refund applied to a fully paid, confirmed appointment. 2. Cancel Paid & Pending Appointment: Tests the same 80% refund behavior for a paid appointment awaiting vet approval. 3. GET Request Does Nothing: Ensures a GET request doesn't cancel or change anything.
Actions:	<ol style="list-style-type: none"> 1. Cancel Paid & Confirmed Appointment: <ul style="list-style-type: none"> • Sample Data: {Appointment ID: self.paid_confirmed_appointment.id,

	<p>Status: confirmed, Amount Paid: 1000 NPR}</p> <ul style="list-style-type: none">• Logged in as pet owner• Sent POST request to cancel endpoint• Verified:<ol style="list-style-type: none">i. Status changed to cancelledii. Refund of 800 NPR creditediii. Flash message shows refundiv. Redirects to appointment_list <p>2. Cancel Paid & Pending Appointment:</p> <ul style="list-style-type: none">• Sample Data:<p>{Appointment ID: self.paid_pending_appointment.id, Status: paid_pending_approval, Amount Paid: 1000 NPR}</p>• Logged in as pet owner• Sent POST request to cancel endpoint• Sent GET request to reject_appointment• Verified:<ol style="list-style-type: none">i. Status changed to cancelled
--	---

	<p>ii. Refund of 640 NPR credited</p> <p>iii. Flash message shows refund</p> <p>3. GET Request Does Nothing:</p> <ul style="list-style-type: none"> • Sent GET request to cancellation endpoint • Verified: <ul style="list-style-type: none"> i. Status remains confirmed ii. No refund processed iii. Redirects without performing action
Expected Results	<ul style="list-style-type: none"> • Cancellation (POST): Appointment status becomes cancelled, 80% of amount paid is refunded, success message shown. • GET Requests: No changes made; user is redirected.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 27:Unit Test - Booking Cancellation

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

System check identified no issues (0 silenced).
test_cancel_paid_confirmed_appointment (appointment.tests.CancelAppointmentTestCase.test_cancel_paid_confirmed_appointment)
Test cancelling a paid, confirmed appointment with 80% refund ... ok
test_cancel_paid_pending_appointment (appointment.tests.CancelAppointmentTestCase.test_cancel_paid_pending_appointment)
Test cancelling a paid but pending approval appointment ... ok
test_get_request_does_nothing (appointment.tests.CancelAppointmentTestCase.test_get_request_does_nothing)
Test that GET requests don't cancel appointments ... ok

-----
Ran 3 tests in 10.535s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

Figure 193: Unit Test - Booking Cancellation result

4.2.16 Test: Appt9 Khalti Payment Functionality

Test No.	Appt9
Objectives:	<p>Verify end-to-end behavior of Khalti payment integration:</p> <ul style="list-style-type: none"> • Payment initiation and redirection • Verification for successful, failed, and invalid transactions • Correct appointment status/payment status updates • Error handling with missing/invalid parameters
Test Cases:	<ol style="list-style-type: none"> 1. Initiate Khalti Payment (Success) 2. Initiate Khalti Payment (Failure) 3. Verify Khalti Payment (Success) 4. Verify Khalti Payment (Failure) 5. Verify Khalti Payment (Request Error) 6. Verify Khalti Payment (Missing PIDX)
Actions:	<ol style="list-style-type: none"> 1. GET request to initiate endpoint with valid setup

	<ol style="list-style-type: none"> 2. Simulated failure from Khalti API 3. GET with valid pidx, mocked success 4. GET with valid pidx, mocked failure 5. Simulated exception during verification 6. GET without pidx
Expected Results	<ol style="list-style-type: none"> 1. Redirect (302) to Khalti payment URL 2. Renders payment_error.html 3. Status: paid_pending_approval, redirect (302) 4. Status unchanged(unpaid), redirect to failure page 5. Renders payment_error.html 6. Renders payment_error.html
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 28:Unit Test - Khalti Payment

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell - myproject + 

System check identified no issues (0 silenced).
test_initiate_khalti_payment_failure (appointment.tests.KhaltiPaymentTestCase.test_initiate_khalti_payment_failure)
Test failed payment initiation ... ok
test_initiate_khalti_payment_success (appointment.tests.KhaltiPaymentTestCase.test_initiate_khalti_payment_success)
Test successful payment initiation ... ok
test_verify_khalti_missing_pidx (appointment.tests.KhaltiPaymentTestCase.test_verify_khalti_missing_pidx)
Test verification with missing pidx parameter ... ok
test_verify_khalti_payment_error (appointment.tests.KhaltiPaymentTestCase.test_verify_khalti_payment_error)
Test error during verification ... ok
test_verify_khalti_payment_failed (appointment.tests.KhaltiPaymentTestCase.test_verify_khalti_payment_failed)
Test failed payment verification ... ok
test_verify_khalti_payment_success (appointment.tests.KhaltiPaymentTestCase.test_verify_khalti_payment_success)
Test successful payment verification ... ok

-----
Ran 6 tests in 20.075s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 194: Unit Test - Khalti payment result

4.2.17 Test: Appt10 Review Functionality

Test No.	Appt10
Objectives:	Validate the review system for vets—ensuring only eligible pet owners can submit reviews for completed and paid appointments, only once per appointment, and that correct templates/messages are used.
Test Cases:	<ol style="list-style-type: none"> 1. Unauthorized User Cannot Review 2. Incomplete Appointment Cannot Be Reviewed 3. Prevent Duplicate Reviews 4. Successful Review Submission 5. Correct Template Rendered on Review Page
Actions:	<p>1. Unauthorized User Cannot Review:</p> <ul style="list-style-type: none"> • Create an unrelated pet owner • Attempt to access review page for an appointment not belonging to them • Expect HTTP 403 Forbidden response

	<p>2. Incomplete Appointment Cannot Be Reviewed:</p> <ul style="list-style-type: none">• Change appointment status to 'paid_pending_approval'• Attempt to access review page• Expect redirect to appointment detail page• Expect error message about needing a completed appointment <p>3. Prevent Duplicate Reviews:</p> <ul style="list-style-type: none">• Create a review for the appointment• Attempt to access review page again• Expect redirect to appointment detail page• Expect message about already reviewed <p>4. Successful Review Submission:</p> <ul style="list-style-type: none">• Pet owner submits POST with rating/comment for completed and paid appointment
--	---

	<ul style="list-style-type: none"> • Check review is saved • Redirect to appointment detail <p>5. Correct Template Rendered on Review Page:</p> <ul style="list-style-type: none"> • Pet owner accesses review page via GET • Check correct template (review_vet.html) is rendered • Verify appointment context is passed to template
Expected Results	<ul style="list-style-type: none"> • Only rightful pet owner of a completed and paid appointment can review • One review per appointment • Proper messages and templates used throughout
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 29:Unit Test - Review

The screenshot shows a terminal window with the following output:

```
System check identified no issues (0 silenced).
test_review_vet_already_reviewed (authUser.tests.ReviewVetTestCase.test_review_vet_already_reviewed)
Test that a user can't review the same appointment twice ... ok
test_review_vet_not_completed_appointment (authUser.tests.ReviewVetTestCase.test_review_vet_not_completed_appointment)
Test that only completed appointments can be reviewed ... ok
test_review_vet_successful_submission (authUser.tests.ReviewVetTestCase.test_review_vet_successful_submission)
Test successful review submission ... ok
test_review_vet_template_used (authUser.tests.ReviewVetTestCase.test_review_vet_template_used)
Test that the correct template is used for GET request ... ok
test_review_vet_unauthorized_user (authUser.tests.ReviewVetTestCase.test_review_vet_unauthorized_user)
Test that only the pet owner can review the appointment ... ok

-----
Ran 5 tests in 15.653s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
```

Figure 195: Unit Test - Review result

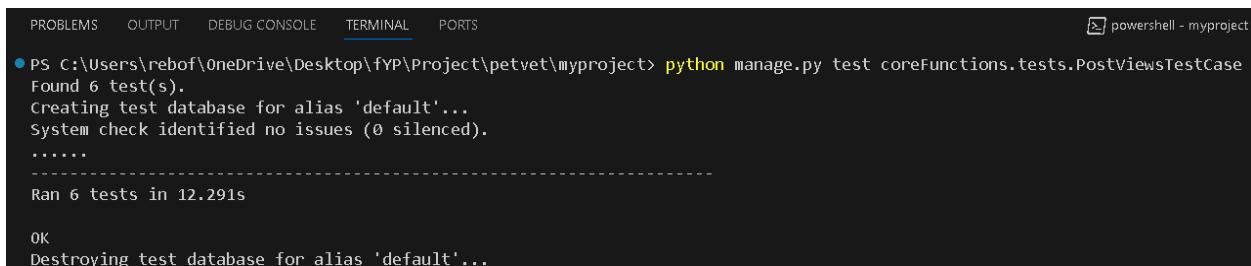
4.2.18 Test: Po1 Post Creation and Deletion Functionality

Test No.	Po1
Objectives:	Verify the behavior of post-related views, ensuring that post creation, image uploading, authentication, and deletion are handled correctly, with proper error handling for invalid requests and unauthorized actions.
Test Case	<ol style="list-style-type: none"> 1. Create Post Successfully 2. Create Post with Image Upload 3. Prevent Unauthenticated Users from Creating Post 4. Delete Post Successfully 5. Handle Invalid Method for Post Deletion 6. Prevent Unauthorized Post Deletion
Actions:	<ol style="list-style-type: none"> 1. Create Post Successfully: <ul style="list-style-type: none"> • Submit a valid post creation request • Expect redirection to post-detail view 2. Create Post with Image Upload:

	<ul style="list-style-type: none">• Submit a valid post creation request with an image• Redirection to post-detail view• Verify that the image was successfully uploaded <p>3. Prevent Unauthenticated User from Creating Post:</p> <ul style="list-style-type: none">• Log out the current user and attempt to create a post• Redirection to login page• Verify no new post is created <p>4. Delete Post Successfully:</p> <ul style="list-style-type: none">• Create a post and submit a delete request• JSON response with success status <p>5. Handle Invalid Method for Post Deletion:</p> <ul style="list-style-type: none">• Attempt to delete a post using a GET request
--	---

	<ul style="list-style-type: none"> • 400 error with an appropriate message <p>6. Prevent Unauthorized Post Deletion:</p> <ul style="list-style-type: none"> • Create a post and attempt deletion with a different user • Expect the post to remain unchanged • Verify that the post deletion count remains the same
Expected Results	<ul style="list-style-type: none"> • Posts should be created and deleted correctly with proper authentication checks • Images should be uploaded with posts and the correct URL should be returned • Unauthorized users should not be able to create or delete posts • Invalid requests (e.g., using GET for deletion) should return appropriate error messages
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 30:Unit Test - Post Creation and Deletion



The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - myproject

● PS C:\Users\rebof\OneDrive\Desktop\fYP\Project\petvet\myproject> python manage.py test coreFunctions.tests.PostViewsTestCase
Found 6 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

.....
-----
Ran 6 tests in 12.291s

OK
destroying test database for alias 'default'...
```

Figure 196: Unit Test - Post creation and Deletion result

4.2.19 Test: Po2 Post Updating and Liking Functionality

Test No.	Po2
Objectives:	Verify the functionality of post editing (with and without image upload) and the "like/unlike" feature. Ensure that only authorized users can edit posts and that liking posts via an AJAX request works correctly.
Test Case	<ol style="list-style-type: none"> 1. Edit Post Successfully 2. Edit Post with Image Upload 3. Unauthorized User Cannot Edit Post 4. Like Post Successfully (Toggle) 5. Like Post with Invalid Method
Actions:	<ol style="list-style-type: none"> 1. Edit Post Successfully: <ul style="list-style-type: none"> • Submit a valid post edit request • Redirect to the post-detail view • Verify the updated title and body in the database 2. Edit Post with Image Upload: <ul style="list-style-type: none"> • Submit a valid post edit request with an image

	<ul style="list-style-type: none">• Verify the image is uploaded and linked to the post• A successful redirect <p>3. Unauthorized User Cannot Edit Post:</p> <ul style="list-style-type: none">• Try to edit a post created by another user• 404 error (post does not belong to the current user) <p>4. Like Post Successfully (Toggle):</p> <ul style="list-style-type: none">• Like a post and verify the "like" status is toggled to True• Un-like the post and verify the "like" status is toggled to False• Correct like count is returned in the response <p>5. Like Post with Invalid Method:</p> <ul style="list-style-type: none">• Like a post using a GET request instead of POST
--	--

	<ul style="list-style-type: none"> • 400 error with an appropriate message
Expected Results	<ul style="list-style-type: none"> • Post editing should work correctly, updating the post details in the database • Image uploads should be successfully associated with posts during editing • Unauthorized users should not be able to edit posts created by others • The like/unlike toggle feature should function correctly via AJAX, updating the like count and status • Invalid HTTP methods for liking posts should return a 400 error with a helpful message
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 31:Unit Test - Post Updating and Liking

```
python manage.py test
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
Ran 5 tests in 10.308s
OK
Destroying test database for alias 'default'...
```

Figure 197: Unit Test - Post Updating and Liking result

4.2.20 Test: Po3 Liking, Deleting and Adding Comments Functionality

Test No.	Po3
Objectives:	Verify the functionality of adding, deleting, and liking comments. Ensure proper error handling when interacting with non-existent comments or posts, and validate unauthorized actions.
Test Case	<p>1. Like Comment Tests</p> <ul style="list-style-type: none"> • Test successful like/unlike toggle • Test liking a non-existent comment <p>2. Delete Comment Tests</p> <ul style="list-style-type: none"> • Test successful comment deletion • Test unauthorized comment deletion <p>3. Add Comments Tests</p> <ul style="list-style-type: none"> • Test successful comment creation • Test comment creation with empty content • Test comment creation on a non-existent post

Actions:	<p>Like Comment Tests:</p> <ol style="list-style-type: none"> 1. Test Successful Like/Unlike Toggle: <ul style="list-style-type: none"> • Like a comment and verify that the like count increases. • Unlike the same comment and verify that the like count resets to 0. 2. Test Liking a Non-Existent Comment: <ul style="list-style-type: none"> • Attempt to like a non-existent comment by using an invalid comment ID. • Verify that the system returns a 404 error page <p>Delete Comment Tests:</p> <ol style="list-style-type: none"> 3. Test Successful Comment Deletion: <ul style="list-style-type: none"> • Delete an existing comment and verify that it is successfully removed from the database.
----------	---

	<p>4. Test Unauthorized Comment Deletion:</p> <ul style="list-style-type: none">• Attempt to delete a comment owned by another user and verify that the system returns a 404 error page. <p>Add Comment Tests:</p> <p>5. Test Successful Comment Creation:</p> <ul style="list-style-type: none">• Add a new comment to a post and verify that it is successfully created, returning the comment's ID and HTML content in the response. <p>6. Test Comment Creation with Empty Content:</p> <ul style="list-style-type: none">• Attempt to add a comment with empty content and verify that the system returns a 400 error with an appropriate error message. <p>7. Test Comment Creation on a Non-Existent Post:</p>
--	---

	<ul style="list-style-type: none"> Attempt to add a comment to a non-existent post by using an invalid post slug. Verify that the system returns a 404 error page.
Expected Results	<ul style="list-style-type: none"> Comments can be successfully liked, unliked, added, and deleted. Unauthorized users should not be able to delete comments they do not own. Non-existent comments or posts should result in 404 error pages. Empty comments should trigger a 400 error with a relevant message.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 32:Unit Test - Like, Delete and Add Comments

```
Found 7 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
Ran 7 tests in 14.421s

OK
Destroying test database for alias 'default'...
PS C:\Users\rebof\OneDrive\Desktop\fYP\Project\petvet\myproject> 
```

Figure 198: Unit Test - Like, Delete and Add Comments Results

4.2.21 Test: Po4 Reply Creation and Deletion Functionality

Test No.	Po4
Objectives:	Verify the functionality of adding and deleting replies to comments. Ensure that unauthorized users cannot delete replies, and handle invalid methods or non-existent comments properly.
Test Case	<p>1. Delete Reply Tests</p> <ul style="list-style-type: none"> • Test Successful reply deletion • Test unauthorized reply deletion • Test invalid HTTPPP method for reply deletion <p>2. Add Reply Tests</p> <p>3. Test successful reply creation</p> <p>4. Test reply creation with empty content</p> <p>5. Test reply creation for a non-existent comment</p> <p>6. Test invalid HTTP method for adding a reply</p>
Actions:	<p>Delete Reply Tests:</p> <p>1. Test Successful Reply Deletion:</p>

	<ul style="list-style-type: none">• Create a reply to a comment and delete it• Verify the reply is successfully deleted, and the response indicates success <p>2. Test Unauthorized Reply Deletion:</p> <ul style="list-style-type: none">• Create a reply owned by another user• Attempt to delete this reply while logged in as a different user• Verify that the system returns a 404 error page <p>3. Test Invalid Method for Reply Deletion:</p> <ul style="list-style-type: none">• Try to delete a reply using a GET request instead of a POST request• Verify that the system returns a 400 error with an appropriate error message
	Add Reply Tests:

	<p>4. Test Successful Reply</p> <p>Creation:</p> <ul style="list-style-type: none">• Post a reply to a comment using an AJAX request• Verify that the reply is successfully created, and the response includes the new reply's ID and HTML content <p>5. Test Reply Creation with Empty Content:</p> <ul style="list-style-type: none">• Post an empty reply and verify that the system returns a 400 error <p>6. Test Reply Creation for Non-Existent Comment</p> <ul style="list-style-type: none">• Try posting a reply to a non-existent comment (using an invalid comment ID)• Verify that the system returns a 404 error page <p>7. Test Invalid Method for Adding Reply:</p>
--	---

	<ul style="list-style-type: none"> • Try adding a reply using a GET request instead of a POST request • Verify that the system returns a 400 error with an appropriate error message
Expected Results	<ul style="list-style-type: none"> • Replies can be added or deleted as expected. • Unauthorized users should not be able to delete replies that they do not own. • Invalid HTTP methods should be properly handled and return a 400 error. • Trying to reply to a non-existent comment should result in a 404 error. • Successful replies should return the correct response data, including the reply's ID and HTML content.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 33:Unit Test - Add and Delete Reply

```
Found 7 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
Ran 7 tests in 14.201s

OK
Destroying test database for alias 'default'...
```

Figure 199: Unit Test - Reply creation and deletion result

4.2.22 Test: RTC1 Real-Time Chat Functionality

Test No.	RTC1
Objectives:	To ensure WebSocket-based chat and notification systems work as intended, with proper message delivery, persistence, and user handling. Test the robustness and responsiveness of the system under various scenarios.
Test Case	<ol style="list-style-type: none"> 1. WebSocket Connection Test <ul style="list-style-type: none"> • Test chat connection establishment 2. Message Exchange Test <ul style="list-style-type: none"> • Verify bidirectional real-time message flow between users 3. Message Persistence Test <ul style="list-style-type: none"> • Ensure sent messages are saved in the database 4. Notification System Test <ul style="list-style-type: none"> • Verify notification is sent to the recipient upon message receipt 5. Invalid User Handling Test <ul style="list-style-type: none"> • Check system stability when interacting with a non-existent user 6. Concurrent Messaging Test <ul style="list-style-type: none"> • Validate system behavior under multiple rapid message sends
Actions:	<ol style="list-style-type: none"> 1. Chat Connection Test:

	<ul style="list-style-type: none">• Connect to the WebSocket endpoint:/ws/chat/chat_<user1_id>_<user2_id>/• Confirm that the connection status is True. <p>2. Message Exchange Test:</p> <ul style="list-style-type: none">• Connect both user1 and user2 to the same chat room.• Send message: "Hello from User1!" from user1.• Verify that user2 receives the exact message. <p>3. Message Persistence Test:</p> <ul style="list-style-type: none">• Connect user1 to the chat room.• Send message: "This is a Test persistence message".• Disconnect and verify that the message is stored in the database with the correct sender (user1) and receiver (user2). <p>4. Notification Flow Test:</p> <ul style="list-style-type: none">• Connect user2 to the notifications WebSocket: /ws/notifications/user2/• Connect user1 to the chat room.• Send message: "This should trigger notification" from user1.
--	--

	<ul style="list-style-type: none"> Verify that user2 receives a real-time notification with the correct message content. <p>5. Invalid User Test:</p> <ul style="list-style-type: none"> Connect to the chat room. Send invalid message: "This is a Test message" with receiver nonexistent_user Send valid message: "Hello! Valid message" to user2. Confirm that the valid message is processed normally without system errors. <p>6. Concurrent Messaging Test:</p> <ul style="list-style-type: none"> Connect user1 and user2 to the same chat room. Send messages in rapid succession: <ul style="list-style-type: none"> i. "Message 1" ii. "Message 2" iii. "Message 3" Confirm that user2 receives all messages in the same order they were sent.
Expected Results	<ul style="list-style-type: none"> WebSocket connects successfully. Real-time messaging works between users.

	<ul style="list-style-type: none"> • Each message is saved in the database correctly. • Notification system triggers on message receipt. • System handles non-existent users gracefully. • All concurrent messages are received without data loss or order mismatch.
Actual Results	All test cases passed (ok in test output)
Conclusion	The test was successful

Table 34:Unit Test - Real Time Chat

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Found 6 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.Message saved to database: user1 to user2: Hello from User1!
.Message saved to database: user1 to user2: Message 1
Message saved to database: user1 to user2: Message 2
Message saved to database: user1 to user2: Message 3
.Error: User not found - User matching query does not exist.
Message saved to database: user1 to user2: Hello! Valid message
.Message saved to database: user1 to user2: This is a Test persistence message
.Message saved to database: user1 to user2: This should trigger notification
.

Ran 6 tests in 18.123s

OK
Destroying test database for alias 'default'...

```

Figure 200: Unit Test - Real time chat result

4.3 System Testing

4.3.1 System Test: Auth1 Pet Owner Registration Functionality

Test no.	Auth1
Objective	To verify that the Pet Owner registration form, profile form, and OTP verification process function correctly when valid data is provided and appropriate error messages are displayed when required or invalid fields are submitted.
Testcase	<ol style="list-style-type: none"> 1. Missing Credentials for Registration Form 2. Successful Completion of Registration Form 3. Missing Credentials for Profile Form 4. Invalid Credentials for Profile Form 5. Successful Completion of Profile Form 6. Invalid OTP Check 7. Valid OTP Check
Action	<ol style="list-style-type: none"> 1. Attempted to register without entering a phone number → system displayed: “Phone number is required”

	<p>2. Filled the registration form with all valid credentials → redirected to the profile completion form.</p> <p>3. Left the pets owned field empty in the profile form → form did not proceed, asked to fill the field.</p> <p>4. Entered a negative number in the pets owned field → validation triggered: "Please enter a valid number".</p> <p>5. Entered a valid number (2) in the field → successfully redirected to the OTP verification page.</p> <p>6. Entered invalid OTP (99999) → system rejected the code, did not proceed.</p> <p>7. Entered correct OTP (810084) → redirected to the community/home page.</p>
Expected Result	<ul style="list-style-type: none"> • System must validate each form step and display appropriate error messages for: <ul style="list-style-type: none"> ✓ Missing or invalid inputs (e.g., phone number, pet count, OTP). • On valid input:

	<ul style="list-style-type: none"> ✓ Redirect to the next form or page in the flow. • OTP must be verified before granting access to the main platform.
Actual Results	The system displayed appropriate validation messages and allowed progression only after all valid inputs were provided.
Conclusion	The test was successful

Table 35: System Test - Pet Owner Registration

Result:

The screenshot shows the PetVet mobile application's registration interface. The 'Register' tab is active. The form fields are as follows:

- Full Name: Test Name
- Email: testuser@gmail.com
- Username: testusername
- Phone: (Empty field with a red border and the error message "Phone number is required." below it)
- Gender: Male
- User Type: Pet Owner
- Password: (Empty field)
- Confirm Password: (Empty field)

A green 'Register' button is located at the bottom of the form. Below the form, there is a link to log in: "Already have an account? [Login here](#)".

Figure 201: Leaving the phone number field empty

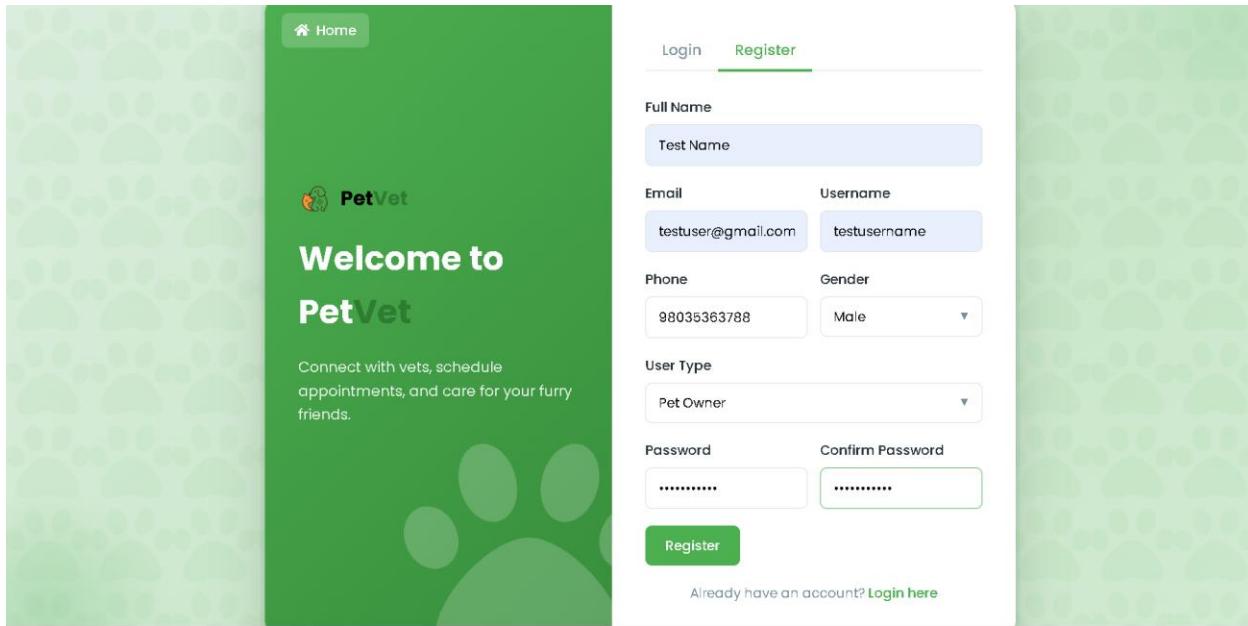
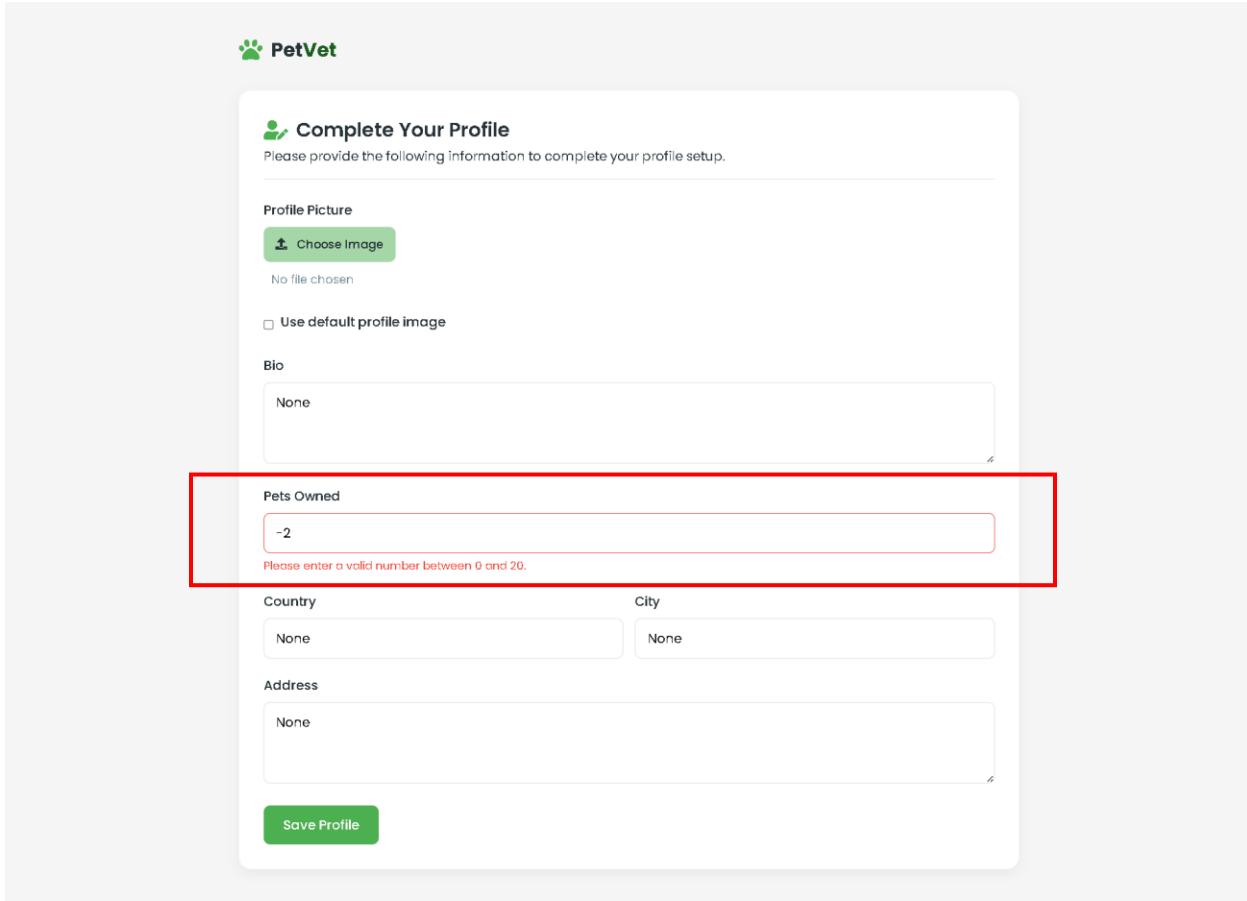


Figure 202: Registering with correct credentials

The screenshot shows the PetVet profile completion page. At the top, there is a section titled 'Complete Your Profile' with a sub-section 'Profile Picture' containing a 'Choose Image' button and a note 'No file chosen'. Below it is a checkbox 'Use default profile image'. The next section is 'Bio' with a text area containing 'None'. The 'Pets Owned' section is highlighted with a red border; it has a placeholder 'e.g., 2 dogs, 1 cat' and a note 'Please enter a valid number.' Below this are fields for 'Country' (Nepal) and 'City' (Lalitpur). The 'Address' field contains 'Bagdol'. At the bottom is a green 'Save Profile' button.

Figure 203: Leaving the Pets owned field empty



The screenshot shows a 'Complete Your Profile' form on the PetVet platform. The 'Profile Picture' section includes a file input field with 'Choose Image' and a note 'No file chosen'. A checkbox for 'Use default profile image' is present. The 'Bio' section contains a text area with 'None'. The 'Pets Owned' section is highlighted with a red border; it has a text input field containing '-2' and an error message 'Please enter a valid number between 0 and 20.' Below this are fields for 'Country' (None) and 'City' (None). An 'Address' field contains 'None'. At the bottom is a green 'Save Profile' button.

Figure 204: Entering negative integer in the Pet owned field

The screenshot shows the 'Complete Your Profile' page of the PetVet application. At the top, there's a logo with a paw print icon and the word 'PetVet'. Below it, a section titled 'Complete Your Profile' with a subtitle 'Please provide the following information to complete your profile setup.' A 'Profile Picture' field has a placeholder 'Choose Image' and a note 'No file chosen'. There's also a checkbox for 'Use default profile image'. A 'Bio' field contains the text 'None'. Under 'Pets Owned', the number '2' is entered in a field with a validation message 'Please enter a valid number.'. Below this, 'Country' is set to 'Nepal' and 'City' is set to 'Lalitpur'. An 'Address' field contains 'Bagdog'. At the bottom is a green 'Save Profile' button.

Figure 205: Entering a valid data and Submitting the profile

The screenshot shows the 'Email Verification' page. At the top, a green banner says 'Profile successfully updated!'. Below it, a section says 'Hi Test Name, please verify your email address by entering the OTP code sent to your email.' An 'Enter OTP' field contains a dashed code '-----'. A green 'Verify' button is below it. At the bottom is a red 'Logout' button.

Figure 206: Redirected to OTP Verification

284

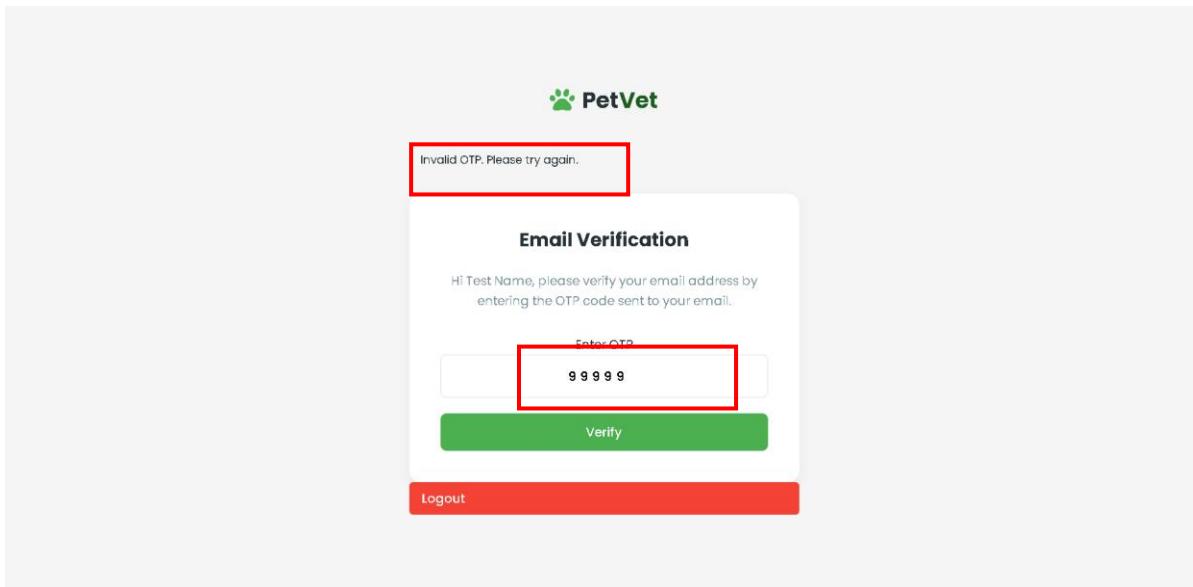


Figure 207: Entering invalid OTP

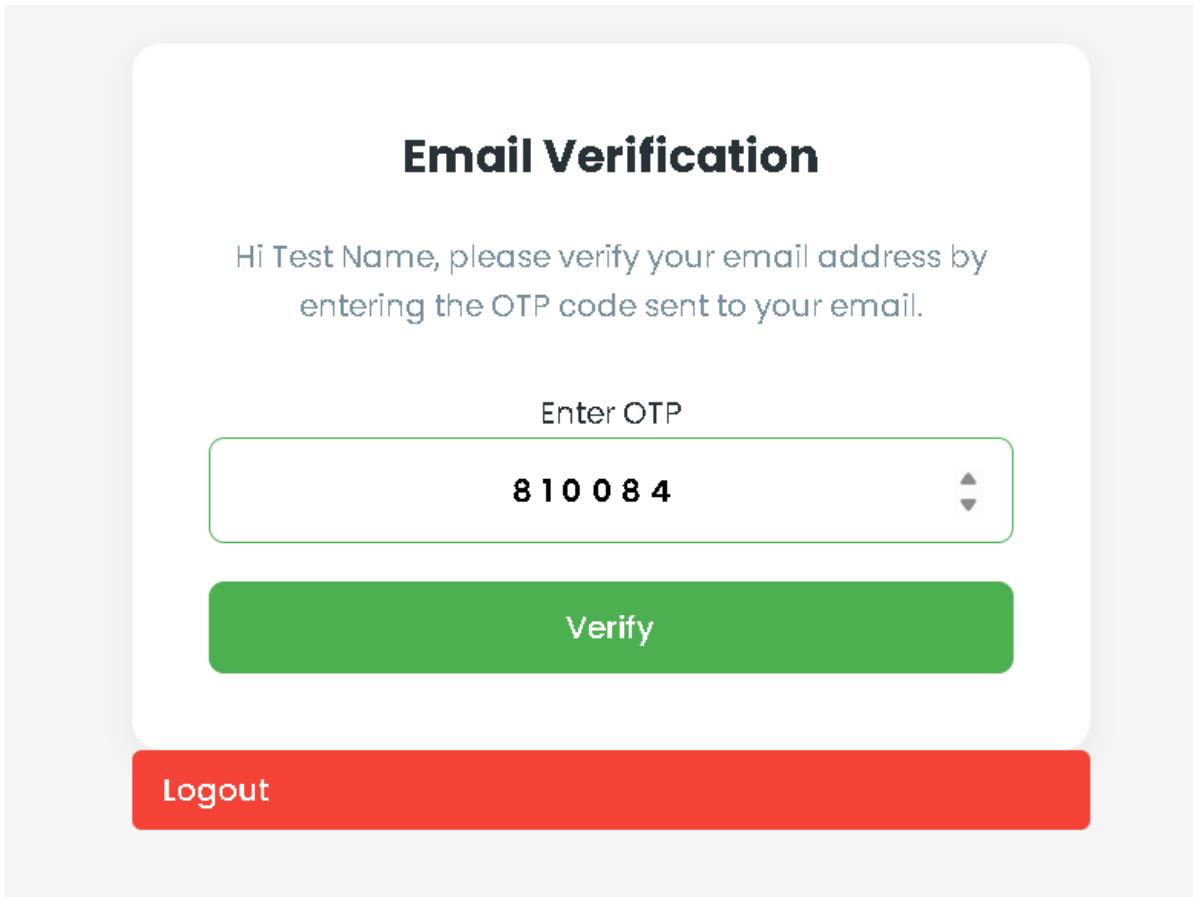


Figure 208: Entering the correct OTP

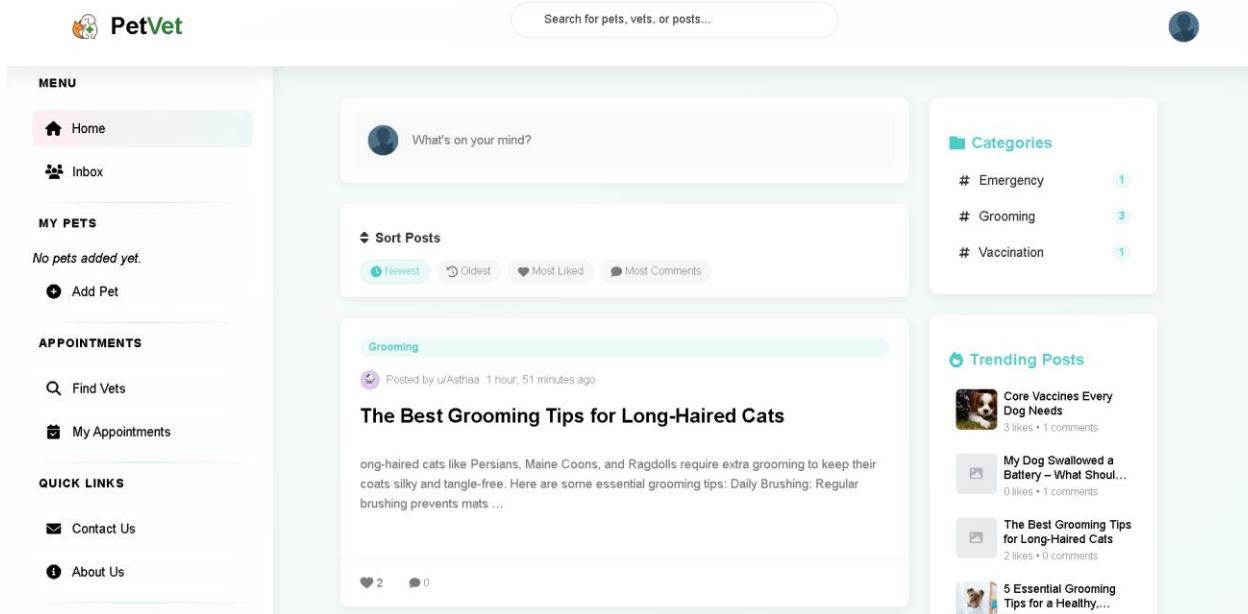


Figure 209: Successful Pet Owner Registration

4.3.2 System Test: Auth2 Login and Logout Functionality

Test no.	Auth2
Objective	To verify that the login and logout functionality works correctly, handling invalid credentials and confirming proper redirection on success.
Testcase	<ol style="list-style-type: none"> 1. Login Attempt with Invalid Credentials 2. Login Attempt with Valid Credentials 3. Successful Logout
Action	<ol style="list-style-type: none"> 1. Entered testuser@gmail.com with an incorrect password — prompted with “Invalid username or password.” 2. Entered the same email with the correct password — successfully redirected to the homepage. 3. Clicked logout — redirected to the login page with a message confirming successful logout.
Expected Result	<ul style="list-style-type: none"> • The system should reject incorrect credentials with an error message. • Upon valid credentials, the user should be logged in and taken to the homepage.

	<ul style="list-style-type: none"> • Logging out should end the session and redirect to the login page with a confirmation message.
Actual Results	The system handled invalid credentials with an error, allowed access with valid input, and confirmed logout with appropriate feedback.
Conclusion	The test was successful

Table 36: System Test - Login and Logout

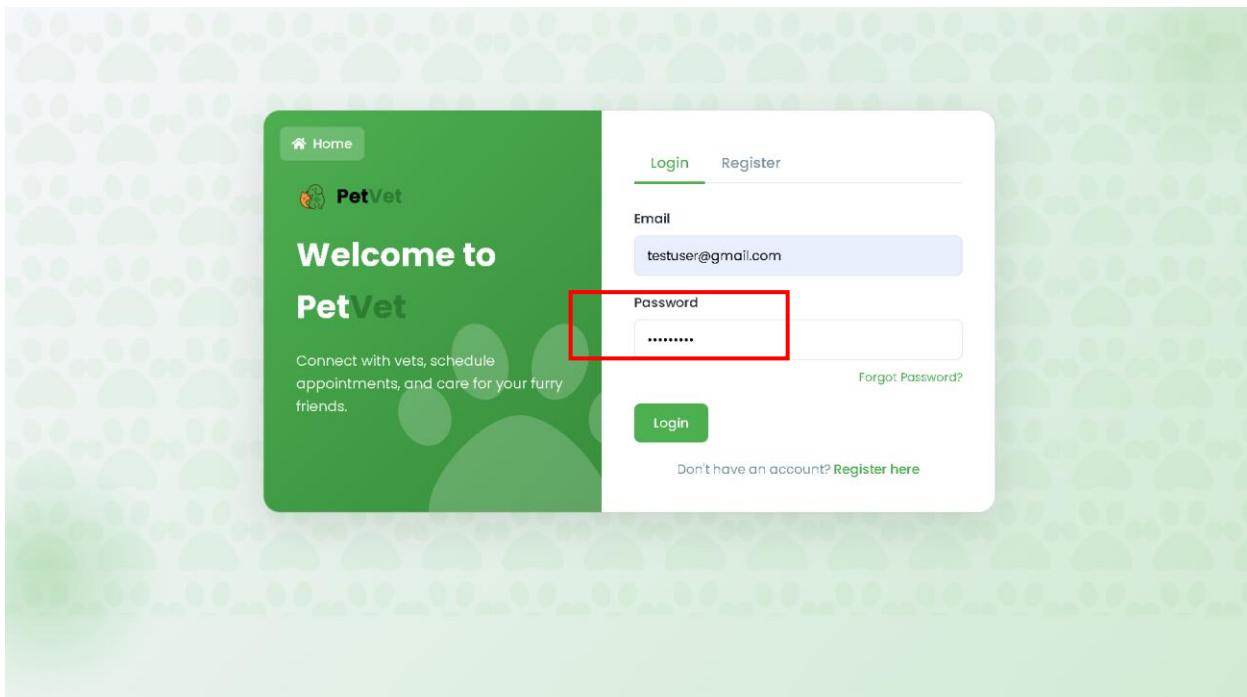


Figure 210: Entering incorrect credentials

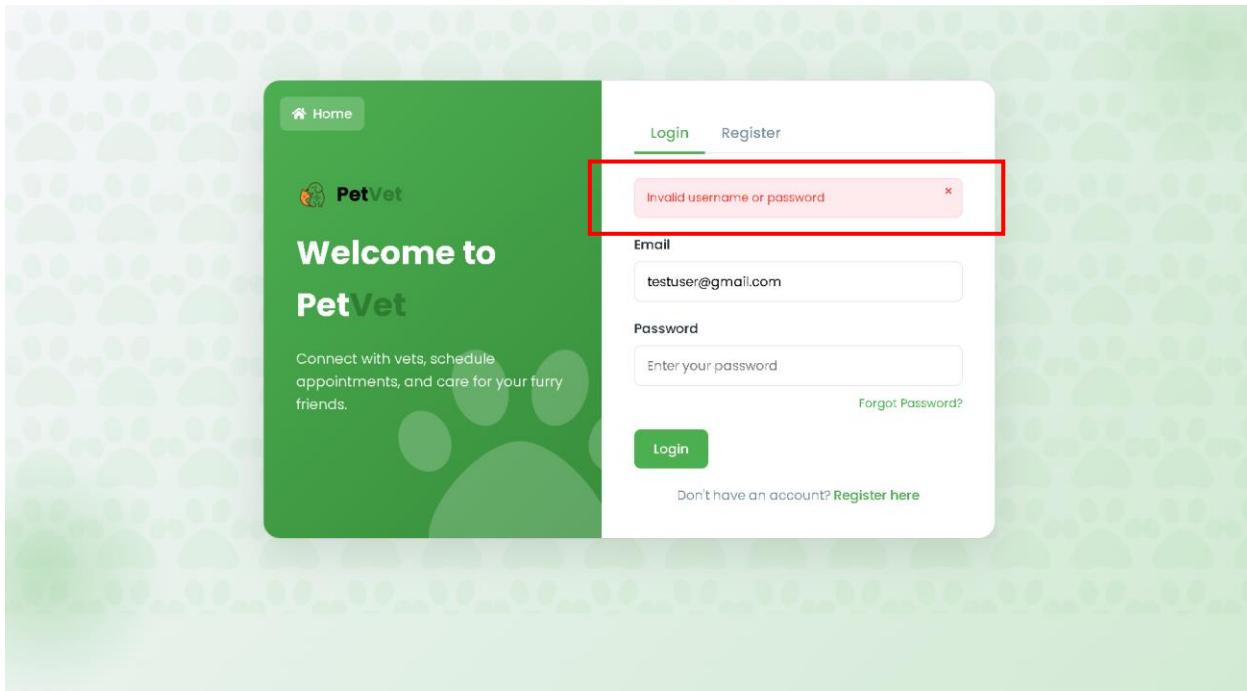


Figure 211: Authentication error message displayed upon invalid login attempt.

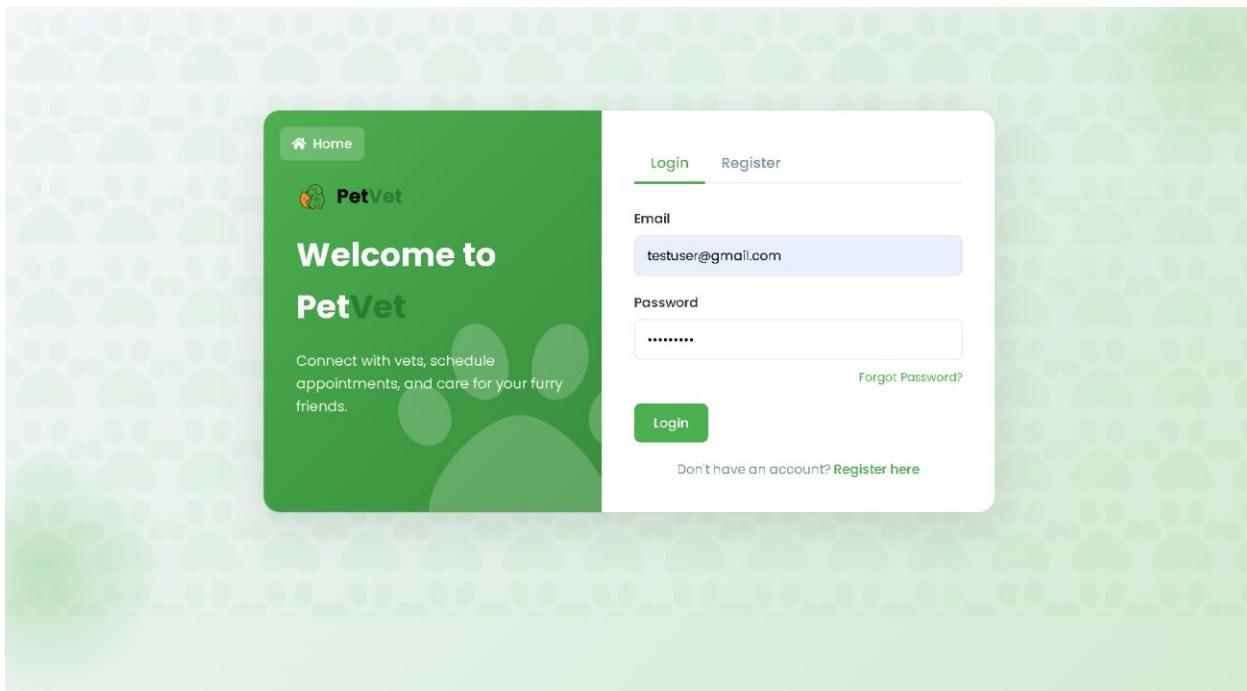


Figure 212: Entering correct credentials

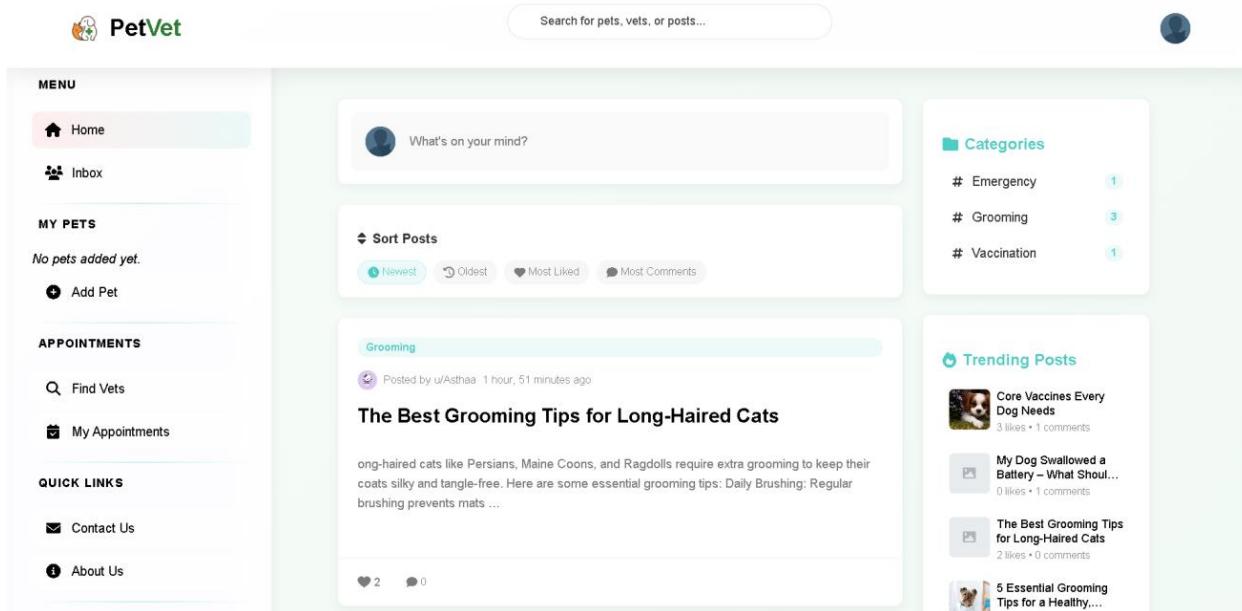


Figure 213: Successfully Logged In

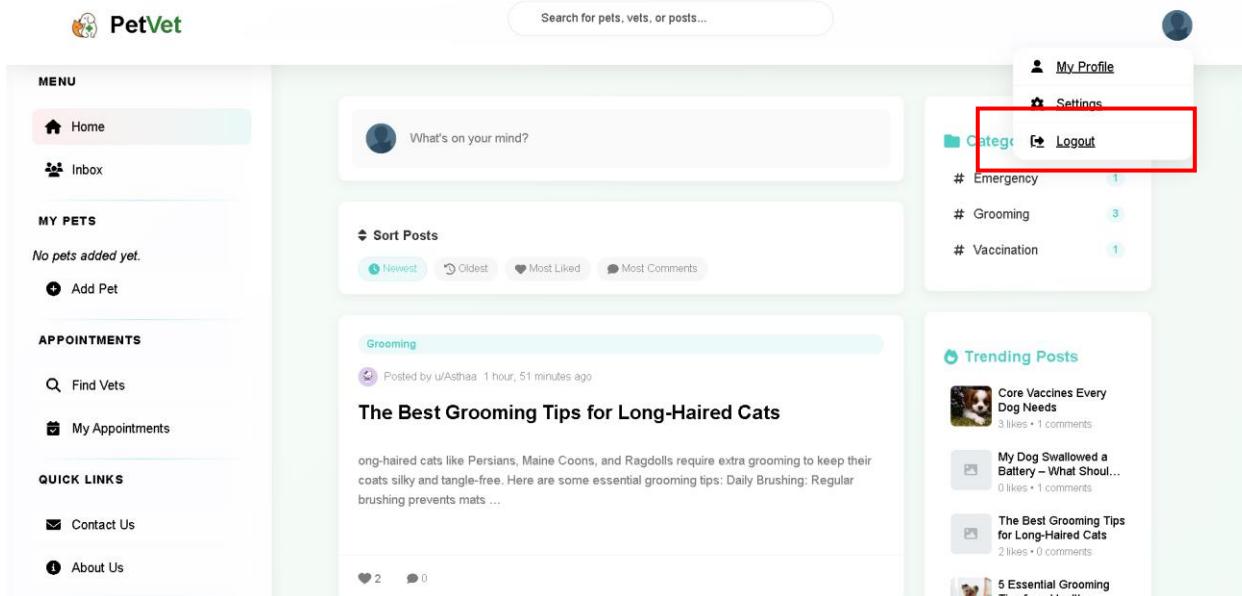


Figure 214: Clicking the logout button

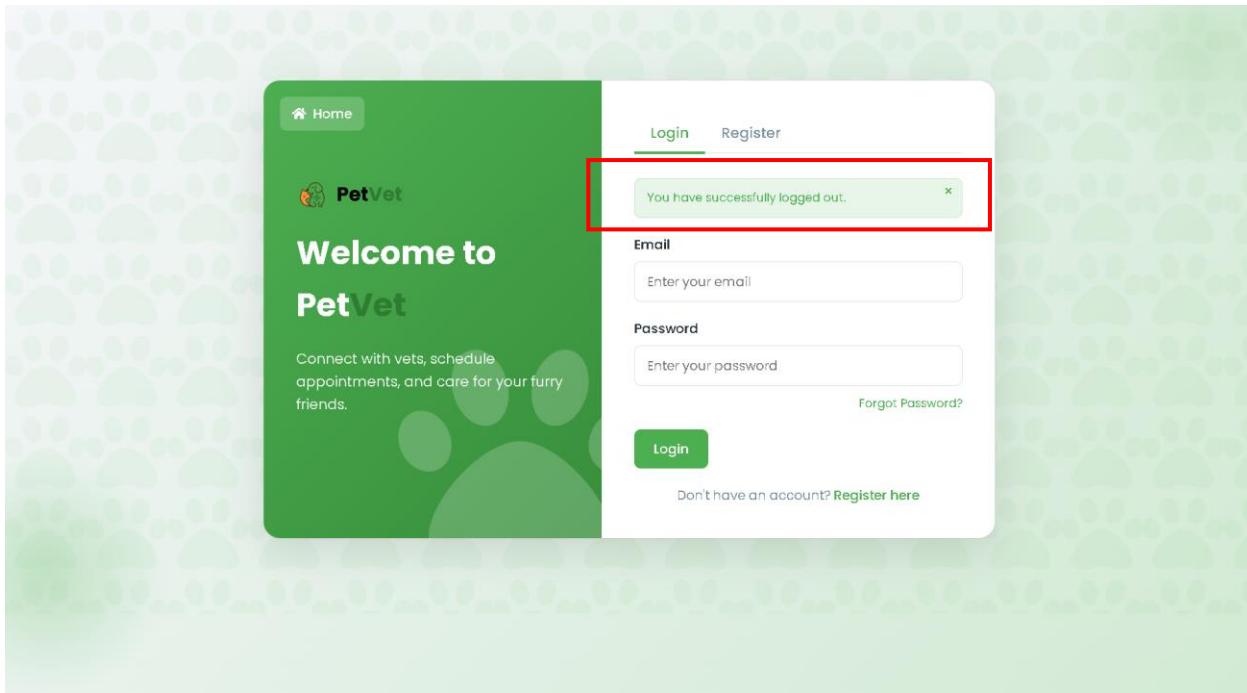


Figure 215: Successfully Logs out

4.3.3 System Test: Auth3 Vet Registration Functionality

Test no.	Auth3
Objective	To verify that the vet registration process correctly validates input, guides users through profile completion, and allows admin approval or rejection accordingly.
Testcase	<ol style="list-style-type: none"> 1. Missing Username in Registration Form 2. Password Mismatch during Registration 3. Successful Completion of Vet Registration Form 4. Missing Credentials in Vet Profile Form 5. Successful Vet Profile Completion 6. Admin Declines Vet Profile 7. Re-attempt with Correct Profile Info 8. Admin Approves Vet Profile
Action	<ol style="list-style-type: none"> 1. All fields were correctly filled except the username field, which was left empty — prompted with a message to fill it. 2. Then a password mismatch was submitted — received a prompt to match passwords.

	<p>3. On entering valid data, successfully redirected to the vet-specific profile form.</p> <p>4. Left the “number of years of experience” field empty — prompted to complete it.</p> <p>5. Entered 2 as valid input — redirected to a “verification pending” page.</p> <p>6. Admin logged into the dashboard and declined the vet profile — vet was redirected to the profile form for corrections.</p> <p>7. After submitting corrected profile info, admin approved it — vet was now fully registered and verified.</p>
Expected Result	<ul style="list-style-type: none"> • System should validate each step and prevent submission of incomplete or mismatched data. • Upon admin decline, the vet should be able to re-submit their profile. • Admin should be able to approve or decline based on profile correctness.
Actual Results	System enforced validation at every step, redirected appropriately, and

	allowed profile re-submission upon decline, followed by successful registration after admin approval.
Conclusion	The test was successful

Table 37: System Test - Vet Registration

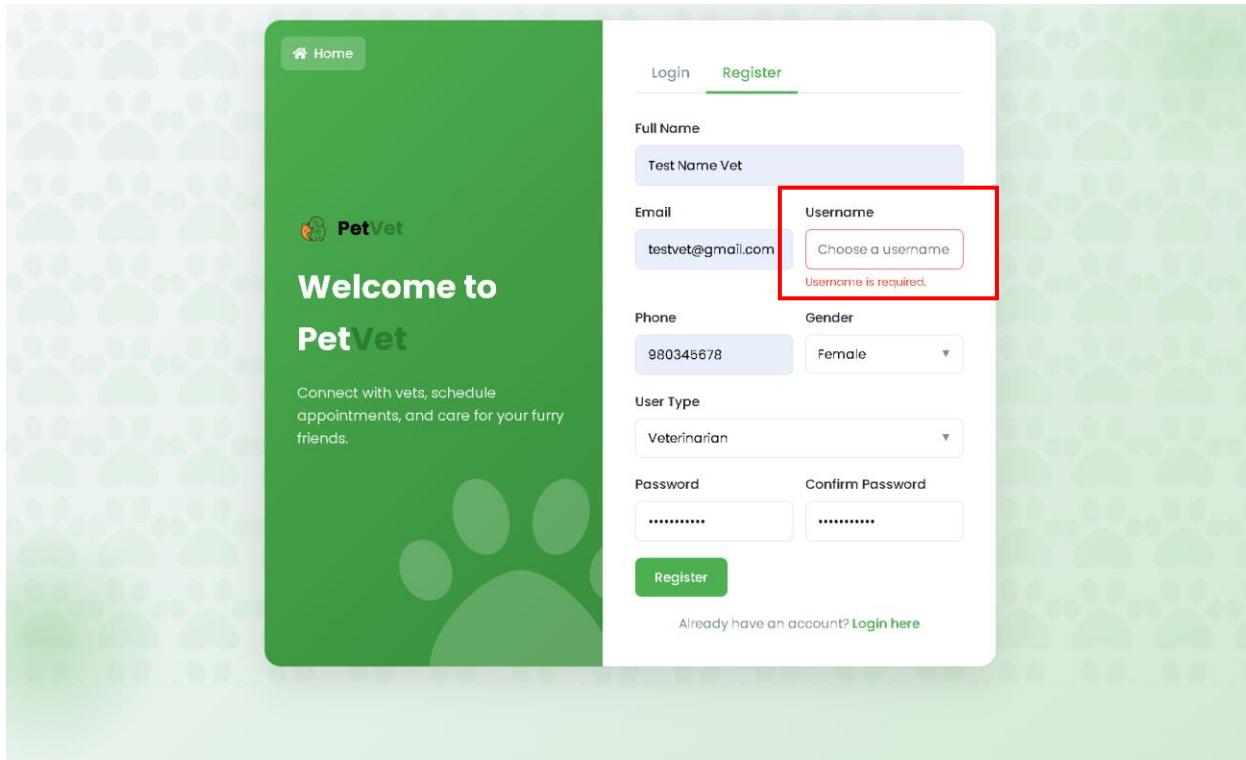


Figure 216: Leaving the username field empty

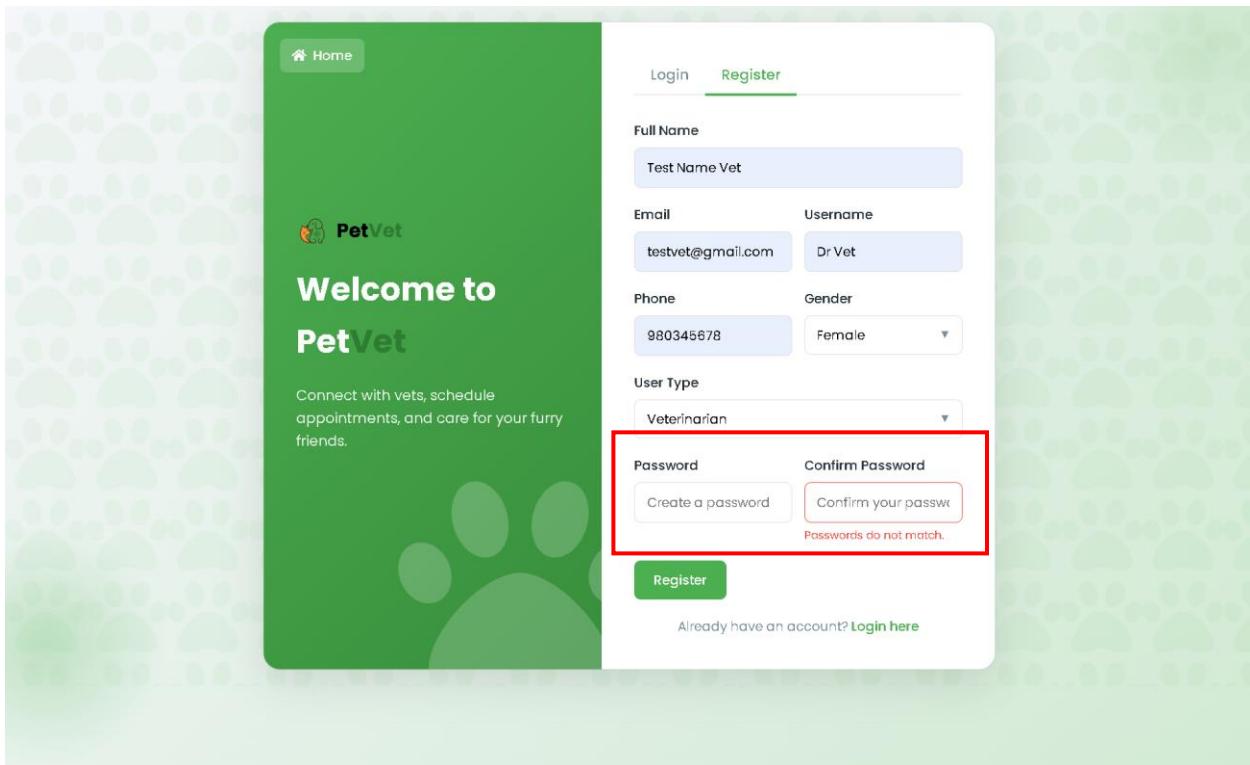


Figure 217: Mismatching the password fields

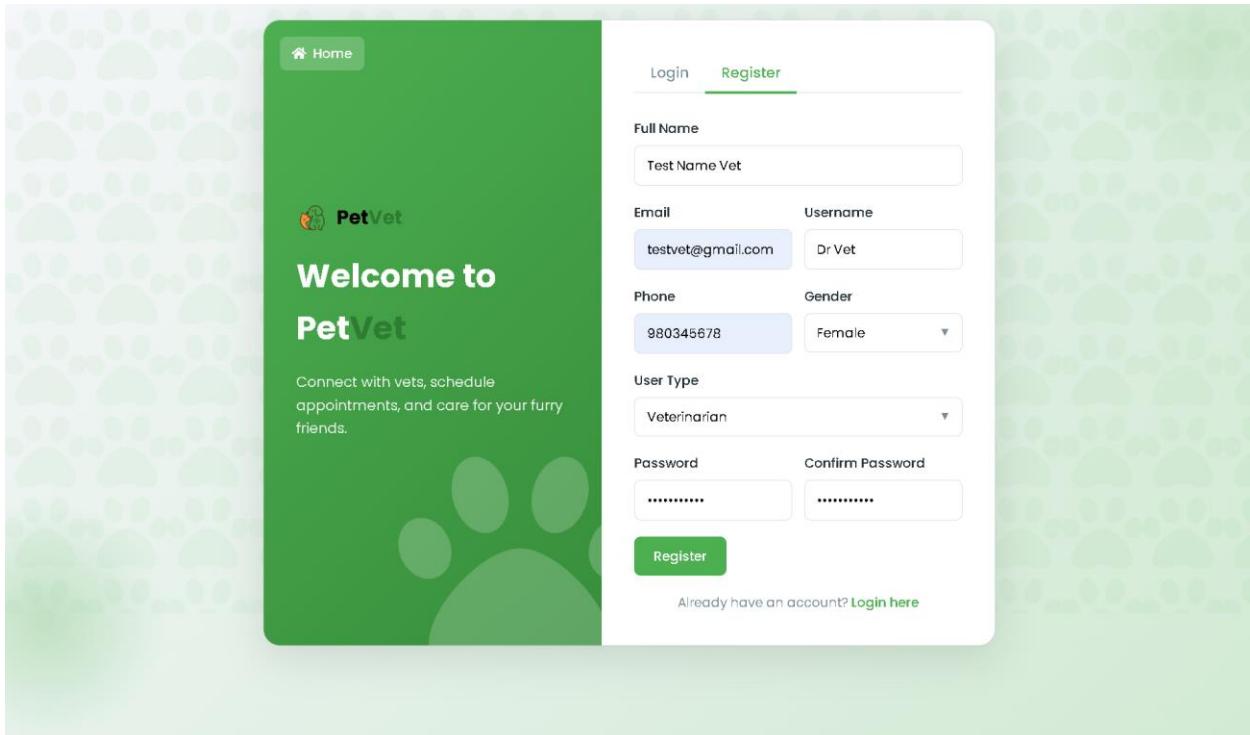


Figure 218: Entering the Correct fields

The screenshot shows the 'Complete Your Profile' page of the PetVet application. The page has a header with the PetVet logo and a title 'Complete Your Profile'. It asks for information to complete profile setup. The fields include:

- Profile Picture:** A file input field with a 'Choose Image' button and a note 'No file chosen'. There is also a checkbox for 'Use default profile image'.
- Professional Summary:** A text area containing 'None'.
- Clinic Name:** A text area containing 'None'.
- Specialization:** A text area containing 'None'.
- Years of Experience:** A text input field with a placeholder 'Enter years of experience' and a validation message 'Please enter a valid number.' This field is highlighted with a red border.
- License Number:** A text area containing 'None'.
- Country:** A text area containing 'None'.
- City:** A text area containing 'None'.
- Clinic Address:** A text area containing 'None'.

A green 'Save Profile' button is at the bottom.

Figure 219: Leaving the years of experience field empty

After entering the correct information, we are redirected to waiting page.

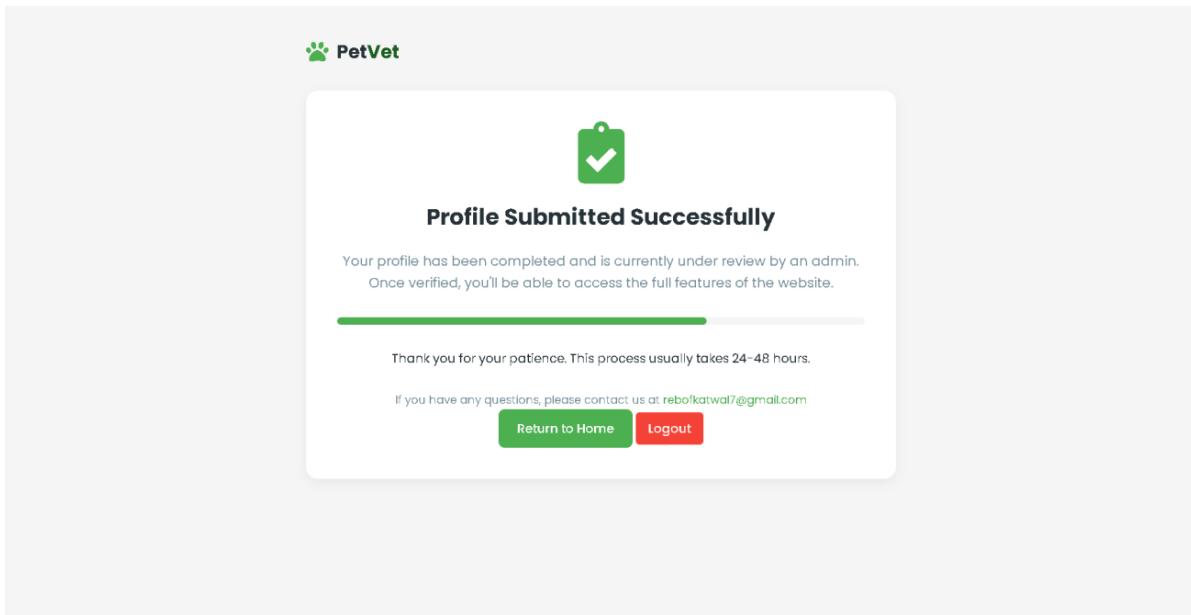


Figure 220: Redirected to Waiting page

PetVet Admin

Vet Approvals

Pending Approvals 1 Vets waiting for approval

Recently Approved 3 Vets approved in the last 7 days

Recently Declined 1 Vets declined in the last 7 days

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
Test Name Vet	testvet@gmail.com	None	NVC-NP-2024-89897	2 years	Apr 25, 2025	View Approve Decline

Recently Approved Vets

Name	Email	Clinic Name	License Number	Approved At
Shubham Datta Mishra	shubhammishra@gmail.com	Mishra Pet Clinic	PAW-1234556	Apr 25, 2025
Avishek Gautam	avishekgautam@gmail.com	Where Pets Come First Clinic	PAW-123999	Apr 25, 2025
Gage Colon	rykono@mailinator.com	Tanya Patton	638	Apr 25, 2025

Recently Declined Vets

Name	Email	Clinic Name	License Number	Declined At
Jordan Paul	lagazypuw@mailinator.com	Burke Kirkland	857	Apr 25, 2025

[Logout](#)

Figure 221: Admin dashboard for vet approval

Vet Approvals

Veterinarian 'Test Name Vet' has been declined.

Pending Approvals	Recently Approved	Recently Declined
0 Vets waiting for approval	3 Vets approved in the last 7 days	2 Vets declined in the last 7 days

Pending Vet Approvals

No pending vet approvals.

Recently Approved Vets

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
Shubham Datta Mishra	shubhammishra@gmail.com	Mishra Pet Clinic	PAW-1234556		Apr 25, 2025	
Avishek Gautam	avishekga@gmail.com	Where Pets Come First Clinic	PAW-123999		Apr 25, 2025	
Gage Colon	rykono@mailinator.com	Tanya Patton	638		Apr 25, 2025	

Recently Declined Vets

Name	Email	Clinic Name	License Number	Declined At
Test Name Vet	testvet@gmail.com	None	NVC-NP-2024-89897	Apr 25, 2025
Jordan Paul	lagyzypuw@mailinator.com	Burke Kirkland	857	Apr 25, 2025

[Logout](#)

Figure 222: Vet has been declined

If the vet attempts to log in again, they will be redirected to the profile page, where they must re-enter their information for approval.

The screenshot shows the PetVet profile setup page. At the top, there is a green header bar with the text "Profile successfully updated!". Below this, the main form has a title "Complete Your Profile" with the sub-instruction "Please provide the following information to complete your profile setup." The form includes the following fields:

- Profile Picture:** A file input field with a "Choose Image" button. It displays the message "No file chosen". There is also a checkbox option "Use default profile image".
- Professional Summary:** A text area containing the text "Good with small animals as well".
- Clinic Name:** An input field containing "Test clinic".
- Specialization:** An input field containing "Exotic Birds".
- Years of Experience:** An input field containing "2".
- License Number:** An input field containing "NVC-NP-2024-89897".
- Country:** An input field containing "Nepal".
- City:** An input field containing "Kathmandu".
- Clinic Address:** An input field containing "Sitapaila".

At the bottom of the form is a green "Save Profile" button.

Figure 223: Redirected to profile page again

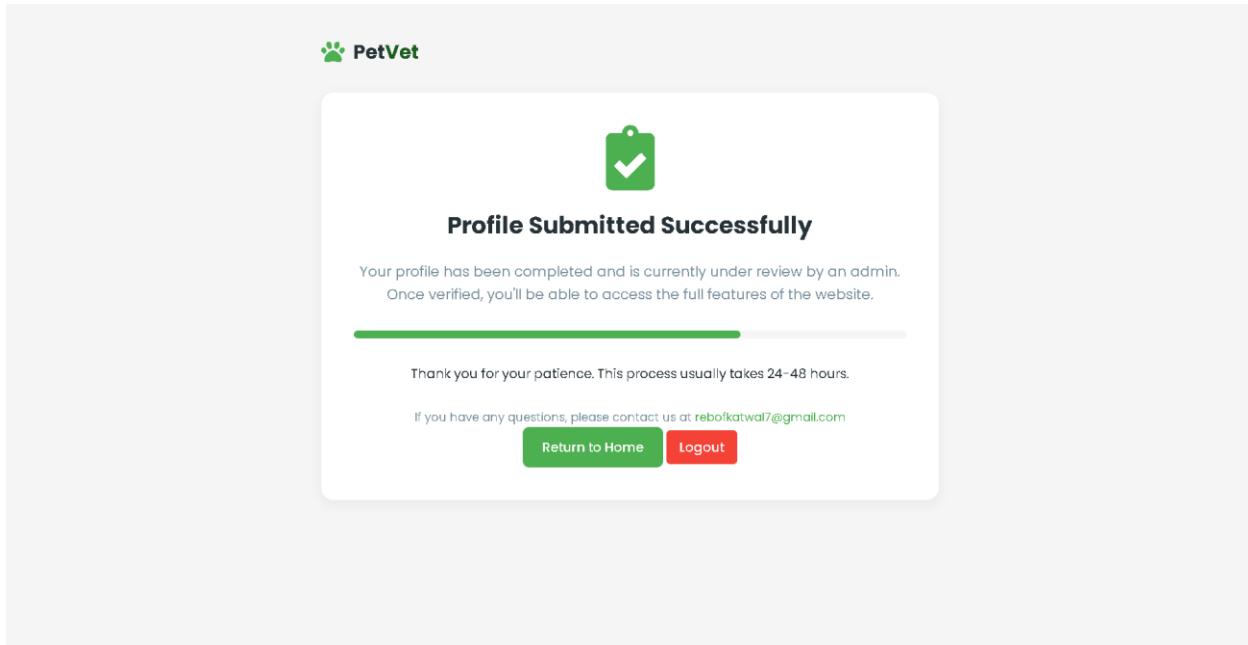


Figure 224: Waiting for approval again

Vet Approvals

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
Test Name Vet	testvet@gmail.com	None	NVC-NP-2024-89897	2 years	Apr 25, 2025	View Approve Decline

Recently Approved Vets

Name	Email	Clinic Name	License Number	Approved At
Shubham Datta Mishraa	shubhammishra@gmail.com	Mishra Pet Clinic	PAW-1234556	Apr 25, 2025
Avishek Gautam	avishekgautam@gmail.com	Where Pets Come First Clinic	PAW-123999	Apr 25, 2025
Gage Colon	rykono@mailinator.com	Tanya Patton	638	Apr 25, 2025

Recently Declined Vets

Name	Email	Clinic Name	License Number	Declined At
Jordan Paul	lagyzyupw@mailinator.com	Burke Kirkland	857	Apr 25, 2025

[Logout](#)

Figure 225: Admin views the profile

PetVet Admin

Vet Details

Test Name Vet Veterinarian Unverified

Account Information

Username Dr Vet	Email testvet@gmail.com	Full Name Test Name Vet	Phone 980345678
Gender F	Joined April 25, 2025		

Professional Information

Clinic Name Test clinic	License Number NVC-NP-2024-89897	Specialization Exotic Birds	Experience 2 years
Summary Good with small animals as well	Average Rating 0 / 5	Country Nepal	City Kathmandu
Address Sitapaila			

[Logout](#)

Figure 226:Admin approves of the Vet

The screenshot shows the PetVet Admin application interface. On the left is a sidebar with a user profile for 'adminRebof' (Administrator), navigation links for Dashboard, User Management, Post Management, and Vet Approvals (which is currently selected and highlighted in pink), and a Categories link. The main content area is titled 'Vet Approvals'. It features a success message: 'Veterinarian 'Test Name Vet' has been approved successfully.' Below this are three summary boxes: 'Pending Approvals' (0, Vets waiting for approval), 'Recently Approved' (4, Vets approved in the last 7 days), and 'Recently Declined' (1, Vets declined in the last 7 days). The 'Recently Approved' section is highlighted with a red box. The 'Pending Vet Approvals' table is empty. The 'Recently Approved Vets' table lists four entries:

Name	Email	Clinic Name	License Number	Approved At
Test Name Vet	testvet@gmail.com	Test clinic	NVC-NP-2024-89897	Apr 25, 2025
Gage Colon	rykono@mailinator.com	Tanya Patton	638	Apr 25, 2025
Shubham Datta Mishraa	shubhammishra@gmail.com	Mishra Pet Clinic	PAW-1234556	Apr 25, 2025
Avishek Gautam	avishekga@gmail.com	Where Pets Come First Clinic	PAW-123999	Apr 25, 2025

The 'Recently Declined Vets' table lists one entry:

Name	Email	Clinic Name	License Number	Declined At
Jordan Paul	lagyzypuwi@mailinator.com	Burke Kirkland	857	Apr 25, 2025

[Logout](#)

Figure 227: Approved Vet has been listed in the dashboard

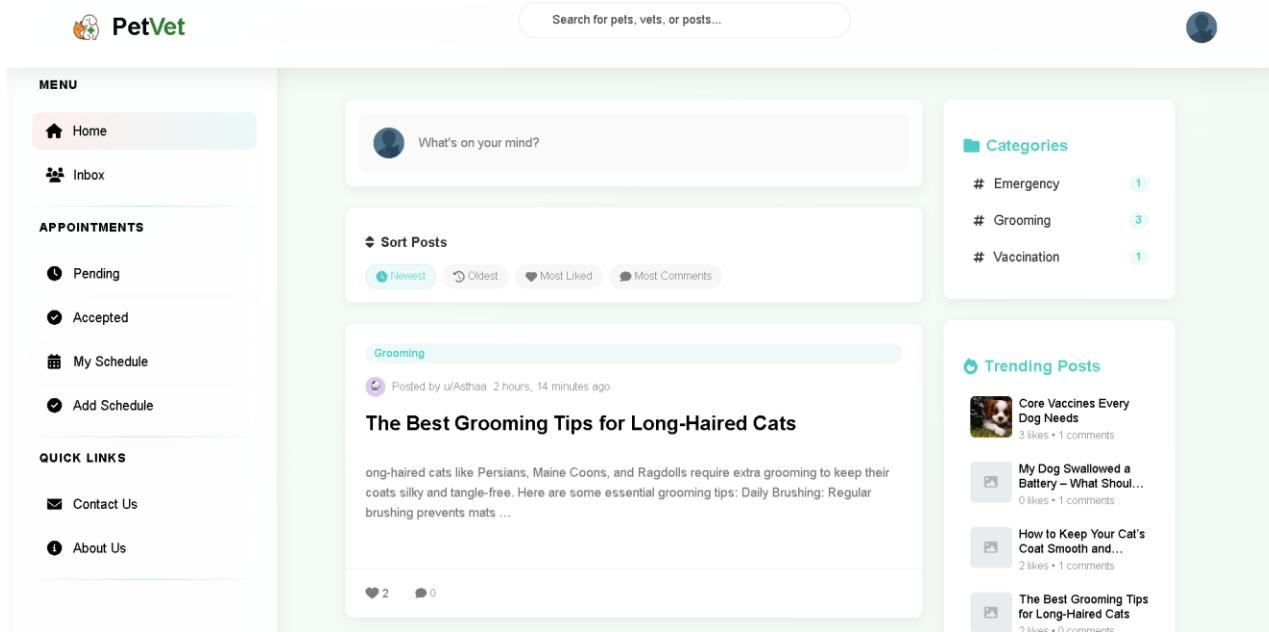


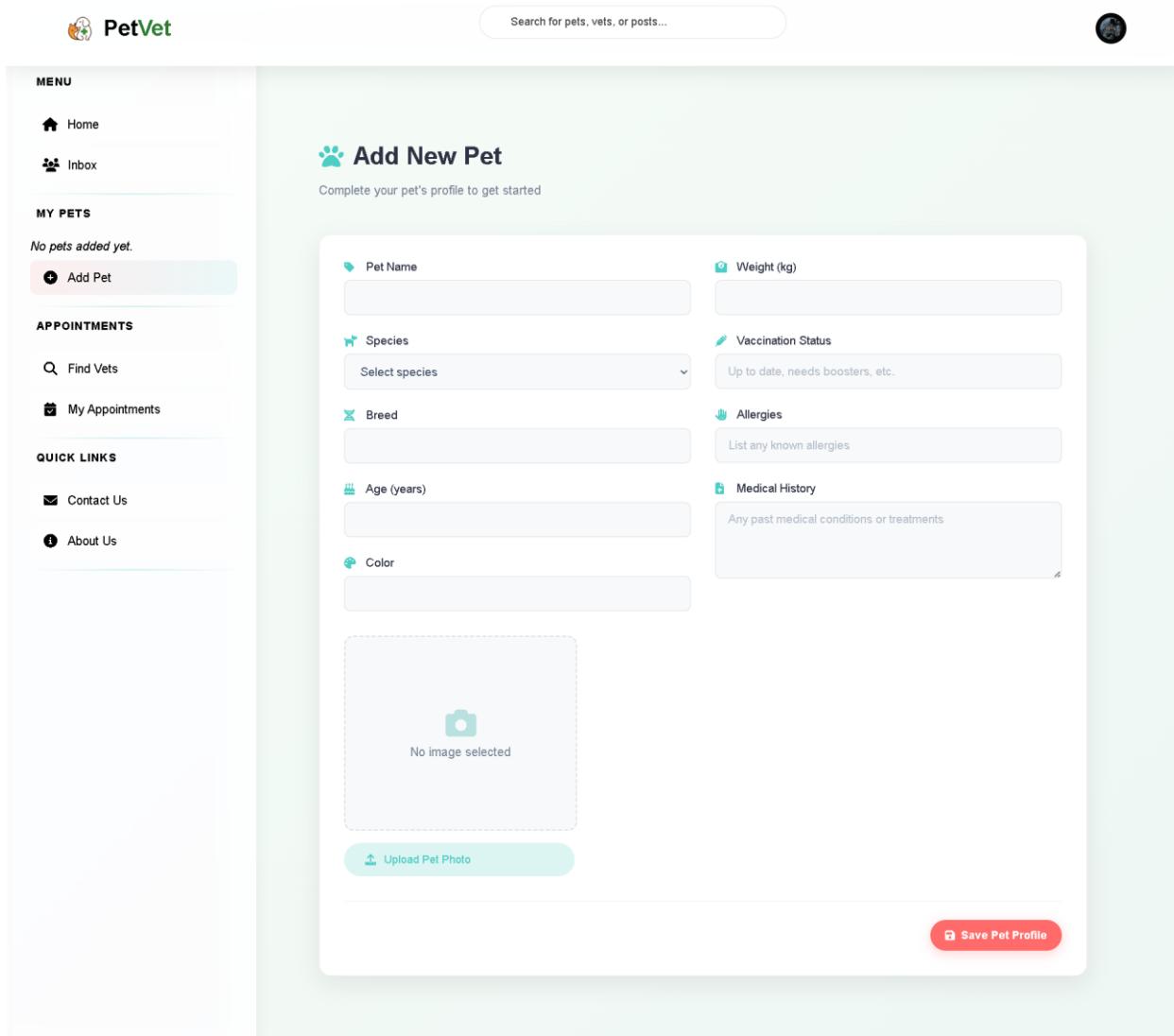
Figure 228: Now the Vet can access the home page

4.3.4 System Test: Appt1 Pet Addition, Deletion and Updating Functionality

Test no.	Appt1
Objective	To verify that the pet addition, update, and deletion process works correctly with proper validations and user feedback throughout.
Testcase	<ol style="list-style-type: none"> 1. Attempt to Add Pet with Missing Fields 2. Successfully Add a Pet 3. Attempt to Update Pet with Invalid Data 4. Successfully Update Pet 5. Delete Pet with Confirmation
Action	<ol style="list-style-type: none"> 1. Left fields like pet name, age, and color empty — prompted to fill all required fields. 2. Entered valid data along with a picture — pet was successfully registered. 3. Attempted to update pet info with - 1 as age — prompted to enter a positive integer. 4. Updated age to 9 — pet info updated successfully.

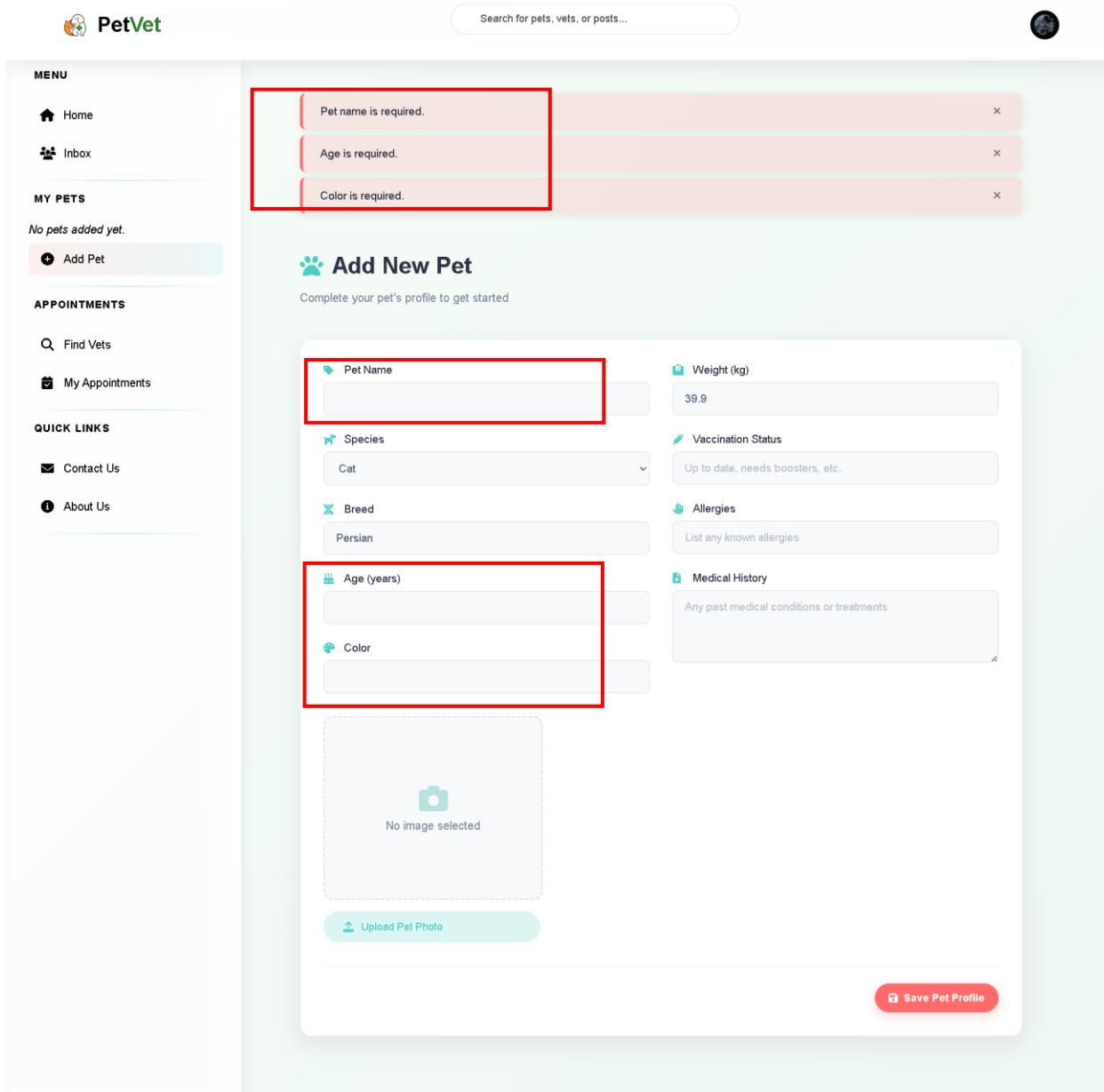
	<p>5. Clicked the delete button — received an alert box asking for confirmation.</p> <ul style="list-style-type: none"> ✓ Confirmed deletion — pet was successfully removed from the website with a deletion message.
Expected Result	<ul style="list-style-type: none"> • Form should validate required fields and allow only valid data. • Pet data should be added, updated, or deleted with appropriate user feedback.
Actual Results	The system validated inputs correctly, performed CRUD operations as expected, and displayed appropriate messages for each action.
Conclusion	The test was successful

Table 38: System Test - Pet Adding, Updating and Deleting



The screenshot shows the 'Add New Pet' page from the PetVet website. The left sidebar contains a 'MENU' with 'Home', 'Inbox', and 'Add Pet' (which is highlighted with a light green background). Below that are sections for 'MY PETS' (with a message 'No pets added yet.'), 'APPOINTMENTS' (with 'Find Vets' and 'My Appointments'), and 'QUICK LINKS' (with 'Contact Us' and 'About Us'). The main content area has a title 'Add New Pet' and a sub-instruction 'Complete your pet's profile to get started'. It features several input fields: 'Pet Name' (placeholder 'Pet Name'), 'Weight (kg)' (placeholder 'Weight (kg)'), 'Species' (dropdown menu 'Select species'), 'Vaccination Status' (text input 'Up to date, needs boosters, etc.'), 'Breed' (placeholder 'Breed'), 'Allergies' (text input 'List any known allergies'), 'Age (years)' (placeholder 'Age (years)'), 'Medical History' (text input 'Any past medical conditions or treatments'), and 'Color' (placeholder 'Color'). Below these is a dashed box placeholder for a photo with the text 'No image selected' and a blue button 'Upload Pet Photo'. At the bottom right is a red button 'Save Pet Profile'.

Figure 229: Add pet page



The screenshot shows the PetVet mobile application interface. On the left is a sidebar with 'MENU' containing 'Home', 'Inbox', and 'MY PETS' (which says 'No pets added yet.'), and 'APPOINTMENTS' with 'Find Vets' and 'My Appointments'. Below that is a 'QUICK LINKS' section with 'Contact Us' and 'About Us'. On the right is the main content area titled 'Add New Pet' with the sub-instruction 'Complete your pet's profile to get started'. It features a form with several fields: 'Pet Name' (highlighted with a red box), 'Species' (set to 'Cat'), 'Breed' (set to 'Persian'), 'Age (years)' (highlighted with a red box), 'Color' (highlighted with a red box), 'Weight (kg)' (set to '39.9'), 'Vaccination Status' (set to 'Up to date, needs boosters, etc.'), 'Allergies' (text input field), and 'Medical History' (text input field). At the bottom are 'Upload Pet Photo' and 'Save Pet Profile' buttons.

Pet name is required.
Age is required.
Color is required.

Add New Pet

Complete your pet's profile to get started

Pet Name

Species

Cat

Breed

Persian

Age (years)

Color

Weight (kg)

39.9

Vaccination Status

Up to date, needs boosters, etc.

Allergies

List any known allergies

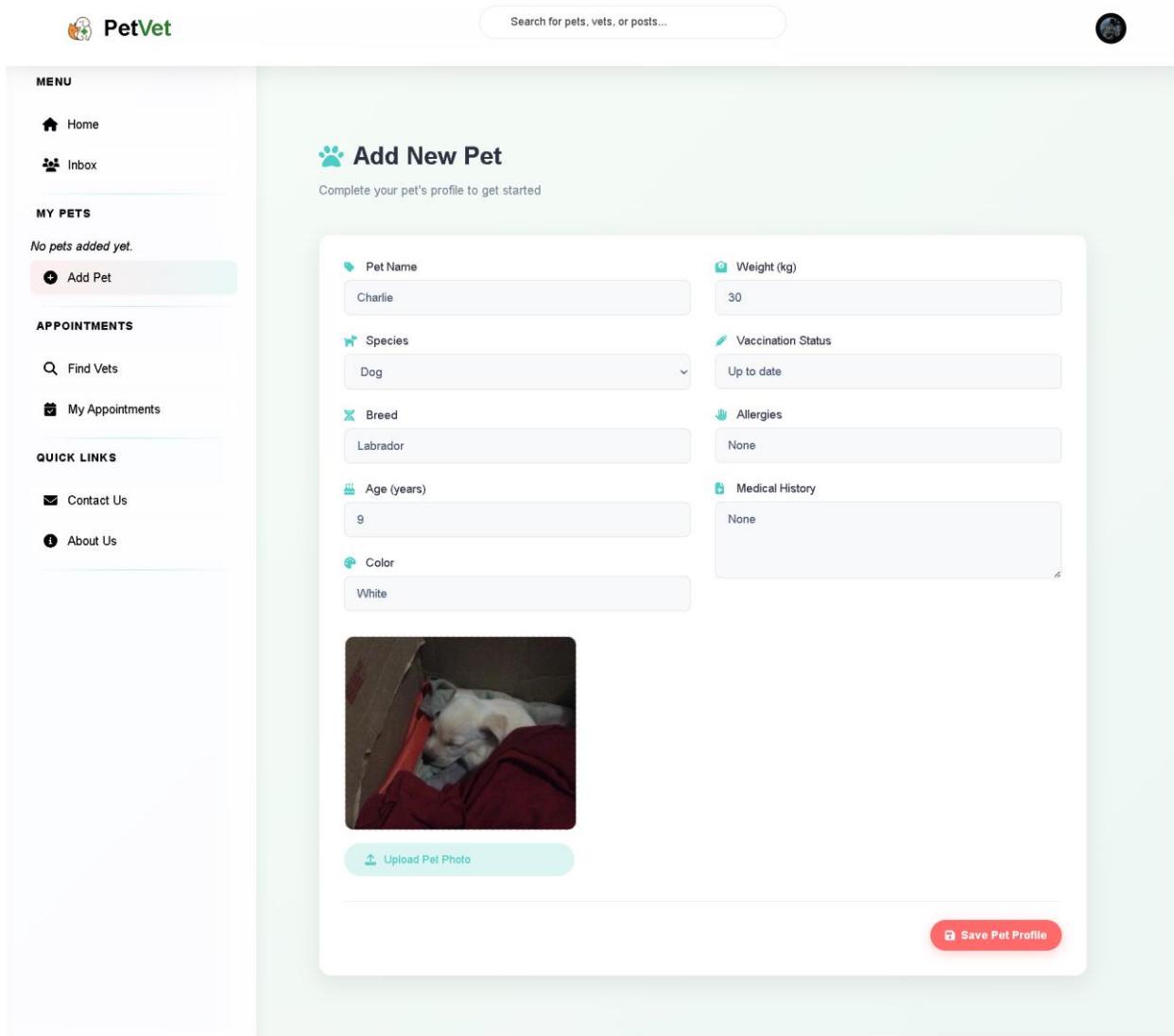
Medical History

No image selected

Upload Pet Photo

Save Pet Profile

Figure 230: Leaving the Age, Color and Name field empty



The screenshot shows the PetVet website interface. On the left, there's a sidebar with a logo, a search bar at the top, and several menu sections: **MENU** (Home, Inbox), **MY PETS** (No pets added yet, Add Pet), **APPOINTMENTS** (Find Vets, My Appointments), and **QUICK LINKS** (Contact Us, About Us). The main content area is titled **Add New Pet** with the sub-instruction **Complete your pet's profile to get started**. The form fields include: **Pet Name** (Charlie), **Weight (kg)** (30); **Species** (Dog), **Vaccination Status** (Up to date); **Breed** (Labrador), **Allergies** (None); **Age (years)** (9), **Medical History** (None); and **Color** (White). Below the form is a thumbnail image of a white dog lying down, with a button to **Upload Pet Photo**. At the bottom right is a red **Save Pet Profile** button.

Figure 231: Entering the complete information for the pet

The screenshot shows the PetVet mobile application interface. On the left is a vertical navigation menu with sections: MENU (Home, Inbox), MY PETS (Charlie (dog) selected, Add Pet), APPOINTMENTS (Find Vets, My Appointments), and QUICK LINKS (Contact Us, About Us). At the top right is a search bar with placeholder text "Search for pets, vets, or posts...". The main content area is titled "Charlie's Profile" with a subtitle "View and manage your pet's information". It features a circular profile picture of a dog named Charlie. Below the picture are three status indicators: "dog", "9 years old", and "Labrador". A blue "Edit Profile" button is located in the top right corner of this section. The main content area is divided into three sections: "Basic Information" (Color: White, Weight: 30.00 kg), "Health Information" (Vaccination Status: Up to date, Allergies: None), and "Medical History" (None). A red "Delete Pet" button is located at the bottom right of the main content area.

Figure 232: Successfully adding the pet

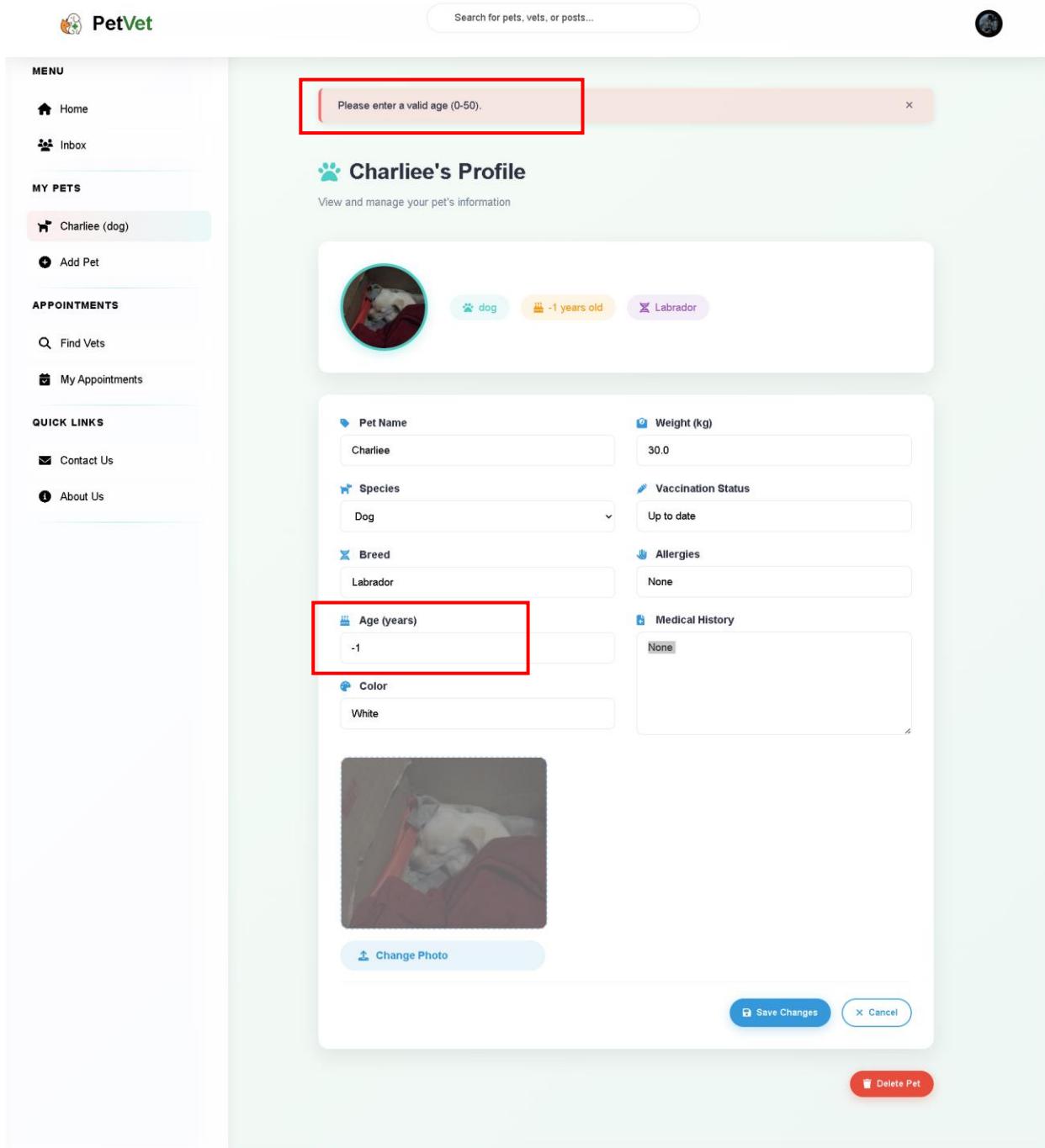


Figure 233: Attempting to update the age field with negative integer

Again, updating the age with positive integer “9”.

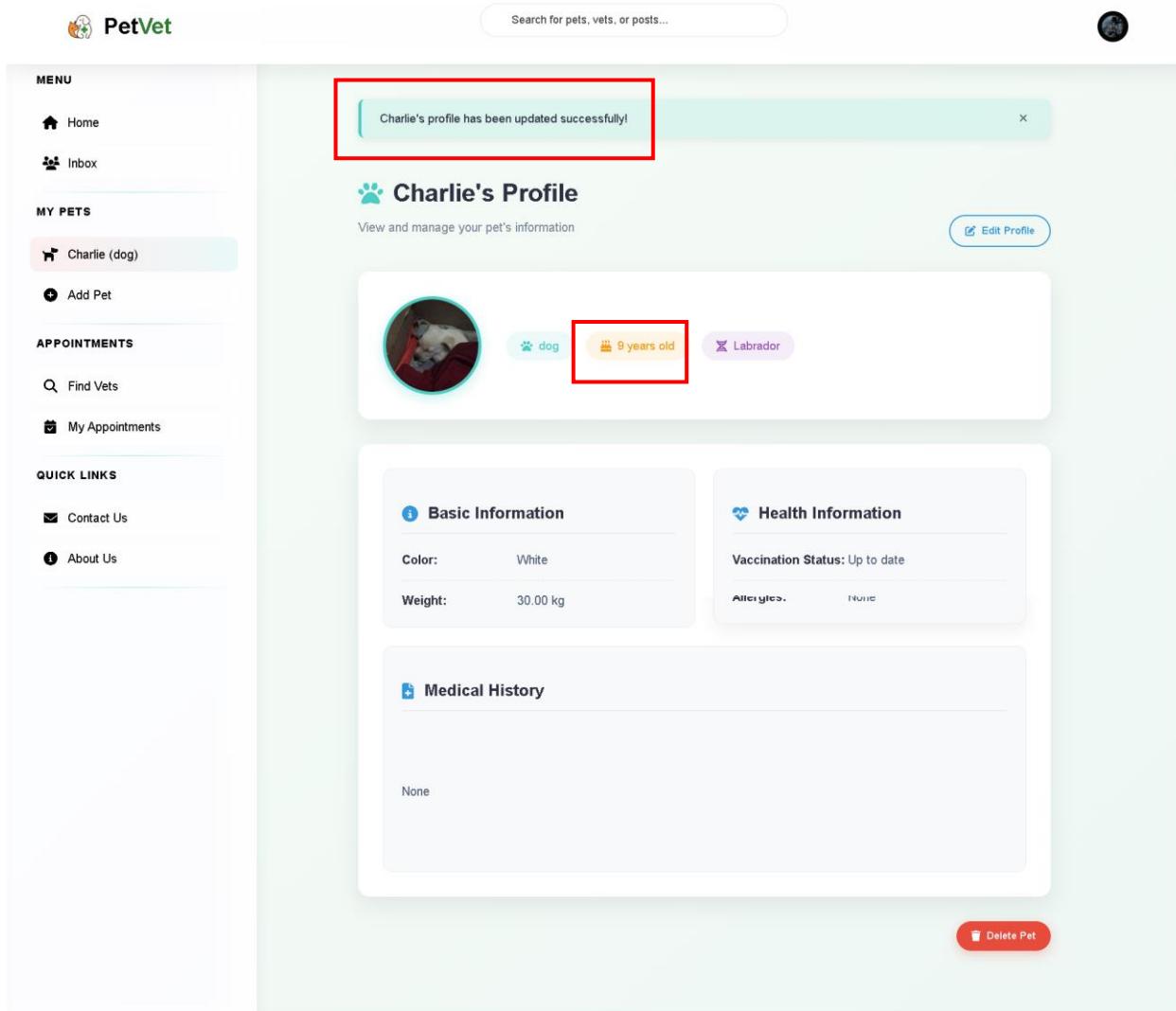


Figure 234: Successfully updated the pet profile

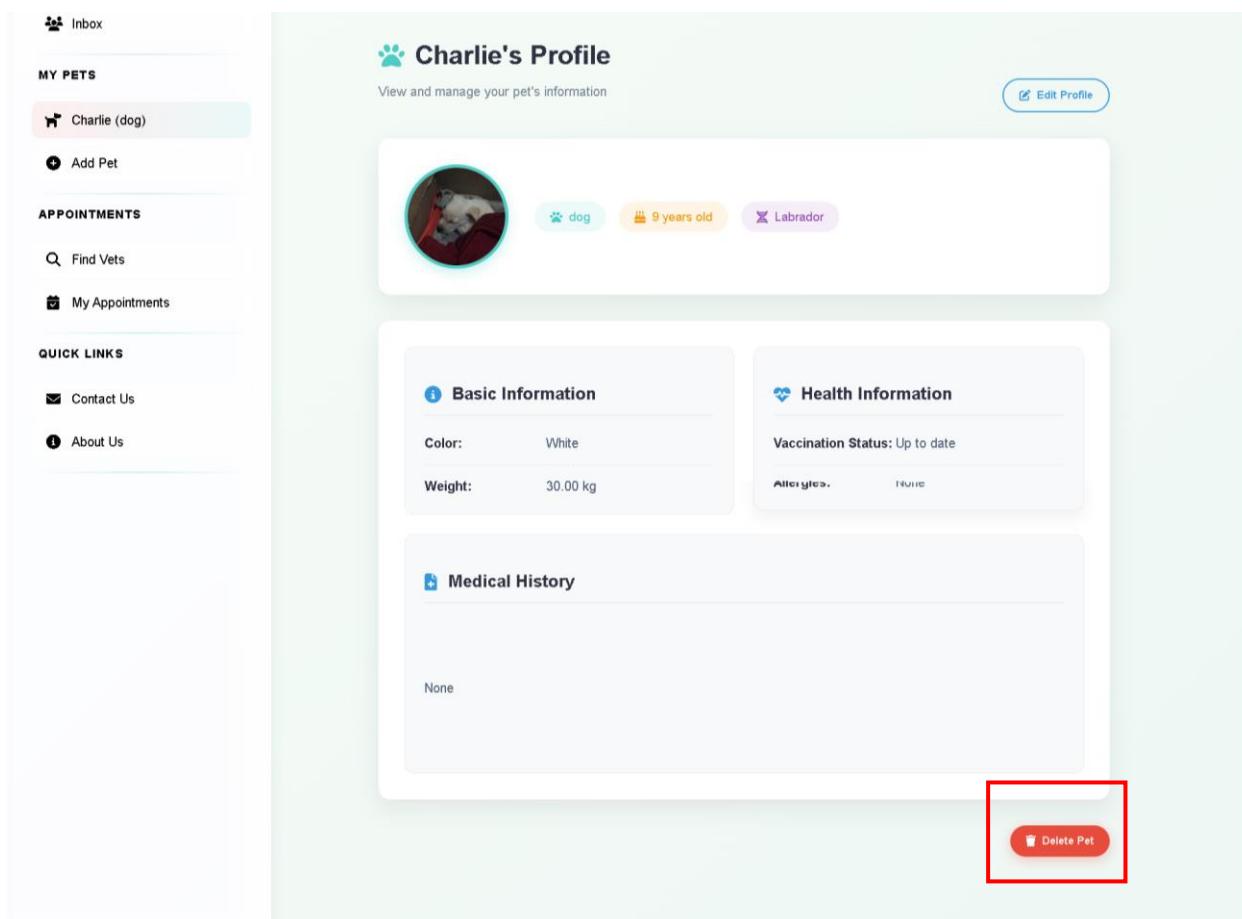


Figure 235: Clicking on the delete button

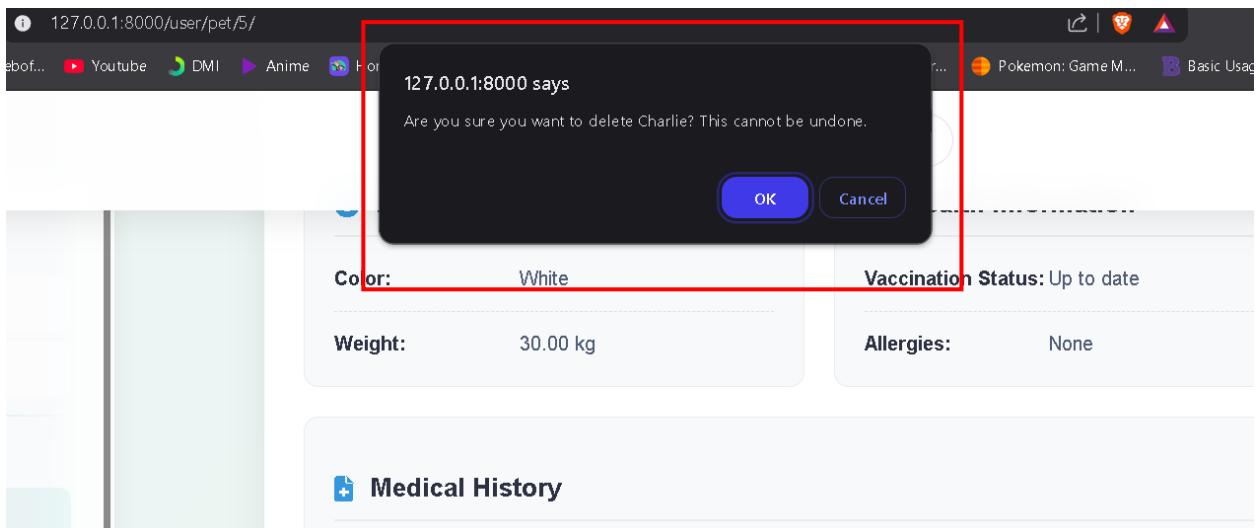


Figure 236: An alert box pops up

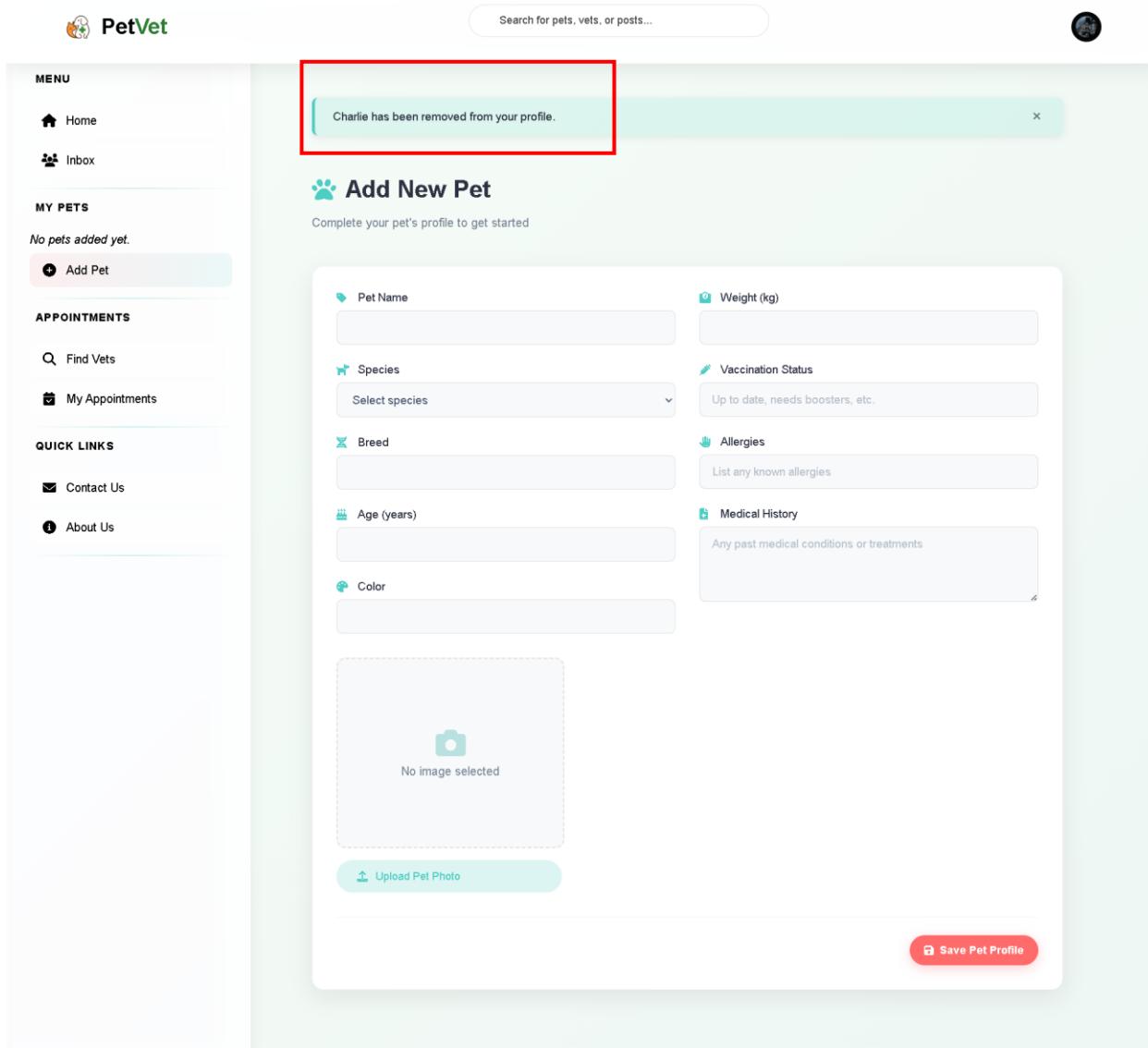


Figure 237: Redirected to add pet page after deletion

4.3.5 System Test: Appt2 Vet Schedule Addition, Deletion and Updating Check

Test no.	Appt2
Objective	To verify that the vet schedule system handles overlapping detection, valid time constraints, editing, and deletion correctly.
Testcase	<ol style="list-style-type: none"> 1. Attempt to Add Overlapping Schedule 2. Successfully Add Non-overlapping Schedule 3. Attempt to Edit Schedule to Overlap 4. Attempt to Set End Time Before Start Time 5. Successfully Edit Schedule 6. Successfully Delete Schedule
Action	<ol style="list-style-type: none"> 1. Overlapping Schedule Addition <ul style="list-style-type: none"> • Logged in as vet "Shubham Datta Mishra" and viewed existing Thursday schedule (10:30 AM – 11:30 AM). • Attempted to add overlapping schedule on Thursday (10:40 AM – 10:30 AM) — prompted that schedule overlaps.

