**Eatonomy**



**Project ID: Fall-2023-51**

**Session: BSCS Fall 2020 to 2024**

**Submitted By**

|  |  |
| --- | --- |
| Haider Ali | 70069779 |
| Muhammad Ali | 70109014 |
|  |  |
|  |  |



Department of Computer Science & IT

The University of Lahore

Lahore, Pakistan

## *Declaration*

We have read the project guidelines and we understand the meaning of academic dishonesty, in particular plagiarism and collusion. We hereby declare that the work we submitted for our final year project, entitled **Eatonomy** is original work and has not been printed, published, or submitted before as a final year project, research work, publication, or any other documentation.

## 

**Group Member 1 Name: Mr. Haider Ali**

**SAP No: 70069779**

**Signature: …………………………**

**Group Member 2 Name: Mr. Muhammad Ali**

**SAP No: 70109014**

**Signature: …………………………**

## *Statement of Submission*

This is to certify that **Mr.** **Haider Ali** Roll No. **70069779 and Mr. Muhammad Ali** Roll No. **70109014** have successfully submitted the final project named **Eatonomy**, at the Computer Science & IT Department, The University of Lahore, Lahore, Pakistan, to fulfill the partial requirement of the degree of **BS in Computer Science**.

**Supervisor Name: Mr. Abdul Ghaffar**

**Signature: …………………………**

**Date: ………………………**

## *Dedication*

This project is dedicated to our fathers, who taught each of us that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to our mothers, who taught us that even the largest task can be accomplished if it is done one step at a time.

## *Acknowledgment*

## 

We truly acknowledge the cooperation and help made by Name of **Mr. Abdul Ghaffar**, **Professor** of the **University of Lahore**. He has been a constant source of guidance throughout the course of this project. We are also thankful to our friends and families whose silent support led us to complete our project.

Date:

June 03, 2024

## *Abstract*

This project focuses on the issues that are yet to be resolved by other food delivery apps, such as giving food recommendations when the user is unable to decide, be it alone or within a group. This app will also give users healthier options so that they can enjoy a balanced diet.

***Area of the Project***

Our project involves the development of mobile applications for both iOS and Android platforms using the Flutter framework, which is enriched with artificial intelligence capabilities to enhance the user experience and provide intelligent features.

***Technologies used***

* Kotlin [[16]](#sixteen)
* Dart [[17]](#seventeen)
* Flutter [[18]](#eighteen)
* Swift [8]
* Firebase [5]
* Google Maps [7]
* NoSQL [6]
* Flutter Flow [1]
* Figma [2]
* Xcode [9]
* Android Studio [11]

## List of Figures

[Figure 0 Example of User Interface 4](#_Toc171141640)

[Figure 1 Usecase Diagram - User 23](#_Toc171141641)

[Figure 2 Use Case: Sign Up 24](#_Toc171141643)

[Figure 3 Use Case: Login 25](#_Toc171141645)

[Figure 4 Use Case: Manage Account 26](#_Toc171141646)

[Figure 5 Use Case: Edit Address 27](#_Toc171141647)

[Figure 6 Use Case: Add Address 28](#_Toc171141648)

[Figure 7 Use Case: Remove Address 29](#_Toc171141649)

[Figure 8 Use Case: Change Password 30](#_Toc171141650)

[Figure 9 Use Case: Edit Phone Number 31](#_Toc171141651)

[Figure 10 Use Case: Add Phone Number 32](#_Toc171141652)

[Figure 11 Use Case: Remove Phone Number 33](#_Toc171141653)

[Figure 12 Use Case: Edit Picture 34](#_Toc171141654)

[Figure 13 Use Case: Change Name 35](#_Toc171141655)

[Figure 14 Use Case: View Restaurants 36](#_Toc171141656)

[Figure 15 Use Case: View Restaurant Details 37](#_Toc171141657)

[Figure 16 Use Case: View Restaurant Menu 38](#_Toc171141658)

[Figure 17 Use Case: View Food Recommendations 39](#_Toc171141659)

[Figure 18 Use Case: View Preferred Restaurants 40](#_Toc171141660)

[Figure 19 Use Case: View Restaurant Menu (Preferred) 41](#_Toc171141661)

[Figure 20 Use Case: View Cart 42](#_Toc171141662)

[Figure 21 Use Case: Remove Item from Cart 43](#_Toc171141663)

[Figure 22 Use Case: Place Order 44](#_Toc171141664)

[Figure 23 Use Case: Payment 45](#_Toc171141665)

[Figure 24 Use Case: Confirm Order 46](#_Toc171141666)

[Figure 25 Use Case: View Favorites 47](#_Toc171141667)

[Figure 26 Use Case: Order History 48](#_Toc171141668)

[Figure 27 Use Case: Payment Method - Cash on Delivery 49](#_Toc171141669)

[Figure 28 Architecture Diagram 50](#_Toc171141670)

[Figure 29 ERD 51](#_Toc171141671)

[Figure 30 Level 0 DFD 52](#_Toc171141672)

[Figure 31 Level 1 DFD 52](#_Toc171141673)

[Figure 32 Class Diagram 53](#_Toc171141674)

[Figure 33 Activity diagram Create Account 54](#_Toc171141675)

[Figure 34 Activity Diagram Login Account 55](#_Toc171141676)

[Figure 35 Activity Diagram Forgot password 56](#_Toc171141677)

[Figure 36 Activity Diagram Food Recommendation 57](#_Toc171141678)

[Figure 37 Activity Diagram Account Management 58](#_Toc171141679)

[Figure 38 Activity Diagram View Restaurant 59](#_Toc171141680)

[Figure 39 Activity Diagram View Cart 60](#_Toc171141681)

[Figure 40 Sequence Diagram Create Account 61](#_Toc171141682)

[Figure 41 Sequence Diagram Login Account 62](#_Toc171141683)

[Figure 42 Sequence Diagram Forgot Password 63](#_Toc171141684)

[Figure 43 Sequence Diagram Food Recommendation 64](#_Toc171141685)

[Figure 44 Sequence Diagram manage account 65](#_Toc171141686)

[Figure 45 Sequence Diagram View Restaurants 66](#_Toc171141687)

[Figure 46 Sequence Diagram View Cart 67](#_Toc171141688)

[Figure 47 Collaboration Diagram 68](#_Toc171141689)

[Figure 48 State Transition Diagram 69](#_Toc171141690)

[Figure 49 Component Diagram 70](#_Toc171141691)

[Figure 50 Deployment Diagram 71](#_Toc171141692)

[Figure 51: Login Screen 151](#_Toc171141693)

[Figure 52: Sign Up Screen 152](#_Toc171141694)

[Figure 53: Food Preferences Screen 153](#_Toc171141695)

[Figure 54: Home Screen 154](#_Toc171141696)

[Figure 55: Cart Screen 155](#_Toc171141697)

[Figure 56: Edit Profile Screen 156](#_Toc171141698)

[Figure 57: Favorite Dishes Screen 157](#_Toc171141699)

[Figure 58: Favorite Restaurants Screen 158](#_Toc171141700)

[Figure 59: Restaurant Menu Screen 159](#_Toc171141701)

[Figure 60: Orders Screen 160](#_Toc171141702)

[Figure 61: No Internet Screen 161](#_Toc171141703)

[Figure 62: Profile Settings Screen 162](#_Toc171141704)

## List of Tables

[Table 1: Functional Requirements – Register 6](#_Toc170626713)

[Table 2: Functional Requirements - Login 7](#_Toc170626714)

[Table 3: Functional Requirements – Forget Password 8](#_Toc170626715)

[Table 4: Functional Requirements – Reset Password 9](#_Toc170626716)

[Table 5: Functional Requirements – Manage Account 10](#_Toc170626717)

[Table 6: Functional Requirements – Food preferences 11](#_Toc170626718)

[Table 7: Functional Requirements – View Restaurants 12](#_Toc170626719)

[Table 8: Functional Requirements - Search 13](#_Toc170626720)

[Table 9: Functional Requirements – Restaurant Details 14](#_Toc170626721)

[Table 10: Functional Requirements – Food categories 15](#_Toc170626722)

[Table 11: Functional Requirements – Popular Items 16](#_Toc170626723)

[Table 12: Functional Requirements – View Cart 17](#_Toc170626724)

[Table 13: Functional Requirements – View Favorites 18](#_Toc170626725)

[Table 14: Functional Requirements – Order History 19](#_Toc170626726)

[Table 15: Functional Requirements – Payment 20](#_Toc170626727)

[Table 16: Use Case – Customer Sign Up 24](#_Toc170626736)

[Table 17: Use Case – Customer Login 25](#_Toc170626737)

[Table 18: Use Case – Customer Manage Account 26](#_Toc170626738)

[Table 19: Use Case – Customer Edit Address 27](#_Toc170626739)

[Table 20: Use Case – Customer Add Address 28](#_Toc170626740)

[Table 21: Use Case – Customer Remove Address 29](#_Toc170626741)

[Table 22: Use Case – Customer Change Password 30](#_Toc170626742)

[Table 23: Use Case – Customer Edit Phone Number 31](#_Toc170626743)

[Table 24: Use Case – Customer Add Phone Number 32](#_Toc170626744)

[Table 25: Use Case – Customer Remove Phone Number 33](#_Toc170626745)

[Table 26: Use Case – Customer Edit Picture 34](#_Toc170626746)

[Table 27: Use Case – Customer Change Name 35](#_Toc170626747)

[Table 28: Use Case – Customer View Restaurants 36](#_Toc170626748)

[Table 29: Use Case – Customer View Restaurant Details 37](#_Toc170626749)

[Table 30: Use Case – Customer View Restaurant Menu 38](#_Toc170626750)

