



islington college
(इस्लिंग्टन कॉलेज)

Module Code & Module Title

CS6P05NI Final Year Project

Assessment Weightage & Type

40% Final Report

Semester

2023-24 Autumn

PROJECT TITLE: Farmers' Bag Application

Student Name: SUMAN K.C.

London Met ID: 22015791

College ID: NP01CP4S220102

Internal Supervisor: Mr. DIPESHWOR SILWAL

External Supervisor: Mr. AASHUTOSH CHAUHAN

Assignment Due Date: 24th April 2024

Assignment Submission Date: 24th April 2024

Word Count (Where Required): 7900

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

ACKNOWLEDGEMENT

I want to express my heartfelt gratitude to several key individuals and institutions who have supported me throughout my academic journey. Firstly, I am deeply thankful to my family for their unwavering love, encouragement, and motivation. Their support has been the foundation of my success. Additionally, I extend my sincere thanks to Islington College and the FYP Support team for granting me the opportunity to undertake this project. Their guidance and assistance have been invaluable in shaping the project and helping me achieve my goals. Furthermore, I would like to acknowledge and thank my supervisors, Mr. Dipeshwor Silwal and Mr. Ashutosh Chauhan, for their continuous support, mentorship, and invaluable feedback. Their guidance has been instrumental in navigating challenges and refining my work. Overall, I am profoundly grateful to everyone who has contributed to my academic journey, helping me reach this significant milestone.

ABSTRACT

The Farmer Bag Application is a new and important project made to help farmers around the world, especially in Nepal. It was crafted as an undergraduate Final Year Project. The project combines advanced technology, education, and an online marketplace to support farmers. The main part of the application uses a special type of machine learning called EfficientNet B0 to automatically detect plant diseases by analyzing leaves. This allows farmers to identify and treat diseases early. The application also provides educational materials to teach farmers about taking care of plants. Additionally, there is a virtual marketplace where farmers can buy supplies and medicines from agricultural companies. This makes it easier for farmers to get the resources they need. The goals of the project include creating a user-friendly website, developing accurate disease detection technology, setting up a secure online marketplace, and ensuring data privacy and security. Overall, the Farmer Bag Application aims to improve agriculture, promote sustainable farming methods, and help ensure there is enough food for everyone across the world.

Table of Contents

1. INTRODUCTION.....	1
1.1. PROJECT DESCRIPTION	1
1.2. CURRENT SCENARIO	2
1.2.1. NEPAL.....	2
1.2.2. WORLD.....	3
1.3. PROBLEM DOMAIN	4
1.4. PROJECT AS SOLUTION.....	5
1.5. AIMS AND OBJECTIVES	6
1.5.1. PROJECT OBJECTIVES	6
1.5.2. ACADEMIC OBJECTIVES.....	7
14.2. STRUCTURE OF THE REPORT	8
14.2.1. BACKGROUND	8
14.2.2. DEVELOPMENT	8
14.2.3. TESTING AND ANALYSIS	8
14.2.4. CONCLUSION.....	9
2. BACKGROUND	10
2.1. ABOUT END-USERS.....	10
2.2. UNDERSTANDING THE SOLUTIONS	11
2.2.1. TECHNICAL TERMS AND DEFINITIONS	11
2.2.2. DEEP LEARNING	13
2.3. SIMILAR PROJECTS	14
2.3.1. FARM BRITE.....	14
2.3.2. FARM KART	15
2.3.3. DE HAAT	16
2.4. COMPARASIONS	17
3. DEVELOPMENT	18
3.1. CONSIDERED METHODOLOGY	18
3.1.1. SPIRAL METHODOLOGY.....	18
3.2. PHASES OF SELECTED METHODOLOGY	20
3.4. SURVEY RESULTS	21

3.4.3. PRE-SURVEY RESULTS	21
3.4.2. POST-SURVEY RESULTS	22
3.4. REQUIREMENT ANALYSIS	23
3.4.1. WORK BREAKDOWN STRUCTURE.....	24
3.4.1. GANTT CHART	25
3.5. DESIGNS.....	26
3.5.1. APPLICATION LOGO	26
3.5.2. WIREFRAME.....	27
3.5.3. USER INTERFACE(UI)	41
3.5.4. UML DESIGN	49
3.5.5. ENTITY RELATIONALSHIP DIAGRAM	55
3.6. IMPLEMENTATION	57
3.6.1. DATABASE CREATION	57
3.6.2. PROJECT STRUCTURE	69
3.6.3. NORMAL FEATURES	71
3.6.4. CORE FEATURES.....	83
3.6.5. SYTEM ARCHITECTURE DIAGRAM	87
4. TESTING AND ANALYSIS.....	88
4.1. TEST PLAN.....	88
4.1.1. BLACK-BOX TESTING.....	88
4.1.2. WHITE-BOX TESTING	88
4.1.3. INTEGRATION TESTING.....	88
4.2. BLACK-BOX TESTING.....	89
4.2.1. TEST 1 – TO CHANGE PASSWORD	89
4.2.2. TEST 2 – PREDICTION	93
4.2.2. TEST 3 – SEARCH BAR BUTTON.....	99
4.3. WHITE BOX TESTING.....	103
4.4.1. TEST 1 -TO LOGOUT FROM THE APPLICATION.....	103
4.2.2. TEST 2 – TO MANAGE ORDER.....	106
4.2.3. TEST 3 – TO PREDICT PLANT DISEASE.....	109
4.2.4. TEST 4 – TO RATING AND REVIEW PRODUCTS	116
4.2.5. TEST 5 – TO DELETE PROUDCT	120

4.2.6. TEST 6 – TO ADD TO CART	123
4.2.7. TEST 7 -TO VIEW ADMIN DASHBOARD	127
4.2.8. TEST 8 – TO REGISTER USER TO APPLICATION.....	129
4.2.9. TEST 9 – TO LOGIN USER TO THE APPLICATION.....	134
4.2.10. TEST 10 – TO RESET PASSWORD.....	138
4.2.11. TEST 11 – TO EDIT PROFILE	147
4.2.12. TEST 12 – TO VIEW MARKETPLACE: LOGGED IN USERS .Error! Bookmark not defined.	
4.2.13. TEST 13 – TO VIEW STUDY BLOG	152
4.2.14. TEST 14 – TO SEARCH PRODUCT	156
4.2.15. TEST 15 – PAYMENT.....	160
4.2.16. TEST 16 – TO DELETE USER ACCOUNT	168
4.2.17. TEST 17 – TO FILTER PRODUT BY PRICE RANGE	172
4.2.18. TEST 18 – TO SEARCH PRODUCT BY CATEGORY	177
4.2.19. TEST 19 – TO USE PAGINATION	181
4.4. INTEGRATION TESTING.....	184
4.4.1. TEST 1 -TO REMOVE PRODUCT FROM CART.....	184
4.4.2. TEST 4 – TO ADD PRODUCT POST	186
4.4.3. TEST 2 – TO UPDATE PRODUCT	189
4.4.4. TEST 3 – TO DELETE PROUDCT	192
4.4.5. TEST 4 – TO ADD BLOG POST	194
4.5.6. TEST 5 – TO UPDATE BLOG POST	197
4.5.7. TEST 6 –TO DELETE BLOGS POST.....	201
4.4. CRITICAL ANALYSIS	203
5. CONCLUSION	205
5.1. LEGAL, SOCIAL AND ETHICAL ISSUES	205
5.1.1. LEGAL ISSUES	205
5.1.2. SOCIAL ISSUES.....	206
5.1.3. ETHICAL ISSUES	206
5.2. ADVANTAGES	207
5.3. LIMITATIONS	208
5.2. FUTURE WORK.....	209
6. REFERENCES AND BIBLIOGRAPHY	210

7. APPENDIX	214
7.1. APPENDIX A: PRE-SURVEY	214
7.1.1. PRE-SURVEY FORM	214
7.1.2. SAMPLE OF FILLED PRE-SURVEY FORM.....	217
7.1.3. RESULT OF PRE-SURVEY FORM	220
7.2. APPENDIX B: POST-SURVEY	225
7.2.1. POST-SURVEY FORM	225
7.2.2. SAMPLE OF FILLED SURVEY FORM.....	230
7.2.3. POST SURVEY RESUTS	235
7.3. APPENDIX C: REQUIREMENT ANALYSIS	245
7.3.1. SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)	245
7.4. APPENDIX D: BACKGROUND.....	254
7.4.1. UNDERSTANDING THE SOULTION.....	254
7.4.2. SIMILAR PROJECTS	263
7.4.4. PHASES OF CONSIDERED METHODOLOGY (RUP).....	264
7.4.4.1. REASONS FOR CHOOSING RATIONAL UNIFIED PROCESS	266
7.5. APPENDIX E: DESIGN.....	267
7.5.1. USE CASE DIAGRAM DESCRIPTION.....	267
7.5.2. SEQUENCE DIAGRAM.....	302
7.5.3. ACTIVITY DIAGRAM.....	312
7.5.4. COLLABORATION DIAGRAM	318
7.5.5. IMPLEMENTATIONS.....	328
7.6. APPENDIX F: CODE SAMPLE.....	331
7.6.1. SAMPLE CODE OF THE UI.....	331
7.7. APPENDIX G: CONCLUSION	339
7.7.1. FUTURE WORK.....	339

TABLE OF FIGURES

Figure 1: Insufficient Crop Production By 2050 (Deepak K. Ray ,Nathaniel D. Mueller,Paul C. West,Jonathan A. Foley, 2013).....	2
Figure 2: Late Blight Outbreaks in the United States (Jean B. Ristaino, 2021).....	4
Figure 3: Project as a Solution: User Response.....	5
Figure 4: End Users diagrammatic representation.....	10
Figure 5: Project as Solution: Deep Learning.....	13
Figure 6: Similar Project 1: Home page of Farm Brite	14
Figure 7: Similar Project 2: Homepage of Farm Kart Application	15
Figure 8: Similar Project 3: Homepage of DeHaat Application	16
Figure 9: Considered Methodology: Spiral	18
Figure 10: Phases of Methodology (Krutch, 2004)	20
Figure 11: Pre-Survey: Farmer Bag Application.....	21
Figure 12: Pre-Survey Result: Question 1	21
Figure 13: Farmer's Bag Post-Survey	22
Figure 14: Post-Survey: Question 1	22
Figure 15: Work Breakdown Structure	24
Figure 16: Design: Application Logo	26
Figure 17: Wireframe: Home Page	27
Figure 18: WireFrame: Change Password Page	28
Figure 19: Wire Frame: Update Profile Page.....	29
Figure 20: Wire Frame: Checkout Page	30
Figure 21: Wire Frame: Cart Page	31
Figure 22: Wire Frame: Product Description and Review Page	32
Figure 23: Wire Frame: Blog Page.....	33
Figure 24: Wire Frame: View User	34
Figure 25: Wire Frame: Create Blog Post	35
Figure 26: Wireframe: Create Product.....	36
Figure 27: Wire Frame: Admin Dashboard	37
Figure 28: Wire Frame: Admin Page.....	38
Figure 29: Wire Frame: Forgot Password	39
Figure 30: Wire Frame: Login Page	40
Figure 31: User Interface: Home Page.....	41
Figure 32: User Interface: Checkout Page.....	42
Figure 33: User Interface: Edit Profile.....	42
Figure 34: User Interface: Change Password.....	43
Figure 35: User Interface: Sign in	43
Figure 36: User Interface: Forget Password	44
Figure 37: User Interface: User Dashboard	44
Figure 38: User Interface: Product Description.....	45
Figure 39: User Interface: Admin Dashboard	46
Figure 40: User Interface: Add to Cart	46

Figure 41: User Interface: Admin Add Product	47
Figure 42: User Interface: Admin Create Blog Post	47
Figure 43: User Interface: Manage User Account	48
Figure 44: Design: Use Case Diagram.....	49
Figure 45: Design: Data Flow Diagram	50
Figure 46: Sequence Diagram: NON-Registered Users	52
Figure 47: Activity Diagram: Un-Registered Users.....	53
Figure 48: Collaboration Diagram: Register Accounts	54
Figure 49: Entity Relationship Diagram	55
Figure 50: Database Creation: ACCOUNTS	57
Figure 51: Database Creation: ACCOUNTS (2).....	58
Figure 52: Database Creation: SQLLITE (ACCOUNT)	58
Figure 53: Database Creation: ACCOUNTS (3).....	59
Figure 54: Database Creation: SQLLITE (USERPROFILE).....	59
Figure 55: Database Creation: BLOG.....	60
Figure 56: Database Creation: SQLLITE (BLOGPOST)	60
Figure 57: Database Creation: CART	61
Figure 58: Database Creation: SQLLITE (CART)	61
Figure 59: Database Creation: SQLLITE (CART ITEM)	62
Figure 60: Database Creation: Category	63
Figure 61: Database Creation: SQLLITE (CATEGORY)	63
Figure 62: Database Creation: ORDER	64
Figure 63: Database Creation: SQLLITE (ORDER)	65
Figure 64: Database Creation: SQLLITE (ORDERITEM).....	65
Figure 65: Database Creation: SQLLITE (PAYMENT)	65
Figure 66: Database Creation: PRODUCT.....	66
Figure 67: Database Creation: SQLLITE (PRODUCT).....	67
Figure 68: Database Creation: Review	67
Figure 69: Database Creation: SQLite (REVIEW).....	68
Figure 70: Implementation: Django Applications	69
Figure 71: Screenshot of registering apps to settings.py file.....	70
Figure 72: Registering Static file in core app.....	71
Figure 73: Normal Features: Registered Static Files.....	71
Figure 74: Normal Features: Media File Registration.....	72
Figure 75: Normal Features: Registered Media Files.....	72
Figure 76: Normal Features: Database Registration	73
Figure 77: Normal Features: Database: SQLite	73
Figure 78: Normal Features: Registering Project URLs	74
Figure 79: Normal Features: Registering Templates	74
Figure 80 : Normal Features: Registered Templates 1 Figure 81: Normal Features: Registered Templates 2	75
Figure 82: Normal Features: Register User Function	76
Figure 83: Normal Features: Login User Function	77

Figure 84: Normal Functions: Login UI	77
Figure 85: Normal Features: Register User UI	78
Figure 86: Normal Features: Logout Function.....	78
Figure 87: Normal Feature: Logout UI	79
Figure 88: Normal Feature: Edit Profile	79
Figure 89: Normal Feature: Edit Profile UI.....	80
Figure 90: Normal Feature: Admin Dashboard.....	81
Figure 91: Normal Feature: Admin Dashboard UI(1)	81
Figure 92: Normal Feature: Admin Dashboard UI (2)	82
Figure 93: Screenshot Importing Libraries for Plant Disease Detection.	83
Figure 94: Screenshot prediction function of Plant disease detection.	83
Figure 95: Screenshot of loading and predicting function of Plant Disease Detection	84
Figure 96: Screenshot of defining class labels.	84
Figure 97: Screenshot of loading model and creating recommend product function.....	85
Figure 98: Screenshot of Plant disease detection function.....	85
Figure 99: Screenshot of UI of Prediction Page	86
Figure 100:Screenshot of UI of Recommend Product.....	86
Figure 101: System Architecture Diagram	87
Figure 102: Change Password: Form.....	90
Figure 103: Change Password: Valid Details	90
Figure 104: Change Password: Redirect to login page with success message.	91
Figure 105: Change Password: Invalid Details	91
Figure 106: Change Password: Error Message Displays and 404 Not Found Error	92
Figure 107: Prediction: System through Error "Uncaught Reference Error".....	94
Figure 108: Prediction: System through MultiValueDictKey Error.....	94
Figure 109: Prediction: Error in console.	95
Figure 110: Prediction: Place this code to try and expect block with MultiValueDictKey Error	96
Figure 111: Prediction: Place code inside try and expect block with MultiValueDictKey Error	97
Figure 112: Prediction: Message Displayed "Please Upload a file"	98
Figure 113: Search Bar: Click on Search Bar Button without any keywords	99
Figure 114: Search Bar: System Displays Error Message	100
Figure 115: Search Bar: Error "Product Not associated to value".	100
Figure 116: Search Product: Set Products = None, product count = 0	101
Figure 117: Search Bar Button: System successfully displays "No result found" message.....	102
Figure 118: Screenshot of Logging out from application (1)	104
Figure 119: Screenshot of Logging out from application (2)	104
Figure 120: Screenshot of redirected page after logging out.....	105
Figure 121: Screenshot of Logout Function Code (1).....	105
Figure 122: Screenshot of Logout Function Used in Navigation Bar.....	105
Figure 123: Screenshot of Logout Function Used in Dashboard	106
Figure 124: Manage Order: Navigate to My Order	107
Figure 125: Manage Order: Click on delete order.	107
Figure 126: Manage Order: Code (1)	108

Figure 127: Manage Order: Code (2)	108
Figure 128: Prediction: Navigate to Predict Page.....	110
Figure 129: Prediction: Drag and Drop the leaf photo.	110
Figure 130: Prediction: Predict the 'Tomato_Bacterial_spot'.....	111
Figure 131: Prediction: Recommendation according to detected disease	111
Figure 132:Prediction: Choose Potato Leaf.....	112
Figure 133: Prediction: Disease Detected on Leaf	112
Figure 134: Prediction: Product Recommendation according to disease detected.	113
Figure 135: Prediction: Handling Unknown Class	113
Figure 136: Prediction: Handled Unknown Class	114
Figure 137: Recommend Product Function	114
Figure 138: Image Prediction Function.....	115
Figure 139: Rate Review: Product Description Page	117
Figure 140: Rate and Review: Form.....	117
Figure 141: Rate and Review: Providing Ratings	117
Figure 142: Rate and Review: Successfully Provide Rate and Review	118
Figure 143: Delete Product: Navigate to Admin Product and Select 'delete' Product.....	121
Figure 144: Delete Product: Successfully deleted product	121
Figure 145: Delete Product: Product Not in database.	122
Figure 146: Delete Product: Code (1).....	122
Figure 147: Delete Product: Code (2).....	122
Figure 148: Add to Cart: Navigate to Product Description and click on "Add to Cart".....	124
Figure 149: Add to Cart: Cart Added Successfully	124
Figure 150: Add to Cart: IF single product in cart.....	125
Figure 151: Add to Cart: Code (1)	125
Figure 152: Add to Cart: Code (2)	126
Figure 153: Admin Dashboard: Navigate admin to dashboard.....	127
Figure 154: Admin Dashboard: Chart and KPIs (1)	128
Figure 155: Admin Dashboard: Chart (1)	128
Figure 156: Screenshot of registering user to system (1).....	130
Figure 157: Screenshot of error while registering user to system with same account.....	130
Figure 158: Screenshot of activation link after registering account.	131
Figure 159: Screenshot of Registering User: Activation Link	131
Figure 160: Screenshot of registering user: Success Message	132
Figure 161: Screenshot of registering user: Code (1).....	132
Figure 162:Screenshot of registering user: Code (2).....	133
Figure 163: Screenshot of Login: Registered User	135
Figure 164: Screenshot of Login: After Login Dashboard	135
Figure 165: Screenshot of Login: Code (1).	136
Figure 166: Screenshot of Login: Code (2)	136
Figure 167: Screenshot of Login: Code (3).	137
Figure 168: Reset password: Login Form	139
Figure 169: Reset Password: Forgot Password Form	139

Figure 170: Reset password: Enter email address.....	140
Figure 171: Reset password: reset link confirmation link.....	140
Figure 172: Reset password: Gmail Inbox	141
Figure 173: Reset password: reset password link.....	141
Figure 174: Reset password: reset password form	142
Figure 175: Reset password: Creating new password:.....	142
Figure 176: Reset Password: Screenshot of successful message.....	143
Figure 177: Reset Password: Login with new password	143
Figure 178: Reset Password: Dashboard.....	144
Figure 179: Reset Password: Code(1).....	144
Figure 180: Reset Password: Code (2).....	145
Figure 181: Reset Password: Code (3).....	146
Figure 182: Edit Profile: Navigate to Dashboard	148
Figure 183: Edit Profile: Navigate to Edit Profile and Enter Details	148
Figure 184: Edit Profile: Click on save button.....	149
Figure 185: Edit Profile: Success Message Displays	149
Figure 186: Edit Profile: Code (1).....	150
Figure 187: Edit Profile: Code (3).....	150
Figure 188: Edit Profile: Code (4).....	151
Figure 189: Study Blog: Dashboard	153
Figure 190: Study Blog: Navigate to Blog at Navigation Bar.....	153
Figure 191: Blog: System redirected to Blog Page.....	154
Figure 192: Blog: Code (1)	154
Figure 193: Blog: Code (2)	155
Figure 194: Search Product: Navigate to Search Bar.....	157
Figure 195: Search Product: Results of "Tomato".....	157
Figure 196: Search Product: Search Random Product	158
Figure 197: Search Product: Message "No result found"	158
Figure 198: Search Product: Code (1)	159
Figure 199: Search Product: Code (2)	159
Figure 200: Payment: Navigate to Product Page.....	161
Figure 201: Payment: Navigate to Product Description Page	161
Figure 202: Payment: Product Added to Cart	162
Figure 203: Payment: Redirected to Checkout Page and choose khalti	162
Figure 204: Payment: Khalti Interface (1).....	163
Figure 205: Payment: Khalti Interface (2).....	163
Figure 206: Payment: Khalti Interface (2).....	164
Figure 207: Payment: Code (1).....	164
Figure 208:Payment: Code (2).....	165
Figure 209:Payment: Code (3).....	166
Figure 210: Payment: Code (4).....	166
Figure 211: Payment: Code (5).....	167
Figure 212: Payment: Code (6).....	167

Figure 213: Delete Account: Admin Dashboard	169
Figure 214: Delete Account: Navigate to Users	169
Figure 215: Delete Account: Click on Delete Account	170
Figure 216: Delete Account: Code (1).....	170
Figure 217: Delete Account: Code (2).....	171
Figure 218: Filter Product By Price: Market Page.....	173
Figure 219: Filter by Price: Select Price Range	173
Figure 220: Filter by Price: Choose Apply button and Product Displays	174
Figure 221: Filter by price: Code (1).....	175
Figure 222: Filter by price: Code (2).....	176
Figure 223: Search Product by category: Market Page and Select Category on left side	178
Figure 224: Search Product by category: Displays Product by category.	178
Figure 225: Search Product by category: Displays Product by category 'Potato".....	179
Figure 226Search Product by category: Displays Product by category 'E wears'.....	179
Figure 227:Search Product by category: Code (1).....	180
Figure 228: Pagination: Navigate to Market page and Select next or '1' from bottom of page.	182
Figure 229: Pagination: Page '2' and 'Next' Page displays.	182
Figure 230: Pagination: Code (1)	183
Figure 231 : Pagination: Code (2)	183
Figure 232: Remove Product: Navigate to Cart Page	185
Figure 233: Remove Cart: Click on Remove button "Empty Cart Display"	185
Figure 234: Add Product: Navigate to Add Product Form	187
Figure 235: Add Product: Providing all the details.....	187
Figure 236: Add Product: Product added to Store page	188
Figure 237: Add Product: Product successfully added to database.	188
Figure 238: Update Product: Change the Product Price and Stock	190
Figure 239: Update Product: Price Changed in Product Description Page	190
Figure 240: Update Product: Product "price" and "stock" updated in database	191
Figure 241: Delete Product: Navigate to Product Page and Select Delete button.	192
Figure 242: Delete Product: Product Deleted from Database.....	193
Figure 243: Blog: Navigate to Add Blog Post	195
Figure 244: Blog: Create A Blog Post and Click on Submit Button	195
Figure 245: Blogs: Blog appeared in Blog Page	196
Figure 246: Blogs: Created blog is in database.	196
Figure 247: Update Blog: Navigate to Blog and Click on Update button.	198
Figure 248: Update Blog: Update Blogs Details	198
Figure 249: Update Blogs: Click on submit button.	199
Figure 250: Update Blog: Blog Post Updated.....	199
Figure 251: Update Blog: Blog Post Updated in database.	200
Figure 252: Delete Blog: Navigate to blog and click on delete button.	202
Figure 253: Delete Blog: Blog Deleted from Database.....	202
Figure 254: Pre-Survey Form: Question 1	214
Figure 255: Pre-Survey Form: Question 2	214

Figure 256: Pre-Survey Form: Question 3	215
Figure 257: Pre-Survey Form: Question 4	215
Figure 258: Pre-Survey Form: Question 5	216
Figure 259: Sample of Filled Pre-Survey Survey (1).....	217
Figure 260:Sample of Filled Pre-Survey Survey (2).....	217
Figure 261:Sample of Filled Pre-Survey Survey (3).....	218
Figure 262: Sample of Filled Pre-Survey Survey (4).....	218
Figure 263:Sample of Filled Pre-Survey Survey (5).....	219
Figure 264: Pre-Survey: Question 1	220
Figure 265: Pre-Survey: Question 2	221
Figure 266: Pre-Survey: Question 3	222
Figure 267: Pre-Survey: Question 4	223
Figure 268: Pre-Survey: Question 5	224
Figure 269: Post-Survey Form: Question 1	225
Figure 270: Post-Survey Form: Question 2	225
Figure 271: Post-Survey Form: Question 3	226
Figure 272: Post-Survey Form: Question 4	226
Figure 273: Post-Survey Form: Question 5	227
Figure 274: Post-Survey Form: Question 6	227
Figure 275: Post-Survey Form: Question 7	228
Figure 276: Post-Survey Form: Question 8	228
Figure 277: Post-Survey Form: Question 9	229
Figure 278: Post-Survey Form: Question 10	229
Figure 279: Sample of Filled Post-Survey (1)	230
Figure 280:Sample of Filled Post-Survey (2)	230
Figure 281: Sample of Filled Post-Survey (3)	231
Figure 282: Sample of Filled Post-Survey (4)	231
Figure 283: Sample of Filled Post-Survey (5)	232
Figure 284: Sample of Filled Post-Survey (6).....	232
Figure 285: Sample of Filled Post-Survey (7)	233
Figure 286:Sample of Filled Post-Survey (8)	233
Figure 287: Sample of Filled Post-Survey (9)	234
Figure 288: Sample of Filled Post-Survey (10).....	234
Figure 289: Post-Survey: Question 1.....	235
Figure 290: Post-Survey: Question 2.....	236
Figure 291: Post Survey: Question 3	237
Figure 292: Post-Survey: Question 4.....	238
Figure 293: Post-Survey: Question 5.....	239
Figure 294: Post-Survey: Question 6.....	240
Figure 295: Post-Survey: Question 7.....	241
Figure 296: Post-Survey: Question 8.....	242
Figure 297: Post-Survey: Question 9.....	243
Figure 298: Post-Survey: Question 10.....	244

Figure 299: Transfer Learning Used Model	254
Figure 300: Importing Necessary Libraries	255
Figure 301: Creating two directory path.	255
Figure 302: Code to returning no of files.	255
Figure 303: Setting Batch size and Image shape.	256
Figure 304: Data Augmentation.....	257
Figure 305: Implementing Transfer Learning Model.....	258
Figure 306: Compiling Model	258
Figure 307: Printing Model Summary	259
Figure 308: Training Model in 5 Epochs	259
Figure 309: Plotting Traiing and Validation Chart	260
Figure 310: Plotting Validation and Accuracy Chart.....	260
Figure 311: Printing precision, recall and f1score	261
Figure 312: Building Confusion Matrix	262
Figure 313: Saving the model.....	262
Figure 314: Considered Methodology: Rapid Application Development (kissflow, 2023)....	263
Figure 315: Phases of Methodology: Rational Unified Process	264
Figure 316: Sequence Diagram: Registered Users	303
Figure 317: Sequence Diagram: Admin Portal	304
Figure 318: Sequence Diagram: All Users (Farmers)	305
Figure 319: Sequence Diagram for All Users: Edit Profile.....	306
Figure 320: Sequence Diagram for All Users: Change Password.....	306
Figure 321: Sequence Diagram Admin/User: Logout.....	307
Figure 322: Sequence Diagram: Browse Product.....	308
Figure 323: Sequence Diagram All Users: Add to Cart	309
Figure 324: Sequence Diagram: Payment	310
Figure 325: Khalti Payment Flow	310
Figure 326: Sequence Diagram All Users: Prediction.....	311
Figure 327: Activity Diagram: Registered Users	312
Figure 328: Activity Diagram: All Users Functionality	313
Figure 329: Activity Diagram: Add to Cart.....	314
Figure 330: Activity Diagram: Prediction	315
Figure 331: Activity Diagram: Admin.....	317
Figure 332: Collaboration Diagram: Login Users	319
Figure 333: Collaboration Diagram: Logout Users	319
Figure 334: Collaboration Diagram: Edit Profile	320
Figure 335: Collaboration Diagram: Change Password	320
Figure 336: Collaboration Diagram: Blogs Information	321
Figure 337: Collaboration Diagram: Search Product	321
Figure 338: Collaboration Diagram: Rate Review	322
Figure 339: Collaboration Diagram: Prediction	322
Figure 340: Collaboration Diagram: Payment	323
Figure 341: Collaboration Diagram: Add to Cart	323

Figure 342: Collaboration Diagram: Admin Portal Add Product	324
Figure 343: Collaboration Diagram: Admin Portal Update Product	324
Figure 344: Collaboration Diagram: Admin Portal Delete Product	325
Figure 345: Collaboration Diagram: Admin Portal Add Blogs Post	325
Figure 346: Collaboration Diagram: Admin Portal Update Blog	326
Figure 347: Collaboration Diagram: Admin Portal Delete	326
Figure 348: Collaboration Diagram: Admin Portal Manage User Account	327
Figure 349: Application Architecture: Model View Template (MVT).....	329
Figure 350: System Architecture Diagram: Clean Architecture	330
Figure 351: Sample Code Of Ui: Admin Dashboard	331
Figure 352: Sample Code Of Ui: Change Password	331
Figure 353: Sample Code Of Ui: Prediction	332
Figure 354: Sample Code Of Ui: Store	332
Figure 355: Sample Code Of Ui: Product Description.....	333
Figure 356: Sample Code Of Ui: Checkout.....	333
Figure 357: Sample Code Of Ui: Cart	334
Figure 358: Sample Code Of Ui: Blog Post	334
Figure 359: Sample Code Of Ui: Login	335
Figure 360: Sample Code Of Ui: Forgot Password	336
Figure 361: Sample Code Of Ui: View Users.....	336
Figure 362: Sample Code Of Ui: View Ratings	337
Figure 363: Sample Code Of Ui: Base.....	337
Figure 364: Sample Code Of Ui: Home	338

LIST OF TABLES

Table 1: Similar Project Comparison.....	17
Table 2: Black-Box Testing: To Change Password	89
Table 3: Test 2- Black Box Testing: Prediction “without selecting image”	93
Table 4: Black Box Testing: Handling MultiValueKeyDict Error.....	95
Table 5: Black Box Testing: Search Bar Button	99
Table 6: Test 3: Handling Error	101
Table 7: Black-Box Testing: Test 1- To Logout from The Application	103
Table 8: White Box Testing: Manage Order.....	106
Table 9: White Box Testing: Prediction & Recommendation	109
Table 10: White Box Testing: Rate and Review	116
Table 11: White Box Testing: To Delete Product	120
Table 12: White Box Testing: Add To Cart	123
Table 13: White Box Testing: Admin Dashboard	127
Table 14: White Box Testing: To register User to application.....	129
Table 15: White Box Testing: To Login User to the Application	134
Table 16: White Box Testing: To Reset Password	138
Table 17: White Box Testing: To Edit Profile	147
Table 18: White Box Testing: To View Study Blog	152
Table 19: White Box Testing: To Search Product	156
Table 20: White Box Testing: Payment	160
Table 21: White Box Testing: To Delete User Account.....	168
Table 22: White Box Testing: To Filter Product by Price	172
Table 23:White Box Testing: To Search Product by Category	177
Table 24: White Box Testing: To Use Pagination	181
Table 25: Integration Testing: Remove Cart	184
Table 26: Integration Testing: To Add Product	186
Table 27: Integration Testing: To Update Product	189
Table 28: Integration Testing: To Delete Product	192
Table 29: Integration Testing: To Add Blog Post.....	194
Table 30: Integration Testing: To Update Blog Post	197
Table 31: Integration Testing: To Delete Blog Post	201
Table 32:High Level Use Case Diagram: Register	267
Table 33: High Level Use Case Diagram: Dashboard	268
Table 34: High Level Use Case Diagram: Login/Logout	268
Table 35: High Level Use Case Diagram: Browse Product	269
Table 36: High Level Use Case Diagram: Search Product	269
Table 37:High Level Use Case Diagram: View Cart	270
Table 38:High Level Use Case Diagram: Update Cart	270
Table 39:High Level Use Case Diagram: Ratings	271
Table 40:High Level Use Case Diagram: Predict Disease.....	271
Table 41:High Level Use Case Diagram: View Study Place	272

Table 42: High Level Use Case Diagram: Pay Bill	272
Table 43: High Level Use Case Diagram: Manage Accounts	273
Table 44: High Level Use Case Diagram: Add Product.....	273
Table 45: High Level Use Case Diagram: Delete Product.....	274
Table 46: High Level Use Case Diagram: Add Categories.....	274
Table 47: High Level Use Case Diagram: Delete Categories	275
Table 48: High Level Use Case Diagram of Manage Payments.....	275
Table 49: Expanded Level Use Case Diagram: Register	276
Table 50: Expanded Level Use Case Diagram: View Dashboard.....	278
Table 51: Expanded Level Use Case Diagram: Login/Logout.....	280
Table 52: Expanded Level Use Case Diagram: Browse Product.....	282
Table 53: Expanded Level Use Case Diagram: Search Product	283
Table 54: Expanded Level Use Case Diagram: View Cart	284
Table 55: Expanded Level Use Case Diagram: Update Cart	285
Table 56: Expanded Level Use Case Diaram: Ratings.....	287
Table 57: Expanded Level Use Case Diagram: Predict Disease.....	288
Table 58: Expanded Level Use Case Diagram: View Study Place	289
Table 59: Expanded Level Use Case Diagram: Pay Bill	291
Table 60: Expanded Level Use Case Diagram: Manage Accounts	293
Table 61: Expanded Level Use Case Diagram: Add Product.....	295
Table 62: Expanded Level Use Case Diagram: Delete Product	297
Table 63: Expanded Level Use Case Diagram: Add Categories	298
Table 64: Expanded Level Use Case Diagram: Delete Categories	299
Table 65: Expanded Level Use Case Diagram: Manage Payments	301
Table 66: Activity Diagram: Payment	316

1. INTRODUCTION

1.1. PROJECT DESCRIPTION

As a Final Year Project (FYP) for the undergraduate program, the Farmer Bag Application is an intelligent and affordable app designed to empower farmers, gardeners, and plant enthusiasts by providing them with crucial tools and resources. This project aims to demonstrate the practical knowledge and learning outcomes accumulated throughout the research and completed courses.

At its core, the Farmer Bag Application leverages advanced technology to automatically detect crop diseases through leaf analysis, enabling early identification and swift action. However, the app goes beyond disease detection by serving as a comprehensive educational repository, equipping users with valuable knowledge and insights to continuously enhance their plant health management skills. By fostering a deeper understanding of best practices, users can make informed decisions regarding crop selection, input usage, and sustainable farming methods.

Moreover, the app functions as a virtual marketplace, seamlessly connecting farmers, gardeners, and plant enthusiasts with agrovet companies and suppliers of agricultural medicines and inputs. This integration streamlines the procurement process, ensuring timely access to essential resources for maintaining crop health and productivity.

Through this fusion of cutting-edge technology, educational resources, and a user-friendly marketplace, the Farmer Bag Application aims to strengthen agriculture, uplift farmers' livelihoods, and contribute to a more sustainable and abundant future for our communities. The project is being developed and documented in a structured and behavioral manner, under the supervision of external and internal supervisors, with the expected outcome of delivering a full-stack application.

1.2. CURRENT SCENARIO

1.2.1. NEPAL

In context of Nepal, we know that it's an agricultural country, and around 67% of the total population is involved in farming. Most of them farm to sustain their lives. Currently, there are various challenges in Nepalese agriculture. One of the major problems is plant diseases, especially bacterial diseases. The Bacterial wilt of potato, tomato, soyabean, and corn is a significant issue, causing sustainable losses for framers. It's crucial to address this problem to ensure there's enough food, and farming can continue successfully in Nepal (Poudel & Neupane, 2018).

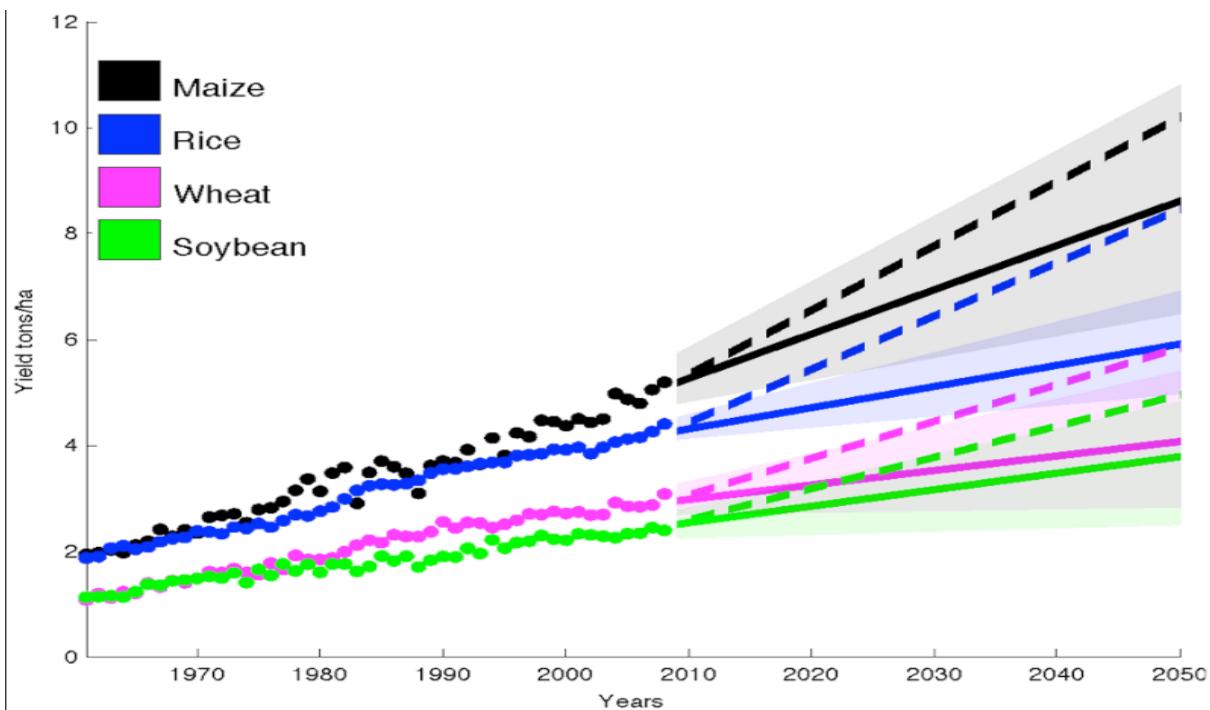


Figure 1: Insufficient Crop Production By 2050 (Deepak K. Ray ,Nathaniel D. Mueller,Paul C. West,Jonathan A. Foley, 2013)

1.2.2. WORLD

Plant diseases are a big problem for farming around the world. They cause up to 40% of crops to be lost every year. This results in economic losses of \$220 billion. As the world's population is expected to reach 9.7 billion by 2050, more food production is needed (FAO, 2017). Farmers often use intensive farming practices to increase yields, but this can help plant diseases spread more easily. The increase in global trade also introduces new plant diseases from other countries, which can severely impact poorer nations. To address this in a sustainable way, integrated pest management (IPM) is important. IPM aims to reduce the use of chemical pesticides by using natural methods to control pests and only using pesticides when absolutely necessary. The European Commission's Green Deal, started in 2019, promotes sustainable farming practices. Quickly detecting plant diseases is crucial for effective disease management. This allows for targeted treatment, leading to less pesticide use and more sustainable agriculture (McDonald, 2016).

1.3. PROBLEM DOMAIN

The farming sector is super important for feeding everyone around the world. But there's a big problem: plant diseases are hurting this crucial industry. Farmers struggle to figure out and fix these diseases in their crops. This can lead to losing money and hurting the environment by using too many pesticides. This problem is serious, and there are a lot of important things to think about.

- The increasing global population presents a pressing challenge: ensuring food security. Crop diseases pose a significant threat by reducing both crop yields and quality, thereby endangering the world's food supply. (FAO, 2017)
- A big problem is that people are using too many pesticides to fight plant diseases, which hurts the environment (USADA, 2022). Also, because people don't know enough about taking care of plants, about one-third of crops get lost or wasted after they're harvested. This wastes time, money, and the stuff farmers need to grow crops (Health, 2017)

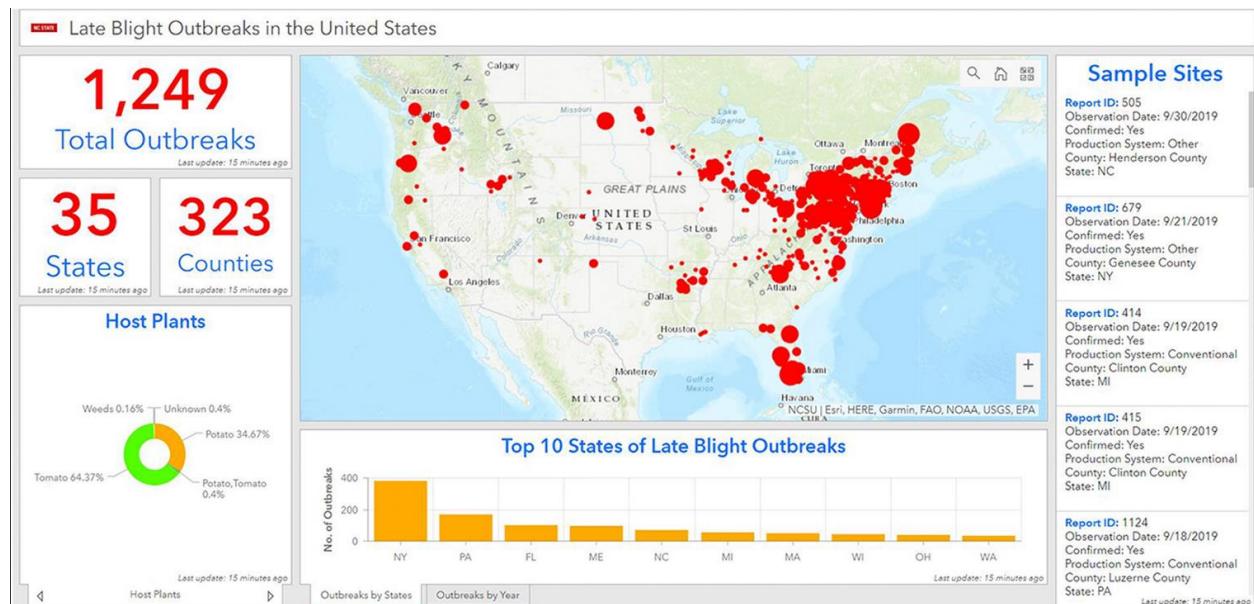


Figure 2: Late Blight Outbreaks in the United States (Jean B. Ristaino, 2021).

1.4. PROJECT AS SOLUTION

The project seeks to create a web application that can detect plant diseases. With the assistance of this web application, farmers will be able to learn about plant diseases, obtain information on them, understand how to eradicate the diseases, and purchase medicines and crops. According to the survey, 94% of respondents believe that the Farmer's Bag Application will be helpful for farmers. This strong endorsement underscores widespread support for the project's potential benefits. This application is beneficial for farmers as it facilitates the early identification of diseases, leading to timely intervention and minimized crop damage. Precision in treatment not only boosts crop yields but also results in cost savings by avoiding unnecessary expenses on the widespread use of pesticides. It aids farmers in using pesticides more efficiently, promoting sustainable agricultural practices and reducing environmental impact.

How satisfied are you with the user interface and overall usability of the Farmer Bag Application?

50 responses

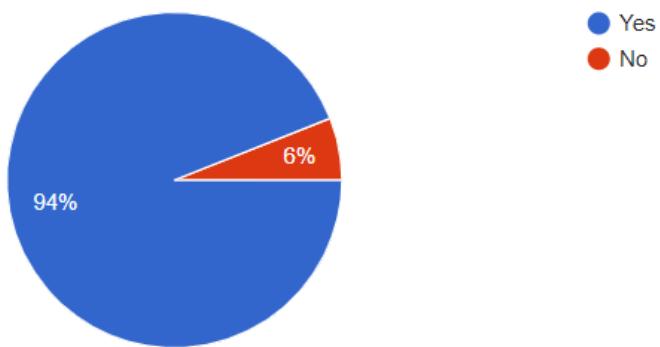


Figure 3: Project as a Solution: User Response

1.5. AIMS AND OBJECTIVES

The Farmers' Bag Application aims to revolutionize global agriculture through advanced technology, specifically CNN, for predictive plant disease management. This project not only addresses global farming challenges but also prioritizes farmer education with an integrated Study Blog, alongside a marketplace where farmers can conveniently purchase medicines and crops, promoting a holistic and sustainable approach to modern farming.

1.5.1. PROJECT OBJECTIVES

The project objective of the project is: -

- **User Friendly Web Application:** Create a simple web application for farmers that's easy to use and works well on different devices like phones and tablets.
- **Disease Detection and Information:** Implement a deep learning model, especially Convolution Neural Network (CNNs), for accurate and automated analysis of plant leaf image to detect diseases and provide detailed information about identified diseases.
- **Marketplace Integration:** Develop a secure marketplace platform integrated into the application for farmers to easily purchase medicines and crops.
- **Payment Gateway Integration:** Develop a Khalti payment gateway.
- **Data Security Measures:** Implement robust authentication measures for data security.

14.1.2. ACADEMIC OBJECTIVES

The academic objectives of the project are as follows:

- To acquire knowledge about the Python Framework (Django) and its libraries, exploring the features provided by its ecosystem.
- To gain understanding and proficiency in Deep Learning Algorithms, especially Convolutional Neural Networks (CNN) for image processing.
- To gain knowledge in transfer learning.
- To implement a structured database management system.
- To learn the process of building web applications.
- To understand and implement authentication features in a more secure manner.
- To develop skills in troubleshooting and resolving errors and bugs during the development process.
- To explore APIs and utilize them for backend development purposes.
- To gain insights into the different phases of software engineering and effectively implement the chosen methodology throughout the development lifecycle.
- To adhere to the Work Breakdown Structure and Gantt Chart to monitor and track progress.

14.2. STRUCTURE OF THE REPORT

14.2.2. BACKGROUND

The background section establishes the theoretical foundation for the projects, gather and analyzes user requirements to identify the problem, proposes a suitable solution with an appropriate technology stack, and compares the project to similar existing approaches. This groundwork informs the development phase of the final year project.

14.2.3. DEVELOPMENT

The development section outlines the project's construction using the Rational Unified Process (RUP) methodology. RUP covers all phases (inception, elaboration, construction, transition), including requirement gathering, time management, pre- and post-evaluations, design diagrams, creating software requirement specifications (SRS), implementing features, testing procedures, and deployment strategies.

14.2.4. TESTING AND ANALYSIS

This section focuses on the functionality and performance of the project. It consists of a test plan with components for unit testing and system testing. For this project, black box, white box, and integration testing were performed. It also provides a critical analysis of the project's positive and negative aspects.

14.2.5. CONCLUSION

The section highlights the project's development challenges, addresses legal and ethical considerations, outlines advantages and limitations, and proposes future improvements to enhance the application's functionality and accessibility. The section discusses development challenges, legal/ethical considerations, and project advantages/limitations. It addresses prior knowledge gaps, data protection, and accessibility concerns. Future work proposals involve framework migration, mobile app development, and AI integration for enhanced functionality.

2. BACKGROUND

2.1. ABOUT END-USERS

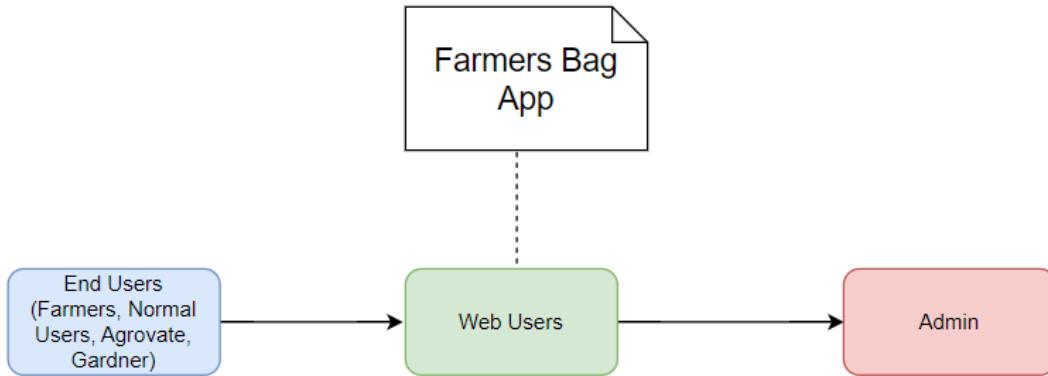


Figure 4: End Users diagrammatic representation.

This Farmer Bag Application involves multiple end-users. The farmers, plant enthusiasts, and normal users such as gardeners and agrovet companies are part of this Farmer Bag project, while the admin is the end-user for the admin portal. The application aims to provide a convenient platform for farmers, gardeners, and plant enthusiasts to interact and access resources related to agriculture. It also serves as a marketplace for agrovet companies to offer their products and services to the target audience.

2.2. UNDERSTANDING THE SOLUTIONS

2.2.1. TECHNICAL TERMS AND DEFINITIONS

Some of the Technical Terms and Definitions used while building this project are explained below:

- **Django:**

Django is a robust Python web framework that simplifies the process of building dynamic websites. It provides a comprehensive toolkit for developers to handle various web development tasks efficiently, such as URL routing and database management. With built-in features like authentication, security, and templating, Django enables developers to focus on building high-quality web applications without getting bogged down by repetitive tasks. (Sara A. Metwalli, 2023)

- **TensorFlow and TensorFlow Hub:**

TensorFlow is an open-source machine learning framework developed by Google for building and training neural networks. It offers a flexible ecosystem for constructing complex machine learning models, implementing deep learning algorithms, and performing numerical computations efficiently (tensorflow, 2023). TensorFlow Hub is a library and platform that allows users to discover, share, and reuse pre-trained machine learning models and embeddings to enhance their TensorFlow projects. (tensorflow, 2023)

- **Keras:**

Keras is a high-level neural networks API written in Python, designed to enable fast experimentation with deep learning models. It provides a user-friendly interface for building and training neural networks, allowing developers to prototype and deploy deep learning models with ease. Keras supports various backend engines, including TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK), making it a versatile choice for deep learning projects. (Javapoint, 2023)

- **Pandas:**

Pandas is a powerful data manipulation and analysis library for Python, built on top of NumPy. It offers intuitive data structures like Data Frames and Series, along with a vast array of functions for data cleaning, transformation, aggregation, and visualization. Pandas simplifies the handling of structured data, making it an indispensable tool for data scientists, analysts, and developers working with tabular data.

- **Visual Studio Code (VS Code):**

Visual Studio Code is a lightweight, cross-platform source code editor developed by Microsoft. It provides a rich set of features for code editing, debugging, and version control, along with an extensive marketplace of extensions to customize and enhance its functionality. With its built-in support for various programming languages and frameworks, including Python, VS Code is widely used by developers for software development across different platforms. (Microsoft, 2024)

- **Jupyter Notebook:**

Jupyter notebook is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It supports various programming languages, including Python, R, and Julia, and provides an interactive computing environment for data analysis, machine learning, scientific research, and education. Jupyter Notebooks are widely used in data science and research communities for prototyping, experimentation, and collaboration. (Jupyter, 2015).

2.2.2. DEEP LEARNING

Deep Learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. Deep learning models can be taught to perform classification tasks and recognize patterns in photos, text, audio, and other various data. It is also used to automate tasks that would normally require human intelligence, such as describing images or transcribing audio files.

Deep Learning, particularly Convolutional Neural Networks (CNNs), serves as the backbone of my final year project by enabling automatic detection of crop diseases through leaf analysis. Harnessing deep learning algorithms, the project empowers farmers with timely and precise information about identified diseases, facilitating early intervention and minimizing crop damage. CNNs excel at feature extraction and pattern recognition, allowing them to learn intricate patterns indicative of various diseases, thus promoting precision agriculture by enabling targeted treatment of affected crops.

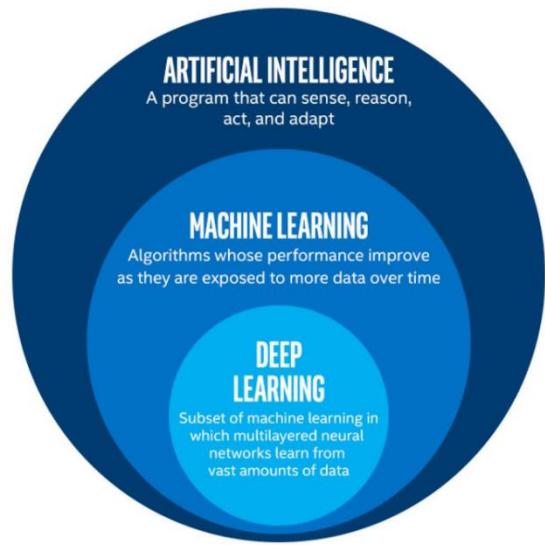


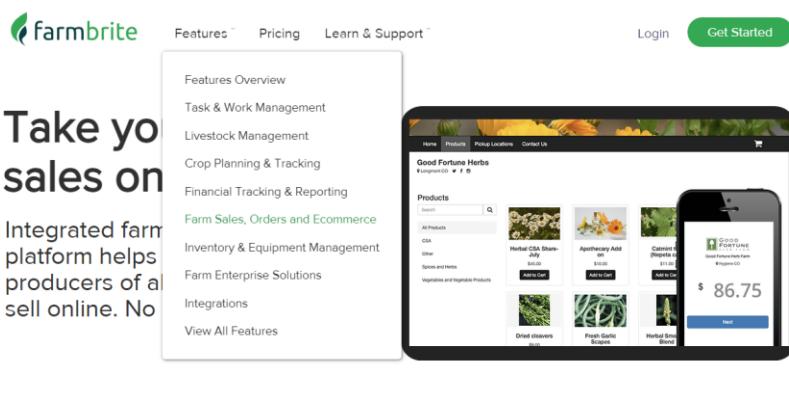
Figure 5: Project as Solution: Deep Learning

For my final year project, the aim is to develop a practical solution using cutting-edge technologies such as deep learning and computer vision to tackle real-world challenges encountered by farmers in combating crop diseases. [Transfer Learning](#), a method in machine learning where a pre-existing model is repurposed for another task, is employed here. This approach is favoured in deep learning as it facilitates the training of deep neural networks with less data compared to building a model from scratch, thereby reducing the need for extensive compute resources and time (Alexander S. Gillis, 2023). Utilizing the [EfficientNetB0 Model](#) as a starting point for this project capitalizes on the knowledge embedded within pretrained models, accelerating the training process, and potentially enhancing the effectiveness of our solution in addressing crop diseases.

2.3. SIMILAR PROJECTS

2.3.1. FARM BRITE

Farm Brite is an agricultural technology (AgTech) company with a mission to assist individuals in starting and succeeding in agriculture. It operates as an e-commerce platform designed to aid farmers and producers across various markets. Farmers can purchase a wide range of products, including machinery and equipment, through the platform. (Farmbrite, 2023)



Easy to setup, simple to manage online farm sales software.
Automate your sales, orders and inventory.

[Figure 6: Similar Project 1: Home page of Farm Brite](#)

2.3.2. FARM KART

The Farm kart application endeavors to create a technology-driven ecosystem that fosters agricultural excellence by providing affordable and accessible farming solutions for farmers. This innovative platform serves as an AI-enabled ecommerce hub, offering a diverse range of globally sourced agricultural products. With the Farmkart application, users can seamlessly navigate and purchase a comprehensive array of products, including seeds, fertilizer, pesticides, and farm machinery, all from the convenience of their home. (farmkart, 2023)

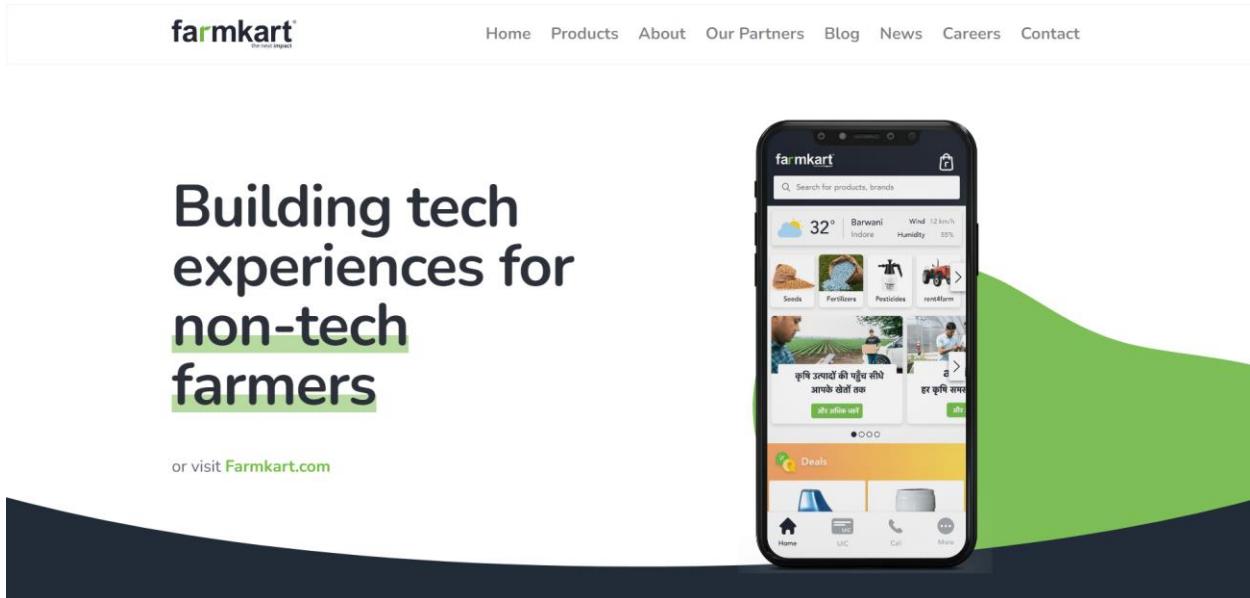


Figure 7: Similar Project 2: Homepage of Farm Kart Application

2.3.3. DE HAAT

De Haat Farmer is a comprehensive platform that provides a variety of services, one of which is crop reminders. This feature allows farmers to receive timely notifications about crucial events related to their crops. The app offers valuable information on chemicals, crop health, and the usage of high-quality agricultural products. In addition to serving as an information hub, the platform enables farmers to engage in the buying and selling of various agricultural products. Overall, De Haat Farmer aims to empower farmers by offering a range of tools and resources to enhance their crop management and trade activities. (DeHaat, 2023)



Solution for farmers

DeHaat Farmer App is a onestop platform that provides multiple services, including frequent crop reminders, voice calls in regional languages, crop advisories, weather reports, local mandi rates, etc. to over 1.4 million farmers in 12 major agri-based states in India.

Farmers can use the App to get direct consultations from Agri-experts on chemical dosage, crop health advice, and the use of high-quality input products. Purchase and sale of input or output products can also be done through the app with convenience.

Farmers can avail customised advisory services based on real-time information through the app right at their fingertips.



Figure 8: Similar Project 3: Homepage of DeHaat Application

2.4. COMPARASIONS

Feature	Farm Brite	Farm Kart	De Haat	Farmers Bag
Login	✓	✓	✓	✓
Register	✓	✓	✓	✓
User-friendly interface	✓	✓	✓	✓
Disease Detection	✗	✗	✗	✓
Recommend Product	✓	✗	✗	✓
Study Place	✗	✓	✗	✓
Disease Insights	✗	✗	✗	✓
Shopping list	✓	✓	✓	✓
Buy machinery equipment/products	✓	✗	✗	✓
Rating Review	✓	✓	✓	✓
Payment Integration	✓	✓	✓	✓

Table 1: Similar Project Comparison

3. DEVELOPMENT

3.1. CONSIDERED METHODOLOGY

3.1.1. SPIRAL METHODOLOGY

The spiral model, purposed by Boehm, is a pivotal Software Development Life Cycle (SDLC) known for effective risk management. It blends prototyping's flexibility with the structure of the linear sequential model, allowing rapid software development. The process works in a loop, each stage adjustable. Early iterations may involve paper models or prototypes, evolving into more complete versions with each cycle. This dynamic approach ensures continuous improvement and maturity in software development. (Doshi, et al., 2021).

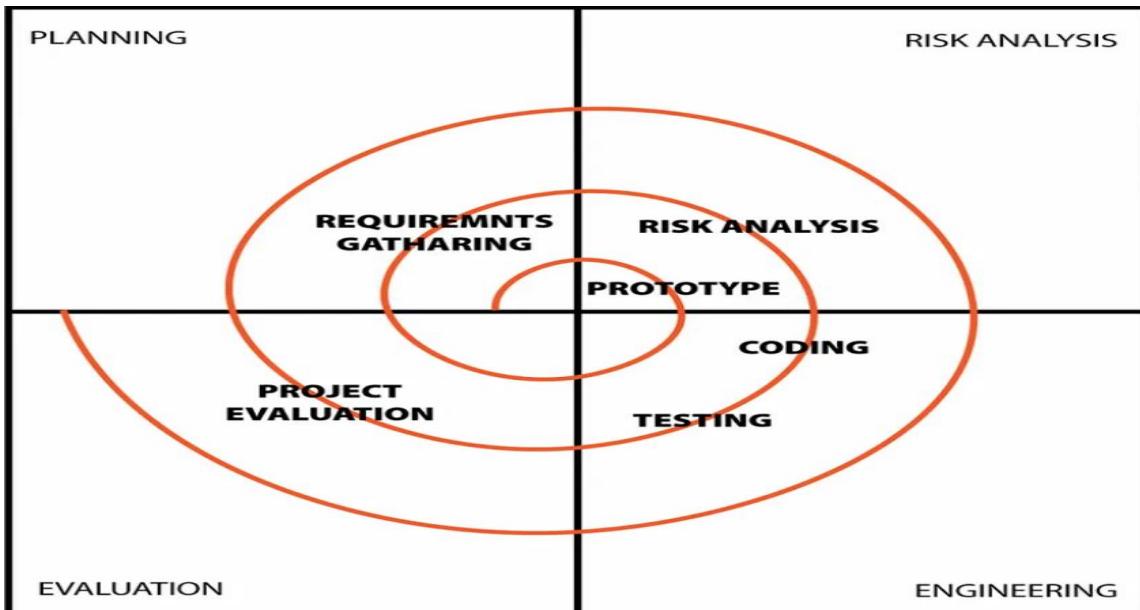


Figure 9: Considered Methodology: Spiral

Reason for not choosing Spiral Methodology:

- Detailed planning and constant reassessment may result require more resources,
- Complexity may not fully benefit a small-scale project,
- Resource-intensive nature may pose challenges for a small project,
- Detailed planning and reassessment may be resource-intensive,
- Iterative nature may extend the project timeline, potentially conflicting with scope and deadlines.

Other considered methodologies are available in the appendix section.

Appendix: [7.4.2.1. READING FOR OTHER CONSIDERED METHODOLOGIES]

3.2. PHASES OF SELECTED METHODOLOGY

As per 'Philippe Kruchten', the lead architect of the Rational Unified Process, in his book "Rational Unified Process: An Introduction", the Rational Unified Process is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, which follows four phases for the successful completion of the project: Inception, Elaboration, Construction, and Transition (Krutchen, 2004).

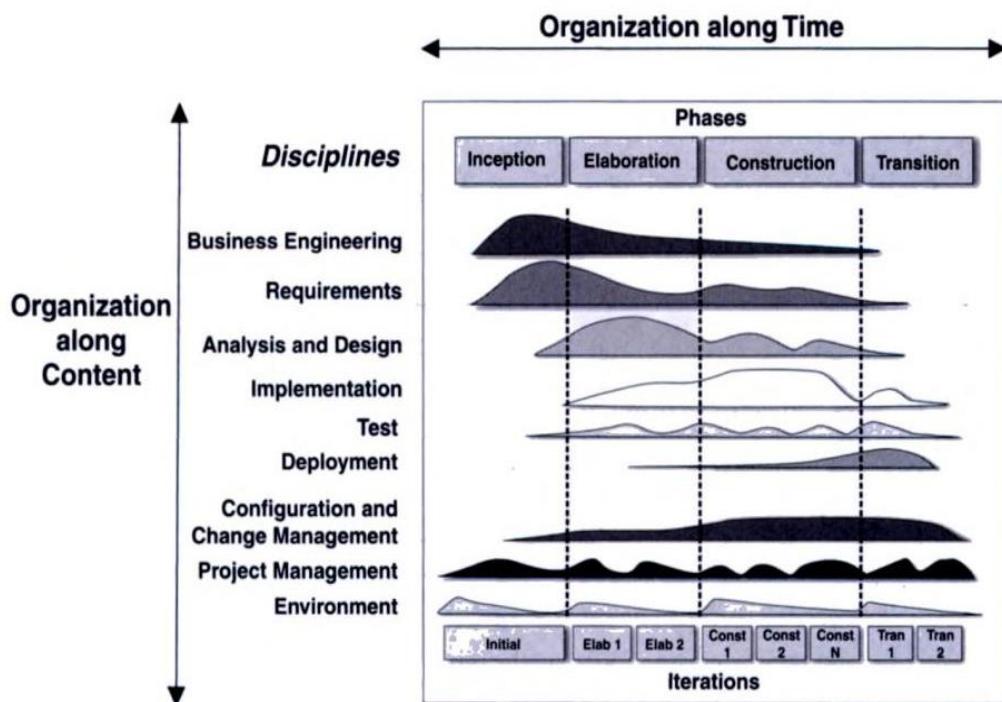


Figure 10: Phases of Methodology (Krutchen, 2004)

Appendix: [\[7.4.1. RATIONAL UNIFIED PROCESS \(RUP\) PHASES\]](#)

3.4. SURVEY RESULTS

3.4.3. PRE-SURVEY RESULTS

Farmers' Bag Application

The Farmers Bag Application project is a friend to farmers who want to keep their plants healthy and productive. It can spot plant diseases by looking at the leaves, suggest the best medicines to treat them, and guide the farmers through the steps to get rid of the diseases. It also makes it easy for the farmers to buy the medicines they need and the crops they want from the app itself.

Figure 11: Pre-Survey: Farmer Bag Application

Question no 1: Do you think Farmers' Bag Application will be helpful for farmers?

Do you think Farmers' Bag Application will be helpful for farmers?

Copy

50 responses

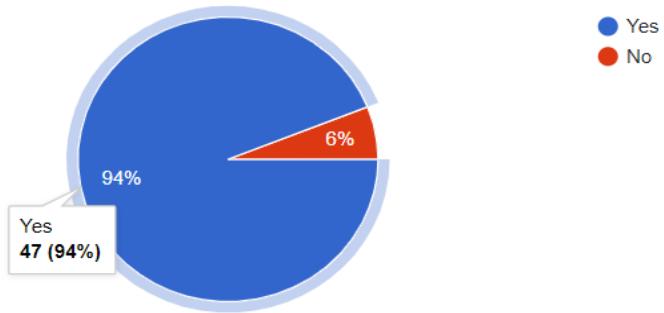


Figure 12: Pre-Survey Result: Question 1

Other screenshots of Pre-Survey results are provided in appendix section.

Appendix: [\[7.1. PRE-SURVEY RESULTS\]](#)

3.4.2. POST-SURVEY RESULTS

Farmer's Bag Post-Survey Questions

B I U ↲ ✖

The Farmers Bag Application project is a friend to farmers who want to keep their plants healthy and productive. It can spot plant diseases by looking at the leaves, suggest the best medicines to treat them, and guide the farmers through the steps to get rid of the diseases. It also makes it easy for the farmers to buy the medicines they need and the crops they want from the app itself.

Key Features of the Farmer Bag Application:

- **Disease Detection:** Utilizes CNNs (Transfer Learning) to spot plant diseases early by analyzing leaf images.
- **Educational Resources:** Offers extensive information on disease identification and management.
- **Guided Treatment:** Provides step-by-step guidance for effective disease management.
- **Marketplace Integration:** Connects farmers with suppliers for convenient purchases.
- **User-Friendly Interface:** Intuitive design for easy navigation by farmers of all levels.
- **Secure Payment Gateway:** Ensures safe transactions via Khalti integration.
- **Data Security Measures:** Implements robust authentication for user data protection.

Figure 13: Farmer's Bag Post-Survey

Question 1: What type of training or support do you think will be required to help farmers for using Farmer Bag Application.

What type of training or support do you think will be required to help Farmers for using Farmer Bag Application?  Copy

43 responses

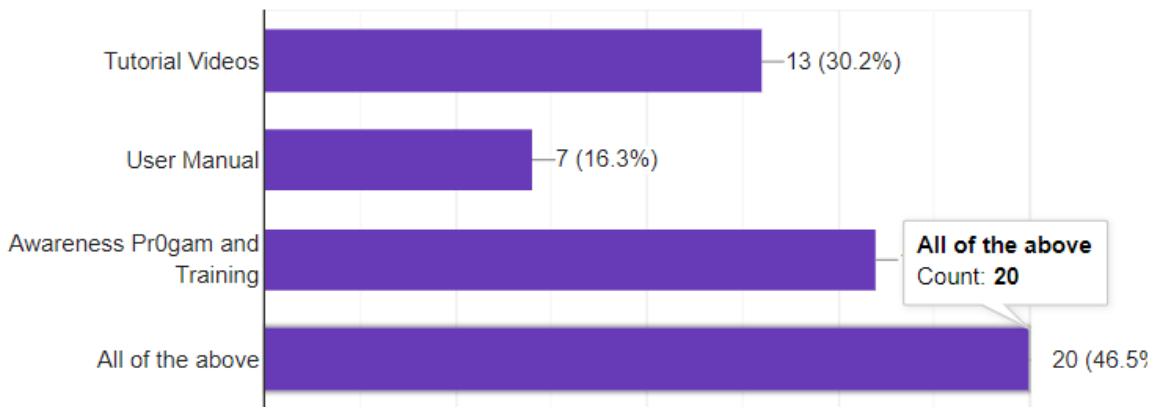


Figure 14: Post-Survey: Question 1

Other screenshots of post-survey results are provided in appendix section.

Appendix: [\[7.2. POST-SURVEY RESULTS\]](#)

3.4. REQUIREMENT ANALYSIS

Requirement Engineering, often called Requirement Analysis, Requirements Gathering, or Requirements Capture in software engineering, is a crucial part of the Software Development Life Cycle (SDLC) to achieve successful project development. It is the process of defining user expectations for new or changed software. The task to determine the actual needs or conditions for a new or changed product/project, identify potentially conflicting requirements, analyze, validate, manage, and document software or system requirements are all included in requirement analysis. The requirements are gathered by identifying the personal/customer's needs, evaluating system feasibility, performing technical analysis, determining functions per system elements, establishing the project schedule and limitations, and creating system definitions (i.e., UML models).

The gathered requirements are documented in the Software Requirements Specification (SRS) document and are available in the appendix section.

Appendix: [7.3.1. READINGS FOR SOFTWARE REQUIREMENT SPECIFICATION \(SRS\)](#)

3.4.1. WORK BREAKDOWN STRUCTURE

Work Breakdown Structure (WBS) is a software engineering process that breaks down a complex task into its component parts. It's often used in larger software projects to help organize work, identify bottlenecks, and improve communications (Organ, et al., 2022)

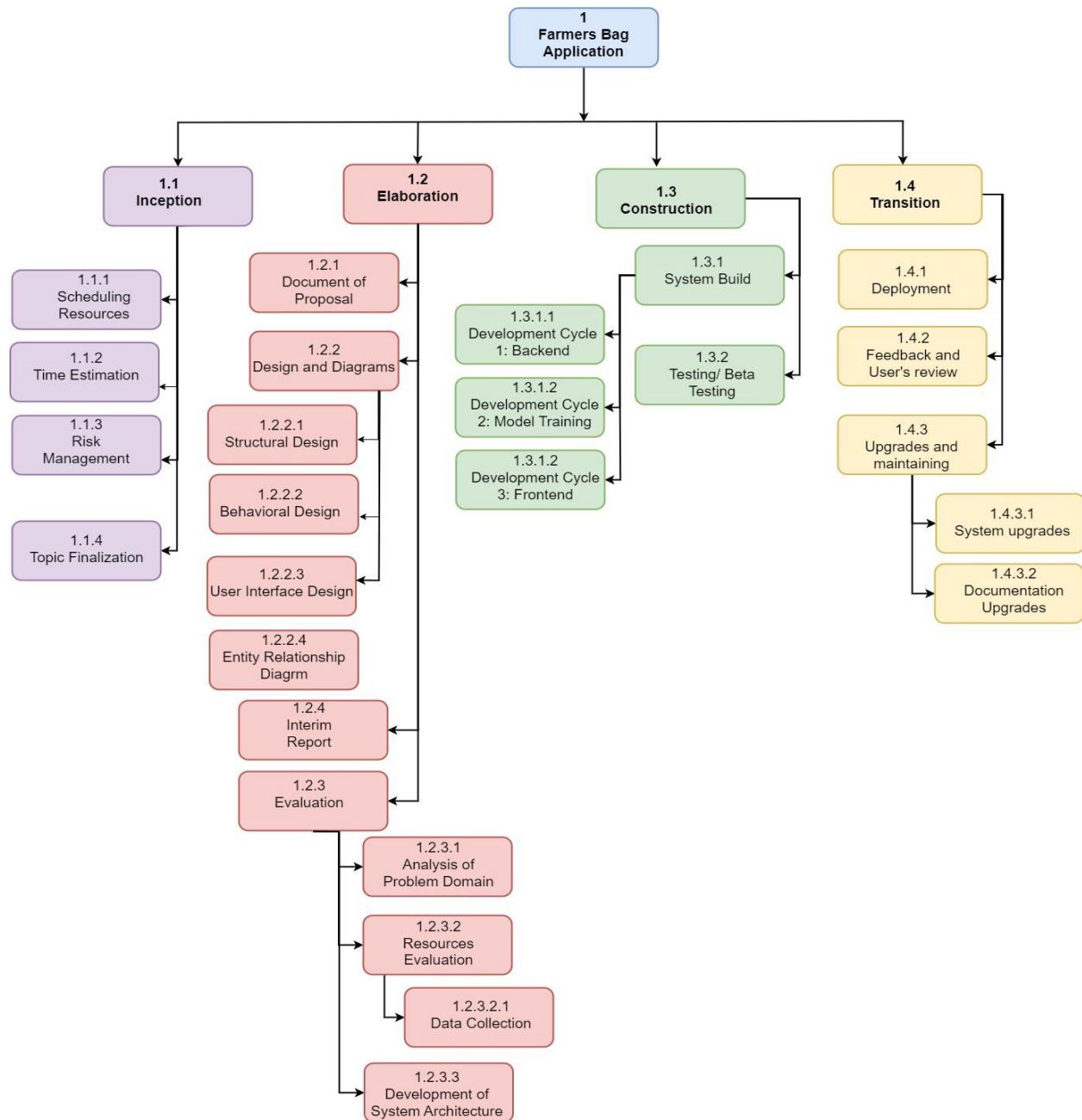
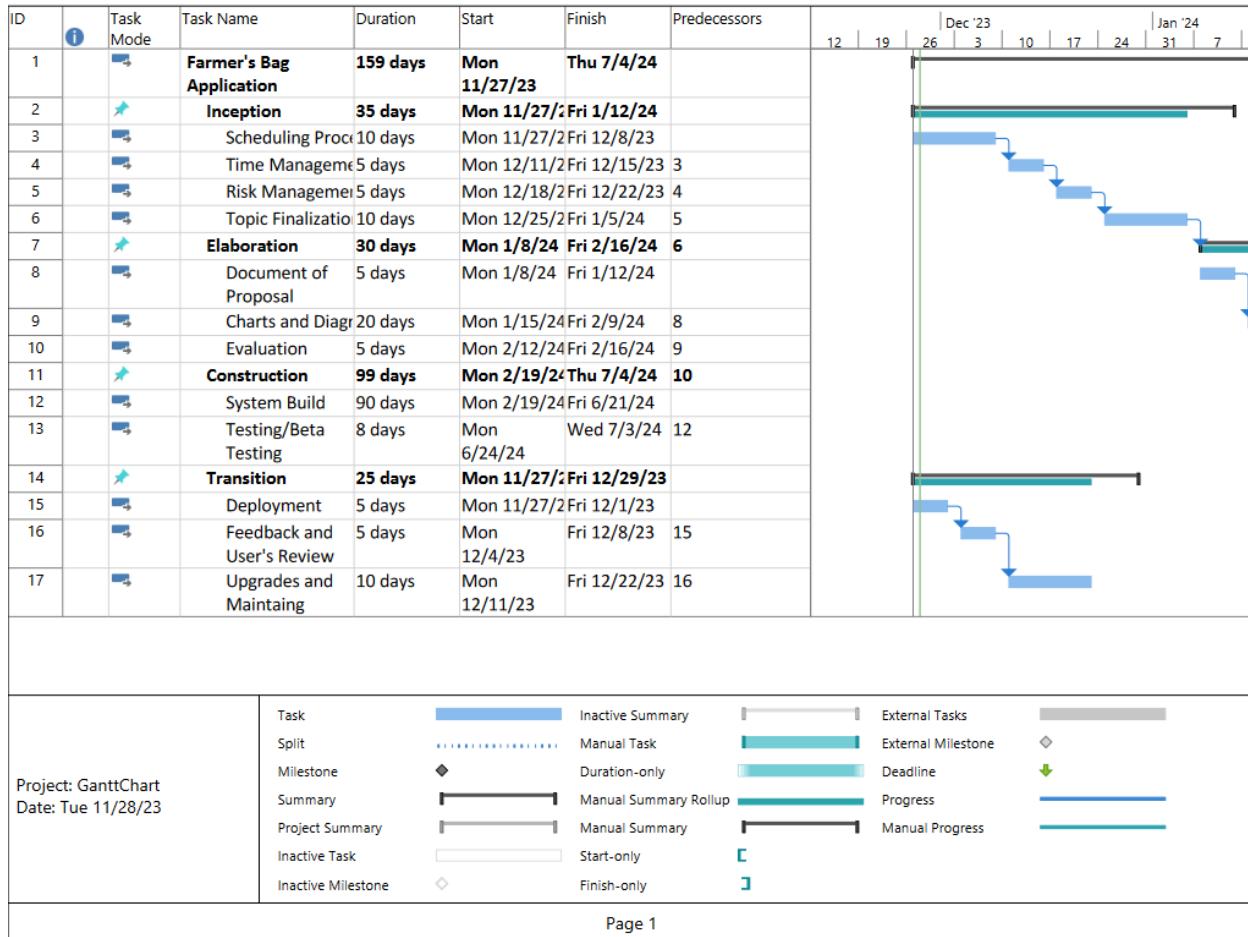


Figure 15: Work Breakdown Structure

3.4.1. GANTT CHART



The diagram below illustrates the proposed Gantt Chart representation for the project. In this Gantt chart, the time for each task was estimated. As the development process progressed, tasks were completed accordingly. Despite the rough time estimates, the project was completed within the planned timeline. The Gantt Chart will be updated at the project's conclusion to reflect the actual number of days required to finish tasks, as it is challenging to pinpoint the exact duration during the planning phase.

3.5. DESIGNS

3.5.1. APPLICATION LOGO



Figure 16: Design: Application Logo

3.5.2. WIREFRAME

Wireframes in UI/UX Design are crucial for outlining digital application structures, addressing user needs early, and obtaining stakeholder approval before the creative phase. Despite lacking design details, they facilitate prototyping usability tests and foster team collaboration. Challenges include translating them to final designs and handling added content, but overall, wireframes are indispensable for early approvals and optimizing resources in later project stages. (UX, 2022)

3.5.2.1 WIREFRAME: HOME PAGE

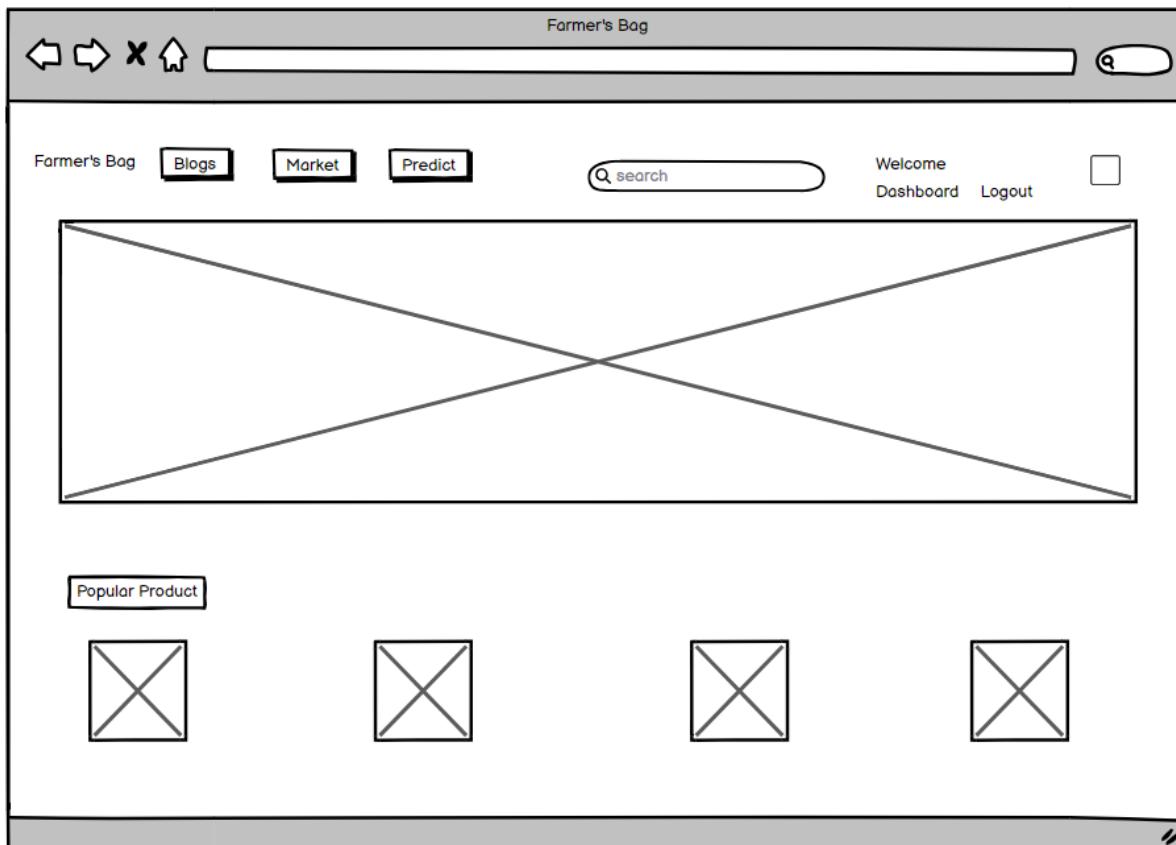


Figure 17: Wireframe: Home Page

3.5.2.2. WIREFRAME: CHANGE PASSWORD

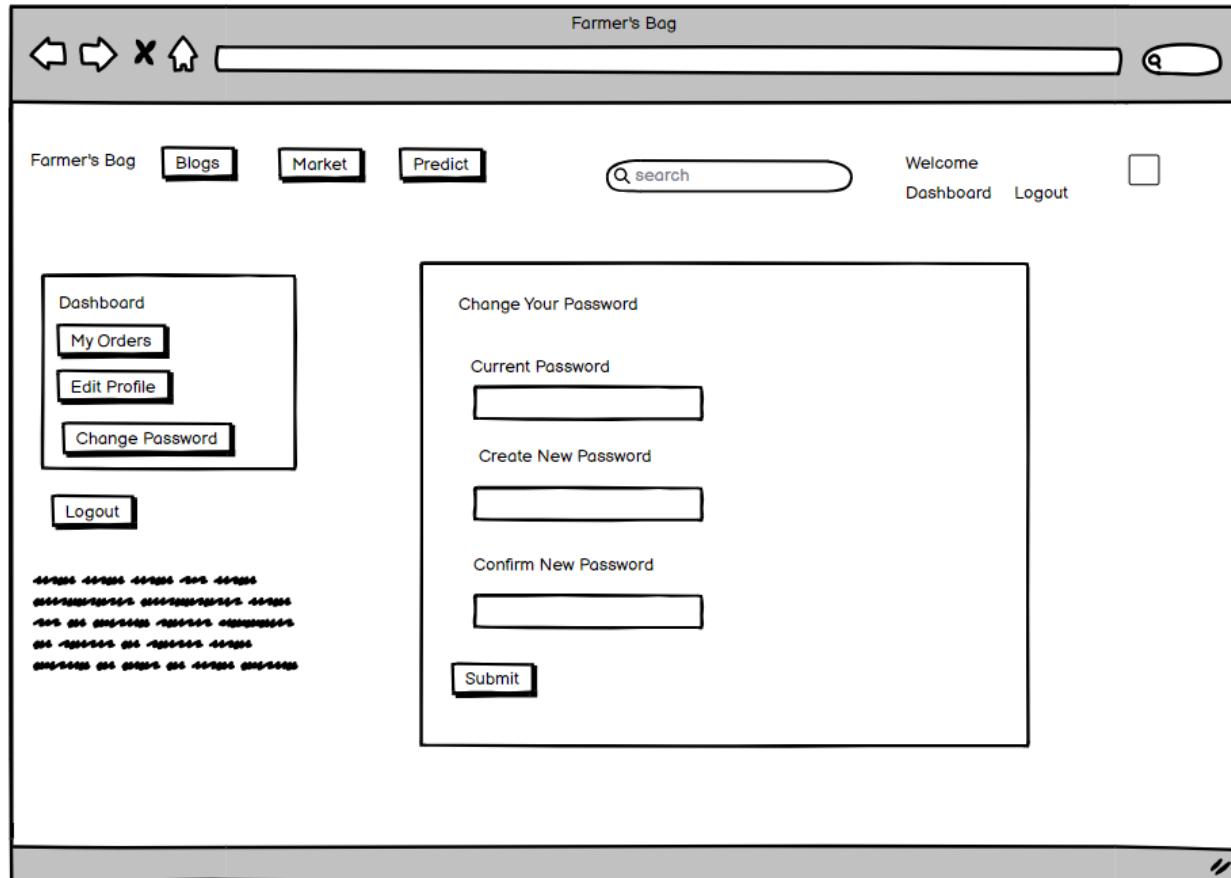
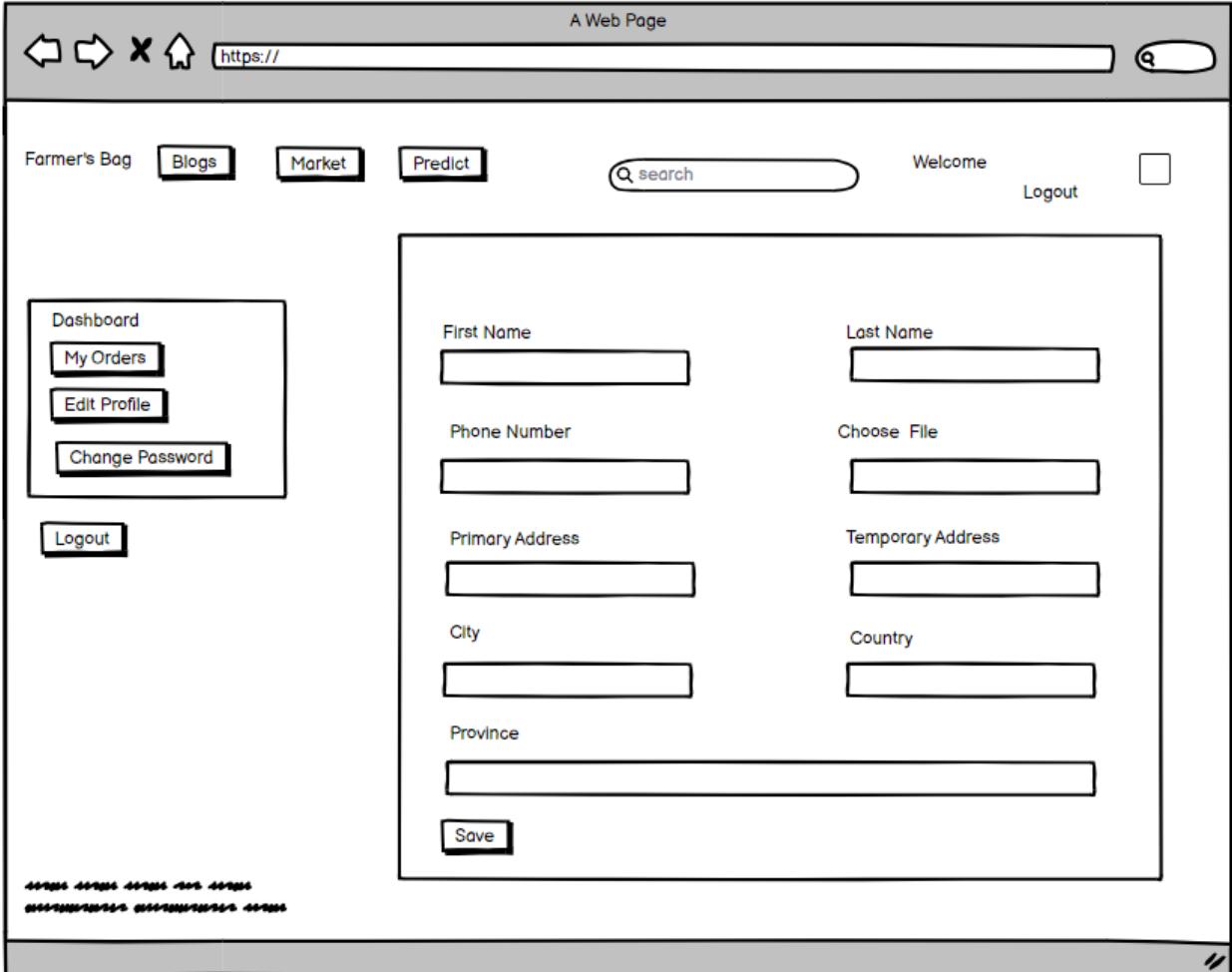


Figure 18: Wireframe: Change Password Page

3.5.2.3. WIREFRAME: UPDATE PROFILE



The wireframe depicts a web browser window titled "A Web Page" with a URL bar showing "https://". The header includes standard navigation icons (back, forward, search, etc.) and a "Welcome" message with a "Logout" link. Below the header, there are four main menu items: "Farmer's Bag", "Blogs", "Market", and "Predict". A search bar is also present. On the left side, a sidebar labeled "Dashboard" contains buttons for "My Orders", "Edit Profile" (which is highlighted), and "Change Password", along with a "Logout" button. The main content area is a large form for updating profile information. It features two columns of input fields. The left column includes "First Name" (text input), "Phone Number" (text input), "Primary Address" (text input), "City" (text input), and "Province" (text input). The right column includes "Last Name" (text input), "Choose File" (file input), "Temporary Address" (text input), "Country" (text input), and a "Save" button. At the bottom of the page is a decorative footer section with a repeating pattern of small icons.

