



Module Code & Title

CS6P05 Final Year Project MAD
Food Share – Android App

Assessment Weightage & Type

Final Report

Supervisor By

Mr. Shekhar Timsina
Mr. Bishal Gharti Chhetri (GC)

Student Details

Name: Sita Ram Thing
London Met Id: 22015892
College Id: NP01MA4S220003
Islington College, Kathmandu
24 April 2024

Git Hub: [FYP-23-24/22015892-SitaRamThing \(github.com\)](https://github.com/FYP-23-24/22015892-SitaRamThing)

Contents

CHAPTER 1: INTRODUCTION.....	1
1.1 Project Introduction	1
1.2. Current Scenario	2
1.3. Problem Domain and Project as a Solution.....	4
1.4. Aim and objectives.....	5
1.4.1. Aim	5
1.4.2. Objectives.....	5
1.5. Structure of the Report.....	6
1.5.1. Background	6
1.5.2. Development.....	6
1.5.3. Testing and analysis.....	7
1.5.4. Conclusion	7
CHAPTER 2: BACKGROUND	8
2.1. About the End Users	8
2.2. Understanding the solution	10
2.2.1. Overview of the system.....	11
2.2.2. Technical Terms and Definition (Technical Terms related to the project)	12
2.2.3. Function and Features	16
2.3. Comparison.....	18
2.3.1. Similar Projects	18
2.3.2. Comparisons (Comparing the features and critical evaluation of the solution)	
.....	21
CHAPTER 3: DEVELOPMENT	22
3.1. Methodology	22

3.1.1. Considered Methodologies.....	22
3.1.2. Justification for Not Selected Methodologies.....	29
3.1.3. Justification for Selected Methodology	33
3.2 Survey Result.....	36
3.2.1. Pre-Survey Results	36
3.2.2. Post-Survey Question	43
3.2.3 Post survey result.....	44
3.3. Requirement Analysis	48
3.3.1 Introduction	48
3.3.2 Purpose.....	48
3.3.3 Scope.....	48
3.3.4 Reference.....	49
3.3.5 Overall Descriptions	52
3.3.6 Functional Requirement	53
3.3.7 External Interface Requirements.....	54
3.3.8 External Requirement	57
3.3.9 Non-Functional Requirement	59
3.9.1 Usability.....	59
3.9.2 Security	59
3.9.3 Performance.....	59
3.9.4 Error Handling	59
3.9.5 Ease of Use.....	60
3.9.6 Maintainability	60
3.9.7 Portability	60

3.4. Design (Concerning methodology what are the core design techniques. For eg: If the methodology is USDP, UML diagrams are a must)	61
3.4.2. Use Case	61
3.2.1. High-Level Use Case	63
3.3. Sequence Diagram	68
3.3.1. View profile sequence diagram	68
3.3.4. Donation rating sequence diagram	71
3.3.5. Complaint with admin.....	72
3.4. Context Diagram	73
3.5. Data Flow Diagram Level -1.....	74
3.6. Activity Diagram	75
3.6.1. Take Membership Activity.....	75
3.6.2. Donation Activity.....	76
3.6.3. View History Activity	77
3.6.4. Donation Rating Activity	77
3.6.5. Complaint with Admin Activity.....	78
3.7. Wireframes.....	79
3.7.1 Mobile UI	79
3.7.2. Web UI for Admin	93
3.8. ER-Diagram	95
3.9. System Architecture Diagram.....	97
3.10. Class Diagram	98
3.11. DFD's level 2	99
3.11.1. Take Membership	99
3.5. Implementation	105

3.5.1. System Development (Important screenshots of development core features and architecture)	105
3.5.2. System Architecture	123
CHAPTER 4: TESTING AND ANALYSIS	126
4.1. Test Plan	126
4.1.1. Unit Testing	126
4.1.2. System Testing.....	127
4.2. Test Execution.....	128
4.2.1. Unit Testing	128
4.2.2. System Testing.....	152
4.3. Critical Analysis.....	231
CHAPTER 5: CONCLUSION	233
5.1. LEGAL, SOCIAL AND ETHICAL ISSUES	234
5.1.1. LEGAL ISSUES.....	234
5.1.2. SOCIAL ISSUES	234
5.1.3. ETHICAL ISSUES	234
5.2. Advantages of waste food or surplus food donation system	235
5.4. Limitations.....	236
5.1.1 Limited User Control:	236
5.1.2 Risk of Misuse:.....	236
5.1.3 Potential for Inefficiency:	236
5.1.4 Limited Transparency:.....	236
5.1.5 Dependency on Volunteer Availability:	237
5.1.6 Geographical Constraints:.....	237
5.1.7 Communication Challenges:	237

5.1.8 Limited Platform Availability:.....	237
5.1.9 Unequal Access:	237
5.3. FUTURE WORK	238
5.3.1 Dark Mode.....	238
5.3.2 Chat (Messaging)s	239
5.3.3. Kotlin multi-platform	241
CHAPTER 6: References.....	243
References.....	243
CHAPTER 7: APPENDIX	246
7.1 Appendix A: Introduction	246
7.1.1 Current Scenario Additional information.....	246
7.2 Appendix B: Background.....	247
7.2.1 About the End User	247
2.2.1.1 Web (Admin)	247
2.2.1.2 Mobile.....	247
7.3 Appendix C: Methodology	249
7.3.1 Survey.....	249
7.3.2 UML Diagram	267
7.4 Appendix D: Implementation	318
7.4.1 Unit Test Plans	318
7.5 Appendix E: Testing and Analysis	322
7.5.1 System Testing.....	322
7.5 Appendix F: Deployment (Not real deployment).....	345

List of table

Table 1: Comparison to a similar project	21
Table 2: Justification scenario 1 for waterfall methodology.....	29
Table 3: Justification scenario 2 for waterfall methodology.....	30
Table 4: Justification scenario 3 for waterfall methodology.....	30
Table 5: Justification scenario 1 for prototype methodology	31
Table 6: Justification scenario 2 for Prototype methodology.....	32
Table 7: Justification scenario 3 for Prototype methodology.....	32
Table 8: Hardware interface Requirement.....	54
Table 9: Register user high-level use case.....	63
Table 10: Take membership user high-level use case	64
Table 11: Login user high-level use case.....	64
Table 12: Food donation high-level use case	65
Table 13: View donation info high-level use case.....	65
Table 14: Donation rating high-level use case.....	66
Table 15: View the history of high-level use case.....	66
Table 16: View profile high-level use case.....	66
Table 17: Complaint with high-level use case.....	67
Table 18: Register Unit Test	128
Table 19: Login Authentication Unit test	131
Table 20: Device token save unit.....	134
Table 21: View history unit test.....	137
Table 22: Report to admin history unit test	139
Table 23: Donation Rating	142
Table 24: The number of system ration	145
Table 25: Email verify unit test.....	147
Table 26: NOG register unit test	150
Table 27: Web testing for login table	153
Table 28: Empty web login screen.....	154
Table 29: Web testing to login screen in unauthenticated user	156
Table 30: Web testing register user table	158

Table 31: Web test in enter the email, password and confirm password	159
Table 32: Web test update password success to navigate to the login page.....	159
Table 33: Test for the home page has a history in pie-chart	160
Table 34: Web testing for View history tables	162
Table 35: Web testing for deleting table details	164
Table 36: Web test for adding new user table	166
Table 37: Web testing for user account activate table	169
Table 38: Web Logout Testing Table.....	171
Table 39: Registered user test in mobile app	174
Table 40: Login credential user account test in the mobile app	177
Table 41: Login credential user account test in the mobile app	180
Table 42: Forgot Password test in mobile app.....	183
Table 43: View the History test in the mobile app	187
Table 44: View the Donation location test in the mobile app	190
Table 45: View user profile test in the mobile app	194
Table 46: Donate food test in mobile app	197
Table 47: View notification history	202
Table 48: Report to admin test in mobile app	223
Table 49: Update profile details test in the mobile app	227
Table 56: Register user expanded use case table.....	270
Table 57: Table membership Expanded use case	271
Table 58: Take membership expanded use case table.....	272
Table 59: Login expended use case diagram	273
Table 60: Login user expanded use case table	274
Table 61: Food donation expended use case diagram	275
Table 62: Food donation user expanded use case table	276
Table 63: View Donation history expended use case diagram	277
Table 64: View history user expanded use case table.....	278
Table 65: Donation rating expended use case diagram.....	279
Table 66: Donating rating expanded use case table.....	280
Table 67: Complaint with admin expanded use case diagram.....	281

Table 68: Complaint with admin expanded use case table.....	282
Table 69: Unit testing planning tables.....	321
Table 50: Update profile image test in the mobile app.....	322
Table 51: Deactivate user account test in mobile app	326
Table 52: Change password test mobile app.....	330
Table 53: Logout test mobile app	334
Table 54: Logout test mobile app	337
Table 55: View the History test in the mobile app	340

List of Figures

Figure 1: Current Scenario of Nepal context food problems.....	3
Figure 2: Client Approvable latter.....	9
Figure 3: Flash floods similar app image.....	18
Figure 4. Food Rescue similar project image	19
Figure 5: No Food Waste similar projects image.....	20
Figure 6: Waterfall methodology.....	24
Figure 7: Scrum Methodology	26
Figure 8: Prototype methodology.....	28
Figure 9: Justification scenario 1 for Scrum methodology.	33
Figure 10: Justification scenario 2 for Scrum methodology.	33
Figure 11: Justification scenario 3 for Scrum methodology.	34
Figure 12: Justification scenario 4 for Scrum methodology.	34
Figure 13: Justification scenario 5 for Scrum methodology.	35
Figure 14: Justification scenario 6 for Scrum methodology.	35
Figure 15: Pre-survey form introduction with username	36
Figure 16: Pre-servery percentage of user age.....	37
Figure 17: Pre-servery user occupations.....	38
Figure 18: Preservay result of excess food for donated	38
Figure 19: Pre-survey result per cent of users participated in the donation program ...	39
Figure 20: Pre-survey result user can food donates or dump.....	39
Figure 21: Pre-survey result of user interest directly or through donations organisations	40
Figure 22: Pre-survey result of user interest in food donation	40
Figure 23: Pre-survey rating point of food donation idea.....	41
Figure 24: Pre-survey results for use need donation tracking	41
Figure 25: Pre-survey result of the user is interested in receiving the update	42
Figure 26: User feeb back for system requirement	42
Figure 27: Post-survey introductions with questions of email and username	43
Figure 28: Post-survey of User Occupations.....	44
Figure 29: Post-survey results of user experience	44

Figure 30: Post-survey results used to recommend the system.....	45
Figure 31: Post-survey of system navigation survey response	45
Figure 32: Post-survey result of finding the helpful	46
Figure 33: Post survey results of recommendations for donations.....	46
Figure 34: Post-survey of overall system rating	47
Figure 35: Use Case Diagram.....	62
Figure 36: View profile sequence diagram	68
Figure 37: View the history sequence diagram.....	69
Figure 38: Food Post for donation sequence diagram.....	70
Figure 39: Donation rating sequence diagram	71
Figure 40: Complaint with admin sequence diagram	72
Figure 41: Context diagram.....	73
Figure 42: DFD Level-1	74
Figure 43: Take the membership activity diagram	75
Figure 44: Food donation system activity diagram	76
Figure 45: View the history system activity diagram.....	77
Figure 46: Rating donation system activity diagram	77
Figure 47: Complaint with admin system activity diagram.....	78
Figure 48: Walkthrough screen mobile UI	79
Figure 49: Welcome screen mobile UI	80
Figure 50: Login screen mobile UI	81
Figure 51: Register screen mobile UI	82
Figure 52: Forgot password screen mobile UI.....	83
Figure 53: Donor Home screen mobile UI	84
Figure 54: Donor History screen mobile UI	85
Figure 55: Donor Post/Donation screen mobile UI	86
Figure 56: Volunteer Home screen UI	87
Figure 57: Volunteer Pending screen UI.....	88
Figure 58: Volunteer History Screen.....	89
Figure 59: NGO Home screen UI	90
Figure 60: NGO History screen UI	91

Figure 61: NGO Setting Screen UI	92
Figure 62: Admin login page web UI	93
Figure 63: Admin forgot password UI	93
Figure 64: Admin dashboard UI.....	94
Figure 65: ER-Diagram	96
Figure 66: System Architecture Diagram.....	97
Figure 67: Class diagram of food donation application.....	98
Figure 68: Take membership details DFD Level-2.....	100
Figure 69: Food donation DFD Level-2	101
Figure 70: View Donation History DFD Level-2.....	102
Figure 71: Donation rating DFD Level-2.....	103
Figure 72: Complaint with admin DFD Level-2.....	104
Figure 73: Implementation of the database user model	106
Figure 74: Implementation of database food model.	107
Figure 75: Implementation of database report model.....	108
Figure 76: Implementation of database history model.....	108
Figure 77: Implementation of database notification device token.....	108
Figure 78: Implementation of Firebase service account key- backend.....	109
Figure 79: Implementation of service account key in Firebase admin.....	109
Figure 80: Implementation of notification send function.	110
Figure 81: implementation for receiving the notification message.....	110
Figure 82: Implementation for generating notifications.....	111
Figure 83: Implement Food donations (post) - backend.....	112
Figure 84: Implementations of view donor history	114
Figure 85: Implementation of view volunteer history.....	115
Figure 86: Implementation of Donation rating - backend.....	116
Figure 87: Implementation of donation rating frontend.....	117
Figure 88: Implementation of the report - backend.....	118
Figure 89: Implementation of report-frontend.....	119
Figure 90: Implementation of map key	120
Figure 91: Implementation of map access to the current location late value	121

Figure 92: Implementation of map state man tent.....	122
Figure 93: Implementation of Google Maps.....	122
Figure 94: System Architecture	123
Figure 95: Register User unit test function	129
Figure 96: Register User unit test failed	129
Figure 97: Register success unit test function.....	130
Figure 98: Register success unit test response.....	130
Figure 99: Login User unit test function.....	131
Figure 100: Register User unit test failer response	132
Figure 101: Register success unit test function.....	133
Figure 102: Login success unit test response	133
Figure 103: Device token save unit test function	134
Figure 104: Device token save unit test failer response	135
Figure 105: Save device token success unit test function	135
Figure 106: Save device token success unit test response.....	136
Figure 107: History details get unit test failer response.....	138
Figure 108: Report to admin unit test function.....	139
Figure 109: Report to admin unit test response	140
Figure 110: Report to admin unit test success response.....	140
Figure 111: Report to admin unit test success function	141
Figure 112: Donation Rating unit testing function	143
Figure 113: Donation Rating unit testing failer response	143
Figure 114: Donation rating unit test function	144
Figure 115: Donation rating unit test failer response	144
Figure 116: Number of system ration unit test function	145
Figure 117: Number of system ration unit test failer response	146
Figure 118: Number of r action unit test success functions	146
Figure 119: Number of ratio unit test success responses.....	147
Figure 120: Email Evrify unit test function	148
Figure 121: Email Evrify unit test failer response	148
Figure 122: Email verify success function	148

Figure 123: Email verify unit test success response.....	149
Figure 124: Ngo register unit test is a success function	150
Figure 125: Ngo register unit test success response.....	151
Figure 126: Login screen with filled the email and password	154
Figure 127: successfully navigate to the dashboard web admin panel.....	155
Figure 128: Register data saved in a database	156
Figure 129: Web testing to fill in the login details	157
Figure 130: Display the Aunatheneticate user doesn't login to the web admin panel ..	157
Figure 131: Display the piechart with data	160
Figure 132: Display the percentage of food history	161
Figure 133: Display the percentage of food users.....	161
Figure 134: Web testing for history table list	162
Figure 135: View history of user details.....	163
Figure 136: View the history of food details.....	163
Figure 137: Web test for deleting item show	165
Figure 138: Web test deletes an item that does not show.....	165
Figure 139: New user register empty register screen.....	167
Figure 140: Web testing to enter the user details to register	168
Figure 141: Web testing to register user display in Table	168
Figure 142: Web testing shows the inactive account.....	170
Figure 143: Web testing user account is successfully activated.....	170
Figure 144: Logout view card with logout button	172
Figure 145: Web logout testing to successfully navigate the login page	173
Figure 146: Register user filled in the user details in the mobile app	175
Figure 147: Register user success in the mobile app.....	176
Figure 148: Register user details in the database	177
Figure 149: Login to inactivate the user account.....	178
Figure 150: show the unauthenticated error message in the dialogue box	179
Figure 151: Activate the user account from the admin	180
Figure 152: Login to activate the user account.....	181
Figure 153: Login success message	182

Figure 154: Enter the valid email address to verify the user.....	184
Figure 155: Enter the new password and valid both same password	185
Figure 156: Display the success message daily box	186
Figure 157: View list of food history history	188
Figure 158: View food details	189
Figure 159: View donation location content.....	191
Figure 160: View the location on the map with the pick-up point.....	192
Figure 161: View location details with distance	193
Figure 162: View the user profile content	195
Figure 163: View user profile details	196
Figure 164: Empty donation details	198
Figure 165: Donate food details filled	199
Figure 166: Donation success message	200
Figure 167: View the latest donated food on the home page	201
Figure 168: View donate food detail for notification.....	203
Figure 169: Received push notification	204
Figure 170: View the number of notifications in the badge.....	206
Figure 171: View notification history.....	207
Figure 172: Enter the donation completed info with rating and images	209
Figure 173: Donation rating completed message.....	210
Figure 174: Distribute the location image and send	it to 211
Figure 175: View history with rating star.....	212
Figure 176: Update food Item.....	213
Figure 177: Change food details for an update	214
Figure 178: Display Updated food details.....	215
Figure 179: Delete donated food Item	216
Figure 180: Before detailing the food	217
Figure 181: Delete confirmation dialogue	218
Figure 182: Update donate food image	219
Figure 183: Before updating the food image	220
Figure 184: Food image update confirmation dialogue box.....	221

Figure 185: Updated food image	222
Figure 186: Report Item Menus.....	224
Figure 187: Write the report descriptions in the dialogue box	225
Figure 188: Report successful message	226
Figure 189: Before updating the username in the profile.....	228
Figure 190: Update food profile username	229
Figure 191: Latest updated user profile details	230
Figure 192: Future work Socket implementation for Chat.	240
Figure 193: Future work for multi-platform.	242
Figure 194: Multiplatform connection	242
Figure 195: Pre-survey form introduction with username	249
Figure 196: Pre-survey questions of age and occupation	250
Figure 197: Pre-survey questions of what could be donated and participants to the event	251
Figure 198: Preservay questions about the food of dump or donate and directly donate or through organizations.....	252
Figure 199: Pre-survey question of interest for donation and rating point of the system	253
Figure 200: Preservay questions of impact, updates, and suggestions for the system	254
Figure 201: Pre-survey questions filled with a sample of age and occupation	255
Figure 202: Pre-survey questions filled sample of what could be donated and participants to the event	256
Figure 203: Preservay questions about the food-filled sample of dump or donate and directly donate or through organizations	257
Figure 204: Pre-survey questions filled sample of interest for donation and rating point of the system	257
Figure 205: Preservay questions filled sample of impact, updates, and suggestions for the system.....	258
Figure 206: Post-survey questions of user experience	259
Figure 207: Post survey questions of user role	259

Figure 208: Post survey question of system navigation of surplus food donations system	260
Figure 209: Post-survey question of donations system which features is helpful.....	260
Figure 210: Post-survey of user recommend questions food donations system	261
Figure 211: Post-survey questions of a rating point to the overall system.....	261
Figure 212: Post-survey of feedback optional questions	262
Figure 213: Post-survey questions filled sample of user experience.....	263
Figure 214: Post survey questions filled sample of user role	263
Figure 215: Post survey question filled sample of system navigation of surplus food donations system	264
Figure 216: Post-survey question filled sample of donations system which features is helpful.....	264
Figure 217: Post-survey filled sample of user recommend questions food donations system.....	265
Figure 218: Post-survey questions filled sample of a rating point to the overall system	265
Figure 219: Post-survey of feedback optional filled sample	266
Figure 220: Register expanded use case.....	269
Figure 221: Sequence diagram of registered user	283
Figure 222: Login system activity diagram	285
Figure 223: Register system activity diagram.....	286
Figure 224: Logout system activity diagram	287
Figure 225: Splash screen mobile UI	288
Figure 226: Register response message mobile UI	289
Figure 227: Donor Profile screen mobile UI	290
Figure 228: Donor Setting screen mobile UI	291
Figure 229: Volunteer Setting screen UI.....	292
Figure 230: NGO Users screen UI	293
Figure 231: NGO Profile screen UI.....	294
Figure 232: Initial or Actual ER-Diagram	295
Figure 233: Final ER-Diagram.....	296

Figure 234: Register details DFD Level-2	298
Figure 235: Login details DFD Level-2.....	299
Figure 236: Login feature implementation in login API.....	300
Figure 237: Login Api Call	300
Figure 238: Login Repository Implementation.....	301
Figure 239: Login user use-case implementation.....	301
Figure 240: Login authentication view model implementation.....	302
Figure 241: Register user Api view implementation.	303
Figure 242: Register user repository implementation.....	303
Figure 243: Register user use case implementation.	303
Figure 244: Register user view model implementations.....	304
Figure 245: Update password Api view implementations.	305
Figure 246: Forgot password API call implementation.	305
Figure 247: Forgot password repository implementation.....	306
Figure 248: Forgot password use case implementation.	306
Figure 249: Forgot password view model implementation.	307
Figure 250: Update profile API call.....	308
Figure 251: Update profile repository implementation.	308
Figure 252: Update profile use case implementation.	309
Figure 253: Update user profile view model implementation.	309
Figure 254: Food update Api function implementation.	310
Figure 255: Update food details function and implementation in the view model	310
Figure 256: Update food details implementation in the use case	311
Figure 257: Implementation of the repositoryImpl in updated food details.	311
Figure 258: Update food details backend API call.....	312
Figure 259: Delete Api function implement.....	313
Figure 260: Delete Api call function Mobile.	313
Figure 261: Repositorium for Delete function	314
Figure 262: Food deletes use case implement.....	314
Figure 263: Food Delete View Model is implemented with state management.	315
Figure 264: Create the notification channel.....	316

Figure 265: Generating the notification permission and asking for permission.	317
Figure 266: Publish the local notification.....	317
Figure 267: Select the profile image.....	323
Figure 268: Display the confirmation message dialogue	324
Figure 269: View the latest updated profile image.....	325
Figure 270: Display Account deactivate dialogue box	327
Figure 271: Navigate the login page	328
Figure 272: Account deactivate and out logout the system with get local notification .	329
Figure 273: Change password input dialogue box	331
Figure 274: Enter the new and confirmation passwords in the dialogue box.....	332
Figure 275: Display the success message7.5.1.4 Logout App	333
Figure 276: Logout confirmation dialogue box	335
Figure 277: Logout success dialogue box.....	336
Figure 278: Before the Internet disconnects page.....	338
Figure 279: Display the no internet connection fails the screen	339
Figure 280: Display the local notification	341
Figure 281: Show more details about the local notification	342
Figure 282: System testing for Google Maps permission	343
Figure 283: System test for Google Maps permission.....	344

Acknowledgement

In closing, I would like to thank my external supervisor sincerely, Mr. Shekhar Timsina, and internal supervisor, Mr. Bishal GC, for their invaluable advice, support, and upbeat feelings during the completion of this report. I could not have finished this report with such excitement without their joint continuous work from the beginning and the helpful feedback that motivated us. Additionally, I would like to sincerely thank all the seniors and other students who helped me with the composition and compilation of this coursework. Their direct and indirect assistance inspired me to complete this coursework before the deadline. Their invaluable input and continuous commitment to editing several iterations of the report have given me the best chance to enhance my report-writing abilities and the standard of my upcoming and present coursework.

Finally, I want to sincerely thank the RTE Department, London Metropolitan University, and other professors and staff members for giving us the coursework requirements and setting up a conducive environment so that we could do our coursework without any confusion.

CHAPTER 1: INTRODUCTION

Many countries, including Nepal, grapple with development challenges, where significant portions of the population face unemployment and poverty. In urban areas, the plight of the homeless and impoverished individuals struggling to afford basic meals is particularly pronounced. Recognizing this pressing issue, initiatives have emerged to address food scarcity by redistributing surplus food resources to those in need. Among these initiatives is the Food Donation App, known as "Food Share," which harnesses technology to streamline the process of collecting and distributing surplus food.

1.1 Project Introduction

In the ever-evolving landscape of technological advancements, the Food Donation App, "Food Share," stands as a testament to the transformative power of innovation in addressing social challenges. Spearheaded by Yuvraj Tamang of ING Food Company, this project aims to revolutionise the charitable food donation sector by leveraging digital solutions. By providing a platform for efficient surplus food collection and distribution, the Food Share app bridges the gap between surplus food sources and vulnerable communities, reducing food waste and alleviating hunger.

By consolidating surplus food resources and facilitating their redistribution to those in need, the Food Share app embodies a paradigm shift in charitable food donations. Its user-friendly interface and innovative features empower individuals and organizations to contribute to the fight against food scarcity and waste. Through the collaborative efforts of stakeholders, this digital solution promises to make a meaningful impact in ensuring that no one goes hungry in communities where access to food remains a daily challenge.

1.2. Current Scenario

The Nepalese Government established the right to food as a fundamental right for its residents. In response, the Nepali government made it a top priority to guarantee that all of its residents had access to wholesome food. To do this, the government used the UN Food Systems Summit 2021 to evaluate the current food systems and investigate ideas for developing just, resilient, and sustainable systems. Aligning with the objectives of Periodic Plans, the Sustainable Development Goals (SDGs) by 2030, and the aim for national prosperity necessitated this.

In keeping with this goal, the National and Provincial Food Systems Conversations were held by the National Planning Commission (NPC) with the subject "Nepal towards an equitable, resilient, and sustainable food system." Important players from various food system domains convened for these discussions to examine the state of affairs at the municipal and provincial levels. With a focus on the policy environment, they concentrated on five action tracks that complemented the goals of the UN Food Systems Summit in 2021. Under the policy environment track, the Food Sovereignty Act and the Right to Food serve as the legal foundations for reforming Nepal's food systems, and these frameworks were the focus of discussion.

The previously mentioned discussions have been crucial in cultivating cooperation among many actors across the country and raising consciousness of the significance of food systems in advancing the well-being of humanity and the environment. It was clear that food systems are essential to fulfilling national plans, attaining national ambitions, and advancing the Sustainable Development Goals, which serve as the roadmap for all development initiatives, including Nepal's attempt to leave the list of least developed countries (Nepal, Nepal towards an equitable, resilient and sustainable food systems, 2020-2021). Developed nations discard vast quantities of perfectly edible food daily, contributing to environmental degradation and resource inefficiency. Supermarkets, restaurants, and households are major contributors to this surplus, often disposing of food due to overproduction, expiration dates, or aesthetic imperfections. The resulting environmental impact, including methane emissions from landfills, underscores the urgency of addressing this issue.



Dil Bahadur Gurung, PhD
Acting Vice Chairman

Government of Nepal
National Planning Commission
Singha Durbar, Kathmandu

Foreword

Globally more than 690 million people are suffering from hunger, about two billion adults are overweight and 462 million are underweight. It is quite disappointing to note that this problem is being aggravated by global crisis due to COVID 19 pandemic and present Russia-Ukraine war resulting in increased poverty and food insecurity.

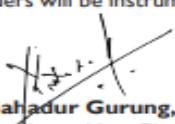
Nepal is not exception, despite good progress in food security and nutrition, we are living with a situation of unsustainable food systems, and our **unique mountain food systems** is **highly vulnerable** to the impact of climate change and other natural disasters. It is utmost important for us to achieve an equitable, resilient and sustainable food systems, which will also contribute to achieve Sustainable Development Goals (SDGs) by 2030. Realizing this, and as a member state of UN, the Government of Nepal in collaboration with stakeholders actively engaged in UN Food Systems Summit 2021 process and accorded high priority to transform its existing food systems.

The Food systems Dialogues (three National and seven Provincial) convened in 2021 identified six national pathways for transforming the food systems. **First** pathway emphasizes on ensuring policy coherence (especially in the agriculture, food security and nutrition, education, and health) and food governance; encourage farmers/youth to engage in agricultural sector to intensify the production of affordable, safe, healthy and nutritious food for all people in a sustainable way; **second** focuses on setting up of regulatory mechanisms to effectively monitor the quality of foods and educating people to consume healthy and nutritious local food, reduce food losses/waste, and promoting, protecting and supporting breastfeeding practices. Similarly, **third** pathway acknowledges the importance of adopting agroecosystem based resilient planning and revitalization of indigenous food systems and conserving and utilizing biodiversity. The **fourth** one emphasizes on investing in Research and Development and innovation in the agricultural sector to diversify the food systems and promote value chains; developing entrepreneurship skills of farmers including SMEs to raise their income and improve their livelihoods. **Fifth** pathway prioritizes for ensuring longer-term investments on developing resilient food systems and community to withstand shocks and stresses and final **sixth** one further highlights the role of the Right to Food and Food Sovereignty Act, to ensure accountable food governance at all levels for transforming the food systems.

This report elaborates all the pathways and strategic actions and provides a useful reference to all stakeholders, experts and practitioners.

I would like to extend my sincere thanks and appreciation to the UN System, development partners and key stakeholders for their strong commitments to contribute to transform our food systems and achieve SDGs by 2030. UN World Food Programme deserves special thanks for the support in coordinating provincial and national dialogues events as focal agency on behalf of UN agencies.

We understand that the countries like Nepal need to avail resources for increased investment, technology transfer, and capacity development that may suit the local needs. It is my firm belief that concerted efforts and joint actions by the governments, development partners, private sector and all stakeholders will be instrumental in this endeavour.


Dil Bahadur Gurung, PhD
Acting Vice Chairman
National Planning Commission

Tel: 977-01-4211970, Fax: 977-01-4211700, E-mail: dbgurung@npc.gov.np, Website: www.npc.gov.np

Figure 1: Current Scenario of Nepal context food problems.

Appendix A Current Scenario

1.3. Problem Domain and Project as a Solution

In the face of the multifaceted challenges posed by surplus food, the Food Share app emerges as a transformative solution. This visionary project leverages technology to streamline the collection and redistribution of surplus food, addressing not only the logistical hurdles of surplus food management but also the underlying issues of environmental degradation and food insecurity. By serving as a comprehensive platform, the app revolutionizes the process of surplus food distribution, ensuring that valuable resources are efficiently allocated to those in need.

The Food Share app lies in its ability to bridge the gap between surplus food providers and marginalized communities. Through its user-friendly interface, donors can easily input surplus food details, including type, quantity, and expiry dates, with location facilitating a seamless donation process. The app then matches surplus food to the specific requirements of shelters and communities, ensuring that resources reach where they are most needed. Moreover, the inclusion of educational resources and awareness campaigns empowers both donors and volunteers (recipients) with knowledge about waste reduction and responsible food utilization, fostering a culture of social responsibility.

By facilitating efficient surplus food distribution and promoting education and awareness, the Food Share app embodies a holistic approach to addressing the global challenge of food. Through its innovative features and commitment to sustainability, this project promises to make a meaningful impact in ensuring that no one goes hungry while valuable resources are not needlessly discarded.

1.4. Aim and objectives.

1.4.1. Aim

The food donation system aims to minimize food waste by efficiently redistributing surplus food to needy communities, reducing hunger, and fostering a more equitable and sustainable food distribution network.

1.4.2. Objectives

- Develop a user-friendly interface to facilitate surplus food input, donation matching, and recipient organization interaction.
- Establish a secure database management system to store and retrieve surplus food data efficiently, ensuring data integrity and confidentiality.
- Integrate an effective communication system between food providers and recipient organizations through API programming, enabling seamless coordination and information exchange.
- Implement innovative design elements, incorporating UI/UX principles and material design for an engaging user experience, enhancing accessibility and usability.
- Conduct comprehensive research on existing technologies and tools, identifying and integrating the most suitable ones for optimal application functionality, ensuring the app's effectiveness and efficiency.
- Ensure scalability and adaptability of the food donation platform to accommodate potential expansion and growth, allowing it to meet increasing demands and evolving needs.
- Promote community engagement and awareness through features that educate users about food waste reduction and proper food utilization, fostering a culture of responsible consumption and social responsibility.

1.5. Structure of the Report

1.5.1. Background

The development phase of the Food Share app involved the creation of both web and mobile platforms. The mobile platform, developed using Kotlin and Jetpack Compose, allows users to access the app's features seamlessly. On the other hand, the web platform, built using Django, HTML, CSS, and Bootstrap, serves as an administrative interface for managing the app's operations. Design considerations encompassed creating a user-friendly interface, establishing database management systems, and integrating communication systems for efficient food distribution. Throughout the implementation process, various challenges were encountered and addressed. These challenges ranged from technical complexities in integrating APIs to ensuring data security and scalability.

1.5.2. Development

The development phase of the Food Share app involved the creation of both web and mobile platforms. The mobile platform, developed using Kotlin and Jetpack Compose, allows users to access the app's features seamlessly. On the other hand, the web platform, built using Django, HTML, CSS, and Bootstrap, serves as an administrative interface for managing the app's operations. Design considerations encompassed creating a user-friendly interface, establishing database management systems, and integrating communication systems for efficient food distribution. Throughout the implementation process, various challenges were encountered and addressed. These challenges ranged from technical complexities in integrating APIs to ensuring data security and scalability.

1.5.3. Testing and analysis

The testing phase involved rigorous quality assurance measures to ensure the app's functionality, performance, and user experience met the desired standards. Testing methodologies included unit testing and system testing with user feedback. Testing focused on individual components of the app, verifying their correctness and robustness. Integration testing validated the interactions between different modules and systems, ensuring seamless functionality. The analysis of test results revealed areas of strength and areas for improvement. Positive outcomes included the app's intuitive user interface, efficient food donation matching, and secure data management. Areas identified for improvement included performance optimization, user engagement features, and accessibility enhancements.

1.5.4. Conclusion

The Food Share app project represents a significant step forward in leveraging technology to address surplus food and food insecurity. Despite challenges encountered during development, the project has succeeded in creating a platform that streamlines the process of surplus food redistribution and fosters community engagement.

Looking ahead continued efforts will be directed towards refining the app's features, addressing user feedback, and expanding its reach to serve more communities in need. By harnessing the power of technology and collaboration, the Food Share app stands poised to make a meaningful impact in reducing food waste and alleviating hunger.

CHAPTER 2: BACKGROUND

2.1. About the End Users

While most of Food Share's end users are members of the public who value and want to purchase art, the platform was originally designed to meet the requirements of a single artist who recognizes the importance of internet exposure and giving. The end users for the food share donation project encompass a variety of roles, each with specific functionalities and needs:

Client Name: **Yubaraj Tamang**

Client Information and Idea:

Yubraj Tamang is the employer of the ING Food Company's branch in Naxsal, which currently operates under ING Group. He is a beast innovator, helpful, and skilled in cooking. He has a knack for creating numerous food dishes and recipes. His passion for cooking extends beyond the kitchen, as he actively seeks solutions to minimize food wastage and contribute to the community through his culinary skills.

Currently, he works as a Cookman at Naxsal Herald College. Mr Tamang has been involved in cooking food and delivering it from Naxsal to Innovate Tech. He mentioned having an idea regarding the issue of food wastage, where sometimes food goes to waste due to overproduction or the absence of some employees. Consequently, this surplus food cannot be utilized the next day. He suggests that this excess food could be donated to poor or homeless people, thereby solving the problem of wastage. Thus, the food would not be discarded or wasted.

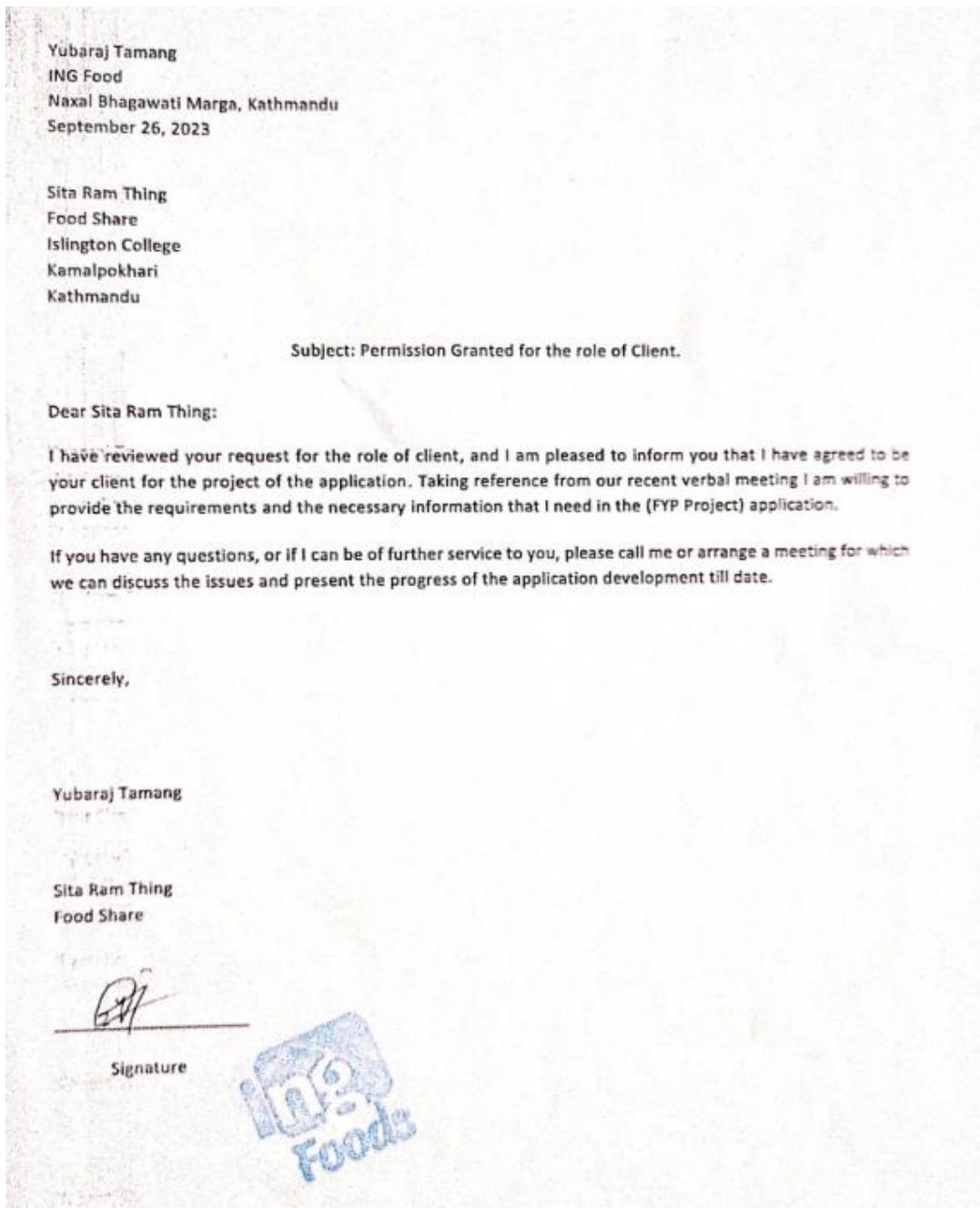


Figure 2: Client Approvable latter.

Appendix: B About the End User

2.2. Understanding the solution

The Food Share app addresses surplus food management by helping its collection and distribution to those in need. It runs on a web interface for administration and a mobile platform for funders, volunteers, and farmers. The system is intended to effectively connect surplus food sources with beneficiaries, reducing food waste and combating food insecurity. Here's a breakdown of its main components.

Web Interface (Admin):

Administrators control the whole system via a web browser. They handle user accounts, oversee contribution and allocation procedures, analyze data, and configure system settings.

Mobile Platform:

People or organizations may contribute food using the smartphone app. They may examine contribution history and receipts, as well as schedule pickups. Facilitate the collection and delivery of donated food, get information about pickups or distribution events, and track the status of deliveries. Coordinate surplus food contributions using the app by registering, providing produce information, indicating availability, and communicating about collection logistics. The registration procedure includes user verification by the administrator, which grants authenticated access. After logging in, users may view role-specific dashboards. The software supports password management, profile changes, and account deactivation.

2.2.1. Overview of the system

The Food waste food management system is the surplus food collected by the distribution system. The food provider (Donor) and volunteers (received) can communicate to pick the food from the donor and go to distribute to the poor and homeless people. Where the web is using the admin to control the system. The mobile user is a donor and volunteer where the donor can donate the available food and the volunteer can contract to receive the food from a donor.

The new user can register user details from mobile and admin goes to verify the user and give the authenticate access for using the system. After receiving the access, the user can log in to the food share app with, his/her selection role. The donor dashboard has all user's donated latest data. The donor can go to the post screen and new food is posted with all the food details. The donor after donating then goes to check the view history with details. Then the donor home page has multiple donated food view details with a location map and contract to the donor and received food and goes to distribute the food. After completing the distribution then view history with rating. The app has additional features such as forgetting passwords or changing passwords, permanent account deactivation, profile details update and more.

2.2.2. Technical Terms and Definition (Technical Terms related to the project)

To ensure understanding, technical terms related to the project should be defined. This section elucidates terms such as API, database management system, UI/UX, scalability, framework, programming language, firebase push notification etc., providing readers with a comprehensive understanding of the technology and concepts underpinning the Food Share app.

2.2.2.1 Django Rest Framework

Django is a framework used for creating the "behind-the-scenes" parts of websites. It organizes code and helps handle tasks like showing web pages, connecting to databases, and managing user accounts. With Django, developers can build these parts quickly and efficiently using the Python programming language. The rest framework is used to develop the APIs (Application Programming Interface). (MkDocs, 2024)

2.2.2.2 Application Programming Interface (API)

An API is a set of rules, protocols, and tools that allow different software applications to communicate with each other. It defines the methods and data formats that one application can use to request or manipulate data from another application. A computer code collection called an API allows data transfer across different software products. The conditions of this data exchange are also included. (Singtel, 2024)

2.2.2.3 HTML, CSS, JavaScript

The front end is what users see and interact with directly on a website. It includes things like buttons, forms, and menus, as well as the code that makes them work. Developers use technologies like HTML, CSS, and JavaScript to create these user interfaces, making sure they look good and respond smoothly to user actions.

2.2.2.4 MySQL Database

MySQL is a type of software used to store and organize data in a structured way. It's commonly used for websites and other applications that need to manage large amounts of information. MySQL offers features that help ensure data is stored securely, accessed quickly, and managed efficiently.

2.2.2.5 Room Database

Room is a tool used in Android app development to help manage data locally on a user's device. It makes it easier for developers to work with databases by providing helpful features and simplifying common tasks. The room helps ensure that data in apps is stored and accessed reliably, improving the overall performance and user experience. Verification of SQL queries at compile time. (Developer, Developer, 2024)

2.2.2.6 Python

Python is a programming language often used with it. Together, they make it easier for developers to build web applications by providing tools and resources to handle common tasks. Python is known for being easy to read and write, making it a popular choice for developers.

2.2.2.7 Kotlin

Kotlin is a statically typed programming language that runs on the Java Virtual Machine (JVM). It is fully interoperable with Java and is used for Android app development. It works with Kotlin, a programming language commonly used for Android development.

2.2.2.8 Jetpack Compose

Jetpack Compose is a tool used in Android app development to create user interfaces (UI) more easily. Jetpack Compose simplifies the process of building UIs, making it faster and more intuitive for developers to create visually appealing and interactive apps. Jetpack Compose is a modern UI toolkit for building native Android UIs. (Developer, Google for developer, 2024)

2.2.2.9 Firebase Push Notification

Firebase is a platform provided by Google that offers various tools and services to help developers build and manage apps more efficiently. It includes features like authentication (managing user accounts), real-time database (storing and syncing data across devices in real-time), cloud messaging (sending messages to users), and hosting (hosting web content). Firebase helps developers focus on creating great user experiences without having to worry about managing server infrastructure.

2.2.2.10 Serializer

A serializer is a program that translates the data or state of an object into a format that can be stored, transmitted, or reconstructed later in the same or another computer environment. With the use of serializers, complicated data such as query sets and model instances can be simply transformed into native Python datatypes for rendering into other content types, such as JSON or XML. After verifying the incoming data, serializers also offer deserialization, which enables parsed data to be transformed back into complicated types. (Django Rest Framework, 2024)

2.2.2.11 Data Transfer Object (DTO)

An object that carries data between processes. It's often used to transfer data between a client and a server in a distributed application. Data transfer objects (DTOs) differ from business objects or data access objects (DTOs) in that DTOs only exhibit the ability to store, retrieve, serialize, and deserialize data. (Weir, 2024)

2.2.2.12 Authentication

The process of verifying the identity of a user or system. Finding out if someone or something is who or what they claim to be is the process of authentication. By comparing a user's credentials to those in a data authentication server or a database of authorized users, authentication technology grants access control for systems. (Nick Barney, 2024)

2.2.2.13 Pojo (Plain Old Java Object)

A Java object that encapsulates only fields and doesn't contain any business logic. It is used to transfer data between different components of an application. Any Java program may use a plain old Java object (POJO), which is a class declaration unrelated to any Java framework. There are no specific naming guidelines or limitations for properties and methods in a POJO. Their main benefits are simplicity and reusability. (StevenSwiniarski, 2024)

2.2.2.14 FCM Token (Firebase Cloud Messaging Token)

A unique token is assigned to a mobile device by the Firebase Cloud Messaging (FCM) server. It is used to send push notifications to that specific device. A simple notification

channel with default parameters is offered by FCM. Set the default notification channel ID to the ID of your notification channel object as indicated if you would rather make and use your default channel. FCM will use this value if incoming messages do not specifically set a notification channel. (developer, 2024)

2.2.2.15 Access Token

A credential is used to access protected resources on behalf of an application. Through the use of an access token, a client application can access a certain resource and carry out particular tasks on the user's behalf. (Chiarelli, 2024)

2.2.16 Share Preference

A simple way to store small key-value pairs of primitive data types in Android applications is provided by a SharedPreferences object. It offers simple methods for reading and writing key-value pairs and refers to a file storing them. Each SharedThe preferences file has its structure and can be made private or shared. (Google, 2024)

2.2.3. Function and Features

The system has so many functions and features where some major functions and features are given below:

2.2.3.1. Login and Registration:

New users can create or register for the system with valid data and select their role. Authenticated users can access the app's functionalities. Once authenticated, users cannot log in to the system again.

2.2.3.3. Forgot/Reset Password:

Users can change or update their password before logging in if it's forgotten, and they can recover or reset forgotten passwords. The option to reset the password is available both before and after logging into the system.

2.2.3.4. Food Donation:

Enables donors to post surplus food details for donation. Donors can access information about the surplus food they've donated and notify all users.

2.2.3.5. View Food Details:

Provides comprehensive information about available food items. Users can view donated food items, the details of the donating user, the donation location, and the food's status. After completing the donation, users can view who will receive the food for distribution.

2.2.3.6. View Donation Locations:

Displays the locations for donation pickups. Both donors and volunteers can view donation locations via Google Maps. The map shows both the current user location and the food donation locations.

2.2.3.7. View Profile:

Users can access and update their profile information. All authenticated users can view user profile details. Also, users can view the NGO profiles.

2.2.3.8. Account Deactivation:

Allows users to permanently deactivate their accounts. If users are no longer interested in keeping their accounts active, they have the option to permanently deactivate them.

2.2.3.9. Report/Complaint:

If a donor donates food and the volunteer accepts it for distribution, but the donor cannot give the food to the volunteer, the volunteer can report to the admin. Similarly, if a volunteer cannot pick up the donated food, the donor can report to the admin.

2.2.3.10. Notification (Firebase push notification and Local notification):

When a donor donates food, all user devices receive a notification. Additionally, local notifications are triggered for actions such as forgetting passwords and account deactivation.

2.2.3.11. Rating

Volunteers can collect feedback from food consumers and provide ratings for the donated food. They can assign a rating ranging from a minimum of 0 to a maximum of 5 stars. This rating system helps to evaluate the quality of donated food and ensures continuous improvement in the donation process. Additionally, it encourages transparency and accountability within the food distribution network.

2.2.3.12 View Location in Map

The volunteer can utilize the map feature to locate donations or pickup points efficiently. Both donors and volunteers have access to view donation locations through the integrated Google Maps feature. This functionality enables volunteers to confirm and accept food donations seamlessly.

These functions and features ensure a seamless user experience, facilitating efficient surplus food management and distribution while promoting user engagement and accountability.

2.3. Comparison

The Food system is so many available in different countries on different platforms where similar projects to some features are different in food share applications. A similar project is Flashfloods, Food Reduce No Waste Food etc.

2.3.1. Similar Projects

2.3.1.1 Flashfoods

Author: local government bodies, NGOs,

Flashfoods is a mobile application that connects users with surplus food from local restaurants and grocery stores at discounted prices. Users can browse available food items, purchase them directly through the app, and pick them up from designated locations. Flashfoods aims to reduce food waste by offering discounted surplus food to consumers while providing additional revenue streams for businesses. Flashfood gives unbeatable deals on groceries at peak deliciousness. This application provides the save big on fruit, vegetables, meat, milk, cheese, pantry staples and more but is not fully free distributed. (local government bodies, 2024)

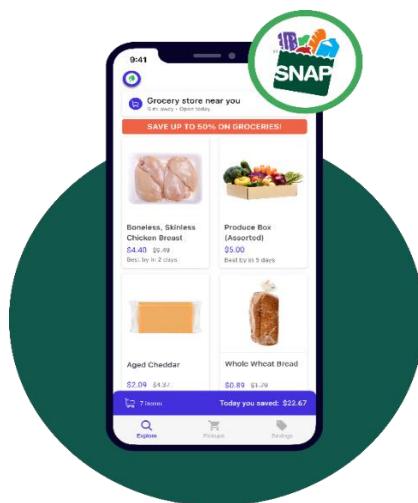


Figure 3: Flash floods similar app image

2.3.1.2 Food Rescue

Author: Dave Lampert, Melissa Spiesman, Jeff Schacher and Kevin Mullins

Jeff Schacher and Kevin Mullins founded Food Rescue as Community Plates in Fairfield County in 2011, in response to serious concerns about food insecurity and waste. The system's main goals were to reduce food waste and provide meals to people in need. This innovative approach garnered widespread support from volunteers, food donors, and social service organizations. Over time, they expanded and eventually renamed the nonprofit Food Rescue. It reduces food waste by enabling food donors to donate their surplus food to social service agencies. It also provides meals to those facing food insecurity by allowing social service agencies to join and receive food. To collect and share surplus food seven days a week, and there is no cost to the food donor or the recipient. (Dave Lampert, Now Fair Food NZ, 2024)



Figure 4. Food Rescue similar project image

2.3.1.3 No Food Waste

Author: Anne Frank

"No Food Waste" also collaborates with businesses, schools, and government agencies to implement strategies for reducing food waste in various settings. This includes initiatives such as food rescue programs, composting initiatives, and policy advocacy for standardized date labelling and food donation regulations. That aims to foster a culture of mindfulness and responsibility towards food consumption, ultimately contributing to a more sustainable and resilient food system. Through collective efforts and partnerships, the program seeks to minimize food waste and maximize the utilization of resources, ensuring that everyone has access to nutritious food while reducing the environmental impact of food production and disposal. (Dave Lampert, Food Reduces, 2024)



Figure 5: No Food Waste similar projects image.

2.3.2. Comparisons (Comparing the features and critical evaluation of the solution)

Features	Flash flood	Food Rescue	No Food Waste	Food Share
Food Post/Donation	Yes	Yes	Yes	Yes
View Location on Google Map	Yes	Yes	No	Yes
Push Notification	Yes	Yes	Yes	Yes
Local Notification	No	No	No	Yes
Email Notification for Distributed Message	Yes	Yes	Yes	Yes
Donation complaint for administration	No	Yes	Yes	Yes
Rating	No	No	No	Yes
User View History	No	No	No	Yes

Table 1: Comparison to a similar project

when comparing the features of Flashfood, Food Rescue, No Food Waste, and my system "Food Share", I find that all platforms support basic functionalities like food posting and donation, as well as push notifications. However, Food Share distinguishes itself with additional features such as local notifications, donation complaint functionality, rating, and user view history. Flashfood lacks donation-compliant capabilities, while No Food Waste does not offer location viewing on Google Maps. Food Rescue lacks a rating feature and user view history. Overall, Food Share provides a more comprehensive set of features, enhancing user experience and administrative capabilities.

CHAPTER 3: DEVELOPMENT

3.1. Methodology

The methodology used to develop the system where it can be helpful to the development process. The different types of projects are suitable for different methodologies.

3.1.1. Considered Methodologies

It is a broad topic that can be explored in various contexts, especially in the realm of research and project development. For a final-year project, understanding and explaining considered methodologies typically involves discussing the approaches, frameworks, or techniques that were evaluated and chosen for the project's development or research process. The three methodologies are research like waterfall, scrum and prototype.

3.1.1.1 Water Methodology

The waterfall methodology is a sequential, linear approach to project management in which a project's phases must be finished one after the other. It places a strong emphasis on comprehensive documentation, clearly stated project requirements, and an organized, set plan (Theil, 2024). The waterfall methodology has some phases available where planning, designing, development testing and deployment.

Its structured nature can be beneficial for projects with well-defined requirements and stable scope. However, the waterfall methodology also has limitations. For instance, it may not accommodate changes or updates well once the project has progressed beyond the initial planning phase. This rigidity can lead to delays or difficulties in adapting to evolving requirements or unforeseen challenges. Additionally, the waterfall approach may result in longer development cycles, as each phase must be completed before moving on to the next, potentially slowing down the overall project timeline. Therefore, while the waterfall methodology can be effective for certain types of projects, it may not be suitable for those that require flexibility, iterative development, or frequent stakeholder involvement.

The advantages and disadvantages of the Waterfall Methodology in the format you've provided:

Advantages of waterfall methodology:

- Clear project organization and control.
- Early error detection, saving development time.
- Efficient time and cost estimation.
- Defect recognition before advancing phases.

The disadvantage of waterfall methodology:

- Requires upfront clear requirements.
- Limited flexibility for changes.
- Lack of risk and security management.
- Not suitable for large projects due to sequential completion.

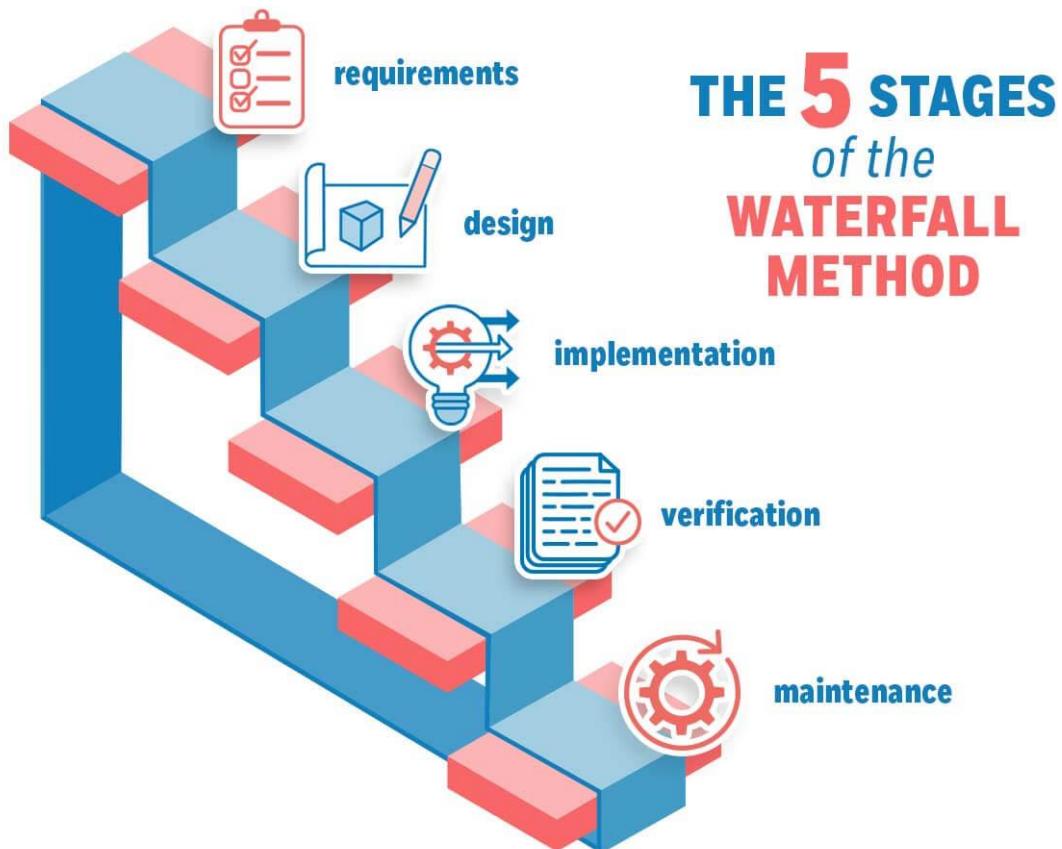


Figure 6: Waterfall methodology.

3.1.1.2 Scrum Methodology

Scrum is, in fact, an advancement of Agile Management. The Scrum technique is predicated on well-defined roles and processes that need to be adhered to throughout the software development lifecycle. This adaptable approach incentivizes the use of the 12 agile principles inside a framework that has been decided upon by every member of the product team (S, 2024). Scrum is a management methodology used to self-organize and work together toward common goals. It outlines a sequence of meetings, materials, and duties to expedite project completion. Scrum approaches enable teams to manage themselves, learn from prior experiences, and adjust to changing conditions, much like a sports team prepares for an important competition (AWS, 2024).

While Scrum offers flexibility and responsiveness, it necessitates ongoing management and engagement to ensure smooth progress throughout iterative cycles. In the dynamic realm of software development, Scrum orchestrates agile processes akin to a conductor leading a symphony.

Advantages of Scrum Methodology:

- Ensures efficient utilization of time and resources.
- Breaks down large projects into smaller, manageable sprints, enhancing focus and productivity.
- Ideal for rapid development projects, facilitating quick iterations and delivery.
- Enables concurrent coding and testing during sprint reviews, promoting continuous improvement.
- Embraces agility by actively soliciting feedback from stakeholders and customers, fostering collaboration and alignment.

Disadvantages of Scrum Methodology:

- Lack of a set completion date can lead to scope creep, potentially affecting project timelines and deliverables.

- Project failure is more likely in the absence of commitment or cooperation from team members.
- Implementing the Scrum framework can be challenging in large teams, requiring additional coordination and communication.
- Projects may face significant setbacks if key team members depart midway, impacting continuity and progress.



Figure 7: Scrum Methodology

3.1.1.3 Prototype Methodology

According to the prototype process, a functional system prototype must be constructed before any real software development is done. A model that serves as an implementation of the system is called a prototype. In most cases, a prototype is a relatively unfinished version of the real system that may have few functional features, poor dependability, and ineffective performance when compared to the real software. Often, the client's expectations of the software product are only partially clear. The prototype approach may be used in this situation if there is a lack of precise information about the system's input, processing requirements, and output requirements (JavaTpoint, 2024).

The prototyping methodology is an iterative software development approach where a prototype (a preliminary version of a product) is created to validate and refine design concepts before full-scale development. In this methodology, a basic version of the software or product is quickly developed and presented to stakeholders or end-users for feedback and evaluation. The prototype methodology allows for rapid development, early validation of ideas, and continuous improvement, leading to the development of successful and user-friendly products. Prototyping helps in refining requirements, understanding user needs, and validating design concepts before investing heavily in full-scale development. (LUMITEX, 2024)

Advantages of Prototype Methodology:

- Prototype Methodology provides flexibility in design, allowing for adaptable design processes.
- It aids in the easy identification of errors during development.
- The opportunity for refinement facilitates the smooth integration of new requirements.
- It enhances customer satisfaction and engagement through active user participation.
- User involvement in the development phase promotes collaboration and user-centric solutions.

Disadvantages of Prototype Methodology:

- The implementation of this approach may lead to high development costs.
- Variations in requirements can result in inconsistencies and frequent modifications.
- Rapid prototype development may lead to rushed solutions, compromising quality.
- Increasing complexity may arise within the system as prototypes evolve.
- Inadequate problem analysis may result in deficiencies within the model.

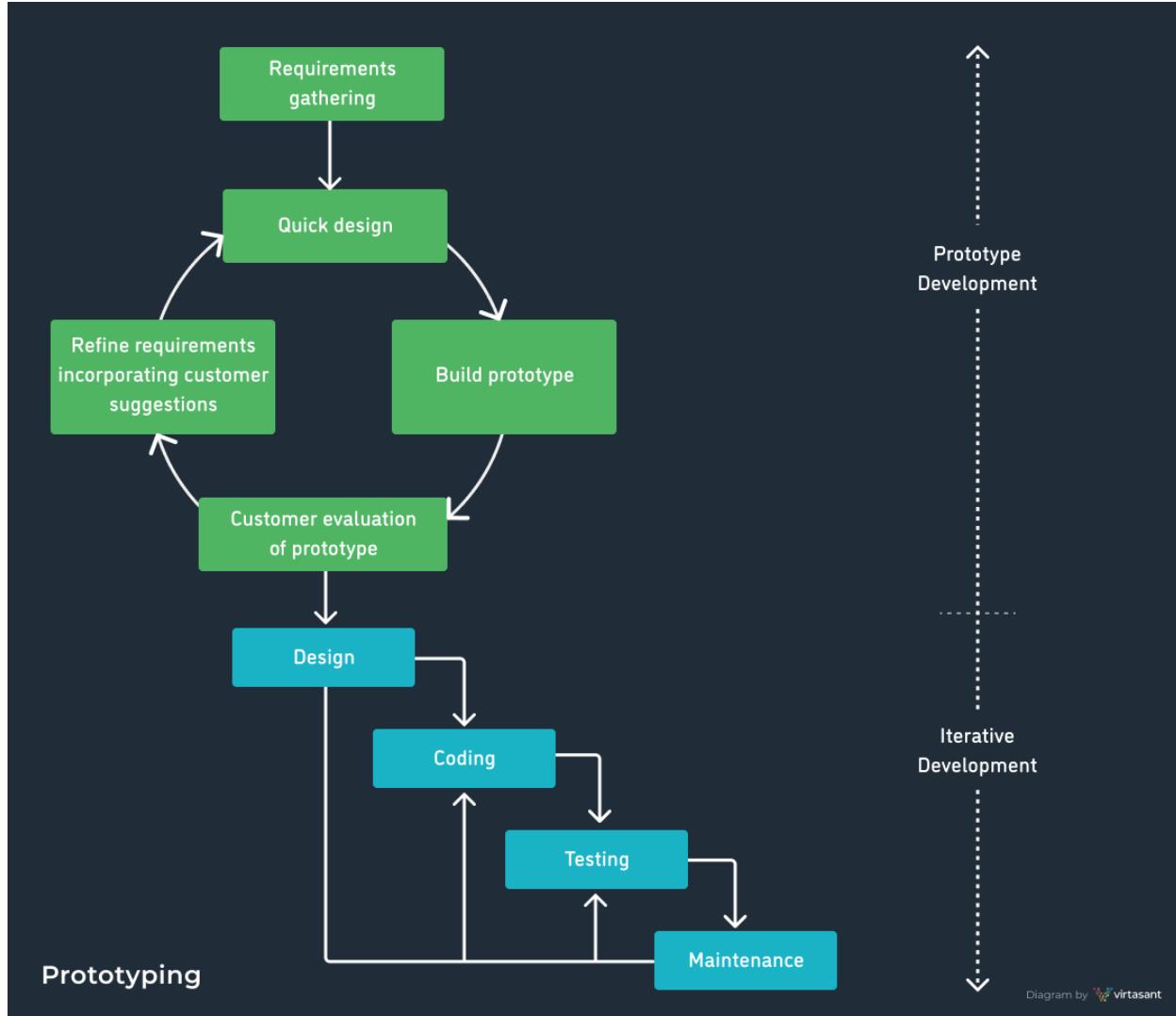


Figure 8: Prototype methodology.

3.1.2. Justification for Not Selected Methodologies

The three methodologies are researched to determine which is better suited for final-year project development. The waterfall and prototype methodologies are not selected due to their limitations, making them unsuitable for the project. Therefore, some limitations, problems, and issues to not suitable for the project so they are to not of these methodologies were chosen for the development of the FYP (Waste Food Management System).

3.1.2.1 Reason for not choosing Waterfall methodology.

Scenario	The food donation project requires a flexible approach due to evolving requirements and frequent adjustments during implementation.
Methodology	Waterfall Methodology
Justification	The Waterfall Methodology's linear progression may struggle to accommodate evolving project needs and frequent adjustments. In dynamic projects like food donation, where requirements evolve and continuous improvement is vital, a more adaptable methodology is preferred. Waterfall's rigid structure could hinder the project's ability to respond effectively to changing circumstances, potentially compromising its success.

Table 2: Justification scenario 1 for waterfall methodology

Scenario	Active stakeholder involvement and feedback are essential for the success of the food donation project, but the Waterfall methodology limits stakeholder engagement until the project's final stages.
Methodology	Waterfall Methodology
Justification	Waterfall typically restricts stakeholder involvement until late project stages, limiting opportunities for crucial feedback and collaboration throughout the project lifecycle. In projects like food donation, where stakeholder input is crucial for alignment with project goals and user needs, this restricted engagement could lead to misunderstandings and misaligned expectations. A more iterative methodology would better facilitate ongoing stakeholder involvement and feedback, enhancing project success.

Table 3: Justification scenario 2 for waterfall methodology

Scenario	The food donation project requires rapid iteration and learning from ongoing experiences to ensure its effectiveness.
Methodology	Waterfall Methodology
Justification	Waterfall's sequential approach does not support rapid iteration and learning from ongoing experiences. In projects like food donation, where continuous improvement and responsiveness are essential, a more iterative methodology is preferable. Waterfall's lack of flexibility and adaptability may hinder the project's ability to incorporate learnings and make necessary adjustments quickly, potentially affecting its effectiveness and success.

Table 4: Justification scenario 3 for waterfall methodology

The Waterfall methodology, known for its linear progression through predefined stages, may not be the most suitable choice for a Food Donation final-year project (FYP). Its rigid structure poses challenges in accommodating evolving project requirements and lacks flexibility for continuous adaptation. In a dynamic project like food donation, where needs might change, Waterfall's fixed approach could hinder the project's responsiveness to these changes. Therefore, opting for a more adaptable and flexible methodology, such as Agile or iterative approaches, could better cater to the project's evolving needs, fostering continuous improvement and responsiveness throughout the project's lifecycle. So, this methodology is not suitable for the project and is not chosen for system development.

3.1.2.2 Reason for not Choosing Prototype Methodology.

Scenario	The FYP project encounters ongoing changes in project scope due to evolving community needs or unforeseen challenges.
Methodology	Prototype Methodology
Justification	While Prototype Methodology emphasizes building and refining a working model based on clear initial requirements, it may struggle to keep pace with rapid changes in a project with continually shifting needs. The iterative nature of prototyping could lead to frequent modifications, creating inefficiencies and making it challenging to maintain a stable prototype aligned with evolving project needs.

Table 5: Justification scenario 1 for prototype methodology

Scenario	The FYP project has strict deadlines and limited resources for development and testing.
Methodology	Prototype Methodology
Justification	Prototyping involves iterative cycles of development, testing, and refinement. In environments with constrained time and resources, the iterative nature of prototypes may lead to delays or inefficiencies. This methodology may not be suitable when fixed deadlines must be met or when resource utilization needs to be optimized within a confined timeframe.

Table 6: Justification scenario 2 for Prototype methodology

Scenario	The FYP project lacks initial clarity regarding client requirements and expectations.
Methodology	Prototype Methodology
Justification	Prototyping relies on a clear understanding of requirements and continuous client involvement for iterative improvements. If client needs or project expectations are unclear initially, this methodology may struggle to deliver a refined prototype that aligns with evolving client expectations. The lack of clarity in the early stages could impede the iterative development process.

Table 7: Justification scenario 3 for Prototype methodology

In these scenarios, the Prototype methodology might face challenges in accommodating evolving scopes, managing limited resources and time constraints, and aligning with unclear stakeholder requirements. These limitations could impact its effectiveness for an FYP project focused on food donation. so will not select the prototype methodology.

3.1.3. Justification for Selected Methodology

The scrum methodology is selected for project development where the justification and reasons for Choosing the Scrum Methodology are given.

Scenario	The project encounters frequent changes in community needs or regulatory requirements.
Methodology	Prototype Methodology
Justification	Scrum allows seamless adaptation to changes. Its iterative nature accommodates evolving requirements, ensuring that the project remains aligned with dynamic community needs throughout development.

Figure 9: Justification scenario 1 for Scrum methodology.

Scenario	New features or functionalities need to be incorporated swiftly into the project.
Methodology	Prototype Methodology
Justification	Scrum enables the quick integration of new features. Its iterative cycles prioritize and incorporate new functionalities into subsequent sprints, ensuring timely integration without disrupting the development flow.

Figure 10: Justification scenario 2 for Scrum methodology.

Scenario	Continuous client review and feedback are crucial for project success.
Methodology	Prototype Methodology
Justification	Scrum allows seamless adaptation to changes. With its iterative nature, Scrum promotes regular review and feedback loops. Through sprint reviews and retrospectives, stakeholders can provide continuous input, ensuring alignment with their expectations throughout the project lifecycle.

Figure 11: Justification scenario 3 for Scrum methodology.

Scenario	Thorough testing for each feature or component completed in the project is essential.
Methodology	Prototype Methodology
Justification	Scrum mandates testing for every increment. By completing and testing features within sprints, the methodology ensures each component is thoroughly examined, reducing the risk of undetected errors or bugs.

Figure 12: Justification scenario 4 for Scrum methodology.

Scenario	Continuous improvement is essential to enhance project outcomes, where Scrum aids in the development phase.
Methodology	Prototype Methodology
Justification	Scrum fosters iterative improvement. Each sprint's retrospective allows the team to reflect on what went well and what needs improvement, leading to incremental enhancements throughout the project.

Figure 13: Justification scenario 5 for Scrum methodology.

Scenario	The project may encounter sudden priority shifts due to unforeseen emergencies.
Methodology	Prototype Methodology
Justification	Scrum's flexibility in prioritizing tasks helps handle emergencies. The method allows reprioritization within sprints, ensuring that urgent tasks can be addressed without disrupting the project's overall progress.

Figure 14: Justification scenario 6 for Scrum methodology.

Scrum's adaptability to change, rapid feature integration, regular review cycles, comprehensive testing, task prioritization flexibility, and iterative improvement capabilities make it a highly suitable methodology for the structured and efficient development of a Food Donation System FYP project. Therefore, I chose the Scrum methodology.

3.2 Survey Result

3.2.1. Pre-Survey Results

When developing the project, I researched to gather information related to the food donation system. Then, I created the survey form and provided it to both the end users and my client. The survey aimed to collect data from users and clients, as outlined.

3.2.1.1 Pre-survey questions



The figure shows a pre-survey form titled "Waste Food Management Survey Form". At the top, there is an illustration of two hands reaching towards each other over a sign that reads "FOOD DONATION". Below the title, there is a toolbar with icons for bold, italic, underline, link, and delete. A descriptive text block states: "This survey will help in understanding user needs and preferences before designing or improving a waste food management system. This survey has been conducted as a part of the Final Year Project. The survey may take around a few minutes but is not compulsory if you are a helpful person and have free time please give the right or wrong response. Thank you." Two input fields are present: one for "Write your full name? *", which is a short-answer text field, and another for "Write your email address? *", also a short-answer text field.

Figure 15: Pre-survey form introduction with username

Appendix: C Pre-Servay Form Questions

3.2.2.2 Pre-survey response

Upon completion of system development and its dissemination to users and clients, feedback was collected to assess the effectiveness and gather insights for further improvements. Users and clients provided valuable input regarding the system's functionality, usability, and overall impact on surplus food donation initiatives. Their feedback highlighted areas of success, such as increased awareness, improved accessibility to donation locations through the integrated map feature, and streamlined communication between volunteers and donors. Additionally, constructive criticism was offered, identifying areas for refinement, including user interface enhancements and logistical optimizations for food distribution processes. The pre-survey result response is given below:

Select your age?

30 responses

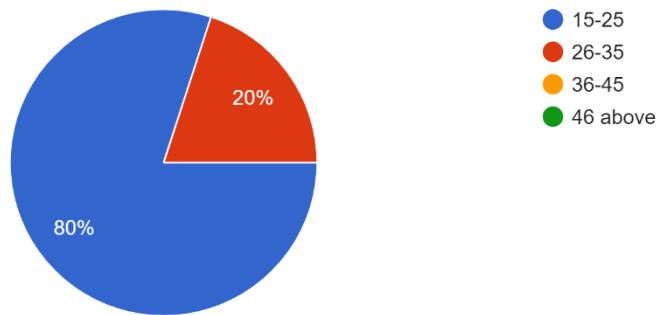


Figure 16: Pre-survey percentage of user age

Select your occupation?

30 responses

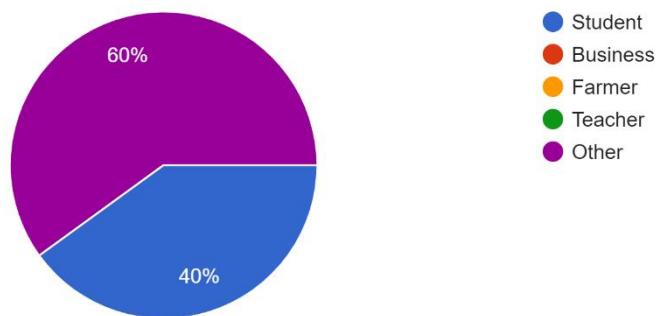


Figure 17: Pre-servery user occupations

In your life how often do you find yourself with excess food that could be donated?

30 responses

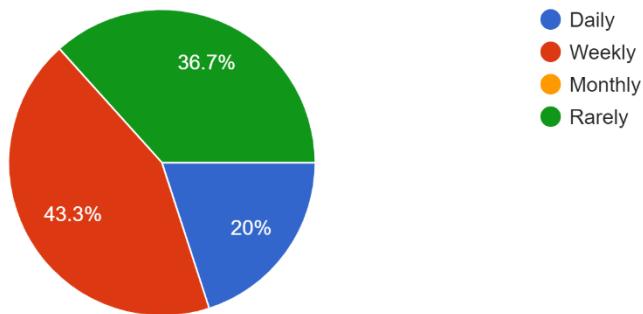


Figure 18: Preservay result of excess food for donated

Any social programs have you heard of or participated in for food donation
30 responses

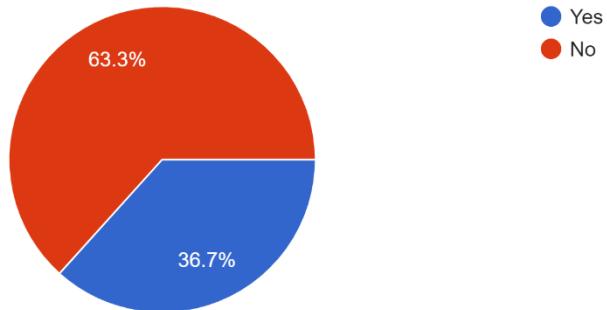


Figure 19: Pre-survey result per cent of users participated in the donation program

In society why do so many people not donate food and instead waste it in a dump?
30 responses

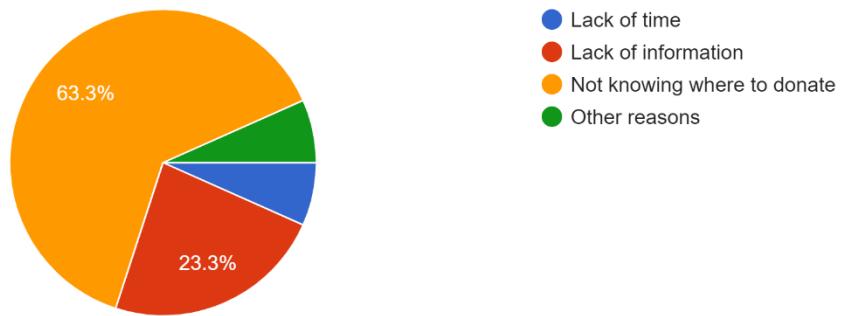


Figure 20: Pre-survey result user can food donates or dump

If you are interested in food donate you prefer to donate food directly to individuals or through organizations?

30 responses

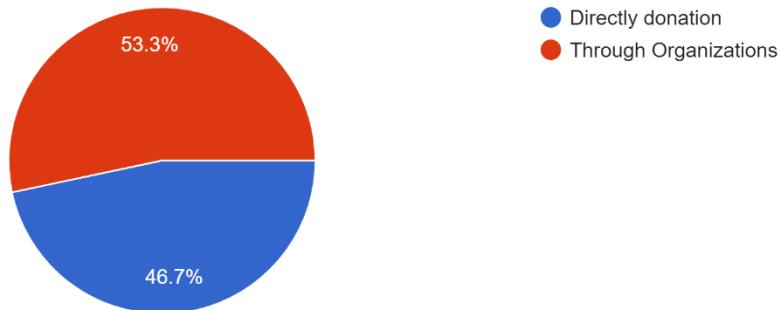


Figure 21: Pre-survey result of user interest directly or through donations organizations

In upcoming days you have extra food available are you interested in donating to homeless or poor people?

30 responses

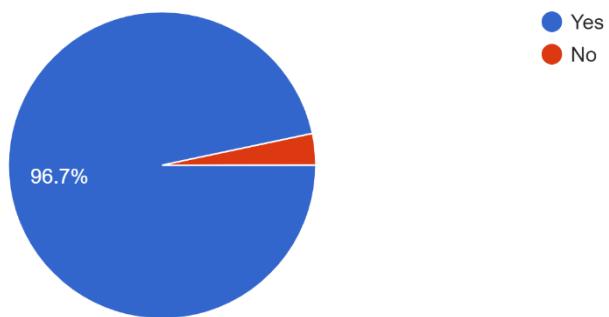


Figure 22: Pre-survey result of user interest in food donation

how would you rate an easy and quick online food donation process?(Rating Scale: 1-5)

30 responses

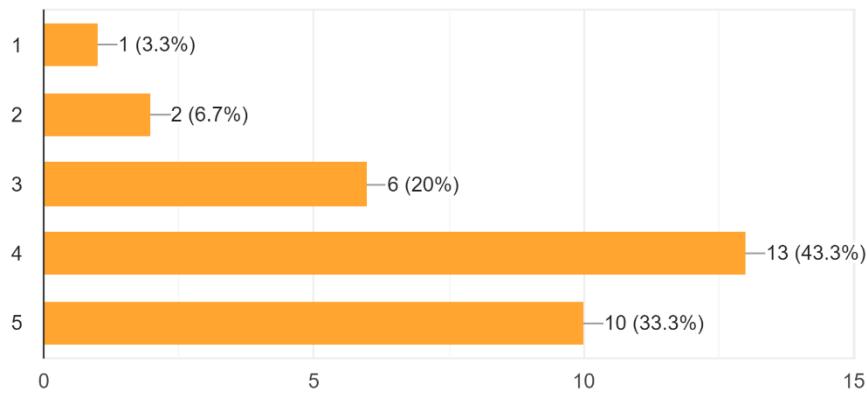


Figure 23: Pre-survey rating point of food donation idea

If you like to track the impact of food donations what kind of impact tracking would interest you?