[Table 31: Use Case – Customer View Food Recommendations 39](#_Toc170626751)

[Table 32: Use Case – Customer View Preferred Restaurants 40](#_Toc170626752)

[Table 33: Use Case – Customer View Preferred Restaurant Menu 41](#_Toc170626753)

[Table 34: Use Case – Customer View Cart 42](#_Toc170626754)

[Table 35: Use Case – Customer Remove Item from Cart 43](#_Toc170626755)

[Table 36: Use Case – Customer Place Order 44](#_Toc170626756)

[Table 37: Use Case – Customer Payment 45](#_Toc170626757)

[Table 38: Use Case – Customer Confirm Order 46](#_Toc170626758)

[Table 39: Use Case – Customer View Favorites 47](#_Toc170626759)

[Table 40: Use Case – Customer Order History 48](#_Toc170626760)

[Table 41: Use Case – Customer Payment Method 49](#_Toc170626761)

[Table 42: Positive Test Case – Address Widget 72](#_Toc170626778)

[Table 43: Negative Test Case – Address Widget 72](#_Toc170626779)

[Table 44: Positive Test Case – forgot password 73](#_Toc170626780)

[Table 45: Negative Test Case – forgot password 73](#_Toc170626781)

[Table 46: Positive Test Case – Login 74](#_Toc170626782)

[Table 47: Negative Test Case – Login 74](#_Toc170626783)

[Table 48: Positive Test Case – sign up 75](#_Toc170626784)

[Table 49: Negative Test Case – sign up 75](#_Toc170626785)

[Table 50: Positive Test Case – cart 76](#_Toc170626786)

[Table 51: Negative Test Case – cart 76](#_Toc170626787)

[Table 52: Positive Test Case – edit profile 77](#_Toc170626788)

[Table 53: Negative Test Case – edit profile 77](#_Toc170626789)

[Table 54: Positive Test Case – favorite restaurant 78](#_Toc170626790)

[Table 55: Negative Test Case – favorite restaurant 78](#_Toc170626791)

[Table 56: Positive Test Case – favorite item 79](#_Toc170626792)

[Table 57: Negative Test Case – favorite item 79](#_Toc170626793)

[Table 58: Positive Test Case – food categories 80](#_Toc170626794)

[Table 59: Negative Test Case – food categories 80](#_Toc170626795)

[Table 60: Positive Test Case – food preferences 81](#_Toc170626796)

[Table 61: Negative Test Case – food preferences 81](#_Toc170626797)

[Table 62: Positive Test Case – popular items 82](#_Toc170626798)

[Table 63: Negative Test Case – popular items 82](#_Toc170626799)

[Table 64: Positive Test Case – homescreen 83](#_Toc170626800)

[Table 65: Negative Test Case – homescreen 83](#_Toc170626801)

[Table 66: Positive Test Case – recommended restaurant 84](#_Toc170626802)

[Table 67: Negative Test Case – recommended restaurant 84](#_Toc170626803)

[Table 68: Positive Test Case – no internet 85](#_Toc170626804)

[Table 69: Negative Test Case – no internet 85](#_Toc170626805)

[Table 70: Positive Test Case – orders 86](#_Toc170626806)

[Table 71: Negative Test Case – orders 86](#_Toc170626807)

[Table 72: Positive Test Case – product details 87](#_Toc170626808)

[Table 73: Negative Test Case – product details 87](#_Toc170626809)

[Table 74: Positive Test Case – profile 88](#_Toc170626810)

[Table 75: Negative Test Case – profile 88](#_Toc170626811)

[Table 76: Positive Test Case – restaurant menu 89](#_Toc170626812)

[Table 77: Negative Test Case – restaurant menu 89](#_Toc170626813)

[Table 78: Positive Test Case – restaurants 90](#_Toc170626814)

[Table 79: Negative Test Case – restaurants 90](#_Toc170626815)

[Table 80: Positive Test Case – recommended restaurants 91](#_Toc170626816)

[Table 81: Negative Test Case – recommended restaurants 91](#_Toc170626817)

[Table 82: Valid Classes and Invalid Classes – Login username 92](#_Toc170626818)

[Table 83: Valid Classes and Invalid Classes – Login password 92](#_Toc170626819)

[Table 84: Valid Classes and Invalid Classes - Sign Up Email 93](#_Toc170626820)

[Table 85: Valid Classes and Invalid Classes - Sign Up password 93](#_Toc170626821)

[Table 86: Valid Classes and Invalid Classes - Sign Up confirm password 93](#_Toc170626822)

[Table 87: Valid Classes and Invalid Classes - Forgot Password 93](#_Toc170626823)

[Table 88: Valid Classes and Invalid Classes - Address 93](#_Toc170626824)

[Table 89: Valid Classes and Invalid Classes - Cart 94](#_Toc170626825)

[Table 90: Valid Classes and Invalid Classes - Profile 94](#_Toc170626826)

[Table 91: Valid Classes and Invalid Classes - Restaurant Menu 94](#_Toc170626827)

[Table 92: Valid Classes and Invalid Classes - Orders 94](#_Toc170626828)

[Table 93: Valid Classes and Invalid Classes - Food Preferences 94](#_Toc170626829)

[Table 94: Valid Classes and Invalid Classes - Favorite Restaurant 94](#_Toc170626830)

[Table 95: Valid Classes and Invalid Classes - Popular Items 94](#_Toc170626831)

[Table 96: Boundary Value Analysis – Login username 95](#_Toc170626832)

[Table 97: Boundary Value Analysis – Login password 95](#_Toc170626833)

[Table 98: Boundary Value Analysis - Sign Up Email 95](#_Toc170626834)

[Table 99: Boundary Value Analysis - Sign Up Password 95](#_Toc170626835)

[Table 100: Boundary Value Analysis - Sign Up Confirm Password 95](#_Toc170626836)

[Table 101: Boundary Value Analysis - Forgot Password 96](#_Toc170626837)

[Table 102: Boundary Value Analysis - Address 96](#_Toc170626838)

[Table 103: Boundary Value Analysis - Cart 96](#_Toc170626839)

[Table 104: Boundary Value Analysis – Profile Name 96](#_Toc170626840)

[Table 105: Boundary Value Analysis – Profile Email 96](#_Toc170626841)

[Table 106: Boundary Value Analysis - Restaurant Menu 96](#_Toc170626842)

[Table 107: Boundary Value Analysis - Orders 97](#_Toc170626843)

[Table 108: Boundary Value Analysis - Food Preferences 97](#_Toc170626844)

[Table 109: Boundary Value Analysis - Favorite Restaurant 97](#_Toc170626845)

[Table 110: Boundary Value Analysis - Popular Items 97](#_Toc170626846)

[Table 111: Decision Table - Login 98](#_Toc170626847)

[Table 112: Decision Table - Sign Up 98](#_Toc170626848)

[Table 113: Decision Table - Forgot Password 98](#_Toc170626849)

[Table 114: Decision Table - Address 99](#_Toc170626850)

[Table 115: Decision Table - Cart 99](#_Toc170626851)

[Table 116: Decision Table - Profile 99](#_Toc170626852)

[Table 117: Decision Table - Restaurant Menu 99](#_Toc170626853)

[Table 118: Decision Table - Orders 100](#_Toc170626854)

[Table 119: Decision Table - Food Preferences 100](#_Toc170626855)

[Table 120: Decision Table - Favorite Restaurant 100](#_Toc170626856)

[Table 121: Decision Table - Popular Items 100](#_Toc170626857)

[Table 122: State Transition Table - Login 101](#_Toc170626858)

[Table 123: State Transition Table - Sign Up 101](#_Toc170626859)

[Table 124: State Transition Table - Forgot Password 101](#_Toc170626860)

[Table 125: State Transition Table - Address 102](#_Toc170626861)

[Table 126: State Transition Table - Cart 102](#_Toc170626862)

[Table 127: State Transition Table - Profile 102](#_Toc170626863)

[Table 128: State Transition Table - Restaurant Menu 103](#_Toc170626864)

[Table 129: State Transition Table - Orders 103](#_Toc170626865)

[Table 130: State Transition Table - Food Preferences 103](#_Toc170626866)

[Table 131: State Transition Table - Favorite Restaurant 104](#_Toc170626867)

[Table 132: State Transition Table - Popular Items 104](#_Toc170626868)

[Table 133: Use Case Testing - Login 105](#_Toc170626869)

[Table 134: Use Case Testing - Sign Up 106](#_Toc170626870)

[Table 135: Use Case Testing - Forgot Password 107](#_Toc170626871)

[Table 136: Use Case Testing - Manage Address 108](#_Toc170626872)

[Table 137: Use Case Testing - Manage Cart 109](#_Toc170626873)

[Table 138: Use Case Testing - Manage Profile 110](#_Toc170626874)

[Table 139: Use Case Testing - View Restaurant Menu 110](#_Toc170626875)

[Table 140: Use Case Testing - Manage Orders 111](#_Toc170626876)

[Table 141: Use Case Testing - Manage Food Preferences 112](#_Toc170626877)

[Table 142: Use Case Testing - Manage Favorite Restaurant 113](#_Toc170626878)

[Table 143: Use Case Testing - View Popular Items 114](#_Toc170626879)

[Table 144: Cyclomatic Complexity - Login Function 115](#_Toc170626880)

[Table 145: Cyclomatic Complexity - Sign Up Function 115](#_Toc170626881)

[Table 146: Cyclomatic Complexity - Cart Management Function 115](#_Toc170626882)

[Table 147: Cyclomatic Complexity - Profile Update Function 115](#_Toc170626883)

[Table 148: Cyclomatic Complexity - Viewing Popular Items Function 115](#_Toc170626884)