	<ol style="list-style-type: none"> 2. Added valid new schedule (Thursday 5:30 PM – 6:30 PM) — addition successful. 3. Edited original Thursday schedule (10:30 AM – 11:30 AM) to overlap with the new one (starting at 5:35 PM) — prompted for overlap, reverted to original time. 4. Tried setting end time before start time (Start: 10:30 AM, End: 9:30 AM) — prompted that end time must be after start time. 5. Properly updated to Thursday 10:30 AM – 12:00 PM — update successful. 6. Clicked delete button — received alert box for confirmation and confirmed — schedule deleted with success message.
Expected Result	<ul style="list-style-type: none"> • System should block overlapping time entries and invalid time logic. • Valid schedules should be added or updated successfully. • Deletion should prompt for confirmation and proceed accordingly.

Actual Results	The schedule system handled validation, overlap prevention, and CRUD operations smoothly with appropriate prompts.
Conclusion	The test was successful

Table 39: System Test - Schedule Addition, Deletion and Updating

The screenshot shows the PetVet application interface. On the left, there is a sidebar with a logo, a search bar at the top, and several menu items: Home, Inbox, Pending, Accepted, My Schedule (which is selected and highlighted in green), and Add Schedule. Below these are Quick Links for Contact Us and About Us. The main content area is titled "Dr. Shubham Datta Mishraa's Schedule" and shows a list of days: Sunday, Monday, Tuesday, Wednesday, Thursday (which is selected and highlighted in blue), Friday, Saturday. Under Thursday, there is a time slot listed as "10:30 AM - 11:30 AM". This time slot is enclosed in a red rectangular box. To the right of the slot, there are two buttons: "Available" (green) and "Edit" (yellow). At the bottom of the schedule view, there is a button labeled "+ Add New Slot".

Figure 238: Viewing an old time slot

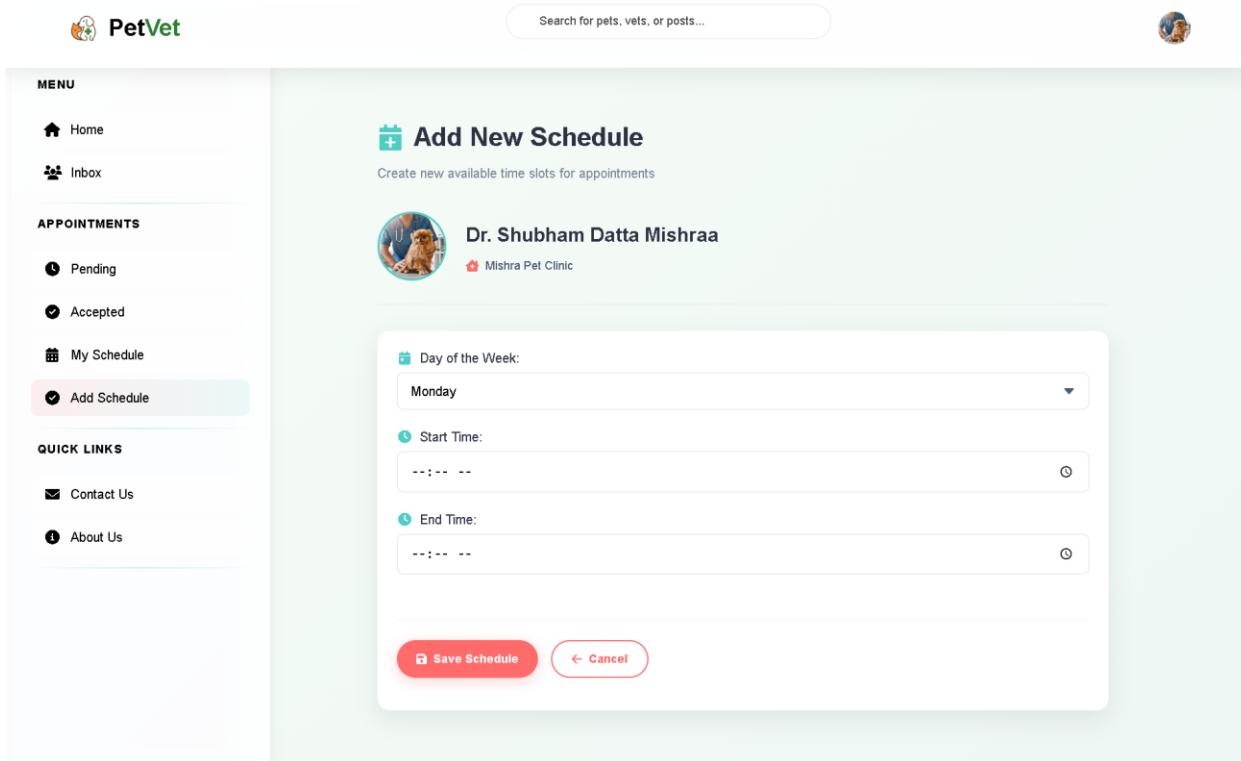


Figure 239: Adding a new time slot

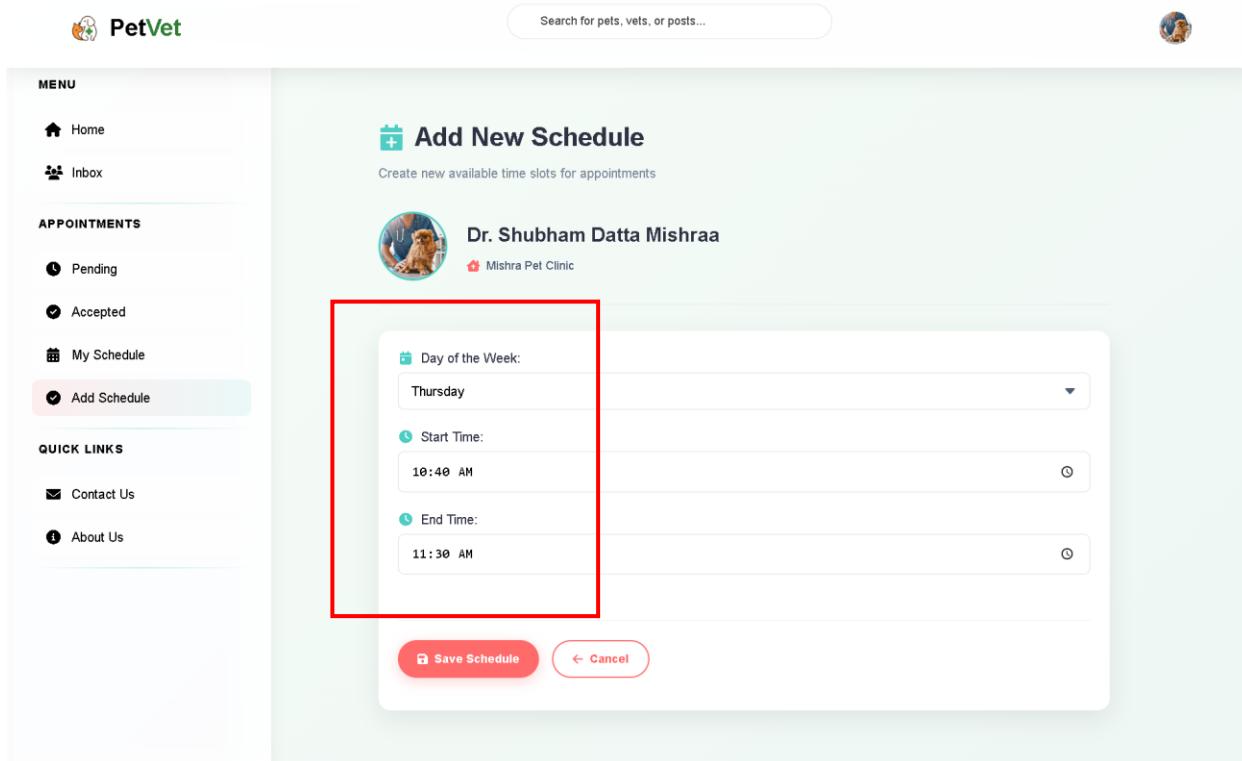


Figure 240: Creating a time slot which overlaps with the old time slot

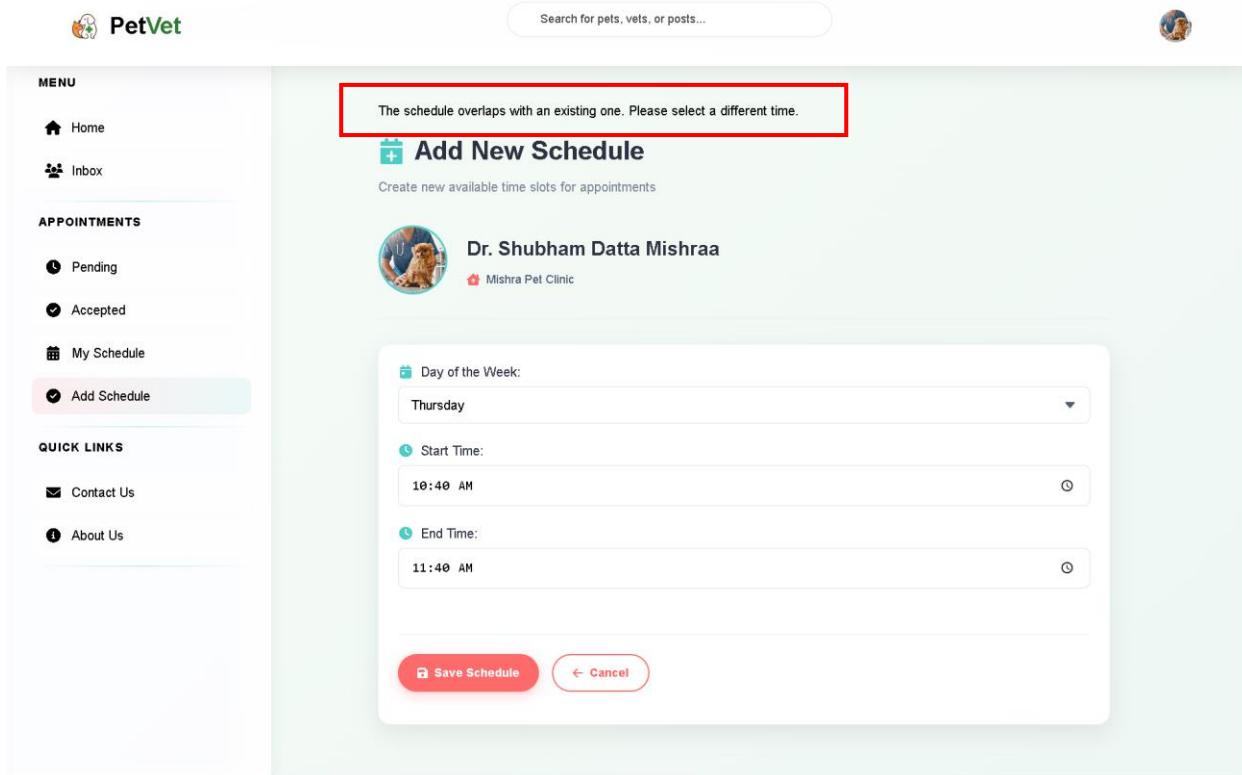


Figure 241: Overlapping could not be done

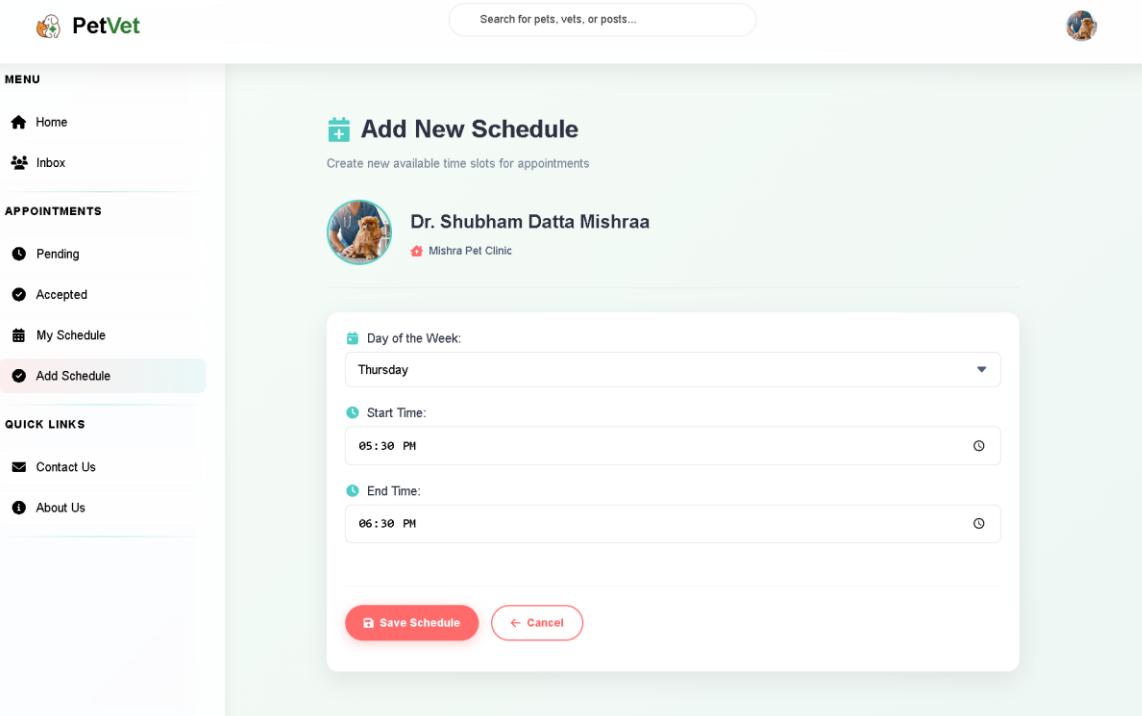


Figure 242: Adding a proper time slot

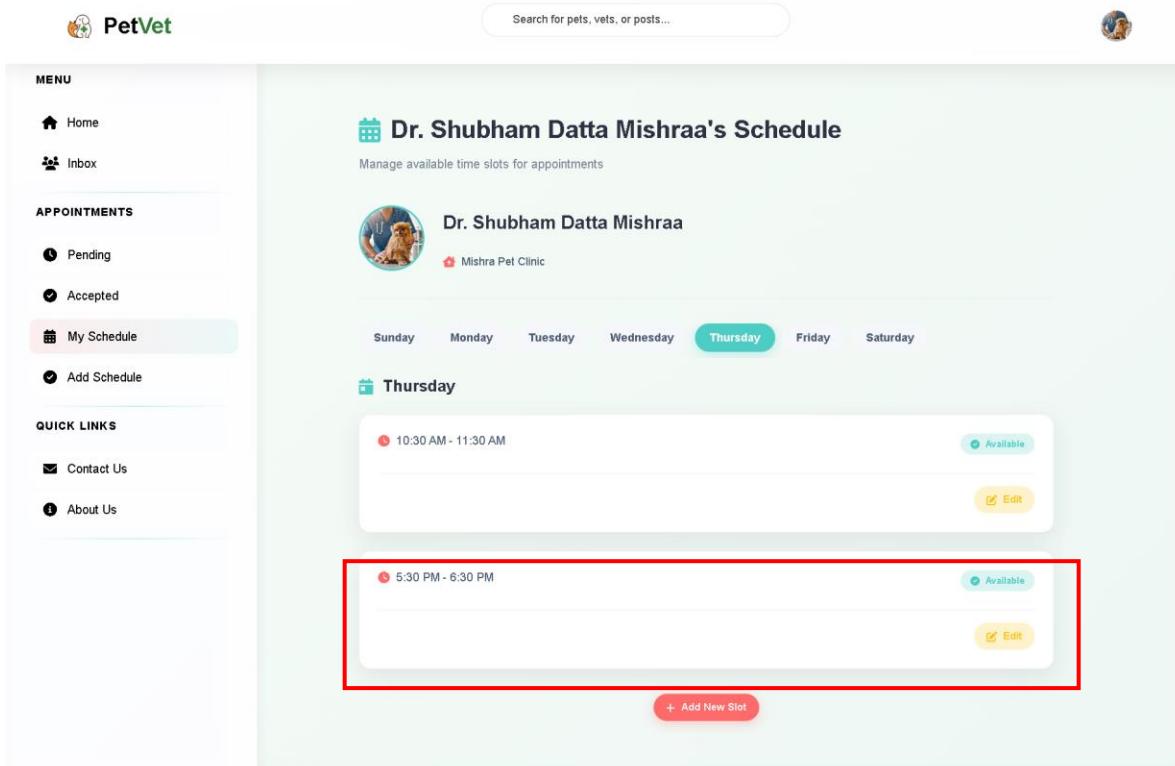


Figure 243: Time Slot has been added

Now updating a time slot and overlapping the times between them.

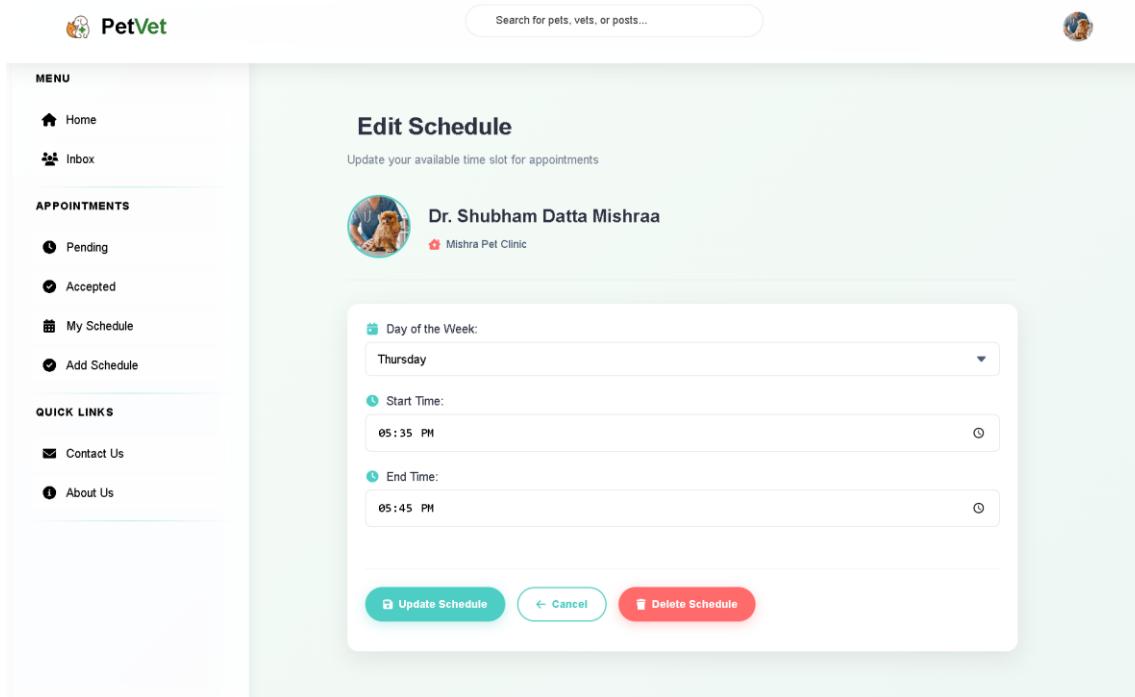


Figure 244: Updating a time slot so it overlaps with another

Reverts the time slot to its original time as well.

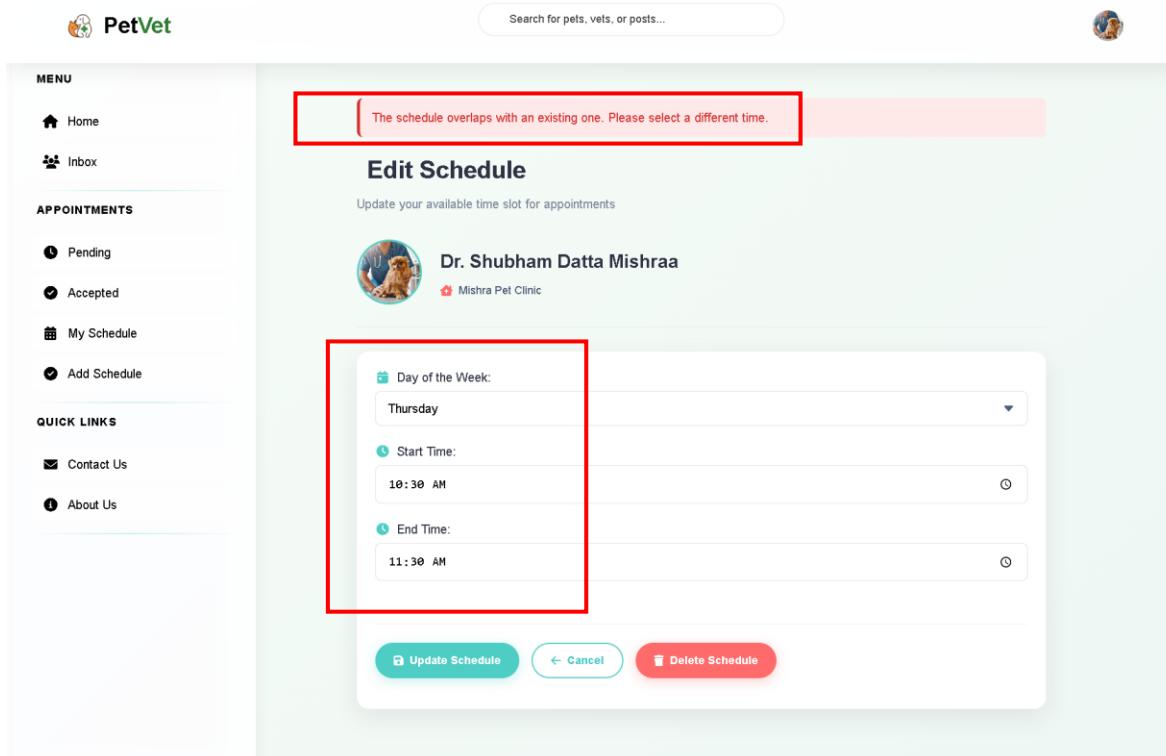


Figure 245: Updating failed as overlapping couldn't be done

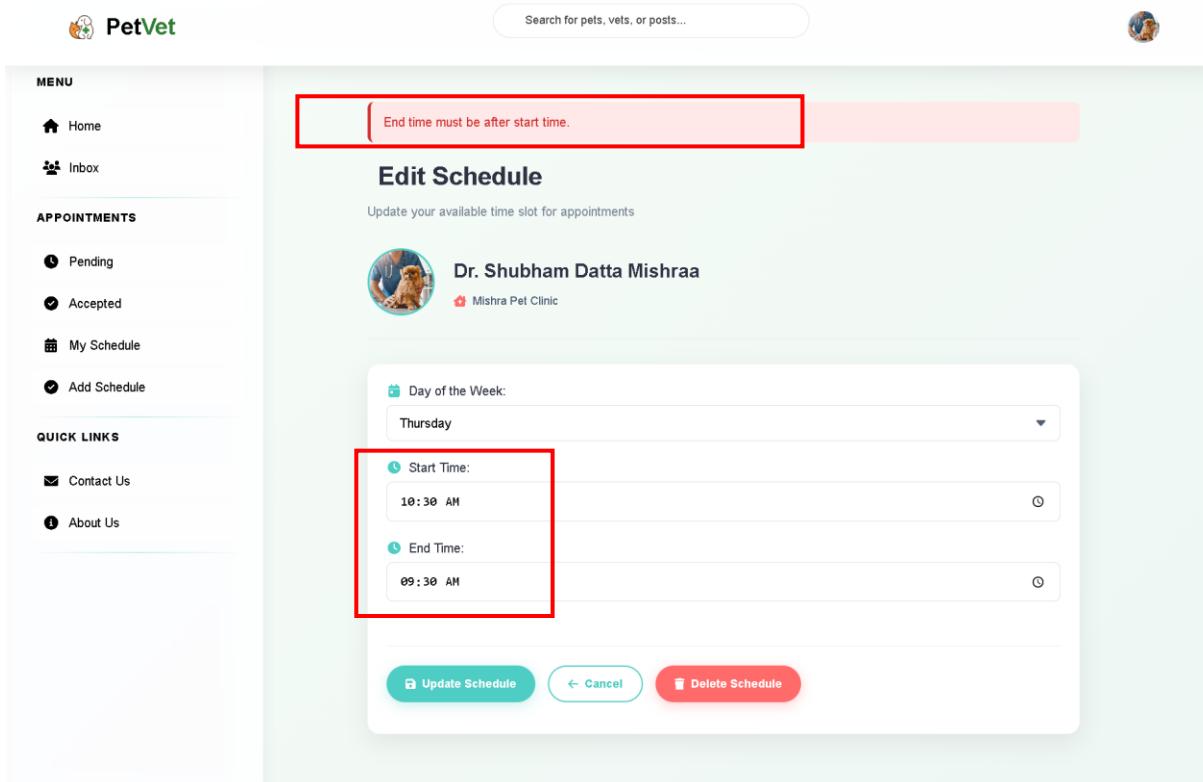


Figure 246: Making the Start time and End time invalid

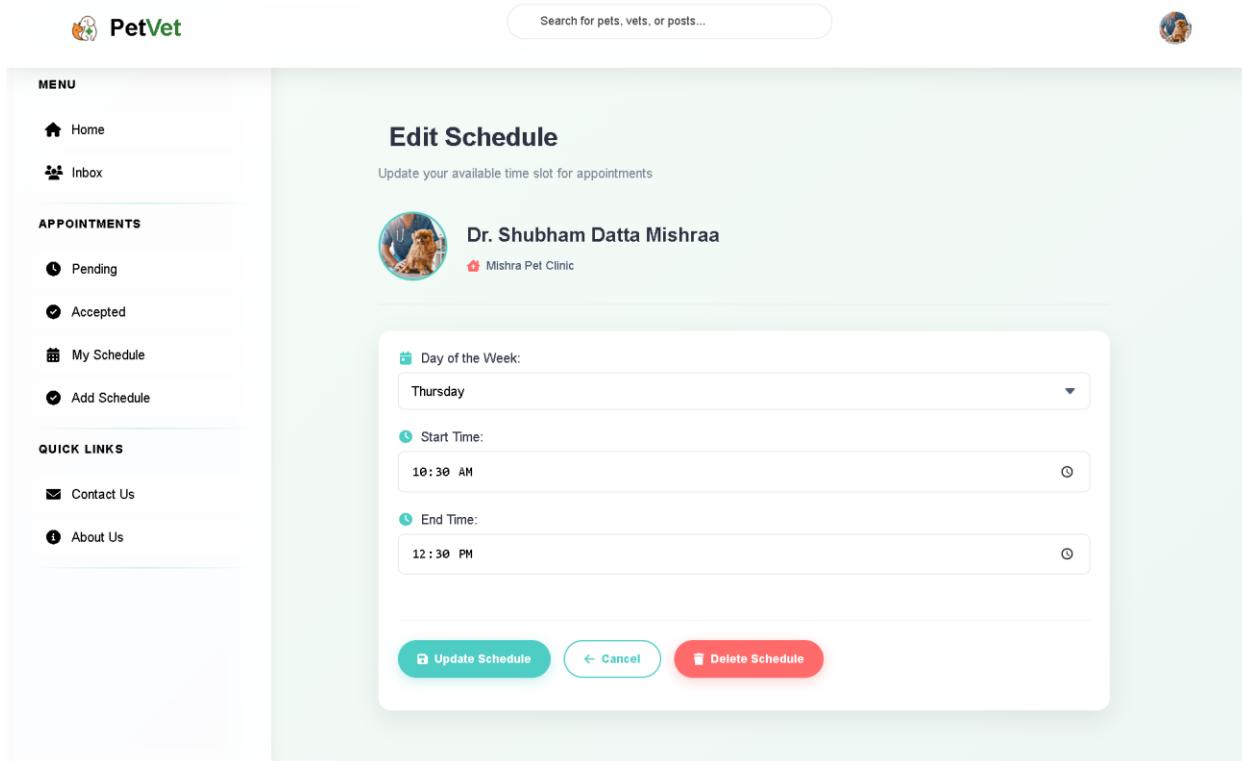


Figure 247: Updating the time slot with a valid time

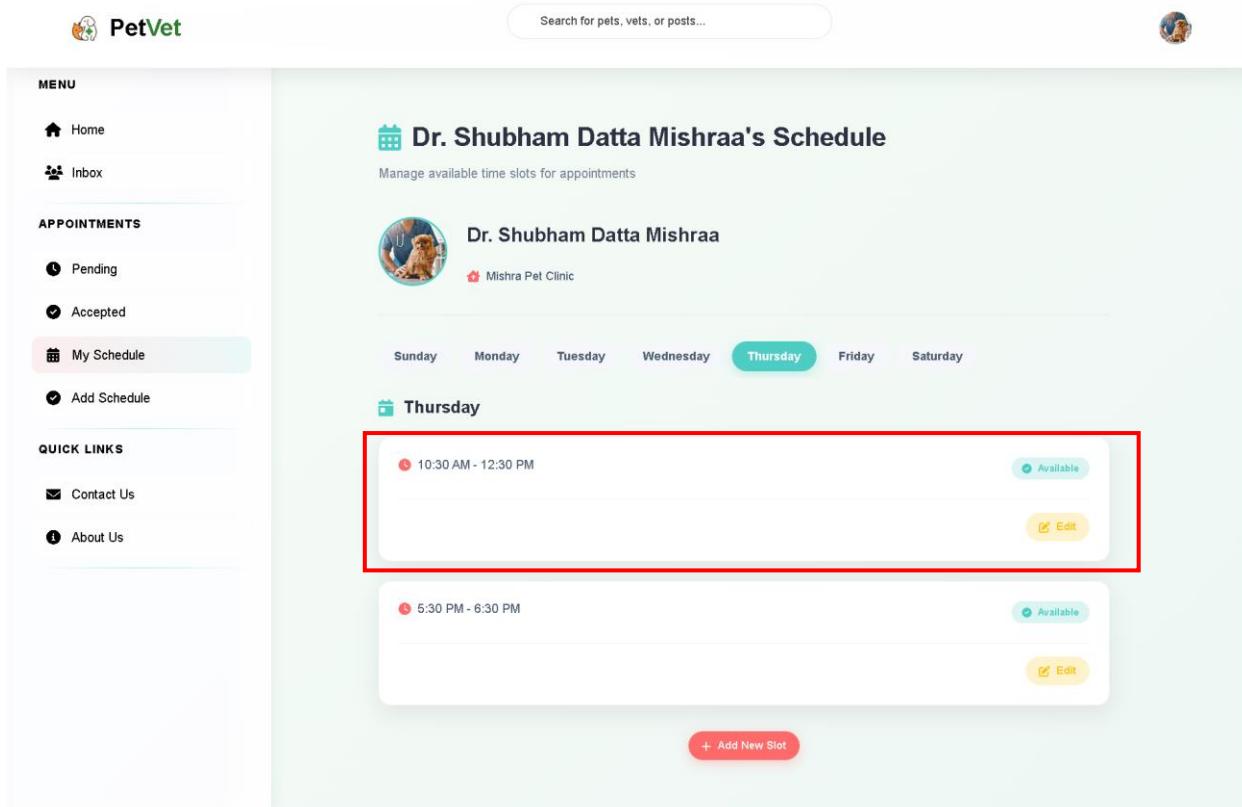


Figure 248: Updated the time Slot

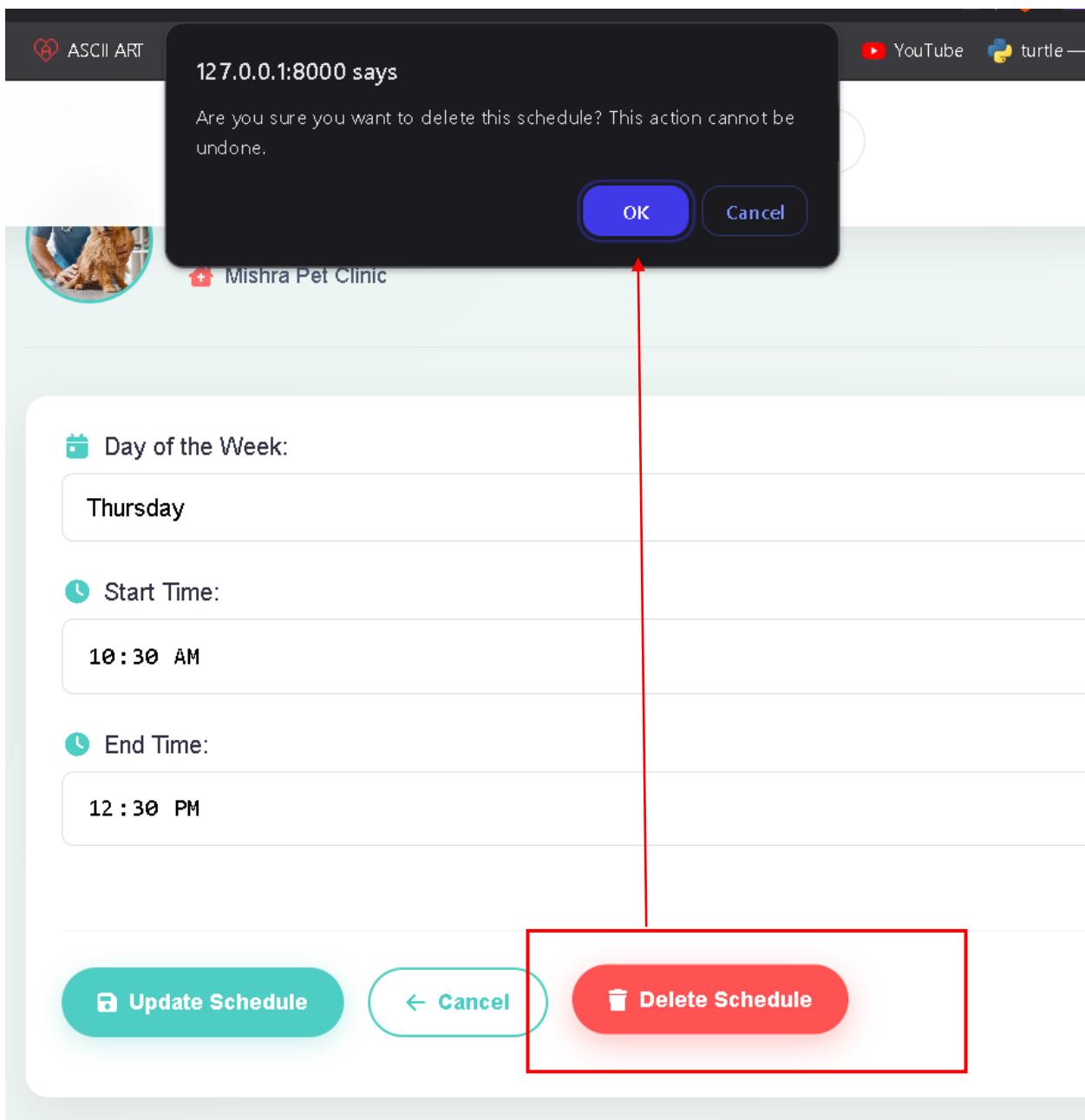


Figure 249: Deleting the Time slot

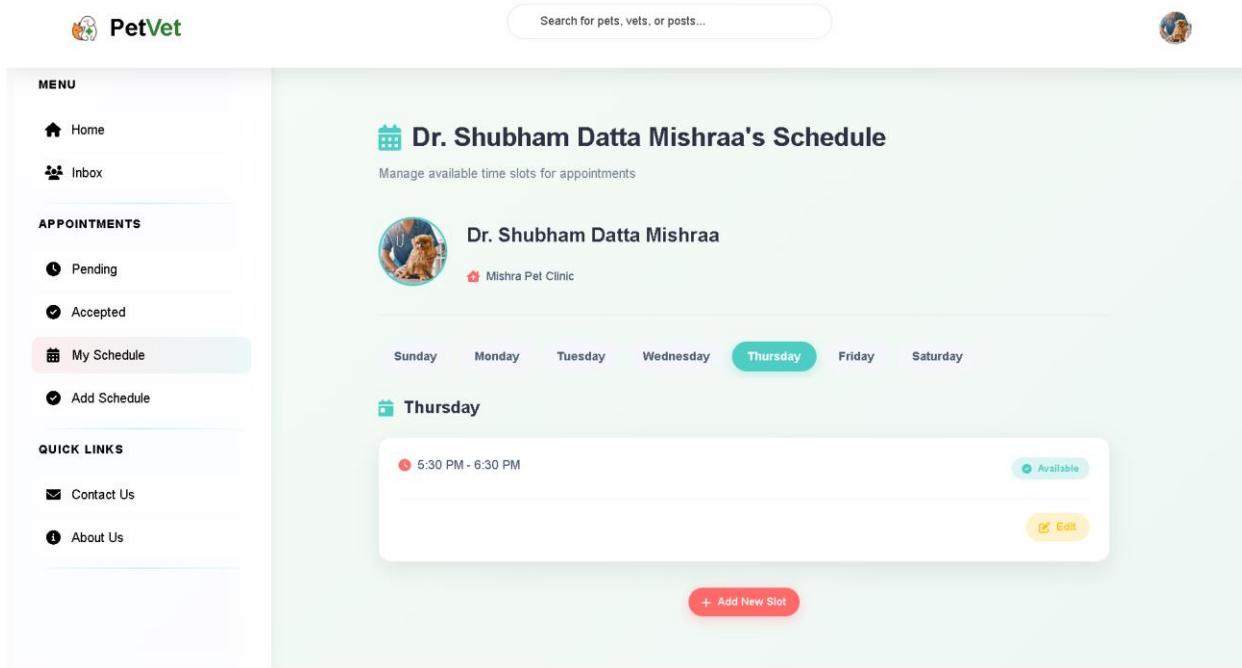


Figure 250: Successfully deleted the time slot

4.3.6 System Test: Appt3 Book Appointment via Khalti Flow

Test no.	Appt3
Objective	To verify that the complete appointment booking flow — from pet owner to vet approval and completion — works correctly including payment integration
Testcase	<ol style="list-style-type: none"> 1. Successful Appointment Booking with Khalti 2. Vet Acceptance Flow 3. Mark Appointment as Completed
Action	<ol style="list-style-type: none"> 1. Successful Appointment Booking with Khalti <ul style="list-style-type: none"> • Logged in as pet owner “Rebof Katwal” • Navigated to Find Vets, selected vet Shubham Datta Mishra • Clicked Book Appointment, selected available time slot • Chose a pet and provided a reason for the visit • Clicked Book Now, proceeded to Khalti Payment • Entered test Khalti credentials, received

	<p>successful payment confirmation</p> <ul style="list-style-type: none">• After payment completion, the system redirects to the Appointment Details page showing a 'Paid Pending Approval' tag while waiting for veterinarian confirmation. <p>2. Vet Acceptance Flow</p> <ul style="list-style-type: none">• Logged in as vet "Shubham Datta Mishra", checked Pending Appointments• Clicked Accept, email sent to pet owner <p>3. Mark Appointment as Completed</p> <ul style="list-style-type: none">• After accepting the appointment, redirected to Accepted Appointment list• Clicked Mark as Complete to finish the appointment process• After logging in as the pet owner, the 'Completed' status tag is now visible along with the review button.
--	---

Expected Result	<ul style="list-style-type: none"> • Appointment is booked successfully with Khalti payment • Vet can view, accept, and mark the appointment as completed
Actual Results	Appointment was booked, approved, and marked complete smoothly with proper redirection.
Conclusion	The test was successful

Table 40: System Test - Book Appointment via Khalti

PetVet

Search for pets, vets, or posts...

MENU

- Home
- Inbox

MY PETS

- Charlie (dog)
- Add Pet

APPOINTMENTS

- Find Vets
- My Appointments

QUICK LINKS

- Contact Us
- About Us

Find Veterinarians

Browse our network of trusted veterinary professionals

Sort by: All Vets Location: City or area...

Dr. Shubham Datta Mishra Verified Mishra Pet Clinic Lalitpur, Nepal ★★★★★ 0.0 (0 reviews)	Dr. Test Name Vet Test clinic Kathmandu, Nepal ★★★★★ 0.0 (0 reviews)	Dr. Avishek Gautam Where Pets Come First Clinic Lalitpur, Nepal ★★★★★ 0.0 (0 reviews)
Good with small animals as well		
2 years experience Specializes in Small Animal Care & Surgery		
4 years experience Specializes in Exotic Birds		
Advocate for Preventive Care and Holistic Treatments 2 years experience Specializes in None		
Book Appointment	Book Appointment	Book Appointment
View Profile	View Profile	View Profile

Figure 251: Finding Vet page

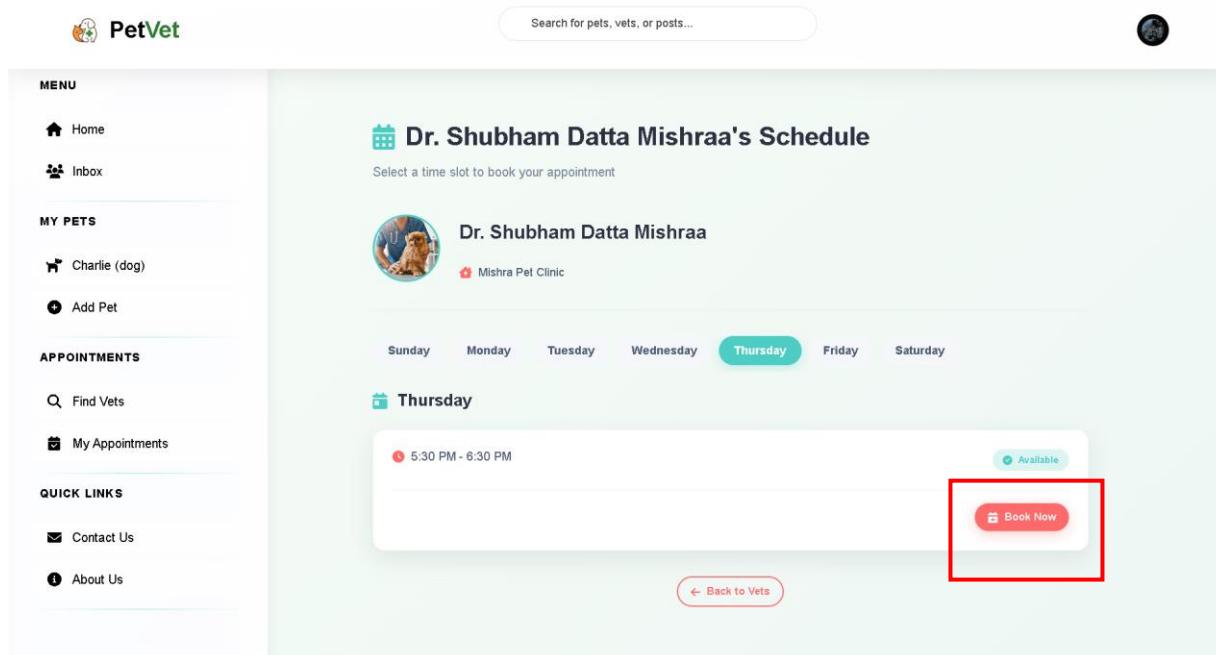


Figure 252: Select a time slot

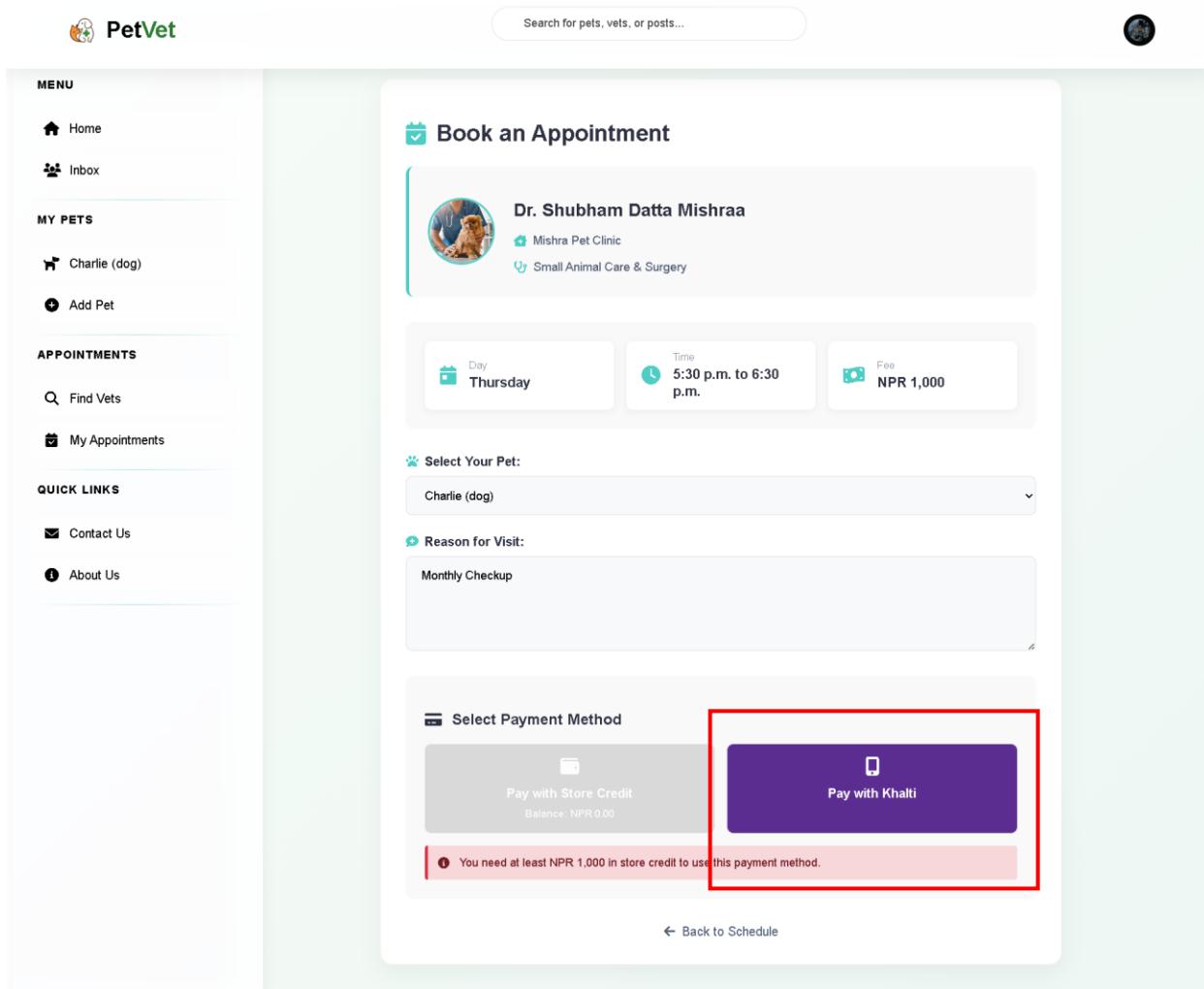


Figure 253: Click pay with khalti option

Payment Details

This payment will expire on Apr 25, 2025 18:16 PM

Billed To:
adminRebof
9800000001 , rebofkatwal7@gmail.com

TEST khadka

Amount Summary:

Total Payable Amount	Rs 1000.00
----------------------	------------

Enter Khalit ID

Khalit Mobile Number: 9800000001

Khalit Password / MPIN: ****

Submit

Forgot your password? [Reset Password](#)

PAYMENT POWERED BY

Cancel Payment

Figure 254: Redirected to khalit's payment page

PetVet

Search for pets, vets, or posts...

MENU

- Home
- Inbox

MY PETS

- Charlie (dog)
- Add Pet

APPOINTMENTS

- Find Vets
- My Appointments

QUICK LINKS

- Contact Us
- About Us

Payment Successful

Your appointment payment has been processed.

Thank you for your payment!

Your payment for the appointment with Dr. Datta18 has been successfully processed.

Amount Paid:	NPR 1000
Appointment ID:	#5
Status:	Pending Vet Confirmation

The vet will now review your appointment request. You'll be notified once they accept or decline it.

View Appointment Details

Figure 255: Payment Successful

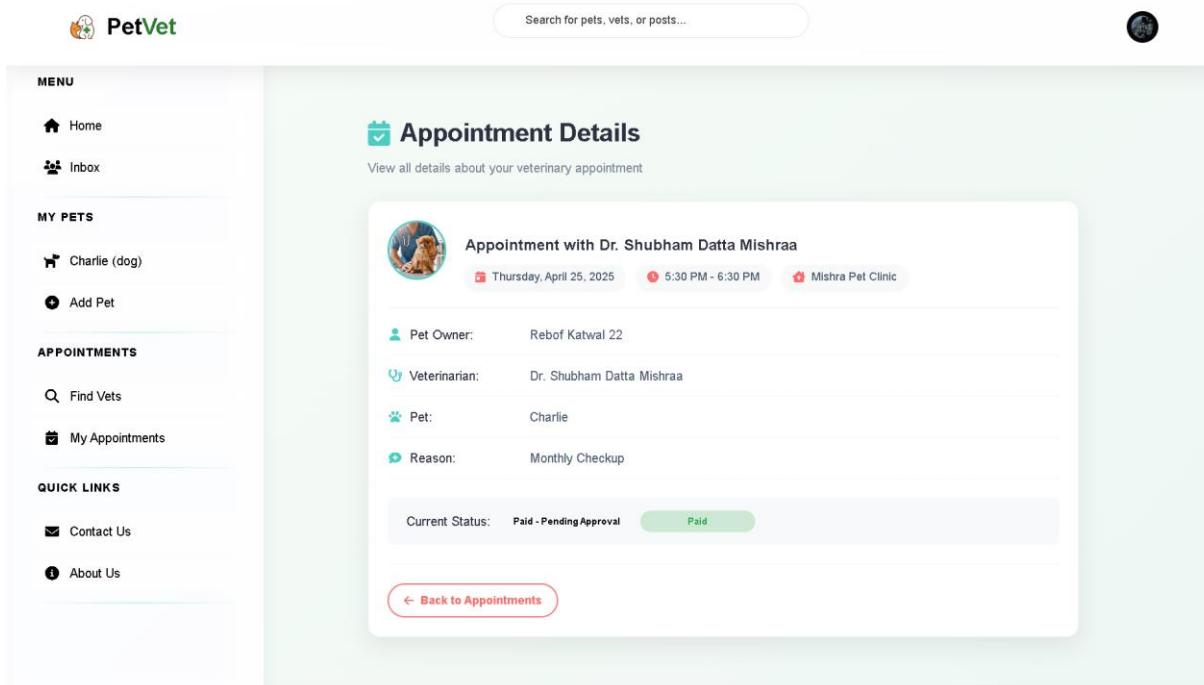


Figure 256: Appointment Details page after payment

Logging as a Vet, and navigating to Pending Appointment Page.

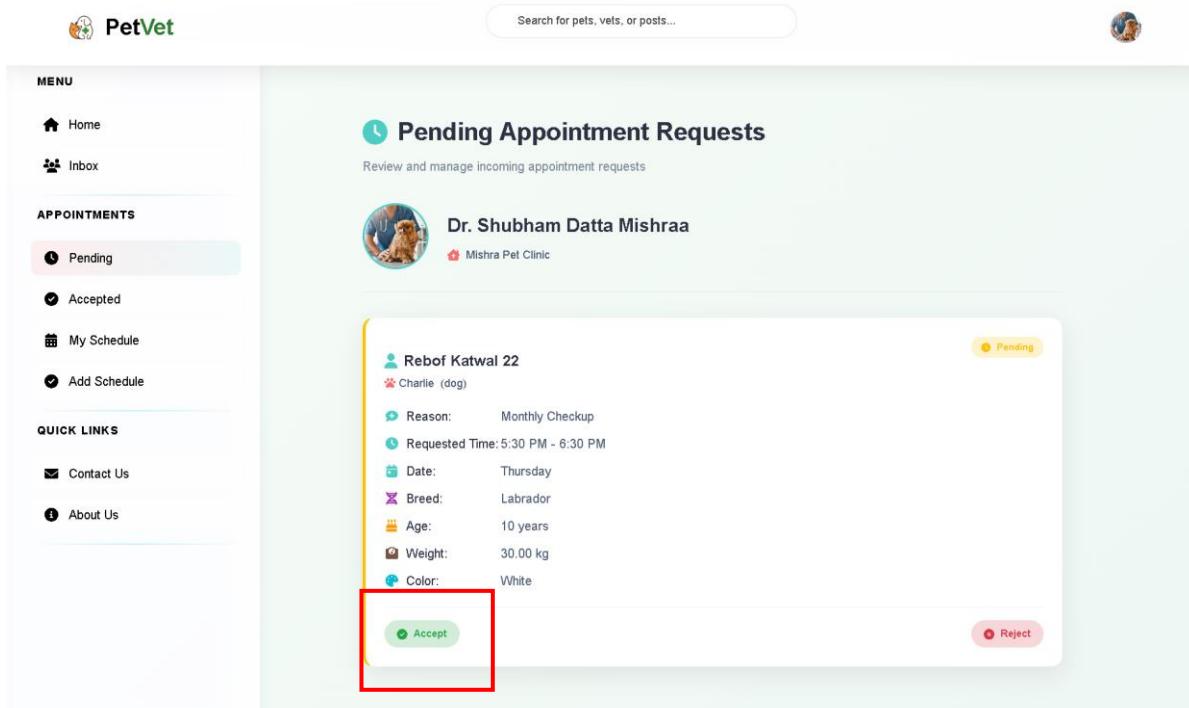


Figure 257: Accepting the appointment request

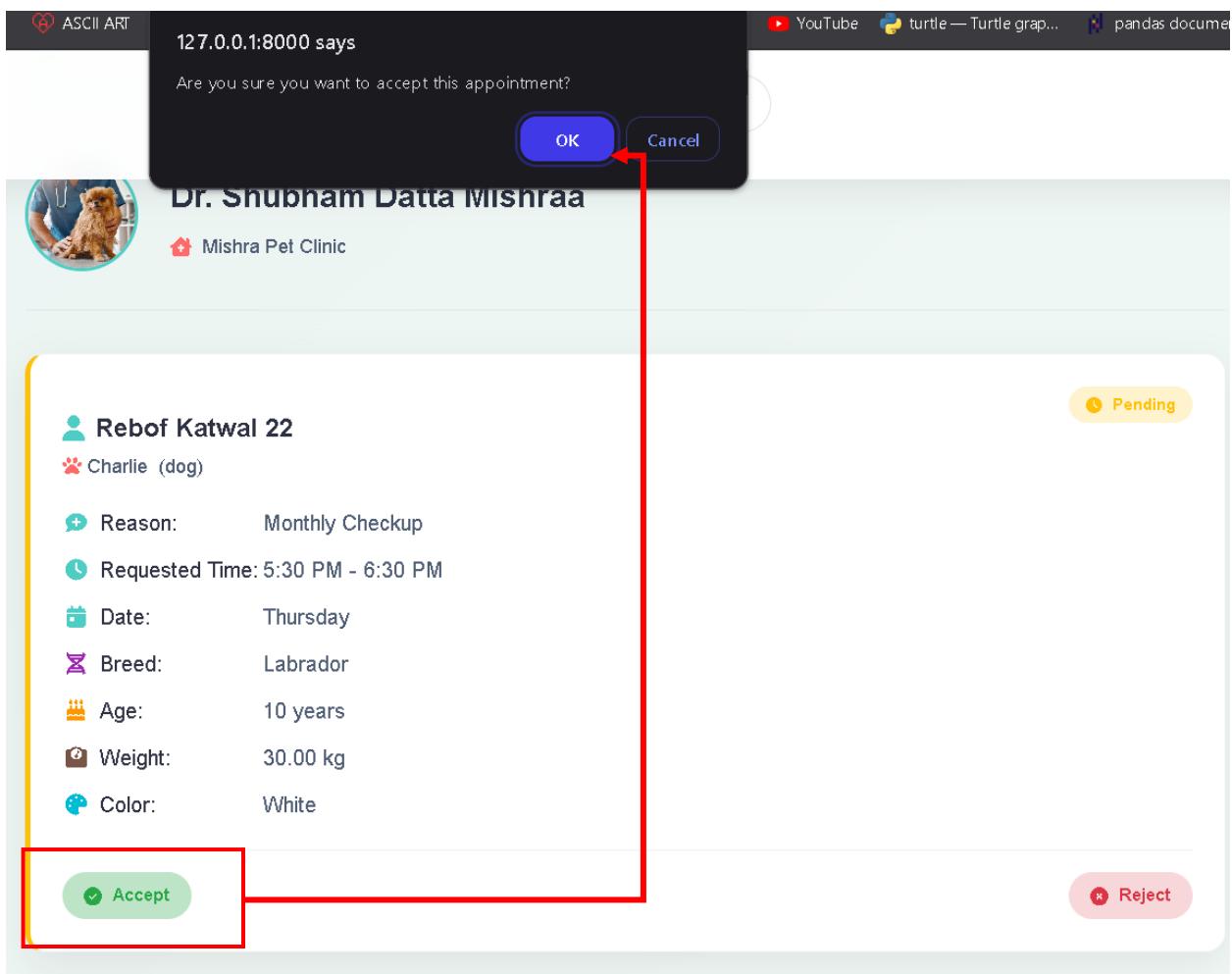


Figure 258: Alert box after accepting

Accepted Appointments

Manage your upcoming veterinary appointments

Dr. Shubham Datta Mishraa
Mishra Pet Clinic

Rebof Katwal 22

Charlie ()

Reason: Monthly Checkup
Scheduled Time: 5:30 PM - 6:30 PM
Date: Thursday, April 25, 2025
Breed: Labrador
Age: 10 years
Weight: 30.00 kg
Color: White

Mark as Completed

Figure 259: Redirected to accepted appointments page

Logging back as the pet owner, we can see the status change.

Appointment Details

View all details about your veterinary appointment

Appointment with Dr. Shubham Datta Mishraa

Thursday, April 25, 2025 | 5:30 PM - 6:30 PM | Mishra Pet Clinic

Pet Owner: Rebof Katwal 22
Veterinarian: Dr. Shubham Datta Mishraa
Pet: Charlie
Reason: Monthly Checkup

Current Status: **Confirmed** Paid

[Back to Appointments](#) [Cancel Appointment](#)

Figure 260: Appointment status change after accepting

Assuming the appointment has been completed, the vet marks it as completed.

The screenshot shows the PetVet application interface. On the left, there's a sidebar with a logo, a search bar, and sections for MENU (Home, Inbox), APPOINTMENTS (Pending, Accepted, My Schedule, Add Schedule), and QUICK LINKS (Contact Us, About Us). The main area is titled "Accepted Appointments" and shows "Manage your upcoming veterinary appointments". It lists an appointment for "Dr. Shubham Datta Mishraa" at "Mishra Pet Clinic". Below this, a detailed view of an appointment for "Rebof Katwal 22" is shown, with a red arrow pointing to the "Mark as Completed" button. The appointment details are:

Attribute	Value
Reason	Monthly Checkup
Scheduled Time	5:30 PM - 6:30 PM
Date	Thursday, April 25, 2025
Breed	Labrador
Age	10 years
Weight	30.00 kg
Color	White

A green "Confirmed" button is visible in the top right corner of the appointment card.

Figure 261: Mark as complete checkbox clicked

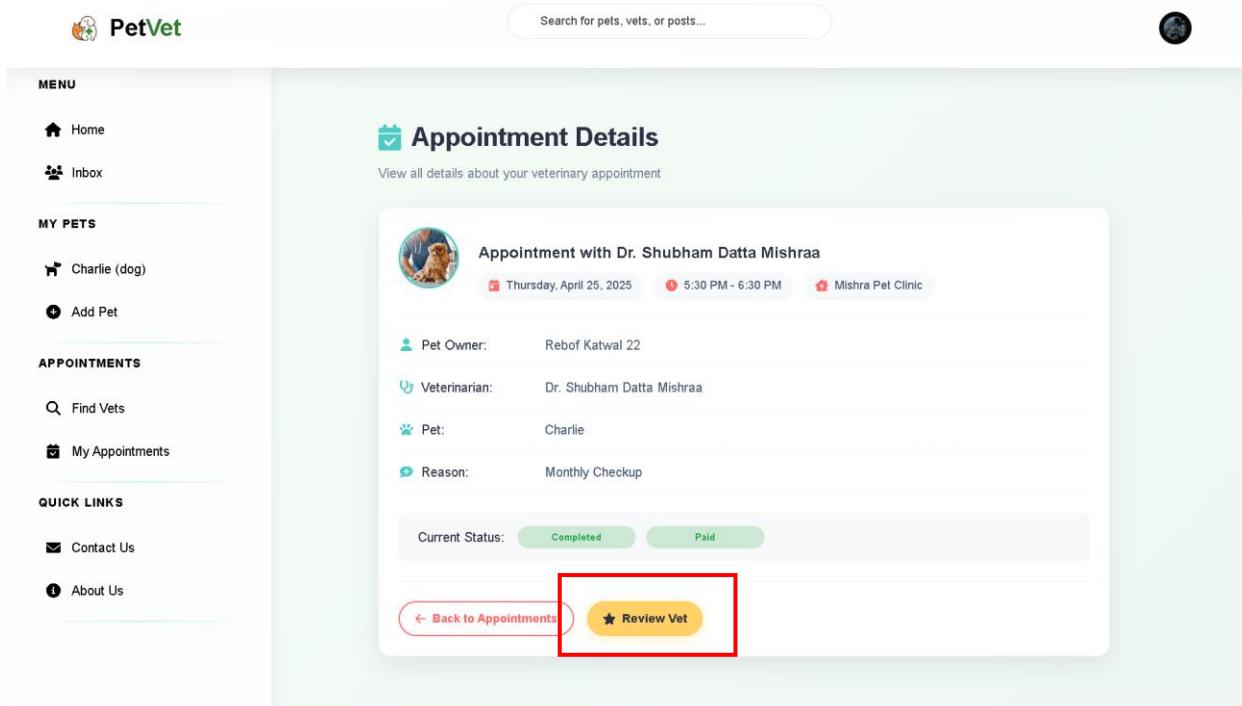


Figure 262: After mark as complete, review button appears

4.3.7 System Test: Appt4 Review Functionality

Test no.	Appt4
Objective	To verify that a pet owner can submit a review only after successfully completing an appointment and that the review updates the vet's profile and visibility across the platform.
Testcase	<ol style="list-style-type: none"> 1. Validation for Empty Review Field 2. Successful Review Submission 3. Rating Update in Vet
Action	<ol style="list-style-type: none"> 1. Validation for Review <ul style="list-style-type: none"> • Reached the Review Page after completing an appointment with vet Shubham Datta Mishra • Tried submitting the review form with an empty comment, prompted to fill the review field 2. Successful Review Submission <ul style="list-style-type: none"> • Entered a valid review message which was "Awesome vet!" with 5-star rating and submitted.

	<ul style="list-style-type: none"> • Redirected to Appointment Details page; review button now disappears <p>3. Rating and Review Update in Vet</p> <ul style="list-style-type: none"> • Checked Find Vets list – updated rating displayed for the vet • Opened vet's profile – confirmed the submitted review is visible there
Expected Result	<ul style="list-style-type: none"> • Review field validation prevents empty submission • Review submission redirects user and hides review button • Vet's rating updates in listings and review appears in profile
Actual Results	Review was submitted successfully, button was hidden after submission, and updated review appeared in vet profile and listing.
Conclusion	The test was successful

Table 41: System Test - Review

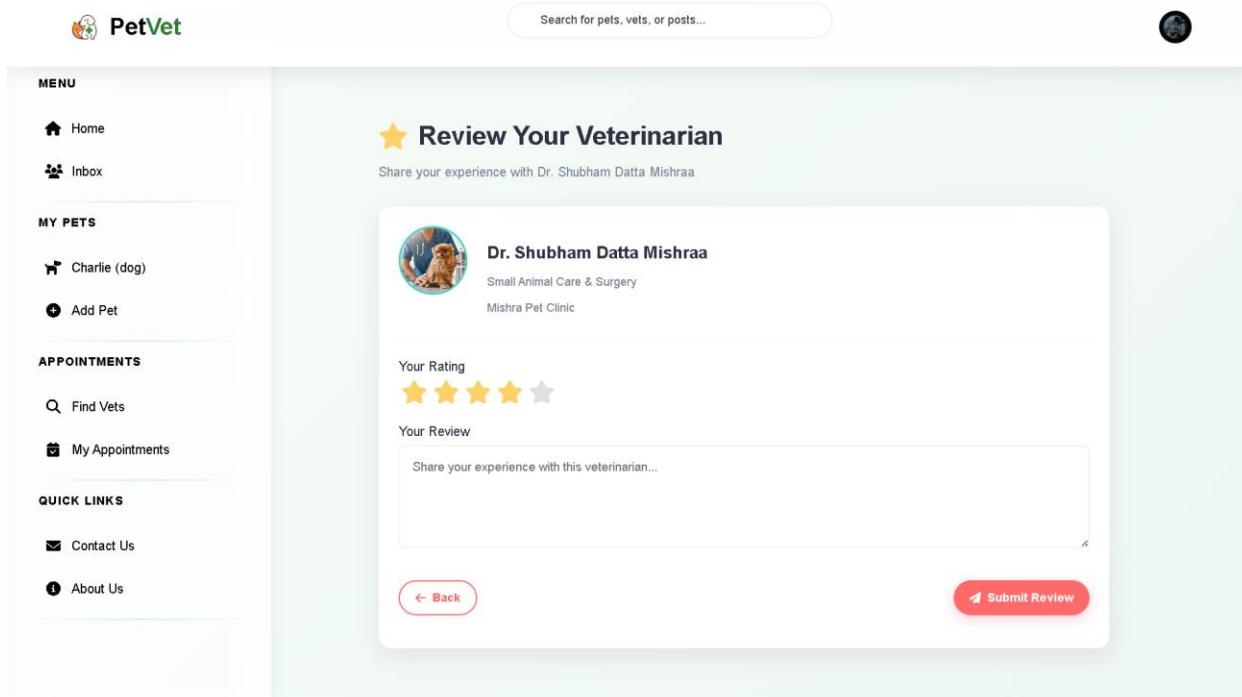


Figure 263: Redirected to review page after clicking the review button

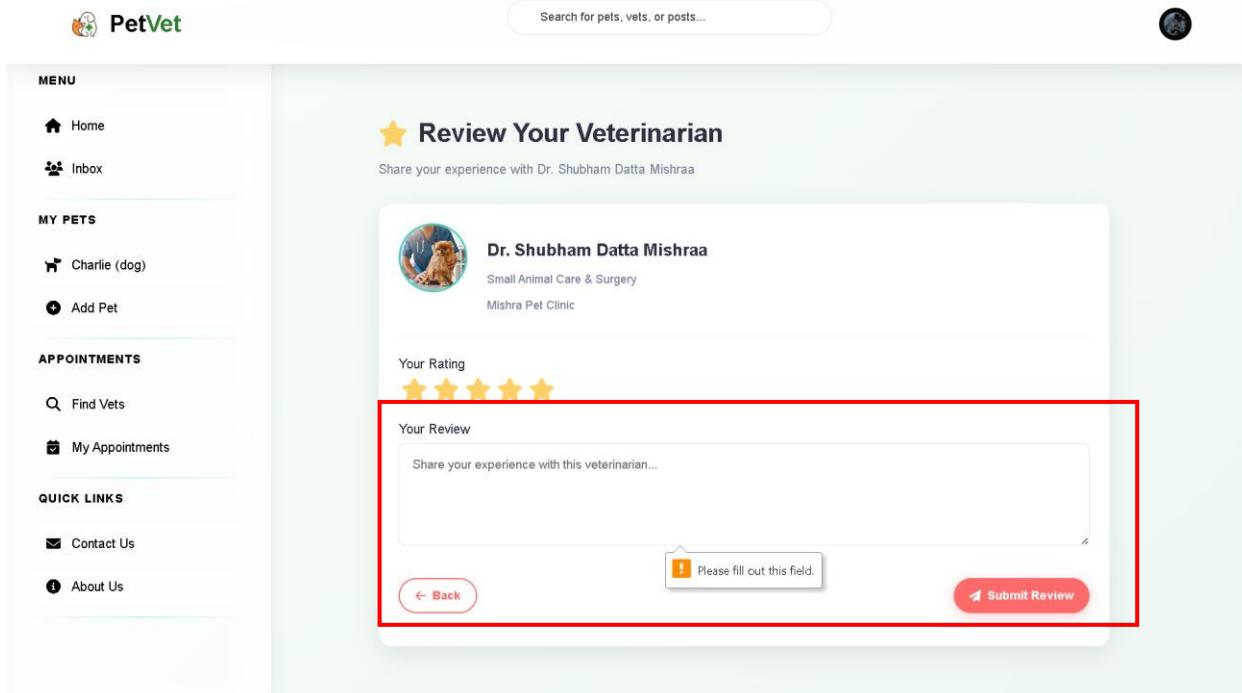


Figure 264: Leaving the review field empty

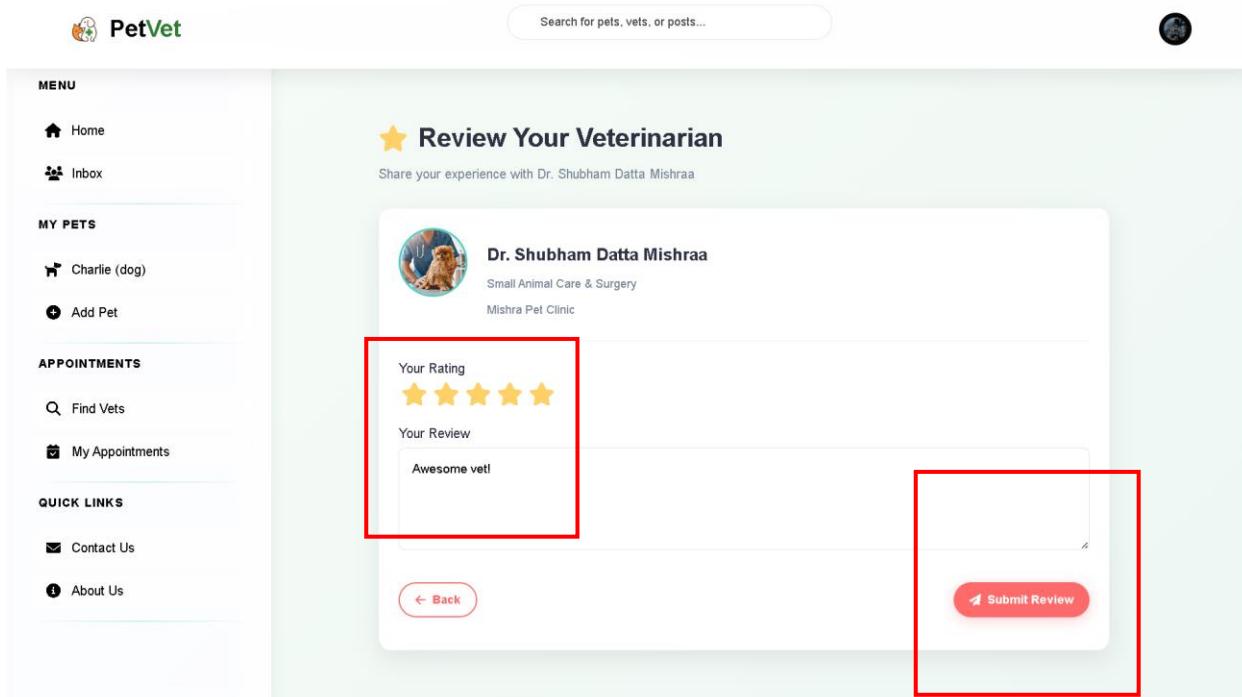


Figure 265: Entering a proper review and submitting

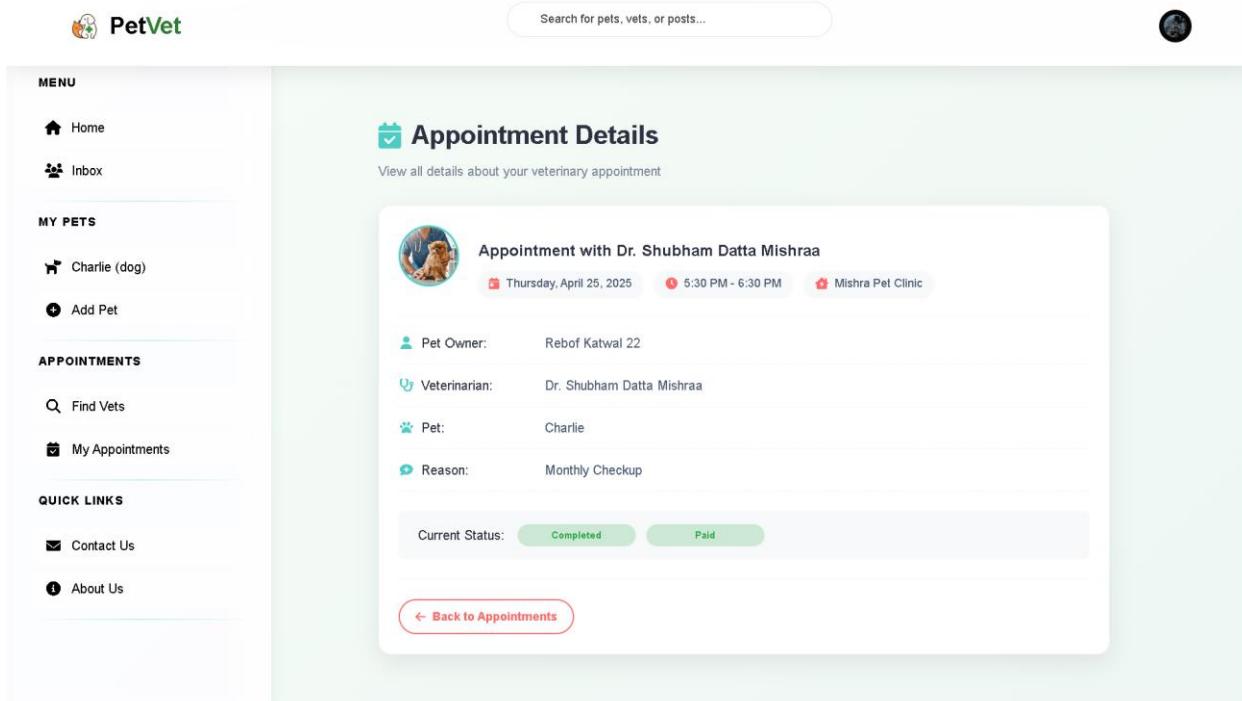


Figure 266: Review button disappears after reviewing

MENU

- Home
- Inbox

MY PETS

- Charlie (dog)
- Add Pet

APPOINTMENTS

- Find Vets
- My Appointments

QUICK LINKS

- Contact Us
- About Us

Find Veterinarians

Browse our network of trusted veterinary professionals

Sort by: All Vets Location: City or area...

Veterinarian Profile	Details	Reviews
Dr. Shubham Datta Mishra Verified Mishra Pet Clinic Lalitpur, Nepal	Passionate about pet health & well-being 4 years experience Specializes in Small Animal Care & Surgery	5.0 (1 reviews)
Dr. Test Name Vet Verified Test clinic Kathmandu, Nepal	Good with small animals as well 2 years experience Specializes in Exotic Birds	0.0 (0 reviews)
Dr. Avishek Gautam Verified Where Pets Come First Clinic Lalitpur, Nepal	Advocate for Preventive Care and Holistic Treatments 2 years experience Specializes in None	0.0 (0 reviews)

Book Appointment **View Profile**

Figure 267: Rating updated after review

Reviews

Rebof Katwal 22
Apr 25, 2025

Awesome vet!

★★★★★

Figure 268: Review added to the Vet's profile

4.3.8 System Test: Appt5 Decline Appointment and Refunding

Test no.	Appt5
Objective	To verify the booking flow with Khalti payment when a vet declines the appointment, ensuring the refund is correctly issued to the pet owner's store credit.
Testcase	1. Declining Appointment by Vet and Verification of Refund Issuance
Action	<p>1. Declining Appointment by Vet</p> <ul style="list-style-type: none"> • Logged in as Rebof Katwal (pet owner) • Navigated to Find Vets, selected Shubham Datta Mishra, chose a valid time slot • Selected a pet and provided reason: "Weekly checkup" • Proceeded to Khalti payment, entered valid test credentials • Payment was successful and redirected to appointment detail page (pending approval)

	<ul style="list-style-type: none"> • Checked pet owner profile before vet's response – store credit was 0 • Opened pending appointments, clicked Decline, alert box prompted confirmation • Confirmed decline • Logged back in as pet owner, checked profile – store credit updated with 1000 refund
Expected Result	<ul style="list-style-type: none"> • Appointment payment goes through successfully • Decline by vet triggers proper warning • Refund of full amount is credited as store credit to pet owner
Actual Results	Payment was made successfully. After the vet declined, a refund of 1000 was added to the pet owner's store credit balance.
Conclusion	The test was successful

Table 42: System Test - Decline Appointment

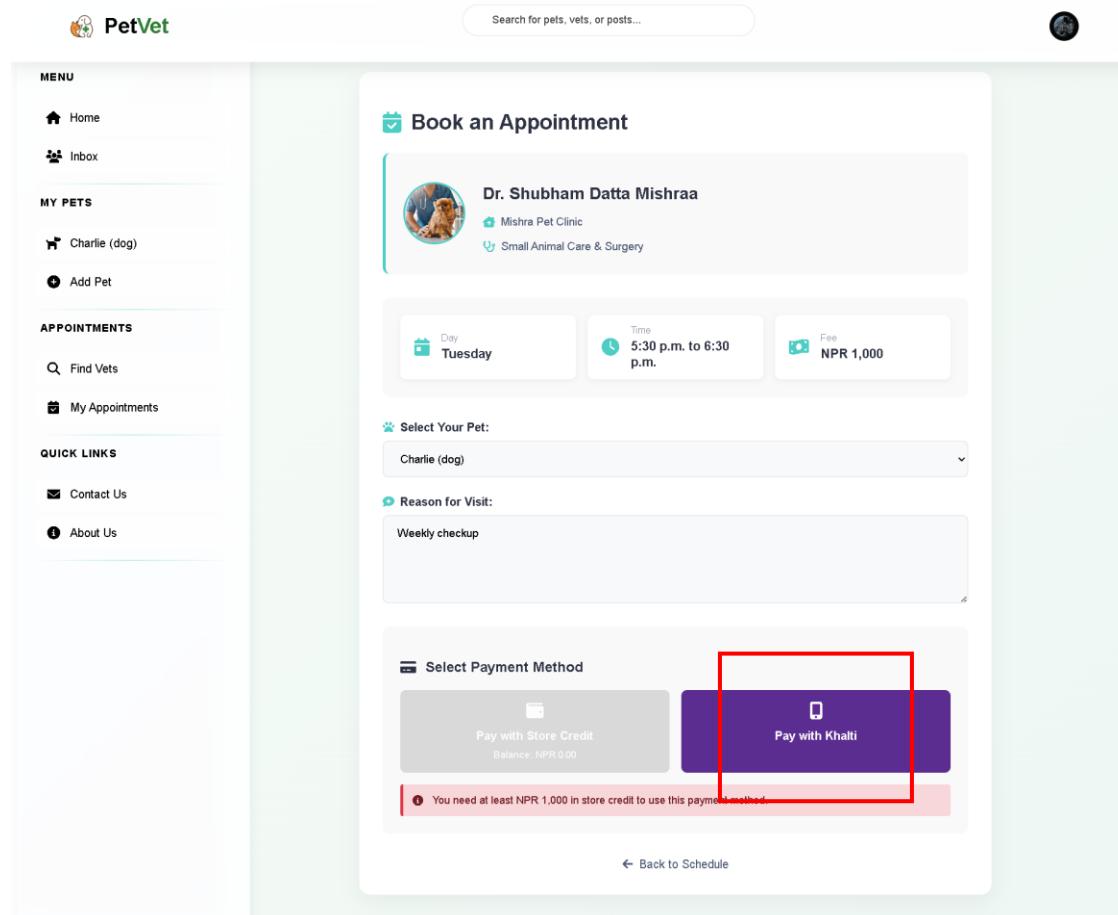


Figure 269: Payment done through Khalti

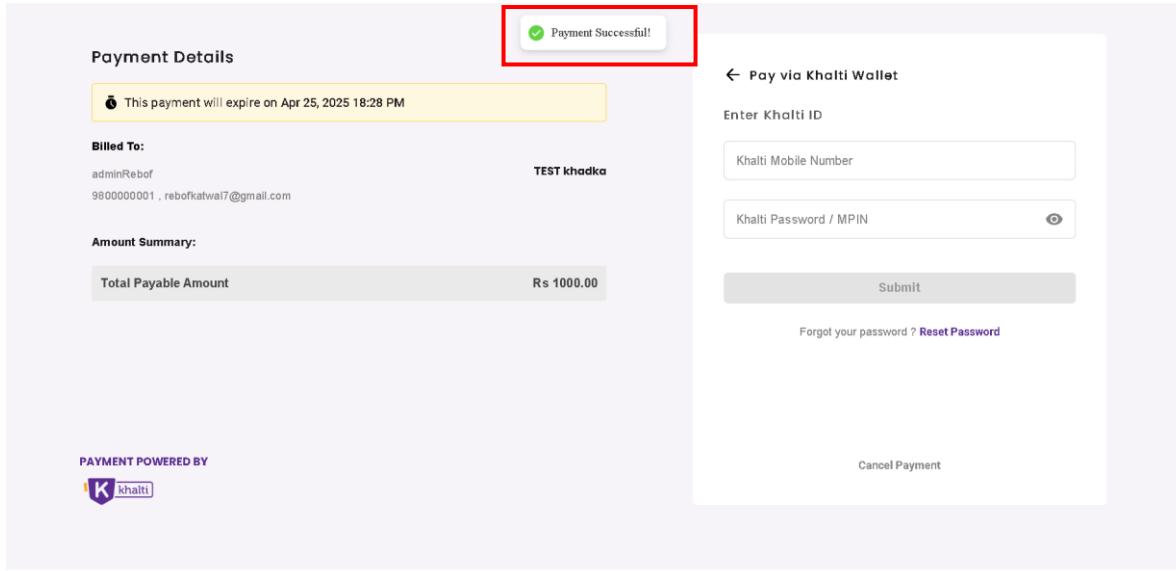


Figure 270: After successful payment, shows a popup and starts redirecting to payment success page

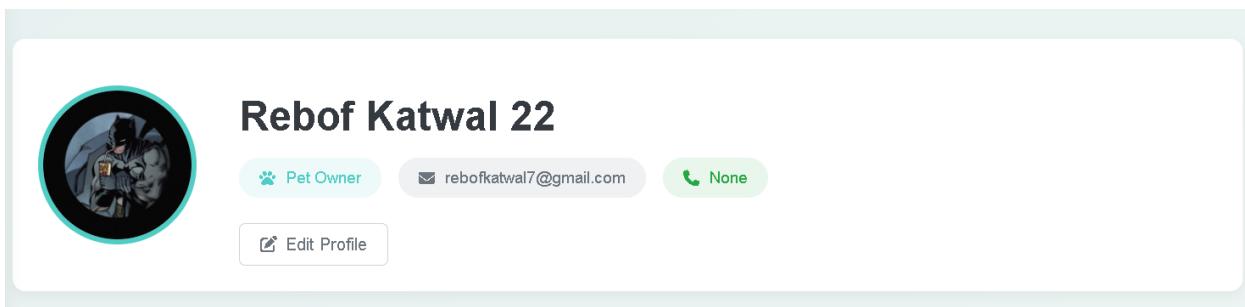


Figure 271: User profile before decline

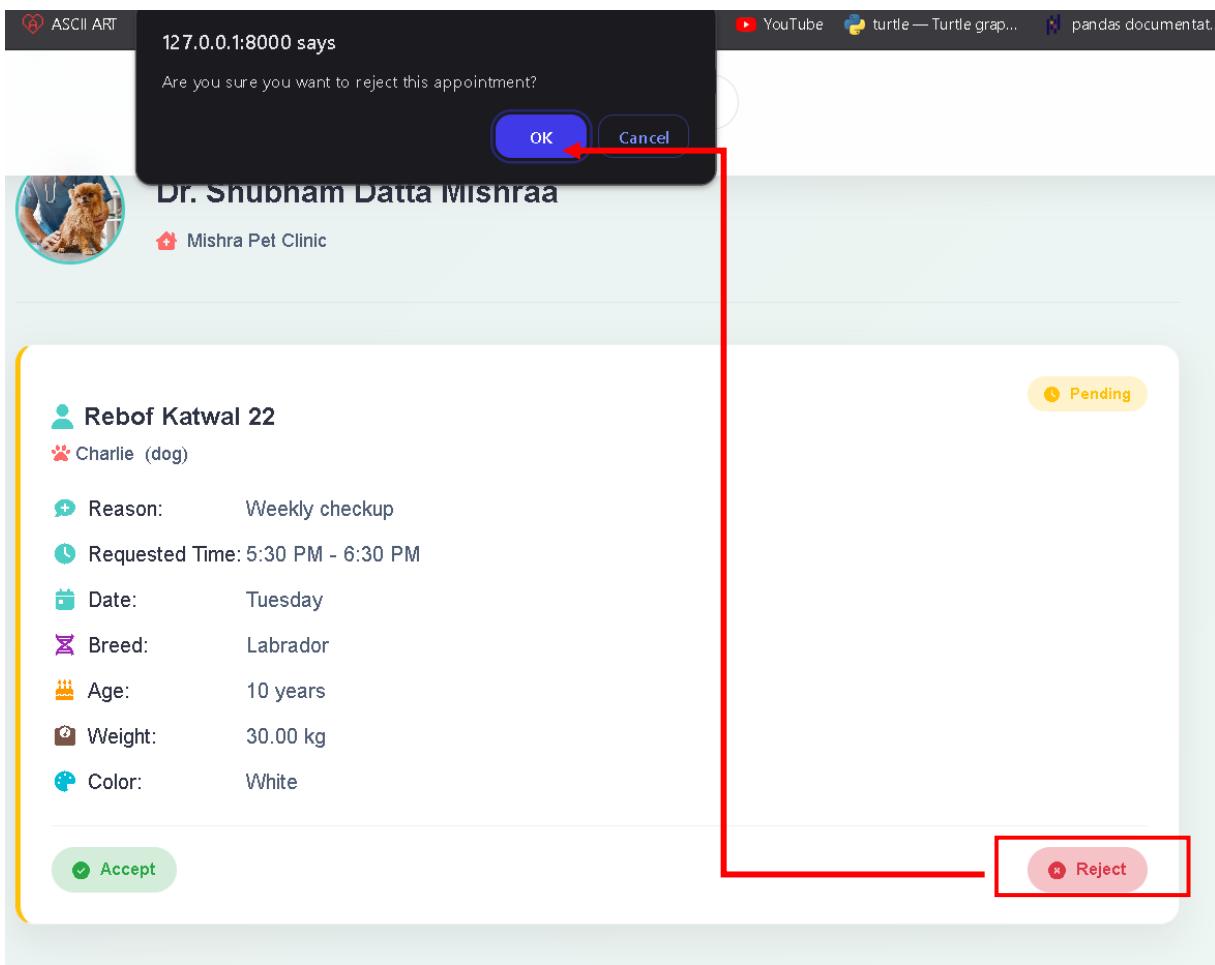


Figure 272: Decline with an alert box

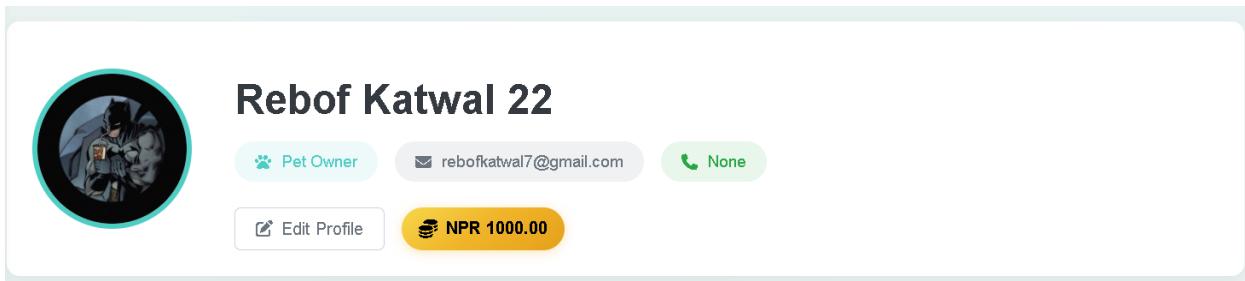


Figure 273: After decline, the user is refunded

4.3.9 System Test: Appt6 Book Appointment via Store Credit and Cancellation

Test no.	Appt6
Objective	To verify that appointments can be paid using store credit and that cancellation by the pet owner results in an 80% refund of the credited amount.
Testcase	<ol style="list-style-type: none"> 1. Booking Appointment Using Store Credit 2. Cancelling Appointment as Pet Owner and Verification of Partial Refund (80%)
Action	<ol style="list-style-type: none"> 1. Booking Appointment Using Store Credit <ul style="list-style-type: none"> • Logged in as Rebof Katwal (pet owner) • Navigated to Find Vets, selected Shubham Datta Mishra, chose a valid time slot • Selected a pet and provided reason: “Fever” • Since previous test provided 1000 store credit, “Pay with Store Credit” option was available and selected

	<ul style="list-style-type: none"> • Successfully redirected to appointment details page <p>2. Cancelling Appointment as Pet Owner and Verification of Partial Refund (80%)</p> <ul style="list-style-type: none"> • Checked pet owner profile — store credit was now 0 • Vet received request in their pending appointment list • Pet owner clicked Cancel Appointment • Alert box appeared with warning • Proceeded with cancellation • Refund of 800 (80% of 1000) was credited back to pet owner's store credit
Expected Result	<ul style="list-style-type: none"> • Store credit payment works correctly • Upon cancellation, user is properly alerted about partial refund • 80% of the credited amount is refunded as store credit

Actual Results	The appointment was booked using store credit. On cancellation, an alert was shown and a refund of 800 was correctly credited back to the pet owner's store credit.
Conclusion	The test was successful

Table 43: System Test - Book Appointment via Store Credit and Cancellation

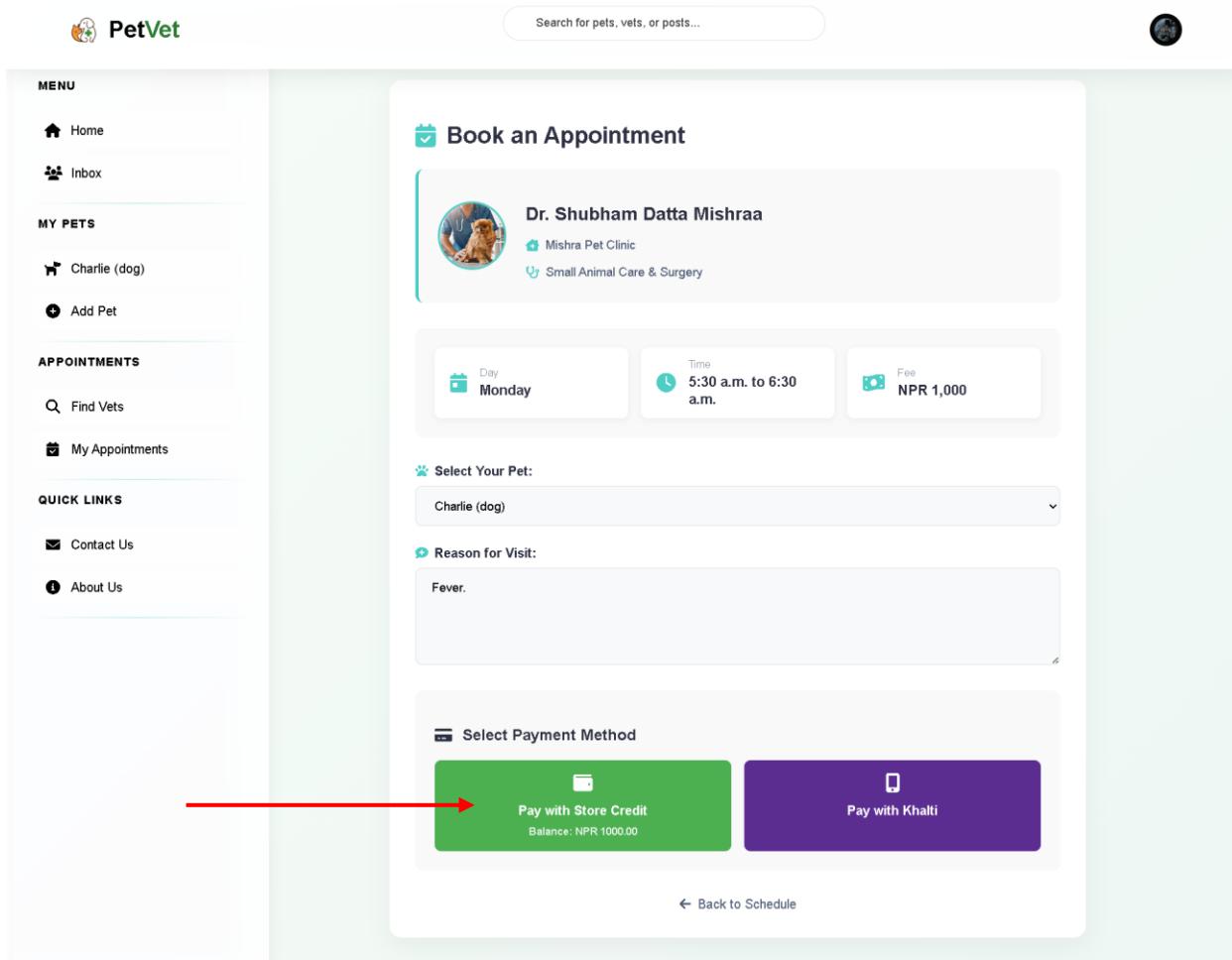


Figure 274: Pay with Store Credit option clicked

Appointment Details

View all details about your veterinary appointment

Appointment with Dr. Shubham Datta Mishraa

Monday, April 25, 2025 | 5:30 AM - 6:30 AM | Mishra Pet Clinic

Pet Owner: Rebof Katwal 22

Veterinarian: Dr. Shubham Datta Mishraa

Pet: Charlie

Reason: Fever.

Current Status: Paid - Pending Approval

Paid

Figure 275: Successful payment redirects to appointment details page

In previous test (4.3.8 System Test: Appt5 Decline Appointment and Refunding), the store credit was 1000 due to a refund. After using the store credit to book again, the balance became 0, so it is no longer showing.