30 responses

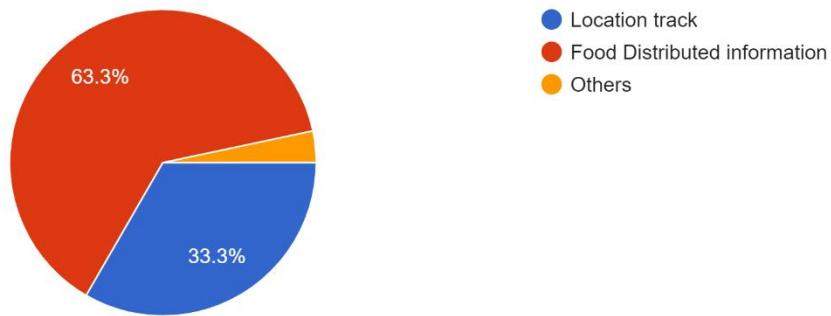


Figure 24: Pre-survey results for use need donation tracking

How would you prefer to receive updates from a food donation app?

30 responses

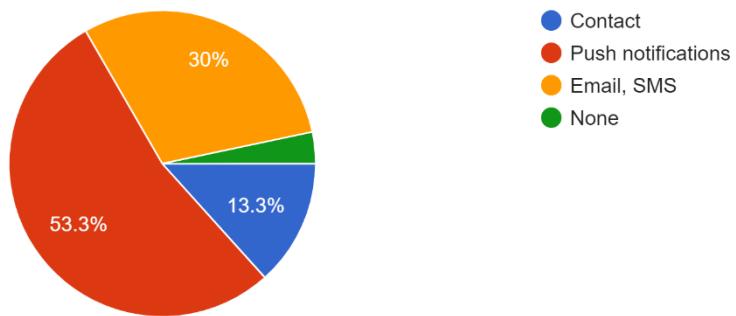


Figure 25: Pre-survey result of the user is interested in receiving the update

Do you have any ideas about food management that you could suggest for the food donation app?

30 responses

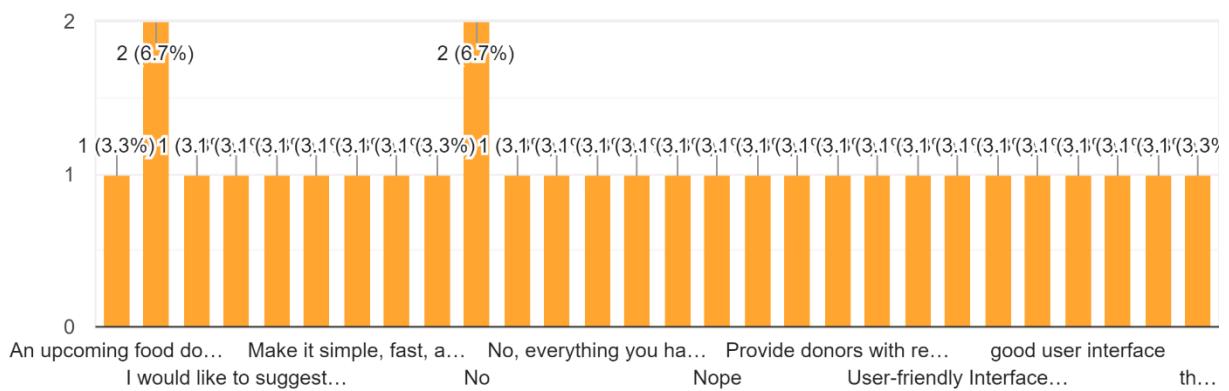


Figure 26: User fee back for system requirement

Users provided valuable information and feedback, from which I collected the best ideas and data. Following the survey, several features of the project were changed, primarily based on user suggestions. Users mainly focused on User Interface improvements and enhancements related to the donation process and donation information.

3.2.2. Post-Survey Question

In the post-survey analysis, following the completion of system development and its distribution to users and clients, feedback was collected to assess its effectiveness and gather insights for further improvements. Users and clients provided valuable input regarding the system's functionality, usability, and overall impact on surplus food donation initiatives. Their feedback highlighted areas of success, such as increased awareness, improved accessibility to donation locations through the integrated map feature, and streamlined communication between volunteers and donors.

The image shows a screenshot of a survey form titled "Waste Food Donation System (Food Share)". The form has a decorative header featuring coffee cups and a French press. Below the title, there are text input fields for "Email *". The placeholder text in the first field is "Valid email address". A note below the fields states, "This form is collecting email addresses. [Change settings](#)". There is also a field for "Please fill your full name *". The placeholder text in this field is "Short-answer text". At the bottom of the form, there is a blue footer bar.

Figure 27: Post-survey introductions with questions of email and username

[Appendix: C Post-Survey Questions](#)

3.2.3 Post survey result

The overall system development part is completed and the release build creates and uploads the wormhole and shares the app for app downloads with the survey form. The user can give the best response that is given below:

What is your occupation?

19 responses

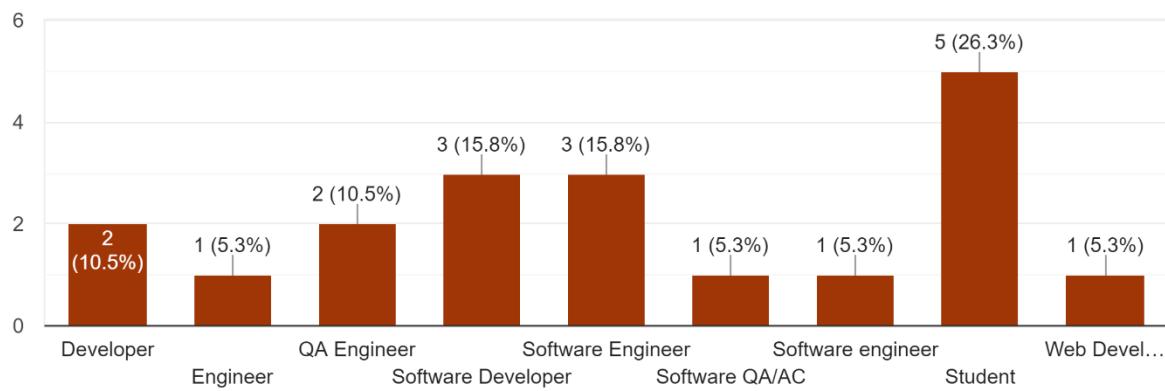


Figure 28: Post-survey of user occupations

How satisfied are you with your experience using our surplus food donation system?

20 responses

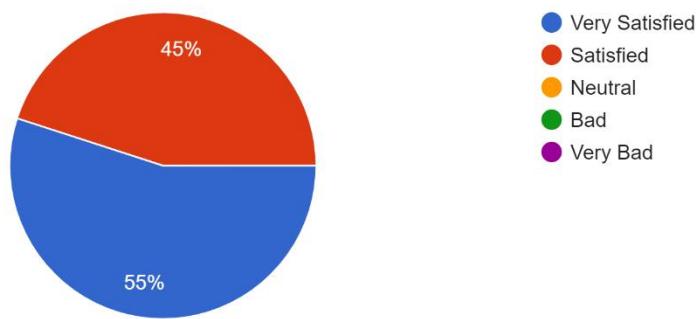


Figure 29: Post survey results of user experience

Which role did you primarily use the surplus food donation system for?

20 responses

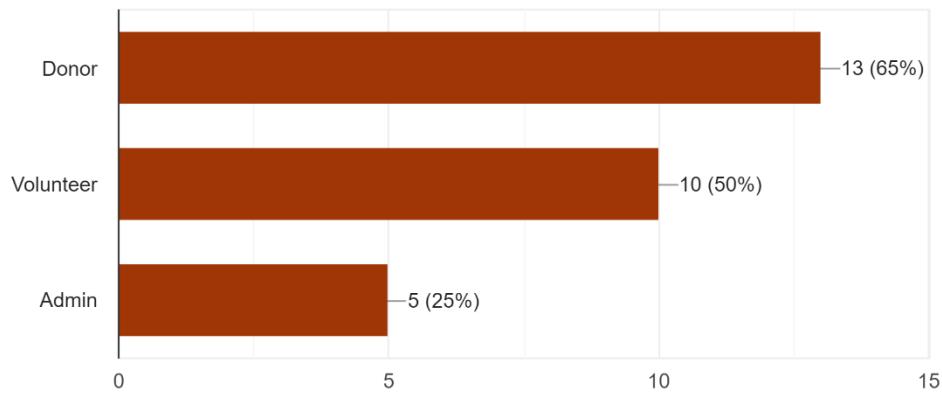


Figure 30: Post-survey results used to recommend the system

How easy was it to navigate and use the surplus food donation system?

20 responses

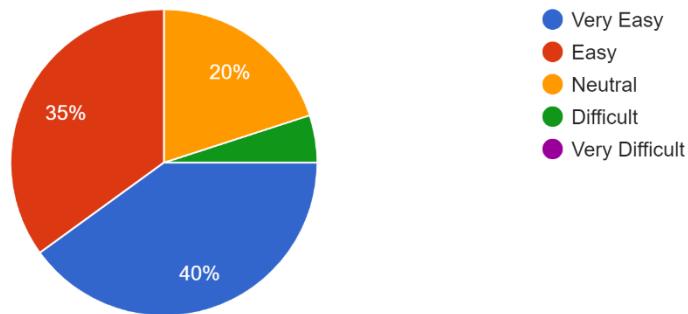


Figure 31: Post-survey of system navigation survey response

Which features of the surplus food donation system did you find most helpful?

20 responses

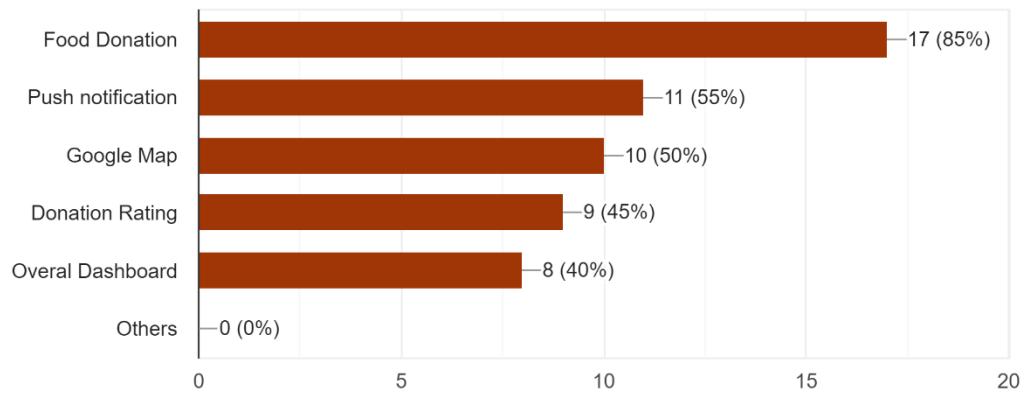


Figure 32: Post-survey result of finding the helpful

How likely are you to recommend our surplus food donation system to others?

20 responses

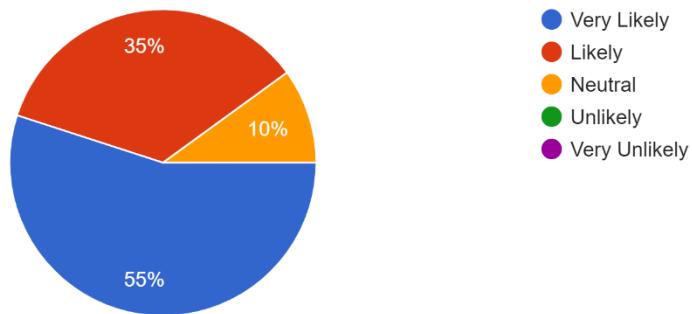


Figure 33: Post survey results of recommendations for donations

Rate the ease and speed of an online food donation process from 1 to 5, with 1 being the lowest and 5 the highest.

20 responses

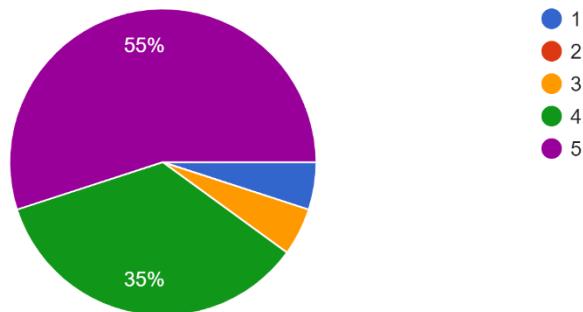


Figure 34: Post-survey of overall system rating

Additionally, the constructive criticism was offered, identifying areas for refinement, including user interface enhancements and logistical optimizations for food distribution processes. Overall, the post-survey results serve as a valuable resource for ongoing system refinement and the continued success of surplus food donation efforts. The survey form details and response with collection results are given below:

3.3. Requirement Analysis

3.3.1 Introduction

Requirement analysis for a surplus food management system involves understanding the needs and expectations of stakeholders and defining the specific functionalities and features that the system should have. The "Food Share" application is used in different locations, such as food companies, hotels, restaurants, and more sectors to surplus food to distribute to poor or homeless people. A recent analysis found that 1.3 billion tons of food are wasted annually. Further, leftovers account for one-third of all the food consumed. Currently, those who would like to donate food or other supplies must go in person to the organizations. If they need help, they will need to look for websites where they can give extra food. Large producers, distributors, and the organized community donate food to food banks or waste tons of food daily (Kruthika V, INTEGRATED APPROACH FOR FOOD DONATION SYSTEM, RESTAURANT, 07, July-2023).

3.3.2 Purpose

The purpose of the requirement analysis is to identify and document the functional and non-functional requirements of the "Food Share" application. By gathering input from stakeholders and understanding their needs, the analysis aims to define the scope of the system and specify the features and capabilities it should provide. Additionally, the requirement analysis helps prioritize requirements, identify constraints, and lay the foundation for the design and development of the system.

3.3.3 Scope

The scope of the "Food Share" application encompasses the following aspects:

- **Donor Management:** The system should allow food companies, hotels, restaurants, and other donors to register, log in, and submit details about surplus food donations, including type, quantity, and expiration date.

- Recipient Management: Recipient organizations, such as shelters, food banks, and community centres, should be able to register, request specific types of food donations based on their needs, and coordinate pickups.
- Logistics and Coordination: The system should facilitate the scheduling and coordination of food pickups between donors and recipients, optimizing routes and ensuring timely delivery.
- Reporting and Analytics: Administrators should have access to reporting tools to track donation activities, analyze trends, and measure the impact of the system in reducing food waste and addressing food insecurity.
- User Authentication and Security: The system should implement secure user authentication mechanisms to protect sensitive information and prevent unauthorized access.
- Usability and Accessibility: The user interface should be intuitive and accessible, catering to users with varying levels of technical proficiency and accessibility needs.
- Performance and Scalability: The system should be able to handle a large volume of donations and users, ensuring smooth operation and scalability as the user base grows.
- Regulatory Compliance: The system should comply with relevant regulations and standards related to food safety, data protection, and privacy.

By defining the purpose and scope of the requirement analysis, stakeholders can align their expectations and ensure that the "Food Share" application meets the needs of all parties involved in surplus food management.

3.3.4 Reference

AWS. (2024, 04 13). *What is Scrum?* Retrieved from AWS:
<https://aws.amazon.com/what-is/scrum/#:~:text=Scrum%20is%20a%20management%20framework,experience%2C%20and%20adapt%20to%20change>.

Chiarelli, A. (2024, 04 19). *Auth0 by Okta*. Retrieved from THE CONFUSED DEVELOPER: <https://auth0.com/blog/id-token-access-token-what-is-the-difference/>

Dave Lampert, M. S. (2024, 04 07). *Now Fair Food NZ*. Retrieved from Food Rescue: <https://fairfood.org.nz/about-us/our-story/#impact>

Dave Lampert, M. S. (2024, 04 07). *Food Reduces*. Retrieved from Food Reduces Us: <https://foodrescue.us/our-app/>

Developer, G. (2024, 04 23). *Developer*. Retrieved from Save data in a local database using Room: <https://developer.android.com/training/data-storage/room>

developer, G. (2024, 04 18). *FCM*. Retrieved from Firebase: <https://firebase.google.com/docs/cloud-messaging/android/client>

Developer, G. (2024, 04 23). *Google for developer*. Retrieved from Build Better Apps Faster with: <https://developer.android.com/develop/ui/compose>

Dhanajay Jhala, K. s. (2021). Spread the smile. *Food Donation*, p. 51.

Django Rest Framework. (2024, 04 22). Retrieved from Serializers: <https://www.django-rest-framework.org/api-guide/serializers/>

GeekForGeek. (2024, 04 21). *GeekForGeek*. Retrieved from Levels in Data Flow Diagrams (DFD): <https://www.geeksforgeeks.org/levels-in-data-flow-diagrams-dfd/>

GeekForGeek. (2024, 04 21). *Unified Modeling Language (UML)*. Retrieved from Use Case Diagrams: <https://www.geeksforgeeks.org/use-case-diagram/>

Google. (2024, 04 17). *Developer*. Retrieved from Save simple data with SharedPreferences: <https://developer.android.com/training/data-storage/shared-preferences>

Inc, L. S. (2024, 04 21). *Lucidechart*. Retrieved from Entity Relationship Diagram: <https://www.lucidchart.com/pages/er-diagrams>

JavaTpoint. (2024, 04 13). *Software Engineering*. Retrieved from www.javatpoint.com: <https://www.javatpoint.com/software-engineering-prototype-model>

Kruthika V, L. H. (07, July-2023). INTEGRATED APPROACH FOR FOOD DONATION SYSTEM, RESTAURANT. *International Research Journal of Modernization in Engineering Technology and Science*, 8.

Kruthika V, L. H. (07, July-2023). INTEGRATED APPROACH FOR FOOD DONATION SYSTEM, RESTAURANT. *International Research Journal of Modernization in Engineering Technology and Science*, 8.

Levan, M. (2024, 04 20). *TeachTargate*. Retrieved from Unit Testing: <https://www.techtarget.com/searchsoftwarequality/definition/unit-testing>

local government bodies, N. (2024, 04 07). *Flashfood Grocery deals*. Retrieved from Google Play Store:

<https://play.google.com/store/apps/details?id=com.flashfoodapp.android&hl=en&gl=US>

LUMITEX. (2024, 04 13). Prototyping Methodology. *Dow Circle Strongsville, Ohio*, p. 10.

Mello, M. (2024, 04 21). *System Architecture*. Retrieved from The Blueprint for Successful Systems: https://dev.to/msmello_/system-architecture-the-blueprint-for-successful-systems-35nn

MkDocs. (2024, 04 21). *Django REST framework*. Retrieved from Django: <https://www.django-rest-framework.org/api-guide/serializers/>

Nepal, G. o. (2020-2021). Nepal towards an equitable, resilient and sustainable food system. *Nepal's Food Systems Transformation*, 64.

Nepal, G. o. (2020-2021). Nepal towards equitable, resilient and sustainable food systems. *Nepal's Food Systems Transformation*: p. 64.

Nick Barney, M. E. (2024, 04 21). *TeachTargate*. Retrieved from authentication: <https://www.techtarget.com/searchsecurity/definition/authentication>

Reichert, A. (2024, 04 20). *TeachTargate*. Retrieved from system testing: <https://www.techtarget.com/searchsoftwarequality/definition/system-testing>

S, B. (2024, 04 13). *nimble Humanize work*. Retrieved from Scrum Project Management: <https://www.nimblework.com/agile/scrum-methodology/>

Singtel, P. (2024, 04 23). *Altexsoft*. Retrieved from What is API: <https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/>

StevenSwiniarski. (2024, 04 18). *Codecademy*. Retrieved from POJO: <https://www.codecademy.com/resources/docs/java/pojo>

Theil, D. (2024, 05 13). *LogRocket*. Retrieved from Understanding the waterfall methodology: <https://blog.logrocket.com/product-management/waterfall-methodology/#:~:text=The%20waterfall%20methodology%20is%20a%20linear%20and%20sequential,well-defined%20project%20requirements%2C%20and%20a%20structured%2C%20pre-determined%20timeline.>

Weir, S. P. (2024, 04 22). *Wikipedia*. Retrieved from Data Transfer Object: https://en.wikipedia.org/wiki/Data_transfer_object

Nepal, G. o. (2024, 04 15). Nepal's Food Systems Transformation: *Nepal towards an equitable, resilient and sustainable food system*, p. 64.

3.3.5 Overall Descriptions

- Product Perspectives: This section will describe how the process of donating food fits into the larger scheme of charity activities and community assistance programs. It would go over the many parties involved, including administrators, volunteers, donors, and recipients, and how the system attempts to meet their requirements while rendering coordination and communication easier between them.
- Product Functionality: The characteristics and capabilities of the food donation system are described in detail in this section. For volunteers, donors, and beneficiaries, user registration and verification are required. Interface for submitting donations that allow food givers to list available food products with information such as kind, amount, expiration date, and preferred pickup or delivery. Recipients can use the search and browse features to locate available donations according to their location and dietary requirements. Tools for scheduling and coordinating the delivery or pickup of donations between donors and receivers. Tools for managing volunteers that are used to recruit, assign, and monitor volunteer tasks including gathering, sorting, and distributing food. Administrators may quantify the impact, track donation trends, and improve operations with the help of reporting and analytics capabilities.
- System Environment: The technical setting in which the food donation system functions is covered in this section. It would contain information like The architecture or platform that was utilized to create the system (such as a mobile app or web-based application). Requirements for the backend infrastructure, including servers, databases, and APIs. Compatibility across a range of hardware and operating systems. Security protocols are put in place that protect user information and maintain system integrity. Integration with external services or platforms, such as mapping services for location-based functionality or payment gateways for contribution transactions.

In general, the goal of a food donation system is to make the act of giving food to those in need easier by using technology to match surplus food donors with receivers who can use it, and by enlisting volunteers to help with the logistical aspects of the program.

3.3.6 Functional Requirement

- Register Membership: Users can register their details and submit them for membership. The admin reviews the user details and verifies the account, granting privileges or permissions accordingly.
- View History: Both donors and volunteers can view their donation history. Donors can see who accepted their donated food, while volunteers can view the history of accepted donations available for distribution.
- View Donation Details: Volunteers can access the latest donated food details, including user information and location.
- View Map: All users can access the food donation locations to find the nearest pickup and distribution points.
- Push Notification: When a donor donates new food with all the necessary details, all volunteer users receive a notification alerting them to the donation, including the food name.
- Complaint/Report: Users have the option to file a complaint with the admin. If a donor fails to provide the promised food or if volunteers are unable to pick up the food, both parties can report the issue to the admin.
- Donation Rating: After completing a donation, volunteers can provide feedback and rate the donation.
- View User Profile: Users can view their profile details.

3.3.7 External Interface Requirements

This project needs various hardware and software tools, technology is given below:

- Emulator: Android Mobile or Default Android Studio Emulator
- Internet Connection • IDE: Android Studio, Ngrok, wormhole etc.
- Code Editor: Visual Studio Code
- Programming Language: Kotlin, Python, HTML/CSS
- Framework: Django, Rest and JetPack Compose
- Database: MySQL, Room
- Version Control: GitHub
- Tools: Draw.io, TeamGantt, Balsmiq, Scrcp, wormhole, Drive etc.

3.3.7.1 Hardware Requirements

S. N	Requirement	Version	Resources
1	Development Device	Computer Window 11	Computer (Window)
2	Test Device	Android 13 (above 8)	Mobile (Android)

Table 8: Hardware interface requirement

3.3.7.2 Software Requirements

S. N	Tools & Technology	Version	Purpose
1	Django (DRF)	4.2.5	Django Rest Framework Integrated for Api Development
2	Python	3.11.2	The Python programming language used to develop the Django backend.

3	MySQL Database	8.0.3	The MySQL database is used to data store in the cloud.
4	HTML, CSS, JavaScript		The HTM, CSS and JavaScript were used to develop the frontend web admin panel for system management.
5	Kotli	1.9.0	The Kotlin programming language was used to develop the Mobile application.
6	Jetpack Compose	1.6.5	The Jetpack compose is used to develop the Mobile User Interface (UI).
7	Room Database	2.5.2	The room database is a local database where it is used for temporary data saved for local storage.
8	GitHub	2.38.1	The GitHub is a version control tool that is used for all-project development and is stored in the cloud.
9	Android Studio	2023.1.1	The Android Studio is the IDE that is used to write the mobile app development code Kotlin and compose.
10	Visual Studio Code	1.70	The Visual Studio Code is used to develop backend Django rest framework and Frontend web applications.
11	Design and Documentation Tools		Design and documentation tools like MS Word were used for documentation, Draw.io, was used for class diagram development and Team Gent was used for Gantt chart development also more tools were used for the project.

3.3.7.3 User Interface

To address the issue of the food donation system, here are some suggestions for improvement:

- Clarity and Consistency: Ensure that the user interface (UI) is clear and consistent throughout the system. This includes using consistent design elements such as colours, fonts, and layout across all pages. Consistency helps users navigate the system more easily and reduces confusion.
- Simplicity: Simplify the UI to make it more user-friendly. Avoid cluttered layouts and excessive text. Instead, focus on presenting key information and features clearly and concisely. This improves usability and enhances the overall user experience.
- User-Friendly Language: Use language that is familiar and easily understandable to the system's intended users. Avoid technical jargon or complex terminology that may be confusing. Use simple language to guide users through the system's features and functionalities.
- User Guidance: Provide clear instructions and guidance to users at each step of their journey through the system. This includes providing tooltips, hints, or on-screen prompts to help users understand how to navigate the system and perform various actions.
- Accessibility: Ensure that the UI is accessible to all users, including those with disabilities. This may involve providing alternative text for images, ensuring proper contrast for text and background colours, and implementing keyboard navigation shortcuts.
- Feedback Mechanisms: Incorporate feedback mechanisms into the UI to allow users to provide input or report issues. This could include feedback forms, rating systems,

or contact options for reaching out to support staff. Feedback helps identify areas for improvement and ensures that user needs are addressed.

By implementing these suggestions, the food donation system can enhance its user interface to make it more attractive, user-friendly, and effective for donors, volunteers, and other users.

3.3.8 External Requirement

To develop an effective food donation system in Nepal, adhering to national and international best practices would be necessary. An overview of potential external requirements is provided below:

3.3.8.1 Legal Compliance:

Make sure that your food donation activities adhere to all applicable laws and rules in Nepal concerning food distribution, safety, and charitable purposes. This could entail getting the required permits and registering your organisation with the appropriate government agencies. Follow the food safety guidelines established by the Nepali government or by international agencies like the Food and Agriculture Organization (FAO) and the World Health Organization (WHO). To avoid infection and spoiling, donated food must be handled, stored, and transported properly.**Invalid source specified.**

3.3.8.2 Health Rules:

Adhere to health rules concerning the handling and distribution of food, including standards of hygiene for those who are volunteering or engaged in the gathering, preparing, and distributing of food. Cultural The sensitivity is When choosing and allocating donated food items, consider Nepalese customs and dietary preferences. Consider local eating customs as well as religious dietary requirements and preferences.

3.3.8.3 Working with Government Agencies:

Organize with government organizations in charge of Nepal's disaster assistance and food security initiatives. Form alliances to guarantee the effective delivery of food

donations to places in need, particularly in times of emergency like natural disasters or humanitarian crises.

3.3.8.4 Identification & Reporting:

Keep thorough records of all food donations that are received and distributed, including information on the quantity, the helpfulness, and the recipients of the donations. Provide a monthly update outlining the work your organization is doing and the results it has achieved for the public, government agencies, and donors, among other stakeholders.

3.3.8.5 Community Involvement:

Get to know your neighbourhood's needs and preferences when it comes to food aid programs. To ensure the relevance and efficacy of food donation projects, involve members of the community in their development and execution.

3.3.8.6 Sustainability:

To guarantee long-term impact and efficacy, develop sustainable practices for food donation initiatives. This could entail encouraging the production and sourcing of food locally and minimizing food waste using careful planning and administration while implementing systems up and running to address the root causes of Nepal's food insecurity.

3.3.9 Non-Functional Requirement

The primary objective of this SRS document is to define the software requirements for the food donation system. The system aims to facilitate the process of donating surplus food to those in need. In this section, specific design and implementation information will be provided. Non-functional requirements, as the title suggests, focus on system characteristics beyond its core functionalities. Some of the non-functional requirements for the food donation system are as follows:

3.9.1 Usability

The system should ensure ease of use and effectiveness to satisfy end users. Each interface should include a help and information section to guide users in interacting with the system. The system's documentation should provide comprehensive guidance for remote management.

3.9.2 Security

The system must prioritize security to protect against unauthorized access. Users are required to provide a login and password, with passwords being at least eight characters long. Only authorized users with active credentials should be able to access the system's secured pages. The system should maintain a high level of security and integrity for the data it holds.

3.9.3 Performance

The system should respond to user instructions within 10 seconds at most. It should operate quickly and efficiently, with response times typically within milliseconds for most tasks. The system is expected to deliver optimal performance and complete tasks promptly.

3.9.4 Error Handling

The system should minimize errors and provide clear error correction instructions to help users recover from them. User inputs must be thoroughly validated, and the system should handle errors automatically, with minimal impact on user experience.

3.9.5 Ease of Use

The system should feature a simplified yet high-quality user interface to accommodate users of varying levels of understanding. It should require minimal training for users to navigate and utilize its functionalities effectively.

3.9.6 Maintainability

The system should allow users to reset all optional and stored user variables to default settings easily. It should be designed for easy maintenance, with automated maintenance processes utilizing stored user data to provide relevant information.

3.9.7 Portability

The software should be easily adaptable to different networks or operating systems, facilitating its distribution and deployment across various environments.

This section outlines the non-functional requirements that are essential for ensuring the effectiveness, security, performance, availability, error handling, usability, maintainability, and portability of the food donation system.

3.4. Design (Concerning methodology what are the core design techniques.

For eg: If the methodology is USDP, UML diagrams are a must)

3.4.2. Use Case

A use case diagram is a visual representation that illustrates the interactions between actors and a system. It showcases various use cases of a system and how different actors interact with those use cases. The most important instrument for system design is the use case diagram, which shows users' interactions with a system visually. It acts as a road map for understanding a system's functional needs from the point of view of its users, enabling stakeholder communication and leading the development process. (GeekForGeek, Unified Modeling Language (UML), 2024)

The system has three types of end users where admin, donor and volunteer. The use case represents which user can do the system. Where the system can show the user connection and work in the system flow.

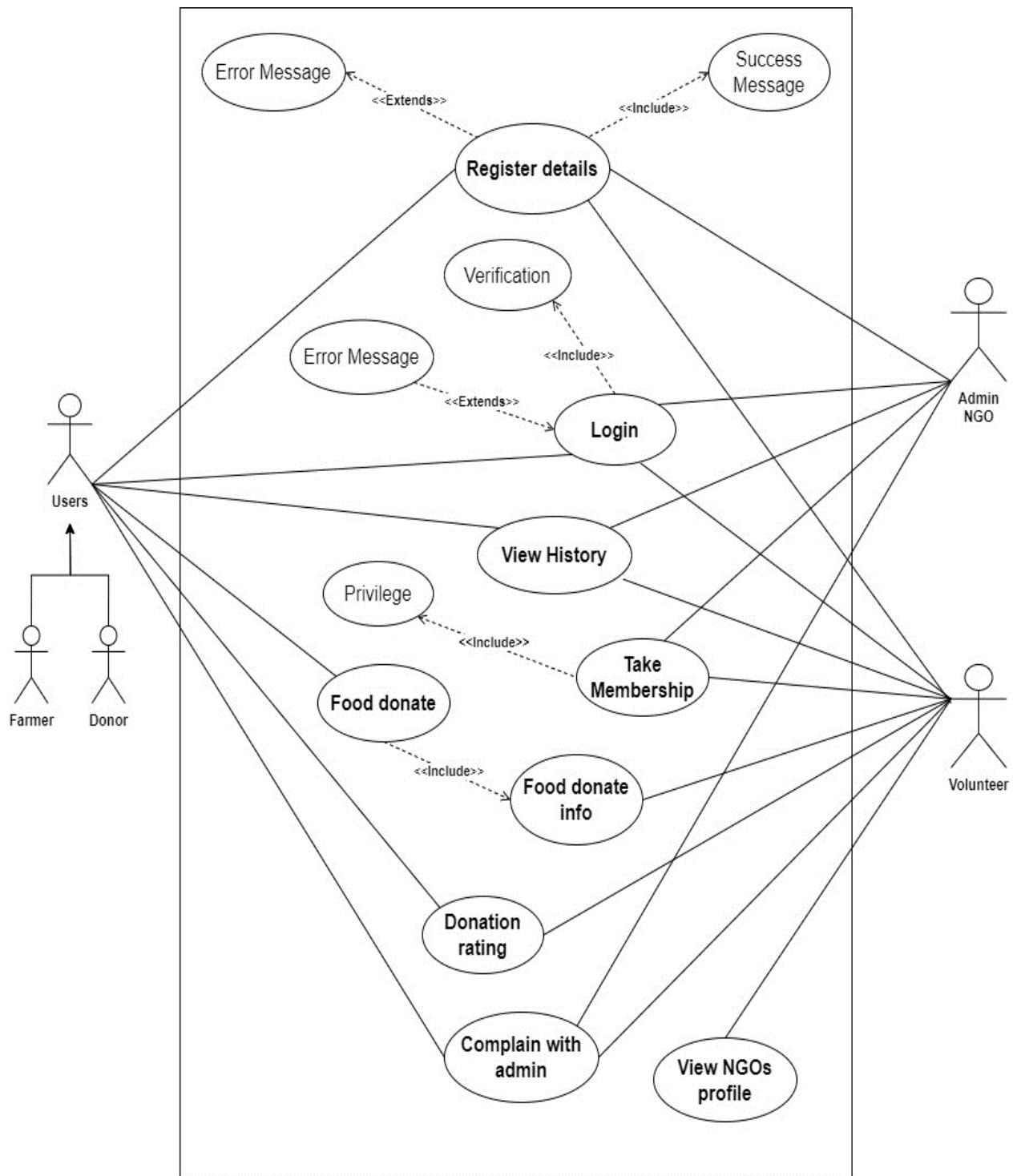


Figure 35: Use Case Diagram

Appendix C: Use-Case Diagram

3.2.1. High-Level Use Case

The system includes an extensive collection of use cases, many of which are high-level use cases that facilitate system design and development. Each high-level use case focuses on providing a brief explanation of each procedure as it develops and as a result, would be recognized by the customer.

The high-level use case is given below:

3.2.1.1. Register user.

Use Case:	Register Details
Actors:	Donor, Volunteer, NGO, Farmer
Descriptions:	All users can input their respective details into the system. Upon submission, the system automatically registers the provided information, ensuring seamless integration and efficient data management. If the register details are correct show the success message otherwise display the error message.

Table 9: Register user high-level use case

3.2.1.2. Take Membership

Use Case:	Take Membership
Actors:	Volunteer, NGO
Descriptions:	A new volunteer provides the personal details, and his/her details are registered with the system. The NGO provide the membership, and then volunteers take the new membership.

Table 10: Take membership user high-level use case

Use Case:	Privilege
Actors:	Volunteer
Descriptions:	After taking a new member, a new volunteer gets the privilege of the food donation system.

3.2.1.3. Login System

Use Case:	Login
Actors:	Donor, Volunteer, Farmer, NGO
Descriptions:	After registering details in the system, all the users can provide valid details and log in to the system. Then successfully log in to the system. If the login details are valid display the success message otherwise error message.

Table 11: Login user high-level use case

3.2.1.4. Food Donate

Use Case:	Food Donate
Actors:	Donor, Farmer, Volunteer
Descriptions:	The Donor or Farmer can donate the proper food information and details with location. The system can show the donation food details in the history after posting the donated food. All the volunteers can get the donation information (Notification).

Table 12: Food donation high-level use case

3.2.1.5. View Donation info

Use Case:	Donation Info
Actors:	Volunteer
Descriptions:	After receiving the donation info, the volunteer can view the donation details if it is possible or not possible to distribute.

Table 13: View donation info high-level use case

3.2.1.6. Donation Rating

Use Case:	Donation Rating
Actors:	Volunteer, Donor
Descriptions:	After the food is completely donated to some people the volunteer can give the donation rating to the donor with food distributed information.

Table 14: Donation rating high-level use case

3.2.1.7. View History

Use Case:	View History
Actors:	Donor, Volunteer, NGO, Farmer
Descriptions:	All the users can view the history of food donation rate, where to donate or more details.

Table 15: View the history of high-level use case

3.2.1.8. View NGO profile

Use Case:	View NGO profile
Actors:	Volunteer, NGO
Descriptions:	The volunteer can view the NGO profile details where some information gets more details.

Table 16: View profile high-level use case

3.2.1.9. Complain with Administration

Use Case:	Complain with Admin
Actors:	Donor, Volunteer, Admin
Descriptions:	After contact with donors and volunteers the donors cannot be provided or donate food and the receiver cannot come to receive the donation food, they can complain to the admin

Table 17: Complaint with high-level use case

[\(Expanded use case for appendix\)](#)

3.3. Sequence Diagram

3.3.1. View profile sequence diagram

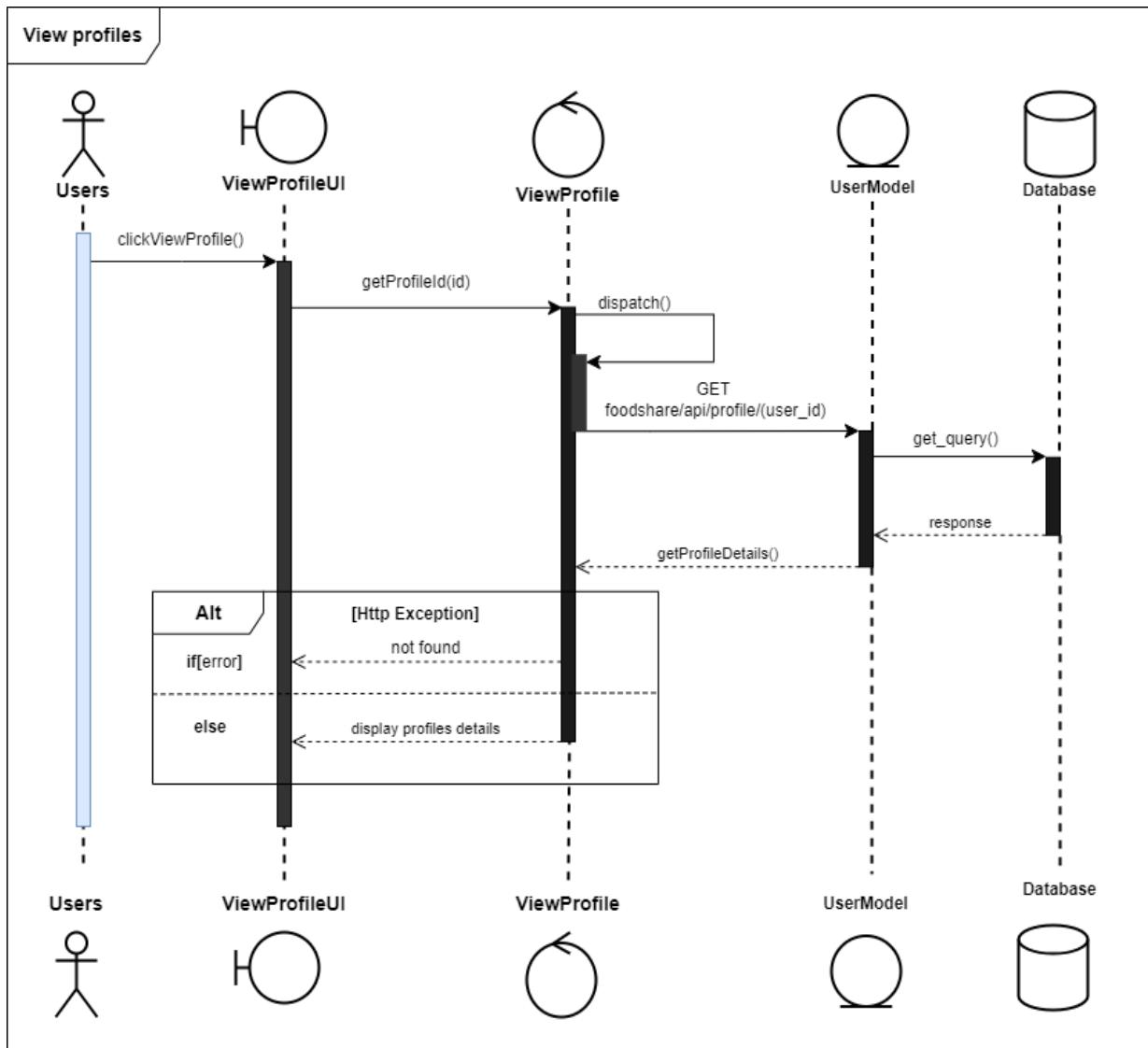


Figure 36: View profile sequence diagram

3.3.2. View history sequence diagram

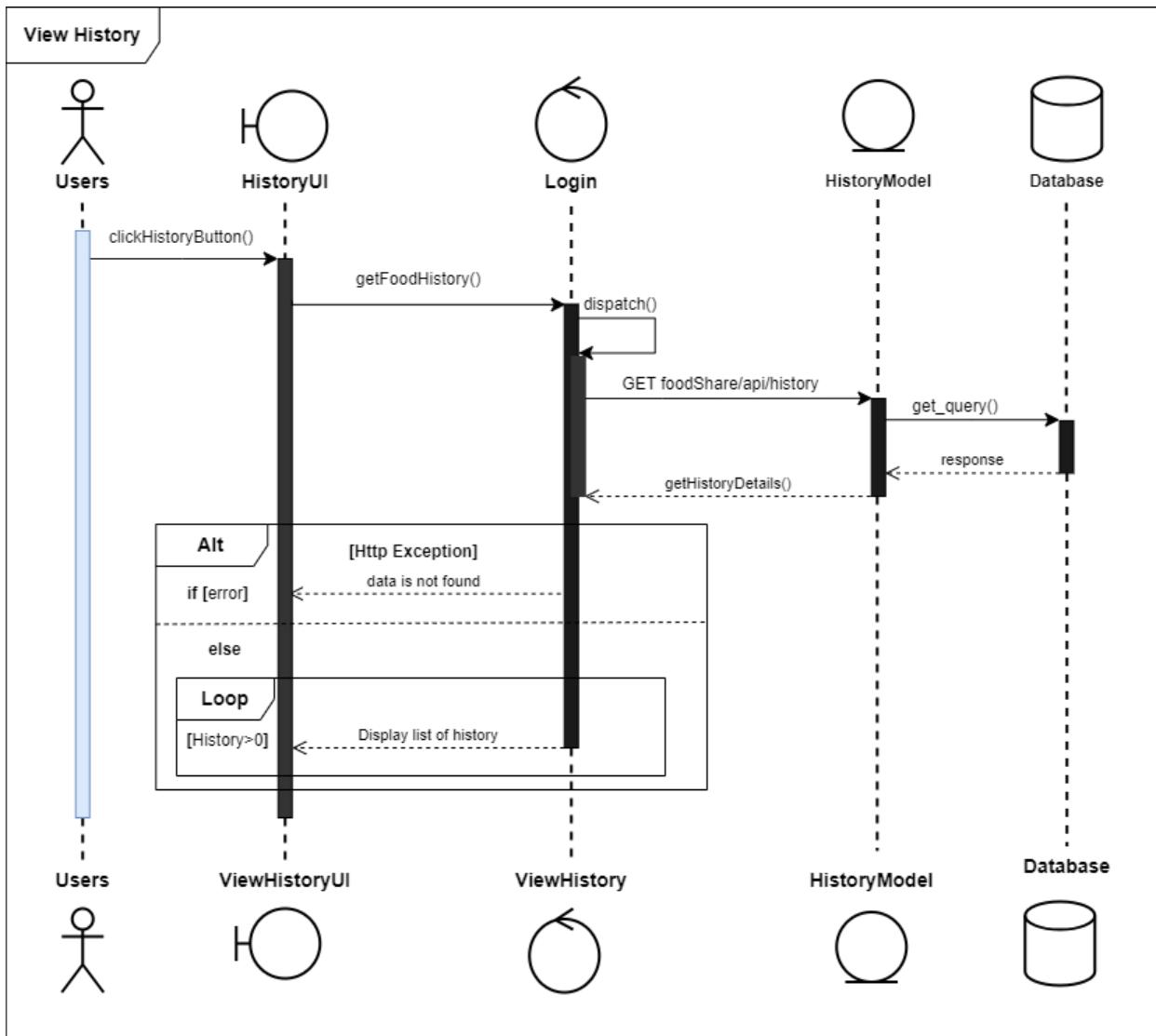


Figure 37: View the history sequence diagram

3.3.3. Post food donation sequence diagram

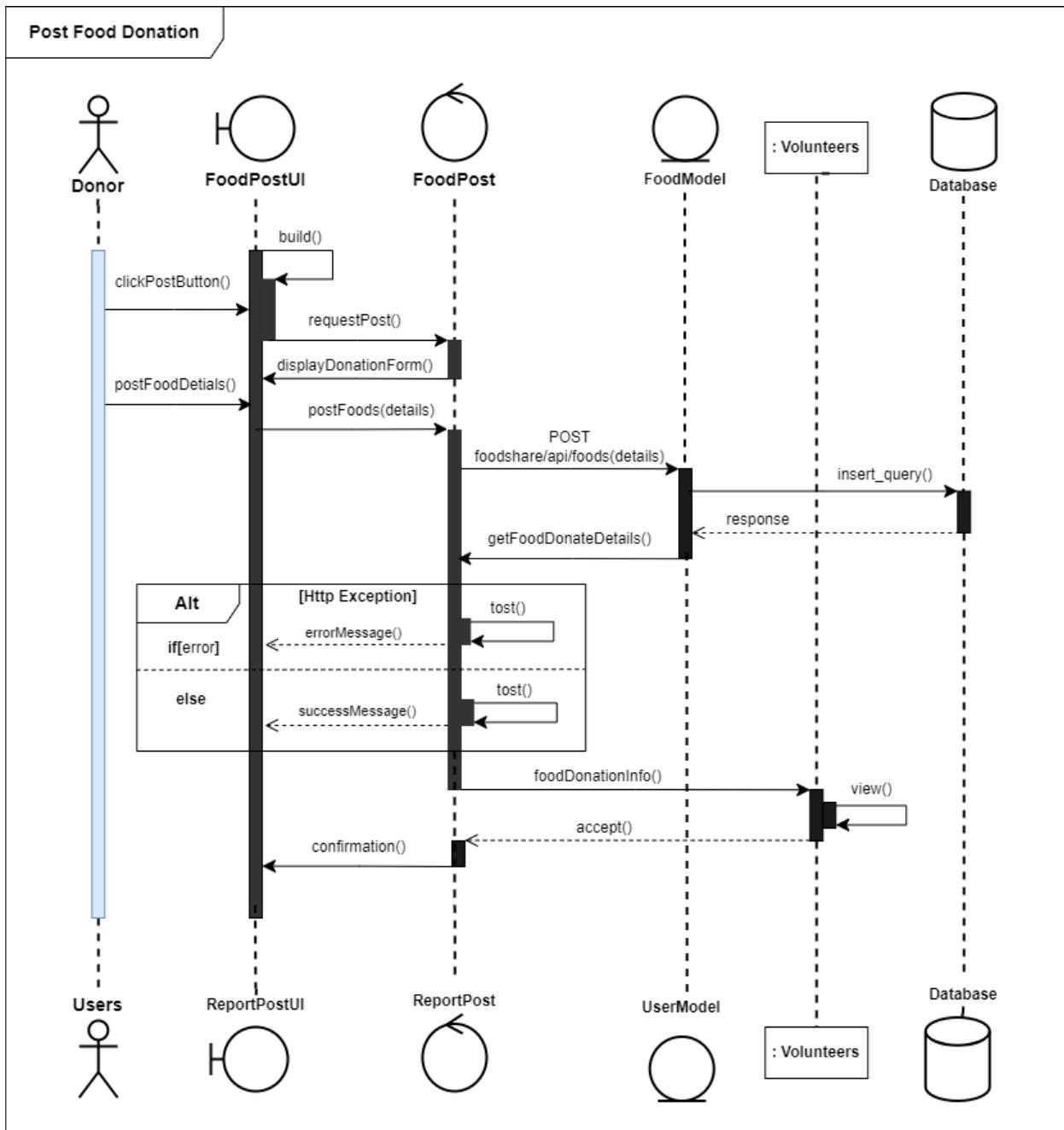


Figure 38: Food Post for donation sequence diagram

3.3.4. Donation rating sequence diagram

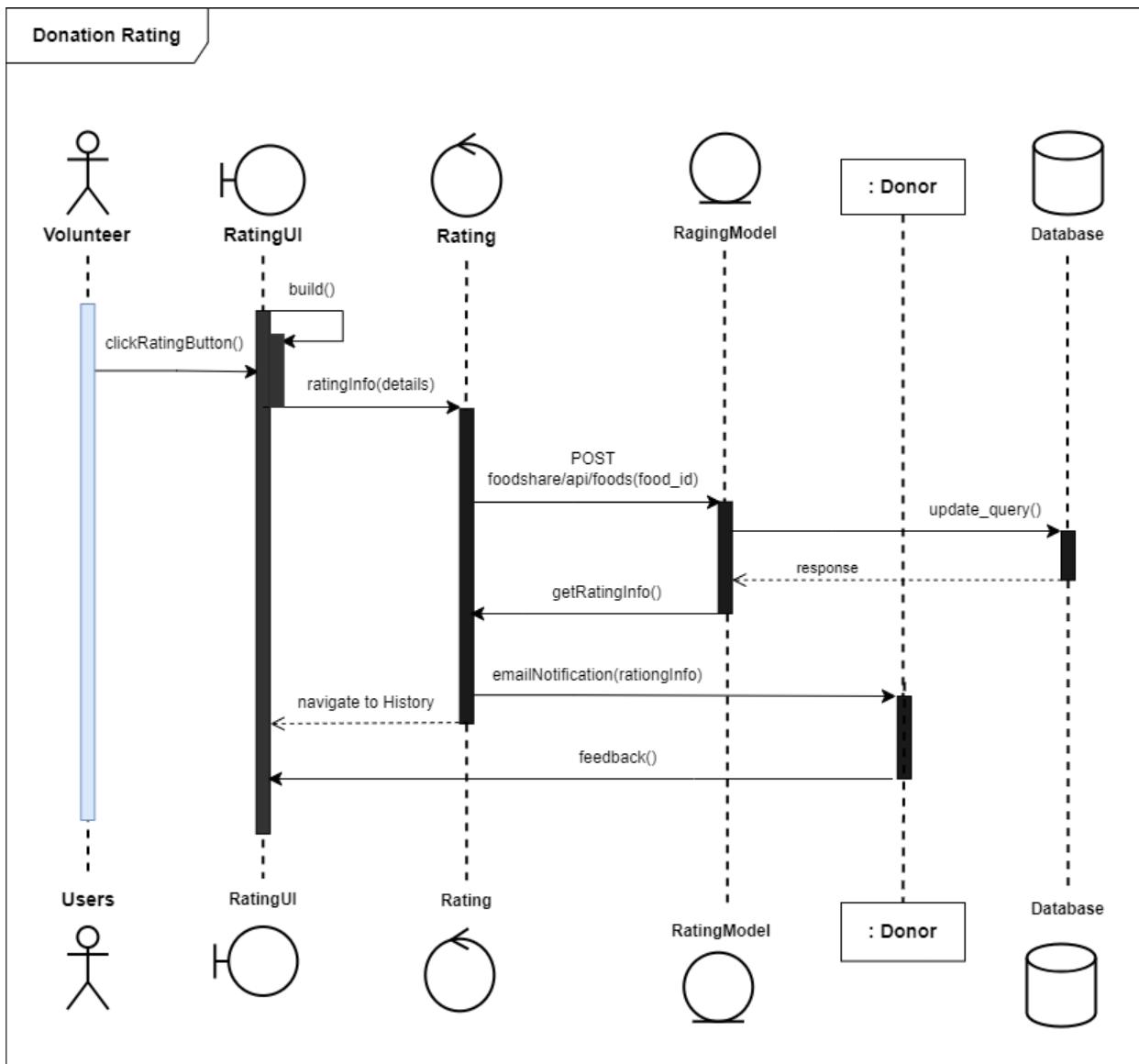


Figure 39: Donation rating sequence diagram

3.3.5. Complaint with admin

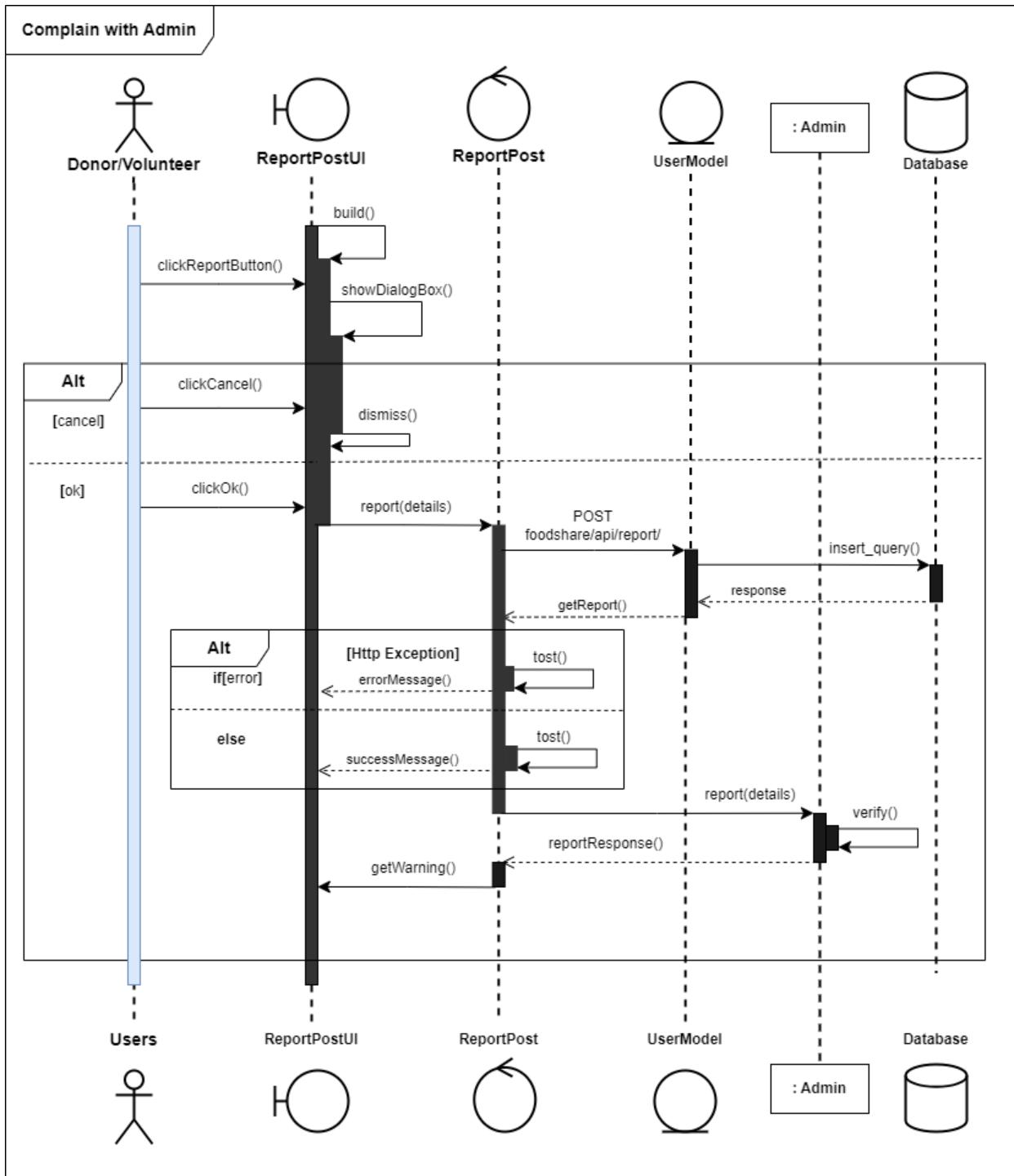


Figure 40: Complaint with admin sequence diagram

[\(Sequence Diagram for appendix\)](#)

3.4. Context Diagram

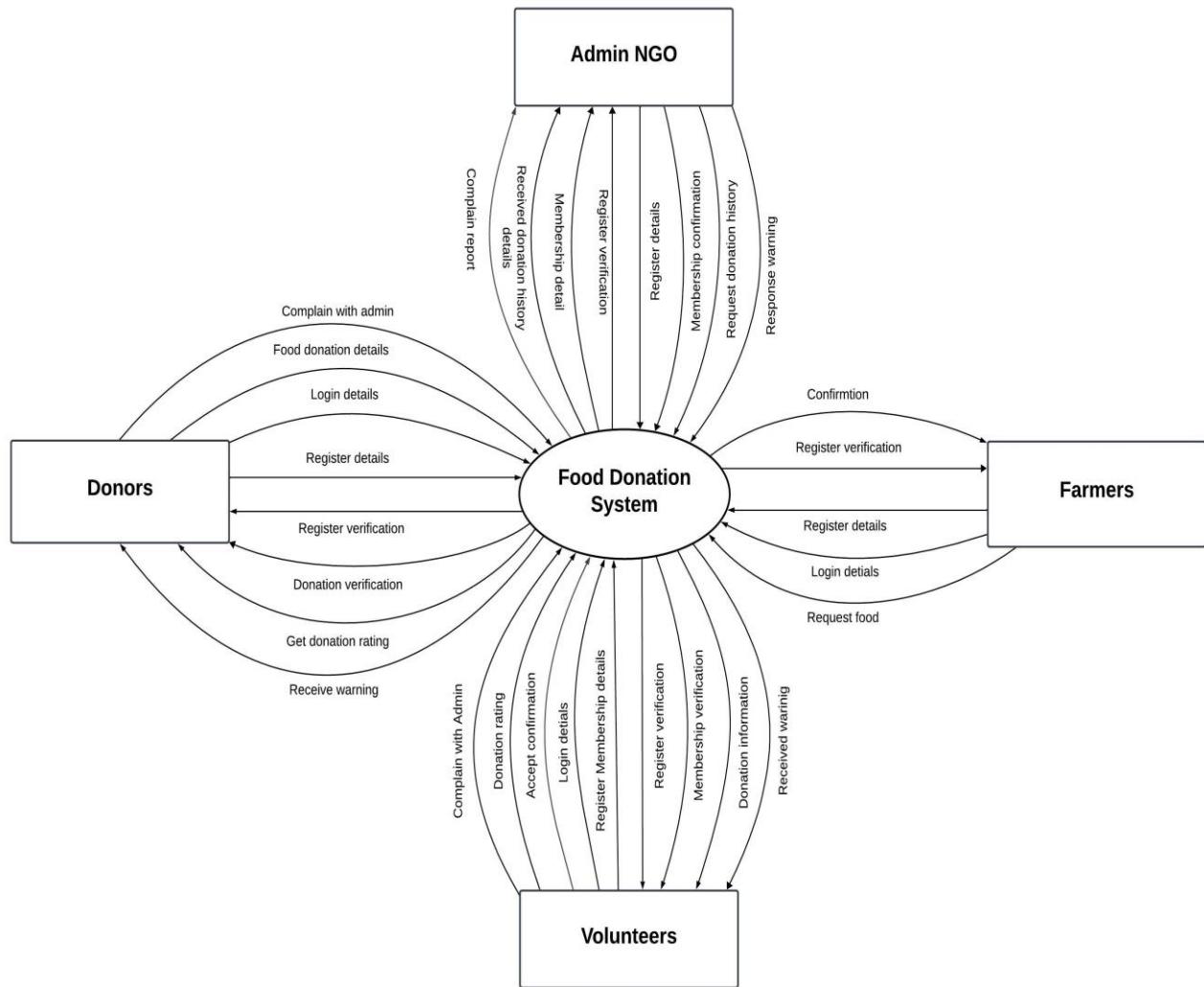


Figure 41: Context diagram

3.5. Data Flow Diagram Level -1

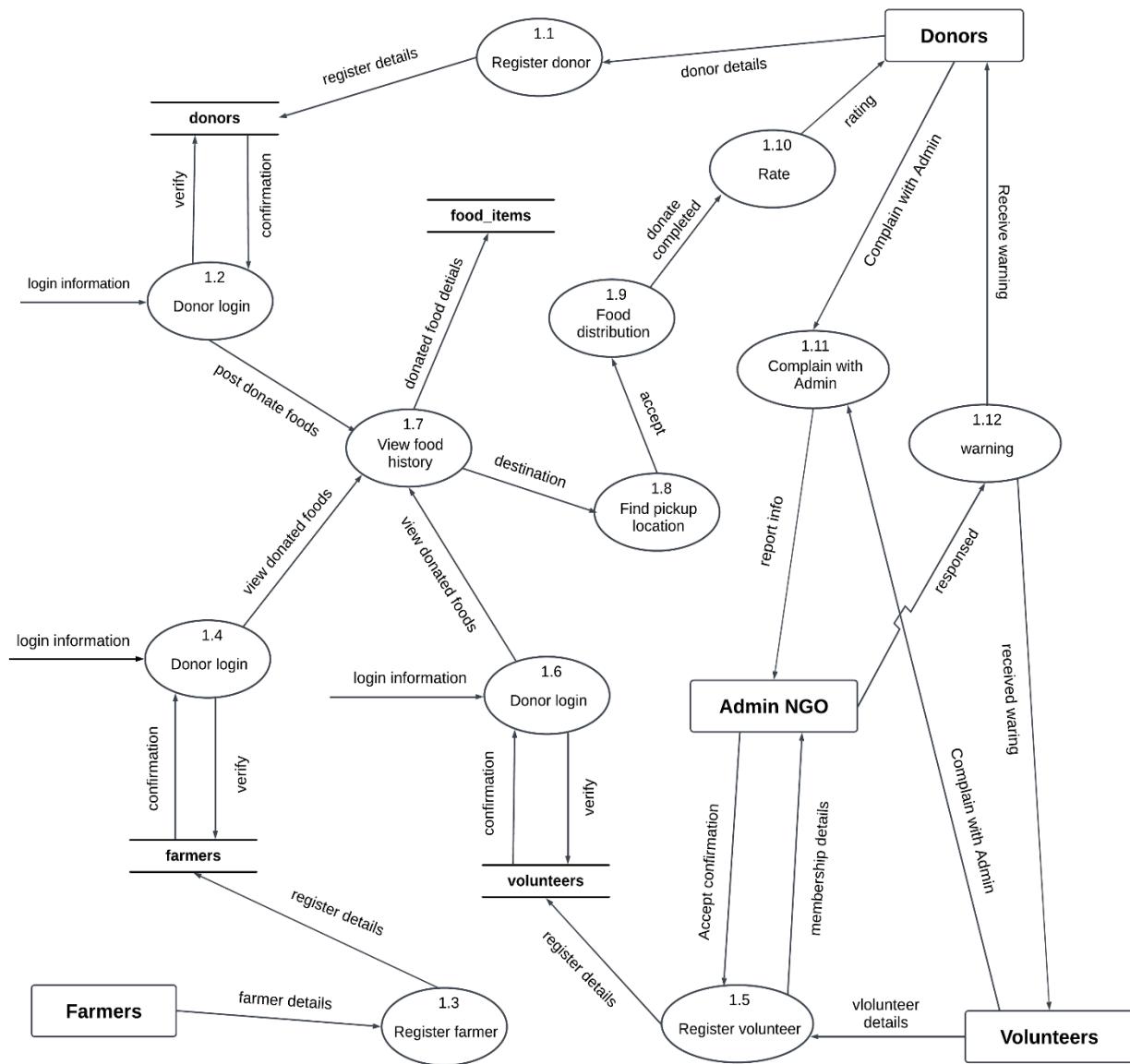


Figure 42: DFD Level-1

3.6. Activity Diagram

3.6.1. Take Membership Activity

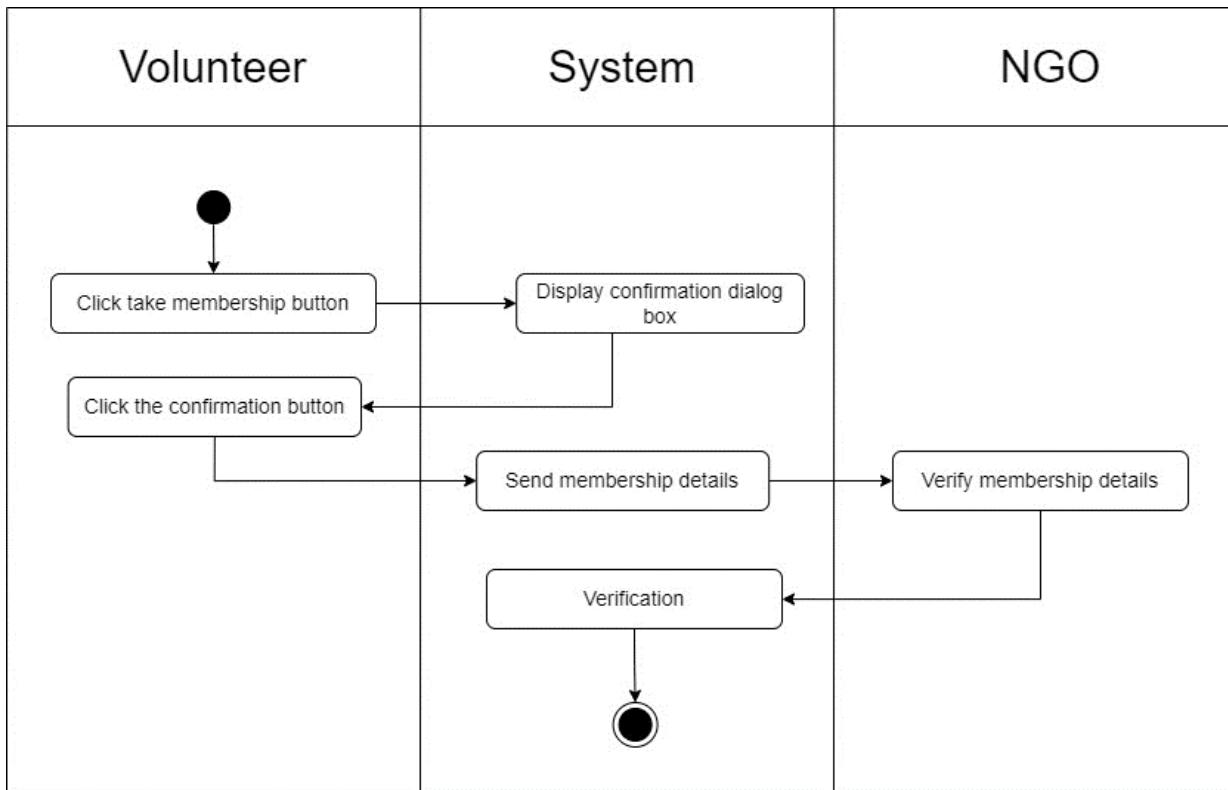


Figure 43: Take the membership activity diagram

3.6.2. Donation Activity

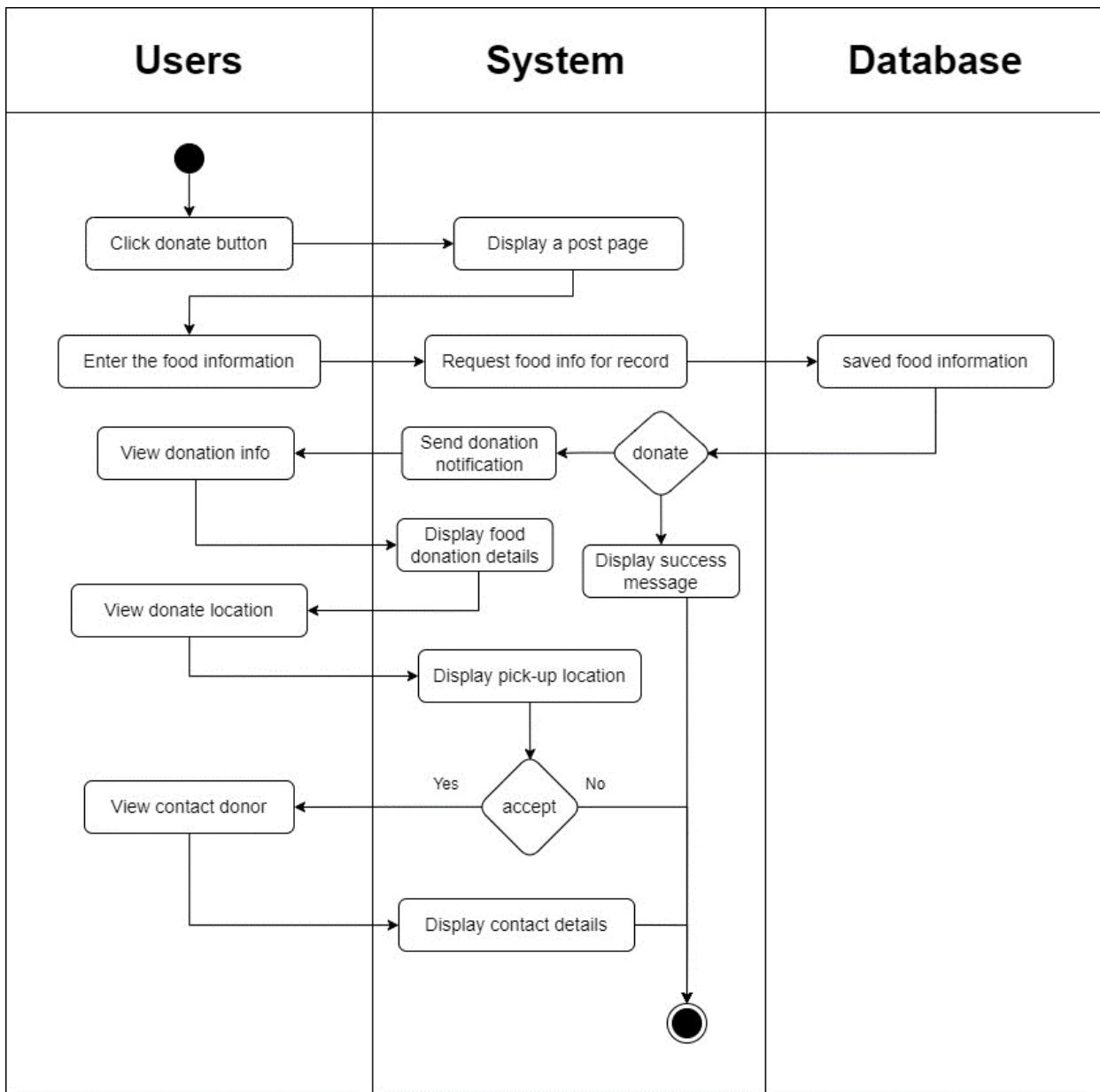


Figure 44: Food donation system activity diagram

3.6.3. View History Activity

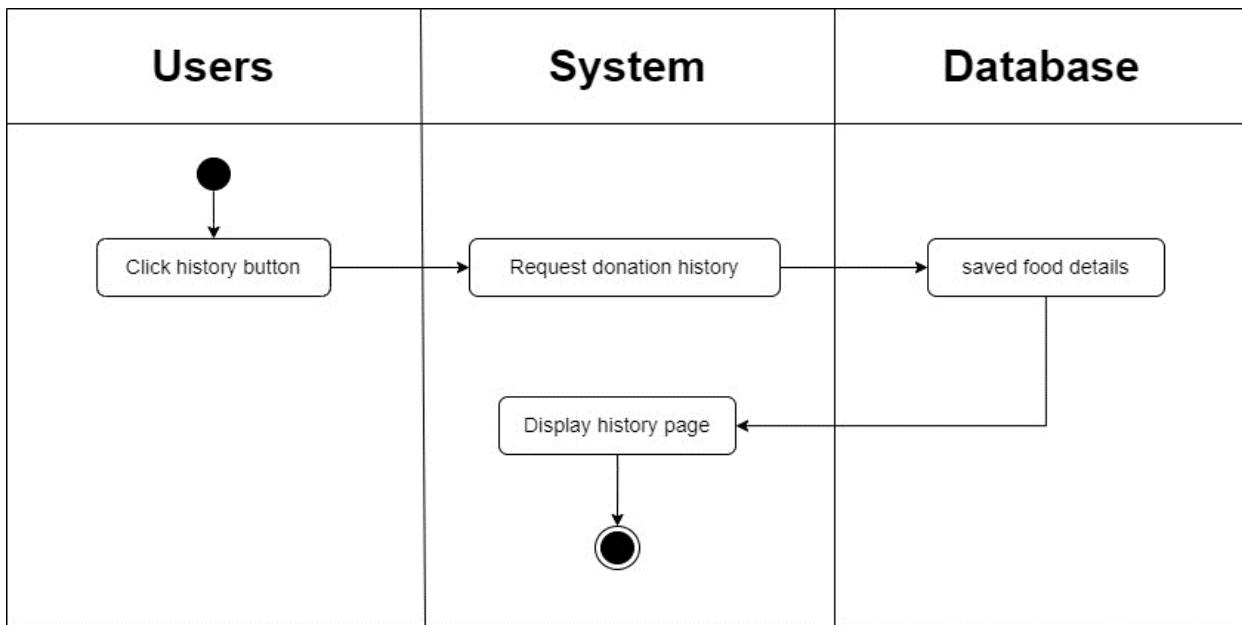


Figure 45: View the history system activity diagram.