[Table 149: Performance Test Case – Login 116](#_Toc170626885)

[Table 150: Performance Test Case – Sign Up 117](#_Toc170626886)

[Table 151: Performance Test Case – Home Screen 118](#_Toc170626887)

[Table 152: Performance Test Case – Adding Item to Cart 119](#_Toc170626888)

[Table 153: Performance Test Case – Viewing Cart 120](#_Toc170626889)

[Table 154: Performance Test Case – Profile Update 120](#_Toc170626890)

[Table 155: Performance Test Case – Viewing Orders 121](#_Toc170626891)

[Table 156: Performance Test Case – Viewing Restaurant Menu 121](#_Toc170626892)

[Table 157: Performance Test Case – Viewing Popular Items 122](#_Toc170626893)

[Table 158: Stress Test Case – Login Under Heavy Load 123](#_Toc170626894)

[Table 159: Stress Test Case – Sign Up Under Heavy Load 124](#_Toc170626895)

[Table 160: Stress Test Case – Viewing Home Screen Under Heavy Load 125](#_Toc170626896)

[Table 161: Stress Test Case – Adding Items to Cart Under Heavy Load 126](#_Toc170626897)

[Table 162: Stress Test Case – Viewing Cart Under Heavy Load 127](#_Toc170626898)

[Table 163: Stress Test Case – Profile Update Under Heavy Load 128](#_Toc170626899)

[Table 164: Stress Test Case – Viewing Orders Under Heavy Load 129](#_Toc170626900)

[Table 165: Stress Test Case – Viewing Restaurant Menu Under Heavy Load 130](#_Toc170626901)

[Table 166: Stress Test Case – Viewing Popular Items Under Heavy Load 131](#_Toc170626902)

[Table 167: System Test Case – Login and Navigation 132](#_Toc170626903)

[Table 168: System Test Case – Sign Up and Profile Update 133](#_Toc170626904)

[Table 169: System Test Case – Adding Items to Cart and Checkout 134](#_Toc170626905)

[Table 170: System Test Case – Viewing and Removing Items from Cart 135](#_Toc170626906)

[Table 171: System Test Case – Viewing and Adding Favorite Items 136](#_Toc170626907)

[Table 172: System Test Case – Viewing Order History 137](#_Toc170626908)

[Table 173: System Test Case – Viewing Restaurant Menu 138](#_Toc170626909)

[Table 174: System Test Case – Viewing Popular Items 139](#_Toc170626910)

[Table 175: System Test Case – Handling No Internet Connection 140](#_Toc170626911)

[Table 176: Final Regression Test Case – Login 142](#_Toc170626912)

[Table 177: Normal Regression Test Case – Login 142](#_Toc170626913)

[Table 178: Final Regression Test Case – Sign Up 143](#_Toc170626914)

[Table 179: Normal Regression Test Case – Sign Up 143](#_Toc170626915)

[Table 180: Final Regression Test Case – Adding Items to Cart 144](#_Toc170626916)

[Table 181: Normal Regression Test Case – Adding Items to Cart 144](#_Toc170626917)

[Table 182: Final Regression Test Case – Viewing Orders 145](#_Toc170626918)

[Table 183: Normal Regression Test Case – Viewing Orders 145](#_Toc170626919)

[Table 184: Final Regression Test Case – Viewing Restaurant Menu 146](#_Toc170626920)

[Table 185: Normal Regression Test Case – Viewing Restaurant Menu 146](#_Toc170626921)

**Table of Content**

[*Declaration* i](#_Toc170627250)

[*Statement of Submission* ii](#_Toc170627251)

[*Dedication* iii](#_Toc170627252)

[*Acknowledgment* iv](#_Toc170627253)

[*Abstract* v](#_Toc170627254)

[List of Figures vii](#_Toc170627255)

[List of Tables x](#_Toc170627256)

[Chapter 1: Introduction to the Problem 1](#_Toc170627257)

[1.1 Introduction 1](#_Toc170627258)

[1.2 Purpose 1](#_Toc170627259)

[1.3 Objective 1](#_Toc170627260)

[1.4 Existing Solution 1](#_Toc170627261)

[1.5 Proposed Solution 1](#_Toc170627262)

[Chapter 2: Software Requirement Specification 2](#_Toc170627263)

[2.1 Introduction 2](#_Toc170627264)

[2.1.1 Purpose 2](#_Toc170627265)

[2.1.2 Scope 2](#_Toc170627266)

[2.1.3 Definitions, acronyms, and abbreviations 2](#_Toc170627267)

[2.2 Overall description 3](#_Toc170627268)

[2.2.1 Product perspective 3](#_Toc170627269)

[2.2.2 Product functions 5](#_Toc170627270)

[2.2.3 User characteristics 21](#_Toc170627286)

[2.2.4 Constraints 21](#_Toc170627287)

[2.2.5 Assumptions and dependencies 21](#_Toc170627288)

[2.2.6 Apportioning of Requirements 21](#_Toc170627289)

[2.3 Specific requirements 21](#_Toc170627290)

[2.3.1 Functional Requirement 21](#_Toc170627291)

[2.3.2 Non-functional Requirements 22](#_Toc170627292)

[Chapter 3: Use Case Analysis 23](#_Toc170627293)

[Chapter 4: Design 50](#_Toc170627320)

[4.1 Architecture Diagram 50](#_Toc170627321)

[4.2 ERD with data dictionary 51](#_Toc170627322)

[4.3 Data Flow diagram 52](#_Toc170627323)

[4.3.1 The level 0 52](#_Toc170627324)

[4.3.1 The level 1 52](#_Toc170627325)

[4.4 Class Diagram 53](#_Toc170627326)

[4.5 Activity Diagram 54](#_Toc170627327)

[4.6 Sequence Diagram 61](#_Toc170627335)

[4.7 Collaboration Diagram 68](#_Toc170627343)

[4.8 State Transition Diagram 69](#_Toc170627344)

[4.9 Component Diagram 70](#_Toc170627345)

[4.10 Deployment Diagram 71](#_Toc170627346)

[Chapter 5: Testing 72](#_Toc170627347)

[5.1 Test Case Specifications 72](#_Toc170627348)

[5.2 Black Box Test Cases 92](#_Toc170627409)

[5.3 White Box Test Cases 115](#_Toc170627472)

[5.4 Performance testing 116](#_Toc170627478)

[5.5 Stress Testing 123](#_Toc170627488)

[5.6 System Testing 132](#_Toc170627498)

[5.7 Regression Testing 141](#_Toc170627508)

[Chapter 6: Tools and Techniques 147](#_Toc170627519)

[6.1 Languages Used in Development 147](#_Toc170627520)

[6.2 Applications and Tools 147](#_Toc170627521)

[6.3 Libraries and Extensions 148](#_Toc170627522)

[Chapter 7: Summary and Conclusion 149](#_Toc170627523)

[Chapter 8: User Manual 151](#_Toc170627524)

[Chapter 9: Lessons Learnt and Future Work 163](#_Toc170627548)

[References 166](#_Toc170627551)

[Appendix 167](#_Toc170627552)

### 

### Chapter 1: Introduction to the Problem

##### Introduction

Whenever we go out to eat, we normally take a lot of time to decide what we want to eat and from what restaurant we want to eat, and we are also restricted by our budget as well. However, there is no app that can provide personalized and optimal food suggestions based on these factors and the user’s order history. We end up wasting a lot of time in thinking about what to order and may even end up choosing something that is not near us or even miss out on many options that do not come into our minds.

##### Purpose

Many factors affect our ability to select what food we want to eat. Our mood, different varieties of cuisine, and the budget that we can spend all affect our decisions about what we want to eat. This app helps us make our lives easier and saves us a lot of time in deciding what to eat by offering us the best suggestions based on our previous order history as well as asking for what we might be interested in eating and giving us a personalized menu even if we have a group of more than one person.

##### Objective

Our objective for this app is to develop a user-friendly interface that allows users to input their dietary preferences and location. Our app will use an algorithm that analyzes user data and recommends appropriate food choices from local restaurants. The app will have a comprehensive restaurant database to provide a wide range of food options to users. Data privacy and security will be ensured by adhering to relevant regulations and implementing robust security measures. We want to achieve a high level of user satisfaction and engagement with the app, leading to increased usage and user retention.

##### Existing Solution

None of our competitors have a viable solution that aids the user while ordering food. They just have a basic recommendations section that is mostly based on what other people might be ordering. Or they just display the sponsored menu items. These may not even be relevant to each individual user or what the user might want.

##### Proposed Solution

Our solution to help with the food recommendation problem revolves around utilizing Artificial Intelligence and machine learning to help making the decision making process easier. We will be analyzing user preferences, historical choices, and real-time trends. User profiles will play a pivotal role, allowing individuals to specify dietary preferences, preferred cuisines, and ambiance choices. By prioritizing user privacy, employing advanced recommendation technology, and incorporating real-time updates, Eatonomy aims to be the forefront of the culinary exploration, setting a new standard for tailored food discovery."

### Chapter 2: Software Requirement Specification

##### Introduction

##### Purpose

This document's objective is to give a comprehensive overview of our software product, including its specifications and objectives. As this will outline what the developers must accomplish and the final product we hope to deliver, the invigilators, project advisers, and project leaders are the target audiences. This research project will aid future students in their research projects by compiling all the relevant data in one place. In an effort to improve knowledge and data gathering, we are working.

##### Scope

The project involves the development of a food recommender application for iOS and Android platforms. It will enable users to specify their culinary preferences, dietary restrictions, and the context of their dining occasions. The application, enhanced by artificial intelligence, will intelligently curate restaurant and menu recommendations that align with individual tastes. Eatonomy's primary aim is to deliver a tailored and user-friendly food ordering experience while promoting local restaurant growth. This scope aligns with the overarching project vision and maintains consistency with higher-level system requirements for a strategic and coherent development approach.