Figure 19: Wire Frame: Update Profile Page

3.5.2.4. WIREFRAME: CHECKOUT PAGE

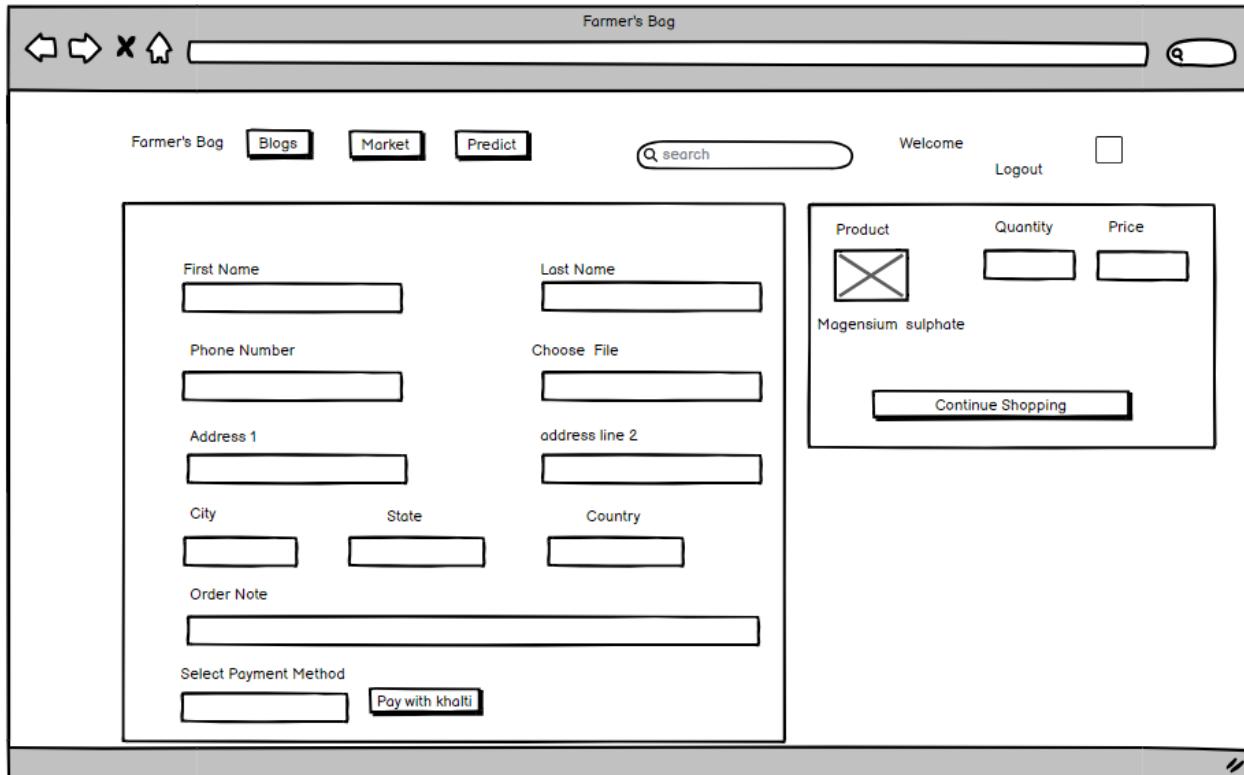


Figure 20: Wire Frame: Checkout Page

3.5.2.5. WIREFRAME: CART PAGE

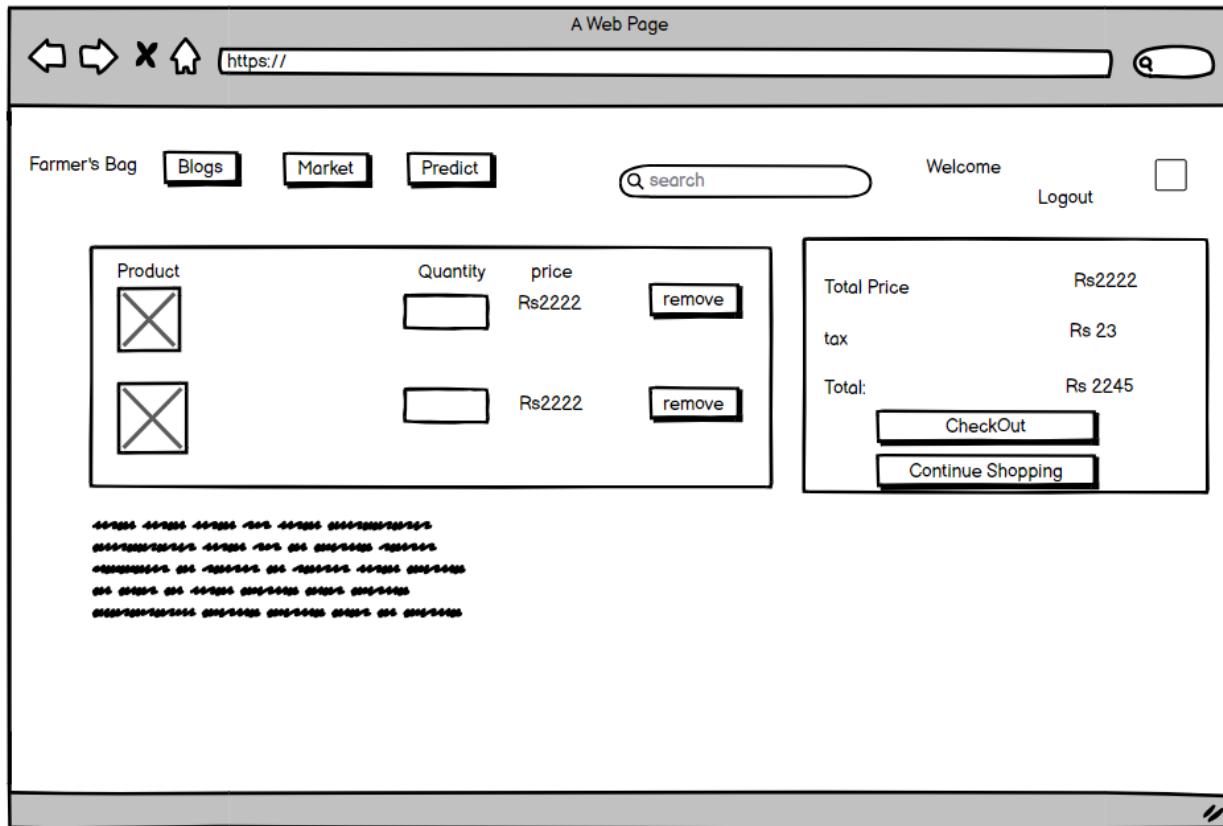


Figure 21: Wire Frame: Cart Page

3.5.2.6. WIREFRAME: PRODUCT DESCRIPTION AND RATING PAGE

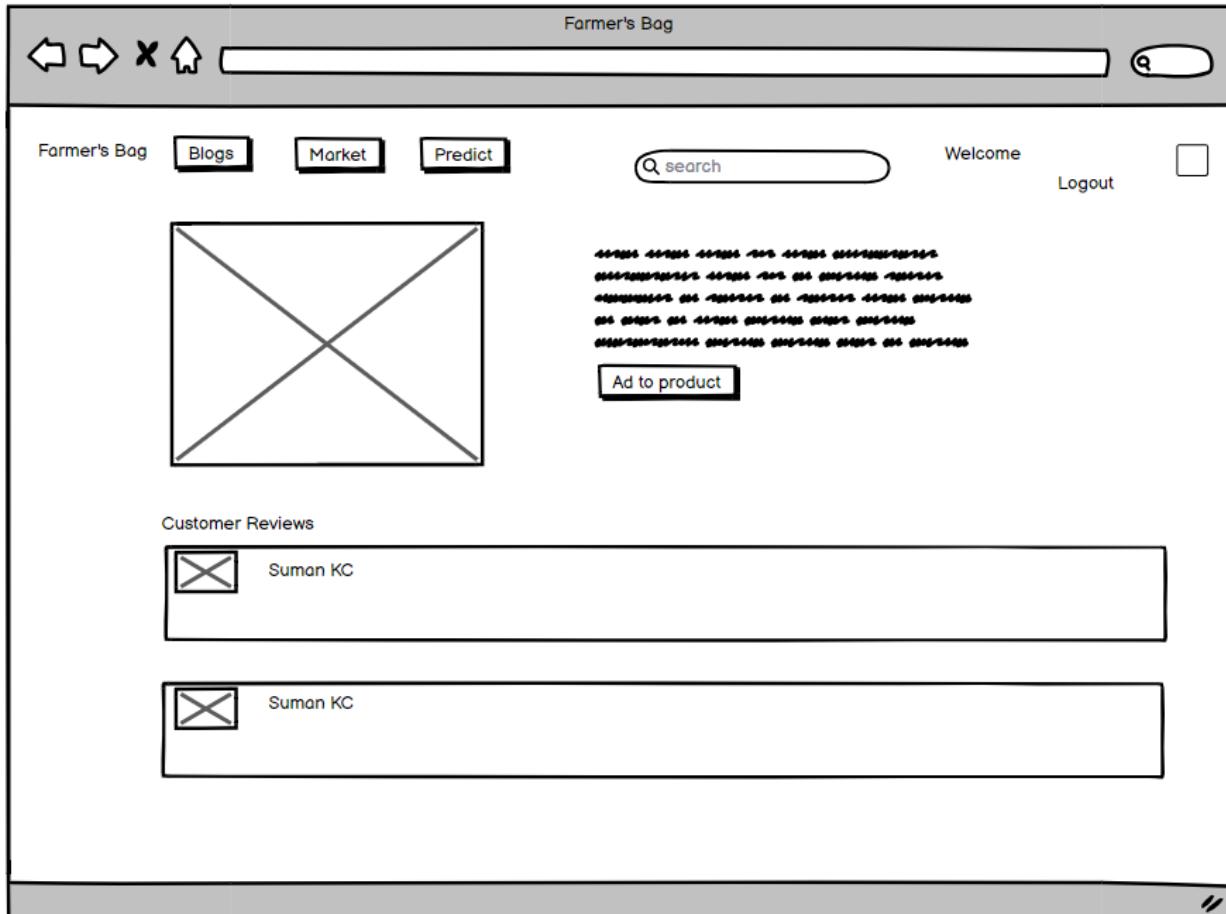


Figure 22: Wire Frame: Product Description and Review Page

3.5.2.7. WIREFRAME: BLOG PAGE

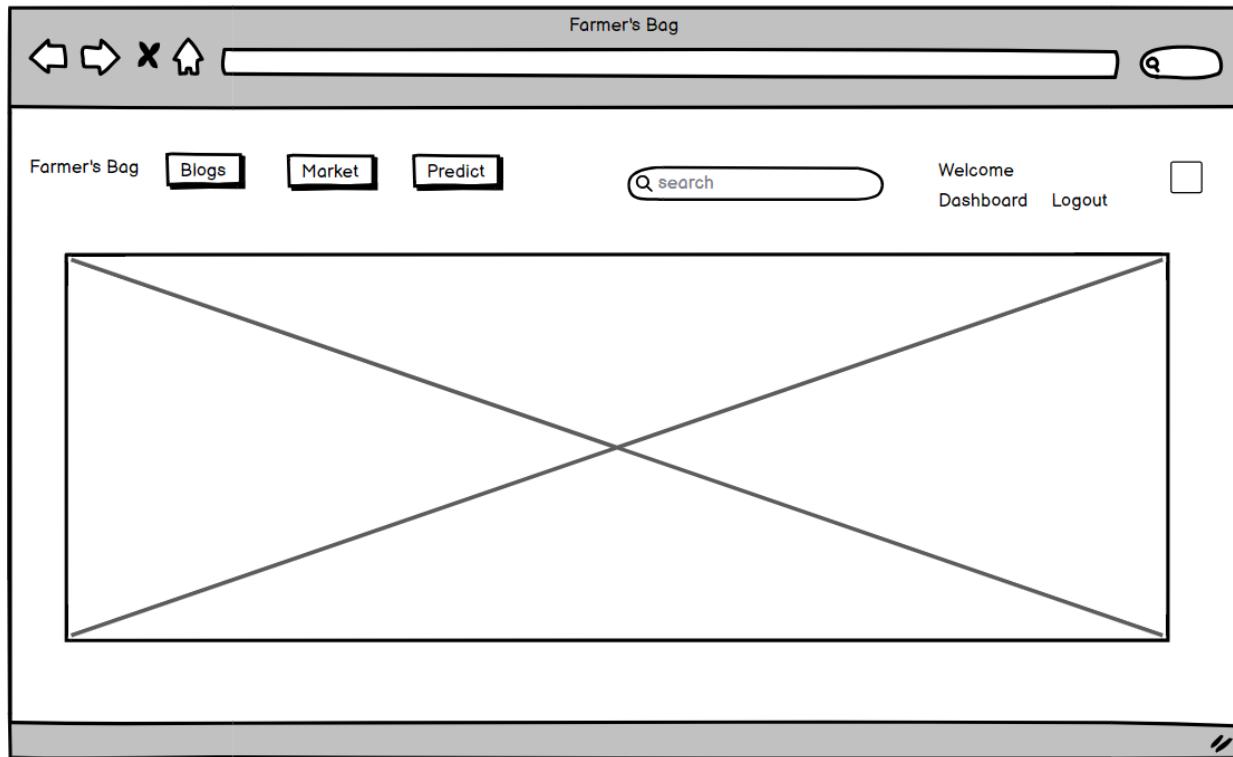


Figure 23: Wire Frame: Blog Page

3.5.2.9 WIREFRAME: ADMIN VIEW USERS



Figure 24: Wire Frame: View User

3.5.2.10 WIREFRAME: CREATE BLOG

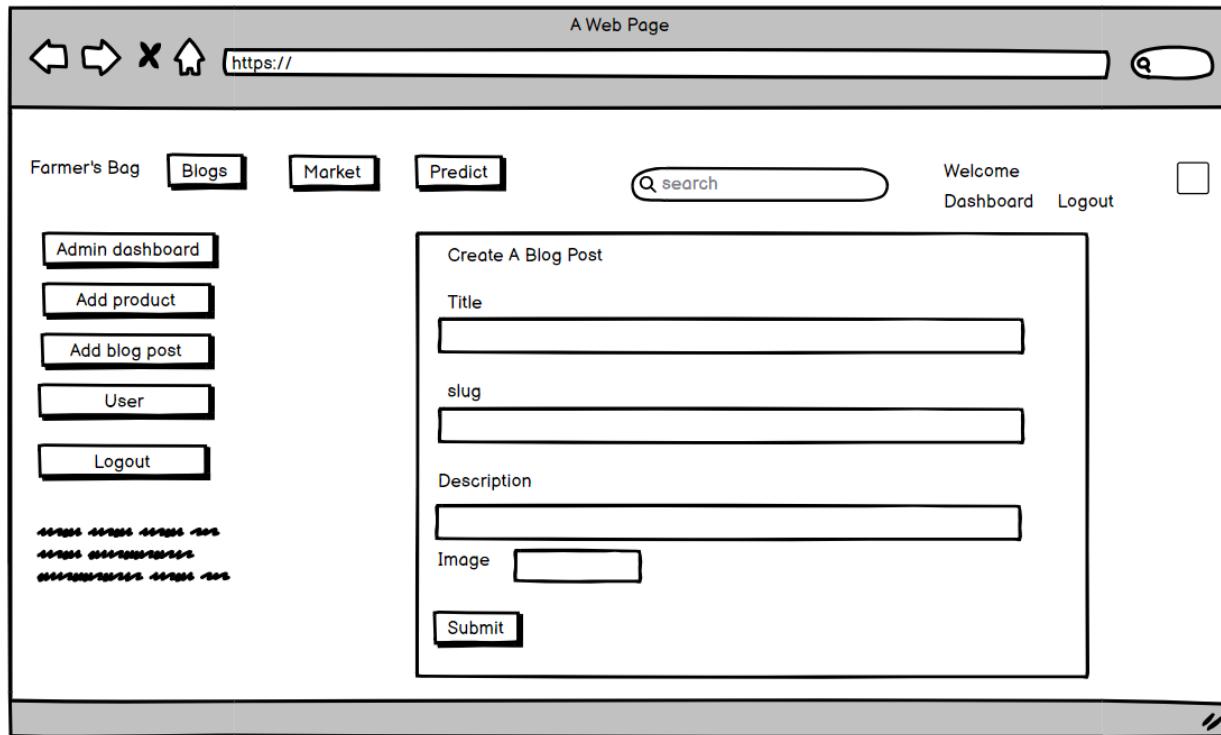


Figure 25: Wire Frame: Create Blog Post

3.5.2.11 WIREFRAME: CREATE PRODUCT

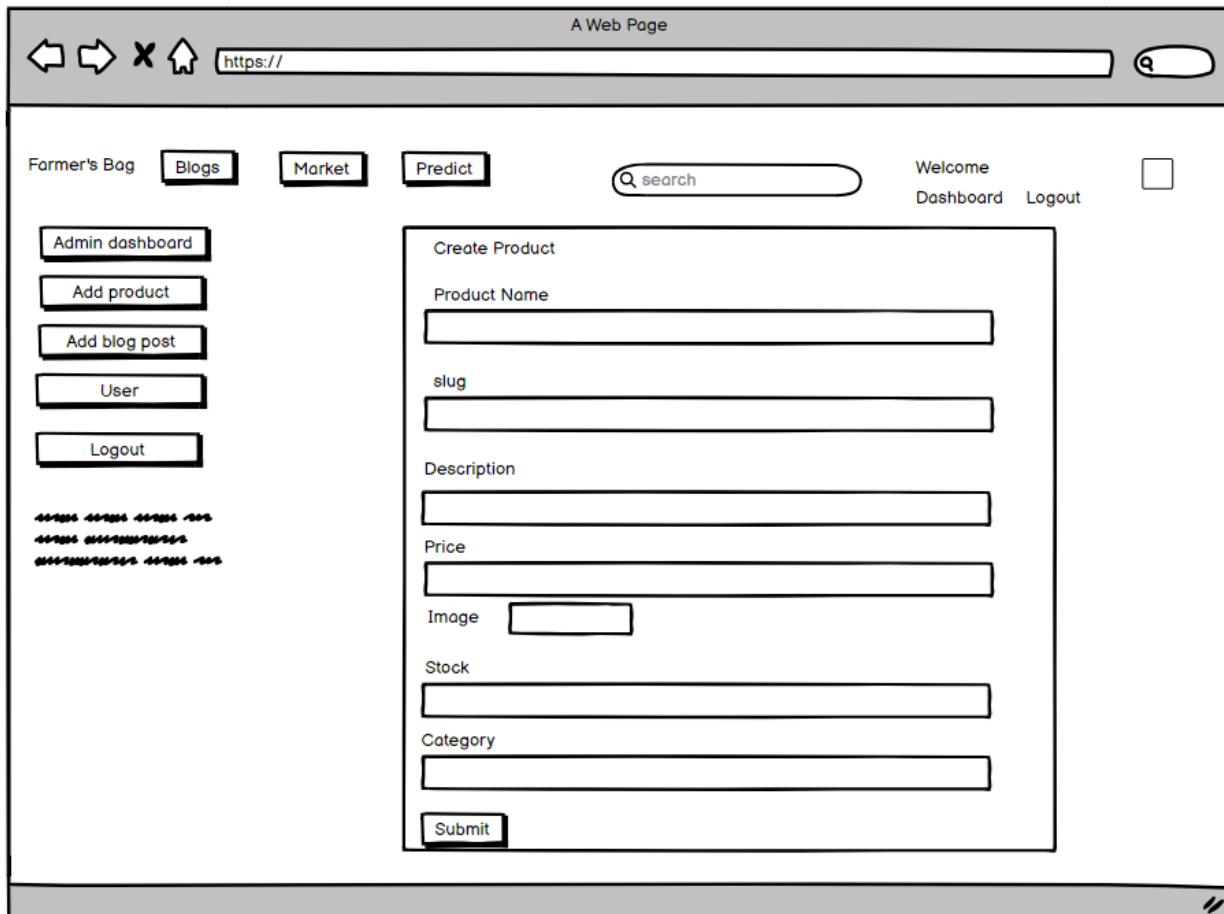


Figure 26: Wireframe: Create Product

3.5.2.12 WIREFRAME: ADMIN DASHBOARD

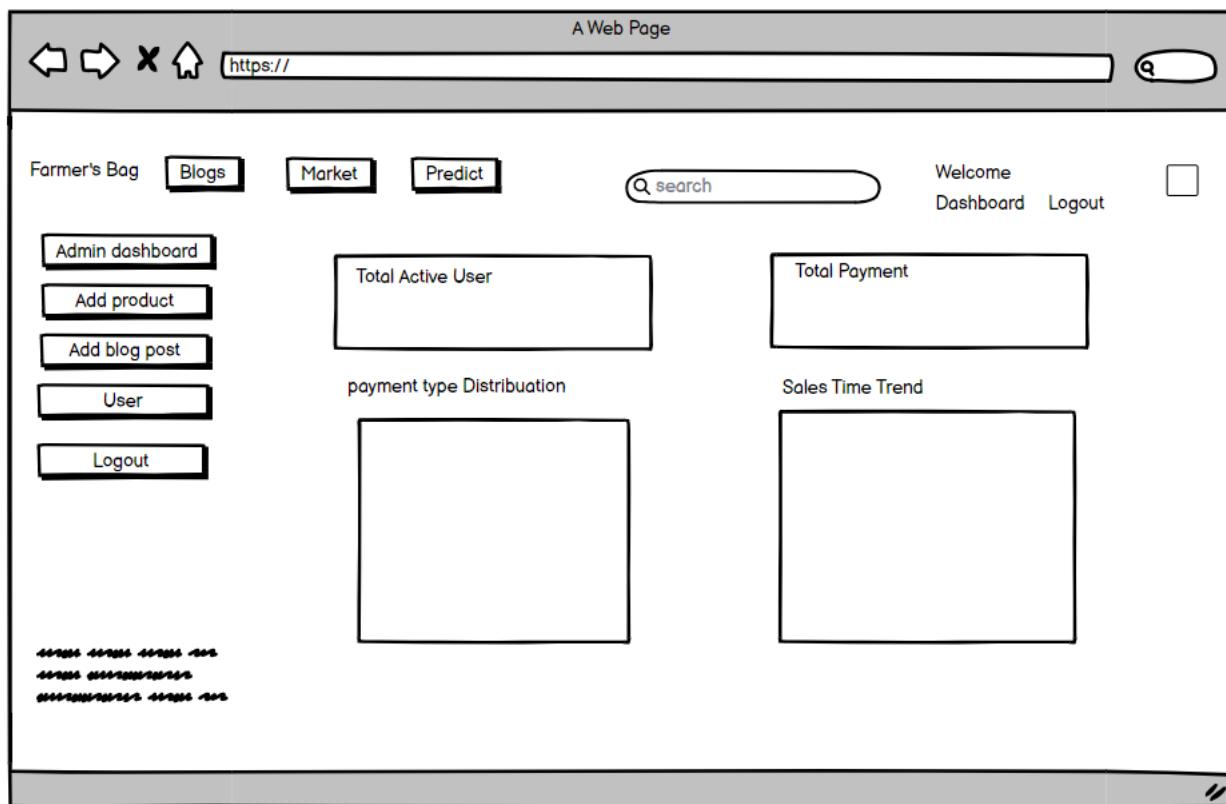


Figure 27: Wire Frame: Admin Dashboard

3.5.2.13 WIREFRAME: DASHBOARD

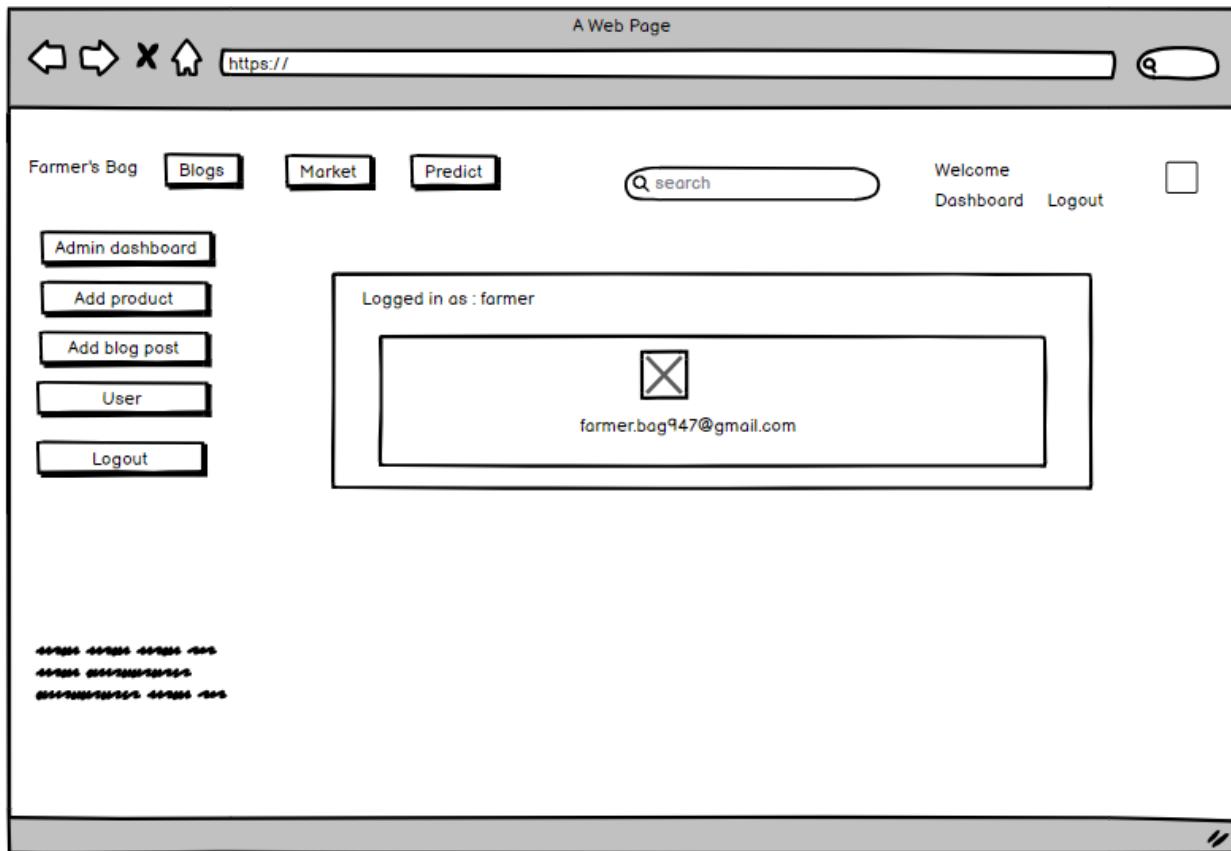


Figure 28: Wire Frame: Admin Page

3.5.2.14 WIREFRAME: FORGET PASSWORD

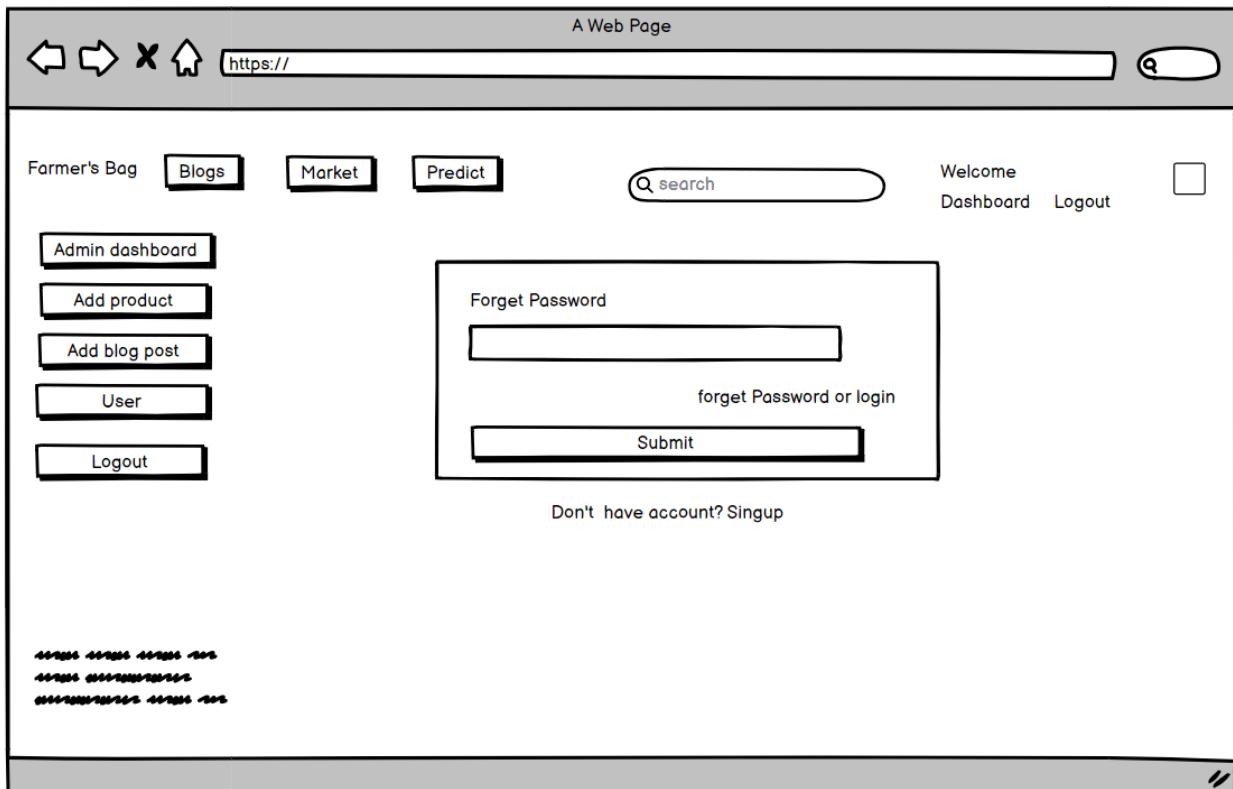


Figure 29: Wire Frame: Forgot Password

3.5.2.15 WIREFRAME: SIGN PAGE

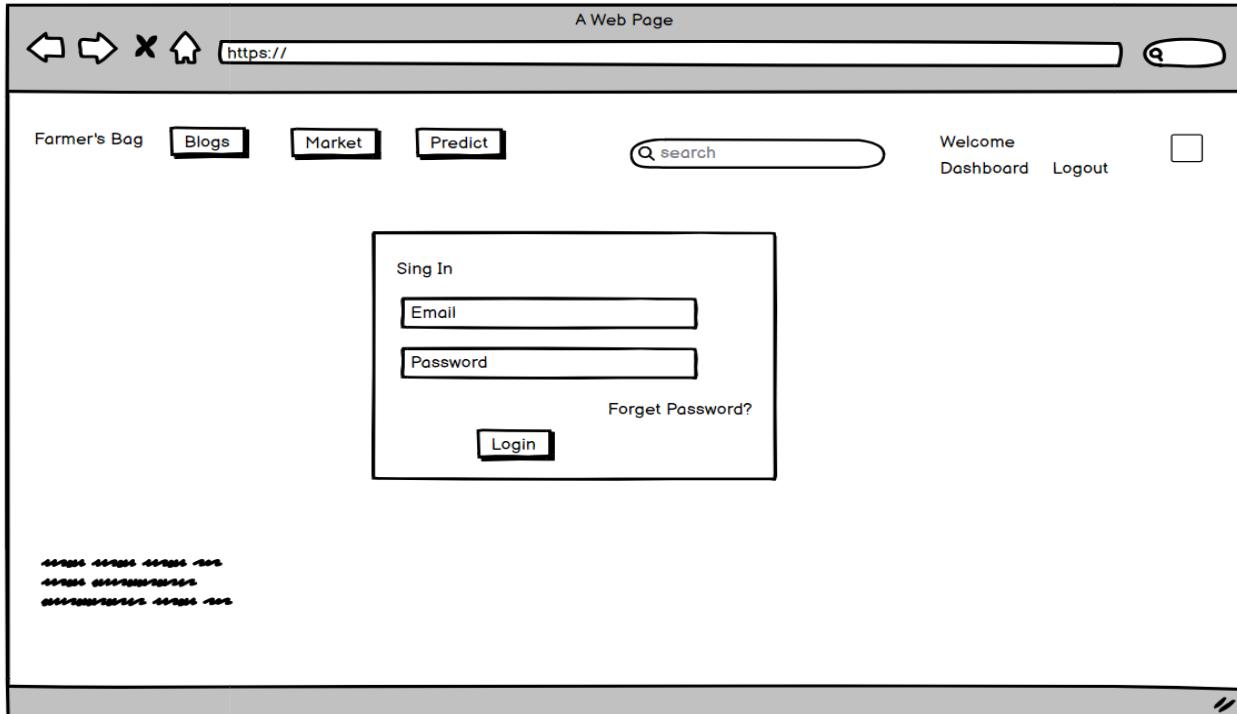
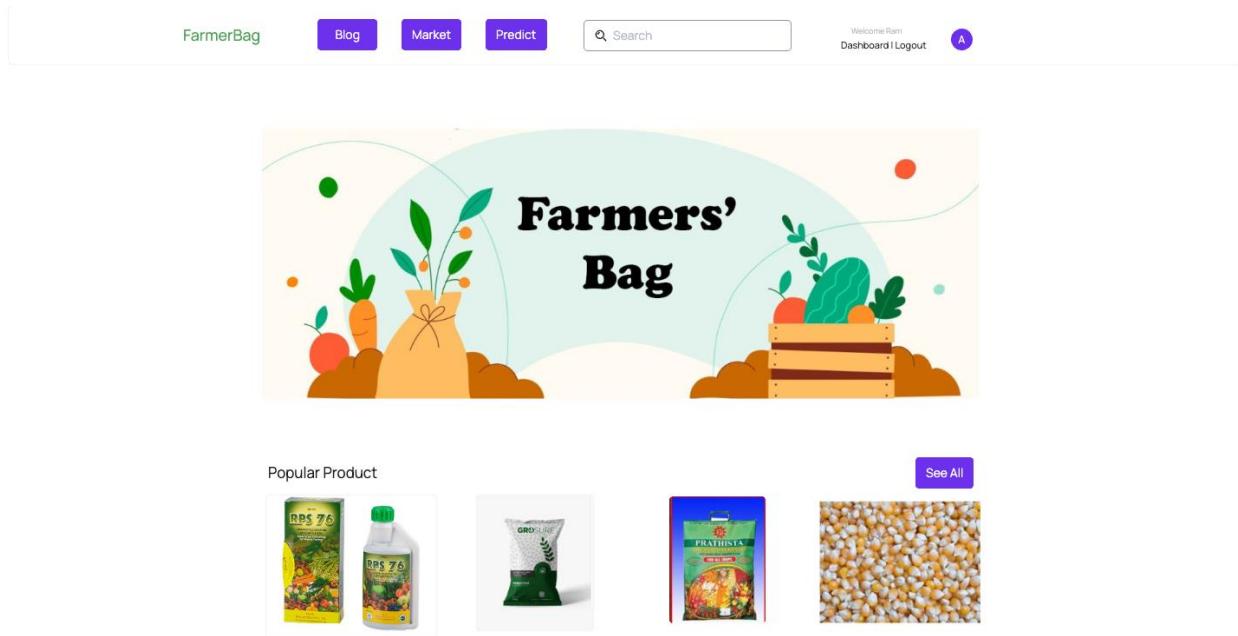


Figure 30: Wire Frame: Login Page

3.5.3. USER INTERFACE(UI)

3.5.3.1 USER INTERFACE: HOME PAGE



Popular Product



Figure 31: User Interface: Home Page

3.5.3.2. USER INTERFACE: CHECKOUT PAGE

Billing Address

First Name

Last Name

Email

Phone Number

Address Line 1

Address Line 2

City State Country

Order Note

Select Payment Method: COD -

© 2024 Farmers Bag
farmer.bag974@gmail.com +977-9744254833 Chorhi Dang-15
Social Bar

Figure 32: User Interface: Checkout Page

3.5.3.3. USER INTERFACE: EDIT PROFILE

Dashboard

My Orders

Edit Profile

Change Password

Logout

Edit your Profile

First Name

Last Name

Phone Number

Primary Address

City Hetauda

Province

Profile Picture

Temporary Address

Country Nepal

Province

© 2024 Farmers Bag
farmer.bag974@gmail.com +977-9744254833 Chorhi Dang-15
Social Bar

Figure 33: User Interface: Edit Profile

3.5.3.4 USER INTERFACE: CHANGE PASSWORD

The screenshot shows a web application interface for changing a password. At the top, there is a navigation bar with links for 'FarmerBag', 'Blog', 'Market', 'Predict', and a search bar. On the right side of the navigation bar, there are links for 'Welcome Ram', 'Dashboard | Logout', and a user icon labeled 'A'. Below the navigation bar, on the left, is a sidebar menu with options: 'Dashboard' (selected), 'My Orders', 'Edit Profile', 'Change Password' (selected), and 'Logout'. The main content area contains a form titled 'Change your Password' with four input fields: 'Current Password', 'Create new password', 'Confirm new password', and a 'Submit' button. At the bottom of the page, there is a footer with copyright information: '© 2024 Farmers Bag', 'farmer.bag974@gmail.com +977-9744254833 Chorhi Dang-15', and a 'Social Bar' link.

Figure 34: User Interface: Change Password

3.5.3.5. USER INTERFACE: SIGN IN

The screenshot shows a web application interface for signing in. At the top, there is a navigation bar with links for 'FarmerBag', 'Blog', 'Market', 'Predict', and a search bar. On the right side of the navigation bar, there are links for 'Welcome Ram', 'Dashboard | Logout', and a user icon labeled 'A'. The main content area contains a 'Sign in' form with two input fields: 'Email Address' and 'Password'. Below the password field is a 'Forgot password?' link. A 'Login' button is located at the bottom of the form. At the very bottom of the page, there is a footer with copyright information: '© 2024 Farmers Bag', 'farmer.bag974@gmail.com +977-9744254833 Chorhi Dang-15', and a 'Social Bar' link.

Figure 35: User Interface: Sign in

3.5.3.6. USER INTERFACE: FORGOT PASSWORD

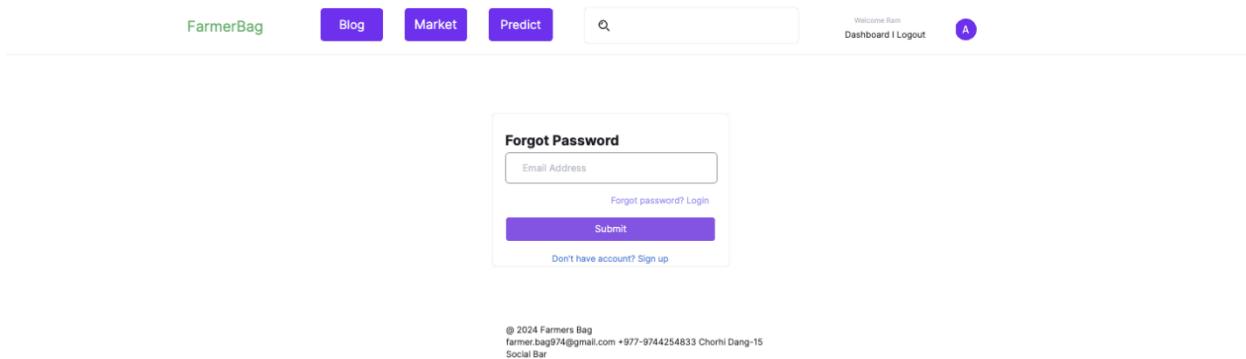


Figure 36: User Interface: Forget Password

3.5.3.7 USER INTERFACE: DASHBOARD

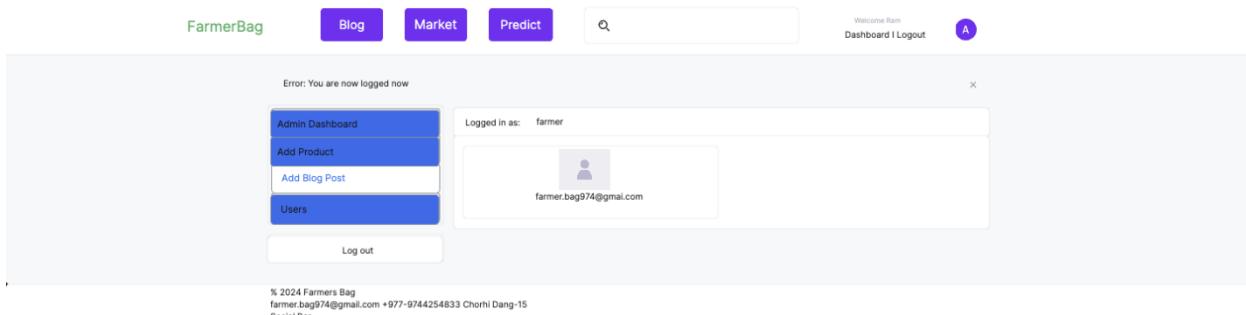


Figure 37: User Interface: User Dashboard

3.5.3.8. USER INTERFACE: PRODUCT DESCRIPTION

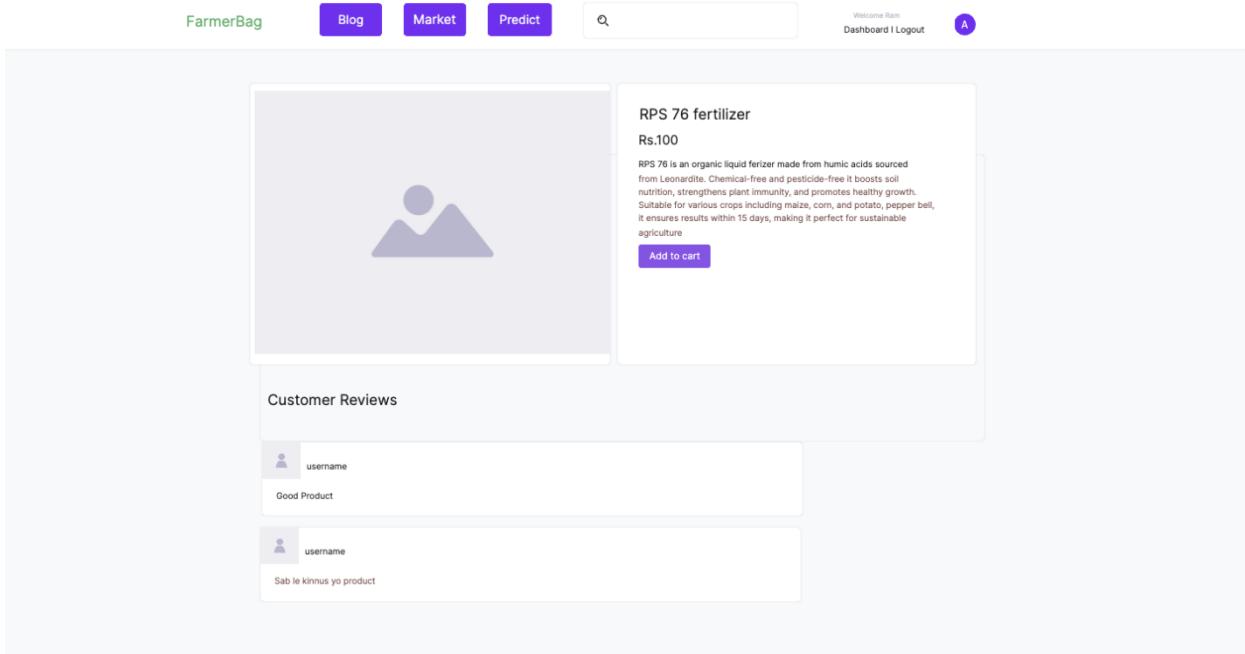


Figure 38: User Interface: Product Description

3.5.3.9 USER INTERFACE: ADMIN DAHBOARD

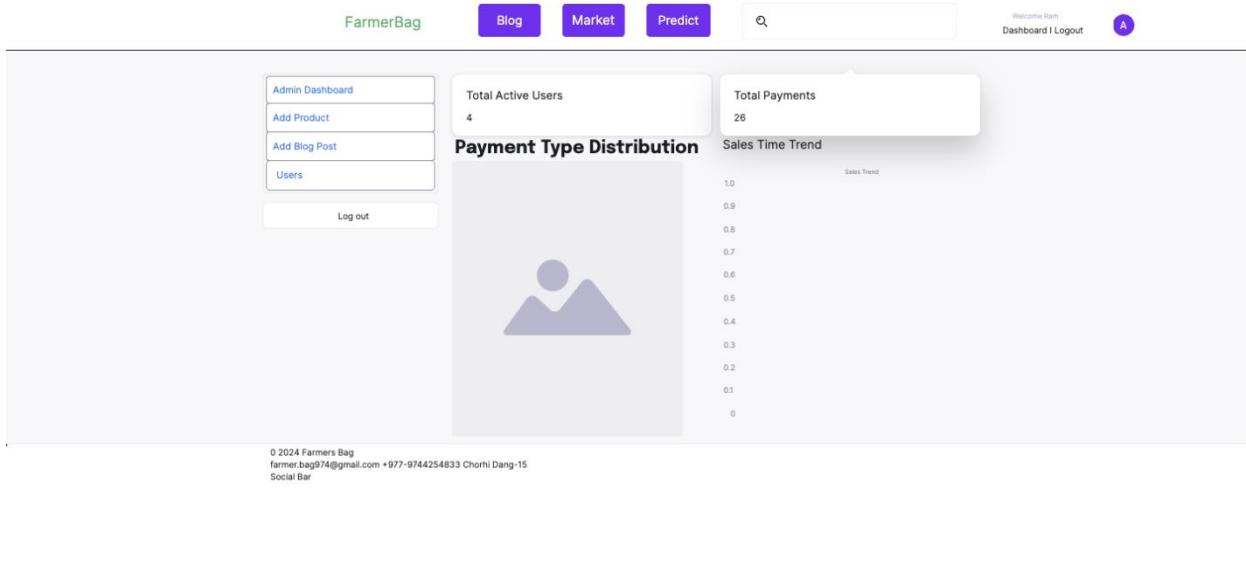


Figure 39: User Interface: Admin Dashboard

3.5.3.10. USER INTERFACE: CART

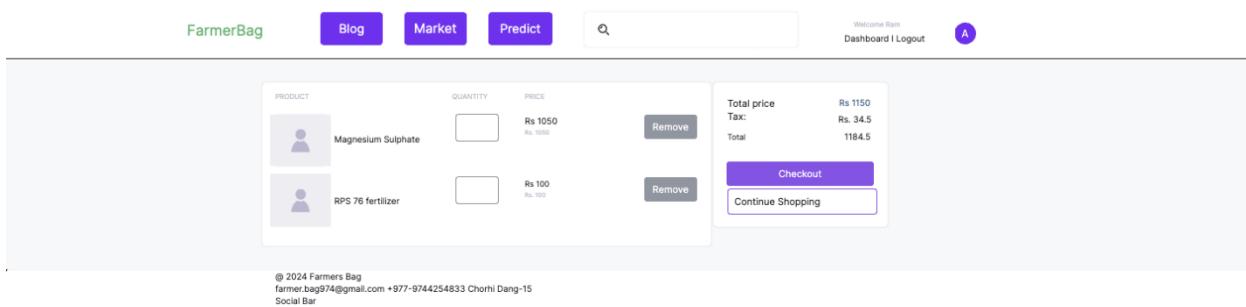


Figure 40: User Interface: Add to Cart

3.5.3.11. USER INTERFACE: ADD PRODUCT

The screenshot shows the 'Create Product' form. On the left, there is a sidebar with the following menu items:

- Admin Dashboard
- Add Product** (highlighted in blue)
- Add Blog Post
- Users

Below the sidebar is a 'Log out' button. The main form has the following fields:

- Product name:
- Slug:
- Description:
- Price:
- Images:
- Stock:
- Is available:
- Category:
- Submit:

At the bottom left, it says "2024 Farmers Bag".

Figure 41: User Interface: Admin Add Product

3.5.3.12 USER INTERFACE: BLOG POST

The screenshot shows the 'Create Blog Post' form. On the left, there is a sidebar with the following menu items:

- Admin Dashboard
- Add Product
- Add Blog Post** (highlighted in blue)
- Users

Below the sidebar is a 'Log out' button. The main form has the following fields:

- Title:
- Slug:
- Description:
- Image:
- Submit:

At the bottom left, it says "2024 Farmers Bag" and "farmer.bag974@gmail.com +977-9744254833 Chorhi Dang-15".

Figure 42: User Interface: Admin Create Blog Post

3.5.3.13 USER INTERFACE: MANAGE ACCOUNT

The screenshot shows a web-based application interface for managing user accounts. At the top, there is a navigation bar with links for 'FarmerBag', 'Blog', 'Market', 'Predict', a search bar, and a user profile section with 'Welcome Ram' and 'Logout'. Below the navigation bar is a sidebar on the left containing buttons for 'Admin Dashboard', 'Add Product', 'Add Blog Post', and 'Logout'. The main content area is titled 'User List' and displays a table with columns: Username, Email, First Name, Last Name, and Action. The table contains four rows of data. At the bottom of the page, there is a footer with copyright information: '2024 Farmers Bag', 'farmer.bag974@gmail.com +877-9744254833 Chorhi Dang-15', and 'Social Bar'.

Username	Email	First Name	Last Name	Action
farmer	farmer.bag974@gmail.com	farmer	farmer	<button>Delete</button>
kcsuman1203	kcsuman1203@gmail.com	Rajesh	tori	<button>Delete</button>
1203lozer	1203lozer@gmail.com	Alin	K.C	<button>Delete</button>
kcsumankc1203	kcsumankc1203@gmail.com	SUMAN	K.C	<button>Delete</button>

Figure 43: User Interface: Manage User Account

3.5.4. UML DESIGN

3.5.4.1. USE CASE DIAGRAM

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality incorporating the use case, actors, and their relationships. It models the tasks, services, and functionality of a system and also tells how the user handles a system. (javatpoint, 2023)

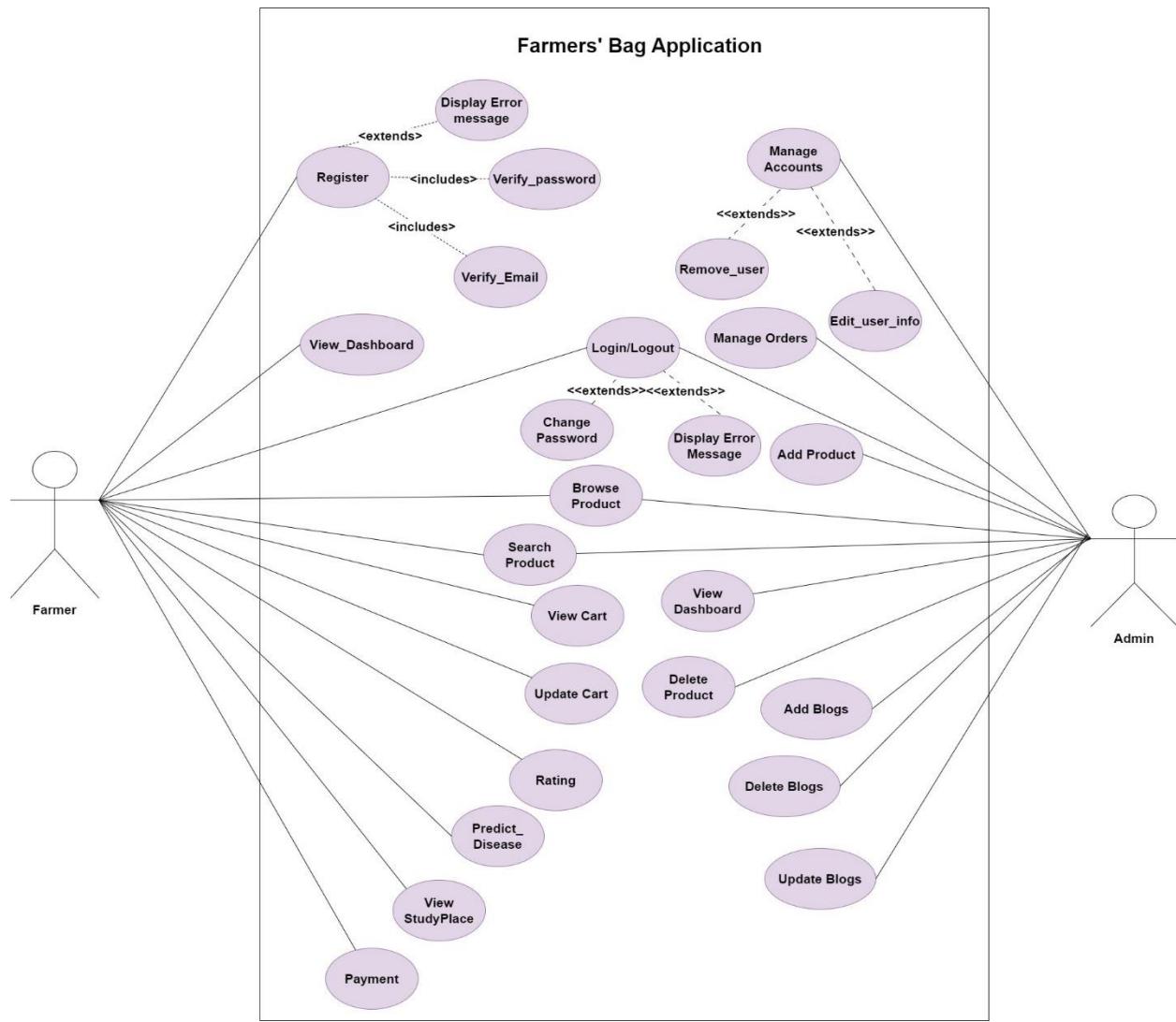


Figure 44: Design: Use Case Diagram

Appendix: [\[7.5.1.1. DESCRIPTION OF HIGH-LEVEL USE CASE DIAGRAM\]](#)

Appendix: [\[7.5.1.2. DESCRIPTION OF EXPANDED-LEVEL USE CASE DIAGRAM\]](#)

3.5.4.2. DATA FLOW DIAGRAM

A context diagram is a visual illustration of the connection between data and business operations. It provides the events and factors one should think about when creating a system. The three essential parts of this diagram are the external entities, system operations and data flow. It will enable the individual to determine the scope, boundaries, and system requirements. It does not, however, include the sequence of events, timing, or any technical information. As a result, it is a great tool for helping business analysts, stakeholders, and developers understand the system. (Pedriquez, 2022)

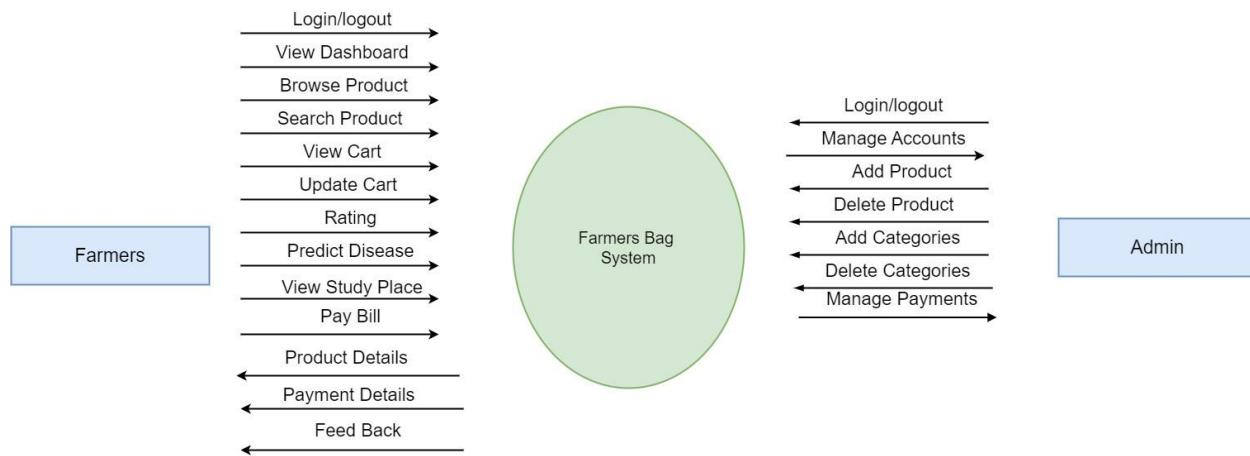


Figure 45: Design: Data Flow Diagram

3.5.4.3. SEQUENCE DIAGRAM

A sequence diagram, a powerful visual tool in software engineering and system design, illustrates the chronological interactions between different components or objects within a system over time. It offers a graphical representation of the system's behaviour, depicting the sequential flow of messages exchanged among objects through lifelines representing each entity and arrows denoting the message exchanges. The diagram starts at the top and flows downward, clearly portraying the order of events and message occurrences, enabling developers and stakeholders to easily visualize the step-by-step processes and comprehend complex interactions. As part of the Unified Modelling Language (UML), sequence diagrams adhere to standardized notations, seamlessly integrating into various development stages from requirements gathering to implementation and testing. By mapping out the Collaboration patterns between objects, these diagrams aid in understanding the intricate dynamics, identifying potential issues or optimizations, and grasping how the system operates in different scenarios or use cases, each represented by distinct sequence diagrams. Consequently, sequence diagrams emerge as invaluable assets, providing a clear and concise representation of object interactions, facilitating a profound comprehension of the system's behaviour, and playing a crucial role in software development by enabling effective Collaboration, documentation, and analysis. (Wisbey, 2022)

3.5.4.3.1. SEQUENCE DIAGRAM: NON-REGISTERED USERS

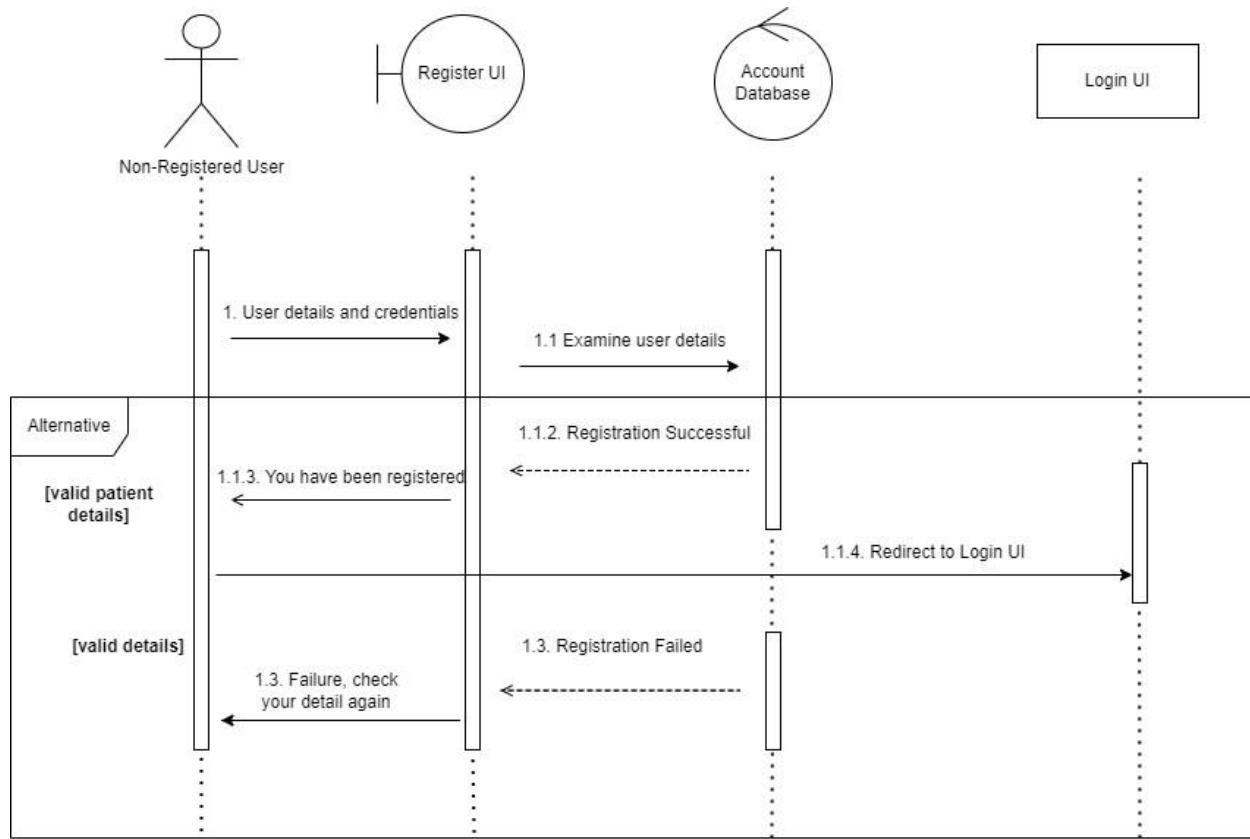


Figure 46: Sequence Diagram: NON-Registered Users

Other Sequence Diagram are included in Appendix Section.

Appendix: [\[7.5.2. SEQUENCE DIAGRAMS\]](#)

3.5.2.4. ACTIVITY DIAGRAM

An Activity Diagram is an important behavioural diagram in UML that depicts the dynamic aspects of a system. It serves as an advanced version of a flow chart, illustrating the flow from one activity to another. It describes how activities are coordinated to provide a service at various levels of abstraction. Typically, an event is achieved through operations, especially when these operations intend to accomplish multiple tasks simultaneously. (visual-paradigm, 2023)

3.5.2.4.1. ACTIVITY DIAGRAM: UN-REGISTERED USERS

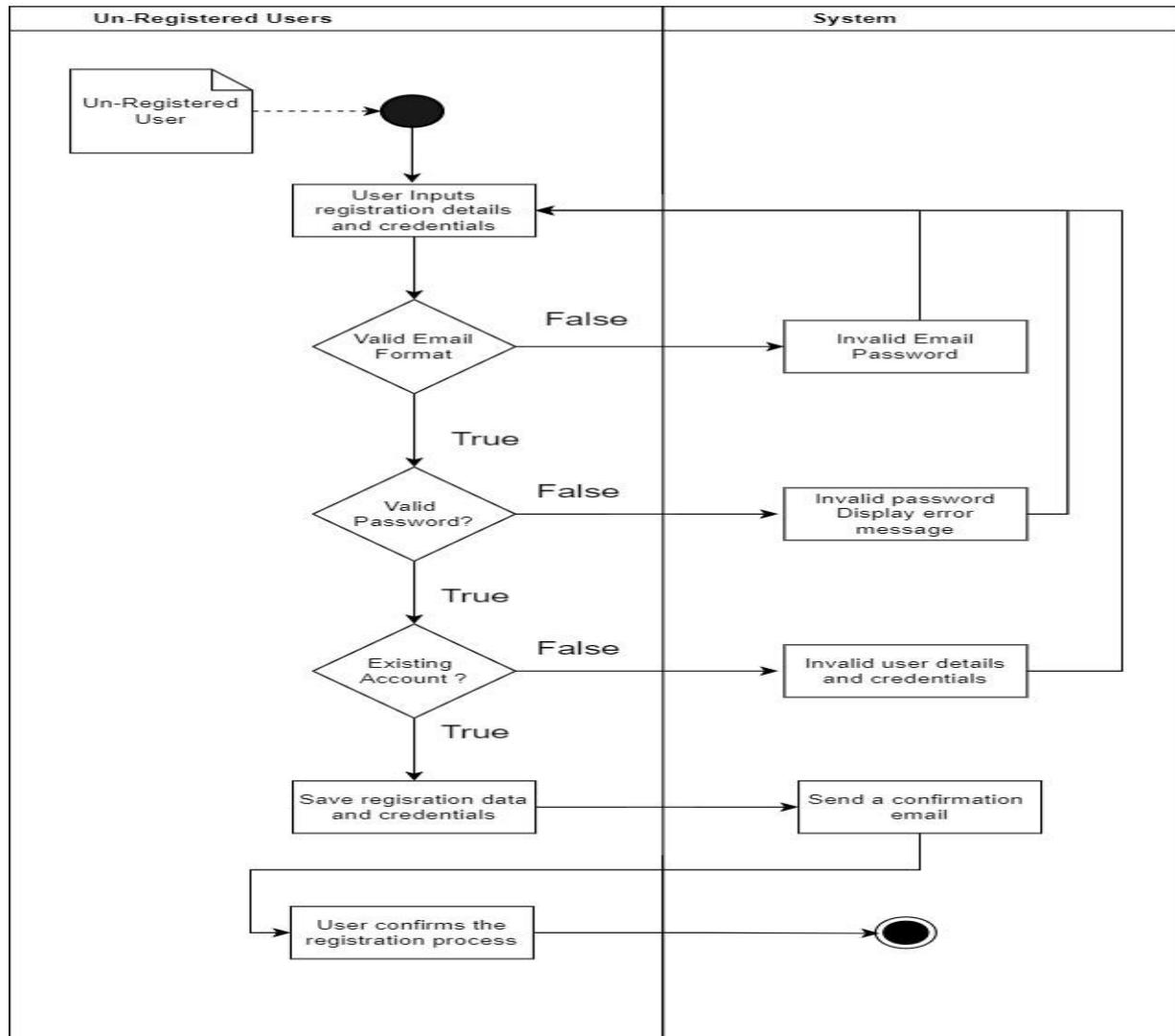


Figure 47: Activity Diagram: Un-Registered Users

Appendix: [\[7.5.3. ACTIVITY DIAGRAMS\]](#)

3.5.4.5. COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML). Developers can use these diagrams to depict the dynamic behaviour of a particular use case and define the role of each object. To create a collaboration diagram, first identify the structural elements required to carry out the functionality of an interaction. Then construct a model using the relationships between those elements. Several vendors offer software tools for creating and editing collaboration diagrams. (Lewis, 2023)

3.5.4.5.1. COLLABORATION DIAGRAM: REGISTER ACCOUNT

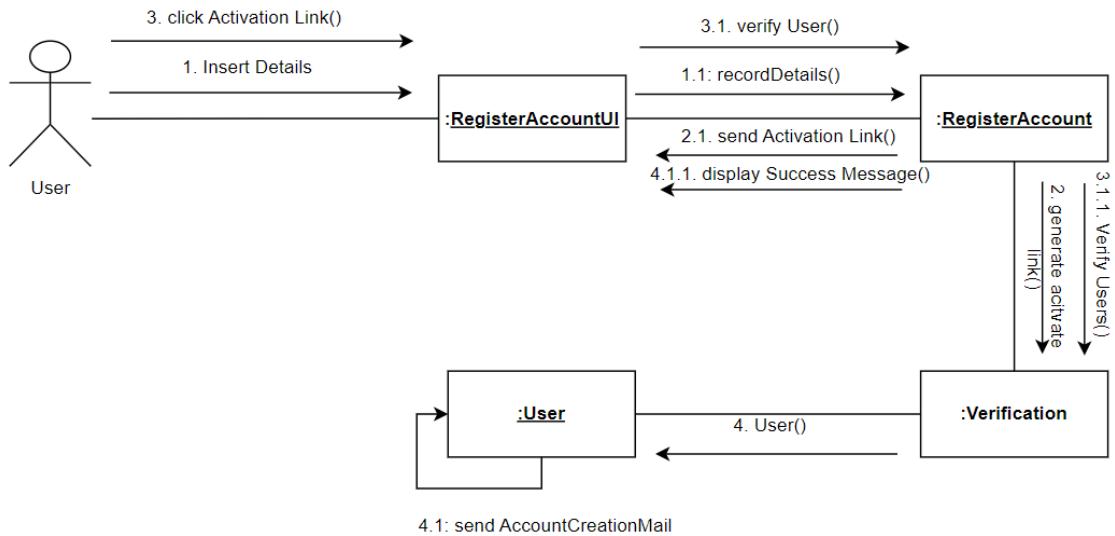


Figure 48: Collaboration Diagram: Register Accounts

Other Collaboration Diagram are included in appendix section.

Appendix: [\[7.5.4. COLLABORATION DIAGRAM\]](#)

3.5.5. ENTITY RELATIONSHIP DIAGRAM

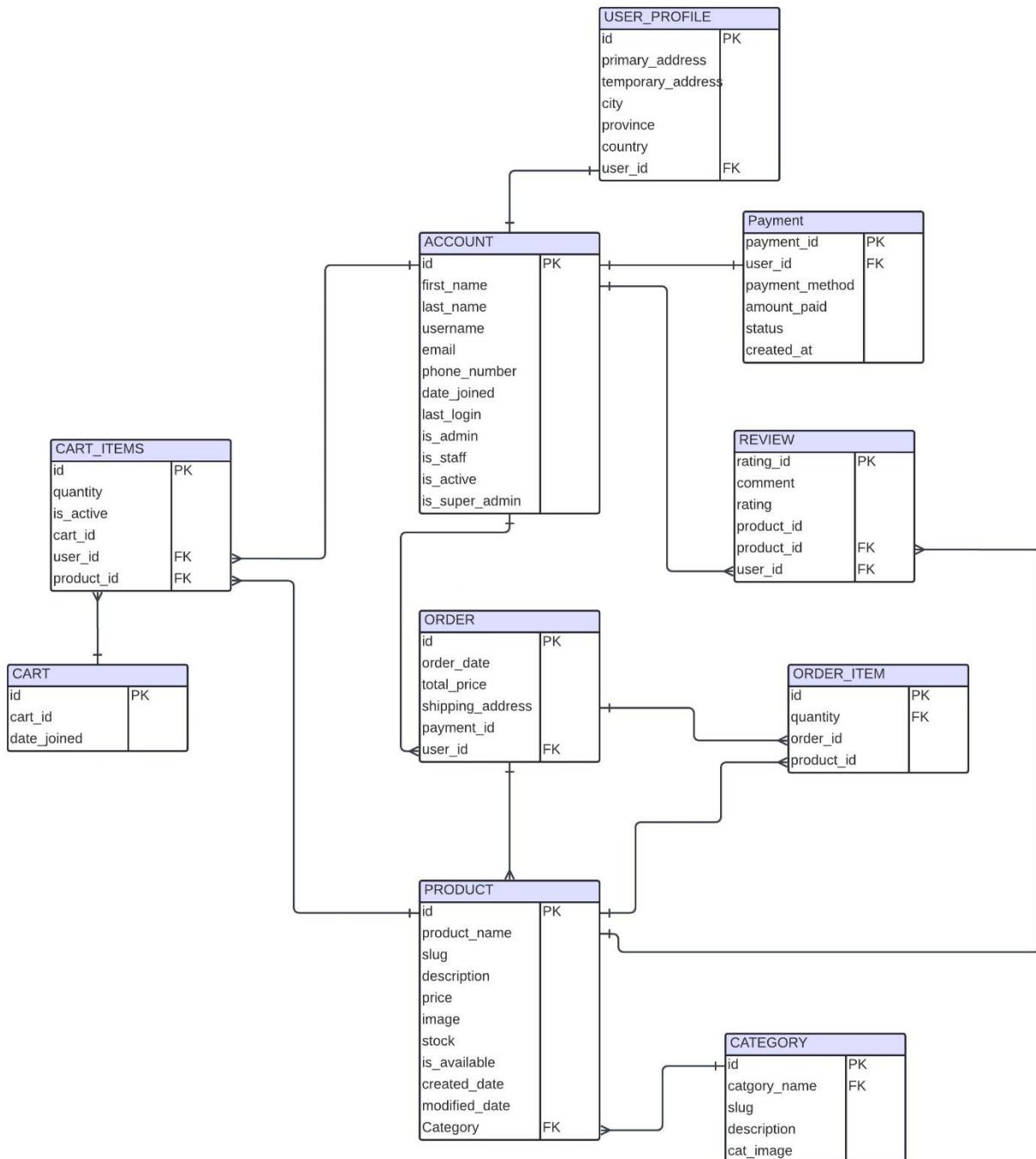


Figure 49: Entity Relationship Diagram

An Entity Relationship Diagram (ERD), also known as entity relationship model, visually represents relationships among entities in an information technology system, encompassing entities, attributes, and relationships. Crucial in foundational database design, ERDs serve as a communication tool, fostering shared understanding among stakeholders. They act as a common language, aiding in requirement analysis and ensuring a comprehensive grasp of the system's dynamics. Beyond the initial phase, ERDs remain valuable reference points, contributing to database maintenance, debugging, and process efficiency improvements. While excelling in structuring data, ERDs may have limitations with semi-structured or unstructured data. Nevertheless, their importance in providing a visual starting point for database design and enhancing communication with organizations underscores their significance in information technology. (Biscobing, 2023)

3.6. IMPLEMENTATION

3.6.1. DATABASE CREATION

The Entity-Relationship Diagram (ERD) served as a blueprint for meticulously designing the database architecture, ensuring optimal structure and efficiency. Leveraging Django's powerful models facilitated the creation process, establishing a robust foundation for future enhancements and scalability. SQLite was employed for database creation, providing a reliable platform for data storage and management.

3.6.1.1. DATABASE CREATION: ACCOUNTS

This code creates a custom user model and a manager for handling user accounts in a Django website. The `MyAccountManager` class manages user creation, including a method for creating super users (admins). The `Account` model defines fields for user details like first name, last name, email, and username. It also has fields for managing user permissions and status (admin, staff, active, super admin). The `UserProfile` model is connected to the `Account` model and stores extra user information like addresses, profile picture, and location details.

```
File Edit Selection View Go Run Terminal Help < > FarmerBag

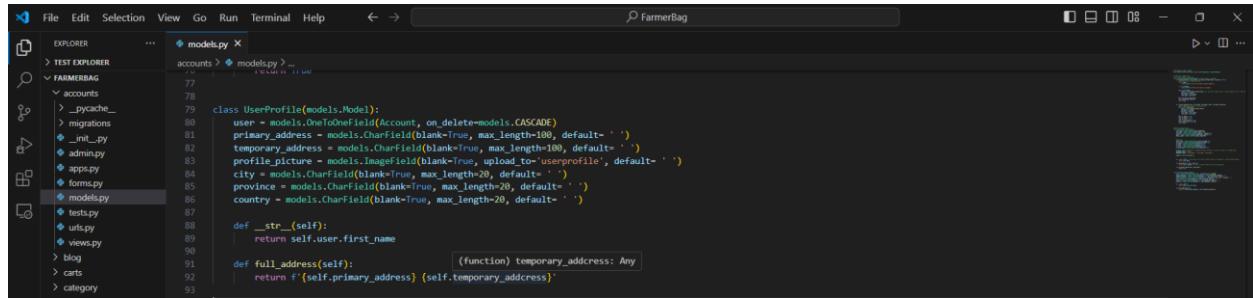
EXPLORER
> TEST EXPLORER
> FARMERBAG
accounts > models.py > ...
1  from django.db import models
2  from django.contrib.auth.models import AbstractBaseUser, BaseUserManager
3
4
5  # Create your models here.
6  # this model handles custom user
7  class MyAccountManager(BaseUserManager):
8      def create_user(self, first_name, last_name,username,email, password = None):
9          if not email:
10              raise ValueError('user must have an email address')
11
12          if not username:
13              raise ValueError('User must have an username')
14
15          user = self.model(
16              email = self.normalize_email(email), #if you enter capital letter in email address then it converts to lower letter
17              username = username,
18              first_name = first_name,
19              last_name = last_name,
20          )
21          user.set_password(password)
22          user.save(using= self._db)
23
24          return user
25
26
27  def create_superuser(self, first_name, last_name, email, username, password):
28      #using creatuser param in superuser param
29      user = self.create_user(
30          email = self.normalize_email(email),
31          username = username,
32          password = password,
33          first_name = first_name,
34          last_name = last_name,
35      )
36      user.is_admin = True
37      user.is_active = True
38      user.is_staff = True
39      user.is_super_admin = True
40      user.save(using= self._db)
41
42      return user
```

Figure 50: Database Creation: ACCOUNTS

Figure 51: Database Creation: ACCOUNTS (2)

		Table name: accounts_account							<input type="checkbox"/> WITHOUT ROWID	<input type="checkbox"/> STRICT	
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value	
1	id	integer	🔑				⚠️			NULL	
2	password	varchar (128)					⚠️			NULL	
3	first_name	varchar (50)					⚠️			NULL	
4	last_name	varchar (50)					⚠️			NULL	
5	username	varchar (50)			📦		⚠️			NULL	
6	email	varchar (100)			📦		⚠️			NULL	
7	phone_number	varchar (50)					⚠️			NULL	
8	date_joined	datetime					⚠️			NULL	
9	last_login	datetime					⚠️			NULL	
10	is_admin	bool					⚠️			NULL	
11	is_staff	bool					⚠️			NULL	
12	is_active	bool					⚠️			NULL	
13	is_superuser	bool					⚠️			NULL	

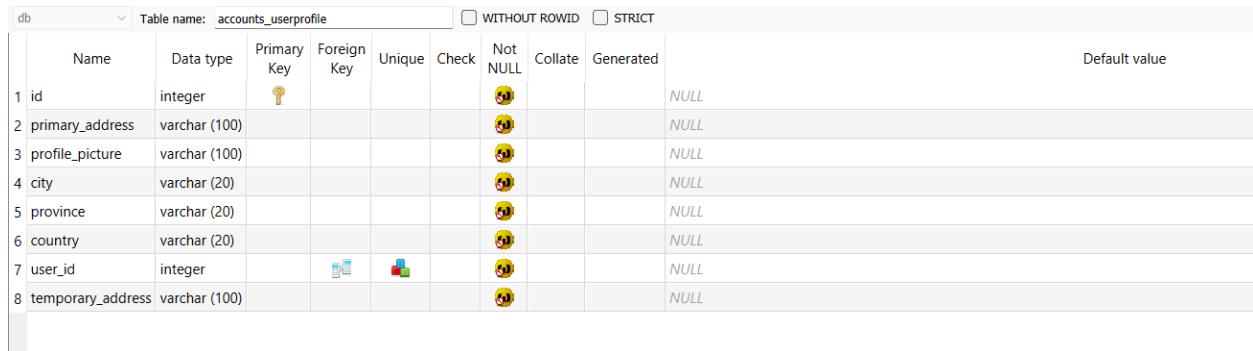
Figure 52: Database Creation: SQLLITE (ACCOUNT)



```

accounts > models.py > ...
77
78
79 class UserProfile(models.Model):
80     user = models.OneToOneField(Account, on_delete=models.CASCADE)
81     primary_address = models.CharField(blank=True, max_length=100, default=' ')
82     temporary_address = models.CharField(blank=True, max_length=100, default=' ')
83     profile_picture = models.ImageField(blank=True, upload_to='userprofile', default=' ')
84     city = models.CharField(blank=True, max_length=20, default=' ')
85     province = models.CharField(blank=True, max_length=20, default=' ')
86     country = models.CharField(blank=True, max_length=20, default=' ')
87
88     def __str__(self):
89         return self.user.first_name
90
91     def full_address(self):
92         (function) temporary_address: Any
93         return f'{self.primary_address} {self.temporary_address}'

```

Figure 53: Database Creation: ACCOUNTS (3)


	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer	key							NULL
2	primary_address	varchar (100)								NULL
3	profile_picture	varchar (100)								NULL
4	city	varchar (20)								NULL
5	province	varchar (20)								NULL
6	country	varchar (20)								NULL
7	user_id	integer		table	cube					NULL
8	temporary_address	varchar (100)								NULL

Figure 54: Database Creation: SQLLITE (USERPROFILE)

3.6.1.2. DATABASE CREATION: BLOG

This code creates a Django model called `BlogPost` with fields for title, slug, description, image, `created_date`, and `modified_date`. It represents individual blog posts within a web application, allowing users to create and manage blog content.

```
File Edit Selection View Go Run Terminal Help < > FarmerBag

EXPLORER ... models.py accounts models.py blog x
> TEST EXPLORER
> FARMERBAG
> accounts
> blog
> _pycache_
> migrations
> __init__.py
> admin.py
> apps.py
> models.py
> test.py
> urls.py
> views.py
> carts
> category
> env
> farmersbag
> media
> models
> order
> _pycache_
> migrations
> __init__.py
> admin.py
> apps.py
> models.py

blog > models.py ...
7     class BlogPost(models.Model):
8         title = models.CharField(max_length=200)
9         slug = models.SlugField(max_length=200, unique=True)
10        description = models.TextField(max_length=500)
11        image = models.ImageField(upload_to='photos/blog')
12        created_date = models.DateTimeField(auto_now_add=True)
13        modified_date = models.DateTimeField(auto_now=True)
14
15
16    def __str__(self):
17        return self.title
18
19
20
21    class BlogPostForm(forms.ModelForm):
22        class Meta:
23            model = BlogPost
24            fields = ['title', 'slug', 'description', 'image']
25
26        widgets = {
27            'title': forms.TextInput(attrs={'class': 'form-control'}),
28            'slug': forms.TextInput(attrs={'class': 'form-control'}),
29            'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
30            'image': forms.ClearableFileInput(attrs={'class': 'form-control-file'}),
31        }
32
33    }


```

Figure 55: Database Creation: BLOG

db	Table name:	blog_blogpost							<input type="checkbox"/> WITHOUT ROWID	<input type="checkbox"/> STRICT
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer	🔑				🟡			NULL
2	title	varchar (200)					🟡			NULL
3	slug	varchar (200)			📦		🟡			NULL
4	description	text					🟡			NULL
5	image	varchar (100)					🟡			NULL
6	created_date	datetime					🟡			NULL
7	modified_date	datetime					🟡			NULL

Figure 56: Database Creation: SQLLITE (BLOGPOST)

3.6.1.3. DATABASE CREATION: CART & CART ITEM

This code creates Django models for managing a shopping cart system. The Cart model tracks cart details like its identifier and creation date. The CartItem model associates items with users, products, and carts, while also recording quantities and activity status. It includes a method to calculate subtotal costs.

The screenshot shows a Python IDE interface with the following details:

- File Menu:** File, Edi, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Back, Forward, Home, Search, Refresh, Stop, Minimize, Maximize, Close.
- Explorer:** Shows the project structure under "FARMERBAG".
 - models.py:** Contains code for the Cart model, including imports for django.db, store.models, accounts.models, and a class Cart with fields cart_id, date_added, and a __str__ method returning self.cart_id.
 - models.py (cont'd):** Contains code for the CartItem model, including imports for models, and a class CartItem with fields user, product, cart, quantity, and is_active.
- Code Editor:** Displays the same content as the Explorer pane, with syntax highlighting for Python code.

Figure 57: Database Creation: CART

db		Table name: carts_cart				<input type="checkbox"/> WITHOUT ROWID	<input type="checkbox"/> STRICT			
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								
2	cart_id	varchar (250)								
3	date_added	date								

Figure 58: Database Creation: SOLLITE (CART)

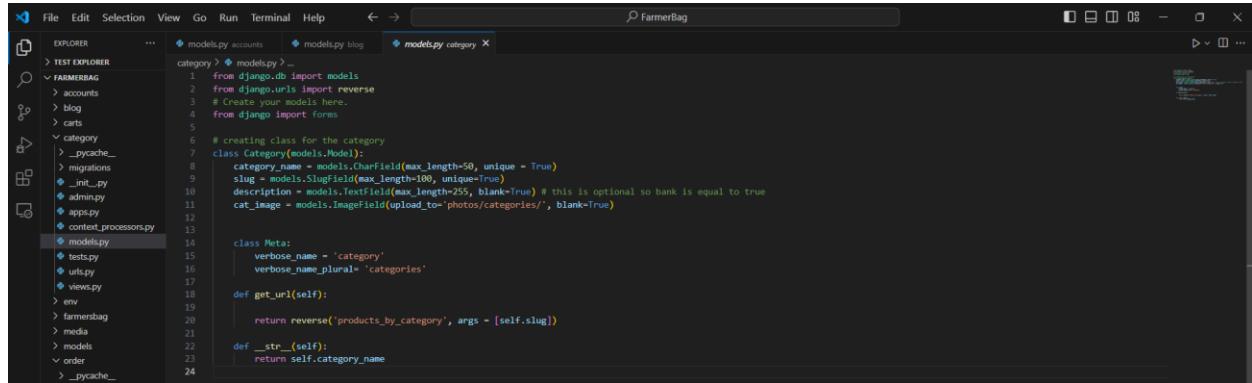
The screenshot shows a database creation interface with the following details:

- Table name: carts_cartitem
- Checkboxes: WITHOUT ROWID, STRICT
- Columns (6 rows):
 - 1 id: integer, Primary Key, Not NULL, Default value: NULL
 - 2 quantity: integer, Unique, Not NULL, Default value: NULL
 - 3 is_active: bool, Unique, Not NULL, Default value: NULL
 - 4 cart_id: integer, Foreign Key, Default value: NULL
 - 5 product_id: integer, Foreign Key, Default value: NULL
 - 6 user_id: integer, Foreign Key, Default value: NULL

Figure 59: Database Creation: SOLLITE (CART ITEM)

3.6.1.4. DATABASE CREATION: CATEGORY

This code creates a Django model for managing product categories within a web application. It defines fields for category name, slug, description, and category image. Additionally, it specifies metadata for the model's human-readable names and includes a method to generate category URLs. The model's string representation is based on the category name.

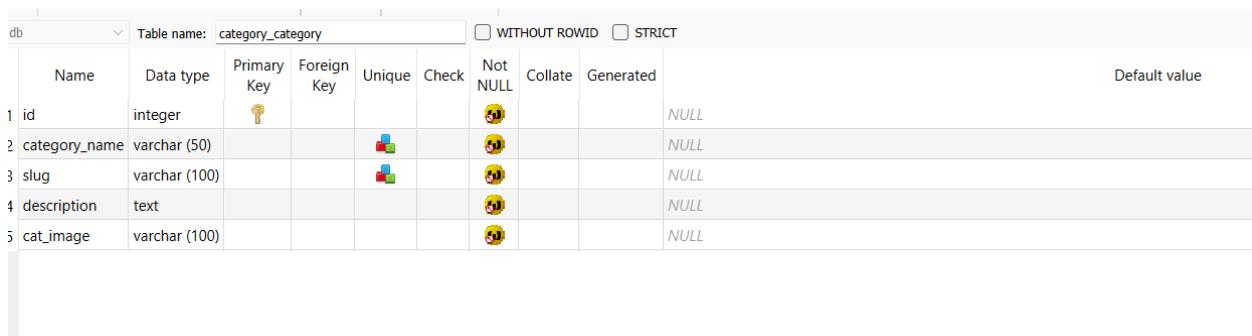


```

File Edit Selection View Go Run Terminal Help ← → FarmerBag
EXPLORER models.py accounts models.py blog models.py category
> TEST EXPLORER
> FARMERBAG
> accounts
> blog
> carts
> category
> __pycache__
> migrations
> __init__.py
> admin.py
> apps.py
> context_processors.py
> models.py
> tests.py
> urls.py
> views.py
> env
> farmerbag
> media
> models
> order
> __pycache__ 1 from django.db import models
2 from django.urls import reverse
3 # Create your models here.
4 from django import forms
5
6 # creating class for the category
7 class Category(models.Model):
8     category_name = models.CharField(max_length=50, unique = True)
9     slug = models.SlugField(max_length=100, unique=True)
10    description = models.TextField(max_length=255, blank=True) # this is optional so bank is equal to true
11    cat_image = models.ImageField(upload_to="photos/categories/", blank=True)
12
13    class Meta:
14        verbose_name = 'category'
15        verbose_name_plural= 'categories'
16
17    def get_url(self):
18        return reverse('products_by_category', args = [self.slug])
19
20    def __str__(self):
21        return self.category_name
22
23
24

```

Figure 60: Database Creation: Category



The screenshot shows the SQLite Manager interface with the following table definition:

		Table name: category						<input type="checkbox"/> WITHOUT ROWID		<input type="checkbox"/> STRICT	
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value	
1	id	integer								NULL	
2	category_name	varchar (50)								NULL	
3	slug	varchar (100)								NULL	
4	description	text								NULL	
5	cat_image	varchar (100)								NULL	

Figure 61: Database Creation: SOLLITE (CATEGORY)

3.6.1.5. DATABASE CREATION: ORDER

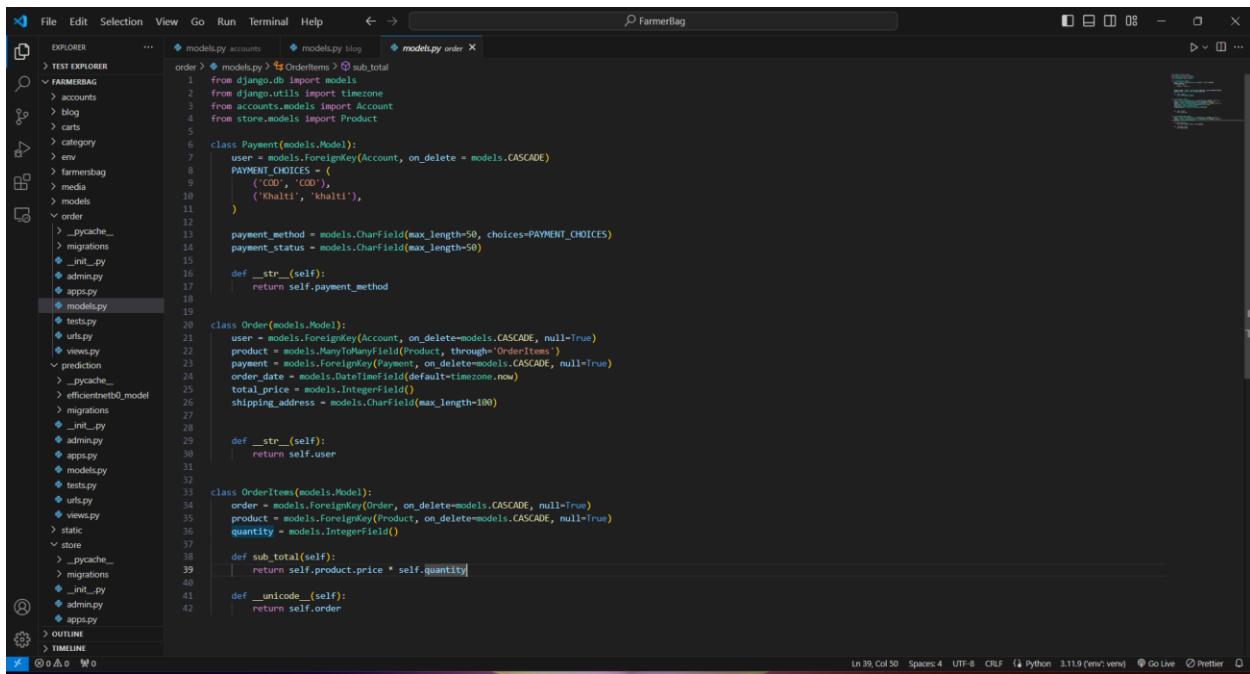
This code snippet defines Django models to manage orders and payments within a system.

The Payment model stores information about payment transactions, including the associated user, payment method, and payment status. The payment_method field offers choices between Cash on Delivery ('COD') and an online payment method ('Khalti'). The payment_status field indicates the status of the payment transaction.

The Order model represents individual orders placed by users. It includes fields for the user who placed the order, the products being ordered, the associated payment transaction, the order date, total price, and shipping address.

The OrderItems model serves as an intermediary between orders and products, allowing for the association of multiple products with each order and specifying the quantity of each product.