3.6.4. Donation Rating Activity

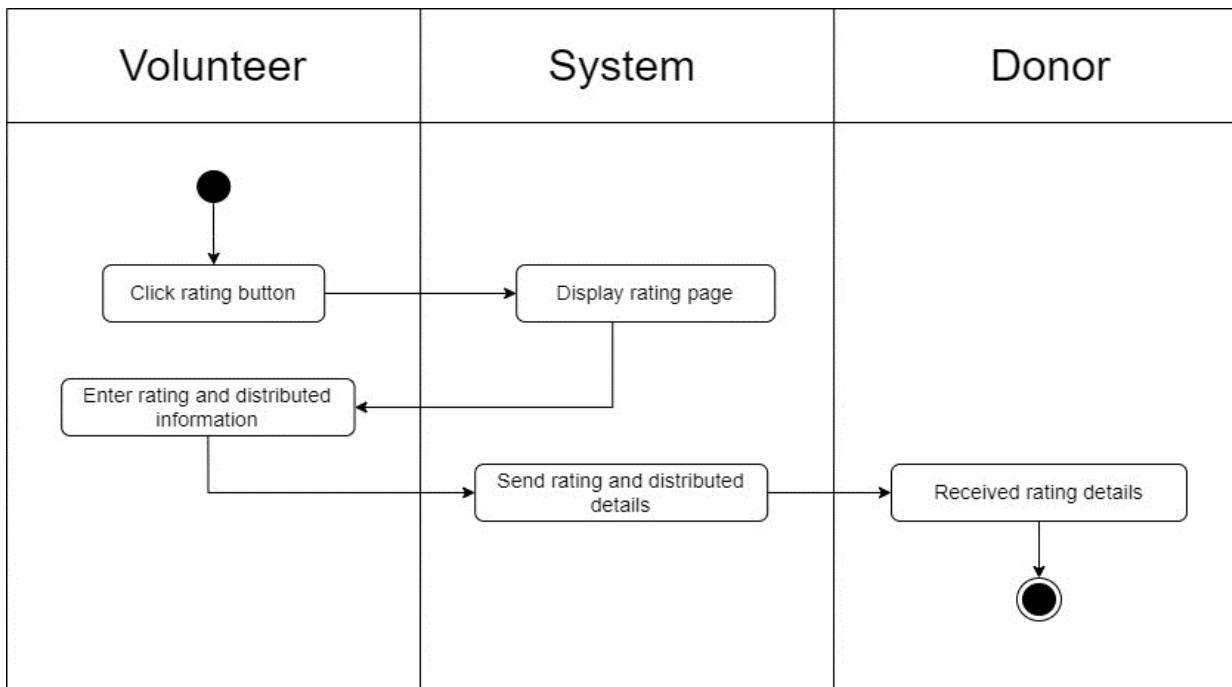


Figure 46: Rating donation system activity diagram

3.6.5. Complaint with Admin Activity

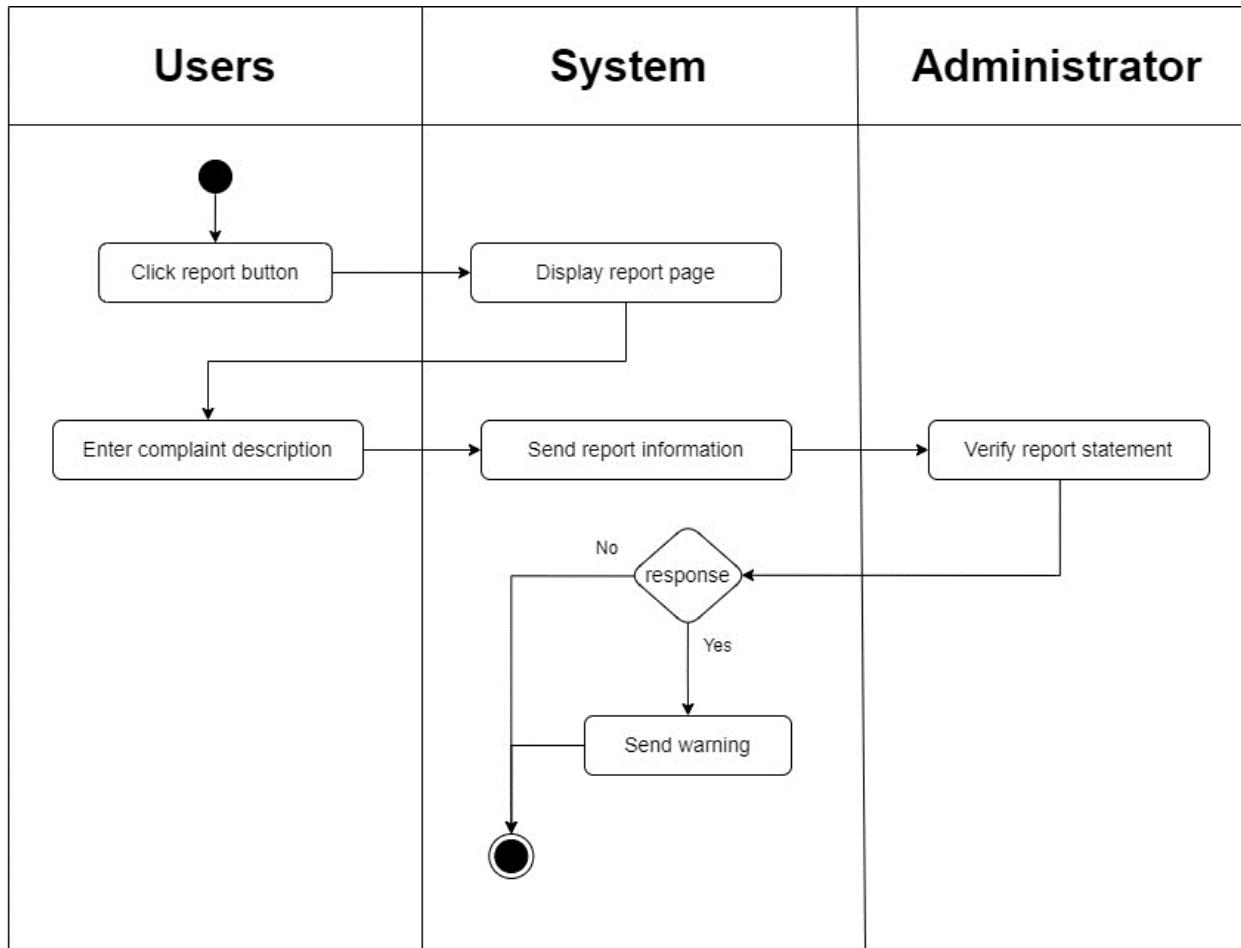


Figure 47: Complaint with admin system activity diagram

[\(Activity diagram for appendix\)](#)

3.7. Wireframes

3.7.1 Mobile UI

3.7.1.1 Walkthrough Screen

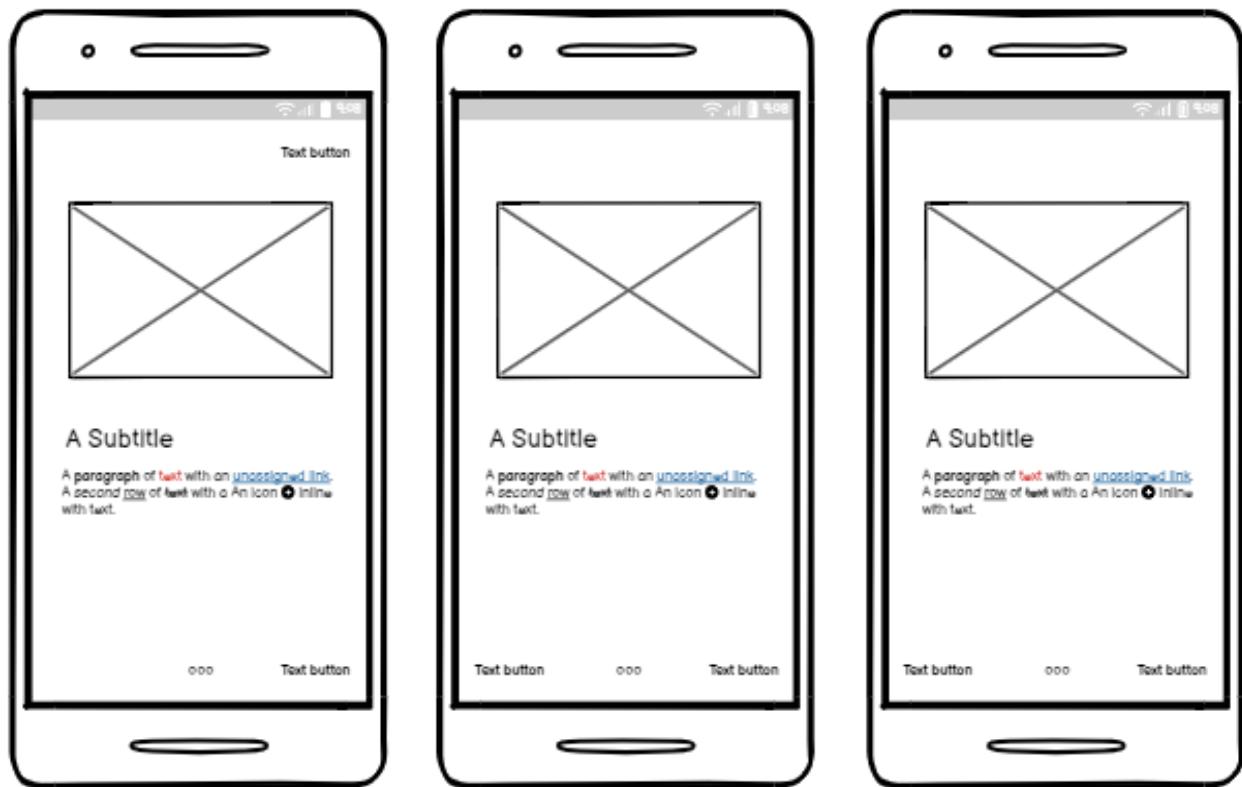


Figure 48: Walkthrough screen mobile UI

3.7.1.2 Welcome Screen

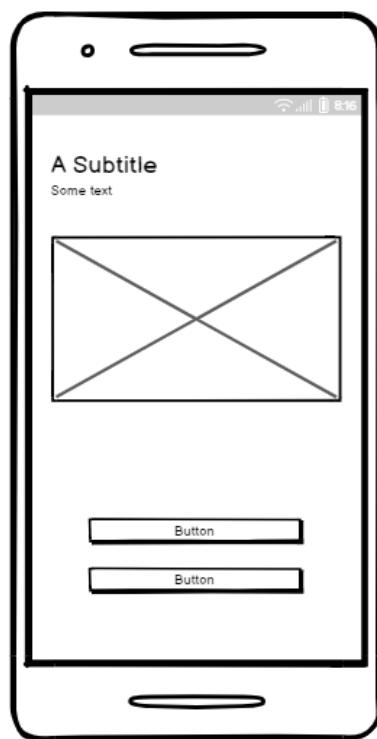


Figure 49: Welcome screen mobile UI

3.7.1.3 Login Screen

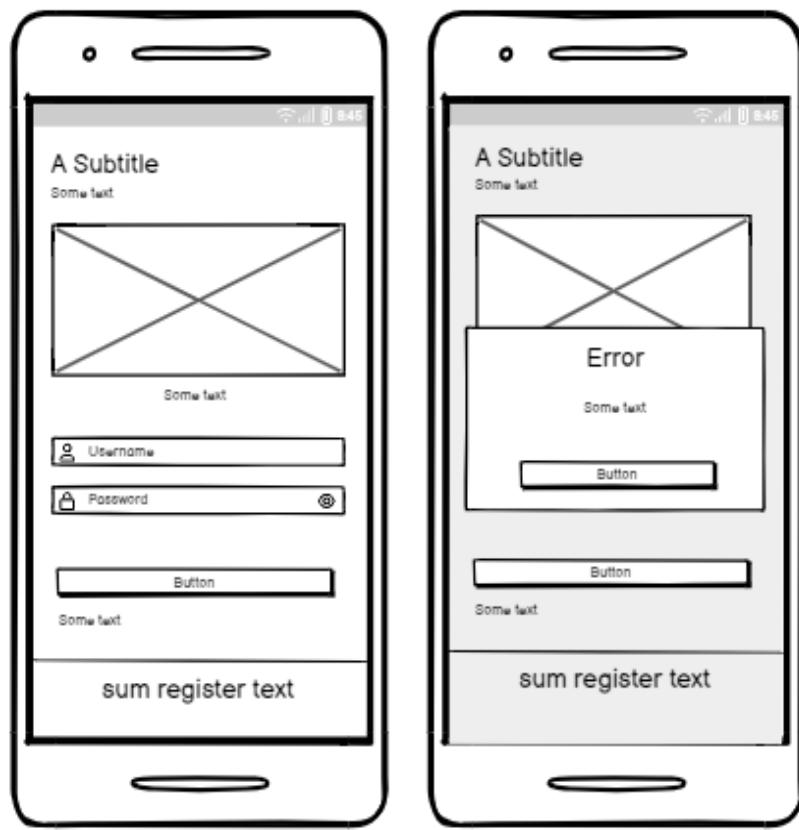


Figure 50: Login screen mobile UI

3.7.1.4 Register Screen

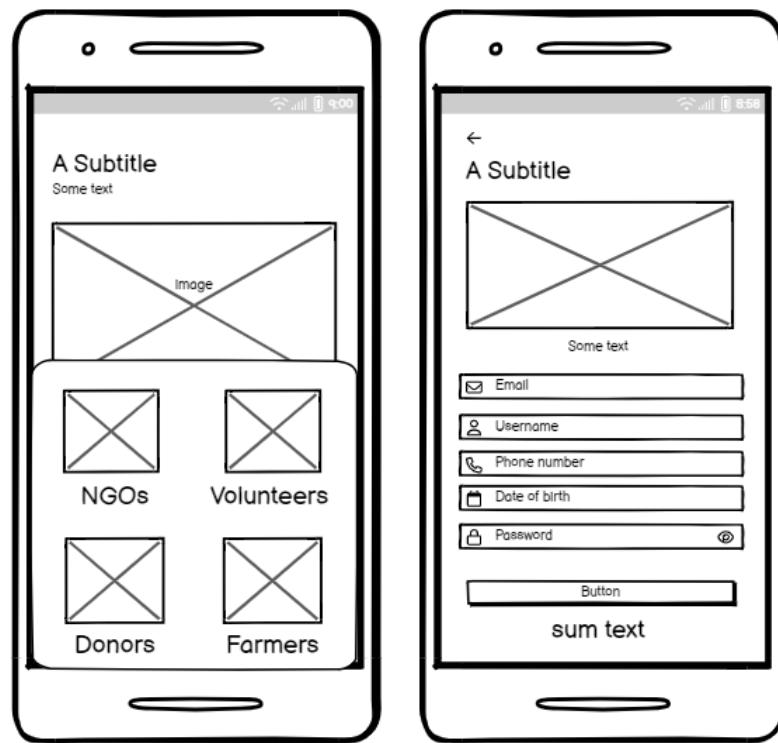


Figure 51: Register screen mobile UI

3.7.1.5 Forgot Password Screen

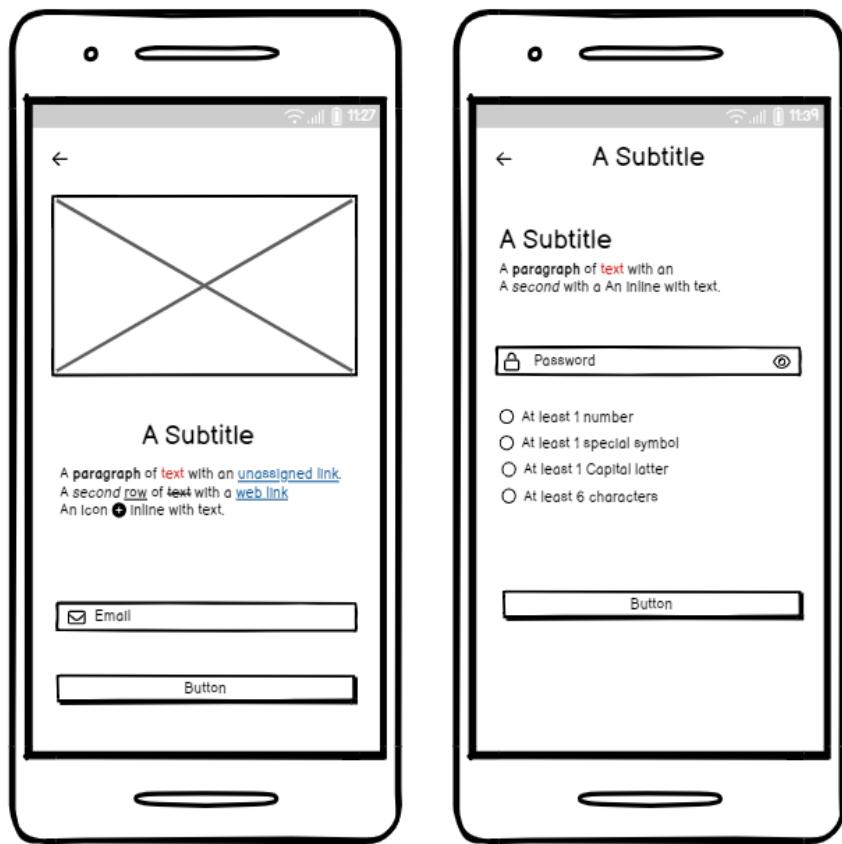


Figure 52: Forgot password screen mobile UI

3.7.1.6. Dashboard for Donors

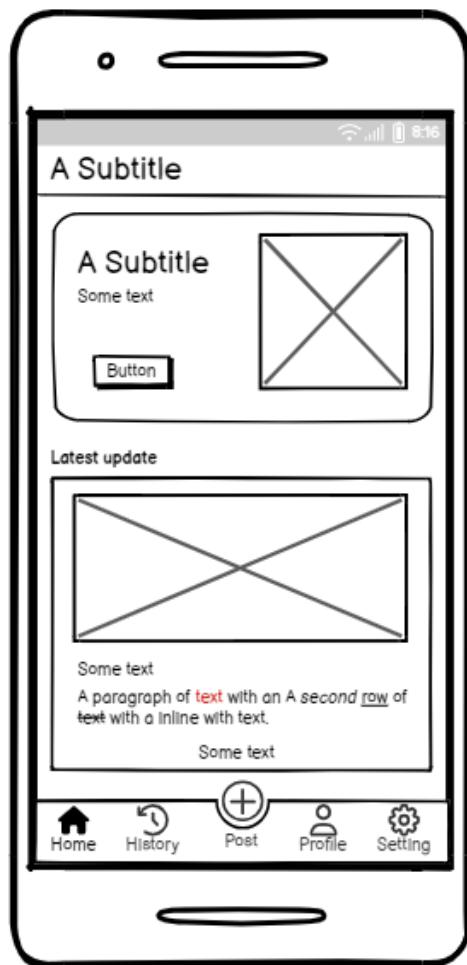


Figure 53: Donor Home screen mobile UI

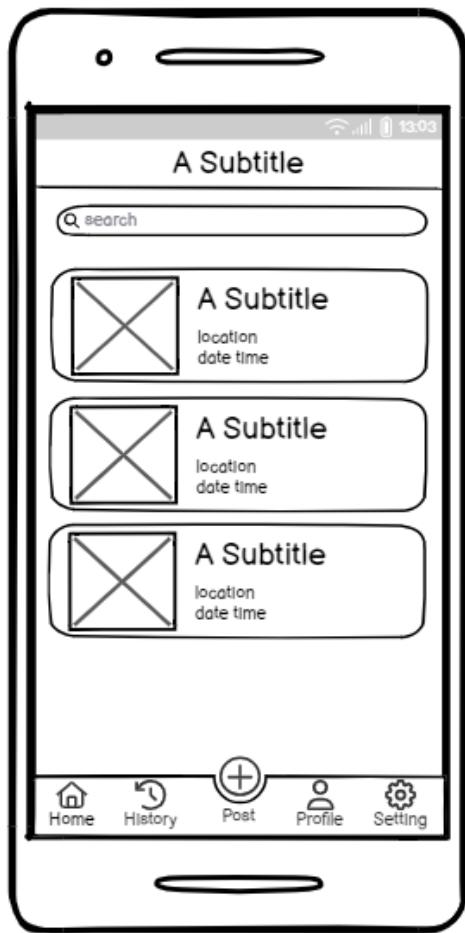


Figure 54: Donor History screen mobile UI

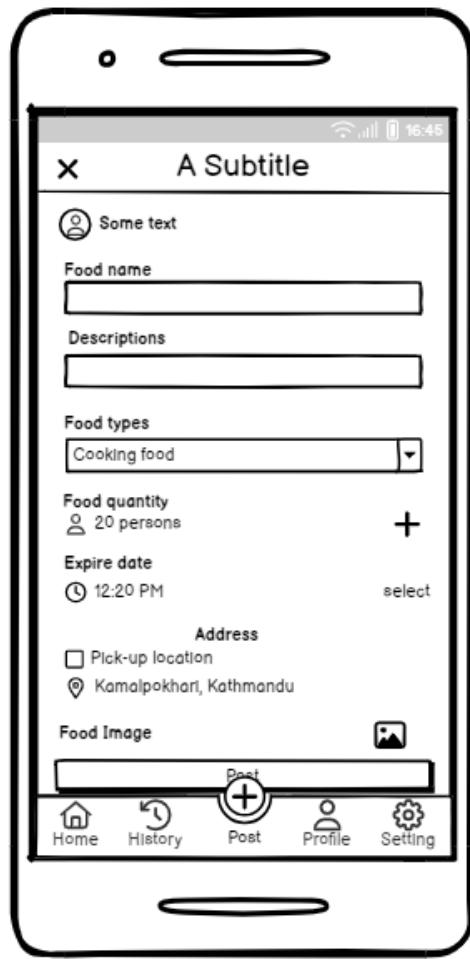


Figure 55: Donor Post/Donation screen mobile UI

3.7.1.7. Dashboard for Volunteer and Farmer

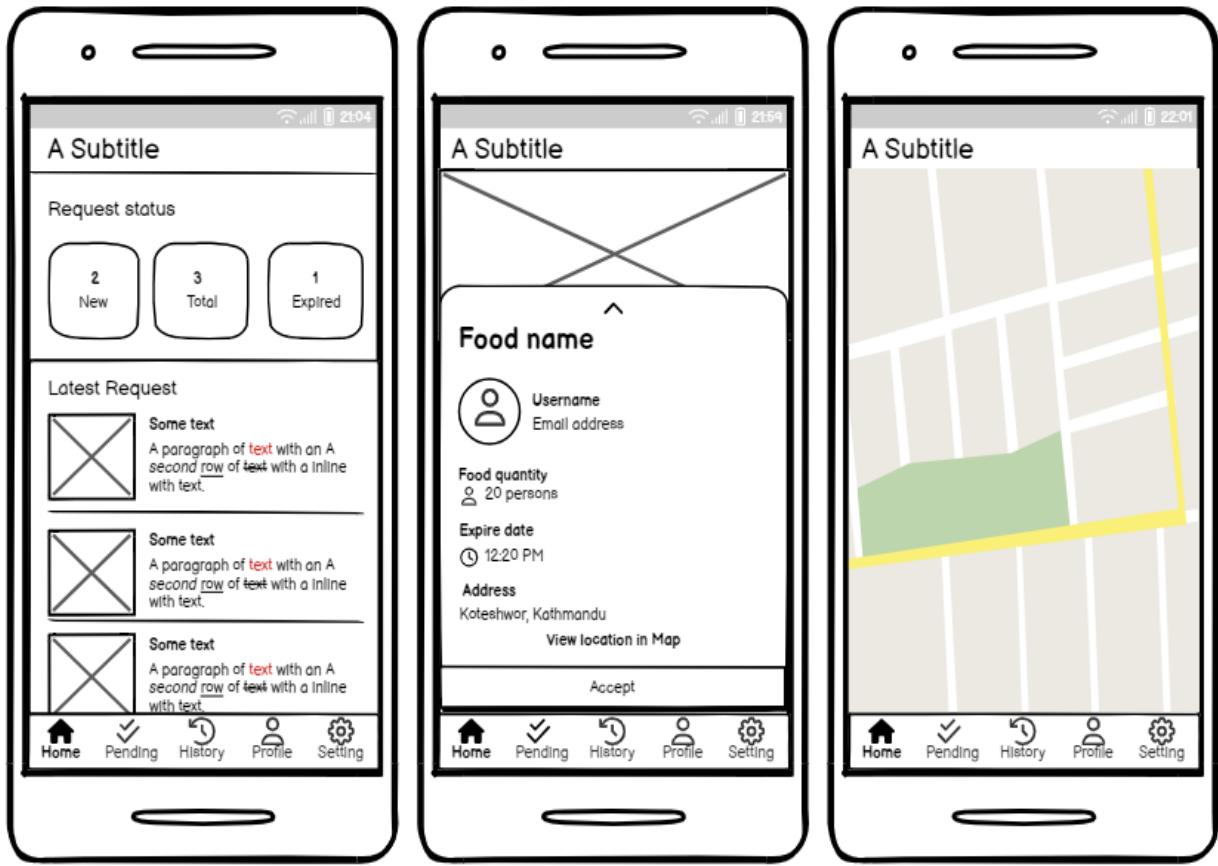


Figure 56: Volunteer Home screen UI

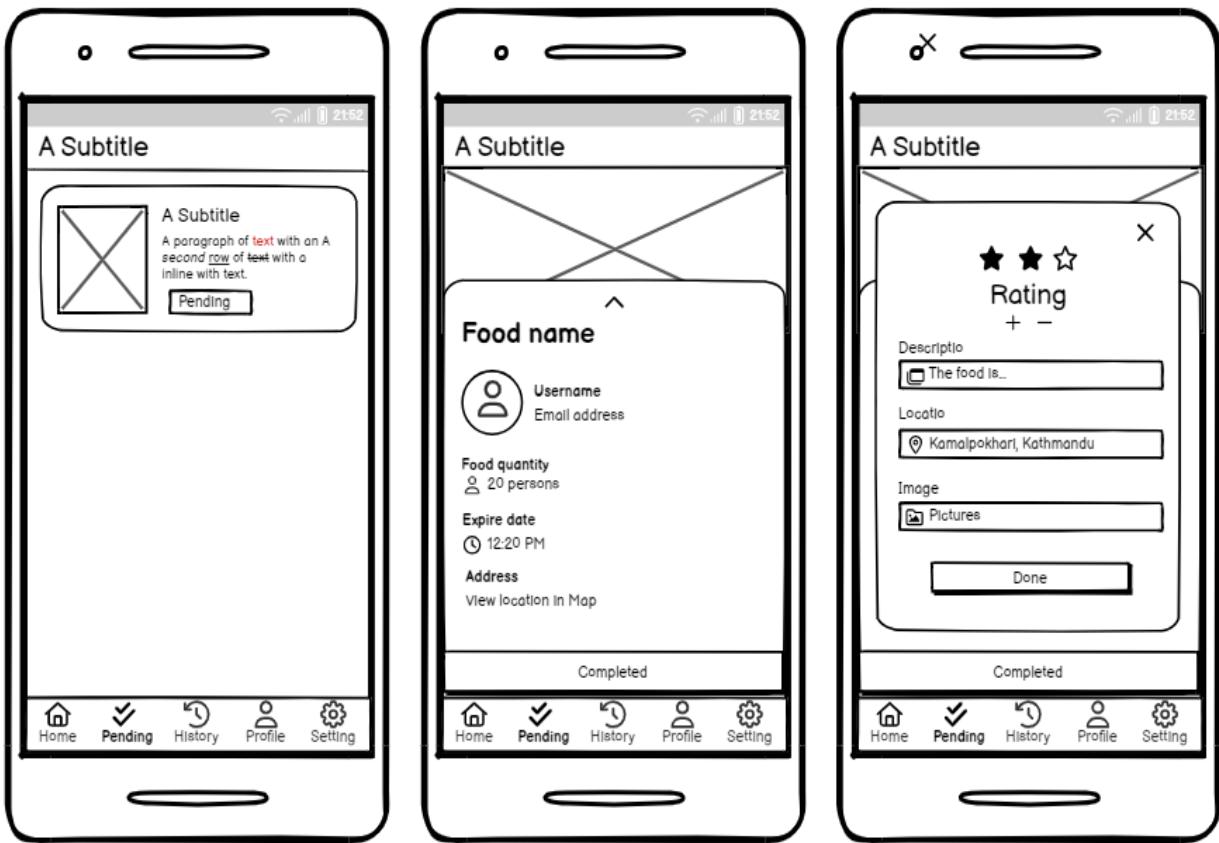


Figure 57: Volunteer Pending screen UI.

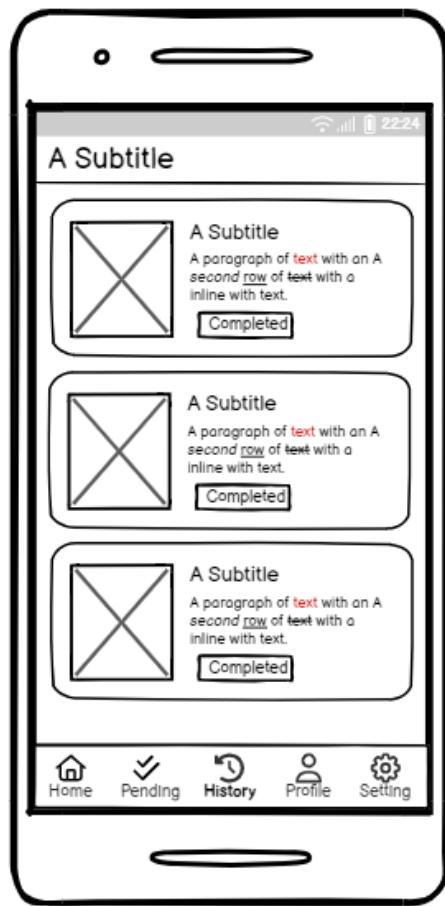


Figure 58: Volunteer History screen

3.7.1.8. NOG Dashboard

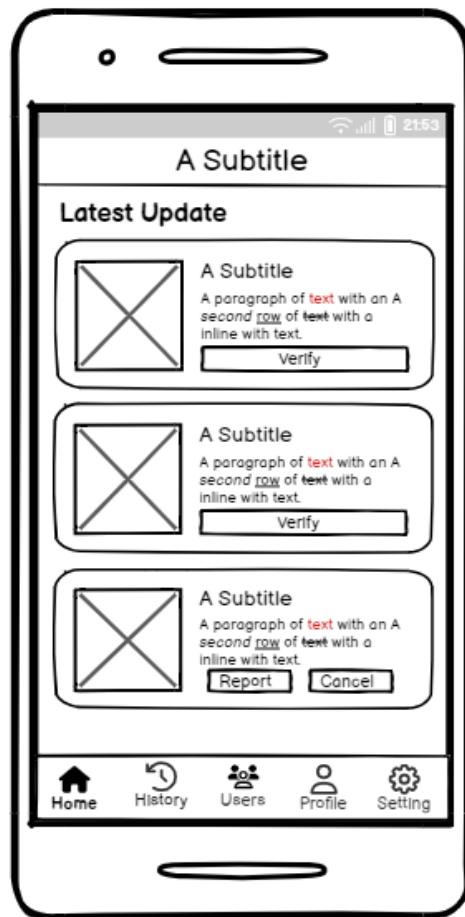


Figure 59: NGO Home screen UI

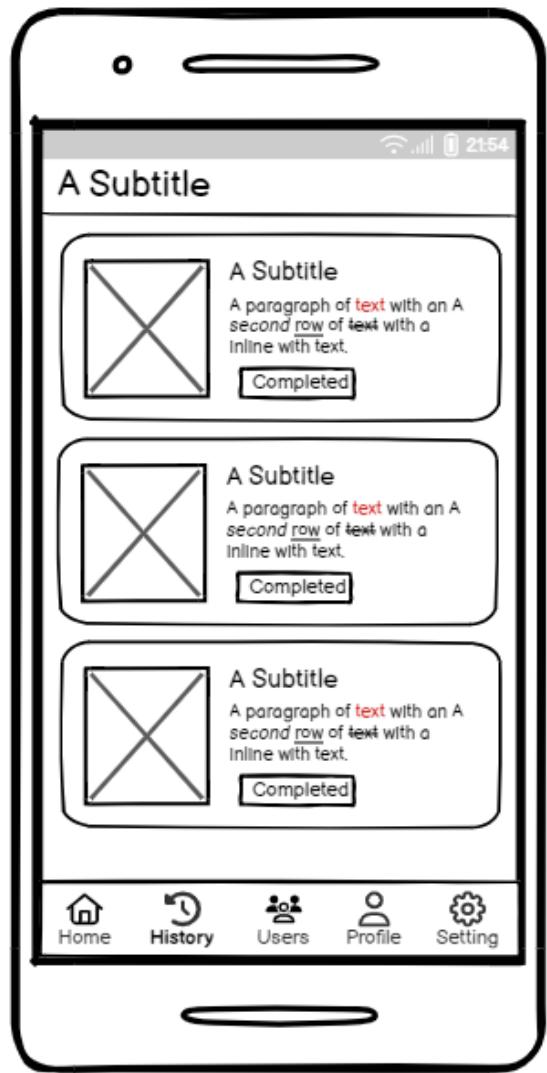
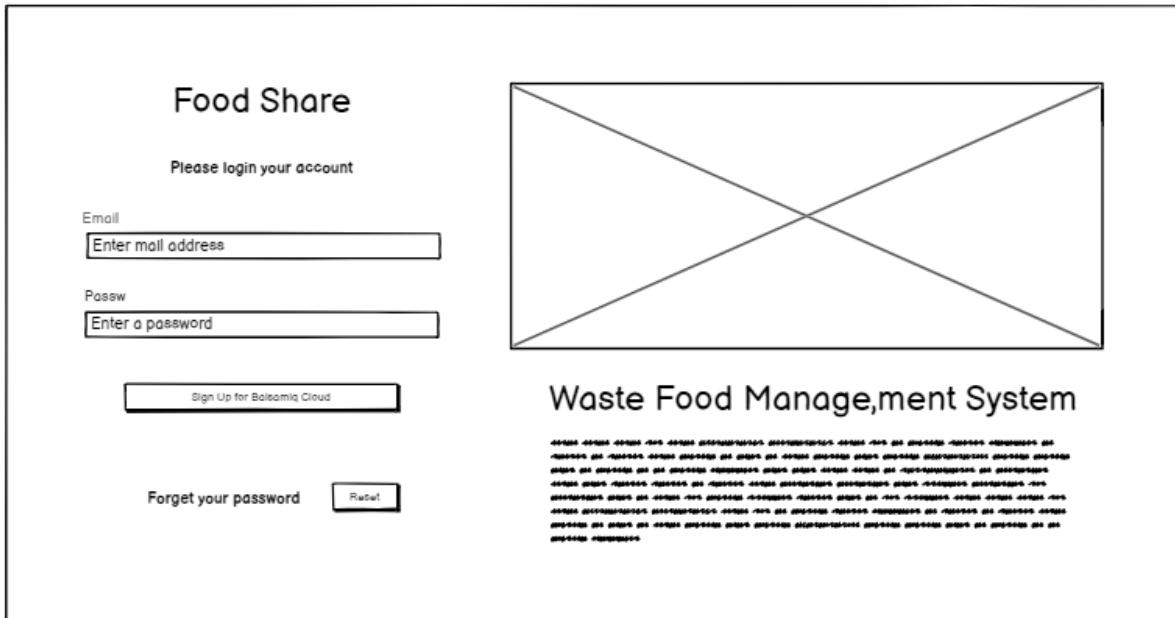


Figure 60: NGO History screen UI



Figure 61: NGO Setting Screen UI

3.7.2. Web UI for Admin



The image shows the Admin login page for the Waste Food Management System. The page has a header "Food Share" and a sub-header "Please login your account". It contains fields for "Email" (with placeholder "Enter mail address") and "Password" (with placeholder "Enter a password"). Below these are two buttons: "Sign Up for Baisamiq Cloud" and "Forgot your password". To the right of the login form is a large rectangular area with a large 'X' drawn through it, and the text "Waste Food Management System" below it.

Figure 62: Admin login page web UI



The image shows the Admin forgot password UI. It features a title "Rest Password" at the top. Below it are three input fields: "Email", "New Password", and "Confirm Password". At the bottom are two buttons: "Cancel" and "Confirm".

Figure 63: Admin forgot password UI.

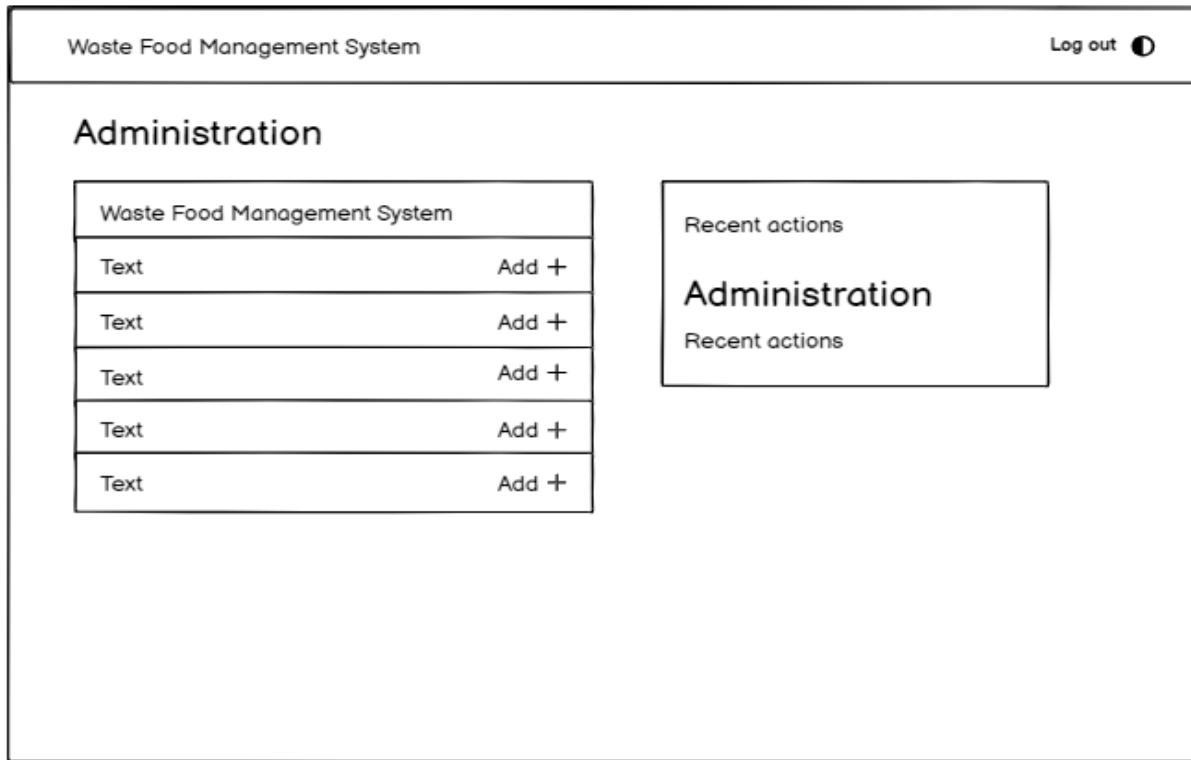


Figure 64: Admin dashboard UI

[\(Wireframe UI for appendix\)](#)

3.8. ER-Diagram

An Entity Relationship (ER) Diagram is a diagram that shows how "entities" like as persons, items, or ideas interact inside a system. ER Diagrams are commonly used to create or debug relational databases in software engineering, business-oriented systems, learning, and research. Also known as ERDs or ER Models, they demonstrate the interrelationships of entities, relationships, and their properties using a predefined set of symbols such as rectangles, diamonds, ovals, and connecting lines. They reflect linguistic structure, with items serving as nouns and relationships as verbs. (Inc, 2024)

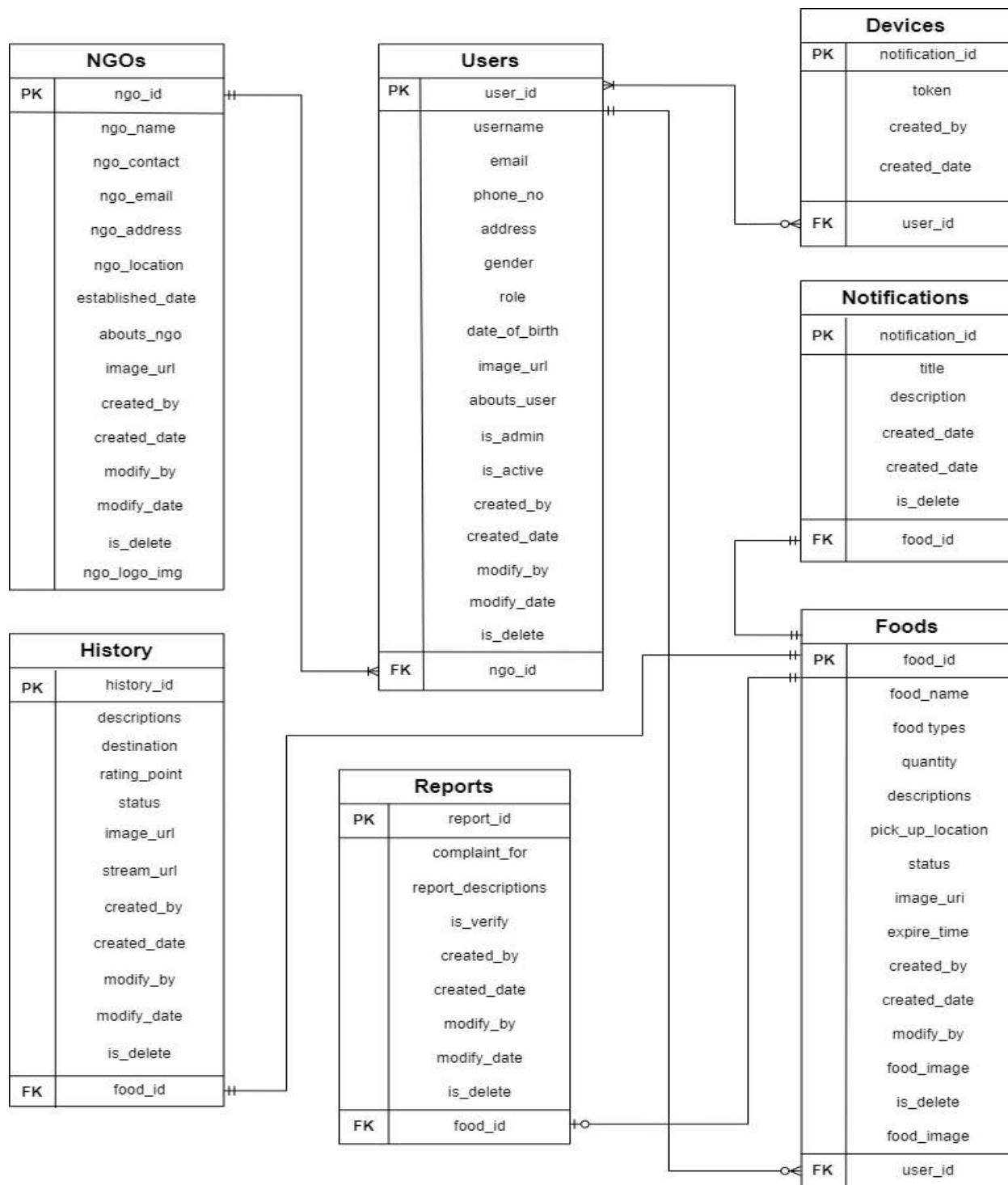


Figure 65: ER-Diagram

Appendix C: Entity Relation -Diagram

3.9. System Architecture Diagram

The system architecture is the conceptual model defining the structure, behaviour, and various views of a system. It involves identifying the components of the system, their relationships, and how they interact. System architecture reflects how the system functions and interacts with other systems and the external environment. It describes the interconnection of all system components and the data links between them. A system architecture consists of system components and subsystems that work together to implement the overall system (Mello, 2024). In waste food management have used the backend Django Rest Framework, database MySQL, as a frontend when HTML, CSS and mobile have Kotlin with Jet pack compose. The front-to-API through interaction backed the server where the frontend requests and the backend gives the response.

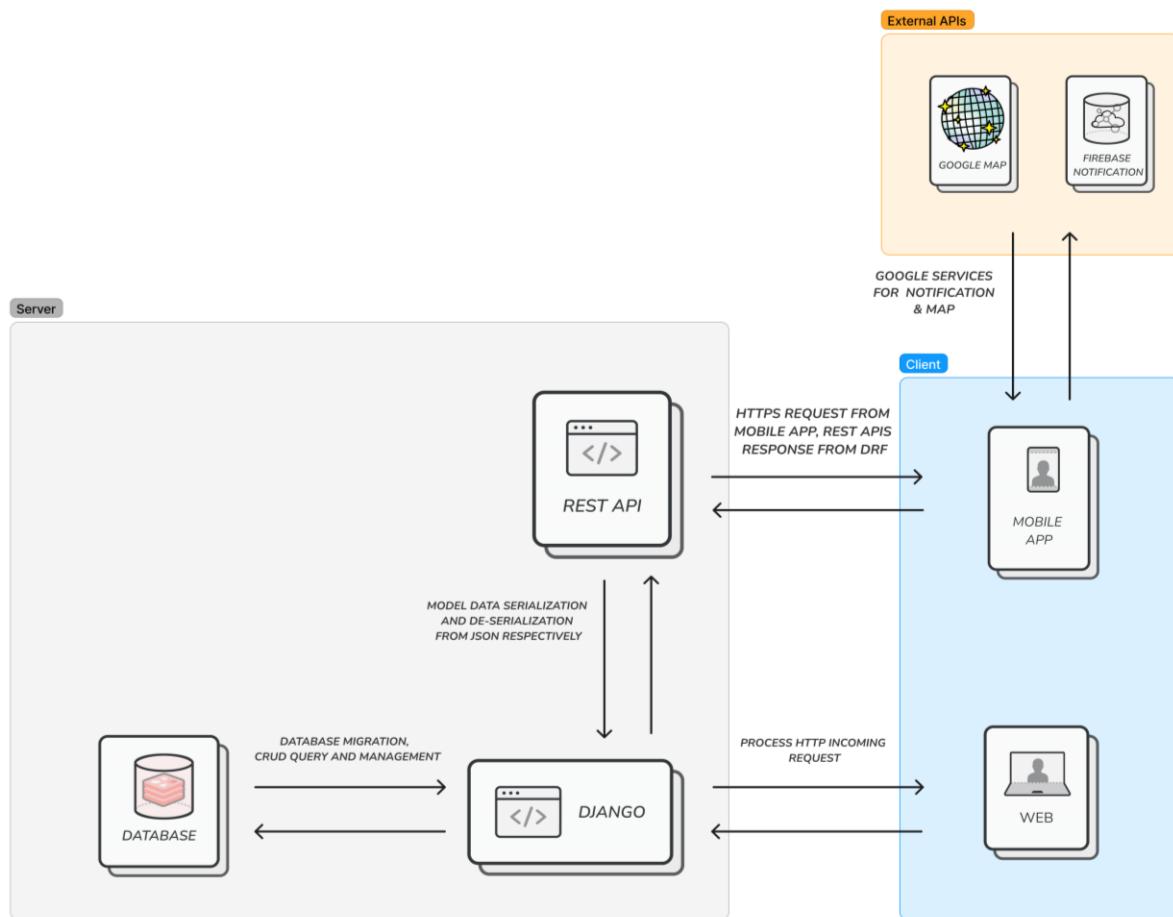


Figure 66: System Architecture Diagram

3.10. Class Diagram

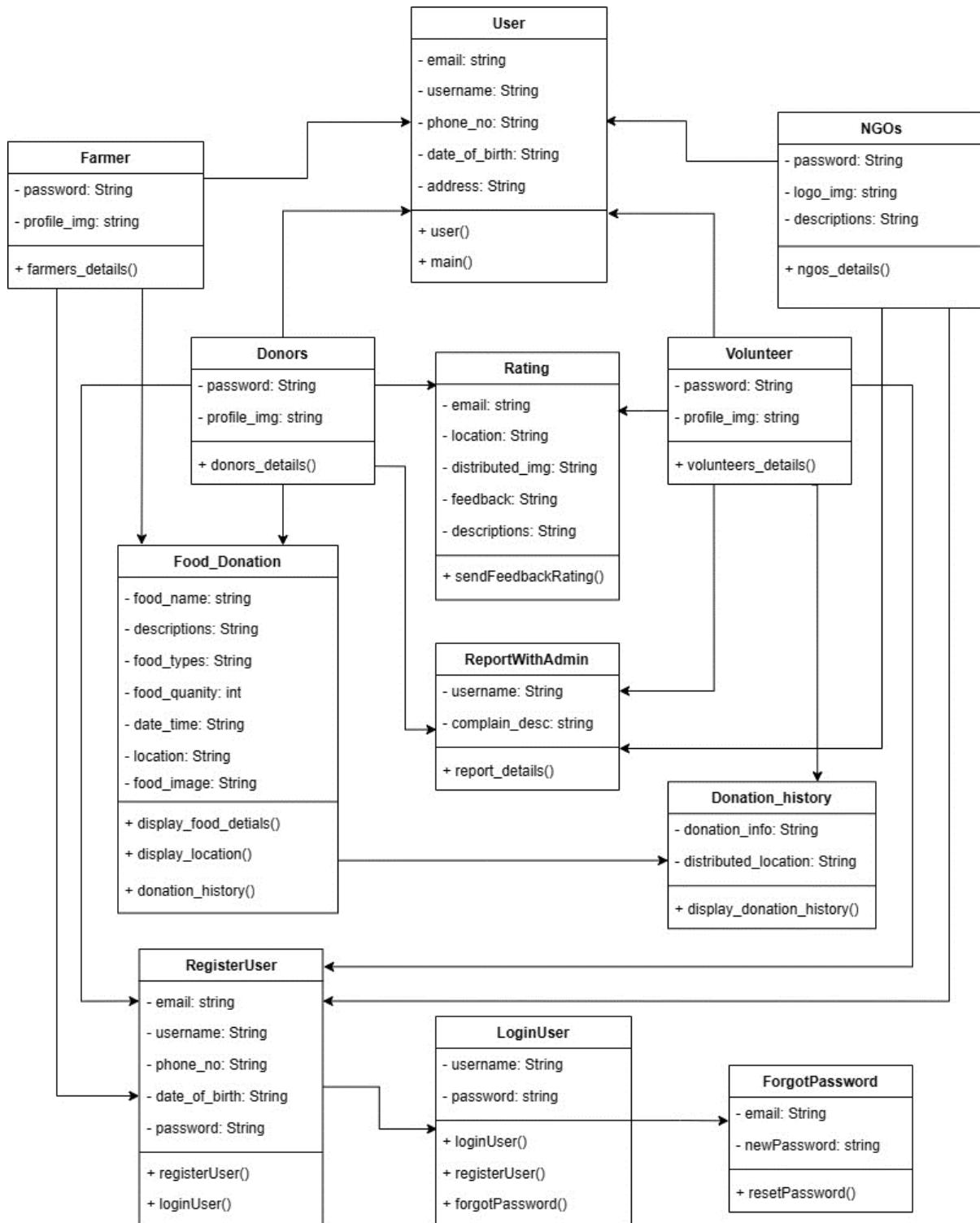


Figure 67: Class diagram of food donation application

3.11. DFD's level 2

This level gives a more thorough perspective of the system by dividing the sub-processes found in the level 1 DFD into further sub-processes. Each sub-process is represented as a separate process on the level 2 DFD. The data flows and storage connected with each sub-process are also shown. (GeekForGeek, GeekForGeek, 2024). This DFD's Level 2 (Data Flow Diagram's Level 2) has a more detailed explanation of the features or system flow. Level 2 DFD breaks down the sub-processes identified in Level 1 into further sub-processes, allowing for a more detailed analysis. Each sub-process is represented as a separate process on the Level 2 DFD. This level provides a more comprehensive understanding of the system's functionality by offering detailed insights into each sub-process. It illustrates the data flows and storage associated with each sub-process, providing a clearer picture of how data moves through the system.

3.11.1. Take Membership

Users can provide their membership details to sign up for a membership. The system verifies the validity of the provided details. If the details are valid, the system records the data in the database and grants the user membership privileges. However, if the details are not valid, the membership is rejected.

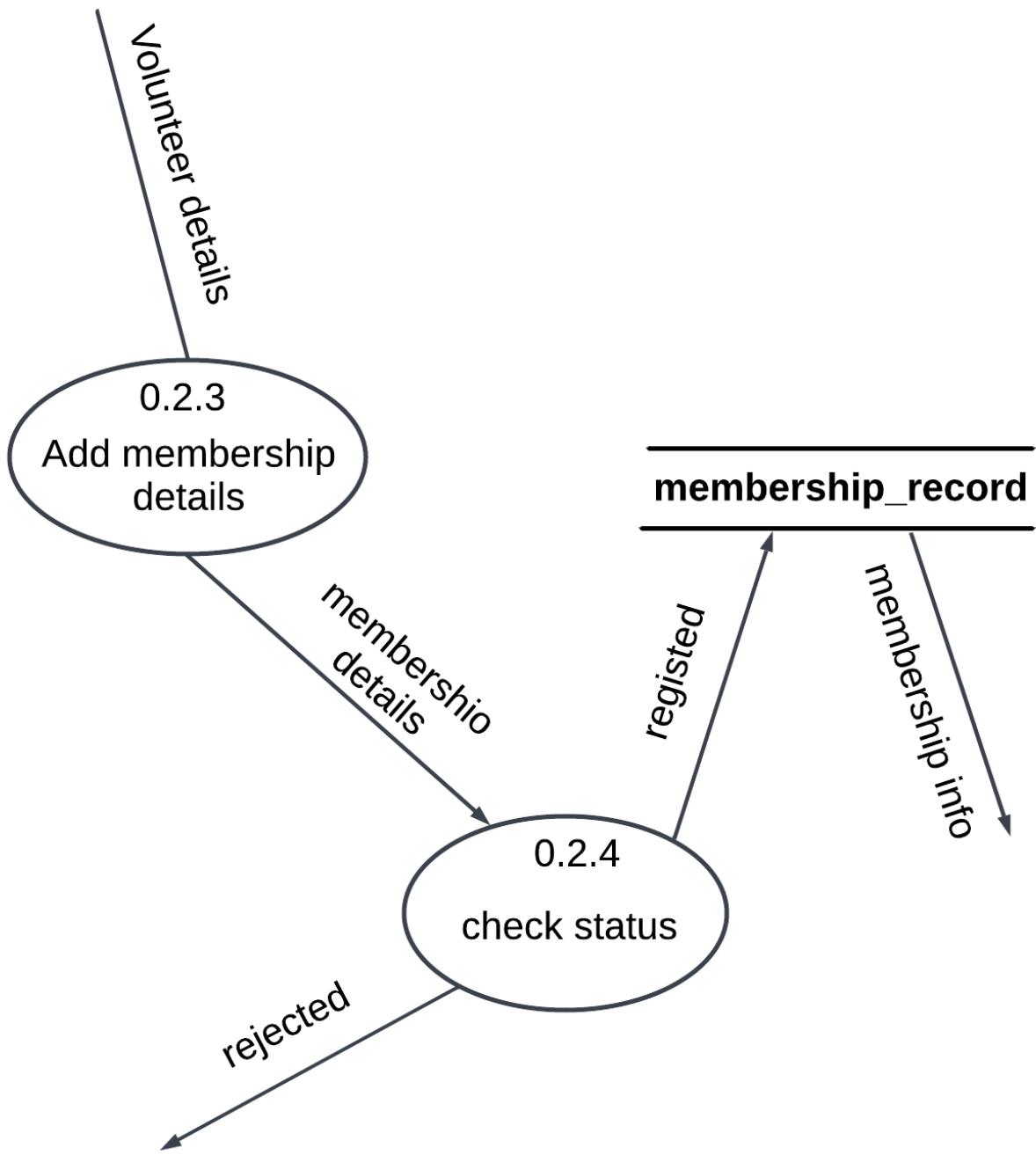


Figure 68: Take membership details DFD Level-2

3.11.2. Food Donate

The donor can donate or post food details for donation. The system checks whether the provided data is valid or not. If the posted information status details are not valid, the system shows an error message; otherwise, if the details are valid, the data is recorded in the database, and a success message is displayed for the donor. Additionally, another user can receive push notifications.

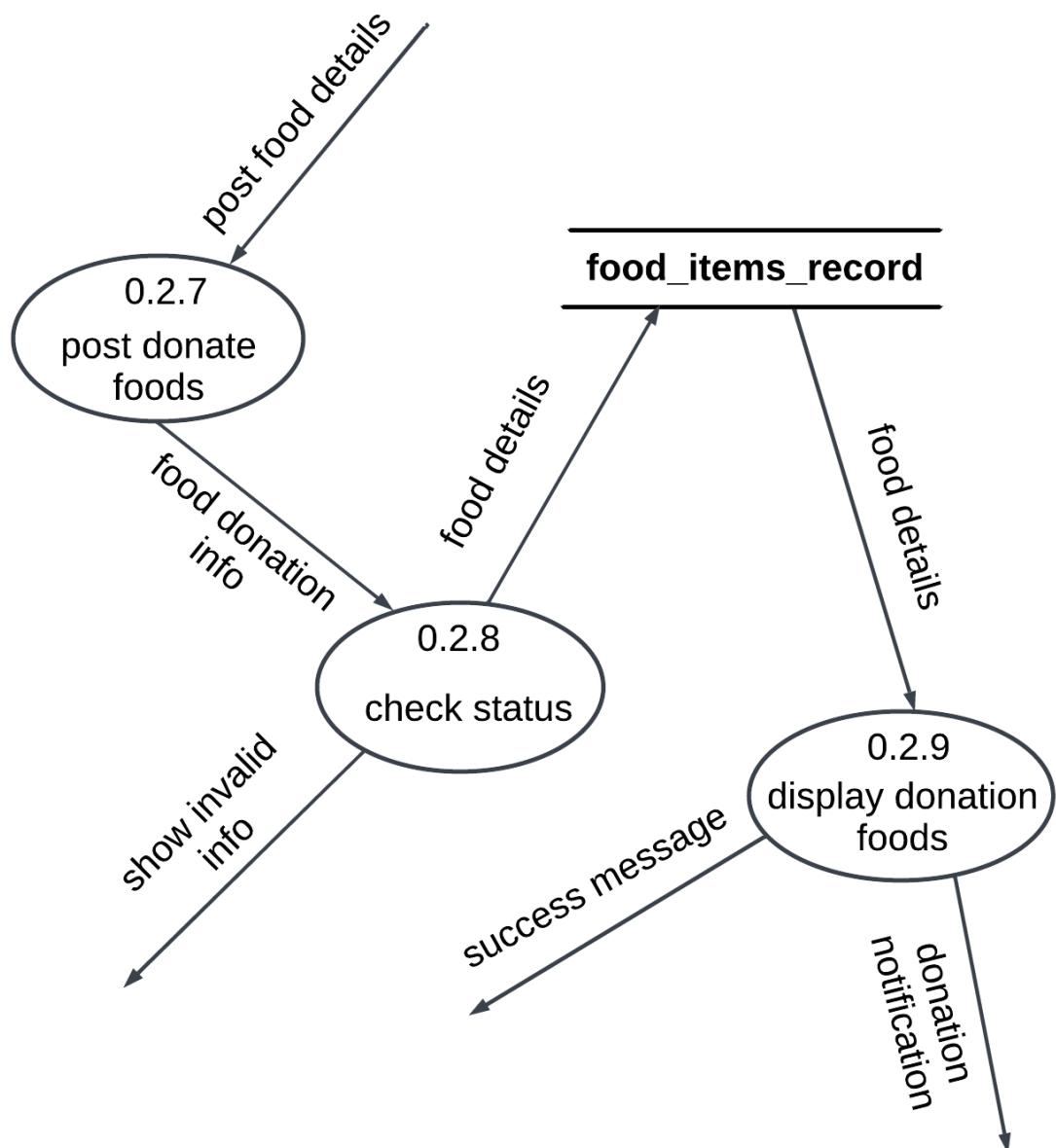


Figure 69: Food donation DFD Level-2

3.11.3. View Donation History

The user can request to view post or donation information details for their history. The request is verified by the system, and the data is then displayed to the user. The user can view more details about their history.

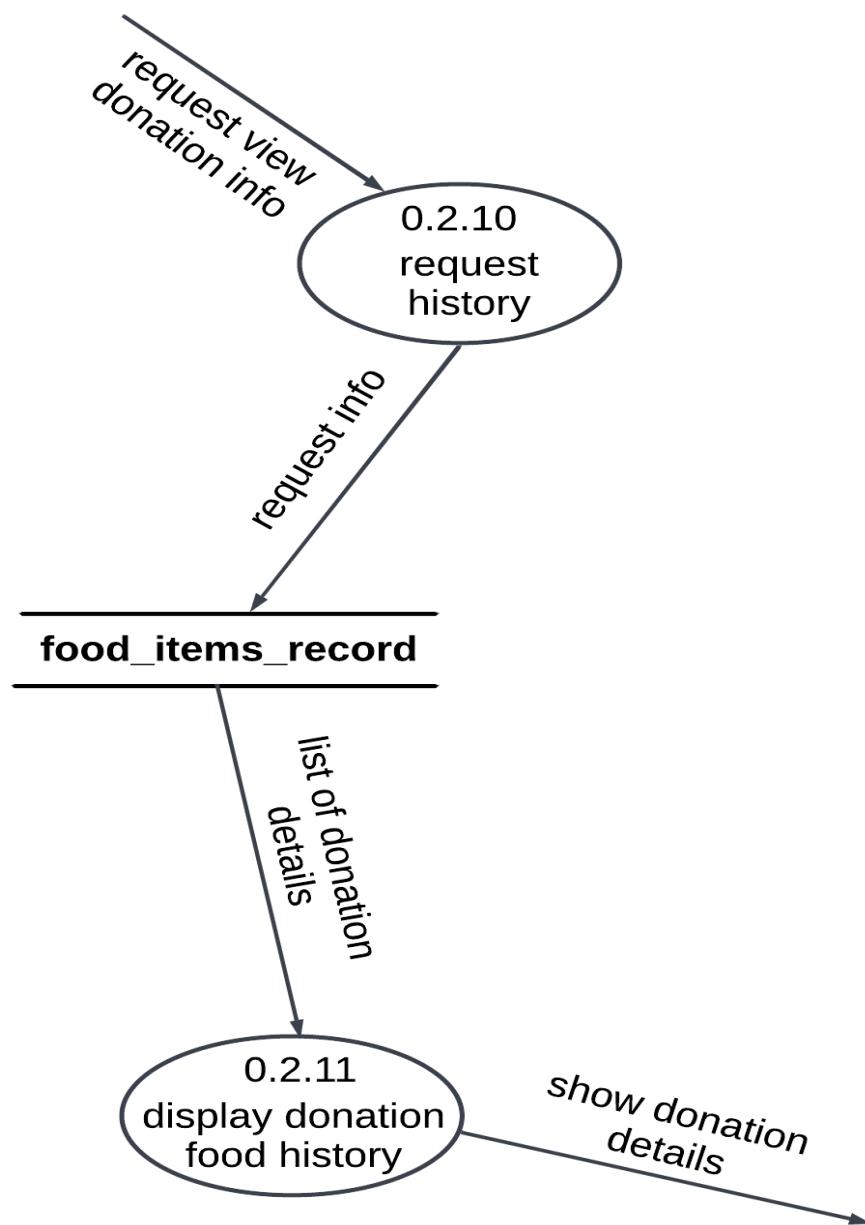


Figure 70: View Donation History DFD Level-2

3.11.4. Donation Rating

The volunteer can donate food received from donors and distribute the food to poor and homeless people. Then after completing the donation, the volunteer can provide the donation success information for a donor with a donation rating. The volunteer provides the rating details required for the system and it is verified and then recorded in the database displays the rating details with history details for the view more additional information.

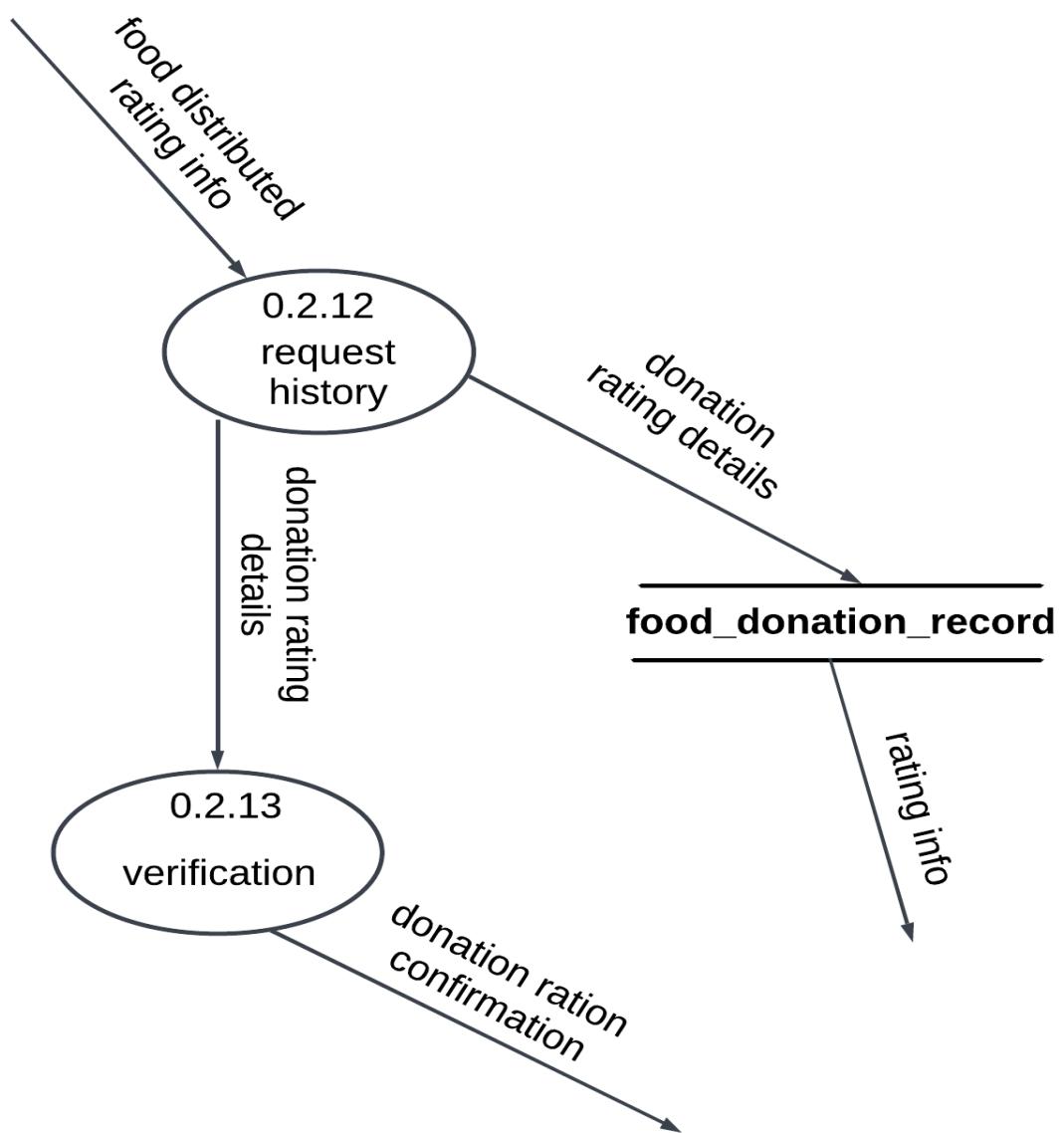


Figure 71: Donation rating DFD Level-2

3.11.5. Complain with the Admin

The user can add the report details to the system the send the admin the information if valid the records in the database other how the error message report details are valid the system can send the admin dashboard and the admin can verify.

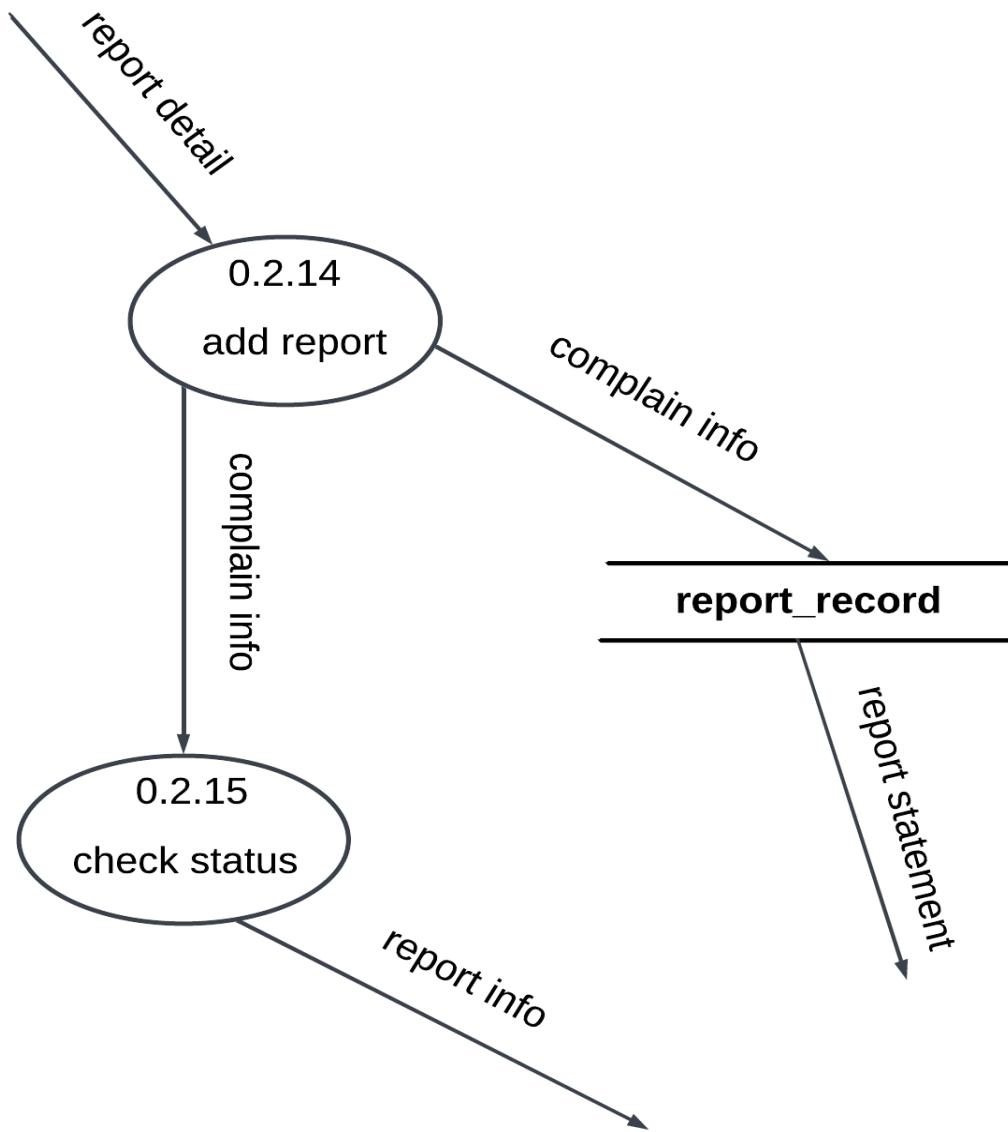


Figure 72: Complaint with admin DFD Level-2

[\(DFD Level-2 for appendix\)](#)

3.5. Implementation

The implementation phase of the system development involves translating the finalized design into a functional system.

3.5.1. System Development (Important screenshots of development core features and architecture)

3.5.1.1 Database model implementation

This is the just database model implemented where more core features implemented helpful. This phase is crucial as it lays the foundation for storing and managing data efficiently. The user model contains details about users, such as their personal information, login credentials, and any other relevant data. The food model stores information about food items, including their type, quantity, expiration date, and any additional attributes. The history model tracks the history of actions or events within the system, providing a log of user interactions, food donations, or other relevant activities. The notification model manages device tokens, enabling the system to send notifications to users' devices for important updates or events. This phase focuses on structuring the database schema to accommodate the core features of the application, ensuring seamless data management and retrieval throughout the system.

```
42
43 # create custom User table AbstractBaseUser
44 class Users(AbstractBaseUser):
45     email = models.EmailField(verbose_name='Email', max_length=255, unique=True)
46     username = models.CharField(max_length=100)
47     role = models.CharField(max_length=10, null=True)
48
49     address = models.CharField(max_length=100, null=True)
50     contact_number = models.CharField(max_length=16, unique=True, null=True)
51     gender = models.CharField(max_length=10, null=True)
52     date_of_birth = models.DateField(default=datetime.now, null=True)
53     abouts_user = models.TextField(max_length=500, null=True)
54     photo_url = models.ImageField(upload_to='user_images/', null=True, max_length=500)
55
56     is_admin = models.BooleanField(default=False)
57     is_active = models.BooleanField(default=False)
58     created_by = models.CharField(max_length=100, null=True, default='Self')
59     created_date = models.DateField(auto_now_add=True)
60     modify_by = models.CharField(max_length=50, null=True)
61     modify_date = models.DateField(null=True)
62     is_delete = models.BooleanField(default=False)
```

Figure 73: Implementation of the database user model

```

86     # crate the Food model
87     class Food(models.Model):
88         FOOD_TYPE_CHOICES = (
89             ('Others', 'Others'),
90             ('Cake', 'Cake'),
91             ('Green vegetables', 'Green vegetables'),
92             ('Biscuits & Chocolates', 'Biscuits & Chocolates'),
93             ('Sweet Snack', 'Sweet Snack'),
94             ('Stable Food', 'Stable Food'),
95             ('Fruits', 'Fruits'),
96             ('Meets', 'Meets'),
97             ('Water & Cold Drinks', 'Water & Cold Drinks'),
98         )
99         STATUS_CHOICES = ((('New', 'New'), ('Pending', 'Pending'), ('Completed', 'Completed'),),
100
101         food_name = models.CharField(max_length=100)
102         food_types = models.CharField(max_length=50, choices=FOOD_TYPE_CHOICES, default='Others')
103         quantity = models.IntegerField(null=True)
104         expire_time = models.CharField(max_length=10, null=True)
105         pick_up_location = models.CharField(max_length=100)
106         latitude = models.DecimalField(max_digits=22, decimal_places=16, default=0.0)
107         longitude = models.DecimalField(max_digits=22, decimal_places=16, default=0.0)
108         descriptions = models.TextField(null=True)
109         stream_url = models.ImageField(upload_to='food_images/', null=True, max_length=500)
110         status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='New')
111         created_by = models.CharField(max_length=100, null=True)
112         created_date = models.DateField(auto_now_add=True)
113         modify_by = models.CharField(max_length=50, null=True)
114         modify_date = models.DateField(null=True)
115         donor = models.ForeignKey(Users, on_delete=models.CASCADE, null=True) # FK (donor id)
116         is_delete = models.BooleanField(default=False)
117

```

Figure 74: Implementation of database food model.

```

118     # Reports
119     class Report(models.Model):
120         COMPLAINT_CHOICES = [
121             ('donor', 'Complaint to Donor'),
122             ('volunteer', 'Complaint to Volunteer'),
123             ('farmer', 'Complaint to Farmer'),
124         ]
125         complaint_to = models.CharField(max_length=30, choices=COMPLAINT_CHOICES, null=True)
126         descriptions = models.TextField(null=True)
127         is_verify = models.BooleanField(default=False)
128         created_by = models.CharField(max_length=50, null=True)
129         created_date = models.DateTimeField(auto_now_add=datetime.now)
130         modify_by = models.CharField(max_length=50, null=True)
131         modify_date = models.DateField(null=True)
132         is_delete = models.BooleanField(default=False)
133         food = models.ForeignKey(Food, on_delete=models.CASCADE) #FK
134

```

Figure 75: Implementation of database report model.

```

135     # create the History table
136     class History(models.Model):
137         STATUS_CHOICES = (('Pending', 'Pending'), ('Completed', 'Completed'),)
138
139         descriptions = models.TextField(max_length=300, null=True)
140         distributed_location = models.CharField(max_length=100, null=True)
141         rating_point = models.IntegerField(validators=[MinValueValidator(0), MaxValueValidator(5)], default=0)
142         distributed_date = models.DateField(null=True)
143         status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='Pending')
144         created_by = models.CharField(max_length=50, null=True)
145         created_date = models.DateTimeField(auto_now_add=datetime.now)
146         modify_by = models.CharField(max_length=50, null=True)
147         modify_date = models.DateField(null=True)
148         is_delete = models.BooleanField(default=False)
149         volunteer = models.ForeignKey(Users, on_delete=models.CASCADE, null=True) # FK (volunteer id)
150         food = models.ForeignKey(Food, on_delete=models.CASCADE, null=True) # FK (food id)
151

```

Figure 76: Implementation of database history model.

```

168     # create the notification table
169     class Notification(models.Model):
170         token = models.TextField(null=True)
171         created_by = models.CharField(max_length=100, null=True)
172         created_date = models.DateTimeField(auto_now_add=True)
173         is_delete = models.BooleanField(default=False)
174         user = models.ForeignKey(Users, on_delete=models.CASCADE) # FK

```

Figure 77: Implementation of database notification device token.

3.5.1.1 Implementation of Push Notification

The system integrates Firebase push notifications for both the front end (mobile) and back end (Django). The system utilizes Firebase Cloud Messaging (FCM) to enable push notifications between the mobile application and the Django backend. Firebase provides a robust and scalable platform for sending messages to devices in real time. On the front (mobile) side, the application leverages the Firebase Messaging library to send and receive push notifications. This allows the application to deliver timely updates and alerts to users' devices, keeping them informed about important events or changes. On the backend (Django) side, the system integrates Firebase Admin SDK to send push notifications to mobile devices. This allows the backend server to trigger notifications based on certain events or conditions, such as new updates or notifications from users. Overall, the implementation of Firebase push notifications enhances the system's communication capabilities, enabling seamless and efficient delivery of notifications to users across both frontend and backend components.

```

1  {
2      "type": "service_account",
3      "project_id": "food-share-833ef",
4      "private_key_id": "a53abd90b79ff0e960f856735333bb5d731a97f0",
5      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggsSkAgEAAoIBAQDAzX9VADnnnPQV\nRGZcBZqTEahLf17
6      "client_email": "firebase-adminsdk-hbwz@food-share-833ef.iam.gserviceaccount.com",
7      "client_id": "107582055280805840834",
8      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
9      "token_uri": "https://oauth2.googleapis.com/token",
10     "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
11     "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509.firebaseio-adminsdk-hbwz%40food-share-833ef.iam.gserviceaccount.com",
12     "universe_domain": "googleapis.com"
13 }
```

Figure 78: Implementation of Firebase service account key- backend.

```

194 # Initialize Firebase Admin SDK
195 cred = credentials.Certificate('serviceAccountKey.json')
196 firebase_admin.initialize_app(cred)
```

Figure 79: Implementation of service account key in Firebase admin.

```

21 # Firebase push notification
22 def send_notifications(title, body, list_of_tokens):
23     try:
24         # Create the message
25         message = messaging.MulticastMessage()
26             notification=messaging.Notification(
27                 title=title,
28                 body=body,
29             ),
30             tokens=list_of_tokens,
31         )
32
33     # Define headers
34     headers = {
35         'Content-Type': 'application/json',
36         'Authorization': 'AAAAw-NCzFM:APA91bHGq2kIJh4m9wBnNEFWIdMfwI_jSCcBxOCSPWDi0Sgvr_A2B8LyQvpPaA5baBVYmGV2ld5Q8osOTD0oJVf
37     }
38     messaging.send_multicast(message) # send message for multi device
39     return Response({'message': 'Notifications sent to all devices', 'is_success': True, 'status': 200})
40
41 except Exception as e:
42     return Response({'message': 'Failed to send notifications', 'is_success': False, 'status': 500})

```

Figure 80: Implementation of notification send function.

```

21 @SuppressLint("MissingFirebaseInstanceTokenRefresh")
22 class MyFirebaseMessagingService: FirebaseMessagingService() {
23     override fun onNewToken(token: String) {
24         super.onNewToken(token)
25         Log.e( tag: "Token", token)
26     }
27
28     // New fcm access token generated
29     fun getTokenInstance(): Task<String> {
30         return FirebaseMessaging.getInstance().token.addOnCompleteListener { task -
31             if (task.isSuccessful) task.result else task.exception
32         }
33     }
34
35     // Received notification
36     override fun onMessageReceived(remoteMessage: RemoteMessage) {
37         if (remoteMessage.notification != null) {
38             generateNotification( title: remoteMessage.notification?.title ?: "", message: remoteMessage.notification
39         }
40     }

```

Figure 81: implementation for receiving the notification message

```
 5  Sita Ram Thing "
6
7  private fun showNotification(title: String, description: String) {
8
9      val intent = Intent( packageContext: this, MainActivity::class.java)
10     intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
11
12     val pendingIntent: PendingIntent? = TaskStackBuilder.create(this).run {
13         addNextIntentWithParentStack(intent)
14         getPendingIntent( requestCode: 0, flags: PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE)
15     }
16
17     // Create the notification
18     val notification = NotificationCompat.Builder( context: this, CHANNEL_ID)
19         .setSmallIcon(R.mipmap.ic_launcher_foreground)
20         .setContentTitle(title)
21         .setContentText(description)
22         .setColor(ContextCompat.getColor( context: this, R.color.primary))
23         .setContentIntent(pendingIntent)
24         .setAutoCancel(true) // Dismiss the notification when clicked
25         .build()
26
27
28     // Show the notification
29     val notificationManager = this.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
30     notificationManager.notify( id: 1, notification)
31
32
33 }
```

Figure 82: Implementation for generating notifications.