##### Definitions, acronyms, and abbreviations

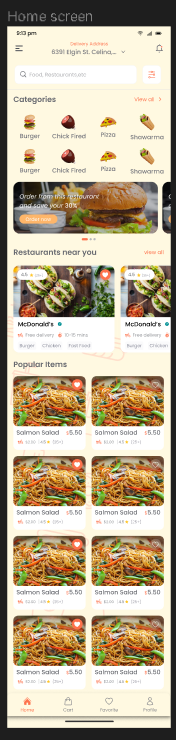
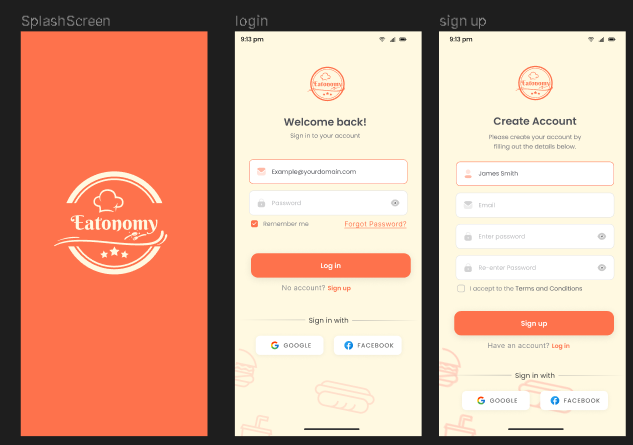
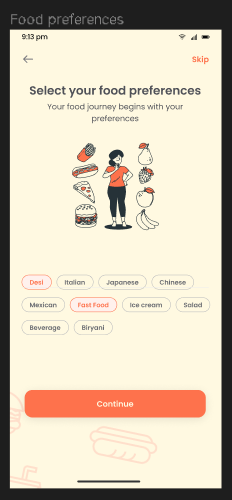
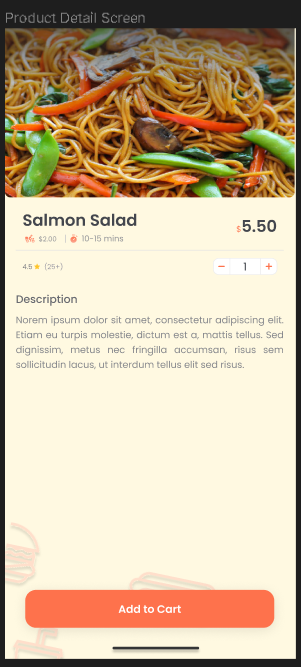
Acronyms:

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| AI | Artificial Intelligence |
| App | Application |
| CSS | Cascading Style Sheets |
| DFD | Data Flow Diagram |
| ERD | Entity Relationship Diagram |
| GPS | Global Positioning System |
| HTTPS | Hypertext Transfer Protocol Secure |
| iOS | iPhone Operating System |
| NoSQL | Not Only SQL |
| RAM | Random Access Memory |
| UML | Unified Modeling Language |
| UI | User Interface |

##### 

##### Overall description

##### Product perspective

* **System interfaces**: The Eatonomy app interfaces with a variety of systems to provide a seamless user experience. It connects with iOS and Android platforms, interacts with an artificial intelligence engine for personalized recommendations, accesses a restaurant database for up-to-date menu information, and integrates with payment gateways for secure transactions. The app also utilizes GPS and location services, user profile databases, and push notification services to enhance user engagement. These interfaces work in concert to deliver a tailored and efficient food-ordering solution to our users.
* **User interfaces: (define the layout of the user end including the screenshots)**   

###### *Figure 0 Example of User Interface*

* **Hardware interfaces:** Eatonomy interfaces with mobile devices on both iOS and Android platforms, utilizing hardware features such as GPS for location-based recommendations and network connectivity for data communication.
* **Software interfaces:** The device needs to have an internet connection and should have sufficient storage to download the app.
* **Communications interfaces:** Communication interfaces in the app ensures secure data transmission via HTTPS for order placement, profile updates, and recommendation retrieval. Users are required to have a valid Gmail ID for authentication. The interfaces support real-time order management, partnerships with local restaurants and delivery services, secure payment processing, and user notifications.
* **Memory:** The app is at least 100 MB in size, which includes executable code, resources, and data. Memory management is optimized to work within the constraints of a user's device, efficiently utilizing both primary (RAM) and secondary memory (storage) resources to provide a smooth user experience.
* **Operations:** For data backup and recovery in our app, user-specific data, including preferences, order history, and other relevant information, is stored in a secure database associated with the user's Gmail account. This approach ensures data integrity and allows for seamless recovery in the event of data loss or system failures.
* **Site adaptation requirements:** As of the current version, Eatonomy does not include any site adaptations; however, it remains open to the possibility of incorporating such features in future iterations

##### Product functions

Product functions include the functional requirement of your project

Now what is the functional requirements suppose you have a module of Account

Registration in your project then the functional requirement for this module are Create Account, View Account, Delete Account, Update Account, Login Account, Logout Account, etc. and many more according to the situation you are in.

The following things are included in each of the functional requirement of your project

* ID
* Name
* Description
* Input
* Output
* Basic Workflow
* Requirements (optional)

For example, you create account functional requirement look like this and you have to follow this template for writing your functional requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_01** |  |  |  |
| Name: | Register |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This functionality enables users to create an account by entering accurate details. | **User Details (e.g., Name, Email, Password):** Muhammad Ali, 70109014@gmail.com, 12345 | **Account Creation Confirmation:** Confirmation message indicating the successful creation of the account. | **Internet Connectivity:** Essential for user interaction and communication with the server.  **Database Integration:** System must save user records in the database. | Muhammad Ali initiates the registration process by entering accurate details—name, email (70109014@gmail.com), and password (12345). With internet connectivity, he submits the information, prompting the system to validate the data and subsequently create a new user record in the database. A confirmation message is relayed to Muhammad Ali, acknowledging the successful account creation. If desired, he can immediately log in using the newly established credentials. The workflow prioritizes user guidance and error handling, ensuring a seamless and professional registration experience. |

##### *Table 1: Functional Requirements – Register*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_02** |  |  |  |
| Name: | Login |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature facilitates user access by allowing them to log in with their registered credentials. | **User Credentials (e.g., Email, Password):** 70109014@gmail.com, 12345 | **Login Approval:** Confirmation of successful login. | **Internet Connectivity:** Necessary for user interaction and communication with the server.  **Existing User Data:** The system should have previously saved user data for authentication. | Upon entering correct login information, such as email (70109014@gmail.com) and password (12345), the system authenticates the data. If successful, access is granted with a confirmation of login approval, redirecting the user to their dashboard. In case the entered data is not found, the system prompts the user to sign up for an account, ensuring a seamless and secure login experience. |

##### *Table 2: Functional Requirements - Login*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_03** |  |  |  |
| Name: | Forget Password |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to recover their forgotten passwords. | **User Identifier (e.g., email, username):** The information used to identify the user account.  Security **Verification (e.g., CAPTCHA, security questions):** To ensure the request is legitimate.  **New Password:** The desired new password for the user account. | **Success/Failure Message:** Confirmation that the password has been successfully reset or an error message if unsuccessful. | **Internet Connectivity required:**  Required for user interaction and communication with the server. | The **"Forget Password"** workflow begins when a user, unable to remember their password, clicks on the associated link. After providing their account identifier and passing security verification, a unique token or link is generated and sent to the user. Upon clicking the link, the user accesses a secure page to set a new password. Once the password is updated, the system confirms the change, and the user receives a success message. Error handling is integrated at each step to guide the user in case of issues, ensuring a smooth and secure forget password process. |

##### *Table 3: Functional Requirements – Forget Password*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_04** |  |  |  |
| Name: | Reset Password |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to securely reset their password in the event of forgetting or needing to update it. | **User Identifier (e.g., Email):** User provides the identifier associated with their account.  Security **Verification (e.g., CAPTCHA, Security Questions):** Ensures the legitimacy of the password reset request.  **New Password:** The desired new password for the user account. | **Reset Confirmation:** Confirmation message indicating the successful reset of the password. | **Internet Connectivity required:**  Required for user interaction and communication with the server. | The **Reset Password** workflow commences when users click the "Reset Password" link on the login page and provide their account identifier. After completing security verification measures, a unique token or link is generated and sent to the user. By clicking on this link, users gain access to a secure interface to set a new password, and upon confirmation, the system updates their password, concluding the process with a success message. This straightforward and secure workflow ensures a seamless experience for users seeking to reset their passwords. |

##### *Table 4: Functional Requirements – Reset Password*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_05** |  |  |  |
| Name: | Manage Account |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This functionality empowers users to maintain and update their profile details, including picture, name, phone number, address, and password. | **Function: Change Name**  New Name: Haider  **Function: Change Password**  New Password: 786786  **Function: Change Address**  New Address: Wapda Town | **Profile Modification Confirmation:**  Confirmation message indicating successful changes to the profile. | **Internet Connectivity:** Necessary for user interaction and communication with the server.  **Existing User Data:** The system should have previously saved user data for modification. | In the Manage Account workflow, users select the desired function (Change Name, Change Password, Change Address) and input new information. After clicking Submit, the system validates the data and updates the user's profile in the database. A confirmation message is then delivered, and the modified details are immediately reflected in the user's profile, with robust error handling in place for any potential issues during the process. |