It also calculates the subtotal for each order item based on the product price and quantity. Both the Order and Payment models have `__str__` methods defined to return human-readable representations, aiding in their identification and administration within the system.



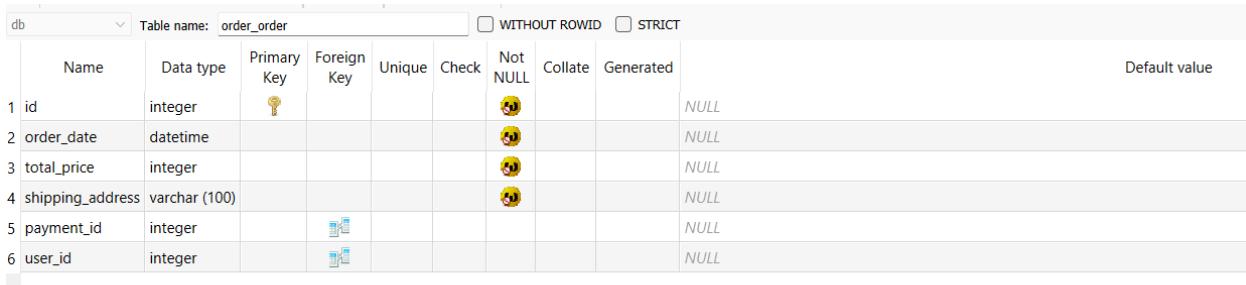
The screenshot shows a code editor with the file `models.py` open. The code defines three Django models: Payment, Order, and OrderItems. The Payment model has a foreign key to Account and a choice field for payment method ('COD' or 'Khalti'). The Order model has a foreign key to Account, a ManyToManyField to Product via OrderItems, and fields for payment transaction, order date, total price, and shipping address. The OrderItems model has foreign keys to Order and Product, and a quantity field. The `__str__` methods for all three models return relevant information.

```

File Edit Selection View Go Run Terminal Help ← → ⌘ FarmerBag
EXPLORER ... models.py accounts models.py blog models.py order
> TEST EXPLORER
> FARMERBAG
> accounts
> blog
> carts
> category
> env
> farmersbag
> media
> models
> order
> __pycache__
> migrations
> __init__.py
> admin.py
> apps.py
> models.py
> tests.py
> urls.py
> views.py
> prediction
> __pycache__
> efficientneibb_model
> migrations
> __init__.py
> admin.py
> apps.py
> models.py
> tests.py
> urls.py
> views.py
> static
> store
> __pycache__
> migrations
> __init__.py
> admin.py
> apps.py
> OUTLINE
> TIMELINE
models.py
File Edit Selection View Go Run Terminal Help ← → ⌘ FarmerBag
models.py accounts models.py blog models.py order
> FARMERBAG
> accounts
> blog
> carts
> category
> env
> farmersbag
> media
> models
> order
> __pycache__
> migrations
> __init__.py
> admin.py
> apps.py
> models.py
> tests.py
> urls.py
> views.py
> prediction
> __pycache__
> efficientneibb_model
> migrations
> __init__.py
> admin.py
> apps.py
> models.py
> tests.py
> urls.py
> views.py
> static
> store
> __pycache__
> migrations
> __init__.py
> admin.py
> apps.py
> OUTLINE
> TIMELINE
models.py
1 from django.db import models
2 from django.utils import timezone
3 from accounts.models import Account
4 from store.models import Product
5
6 class Payment(models.Model):
7     user = models.ForeignKey(Account, on_delete=models.CASCADE)
8     PAYMENT_CHOICES = (
9         ('COD', 'COD'),
10        ('Khalti', 'Khalti'),
11    )
12
13     payment_method = models.CharField(max_length=50, choices=PAYMENT_CHOICES)
14     payment_status = models.CharField(max_length=50)
15
16     def __str__(self):
17         return self.payment_method
18
19
20 class Order(models.Model):
21     user = models.ForeignKey(Account, on_delete=models.CASCADE, null=True)
22     product = models.ManyToManyField(Product, through='OrderItems')
23     payment = models.ForeignKey(Payment, on_delete=models.CASCADE, null=True)
24     order_date = models.DateTimeField(default=timezone.now)
25     total_price = models.IntegerField()
26     shipping_address = models.CharField(max_length=100)
27
28     def __str__(self):
29         return self.user
30
31
32
33 class OrderItems(models.Model):
34     order = models.ForeignKey(Order, on_delete=models.CASCADE, null=True)
35     product = models.ForeignKey(Product, on_delete=models.CASCADE, null=True)
36     quantity = models.IntegerField()
37
38     def sub_total(self):
39         return self.product.price * self.quantity
40
41     def __unicode__(self):
42         return self.order

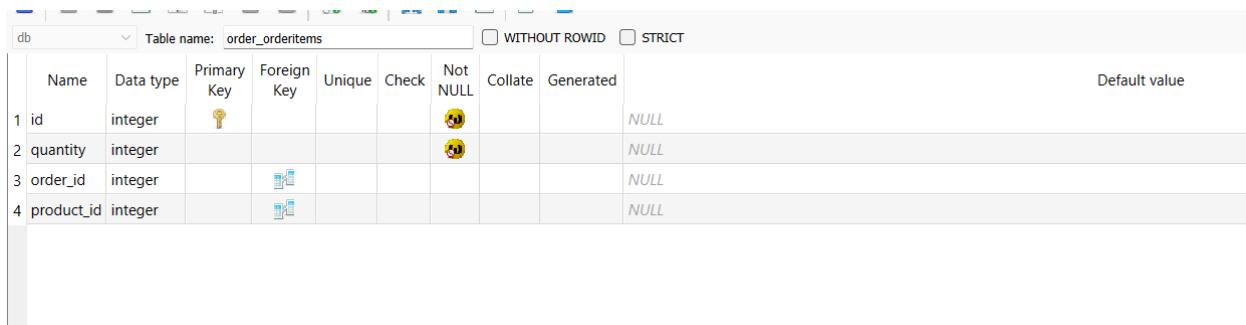
```

Figure 62: Database Creation: ORDER



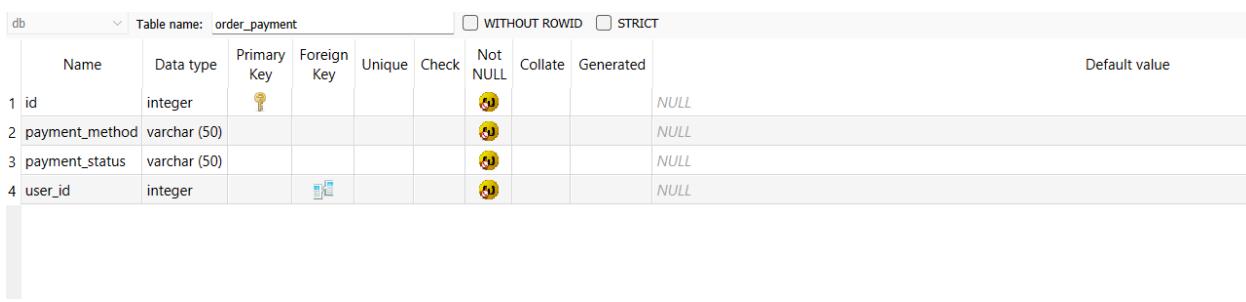
The screenshot shows the SOLLITE database creation interface with the table name set to "order_order". The table structure is defined as follows:

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	order_date	datetime								NULL
3	total_price	integer								NULL
4	shipping_address	varchar (100)								NULL
5	payment_id	integer								NULL
6	user_id	integer								NULL

Figure 63: Database Creation: SOLLITE (ORDER)


The screenshot shows the SOLLITE database creation interface with the table name set to "order_orderitems". The table structure is defined as follows:

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	quantity	integer								NULL
3	order_id	integer								NULL
4	product_id	integer								NULL

Figure 64: Database Creation: SOLLITE (ORDERITEM)


The screenshot shows the SOLLITE database creation interface with the table name set to "order_payment". The table structure is defined as follows:

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	payment_method	varchar (50)								NULL
3	payment_status	varchar (50)								NULL
4	user_id	integer								NULL

Figure 65: Database Creation: SOLLITE (PAYMENT)

3.6.1.6. DATABASE CREATION: PRODUCT

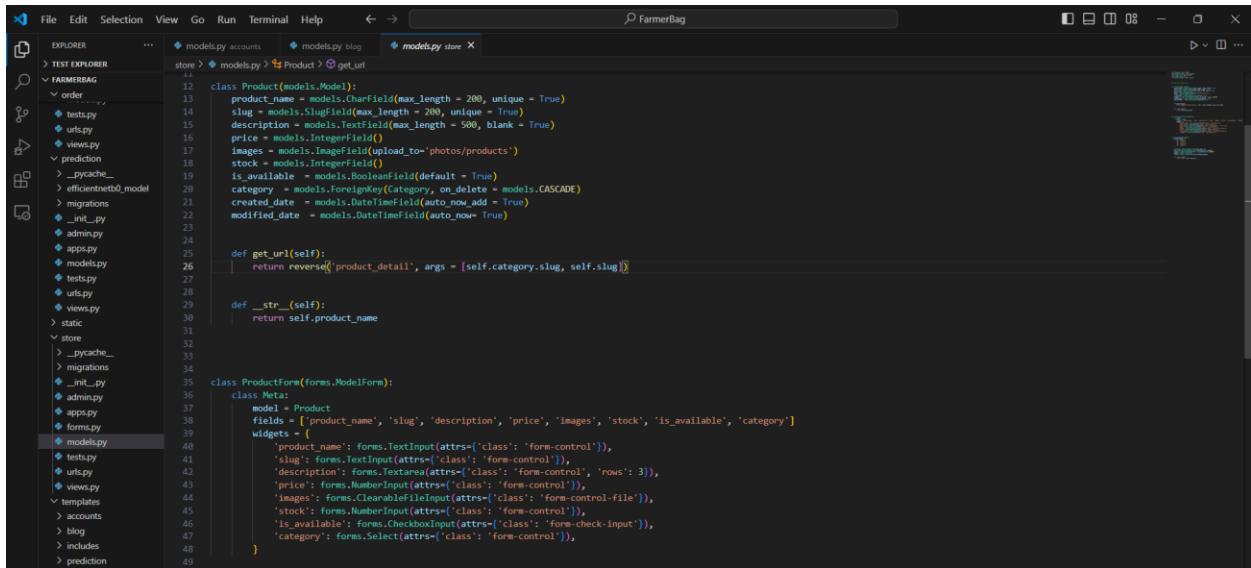
This code snippet defines Django models and forms to manage products and reviews within an e-commerce system.

The Product model represents individual products available for purchase. It includes fields for the product's name, a URL-friendly slug, description, price, images, stock availability, whether the product is currently available for purchase, the category to which it belongs, and timestamps for creation and modification.

Additionally, a method `get_url()` is defined to generate and return the URL for accessing the product's detail page. The `__str__` method is implemented to return the product's name for easy identification.

The Review model stores user reviews for products. It includes fields for the user who submitted the review, the product being reviewed, the comment or review text, and the rating given by the user, with choices ranging from 1 to 5 stars.

The `__str__` method is implemented to return a string representation of the review, including its unique identifier (rating_id).



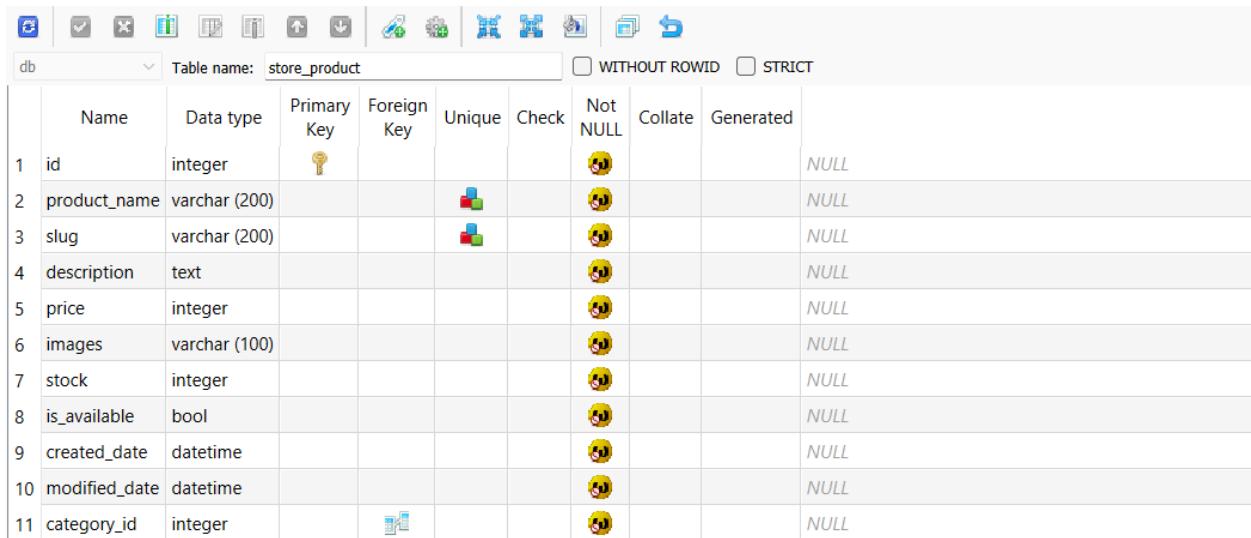
The screenshot shows a code editor with the file `models.py` open. The code defines a `Product` model with various fields and methods. The `__str__` method is overridden to return the product name. The `get_url` method is defined to return the URL for the product detail page. The `ProductForm` form is also defined, mapping fields to their respective input types and attributes.

```

File Edit Selection View Go Run Terminal Help ← → 🔍 FarmerBag
EXPLORER ... models.py accounts models.py blog models.py store ✘ TEST EXPLORER
> FARMERBAG
  > order
    tests.py
    urls.py
    views.py
  > prediction
    > __pycache__
    efficientnetb0.model
    migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      urls.py
      views.py
    static
  > store
    > __pycache__
    migrations
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      urls.py
      views.py
    templates
      > accounts
      > blog
      > includes
      > prediction
store > models.py 7 Product 7 get_url
12     class Product(models.Model):
13         product_name = models.CharField(max_length = 200, unique = True)
14         slug = models.SlugField(max_length = 200, unique = True)
15         description = models.TextField(max_length = 500, blank = True)
16         price = models.IntegerField()
17         images = models.ImageField(upload_to='photos/products')
18         stock = models.IntegerField()
19         is_available = models.BooleanField(default = True)
20         category = models.ForeignKey(Category, on_delete = models.CASCADE)
21         created_date = models.DateTimeField(auto_now_add = True)
22         modified_date = models.DateTimeField(auto_now=True)
23
24     def get_url(self):
25         return reverse('product_detail', args = [self.category.slug, self.slug])
26
27     def __str__(self):
28         return self.product_name
29
30
31
32
33
34
35     class ProductForm(forms.ModelForm):
36         class Meta:
37             model = Product
38             fields = ['product_name', 'slug', 'description', 'price', 'images', 'stock', 'is_available', 'category']
39             widgets = {
40                 'product_name': forms.TextInput(attrs={'class': 'form-control'}),
41                 'slug': forms.TextInput(attrs={'class': 'form-control'}),
42                 'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
43                 'price': forms.NumberInput(attrs={'class': 'form-control'}),
44                 'images': forms.ClearableFileInput(attrs={'class': 'form-control-file'}),
45                 'stock': forms.NumberInput(attrs={'class': 'form-control'}),
46                 'is_available': forms.CheckboxInput(attrs={'class': 'form-check-input'}),
47                 'category': forms.Select(attrs={'class': 'form-control'})
48             }
49

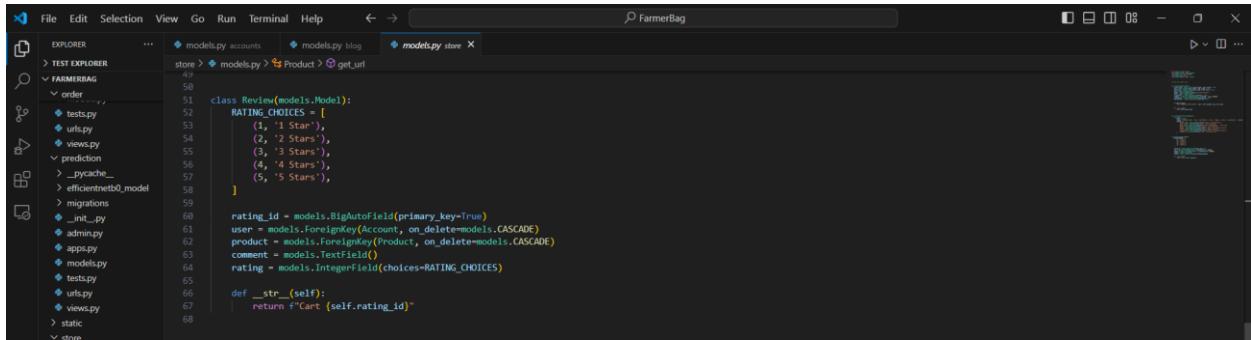
```

Figure 66: Database Creation: PRODUCT



The screenshot shows the SOLLITE database interface with the table 'store_product' created. The table has 11 columns:

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	id	integer	key							NULL
2	product_name	varchar (200)			cube					NULL
3	slug	varchar (200)			cube					NULL
4	description	text								NULL
5	price	integer								NULL
6	images	varchar (100)								NULL
7	stock	integer								NULL
8	is_available	bool								NULL
9	created_date	datetime								NULL
10	modified_date	datetime								NULL
11	category_id	integer		cube						NULL

Figure 67: Database Creation: SOLLITE (PRODUCT)


```

File Edit Selection View Go Run Terminal Help ← → FarmerBag
EXPLORER TEST EXPLORER FARMERBAG order
models.py accounts models.py blog models.py store
store > models.py Product > get_url
49
50
51 class Review(models.Model):
52     RATING_CHOICES = [
53         (1, '1 Star'),
54         (2, '2 Stars'),
55         (3, '3 Stars'),
56         (4, '4 Stars'),
57         (5, '5 Stars'),
58     ]
59
60 rating_id = models.BigAutoField(primary_key=True)
61 user = models.ForeignKey(Account, on_delete=models.CASCADE)
62 product = models.ForeignKey(Product, on_delete=models.CASCADE)
63 comment = models.TextField()
64 rating = models.IntegerField(choices=RATING_CHOICES)
65
66 def __str__(self):
67     return f"Cart {self.rating_id}"
68

```

Figure 68: Database Creation: Review

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated		Default value
1	rating_id	integer	key				not null				NULL
2	comment	text					not null				NULL
3	rating	integer					not null				NULL
4	product_id	integer		key			not null				NULL
5	user_id	integer		key			not null				NULL

Figure 69: Database Creation: SOLite (REVIEW)

3.6.2. PROJECT STRUCTURE

3.6.2.1. DJANGO APPLICATIONS

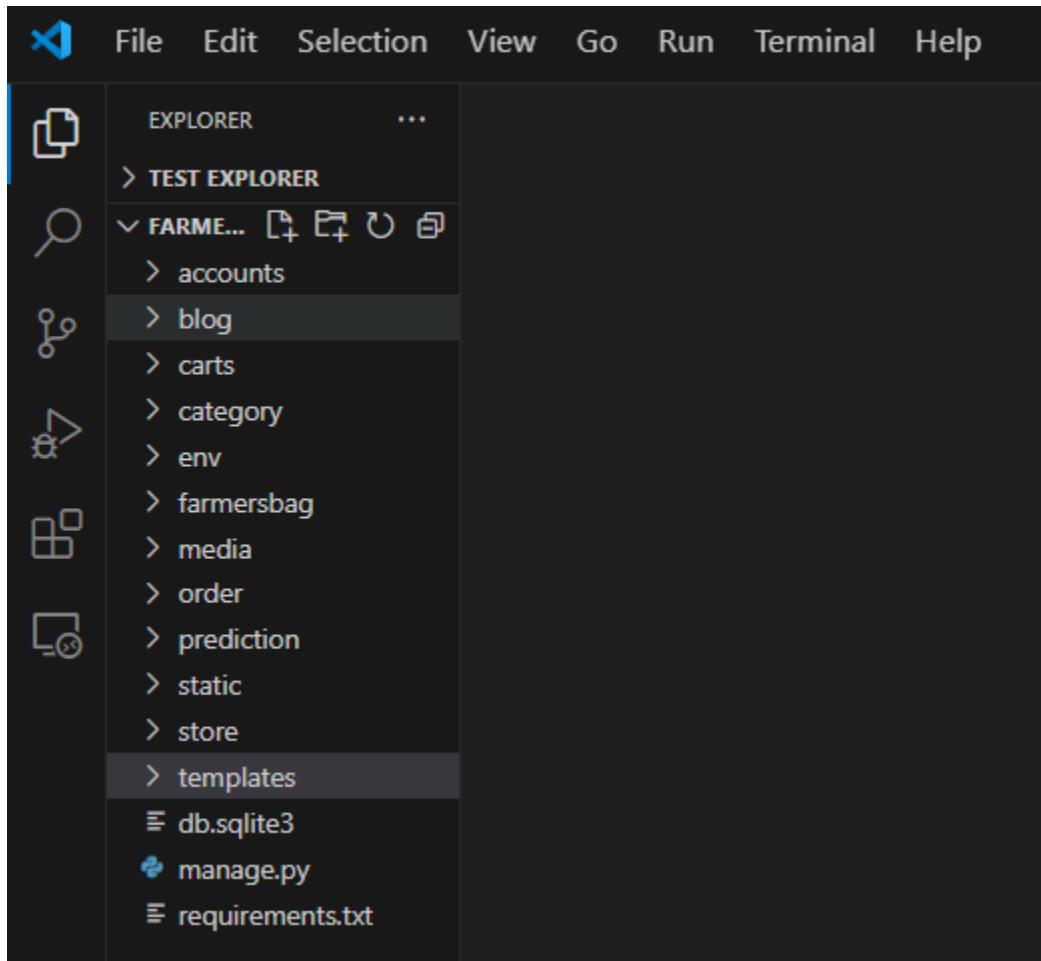
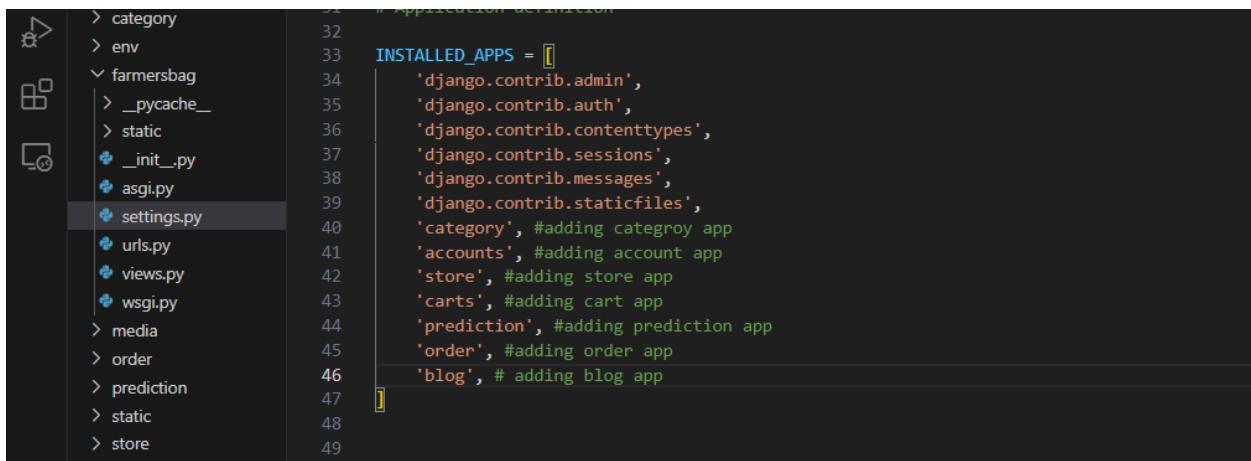


Figure 70: Implementation: Django Applications

This Django project consists of multiple apps, including accounts, blog, carts, category, farmer bag (the core app), order, prediction, and store. It follows the Model View Template (MVT) architecture, which promotes code organization and modularity by separating functionality into distinct apps. This structure simplifies debugging, future improvements, and collaboration by isolating app responsibilities. Overall, it leverages Django's features to create a well-organized and scalable web application.

3.6.2.1. REGISTERED DJANGO APPLICATIONS



```
 31     # Application definition
 32
 33 INSTALLED_APPS = [
 34     'django.contrib.admin',
 35     'django.contrib.auth',
 36     'django.contrib.contenttypes',
 37     'django.contrib.sessions',
 38     'django.contrib.messages',
 39     'django.contrib.staticfiles',
 40     'category', #adding category app
 41     'accounts', #adding account app
 42     'store', #adding store app
 43     'carts', #adding cart app
 44     'prediction', #adding prediction app
 45     'order', #adding order app
 46     'blog', # adding blog app
 47
 48
 49 ]
```

Figure 71: Screenshot of registering apps to settings.py file.

Since, farmer bag is the core app where we need to registered other apps in the setting.py. Since the below code shows the accounts, category, store, carts, prediction, order and blog are registered in the installed apps section.

3.6.3. NORMAL FEATURES

3.6.3.1. STATIC FILE REGISTRATION

```
154
135     STATIC_URL = '/static/'
136     STATIC_ROOT = os.path.join(BASE_DIR, 'static')
137     STATICFILES_DIRS = [
138         'farmersbag/static'
139     ]
140
141 #media files configurations
```

Figure 72: Registering Static file in core app.

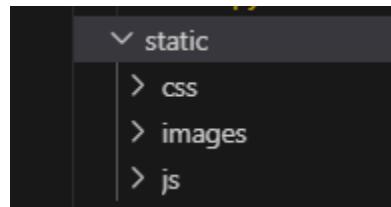


Figure 73: Normal Features: Registered Static Files

3.6.3.2. MEDIA FILE REGISTRATION

```
#media files configurations
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR /'media'
```

Figure 74: Normal Features: Media File Registration

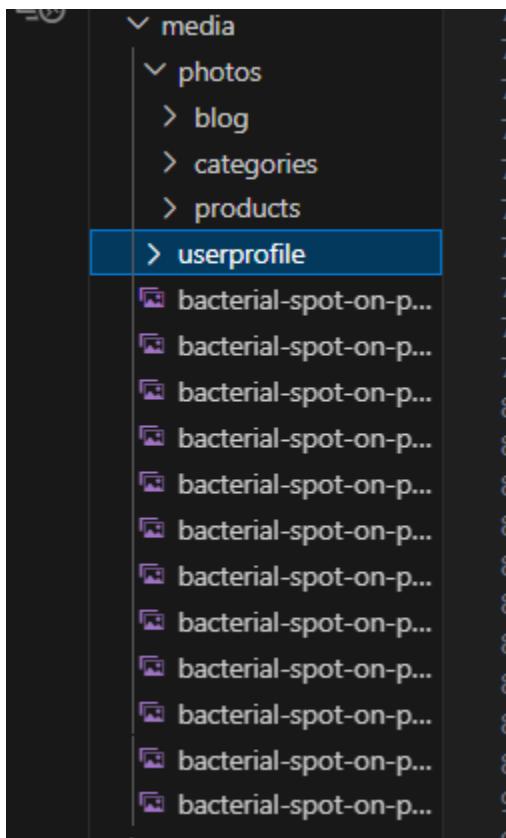


Figure 75: Normal Features: Registered Media Files

3.6.3.3. DATABASE REGISTRATION

```
# using default database as sql lite3
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Figure 76: Normal Features: Database Registration

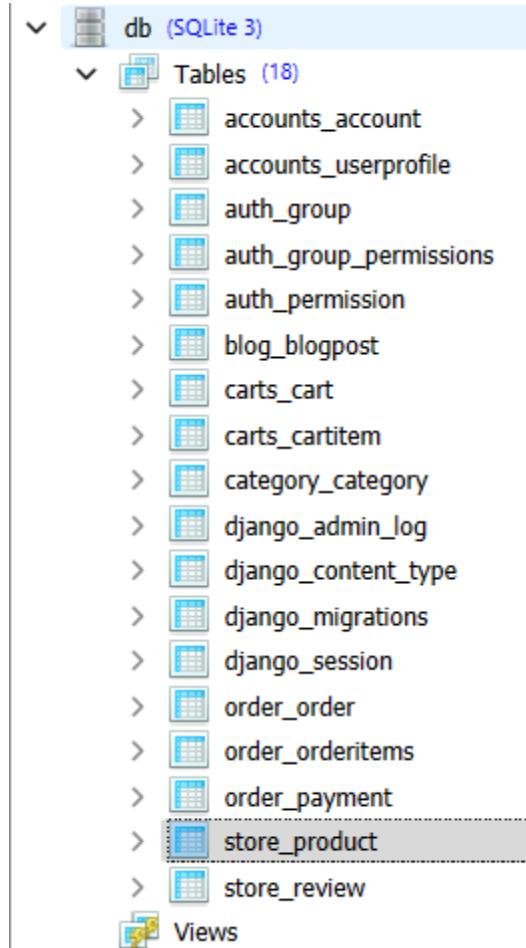


Figure 77: Normal Features: Database: SQLite

3.6.3.4. REGISTERING THE PROJECT URL'S

```

1  from django.contrib import admin
2  from django.urls import path, include
3  from . import views
4  from django.conf import settings
5  from django.conf.urls.static import static
6
7  urlpatterns = [
8      path('admin/', admin.site.urls),
9      path('', views.home, name='home'),
10     path('store/', include('store.urls')),
11     path('cart/', include('carts.urls')),
12     path('prediction/', include ('prediction.urls')),
13     path('', include('blog.urls')),
14
15     path('accounts/', include('accounts.urls')),
16     path('order/', include('order.urls')),
17
18 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
19
20
21
22

```

Figure 78: Normal Features: Registering Project URLs

3.6.3.5. TEMPLATES REGISTRATION

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'], #we are usong templates so that we should declare this.
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'category.context_processors.menu_links',
                'carts.context_processors.counter',
            ],
        },
    },
]

```

Figure 79: Normal Features: Registering Templates

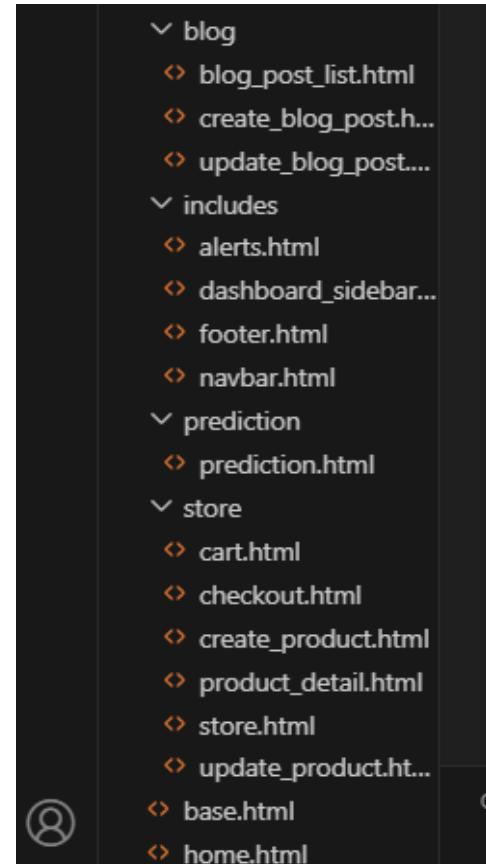
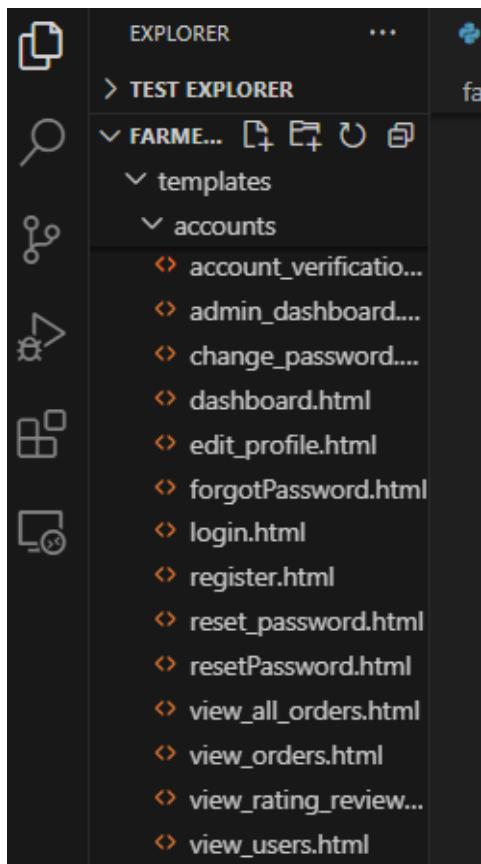


Figure 80 : Normal Features: Registered Templates 1

Figure 81: Normal Features: Registered Templates 2

3.6.3.6. REGISTER AND LOGIN USERS

```

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            first_name = form.cleaned_data['first_name']
            last_name = form.cleaned_data['last_name']
            phone_number = form.cleaned_data['phone_number']
            email = form.cleaned_data['email']
            password = form.cleaned_data['password']
            username = email.split("@")[0]

            # Creating user object
            user = Account.objects.create_user(first_name=first_name, last_name=last_name, email=email, username=username, password=password)
            user.phone_number = phone_number
            user.save()

            # USER ACTIVATION

            current_site = get_current_site(request)
            domain = current_site.domain
            mail_subject = 'Please Activate Your Account'
            message = render_to_string('accounts/account_verification_email.html', {
                'user': user,
                'domain': domain,
                'uid': urlsafe_base64_encode(force_bytes(user.pk)),
                'token': default_token_generator.make_token(user),
            })

            to_email = email
            send_email = EmailMessage(mail_subject, message, to=[to_email])
            send_email.send()

            #messages.success(request, 'Thank You for registering with us...')
            return redirect('/accounts/login/?command=verification&email=' + email)
        else:
            form = RegistrationForm()
            context = {'form': form}
            return render(request, 'accounts/register.html', context)
    
```

Figure 82: Normal Features: Register User Function

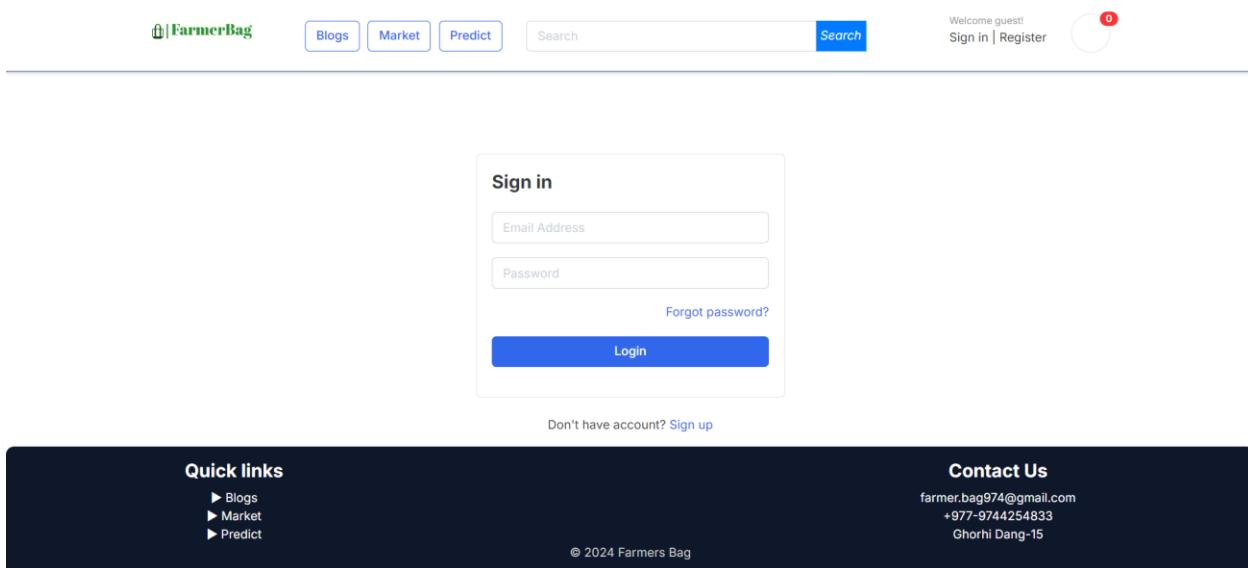
```

def login(request):
    if request.method == "POST":
        email = request.POST['email']
        password = request.POST['password']
        user = auth.authenticate(email = email, password=password)

        if user is not None:
            try:
                cart = Cart.objects.get(cart_id = _cart_id(request))
                print(cart)
                is_cart_item_exists = CartItem.objects.filter(cart=cart).exists()
                if is_cart_item_exists:
                    cart_item = CartItem.objects.filter(cart=cart)
                    print(cart_item)
                    for item in cart_item:
                        item.user = user
                        item.save()

            except:
                pass
            auth.login(request, user)
            messages.error(request, 'You are now logged now')
            return redirect('dashboard')
        else:
            messages.error(request, "Invalid login credentials")
            return redirect('login')
    return render (request, 'accounts/login.html')

```

Figure 83: Normal Features: Login User Function*Figure 84: Normal Functions: Login UI*

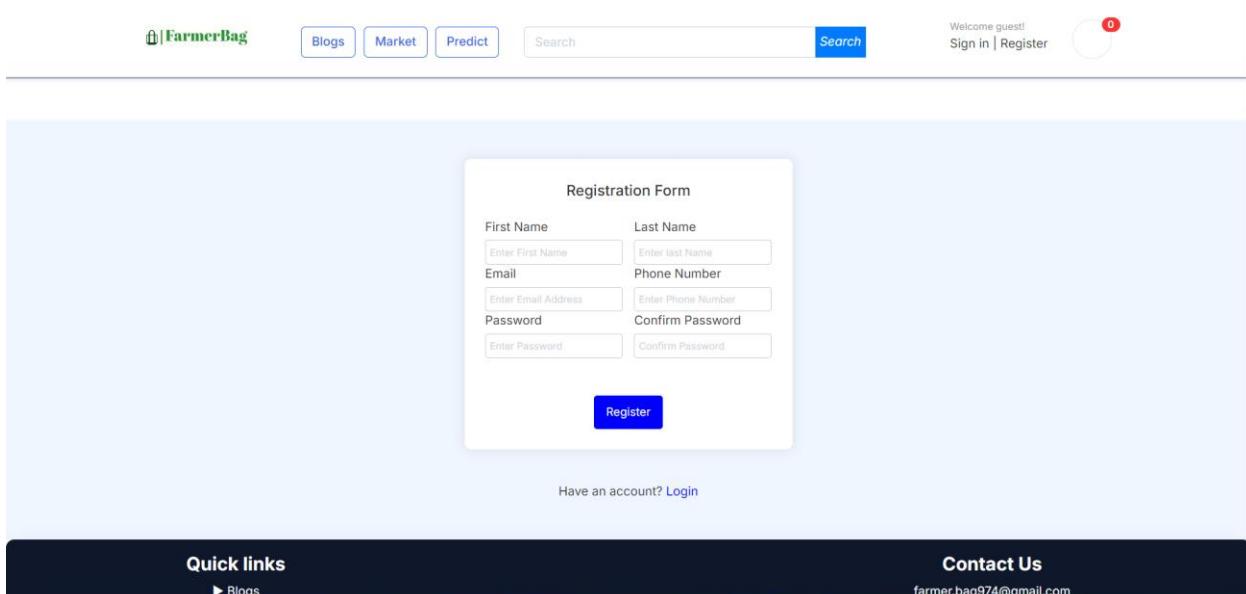
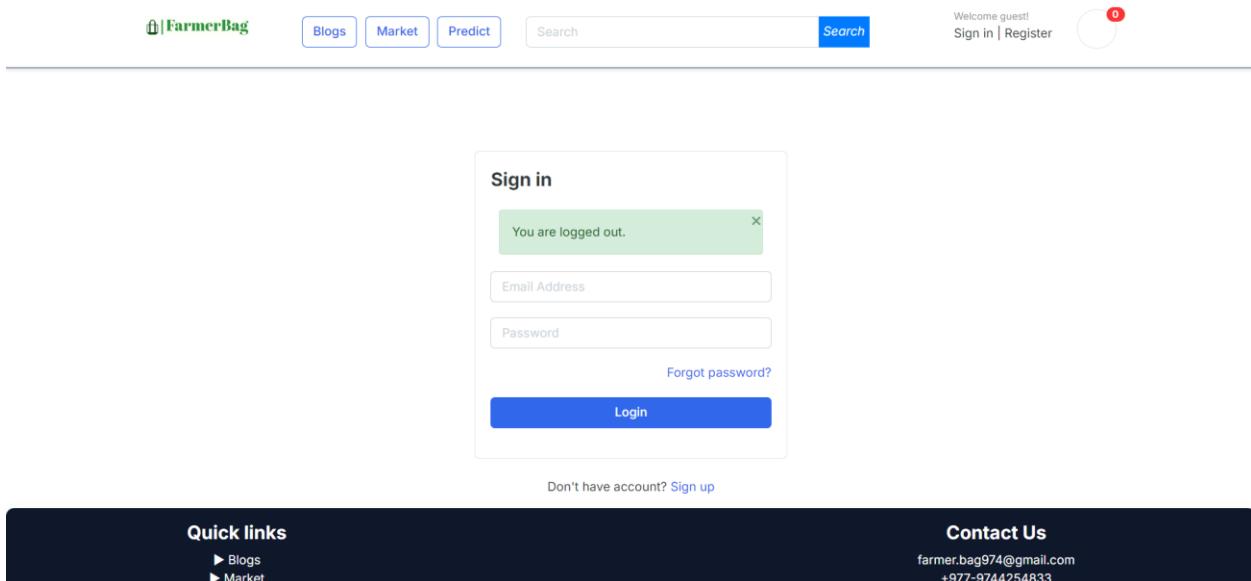


Figure 85: Normal Features: Register User UI

3.6.3.7. LOGOUT USER

```
@login_required(login_url='login')
def logout(request):
    auth.logout(request)
    messages.success(request, 'You are logged out.')
    return redirect('login')
```

Figure 86: Normal Features: Logout Function

**Figure 87: Normal Feature: Logout UI**

3.6.3.8. EDIT PROFILE

```
@login_required(login_url='login')
def edit_profile(request):
    # Retrieve or create UserProfile for the current user
    userprofile, created = UserProfile.objects.get_or_create(user=request.user)

    if request.method == 'POST':
        user_form = UserForm(request.POST, instance=request.user)
        profile_form = UserProfileForm(request.POST, request.FILES, instance=userprofile)
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
            messages.success(request, 'Your profile has been updated.')
            return redirect('edit_profile')
    else:
        user_form = UserForm(instance=request.user)
        profile_form = UserProfileForm(instance=userprofile)

    context = {
        'user_form': user_form,
        'profile_form': profile_form,
        'userprofile': userprofile,
    }
    return render(request, 'accounts/edit_profile.html', context)
```

Figure 88: Normal Feature: Edit Profile

The screenshot shows the 'Edit your profile' page of the FarmerBag application. On the left, there is a sidebar with navigation links: Dashboard, My Orders, Edit Profile (which is highlighted in blue), and Change Password. Below these are Log out and a 'Save' button. The main area contains fields for First Name (Bro), Last Name (K.C.), Phone Number (9744254833), Primary Address, Temporary Address, City, Country, and Province. There is also a 'Profile Picture' section with a placeholder 'Choose File No file chosen'. A circular profile picture placeholder is shown on the right. At the top, there is a header bar with the logo 'FarmerBag', navigation tabs for Blogs, Market, Predict, a search bar, a 'Search' button, and user information 'Welcome Bro! Dashboard | Logout'.

Figure 89: Normal Feature: Edit Profile UI

3.6.3.9. ADMIN DASHBOARD

```

def admin_dashboard(request):
    active_users_count = Account.objects.filter(is_active=True).count()

    payment_types = Payment.objects.values('payment_method').annotate(count=Count('id'))

    end_date = datetime.now()
    start_date = end_date - timedelta(days=30) # Sales trend for the last 30 days
    sales_trend_data = Order.objects.filter(order_date__range=(start_date, end_date))\
        .annotate(day=Count('order_date'))\
        .values('day')

    total_payment_price = Payment.objects.aggregate(total_price=Count('id'))

    categories = Category.objects.all()
    product_counts = []
    for category in categories:
        product_count = Product.objects.filter(category=category).count()
        product_counts.append({'category_name': category.category_name, 'product_count': product_count})

    total_rating = Review.objects.aggregate(total_rating=Sum('rating'))

    context = {
        'active_users_count': active_users_count,
        'payment_types': payment_types,
        'sales_trend_data': sales_trend_data,
        'total_payment_price': total_payment_price,
        'product_counts': product_counts,
        'total_rating': total_rating['total_rating'], # Add total rating data
    }
    return render(request, 'accounts/admin_dashboard.html', context)

```

Figure 90: Normal Feature: Admin Dashboard

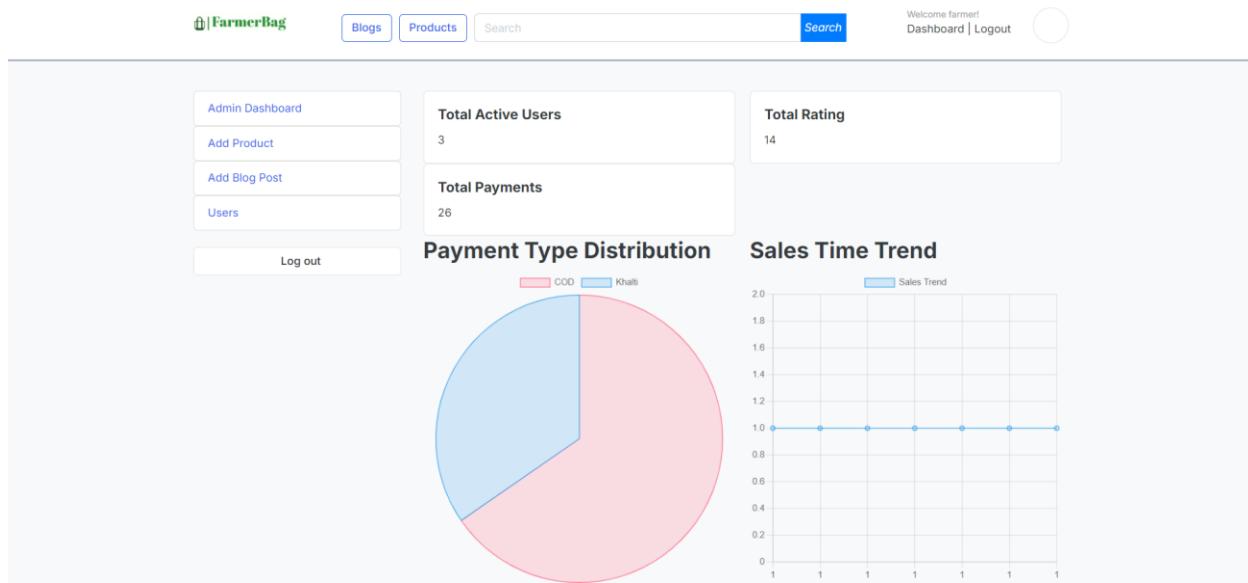


Figure 91: Normal Feature: Admin Dashboard UI(I)

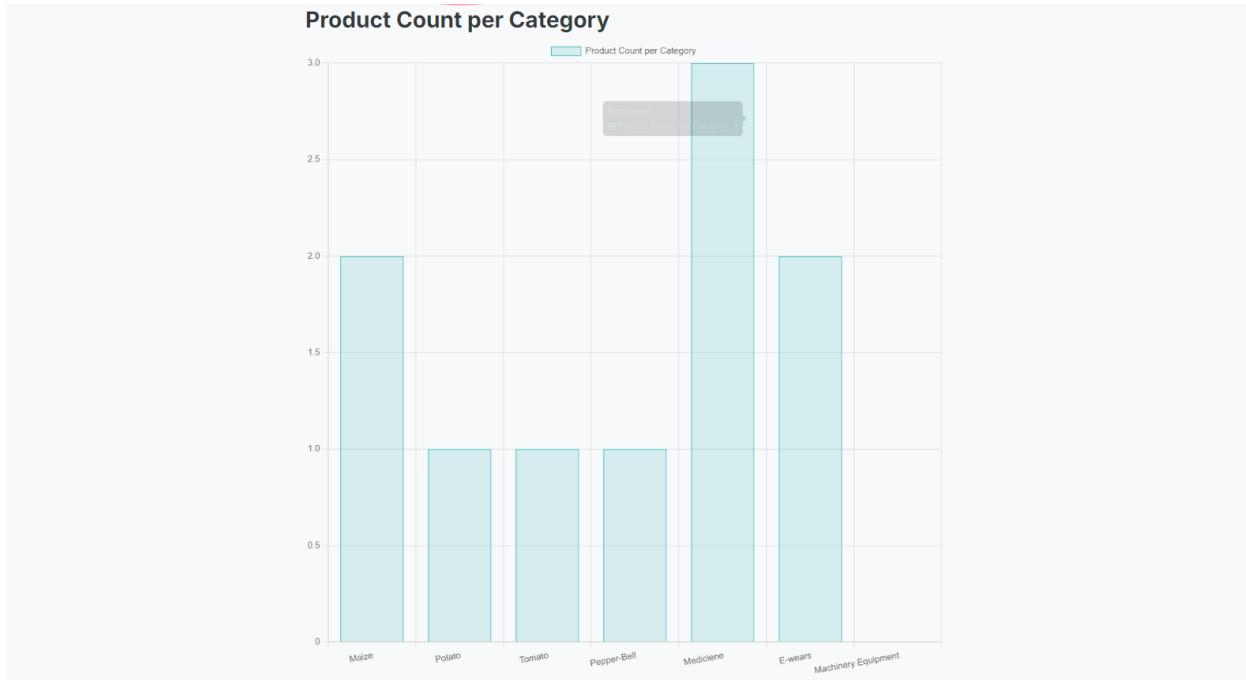


Figure 92: Normal Feature: Admin Dashboard UI (2)

3.6.4. CORE FEATURES

3.6.2.1. PLANT DISEASE DETECTION AND RECOMMENDATIONS

```
from django.shortcuts import render
from django.core.files.storage import FileSystemStorage
import numpy as np
import tensorflow as tf
from keras.preprocessing import image
import json
from store.models import Product
from django.db.models import Q
from django.contrib.auth.decorators import login_required
import tensorflow.keras.models as models
import os
from django.http import HttpResponseRedirect
from django.utils.datastructures import MultiValueDictKeyError
```

Figure 93: Screenshot Importing Libraries for Plant Disease Detection.

```
# Create your views here.
@login_required(login_url='login')
def prediction(request):
    return render(request, 'prediction/prediction.html')

img_height, img_width = 224, 224
```

Figure 94: Screenshot prediction function of Plant disease detection.

```
# Import OpenCV
import cv2

# Utility
import os
from glob import iglob
import numpy as np

IMAGE_SHAPE = (224, 224)

def load_image(filename):
    img = cv2.imread(filename)
    assert not isinstance(img, type(None)), 'image not found'

    img = cv2.resize(img, (IMAGE_SHAPE[0], IMAGE_SHAPE[1]))
    img = img / 255.0

    return img

def predict(model, image):
    probabilities = model.predict(np.asarray([image]))[0]
    class_idx = np.argmax(probabilities)

    return int(class_idx), probabilities[class_idx]
```

Figure 95: Screenshot of loading and predicting function of Plant Disease Detection

```
class_labels = ['Apple__Black_rot', 'Apple__healthy', 'Corn_(maize)__Cercospora_leaf_spot_Gray_leaf_spot',
'Corn_(maize)__Common_rust', 'Corn_(maize)__Northern_Leaf_Blight', 'Corn_(maize)__healthy',
'Grape__Black_rot', 'Grape__Esca_(Black_Measles)', 'Grape__healthy', 'Peach__Bacterial_spot',
'Peach__healthy', 'Pepper,_bell__Bacterial_spot', 'Pepper,_bell__healthy', 'Potato__Early_blight',
'Potato__Late_blight', 'Potato__healthy', 'Strawberry__Leaf_scorch', 'Strawberry__healthy',
'Tomato__Bacterial_spot', 'Tomato__Early_blight', 'Tomato__Late_blight', 'Tomato__Leaf_Mold',
'Tomato__Septoria_leaf_spot', 'Tomato__healthy', 'Unknown_Class']
```

Figure 96: Screenshot of defining class labels.

```
# Load the model
model = models.load_model("prediction/efficientnetb0_model")

def recommend_products(predicted_label):
    # Convert predicted_label index to string label
    predicted_label_str = class_labels[predicted_label]

    # Extract the main category label
    main_category = predicted_label_str.split("__")[0]

    # Query products with descriptions or titles containing the main category label
    products = Product.objects.filter(
        Q(description__icontains=main_category) |
        Q(product_name__icontains=main_category)
    )

    return products
```

Figure 97: Screenshot of loading model and creating recommend product function

```
def imgPrediction(request):
    if request.method == 'POST':
        try:
            fileObj = request.FILES['filePath']
            fs = FileSystemStorage()
            filePathNames = fs.save(fileObj.name, fileObj)
            filePathName = './media/' + filePathNames

            testimage = filePathName

            img = load_image(testimage)
            predicted_label, confidence = predict(model, img)
            predictedLabel = class_labels[predicted_label]

            # Get recommended products based on the predicted label
            recommended_products = recommend_products(predicted_label=predicted_label)

            context = {
                'filePathName': filePathName,
                'predictedLabel': predictedLabel,
                'confidence': confidence,
                'recommended_products': recommended_products, # Pass recommended products to the context
            }

            return render(request, 'prediction/prediction.html', context)

        except MultiValueDictKeyError:
            # Handle the case where 'filePath' is not found in request.FILES
            error_message = "Please upload a file."
            context = {'error_message': error_message}
            return render(request, 'prediction/prediction.html', context)

    # Handle GET requests or other methods
    return HttpResponseBadRequest("Method not allowed")
```

Figure 98: Screenshot of Plant disease detection function

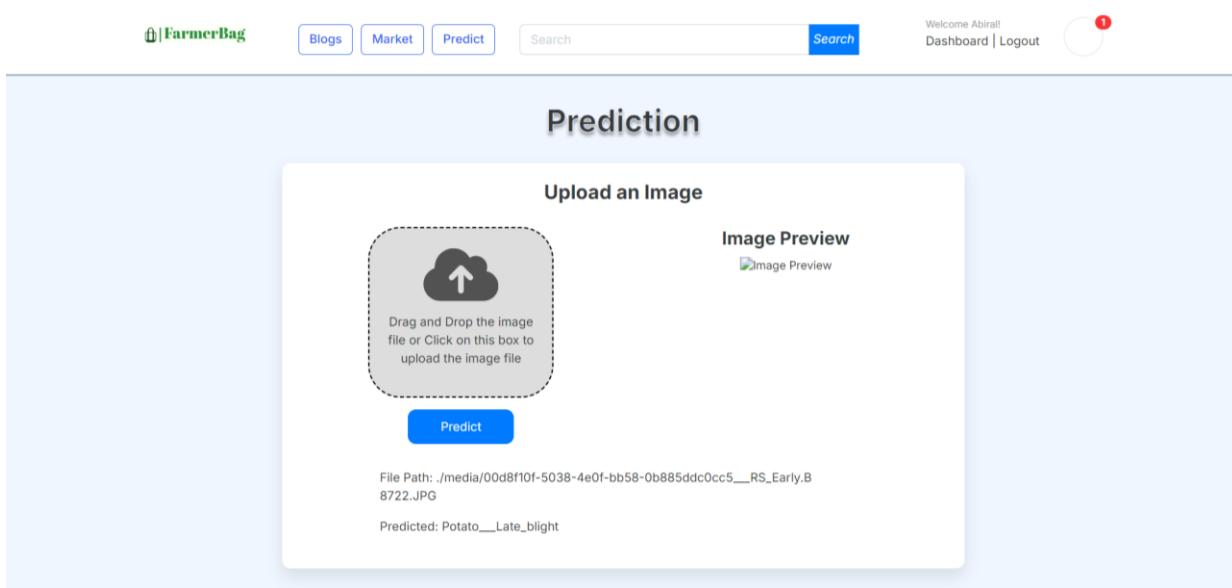


Figure 99: Screenshot of UI of Prediction Page

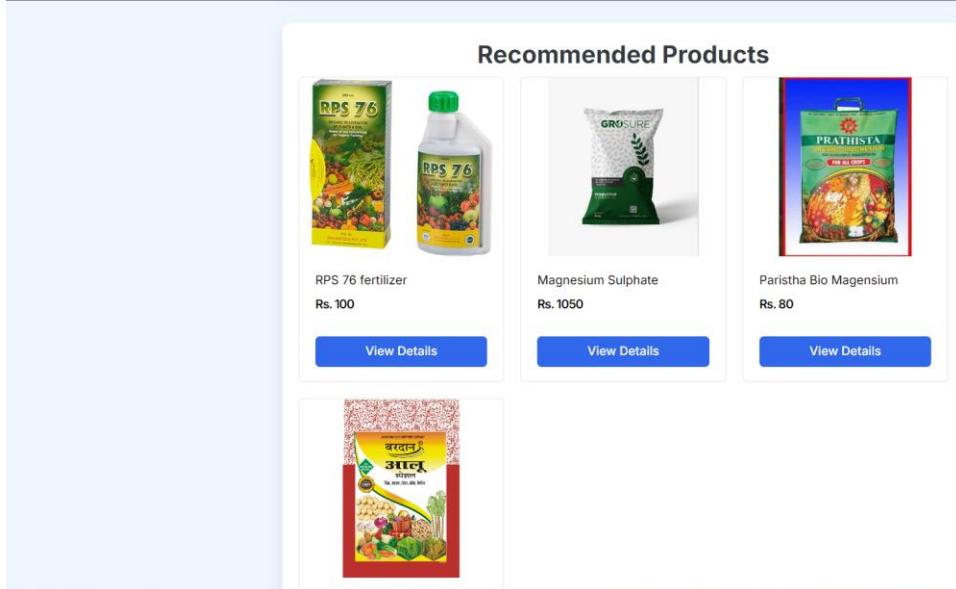


Figure 100: Screenshot of UI of Recommend Product

3.6.5. SYSTEM ARCHITECTURE DIAGRAM

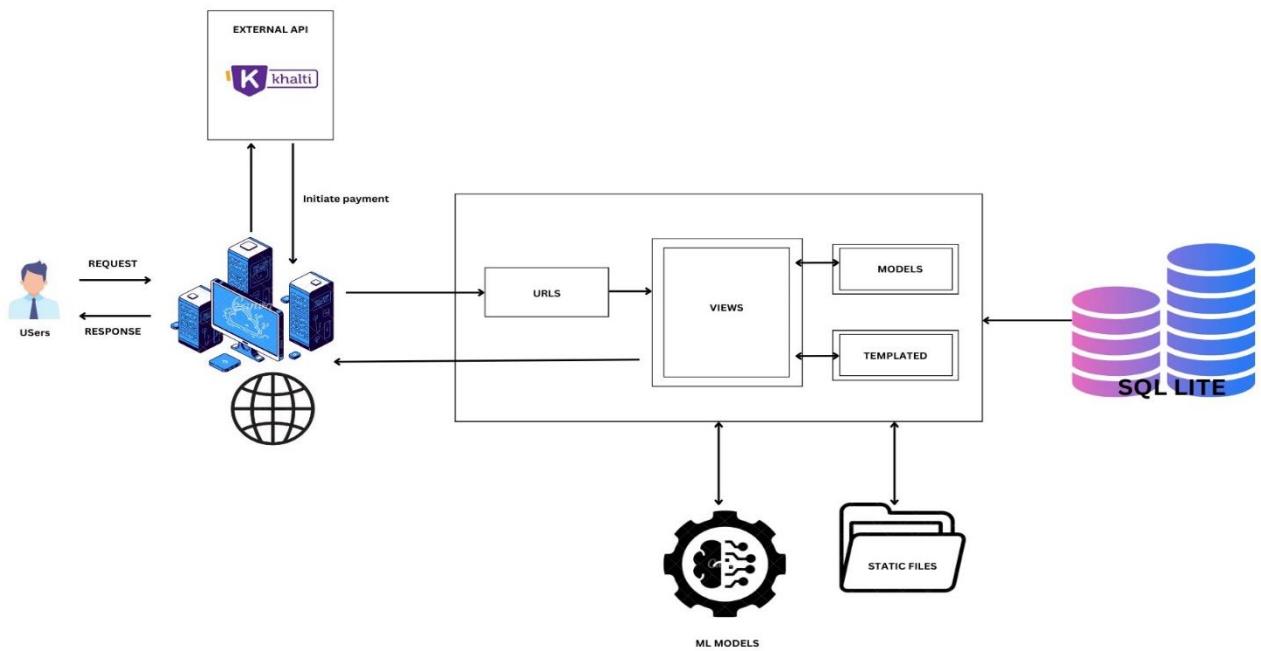


Figure 101: System Architecture Diagram

Appendix: [7.5.4.1. SYSTEM ARCHITECTURE DIAGRAM]

4. TESTING AND ANALYSIS

4.1. TEST PLAN

The test plan will provide detailed guidelines for evaluating the application's security, performance, and functionality. It will have comprehensive test cases to ensure the system's strength and reliability. The plan will also include approaches and tools for extensive testing in various situations to identify and fix any potential issues before the application is launched.

4.1.1. BLACK-BOX TESTING

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of the internal code structure, implementation details, and internal paths. Black Box Testing mainly focuses on the inputs and outputs of software applications, and it is entirely based on software requirements and specifications. It is also known as Behavioural Testing. (Thomas Hamilton, 2024)

4.1.2. WHITE-BOX TESTING

White box testing evaluates a software application's code and internal structure. It ensures that internal operations are carried out according to the specifications when you know the application's internal structure. Additionally, each internal component must follow the proper framework. (LambdaTest, 2024)

4.1.3. INTEGRATION TESTING

Integration testing - also known as integration and testing (I&T) - is a type of software testing in which the different units, modules or components of a software application are tested as a combined entity, even though these modules may be coded by different programmers. The aim of integration testing is to test the interfaces between the modules and expose any defects that may arise when these components are integrated and need to interact with each other. (Awati, 2023)

4.2. BLACK-BOX TESTING

4.2.1. TEST 1 – TO CHANGE PASSWORD

TEST	One
Objective	To verify logged in user can change their password if they want it.
Action	<ul style="list-style-type: none"> ➤ Navigate to the user Dashboard. ➤ Click on the “Change Password” on the left bottom side. ➤ Change password form displays and enter previous set password and new password. ➤ Click on submit button. ➤ The system redirects to the login page. ➤ Enter invalid details system should fail the test case.
Expected Result	<ul style="list-style-type: none"> ➤ The user should successfully be able to change password and redirect to login page and after filling new credentials user should successfully redirect to dashboard.
Actual Result	The user was successfully able to change the password and was able to redirect to dashboard with new credentials.
Conclusion	The test was successful.

Table 2: Black-Box Testing: To Change Password

Quick links

- Blogs
- Market
- Predict

© 2024 Farmers Bag

Contact Us

farmer.bag974@gmail.com
+977-9744254833
Ghorhi Dang-15

Figure 102: Change Password: Form

Quick links

- Blogs
- Market
- Predict

© 2024 Farmers Bag

Contact Us

farmer.bag974@gmail.com
+977-9744254833
Ghorhi Dang-15

Figure 103: Change Password: Valid Details

The screenshot shows the FarmerBag login page. At the top, there is a navigation bar with links for Blogs, Market, Predict, and Search. A user profile icon indicates 0 notifications. The main area features a "Sign in" form with fields for Email Address and Password, and links for "Forgot password?" and "Login". A green success message box at the top says "Password updated successfully." Below the form, a link says "Don't have account? Sign up". At the bottom, there are "Quick links" for Blogs, Market, and Predict, and a "Contact Us" section with email and phone number.

Figure 104: Change Password: Redirect to login page with success message.

The screenshot shows the FarmerBag change password page. The left sidebar has links for Dashboard, My Orders, Edit Profile, and a prominent blue "Change Password" button. The main area contains a "Change Your Password" form with fields for Current Password, Create new password, and Confirm new password, all containing placeholder text "*****". A "Submit" button is at the bottom. At the top, there is a navigation bar with Blogs, Market, Predict, and Search, and a user profile icon indicating 1 notification. The bottom features a "Quick links" sidebar and a "Contact Us" section with contact information.

Figure 105: Change Password: Invalid Details

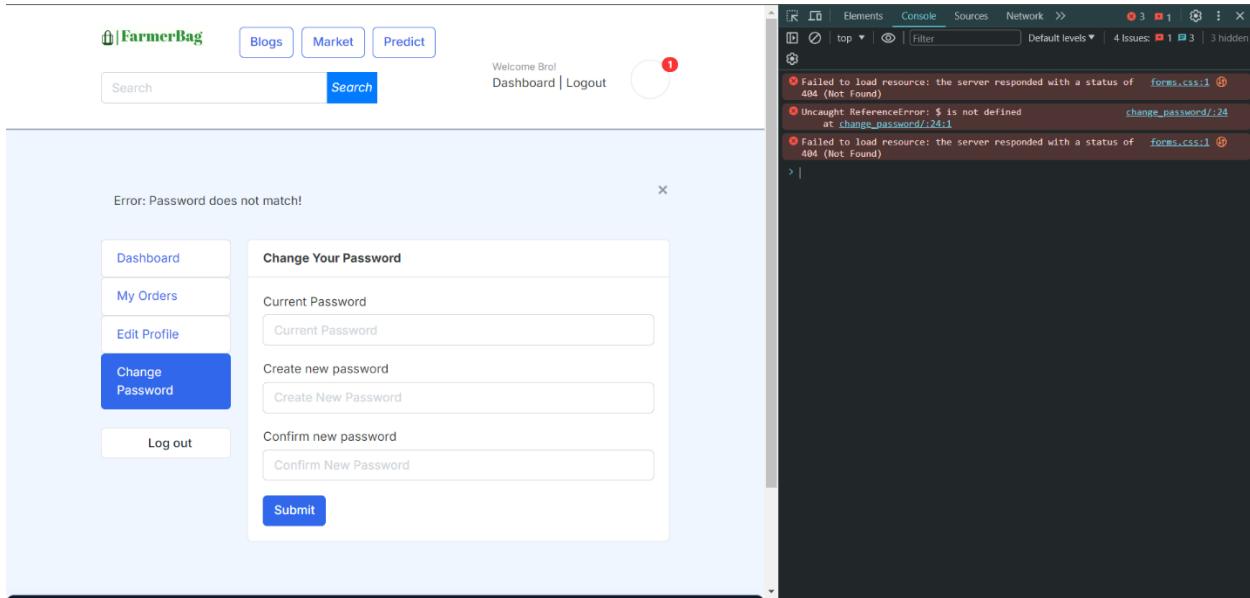


Figure 106: Change Password: Error Message Displays and 404 Not Found Error

4.2.2. TEST 2 – PREDICTION

TEST	Two
Objective	To detect the plant disease of leaf.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Navigation bar. ➤ Click on the “Predict” button on navigation bar. ➤ System should redirect to Prediction page. ➤ Click on predict button without selecting image.
Expected Result	System should redirect to the same page.
Actual Result	System through the MultiValueDictKeyError.
Conclusion	The test was unsuccessful.

Table 3: Test 2- Black Box Testing: Prediction “without selecting image”.

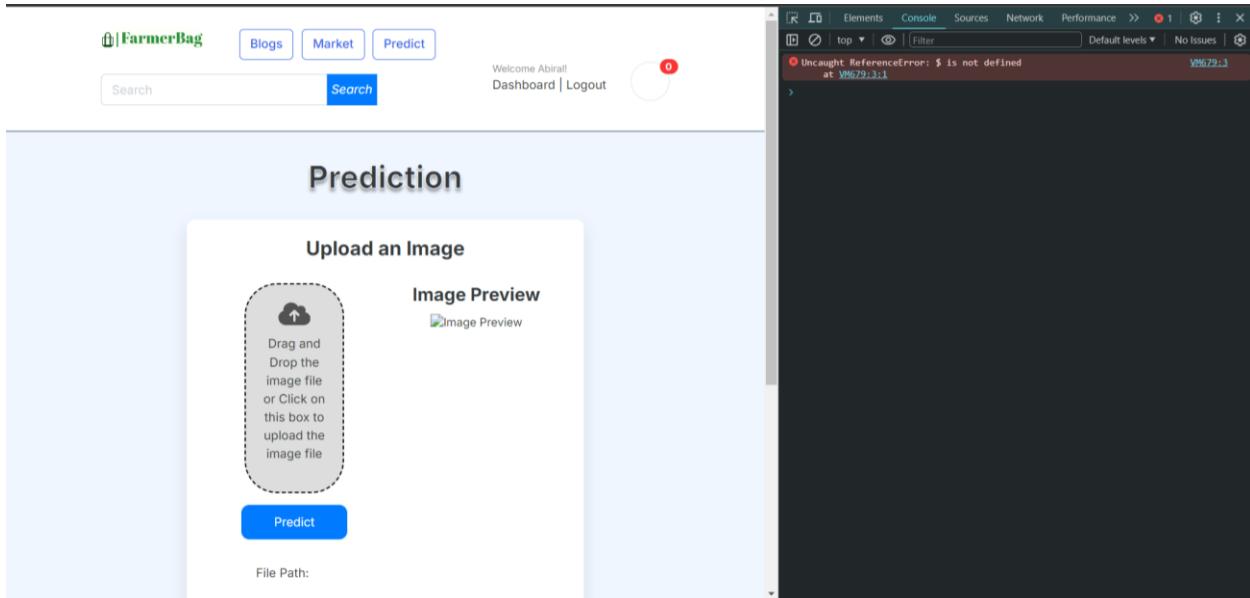


Figure 107: Prediction: System through Error "Uncaught Reference Error".

```
MultiValueDictKeyError at /prediction/imgPrediction
'filePath'

Request Method: POST
Request URL: http://127.0.0.1:8000/prediction/imgPrediction
Django Version: 3.1
Exception Type: MultiValueDictKeyError
Exception Value: 'filePath'
Exception Location: D:\@ Islington College\~\@ 3rd Year\~\Final Year Project\Development\FarmerBag\env\lib\site-packages\django\utils\datastructures.py, line 78, in __getitem__
Python Executable: D:\@ Islington College\~\@ 3rd Year\~\Final Year Project\Development\FarmerBag\env\Scripts\python.exe
Python Version: 3.11.9
Python Path: ['D:\@ Islington College\~\@ 3rd Year\~\Final Year ','Project\Development\farmebag','C:\Program Files\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbzsn2kfrdp0\python311.zip','C:\Program Files\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbzsn2kfrdp0\DLLs','C:\Program Files\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbzsn2kfrdp0\LIB','C:\Program Files\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbzsn2kfrdp0','D:\@ Islington College\~\@ 3rd Year\~\Final Year ','Project\Development\farmebag\env','D:\@ Islington College\~\@ 3rd Year\~\Final Year ','Project\Development\farmebag\env\lib\site-packages']
Server time: Tue, 24 April 2024 19:28:53 +0000

Traceback (most recent call last):
  File "D:\@ Islington College\~\@ 3rd Year\~\Final Year Project\Development\FarmerBag\env\lib\site-packages\django\core\handlers\exception.py", line 47, in inner
    response = get_response(request)
  File "D:\@ Islington College\~\@ 3rd Year\~\Final Year Project\Development\FarmerBag\env\lib\site-packages\django\core\handlers\base.py", line 179, in get_response
    response = self.process_exception_by_middleware(e, request)

During handling of the above exception ('MultiValueDictKeyError'), another exception occurred:
  File "D:\@ Islington College\~\@ 3rd Year\~\Final Year Project\Development\FarmerBag\env\lib\site-packages\django\core\handlers\exception.py", line 47, in inner
    response = get_response(request)
  File "D:\@ Islington College\~\@ 3rd Year\~\Final Year Project\Development\FarmerBag\env\lib\site-packages\django\core\handlers\base.py", line 179, in get_response
    response = self.get_response(request)
```

Figure 108: Prediction: System through MultiValueDictKey Error

```

prediction > views.py > imgPrediction > predictedLabel
117 def imgPrediction(request):
118     # Get recommended products based on the predicted label
119     recommended_products = recommend_products(predicted_label=predictedLabel)
120
121     context = {
122         'filePathName': filePathName,
123         'predictedLabel': predictedLabel,
124         'confidence': confidence,
125         'recommended_products': recommended_products, # Pass recommended products to the context
126     }
127
128     return render(request, 'prediction/base.html', context)
129
130
131
132
133
134
135
136
137
138

```

Detailed description: This screenshot shows a code editor interface with several tabs open. The main tab contains Python code for a view named 'imgPrediction'. The code imports necessary modules and defines a function that retrieves recommended products based on a predicted label. It then creates a context dictionary containing the file path name, predicted label, confidence, and recommended products, and renders a template ('base.html') with this context. Below the code editor is a terminal window displaying a stack trace for a Django exception. The exception occurred at line 47 of 'exception.py' in the 'django.core.handlers' module, specifically in the 'get_response' method. The error message indicates a 'MultiValueDictKeyError' was raised because the key 'callback' was not found in the request.FILES dictionary.

Figure 109: Prediction: Error in console.

TEST	Handling Test 2 - Error
Action	Handling MultiValueDictKey exception using try and except.
Expected Result	<ul style="list-style-type: none"> ➤ System should redirect to the same page. ➤ System should displays the “Please Upload File”
Actual Result	<ul style="list-style-type: none"> ➤ System redirected user to the same page. ➤ System displayed the “Please Upload File” if predict button was clicked.
Conclusion	The test was successful.

Table 4: Black Box Testing: Handling MultiValueKeyDict Error

```
def imgPrediction(request):
    if request.method == 'POST':

        fileObj = request.FILES['filePath']
        fs = FileSystemStorage()
        filePathNames = fs.save(fileObj.name, fileObj)
        filePathName = './media/' + filePathNames

        testimage = filePathName

        img = load_image(testimage)
        predicted_label, confidence = predict(model, img)
        predictedLabel = class_labels[predicted_label]

        # Get recommended products based on the predicted label
        recommended_products = recommend_products(predicted_label=predicted_label)

        context = {
            'filePathName': filePathName,
            'predictedLabel': predictedLabel,
            'confidence': confidence,
            'recommended_products': recommended_products, # Pass recommended products to the context
        }

    return render(request, 'prediction/prediction.html', context)
```

Figure 110: Prediction: Place this code to try and expect block with MultiValueDictKey Error

```

def imgPrediction(request):
    if request.method == 'POST':
        try:
            fileObj = request.FILES['filePath']
            fs = FileSystemStorage()
            filePathNames = fs.save(fileObj.name, fileObj)
            filePathName = './media/' + filePathNames

            testimage = filePathName

            img = load_image(testimage)
            predicted_label, confidence = predict(model, img)
            predictedLabel = class_labels[predicted_label]

            # Get recommended products based on the predicted label
            recommended_products = recommend_products(predicted_label=predicted_label)

            context = {
                'filePathName': filePathName,
                'predictedLabel': predictedLabel,
                'confidence': confidence,
                'recommended_products': recommended_products, # Pass recommended products to the context
            }

            return render(request, 'prediction/prediction.html', context)

        except MultiValueDictKeyError:
            # Handle the case where 'filePath' is not found in request.FILES
            error_message = "Please upload a file."
            context = {'error_message': error_message}
            return render(request, 'prediction/prediction.html', context)

    # Handle GET requests or other methods
    return HttpResponseBadRequest("Method not allowed")

```

Figure 111: Prediction: Place code inside try and expect block with MultiValueDictKey Error

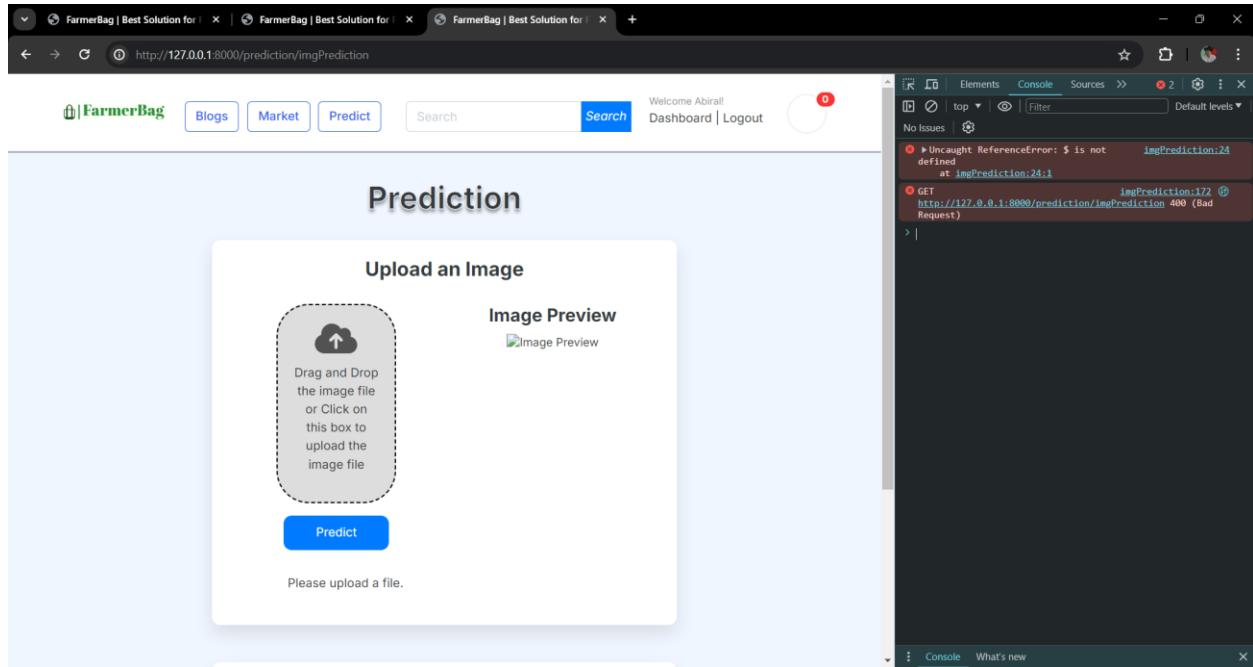


Figure 112: Prediction: Message Displayed "Please Upload a file".

4.2.2. TEST 3 – SEARCH BAR BUTTON

TEST	Three
Objective	To search product, form the search bar.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Navigation bar. ➤ Click on the “search” button without any keywords. ➤ System should redirect to store page with message “No Product Found”.
Expected Result	System should redirect to store page with “No Product Found” message.
Actual Result	System displays error.
Conclusion	The test was unsuccessful.

Table 5: Black Box Testing: Search Bar Button

The screenshot shows the 'Our Store' page of the FarmerBag website. At the top, there is a navigation bar with links for Blogs, Market, Predict, and a search bar. The search bar is currently empty. To the right of the search bar, it says 'Welcome Abiraj!' and provides links for Dashboard and Logout. Below the navigation bar, the page title 'Our Store' is displayed. On the left, there are two sidebar sections: 'Categories' (listing All Products, Maize, Potato, Tomato, Pepper-Bell, Mediciene, E-wears, and Machinery Equipment) and 'Price range' (with dropdown menus for Min and Max price, and an 'Apply' button). The main content area shows a grid of products. At the top of the grid, it says 'Found 10 items found'. The first three products listed are RPS 76 fertilizer (Rs. 100), Magnesium Sulphate (Rs. 1050), and Paristha Bio Magensium (Rs. 80). Each product has a 'View Details' button below its image.

Figure 113: Search Bar: Click on Search Bar Button without any keywords

```
UnboundLocalError at /store/search/
cannot access local variable 'products' where it is not associated with a value

Request Method: GET
Request URL: http://127.0.0.1:8000/store/search/?keyword=
Django Version: 3.1
Exception Type: UnboundLocalError
Exception Value: cannot access local variable 'products' where it is not associated with a value
Exception Location: D:\@ Islington College\@ 3rd Year~\Final Year Project\Development\FarmerBag\store\views.py line 104, in search
Python Executable: D:\@ Islington College\@ 3rd Year~\Final Year Project\Development\FarmerBag\env\Scripts\python.exe
Python Version: 3.11.9
Python Path: ['D:\\@ Islington College\\@ 3rd Year~\\Final Year~\\Project\\Development\\FarmerBag', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2kfrap0\\python311.zip', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2kfrap0\\DLLs', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2kfrap0\\Lib', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64_qbz5n2kfrap0', 'D:\\@ Islington College\\@ 3rd Year~\\Final Year~\\Project\\Development\\FarmerBag\\env', 'D:\\@ Islington College\\@ 3rd Year~\\Final Year~\\Project\\Development\\FarmerBag\\env\\lib\\site-packages']
Server time: Tue, 23 Apr 2024 19:56:57 +0000

Traceback Switch to copy-and-paste view
D:\\@ Islington College\\@ 3rd Year~\\Final Year Project\\Development\\FarmerBag\\env\\lib\\site-packages\\django\\core\\handlers\\exception.py, line 47, in inner
    response = get_response(request)
▶ Local vars
D:\\@ Islington College\\@ 3rd Year~\\Final Year Project\\Development\\FarmerBag\\env\\lib\\site-packages\\django\\core\\handlers\\base.py, line 179, in _get_response
    response = wrapped_callback(request, *callback_args, **callback_kwargs)
▶ Local vars
D:\\@ Islington College\\@ 3rd Year~\\Final Year Project\\Development\\FarmerBag\\store\\views.py, line 104, in search
    104.         'products': products,

```

Figure 114: Search Bar: System Displays Error Message

The screenshot shows the Visual Studio Code interface. The code editor displays a Python file named `views.py` with the following code:

```
def search(request):
    if 'keyword' in request.GET:
        keyword = request.GET['keyword']

    if keyword:
        products = Product.objects.order_by('-created_date').filter(Q(description_icontains= keyword) | Q(product_name_icontains= keyword))
        product_count = products.count()

    context = {
        'products': products,
        'product_count': product_count,
    }

    return render(request, 'store/store.html', context)

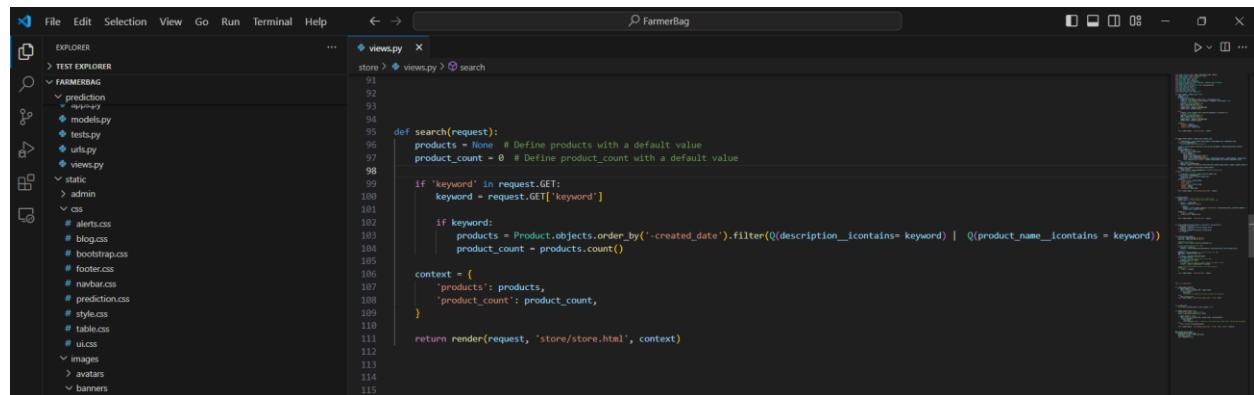
def filter_products_by_price(products, min_price=None, max_price=None):
    if min_price is not None:
        products = products.filter(price_gte=min_price)
    if max_price is not None:
        products = products.filter(price_lte=max_price)
    return products
```

The terminal below the code editor shows the error messages:

```
UnboundLocalError: cannot access local variable 'products' where it is not associated with a value
[24/Apr/2024 01:41:57] "GET /store/search/?keyword= HTTP/1.1" 500 69550
Internal Server Error: /store/search/
Traceback (most recent call last):
  File "D:\\@ Islington College\\@ 3rd Year~\\Final Year Project\\Development\\FarmerBag\\env\\lib\\site-packages\\django\\core\\handlers\\exception.py", line 47, in inner
    response = get_response(request)
           ^^^^^^^^^^^^^^
  File "D:\\@ Islington College\\@ 3rd Year~\\Final Year Project\\Development\\FarmerBag\\env\\lib\\site-packages\\django\\core\\handlers\\base.py", line 179, in _get_response
    response = wrapped_callback(request, *callback_args, **callback_kwargs)
           ^^^^^^^^^^^^^^
  File "D:\\@ Islington College\\@ 3rd Year~\\Final Year Project\\Development\\FarmerBag\\store\\views.py", line 104, in search
    ^^^^^^^
UnboundLocalError: cannot access local variable 'products' where it is not associated with a value
[24/Apr/2024 01:41:57] "GET /store/search/?keyword= HTTP/1.1" 500 69550
```

Figure 115: Search Bar: Error "Product Not associated to value".

TEST	Handling Test 3 - Error
Action	➤ In search function set default value product = None and product_count = 0
Expected Result	➤ System should display “No result found”
Actual Result	➤ System displayed “No result found” with product count 0
Conclusion	The test was successful.

Table 6: Test 3: Handling Error


A screenshot of a code editor (Visual Studio Code) showing a Python file named `views.py`. The code defines a `search` function that handles a search request. It initializes `products` to `None` and `product_count` to `0`. If a keyword is provided in the GET parameters, it filters products by the keyword and updates `product_count`. A context dictionary is then created containing the filtered products and the count. Finally, the `render` function is called with the store template and the context.

```

File Edit Selection View Go Run Terminal Help < - > FarmerBag
EXPLORER TEST EXPLORER FARMERBAG prediction models.py tests.py urls.py views.py
store > views.py > search
91
92
93
94
95
96 def search(request):
97     products = None # Define products with a default value
98     product_count = 0 # Define product_count with a default value
99
100    if 'keyword' in request.GET:
101        keyword = request.GET['keyword']
102
103        if keyword:
104            products = Product.objects.order_by('-created_date').filter(Q(description__icontains= keyword) | Q(product_name__icontains= keyword))
105            product_count = products.count()
106
107    context = {
108        'products': products,
109        'product_count': product_count,
110    }
111
112    return render(request, 'store/store.html', context)
113
114
115

```

Figure 116: Search Product: Set Products = None, product count = 0

The screenshot shows a web application interface for 'FarmerBag'. At the top, there is a navigation bar with links for 'Blogs', 'Market', 'Predict', 'Search' (which is highlighted in blue), and 'Dashboard | Logout'. A user profile icon with a red notification badge is also present. The main content area has a title 'Search Results' and a message 'Found 0 items found'. Below this, a bold message reads 'No result found. Please Try again!'. On the left side, there is a sidebar with two sections: 'Categories' and 'Price range'. The 'Categories' section lists items like All Products, Maize, Potato, Tomato, Pepper-Bell, Mediciene, E-wears, and Machinery Equipment. The 'Price range' section contains input fields for 'Min' (set to 'Rs. 0') and 'Max' (set to 'Rs.50'), with an 'Apply' button below them.

Figure 117: Search Bar Button: System successfully displays "No result found" message

4.3. WHITE BOX TESTING

4.4.1. TEST 1 -TO LOGOUT FROM THE APPLICATION

TEST	One
Objective	To verify that the user can successfully log out from their application and their session is terminated securely.
Action	<ul style="list-style-type: none"> ➤ From the navigation bar, click on dashboard. ➤ Click on “Log Out” which is in navigation bar and sidebar.
Expected Result	<ul style="list-style-type: none"> ➤ The user should be logged out from the application and redirected to login page. ➤ The session must be removed from the local storage.
Actual Result	<ul style="list-style-type: none"> ➤ The user is successfully logged out from the application.
Conclusion	The test was successful.