3.5.1.2 Implementation of Donate/Post-Food

In the "Food Share" application, users, referred to as donors, are provided with the functionality to donate food items. This feature allows individuals or organizations who are interested in contributing to food donation initiatives to participate actively. Donors who have surplus food available can utilize this feature to post details about the food they intend to donate. This information may include the type of food, quantity, expiration date (if applicable), and any other relevant details. By utilizing this functionality, donors can easily contribute to the food donation process, thereby helping to reduce food waste and support those in need within the community.

```

408     # Donate food
409     class AddNewFoodView(APIView):
410         authentication_classes = [TokenAuthentication]
411         permission_classes = [AllowAny]
412         parser_classes = [MultiPartParser, FormParser]
413
414         def post(self, request):
415             try:
416                 serializer = FoodSerializer(data=request.data)
417
418                 # Retrieve FCM tokens from Notification model
419                 fcm_tokens = Notification.objects.all().values_list('token', flat=True)
420                 list_of_tokens = list(fcm_token (method) save: (**kwargs: Any) -> (Any | list)) | Any
421                 if serializer.is_valid():
422                     food_instance = serializer.save()
423                     title = food_instance.food_name
424                     body = food_instance.descriptions
425
426                     # Send notifications to all users
427                     result = send_notifications(title=title, body=body, list_of_tokens=list_of_tokens)
428
429                     # Check if notifications were sent successfully
430                     if result.status_code == 200:
431                         return Response({"message": "Donation successful", "is_success": True, "status": 200})
432                     else:
433                         return Response({"message": "Failed to send notifications", "is_success": False, "status": 400})
434                 else:
435                     return Response({"message": 'Enter the valid data', "is_success": False, "status": 400})
436             except Exception as e:
437                 return Response({"message": "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
438
439

```

Figure 83: Implement Food donations (post) - backend.

```
52  
53     // FoodDetails Donation page  
54     ▲ Sita Ram Thing  
55     @Multipart  
56     @POST(ApiUrl.NEW_FOOD_POST)  
57     suspend fun newFoodDonation(  
58         @PartMap dynamicParamsMap: Map<String, @JvmSuppressWildcards RequestBody>?,  
59         @Part imagePart: MultipartBody.Part?,  
60     ): ResponsePojo?
```

Figure 84: Food donation API call

3.5.1.3 Implementation of View History

Users, whether donors or volunteers, can view the history of donations made or actions taken within the system. This includes details such as the type and quantity of donated food, the date and time of donation, and any additional relevant information. Donors can access volunteer profile details, allowing them to view information about the volunteers who have participated in the donation process. Similarly, volunteers can view donor profile details, providing transparency and fostering a sense of community within the platform. This feature enhances user experience by providing access to comprehensive donation history details and facilitating interaction between donors and volunteers within the system.

```

715
716 class DonationHistory(APIView):
717     authentication_classes = [TokenAuthentication]
718     permission_classes = [AllowAny]
719
720     def get(self, request):
721         try:
722             donor_id = request.query_params.get("id") # Using query_params to get the donor_id
723
724             # Query to get all foods donated by the donor
725             donated_foods = Food.objects.filter(donor_id=donor_id, is_delete=False).order_by('-created_date')
726             if not donated_foods:
727                 return Response({"message": "No food history found", "is_success": False, "status": 400})
728
729             # Initialize an empty list to store donation entries
730             donation_entries = []
731             # Loop through each donated food
732             for food in donated_foods:
733                 food_data = FoodSerializer(food).data # Serialize the food data
734                 donor_data = UserSerializer(food.donor).data
735
736                 # Get the history for the current food item
737                 history = History.objects.filter(food=food).first()
738
739                 if history:
740                     history_data = HistorySerializer(history).data # Serialize the history data
741                     volunteer_data = UserSerializer(history.volunteer).data # Serialize the volunteer data
742                 else:
743                     history_data = {} # Empty dictionary if history is not available
744                     volunteer_data = {} # Empty dictionary if history is not available
745
746                 # Constructing the donation entry
747                 donation_entry = {
748                     'foods': food_data,
749                     'histories': history_data,
750                     'volunteer': volunteer_data,
751                     'donor': donor_data
752                 }
753                 donation_entries.append(donation_entry)
754
755             return Response({"message": "success", "is_success": True, "status": 200, "data": donation_entries})
756
757         except ObjectDoesNotExist:
758             raise NotFound({"message": "Donor or donation history not found.", "is_success": False, "status": 400}) # Raise a 404 Not Found exception
759         except Exception:
760             return Response({"message": "Sorry, something went wrong on our end. Please try again later.", "is_success": False, "status": 500})

```

Figure 85: Implementations of view donor history

```

521     # History
522     class GetFoodHistories(APIView):
523         authentication_classes = [TokenAuthentication]
524         permission_classes = [AllowAny]
525
526         def get(self, request):
527             histories = []
528             try:
529                 volunteer = request.query_params.get('query')
530                 if volunteer:
531                     history_details = History.objects.filter(volunteer=volunteer, is_delete=False).all()
532                     if not history_details:
533                         return Response({"is_success": True, "message": "No history found for this volunteer.", "histories": []})
534
535                     for entry in history_details:
536                         food_data = FoodSerializer(entry.food).data
537                         donor_data = UserSerializer(entry.food.donor).data
538                         volunteer_data = UserSerializer(entry.volunteer).data
539
540                         history = {
541                             'food_history': HistorySerializer(entry).data,
542                             'food': food_data,
543                             'donor': donor_data,
544                             'volunteer': volunteer_data
545                         }
546
547                         histories.append(history)
548
549             else:
550                 all_history = History.objects.all()
551                 for entry in all_history:
552                     food_data = FoodSerializer(entry.food).data
553                     donor_data = UserSerializer(entry.food.donor).data
554                     volunteer_data = UserSerializer(entry.volunteer).data
555
556                     history = {
557                         'Food_history': HistorySerializer(entry).data,
558                         'food': food_data,
559                         'donor': donor_data,
560                         'volunteer': volunteer_data
561                     }
562
563                     histories.append(history)
564
565             return Response({"is_success": True, "message": "History details successfully received.", "histories": histories})
566         except Exception as e:
567             return Response({"message": "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
568

```

Figure 86: Implementation of view volunteer history.

3.5.1.4 Implementation of Rating

The system incorporates a functionality where volunteers can provide feedback and rating points for donors who have donated food. Volunteers can give ratings and provide feedback regarding the donation experience to donors. This feedback may include factors such as the quality of the donated food, the donor's communication and cooperation, and overall satisfaction with the donation process. The system allows volunteers to collect and manage this feedback, ensuring that donors receive constructive input on their contributions. Additionally, volunteers can assign rating points to donors based on their evaluation of the donation experience. This feature promotes accountability and transparency within the donation ecosystem, allowing volunteers to recognize and appreciate donors for their contributions while also providing valuable feedback to enhance the donation process.

```

860 # Update User Profile Image
861 class DonationCompletedRating(APIView):
862     authentication_classes = [TokenAuthentication]
863     permission_classes = [IsAuthenticated]
864     permission_classes = [AllowAny]
865
866     def patch(self, request):
867         try:
868             history_id = request.data.get('id')
869             descriptions = request.data.get('descriptions')
870             location = request.data.get('location')
871             rating = request.data.get('rating')
872             if not history_id:
873                 return Response({"message": "No history Id provided", "is_success": False, "status": 400})
874             history = History.objects.filter(id=history_id).first()
875             if not history:
876                 return Response({"message": "History not found", "is_success": False, "status": 404})
877             history.distributed_date = datetime.now().strftime('%Y-%m-%d')
878             history.descriptions = descriptions
879             history.distributed_location = location
880             history.rating_point = rating
881             history.status = 'Completed'
882             history.save()
883             food_id = history.food.id
884             food = Food.objects.filter(id=food_id).first()
885             if food:
886                 food.status = 'Completed'
887                 food.save()
888                 return Response({"message": "Food donation is successful.", "is_success": True, "status": 200})
889             else:
890                 return Response({"message": "Food not found for the given history ID.", "is_success": False, "status": 404})
891         except Exception as e:
892             return Response({"message": "Sorry, something went wrong on our end. Please try again later.", "is_success": False, "status": 500})

```

Figure 87: Implementation of Donation rating - backend.

```
124  
125 💡 // Food donation rating|  
     ↳ Sita Ram Thing *  
126      @PATCH(ApiUrl.COMPLETED_FOOD_RATING)  
127      suspend fun donationRating(  
128          @Body foodDonateRatingDto: FoodDonateRatingDto?,  
129      ): ResponsePojo?  
130
```

Figure 88: Implementation of donation rating frontend.

3.5.1.6 Implementation of Complaint/Report

The system incorporates a feature that allows donors and volunteers to report issues or complaints to the admin. Both donors and volunteers have the option to report issues or file complaints to the system administrator. These reports may pertain to various scenarios, such as instances where a donor is unable to provide the promised food donation or when a volunteer fails to pick up the donated food. By providing this functionality, the system ensures that any discrepancies or problems encountered during the donation process can be promptly addressed and resolved. The admin can review the reported issues and take appropriate actions, such as facilitating communication between the parties involved or addressing any underlying issues within the system. The complaint/report feature enhances transparency and accountability within the donation platform, fostering trust and reliability among donors, volunteers, and administrators alike.

```

842 # Report User
843 class ReportToUser(APIView):
844     authentication_classes = [TokenAuthentication]
845     permission_classes = [IsAuthenticated]
846     permission_classes = [AllowAny]
847
848     def post(self, request):
849         try:
850             serializer = ReportSerializer(data=request.data)
851             if serializer.is_valid():
852                 serializer.save() # Save the serializer instance to the database
853                 return Response({"message": "Report has been submitted", "is_success": True, "status": 200})
854             else:
855                 return Response({"message": "Report is not sumbit", "is_success": False, "status": 400})
856         except Exception:
857             return Response({"message": "Sorry, something went wrong on our end. Please try again later.", "is_success": False, "status": 500})
858

```

Figure 89: Implementation of report - backend.

```
297     onDismissRequest = { isReportConfirmation = false },
298     enabled = message.isNotEmpty(),
299     confirmButton = {
300         if (isConnected) {
301             val reportDTO = ReportDTO(
302                 complaintTo = context.getString(R.string.volunteer),
303                 descriptions = message.trim(),
304                 createdBy = username,
305                 food = foodId
306             )
307             donorHistoryViewModel.getReportToUser(reportDTO)
308         } else {
309             showToast(context, "No Internet connection")
310         }
311         isReportConfirmation = false
312     }
313 }
314 }
```

Figure 90: Implementation of report-frontend.

3.5.1.7 Implementation of Map

The system incorporates a feature that utilizes maps to display donation locations. The system integrates mapping functionality to enable users to visualize the locations where food donations are made. Users, such as donors or volunteers, can access a map view within the application or platform to see the geographic distribution of donation points. By leveraging mapping technology, users can easily identify nearby donation locations and plan their donation or pickup activities accordingly. The map may display markers or pins indicating specific donation sites, allowing users to navigate and locate them with ease. Overall, the map implementation enhances the user experience by providing a visual representation of donation locations, facilitating efficient navigation, and promoting active participation in the donation process.

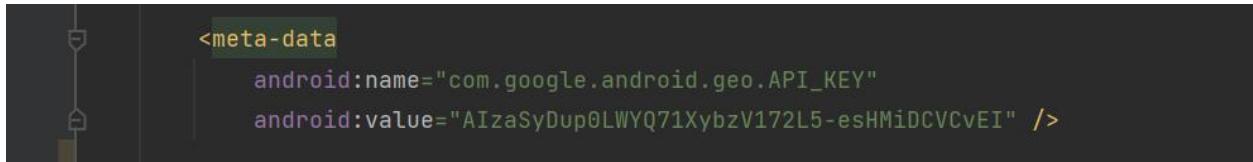


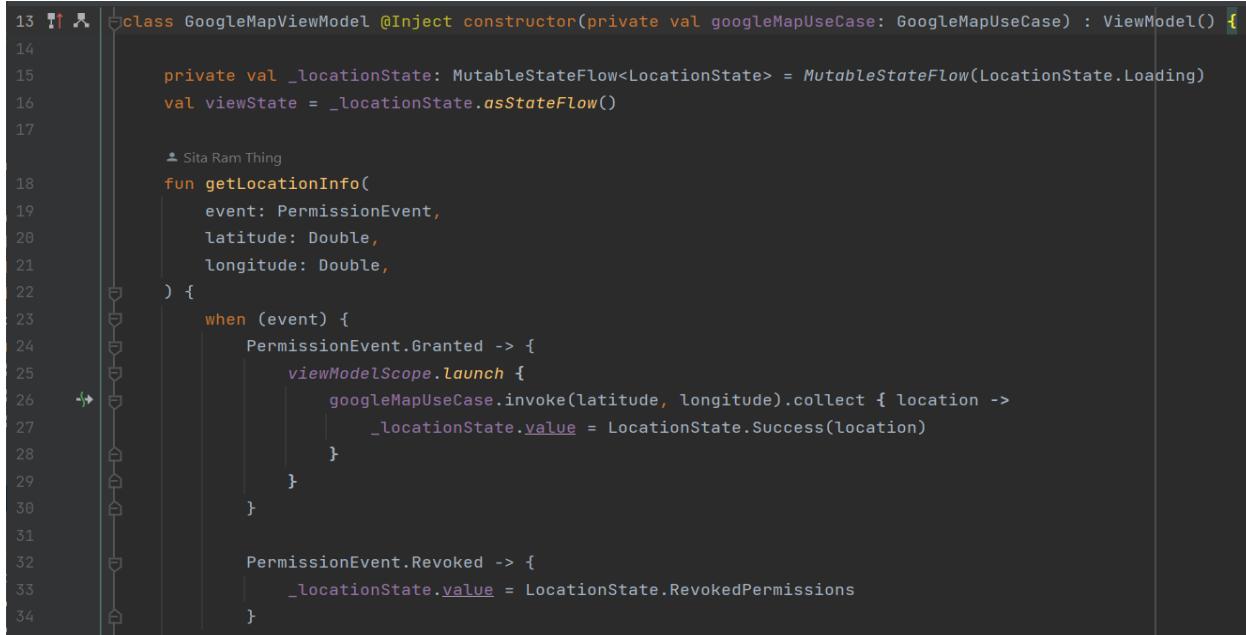
Figure 91: Implementation of map key

```
// Check if the app has location permission
if (!hasViewLocationPermission(context)) {
    trySend(element: null)
    return@callbackFlow
}

// Define the location request parameters
val request = LocationRequest.Builder(intervalMillis: 1L)
    .setIntervalMillis(1L)
    .setPriority(Priority.PRIORITY_HIGH_ACCURACY)
    .build()

// Define the callback for location updates
val currentLocation = object : LocationCallback() {
    override fun onLocationResult(locationResult: LocationResult) {
        if (latitude == 0.0 || longitude == 0.0) {
            locationResult.locations.lastOrNull()?.let {
                trySend(LatLng(it.latitude, it.longitude))
            }
        } else {
            trySend(LatLng(latitude, longitude))
        }
    }
}
```

Figure 92: Implementation of map access to the current location late value

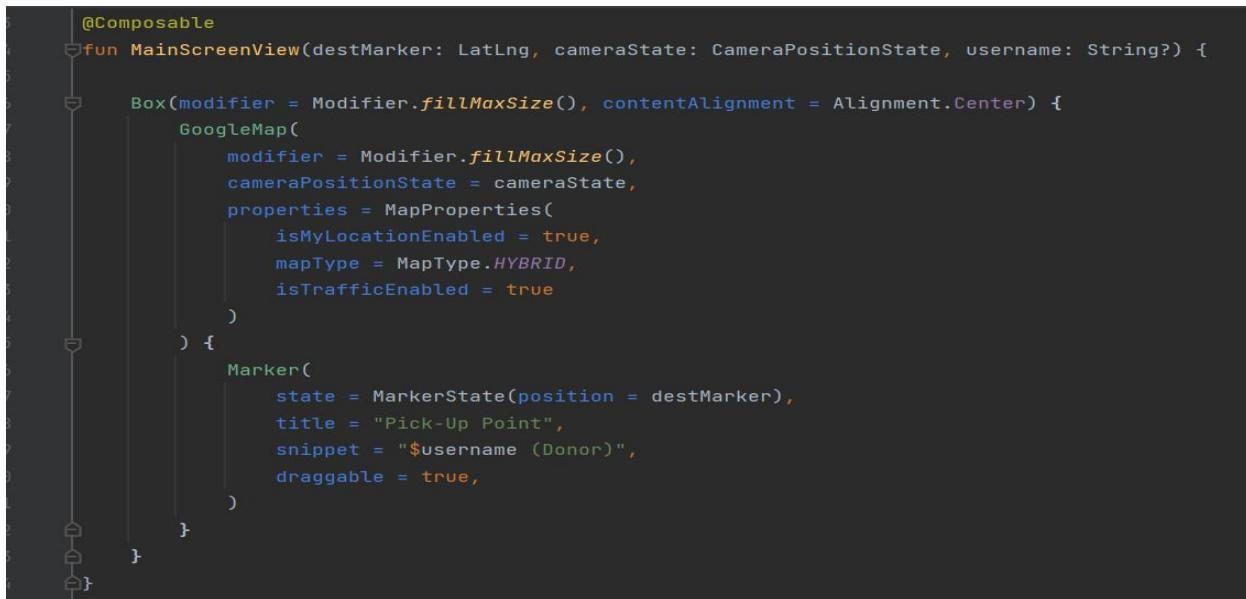


```

13  class GoogleMapViewModel @Inject constructor(private val googleMapUseCase: GoogleMapUseCase) : ViewModel() {
14
15     private val _locationState: MutableStateFlow<LocationState> = MutableStateFlow(LocationState.Loading)
16     val viewState = _locationState.asStateFlow()
17
18     fun getLocationInfo(
19         event: PermissionEvent,
20         latitude: Double,
21         longitude: Double,
22     ) {
23         when (event) {
24             PermissionEvent.Granted -> {
25                 viewModelScope.launch {
26                     googleMapUseCase.invoke(latitude, longitude).collect { location ->
27                         _locationState.value = LocationState.Success(location)
28                     }
29                 }
30             }
31
32             PermissionEvent.Revoked -> {
33                 _locationState.value = LocationState.RevokedPermissions
34             }
35         }
36     }
37
38     companion object {
39         const val TAG = "GoogleMapViewModel"
40     }
41 }

```

Figure 93: Implementation of map state man tent.



```

1  @Composable
2  fun MainScreenView(destMarker: LatLng, cameraState: CameraPositionState, username: String?) {
3
4      Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
5          GoogleMap(
6              modifier = Modifier.fillMaxSize(),
7              cameraPositionState = cameraState,
8              properties = MapProperties(
9                  isMyLocationEnabled = true,
10                 mapType = MapType.HYBRID,
11                 isTrafficEnabled = true
12             )
13         )
14         Marker(
15             state = MarkerState(position = destMarker),
16             title = "Pick-Up Point",
17             snippet = "$username (Donor)",
18             draggable = true,
19         )
20     }
21 }

```

Figure 94: Implementation of Google Maps.

Appendix C: Implementations of more features

3.5.2. System Architecture

A system's development reflects its intended usage as well as its interactions with external systems and other systems. It explains how every part of the system is connected to every other part of the system through a data link. A system's architecture is the reflection of how its structure, functions, and interactions are considered. The Food share system allows users to interact with both mobile and desktop devices, where these devices can connect from the front end to the back end through an API built with the Django REST Framework. The Django server can connect to the MySQL database. Through the REST API, data can be stored in the database and fetched from it.

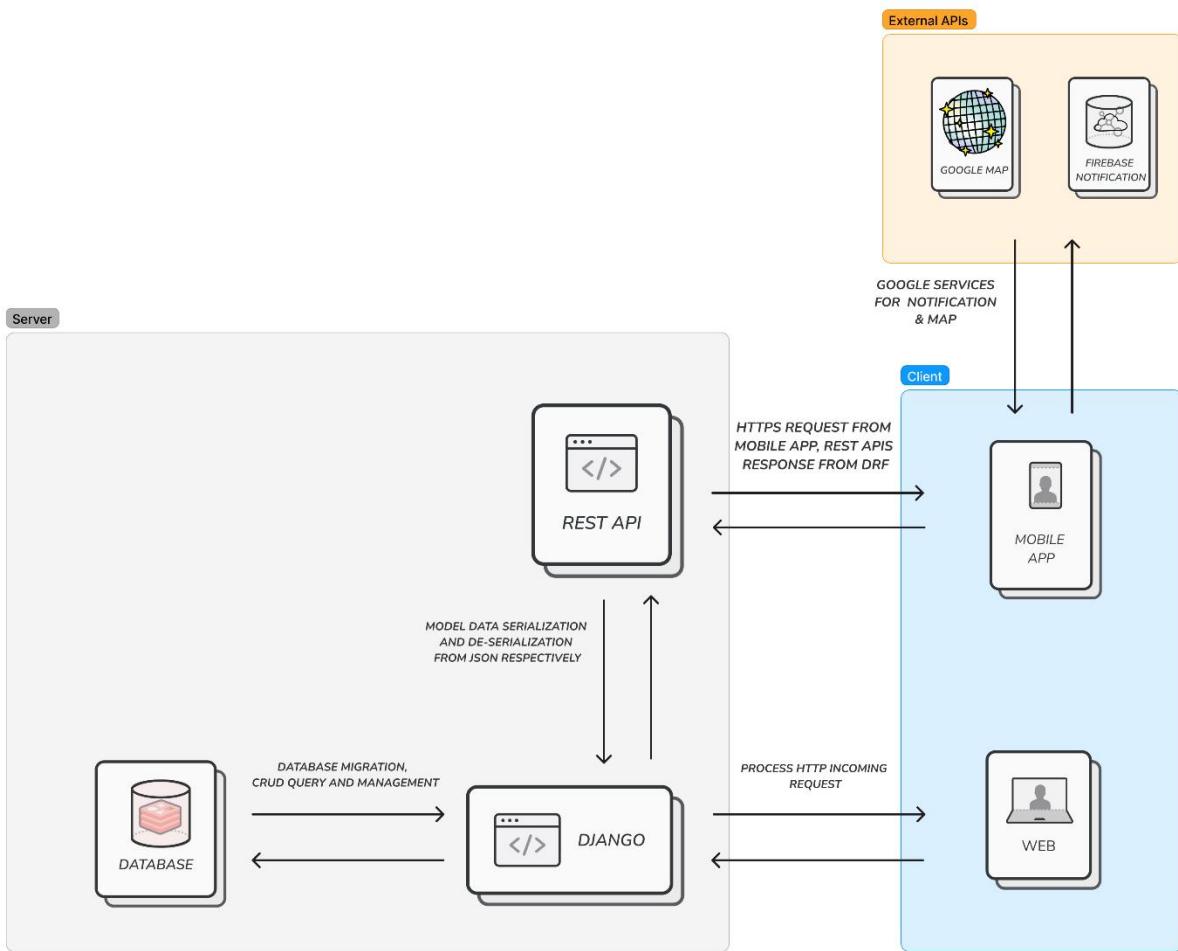


Figure 95: System Architecture

The system can be developed following the Clean Architecture with Dependency Injection implementation. This approach emphasizes modular design and separation of concerns, enabling easier maintenance and scalability. Let's delve into the details of how each layer contributes to the system's components:

3.5.3.1 Presentation Layer:

The Presentation Layer is responsible for handling the user interface (UI) components, including both web and mobile interfaces. In this layer, the frontend UI and view models are developed, with a focus on maintaining the application's state. In a web application, HTML, CSS, and JavaScript would be utilized to develop the user interface, while in a mobile app Kotlin with Jetpack compose is implemented. Integration with external services such as Google Maps and Firebase push notifications is implemented within this layer to enhance user experience and functionality.

3.5.2.2 Domain Layer:

The Domain Layer contains the business logic of the application. Here, different business rules and functionalities are defined to ensure the core operations of the system. Interfaces such as repositories, use cases, and data transfer objects (DTOs) are created within this layer to abstract the interactions between the Presentation and Data layers. These interfaces serve as contracts that define how data is exchanged between layers. The domain layer may include use cases for managing donations, handling user authentication, and coordinating logistics.

3.5.2.3 Data Layer:

The Data Layer is responsible for API implementation to manage data storage, retrieval, and persistence. Here, the implementation of database connections is carried out. Two types of database connections are typically implemented: local and remote. The local database often utilizes technologies like Room for Android services as a cache for frequently accessed data. On the other hand, the remote database, such as MySQL stores the primary dataset and is accessed over the network. The Data Layer communicates with the databases and abstracts away the specific details of database management from the rest of the application. By adhering to the Clean Architecture

principles, the system components are effectively organized into layers, each with its distinct responsibilities. This modular approach enables easier maintenance, testing, and future expansion of the system. Additionally, the implementation of Dependency Injection facilitates loose coupling between components, allowing for greater flexibility and ease of integration.

CHAPTER 4: TESTING AND ANALYSIS

4.1. Test Plan

Before initiating any tests, a detailed test plan was meticulously developed to streamline the testing process, adhere to testing protocols, and allocate responsibilities for both system and unit testing. This plan served as a schematic, facilitating the testing process and ensuring the orderly distribution of materials following various testing rules. Below, we outline the test plans for the two main tests conducted on the application.

4.1.1. Unit Testing

Unit testing is testing individual software modules or components separately to ensure they operate as intended. A single function, method, or class is usually represented by each unit, which is the smallest tested component of the program. The purpose of unit tests is to confirm how these units behave under different input scenarios and edge situations. Developers can find and address defects early in the development cycle, resulting in more dependable and maintainable code, by isolating each unit and testing it separately. Throughout the development process, unit tests are often automated and run, which enables developers to identify regressions fast and make sure that modifications to the codebase don't generate new defects. Compound errors occur less frequently the earlier an issue is discovered. (Levan, 2024)

An essential part of our testing procedure is unit testing. After finishing each unit or feature, it's crucial to thoroughly test every component. My primary focus will be planning the project's unit testing phase and ensuring that each system component is put through a thorough testing process. The system has some components or features that are unit testing.

[Appendix D: Unit test plans](#)

4.1.2. System Testing

For interconnected systems, system testing is essential because any software or system flaw could put users through serious problems. System testing is the process of examining the overall system's performance and quality. It is also known as system-level testing or system integration testing (Reichert, 2024).

I completed the development of my wasted food donation system with all features implemented to meet the client's requirements. To ensure the system's functionality and user satisfaction, I will devise a plan for end-users to collect feedback, report issues, and receive responses. The system will undergo comprehensive testing to ensure all features work seamlessly. In case any issues arise during testing, a resolution plan will be created and implemented promptly. System testing will involve end-users and the client to gather feedback and ensure the system meets their expectations. This collaborative approach will ensure that the wasted food donation system is not only functional but also user-friendly and meets the client's needs effectively. The major test for UI and UX with system performance, google Maps, notification permission and internet status check, and stat management testing with the user.

4.2. Test Execution

The test plan is completed and now going to test. After applying unit tests and system testing, the actual results were found, and any errors were resolved. The identified issues are currently being researched to improve the test implementation. the testing is given below:

4.2.1. Unit Testing

4.2.1.1 Register Unit Test

Test Case	Descriptions
Objectives	To register under the test case function
Actions	<p>Insert sample data (email, username, role, and password) to test the API and run the test function.</p> <p>Input the :</p> <ul style="list-style-type: none"> email: test@example.com, username: tester, role: normal, password: test
Expected Result	The user is registered successfully without any error. The response status code is 200, the message is registered successfully, the 'is_success' attribute is True, and the 'status' attribute is 200
Actual Result	The register unit test is a success after failure resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 18: Register Unit Test

```

class RegisterUserTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()

    def test_register_user(self):
        data = {
            'email': 'test@example.com',
            'username': 'testuser',
            'role': 'normal',
            'password': 'password'
        }
        request = self.factory.post('api/register/user', data)
        view = RegisterUser.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['message'], 'User registered successfully')
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['status'], 200)

        # Check if the user is created
        user = Users.objects.get(email='test@example.com')
        self.assertEqual(user.email, 'test@example.com')
        self.assertEqual(user.username, 'testuser')
        self.assertEqual(user.role, 'normal')
        self.assertTrue(user.check_password('password'))
        self.assertFalse(user.is_admin)
        self.assertTrue(user.is_active)

```

Figure 96: Register User unit test function

```

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Backend\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_register_user (foodshare.tests.RegisterUserTestCase.test_register_user)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Backend\Django\foodshare\tests.py", line 43, in test_register_user
    self.assertTrue(user.is_active)
AssertionError: False is not true

-----
Ran 1 test in 0.677s

FAILED (failures=1)
Destroying test database for alias 'default'...

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Backend\Django>

```

Figure 97: Register User unit test failed

Register success unit test

```

class RegisterUserTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()

    def test_register_user(self):
        data = {
            'email': 'test@example.com',
            'username': 'testuser',
            'role': 'normal',
            'password': 'password'
        }
        request = self.factory.post('api/register/user', data)
        view = RegisterUser.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['message'], 'User registered successfully')
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['status'], 200)

        # Check if the user is created
        user = Users.objects.get(email='test@example.com')
        self.assertEqual(user.email, 'test@example.com')
        self.assertEqual(user.username, 'testuser')
        self.assertEqual(user.role, 'normal')
        self.assertTrue(user.check_password('password'))
        self.assertFalse(user.is_admin)

```

Figure 98: Register success unit test function

```

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Backend\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

.
-----
Ran 1 test in 0.732s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Backend\Django>

```

Figure 99: Register success unit test response

4.2.1.2 Login Authentication Unit test

Test Case	Descriptions
Objectives	To develop the login authenticate the test case function
Actions	<p>Insert sample data (email, username, role, and password) to test the API and run the test function.</p> <p>Input the :</p> <p style="padding-left: 40px;">email: test@example.com,</p> <p style="padding-left: 40px;">password: team password</p>
Expected Result	The user is logged in successfully without any error. The response status code is 200, the message is 'Log in successful', the 'is_success' attribute is True, and the 'status' attribute is 200
Actual Result	The login unit test was a success after failure resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 19: Login Authentication Unit test

```
class LoginUserTestCase(TestCase):

    def setUp(self):
        self.factory = APIRequestFactory()

    def test_login_user(self):
        request = self.factory.post('/api/authenticate/token/', {'email': 'test@example.com', 'password': 'testpassword'})
        view = LoginUser.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['message'], 'Login successful')
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['status'], 200)
        self.assertIn('auth', response.data)
        auth_data = response.data['auth']
        self.assertEqual(auth_data['username'], 'testuser')
        self.assertEqual(auth_data['email'], 'test@example.com')
```

Figure 100: Login User unit test function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_login_user (foodshare.tests.LoginUserTestCase.test_login_user)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 85, in test_login_user
    self.assertEqual(response.data['message'], 'Login successful')
AssertionError: 'Your account is not activate' != 'Login successful'
- Your account is not activate
+ Login successful

-----
Ran 1 test in 0.562s

FAILED (failures=1)
Destroying test database for alias 'default'...

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 101: Register User unit test failer response

Login success unit testing

```

6  class LoginUserTestCase(TestCase):
7
8      def setUp(self):
9          self.factory = APIRequestFactory()
10         # Register the user
11         data = {
12             'email': 'test@example.com',
13             'username': 'testuser',
14             'role': 'normal',
15             'password': 'test'
16         }
17         request = self.factory.post('/api/register/user/', data)
18         view = RegisterUser.as_view()
19         response = view(request)
20
21         # Get the user
22         self.user = Users.objects.get(email='test@example.com')
23         self.user.is_active = True
24         self.user.save()
25
26         # Create a token for the user
27         self.token = Token.objects.create(user=self.user)
28
29     def test_login_user(self):
30         # Log in the user
31         request = self.factory.post('/api/authenticate/token/', {'email': 'test@example.com', 'password': 'test'})
32         view = LoginUser.as_view()
33         response = view(request)
34
35         self.assertEqual(response.status_code, 200)
36         self.assertEqual(response.data['message'], 'Login successful')
37         self.assertEqual(response.data['is_success'], True)
38         self.assertEqual(response.data['status'], 200)
39
40

```

Figure 102: Register success unit test function

```

Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

Ran 1 test in 0.628s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>

```

Figure 103: Login success unit test response

4.2.1.3 Device token save unit

Test Case	Descriptions
Objectives	To develop the device token save the test case function
Actions	<p>Insert sample data (user_id, email and created_by) to test the API and run the test function.</p> <p>Input the :</p> <ul style="list-style-type: none"> user_id: userId, token: new_device_token created_by: test_user
Expected Result	The user device token is saved successfully without any error. The response status code is 200, the message is 'New device token saved', the 'is_success' attribute is True, and the 'status' attribute is 200
Actual Result	The device token save unit test is a success after failure resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 20: Device token save unit

```
class DeviceTokenViewTestCase(TestCase):

    def setUp(self):
        self.factory = APIRequestFactory()

    def test_save_device_token(self):
        request = self.factory.post('/api/fcm/device/token/save/', {'user_id': 39, 'token': 'new_test_token_save', 'created_by': 'Test ser'})
        view = DeviceTokenView.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['message'], 'New device token saved')
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['status'], 200)
```

Figure 104: Device token save unit test function

```
System check identified no issues (0 silenced).
F
=====
FAIL: test_save_device_token (foodshare.tests.DeviceTokenViewTestCase.test_save_device_token)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 71, in test_save_device_token
    self.assertEqual(response.data['message'], 'New device token saved')
AssertionError: 'Sorry, something went wrong on our end. Please try again later.' != 'New device token saved'
- Sorry, something went wrong on our end. Please try again later.
+ New device token saved

-----
Ran 1 test in 0.005s

FAILED (failures=1)
Destroying test database for alias 'default'...
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>|
```

Figure 105: Device token save unit test failer response

Save device token unit test

```
# Unit test
class DeviceTokenViewTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='role', password='testpassword')

    def test_device_token_view(self):
        user_id = self.user.id
        token = "new_device_token"
        created_by = "test_user"

        request = self.f (variable) request: WSGIRequest en/save?user_id={user_id}&token={token}&created_by={created_by}'')
        view = DeviceTok
        response = view(request)

        self.assertEqual(response.status_code, 200)

        # Check if the device token is saved correctly
        device = Device.objects.get(user_id=user_id)
        self.assertEqual(device.token, token)
        self.assertEqual(device.created_by, created_by)
```

Figure 106: Save device token success unit test function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
-----
Ran 1 test in 0.328s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 107: Save device token success unit test response

4.2.1.4 View history unit test

Test Case	Descriptions
Objectives	To develop the history details of the test case function
Actions	To call the food history details API function.
Expected Result	The food history details were successful without any errors. The response status code is 200, the message is 'Success', and the 'is_success' attribute is True.
Actual Result	The history unit test was a success after failure and resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 21: View history unit test

```
# History
class HistoryDetailsTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='donor', password='testpassword')

    def test_history_details(self):
        request = self.factory.get('/api/food/history/all/')
        force_authenticate(request, user=self.user)
        view = HistoryDetails.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['message'], "Success")
```

Figure 13: History details get unit test function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_history_details (foodshare.tests.HistoryDetailsTestCase.test_history_details)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 198, in test_history_details
    self.assertEqual(response.data['is_success'], True)
AssertionError: False != True

-----
Ran 1 test in 0.360s

FAILED (failures=1)
Destroying test database for alias 'default'...

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>[]
```

Figure 108: History details get unit test failer response

View success history details

```
# History
class HistoryDetailsTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='donor', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)

    def test_history_details(self):
        request = self.factory.get('/api/food/history/all/')
        force_authenticate(request, user=self.user)
        view = HistoryDetails.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['message'], "Success")
|
```

Figure 14: View history success unit test case

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
-----
Ran 1 test in 0.328s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>[]
```

Figure 15: View history success unit test response

4.2.1.5 Report to admin history unit test

Test Case	Descriptions
Objectives	To develop the history details of the test case function
Actions	To call the food history details API function. Food: food_id Complaint_to: complaint_update@example.com Descriptions: Update descriptions Created_by: user_id
Expected Result	The complaint to admin details was successful without any errors. The response status code is 200, and the 'is_success' attribute is True.
Actual Result	The report to the admin unit test was a success after failure and resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 22: Report to admin history unit test

```

class ReportToUserTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='donor', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)
        self.report = Report.objects.create(food=self.food, complaint_to='complaint@example.com', descriptions='Test description', created_by=self.user)

    def test_report_update(self):
        request = self.factory.post('/api/user/report/')
        data = [
            'food': self.food.id,
            'complaint_to': 'complaint_updated@example.com',
            'descriptions': 'Updated description',
            'created_by': self.user.id
        ]
        request.data = data
        view = ReportToUser.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
    
```

Figure 109: Report to admin unit test function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_report_update (foodshare.tests.ReportToUserTestCase.test_report_update)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 221, in test_report_update
    self.assertEqual(response.data['is_success'], True)
AssertionError: False != True

-----
Ran 1 test in 0.339s

FAILED (failures=1)
Destroying test database for alias 'default'...

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 110: report to admin unit test response

Report to administrator

```
# Report and complaint to admin
class ReportToUserTestCase(TestCase):
    def setup(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='donor', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)
        self.report = Report.objects.create(food=self.food, complaint_to='complaint@example.com', descriptions='Test description', created_by=self.user)

    def test_report_update(self):
        data = {
            'food': self.food.id,
            'complaint_to': 'complaint_updated@example.com',
            'descriptions': 'Updated description',
            'created_by': self.user.id
        }
        request = self.factory.post('/api/user/report/', data)
        view = ReportToUser.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
```

Figure 111: Report to admin unit test success response

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

.
-----
Ran 1 test in 0.346s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 112: Report to admin unit test success function

4.2.1.6 Donation Rating

Test Case	Descriptions
Objectives	To develop the donation rating of the test case function
Actions	<p>To call the food donate completed then give the donation rating for API function.</p> <p>Food: food_id Complaint_to: complaint_update@example.com Descriptions: Update descriptions Created_by: user_id</p>
Expected Result	The complaint to admin details was successful without any errors. The response status code is 200, and the 'is_success' attribute is True.
Actual Result	The report to the admin unit test was a success after failure and resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 23: Donation Rating

```
# Donation Rating
class DonationCompletedRatingTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='volunteer', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)
        self.history = History.objects.create(food=self.food, volunteer=self.user)

    def test_donation_completed_rating(self):
        request = self.factory.patch('/api/donation/completed/rating/')
        data = [
            {
                'id': self.history.id,
                'descriptions': 'Test description',
                'location': 'Test location',
                'rating': 5
            }
        ]
        request.data = data
        view = DonationCompletedRating.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
```

Figure 113: Donation Rating unit testing function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_donation_completed_rating (foodshare.tests.DonationCompletedRatingTestCase.test_donation_completed_rating)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 243, in test_donation_completed_rating
    self.assertEqual(response.data['is_success'], True)
AssertionError: False != True

-----
Ran 1 test in 0.320s

FAILED (failures=1)
Destroying test database for alias 'default'...

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 114: Donation Rating unit testing failer response

Donation Rating success

```
# Donation Rating
class DonationCompletedRatingTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='volunteer', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)
        self.history = History.objects.create(food=self.food, volunteer=self.user)

    def test_donation_completed_rating(self):
        data = {
            'id': self.history.id,
            'descriptions': 'Test description',
            'location': 'Test location',
            'rating': 5
        }
        request = self.factory.patch('/api/donation/completed/rating/', data)
        view = DonationCompletedRating.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
```

Figure 115: Donation rating unit test function

```
FAILED (failures=1)
Destroying test database for alias 'default'...

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.

-----
Ran 1 test in 0.340s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>[]
```

Figure 116: Donation rating unit test failer response

4.2.1.7 The number of system ration

Test Case	Descriptions
Objectives	To develop the user email verified of the test case function
Actions	To call the number of system data then give the forgotten or ration of data for the API function.
Expected Result	The data ratio that the password update was successful without any errors. The response status code is 200, and the 'is_success' attribute is True, the status is 20, several food is 1, the number of users is 1, the several history is 1, and several reports is also 1.
Actual Result	The data ratio of the unit test was a success after failure and resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 24: The number of system ration

```
# Get number of data
class GetAllNumberOfDataViewTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='admin', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)
        self.history = History.objects.create(food=self.food, volunteer=self.user)

    def test_get_all_number_of_data(self):
        request = self.factory.get('/api/get/all/number/of/data/')
        view = GetAllNumberOfDataView.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['message'], "Success")
        self.assertEqual(response.data['data'][0]['food'], 1)
        self.assertEqual(response.data['data'][0]['user'], 1)
        self.assertEqual(response.data['data'][0]['history'], 1)
        self.assertEqual(response.data['data'][0]['report'], 1)
```

Figure 117: Number of system ration unit test function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_get_all_number_of_data (foodshare.tests.GetAllNumberOfDataViewTestCase.test_get_all_number_of_data)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 262, in test_get_all_number_of_data
    self.assertEqual(response.data['data']['report'], 1)
AssertionError: 0 != 1

Ran 1 test in 0.325s

FAILED (failures=1)
Destroying test database for alias 'default' ...
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 118: Number of system ration unit test failer response

```
class GetAllNumberOfDataViewTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role='admin', password='testpassword')
        self.food = Food.objects.create(food_name='Test Food', donor=self.user)
        self.history = History.objects.create(food=self.food, volunteer=self.user)
        self.report = Report.objects.create(food=self.food, complaint_to=self.user, descriptions='Test Report')
        self.device = Device.objects.create(token='test_token', user=self.user)

    def test_get_all_number_of_data(self):
        request = self.factory.get('/api/get/all/number/of/data/')
        view = GetAllNumberOfDataView.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['message'], "Success")
        self.assertEqual(response.data['data'][0]['food'], 1)
        self.assertEqual(response.data['data'][0]['user'], 1)
        self.assertEqual(response.data['data'][0]['history'], 1)
        self.assertEqual(response.data['data'][0]['report'], 1)
```

Figure 119: Number of r action unit test success functions

```

Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

.

-----
Ran 1 test in 0.349s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>

```

Figure 120: Number of ratio unit test success response

4.2.1.8 Email verify unit test

Test Case	Descriptions
Objectives	To develop the user email verified of the test case function
Actions	To call the email verify then give the forgotten or updated password for the API function. email: test@gmail.com
Expected Result	The email verified that the password update was successful without any errors. The response status code is 200, and the 'is_success' attribute is True.
Actual Result	The email verified the unit test was a success after failure and resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 25: Email verify unit test

```
# Email verify
class EmailVerifyTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role="donor", password='test')

    def test_email_verify_success(self):
        email = 'test@example.com'
        request = self.factory.get('/api/email/verify/', {'query': email})
        view = EmailVerify.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['is_success'], True)
```

Figure 121: Email Evrify unit test function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
=====
FAIL: test_email_verify_success (foodshare.tests.EmailVerifyTestCase.test_email_verify_success)
-----
Traceback (most recent call last):
  File "C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django\foodshare\tests.py", line 163, in test_email_verify_success
    self.assertEqual(response.data['is_success'], True)
AssertionError: False != True

-----
Ran 1 test in 0.387s

FAILED (failures=1)
Destroying test database for alias 'default'...
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 122: Email Evrify unit test failer response

Email Verify success

```
# Email verify
class EmailVerifyTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.user = Users.objects.create_user(username='testuser', email='test@example.com', role="donor", password='test')

    def test_email_verify_success(self):
        email = 'test@example.com'
        request = self.factory.get('/api/email/verify/', {'query': email})
        view = EmailVerify.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
```

Figure 123: Email verify success function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
-----
Ran 1 test in 0.317s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>[]
```

Figure 124: Email verify unit test success response

4.2.1.9 NOG Register Unit Test

Test Case	Descriptions
Objectives	To develop the register ngo of the test case function
Actions	To call the register ngo then give for the API function. ngo_name: Test_Ngo ngo_email: test@gmail.com ngo_contact: 1234567890
Expected Result	The new Ngo register was successful without any errors. The response status code is 200, and the 'is_success' attribute is True and more.
Actual Result	The new NGO registered the unit test as a success after failure and resolved the issues to ensure the test case was successful.
Test	The test was successful.

Table 26: NOG register unit test

```
class NgoProfileTestCase(TestCase):
    def setUp(self):
        self.factory = APIRequestFactory()
        self.ngo = Ngo.objects.create(ngo_name="Test NGO", ngo_email="test@example.com", ngo_contact="1234567890")

    def test_get ngo_profile(self):
        request = self.factory.get('/api/ngo-profile/')
        view = NgoProfile.as_view()
        response = view(request)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['message'], 'Success')
        self.assertEqual(response.data['is_success'], True)
        self.assertEqual(response.data['status'], 200)
        self.assertEqual(response.data['ngo'][ 'ngo_name'], self.ngo.ngo_name)
```

Figure 125: Ngo register unit test is a success function

```
(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>python manage.py test foodshare
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
-----
Ran 1 test in 0.013s