Rebof Katwal 22

Pet Owner rebofkatwal7@gmail.com **None**

Edit Profile

Figure 276: After the payment via store credit, the store credit is 0

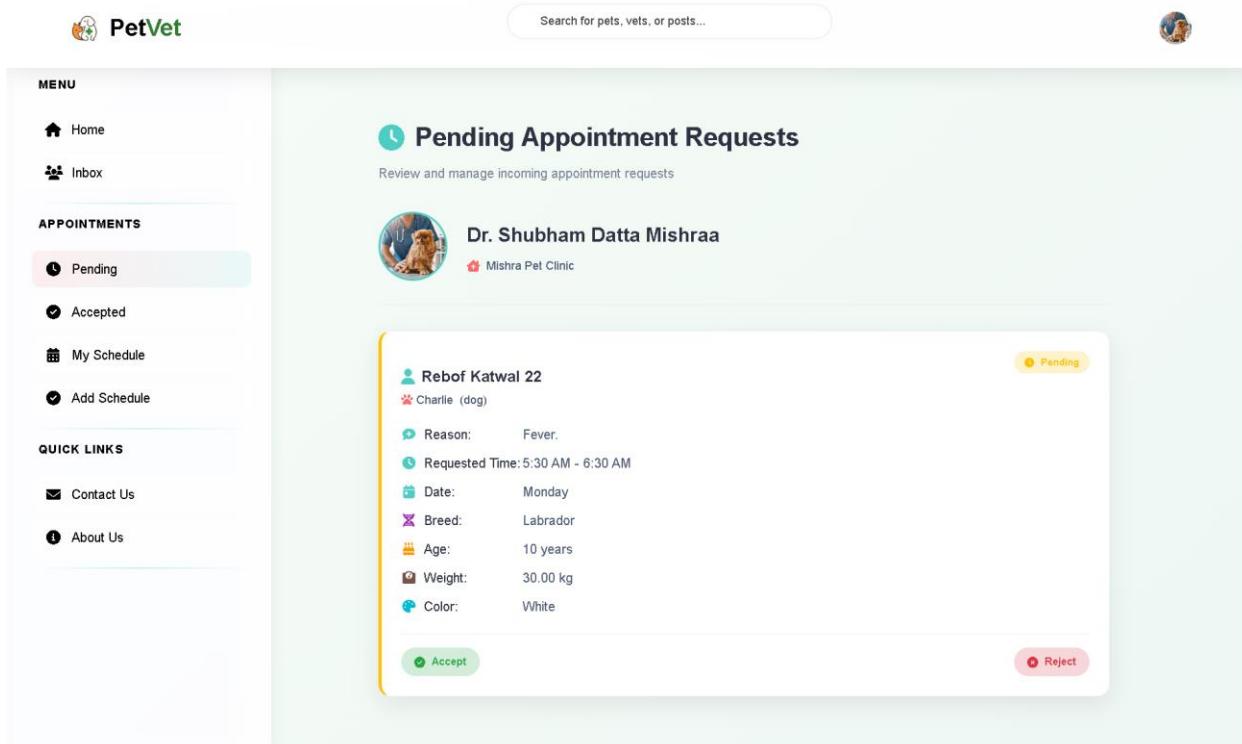


Figure 277: The Vet has received the request

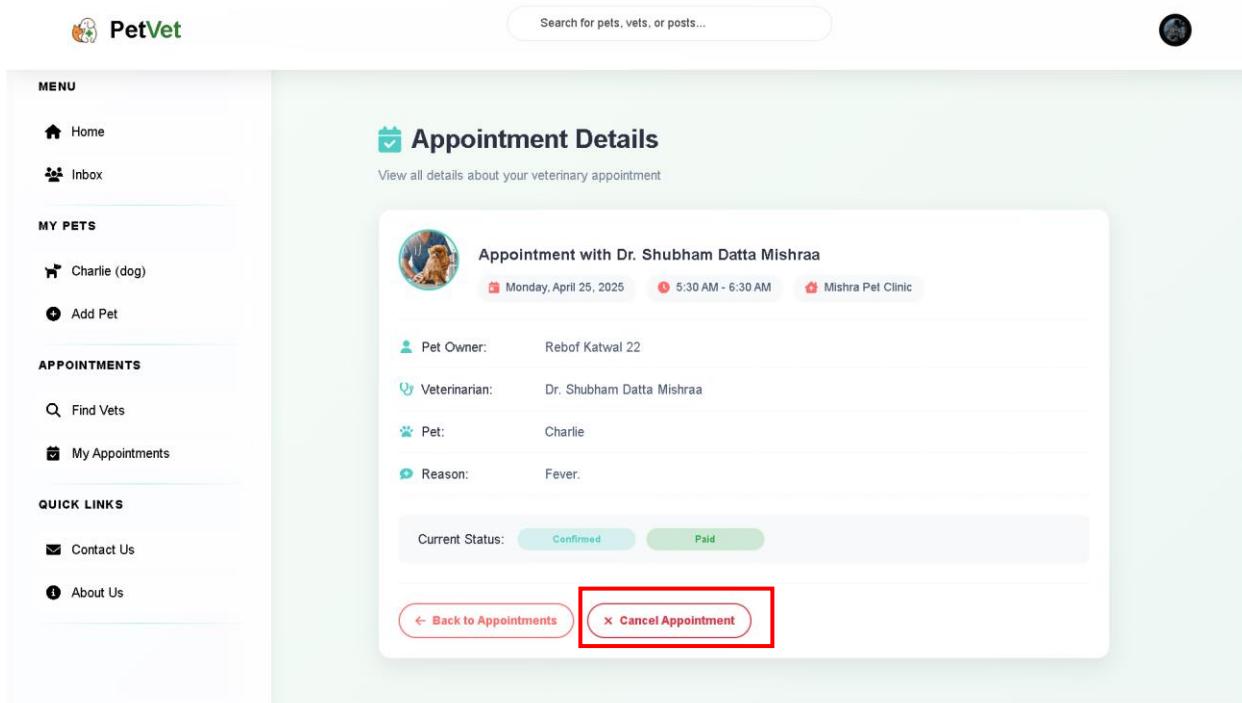


Figure 278: Clicking the cancellation button

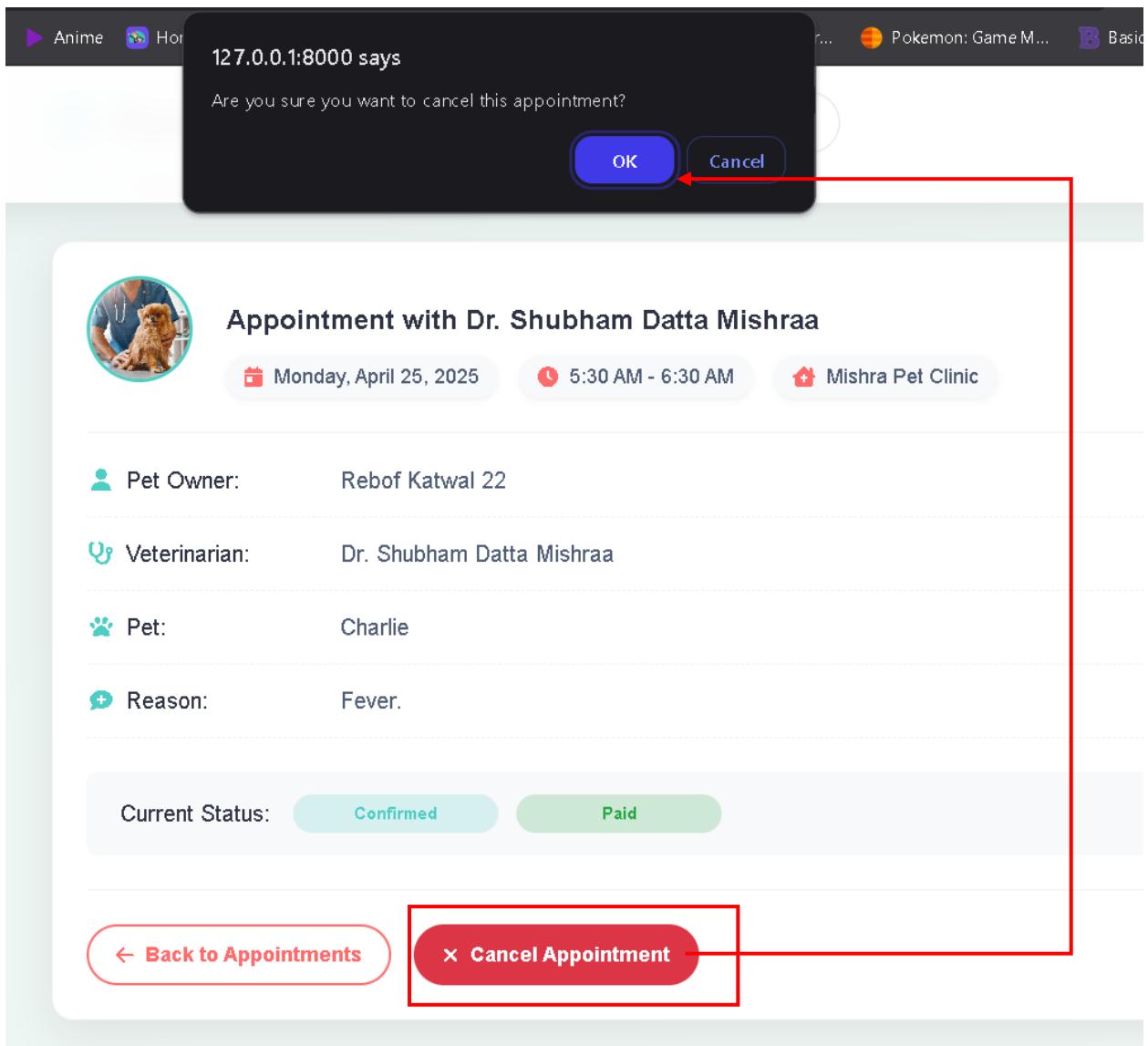


Figure 279: Cancellation with alert box

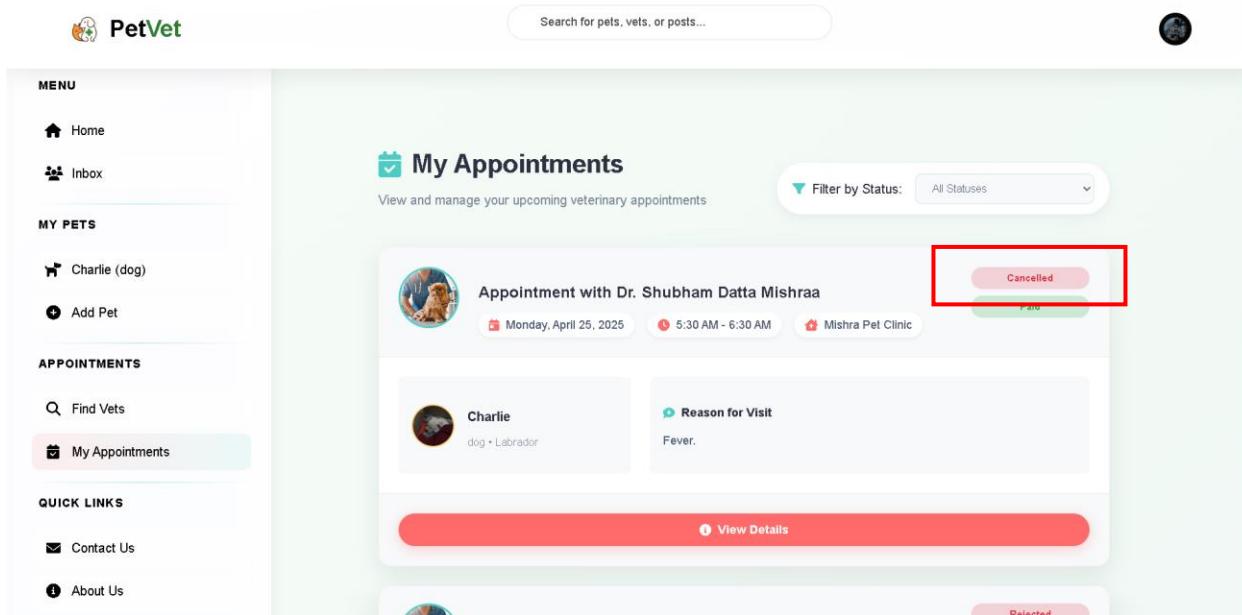


Figure 280: Status change to cancelled

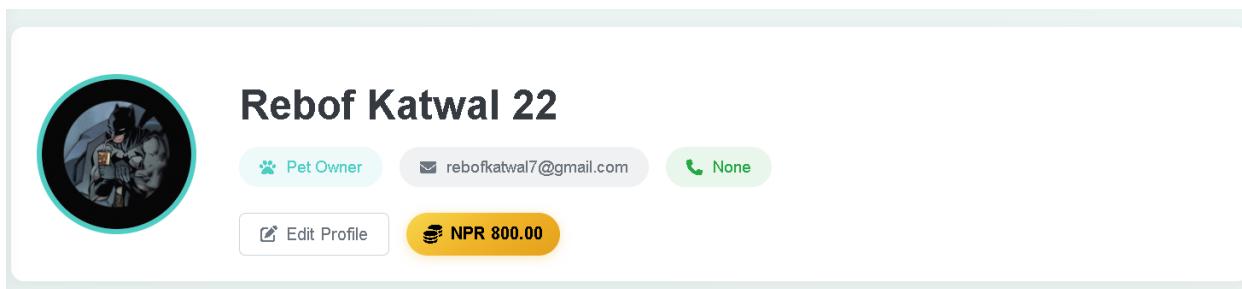


Figure 281: Only 80% of the original amount refunded

4.3.10 System Test: Appt7 Book Appointment Time Slot Availability Checking

Test no.	Appt7
Objective	To verify that appointment time slots are locked once confirmed and are released after completion, ensuring no double bookings.
Testcase	<ol style="list-style-type: none"> 1. Time Slot Unavailability on Vet's Confirmed Appointment 2. Time Slot Availability After Appointment Completion
Action	<ol style="list-style-type: none"> 1. Time Slot Unavailability on Vet's Confirmed Appointment <ul style="list-style-type: none"> • Logged in as Rebof Katwal (pet owner) • Had a confirmed appointment with vet Shubham Datta Mishra for Monday 5:30am - 6:30am • logged in as Astha Thapa (another pet owner) • Navigated to Find Vets, selected Shubham Datta Mishra, Checked Monday 5:30am - 6:30am slot which showed as Booked, unable to proceed with booking

	<p>2. Time Slot Availability After Appointment Completion</p> <ul style="list-style-type: none"> • Logged back in as vet Shubham Datta Mishra • Marked Rebof's appointment as complete • Logged in again as Astha Thapa • Checked same time slot (Monday 5:30am - 6:30am) — now available for booking
Expected Result	<ul style="list-style-type: none"> • Confirmed appointments lock the time slot for others • Once an appointment is marked complete, the slot becomes available again for new bookings
Actual Results	The confirmed time slot was correctly locked from other users. After vet completed the appointment, the slot was released and shown as available for booking.
Conclusion	The test was successful

Table 44: System Test - Time Slot Availability Checking

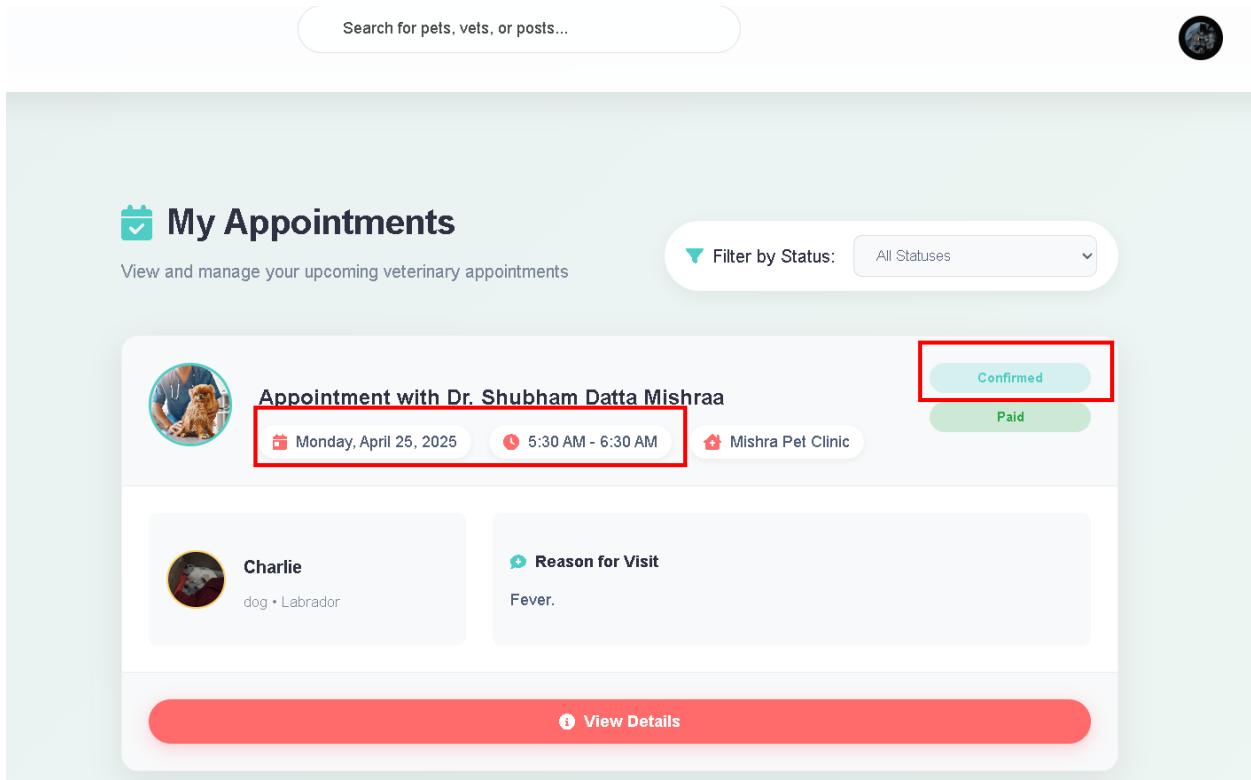


Figure 282: Appointment confirmed of Pet owner Rebof

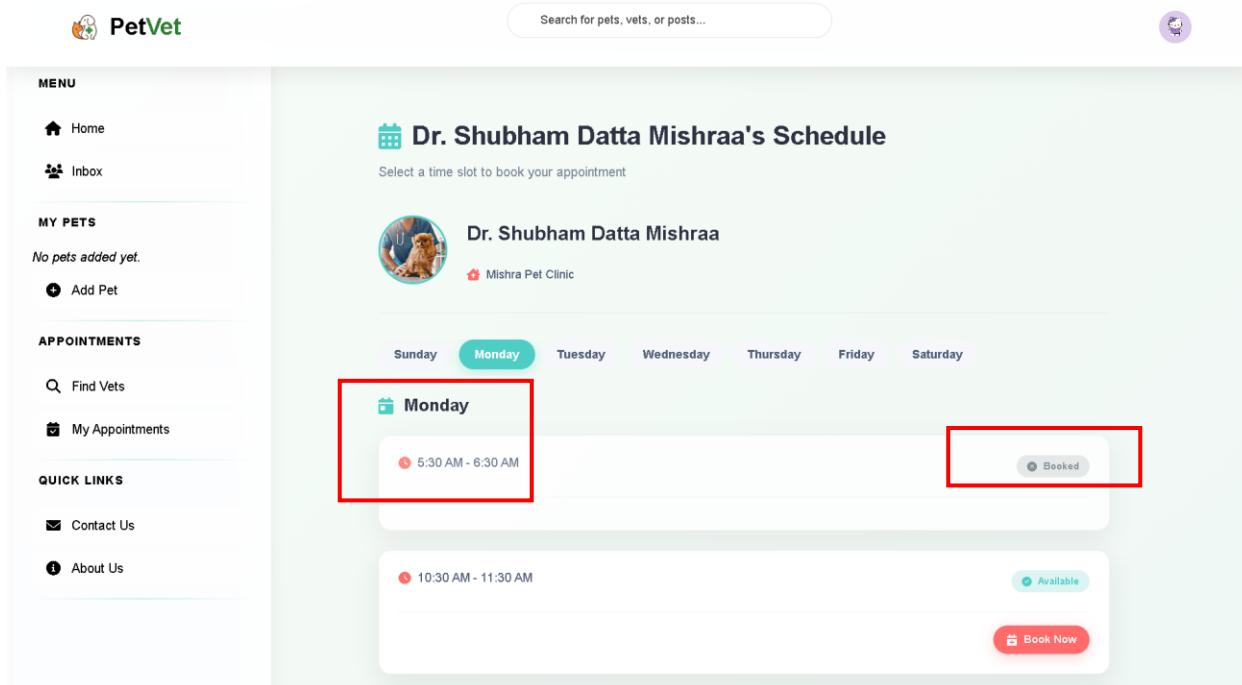


Figure 283: Astha Pet owner trying to book the same time slot

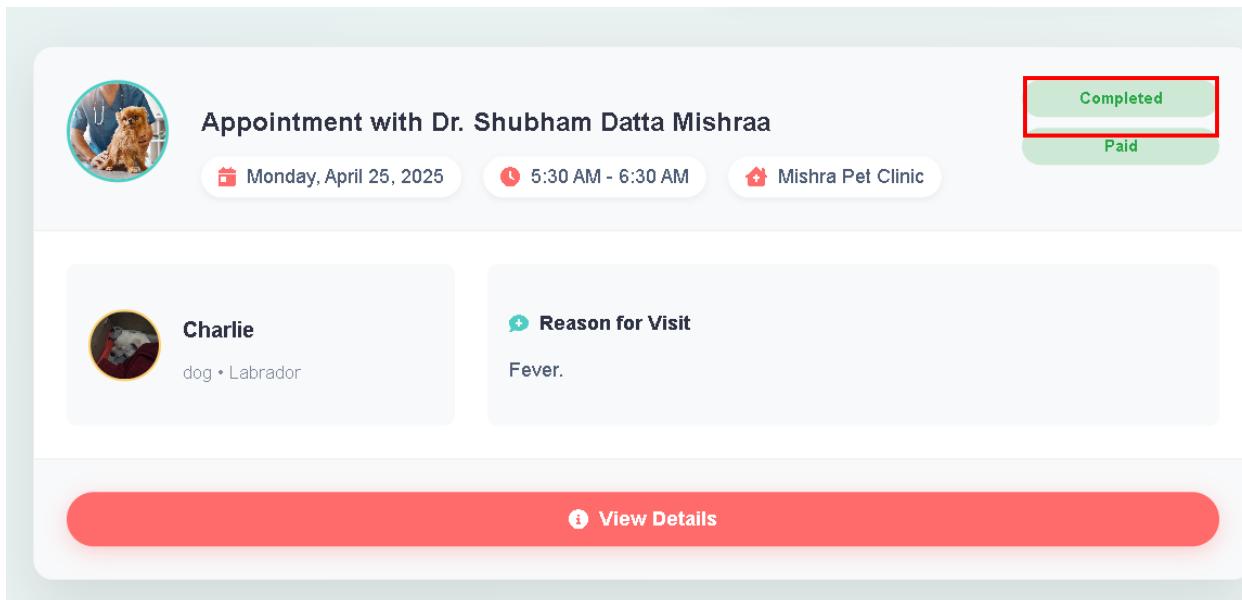


Figure 284: The appointment is completed of Pet owner Rebof

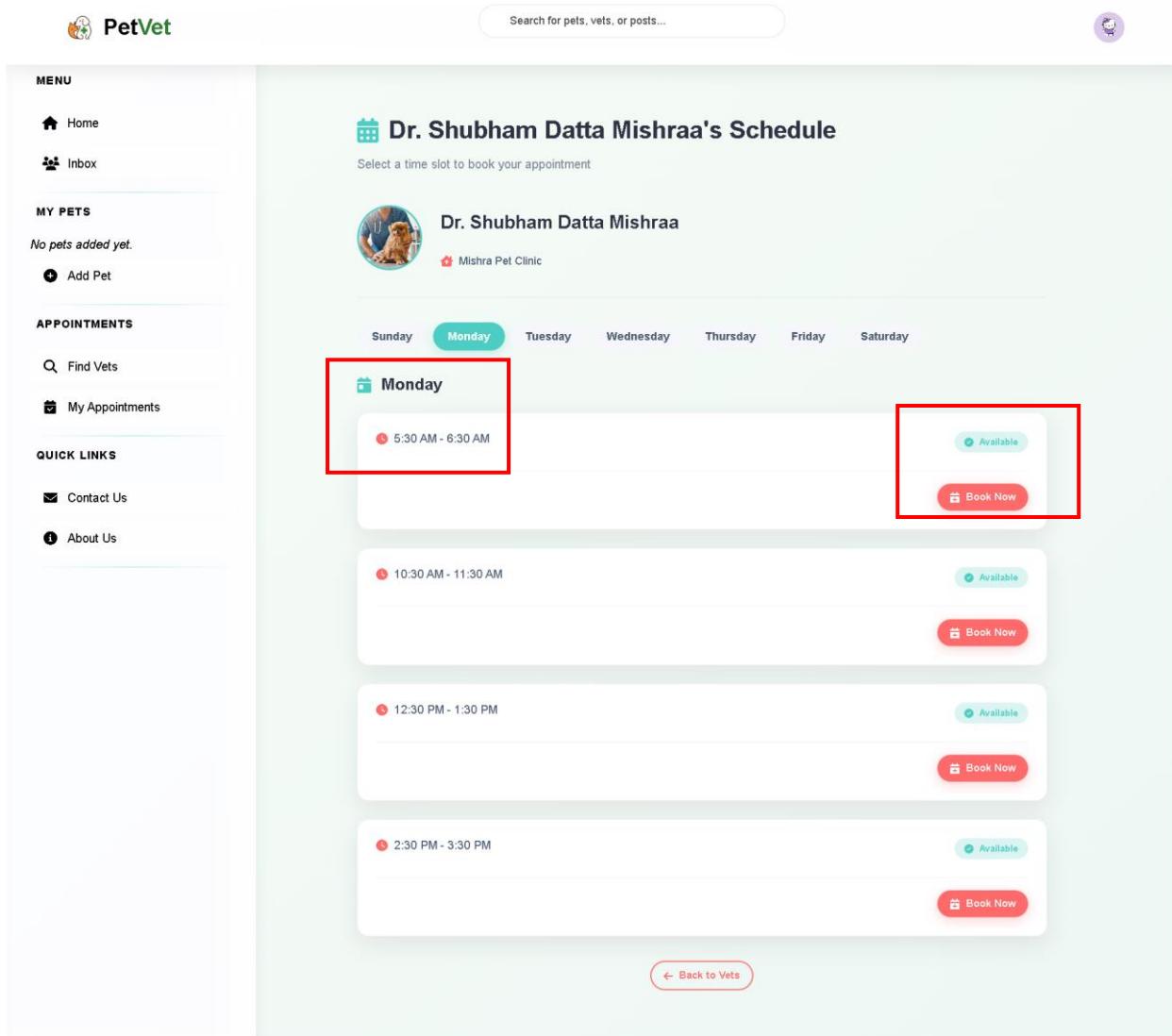


Figure 285: The time slot has opened up

4.3.11 System Test: RTC1 Real Time Chat Functionality

Test no.	RTC1
Objective	To verify that the real-time messaging system works correctly between a vet and pet owner, including message delivery, read receipts, and UI updates.
Testcase	1. Real-time Message Send and Receive and Read Receipt Functionality
Action	<ul style="list-style-type: none"> • Logged in as Shubham Datta Mishra (vet) • Sent message to Rebof Katwal: "Hey Rebof" → single tick appeared (message sent but unread) • Logged in as Rebof Katwal (pet owner) • Saw new message notification from Shubham • Opened the chat and replied: "hello today was a great session for my dog" • Shubham's original message now shows double tick (seen) • Logged back in as Shubham, saw the reply

	<p>and double tick indicating Rebof saw the original message</p> <ul style="list-style-type: none"> • All updates occurred in real-time without page refresh
Expected Result	<ul style="list-style-type: none"> • Messages should send and appear instantly with proper tick icons indicating sent and read status in real time.
Actual Results	Messages were sent, received, and marked as read in real time with accurate single/double tick indicators.
Conclusion	The test was successful

Table 45: System Test - Real Time Chat

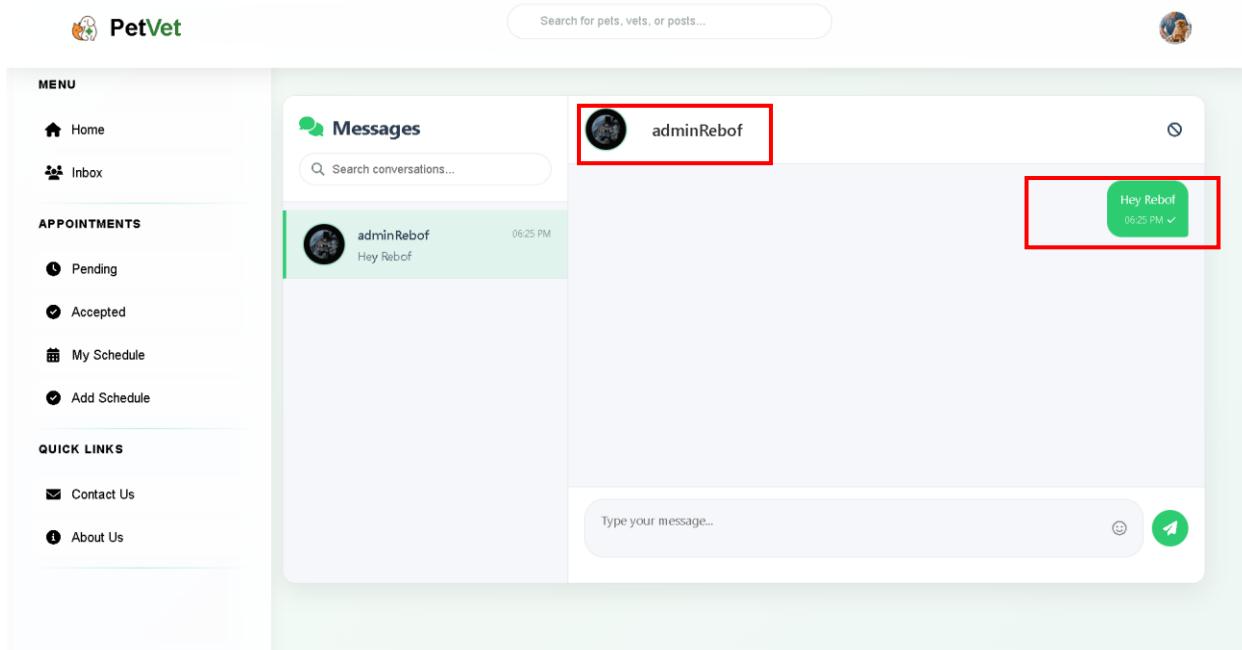


Figure 286: Sending message to adminRebof as a Vet

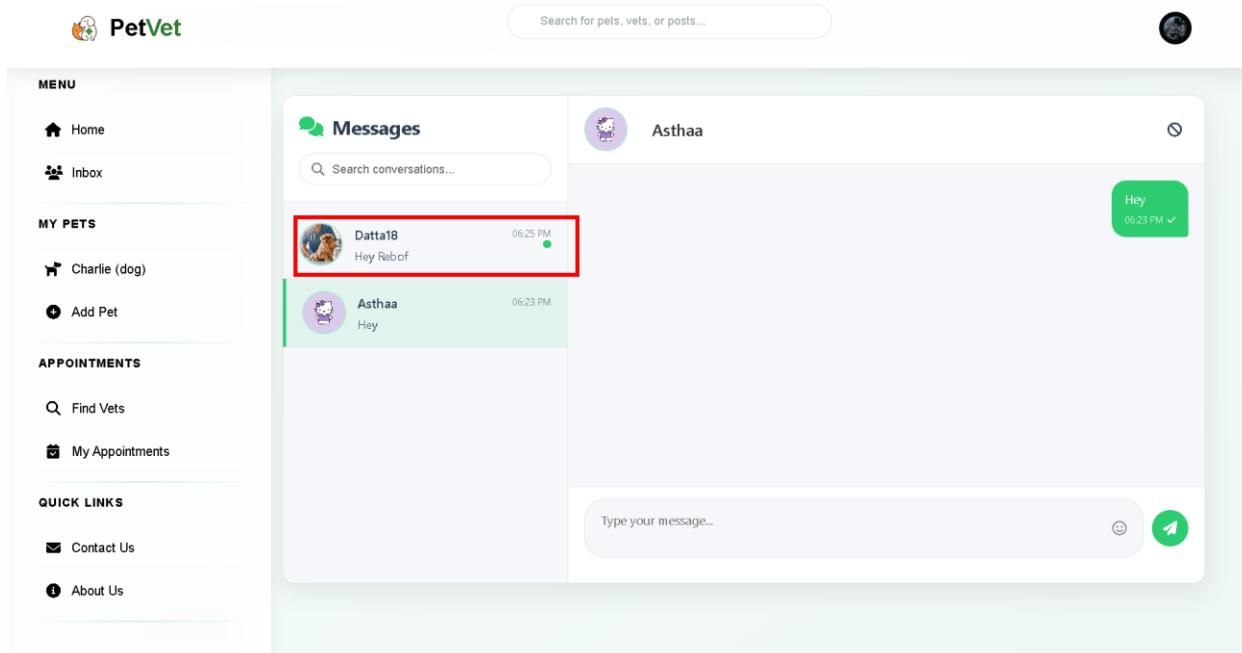


Figure 287: Receiving message from the Vet

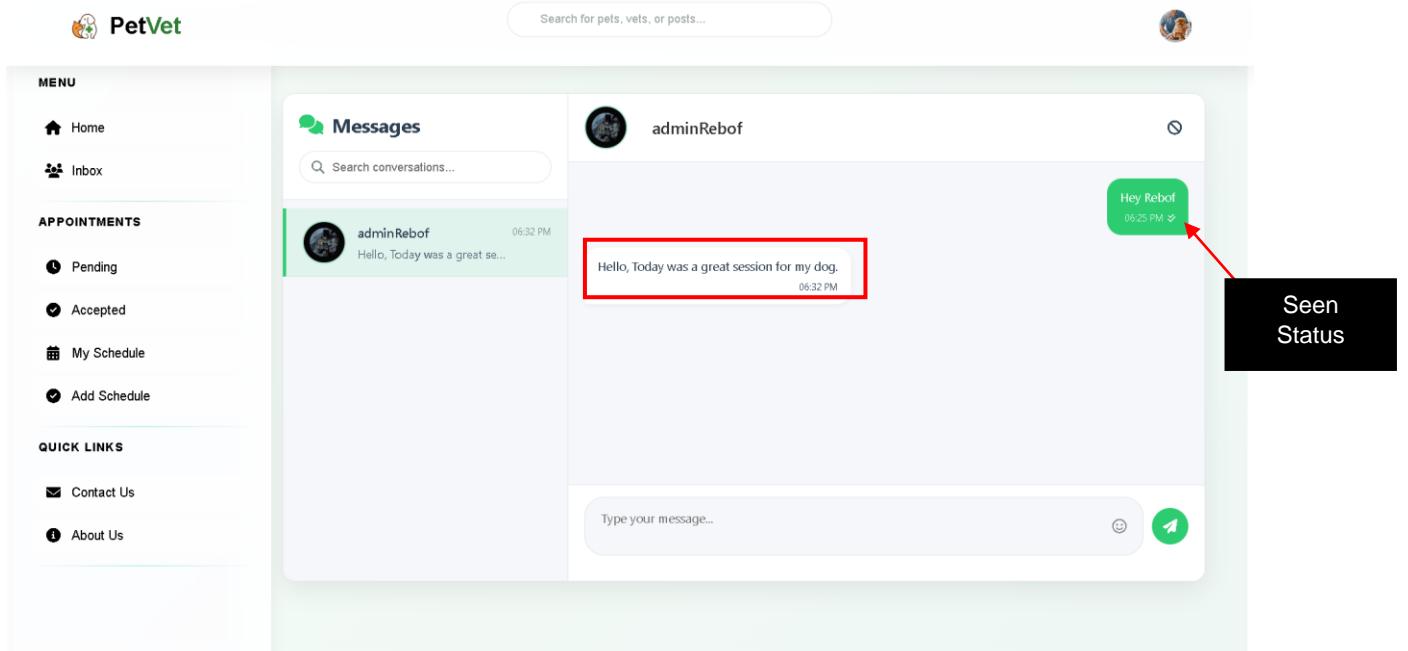


Figure 288: Vet receives message from adminRebof

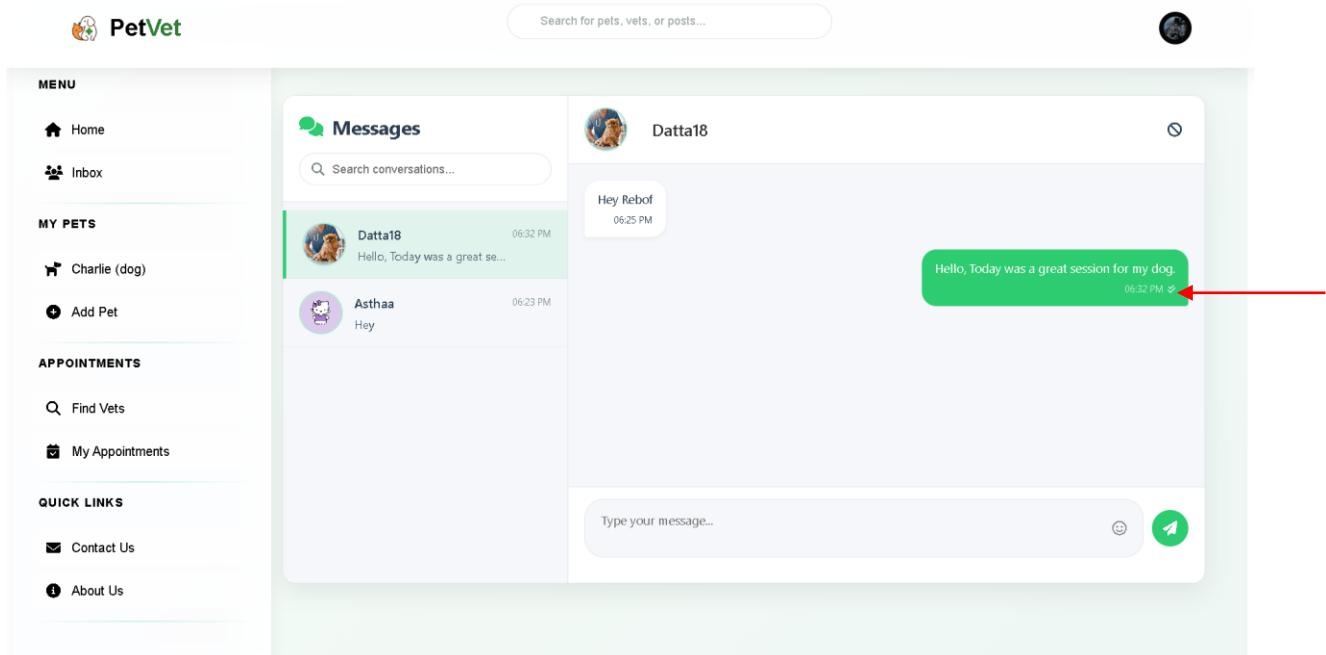


Figure 289: Receiving seen status after vet sees the meesage

4.3.12 System Test: Po1 Reply Creation and Deletion Functionality

Test no.	Po1
Objective	To test the reply and delete functionality in the post detail page ensuring required fields are validated and deletions are confirmed.
Testcase	<ol style="list-style-type: none"> 1. Reply with Empty Content 2. Successful Reply Submission 3. Reply Deletion with Confirmation
Action	<ol style="list-style-type: none"> 1. Reply with Empty Content <ul style="list-style-type: none"> • Logged in as Rebof Katwal (pet owner) • Navigated to Home Page and clicked on the first post • Landed on Post Details Page • Attempted to submit a reply without content → got a prompt: "Please fill in this field" 2. Entered valid reply content: "True" → reply successfully posted 3. Reply Deletion <ul style="list-style-type: none"> • Clicked Delete on the posted reply → shown a confirmation alert

	<ul style="list-style-type: none"> Confirmed deletion → reply successfully removed from the post
Expected Result	<ul style="list-style-type: none"> Form should validate empty input, successful reply should appear immediately, and deletion should prompt for confirmation before removing the reply.
Actual Results	All behaviors matched the expected result, with validation and deletion functioning as intended.
Conclusion	The test was successful

Table 46: System Test - Reply Addition and Deletion

Home

Inbox

MY PETS
Charlie (dog)

Add Pet

APPOINTMENTS
Find Vets
My Appointments

QUICK LINKS
Contact Us
About Us

Sort Posts
Newest Old Most Liked Most Comments

Grooming
Posted by u/Juliette 4 hours, 8 minutes ago

The Best Grooming Tips for Long-Haired Cats

long-haired cats like Persians, Maine Coons, and Ragdolls require extra grooming to keep their coats silky and tangle-free. Here are some essential grooming tips: Daily Brushing: Regular brushing prevents mats ...

1 comment

Vaccination
Posted by u/Catata18 4 hours, 9 minutes ago

Core Vaccines Every Dog Needs

Vaccines are vital for protecting your dog against dangerous diseases. Here are the core vaccines most vets recommend: Rabies – Required by law in many places; this protects both pets ...

2 comments

Grooming
Posted by u/Juliette 4 hours, 27 minutes ago

How to Keep Your Cat's Coat Smooth and Tangle-Free

Cats are natural groomers, but sometimes they need a little help—especially long-haired breeds. Here's how to keep their fur fabulous: Brush often: Daily brushing removes loose fur and prevents matting ...

2 comments

Grooming
Posted by u/Juliette 4 hours, 29 minutes ago

5 Essential Grooming Tips for a Healthy, Happy Dog

Grooming isn't just about looking good—it's about your dog's overall health and comfort. Here are five tips to keep your furry friend in top shape: Brush regularly—Especially for ...

5 comments

Emergency
Posted by u/Hobbit 4 hours, 30 minutes ago

My Dog Swallowed a Battery – What Should I Do?

Last night, my beagle somehow got hold of a TV remote and swallowed a battery. We rushed to the emergency vet, but I'm still shaken. Has anyone dealt with this ...

0 comments

Categories
Emergency
Grooming
Vaccination

Trending Posts

- My Dog Swallowed a Battery – What Should I Do... 0 likes + 1 comment
- Cat Vaccines Every Dog Needs 2 likes + 1 comment
- How to Keep Your Cat's Coat Smooth and... 2 likes + 1 comment
- The Best Grooming Tips for Long-Haired Cats 1 like + 0 comments
- 5 Essential Grooming Tips for a Healthy,... 3 likes + 2 comments

Figure 290: Selecting a post

The screenshot shows a mobile application interface for 'PetVet'. At the top, there is a search bar with placeholder text 'Search for pets, vets, or posts...'. On the right side of the header is a user profile icon.

Left Sidebar (Menu):

- Home
- Inbox
- MY PETS**
 - Charlie (dog)
 - Add Pet
- APPOINTMENTS
 - Find Vets
 - My Appointments
- QUICK LINKS
 - Contact Us
 - About Us

Post Details:

Category: Grooming

Posted by: uRebofff 4 hours, 36 minutes ago

5 Essential Grooming Tips for a Healthy, Happy Dog

Image: A small dog wrapped in a white towel, standing in front of bubbles.

Text: Grooming isn't just about looking good—it's about your dog's overall health and comfort. Here are five tips to keep your furry friend in top shape:

- Brush regularly** – Especially for long-haired breeds, daily brushing prevents painful matting and reduces shedding.
- Use the right shampoo** – Always choose a pet-safe, pH-balanced shampoo. Avoid human shampoos as they can irritate your dog's skin.
- Check ears and eyes** – Clean ears with vet-approved solutions and keep an eye out for redness or discharge.
- Nail trimming matters** – Overgrown nails can cause discomfort or even affect posture. Trim carefully or get help from a groomer.
- Make it positive** – Use treats and gentle encouragement to make grooming a stress-free experience.

Consistency is key—regular grooming builds trust and keeps your dog healthy and happy.

Engagement: 5 likes, 2 comments.

Comment Section:

What are your thoughts?

Comments:

- uAsthaa** 4 hours, 27 minutes ago

Resourcefull

[Reply](#) [Share](#)
- uDatta18** 4 hours, 13 minutes ago

Indeed
- uAvi** 4 hours, 1 minute ago

Agreed
- uteju** 3 hours, 56 minutes ago

umm nice

[Reply](#) [Share](#)

Figure 291: Redirected to post details page

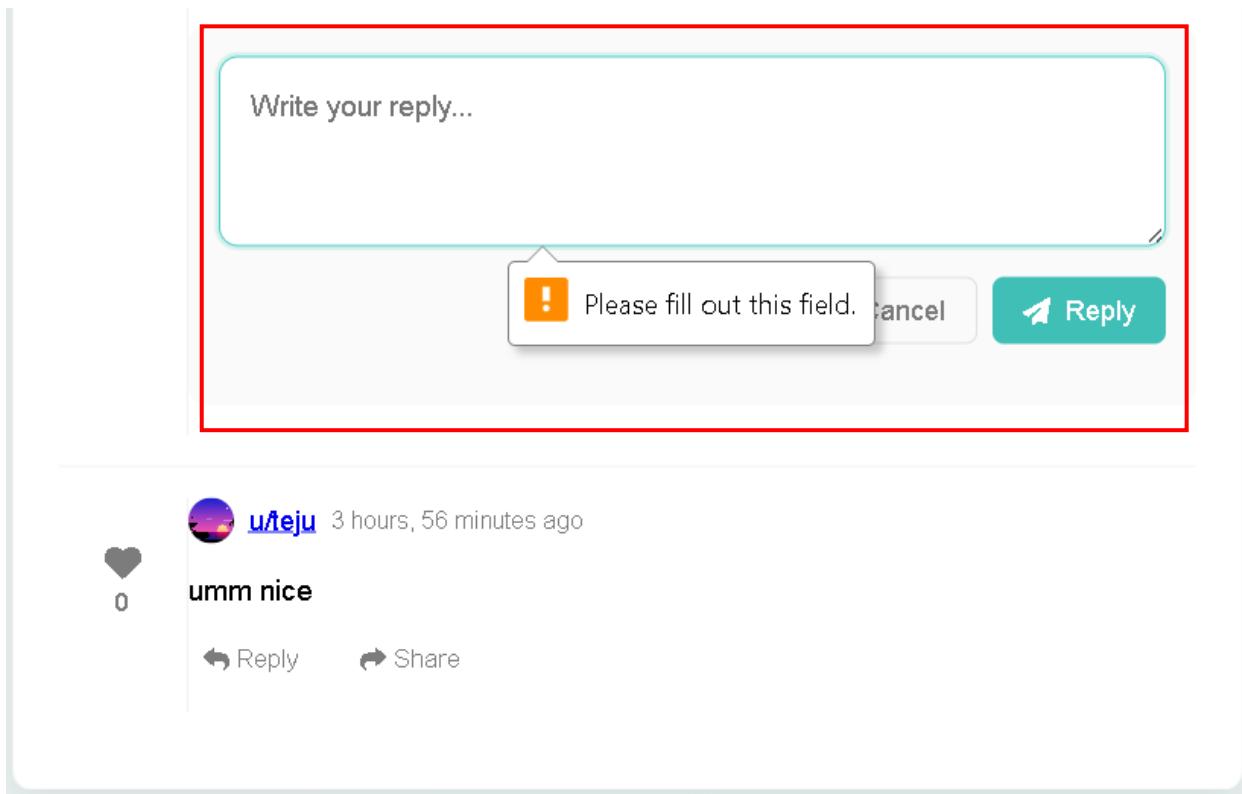


Figure 292: Leaving the reply empty

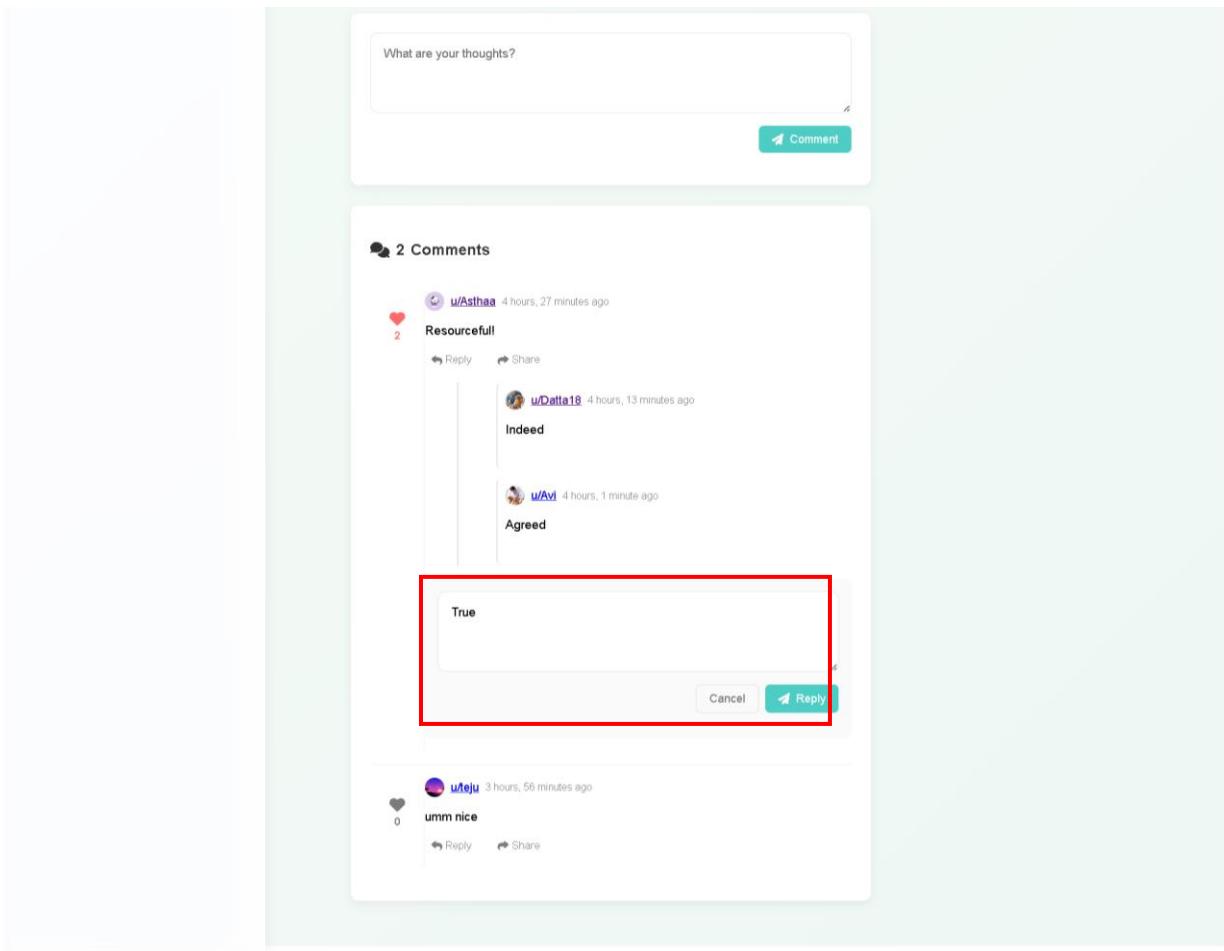


Figure 293: Submitting a proper reply

The image shows a social media interface with a light gray background. At the top left is a user icon with a purple gradient and the text "u/Asthaa". To its right is the text "4 hours, 27 minutes ago". Below this is a red heart icon with the number "2" next to it. The main text of the post is "Resourcefull!". Underneath the post are two interaction buttons: "Reply" with a left arrow icon and "Share" with a right arrow icon. A vertical gray line separates this from the next comment. The second comment starts with a user icon of a dog and the text "u/Datta18". To its right is the timestamp "4 hours, 13 minutes ago". The text of the comment is "Indeed". Another vertical gray line follows. The third comment starts with a user icon of a person and the text "u/Avi". To its right is the timestamp "4 hours, 1 minute ago". The text of the comment is "Agreed". A vertical gray line follows. The fourth comment starts with a user icon of a cat and the text "u/adminRebof". To its right is the timestamp "0 minutes ago". The text of the comment is "True". Below this comment is a small "Delete" button with a trash can icon. A horizontal gray line runs across the screen below these comments. At the bottom left is another user icon with a purple gradient and the text "u/teju". To its right is the timestamp "3 hours, 56 minutes ago". Below this is the text "umm nice". To the right of "umm nice" is a teal-colored rectangular button with white text that says "Reply added successfully!".

Figure 294: Reply has been added

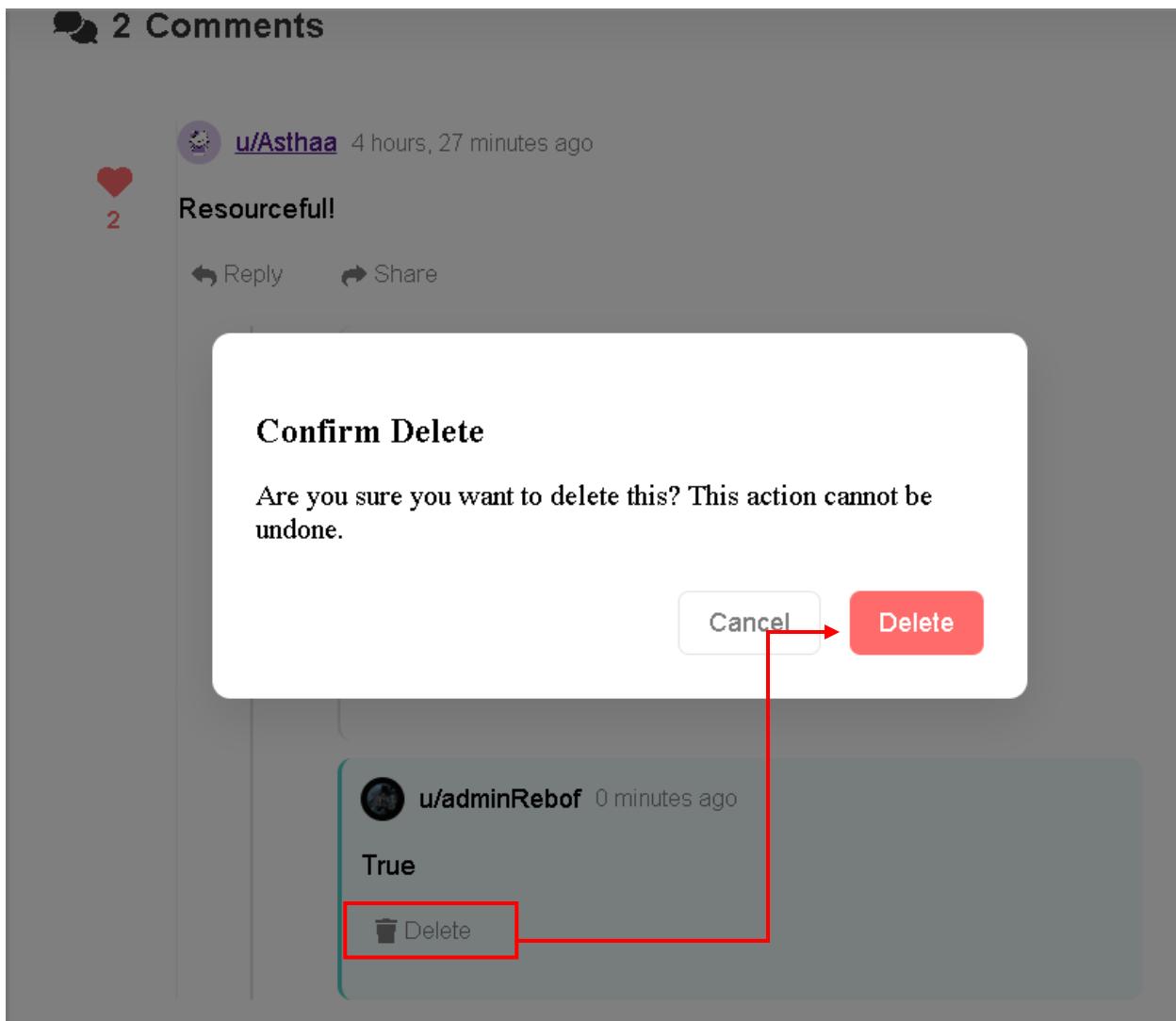


Figure 295: Deleting the added reply

2 Comments

 u/Asthaa 4 hours, 27 minutes ago

 2
Resourcefull!

 Reply  Share

 u/Datta18 4 hours, 13 minutes ago

Indeed

 u/Avi 4 hours, 1 minute ago

Agreed

 u/teju 3 hours, 56 minutes ago

 0
umm nice

 Reply 

Item deleted successfully

Figure 296: Reply has been deleted

4.3.13 System Test: Po2 Comment Creation and Deletion Functionality

Test no.	Po2
Objective	To test the comment and delete functionality in the post detail page ensuring required fields are validated and deletions are confirmed.
Testcase	<ol style="list-style-type: none"> 1. Comment with Empty Content 2. Successful Comment Submission 3. Comment Deletion with Confirmation
Action	<ol style="list-style-type: none"> 4. Comment with Empty Content <ul style="list-style-type: none"> • Logged in as Rebof Katwal (pet owner) • Navigated to Home Page and clicked on the first post • Landed on Post Details Page • Attempted to submit a comment without content → got a prompt: "Please fill in this field" 5. Entered valid reply content: "Wonder! I will use th tips" → comment successfully posted 6. Comment Deletion

	<ul style="list-style-type: none"> • Clicked Delete on the posted reply → shown a confirmation alert • Confirmed deletion → reply successfully removed from the post
Expected Result	<ul style="list-style-type: none"> • Form should validate empty input, successful comment should appear immediately, and deletion should prompt for confirmation before removing the comment.
Actual Results	All behaviors matched the expected result, with validation and deletion functioning as intended.
Conclusion	The test was successful

Table 47: System Test - Comment Addition and Deletion

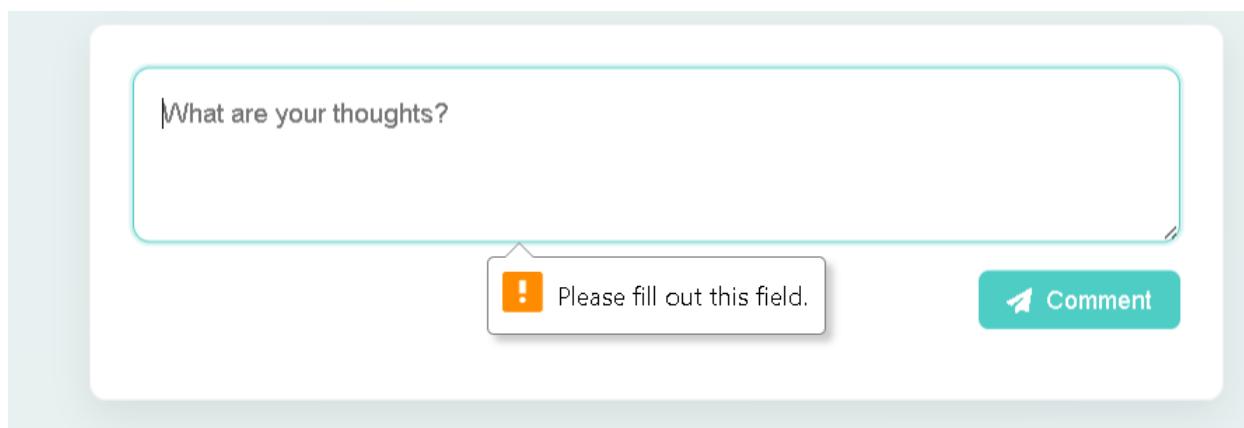


Figure 297: Leaving the comment field empty

The screenshot shows a user interface for adding a comment. At the top, there is a text input field with the placeholder "What are your thoughts?". Below it is a teal-colored button labeled "Comment" with a white icon. In the main content area, there is a section titled "3 Comments". The first comment is by "u/adminRebof" (0 minutes ago), which reads "Wonderful! I will use the tips." This comment has a red rectangular box around it. Below the comment are three interaction buttons: "Reply", "Share", and "Delete". To the left of the comment, there is a heart icon with the number "0". The second comment is by "u/Asthaa" (4 hours, 27 minutes ago), which reads "Resourceful!". This comment has a red heart icon with the number "2" next to it. Below the comment are two interaction buttons: "Reply" and "Share". A teal-colored success message box is overlaid on the right side of the comment, stating "Comment added successfully!".

Figure 298: Added a valid comment

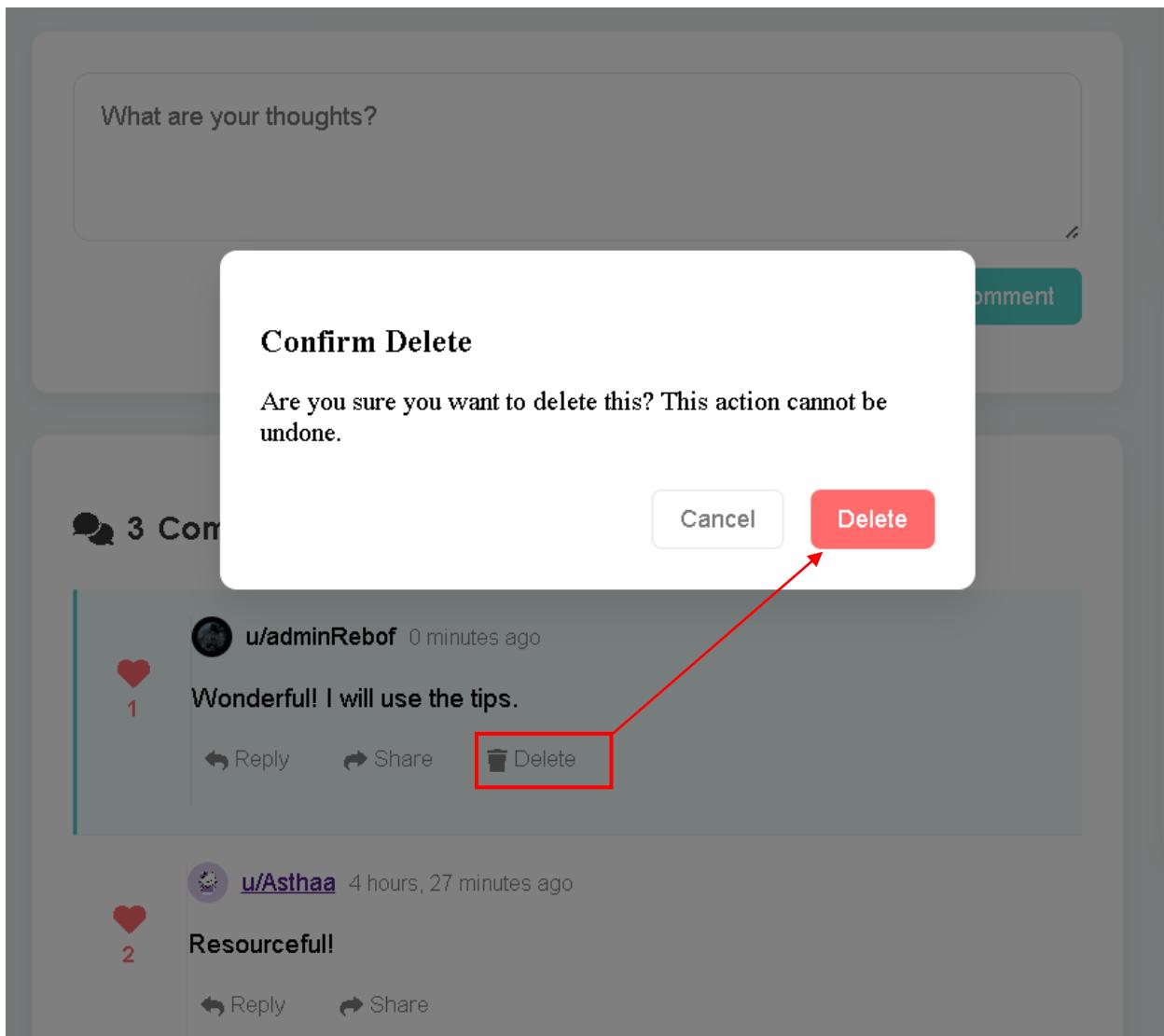


Figure 299: Deleting the comment

The image shows a comment section with the following details:

- Comments:** 2 Comments
- Comment 1:** u/Asthaa 4 hours, 27 minutes ago
Resourcefull!
2 hearts
- Comment 2:** u/Datta18 4 hours, 13 minutes ago
Indeed
- Comment 3:** u/Avi 4 hours, 1 minute ago
Agreed
- Action:** A teal button labeled "Item deleted successfully" is visible, indicating that the third comment has been deleted.

Figure 300: Comment has been deleted

4.3.14 System Test: Po3 Like and Unlike Functionality

Test no.	Po3
Objective	To verify that the like/unlike functionality works correctly for both comments and posts, updating the UI and like count accordingly.
Testcase	<ol style="list-style-type: none"> 1. Like and Unlike a Comment 2. Like and Unlike a Post
Action	<ol style="list-style-type: none"> 1. Like and Unlike a Comment <ul style="list-style-type: none"> • Navigated to a post detail page • Located a comment with 0 likes and a grey heart • Clicked the button → like count became 1 and heart turned red • Clicked the button again → like count dropped to 0 and heart turned grey 2. Like and Unlike a Post <ul style="list-style-type: none"> • Located the post with 0 likes and a grey heart • Clicked the like button → like count became 1 and heart turned red

	<ul style="list-style-type: none"> Clicked the button again → like count dropped to 0 and heart turned grey
Expected Result	<ul style="list-style-type: none"> Clicking like should toggle the like count and heart color appropriately for both comments and posts.
Actual Results	Like/unlike behavior reflected in near real-time for both comments and posts as expected.
Conclusion	The test was successful

Table 48: System Test - Like and Unlike Toggle

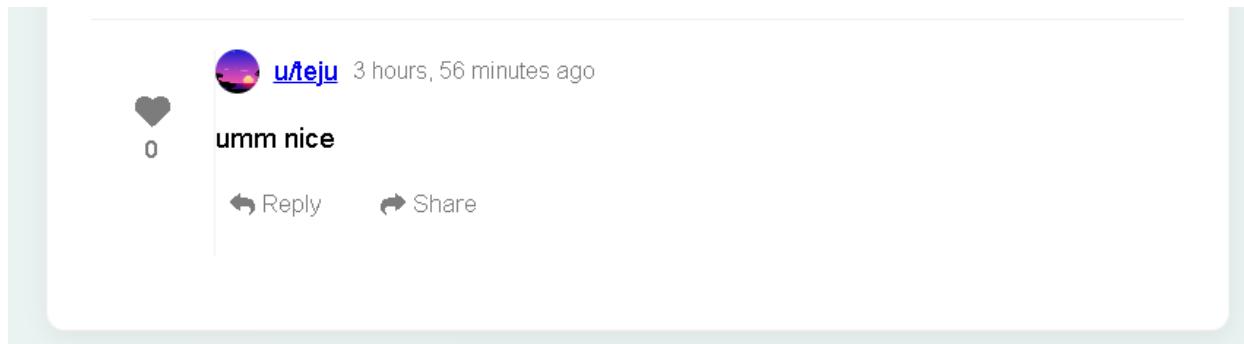


Figure 301: A comment with 0 likes

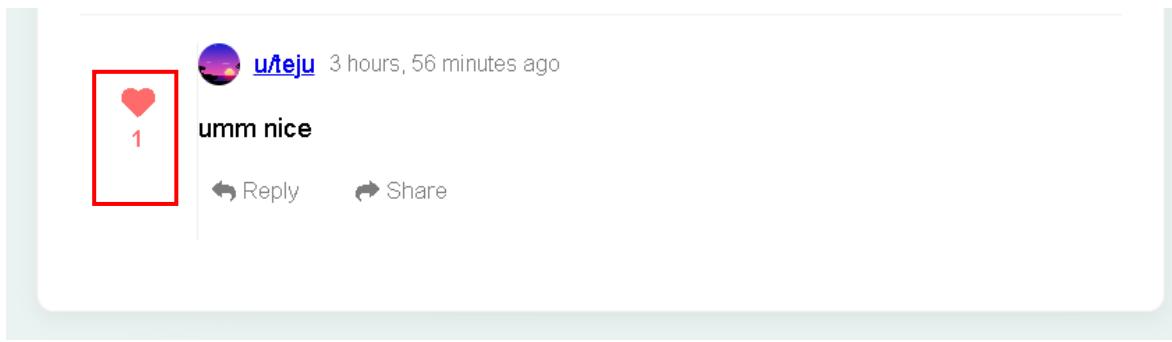


Figure 302: Liking the comment

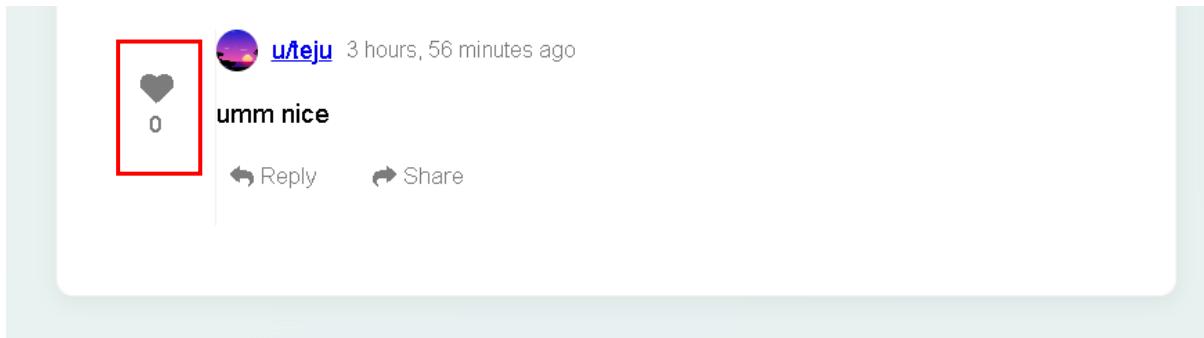


Figure 303: Unlike the comment

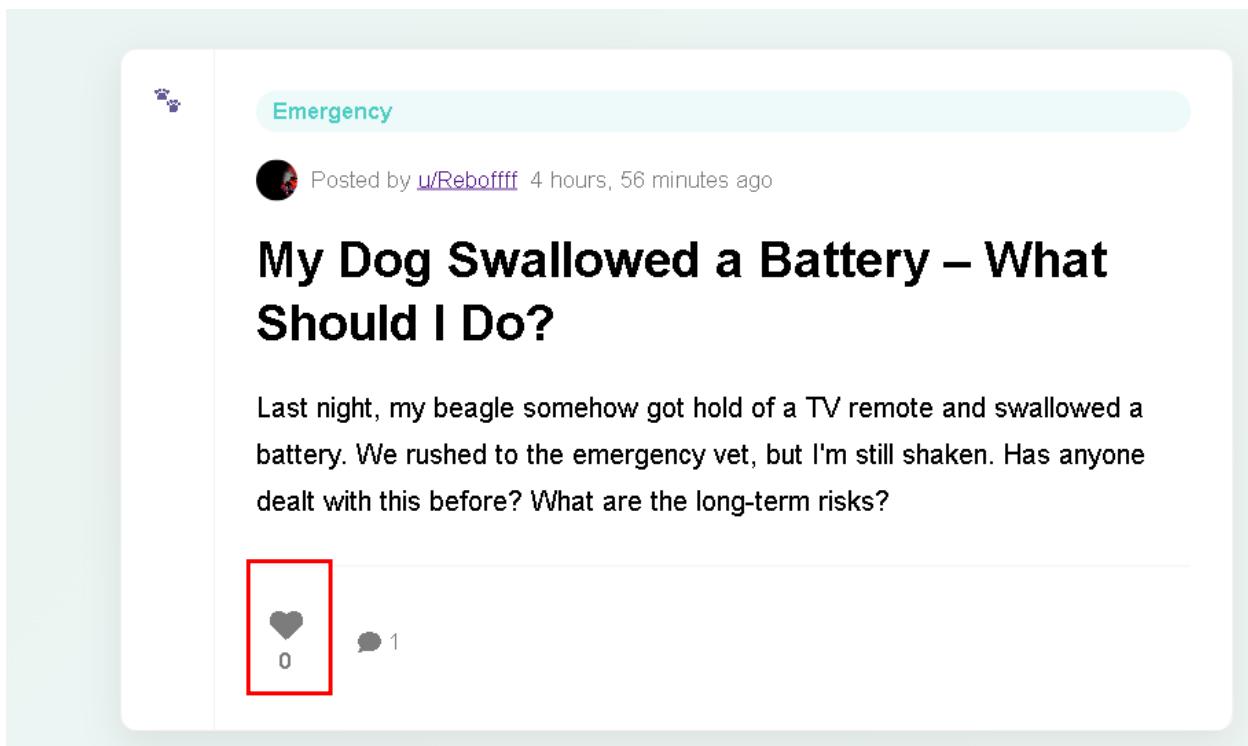


Figure 304: A post with zero likes

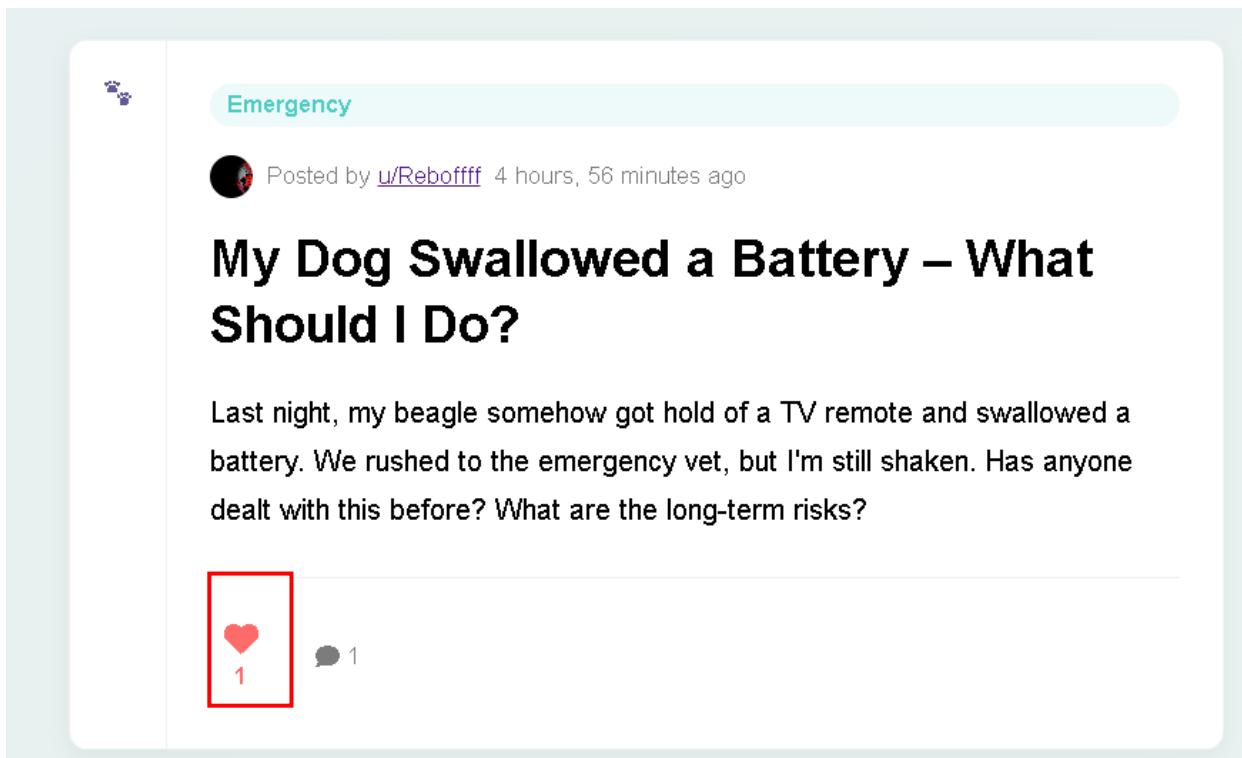


Figure 305: Like the post

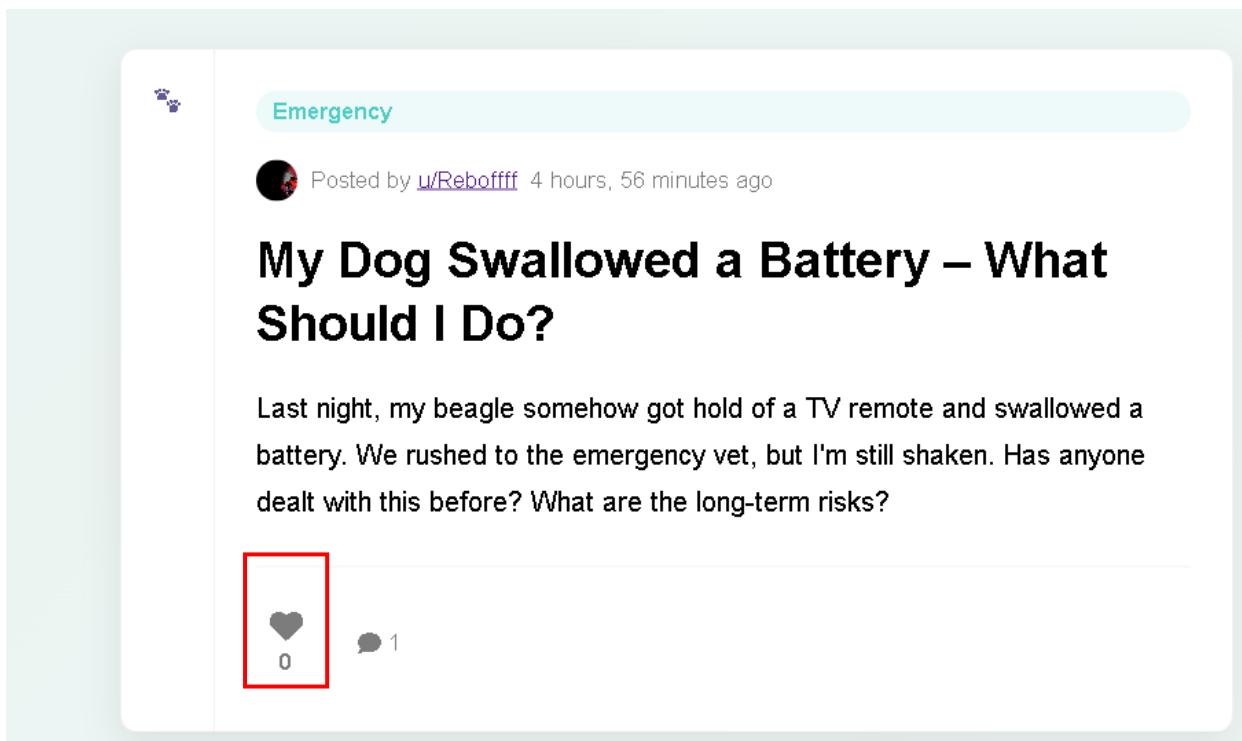


Figure 306: Unlike the post

4.3.15 System Test: Po4 Post Functionality

Test no.	Po4
Objective	To ensure the post creation, editing, and deletion functionalities work correctly and validations are enforced during the process.
Testcase	<ol style="list-style-type: none"> 1. Post Creation with Missing Category 2. Successful Post Creation 3. Post Edit with Missing Title 4. Successful Post Update 5. Post Deletion with Confirmation
Action	<ol style="list-style-type: none"> 1. Post Creation with Missing Category <ul style="list-style-type: none"> • Clicked on "What's on your mind?", which opened the create post modal • Entered title: "Why vaccinating our pet is a must" but left category empty • Prompted to fill the required category field 2. Successful Post Creation

	<ul style="list-style-type: none"> • Filled in all fields properly, including a picture, and submitted the post • Redirected to the post detail page <p>3. Clicked edit post, left title empty → prompted to fill the title</p> <p>4. Updated title to: "Protect your pet: The power of vaccination" → successfully updated</p> <p>5. Successful Deletion of the Post</p> <ul style="list-style-type: none"> • Clicked delete, confirmation popup appeared • Confirmed and post was successfully deleted
Expected Result	<ul style="list-style-type: none"> • Fields must be validated before submission • Post creation, update, and deletion should reflect changes properly and redirect as expected
Actual Results	Post operations (create, edit, delete) performed correctly with appropriate validations and feedback.
Conclusion	The test was successful

Table 49: System Test - Post Addition, Deletion and Updating

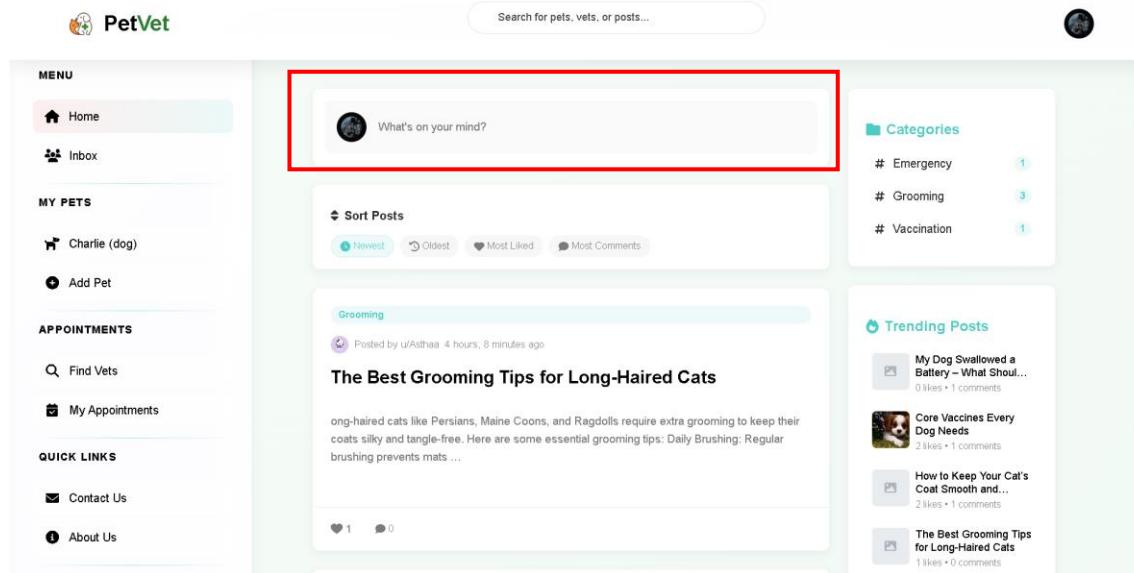


Figure 307: Clicking one "What's on your mind?"

The screenshot shows the 'Create a Post' modal. At the top is a title 'Create a Post' and a close button 'X'. Below is a 'Title' input field containing 'Why Vaccinating Your Pet is a Must' with a small icon. A 'Category' dropdown is shown with a placeholder 'Select a category'. A 'Body' input field below it contains a warning message: 'Please select an item in the list.' A large input field for 'What's on your mind?' is at the bottom. An 'Image' section shows a dashed box for selecting an image, with a placeholder 'No image selected' and a camera icon.

Figure 308: Leaving the category field empty

Why Vaccinating Your Pet is a Must 🐶💉

Category

Vaccination

Body

Vaccinations protect your furry friends from serious diseases like rabies, parvo, and distemper. Not only do they boost your pet's immunity, but they also prevent the spread of infections to other animals and even humans. Keep your pet's shots up-to-date and always follow your vet's vaccination schedule. A little poke today can save a lot of trouble tomorrow!

Image



Upload Image

Figure 309: Creating a proper post

The screenshot displays the PetVet mobile application interface. On the left, a sidebar contains a navigation menu with options like Home, Inbox, Add Pet, Find Vets, My Appointments, Contact Us, and About Us. The main content area shows a post titled "Why Vaccinating Your Pet is a Must" by "adminRebof". The post includes a photo of a golden retriever dog. A success message at the top says "Post created successfully". To the right of the post, there's a sidebar with categories (Emergency, Grooming, Vaccination) and trending posts. At the bottom, there's a comment section with a placeholder "What are your thoughts?" and a "Comment" button.

Figure 310: Post has been added

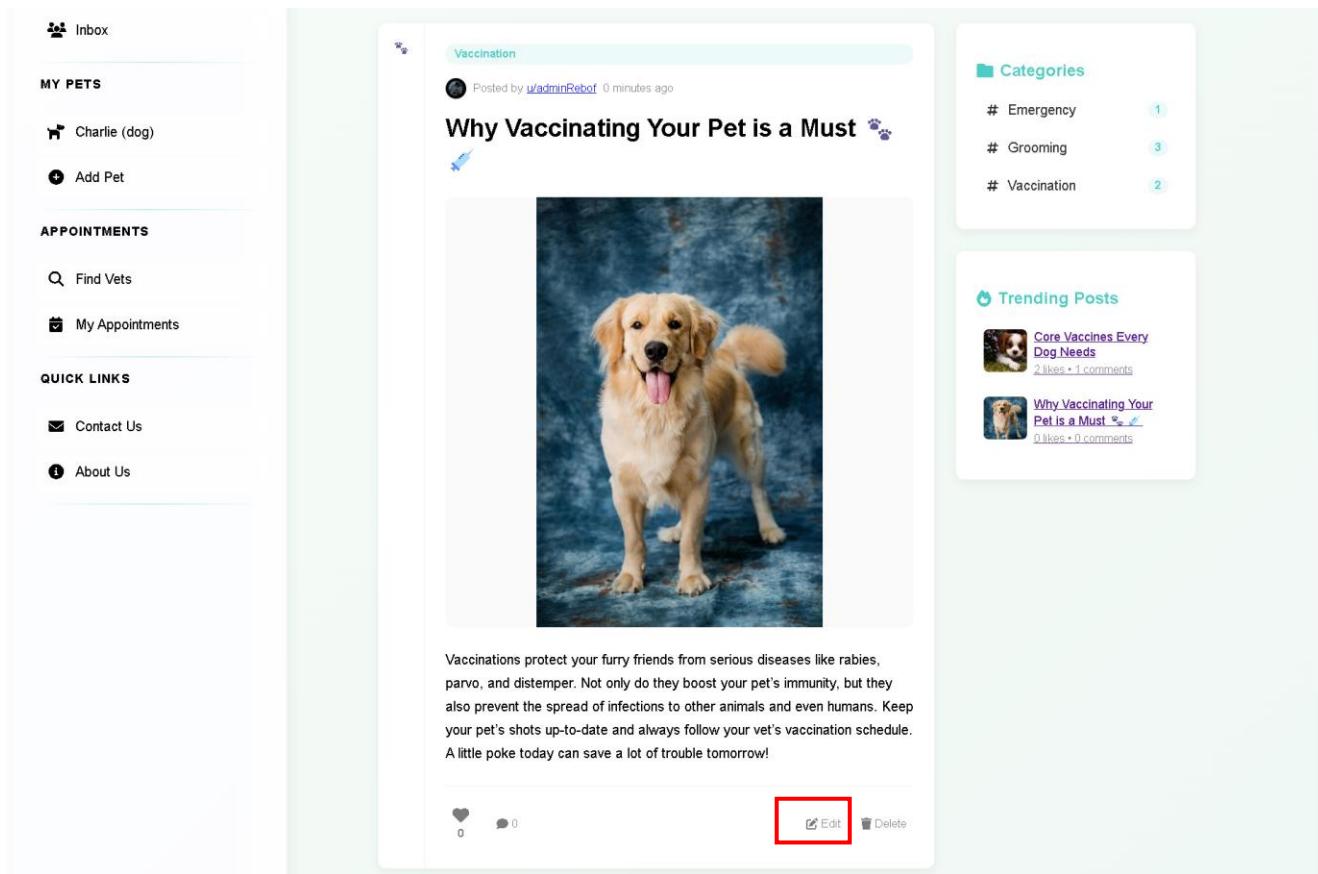


Figure 311: Selecting the edit option

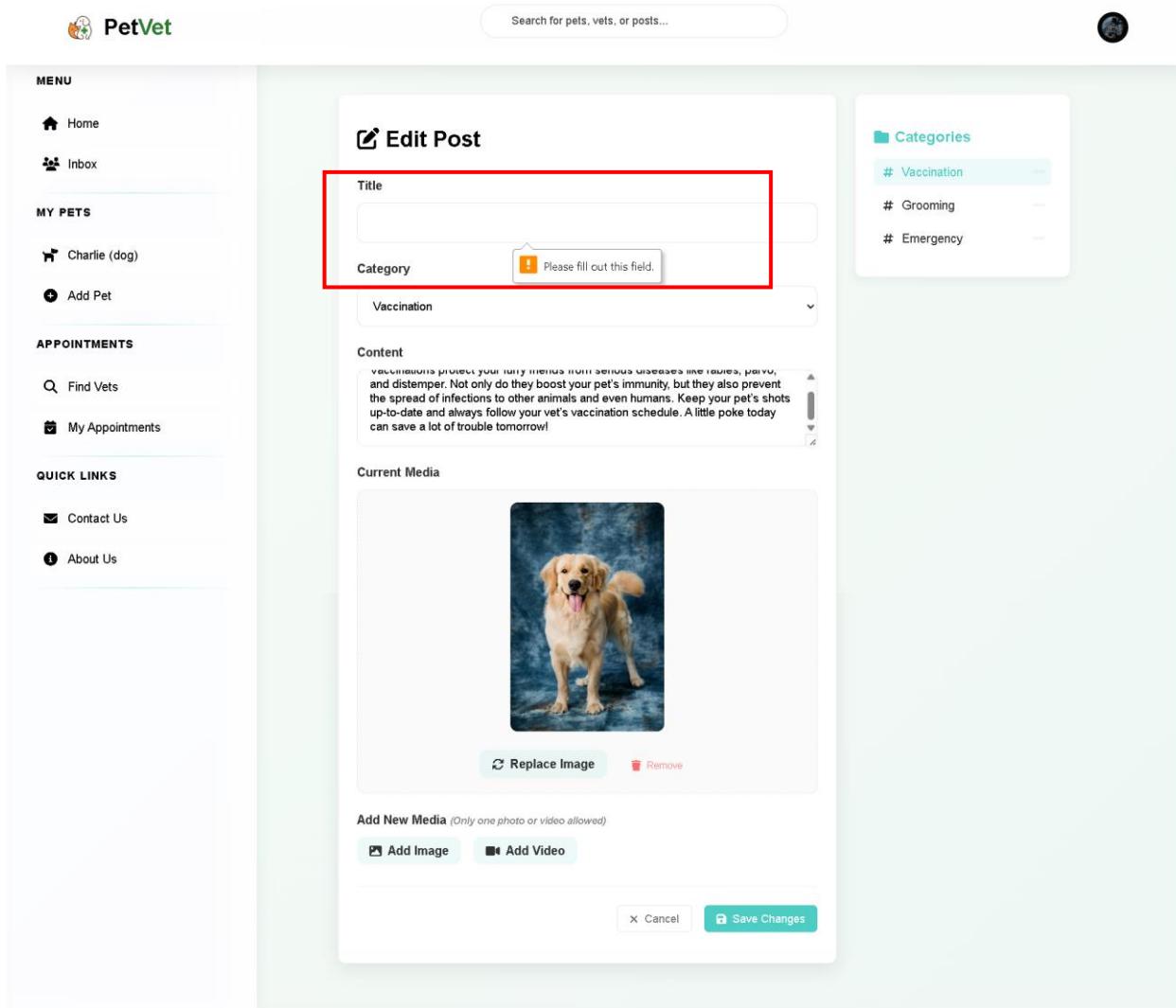


Figure 312: Leaving the title empty in edit mode

MENU

- Home
- Inbox

MY PETS

- Charlie (dog)
- Add Pet

APPOINTMENTS

- Find Vets
- My Appointments

QUICK LINKS

- Contact Us
- About Us

Categories

- Vaccination
- Grooming
- Emergency

Edit Post

Title

Protect Your Pet: The Power of Vaccination 🐶

Category

Vaccination

Content

VACCINATIONS PROTECT YOUR PET FROM SERIOUS DISEASES LIKE RABIES, PARVO, AND DISTEMPER. NOT ONLY DO THEY BOOST YOUR PET'S IMMUNITY, BUT THEY ALSO PREVENT THE SPREAD OF INFECTIONS TO OTHER ANIMALS AND EVEN HUMANS. KEEP YOUR PET'S SHOTS UP-TO-DATE AND ALWAYS FOLLOW YOUR VET'S VACCINATION SCHEDULE. A LITTLE POKES TODAY CAN SAVE A LOT OF TROUBLE TOMORROW!

Current Media

Replace Image **Remove**

Add New Media (Only one photo or video allowed)

Add Image **Add Video**

Cancel **Save Changes**

Figure 313: Updating the title

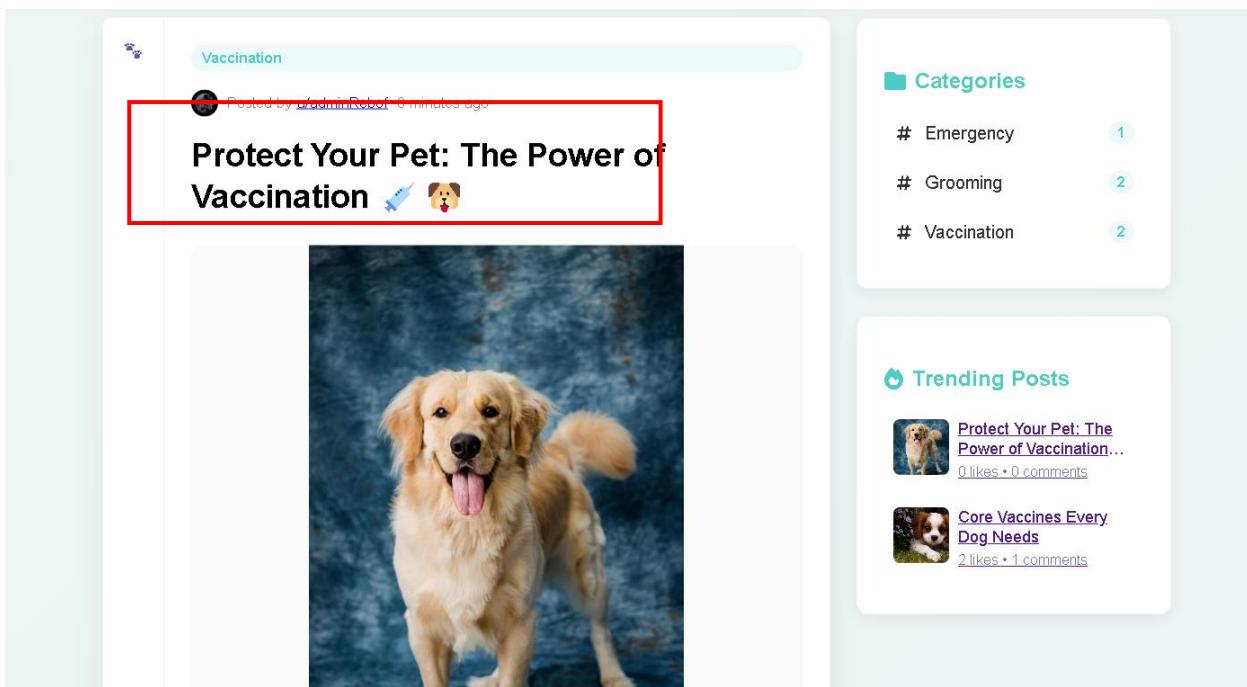


Figure 314: Successfully Updated the post title

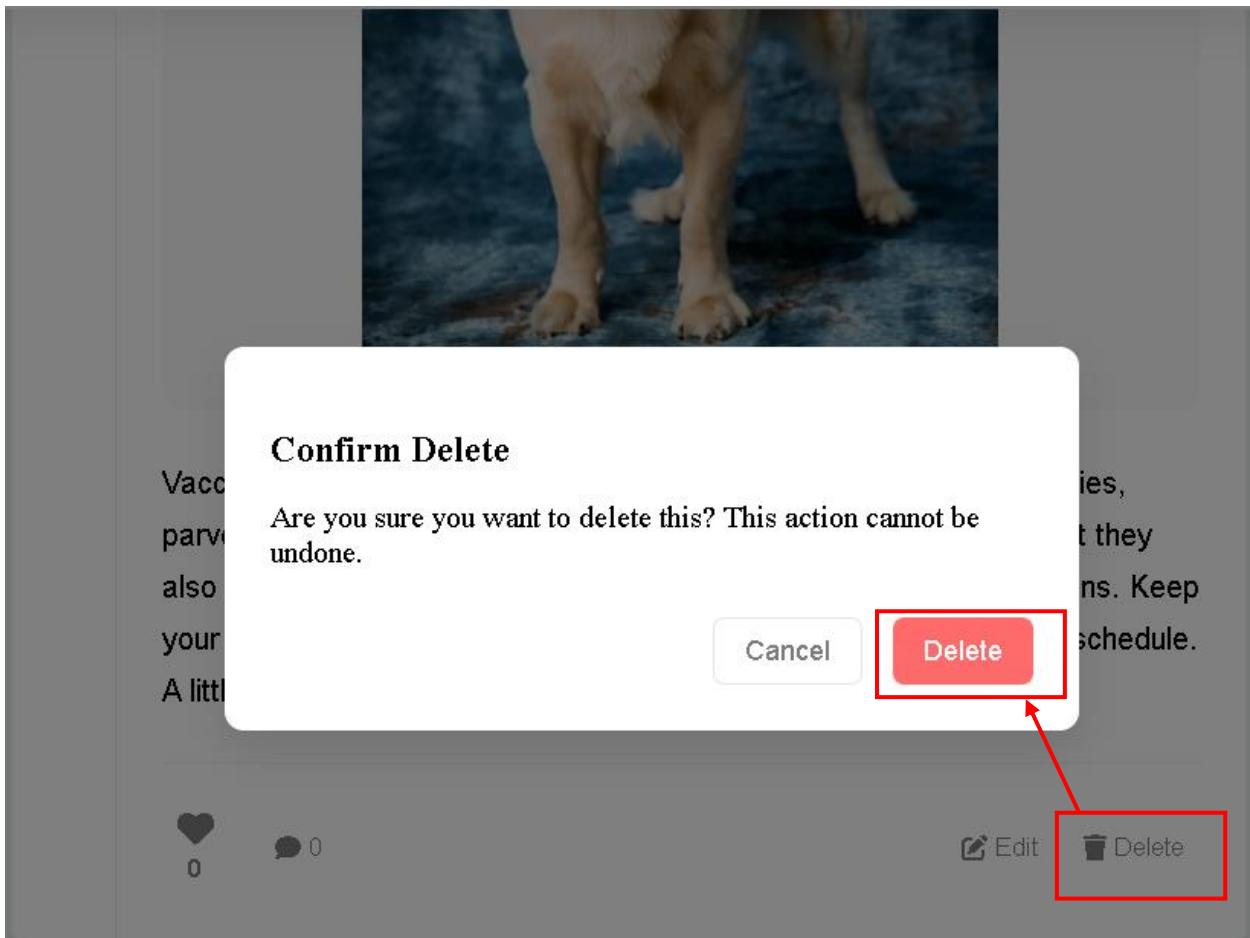


Figure 315: Delete option has been clicked

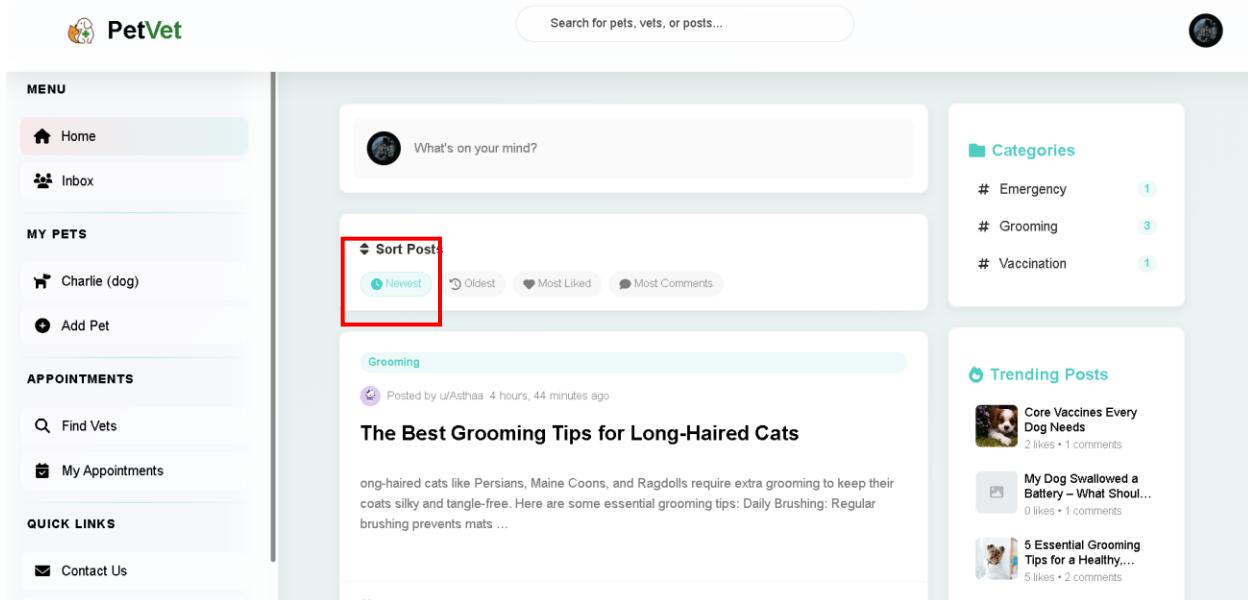


Figure 316: Post has been deleted as newest post is different

4.3.16 System Test: Admin1 Administrator Login Functionality

Test no.	Admin1
Objective	To verify that only users with admin privileges can access the Django admin dashboard.
Testcase	<ol style="list-style-type: none"> 1. Regular User Attempting to Access Admin Page 2. Admin User Accessing Admin Page Successfully
Action	<ol style="list-style-type: none"> 1. Regular User Attempt <ul style="list-style-type: none"> • Logged in with rebofkatwal8@gmail.com (a regular user) • Entered password and tried to access the admin dashboard • Received a message stating: “Invalid email or password or insufficient permission” 2. Admin User Accessing <ul style="list-style-type: none"> • Logged in with rebofkatwal7@gmail.com (an admin user) and correct password • Successfully accessed the admin dashboard

Expected Result	<ul style="list-style-type: none"> • Regular users should not be allowed to access admin panel • Admin users should be able to log in and view the dashboard
Actual Results	Admin panel access restricted properly based on user role.
Conclusion	The test was successful

Table 50: System Test - Administrator Login

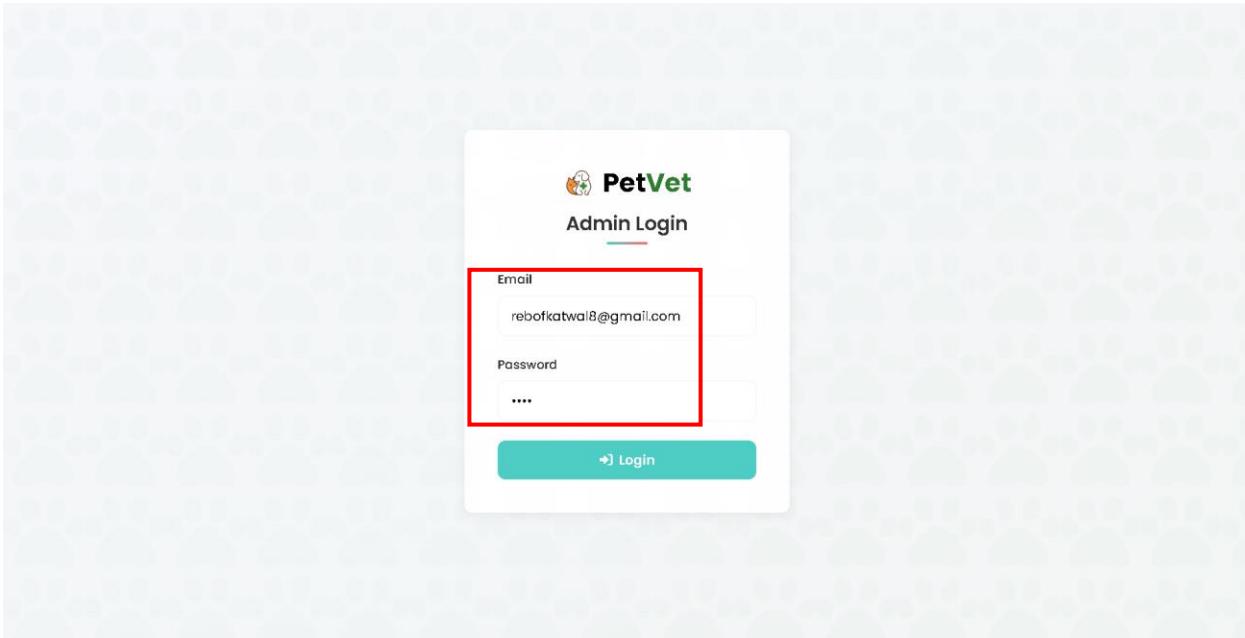


Figure 317: Entering invalid admin credentials

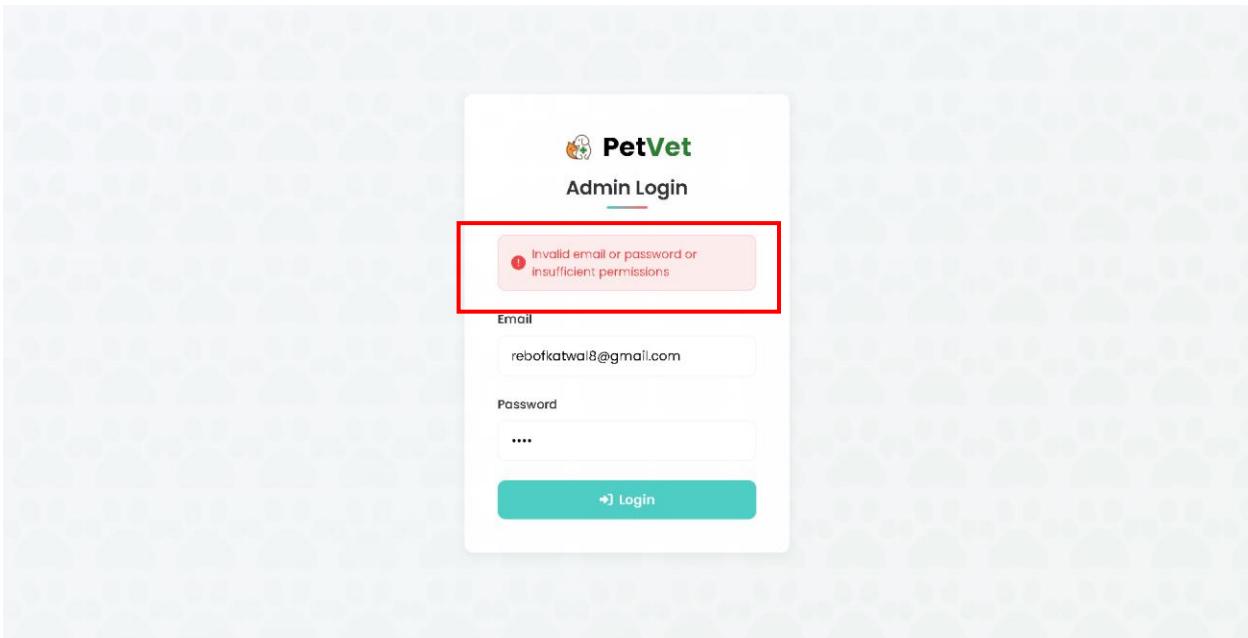


Figure 318: Authentication Error message displayed

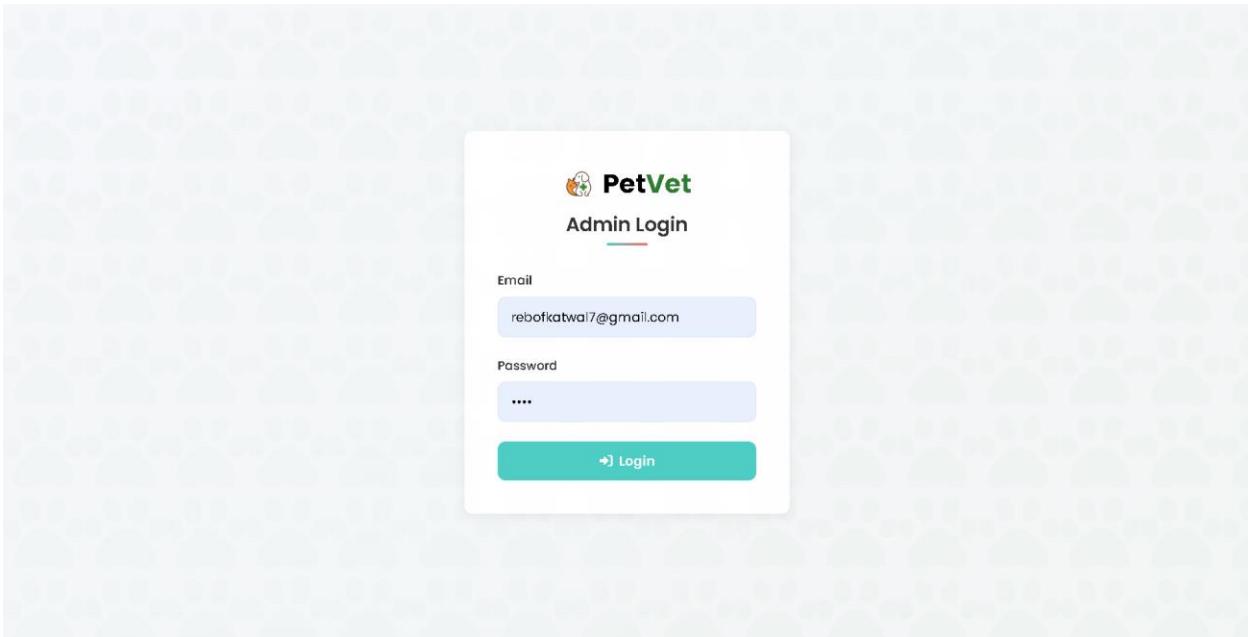


Figure 319: Entering correct credentials

The screenshot shows the PetVet Admin dashboard after successful login. The left sidebar includes links for User Management, Post Management, Vet Approvals, and Categories. The main dashboard features a summary section with icons and counts for Total Users (9), Veterinarians (3), Pet Owners (6), and Total Posts (5). Below this are two charts: a donut chart titled 'User Statistics' showing 60% Veterinarians and 40% Pet Owners, and a bar chart titled 'Posts by Category' showing Grooming (3), Emergency (1), and Vaccination (1). The 'Recent Posts' section lists five articles with titles like 'The Best Grooming Tips for Long-Haired Cats' and 'Core Vaccines Every Dog Needs'. The 'Pending Vet Approvals' section indicates no pending approvals.

Title	Author	Category	Date	Actions
The Best Grooming Tips for Long-Haired Cats	Asthaa	Grooming	Apr 25, 2025	View
Core Vaccines Every Dog Needs	Datta18	Vaccination	Apr 25, 2025	View
How to Keep Your Cat's Coat Smooth and Tangle-Free	Asthaa	Grooming	Apr 25, 2025	View
5 Essential Grooming Tips for a Healthy, Happy Dog	Reboffff	Grooming	Apr 25, 2025	View
My Dog Swallowed a Battery – What Should I Do?	Reboffff	Emergency	Apr 25, 2025	View

Name	Email	Clinic Name	Requested At	Actions
No pending vet approvals.				

[Logout](#)

Figure 320: Successfully logged into admin dashboard

4.3.17 System Test: Admin2 Category Management Functionality

Test no.	Admin2
Objective	To verify that category creation, update, and deletion work as expected through the admin dashboard, and that validation is properly enforced.
Testcase	<ol style="list-style-type: none"> 1. Adding Category with Missing Name 2. Successfully Adding a Category 3. Editing Category with Missing Name 4. Successfully Updating a Category 5. Deleting a Category
Action	<ol style="list-style-type: none"> 1. Adding Category with Missing Name <ul style="list-style-type: none"> • Navigated to the Category management section in the admin dashboard • Tried adding a category without entering a name — got prompted to "Please fill out this field." 2. Successful Category Addition <ul style="list-style-type: none"> • Entered name: "Wildlife", description: "wild animal"

	<p>related" — category added successfully</p> <ul style="list-style-type: none"> • Verified the new category is visible in the home page's category section <ol style="list-style-type: none"> 3. Tried editing the category and left the name field empty — again prompted to fill it 4. Updated the name to "Wild Life" — category updated successfully 5. Clicked the Delete button, got a confirmation prompt, confirmed — category deleted successfully
Expected Result	<ul style="list-style-type: none"> • Admin should not be able to submit without a category name • Successful additions/edits should reflect on the site • Category should be deleted from both admin and frontend view
Actual Results	Category CRUD operations and validations working correctly in admin panel.
Conclusion	The test was successful

Table 51: System Test - Category Management

The screenshot shows the 'Category Management' section of the PetVet Admin interface. On the left, a sidebar menu includes 'Dashboard', 'User Management', 'Post Management', 'Vet Approvals', and 'Categories'. The 'Categories' item is highlighted with a pink background. The main content area has a header 'Category Management'. On the left, a form titled 'Add New Category' has a red box around the 'Category Name' input field, which is empty. A tooltip says 'Please fill out this field.' Below it is a 'Description' input field containing 'Wild animals related'. At the bottom of the form is a teal 'Add Category' button. To the right is a table titled 'Categories' listing three existing categories: 'Emergency', 'Grooming', and 'Vaccination'. Each row has 'Edit' and 'Delete' buttons.

Figure 321: Leaving the category title field empty

This screenshot shows the same 'Category Management' page after the missing data has been entered. The 'Category Name' input field now contains the value 'Wildlife', which is highlighted with a blue background. The 'Description' input field still contains 'Wild animals related'. The rest of the page, including the table of existing categories, remains the same as in Figure 321.

Figure 322: Entering the missing data

The screenshot shows the PetVet Admin interface under the 'Category Management' section. On the left sidebar, 'Categories' is selected. A success message at the top states 'Category 'Wildlife' has been added successfully.' The main area displays a table of categories with columns: Name, Slug, Description, Post Count, and Actions (Edit, Delete). A new row for 'Wildlife' is highlighted with a red border, showing it was added successfully. The table data is as follows:

Name	Slug	Description	Post Count	Actions
Emergency	emergency-fo	Urgent medical attention	1	<button>Edit</button> <button>Delete</button>
Wildlife	wildlife-ha	Wild animals related	0	<button>Edit</button> <button>Delete</button>
Grooming	grooming-hc	Bathing, trimming, and styling	3	<button>Edit</button> <button>Delete</button>
Vaccination	vaccination-xs	Pet immunization and booster shots	1	<button>Edit</button> <button>Delete</button>

Figure 323: Added a new category

The screenshot shows the PetVet home page. The left sidebar includes sections for 'MENU' (Home, Inbox), 'MY PETS' (No pets added yet, Add Pet), 'APPOINTMENTS' (Find Vets, My Appointments), and 'QUICK LINKS' (Contact Us, About Us). The main content area features a search bar, a central post area with sorting options (Newest, Oldest, Most Liked, Most Comments), and a sidebar on the right. The sidebar includes a 'Categories' section with a red box around the 'Wildlife' category, which has 0 posts. Other categories listed are Emergency (1 post), Grooming (2 posts), and Vaccination (1 post). Below this is a 'Trending Posts' section.

Figure 324: Showing the new category in the home page

Edit Category

Category Name

Description



Please fill out this field.

Wild animals related

Cancel

Save Changes

Figure 325: Leaving category name field empty while updating

Edit Category

Category Name

Wild Life

Description

Wild animals related

Cancel

Save Changes

Figure 326: Updating the category name field

The screenshot shows the PetVet Admin interface. On the left, there's a sidebar with links: Dashboard, User Management, Post Management, Vet Approvals, and Categories (which is highlighted). The main area is titled "Category Management". A success message at the top says "Category 'Wild Life' has been updated successfully." Below it, there's a form to "Add New Category" with fields for "Category Name" and "Description", and a "Add Category" button. To the right is a table titled "Categories" with columns: Name, Slug, Description, Post Count, and Actions (Edit and Delete buttons). The table contains four rows: Emergency (slug: emergency-fo), Wild Life (slug: wildlife-ha), Grooming (slug: grooming-hc), and Vaccination (slug: vaccination-xs). The "Wild Life" row is highlighted with a red box.

Figure 327: Updated the category name successfully

This screenshot shows a modal dialog box titled "Confirm Deletion" asking if the user is sure they want to delete the category "Wild Life". It states that posts in this category will be set to uncategorized. The "Delete" button in the dialog is highlighted with a red box. In the background, the "Categories" table is visible, showing the same four categories as Figure 327, with the "Wild Life" row being the one targeted for deletion.

Figure 328: Deleting the category

The screenshot shows the PetVet Admin interface. On the left, a sidebar menu includes options like Dashboard, User Management, Post Management, Vet Approvals, and Categories (which is selected). The main area is titled "Category Management". A success message at the top states "Category 'Wild Life' has been deleted successfully." Below this, there's a form for "Add New Category" with fields for "Category Name" and "Description", and a "Add Category" button. To the right is a table titled "Categories" listing three items: Emergency, Grooming, and Vaccination. Each row has "Edit" and "Delete" buttons.

Name	Slug	Description	Post Count	Actions
Emergency	emergency-fo	Urgent medical attention	1	<button>Edit</button> <button>Delete</button>
Grooming	grooming-hc	Bathing, trimming, and styling	3	<button>Edit</button> <button>Delete</button>
Vaccination	vaccination-xs	Pet immunization and booster shots	1	<button>Edit</button> <button>Delete</button>

Figure 329: Successfully deleted the category

4.3.18 System Test: Admin3 Content Moderation Functionality

Test no.	Admin3
Objective	To ensure the post creation, editing, and deletion functionalities work correctly and validations are enforced during the process.
Testcase	<ol style="list-style-type: none"> 1. Post Creation with Missing Category 2. Successful Post Creation 3. Post Edit with Missing Title 4. Successful Post Update 5. Post Deletion with Confirmation
Action	<ol style="list-style-type: none"> 1. Post Creation with Missing Category <ul style="list-style-type: none"> • Navigated to the Category management section in the admin dashboard • Clicked on "What's on your mind?", which opened the create post modal • Entered title: "Why vaccinating our pet is a must" but left category empty • Prompted to fill the required category field

	<p>2. Successful Post Addition</p> <ul style="list-style-type: none"> • Filled in all fields properly, including a picture, and submitted the post • Redirected to the post detail page <p>3. Clicked edit post, left title empty → prompted to fill the title</p> <p>4. Updated title to: "Protect your pet: The power of vaccination" → successfully updated</p> <p>5. Successful Deletion</p> <ul style="list-style-type: none"> • Clicked delete, confirmation popup appeared • Confirmed and post was successfully deleted
Expected Result	<ul style="list-style-type: none"> • Fields must be validated before submission • Post creation, update, and deletion should reflect changes properly and redirect as expected
Actual Results	Post operations (create, edit, delete) performed correctly with appropriate validations and feedback.
Conclusion	The test was successful

Table 52: System Test - Content Moderation

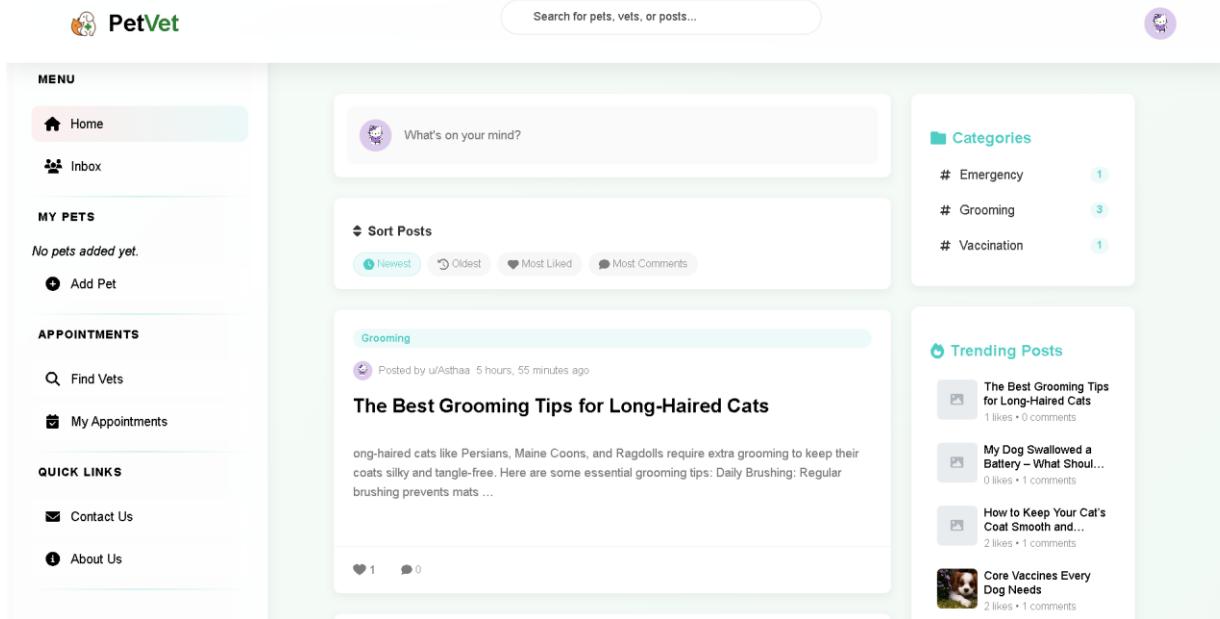


Figure 330: The first post is showing in the home page before content moderation

The screenshot shows the PetVet Admin interface. On the left, there's a sidebar with 'PetVet Admin' at the top, followed by 'Dashboard', 'User Management', 'Post Management' (which is highlighted in pink), 'Vet Approvals', 'Categories', and 'Logout'. The main content area is titled 'View Post' and shows the same 'The Best Grooming Tips for Long-Haired Cats' post from the previous screenshot. The post details are identical. At the bottom of the post view, there are two buttons: 'Delete Post' (in red) and 'Deactivate Post' (in orange). The 'Deactivate Post' button is highlighted with a red box. Below the post, there's a section for 'Comments (0)' with the note 'No comments yet.'

Figure 331: The post was selected for deactivation

The screenshot shows the PetVet Admin dashboard with the user 'adminRebof' logged in. The main content area is titled 'View Post' and displays a message: 'Post 'The Best Grooming Tips for Long-Haired Cats' has been deactivated successfully.' Below this message, the post title 'The Best Grooming Tips for Long-Haired Cats' is shown, along with its details: 'Posted by Asthaa on April 25, 2025 Category: Grooming Views: 0 Likes: 1'. The post content includes tips for grooming long-haired cats, such as daily brushing, trimming fur around paws, bathing, ear care, and keeping claws trimmed. At the bottom of the post view, there are two buttons: 'Delete Post' (red) and 'Activate Post' (green). A sidebar on the left lists navigation options: Dashboard, User Management, Post Management (which is currently selected), Vet Approvals, and Categories. A bottom footer bar includes a 'Logout' link.

Figure 332: The post has been deactivated

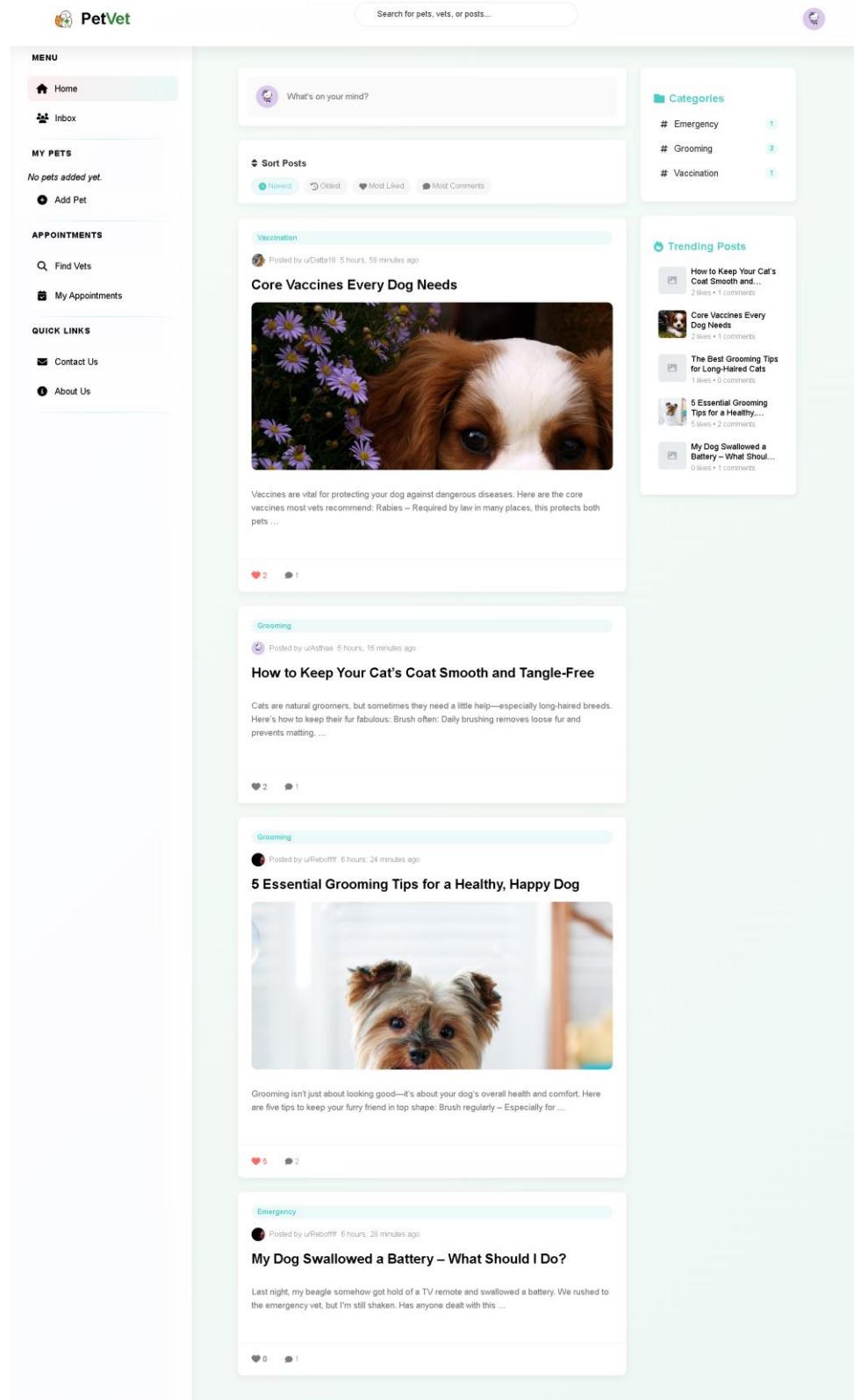


Figure 333: The post disappears from the home page

The screenshot displays the PetVet mobile application interface. At the top, there is a navigation bar with a search bar containing the placeholder "Search for pets, vets, or posts...". On the right side of the top bar is a purple circular icon with a white dog silhouette.

Left Sidebar (Menu):

- MENU**
 - [Home](#)
 - [Inbox](#)
- MY PETS**

No pets added yet.

[Add Pet](#)
- APPOINTMENTS**
 - [Find Vets](#)
 - [My Appointments](#)
- QUICK LINKS**
 - [Contact Us](#)
 - [About Us](#)

User Profile Section:

A profile card for "Asththa Thapa" is shown. It features a circular profile picture of Hello Kitty. Below the picture, the name "Asththa Thapa" is displayed, along with her status as a "Pet Owner" and her email address "asththa.thapa311@gmail.com" and phone number "9803637886". There is also a "Edit Profile" button.