Table 7: Black-Box Testing: Test 1- To Logout from The Application

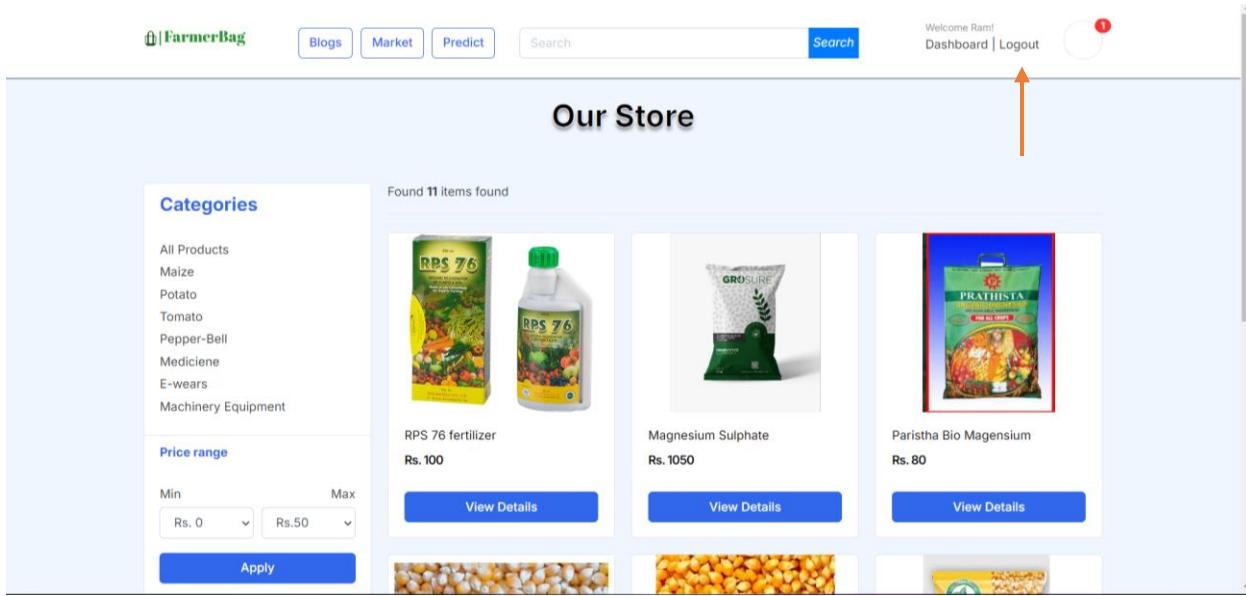


Figure 118: Screenshot of Logging out from application (1)

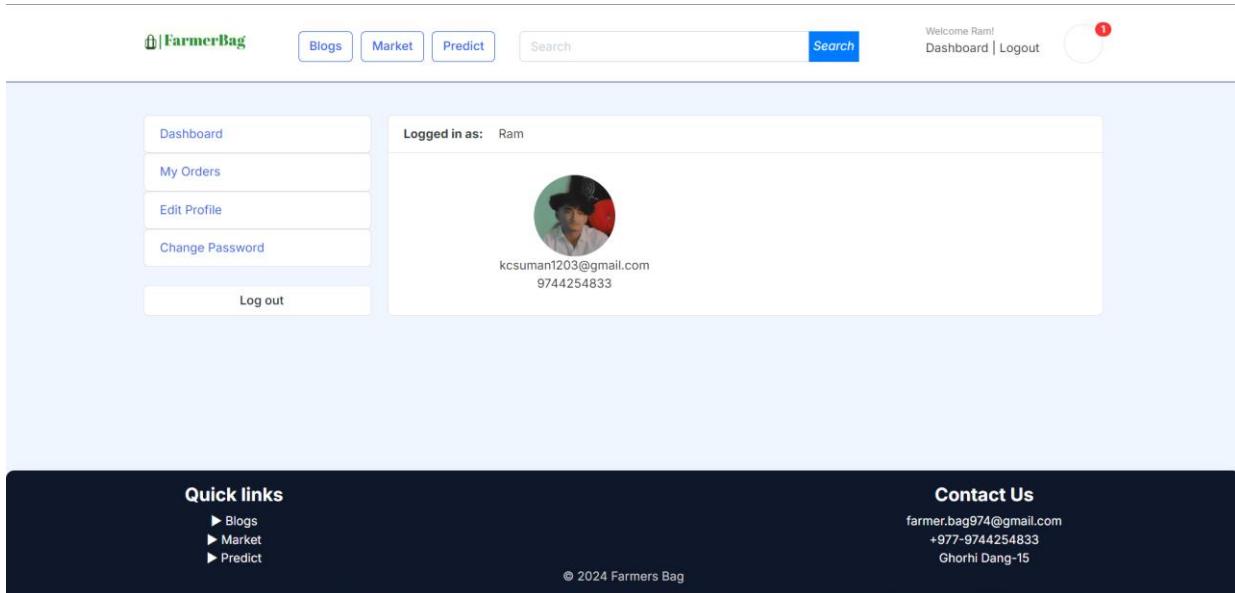


Figure 119: Screenshot of Logging out from application (2)

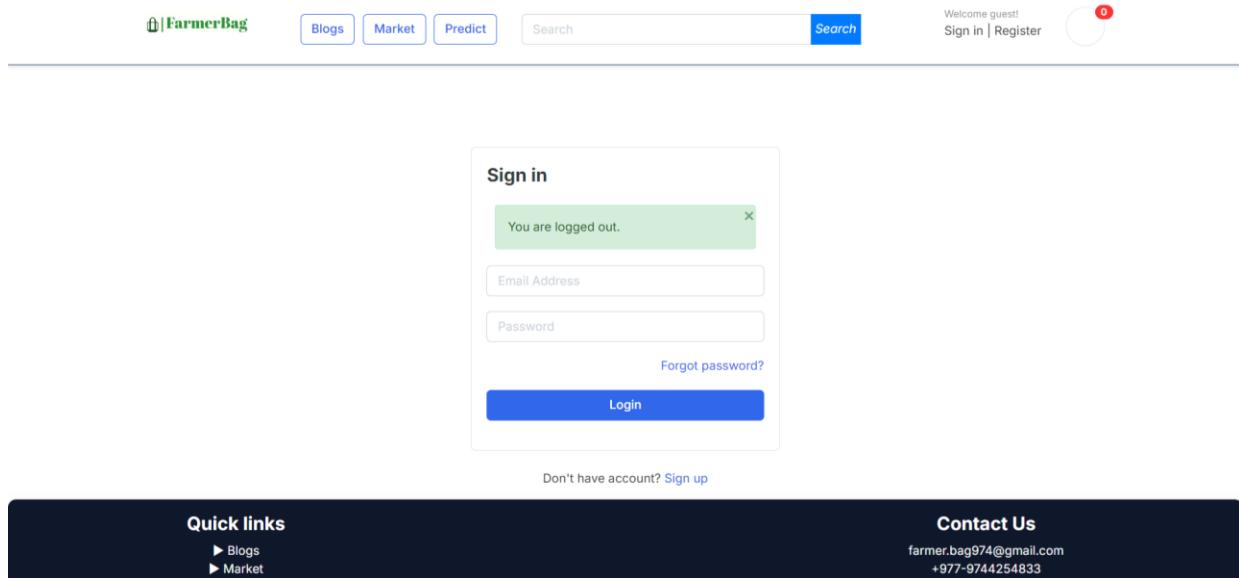


Figure 120: Screenshot of redirected page after logging out.

```

urls.py
views.py
> blog
> carts
> category
> env
> farmerbag
104
105
106 @login_required(login_url='login')
107 def logout(request):
108     auth.logout(request)
109     messages.success(request, 'You are logged out.')
110
111

```

Figure 121: Screenshot of Logout Function Code (1)

```

    </div>
</div>
{% else %}
<div class="widget-header">
    <small class="title text-muted">Welcome {{user.first_name}}!</small>
    <div>
        <a href="{% url 'dashboard' %}">Dashboard</a> <span class="dark-transp"> | </span>
        <a href="{% url 'logout' %}"> Logout</a>
    </div>
</div>
{% endif %}
<a href="{% url 'cart' %}" class="widget-header pl-3 ml-3">
    <div class="icon icon-sm rounded-circle border"><i class="fa fa-shopping-cart"></i></div>
    <span class="badge badge-pill badge-danger notify">{{cart_count}}</span>
</a>

```

Figure 122: Screenshot of Logout Function Used in Navigation Bar

```

    <li><a class="list-group-item" href="{% url 'view_users' %}>Users</a></li>
  {% else %}
    <li><a class="list-group-item" href="{% url 'dashboard' %}">Dashboard</a></li>
    <li><a class="list-group-item" {% if '/orders/' in request.path %}active{% endif %}" href="{% url 'view_orders' %}">My Orders</a></li>
    <li><a class="list-group-item" {% if '/edit_profile/' in request.path %}active{% endif %}" href="{% url 'edit_profile' %}">Edit Profile</a></li>
    <li><a class="list-group-item" {% if '/change_password/' in request.path %}active{% endif %}" href="{% url 'change_password' %}">Change Password</a>
  {% endif %}
</ul>
<br>
<a class="btn btn-light btn-block" href="{% url 'logout' %}> <i class="fa fa-power-off"></i> <span class="text">Log out</span> </a>

```

Figure 123: Screenshot of Logout Function Used in Dashboard**4.2.2. TEST 2 – TO MANAGE ORDER**

TEST	THREE
Objective	To verify logged users to view and delete their order details history.
Action	<ul style="list-style-type: none"> ➤ Navigate to the user Dashboard. ➤ Click on the “My Order” on the left side. ➤ User can view the order history and delete the order history.
Expected Result	<ul style="list-style-type: none"> ➤ The user should successfully be able to view their order history. ➤ The user should be able to delete their order history.
Actual Result	The user was able to view and delete order history.
Conclusion	The test was successful.

Table 8: White Box Testing: Manage Order

Orders

Order Date	Total Price	Shipping Address	User	Action
April 8, 2024, 7:10 a.m.	10	hello	kcsuman1203@gmail.com	Delete
April 8, 2024, 7:42 a.m.	10	hello	kcsuman1203@gmail.com	Delete
April 12, 2024, 10:07 a.m.	100	hello	kcsuman1203@gmail.com	Delete
April 14, 2024, 5:19 a.m.	100	hello	kcsuman1203@gmail.com	Delete
April 16, 2024, 5:18 a.m.	100	hello	kcsuman1203@gmail.com	Delete
April 16, 2024, 7:31 a.m.	100	hello	kcsuman1203@gmail.com	Delete

Figure 124: Manage Order: Navigate to My Order

Orders

Order Date	Total Price	Shipping Address	User	Action
April 8, 2024, 7:10 a.m.	10	hello	kcsuman1203@gmail.com	Delete
April 8, 2024, 7:42 a.m.	10	hello	kcsuman1203@gmail.com	Delete
April 12, 2024, 10:07 a.m.	100	hello	kcsuman1203@gmail.com	Delete
April 14, 2024, 5:19 a.m.	100	hello	kcsuman1203@gmail.com	Delete
April 16, 2024, 5:18 a.m.	100	hello	kcsuman1203@gmail.com	Delete
April 16, 2024, 7:31 a.m.	100	hello	kcsuman1203@gmail.com	Delete

Figure 125: Manage Order: Click on delete order.

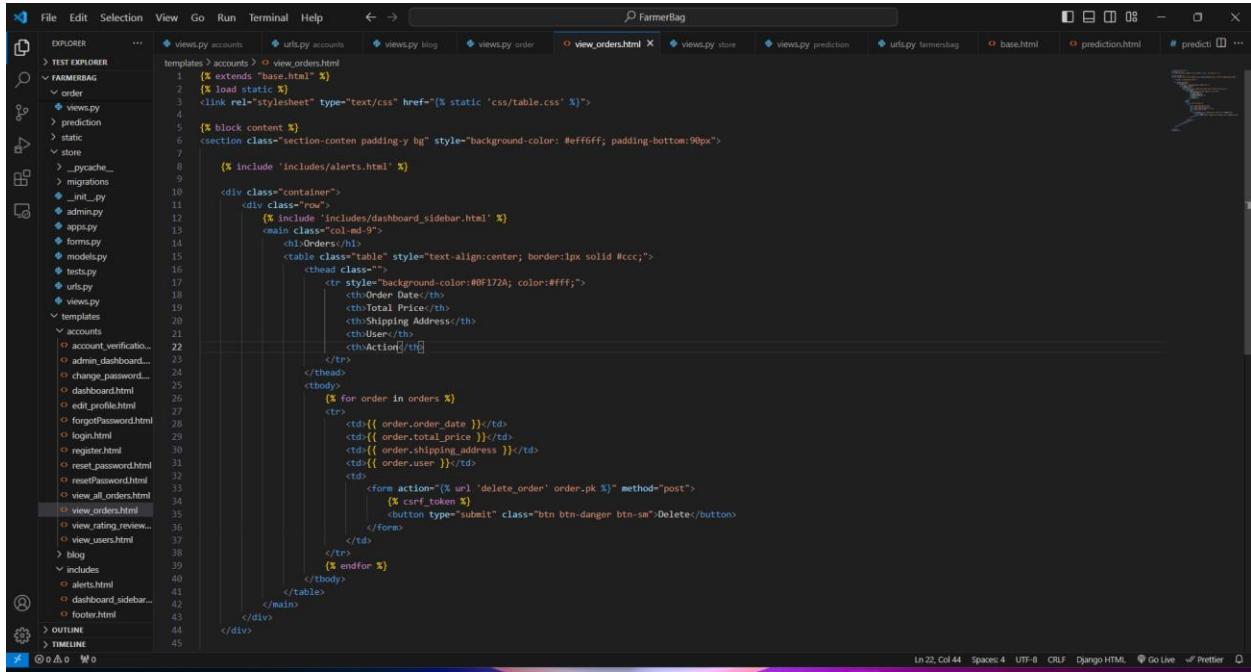
```

def view_orders(request):
    # Retrieve orders for the current user from the database
    if request.user.is_authenticated:
        orders = Order.objects.filter(user=request.user)
    else:
        orders = []

    # Pass orders to the template for rendering
    return render(request, 'accounts/view_orders.html', {'orders': orders})

def delete_order(request, order_id):
    if request.method == 'POST':
        order = get_object_or_404(Order, pk=order_id)
        # Add any additional checks here, such as verifying the user's permission to delete the order
        order.delete()
        return redirect('view_orders')
    else:
        return redirect('view_orders')

```

Figure 126: Manage Order: Code (1)


```

<!DOCTYPE html>
<html>
    <head>
        <title>FarmerBag</title>
        <link rel="stylesheet" type="text/css" href="{% static 'css/table.css' %}">
    </head>
    <body>
        <h1>Orders</h1>
        <table border="1">
            <thead>
                <tr>
                    <th>Order ID</th>
                    <th>Order Date</th>
                    <th>Total Price</th>
                    <th>Shipping Address</th>
                    <th>User</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>{{ order.order_id }}</td>
                    <td>{{ order.order_date }}</td>
                    <td>{{ order.total_price }}</td>
                    <td>{{ order.shipping_address }}</td>
                    <td>{{ order.user }}</td>
                    <td><a href="{% url 'delete_order' order.pk %}">Delete</a></td>
                </tr>
            </tbody>
        </table>
    </body>
</html>

```

Figure 127: Manage Order: Code (2)

4.2.3. TEST 3 – TO PREDICT PLANT DISEASE

TEST	Three
Objective	To verify users can predict the plant leaf disease.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Navigation bar. ➤ Click on the “Predict” button on navigation bar. ➤ System should redirect to Prediction page. ➤ Upload image and click on predict button. ➤ System should provide the disease according to the plant disease.
Expected Result	<ul style="list-style-type: none"> ➤ The system should be able to predict the disease. ➤ The system should recommend product according to the disease. ➤ The system should handle unknown class properly.
Actual Result	The system was able to predict the actual plant disease and the system was handling unknown class properly.
Conclusion	The test was successful.

Table 9: White Box Testing: Prediction & Recommendation

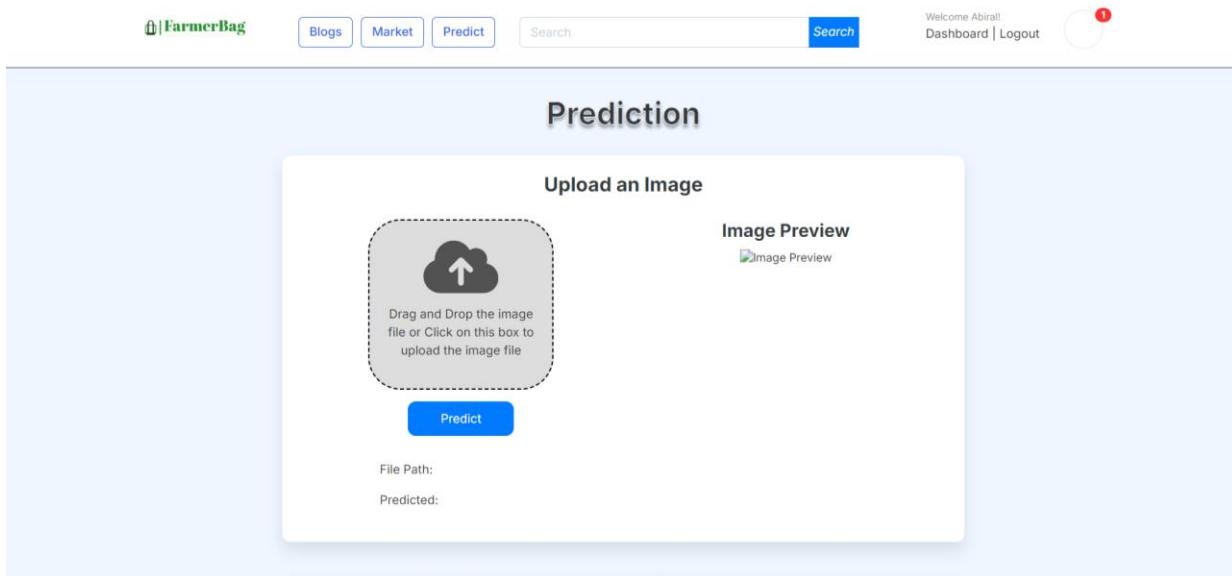


Figure 128: Prediction: Navigate to Predict Page

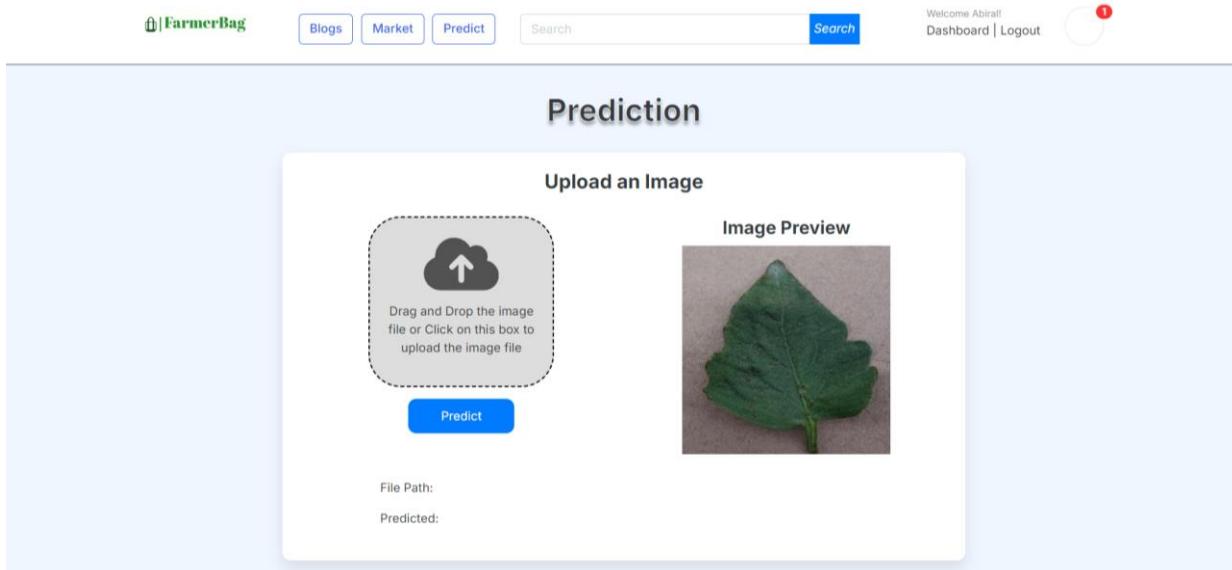


Figure 129: Prediction: Drag and Drop the leaf photo.

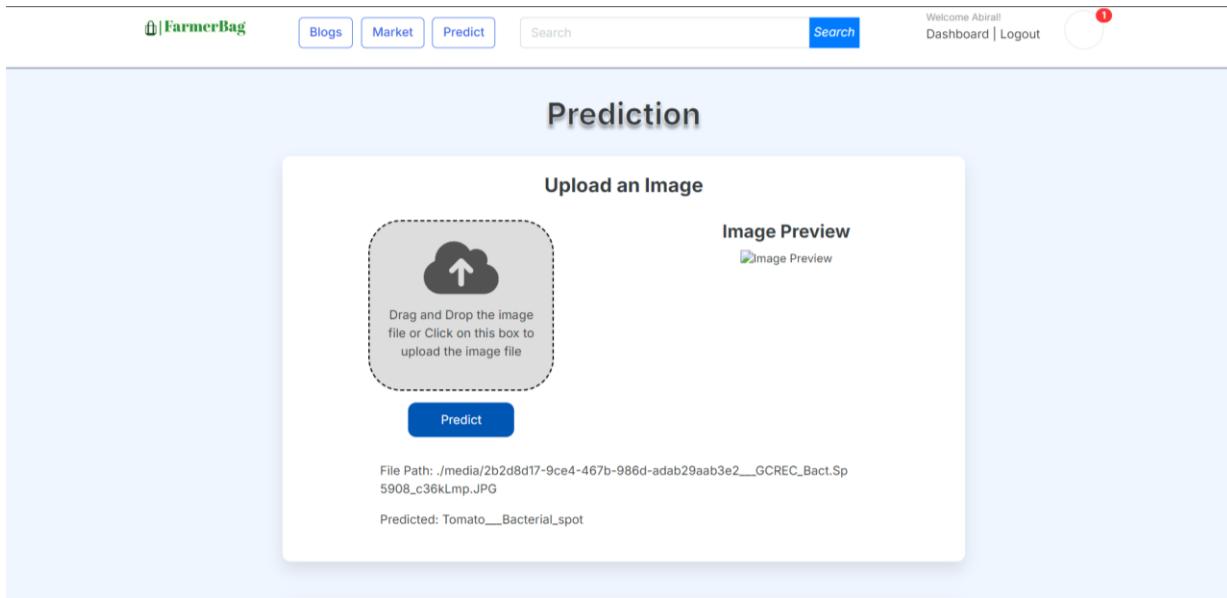


Figure 130: Prediction: Predict the ‘Tomato Bacterial spot’.

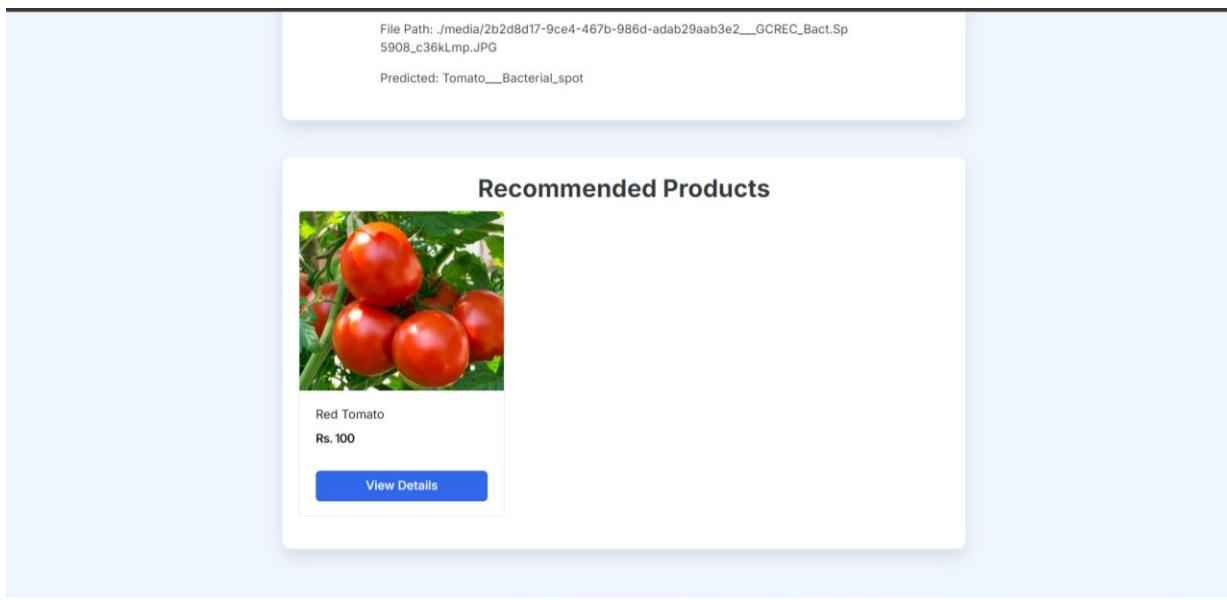


Figure 131: Prediction: Recommendation according to detected disease

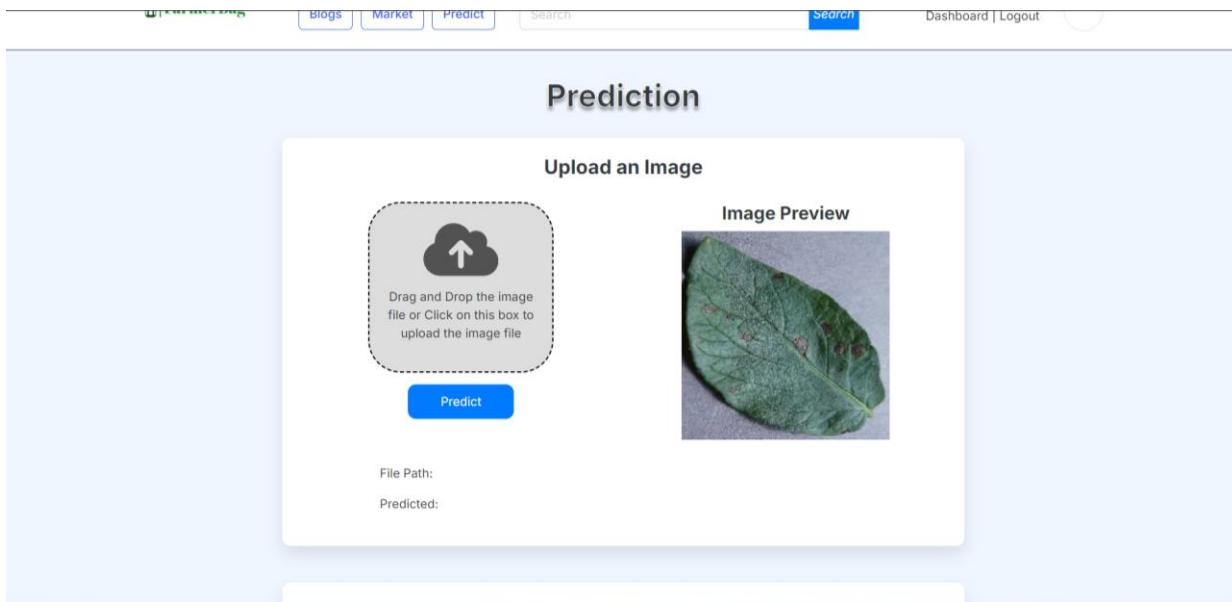


Figure 132: Prediction: Choose Potato Leaf

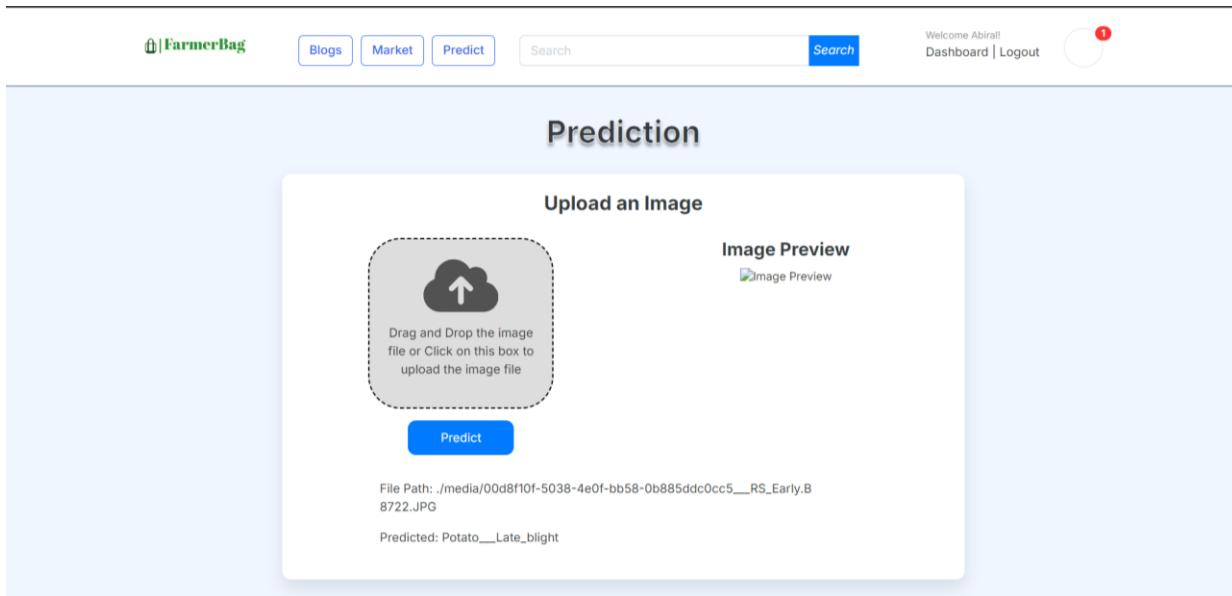


Figure 133: Prediction: Disease Detected on Leaf

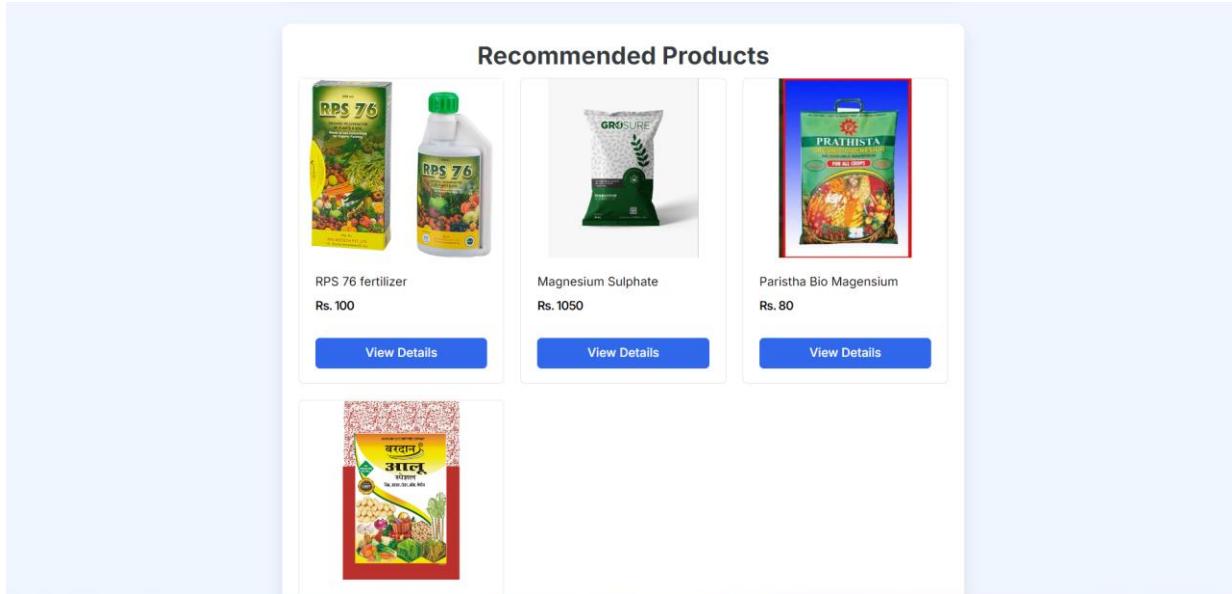


Figure 134: Prediction: Product Recommendation according to disease detected.

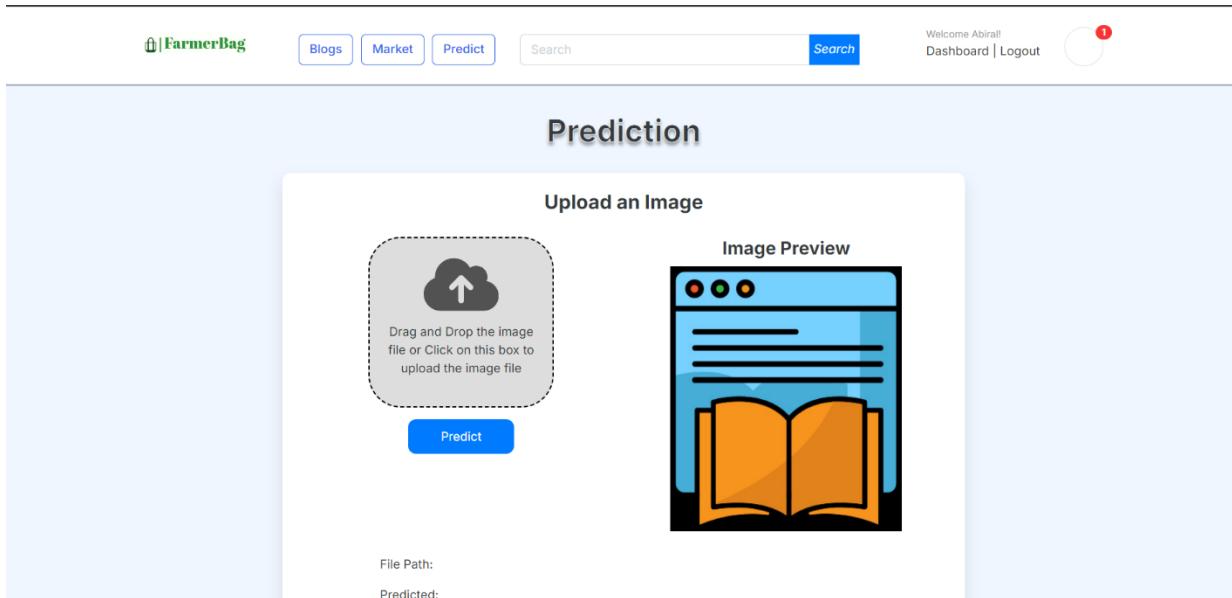


Figure 135: Prediction: Handling Unknown Class

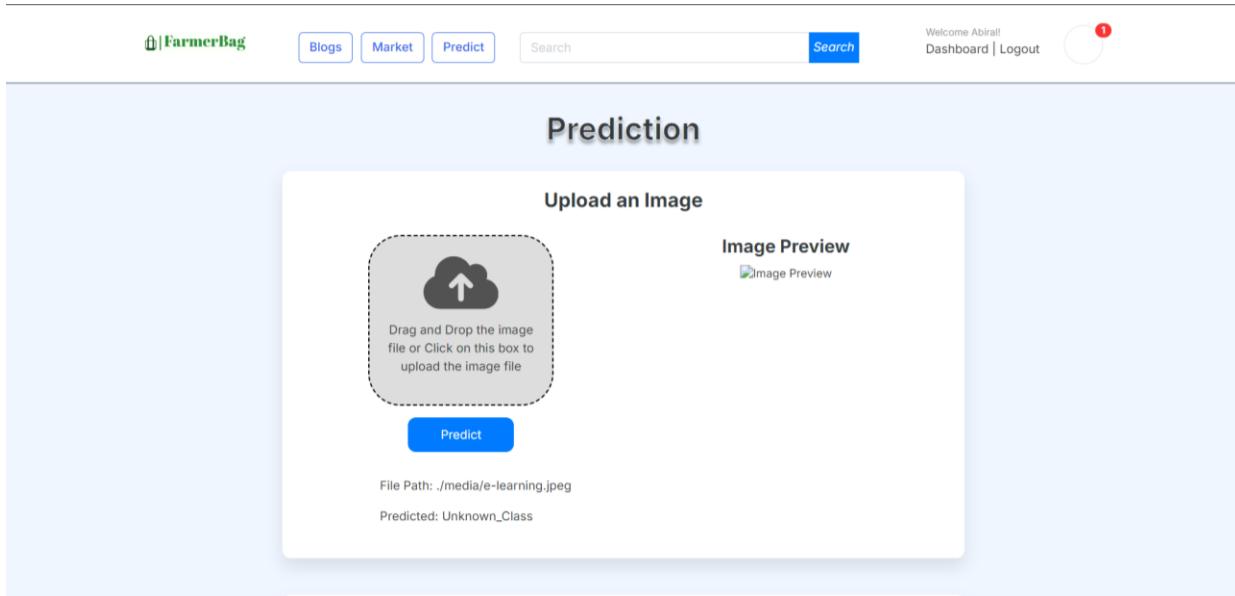


Figure 136: Prediction: Handled Unknown Class

```
def recommend_products(predicted_label):
    # Convert predicted_label index to string label
    predicted_label_str = class_labels[predicted_label]

    # Extract the main category label
    main_category = predicted_label_str.split("__")[0]

    # Query products with descriptions or titles containing the main category label
    products = Product.objects.filter(
        Q(description__icontains=main_category) |
        Q(product_name__icontains=main_category)
    )

    return products
```

Figure 137: Recommend Product Function

```

def imgPrediction(request):
    if request.method == 'POST':
        try:
            fileObj = request.FILES['filePath']
            fs = FileSystemStorage()
            filePathNames = fs.save(fileObj.name, fileObj)
            filePathName = './media/' + filePathNames

            testimage = filePathName

            img = load_image(testimage)
            predicted_label, confidence = predict(model, img)
            predictedLabel = class_labels[predicted_label]

            # Get recommended products based on the predicted label
            recommended_products = recommend_products(predicted_label=predicted_label)

            context = {
                'filePathName': filePathName,
                'predictedLabel': predictedLabel,
                'confidence': confidence,
                'recommended_products': recommended_products, # Pass recommended products to the context
            }

            return render(request, 'prediction/prediction.html', context)

        except MultiValueDictKeyError:
            # Handle the case where 'filePath' is not found in request.FILES
            error_message = "Please upload a file."
            context = {'error_message': error_message}
            return render(request, 'prediction/prediction.html', context)

    # Handle GET requests or other methods
    return HttpResponseBadRequest("Method not allowed")

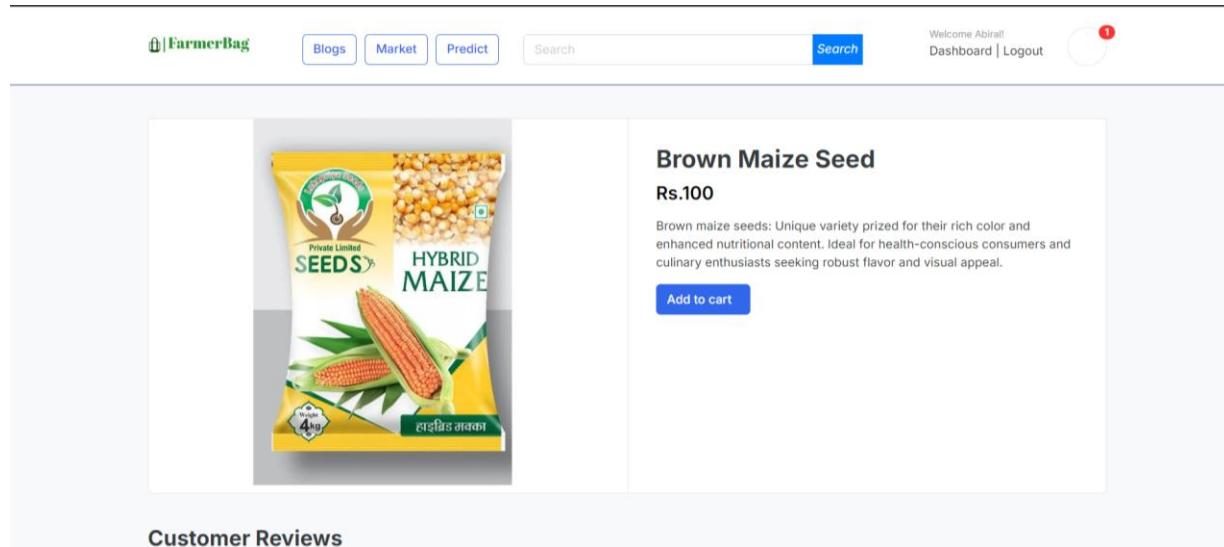
```

Figure 138: Image Prediction Function.

4.2.4. TEST 4 – TO RATING AND REVIEW PRODUCTS

TEST	Four
Objective	To verify users can rate and review the single products.
Action	<ul style="list-style-type: none"> ➤ User should click on specific product and system redirected to the product description page. ➤ At bottom user can post comment and provide ratings to the specific product.
Expected Result	The user should be able to rate and post comment on the specific product.
Actual Result	The user was able to rate and post comment to the specific product.
Conclusion	The test was successful.

Table 10: White Box Testing: Rate and Review



The screenshot shows a product page for "Brown Maize Seed" on the FarmerBag website. The page includes a product image of a yellow bag labeled "SEEDS" and "HYBRID MAIZE" with a weight of "4kg". To the right of the image, the product name "Brown Maize Seed" and price "Rs.100" are displayed. A brief description states: "Brown maize seeds: Unique variety prized for their rich color and enhanced nutritional content. Ideal for health-conscious consumers and culinary enthusiasts seeking robust flavor and visual appeal." Below the description is a blue "Add to cart" button. At the bottom of the page, there is a section titled "Customer Reviews". The top navigation bar features links for Blogs, Market, Predict, Search, Welcome Abiraj!, Dashboard, and Logout. There is also a notification icon with a '1' indicating one new message.

Figure 139: Rate Review: Product Description Page

Comment:
|

Rating:

Submit Review

Quick links

- ▶ Blogs
- ▶ Market
- ▶ Predict

Contact Us

farmer.bag974@gmail.com
+977-9744254833
Ghorhi Dang-15

© 2024 Farmers Bag

Figure 140: Rate and Review: Form

Customer Reviews

Comment:
Very good Product.....

Rating:
4 Stars

Submit Review

Quick links

- ▶ Blogs
- ▶ Market
- ▶ Predict

Contact Us

farmer.bag974@gmail.com
+977-9744254833
Ghorhi Dang-15

© 2024 Farmers Bag

Figure 141: Rate and Review: Providing Ratings

Brown Maize Seed

Rs.100

Brown maize seeds: Unique variety prized for their rich color and enhanced nutritional content. Ideal for health-conscious consumers and culinary enthusiasts seeking robust flavor and visual appeal.

Add to cart

Customer Reviews

kcsuman1203
Very good Product.....

Comment:

Figure 142: Rate and Review: Successfully Provide Rate and Review

```
def product_detail(request, category_slug, product_slug):
    try:
        single_product = Product.objects.get(category_slug=category_slug, slug=product_slug)
    except Product.DoesNotExist:
        raise Http404("Product does not exist")

    in_cart = CartItem.objects.filter(cart__cart_id=_cart_id(request), product=single_product).exists()
    ordered = False
    overall_rating = None
    if request.method == 'POST':
        form = ReviewForm(request.POST)
        if form.is_valid():
            user = request.user
            comment = form.cleaned_data['comment']
            rating = form.cleaned_data['rating']
            review = Review.objects.create(user=user, product=single_product, comment=comment, rating=rating)
            return redirect('product_detail', category_slug=category_slug, product_slug=product_slug)
    else:
        form = ReviewForm()
    if request.user.is_authenticated:
        ordered = request.user.order_set.filter(orderitems__product=single_product, payment__payment_status='done').exists()

    reviews = Review.objects.filter(product=single_product)
    if reviews.exists():
        overall_rating = reviews.aggregate(Avg('rating'))['rating__avg']

    # Retrieve or create UserProfile
    try:
        userprofile = UserProfile.objects.get(user=request.user)
    except UserProfile.DoesNotExist:
        userprofile = UserProfile(user=request.user)
        userprofile.save()
    context = [
        'single_product': single_product,
        'in_cart': in_cart,
        'form': form,
        'overall_rating': overall_rating,
        'ordered': ordered,
        'reviews': reviews,
        'userprofile': userprofile,
    ]
    return render(request, 'store/product_detail.html', context)
```

```
<section class="section-content padding-y bg">
  <div class="container">
    <div class="row">
      <div class="col-md-9">
        <header class="section-heading">
          |   <h3>Customer Reviews</h3>
        </header>
        {% for review in reviews %}
          {% if not request.user.is_admin %}
            <article class="box mb-3">
              <div class="icontext w-100">
                {% if userprofile.profile_picture %}
                  
                {% else %}
                  <!-- Provide a default profile picture or a placeholder -->
                  
                {% endif %}
                <div class="text">
                  <span class="date text-muted float-md-right">{{ review.created_at }}</span>
                  <h6 class="mb-1">{{ review.user.username }}</h6>
                </div>
              </div>
              <div class="mt-3">
                <p>{{ review.comment }}</p>
              </div>
            </article>
          {% endif %}
        {% endfor %}

        <!-- Review Form -->
        {% if not request.user.is_admin %}
          <div class="mt-5">
            <form method="post" action="{% url 'product_detail' category_slug=single_product.category.slug product_slug=single_product.slug %}">
              {% csrf_token %}
              {{ form.as_p }}
              <button type="submit" class="btn btn-primary">Submit Review</button>
            </form>
          </div>
        {% endif %}
        <!-- End Review Form -->
      </div>
    </div>
  </div>
</section>
```

4.2.5. TEST 5 – TO DELETE PROUDCT

TEST	Five
Objective	To ensure that admin can successfully delete product form the system.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Products” and system displays product. ➤ Click on “Delete” icon button. ➤ Verify that the product is removed from the system.
Expected Result	The associated products should be deleted from the system.
Actual Result	The associated product was deleted from the system.
Conclusion	The test was successful.

Table 11: White Box Testing: To Delete Product

Our Store

Found 3 items found

Category	Image	Name	Price	Action
All Products		Grade Maize	Rs. 50	Delete UpdateProduct View Details
		Yellow Maize	Rs. 50	Delete UpdateProduct View Details
		Brown Maize Seed	Rs. 100	Delete UpdateProduct View Details

Figure 143: Delete Product: Navigate to Admin Product and Select ‘delete’ Product

Our Store

Found 2 items found

Category	Image	Name	Price	Action
All Products		Yellow Maize	Rs. 50	Delete UpdateProduct View Details
		Brown Maize Seed	Rs. 100	Delete UpdateProduct View Details

Figure 144: Delete Product: Successfully deleted product

	id	product name	slug	description	price	images
1	2	RPS 76 fertilizer	rps-76-fertilizer	RPS 76 is an organic liquid fertilizer made from humic acids sourced from Leonardite. Chemical-free and ...	100	photos/products/rps-76-bio-fertilizers_exFIPwe.webp
2	3	Magnesium Sulphate	magnesium-sulphate	Magnesium Sulphate is a vital nutrient for plant growth, particularly beneficial for crops like maize, corn, and ...	1050	photos/products/magnesium-sulphate-mag-sul-fertiliz...
3	4	Paristha Bio Magensium	paristha-bio-magensium	Paristha Bio Magensium is a vital nutrient for plant growth, particularly beneficial for crops like maize, corn, an...	80	photos/products/Paristha_bio_magensium_xOzafZS.p...
4	5	Yellow Maize	yellow-maize	Yellow maize, also known as corn, is a versatile crop used for various purposes, including making popcorn. This...	50	photos/products/yellow-maize_bHMJMaL.webp
5	6	Brown Maize Seed	brown-maize-seed	Brown maize seeds: Unique variety prized for their rich color and enhanced nutritional content. Ideal for health...	100	photos/products/Brown_Hybrid_Maize_Seed_FWPNES...
6	7	Agricultural Boots	agricultural-boots	Agricultural boots are worn by farmers in the rainy season to protect their legs from moisture, mud, and other ...	1000	photos/products/agricultural_boots_OGKDR2v.jpg
7	8	Agricultural Gloves	agricultural-gloves	Agricultural gloves are worn by farmers in the rainy season to protect their hands from moisture, mud, and ...	99	photos/products/agriculture_gloves_RS8bYEx.webp
8	9	Wardan Potato Crops	wardan-potato-crops	At Wardan, we're dedicated to cultivating top-quality potatoes through innovation and sustainability. From se...	100	photos/products/Potato_UWm6Xj.jpg
9	10	Red Tomato	red-tomato	Tomatio...	100	photos/products/images.jpg
10	12	Pepper Bell	pepper-bell	Pepper-bell...		

Figure 145: Delete Product: Product Not in database.

```
@user_passes_test(is_admin)
def delete_product(request, pk):
    eachProduct = Product.objects.get(id=pk)
    eachProduct.delete()
    return redirect('store')
```

Figure 146: Delete Product: Code (1)

```
<a href="{{ product.get_url }}" class="btn btn-block btn-primary">View Details</a>
{% else %}
    <button class="btn btn-block btn-danger" onclick="window.location.href='{{ url 'delete_product' product.pk }}'">Delete</button>
    <button class="btn btn-block btn-success" onclick="window.location.href='{{ url 'update_product' product.pk }}'">UpdateProduct</button>
    <a href="{{ product.get_url }}" class="btn btn-block btn-primary">View Details</a>
{% endif %}
</figcaption>
```

Figure 147: Delete Product: Code (2)

4.2.6. TEST 6 – TO ADD TO CART

TEST	SEVEN
Objective	To verify that the user can successfully add the product to cart within the application.
Action	<ul style="list-style-type: none"> ➤ Navigate to Product Description Page ➤ Click on “Add to cart” button. ➤ Check the Cart to confirm that the newly added product is in cart. ➤ Display “Empty Cart” If there are no product at cart.
Expected Result	<ul style="list-style-type: none"> ➤ User should be able to add product in cart. ➤ User should be able to add multiple products in cart.
Actual Result	<ul style="list-style-type: none"> ➤ Users can add product in cart. ➤ User can add multiple products in cart.
Conclusion	The test was successful.

Table 12: White Box Testing: Add To Cart

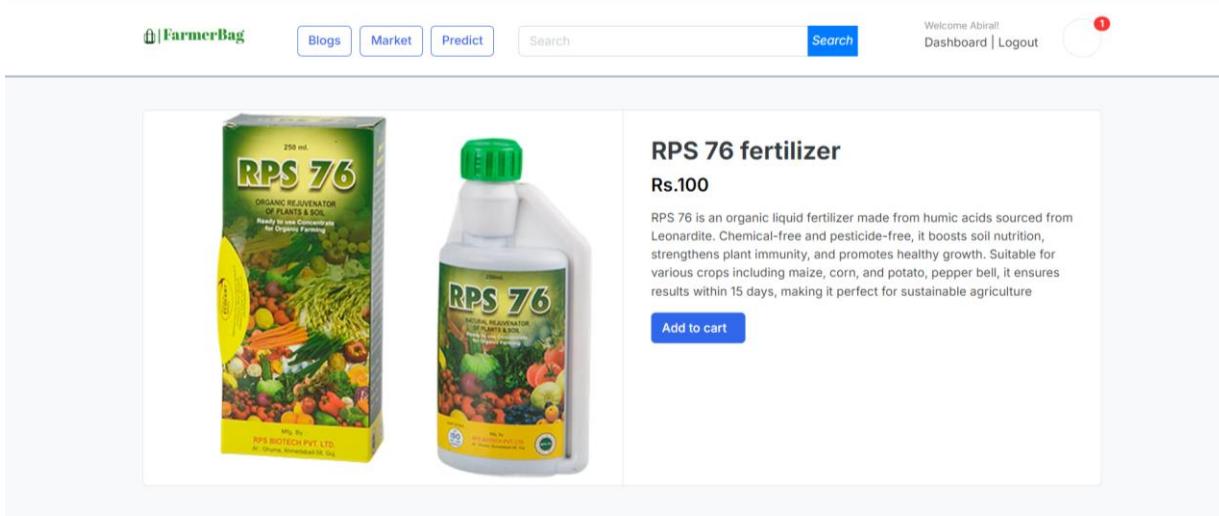


Figure 148: Add to Cart: Navigate to Product Description and click on "Add to Cart".

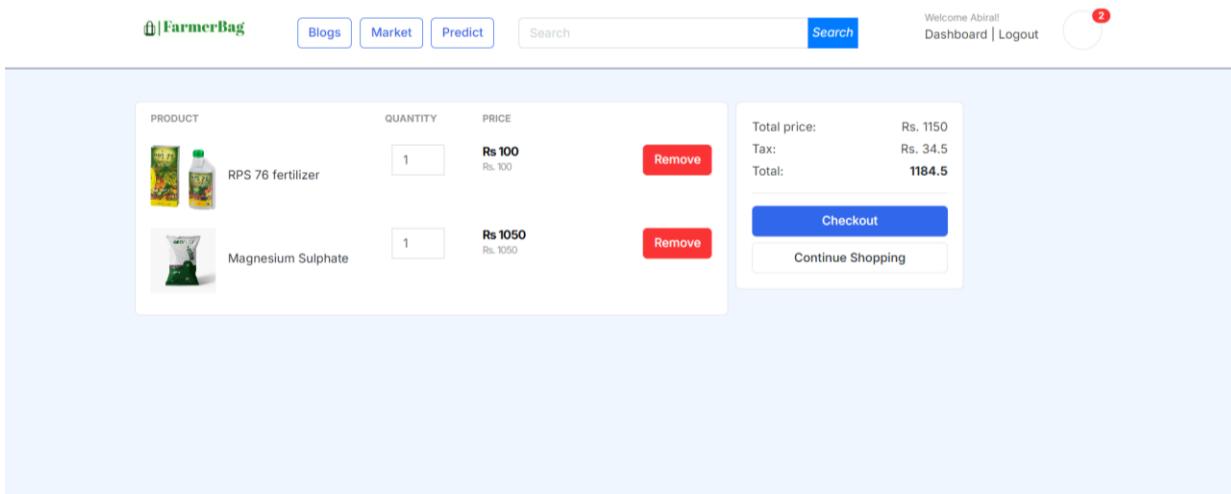


Figure 149: Add to Cart: Cart Added Successfully

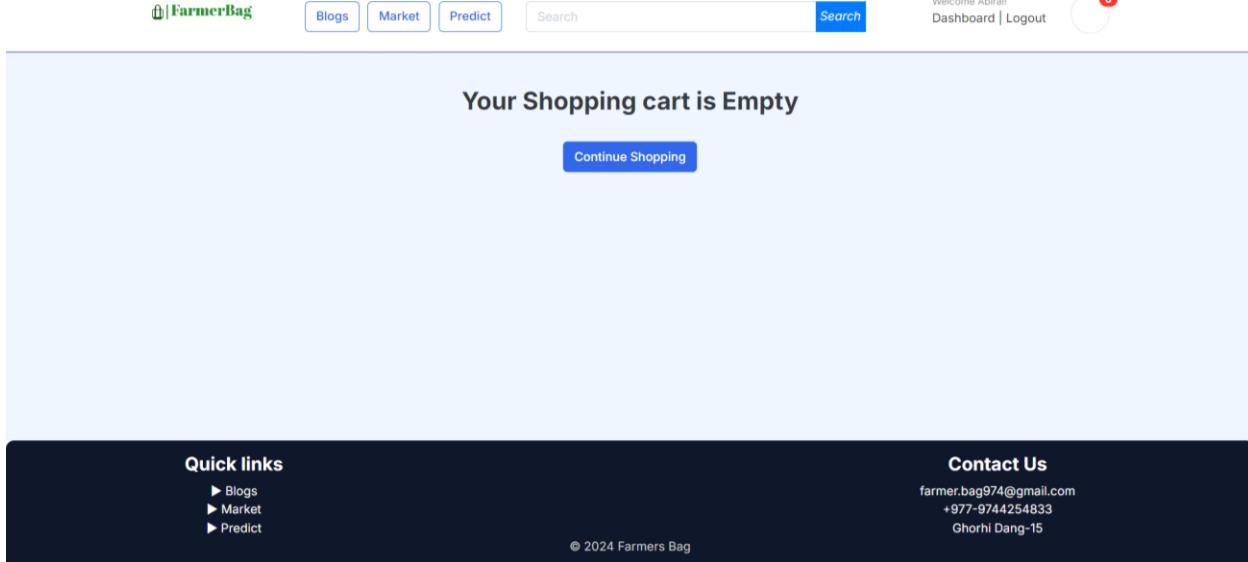


Figure 150: Add to Cart: IF single product in cart

```
def add_cart(request, product_id):
    product = Product.objects.get(id=product_id)
    try:
        # Get the cart associated with the session
        cart = Cart.objects.get(cart_id=_cart_id(request))
    except Cart.DoesNotExist:
        # If the cart doesn't exist, create a new one
        cart = Cart.objects.create(cart_id=_cart_id(request))

    # Save the cart to the database
    cart.save()

    #check if the product already exists in the cart
    is_cart_item_exists = CartItem.objects.filter(product=product, cart=cart).exists()

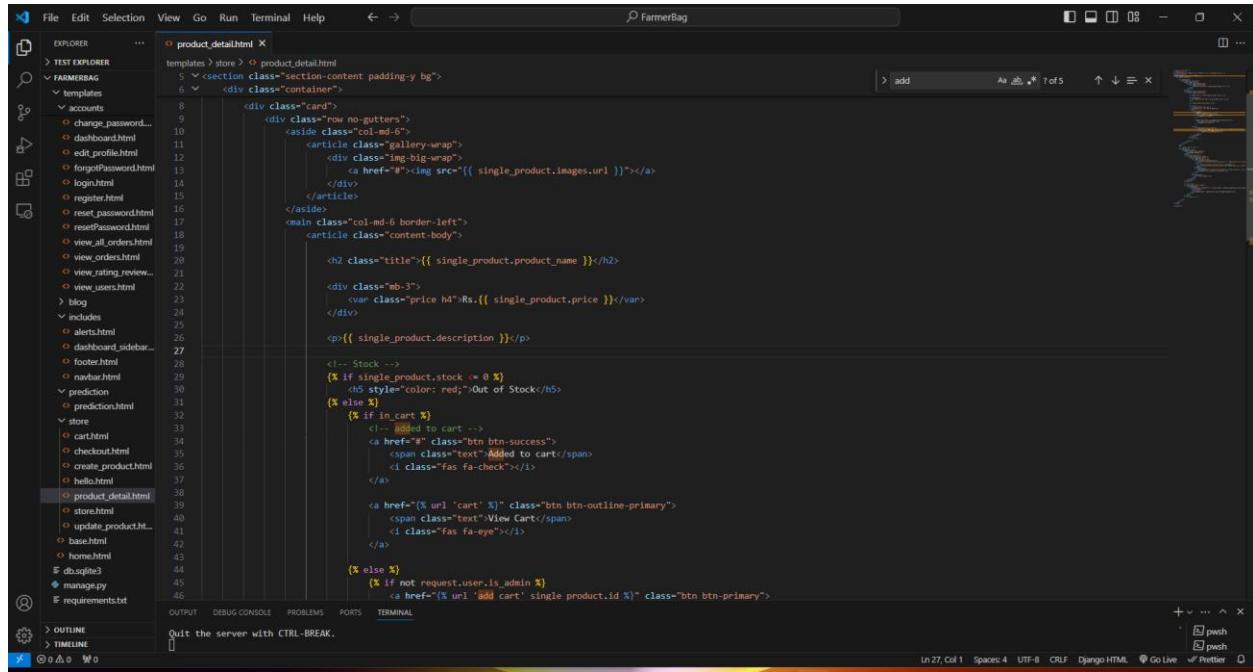
    if is_cart_item_exists:
        cart_item = CartItem.objects.filter(product=product, cart=cart)
        cart_item.save()

    else:
        # If the product does not exist in the cart, create a new cart item
        cart_item = CartItem.objects.create(
            product=product,
            quantity=1,
            cart=cart,
        )

        if request.user.is_authenticated:
            cart_item.user = request.user
            cart_item.save()

    # Redirect to the cart view after adding the item
    return redirect('cart')
```

Figure 151: Add to Cart: Code (1)



```

<div class="row no-gutters">
    <aside class="col-md-6">
        <article class="gallery-wrap">
            <div class="img-big-wrap">
                <a href="#"></a>
            </div>
        </article>
    </aside>
    <main class="col-md-6 border-left">
        <article class="content-body">
            <h2 class="title">{{ single_product.product_name }}</h2>
            <div class="mb-3">
                <var class="price h4">Rs.{{ single_product.price }}</var>
            </div>
            <p>{{ single_product.description }}</p>
            <!-- Stock -->
            {% if single_product.stock <= 0 %}
                <h5 style="color: red;">Out of Stock</h5>
            {% else %}
                {% if in_cart %}<br/>
                    <!-- added to cart -->
                    <a href="#" class="btn btn-success">
                        <span class="text">Added to Cart</span>
                        <i class="fas fa-check">/i</i>
                    </a>
                {% else %}
                    {% if not request.user.is_admin %}
                        <a href="{% url 'add_cart' single_product.id %}" class="btn btn-primary">
                            <span class="text">View Cart</span>
                            <i class="fas fa-eye">/i</i>
                        </a>
                    {% else %}
                        <a href="#" class="btn btn-primary">
                            <span class="text">Add to Cart</span>
                            <i class="fas fa-plus">/i</i>
                        </a>
                    {% endif %}
                {% endif %}
            {% endif %}
        </article>
    </main>
</div>

```

Figure 152: Add to Cart: Code (2)

4.2.7. TEST 7 -TO VIEW ADMIN DASHBOARD

TEST	EIGHT
Objective	To ensure that the user can view the user dashboard.
Action	<ul style="list-style-type: none"> ➤ Navigate to the admin dashboard. ➤ Admin can view the different charts and KPIs.
Expected Result	<ul style="list-style-type: none"> ➤ The admin should view the reports in the dashboard.
Actual Result	The admin successfully views the reports in the dashboard,
Conclusion	The test was successful.

Table 13: White Box Testing: Admin Dashboard

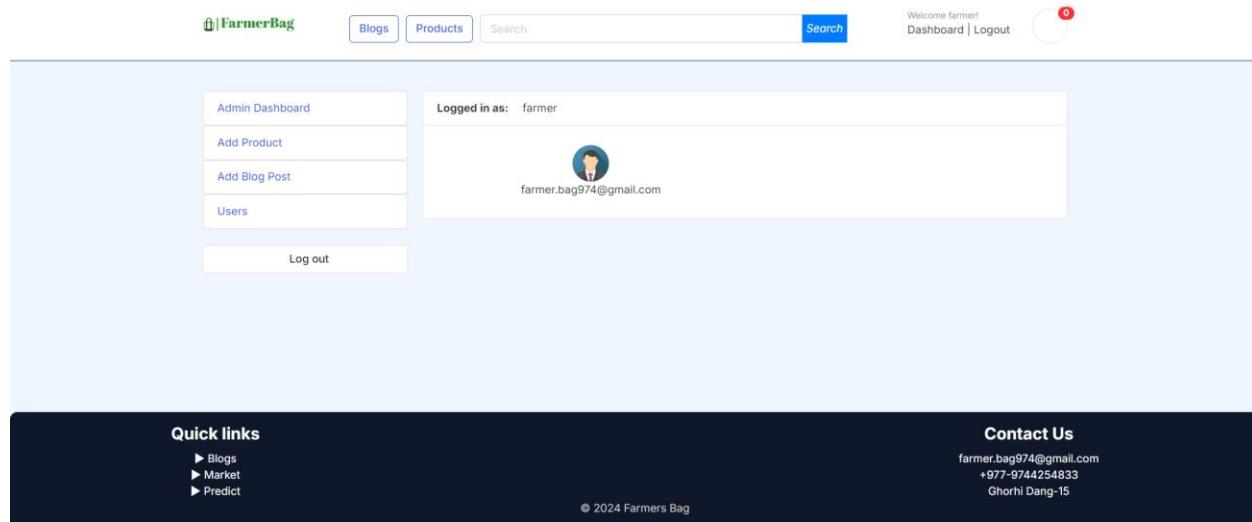


Figure 153: Admin Dashboard: Navigate admin to dashboard.

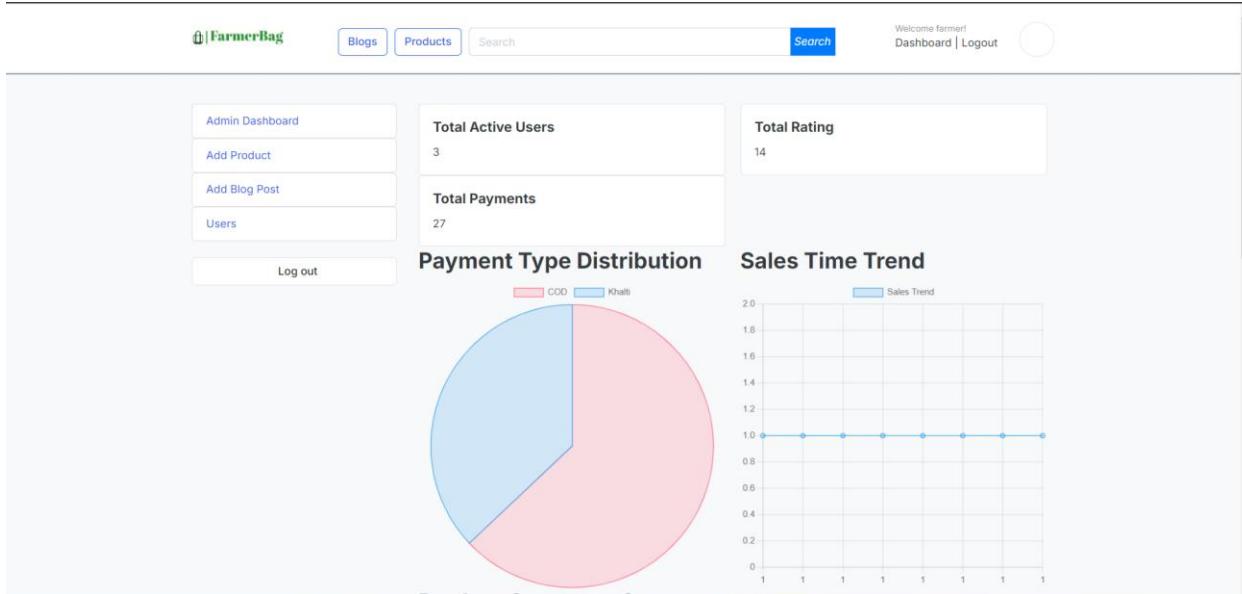


Figure 154: Admin Dashboard: Chart and KPIs (1)

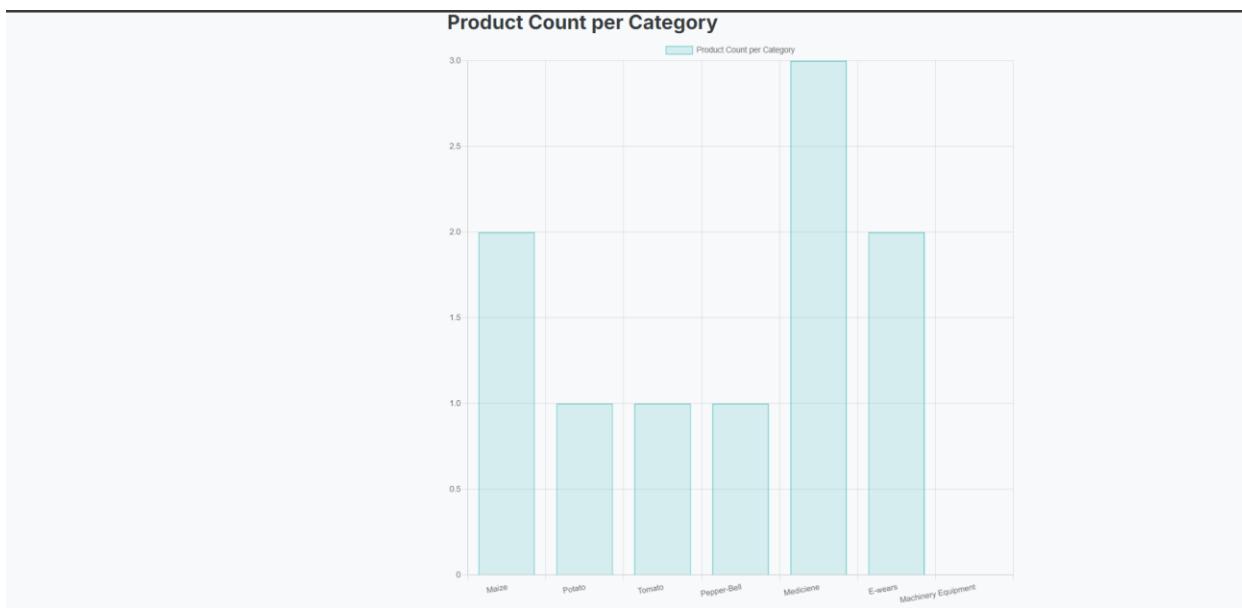


Figure 155: Admin Dashboard: Chart (1)

4.2.8. TEST 8 – TO REGISTER USER TO APPLICATION

TEST	Eight
Objective	To verify that user can successfully register to the application using valid credentials.
Action	<ul style="list-style-type: none"> ➤ Click on the “Register” button in the Home Page. ➤ Fill the form with the required details. ➤ Click on the “Register” button. ➤ Check the Gmail inbox for the activation link and click on it. ➤ Click on the “Verify your account” link on email which redirect to login page.
Expected Result	<ul style="list-style-type: none"> ➤ The user should be successfully registered and redirect to login page. ➤ A confirmation email should be sent to their email address for account verification which contain account activation link. ➤ After clicking on activation link user should be redirected to the login page with success message.
Actual Result	The user account was successfully created, and a confirmation email was sent to the user with activation link. User was redirected to login page after clicking on activation link.
Conclusion	The test was successful.

Table 14: White Box Testing: To register User to application.

The screenshot shows the registration form for the FarmerBag system. The form fields are filled as follows:

First Name	Last Name
SUMAN	KC
Email	Phone Number
kcsuman1203@gmail.com	9744254833
Password	Confirm Password
*****	*****

Registration Form

[Register](#)

Have an account? [Login](#)

Figure 156: Screenshot of registering user to system (1)

The screenshot shows the registration form for the FarmerBag system. The form fields are filled as follows:

First Name	Last Name
SUMAN	KC
Email	Phone Number
kcsuman1203@gmail.com	9744254833
Password	Confirm Password
Enter Password	Confirm Password

Registration Form

• Account with this Email already exists.

[Register](#)

Have an account? [Login](#)

Figure 157: Screenshot of error while registering user to system with same account.

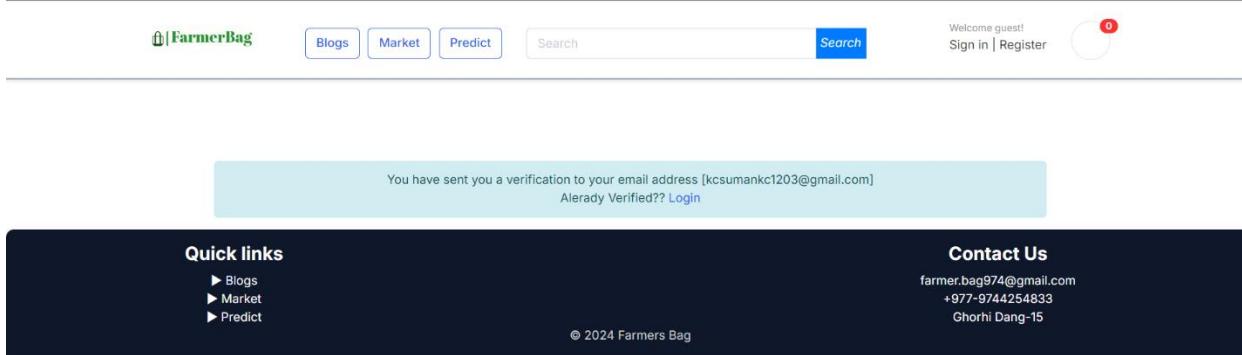


Figure 158: Screenshot of activation link after registering account.

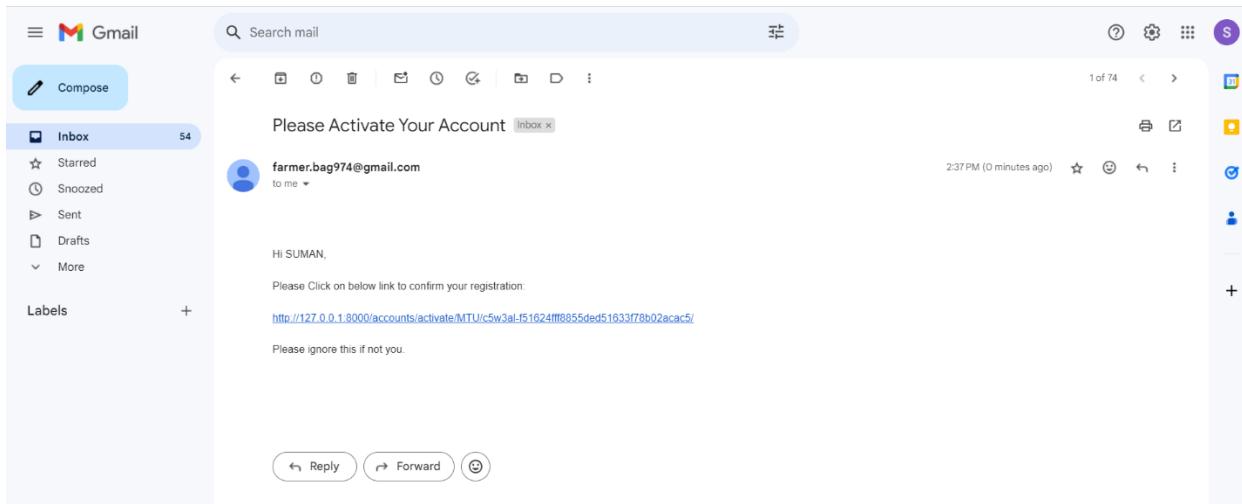


Figure 159: Screenshot of Registering User: Activation Link

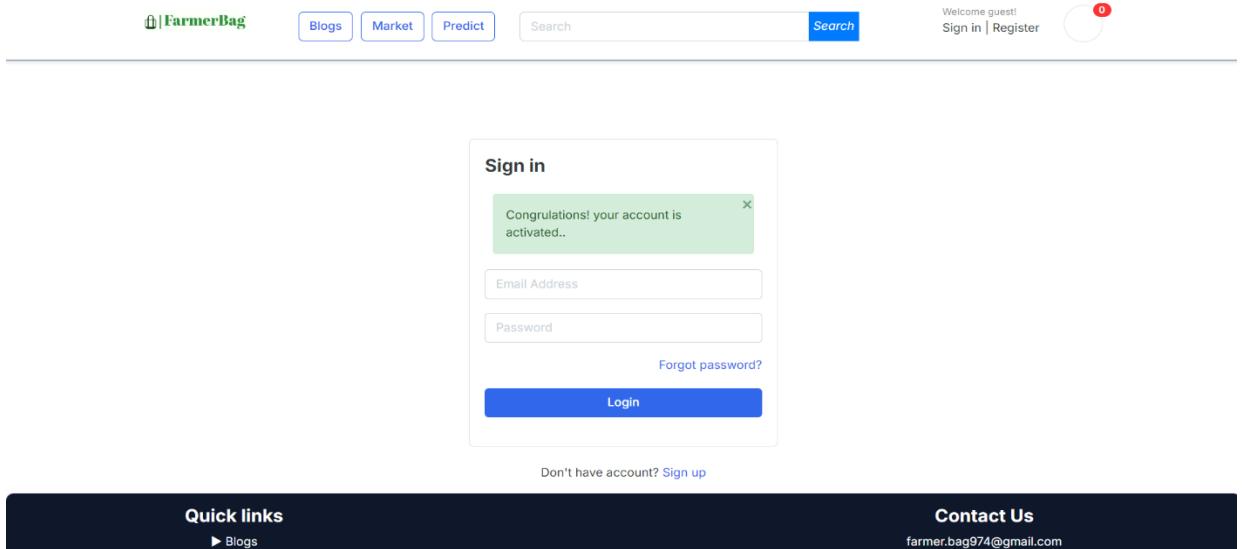


Figure 160: Screenshot of registering user: Success Message

```

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            first_name = form.cleaned_data['first_name']
            last_name = form.cleaned_data['last_name']
            phone_number = form.cleaned_data['phone_number']
            email = form.cleaned_data['email']
            password = form.cleaned_data['password']
            username = email.split("@")[0]

            # Creating user object
            user = Account.objects.create_user(first_name=first_name, last_name=last_name, email=email, username=username, password=password)
            user.phone_number = phone_number
            user.save()

            # USER ACTIVATION

            current_site = get_current_site(request)
            domain = current_site.domain
            mail_subject = 'Please Activate Your Account'
            message = render_to_string('accounts/account_verification_email.html', {
                'user': user,
                'domain': domain,
                'uid': urlsafe_base64_encode(force_bytes(user.pk)),
                'token': default_token_generator.make_token(user),
            })

            to_email = email
            send_email = EmailMessage(mail_subject, message, to=[to_email])
            send_email.send()

            #messages.success(request, 'Thank You for registering with us...')
            return redirect('/accounts/login/?command=verification&email=' + email)
        else:
            form = RegistrationForm()
            context = {'form': form}
            return render(request, 'accounts/register.html', context)
    
```

Figure 161: Screenshot of registering user: Code (1)

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}">
    </head>
    <body>
        <div class="custom-registration-container" style="background-color: #eff6ff;">
            <div class="custom-main-containner">
                <div class="custom-registration-content" style="background-color: #fff;">
                    <div class="custom-registration-form" style="background-color: #fff;">
                        <form action="{% url 'register' %}" method="POST">
                            <div class="form-group">
                                <label for="first-name">First Name</label>
                                <input type="text" name="first_name" value="{{ form.first_name }}"/>
                            </div>
                            <div class="form-group">
                                <label for="last-name">Last Name</label>
                                <input type="text" name="last_name" value="{{ form.last_name }}"/>
                            </div>
                            <div class="form-group">
                                <label for="email">Email</label>
                                <input type="text" name="email" value="{{ form.email }}"/>
                            </div>
                            <div class="form-group">
                                <label for="phone-number">Phone Number</label>
                                <input type="text" name="phone_number" value="{{ form.phone_number }}"/>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

Figure 162: Screenshot of registering user: Code (2)

4.2.9. TEST 9 – TO LOGIN USER TO THE APPLICATION

TEST	Two
Objective	To verify that registered user can log in to the application using valid credentials.
Action	<ul style="list-style-type: none"> ➤ Click on the “Login” button in the Home Page. ➤ Provide email and password in the form ➤ Click on the “Login” button. ➤ The system redirects to the dashboard.
Expected Result	<ul style="list-style-type: none"> ➤ The user should be successfully logged in and redirected to the dashboard.
Actual Result	The user is successfully logged in and redirected to the dashboard page.
Conclusion	The test was successful.

Table 15: White Box Testing: To Login User to the Application

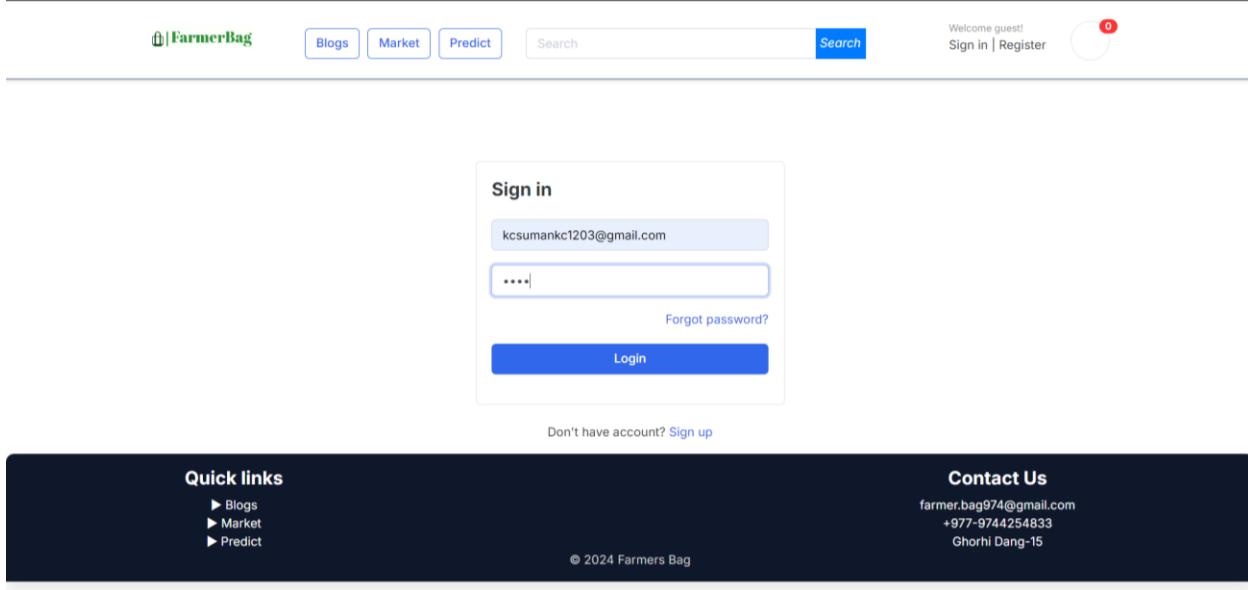


Figure 163: Screenshot of Login: Registered User

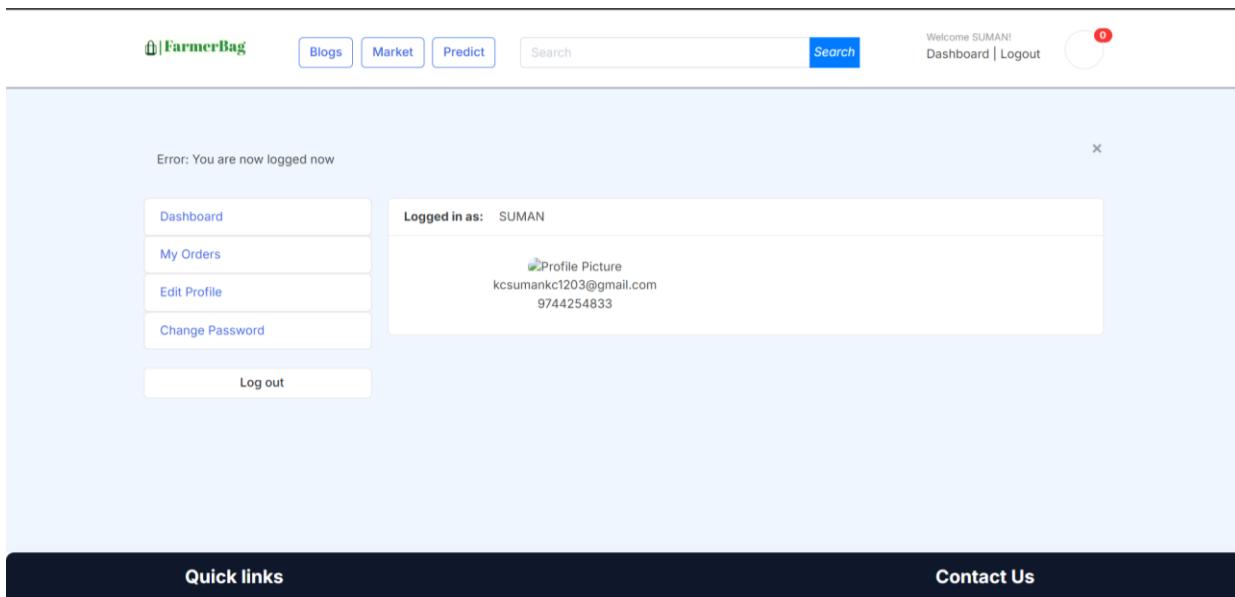


Figure 164: Screenshot of Login: After Login Dashboard

```

def login(request):
    if request.method == "POST":
        email = request.POST['email']
        password = request.POST['password']
        user = auth.authenticate(email = email, password=password)

        if user is not None:
            try:
                cart = Cart.objects.get(cart_id = _cart_id(request))
                print(cart)
                is_cart_item_exists = CartItem.objects.filter(cart=cart).exists()
                if is_cart_item_exists:
                    cart_item = CartItem.objects.filter(cart=cart)
                    print(cart_item)
                    for item in cart_item:
                        item.user = user
                        item.save()

            except:
                pass
            auth.login(request, user)
            messages.error(request, 'You are now logged now')
            return redirect('dashboard')
        else:
            messages.error(request, "Invalid login credentials")
            return redirect('login')
    return render (request, 'accounts/login.html')

```

Figure 165: Screenshot of Login: Code (1).

```

{% extends "base.html" %}
{% load static %}

{% block content %}
<link rel="stylesheet" type="text/css" href="{% static 'css/forms.css' %}">

{% if request.GET.command == 'verification' %}

<div class="container mx-auto alert alert-info text-center" role = 'alert' style="max-width: 1024px; margin-top:100px;">
    You have sent you a verification to your email address [{{request.GET.email}}]
    <br>
    Alerady Verified?? <a href = "{% url 'login'%}"> Login </a>
</div>
    
```

{% else %}

Figure 166: Screenshot of Login: Code (2)

```
{% else %}

<div class="card mx-auto" style="max-width: 380px; margin-top:100px;">

    <div class="card-body">

        <h4 class="card-title mb-4">Sign in</h4>
        {% include 'includes/alerts.html' %}

        <form action ="{% url 'login' %}" method="POST">
            {% csrf_token %}
            <div class="form-group">
                <input type="email" class="form-control" placeholder="Email Address" name = 'email'>
            </div>
            <div class="form-group">
                <input type="password" class="form-control" placeholder="Password" name = 'password'>
            </div>

            <div class="form-group">
                <a href="{% url 'forgotPassword' %}" class="float-right">Forgot password?</a>
            </div>
            <div class="form-group">
                <button type="submit" class="btn btn-primary btn-block"> Login </button>
            </div>
        </form>
    </div>
</div>
<p class="text-center mt-4">Don't have account? <a href="{% url 'register'%}">Sign up</a></p>
{% endif %}

{% endblock %}
```

Figure 167: Screenshot of Login: Code (3).

4.2.10. TEST 10 – TO RESET PASSWORD

TEST	Three
Objective	To ensure that the user can reset their password if they forget it.
Action	<ul style="list-style-type: none"> ➤ Click on the “Forget Password” link on the login form. ➤ Enter the email addressed which was provided while registering the account. ➤ Click on “submit” button ➤ Check the inbox in Gmail for reset password link and click on that which redirect to the reset password form. ➤ Enter new password and confirm it. ➤ Click on “Reset” button.
Expected Result	<ul style="list-style-type: none"> ➤ A reset password should be sent to the user’s email containing a link to reset password. ➤ The link should redirect to Reset password page. ➤ After successfully resetting password, the user should be redirected to the login page
Actual Result	The password reset email is successfully sent to the user’s email address containing link. The password is changed, and the user id redirected to login page
Conclusion	The test was successful.

Table 16: White Box Testing: To Reset Password

The screenshot shows the login page of the FarmerBag website. At the top, there is a navigation bar with a logo, 'Blogs', 'Market', 'Predict', a search bar, and a 'Search' button. To the right of the search bar are links for 'Welcome guest!', 'Sign in | Register', and a notification badge with the number '0'. Below the navigation bar is a 'Sign in' form with fields for 'Email Address' and 'Password', a 'Forgot password?' link, and a blue 'Login' button. Below the form, a message says 'Don't have account? [Sign up](#)'. At the bottom of the page is a dark footer bar containing 'Quick links' (with options for Blogs, Market, Predict), 'Contact Us' (with email, phone number, and address), and the copyright notice '© 2024 Farmers Bag'.

Figure 168: Reset password: Login Form

This screenshot shows the forgot password page of the FarmerBag website. It features a similar header and footer to Figure 168. The main content is a 'Forgot Password' form with a single input field for 'Email Address', a 'Forgot password? Login' link, and a blue 'Submit' button. Below the form, a message says 'Don't have account? [Sign up](#)'. The footer contains 'Quick links' and 'Contact Us' information.

Figure 169: Reset Password: Forgot Password Form

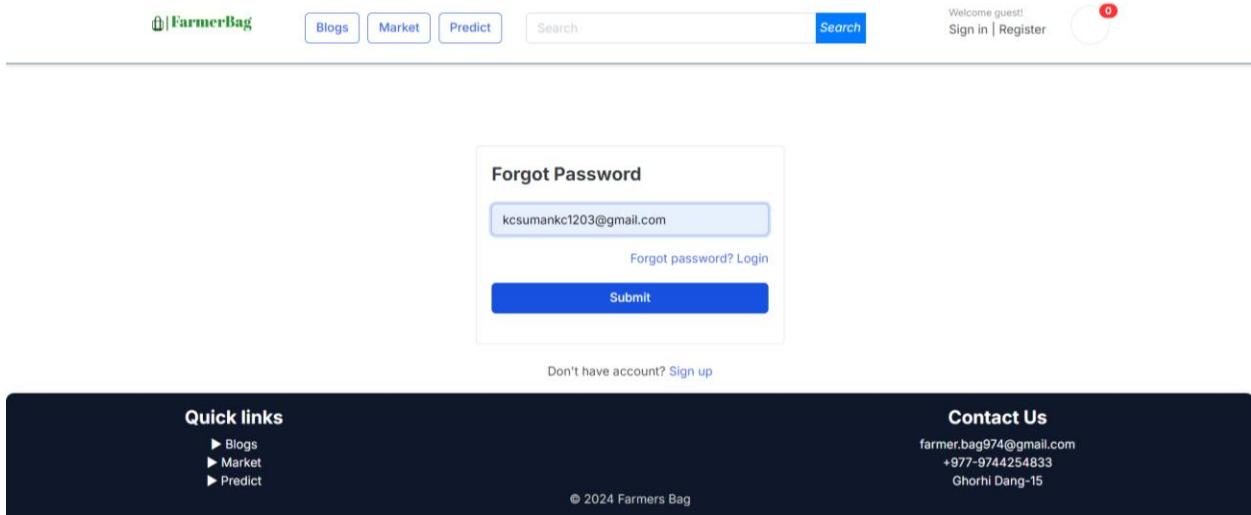


Figure 170: Reset password: Enter email address.

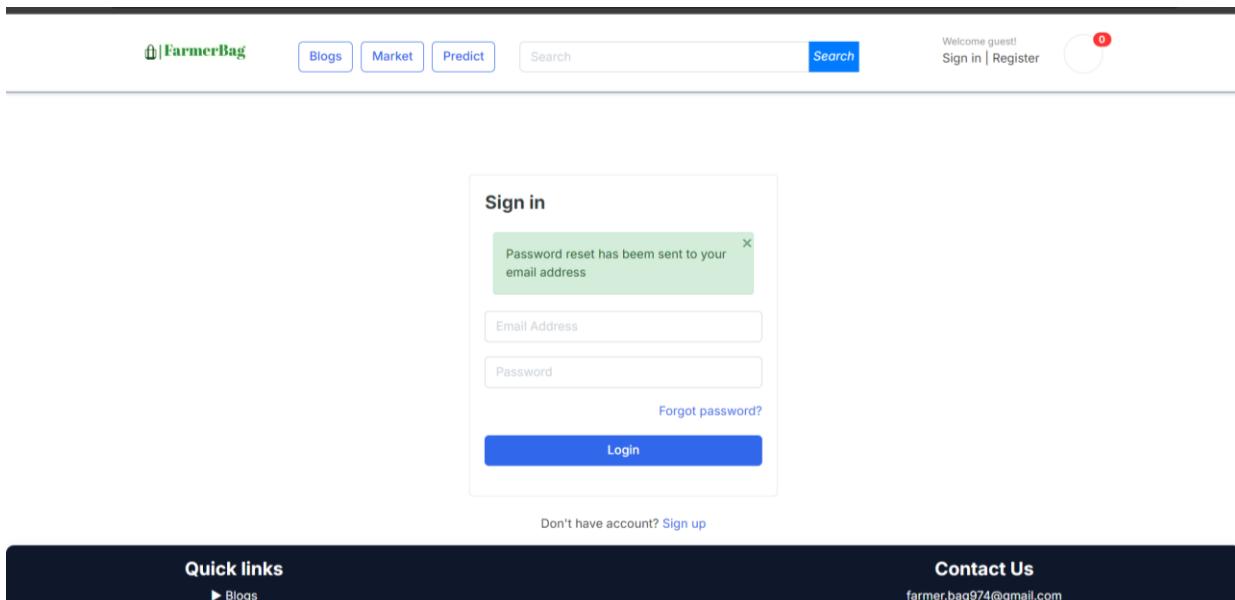
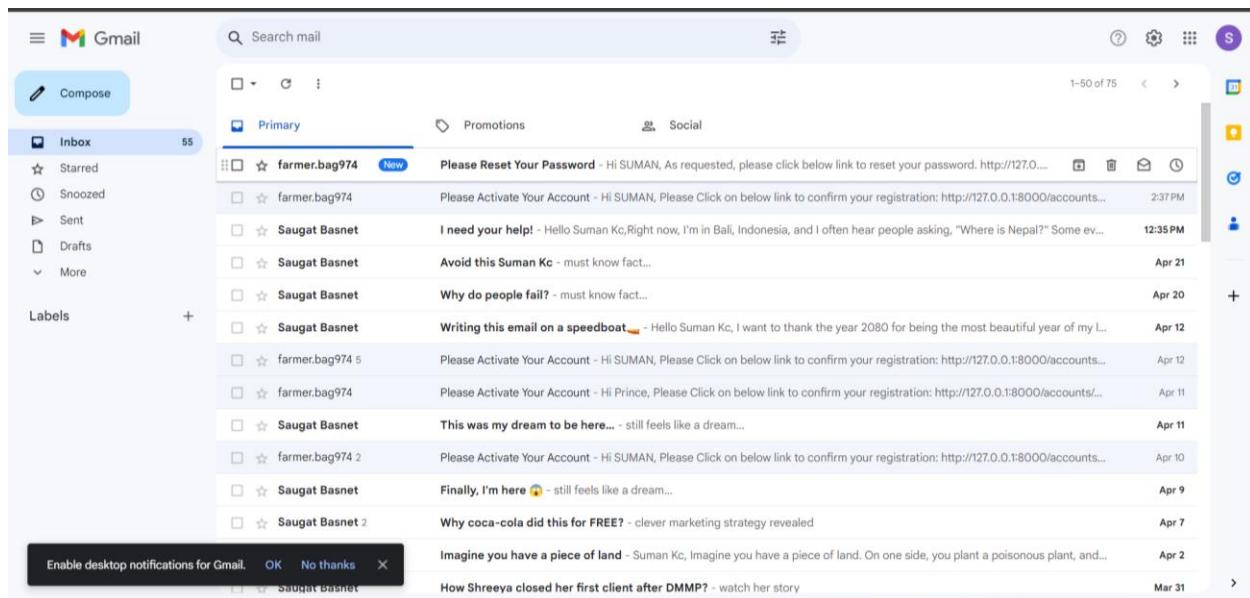
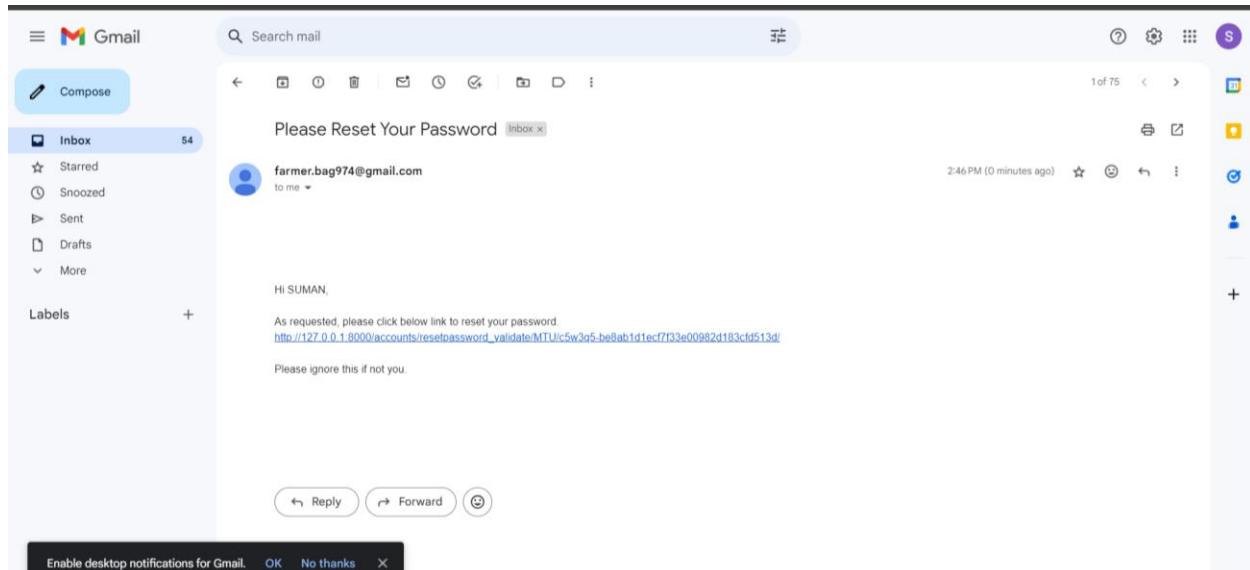


Figure 171: Reset password: reset link confirmation link.