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

(env) C:\Documents\22015892_SITARAM_THING_FYP\Developments\Web_And_API\Django>
```

Figure 126: Ngo register unit test success response

4.2.2. System Testing

When I complete the development of the system, I will upload the wormhole server for 24 hours of free use. The app will be provided to both end-users and clients. The overall user interface part(UI) successfully accepts the end user with clients. After completing the development of the system, it was provided to both end-users and clients for system testing. The system testing was successful but identified several issues. End-user feedback indicated that the user interface (UI) and performance were satisfactory.

The system is now available for use, and users can also participate in a post-usage survey. Some UX and UI issues were encountered during testing, which are being addressed. The main issues identified during system testing were related to permissions, specifically regarding location, notifications, and user interaction when internet connectivity is unavailable. These issues are currently being resolved.

Web Application Testing

4.2.2.1 Login credential Admin account

Test Case	Descriptions
Objectives	To ensure the User can log in using credentials
Actions	The user enters the appropriate user's credentials on the login page and clicks the login button.
Expected Result	The user should be logged into the system and redirected to the home page.
Actual Result	The user was logged into the system and redirected to the home page.
Test	The test was successful.

Table 27: Web testing for login table

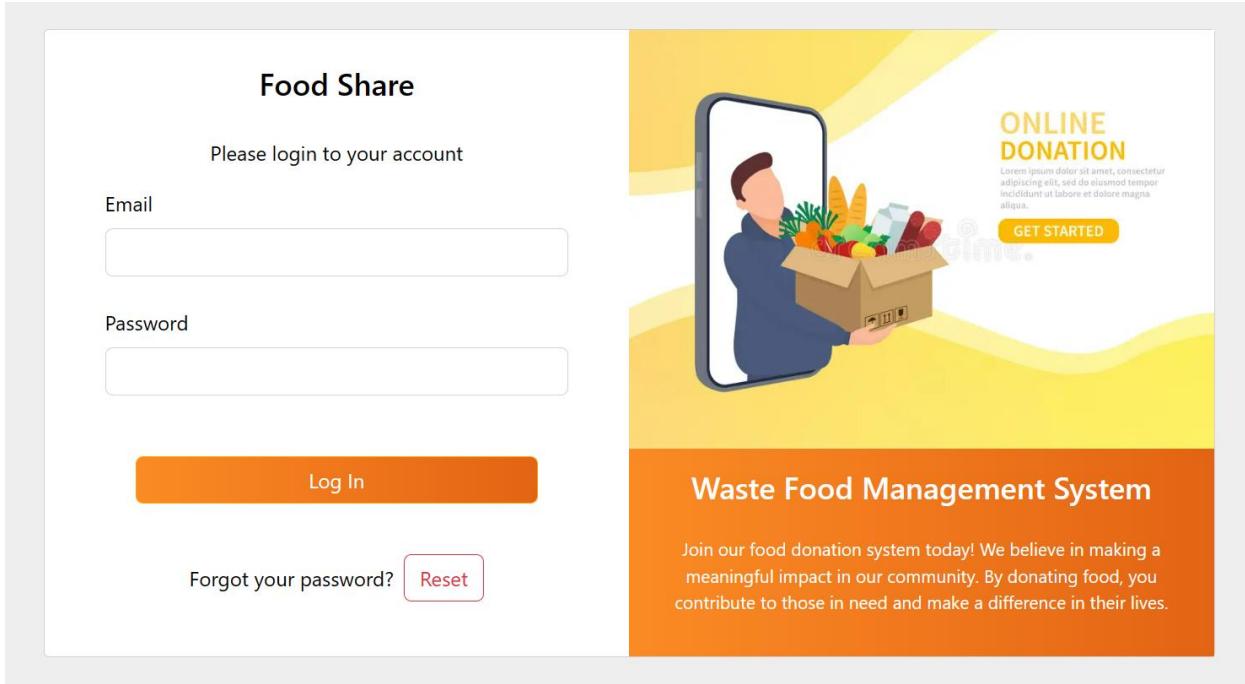


Table 28: Empty web login screen

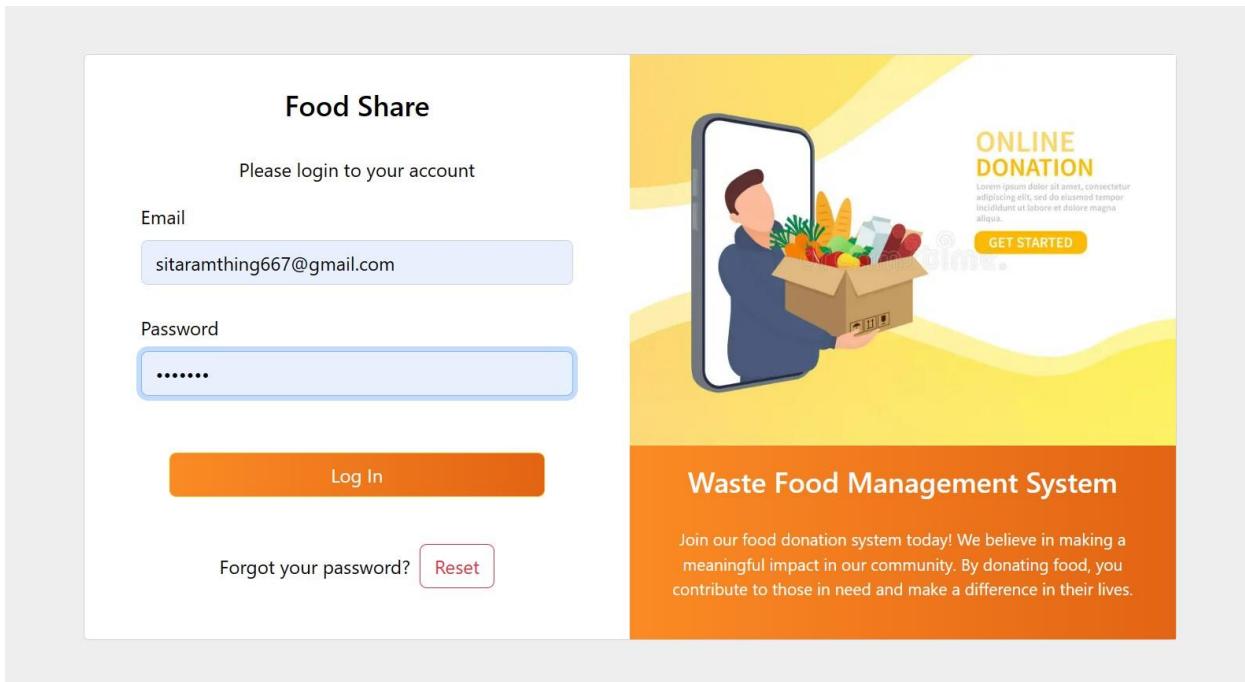


Figure 127: Login screen with filled the email and password

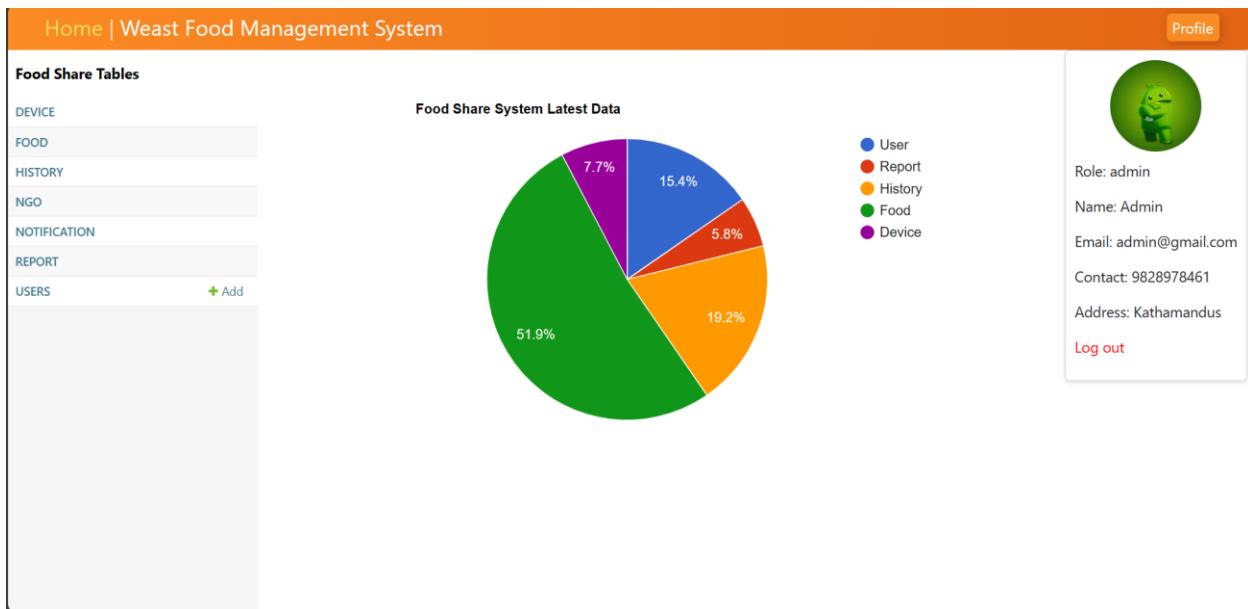


Figure 128: successfully navigate to the dashboard web admin panel

4.2.2.2 Active and registered users try to log in to the admin panel

Test Case	Descriptions
Objectives	To ensure the User can try registering an account login to the system in the admin panel.
Actions	The user enters a resisted account details and clicks the login button.
Expected Result	The form will display an error message at the bottom that states, “Invalid Credentials”.
Actual Result	The form displays an error message at the bottom that states, “Invalid Credentials”.
Test	The test was successful.

Table 29: Web testing to login screen in unauthenticated user

Data View for foodshare_users

ID	PASSWORD	LAST_LOGIN	EMAIL	USERNAME	ROLE	IS_ACTIVE	CREATED_BY	CREATED_DATE	MODIFY_BY	MODIFY_DATE
21	pbkdf2_sha256\$600000\$Kjpo...	March 17, 2024, 2:22 a.m.	sitaramthing667@gmail.com	Sitaram Thing	admin	1	Admin	March 13, 2024	Sitaram Thing	March 16, 2024
22	pbkdf2_sha256\$600000\$n3M...	Feb. 22, 2024, 6 p.m.	shrestha.sangam01@gmail.com	Sangam Shrestha	Donor	1	Admin	March 13, 2024	Admin	Dec. 13, 2023
24	pbkdf2_sha256\$600000\$FBIU...	None	volunteer@gmail.com	Ak	Volunteer	1	None	March 14, 2024	None	None
25	pbkdf2_sha256\$600000\$dVMf...	None	donor@gmail.com	Donor	Donor	1	None	March 16, 2024	None	None

Figure 129: Register data saved in a database

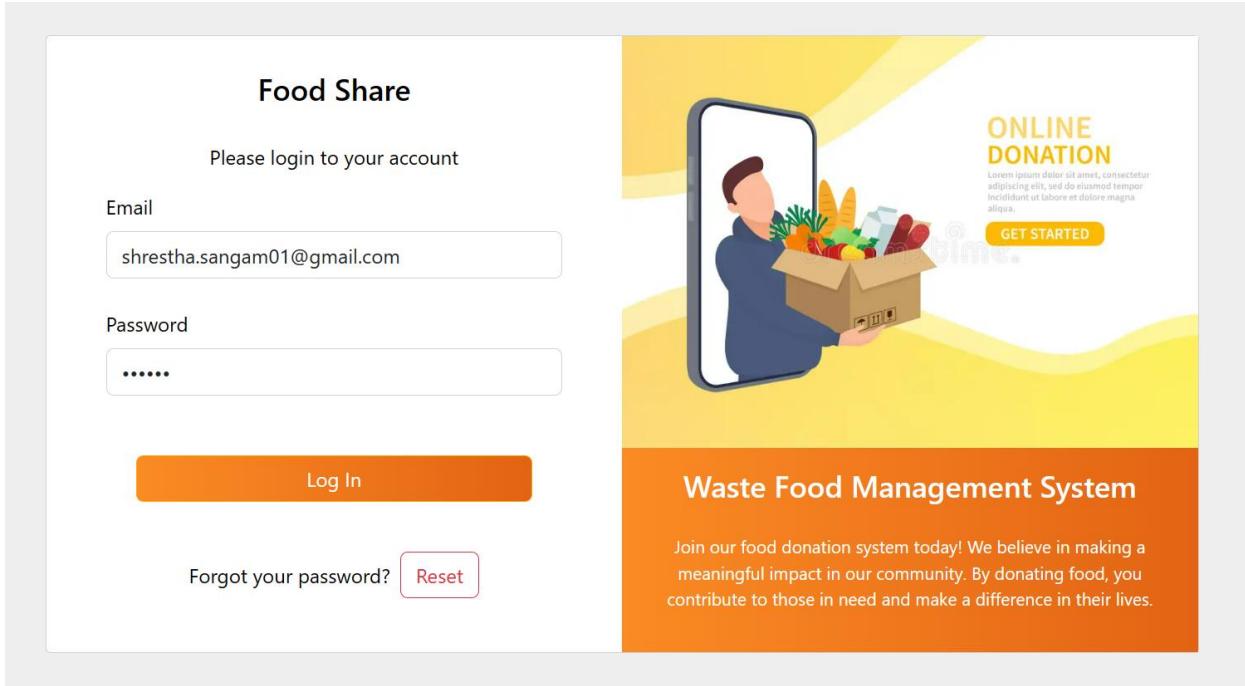


Figure 130: Web testing to fill in the login details

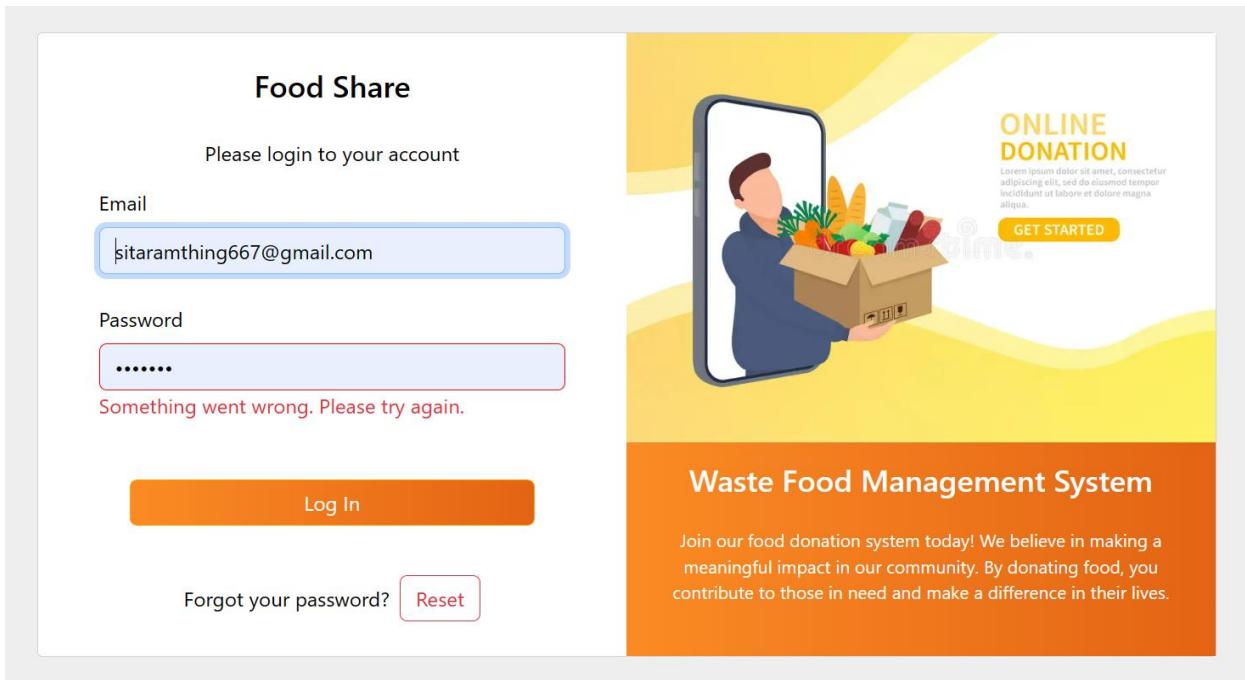


Figure 131: Display the Aunatheneticate user doesn't login to the web admin panel

4.2.2.3 Reset Password Page

Test Case	Descriptions
Objectives	To provide email and both password fields and redirect to the login page.
Actions	<p>The admin provides the email and password the valid email and same password then clicks the confirm button.</p> <p>Update password details:</p> <ul style="list-style-type: none"> • Email: sitaramthing667@gmail.com • New Passswoed: sitaram • Confirm Password: sitaram
Expected Result	The password is successfully reset and navigate to the login screen.
Actual Result	The password is successfully reset and without any exception redirected to the login page.
Test	The test was successful.

Table 30: Web testing register user table

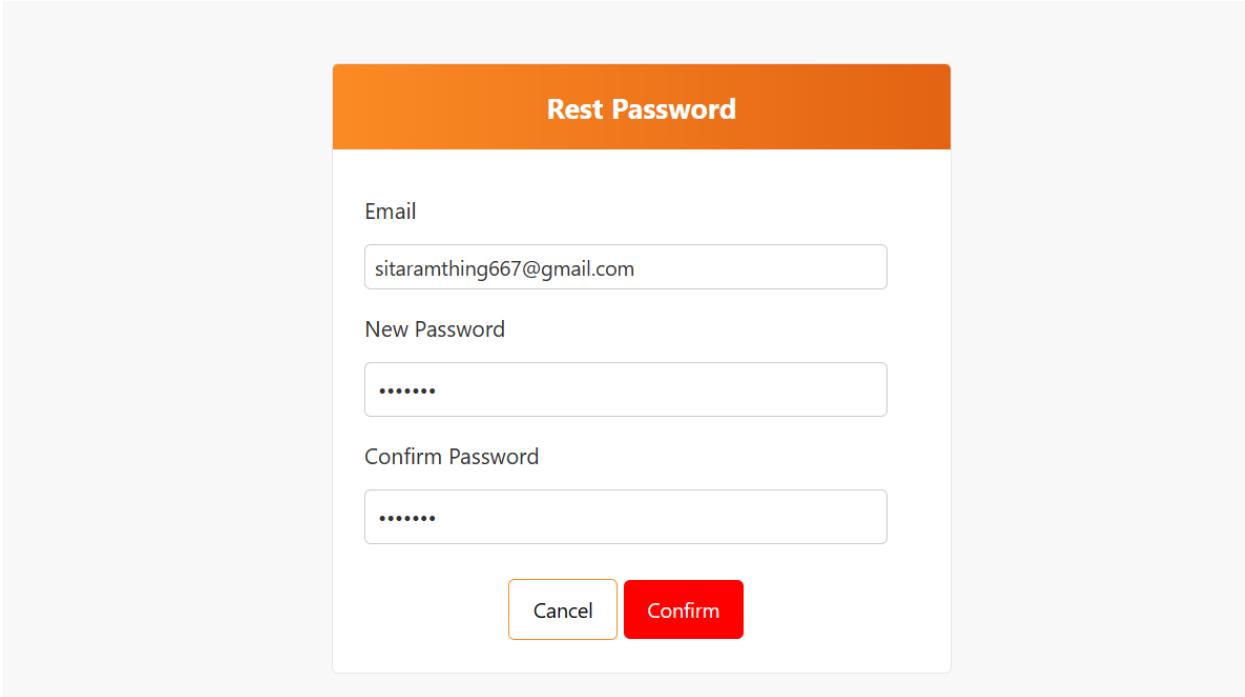


Table 31: Web test in enter the email, password and confirm password

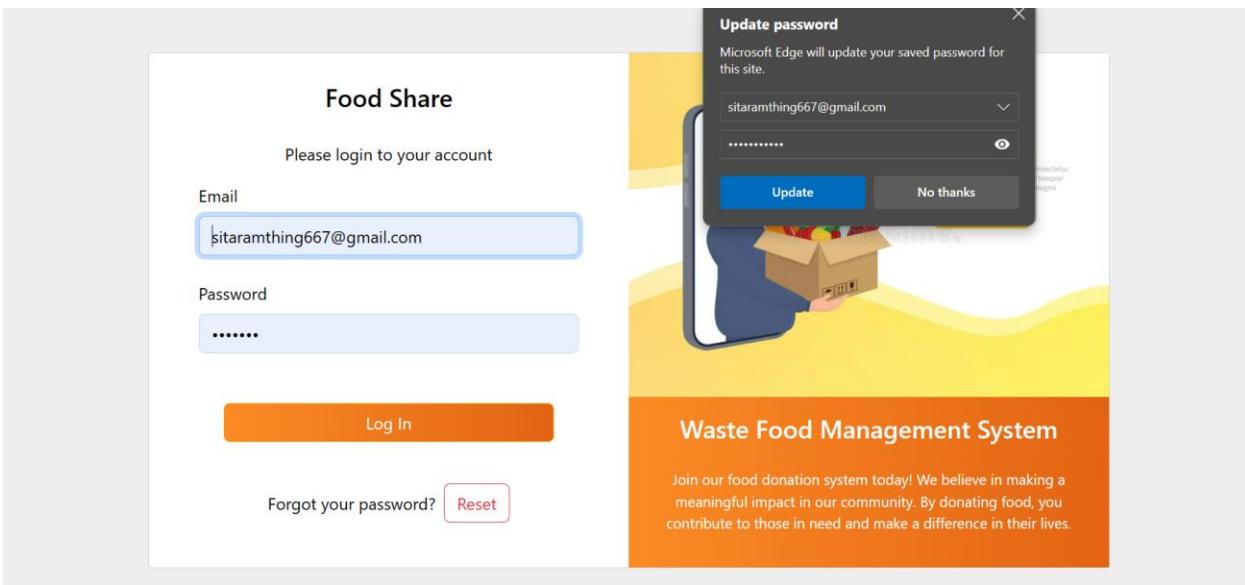


Table 32: Web test update password success to navigate to the login page

4.2.2.4 View History in Paichart (Home Page):

Test Case	Descriptions
Objectives	To navigate the home page hover or click to view the ratio of data in percentage.
Actions	Open the home page and click or hover over the pie chart.
Expected Result	The pie chart should display different types of data. Clicking or hovering over it should display the percentage of each data type.
Actual Result	The home page successfully displayed the pie chart with data and the percentage ratio.
Test	The test was successful.

Table 33: Test for the home page has a history in pie-chart

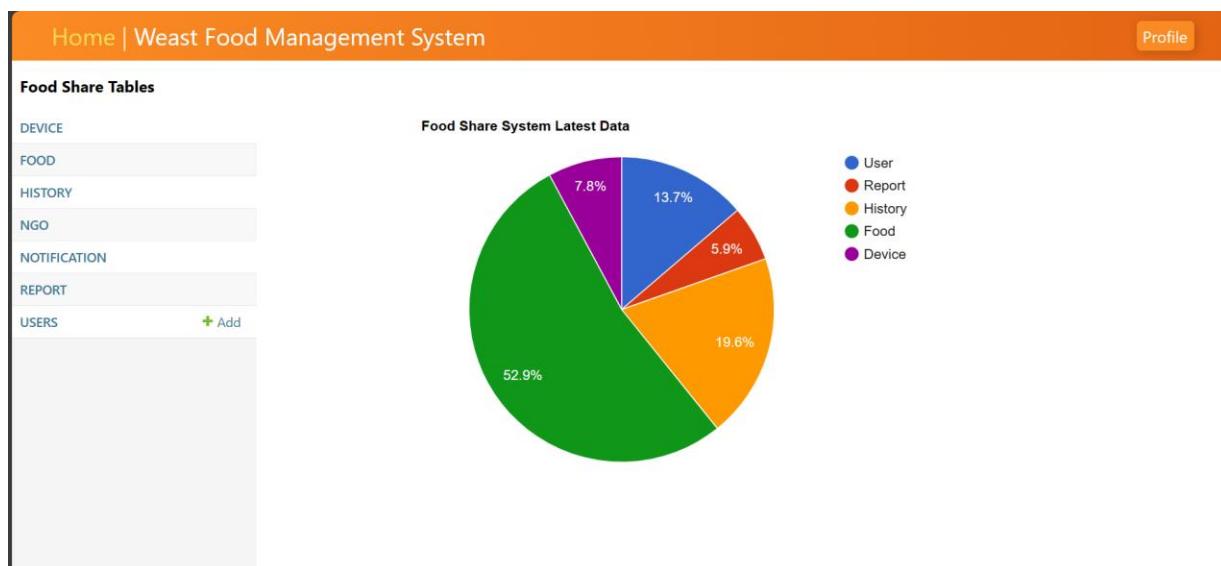


Figure 132: Display the piechart with data

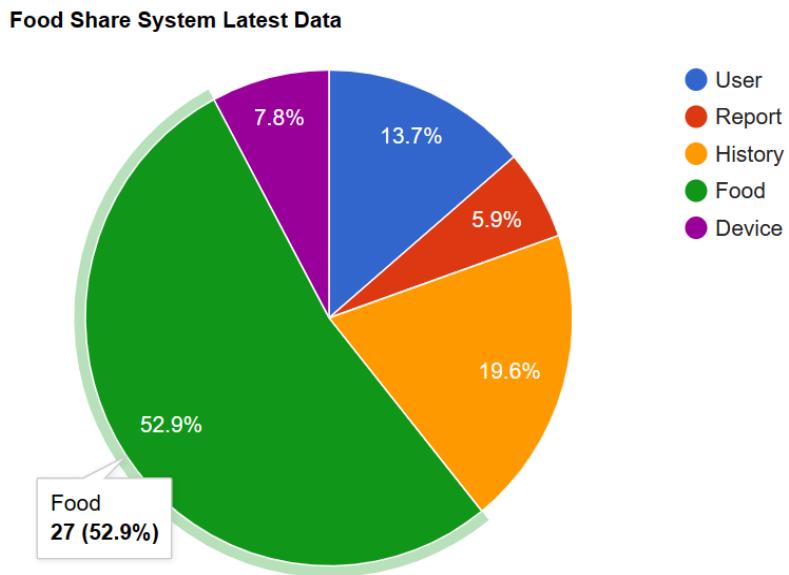


Figure 133: Display the percentage of food history

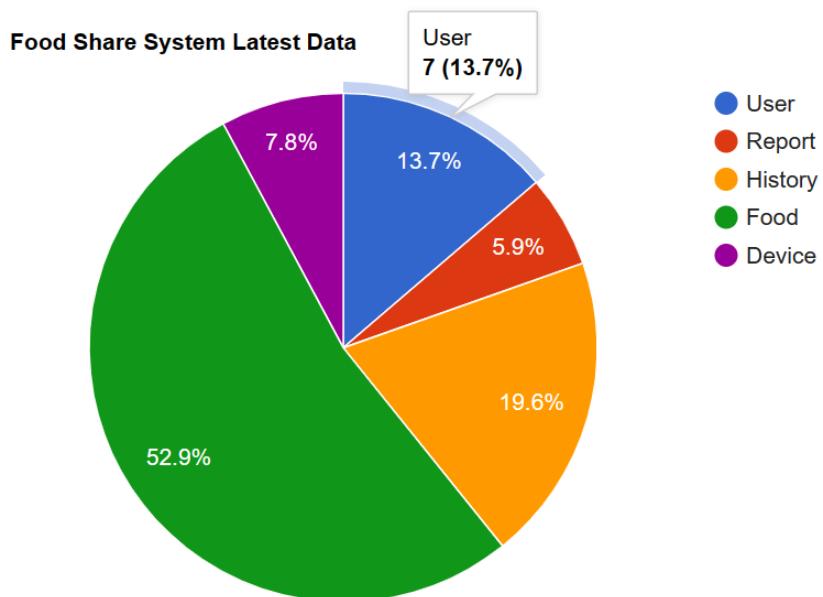


Figure 134: Display the percentage of food user

4.2.2.5 View history data

Test Case	Descriptions
Objectives	Navigate to the data view screen and display all table details in a table format.
Actions	Click on the table name in the sidebar to display the data and view it.
Expected Result	Clicking on the table name in the sidebar should successfully display the data within the table.
Actual Result	The data is successfully displayed in a table format.
Test	The test was successful.

Table 34: Web testing for View history tables

Food Share Tables

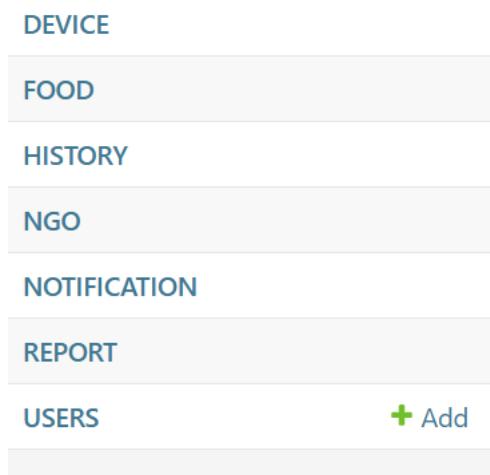
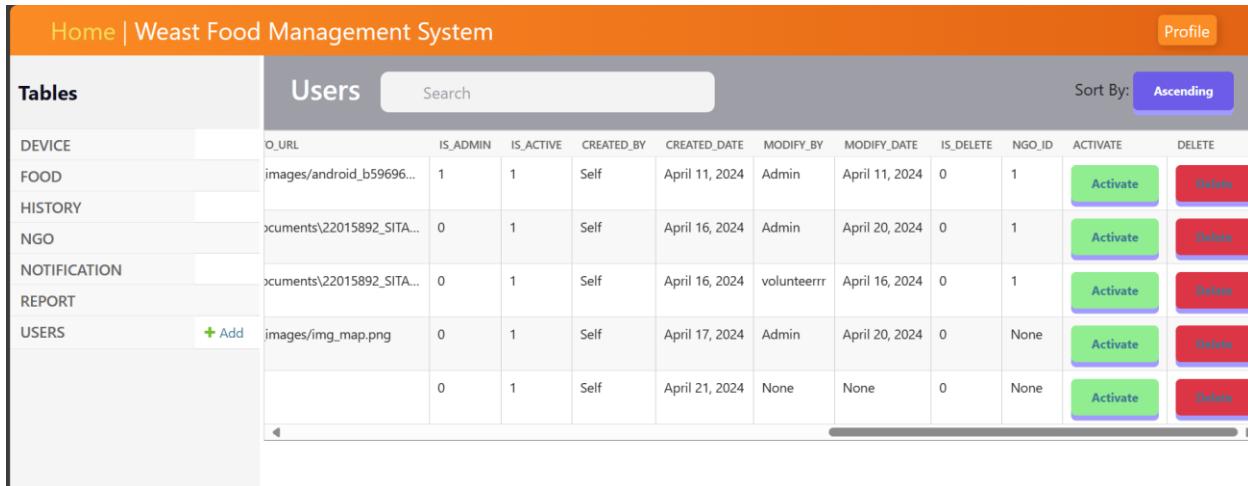


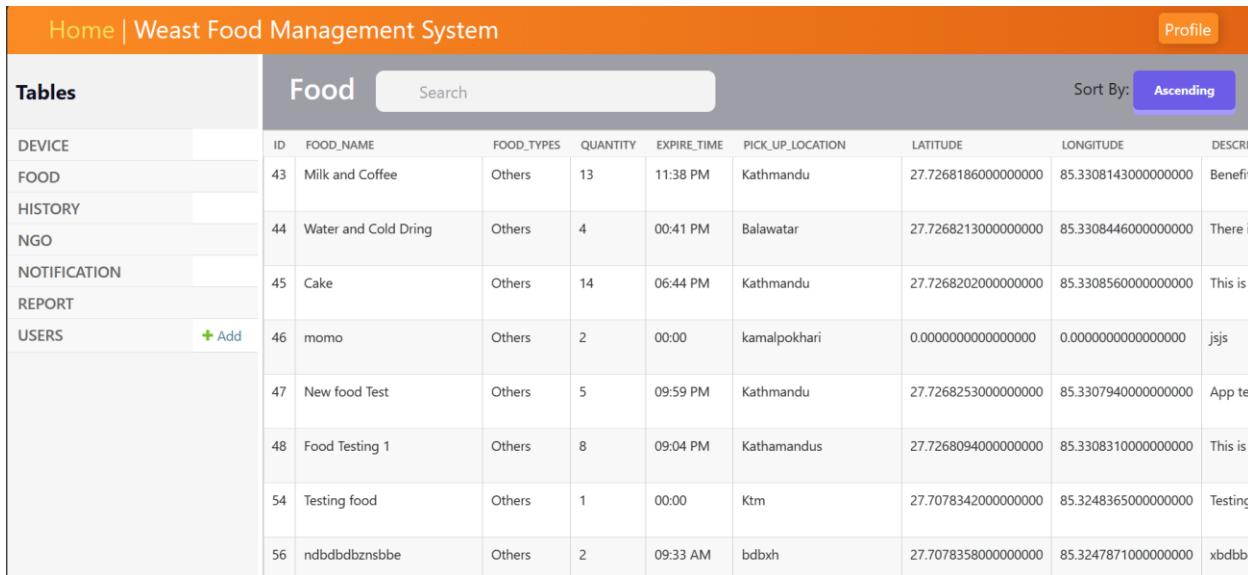
Figure 135: Web testing for history table list



The screenshot shows a web-based application interface for managing food users. At the top, there's a header bar with the title "Home | Weast Food Management System" and a "Profile" button. Below the header is a sidebar titled "Tables" containing links to DEVICE, FOOD, HISTORY, NGO, NOTIFICATION, REPORT, and USERS. The main content area is titled "Users" and contains a table with columns: O_URL, IS_ADMIN, IS_ACTIVE, CREATED_BY, CREATED_DATE, MODIFY_BY, MODIFY_DATE, IS_DELETE, NGO_ID, ACTIVATE, and DELETE. There are two rows of data in the table, each with a "Search" input field and "Activate" and "Delete" buttons.

Users											
Search Sort By: Ascending											
DEVICE	O_URL	IS_ADMIN	IS_ACTIVE	CREATED_BY	CREATED_DATE	MODIFY_BY	MODIFY_DATE	IS_DELETE	NGO_ID	ACTIVATE	DELETE
FOOD	images/android_b59696...	1	1	Self	April 11, 2024	Admin	April 11, 2024	0	1	<button>Activate</button>	<button>Delete</button>
HISTORY	documents\22015892_SITA...	0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>
NGO	documents\22015892_SITA...	0	1	Self	April 16, 2024	volunteerr	April 16, 2024	0	1	<button>Activate</button>	<button>Delete</button>
NOTIFICATION	documents\22015892_SITA...	0	1	Self	April 17, 2024	Admin	April 20, 2024	0	None	<button>Activate</button>	<button>Delete</button>
REPORT	images/img_map.png	0	1	Self	April 21, 2024	None	None	0	None	<button>Activate</button>	<button>Delete</button>
USERS	+ Add										
		0	1	Self	April 21, 2024	None	None	0	None	<button>Activate</button>	<button>Delete</button>

Figure 136: View history of user details



The screenshot shows a web-based application interface for managing food items. At the top, there's a header bar with the title "Home | Weast Food Management System" and a "Profile" button. Below the header is a sidebar titled "Tables" containing links to DEVICE, FOOD, HISTORY, NGO, NOTIFICATION, REPORT, and USERS. The main content area is titled "Food" and contains a table with columns: ID, FOOD_NAME, FOOD_TYPES, QUANTITY, EXPIRE_TIME, PICK_UP_LOCATION, LATITUDE, LONGITUDE, and DESCRIPT. There are ten rows of data in the table, each with a "Search" input field and "Activate" and "Delete" buttons.

Food								
Search Sort By: Ascending								
DEVICE	ID	FOOD_NAME	FOOD_TYPES	QUANTITY	EXPIRE_TIME	PICK_UP_LOCATION	LATITUDE	LONGITUDE
FOOD	43	Milk and Coffee	Others	13	11:38 PM	Kathmandu	27.7268186000000000	85.3308143000000000
HISTORY	44	Water and Cold Dring	Others	4	00:41 PM	Balawatar	27.7268213000000000	85.3308446000000000
NGO	45	Cake	Others	14	06:44 PM	Kathmandu	27.7268202000000000	85.3308560000000000
NOTIFICATION	46	momo	Others	2	00:00	kamalpokhari	0.0000000000000000	0.0000000000000000
REPORT	47	New food Test	Others	5	09:59 PM	Kathmandu	27.7268253000000000	85.3307940000000000
USERS	48	Food Testing 1	Others	8	09:04 PM	Kathamandus	27.7268094000000000	85.3308310000000000
	54	Testing food	Others	1	00:00	Ktm	27.7078342000000000	85.3248365000000000
	56	ndbdbdbznsbbe	Others	2	09:33 AM	bdbxh	27.7078358000000000	85.3247871000000000

Figure 137: View the history of food details

4.2.2.6 Delete Item

Test Case	Descriptions
Objectives	The Admin can verify the data and delete unnecessary data by clicking the delete button.
Actions	Find the unnecessary data, click the delete button, and delete it temporarily, but not permanently, from the database.
Expected Result	The data should be successfully deleted from the web interface, and the deleted data should not be visible on the front end.
Actual Result	The data was successfully deleted from the front end.
Test	The test was successful.

Table 35: Web testing for deleting table details

Tables		Notification						
		Search		Sort By: Ascending				
ID	TITLE	DESCRIPTIONS	CREATED_BY	CREATED_DATE	IS_DELETE	FOOD_ID	DELETE	
1	jsnehd	b,hsbsvsv	Donor 1khihib	April 18, 2024	0	57	<button>Delete</button>	
3	Nesting box	kdjd	Donor 1khihib	April 18, 2024	0	59	<button>Delete</button>	
4	Hello bro	jhdbd	Donor 1khihib	April 18, 2024	0	60	<button>Delete</button>	
5	hshsb	ddnhdb	Donor 1khihib	April 18, 2024	0	61	<button>Delete</button>	
6	Testing notice	new notification	Donor 1khihib	April 18, 2024	0	62	<button>Delete</button>	
11	jbiviv	nvjvuv	Donor 1khihib	April 20, 2024	0	67	<button>Delete</button>	
12	New Food Updated foyiu	Also available	Food Donor	April 21, 2024	0	68	<button>Delete</button>	
13	Food donatw	food donate	Food Donor	April 21, 2024	0	69	<button>Delete</button>	

Figure 138: Web test for deleting item show

Home Weast Food Management System								
Tables		Notification			Search			Sort By: Ascending
ID	TITLE	DESCRIPTIONS	CREATED_BY	CREATED_DATE	IS_DELETE	FOOD_ID	DELETE	
DEVICE								
FOOD								
HISTORY								
NGO								
NOTIFICATION								
REPORT								
USERS	<button>+ Add</button>							

Figure 139: Web test delete an item not show

4.2.2.7 Register a user account

Test Case	Descriptions
Objectives	To add new user details and register successfully.
Actions	Open the registration screen, enter the username, email, password, and role, and then click the register button.
Expected Result	Result: The admin successfully adds a new user without any errors.
Actual Result	The user account was successfully registered.
Test	The test was successful.

Table 36: Web test for adding new user table

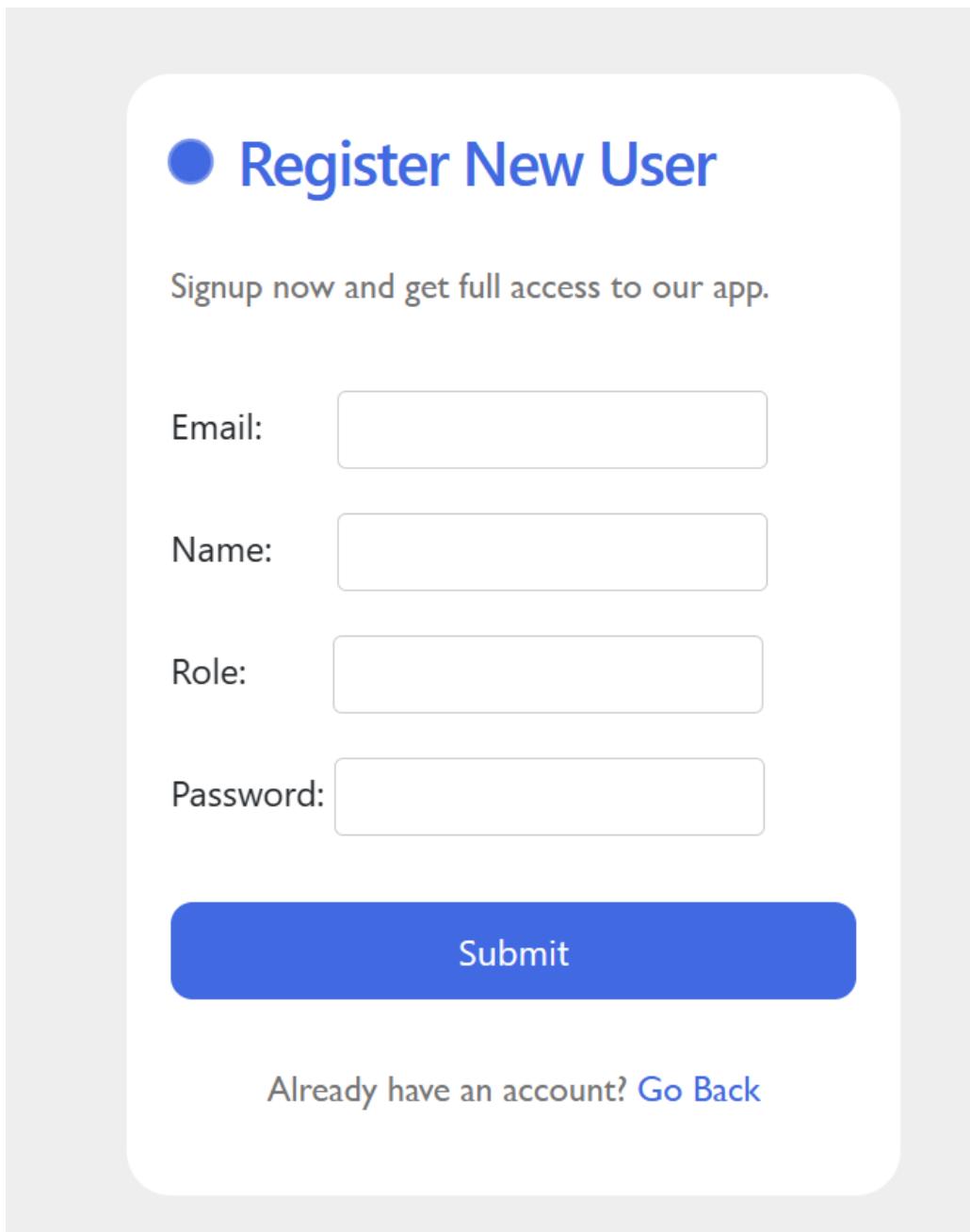


Figure 140: New user register empty register screen

Register New User

Signup now and get full access to our app.

Email: daimond@gmail.com

Name: Daimand Nepal

Role: donor

Password:

Submit

Already have an account? [Go Back](#)

Figure 141: Web testing to enter the user details to register

Home | Weast Food Management System

Tables

ID	PASSWORD	LAST_LOGIN	EMAIL	USERNAME	ROLE	ADDRESS	CONTACT_NUMBER	GENDER	DATE_OF_BIRTH
36	pbkdf2_sha256\$600000\$5oHb...	None	daimond@gmail.com	Daimand Nepal	donor	None	None	None	April 22, 2024
37	pbkdf2_sha256\$600000\$5XZT...	None	test@test.com	test	Volunteer	None	None	None	April 21, 2024
21	pbkdf2_sha256\$600000\$HXjN...	None	n@gmail.com	laxman	Donor	Manabg	123456	Male	April 17, 2024
14	pbkdf2_sha256\$600000\$Kgs...	None	volunteer@gmail.com	volunteerr	Volunteer	dvdv	987654321	Others	April 16, 2024
13	pbkdf2_sha256\$600000\$ofIBB...	April 20, 2024, 7:46 p.m.	donor@gmail.com	Food Donor	Donor	Admin	9800000000	Others	April 16, 2024
1	pbkdf2_sha256\$600000\$B2jm...	April 20, 2024, 4:12 p.m.	admin@gmail.com	Admin	admin	Kathmandus	9828978461	Others	April 11, 2024

Figure 142: Web testing to register user display in a table

4.2.2.8 Activate user account

Test Case	Descriptions
Objectives	The Admin can find the newly registered user and activate their user account
Actions	Find the new user and verify their details, then activate their user account by clicking the "activate" button.
Expected Result	Clicking the "activate" button should successfully activate the user account.
Actual Result	The user account was successfully activated without any exceptions.
Test	The test was successful.

Table 37: Web testing for user account activate table

Tables	Users										Sort By: Descending	
	PHOTO_URL	IS_ADMIN	IS_ACTIVE	CREATED_BY	CREATED_DATE	MODIFY_BY	MODIFY_DATE	IS_DELETE	NGO_ID	ACTIVATE	DELETE	
DEVICE		0	0	Self	April 22, 2024	None	None	0	None	<button>Activate</button>	<button>Delete</button>	
FOOD		0	1	Self	April 21, 2024	None	None	0	None	<button>Activate</button>	<button>Delete</button>	
HISTORY		0	1	Self	April 17, 2024	Admin	April 20, 2024	0	None	<button>Activate</button>	<button>Delete</button>	
NGO		0	1	Self	April 16, 2024	volunteerr	April 16, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
NOTIFICATION		0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
REPORT		0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
USERS	+ Add	0	1	Self	April 11, 2024	Admin	April 11, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
	donor.	C:\Documents\22015892_SITA...	0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>
	istrator of food ...	user_images/android_b59696...	1	1	Self	April 11, 2024	Admin	April 11, 2024	0	1	<button>Activate</button>	<button>Delete</button>

Figure 143: Web testing shows the inactive account

Tables	Users										Sort By: Descending	
	PHOTO_URL	IS_ADMIN	IS_ACTIVE	CREATED_BY	CREATED_DATE	MODIFY_BY	MODIFY_DATE	IS_DELETE	NGO_ID	ACTIVATE	DELETE	
DEVICE		0	1	Self	April 22, 2024	None	None	0	None	<button>Activate</button>	<button>Delete</button>	
FOOD		0	1	Self	April 21, 2024	None	None	0	None	<button>Activate</button>	<button>Delete</button>	
HISTORY		0	1	Self	April 17, 2024	Admin	April 20, 2024	0	None	<button>Activate</button>	<button>Delete</button>	
NGO		0	1	Self	April 16, 2024	volunteerr	April 16, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
NOTIFICATION		0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
REPORT		0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
USERS	+ Add	0	1	Self	April 11, 2024	Admin	April 11, 2024	0	1	<button>Activate</button>	<button>Delete</button>	
	donor.	C:\Documents\22015892_SITA...	0	1	Self	April 16, 2024	Admin	April 20, 2024	0	1	<button>Activate</button>	<button>Delete</button>
	istrator of food ...	user_images/android_b59696...	1	1	Self	April 11, 2024	Admin	April 11, 2024	0	1	<button>Activate</button>	<button>Delete</button>

Figure 144: Web testing user account is successfully activated

4.2.2.9 Logout administrator

Test No	Descriptions
Objectives	To verify that the administrator can successfully log out of the system.
Actions	Click the logout button and successfully navigate to the login page.
Expected Result	The administrator is successfully logged out of the system.
Actual Result	The administrator was successfully logged out.
Test	The test was successful.

Table 38: Web Logout testing table

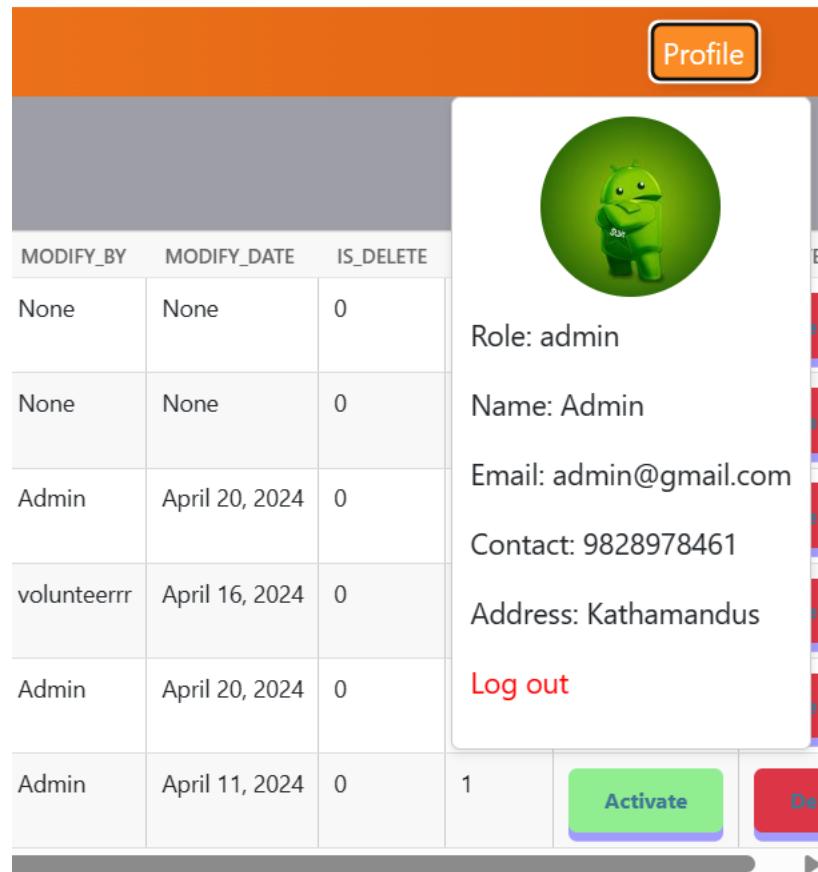


Figure 145: Logout view card with logout button

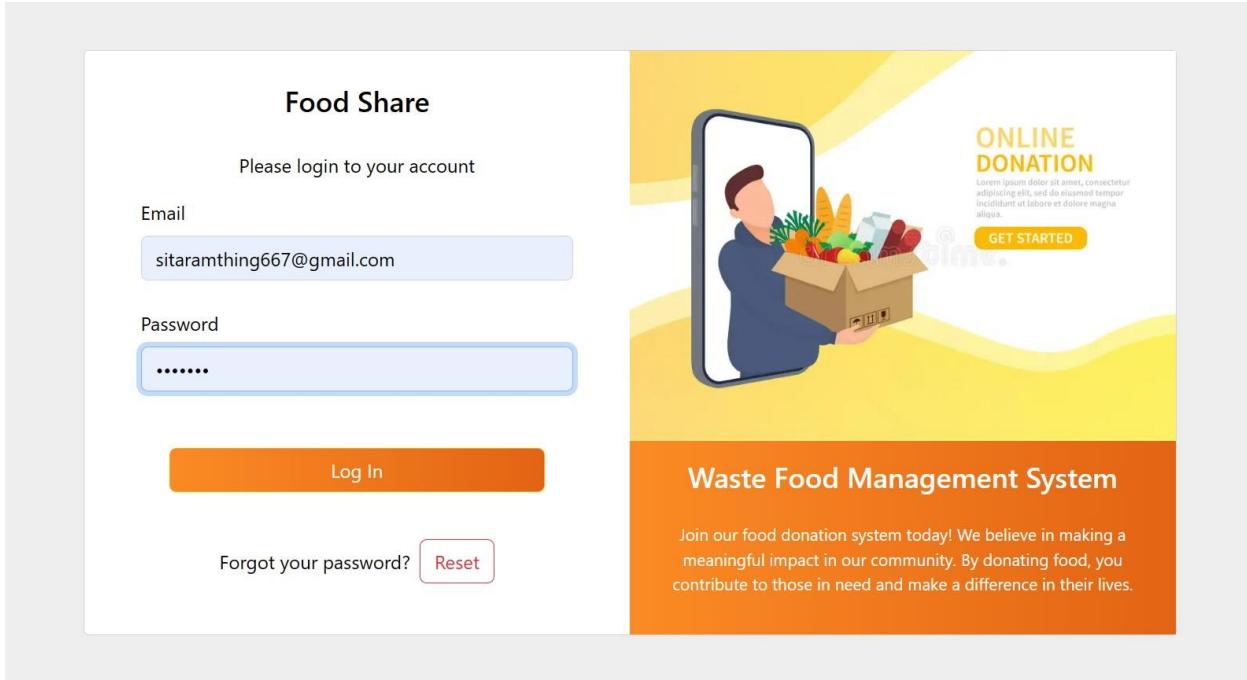


Figure 146: Web logout testing to successfully navigate the login page

Mobile Application Testing

4.2.2.10 Registered user

Test Case	Descriptions
Objectives	To ensure that the user can successfully register membership details to access the system.
Actions	The user enters new and valid registration details and clicks the register button. Email: Shrestha.sangam01@gamil.com Username: Sangam Shrestha Role: Donor Password: Sangam
Expected Result	The user successfully registers and is redirected to the login page.
Actual Result	The user is successfully registered and redirected to the login page.
Test	The test was successful.

Table 39: Registered user test in mobile app

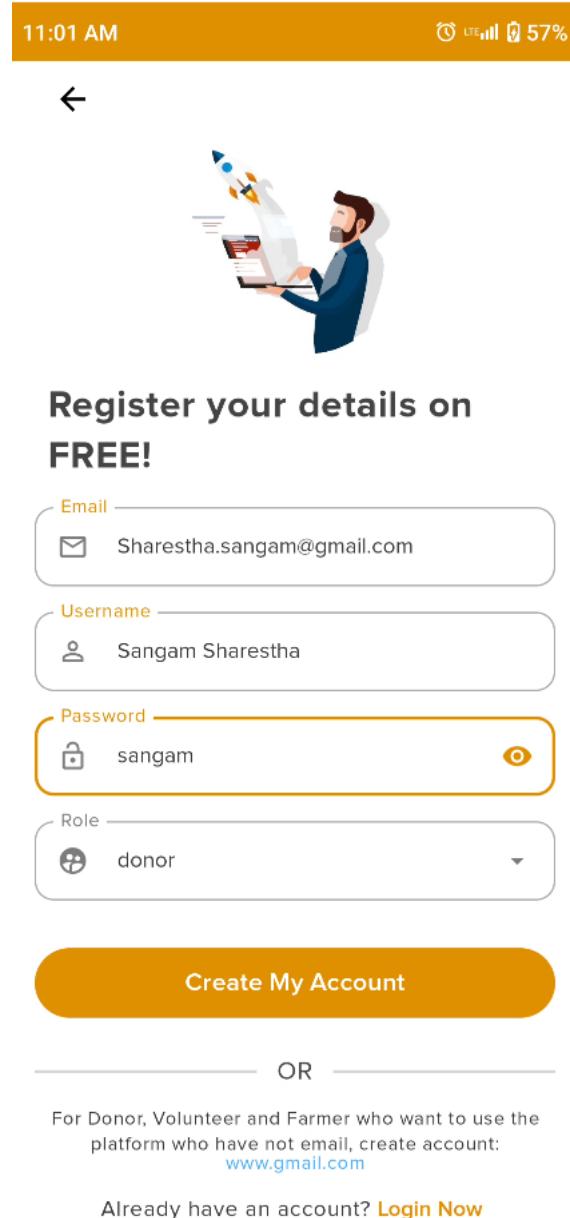


Figure 147: Register user filled in the user details in the mobile app

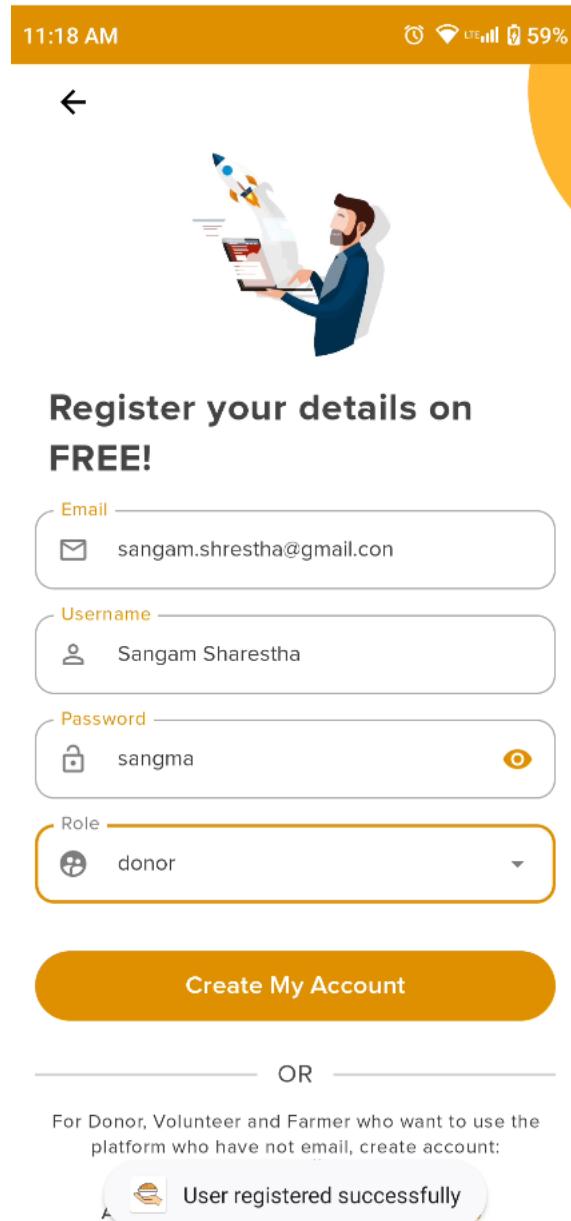


Figure 148: Register user success in the mobile app

Data View for foodshare_users

ID	PASSWORD	LAST_LOGIN	EMAIL	USERNAME	ROLE	IS_ACTIVE	CREATED_BY	CREATED_DATE	MODIFY_BY	MODIFY_DATE
21	pbkdf2_sha256\$600000\$KJpo...	March 17, 2024, 2:22 a.m.	sitaramthing667@gmail.com	Sitaram Thing	admin	1	Admin	March 13, 2024	Sitaram Thing	March 16, 2024
22	pbkdf2_sha256\$600000\$n3M...	Feb. 22, 2024, 6 p.m.	shrestha.sangam01@gmail.com	Sangam Shrestha	Donor	1	Admin	March 13, 2024	Admin	Dec. 13, 2023
24	pbkdf2_sha256\$600000\$FBIU...	None	volunteer@gmail.com	Ak	Volunteer	1	None	March 14, 2024	None	None
25	pbkdf2_sha256\$600000\$dVMf...	None	donor@gmail.com	Donor	Donor	1	None	March 16, 2024	None	None

Figure 149: Register user details in the database

4.2.2.11 Login inactivate the user account

Test Case	Descriptions
Objectives	To ensure the User can log in using uncredentials with valid login details.
Actions	The user enters the appropriate user's credentials on the login details and clicks the login button. <ul style="list-style-type: none"> ➤ Email: donor@gmail.com ➤ Password: donor
Expected Result	The user should be logged into the system and show the error message in the dialogue box.
Actual Result	The user logged in to an unauthenticated account and did not succeed in logging into the system.
Test	The test was successful.

Table 40: Login credential user account test in the mobile app

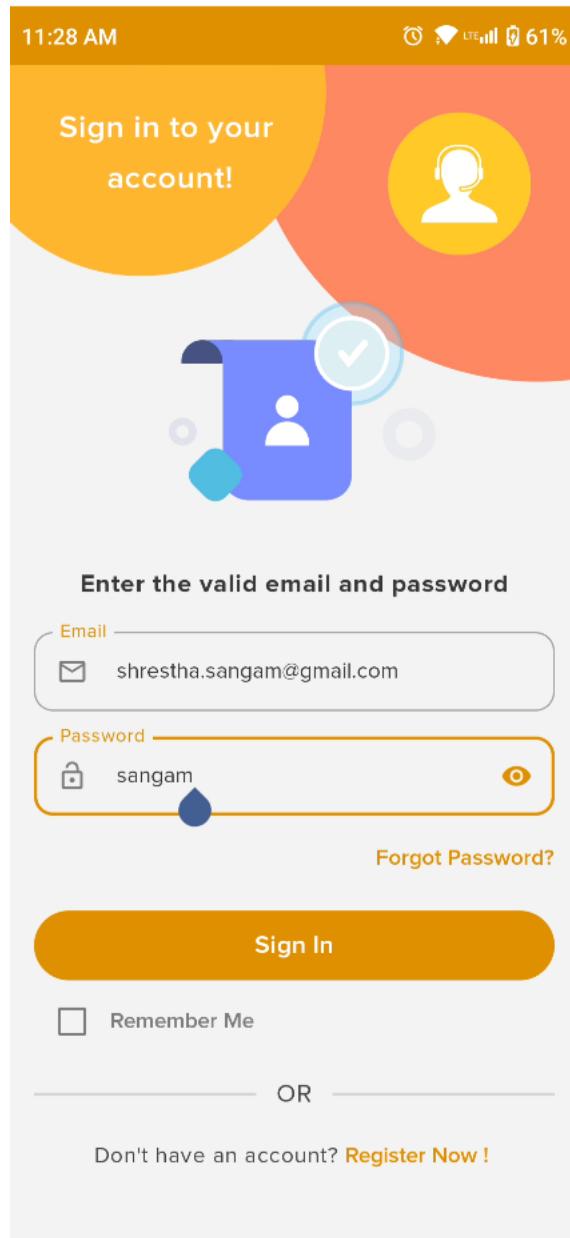


Figure 150: Login to deactivate the user account

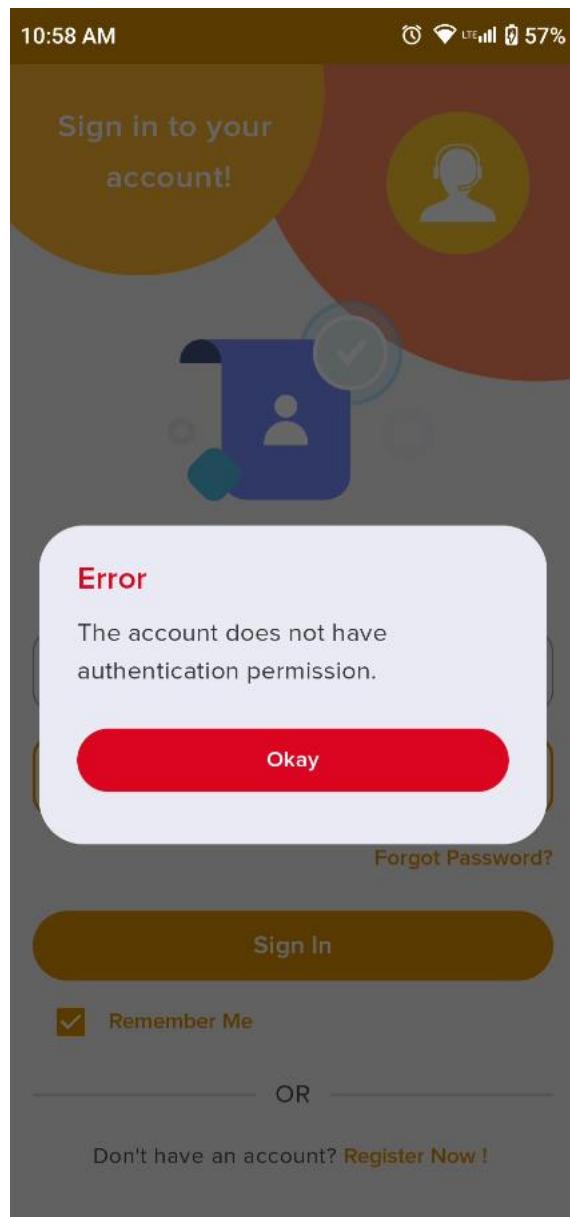


Figure 151: show the unauthenticated error message in the dialogue box

4.2.2.12 Login credential user account

Test Case	Descriptions
Objectives	To ensure the User can log in using credentials with valid login details.
Actions	The user enters the appropriate user's credentials on the login details and clicks the login button. <ul style="list-style-type: none"> ➤ Email: donor@gmail.com ➤ Password: donor
Expected Result	The user should be logged into the system and successfully navigate to the home page without any exceptions.
Actual Result	The user login is successfully authenticated to log into the system
Test	The test was successful.

Table 41: Login credential user account test in the mobile app

Users														Sort By Ascending
EMAIL	USERNAME	ROLE	ADDRESS	CONTACT_NUMBER	GENDER	DATE_OF_BIRTH	ABOUTS_USER	PHOTO_URL	IS_ADMIN	IS_ACTIVE	CREATED_BY	CREATED_DATE		
admin@gmail.com	Admin	admin	Kathmandus	9828978461	Others	April 11, 2024	We are administrator of food ...	user_images/android_b59696...	1	1	Self	April 11, 2024		
donor@gmail.com	Food Donor	Donor	Admin	9800000000	Others	April 16, 2024	I am a food donor.	C:\Documents\22015892_SITA...	0	1	Self	April 16, 2024		
volunteer@gmail.com	volunteerr	Volunteer	dvdv	987654321	Others	April 16, 2024	sdvsdv	C:\Documents\22015892_SITA...	0	1	Self	April 16, 2024		
n@gmail.com	laxman	Donor	Manabg	123456	Male	April 17, 2024	adad	user_images/img_map.png	0	1	Self	April 17, 2024		
daimond@gmail.com	Daimand Nepal	donor	None	None	None	April 22, 2024	None		0	1	Self	April 22, 2024		
sita@gmail.com	Sita	volunteer	None	None	None	April 22, 2024	None		0	0	Self	April 22, 2024		
sangam.shrestha@gmail.com	Sangam Sharetha	donor	None	None	None	April 23, 2024	None		0	1	Self	April 23, 2024		

Figure 152: Activate the user account from the admin

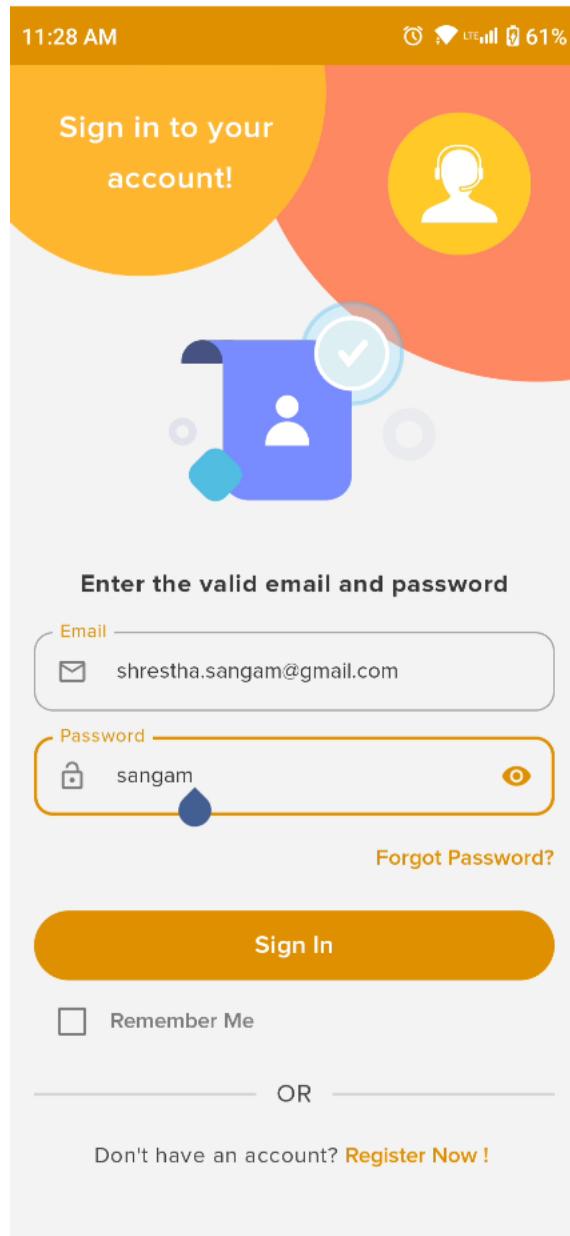


Figure 153: Login to activate the user account

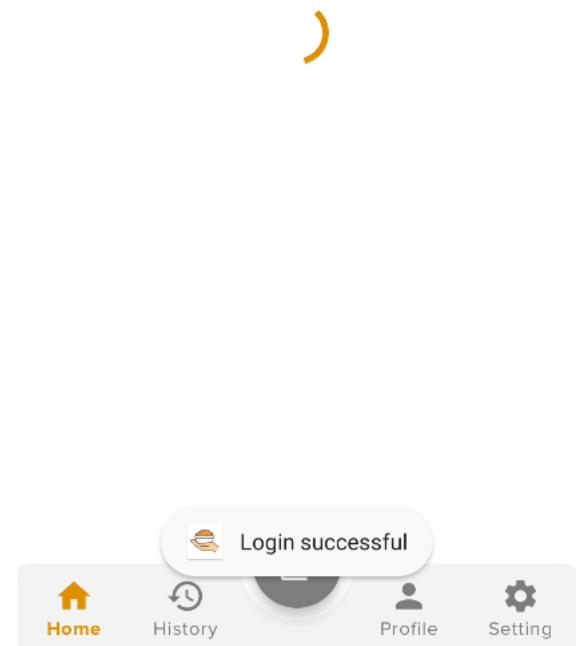


Figure 154: Login success message

4.2.2.13 Forgot Password

Test Case	Descriptions
Objectives	To provide an email to verify the active user enter both password fields and change the password.
Actions	The admin provides the email clicks the email verify button and after successfully verified enters the new password confirms the password is the same then clicks the confirm button.
Expected Result	The password is successfully reset and shows the success message.
Actual Result	The password is successfully reset and shows the success message and notification.
Test	The test was successful.

Table 42: Forgot Password test in mobile app

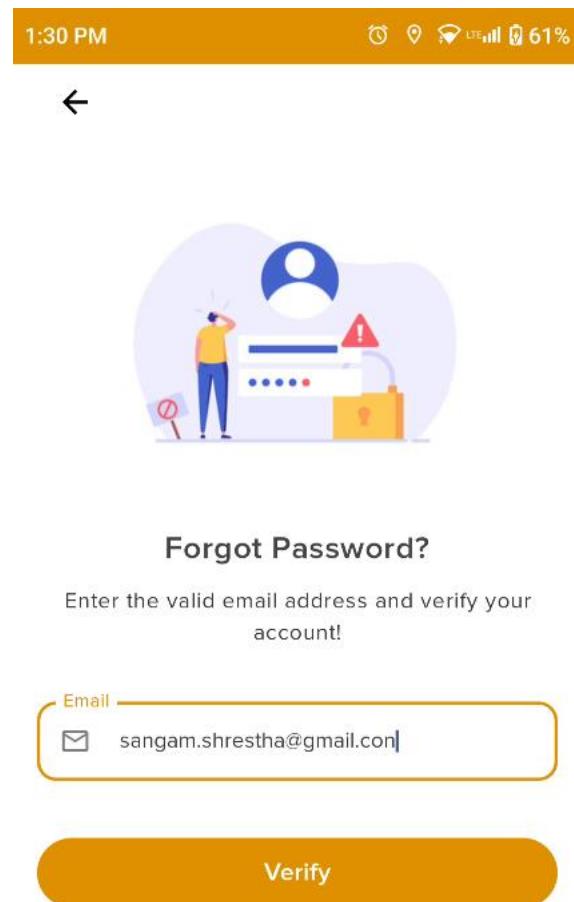


Figure 155: Enter the valid email address to verify the user

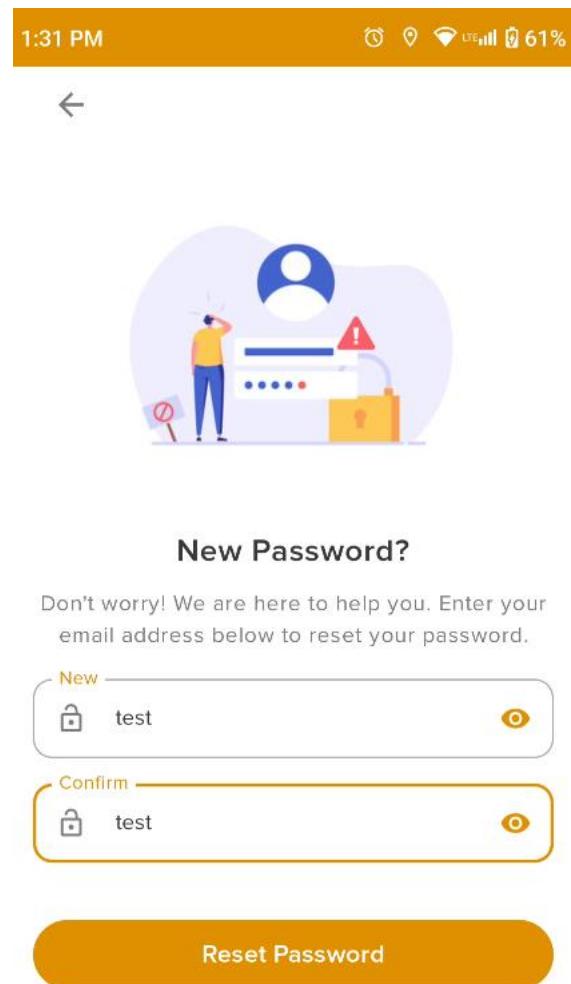


Figure 156: Enter the new password and valid both same password

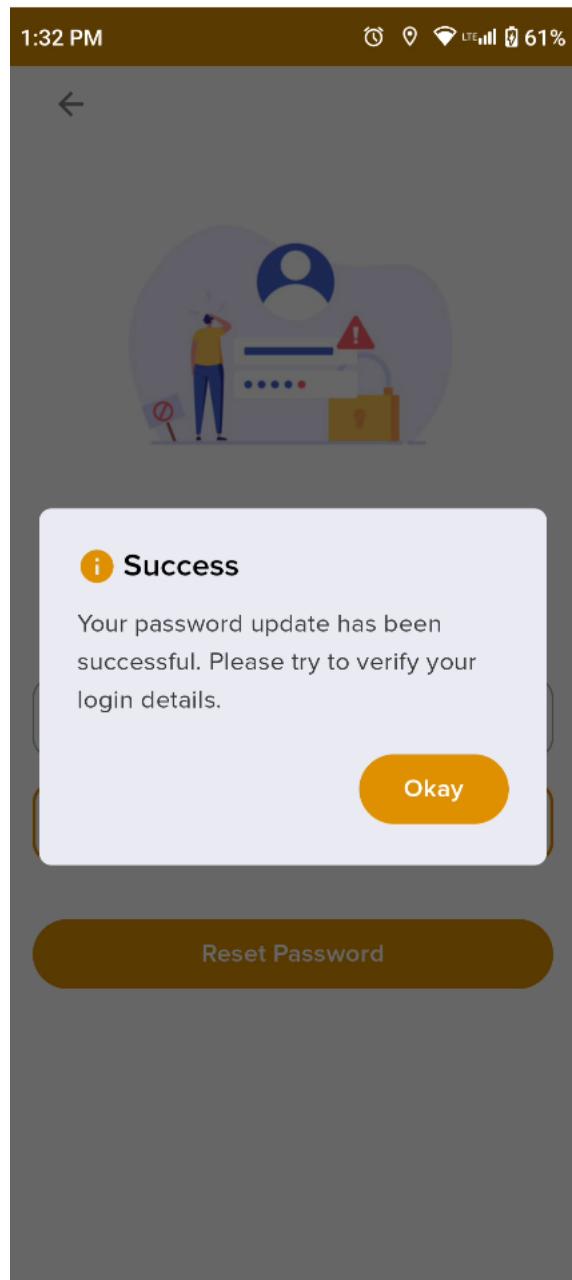


Figure 157: Display the success message daily box

4.2.2.14 View History

Test Case	Descriptions
Objectives	To navigate the history to view the all won history details.
Actions	Click the history button navigation bar and display the history list of history details.
Expected Result	To navigate the history page and show the history details without any error.
Actual Result	The histories are shown successfully.
Test	The test was successful.

Table 43: View the History test in the mobile app

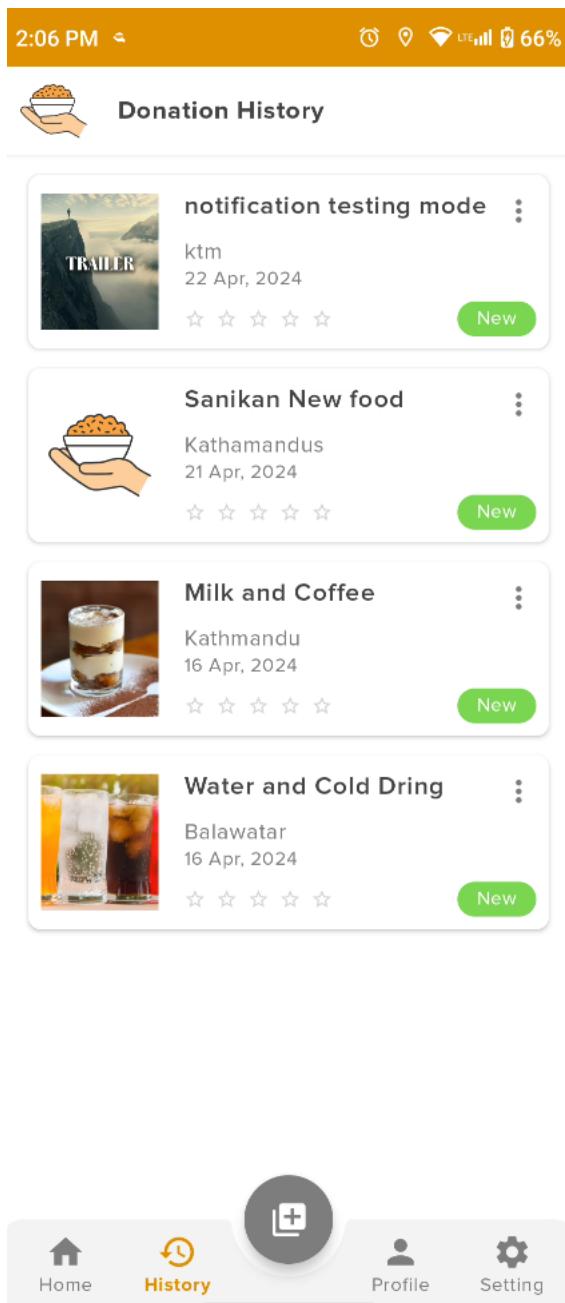


Figure 158: View list of food history history

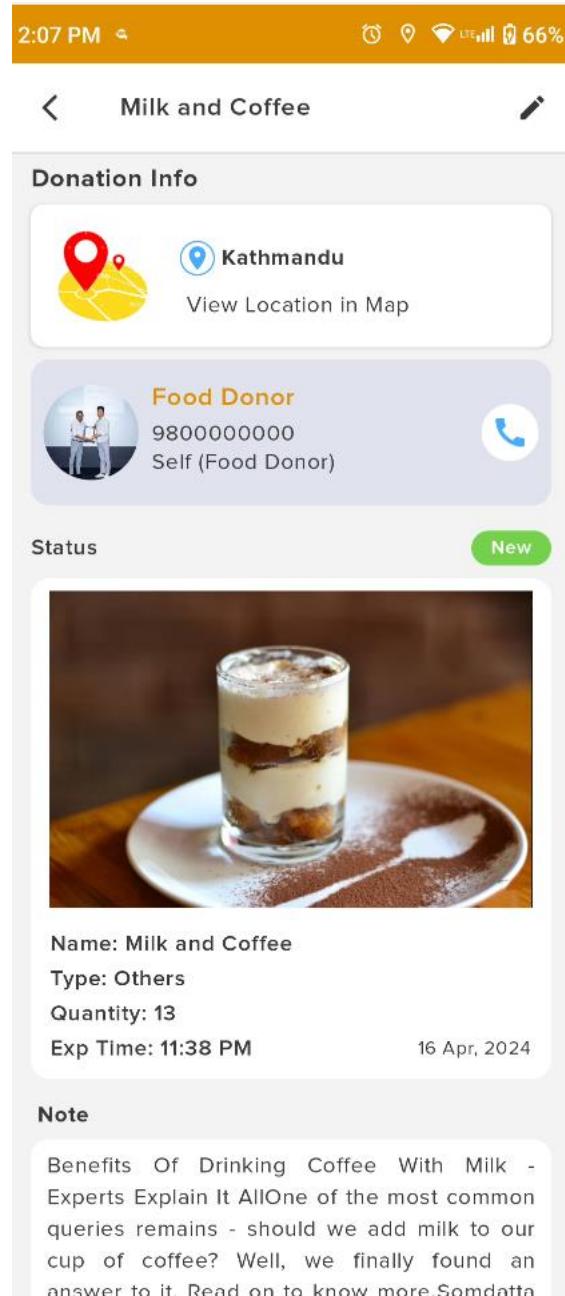


Figure 159: View food details

4.2.2.15 View Donation location

Test Case	Descriptions
Objectives	The user can view the donation location details with distance from the current location donation location.
Actions	Click the view location item and navigate to display the Google map if permission allows.
Expected Result	To navigate the view location screen and show the location distance.
Actual Result	The Google map is shown successfully.
Test	The test was successful.

Table 44: View the Donation location test in the mobile app

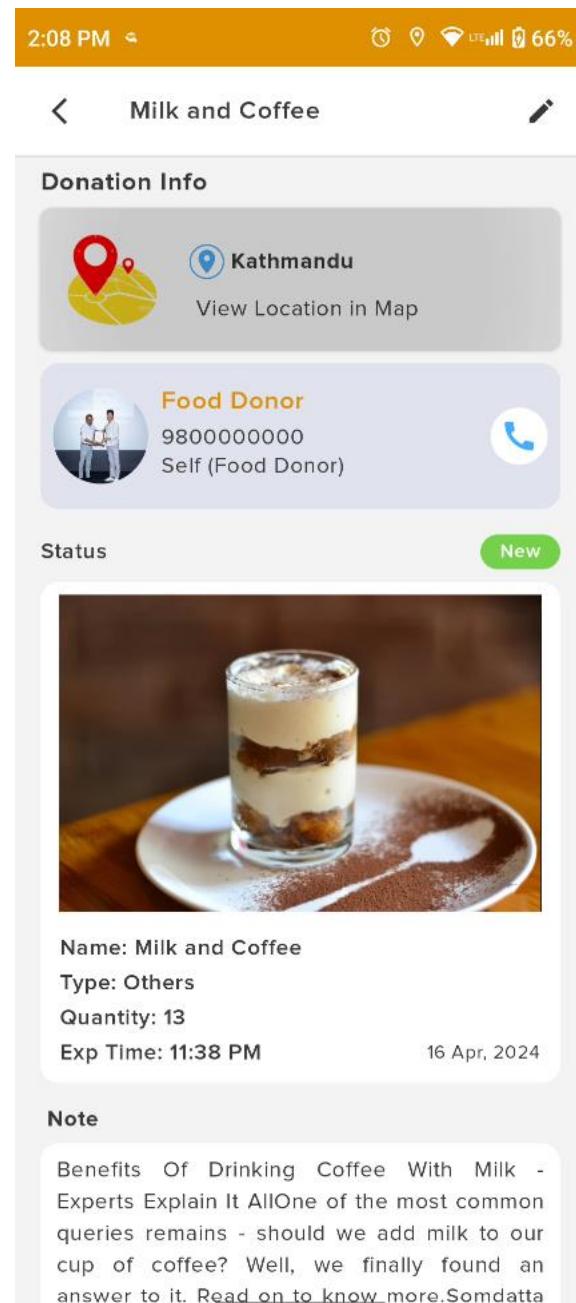


Figure 160: View donation location content

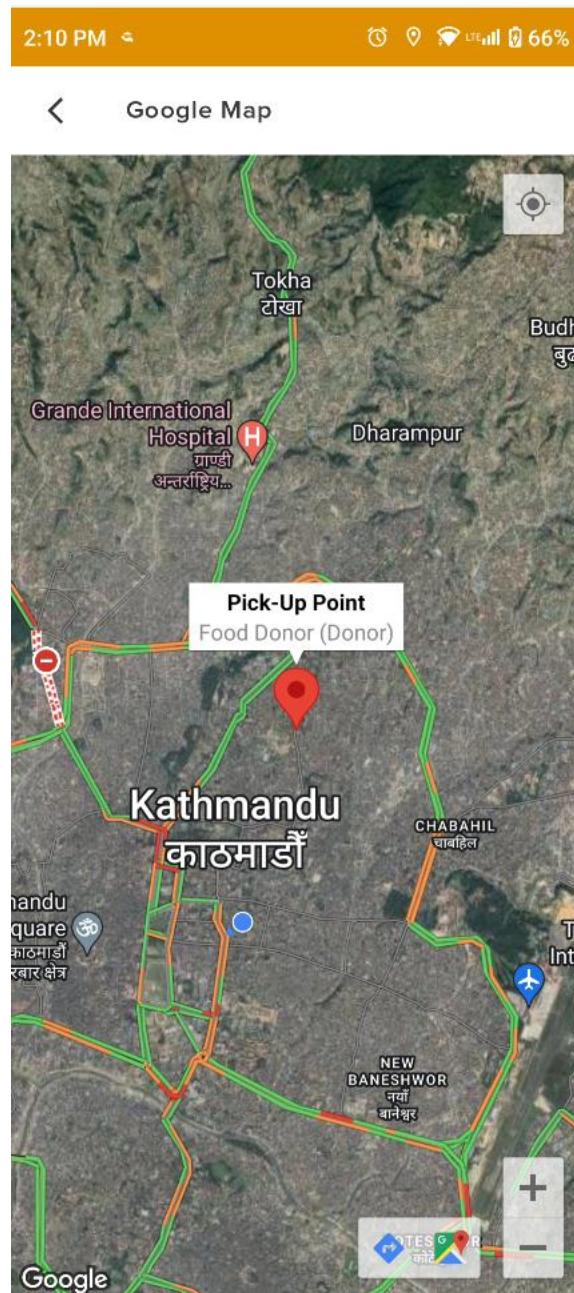
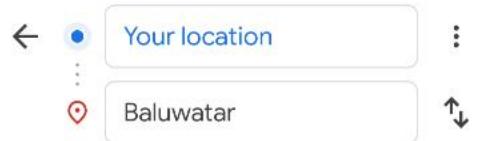
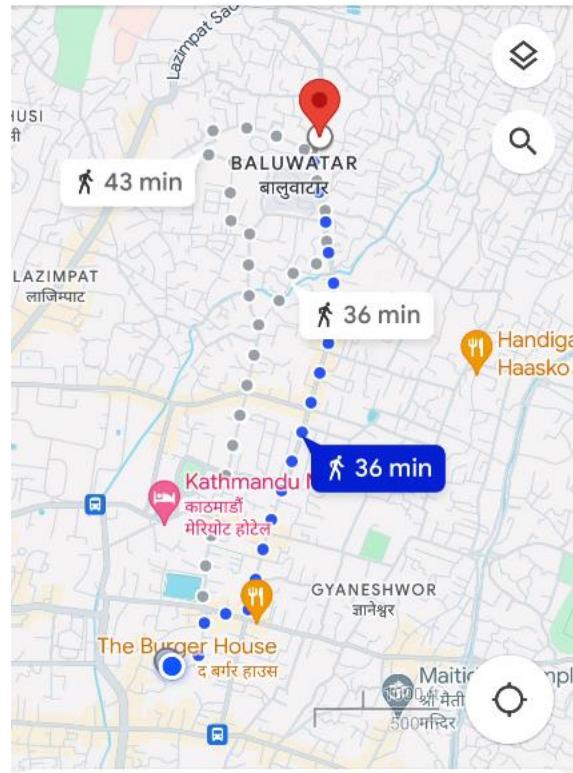


Figure 161: View the location on the map with the pick-up point

2:11 PM ⓘ ⚡ 66%



🚗 11 min 🚎 36 min 🚶 36 min



36 min (2.5 km) Mostly flat

via Thirbam Sadak

⚠ Start

📍 Live View

≡ Steps

Figure 162: View location details with distance

4.2.2.16 View user profile

Test Case	Descriptions
Objectives	The user can view the other system user profile details are view and contacts.
Actions	To find the post food and navigate the food details then click the user profile and view user profile details.
Expected Result	The user profile is displayed without any exceptions.
Actual Result	The user profile is successfully displayed.
Test	The test was successful.

Table 45: View user profile test in the mobile app

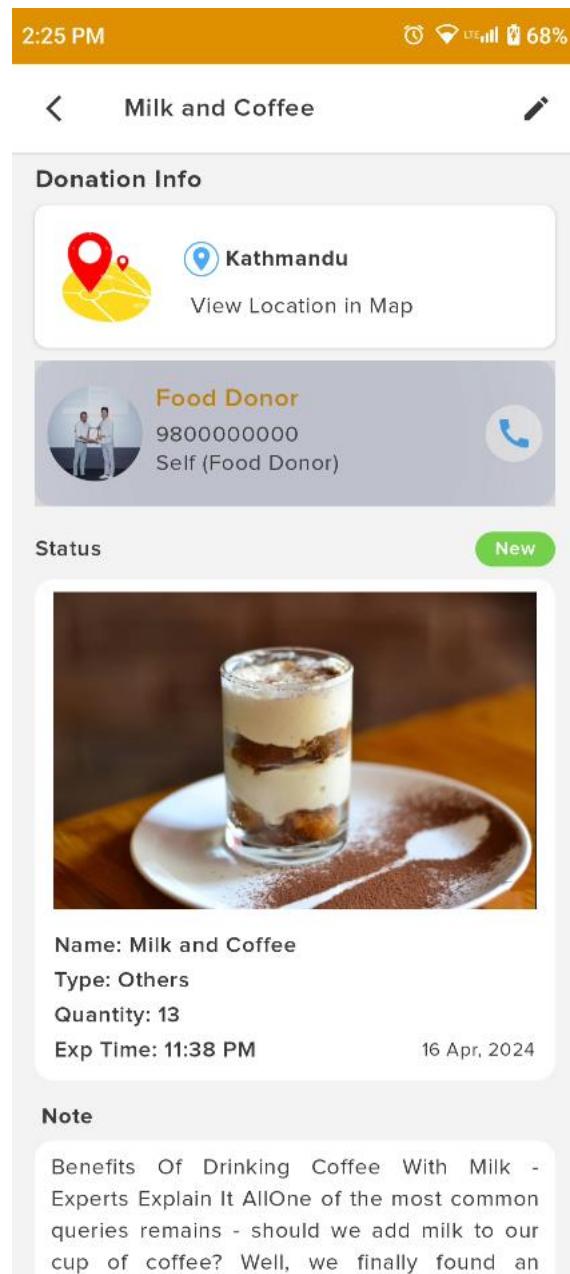


Figure 163: View the user profile content



Profile Details



Food Donor
Profile Id. 13

Profile Information

- Role
Donor
- Address
Admin
- Gender
Others
- Date of Birth
2024-04-16

Contact Us

- Email
donor@gmail.com
- Contact number
9800000000

Account Status

- Admin
false
- Active
true Verified

Figure 164: View user profile details

4.2.2.17 Donate food (Post new food)

Test Case	Descriptions
Objectives	To navigate the post screen and filled the required food details and publisher or post the system.
Actions	<p>Enter the food details in the filled food details like food name, and description, click the drop-down to food types, and quantity, select the donate time, fill in the donation location click the select icon and get the image the successfully post the food.</p> <p>Food name: Mo: Mo</p> <p>Descriptions: This food is donated to homeless persons.</p> <p>Time: 12:00 PM</p> <p>Quantity: 12</p> <p>Food types: Stable food</p> <p>Location: Putalishadak</p> <p>Image: momo's food image</p>
Expected Result	The food is successfully donated and if it is successful navigate the home screen and display the new food otherwise show the error message in the dialogue box.
Actual Result	The food was donated successfully without any errors or exceptions.
Test	The test was successful.

Table 46: Donate food test in monile app

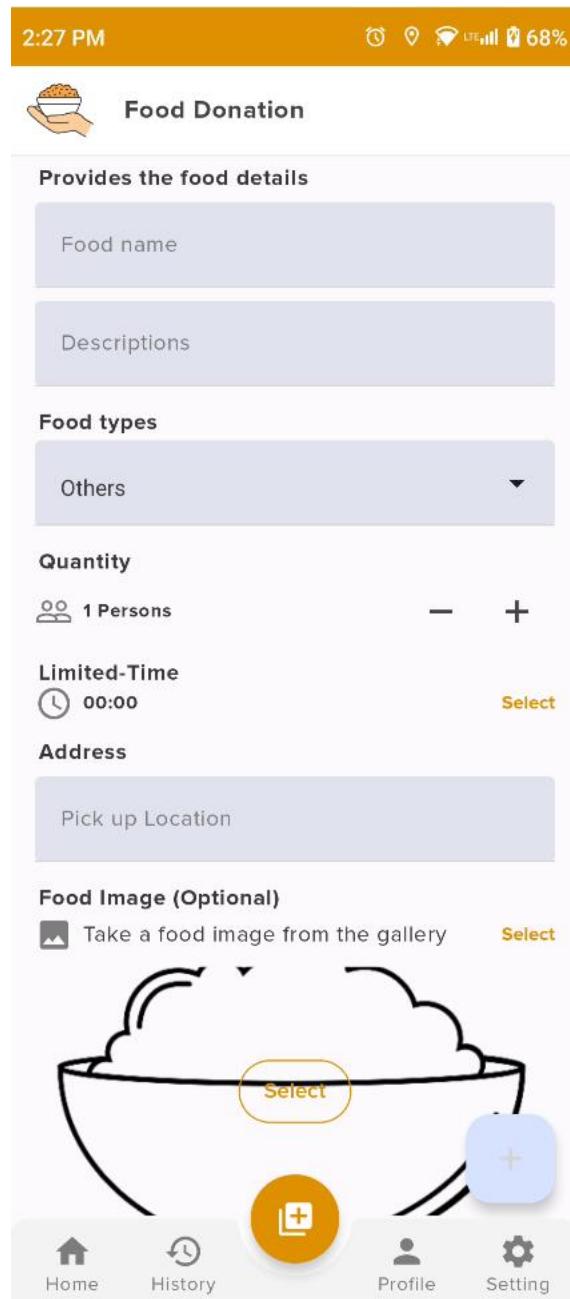


Figure 165: Empty donation details

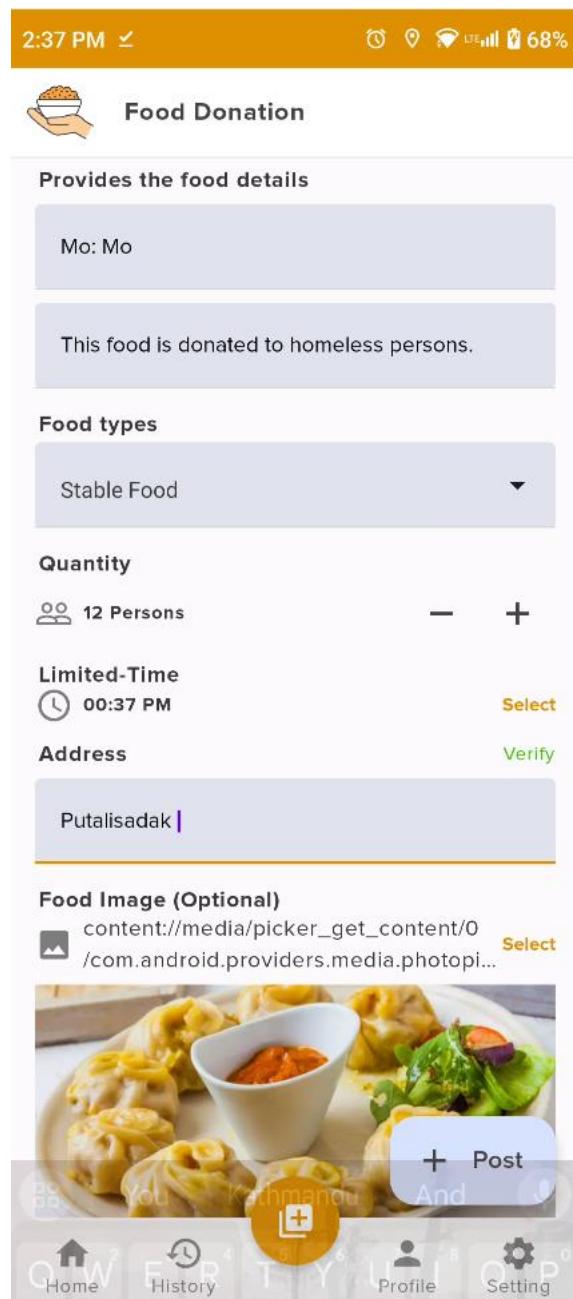


Figure 166: Donate food details filled

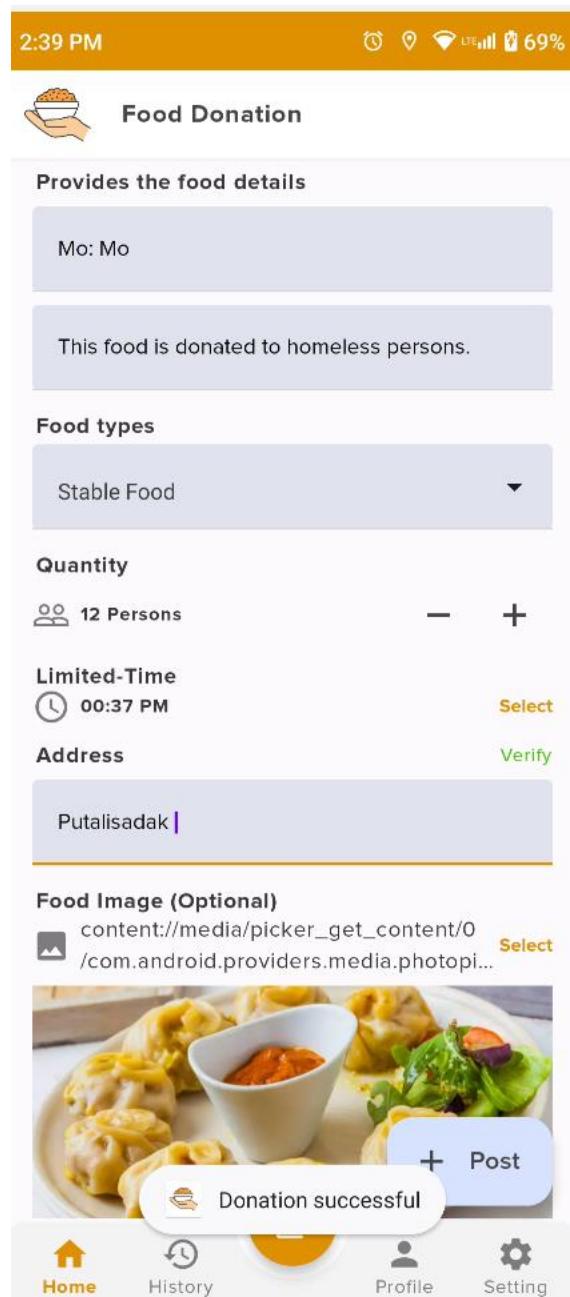


Figure 167: Donation success message

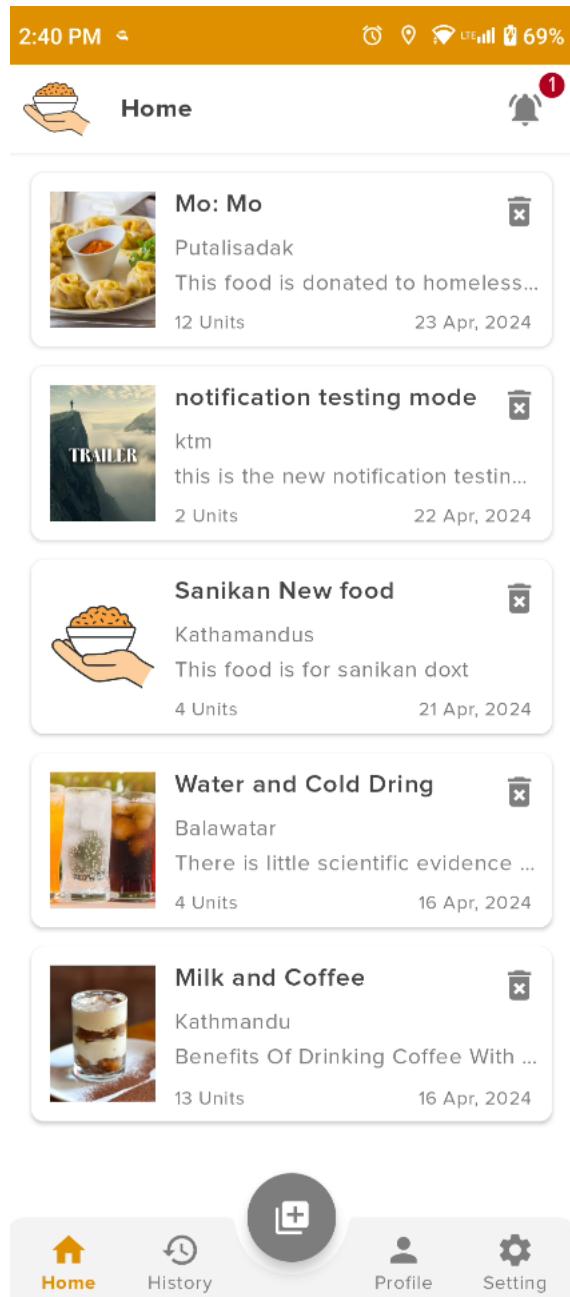


Figure 168: View the latest donated food on the home page

4.2.2.18 Push Notification

Test Case	Descriptions
Objectives	To new food donations from donors, all the users have come to the push notification.
Actions	The donor can donate the food with all the food details and a push notification can send the food name and descriptions where it is received to the other user. Food name: Mo: Mo Descriptions: This food is donated to homeless persons.
Expected Result	The push notification was received by all users without any exceptions.
Actual Result	The donate food push notification is successful.
Test	The test was successful.

Table 47: View notification history

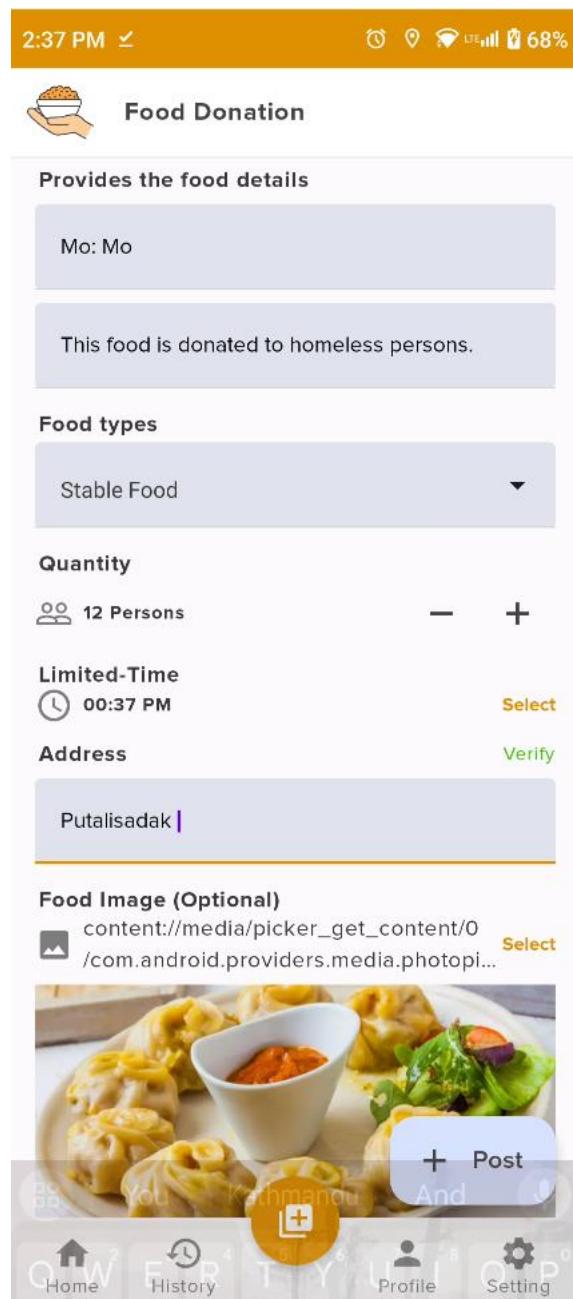


Figure 169: View donate food detail for notification

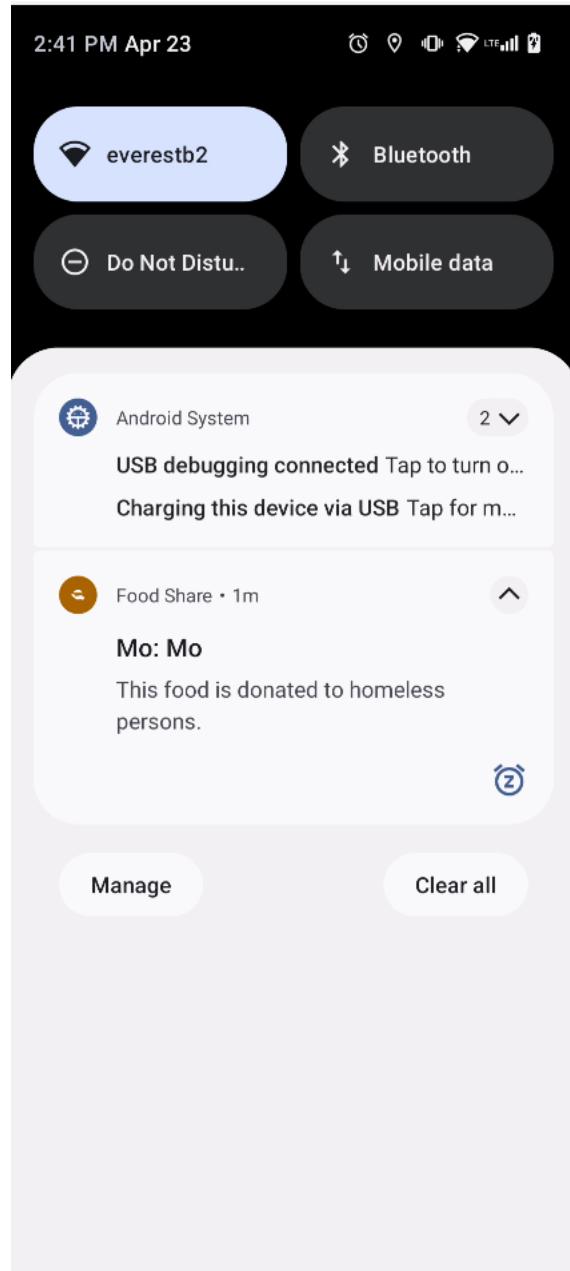


Figure 170: Received push notification

4.2.2.19 View Notification History

Test Case	Descriptions
Objectives	The user can view the latest number of notifications and notification details in the notification details.
Actions	Click the notification icon and navigate to the notification screen.
Expected Result	They view the number of notifications and display the identification history.
Actual Result	The notification was successfully displayed with history.
Test	The test was successful.

Table 48: View Notification History

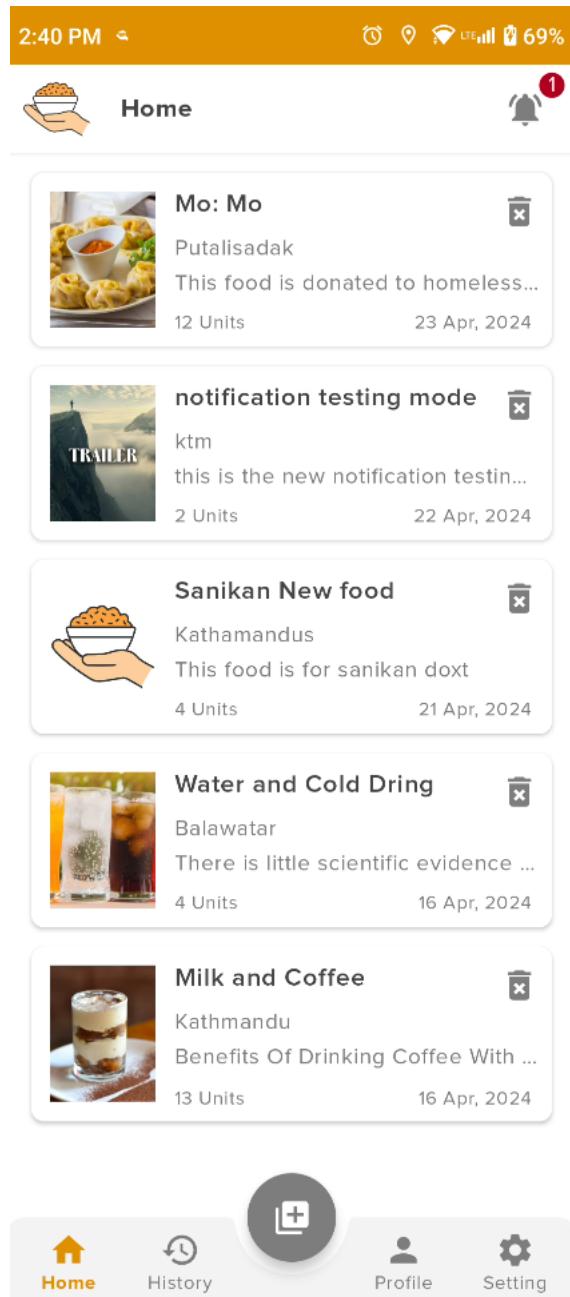


Figure 171: View the number of notifications in the badge

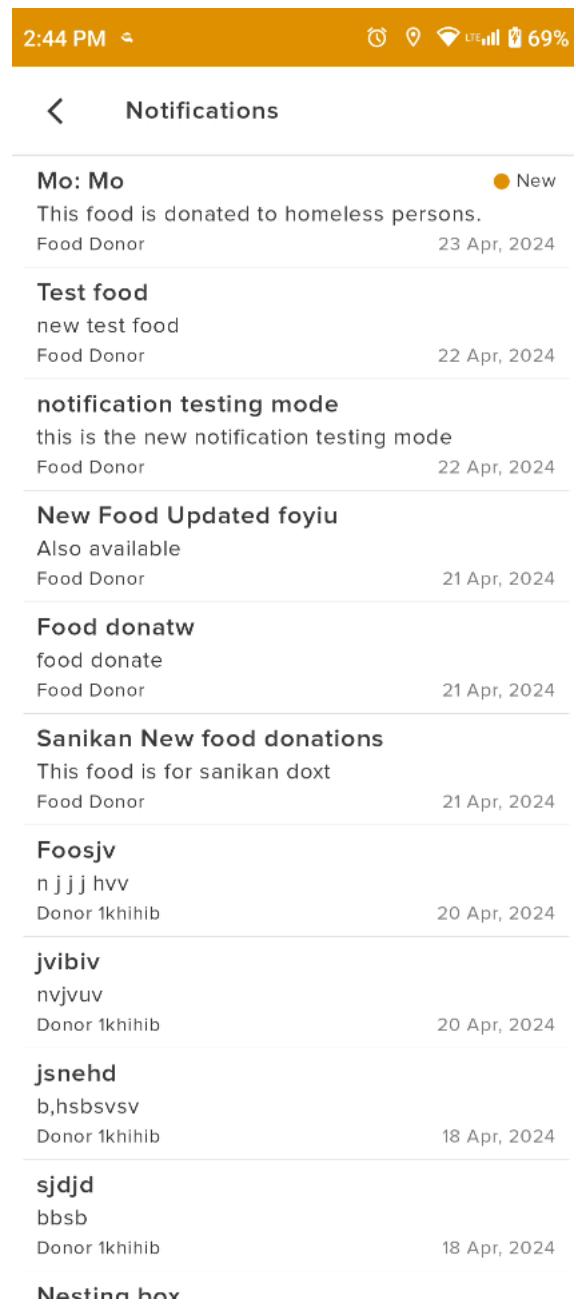


Figure 172: View notification history

4.2.2.20 Donation Rating

Test Case	Descriptions
Objectives	The volunteer can give the donation rating with the collect the feedback of food consumers.
Actions	The pending food is picked for distribution and completed the donation process then volunteer can input the distributed location, description rating and image if the donor can say donation completed evidence.
Expected Result	The donation rating is completed then show the history screen with a rating star.
Actual Result	The donation rating is successful and displays the history screen.
Test	The test was successful.

Table 49: Donation Rating

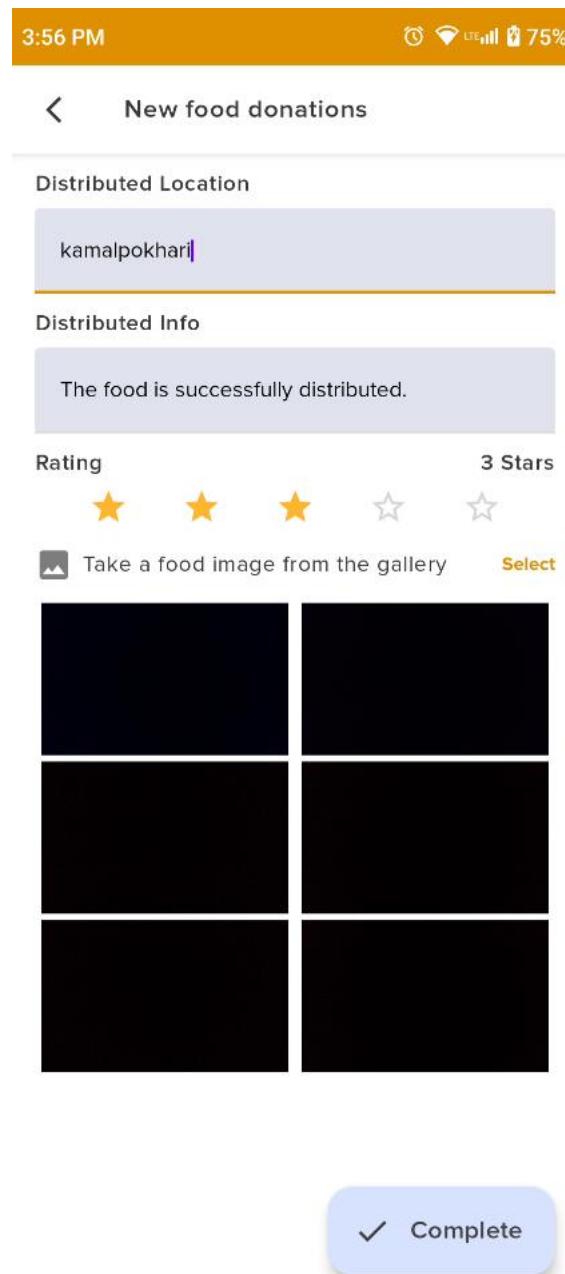


Figure 173: Enter the donation completed info with rating and images

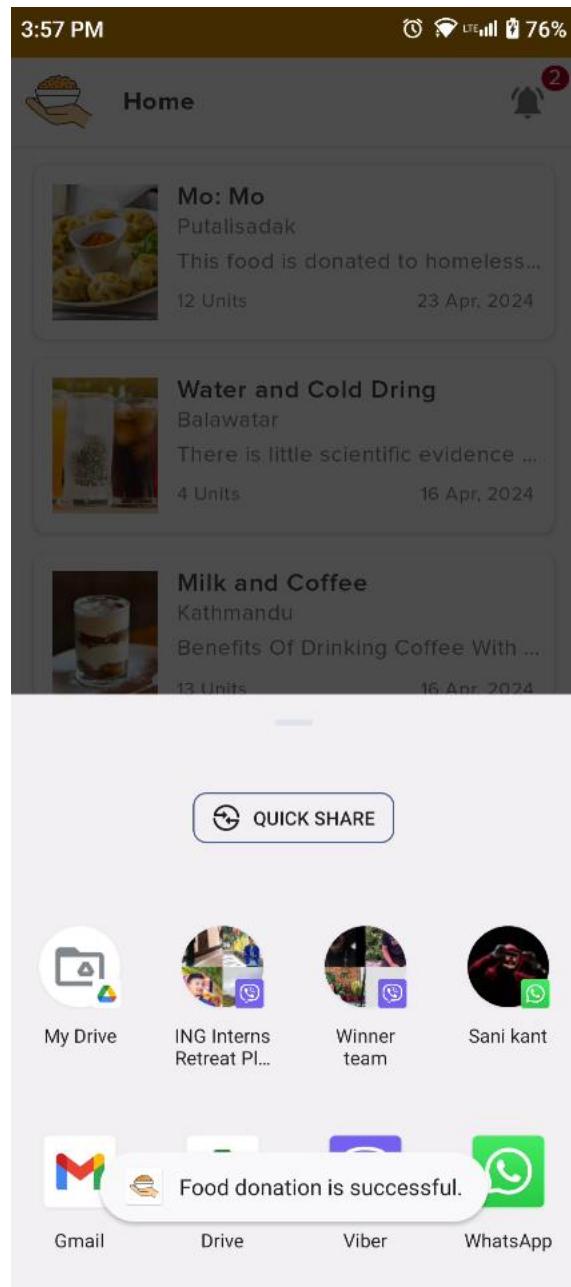


Figure 174: Donation rating completed message

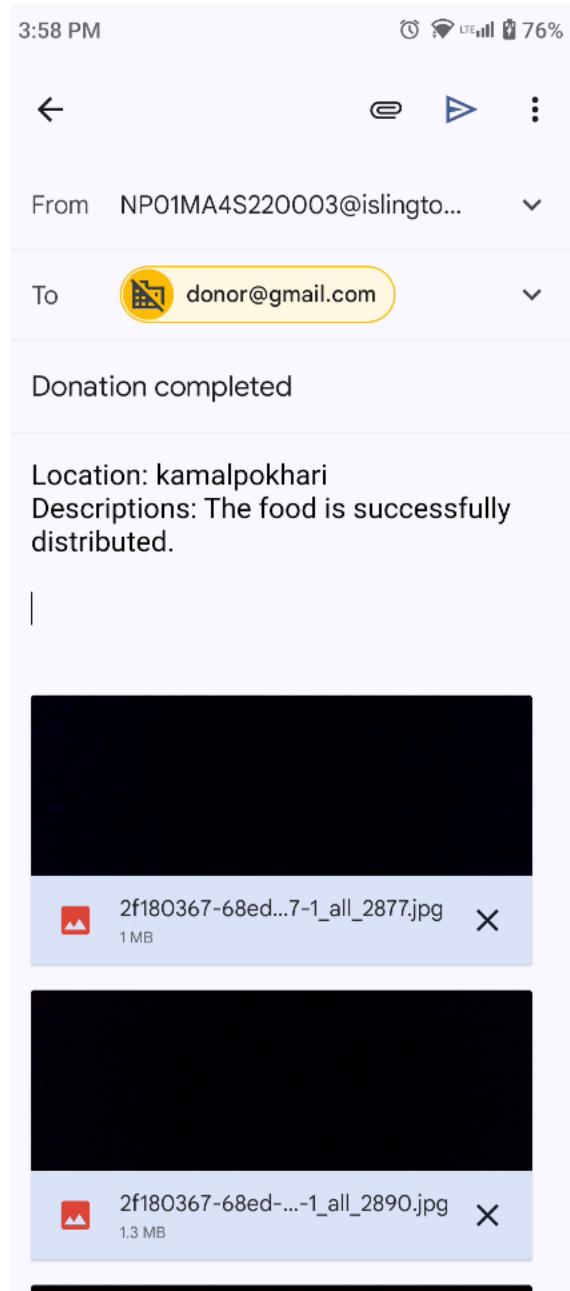


Figure 175: Distribute the location image and send

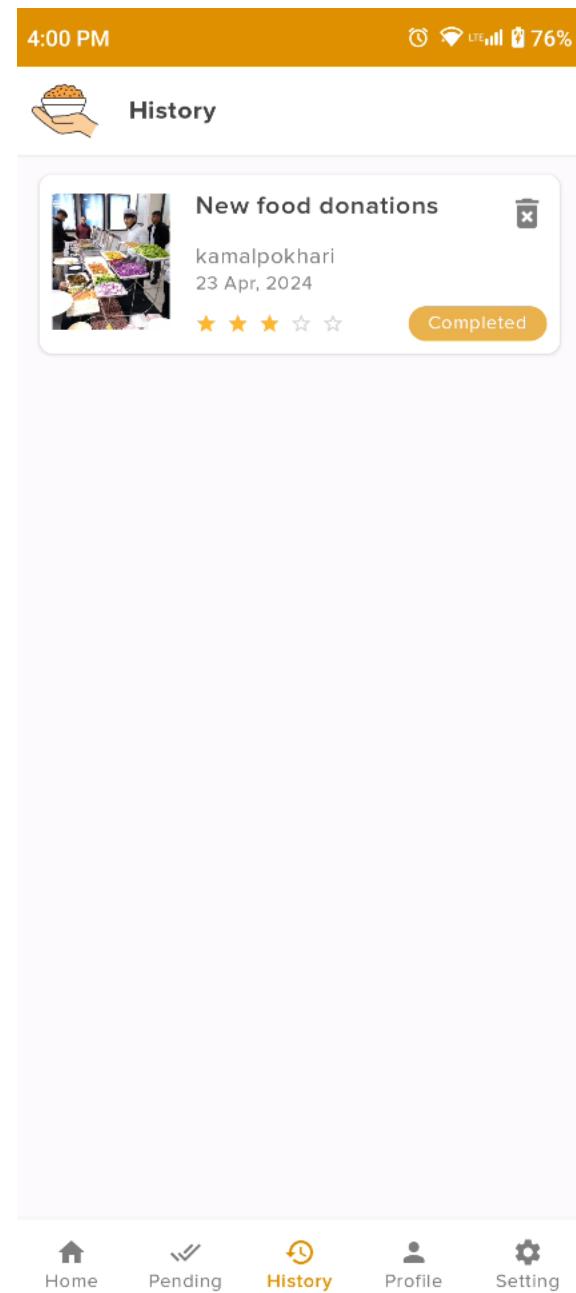


Figure 176: View history with rating star

4.2.2.21 Update food Item

Test Case	Descriptions
Objectives	The donor can donate food if this needs to change then they can update the food details.
Actions	The donor can click the edit icon and update the necessary items update.
Expected Result	The previously donated food details are successfully updated without any errors or exceptions.
Actual Result	The post was successfully updated
Test	The test was successful.

Figure 177: Update food Item

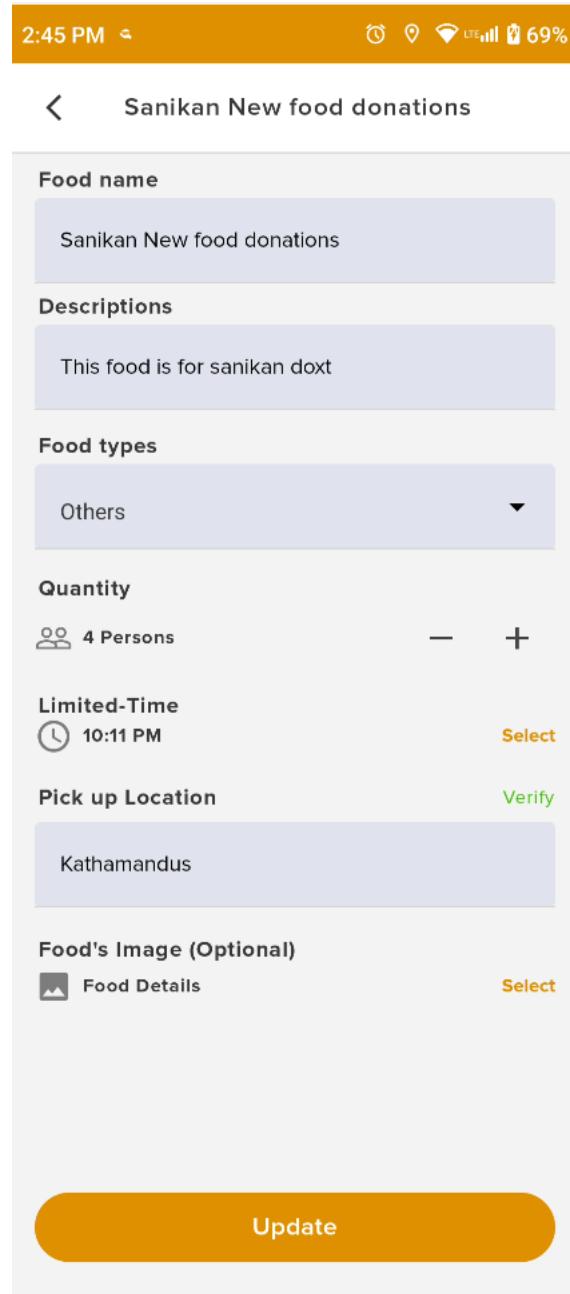


Figure 178: Change food details for an update

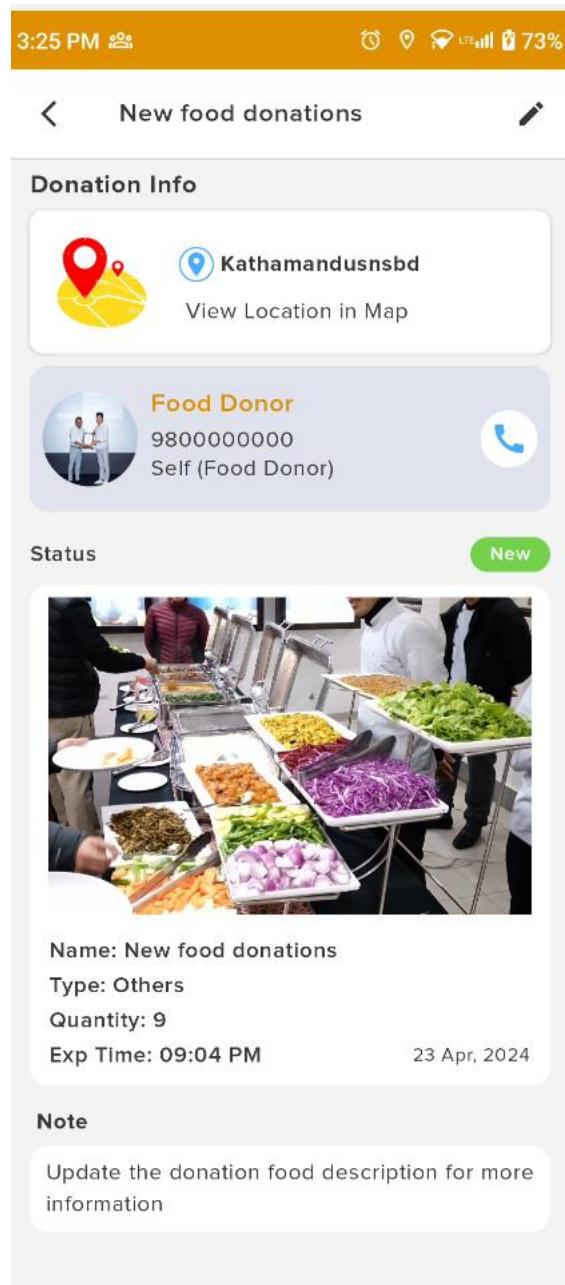


Figure 179: Display Update food details

4.2.2.22 Delete donate food Item

Test Case	Descriptions
Objectives	The donor can donate food if it can not be visible to donate the food.
Actions	The donated food items have a deleted icon and click the delete icon to show the confirmation and click the okay button.
Expected Result	The post food is deleted successfully without any exceptions.
Actual Result	The data was successfully deleted from the front end.
Test	The test was successful.

Figure 180: Delete donated food Item

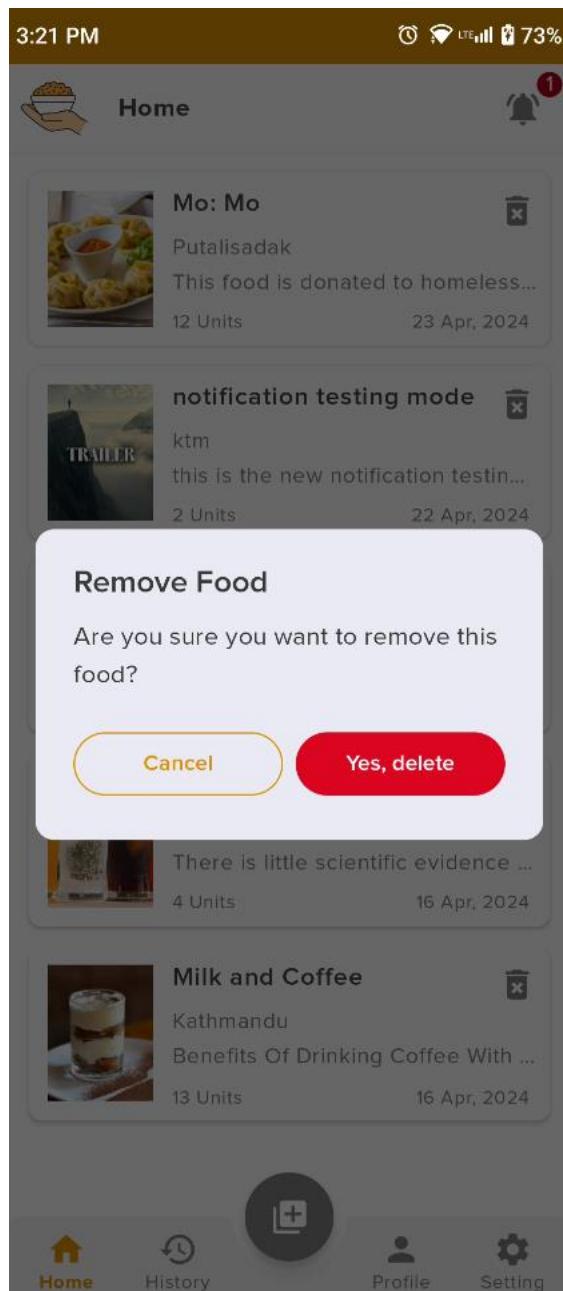


Figure 181: Before detailing the food

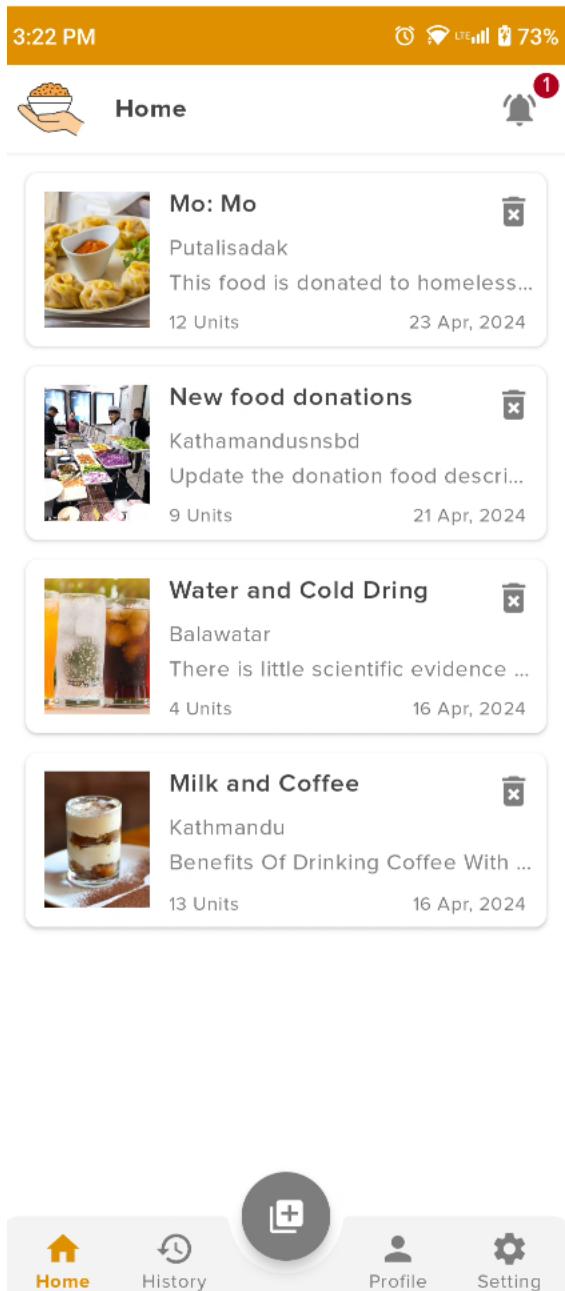


Figure 182: Delete confirmation dialogue

4.2.2.23 Update donate food image

Test Case	Descriptions
Objectives	The user can update the user profile image if the user can needed.
Actions	The user can click the profile edit icon update the food image and click the update confirm button.
Expected Result	The donated food images are successfully updated without any errors or exceptions.
Actual Result	The donated food image was successfully updated.
Test	The test was successful.

Figure 183: Update donate food image

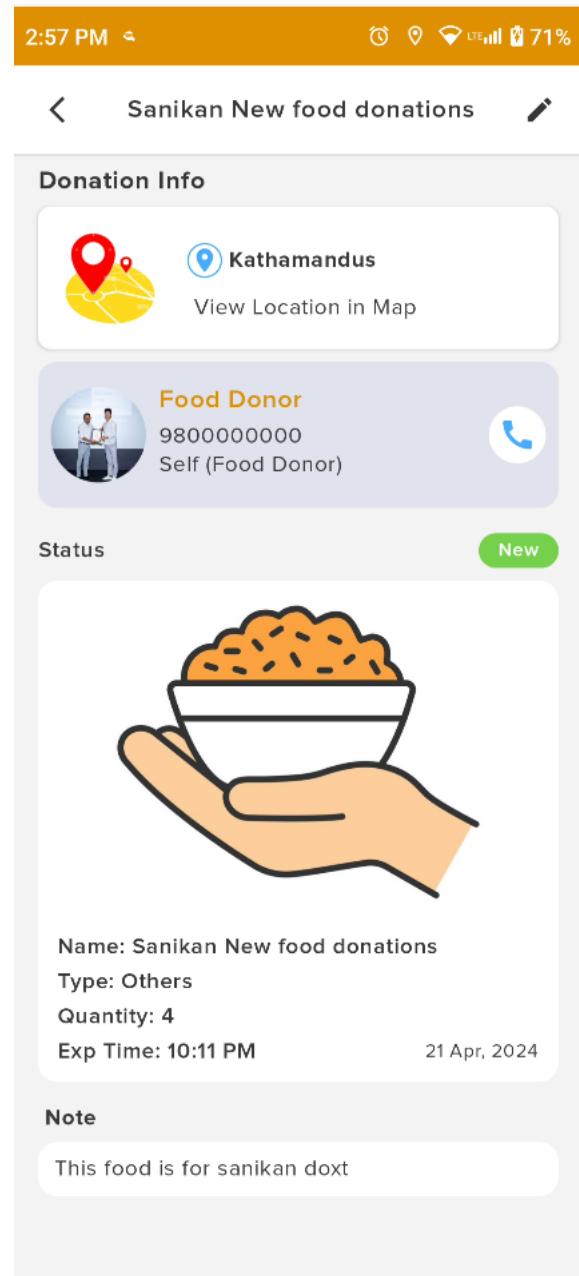


Figure 184: Before updating the food image

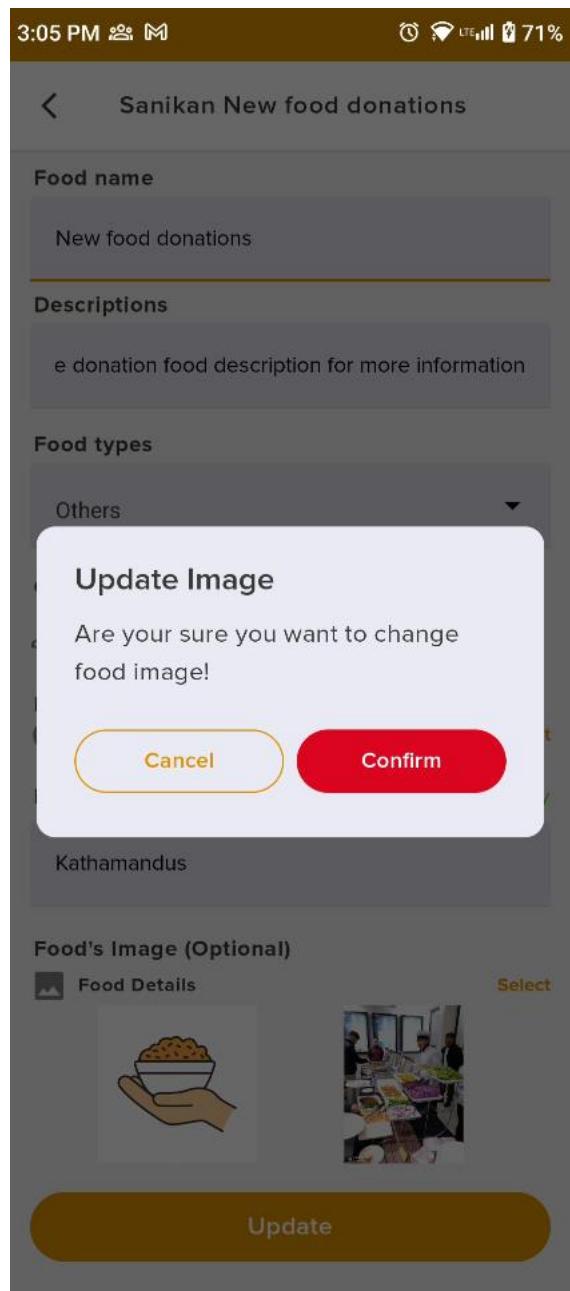


Figure 185: Food image update confirmation dialogue box

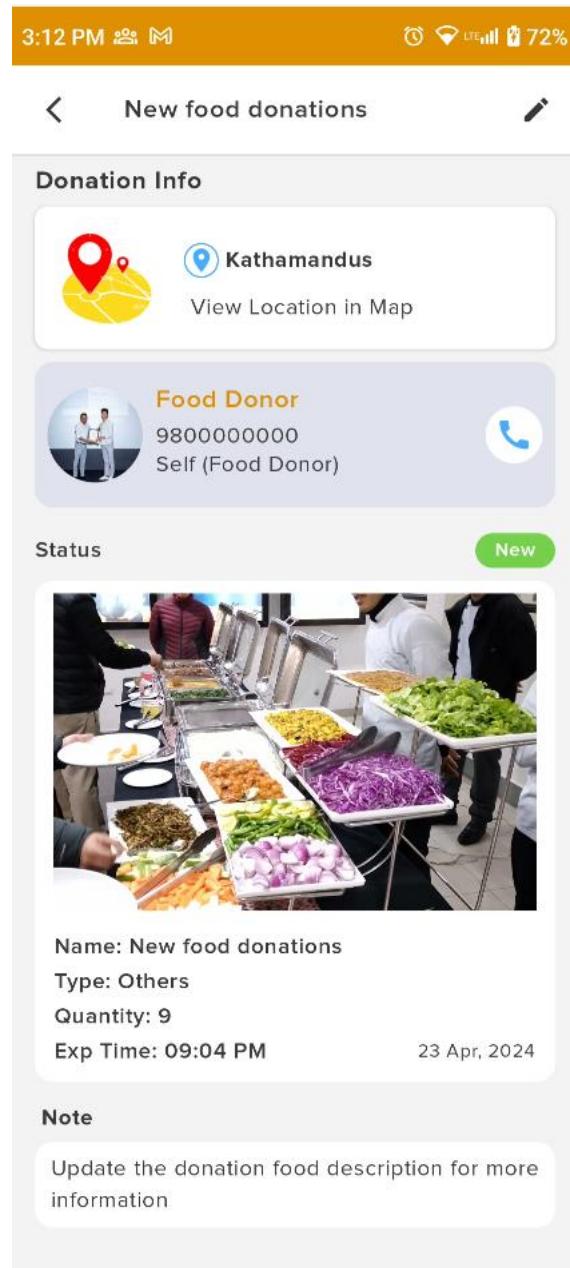


Figure 186: Updated food image

4.2.2.24 Report to admin

Test Case	Descriptions
Objectives	When food cannot be donated and volunteers are unable to pick up the food from the company, users should be able to report this issue to the admin.
Actions	Click the report icon, fill in the report description, and click the submit button.
Expected Result	The report is successfully submitted to the admin.
Actual Result	The report was successfully submitted to the admin.
Test	The test was successful.

Table 50: Report to admin test in mobile app

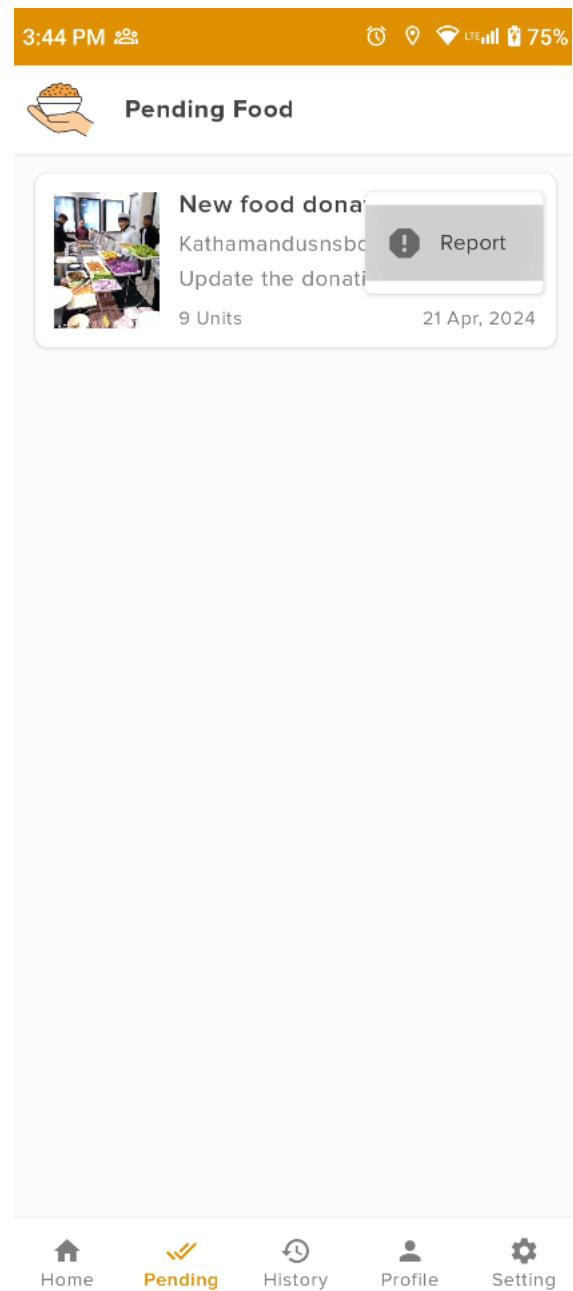


Figure 187: Report Item Menus

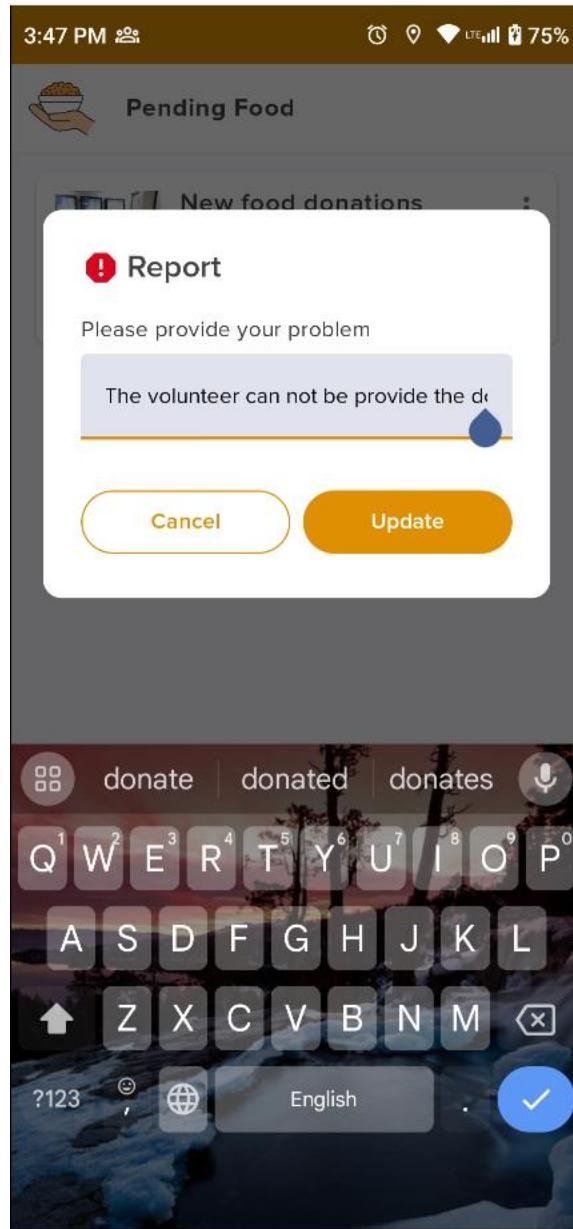


Figure 188: Write the report descriptions in the dialogue box

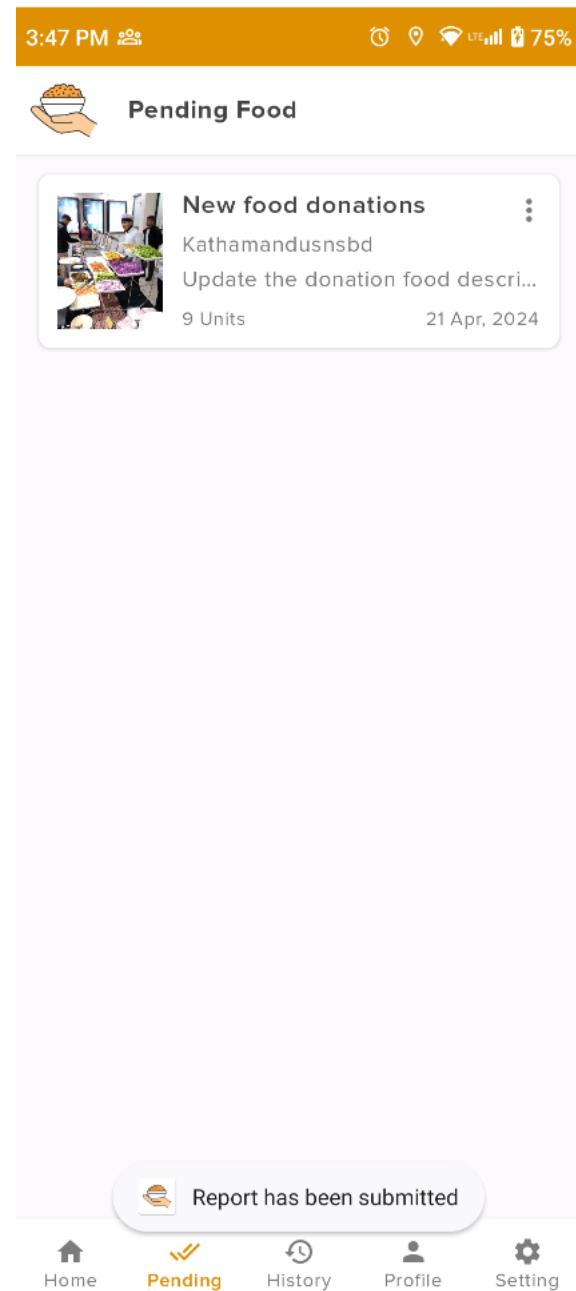


Figure 189: Report successful message

4.2.2.25 Update profile details

Test Case	Descriptions
Objectives	The user can update user profile details like username, gender, contact number, location and about details.
Actions	The user can click the profile edit icon update the profile details and click the update button.
Expected Result	The user profile details are successfully updated without any errors or exceptions.
Actual Result	The user profile details were successfully updated.
Test	The test was successful.

Table 51: Update profile details test in the mobile app

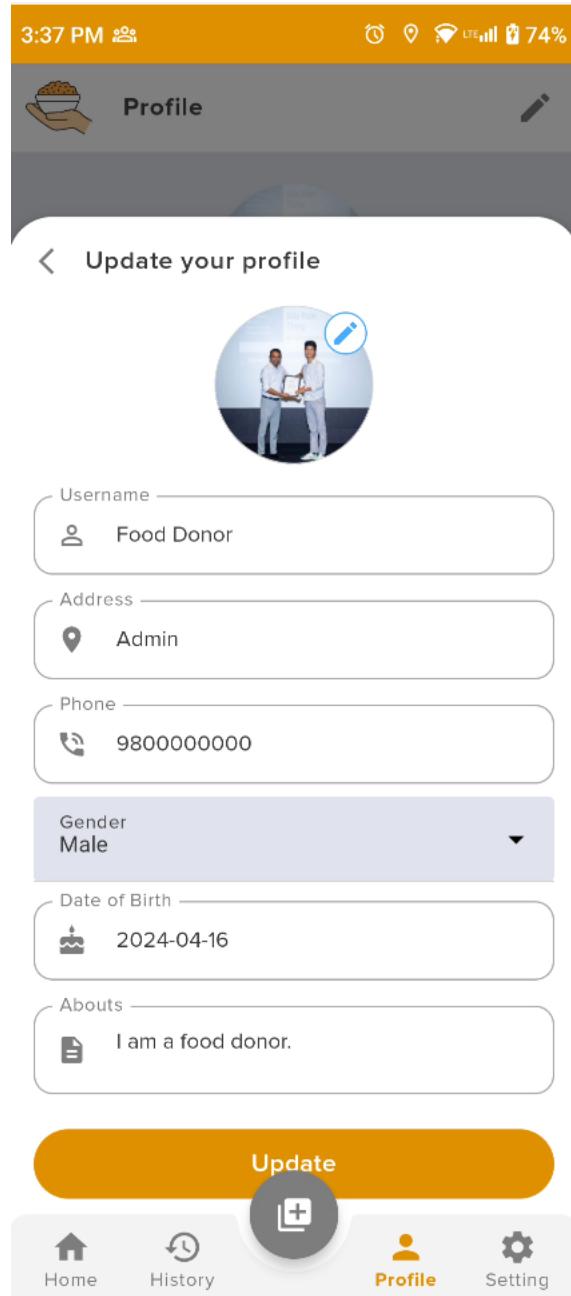


Figure 190: before updating the username in the profile

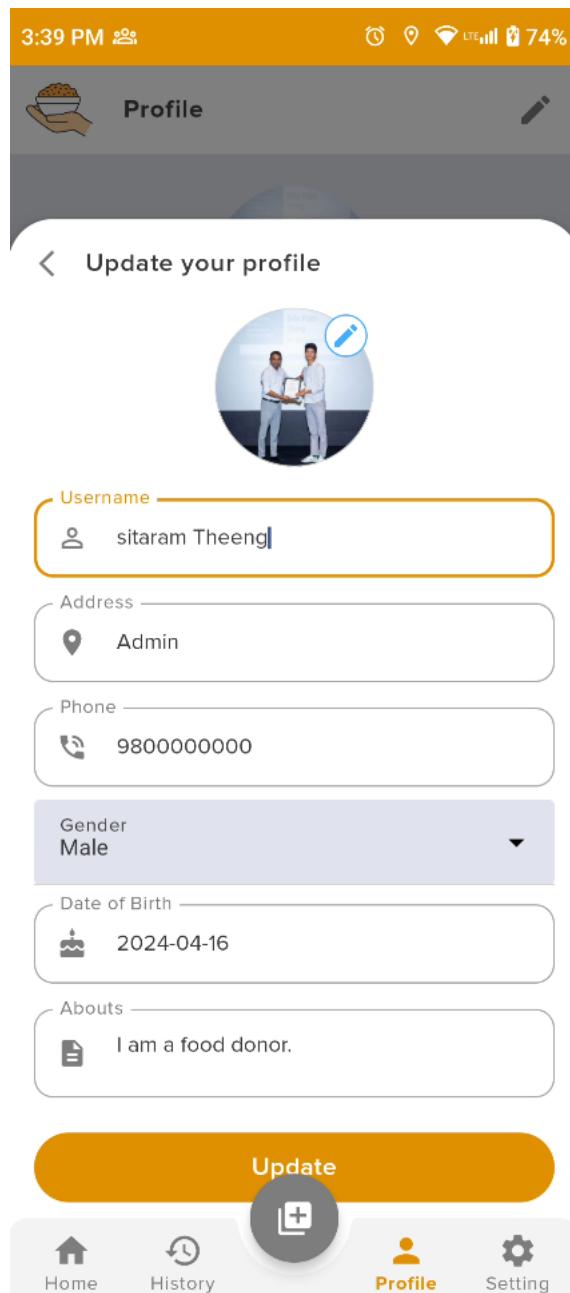


Figure 191: Update food profile username

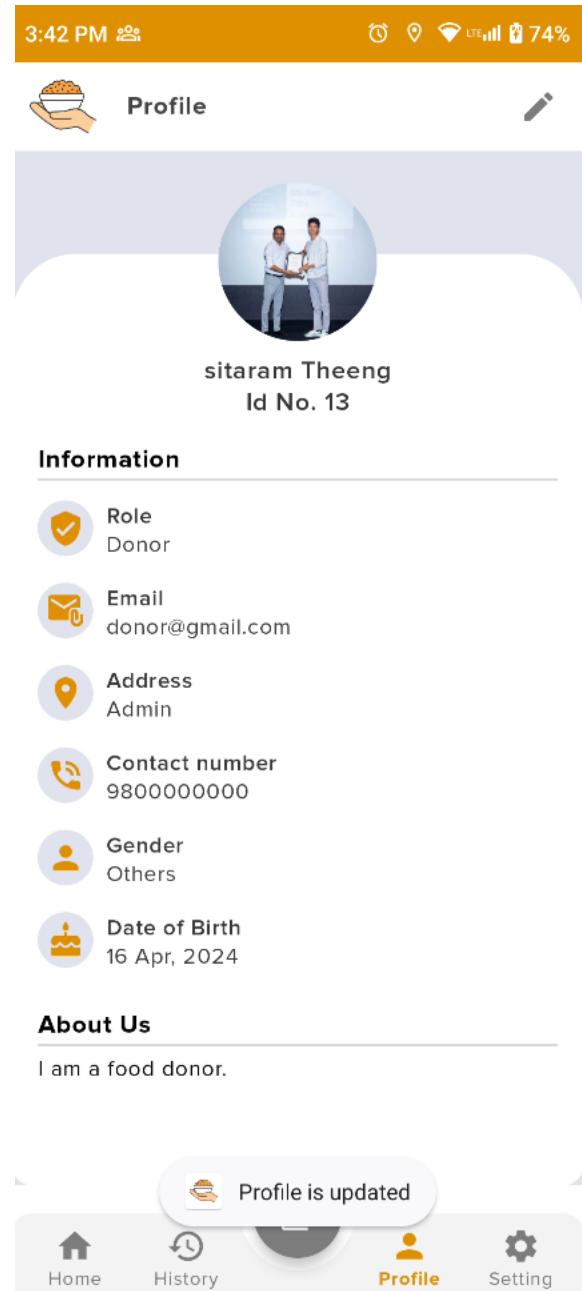


Figure 192: Latest updated user profile details

System testing findings and identified errors are detailed in the appendix.

4.3. Critical Analysis

The critical part of the systems development process lies in researching different topics and identifying the best approach for waste food management and food donation. Research plays a pivotal role in collecting data and identifying potential clients. The client then articulates their requirements, and the system's overall architecture is designed accordingly. Subsequently, the system's UI/UX design and database architecture are developed. Once the database development is complete, the Django Rest API is developed from Django to establish the backend. The backend facilitates the creation of the frontend web admin panel and mobile app for users. Each completed development part undergoes thorough testing, including API tests to ensure resolution and implementation of the front end. After completing the unit and system testing phases, a post-survey is conducted to fine-tune design and business logic. The final system test is then conducted to validate all features. The systems development phase encountered a major issue when the backend server failed to run, and the database decryption error arose. After some time, the issues were identified and resolved. Nonetheless, during the final stages, the mobile app encountered significant challenges, including class duplication and version duplication errors in Jetpack Compose. These issues necessitated further troubleshooting and resolution efforts.

The "Waste Food Management System" is developed by using stack in Kotlin, Jetpack Compose, Room database implemented to Android Mobile Applications and Python Django Rest Framework, HTML, CSS, MySQL Database implement to develop the Web and Backend system where it encountered several some problems. Despite these initial challenges, the system persevered, with errors being identified and corrected as development progressed. Eventually, all components of the project passed through both self-conducted unit and system tests, indicating a robust development process. The system now stands fully operational, boasting all the features outlined in its initial proposal. However, the testing phase not only identified areas for correction but also provided valuable insights into potential improvements and future development avenues.

User system testing for the latest version of the system provided crucial feedback from potential end-users. While certain features such as data capture and visualization were well-received, other aspects, notably the web user interface, did not garner the expected attention. User feedback highlighted the need for real-time comprehensive descriptions of intruders, suggesting a crucial area for future development. Additionally, users expressed a preference for mobile message alerts over email notifications, firebase push notifications, Google Maps, and Local Notification, for admin alerts, indicating an opportunity to enhance the system's alerting mechanisms. Furthermore, the successful incorporation of machine learning algorithms, particularly Decision Tree and Logistic Regression, showcased the system's advanced capabilities in traffic analysis. However, there remains room for exploration of more complex models to further enhance accuracy and predictive capabilities. Where issues and problems are successfully solved with the help of research and development skills.

In conclusion, the food donation system exhibits resilience and adaptability in its development journey, overcoming initial challenges to emerge as a fully functional platform. While it successfully implements key features and demonstrates advanced capabilities in traffic analysis, user feedback has illuminated areas for improvement, such as real-time data visualization and enhanced alerting mechanisms. Moving forward, a focus on addressing user needs, refining system functionalities, and exploring advanced modelling techniques will be crucial to ensuring the system's continued effectiveness and relevance in addressing food waste and security challenges.

CHAPTER 5: CONCLUSION

In conclusion, developing the "Food Share" app which is a waste or surplus food management system represents a significant stride towards addressing food waste while fostering community engagement and social responsibility. Through the adoption of a scrum methodology, the project progressed methodically, overcoming complexities in mobile development and user interface design. Guided by the goal of creating a comprehensive solution accessible to various user groups involved in the donation process, the platform aims to streamline interactions and facilitate a cohesive donation process.

Utilizing technologies like Kotlin with Jetpack Compose for mobile development and HTML, CSS, and Bootstrap for web development, user experience has been prioritised across both platforms. Features like notifications and Google Maps integration enhance user engagement and highlight donation locations. As the project nears its conclusion, the focus shifts to final testing to ensure a flawless user experience before deployment. Plans include ongoing expansion and improvement, particularly in enhancing communication features to foster better collaboration among users.

Despite initial simplicity, the project presented obstacles requiring unwavering commitment and correction of mistakes during development. Guided by provided guidelines and constant advice, adjustments were made to ensure the project's quality met expectations. The project provided valuable learning experiences in various technologies including Jet Pack Compose, Kotlin, Bootstrap, Python, Django, MVVM with Clean Architecture, SQL and Room Databases, and Scrum Methodology.

Working on the project provided insight into real-world app development and collaboration with clients, contributing to a sense of professionalism. Additionally, my experience with version control using git, task management with Trello, and comprehensive documentation creation enriched my understanding and skills. Ultimately, the project facilitated learning and practical application of new languages and frameworks, enhancing readiness for future career endeavours.

5.1. LEGAL, SOCIAL AND ETHICAL ISSUES

5.1.1. LEGAL ISSUES

During the development of the "Food Share" app, we ensured strict adherence to legal standards. The system was meticulously designed to comply with user data protection laws and regulations. Clear terms and conditions were provided to users regarding data handling practices. To safeguard privacy, user activity logs were anonymized, and users were given the option to control their data-sharing preferences. Additionally, comprehensive documentation of all project materials and resources further reinforced our commitment to legal compliance, fostering transparency and accountability throughout the development process.

5.1.2. SOCIAL ISSUES

The "Food Share" app catalyzes inclusivity and community empowerment. It transcends cultural and economic barriers, offering accessibility to all sectors of society. By prioritizing transparency and integrity, the app cultivates trust and collaboration among users, thereby enhancing social cohesion. With a steadfast commitment to fair and equitable resource distribution, the app works towards mitigating disparities and advancing collective welfare.

5.1.3. ETHICAL ISSUES