Basic Information:

Full Name:	Asththa Thapa
Email:	asththa.thapa311@gmail.com
Phone:	9803637886
Gender:	Female

Profile Information:

Bio:	I love cats!
Pets Owned:	0

Address Information:

Country:	Nepal
City:	Kathmandu
Address:	Basundhara Chowk

Posts Section:

Two posts are listed:

- How to Keep Your Cat's Coat Smooth and Tangle-Free** (Grooming)

Cats are natural groomers, but sometimes they need a little help—especially long-haired breeds. Here's how to keep their fur fabulous: Brush often...

Apr 25, 2025 | 2 likes | 1 comments

[View](#) [Edit](#)
- The Best Grooming Tips for Long-Haired Cats** (Grooming)

long-haired cats like Persians, Maine Coons, and Ragdolls require extra grooming to keep their coats silky and tangle-free. Here are some essential gr...

Apr 25, 2025 | 1 likes | 0 comments

[View](#) [Edit](#)

Comments Section:

Three comments are shown under the first post:

- On Post: 5 Essential Grooming Tips for a Healthy, Happy Dog** (1 like)

Resourcefull

Apr 25, 2025 | 2 replies
- On Post: My Dog Swallowed a Battery – What Should I Do?** (0 likes)

Update?

Apr 25, 2025 | 0 replies
- On Post: Core Vaccines Every Dog Needs** (0 likes)

Noted :)

Apr 25, 2025 | 0 replies

Logout Button:

[Logout](#)

Figure 334: The post can be seen in the user's profile while deactivated

PetVet Admin

View Post

The Best Grooming Tips for Long-Haired Cats

Posted by Asthaa on April 25, 2025 Category: Grooming Views: 0 Likes: 1

long-haired cats like Persians, Maine Coons, and Ragdolls require extra grooming to keep their coats silky and tangle-free. Here are some essential grooming tips:

- Daily Brushing: Regular brushing prevents mats and tangles. Use a wide-toothed comb or slicker brush to gently detangle knots without pulling.
- Trim the Fur Around the Paws: Long hair around the paws can trap dirt and debris, causing irritation. Trim carefully or have a groomer do it.
- Bathing: Cats generally groom themselves, but occasionally, a bath is necessary—especially if they get into something sticky or smelly. Use cat-safe shampoo and warm water.
- Ear Care: Check your cat's ears for dirt and wax buildup. Wipe gently with a cotton ball soaked in a pet-safe ear cleaner.
- Keep Claws Trimmed: Long-haired cats often scratch more, so keep their claws trimmed to prevent damage to furniture or accidental scratches.

With consistent grooming, your long-haired cat will stay comfortable, clean, and looking beautiful.

Delete Post **Activate Post**

Comments (0)
No comments yet.

[Logout](#)

Figure 335: Activating the post

PetVet Admin

View Post

The Best Grooming Tips for Long-Haired Cats

Posted by Asthaa on April 25, 2025 Category: Grooming Views: 0 Likes: 1

long-haired cats like Persians, Maine Coons, and Ragdolls require extra grooming to keep their coats silky and tangle-free. Here are some essential grooming tips:

- Daily Brushing: Regular brushing prevents mats and tangles. Use a wide-toothed comb or slicker brush to gently detangle knots without pulling.
- Trim the Fur Around the Paws: Long hair around the paws can trap dirt and debris, causing irritation. Trim carefully or have a groomer do it.
- Bathing: Cats generally groom themselves, but occasionally, a bath is necessary—especially if they get into something sticky or smelly. Use cat-safe shampoo and warm water.
- Ear Care: Check your cat's ears for dirt and wax buildup. Wipe gently with a cotton ball soaked in a pet-safe ear cleaner.
- Keep Claws Trimmed: Long-haired cats often scratch more, so keep their claws trimmed to prevent damage to furniture or accidental scratches.

With consistent grooming, your long-haired cat will stay comfortable, clean, and looking beautiful.

Delete Post

Comments (0)
No comments yet.

[Logout](#)

Figure 336: The post has been activated

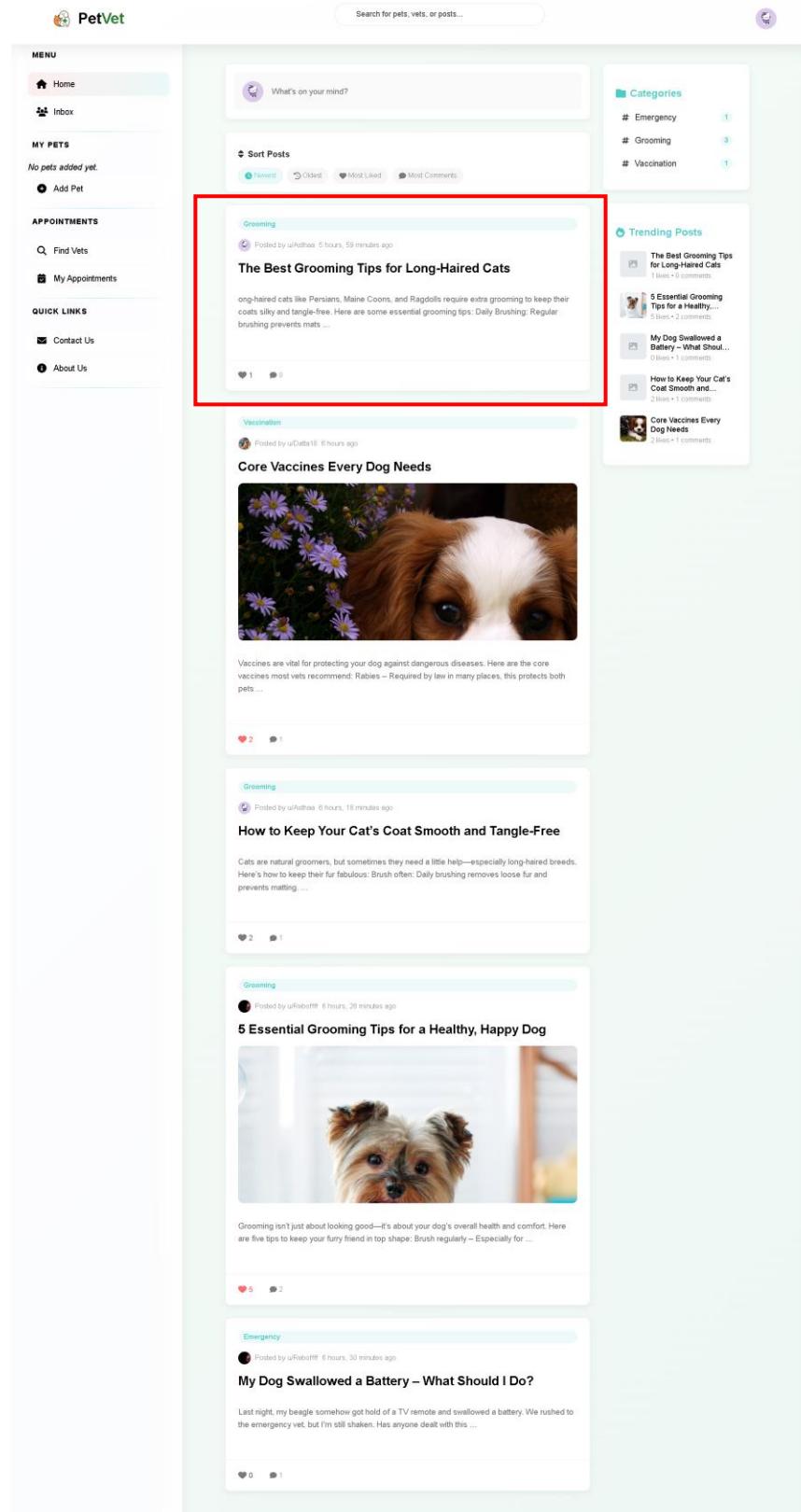


Figure 337: The post can be seen in the home page after activation

414

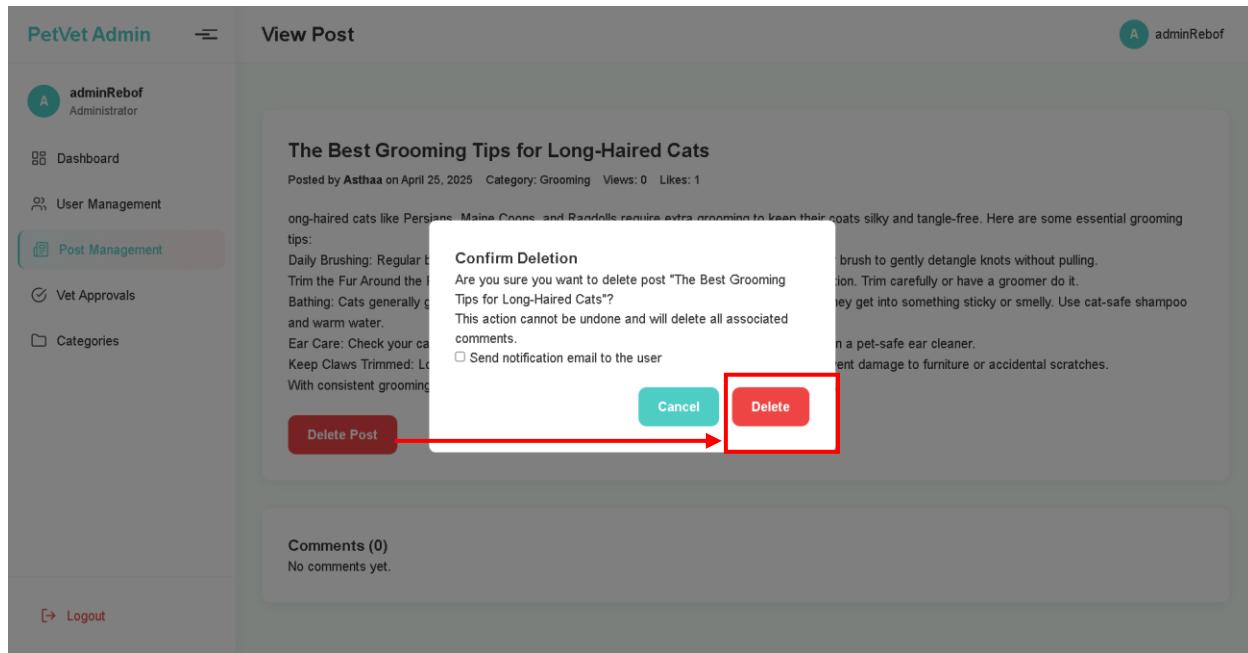


Figure 338: Deleting the Post

Title	Author	Category	Date	Likes	Status	Actions
Core Vaccines Every Dog Needs	Datta18	Vaccination	Apr 25, 2025	2	Active	View Delete
How to Keep Your Cat's Coat Smooth and Tangle-Free	Asthaa	Grooming	Apr 25, 2025	2	Active	View Delete
5 Essential Grooming Tips for a Healthy, Happy Dog	Rebofff	Grooming	Apr 25, 2025	5	Active	View Delete
My Dog Swallowed a Battery – What Should I Do?	Rebofff	Emergency	Apr 25, 2025	0	Active	View Delete

Figure 339: The post has been deleted

The screenshot displays the PetVet application interface. On the left, a sidebar contains a navigation menu with options like Home, Inbox, Add Pet, Find Vets, My Appointments, Contact Us, and About Us. The main area shows a user profile for "Asththa Thapa" with a Hello Kitty profile picture. The profile includes basic information (Full Name: Astha Thapa, Email: asthathapa311@gmail.com, Phone: 9803637886, Gender: Female), profile information (Bio: I love cats!, Pets Owned: 0), and address information (Country: Nepal, City: Kathmandu, Address: Basundhara Chowk). Below the profile is a post titled "How to Keep Your Cat's Coat Smooth and Tangle-Free" under the Grooming category. The post was created on April 25, 2025, has 2 likes, and 1 comment. Buttons for View and Edit are present. At the bottom of the screen, there is a "Logout" button.

Figure 340: After admin deletes the post, it completely disappears

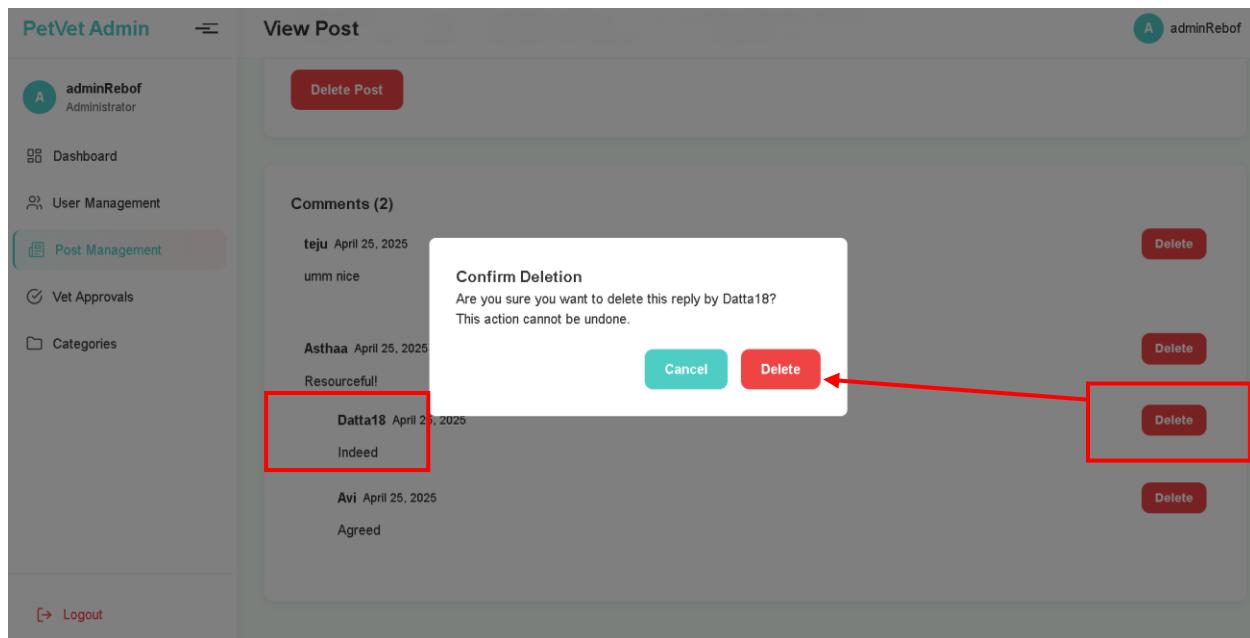


Figure 341: Admin deleting a reply

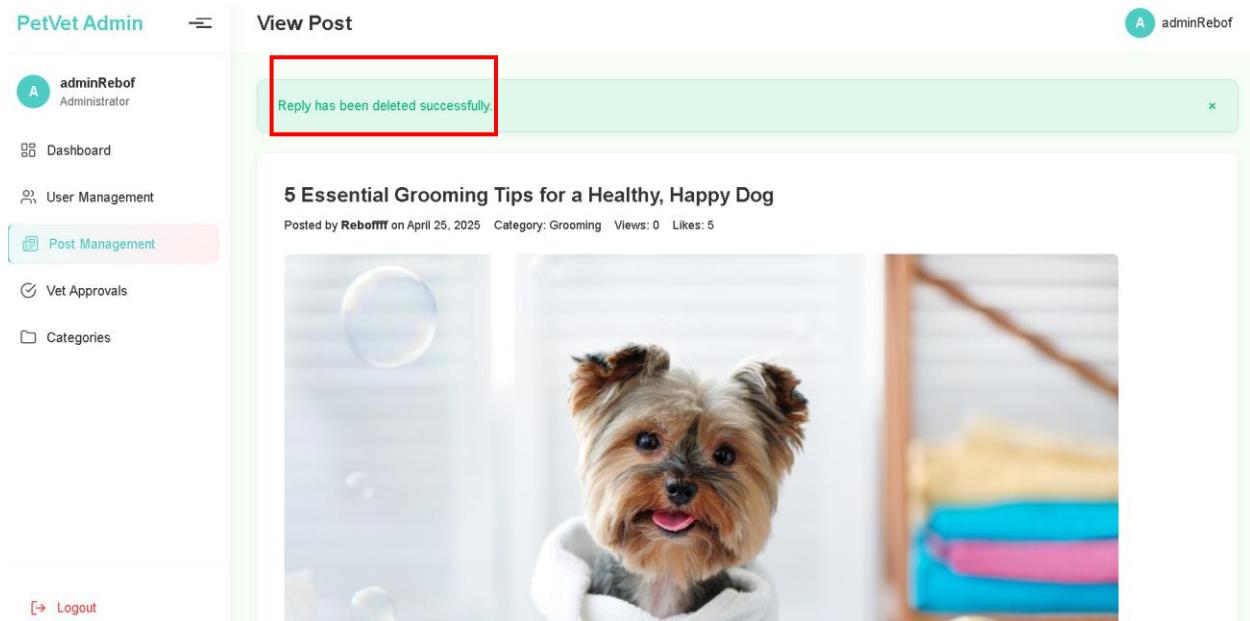


Figure 342: Reply was successfully deleted

The image shows a social media interface with a light blue header bar. Below it, a white rectangular area contains a comment section. At the top left of this area, there is a dark gray speech bubble icon followed by the text "2 Comments".

The first comment is from a user named "u/Asthaa" (represented by a purple profile picture with a white flower icon) posted 6 hours, 20 minutes ago. The comment text is "Resourcefull". It has a dark gray heart icon with the number "1" below it, and three interaction buttons: "Reply", "Share", and "Delete".

The second comment is from a user named "u/Avi" (represented by a brown profile picture with a white flower icon) posted 5 hours, 54 minutes ago. The comment text is "Agreed".

Below these two comments, there is a third comment from a user named "u/teju" (represented by a purple profile picture with a yellow flower icon) posted 5 hours, 50 minutes ago. The comment text is "umm nice". This comment also has a dark gray heart icon with the number "0" below it, and "Reply", "Share" buttons.

The entire white area is enclosed in a light blue border, which is itself inside a larger white background.

Figure 343: Shubham's reply is nowhere to seen

4.3.19 System Test: Utils1 Password Change Functionality

Test no.	Utils1
Objective	To verify that the password change functionality works correctly by validating the current password and ensuring strong new password compliance.
Testcase	<ol style="list-style-type: none"> 1. Incorrect Current Password 2. Weak New Password 3. Successful Password Change and Login
Action	<ol style="list-style-type: none"> 1. Attempted to change password with incorrect current password and a valid new password → received "Old password is incorrect" message 2. Entered correct current password rebof123 and a new password root → received errors "Password is too short" and "Password is too common" 3. Successful Password Change <ul style="list-style-type: none"> • Entered correct current password rebof123 and a strong new password Rebof123! → password was successfully updated

	<ul style="list-style-type: none"> Logged out and logged back in using new password Rebof123! → login successful
Expected Result	<ul style="list-style-type: none"> System should reject incorrect old passwords Weak passwords should trigger validation errors Strong passwords should be accepted and updated successfully
Actual Results	Password change validation and update flow worked as expected with accurate error messages and successful authentication.
Conclusion	The test was successful

Table 53: System Test - Password Change

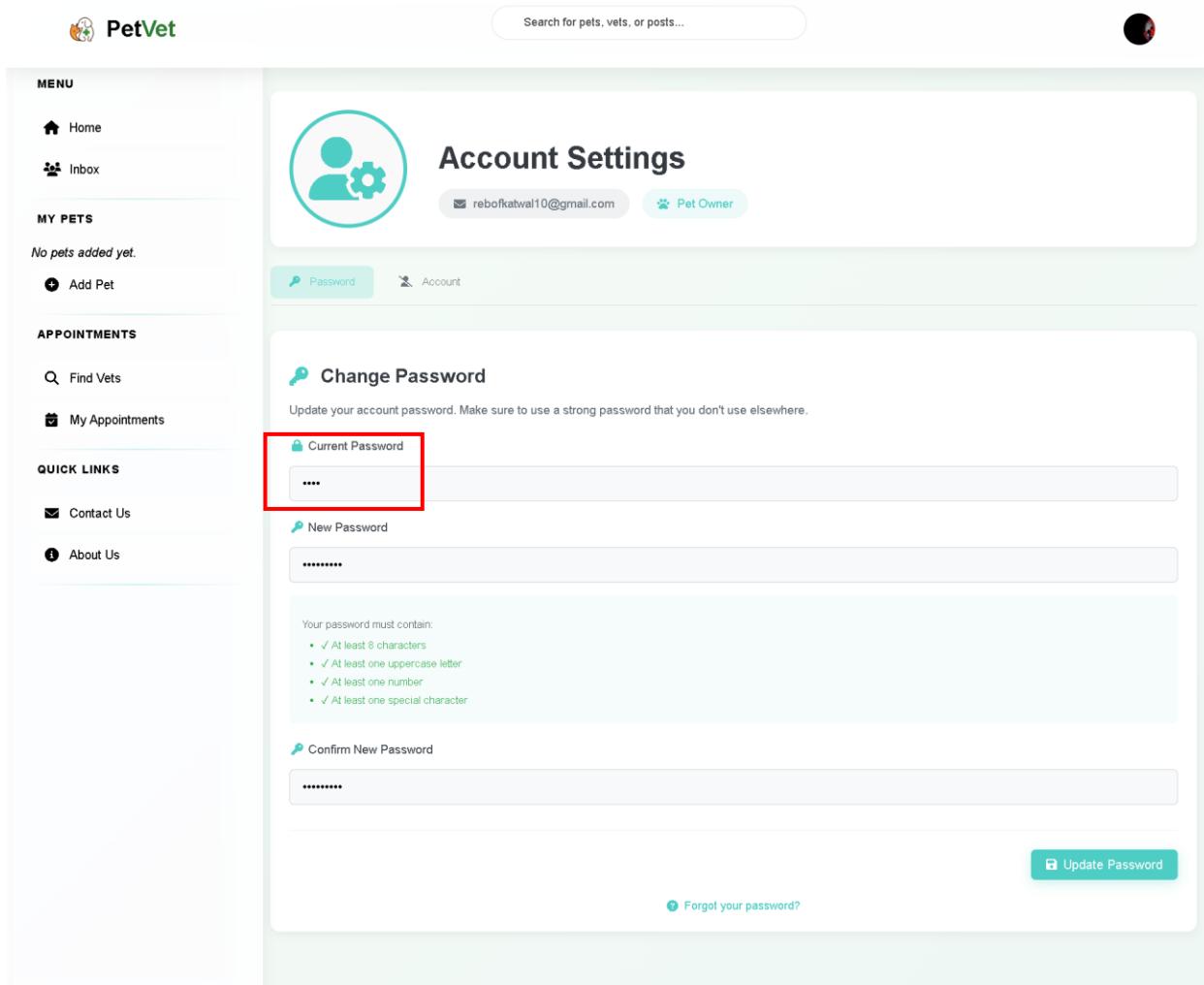


Figure 344: Wrong current password

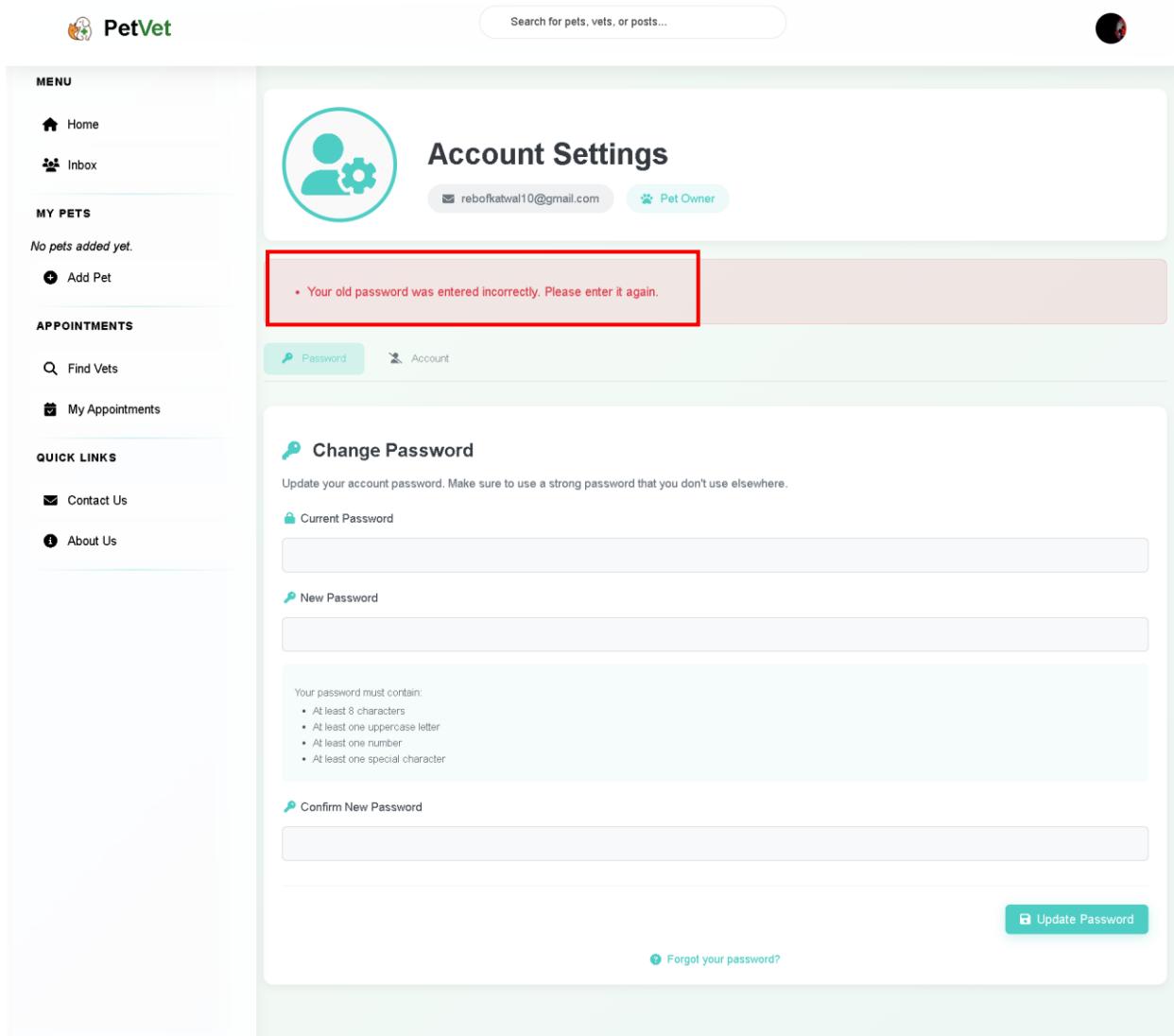


Figure 345: Entering the wrong current password

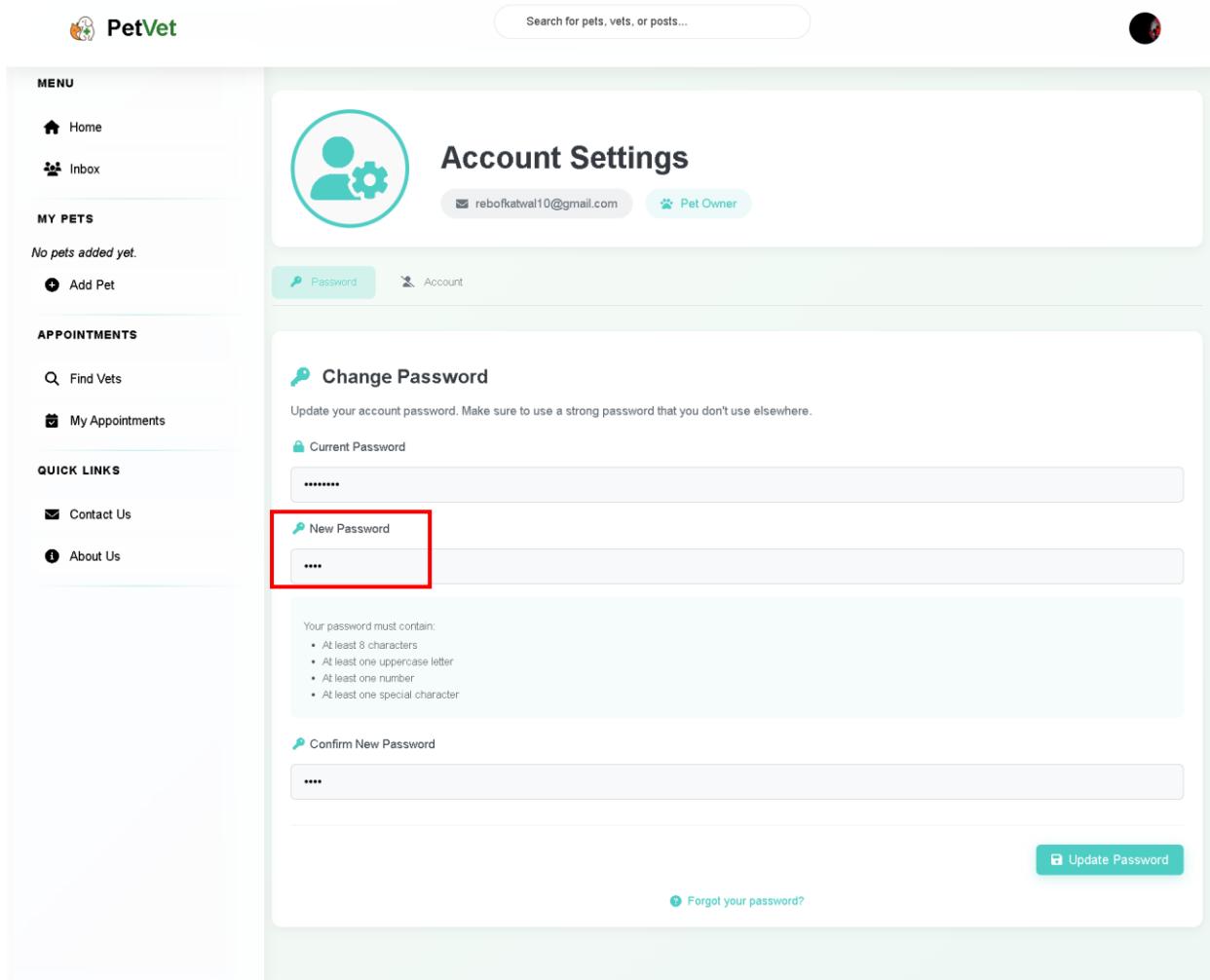


Figure 346: Entering a short new password

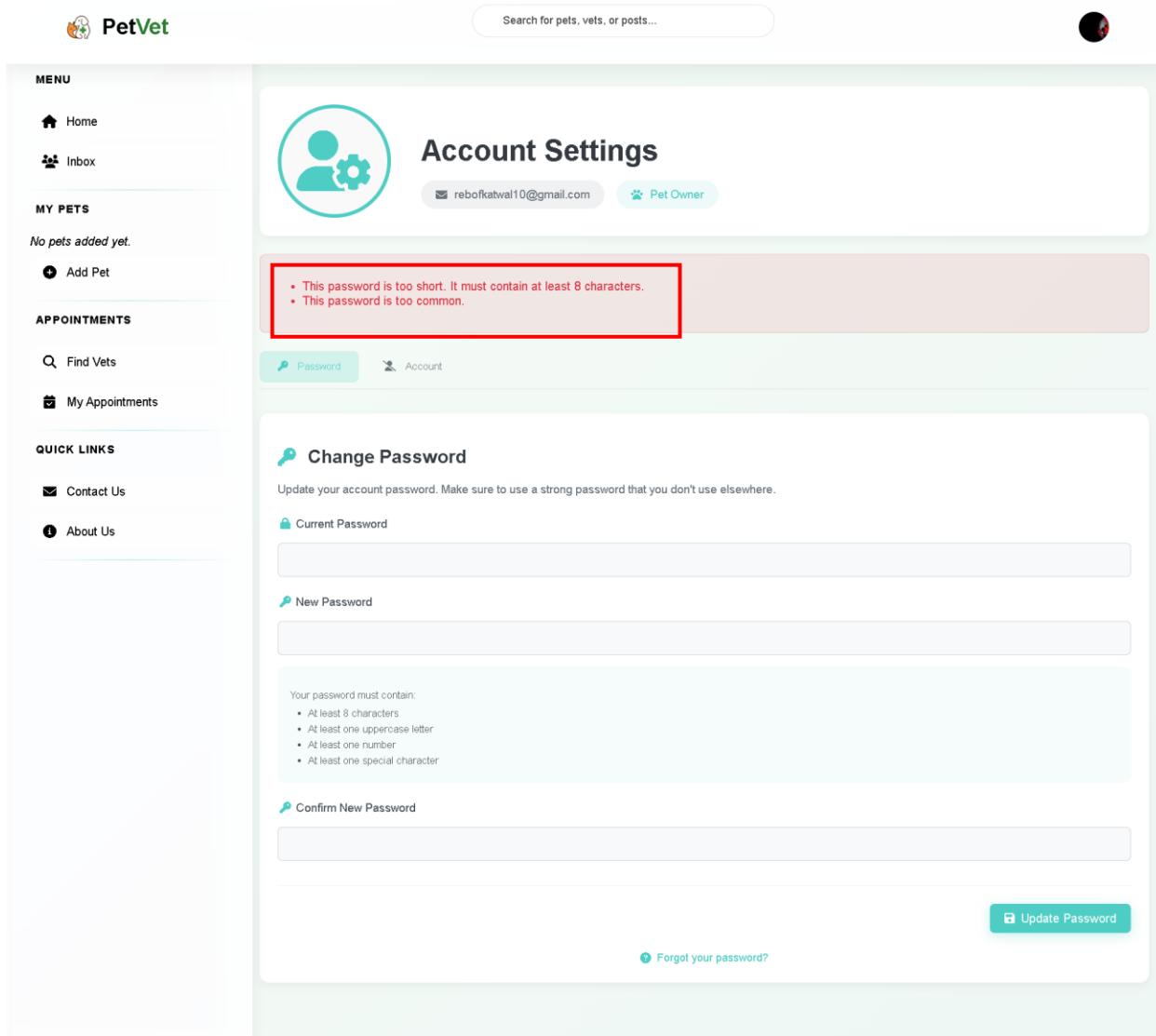


Figure 347: Error message displaying the password is short and common

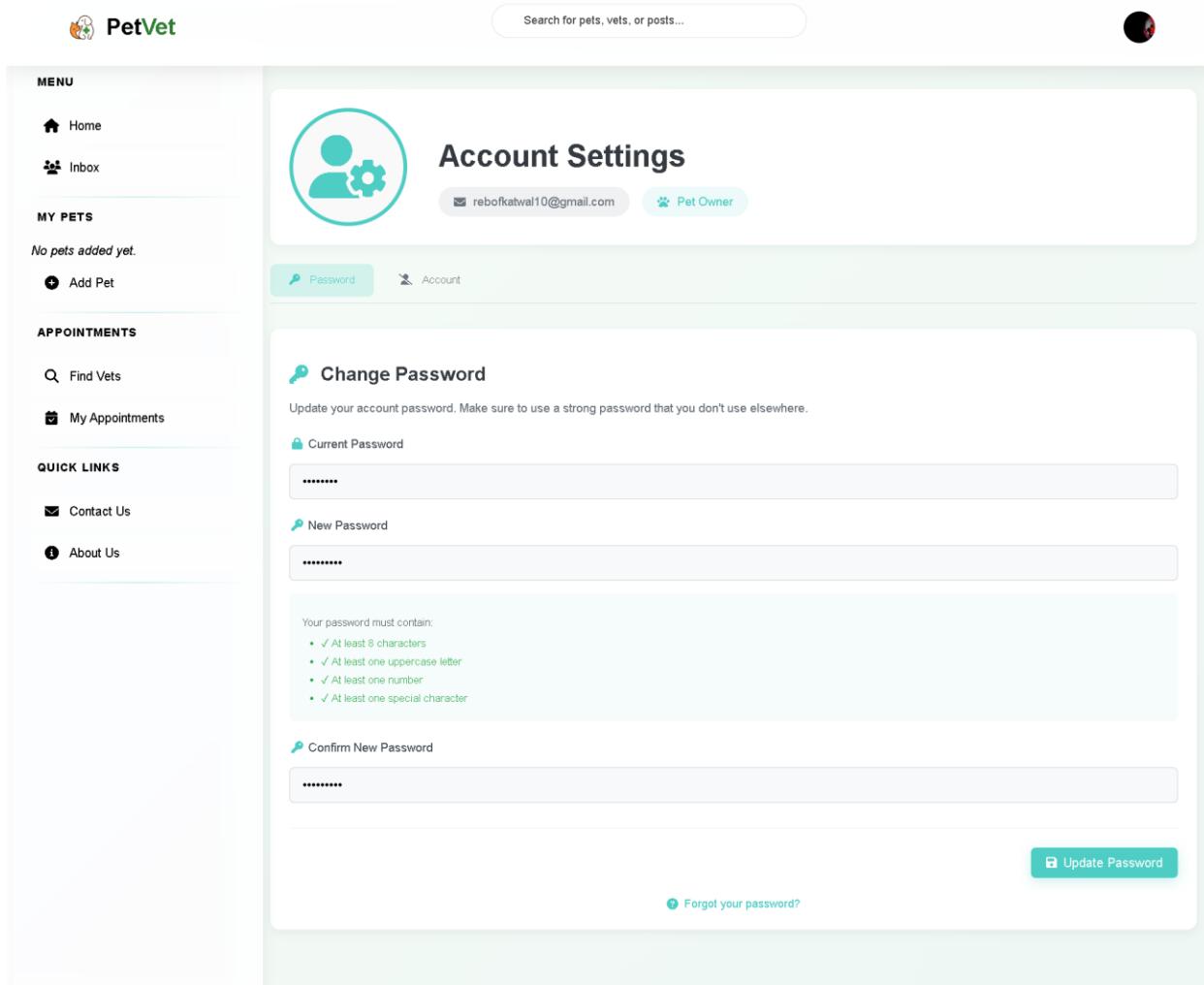


Figure 348: Entering a valid new password and correct current password

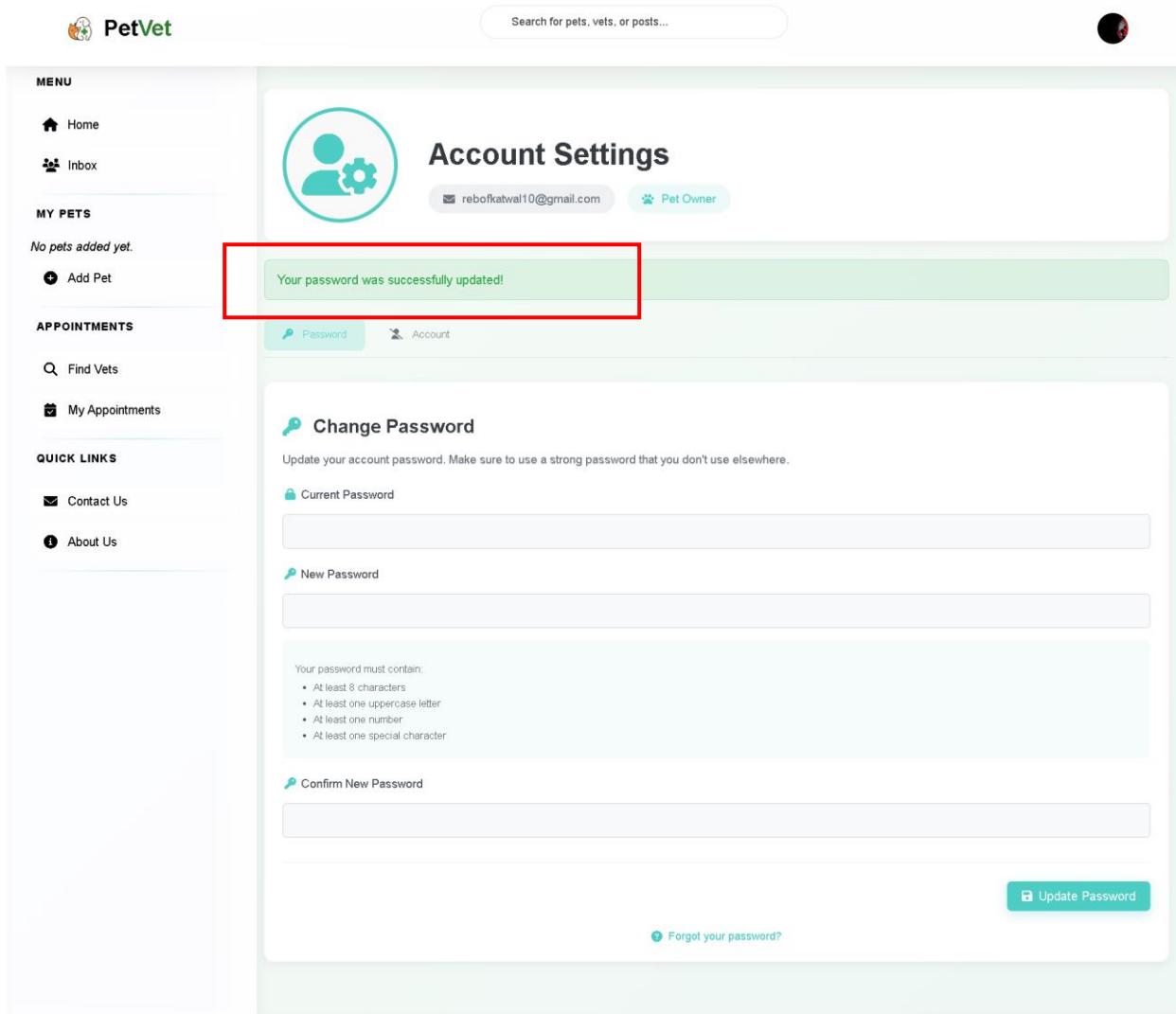


Figure 349: The password was successfully updated

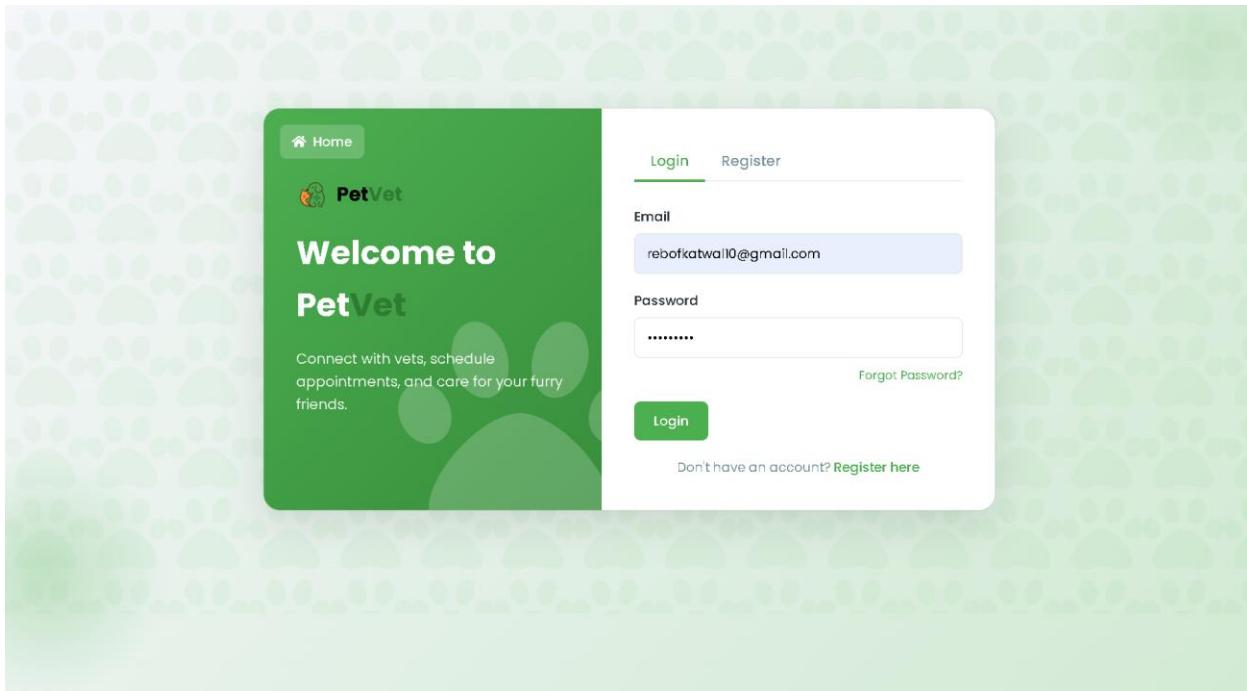


Figure 350: Logging in with the new changed password

Figure 351: Logged in with the new password

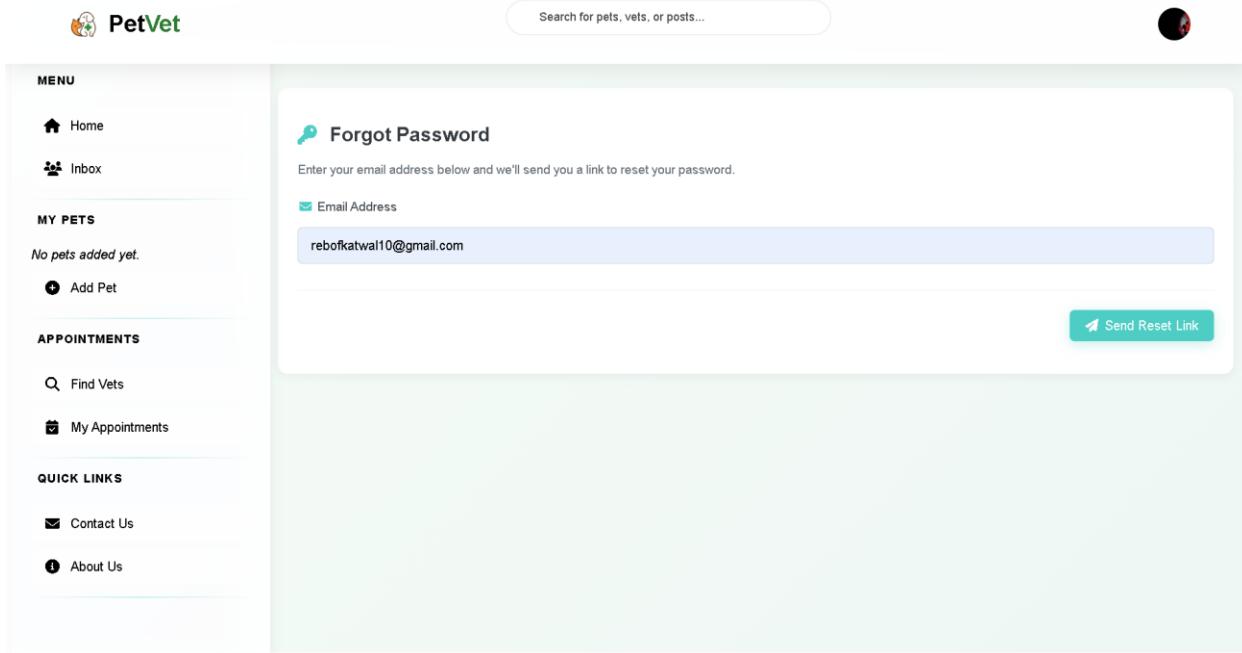
4.3.20 System Test: Utils2 Forgot Password Functionality

Test no.	Utils2
Objective	To verify that the "Forgot Password" and "Reset Password" features work correctly, with proper email validation, password rules, and secure token usage.
Testcase	<ol style="list-style-type: none"> 1. Forgot Password Email Flow 2. Reset with Weak Password 3. Reset with Mismatched Passwords 4. Successful Password Reset 5. Expired/Used Token Handling
Action	<ol style="list-style-type: none"> 1. Forgot Password Flow <ul style="list-style-type: none"> • Clicked "Forgot Password", entered rebofkatwal10@gmail.com → reset link sent via email • Clicked the link, redirected to Reset Password page 2. Entered root as the new password → prompted to use a longer password 3. Entered mismatched passwords (Katwal123 and Katwal123!) → prompted with "Passwords do not match"

	4. Entered matching strong passwords (Katwal123!) → successful reset and logged out 5. Clicked the same reset link again → redirected to login with “Invalid or expired token” message
Expected Result	<ul style="list-style-type: none"> • Reset link should be sent only if email exists • Weak or mismatched passwords should be rejected • Strong, matching passwords should reset account successfully • Expired or reused token should redirect to login with error
Actual Results	All flows, validations, and token-based restrictions worked as expected.
Conclusion	The test was successful

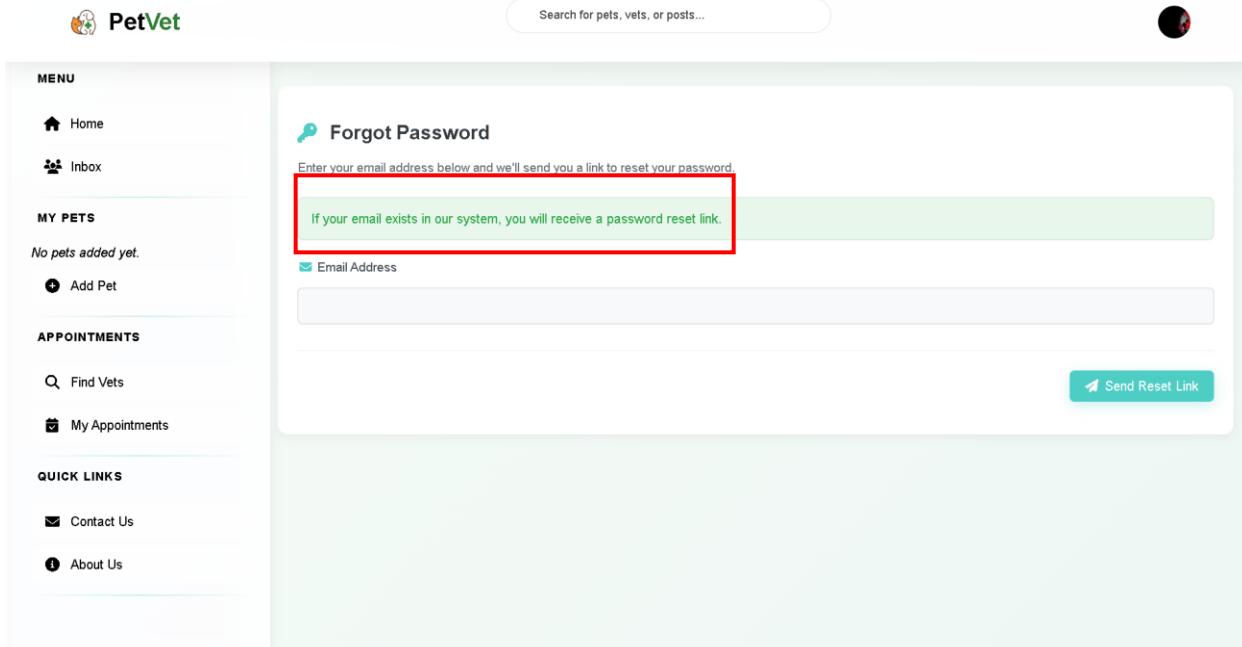
Table 54: System Test - Forgot Password

Note: Same flow achieved by triggering "Forgot Password" from login without being authenticated



The screenshot shows the PetVet website's 'Forgot Password' page. On the left, there is a sidebar with a 'MENU' section containing links for Home, Inbox, Add Pet, Find Vets, My Appointments, Contact Us, and About Us. Below this is a 'QUICK LINKS' section with similar links. The main content area has a search bar at the top. It displays a 'Forgot Password' form with a placeholder 'Enter your email address below and we'll send you a link to reset your password.' A text input field contains the email 'rebofkatwal10@gmail.com'. A teal button labeled 'Send Reset Link' is located to the right of the input field.

Figure 352: Entering the email for forgot password



This screenshot shows the same 'Forgot Password' page as Figure 352, but with a green success message box in the center stating 'If your email exists in our system, you will receive a password reset link.' This message is enclosed in a red rectangular box. The rest of the interface is identical to Figure 352, including the sidebar and the 'Send Reset Link' button.

Figure 353: Email will be sent to the email if it exists in the system

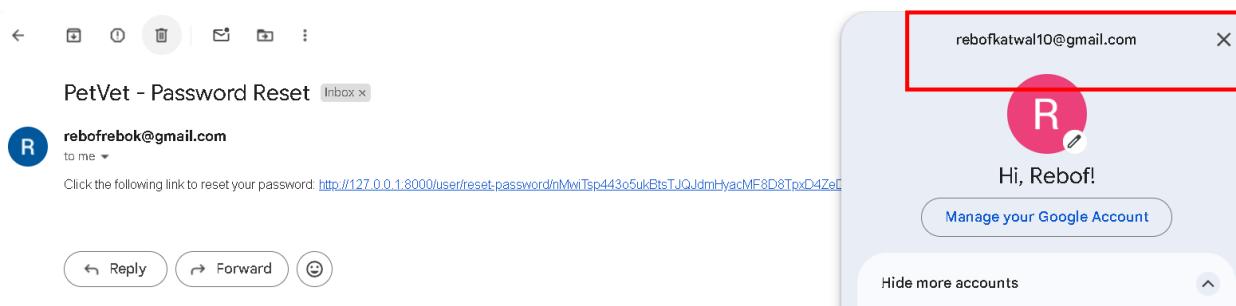


Figure 354: The link was sent to the email

The screenshot shows the PetVet mobile application. On the left, a sidebar menu includes 'Home', 'Inbox', 'Add Pet', 'Find Vets', 'My Appointments', and 'About Us'. The main content area is titled 'Reset Password' and contains instructions: 'Create a new password for your account. Make sure to use a strong password that you don't use elsewhere.' Below this are two input fields: 'New Password' and 'Confirm New Password'. A note below the first field specifies password requirements: 'Your password must contain:

- At least 8 characters
- At least one uppercase letter
- At least one number
- At least one special character

'. A green 'Reset Password' button is located at the bottom right of the form.

Figure 355: Clicking the link will lead to a reset password page

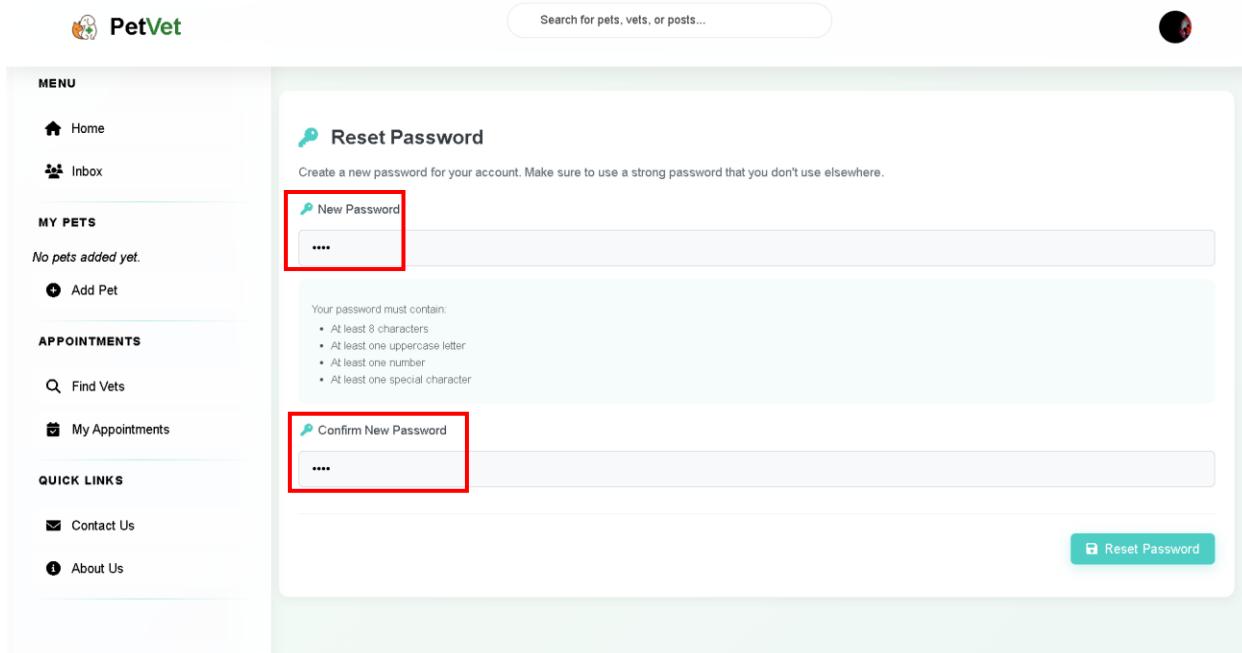


Figure 356: The new password was made very small

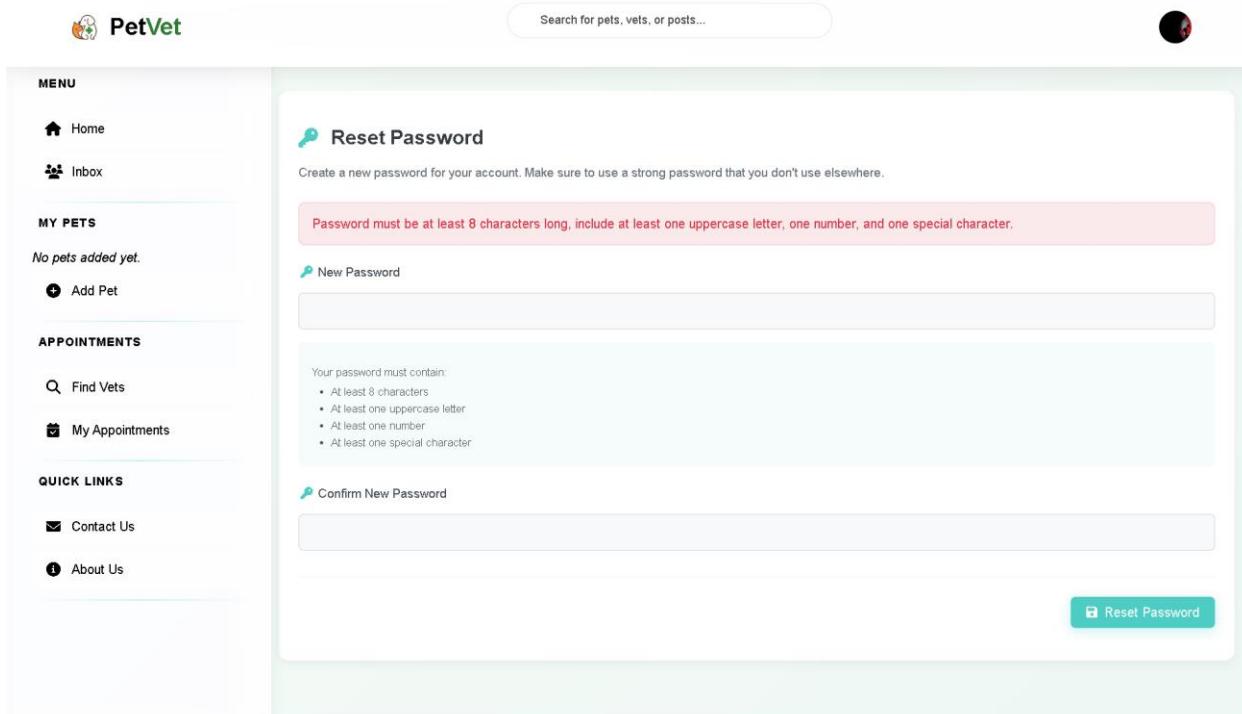
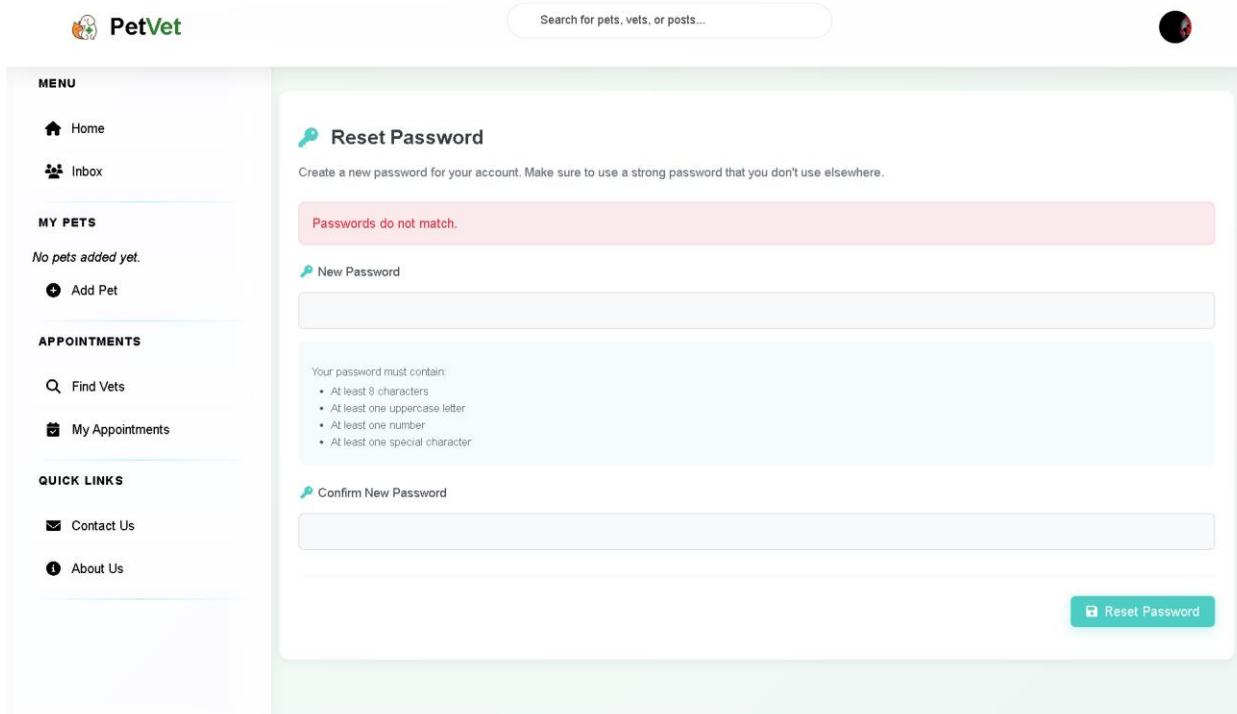


Figure 357: Error message was displayed as the password was too small

Keeping the passwords Katwal123 and another Katwal123!



The screenshot shows the PetVet application's password reset interface. On the left, there's a sidebar with a 'MENU' section containing 'Home', 'Inbox', 'Add Pet', 'Find Vets', 'My Appointments', 'Contact Us', and 'About Us'. Below this are 'MY PETS' (with a note 'No pets added yet.') and 'APPOINTMENTS' sections. On the right, the main area has a title 'Reset Password' with the sub-instruction 'Create a new password for your account. Make sure to use a strong password that you don't use elsewhere.' A red error message box contains the text 'Passwords do not match.' Below it is a 'New Password' input field. Further down, a note says 'Your password must contain:' followed by a bulleted list: 'At least 8 characters', 'At least one uppercase letter', 'At least one number', and 'At least one special character'. Another input field is labeled 'Confirm New Password'. At the bottom right is a teal-colored 'Reset Password' button.

Figure 358: Mismatching two passwords

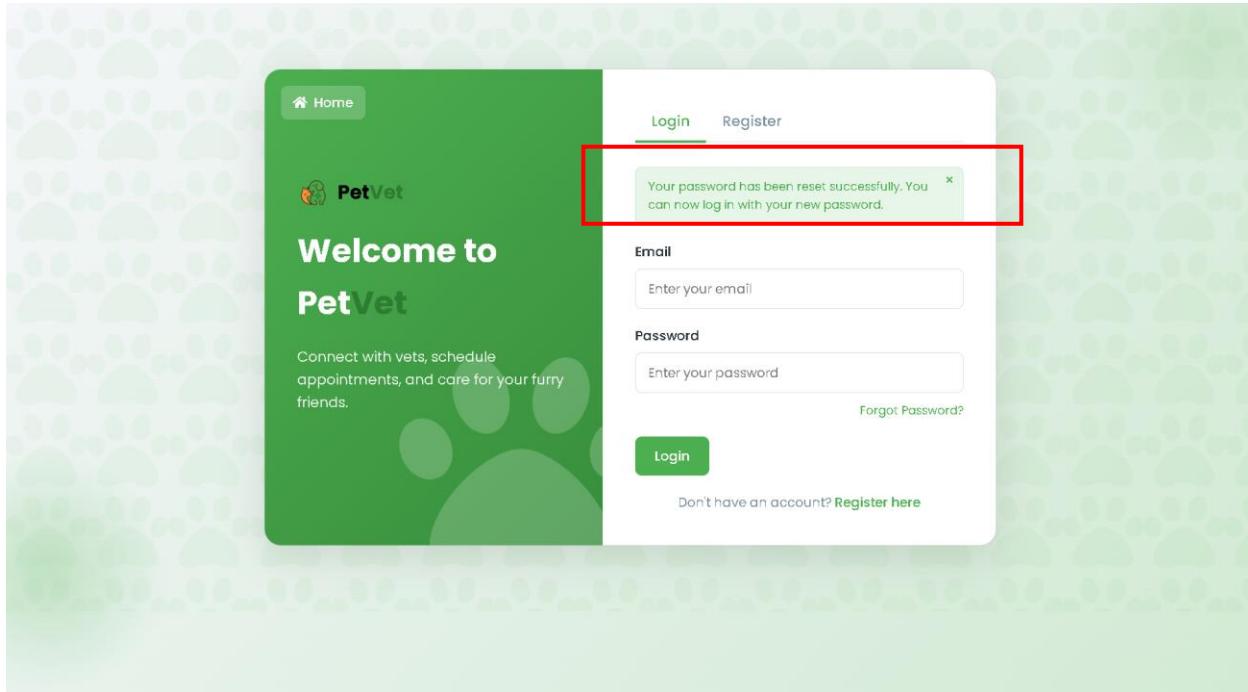


Figure 359: Redirected to log in page

If the link is clicked again, an error message will be shown.

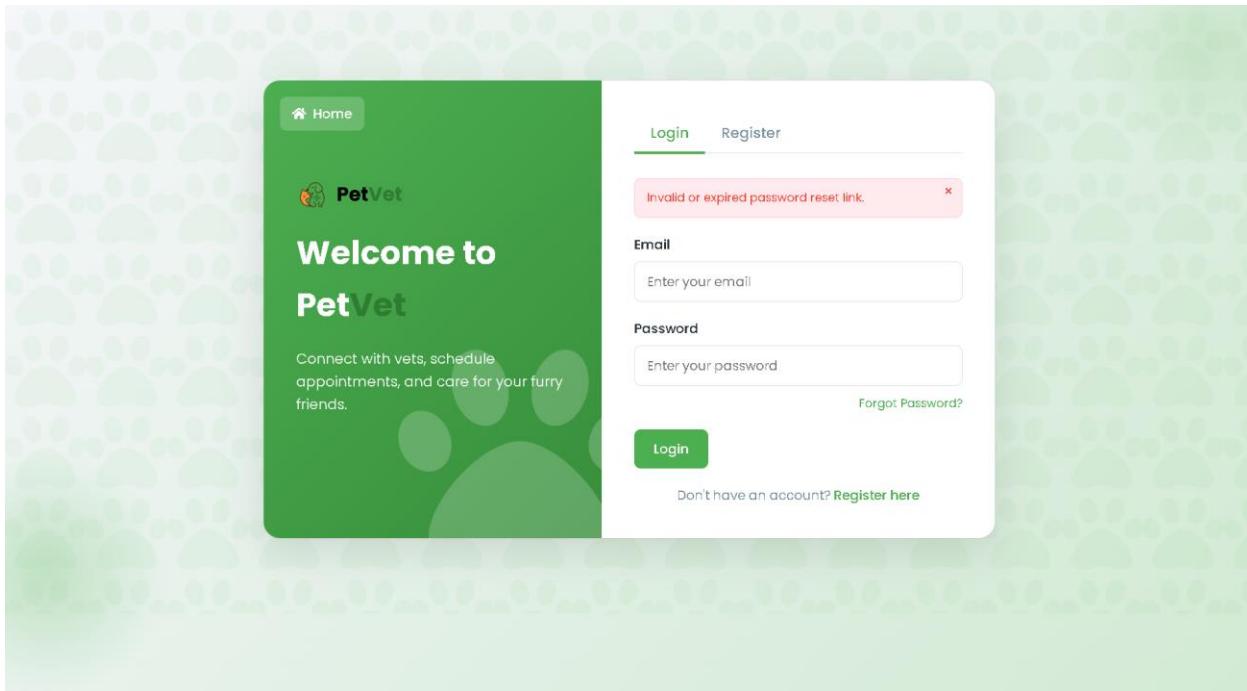


Figure 360: Error message will show if the link is clicked again

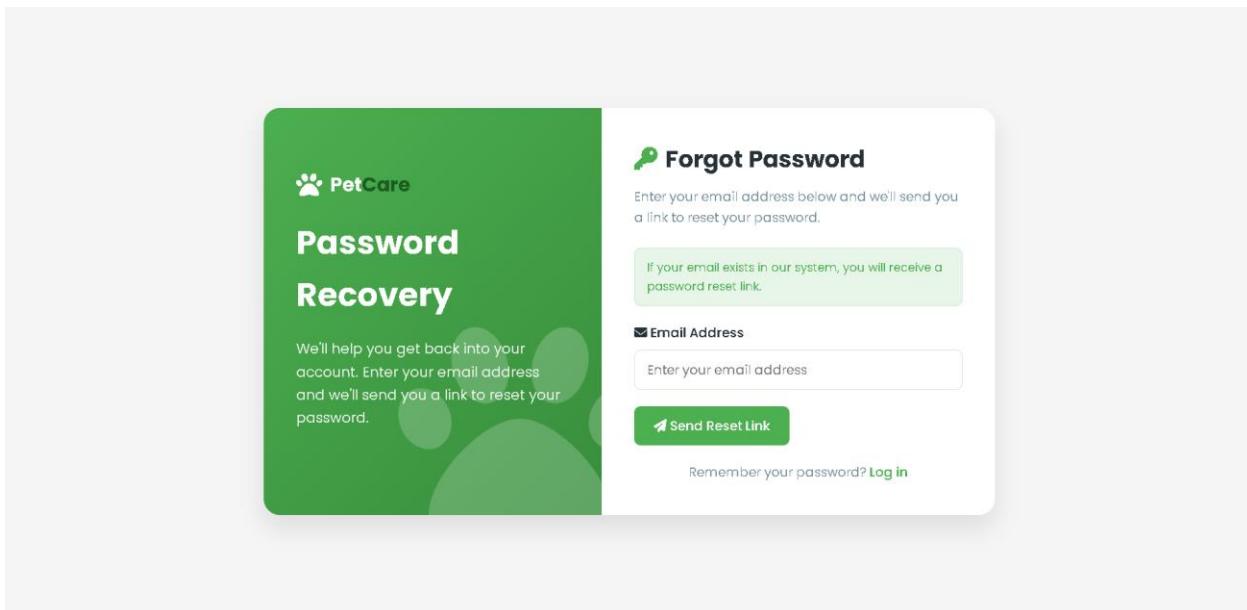


Figure 361: Clicking on the forgot password button from the login

4.3.21 System Test: Utils3 Search Functionality

Test no.	Utils3
Objective	To verify that the search feature accurately returns users or posts that match the input query based on full name, username, or post title.
Testcase	<ol style="list-style-type: none"> 1. Search User (Vet) 2. Search User (Pet Owner) 3. Search Post by Title
Action	<ol style="list-style-type: none"> 1. Typed "shubham" → returned one vet user: Shubham Datta Mishra 2. Typed "rebof" → returned two pet owner users with the name Rebof 3. Typed "essential grooming tips" → returned post titled “5 Essentials Grooming Tips for a Healthy, Happy Dog”
Expected Result	<ul style="list-style-type: none"> • Search should display relevant users or posts that match the keyword partially or fully in name, username, or post title
Actual Results	Relevant search results displayed correctly based on input queries.
Conclusion	The test was successful

Table 55: System Test - Search

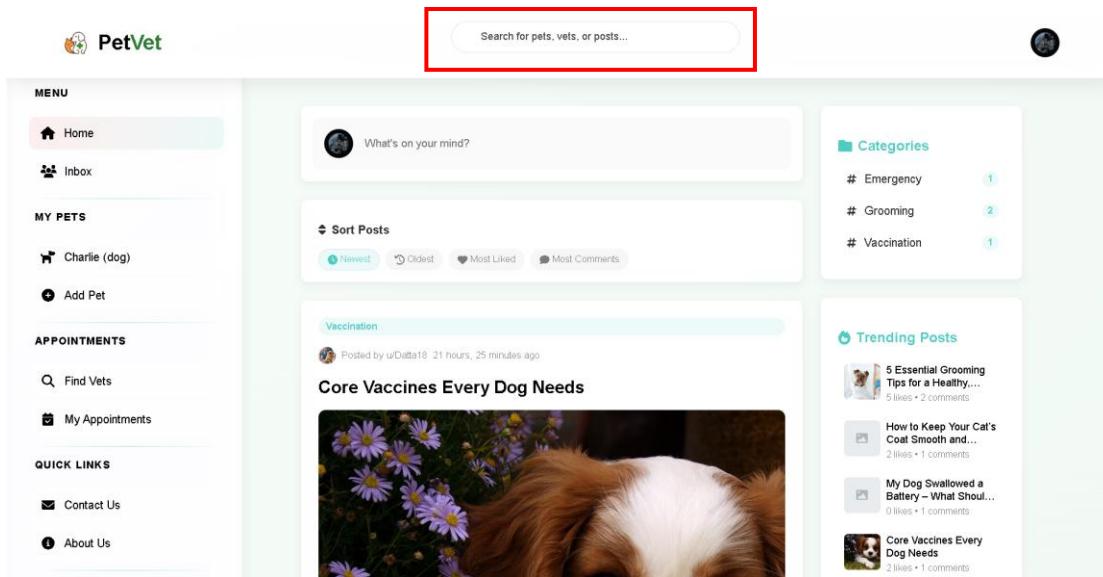


Figure 362: Clicking the search

This screenshot shows the search results for 'Shubham' on the PetVet website. The search bar at the top contains the name 'Shubham' with a red box highlighting it. The results page is titled 'Search Results for "Shubham"' and indicates 'Found across posts, veterinarians, and pet owners'. The first result is for 'Veterinarians (1)', listing 'Dr. Shubham Datta Mishraa' from 'Mishra Pet Clinic' in 'Lalitpur, Nepal'. It shows a profile picture of a dog, a verified badge, a 5.0 rating based on 1 review, and a summary: 'Experienced veterinarian dedicated to providing compassionate care for your pets.' Below this are buttons for 'Book Appointment' and 'View Profile'.

Figure 363: Searching for "Shubham"



Figure 365: Searching for "Rebof"

A screenshot of the PetVet mobile application showing search results. On the left is a sidebar with "MENU" (Home, Inbox), "MY PETS" (Charlie (dog), Add Pet), "APPOINTMENTS" (Find Vets, My Appointments), and "QUICK LINKS" (Contact Us, About Us). The main area shows a search bar at the top with the query "Rebof". Below it is the heading "Search Results for 'Rebof'" with a subtitle "Found across posts, veterinarians, and pet owners". Underneath is a section titled "Pet Owners (2)" with two entries. Both entries show a circular profile picture, the name "Rebof Katwal", the age "22", the title "Pet Owner", and a status message: "None" for the first and "Hi, I used to own a Labrador!" for the second. Each entry has a "View Profile" button at the bottom.

Figure 366: Search result for "Rebof"



Figure 367: Searching for "Essential Grooming Tips"

A screenshot of the PetVet website showing search results. The left sidebar contains a "MENU" with links to Home, Inbox, and Add Pet; a "MY PETS" section with Charlie (dog); and sections for APPOINTMENTS (Find Vets, My Appointments) and QUICK LINKS (Contact Us, About Us). The main content area shows a search bar at the top with the query "Essential Grooming Tips". Below it is a heading "Search Results for 'Essential Grooming Tips'" with a subtitle "Found across posts, veterinarians, and pet owners". A card titled "Posts (1)" displays the result: "5 Essential Grooming Tips for a Healthy, Happy Dog". The card text reads: "Grooming isn't just about looking good—it's about your dog's overall health and comfort. Here are five tips to keep your fury friend in top shape: ...". At the bottom of the card are icons for a message and a person.

Figure 368: Search result for "Essential Grooming Tips"

4.3.22 System Test: Utils4 User Information Update Functionality

Test no.	Utils4
Objective	To verify that the user can successfully update their profile information and updates reflect properly after changes.
Testcase	1. Update Address Field and Update Profile Picture
Action	<ul style="list-style-type: none"> • Logged in as vet Shubham Datta Mishra • Clicked edit profile, changed the address from "behind sumeru hospital, dhapakhel" to "Behind ANFA" • Updated the profile picture • Changes to address reflected immediately • Profile picture update took multiple refreshes before reflecting correctly
Expected Result	<ul style="list-style-type: none"> • Both the address and profile picture should update immediately after saving changes.
Actual Results	Address updated instantly; profile picture updated but visible only after multiple refreshes.
Solution	<ul style="list-style-type: none"> • Since there were no errors detected but it still took some time to reflect the changes, I concluded that the issue was due to the browser cache. • To fix this, I added a new field in both the PetOwnerProfile and VetProfile models: <ul style="list-style-type: none"> ✓ last_updated = models.DateTimeField(auto_now=True)

	<ul style="list-style-type: none"> Then, I updated the image tags in the template like this: <pre> ✓ ✓ </pre> <ul style="list-style-type: none"> By appending ?v=timestamp, the browser is forced to immediately fetch the updated image instead of using the cached version.
Conclusion	The test was successful

Table 56: System Test - User Profile Update

The screenshot shows the PetVet user profile page for Shubham Datta Mishraa. The page is divided into several sections:

- Header:** PetVet logo, search bar, and account info.
- Left Sidebar (MENU):**
 - Home
 - Inbox
 - APPOINTMENTS
 - Pending
 - Accepted
 - My Schedule
 - Add Schedule
 - QUICK LINKS
 - Contact Us
 - About Us
- User Profile Section:**

Shubham Datta Mishraa

Profile picture, email (shubhammishra@gmail.com), phone (98030340556), and verified status.

Edit Profile button (highlighted with a red box).
- Basic Information:**

Full Name:	Shubham Datta Mishraa
Email:	shubhammishra@gmail.com
Phone:	98030340556
Gender:	Male
- Profile Information:**

Clinic Name:	Mishra Pet Clinic
Specialization:	Small Animal Care & Surgery
Experience:	4 years
License Number:	PAW-1234556

Rating: 5.0 (1 review)

Summary: Passionate about pet health & well-being
- Address Information:**

Country: Nepal
City: Lalitpur
Address: Behind Sumeru Hospital, Dhapakhel
- Posts:**

Core Vaccines Every Dog Needs

Vaccination post with a photo of a dog's head.

View and Edit buttons.
- Comments:**

On Post: How to Keep Your Cat's Coat Smooth and Tangle-Free

Informative! comment by Rebof Katwal 22.
- Reviews:**

Rebof Katwal 22 gave 5 stars.

Awesome vet!

Figure 369: User profile page for updating

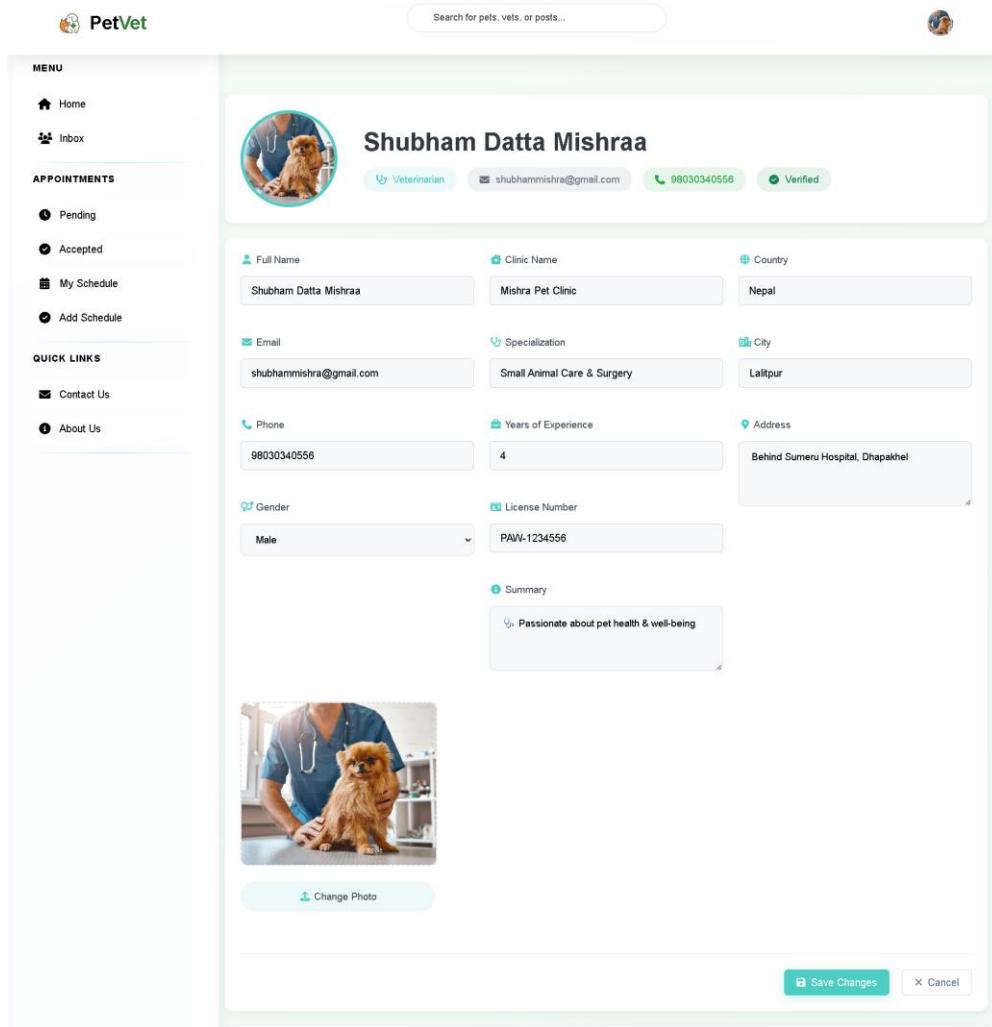


Figure 370: Edit mode is activated

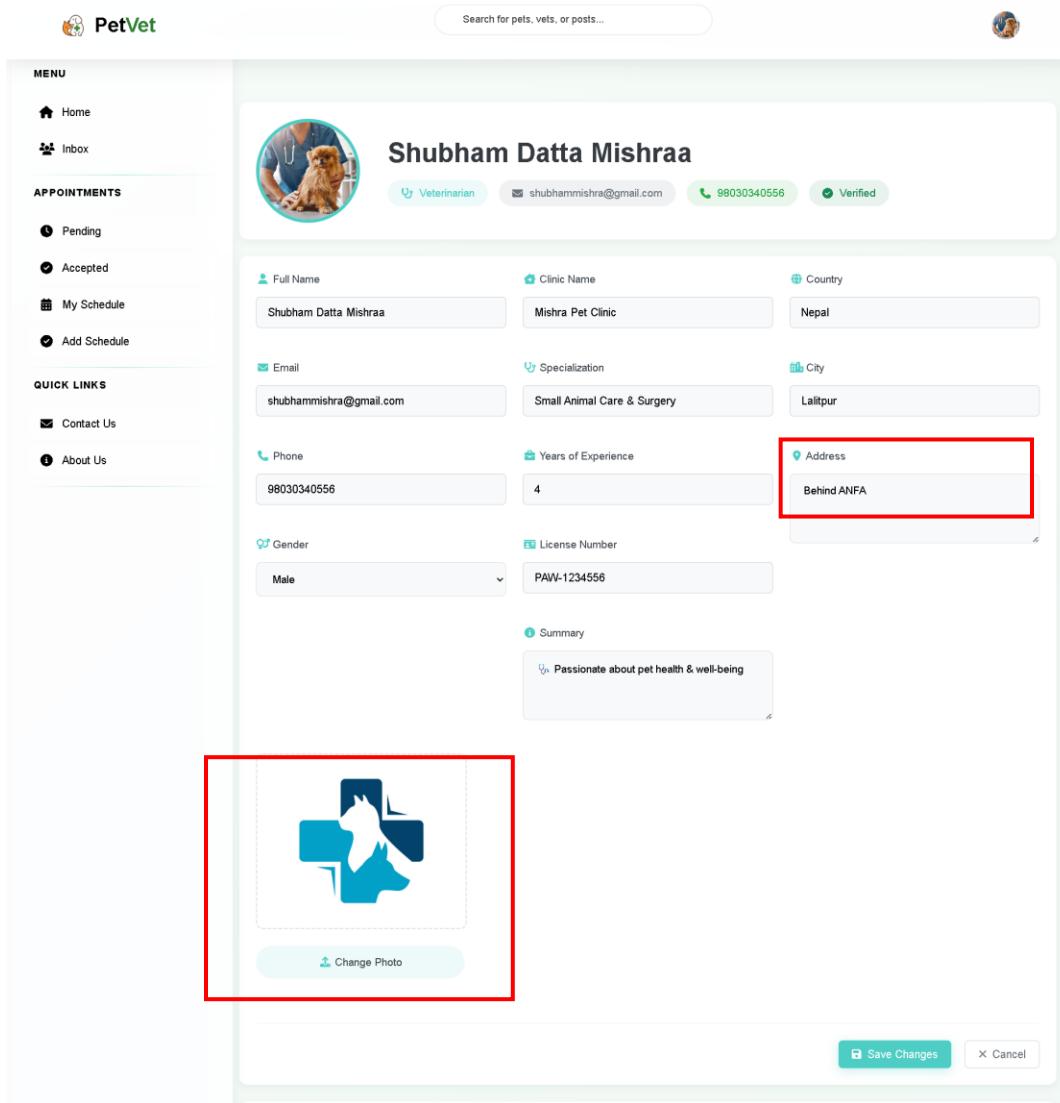


Figure 371: Updated the fields

Changes made to the address field are visible, but the profile does not update immediately. After multiple refreshes of the post, the changes become visible.

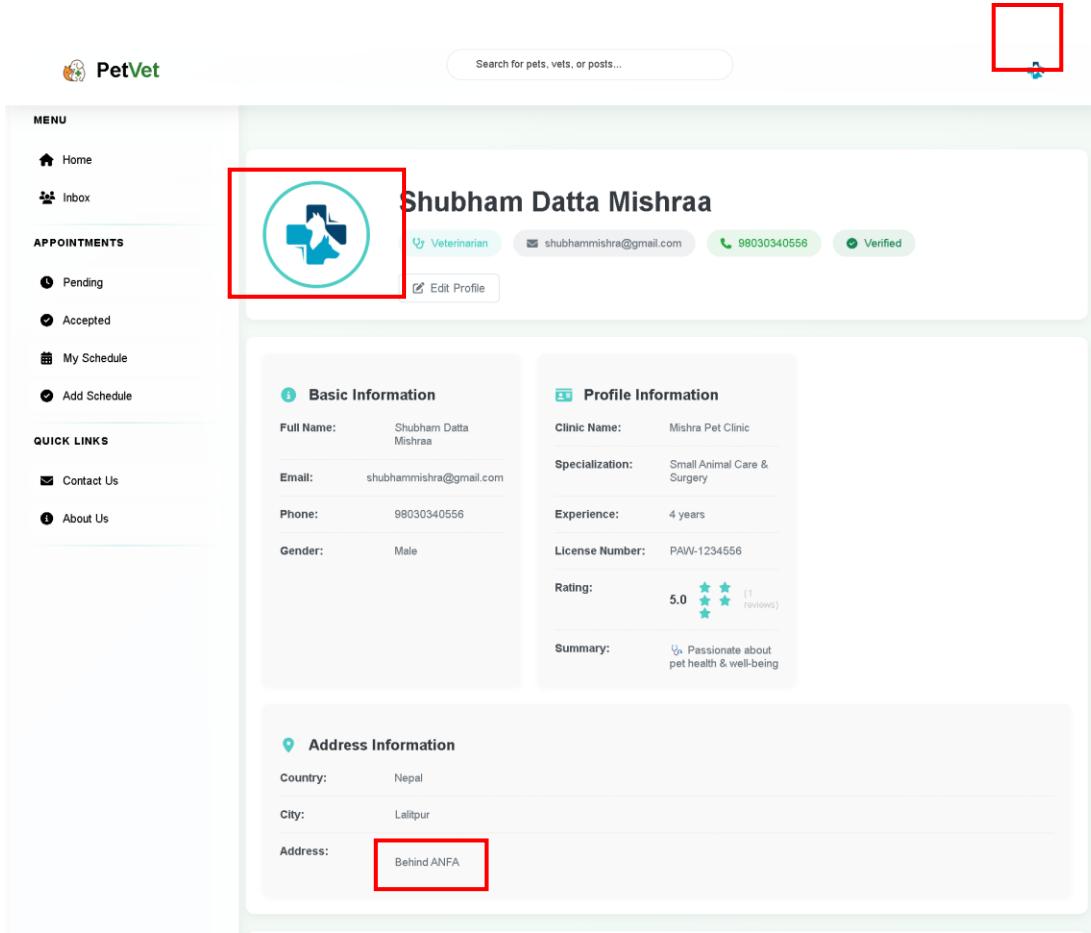


Figure 372: The fields are updated but need multiple refreshs for the change to show

But we need immediate change.

```
<div class="user-profile-header">
    <div class="user-avatar">
        {% if user.petownerprofile %}
            {% if user.petownerprofile.human_image %}
                
            {% else %}
                <div class="no-image">
                    |   <i class="fas fa-user-circle"></i>
                </div>
            {% endif %}
        {% elif user.vetprofile %}
            {% if user.vetprofile.vet_image %}
                
            {% else %}
                <div class="no-image">
                    |   <i class="fas fa-user-circle"></i>
                </div>
            {% endif %}
        {% else %}
    
```

Figure 373: HTML code before the solution

To resolve this issue, a field was added to the model.

```
last_updated = models.DateTimeField(auto_now=True)
```

Figure 374: Newly Added field to the model

```
{% if user.petownerprofile %}
    {% if user.petownerprofile.human_image %}
        
            |   <i class="fas fa-user-circle"></i>
        </div>
    {% endif %}
{% elif user.vetprofile %}
    {% if user.vetprofile.vet_image %}
        
            |   <i class="fas fa-user-circle"></i>
        </div>
    {% endif %}
{% else %}

```

Figure 375: The added field used as the solution

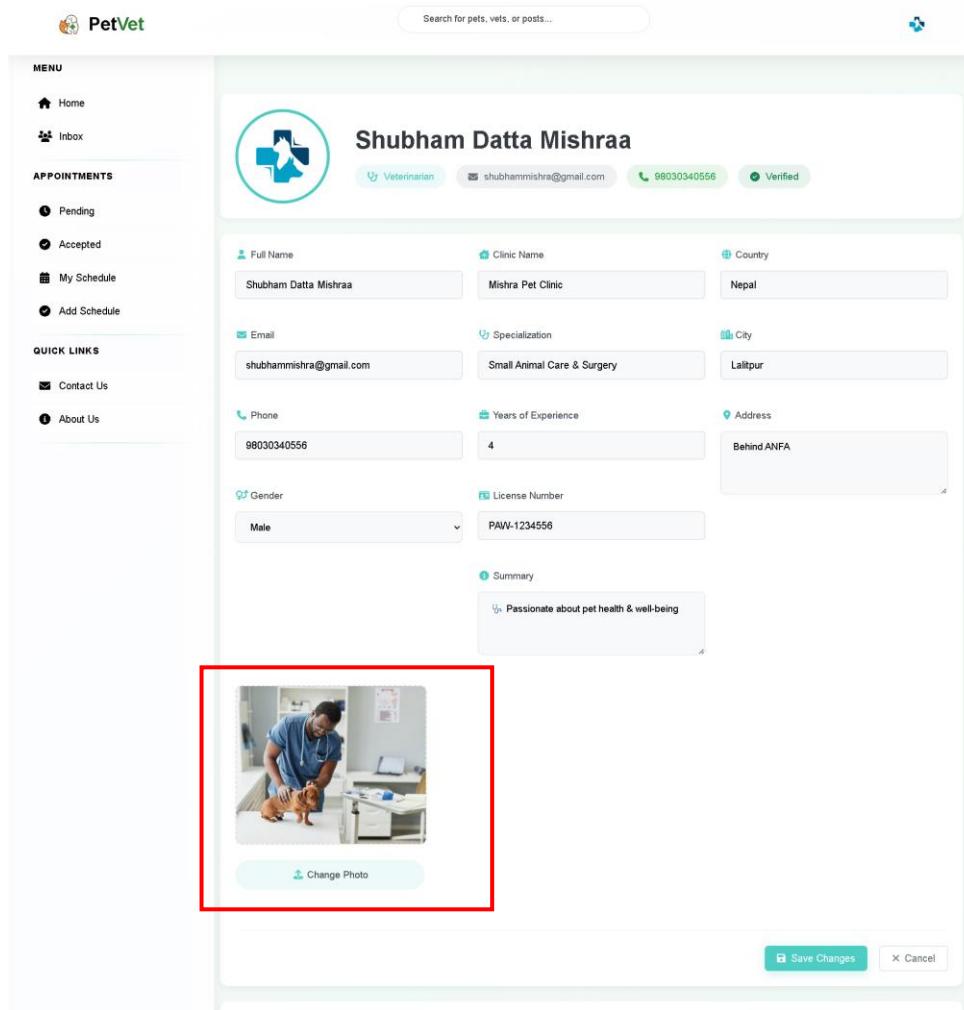


Figure 376: Updating the profile picture again

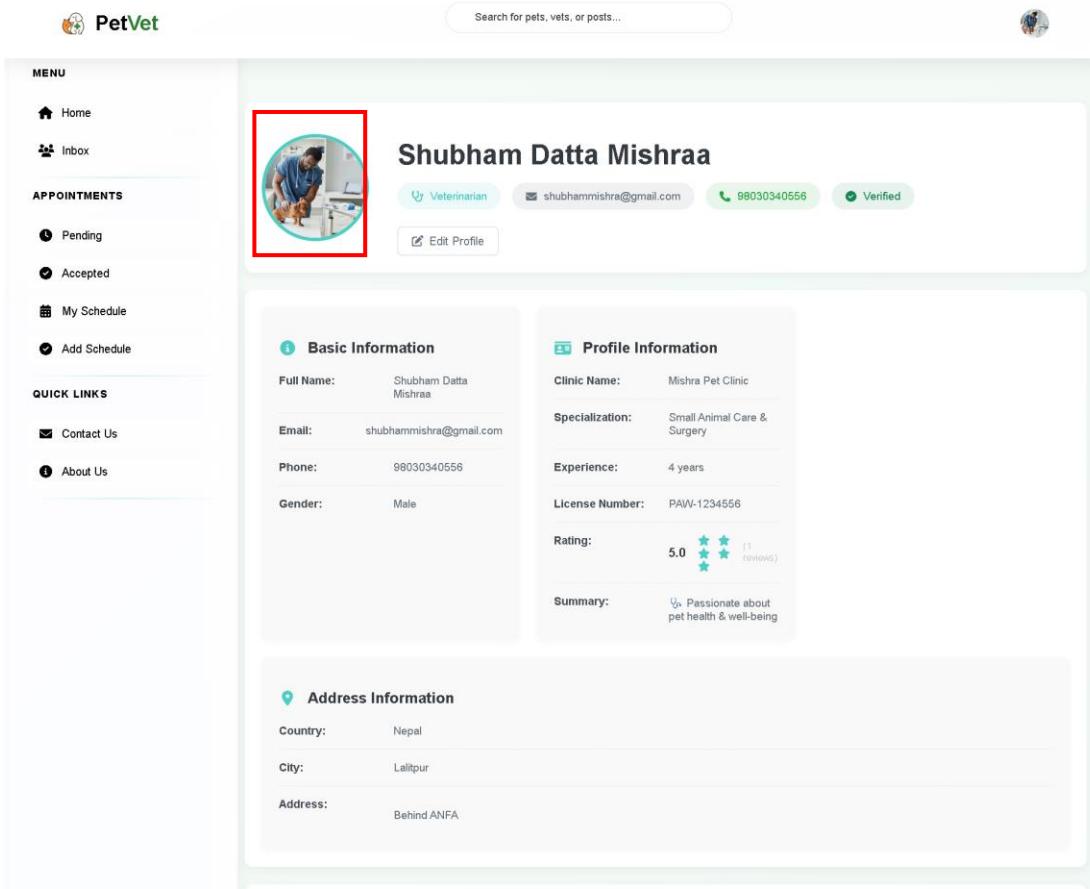


Figure 377: Profile picture was immediately updated

4.3.23 System Test: Utils5 Contact Form and Email Notification Checking

Test no.	Utils5
Objective	To verify the behavior of user creation (with OTP and vet approval emails), appointment booking (conflict management), and contact form email sending.
Testcase	<ol style="list-style-type: none"> 1. Pet Owner User Creation and OTP Email 2. Vet User Creation and Approval Email 3. Booking Appointments with Overlapping Time Slots 4. Appointment Acceptance and Automatic Rejection of Overlapping Requests 5. Contact Form Email Sending
Action	<ol style="list-style-type: none"> 1. Created a pet owner with email shuvhamdatta@gmail.com; OTP was sent successfully. 2. Created a vet user with email kumarkatuwal1964@gmail.com; vet approval email was sent. 3. Logged in as Rebof Katwal, booked an appointment via Khalti for vet Kumar Katwal

	<p>(Monday 3:15–5:15 PM); vet received booking request email.</p> <ol style="list-style-type: none"> 4. Logged in as Shubham, tried to book same time slot using store credit; vet received second booking request email. 5. Logged in as vet Kumar Katwal, accepted Rebof's booking: <ul style="list-style-type: none"> • Confirmation email was sent to Rebof • Shubham's booking was automatically rejected and rejection email sent. 6. For contact form: <ul style="list-style-type: none"> • Name: Astha Thapa • Email: asthapa311@gmail.com • Title: Reporting a post • Message: "Hey, I want something to be remove." • Email was successfully sent.
Expected Result	<ul style="list-style-type: none"> • OTP and approval emails should be properly sent on user creation.

	<ul style="list-style-type: none"> • Vet appointment system should prevent double-booking, confirming one and rejecting others automatically. • Contact form should successfully send a report email.
Actual Results	All email notifications and booking behaviors worked exactly as expected.
Conclusion	The test was successful

Table 57: System Test - Email Notifications and Contact Form

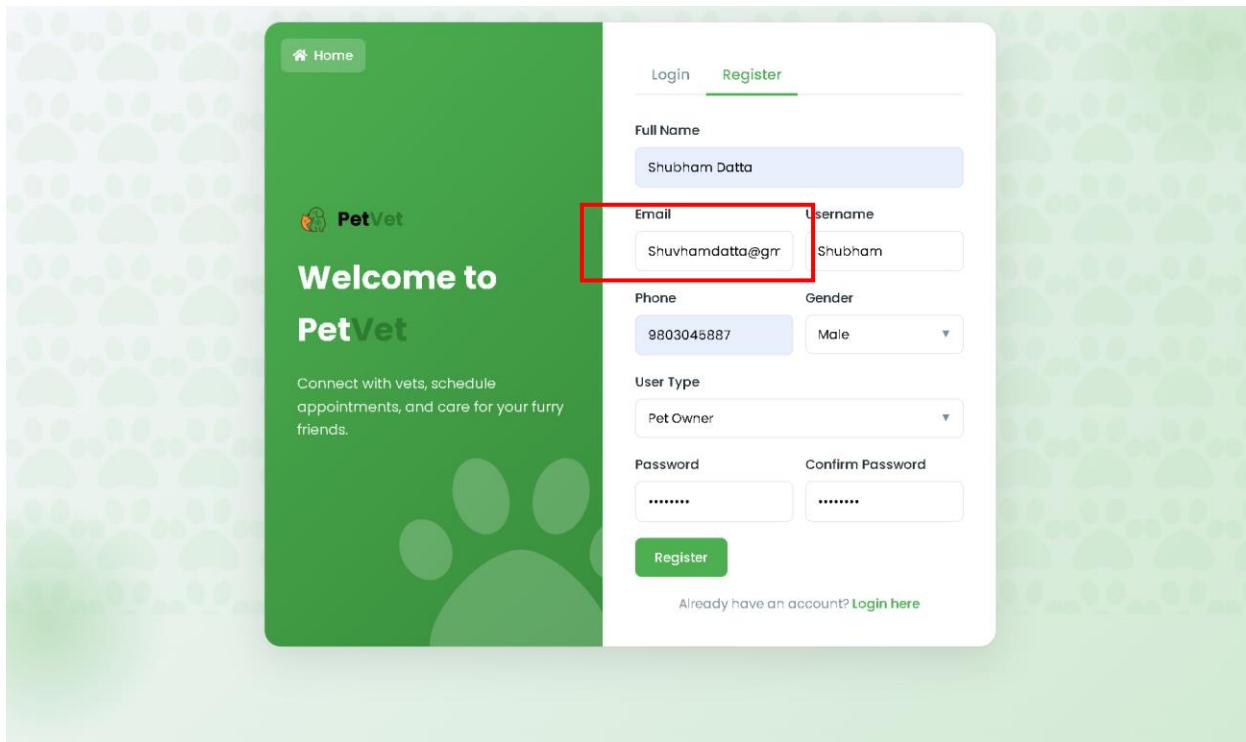


Figure 378: Registering a pet owner user for otp email check

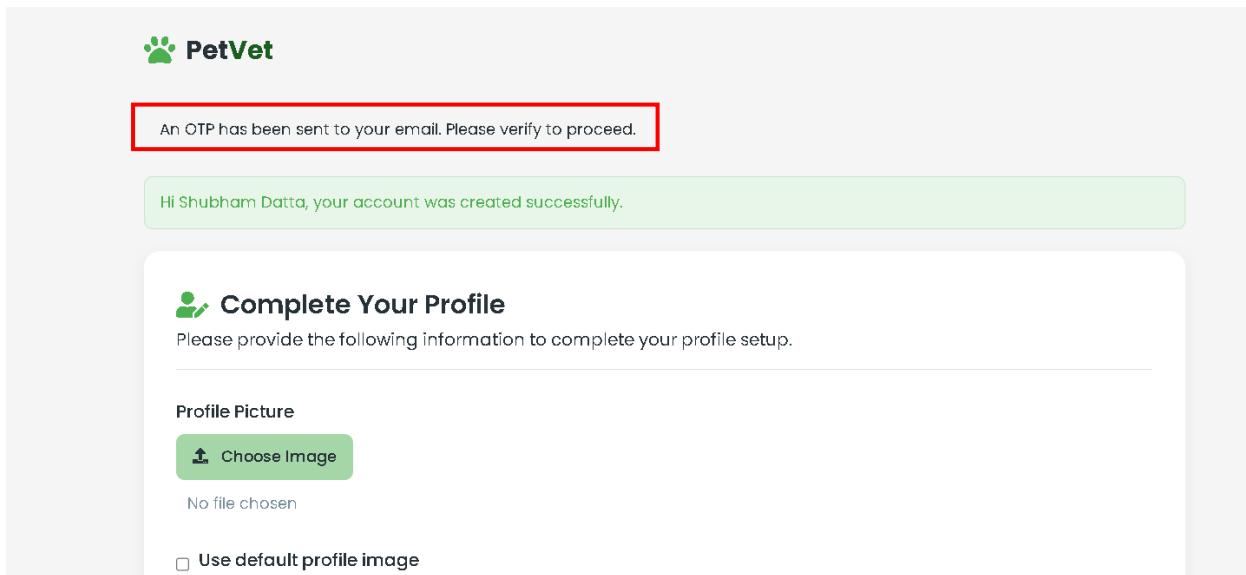


Figure 379: The otp was sent to the email

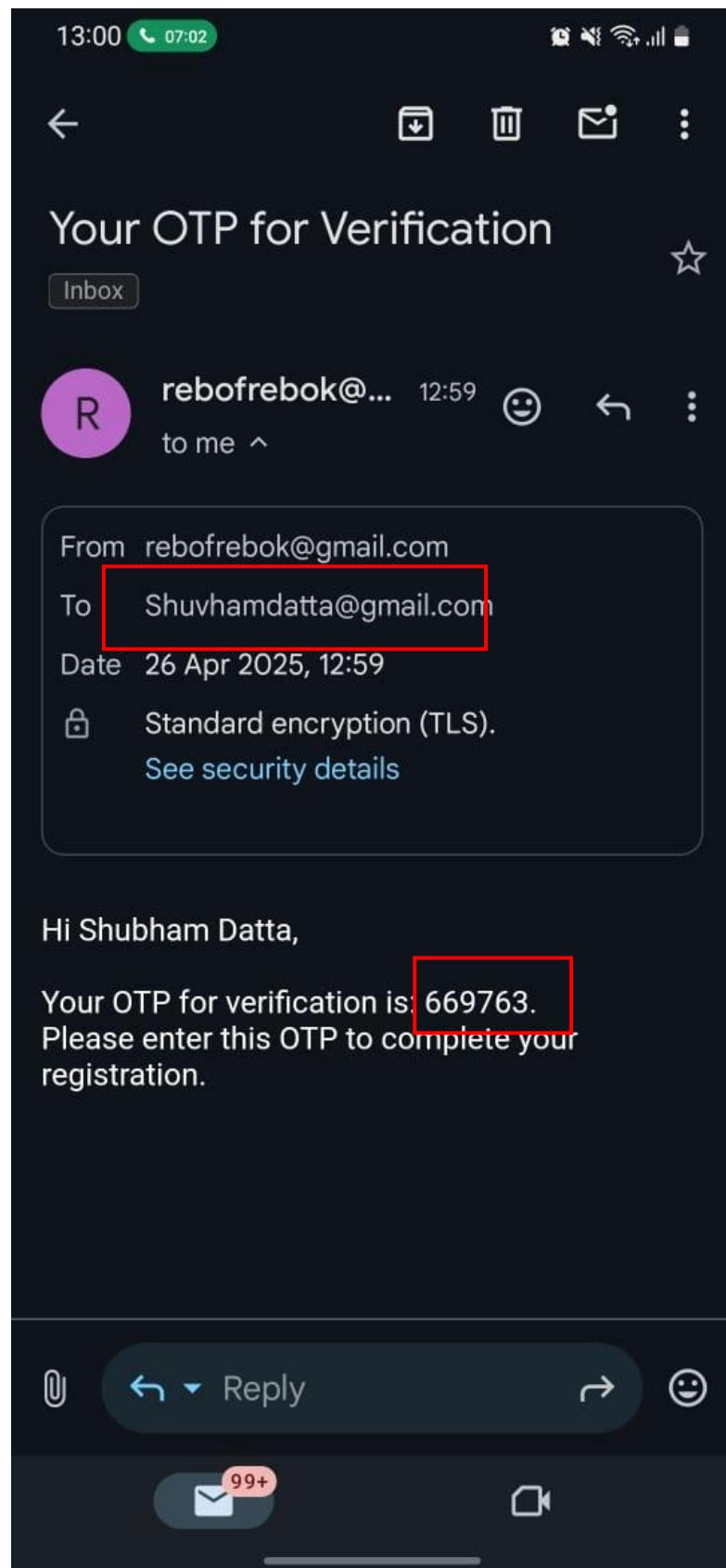


Figure 380: The otp was sent

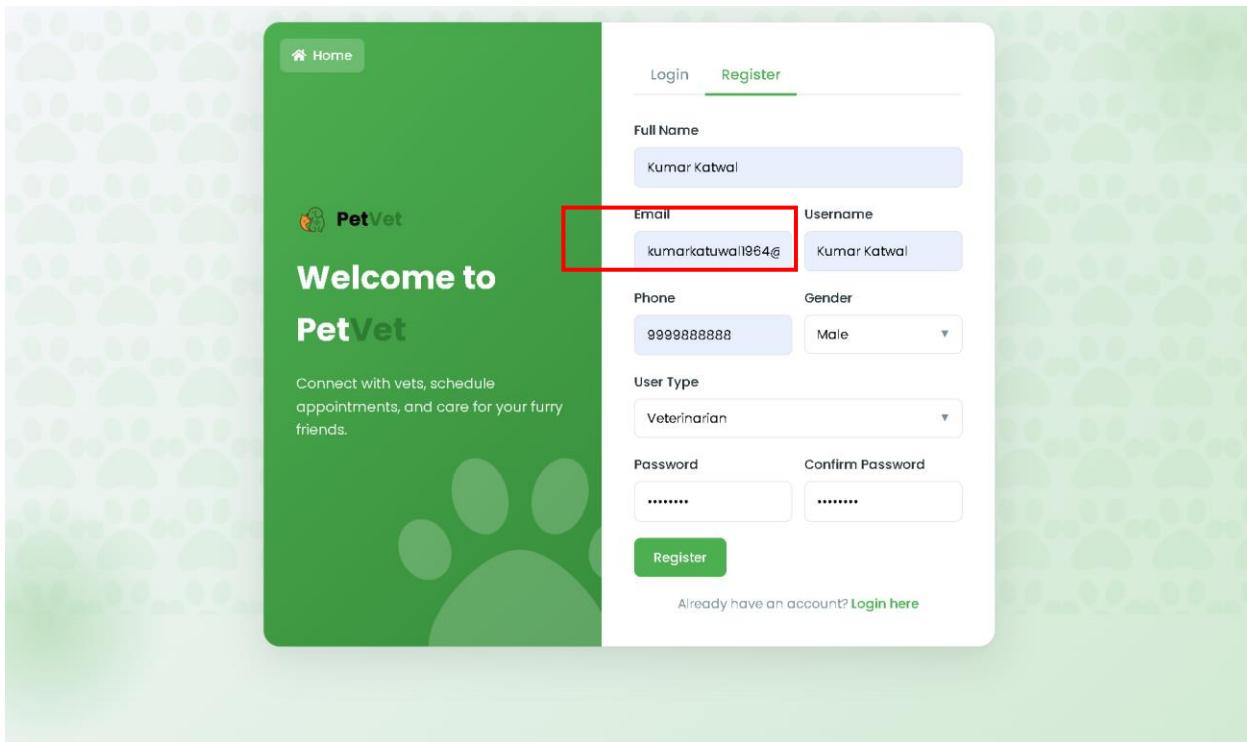


Figure 381: Creating a new vet account for approval email notification check

Vet Approvals

Veterinarian 'Kumar Katwal' has been approved successfully.

Pending Approvals	Recently Approved	Recently Declined
0 Vets waiting for approval	4 Vets approved in the last 7 days	0 Vets declined in the last 7 days

Pending Vet Approvals

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
No pending vet approvals.						

Recently Approved Vets

Name	Email	Clinic Name	License Number	Approved At
Test Name Vet	testvet@gmail.com	Test clinic	NVC-NP-2024-89897	Apr 25, 2025
Kumar Katwal	kumarkatuwal1964@gmail.com	Katwal Pet Clinic	PAW-777777	Apr 26, 2025
Avishek Gautam	avishekgautam@gmail.com	Where Pets Come First Clinic	PAW-123999	Apr 25, 2025
Shubham Datta Mishraa	shubhammishra@gmail.com	Mishra Pet Clinic	PAW-1234556	Apr 25, 2025

Recently Declined Vets

Name	Email	Clinic Name	License Number	Declined At
No recently declined vets.				