**Figure 172: Reset password: Gmail Inbox****Figure 173: Reset password: reset password link.**

Reset Password

Please reset your password

Create Password

Confirm Password

Reset

Don't have an account? [Sign up](#)

Quick links

- Blogs
- Market
- Predict

Contact Us

farmer.bag974@gmail.com
+977-9744254833
Ghorhi Dang-15

© 2024 Farmers Bag

Figure 174: Reset password: reset password form

Reset Password

Please reset your password

....

....

Reset

Don't have an account? [Sign up](#)

Quick links

- Blogs
- Market
- Predict

Contact Us

farmer.bag974@gmail.com
+977-9744254833
Ghorhi Dang-15

© 2024 Farmers Bag

Figure 175: Reset password: Creating new password:

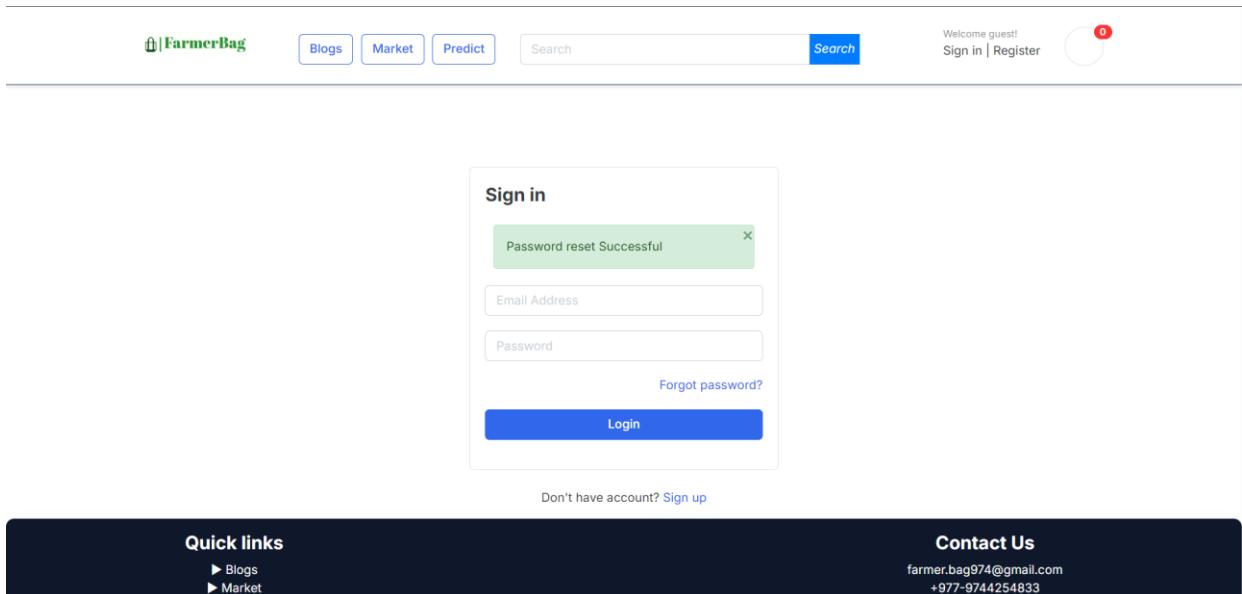


Figure 176: Reset Password: Screenshot of successful message.

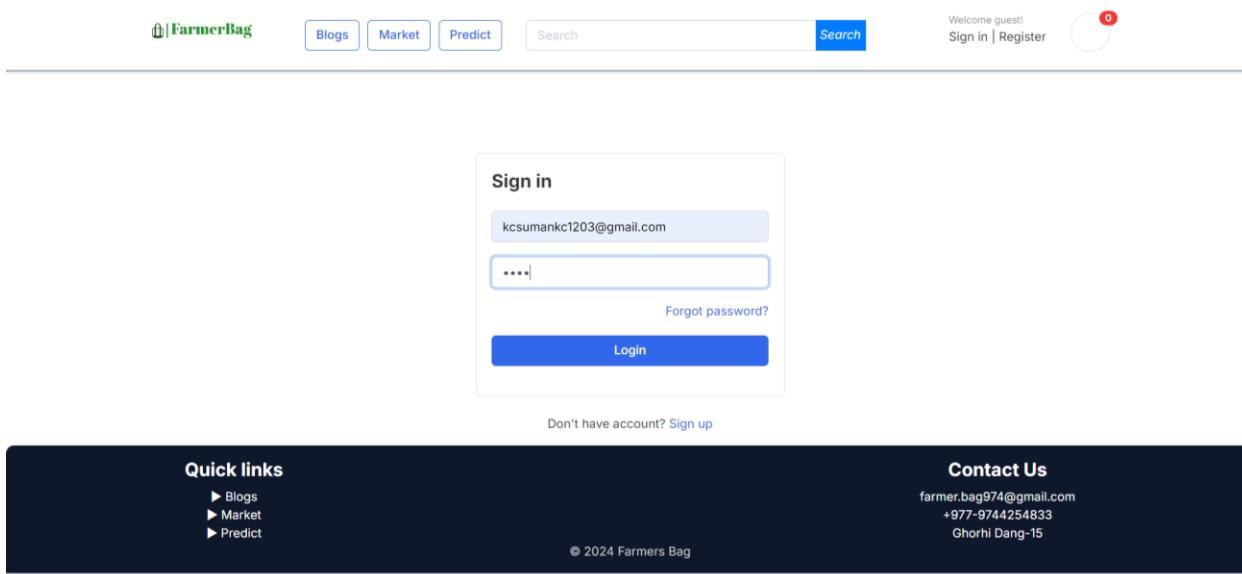
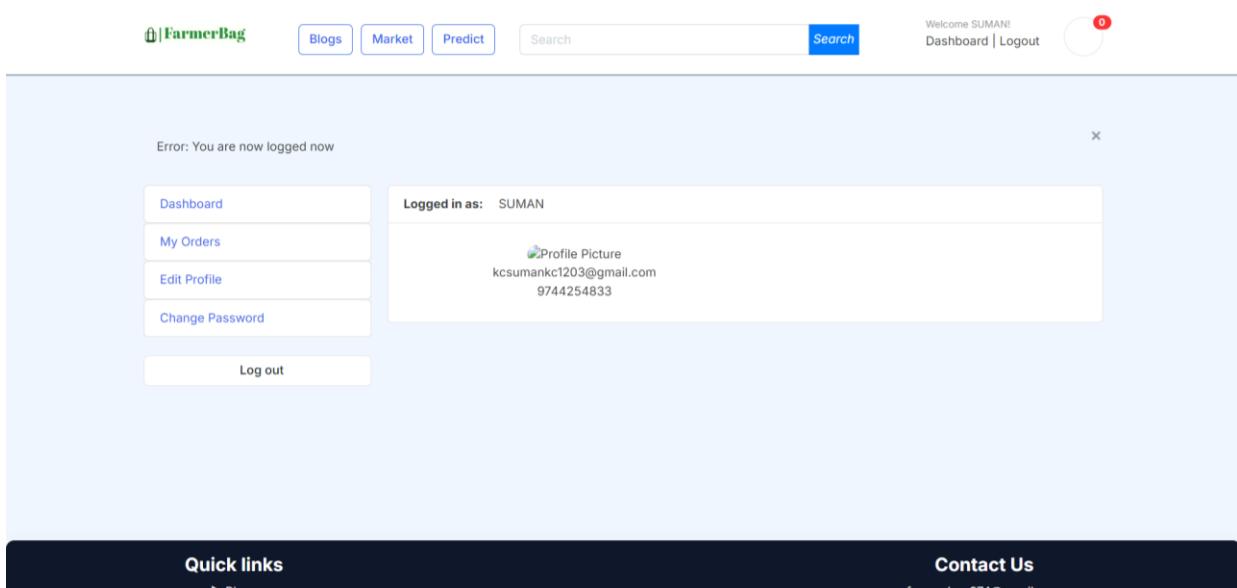


Figure 177: Reset Password: Login with new password

**Figure 178: Reset Password: Dashboard**

```

def resetpassword_validate(request, uidb64, token):
    try:
        uid = urlsafe_base64_decode(uidb64).decode() #this will decode uidb and store to uidb
        user = Account._default_manager.get(pk=uid)
    except(TypeError, ValueError, OverflowError, Account.DoesNotExist):
        user= None
    # to know whether is secure request ho ki hoina
    if user is not None and default_token_generator.check_token(user, token):
        request.session['uid'] = uid
        messages.success(request, 'Please reset your password')
        return redirect('resetPassword')
    else:
        messages.error(request, 'This link has been expired or is invalid..')
        return redirect('login')

def resetPassword(request):
    if request.method == 'POST':
        password = request.POST['password']
        confirm_password = request.POST['confirm_password']

        if password == confirm_password:
            uid = request.session.get('uid')
            user = Account.objects.get(pk=uid)
            user.set_password(password)
            user.save()
            messages.success(request, 'Password reset Successful')
            return redirect('login')
        else:
            messages.error(request, 'Password does not match')
            return redirect('resetPassword')
    else:
        return render(request, 'accounts/resetPassword.html')

```

Figure 179: Reset Password: Code(1)

```
def forgotPassword(request):
    if request.method == 'POST':
        email = request.POST['email']
        if Account.objects.filter(email = email).exists():
            # check email in database same to same or not
            user = Account.objects.get(email__exact= email)

            # Reset Password Email
            current_site = get_current_site(request)
            domain = current_site.domain
            mail_subject = 'Please Reset Your Password'
            message = render_to_string('accounts/reset_password.html',
            {
                'user': user,
                'domain': domain,
                'uid': urlsafe_base64_encode(force_bytes(user.pk)),
                'token': default_token_generator.make_token(user),
            })

            to_email = email
            send_email = EmailMessage(mail_subject, message, to=[to_email])
            send_email.send()

            messages.success(request, "Password reset has been sent to your email address")

            return redirect('login')

    else:
        messages.error(request, 'Account doesnot exists')
        return redirect('forgotPassword')

return render(request, 'accounts/forgotPassword.html')
```

Figure 180: Reset Password: Code (2)

```
{% extends "base.html" %}  
{% load static %}  
  
{% block content %}  
  
<div class="card mx-auto" style="max-width: 380px; margin-top:100px;">  
    <div class="card-body">  
        <h4 class="card-title mb-4">Reset Password</h4>  
        {% include 'includes/alerts.html' %}  
        <form action="{% url 'resetPassword' %}" method="POST">  
            {% csrf_token %}  
            <div class="form-group">  
                <input type="password" class="form-control" placeholder="Create Password" name="password">  
            </div>  
            <div class="form-group">  
                <input type="password" class="form-control" placeholder="Confirm Password" name="confirm_password">  
            </div>  
            <div class="form-group">  
                <button type="submit" class="btn btn-primary btn-block">Reset</button>  
            </div>  
        </form>  
    </div>  
<p class="text-center mt-4">Don't have an account? <a href="{% url 'register' %}">Sign up</a></p>  
  
{% endblock %}
```

Figure 181: Reset Password: Code (3)

4.2.11. TEST 11 – TO EDIT PROFILE

TEST	TWO
Objective	To verify logged in user to edit their profile details.
Action	<ul style="list-style-type: none"> ➤ Navigate to the user Dashboard. ➤ Click on the “Edit Profile” on the left bottom side. ➤ Enter details according to edit the profile. ➤ Click on “save” button. ➤ The system should save the new data of users.
Expected Result	<ul style="list-style-type: none"> ➤ The user should successfully be able to edit their profile. ➤ The user should view the edited profile.
Actual Result	The user was able to edit profile successfully.
Conclusion	The test was successful.

Table 17: White Box Testing: To Edit Profile

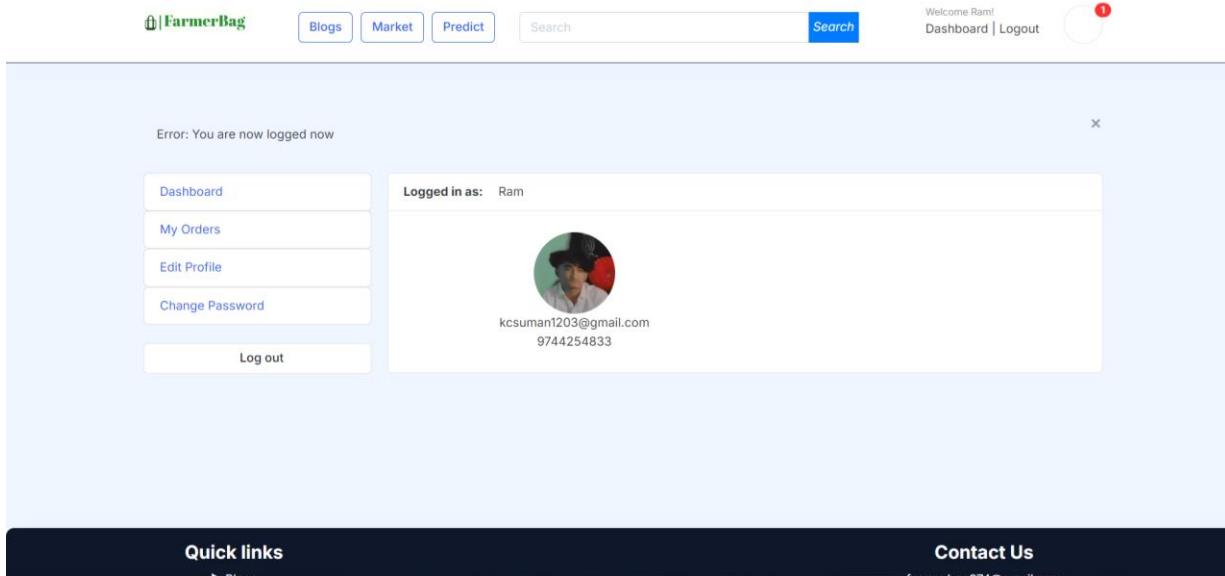


Figure 182: Edit Profile: Navigate to Dashboard

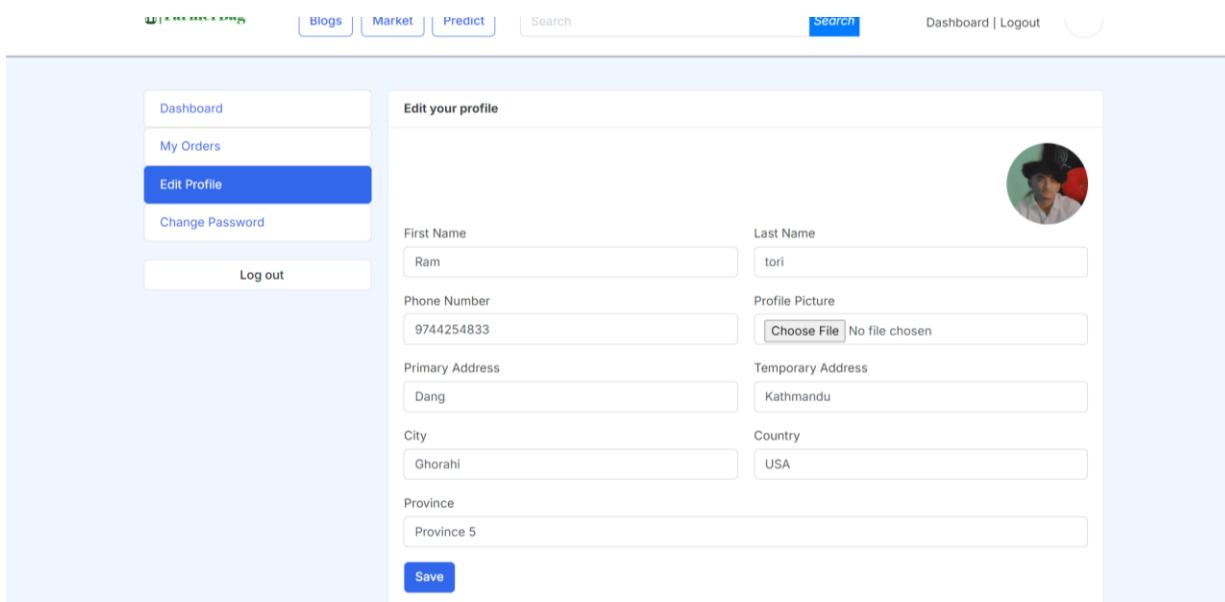


Figure 183: Edit Profile: Navigate to Edit Profile and Enter Details

Dashboard

My Orders

Edit Profile

Change Password

Log out

Edit your profile

First Name: Abiral

Last Name: Pradhanang

Phone Number: 9744254833

Primary Address: Kathmandu

Temporary Address: Nepaljung

City: Kuleshwor

Country: Nepal

Province: Province 3

Profile Picture: Choose File No file chosen

Save

Figure 184: Edit Profile: Click on save button.

Welcome Abiral!

Dashboard | Logout

Your profile has been updated.

Dashboard

My Orders

Edit Profile

Change Password

Log out

Edit your profile

First Name: Abiral

Last Name: Pradhanang

Phone Number: 9744254833

Primary Address: Kathmandu

Temporary Address: Nepaljung

City: Kuleshwor

Country: Nepal

Profile Picture: Choose File No file chosen

Figure 185: Edit Profile: Success Message Displays

```

@login_required(login_url='login')
def edit_profile(request):
    # Retrieve or create UserProfile for the current user
    userprofile, created = UserProfile.objects.get_or_create(user=request.user)

    if request.method == 'POST':
        user_form = UserForm(request.POST, instance=request.user)
        profile_form = UserProfileForm(request.POST, request.FILES, instance=userprofile)
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
            messages.success(request, 'Your profile has been updated.')
            return redirect('edit_profile')
    else:
        user_form = UserForm(instance=request.user)
        profile_form = UserProfileForm(instance=userprofile)

    context = {
        'user_form': user_form,
        'profile_form': profile_form,
        'userprofile': userprofile,
    }

    return render(request, 'accounts/edit_profile.html', context)

```

Figure 186: Edit Profile: Code (1)

The screenshot shows a code editor with the 'edit_profile.html' file open. The file content is as follows:

```

<section class="section-content padding-y bg" style="background-color: #ffffff; >
    <div class="row" style="margin-bottom: 10px;">
        <div class="col-md-9">
            <main class="card">
                <header class="card-header">
                    <strong class="d-inline-block mr-3">Edit your profile</strong>
                </header>
                <div class="card-body">
                    <div class="text-right">
                        <small><small>If you're a Farmer</small></small></small>
                    </div>
                    <div class="row">
                        <div class="col-md-6">
                            <form actions="{% url 'edit_profile' %}" method="POST" enctype="multipart/form-data">
                                <input type="hidden" name="csrf_token" value={{ csrf_token }}>
                                <div class="form-group">
                                    <label>First Name</label>
                                    {{ user_form.first_name }}
                                </div>
                                <div class="form-group">
                                    <label>Phone Number</label>
                                    {{ user_form.phone_number }}
                                </div>
                                <div class="form-group">
                                    <label>Primary Address</label>
                                    {{ profile_form.primary_address }}
                                </div>
                                <div class="form-group">
                                    <label>City</label>
                                    {{ profile_form.city }}
                                </div>
                            </form>
                        </div>
                    </div>
                </div>
            </main>
        </div>
    </div>
</section>

```

Figure 187: Edit Profile: Code (3)

```

File Edit Selection View Go Run Terminal Help ← → FarmerBag
EXPLORER TEST EXPLORER
FARMERBAG
static
store
pycache_
migrations
int_.py
admin.py
apps.py
forms.py
models.py
tests.py
urls.py
views.py
templates
accounts
account_verificatio...
admin_dashboard...
change_password...
dashboard.html
edit_profile.html
forgotPassword.html
login.html
register.html
reset_password.html
resetPassword.html
view_all_orders.html
view_orders.html
view_rating_review...
view_users.html
blog
includes
alerts.html
dashboard_sidebar...
footer.html
navbars.html
prediction
prediction.html
outputs
timeline
views.py accounts
edit_profile.html
urls.py accounts
views.py blog
views.py order
views.py store
views.py prediction
urls.py farmerbag
base.html
prediction.html
predictic ...
11 <div class="container">
12   <div class="row">
13     <main class="col-md-9">
14       <article class="card">
15         <div class="card-body">
16           <div class="form-group">
17             <label>Primary Address</label>
18             {{ profile_form.primary_address }}
19           </div>
20           <div class="form-group">
21             <label>City</label>
22             {{ profile_form.city }}
23           </div>
24         </div>
25         <div class="col-md-6">
26           <div class="form-group">
27             <label>Last Name</label>
28             {{ user_form.last_name }}
29           </div>
30           <div class="form-group">
31             <label>Profile Picture</label>
32             {{ profile_form.profile_picture }}
33           </div>
34           <div class="form-group">
35             <label>Temporary Address</label>
36             {{ profile_form.temporary_address }}
37           </div>
38           <div class="form-group">
39             <label>Country</label>
40             {{ profile_form.country }}
41           </div>
42           <div class="form-group col-md-12">
43             <label>Province</label>
44             {{ profile_form.province }}
45           </div>
46           <div>
47             <form>
48               <div>
49                 <div>
50                   <input type="submit" value="Save" class="btn btn-primary" style="margin-left: 1.2%;"/>
51                 </div>
52               </div>
53             </form>
54           </div>
55         </div>
56       </article>
57     </main>
58   </div>
59 </div>
60 </div>
61 </div>
62 </div>
63 </div>
64 </div>
65 </div>
66 </div>
67 </div>
68 </div>
69 </div>
70 </div>
71 </div>
72 </div>
73 </div>
74 </div>
75 </div>
76 </div>

```

Ln 35, Col 65 Spaces: 4 UTF-8 CRLF Django HTML Go Live Prettier

Figure 188: Edit Profile: Code (4)

4.2.12. TEST 12 – TO VIEW STUDY BLOG

TEST	Six
Objective	To verify that the user can access to the study blog for the information.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Navigation bar. ➤ Click on the “Blog” on navigation bar. ➤ System should redirect to blog section.
Expected Result	The user should successfully redirect to blog section.
Actual Result	The user was redirected to the blog section and have access to the information.
Conclusion	The test was successful.

Table 18: White Box Testing: To View Study Blog

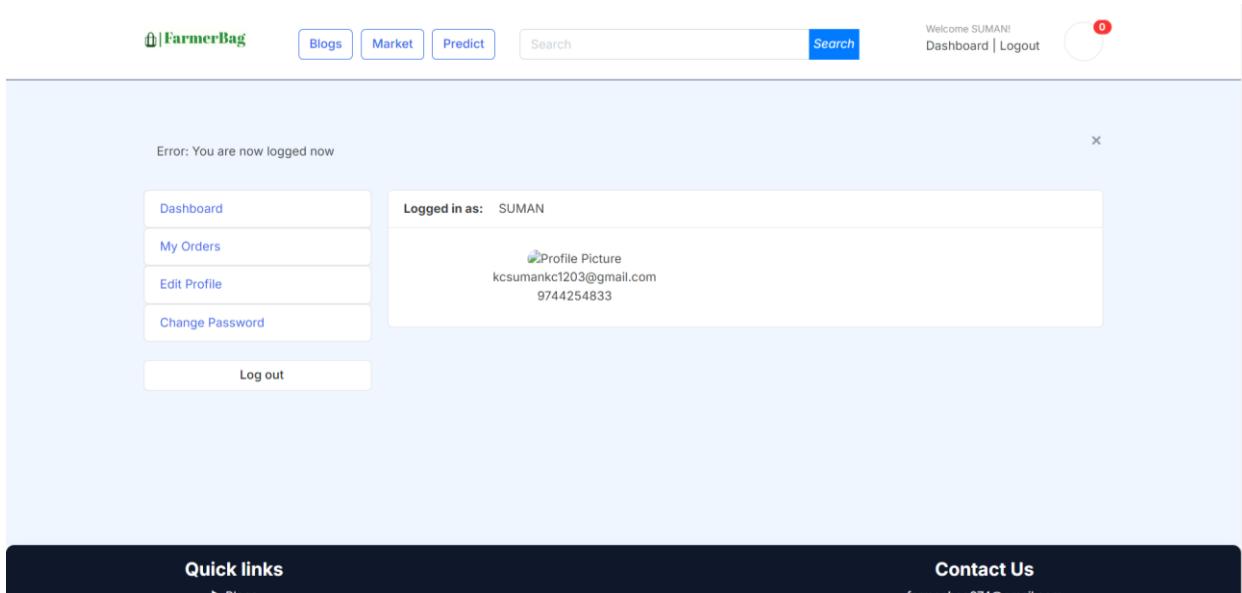


Figure 189: Study Blog: Dashboard

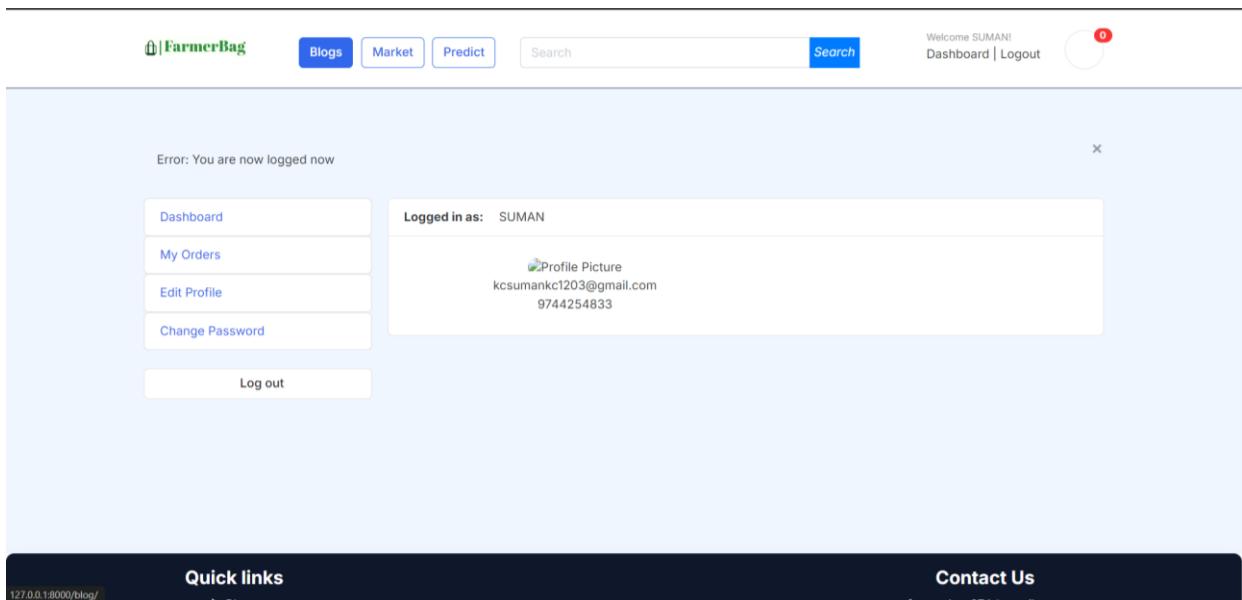


Figure 190: Study Blog: Navigate to Blog at Navigation Bar

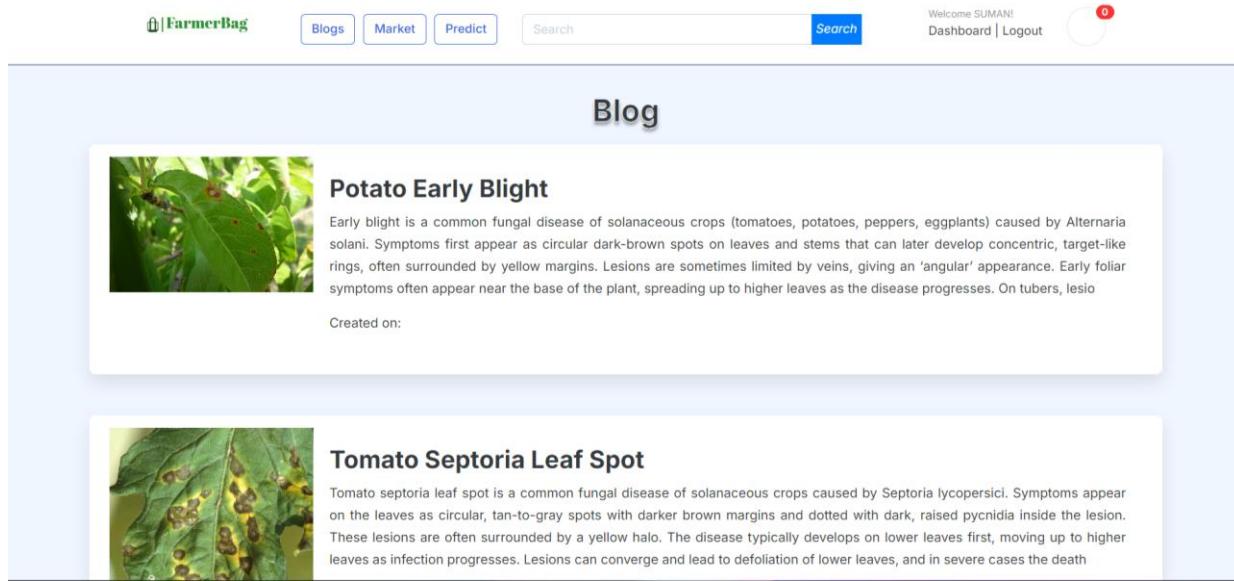


Figure 191: Blog: System redirected to Blog Page

```
def blog_list(request):
    blogs = BlogPost.objects.all()
    return render(request, 'blog/blog_post_list.html', {'blogs': blogs})

def blog_detail(request, slug):
    blog_post = get_object_or_404(BlogPost, slug=slug)
    return render(request, 'blog/blog_post_list.html', {'blog_post': blog_post})
```

Figure 192: Blog: Code (1)

```
{% extends "base.html" %}  
{% load static %}  
  
{% block content %}  
<link rel="stylesheet" type="text/css" href="{% static 'css/blog.css' %}">  
  
<div class="blog-container">  
    <h1>Blog</h1>  
    <div class="blog-post-container">  
        {% for post in blogs %}  
            <div class="blog-post">  
                <div class="blog-image">  
                      
                </div>  
                <div class="blog-content">  
                    <h2 class="blog-title">{{ post.title }}</h2>  
                    <p class="blog-description">{{ post.description }}</p>  
                    <p class="created-at">Created on: {{ post.created_at }}</p>  
  
                    <div class="buttons-blog">  
                        {% if request.user.is_authenticated and request.user.is_admin %}  
                            <button class="blog_update_button" onclick="window.location.href='{{ url 'update_blog_post' post.slug }}'">Update</button>  
                            <button class="blog_delete_button" onclick="window.location.href='{{ url 'delete_blog_post' post.slug }}'">Delete</button>  
                        {% endif %}  
                    </div>  
                </div>  
            {% endfor %}  
        </div>  
    </div>  
    <div>  
        <a href="#"> > add  
    </div>  
{% endblock %}
```

Figure 193: Blog: Code (2)

4.2.13. TEST 13 – TO SEARCH PRODUCT

TEST	Seven
Objective	To verify users can search the product form the search bar.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Search bar in Navigation bar. ➤ Enter valid product name. ➤ System should display the product according to the product name and system should displays “No searched product” if there are not any products.
Expected Result	<ul style="list-style-type: none"> ➤ The system should be able to displays the searched product. ➤ The system should display the “No Product Found Message” if there are no products.
Actual Result	The system was able show the searched product and able to displays the “No product found” if there are not any products.
Conclusion	The test was successful.

Table 19: White Box Testing: To Search Product

The screenshot shows the 'Our Store' page of the FarmerBag website. At the top, there is a navigation bar with links for 'Blogs', 'Market', 'Predict', a search bar containing 'Tomato', and a 'Search' button. To the right, there is a welcome message 'Welcome SUMANI!' and links for 'Dashboard' and 'Logout'. A circular icon with a red notification dot is also present.

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Medicine
- E-wears
- Machinery Equipment

Price range

Min: Rs. 0 Max: Rs.50

Apply

Found 10 items found

	RPS 76 fertilizer Rs. 100 View Details		Magnesium Sulphate Rs. 1050 View Details
	Paristha Bio Magnesium Rs. 80 View Details		

Figure 194: Search Product: Navigate to Search Bar

The screenshot shows the 'Search Results' page of the FarmerBag website. At the top, there is a navigation bar with links for 'Blogs', 'Market', 'Predict', a search bar containing 'Search', and a 'Search' button. To the right, there is a welcome message 'Welcome SUMANI!' and links for 'Dashboard' and 'Logout'. A circular icon with a red notification dot is also present.

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Medicine
- E-wears
- Machinery Equipment

Price range

Min: Rs. 0 Max: Rs.50

Apply

Found 1 items found

	Red Tomato Rs. 100 View Details
--	---

Figure 195: Search Product: Results of "Tomato".

The screenshot shows the search results for a product. On the left, there is a sidebar with 'Categories' and 'Price range' filters. The main area displays a single item: 'Red Tomato' priced at 'Rs. 100'. A 'View Details' button is present.

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Medicine
- E-wears
- Machinery Equipment

Price range

Min: Rs. 0 Max: Rs.50

Apply

Search Results

Found 1 items found

Red Tomato
Rs. 100

[View Details](#)

Figure 196: Search Product: Search Random Product

The screenshot shows the search results for a product. On the left, there is a sidebar with 'Categories' and 'Price range' filters. The main area displays a message: 'No result found. Please Try again!'

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Medicine
- E-wears
- Machinery Equipment

Price range

Min: Rs. 0 Max: Rs.50

Apply

Search Results

Found 0 items found

No result found. Please Try again!

Figure 197: Search Product: Message "No result found"

```

def search(request):
    products = None # Define products with a default value
    product_count = 0 # Define product_count with a default value

    if 'keyword' in request.GET:
        keyword = request.GET['keyword']

        if keyword:
            products = Product.objects.order_by('-created_date').filter(Q(description__icontains= keyword) | Q(product_name__icontains = keyword))
            product_count = products.count()

    context = {
        'products': products,
        'product_count': product_count,
    }

    return render(request, 'store/store.html', context)

```

Figure 198: Search Product: Code (1)

```

<div class="col-lg col-md-6 col-sm-12 col">
    <form action="{% url 'search' %}" class="search" method='GET'>
        <div class="input-group w-100">
            <input type="text" class="form-control" style="width:60%;" placeholder="Search" name="keyword">
            <div class="input-group-append">
                <button class="" type="submit" style="background-color: #007bff; border:none; color:#fff">
                    <i class="fa fa-search" >Search</i>
                </button>
            </div>
        </div>
    </form>
</div>

```

Figure 199: Search Product: Code (2)

4.2.14. TEST 14 – PAYMENT

TEST	Eight
Objective	To verify users can rate and review the single products.
Action	<ul style="list-style-type: none"> ➤ User should navigate to the product and click on the specific product. ➤ System redirects to Product Description Page. ➤ Click on Add to cart button and system redirect to Cart page. ➤ Click on “Checkout” button and system redirect to checkout page. ➤ Fill the form with all details for the further payment process. ➤ Select Khati for payment option. ➤ System redirects to Khalti Sandbox account and fill the details and payment will be successful after that. ➤ Select COD for payment option. ➤ System redirects to cart page.
Expected Result	<ul style="list-style-type: none"> ➤ System should redirect user to checkout page and fill all the details in checkout page. ➤ System should redirect to user to Khalti if pay button is pressed. ➤ User should buy the product from Khalti.
Actual Result	The user was able to buy the product.
Conclusion	The test was successful.

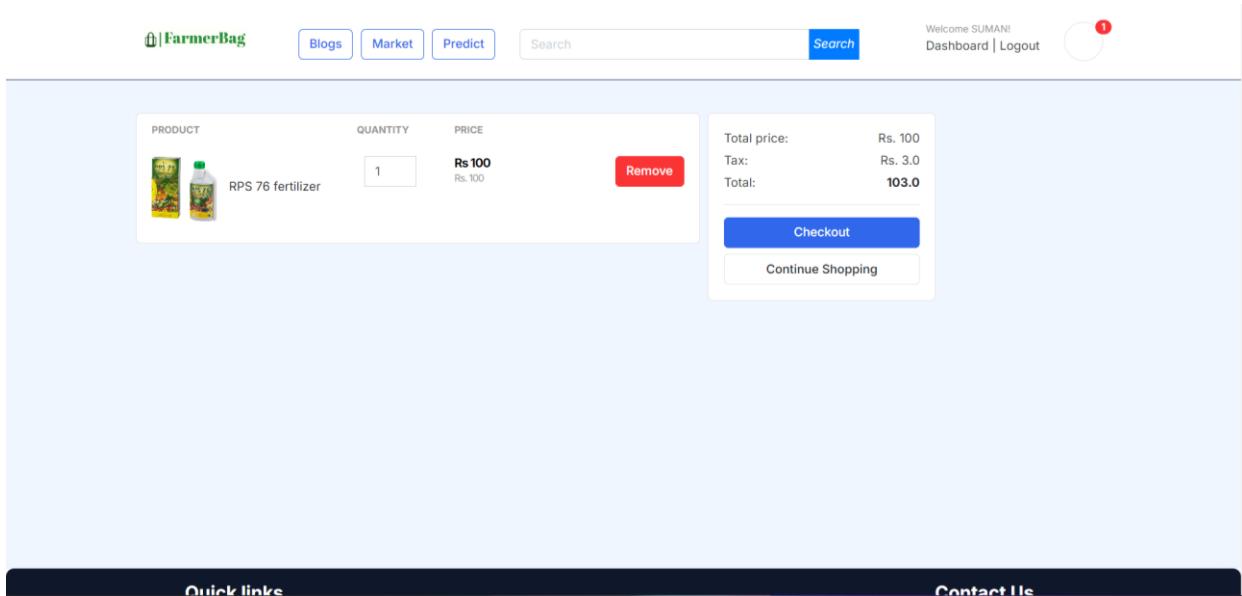
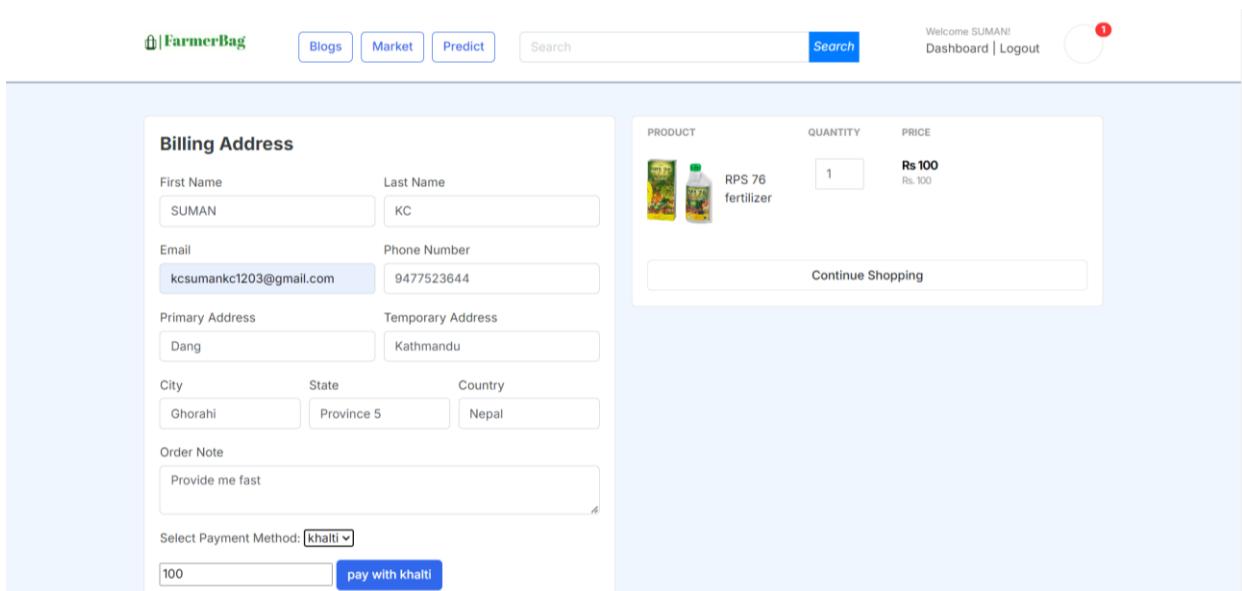
Table 20: White Box Testing: Payment

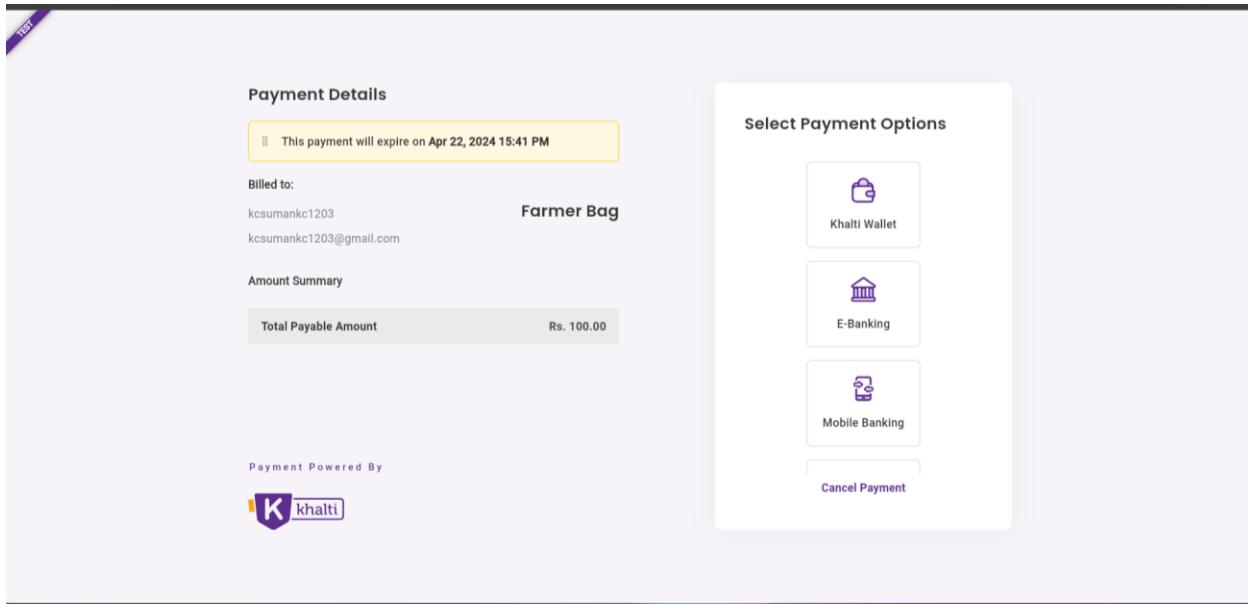
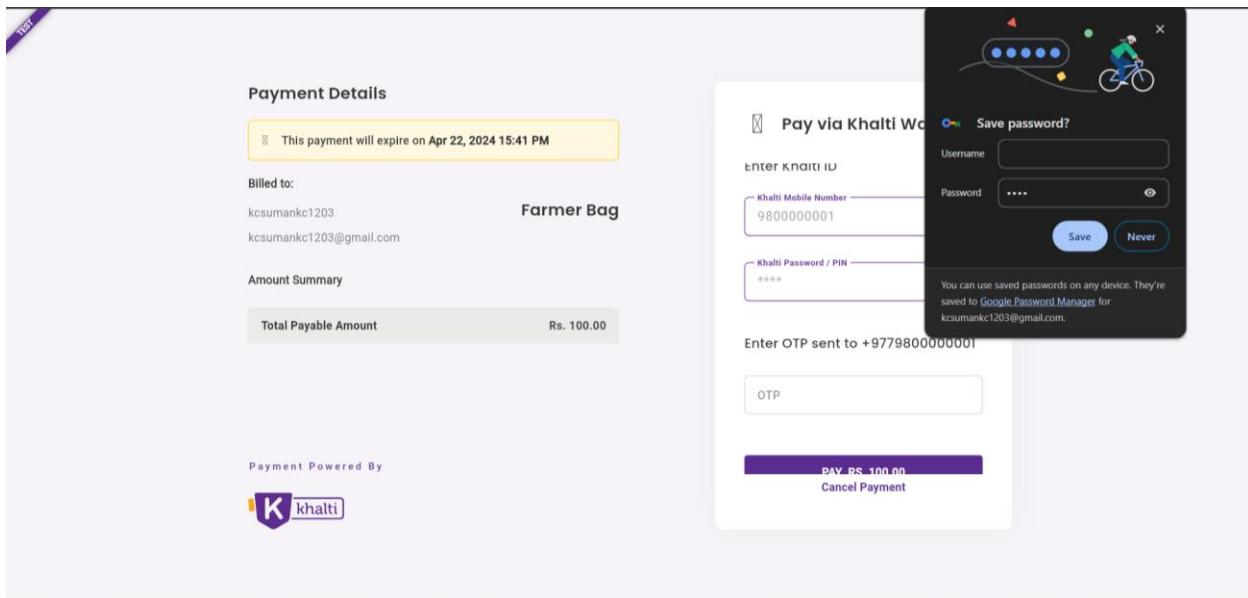
The screenshot shows the 'Our Store' section of the FarmerBag website. On the left, there's a sidebar with 'Categories' (All Products, Maize, Potato, Tomato, Pepper-Bell, Mediciene, E-wears, Machinery Equipment) and a 'Price range' filter (Min: Rs. 0, Max: Rs. 50, Apply button). The main area displays 10 items found, with three visible in the first row: 'RPS 76 fertilizer' (Rs. 100), 'Magnesium Sulphate' (Rs. 1050), and 'Paristha Bio Magnesium' (Rs. 80). Each item has a 'View Details' button below its image.

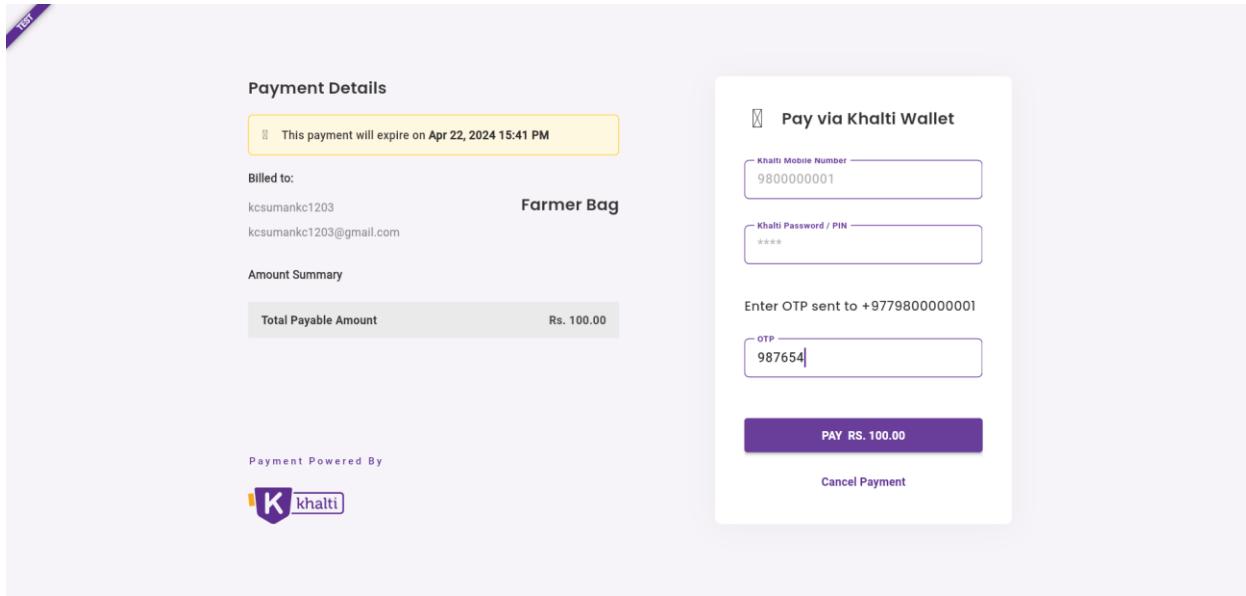
Figure 200: Payment: Navigate to Product Page

The screenshot shows the product details for 'RPS 76 fertilizer'. It features two images of the product: a 250 ml box and a 500 ml bottle. To the right, the product name 'RPS 76 fertilizer' and price 'Rs.100' are displayed. A detailed description follows: 'RPS 76 is an organic liquid fertilizer made from humic acids sourced from Leonardite. Chemical-free and pesticide-free, it boosts soil nutrition, strengthens plant immunity, and promotes healthy growth. Suitable for various crops including maize, corn, and potato, pepper bell, it ensures results within 15 days, making it perfect for sustainable agriculture.' An 'Add to cart' button is located below the description. At the bottom, there's a 'Customer Reviews' section and a URL bar showing '127.0.0.1:8000/cart/add_cart/2/ kcsuman1203'.

Figure 201: Payment: Navigate to Product Description Page

**Figure 202: Payment: Product Added to Cart****Figure 203: Payment: Redirected to Checkout Page and choose khalti**

**Figure 204: Payment: Khalti Interface (1)****Figure 205: Payment: Khalti Interface (2)**

***Figure 206: Payment: Khalti Interface (2)***

```
def payment(request):
    payment_method_choices = Payment.PAYMENT_CHOICES

    # Get the user's cart item
    user_cart_item = CartItem.objects.get(user=request.user)

    # Retrieve the cart associated with the user's cart item
    user_cart = user_cart_item.cart

    # Retrieve cart items associated with the user's cart
    cart_items = user_cart.cartitem_set.all()

    total_price = sum(item.sub_total() for item in cart_items) * 100
    total_display_price = sum(item.sub_total() for item in cart_items)

    context = {
        'cart_items': cart_items,
        'total_price': total_price,
        'total_display_price': total_display_price,
        'payment_method_choices': payment_method_choices
    }
    return render(request, 'store/checkout.html', context)
```

Figure 207: Payment: Code (1)

```

def Khalti_payment(request):
    payment_method = request.POST.get('payment_method')

    if payment_method == 'Khalti':
        url = "https://a.khalti.com/api/v2/epayment/initiate/"

        return_url = request.POST.get('return_url')
        purchase_order_id = request.POST.get('purchase_order_id')
        amount = request.POST.get('amount')

        print("return_url", return_url)
        print("purchase_order_id", purchase_order_id)
        print("amount", amount)
        user = request.user

        payload = json.dumps({
            "return_url": return_url,
            "website_url": "http://127.0.0.1:8000",
            "amount": amount,
            "purchase_order_id": purchase_order_id,
            "purchase_order_name": "test",
            "customer_info": {
                "name": user.username,
                "email": user.email,
            }
        })
        headers = {
            'Authorization': 'key 5a9e103a394141f58384b56fe50573a0',
            'Content-Type': 'application/json',
        }

        response = requests.request("POST", url, headers=headers, data=payload)
        print(response.text)
        new_res = json.loads(response.text)
        print(new_res)
        return redirect(new_res['payment_url'])
    else:
        return redirect('COD')

```

Figure 208:Payment: Code (2)

```

def COD(request):
    payment_method = 'COD'
    payment_status = 'Pending.....'
    user = request.user
        (method) def create(**kwargs: Any) -> Payment
    payment = Payment.objects.create(payment_method=payment_method, payment_status=payment_status, user=user)

    user_cart_item = CartItem.objects.get(user=user)

    # Retrieve the cart associated with the user's cart item
    user_cart = user_cart_item.cart

    # Retrieve cart items associated with the user's cart
    cart_items = user_cart.cartitem_set.all()
    total_price = sum(item.sub_total() for item in cart_items)
    shipping_address = "hello"

    order = Order.objects.create(user=user, total_price=total_price, shipping_address=shipping_address, payment=payment)

    for cart_item in cart_items:
        OrderItems.objects.create(order=order, product=cart_item.product, quantity=cart_item.quantity)

    return render(request, 'store/cart.html')

```

Figure 209:Payment: Code (3)

```

<% extends "base.html" %>
<% load static %>
<div rel="stylesheet" type="text/css" href="{% static 'css/checkout.css' %}>
<section class="section-content padding-y bg" style="background-color: #eff6ff;">
<div class="container">
    <div class="row">
        <aside class="col-lg-6">
            <div class="card">
                <div class="card-body">
                    <h4 class="card-title mb-4">Billing Address</h4>
                    <form action="{% url 'khalti_payment' %}" method="POST">
                        <input type="hidden" name="csrf_token" value="X" />
                        <div class="form-row">
                            <div class="col form-group">
                                <label for="first_name">First Name</label>
                                <input type="text" name="first_name" class="form-control" required>
                            </div>
                            <div class="col form-group">
                                <label for="last_name">Last Name</label>
                                <input type="text" name="last_name" class="form-control" required>
                            </div>
                        </div>
                        <div class="form-row">
                            <div class="col form-group">
                                <label for="email">Email</label>
                                <input type="email" name="email" class="form-control" required>
                            </div>
                            <div class="col form-group">
                                <label for="phone">Phone Number</label>
                                <input type="text" name="phone" class="form-control" required>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </aside>
    </div>
</div>

```

Figure 210: Payment: Code (4)

```

<section class="section-content padding-y bg" style="background-color: #eff6ff;">
    <div class="container">
        <div class="row">
            <div class="col-lg-6">
                <div class="form-group">
                    <label for="Phone Number"></label>
                    <input type="text" name="phone" class="form-control" required/>
                </div>
                <div class="form-row">
                    <div class="col form-group">
                        <label for="Primary Address"></label>
                        <input type="text" name="address_line_1" class="form-control" required/>
                    </div>
                    <div class="col form-group">
                        <label for="Temporary Address"></label>
                        <input type="text" name="address_line_2" class="form-control"/>
                    </div>
                </div>
                <div class="form-row">
                    <div class="col form-group">
                        <label for="City"></label>
                        <input type="text" name="city" class="form-control" required/>
                    </div>
                    <div class="col form-group">
                        <label for="State"></label>
                        <input type="text" name="state" class="form-control" required/>
                    </div>
                    <div class="col form-group">
                        <label for="Country"></label>
                        <input type="text" name="country" class="form-control" required/>
                    </div>
                </div>
                <div class="form-row">
                    <div class="col form-group">
                        <label for="Order Note"></label>
                        <textarea name="order_note" rows="2" class="form-control">
                    </div>
                </div>
            </div>
        </div>
    </div>

```

Figure 211: Payment: Code (5)

```

<div class="col-lg-6">
    <div class="card">
        <div class="card-body">
            <div class="form-group">
                <label for="order_note"></label>
                <textarea name="order_note" rows="2" class="form-control">
            </div>
            <div class="form-group">
                <p>Select Payment Method:
                <select name="payment_method" id="payment-method-select">
                    <% for method in payment_method_choices %>
                    <option value="{{ method.0 }}">{ { method.1 }}</option>
                    <% endfor %}
                </select>
            </div>
            <div class="form-group">
                <input type="hidden" name="purchase_order_id" value="1"/>
                <input type="hidden" name="amount" value="{{ total_price }}">
                <input type="hidden" name="return_url" value="http://127.0.0.1:8000/order/Khalti_payments">
                <input type="text" value="{{ total_display_price }}" readonly/>
                <input type="submit" class="btn btn-primary" value="pay with khalti"/>
            </div>
        </div>
    </div>
</div>
<div class="col-lg-6">
    <div class="card">
        <table class="table table-bordered table-shopping-cart">
            <thead>
                <tr>
                    <th colspan="5" style="text-align: center;">Cart Items
                </tr>
                <tr>
                    <th style="text-align: left;">Product
                    <th style="text-align: center;">Quantity
                    <th style="text-align: center;">Price
                    <th style="text-align: right;">Total
                    <th style="width: 5%;">Action
                </tr>
            <tbody>
                <% for cart_item in cart_items %>
                <tr>
                    <td style="text-align: left;">{{ cart_item.product }}
                    <td style="text-align: center;">{{ cart_item.quantity }}
                    <td style="text-align: center;">{{ cart_item.price }}
                    <td style="text-align: right;">{{ cart_item.total }}
                    <td style="text-align: center;">
                        <a href="#">View
                        <a href="#">Edit
                        <a href="#">Delete
                    </td>
                </tr>
                <% endfor %}
            </tbody>
        </table>
    </div>

```

Figure 212: Payment: Code (6)

4.2.15. TEST 15 – TO DELETE USER ACCOUNT

TEST	Nine
Objective	To ensure that admin can successfully delete user account from the system.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “View User” and system displays the total user account that are registered. ➤ Click on “Delete” icon button. ➤ Verify that the user account is removed from the system.
Expected Result	The user account associated with the user is removed form the system.
Actual Result	The user account associated with the user was deleted from the system.
Conclusion	The test was successful.

Table 21: White Box Testing: To Delete User Account

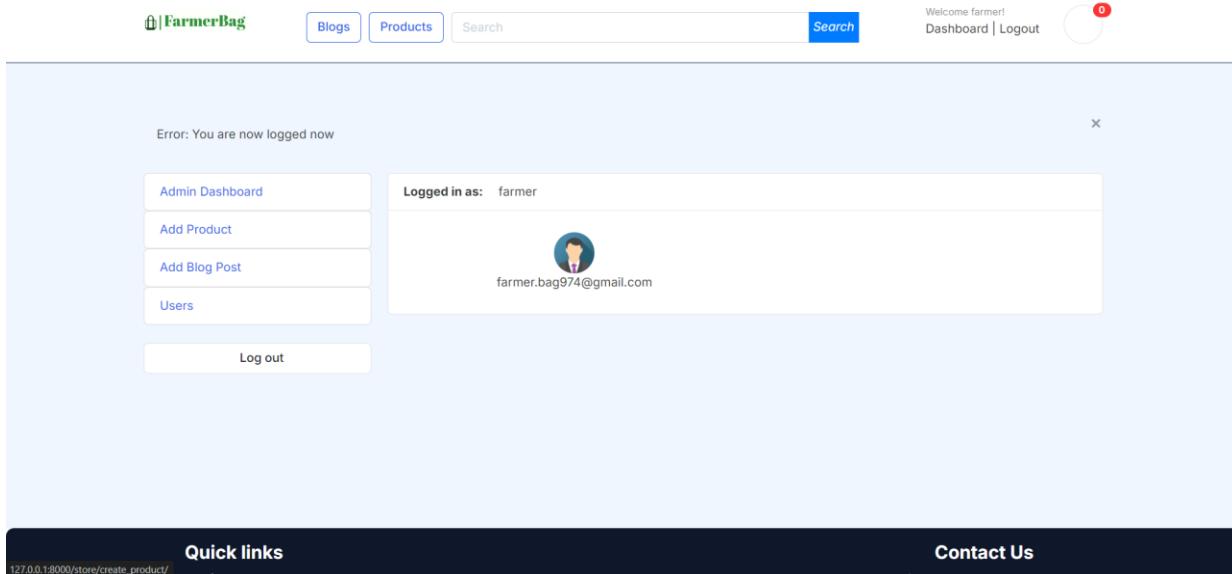


Figure 213: Delete Account: Admin Dashboard

The screenshot shows the User List page of the FarmerBag application. At the top, there is a navigation bar with links for Blogs, Products, and Search. On the right side of the header, it says "Welcome farmer!" followed by "Dashboard | Logout". A circular icon with a red notification dot is also present. The main content area has a title "User List". Below the title is a table with columns: Username, Email, First Name, Last Name, and Action. The table contains five rows of data:

Username	Email	First Name	Last Name	Action
farmer	farmer.bag974@gmail.com	farmer	farmer	Delete
kcsuman1203	kcsuman1203@gmail.com	Abiral	Pradhanang	Delete
1203lozer	1203lozer@gmail.com	Bro	K.C.	Delete
kcsumankc1203	kcsumankc1203@gmail.com	SUMAN	KC	Delete

At the bottom of the page, there is a "Quick links" section with links for Blogs, Market, and Predict. On the right side, there is a "Contact Us" section with the email "farmer.bag974@gmail.com", phone number "+977-9744254833", and address "Ghorhi Dang-15". The footer of the page includes the copyright notice "© 2024 Farmers Bag" and the URL "127.0.0.1:8000/accounts/delete_user/15".

Figure 214: Delete Account: Navigate to Users

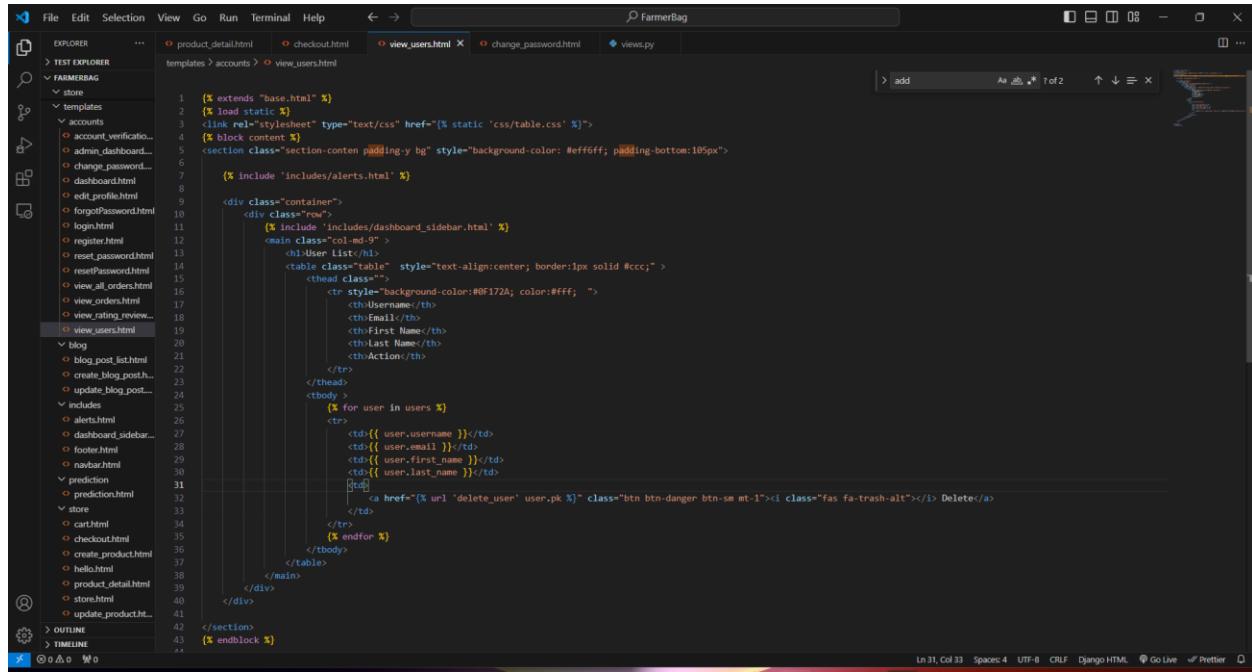
Username	Email	First Name	Last Name	Action
farmer	farmer.bag974@gmail.com	farmer	farmer	Delete
kcsuman1203	kcsuman1203@gmail.com	Abiral	Pradhanang	Delete
1203lozer	1203lozer@gmail.com	Bro	K.C.	Delete

Figure 215: Delete Account: Click on Delete Account

```
def view_users(request):
    users = Account.objects.all()
    context = {'users': users}
    return render(request, 'accounts/view_users.html', context)

def delete_user(request, pk):
    user = get_object_or_404(Account, id=pk)
    user.delete()
    return redirect('view_users')
```

Figure 216: Delete Account: Code (1)



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists various Django files and templates. The main area displays a portion of a Django template named `view_users.html`. The code includes logic for displaying a list of users and providing a delete link for each user.

```

1  {% extends "base.html" %}>
2  {% load static %}
3  <link rel="stylesheet" type="text/css" href="{% static 'css/table.css' %}">
4  {% block content %}
5    <section class="section-content padding-y bg" style="background-color: #ffffff; padding-bottom:10px">
6      {% include 'includes/alerts.html' %}
7      <div class="container">
8        <div class="main">
9          {% include 'includes/dashboard_sidebar.html' %}
10         <main class="col-md-9">
11           <h1>User List</h1>
12           <table class="table" style="text-align:center; border:1px solid #ccc;">
13             <thead class="">
14               <tr style="background-color:#0f172a; color:#fff; ">
15                 <th>Username</th>
16                 <th>Email</th>
17                 <th>First Name</th>
18                 <th>Last Name</th>
19                 <th>Action</th>
20               </tr>
21             </thead>
22             <tbody>
23               {% for user in users %}
24                 <tr>
25                   <td>{{ user.username }}</td>
26                   <td>{{ user.email }}</td>
27                   <td>{{ user.first_name }}</td>
28                   <td>{{ user.last_name }}</td>
29                   <td>
30                     <a href="{% url 'delete_user' user.pk %}" class="btn btn-danger btn-sm mt-1"><i class="fas fa-trash-alt"></i> Delete</a>
31                   </td>
32                 </tr>
33               {% endfor %}
34             </tbody>
35           </table>
36         </main>
37       </div>
38     </div>
39   </section>
40   </div>
41 </div>
42 </section>
43 {% endblock %}

```

Figure 217: Delete Account: Code (2)

4.2.16. TEST 16 – TO FILTER PRODUT BY PRICE RANGE

TEST	Ten
Objective	To ensure that the user can search or filter product by price.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Marketplace. ➤ Select max and min price form the Filter form. ➤ Choose “Apply” button.
Expected Result	<ul style="list-style-type: none"> ➤ The system should display the product by price range.
Actual Result	<ul style="list-style-type: none"> ➤ The system successfully displays the product by price range.
Conclusion	The test was successful.

Table 22: White Box Testing: To Filter Product by Price

The screenshot shows the 'Our Store' section of the FarmerBag website. On the left, there's a sidebar with 'Categories' and a 'Price range' filter. The 'Price range' filter has 'Min' set to 'Rs. 0' and 'Max' set to 'Rs.50'. Below it is an 'Apply' button. The main area displays a search result titled 'Found 2 items found'. It shows two products: 'Agricultural Boots' (Rs. 1000) and 'Agricultural Gloves' (Rs. 99). Each product has a 'View Details' button.

Figure 218: Filter Product By Price: Market Page

This screenshot is similar to Figure 218, showing the 'Our Store' section. The 'Price range' filter now has 'Min' set to 'Rs. 0' and 'Max' set to 'Rs.1000'. The 'Apply' button is visible below the filters. The search result still shows 'Found 2 items found' for 'Agricultural Boots' (Rs. 1000) and 'Agricultural Gloves' (Rs. 99), each with a 'View Details' button. A URL at the bottom left of the page reads '127.0.0.1:8000/store/category/e-wears/#'.

Figure 219: Filter by Price: Select Price Range

FarmerBag Blogs Market Predict Search Welcome SUMAN! Dashboard | Logout

Our Store

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Mediciene
- E-wears
- Machinery Equipment

Price range

Min Max

Rs. 0 Rs.50

Apply

Found items found

RPS 76 fertilizer Rs. 100 View Details

Paristha Bio Magensium Rs. 80 View Details

Yellow Maize Rs. 50 View Details

Previous 1 2 3 Next

The screenshot displays a user interface for a e-commerce platform named 'FarmerBag'. On the left, there's a sidebar with various product categories and a price range filter set from Rs. 0 to Rs. 50. The main content area is titled 'Our Store' and shows three products: 'RPS 76 fertilizer' (Rs. 100), 'Paristha Bio Magensium' (Rs. 80), and 'Yellow Maize' (Rs. 50). Each product listing includes a 'View Details' button.

Figure 220: Filter by Price: Choose Apply button and Product Displays

```
def filter_products_by_price(products, min_price=None, max_price=None):
    if min_price is not None:
        products = products.filter(price__gte=min_price)
    if max_price is not None:
        products = products.filter(price__lte=max_price)
    return products


def filter_by_price(request):
    min_price = request.GET.get('min_price')
    max_price = request.GET.get('max_price')

    # Retrieve all products
    products = Product.objects.filter(is_available=True)

    # Filter products based on price range
    if min_price or max_price:
        products = filter_products_by_price(products, min_price=min_price, max_price=max_price)

    # Pagination
    paginator = Paginator(products, 3) # Show 3 products per page
    page_number = request.GET.get('page')
    try:
        products = paginator.page(page_number)
    except PageNotAnInteger:
        # If page is not an integer, deliver first page.
        products = paginator.page(1)
    except EmptyPage:
        # If page is out of range (e.g. 9999), deliver last page of results.
        products = paginator.page(paginator.num_pages)

    # Pass filtered and paginated products to the store template
    context = {
        'products': products,
    }

    return render(request, 'store/store.html', context)
```

Figure 221: Filter by price: Code (1).

```
<form method="get" action="{% url 'filter_by_price' %}">
  <article class="filter-group">
    <header class="card-header">
      <a href="#" data-toggle="collapse" data-target="#collapse_3" aria-expanded="true" class="">
        <i class="icon-control fa fa-chevron-down"></i>
        <h6 class="title">Price range</h6>
      </a>
    </header>
    <div class="filter-content collapse show" id="collapse_3" style="">
      <div class="card-body">
        <div class="form-row">
          <div class="form-group col-md-6">
            <label>Min</label>
            <select name="min_price" class="mr-2 form-control">
              <option value="0">Rs. 0</option>
              <option value="50">Rs.50</option>
              <option value="100">Rs.100</option>
              <option value="150">Rs.150</option>
              <option value="200">Rs.200</option>
              <option value="500">Rs.500</option>
              <option value="1000">Rs.1000</option>
            </select>
          </div>
          <div class="form-group text-right col-md-6">
            <label>Max</label>
            <select name="max_price" class="mr-2 form-control">
              <option value="50">Rs.50</option>
              <option value="100">Rs.100</option>
              <option value="150">Rs.150</option>
              <option value="200">Rs.200</option>
              <option value="500">Rs.500</option>
              <option value="1000">Rs.1000</option>
              <option value="2000">Rs.2000</option>
            </select>
          </div>
        </div>
        <button type="submit" class="btn btn-block btn-primary">Apply</button>
      </div>
    </div>
  </article>
</form>
```

Figure 222: Filter by price: Code (2)

4.2.17. TEST 17 – TO SEARCH PRODUCT BY CATEGORY

TEST	Eleven
Objective	To ensure that the user can search the product by category.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Product Page. ➤ Navigate Category Section and select category name from that.
Expected Result	<ul style="list-style-type: none"> ➤ The system should display the product by category. ➤ User should be able to view the product by category.
Actual Result	The system successfully displays the product by category and user successfully views the product by category.
Conclusion	The test was successful.

Table 23:White Box Testing: To Search Product by Category

The screenshot shows the 'Market' section of the FarmerBag website. At the top, there are navigation links for 'Blogs', 'Market', 'Predict', and a search bar. A user profile is shown with 'Welcome SUMAN!' and 'Dashboard | Logout'. Below the header, the title 'Our Store' is displayed. On the left, a sidebar titled 'Categories' lists product types: All Products, Maize, Potato, Tomato, Pepper-Bell, Mediciene, E-wears, and Machinery Equipment. Under 'Price range', there are dropdown menus for 'Min' (Rs. 0) and 'Max' (Rs. 50), with an 'Apply' button below them. The main area shows a grid of products found: 'RPS 76 fertilizer' (Rs. 100), 'Magnesium Sulphate' (Rs. 1050), and 'Paristha Bio Magensium' (Rs. 80). Each product has a 'View Details' button.

Figure 223: Search Product by category: Market Page and Select Category on left side

This screenshot shows the same Market page as Figure 223, but with a specific category selected: 'Potato'. The sidebar now highlights 'Potato' under the 'Categories' section. The search results show one item: 'Wardan Potato Crops' at Rs. 100. The product image is visible, along with its name and price, and a 'View Details' button.

Figure 224: Search Product by category: Displays Product by category.

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Medicine
- E-wears
- Machinery Equipment

Price range

Min: Rs. 0 Max: Rs.50

Apply

Found 1 items found

Wardan Potato Crops
Rs. 100

View Details

127.0.0.1:8000/store/category/e-wears/

Figure 225: Search Product by category: Displays Product by category 'Potato'.

Categories

- All Products
- Maize
- Potato
- Tomato
- Pepper-Bell
- Medicine
- E-wears
- Machinery Equipment

Price range

Min: Rs. 0 Max: Rs.50

Apply

Found 2 items found

Agricultural Boots
Rs. 1000

View Details

Agricultural Gloves
Rs. 99

View Details

Figure 226Search Product by category: Displays Product by category 'E wears'.

```
<div class="card">
  <article class="filter-group">
    <header class="card-header">
      <a href="#" data-toggle="collapse" data-target="#collapse_1" aria-expanded="true" class="">
        <i class="icon-control fa fa-chevron-down"></i>
        <h4 class="title">Categories</h4>
      </a>
    </header>
    <div class="filter-content collapse show" id="collapse_1" >
      <div class="card-body">

        <ul class="list-menu">
          <li><a href="{% url 'store' %}">All Products </a></li>

          {% for category in links %}
            <li><a href="{{ category.get_url }}">{{category.category_name}}</a></li>
          {% endfor %}
        </ul>
      </div> <!-- card-body.-->
    </div>
  </article> <!-- filter-group .-->
```

Figure 227:Search Product by category: Code (1)

4.2.18. TEST 18 – TO USE PAGINATION

TEST	Twelve
Objective	To ensure that the user can use pagination to view the products from store page.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Product Page. ➤ Click on the ‘Next button’ or ‘page number’. ➤ System redirects to the selected page number or the next page.
Expected Result	<ul style="list-style-type: none"> ➤ The system should redirect tot the selected page or page number. ➤ User should view the products of the next page.
Actual Result	The user successfully viewed the product of the next page..
Conclusion	The test was successful.

Table 24: White Box Testing: To Use Pagination

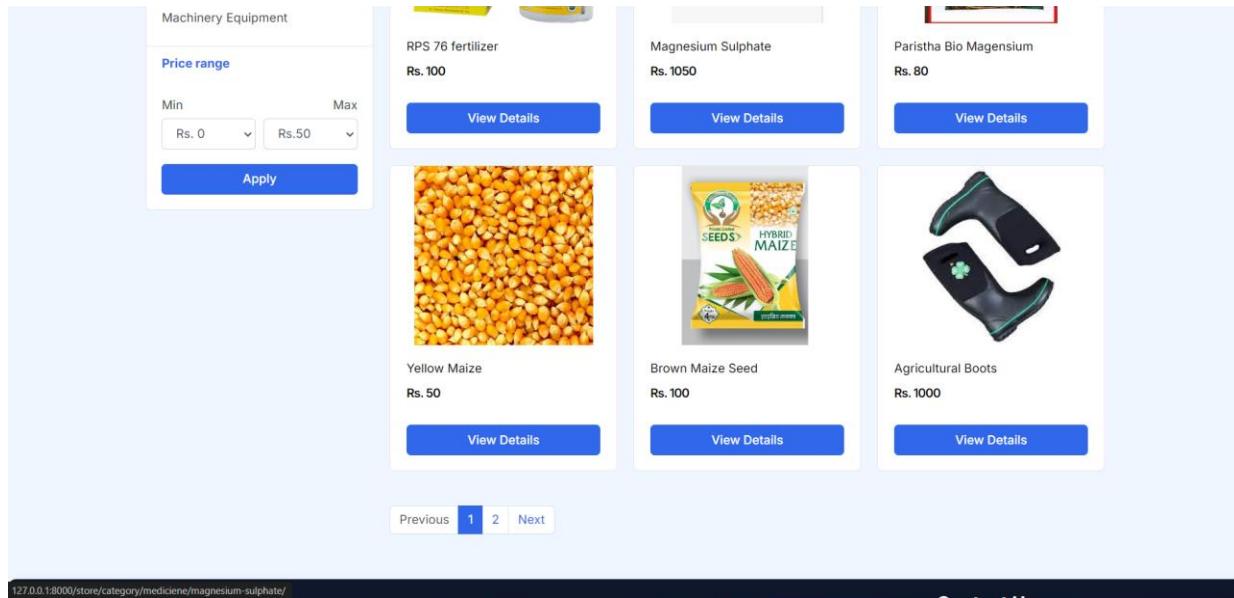


Figure 228: Pagination: Navigate to Market page and Select next or '1' from bottom of page.

This screenshot shows the 'Our Store' page. At the top, there is a navigation bar with links for 'Blogs', 'Market' (highlighted in blue), 'Predict', a search bar, and a 'Search' button. To the right, it says 'Welcome SUMANI', 'Dashboard | Logout', and a user icon with a red notification dot. The main content area is titled 'Our Store' and displays a message 'Found 10 items found'. On the left, there is a sidebar with 'Categories' (All Products, Maize, Potato, Tomato, Pepper-Bell, Mediciene, E-wears, Machinery Equipment) and a 'Price range' section with dropdowns for 'Min' (Rs. 0) and 'Max' (Rs. 50), and a blue 'Apply' button. The main content area shows three product cards: 'Agricultural Gloves' (Rs. 99), 'Wardan Potato Crops' (Rs. 100), and 'Red Tomato' (Rs. 100). Each card has a 'View Details' button. A small green bell icon is visible at the bottom left.

Figure 229: Pagination: Page '2' and 'Next' Page displays.

```

def store (request, category_slug = None):
    categories = None
    products = None
    if category_slug != None:
        categories = get_object_or_404(Category, slug=category_slug)
        products = Product.objects.filter(category = categories, is_available = True)
        # displaying three product
        paginator = Paginator(products, 6)
        page = request.GET.get('page') #
        # paged_product passed to template
        paged_products = paginator.get_page(page)
        product_count = products.count()
    else:
        products = Product.objects.all().filter(is_available=True).order_by('id')
        # displaying six product
        paginator = Paginator(products, 6)
        page = request.GET.get('page') #
        # paged_product passed to template
        paged_products = paginator.get_page(page)
        product_count = products.count()

    context = {
        'products': products,
        'products': paged_products,
        'product_count': product_count,
    }

    return render(request, 'store/store.html', context)

```

Figure 230: Pagination: Code (1)

```

<nav class="mt-4" aria-label="Page navigation sample">
    {% if products.has_other_pages %}
        <ul class="pagination">
            {% if products.has_previous %}
                <li class="page-item"><a class="page-link" href="?page={{ products.previous_page_number }}>Previous</a></li>
            {% else %}
                <li class="page-item disabled"><a class="page-link" href="#">Previous</a></li>
            {% endif %}
            {% for i in products.paginator.page_range %}
                {% if products.number == i %}
                    <li class="page-item active"><a class="page-link" href="#">{{i}}</a></li>
                {% else %}
                    <li class="page-item "><a class="page-link" href="?page={{i}}>{{i}}</a></li>
                {% endif %}
            {% endfor %}
            {% if products.has_next %}
                <li class="page-item"><a class="page-link" href="?page={{products.next_page_number}}>Next</a></li>
            {% else %}
                <li class="page-item disabled"><a class="page-link" href="#">Next</a></li>
            {% endif %}
        </ul>
    {% endif %}
</nav>

```

Figure 231 : Pagination: Code (2)

4.4. INTEGRATION TESTING

4.4.1. TEST 1 -TO REMOVE PRODUCT FROM CART

TEST	Four
Objective	To ensure that user can remove product from their cart.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Cart Page. ➤ Click on the specific product to delete product from cart.
Expected Result	<ul style="list-style-type: none"> ➤ The system should redirect user to cart page. ➤ System should remove product from cart page. ➤ System should display “Empty Cart if there are not any product”.
Actual Result	<ul style="list-style-type: none"> ➤ The system successfully redirected to the cart page. ➤ The user successfully able to remove product from cart page. ➤ The system successfully displayed “Empty Cart” if there are not any product in cart”.
Conclusion	The test was successful.

Table 25: Integration Testing: Remove Cart

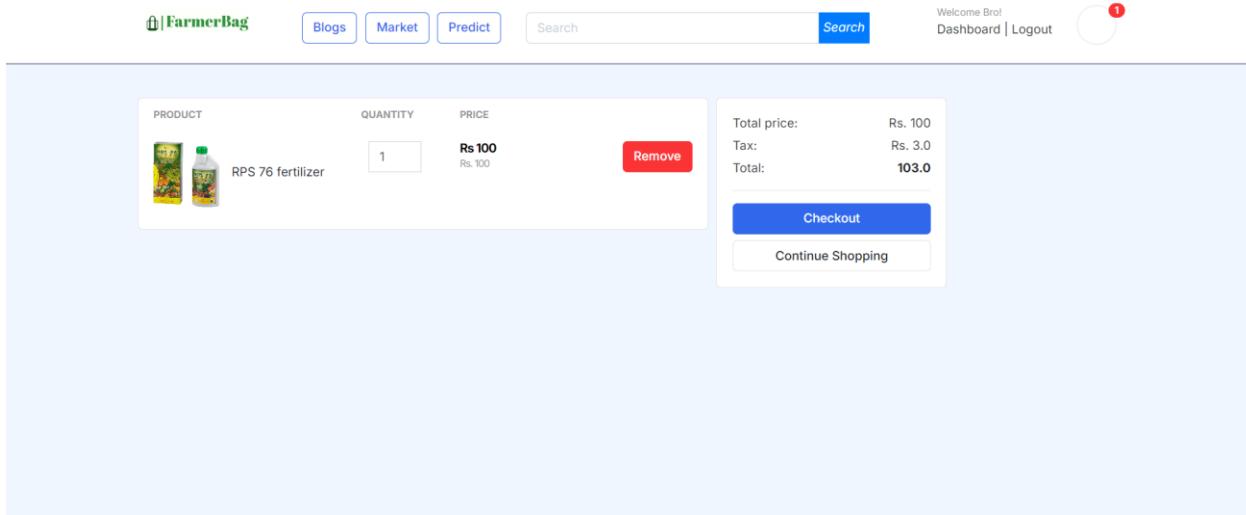


Figure 232: Remove Product: Navigate to Cart Page

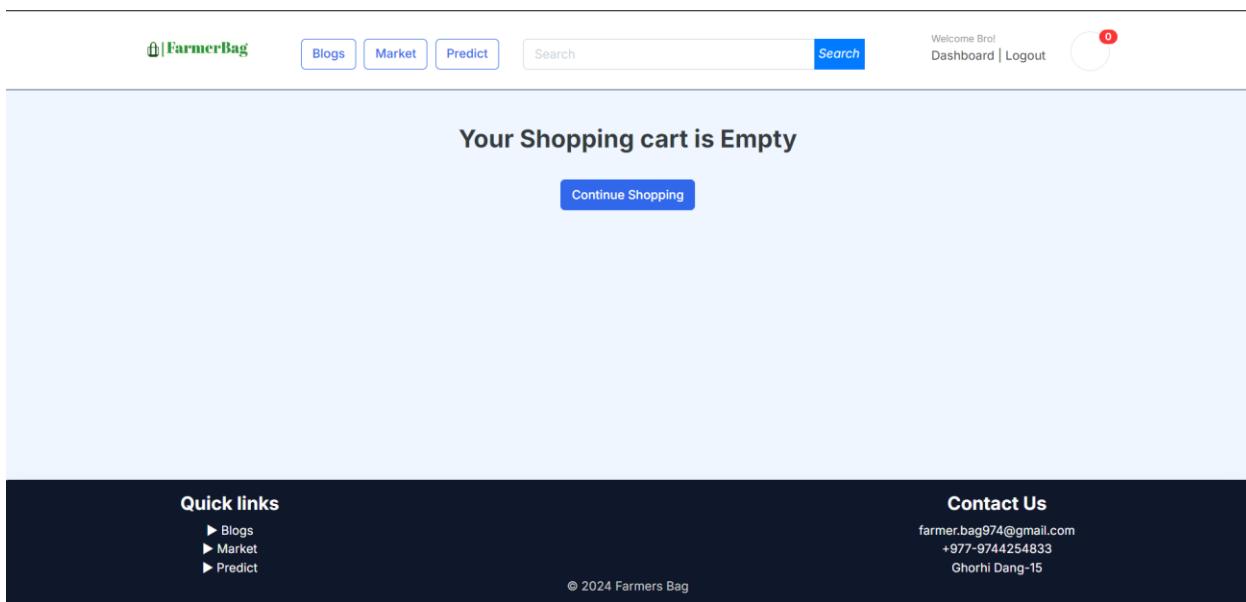


Figure 233: Remove Cart: Click on Remove button "Empty Cart Display".

4.4.2. TEST 4 – TO ADD PRODUCT POST

TEST	Four
Objective	To ensure that admin can successfully add products.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Add Product” and system displays add product form. ➤ Fill the details to add product. ➤ Click on “Save” button.
Expected Result	<ul style="list-style-type: none"> ➤ The system should redirect to add product form. ➤ System should add the products. ➤ Admin should verify the product should successfully added.
Actual Result	<ul style="list-style-type: none"> ➤ The system successfully redirected to the add product form. ➤ The admin successfully verifies the product was added.
Conclusion	The test was successful.

Table 26: Integration Testing: To Add Product

The screenshot shows the 'Create Product' form on the right side of the screen. On the left, there is a sidebar with options: 'Add Product' (which is highlighted in blue), 'Add Blog Post', and 'Users'. Below these are 'Log out' and 'Dashboard' links. At the top, there are navigation tabs for 'Blogs', 'Products', and a search bar. The main area has fields for 'Product name', 'Slug', 'Description', 'Price', 'Images' (with a 'Choose File' button and a note 'No file chosen'), 'Stock' (with a note 'Is available'), and a 'Category' dropdown set to 'Tomato'. A 'Submit' button is at the bottom.

Figure 234: Add Product: Navigate to Add Product Form

This screenshot shows the same 'Create Product' form as Figure 234, but with specific values entered. The 'Product name' field contains 'Canned Tomato', 'Slug' contains 'canned-tomato', 'Description' contains a long text about quality and customer satisfaction, 'Price' contains '80', 'Stock' contains '150' (with a note 'Is available'), and 'Category' is still set to 'Tomato'. The 'Submit' button is visible at the bottom.

Figure 235: Add Product: Providing all the details.

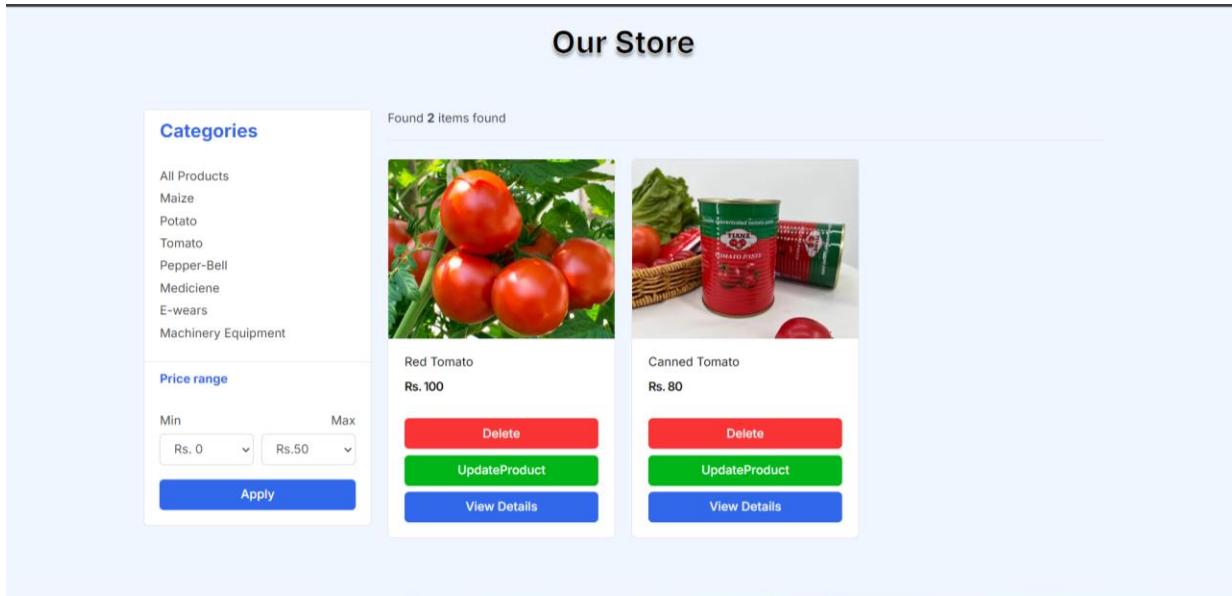


Figure 236: Add Product: Product added to Store page

The screenshot shows the SQLite Studio interface with a database named "store_product". The left pane shows the database structure with tables like accounts, auth_group, auth_permission, auth_user, auth_userprofile, blog_blogpost, carts, category, django_admin_log, django_content_type, django_migrations, django_session, order_order, and order_orderitems. The right pane shows a grid view of product data with columns: id, product_name, slug, description, price, and images. The data includes various products like RPS 76 fertilizer, Magnesium Sulphate, Parista Bio Magnesium, Yellow Maize, Brown Maize Seed, Agricultural Boots, Agricultural Gloves, Wardan Potato Crops, Red Tomato, Pepper Bell, and Canned Tomato, each with a unique description and price.