Ethical integrity is paramount in the development and operation of the "Food Share" app. Upholding intellectual property rights and fostering a culture of integrity in software development are central tenets. Ethical considerations extend to data privacy and security, with robust measures implemented to protect user information from unauthorized access or misuse. By adhering to principles of transparency and respecting user autonomy, the app upholds ethical standards in all interactions.

5.2. Advantages of waste food or surplus food donation system

Advantages of the "Food Share" App:

- Utilize waste food effectively and efficiently, reducing food waste and promoting sustainability.
- Raise social awareness about food waste, fostering a culture of responsible consumption and environmental stewardship.
- Facilitate food donation and distribution, connecting food donors with those in need and contributing to hunger alleviation efforts.
- Encourage responsible food handling and safety practices, ensuring the safety and quality of donated food items.
- Promote a sense of community and social responsibility, fostering collaboration and collective action towards addressing food insecurity and waste reduction.
- Efficiently reduce food waste through surplus food management, optimizing resource utilization and minimizing environmental impact.
- Alleviate hunger by efficiently connecting food donors with individuals experiencing food insecurity, ensuring equitable access to nutritious meals.
- Promote community engagement and social responsibility by facilitating collaboration among users and encouraging active participation in food donation initiatives.
- Streamline the donation process with user-friendly interfaces and communication features, enhancing accessibility and usability for all stakeholders.
- Provide opportunities for skill development and learning for project contributors, fostering professional growth and expertise in social impact initiatives.
- Ensure scalability and continuous improvement, enabling the app to adapt to evolving user needs and challenges and maximize its positive social impact.

5.4. Limitations

Every application has its own set of benefits and drawbacks. This system lacks full user control, where donors have access to donate food and maintain their profile details, including updating, deleting, and managing them, while volunteers also have access to view donor details and donation information, along with the capability to communicate with donors and accept food for distribution. Volunteers can then pick up the food and proceed to distribute it to poor and homeless people in certain areas. In addition to the benefits listed above, this system has certain limitations as listed below:

5.1.1 Limited User Control:

Particularly volunteers and donors, may not have full control over their profile information and donation activities. This lack of control could lead to concerns about privacy and security.

5.1.2 Risk of Misuse:

Since donors can update, delete, and maintain their profile details, there's a risk of misuse or inaccurate information being provided, which could impact the effectiveness of the donation process.

5.1.3 Potential for Inefficiency:

Allowing donors to directly update and delete their donation information may lead to inefficiencies in the system, such as duplicate entries or inaccurate inventory management if proper oversight and validation mechanisms are not in place.

5.1.4 Limited Transparency:

While volunteers have access to view donor profiles and donation details, there may be limitations in transparency regarding the handling and distribution of donated food. This lack of transparency could raise concerns about accountability and trust within the community.

5.1.5 Dependency on Volunteer Availability:

The system relies heavily on the availability and commitment of volunteers to pick up and distribute donated food to the intended recipients. Fluctuations in volunteer availability could lead to inconsistencies in food distribution and service delivery.

5.1.6 Geographical Constraints:

The system's distribution model, where volunteers distribute food to specific areas, may create geographical constraints, limiting access to surplus food for individuals in remote or underserved locations.

5.1.7 Communication Challenges:

While volunteers can communicate with donors to accept food donations, there may be challenges in communication efficiency and effectiveness, particularly if there's a lack of standardized communication protocols or language barriers between volunteers and donors.

5.1.8 Limited Platform Availability:

The exclusivity to the Android app means users without Android devices, such as iOS users or those preferring web platforms, cannot access the system, restricting their participation and potentially reducing the overall impact of the surplus food donation initiative.

5.1.9 Unequal Access:

The system's reliance on volunteers and donors with internet access and digital literacy skills may create barriers for individuals or organizations without access to technology, potentially excluding certain demographics from participating in the surplus food donation process.

Addressing these limitations through continuous monitoring, feedback mechanisms, and system enhancements can help improve the efficiency, transparency, and inclusivity of the surplus food donation system, ensuring its effectiveness in addressing food waste and supporting vulnerable communities.

5.3. FUTURE WORK

The system is currently developed solely for native Android. However, several crucial features need to be implemented in the future to enhance the systematic handling of food donations and improve performance. Users vary in their preferences, with some preferring a light mode while others may have visual impairments necessitating a different theme. Real-time communication is far more efficient than email or calls, allowing donors and volunteers to interact seamlessly, thereby improving performance and communication. The system would greatly benefit from being accessible across multiple platforms, allowing all users to utilize different platforms at their convenience. Therefore, the project needs to expand beyond native Android to accommodate this requirement.

The "Food Share" waste food management system application is a commendable project that aims to assist both surplus food managers and the impoverished and homeless. The three major features Dark Mode, Chat (Message), and Multi-platform are work in future.

5.3.1 Dark Mode

In the future, I will implement a dark mode feature to enhance the user experience. This addition will provide an alternative colour scheme that is easier on the eyes, especially in low-light conditions. By incorporating dark mode, the application will become more accessible and user-friendly, catering to a wider audience, and improving overall usability. Additionally, dark mode can help reduce eye strain and battery consumption on devices with older screens, making the application more energy efficient. The features are research online, websites and reference material guidelines, Jet Pack compose theming part and implements in future. The user can activate the theme of the appropriate program, or the user can enable it via his system settings. The application has a dark theme, giving users an additional experience and can even increase engagement through user identification.

5.3.2 Chat (Messaging)

Integrating a chat or messaging functionality into the application will enable users, donors, and volunteers to communicate directly within the platform. This feature will facilitate seamless coordination between users, offering real-time assistance, and fostering a sense of community among participants in the food donation process. By allowing direct communication, the application will enhance user engagement and satisfaction. Moreover, the messaging feature can serve as a valuable tool for providing updates, notifications, and reminders related to food donations, improving overall user experience and engagement. This feature can facilitate coordination between users, provide real-time assistance, and foster a sense of community among participants in the food donation process.

The Chat or messaging features work in future where the shock is implemented to develop the messaging system. In future, I will learn about the more advanced Django Channel and develop the socket API for two-way communication. This can be better and easier for the system user and flexible. With the help of Django Channels, a potent Django extension, you can create WebSocket-based real-time apps. With WebSockets, you can push changes to the client without the client having to constantly request data because they allow bidirectional communication between the client and the server. The fundamentals of Django Channels and WebSockets, as well as how to leverage them to create real-time applications, will be addressed in this section **Invalid source specified..** The web socket implementation to fully develop the messaging services in future.

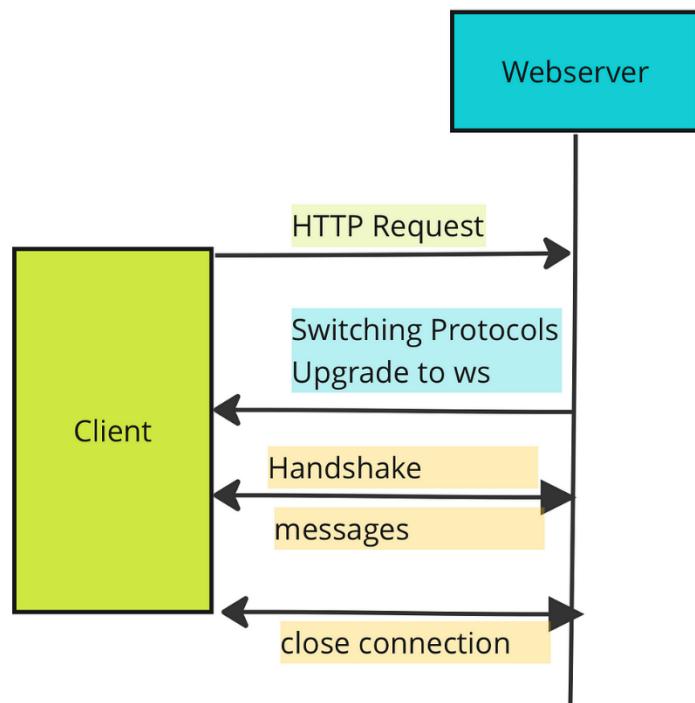


Figure 193: Future work Socket implementation for Chat.

5.3.3. Kotlin multi-platform

Adopting Kotlin multi-platform development in the future will revolutionize our approach to app development. This strategy will enable us to share code logic across different platforms, including Android, iOS, and the web. Leveraging Kotlin's multi-platform capabilities, we can streamline development efforts, reduce code duplication, and ensure consistency across all versions of the application. By embracing this approach, we will enhance maintainability and scalability while optimizing resource utilization, leading to a more efficient and robust system overall. Additionally, Kotlin's interoperability with existing Java codebases will facilitate seamless integration and transition, minimizing development time and effort. Adopting Kotlin multi-platform development would enable us to share code logic across different platforms, such as Android, iOS, and the web. By leveraging Kotlin's multi-platform capabilities, we can streamline development efforts, reduce code duplication, and ensure consistency across all versions of the application. This approach enhances maintainability and scalability while optimizing resource utilization.

Kotlin Multiplatform technology makes it easier to construct cross-platform solutions. It saves time on creating and maintaining the same code for several platforms while preserving the flexibility and benefits of native programming. Check out the Get Started with Kotlin Multiplatform and Construct a Multiplatform App Using Ktor and SQL Delight lessons. These lessons will teach you how to construct Android and iOS mobile apps with desktop and web applications incorporating a module with shared code for both platforms. **Invalid source specified.**

Declarative framework for sharing UIs across multiple platforms. Based on Kotlin and Jetpack Compose.

Developed by JetBrains and open-source contributors.

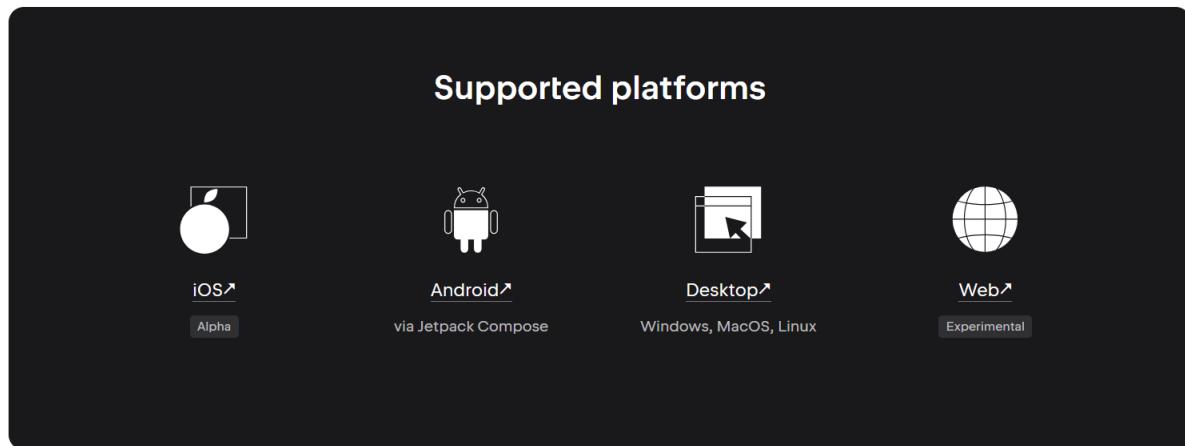


Figure 194: Future work for multi-platform.

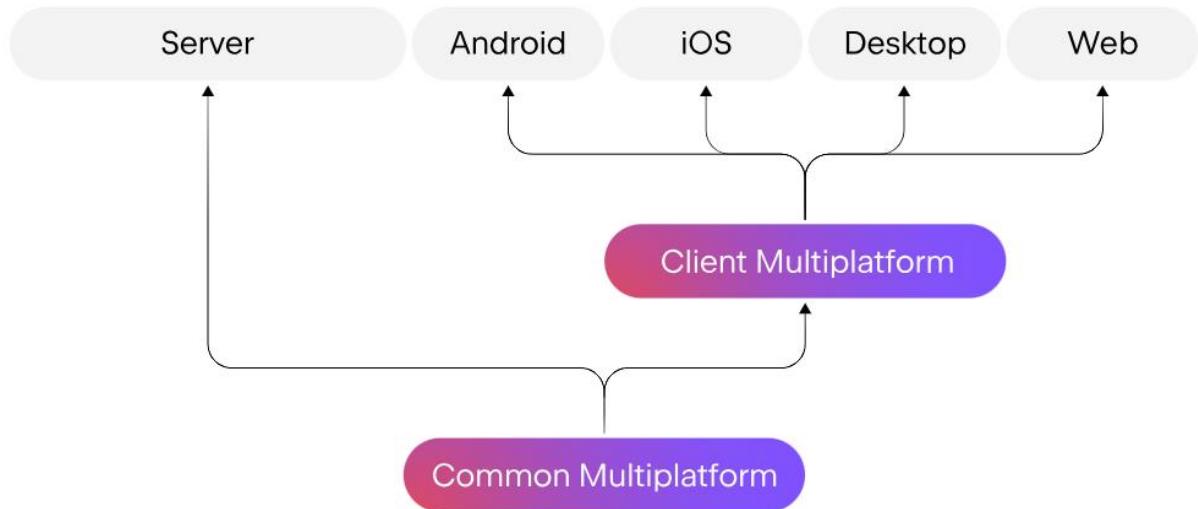


Figure 195: Multiplatform connection

CHAPTER 6: References

- AWS. (2024, 04 13). *What is Scrum?* Retrieved from AWS:
<https://aws.amazon.com/what-is/scrum/#:~:text=Scrum%20is%20a%20management%20framework,experience%2C%20and%20adapt%20to%20change>.
- Chiarelli, A. (2024, 04 19). *Auth0 by Okta*. Retrieved from THE CONFUSED DEVELOPER: <https://auth0.com/blog/id-token-access-token-what-is-the-difference/>
- Dave Lampert, M. S. (2024, 04 07). *Now Fair Food NZ*. Retrieved from Food Rescue:
<https://fairfood.org.nz/about-us/our-story/#impact>
- Dave Lampert, M. S. (2024, 04 07). *Food Reduces*. Retrieved from Food Reduces Us:
<https://foodrescue.us/our-app/>
- Developer, G. (2024, 04 23). *Developer*. Retrieved from Save data in a local database using Room: <https://developer.android.com/training/data-storage/room>
- developer, G. (2024, 04 18). *FCM*. Retrieved from Firebase:
<https://firebase.google.com/docs/cloud-messaging/android/client>
- Developer, G. (2024, 04 23). *Google for the developer*. Retrieved from Build Better Apps Faster with: <https://developer.android.com/develop/ui/compose>
- Dhanajay Jhala, K. s. (2021). Spread the smile. *Food Donation*, p. 51.
- Django Rest Framework*. (2024, 04 22). Retrieved from Serializers: <https://www.django-rest-framework.org/api-guide/serializers/>
- GeekForGeek. (2024, 04 21). *GeekForGeek*. Retrieved from Levels in Data Flow Diagrams (DFD): <https://www.geeksforgeeks.org/levels-in-data-flow-diagrams-dfd/>
- GeekForGeek. (2024, 04 21). *Unified Modeling Language (UML)*. Retrieved from Use Case Diagrams: <https://www.geeksforgeeks.org/use-case-diagram/>

Google. (2024, 04 17). *Developer*. Retrieved from Save simple data with SharedPreferences: <https://developer.android.com/training/data-storage/shared-preferences>

Inc, L. S. (2024, 04 21). *Lucidechart*. Retrieved from Entity Relationship Diagram: <https://www.lucidchart.com/pages/er-diagrams>

JavaTpoint. (2024, 04 13). *Software Engineering*. Retrieved from www.javatpoint.com: <https://www.javatpoint.com/software-engineering-prototype-model>

Kruthika V, L. H. (07, July-2023). INTEGRATED APPROACH FOR FOOD DONATION SYSTEM, RESTAURANT. *International Research Journal of Modernization in Engineering Technology and Science*, 8.

Kruthika V, L. H. (07, July-2023). INTEGRATED APPROACH FOR FOOD DONATION SYSTEM, RESTAURANT. *International Research Journal of Modernization in Engineering Technology and Science*, 8.

Levan, M. (2024, 04 20). *TeachTargate*. Retrieved from Unit Testing: <https://www.techtarget.com/searchsoftwarequality/definition/unit-testing>

local government bodies, N. (2024, 04 07). *Flashfood Grocery deals*. Retrieved from Google Play Store: <https://play.google.com/store/apps/details?id=com.flashfoodapp.android&hl=en&gl=US>

LUMITEX. (2024, 04 13). Prototyping Methodology. *Dow Circle Strongsville, Ohio*, p. 10.

Mello, M. (2024, 04 21). *System Architecture*. Retrieved from The Blueprint for Successful Systems: https://dev.to/msmello_/system-architecture-the-blueprint-for-successful-systems-35nn

MkDocs. (2024, 04 21). *Django REST framework*. Retrieved from Django: <https://www.django-rest-framework.org/api-guide/serializers/>

Nepal, G. o. (2020-2021). Nepal towards an equitable, resilient and sustainable food system. *Nepal's Food Systems Transformation*, 64.

Nepal, G. o. (2020-2021). Nepal towards an equitable, resilient and sustainable food system. *Nepal's Food Systems Transformation*: p. 64.

Nick Barney, M. E. (2024, 04 21). *TeachTargate*. Retrieved from authentication:
<https://www.techtarget.com/searchsecurity/definition/authentication>

Reichert, A. (2024, 04 20). *TeachTargate*. Retrieved from system testing:
<https://www.techtarget.com/searchsoftwarequality/definition/system-testing>

S, B. (2024, 04 13). *nimble Humanize work*. Retrieved from Scrum Project Management: <https://www.nimblework.com/agile/scrum-methodology/>

Singtel, P. (2024, 04 23). *Altexsoft*. Retrieved from What is API:
<https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/>

StevenSwiniarski. (2024, 04 18). *Codecademy*. Retrieved from POJO:
<https://www.codecademy.com/resources/docs/java/pojo>

Theil, D. (2024, 05 13). *LogRocket*. Retrieved from Understanding the waterfall methodology: <https://blog.logrocket.com/product-management/waterfall-methodology-guide/#:~:text=The%20waterfall%20methodology%20is%20a%20linear%20and%20sequential,well-defined%20project%20requirements%2C%20and%20a%20structured%2C%20pre-determined%20timeline.>

Weir, S. P. (2024, 04 22). *Wikipedia*. Retrieved from Data Transfer Object:
https://en.wikipedia.org/wiki/Data_transfer_object

CHAPTER 7: APPENDIX

7.1 Appendix A: Introduction

7.1.1 Current Scenario Additional information

Concurrently, in many developing nations like Nepal, food insecurity remains a pressing concern despite ample food resources. Limited infrastructure for storage, transportation and costly prices to some low-income sources people cannot afford food coupled with poor market linkages and awareness, exacerbates the problem of food wastage. Even as a significant portion of the population faces hunger and malnutrition, surplus food, often deemed unsuitable for sale due to cosmetic imperfections or nearing expiration dates, goes to waste. This paradox highlights the critical need for innovative solutions to bridge the gap between surplus food sources and vulnerable communities.

In Nepal, where 5.1% of the employed population lives below the poverty line of \$1.90 purchasing power parity/day, the challenge of food waste exacerbates existing inequalities. Effectively managing this surplus could significantly alleviate hunger and malnutrition in communities where access to food remains a daily challenge. Addressing this issue requires a multifaceted approach, including the implementation of efficient distribution channels, community education on food preservation and utilization, and the creation of systems that redirect surplus food to those in need. By ensuring that valuable resources are not needlessly discarded, while many go without adequate nutrition, these efforts can contribute to a more equitable and sustainable food system.

[Go back to the same page](#)

7.2 Appendix B: Background

7.2.1 About the End User

2.2.1.1 Web (Admin)

The admin interface is primarily accessed through a web browser. Administrators are responsible for managing and overseeing the entire system. Their tasks may include:

- Managing user accounts and permissions.
- Monitoring donation and distribution processes.
- Analyzing data and generating reports.
- Configuring system settings and parameters.

2.2.1.2 Mobile

Donor: Individuals or organizations who wish to donate food can use the mobile app to initiate and track their donations. Donors may perform tasks such as:

- Registering as a donor and providing necessary information.
- After verifying the account verified then log in to the system.
- Scheduling for food donations and donations, updates etc.
- Viewing donation history and receipts.
- Update his/her profile account details.
- Complain the volunteers if they cannot pick up the food.
- View donates food and updates information.

Volunteer: Volunteers play a crucial role in facilitating the collection and distribution of donated food. Their responsibilities may include:

- Registering as volunteers and specifying availability.
- Receiving notifications about donation pickups or distribution events.
- Updating the status of pickups or deliveries.
- Donation completed the send the email to the donor.
- Update his/her profile account details.

- Complaint to the donor if can't provide the food.
- View donated food details with Google Maps to the location.
- View donor profile details.

Admin: Admin who also used the mobile app for user account verification and report verification.

- Log in to the application on mobile devices.
- Verify the new user and give access or authentication for using the system.
- The donor and volunteer can complete or report verification where food donation time has any problem and give the punishment.
- Update the admin user account details.

Farmer: Farmers who produce surplus food can use the mobile app to coordinate with the platform for donations. Their tasks may involve:

- Registering as farmers and providing details about their produce.
- Indicating availability for donation.
- Communicating with donors or administrators regarding pickup logistics. the farmer can date expired food collected to be used for farming.
- The unhealthy food received from the donor.

Each end-user group has specific functionalities tailored to their role within the food share donation ecosystem, ensuring smooth operation and effective utilization of resources.

[Go back to the same page](#)

7.3 Appendix C: Methodology

7.3.1 Survey

7.3.1.1 Pre-Servay Form Questions

The pre-survey forma questions are given below:



Waste Food Management Survey Form

B I U ⊖ X

This survey will help in understanding user needs and preferences before designing or improving a waste food management system. This survey has been conducted as a part of the Final Year Project. The survey may take around a few minutes but is not compulsory if you are a helpful person and have free time please give the right or wrong response. Thank you.

Write your full name? *

Short-answer text

Write your email address? *

Short-answer text

Figure 196: Pre-survey form introduction with username

<p>Select your age? *</p> <p><input type="radio"/> 15-25</p> <p><input type="radio"/> 26-35</p> <p><input type="radio"/> 36-45</p> <p><input type="radio"/> 46 above</p>	
<p>Select your occupation? *</p> <p><input type="radio"/> Student</p> <p><input type="radio"/> Business</p> <p><input type="radio"/> Farmer</p> <p><input type="radio"/> Teacher</p> <p><input type="radio"/> Other</p>	

Figure 197: Pre-survey questions of age and occupation

In your life how often do you find yourself with excess food that could be donated? *

- Daily
- Weekly
- Monthly
- Rarely

Any social programs have you heard of or participated in for food donation? *

- Yes
- No

Figure 198: Pre-survey questions of what could be donated and participants to the event

In society why do so many people not donate food and instead waste it in a dump? *

- Lack of time
- Lack of information
- Not knowing where to donate
- Other reasons

If you are interested in food donate you prefer to donate food directly to individuals or through * organizations?

- Directly donation
- Through Organizations

Figure 199: Preservay questions about the food of dump or donate and directly donate or through organizations

In upcoming days you have extra food available are you interested in donating to homeless or *
poor people?

- Yes
- No

how would you rate an easy and quick online food donation process?(Rating Scale: 1-5) *

- 1
- 2
- 3
- 4
- 5

Figure 200: Pre-survey question of interest for donation and rating point of the system

If you like to track the impact of food donations what kind of impact tracking would interest you? *

- Location track
- Food Distributed information
- Others

How would you prefer to receive updates from a food donation app? *

- Contact
- Push notifications
- Email, SMS
- None

Do you have any ideas about food management that you could suggest for the food donation app? *

Short-answer text

Figure 201: Preservay questions of impact, updates, and suggestions for the system

7.3.1.2 Pre-Servay Form Filled Sample

The pre-survey forma filled sample is given below:

Select your age? *	<input checked="" type="radio"/> 15-25 <input type="radio"/> 26-35 <input type="radio"/> 36-45 <input type="radio"/> 46 above
Select your occupation? *	<input type="radio"/> Student <input type="radio"/> Business <input type="radio"/> Farmer <input type="radio"/> Teacher <input checked="" type="radio"/> Other

Figure 202: Pre-survey questions filled with a sample of age and occupation

In your life how often do you find yourself with excess food that could be donated? *

Daily
 Weekly
 Monthly
 Rarely

Any social programs have you heard of or participated in for food donation? *

Yes
 No

Figure 203: Pre-survey questions filled sample of what could be donated and participants to the event

In upcoming days you have extra food available are you interested in donating to homeless or poor people? *

Yes
 No

how would you rate an easy and quick online food donation process?(Rating Scale: 1-5) *

1
 2
 3
 4
 5

Figure 204: Preservay questions about the food-filled sample of dump or donate and directly donate or through organizations

In society why do so many people not donate food and instead waste it in a dump? *

Lack of time
 Lack of information
 Not knowing where to donate
 Other reasons

If you are interested in food donate you prefer to donate food directly to individuals or through organizations? *

Directly donation
 Through Organizations

Figure 205: Pre-survey questions filled sample of interest for donation and rating point of the system

If you like to track the impact of food donations what kind of impact tracking would interest you? *

Location track
 Food Distributed information
 Others

How would you prefer to receive updates from a food donation app? *

Contact
 Push notifications
 Email, SMS
 None

Do you have any ideas about food management that you could suggest for the food donation app? *

No

Figure 206: Preservay questions filled sample of impact, updates, and suggestions for the system

[Go back to the same page](#)

7.2.1.3 Post-Servay Form Questions

How satisfied are you with your experience using our surplus food donation system? *

...
...

- Very Satisfied
- Satisfied
- Neutral
- Bad
- Very Bad

Figure 207: Post-survey questions of user experience

Which role did you primarily use the surplus food donation system for? *

...
...

- Donor
- Volunteer
- Admin

Figure 208: Post survey questions of user role

How easy was it to navigate and use the surplus food donation system? *

- Very Easy
- Easy
- Neutral
- Difficult
- Very Difficult

Figure 209: Post survey question of system navigation of surplus food donations system

Which features of the surplus food donation system did you find most helpful? *

- Food Donation
- Push notification
- Google Map
- Donation Rating
- Overall Dashboard
- Others

Figure 210: Post-survey question of donations system which features is helpful

How likely are you to recommend our surplus food donation system to others? *

...
...

- Very Likely
- Likely
- Neutral
- Unlikely
- Very Unlikely

Figure 211: Post-survey of user recommend questions food donations system

Rate the ease and speed of an online food donation process from 1 to 5, with 1 being the lowest and 5 the highest. *

...
...

- 1
- 2
- 3
- 4
- 5

Figure 212: Post-survey questions of a rating point to the overall system

Thank you for taking the time to provide your valuable feedback! Your input will help us improve our surplus food donation system for all users. Are there any recommendation or improvements that would make the surplus food donation system more user-friendly or efficient? If so, please provide your answer.

Long-answer text

Figure 213: Post-survey of feedback optional questions

7.2.1.4 Post-Servay Form Filled sample

What is your occupation?

Developer

How satisfied are you with your experience using our surplus food donation system? *

Very Satisfied
 Satisfied
 Neutral
 Bad
 Very Bad

Figure 214: Post-survey questions filled sample of user experience

Which role did you primarily use the surplus food donation system for? *

Donor
 Volunteer
 Admin

Figure 215: Post survey questions filled sample of user role

How easy was it to navigate and use the surplus food donation system? *

- Very Easy
- Easy
- Neutral
- Difficult
- Very Difficult

Figure 216: Post survey question filled sample of system navigation of surplus food donations system

Which features of the surplus food donation system did you find most helpful? *

- Food Donation
- Push notification
- Google Map
- Donation Rating
- Overall Dashboard
- Others

Figure 217: Post-survey question filled sample of donations system which features is helpful

How likely are you to recommend our surplus food donation system to others? *

- Very Likely
- Likely
- Neutral
- Unlikely
- Very Unlikely

Figure 218: Post-survey filled sample of user recommend questions food donations system

Rate the ease and speed of an online food donation process from 1 to 5, with 1 being the lowest and 5 the highest.

- 1
- 2
- 3
- 4
- 5

*

Figure 219: Post-survey questions filled sample of a rating point to the overall system

Thank you for taking the time to provide your valuable feedback! Your input will help us improve our surplus food donation system for all users. Are there any recommendation or improvements that would make the surplus food donation system more user-friendly or efficient? If so, please provide your answer.

Very good work, Keep it up brother

Submitted 16/04/2024, 15:08

Figure 220: Post-survey of feedback optional filled sample

[Go back to the same page](#)

7.3.2 UML Diagram

7.3.2.1 Use Case

➤ User Registration

Description: Users can register on the system. The admin verifies the registered details. If the registration details are valid, a success message is displayed; otherwise, an error message is shown.

- The user provides registration details.
- Admin verifies the registration details.
- If the details are valid, a success message is displayed.
- If the details are not valid, an error message is shown.

➤ User Login

Description: Registered users can provide their login details to access the system. The system authenticates the login details. If authentication is successful, the user can log in; otherwise, an error message is displayed.

- The user provides login details.
- The system authenticates the login details.
- If authentication is successful, the user is logged in.
- If authentication fails, an error message is displayed.

➤ View History

Description: Users can view their donation history. The system verifies the request and shows the donation history details.

- The user requests to view donation history.
- The system verifies the request.
- The system displays the donation history details.

➤ **Donate Food**

Description: Donors can post new food donations on the system. Users can view the details of food donations. Volunteers accept donated food for distribution. After distribution, users give feedback by giving a donation rating for the food received.

- Donor posts new food donations.
- Users view food donation details.
- Volunteers accept donated food for distribution.
- After distribution, users provide feedback by giving a donation rating.

➤ **Complaints**

Description: Users can file complaints to the admin when Donated food is not provided during the designated time and no one comes to receive the donated food.

- The user files a complaint to the admin.
- Admin addresses the complaint.
- These use cases outline the various functionalities of the system and the flow of actions within each scenario.

[Go back to the same page](#)

7.3.2.2 Expanded Use Case

The expanded use case is given below:



Figure 221: Register expanded use case

Use Case:	Register Details
Actors:	Donor, Volunteer, NGO, Farmer
Descriptions:	All users can input their respective details into the system. The system automatically registers the provided information upon submission, ensuring seamless integration and efficient data management.
Typical Courses of Events:	
Donor, Volunteer, NGO, Farmer, Admin	System Response
1. A user can provide the personal details for the register in the system.	
	2. The system checks whether the provided details are valid or not.
3. Request for register in the system.	
	4. Conform register with a success message if no valid data, then show the error message.

Table 52: Register user expanded use case table

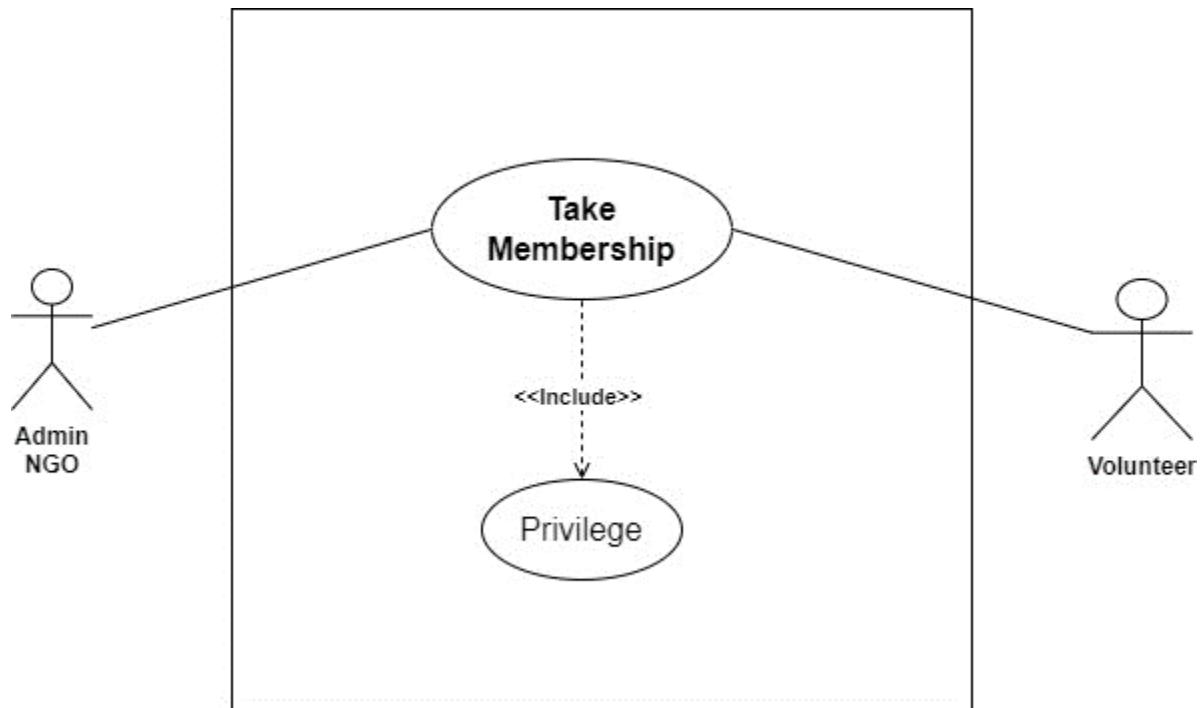


Table 53: Table membership Expanded use case

Use Case:	Take Membership
Actors:	Volunteer, NGO
Descriptions:	A new volunteer provides the personal details, and his/her details are registered with the system. The NGO provide the membership, and then volunteers take the new membership.
Typical Courses of Events:	
Volunteer, NGO	System Response
1. A new volunteer can provide the personal details for the registered membership in the NGO.	
	2. Check the Volunteer details.
3. Request to take membership in the NGO.	
	4. Confirm the registered membership and give the privilege.

Table 54: Take membership expanded use case table

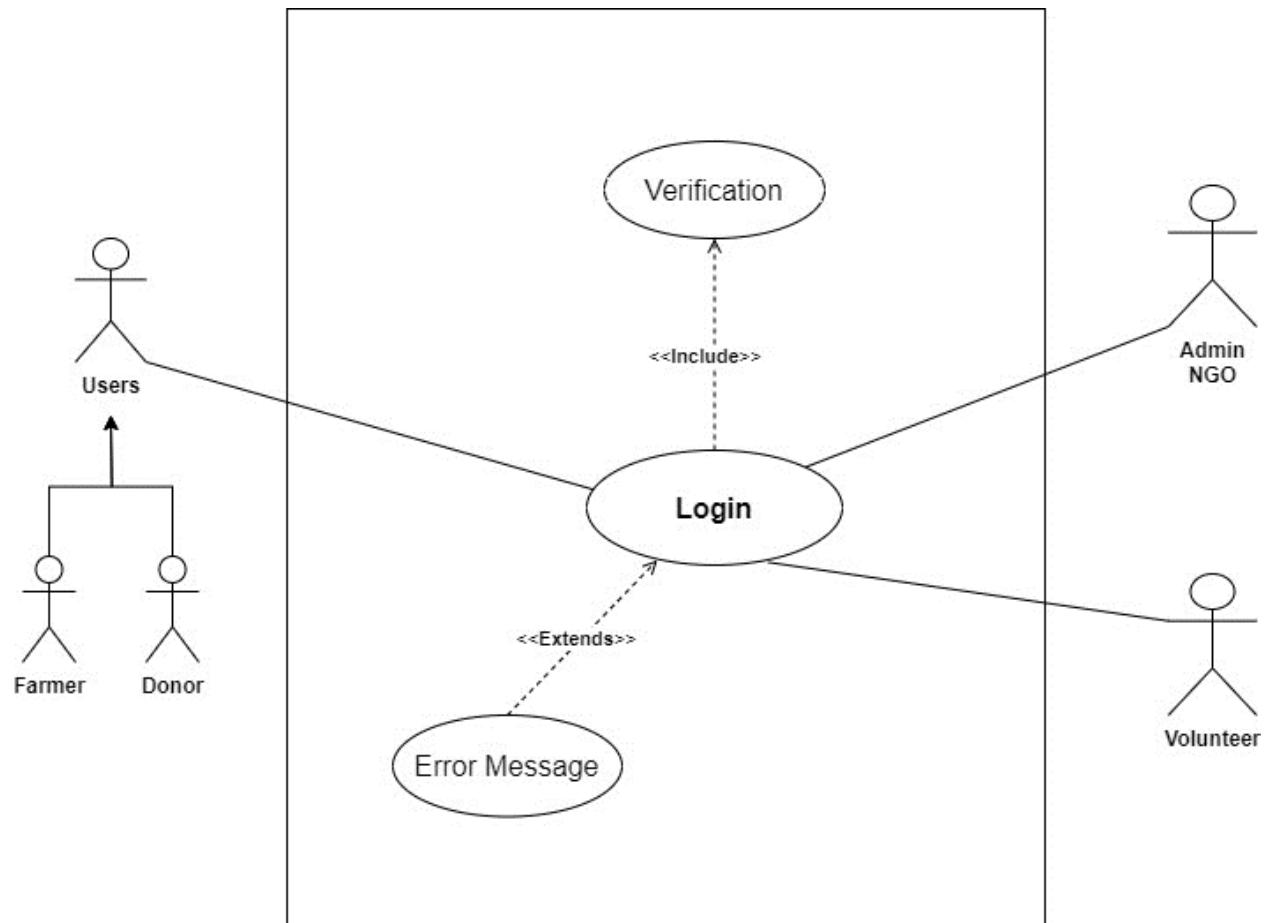


Table 55: Login expended use case diagram

Use Case:	Login
Actors:	Donor, Volunteer, Farmer
Descriptions:	After registering details in the system, all the users can provide valid details and log in to the system Then successfully log in to the system.
Typical Courses of Events:	
Donor, Volunteer, Farmer	System Response
1. A new user can provide the login details in the system.	
	2. The system checks whether the provided details are valid or not.
3. Request for login in the system.	
	4. The system can navigate to the dashboard if valid data is provided to log otherwise shows the error message.

Table 56: Login user expanded use case table

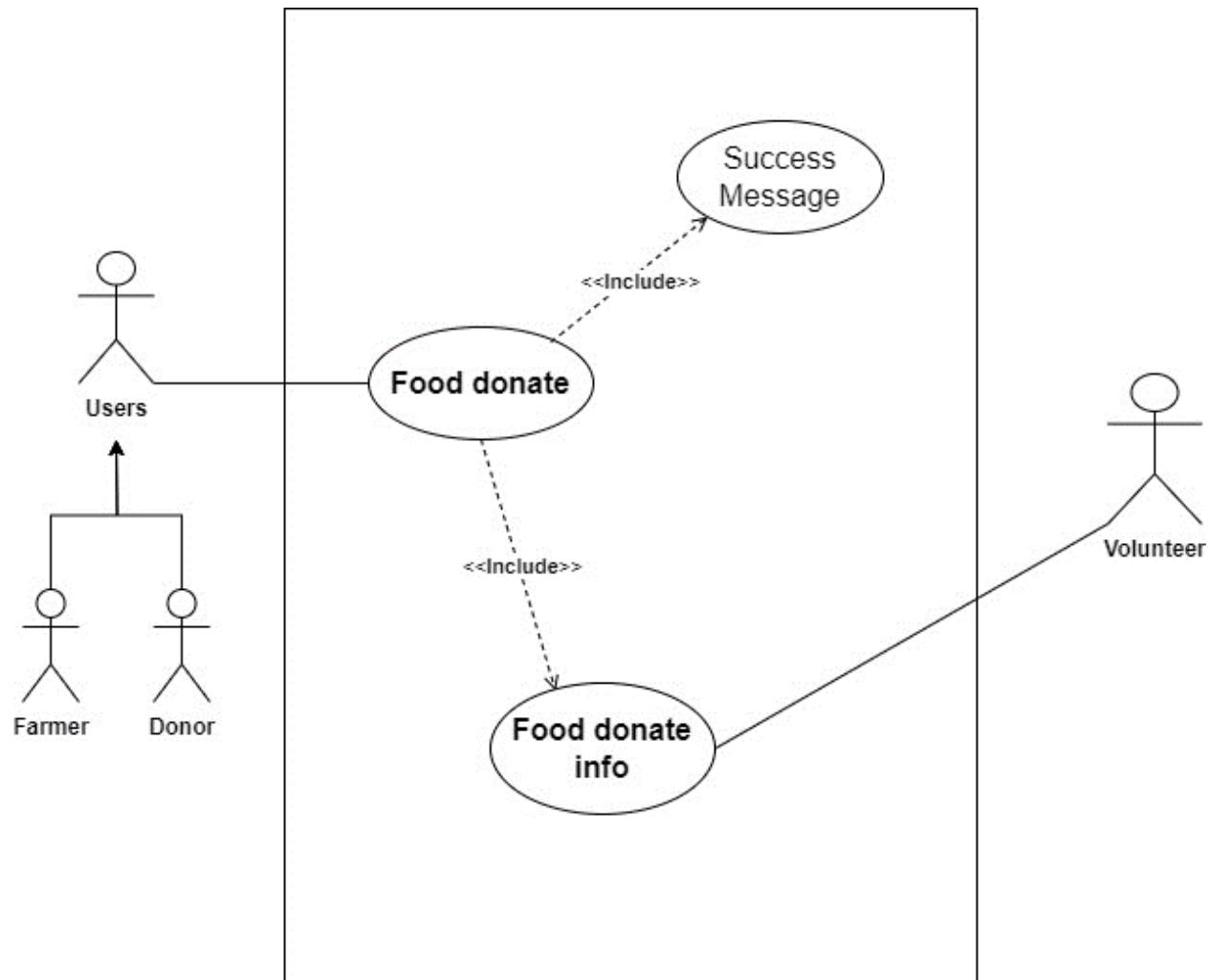


Table 57: Food donation expended use case diagram

Use Case:	Food Donation
Actors:	Donor, Farmer, volunteers
Descriptions:	The Donor or Farmer can donate the proper food information and details with location. The system can show the donation food details in the history after posting the donated food.
Typical Courses of Events:	
Donor, Farmer	Volunteers
1. A donor can post the food details for donation.	
	2. After getting the donation information volunteers respond to the request to accept the food.
3. Donors confirm to provide the food when contacted with the donor.	
	4. Volunteers receive the food and go to distribute it.

Table 58: Food donation user expanded use case table

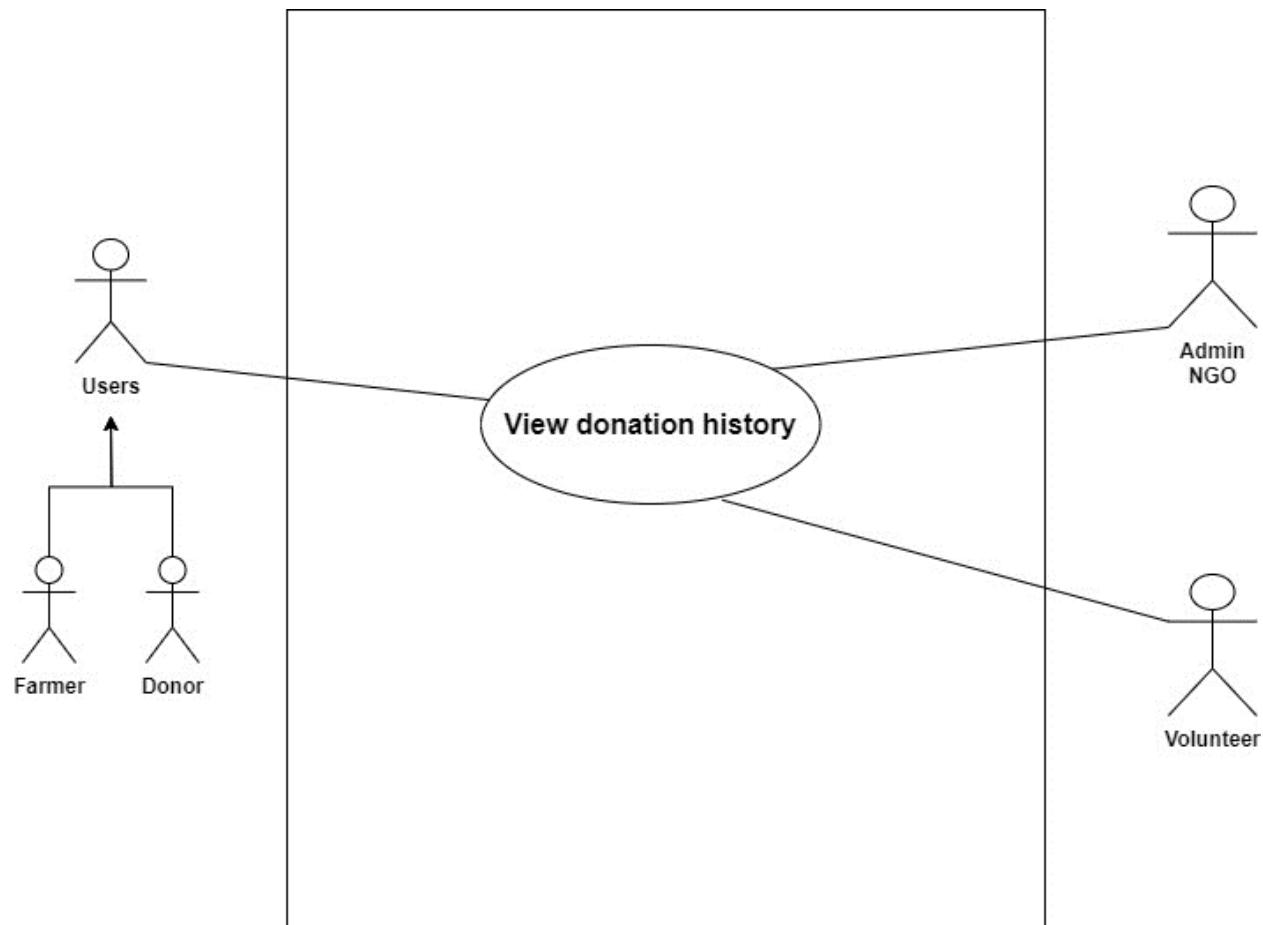


Table 59: View Donation history expended use case diagram

Use Case:	View Donation History
Actors:	Volunteer, Farmer
Descriptions:	All users can input their respective details into the system. The system automatically registers the provided information upon submission, ensuring seamless integration and efficient data management.
Typical Courses of Events:	
Volunteer, Farmer	System Response
1. A volunteer can view the donation post food details.	
	2. show the donation food details with all information like food, donor, and location details.
3. View the donated location.	
	4. The system can show the food donated location with Google Maps.

Table 60: View history user expanded use case table.

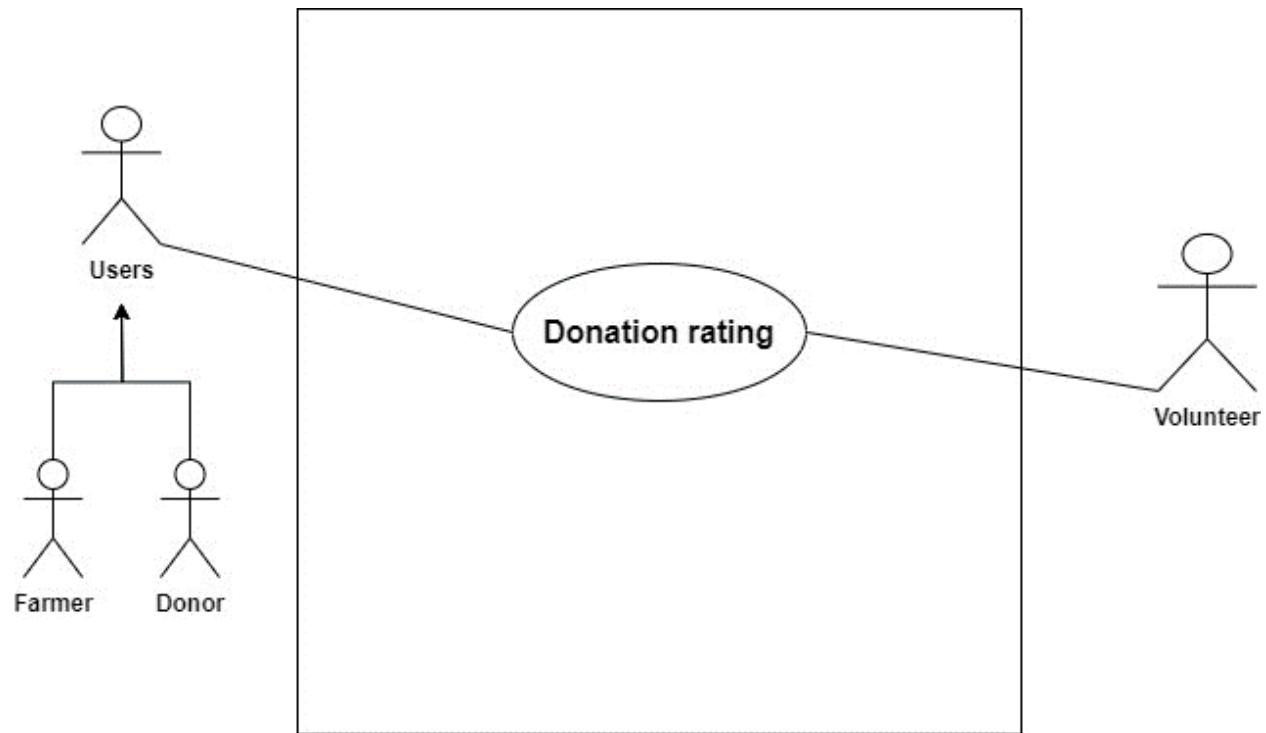


Table 61: Donation rating expanded use case diagram.

Use Case:	Donation Rating
Actors:	Donor, Volunteer, Farmer
Descriptions:	After the food is completely donated to some people the volunteer can give the donation rating to the donor with food distributed information.
Typical Courses of Events:	
Volunteer	Donor
1. After the food is received, give the donation rating with all the information.	
	2. Get the donated rating info.
3. Not possible to accept the donation of food if the volunteer cannot give a response.	
	4. Volunteers do not respond to receive the food and when the expiration date is over the donor gets the expiration notification.

Table 62: Donating rating expanded use case table

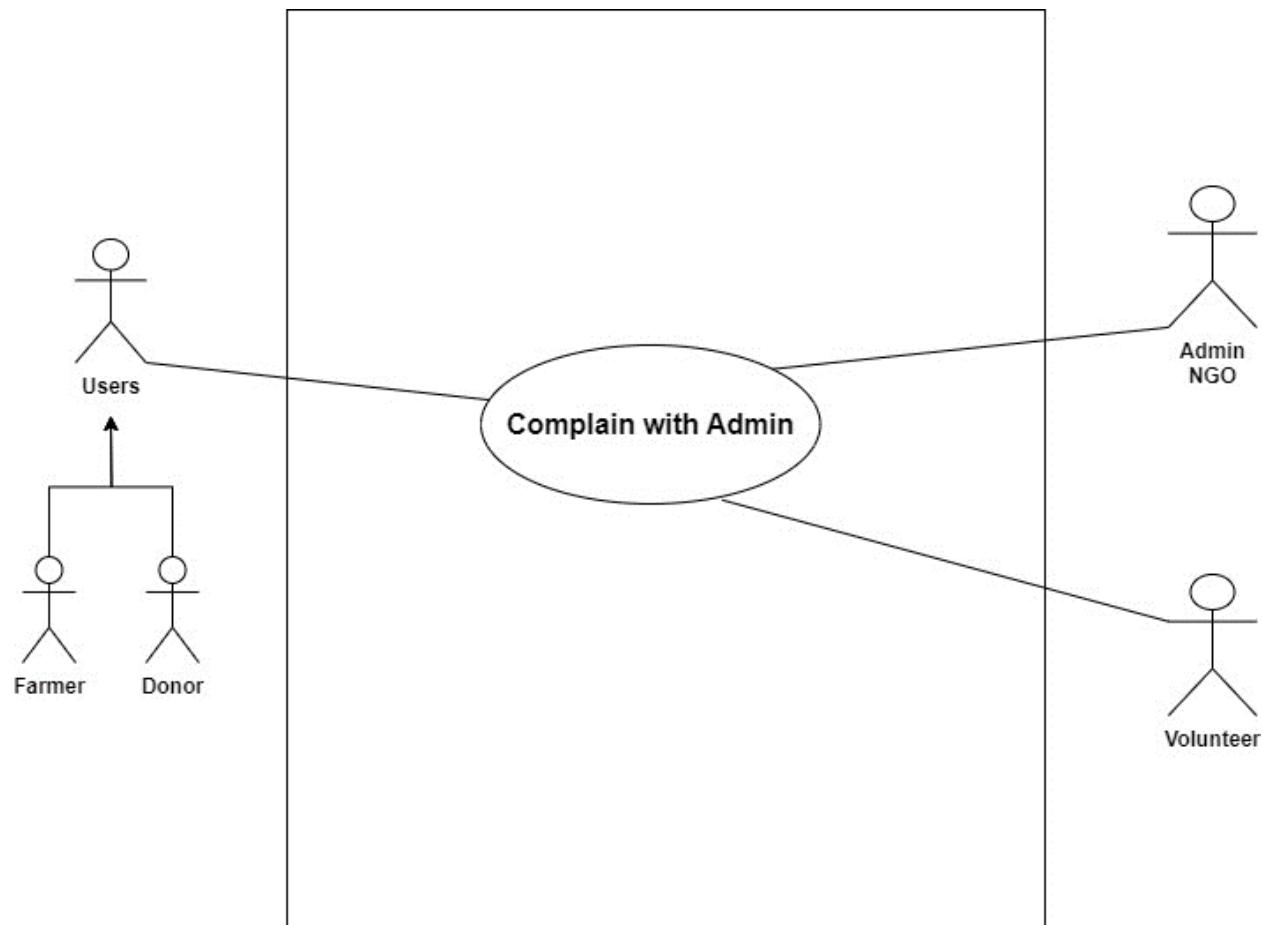


Table 63: Complaint with admin expanded use case diagram

Use Case:	Complain with Admin
Actors:	Donor, Volunteer, Farmer
Descriptions:	All users can input their respective details into the system. The system automatically registers the provided information upon submission, ensuring seamless integration and efficient data management.
Typical Courses of Events:	
Donor, Volunteer, Farmer	Admin Response
1. The food donation time donor volunteer and farmer can complain to the admin.	
	2. Admin verifies the complaint and gives the warning.
3. Received warning.	

Table 64: Complaint with admin expanded use case table

[\(Go back to the same page\)](#)

7.3.2 Sequence Diagram

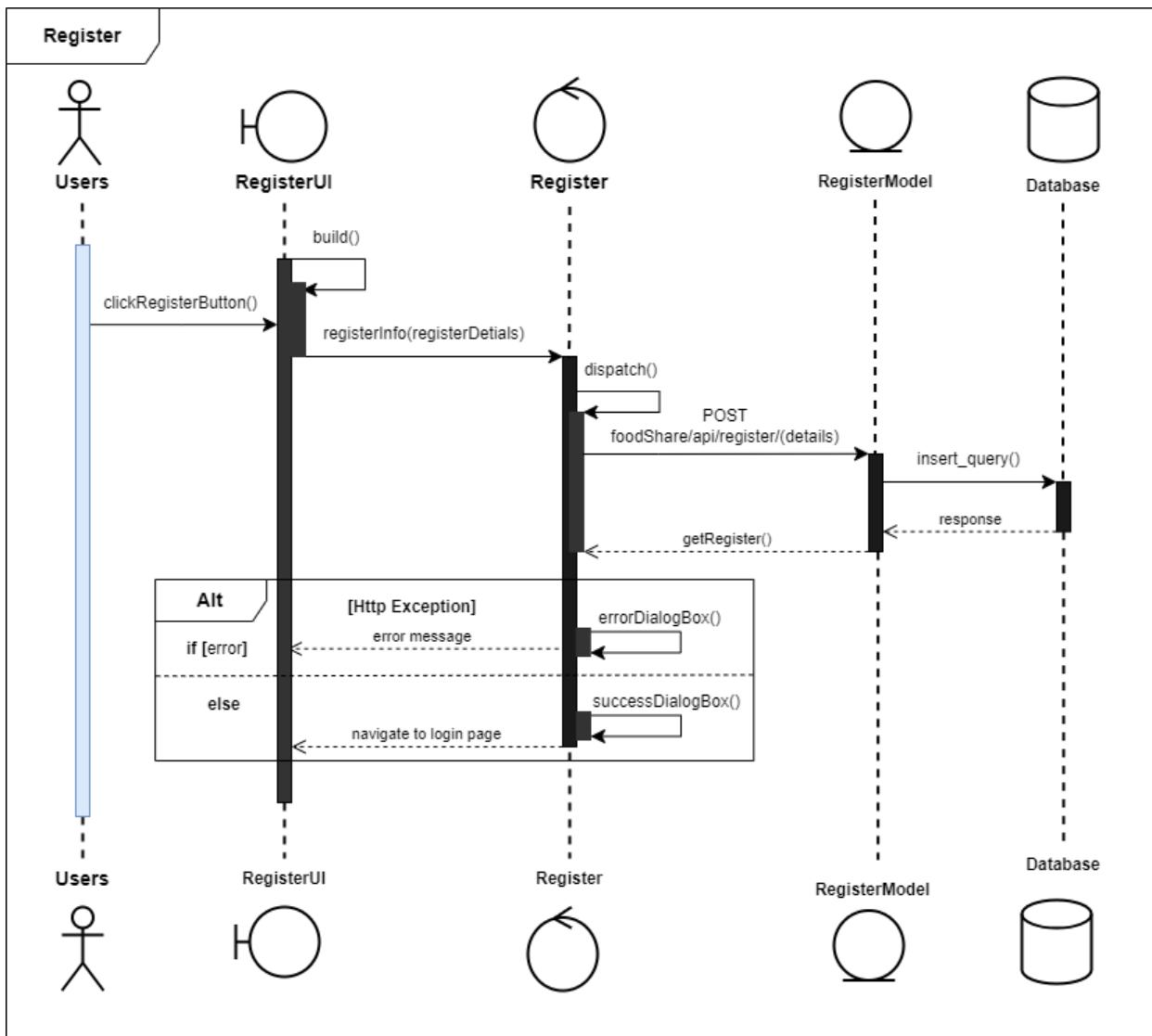


Figure 222: Sequence diagram of registered user

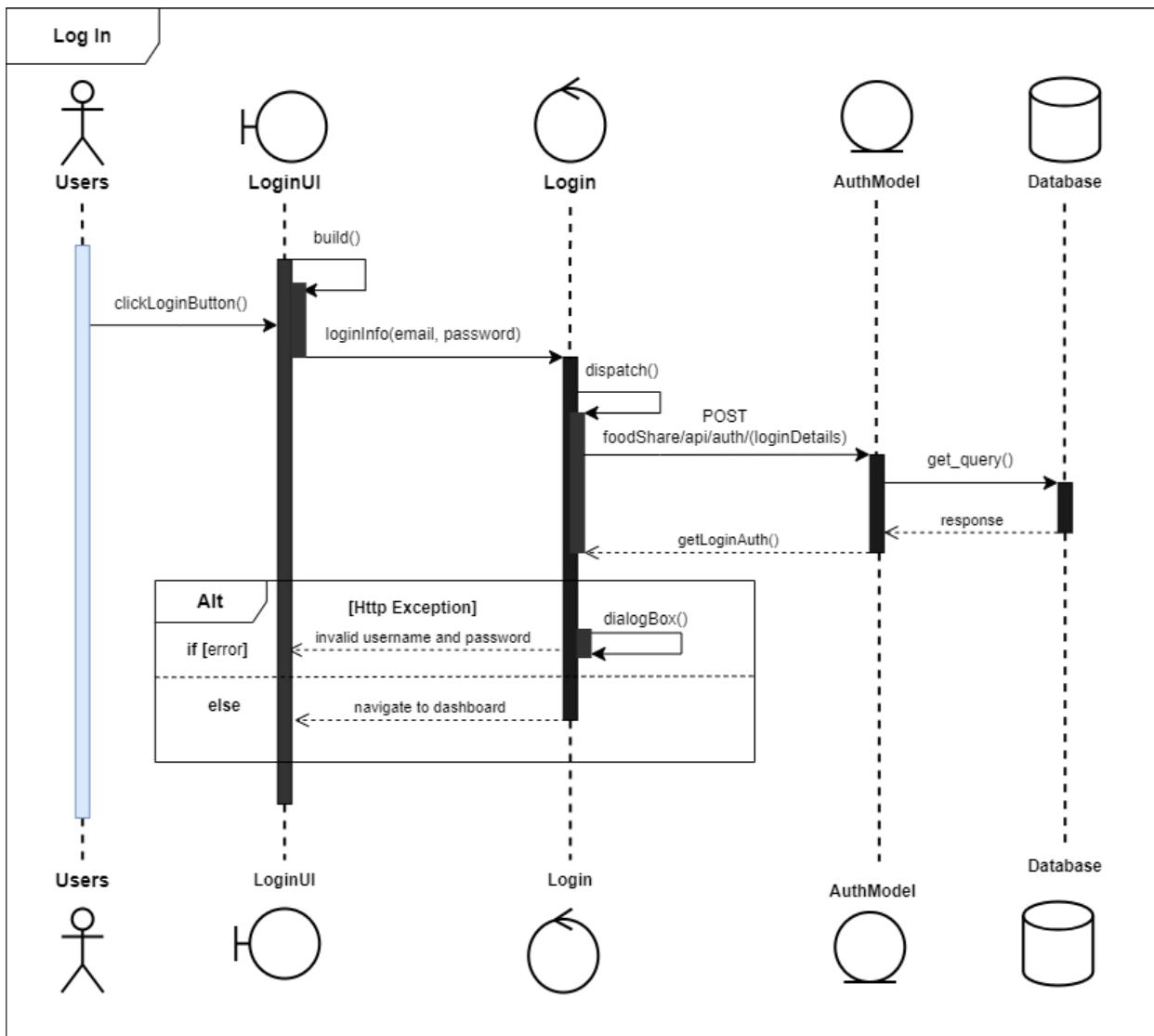


Table 65: Login sequence diagram

[\(Go back to the same page\)](#)

7.3.2.3 Activity Diagram

Login Activity

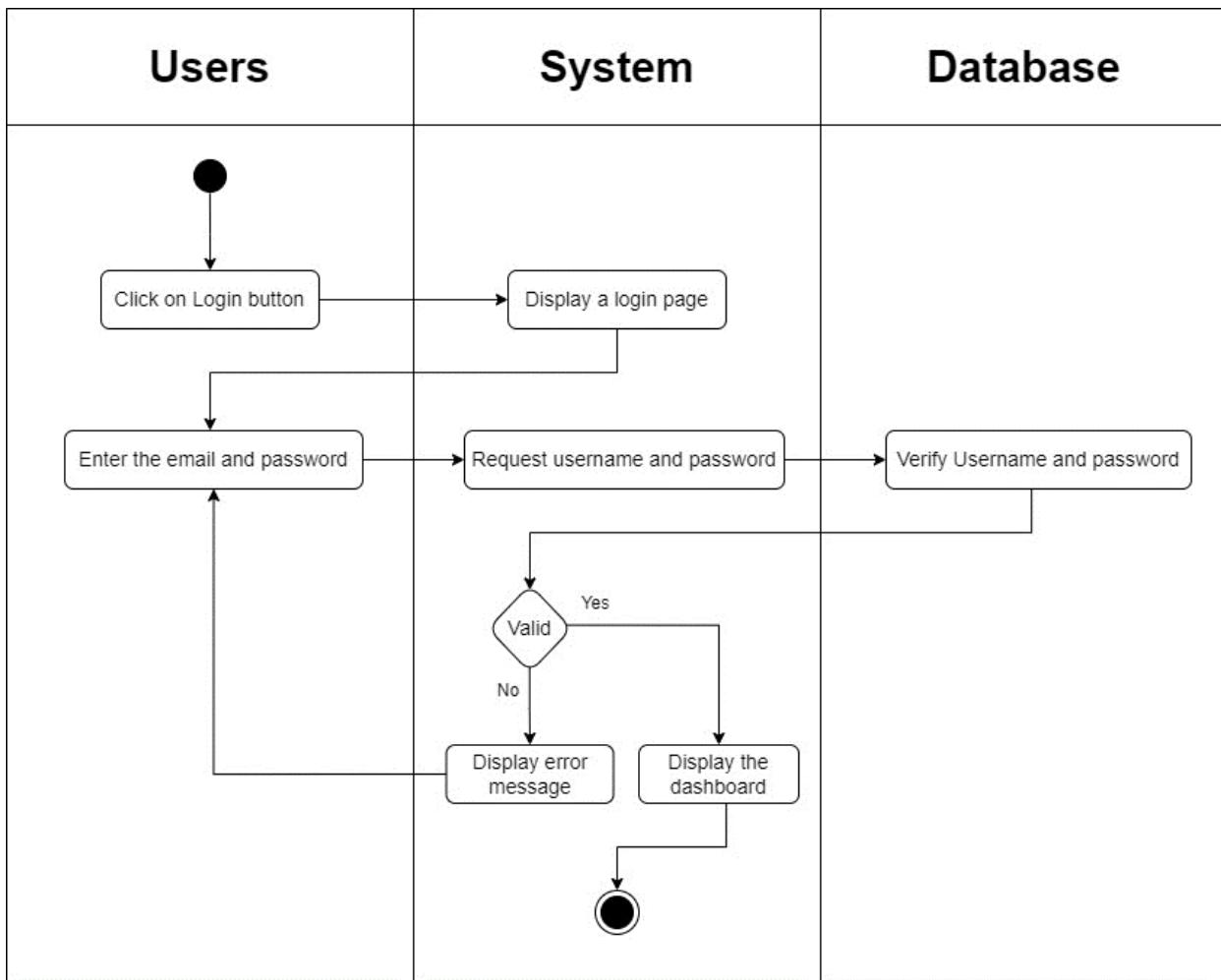


Figure 223: Login system activity diagram

Register Activity

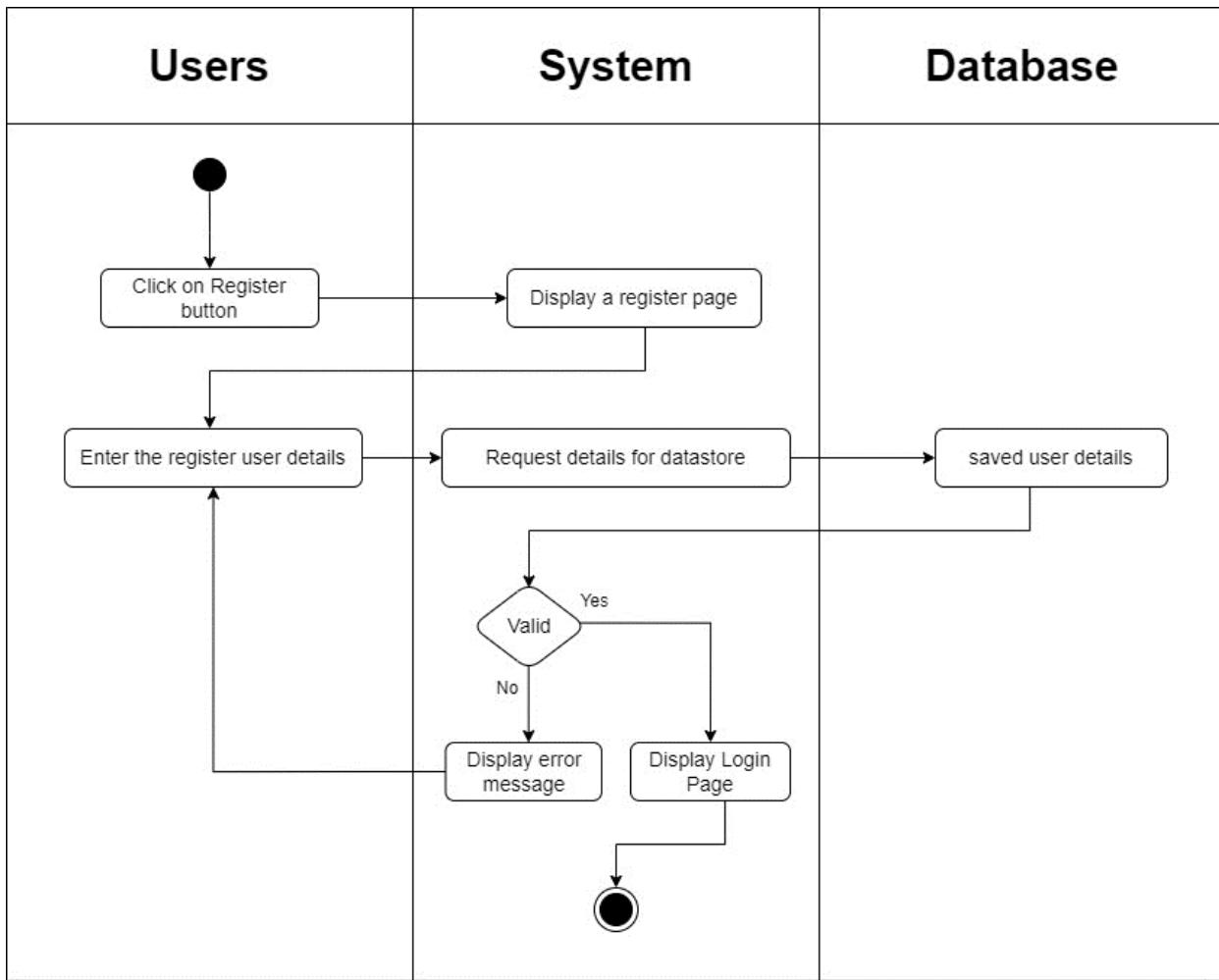


Figure 224: Register system activity diagram

Logout Activity

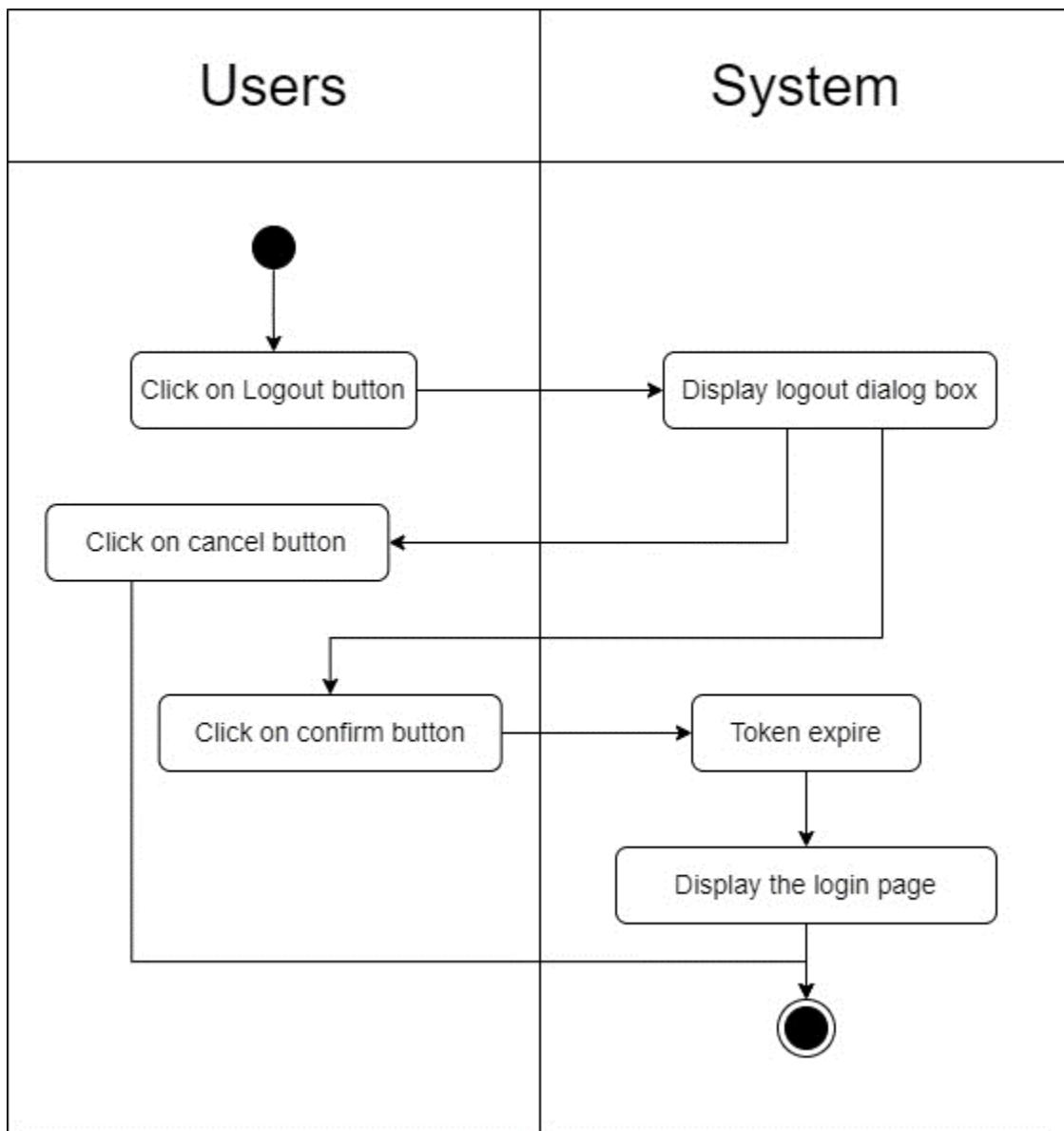


Figure 225: Logout system activity diagram

[\(Go back to the same page\)](#)

7.3.2.4 Wireframe UI

Splash screen

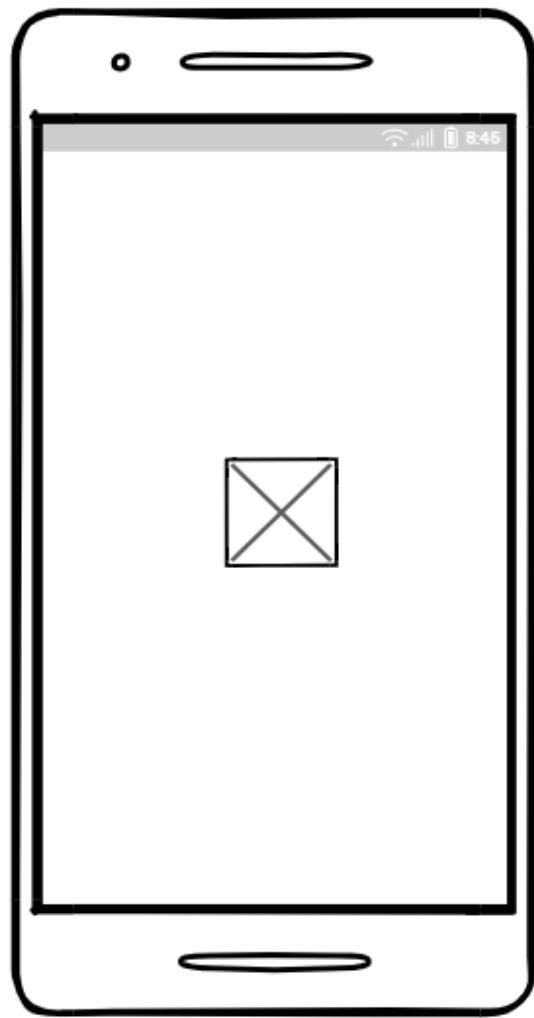


Figure 226: Splash screen mobile UI

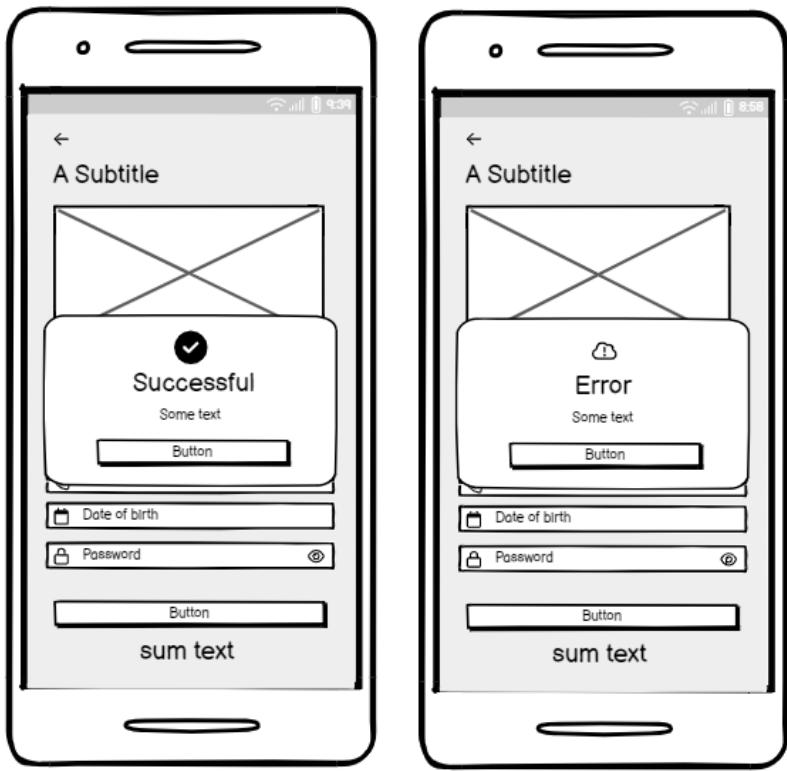


Figure 227: Register response message mobile UI

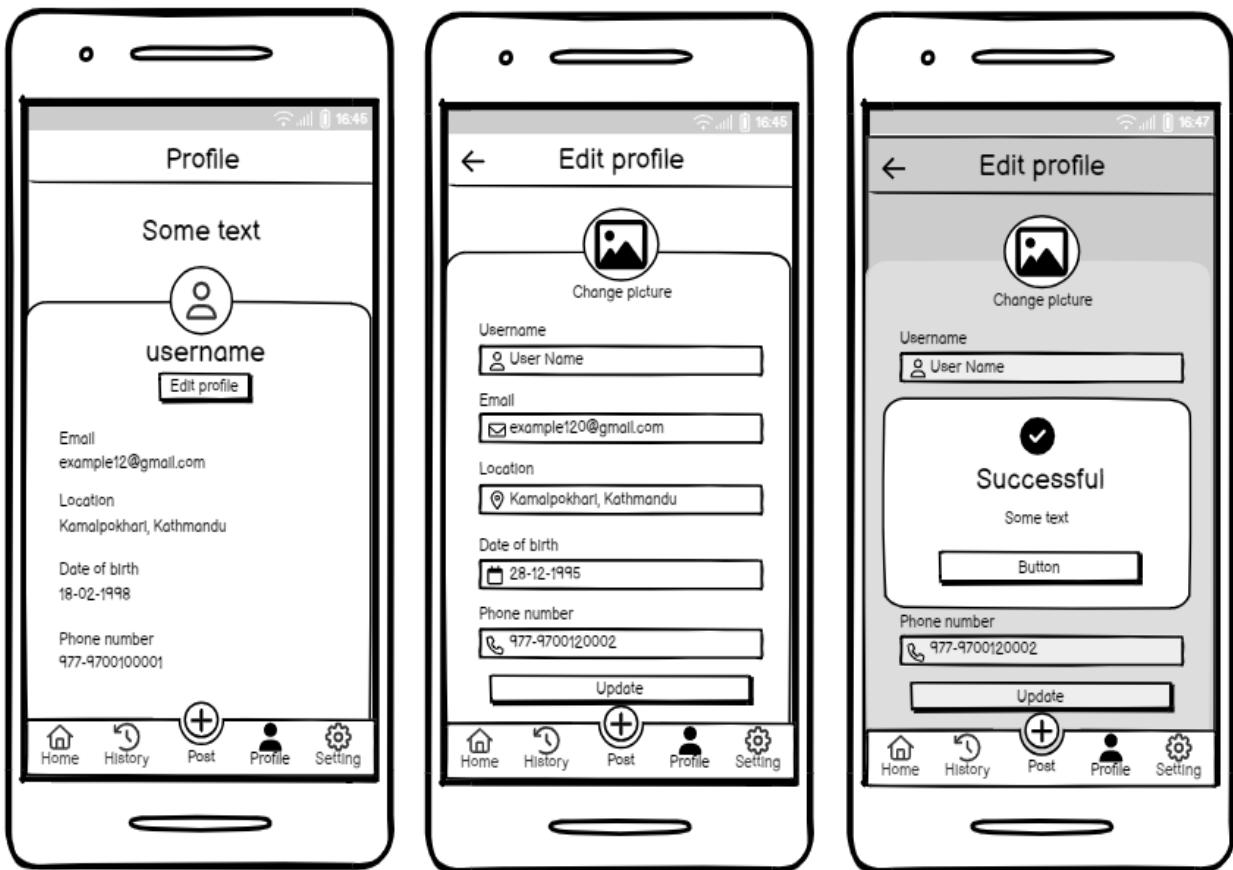


Figure 228: Donor Profile screen mobile UI



Figure 229: Donor Setting screen mobile UI



Figure 230: Volunteer Setting screen UI

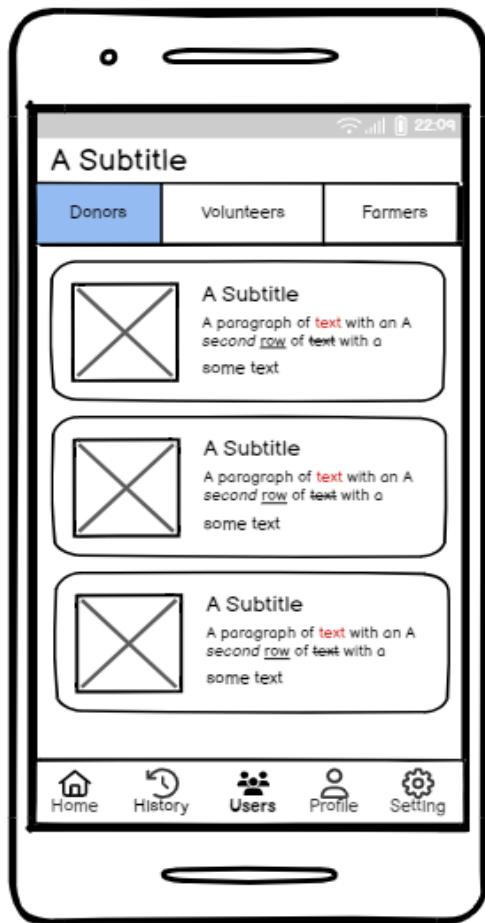


Figure 231: NGO Users screen UI

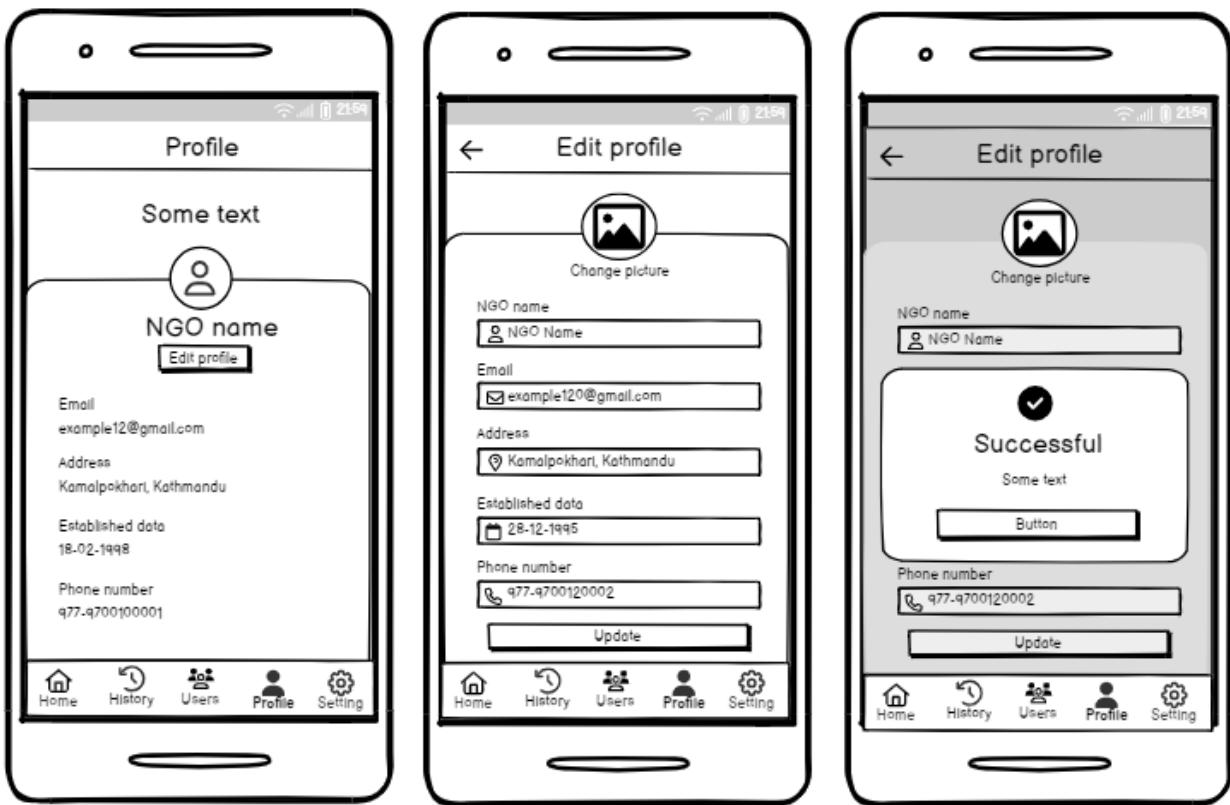


Figure 232: NGO Profile screen UI

[\(Go back to the same page\)](#)

7.3.2.5 Entity Relation Diagram

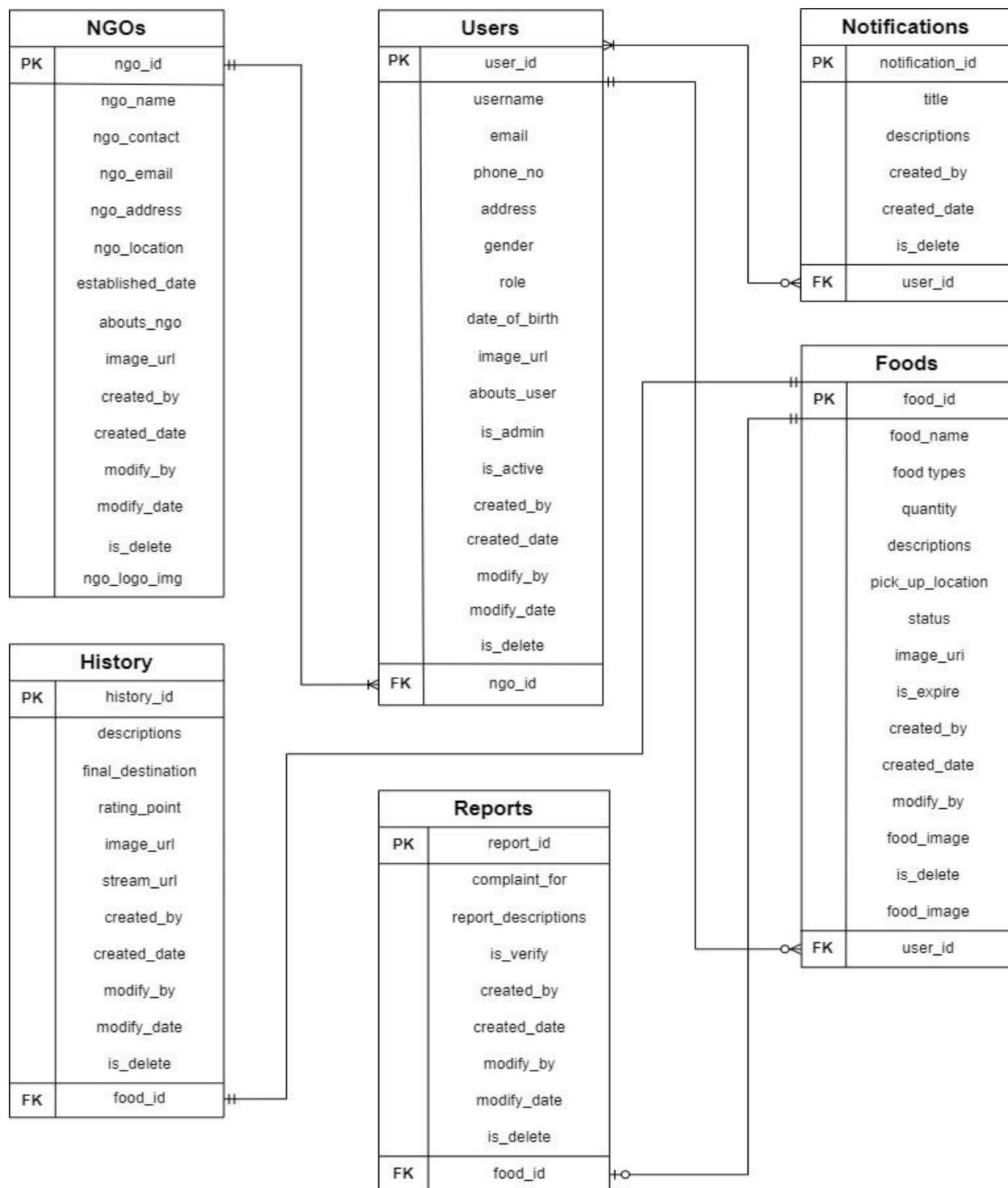


Figure 233: Initial or Actual ER-Diagram

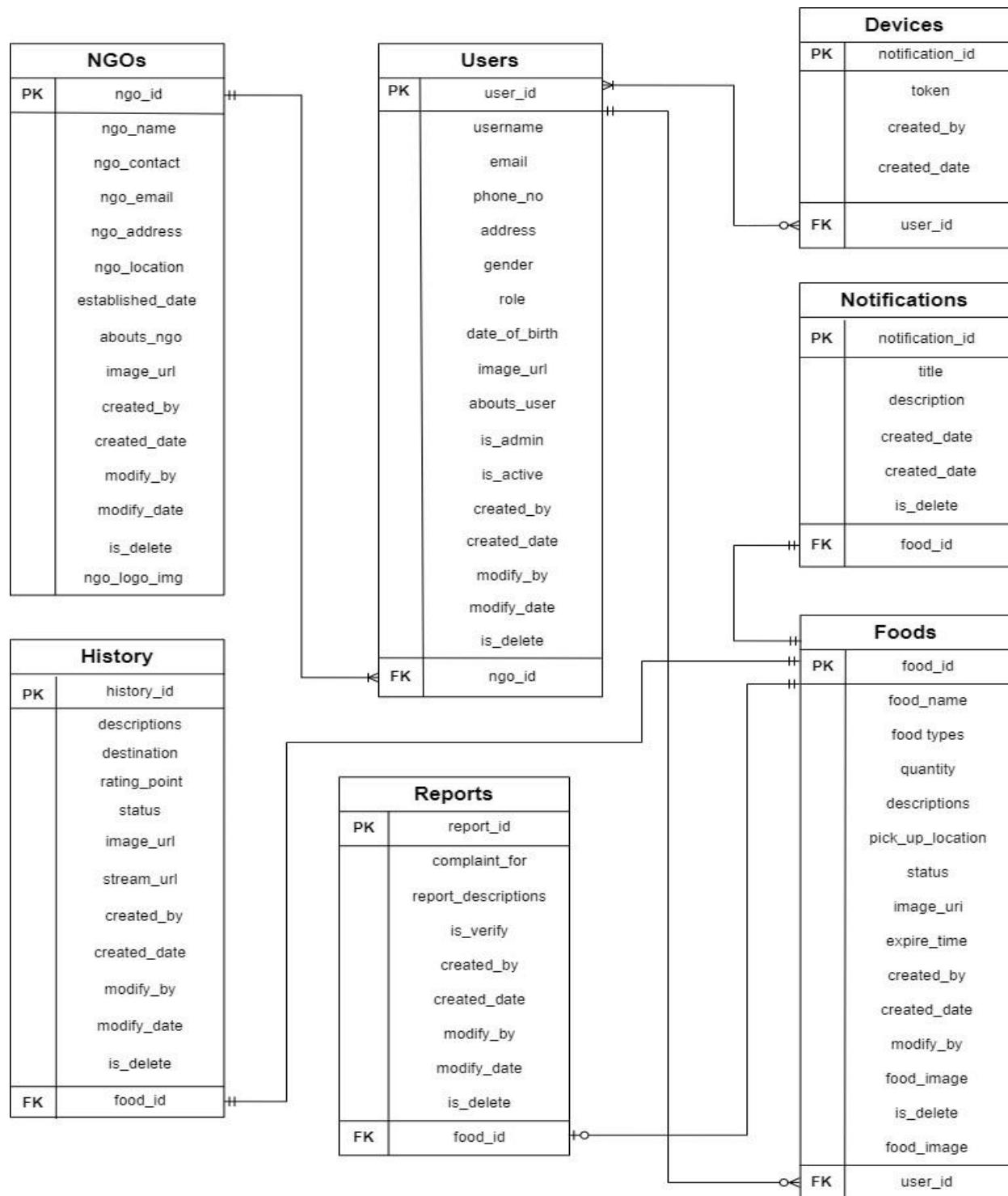


Figure 234: Final ER-Diagram

[Go back to the same page](#)

7.3.2.6 DFD Leve

Data flows are represented with arrows, indicating the direction of data movement between processes, data stores, and external entities. Each data flow is labelled to specify the type of data being transferred. Each process is represented by a rectangle with the process name written inside. Sub-processes are depicted by breaking down the main processes into smaller rectangles. By breaking down processes into sub-processes and illustrating data flows, Level 2 DFD enhances understanding of the system's functionality. It serves as a valuable communication tool between system designers, developers, and stakeholders, ensuring everyone has a clear understanding of the system's intricacies.

Register Details

The user can register details provided to the system by adding the registration details. The system receives the registration details and checks if they are valid. If the registration details are not valid, the system shows an error message indicating the invalid information. If the details are valid, the system records them in the database and provides a success message.

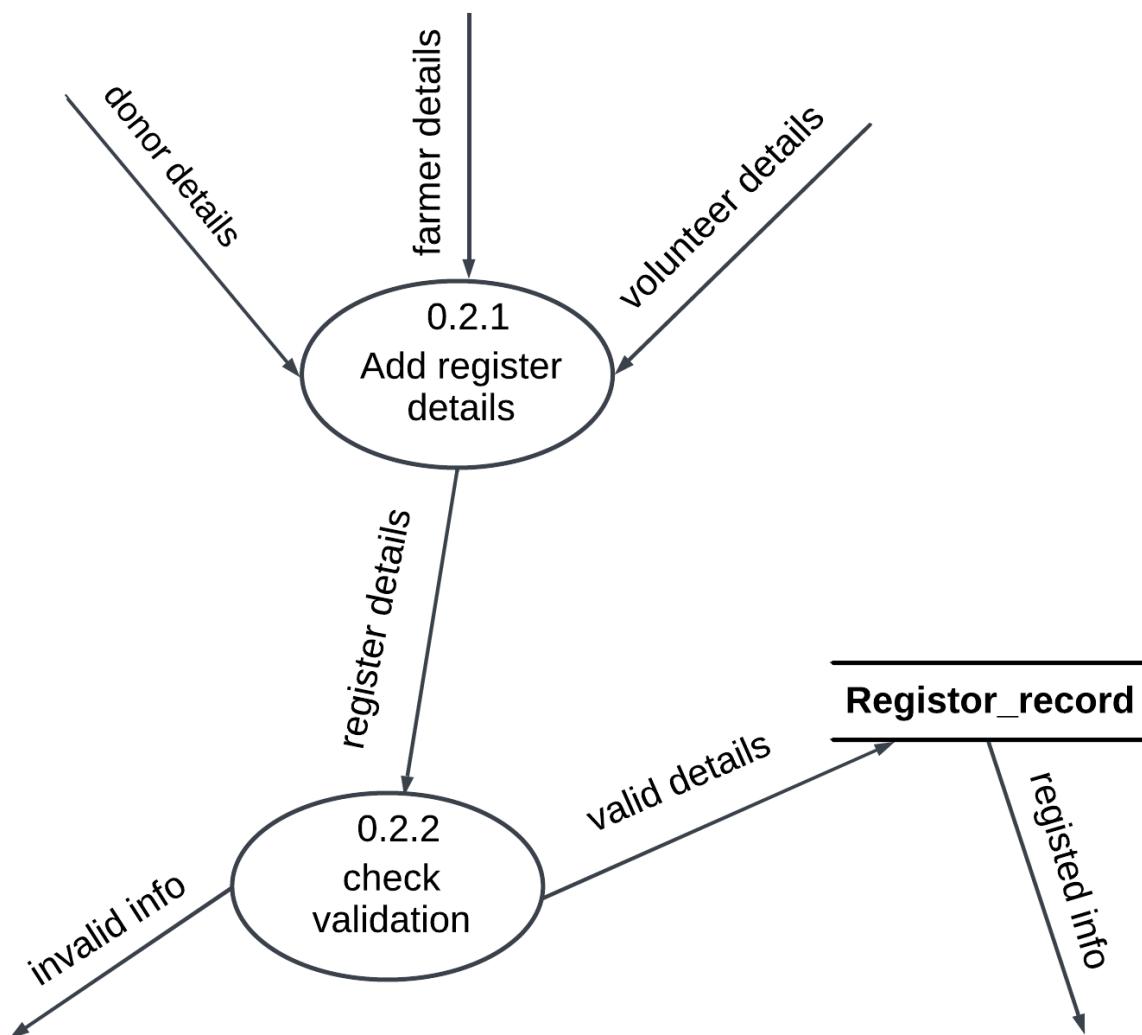


Figure 235: Register details DFD Level-2

Login System

The user can verify their registration account to log into the system using their email and password. The system checks the database to verify if the credentials are available. If the account has login authentication, the system displays a success message, and the user can navigate to the dashboard.

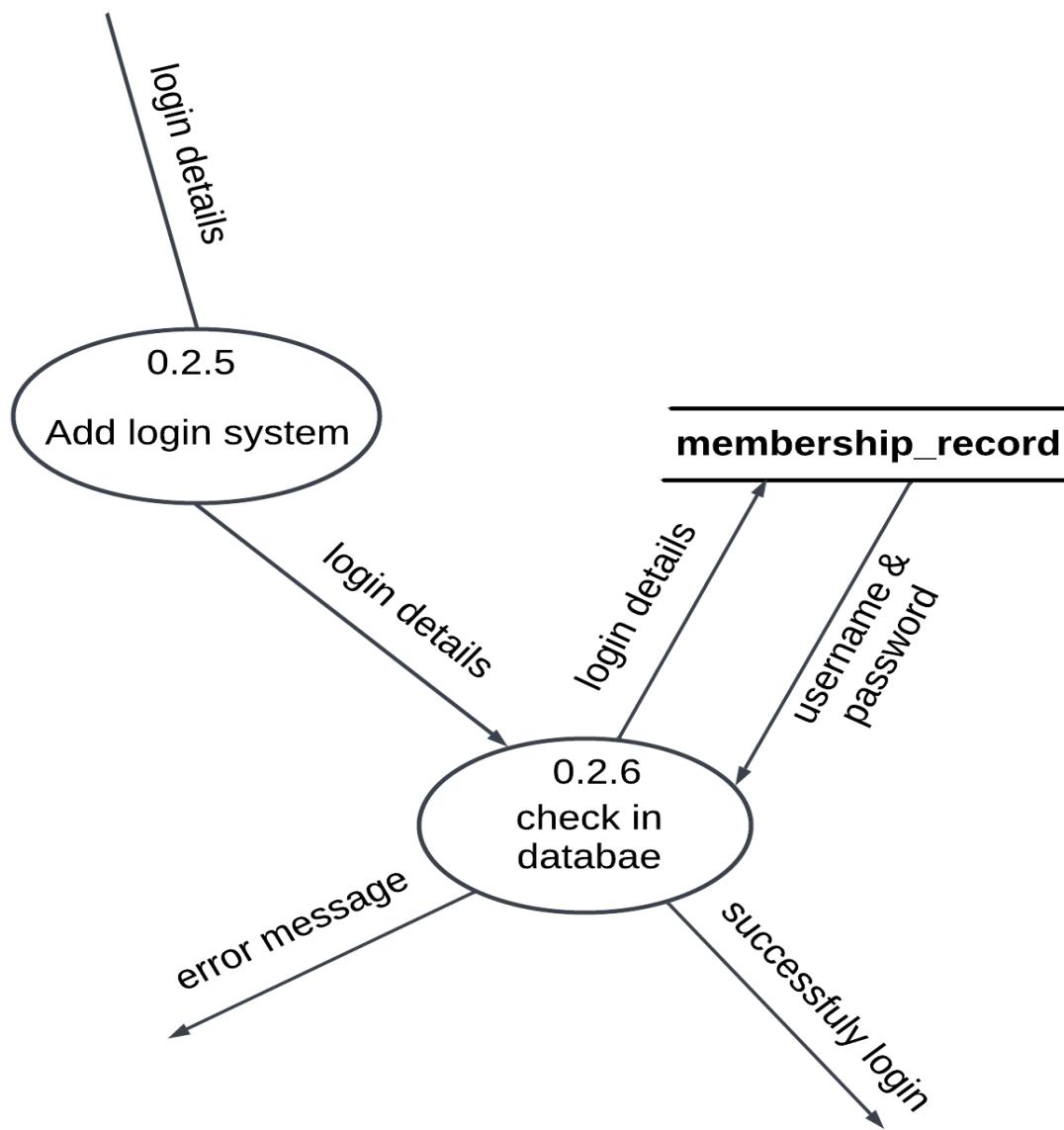


Figure 236: Login details DFD Level-2

[\(Go Back to the same page\)](#)

7.3.2.7 More features implementations

➤ Implementations of Login Features