[Logout](#)

Figure 382: The Vet was approved

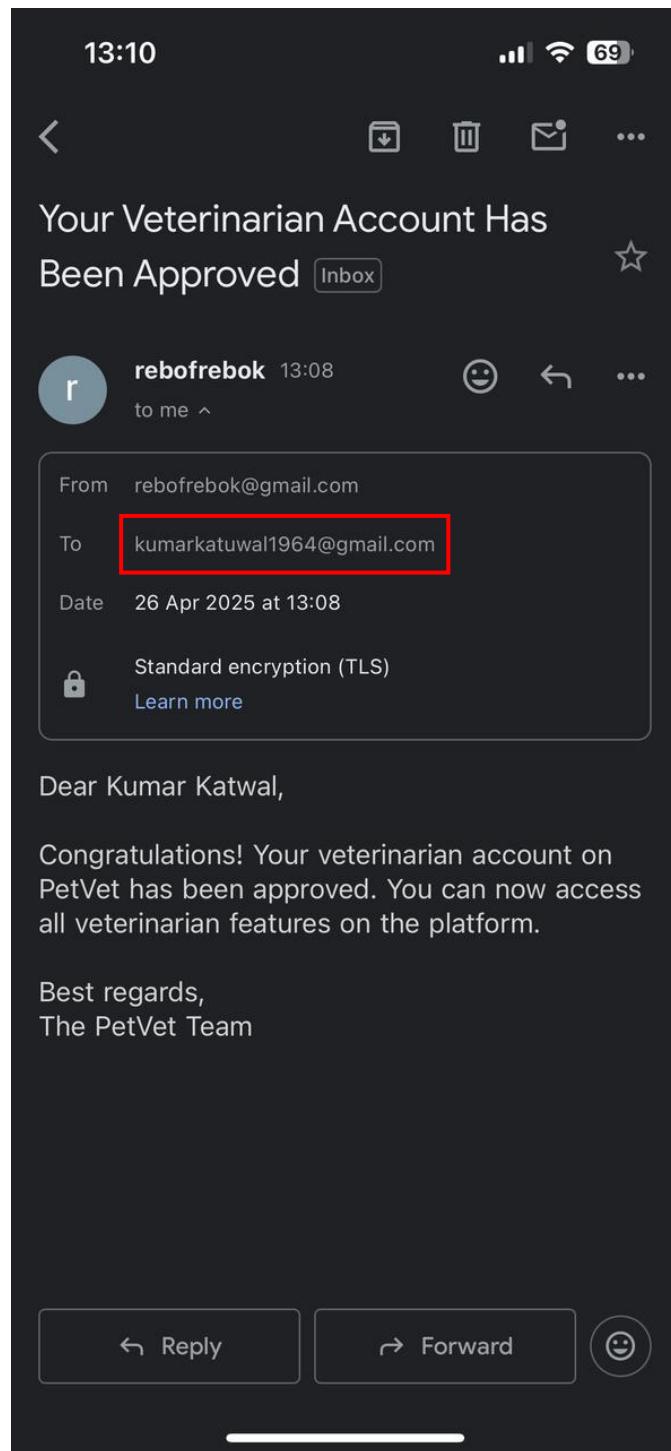


Figure 383: The approval email was received

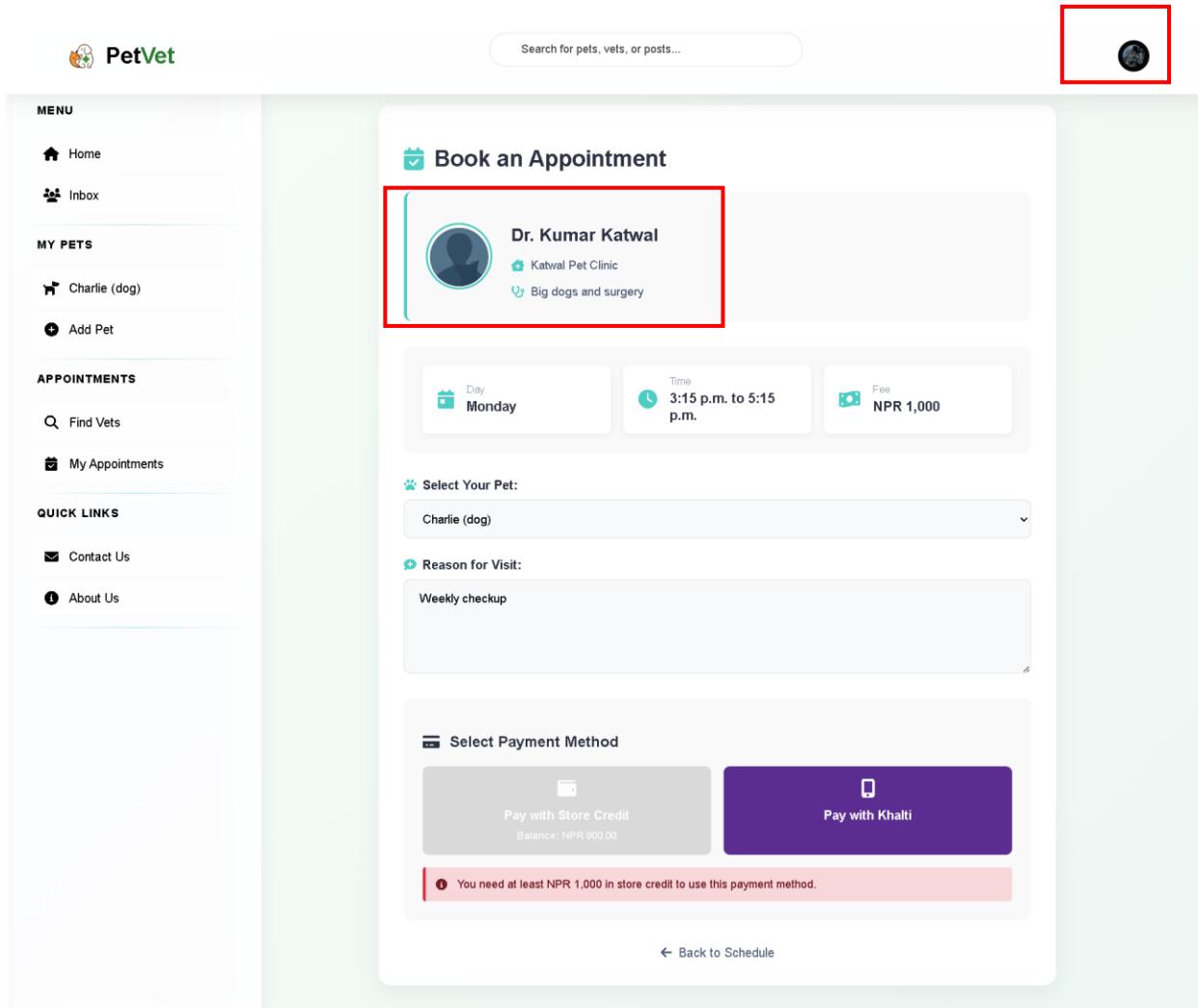


Figure 384: Booking the appointment with Khalti as Rebof

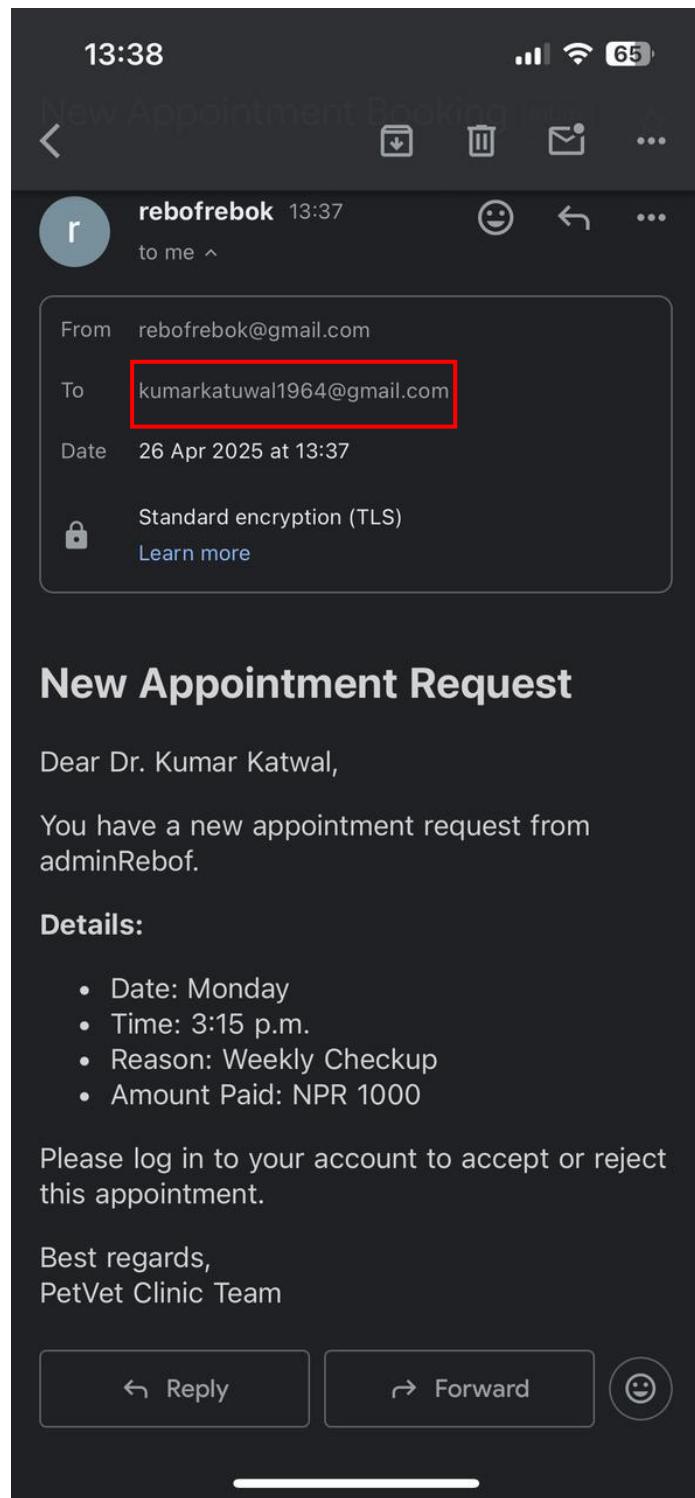


Figure 385: The appointment email notification is sent to the Vet

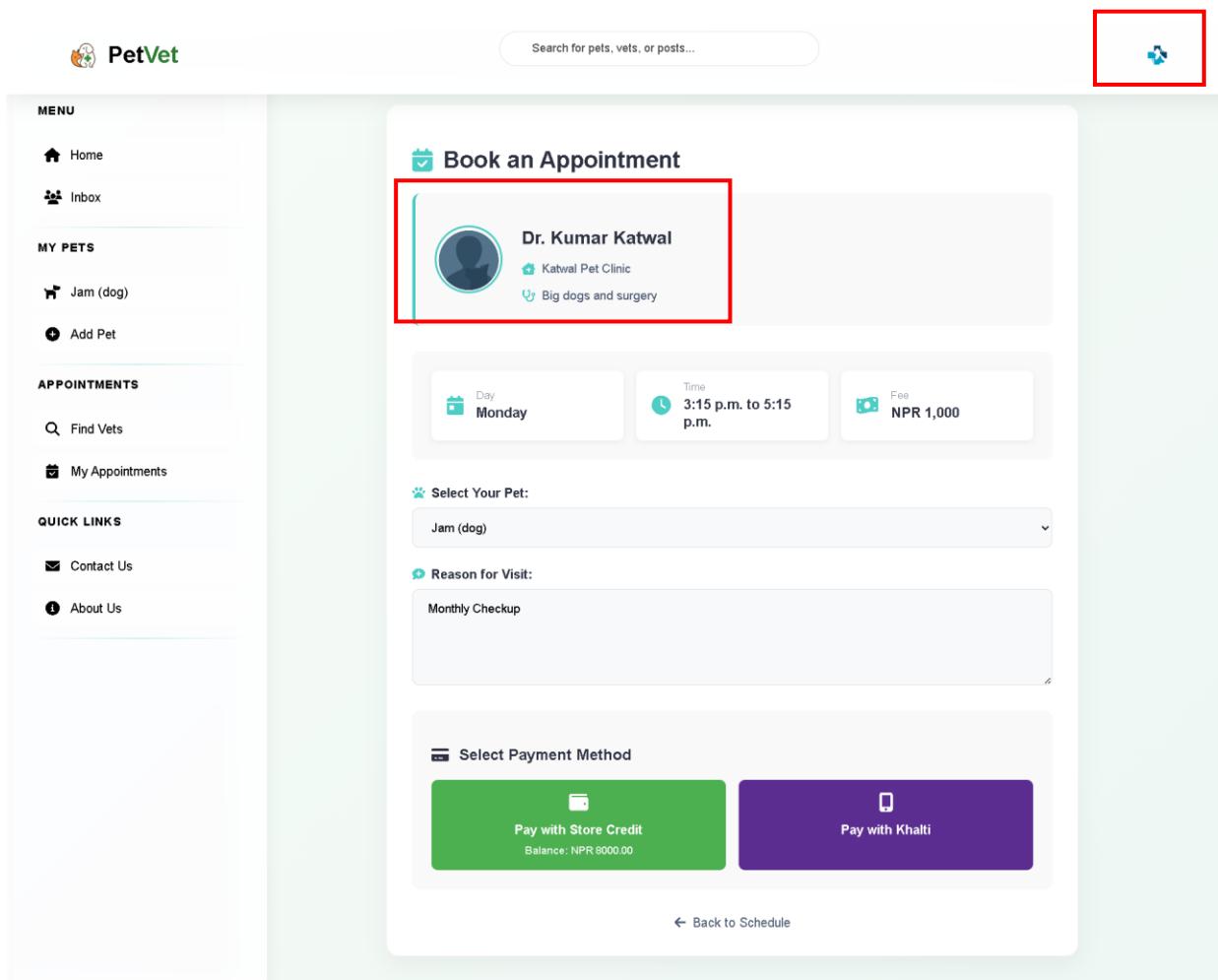


Figure 386: Booking the appointment by using Store Credit as Shubham

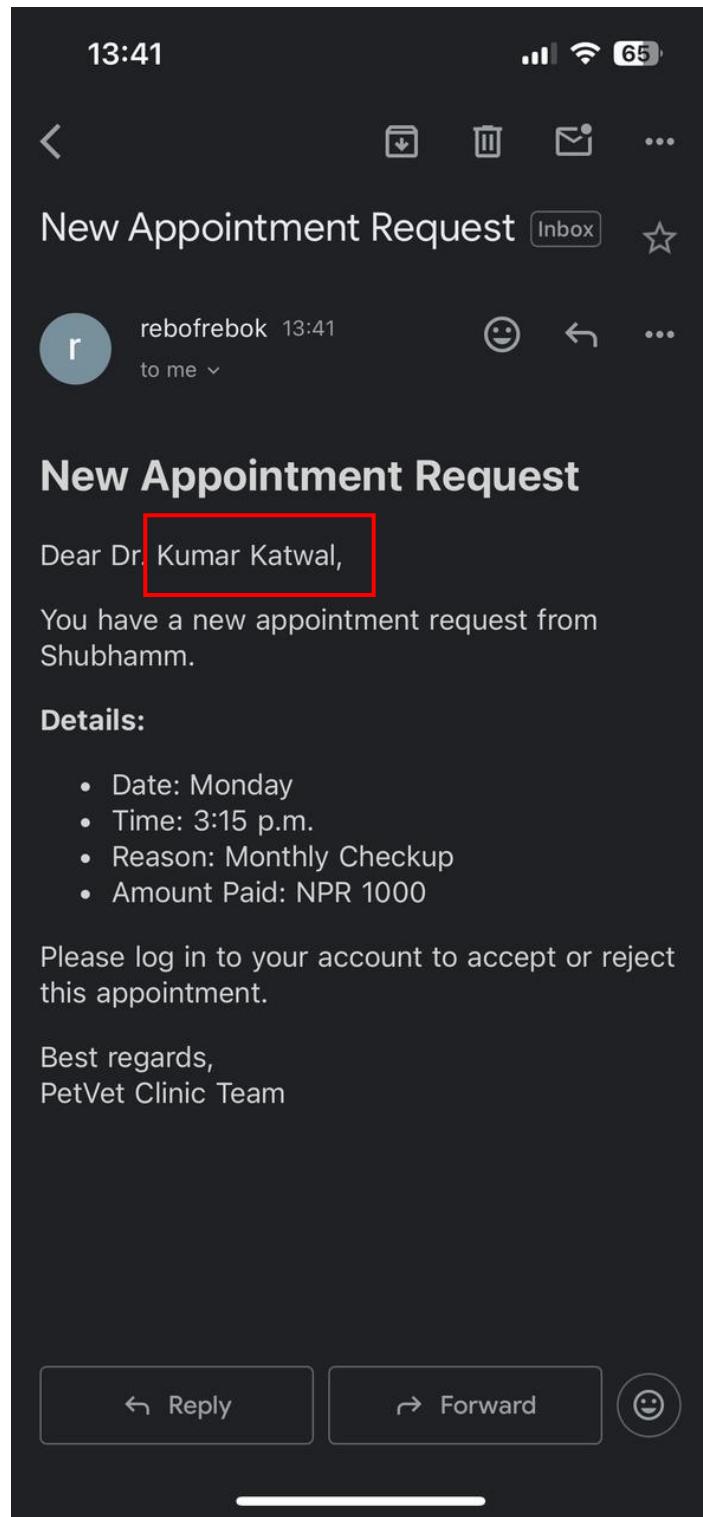


Figure 387: Email notification sent to the vet

The screenshot displays the PetVet mobile application interface. At the top, there is a navigation bar with a search bar containing the placeholder "Search for pets, vets, or posts...". On the right side of the top bar is a user profile icon. Below the top bar is a "MENU" section on the left, which includes links for "Home", "Inbox", "APPOINTMENTS" (with "Pending" highlighted), "Accepted", "My Schedule", and "Add Schedule". Under "QUICK LINKS", there are links for "Contact Us" and "About Us". The main content area is titled "Pending Appointment Requests" and sub-titled "Review and manage incoming appointment requests". It shows two pending appointment requests:

- Rebof Katwal 22** (Pending)
 - Owner:** Charlie (dog)
 - Reason:** Weekly Checkup
 - Requested Time:** 3:15 PM - 5:15 PM
 - Date:** Monday
 - Breed:** Labrador
 - Age:** 10 years
 - Weight:** 30.00 kg
 - Color:** White
 Buttons: **Accept** (green) and **Reject** (red).
- Shubham Datta** (Pending)
 - Owner:** Jam (dog)
 - Reason:** Monthly Checkup
 - Requested Time:** 3:15 PM - 5:15 PM
 - Date:** Monday
 - Breed:** Great Dane
 - Age:** 6 years
 - Weight:** 59.90 kg
 - Color:** Black
 - Vaccination:** Rabies Vaccination Done
 - Allergies:** Peanut Allergy
 - Medical History:** None
 Buttons: **Accept** (green) and **Reject** (red).

Figure 388: Both booking appointment request received

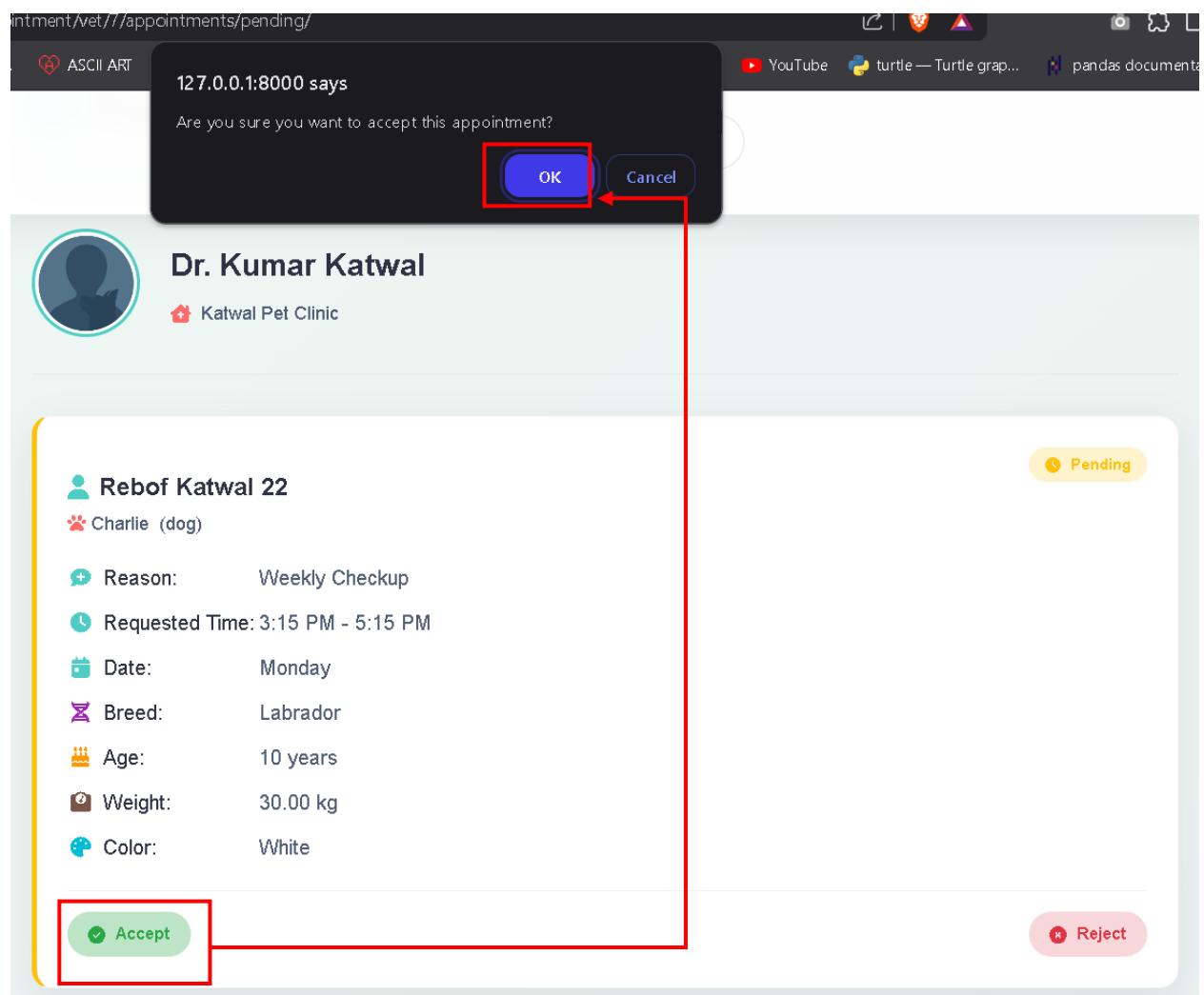


Figure 389: Accepting Rebof's booking

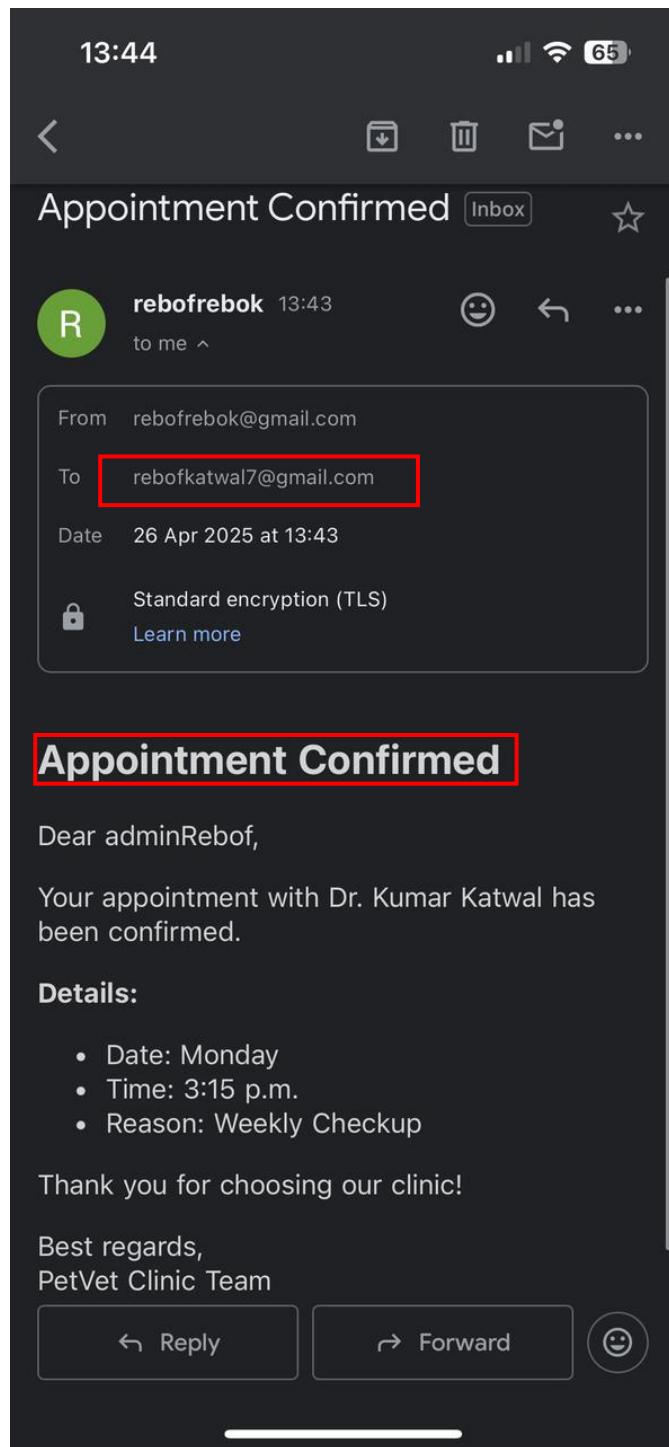


Figure 390: Appointment confirmation sent to Rebof

After accepting a booking, all other bookings for the same time slot will be automatically rejected, and a rejection email will be sent to the affected users.

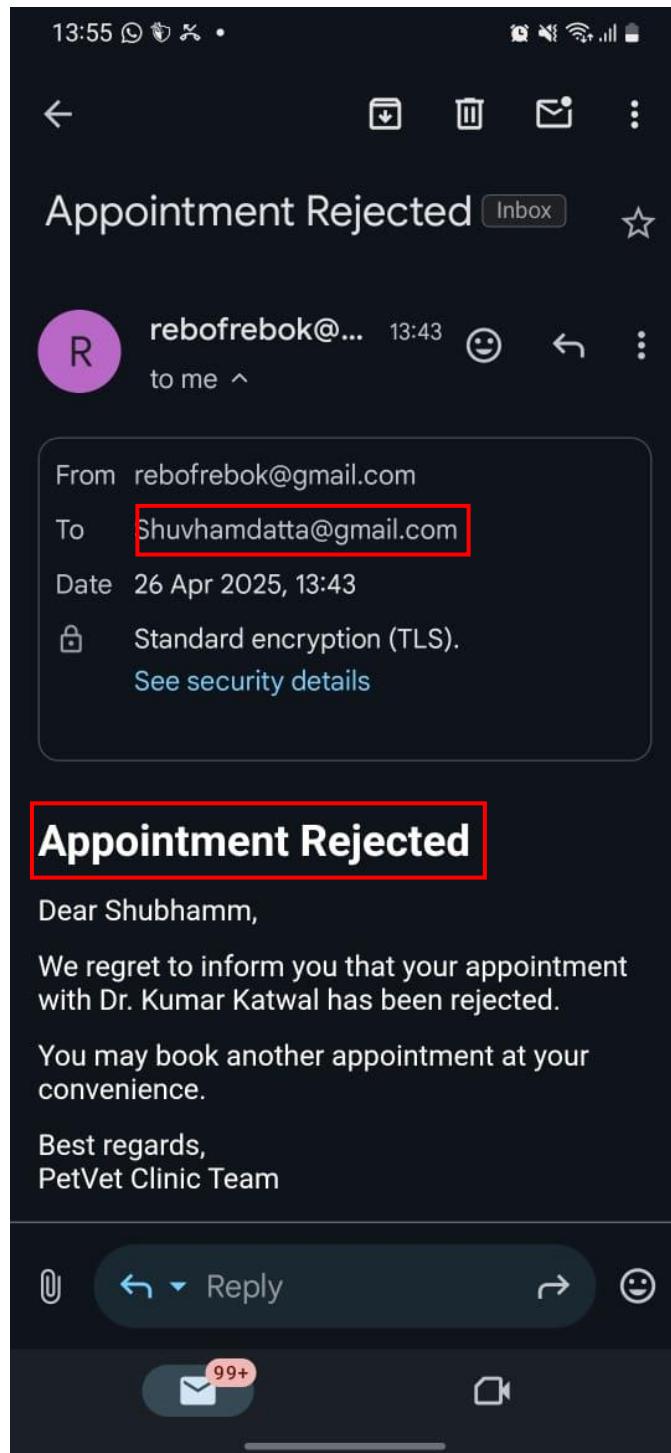
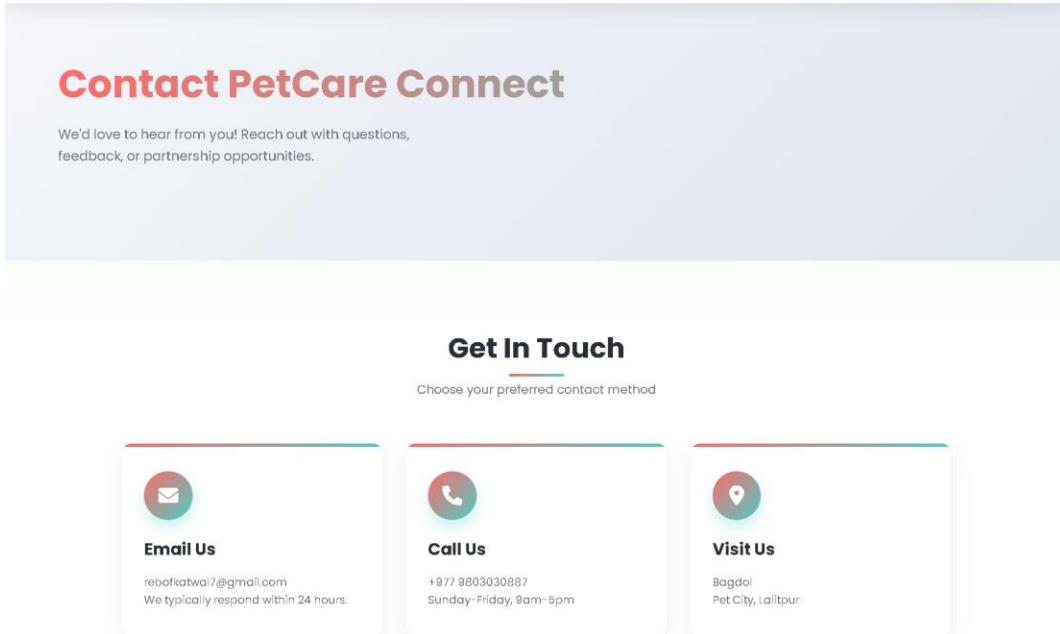


Figure 391: Rejection email sent to Shubham



Contact PetCare Connect

We'd love to hear from you! Reach out with questions, feedback, or partnership opportunities.

Get In Touch

Choose your preferred contact method



Email Us

rebofkatwal7@gmail.com
We typically respond within 24 hours.



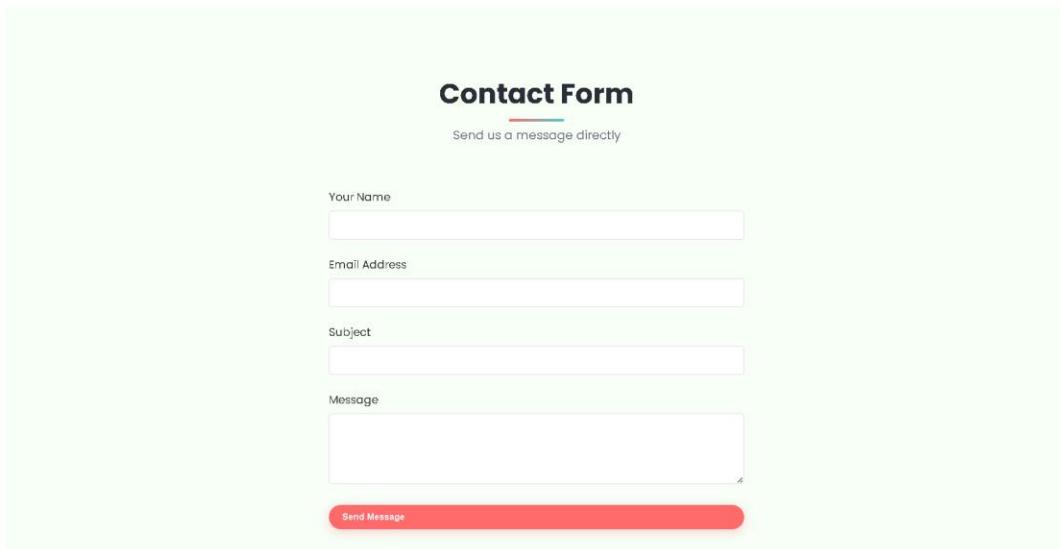
Call Us

+977 9803030887
Sunday-Friday, 9am-5pm



Visit Us

Bagdol
Pet City, Lalitpur



Contact Form

Send us a message directly

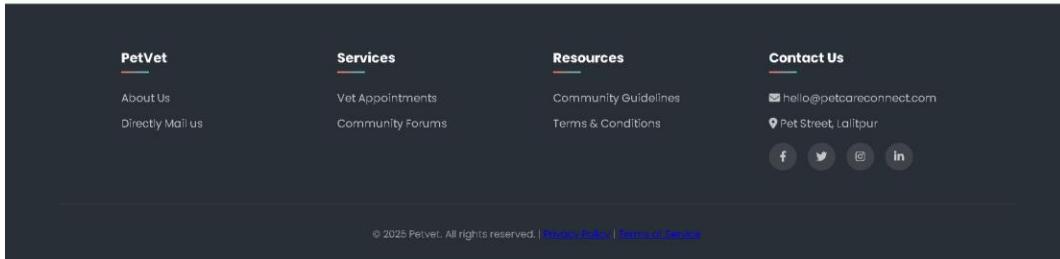
Your Name

Email Address

Subject

Message

Send Message



- PetVet**
 - [About Us](#)
 - [Directly Mail us](#)
- Services**
 - [Vet Appointments](#)
 - [Community Forums](#)
- Resources**
 - [Community Guidelines](#)
 - [Terms & Conditions](#)
- Contact Us**
 - [✉ hello@petcareconnect.com](mailto:hello@petcareconnect.com)
 - [📍 Pet Street, Lalitpur](#)

© 2025 Petvet. All rights reserved. | [Privacy Policy](#) | [Terms of Service](#)

Figure 392: Contact form for email notification checking

The screenshot shows a contact form titled "Contact Form" with a red border around the input fields. The form includes fields for "Your Name" (Asththa Thapa), "Email Address" (asththa.thapa311@gmail.com), "Subject" (Reporting a post), and a "Message" area (Hey, I want something to be removed.). A "Send Message" button is at the bottom.

Contact Form

Send us a message directly

Your Name
Astha Thapa

Email Address
asththa.thapa311@gmail.com

Subject
Reporting a post

Message
Hey, I want something to be removed.

Send Message

PetVet

About Us
Directly Mail us

Services

Vet Appointments
Community Forums

Resources

Community Guidelines
Terms & Conditions

Contact Us

hello@petcareconnect.com
Pet Street, Lalitpur

f t i

© 2025 PetVet. All rights reserved. | [Privacy Policy](#) | [Terms of Service](#)

Figure 393: Filling the contact form

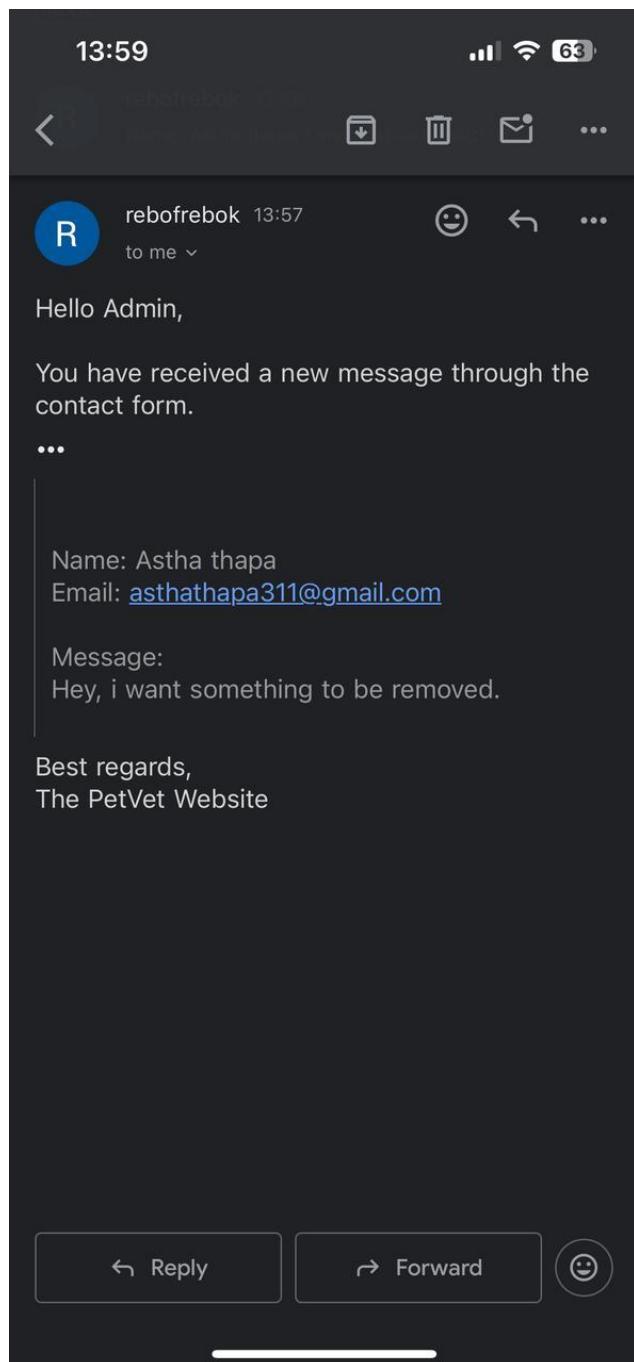


Figure 394: Admin receives the email

4.4 Critical Analysis

During unit testing phase, two crucial issues was identified and fixed.

1. **Unauthorised Access:** There was a vulnerability found for which unauthorised users could access the restricted content and execute commands that should not be executed. The solution to this issue was to put in place proper authentication and authorization checks to prevent unauthorized users from performing certain actions.
2. **Refund Logic Bug:** The logic behind the refund process was buggy and after a refund was processed, the user balance was doubled. The cause of this was traced to improper logic in the code, and the logic was fixed to account for refunds and to properly update the balance.

In the system testing phase, the update of the user profile was a problem. Changes to the profile were not immediately displayed because the browser cache was preventing it. The problem was solved by making sure the latest data is fetched and cache problems were handled accordingly.

Chapter 5: Conclusion

The VetPet web application was developed successfully in order to achieve the core objectives of improving the accessibility of veterinary care and streamlining communication between pet owners and veterinarians. The project went through a series of carefully planned iterations that included the foundational features of authentication and blogging, before reaching the status of a fully functional system serving appointment management, real time chat, admin controls and review. The features that have been implemented role specific that is user centered which has ensured that the experience of using system is entirely user centered that is based on needs and demands of users and also generated more impact to the reliability and usability purpose of the system. We demonstrated that this platform not only brings the communication gap to healthcare pet, but also provides the basis for more future enhancements, such as AI recommendations, analytics, and mobile expansion.

I was able to get a full understanding of the entire web development life cycle from the beginning of the planning and prototyping to development, testing and writing the final report, which gave me a full experience of building a real world application.

5.1 Legal, Social and Ethical Issues

5.1.1 Legal Issues

The development and deployment of the PetVet platform will need to adhere to Nepal's legal framework governing digital platforms, data of users, and online communication. Key legal considerations are:

- **Compliance to Privacy Acts:** The Nepalese Individual Privacy Act, 2018 (2075) and the Individuals' Privacy Rule, 2020 (2077) stipulate management of personal information properly. PetVet collection of user information such as names, contact details, and information relating to animals, which all constitute "personal information" under the Privacy Act—biometric information, identity number, professional opinions, and all such information.
- **Protection of Data Requirements:** Under the Privacy Act, PetVet will ensure that individual data is free from abuse, not dispersed with the users' authorization, or unsecured. It shall be protected using superior data encryption, access controls, and privacy policies to ensure the safety of the users' data.
- **Advertising and Content Control:** Since PetVet involves user-generated content via comments and posts, it comes under the Advertisement Act, 2019 (2076) and Advertisement Regulation, 2020 (2076). Any advertisement or promotional material published has to be consistent with legal norms and morality so that it does not involve misinformation or exploitation.

By adhering to these legal requirements, PetVet guarantees the users' privacy, keeps lawful data processes in place, and provides a secure and reliable user platform to engage with.

5.1.2 Social Issues

The PetVet website promotes responsible keeping of animals through facilitated communication between the veterinary professionals and owners of animals. It promotes awareness of the welfare of animals and their health through use of the community posting facility that allows free exchange of information. Nevertheless, some social issues emerge:

- **Maintaining Respectful Discussion:** With public posting, live chat, and review, inappropriate or offensive behavior can be likely to occur. Aligned community guidelines and live moderation should exist to ensure respectful discussion and prevent harassment or disinformation.
- **Personal Tragedies & Emotional Disturbance:** Discussion of sick or deceased animals can be emotionally traumatic. The site should be careful to be sensitive to this, and provide the user with an option to moderate such likely to upset material.
- **Professional Consequences and Public Scrutiny:** Vets face public reviews, which impact professional reputation. Although this ensures accountability, it imposes pressure upon the vets, which may lead to public backlash.

By helping with these social issues, PetVet strives to be a comforting, educational, and understanding venue for all members.

5.1.3 Ethical Issues

In addition to legal and social considerations, ethical issues play a crucial role in the development and operation of PetVet. The platform must ensure the responsible and ethical handling of its users and the information shared. Key ethical issues include:

- **Prevent Hate Speech and Discrimination:** The platform should not allow any kind of hate speech or discrimination. PetVet will take measures to prevent hate speech, discrimination and offensive content from being promoted or tolerated. Proactive moderation of user created content combined with following up the community guidelines which support the inclusivity and respect.
- **Copyright and Intellectual Property:** PetVet is committed to upholding copyright and intellectual property rights and makes sure that the content from the platform (whether generated or sourced from outside) is in accordance to intellectual property laws. There will be systems on the platform to prevent plagiarism and unauthorized use of copyrighted material to create a legal and ethical environment for content sharing.
- **Avoid Deceptive Practices:** PetVet will not use any deceptive practices that could deceive users into making unwanted actions like clicking on irrelevant ads or giving personal data without clear consent. It is critical to make data to data decisions transparent for the users, so they are able to make informed decisions.

5.2 Advantages

The implementation of this system brings several key benefits that enhance the overall pet care experience and streamline the interaction between pet owners and veterinary professionals:

- **Centralized Platform:** Combines community discussions, appointment scheduling, and real-time messaging in one unified system, offering a seamless user experience.
- **Improved Accessibility & Communication:** Enhances the vet-pet owner relationship, making vet services more responsive and accessible to the needs of the owners.
- **Flexible Vet Scheduling:** Vets can self-manage their weekly availability alongside having complete authority to accept or decline appointments, so that they can more effectively manage time and workloads.
- **Fair Refund System:** The store credit-based refund approach ensures fairness in appointment cancellations, balancing the needs of both vets and pet owners.
- **Transparency Through Reviews:** The vet review system fosters trust by promoting accountability and transparency in service quality.
- **Effective moderation:** The administrator dashboard offers close supervision of the site activity to make sure that the posts, comments, and interactions are of high quality and in a secure environment.
- **Promotes Timely Pet Attention:** Finally, the site sees to it that pets receive attention at the right time, fostering general well-being of the pets.

5.3 Limitations

Despite efforts to minimize errors and address logical flaws, the current version of the application has a few limitations that can be improved upon in future iterations:

- **Store Credit & Khalti Limitation:** Store credit is only granted when an appointment booked via Khalti is rejected by a vet—there are no direct refunds. This limits access for international users, as Khalti is only available within Nepal.
- **Web-Based Accessibility:** Because the current system is web-based, it may not be immediately accessible under emergency situations, particularly to mobile device users who would use instant appointment booking services.
- **Reliance on User Integrity:** This community is based on the trustworthiness of the users, who give balanced and truthful comments, reviews, and ratings. Without advanced moderation or verification systems in place, there is potential for misuse or biased feedback, which could affect the platform's credibility.

5.4 Future work

The system is complete with all the core features as initially planned. However, to further enhance usability, reach, and social impact, the following features could be considered in future iterations of the application:

- AI-Powered Symptom Checker
- Mobile Application Development
- Campaigning & Event Features
- Rescue Mission Coordination
- Real-Time Notifications Within the App
- Telemedicine Integration

Other ideas discussed in the [7.7.1 Reading for Future Work.](#)

6. References

- Adithya, P. et al., 2023. Platform for Pet Care and Maintenance. 25(2), pp. 53-64.
- AVMA.org, 2018. AVMA.org. [Online]
Available at: <https://www.avma.org/blog/study-shows-communication-gaps-between-veterinarians-and-clients>
[Accessed 17 September 2024].
- Carvalho, B. V. d. & Mello, C. H. P., 2011. Scrum Agile Product Development Method - Literature Review, Analysis, and Classification. *Product Management & Development*, 9(1), pp. 39-49.
- DaySmart Pet, 2025. *DaySmart Pet*. [Online]
Available at: <https://www.daysmart.com/pet/>
[Accessed 11 2025].
- DocPath, 2023. *DocPath*. [Online]
Available at: <https://docpath.com/art-benefits-of-digital-communication/>
[Accessed 17 September 2024].
- Doyle, R., Saville, K., Grace, D. & Campbell, A., 2021. The importance of animal welfare and Veterinary Services in a changing world. *Revue Scientifique et Technique de l'OIE*, 40(2), p. 511–527.
- Fojtik, R., 2010. *Extreme Programming in development of specific software* , Ostrava: Procedia Computer Science.
- geeksforgeeks, 2024. *geeksforgeeks*. [Online]
Available at: <https://www.geeksforgeeks.org/software-engineering-prototyping-model/#steps-of-prototyping-model>
[Accessed 11 11 2024].
- Heath For Animals, 2022. *Heath For Animals*. [Online]
Available at: <https://healthforanimals.org/reports/pet-care-report/global-trends-in-the-pet-population/>
[Accessed 17 September 2024].
- Kc, Z., 2016. *Dog business booms as pet ownership expands*, Kathmandu: The Kathmandu Post.
- López, D. C., Colca, L. G., Andrade-Arenas, L. & Cabanillas-Carbonell, M., 2023. Design of a Mobile Application for the Control of Pet. 71(3), pp. 395-406.
- Mahfouz, M. S. et al., 2023. Evaluation of Patient Satisfaction With the New Web-Based Medical Appointment Systems “Mawid” at Primary Health Care Level in Southwest Saudi Arabia. 25(2), pp. 2-9.

- Megna, M., 2024. *Forbes*. [Online]
Available at: <https://www.forbes.com/advisor/pet-insurance/pet-ownership-statistics/>
[Accessed 17 September 2024].
- Mortale, S. et al., 2024. PETBOOK: A Platform for Pet Owners for Various Features About Pets. *International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS)*, 6(11), pp. 2038-2040.
- Nirmala, K. & Dhivya, D., 2018. Study on Integration Testing and System Testing. *International Journal of Creative Research Thoughts (IJCRT)*, 6(2), p. 796.
- Pet Desk, 2025. *Pet Desk*. [Online]
Available at: <https://petdesk.com/>
[Accessed 11 2025].
- Republica, 2023. *Nepal exported pet food of Rs 2.26 billion between mid-July and mid-April, a rise of Rs 300 million*, Kathmandu: Republica.
- Runeson, P., 2006. A Survey of Unit Testing Practices. *IEEE Software*, 23(4), pp. 22-29.
- Sabale, R. G. & Dani, A., 2012. Comparative Study of Prototype Model For Software. *IOSR Journal of Engineering (IOSRJEN)*, 2(7), pp. 21-24.
- Scoresby, K. J., Strand, E. B., NG, Z. & Brown, K. C., 2021. Pet Ownership and Quality of Life: A Systematic Review of the Literature. *Veterinary Sciences (Vet Sci)*, 8(12), p. 332.
- Simplilearn, 2024. *Simplilearn*. [Online]
Available at: <https://www.simplilearn.com/what-is-kanban-article>
[Accessed 11 2025].
- Slideshare, 2023. *Slideshare.net*. [Online]
Available at: <https://www.slideshare.net/slideshow/prototyping-model-evolution-and-spiral-modelspdf/255708174>
[Accessed 20 11 2024].
- Sutipitakwong, S. & Jamsri, P., 2020. *Pros and Cons of Tangible and Digital Wireframes*. Uppsala, IEEE (Institute of Electrical and Electronics Engineers).
- Vetster, 2025. *Vetster*. [Online]
Available at: <https://vetster.com/en>
[Accessed 1 January 2025].
- Vetstoria, 2025. *Vetstoria*. [Online]
Available at: <https://www.vetstoria.com/>
[Accessed 1 January 2025].

Zhao, P. et al., 2017. Web-Based Medical Appointment Systems: A Systematic Review. 19(4).

Chapter 7: Appendix

7.1 Appendix A: Pre-Survey

7.1.1 Pre-Survey Form

7.1.2 Sample of Filled Pre-Survey Forms

Responses cannot be edited

Pre-Survey Form

Thank you for taking the time to help us! This short survey is part of a final year project aimed at developing **PetVet**, a smart platform designed to simplify pet healthcare and vet bookings online.

Please note: You don't need to have used the platform yet — we just want to learn more about your pet care habits, challenges, and expectations.

Your responses will remain anonymous and will only be used for academic research and system design improvements.

Estimated time: 3–5 minutes

* Indicates required question

1. What is your name? *
Your name will only be used for research purposes within this project.
 irvan Bir Singh Kansakar

2. What is your age group? *

- Under 18
- 18–24
- 25–34
- 35–44
- 45+

3. Do you own a pet? *

- Yes
- No

4. How long have you owned pets? *

- Less than 1 year
- 1–3 years
- 3–5 years
- More than 5 years

5. What type(s) of pets do you currently own? (Select all that apply)

- Dog
- Cat
- Bird
- Rabbit
- Other:

Figure 395: Filled Presurvey Form Sample

Section 2: Your Pet Care Habits

6. How often do you take your pet(s) to the vet? *

Regularly (every 3–6 months)
 Only when they are sick
 Rarely
 Never

7. Have you ever used a digital platform to book vet appointments? *

Yes
 No

8. Do you face difficulties in finding or booking a vet? *

Yes
 Sometimes
 No

9. What method do you currently use to find a vet? *

Google search
 Social media or forums
 Friend/Family recommendation
 Local vet I know
 I don't actively look for vets

10. How important is it for you to access vet services online? (Scale: 1 = Not important, 5 = Very important) *

1	2	3	4	5	
Not important	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Important

11. Would you be interested in a platform that lets you: (Select all that apply) *

Book appointments online
 View vet availability
 Chat with a vet online or other pet owners
 Post pet related content(blogs, images, videos)

Figure 396: Filled Presurvey Form Sample 2

Section 3: Online Preferences

12. How tech-savvy are you? (Scale: 1 = Not at all, 5 = Very tech-savvy)

1	2	3	4	5		
Not at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Very tech-savvy

13. Would you use a Reddit-style feature where pet owners and vets can post, comment, and discuss topics? *

Yes
 No
 Maybe

14. What kind of content would you like to see in a pet care community/forum? *

Vet advice
 Pet care tips
 Funny/cute pet stories
 Emergency help
 Adoption stories
 Other: _____

15. Would you personally post or ask questions in a community forum? *

Yes
 No
 Occasionally

16. How often do you visit pet-related pages on social media or forums?

Daily
 Weekly
 Occasionally
 Never

Section 4: Expectations & Feature Ideas

17. Would you trust an online platform for pet healthcare management? *

Yes
 No
 Maybe

18. Would you still use PetVet if you were not a vet or a pet owner?

Yes, I'd still find value in it
 Maybe, depending on the features
 No, it wouldn't be useful for me

19. What is the biggest challenge you face in caring for your pet? (Open text)
 Having them emotionally attached. They follow me anywhere I go which could give me problems if I ever have to go away from them for longer durations (1 year +)

20. What additional features or tools would you like to see in an online pet care platform like PetVet?
 (Note: PetVet already includes real-time chat, email notifications for appointment changes, a blog-style post system, vet scheduling, and reviews.)
 Please share any ideas you think would improve your experience.

Figure 397: Filled Presurvey Form Sample 3

7.1.3 Pre-Survey Result

2. What is your age group?

26 responses

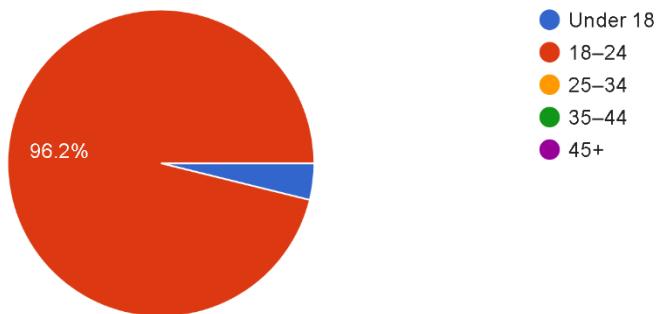


Figure 398: Presurvey Form Question 2

3. Do you own a pet?

26 responses

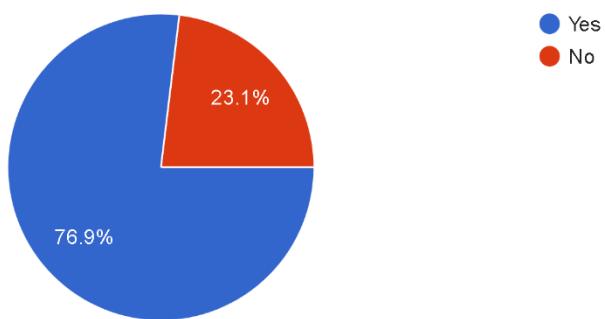
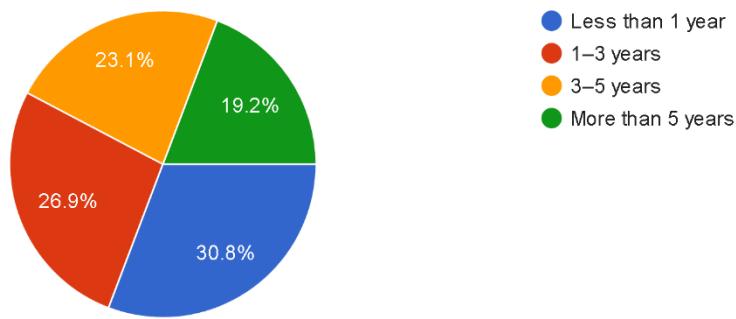


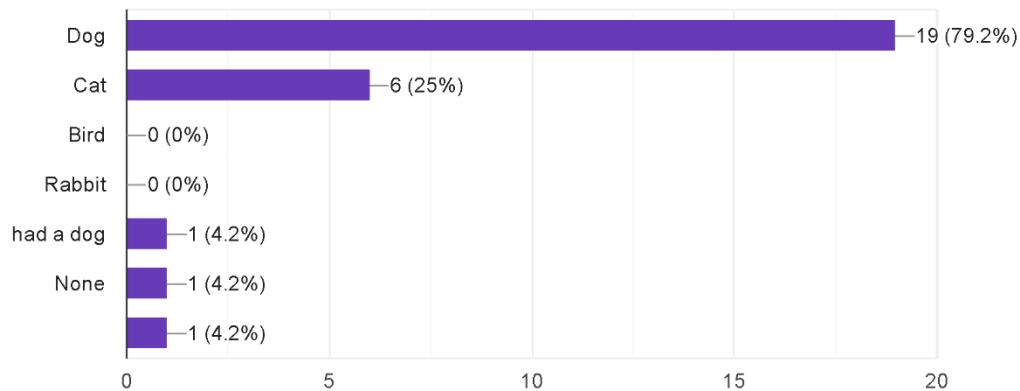
Figure 399: Presurvey Form Question 3

4. How long have you owned pets?

26 responses

*Figure 400: Presurvey Form Question 4***5. What type(s) of pets do you currently own? (Select all that apply)**

24 responses

*Figure 401: Presurvey Form Question 5*

6. How often do you take your pet(s) to the vet?

26 responses

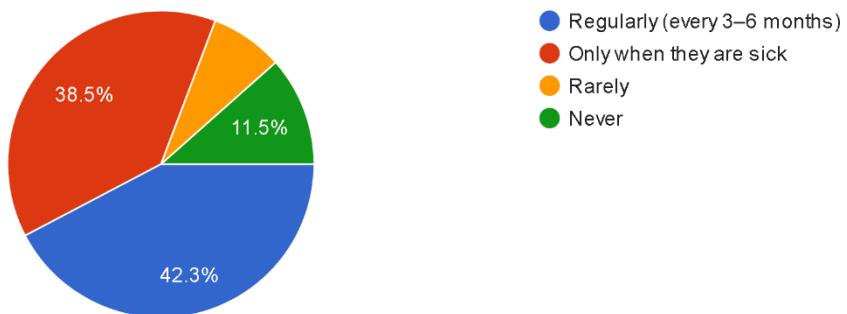


Figure 402: Presurvey Form Question 6

7. Have you ever used a digital platform to book vet appointments?

26 responses

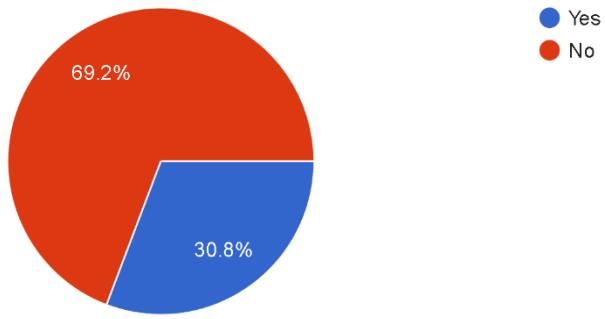


Figure 403: Presurvey Form Question 7

8. Do you face difficulties in finding or booking a vet?

26 responses

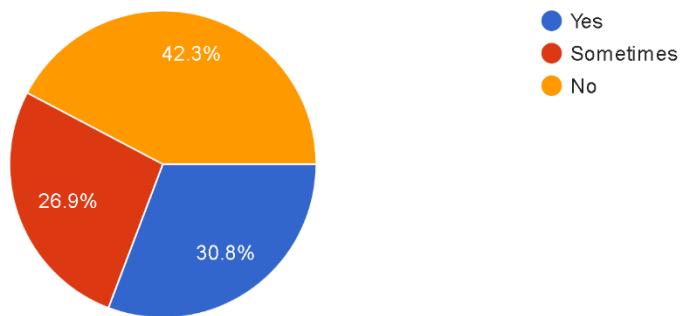


Figure 404: Presurvey Form Question 8

9. What method do you currently use to find a vet?

26 responses

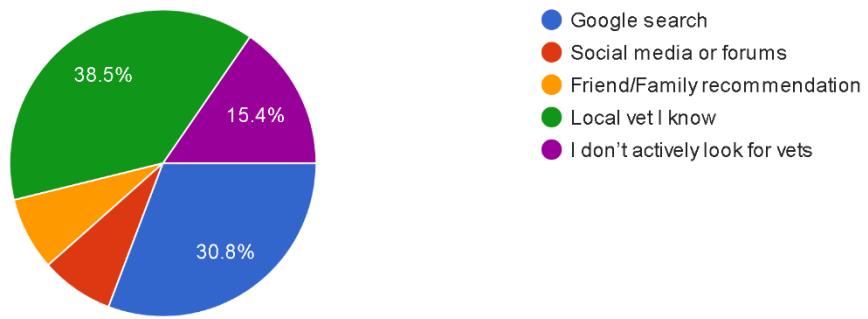


Figure 405: Presurvey Form Question 9

10. How important is it for you to access vet services online? (Scale: 1 = Not important, 5 = Very important)

26 responses

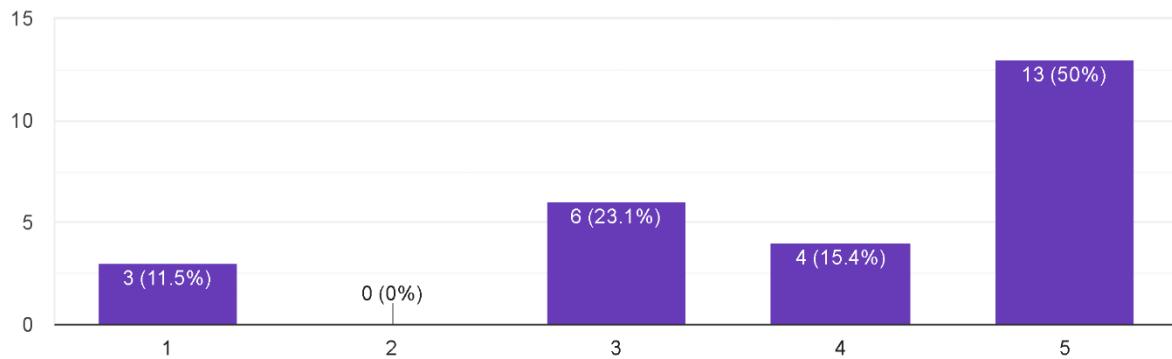


Figure 406: Presurvey Form Question 10

11. Would you be interested in a platform that lets you: (Select all that apply)

26 responses

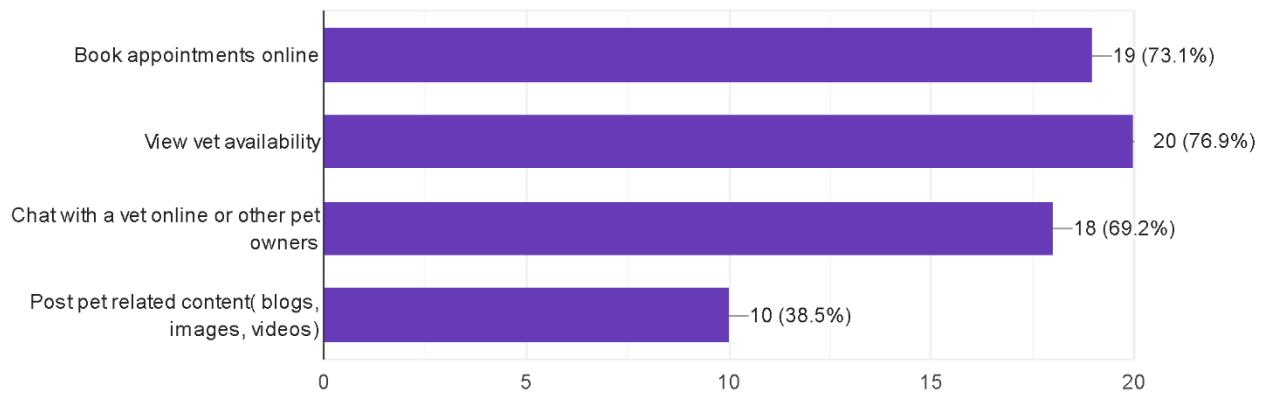


Figure 407: Presurvey Form Question 11

12. How tech-savvy are you? (Scale: 1 = Not at all, 5 = Very tech-savvy)

26 responses

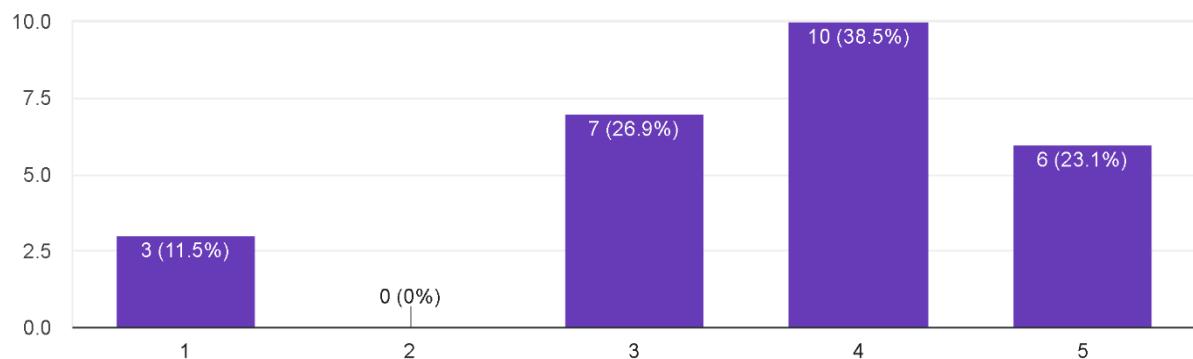


Figure 408: Presurvey Form Question 12

13. Would you use a Reddit-style feature where pet owners and vets can post, comment, and discuss topics?

26 responses

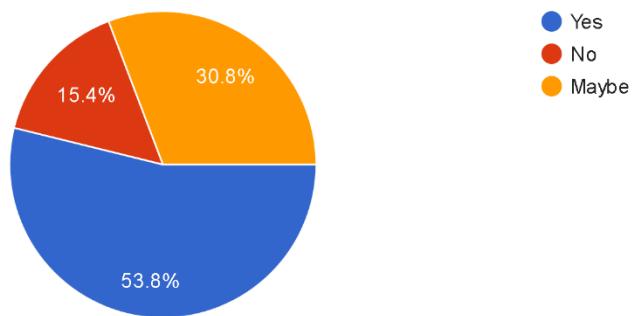


Figure 409: Presurvey Form Question 13

14. What kind of content would you like to see in a pet care community/forum?

26 responses

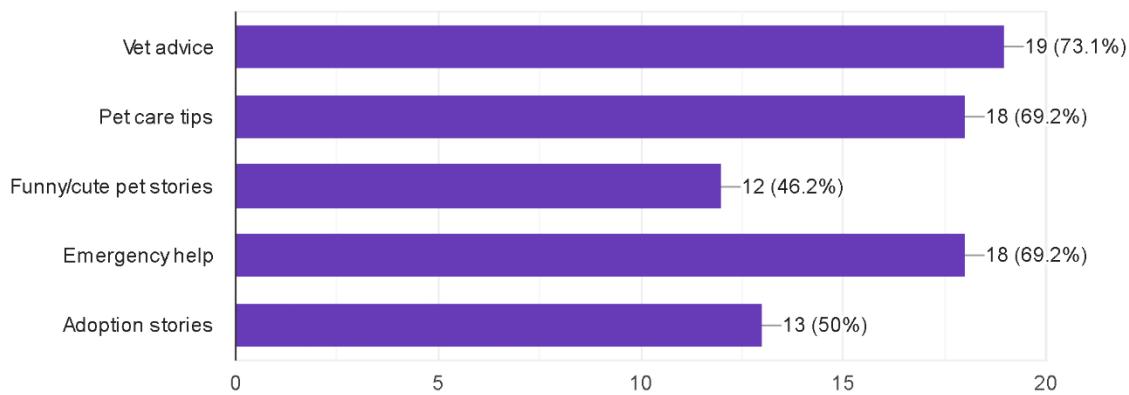


Figure 410: Presurvey Form Question 14

15. Would you personally post or ask questions in a community forum?

26 responses

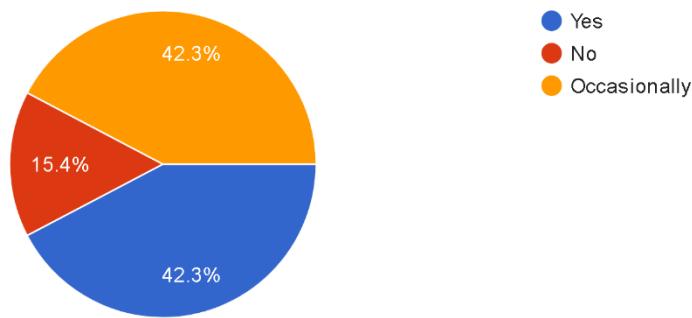


Figure 411: Presurvey Form Question 15

16. How often do you visit pet-related pages on social media or forums?

26 responses

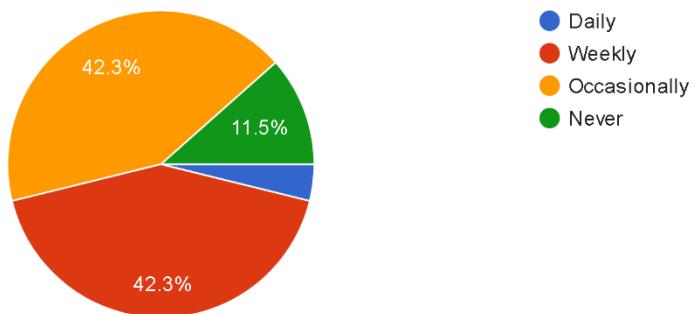


Figure 412: Presurvey Form Question 16

17. Would you trust an online platform for pet healthcare management?

26 responses

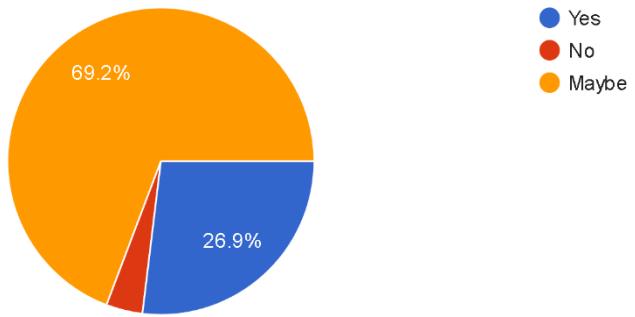


Figure 413: Presurvey Form Question 17

18. Would you still use PetVet if you were not a vet or a pet owner?

25 responses

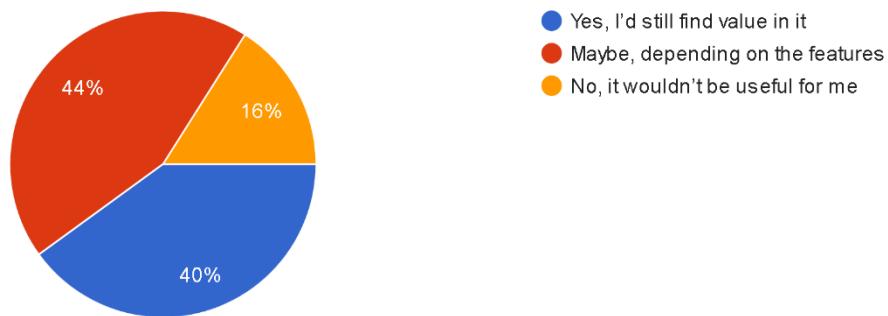


Figure 414: Presurvey Form Question 18

Here are the key findings from the pre-survey:

1. Demographics

i. Age group:

- 96% were aged 18 or older.
- The rest were under 18.

ii. Pet ownership:

- 76% currently own a pet.
- 23% do not own a pet.

iii. Duration of pet ownership:

- 30% owned pets for less than 1 year.
- 26% owned pets for 1–3 years.
- 23% for 2–5 years.
- 19.2% for more than 5 years.

iv. Types of pets owned:

- 79% own dogs.
- 25% own cats.
- 4.2% previously had dogs but currently have no pets.
- 4.2% currently have no pets.

2. Current Behavior and Preferences

i. Frequency of vet visits:

- 42% visit the vet regularly.
- 38% visit only when their pets are sick.
- 11% never visit the vet.
- The rest visit rarely.

ii. Experience with digital vet booking platforms:

- 69% have never used a digital platform to book vet appointments.
- 30% have used one.

iii. Challenges in finding or booking vets:

- 30% said yes they face difficulties.
- 26% said sometimes.
- 42% said no.

iv. Methods used to find vets:

- 38.5% use local vet clinics.
- 30% search on Google.
- 7% use social media.

- 7% rely on friends and family.
- 15% do not actively search for vets.

3. Interest in PetVet Features

i. Importance of accessing vet services online (1–5 scale):

- 50% rated it 5 (Very important).
- 15% rated it 4.
- 23% rated it 3.
- 11% rated it 1 (Not important).

ii. Features users are interested in:

- 76% want to view vet availability.
- 73% want to book appointments online.
- 69% are interested in chatting with vets or other owners.
- 38% would like to post pet-related content.

iii. Tech-savviness of users (1–5 scale):

- 23% rated themselves 5 (Very tech-savvy).
- 38% rated 4.
- 26% rated 3.
- 11% rated 1 (Not tech-savvy).

4. Community Forum Insights

i. Interest in a Reddit-style pet care community:

- 53% said yes.
- 30% said maybe.

- 15% said no.

ii. Desired community content:

- 73% want vet advice.
- 69% want pet care tips.
- 69% want emergency help posts.
- 50% want adoption stories.
- 46% want funny or cute pet stories.

iii. Willingness to post or ask questions:

- 42% said yes.
- 40% said maybe.
- 15% said no.

iv. Frequency of visiting pet-related pages:

- 42% visit occasionally.
- 42% visit weekly.
- 11% never visit such pages.
- 3% visit daily.

5. Trust in Online Platforms

i. Would trust an online platform for pet healthcare management:

- 69% said maybe.
- 26% said yes.
- 3% said no.

ii. Would still use PetVet if not a pet owner or vet:

- 40% said yes.
- 44% said maybe, depending on the features.
- 16% said no.

6. Key Challenges in Pet Care (Open-ended responses)

Some of the biggest challenges users face:

- Understanding the correct diet and food allergies for pets.
- Finding vets quickly during emergencies.
- Emotional attachment and separation anxiety.
- Training and obedience challenges.
- Managing pet medications and vaccinations.
- Lack of nearby vet services.

7. Suggested Features for PetVet

User recommended:

- Selling pet-related items (food, toys, grooming supplies).
- Adding live chat functionality (already planned for PetVet).
- Gradually expanding features based on user needs.

7.2 Appendix B: Post-Survey

7.2.1 Post-Survey Form

7.2.2 Sample of Filled Post-Survey Forms

Responses cannot be edited

Post-Survey Form

Thank you for using PetVet, a smart platform for pet owners and vets to connect and manage care. This short survey is part of my final year project to evaluate how users experienced the platform and what we can improve.

Whether you're a pet owner, vet, or general user, your feedback is valuable. Your responses will remain anonymous and used only for academic and performance analysis.

* indicates required question

1. What is your name? *

Radhika Shrestha

2. What is your age group? *

Under 18
 18–24
 25–34
 35–44
 45+

Section 2: Your Experience

3. Did you use PetVet as a: *

Pet Owner
 Vet
 General User/Tester

4. Did you explore the Reddit-style posting and commenting system? *

Yes
 No
 Just briefly

5. How helpful was the category-based post system with title, body, and image support? (1 = Not helpful, 5 = Very helpful) *

1 2 3 4 5
 Not helpful Very helpful

6. Did you find the appointment booking and vet schedule system easy to use? *

Yes
 No
 Somewhat

Figure 415: Filled Post survey Form

7. Was the payment and refund process (including store credit logic) clear and fair? *

Strongly disagree
 Disagree
 Neutral
 Agree
 Strongly agree
 I didn't use this feature

8. Did the real-time chat feature enhance your experience or help you connect? *

Yes
 No
 A little

9. Did you use the review system after a completed appointment? *

Yes
 No
 I didn't book an appointment

10. Did the admin monitoring system make you feel safer using the platform? *

Yes
 No
 Somewhat

11. Would you continue using PetVet if it became a public app? *

Yes
 No
 Maybe

Section 3: Feedback & Future Improvements

13. Do you have any feedback, suggestions, or issues you noticed while using PetVet?
 Its very helpful my pet's health needs were addressed with great care and i felt well informed throughout the process

14. Which of the following would you like to see in future updates? (Select all that apply) *

Mobile app version
 Video consultations with vets
 Push notifications
 Pet vaccination tracking
 None
 Other: _____

Figure 416: Filled Post survey Form 2

7.2.3 Post-Survey Result

2. What is your age group?

33 responses

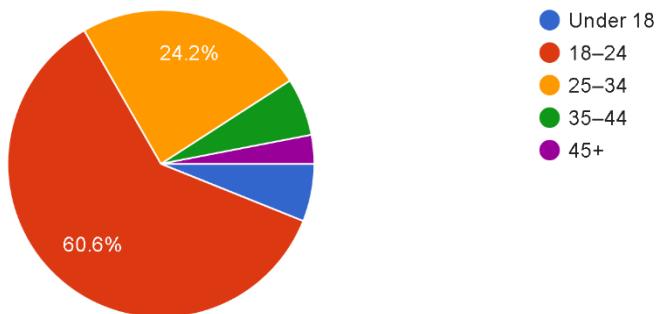


Figure 417: Post survey Form Question 1

3. Did you use PetVet as a:

33 responses

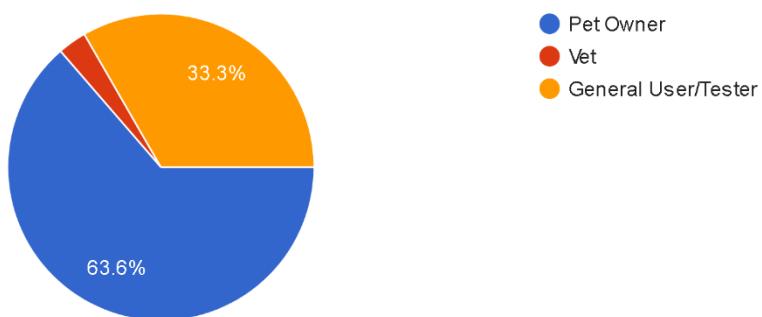


Figure 418: Post survey Form Question 3

4. Did you explore the Reddit-style posting and commenting system?

33 responses

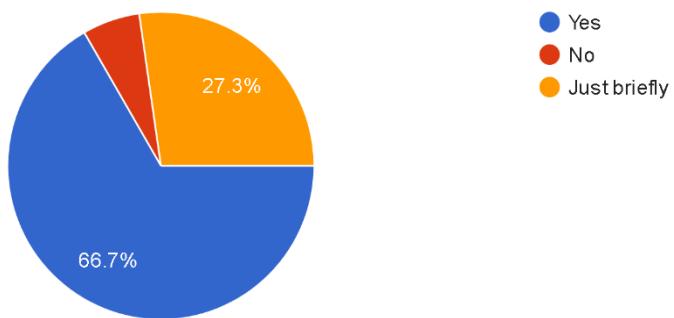


Figure 419: Post survey Form Question 4

5. How helpful was the category-based post system with title, body, and image support? (1 = Not helpful, 5 = Very helpful)

33 responses

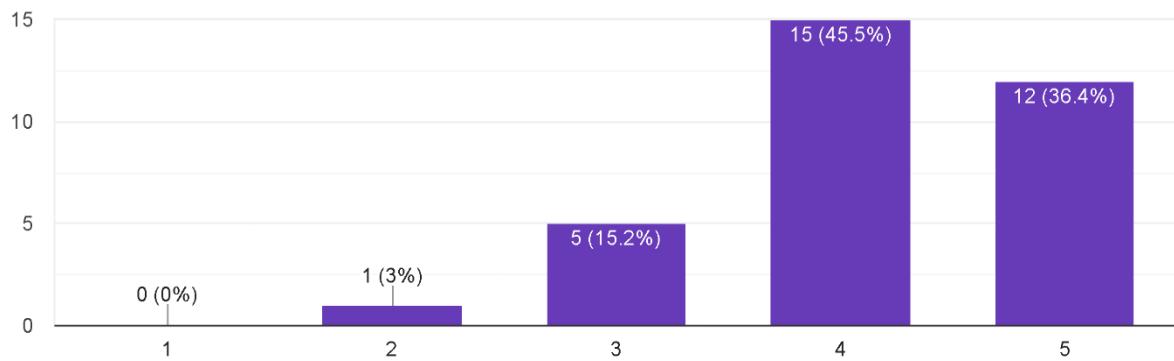


Figure 420: Post survey Form Question 5

6. Did you find the appointment booking and vet schedule system easy to use?

33 responses

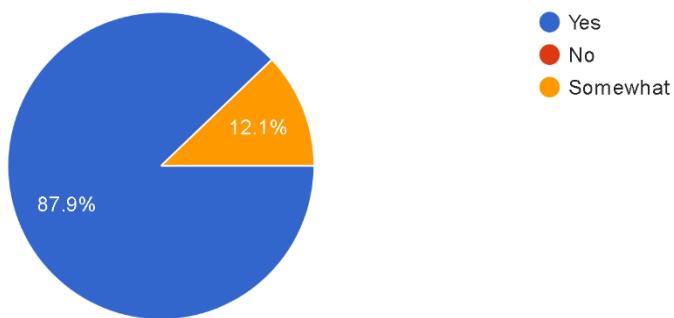


Figure 421: Post survey Form Question 6

7. Was the payment and refund process (including store credit logic) clear and fair?

33 responses

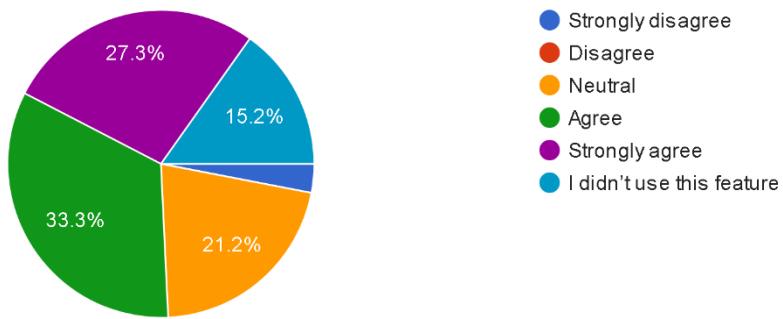


Figure 422: Post survey Form Question 7

8. Did the real-time chat feature enhance your experience or help you connect?

33 responses

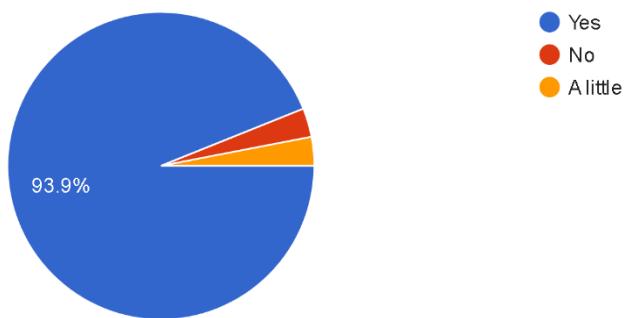


Figure 423: Post survey Form Question 8

9. Did you use the review system after a completed appointment?

33 responses

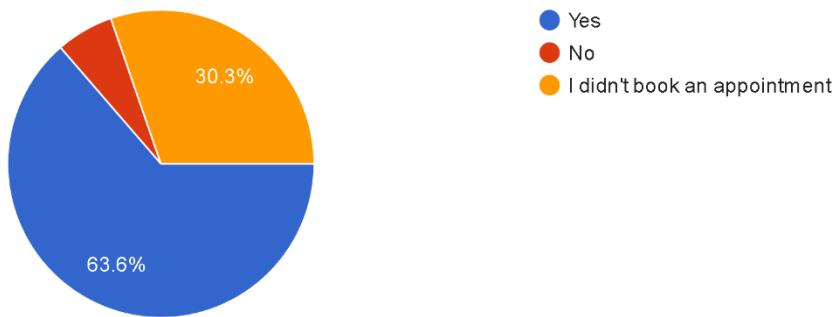


Figure 424: Post survey Form Question 9

10. Did the admin monitoring system make you feel safer using the platform?

33 responses

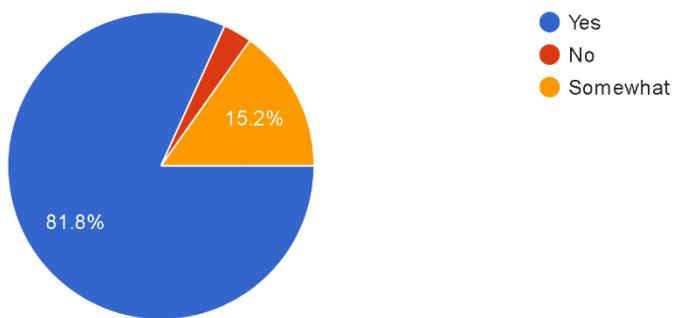


Figure 425: Post survey Form Question 10

11. Would you continue using PetVet if it became a public app?

33 responses

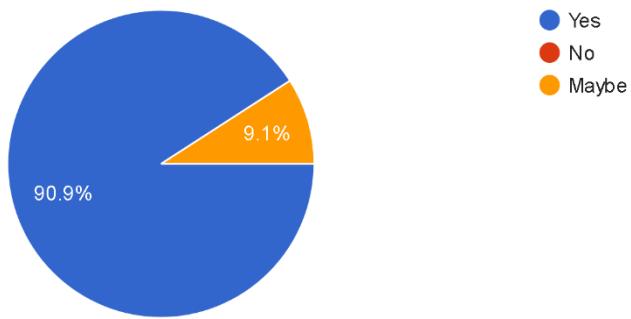


Figure 426: Post survey Form Question 11

13. Do you have any feedback, suggestions, or issues you noticed while using PetVet?

9 responses

Its very helpful my pet's health needs were addressed with great care and i felt well informed throughout the process

I really liked the quick access to pet health records and appointment scheduling

The structure and content of the website are very good. Improving a few design elements or adding more interactive features could make it even better.

The website looks great overall! One small suggestion would be to maybe add a bit more color or images to make it even more engaging

The website loads quickly and is easy to use. The menu is clear, and everything feels intuitive. It was a pleasant browsing experience.

The website has a clean and professional design. It's easy to navigate, and the content is well-organized.

Its easy to book appointments, get reminders and access health records
i had great experience using petvet

Figure 427: Post survey Form Question 13

14. Which of the following would you like to see in future updates? (Select all that apply)

33 responses

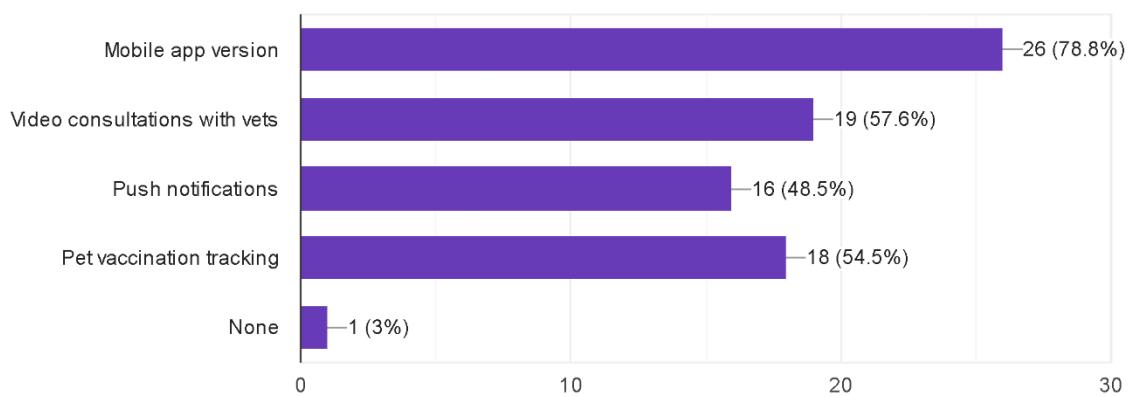


Figure 428: Post survey Form Question 14

After launching the initial version of PetVet, a post-survey was conducted to gather feedback on the platform's features, usability, and overall user experience.

Here are the key findings from the post-survey:

1. Target Audience Demographics:

- The largest group of users (60%) were between 18–24 years old, followed by 24% between 25–34 years.
- Smaller groups included users under 18 (6%), aged 35–44 (6.1%), and 45+ (3%).

2. Type of Users on PetVet:

- 63% of the participants were actual pet owners.
- 33.3% were general users or testers.
- 3% were practicing vets.

3. Engagement with Reddit-Style Posting and Commenting System:

- 66% of users fully explored the posting system.
- 27% explored it only briefly.
- 6.1% did not use this feature.

4. Helpfulness of Category-Based Post System

- 36.4% rated it as very helpful (5/5).
- 45% rated it 4/5.
- 15% gave it a neutral score (3/5).
- Only 3% found it slightly helpful (2/5).

5. Ease of Appointment Booking and Vet Scheduling

- 87% found the appointment and scheduling system easy to use.

- 12% said it was somewhat easy.

6. Clarity and Fairness of Payment and Refund System (Including Store Credit):

- 15% of users did not use this feature.
- Among those who did: 33% agreed it was clear and fair, 27% strongly agreed, and 21% remained neutral.
- Only 3% strongly disagreed.

7. Real-Time Chat Feature Experience

- 93% said the real-time chat feature improved their experience.
- 3% said it helped "a little"
- 3% said it did not help.

8. Usage of Appointment Review System:

- 63% submitted a review after completing their appointment.
- 30% said they didn't book an appointment at all.
- 6% did not leave a review.

9. Feeling of Safety through Admin Monitoring:

- 81% of users felt safer because of the admin monitoring system.
- 15% felt somewhat safer.
- 3% felt no difference.

10. Willingness to Continue Using PetVet:

- 90.9% stated they would continue using PetVet if it launched publicly.
- 9.1% were unsure (maybe).

11. Most Requested Future Features:

- 78% requested a dedicated mobile app version.
- 57% wanted video consultations with vets.
- 54% requested pet vaccination tracking.
- 48% asked for push notifications.
- Only 3% said no additional features were needed.

7.3 Appendix C: Unit Test Codes

7.3.1 Code of the Automation Script

7.3.1.1 Script Unit Test Auth1

```

10  class RegisterViewTest(TestCase):
11      def setUp(self):
12          self.factory = RequestFactory()
13          self.register_url = reverse('authUser:register')
14          self.valid_data = {
15              'full_name': 'Test User',
16              'email': 'rebof@example.com',
17              'username': 'testuser',
18              'phone': '1234567890',
19              'gender': 'male',
20              'user_type': 'pet_owner',
21              'password1': 'testpassword123',
22              'password2': 'testpassword123',
23          }
24

```

Figure 429: Unit Test Auth1 Setup

```

25  def test_register_view_get(self):
26      '''renders the proper template'''
27      response = self.client.get(self.register_url)
28      self.assertEqual(response.status_code, 200)
29      self.assertTemplateUsed(response, 'authUser/register.html')
30

```

Figure 430: Unit Test Auth1 Automation Script 1

```

def test_successful_pet_owner_registration(self):
    '''registration checking for pet owner'''
    response = self.client.post(self.register_url, data=self.valid_data)
    self.assertEqual(response.status_code, 302)

    # Check redirection
    self.assertRedirects(response, reverse('complete-profile'))

    # Check user creation
    user = User.objects.get(email='rebof@example.com')
    self.assertEqual(user.full_name, 'Test User')
    self.assertEqual(user.user_type, 'pet_owner')

    # Check profile creation
    self.assertTrue(PetOwnerProfile.objects.filter(user=user).exists())

    # Check OTP email
    self.assertEqual(len(mail.outbox), 1)
    self.assertIn(str(user.otp), mail.outbox[0].body)

# Check messages - now looking at both messages
messages = list(get_messages(response.wsgi_request))
self.assertEqual(str(messages[0]), "An OTP has been sent to your email. Please verify to proceed.")
self.assertEqual(str(messages[1]), "Hi Test User, your account was created successfully.")

```

Figure 431: Unit Test Auth1 Automation Script 2

```

def test_successful_vet_registration(self):
    '''registration checking for vet'''
    vet_data = self.valid_data.copy()
    vet_data.update({
        'user_type': 'vet',
        'email': 'votrebof@example.com',
        'username': 'testvet'
    })

    response = self.client.post(self.register_url, data=vet_data)

    # Check vet profile creation
    user = User.objects.get(email='votrebof@example.com')
    self.assertTrue(VetProfile.objects.filter(user=user).exists())

```

Figure 432: Unit Test Auth1 Automation Script 3

```

def test_password_mismatch(self):
    '''testing the two passwords for a match'''
    invalid_data = self.valid_data.copy()
    invalid_data['password2'] = 'wrongpassword'

    response = self.client.post(self.register_url, data=invalid_data)

    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), "Passwords do not match.")

```

Figure 433: Unit Test Auth1 Automation Script 4

```

def test_existing_email(self):
    '''checking the uniqueness of the email'''
    User.objects.create_user(
        email='existinguser@example.com',
        username='existinguser',
        password='testpass123'
    )

    invalid_data = self.valid_data.copy()
    invalid_data['email'] = 'existinguser@example.com'

    response = self.client.post(self.register_url, data=invalid_data)

    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), "Email already exists.")

```

Figure 434: Unit Test Auth1 Automation Script 5

8.3.1.2 Script of Unit Test Auth2

```
class LoginViewTest(TestCase):
    def setUp(self):
        self.client = Client()
        self.login_url = reverse('authUser:loginUser')
        self.register_url = reverse('authUser:register')

        # Create test user with complete profile
        self.valid_user = User.objects.create_user(
            email='rebof@gmail.com',
            username='rebofkatwal',
            password='testkatwal123',
            full_name='Rebof Katwal',
            profile_completed=True,
            status_verification=True
        )
```

Figure 435: Unit Test Auth2 Setup

```
def test_login_view_get(self):
    """Test 1: Verify login page loads correctly"""
    response = self.client.get(self.login_url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'authUser/register.html')
```

Figure 436: Unit Test Auth2 Automation Script 1

```
def test_successful_login(self):
    """Test 2: Verify successful login with correct credentials"""
    response = self.client.post(self.login_url, {
        'email': 'rebof@gmail.com',
        'password': 'testkatwal123'
    }, follow=True)

    self.assertTemplateUsed(response, 'coreFunctions/index.html')
```

Figure 437: Unit Test Auth2 Automation Script 2

```

def test_invalid_credentials(self):
    """Test 3: Verify error handling for wrong password"""
    response = self.client.post(self.login_url, {
        'email': 'rebof@gmail.com',
        'password': 'wrongpassword'
    }, follow=True)
    messages = list(get_messages(response.wsgi_request))
    self.assertIn("Invalid username or password", [str(m) for m in messages])
    self.assertTemplateUsed(response, 'authUser/register.html')

```

Figure 438: Unit Test Auth2 Automation Script 3

```

def test_incomplete_profile_redirect(self):
    """Test 4: Verify redirect for users with incomplete profiles"""
    incomplete_user = User.objects.create_user(
        email='incomplete@gmail.com',
        username='incomplete',
        password='testkatwal123',
        full_name='Incomplete User',
        profile_completed=False
    )
    response = self.client.post(self.login_url, {
        'email': 'incomplete@gmail.com',
        'password': 'testkatwal123'
    }, follow=True)
    self.assertTemplateUsed(response, 'authUser/profile.html')

```

Figure 439: Unit Test Auth2 Automation Script 4

```

def test_already_authenticated_redirect(self):
    """Test 5: Verify redirect for already authenticated users"""
    self.client.force_login(self.valid_user)
    response = self.client.get(self.login_url, follow=True)
    # Should redirect to index page
    self.assertRedirects(response, reverse('coreFunctions:index'))

```

Figure 440: Unit Test Auth2 Automation Script 5

8.3.1.3 Script of Unit Test Auth3

```
class AdminLoginTest(TestCase):
    def setUp(self):
        self.client = Client()
        self.login_url = reverse('django_admin:admin_login')

        # Create admin user
        self.admin_user = User.objects.create_user(
            email='admin@gmail.com',
            password='admin123',
            username='adminuser',
            is_staff=True
        )

        # Create regular user
        self.regular_user = User.objects.create_user(
            email='user@gmail.com',
            username='regular',
            password='user123',
            is_staff=False
        )
```

Figure 441: Unit Test Auth3 Setup

```
def test_admin_login_view_get(self):
    """Test GET request to admin login page"""
    response = self.client.get(self.login_url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'django_admin/login.html')
```

Figure 442: Unit Test Auth3 Automation Script 1

```

def test_successful_admin_login(self):
    """Test successful admin login"""
    response = self.client.post(self.login_url, {
        'email': 'admin@gmail.com',
        'password': 'admin123'
    }, follow=True)

    # Check redirect to admin dashboard
    self.assertRedirects(response, reverse('django_admin:admin_dashboard'))
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), "Login successful!")

```

Figure 443: Unit Test Auth3 Automation Script 2

```

def test_regular_user_login_attempt(self):
    """Test regular user cannot access admin panel"""
    response = self.client.post(self.login_url, {
        'email': 'user@gmail.com',
        'password': 'user123'
    }, follow=True)

    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), "Invalid email or password or insufficient permissions")
    self.assertTemplateUsed(response, 'django_admin/login.html')

```

Figure 444: Unit Test Auth3 Automation Script 3

```

def test_invalid_credentials(self):
    """Test invalid login credentials"""
    response = self.client.post(self.login_url, {
        'email': 'admin@gmail.com',
        'password': 'wrongpassword'
    }, follow=True)

    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), "Invalid email or password or insufficient permissions")
    self.assertTemplateUsed(response, 'django_admin/login.html')

```

Figure 445: Unit Test Auth3 Automation Script 4

```
def test_missing_credentials(self):
    """Test missing credentials"""
    response = self.client.post(self.login_url, {
        'email': '',
        'password': ''
    }, follow=True)

    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), "Invalid email or password or insufficient permissions")
```

Figure 446: Unit Test Auth3 Automation Script 5

8.3.1.4 Script of Unit Test Admin1

```
class ApproveVetTestCase(TestCase):
    def setUp(self):
        self.client = Client()

        # Create admin user (must have is_staff=True)
        self.admin_user = User.objects.create_user(
            email='admin@example.com',
            username='admin',
            password='testpass123',
            user_type='admin',
            profile_completed=True,
            status_verification=True,
            is_staff=True
        )

        # Create unverified vet user
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=False,
            full_name='Dr. Vet User'
        )
        self.vet_profile = self.vet_user.vetprofile
```

Figure 447: Unit Test Admin1 Automation Setup

```

def test_approve_vet_success(self):
    """Test successful vet approval by admin"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:approve_vet', args=[self.vet_profile.id])
    )

    self.vet_user.refresh_from_db()
    self.vet_profile.refresh_from_db()

    self.assertTrue(self.vet_user.status_verification)
    self.assertTrue(self.vet_profile.verified)
    self.assertRedirects(response, reverse('django_admin:vet_approvals'))

```

Figure 448: Unit Test Admin1 Automation Script 1

```

def test_approve_vet_unauthorized_redirect(self):
    """Test non-admin users are redirected to login"""
    # Create regular non-staff user
    regular_user = User.objects.create_user(
        email='user@example.com',
        username='regular',
        password='testpass123',
        user_type='pet_owner',
        is_staff=False,
        profile_completed=True,
        status_verification=True
    )

    self.client.force_login(regular_user)
    response = self.client.post(
        reverse('django_admin:approve_vet', args=[self.vet_profile.id]),
        follow=True # Follow the redirect
    )

    # Should redirect to admin login
    self.assertRedirects(
        response,
        reverse('django_admin:admin_login'),
        status_code=302,
        target_status_code=200
    )

    # Check vet was not approved
    self.vet_user.refresh_from_db()
    self.assertFalse(self.vet_user.status_verification)

```

Figure 449 Unit Test Admin1 Automation Script 2

```
def test_approve_vet_unauthenticated_redirect(self):
    """Test unauthenticated users are redirected to login"""
    response = self.client.post(
        reverse('django_admin:approve_vet', args=[self.vet_profile.id]),
        follow=True
    )

    self.assertRedirects(
        response,
        reverse('django_admin:admin_login'),
        status_code=302,
        target_status_code=200
    )

    # Check vet was not approved
    self.vet_user.refresh_from_db()
    self.assertFalse(self.vet_user.status_verification)
```

Figure 450 Unit Test Admin1 Automation Script 3

8.3.1.5 Script of Unit Test Admin2

```
class DeclinevetTestCase(TestCase):
    def setUp(self):
        self.client = Client()

        # Create admin user (must have is_staff=True)
        self.admin_user = User.objects.create_user(
            email='admin@example.com',
            username='admin',
            password='testpass123',
            user_type='admin',
            profile_completed=True,
            status_verification=True,
            is_staff=True
        )

        # Create unverified vet user
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=False,
            full_name='Dr. Vet User'
        )
        self.vet_profile = self.vet_user.vetprofile
```

Figure 451 Unit Test Admin2 Setup

```

def test_decline_vet_email_notification(self):
    """Test email notification is sent on successful decline"""
    with patch('django_admin.views.send_mail') as mock_send_mail:
        # Configure the mock to return successfully
        mock_send_mail.return_value = 1 # Indicates 1 email sent

        self.client.force_login(self.admin_user)
        response = self.client.post(
            reverse('django_admin:decline_vet', args=[self.vet_profile.id])
        )

    # Check email was sent with correct parameters
    mock_send_mail.assert_called_once()
    args, kwargs = mock_send_mail.call_args

    self.assertIn('declined', args[1])

    # Verify the vet was still declined
    self.vet_user.refresh_from_db()
    self.assertFalse(self.vet_user.profile_completed)

    # Verify redirect
    self.assertRedirects(response, reverse('django_admin:vet_approvals'))

```

Figure 452 Unit Test Admin2 Automation Script 1

```

def test_decline_vet_unauthenticated_redirect(self):
    """Test unauthenticated users are redirected to login"""
    response = self.client.post(
        reverse('django_admin:decline_vet', args=[self.vet_profile.id]),
        follow=True
    )

    self.assertRedirects(
        response,
        reverse('django_admin:admin_login'),
        status_code=302,
        target_status_code=200
    )

    # Check vet was not declined
    self.vet_user.refresh_from_db()
    self.assertTrue(self.vet_user.profile_completed)

```

Figure 453 Unit Test Admin2 Automation Script 2

```
def test_decline_vet_unauthorized_redirect(self):
    """Test non-admin users are redirected to login"""
    # Create regular non-staff user
    regular_user = User.objects.create_user(
        email='user@example.com',
        username='regular',
        password='testpass123',
        user_type='pet_owner',
        is_staff=False,
        profile_completed=True,
        status_verification=True
    )

    self.client.force_login(regular_user)
    response = self.client.post(
        reverse('django_admin:decline_vet', args=[self.vet_profile.id]),
        follow=True # Follow the redirect
    )

    # Should redirect to admin login
    self.assertRedirects(
        response,
        reverse('django_admin:admin_login'),
        status_code=302,
        target_status_code=200
    )

    # Check vet was not declined
    self.vet_user.refresh_from_db()
    self.assertTrue(self.vet_user.profile_completed)
```

Figure 454 Unit Test Admin2 Automation Script 3

```
def test_decline_vet_success(self):
    """Test successful vet decline by admin"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:decline_vet', args=[self.vet_profile.id])
    )

    self.vet_user.refresh_from_db()
    self.vet_profile.refresh_from_db()

    # Check vet was declined
    self.assertFalse(self.vet_user.profile_completed)
    self.assertIsNotNone(self.vet_profile.status_change)
    self.assertRedirects(response, reverse('django_admin:vet_approvals'))
```

Figure 455 Unit Test Admin2 Automation Script 4

8.3.1.6 Script of Unit Test Admin3

```
class AdminDeleteContentTestCase(TestCase):
    def setUp(self):
        self.client = Client()

        # Create admin user
        self.admin_user = User.objects.create_user(
            email='admin@example.com',
            username='admin',
            password='testpass123',
            user_type='admin',
            is_staff=True,
            profile_completed=True
        )

        # Create regular user who will own the content
        self.regular_user = User.objects.create_user(
            email='user@example.com',
            username='regular',
            password='testpass123',
            user_type='pet_owner',
            full_name='Regular User'
        )

        # Create test post
        self.post = Post.objects.create(
            user=self.regular_user,
            title='Test Post',
            body='This is a test post'
        )

        # Create test comment
        self.comment = Comment.objects.create(
            user=self.regular_user,
            post=self.post,
            comment='Test comment'
        )
```

Figure 456 Unit Test Admin3 Setup

```
# Create test reply
self.reply = ReplyComment.objects.create(
    user=self.regular_user,
    comment=self.comment,
    reply='Test reply'
)
```

Figure 457 Unit Test Admin3 Setup 2

```
# Post deletion tests
def test_delete_post_success(self):
    """Test successful post deletion"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:delete_post', args=[self.post.id]),
        {'notify_user': 'false'}
    )

    # Check post was deleted
    with self.assertRaises(Post.DoesNotExist):
        Post.objects.get(id=self.post.id)

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('deleted successfully', str(messages[0]))

    # Check redirect
    self.assertRedirects(response, reverse('django_admin:post_management'))
```

Figure 458 Unit Test Admin3 Automation Script 1

```
# Comment deletion tests
def test_delete_comment_success(self):
    """Test successful comment deletion"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:delete_comment', args=[self.comment.id])
    )

    # Check comment was deleted
    with self.assertRaises(Comment.DoesNotExist):
        Comment.objects.get(id=self.comment.id)

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('deleted successfully', str(messages[0]))

    # Check redirect
    self.assertRedirects(response, reverse('django_admin:view_post', args=[self.post.id]))
```

Figure 459 Unit Test Admin3 Automation Script 2

```
# Reply deletion tests
def test_delete_reply_success(self):
    """Test successful reply deletion"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:delete_reply', args=[self.reply.id])
    )

    # Check reply was deleted
    with self.assertRaises(ReplyComment.DoesNotExist):
        ReplyComment.objects.get(id=self.reply.id)

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('deleted successfully', str(messages[0]))

    # Check redirect
    self.assertRedirects(response, reverse('django_admin:view_post', args=[self.post.id]))
```

Figure 460 Unit Test Admin3 Automation Script 3

```
# Authorization tests
def test_delete_content_unauthorized(self):
    """Test non-admin cannot delete content"""
    regular_user2 = User.objects.create_user(
        email='user2@example.com',
        username='regular2',
        password='testpass123',
        user_type='pet_owner',
        profile_completed=True,
        status_verification=True
    )

    self.client.force_login(regular_user2)

    # Try to delete post
    response = self.client.post(
        reverse('django_admin:delete_post', args=[self.post.id]),
        follow=True
    )
    self.assertRedirects(
        response,
        reverse('django_admin:admin_login'),
        status_code=302,
        target_status_code=200
    )

    # Try to delete comment
    response = self.client.post(
        reverse('django_admin:delete_comment', args=[self.comment.id]),
        follow=True
    )
    self.assertRedirects(
        response,
        reverse('django_admin:admin_login'),
        status_code=302,
        target_status_code=200
    )
```

Figure 461 Unit Test Admin3 Automation Script 4

```
# Try to delete reply
response = self.client.post(
    reverse('django_admin:delete_reply', args=[self.reply.id]),
    follow=True
)
self.assertRedirects(
    response,
    reverse('django_admin:admin_login'),
    status_code=302,
    target_status_code=200
)
```

Figure 462 Unit Test Admin3 Automation Script 4

8.3.1.7 Script of Unit Test Admin4

```
class AdminCategoryManagementTestCase(TestCase):
    def setUp(self):
        self.client = Client()

        # Create admin user
        self.admin_user = User.objects.create_user(
            email='admin@example.com',
            username='admin',
            password='testpass123',
            user_type='admin',
            is_staff=True,
            profile_completed=True
        )

        # Create test categories
        self.category1 = Category.objects.create(
            name='Pets',
            description='All about pets'
        )
        self.category2 = Category.objects.create(
            name='Veterinary',
            description='Veterinary discussions'
        )

        # Create a test post associated with category1
        self.post = Post.objects.create(
            title='Test Post',
            body='Test content',
            category=self.category1,
            user=self.admin_user
        )
```

Figure 463 Unit Test Admin4 Setup

```

def test_category_management_view(self):
    """Test the category management page loads correctly"""
    self.client.force_login(self.admin_user)
    response = self.client.get(reverse('django_admin:category_management'))

    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'django_admin/category_management.html')

    # Check categories are in context
    categories = response.context['categories']
    self.assertEqual(categories.count(), 2)

    # Check post counts are annotated
    pets_category = categories.get(name='Pets')
    self.assertEqual(pets_category.post_count, 1)

    # Check admin user is in context
    self.assertEqual(response.context['admin_user'], self.admin_user)

```

Figure 464 Unit Test Admin4 Script Automation 1

```

def test_add_category_success(self):
    """Test successfully adding a new category"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:add_category'),
        {
            'name': 'New Category',
            'description': 'New category description'
        },
        follow=True
    )

    # Check category was created
    self.assertTrue(Category.objects.filter(name='New Category').exists())

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('added successfully', str(messages[0]))

    # Check redirect
    self.assertRedirects(response, reverse('django_admin:category_management'))

```

Figure 465 Unit Test Admin4 Script Automation 2

```

def test_add_category_duplicate(self):
    """Test adding a duplicate category"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:add_category'),
        {
            'name': 'Pets', # Already exists
            'description': 'Duplicate category'
        },
        follow=True
    )

    # Check category wasn't created again
    self.assertEqual(Category.objects.filter(name='Pets').count(), 1)

    # Check error message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('already exists', str(messages[0]))

```

Figure 466 Unit Test Admin4 Script Automation 3

```

def test_edit_category_success(self):
    """Test successfully editing a category"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:edit_category', args=[self.category1.id]),
        {
            'name': 'Updated Pets',
            'description': 'Updated description'
        },
        follow=True
    )

    # Check category was updated
    updated_category = Category.objects.get(id=self.category1.id)
    self.assertEqual(updated_category.name, 'Updated Pets')
    self.assertEqual(updated_category.description, 'Updated description')

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('updated successfully', str(messages[0]))

```

Figure 467 Unit Test Admin4 Script Automation 4

```

def test_edit_category_duplicate(self):
    """Test editing a category to duplicate another category's name"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:edit_category', args=[self.category1.id]),
        {
            'name': 'Veterinary', # Already exists for category2
            'description': 'Trying to duplicate'
        },
        follow=True
    )

    # Check category wasn't updated
    category = Category.objects.get(id=self.category1.id)
    self.assertEqual(category.name, 'Pets') # Original name

    # Check error message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('already exists', str(messages[0]))

```

Figure 468 Unit Test Admin4 Script Automation 5

```

def test_delete_category_success(self):
    """Test successfully deleting a category"""
    self.client.force_login(self.admin_user)
    response = self.client.post(
        reverse('django_admin:delete_category', args=[self.category1.id]),
        follow=True
    )

    # Check category was deleted
    with self.assertRaises(Category.DoesNotExist):
        Category.objects.get(id=self.category1.id)

    # Check associated post's category was set to None
    post = Post.objects.get(id=self.post.id)
    self.assertIsNone(post.category)

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn('deleted successfully', str(messages[0]))

    # Check redirect
    self.assertRedirects(response, reverse('django_admin:category_management'))

```

Figure 469 Unit Test Admin4 Script Automation 6

```
def test_category_management_unauthorized(self):
    """Test non-admin users can't access category management"""
    regular_user = User.objects.create_user(
        email='user@example.com',
        username='regular',
        password='testpass123',
        user_type='pet_owner',
        profile_completed=True,
        status_verification=True
    )

    self.client.force_login(regular_user)

    # Test all category management views
    views = [
        reverse('django_admin:category_management'),
        reverse('django_admin:add_category'),
        reverse('django_admin:edit_category', args=[1]),
        reverse('django_admin:delete_category', args=[1]),
    ]

    for view in views:
        response = self.client.get(view, follow=True)
        self.assertRedirects(
            response,
            reverse('django_admin:admin_login'),
            status_code=302,
            target_status_code=200
        )
```

Figure 470 Unit Test Admin4 Script Automation 7

8.3.1.8 Script of Unit Test Appt1

```
class AddScheduleTests(TestCase):
    def setUp(self):
        self.client = Client()
        self.vet_user = User.objects.create_user(
            email='vet@gmail.com',
            username='Dr_Katwal',
            password='test123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        # Get the vet profile created by the signal
        self.vet_profile = self.vet_user.vetprofile
        self.client.login(email='vet@gmail.com', password='test123')
        self.url = reverse('appointment:add_schedule')
```

Figure 471 Unit Test Appt1 Setup

```
def test_successful_schedule_creation(self):
    """Test creating a valid schedule"""
    response = self.client.post(self.url, {
        'day_of_week': 'Monday',
        'start_time': '10:00',
        'end_time': '12:00',
    })
    self.assertRedirects(response, reverse('appointment:vet_schedule', kwargs={'vet_id': self.vet_profile.id}))
    self.assertTrue(VetSchedule.objects.filter(vet=self.vet_profile, day_of_week='Monday').exists())
```

Figure 472 Unit Test Appt1 Automation Script 1

```
def test_schedule_end_time_before_start_time(self):
    """Test invalid time range (end before start)"""
    response = self.client.post(self.url, {
        'day_of_week': 'Tuesday',
        'start_time': '14:00',
        'end_time': '13:00',
    })
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'End time must be after start time.')
```

Figure 473 Unit Test Appt1 Automation Script 2

```
def test_overlapping_schedule(self):
    """Test overlapping schedule detection"""
    # Create initial schedule
    VetSchedule.objects.create(
        vet=self.vet_profile,
        day_of_week='Wednesday',
        start_time=time(9, 0),
        end_time=time(11, 0),
        available=True
    )

    # Try to create overlapping schedule
    response = self.client.post(self.url, {
        'day_of_week': 'Wednesday',
        'start_time': '10:00',
        'end_time': '12:00',
    })
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'The schedule overlaps with an existing one.')
```

Figure 474 Unit Test Appt1 Automation Script 3

8.3.1.9 Script of Unit Test Appt2

```
class EditScheduleTests(TestCase):
    def setUp(self):
        self.client = Client()

        # Create vet user and profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create initial schedule
        self.schedule = Vetschedule.objects.create(
            vet=self.vet_profile,
            day_of_week='Monday',
            start_time=time(9, 0),
            end_time=time(12, 0)
        )

        # Login the vet
        self.client.login(email='vet@example.com', password='testpass123')
```

Figure 475 Unit Test Appt2 Setup

```
def test_edit_schedule_get(self):
    """Test GET request to edit schedule page"""
    response = self.client.get(reverse('appointment:edit_schedule', args=[self.schedule.id]))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/edit_schedule.html')
    self.assertEqual(response.context['schedule'], self.schedule)
```