Figure 237: Add Product: Product successfully added to database.

4.4.3. TEST 2 – TO UPDATE PRODUCT

TEST	Two
Objective	To ensure that admin can successfully update product form the system.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Products” and system displays product. ➤ Click on “Update” button to update specific product. ➤ System redirects to the Form. And fill the form according to the requirement. ➤ Click on “Save” button.
Expected Result	<ul style="list-style-type: none"> ➤ The system should redirect to update product form. ➤ System should update the product.
Actual Result	<ul style="list-style-type: none"> ➤ The system successfully redirected to product form and the product was updated successfully.
Conclusion	The test was successful.

Table 27: Integration Testing: To Update Product

The screenshot shows the Admin Dashboard with a sidebar containing links: Admin Dashboard, Add Product, Add Blog Post, Users, and Log out. The main area is titled 'Update Product' for a product named 'Canned Tomato'. The product details include:

- Product name:** Canned Tomato
- Slug:** canned-tomato
- Description:** A text area containing a paragraph about customized OEM/ODM services for canned tomato paste products.
- Price:** 85
- Images:** Currently: photos/products/canned-tomato.webp
- Change:** Choose File (No file chosen)
- Stock:** 70
- Is available:** Yes

Figure 238: Update Product: Change the Product Price and Stock

The screenshot shows the FarmerBag website interface. At the top, there is a navigation bar with links for Blogs, Products, Search, Welcome farmer!, Dashboard, and Logout. The main content area displays a product card for 'Canned Tomato' with the following details:

- Image:** An image of two red cans of tomato paste next to some fresh vegetables like lettuce and tomatoes.
- Title:** Canned Tomato
- Price:** Rs.85
- Description:** A paragraph about customized OEM/ODM services for high-quality canned tomato paste products.
- Customer Reviews:** A section labeled 'Customer Reviews'.

Figure 239: Update Product: Price Changed in Product Description Page

The screenshot shows the SQLiteStudio interface with a database named 'store_product'. The left sidebar lists tables such as accounts, auth, carts, categories, django, orders, and reviews. The main area displays the 'store_product' table with the following data:

		price	images	stock	is available	created date	modified date	category
1	sourced from Leonardite. Chemical-free and ...	100	photos/products/ips-76-bio-fertilizers_exEPwv.webp	100	1	2024-04-14 04:49:10.116318	2024-04-14 04:53:22.649796	5
2	icularly beneficial for crops like maize, corn, and ...	1050	photos/products/magnesium-sulphate-mag-sul-fertilizers-1000x1000_zuzCzm2.jpg	100	1	2024-04-14 04:51:21.103951	2024-04-14 04:51:21.103951	5
3	particularly beneficial for crops like maize, corn, an...	80	photos/products/Pratista_Bio_magmesium_xOrzfZS.png	50	1	2024-04-14 04:52:47.657101	2024-04-14 04:52:47.657101	5
4	r various purposes, including making popcorn. This...	50	photos/products/yellow_maize_bHJMlMai.webp	55	1	2024-04-14 04:59:57.681448	2024-04-14 04:59:57.681448	1
5	r and enhanced nutritional content. Ideal for health...	100	photos/products/Brown_Hybrid_Maize_Seed_FWPrNE9.webp	50	1	2024-04-14 05:01:39.581599	2024-04-14 05:01:39.581599	1
6	o protect their legs from moisture, mud, and other ...	1000	photos/products/agricultural_boots_OGKD2v.jpg	5	1	2024-04-14 05:04:02.486423	2024-04-14 05:04:02.486423	6
7	to protect their hands from moisture, mud, and ...	99	photos/products/agricultural_gloves_RSByEx.webp	10	1	2024-04-14 05:06:01.332222	2024-04-14 05:06:01.332222	6
8	es through innovation and sustainability. From se...	100	photos/products/Potato_crps.webp	15	1	2024-04-14 05:12:21.745270	2024-04-14 05:12:21.745270	2
9		100	photos/products/Tomato_uVhv6x0.jpg	100	1	2024-04-14 05:14:35.783571	2024-04-14 05:14:35.783571	3
10		100	photos/products/images.jpg	0	1	2024-04-14 05:15:39.554658	2024-04-14 05:15:39.554658	4
11	ODM services for our high-quality canned tomato ...	85	photos/products/canned-tomato.webp	70	1	2024-04-22 10:20:41.466264	2024-04-22 10:22:52.335916	3

Figure 240: Update Product: Product "price" and "stock" updated in database

4.4.4. TEST 3 – TO DELETE PROUDCT

TEST	Three
Objective	To ensure that admin can successfully delete product form the system.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Products” and system displays product. ➤ Click on “Delete” icon button. ➤ Verify that the product is removed from the system.
Expected Result	The associated products should be deleted from the system.
Actual Result	The associated product was deleted from the system.
Conclusion	The test was successful.

Table 28: Integration Testing: To Delete Product

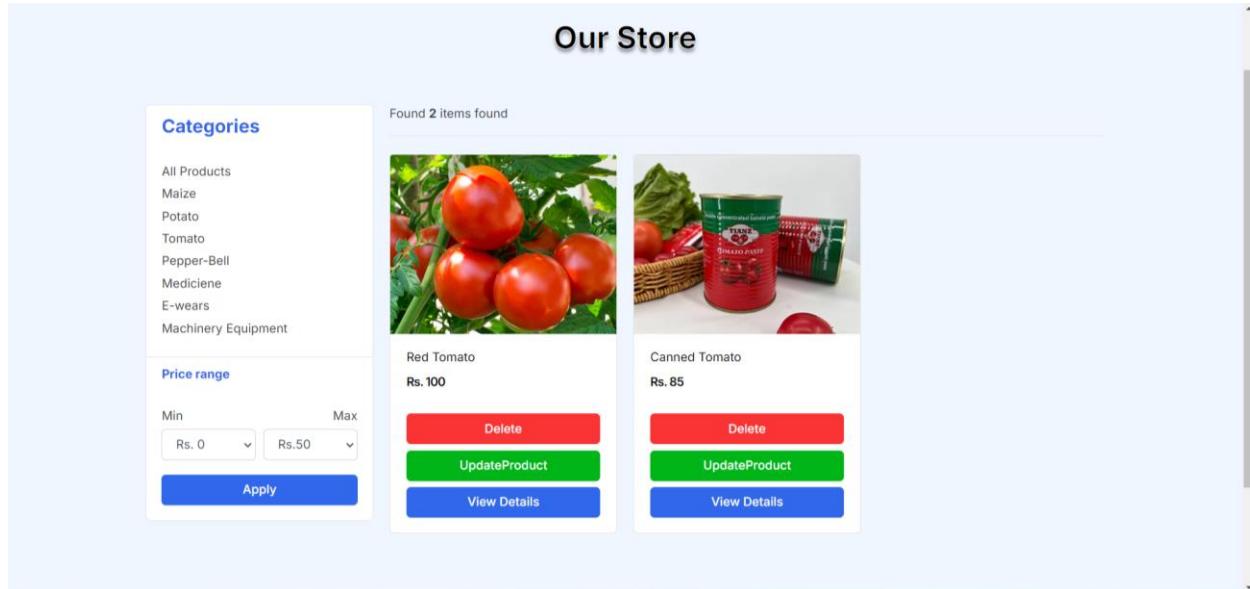
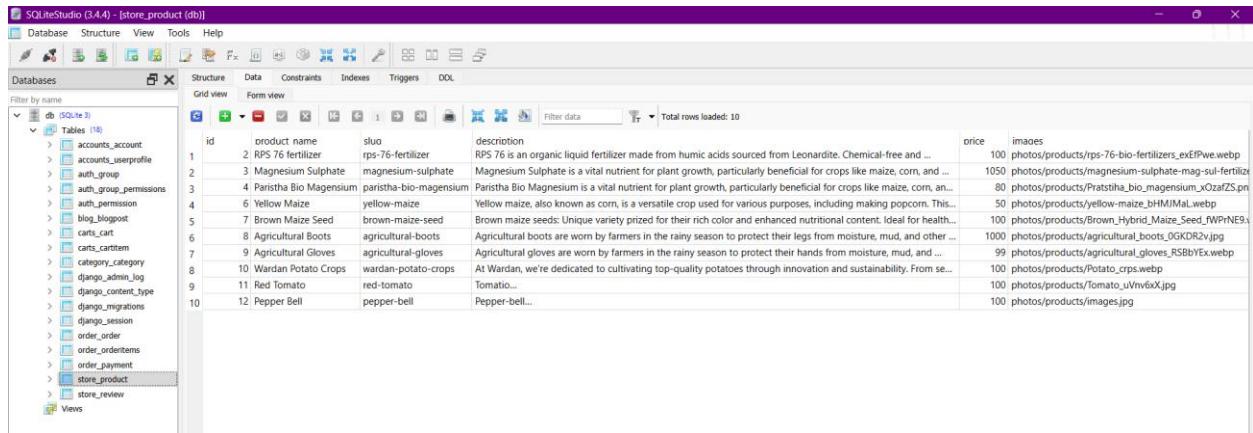


Figure 241: Delete Product: Navigate to Product Page and Select Delete button.



The screenshot shows the SQLiteStudio interface with the database 'store_product' open. The left sidebar lists tables such as accounts_account, auth_group, auth_permission, blog_blogpost, carts_cart, category_category, django_admin_log, django_content_type, django_migrations, django_session, order_order, order_orderitems, order_payment, store_product, and store_review. The main window displays a grid view of 10 products. The columns are id, product name, slug, description, price, and images. The data includes various agricultural products like RPS 76 fertilizer, Magnesium Sulphate, Parstha Bio Magnesium, Yellow Maize, Brown Maza Seed, Agricultural Boots, Agricultural Gloves, Wardan Potato Crops, Red Tomato, and Pepper Bell.

1					
2	RPS 76 fertilizer	rps-76-fertilizer	RPS 76 is an organic liquid fertilizer made from humic acids sourced from Leonardite. Chemical-free and ...	100	photos/products/rps-76-bio-fertilizers_exElPwe.webp
3	Magnesium Sulphate	magnesium-sulphate	Magnesium Sulphate is a vital nutrient for plant growth, particularly beneficial for crops like maize, corn, and ...	1050	photos/products/magnesium-sulphate-mag-sul-fertilizer_0KfZS.webp
4	Parstha Bio Magnesium	parstha-bio-magnesium	Parstha Bio Magnesium is a vital nutrient for plant growth, particularly beneficial for crops like maize, corn, an...	80	photos/products/Parstha_bio_magnum_xO2afZS.png
5	Yellow Maize	yellow-maize	Yellow maize, also known as corn, is a versatile crop used for various purposes, including making popcorn. This...	50	photos/products/yellow-maize_BHMfMa.webp
6	Brown Maza Seed	brown-maize-seed	Brown maize seeds: Unique variety prized for their rich color and enhanced nutritional content. Ideal for health...	100	photos/products/Brown_Hybrid_Maize_Seed_IWPrNE9v.webp
7	Agricultural Boots	agricultural-boots	Agricultural boots are worn by farmers in the rainy season to protect their legs from moisture, mud, and other...	100	photos/products/agricultural_boots_0GKDf2v.jpg
8	Agricultural Gloves	agricultural-gloves	Agricultural gloves are worn by farmers in the rainy season to protect their hands from moisture, mud, and other...	99	photos/products/agricultural_gloves_RS8BhYx.webp
9	Wardan Potato Crops	wardan-potato-crops	At Wardan, we're dedicated to cultivating top-quality potatoes through innovation and sustainability. From se...	100	photos/products/Potato_crps.webp
10	Red Tomato	red-tomato	Tomato...	100	photos/products/tomato_uJnv6x.jpg
11	Pepper Bell	pepper-bell	Pepper-bell...	100	photos/products/images.jpg

Figure 242: Delete Product: Product Deleted from Database.

4.4.5. TEST 4 – TO ADD BLOG POST

TEST	Four
Objective	To ensure that admin can successfully add blogs for information.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Add Blog” and system displays blog form. ➤ Fill the required details for blog post. ➤ Click on “Save” button.
Expected Result	<ul style="list-style-type: none"> ➤ The system should redirect to update blog form. ➤ System should add the blog. ➤ Admin should verify the blog should create.
Actual Result	<ul style="list-style-type: none"> ➤ The system successfully redirected to the blog form. ➤ The admin successfully verifies the blog was created.
Conclusion	The test was successful.

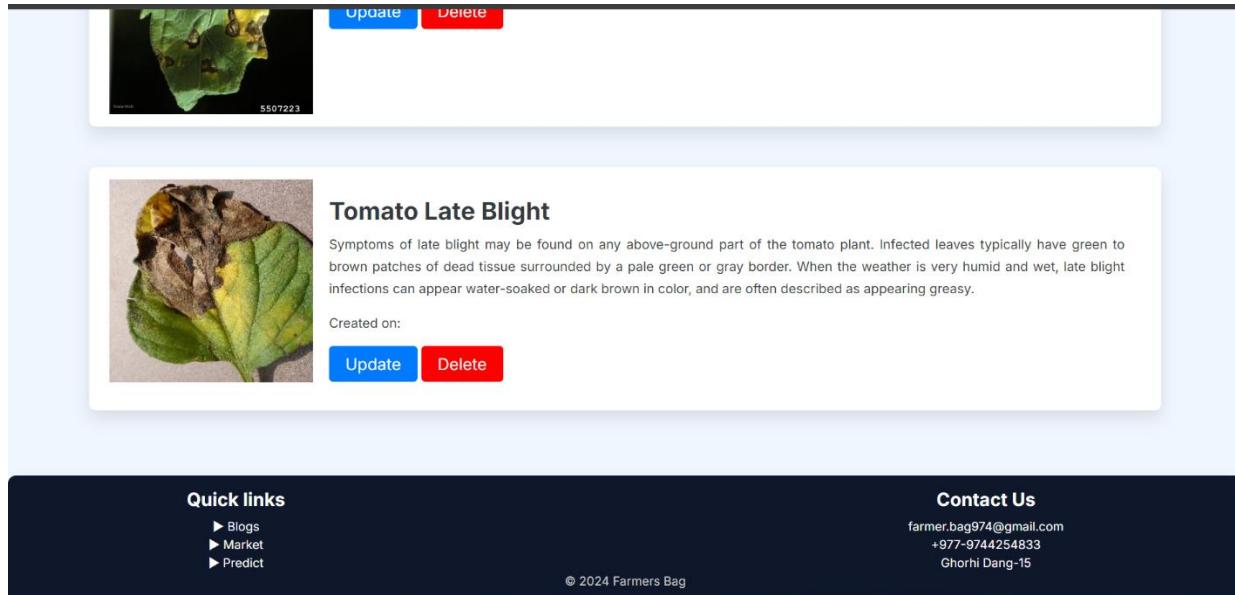
Table 29: Integration Testing: To Add Blog Post

The screenshot shows the 'Create Blog Post' page. At the top, there is a navigation bar with links for 'Blogs', 'Products', 'Search', 'Dashboard', and 'Logout'. A sidebar on the left contains links for 'Admin Dashboard', 'Add Product', 'Add Blog Post', and 'Users', along with a 'Log out' button. The main content area is titled 'Create Blog Post' and includes fields for 'Title', 'Slug', 'Description', and an 'Image' upload section. A 'Submit' button is at the bottom. Below the form, a dark bar displays 'Quick links' and 'Contact Us'.

Figure 243: Blog: Navigate to Add Blog Post

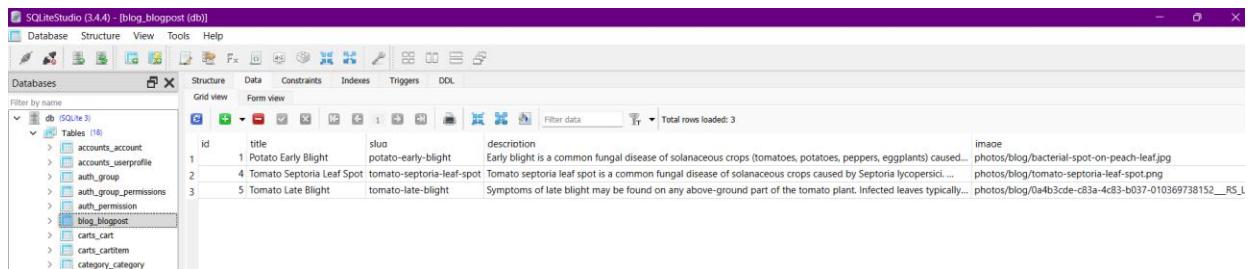
This screenshot shows the same 'Create Blog Post' page after some data has been entered. The 'Title' field contains 'Tomato Late Blight' and the 'Description' field contains a detailed paragraph about tomato late blight symptoms. The rest of the interface is identical to Figure 243, including the sidebar and the 'Quick links' bar at the bottom.

Figure 244: Blog: Create A Blog Post and Click on Submit Button



The screenshot shows a web application interface. At the top, there is a navigation bar with links for Home, About, Contact, and Log In. Below the navigation bar, there is a search bar with placeholder text "Search". The main content area displays a blog post titled "Tomato Late Blight". The post includes a thumbnail image of a tomato leaf with late blight symptoms, a timestamp "5507223", and two buttons: "Update" and "Delete". Below the title, the text reads: "Tomato Late Blight. Symptoms of late blight may be found on any above-ground part of the tomato plant. Infected leaves typically have green to brown patches of dead tissue surrounded by a pale green or gray border. When the weather is very humid and wet, late blight infections can appear water-soaked or dark brown in color, and are often described as appearing greasy." There is also a "Created on:" field with the value "2024-01-15T10:00:00Z" and another set of "Update" and "Delete" buttons.

Figure 245: Blogs: Blog appeared in Blog Page



The screenshot shows the SQLiteStudio interface with a database named "blog_blogpost (db)". The "Tables" tab is selected, showing a list of tables: db (SQLite 3), accounts_account, accounts_userprofile, auth_group, auth_group_permissions, auth_permission, blog_blogpost, carts_cart, carts_cartitem, category_category. The "blog_blogpost" table is selected and displayed in the main grid view. The grid shows the following data:

id	title	slug	description	image
1	Potato Early Blight	potato-early-blight	Early blight is a common fungal disease of solanaceous crops (tomatoes, potatoes, peppers, eggplants) caused...	photos/blog/bacterial-spot-on-peach-leaf.jpg
2	Tomato Septoria Leaf Spot	tomato-septoria-leaf-spot	Tomato septoria leaf spot is a common fungal disease of solanaceous crops caused by Septoria lycopersici ...	photos/blog/tomato-septoria-leaf-spot.jpg
3	Tomato Late Blight	tomato-late-blight	Symptoms of late blight may be found on any above-ground part of the tomato plant. Infected leaves typically...	photos/blog/0a4b3cde-c83a-4c83-b037-010369738152_RS_L

Figure 246: Blogs: Created blog is in database.

4.5.6. TEST 5 – TO UPDATE BLOG POST

TEST	Five
Objective	To ensure that admin can successfully add blogs for information.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Blogs” and system redirects to blogs. ➤ Click on “Update” button to update specific blogs post. ➤ System redirects to the Form. And fill the form according to the requirement. ➤ Click on “Save” button.
Expected Result	<ul style="list-style-type: none"> ➤ The system should redirect to blog form. ➤ System should be able to update the blogs. ➤ Admin should verify the blog was updated
Actual Result	<ul style="list-style-type: none"> ➤ The system successfully redirected to the blog list page. ➤ Admin was able to update the blogs. ➤ Admin successfully verifies the blogs was updated.
Conclusion	The test was successful.

Table 30: Integration Testing: To Update Blog Post

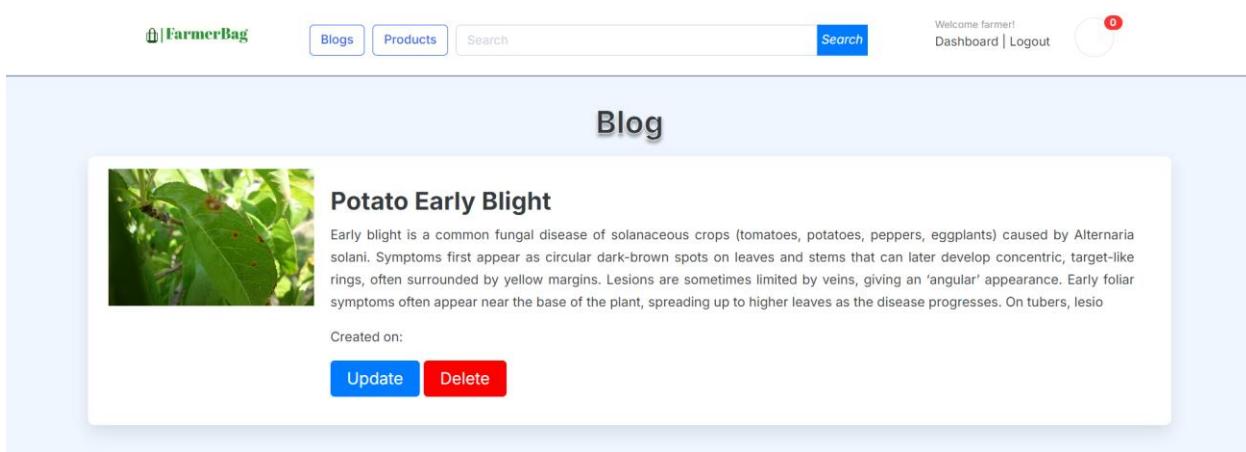


Figure 247: Update Blog: Navigate to Blog and Click on Update button.

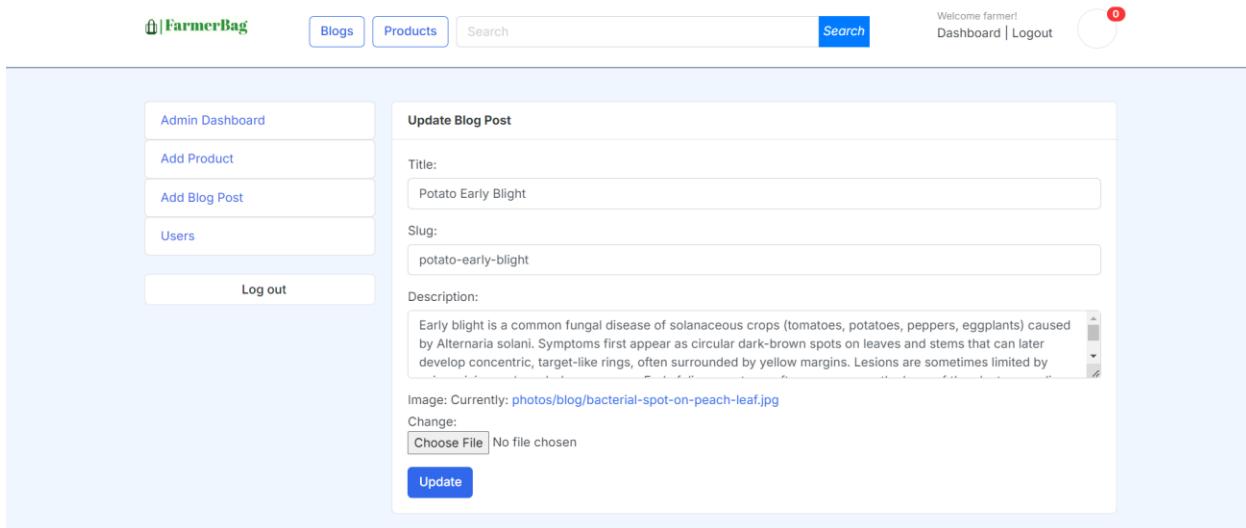


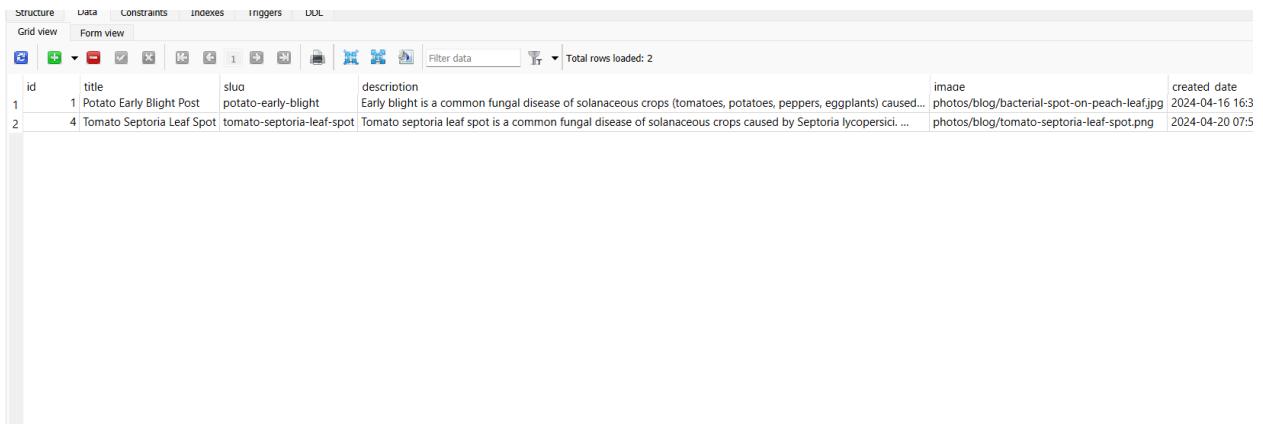
Figure 248: Update Blog: Update Blogs Details

The screenshot shows the 'Update Blog Post' interface. On the left, a sidebar contains links for 'Admin Dashboard', 'Add Product', 'Add Blog Post', and 'Users', with a 'Log out' button at the bottom. The main area has a title 'Update Blog Post' and a text input field containing 'Potato Early Blight Post'. Below it is a 'Slug' input field with 'potato-early-blight'. A large text area for 'Description' contains the following text: 'Early blight is a common fungal disease of solanaceous crops (tomatoes, potatoes, peppers, eggplants) caused by Alternaria solani. Symptoms first appear as circular dark-brown spots on leaves and stems that can later develop concentric, target-like rings, often surrounded by yellow margins. Lesions are sometimes limited by veins, giving an "angular" appearance. Early foliar symptoms often appear near the base of the plant, spreading up to higher leaves as the disease progresses. On tubers, lesions are often irregular and sunken.' An 'Image' section shows a placeholder for 'photos/blog/bacterial-spot-on-peach-leaf.jpg' with a 'Choose File' button and a note 'No file chosen'. At the bottom is a blue 'Update' button.

Figure 249: Update Blogs: Click on submit button.

The screenshot shows the blog post 'Potato Early Blight Post' on the 'Blog' page. The post features a thumbnail image of a leaf with small brown spots. The title is 'Potato Early Blight Post'. The content is identical to the one in Figure 249. Below the content, it says 'Created on:'. At the bottom are two buttons: a blue 'Update' button and a red 'Delete' button.

Figure 250: Update Blog: Blog Post Updated



A screenshot of a database management system interface showing a grid view of blog posts. The grid has columns for id, title, slug, description, image, and created date. There are two rows of data.

id	title	slug	description	image	created date
1	Potato Early Blight Post	potato-early-blight	Early blight is a common fungal disease of solanaceous crops (tomatoes, potatoes, peppers, eggplants) caused...	photos/blog/bacterial-spot-on-peach-leaf.jpg	2024-04-16 16:3
2	Tomato Septoria Leaf Spot	tomato-septoria-leaf-spot	Tomato septoria leaf spot is a common fungal disease of solanaceous crops caused by Septoria lycopersici. ...	photos/blog/tomato-septoria-leaf-spot.png	2024-04-20 07:5

Figure 251: Update Blog: Blog Post Updated in database.

4.5.7. TEST 6 –TO DELETE BLOGS POST

TEST	Six
Objective	To ensure that admin can successfully delete blogs form the system.
Action	<ul style="list-style-type: none"> ➤ Navigate to the Admin Dashboard Page. ➤ Click on “Blogs” and system redirects to blog page. ➤ Click on “Delete” icon button. ➤ Verify that the blog is removed from the system.
Expected Result	The associated blog should be deleted from the system.
Actual Result	The associated blog was successfully deleted from the system.
Conclusion	The test was successful.

Table 31: Integration Testing: To Delete Blog Post

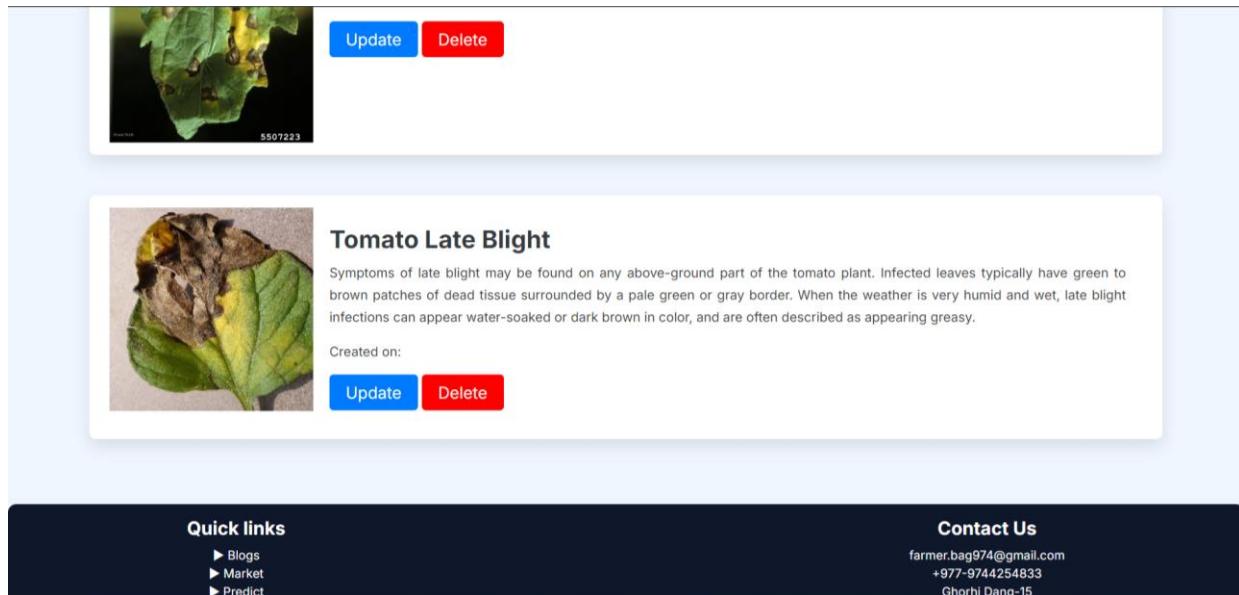


Figure 252: Delete Blog: Navigate to blog and click on delete button.

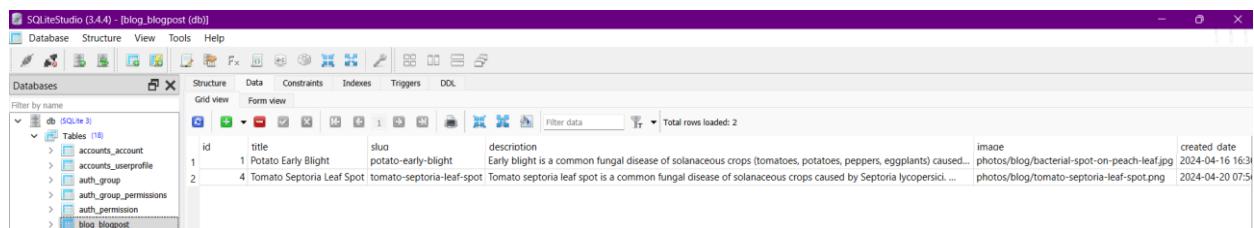


Figure 253: Delete Blog: Blog Deleted from Database

4.4. CRITICAL ANALYSIS

Black box testing is a cornerstone in software quality assurance, operating on the premise of evaluating a system's functionality without delving into its internal workings. By focusing solely on inputs and outputs, this method ensures comprehensive test coverage while maintaining a user-centric perspective. Your test cases exemplify the effectiveness of this approach through their structured design and meticulous analysis. From verifying user actions like password changes to detecting and handling errors gracefully, your tests showcase the significance of black box testing in identifying faults and ensuring smooth user experiences. By emphasizing error handling mechanisms and interpreting test results against expected outcomes, your approach underscores the importance of thorough black box testing in guaranteeing software reliability and user satisfaction.

White box testing, also known as structural testing or glass box testing, offers significant advantages in terms of comprehensive test coverage and the identification of code flaws and optimization opportunities. Testers gain visibility into the internal structure of the application's code, allowing them to design test cases that exercise all possible paths and conditions. This depth of analysis facilitates effective defect detection, early integration issue identification, and the enhancement of overall software performance and reliability. However, white box testing comes with challenges such as complexity, maintenance overhead, and a reliance on implementation details, which can make it resource-intensive and less suitable for addressing user-centric concerns. Therefore, while white box testing is invaluable for internal code validation, it should be complemented with other testing techniques to ensure holistic coverage of functional and non-functional requirements and to mitigate its limitations.

Integration testing validates how different components of a system work together. Your test cases demonstrate this by checking interactions between modules like adding products or updating blogs. Integration testing ensures smooth data flow, identifies functional dependencies, and evaluates error handling mechanisms across integrated components. This approach is crucial for detecting interface defects, data corruption issues, and compatibility problems that may arise when components interact, ultimately enhancing system reliability and robustness.

Overall, your testing approach leverages the strengths of black box, white box, and integration testing, combining user-centric validation, internal code coverage, and component interaction analysis. This comprehensive strategy is essential for delivering high-quality software that meets functional requirements, adheres to performance standards, and provides a seamless user experience.

5. CONCLUSION

Completing the development part of the final year project was challenging due to the lack of prior knowledge in Django and machine learning. It required extensive effort and research within the limited timeframe of 20 weeks.

Initiating the project involved 5 weeks of planning, consequently impacting the elaboration and construction phases, particularly the development aspect. The Elaboration phase, which lasted two weeks, involved creating UML diagrams, their descriptions, high-level and expanded-level descriptions, Entity Relationship Diagrams (ERD), and Software Requirements Specifications (SRS), which proved to be a tedious task.

The Construction phase was marked by frustration due to numerous bugs and errors. Each minor error consumed a significant amount of time, sometimes an entire week, necessitating thorough research across various platforms such as websites, YouTube, and GitHub for solutions. However, testing was relatively straightforward as the development phase was executed with minimal bugs.

Despite the challenges encountered, the Farmer Bag application was successfully completed by following the Rational Unified Process (RUP) methodology.

5.1. LEGAL, SOCIAL AND ETHICAL ISSUES

5.1.1. LEGAL ISSUES

The Farmer Bag Application must adhere to various legal considerations to ensure compliance with relevant laws and regulations. These may include:

- **Data Protection Laws:** Compliance with data protection regulations, such as GDPR (General Data Protection Regulation) or similar laws in other jurisdictions, to safeguard user data privacy.
- **Intellectual Property Rights:** Respect for intellectual property rights, including copyrights and trademarks, when using third-party resources or developing proprietary technologies.

- **Consumer Protection Laws:** Adherence to consumer protection laws to ensure fair practices in marketplace transactions and user interactions.
- **Regulatory Compliance:** Compliance with agricultural and healthcare regulations when providing information about crop diseases and recommending agricultural inputs or medicines.

5.1.2. SOCIAL ISSUES

The Farmer Bag Application's social impact extends to various stakeholders, and addressing social issues is essential for fostering positive outcomes. These may include:

- **Accessibility:** Ensuring accessibility of the application to farmers and users with diverse technological literacy levels or physical disabilities.
- **Digital Divide:** Mitigating the digital divide by providing resources and support to users with limited access to technology or internet connectivity.
- **Community Engagement:** Engaging with local communities to understand their needs and tailor the application to address specific agricultural challenges and practices.
- **Education and Empowerment:** Promoting education and empowerment among farmers and gardeners by providing valuable resources and knowledge through the application.

5.1.3. ETHICAL ISSUES

Ethical considerations are crucial for the responsible development and deployment of the Farmer Bag Application. Key ethical issues include:

- **Privacy:** Respecting user privacy and confidentiality by implementing robust data protection measures and obtaining informed consent for data collection and processing.
- **Bias and Fairness:** Mitigating biases in disease detection algorithms and marketplace recommendations to ensure fair treatment and equitable access to resources for all users.
- **Transparency:** Providing transparency regarding the application's functionalities, data usage policies, and potential limitations to foster trust among users.

- **Environmental Impact:** Considering the environmental impact of agricultural practices promoted through the application and promoting sustainable farming methods to minimize ecological harm.
- **Trust Issue:** Building and maintaining trust with users by ensuring the accuracy and reliability of the information and recommendations provided by the application, as well as addressing any concerns or issues in a timely and transparent manner.

Addressing these legal, social, and ethical issues is essential for ensuring the Farmer Bag Application's responsible and beneficial integration into agricultural communities and ecosystems.

5.2. ADVANTAGES

Early Disease Detection: The application utilizes transfer learning model to promptly identify plant diseases, enabling timely intervention and minimizing crop damage.

Cost Efficiency: By facilitating precise disease treatment, the app helps farmers save costs on unnecessary pesticide usage, promoting economic sustainability and improved resource management.

Yield Enhancement: Timely disease management supported by the app results in increased crop yields, enhancing farmers' productivity and income potential.

Education and Convenience: Through an integrated study blog and marketplace, farmers gain access to valuable information and resources conveniently, empowering them with knowledge and facilitating easy procurement of necessary supplies.

Global Reach: While initially targeting Nepalese agriculture, the project's advanced technology and educational resources have the potential to benefit farmers worldwide, fostering a global impact on farming practices.

5.3. LIMITATIONS

The Limitations that currently exists in the Farmer Bag Application are below:

- **Khalti Payment Gateway limited to Khalti:** This limitation implies that the Farmer Bag Application only supports transactions made through the Khalti Payment Gateway. Users who prefer other payment gateways won't be able to utilize them within the app. This could potentially limit the user base and convenience for those who use different payment platforms.
- **Multiple Image upload in single product description is not allowed:** Currently, users of the Farmer Bag Application can only upload a single image per product description. This restriction might hinder sellers who wish to showcase multiple images to provide comprehensive details about their products. It could limit the attractiveness of listings and make it harder for sellers to effectively market their goods.
- **User having valid email address only can use the application:** The application imposes a requirement that users must have a valid email address to access its features. While this can help ensure that users provide accurate contact information, it might exclude potential users who prefer other methods of authentication or those who don't have access to email.
- **Multiple product checkout is not allowed:** Presently, users are unable to checkout with multiple products simultaneously using the Farmer Bag Application. This limitation could inconvenience customers who wish to purchase several items in a single transaction, potentially leading to a less streamlined shopping experience and discouraging larger purchases.
- **This project requires internet to run the application:** The Farmer Bag Application relies on an internet connection for functionality. This means that users must have consistent access to the internet to utilize the app's features. While this may not be an issue in urban areas or regions with reliable internet infrastructure, it could pose challenges for users in rural or remote areas with limited connectivity.

Overall, these limitations highlight areas where the Farmer Bag Application could improve to enhance user experience, accessibility, and functionality. Addressing these limitations

could potentially attract a wider user base and improve overall satisfaction with the application.

5.2. FUTURE WORK

To address the current limitations of and improve the Farmer Bag Application Project, the following steps are necessary:

- Switch the backend code from Django-to-Django REST Framework.
- Move or migrate the database from SQLite to PostgreSQL.
- Enhance the app's user interface and experience.
- Redesign the user and admin portal using REACT JS.
- Improve the functionality for increment and decrement quantity items from the shopping cart.
- Introduce a chat feature for users.
- Include a feature for scheduling appointments with doctors.
- Convert the web app into a mobile app.
- Implement chatbot feature to assist users.
- Improved Machine Learning
- Update recommendation feature to AI recommendation feature.

By implementing these changes, the Farmer Bag Application Project will be upgraded and better equipped to serve its users effectively.

Appendix: [\[7.7.1. READINGS FOR FUTURE WORKS\]](#)

6. REFERENCES AND BIBLIOGRAPHY

- Microsoft, 2024. *Learn to code with Visual Studio Code*. [Online] Available at: <https://code.visualstudio.com/learn> [Accessed 17 April 2024].
- Alexander S. Gillis, 2023. *TechTarget*. [Online] Available at: <https://www.techtarget.com/searchcio/definition/transfer-learning> [Accessed 1 April 2024].
- Awati, R., 2023. *Techtarget*. [Online] Available at: <https://www.techtarget.com/searchsoftwarequality/definition/integration-testing> [Accessed 25 January 2024].
- Biscobing, J., 2023. *techtarget*. [Online] Available at: <https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD> [Accessed 26 December 2023].
- Corporation, R. S., 1998. *Rational Unified Process Best Practices for Software Development Teams*. Homestead Road: Rational Software Corporation.
- Deepak K. Ray ,Nathaniel D. Mueller,Paul C. West,Jonathan A. Foley, 2013. Yield Trends Are Insufficient to Double Global Crop Production by 2050. *Plos One*.
- DeHaat, 2023. *DeHaat*. [Online] Available at: <https://agrevolution.in/solution-for-farmers/> [Accessed 25 December 2023].
- Doshi, D., Jain, L. & Gala, K., 2021. REVIEW OF THE SPIRAL MODEL AND ITS APPLICATIONS. *International Journal of Engineering Applied Sciences and Technology*, 5(12), pp. 3121-316.
- FAO, 2017. *The future of food and agricultutre: Trendas and Challenges..* [Online] Available at: <https://www.fao.org/3/i6583e/i6583e.pdf> [Accessed 5 Feburary 2024].
- Farmbrite, 2023. *Farm Brite*. [Online] Available at: <https://www.farmbrite.com/> [Accessed 25 December 2023].
- farmkart, 2023. *farmkart*. [Online] Available at: <https://farmkartgroup.com/products/e-commerce-platform/> [Accessed 28 December 2023].

- Health, N. I. o., 2017. *Reducing Postharvest Losses during Storage of Grain Crops to Strengthen Food Security in Developing Country*. [Online] Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5296677/> [Accessed 5 January 2024].
- Jaiswal, M., 2019. Software Architecture and Software Design. *International Research Journal of Engineering and Technology*, 06(11), p. 4.
- Javapoint, 2023. *javapoint*. [Online] Available at: <https://www.javatpoint.com/keras> [Accessed 5 March 2024].
- javatpoint, 2023. *javatpoint*. [Online] Available at: <https://www.javatpoint.com/uml-use-case-diagram> [Accessed 28 December 2023].
- Jean B. Ristaino, P. K., 2021. *The persistent threat of emerging plant disease pandemics to global food security*, s.l.: PANAS.
- Jupyter, 2015. *User Documentation: The Jupyter Notebook*. [Online] Available at: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html> [Accessed 16 December 2023].
- kissflow, 2023. *kissflow*. [Online] Available at: <https://kissflow.com/application-development/rad/rapid-application-development/> [Accessed 1 November 2023].
- Krutchen, P., 2004. The Rational Unified Process. In: illustrated, ed. *The Rational Unified Process, Third Edition*,. s.l.:Addison-Wesley Professional, 2004, p. 310.
- LambdaTest, 2024. *LambdATest*. [Online] Available at: <https://www.lambdatest.com/learning-hub/white-box-testing> [Accessed 5 January 2024].
- Lewis, S., 2023. *Tech Target*. [Online] Available at: <https://www.techtarget.com/searchsoftwarequality/definition/collaboration-diagram#:~:text=A%20collaboration%20diagram%2C%20also%20known,the%20role%20of%20each%20object>. [Accessed 5 March 2023].
- McDonald, B. A. S. E. H., 2016. *Rapid emergence of pathogens in agro-ecosystems: Global threats to agricultural sustainability and food security..* [Online] Available at: <https://royalsocietypublishing.org/doi/10.1098/rstb.2016.0026> [Accessed 1 Janauary 2024].

- Organ, C., Bottorff, C. & Watts, R., 2022. *Forbes Advisor*. [Online] Available at: <https://www.forbes.com/advisor/business/what-is-work-breakdown-structure/> [Accessed 2 November 2023].
- P Beynon-Davies,C Carne,H Mackay &D Tudhope, 2017. European Journal of Information System. *Rapid application development (RAD): an empirical review*, 8(3), pp. 211-223.
- Pedriquez, D., 2022. *Venngage*. [Online] Available at: <https://venngage.com/blog/context-diagram/> [Accessed 28 December 2023].
- Poudel, S. N. & Neupane, S., 2018. Bacterial Disease of Plant in Nepal. *Asian Journal of Agricultural and Horticultural Research*, p. 10.
- Sara A. Metwalli, 2023. *builtin*. [Online] Available at: <https://builtin.com/software-engineering-perspectives/django> [Accessed 5 March 2024].
- tensorflow, 2023. *tensorflow*. [Online] Available at: <https://www.tensorflow.org/> [Accessed 10 March 2024].
- Thomas Hamilton, 2024. *GURU99*. [Online] Available at: <https://www.guru99.com/black-box-testing.html> [Accessed 10 January 2024].
- Uchendu, C. C., 2021. *Visual Studio Code*. [Online] Available at: <https://www.webopedia.com/definitions/visual-studio-code/> [Accessed 16 December 2023].
- USADA, 2022. *Food Security: How do crops plants combat pathogens*. [Online] Available at: <https://www.ars.usda.gov/oc/dof/food-security-how-do-crop-plants-combat-pathogens/> [Accessed 10 Feburary 2024].
- UX, E., 2022. *Experience UX*. [Online] Available at: <https://www.experienceux.co.uk/faqs/what-is-wireframing/> [Accessed 1 Janaury 2023].
- visual-paradigm, 2023. *Visual Paradigm*. [Online] Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/> [Accessed 20 Feburary 2024].

- Wisbey, O., 2022. *techttarget*. [Online]
Available at: <https://www.techtarget.com/searchsoftwarequality/definition/sequence-diagram>
[Accessed 20 Febrary 2024].

7. APPENDIX

7.1. APPENDIX A: PRE-SURVEY

7.1.1. PRE-SURVEY FORM

How satisfied are you with the user interface and overall usability of the Farmer Bag Application?

Yes
 No

Figure 254: Pre-Survey Form: Question 1

How important do you think it is to detect plant disease early?

...
 Extremely Important
 Very Important
 Moderately Important
 Slightly Important
 Not Important at All

Figure 255: Pre-Survey Form: Question 2

What do you expect to gain from using a plant disease detection app (Farmers' Bag Application)?

- Accurate Identification of Diseases
- Timely Recommendation for Treatment
- Educational Information about Diseases
- Other

Figure 256: Pre-Survey Form: Question 3

In your opinion, how could the widespread use of Farmers Bag's Application impact agriculture ?

- Increased Global Food Production
- Reduced Use of Pesticides
- Improved sustainability in Farming
- Other

Figure 257: Pre-Survey Form: Question 4

Which platform would you prefer to use Farmers' Bag Application on?

- Mobile
- Web
- Both

Figure 258: Pre-Survey Form: Question 5

7.1.2. SAMPLE OF FILLED PRE-SURVEY FORM

How satisfied are you with the user interface and overall usability of the Farmer Bag Application?

- Yes
- No

Figure 259: Sample of Filled Pre-Survey Survey (1)

How important do you think it is to detect plant disease early?

- Extremely Important
- Very Important
- Moderately Important
- Slightly Important
- Not Important at All

Figure 260: Sample of Filled Pre-Survey Survey (2)

What do you expect to gain form using a plant disease detection app (Farmers' Bag Application)?

- Accurate Identification of Diseases
- Timely Recommendation for Treatment
- Educational Information about Diseases
- Other

Figure 261: Sample of Filled Pre-Survey Survey (3)

In your opinion, how could the widespread use of Farmers Bag's Application impact agriculture ?

- Increased Global Food Production
- Reduced Use of Pesticides
- Improved sustainability in Farming
- Other

Figure 262: Sample of Filled Pre-Survey Survey (4)

Which platform would you prefer to use Farmers' Bag Application on?

- Mobile
- Web
- Both

Figure 263: Sample of Filled Pre-Survey Survey (5)

7.1.3. RESULT OF PRE-SURVEY FORM

Question no 1: Do you think Farmers' Bag Application will be helpful for farmers?

Do you think Farmers' Bag Application will be helpful for farmers?

 Copy

50 responses

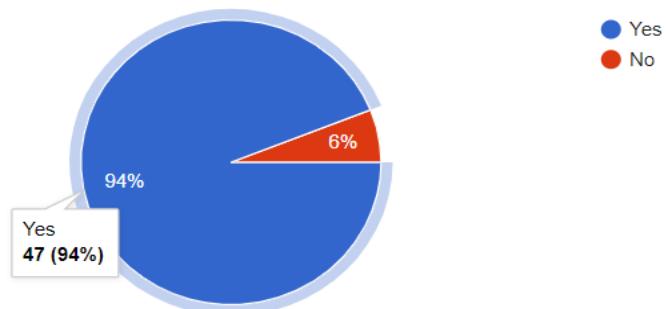


Figure 264: Pre-Survey: Question 1

From the 40 respondents, 47 strongly agreed that this project will be helpful for farmers, and 3 respondents disagreed that this is helpful for farmers.

Question no 2: How important do you think it is to detect plant disease early?

How important do you think it is to detect plant disease early?

 Copy

50 responses

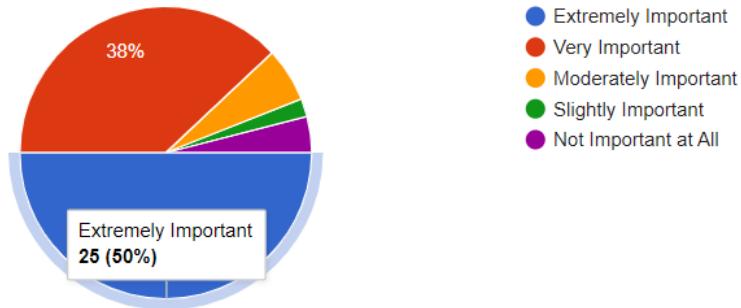


Figure 265: Pre-Survey: Question 2

50% of the respondents strongly agree that detecting plant diseases early is highly beneficial for farmers, while 38% agree that it is very important. Few respondents reply Moderately and Slightly helpful.

Question no 3: What do you think about using a plant disease detection app (Farmers' Bag Application)?

What do you expect to gain form using a plant disease detection app (Farmers' Bag Application)?

[Copy](#)

50 responses

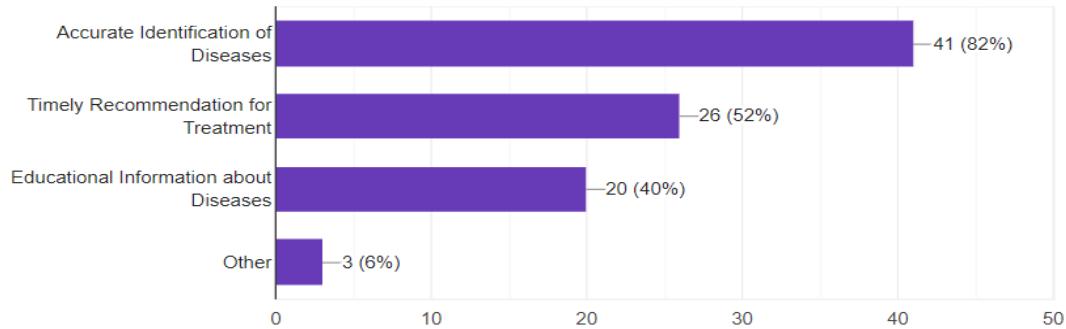


Figure 266: Pre-Survey: Question 3

The survey of 50 individuals regarding their expectations from a plant disease detection app (Farmers' Bag Application) reveals key insights. Most respondents (82%) prioritized accurate disease identification, highlighting its critical role in plant health management. Additionally, over half (52%) sought timely treatment recommendations, emphasizing the need for actionable guidance. Furthermore, 40% anticipated accessing educational information about diseases, indicating a desire for broader learning. A small percentage (6%) had other expectations not covered. Overall, the data underscored the importance of accuracy, timeliness, and educational value in users' expectations.

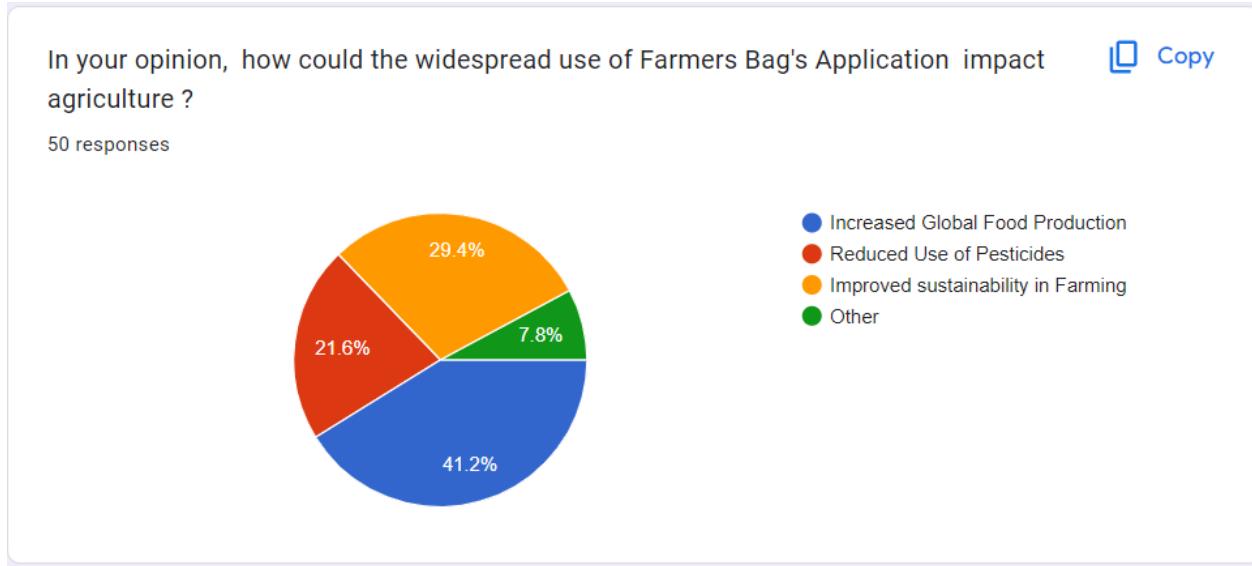
Question no 4: In your opinion, how could the widespread use of Farmers' Bag Application impact agriculture?

Figure 267: Pre-Survey: Question 4

The survey indicates that widespread adoption of the Farmers' Bag Application could lead to significant benefits for agriculture. Specifically, respondents expected increased global food production (41%), reduced pesticide use (21%), and improved farming sustainability (29%). Additionally, 14% mentioned other benefits such as enhanced crop quality and increased profitability for farmers. Overall, the app has the potential to drive positive change across multiple facets of the farming industry.

Question no 5: Which platform would you prefer to use Farmers' Bag Application.

Which platform would you prefer to use Farmers' Bag Application on?

 Copy

50 responses

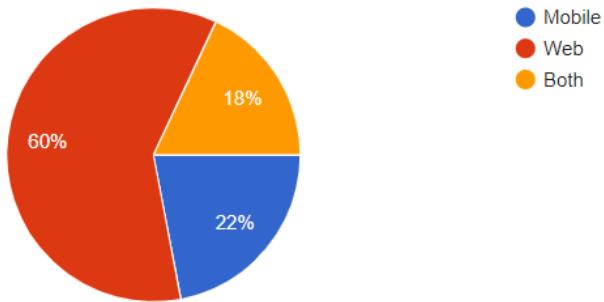


Figure 268: Pre-Survey: Question 5

According to the survey, 60% of respondents prefer a web application over a mobile application. Therefore, this project will focus on developing a web application using Django and machine learning algorithms.

[**GO TO TOP**](#)

7.2. APPENDIX B: POST-SURVEY

7.2.1. POST-SURVEY FORM

What type of training or support do you think will be required to help Farmers for using Farmer Bag Application?

...
...

Tutorial Videos
 User Manual
 Awareness Pr0gam and Training
 All of the above

Figure 269: Post-Survey Form: Question 1

Did you find the disease detection feature of the app helpful in identifying plant diseases accurately?

- Extremely helpful
 Moderately helpful
 Somewhat helpful
 Not very helpful
 Not helpful at all

Figure 270: Post-Survey Form: Question 2

How would you rate the quality and accuracy of the information provided about identified plant diseases?



Figure 271: Post-Survey Form: Question 3

Were you able to find and purchase the necessary medicines and crops easily through the marketplace?

- Yes, very easily
- Yes, with some difficulty
- No, it was somewhat difficult
- No, it was very difficult
- I did not use this feature

Figure 272: Post-Survey Form: Question 4

Which of the following features of the Farmer Bag Application did you find most useful?

- Disease detection capability for early identification and intervention
- Access to detailed information about identified plant diseases
- Marketplace integration for purchasing medicines and crops
- Khalti payment gateway for seamless transactions
- Educational resources, such as the Study Blog, for improving plant health management skills

Figure 273: Post-Survey Form: Question 5

Did you encounter any difficulties during the payment process using the Khalti payment gateway?

- No difficulties at all
- Minor difficulties
- Some difficulties
- Major difficulties
- I did not use this feature

Figure 274: Post-Survey Form: Question 6

:::

Did you find the educational resources, such as the Study Blog, helpful in improving your plant health management skills?

- Yes
- No
- Maybe

Figure 275: Post-Survey Form: Question 7

What additional features or improvements would you like to see in future updates of the application?

- Improved disease detection accuracy
- Expanded educational resources
- Enhanced marketplace features
- Better user interface
- Other (please specify)

Figure 276: Post-Survey Form: Question 8

Overall, how would you rate your experience using the Farmer Bag Application?



Figure 277: Post-Survey Form: Question 9

Would you recommend the Farmer Bag Application to other farmers, gardeners, or plant enthusiasts?

- Yes
- No
- Maybe

Figure 278: Post-Survey Form: Question 10

7.2.2. SAMPLE OF FILLED SURVEY FORM

What type of training or support do you think will be required to help Farmers for using Farmer Bag Application?

- Tutorial Videos
- User Manual
- Awareness Pr0gam and Training
- All of the above

Figure 279: Sample of Filled Post-Survey (1)

Did you find the disease detection feature of the app helpful in identifying plant diseases accurately?

- Extremely helpful
- Moderately helpful
- Somewhat helpful
- Not very helpful
- Not helpful at all

Figure 280: Sample of Filled Post-Survey (2)

How would you rate the quality and accuracy of the information provided about identified plant diseases?

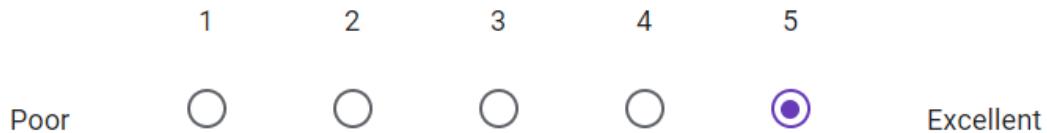


Figure 281: Sample of Filled Post-Survey (3)

Were you able to find and purchase the necessary medicines and crops easily through the marketplace?

- Yes, very easily
- Yes, with some difficulty
- No, it was somewhat difficult
- No, it was very difficult
- I did not use this feature

Figure 282: Sample of Filled Post-Survey (4)

Which of the following features of the Farmer Bag Application did you find most useful?

- Disease detection capability for early identification and intervention
- Access to detailed information about identified plant diseases
- Marketplace integration for purchasing medicines and crops
- Khalti payment gateway for seamless transactions
- Educational resources, such as the Study Blog, for improving plant health management skills

Figure 283: Sample of Filled Post-Survey (5)

Did you encounter any difficulties during the payment process using the Khalti payment gateway?

- No difficulties at all
- Minor difficulties
- Some difficulties
- Major difficulties
- I did not use this feature

Figure 284: Sample of Filled Post-Survey (6)

Did you find the educational resources, such as the Study Blog, helpful in improving your plant health management skills?

- Yes
- No
- Maybe

Figure 285: Sample of Filled Post-Survey (7)

What additional features or improvements would you like to see in future updates of the application?

- Improved disease detection accuracy
- Expanded educational resources
- Enhanced marketplace features
- Better user interface
- Other (please specify)

Figure 286: Sample of Filled Post-Survey (8)

Overall, how would you rate your experience using the Farmer Bag Application?

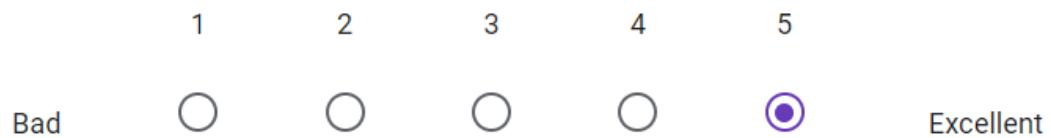


Figure 287: Sample of Filled Post-Survey (9)

Would you recommend the Farmer Bag Application to other farmers, gardeners, or plant enthusiasts?

- Yes
 - No
 - Maybe

Figure 288: Sample of Filled Post-Survey (10)

7.2.3. POST SURVEY RESULTS

What type of training or support do you think will be required to help Farmers for using Farmer Bag Application? [!\[\]\(f8c0820b29abb412a6e53689331f7c27_img.jpg\) Copy](#)

43 responses

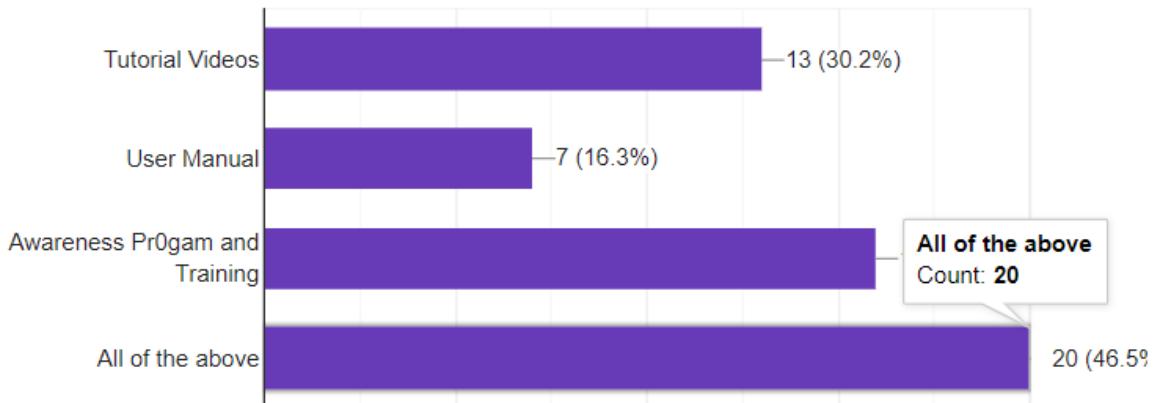


Figure 289: Post-Survey: Question 1

Survey responses highlighted the need for a comprehensive support system to help farmers effectively use the Farmer Bag Application. Tutorial videos, providing step-by-step guidance, and a user manual for reference were popular choices. Additionally, awareness programs and tailored training sessions offered hands-on learning opportunities. Combining these approaches creates a holistic framework, empowering farmers to confidently and efficiently engage with the app.

Did you find the disease detection feature of the app helpful in identifying plant diseases accurately?

 Copy

44 responses

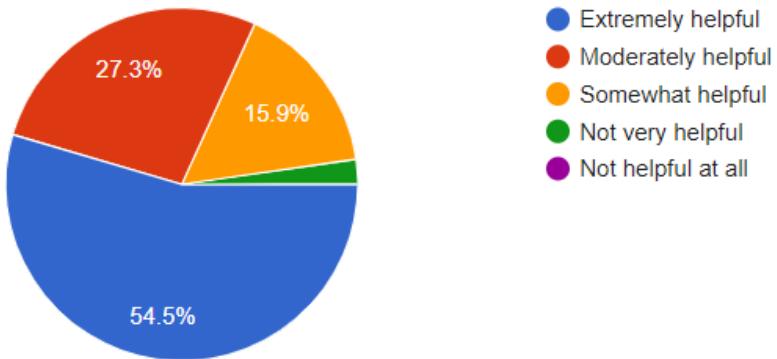


Figure 290: Post-Survey: Question 2

According to the survey, 56.3% of respondents found the disease detection feature of the app extremely helpful, 27% found it moderately helpful, and 14.6% found it somewhat helpful. Notably, none of the respondents considered it unimportant. These findings indicate a high level of satisfaction with the feature's accuracy and its value in aiding farmers with disease identification.

How would you rate the quality and accuracy of the information provided about identified plant diseases?

 Copy

44 responses

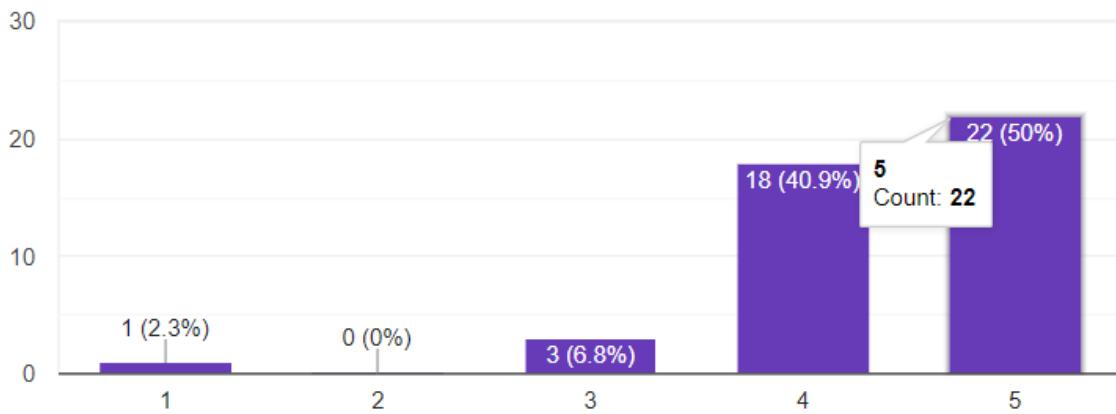


Figure 291: Post Survey: Question 3

According to the survey, 24 respondents gave the highest rating of 5 for the quality and accuracy of the information about identified plant diseases. Additionally, 20 respondents rated it as 4, indicating a very good level of satisfaction. Furthermore, 3 respondents provided a rating of 3, suggesting an average perception. Only 1 respondent rated it as 2. These varied ratings highlight a predominantly positive reception of the information, with room for minor improvements.

Were you able to find and purchase the necessary medicines and crops easily through the marketplace?

 Copy

44 responses

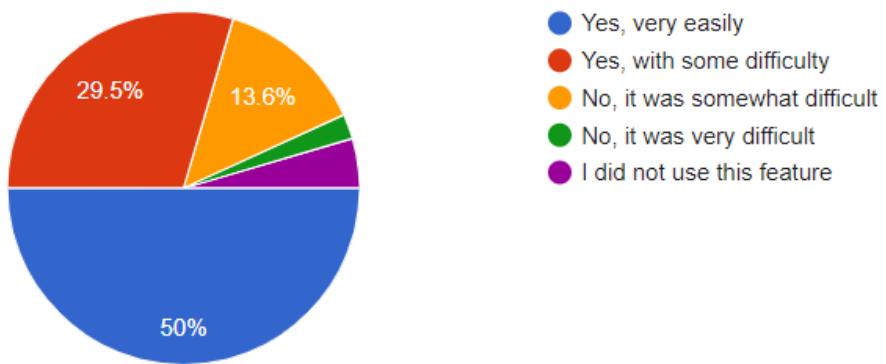


Figure 292: Post-Survey: Question 4

The survey revealed that 50% of respondents found it very easy to locate and purchase medicines and crops through the marketplace. This suggests a high level of satisfaction with the marketplace's usability and efficiency.

Which of the following features of the Farmer Bag Application did you find most useful?

[Copy](#)

48 responses

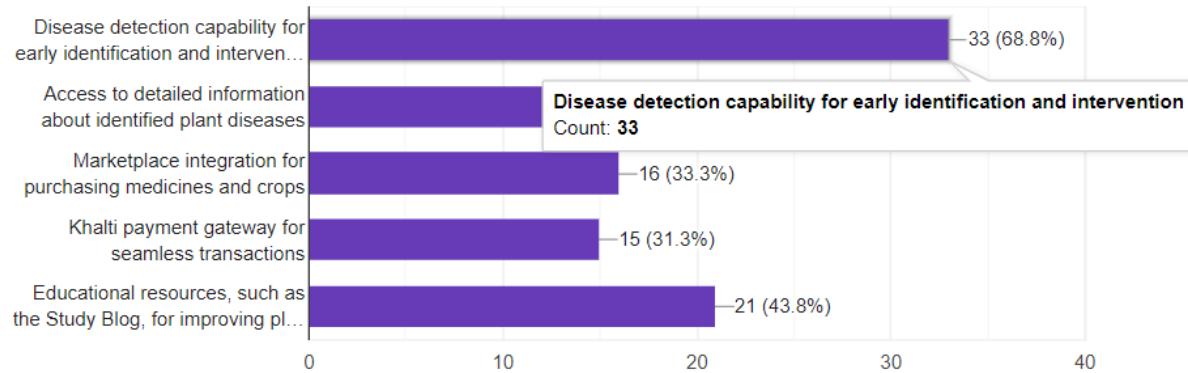


Figure 293: Post-Survey: Question 5

The survey findings highlight the disease detection capability as the most useful feature of the Farmer Bag Application, with 68.8% of respondents expressing its significance in aiding early disease identification and intervention. Access to detailed plant disease information, appreciated by 64.6% of respondents, played a crucial role in effective disease management. Additionally, 33.3% found the marketplace integration valuable for purchasing medicines and crops, while 31.3% expressed concerns with the Khalti payment gateway's functionality. The Study Blog's educational resources were deemed beneficial by 43.8% of respondents, underscoring the importance of ongoing learning. Overall, the survey emphasizes the importance of disease management tools and educational resources, while suggesting improvements to the payment gateway.

Did you encounter any difficulties during the payment process using the Khalti payment gateway?

 Copy

44 responses

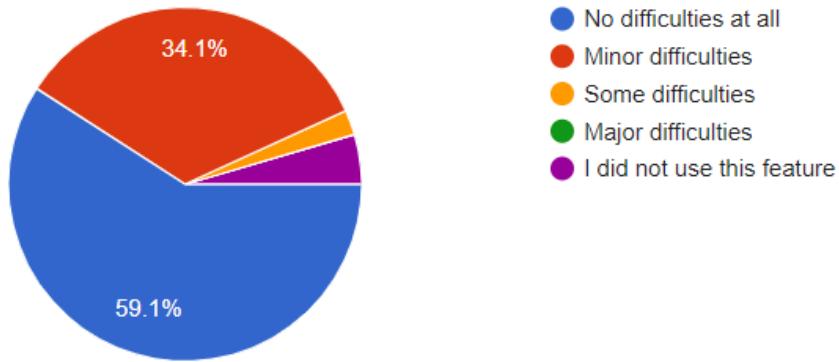


Figure 294: Post-Survey: Question 6

According to the survey, 59.1% of respondents agreed that they had not encountered any issues with the Khalti payment gateway. However, 34% of respondents reported minor difficulties, which happened due to server errors.

Did you find the educational resources, such as the Study Blog, helpful in improving your plant health management skills?

 Copy

43 responses

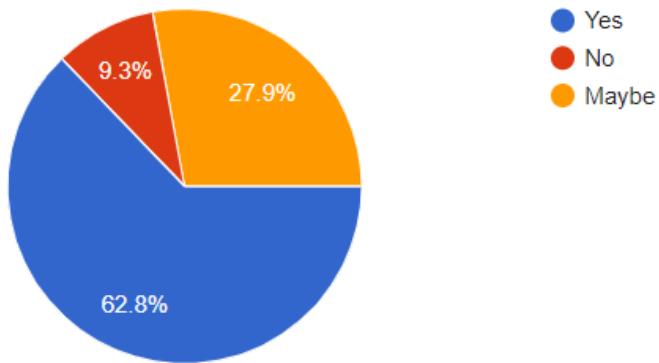


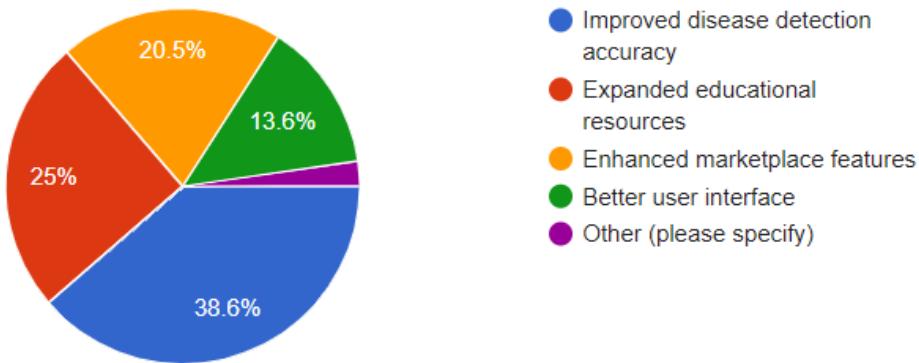
Figure 295: Post-Survey: Question 7

According to the survey, 62.8% of people agreed that the Study blog in the project is a helpful educational resource, 27% selected 'maybe', and 9% of respondents did not agree that this feature is useful.

What additional features or improvements would you like to see in future updates of the application?

 Copy

44 responses



[Figure 296: Post-Survey: Question 8](#)

As per this question, 38.6% of people need improvements in machine learning accuracy, 25% of respondents need improvements in educational resources, and 20.5% of respondents need improvements in marketplace features.

Overall, how would you rate your experience using the Farmer Bag Application?

 Copy

44 responses

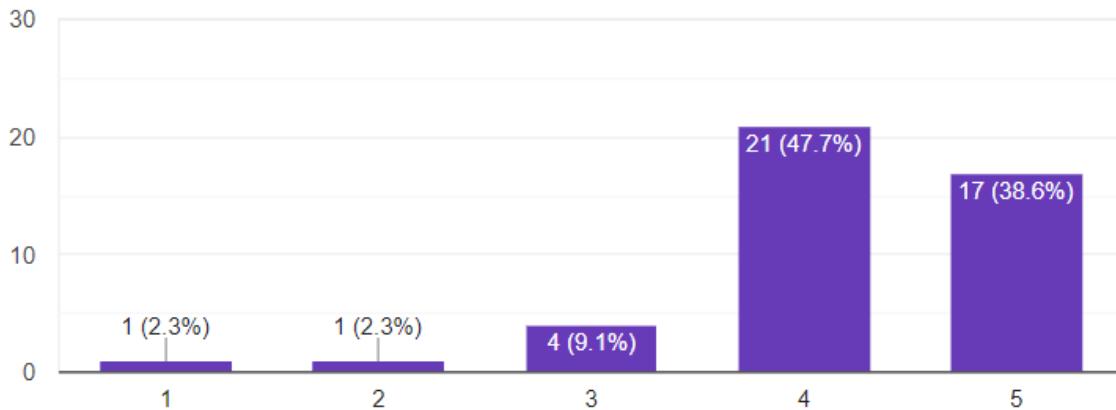


Figure 297: Post-Survey: Question 9

From the survey responses gathered from 44 participants regarding their experience with the Farmer Bag Application, it's evident that opinions varied. Nearly half of the respondents, accounting for 47.7%, rated their experience as 4, indicating a generally positive impression of the application. Following closely, 38.6% rated their experience as 3, suggesting a relatively neutral standpoint. However, a small fraction, 9.1%, rated their experience as 2, indicating some level of dissatisfaction. Notably, there were extreme ratings as well, with 2.3% rating their experience as 1, representing significant dissatisfaction, and 2.3% rating their experience as 5, representing exceptional satisfaction. While the majority seemed to have a positive or neutral experience, the presence of dissatisfied respondents suggests potential areas for improvement in the application's usability or features.

Would you recommend the Farmer Bag Application to other farmers, gardeners, or plant enthusiasts?

[!\[\]\(609a4f19882a3f10f6aff2fccbf005d4_img.jpg\) Copy](#)

42 responses

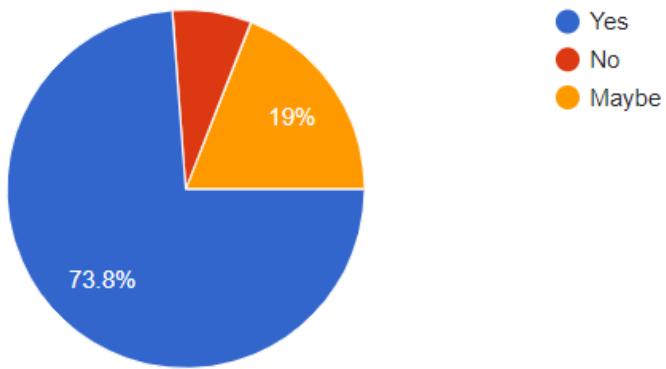


Figure 298: Post-Survey: Question 10

73.8% of respondents agreed to recommend this product to farmers, gardeners, and plant enthusiasts, showing an extreme level of satisfaction. Meanwhile, 19% may or may not recommend it to others.

[GO TO TOP](#)

7.3. APPENDIX C: REQUIREMENT ANALYSIS

7.3.1. SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)

1. Introduction

1.1. Purpose

The purpose of this document is to outline the requirements and specifications for the development of the Farmers' Bag Application, which aims to assist farmers in Nepal and globally by providing tools to detect plant diseases, accessing information and purchase agricultural supplies.

1.2. Document Conventions

1.2.1. Priority Levels

Requirements are assigned priority levels to indicate their relative importance. The priority levels used are:

- High (H): These are critical requirements that are essential for the core functionality and success of the application.
- Medium (M): These are important requirements that significantly contribute to the application's functionality, but there is some flexibility with them.
- Low (L): These are desirable requirements that enhance the user experience or provide additional features, but they are not critical to the application's core functionality.

1.2.2. Formatting Conventions

Bold Text: Used to highlight section headings, table heading, and important terms.

Normal Text: Used for paragraphs.

1.2.3. Use of Sections

The document is organized into sections, each addressing specific aspects of the software requirements, including introduction, objectives, features and detailed specifications.

1.2.3. Acronyms and Abbreviations

Acronyms and abbreviations are defined in the glossary (Appendix A) for any reference.

1.2.3. Consistency in Language

Clear and consistent language is used throughout the document to ensure understanding and readability.

1.2.3. Revision

The document is subject to revision and as needed to reflect changes in project requirements scope or objectives.

1.3. Project Scope

The project scope encompasses the development of a web application focusing on automated plant disease detection, education on plant diseases, and a marketplace for agricultural products. It aims to address the challenges faced by farmers globally, with a special emphasis on Nepal, where agriculture plays a vital role in sustaining livelihoods. By leveraging technology, the project seeks to empower farmers with tools for disease identification, knowledge acquisition, and access to essential supplies.

1.4. Intended Audience and Readings Suggestions

The intended audience for this document includes:

- Developer: Individuals responsible for software development.
- Product Manager: Oversees the development process and ensures alignment with project goals.
- Tester: Conducts testing to ensure the application functions correctly.
- Users: Farmers who utilize the application.
- Supervisors: Supervisors of the project can only view this part.

Reading Suggestions:

Developers should focus on understanding the technical requirements and specifications outlined in the document to guide the development process effectively.

Product managers should pay attention to the overall project objectives and scope to ensure alignment with organizational goals and stakeholder expectations.

Testers should familiarize themselves with the functional and non-functional requirements to develop comprehensive test plans and ensure the application meets quality standards.

Users should review sections related to application features and functionality to gain insights into how the application will benefit them and meet their needs.

Supervisors should oversee the document to ensure that project requirements are clearly defined and understood by all stakeholders, facilitating smooth project execution.

1.5. References

[*IEEE Recommended Practice for Software Requirements Specifications*](#)

2. Overall Description

2.1. Product Perspective

The Farmers' Bag Application is an innovative solution that operates within modern agricultural practices, leveraging technology to address challenges faced by farmers worldwide. It integrates automated plant disease detection using deep learning algorithms, educational resources on plant diseases, and a marketplace for agricultural products. The application aims to enhance farming efficiency, productivity, and sustainability, contributing to global food security initiatives. It interacts with external systems like databases, payment gateways, and internet protocols. As a standalone web application accessible on desktop and mobile, it offers convenience and accessibility. Aligning with efforts to integrate digital solutions into farming, the Farmers' Bag Application represents a significant advancement in precision agriculture, harnessing technology's potential to empower agriculture-dependent communities and address real-world challenges.

2.2. Product Functions

2.2.1. E-commerce Platform:

This section is for users to purchase products, including medicine related to farming.

2.2.2. Plant Disease Detection:

This is a machine learning model that uses deep learning Convolutional Neural Networks (CNNs) to classify plant diseases based on plant leaves.

2.2.3. Cart:

This is a feature for users to bundle products, enhancing their shopping experience.

2.2.4. Search and Filter:

These features enable users to search for products and filter them by price.

2.2.5. Rating and Feedback:

This feature allows users to rate the products they use and provide feedback on those products.

2.2.6. Payment:

This feature enables users to pay for the products they buy.

2.2.7. Profile Management:

This feature enables users to update their passwords and profile information.

2.2.8. Admin Panel:

This feature enables admin to add, delete, and update products, users, and manage orders. The admin can also view charts on the dashboard.

2.3. User Classes and Characteristics

This software supports or will be used by different variety of users types, including

- **Farmers**
- **Customers**
- **Admin**

2.4. Operational Environment

The software operates in the following environments:

2.4.1. Hardware

This software can be access through both laptop and mobile through browser.

2.4.2. Software

This software is built using HTML, CSS, and Bootstrap for the frontend technology. Django serves as the backend framework, facilitating server-side operations and logic. For data storage, the

application utilizes SQLite, providing a relational database management system to store and manage data effectively.

2.4.3. Network

Software requires internet connection to access.

2.5. Design and Implementation Constraints

The design and implementation constraints are mentioned below:

User must be login to purchase any product, detect plant diseases.

User can only view the product if they are not logged in.

2.6. User Document

The "Farmer Bag" software primarily relies on this SRS document as the reference for users to understand the functionality and features of the application. However, the user interface (UI) of this software is designed to enhance user experience, ensuring ease of use and comfort while navigating and utilizing the "Farmer Bag" software.

2.7. Assumption and Dependencies

The assumption and dependencies are mentioned below:

-

3. External Interfaces Requirements

3.1. User Interface

The “Farmer Bag Application” web application provides a user interface through which users interact with various features. Key users interact includes:

- User Authentication Pages (Login, Register, forgot password)
- User Profile Management Pages
- Home Page
- Prediction Page
- Marketplace
- Cart Page
- Checkout Page
- Admin Panel Page

3.2. Hardware Interfaces

As the application must operate over the internet, all hardware necessary for internet connectivity will be facilitated through the system's hardware interface.

3.3. Software Interfaces

- The application shall communicate with the Khalti Payment gateway for processing payments.
- The application shall communicate with the backend to fulfill all functional requirements.
- The application shall interact with the database to perform CRUD operations.

- The application shall communicate with machine learning model for plant disease detection using API.
- The application shall use browser for using application.

3.4. Collaboration Interfaces

The Farmer Bag Application system will utilize the HTTP protocol and API for Collaboration over the internet, enabling the seamless transfer of data back and forth.

4. Non-Functional Requirement

4.1 Performance Requirement

4.2 Safety Requirement

Encryption (HTTP/HTTPS) must be employed for all data transmission. User authentication and authorization mechanisms should be in place to prevent unauthorized access to sensitive information.

Data of one user cannot be seen or accessed by other users.

4.3 Business Rules

- Only registered users with valid credentials can login into the application.
- Only logged in user can buy products.
- Logged in user that they buy the product can give the rating and reviews.
- User can view the order details, edit profile, and change password if they are logged in.
- Authenticated user only can buy the product.

5. Other Requirements

5.1. Scalability and Capacity

The scalability and capacity considerations for this software are as follows:

- The application's architecture should be designed with scalability in mind to accommodate potential increases in both user base and data volume.
- Scalability testing should be conducted to ensure the system can effectively handle growing demands without compromising performance.
- The software should be capable of efficiently managing increasing user loads as the application gains popularity.
- The software should be designed to scale and support a large number of products as the platform grows.

5.2. Compatibility

Compatibility requirements with other systems or software are as follows:

- The software should be compatible with the latest versions of popular web browsers, including Chrome and Firefox.
- The software should be easily accessible for both desktop/laptop and mobile users through web browsers, ensuring seamless usability across different devices.

5.3. Project Management Tools

GitHub and Google Drive will be utilized for backups, while Trello will be employed to manage project tasks.

[GO TO TOP](#)

7.4. APPENDIX D: BACKGROUND

7.4.1. UNDERSTANDING THE SOULTION

7.4.1.1. IMPLEMENTATION OF TRANSFER LEARNING

DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3

Figure 299: Transfer Learning Used Model

Initially, plant disease detection was done using the Keras documentation and creating a sequential model. However, due to a lack of accuracy, the approach was switched to transfer learning where the EfficientNetB0 model was utilized, achieving an accuracy of 98 percent at 5 epochs. The number of epochs was not increased because it would obviously lead to overfitting. During this process, the negative class was handled by introducing the 'Unknown' Class in the train and test data. In this project, 25 different classes were used to predict plant diseases.

This code imports TensorFlow and its related libraries for building and training neural networks. It also imports necessary modules from the Keras API, such as layers and utilities for constructing

models. Additionally, it includes tools for image data preprocessing and the Adam optimizer for model optimization. Overall, it sets up the environment for developing deep learning models efficiently.

```
import tensorflow as tf
import tensorflow_hub as hub
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
```

Figure 300: Importing Necessary Libraries

In these lines, two directory paths are created. Firstly, root_dir is set to the current directory. Then, train_dir and val_dir are formed by appending 'train' and 'valid' to root_dir, respectively. These paths are crucial for organizing training and validation data in a machine learning project.

```
root_dir = '.'

train_dir = os.path.join(root_dir, 'train')
val_dir = os.path.join(root_dir, 'valid')
```

Figure 301: Creating two directory path.

```
import time
import os
from os.path import exists

def count(dir, counter=0):
    "returns number of files in dir and subdirs"
    for pack in os.walk(dir):
        for f in pack[2]:
            counter += 1
    return dir + " : " + str(counter) + "files"
```

Figure 302: Code to returning no of files.

This code prints the total number of images in train_dir and val_dir. The count function is used to count the total number of images in train_dir and val_dir. The code below defines the constants for the image shape and batch size to be used in the model. Image_Shape is set to (224, 224), indicating that the images will be resized to a square shape with dimensions of 224x224 pixels.

The Batch_Size is set to 64, which determines the number of samples processed before the model's internal parameters are updated during training. It specifies how many images are fed into the model at once for preprocessing.

```
print('total images for training :', count(train_dir))
print('total images for validation :', count(val_dir))

total images for training : ./train : 25359files
total images for validation : ./valid : 6427files

IMAGE_SHAPE = (224, 224)

BATCH_SIZE = 64
```

Figure 303: Setting Batch size and Image shape.

This code segment establishes data generators for training and validation datasets, integral for efficiently feeding data into neural network models during training. Initially, directory paths for the training (`train_dir`) and validation (`val_dir`) datasets are specified.

Subsequently, an image data generator (`validation_datagen`) is instantiated for the validation dataset, rescaling pixel values to a range between 0 and 1 and configuring various parameters such as image size, batch size, and color mode.

Additionally, a conditional statement determines whether data augmentation should be applied. If enabled, a new generator (`train_datagen`) with augmentation techniques like rotation, flipping, and shifting is created. Conversely, if augmentation is disabled, `train_datagen` is set to the same configuration as `validation_datagen`.

Lastly, a generator (`train_generator`) is generated for the training dataset, employing the configured `train_datagen` and following similar parameter settings as the validation generator. This setup ensures that both training and validation data are appropriately processed and augmented before being fed into the neural network model.

```
val_dir = './valid'
train_dir = './train'
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    val_dir,
    shuffle=False,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)

do_data_augmentation = True #@param {type:"boolean"}
if do_data_augmentation:
    train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale = 1./255,
        rotation_range=40,
        horizontal_flip=True,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        fill_mode='nearest' )
else:
    train_datagen = validation_datagen

train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)
```

Found 6427 images belonging to 24 classes.

Found 25359 images belonging to 24 classes.