```

76     # login authentication
77     class LoginUser(APITView):
78         authentication_classes = [TokenAuthentication]
79         permission_classes = [AllowAny]
80
81     def post(self, request):
82         try:
83             email = request.data.get('email')
84             password = request.data.get('password')
85
86             if email and password:
87                 auth_user = authenticate(request, username=email, password=password)
88                 if auth_user:
89                     user = Users.objects.filter(email=email).first()
90                     if auth_user.is_active and not user.is_delete: # Fix the condition
91                         refresh = RefreshToken.for_user(auth_user)
92                         access_token = str(refresh.access_token)
93
94                         serializer = UserSerializer(user)
95                         profile = user.photo_url.url if user.photo_url else None
96
97                         response_auth = {
98                             'id': serializer.data['id'],
99                             'username': serializer.data['username'],
10                             'email': serializer.data['email'],
11                             'profile': profile,
12                             'role': serializer.data['role'],
13                             'access_token': access_token,
14                         }
15                         return Response({'message': 'Login successful', 'is_success': True, 'status': 200, "auth": response_auth})
16                     else:
17                         return Response({'message': 'Your account is not activate', 'is_success': False, 'status': 401})
18                 else:
19                     return Response({'message': 'The account does not have authentication permission.', 'is_success': False, 'status': 401})
20             else:
21                 return Response({'message': 'Please provide email and password', 'is_success': False, 'status': 400})
22         except Exception as e:
23             return Response({'message': "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414

```

```
class LoginRepositoryImpl(private val apiService: ApiService) : LoginRepository {

    // login authentications
    override suspend fun getLoginUserAuth(email: String, password: String): LoginAuthPojo? {
        val loginModel = LoginModelDTO(email = email, password = password)
        return apiService.getLoginUserAuth(loginModel)
    }

    override suspend fun setDeviceFcmTokenSave(userId: Int?, fcmToken: String?, userName: String?): ResponsePojo? {
        return apiService.deviceFcmTokenSave(userId, fcmToken, userName)
    }
}
```

Figure 239: Login Repository Implementation

```
class LoginUseCase(private val loginRepository: LoginRepository) {

    operator fun invoke(email: String, password: String): Flow<Resource<LoginAuthPojo>> = flow {
        emit(Resource.Loading())
        try {
            val result = loginRepository.getLoginUserAuth(email, password)
            if (result?. isSuccess == true) {
                emit(Resource.Success(data = result))
            } else {
                emit(Resource.Error(message = result?. message))
            }
        } catch (e: Exception) {
            emit(Resource.Error(message = "Unable to connect to the server."))
        }
    }
}
```

Figure 240: Login user use-case implementation.

```
👤 Sita Ram Thing
    fun getLoginUserAuth(email: String, password: String) {
        _signInState = LogInState(isLoading = true)
        loginUseCase.invoke(email.trim(), password.trim()).onEach { result ->
            _signInState = when (result) {
                is Resource.Loading -> {
                    LogInState(isLoading = true)
                }

                is Resource.Success -> {
                    LogInState(data = result.data)
                }

                is Resource.Error -> {
                    LogInState(error = result.data?.message ?: result.message)
                }
            }
        }.launchIn(viewModelScope)
    }
}
```

Figure 241: Login authentication view model implementation.

➤ Implementations of Register

```

4
5  # Register user |
6  class RegisterUser(APIView):
7      authentication_classes = [TokenAuthentication] # Ensure TokenAuthentication is included
8      permission_classes = [AllowAny]
9
10     def post(self, request):
11         try:
12             serializer = UserSerializer(data=request.data)
13             if serializer.is_valid():
14                 # Extract validated data from the serializer
15                 email = serializer.validated_data['email']
16                 username = serializer.validated_data['username']
17                 role = serializer.validated_data['role']
18                 password = request.data.get('password')
19                 # Register user
20                 try:
21                     Users.objects.create_user(email=email, username=username, role=role, password=password)
22                     return Response({'message': 'User registered successfully', 'is_success': True, 'status': 200, 'system_token': settings.SECRET_KEY})
23                 except Exception as e:
24                     return Response({'message': "Invalid user", 'is_success': False, 'status': 500})
25             else:
26                 return Response({'message': 'Enter a valid email address!', 'is_success': False, 'status': 400})
27         except Exception as e:
28             return Response({'message': "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
```

```
  Sita Ram Thing
  fun registerUser(email: String, username: String, role: String, password: String) {
    _registerState = RegisterState(isLoading = true)
    val registerModelDAO = RegisterModelDAO(email.trim(), username.trim(), password.trim(), role.trim())
    registerUseCase.invoke(registerModelDAO).onEach { result ->
      _registerState = when (result) {
        is Resource.Loading -> {
          RegisterState(isLoading = true)
        }

        is Resource.Success -> {
          RegisterState(data = result.data)
        }

        is Resource.Error -> {
          RegisterState(error = result.data?.message ?: result.message)
        }
      }
    }.launchIn(viewModelScope)
  }
```

Figure 245: Register user view model implementations.

➤ Implementations of Forgot password.

```
# update user account email and password
class UpdatePassword(APIView):
    authentication_classes = [TokenAuthentication]
    permission_classes = [AllowAny]

    def patch(self, request):
        try:
            email = request.data.get('email')
            new_password = request.data.get('new_password')

            if not email:
                return Response({"message": "The user email is empty!", "is_success": False, 'status': 400})

            user = Users.objects.filter(email=email).first()
            if not user:
                return Response({"message": "User not found", "is_success": False, 'status': 404})

            if user is not None:
                if new_password != '' and isinstance(new_password, str):
                    user.set_password(new_password)
                    user.modify_by = user.username
                    user.modify_date = datetime.now().strftime('%Y-%m-%d')
                    user.save()
                return Response({"message": "The password has been successfully updated.", "is_success": True, 'status': 200})

            return Response({"message": "Both email and password are empty, no changes made", "is_success": False, 'status': 400})

        except Exception as e:
            return Response({"message": "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
```

Figure 246: Update password Api view implementations.

```
// Forgot password
➊ Sita Ram Thing
@PATCH(ApiUrl.UPDATE_PASSWORD)
suspend fun forgotPassword(@Body request: ForgotPasswordDAO?): ResponsePojo?
```

Figure 247: Forgot password API call implementation.

```
class ForgotPasswordRepositoryImpl(private val apiService: ApiService) : ForgotPasswordRepository {  
    override suspend fun getVerifyEmail(email: String): ResponsePojo? {  
        return apiService.getVerifyEmail(email = email)  
    }  
  
    override suspend fun getUpdatePassword(email: String?, password: String?): ResponsePojo? {  
        return apiService.forgotPassword(ForgotPasswordDAO(email = email, newPassword = password))  
    }  
}
```

Figure 248: Forgot password repository implementation.

```
operator fun invoke(email: String?, password: String?): Flow<Resource<ResponsePojo?>> = flow {  
    emit(Resource.Loading())  
    try {  
        val result = forgotPasswordRepository.getUpdatePassword(email, password)  
        if (result?.isSuccess == true){  
            emit(Resource.Success(data = result))  
        } else {  
            emit(Resource.Error(message = result?.message))  
        }  
    } catch (e: Exception) {  
        emit(Resource.Error(message = "Not found!"))  
    }  
}
```

Figure 249: Forgot password use case implementation.