Figure 476 Unit Test Appt2 Automation Script 1

```

def test_valid_schedule_update(self):
    """Test updating schedule with valid data"""
    response = self.client.post(reverse('appointment:edit_schedule', args=[self.schedule.id]), {
        'day_of_week': 'Tuesday',
        'start_time': '10:00',
        'end_time': '15:00'
    })

    self.assertRedirects(response, reverse('appointment:vet_schedule', kwargs={'vet_id': self.vet_profile.id}))

    # Refresh schedule from DB
    updated_schedule = vetschedule.objects.get(pk=self.schedule.id)
    self.assertEqual(updated_schedule.day_of_week, 'Tuesday')
    self.assertEqual(updated_schedule.start_time, time(10, 0))
    self.assertEqual(updated_schedule.end_time, time(15, 0))

```

Figure 477 Unit Test Appt2 Automation Script 2

```

def test_invalid_time_range(self):
    """Test updating with end time before start time"""
    response = self.client.post(reverse('appointment:edit_schedule', args=[self.schedule.id]), {
        'day_of_week': 'Wednesday',
        'start_time': '14:00',
        'end_time': '12:00'
    })

    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'End time must be after start time.')

    # Verify schedule wasn't updated
    unchanged_schedule = vetschedule.objects.get(pk=self.schedule.id)
    self.assertEqual(unchanged_schedule.day_of_week, 'Monday')

```

Figure 478 Unit Test Appt2 Automation Script 3

```

def test_overlapping_schedule(self):
    """Test updating with overlapping time"""
    # Create another schedule that would cause overlap
    vetSchedule.objects.create(
        vet=self.vet_profile,
        day_of_week='Thursday',
        start_time=time(10, 0),
        end_time=time(12, 0)
    )

    response = self.client.post(reverse('appointment:edit_schedule', args=[self.schedule.id]), {
        'day_of_week': 'Thursday', # Same day as existing schedule
        'start_time': '11:00',      # Overlaps with 10-12
        'end_time': '13:00'
    })

    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'The schedule overlaps with an existing one.')

    # Verify schedule wasn't updated
    unchanged_schedule = vetSchedule.objects.get(pk=self.schedule.id)
    self.assertEqual(unchanged_schedule.day_of_week, 'Monday')

```

Figure 479 Unit Test Appt2 Automation Script 3

```

def test_schedule_deletion(self):
    """Test deleting a schedule"""
    response = self.client.post(reverse('appointment:edit_schedule', args=[self.schedule.id]), {
        'delete_schedule': 'true'
    })

    self.assertRedirects(response, reverse('appointment:vet_schedule', kwargs={'vet_id': self.vet_profile.id}))
    self.assertFalse(vetSchedule.objects.filter(pk=self.schedule.id).exists())

```

Figure 480 Unit Test Appt2 Automation Script 4

8.3.1.10 Script of Unit Test Appt3

```
class BookAppointmentTests(TestCase):
    def setUp(self):
        self.client = Client()
        # Create vet user and profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile
        # Create pet owner user and profile
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
        # Create pet
        self.pet = Pet.objects.create(
            owner=self.pet_owner,
            name='Fluffy',
            breed='Persian',
            species='Cat',
            age=4
        )
        # Create schedule
        self.schedule = VetSchedule.objects.create(
            vet=self.vet_profile,
            day_of_week='Monday',
            start_time=time(9, 0),
            end_time=time(12, 0)
        )
    self.client.login(email='owner@example.com', password='testpass123') # Login the pet owner
```

Figure 481 Unit Test Appt3 Setup

```
def test_book_appointment_get(self):
    """Test GET request to book appointment page"""
    response = self.client.get(reverse('appointment:book_appointment', args=[self.vet_profile.id, self.schedule.id]))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/book_appt.html')
    self.assertEqual(response.context['vet'], self.vet_profile)
    self.assertEqual(response.context['schedule'], self.schedule)
    self.assertIn(self.pet, response.context['pets'])
```

Figure 482 Unit Test Appt3 Automation Script 1

```

def test_successful_booking(self):
    """Test successful appointment booking"""
    response = self.client.post(
        reverse('appointment:book_appointment', args=[self.vet_profile.id, self.schedule.id]),
        {
            'pet': str(self.pet.id),
            'reason': 'Annual checkup'
        }
    )

    # Verify we got a redirect response (302)
    self.assertEqual(response.status_code, 302)

    # Verify the redirect URL pattern
    appointment = Appointment.objects.first()
    expected_url = reverse('appointment:initialize_khalti_payment', kwargs={'appointment_id': appointment.id})
    self.assertRedirects(response, expected_url, fetch_redirect_response=False)

    # Verify appointment was created with correct data
    self.assertEqual(appointment.pet_owner, self.pet_owner)
    self.assertEqual(appointment.vet, self.vet_profile)
    self.assertEqual(appointment.pet, self.pet)
    self.assertEqual(appointment.status, 'unpaid')

```

Figure 483 Unit Test Appt3 Automation Script 2

```

def test_missing_pet_selection(self):
    """Test booking without selecting a pet"""
    response = self.client.post(reverse('appointment:book_appointment', args=[self.vet_profile.id, self.schedule.id]), {
        'reason': 'Checkup'
        # Missing pet field
    })
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, 'Please select a pet')
    self.assertEqual(Appointment.objects.count(), 0)

```

Figure 484 Unit Test Appt3 Automation Script 3

8.3.1.11 Script of Unit Test Appt4

```

class BookWithCreditTests(TestCase):
    def setUp(self):
        self.client = Client()
        # Create vet user and profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile
        # Create pet owner user with sufficient credit
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
        self.pet_owner.credit_balance = 1000 * 100 # 1k NPR in paisa
        self.pet_owner.save()
        self.pet = Pet.objects.create( # Create pet
            owner=self.pet_owner,
            name='Fluffy',
            breed='Persian',
            species='Cat',
            age=9
        )
        self.schedule = Vetschedule.objects.create() # Create schedule
        self.schedule.vet=self.vet_profile,
        self.schedule.day_of_week='Monday',
        self.schedule.start_time=time(9, 0),
        self.schedule.end_time=time(12, 0)
    )
    self.client.login(email='owner@example.com', password='testpass123') # Login the pet owner

```

Figure 485 Unit Test Appt4 Setup

```
def test_successful_booking_with_credit(self):
    """Test successful booking with sufficient credit"""
    initial_credit = self.pet_owner.credit_balance

    response = self.client.post(
        reverse('appointment:book_with_credit', args=[self.vet_profile.id, self.schedule.id]),
        {
            'pet': str(self.pet.id),
            'reason': 'Annual checkup'
        },
        follow=True
    )

    # Check redirect to appointment detail
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/appointment_detail.html')

    # Verify appointment was created with correct status
    appointment = Appointment.objects.first()
    self.assertEqual(appointment.status, 'paid_pending_approval')
    self.assertEqual(appointment.payment_status, 'paid')
    self.assertEqual(appointment.amount_paid, 1000 * 100)

    # Verify credit was deducted
    self.pet_owner.refresh_from_db()
    self.assertEqual(self.pet_owner.credit_balance, initial_credit - 1000 * 100)
```

Figure 486 Unit Test Appt4 Automation Script 1

```

def test_insufficient_credit(self):
    """Test booking with insufficient credit"""
    # Set low credit balance
    self.pet_owner.credit_balance = 500 * 100 # 500 NPR
    self.pet_owner.save()

    response = self.client.post(
        reverse('appointment:book_with_credit', args=[self.vet_profile.id, self.schedule.id]),
        {
            'pet': str(self.pet.id),
            'reason': 'Checkup'
        },
        follow=True
    )

    # Should redirect back to booking page with error
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/book_appt.html')

    # Check error message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(str(messages[0]), 'Insufficient credit balance')

    # No appointment should be created
    self.assertEqual(Appointment.objects.count(), 0)

```

Figure 487 Unit Test Appt4 Automation Script 2

```

def test_credit_rollback_on_error(self):
    """Test credit is rolled back if error occurs during booking"""
    initial_credit = self.pet_owner.credit_balance
    print(f"Initial credit balance: {initial_credit}") # Debugging line

    # Force an error by providing invalid data
    response = self.client.post(
        reverse('appointment:book_with_credit', args=[self.vet_profile.id, self.schedule.id]),
        {
            'pet': 'invalid', # Will cause error (invalid pet ID)
            'reason': 'Checkup'
        }
    )

    # Verify credit was not deducted
    self.pet_owner.refresh_from_db()
    print(f"Credit balance after error: {self.pet_owner.credit_balance}") # Debugging line

    self.assertEqual(self.pet_owner.credit_balance, initial_credit)

```

Figure 488 Unit Test Appt4 Automation Script 3

8.3.1.12 Script of Unit Test Appt5

```

class AcceptAppointmentTestCase(TestCase):
    def setUp(self):
        self.factory = RequestFactory()

        # Create vet user with profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create another vet user for negative testing
        self.other_vet_user = User.objects.create_user(
            email='othervet@example.com',
            username='othervet',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.other_vet_profile = self.other_vet_user.vetprofile

        # Create pet owner user with sufficient credit
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
        self.pet_owner.credit_balance = 1000 * 100 # 1k NPR in paisa
        self.pet_owner.save()
    
```

Figure 489 Unit Test Appt5 Setup

```
# Create pet
self.pet = Pet.objects.create(
    owner=self.pet_owner,
    name='Fluffy',
    breed='Persian',
    species='Cat',
    age=9
)

# Create schedules
self.schedule = VetSchedule.objects.create(
    vet=self.vet_profile,
    day_of_week='Monday',
    start_time=time(9, 0),
    end_time=time(12, 0),
    available=True
)

self.other_schedule = VetSchedule.objects.create(
    vet=self.other_vet_profile,
    day_of_week='Tuesday',
    start_time=time(10, 0),
    end_time=time(11, 0),
    available=True
)

# Create appointments
self.appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='paid_pending_approval',
    payment_status='paid',
    pet=self.pet,
    amount_paid=100 * 100 # 1000 NPR in paisa
)
```

Figure 490 Unit Test Appt5 Setup 2

```
self.other_appointment = Appointment.objects.create(  
    pet_owner=self.pet_owner,  
    vet=self.vet_profile,  
    schedule=self.schedule,  
    status='paid_pending_approval',  
    payment_status='paid',  
    pet=self.pet,  
    amount_paid=100 * 100  
)  
  
self.different_vet_appointment = Appointment.objects.create(  
    pet_owner=self.pet_owner,  
    vet=self.other_vet_profile,  
    schedule=self.other_schedule,  
    status='paid_pending_approval',  
    payment_status='paid',  
    pet=self.pet,  
    amount_paid=100 * 100  
)
```

Figure 491 Unit Test Appt5 Setup 3

```
@patch('appointment.views.send_appointment_notification')
def test_accept_appointment_success(self, mock_send_notification):
    # Create request
    request = self.factory.get('/')
    request.user = self.vet_user

    # Call the view
    response = accept_appointment(request, self.appointment.id)

    # Refresh objects from db
    self.appointment.refresh_from_db()
    self.other_appointment.refresh_from_db()
    self.schedule.refresh_from_db()

    # Check appointment status changed
    self.assertEqual(self.appointment.status, 'confirmed')

    # Check other appointment was rejected
    self.assertEqual(self.other_appointment.status, 'rejected')

    # Check schedule is now unavailable
    self.assertFalse(self.schedule.available)

    # Check notifications were sent
    self.assertEqual(mock_send_notification.call_count, 2)

    # Check redirect
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:vet_pending_appointments', args=[self.vet_profile.id])
    )
```

Figure 492 Unit Test Appt5 Automation Script 1

```

def test_accept_appointment_wrong_vet(self):
    # Create request with other vet user
    request = self.factory.get('/')
    request.user = self.other_vet_user

    # Call the view
    response = accept_appointment(request, self.appointment.id)

    # Check redirect is to the correct appointment's vet page, not current user
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:vet_pending_appointments', args=[self.vet_profile.id])
    )

    # Confirm status did not change
    self.appointment.refresh_from_db()
    self.assertEqual(self.appointment.status, 'paid_pending_approval')

```

Figure 493 Unit Test Appt5 Automation Script 2

```

@patch('appointment.views.send_appointment_notification')
def test_accept_appointment_rejects_others(self, mock_send_notification):
    # Create another appointment for the same schedule
    another_appointment = Appointment.objects.create(
        pet_owner=self.pet_owner,
        vet=self.vet_profile,
        schedule=self.schedule,
        status='paid_pending_approval',
        payment_status='paid',
        pet=self.pet,
        amount_paid=500 * 100
    )

    # Create request
    request = self.factory.get('/')
    request.user = self.vet_user

    # Call the view
    response = accept_appointment(request, self.appointment.id)

    # Refresh objects
    another_appointment.refresh_from_db()

    # Check other appointment was rejected
    self.assertEqual(another_appointment.status, 'rejected')

    # Check notification was sent for rejection
    mock_send_notification.assert_any_call(
        another_appointment,
        "Appointment Rejected",
        "appointment/rejection_email.html",
        is_rejection=True
    )

```

Figure 494 Unit Test Appt5 Automation Script 3

8.3.1.13 Script of Unit Test Appt6

```

class RejectAppointmentTestCase(TestCase):
    def setup(self):
        self.client = client() # Use Client instead of RequestFactory for messages
        self.factory = RequestFactory()

        # Create vet user with profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create another vet user for negative testing
        self.other_vet_user = User.objects.create_user(
            email='othervet@example.com',
            username='othervet',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.other_vet_profile = self.other_vet_user.vetprofile

        # Create pet owner user with sufficient credit
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
        self.pet_owner.credit_balance = 1000 * 100 # 1k NPR in paisa
        self.pet_owner.save()
    
```

Figure 495 Unit Test Appt6 Setup

```
# Create pet
self.pet = Pet.objects.create(
    owner=self.pet_owner,
    name='Fluffy',
    breed='Persian',
    species='Cat',
    age=9
)

# Create schedules
self.schedule = VetSchedule.objects.create(
    vet=self.vet_profile,
    day_of_week='Monday',
    start_time=time(9, 0),
    end_time=time(12, 0),
    available=True
)

# Create paid appointment
self.paid_appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='paid_pending_approval',
    payment_status='paid',
    pet=self.pet,
    amount_paid=500 * 100 # 500 NPR in paisa
)
```

Figure 496 Unit Test Appt6 Setup 2

```

@patch('appointment.views.send_appointment_notification')
def test_successful_rejection_with_refund(self, mock_send_notification):
    """Test rejecting a paid appointment credits the owner's account"""
    initial_balance = self.pet_owner.credit_balance

    # Use Client instead of RequestFactory to support messages
    self.client.force_login(self.vet_user)
    response = self.client.get(
        reverse('appointment:reject_appointment', args=[self.paid_appointment.id])
    )

    # Refresh from DB
    self.paid_appointment.refresh_from_db()
    self.pet_owner.refresh_from_db()

    # Check status changed
    self.assertEqual(self.paid_appointment.status, 'rejected')

    # Check credit was added
    self.assertEqual(
        self.pet_owner.credit_balance,
        initial_balance + self.paid_appointment.amount_paid
    )
    # Check notification sent
    mock_send_notification.assert_called_once_with(
        self.paid_appointment,
        "Appointment Rejected",
        "appointment/rejection_email.html",
        is_rejection=True
    )
    # Check redirect
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:veterinarian_pending_appointments', args=[self.vet_profile.id])
    )

```

Figure 497 Unit Test Appt6 Automation Script 1

```

def test_rejection_by_wrong_vet(self):
    """Test that other vets can't reject the appointment"""
    self.client.force_login(self.other_vet_user)
    response = self.client.get(
        reverse('appointment:reject_appointment', args=[self.paid_appointment.id])
    )

    # Check appointment wasn't changed
    self.paid_appointment.refresh_from_db()
    self.assertEqual(self.paid_appointment.status, 'paid_pending_approval')

    # Check redirect to the correct appointment's vet pending page
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:veterinarian_pending_appointments', args=[self.vet_profile.id]) # not other_vet_profile!
    )

```

Figure 498 Unit Test Appt6 Automation Script 2

```
@patch('appointment.views.send_appointment_notification')
def test_rejection_error_handling(self, mock_send_notification):
    """Test error during rejection is properly handled"""
    # Simulate an error in credit processing
    original_credit = self.pet_owner.credit_balance
    with patch.object(PetOwnerProfile, 'save', side_effect=Exception("Test error")):
        self.client.force_login(self.vet_user)
        response = self.client.get([
            reverse('appointment:reject_appointment', args=[self.paid_appointment.id])
        ])

        # Check credit wasn't changed
        self.pet_owner.refresh_from_db()
        self.assertEqual(self.pet_owner.credit_balance, original_credit)

        # Check redirect
        self.assertEqual(response.status_code, 302)
        self.assertEqual(
            response.url,
            reverse('appointment:veterinarian_pending_appointments', args=[self.vet_profile.id])
    )
```

Figure 499 Unit Test Appt6 Automation Script 3

8.3.1.14 Script of Unit Test Appt7

```

class MarkCompletedTestCase(TestCase):
    def setUp(self):
        self.client = Client()
        self.factory = RequestFactory()

        # Create vet user with profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create another vet user for negative testing
        self.other_vet_user = User.objects.create_user(
            email='othervet@example.com',
            username='othervet',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.other_vet_profile = self.other_vet_user.vetprofile

        # Create pet owner user
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
    
```

Figure 500 Unit Test Appt7 Setup

```
# Create pet
self.pet = Pet.objects.create(
    owner=self.pet_owner,
    name='Fluffy',
    breed='Persian',
    species='Cat',
    age=9
)

# Create schedule (initially unavailable)
self.schedule = Vetschedule.objects.create(
    vet=self.vet_profile,
    day_of_week='Monday',
    start_time=time(9, 0),
    end_time=time(12, 0),
    available=False # Starts as unavailable
)

# Create confirmed appointment
self.appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='confirmed',
    payment_status='paid',
    pet=self.pet
)
```

Figure 501 Unit Test Appt7 Setup

```

def test_successful_completion(self):
    """Test marking an appointment as completed"""
    # Verify initial state
    self.assertEqual(self.appointment.status, 'confirmed')
    self.assertFalse(self.schedule.available)

    self.client.force_login(self.vet_user)
    response = self.client.post(
        reverse('appointment:mark_completed', args=[self.appointment.id])
    )

    # Refresh from DB
    self.appointment.refresh_from_db()
    self.schedule.refresh_from_db()

    # Check status changed
    self.assertEqual(self.appointment.status, 'completed')

    # Check schedule is now available
    self.assertTrue(self.schedule.available)

    # Check redirect
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:veterinarian_accepted_appointments', args=[self.vet_profile.id])
    )

```

Figure 502 Unit Test Appt7 Automation Script 1

```

def test_get_request_does_nothing(self):
    """Test that GET requests don't change status"""
    self.client.force_login(self.vet_user)
    response = self.client.get(
        reverse('appointment:mark_completed', args=[self.appointment.id])
    )

    # Status should remain unchanged
    self.appointment.refresh_from_db()
    self.assertEqual(self.appointment.status, 'confirmed')

    # Should still redirect
    self.assertEqual(response.status_code, 302)

def test_wrong_vet_CANNOT_complete(self):
    """Test that other vets can't mark appointments as completed"""
    self.client.force_login(self.other_vet_user)
    response = self.client.post(
        reverse('appointment:mark_completed', args=[self.appointment.id])
    )

    # Appointment should remain unchanged
    self.appointment.refresh_from_db()
    self.assertEqual(self.appointment.status, 'confirmed')

    # Should redirect to the correct appointment's vet accepted page
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:veterinarian_accepted_appointments', args=[self.vet_profile.id]) # not other_vet_profile!
    )

```

Figure 503 Unit Test Appt7 Automation Script 2

```
def test_unauthorized_user_redirected(self):
    """Test that non-logged-in users are redirected to login"""
    response = self.client.post(
        reverse('appointment:mark_completed', args=[self.appointment.id])
    )
    self.assertEqual(response.status_code, 302)
    self.assertTrue(response.url.startswith('/accounts/login/'))
```

Figure 504 Unit Test Appt7 Automation Script 3

8.3.1.15 Script of Unit Test Appt8

```

class CancelAppointmentTestCase(TestCase):
    def setUp(self):
        self.client = Client()
        self.factory = RequestFactory()

        # Create vet user
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create pet owner user with initial credit
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
        self.pet_owner.credit_balance = 1000 * 100 # 1000 NPR in paisa
        self.pet_owner.save()

        # Create pet
        self.pet = Pet.objects.create(
            owner=self.pet_owner,
            name='Fluffy',
            breed='Persian',
            species='Cat',
            age=3
        )
    
```

Figure 505 Unit Test Appt8 Setup

```
# Create schedule
self.schedule = VetSchedule.objects.create(
    vet=self.vet_profile,
    day_of_week='Monday',
    start_time=time(9, 0),
    end_time=time(10, 0),
    available=False
)

# Create different appointment types
self.paid_confirmed_appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='confirmed',
    payment_status='paid',
    pet=self.pet,
    amount_paid=1000 * 100 # 1000 NPR in paisa
)

self.paid_pending_appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='paid_pending_approval',
    payment_status='paid',
    pet=self.pet,
    amount_paid=800 * 100 # 800 NPR in paisa
)
```

Figure 506 Unit Test Appt8 Setup

```

def test_cancel_paid_confirmed_appointment(self):
    """Test cancelling a paid, confirmed appointment with 80% refund"""
    initial_balance = self.pet_owner.credit_balance

    self.client.force_login(self.owner_user)
    response = self.client.post(
        reverse('appointment:cancel_appointment', args=[self.paid_confirmed_appointment.id])
    )

    # Refresh from DB
    self.paid_confirmed_appointment.refresh_from_db()
    self.pet_owner.refresh_from_db()

    # Check status changed
    self.assertEqual(self.paid_confirmed_appointment.status, 'cancelled')

    # Check credit was added (80% of 1000 = 800 NPR)
    expected_refund = int(1000 * 100 * 0.8)
    self.assertEqual(
        self.pet_owner.credit_balance,
        initial_balance + expected_refund
    )

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn("800.00 NPR (80%) credited", str(messages[0]))

    # Check redirect
    self.assertEqual(response.status_code, 302)
    self.assertEqual(response.url, reverse('appointment:appointment_list'))

```

Figure 507 Unit Test Appt8 Automation Script 1

```

def test_cancel_paid_pending_appointment(self):
    """Test cancelling a paid but pending approval appointment"""
    initial_balance = self.pet_owner.credit_balance

    self.client.force_login(self.owner_user)
    response = self.client.post(
        reverse('appointment:cancel_appointment', args=[self.paid_pending_appointment.id])
    )

    # Refresh from DB
    self.paid_pending_appointment.refresh_from_db()
    self.pet_owner.refresh_from_db()

    # Check status changed
    self.assertEqual(self.paid_pending_appointment.status, 'cancelled')

    # Check credit was added (80% of 800 = 640 NPR)
    expected_refund = int(800 * 100 * 0.8)
    self.assertEqual(
        self.pet_owner.credit_balance,
        initial_balance + expected_refund
    )

    # Check success message
    messages = list(get_messages(response.wsgi_request))
    self.assertEqual(len(messages), 1)
    self.assertIn("640.00 NPR (80%) credited", str(messages[0]))

```

Figure 508 Unit Test Appt8 Automation Script 2

```

def test_get_request_does_nothing(self):
    """Test that GET requests don't cancel appointments"""
    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:cancel_appointment', args=[self.paid_confirmed_appointment.id])
    )

    # Status should remain unchanged
    self.paid_confirmed_appointment.refresh_from_db()
    self.assertEqual(self.paid_confirmed_appointment.status, 'confirmed')

    # Should still redirect
    self.assertEqual(response.status_code, 302)

```

Figure 509 Unit Test Appt8 Automation Script 3

8.3.1.16 Script of Unit Test Appt9

```
class KhaltiPaymentTestCase(TestCase):
    def setUp(self):
        self.client = Client()

        # Create vet user
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create pet owner user
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile

        # Create pet
        self.pet = Pet.objects.create(
            owner=self.pet_owner,
            name='Fluffy',
            breed='Persian',
            species='Cat',
            age=9
        )
```

Figure 510 Unit Test Appt9 Setup

```

# Create schedule
self.schedule = Vetschedule.objects.create(
    vet=self.vet_profile,
    day_of_week='Monday',
    start_time=time(9, 0),
    end_time=time(10, 0)
)

# Create unpaid appointment
self.appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='unpaid',
    payment_status='unpaid',
    pet=self.pet
)

```

Figure 511 Unit Test Appt9 Setup

```

@patch('appointment.views.requests.post')
def test_initiate_khalti_payment_success(self, mock_post):
    """Test successful payment initiation"""
    mock_response = Mock()
    mock_response.json.return_value = {'payment_url': 'https://khalti.com/payment/test123'}
    mock_response.raise_for_status = Mock()
    mock_post.return_value = mock_response

    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:initiate_khalti_payment', args=[self.appointment.id])
    )

    self.assertEqual(response.status_code, 302)
    self.assertEqual(response.url, 'https://khalti.com/payment/test123')

```

Figure 512 Unit Test Appt9 Automation Script 1

```

@patch('appointment.views.requests.post')
def test_initiate_khalti_payment_failure(self, mock_post):
    """Test failed payment initiation"""
    # Mock the exception being raised
    mock_post.side_effect = requests.exceptions.RequestException("API Error")

    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:initiate_khalti_payment', args=[self.appointment.id])
    )

    # Should render error template
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/payment_error.html')

```

Figure 513 Unit Test Appt9 Automation Script 2

```

@patch('appointment.views.requests.post')
@patch('appointment.views.send_appointment_notification')
def test_verify_khalti_payment_success(self, mock_notify, mock_post):
    """Test successful payment verification"""
    mock_response = Mock()
    mock_response.json.return_value = {
        'status': 'Completed',
        'total_amount': 1000 * 100,
        'pidx': 'test_pidx_123'
    }
    mock_response.raise_for_status = Mock()
    mock_post.return_value = mock_response

    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:verify_khalti_payment', args=[self.appointment.id]),
        {'pidx': 'test_pidx_123'}
    )

    self.appointment.refresh_from_db()
    self.assertEqual(self.appointment.status, 'paid_pending_approval')
    self.assertEqual(response.status_code, 302)

```

Figure 514 Unit Test Appt9 Automation Script 3

```

@patch('appointment.views.requests.post')
def test_verify_khalti_payment_failed(self, mock_post):
    """Test failed payment verification"""
    mock_response = Mock()
    mock_response.json.return_value = {'status': 'Failed'}
    mock_response.raise_for_status = Mock()
    mock_post.return_value = mock_response

    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:verify_khalti_payment', args=[self.appointment.id]),
        {'pidx': 'test_pidx_123'}
    )

    # Should redirect to failure page
    self.assertEqual(response.status_code, 302)
    self.assertEqual(
        response.url,
        reverse('appointment:payment_failed', args=[self.appointment.id])
    )

    # Verify appointment was updated
    self.appointment.refresh_from_db()
    self.assertEqual(self.appointment.status, 'unpaid')
    self.assertEqual(self.appointment.payment_status, 'failed')

```

Figure 515 Unit Test Appt9 Automation Script 4

```

@patch('appointment.views.requests.post')
def test_verify_khalti_payment_error(self, mock_post):
    """Test error during verification"""
    mock_post.side_effect = requests.exceptions.RequestException("Verification Error")

    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:verify_khalti_payment', args=[self.appointment.id]),
        {'pidx': 'test_pidx_123'}
    )

    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/payment_error.html')

```

Figure 516 Unit Test Appt9 Automation Script 5

```
def test_verify_khalti_missing_pidx(self):
    """Test verification with missing pidx parameter"""
    self.client.force_login(self.owner_user)
    response = self.client.get(
        reverse('appointment:verify_khalti_payment', args=[self.appointment.id])
    )

    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/payment_error.html')
    self.assertIn('We encountered an issue with your payment', response.content.decode())
```

Figure 517 Unit Test Appt9 Automation Script 6

8.3.1.17 Script of Unit Test Appt10

```
class ReviewVetTestCase(TestCase):
    def setUp(self):
        # Create vet user and profile
        self.vet_user = User.objects.create_user(
            email='vet@example.com',
            username='vetuser',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.vet_user.vetprofile

        # Create owner user and profile
        self.owner_user = User.objects.create_user(
            email='owner@example.com',
            username='petowner',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner = self.owner_user.petownerprofile
        # Create pet
        self.pet = Pet.objects.create(
            owner=self.pet_owner,
            name='Fluffy',
            breed='Persian',
            species='Cat',
            age=9
        )
```

Figure 518 Unit Test Appt10 Setup

```

# Create schedule
self.schedule = Vetschedule.objects.create(
    vet=self.vet_profile,
    day_of_week='Monday',
    start_time='09:00:00',
    end_time='12:00:00',
    available=True
)

# Create completed and paid appointment
self.appointment = Appointment.objects.create(
    pet_owner=self.pet_owner,
    vet=self.vet_profile,
    schedule=self.schedule,
    status='completed',
    payment_status='paid',
    pet=self.pet,
    amount_paid=100 * 100 # 1000 NPR in paisa
)

```

Figure 519 Unit Test Appt10 Setup 2

```

def test_review_vet_unauthorized_user(self):
    """Test that only the pet owner can review the appointment"""
    # Create another user who didn't make the appointment
    other_user = User.objects.create_user(
        email='other@example.com',
        username='otheruser',
        password='testpass123',
        user_type='pet_owner',
        profile_completed=True,
        status_verification=True
    )
    self.client.force_login(other_user)

    url = reverse('authUser:review_vet', args=[self.appointment.id])
    response = self.client.get(url)
    self.assertEqual(response.status_code, 403)

```

Figure 520 Unit Test Appt10 Automation Script 1

```

def test_review_vet_not_completed_appointment(self):
    """Test that only completed appointments can be reviewed"""
    # Change appointment status to pending
    self.appointment.status = 'paid_pending_approval'
    self.appointment.save()

    self.client.force_login(self.owner_user)
    url = reverse('authUser:review_vet', args=[self.appointment.id])
    response = self.client.get(url)

    self.assertRedirects(response, reverse('appointment:appointment_detail', args=[self.appointment.id]))

    # Check for error message
    messages_list = list(messages.get_messages(response.wsgi_request))
    self.assertEqual(len(messages_list), 1)
    self.assertIn("You can only review completed and paid appointments", str(messages_list[0]))

```

Figure 521 Unit Test Appt10 Automation Script 2

```

def test_review_vet_already_reviewed(self):
    """Test that a user can't review the same appointment twice"""
    # Create existing review
    Review.objects.create(
        vet=self.vet_profile,
        reviewer=self.owner_user,
        rating=5,
        comment='Great!',
        appointment=self.appointment
    )

    self.client.force_login(self.owner_user)
    url = reverse('authUser:review_vet', args=[self.appointment.id])
    response = self.client.get(url)

    self.assertRedirects(response, reverse('appointment:appointment_detail', args=[self.appointment.id]))

    # Check for info message
    messages_list = list(messages.get_messages(response.wsgi_request))
    self.assertEqual(len(messages_list), 1)
    self.assertIn("You have already reviewed this appointment", str(messages_list[0]))

```

Figure 522 Unit Test Appt10 Automation Script 3

```

def test_review_vet_successful_submission(self):
    """Test successful review submission"""
    self.client.force_login(self.owner_user)
    url = reverse('authUser:review_vet', args=[self.appointment.id])

    data = {
        'rating': 5,
        'comment': 'Excellent service!'
    }

    response = self.client.post(url, data)

    # Check if review was created
    self.assertTrue(Review.objects.filter(appointment=self.appointment).exists())

    # Check redirect
    self.assertRedirects(response, reverse('appointment:appointment_detail', args=[self.appointment.id]))

    # Check success message
    messages_list = list(messages.get_messages(response.wsgi_request))
    self.assertEqual(len(messages_list), 1)
    self.assertIn("Thank you for your review", str(messages_list[0]))

```

Figure 523 Unit Test Appt10 Automation Script 4

```

def test_review_vet_template_used(self):
    """Test that the correct template is used for GET request"""
    self.client.force_login(self.owner_user)
    url = reverse('authUser:review_vet', args=[self.appointment.id])
    response = self.client.get(url)

    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'appointment/review_vet.html')
    self.assertEqual(response.context['appointment'], self.appointment)

```

Figure 524 Unit Test Appt10 Automation Script 5

8.3.1.18 Script of Unit Test Po1

```
class PostViewsTestCase(TestCase):
    def setUp(self):
        self.client = Client()

        self.user = User.objects.create_user(
            username='petowner',
            email='owner@example.com',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )

        self.pet_owner_profile = self.user.petownerprofile
        self.client.force_login(self.user)

        self.category = Category.objects.create(
            name='Test Category',
            slug='test-category'
        )
```

Figure 525 Unit Test Po1 Setup

```

def test_create_post_success(self):
    """Test post creation redirects to post-detail"""
    response = self.client.post(
        reverse('coreFunctions:create-post'),
        {
            'title': 'New Test Post',
            'body': 'Test content',
            'category': self.category.id
        }
    )
    self.assertEqual(response.status_code, 302)
    self.assertRedirects(response, reverse('coreFunctions:post-detail',
                                         kwargs={'slug': Post.objects.last().slug}))

```

Figure 526 Unit Test Po1 Automation Script 1

```

def test_create_post_with_image(self):
    """Test image upload redirects to post-detail"""
    image = SimpleUploadedFile(
        "test.jpg", b"file_content", content_type="image/jpeg"
    )
    response = self.client.post(
        reverse('coreFunctions:create-post'),
        {
            'title': 'Post with Image',
            'body': 'Test content',
            'category': self.category.id,
            'image': image
        },
        format='multipart/form-data'
    )
    self.assertEqual(response.status_code, 302)
    self.assertTrue(response.url.startswith('/post/'))

```

Figure 527 Unit Test Po1 Automation Script 2

```
def test_create_post_unauthenticated(self):
    """Test unauthenticated user cannot create post"""
    # Logout the current user
    self.client.logout()

    initial_count = Post.objects.count()

    response = self.client.post(
        reverse('coreFunctions:create-post'),
        {
            'title': 'Unauthenticated Post',
            'body': 'Should not work',
            'category': self.category.id
        }
    )

    # Should redirect to login page
    self.assertEqual(response.status_code, 302)
    self.assertTrue(response.url.startswith('/accounts/login/'))

    # No new post should be created
    self.assertEqual(Post.objects.count(), initial_count)
```

Figure 528 Unit Test Po1 Automation Script 3

```

def test_delete_post_success(self):
    """Test post deletion returns JSON success"""
    post = Post.objects.create(
        title='Test Delete Post',
        body='Content',
        user=self.user,
        category=self.category,
        slug='test-delete-post'
    )
    response = self.client.post(
        reverse('coreFunctions:delete_post', args=[post.id])
    )
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {'success': True})

```

Figure 529 Unit Test Po1 Automation Script 4

```

def test_delete_post_invalid_method(self):
    """Test GET request returns JSON error"""
    post = Post.objects.create(
        title='Invalid Method Post',
        body='Content',
        user=self.user,
        category=self.category,
        slug='invalid-method-post'
    )
    response = self.client.get(
        reverse('coreFunctions:delete_post', args=[post.id])
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json(), {
        'success': False,
        'error': 'Invalid request method'
    })

```

Figure 530 Unit Test Po1 Automation Script 5

```
def test_delete_post_unauthorized(self):
    """Test unauthorized deletion returns JSON error"""
    # Create another user
    other_user = User.objects.create_user(
        username='otheruser',
        email='other@example.com',
        password='testpass123',
        user_type='pet_owner',
        profile_completed=True,
        status_verification=True
    )

    # Create post owned by main user
    post = Post.objects.create(
        title='Unauthorized Post',
        body='Content',
        user=self.user,
        category=self.category,
        slug='unauthorized-post'
    )

    # Get initial count
    initial_count = Post.objects.count()

    # Switch to other user and attempt deletion
    self.client.force_login(other_user)
    self.client.post(
        reverse('coreFunctions:delete_post', args=[post.id])
    )

    # Verify post still exists and count unchanged
    self.assertEqual(Post.objects.count(), initial_count)
    self.assertTrue(Post.objects.filter(id=post.id).exists())
```

Figure 531 Unit Test Po1 Automation Script 6

8.3.1.19 Script of Unit Test Po2

```
class PostEditLikeTestCase(TestCase):
    def setUp(self):
        self.client = Client()
        self.user = User.objects.create_user(
            username='petowner',
            email='owner@example.com',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner_profile = self.user.petownerprofile
        self.client.force_login(self.user)

        self.category = Category.objects.create(
            name='Test Category',
            slug='test-category'
        )

    # Create a test post
    self.post = Post.objects.create(
        title='Test Post',
        body='Test Content',
        user=self.user,
        category=self.category,
        slug='test-post'
    )
```

Figure 532 Unit Test Po2 Setup

```
# Edit Post Tests
def test_edit_post_success(self):
    """Test successful post editing"""
    response = self.client.post(
        reverse('coreFunctions:edit_post', kwargs={'slug': self.post.slug}),
        {
            'title': 'Updated Title',
            'body': 'Updated Content',
            'category': self.category.id
        }
    )

    # Verify redirect to post detail
    self.assertEqual(response.status_code, 302)
    self.assertRedirects(response, reverse('coreFunctions:post-detail', kwargs={'slug': self.post.slug}))

    # Refresh post from db
    self.post.refresh_from_db()
    self.assertEqual(self.post.title, 'Updated Title')
    self.assertEqual(self.post.body, 'Updated Content')
```

Figure 533 Unit Test Po2 Automation Script 1

```
def test_edit_post_with_image(self):
    """Test post editing with image upload"""
    image = SimpleUploadedFile(
        "test.jpg", b"file_content", content_type="image/jpeg"
    )

    response = self.client.post(
        reverse('coreFunctions:edit_post', kwargs={'slug': self.post.slug}),
        {
            'title': 'Post with Image',
            'body': 'Content',
            'category': self.category.id,
            'image': image
        },
        format='multipart/form-data'
    )

    # Refresh post from db
    self.post.refresh_from_db()
    self.assertTrue(bool(self.post.image))
    self.assertEqual(response.status_code, 302)
```

Figure 534 Unit Test Po2 Automation Script 2

```
def test_edit_post_unauthorized(self):
    """Test unauthorized user cannot edit post"""
    other_user = User.objects.create_user(
        username='otheruser',
        email='other@example.com',
        password='testpass123',
        user_type='pet_owner',
        profile_completed=True,
        status_verification=True
    )

    # Create post owned by other user
    other_post = Post.objects.create(
        title='Other Post',
        body='Content',
        user=other_user,
        category=self.category,
        slug='other-post'
    )

    response = self.client.post(
        reverse('coreFunctions:edit_post', kwargs={'slug': other_post.slug}),
        {
            'title': 'Try to Edit',
            'body': 'should not work',
            'category': self.category.id
        }
    )

    # Should return 404 since post doesn't belong to user
    self.assertEqual(response.status_code, 404)
```

Figure 535 Unit Test Po2 Automation Script 3

```
# Like Post Tests
def test_like_post_success(self):
    """Test successful like/unlike toggle"""
    # First like
    response = self.client.post(
        reverse('coreFunctions:like_post', args=[self.post.id]),
        HTTP_X_REQUESTED_WITH='XMLHttpRequest' # Simulate AJAX
    )

    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {
        'status': 'success',
        'likes_count': 1,
        'liked': True
    })

    # Then unlike
    response = self.client.post(
        reverse('coreFunctions:like_post', args=[self.post.id]),
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )

    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {
        'status': 'success',
        'likes_count': 0,
        'liked': False
    })
```

Figure 536 Unit Test Po2 Automation Script 4

```
def test_like_post_invalid_method(self):
    """Test like post with invalid HTTP method"""
    response = self.client.get(
        reverse('coreFunctions:like_post', args=[self.post.id])
    )

    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json(), {
        'status': 'error',
        'message': 'Invalid request'
    })
```

Figure 537 Unit Test Po2 Automation Script 5

8.3.1.20 Script of Unit Test Po3

```
class CommentViewsTestCase(TestCase):
    def setUp(self):
        self.client = Client()
        self.user = User.objects.create_user(
            username='petowner',
            email='owner@example.com',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.client.force_login(self.user)

        self.category = Category.objects.create(name='Test Category', slug='test-category')
        self.post = Post.objects.create(
            title='Test Post',
            body='Test Content',
            user=self.user,
            category=self.category,
            slug='test-post'
        )
        self.comment = Comment.objects.create(
            post=self.post,
            user=self.user,
            comment='Test comment'
        )
```

Figure 538 Unit Test Po3 Setup

```

# Like Comment Tests
def test_like_comment_success(self):
    """Test successful like/unlike toggle"""
    # First like
    response = self.client.post(
        reverse('coreFunctions:like_comment', args=[self.comment.id]),
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {
        'success': True,
        'likes': 1,
        'liked': True
    })

    # Then unlike
    response = self.client.post(
        reverse('coreFunctions:like_comment', args=[self.comment.id]),
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {
        'success': True,
        'likes': 0,
        'liked': False
    })

```

Figure 539 Unit Test Po3 Automation Script 1

```

def test_like_comment_invalid_comment(self):
    """Test like non-existent comment shows 404 page"""
    response = self.client.post(
        reverse('coreFunctions:like_comment', args=[999]), # Invalid comment ID
        follow=True # Follow redirect to 404 page
    )
    self.assertEqual(response.status_code, 404)
    self.assertTemplateUsed(response, 'errors/404.html')

```

Figure 540 Unit Test Po3 Automation Script 2

```
# Delete Comment Tests
def test_delete_comment_success(self):
    """Test successful comment deletion"""
    response = self.client.post(
        reverse('coreFunctions:delete_comment', args=[self.comment.id])
    )
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {'success': True})
    self.assertFalse(Comment.objects.filter(id=self.comment.id).exists())
```

Figure 541 Unit Test Po3 Automation Script 3

```
def test_delete_comment_unauthorized(self):
    """Test unauthorized comment deletion shows 404 page"""
    other_user = User.objects.create_user(
        username='otheruser',
        email='other@example.com',
        password='testpass123',
        profile_completed=True,
        status_verification=True
    )
    other_comment = Comment.objects.create(
        post=self.post,
        user=other_user,
        comment='other comment'
    )
    response = self.client.post(
        reverse('coreFunctions:delete_comment', args=[other_comment.id]),
        follow=True # Follow redirect to 404 page
    )
    self.assertEqual(response.status_code, 404)
    self.assertTemplateUsed(response, 'errors/404.html')
```

Figure 542 Unit Test Po3 Automation Script 4

```
# Add Comment Tests
def test_add_comment_success(self):
    """Test successful comment creation"""
    response = self.client.post(
        reverse('coreFunctions:add_comment', kwargs={'slug': self.post.slug}),
        {'comment': 'New test comment'},
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )
    self.assertEqual(response.status_code, 200)
    response_data = response.json()
    self.assertTrue(response_data['success'])
    self.assertIn('comment_html', response_data)
    self.assertIn('comment_id', response_data)
    self.assertTrue(Comment.objects.filter(id=response_data['comment_id']).exists())
```

Figure 543 Unit Test Po3 Automation Script 5

```
def test_add_comment_empty(self):
    """Test comment with empty content"""
    response = self.client.post(
        reverse('coreFunctions:add_comment', kwargs={'slug': self.post.slug}),
        {'comment': ''},
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json(), {
        'success': False,
        'error': 'Comment cannot be empty'
    })
```

Figure 544 Unit Test Po3 Automation Script 6

```
def test_add_comment_invalid_post(self):
    """Test comment on non-existent post shows 404 page"""
    response = self.client.post(
        reverse('coreFunctions:add_comment', kwargs={'slug': 'non-existent-slug'}),
        {'comment': 'Test comment'},
        follow=True # Follow redirect to 404 page
    )
    self.assertEqual(response.status_code, 404)
    self.assertTemplateUsed(response, 'errors/404.html')
```

Figure 545 Unit Test Po3 Automation Script 7

8.3.1.21 Script of Unit Test Po4

```

class ReplyViewsTestCase(TestCase):
    def setUp(self):
        self.client = Client()
        self.user = User.objects.create_user(
            username='petowner',
            email='owner@example.com',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.client.force_login(self.user)

        self.category = Category.objects.create(name='Test Category', slug='test-category')
        self.post = Post.objects.create(
            title='Test Post',
            body='Test Content',
            user=self.user,
            category=self.category,
            slug='test-post'
        )
        self.comment = Comment.objects.create(
            post=self.post,
            user=self.user,
            comment='Test comment'
        )
    
```

Figure 546 Unit Test Po4 Setup

```

# Delete Reply Tests
def test_delete_reply_success(self):
    """Test successful reply deletion"""
    reply = ReplyComment.objects.create(
        comment=self.comment,
        user=self.user,
        reply='Test reply'
    )
    response = self.client.post(
        reverse('coreFunctions:delete_reply', args=[reply.id])
    )
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json(), {'success': True})
    self.assertFalse(ReplyComment.objects.filter(id=reply.id).exists())

```

Figure 547 Unit Test Po4 Automation Script 1

```

def test_delete_reply_unauthorized(self):
    """Test unauthorized reply deletion returns 404 page"""
    other_user = User.objects.create_user(
        username='otheruser',
        email='other@example.com',
        password='testpass123'
    )
    reply = ReplyComment.objects.create(
        comment=self.comment,
        user=other_user, # Owned by different user
        reply='Test reply'
    )
    response = self.client.post(
        reverse('coreFunctions:delete_reply', args=[reply.id]),
        follow=True # Follow redirect to 404 page
    )
    self.assertEqual(response.status_code, 404)
    self.assertTemplateUsed(response, 'errors/404.html') # Verify 404 page is shown

```

Figure 548 Unit Test Po4 Automation Script 2

```

def test_delete_reply_invalid_method(self):
    """Test reply deletion with invalid HTTP method"""
    reply = ReplyComment.objects.create(
        comment=self.comment,
        user=self.user,
        reply='Test reply'
    )
    response = self.client.get(
        reverse('coreFunctions:delete_reply', args=[reply.id])
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json(), {
        'success': False,
        'error': 'Invalid request method'
    })

```

Figure 549 Unit Test Po4 Automation Script 3

```
# Add Reply Tests
def test_add_reply_success(self):
    """Test successful reply creation"""
    response = self.client.post(
        reverse('coreFunctions:add_reply', args=[self.comment.id]),
        {'reply': 'Test reply content'},
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )
    self.assertEqual(response.status_code, 200)
    response_data = response.json()
    self.assertTrue(response_data['success'])
    self.assertIn('reply_html', response_data)
    self.assertIn('reply_id', response_data)
    self.assertTrue(ReplyComment.objects.filter(id=response_data['reply_id']).exists())
```

Figure 550 Unit Test Po4 Automation Script 4

```
def test_add_reply_empty(self):
    """Test reply with empty content"""
    response = self.client.post(
        reverse('coreFunctions:add_reply', args=[self.comment.id]),
        {'reply': ''},
        HTTP_X_REQUESTED_WITH='XMLHttpRequest'
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json(), {
        'success': False,
        'error': 'Reply cannot be empty'
    })
```

Figure 551 Unit Test Po4 Automation Script 5

```
def test_add_reply_invalid_comment(self):
    """Test reply to non-existent comment shows 404 page"""
    response = self.client.post(
        reverse('coreFunctions:add_reply', args=[999]), # Invalid comment ID
        {'reply': 'Test reply'},
        follow=True # Follow redirect to 404 page
    )
    self.assertEqual(response.status_code, 404)
    self.assertTemplateUsed(response, 'errors/404.html') # Verify 404 page is shown
```

Figure 552 Unit Test Po4 Automation Script 6

```
def test_add_reply_invalid_method(self):
    """Test add reply with invalid HTTP method"""
    response = self.client.get(
        reverse('coreFunctions:add_reply', args=[self.comment.id])
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json(), {
        'success': False,
        'error': 'Invalid request'
    })
```

Figure 553 Unit Test Po4 Automation Script 7

8.3.1.22 Script of Unit Test RTC1

```

class ChatConsumerTestCase(TestCase):
    def setUp(self):
        """Initialize test users and their profiles"""
        # Create pet owner user and profile
        self.user1 = User.objects.create_user(
            username='user1',
            email='user1@example.com',
            password='testpass123',
            user_type='pet_owner',
            profile_completed=True,
            status_verification=True
        )
        self.pet_owner_profile = self.user1.petownerprofile # Access profile via reverse relation

        # Create vet user and profile
        self.user2 = User.objects.create_user(
            username='user2',
            email='user2@example.com',
            password='testpass123',
            user_type='vet',
            profile_completed=True,
            status_verification=True
        )
        self.vet_profile = self.user2.vetprofile # Access profile via reverse relation

        # Create unique room name for chat
        self.room_name = f"chat_{self.user1.id}_{self.user2.id}"
    
```

Figure 554 Unit Test RTC1 Setup

```

async def test_chat_connection(self):
    """Test basic WebSocket connection setup"""
    communicator = WebsocketCommunicator(
        ChatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    # Manually add URL route params to scope
    communicator.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    connected, _ = await communicator.connect()
    self.assertTrue(connected)
    await communicator.disconnect()
    
```

Figure 555 Unit Test RTC1 Automation Script 1

```

async def test_chat_message_exchange(self):
    """Test complete message flow between two connected users"""
    # Setup first user connection
    communicator1 = WebSocketCommunicator(
        chatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    communicator1.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await communicator1.connect()

    # Setup second user connection
    communicator2 = WebSocketCommunicator(
        chatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    communicator2.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await communicator2.connect()
    # Send test message from user1
    test_message = "Hello from User1!"
    await communicator1.send_json_to({
        'message': test_message,
        'sender': 'user1',
        'receiver': 'user2',
        'timestamp': datetime.datetime.now().isoformat()
    })

```

Figure 556 Unit Test RTC1 Automation Script 2

```
# Verify user2 receives the message
response = await communicator2.receive_json_from()
self.assertEqual(response['message'], test_message)
self.assertEqual(response['sender'], 'user1')
self.assertEqual(response['receiver'], 'user2')

# Clean up connections
await communicator1.disconnect()
await communicator2.disconnect()
```

Figure 557 Unit Test RTC1 Automation Script 3

```
async def test_message_persistence(self):
    """Verify messages are properly saved to database"""
    # Get initial message count
    initial_count = await sync_to_async(ChatMessage.objects.count)()

    # Setup connection and send test message
    communicator = WebsocketCommunicator(
        ChatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    communicator.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await communicator.connect()

    test_msg_content = "This is a Test persistence message"
    await communicator.send_json_to({
        'message': test_msg_content,
        'sender': 'user1',
        'receiver': 'user2',
        'timestamp': datetime.datetime.now().isoformat()
    })
    await communicator.disconnect()

    # Verify message was saved
    final_count = await sync_to_async(ChatMessage.objects.count)()
    self.assertEqual(final_count, initial_count + 1)

    # Verify message content and participants
    message = await sync_to_async(ChatMessage.objects.last)()
    self.assertEqual(message.message.lower(), test_msg_content.lower()) # Case-insensitive check
    self.assertEqual(await sync_to_async(lambda: message.sender.username)(), 'user1')
    self.assertEqual(await sync_to_async(lambda: message.receiver.username)(), 'user2')
```

Figure 558 Unit Test RTC1 Automation Script 4

```
async def test_notification_flow(self):
    """Test notification system triggers properly"""
    # Connect user2 to notifications
    notif_communicator = WebSocketCommunicator(
        NotificationConsumer.as_asgi(),
        f"/ws/notifications/{self.user2.username}/"
    )
    notif_communicator.scope.update({
        'url_route': {
            'kwargs': {'username': self.user2.username}
        }
    })
    await notif_communicator.connect()

    # Connect user1 to chat and send message
    chat_communicator = WebSocketCommunicator(
        ChatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    chat_communicator.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await chat_communicator.connect()

    test_message = "This should trigger notification"
    await chat_communicator.send_json_to({
        'message': test_message,
        'sender': 'user1',
        'receiver': 'user2',
        'timestamp': datetime.datetime.now().isoformat()
    })
```

Figure 559 Unit Test RTC1 Automation Script 5

```

# Verify user2 receives notification
notif_response = await notif_communicator.receive_json_from()
self.assertEqual(notif_response['type'], 'new_message')
self.assertEqual(notif_response['message'], test_message)
self.assertEqual(notif_response['sender'], 'user1')

# Clean up
await notif_communicator.disconnect()
await chat_communicator.disconnect()

```

Figure 560 Unit Test RTC1 Automation Script 5

```

async def test_invalid_user_handling(self):
    """Test system handles invalid users gracefully"""
    communicator = WebsocketCommunicator(
        ChatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    communicator.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await communicator.connect()

    # Test invalid user doesn't crash system
    await communicator.send_json_to({
        'message': 'Test message',
        'sender': 'user1',
        'receiver': 'nonexistent_user',
        'timestamp': datetime.datetime.now().isoformat()
    })

    # Verify valid message still works
    valid_msg = 'Hello! Valid message'
    await communicator.send_json_to({
        'message': valid_msg,
        'sender': 'user1',
        'receiver': 'user2',
        'timestamp': datetime.datetime.now().isoformat()
    })

    response = await communicator.receive_json_from()
    self.assertEqual(response['message'], valid_msg)

    await communicator.disconnect()

```

Figure 561 Unit Test RTC1 Automation Script 6

```
async def test_concurrent.messaging(self):
    """Test handling of multiple rapid messages"""
    # Setup sender and receiver connections
    communicator1 = WebSocketCommunicator(
        ChatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    communicator1.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await communicator1.connect()

    communicator2 = WebSocketCommunicator(
        ChatConsumer.as_asgi(),
        f"/ws/chat/{self.room_name}/"
    )
    communicator2.scope.update({
        'url_route': {
            'kwargs': {'room_name': self.room_name}
        }
    })
    await communicator2.connect()

    # Send multiple messages quickly
    messages = ["Message 1", "Message 2", "Message 3"]
    for msg in messages:
        await communicator1.send_json_to({
            'message': msg,
            'sender': 'user1',
            'receiver': 'user2',
            'timestamp': datetime.datetime.now().isoformat()
        })
```

Figure 562 Unit Test RTC1 Automation Script 7

```
# Verify all messages arrive in order
for expected_msg in messages:
    response = await communicator2.receive_json_from()
    self.assertEqual(response['message'], expected_msg)

# Clean up
await communicator1.disconnect()
await communicator2.disconnect()
```

Figure 563 Unit Test RTC1 Automation Script 7

7.4 Appendix D: Designs

7.4.1 Gantt Chart

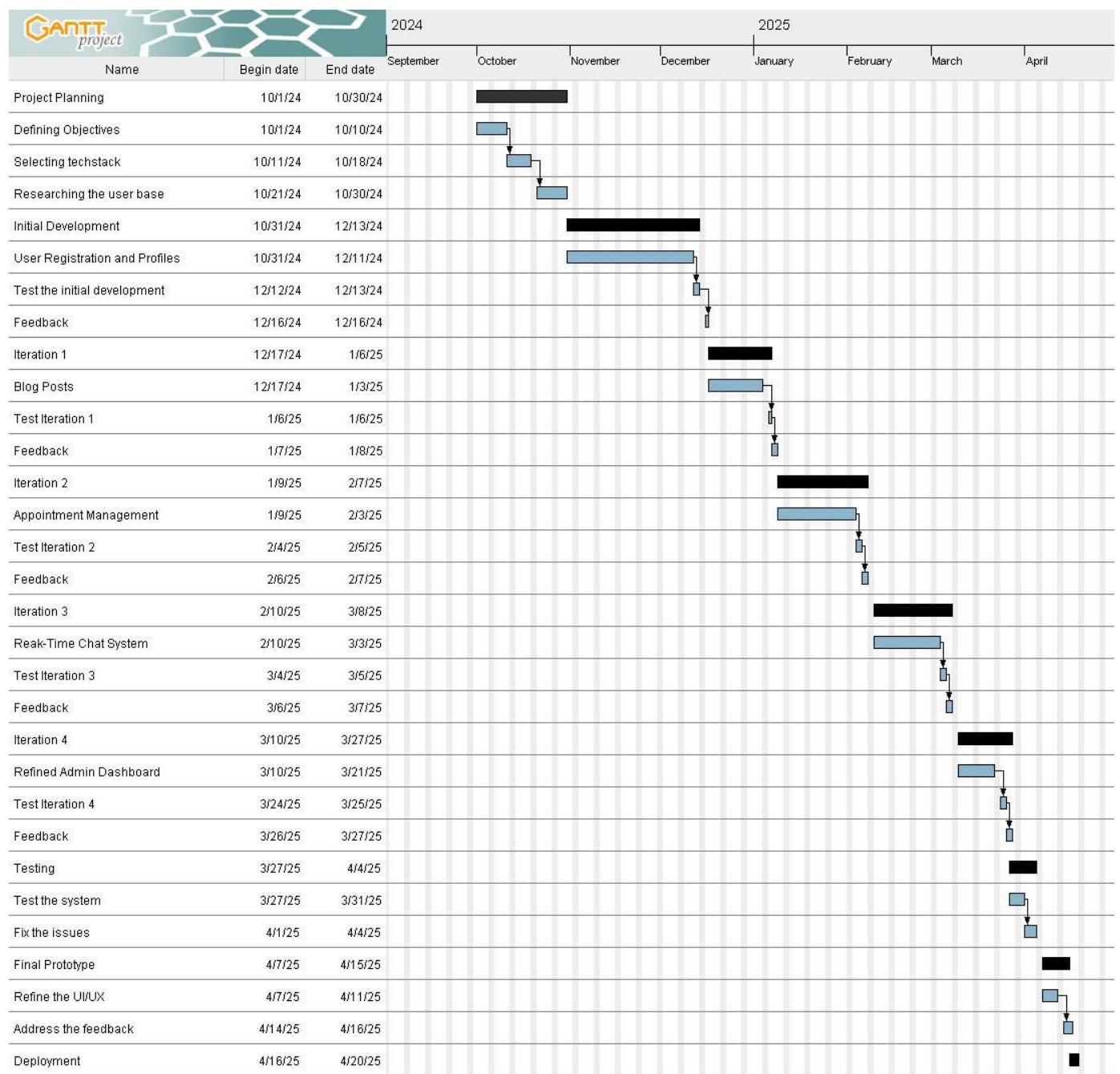


Figure 564 Old Gantt Chart

Several important omissions have scrapped the original Gantt chart which was created to track the project's progress. The missing or incorrect sections were the following:

- The design phase where the user interface is created using Figma was not included in Figma UI Design. This is a very important step in the project, because it determines the layout, user experience and overall design of the platform, and it directly affects the development process.
- The original Gantt chart did not consider the time needed to design UML (Unified Modeling Language) diagrams which are necessary for planning and visualizing the system's architecture and the relationship between different components. Prior to going into coding these diagrams are important in understanding and communicating the system's structure.
- Gantt chart: The Gantt chart was also stripped of the deployment phase, as deployment will not be carried out at this stage. It was decided to concentrate on the development and testing phases and to defer the deployment until a later point in the project.

Consequently, the Gantt chart was modified to depict the correct sequence of UI design, UML diagram creation and the elimination of the deployment phase. The new Gantt chart represents a more accurate view of the project's progress and coming milestones.

Link to the updated chart/diagram: [Figure 10: Final Gantt Chart](#)

7.4.2 Work Breakdown Structure

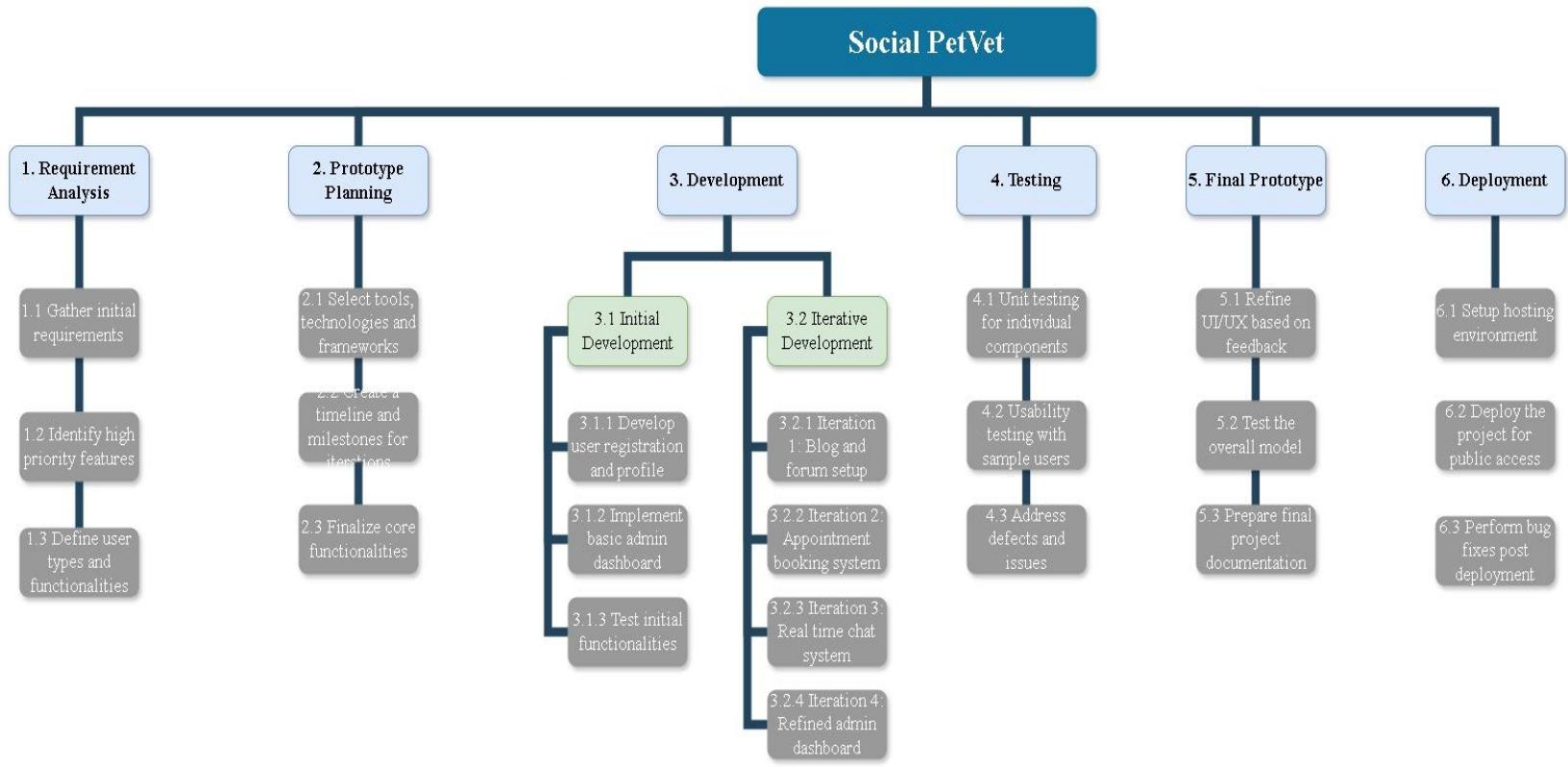


Figure 565 Old WBS

Key changes were reflected on the WBS.

- Initial Design Phase Added: These were tasks for UI design in Figma, Wireframes as well as creating UML diagrams (system architecture, use cases, etc.) that were necessary to begin the project.
- Removed: Deployment phase was removed because it will not be done at this stage. The current work is on design, development, and testing.

These changes make the WBS more suitable to the project's needs.

Link to the updated chart/diagram: [Figure 9: Final Work Breakdown Structure](#)

7.4.3 Entity Relation Diagram

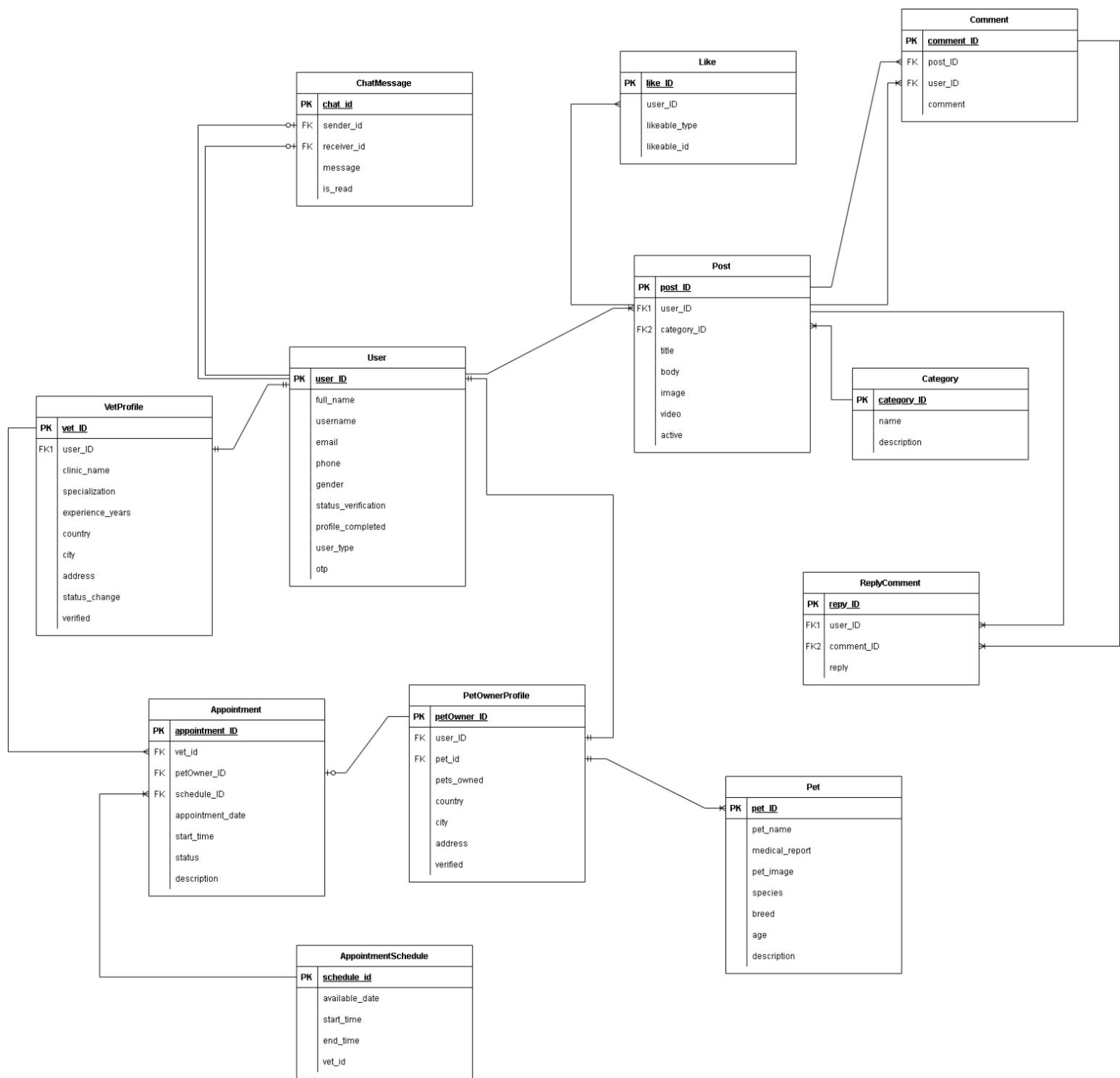


Figure 566 Old ERD

ERD Update:

- Relationship Added: Review was 1 to 1 with Appointment, meaning that each appointment by a particular pet owner can only have one review.
- Redundancy is Fixed: Foreign keys were removed that were redundant from the Appointment Schedule, Appointment, and Vet Profile tables to simplify the database schema.

Link to the updated chart/diagram: [Figure 12: Final Class Diagram](#)

7.4.4 Data Flow Diagram 0

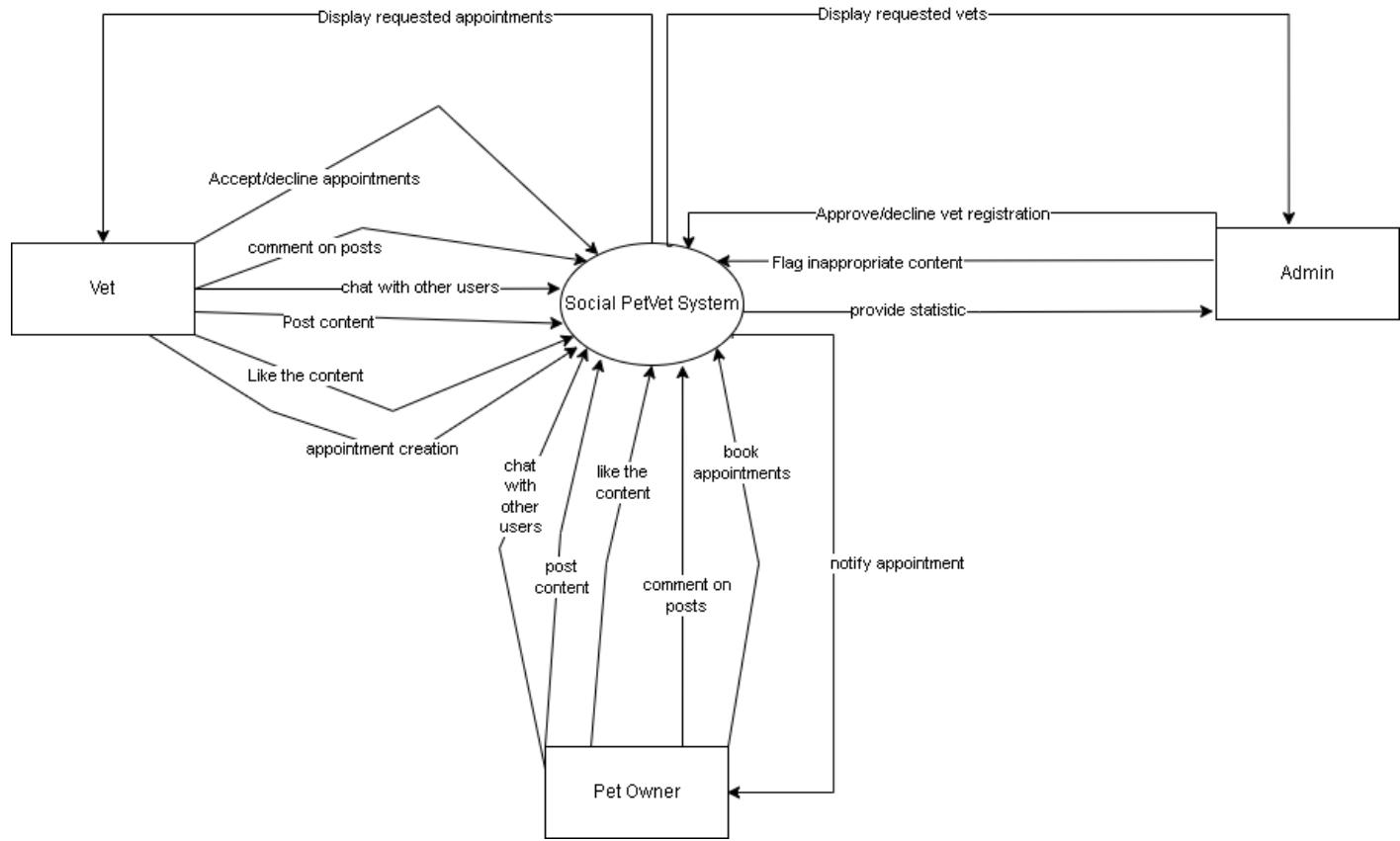


Figure 567 Old DFDO

DFD Update:

The DFD was updated to include the payment option, mark as complete feature, and review system. These additions ensure accurate representation of user actions post-appointment and streamline the overall data flow.

Link to the updated chart/diagram: [Figure 13: Final Data Flow Diagram](#)

7.4.5 Use Case

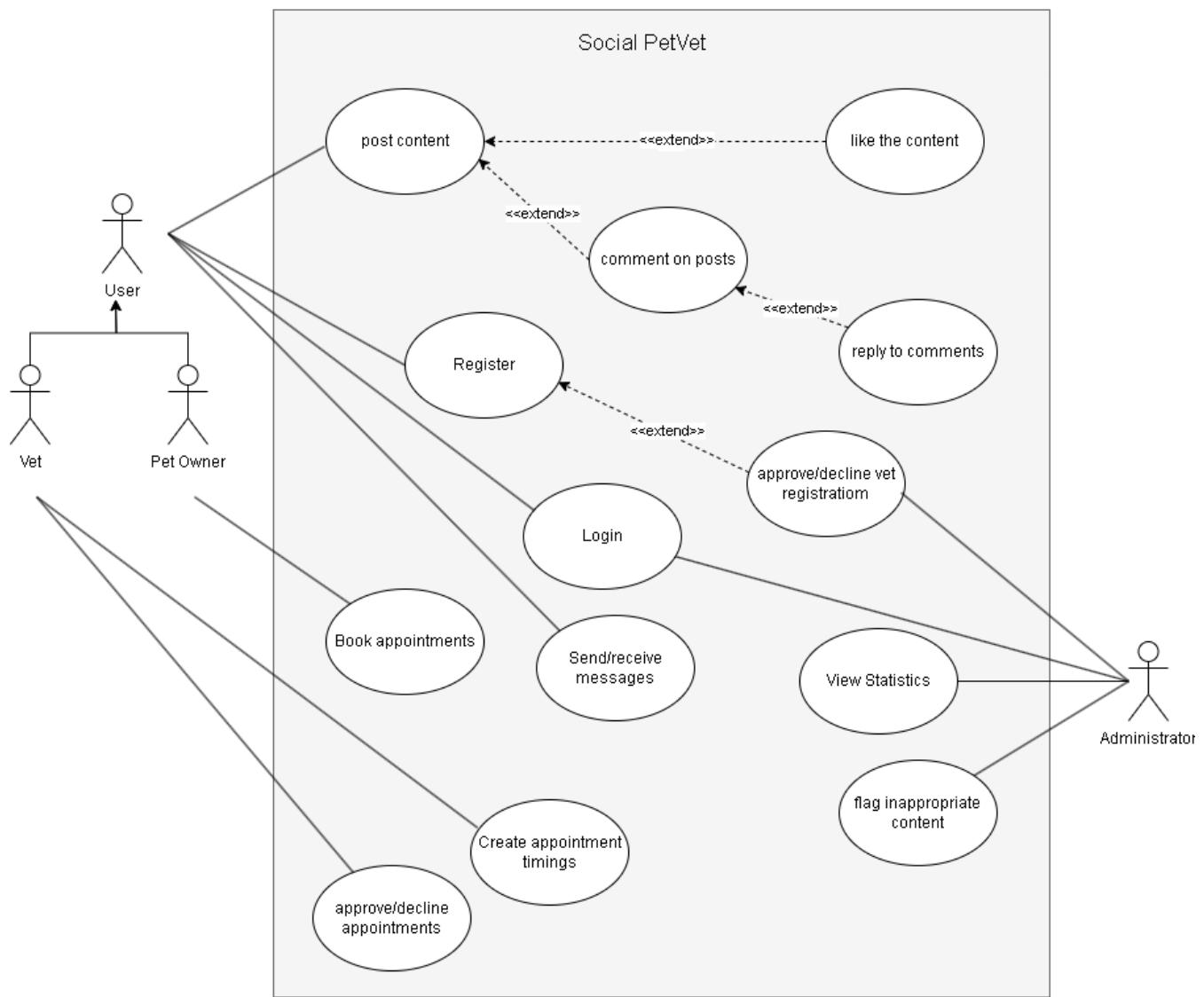


Figure 568 Old Usecase

Usecase update:

The use case diagram was updated to include payment as a required step after booking an appointment, with review marked as optional. Search functionality and liking comments were added, and OTP verification was incorporated for secure user authentication.

Link to the updated chart/diagram: [Figure 14: Final Usecase Diagram](#)

7.4.6 Activity Diagram

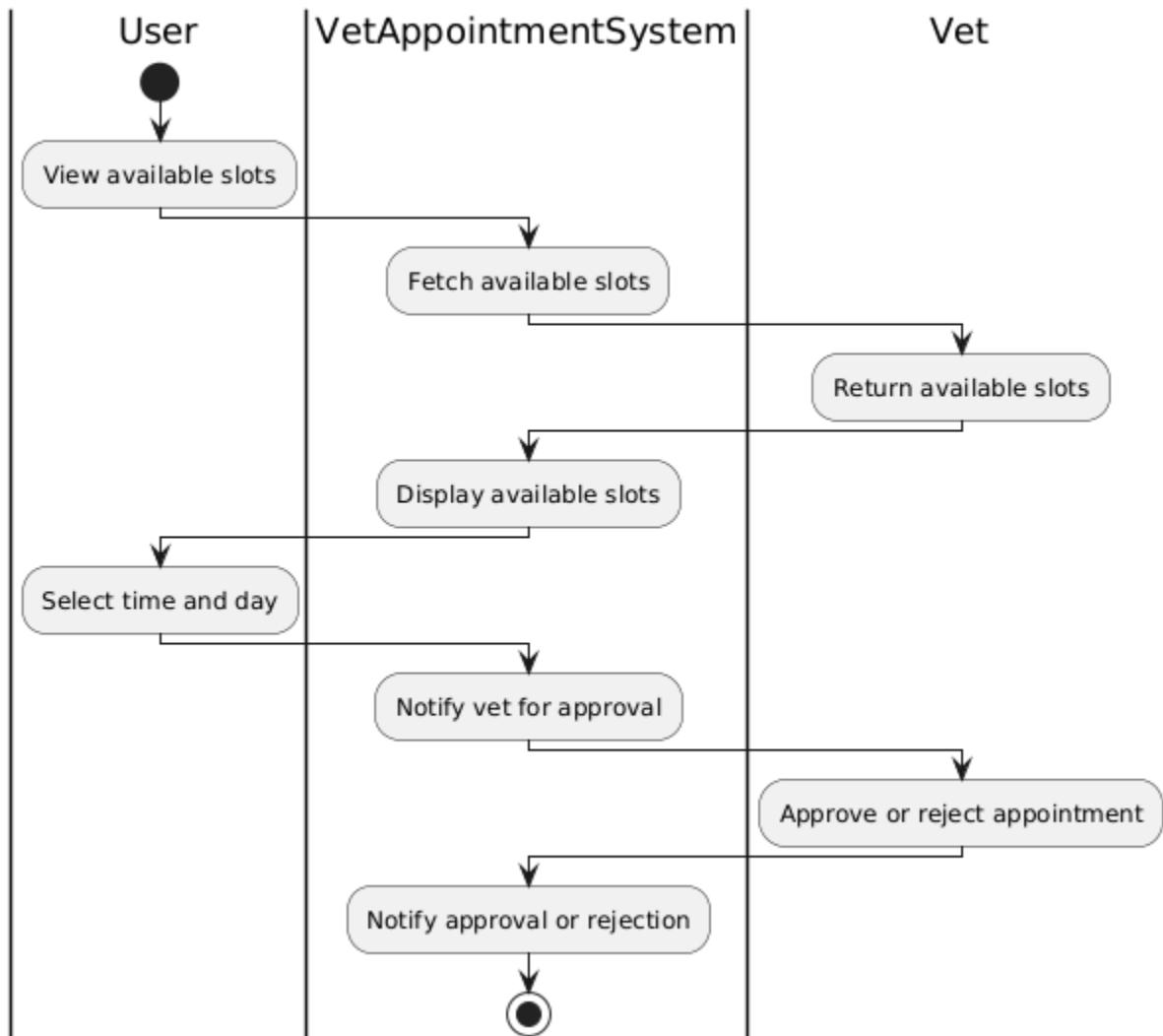


Figure 569 Old Activity Diagram for appointment

Activity Diagram Update:

The activity diagram was updated to include both the payment gateway and the review feature, showing that users now proceed to payment after booking an appointment and optionally submit a review once the appointment is marked complete.

Link to the updated chart/diagram: [Figure 108: Activity diagram for appointment](#)

7.4.7 Sequence Diagram

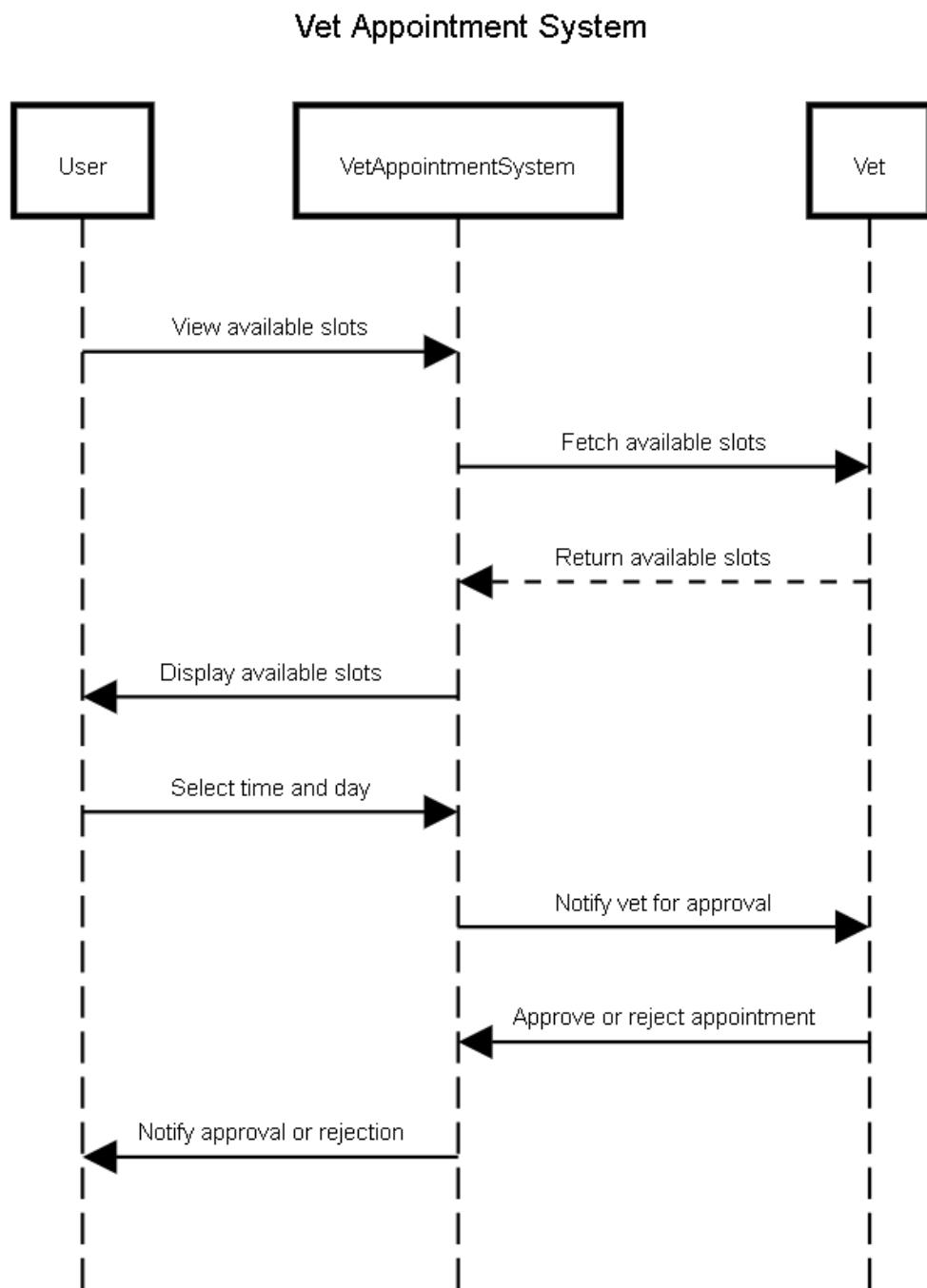


Figure 570 Old Sequence Diagram for appointment

Sequence Diagram Update:

In the sequence diagram, new interactions were added to capture the flow of submitting a review after a successful appointment, alongside the previously introduced payment sequence, ensuring all user actions are properly represented.

Link to the updated chart/diagram: [Figure 109: Sequence diagram for appointment](#)

7.4.8 Collaboration Diagram



Figure 571 Old Collaboration Diagram for appointment

Collaboration Diagram Update:

The collaboration diagram was revised to reflect the exchange between the user, system, and database for both payment processing and review submission, illustrating how system components now work together to support these expanded functionalities.

Link to the updated chart/diagram: [Figure 110: Collaboration for appointment](#)

7.4.9 Class Diagram

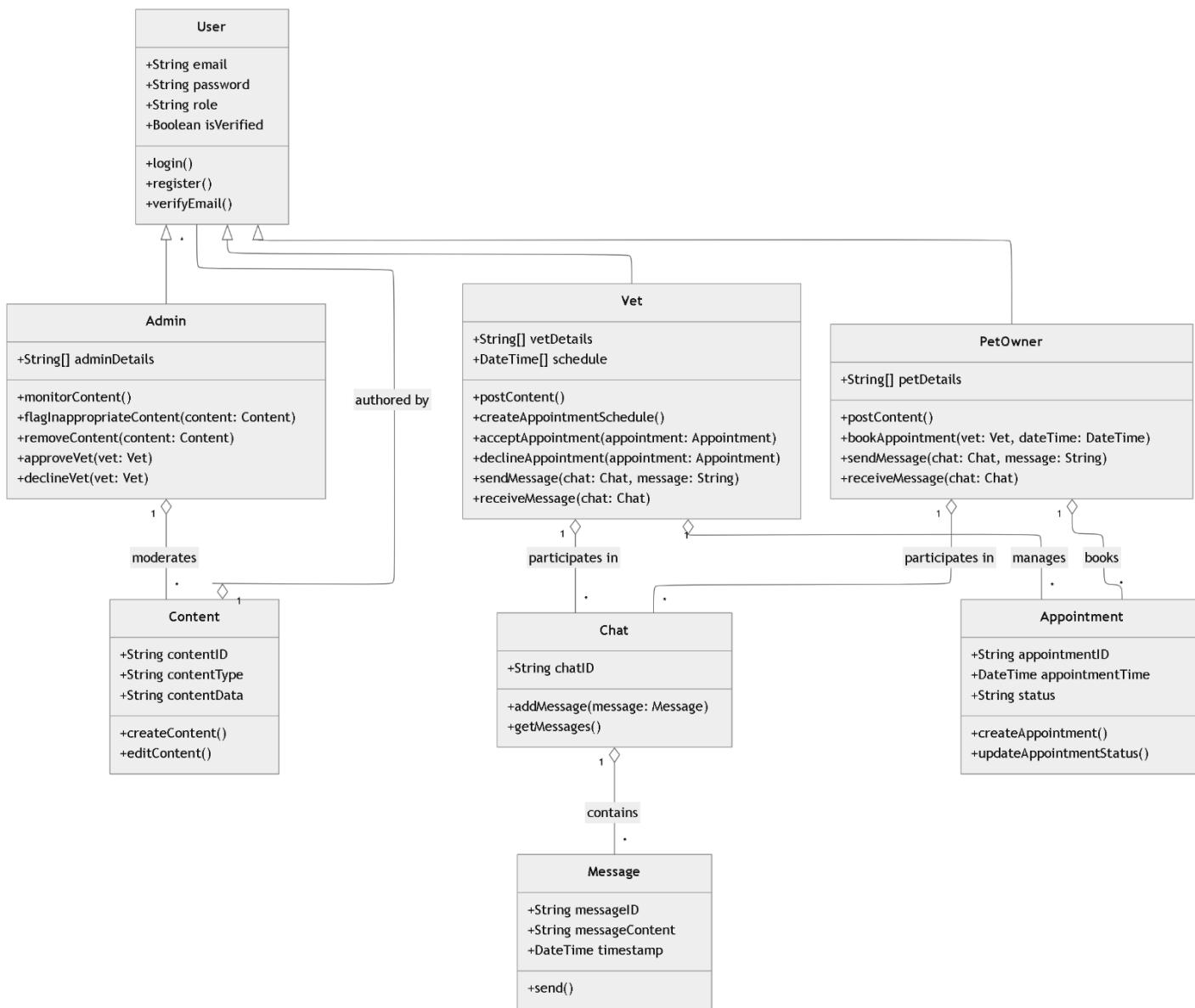


Figure 572 Old Class Diagram

Class Diagram Update:

In addition to the VetSchedule class, the Pet and Review classes have been added to the system. The content structure was also refactored into distinct sections, including posts, replies, and comments, to improve organization and clarity.

Link to the updated chart/diagram: [Figure 12: Final Class Diagram](#)

7.4.10 Architectural Choice

Django's framework for organizing a web application's codebase and workflow is called the MVT pattern. Its three main parts—Model, View, and Template—are each in charge of particular duties that complement one another well.

Here is a brief synopsis of the elements and their functions:

- **Model:** Also referred to as the data layer, it manages database interactions and data.
- **View:** Also known as the logic layer, it acts as a mediator between the Model and Template, controlling data flow and managing logic.
- **Template:** It renders HTML content for the user interface and serves as the presentation layer.

The "Social Petvet" project uses Django's MVT (Model-View-Template) architecture, which provides a well-organized and efficient framework for building web applications. Here's how MVT benefits the project

1. Separation of Concerns

MVT makes the application easier to manage and maintain by ensuring that data handling, business logic, and the user interface are clearly separated.

- **Model:** Models are used to define the project's primary data structures, including Pet, Owner, and Post. These models give a clear relationship between entities by interacting directly with the database.
- **View:** The views work with the data and handle user requests. A view might, for instance, retrieve a pet's information and present it on the pet's profile page.
- **Template:** Templates manage the user's view of data. The name, breed, and posts related to a pet, for instance, can be rendered on the pet's profile page using a template.

2. Development Ease

By dividing the data (model), logic (view), and presentation (template), the MVT framework simplifies the development process. Developers may concentrate on producing features effectively thanks to this well-defined framework.

For instance, users can discuss pet updates or submit details on pets up for adoption on the "Social Petvet" project. MVT makes it easy and quick to deploy this feature.

3. Object-Relational Mapping (ORM)

Django makes database management easier. Without writing intricate SQL queries, developers may quickly create, read, edit, and delete records using Python code. This is especially useful for the "Petvet" project, which requires managing many entities in the database, including pets, owners, and posts.

7.4.11 Draft Figma Design's Board

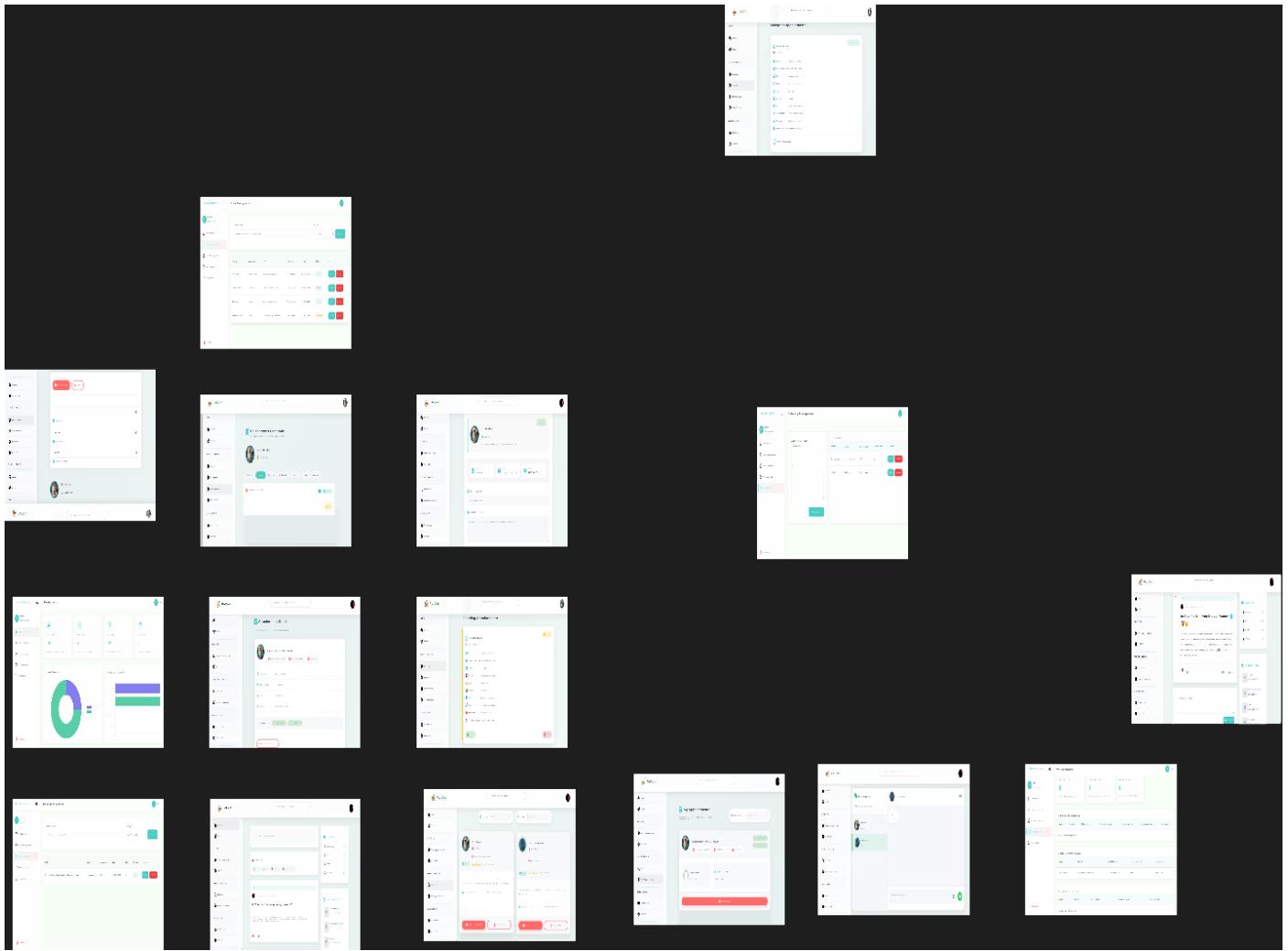


Figure 573 Draft Figma Design's Board

7.4.12 Final Figma Design's Board



Figure 574 Main Figma Design's Board

7.6 Appendix F: User Feedback

7.6.1 User Feedback Form for Initial Development Results

What is your age?

29 responses

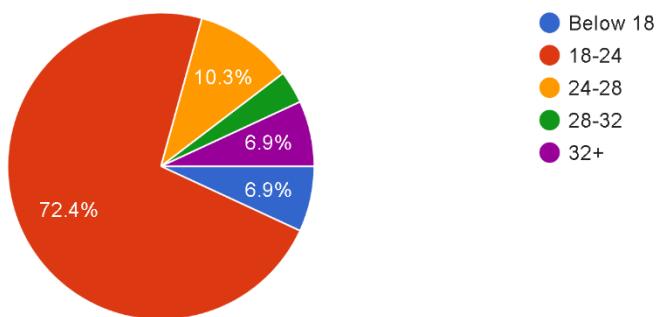


Figure 575 User Feedback form: Initial Development Question 2

How easy was it to register as a pet owner or veterinarian? (1 = Very difficult, 5 = Very easy)

30 responses

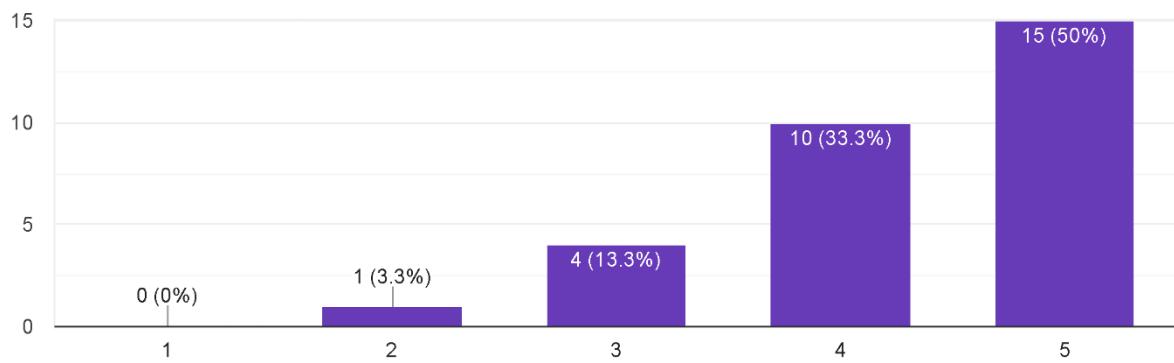


Figure 576 Feedback form: Initial Development Question 3

How clear and understandable was the profile setup process after registration? (1 = Very difficult, 5 = Very easy)
30 responses

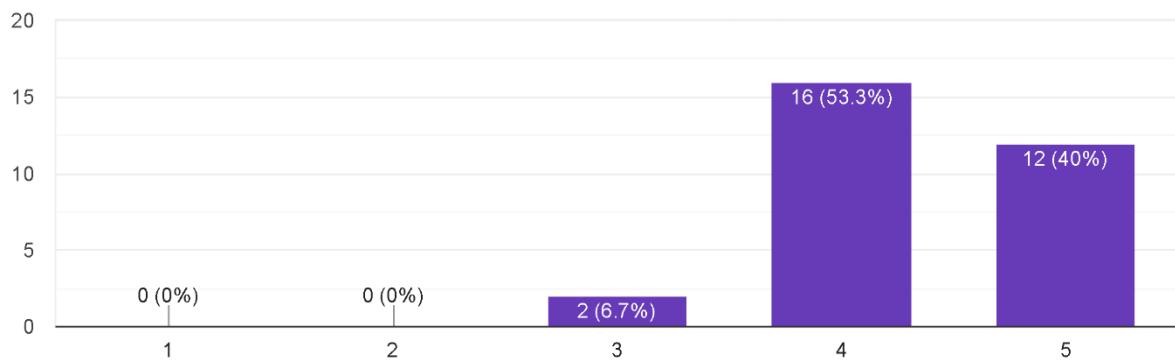


Figure 577 Feedback form: Initial Development Question 4

How smooth was the transition to the OTP verification page (for pet owners)? (1 = Very confusing, 5 = Very smooth)
30 responses

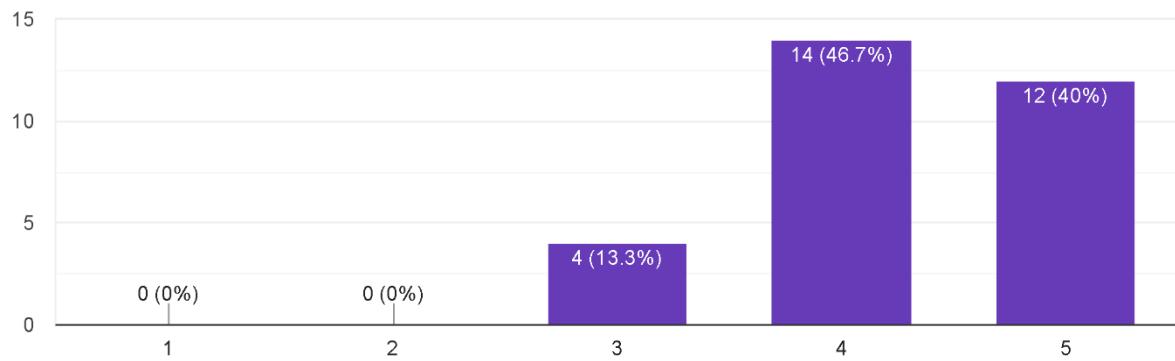


Figure 578 Feedback form: Initial Development Question 5

How clear was the "waiting for approval" page experience (for vets)? (1 = Very unclear, 5 = Very clear)

30 responses

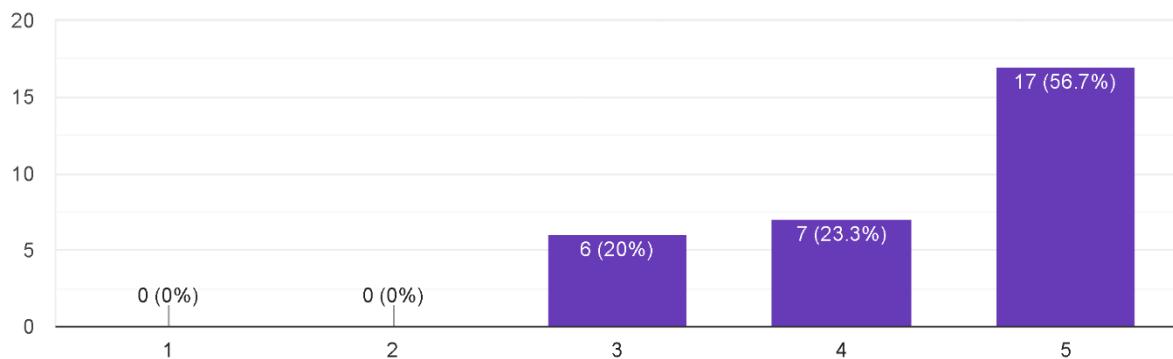


Figure 579 Feedback form: Initial Development Question 6

Did you face any technical issues during the registration or approval flow? Elaborate.

9 responses

Everything worked smoothly. I didn't encounter any problems during registration or approval.

Registration and approval were seamless. No errors or bugs occurred for me.

No i didnt face any technical issues
the website worked smoothly, and the registration was quick and straightforward

The process was very user-friendly, and I didn't experience any technical problems.

Noo the app was easy to use
the steps were very clear

No issues faced. The registration and approval steps were clear and easy to complete

No technical issues at all. The whole process was straightforward and worked perfectly.

i didnt face any technical issues during the registration

Figure 580 Feedback form: Initial Development Question 7

Overall, how satisfied are you with the user registration and onboarding flow? (1 = Very dissatisfied , 5 = Very satisfied)

30 responses

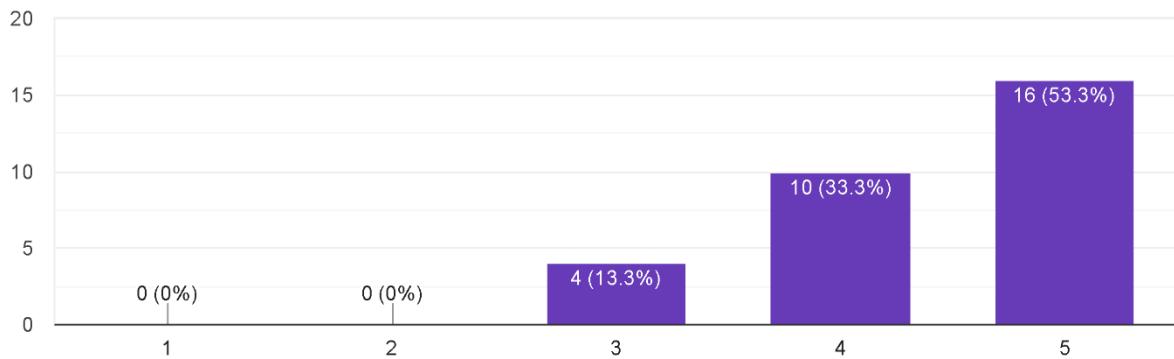


Figure 581 Feedback form: Initial Development Question 8

In general, the users were pleased with the registration and onboarding process.

- **Ease of Registration:** A large portion (83%) reported that registering as a pet owner or as a veterinarian was either easy or very easy (4.3 out of 5 average). This means that the process was easy and intuitive for most of the users.
- **Profile Setup Clarity:** Almost all (93%) found the profile setup process to be clear, a score of 4.4, or clear or very clear. This implies that the post registration instructions and fields were structured and easy to follow.
- **OTP Verification (Pet Owners):** 86% of users (4.5 on average) rated the transition to the OTP verification page as smooth. This proves that pet owners were able to verify the step seamlessly.
- **Vets Approval Process:** 81% of users found the "Waiting for Approval" page for veterinarians to be clear or very clear (58%), and the total of 58% considered it very clear. This page is very clear and has a good design, so the feedback is very good (average score of 4.7).

- **Feedback:** Many users reported no issues, and they stated that the entire process from registration to approval went without errors. This implies that the system was technically sound and user friendly.
- **User Registration and Onboarding Flow:** In general, the satisfaction with the user registration and onboarding flow was very positive, with 86 percent of the users being very satisfied or satisfied, which gave the average rating of 4.5. This indicates a very good level of user experience.

The users found that the registration and onboarding flow was intuitive, easy to navigate.

7.6.2 User Feedback Form for Initial Development Sample Answer

Responses cannot be edited

◆ PetVet Initial Development – User Feedback Form ◆

Thank you for testing the first prototype of PetVet!

This version focuses on user registration (pet owners and vets), profile setup, vet approval system through the admin dashboard, and the flow to OTP verification or waiting-for-approval page.

Please take a few minutes to share your experience to help us improve!

* Indicates required question

What is your name?
Diya Poudel

What is your age?

Below 18
 18-24
 24-28
 28-32
 32+

How easy was it to register as a pet owner or veterinarian? (1 = Very difficult, 5 = Very easy) *

1	2	3	4	5		
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Very Easy

How clear and understandable was the profile setup process after registration? (1 = Very difficult, * 5 = Very easy)

1	2	3	4	5		
Very Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	Very Easy

How smooth was the transition to the OTP verification page (for pet owners)? (1 = Very confusing, 5 = Very smooth)

1	2	3	4	5		
Very Confusing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Very Smooth

How clear was the "waiting for approval" page experience (for vets)? (1 = Very unclear, 5 = Very clear)

1	2	3	4	5		
Very Unclear	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Very Clear

Did you face any technical issues during the registration or approval flow? Elaborate.
Noo the app was easy to use
the steps were very clear

Overall, how satisfied are you with the user registration and onboarding flow? (1 = Very dissatisfied, 5 = Very satisfied)

1	2	3	4	5		
Very dissatisfied	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Very satisfied

Figure 582: Initial Development Sample User Feedback Form

7.6.3 User Feedback Form for Prototype 1 Results

How easy was it to create a post? (1 = Very difficult, 5 = Very easy)

30 responses

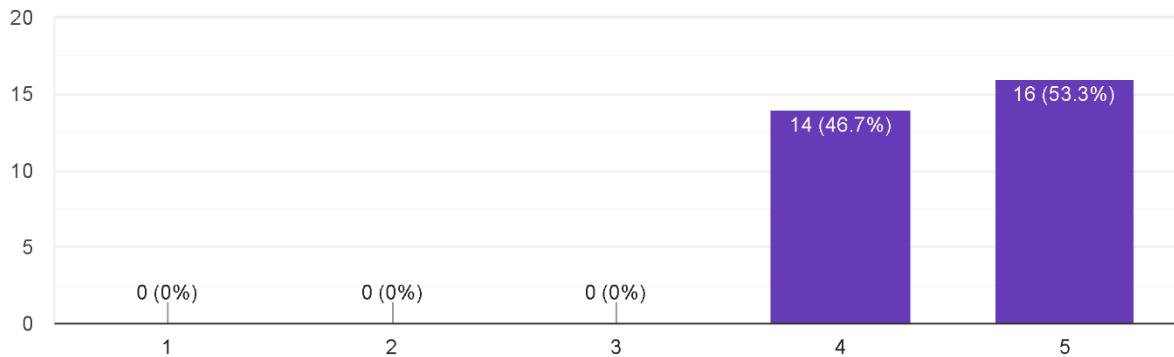


Figure 583 Feedback form: Prototype 1 Question 1

How easy was it to comment and reply to posts? (1 = Very confusing, 5 = Very easy)

30 responses

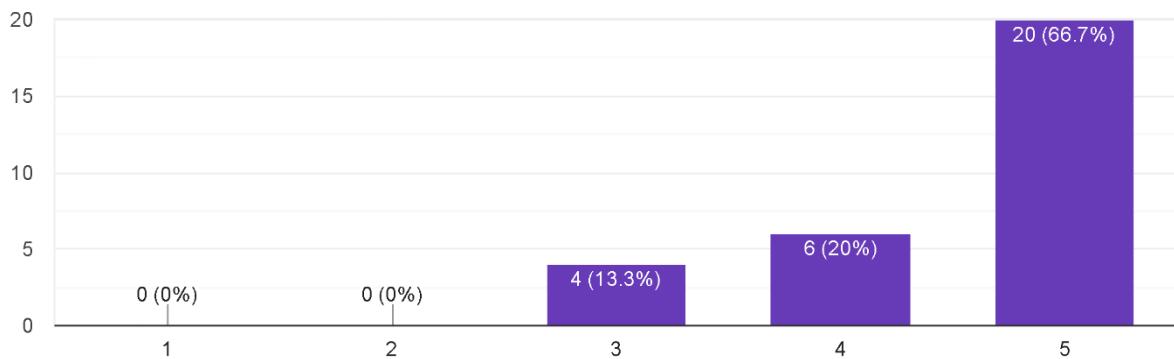


Figure 584 Feedback form: Prototype 1 Question 2

Was the blog interface visually clear and user-friendly? (1 = Not clear at all, 5 = Very clear)
30 responses

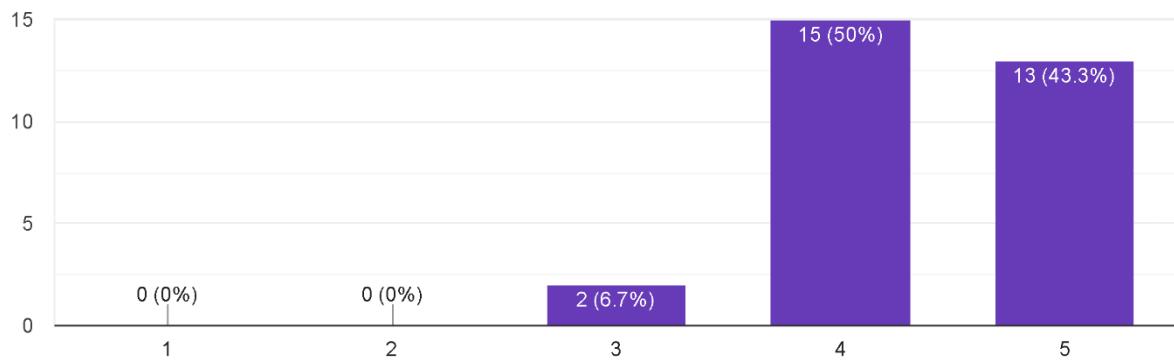


Figure 585 Feedback form: Prototype 1 Question 4

How engaging do you find the posting, commenting and liking features? (1 = Not engaging, 5 = Very engaging)

30 responses

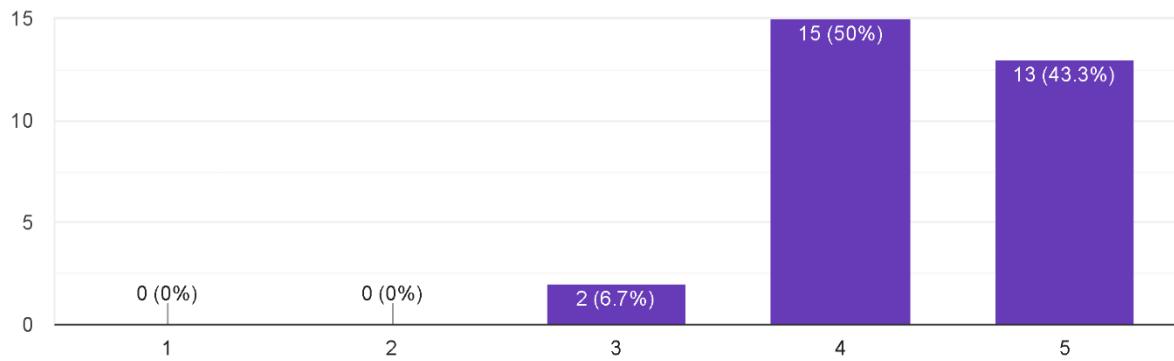


Figure 586 Feedback form: Prototype 1 Question 5

What improvements would you suggest for the blogging experience?

11 responses

Use high-quality relevant images for each post to make the blog visually appealing

The blogging experience is excellent. Everything is intuitive and works as expected!

Share success stories

The blogging experience is great as it is.

A better UI

Everything is great with the blogging experience! No improvements needed from my end.

Everything works well and the blogging experience is smooth.

Improve navigation

Everything feels polished and works perfectly for blogging.