##### *Table 5: Functional Requirements – Manage Account*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_06** |  |  |  |
| Name: | Food Preferences |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to narrow down restaurant options based on their food preferences. | **User Food Preferences:**  Preferences: Fast Food, Italian, and Desi | **Filtered Restaurant List:** A list of restaurants that match the user's selected food preferences. | **Internet Connectivity:** Necessary for user interaction and communication with the server. | To utilize the Food Preferences feature, users launch the Android application, log in with their official account credentials, and navigate to the preferences screen. Once there, users can select their food preferences, such as Fast Food, Italian, and Desi, to refine restaurant recommendations. The application then filters and presents a list of restaurants tailored to the chosen criteria, ensuring personalized dining options aligned with the user's taste preferences. |

##### *Table 6: Functional Requirements – Food preferences*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_07** |  |  |  |
| Name: | View Restaurants |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature provides users with a curated list of restaurants based on their location and ratings, enhancing the dining experience. | User Action:  Function: View Restaurants  Restaurants: KFC, Gloria Jeans, etc. | List of Restaurants: Displayed with detailed information, including ratings and operating hours. | Internet Connectivity: Essential for user interaction and real-time retrieval of restaurant data.  Restaurant Availability: The functionality relies on up-to-date and available restaurant information. | To access the View Restaurants feature, users launch the Android application, authenticate using official account credentials, and navigate to the preferences screen. Subsequently, users can choose to skip the selection process, prompting the application to present a comprehensive list of available restaurants meticulously shortlisted based on location and ratings. This approach ensures that users are presented with pertinent dining options, complete with essential details like ratings and operating hours, fostering an informed and personalized restaurant selection experience. |

##### *Table 7: Functional Requirements – View Restaurants*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_08** |  |  |  |
| Name: | Search for Restaurant and Food Items |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to search for specific restaurants and food items, enhancing the browsing and selection experience. | **Search Query:** Users input keywords related to restaurant names or specific food items. | **Search Results:** Displayed in a list format, including relevant restaurants and food items matching the search query. | Internet Connectivity: Essential for real-time retrieval of search results. | Users initiate the "Search for Restaurant and Food Items" feature by entering relevant keywords into the search bar. The system processes the query, retrieves matching results from the database, and displays a list of restaurants and food items meeting the specified criteria. Users can seamlessly explore and select options from the search results, providing an efficient and tailored experience for discovering preferred dining choices within the application. |

##### *Table 8: Functional Requirements - Search*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_09** |  |  |  |
| Name: | Restaurant Profile |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature provides users with comprehensive information about a specific restaurant, including its name, menu, availability, rating, and reviews. | **Function:** View Restaurant Profile  Restaurant **Identifier**: Unique identifier for the desired restaurant | **Restaurant Details:** Displayed information about the restaurant, including name, menu, availability, rating, and reviews. | **Internet Connectivity:** Necessary for real-time retrieval of restaurant profile data. | Users access the "Restaurant Profile" feature by selecting a specific restaurant or entering its unique identifier. The system processes the request, retrieving and prominently displaying essential details such as the restaurant's name, menu items, availability status, overall rating, and customer reviews. This streamlined workflow empowers users to make informed dining decisions by providing comprehensive insights into a chosen restaurant's offerings and reputation. |

##### *Table 9: Functional Requirements – Restaurant Details*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_10** |  |  |  |
| Name: | Food Categories |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature categorizes restaurants and food items, allowing users to explore specific types of cuisine. | **Category Selection:** Users choose a specific category, such as Burger, Pizza, Shawarma, etc. | **Category Contents:** Displayed list of restaurants and food items falling under the selected category. | Internet Connectivity: Necessary for real-time retrieval of category-specific data. | Users access the "Categories" feature by selecting a specific category, such as Burger, Pizza, or Shawarma. The system then retrieves and displays a list of restaurants and food items falling under the chosen category, allowing users to explore and select options within their preferred cuisine. This streamlines the user experience, facilitating efficient navigation and discovery of food choices based on specific categories. |

##### *Table 10: Functional Requirements – Food categories*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_11** |  |  |  |
| Name: | Popular Items |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature showcases a curated list of popular food items based on user preferences and ratings. | **User Preferences:** The system considers individual user preferences and historical ordering data. | **Popular Items Display:** A dynamic list of food items popular among users, accompanied by ratings and recommendations | **Internet Connectivity:** Essential for real-time retrieval and updating of popular item data. | The "Popular Items" feature leverages user preferences and historical data to compile a dynamic list of highly-rated and frequently chosen food items. Users are presented with this curated selection, allowing them to explore and select popular dishes directly, enhancing their ordering experience with recommendations that align with broader preferences. The system continuously updates popular items based on real-time user feedback, ensuring the displayed choices remain relevant and reflective of current user preferences. |

##### *Table 11: Functional Requirements – Popular Items*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_12** |  |  |  |
| Name: | View Cart |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature enables users to review their selected items in the cart along with the total amount, providing a transparent overview of their order. | **User Action:**  **Function: View Cart**  Cart: Burger, Pizza, Naan Chana | **Selected Items in Cart:** Displayed with individual prices and a prominently featured total amount. | **Internet Connectivity:** Essential for user interaction and real-time retrieval of cart data. | By clicking the dedicated cart icon, users can seamlessly examine the contents of their cart, revealing a comprehensive list of the items they have chosen, including Burger, Pizza, and Naan Chana. Each food item is associated with its respective price, and for a thorough overview, the total cost of all selected items is prominently displayed. This approach ensures users have a clear and detailed summary of their order, facilitating a more informed decision-making process. |

##### *Table 12: Functional Requirements – View Cart*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_13** |  |  |  |
| Name: | View Favorites |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to access and view their favorite restaurants and items for a personalized and efficient ordering experience. | **User Action:**  **Function:** View Favorites | **Favorite Restaurants and Items:** Displayed with relevant details, such as restaurant names and favorite food items. | **Internet Connectivity:** Essential for user interaction and real-time retrieval of favorite data. | By selecting the View Favorites function, users can conveniently access a curated list of their favorite restaurants and items. This personalized view includes pertinent details such as restaurant names and specific favorite food items. This feature aims to streamline the ordering process for users, offering quick access to their preferred choices for a more tailored and enjoyable dining experience. |

##### *Table 13: Functional Requirements – View Favorites*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_14** |  |  |  |
| Name: | Order History |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to access and view both their current and previous orders, providing a comprehensive overview of their order history. | **User Action:**  **Function:** View Order History | **Order Details:** Displayed in a list format, including information on current and completed orders. | **Internet Connectivity:**  Essential for real-time retrieval and display of order history. | Users access the "Order History" feature to review both current and completed orders. The system retrieves and displays relevant details, allowing users to explore their current orders' status and revisit information on previous transactions for a comprehensive overview of their ordering history. |

##### *Table 14: Functional Requirements – Order History*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID: | **FR\_15** |  |  |  |
| Name: | Payment Method - Cash on Delivery (COD) |  |  |  |
| Description | Input | Output | Requirements | Basic Work Flow |
| This feature allows users to make payments for their orders using the Cash on Delivery (COD) method. | **User Selection:**  **Function:** Select Payment Method  **Payment Method:** Cash on Delivery | **Payment Confirmation:**  Confirmation message acknowledging the selection of Cash on Delivery as the payment method. | **Internet Connectivity:**  Necessary for the user to interact with the application and confirm the payment method. | Users select the "Payment Method" during the checkout process and specifically opt for Cash on Delivery. The system confirms the selection with a message, and upon order completion, users pay in cash upon the delivery of their items. This streamlined workflow ensures a seamless and user-friendly experience for customers choosing the Cash on Delivery payment method. |

##### *Table 15: Functional Requirements – Payment*

##### User characteristics

Users of this app are individuals who want it to suggest the best local restaurants for them and their families based on their cuisine preferences.

##### Constraints

Our food app Eatonomy faces hardware and usability limitations, supporting newer Android versions only. While the initial interface might seem unfamiliar, it's designed for a user-friendly experience. Seamless app integration with other apps is a work in progress, and location control is available for preferences. Two-factor authentication and password encryption will be added for enhanced security, and regulatory compliance is a top priority.

##### Assumptions and dependencies

Within the framework of our app, we operate under the assumption that essential shop-related details, including the shop's name, menu, and operating hours, will be provided manually to the system administrator. The administrator's responsibility is to accurately input this information into the app's database, ensuring that it is readily available for user consumption. This seamless collaboration between the shop owners and the administrator is a critical dependency for the successful and accurate functioning of the app, as it forms the foundation of our service's core data.

##### Apportioning of Requirements

In our approach to the apportioning of requirements for the Eatonomy app, we will prioritize core features for the initial release, including user profile creation, restaurant search, and recommendation algorithms. Subsequent phases will focus on incorporating advanced functionalities such as a chat feature for customer-restaurant owner interaction, user reviews, and enhanced filtering options, allowing us to progressively expand the app's capabilities and provide a dynamic and responsive platform for food discovery and dining experiences.

##### Specific requirements

##### Functional Requirement

**User registration and profiles:**

Users can create and manage their profiles, including personal information, dietary preferences, and location.

**Restaurant Data Management:**

The system will store and manage data related to restaurants, including names, menus, operating hours, cuisine types, and locations.

**Recommendation Algorithm:**

Implement an algorithm that suggests restaurants to users based on their profile information and preferences.

**Search and Filtering:**

Users can search for restaurants by name, location, cuisine type, and specific dishes. Advanced filtering options are available.

**User Reviews and Ratings:**

Users can leave reviews and ratings for restaurants. The system will calculate average ratings and display them for other users.