Figure 304: Data Augmentation

This code creates a neural network model for image classification. It utilizes a pre-trained EfficientNet-B0 model from TensorFlow Hub as a feature extractor and adds a dense output layer for classification. The feature extractor's weights are frozen during training, and the output layer uses softmax activation for multi-class classification.

```
model = tf.keras.Sequential([
    hub.KerasLayer("https://www.kaggle.com/models/tensorflow/efficientnet/frameworks/TensorFlow2/variations/b0-feature-vector/versions/1",
                  trainable=False,
                  input_shape = (224, 224, 3)),
    tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
```

Figure 305: Implementing Transfer Learning Model

This code snippet configures the training process for a neural network model. It sets up the optimizer as Adam, a commonly used optimization algorithm, the loss function as categorical cross-entropy, which is suitable for multi-class classification tasks, and the metric to monitor during training as accuracy. These settings ensure that the model is trained effectively for image classification, optimizing towards accurate predictions while monitoring its performance.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

Figure 306: Compiling Model

```
model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)        Output Shape         Param #
keras_layer_1 (KerasLayer)  (None, 1280)      4049564
dense_1 (Dense)      (None, 24)           30744
_________________________________________________________________
Total params: 4080308 (15.57 MB)
Trainable params: 30744 (120.09 KB)
Non-trainable params: 4049564 (15.45 MB)
```

Figure 307: Printing Model Summary

This code trains a neural network model for 5 epochs using the provided training and validation data generators. The fit method is called on the model object with parameters specifying the data generators, the number of training epochs, and validation data. During training, the model adjusts its weights based on the provided loss function and optimization algorithm, while monitoring its performance on the validation data. The training history is stored for further analysis.

```
EPOCHS=5

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples//train_generator.batch_size,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples//validation_generator.batch_size)

Epoch 1/5
2024-04-18 16:07:42.430054: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:442] Loaded cuDNN version 8700
2024-04-18 16:07:42.855997: I tensorflow/tsl/platform/default/subprocess.cc:304] Start cannot spawn child process: No such file or directory
2024-04-18 16:07:43.515413: I tensorflow/tsl/platform/default/subprocess.cc:304] Start cannot spawn child process: No such file or directory
2024-04-18 16:07:43.693871: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x56094d686410 initialized for platform CUDA (this does not
2024-04-18 16:07:43.693913: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): NVIDIA GeForce RTX 3060 Laptop GPU, Comput
2024-04-18 16:07:43.704871: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRA
2/396 [=====] - ETA: 32s - loss: 3.1620 - accuracy: 0.0703
2024-04-18 16:07:43.821811: I ./tensorflow/compiler/jit/device_compiler.h:186] Compiled cluster using XLA! This line is logged at most once for the
396/396 [=====] - 233s 569ms/step - loss: 0.7182 - accuracy: 0.8442 - val_loss: 0.4180 - val_accuracy: 0.9038
Epoch 2/5
396/396 [=====] - 218s 549ms/step - loss: 0.2466 - accuracy: 0.9421 - val_loss: 0.2914 - val_accuracy: 0.9322
Epoch 3/5
396/396 [=====] - 216s 545ms/step - loss: 0.1752 - accuracy: 0.9566 - val_loss: 0.2563 - val_accuracy: 0.9320
Epoch 4/5
396/396 [=====] - 205s 518ms/step - loss: 0.1408 - accuracy: 0.9642 - val_loss: 0.2146 - val_accuracy: 0.9448
Epoch 5/5
396/396 [=====] - 237s 597ms/step - loss: 0.1214 - accuracy: 0.9692 - val_loss: 0.1978 - val_accuracy: 0.9458

model.evaluate(validation_generator)
101/101 [=====] - 9s 90ms/step - loss: 0.1992 - accuracy: 0.9454
[0.19924651086330414, 0.9453866481781006]
```

Figure 308: Training Model in 5 Epochs

```

import matplotlib.pyplot as plt
import numpy as np

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.show()

```

Figure 309: Plotting Training and Validation Chart

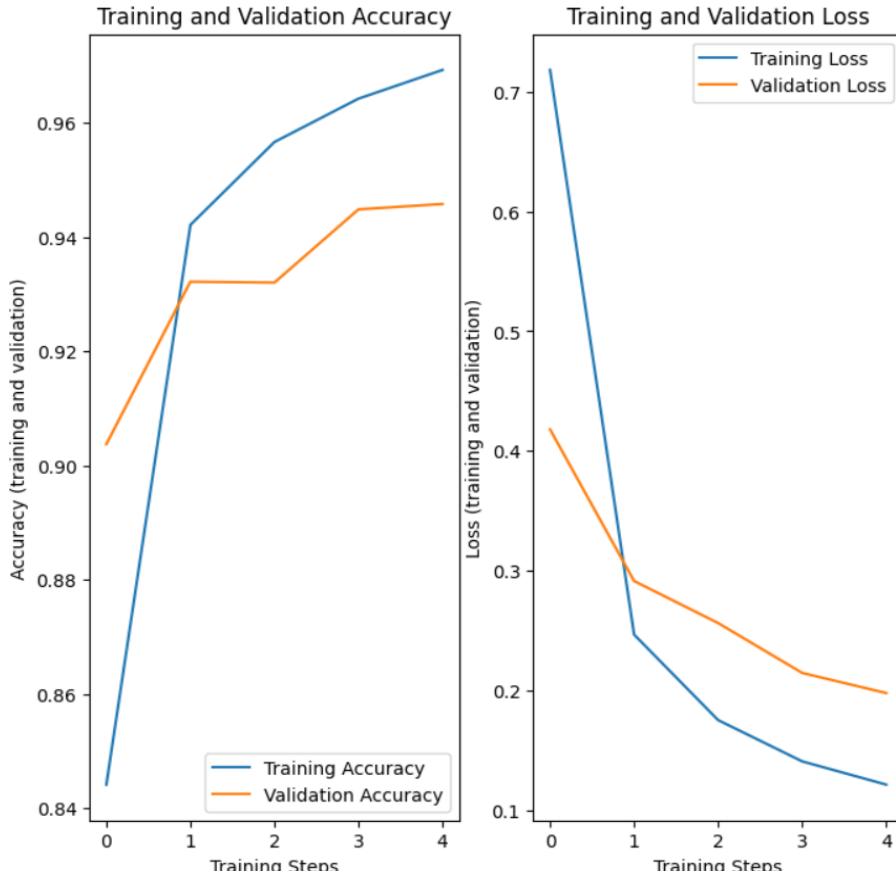
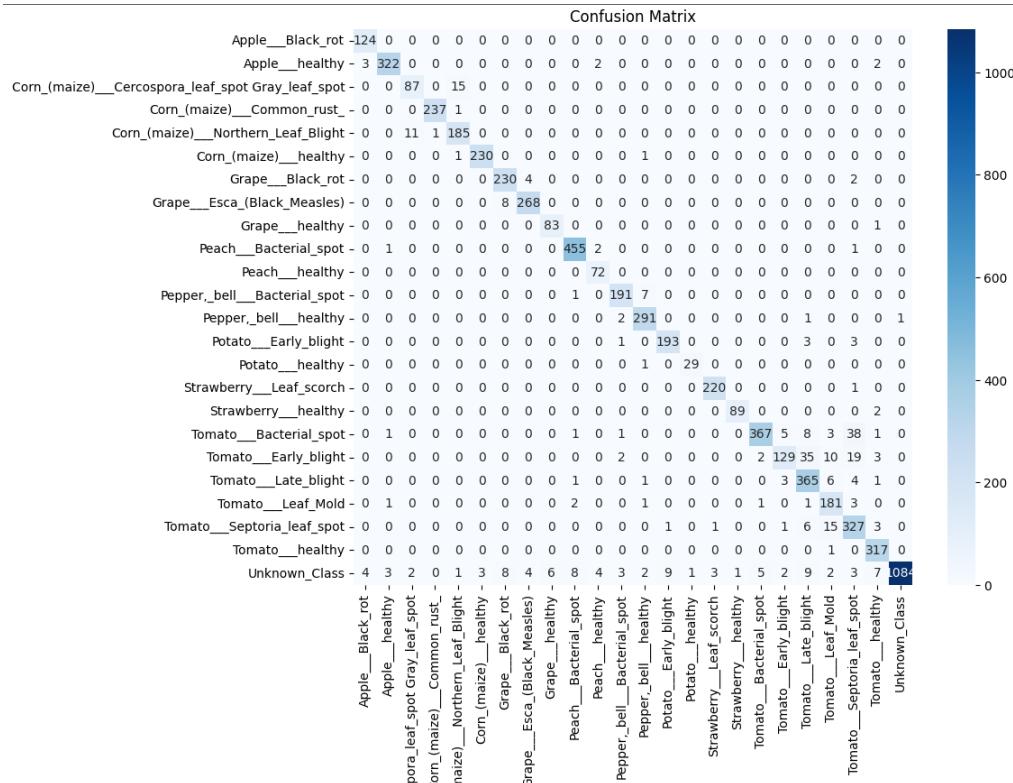


Figure 310: Plotting Validation and Accuracy Chart

This code snippet employs scikit-learn's classification_report function to evaluate the performance of a neural network model on a validation dataset. It makes predictions on the validation set, converts them into class labels, retrieves true class labels, and generates a detailed classification report containing metrics like precision, recall, and F1-score for each class. Finally, the report is printed to assess the model's performance quickly and comprehensively.

		precision	recall	f1-score	support
	Apple__Black_rot	0.95	1.00	0.97	124
	Apple__healthy	0.98	0.98	0.98	329
Corn_(maize)	Cercospora_leaf_spot_Gray_leaf_spot	0.87	0.85	0.86	102
	Corn_(maize)_Common_rust_	1.00	1.00	1.00	238
	Corn_(maize)_Northern_Leaf_Blight	0.91	0.94	0.93	197
	Corn_(maize)_healthy	0.99	0.99	0.99	232
	Grape__Black_rot	0.93	0.97	0.95	236
Grape	Esca_(Black_Measles)	0.97	0.97	0.97	276
	Grape__healthy	0.93	0.99	0.96	84
	Peach__Bacterial_spot	0.97	0.99	0.98	459
	Peach__healthy	0.90	1.00	0.95	72
Pepper,_bell	Bacterial_spot	0.95	0.96	0.96	199
	Pepper,_bell_healthy	0.96	0.99	0.97	295
	Potato__Early_blight	0.95	0.96	0.96	200
	Potato__healthy	0.97	0.97	0.97	30
	Strawberry__Leaf_scorch	0.98	1.00	0.99	221
	Strawberry__healthy	0.99	0.98	0.98	91
	Tomato__Bacterial_spot	0.98	0.86	0.92	425
	Tomato__Early_blight	0.92	0.65	0.76	200
	Tomato__Late_blight	0.85	0.96	0.90	381
	Tomato__Leaf_Mold	0.83	0.95	0.89	190
Tomato	Septoria_leaf_spot	0.82	0.92	0.87	354
	Tomato__healthy	0.94	1.00	0.97	318
	Unknown_Class	1.00	0.92	0.96	1174
	accuracy			0.95	6427
	macro avg	0.94	0.95	0.94	6427
	weighted avg	0.95	0.95	0.94	6427

Figure 311: Printing precision, recall and f1score

**Figure 312: Building Confusion Matrix**

At last, saving the model which saves weight, architecture and model.

```
# Save the model
model.save("efficientnetb0_model")

INFO:tensorflow:Assets written to: efficientnetb0_model/assets
INFO:tensorflow:Assets written to: efficientnetb0_model/assets
```

Figure 313: Saving the model

[GO TO TOP](#)

7.4.2. SIMILAR PROJECTS

7.4.2.1 RAPID APPLICATION DEVELOPMENT

Rapid Application Development (RAD) is a flexible methodology that enables fast and efficient creation and deployment of software applications. With RAD, software developers can make multiple iterations and updates without starting from scratch each time. This ensures that the final product is more quality-oriented and aligned with the needs of end-users. RAD also allows for continuous adaptation and incorporation of new features and functions at every stage of the development process. James Martin introduced this innovative approach in the 1980s. (P Beynon-Davies,C Carne,H Mackay &D Tudhope, 2017)

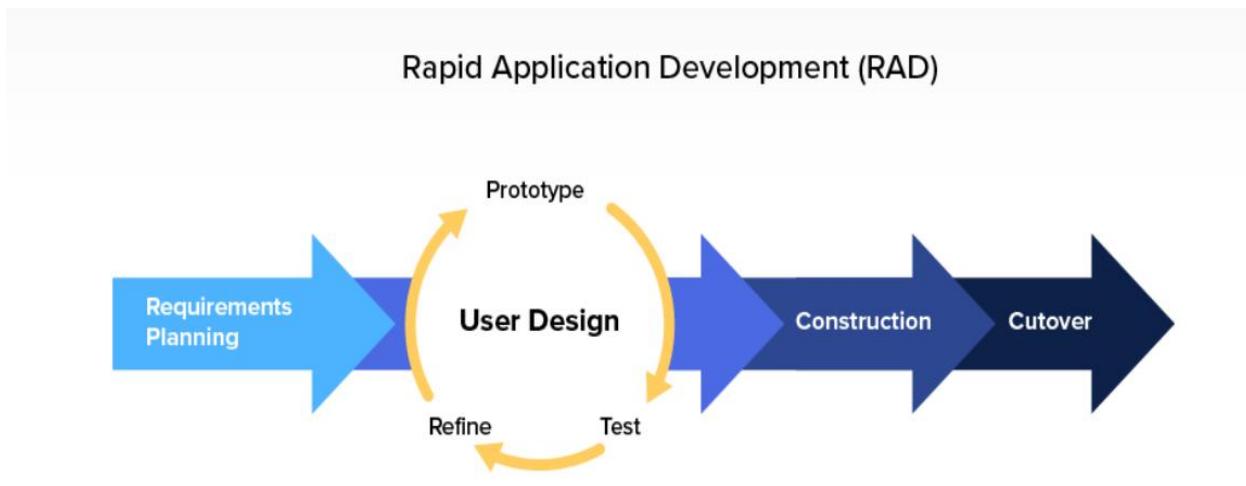


Figure 314: Considered Methodology: Rapid Application Development (kissflow, 2023).

Reason for not choosing Rapid Application Development:

- Limited control over the development process,
- Ideal for smaller projects; may not scale well for larger applications,
- Emphasis on speed might lead to potential stability and security concerns,
- May face challenges in managing a project with specific requirements,
- Requires less control but might compromise on stability,
- Quick iterative development may align with project deadlines.

[GO TO TOP](#)

7.4.4. PHASES OF CONSIDERED METHODOLOGY (RUP)

RUP, which is a software development process from Rational, a division of IBM, has four phases as follows:

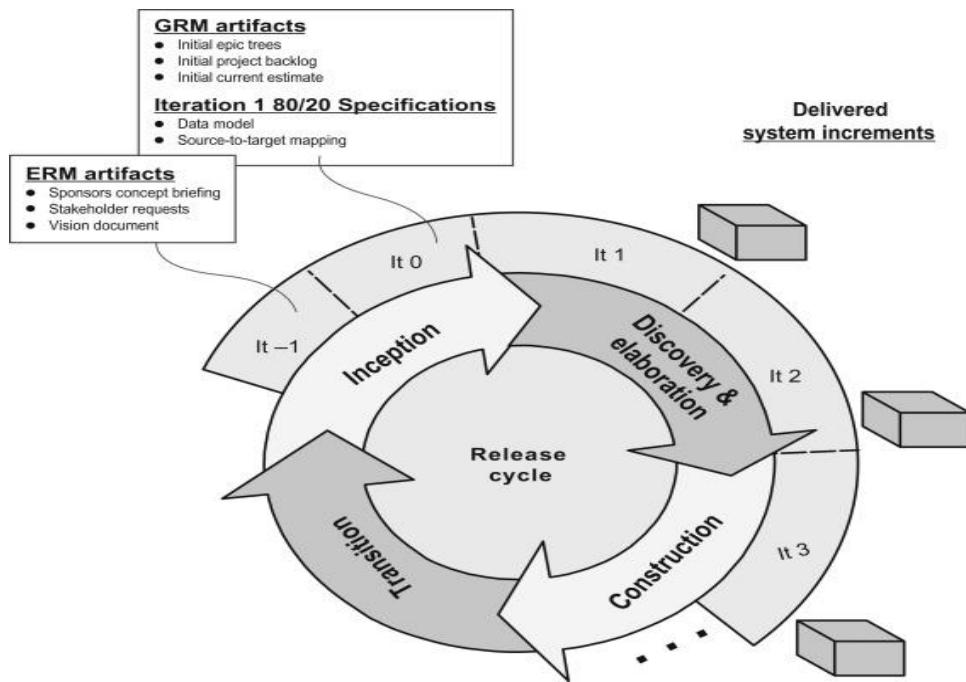


Figure 315: Phases of Methodology: Rational Unified Process

Inception Phase:

- Evaluate the entire project thoroughly.
- Determine if the project is worth pursuing.
- Communication and planning are key activities.
- Identify project scope using a use-case model.
- Estimate costs and time required.
- Identify customer requirements.

- Develop the project plan, goals, risks, use-case model, and project description.
- Check project against milestone criteria.
- If the criteria are not met, consider canceling or redesigning the project.

Elaboration Phase:

- Conduct in-depth evaluation of the project.
- Develop a detailed development plan and architecture.
- Evaluate required resources.
- Planning and modeling are primary activities.
- Revise or redefine the use-case model, business case, and risks.
- Check project against milestone criteria again.
- Consider canceling or redesigning if criteria are not met.
- Establish executable architecture baseline.

Construction Phase:

- Create the software system.
- Develop all features and components.
- Coding and testing activities take place.
- Developers complete the project.
- Transition Phase:
- Hand over the final product to the user.
- Address any issues or queries arising post-completion.
- Release the final project to the public.

Transition Phase

- Update project documentation.
- Conduct beta testing.
- Remove defects based on public feedback.

7.4.4.1. REASONS FOR CHOOSING RATIONAL UNIFIED PROCESS

The reason for choosing Rational Unified Process (RUP) Methodology is:

- The methodology provides a specific plan for each development process of the project,
- Using this methodology helps to prevent all the resources used in the project from being waste,
- Helps to manage track of my project in systematic order thus results to low chance of failure,
- Helps to customize the framework as per needs,
- Helps to produce high quality software relating to project plan,
- Allows to verify my project quality more efficiently,
- It matches perfectly for the small and intermediate project (Perfect for FYP projects),
- It provides smooth and comfortable development processes to fast-moving projects.

[GO TO TOP](#)

7.5. APPENDIX E: DESIGN

7.5.1. USE CASE DIAGRAM DESCRIPTION

7.5.1.1. DESCRIPTION OF HIGH-LEVEL DESCRIPTION

In the late initial phase, it is essential to provide a concise description of the task in the form of unstructured text. This step is crucial as it outlines the high-level functions and scope of a system. Additionally, these diagrams help identify interactions between the system and its actors. They describe what the system does and how the actors use it, although they do not delve into the details of how the system operates.

7.5.1.1.1: HIGH LEVEL USE CASE DESCRIPTION OF REGISTER

Name	Register
Actor(s)	Farmers'
Descriptions	Farmers' can create new accounts by providing the required information to access the application features.

Table 32:High Level Use Case Diagram: Register

7.5.1.1.2: HIGH LEVEL USE CASE DESCRIPTION OF DASHBOARD

Name	Dashboard
Actor(s)	User, Admin
Descriptions	Users and admin can view their dashboard which displays their relevant information.

Table 33: High Level Use Case Diagram: Dashboard**7.5.1.1.3: HIGH LEVEL USE CASE DESCRIPTION OF LOGIN/LOGOUT**

Name	Login/Logout
Actor(s)	User
Descriptions	Before using the features of Farmers' Bag, the user or admin should log in to the system using their email and password. The system also provides the feature for logging out.

Table 34: High Level Use Case Diagram: Login/Logout**7.5.1.1.4: HIGH LEVEL USE CASE DESCRIPTION OF BROWSE PRODUCT**

Name	Browse Product
Actor(s)	User and Admin
Descriptions	Users and admin can browse the available products. This involves navigating through various categories and listings to view details about different agricultural products.

Table 35: High Level Use Case Diagram: Browse Product**7.5.1.1.5: HIGH LEVEL USE CASE DESCRIPTION OF SEARCH PRODUCT**

Name	Search Product
Actor(s)	User and Admin
Descriptions	Users and admins can search for specific products based on keywords, categories, or other criteria. The system will then display relevant search results.

Table 36: High Level Use Case Diagram: Search Product**7.5.1.1.6: HIGH LEVEL USE CASE DESCRIPTION OF VIEW CART**

Name	View Cart
Actor(s)	User
Descriptions	Users can view the contents of their shopping cart. This includes detailed information about the selected products, such as product details, images, pricing, and other relevant information.

Table 37:High Level Use Case Diagram: View Cart**7.5.1.1.7: HIGH LEVEL USE CASE DESCRIPTION OF UPDATE CART**

Name	Update Cart
Actor(s)	User
Descriptions	Users have the capability to update their shopping cart. This involves modifying quantities, removing items, or making other adjustments to the contents of the cart.

Table 38:High Level Use Case Diagram: Update Cart

7.5.1.1.8: HIGH LEVEL USE CASE DESCRIPTION OF RATINGS

Name	Ratings
Actor(s)	User
Descriptions	Users can provide ratings and reviews for products. This feature helps in building trust among users and providing feedback on the quality of products.

Table 39:High Level Use Case Diagram: Ratings**7.5.1.1.9: HIGH LEVEL USE CASE DESCRIPTION OF PREDICT DISEASE**

Name	Predict Disease
Actor(s)	User
Descriptions	Users can utilize the disease prediction feature to identify potential diseases affecting plants. This might involve uploading images or providing relevant information for analysis.

Table 40:High Level Use Case Diagram: Predict Disease

7.5.1.1.10: HIGH LEVEL USE CASE DESCRIPTION OF VIEW STUDY PLACE

Name	Pay Bill
Actor(s)	User
Descriptions	Users can make payments for their purchases. This involves selecting products, adding them to the cart, and completing the payment process.

*Table 41:High Level Use Case Diagram: View Study Place***7.5.1.1.11: HIGH LEVEL USE CASE DESCRIPTION OF PAY BILL**

Name	Pay Bill
Actor(s)	User
Descriptions	Users can make payments for their purchases. This involves selecting products, adding them to the cart, and completing the payment process.

Table 42: High Level Use Case Diagram: Pay Bill

7.5.1.1.12: HIGH LEVEL USE CASE DESCRIPTION OF MANAGE ACCOUNTS

Name	Mange Accounts
Actor(s)	User and Admins
Descriptions	Users and admins can manage their accounts. This includes activities such as updating personal information, changing passwords, and other account-related tasks

Table 43: High Level Use Case Diagram: Manage Accounts**7.5.1.1.13: HIGH LEVEL USE CASE DESCRIPTION OF ADD PRODUCTS**

Name	Add Products
Actor(s)	Admin
Descriptions	Admins have the capability to add new products to the system. This involves providing details such as product name, description, pricing, and other relevant information.

Table 44: High Level Use Case Diagram: Add Product

7.5.1.1.14: HIGH LEVEL USE CASE DESCRIPTION OF DELETE PRODUCT

Name	Delete Product
Actor(s)	Admin
Descriptions	Admins can delete product from the system. This might be necessary for managing product catalogue.

Table 45: High Level Use Case Diagram: Delete Product**7.5.1.1.15: HIGH LEVEL USE CASE DESCRIPTION OF ADD CATEGORIES**

Name	Add Categories
Actor(s)	Admin
Descriptions	Admins have the capability to add new categories to the system. This involves creating a structure for organizing products and enhancing the user experience.

Table 46: High Level Use Case Diagram: Add Categories

7.5.1.1.16: HIGH LEVEL USE CASE DESCRIPTION OF DELETE CATEGORIES

Name	Delete Categories
Actor(s)	Admin
Descriptions	Admins have the capability to delete existing categories from the system. This involves removing categories that are no longer relevant or necessary for organizing products.

*Table 47: High Level Use Case Diagram: Delete Categories***7.5.1.1.17: HIGH LEVEL USE CASE DESCRIPTION OF MANAGE PAYMENTS**

Name	Manage Payments
Actor(s)	Admin
Descriptions	Admins oversee payment transactions, including viewing details, resolving issues, and maintaining transaction records for auditing. This use case ensures efficient monitoring and integrity maintenance of the payment process in the Farmers Bag application.

Table 48: High Level Use Case Diagram of Manage Payments[GO TO TOP](#)

7.5.1.2. DESCRIPTION OF EXPANDED-LEVEL DIAGRAM

7.4.1.2.1: EXPANDED LEVEL USE CASE DESCRIPTION OF REGISTER

Name: Register

Actor(s): Admin

Purpose: Record Farmers details

Overview: Farmers' can create new accounts by providing the required information to access the application features.

Type: Primary, Essential

Action Steps:

Actor Action	System Response
1. Farmers open web browser.	
2. Farmers enters URL into the web browser.	
	3. System displays Login Page
4. Farmers enters required information to register into this.	
	5. System validates required fields
6. Farmers registered to application	

Table 49: Expanded Level Use Case Diagram: Register

Alternative Flow (Error Handling)

5. System displays error message indicating specific validation issues.

5. Farmers are prompted to correct the errors and resubmit the register from

7.4.1.2.2: EXPANDED LEVEL USE CASE DESCRIPTION OF DASHBOARD**Name:** Dashboard**Actor(s):** User**Purpose:** Provide a personalized view of information for users and administrators.**Overview:** User can view their dashboard, which displays relevant information such as total no of orders, payment details etc and other features. Admins have access to admin dashboard with additional functionality.**Type:** Primary, Essential**Action Steps:**

Actor Action	System Response
1. User opens the browser	
2. User enters the URL into the browser	
	3. System displays Login Page
4. User logs in using their credentials	
	5. System validates credentials. If the credentials are valid the system directs the user to their personalized dashboard.
6. User views personalized dashboard information	
	7. System presents relevant information such as order history, carts, payments, and other user-specific data.
8. Admin logs in using admin credentials	

	9. System verifies the admin credentials. And if they are valid the system directs the admin to the admin dashboard.
	10. System presents relevant information or functionalities for administrators, such as user management, payment management, product management.

Table 50: Expanded Level Use Case Diagram: View Dashboard

Alternative Flow (Error Handling)

4,8: System displays error message indicating incorrect credentials and are prompted to re-enter their login information.

7.4.1.2.3: EXPANDED LEVEL USE CASE DESCRIPTION OF LOGIN/LOGOUT

Name: Login/Logout

Actor(s): User, Admin

Purpose: Authenticate users and administrators to access the application features.

Overview: Before using the Farmers' Bag Application, the user or admin should log in to the system using their email and password. The system also provides the features for logging out.

Type: Primary, Essential

Action Steps:

Actor Action	System Response
1. User or admin opens the browser	
2. User or admin enters the url into the browser	
	3. System displays Login Page
4. User or logs in using their credentials	
	5. System validates credentials. If the credential are valid for users, the system grants access to the user's account and if the credentials are valid for admin, the system grants access to the admin accounts.
6. User and admin can view their personalized dashboard information	
	7. System presents relevant information such as order history, carts, payments and other user-specific data for users and for admin system presents relevant information such as product

	management, user management, payment management.
--	---

Table 51: Expanded Level Use Case Diagram: Login/Logout

Alternative Flow (Error Handling):

4. System displays error message indicating incorrect credentials and are prompter to re-enter their login information.

7.4.1.2.4: EXPANDED LEVEL USE CASE DESCRIPTION OF BROWSE PRODUCT

Name: Browse Product

Actor(s): User, Admin

Purpose: Allow users and admins to explore and view products available on the platform.

Overview: Users and admins can browse through the available products, exploring different categories and accessing detailed information about each product.

Type: Primary, Essential

Action Steps

Actor Action	System Response
1. User opens the browser and enters URL into the web browser	
	2. System displays Login Page
3. User logs in using their credentials	
	4. System validates credentials. If the credentials are valid the system directs the user to their personalized dashboard.
5. User clicks the on the Product.	
	6. System presents the list of categories.
7. Users select product categories and click on a specific product to view details.	
	8. System displays a list of products within the selected categories and present the detailed information about he selected product, including images, description, and pricing.

9. Admin opens the browser and enters URL into the web browser and logs in using admin credentials.	
	10. System displays login admin login page and validates admin credentials and grants access to the admin's account
11. Admin click the "Product" option and can view and manage products.	
	12. System provides access to the product management interface and system allows the admin to view, add, edit, or remove product from the platform.

Table 52: Expanded Level Use Case Diagram: Browse Product

Alternative Flow (Error Handling)

12. System displays an error message indicating the issue.

7.4.1.2.5: EXPANDED LEVEL USE CASE DESCRIPTION OF SEARCH PRODUCT

Name: Search Product

Actor(s): User and admin

Purpose: Allow users and admin to search for specific products within the Farmers Bag application.

Overview: Users can utilize the search functionality to find products based on keywords, categories, or other search criteria. The system retrieves and presents relevant product information matching the user's search query.

Type: Primary, Essential

Action Steps

Actor Action	System Response
1. User and admin open the application.	
2. User and admin enter the search query in the search bar.	
	3. System retrieves the relevant product information matching the search criteria.
	4. System displays the list of the match product.
5. User and admin select a specific product from the search results.	
6. User and admin interact with product information and can add the product to the cart	

Table 53: Expanded Level Use Case Diagram: Search Product

7.4.1.2.6: EXPANDED LEVEL USE CASE DESCRIPTION OF VIEW CART**Name:** View Cart**Actor(s):** User**Purpose:** Allow users to view the contents of their shopping cart.

Overview: User can access their shopping cart to review the items they have added. The system displays detailed information about each product in the cart, including names, quantities, prices etc. This enables users to make informed decisions about their purchases.

Type: Primary, Essential**Action Steps**

Actor Action	System Response
1. User navigates to the shopping cart.	
	2. System retrieves and displays the content of user's cart.
3. User views and reviews the products in the shopping cart.	
4. User can proceed to checkout, remove item from the cart.	
	5. System provides options for user to take further actions with cart.
6. User interacts with additional cart details or proceeds with checkout.	

Table 54: Expanded Level Use Case Diagram: View Cart**Alternative Flow (Error Handling)**

2. If the cart is empty system displays a message indicating that the shopping cart is empty.

7.4.1.2.7: EXPANDED LEVEL USE CASE DESCRIPTION OF UPDATE CART**Name:** View Cart**Actor(s):** User**Purpose:** Allow users to update the contents of their shopping cart.

Overview: Users have the capability to modify their shopping cart by updating quantities, removing items, or making other adjustments. This functionality ensures that users can manage their selections before proceeding to checkout.

Type: Primary, Essential**Action Steps**

Actor Action	System Response
1. User navigates to the shopping cart.	
	2. System retrieves and displays the content of user's cart.
3. User selects a specific item in the cart and updates the selected item in the cart.	
	4. System provides to modify the selected item, such as changing the quantity or removing if from cart.
	5. System validates the changes made by the user, updating the cart accordingly.
6. User can proceed to checkout, continue updating, or removing more items.	

Table 55: Expanded Level Use Case Diagram: Update Cart

7.4.1.2.8: EXPANDED LEVEL USE CASE DESCRIPTION OF RATINGS**Name:** Ratings**Actor(s):** User**Purpose:** Allow users to provide ratings and reviews for products on the platform.**Overview:** Users can share their feedback by assigning ratings and writing reviews for products they have purchased or experienced.**Type:** Primary, Essential**Action Steps**

Actor Action	System Response
1. User opens the browser and enters URL into the web browser	
3. User logs in using their credentials	2. System displays Login Page
	4. System validates credentials. If the credentials are valid the system directs the user to their personalized dashboard.
5. User navigates to the “My Orders” or “Product Details” section and user select a specific product to rate and review.	
	6. System provides access to the user’s order history or product details and system displays the product details, including the current ratings and reviews.
7. User assigns a numerical rating to product (eg. 1 to 5stars) and writes a	

review or comments about the product.	
	8. System register the user rating for the selected product and allows user to enter additional feedback and comments.
9. User submits the rating and reviews.	
	10. System stores the user rating and review in the database associated with the respective products.

Table 56: Expanded Level Use Case Diagram: Ratings

Alternative Flow (Error Handling)

7. If there are issues with submitting the rating and reviews, system displays an error message indicating the issue. And user is prompted to try submitting the rating and review again later.

7.4.1.2.9: EXPANDED LEVEL USE CASE DESCRIPTION OF PREDICT DISEASE

Name: Predict Disease

Actor(s): User

Purpose: Allow users to predict and identify disease affecting plants.

Overview: User can use the application's disease prediction feature to analyze plant symptoms and receive predictions about potential disease affecting their crops

Type: Primary, Essential

Actor Action	System Response
1. User opens the browser and enters URL into the web browser	
	2. System displays Login Page
3. User logs in using their credentials	
	4. System validates credentials. If the credentials are valid the system directs the user to their personalized dashboard.
5. User navigates to the "Predict Disease" section and uploads image.	
	6. System provides access to the disease prediction interface and system process the provided image using deep learning algorithm called CNN
	7. System analyses the image using machine learning model and the system predict the potential disease.
	8. System provides the disease name that the plant image is affected.

Table 57: Expanded Level Use Case Diagram: Predict Disease

7.4.1.2.10: EXPANDED LEVEL USE CASE DESCRIPTION OF VIEW STUDY PLACE**Name:** Study Place**Actor(s):** User

Purpose: Allow users to view information about study places or educational resources related to agriculture.

Overview: Users can access information about study places, educational institutions, or resources relevant to agriculture and farming.

Type: Primary, Essential

Actor Action	System Response
1. User opens the browser and enters URL into the web browser	
	2. System displays Login Page
3. User logs in using their credentials	
	4. System validates credentials. If the credentials are valid the system directs the user to their personalized dashboard.
5. User navigates to the “View Study Place” section and explore available study places or educational resources.	
	6. System provide access to information about the study place or educational resources.

Table 58: Expanded Level Use Case Diagram: View Study Place

Alternative Flow (Error Handling)

6. If there are issues in loading information of study place then system displays an error message indicating the issue.

7.4.1.2.11: EXPANDED LEVEL USE CASE DESCRIPTION OF PAY BILL**Name:** Pay Bill**Actor(s):** User**Purpose:** Allow users to pay bills for their purchases within the Farmer Bag Application.**Overview:** User can initiate the payment process for their purchases, providing secure and convenient transactions. The system handles payment details, ensures transaction security, and confirms successful payment to users.**Type:** Primary, Essential

Actor Action	System Response
1. User proceeds to check out in the shopping cart	
	2. System displays Payment options and prompts the user to select a payment method.
3. User selects a preferred payment method and enters payment details.	
	4. System processes and validates payment information provided by the user.
	5. System communicates with the payment gateway to authorize the transaction.
	6. System confirms the successful payment and updates the order.
7. User receives an email or notification with the payment confirmation and order details	
8. User can view the order history and continue shopping.	

Table 59: Expanded Level Use Case Diagram: Pay Bill

7.4.1.2.12: EXPANDED LEVEL USE CASE DESCRIPTION OF MANAGE ACCOUNTS**Name:** Manage Accounts**Actor(s):** Admin**Purpose:** Allow admin to manage user accounts within the Farmers Bag Application.**Overview:** Admins can perform various account management tasks, including user registration, account updates, password resets, and access control. This ensures a secure and organized user management system.**Type:** Primary, Essential

Actor Action	System Response
1. Admin login into the admin interface of the Farmers Bag Application	
2. Admin navigates to the user account management section and view a list of registered user accounts.	
	3. System displays the list of accounts, including relevant details such as usernames, email address etc.
4. Admin selects a specific user account.	
	5. System provides options for the admin to view and update the selected user account.
6. Admin can perform various action such as updating user details, resetting passwords, or deleting users.	
	7. System validates admin actions and updates user account information accordingly.

8. Admin receives a confirmation message after successful account management.	
9. Admin continues managing accounts or navigate to other admin functionalities.	

Table 60: Expanded Level Use Case Diagram: Manage Accounts

Alternative Flow (Error Handling)

7. If there are issue with account management task system displays error message indicating issue.

7.4.1.2.13: EXPANDED LEVEL USE CASE DESCRIPTION OF ADD PRODUCTS**Name:** Add Product**Actor(s):** Admin**Purpose:** Allow admin to add new product to the platform.**Overview:** Admins can use the product management interface to add and update information about new product, making available for users to browse and purchase.**Type:** Primary, Essential

Action Steps

Actor Action	System Response
1. Admin opens the web browser and enters the URL into the web browser	
3. Admin logs in using their credentials	2. System displays Login Page
	4. System validates credentials. If the credentials are valid the system directs the admin to their dashboard.
5. Admin navigates to the product management interface and select the product to update	
	6. System provide access to the admin's product management capabilities and presents the current details of the selected product.
7. Admins modify, add or update product information such as name, description, price, image etc and admin press the update button.	

	8. System validates the changes made by admin and processes the submitted changes and updates the product details in the database.
	9. System displays a confirmation message for the successful update.

Table 61: Expanded Level Use Case Diagram: Add Product

Alternative Flow (Error Handling)

- 7. System displays error message indicating the issue and admin is prompted to try the update again later.
- 8. System displays error message indicating the specific validation issue and admin is prompted to correct the errors and resubmit the updated information.

7.4.1.2.14: EXPANDED LEVEL USE CASE DESCRIPTION OF DELETE PRODUCT**Name:** Delete Product**Actor(s):** Admin**Purpose:** Allow admin to remove existing products from the platform.**Overview:** Admins can use the product management interface to delete product that are no longer available or relevant.**Type:** Primary, Essential

Actor Action	System Response
1. Admin open the web browser and enters URL and logs in using admin credentials.	
3. Admin navigates the product management interface.	2. System displays Login Page
5. Admin views a list existing product	4. System provide access to the admin's product management capabilities.
7. Admin select specific product to delete and confirms the deletion of the product.	6. System displays a list of products available on the platform.
	8. System verifies the admin's intent to delete the product.
	9. System removes the product from the database and from the platform.

	10. System sends a deletion message to the admin.
--	---

Table 62: Expanded Level Use Case Diagram: Delete Product

7.4.1.2.15: EXPANDED LEVEL USE CASE DESCRIPTION OF ADD CATEGORIES

Name: Add Categories

Actor(s): Admin

Purpose: Allow administrators to add new categories for organizing products on the platform.

Overview: Admins can use the product management interface to create and manage product categories, facilitating better organization and navigation for users.

Type: Primary, Essential

Action Steps

Actor Action	System Response
1. Admin opens the web browser and enters the URL into the web browser	
3. Admin logs in using their credentials	2. System displays Login Page
5. Admin navigates to the product management interface and select the product to update	4. System validates credentials. If the credentials are valid the system directs the admin to their dashboard.
	6. System provide access to the admin's product management capabilities and

	presents the current details of the selected product.
7. Admins modify, add or update product information such as name, description, price, image etc and admin press the update button.	
	8. System validates the changes made by admin and processes the submitted changes and updates the product details in the database.
	9. System displays a confirmation message for the successful update.

Table 63: Expanded Level Use Case Diagram: Add Categories

Alternative Flow (Error Handling)

7. System provide confirmation message before clicking to the button.

7.4.1.2.16: EXPANDED LEVEL USE CASE DESCRIPTION OF DELETE CATEGORIES**Name:** Delete Categories**Actor(s):** Admin**Purpose:** Allow admin to delete existing categories from the system.**Overview:** Admins can use the category management interface to select and delete categories that are no longer relevant or necessary for organizing products. This action involves the removal of a category, along with its associated products, from the system.**Type:** Primary, Essential

Actor Action	System Response
1. Admin open the web browser and enters URL and logs in using admin credentials.	
	2. System displays Login Page
3. Admin navigates to the category management interface.	
	4. System provides list of categories available on the platform.
	5. System present option to confirm the deletion of selection category.
6. Admin confirms the deletion of the category	
	7. System identifies the intent of admin and send confirmation message to the admin.
	8. System deletes the category from the database.

Table 64: Expanded Level Use Case Diagram: Delete Categories

Alternative Flow (Error Handling)

10. Admin receives the confirmation message that the category has been successfully deleted.
11. System displays error message if the category if deletion process became failure.

7.4.1.2.17: EXPANDED LEVEL USE CASE DESCRIPTION OF MANAGE PAYMENTS

Name: Manage Payments

Actor(s): Admin

Purpose: Allow administrators to manage and view payment transactions on the platform.

Overview: Admins can access the payment management interface to view transaction details, track payments, and manage payment-related issues.

Type: Primary, Essential

Actor Action	System Response
1. Admin open the web browser and enters URL and logs in using admin credentials.	
	2. System displays Login Page
3. Admin navigates to the payment management interface.	
	4. System provide access to the admin's payment management capabilities.
5. Admin views a list of payment transactions and can select individual payment transactions.	
	6. System displays a list of recent payment transactions on the platform.

	7. System present detailed information about the payment transactions including user details, amount etc.
--	---

Table 65: Expanded Level Use Case Diagram: Manage Payments

[GO TO TOP](#)

7.5.2. SEQUENCE DIAGRAM

To create a suitable sequence diagram, the following steps must be performed illustrate how a function interacts with its components.

Step 1: Defining the Domain Objects and their respective Lifelines.

A domain object is an entity in the domain layer of an application, depicted with its lifeline in a use case, where the lifeline is a named element representing a participant.

Step 2: Creation of Control Object.

A control manages and schedules the operations and components of other system. A control object, as the instance of the class, is given the same name as in the use case and is depicted with its lifeline.

Step 3: Initialization of Boundary Object.

The information or knowledge of a system is portrayed through an entity, a standardized object representing a storage or persistence mechanism. To incorporate the user interface, one adjusts a boundary object with the use case's name and draws its lifeline.

7.5.2.1. Sequence Diagram of All Users: Registered Users

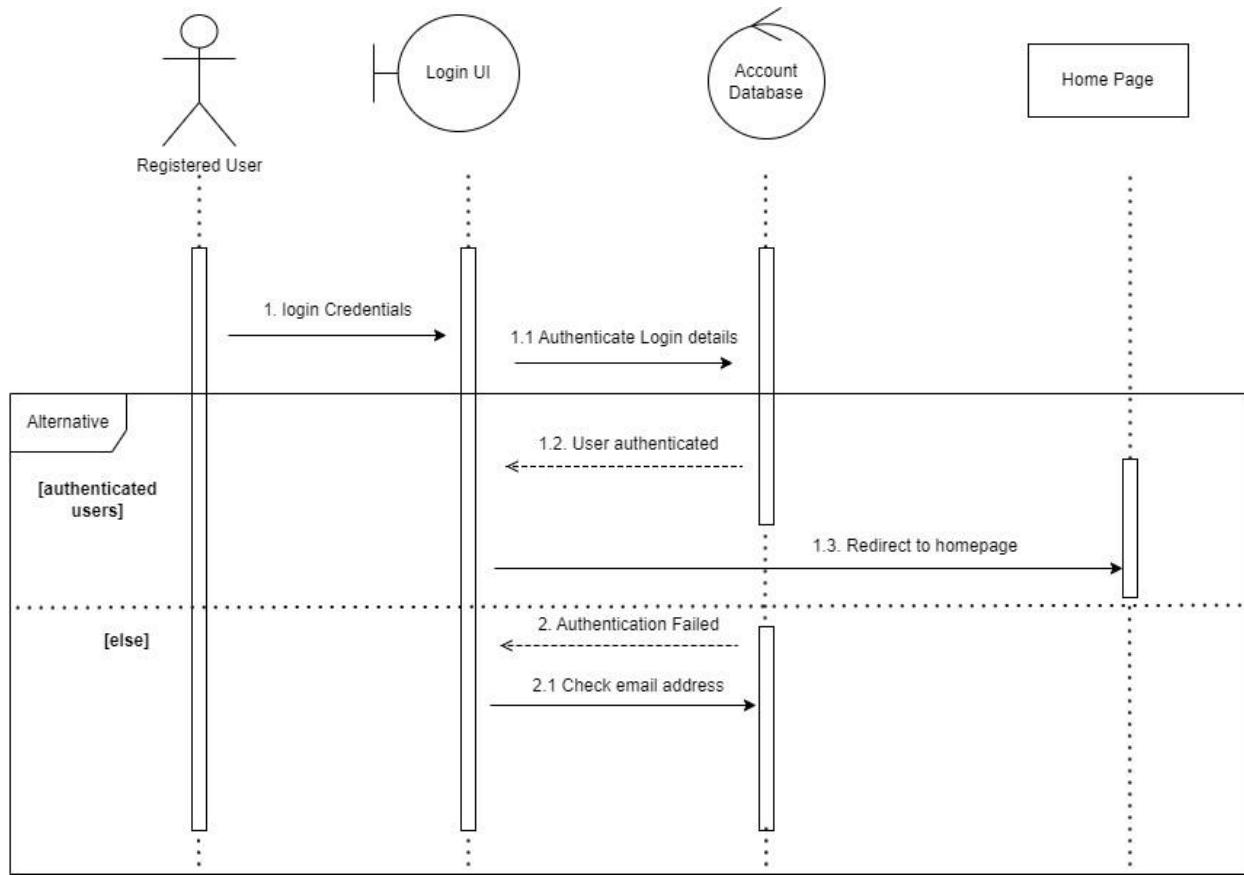


Figure 316: Sequence Diagram: Registered Users

7.5.2.2. Sequence Diagram: Admin Portal

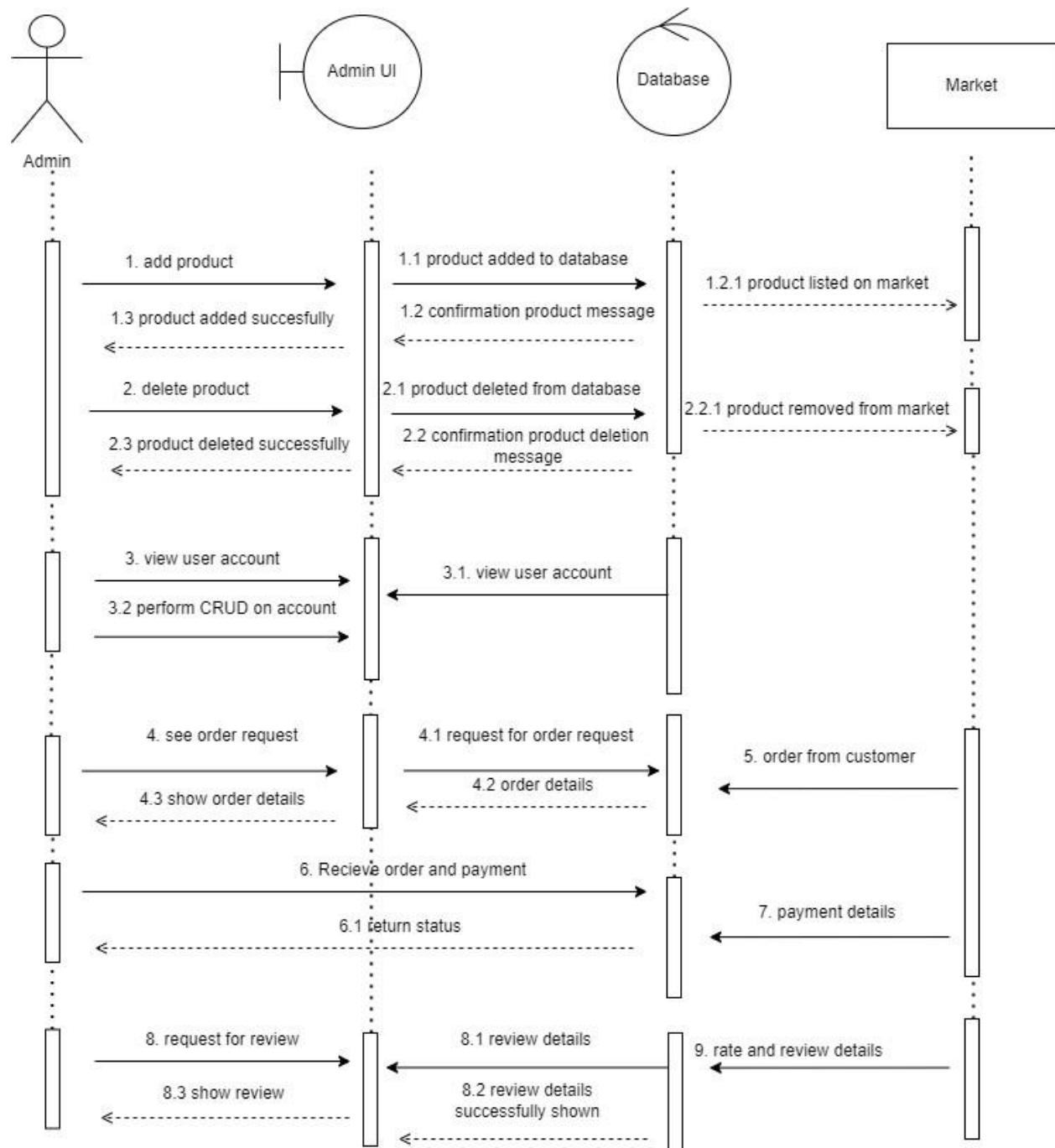


Figure 317: Sequence Diagram: Admin Portal

7.5.2.3. Sequence Diagram: All Users

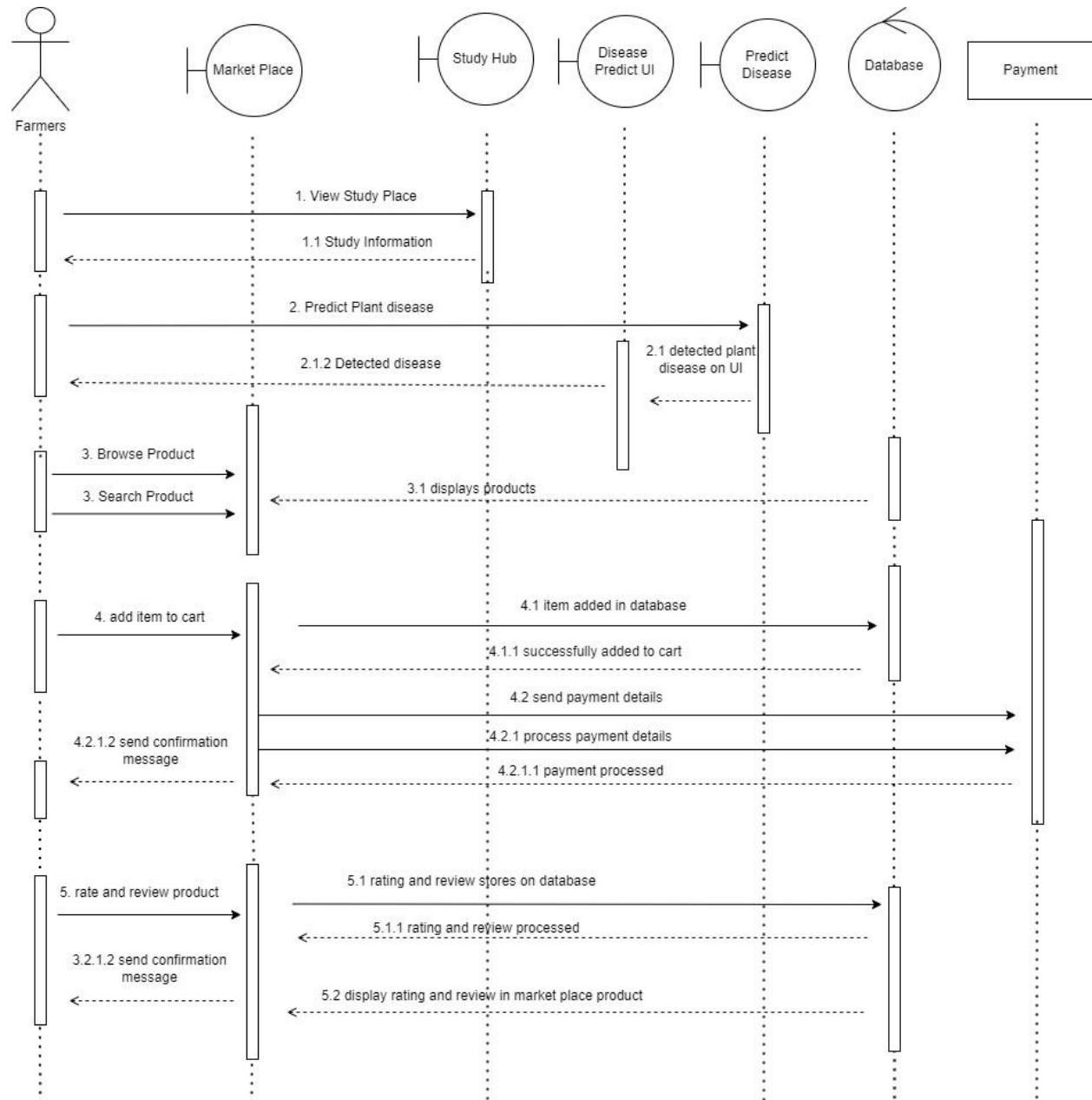


Figure 318: Sequence Diagram: All Users (Farmers)

7.5.2.4. Sequence Diagram All Users: Edit Profile

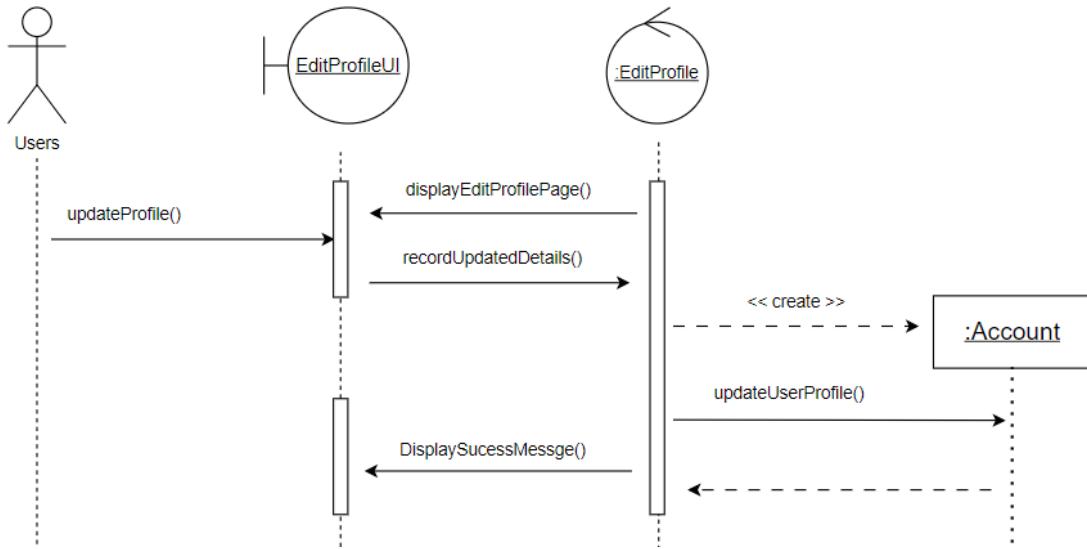


Figure 319: Sequence Diagram for All Users: Edit Profile

7.5.2.5. Sequence Diagram All Users: Change Password

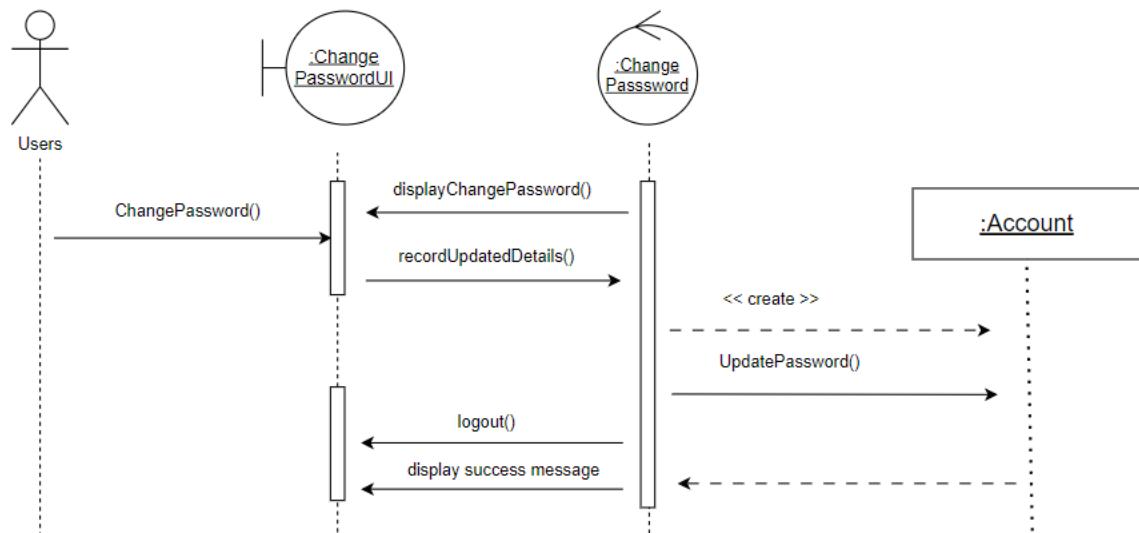


Figure 320: Sequence Diagram for All Users: Change Password

7.5.2.6. Sequence Diagram Admin/User: Logout

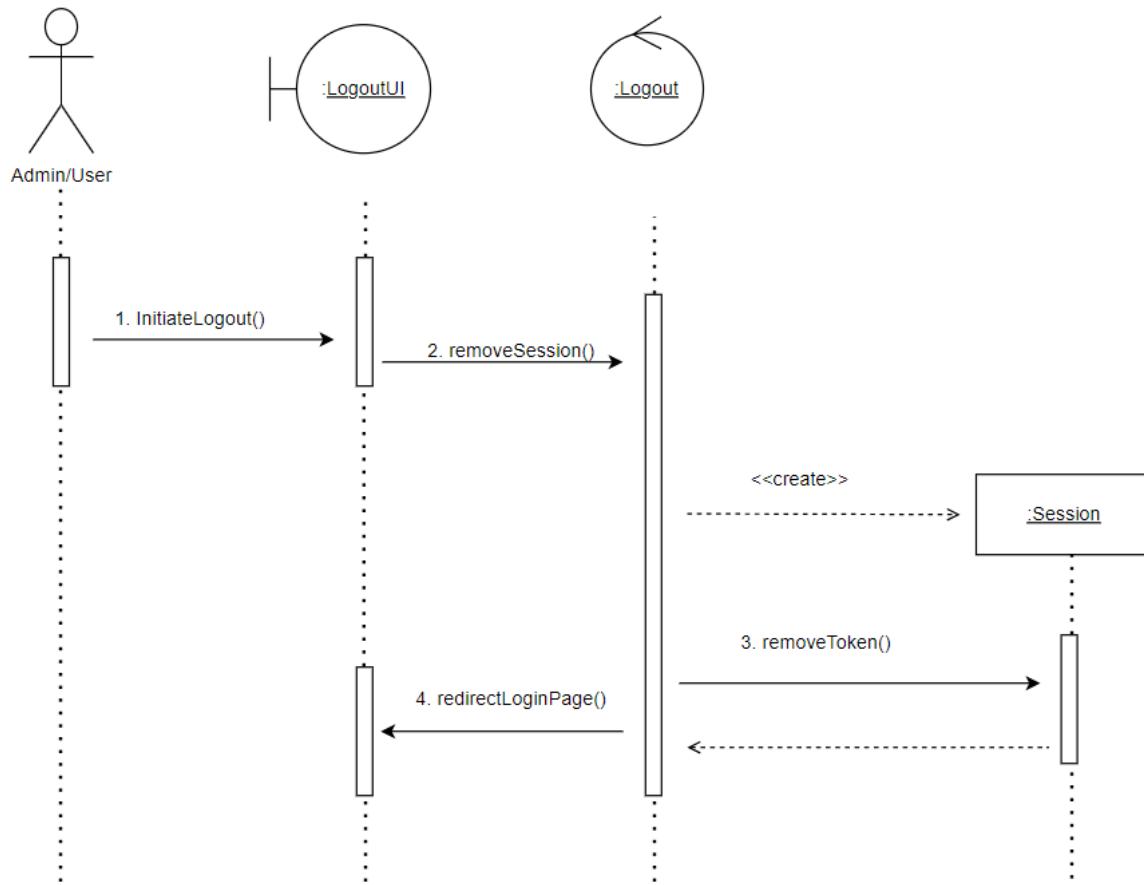


Figure 321: Sequence Diagram Admin/User: Logout

7.5.2.7. Sequence Diagram: Browse Product

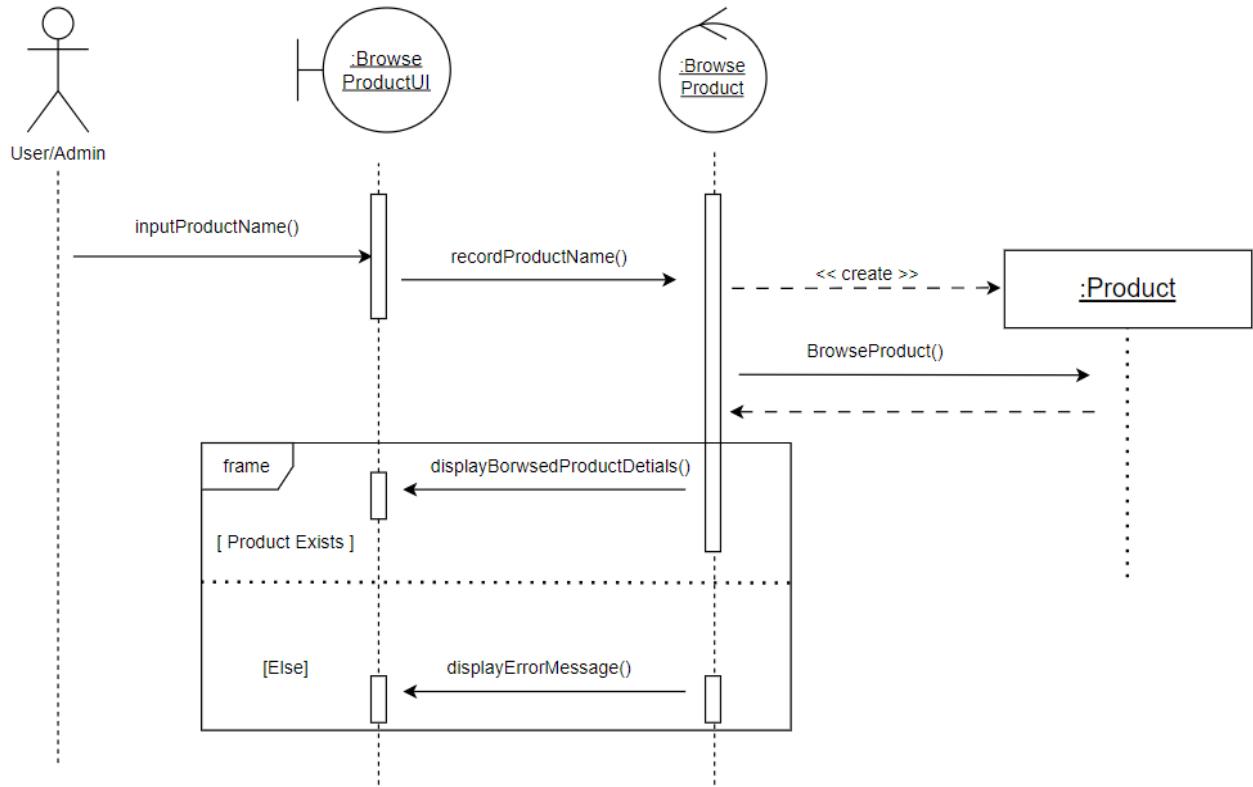


Figure 322: Sequence Diagram: Browse Product

7.5.2.8. Sequence Diagram All Users: Add to Cart

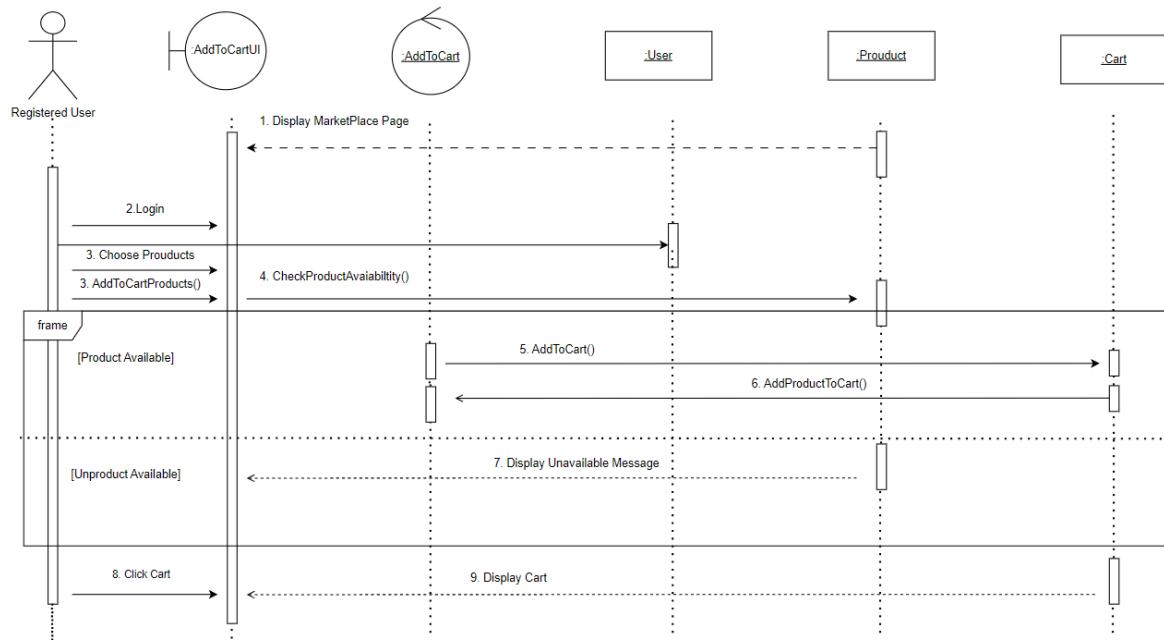


Figure 323: Sequence Diagram All Users: Add to Cart

7.5.2.9. Sequence Diagram All Users: Payment

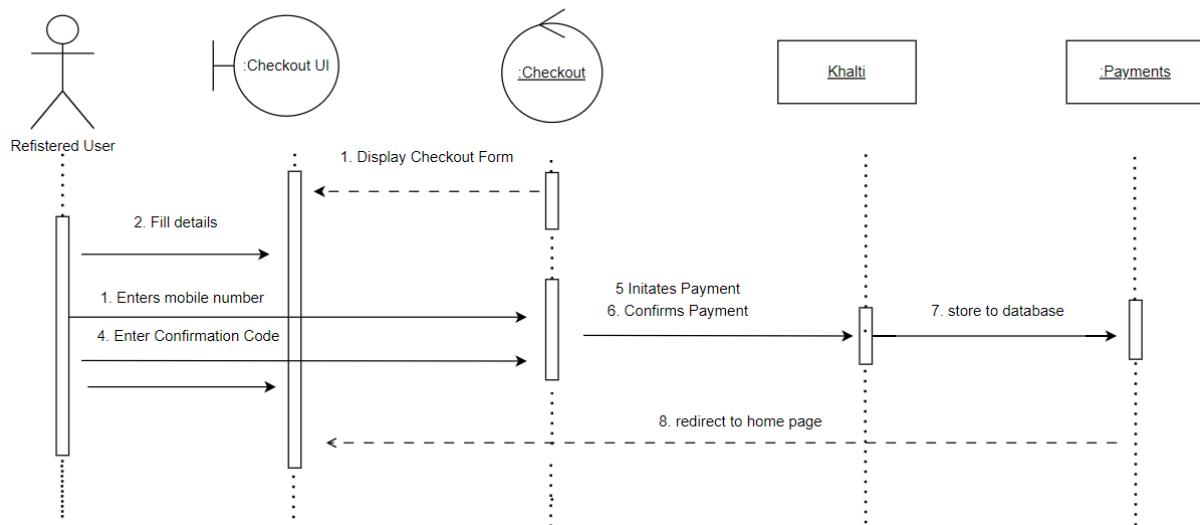


Figure 324: Sequence Diagram: Payment

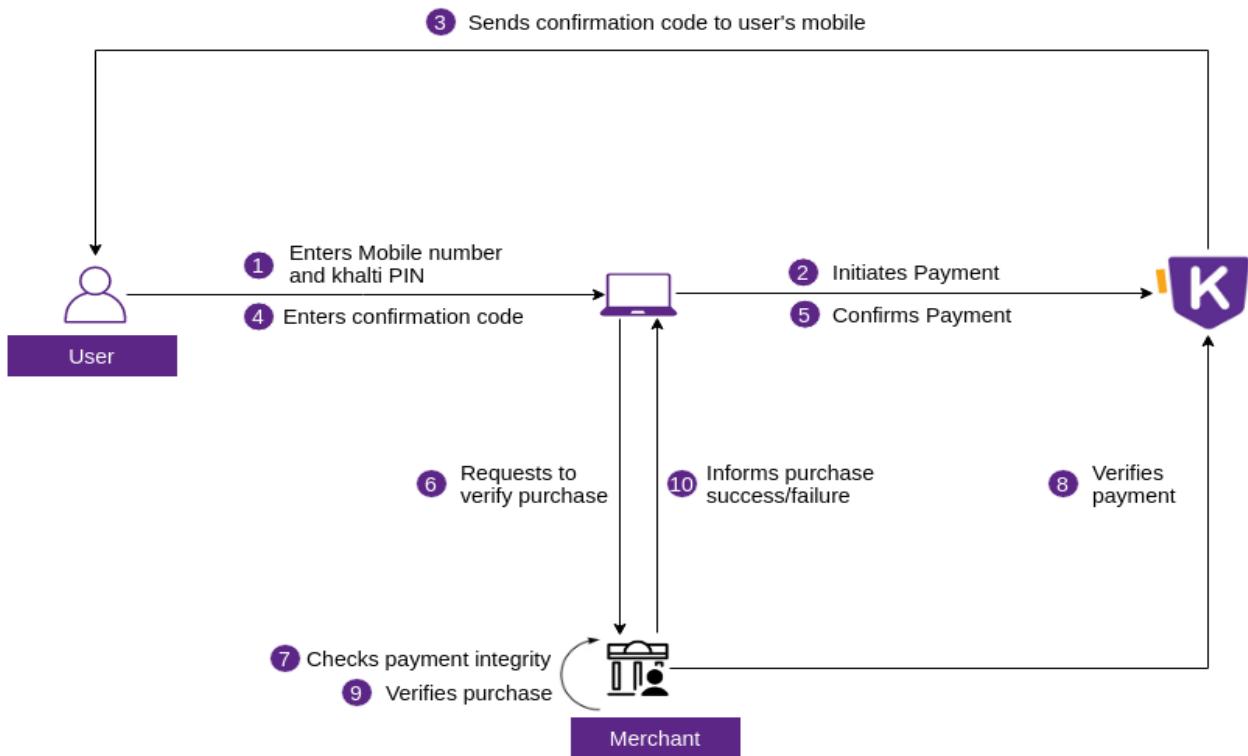


Figure 325: Khalti Payment Flow

7.5.2.10. Sequence Diagram All Users: Prediction

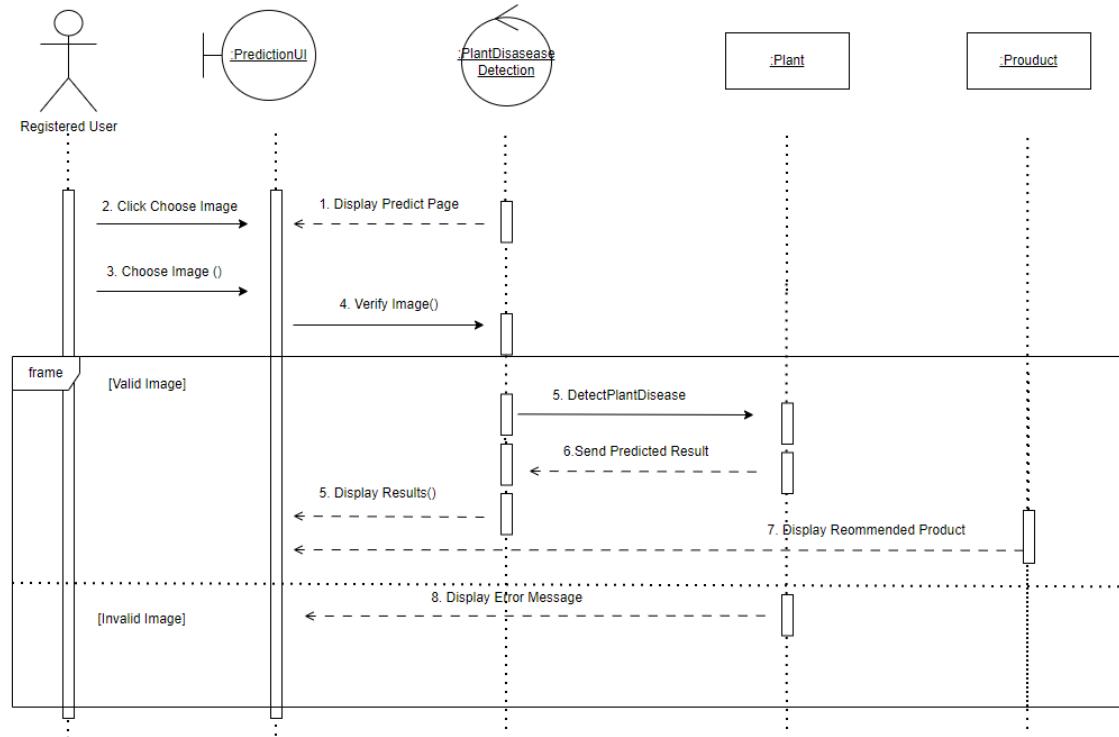


Figure 326: Sequence Diagram All Users: Prediction

[GO TO TOP](#)

7.5.3. ACTIVITY DIAGRAM

7.5.3.1. ACTIVITY DIAGRAM: REGISTERED USERS

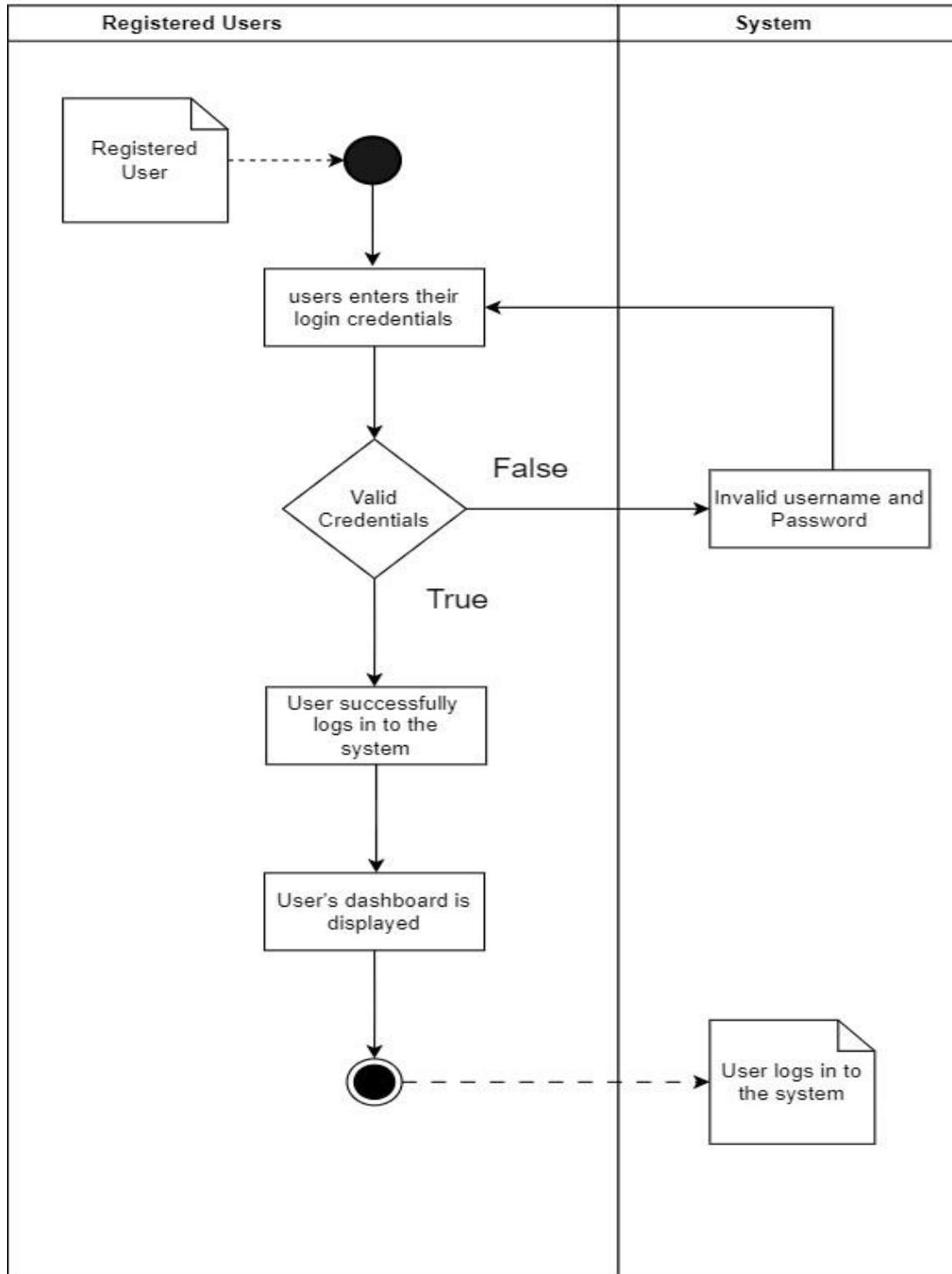


Figure 327: Activity Diagram: Registered Users

7.5.3.2. ACTIVITY DIAGRAM: USER ACCOUNT FUNCTIONALITY

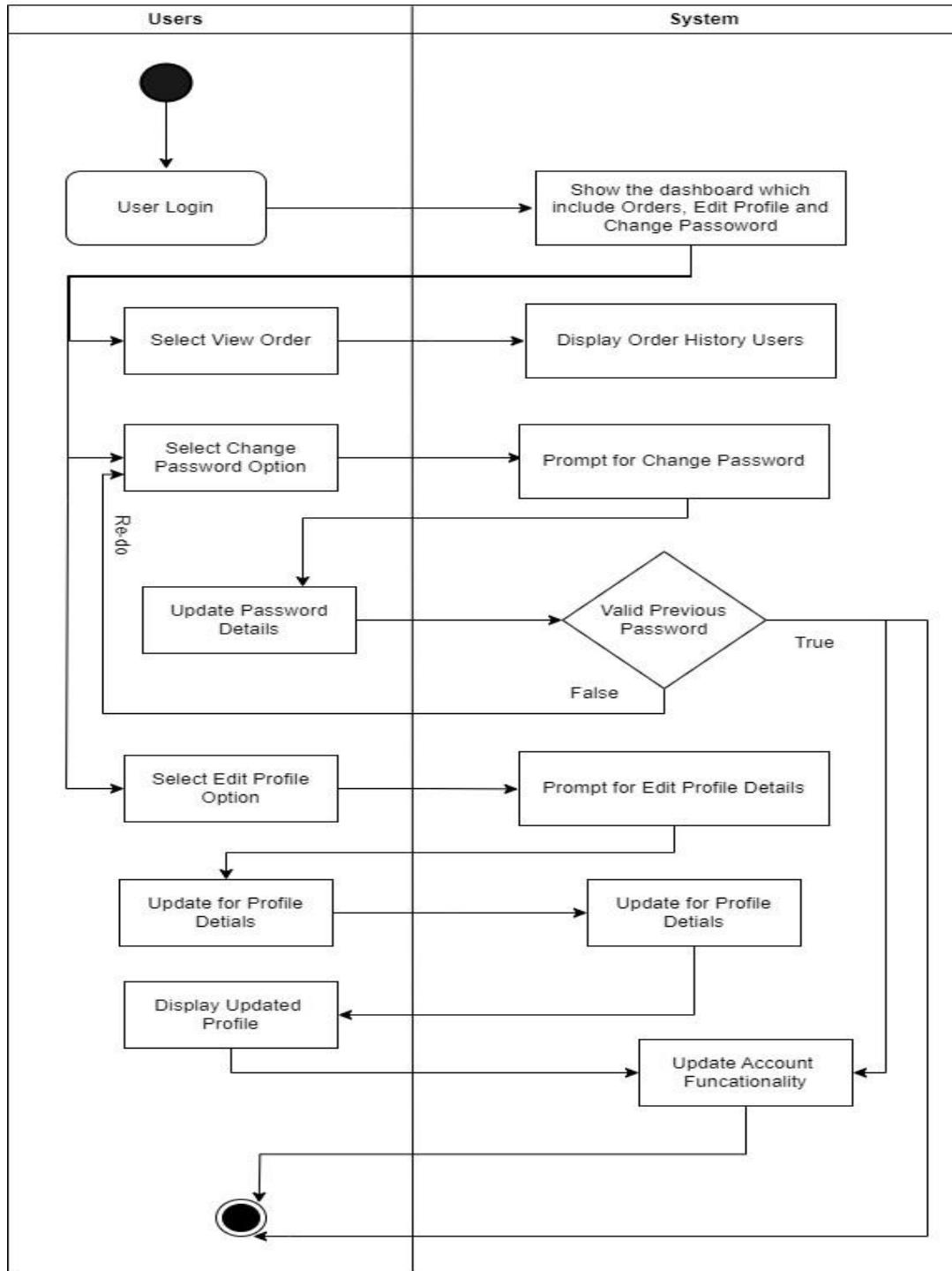


Figure 328: Activity Diagram: All Users Functionality

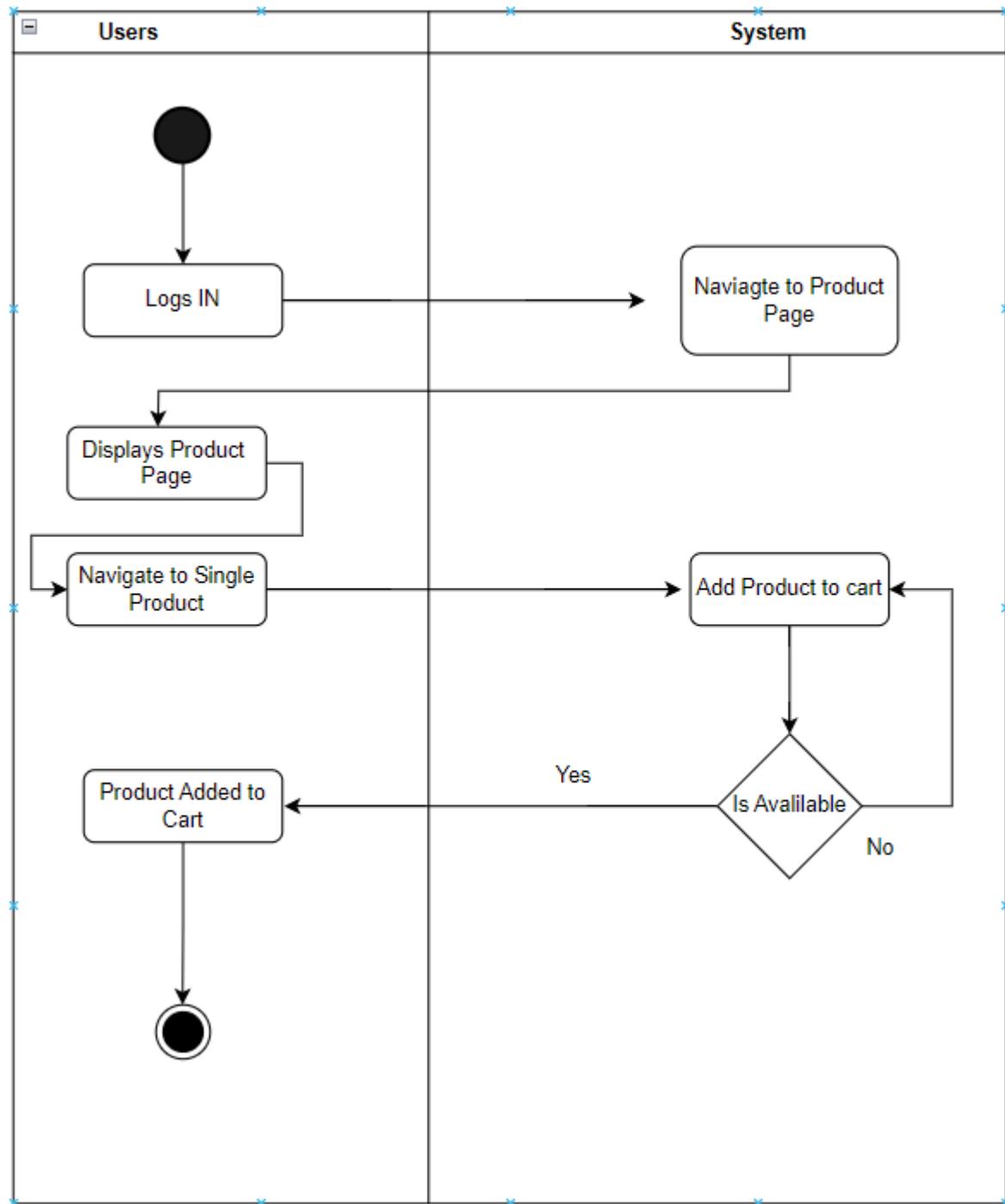
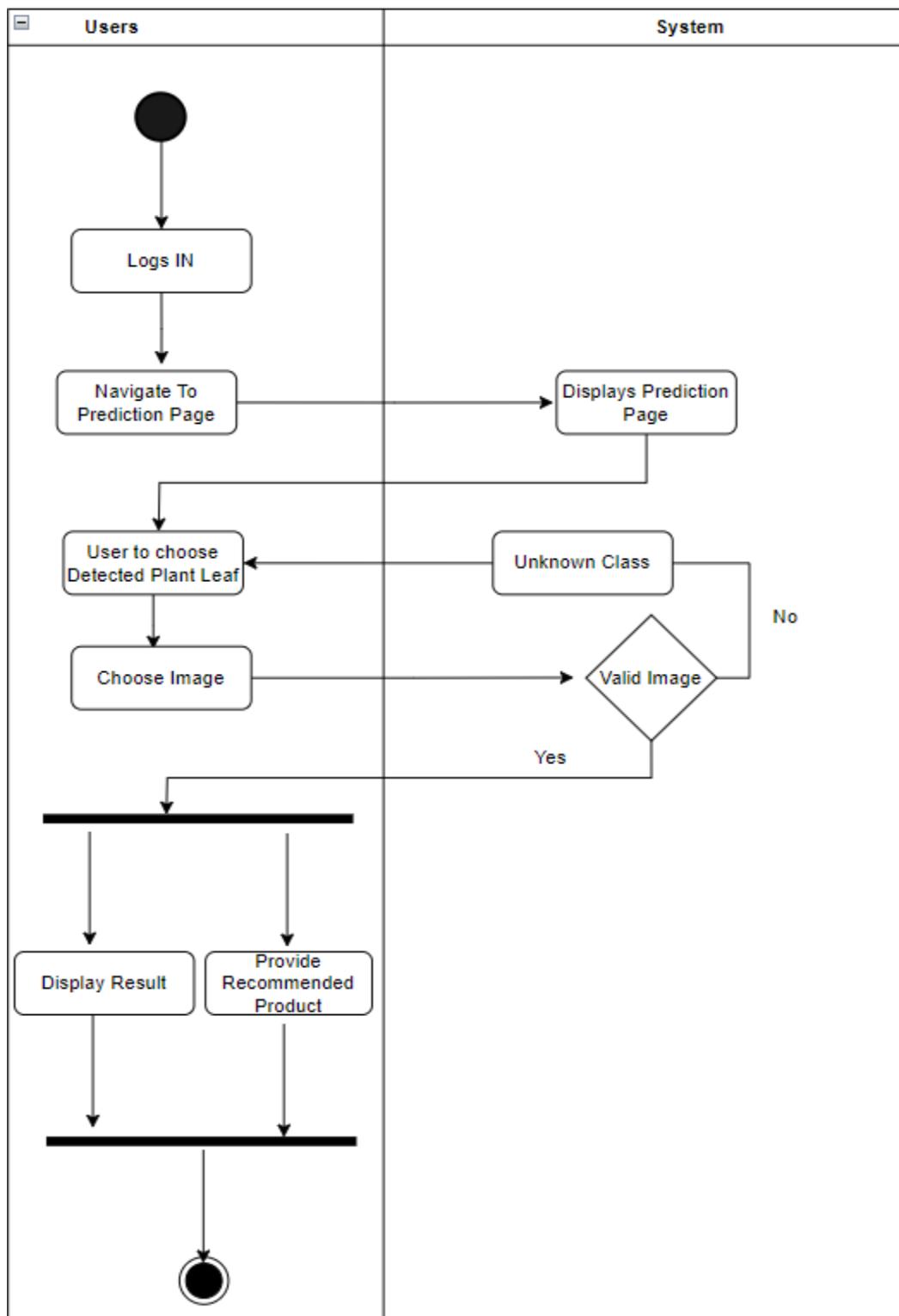
7.5.3.3. ACTIVITY DIAGRAM: ADD TO CART

Figure 329: Activity Diagram: Add to Cart

7.5.3.4. ACTIVITY DIAGRAM: PREDICTION PAGE**Figure 330: Activity Diagram: Prediction**

7.5.3.5. ACTIVITY DIAGRAM: PAYMENT

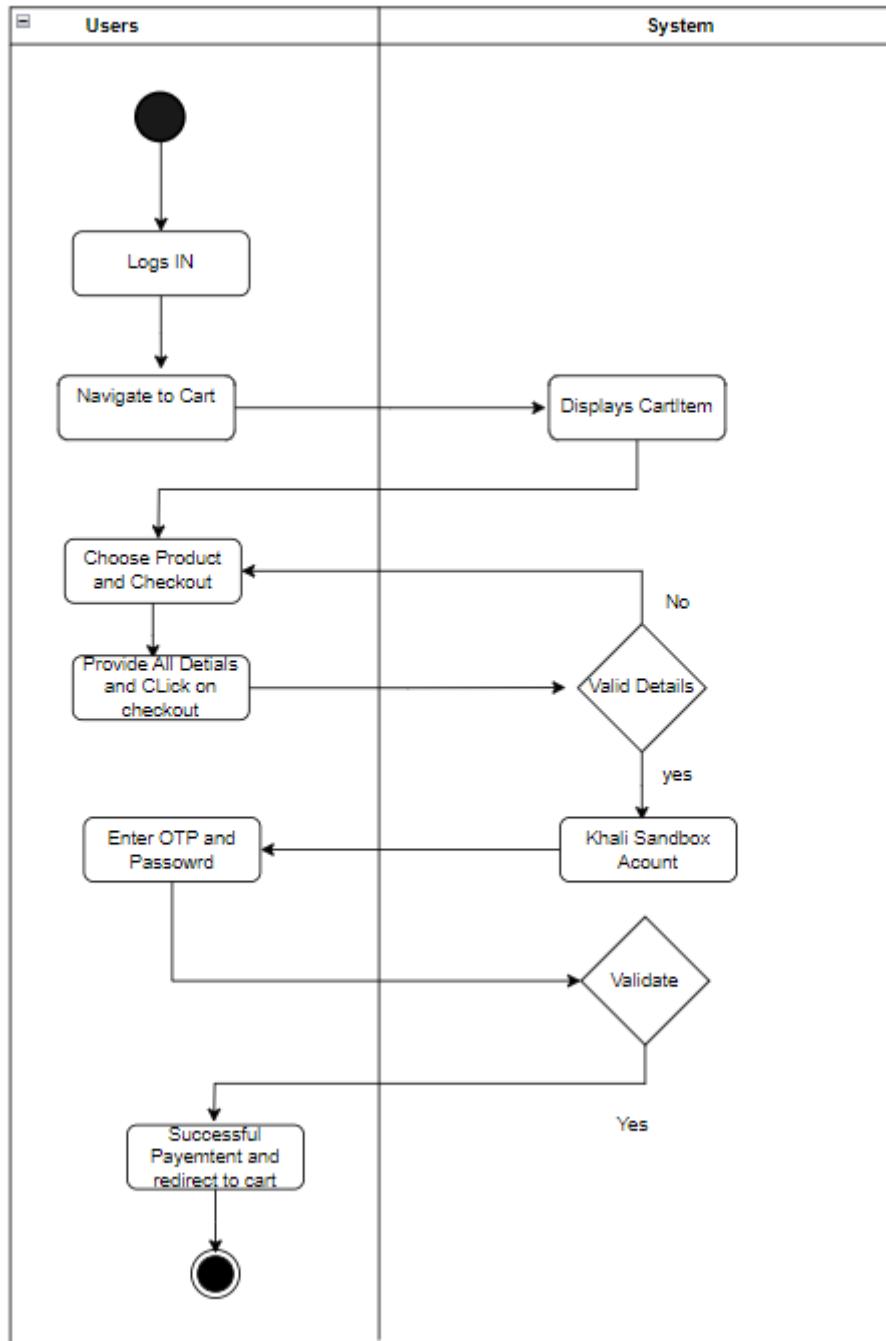


Table 66: Activity Diagram: Payment

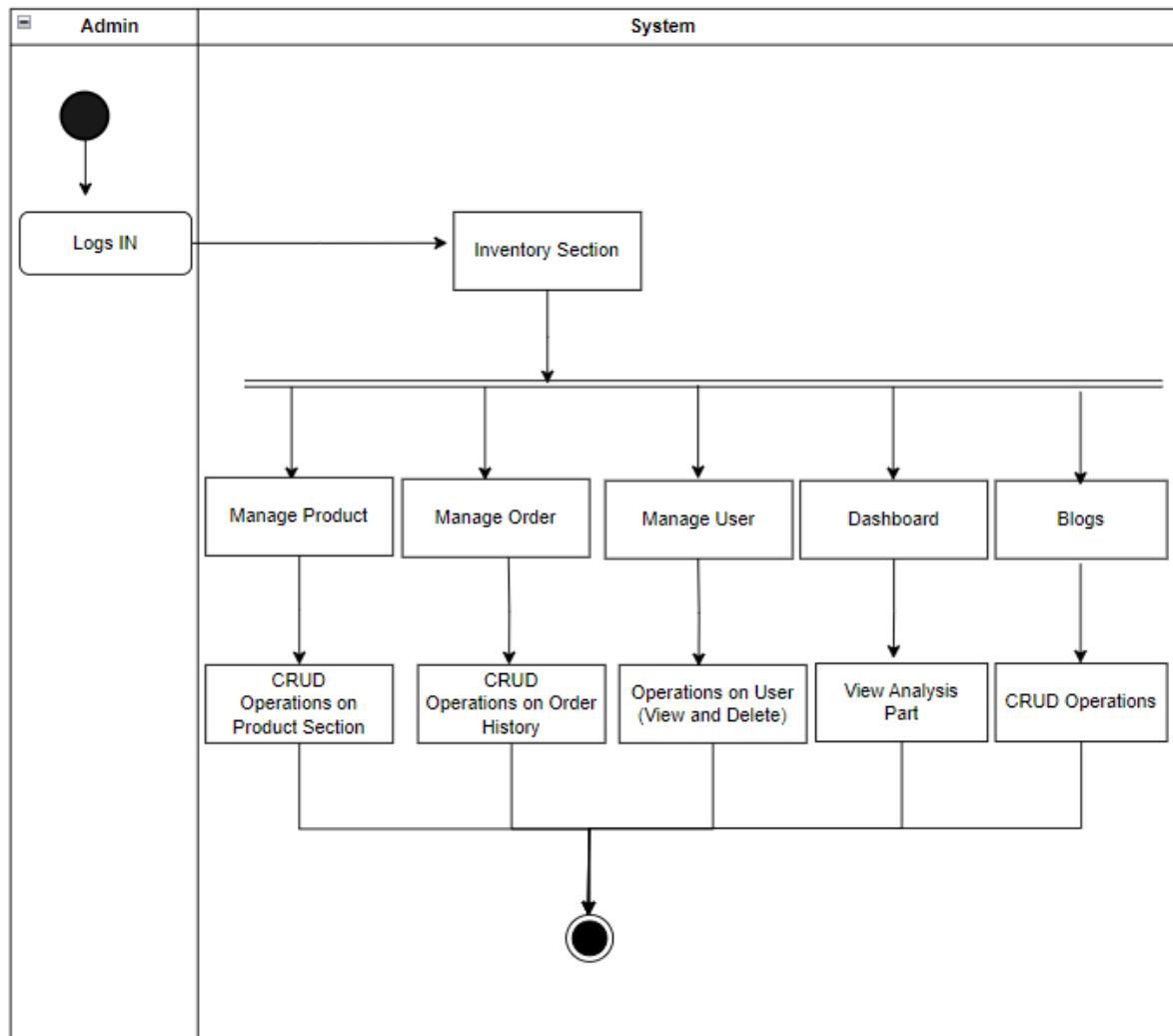
7.5.3.5 ACTIVITY DIAGRAM: ADMIN PORTAL

Figure 331: Activity Diagram: Admin

[GO TO TOP](#)

7.5.4. COLLABORATION DIAGRAM

Notation of Collaboration Diagram

A collaboration diagram looks like a flowchart that shows the roles, functions, and behaviour of individual objects, as well as how the overall system works in real-time. The four main parts of a collaboration diagram are:

Objects: These are shown as rectangles with labels inside. The label follows the format: object name: class name. If an object has a specific property or state that affects the collaboration, this should also be noted.