Figure 587 Feedback form: Prototype 1 Question 6

How important do you think this blogging feature is for PetVet? (1 = Not important, 5 = Very important)

30 responses

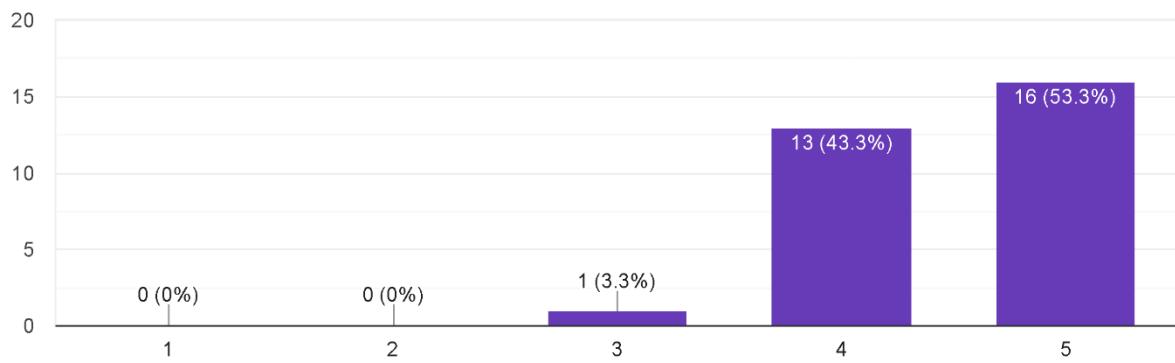


Figure 588 Feedback form: Prototype 1 Question 7

Overall, how satisfied are you with the blogging feature? (Linear scale: 1 = Very dissatisfied, 5 = Very satisfied)

30 responses

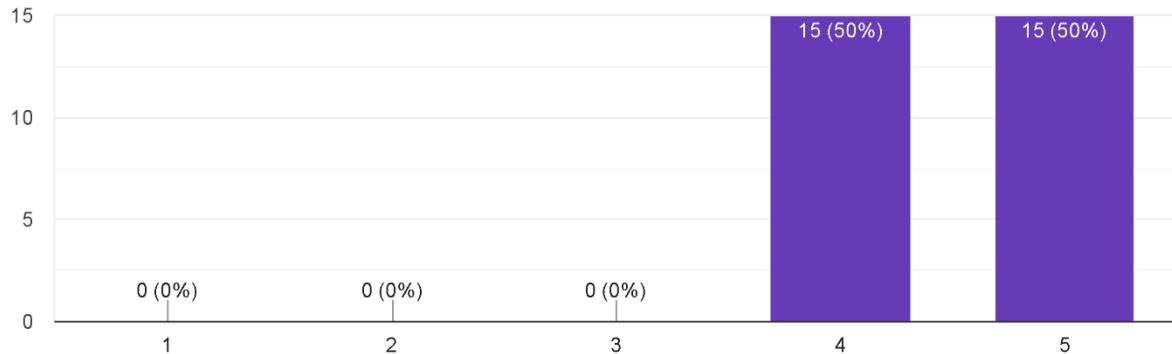


Figure 589 Feedback form: Prototype 1 Question 8

After developing and integrating the blogging functionality, a prototype was tested by the same group of 30 users. The feedback collected was as follows:

1. Ease of creating a post:

- 53% rated it 5 (Very easy)
- 45% rated it 4

2. Ease of commenting and replying to posts:

- 66% rated it 5 (Very easy)
- 20% rated it 4
- 13% rated it 3

3. Clarity and user-friendliness of the blog interface:

- 43% rated it 5 (Very clear)
- 50% rated it 4
- 6.7% rated it 3

4. Engagement with posting, commenting, and liking features:

- 43% rated it 5 (Very engaging)
- 50% rated it 4
- 6.7% rated it 3

5. Importance of the blogging feature for PetVet:

- 53% rated it 5 (Very important)
- 43% rated it 4
- 3.3% rated it 3

6. Overall satisfaction with the blogging feature:

- 50% rated it 5 (Very satisfied)
- 50% rated it 4

Overall, users found the blogging feature to be highly usable, engaging, and important for the PetVet platform. Creating posts and interacting through comments and likes was considered easy and intuitive. The majority of users praised the interface for being clear and user-friendly. Some suggestions for improvement included enhancing the UI, improving navigation, ensuring mobile responsiveness, and using high-quality images in posts to make the blog more visually appealing. However, a significant number of users also mentioned that the experience was already smooth and polished, indicating strong initial success of the feature.

7.6.4 User Feedback Form for Prototype 1 Sample Answer

30 responses

[Link to Sheets](#)

Summary Question Individual

< 4 of 30 >

Responses cannot be edited

◆ PetVet Prototype 1 – User Feedback Form ◆

This version introduces blogging features where users can create posts, comment, and reply in a Facebook-like interaction style.

* Indicates required question

What is your name?
Nischal Marasini

How easy was it to create a post? *

(1 = Very difficult, 5 = Very easy)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

How easy was it to comment and reply to posts?

(1 = Very confusing, 5 = Very easy)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Was the blog interface visually clear and user-friendly?

(1 = Not clear at all, 5 = Very clear)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

How engaging do you find the posting, commenting and liking features?

(1 = Not engaging, 5 = Very engaging)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

What improvements would you suggest for the blogging experience?

Improve navigation

How important do you think this blogging feature is for PetVet?

(1 = Not important, 5 = Very important)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Overall, how satisfied are you with the blogging feature?

(Linear scale: 1 = Very dissatisfied, 5 = Very satisfied)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Submitted 07/07/2023, 1:41:20

Figure 590 Prototype 1 Sample User Feedback form

7.6.5 User Feedback Form for Prototype 2 Results

Were you able to book an appointment successfully?

30 responses

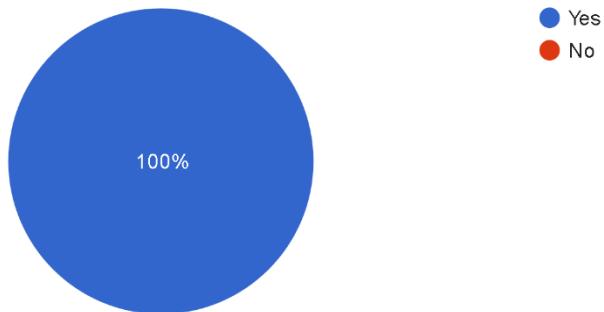


Figure 591 Feedback form: Prototype 2 Question 2

How easy was it for veterinarians to create and manage their available schedules? (1 = Very difficult, 5 = Very easy)

30 responses

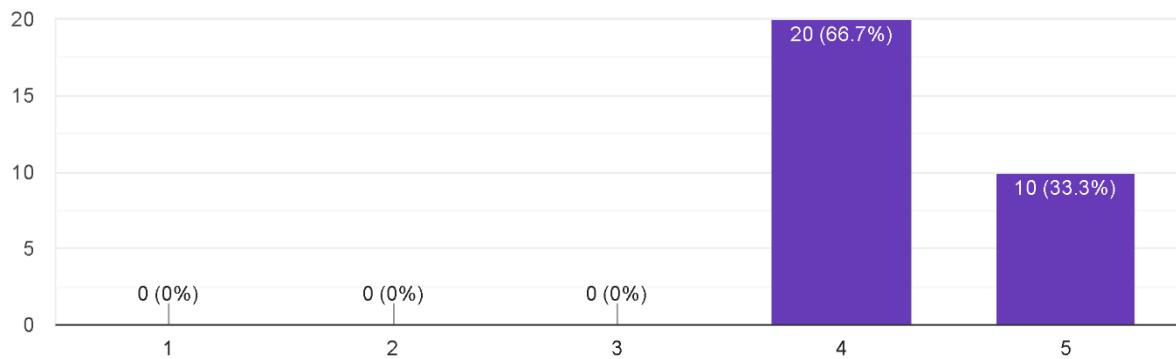


Figure 592 Feedback form: Prototype 2 Question 3

Was the appointment acceptance/rejection system by vets clear and easy to understand? (1 = Very confusing, 5 = Very clear)

30 responses

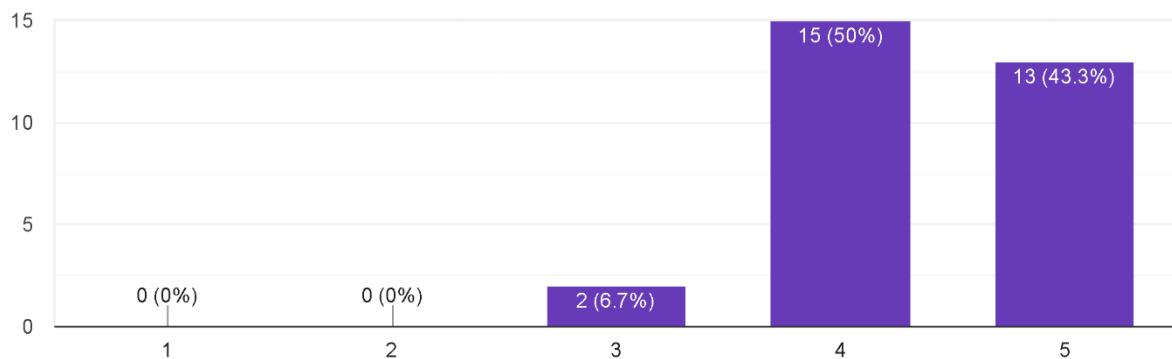


Figure 593 Feedback form: Prototype 2 Question 4

Was the appointment booking process understandable even without a full frontend? (1 = Very confusing, 5 = Very clear)

30 responses

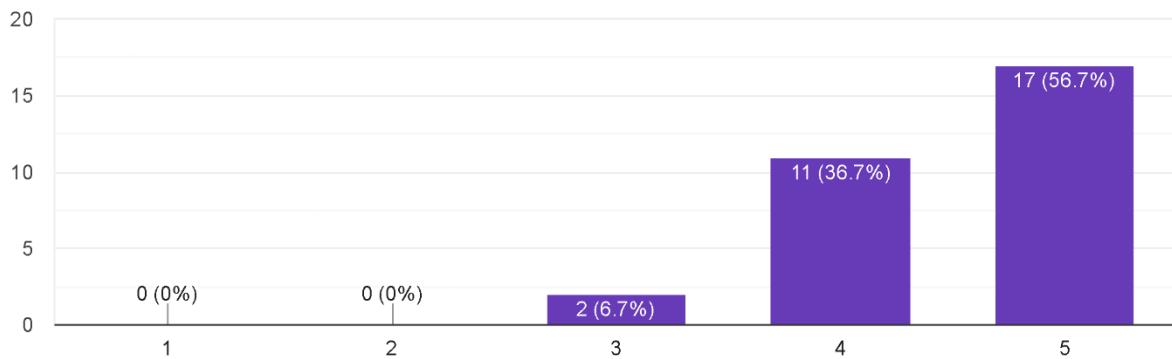


Figure 594 Feedback form: Prototype 2 Question 5

Was the Khalti payment experience smooth? (1 = Very difficult, 5 = Very smooth)
30 responses

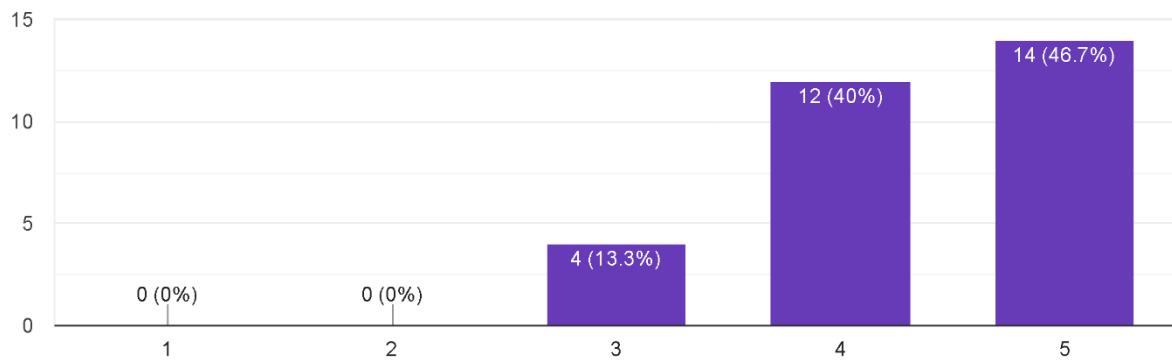


Figure 595 Feedback form: Prototype 2 Question 6

How confident are you about the security of Khalti transactions? (1 = Not confident, 5 = Very confident)
30 responses

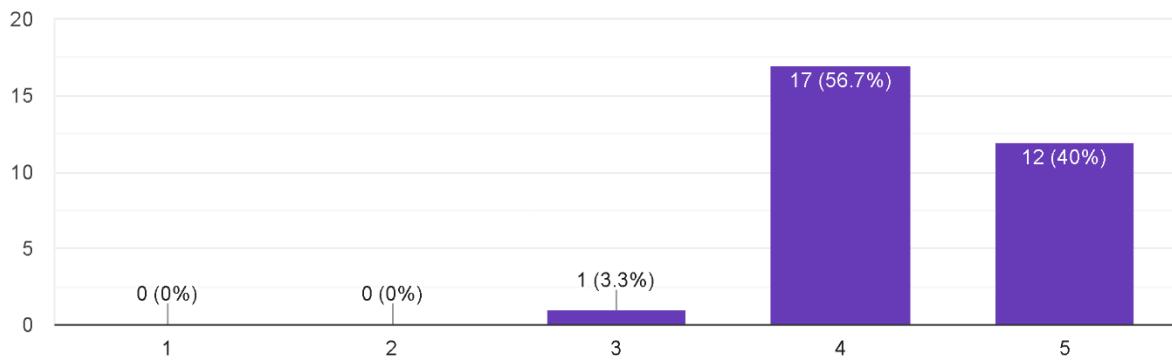


Figure 596 Feedback form: Prototype 2 Question 7

Would you like a system where rejected or canceled appointment payments are stored as platform credits for future use?

30 responses

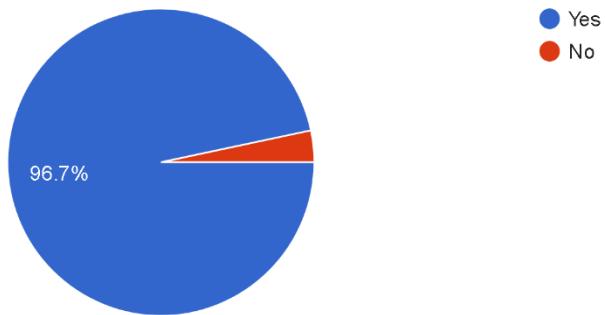


Figure 597 Feedback form: Prototype 2 Question 8

What improvements would you suggest for the appointment or payment system?

7 responses

everything is running well; no improvements needed at this time.

Send automated SMS/email reminders for upcoming appointments, with options to confirm, cancel, or reschedule

Provide options to reschedule or cancel directly from the reminder

Clearly list prices for common procedures and offer estimates during appointment booking.

The appointment and payment systems work smoothly with no issues.

The process is simple and efficient, no changes required.

Reduce no shows by allowing deposits or full pre payments

Figure 598 Feedback form: Prototype 2 Question 9

Overall, how satisfied are you with the appointment and payment process? (1 = Very dissatisfied, 5 = Very satisfied)
29 responses

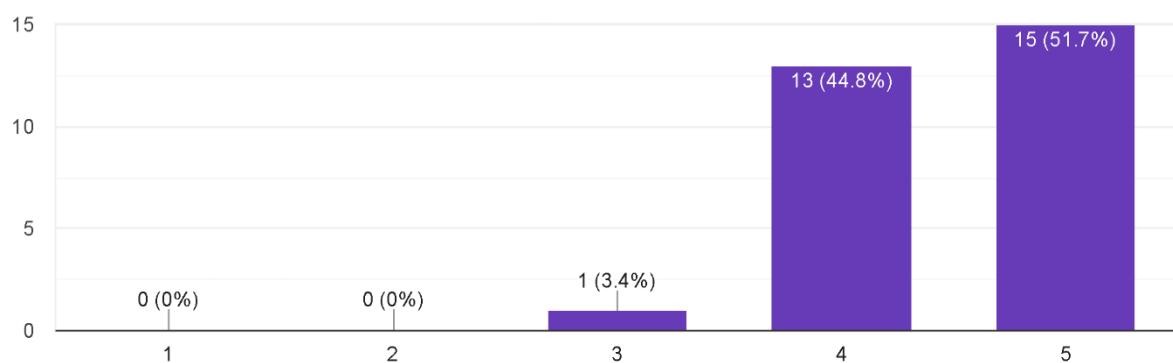


Figure 599 Feedback form: Prototype 2 Question 1

The appointment booking and payment system prototype was tested by the same sample group of 30 users. The feedback collected is summarized below:

1. Were users able to book an appointment successfully?
 - Yes — 100% success rate
2. Ease of veterinarians creating and managing available schedules:
 - 33% rated it 5 (Very easy)
 - 66% rated it 4
3. Clarity of the appointment acceptance/rejection system by vets:
 - 43% rated it 5 (Very clear)
 - 50% rated it 4
 - 6.7% rated it 3
4. Understanding of the appointment booking process even without full frontend UI:
 - 56% rated it 5 (Very clear)

- 36% rated it 4
- 6% rated it 3

5. Smoothness of the Khalti payment experience:

- 46% rated it 5 (Very smooth)
- 40% rated it 4
- 13% rated it 3

6. Confidence in the security of Khalti transactions:

- 40% rated it 5 (Very confident)
- 56% rated it 4
- 3.3% rated it 3

7. Would users like a credit system for rejected/canceled appointment payments?

- Yes — 96% agreed

8. Overall satisfaction with the appointment and payment process:

- 51% rated it 5 (Very satisfied)
- 44% rated it 4
- 3.3% rated it 3

The users received the appointment and payment system very positively. It was possible for everyone to book an appointment. The schedule creation and management was easy for the veterinarians and the appointment acceptance and rejection was clear to users. Despite a lack of a fully polished frontend, users could understand and complete bookings without confusion. It was easy to integrate Khalti payment and users had high confidence in its security.

In addition, 96% of users expressed interest in having rejected or canceled payments stored as platform credits in the future.

Provide clear pricing estimates while booking and suggest SMS/email reminder before appointment. The system was praised overall as efficient, simple and user friendly.

7.6.6 User Feedback Form for Prototype 2 Sample Answer

30 responses

[Link to Sheets](#)

Summary Question Individual

< 9 of 30 >

Responses cannot be edited.

◆ PetVet Prototype 2 – User Feedback Form ◆

This version adds the backend for the appointment system and integrates Khalti payment, although the frontend is still basic.

What is your name?
Hausala Rasalli

Were you able to book an appointment successfully?
 Yes
 No

How easy was it for veterinarians to create and manage their available schedules?
(1 = Very difficult, 5 = Very easy)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Was the appointment acceptance/rejection system by vets clear and easy to understand?
(1 = Very confusing, 5 = Very clear)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Was the appointment booking process understandable even without a full frontend?
(1 = Very confusing, 5 = Very clear)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Was the Khalti payment experience smooth?
(1 = Very difficult, 5 = Very smooth)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

How confident are you about the security of Khalti transactions?
(1 = Not confident, 5 = Very confident)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Would you like a system where rejected or canceled appointment payments are stored as platform credits for future use?
 Yes
 No

What improvements would you suggest for the appointment or payment system?
Send automated SMS/email reminders for upcoming appointments, with options to confirm, cancel, or reschedule

Overall, how satisfied are you with the appointment and payment process?
(1 = Very dissatisfied, 5 = Very satisfied)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Figure 600 Prototype 2 Filled User Feedback form

626

7.6.7 User Feedback Form for Prototype 3 Results

How easy was it to start a chat with another user? (1 = Very difficult, 5 = Very easy)

30 responses

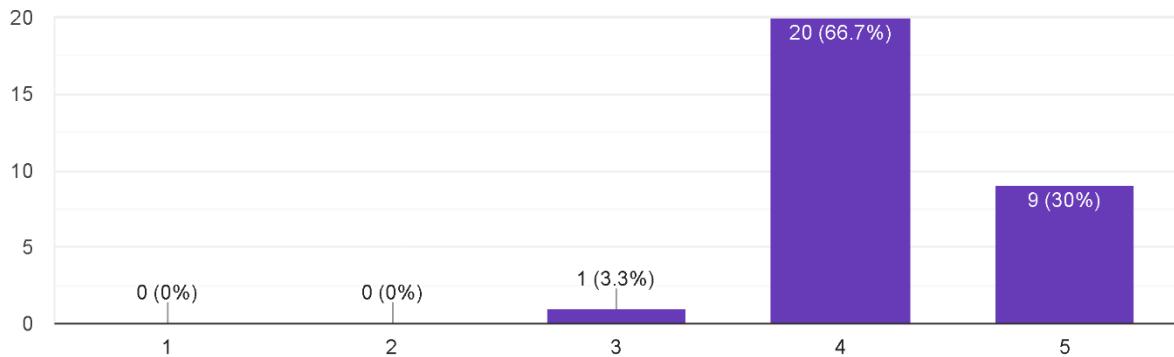


Figure 601 Feedback form: Prototype 3 Question 2

How responsive was the chat system (message speed)? (1 = Very slow, 5 = Very fast)

30 responses

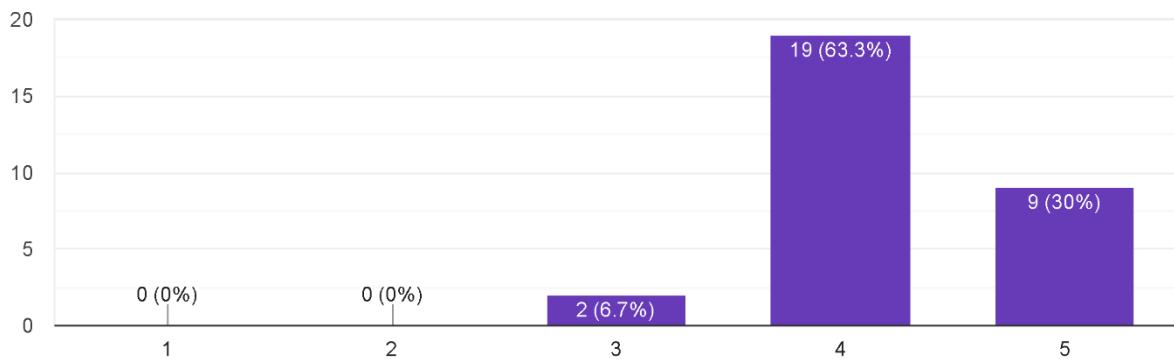


Figure 602 Feedback form: Prototype 3 Question 3

Was the chat interface simple and understandable? (1 = Very confusing, 5 = Very clear)
29 responses

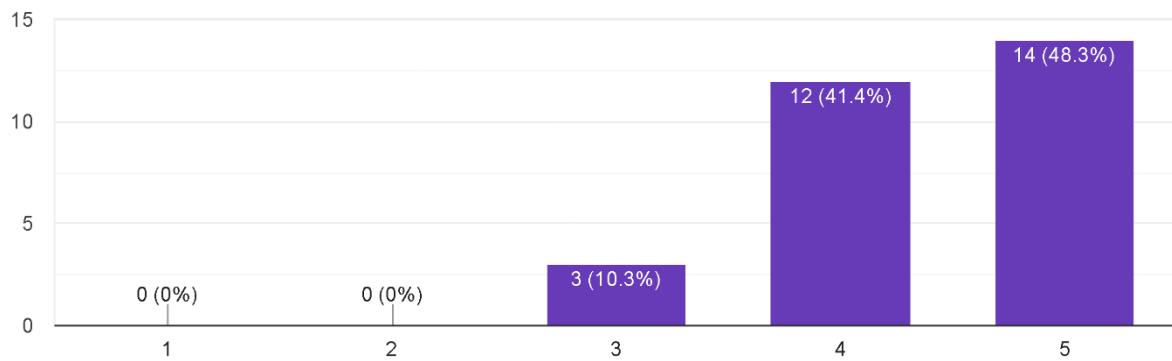


Figure 603 Feedback form: Prototype 3 Question 4

How useful is the chat feature for PetVet users? (1 = Not useful, 5 = Very useful)
30 responses

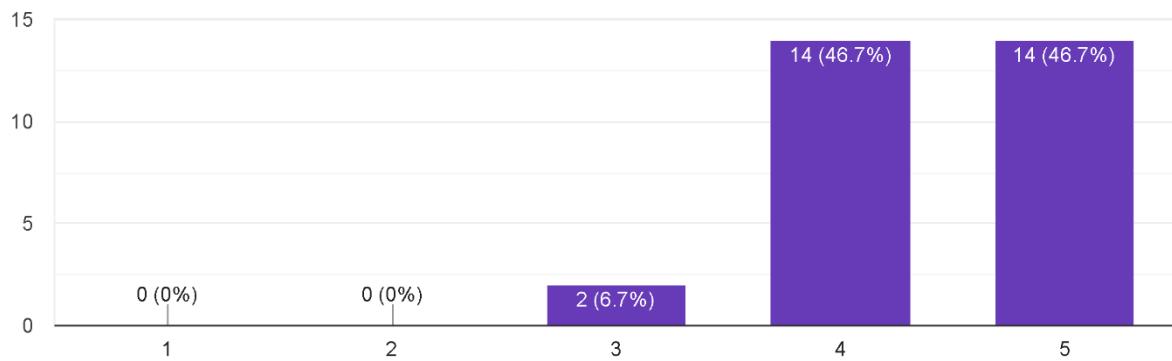


Figure 604 Feedback form: Prototype 3 Question 5

What improvements would you suggest for the chat system?

7 responses

The chat system works well and is easy to use.

The chat system is simple and functional as it is.

If the chat is unavailable, provide an option to leave a message or have a chatbot capture the inquiry, which can be responded to later via email

Make the chat interface simple, visually appealing, and easy to use on both desktop and mobile device

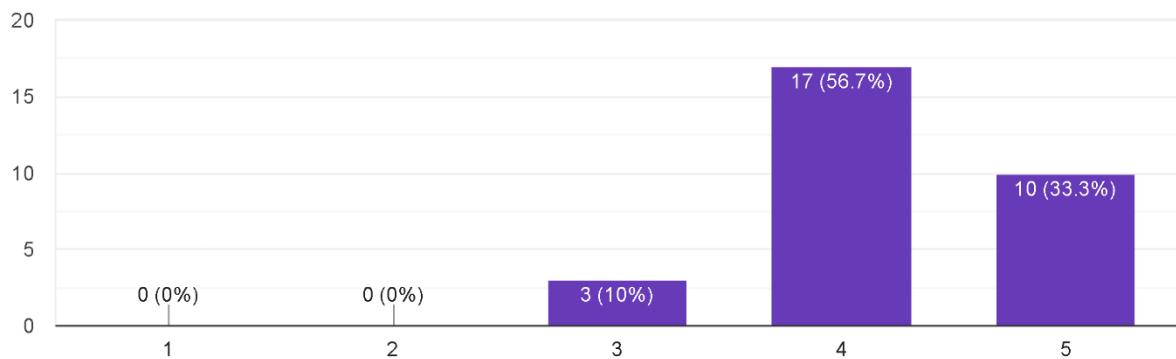
Integrate with various messaging platforms (Facebook Messenger, WhatsApp, etc.) to provide a more personalized and flexible communication option

No improvements needed; the chat is clear and effective.

Everything in the chat system feels smooth and responsive.

*Figure 605 Feedback form: Prototype 3 Question 6***Overall, how satisfied are you with the chat system?**

30 responses

*Figure 606 Feedback form: Prototype 3 Question 7*

The chat system prototype was tested with the same sample group of 30 users. The results of the survey are summarized below:

1. Ease of starting a chat with another user:

- 30% rated it 5 (Very easy)
- 66% rated it 4
- 3.3% rated it 3

2. Responsiveness of the chat system (message speed):

- 30% rated it 5 (Very fast)
- 63% rated it 4
- 6.7% rated it 3

3. Clarity and simplicity of the chat interface:

- 48% rated it 5 (Very clear)
- 41% rated it 4
- 10% rated it 3

4. Usefulness of the chat feature for PetVet users:

- 46% rated it 5 (Very useful)
- 46% rated it 4
- 6.7% rated it 3

5. Overall satisfaction with the chat system:

- 33% rated it 5 (Very satisfied)
- 56% rated it 4
- 10% rated it 3

Overall, the chat system was mainly well received. It was easy to start a chat and most users found message delivery to be fast. Most of the participants found the interface simple, clear and understandable. PetVet's users believed that the chat feature was an important and useful addition to the functionality of PetVet. Users found the chat system to be very smooth, functioning well and very responsive, and there were no major technical problems they reported.

7.6.8 User Feedback Form for Prototype 3 Sample Answer

30 responses

[Link to Sheets](#)

Summary Question Individual

< 7 of 30 >

Responses cannot be edited

◆ PetVet Prototype 3 — User Feedback Form ◆

This version introduces a real-time chat system with a simple user interface to enable communication between pet owners and vets.

* Indicates required question

What is your name
Anshu Khadka

How easy was it to start a chat with another user?
(1 = Very difficult, 5 = Very easy)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

How responsive was the chat system (message speed)?
(1 = Very slow, 5 = Very fast)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Was the chat interface simple and understandable?
(1 = Very confusing, 5 = Very clear)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

How useful is the chat feature for PetVet users?
(1 = Not useful, 5 = Very useful)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

What improvements would you suggest for the chat system?
Make the chat interface simple, visually appealing, and easy to use on both desktop and mobile device

Overall, how satisfied are you with the chat system? *

1	2	3	4	5

Figure 607 Prototype 3 Filled User Feedback Form

632

7.6.9 User Feedback Form for Prototype 4 Results

How easy was it to moderate posts or approve vets? (1 = Very difficult, 5 = Very easy)

31 responses

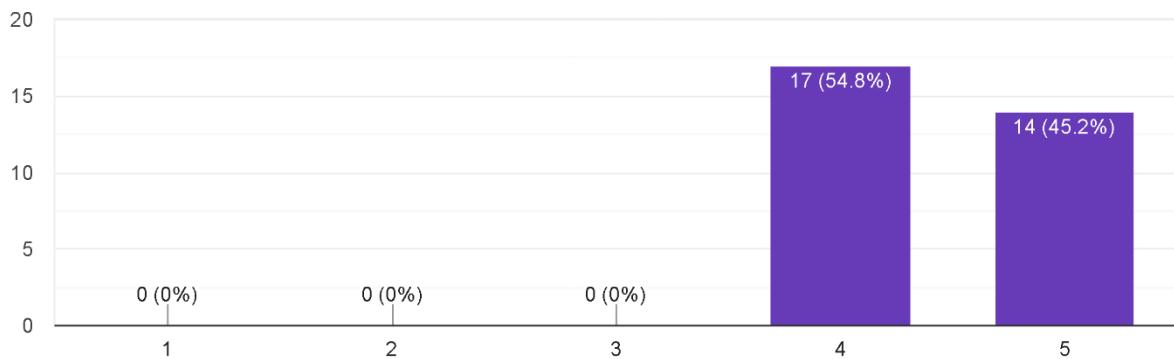


Figure 608 Feedback form: Prototype 4 Question 2

Was the admin dashboard more organized after this update? (1 = Very disorganized, 5 = Very organized)

29 responses

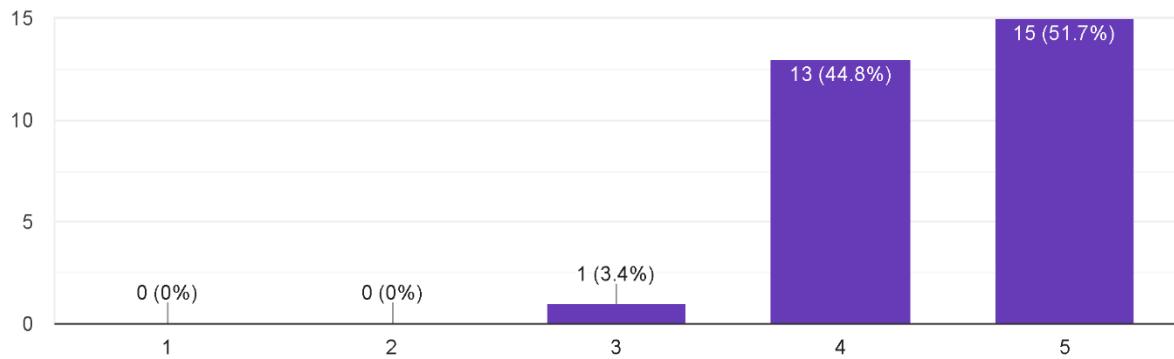


Figure 609 Feedback form: Prototype 4 Question 3

Which admin features did you find most useful?

31 responses

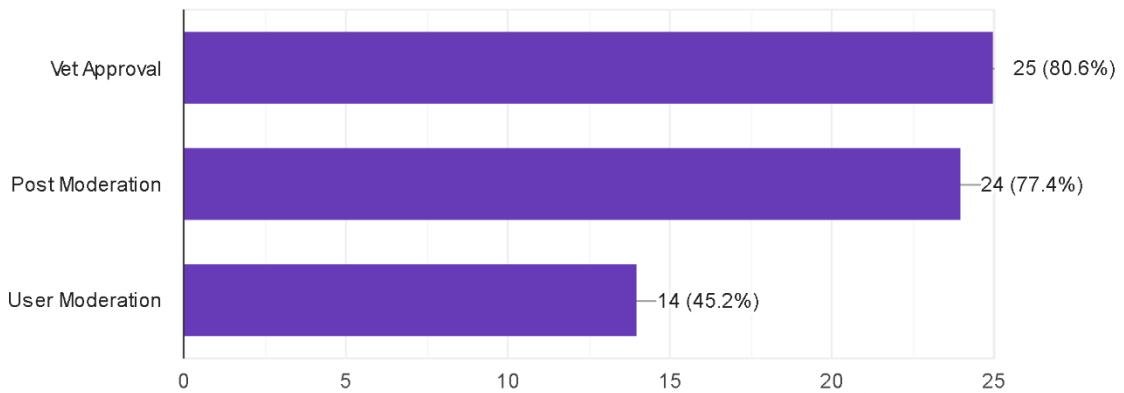


Figure 610 Feedback form: Prototype 4 Question 4

Overall, how satisfied are you with the admin dashboard improvements?

31 responses

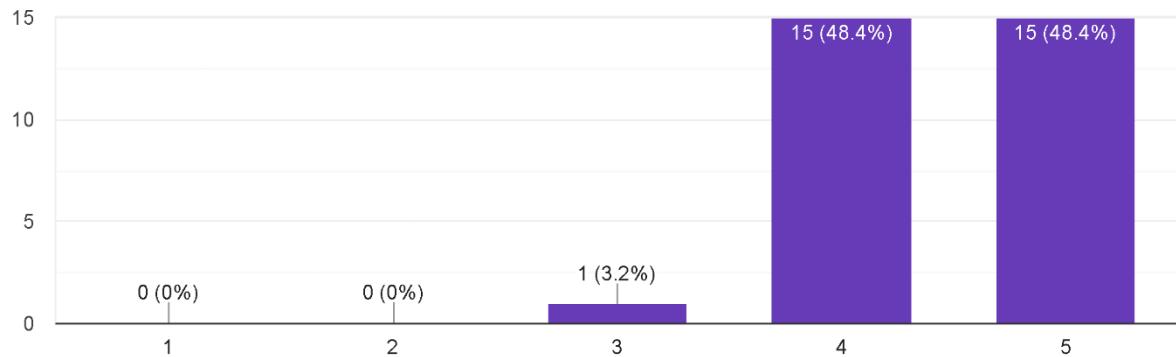


Figure 611 Feedback form: Prototype 4 Question 5

The improved admin dashboard was tested with the same sample group of 30 users.

The summarized results are as follows:

1. Ease of moderating posts or approving vets
 - 45% rated it 5 (Very easy)

- 54.8% rated it 4
2. Organization and clarity of the updated admin dashboard:
- 51.7% rated it 5 (Very organized)
 - 44.8% rated it 4
 - 3.4% rated it 3
3. Most useful admin features:
- Vet approval: 80%
 - Post moderation: 77%
 - User moderation: 45%
4. Overall satisfaction with admin dashboard improvements:
- 48% rated it 5 (Very satisfied)
 - 48% rated it 4
 - 3.2% rated it 3

The admin dashboard update was so well liked. It was easy for users to moderate posts and approve veterinarians. Most of them thought the dashboard was organized, intuitive and effective at managing key tasks. The most valuable functionalities were vet approval and post moderation.

Most users were completely satisfied and had no immediate need for changes, but a few minor suggestions included keeping the layout simple and information easy to see. Overall the feedback was positive with users enjoying the smooth and easy way the new dashboard made managing their clients.

7.6.10 User Feedback Form for Prototype 4 Sample Answer

Responses cannot be edited

◆ PetVet Prototype 4 – User Feedback Form ◆

This version refines the admin dashboard, improving vet approval workflows and introducing content moderation tools.

* Indicates required question

What is your name?
shubham

How easy was it to moderate posts or approve vets?
(1 = Very difficult, 5 = Very easy)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Was the admin dashboard more organized after this update?
(1 = Very disorganized, 5 = Very organized)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Which admin features did you find most useful?

Vet Approval
 Post Moderation
 User Moderation

What improvements would you suggest for the admin dashboard?
The admin dashboard is intuitive and easy to navigate.

Overall, how satisfied are you with the admin dashboard improvements? *

1	2	3	4	5

Figure 612 Prototype 4 Filled User Feedback Form

7.6.11 User Feedback Form for Final Prototype Results

How visually appealing do you find the new design? (Linear scale: 1 = Not appealing, 5 = Very appealing)

30 responses

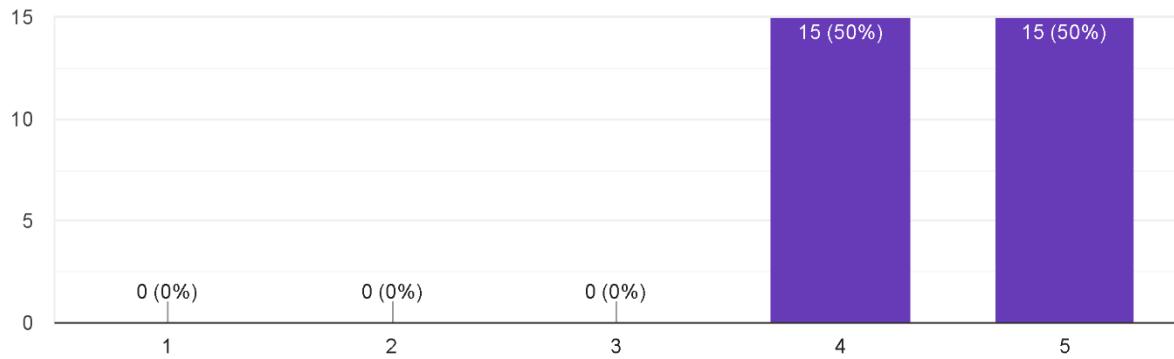


Figure 613 Feedback form: Final Prototype Question 2

Was navigation between features (Blog, Appointment, Chat, etc.) smooth? (1 = Very confusing, 5 = Very smooth)

30 responses

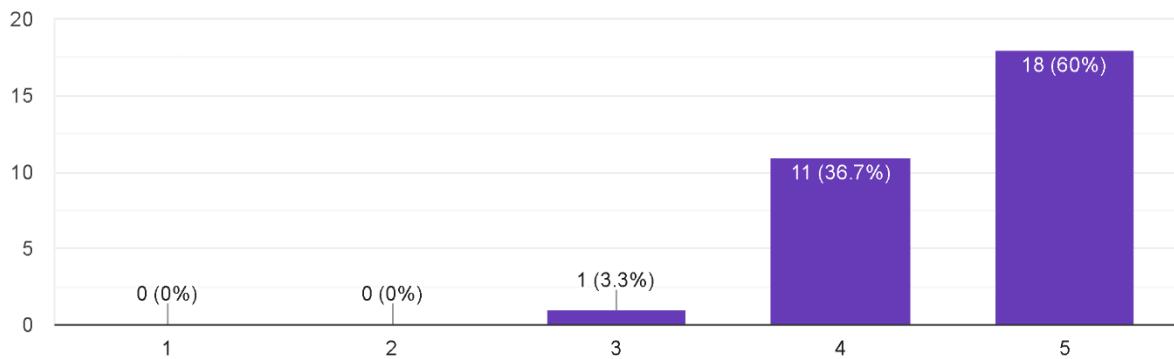


Figure 614 Feedback form: Prototype 4 Question 3

How would you rate the overall platform speed and performance?

30 responses

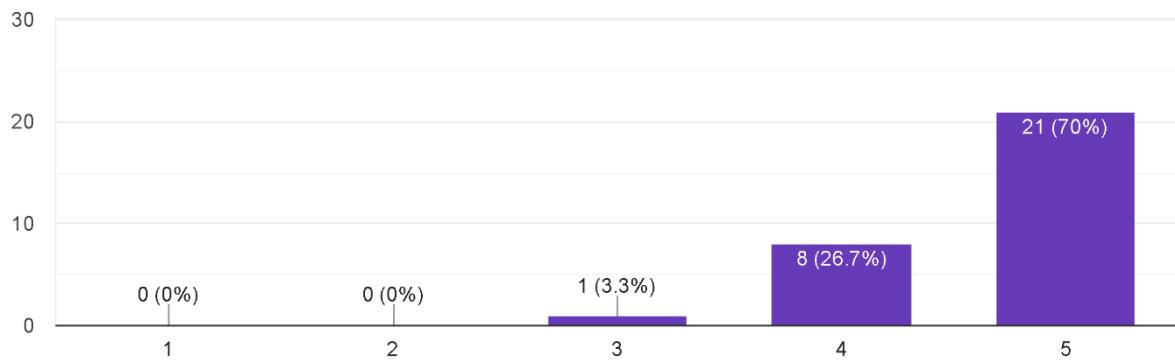


Figure 615 Feedback form: Prototype 4 Question 4

What was your favorite improvement compared to earlier versions?

6 responses

Compared to earlier versions, the new site offers a smoother user experience and easier access to essential services

I really like the faster loading times compared to before.

The new mobile responsiveness is a big plus for me.

My favorite improvement compared to earlier versions is the addition of the Personalized Pet Health Dashboard

The updated design makes the site much easier to use.

One of my favorite improvements in later versions of the PetVet is the enhanced user interface and streamlined appointment booking system

Figure 616 Feedback form: Prototype 4 Question 5

What additional features or improvements would you like to see next?

7 responses

Sell pet meds, food, toys, grooming products

The site is well-equipped as is. Improvements can be made gradually over time.

No additional features needed at the moment, but I'm open to future enhancements.

I'm happy with the current features and excited to see what comes next.

Partner with shelters to show adoptable pets

The website already has great features. I look forward to seeing more updates in the future

Add a calendar-based system with real-time vet availability

Figure 617 Feedback form: Prototype 4 Question 6

Overall, how likely are you to recommend PetVet to others? (1 = Very unlikely, 5 = Very likely)

30 responses

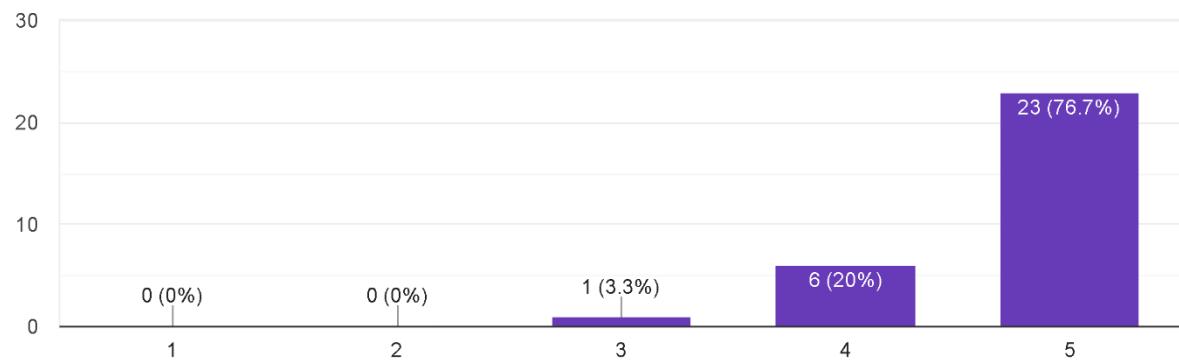


Figure 618 Feedback form: Prototype 4 Question 7

The final version of the PetVet platform was evaluated by users, and the summarized results are as follows:

1. Visual appeal of the new design:

- 50% rated it 5 (Very appealing)
 - 50% rated it 5 (appears a duplicate in survey but implies strong approval)
2. Ease of navigation across features (Blog, Appointment, Chat, etc.):
- 60% rated it 5 (Very smooth)
 - 36% rated it 4
 - 3.3% rated it 3
3. Platform speed and performance:
- 70% rated it 5 (Excellent)
 - 26% rated it 4
 - 3.3% rated it 3
4. Overall likelihood to recommend PetVet:
- 76% rated it 5 (Very likely)
 - 20% rated it 4
5. Favorite Improvements:
- Smoother user experience and easier access to services
 - Faster loading times
 - Improved mobile responsiveness
 - Updated design and better usability
 - Enhanced user interface and streamlined appointment booking
6. Suggestions for Future Features:
- Introduce an online store for pet medicines, food, toys, and grooming products

- Partner with shelters to showcase adoptable pets
- Add a real-time calendar system showing vet availability
- Gradual improvements over time as current features already meet user needs

I would like to say that the feedback from the users to the new design was overwhelmingly praising of the new speed, clarity, and navigation.

The strong recommendation rate shows that the platform's core functionalities and design were well received. The most appreciated elements were the better mobile experience, faster performance and simpler workflow.

Most of them found the platform feature complete and some of the future enhancements that they would like to see include e-commerce integration and real time vet calendars to make the experience better.

7.6.12 User Feedback Form for Final Prototype Sample Answer

Responses cannot be edited

◆ PetVet Final Prototype – User Feedback Form

◆

This final version integrates frontend UI improvements, better logic across the system, and minor bug fixes for a more polished experience.

What is your name?
Ashish Thapa

How visually appealing do you find the new design?
(Linear scale: 1 = Not appealing, 5 = Very appealing)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Was navigation between features (Blog, Appointment, Chat, etc.) smooth?
(1 = Very confusing, 5 = Very smooth)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

How would you rate the overall platform speed and performance?

1	2	3	4	5
★	★	★	★	☆

What was your favorite improvement compared to earlier versions?
One of my favorite improvements in later versions of the PetVet is the enhanced user interface and streamlined appointment booking system

What additional features or improvements would you like to see next?
Add a calendar-based system with real-time vet availability

Overall, how likely are you to recommend PetVet to others?
(1 = Very unlikely, 5 = Very likely)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Figure 619 Final Prototype Filled User Feedback Form

642

7.7 Appendix G: Future Work

7.7.1 Reading for Future Work

Here are four additional future enhancement ideas:

- **Telemedicine Integration:** A secure video consultation feature would introduce the possibility of vet check ups, initial diagnoses or follow up consultations. It is especially helpful in a rural setting or in emergency situations, where in person visits are not possible.
- **Multi-language Support:** Adding multilingual options would also be beneficial for the platform to be more inclusive and more accessible for a wider pool of demography especially in the regions with rich linguistic diversity.
- **AI-Powered Symptom Checker:** For example, it would integrate a basic AI driven symptom checker that will allow pet owners to input the visible symptoms (picture or even a written ones) and receive preliminary suggestions or alerts to guide them when to book an appointment.
- **Mobile Application Development:** While the current version is web-based, developing a dedicated mobile application would significantly improve accessibility and user experience. Mobile apps are widely used for on-the-go interactions, especially for tasks like appointment booking and real-time communication.
- **Campaigning & Event Features:** Having features to promote and organize events such as vaccination drives, adoption drives, and awareness programs can make the platform appeal to pet organizations and NGOs. This will not only enhance community participation but will be one step towards the well-being of the animals too.
- **Rescue Mission Coordination:** Having the option to include a module to plan and execute pet rescue missions would make the platform more valuable for use in emergency situations and community activities. This module can facilitate cooperation with local rescue groups and animal welfare organizations.

- **Real-Time Notifications Within the App:** Although real-time chat has already been implemented, along with sending the appointment updates and cancellations through emails, including real-time in-app notification of appointment status, confirmation, and updates would be immensely helpful to enhance the interaction of the platform. It would keep the user informed too, making the platform responsive so that such crucial updates are not missed.

7.8 Appendix H: Considered Methodologies

Methodology	Overview	Why Not Suitable for This Project
Scrum	A team-oriented framework with fixed sprints, regular stand-ups, and strong collaboration.	<ul style="list-style-type: none"> 1. Requires roles like Scrum Master and Product Owner, which are not relevant in a solo project. 2. Frequent sprint planning and reviews add unnecessary overhead when the developer and decision-maker are the same person.
Extreme Programming	Emphasizes continuous integration, pair programming, and customer feedback.	<ul style="list-style-type: none"> 1. Practices like pair programming and collaborative reviews are infeasible for a single developer. 2. Constant customer interaction is not practical without a live client or end-user during development.

Kanban	Focuses on visualizing tasks and limiting work in progress to improve flow.	<ol style="list-style-type: none"> 1. Designed for teams to manage simultaneous tasks, which is unnecessary in a single-developer setup. 2. Visual task boards offer limited benefit when task tracking is managed directly by the developer.
--------	---	---

Table 58:Reason for not choosing Considered Methodologies

7.9 Appendix I: Research Work

7.9.1 Research 1

The research purpose will focus on the prototype development of a mobile phone application using a mockup tool, Balsamiq, in developing pet care management to enhance monitoring in regard to pets' health and care. This research employed the Rational Unified Process to realize high-quality software development with good feedback during expert evaluation of the application. The application seeks to facilitate the solving of the problem of pet care, normally ignored because of challenges arising in regard to social and health issues. The project is a good opportunity for further working continuously on the application to create more features so pet owners can take better care of their pets (López, et al., 2023).

Original Article

Design of a Mobile Application for the Control of Pet Care

Diana Conde López¹, Lesdi Gavilán Colca², Laberiano Andrade-Arenas³, Michael Cabanillas-Carbonell⁴

^{1,2}Facultad de Ingeniería y Negocios, Universidad Privada Norbert Wiener, Lima, Perú.

³Facultad de Ingeniería, Universidad Tecnológica del Perú, Lima, Perú

⁴Facultad de Ingeniería, Universidad Privada del Norte, Lima, Perú

Corresponding Author: mcabanillas@ieee.org

Received: 07 October 2022

Revised: 16 March 2023

Accepted: 22 March 2023

Published: 25 March 2023

Abstract - Activities they perform. A problem that apparently is not very important arises or becomes more evident. Talking about the carelessness that we often, without being intentional, have with our pets, the circumstances force us mostly to have tragic endings with them. Therefore, with the advancement of technology, we have the opportunity to greatly reduce this end and thus give utility to the current resources being presented. The objective of the research work is to develop a prototype of a mobile application using the Balsamiq mockup tool and take as a reference the care model of experts in animal care. In this way, it is possible to have better control and monitoring of the health and other care of our pets. For this use, the Rational Unified Process (RUP) methodology since it is the most appropriate for producing high-quality software, giving us a complete vision in the development of each phase that this methodology presents. Likewise, a positive result is obtained by the expert judgments of the evaluation of the design of our application. As a result, the proposed objective was achieved by obtaining a prototype that can help meet some needs of people who require support in the care of their pets, thus leaving an open the door to continue adding and improving the development of this project.

Keywords - Balsamiq, Control, Mobile application, Methodology RUP, Pet care.

1. Introduction

Animal welfare is an issue emphasized in society worldwide, both in the natural and domestic environment [1]. The care of domestic animals is of great relevance. Since by enjoying the good care that should be provided to them, humans will also benefit from it [2]. At present, the issue of animal care has been a topic of secondary or low interest. Due to the situation Peru and the world are going through regarding human health, there has been a deficit in our faithful companions' care. As evidenced in parks, streets and other places crowded and inhabited by people in every place. Where animals are neglected and, in the worst cases, abandoned. That is why it is necessary to consider the respective care and attention of our pets, as pointed out by the author [3].

Thus, thanks to the progress that technology has been having day by day, it has become an ally in the development of human tasks. Therefore, technology becomes an auxiliary tool to facilitate our relationship in the care of our children, as mentioned by the author [4]. One can see that we have tools that contribute to the care one should have with our pets. A very clear example of how to apply them is mentioned by [5].

Showing us how one can organize and use the data needed for better care. An analysis was made of people's need for the support of applications that facilitate medical treatments, making use of our smartphones, mentioned by the author [6], demonstrating that the care of our faithful friends cannot be separated from the use and innovation of technology. In human health care, a great contribution is being developed with mobile tools for the control and care each person and situation deserves, as mentioned by the author [7]. So, one can mention that in the same way, mobile applications are also used for animal care. Having seen all of the above and the very fact that people with busy schedules, work, study and other activities also want to enjoy the company of their pets. Moreover, seeing the importance of animal care, the present work developed is of interest [8] since it contributes to the mutual need between man and his pet, which is the welfare and tranquillity of the period of the company between them.

The objective of the research work is to develop a prototype mobile application using the Balsamiq mockup tool and take as a reference the care model of experts in animal care. In this way, we can better control and monitor our pets' health and other care. Likewise, use was made of the RUP methodology.

Figure 620: Screenshot of Research Paper 1

7.9.2 Research 2

This paper is on the design of an e-platform for pet care that will help manage pets' health, appointment records, diet, exercise, and medication. This paper describes how sensors, wearable devices, AI, and machine learning are applied to personalized pet care and the early detection of health problems of pets. The present paper is going to illustrate how the designed e-platform enables good pet care by organizing relevant information about pets in one place and allowing proactive management that contributes to better health outcomes (Adithya, et al., 2023).

Platform for Pet Care And Maintenance

Dr.S.K.Manju Bhargavi¹,Aishwarya A B ²,Christy Theresa Regi ³,
 Govid P.Shangbhag ⁴,Indumathi M ⁵, Karukuri Silpa Kala ,Kofi Immanuel
 Jones⁷,Suman Besra ⁸ Palakati Adithya⁹

¹(Associate Professor/ JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

²(School of CS & IT, JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

³(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

⁴(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

⁵(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

⁶(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

⁷(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

⁸(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

⁹(School of CS & IT,JAIN(DEEMED-TO-BE)UNIVERSITY, INDIA)

Abstract: A platform for pet care and maintenance is an online system designed to assist pet owners in managing their pets' daily needs. The platform allows pet owners to track their pets' health, schedule appointments with veterinarians, and manage their pets' diets, exercise routines, and medications. The platform may also provide access to online communities where pet owners can connect with other pet owners to share advice and support. The platform may use a variety of technologies, such as sensors and wearable , to collect data on pets' behavior, health, and activity levels, and use this information to provide personalized recommendations for pet care. Artificial intelligence and machine learning algorithms may be used to analyze this data and make predictions about potential health problems or changes in behavior. By providing a centralized location for pet-related information and resources, a platform for pet care and maintenance can help pet owners stay organized and on top of their pets' needs. It can also promote the health and well-being of pets by enabling proactive care and early detection of health problems. Overall, a platform for pet care and maintenance can be an invaluable tool for pet owners who want to provide the best possible care for their pets.

Keywords: WSPA, VPH, RDBMS, Voslárvá et al., p),XAMPP,DOM

Date of Submission: 01-04-2023

Date of Acceptance: 13-04-2023

I. INTRODUCTION

Pet ownership is a significant responsibility that requires pet owners to provide care and attention to their pets' daily needs. However, managing pets' health, diet, and exercise routines can be challenging, especially for busy pet owners. A platform for pet care and maintenance can help pet owners stay organized and on top of their pets' needs by providing a centralized location for pet-related information and resources. In this paper, we describe the features and benefits of a platform for pet care and maintenance.

This project is developed in form web Application, which is an attempt to provide the advantages of online shopping to customers of a real shop. The objective of the project is to make an application in online platform to purchase items in an existing shop. In order to build such an online complete web support need to be provided. A complete and efficient web application which can provide the online shopping experience is the basic objective of the project. The web application can be implemented in the form of an online application with web view. Pet Shop Management System is to provide services 24x7. Customer has support to view the shop location, can order and select there pet delivery or pick up. The shop keepers have services to view order and manage the order cancellation or delivery and see the data.

Figure 621: Screenshot of Research Paper 2

7.9.3 Research 3

The paper introduces a novel PIMS, known as PetBook, which centralizes health records, vaccination plans, feeding needs, and all other important care information needed for better health and care of the pets. The research has observed that one-size-fits-all in regard to pet care lacks, and has shown the potential of the system in providing personalized care management for both the domestic and the exotic pets. The study demonstrates, through extensive testing, that this system prevents any kind of omission in cares and enhances access to critical information by either the pet owner or the veterinarian. (Mortale, et al., 2024).



e-ISSN: 2582-5208

International Research Journal of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

Volume:06/Issue:11/November-2024

Impact Factor- 8.187

www.irjmets.com

PETBOOK: A PLATFORM FOR PET OWNERS FOR VARIOUS FEATURES ABOUT PETS

**Sonali Mortale^{*1}, Sakshi Gurme^{*2}, Niraj Naphade^{*3}, Sarvesh Sonmale^{*4},
Janhavi Umbre^{*5}**

^{*1,2,3,4,5}Dept. Of Information Technology, Pimpri Chinchwad Polytechnic, Pune, Maharashtra, India.

DOI : <https://www.doi.org/10.56726/IRJMETS63759>

ABSTRACT

PetBook a platform for pet owners for various features about pets; Importance of managing pet information Pets, like humans, require regular medical care and attention to maintain their well-being. However, pet owners may find it difficult to keep track of all the necessary appointments, vaccinations, medications, and nutritional needs. Moreover, specific information about the pet's breed, behaviour, and preferences is often scattered or forgotten. A PIMS helps consolidate all relevant data, ensuring that pet owners and caregivers have easy access to vital information. This improves the pet's quality of life and reduces the risk of missed vaccinations, overlooked medical issues, or inappropriate feeding practices Need of pet care and information cause Every pet is unique, with its own set of health requirements, dietary restrictions, and activity levels. A one-size-fits-all approach to pet care is insufficient for maintaining optimal health.

Keywords: Pet Care, Information Management, Health Tracking, Veterinary Support, Pet Wellness, Digital Solutions, Pet Technology.

I. INTRODUCTION

This research paper presents an advanced Pet Information Management System (PIMS) designed to improve pet care and management across various settings. The study explores the technical and practical aspects of the system, focusing on its potential to reduce the challenges of pet care while maximizing data accessibility for pet owners, veterinarians, and pet service providers. Through a series of tests and analyses, we evaluate the PIMS's performance under different usage scenarios and types of pets. The results indicate that the system significantly enhances pet owners' ability to manage pet health and wellness, reducing the likelihood of missed care routines compared to traditional methods. Additionally, we analyze the broader applications of this technology for both domestic and exotic pets, highlighting its flexibility and potential to integrate with existing veterinary and pet care frameworks. This paper aims to contribute to the development of more organized and user-friendly pet management solutions, providing insights for future research and practical applications in the field of pet care technology.

II. METHODOLOGY

Procedure that we adopted during the completion of the project is as follows :-

1. Literature Review:

A comprehensive review of existing literature on Pet Information Management Systems (PIMS) and similar animal care management solutions was conducted. This included academic papers, technical reports, and case studies to identify current technologies, design principles, and user experiences. The review aimed to establish a theoretical foundation for the research and highlight gaps in existing knowledge regarding pet health tracking, personalized care, and data management in the pet care sector. Design Analysis:

Figure 622: Screenshot of Research Paper 3

7.9.4 Research 4

The study assessed the satisfaction of patients using the "Mawid" Web-Based Medical Appointment System implemented at Primary Health Care Centers in Southwest Saudi Arabia and yielded a response rate of 94.3%. It helped in managing patient flow, as 89.1% expressed satisfaction, and reduced overcrowding, where 87.7% were satisfied. The satisfaction level among male patients was much higher than among females. These can be taken into consideration while designing and implementing such appointment systems, which will help to improve user experiences (Mahfouz, et al., 2023).

Review began 01/08/2023
 Review ended 01/13/2023
 Published 01/21/2023

© Copyright 2023
 Mahfouz et al. This is an open access article distributed under the terms of the Creative Commons Attribution License CC-BY 4.0., which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Evaluation of Patient Satisfaction With the New Web-Based Medical Appointment Systems "Mawid" at Primary Health Care Level in Southwest Saudi Arabia: A Cross-Sectional Study

Mohammed Salih Mahfouz ¹, Majed A. Ryani ¹, Abdulrahem A. Shubair ², Saud Y. Somill ², Ali A. Majrashi ², Hussam Abdu Zalah ², Adel Ali Khubrani ², Mohammed I. Dabsh ², Abdullatif M. Maashi ²

¹. Department of Family and Community Medicine, Jazan University, Jazan, SAU ². Faculty of Medicine, Jazan University, Jazan, SAU

Corresponding author: Mohammed Salih Mahfouz, mm.mahfouz@gmail.com

Abstract

Background

Patient satisfaction has become an influential corner in the health services process. Web-based appointment scheduling has been expanded for its benefits and has become a popular research topic. This study's objectives were to assess patients' satisfaction and perception with the new Web-Based Medical Appointment System "Mawid" program and determine the associated factors at the Primary Health Care Centers level in Jazan Southwest Saudi Arabia.

Methods

An observational cross-sectional survey was implemented among 424 adults aged 18 years and above, attending a randomly selected 12 primary health care centers in the Jizan region, Southwest Saudi Arabia. The study instrument included socio-demographic background information, perception, and level of satisfaction with the new appointment system. Responses were analyzed using the SPSS program by applying descriptive and inferential statistical techniques.

Results

The overall level of satisfaction was very high at 94.3% with 95% C.I. (91.7-96.1). A large proportion of study participants were highly satisfied with the new Web-Based Medical appointment System "Mawid" as nine satisfaction items scored a level of satisfaction of 90% and above. Regarding the perception, 89.1% of the participants agreed that the appointment booking system regulates the number of patients, while 87.7% of participants considered that the appointment system reduces clinic crowding. More than half of respondents (61.8%) agreed that the community culture might limit the scheduling system's use. Univariate and multivariate logistic regression analysis suggested that male patients were more likely to have a higher level of satisfaction as compared with female (COR= 2.95, 95% C.I.:1.15-7.60, $p = 0.025$) and (AOR= 3.12, 95% C.I:1.14-8.52, $p = 0.026$), respectively.

Conclusions

In conclusion, this study revealed a high level of satisfaction among study the participants with the new Web-Based Medical Appointment System "Mawid." The system effectively improved patients' satisfaction with registration and reduced waiting times. Patients' satisfaction can be assessed regularly and used systematically as a quality and benchmarking instrument in primary health care.

Categories: Public Health, Healthcare Technology, Other

Keywords: "mawid", jazan region, patients satisfaction, appointment system, web-based

Introduction

Patient satisfaction has become an influential corner in health system planning and development. It is a

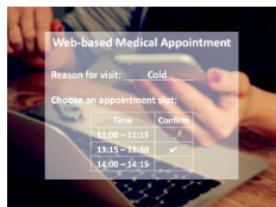
Figure 623: Screenshot of Research Paper 4

7.9.5 Research 5

Health care is moving toward patient-centered systems, and Web-based medical appointment scheduling means better access, fewer no-show patients, and greater patient satisfaction. This review of 36 studies did show that there are, therefore, despite benefits, adoption characteristics that are mostly influenced by specific key barriers—cost and security. The willingness of patients to use these systems is dependent on digital literacy and their communications habits. Overall, this will likely lead to increasing adoption trends with room for further improvement upon future research (Zhao, et al., 2017).

The screenshot shows the header of the Journal of Medical Internet Research website. It includes the journal logo, the title "Journal of Medical Internet Research", a search bar, and navigation links for "Journal Information", "Browse Journal", and "Submit".

Published on 26.04.2017 in Vol 19, No 4 (2017): April



Web-Based Medical Appointment Systems: A Systematic Review

Peng Zhao¹; Illhoi Yoo^{1,2}; Jaie Lavoie³; Beau James Lavoie⁴; Eduardo Simoes^{1,2}

Article	Authors	Cited by (101)	Tweetations (16)	Metrics
Abstract				
<ul style="list-style-type: none"> • Abstract • Introduction • Methods • Results • Discussion • References • Abbreviations • Copyright 	<p>Background: Health care is changing with a new emphasis on patient-centeredness. Fundamental to this transformation is the increasing recognition of patients' role in health care delivery and design. Medical appointment scheduling, as the starting point of most non-urgent health care services, is undergoing major developments to support active involvement of patients. By using the Internet as a medium, patients are given more freedom in decision making about their preferences for the appointments and have improved access.</p> <p>Objective: The purpose of this study was to identify the benefits and barriers to implement Web-based medical scheduling discussed in the literature as well as the unmet needs under the current health care environment.</p>			

Figure 624: Screenshot of Research Paper 5

7.10 Appendix J: System Screenshot

1. Home page

The screenshot displays the PetVet application's home page. On the left, a sidebar titled "MENU" includes links for "Home", "Inbox", "MY PETS" (with "Charlie (dog)" selected), "Add Pet", "APPOINTMENTS" (with "Find Vets" and "My Appointments"), and "QUICK LINKS" (with "Contact Us" and "About Us"). The main content area features a search bar at the top right. Below it is a "Sort Posts" section with options: "Revised", "Oldest", "Most Liked", and "Most Comments". A central post titled "Protect Your Pet: The Power of Vaccination" by "u/AdminRebot" shows a golden retriever and discusses the importance of vaccination against diseases like rabies, parvo, and distemper. To the right, a "Categories" sidebar lists "Emergency", "Grooming", and "Vaccination". A "Trending Posts" sidebar shows five recent articles: "How to Keep Your Cat's Coat Smooth and Tangle-Free", "5 Essential Grooming Tips for a Healthy...", "My Dog Swallowed a Battery - What Should I Do?", "Protect Your Pet: The Power of Vaccination...", and "Core Vaccines Every Dog Needs". The bottom of the page shows other posts from "Vaccination" and "Grooming" categories.

2. Inbox page

The screenshot shows the PetVet mobile application's inbox screen. At the top, there is a search bar labeled "Search for pets, vets, or posts...". On the left, a sidebar menu includes "Home", "Inbox", "MY PETS" (with "Charlie (dog)" listed), "Add Pet", "APPOINTMENTS" (with "Find Vets" and "My Appointments" listed), and "QUICK LINKS" (with "Contact Us" and "About Us" listed). The main area is titled "Messages" and shows a list of conversations. One conversation is expanded, showing messages between the user "Datta18" and "Hey Rebof". The messages are:

- Datta18: "hey" (06:37 PM)
- Hey Rebof: "Hello, Today was a great session for my dog." (06:32 PM)
- Hey Rebof: "hey" (06:37 PM)

At the bottom of the message list, there is a text input field with "Type your message..." and a send button with a paper airplane icon.

3. Find Vets page

The screenshot shows the PetVet mobile application interface. On the left is a vertical sidebar with a navigation menu:

- MENU**
- Home
- Inbox
- MY PETS**
- Charlie (dog)
- Add Pet
- APPOINTMENTS**
- Find Vets
- My Appointments
- QUICK LINKS**
- Contact Us
- About Us

The main content area is titled "Find Veterinarians" and features a search bar at the top. Below the search bar are filters for "Sort by:" (set to "All Vets") and "Location:" (empty). The page lists four veterinarians in a grid:

- Dr. Kumar Katwal** (Verified) - Located in Hetauda, Nepal. Specializes in Big dogs and surgery. 0.0 (0 reviews). Includes a "Book Appointment" button.
- Dr. Shubham Datta Mishra** (Verified) - Located in Lalitpur, Nepal. Specializes in Small Animal Care & Surgery. 5.0 (1 reviews). Includes a "Book Appointment" button.
- Dr. Test Name Vet** (Verified) - Located in Kathmandu, Nepal. Specializes in Exotic Birds. 0.0 (0 reviews). Includes a "Book Appointment" button.
- Dr. Avishhek Gautam** (Verified) - Located in Lalitpur, Nepal. Specializes in None. 0.0 (0 reviews). Includes a "Book Appointment" button.

Below the grid, there are two descriptive sections: "Animal Lover" and "Passionate about pet health & well-being".

4. My appointment lists page

The screenshot shows the 'My Appointments' section of the PetVet mobile application. On the left is a sidebar with a navigation menu:

- MENU**
 - Home
 - Inbox
- MY PETS**
 - Charlie (dog)
 - Add Pet
- APPOINTMENTS**
 - Find Vets
 - My Appointments** (highlighted with a pink background)
- QUICK LINKS**
 - Contact Us
 - About Us

The main content area is titled '**My Appointments**' and displays the following information:

View and manage your upcoming veterinary appointments

Filter by Status: All Statuses

Appointment with Dr. Kumar Katwal

- Charlie** (dog • Labrador)
- Reason for Visit:** Weekly Checkup
- Date:** Monday, April 26, 2025 | **Time:** 3:15 PM - 5:15 PM | **Clinic:** Katwal Pet Clinic
- Status:** Confirmed | Paid

View Details

Appointment with Dr. Shubham Datta Mishraa

- Charlie** (dog • Labrador)
- Reason for Visit:** Fever.
- Date:** Monday, April 25, 2025 | **Time:** 5:30 AM - 6:30 AM | **Clinic:** Mishra Pet Clinic
- Status:** Completed | Paid

View Details

Appointment with Dr. Shubham Datta Mishraa

- Charlie** (dog • Labrador)
- Reason for Visit:** Weekly checkup
- Date:** Tuesday, April 26, 2025 | **Time:** 5:30 PM - 6:30 PM | **Clinic:** Mishra Pet Clinic
- Status:** Rejected | Paid

View Details

Appointment with Dr. Shubham Datta Mishraa

- Charlie** (dog • Labrador)
- Reason for Visit:** Monthly Checkup
- Date:** Thursday, April 25, 2025 | **Time:** 5:30 PM - 6:30 PM | **Clinic:** Mishra Pet Clinic
- Status:** Completed | Paid

View Details

5. Appointment details page

The screenshot shows the PetVet mobile application interface. On the left is a sidebar with a navigation menu:

- MENU**
 - Home
 - Inbox
- MY PETS**
 - Charlie (dog)
 - Add Pet
- APPOINTMENTS**
 - Find Vets
 - My Appointments
- QUICK LINKS**
 - Contact Us
 - About Us

The main content area is titled "Appointment Details" and displays the following information:

- Appointment with Dr. Shubham Datta Mishraa**
- Date: Thursday, April 25, 2025
- Time: 5:30 PM - 6:30 PM
- Clinic: Mishra Pet Clinic
- Pet Owner:** Rebof Katwal 22
- Veterinarian:** Dr. Shubham Datta Mishraa
- Pet:** Charlie
- Reason:** Monthly Checkup
- Current Status:** Completed, Paid

A red button at the bottom left says "← Back to Appointments".

6. Vet schedule page

The screenshot shows the PetVet mobile application interface. On the left is a sidebar with a navigation menu:

- MENU**
 - Home
 - Inbox
- MY PETS**
 - Charlie (dog)
 - Add Pet
- APPOINTMENTS**
 - Find Vets
 - My Appointments
- QUICK LINKS**
 - Contact Us
 - About Us

The main content area is titled "Dr. Kumar Katwal's Schedule" and displays the following information:

- Dr. Kumar Katwal**
- Clinic: Katwal Pet Clinic
- Schedule:** Monday
- Available slot: 3:15 PM - 5:15 PM (Booked)

A red button at the bottom right says "← Back to Vets".

7. Book appointment page

The screenshot shows the 'Book an Appointment' page of the PetVet app. At the top, there is a search bar with placeholder text 'Search for pets, vets, or posts...'. On the right side of the header is a user profile icon.

Left sidebar (MENU):

- Home
- Inbox
- MY PETS**
 - Charlie (dog)
 - Add Pet
- APPOINTMENTS**
 - Find Vets
 - My Appointments
- QUICK LINKS**
 - Contact Us
 - About Us

Center Content:

Book an Appointment

Appointment Details:

- Day:** Monday
- Time:** 5:30 a.m. to 6:30 a.m.
- Fee:** NPR 1,000

Select Your Pet: A dropdown menu with the placeholder text '-- Choose a pet --'.

Reason for Visit: A text input field with the placeholder text 'Please describe your pet's symptoms or reason for the appointment...'.

Select Payment Method:

- Pay with Store Credit:** A green button with the text 'Balance: NPR 6000.00'.
- Pay with Khalti:** A purple button.

Bottom Navigation:

[← Back to Schedule](#)

8. Post details page

The screenshot displays the PetVet mobile application interface. On the left is a vertical navigation menu with sections like Home, Inbox, My Pets (listing Charlie, a dog), Add Pet, Appointments (Find Vets, My Appointments), and Quick Links (Contact Us, About Us). The main content area shows a post titled "Protect Your Pet: The Power of Vaccination" by "adminRebof" posted 2 days, 12 hours ago. The post includes a photo of a golden retriever. Below the photo is a text block about the importance of vaccinations. There are like and comment counts (0 each) and edit/delete buttons. A comment input field is present, and a section for comments shows a placeholder message. To the right, there's a sidebar with categories (Emergency, Grooming, Vaccination) and trending posts.

Vaccination

Posted by [adminRebof](#) 2 days, 12 hours ago

Protect Your Pet: The Power of Vaccination 🐶

Vaccinations protect your furry friends from serious diseases like rabies, parvo, and distemper. Not only do they boost your pet's immunity, but they also prevent the spread of infections to other animals and even humans. Keep your pet's shots up-to-date and always follow your vet's vaccination schedule. A little poke today can save a lot of trouble tomorrow!

0 likes 0 comments

What are your thoughts?

Comment

0 Comments

No comments yet. Be the first to share what you think!

Categories

- # Emergency 1
- # Grooming 2
- # Vaccination 2

Trending Posts

- Protect Your Pet: The Power of Vaccination... 0 likes · 0 comments
- Core Vaccines Every Dog Needs 2 likes · 1 comments

9. Pending booking page

Pending Appointment Requests

Review and manage incoming appointment requests

Dr. Shubham Datta Mishraa
Mishra Pet Clinic

No Pending Appointments

You currently don't have any pending appointment requests.

Pending Appointment Requests

Review and manage incoming appointment requests

Dr. Shubham Datta Mishraa
Mishra Pet Clinic

Rebof Katwal 22
Charlie (dog)

Pending

Reason: Checkup
Requested Time: 5:30 AM - 6:30 AM
Date: Monday
Breed: Labrador
Age: 10 years
Weight: 30.00 kg
Color: White

Accept Reject

10. Accepted booking page

Accepted Appointments

Manage your upcoming veterinary appointments

Dr. Shubham Datta Mishraa
Mishra Pet Clinic

No Accepted Appointments

You currently don't have any accepted appointments.

Accepted Appointments

Manage your upcoming veterinary appointments

Dr. Shubham Datta Mishraa
Mishra Pet Clinic

Rebof Katwal 22

Confirmed

Reason: Checkup
Scheduled Time: 5:30 AM - 6:30 AM
Date: Monday, April 25, 2025
Breed: Labrador
Age: 10 years
Weight: 30.00 kg
Color: White

Mark as Completed

11. Create schedule page

The screenshot shows the PetVet mobile application interface. At the top, there is a navigation bar with a search bar labeled "Search for pets, vets, or posts...". On the left, there is a sidebar titled "MENU" containing links for "Home", "Inbox", "Pending", "Accepted", "My Schedule", and "Add Schedule" (which is highlighted with a pink background). Below the menu is a section titled "APPOINTMENTS" with links for "Day of the Week", "Start Time", and "End Time". The main content area is titled "Add New Schedule" and has a sub-instruction "Create new available time slots for appointments". It displays a profile picture of "Dr. Shubham Datta Mishraa" from "Mishra Pet Clinic". Below this, there are three input fields: "Day of the Week" (set to Monday), "Start Time" (set to --:-- --), and "End Time" (set to --:-- --). At the bottom are two buttons: "Save Schedule" (in red) and "Cancel".

12. Edit schedule page

The screenshot shows the PetVet application interface. At the top left is the PetVet logo. A search bar at the top right contains the placeholder text "Search for pets, vets, or posts...". On the far right is a user profile icon.

The main content area is titled "Edit Schedule" and includes the sub-instruction "Update your available time slot for appointments". Below this, there is a profile card for "Dr. Shubham Datta Mishraa" from "Mishra Pet Clinic", featuring a small profile picture of a person.

The left sidebar contains a "MENU" section with links to "Home" and "Inbox". Under "APPOINTMENTS", there are links for "Pending", "Accepted", and "My Schedule". Under "QUICK LINKS", there are links for "Contact Us" and "About Us".

The central "Edit Schedule" form has three input fields: "Day of the Week:" set to "Monday", "Start Time:" set to "05:30 AM", and "End Time:" set to "06:30 AM". Below these fields are three buttons: a teal "Update Schedule" button, a white "Cancel" button, and a red "Delete Schedule" button.

666

222067711 Rebof Katwal

13. Edit post page

Edit Post

Title
Protect Your Pet: The Power of Vaccination 🐶🐶

Category
Vaccination

Content

Vaccinations protect your fury friends from serious diseases like rabies, parvo, and distemper. Not only do they boost your pet's immunity, but they also prevent the spread of infections to other animals and even humans. Keep your pet's shots up-to-date and always follow your vet's vaccination schedule. A little poke today

Current Media

Add New Media (Only one photo or video allowed)

Add Image **Add Video**

Save Changes

14. Review page

The screenshot shows the PetVet mobile application interface. At the top, there is a navigation bar with a search bar containing the placeholder "Search for pets, vets, or posts...". On the far right of the bar is a user profile icon. Below the navigation bar, the main content area has a light green background. On the left side, there is a vertical sidebar with the following sections and items:

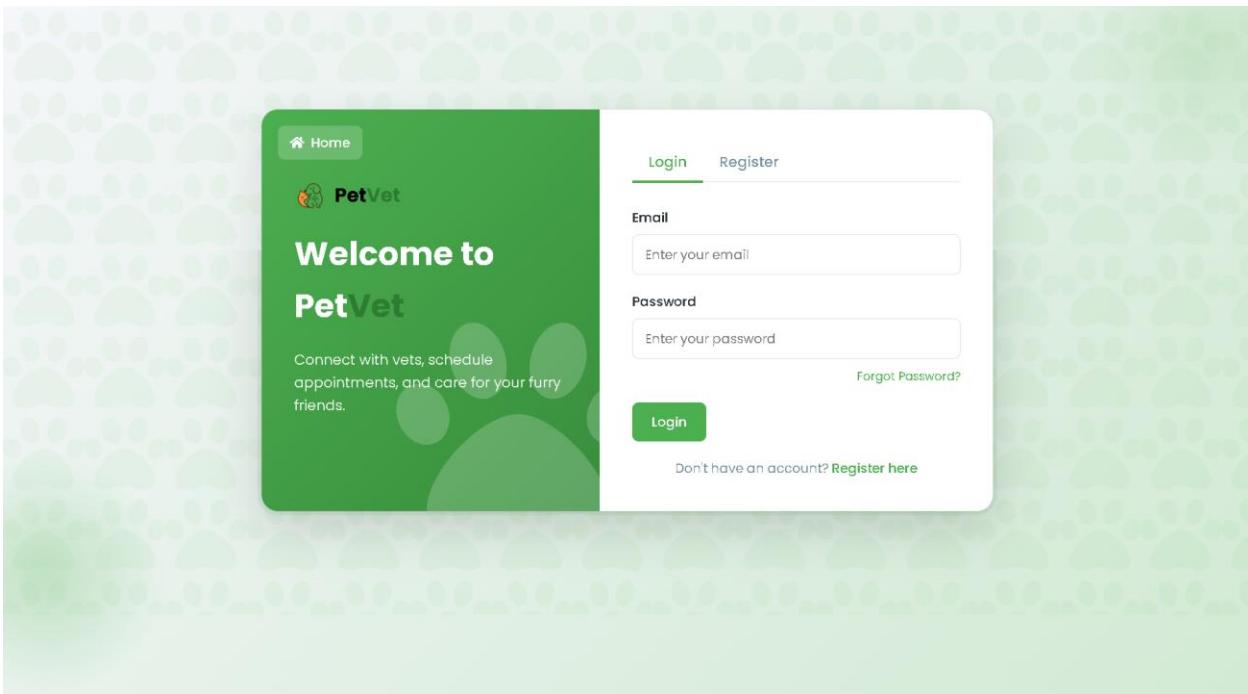
- MENU**
 - [Home](#)
 - [Inbox](#)
- MY PETS**
 - [Charlie \(dog\)](#)
 - [Add Pet](#)
- APPOINTMENTS**
 - [Find Vets](#)
 - [My Appointments](#)
- QUICK LINKS**
 - [Contact Us](#)
 - [About Us](#)

The main content area on the right is titled "★ Review Your Veterinarian" and features a heading "Share your experience with Dr. Shubham Datta Mishraa". It includes a profile picture of a veterinarian, the name "Dr. Shubham Datta Mishraa", the specialization "Small Animal Care & Surgery", and the location "Mishra Pet Clinic". Below this, there is a section for "Your Rating" with five yellow stars, followed by a "Your Review" section with a placeholder "Share your experience with this veterinarian...". At the bottom of this section are two buttons: a white "← Back" button and a red "Submit Review" button.

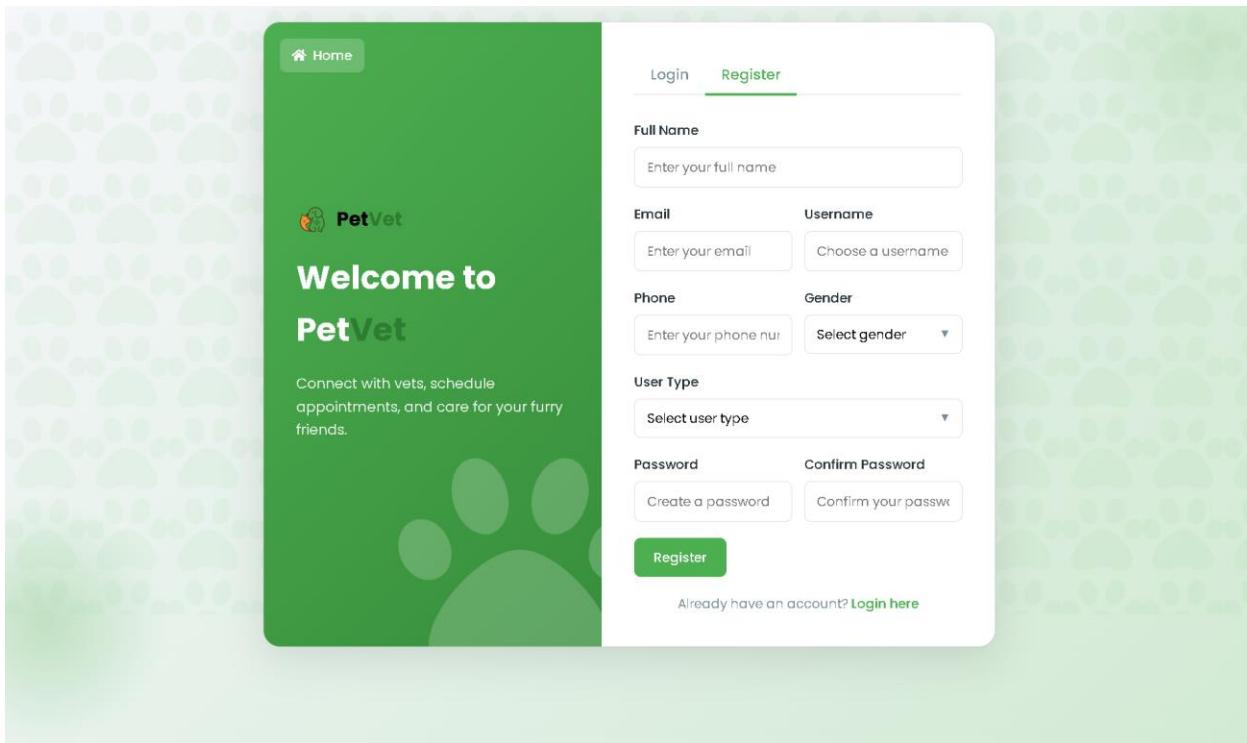
668

222067711 Rebof Katwal

15. Login page



16. Register page



17. Admin dashboard

PetVet Admin

Dashboard

adminRebof Administrator

- Dashboard**
- User Management
- Post Management
- Vet Approvals
- Categories

Total Users **11** Registered users on the platform

Veterinarians **4** Registered veterinarians

Pet Owners **7** Registered pet owners

Total Posts **5** Posts created by users

User Statistics

User Type	Count
Veterinarians	4
Pet Owners	7

Posts by Category

Category	Count
Grooming	2
Vaccination	2
Emergency	1

Recent Posts

[View All](#)

Title	Author	Category	Date	Actions
Protect Your Pet: The Power of Vaccination 🐶💉	adminRebof	Vaccination	Apr 27, 2025	View
Core Vaccines Every Dog Needs	Datta18	Vaccination	Apr 25, 2025	View
How to Keep Your Cat's Coat Smooth and Tangle-Free	Asthaa	Grooming	Apr 25, 2025	View
5 Essential Grooming Tips for a Healthy, Happy Dog	Reboffff	Grooming	Apr 25, 2025	View
My Dog Swallowed a Battery – What Should I Do?	Reboffff	Emergency	Apr 25, 2025	View

Pending Vet Approvals

[View All](#)

Name	Email	Clinic Name	Requested At	Actions
No pending vet approvals.				

[Logout](#)

18. Admin vet approval

PetVet Admin **Vet Approvals**

adminRebof Administrator

Pending Approvals: 0 Vets waiting for approval

Recently Approved: 4 Vets approved in the last 7 days

Recently Declined: 0 Vets declined in the last 7 days

Pending Vet Approvals

Name	Email	Clinic Name	License Number	Experience	Requested At	Actions
No pending vet approvals.						

Recently Approved Vets

Name	Email	Clinic Name	License Number	Approved At
Test Name Vet	testvet@gmail.com	Test clinic	NVC-NP-2024-89897	Apr 25, 2025
Avishek Gautam	avishekgautam@gmail.com	Where Pets Come First Clinic	PAW-123999	Apr 25, 2025
Kumar Katwal	kumarkatwal1964@gmail.com	Katwal Pet Clinic	PAW-777777	Apr 26, 2025
Shubham Datta Mishraa	shubhammishra@gmail.com	Mishra Pet Clinic	PAW-1234556	Apr 25, 2025

Recently Declined Vets

Name	Email	Clinic Name	License Number	Declined At
No recently declined vets.				

[Logout](#)

19. Admin post management

PetVet Admin **Post Management**

adminRebof Administrator

Search Posts

Search by title or content

Category

All Categories

Search

Title **Author** **Category** **Date** **Likes** **Status** **Actions**

Title	Author	Category	Date	Likes	Status	Actions
Protect Your Pet: The Power of Vaccination 🐶💉	adminRebof	Vaccination	Apr 27, 2025	0	Active	View Delete
Core Vaccines Every Dog Needs	Datta18	Vaccination	Apr 25, 2025	2	Active	View Delete
How to Keep Your Cat's Coat Smooth and Tangle-Free	Asthaa	Grooming	Apr 25, 2025	2	Active	View Delete
5 Essential Grooming Tips for a Healthy, Happy Dog	Reboffff	Grooming	Apr 25, 2025	5	Active	View Delete
My Dog Swallowed a Battery – What Should I Do?	Reboffff	Emergency	Apr 25, 2025	0	Active	View Delete

[Logout](#)

20. Admin user detail page

The screenshot shows the 'User Management' section of the PetVet Admin interface. On the left, a sidebar menu includes 'Dashboard', 'User Management' (which is active and highlighted in pink), 'Post Management', 'Vet Approvals', and 'Categories'. The main content area has a header 'User Management' and a search bar with placeholder 'Search by name, email, or username'. A dropdown for 'User Type' is set to 'All Users', with a 'Search' button next to it. Below is a table with the following data:

Name	Username	Email	User Type	Joined	Status	Actions
Kumar Katwal	Kumar Katwal	kumarkatwal1964@gmail.com	Veterinarian	Apr 26, 2025	Verified	<button>View</button>
Shubham Datta	Shubhamm	Shuvhamdatta@gmail.com	Pet Owner	Apr 26, 2025	Verified	<button>View</button>
Test Name Vet	Dr Vet	testvet@gmail.com	Veterinarian	Apr 25, 2025	Verified	<button>View</button>
Test Name	testusername	testuser@gmail.com	Pet Owner	Apr 25, 2025	Verified	<button>View</button>
Kalia Padilla	jodidamu	dowero@mailinator.com	Pet Owner	Apr 25, 2025	Verified	<button>View</button>
Tejas Singh Bhandari	teju	tejas@gmail.com	Pet Owner	Apr 25, 2025	Verified	<button>View</button>
Avishek Gautam	Avi	avishekautam@gmail.com	Veterinarian	Apr 25, 2025	Verified	<button>View</button>
Shubham Datta Mishra	Datta18	shubhammishra@gmail.com	Veterinarian	Apr 25, 2025	Verified	<button>View</button>
Astha Thapa	Asthaa	asthathapa311@gmail.com	Pet Owner	Apr 25, 2025	Verified	<button>View</button>
Rebof Katwal 22	adminRebof	rebofkatwal7@gmail.com	Pet Owner	Apr 25, 2025	Verified	<button>View</button>

At the bottom left of the main content area, there are navigation icons: a left arrow, a right arrow, and a double-right arrow. At the very bottom left, there is a 'Logout' link.

21. Admin category management

The screenshot shows the PetVet Admin interface for managing categories. On the left, there's a sidebar with navigation links: Dashboard, User Management, Post Management, Vet Approvals, and Categories (which is highlighted with a pink background). The main area is titled "Category Management". It has a "Categories" table listing three categories: Emergency, Grooming, and Vaccination. Each row includes columns for Name, Slug, Description, Post Count, and Actions (Edit and Delete buttons). To the left of the table is a form for "Add New Category" with fields for Category Name and Description, and a "Add Category" button.

Name	Slug	Description	Post Count	Actions
Emergency	emergency-fo	Urgent medical attention	1	<button>Edit</button> <button>Delete</button>
Grooming	grooming-hc	Bathing, trimming, and styling	2	<button>Edit</button> <button>Delete</button>
Vaccination	vaccination-xs	Pet immunization and booster shots	2	<button>Edit</button> <button>Delete</button>

674

222067711 Rebof Katwal

22. User info page

The screenshot shows the PetVet user info page for a user named Rebof Katwal 22. The page is divided into several sections:

- MENU:** Home, Inbox, Add Pet.
- MY PETS:** Charlie (dog).
- APPOINTMENTS:** Find Vets, My Appointments.
- QUICK LINKS:** Contact Us, About Us.
- User Profile:** Rebof Katwal 22, Pet Owner, rebofkatwal7@gmail.com, None, Edit Profile, **NPR 6000.00**.
- Basic Information:** Full Name: Rebof Katwal 22, Bio: None, Email: rebofkatwal7@gmail.com, Pets Owned: 1, Phone: None, Gender: Male.
- Profile Information:** Bio: None, Pets Owned: 1.
- Address Information:** Country: None, City: None, Address: None.
- Posts:** A large image placeholder for a post titled "Protect Your Pet: The Power of Vaccination". Below it is a snippet: "Vaccinations protect your furry friends from serious diseases like rabies, parvo, and distemper. Not only do they boost your pet's immunity, but they...". It includes a timestamp (Apr 27, 2025), like and comment counts (0 likes, 0 comments), and View/Edit buttons.
- Comments:** No comments available. A "Browse Posts" button is present.
- Footer:** Logout button.

23. Pet Profile

The screenshot displays the PetVet mobile application interface. At the top, there is a navigation bar with a search bar containing the placeholder text "Search for pets, vets, or posts...". On the far right of the top bar is a user profile icon.

The main content area is titled "Charlie's Profile" with a subtitle "View and manage your pet's information". Below the title is a circular profile picture of a white dog named Charlie. To the right of the picture are three colored tags: a green tag labeled "dog", an orange tag labeled "10 years old", and a purple tag labeled "Labrador".

To the left of the profile picture is a vertical sidebar menu:

- MENU**
 - Home
 - Inbox
- MY PETS**
 - Charlie (dog) (highlighted)
 - Add Pet
- APPOINTMENTS**
 - Find Vets
 - My Appointments
- QUICK LINKS**
 - Contact Us
 - About Us

The central content area is divided into several sections:

- Basic Information**: Shows Color: White and Weight: 30.00 kg.
- Health Information**: Shows Vaccination Status: Not specified and Allergies: None known.
- Medical History**: Displays the message "No medical history recorded".

At the bottom right of the central area is a red button labeled "Delete Pet" with a trash can icon.

24. Add Pet profile

The screenshot shows the PetVet mobile application interface. On the left is a vertical navigation menu with sections: MENU (Home, Inbox), MY PETS (Charlie (dog) highlighted), APPOINTMENTS (Find Vets, My Appointments), and QUICK LINKS (Contact Us, About Us). At the top right is a search bar labeled "Search for pets, vets, or posts..." and a user profile icon. The main content area is titled "Charlie's Profile" and displays "View and manage your pet's information". It features a circular profile picture of a dog, with tags indicating it is a "dog", "10 years old", and a "Labrador". A blue "Edit Profile" button is in the top right. Below the profile are three sections: "Basic Information" (Color: White, Weight: 30.00 kg), "Health Information" (Vaccination Status: Not specified, Allergies: None known), and "Medical History" (No medical history recorded). A red "Delete Pet" button is at the bottom right.

7.11 Appendix K: Sample Codes

7.11.1 Blogs

7.11.1.1 HTML

```
<div class="pet-feed-content">

<div class="post-detail-wrapper">
    <div class="message-container">
        {% for message in messages %}
            <div class="message" {% if message.tags %}{% message.tags %}{% endif %}>
                <span class="message-text">{{ message }}</span>
                <button class="message-close" aria-label="Close message">
                    <i class="fas fa-times"></i>
                </button>
            </div>
        {% endfor %}
    </div>
    <div class="post-detail-container">
        <!-- Main Content Column -->
        <div class="main-content">
            <!-- Main Post Card -->
            <div class="post-card glass-card">
                <!-- Like Button -->
                <div class="like-column">
                    <span class="pet-icon">✿✿</span>
                </div>
            <!-- Post Content -->
        </div>
    </div>
</div>
```

```

<div class="post-content">
    <div class="post-header">
        <span class="post-category">{{ post.category.name }}</span>
        <div class="post-meta">
            <a href="{% url 'authUser:user_profile' post.user.id %}"
            class="username-link">
                {% if post.user.petownerprofile %}
                    
                {% elif post.user.vetprofile %}
                    
                {% else %}
                    
                {% endif %}
            </a>
            <span class="post-author">
                Posted by
                <a href="{% url 'authUser:user_profile' post.user.id %}"
                class="username-link">
                    u/{{ post.user.username }}
                </a>
            </span>
            <span class="post-time">{{ post.date|timesince }} ago</span>
        </div>
    </div>
<h1 class="post-title">{{ post.title }}</h1>

```

```

{%
  if post.image %
    <div class="post-media">
      
    </div>
  {% elif post.video %}
    <div class="post-media">
      <video controls class="post-video">
        <source src="{{ post.video.url }}" type="video/mp4">
      </video>
    </div>
  {% endif %}

  <div class="post-body">
    <p>{{ post.body|linebreaksbr }}</p>
  </div>

  <div class="post-actions">
    <button class="action-btn like-btn {% if request.user in post.likes.all %}active{% endif %}" aria-label="Like" data-post-id="{{ post.id }}>
      <i class="fas fa-heart"></i>
      <span class="like-count">{{ post.likes.count }}</span>
    </button>
    <button class="action-btn comment-btn">
      <i class="fas fa-comment"></i>
      <span class="comment-count">{{ post.comments.count }}</span>
    </button>
  {% if request.user == post.user %}

```

```

<div class="post-owner-actions">
    <button class="action-btn edit-btn" onclick="window.location.href='{{% url 'coreFunctions:edit_post' post.slug %}}'">
        <i class="fas fa-edit"></i> Edit
    </button>
    <button class="action-btn delete-btn" onclick="confirmDeletePost({{ post.id }})">
        <i class="fas fa-trash"></i> Delete
    </button>
</div>
{% endif %}
</div>
</div>

<!-- Comment Form -->
<div class="comment-form-container glass-card">
    <form method="POST" action="{{% url 'coreFunctions:add_comment' post.slug %}}>
        {% csrf_token %}
        <div class="form-group">
            <textarea name="comment" placeholder="What are your thoughts?" required></textarea>
            <div class="form-footer">
                <button type="submit" class="btn btn-primary">
                    <i class="fas fa-paper-plane"></i> Comment
                </button>
            </div>
        </div>
    </div>

```

```

        </form>
    </div>

    <!-- Comments Section -->
    <div class="comments-section glass-card" id="comments-section">
        <h3 class="comments-header">
            <i class="fas fa-comments"></i>
            <span class="comment-count">{{ post.comments.count }}</span>
        <Comments>
            </h3>

            {% for comment in post.comments.all %}

                <div class="comment {% if request.user == comment.user %}current-user-comment{% endif %}" id="comment-{{ comment.id }}">
                    <!-- Comment Like Button -->
                    <div class="like-column">
                        <button class="like-btn {% if request.user in comment.likes.all %}active{% endif %}" data-comment-id="{{ comment.id }}" aria-label="Like comment">
                            <i class="fas fa-heart"></i>
                            <span class="like-count">{{ comment.likes.count }}</span>
                        </button>
                    </div>

                    <!-- Comment Content -->
                    <div class="comment-content">
                        <div class="comment-header">
                            <a href="{% url 'authUser:user_profile' comment.user.id %}" class="username-link">
                                {% if comment.user.petownerprofile %}

```

```



{% elif comment.user.vetprofile %}



{% else %}



{% endif %}

</a>

<span class="comment-author">

<a href="% url 'authUser:user_profile' comment.user.id %"
class="username-link">

u/{{ comment.user.username }}

</a>

</span>

<span class="comment-time">{{ comment.date|timesince }} ago</span>

</div>

<div class="comment-body">

<p>{{ comment.comment }}</p>

</div>

<div class="comment-actions">

<button class="action-btn reply-btn" data-comment-id="{{ comment.id }}">

<i class="fas fa-reply"></i> Reply

</button>

<button class="action-btn share-btn">

<i class="fas fa-share"></i> Share

```

```

        </button>

        {% if request.user == comment.user %}

            <button class="action-btn delete-btn"
onclick="confirmDeleteComment('{{ comment.id }}')>

                <i class="fas fa-trash"></i> Delete

            </button>

        {% endif %}

    </div>

<!-- Replies container --&gt;

&lt;div class="replies-container" id="replies-container-{{ comment.id }}&gt;

    &lt;!-- Replies (nested comments) --&gt;

    {% for reply in comment.replies.all %}

        &lt;div class="comment reply {% if request.user == reply.user %}current-
user-comment{% endif %}" id="reply-{{ reply.id }}&gt;

            &lt;div class="comment-content"&gt;

                &lt;div class="comment-header"&gt;

                    &lt;a href="{% url 'authUser:user_profile' reply.user.id %}"
class="username-link"&gt;

                        {% if reply.user.petownerprofile %}

                            &lt;img src="{{ reply.user.petownerprofile.human_image.url
 }}" alt="{{ reply.user.username }}" class="comment-author-avatar"&gt;

                        {% elif reply.user.vetprofile %}

                            &lt;img src="{{ reply.user.vetprofile.vet_image.url }}" alt="{{
 reply.user.username }}" class="comment-author-avatar"&gt;

                        {% else %}

                            &lt;img src="{% static 'default-avatar.png' %}" alt="Default
Avatar" class="comment-author-avatar"&gt;

                        {% endif %}

                    &lt;/a&gt;

                &lt;/div&gt;
</pre>

```

```

<span class="comment-author">
    <a href="{% url 'authUser:user_profile' reply.user.id %}"
class="username-link">
        u/{{ reply.user.username }}
    </a>
</span>
<span class="comment-time">{{ reply.date|timesince }} ago</span>
</div>
<div class="comment-body">
    <p>{{ reply.reply }}</p>
</div>
<div class="comment-actions">
    {% if request.user == reply.user %}
        <button class="action-btn delete-btn"
onclick="confirmDeleteReply('{{ reply.id }}')">
            <i class="fas fa-trash"></i> Delete
        </button>
    {% endif %}
</div>
</div>
<% endfor %>
</div>

<!-- Reply Form (hidden by default) -->
<div class="reply-form-container" id="reply-form-{{ comment.id }}"
style="display: none;">
    <form method="POST" class="reply-form" action="{% url
'coreFunctions:add_reply' comment.id %}" data-comment-id="{{ comment.id }}">

```

```
{% csrf_token %}

<div class="form-group">

    <textarea name="reply" placeholder="Write your reply..." required></textarea>

    <div class="form-footer">

        <button type="button" class="btn btn-outline cancel-reply" data-comment-id="{{ comment.id }}>Cancel</button>

        <button type="submit" class="btn btn-primary">

            <i class="fas fa-paper-plane"></i> Reply

        </button>

    </div>

</div>

</form>

</div>

</div>

<% empty %>

<div class="empty-comments">

    <i class="far fa-comment-dots"></i>

    <p>No comments yet. Be the first to share what you think!</p>

</div>

<% endfor %>

</div>

</div>

<!-- Category Sidebar -->

<div class="category-sidebar">

    <!-- Categories Card -->
```

```

<div class="category-card glass-card">
    <h3 class="category-header">
        <i class="fas fa-folder"></i> Categories
    </h3>
    <ul class="category-list">
        {% for category in categories %}
            <li class="category-item">
                <a href="{% url 'coreFunctions:category_posts' category.slug %}"
                    class="category-link {% if selected_category == category %}active{% endif %}">
                    <i class="fas fa-hashtag"></i>
                    <span>{{ category.name }}</span>
                    <span class="category-count">{{ category.post_count }}</span>
                </a>
            </li>
        {% empty %}
        <li class="category-item">No categories available</li>
    {% endfor %}
    </ul>
</div>

<!-- Trending Posts --&gt;
&lt;div class="category-card glass-card"&gt;
    &lt;h3 class="category-header"&gt;
        &lt;i class="fas fa-fire"&gt;&lt;/i&gt; Trending Posts
    &lt;/h3&gt;
    &lt;div class="trending-posts"&gt;
        {% for trending_post in trending_posts %}
</pre>

```

```

<a href="{% url 'coreFunctions:post-detail' trending_post.slug %}"
class="trending-post">
    {% if trending_post.image %}
        
    {% else %}
        <div class="trending-post-image" style="background-color: #e9ecef;
display: flex; align-items: center; justify-content: center;">
            <i class="fas fa-image" style="color: #adb5bd;"></i>
        </div>
    {% endif %}
    <div class="trending-post-content">
        <div class="trending-post-title">{{ trending_post.title }}</div>
        <div class="trending-post-meta">
            <span>{{ trending_post.likes.count }} likes</span> •
            <span>{{ trending_post.comments.count }} comments</span>
        </div>
    </div>
    </a>
    {% empty %}
    <div class="trending-post">
        <div class="trending-post-content">
            <div class="trending-post-title">No trending posts available</div>
        </div>
    </div>
    {% endfor %}
</div>
</div>

```

7.9.1.2 VIEWS

```
@login_required(login_url='authUser:register')

def index(request, category_slug=None):
    # Get sort parameter (default to 'newest')
    sort_by = request.GET.get('sort', 'newest')

    # Get date filter parameter
    date_filter = request.GET.get('date')

    # Start with all posts
    posts_query = Post.objects.filter(active=True)

    # Apply category filter if provided
    selected_category = None
    if category_slug:
        selected_category = get_object_or_404(Category, slug=category_slug)
        posts_query = posts_query.filter(category=selected_category)

    # Apply sorting
    if sort_by == 'likes':
        # Sort by most likes
        posts = posts_query.annotate(like_count=Count('likes')).order_by('-like_count')
    elif sort_by == 'comments':
        # Sort by most comments
        posts = posts_query.annotate(comment_count=Count('comments')).order_by('-comment_count')
    elif sort_by == 'oldest':
        posts = posts_query.order_by('date')
```

```

# Sort by oldest first
posts = posts_query.order_by('date')

else:
    # Default: sort by newest
    posts = posts_query.order_by('-date')

# Get all categories with post count
categories = Category.objects.annotate(post_count=Count('post'))

posts_with_engagement = Post.objects.annotate(
    like_count=Count('likes'),
    comment_count=Count('comments'),
    engagement_score=ExpressionWrapper(
        Count('likes') + Count('comments'),
        output_field=IntegerField()
    )
).order_by('-engagement_score')[10] # top 10 by total engagement

# Pick 5 random ones from top 10
trending_posts = sample(list(posts_with_engagement), min(5,
len(posts_with_engagement)))

# Check if this is an AJAX request
if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
    # Return only the posts HTML for AJAX requests
    return render(request, 'main/partials/feed_posts.html', {'posts': posts})

```

```

context = {
    'posts': posts,
    'categories': categories,
    'trending_posts': trending_posts,
    'selected_category': selected_category,
    'sort_by': sort_by,
}

return render(request, 'coreFunctions/index.html', context)

```

```

@login_required
def like_post(request, post_id):
    """Handle post liking via AJAX"""
    if request.method == 'POST':
        try:
            post = Post.objects.get(id=post_id)
            user = request.user

            # Toggle like
            if user in post.likes.all():
                post.likes.remove(user)
                liked = False
            else:
                post.likes.add(user)
                liked = True

            return JsonResponse({
                'status': 'success',

```

```

    'likes_count': post.likes.count(),
    'liked': liked
)
except Post.DoesNotExist:
    return JsonResponse({'status': 'error', 'message': 'Post not found'}, status=404)

return JsonResponse({'status': 'error', 'message': 'Invalid request'}, status=400)

@login_required
def create_post(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        body = request.POST.get('body')
        category_id = request.POST.get('category')
        image = request.FILES.get('image')

        # Validate required fields
        if not title or not body or not category_id:
            messages.error(request, 'Please fill in all required fields')
            return redirect('coreFunctions:index')

        # Get category
        try:
            category = Category.objects.get(id=category_id)
        except Category.DoesNotExist:
            messages.error(request, 'Invalid category')
            return redirect('coreFunctions:index')

```

```

# Create slug from title
slug = slugify(title)

# Check if slug already exists, if so, add a unique identifier
if Post.objects.filter(slug=slug).exists():
    slug = f'{slug}-{shortuuid.uuid()[:6]}'

# Create post
post = Post.objects.create(
    title=title,
    body=body,
    category=category,
    user=request.user,
    slug=slug
)
# Add image if provided
if image:
    post.image = image
    post.save()

messages.success(request, 'Post created successfully')
return redirect('coreFunctions:post-detail', slug=post.slug)

# If not POST, redirect to feed
return redirect('coreFunctions:index')

def feed(request):
    """Alias for index view to match the URL in JavaScript"""
    return index(request)

```

```
def post_detail(request, slug):
    # Get active post by slug
    post = get_object_or_404(Post, slug=slug, active=True)

    # Categories with annotated post counts
    categories = Category.objects.annotate(post_count=Count('post'))

    posts_with_engagement = Post.objects.filter(
        category=post.category, # assuming `post` is defined
        active=True
    ).annotate(
        like_count=Count('likes'),
        comment_count=Count('comments'),
        engagement_score=ExpressionWrapper(
            Count('likes') + Count('comments'),
            output_field=IntegerField()
        )
    ).order_by('-engagement_score')[:10] # top 10 by total engagement

    # Pick 5 random ones from top 10
    trending_posts = sample(list(posts_with_engagement), min(5,
        len(posts_with_engagement)))

    # Active comments and their active replies
```



```
return render(request, 'coreFunctions/contact.html')

def appointment(request):
    return render(request, 'coreFunctions/appointment.html')
```

7.11.2 Appointment

7.11.1.1 HTML

```
<div class="pet-feed-content">  
  <div class="vet-list-container">  
    <!-- Header Section - centered like appointments page -->  
    <div class="section-header">  
      <div class="header-content">  
        <h1><i class="fas fa-search"></i> Find Veterinarians</h1>  
        <p>Browse our network of trusted veterinary professionals</p>  
      </div>  
  
      <div class="vet-filters">  
        <div class="filter-group">  
          <label for="experience-sort"><i class="fas fa-sort"></i> Sort by:</label>  
          <select id="experience-sort" class="filter-select">  
            <option value="">All Vets</option>  
            <option value="experience_high">Experience: High to Low</option>  
            <option value="experience_low">Experience: Low to High</option>  
          </select>  
        </div>  
  
        <div class="filter-group">  
          <label for="location-search"><i class="fas fa-map-marker-alt"></i> Location:</label>  
          <input type="text" id="location-search" class="filter-select" placeholder="City or area...">  
        </div>  
      </div>  
    </div>
```

```

<!-- Vet Cards Grid - updated to match appointments styling -->


{% for vet in vets %}
        <div class="vet-card" data-experience="{{ vet.experience_years|default:0 }}"
            data-location="{{ vet.city|default:"|lower }}" data-verified="{{ vet.verified|yesno:'true,false' }}>
            <div class="vet-card-header">
                <div class="vet-avatar">
                    {% if vet.user.vetprofile.vet_image %}
                        
                    {% else %}
                        
                    {% endif %}
                </div>
                <div class="vet-info">
                    <div class="vet-name-container">
                        <h3>Dr. {{ vet.user.full_name }}</h3>
                        {% if vet.verified %}
                            <div class="verification-badge-inline"><i class="fas fa-check-circle"></i> Verified</div>
                        {% endif %}
                    </div>
                    <div class="vet-meta">
                        {% if vet.clinic_name %}
                            <span class="clinic">
                                <i class="fas fa-clinic-medical"></i> {{ vet.clinic_name }}
                            </span>
                        {% endif %}
                    </div>
                </div>
            </div>
        {% endfor %}
    </div>


```

```

    {% endif %}

    {% if vet.city %}
        <span class="location">
            <i class="fas fa-map-marker-alt"></i> {{ vet.city }}{% if vet.country
%}, {{ vet.country }}{% endif %}
        </span>
    {% endif %}
</div>

<div class="rating">
    {% with rating=vet.average_rating %}
        {% for i in "12345" %}
            {% if forloop.counter <= rating %}
                <i class="fas fa-star"></i> {% Full star %}
            {% elif forloop.counter == rating|add:"0.5"|floatformat:0 and
rating|floatformat:1|slice:"-1" == "5" %}
                <i class="fas fa-star-half-alt"></i> {% Half star %}
            {% else %}
                <i class="far fa-star"></i> {% Empty star %}
            {% endif %}
        {% endfor %}
        <span>{{ rating|floatformat:1 }}/{{ vet.reviews.count }}</span>
    {% endwith %}
</div>
</div>

<div class="vet-card-body">

```

```

<div class="vet-bio">
    <p>{{ vet.summary|default:"Experienced veterinarian dedicated to
providing compassionate care for your pets."|truncatechars:150 }}</p>
</div>

<div class="vet-stats">
    {% if vet.experience_years %}
        <div class="stat">
            <i class="fas fa-graduation-cap"></i>
            <span>{{ vet.experience_years }} year{{ vet.experience_years|pluralize }} experience</span>
        </div>
    {% endif %}

    {% if vet.specialization %}
        <div class="stat">
            <i class="fas fa-stethoscope"></i>
            <span>Specializes in {{ vet.specialization }}</span>
        </div>
    {% endif %}
</div>

<div class="vet-card-footer">
    <a href="{% url 'appointment:vet_schedule' vet.id %}" class="btn btn-primary">
        <i class="fas fa-calendar-alt"></i> Book Appointment
    </a>
</div>

```

```
<a href="{% url 'authUser:user_profile' vet.user.id %}" class="btn btn-outline">
    <i class="fas fa-user"></i> View Profile
</a>
</div>
</div>

{% empty %}

<div class="empty-state">
    <i class="fas fa-search fa-3x"></i>
    <h3>No Veterinarians Found</h3>
    <p>We couldn't find any vets matching your criteria</p>
    <button id="reset-filters" class="btn btn-outline">
        <i class="fas fa-undo"></i> Reset Filters
    </button>
</div>

{% endfor %}

</div>
</div>

</div>

<div class="pet-feed-content">
    <div class="appointment-container">
        <div class="appointment-header">
            <h1><i class="fas fa-calendar-check"></i> Book an Appointment</h1>
            <div class="vet-info">
                <div class="vet-avatar">
```

```

        
    </div>

    <div class="vet-details">
        <h2>Dr. {{ vet.user.full_name }}</h2>
        {% if vet.clinic_name %}
            <p class="clinic-name"><i class="fas fa-clinic-medical"></i> {{ vet.clinic_name }}</p>
        {% endif %}
        {% if vet.specialization %}
            <p class="specialization"><i class="fas fa-stethoscope"></i> {{ vet.specialization }}</p>
        {% endif %}
    </div>
</div>
</div>

<div class="appointment-details">
    <div class="detail-card">
        <div class="detail-item">
            <i class="fas fa-calendar-day"></i>
            <div>
                <span class="label">Day</span>
                <span class="value">{{ schedule.day_of_week }}</span>
            </div>
        </div>
        <div class="detail-item">
            <i class="fas fa-clock"></i>
            <div>

```

```

<span class="label">Time</span>
<span class="value">{{ schedule.start_time }} to {{ schedule.end_time
}}</span>
</div>

</div>

<div class="detail-item">
  <i class="fas fa-money-bill-wave"></i>
  <div>
    <span class="label">Fee</span>
    <span class="value">NPR 1,000</span>
  </div>
</div>
</div>

</div>

<div class="appointment-form-container">
  <form method="POST" id="appointmentForm">
    {% csrf_token %}

    <div class="form-group">
      <label for="pet"><i class="fas fa-paw"></i> Select Your Pet:</label>
      <select name="pet" id="pet">
        <option value="">-- Choose a pet --</option>
        {% for pet in pets %}
          <option value="{{ pet.id }}>{{ pet.name }} ({{ pet.species }})</option>
        {% endfor %}
      </select>
      {% if not pets %}