**User-Owner Chat Feature:**

Introduce a chat feature for direct communication between customers and restaurant owners, enabling users to make inquiries, reservations, or special requests.

**Location-Based Services:**

Implement location services to show restaurants nearby and enable users to set their preferred location for recommendations.

**Order Placement and Payment:**

Allow users to place orders from restaurants and provide payment options, including online payment or in-person payment upon delivery or pickup.

**User Notifications:**

Send notifications to users about new restaurant recommendations, order status updates, and promotions.

**Security and Privacy:**

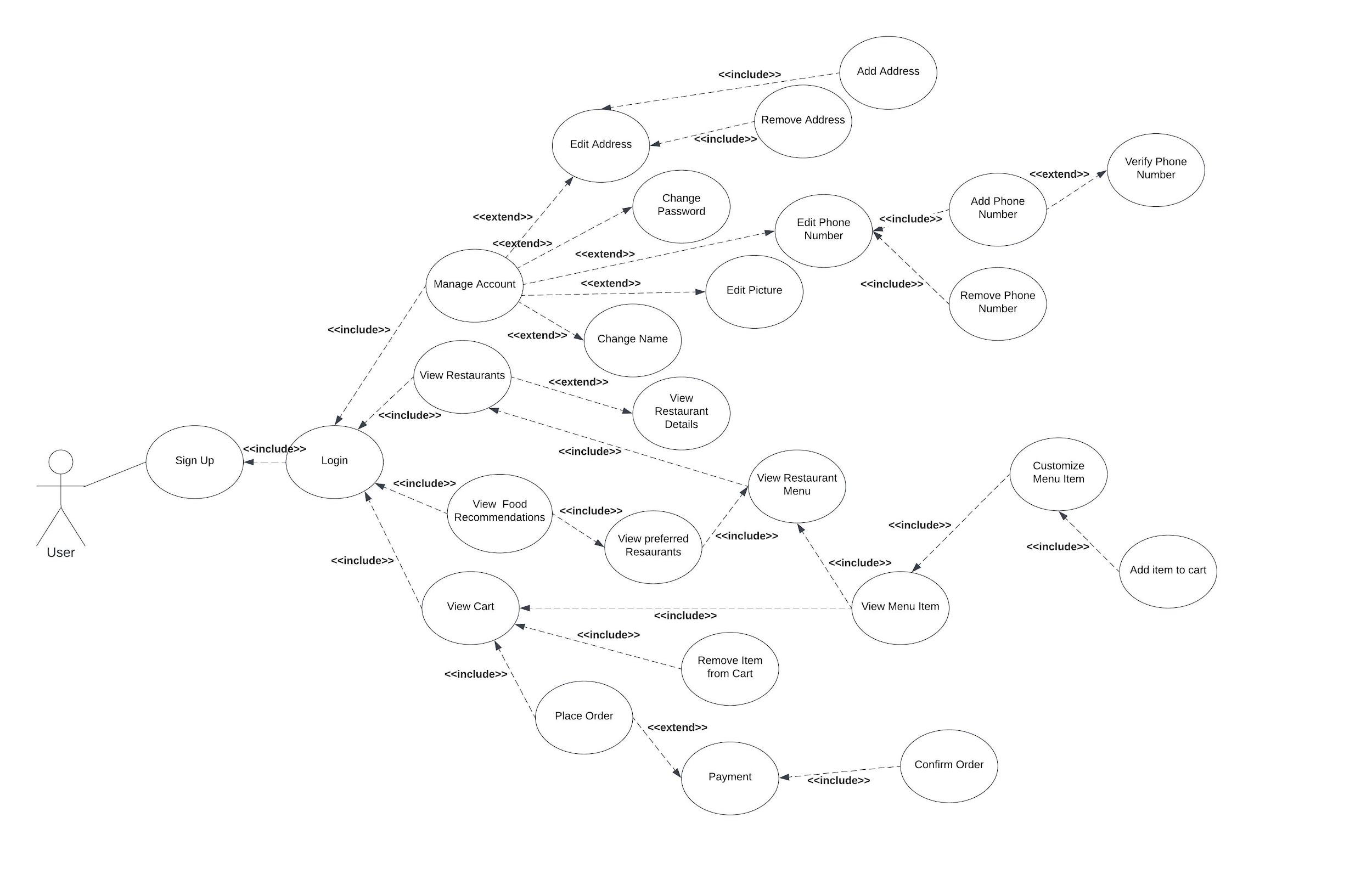
Ensure user data is securely stored and transmitted. Implement user authentication and password encryption for data protection.

##### Non-functional Requirements

This sub-section includes the following

* Usability
* Reliability
* Performance
* Design Constraints
* Portability
* Maintainability
* License Agreement

### Chapter 3: Use Case Analysis



###### *Figure 1 Usecase Diagram - User*

**Use Case: Sign Up (UC\_01)**

###### 

###### *Figure 2 Use Case: Sign Up*

|  |  |
| --- | --- |
| Use Case ID | UC\_01 |
| Use Case Name | Sign Up |
| Description | The Sign Up use case allows users to create an account by providing necessary details such as name, email, and password. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | None |
| Post-Condition | Access to user account |
| Basic Flow | 1. Click on "Sign Up"  2. Enter required details  3. Click "Register" or "Login" |
| Alternate Flow | Ask an admin to create or add the user |

##### *Table 16: Use Case – Customer Sign Up*

**Use Case: Login (UC\_02)**

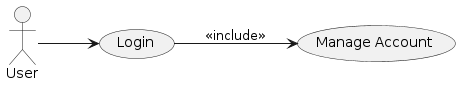
###### 

###### *Figure 3 Use Case: Login*

|  |  |
| --- | --- |
| Use Case ID | UC\_02 |
| Use Case Name | Login |
| Description | The Login use case enables users to access their account by entering their registered email and password. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Registered account |
| Post-Condition | Access to user account |
| Basic Flow | 1. Enter email and password  2. Click "Login" |
| Alternate Flow | None |

##### *Table 17: Use Case – Customer Login*

**Use Case: Manage Account (UC\_03)**

****

###### *Figure 4 Use Case: Manage Account*

|  |  |
| --- | --- |
| Use Case ID | UC\_03 |
| Use Case Name | Manage Account |
| Description | The Manage Account use case allows users to modify various account details such as address, password, phone number, picture, and name. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated account details |
| Basic Flow | 1. Click "Manage Account"  2. Choose from options: Edit Address, Change Password, Edit Phone Number, Edit Picture, Change Name  3. Follow relevant sub-flows |
| Alternate Flow | Ask an admin to make changes |

##### *Table 18: Use Case – Customer Manage Account*

**Use Case: Edit Address (UC\_04)**

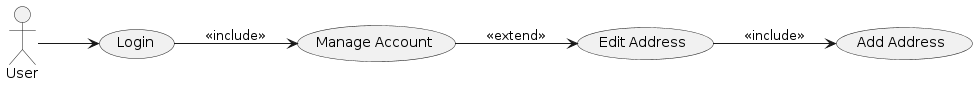
****

###### *Figure 5 Use Case: Edit Address*

|  |  |
| --- | --- |
| Use Case ID | UC\_04 |
| Use Case Name | Edit Address |
| Description | The Edit Address sub-use case allows users to add or remove addresses from their account. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated address list |
| Basic Flow | 1. Click "Edit Address"  2. Choose from options: Add Address, Remove Address  3. Follow relevant sub-flows |
| Alternate Flow | None |

##### *Table 19: Use Case – Customer Edit Address*

**Use Case: Add Address (UC\_05)**

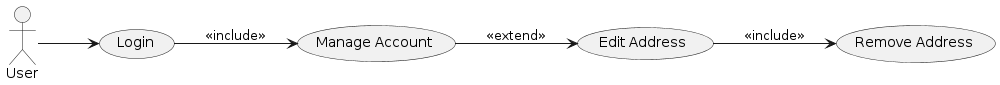
****

###### *Figure 6 Use Case: Add Address*

|  |  |
| --- | --- |
| Use Case ID | UC\_05 |
| Use Case Name | Add Address |
| Description | The Add Address sub-use case allows users to add a new address to their account. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated address list |
| Basic Flow | 1. Click "Add Address"  2. Enter new address details  3. Confirm and save |
| Alternate Flow | None |

##### *Table 20: Use Case – Customer Add Address*

**Use Case: Remove Address (UC\_06)**

****

###### *Figure 7 Use Case: Remove Address*

|  |  |
| --- | --- |
| Use Case ID | UC\_06 |
| Use Case Name | Remove Address |
| Description | The Remove Address sub-use case allows users to delete an existing address from their account. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated address list |
| Basic Flow | 1. Click "Remove Address"  2. Select address to remove  3. Confirm deletion |
| Alternate Flow | None |

##### *Table 21: Use Case – Customer Remove Address*

**Use Case: Change Password (UC\_07)**

****

###### *Figure 8 Use Case: Change Password*

|  |  |
| --- | --- |
| Use Case ID | UC\_07 |
| Use Case Name | Change Password |
| Description | The Change Password sub-use case allows users to update their account password. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated password |
| Basic Flow | 1. Click "Change Password"  2. Enter current and new password  3. Confirm and save |
| Alternate Flow | None |

##### *Table 22: Use Case – Customer Change Password*

**Use Case: Edit Phone Number (UC\_08)**

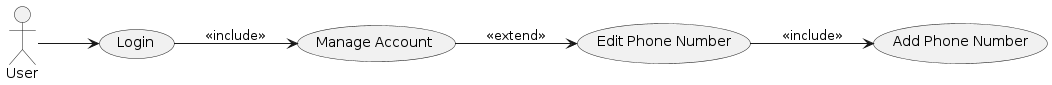
****

###### *Figure 9 Use Case: Edit Phone Number*

|  |  |
| --- | --- |
| Use Case ID | UC\_08 |
| Use Case Name | Edit Phone Number |
| Description | The Edit Phone Number sub-use case allows users to add or remove phone numbers from their account. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated phone number list |
| Basic Flow | 1. Click "Edit Phone Number"  2. Choose from options: Add Phone Number, Remove Phone Number  3. Follow relevant sub-flows |
| Alternate Flow | None |