```
// Forgot password
@Sita Ram Thing
fun setForgotPassword(email: String?, password: String?) {
    forgotPasswordUseCase.invoke(email?.trim(), password?.trim()).onEach { result ->
        _forgotPasswordState = when (result) {
            is Resource.Loading -> {
                ForgotPasswordState(isLoading = true)
            }

            is Resource.Success -> {
                ForgotPasswordState(data = result.data)
            }

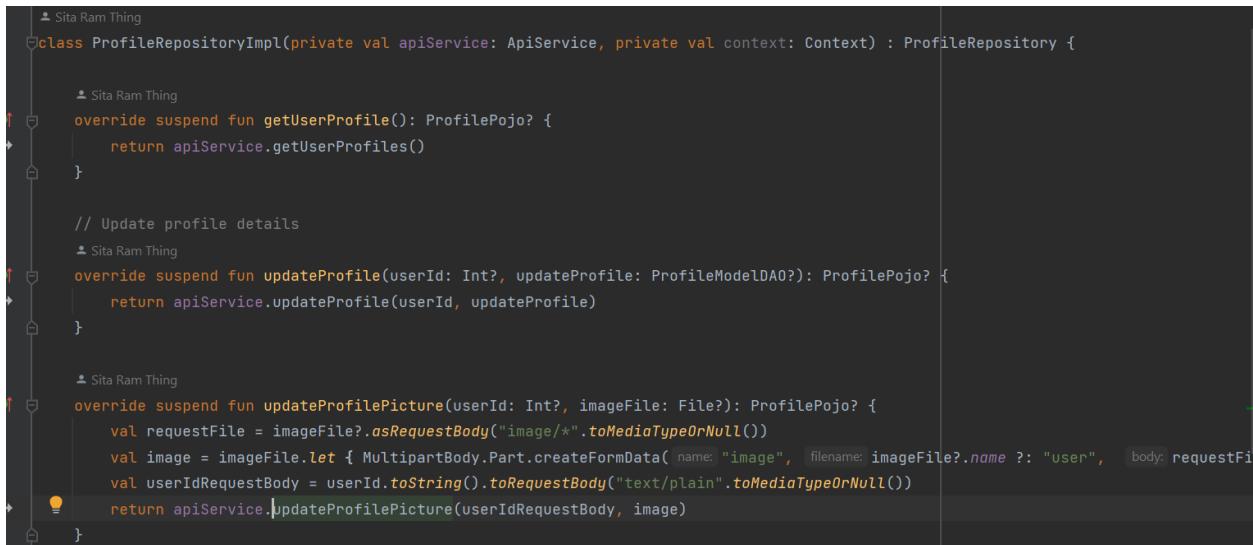
            is Resource.Error -> {
                ForgotPasswordState(isError = result.message)
            }
        }
    }.launchIn(viewModelScope)
}
```

Figure 250: Forgot password view model implementation.

➤ Implementations of Updated profile

```
// Update profile image
@Sita Ram Thing
@Multipart
@PATCH(ApiUrl.UPDATE_PROFILE_IMAGE)
suspend fun updateProfilePicture(
    @Part("id") id: RequestBody?,
    @Part image: MultipartBody.Part?,
): ProfilePojo?
```

Figure 251: Update profile API call



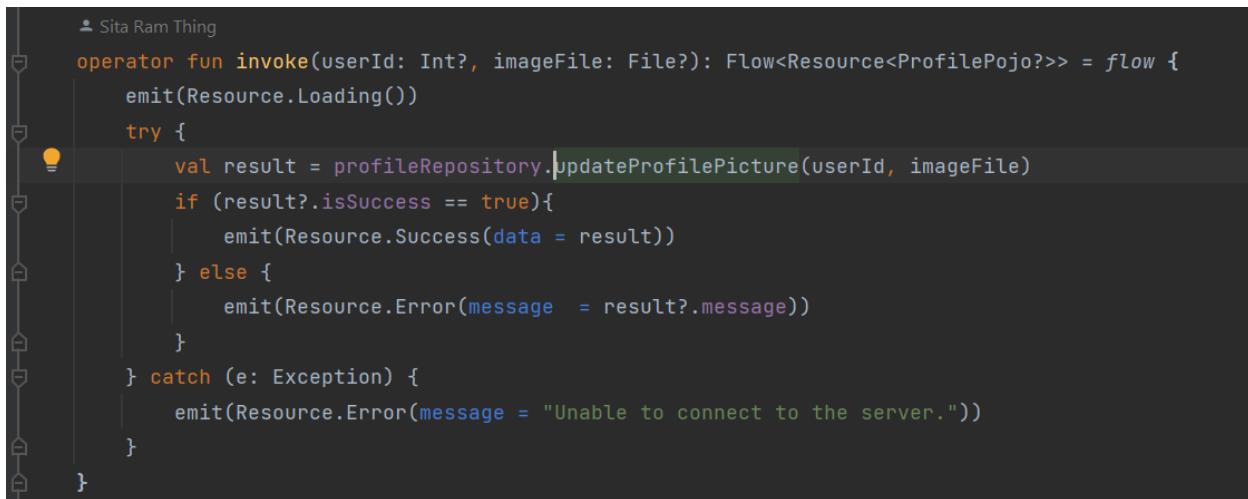
```
class ProfileRepositoryImpl(private val apiService: ApiService, private val context: Context) : ProfileRepository {

    override suspend fun getUserProfile(): ProfilePojo? {
        return apiService.getUserProfiles()
    }

    // Update profile details
    override suspend fun updateProfile(userId: Int?, updateProfile: ProfileModelDAO?): ProfilePojo? {
        return apiService.updateProfile(userId, updateProfile)
    }

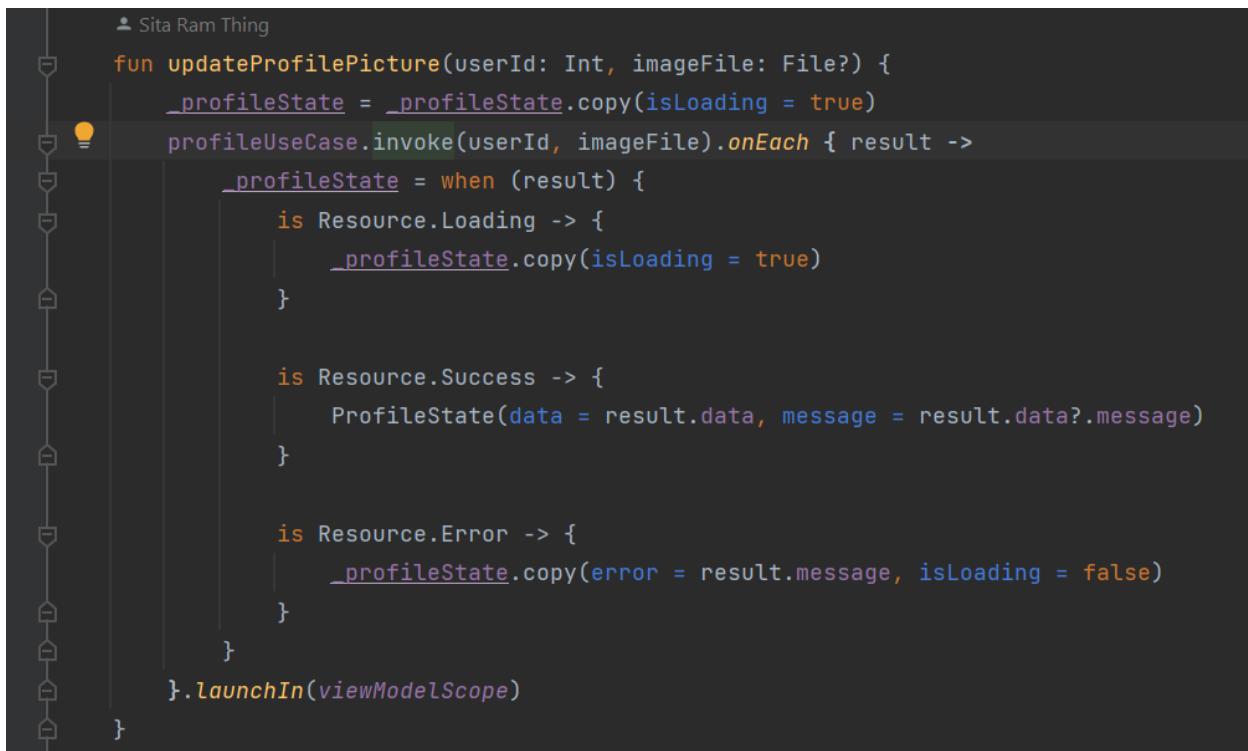
    override suspend fun updateProfilePicture(userId: Int?, imageFile: File?): ProfilePojo? {
        val requestBody = imageFile?.asRequestBody("image/*".toMediaTypeOrNull())
        val image = imageFile.let { MultipartBody.Part.createFormData("image", filename = imageFile?.name ?: "user", body = requestBody) }
        val userIdRequestBody = userId.toString().toRequestBody("text/plain".toMediaTypeOrNull())
        return apiService.updateProfilePicture(userIdRequestBody, image)
    }
}
```

Figure 252: Update profile repository implementation.



```
operator fun invoke(userId: Int?, imageFile: File?): Flow<Resource<ProfilePojo?>> = flow {
    emit(Resource.Loading())
    try {
        val result = profileRepository.updateProfilePicture(userId, imageFile)
        if (result?.isSuccess == true){
            emit(Resource.Success(data = result))
        } else {
            emit(Resource.Error(message = result?.message))
        }
    } catch (e: Exception) {
        emit(Resource.Error(message = "Unable to connect to the server."))
    }
}
```

Figure 253: Update profile use case implementation.



```
fun updateProfilePicture(userId: Int, imageFile: File?) {
    _profileState = _profileState.copy(isLoading = true)
    profileUseCase.invoke(userId, imageFile).onEach { result ->
        _profileState = when (result) {
            is Resource.Loading -> {
                _profileState.copy(isLoading = true)
            }

            is Resource.Success -> {
                ProfileState(data = result.data, message = result.data?.message)
            }

            is Resource.Error -> {
                _profileState.copy(error = result.message, isLoading = false)
            }
        }.launchIn(viewModelScope)
    }
}
```

Figure 254: Update user profile view model implementation.

➤ Implementations of Updated food details

```
# Update the donate food
class DonateFoodupdate(APIView):
    authentication_classes = [TokenAuthentication]
    permission_classes = [AllowAny]

    def patch(self, request):
        try:
            food_id = request.query_params.get('food_id')

            if not food_id:
                return Response({"error": "No food id provided", "is_success": False, "status": 400})

            food = Food.objects.filter(id=food_id).first()

            if not food:
                return Response({"error": "Food not found", "is_success": False, "status": 404})

            serializer = FoodSerializer(food, data=request.data, partial=True)

            if serializer.is_valid():
                food.modify_date = datetime.now().strftime('%Y-%m-%d')
                serializer.save()

            return Response({
                "message": "Updated Successful.",
                "is_success": True,
                "status": 200,
                "food": serializer.data
            })

        except Exception:
            return Response({"message": "Sorry, something went wrong on our end. Please try again later.", "is_success": False, "status": 500 })
```

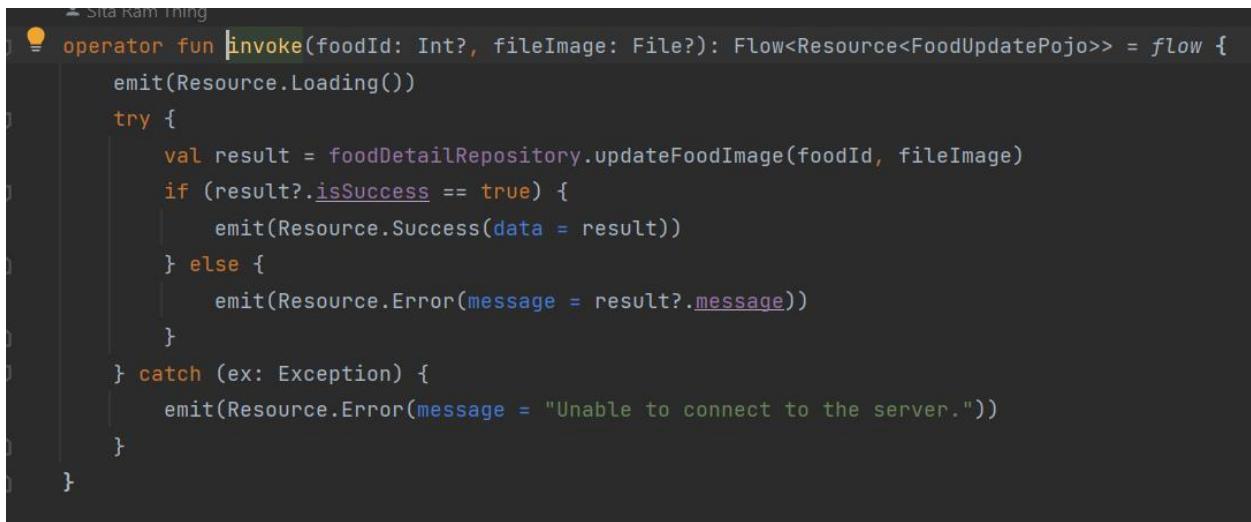
Figure 255: Food update Api function implementation.

```
class SitaRamThing {
    fun getUpdateDonateFoodImage(foodId: Int?, fileImage: File?) {
        _foodDetailState = _foodDetailState.copy(isLoading = true)
        foodDetailUseCase.invoke(foodId, fileImage).onEach { result ->
            _foodDetailState = when (result) {
                is Resource.Loading -> {
                    _foodDetailState.copy(isLoading = true)
                }

                is Resource.Success -> {
                    val foodsEntity = result.data?.food?.let {...}
                    _foodDetailState.copy(data = foodsEntity, isLoading = false, message = result.data?.message)
                }

                is Resource.Error -> {
                    _foodDetailState.copy(error = result.message, isLoading = false)
                }
            }
        }.launchIn(viewModelScope)
    }
}
```

Figure 256: Update food details function and implementation in the view model

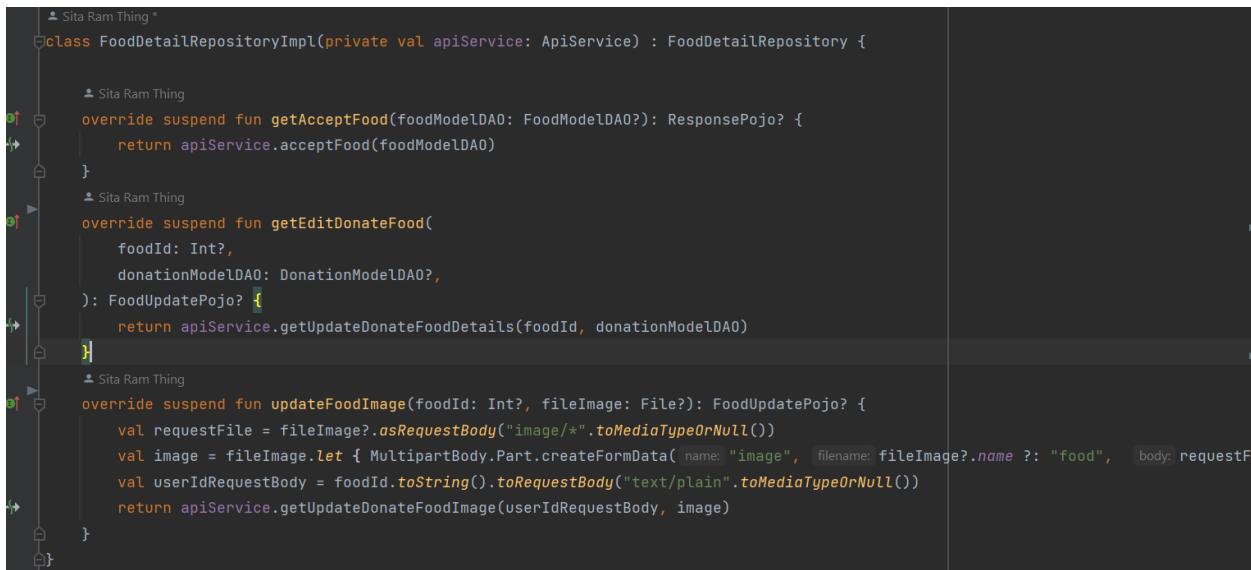


```

operator fun invoke(foodId: Int?, fileImage: File?): Flow<Resource<FoodUpdatePojo>> = flow {
    emit(Resource.Loading())
    try {
        val result = foodDetailRepository.updateFoodImage(foodId, fileImage)
        if (result?.isSuccess == true) {
            emit(Resource.Success(data = result))
        } else {
            emit(Resource.Error(message = result?.message))
        }
    } catch (ex: Exception) {
        emit(Resource.Error(message = "Unable to connect to the server."))
    }
}

```

Figure 257: Update food details implementation in the use case



```

class FoodDetailRepositoryImpl(private val apiService: ApiService) : FoodDetailRepository {

    override suspend fun getAcceptFood(foodModelDAO: FoodModelDAO?): ResponsePojo? {
        return apiService.acceptFood(foodModelDAO)
    }

    override suspend fun getEditDonateFood(
        foodId: Int?,
        donationModelDAO: DonationModelDAO?
    ): FoodUpdatePojo? {
        return apiService.getUpdateDonateFoodDetails(foodId, donationModelDAO)
    }

    override suspend fun updateFoodImage(foodId: Int?, fileImage: File?): FoodUpdatePojo? {
        val requestFile = fileImage?.asRequestBody("image/*".toMediaTypeOrNull())
        val image = fileImage?.let { MultipartBody.Part.createFormData("image", fileImage?.name ?: "food", requestFile) }
        val userIdRequestBody = foodId?.toString()?.toRequestBody("text/plain".toMediaTypeOrNull())
        return apiService.getUpdateDonateFoodImage(userIdRequestBody, image)
    }
}

```

Figure 258: Implementation of the repositoryImpl in updated food details.

```
└─ Sita Ram Thing
    @PATCH(ApiUrl.UPDATE_FOOD)
    suspend fun getUpdateDonateFoodDetails(
        @Query("food_id") foodId: Int?,
        @Body donationModelDAO: DonationModelDAO?,
    ): FoodUpdatePojo?
```

Figure 259: Update food details backend API call.

➤ Implementations of Delete food.

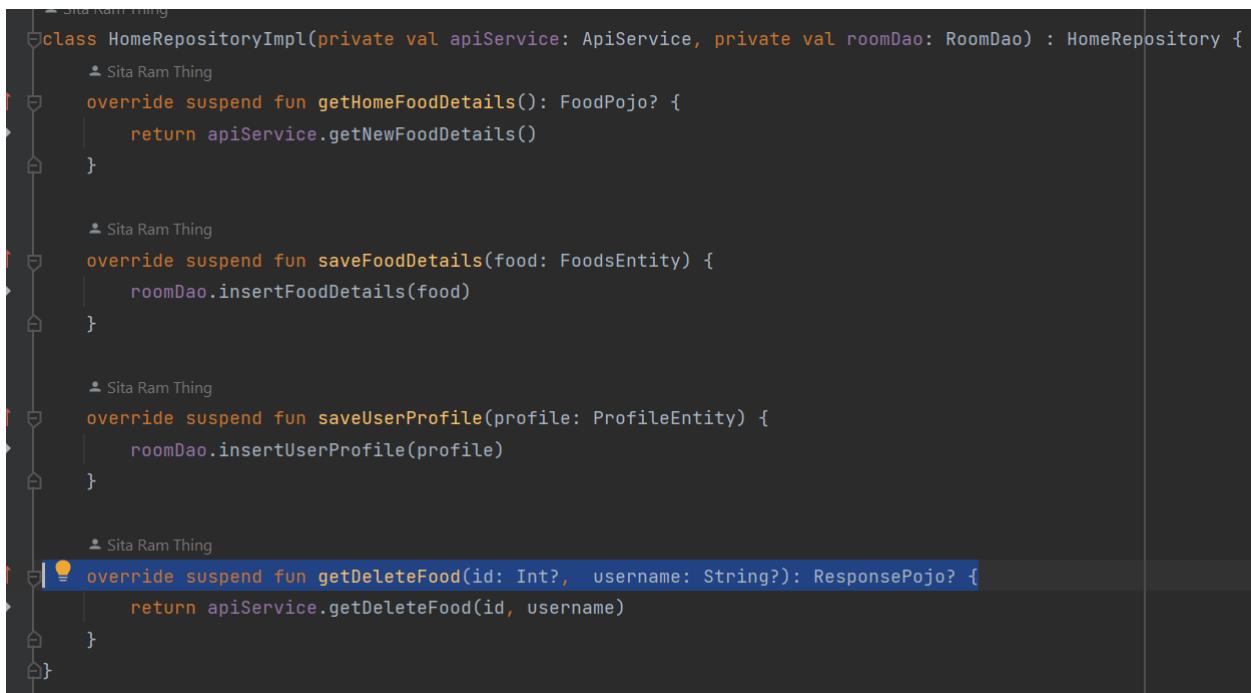
```
# [delete Food
class FoodDelete(APIView):
    authentication_classes = [TokenAuthentication]
    permission_classes = [AllowAny]

    def patch(self, request):
        try:
            food_id = request.query_params.get('food_id')
            username = request.query_params.get('username')
            # Assuming the food id is provided in the request data
            if food_id is not None:
                food = Food.objects.get(id=food_id)
                food.modify_by = username
                food.modify_date = datetime.now().strftime('%Y-%m-%d')
                food.is_delete = True
                food.save()
                return Response({"message": "Deleted successfully", "is_success": True, "status": 200})
            else:
                return Response({"message": "Food item not found", "is_success": False, "status": 404})
        except Exception as e:
            return Response({"message": "Sorry, something went wrong on our end. Please try again later.", 'is_success': False, 'status': 500})
```

Figure 260: Delete Api function implement.

```
// Delete food
└ Sita Ram Thing
  └ PATCH(ApiUrl.DELETE_FOOD)
    suspend fun getDeleteFood(
        @Query("food_id") id: Int?,
        @Query("username") username: String?,
    ): ResponsePojo?
```

Figure 261: Delete Api call function Mobile.



```

class HomeRepositoryImpl(private val apiService: ApiService, private val roomDao: RoomDao) : HomeRepository {
    override suspend fun getHomeFoodDetails(): FoodPojo? {
        return apiService.getNewFoodDetails()
    }

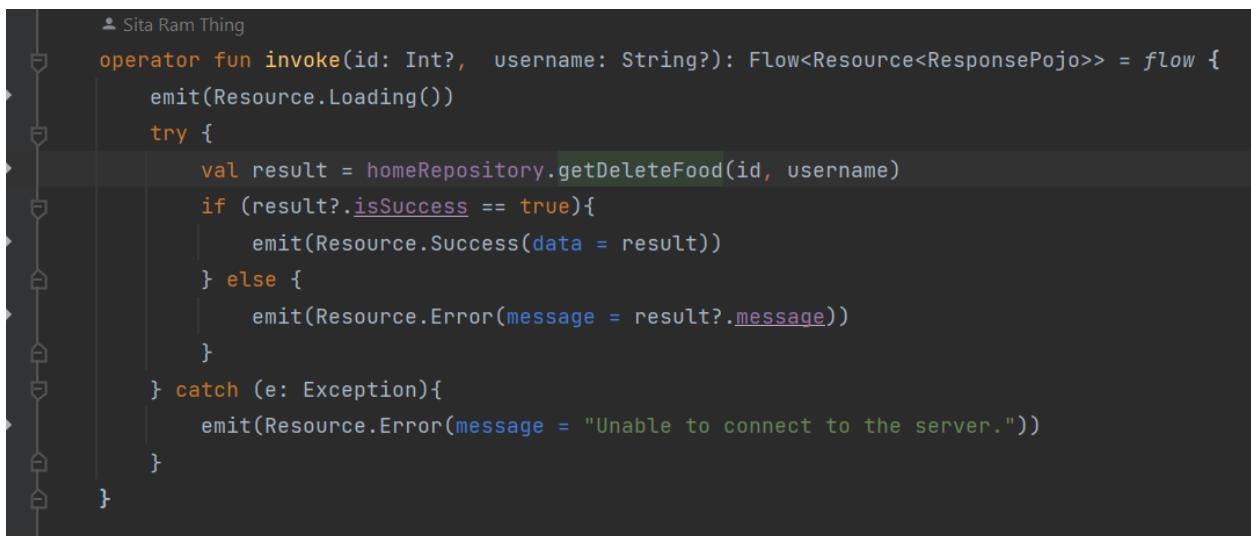
    override suspend fun saveFoodDetails(food: FoodsEntity) {
        roomDao.insertFoodDetails(food)
    }

    override suspend fun saveUserProfile(profile: ProfileEntity) {
        roomDao.insertUserProfile(profile)
    }

    override suspend fun getDeleteFood(id: Int?, username: String?): ResponsePojo? {
        return apiService.getDeleteFood(id, username)
    }
}

```

Figure 262: Repository for Delete function

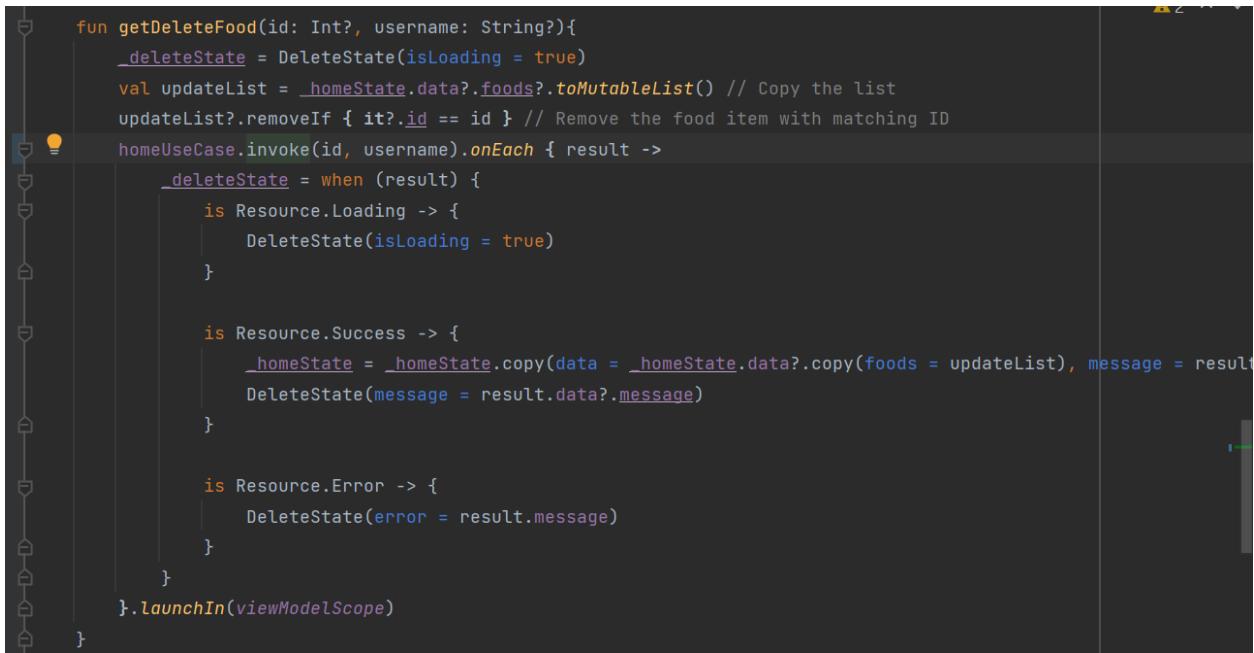


```

operator fun invoke(id: Int?, username: String?): Flow<Resource<ResponsePojo>> = flow {
    emit(Resource.Loading())
    try {
        val result = homeRepository.getDeleteFood(id, username)
        if (result?.isSuccess == true){
            emit(Resource.Success(data = result))
        } else {
            emit(Resource.Error(message = result?.message))
        }
    } catch (e: Exception){
        emit(Resource.Error(message = "Unable to connect to the server."))
    }
}

```

Figure 263: Food deletes use case implement.



```
fun getDeleteFood(id: Int?, username: String?) {
    _deleteState = DeleteState(isLoading = true)
    val updateList = _homeState.data?.foods?.toMutableList() // Copy the list
    updateList?.removeIf { it?.id == id } // Remove the food item with matching ID
    homeUseCase.invoke(id, username).onEach { result ->
        _deleteState = when (result) {
            is Resource.Loading -> {
                DeleteState(isLoading = true)
            }

            is Resource.Success -> {
                _homeState = _homeState.copy(data = _homeState.data?.copy(foods = updateList), message = result.message)
                DeleteState(message = result.data?.message)
            }

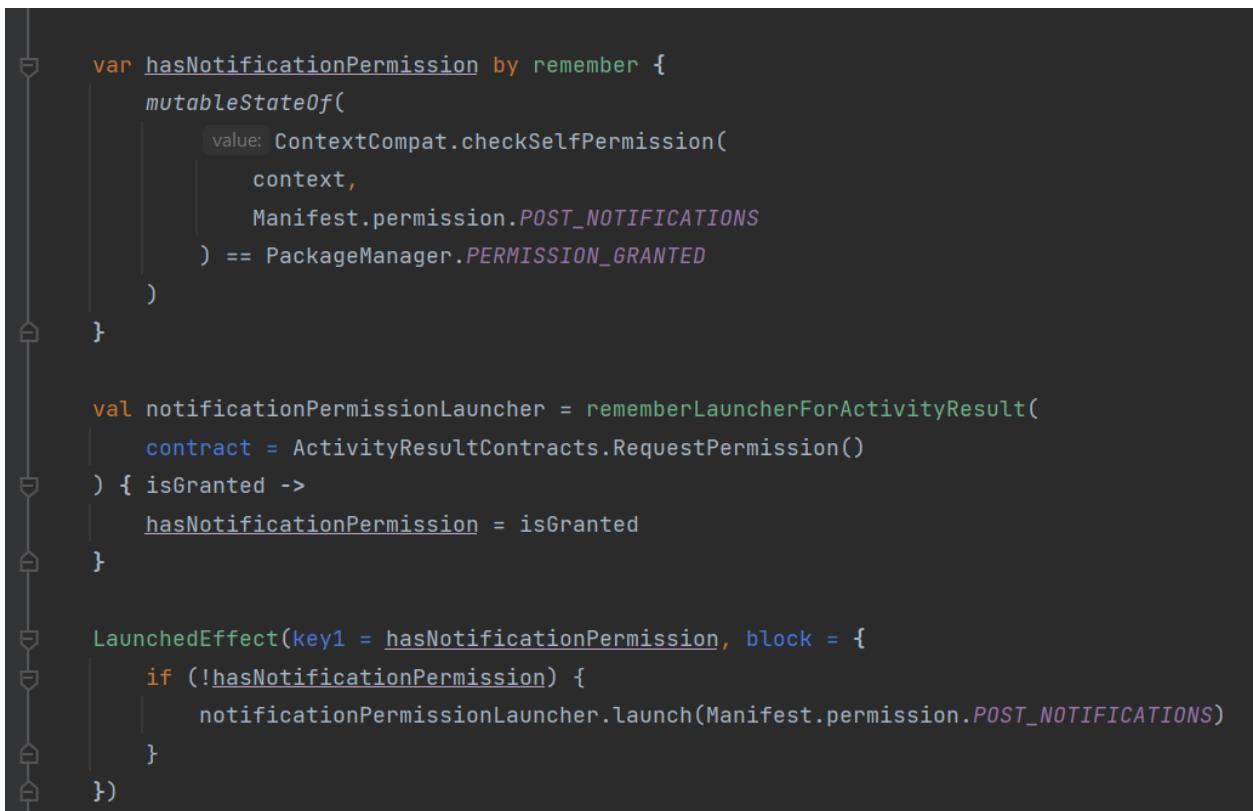
            is Resource.Error -> {
                DeleteState(error = result.message)
            }
        }
    }.launchIn(viewModelScope)
}
```

Figure 264: Food Delete View Model is implemented with state management.

➤ Implementations of Local notification

```
@HiltAndroidApp
class BaseApplication: Application(){
    @SitaRam Thing *
    @SuppressLint("ObsoleteSdkInt")
    override fun onCreate() {
        super.onCreate()
        // Check the SDK version and create the notification channel
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(
                CHANNEL_ID,
                CHANNEL_NAME,
                NotificationManager.IMPORTANCE_HIGH
            )
            val manager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            manager.createNotificationChannel(channel)
        }
    }
}
```

Figure 265: Create the notification channel

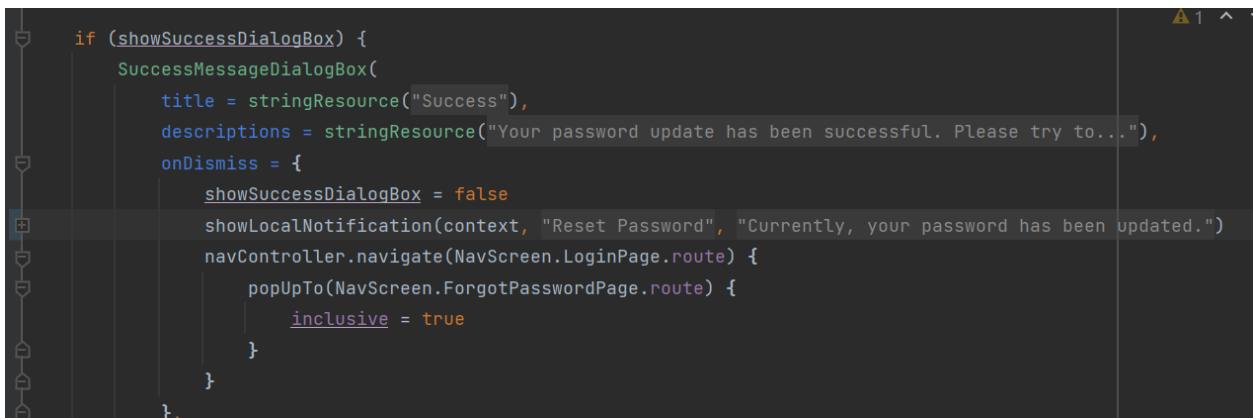


```
    var hasNotificationPermission by remember {
        mutableStateOf(
            value: ContextCompat.checkSelfPermission(
                context,
                Manifest.permission.POST_NOTIFICATIONS
            ) == PackageManager.PERMISSION_GRANTED
        )
    }

    val notificationPermissionLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.RequestPermission()
    ) { isGranted ->
        hasNotificationPermission = isGranted
    }

    LaunchedEffect(key1 = hasNotificationPermission, block = {
        if (!hasNotificationPermission) {
            notificationPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS)
        }
    })
}
```

Figure 266: Generating the notification permission and asking for permission.



```
if (showSuccessDialogBox) {
    SuccessMessageDialogBox(
        title = stringResource("Success"),
        descriptions = stringResource("Your password update has been successful. Please try to..."),
        onDismiss = {
            showSuccessDialogBox = false
            showLocalNotification(context, "Reset Password", "Currently, your password has been updated.")
            navController.navigate(NavScreen.LoginPage.route) {
                popUpTo(NavScreen.ForgotPasswordPage.route) {
                    inclusive = true
                }
            }
        },
    ),
}
```

Figure 267: publish the local notification.

[Go back to the same page](#)

7.4 Appendix D: Implementation

7.4.1 Unit Test Plans

Unit test of waste food management system list of features:

Test plans	Descriptions
All User:	
Login authentication	Enter the valid username and password to log in to the system and check whether the test is successful or not.
Register new user	Enrol the details of the newly registered user and register them in the system. Then, check the expected result.
Forgot password	After successfully verifying the user's email, they can enter a new password to update it. Then, check for a successful response.
Show local notification	After forgetting the password and the account was deactivated successfully the show the notification.
Auto redirect to the dashboard	After successful login without logging out, if the app is closed and reopened, it should reopen directly to the dashboard instead of the login screen.

The home screen only displays newly donated food	The home screen only displays newly donated food that is neither pending nor completed, along with the food donation history.
View donated food details	The user can click on "Donate Food" to view the donation details.
View self-profile	The user can view self-profile details on authenticating permission.
View other donor and volunteer profiles	The user can view any user profile details
View donation locations on the map	The user can view the food donation location with distance.
Update self-profile details	The user can update self-profile details
Receive push notifications	The user can receive push notifications after posting new food for donation.
View only self-history with ratings if the donation is completed	The user can view history only their history details.
Complain or report to the admin	If the donor fails to provide the food and the volunteer fails to pick up the food, they can be reported to the admin.
Logout system	The user can log out of the system without any exceptions, and the local storage will be cleared.

Update password	The user can update their account with a new password.
Account deactivation	The user can permanently deactivate their account if they no longer need it.
For Donors:	
Post or donate food	The donor can post surplus food for donation by providing the food details.
Update post	The donor can update the donated food details in case of any mistakes in the post.
Delete posts	The donor can delete the post if there is any mistake or if it is no longer needed in the history.
For Volunteers:	
Accept food donations	The volunteer can accept the food and the food can show the pending status.
Donation rating	The volunteer can distribute the completed then give the rating.
Delete history	The volunteer can delete food history without any exceptions.
For Admins:	
View system status in Pai Chat to data	The admin can view the system's current status in the pie chart in the home.

Activate new accounts	The admin can verify and activate new user accounts.
Searching the data	The admin can research anywhere found data.
Sorting data in ascending and descending	The admin view the data is sorted by ascending and descending.
Remove the data	The admin can remove the unnecessary data

Table 66: Unit tessting palnning tables

[Go back to the same page](#)

7.5 Appendix E: Testing and Analysis

7.5.1 System Testing

7.5.1.1 Update profile image

Test Case	Descriptions
Objectives	The user can update the user profile image if the user can needed.
Actions	The user can click the profile edit icon update the profile image and click the update confirm button.
Expected Result	The user profile details are successfully updated without any errors or exceptions.
Actual Result	The user profile image was successfully updated.
Test	The test was successful.

Table 67: Update profile image test in the mobile app

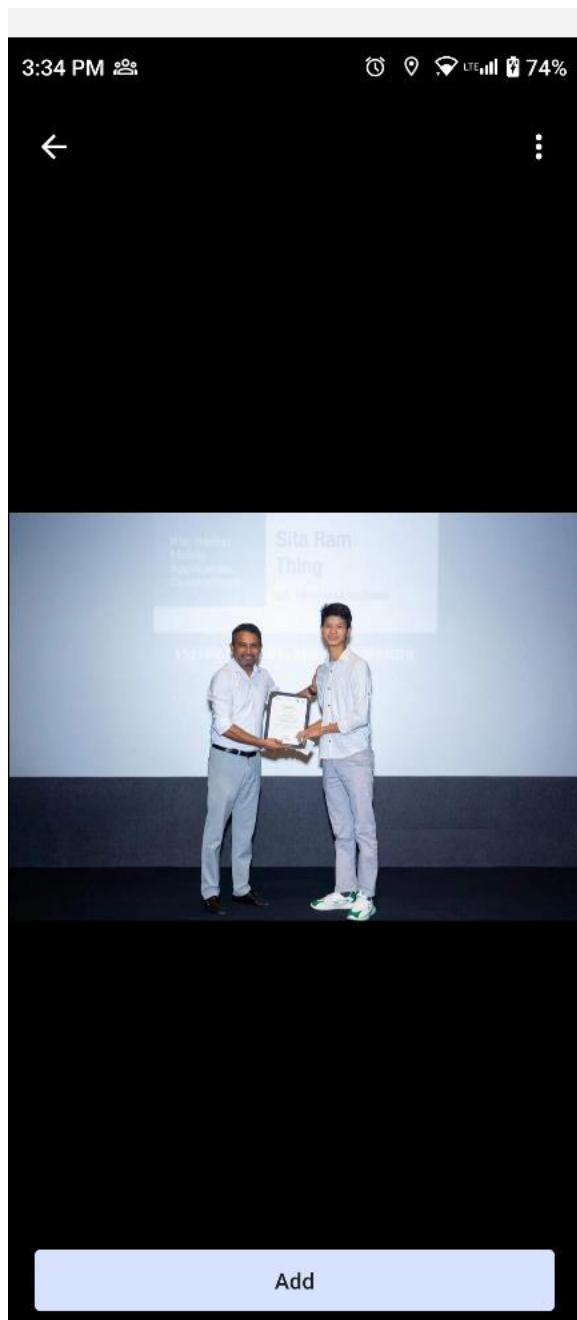


Figure 268: Select the profile image

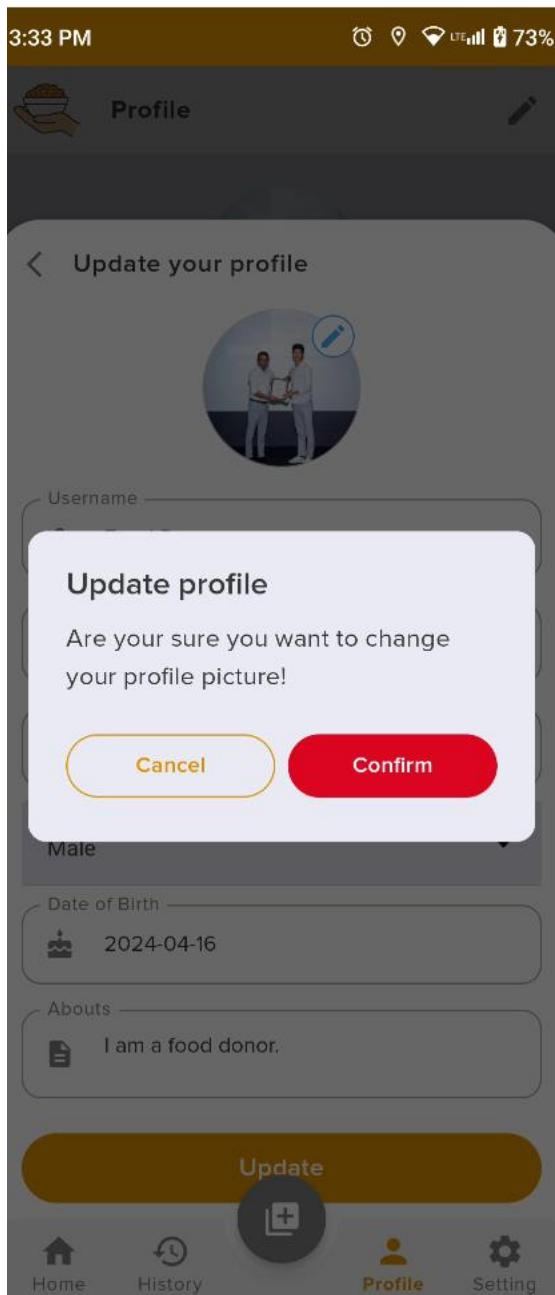


Figure 269: Display the confirmation message dialogue

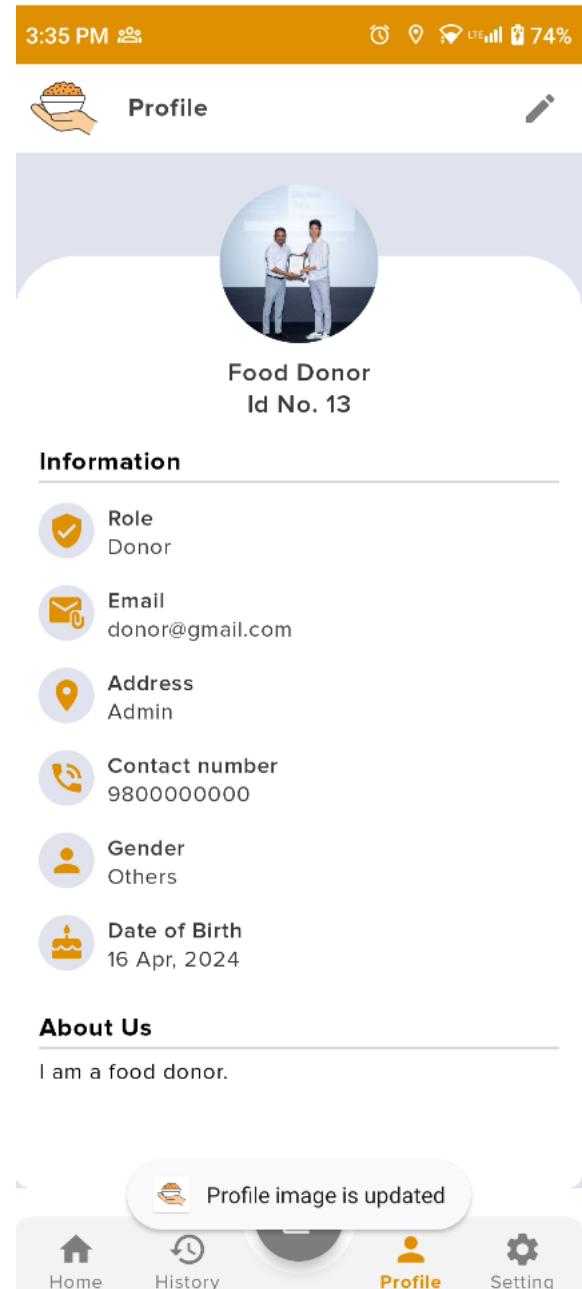


Figure 270: View the latest updated profile image

7.5.1.2 Deactivate user account (Delete account)

Test Case	Descriptions
Objectives	The user can activate the account to deactivate it if the user does not need the user account.
Actions	Click the deactivate button and show to confirmation dialog box the click the deactivate.
Expected Result	The user account is successfully permanently deactivated without logout the application and showing the location notification.
Actual Result	The user account was successfully deactivated.
Test	The test was successful.

Table 68: Deactivate user account test in mobile app

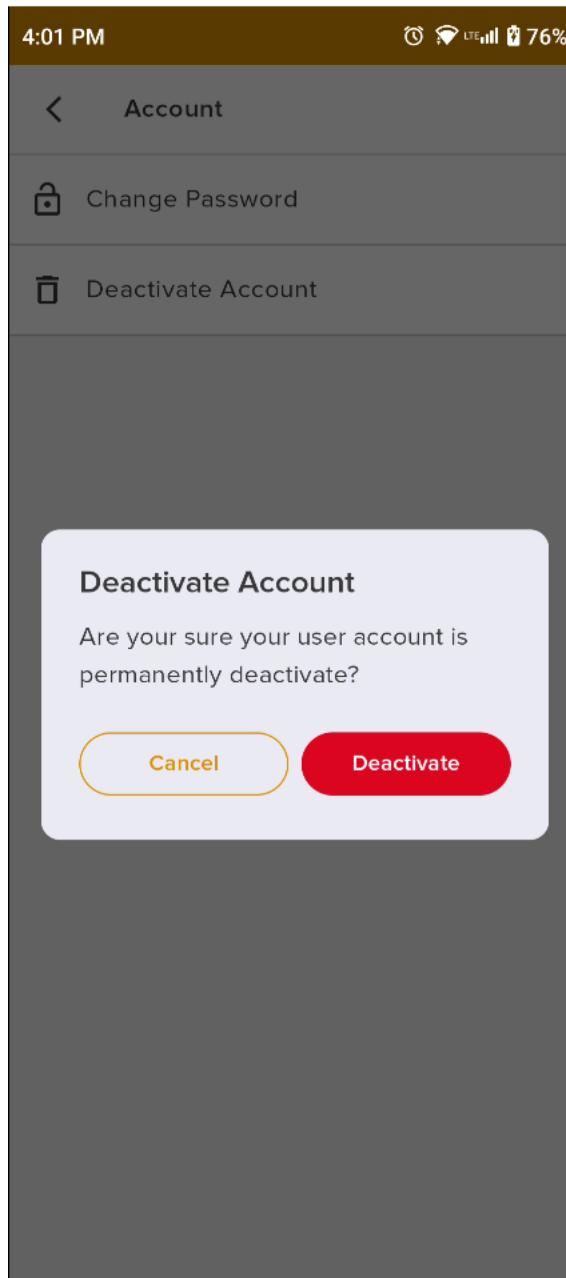


Figure 271: Display Account deactivate dialogue box

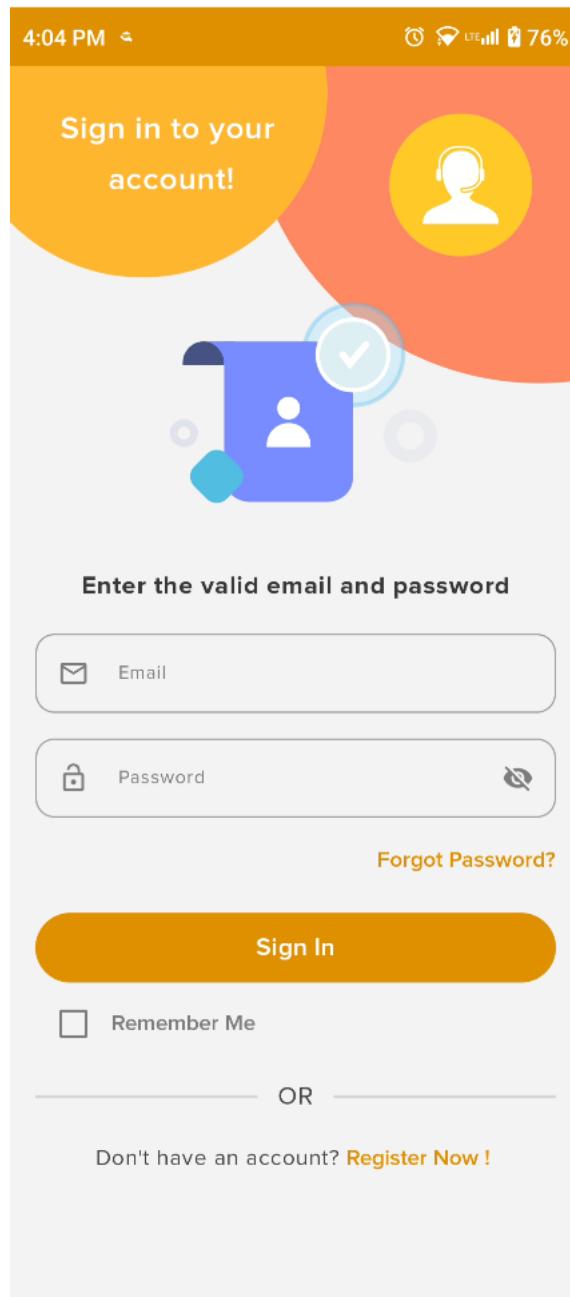


Figure 272: Navigate the login page

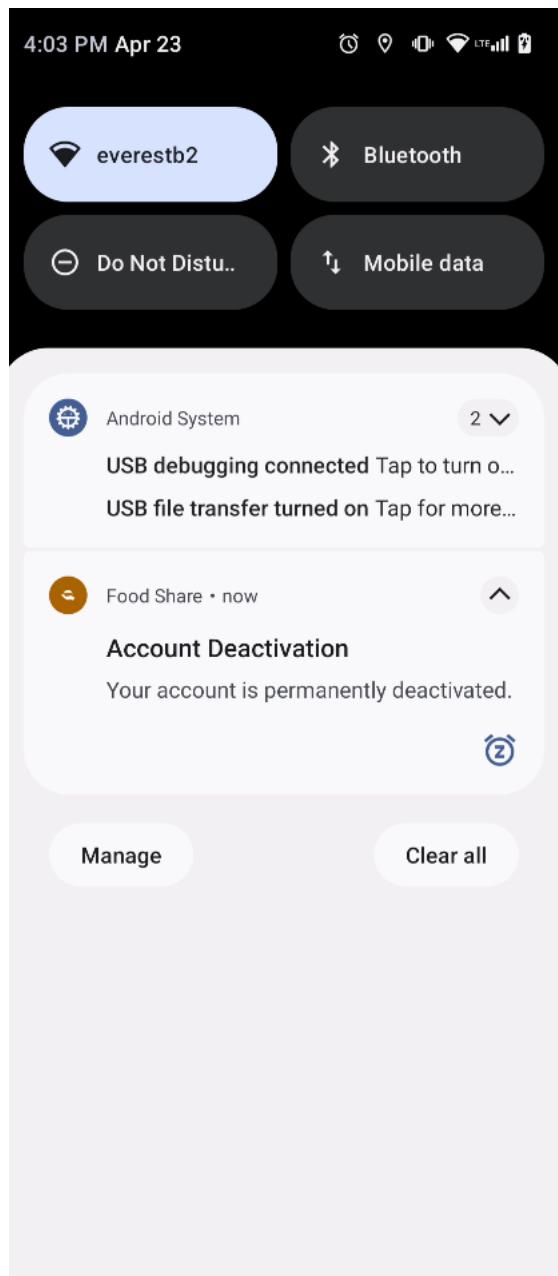


Figure 273: Account deactivate and out logout the system with get local notification

7.5.1.3 Change password

Test Case	Descriptions
Objectives	The user can change the new password for account security and remember the log in to the system.
Actions	Click the change password show the confirmation dialogue to fill in the password and click the update as
Expected Result	The user account should be successfully activated, allowing the user to log in to the system.
Actual Result	The user account was successfully activated.
Test	The test was successful.

Table 69: Change password test mobile app

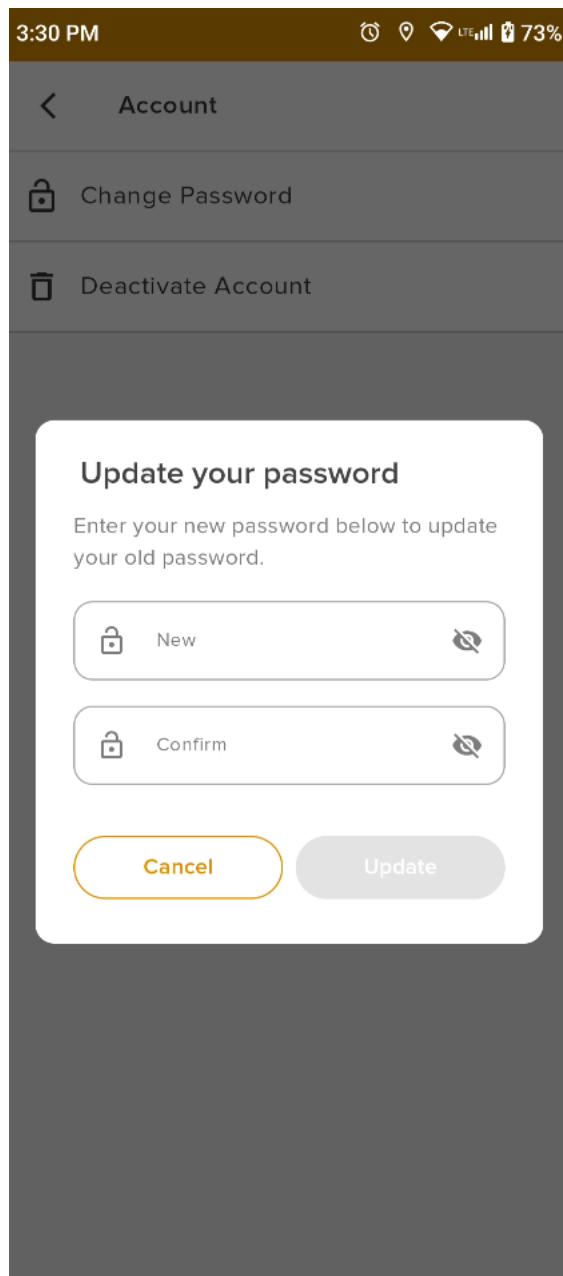


Figure 274: Change password input dialogue box

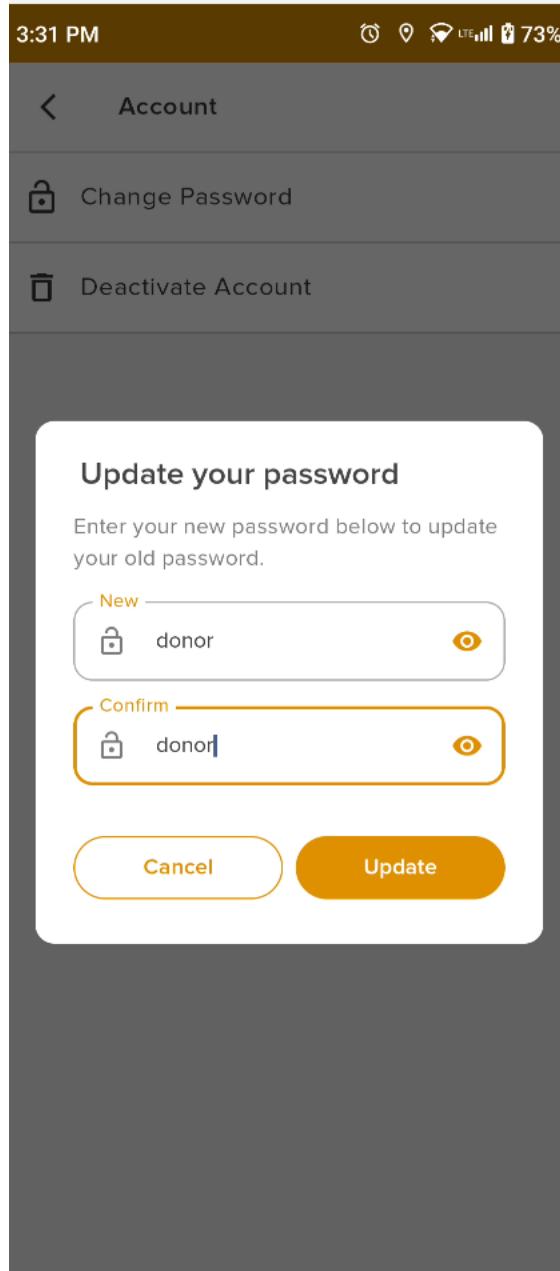


Figure 275: Enter the new and confirmation passwords in the dialogue box

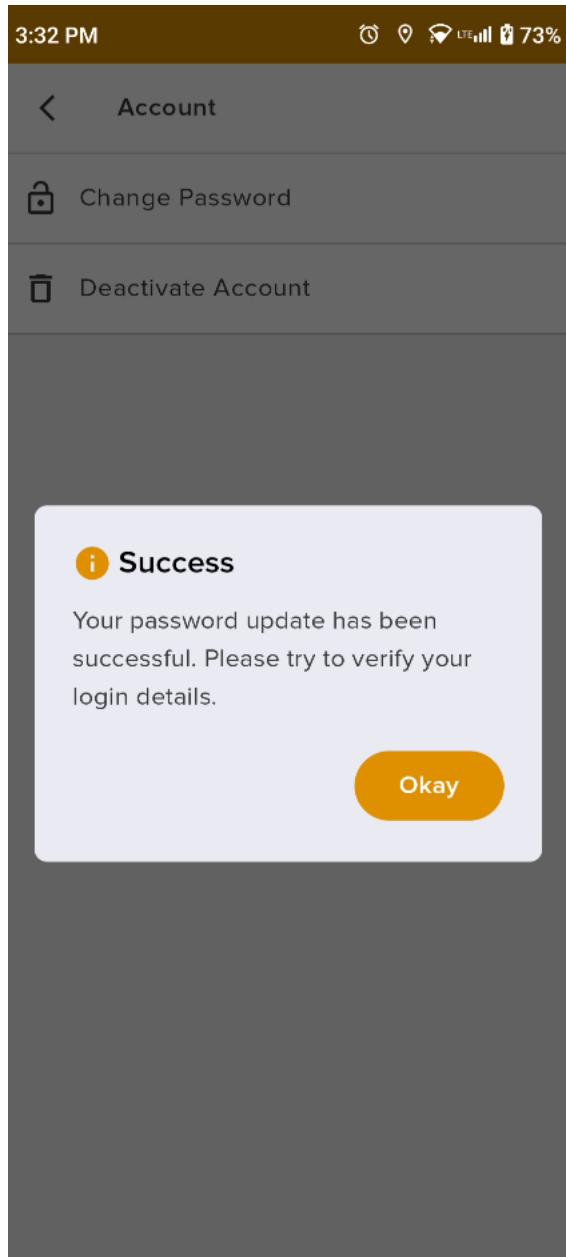


Figure 276: Display the success message

7.5.1.4 Logout App

Test No	1
Objectives	Go to the settings page and the user can log into the system in his account when showing the confirmation dialog box.
Actions	Click the logout button then display the confirmation button and click the logout button. Button: Cancel, Log out
Expected Result	The user can successfully log out of the system and clear the preference with local storage.
Actual Result	The user was successfully logged out.
Test	The test was successful.

Table 70: Logout test mobile app

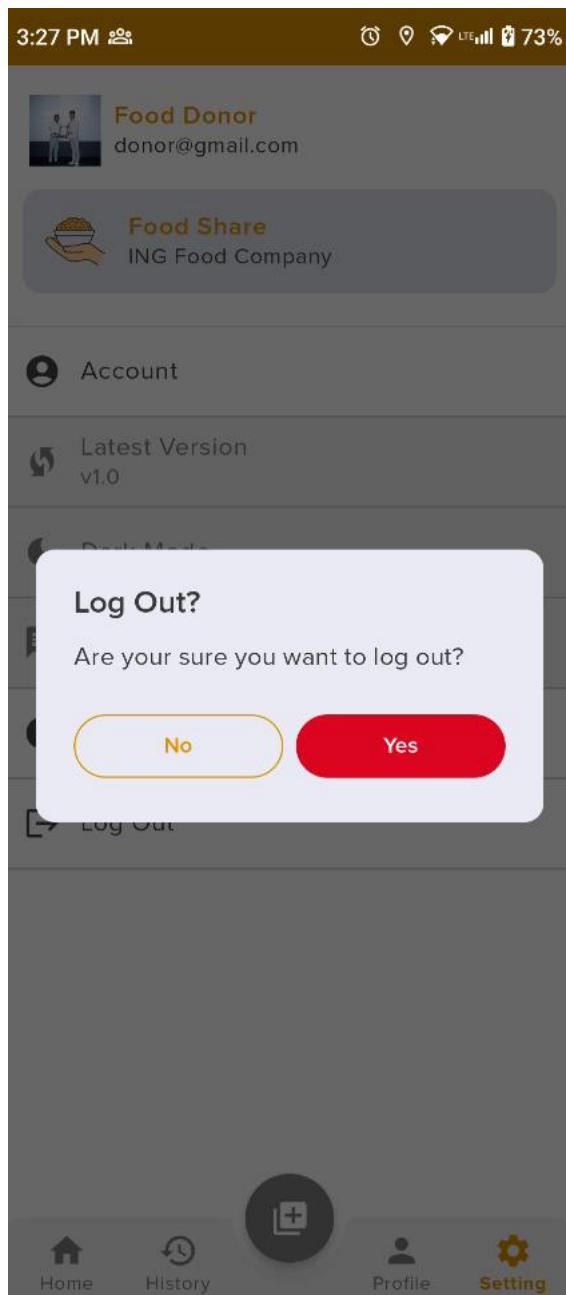


Figure 277: Logout confirmation dialogue box

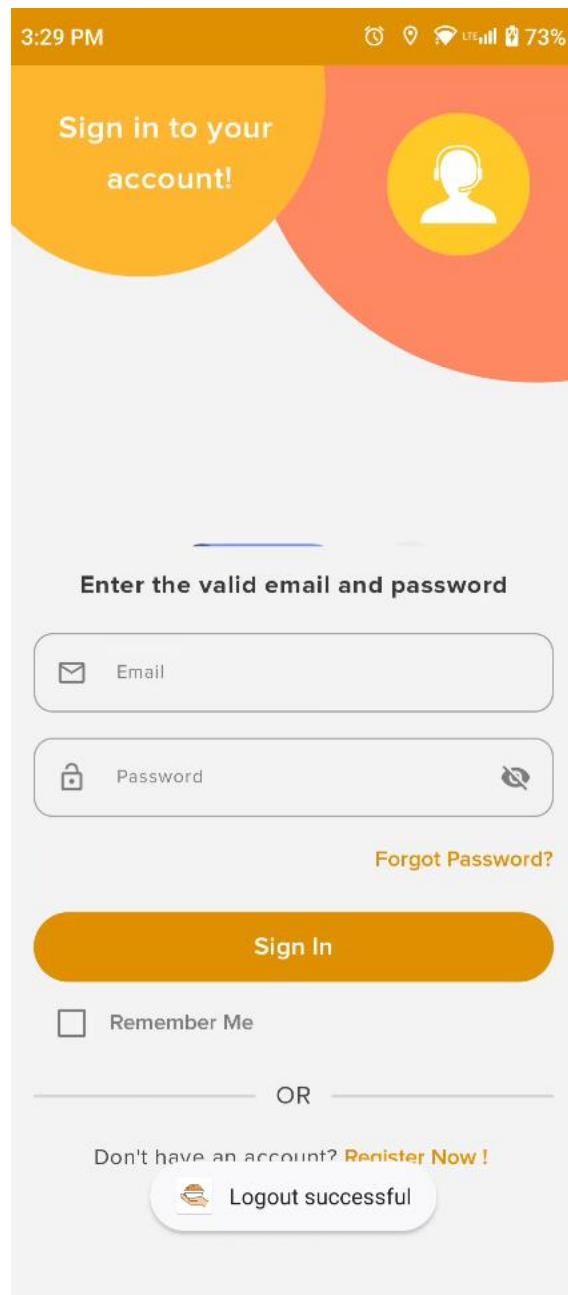


Figure 278: Logout success dialogue box

7.5.1.5 Internet connection test

Test No	1
Objectives	To check the internet connection where if the internet is not available then display the not internet screen.
Actions	Internet off to any activity in the application for API calls.
Expected Result	The user can off the internet and then show the not internet screen in the UI.
Actual Result	The UI is shown successfully.
Test	The test was successful.

Table 71: Logout test mobile app

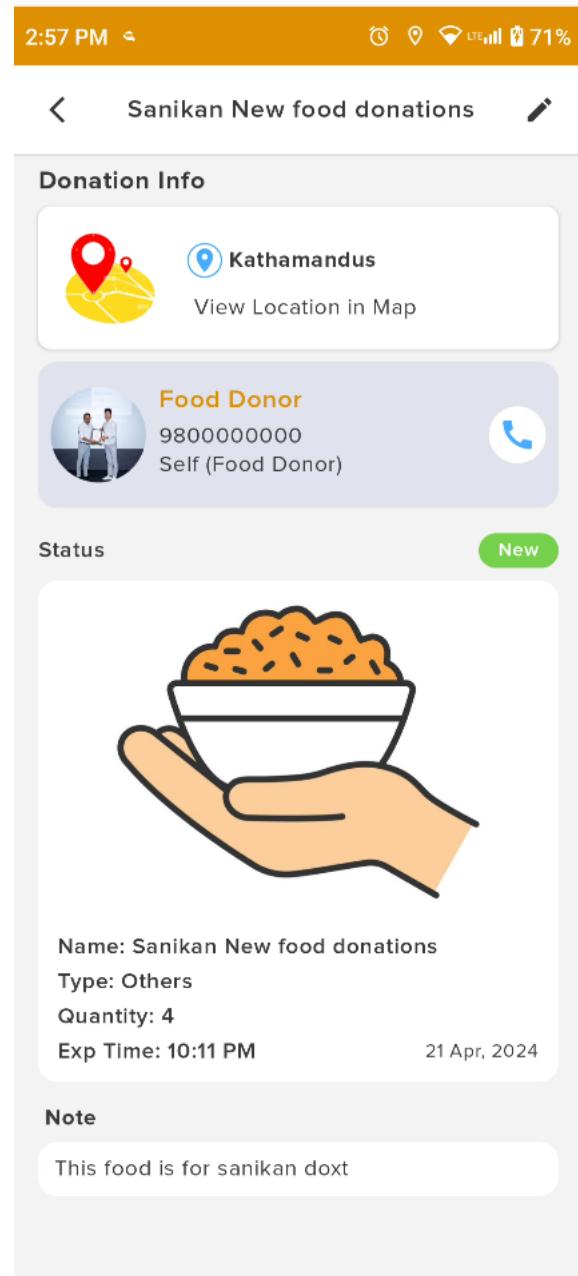


Figure 279: before the Internet disconnects the page

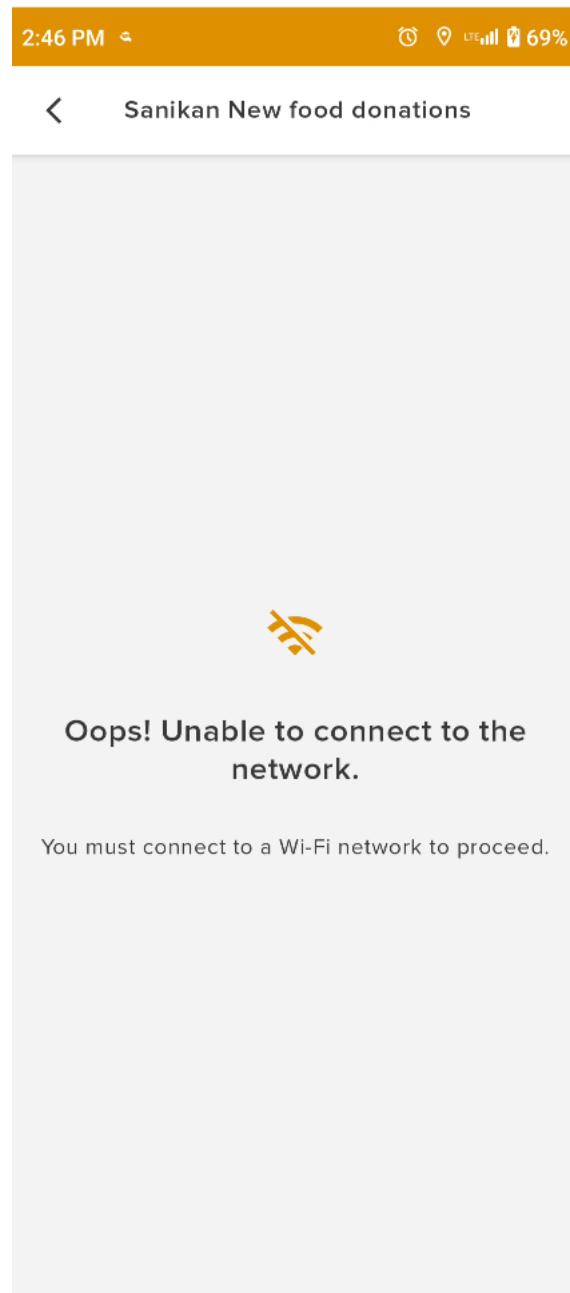


Figure 280: display the no internet connection fails the screen

7.5.1.6 Show the local notification

Test Case	Descriptions
Objectives	To forget the password and deactivate the user account.
Actions	Click the history button navigation bar and display the history list of history details.
Expected Result	To navigate the history page and show the history details without any error.
Actual Result	The histories are shown successfully.
Test	The test was successful.

Table 72: View the History test in the mobile app

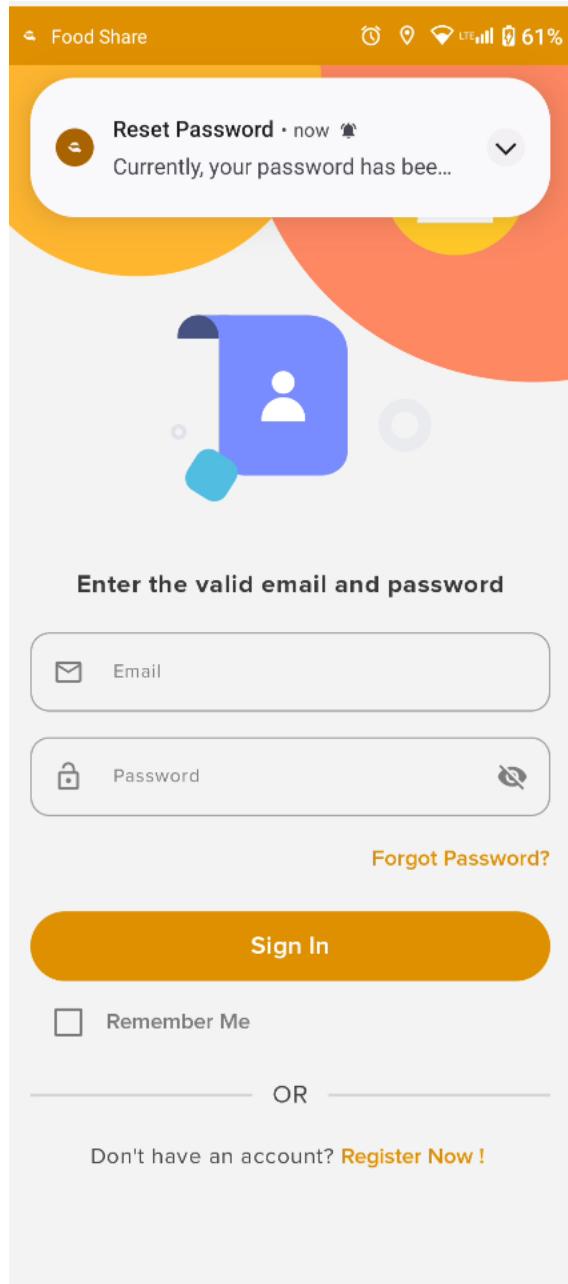


Figure 281: Display the local notification

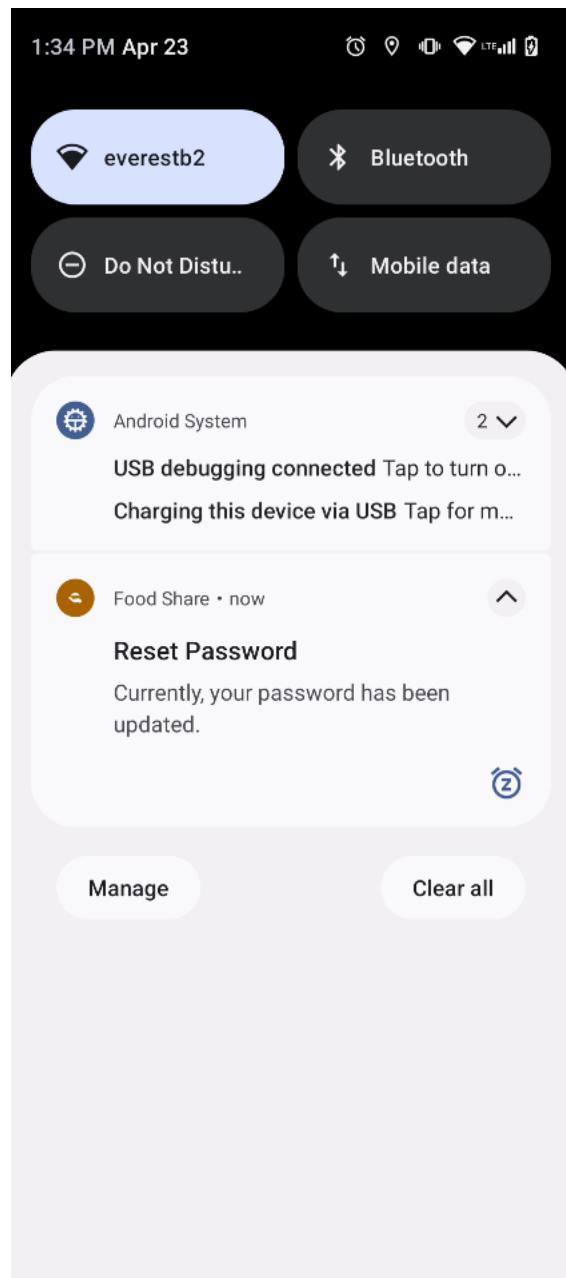


Figure 282: Show more details about the local notification

7.5.1.7 Google map permission

Test Case	Descriptions
Objectives	To display the welcome screen that asks for Google Maps permission after the user successfully installs the app and the intro slider screen is completed to open the welcome screen.
Actions	<p>The user installs the application, completes the installation process, and navigates from the intro slider to the welcome screen. Then, the application displays the Google Maps permission request, where the user can click to allow the permission.</p> <ul style="list-style-type: none"> - Clicks the WHILE USING THE APP button for Map. - Clicks the ALLOW button for permission.
Expected Result	The application can be successfully accessed for Google Maps permission, and successfully displays the map and receives notifications.
Actual Result	The user grants permission to display the Google Map and receives notifications.
Test	The test was successful.

Figure 283: System testing for Google Maps permission

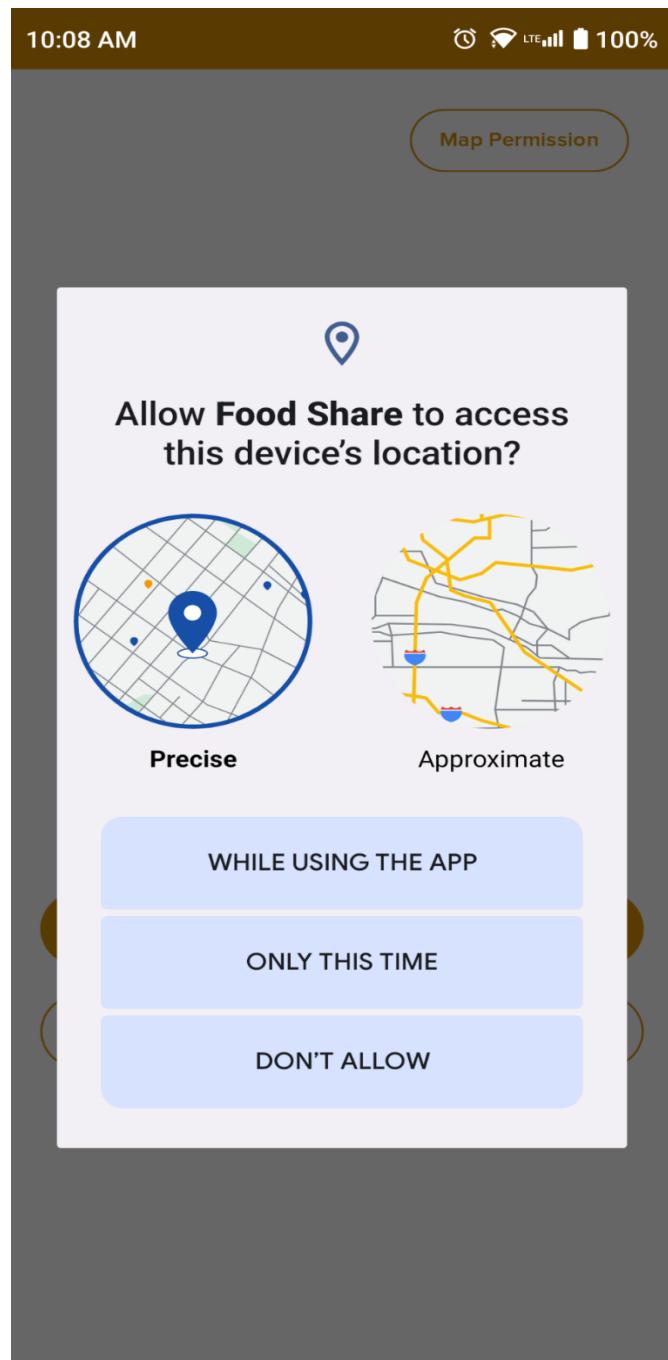


Figure 284: System test for Google Maps permission

[Go back to the same page](#)

7.5 Appendix F: Deployment (Not real deployment)

The system's fully developed planning phase to the development phase is completed. The first step was researching the food donation system and collecting information. Then, more details about the system were designed. After completing the system design and system architecture, they moved to the development phase. In the development phase, the backend was developed using Django Rest framework, the database was MySQL, the web development was done using HTML, CSS, and Javascript, and the mobile app was developed using Kotlin with Jetpack Compose, with local storage using Room database implemented. After development was completed, the system was released for end-user testing. The testing included functional, unit, and system testing to ensure the system was error-free. However, during the release of the mobile app, there was an error in uploading the APK file, which is available for free for 24 hours. Finally, the project was completed with documentation and submitted as the final year project.