```

```

<div class="no-pets-warning">
    <i class="fas fa-exclamation-triangle"></i>
    <span>You don't have any pets registered. <a href="{% url
'authUser:add_pet' %}">Add a pet</a> first.</span>
</div>
{% endif %}
</div>

<div class="form-group">
    <label for="reason"><i class="fas fa-comment-medical"></i> Reason for
Visit:</label>
    <textarea name="reason" id="reason" placeholder="Please describe your
pet's symptoms or reason for the appointment..."></textarea>
</div>

<div class="payment-options">
    <h3><i class="fas fa-credit-card"></i> Select Payment Method</h3>

    <div class="payment-buttons">
        <!-- Store Credit Option -->
        <button type="submit"
            formaction="{% url 'appointment:book_with_credit' vet.id
schedule.id %}"
            class="payment-btn credit-btn {% if
request.user.petownerprofile.credit_balance < 100000 %}disabled-btn{% endif %}"
            {% if request.user.petownerprofile.credit_balance < 100000
%}disabled{% endif %}>
            <i class="fas fa-wallet"></i>
            <span>Pay with Store Credit</span>
            {% with balance=request.user.petownerprofile.credit_balance %}

```

```

<span class="balance-info">
    Balance: NPR {% widtratio balance 100 1 %}.00
</span>
{% endwith %}
</button>

<!-- Khalti Payment Option -->
<button type="submit"
        formaction="{% url 'appointment:book_appointment' vet.id
schedule.id %}"
        class="payment-btn khalti-btn">
    <i class="fas fa-mobile-alt"></i>
    <span>Pay with Khalti</span>
</button>
</div>

{% if request.user.petownerprofile.credit_balance < 100000 %}
<div class="credit-warning">
    <i class="fas fa-info-circle"></i>
    <span>You need at least NPR 1,000 in store credit to use this payment
method.</span>
</div>
{% endif %}
</div>
</form>
</div>

<div class="back-link">
    <a href="{% url 'appointment:vet_schedule' vet.id %}">

```

```

        <i class="fas fa-arrow-left"></i> Back to Schedule
    </a>
</div>
</div>
</div>

<div class="pet-feed-content">
    <div class="appointment-list-container">
        {% if error %}
            <div class="alert alert-danger" role="alert">
                {{ error }}
            </div>
        {% endif %}

        <!-- Header Section - Matching appointments detail page -->
        <div class="section-header">
            <div class="header-content">
                <h1><i class="fas fa-calendar-plus"></i> Add New Schedule</h1>
                <p>Create new available time slots for appointments</p>
            </div>
        </div>

        <!-- Main Content -->
        <div class="vet-appointments-content">
            <div class="vet-profile-header">
                <div class="vet-avatar">
                    {% if vet.user.vetprofile.vet_image %}
                        
                    {% endif %}
                </div>
            </div>
        </div>
    </div>
</div>

```

```

{%- else %}

{%- endif %}

</div>

<div class="vet-info">

    <h2>Dr. {{ vet.user.full_name }}</h2>

    {%- if vet.clinic_name %}

        <p class="clinic-name">

            <i class="fas fa-clinic-medical"></i> {{ vet.clinic_name }}

        </p>

    {%- endif %}

    </div>

</div>

<!-- Schedule Form --&gt;

&lt;div class="form-container"&gt;

    &lt;form method="POST" class="schedule-form"&gt;

        {%- csrf_token %}

        &lt;div class="form-group"&gt;

            &lt;label for="day_of_week" class="form-label"&gt;

                &lt;i class="fas fa-calendar-day"&gt;&lt;/i&gt; Day of the Week:

            &lt;/label&gt;

            &lt;select name="day_of_week" id="day_of_week" class="form-select" required&gt;

                &lt;option value="Monday"&gt;Monday&lt;/option&gt;
                &lt;option value="Tuesday"&gt;Tuesday&lt;/option&gt;
</pre>

```

```

        <option value="Wednesday">Wednesday</option>
        <option value="Thursday">Thursday</option>
        <option value="Friday">Friday</option>
        <option value="Saturday">Saturday</option>
        <option value="Sunday">Sunday</option>
    </select>
</div>

<div class="form-group">
    <label for="start_time" class="form-label">
        <i class="fas fa-clock"></i> Start Time:
    </label>
    <input type="time" name="start_time" id="start_time" class="form-input"
required>
</div>

<div class="form-group">
    <label for="end_time" class="form-label">
        <i class="fas fa-clock"></i> End Time:
    </label>
    <input type="time" name="end_time" id="end_time" class="form-input"
required>
</div>

<div class="form-actions">
    <button type="submit" class="btn btn-primary">
        <i class="fas fa-save"></i> Save Schedule
    </button>

```

```
<a href="{% url 'appointment:vet_schedule' vet.id %}" class="btn btn-outline">  
    <i class="fas fa-arrow-left"></i> Cancel  
</a>  
</div>  
</form>  
</div>  
</div>  
</div>  
</div>
```

7.11.1.2 VIEWS

```
def send_appointment_notification(appointment, subject, template, is_rejection=False):
    context = {
        'appointment': appointment,
        'amount': appointment.amount_paid / 100, # Convert paisa to NPR
        'is_rejection': is_rejection
    }
    html_message = render_to_string(template, context)
    plain_message = strip_tags(html_message)

    send_mail(
        subject,
        plain_message,
        settings.DEFAULT_FROM_EMAIL,
        [appointment.pet_owner.user.email],
        html_message=html_message
    )

def send_vet_appointment_notification(appointment, subject, template):
    """
    Send appointment notification email to the vet.
    """

    context = {
        'appointment': appointment,
        'amount': appointment.amount_paid / 100, # Convert paisa to NPR
    }
    html_message = render_to_string(template, context)
    plain_message = strip_tags(html_message)
```

```

send_mail(
    subject,
    plain_message,
    settings.DEFAULT_FROM_EMAIL,
    [appointment.vet.user.email], # Correctly sending to vet
    html_message=html_message
)

# Payment Views tested
@login_required
def initiate_khalti_payment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Standard amount for demo (1000 NPR in paisa)
    amount_in_paisa = 1000 * 100

    payload = {
        "return_url": f"http://127.0.0.1:8000/appointment/verify_khalti/{appointment_id}/",
        "website_url": 'http://127.0.0.1:8000/',
        "amount": amount_in_paisa,
        "purchase_order_id": str(appointment.id),
        "purchase_order_name": f"Vet Appointment - {appointment.vet.user.username}",
        "customer_info": {
            "name": request.user.username,
            "email": request.user.email,
            "phone": KHALTI_TEST_ID
        }
    }

```

```

}

headers = {
    "Authorization": KHALTI_SECRET_KEY,
    "Content-Type": "application/json"
}

try:
    response = requests.post(
        f"{KHALTI_BASE_URL}/initiate/",
        json=payload,
        headers=headers
    )
    response.raise_for_status()
    data = response.json()
    return redirect(data['payment_url'])

except requests.exceptions.RequestException as e:
    return render(request, 'appointment/payment_error.html', {
        'error': str(e),
        'appointment': appointment
    })

#tested
@login_required
@csrf_exempt

def verify_khali_payment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

```

```

pidx = request.GET.get('pidx')

if not pidx:
    return render(request, 'appointment/payment_error.html', {
        'error': 'Missing payment ID',
        'appointment': appointment
    })

try:
    response = requests.post(
        f"{KHALTI_BASE_URL}/lookup/",
        json={"pidx": pidx},
        headers={
            "Authorization": KHALTI_SECRET_KEY,
            "Content-Type": "application/json"
        }
    )
    response.raise_for_status()
    data = response.json()

    if data.get('status') != 'Completed':
        appointment.status = 'unpaid'
        appointment.payment_status = 'failed'
        appointment.save()
        return redirect('appointment:payment_failed', appointment_id=appointment.id)

    # Payment successful - store details and await vet approval
    appointment.pidx = pidx

```

```

appointment.amount_paid = data['total_amount']
appointment.status = 'paid_pending_approval'
appointment.payment_status = 'paid'
appointment.save()

# Notify vet
send_vet_appointment_notification(
    appointment,
    "New Appointment Request",
    "appointment/new_appointment_email.html"
)

return redirect('appointment:payment_success', appointment_id=appointment.id)

except requests.exceptions.RequestException as e:
    appointment.status = 'unpaid'
    appointment.payment_status = 'failed'
    appointment.delete()
    return render(request, 'appointment/payment_error.html', {
        'error': str(e),
        'appointment': appointment
    })

@login_required
def vet_list(request):
    vets = VetProfile.objects.all()
    return render(request, 'appointment/vet_list.html', {'vets': vets})

```

```
@login_required

def vet_schedule(request, vet_id):
    vet = get_object_or_404(VetProfile, id=vet_id)

    # Get all schedules for this vet
    schedules = VetSchedule.objects.filter(vet=vet)

    # Define the order of days for sorting
    day_order = {
        'Sunday': 0,
        'Monday': 1,
        'Tuesday': 2,
        'Wednesday': 3,
        'Thursday': 4,
        'Friday': 5,
        'Saturday': 6
    }

    # Sort schedules by day of week first, then by start time
    sorted_schedules = sorted(schedules, key=lambda x: (day_order[x.day_of_week],
                                                       x.start_time))

    days_with_schedules = []
    for day in ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
               'Saturday']:
        days_with_schedules.append({
            'name': day,
```

```

'schedules': [schedule for schedule in sorted_schedules if
schedule.day_of_week == day]

})

return render(request, 'appointment/appointment_schedule.html', {
'vet': vet,
'days_with_schedules': days_with_schedules
})

```

#tested

```

def edit_schedule(request, schedule_id):
    user = request.user
    vet = get_object_or_404(VetProfile, user=user)
    schedule = get_object_or_404(VetSchedule, id=schedule_id, vet=vet)

    if request.method == 'POST':
        # Check if this is a delete request
        if 'delete_schedule' in request.POST:
            schedule.delete()
            return redirect('appointment:vet_schedule', vet_id=vet.id)

        # Otherwise, process as an edit request
        day_of_week = request.POST.get('day_of_week')
        start_time = datetime.strptime(request.POST.get('start_time'), '%H:%M').time()
        end_time = datetime.strptime(request.POST.get('end_time'), '%H:%M').time()

        # Validate time range

```

```

if start_time >= end_time:
    return render(request, 'appointment/edit_schedule.html', {
        'vet': vet,
        'schedule': schedule,
        'error': 'End time must be after start time.'
    })

# Check for overlapping schedules (excluding the current schedule)
overlapping_schedules = VetSchedule.objects.filter(
    vet=vet,
    day_of_week=day_of_week
).exclude(
    id=schedule_id # Exclude the current schedule
).exclude(
    # Exclude schedules that don't overlap
    end_time__lte=start_time # Existing ends before new starts
).exclude(
    start_time__gte=end_time # Existing starts after new ends
)

if overlapping_schedules.exists():
    return render(request, 'appointment/edit_schedule.html', {
        'vet': vet,
        'schedule': schedule,
        'error': 'The schedule overlaps with an existing one. Please select a different time.'
    })

```

```

# Update the schedule
schedule.day_of_week = day_of_week
schedule.start_time = start_time
schedule.end_time = end_time
schedule.save()

return redirect('appointment:vet_schedule', vet_id=vet.id)

return render(request, 'appointment/edit_schedule.html', {
    'vet': vet,
    'schedule': schedule
})

#tested

def add_schedule(request):
    user = request.user
    vet = get_object_or_404(VetProfile, user=user)

    if request.method == 'POST':
        day_of_week = request.POST.get('day_of_week')
        start_time = datetime.strptime(request.POST.get('start_time'), '%H:%M').time()
        end_time = datetime.strptime(request.POST.get('end_time'), '%H:%M').time()

        # Validate time range
        if start_time >= end_time:
            return render(request, 'appointment/appointment_creation_vets.html', {
                'vet': vet,
            })

```

```

    'error': 'End time must be after start time.'
})

# Check for overlapping schedules
overlapping_schedules = VetSchedule.objects.filter(
    vet=vet,
    day_of_week=day_of_week,
    available=True # Only check against other available schedules
).exclude(
    # Exclude schedules that don't overlap
    end_time__lte=start_time # Existing ends before new starts
).exclude(
    start_time__gte=end_time # Existing starts after new ends
)

if overlapping_schedules.exists():
    return render(request, 'appointment/appointment_creation_vets.html', {
        'vet': vet,
        'error': 'The schedule overlaps with an existing one. Please select a different
time.'
    })

# If no overlap, create the new schedule
VetSchedule.objects.create(
    pid=vet.pid,
    vet=vet,
    day_of_week=day_of_week,
    start_time=start_time,
)

```

```

        end_time=end_time,
        available=True
    )

    return redirect('appointment:vet_schedule', vet_id=vet.id)

return render(request, 'appointment/appointment_creation_vets.html', {'vet': vet})

#test done
@login_required
def book_appointment(request, vet_id, schedule_id):
    vet = get_object_or_404(VetProfile, id=vet_id)
    schedule = get_object_or_404(VetSchedule, id=schedule_id)
    pet_owner = get_object_or_404(PetOwnerProfile, user=request.user)
    pets = pet_owner.pets.all() # Fetch all pets for the dropdown

    if request.method == "POST":
        pet_id = request.POST.get("pet")
        reason = request.POST.get("reason", "No reason provided")

    try:
        selected_pet = Pet.objects.get(id=pet_id, owner=pet_owner)
    except Pet.DoesNotExist:
        return render(request, "appointment/book_appt.html", {
            "vet": vet,
            "schedule": schedule,
            "pets": pets,
        })

```

```

    "error": "Invalid pet selection."
})

appointment = Appointment.objects.create(
    pet_owner=pet_owner,
    vet=vet,
    schedule=schedule,
    pet=selected_pet,
    reason=reason,
    status="unpaid",
    payment_status='unpaid'
)

return redirect('appointment:initiate_khalti_payment',
appointment_id=appointment.id)

return render(request, "appointment/book_appt.html", {
    "vet": vet,
    "schedule": schedule,
    "pets": pets
})

#test done
@login_required
def book_with_credit(request, vet_id, schedule_id):
    vet = get_object_or_404(VetProfile, id=vet_id)
    schedule = get_object_or_404(VetSchedule, id=schedule_id)
    pet_owner = get_object_or_404(PetOwnerProfile, user=request.user)

```

```
pets = pet_owner.pets.all() # Fetch all pets for the dropdown

if request.method == "POST":
    reason = request.POST.get("reason", "No reason provided")
    pet_id = request.POST.get("pet")

    # Check if slot is already booked
    if Appointment.objects.filter(schedule=schedule, status="confirmed").exists():
        messages.error(request, "This slot is already booked.")
        return redirect('appointment:book_appointment', vet_id=vet_id,
schedule_id=schedule_id)

    # Check if user has sufficient credit (1000 NPR in paisa)
    if pet_owner.credit_balance < 1000 * 100:
        messages.error(request, "Insufficient credit balance")
        return redirect('appointment:book_appointment', vet_id=vet_id,
schedule_id=schedule_id)

try:
    # Check if valid pet is selected
    selected_pet = Pet.objects.get(id=pet_id, owner=pet_owner)

    # Deduct from credit balance
    pet_owner.credit_balance -= 1000 * 100 # Deduct 1000 NPR
    pet_owner.save()

    # Create appointment
    appointment = Appointment.objects.create(
```

```
pet_owner=pet_owner,
vet=vet,
schedule=schedule,
pet=selected_pet, # Assign the selected pet
reason=reason,
status="unpaid",
payment_status='unpaid',
amount_paid=1000 * 100 # 1000 NPR in paisa
)

# Payment successful - store details and await vet approval
appointment.status = 'paid_pending_approval'
appointment.payment_status = 'paid'
appointment.save()

# Send notification to vet
send_vet_appointment_notification(
    appointment,
    "New Appointment Booking",
    "appointment/new_appointment_email.html"
)

return redirect('appointment:appointment_detail',
appointment_id=appointment.id)

except Pet.DoesNotExist:
    messages.error(request, "Invalid pet selection.")
```

```

        return redirect('appointment:book_appointment', vet_id=vet_id,
schedule_id=schedule_id)

    except Exception as e:

        # Revert credit deduction if error occurs

        pet_owner.save()

        messages.error(request, f"Error booking appointment: {str(e)}")

        return redirect('appointment:book_appointment', vet_id=vet_id,
schedule_id=schedule_id)

# Render the page with the pet selection dropdown

return render(request, "appointment/book_appt.html", {

    "vet": vet,

    "schedule": schedule,

    "pets": pets

})

```

```

@login_required

def appointment_list(request):

    appointments = Appointment.objects.filter(
        pet_owner__user=request.user
    ).order_by('-created_at') # or '-date' depending on your model

    return render(request, "appointment/appointment_list.html", {"appointments": appointments})

```

```

@login_required

def appointment_detail(request, appointment_id):

```

```

"""View for displaying appointment details"""

appointment = get_object_or_404(Appointment, id=appointment_id)

# Check if the user is authorized to view this appointment
if request.user != appointment.pet_owner.user and request.user != appointment.vet.user:
    return HttpResponseForbidden("You don't have permission to view this appointment")

# Check if the user has already reviewed this appointment
has_reviewed = False

if appointment.status == 'completed' and appointment.payment_status == 'paid':
    has_reviewed = Review.objects.filter(
        vet=appointment.vet,
        reviewer=request.user,
        appointment=appointment
    ).exists()

context = {
    'appointment': appointment,
    'has_reviewed': has_reviewed
}

return render(request, 'appointment/appointment_detail.html', context)

#tested
@login_required

```

```

def cancel_appointment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)
    try:
        # Only process credit for paid appointments
        if appointment.payment_status == 'paid' and appointment.status in ['confirmed', 'paid_pending_approval']:
            # Calculate 80% refund (in paisa)
            refund_amount = int(appointment.amount_paid * 0.8)

            # Add credit to owner's account
            appointment.pet_owner.credit_balance += refund_amount
            appointment.pet_owner.save()

            # Update appointment
            appointment.status = 'cancelled'
            appointment.save()

            messages.success(request, f"Appointment cancelled. {refund_amount/100:.2f}\
NPR (80%) credited to your account.")

        else:
            # For unpaid appointments or invalid statuses
            appointment.status = 'cancelled'
            appointment.save()

            messages.warning(request, "Appointment cancelled (no refund applicable).")

    except Exception as e:
        messages.error(request, f"Error cancelling appointment: {str(e)}")
        return redirect('appointment:appointment_detail', appointment_id=appointment.id)

```

```

return redirect('appointment:appointment_list')

@login_required
def vet_pending_appointments(request, vet_id):
    vet = get_object_or_404(VetProfile, id=vet_id)
    pending_appointments = Appointment.objects.filter(vet=vet,
    status="paid_pending_approval")
    return render(request, "appointment/vet_pending_appointments.html", {
        "pending_appointments": pending_appointments,
        "vet": vet
    })

#tested
@login_required
def accept_appointment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

    if appointment.vet.user != request.user or appointment.status != "paid_pending_approval":
        return HttpResponseRedirect(reverse('appointment:vet_pending_appointments',
        args=[appointment.vet.id]))

    # Confirm the selected appointment
    appointment.status = "confirmed"
    appointment.save()

    # Reject other appointments for the same schedule and notify them

```

```
other_appointments = Appointment.objects.filter(  
    schedule=appointment.schedule  
).exclude(id=appointment.id)
```

```
for other in other_appointments:
```

```
    other.status = "rejected"
```

```
    other.save()
```

```
# Send rejection email
```

```
send_appointment_notification(
```

```
    other,
```

```
    "Appointment Rejected",
```

```
    "appointment/rejection_email.html",
```

```
    is_rejection=True
```

```
)
```

```
# Mark the schedule as unavailable
```

```
appointment.schedule.available = False
```

```
appointment.schedule.save()
```

```
# Notify the confirmed pet owner
```

```
send_appointment_notification(
```

```
    appointment,
```

```
    "Appointment Confirmed",
```

```
    "appointment/confirmation_email.html"
```

```
)
```

```

        return HttpResponseRedirect(reverse('appointment:vet_pending_appointments',
args=[appointment.vet.id]))


#tested
@login_required
def reject_appointment(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Check permissions
    if not hasattr(appointment, 'vet') or not hasattr(appointment.vet, 'user') or \
       request.user != appointment.vet.user or appointment.status != "paid_pending_approval":
        messages.error(request, "You don't have permission to reject this appointment.")

    return HttpResponseRedirect(reverse('appointment:vet_pending_appointments',
args=[appointment.vet.id]))


try:
    # Add credit to pet owner's account if payment was made
    if appointment.status == "paid_pending_approval" and
appointment.payment_status == 'paid':
        appointment.pet_owner.credit_balance += appointment.amount_paid
        appointment.pet_owner.save()
        messages.success(request, "Appointment rejected. Amount credited to user's
account.")


    # Update appointment status
    appointment.status = "rejected"
    appointment.save()

```

```

# Send rejection notification
send_appointment_notification(
    appointment,
    "Appointment Rejected",
    "appointment/rejection_email.html",
    is_rejection=True
)

except Exception as e:
    messages.error(request, f"Error rejecting appointment: {str(e)}")
    return HttpResponseRedirect(reverse('appointment:vet_pending_appointments',
                                     args=[appointment.vet.id]))

return HttpResponseRedirect(reverse('appointment:vet_pending_appointments',
                                     args=[appointment.vet.id]))


@login_required
def vet_accepted_appointments(request, vet_id):
    vet = get_object_or_404(VetProfile, id=vet_id)
    accepted_appointments = Appointment.objects.filter(vet=vet, status="confirmed")
    return render(request, "appointment/vet_accepted_appointments.html", {
        "accepted_appointments": accepted_appointments,
        "vet": vet
    })

#tested

```

```

@login_required
def mark_completed(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Check permissions
    if not hasattr(appointment, 'vet') or not hasattr(appointment.vet, 'user') or \
        request.user != appointment.vet.user:
        messages.error(request, "You don't have permission to mark this appointment as
completed.")

    return HttpResponseRedirect(reverse('appointment:vet_accepted_appointments',
args=[appointment.vet.id]))

if request.method == "POST":
    appointment.status = "completed"
    appointment.save()
    appointment.schedule.available = True
    appointment.schedule.save()

return redirect("appointment:vet_accepted_appointments", vet_id=appointment.vet.id)

```

Payment Status Views

```

@login_required
def payment_success(request, appointment_id):
    appointment = get_object_or_404(Appointment, id=appointment_id)
    return render(request, 'appointment/payment_success.html', {
        'appointment': appointment
    })

```

```
@login_required  
def payment_failed(request, appointment_id):  
    appointment = get_object_or_404(Appointment, id=appointment_id)  
    return render(request, 'appointment/payment_failed.html', {  
        'appointment': appointment  
    })
```

7.11.3 Chat

7.11.3.1 HTML

```

<meta name="current-user" content="{{ request.user.username }}>
{%- if receiver %}

<meta name="receiver-user" content="{{ receiver.username }}>
{%- endif %}

<div class="pet-feed-content">
  <div class="main_content">
    <div class="chat-container">
      <!-- Chat List -->
      <div class="chat-list">
        <div class="chat-list-header">
          <h2><i class="fas fa-comments"></i> Messages</h2>
          <div class="search-box">
            <input type="text" id="search-contacts" placeholder="Search
conversations...">
            <i class="fas fa-search"></i>
          </div>
        </div>

      <!-- Chat List Items -->
      <div class="chat-list-items">
        {%- for chat in message_list %}

          {# Determine conversation partner #}

          {%- if chat.sender == request.user %}

            {%- with partner=chat.receiver %}

              <a href="{% url 'coreFunctions:inbox_details' partner.username %}"
class="chat-link">

```

```

<div class="chat-item {%- if receiver == partner %}active{%- endif %}" data-username="{{ partner.username }}>
    {% if partner.petownerprofile.human_image %}
        
    {% elif partner.vetprofile.vet_image %}
        
    {% else %}
        
    {% endif %}

    <div class="chat-item-content">
        <div class="chat-item-header">
            <h4>{{ partner.get_full_name|default:partner.username }}</h4>
            <span class="chat-time">{{ chat.date|time:"h:i A" }}</span>
        </div>
        <p>{{ chat.message|truncatechars:28 }}</p>
        {% if chat.receiver == request.user and not chat.is_read %}
            <span class="unread-badge"></span>
        {% endif %}
    </div>
</div>
</a>
{% endwith %}
{% else %}
    {% with partner=chat.sender %}
        <a href="{% url 'coreFunctions:inbox_details' partner.username %}" class="chat-link">
            <div class="chat-item {%- if receiver == partner %}active{%- endif %}" data-username="{{ partner.username }}>

```

```

        {% if partner.petownerprofile.human_image %}

        {% elif partner.vetprofile.vet_image %}

        {% else %}

        {% endif %}

            <div class="chat-item-content">

                <div class="chat-item-header">

                    <h4>{{ partner.get_full_name|default:partner.username }}</h4>

                    <span class="chat-time">{{ chat.date|time:"h:i A" }}</span>

                </div>

                <p>{{ chat.message|truncatechars:28 }}</p>

                {% if chat.receiver == request.user and not chat.is_read %}

                    <span class="unread-badge"></span>

                {% endif %}

            </div>

        </div>

    </a>

    {% endwith %}

    {% endif %}

    {% empty %}

        <div class="empty-chat-list">

            <i class="fas fa-comments"></i>

            <p>No conversations yet</p>

            <small>Start a new conversation from the community page</small>

        </div>

    
```

```

        </div>
    {% endfor %}
</div>
</div>

<!-- Chat Window --&gt;
&lt;div class="chat-window"&gt;
    {% if receiver %}
        &lt;div class="chat-header"&gt;
            &lt;div class="chat-header-user"&gt;
                {% if receiver.petownerprofile.human_image %}
                    &lt;img src="{{ receiver.petownerprofile.human_image.url }}" class="chat-avatar"&gt;
                {% elif receiver.vetprofile.vet_image %}
                    &lt;img src="{{ receiver.vetprofile.vet_image.url }}" class="chat-avatar"&gt;
                {% else %}
                    &lt;img src="{% static 'assets/images/default-avatar.png' %}" class="chat-avatar"&gt;
                {% endif %}
            &lt;div&gt;
                &lt;a href="{% url 'authUser:user_profile' receiver.id %}"
                    style="font-size: 20px; font-weight: 500; color: #333; text-decoration: none;"&gt;
                    {{ receiver.get_full_name|default:receiver.username }}
                &lt;/a&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="chat-header-actions"&gt;
            &lt;button class="action-btn" id="block-user-btn" title="Block User"&gt;
                &lt;i class="fas fa-ban"&gt;&lt;/i&gt;
            &lt;/button&gt;
        &lt;/div&gt;
    {% endif %}
&lt;/div&gt;
</pre>

```

```
</div>
</div>

<div class="chat-messages" id="chat-messages">
    {% for message in message_detail %}
        <div class="message" {% if message.sender == request.user %}sent{% else %}received{% endif %}>
            <div class="message-content">{{ message.message }}</div>
            <div class="message-meta">
                <span class="message-time">{{ message.date|time:"h:i A" }}</span>
                {% if message.sender == request.user %}
                    <span class="message-status">
                        {% if message.is_read %}
                            <i class="fas fa-check-double"></i>
                        {% else %}
                            <i class="fas fa-check"></i>
                        {% endif %}
                    </span>
                {% endif %}
            </div>
        </div>
    {% empty %}
    <div class="empty-messages">
        <div class="empty-messages-content">
            <i class="fas fa-comments"></i>
            <p>No messages yet</p>
            <small>Start the conversation by sending a message below</small>
        </div>
    </div>

```

```

        </div>
        </div>
    {% endfor %}
</div>

<div class="chat-input">
    <form method="POST" action="{% url 'coreFunctions:inbox_details' receiver.username %}" id="chat-f">
        {% csrf_token %}
        <div class="input-wrapper">
            <textarea id="chat-input-area" name="message" placeholder="Type your message..." required></textarea>
            <div class="input-actions">
                <button type="button" class="emoji-btn" title="Add Emoji">
                    <i class="far fa-smile"></i>
                </button>
            </div>
        </div>
        <button id="send-btn" type="submit">
            <i class="fas fa-paper-plane"></i>
        </button>
    </form>
</div>
{% else %}
<div class="empty-chat">
    <div class="empty-chat-content">
        <i class="fas fa-comments"></i>
        <h3>Welcome to Messages</h3>
    </div>
</div>

```

```
<p>Select a conversation to start chatting</p>
</div>
</div>
{%
  endif %
}
</div>
</div>

</div>
</div>

<script>
document.addEventListener("DOMContentLoaded", () => {
  // Track if we've handled the first message
  let isFirstMessageHandled = false;

  // Initialize WebSocket for all users
  setupGlobalWebSocket();

  // Set up WebSocket for the active conversation if there is one
  if (document.querySelector(".chat-header")) {
    setupActiveConversationWebSocket();
  }

  function setupGlobalWebSocket() {
    const currentUser = "{{ request.user.username }}";
    if (!currentUser) return;

    const wsScheme = window.location.protocol === "https:" ? "wss" : "ws";

```

```
const wsPath =
`"${wsScheme}://${window.location.host}/ws/notifications/${currentUser}/`;

let notificationSocket = new WebSocket(wsPath);

notificationSocket.onopen = () => {
    console.log("Notification WebSocket connected");
};

notificationSocket.onmessage = (e) => {
    const data = JSON.parse(e.data);
    console.log("Notification received:", data);

    if (data.type === "new_message") {
        updateChatListPreview(data.sender, data.message, true, data.timestamp);

        // If the user is not currently viewing this chat, show a notification
        if (!document.querySelector(`.chat-item[data-username="${data.sender}"].active`))
    {
        showDesktopNotification(data.sender, data.message);
    }
}
};

notificationSocket.onerror = (error) => {
    console.error("Notification WebSocket error:", error);
    setTimeout(setupGlobalWebSocket, 3000);
};
```

```

notificationSocket.onclose = () => {
  console.log("Notification WebSocket closed, reconnecting...");
  setTimeout(setupGlobalWebSocket, 3000);
};

}

function setupActiveConversationWebSocket() {
  const currentUser = "{{ request.user.username }}";
  const receiverUsername = "{{ receiver.username }}";

  if (!currentUser || !receiverUsername) return;

  // Create consistent room name
  const roomName = [currentUser, receiverUsername].sort().join("_");
  const wsScheme = window.location.protocol === "https:" ? "wss" : "ws";
  const wsPath = `${wsScheme}://${window.location.host}/ws/chat/${roomName}`;
  let chatSocket = new WebSocket(wsPath);

  // Track messages we've added to the UI
  const displayedMessages = new Set();

  chatSocket.onopen = () => {
    console.log("Chat WebSocket connected for room:", roomName);
  };

  chatSocket.onerror = (error) => {
    console.error("Chat WebSocket error:", error);
  };
}

```

```
chatSocket.onclose = () => {
    console.log("Chat WebSocket closed, reconnecting...");
    setTimeout(() => {
        const newSocket = new WebSocket(wsPath);
        chatSocket = newSocket;
    }, 3000);
};

chatSocket.onmessage = (e) => {
    const data = JSON.parse(e.data);
    console.log("Chat message received:", data);

    // Create a unique message ID
    const messageId = `${data.sender}_${data.timestamp} || new Date().toISOString()`;

    if (displayedMessages.has(messageId)) {
        console.log("Skipping duplicate message:", messageId);
        return;
    }

    if ((data.sender === currentUser && data.receiver === receiverUsername) ||
        (data.sender === receiverUsername && data.receiver === currentUser)) {

        // Mark as displayed before adding to UI to prevent duplicates
        displayedMessages.add(messageId);

        // Add message to UI
    }
}
```

```
addMessageToUI(data, data.sender === currentUser);

// Update chat list preview
updateChatListPreview(
    data.sender === currentUser ? receiverUsername : data.sender,
    data.message,
    data.receiver === currentUser,
    data.timestamp || new Date().toISOString()
);

// Update chat header
updateChatHeader(data.sender === currentUser ? receiverUsername : data.sender);
}

};

const chatForm = document.querySelector("#chat-f");
const textarea = document.querySelector("#chat-input-area");

if (chatForm) {
    chatForm.addEventListener("submit", async (e) => {
        e.preventDefault();
        const message = textarea.value.trim();

        if (message && chatSocket.readyState === WebSocket.OPEN) {
            const timestamp = new Date().toISOString();
            const messageData = {
                message: message,
```

```
    sender: currentUser,  
    receiver: receiverUsername,  
    timestamp: timestamp,  
};  
  
// Create unique ID and mark as displayed immediately  
const messageId = `${currentUser}_${timestamp}`;  
displayedMessages.add(messageId);  
  
// Optimistic UI update  
addMessageToUI({  
    message: message,  
    sender: currentUser,  
    timestamp: timestamp  
}, true);  
  
// Clear input  
textarea.value = "";  
textarea.style.height = "auto";  
  
try {  
    // Send via WebSocket  
    chatSocket.send(JSON.stringify(messageData));  
  
    // Update chat header for sender as well - THIS IS THE KEY FIX  
    updateChatHeader(receiverUsername);  
}  
} catch (error) {
```

```

        console.error("Error sending message:", error);
    }
}
});
}

function addMessageToUI(data, isSent) {
    const messageTime = formatTime(data.timestamp || new Date().toISOString());
    const messageDiv = document.createElement("div");
    messageDiv.className = `message ${isSent ? "sent" : "received"}`;
    messageDiv.innerHTML = `
        <div class="message-content">${data.message}</div>
        <div class="message-meta">
            <span class="message-time">${messageTime}</span>
            ${isSent ? '<span class="message-status"><i class="fas fa-check"></i></span>' :
        `}
        </div>
    `;
}

const chatMessages = document.querySelector(".chat-messages");
const emptyMessages = chatMessages.querySelector(".empty-messages");
if (emptyMessages) emptyMessages.remove();

chatMessages.appendChild(messageDiv);
chatMessages.scrollTop = chatMessages.scrollHeight;
}

```

```
function updateChatListPreview(username, message, isUnread, timestamp) {  
    const chatItems = document.querySelectorAll(".chat-item");  
    const formattedTime = formatTime(timestamp);  
  
    let found = false;  
    chatItems.forEach((item) => {  
        if (item.dataset.username === username) {  
            found = true;  
            const preview = item.querySelector("p");  
            if (preview) preview.textContent = message.length > 28 ? `${message.substring(0, 28)}...` : message;  
  
            const timeElement = item.querySelector(".chat-time");  
            if (timeElement) timeElement.textContent = formattedTime;  
  
            if (isUnread) {  
                let badge = item.querySelector(".unread-badge");  
                if (!badge) {  
                    badge = document.createElement("span");  
                    badge.className = "unread-badge";  
                    item.appendChild(badge);  
                }  
            } else {  
                const badge = item.querySelector(".unread-badge");  
                if (badge) badge.remove();  
            }  
        }  
    });  
    // Move to top
```

```
const parent = item.closest(".chat-link");
const container = parent.parentNode;
container.insertBefore(parent, container.firstChild);

}

});

// If chat item doesn't exist yet, we might need to create it
// This would require additional server-side support to get user details
if (!found) {
    console.log("Chat item for user", username, "not found. Consider refreshing the
page.");
}

function updateChatHeader(username) {
    const chatHeader = document.querySelector(".chat-header");
    if (!chatHeader) return;

    // Update the status text to show active
    const statusText = chatHeader.querySelector(".status-text");
    if (statusText) {
        statusText.textContent = "Active now";
    }

    // Make sure the status indicator shows online
    const statusIndicator = chatHeader.querySelector(".status-indicator");
    if (statusIndicator) {
        statusIndicator.className = "status-indicator online";
```

```
    }  
}  
  
function formatTime(timestamp) {  
    try {  
        return new Date(timestamp).toLocaleTimeString([], {  
            hour: "2-digit",  
            minute: "2-digit",  
        });  
    } catch (e) {  
        return new Date().toLocaleTimeString([], {  
            hour: "2-digit",  
            minute: "2-digit",  
        });  
    }  
}
```

```
function showDesktopNotification(sender, message) {  
    if (!("Notification" in window)) return;  
  
    if (Notification.permission === "granted") {  
        new Notification(`New message from ${sender}`, {  
            body: message,  
            icon: "/static/assets/images/notification-icon.png"  
        });  
    } else if (Notification.permission !== "denied") {  
        Notification.requestPermission().then(permission => {  
            if (permission === "granted") {  
                // Show notification here  
            }  
        });  
    }  
}
```

```
new Notification(`New message from ${sender}` , {  
    body: message,  
    icon: "/static/assets/images/notification-icon.png"  
});  
}  
});  
}  
}  
  
// Auto-resize textarea as user types  
const chatInput = document.querySelector("#chat-input-area");  
if (chatInput) {  
    chatInput.addEventListener("input", function() {  
        this.style.height = "auto";  
        this.style.height = (this.scrollHeight) + "px";  
    });  
}  
});  
</script>
```

7.11.3.2 VIEWS

@login_required

```
def inbox_details(request, username=None):
    sender = request.user
    receiver = None
    messages_detail = None

    if username:
        receiver = get_object_or_404(User, username=username)
        # Fetch conversation between sender and receiver, ordered by date
        messages_detail = ChatMessage.objects.filter(
            Q(sender=sender, receiver=receiver) | Q(sender=receiver, receiver=sender)
        ).order_by("date")
        # Mark received messages as read
        messages_detail.filter(receiver=sender).update(is_read=True)

    # Get recent conversations for sidebar
    recent_messages = ChatMessage.objects.filter(
        Q(sender=sender) | Q(receiver=sender)
    ).order_by("-date")

    # Organize conversations by partner, keeping the latest message
    conversation_partners = {}
    for message in recent_messages:
        if not message.sender or not message.receiver:
            continue
        partner = message.sender if message.sender != sender else message.receiver
        if partner.id not in conversation_partners:
```

```
conversation_partners[partner.id] = message

context = {
    "message_detail": messages_detail,
    "receiver": receiver,
    "message_list": conversation_partners.values(),
}

return render(request, 'chat/inbox_detail.html', context)
```

7.11.3.3 Routing

```
from django.urls import re_path
from coreFunctions import consumers

websocket_urlpatterns = [
    re_path(r'ws/chat/(?P<room_name>[\w-]+)/$', consumers.ChatConsumer.as_asgi()),
    re_path(r'ws/notifications/(?P<username>\w+)/$', consumers.NotificationConsumer.as_asgi()),
]

]
```

7.11.3.4 Consumer

```
from django.utils.timesince import timesince
from asgiref.sync import async_to_sync
from channels.generic.websocket import WebsocketConsumer
from channels.generic.websocket import AsyncWebsocketConsumer
import json
import datetime

from authUser.models import User
from coreFunctions.models import ChatMessage

class ChatConsumer(WebSocketConsumer):

    def connect(self):
        self.room_name = self.scope['url_route']['kwargs']['room_name']
        self.room_group_name = 'chat_%s' % self.room_name

        async_to_sync(self.channel_layer.group_add)(
            self.room_group_name,
            self.channel_name
        )

        self.accept()

    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)(
            self.room_group_name,
            self.channel_name
        )
```

```
def receive(self, text_data):
    data = json.loads(text_data)
    message = data.get('message')
    sender_username = data.get('sender')
    timestamp = data.get('timestamp', datetime.datetime.now().isoformat())

    try:
        # Get sender and receiver user objects
        sender = User.objects.get(username=sender_username)
        receiver = User.objects.get(username=data['receiver'])

        # Create and save the chat message
        chat_message = ChatMessage(
            user=sender, # Set user as sender
            sender=sender,
            receiver=receiver,
            message=message,
        )
        chat_message.save()

        print(f"Message saved to database: {sender_username} to {receiver.username}:
{message}")
    
```

```
    # Send to the chat room
    async_to_sync(self.channel_layer.group_send)(
        self.room_group_name,
        {

```

```

        'type': 'chat_message',
        'message': message,
        'sender': sender_username,
        'receiver': receiver.username,
        'timestamp': timestamp,
    }
)

# Also send notification to receiver's notification group
async_to_sync(self.channel_layer.group_send)(
    f'notifications_{receiver.username}',
    {
        'type': 'send_notification',
        'message': message,
        'sender': sender_username,
        'timestamp': timestamp,
    }
)

except User.DoesNotExist as e:
    print(f"Error: User not found - {e}")
except Exception as e:
    print(f"Error saving message: {e}")

def chat_message(self, event):
    self.send(text_data=json.dumps(event))

class NotificationConsumer(AsyncWebSocketConsumer):

```

```
async def connect(self):
    self.username = self.scope['url_route']['kwargs']['username']
    self.room_group_name = f'notifications_{self.username}'

    # Join room group
    await self.channel_layer.group_add(
        self.room_group_name,
        self.channel_name
    )

    await self.accept()

async def disconnect(self, close_code):
    # Leave room group
    await self.channel_layer.group_discard(
        self.room_group_name,
        self.channel_name
    )

async def send_notification(self, event):
    # Send notification to WebSocket
    await self.send(text_data=json.dumps({
        'type': 'new_message',
        'message': event['message'],
        'sender': event.get('sender'),
        'timestamp': event.get('timestamp'),
    }))
```

7.11.4 Admin Dashboard

7.11.4.1 HTML

```
{% extends 'main/base3.html' %}

{% load static %}

{% block title %}Vet Approvals - PetVet Admin{% endblock %}
```

```
{% block page_title %}Vet Approvals{% endblock %}
```

```
{% block content %}

<div class="stats-grid">

    <div class="stat-card">
        <div class="stat-title">Pending Approvals</div>
        <div class="stat-value">{{ pending_vets.count }}</div>
        <div class="stat-description">Vets waiting for approval</div>
    </div>

    <div class="stat-card">
        <div class="stat-title">Recently Approved</div>
        <div class="stat-value">{{ recently_approved_vets.count }}</div>
        <div class="stat-description">Vets approved in the last 7 days</div>
    </div>

    <div class="stat-card">
        <div class="stat-title">Recently Declined</div>
        <div class="stat-value">{{ recently_declined_vets.count }}</div>
        <div class="stat-description">Vets declined in the last 7 days</div>
    </div>

</div>
```

```

<div class="table-container">

    <h3 class="chart-title" style="padding: 1rem 1rem 0 1rem;">Pending Vet Approvals

```

```

        <a href="{% url 'django_admin:approve_vet' vet.id %}" class="btn btn-success btn-sm">Approve</a>
        <a href="{% url 'django_admin:decline_vet' vet.id %}" class="btn btn-danger btn-sm">Decline</a>
    </div>
</td>
</tr>
{% empty %}
<tr>
    <td colspan="7" class="text-center">No pending vet approvals.</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>

```

```

<div class="table-container">
    <h3 class="chart-title" style="padding: 1rem 1rem 0 1rem;">Recently Approved Vets</h3>
    <table class="table">
        <thead>
            <tr>
                <th>Name</th>
                <th>Email</th>
                <th>Clinic Name</th>
                <th>License Number</th>
                <th>Approved At</th>
            </tr>
        </thead>

```

```

<tbody>
  {% for vet in recently_approved_vets %}
    <tr>
      <td>{{ vet.user.full_name }}</td>
      <td>{{ vet.user.email }}</td>
      <td>{{ vet.clinic_name }}</td>
      <td>{{ vet.license_number }}</td>
      <td>{{ vet.status_change|date:"M d, Y" }}</td>
    </tr>
  {% empty %}
  <tr>
    <td colspan="5" class="text-center">No recently approved vets.</td>
  </tr>
  {% endfor %}
</tbody>
</table>
</div>

```

```

<div class="table-container">
  <h3 class="chart-title" style="padding: 1rem 1rem 0 1rem;">Recently Declined
  Vets</h3>
  <table class="table">
    <thead>
      <tr>
        <th>Name</th>
        <th>Email</th>
        <th>Clinic Name</th>
        <th>License Number</th>
    
```

```

<th>Declined At</th>
</tr>
</thead>
<tbody>
  {% for vet in recently_declined_vets %}
    <tr>
      <td>{{ vet.user.full_name }}</td>
      <td>{{ vet.user.email }}</td>
      <td>{{ vet.clinic_name }}</td>
      <td>{{ vet.license_number }}</td>
      <td>{{ vet.status_change|date:"M d, Y" }}</td>
    </tr>
  {% empty %}
  <tr>
    <td colspan="5" class="text-center">No recently declined vets.</td>
  </tr>
  {% endfor %}
</tbody>
</table>
</div>
{% endblock %}

{% extends 'main/base3.html' %}
{% load static %}

{% block title %}Post Management - PetVet Admin{% endblock %}

{% block page_title %}Post Management{% endblock %}

```

```
{% block content %}

<div class="form-container">

    <form method="get" action="{% url 'django_admin:post_management' %}"
    class="search-form">

        <div class="form-group" style="display: flex; gap: 1rem;">

            <div style="flex: 1;">
                <label for="search" class="form-label">Search Posts</label>
                <input type="text" id="search" name="search" class="form-input"
                placeholder="Search by title or content" value="{{ request.GET.search }}">
            </div>

            <div style="width: 200px;">
                <label for="category" class="form-label">Category</label>
                <select id="category" name="category" class="form-select">
                    <option value="">All Categories</option>
                    {% for category in categories %}
                        <option value="{{ category.id }}" {% if request.GET.category ==
                        category.id|stringformat:"i" %}selected{% endif %}>{{ category.name }}</option>
                    {% endfor %}
                </select>
            </div>

            <div style="align-self: flex-end;">
                <button type="submit" class="btn btn-primary">Search</button>
            </div>
        </div>
    </form>
</div>

<div class="table-container">
```

```

<table class="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Author</th>
      <th>Category</th>
      <th>Date</th>
      <th>Likes</th>
      <th>Status</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for post in posts %}
      <tr>
        <td>{{ post.title }}</td>
        <td>{{ post.user.username }}</td>
        <td>{{ post.category.name|default:"Uncategorized" }}</td>
        <td>{{ post.date|date:"M d, Y" }}</td>
        <td>{{ post.likes.count }}</td>
        <td>
          {% if post.active %}
            <span class="badge badge-success">Active</span>
          {% else %}
            <span class="badge badge-danger">Inactive</span>
          {% endif %}
        </td>
      </tr>
    {% endfor %}
  </tbody>
</table>

```

```

<td>
    <div class="table-actions">
        <a href="{% url 'django_admin:view_post' post.id %}" class="btn btn-primary btn-sm">View</a>
        <button type="button" class="btn btn-danger btn-sm"
        onclick="confirmDelete('{{ post.id }}', '{{ post.title|escapejs }}')>Delete</button>
    </div>
</td>
</tr>
{% empty %}
<tr>
    <td colspan="8" class="text-center">No posts found.</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>

{% if posts.has_other_pages %}
<div class="pagination">
    {% if posts.has_previous %}
        <div class="pagination-item">
            <a href="?page=1{{ if request.GET.search }}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.category }}&category={{ request.GET.category }}{{ endif }}"
            class="pagination-link">
                &laquo;
            </a>
        </div>
        <div class="pagination-item">

```

```

    <a href="?page={{ posts.previous_page_number }}{{ if request.GET.search
}}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.category
}}&category={{ request.GET.category }}{{ endif }}" class="pagination-link">
        &lsaquo;
    </a>
</div>
{% endif %}

{% for num in posts.paginator.page_range %}
    {% if posts.number == num %}
        <div class="pagination-item">
            <a class="pagination-link active">{{ num }}</a>
        </div>
    {% elif num > posts.number|add:'-3' and num < posts.number|add:'3' %}
        <div class="pagination-item">
            <a href="?page={{ num }}{{ if request.GET.search }}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.category }}&category={{ request.GET.category }}{{ endif }}" class="pagination-link">{{ num }}</a>
        </div>
    {% endif %}
    {% endfor %}

    {% if posts.has_next %}
        <div class="pagination-item">
            <a href="?page={{ posts.next_page_number }}{{ if request.GET.search
}}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.category
}}&category={{ request.GET.category }}{{ endif }}" class="pagination-link">
                &rsaquo;
            </a>
        </div>
    {% endif %}

```

```

<div class="pagination-item">
    <a href="?page={{ posts.paginator.num_pages }}{{ if request.GET.search
%}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.category
%}&category={{ request.GET.category }}{{ endif }}" class="pagination-link">
        &raquo;
    </a>
</div>
{% endif %}
</div>
{% endif %}

<!-- Delete Confirmation Modal -->
<div id="deleteModal" class="modal" style="display: none;">
    <div class="modal-content">
        <h3>Confirm Deletion</h3>
        <p>Are you sure you want to delete post "<span
id="deletePostTitle"></span>"?</p>
        <p>This action cannot be undone and will delete all associated comments.</p>
        <div class="form-group">
            <label for="notifyUser" class="form-label">
                <input type="checkbox" id="notifyUser" checked> Send notification email to
the user
            </label>
        </div>
        <div class="modal-actions">
            <button type="button" class="btn btn-primary"
onclick="closeModal()">Cancel</button>
            <form id="deleteForm" method="post" action="">
                {% csrf_token %}
                <input type="hidden" name="notify_user" id="notifyUserInput" value="true">
            </form>
        </div>
    </div>
</div>

```

```
<button type="submit" class="btn btn-danger">Delete</button>
</form>
</div>
</div>
</div>

<style>
.modal {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 100;
}

.modal-content {
    background-color: white;
    padding: 2rem;
    border-radius: 0.5rem;
    max-width: 500px;
    width: 100%;
}
```

```
.modal-actions {  
    display: flex;  
    justify-content: flex-end;  
    gap: 1rem;  
    margin-top: 1.5rem;  
}  
  
.badge {  
    display: inline-block;  
    padding: 0.25rem 0.5rem;  
    border-radius: 0.25rem;  
    font-size: 0.75rem;  
    font-weight: 600;  
}  
  
.badge-success {  
    background-color: rgba(16, 185, 129, 0.1);  
    color: var(--success);  
}  
  
.badge-danger {  
    background-color: rgba(239, 68, 68, 0.1);  
    color: var(--danger);  
}  
}
```

```
<script>  
function confirmDelete(postId, postTitle) {
```

```
document.getElementById('deletePostTitle').textContent = postTitle;  
document.getElementById('deleteForm').action = "{% url 'django_admin:delete_post' 0 %}".replace('0', postId);  
document.getElementById('deleteModal').style.display = 'flex';  
}  
  
function closeModal() {  
    document.getElementById('deleteModal').style.display = 'none';  
}  
  
// Update hidden input when checkbox changes  
document.getElementById('notifyUser').addEventListener('change', function() {  
    document.getElementById('notifyUserInput').value = this.checked ? 'true' : 'false';  
});  
  
// Close modal when clicking outside  
window.onclick = function(event) {  
    const modal = document.getElementById('deleteModal');  
    if (event.target === modal) {  
        closeModal();  
    }  
}  
</script>  
{% endblock %}  
  
{% block content %}  
<div class="form-container">
```

```

<form method="get" action="{% url 'django_admin:user_management' %}"
class="search-form">

    <div class="form-group" style="display: flex; gap: 1rem;">

        <div style="flex: 1;">
            <label for="search" class="form-label">Search Users</label>
            <input type="text" id="search" name="search" class="form-input"
placeholder="Search by name, email, or username" value="{{ request.GET.search }}">
        </div>

        <div style="width: 200px;">
            <label for="user_type" class="form-label">User Type</label>
            <select id="user_type" name="user_type" class="form-select">
                <option value="">All Users</option>
                <option value="vet" {% if request.GET.user_type == 'vet' %}selected{%
endif %}>Veterinarians</option>
                <option value="pet_owner" {% if request.GET.user_type == 'pet_owner' %
}selected{%
endif %}>Pet Owners</option>
            </select>
        </div>
        <div style="align-self: flex-end;">
            <button type="submit" class="btn btn-primary">Search</button>
        </div>
    </div>
</form>

</div>

<div class="table-container">
    <table class="table">
        <thead>
            <tr>

```

```

<th>Name</th>
<th>Username</th>
<th>Email</th>
<th>User Type</th>
<th>Joined</th>
<th>Status</th>
<th>Actions</th>

</tr>
</thead>
<tbody>
  {% for user in users %}
    <tr>
      <td>{{ user.full_name|default:"-" }}</td>
      <td>{{ user.username }}</td>
      <td>{{ user.email }}</td>
      <td>{{ user.get_user_type_display }}</td>
      <td>{{ user.date_joined|date:"M d, Y" }}</td>
      <td>
        {% if user.status_verification %}
          <span class="badge badge-success">Verified</span>
        {% else %}
          <span class="badge badge-warning">Unverified</span>
        {% endif %}
      </td>
      <td>
        <div class="table-actions">
          <a href="{% url 'django_admin:view_user' user.id %}" class="btn btn-primary btn-sm">View</a>
        </div>
      </td>
    </tr>
  {% endfor %}
</tbody>

```

```

        <!-- <button type="button" class="btn btn-danger btn-sm"
        onclick="confirmDelete('{{ user.id }}', '{{ user.username }}')>Delete</button> -->
    </div>

    </td>
</tr>
{% empty %}
<tr>
    <td colspan="7" class="text-center">No users found.</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>

{% if users.has_other_pages %}
<div class="pagination">
    {% if users.has_previous %}
        <div class="pagination-item">
            <a href="?page=1{{ if request.GET.search }}&search={{ request.GET.search
}}{% endif %}{% if request.GET.user_type %}&user_type={{ request.GET.user_type
}}{% endif %}" class="pagination-link">
                &laquo;
            </a>
        </div>
        <div class="pagination-item">
            <a href="?page={{ users.previous_page_number }}{{ if request.GET.search
}}&search={{ request.GET.search }}{% endif %}{% if request.GET.user_type
%}&user_type={{ request.GET.user_type }}{% endif %}" class="pagination-link">
                &lsaquo;
            </a>
        </div>
    {% endif %}
</div>

```

```

</div>
{% endif %}

{% for num in users.paginator.page_range %}
    {% if users.number == num %}
        <div class="pagination-item">
            <a class="pagination-link active">{{ num }}</a>
        </div>
    {% elif num > users.number|add:'-3' and num < users.number|add:'3' %}
        <div class="pagination-item">
            <a href="?page={{ num }}{{ if request.GET.search }}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.user_type }}&user_type={{ request.GET.user_type }}{{ endif }}" class="pagination-link">{{ num }}</a>
        </div>
    {% endif %}
    {% endfor %}

    {% if users.has_next %}
        <div class="pagination-item">
            <a href="?page={{ users.next_page_number }}{{ if request.GET.search }}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.user_type }}&user_type={{ request.GET.user_type }}{{ endif }}" class="pagination-link">
                &rsaquo;
            </a>
        </div>
        <div class="pagination-item">
            <a href="?page={{ users.paginator.num_pages }}{{ if request.GET.search }}&search={{ request.GET.search }}{{ endif }}{{ if request.GET.user_type }}&user_type={{ request.GET.user_type }}{{ endif }}" class="pagination-link">
                &raquo;
        </a>
    </div>
    
```

```

    </a>
</div>
{%
  endif %}
</div>
{%
  endif %}

```

7.11.4.2 VIEWS

```

from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from django.utils.timezone import now, timedelta
from django.core.paginator import Paginator
from django.db.models import Count, Q
from django.core.mail import send_mail
from django.conf import settings
import json

from django.core.serializers.json import DjangoJSONEncoder
from authUser.models import User, VetProfile, PetOwnerProfile
from coreFunctions.models import Post, Category, Comment, ReplyComment
from .decorators import admin_required

```

```

def admin_login(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')
        user = authenticate(request, email=email, password=password)
        if user is not None and user.is_staff:
            login(request, user)
            messages.success(request, "Login successful!")

```

```

        return redirect('django_admin:admin_dashboard')
else:
    messages.error(request, "Invalid email or password or insufficient permissions")
return render(request, 'django_admin/login.html')

def admin_logout(request):
    logout(request)
    messages.success(request, "You have been logged out!")
    return redirect('django_admin:admin_login')

@admin_required
def admin_dashboard(request):
    # Get counts for dashboard stats
    total_users = User.objects.count()
    total_vets = User.objects.filter(user_type='vet').count()
    total_pet_owners = User.objects.filter(user_type='pet_owner').count()
    total_posts = Post.objects.count()

    # Get pending vet approvals
    pending_vets = VetProfile.objects.filter(
        user__status_verification=False,
        user__profile_completed=True,
        user__user_type='vet'
    )[:5] # Limit to 5 for dashboard

    # Get recent posts
    recent_posts = Post.objects.all().order_by('-date')[:5]

```

```

# Get post counts by category for chart - modified to ensure proper JSON
serialization

categories = Category.objects.annotate(post_count=Count('post')).order_by('-
post_count')

category_data = [
    {'name': category.name, 'count': category.post_count}
    for category in categories
]

return render(request, 'django_admin/dashboard.html', {
    'total_users': total_users,
    'total_vets': total_vets,
    'total_pet_owners': total_pet_owners,
    'total_posts': total_posts,
    'pending_vets': pending_vets,
    'recent_posts': recent_posts,
    'category_data': json.dumps(category_data, cls=DjangoJSONEncoder), # Single
JSON object
    'admin_user': request.user,
    'active_page': 'dashboard'
})

@admin_required
def user_management(request):
    search_query = request.GET.get('search', "")
    user_type = request.GET.get('user_type', "")

    # Filter users based on search and user type
    users = User.objects.all()

```

```

if search_query:
    users = users.filter(
        Q(username__icontains=search_query) |
        Q(email__icontains=search_query) |
        Q(full_name__icontains=search_query)
    )

if user_type:
    users = users.filter(user_type=user_type)

# Paginate results
paginator = Paginator(users.order_by('-date_joined'), 10)
page = request.GET.get('page', 1)
users = paginator.get_page(page)

return render(request, 'django_admin/user_management.html', {
    'users': users,
    'admin_user': request.user,
    'active_page': 'users'
})

@admin_required
def view_user(request, user_id):
    user = get_object_or_404(User, id=user_id)

    # Get user profile based on user type
    profile = None

```

```

if user.user_type == 'vet':
    profile = VetProfile.objects.filter(user=user).first()
elif user.user_type == 'pet_owner':
    profile = PetOwnerProfile.objects.filter(user=user).first()

# Get user posts
posts = Post.objects.filter(user=user).order_by('-date')

return render(request, 'django_admin/view_user.html', {
    'user_obj': user,
    'profile': profile,
    'posts': posts,
    'admin_user': request.user,
    'active_page': 'users'
})

@admin_required
def delete_user(request, user_id):
    user = get_object_or_404(User, id=user_id)

    if request.method == 'POST':
        # Send email notification
        try:
            send_mail(
                'Your PetVet Account Has Been Deleted',
                f'Dear {user.full_name or user.username},\n\nYour account on PetVet has
been deleted by an administrator. If you believe this is an error, please contact our
support team.\n\nBest regards,\nThe PetVet Team',

```

```

        settings.DEFAULT_FROM_EMAIL,
        [user.email],
        fail_silently=False,
    )
except Exception as e:
    messages.warning(request, f"User deleted but email notification failed: {str(e)}")

# Delete the user
username = user.username
user.delete()
messages.success(request, f"User '{username}' has been deleted successfully.")

return redirect('django_admin:user_management')

return redirect('django_admin:view_user', user_id=user_id)

@admin_required
def post_management(request):
    search_query = request.GET.get('search', "")
    category_id = request.GET.get('category', "")

    # Filter posts based on search and category
    posts = Post.objects.all()

    if search_query:
        posts = posts.filter(
            Q(title__icontains=search_query) |
            Q(body__icontains=search_query)

```

```
)  
  
if category_id:  
    posts = posts.filter(category_id=category_id)  
  
# Get all categories for filter dropdown  
categories = Category.objects.all()  
  
# Paginate results  
paginator = Paginator(posts.order_by('-date'), 10)  
page = request.GET.get('page', 1)  
posts = paginator.get_page(page)  
  
return render(request, 'django_admin/post_management.html', {  
    'posts': posts,  
    'categories': categories,  
    'admin_user': request.user,  
    'active_page': 'posts'  
})
```

```
@admin_required
```

```
def view_post(request, post_id):  
    post = get_object_or_404(Post, id=post_id)  
    comments = Comment.objects.filter(post=post).order_by('-date')  
  
    return render(request, 'django_admin/view_post.html', {  
        'post': post,  
        'comments': comments,
```

```

    'admin_user': request.user,
    'active_page': 'posts'
})

@admin_required
def delete_post(request, post_id):
    post = get_object_or_404(Post, id=post_id)

    if request.method == 'POST':
        # Check if we should notify the user
        notify_user = request.POST.get('notify_user') == 'true'

        if notify_user:
            try:
                send_mail(
                    'Your Post Has Been Removed',
                    f'Dear {post.user.full_name or post.user.username},\n\nYour post\n"{post.title}" has been removed by an administrator for violating our community guidelines. If you believe this is an error, please contact our support team.\n\nBest regards,\nThe PetVet Team',
                    settings.DEFAULT_FROM_EMAIL,
                    [post.user.email],
                    fail_silently=False,
                )
            except Exception as e:
                messages.warning(request, f"Post deleted but email notification failed: {str(e)}")

        # Delete the post

```

```
post_title = post.title
post.delete()
messages.success(request, f"Post '{post_title}' has been deleted successfully.")

return redirect('django_admin:post_management')

return redirect('django_admin:view_post', post_id=post_id)

@admin_required
def toggle_post_status(request, post_id):
    post = get_object_or_404(Post, id=post_id)

    if request.method == 'POST':
        # Toggle the active status
        post.active = not post.active
        post.save()

    status = "activated" if post.active else "deactivated"
    messages.success(request, f"Post '{post.title}' has been {status} successfully.")

    # Notify the user
    try:
        if not post.active: # Only notify when deactivating
            send_mail(
                'Your Post Has Been Deactivated',
```

```

f'Dear {post.user.full_name or post.user.username},\n\nYour post
"{post.title}" has been deactivated by an administrator. If you believe this is an error,
please contact our support team.\n\nBest regards,\nThe PetVet Team',

settings.DEFAULT_FROM_EMAIL,
[post.user.email],
fail_silently=False,
)

except Exception as e:
    messages.warning(request, f"Post status changed but email notification failed:
{str(e)}")

```

return redirect('django_admin:view_post', post_id=post_id)

@admin_required

```

def delete_comment(request, comment_id):
    comment = get_object_or_404(Comment, id=comment_id)
    post_id = comment.post.id

```

if request.method == 'POST':

```

        comment.delete()
        messages.success(request, "Comment has been deleted successfully.")

```

return redirect('django_admin:view_post', post_id=post_id)

@admin_required

```

def delete_reply(request, reply_id):
    reply = get_object_or_404(ReplyComment, id=reply_id)
    post_id = reply.comment.post.id

```

```
if request.method == 'POST':
    reply.delete()
    messages.success(request, "Reply has been deleted successfully.")

return redirect('django_admin:view_post', post_id=post_id)

@admin_required
def vet_approvals(request):
    # Get pending vet approvals
    pending_vets = VetProfile.objects.filter(
        user__status_verification=False,
        user__profile_completed=True,
        user__user_type='vet'
    )

    # Get recently approved vets (approved in the last 7 days)
    recently_approved_vets = VetProfile.objects.filter(
        user__status_verification=True,
        status_change__gte=now() - timedelta(days=7)
    )

    # Get recently declined vets (declined in the last 7 days)
    recently_declined_vets = VetProfile.objects.filter(
        user__status_verification=False,
        user__profile_completed=False,
        status_change__gte=now() - timedelta(days=7)
    )
```

```

return render(request, 'django_admin/vet_approvals.html', {
    'pending_vets': pending_vets,
    'recently_approved_vets': recently_approved_vets,
    'recently_declined_vets': recently_declined_vets,
    'admin_user': request.user,
    'active_page': 'vet_approvals'
})

```

@admin_required

```
def view_vet(request, vet_id):
```

```
    vet_profile = get_object_or_404(VetProfile, id=vet_id)
```

```
    return render(request, 'django_admin/view_vet.html', {
```

```
        'vet': vet_profile,
```

```
        'admin_user': request.user,
```

```
        'active_page': 'vet_approvals'
```

```
    })
```

#tested

@admin_required

```
def approve_vet(request, vet_id):
```

```
    vet_profile = get_object_or_404(VetProfile, id=vet_id)
```

```
    vet_user = vet_profile.user
```

```
    vet_user.status_verification = True
```

```
    vet_profile.verified = True
```

```
    vet_profile.status_change = now()
```

```

vet_user.save()
vet_profile.save()

# Send email notification
try:
    send_mail(
        'Your Veterinarian Account Has Been Approved',
        f'Dear {vet_user.full_name or vet_user.username},\n\nCongratulations! Your
veterinarian account on PetVet has been approved. You can now access all
veterinarian features on the platform.\n\nBest regards,\nThe PetVet Team',
        settings.DEFAULT_FROM_EMAIL,
        [vet_user.email],
        fail_silently=False,
    )
except Exception as e:
    messages.warning(request, f"Vet approved but email notification failed: {str(e)}")

messages.success(request, f"Veterinarian '{vet_user.full_name or
vet_user.username}' has been approved successfully.")
return redirect('django_admin:vet_approvals')

@admin_required
def decline_vet(request, vet_id):
    vet_profile = get_object_or_404(VetProfile, id=vet_id)
    vet_user = vet_profile.user

    vet_user.profile_completed = False
    vet_profile.status_change = now()

```

```
vet_user.save()
vet_profile.save()

# Prepare email details
subject = 'Your Veterinarian Account Application'

message = f'Dear {vet_user.full_name or vet_user.username},\n\nWe regret to inform you that your application to register as a veterinarian on PetVet has been declined. This may be due to incomplete or incorrect information provided. Please update your profile with accurate information and try again.\n\nBest regards,\nThe PetVet Team'

recipient = [vet_user.email]

# Send email notification
try:
    send_mail(
        subject,
        message,
        settings.DEFAULT_FROM_EMAIL,
        recipient,
        fail_silently=False,
    )
except Exception as e:
    messages.warning(request, f"Vet declined but email notification failed: {str(e)}")

    messages.success(request, f"Veterinarian '{vet_user.full_name or vet_user.username}' has been declined.")
    return redirect('django_admin:vet_approvals')
```

```
@admin_required
def category_management(request):
    # Get all categories with post count
    categories = Category.objects.annotate(post_count=Count('post'))

    return render(request, 'django_admin/category_management.html', {
        'categories': categories,
        'admin_user': request.user,
        'active_page': 'categories'
    })
```

```
@admin_required
def add_category(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        description = request.POST.get('description')

        if Category.objects.filter(name=name).exists():
            messages.error(request, f"Category '{name}' already exists.")
        else:
            category = Category.objects.create(name=name, description=description)
            messages.success(request, f"Category '{name}' has been added successfully.")

    return redirect('django_admin:category_management')
```

```
@admin_required
```

```
def edit_category(request, category_id):
    category = get_object_or_404(Category, id=category_id)

    if request.method == 'POST':
        name = request.POST.get('name')
        description = request.POST.get('description')

        # Check if name already exists for another category
        if Category.objects.filter(name=name).exclude(id=category_id).exists():
            messages.error(request, f"Category '{name}' already exists.")

        else:
            category.name = name
            category.description = description
            category.save()

            messages.success(request, f"Category '{name}' has been updated
successfully.")

    return redirect('django_admin:category_management')
```

```
@admin_required
```

```
def delete_category(request, category_id):
    category = get_object_or_404(Category, id=category_id)

    if request.method == 'POST':
        name = category.name

        # Update posts to remove category reference
```

```
Post.objects.filter(category=category).update(category=None)

category.delete()
messages.success(request, f"Category '{name}' has been deleted successfully.")

return redirect('django_admin:category_management')
```

7.11.4.3 Decorator

```
from django.shortcuts import redirect
from django.contrib import messages
```

```
def admin_required(view_func):
    def wrapper(request, *args, **kwargs):
        if not request.user.is_authenticated or not request.user.is_staff:
            messages.error(request, "You don't have permission to access this page.")
            return redirect('django_admin:admin_login')
        return view_func(request, *args, **kwargs)
    return wrapper
```

7.11.5 Authentication

7.11.5.1 HTML

```
<!DOCTYPE html>

<html lang="en">
  <head>
    {% load static %}

    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PetVet - Login & Register</title>
    <link rel="icon" href="{% static 'assets/images/paw.png' %}?v=2" type="image/png">
    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css">
  <style>
    :root {
      --primary: #4CAF50;
      --primary-dark: #388E3C;
      --primary-light: #A5D6A7;
      --secondary: #81C784;
      --accent: #1B5E20;
      --text-dark: #263238;
      --text-light: #78909C;
      --white: #FFFFFF;
      --error: #F44336;
      --success: #4CAF50;
      --green-text: #2E7D32;
    }
  </style>
```

```
* {  
    margin: 0;  
    padding: 0;  
    box-sizing: border-box;  
}  
  
body {  
    font-family: 'Poppins', sans-serif;  
    color: var(--text-dark);  
    line-height: 1.6;  
    background: linear-gradient(-45deg, #f5f7fa, #e8f5e9, #c8e6c9, #f5f7fa);  
    background-size: 400% 400%;  
    animation: gradient-animation 15s ease infinite;  
    position: relative;  
    overflow-x: hidden;  
}  
  
@keyframes gradient-animation {  
    0% {  
        background-position: 0% 50%;  
    }  
    50% {  
        background-position: 100% 50%;  
    }  
    100% {  
        background-position: 0% 50%;  
    }  
}
```

```
}

/* Add decorative elements */

body::before,
body::after {
    content: "";
    position: absolute;
    width: 300px;
    height: 300px;
    border-radius: 50%;
    z-index: -1;
}

.alert {
    transition: opacity 0.3s ease, transform 0.3s ease;
    transform-origin: top;
    position: relative;
}

.alert.hide {
    opacity: 0;
    transform: scaleY(0);
}

body::before {
    top: -100px;
    right: -100px;
```

```
background: radial-gradient(circle, rgba(165, 214, 167, 0.4) 0%, rgba(165, 214, 167, 0) 70%);  
}  
  
body::after {  
    bottom: -100px;  
    left: -100px;  
    background: radial-gradient(circle, rgba(129, 199, 132, 0.4) 0%, rgba(129, 199, 132, 0) 70%);  
}  
  
.background-pattern {  
    position: fixed;  
    top: 0;  
    left: 0;  
    width: 100%;  
    height: 100%;  
    pointer-events: none;  
    z-index: -2;  
    opacity: 0.05;  
    background-image: url("data:image/svg+xml,%3Csvg  
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 512 512'  
fill='%234CAF50'%3E%3Cpath d='M256 224c-79.41 0-192 122.76-192 200.25 0 34.9  
26.81 55.75 71.74 55.75 48.84 0 81.09-25.08 120.26-25.08 39.51 0 71.85 25.08 120.26  
25.08 44.93 0 71.74-20.85 71.74-55.75C448 346.76 335.41 224 256 224zm-147.28-  
12.61c-10.4-34.65-42.44-57.09-71.56-50.13-29.12 6.96-44.29 40.69-33.89 75.34 10.4  
34.65 42.44 57.09 71.56 50.13 29.12-6.96 44.29-40.69 33.89-75.34zm84.72-  
20.78c30.94-8.14 46.42-49.94 34.58-93.36s-46.52-72.01-77.46-63.87-46.42 49.94-  
34.58 93.36c11.84 43.42 46.53 72.02 77.46 63.87zm281.39-29.34c-29.12-6.96-61.15  
15.48-71.56 50.13-10.4 34.65 4.77 68.38 33.89 75.34 29.12 6.96 61.15-15.48 71.56-  
50.13 10.4-34.65-4.77-68.38-33.89-75.34zm-156.27 29.34c30.94 8.14 65.62-20.45  
77.46-63.87 11.84-43.42-3.64-85.21-34.58-93.36s-65.62 20.45-77.46 63.87c-11.84  
43.42 3.64 85.22 34.58 93.36z'%3E%3C/svg%3E");
```

```
background-repeat: repeat;  
background-size: 100px 100px;  
}
```

```
.container {  
    display: flex;  
    min-height: 100vh;  
    align-items: center;  
    justify-content: center;  
    padding: 20px;  
}
```

```
.card {  
    width: 100%;  
    max-width: 900px;  
    background: var(--white);  
    border-radius: 20px;  
    box-shadow: 0 15px 50px rgba(0, 0, 0, 0.1), 0 0 0 1px rgba(0, 0, 0, 0.05);  
    overflow: hidden;  
    display: flex;  
    flex-direction: row;  
    position: relative;  
    z-index: 1;  
    transition: box-shadow 0.3s ease;  
}
```

```
.card:hover {
```

```
    box-shadow: 0 20px 60px rgba(0, 0, 0, 0.15), 0 0 0 1px rgba(0, 0, 0, 0.05), 0 0 0  
5px rgba(76, 175, 80, 0.1);
```

```
}
```

```
.left-panel {  
flex: 1;  
background: linear-gradient(135deg, var(--primary), var(--primary-dark));  
padding: 40px;  
color: var(--white);  
display: flex;  
flex-direction: column;  
justify-content: center;  
position: relative;  
overflow: hidden;  
}
```

```
.home-btn {  
position: absolute;  
top: 20px;  
left: 20px;  
display: flex;  
align-items: center;  
padding: 8px 15px;  
background-color: rgba(255, 255, 255, 0.2);  
color: var(--white);  
border-radius: 8px;  
text-decoration: none;  
font-weight: 500;
```

```
transition: all 0.3s ease;  
z-index: 10;  
}  
  
.home-btn:hover {  
background-color: rgba(255, 255, 255, 0.3);  
transform: translateY(-2px);  
}  
  
.home-btn i {  
margin-right: 8px;  
}  
  
.left-panel h1 {  
font-size: 2.5rem;  
margin-bottom: 20px;  
position: relative;  
z-index: 2;  
}  
  
.left-panel p {  
font-size: 1.1rem;  
margin-bottom: 30px;  
opacity: 0.9;  
position: relative;  
z-index: 2;  
}
```

```

.pet-illustrations {
    position: absolute;
    bottom: -50px;
    right: -50px;
    width: 300px;
    height: 300px;

    background-image: url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 512 512' fill='%23ffffff'
opacity='0.2'%3E%3Cpath d='M256 224c-79.41 0-192 122.76-192 200.25 0 34.9 26.81
55.75 71.74 55.75 48.84 0 81.09-25.08 120.26-25.08 39.51 0 71.85 25.08 120.26 25.08
44.93 0 71.74-20.85 71.74-55.75C448 346.76 335.41 224 256 224zm-147.28-12.61c-
10.4-34.65-42.44-57.09-71.56-50.13-29.12 6.96-44.29 40.69-33.89 75.34 10.4 34.65
42.44 57.09 71.56 50.13 29.12-6.96 44.29-40.69 33.89-75.34zm84.72-20.78c30.94-
8.14 46.42-49.94 34.58-93.36s-46.52-72.01-77.46-63.87-46.42 49.94-34.58
93.36c11.84 43.42 46.53 72.02 77.46 63.87zm281.39-29.34c-29.12-6.96-61.15 15.48-
71.56 50.13-10.4 34.65 4.77 68.38 33.89 75.34 29.12 6.96 61.15-15.48 71.56-50.13
10.4-34.65-4.77-68.38-33.89-75.34zm-156.27 29.34c30.94 8.14 65.62-20.45 77.46-
63.87 11.84-43.42-3.64-85.21-34.58-93.36s-65.62 20.45-77.46 63.87c-11.84 43.42 3.64
85.22 34.58 93.36z'/%3E%3C/svg%3E");

    background-repeat: no-repeat;
    background-position: center;
    z-index: 1;
}

.right-panel {
    flex: 1.2;
    padding: 40px;
}

.tabs {
    display: flex;
    margin-bottom: 30px;
}

```

```
border-bottom: 2px solid #eee;
}

.tab {
  padding: 10px 20px;
  font-size: 1.1rem;
  font-weight: 500;
  cursor: pointer;
  position: relative;
  color: var(--text-light);
  transition: all 0.3s ease;
}

.tab.active {
  color: var(--primary);
}

.tab.active::after {
  content: "";
  position: absolute;
  bottom: -2px;
  left: 0;
  width: 100%;
  height: 3px;
  background-color: var(--primary);
}

.form-container {
```

```
        display: none;  
    }  
  
.form-container.active {  
    display: block;  
}  
  
.form-group {  
    margin-bottom: 20px;  
}  
  
.form-group label {  
    display: block;  
    margin-bottom: 8px;  
    font-weight: 500;  
    color: var(--text-dark);  
}  
  
.form-control {  
    width: 100%;  
    padding: 12px 15px;  
    font-size: 1rem;  
    border: 1px solid #ddd;  
    border-radius: 8px;  
    transition: border-color 0.3s;  
    font-family: 'Poppins', sans-serif;  
}
```

```
.form-control:focus {  
    border-color: var(--primary);  
    outline: none;  
}  
  
.form-control.error {  
    border-color: var(--error);  
}  
  
.error-message {  
    color: var(--error);  
    font-size: 0.85rem;  
    margin-top: 5px;  
}  
  
.btn {  
    display: inline-block;  
    padding: 12px 24px;  
    background-color: var(--primary);  
    color: white;  
    border: none;  
    border-radius: 8px;  
    font-size: 1rem;  
    font-weight: 500;  
    cursor: pointer;  
    transition: background-color 0.3s;  
    font-family: 'Poppins', sans-serif;  
}
```

```
.btn:hover {  
    background-color: var(--primary-dark);  
}
```

```
.form-footer {  
    margin-top: 20px;  
    text-align: center;  
    color: var(--text-light);  
}
```

```
.form-footer a {  
    color: var(--primary);  
    text-decoration: none;  
    font-weight: 500;  
}
```

```
.form-footer a:hover {  
    text-decoration: underline;  
}
```

```
.forgot-password {  
    text-align: right;  
    margin-top: 8px;  
}
```

```
.forgot-password a {  
    color: var(--primary);  
}
```

```
text-decoration: none;  
font-size: 0.9rem;  
}  
  
.forgot-password a:hover {  
    text-decoration: underline;  
}  
  
.alert {  
    padding: 12px 15px;  
    margin-bottom: 20px;  
    border-radius: 8px;  
    font-size: 0.9rem;  
}  
  
.alert-danger {  
    background-color: #FFEBEE;  
    color: var(--error);  
    border: 1px solid #FFCDD2;  
}  
  
.alert-success {  
    background-color: #E8F5E9;  
    color: var(--success);  
    border: 1px solid #C8E6C9;  
}  
  
.form-row {
```

```
display: flex;  
gap: 15px;  
margin-bottom: 20px;  
}  
  
.form-row .form-group {  
flex: 1;  
margin-bottom: 0;  
}  
  
.select-wrapper {  
position: relative;  
}  
  
.select-wrapper::after {  
content: '▼';  
font-size: 0.8rem;  
position: absolute;  
right: 15px;  
top: 50%;  
transform: translateY(-50%);  
pointer-events: none;  
color: var(--text-light);  
}  
  
select.form-control {  
appearance: none;  
padding-right: 30px;
```

```
}
```

```
.logo {  
    display: flex;  
    align-items: center;  
    margin-bottom: 20px;  
    position: relative;  
    z-index: 2;  
}
```

```
.logo-img {  
    height: 40px;  
    width: auto;  
    margin-right: 10px;  
}
```

```
.logo-text {  
    font-size: 1.5rem;  
    font-weight: 700;  
    color: #000000; /* Black color for "Pet" */  
}
```

```
.green-text {  
    color: #2E7D32; /* Brighter shade of #1B5E20 */  
}
```

```
/* Responsive styles */
```

```
@media (max-width: 768px) {
```

```
.card {  
    flex-direction: column;  
}  
  
.left-panel {  
    padding: 30px;  
    min-height: 200px;  
}  
  
.right-panel {  
    padding: 30px;  
}  
  
.form-row {  
    flex-direction: column;  
    gap: 20px;  
}  
  
.home-btn {  
    top: 10px;  
    left: 10px;  
    padding: 6px 12px;  
    font-size: 0.9rem;  
}  
}  
</style>  
</head>  
<body>
```

```

<div class="background-pattern"></div>
<div class="container">
  <div class="card">
    <div class="left-panel">
      <a href="{% url 'coreFunctions:home' %}" class="home-btn">
        <i class="fas fa-home"></i> Home
      </a>
      <div class="logo">
        
        <div class="logo-text">Pet<span class="green-text">Vet</span></div>
      </div>
      <h1>Welcome to Pet<span class="green-text">Vet</span></h1>
      <p>Connect with vets, schedule appointments, and care for your furry friends.</p>
      <div class="pet-illustrations"></div>
    </div>
    <div class="right-panel">
      <div class="tabs">
        <div class="tab active" data-tab="login">Login</div>
        <div class="tab" data-tab="register">Register</div>
      </div>
    </div>
  <% if messages %>
    <% for message in messages %>
      <div class="alert {% if message.tags == 'error' %}alert-danger{% elif message.tags == 'success' %}alert-success{% endif %}">
        {{ message }}
      </div>
    <% endfor %>
  <% endif %>
</div>

```

```
{% endfor %}

{% endif %}

<!-- Login form -->

<div class="form-container active" id="login-form">

    <form method="POST" action="{% url 'authUser:loginUser' %}">

        {% csrf_token %}

        <div class="form-group">

            <label for="email">Email</label>

            <input type="email" class="form-control" id="email" name="email"
placeholder="Enter your email">

        </div>

        <div class="form-group">

            <label for="password">Password</label>

            <input type="password" class="form-control" id="password"
name="password" placeholder="Enter your password">

        <div class="forgot-password">

            <a href="{% url 'authUser:forgot_password' %}">Forgot
Password?</a>

        </div>

        </div>

        <button type="submit" class="btn">Login</button>

        <div class="form-footer">

            <p>Don't have an account? <a href="#" class="tab-link" data-
tab="register">Register here</a></p>

        </div>

    </form>

</div>
```

```

<!-- Registration form -->

<div class="form-container" id="register-form">
    <form method="POST" action="{% url 'authUser:register' %}">
        {% csrf_token %}
        <div class="form-group">
            <label for="full_name">Full Name</label>
            <input type="text" class="form-control" {% if errors.full_name %}error{% endif %}" id="full_name" name="full_name" placeholder="Enter your full name">
            {% if errors.full_name %}<div class="error-message">{{ errors.full_name }}</div>{% endif %}
        </div>
        <div class="form-row">
            <div class="form-group">
                <label for="email">Email</label>
                <input type="email" class="form-control" {% if errors.email %}error{% endif %}" id="email" name="email" placeholder="Enter your email">
                {% if errors.email %}<div class="error-message">{{ errors.email }}</div>{% endif %}
            </div>
            <div class="form-group">
                <label for="username">Username</label>
                <input type="text" class="form-control" {% if errors.username %}error{% endif %}" id="username" name="username" placeholder="Choose a username">
                {% if errors.username %}<div class="error-message">{{ errors.username }}</div>{% endif %}
            </div>
        </div>
        <div class="form-row">
            <div class="form-group">

```

```

        <label for="phone">Phone</label>
        <input type="tel" class="form-control" {% if errors.phone %}error{% endif %}" id="phone" name="phone" placeholder="Enter your phone number">
        {% if errors.phone %}<div class="error-message">{{ errors.phone }}</div>{% endif %}
    </div>

    <div class="form-group">
        <label for="gender">Gender</label>
        <div class="select-wrapper">
            <select class="form-control" {% if errors.gender %}error{% endif %}" id="gender" name="gender">
                <option value="" selected disabled>Select gender</option>
                <option value="M">Male</option>
                <option value="F">Female</option>
            </select>
            {% if errors.gender %}<div class="error-message">{{ errors.gender }}</div>{% endif %}
        </div>
    </div>
    <div class="form-group">
        <label for="user_type">User Type</label>
        <div class="select-wrapper">
            <select class="form-control" {% if errors.user_type %}error{% endif %}" id="user_type" name="user_type">
                <option value="" selected disabled>Select user type</option>
                <option value="pet_owner">Pet Owner</option>
                <option value="vet">Veterinarian</option>
            </select>
        </div>
    </div>

```

```
{% if errors.user_type %}<div class="error-message">{{  
errors.user_type }}</div>{% endif %}  
    </div>  
  </div>  
  <div class="form-row">  
    <div class="form-group">  
      <label for="password1">Password</label>  
      <input type="password" class="form-control" {{ if errors.password1  
}}error{{ endif }} id="password1" name="password1" placeholder="Create a  
password">  
        <% if errors.password1 %><div class="error-message">{{  
errors.password1 }}</div>{% endif %}  
      </div>  
    <div class="form-group">  
      <label for="password2">Confirm Password</label>  
      <input type="password" class="form-control" {{ if errors.password2  
}}error{{ endif }} id="password2" name="password2" placeholder="Confirm your  
password">  
        <% if errors.password2 %><div class="error-message">{{  
errors.password2 }}</div>{% endif %}  
      </div>  
    </div>  
    <button type="submit" class="btn">Register</button>  
  <div class="form-footer">  
    <p>Already have an account? <a href="#" class="tab-link" data-  
tab="login">Login here</a></p>  
  </div>  
 </form>  
</div>  
</div>
```

```
</div>

<script>

    // Auto-hide messages after 5 seconds
    document.addEventListener('DOMContentLoaded', function() {
        const alerts = document.querySelectorAll('.alert');

        alerts.forEach(alert => {
            setTimeout(() => {
                alert.style.opacity = '0';
                setTimeout(() => {
                    alert.style.display = 'none';
                }, 300); // Match this with your CSS transition time
            }, 5000); // 5 seconds
        });
    });

    // Allow manual closing of alerts
    alerts.forEach(alert => {
        const closeBtn = document.createElement('button');
        closeBtn.innerHTML = '&times;';
        closeBtn.style.cssText = `
            position: absolute;
            top: 5px;
            right: 10px;
            background: none;
            border: none;
            font-size: 1.2rem;
        `;
    });
})
```

```

    cursor: pointer;
    color: inherit;
};

closeBtn.addEventListener('click', () => {
    alert.style.opacity = '0';
    setTimeout(() => {
        alert.style.display = 'none';
    }, 300);
});

alert.style.position = 'relative';
alert.style.paddingRight = '30px';
alert.appendChild(closeBtn);

});

document.addEventListener('DOMContentLoaded', function() {
    // Check URL hash on page load
    if (window.location.hash === '#register') {
        switchTab('register');
    }

    // Get all tab elements
    const tabs = document.querySelectorAll('.tab');
    const tabLinks = document.querySelectorAll('.tab-link');
    const formContainers = document.querySelectorAll('.form-container');

    // Function to switch between tabs
    function switchTab(tabId) {

```

```

// Update active tab
tabs.forEach(tab => {
  if (tab.dataset.tab === tabId) {
    tab.classList.add('active');
  } else {
    tab.classList.remove('active');
  }
});

// Show active form
formContainers.forEach(form => {
  if (form.id === tabId + '-form') {
    form.classList.add('active');
  } else {
    form.classList.remove('active');
  }
});

// Update URL hash
window.location.hash = tabId;
}

// Tab click event
tabs.forEach(tab => {
  tab.addEventListener('click', function() {
    switchTab(this.dataset.tab);
  });
});

```

```
// Tab link click event (for "Register here" and "Login here" links)
tabLinks.forEach(link => {
    link.addEventListener('click', function(e) {
        e.preventDefault();
        switchTab(this.dataset.tab);

        // Scroll to top of form for better UX
        document.querySelector('.right-panel').scrollIntoView({
            behavior: 'smooth'
        });
    });
});

// Handle back/forward browser buttons
window.addEventListener('hashchange', function() {
    const hash = window.location.hash.substring(1);
    if (hash === 'login' || hash === 'register') {
        switchTab(hash);
    }
});
});

</script>
</body>
</html>
```

7.11.5.2 VIEWS

@login_required

```
def account_settings(request):
```

```
    """View for displaying and handling account settings."""

```

```
    return render(request, 'authUser/settings.html')
```

@login_required

```
def change_password(request):
```

```
    """View for changing user password."""

```

```
    if request.method == 'POST':
```

```
        form = PasswordChangeForm(request.user, request.POST)
```

```
        if form.is_valid():
```

```
            new_password = form.cleaned_data.get('new_password1')
```

```
            if not validate_password_strength(new_password):
```

```
                messages.error(request, 'Password must be at least 8 characters long,  
include at least one uppercase letter, one number, and one special character.')

```

```
                return redirect('authUser:account_settings')
```

```
            user = form.save()
```

```
            # Keep the user logged in after password change
```

```
            update_session_auth_hash(request, user)
```

```
            messages.success(request, 'Your password was successfully updated!')
```

```
            return redirect('authUser:account_settings')
```

```
        else:
```

```
            for error in form.errors.values():
```

```
                messages.error(request, error)
```

```

return redirect('authUser:account_settings')

def validate_password_strength(password):
    """Validates that the password meets the required strength."""
    if (len(password) < 8 or
        not re.search(r'[A-Z]', password) or
        not re.search(r'\d', password) or
        not re.search(r'[@#$%^&*(),.?":{}|<>]', password)):
        return False
    return True

@login_required
def delete_account(request):
    """View for deleting user account."""
    if request.method == 'POST':
        email = request.POST.get('confirm_email')
        password = request.POST.get('confirm_password')

        # Verify user credentials
        if email == request.user.email:
            user = request.user
            if user.check_password(password):
                # Log the user out first
                logout(request)
                # Delete the user account
                user.delete()
                messages.success(request, 'Your account has been permanently deleted.')

```

```

        return redirect('authUser:loginUser')

    else:
        messages.error(request, 'Incorrect password. Account deletion failed.')

    else:
        messages.error(request, 'Email does not match your account. Deletion failed.')

    return redirect('authUser:account_settings')

def forgot_password(request):
    """View for initiating password reset process."""
    template_name = 'authUser/forgot_passwordunauth.html'
    if request.user.is_authenticated:
        template_name = 'authUser/forgot_password.html'

    if request.method == 'POST':
        email = request.POST.get('email')
        try:
            user = User.objects.get(email=email)
            # Generate token
            token = get_random_string(64)
            # Save token to user profile or a dedicated password reset model
            user.otp = token
            user.save()

            # Create reset link
            reset_link = request.build_absolute_uri(
                reverse('authUser:reset_password', kwargs={'token': token})

```

```

        )

    message = (
        f"Hello {user.full_name},\n\n"
        f"We received a request to reset your password.\n\n"
        f"Please click the link below to reset your password:\n"
        f"{reset_link}\n\n"
        f"If you did not request a password reset, please ignore this email.\n\n"
        f"Best regards,\n"
        f"The PetVet Team"
    )

    # Send email
    send_mail(
        'PetVet - Password Reset',
        message,
        settings.DEFAULT_FROM_EMAIL,
        [email],
        fail_silently=False,
    )

except User.DoesNotExist:
    # Do not reveal user existence for security
    pass

    messages.success(request, 'If your email exists in our system, you will receive a
password reset link.')

return render(request, template_name)

```

```
def reset_password(request, token):
    """View for resetting password using token."""
    try:
        user = User.objects.get(otp=token)

        if request.method == 'POST':
            password1 = request.POST.get('new_password1')
            password2 = request.POST.get('new_password2')

            if password1 != password2:
                messages.error(request, 'Passwords do not match.')
            elif not validate_password_strength(password1):
                messages.error(request, 'Password must be at least 8 characters long,
include at least one uppercase letter, one number, and one special character.')
            else:
                # Set new password
                user.set_password(password1)
                # Clear the token
                user.otp = None
                user.save()

                messages.success(request, 'Your password has been reset successfully. You
can now log in with your new password.')
                return redirect('authUser:loginUser')

    return render(request, 'authUser/reset_password.html')
```

```
except User.DoesNotExist:  
    messages.error(request, 'Invalid or expired password reset link.')  
    return redirect('authUser:loginUser')
```

```
def RegisterView(request):
    if request.user.is_authenticated:
        messages.warning(request, "You are already registered!")
        return redirect("coreFunctions:index")
```

```
if request.method == "POST":  
    # Extract data manually from request.POST  
    full_name = request.POST.get("full_name")  
    email = request.POST.get("email")  
    username = request.POST.get("username")  
    phone = request.POST.get("phone")  
    gender = request.POST.get("gender")  
    user_type = request.POST.get("user_type")  
    password1 = request.POST.get("password1")  
    password2 = request.POST.get("password2")
```

Basic validation

```
errors = {}
```

```
if not full_name:
```

```
errors["full_name"] = "Full name is required."
```

if not email:

```
    errors["email"] = "Email is required."  
    elif User.objects.filter(email=email).exists():  
        errors["email"] = "Email already exists."  
  
    if not username:  
        errors["username"] = "Username is required."  
    elif User.objects.filter(username=username).exists():  
        errors["username"] = "Username already exists."  
  
    if not phone:  
        errors["phone"] = "Phone number is required."  
  
    if not gender:  
        errors["gender"] = "Gender is required."  
  
    if not user_type:  
        errors["user_type"] = "User type is required."  
  
    if not password1:  
        errors["password1"] = "Password is required."  
  
    if not password2:  
        errors["password2"] = "Password confirmation is required."  
    elif password1 != password2:  
        errors["password2"] = "Passwords do not match."  
  
    # If there are errors, return them to the template  
    if errors:
```

```

for field, error in errors.items():
    messages.error(request, error)
return render(request, "authUser/register.html", {"errors": errors})

# Create user
user = User(
    full_name=full_name,
    email=email,
    username=username,
    phone=phone,
    gender=gender,
    user_type=user_type,
)
user.set_password(password1)
user.otp = str(random.randint(100000, 999999))
user.save()

# Send OTP email for pet_owner users
if user_type == "pet_owner":
    subject = "Your OTP for Verification"
    message = (
        f"Hello {full_name},\n\n"
        f"Thank you for registering with PetVet!\n\n"
        f"Your One-Time Password (OTP) for verification is: {user.otp}\n\n"
        f"Please enter this OTP to complete your registration.\n\n"
        f"If you did not request this, please ignore this message.\n\n"
        f"Best regards,\n"
        f"The PetVet Team"
    )

```

```
)  
    send_mail(  
        subject,  
        message,  
        settings.DEFAULT_FROM_EMAIL,  
        [email],  
        fail_silently=False,  
    )  
  
    messages.info(request, "An OTP has been sent to your email. Please verify to proceed.")  
  
user = authenticate(email=email, password=password1)  
  
if user is not None:  
    login(request, user)  
  
    # Create or update the user profile  
    if user_type == 'vet':  
        vet_profile, created = VetProfile.objects.get_or_create(user=user)  
        vet_profile.full_name = full_name  
        vet_profile.save()  
    elif user_type == 'pet_owner':  
        pet_profile, created = PetOwnerProfile.objects.get_or_create(user=user)  
        pet_profile.full_name = full_name  
        pet_profile.save()  
  
    messages.success(request, f"Hi {full_name}, your account was created successfully.")
```

```
# Redirect to complete-profile if the profile is incomplete
if not user.profile_completed:
    return redirect("complete-profile")

return redirect("coreFunctions:index")
else:
    messages.error(request, "There was an issue with your login. Please try again.")

return render(request, "authUser/register.html")

def LoginView(request):
    if request.user.is_authenticated:
        return redirect("coreFunctions:index")

    if request.method == "POST":
        email = request.POST.get("email")
        password = request.POST.get("password")

        user = authenticate(request, email=email, password=password)

    if user is not None:
        # If authentication is successful
        login(request, user)
```

```

# Redirect to complete-profile if the profile is incomplete
if not user.profile_completed:
    # print("redirect bhayo")
    return redirect("complete-profile")

return redirect("coreFunctions:index")

else:
    # If authentication fails
    messages.error(request, "Invalid username or password")
    return redirect("authUser:loginUser")

return render(request, "authUser/register.html")

```

```

def LogoutView(request):
    logout(request)
    messages.success(request, "You have successfully logged out.")
    return redirect("authUser:loginUser")

```

```

@login_required
def user_info(request, user_id=None):
    # If no user_id is provided, show the current user's profile
    if user_id is None:
        user = request.user
    else:
        user = get_object_or_404(User, id=user_id)

```

```
# Get the profile based on user type
if user.user_type == 'vet':
    profile = VetProfile.objects.get(user=user)
    profile_picture_field = 'vet_image'
else:
    profile = PetOwnerProfile.objects.get(user=user)
    profile_picture_field = 'human_image'

if request.method == 'POST' and request.user == user:
    # Handle profile update
    user.full_name = request.POST.get('full_name', user.full_name)
    user.phone = request.POST.get('phone', user.phone)
    user.gender = request.POST.get('gender', user.gender)

    if user.user_type == 'vet':
        profile.clinic_name = request.POST.get('clinic_name', profile.clinic_name)
        profile.specialization = request.POST.get('specialization', profile.specialization)
        profile.experience_years = request.POST.get('experience_years',
profile.experience_years)
        profile.license_number = request.POST.get('license_number',
profile.license_number)
        profile.summary = request.POST.get('summary', profile.summary)
    else:
        profile.bio = request.POST.get('bio', profile.bio)
        profile.pets_owned = request.POST.get('pets_owned', profile.pets_owned)

    profile.country = request.POST.get('country', profile.country)
```

```
profile.city = request.POST.get('city', profile.city)
profile.address = request.POST.get('address', profile.address)

# Handle profile picture upload
if 'profile_picture' in request.FILES:
    current_picture = getattr(profile, profile_picture_field)
    if current_picture:
        current_picture.delete()
    setattr(profile, profile_picture_field, request.FILES['profile_picture'])

user.save()
profile.save()
messages.success(request, 'Profile updated successfully!')
return redirect('authUser:user-info')

# Fetch posts created by the user
posts = Post.objects.filter(user=user)
comments = Comment.objects.filter(user=user)
reply_comments = ReplyComment.objects.filter(user=user)

context = {
    'user': user,
    'profile': profile,
    'profile_picture_field': profile_picture_field,
    'posts': posts,
    'comments': comments,
    'reply_comments': reply_comments,
    'is_own_profile': (request.user == user),
```

```

}

return render(request, 'authUser/user_info.html', context)

def complete_profile(request):
    if not request.user.is_authenticated:
        messages.error(request, "You need to log in to complete your profile.")
        return redirect('authUser:loginUser')

    try:
        if request.user.user_type == 'vet':
            profile = VetProfile.objects.get(user=request.user)
            profile_type = 'vet'
        elif request.user.user_type == 'pet_owner':
            profile = PetOwnerProfile.objects.get(user=request.user)
            profile_type = 'pet_owner'
        else:
            messages.error(request, "Invalid user type.")
            return redirect('coreFunctions:index')
    except (VetProfile.DoesNotExist, PetOwnerProfile.DoesNotExist):
        messages.error(request, "Profile not found. Please contact support.")
        return redirect('coreFunctions:index')

    errors = {}

    if request.method == 'POST':
        # Common fields for both profiles

```

```
country = request.POST.get('country', "").strip()
city = request.POST.get('city', "").strip()
address = request.POST.get('address', "").strip()
use_default_image = request.POST.get('use_default_image') == 'on'

# Validate common fields
if not country:
    errors['country'] = "Country is required."
if not city:
    errors['city'] = "City is required."
if len(address) > 500:
    errors['address'] = "Address cannot exceed 500 characters."

# Profile-specific validation
if profile_type == 'pet_owner':
    bio = request.POST.get('bio', "").strip()
    pets_owned = request.POST.get('pets_owned', '0').strip()

    try:
        pets_owned = int(pets_owned)
        if pets_owned < 0 or pets_owned > 20:
            errors['pets_owned'] = "Please enter a valid number between 0 and 20."
    except ValueError:
        errors['pets_owned'] = "Please enter a valid number."

    if len(bio) > 1000:
        errors['bio'] = "Bio cannot exceed 1000 characters."
```

```
elif profile_type == 'vet':
    summary = request.POST.get('summary', "").strip()
    clinic_name = request.POST.get('clinic_name', "").strip()
    specialization = request.POST.get('specialization', "").strip()
    experience_years = request.POST.get('experience_years', '0').strip()
    license_number = request.POST.get('license_number', "").strip()

if not clinic_name:
    errors['clinic_name'] = "Clinic name is required."
elif len(clinic_name) > 100:
    errors['clinic_name'] = "Clinic name cannot exceed 100 characters."

if not license_number:
    errors['license_number'] = "License number is required."
elif len(license_number) > 50:
    errors['license_number'] = "License number cannot exceed 50 characters."

try:
    experience_years = int(experience_years)
    if experience_years < 0 or experience_years > 60:
        errors['experience_years'] = "Please enter valid experience years (0-60)."
except ValueError:
    errors['experience_years'] = "Please enter a valid number."

if len(summary) > 2000:
    errors['summary'] = "Summary cannot exceed 2000 characters."
if len(specialization) > 100:
    errors['specialization'] = "Specialization cannot exceed 100 characters."
```

```
# Image validation

if not use_default_image:
    image_field = 'vet_image' if profile_type == 'vet' else 'human_image'
    uploaded_image = request.FILES.get(image_field)

if uploaded_image:
    if uploaded_image.size > 5 * 1024 * 1024: # 5MB limit
        errors[image_field] = "Image size cannot exceed 5MB."
    elif not uploaded_image.content_type.startswith('image/'):
        errors[image_field] = "Only image files are allowed."

if not errors:
    try:
        # Update profile based on user type
        if profile_type == 'pet_owner':
            profile.bio = bio
            profile.pets_owned = pets_owned
        else:
            profile.summary = summary
            profile.clinic_name = clinic_name
            profile.specialization = specialization
            profile.experience_years = experience_years
            profile.license_number = license_number

        # Update common fields
        profile.country = country
        profile.city = city
```

```

profile.address = address

# Handle profile image
if use_default_image:
    setattr(profile, f"{profile_type}_image", 'default.png')
elif request.FILES.get(f"{profile_type}_image"):
    setattr(profile, f"{profile_type}_image",
request.FILES.get(f"{profile_type}_image"))

profile.save()

# Mark profile as completed
request.user.profile_completed = True
request.user.save()

messages.success(request, "Profile successfully updated!")

if not request.user.status_verification:
    if request.user.user_type == "vet":
        return redirect('authUser:profile_verification_in_progress')
    else:
        return redirect('authUser:verify_otp')

return redirect('coreFunctions:index')

except Exception as e:
    messages.error(request, f"An error occurred while saving your profile:
{str(e)}")

```

```

# Log the error here if you have logging set up

# Prepare context for rendering the template
context = {
    'profile': profile,
    'profile_type': profile_type,
    'errors': errors,
    'messages': messages.get_messages(request)
}

return render(request, 'authUser/profile.html', context)

def profile_verification_in_progress(request):
    # Check if the user is a vet and their profile is completed but not verified yet
    if request.user.user_type == 'vet' and request.user.profile_completed and not request.user.status_verification:
        return render(request, 'authUser/profile_verification_in_progress.html')

    elif request.user.user_type == 'pet_owner' and request.user.profile_completed and not request.user.status_verification:
        return render(request, 'authUser/verify_otp.html')

    # If the profile is verified, redirect to the feed page
    elif request.user.user_type == 'vet' and request.user.status_verification:
        return redirect("coreFunctions:index")

    else:
        # If the user is not a vet or verification is not needed
        return redirect("coreFunctions:index")

```

```
def verify_otp(request):
    # If user is already verified, redirect them
    if request.user.status_verification:
        return redirect("coreFunctions:index")

    if request.method == 'POST':
        entered_otp = request.POST.get('otp')

        if not entered_otp:
            messages.error(request, "Please enter the OTP")
            return render(request, 'authUser/verify_otp.html')

    # Compare the entered OTP
    if request.user.otp == entered_otp:
        request.user.status_verification = True
        request.user.otp = None # Clear OTP after verification
        request.user.save()
        messages.success(request, "Your email has been verified successfully!")
        return redirect("coreFunctions:index")

    else:
        messages.error(request, "Invalid OTP. Please try again.")

# Handle GET request
return render(request, 'authUser/verify_otp.html')
```

@login_required

```
def add_pet(request):
    errors = {}

    if request.method == 'POST':
        # Get form data
        name = request.POST.get('name', "").strip()
        species = request.POST.get('species', "").strip()
        breed = request.POST.get('breed', "").strip()
        age = request.POST.get('age', "").strip()
        color = request.POST.get('color', "").strip()
        weight = request.POST.get('weight', "").strip()
        vaccination_status = request.POST.get('vaccination_status', "").strip()
        allergies = request.POST.get('allergies', "").strip()
        medical_history = request.POST.get('medical_history', "").strip()
        pet_image = request.FILES.get('pet_image')

        # Validate required fields
        if not name:
            errors['name'] = "Pet name is required."
        if not species:
            errors['species'] = "Species is required."
        if not age:
            errors['age'] = "Age is required."
        elif not age.isdigit() or int(age) < 0 or int(age) > 50:
            errors['age'] = "Please enter a valid age (0-50)."
        if not color:
            errors['color'] = "Color is required."
```

```

# Validate optional fields
if weight and (not weight.replace('.', '').isdigit() or float(weight) <= 0):
    errors['weight'] = "Please enter a valid weight."

# Validate image if provided
if pet_image:
    if pet_image.size > 5 * 1024 * 1024: # 5MB limit
        errors['pet_image'] = "Image size cannot exceed 5MB."
    elif not pet_image.content_type.startswith('image/'):
        errors['pet_image'] = "Only image files are allowed."

if not errors:
    try:
        # Get owner profile
        owner_profile = PetOwnerProfile.objects.get(user=request.user)

        # Create pet with optional fields set to None if empty
        pet = Pet.objects.create(
            owner=owner_profile,
            name=name,
            species=species,
            breed=breed if breed else None,
            age=age,
            color=color,
            weight=float(weight) if weight else None,
            vaccination_status=vaccination_status if vaccination_status else None,
            allergies=allergies if allergies else None,
            medical_history=medical_history if medical_history else None,

```

```

    pet_image=pet_image if pet_image else None
)

messages.success(request, f"{pet.name} has been successfully added to
your pets!")

return redirect('authUser:add_pet')

except PetOwnerProfile.DoesNotExist:
    messages.error(request, "You need to complete your profile before adding
pets.")

return redirect('authUser:complete_profile')

except Exception as e:
    messages.error(request, f"An error occurred while adding your pet: {str(e)}")

else:
    # Store the form data to repopulate the form
    request.session['pet_form_data'] = {
        'name': name,
        'species': species,
        'breed': breed,
        'age': age,
        'color': color,
        'weight': weight,
        'vaccination_status': vaccination_status,
        'allergies': allergies,
        'medical_history': medical_history
    }

for field, error in errors.items():

```

```
    messages.error(request, error)

# Check for saved form data in session
form_data = request.session.pop('pet_form_data', None)
context = {'form_data': form_data} if form_data else {}

return render(request, 'authUser/add_pet.html', context)
```

```
@login_required
```

```
def pet_profile(request, pet_id):
```

```
    pet = get_object_or_404(Pet, id=pet_id, owner__user=request.user)
```

```
    species_choices = [
```

```
        ('dog', 'Dog'),
```

```
        ('cat', 'Cat'),
```

```
        ('bird', 'Bird'),
```

```
        ('fish', 'Fish'),
```

```
        ('rabbit', 'Rabbit'),
```

```
        ('other', 'Other'),
```

```
    ]
```

```
if request.method == 'POST':
```

```
    errors = {}
```

```
    # Get form data
```

```
    name = request.POST.get('name', "").strip()
```

```
    species = request.POST.get('species', "").strip()
```

```
    breed = request.POST.get('breed', "").strip()
```

```

age = request.POST.get('age', "").strip()
color = request.POST.get('color', "").strip()
weight = request.POST.get('weight', "").strip()
vaccination_status = request.POST.get('vaccination_status', "").strip()
allergies = request.POST.get('allergies', "").strip()
medical_history = request.POST.get('medical_history', "").strip()
pet_image = request.FILES.get('pet_image')

# Validate required fields
if not name:
    errors['name'] = "Pet name is required."
if not species:
    errors['species'] = "Species is required."
elif species not in [choice[0] for choice in species_choices]:
    errors['species'] = "Please select a valid species."
if not age:
    errors['age'] = "Age is required."
elif not age.isdigit() or int(age) < 0 or int(age) > 50:
    errors['age'] = "Please enter a valid age (0-50)."
if not color:
    errors['color'] = "Color is required."
if not breed: # Add breed validation if it's required
    errors['breed'] = "Breed is required."

# Validate optional fields
if weight and (not weight.replace('.', '').isdigit() or float(weight) <= 0):
    errors['weight'] = "Please enter a valid weight."

```

```
# Validate image if provided
if pet_image:
    if pet_image.size > 5 * 1024 * 1024: # 5MB limit
        errors['pet_image'] = "Image size cannot exceed 5MB."
    elif not pet_image.content_type.startswith('image/'):
        errors['pet_image'] = "Only image files are allowed."

if not errors:
    try:
        # Update pet information
        pet.name = name
        pet.species = species
        pet.breed = breed # Don't set to None if it's required
        pet.age = age
        pet.color = color
        pet.weight = float(weight) if weight else None
        pet.vaccination_status = vaccination_status if vaccination_status else None
        pet.allergies = allergies if allergies else None
        pet.medical_history = medical_history if medical_history else None

        if pet_image:
            pet.pet_image = pet_image

        pet.save()
        messages.success(request, f"{pet.name}'s profile has been updated successfully!")
        return redirect('authUser:pet_profile', pet_id=pet.id)
```

```

except Exception as e:
    messages.error(request, f"An error occurred while updating the pet profile:
{str(e)}")

else:
    # Store the current valid values to repopulate the form
    pet.name = name
    pet.species = species
    pet.breed = breed # Keep the breed value
    pet.age = age
    pet.color = color
    if weight and weight.replace('.', '').isdigit():
        pet.weight = float(weight)
    pet.vaccination_status = vaccination_status
    pet.allergies = allergies
    pet.medical_history = medical_history

for field, error in errors.items():
    messages.error(request, error)

context = {
    'pet': pet,
    'species_choices': species_choices,
}
return render(request, 'authUser/pet_profile.html', context)

@login_required

```

```
def review_vet(request, appointment_id):
    """View for reviewing a veterinarian after an appointment"""

    appointment = get_object_or_404(Appointment, id=appointment_id)

    # Check if the user is authorized to review this appointment
    if request.user != appointment.pet_owner.user:
        return HttpResponseRedirect("Only the pet owner can review this appointment")

    # Check if the appointment is completed and paid
    if appointment.status != 'completed' or appointment.payment_status != 'paid':
        messages.error(request, "You can only review completed and paid appointments")
        return redirect('appointment:appointment_detail', appointment_id=appointment.id)

    # Check if the user has already reviewed this appointment
    existing_review = Review.objects.filter(
        vet=appointment.vet,
        reviewer=request.user,
        appointment=appointment
    ).first()

    if existing_review:
        messages.info(request, "You have already reviewed this appointment")
        return redirect('appointment:appointment_detail', appointment_id=appointment.id)

    if request.method == 'POST':
        # Process the review submission
        rating = int(request.POST.get('rating', 5))
        comment = request.POST.get('comment', "")
```

```
# Create the review
review = Review.objects.create(
    vet=appointment.vet,
    reviewer=request.user,
    rating=rating,
    comment=comment,
    appointment=appointment
)

messages.success(request, "Thank you for your review!")
return redirect('appointment:appointment_detail', appointment_id=appointment.id)

context = {
    'appointment': appointment
}
return render(request, 'appointment/review_vet.html', context)

def custom_404(request, exception):
    return render(request, 'errors/404.html', status=404)
```

7.11.5.3 Middleware

```
from django.conf import settings
from django.shortcuts import redirect
from django.urls import reverse
import traceback
from django.shortcuts import render
from django.http import HttpResponseRedirect

class ProfileCompletionMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        # List of paths that should bypass all checks
        allowed_paths = [
            '/complete-profile/',
            reverse('authUser:profile_verification_in_progress'),
            reverse('coreFunctions:home'),
            reverse('coreFunctions:appointment'),
            reverse('coreFunctions:community'),
            reverse('coreFunctions:about'),
            reverse('coreFunctions:contact'),
            reverse('authUser:loginUser'),
            reverse('authUser:logoutUser'),
            reverse('authUser:verify_otp'),
            reverse('authUser:forgot_password'),
        ]

```

```
# Path prefixes that should always be accessible
exempt_prefixes = (
    '/static/',
    '/media/',
    '/ws/',
    '/api/',
    '/admin/',
)

user = request.user

if not user.is_authenticated:
    return self.get_response(request)

if user.is_superuser or user.is_staff:
    return self.get_response(request)

# Bypass checks for allowed paths and exempt prefixes
if request.path.startswith(exempt_prefixes) or (request.path in allowed_paths):
    return self.get_response(request)

if not user.profile_completed:
    return redirect('complete-profile')

# Step 2
if not user.status_verification:
```

```
if user.user_type == "vet":  
    return redirect('authUser:profile_verification_in_progress')  
  
else:  
    # Only redirect to OTP verification if user is pet_owner and not verified  
    return redirect('authUser:verify_otp')  
  
  
  
return self.get_response(request)  
  
  
class CustomErrorMiddleware:  
    def __init__(self, get_response):  
        self.get_response = get_response  
  
    def __call__(self, request):  
        try:  
            response = self.get_response(request)  
  
            # You can handle 404 here if status_code is 404  
            if response.status_code == 404:  
                return render(request, 'errors/404.html', status=404)  
  
        return response  
  
  
    except Exception as e:  
        # You can log the error or email the admin here  
        if settings.DEBUG:  
            print(traceback.format_exc())  
        return render(request, 'errors/404.html', {'error': str(e)}, status=404)
```