##### *Table 23: Use Case – Customer Edit Phone Number*

**Use Case: Add Phone Number (UC\_09)**

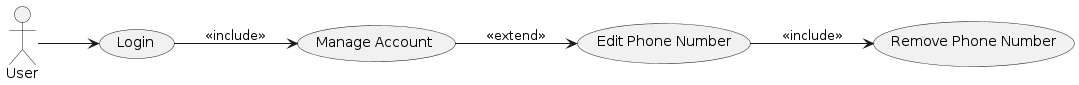
****

###### *Figure 10 Use Case: Add Phone Number*

|  |  |
| --- | --- |
| Use Case ID | UC\_09 |
| Use Case Name | Add Phone Number |
| Description | The Add Phone Number sub-use case allows users to add a new phone number to their account and verify it. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated phone number list |
| Basic Flow | 1. Click "Add Phone Number"  2. Enter new phone number  3. Verify phone number  4. Confirm and save |
| Alternate Flow | None |

##### *Table 24: Use Case – Customer Add Phone Number*

**Use Case: Remove Phone Number (UC\_10)**

****

###### *Figure 11 Use Case: Remove Phone Number*

|  |  |
| --- | --- |
| Use Case ID | UC\_10 |
| Use Case Name | Remove Phone Number |
| Description | The Remove Phone Number sub-use case allows users to delete an existing phone number from their account. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated phone number list |
| Basic Flow | 1. Click "Remove Phone Number"  2. Select phone number to remove  3. Confirm deletion |
| Alternate Flow | None |

##### *Table 25: Use Case – Customer Remove Phone Number*

**Use Case: Edit Picture (UC\_11)**

****

###### *Figure 12 Use Case: Edit Picture*

|  |  |
| --- | --- |
| Use Case ID | UC\_11 |
| Use Case Name | Edit Picture |
| Description | The Edit Picture sub-use case allows users to update their profile picture. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated profile picture |
| Basic Flow | 1. Click "Edit Picture"  2. Upload a new profile picture  3. Confirm and save |
| Alternate Flow | None |

##### *Table 26: Use Case – Customer Edit Picture*

**Use Case: Change Name (UC\_12)**

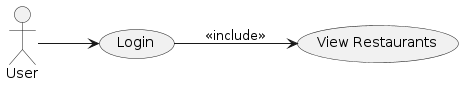
****

###### *Figure 13 Use Case: Change Name*

|  |  |
| --- | --- |
| Use Case ID | UC\_12 |
| Use Case Name | Change Name |
| Description | The Change Name sub-use case allows users to update their displayed name. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated displayed name |
| Basic Flow | 1. Click "Change Name"  2. Enter new name  3. Confirm and save |
| Alternate Flow | None |

##### *Table 27: Use Case – Customer Change Name*

**Use Case: View Restaurants (UC\_13)**

****

###### *Figure 14 Use Case: View Restaurants*

|  |  |
| --- | --- |
| Use Case ID | UC\_13 |
| Use Case Name | View Restaurants |
| Description | The View Restaurants use case provides users with a list of available restaurants based on location and ratings. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | List of available restaurants |
| Basic Flow | 1. Click "View Restaurants"  2. View list of restaurants based on location and ratings |
| Alternate Flow | None |

##### *Table 28: Use Case – Customer View Restaurants*

**Use Case: View Restaurant Details (UC\_14)**

****

###### *Figure 15 Use Case: View Restaurant Details*

|  |  |
| --- | --- |
| Use Case ID | UC\_14 |
| Use Case Name | View Restaurant Details |
| Description | The View Restaurant Details sub-use case displays comprehensive information about a specific restaurant, including its name, menu, availability, rating, and reviews. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Detailed restaurant information |
| Basic Flow | 1. Click "View Restaurant Details"  2. Enter restaurant identifier  3. View detailed information |
| Alternate Flow | None |

##### *Table 29: Use Case – Customer View Restaurant Details*

**Use Case: View Restaurant Menu (UC\_15)**

****

###### *Figure 16 Use Case: View Restaurant Menu*

|  |  |
| --- | --- |
| Use Case ID | UC\_15 |
| Use Case Name | View Restaurant Menu |
| Description | The View Restaurant Menu sub-use case shows the menu of a specific restaurant. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Restaurant menu |
| Basic Flow | 1. Click "View Restaurant Menu"  2. Enter restaurant identifier  3. View menu |
| Alternate Flow | None |

##### *Table 30: Use Case – Customer View Restaurant Menu*

**Use Case: View Food Recommendations (UC\_16)**

**A black and white diagram of a fish

Description automatically generated**

###### *Figure 17 Use Case: View Food Recommendations*

|  |  |
| --- | --- |
| Use Case ID | UC\_16 |
| Use Case Name | View Food Recommendations |
| Description | The View Food Recommendations use case allows users to discover personalized food recommendations based on their preferences. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | List of recommended restaurants |
| Basic Flow | 1. Click "View Food Recommendations"  2. View personalized recommendations |
| Alternate Flow | None |

##### *Table 31: Use Case – Customer View Food Recommendations*

**Use Case: View Preferred Restaurants (UC\_17)**

**A black and white diagram with a white arrow pointing to a white circle

Description automatically generated with medium confidence**

###### *Figure 18 Use Case: View Preferred Restaurants*

|  |  |
| --- | --- |
| Use Case ID | UC\_17 |
| Use Case Name | View Preferred Restaurants |
| Description | The View Preferred Restaurants sub-use case shows users a list of their favorite restaurants. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | List of favorite restaurants |
| Basic Flow | 1. Click "View Preferred Restaurants"  2. View list of favorite restaurants |
| Alternate Flow | None |

##### *Table 32: Use Case – Customer View Preferred Restaurants*

**Use Case: View Restaurant Menu (Preferred) (UC\_18)**

****

###### *Figure 19 Use Case: View Restaurant Menu (Preferred)*

|  |  |
| --- | --- |
| Use Case ID | UC\_18 |
| Use Case Name | View Restaurant Menu (Preferred) |
| Description | The View Restaurant Menu sub-use case displays the menu of a specific restaurant within the preferred list. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Restaurant menu |
| Basic Flow | 1. Click "View Restaurant Menu"  2. Enter restaurant identifier |
| Alternate Flow | None |

##### *Table 33: Use Case – Customer View Preferred Restaurant Menu*

**Use Case: View Cart (UC\_19)**

**A white and black text

Description automatically generated**

###### *Figure 20 Use Case: View Cart*

|  |  |
| --- | --- |
| Use Case ID | UC\_19 |
| Use Case Name | View Cart |
| Description | The View Cart use case enables users to review the selected items in their cart along with the total amount. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Overview of selected items and total amount |
| Basic Flow | 1. Click "View Cart"  2. View list of selected items with prices  3. Display total amount |
| Alternate Flow | None |

##### *Table 34: Use Case – Customer View Cart*

**Use Case: Remove Item from Cart (UC\_20)**

****

###### *Figure 21 Use Case: Remove Item from Cart*

|  |  |
| --- | --- |
| Use Case ID | UC\_20 |
| Use Case Name | Remove Item from Cart |
| Description | The Remove Item from Cart sub-use case allows users to delete a selected item from their cart. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Updated cart with removed item |
| Basic Flow | 1. Click "Remove Item from Cart"  2. Select item to remove  3. Confirm removal |
| Alternate Flow | None |

##### *Table 35: Use Case – Customer Remove Item from Cart*

**Use Case: Place Order (UC\_21)**

**A close-up of a sign

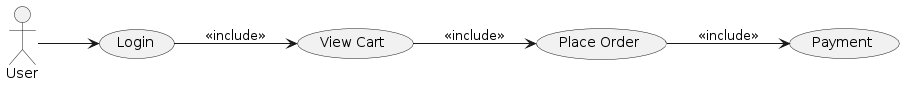
Description automatically generated**

###### *Figure 22 Use Case: Place Order*

|  |  |
| --- | --- |
| Use Case ID | UC\_21 |
| Use Case Name | Place Order |
| Description | The Place Order use case allows users to confirm and place an order for the selected items in their cart. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Order placed |
| Basic Flow | 1. Click "Place Order"  2. Proceed to payment  3. Confirm order |
| Alternate Flow | Payment confirmation |

##### *Table 36: Use Case – Customer Place Order*

**Use Case: Payment (UC\_22)**

****

###### *Figure 23 Use Case: Payment*

|  |  |
| --- | --- |
| Use Case ID | UC\_22 |
| Use Case Name | Payment |
| Description | The Payment sub-use case involves the user selecting a payment method and confirming the order. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Order confirmed |
| Basic Flow | 1. Select payment method  2. Confirm order |
| Alternate Flow | None |

##### *Table 37: Use Case – Customer Payment*

**Use Case: Confirm Order (UC\_23)**

****

###### *Figure 24 Use Case: Confirm Order*

|  |  |
| --- | --- |
| Use Case ID | UC\_23 |
| Use Case Name | Confirm Order |
| Description | The Confirm Order sub-use case finalizes the order placement, and the system acknowledges the successful transaction. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Order confirmed |
| Basic Flow | 1. Confirm order  2. Receive confirmation message |
| Alternate Flow | None |