Actors: These are instances that start the interaction in the diagram. Each actor has a name and role, with one actor starting the whole use case.

Links: These connect objects with actors and are shown as a solid line between two elements. Each link is a place where messages can be sent.

Messages between objects: These are shown as labelled arrows near a link. These messages are communications between objects that give information about the activity and can include a sequence number. (Lewis, 2023)

7.5.4.1. COLLABORATION DIAGRAM: LOGIN USERS

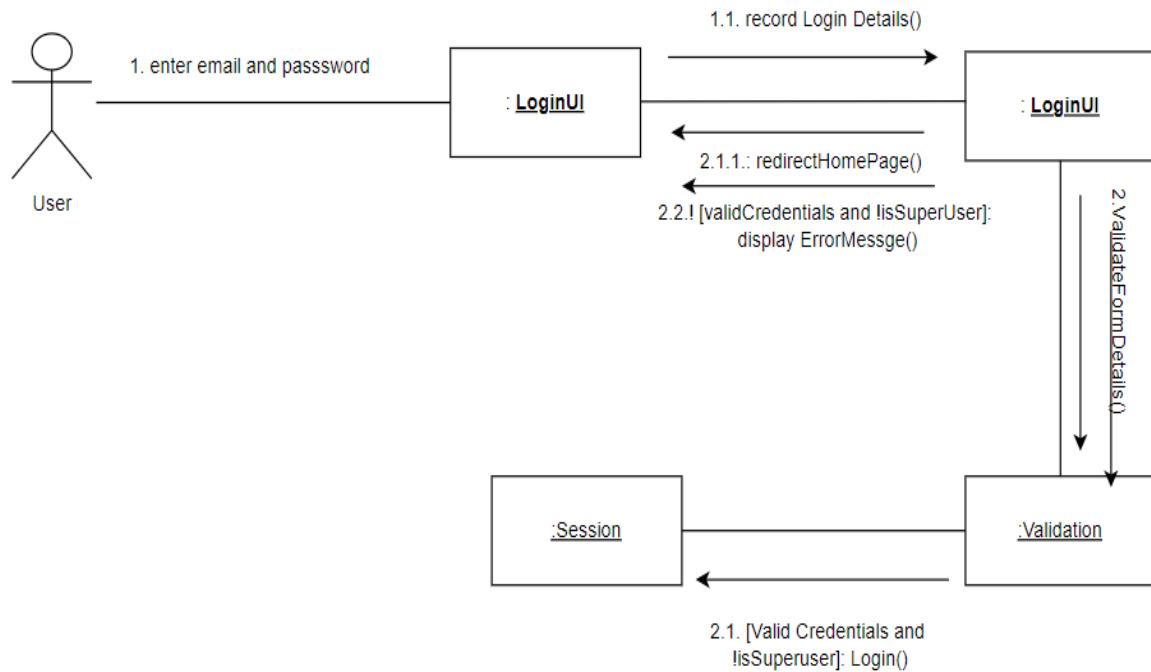


Figure 332: Collaboration Diagram: Login Users

7.5.4.2. COLLABORATION DIAGRAM: LOGOUT USERS

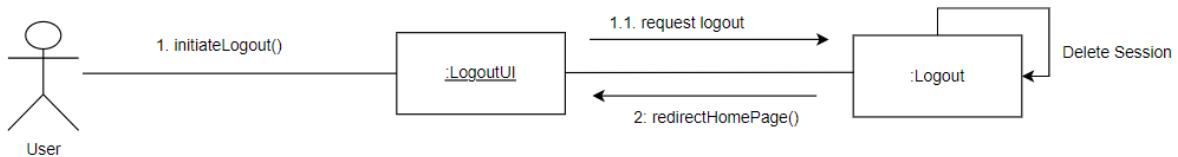


Figure 333: Collaboration Diagram: Logout Users

7.5.4.3. COLLABORATION DIAGRAM: EDIT PROFILE

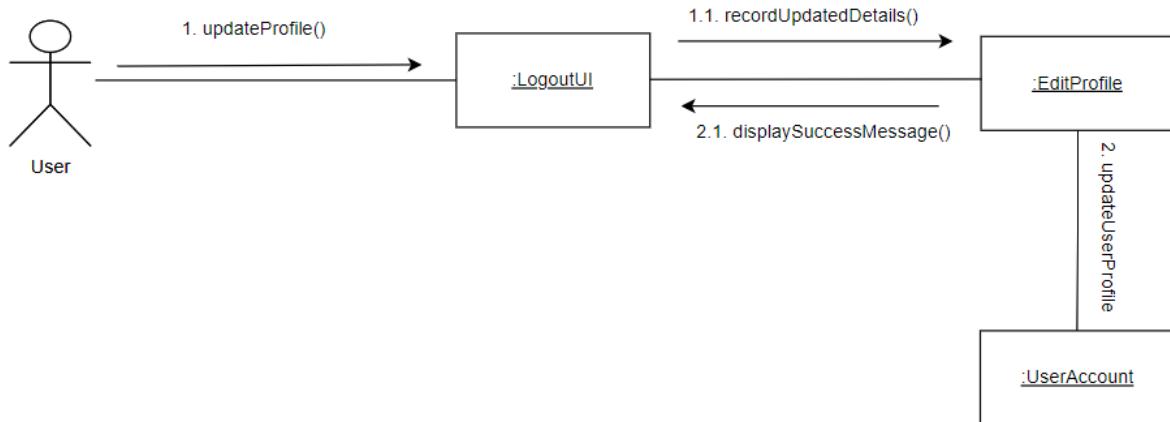


Figure 334: Collaboration Diagram: Edit Profile

7.5.4.4. COLLABORATION DIAGRAM: CHANGE PASSWORD

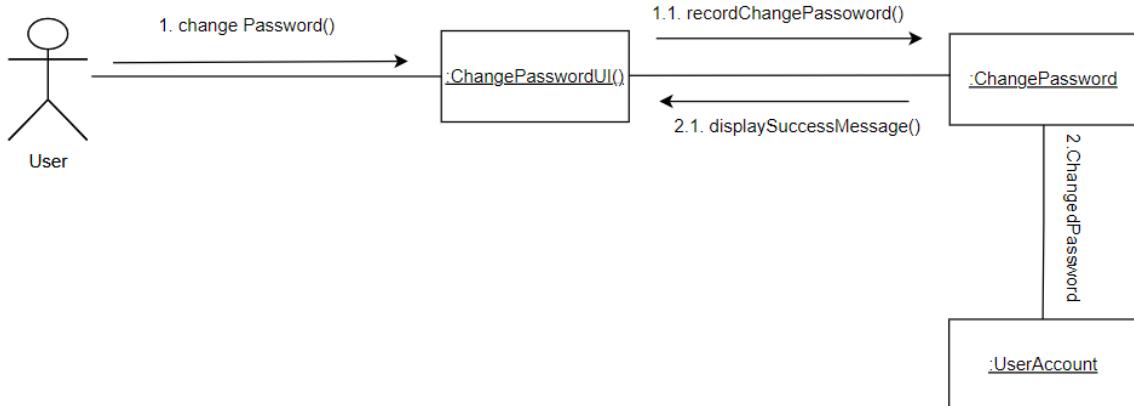


Figure 335: Collaboration Diagram: Change Password

7.5.4.5. COLLABORATION DIAGRAM: BLOGS INFORMATION

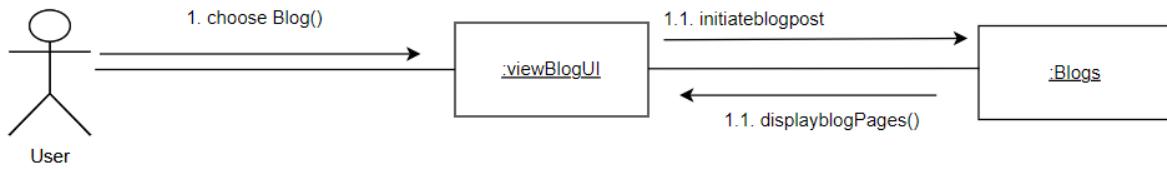


Figure 336: Collaboration Diagram: Blogs Information

7.5.4.6. COLLABORATION DIAGRAM: SEARCH PRODUCTS

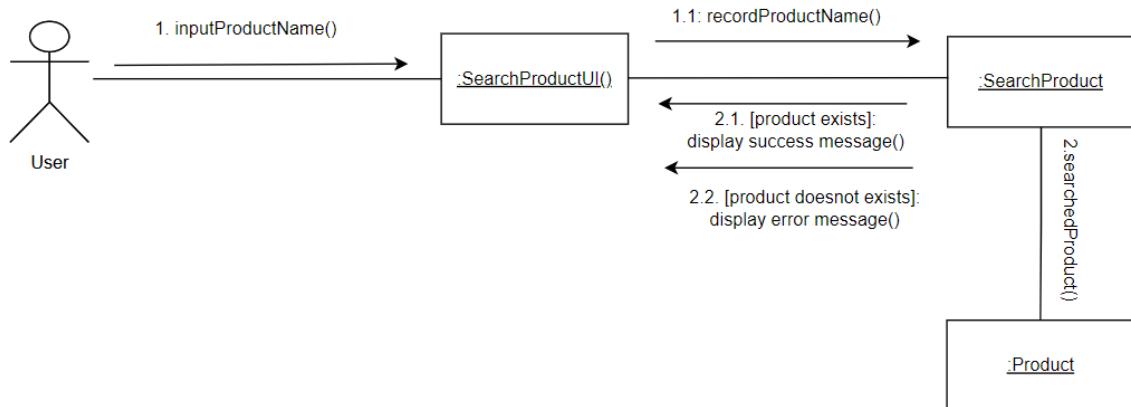


Figure 337: Collaboration Diagram: Search Product

7.5.4.7. COLLABORATION DIAGRAM: RATE REVIEW

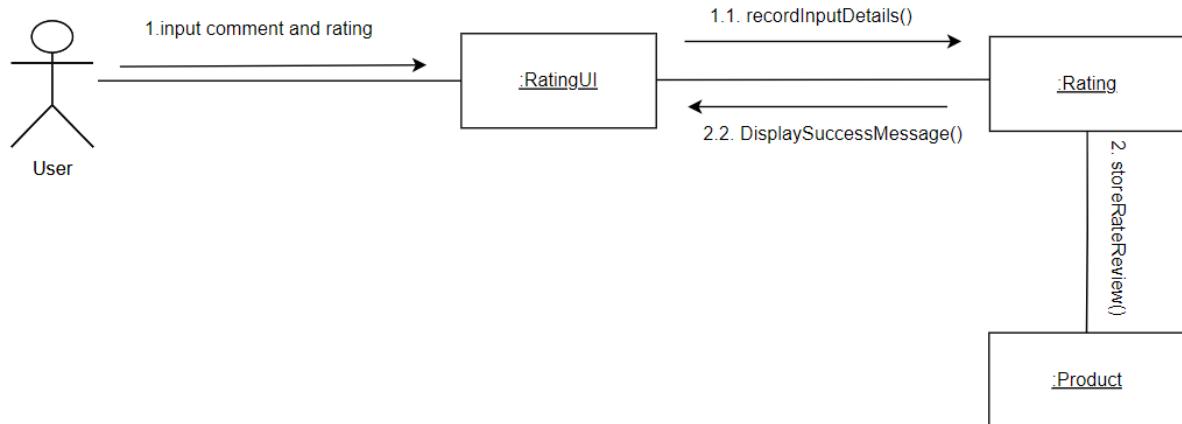


Figure 338: Collaboration Diagram: Rate Review

7.5.4.8. COLLABORATION DIAGRAM: PREDICTION

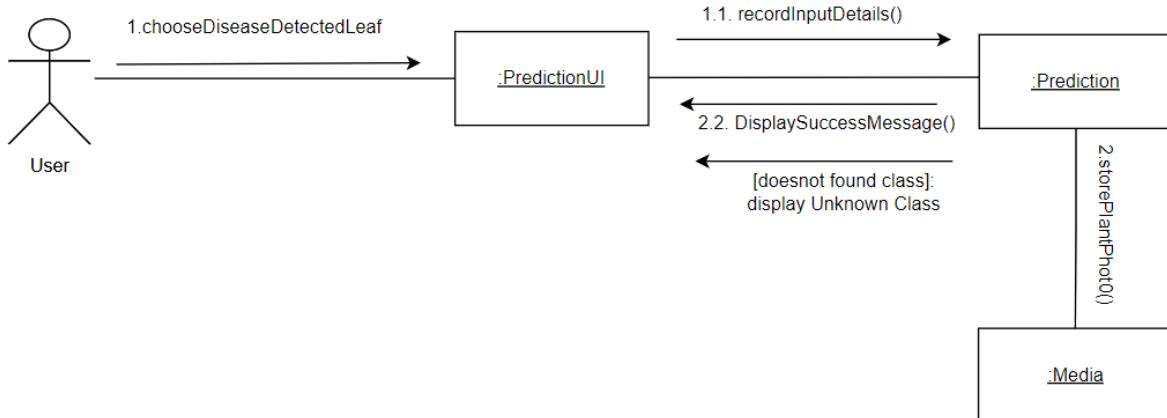


Figure 339: Collaboration Diagram: Prediction

7.5.4.9. COLLABORATION DIAGRAM: PAYMENT

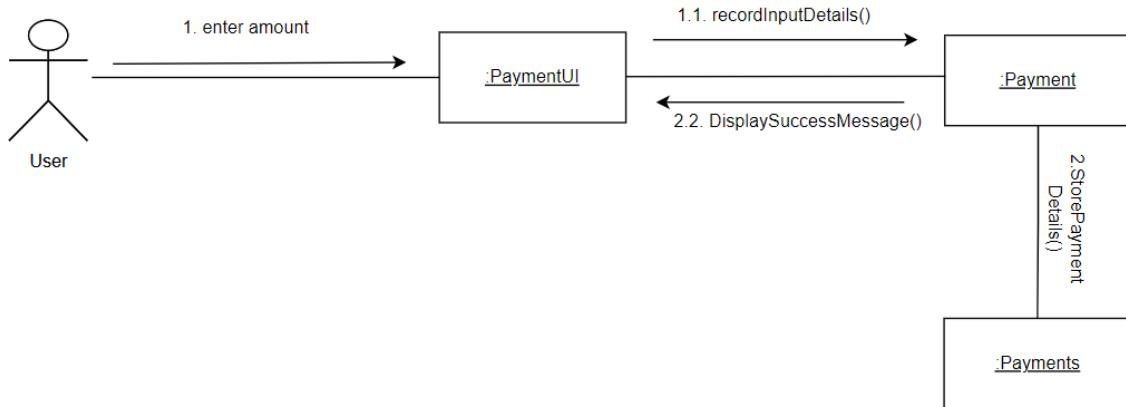


Figure 340: Collaboration Diagram: Payment

7.5.4.10. COLLABORATION DIAGRAM: ADD TO CART

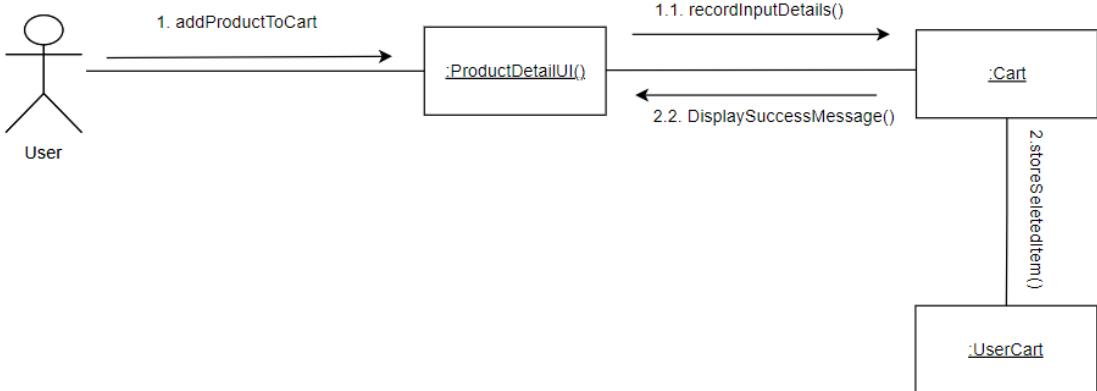


Figure 341: Collaboration Diagram: Add to Cart

7.5.4.11. COLLABORATION DIAGRAM: ADMIN PORTAL

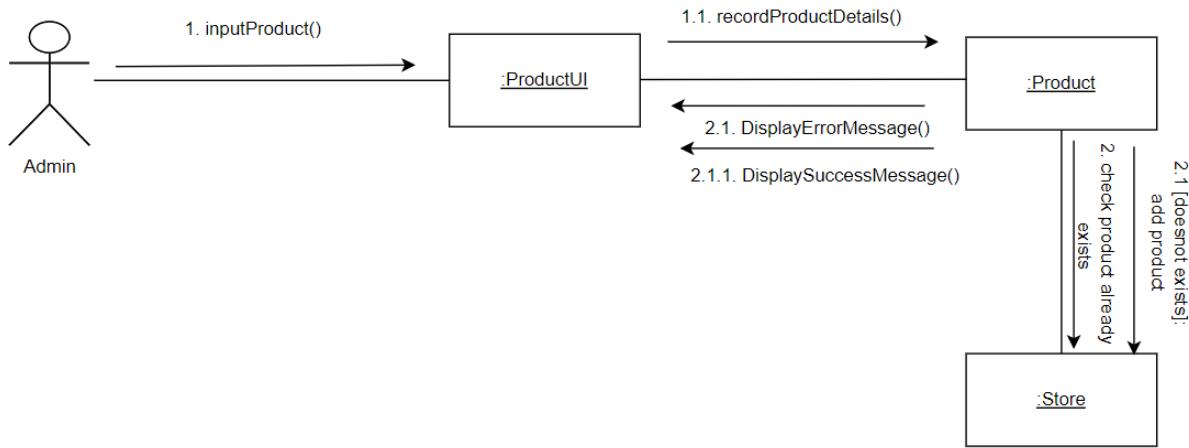


Figure 342: Collaboration Diagram: Admin Portal Add Product

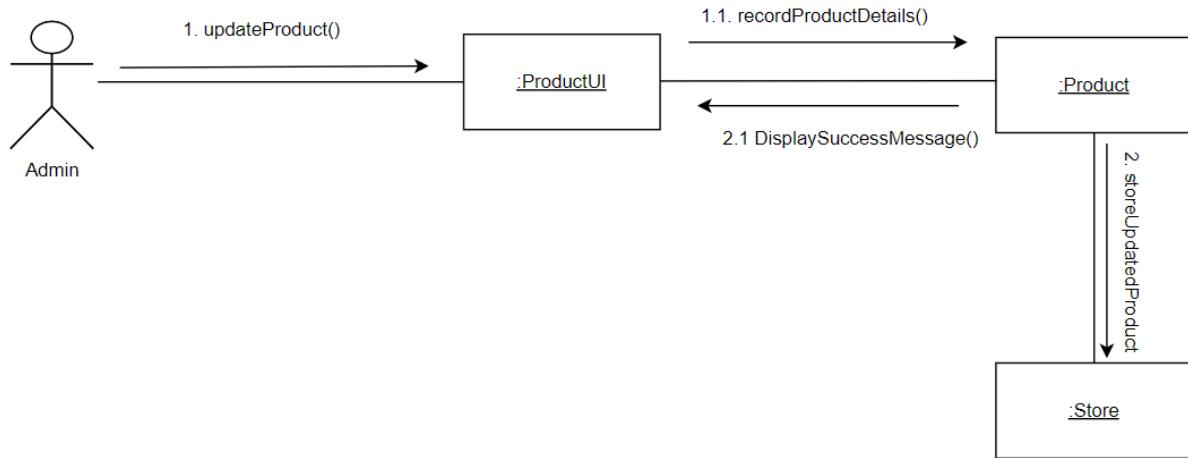
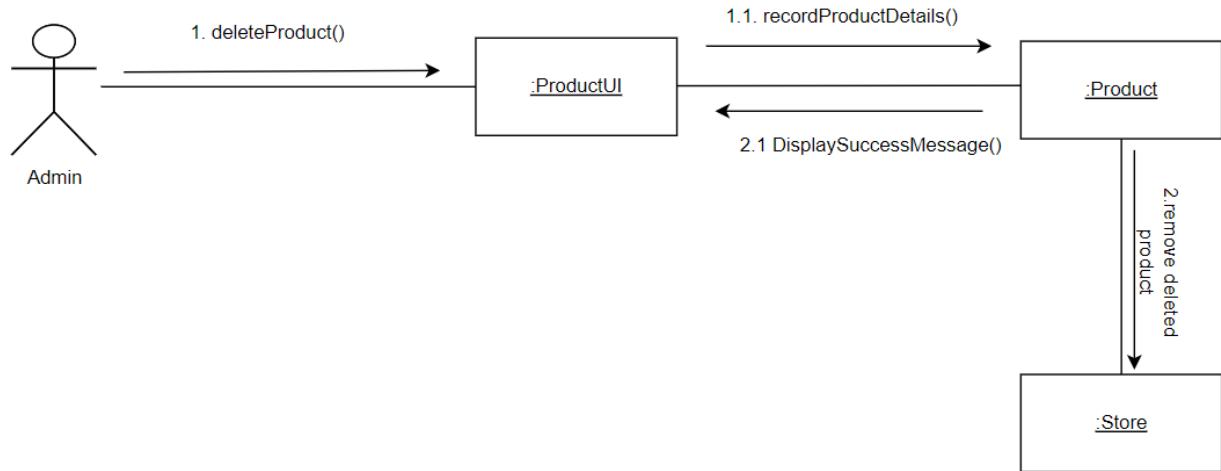
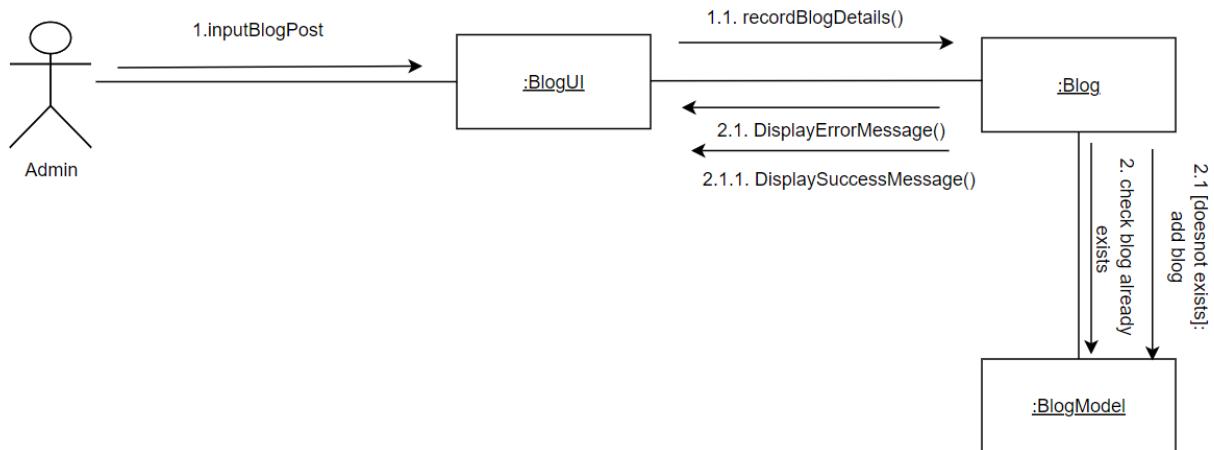
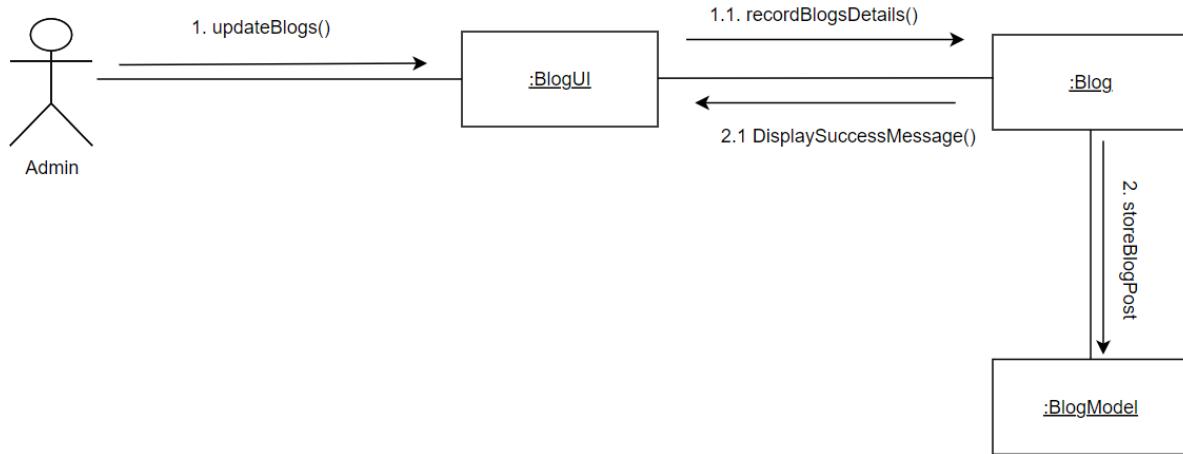
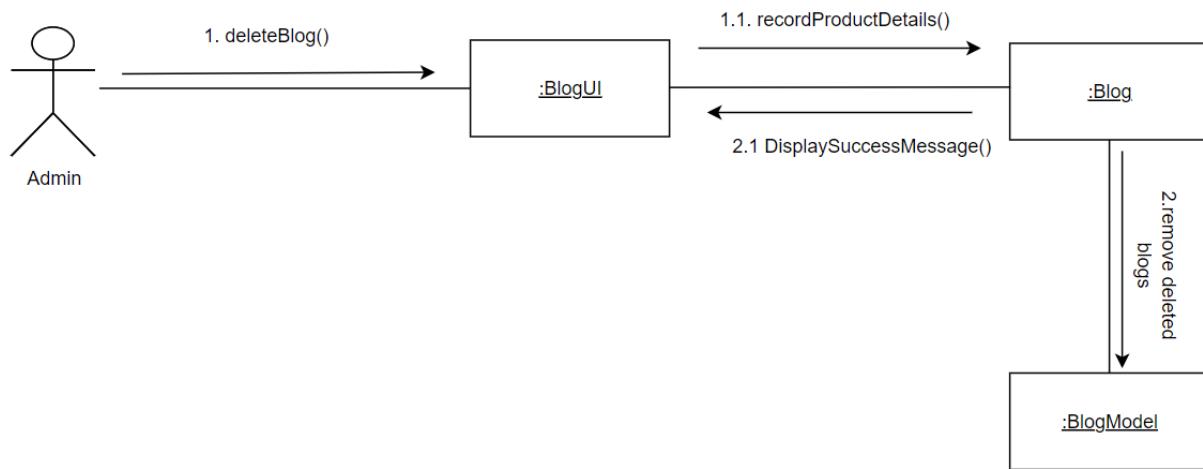


Figure 343: Collaboration Diagram: Admin Portal Update Product

Figure 344: Collaboration Diagram: Admin Portal Delete ProductFigure 345: Collaboration Diagram: Admin Portal Add Blogs Post

*Figure 346: Collaboration Diagram: Admin Portal Update Blog**Figure 347: Collaboration Diagram: Admin Portal Delete*

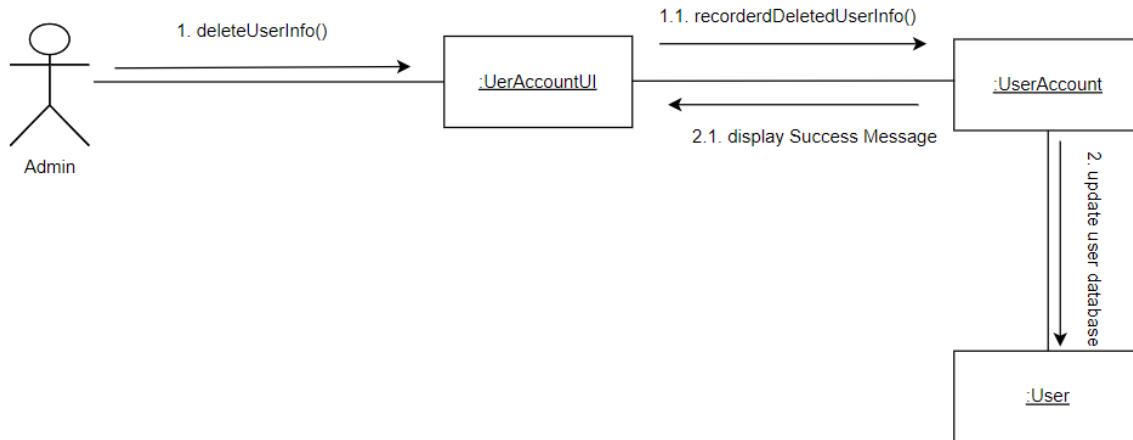


Figure 348: Collaboration Diagram: Admin Portal Manage User Account

[GO TO TOP](#)

7.5.5. IMPLEMENTATIONS

7.5.5.1. SYSTEM ARCHITECTURE DIAGRAM

Software Architecture is a blueprint for a system. It provides an associate abstraction to manage the system complexity and establish a communication and coordination mechanism among elements. It defines a structured resolution to fulfill all the technical and operational necessities, whereas optimizing the common attributes like performance and security. Further, it involves a collection of serious selections regarding the organization associated with software development and each of those selections will have substantial impact on quality, maintainability, performance, and the overall success of ultimate product. (Jaiswal, 2019)

Since the development of the application has already begun, the first step was to define the architecture of the project. This includes following the MVT pattern along with the implementation of System Driven Design with Clean Architecture. Each of these aspects has been described in the following sections:

7.5.5.1.1. APPLICATION ARCHITECTURE (MVT)

When working with Django framework, it's important to follow organized guidelines that shape how the application is structured. Django's use of the MVT architecture pattern helps create complete solutions based on logical connections and consistent principles, making everything work together smoothly.

MVT stands for Model View and Template, and their corresponding explanations are provided below:

- **Model:** The model serves as the data interface, responsible for managing and representing the logical data structure throughout the application. It is implemented through a database (SQLite).
- **View:** View is used to execute the business logic and interact with a model to carry data and renders a template.

- **Template:** A template comprises both static sections of the intended HTML output and specific syntax indicating how dynamic content will be incorporated.

The lists of features provided by Django MVT pattern are as follows:

- Allows easy replacement and modification of components, fostering a modular and flexible structure.
- Logically divides the application into Model, View, and Template components, simplifying the handling of complexity, especially in large projects.
- Promotes test-driven development by facilitating interface-based construction of components, enhancing the overall testability of the program.

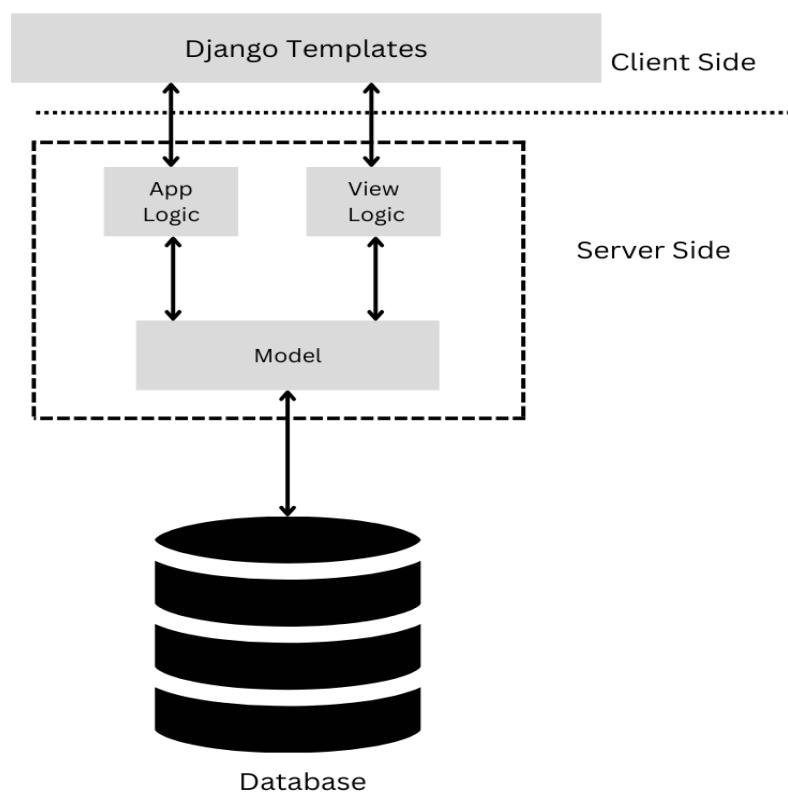


Figure 349: Application Architecture: Model View Template (MVT)

7.5.5.1.2. CLEAN ARCHITECTURE

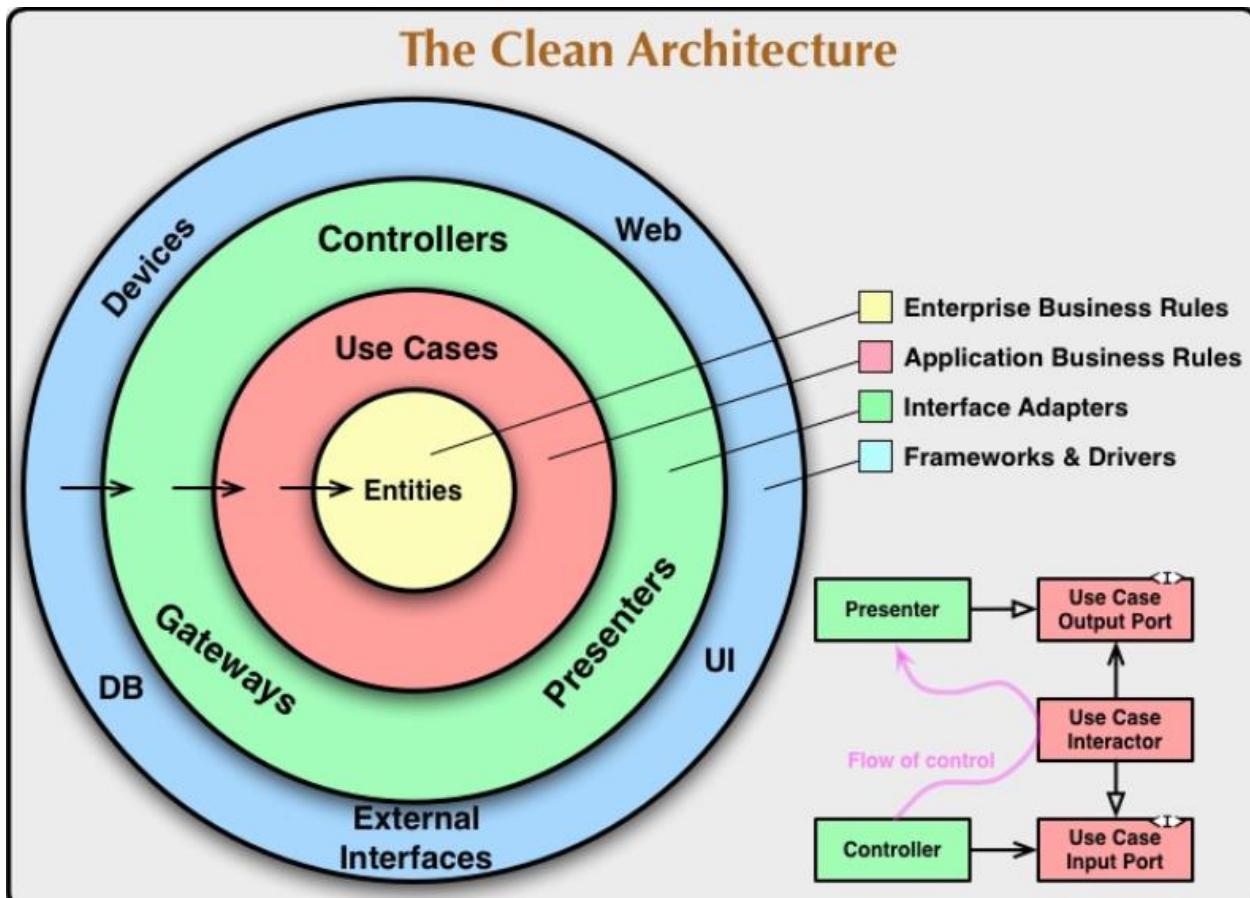
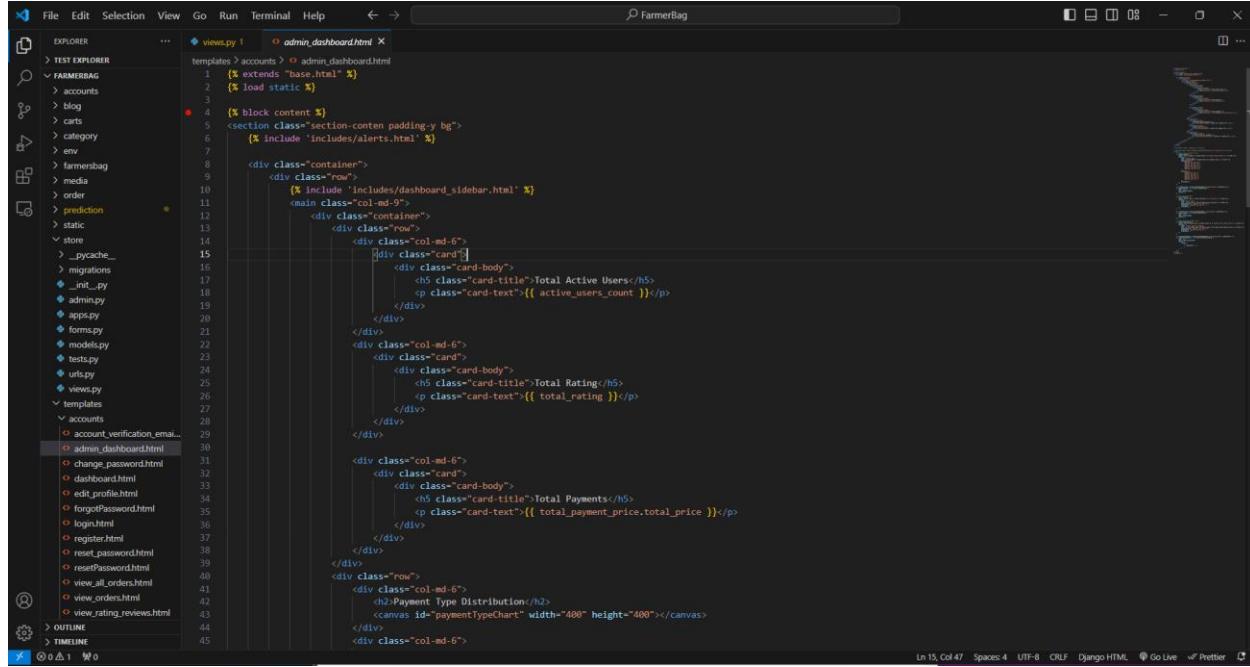


Figure 350: System Architecture Diagram: Clean Architecture

The central objective of clean architecture is to achieve the implementation of use cases with minimal coupling. This involves establishing use cases as a primary organizational structure, which is abstracted from the specific details of frameworks and technologies. In this context, domains represent the central themes or subjects that an application under development revolves around. Clean architecture emphasizes the enduring importance of entities or models that form the foundation of an application, alongside the essential components of business logic, its implementation, and the ensuing best practices.

7.6. APPENDIX F: CODE SAMPLE

7.6.1. SAMPLE CODE OF THE UI



The screenshot shows a code editor interface with the following details:

- File Path:** templates/accounts/admin_dashboard.html
- Content:**

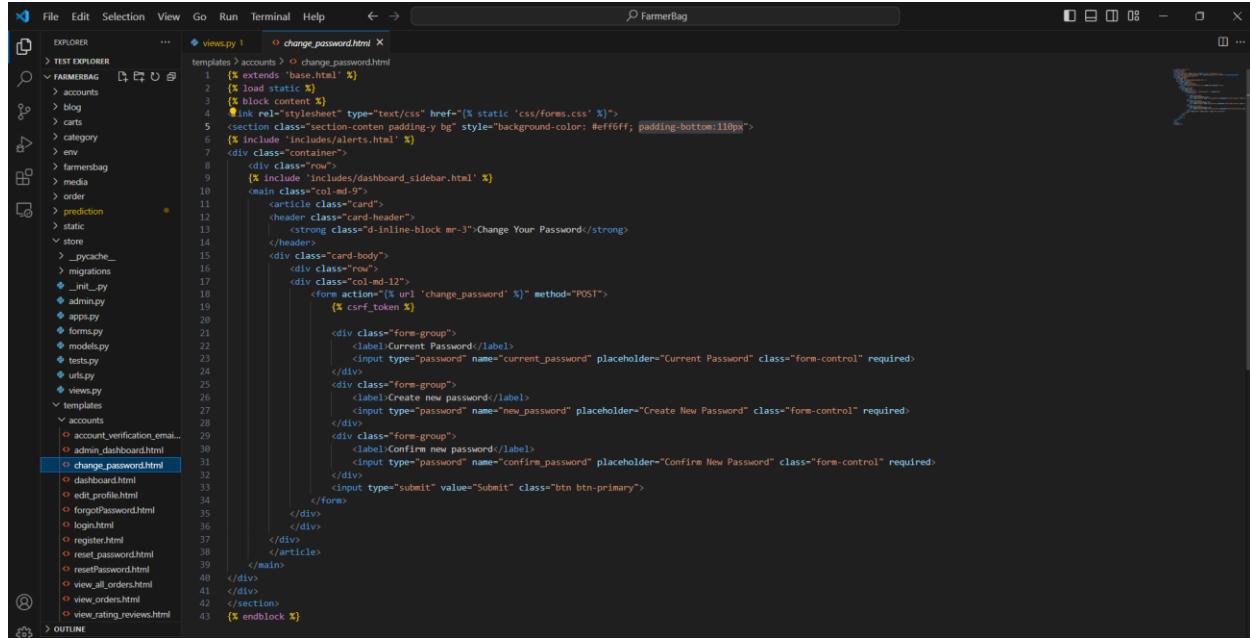
```

1  {% extends "base.html" %}
2  {% load static %}

3  {% block content %}
4      <section class="section-content padding-y bg">
5          {% include 'includes/alerts.html' %}

6          <div class="container">
7              <div class="row">
8                  {% include 'includes/dashboard_sidebar.html' %}
9                  <main class="col-md-9">
10                     <div class="container">
11                         <div class="row">
12                             <div class="col-md-6">
13                                 <div class="card">
14                                     <div class="card-body">
15                                         <h5 class="card-title">Total Active Users</h5>
16                                         <p class="card-text">{{ active_users_count }}</p>
17                                     </div>
18                                 </div>
19                             </div>
20                             <div class="col-md-6">
21                                 <div class="card">
22                                     <div class="card-body">
23                                         <h5 class="card-title">Total Rating</h5>
24                                         <p class="card-text">{{ total_rating }}</p>
25                                     </div>
26                                 </div>
27                             </div>
28                         </div>
29                     </div>
30                 </main>
31             </div>
32         </section>
33     <div class="col-md-6">
34         <div class="card">
35             <div class="card-body">
36                 <h5 class="card-title">Total Payments</h5>
37                 <p class="card-text">{{ total_payment_price,total_price }}</p>
38             </div>
39         </div>
40     </div>
41     <div class="row">
42         <div class="col-md-6">
43             <h2>Payment type Distribution</h2>
44             <div id="paymenttypeChart" width="400" height="400"></div>
45         </div>
46     </div>
47 
```
- Editor Tools:** The bottom status bar shows: In 15, Col 47, Spaces: 4, UTF-8, CRLF, Django HTML, Go Live, Prettier.

Figure 351: Sample Code Of Ui: Admin Dashboard



The screenshot shows a code editor interface with the following details:

- File Path:** templates/accounts/change_password.html
- Content:**

```

1  {% extends "base.html" %}
2  {% load static %}

3  {% block content %}
4      <link rel="stylesheet" type="text/css" href="{% static 'css/Forms.css' %}">
5      <section class="section-content padding-y bg" style="background-color: #fff; padding-bottom:110px">
6          {% include 'includes/alerts.html' %}
7          <div class="container">
8              <div class="row">
9                  {% include 'includes/dashboard_sidebar.html' %}
10                 <main class="col-md-9">
11                     <article class="card">
12                         <header class="card-header">
13                             <strong>Change Your Password</strong>
14                         </header>
15                         <div class="card-body">
16                             <div class="col-md-12">
17                                 <form action="{% url 'change_password' %}" method="POST">
18                                     {% csrf_token %}
19                                     <div class="form-group">
20                                         <label>Current Password:</label>
21                                         <input type="password" name="current_password" placeholder="Current Password" class="form-control" required>
22                                     </div>
23                                     <div class="form-group">
24                                         <label>Create new password:</label>
25                                         <input type="password" name="new_password" placeholder="Create New Password" class="form-control" required>
26                                     </div>
27                                     <div class="form-group">
28                                         <label>Confirm new password:</label>
29                                         <input type="password" name="confirm_password" placeholder="Confirm New Password" class="form-control" required>
30                                     </div>
31                                     <div>
32                                         <input type="submit" value="Submit" class="btn btn-primary" />
33                                     </div>
34                                 </form>
35                             </div>
36                         </article>
37                     </div>
38                 </div>
39             </main>
40         </div>
41     </section>
42 
```
- Editor Tools:** The bottom status bar shows: In 43, Col 1, Spaces: 4, UTF-8, CRLF, Django HTML, Go Live, Prettier.

Figure 352: Sample Code Of Ui: Change Password

```
File Edit Selection View Go Run Terminal Help < > FarmerBag

EXPLORER
> TEST EXPLORER
FARMERBAG
  > store
    > models.py
    > tests.py
    > urls.py
  > views.py
  > templates
    > accounts
      > account_verification_email.html
      > admin_dashboard.html
      > change_password.html
      > dashboard.html
      > edit_profile.html
      > forgotPassword.html
      > login.html
      > register.html
      > reset_password.html
      > resetPassword.html
      > view_all_orders.html
      > view_orders.html
      > view_rating_reviews.html
      > view_users.html
    > blog
      > blog_list.html
      > create_blog_post.html
      > update_blog_post.html
    > includes
    > prediction
      > prediction.html
    > store
      > cart.html
      > checkout.html
      > create_product.html
      > product_detail.html
      > store.html
      > update_product.html
      > base.html
> OUTLINE
> TIMELINE

views.py 1 prediction.html x
templates > prediction > prediction.html
1 1 (% extends "base.html" %)
2 2 (% load static %)
3 3
4 4 (% block content %)
5 5 <link rel="stylesheet" type="text/css" href="{% static 'css/prediction.css' %}">
6 6 <script>
7 7     function previewImage(event) {
8 8         var reader = new FileReader();
9 9         reader.onload = function() {
10 10             var output = document.getElementById('image-preview');
11 11             output.src = reader.result;
12 12         }
13 13         reader.readAsDataURL(event.target.files[0]);
14 14     }
15 15 </script>
16 16
17 17 <div class="prediction-page" >
18 18   <h1>Prediction</h1>
19 19   <div class="prediction-page-container-main">
20 20     <h3>Upload an Image</h3>
21 21     <div class="upload-button-flex">
22 22       <div class="file-upload-box-file-div">
23 23         <form class="file-upload-form" method="post" enctype="multipart/form-data" action="imgPrediction">
24 24           {% csrf_token %}
25 25           <label for="file">File</label>
26 26           <div class="file-upload-design">
27 27             <input type="file" id="file" name="filePath" alt="File Input" style="width: 100%; height: 1em;">
28 28             <div style="display: flex; align-items: center; gap: 10px; margin-top: 5px; font-size: small; color: #ccc; position: relative; width: 100%;">
29 29               <span>Drop or Click here to upload your image file</span>
30 30               <span>Or</span>
31 31               <span>Browse...</span>
32 32             </div>
33 33           <div style="margin-top: 10px; border: 1px solid #ccc; padding: 5px; border-radius: 5px; background-color: #f9f9f9; width: fit-content; margin-left: auto; margin-right: 0; position: absolute; bottom: -10px; left: 50%; transform: translateX(-50%);">
34 34             <p>Drag and Drop the image file or Click on this box to upload the image file</p>
35 35           </div>
36 36         </div>
37 37       <input type="button" value="Predict" id="uploadButton" style="background-color: #007bff; color: white; border: none; padding: 5px 10px; border-radius: 5px; font-weight: bold; font-size: 1em; width: fit-content; margin-left: auto; margin-right: 0; position: absolute; bottom: -10px; left: 50%; transform: translateX(-50%);>
38 38     </div>
39 39   </div>
40 40 </div>
41 41 <div class="image-preview-container-div">
42 42   <div class="image-preview-container"></div>
43 43 </div>
44 44 <div class="predicted-values-label">
45 45   {% if error_message %}
```

Figure 353: Sample Code Of Ui: Prediction

```
File Edit Selection View Go Run Terminal Help < > FarmerBag

EXPLORER ... 
> TEST EXPLORER
FARMERBAG ...
  store
    +-- models.py
    +-- tests.py
    +-- urls.py
    +-- views.py
  templates
    +-- accounts
      +-- account_verification_email.html
      +-- admin_dashboard.html
      +-- change_password.html
      +-- dashboard.html
      +-- edit_profile.html
      +-- forgotPassword.html
      +-- login.html
      +-- register.html
      +-- reset_password.html
      +-- resetPassword.html
      +-- view_all_orders.html
      +-- view_orders.html
      +-- view_rating_reviews.html
      +-- view_users.html
    +-- blog
      +-- blog_list.html
      +-- blog_post_list.html
      +-- create_blog_post.html
      +-- update_blog_post.html
    +-- includes
    +-- prediction
      +-- prediction.html
    +-- store
      +-- cart.html
      +-- checkout.html
      +-- create_product.html
      +-- product_detail.html
      +-- store.html
      +-- update_product.html
      +-- base.html
  OUTLINE ...
  TIMELINE ...

views.py 1 store.html x
templates > store > store.html
1 1 % extends "base.html" % 
2 2 % load static % 
3 3 
4 4 <link rel="stylesheet" type="text/css" href="{% static 'css/forms.css' %}">
5 5 
6 6 [% block content %]
7 7   <div class="store-body-div">
8 8     <section class="section-pagetop-bg">
9 9       <div class="container">
10 10         [% if 'search' in request.path %]
11 11           <!-- Print search Page-->
12 12           <div class="search-title-div">
13 13             <h1> Search Results </h1>
14 14           </div>
15 15         [% else %]
16 16           <h1 class="title-page">Our Store</h1>
17 17         [% endif %]
18 18       </div>
19 19     </section>
20 20 
21 21     <section class="section-content padding-y">
22 22       <div class="container">
23 23         <div class="row">
24 24           <aside class="col-md-3">
25 25 
26 26             <div class="card">
27 27               <article class="filter-group">
28 28                 <header class="card-header">
29 29                   <a href="#" data-toggle="collapse" data-target="#collapse_1" aria-expanded="true" class="dropdown-toggle">
30 30                     <i class="icon-control fa fa-chevron-down"></i>
31 31                   <div class="title">Categories</div>
32 32                 </header>
33 33                 <div class="filter-content collapse show" id="collapse_1" >
34 34                   <div class="card-body">
35 35 
36 36                     <ul class="list-menu">
37 37                       <li><a href="{% url 'store' %}">All Products </a></li>
38 38 
39 39                     <% for category in links %>
40 40                       <li><a href="{{ category.get_url }}"/>{{category.category_name}}</a></li>
41 41                     <% endfor %>
42 42                   </ul>
43 43                 </div> <!-- card-body-->
44 44             </div>
45 45         </div>
46 46     </div>
47 47   </div>
48 48 </div>
```

Figure 354: Sample Code Of Ui: Store

The screenshot shows the VS Code interface with the following details:

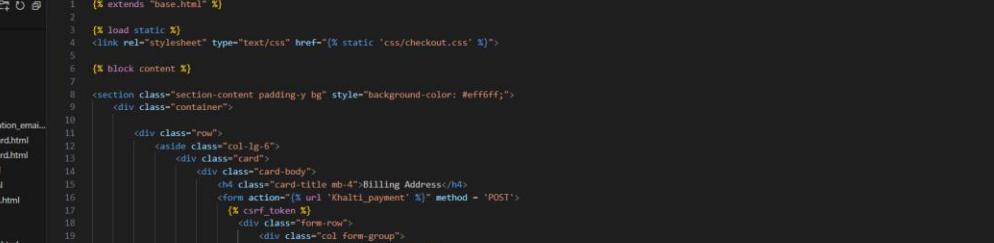
- File Explorer:** Shows the project structure under "FARMERBAG".
- Search Bar:** Displays "FarmerBag".
- Code Editor:** Displays the content of `product_detail.html`. The code is a Jinja template for a product detail page, featuring a main content area with a card, a sidebar with a gallery, and a sidebar for related products.

```
File Edit Selection View Go Run Terminal Help < > FarmerBag

EXPLORER
> TEST EXPLORER
FARMERBAG
templates
  > accounts
    account_verification_email.html
    admin_dashboard.html
    change_password.html
    dashboard.html
    edit_profile.html
    forgotpassword.html
    login.html
    register.html
    reset_password.html
    resetPassword.html
    view_all_orders.html
    view_orders.html
    view_rating_reviews.html
    view_users.html
blog
  blog_post_list.html
  create_blog_post.html
  update_blog_post.html
includes
prediction
  prediction.html
store
  cart.html
  checkout.html
  create_product.html
  product_detail.html
base.html
home.html
dbsqlite3
manage.py
Plant Disease Detection Tran...
README.md

> OUTLINE
> TIMELINE
views.py | product_detail.html
templates > store > product_detail.html
1  {% extends "base.html" %}
2  {% load static %}
3  {% block content %}
4      <section class="section-content padding-y bg">
5          <div class="container">
6              <div class="card">
7                  <div class="row no-gutters">
8                      <aside class="col-md-6">
9                          <article class="gallery-wrap">
10                             <div class="img-big-wrap">
11                                 | <a href="#"></a>
12                             </div>
13                         </article>
14                     </aside>
15                     <main class="col-md-6 border-left">
16                         <article class="content-body">
17                             <h2 class="title">{{ single_product.product_name }}</h2>
18                             <div class="mb-3">
19                                 | <var class="price h4">Rs.{{ single_product.price }}</var>
20                             </div>
21                             <p>{{ single_product.description }}</p>
22                             <!-- Stock -->
23                             {% if single_product.stock <= 0 %}
24                                 <div style="color: red;">Out of Stock</div>
25                             {% else %}
26                                 <% if in_cart %>
27                                     <!-- Added to cart -->
28                                     <a href="#" class="btn btn-success">
29                                         <span class="text">Added to cart</span>
30                                         <i class="fas fa-check"></i>
31                                     </a>
32                                     <a href="{{ url 'cart' }}><span class="text">View Cart</span></a>
33                                     <i class="fas fa-eye"></i>
34                                 </div>
35                                 <% else %>
36                                     <% if not request.user.is_admin %>
```

Figure 355: Sample Code Of Ui: Product Description



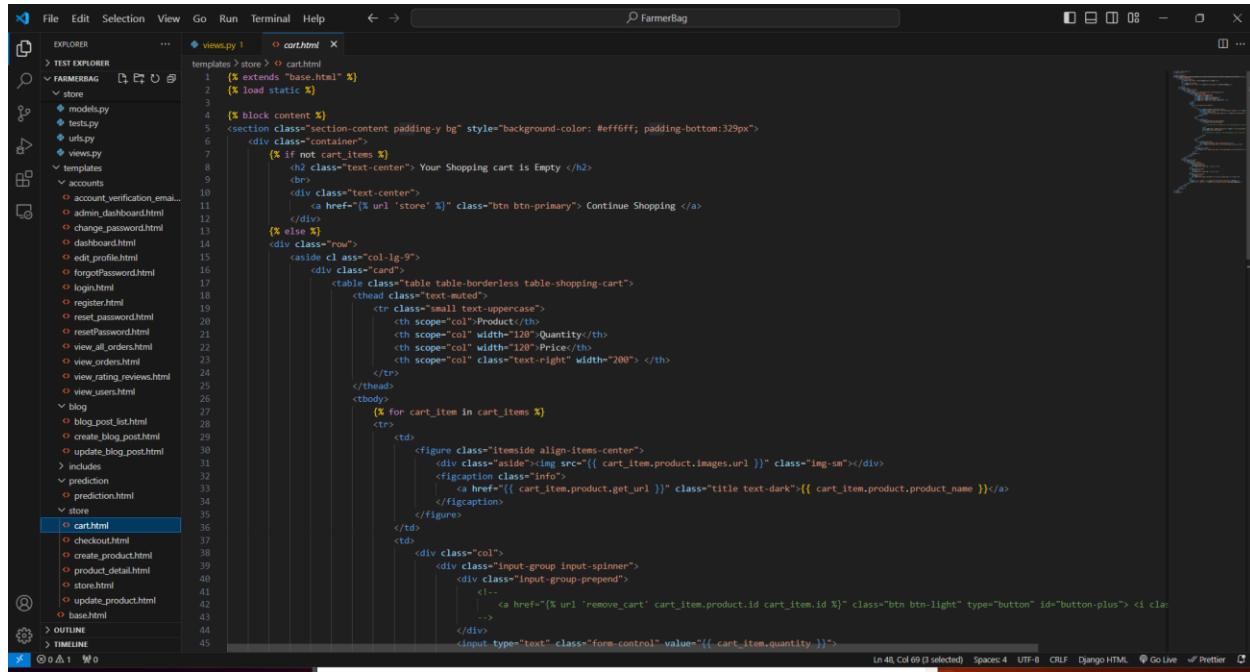
The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "FARMERBAG".
- Search Bar:** Displays "FarmerBag".
- Code Editor:** The current file is "checkout.html".
- Code Content:** The template code uses Bootstrap classes like "row", "col-lg-6", and "card-body". It includes form fields for "Billing Address" with labels and input types "text" and "email". It also includes fields for "Phone Number" and "Primary Address" with "address_line_1" and "address_line_2".

```
<section class="section-content padding-y bg" style="background-color: #eefeff;">
  <div class="container">
    <div class="row">
      <aside class="col-lg-6">
        <div class="card">
          <div class="card-body">
            <h4 class="card-title mb-4">Billing Address</h4>
            <form action="{% url 'Khalti_payment' %}" method="POST">
              {% csrf_token %}
              <div class="form-row">
                <div class="col form-group">
                  <label for="First Name"></label>
                  <input type="text" name="first_name" class="form-control" required>
                </div>
                <div class="col form-group">
                  <label for="Last Name"></label>
                  <input type="text" name="last_name" class="form-control" required>
                </div>
              </div>
              <div class="form-row">
                <div class="col form-group">
                  <label for="Email"></label>
                  <input type="email" name="email" class="form-control" required>
                </div>
                <div class="col form-group">
                  <label for="Phone Number"></label>
                  <input type="text" name="phone" class="form-control" required>
                </div>
              </div>
              <div class="form-row">
                <div class="col form-group">
                  <label for="Primary Address"></label>
                  <input type="text" name="address_line_1" class="form-control" required>
                </div>
                <div class="col form-group">
                  <label for="Temporary Address"></label>
                  <input type="text" name="address_line_2" class="form-control">
                </div>
              </div>
            </form>
          </div>
        </div>
      </aside>
    </div>
  </div>

```

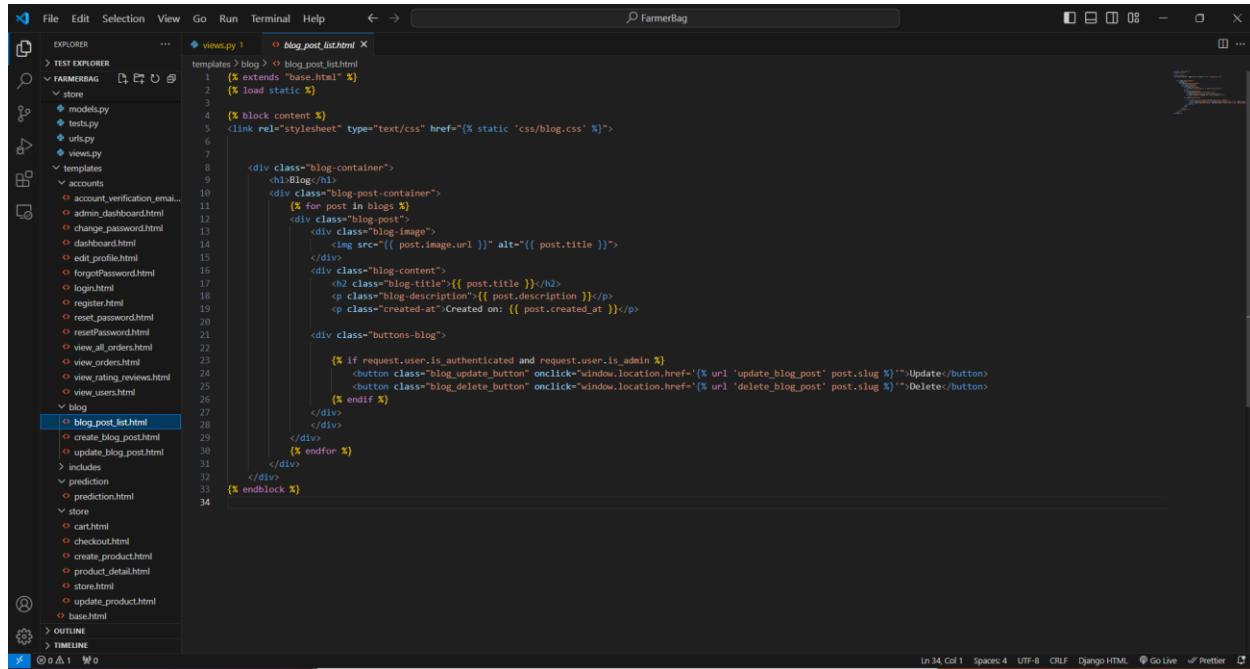
Figure 356: Sample Code Of Ui: Checkout



This screenshot shows a code editor interface with the title bar "FarmerBag". The left sidebar (Explorer) lists project files under "FARMERBAG", including "models.py", "tests.py", "urls.py", and "views.py". The "templates" folder contains "store" and "cart.html" files. The main editor area displays the "cart.html" template code:

```

1  {% extends "base.html" %}
2  {% load static %}
3
4  {% block content %}
5      <section class="section-content padding-y bg" style="background-color: #efffff; padding-bottom:329px">
6          <div class="container">
7              {% if not cart_items %}
8                  <div class="text-center"> Your Shopping cart is Empty </h2>
9                  <br>
10                 <div class="text-center">
11                     <a href="{% url 'store' %}" class="btn btn-primary" Continue Shopping </a>
12                 </div>
13             {% else %}
14                 <div class="row">
15                     <aside class="col-lg-9">
16                         <div class="card">
17                             <table class="table table-bordered borderless table-shopping-cart">
18                                 <thead class="small text-muted">
19                                     <tr>
20                                         <th scope="col" width="120">Product</th>
21                                         <th scope="col" width="120">Quantity</th>
22                                         <th scope="col" class="text-right" width="200">> Price</th>
23                                     </tr>
24                                 </thead>
25                                 <tbody>
26                                     {% for cart_item in cart_items %}
27                                         <tr>
28                                             <td>
29                                                 <figure class="item-side align-items-center">
30                                                     <div class="aside"></div>
31                                                     <figcaption class="info">
32                                                         <a href="{{ cart_item.product.get_url }}>{{ cart_item.product.product_name }}</a>
33                                                     </figcaption>
34                                                 </figure>
35                                             </td>
36                                             <td>
37                                                 <div class="col">
38                                                     <div class="input-group input-spinner">
39                                                         <div class="input-group-prepend">
40                                                             <i>--</i>
41                                                             <a href="{% url 'remove_cart' cart_item.product.id cart_item.id %}" class="btn btn-light" type="button" id="button-plus"> <i class="fa fa-plus" /> </a>
42                                                         </div>
43                                                     <input type="text" class="form-control" value="{{ cart_item.quantity }}"/>
44                                                 </div>
45                                             </td>
46                                         </tr>
47                                     {% endfor %}
48                                 </tbody>
49                             </table>
50                         </div>
51                     </aside>
52                 </div>
53             </div>
54         </section>
55     
```

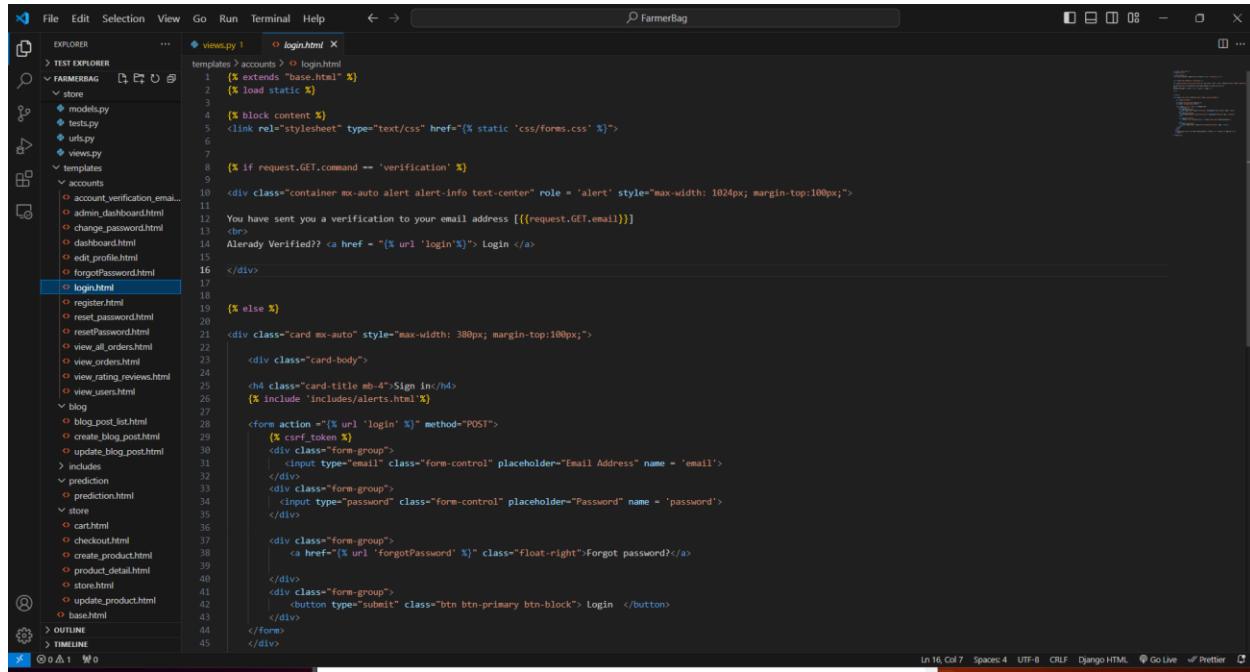
Figure 357: Sample Code Of Ui: Cart


This screenshot shows a code editor interface with the title bar "FarmerBag". The left sidebar (Explorer) lists project files under "FARMERBAG", including "models.py", "tests.py", "urls.py", and "views.py". The "templates" folder contains "blog" and "blog_post_list.html" files. The main editor area displays the "blog_post_list.html" template code:

```

1  {% extends "base.html" %}
2  {% load static %}
3
4  {% block content %}
5      <link rel="stylesheet" type="text/css" href="{% static 'css/blog.css' %}">
6
7      <div class="blog-container">
8          <h1>Blog</h1>
9          <div class="blog-post-container">
10             {% for post in blogs %}
11                 <div class="blog-item">
12                     <div class="blog-image">
13                         
14                     </div>
15                     <div class="blog-content">
16                         <h2>{{ post.title }}</h2>
17                         <p>{{ post.description }}</p>
18                         <p>{{ post.created_at }}</p>
19
20                         <div class="buttons-blog">
21                             {% if request.user.is_authenticated and request.user.is_admin %}
22                                 <button class="blog_update_button" onclick="window.location.href='{% url 'update_blog_post' post.slug %}'>Update</button>
23                                 <button class="blog_delete_button" onclick="window.location.href='{% url 'delete_blog_post' post.slug %}'>Delete</button>
24                             {% endif %}
25                         </div>
26                     </div>
27                 </div>
28             {% endfor %}
29         </div>
30     </div>
31
32     {% endblock %}
33 
```

Figure 358: Sample Code Of Ui: Blog Post

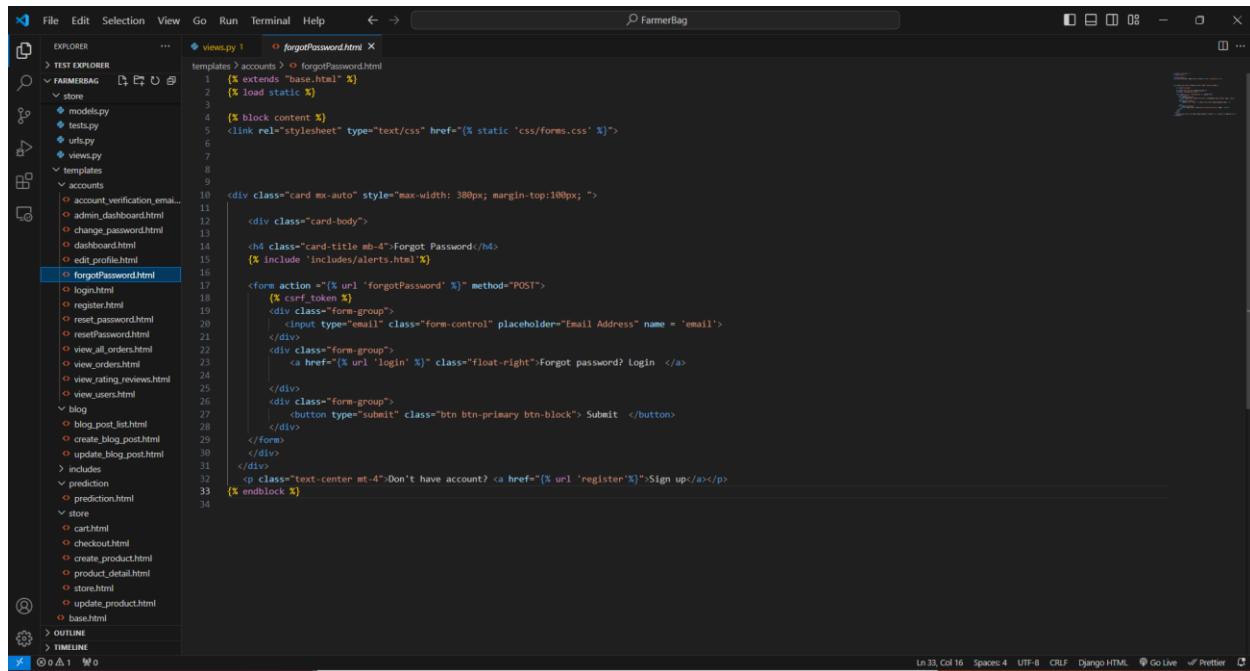


```

File Edit Selection View Go Run Terminal Help < - > FarmerBag
EXPLORER TEST EXPLORER FARMERBAG
store models.py tests.py urls.py views.py
accounts account_verification_email... admin_dashboard.htmlm change_password.html dashboard.html edit_profile.html forgotPassword.html
login.html register.html reset_password.html resetPassword.html view_all_orders.html view_orders.html view_rating_reviews.html view_users.html
blog blog_post_list.html create_blog_post.html update_blog_post.html
includes prediction.html
store cart.html checkout.html create_product.html product_detail.html store.html update_product.html
base.html
OUTLINE TIMELINE
views.py 1 login.html
templates > accounts > login.html
1 [% extends "base.html" %]
2 [% load static %]
3
4 [% block content %]
5 <link rel="stylesheet" type="text/css" href="{% static 'css/forms.css' %}">
6
7
8 [% if request.GET.command == 'verification' %]
9
10 <div class="container mx-auto alert alert-info text-center" role="alert" style="max-width: 1024px; margin-top:100px;">
11 You have sent you a verification to your email address [{% request.GET.email %}]
12 <br>
13 Already Verified?? <a href = "{% url 'login' %}"> Login </a>
14
15 </div>
16
17 [% else %]
18
19 <div class="card mx-auto" style="max-width: 380px; margin-top:100px;">
20
21 <div class="card-body">
22
23 <h4 class="card-title mb-4">Sign in</h4>
24 <% include 'includes/alerts.html' %>
25
26 <form action ="{% url 'login' %}" method="POST">
27   [<% csrf_token %>]
28   <div class="form-group">
29     <input type="email" class="form-control" placeholder="Email Address" name = 'email'>
30   </div>
31   <div class="form-group">
32     <input type="password" class="form-control" placeholder="Password" name = 'password'>
33   </div>
34
35   <div class="form-group">
36     <a href ='{% url 'forgotPassword' %}' class="float-right">Forgot password?</a>
37   </div>
38   <div class="form-group">
39     <button type="submit" class="btn btn-primary btn-block"> Login </button>
40   </div>
41
42 </form>
43
44 </div>
45

```

Ln 16, Col 7 Spaces: 4 UTF-8 CRLF Django HTML Go Live ⚡ Prettier

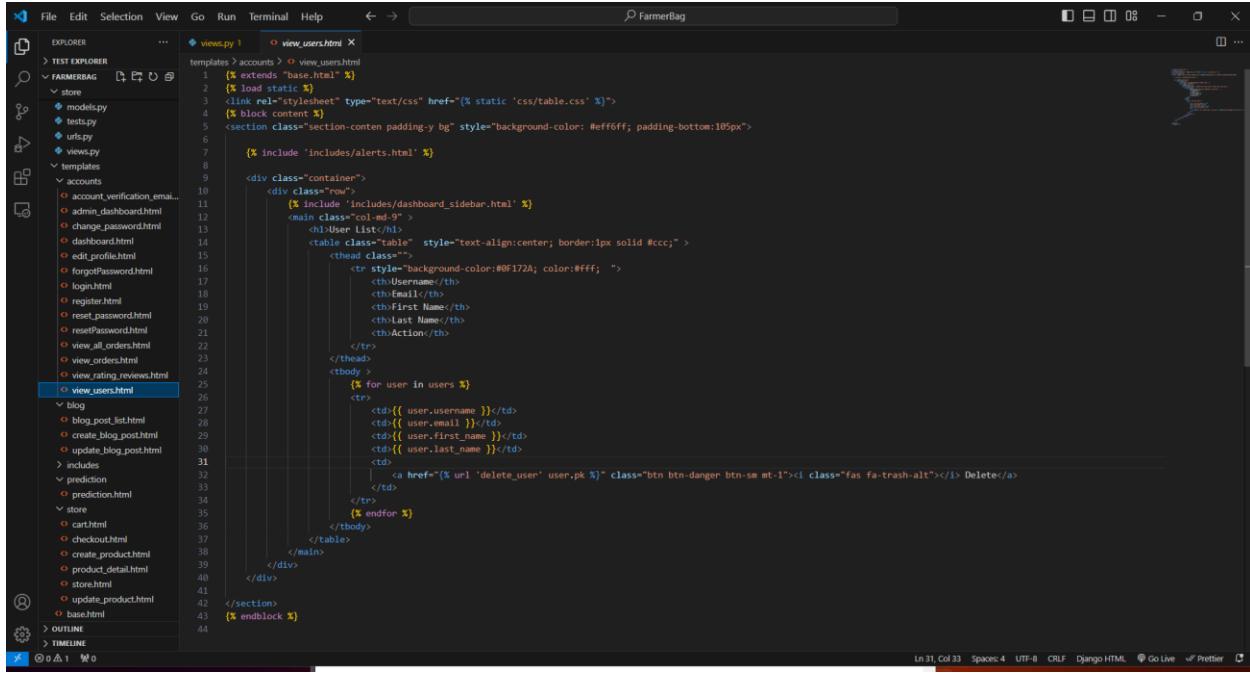
Figure 359: Sample Code Of Ui: Login


```

File Edit Selection View Go Run Terminal Help < - > FarmerBag
EXPLORER TEST EXPLORER FARMERBAG
store models.py tests.py urls.py views.py
accounts account_verification_email... admin_dashboard.htmlm change_password.html dashboard.html edit_profile.html
forgotPassword.html
login.html register.html reset_password.html resetPassword.html view_all_orders.html view_orders.html view_rating_reviews.html view_users.html
blog blog_post_list.html create_blog_post.html update_blog_post.html
includes prediction.html
store cart.html checkout.html create_product.html product_detail.html store.html update_product.html
base.html
OUTLINE TIMELINE
views.py 1 forgotPassword.html
templates > accounts > forgotPassword.html
1 [% extends "base.html" %]
2 [% load static %]
3
4 [% block content %]
5 <link rel="stylesheet" type="text/css" href="{% static 'css/forms.css' %}">
6
7
8 <div class="card mx-auto" style="max-width: 380px; margin-top:100px;">
9
10 <div class="card-body">
11
12 <h4 class="card-title mb-4">Forgot Password:</h4>
13 <% include 'includes/alerts.html' %>
14
15 <form action ="{% url 'forgotPassword' %}" method="POST">
16   [<% csrf_token %>]
17   <div class="form-group">
18     <input type="email" class="form-control" placeholder="Email Address" name = 'email'>
19   </div>
20   <div class="form-group">
21     <a href ='{% url 'login' %}' class="float-right">Forgot password? Login </a>
22   </div>
23
24   <div class="form-group">
25     <button type="submit" class="btn btn-primary btn-block"> Submit </button>
26   </div>
27
28 </form>
29
30 </div>
31
32 <p class="text-center mt-4">Don't have account? <a href ='{% url 'register' %}'>Sign up!</a></p>
33
34 [% endblock %]

```

Ln 33, Col 16 Spaces: 4 UTF-8 CRLF Django HTML Go Live ⚡ Prettier

Figure 360: Sample Code Of Ui: Forgot Password


The screenshot shows a code editor interface with a Python file named `views.py` open. The code is a Django template for a 'Forgot Password' page. It includes imports for `base.html`, `css/table.css`, and `includes/alerts.html`. The template uses a Bootstrap grid system with a main column of width 9 and a sidebar column of width 3. It displays a user list table with columns for Username, Email, First Name, Last Name, and Action (with a delete link). The code is annotated with line numbers from 1 to 44.

```

1  {% extends "base.html" %} 
2  {% load static %} 
3  <link rel="stylesheet" type="text/css" href="{% static 'css/table.css' %}">
4  {% block content %} 
5  <section class="section-conten padding-y bg" style="background-color: #eff6ff; padding-bottom:10px">
6  {% include 'includes/alerts.html' %} 
7  <div class="container">
8  <div class="row">
9  	{% include 'includes/dashboard_sidebar.html' %} 
10 <main class="col-md-9">
11 	<h1>User List</h1>
12 	<table class="table" style="text-align:center; border:1px solid #ccc;">
13  	<thead class="thead">
14     <tr style="background-color:#00f172a; color:#fff; ">
15       <th>Username</th>
16       <th>Email</th>
17       <th>First Name</th>
18       <th>Last Name</th>
19       <th>Action</th>
20     </tr>
21   </thead>
22   <tbody>
23   	{% for user in users %}
24      <tr>
25        <td>{{ user.username }}</td>
26        <td>{{ user.email }}</td>
27        <td>{{ user.first_name }}</td>
28        <td>{{ user.last_name }}</td>
29        <td>
30          | <a href="{% url 'delete_user' user.pk %}" class="btn btn-danger btn-sm mt-1"><i class="fas fa-trash-alt"></i> Delete</a>
31        </td>
32      </tr>
33    {% endfor %}
34  </tbody>
35  </table>
36  </main>
37  </div>
38  </div>
39  </div>
40  </div>
41  </div>
42  </section>
43  {% endblock %} 
44

```

Figure 361: Sample Code Of Ui: View Users

```

<!-- reviews_table.html -->
{% extends "base.html" %}

{% block content %}


| User                       | Rating              | Comment              | Action                                                                          |
|----------------------------|---------------------|----------------------|---------------------------------------------------------------------------------|
| {{ review.user.username }} | {{ review.rating }} | {{ review.comment }} | <a class="btn btn-danger" href="{% url 'delete_review' review.id %}">Delete</a> |


{% endblock %}

```

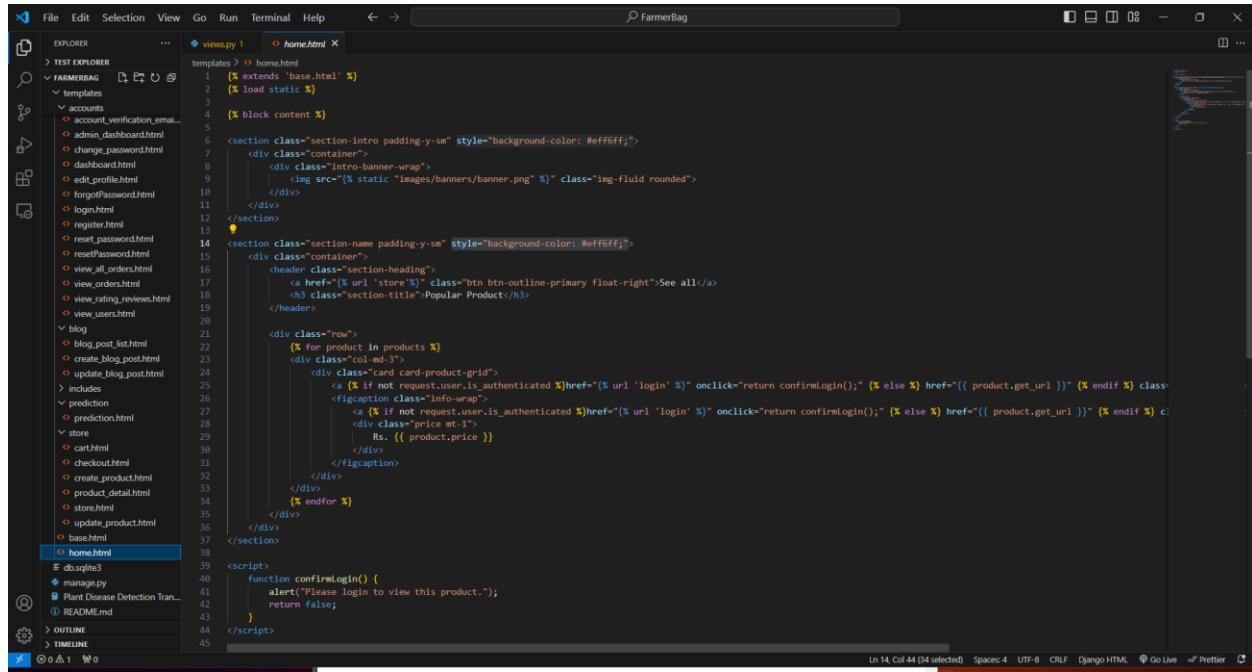
Figure 362: Sample Code Of Ui: View Ratings

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="pragma" content="no-cache" />
    <meta http-equiv="cache-control" content="max-age=604800" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <title>FarmerBag | Best Solution for Farmers</title>
    <!-- Bootstrap files -->
    <link href="{% static 'css/bootstrap.css' %}" rel="stylesheet" type="text/css"/>
    <!-- custom style -->
    <link href="{% static 'css/ui.css' %}" rel="stylesheet" type="text/css"/>
    <link href="{% static 'css/responsive.css' %}" rel="stylesheet" media="only screen and (max-width: 1200px)"/>
    <script type="text/javascript">
        $(document).ready(function() {
            });
    </script>
</head>
<body>
    {% include "includes/navbar.html" %}
    {% block content%}
    {% endblock %}
    <!-- Footer-->
    {% include "includes/footer.html" %}

```

Figure 363: Sample Code Of Ui: Base



The screenshot shows a code editor interface with a dark theme. On the left is a file explorer sidebar containing project files like 'TEST EXPLORER', 'FARMERBAG', 'templates', 'includes', 'prediction', 'store', and various HTML and Python files. The main area displays a Python template file named 'home.html' with line numbers. The code uses Django's templating syntax, including blocks, sections, and conditionals. It includes a header with a banner image and a section for displaying products. A script block at the bottom handles a login confirmation function.

```

1  {% extends 'base.html' %}
2  {% load static %}
3
4  {% block content %}
5
6  <section class="section-intro padding-y-sm" style="background-color: #efffff">
7      <div class="container">
8          <div class="intro-banner-wrap">
9              |     
10             |</div>
11         </div>
12     </section>
13
14     <section class="section-name padding-y-sm" style="background-color: #efffff">
15         <div class="container">
16             <header>
17                 <a href="{% url 'store' %}" class="btn btn-outline-primary float-right">See all</a>
18                 <h3 class="section-title">Popular Product</h3>
19             </header>
20
21             <div class="row">
22                 {% for product in products %}
23                     <div class="col-md-3">
24                         <div class="card card-product-grid">
25                             {% if not request.user.is_authenticated %}
26                                 <a href="{% url 'login' %}" onclick="return confirmLogin();"
27                                     class="info-wrap">
28                                     <a href="{% url 'login' %}" onclick="return confirmLogin();"
29                                         class="price mt-1">
30                                         {{ product.price }}</a>
31                                 </a>
32                             </div>
33                         </div>
34                         {% endif %}
35                     </div>
36                 </div>
37             </section>
38
39             <script>
40                 function confirmLogin() {
41                     alert("Please login to view this product.");
42                     return false;
43                 }
44             </script>

```

Figure 364: Sample Code Of Ui: Home

7.7. APPENDIX G: CONCLUSION

7.7.1. FUTURE WORK

To address the current limitations and enhance the Farmer Bag Application Project, several key improvements are necessary.

Firstly, the backend code should be migrated from Django to the Django REST Framework, which will provide a more robust and scalable architecture for building web APIs. This change will also facilitate better integration with modern frontend frameworks and enable the development of mobile applications in the future.

Secondly, the database should be migrated from SQLite to PostgreSQL, a more powerful and feature-rich relational database management system. PostgreSQL offers improved performance, scalability, and support for advanced features such as data integrity, concurrency control, and replication, which will be crucial as the application grows and handles larger datasets.

To improve the user experience, the app's user interface should be enhanced, incorporating modern design principles and intuitive navigation. Additionally, the user and admin portals should be redesigned using React.js, a popular JavaScript library for building responsive and high-performance user interfaces. React.js will enable the development of dynamic and interactive components, improving the overall usability of the application.

Furthermore, the functionality for incrementing and decrementing quantity items in the shopping cart should be improved to provide a more seamless and user-friendly experience. A chat feature should also be introduced, allowing users to communicate with each other or with customer support representatives in real-time, enhancing the overall customer experience.

To cater to the growing demand for telehealth services, a feature for scheduling appointments with doctors should be incorporated into the application. This will enable users to conveniently book consultations and receive remote medical advice, expanding the app's capabilities beyond its current e-commerce focus.

Moreover, the web application should be converted into a mobile app, ensuring accessibility and convenience for users on-the-go. Additionally, a chatbot feature should be implemented to assist

users with frequently asked questions, product recommendations, and navigation through the app's features, providing a more personalized and efficient user experience.

In the realm of machine learning, improvements should be made to the existing recommendation system by incorporating advanced AI techniques. This will allow for more accurate and personalized product recommendations based on user preferences, purchase history, and other relevant data.

By implementing these comprehensive changes, the Farmer Bag Application Project will be transformed into a more robust, scalable, and user-friendly platform, better equipped to serve its users effectively and meet the evolving demands of the digital age.

[GO TO TOP](#)