##### *Table 38: Use Case – Customer Confirm Order*

**Use Case: View Favorites (UC\_24)**

**A black text on a white background

Description automatically generated**

###### *Figure 25 Use Case: View Favorites*

|  |  |
| --- | --- |
| Use Case ID | UC\_24 |
| Use Case Name | View Favorites |
| Description | The View Favorites use case allows users to access and view their favorite restaurants and items for a personalized ordering experience. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | List of favorite restaurants and items |
| Basic Flow | 1. Click "View Favorites"  2. View list of favorite restaurants and items |
| Alternate Flow | None |

##### *Table 39: Use Case – Customer View Favorites*

**Use Case: Order History (UC\_25)**

**A black text on a white background

Description automatically generated**

###### *Figure 26 Use Case: Order History*

|  |  |
| --- | --- |
| Use Case ID | UC\_25 |
| Use Case Name | Order History |
| Description | The Order History use case allows users to access and view both their current and previous orders. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | List of current and previous orders |
| Basic Flow | 1. Click "Order History"  2. View list of orders |
| Alternate Flow | None |

##### *Table 40: Use Case – Customer Order History*

**Use Case: Payment Method - Cash on Delivery (COD) (UC\_26)**

**A diagram of a flowchart

Description automatically generated**

###### *Figure 27 Use Case: Payment Method - Cash on Delivery*

|  |  |
| --- | --- |
| Use Case ID | UC\_26 |
| Use Case Name | Payment Method - Cash on Delivery (COD) |
| Description | The Payment Method - Cash on Delivery (COD) use case allows users to make payments for their orders using the Cash on Delivery method. |
| Primary Actor | User |
| Secondary Actor | - |
| Pre-Condition | Logged in |
| Post-Condition | Order confirmed with COD payment |
| Basic Flow | 1. Select "Cash on Delivery" as payment method  2. Confirm order |
| Alternate Flow | None |

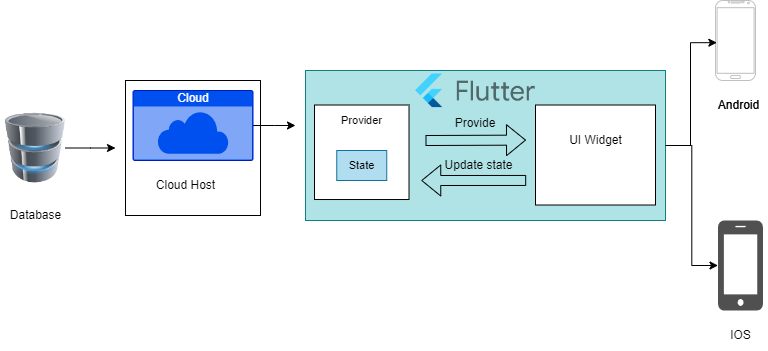
##### *Table 41: Use Case – Customer Payment Method*

# Chapter 4: Design

In this section, we provide the design analysis of our modules including the following designs

1. Architecture Diagram
2. ERD with data dictionary
3. Data Flow diagram
4. Class Diagram
5. Activity Diagram
6. Sequence Diagram
7. Collaboration Diagram
8. State Transition Diagram
9. Component Diagram
10. Deployment Diagram

#### Architecture Diagram

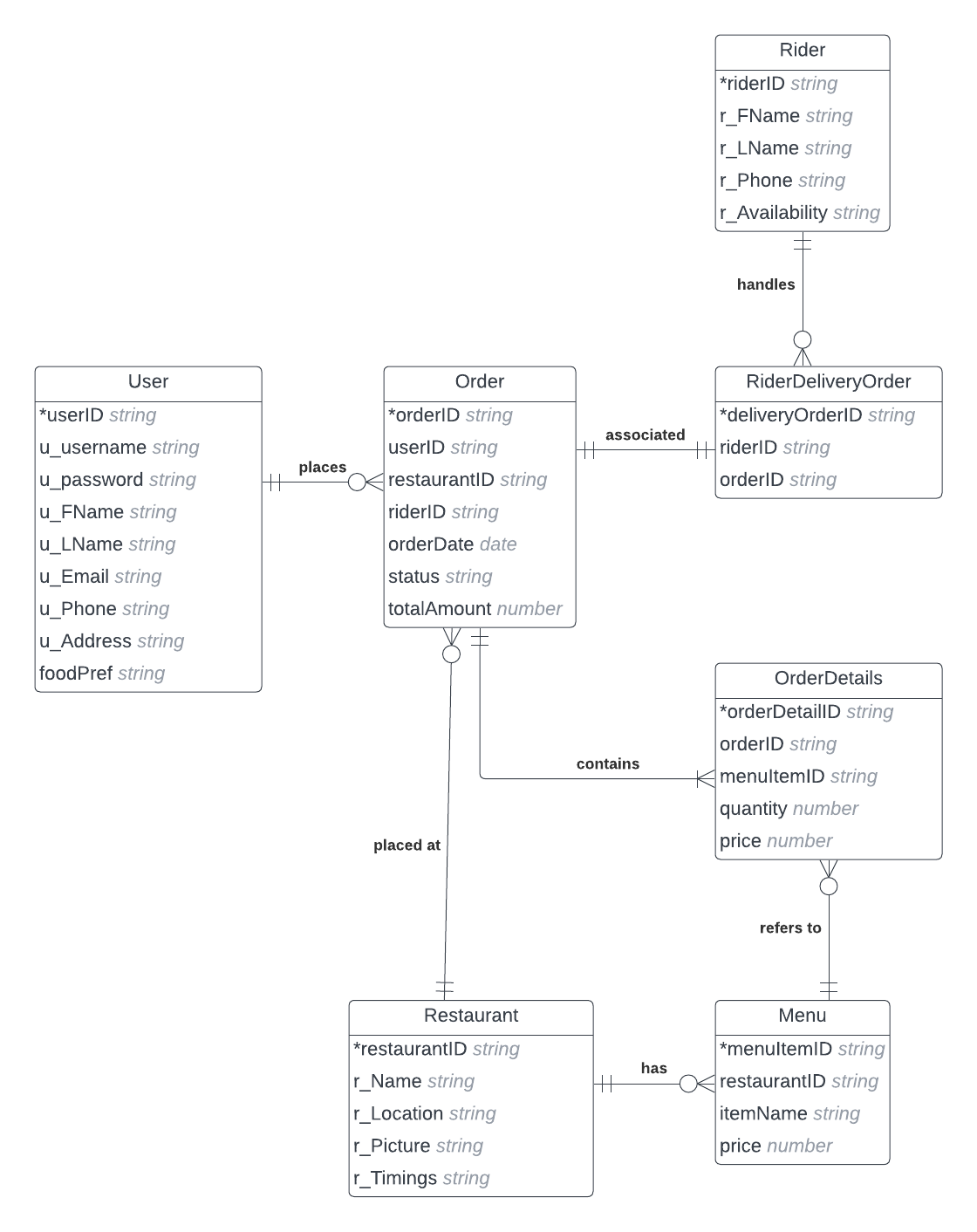


###### *Figure 28 Architecture Diagram*

###### 

#### ERD with data dictionary

EntityRelationshipDiagram with complete relations with dependencies of your project



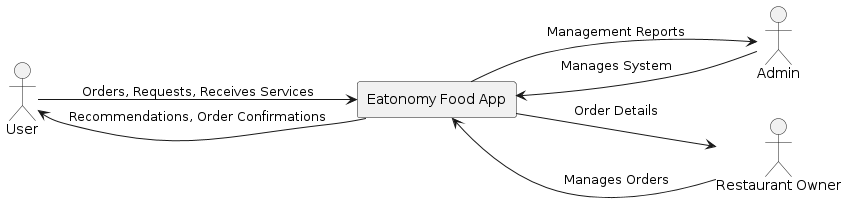
###### *Figure 29 ERD*

#### Data Flow diagram

Data flow diagram includes two levels

##### 4.3.1 The level 0

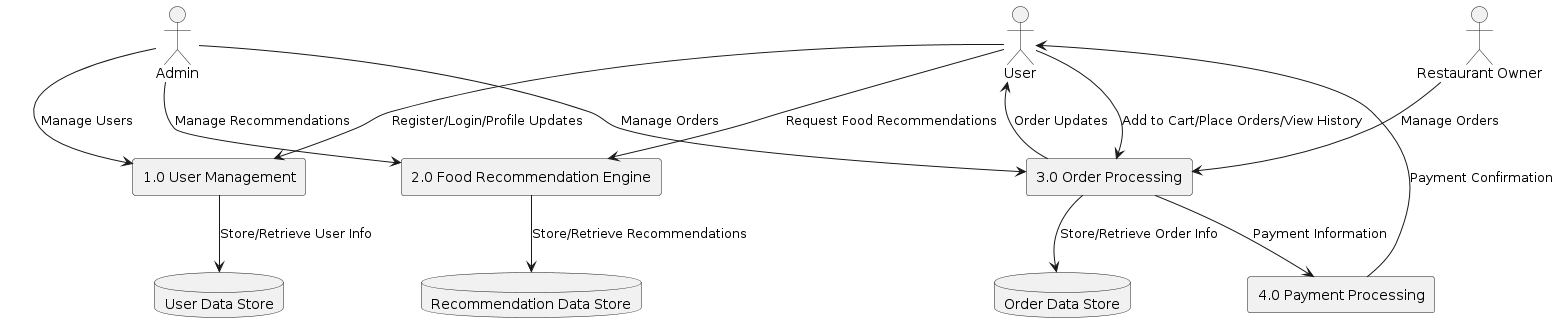
The flow of information inside the system is defined in this level



###### *Figure 30 Level 0 DFD*

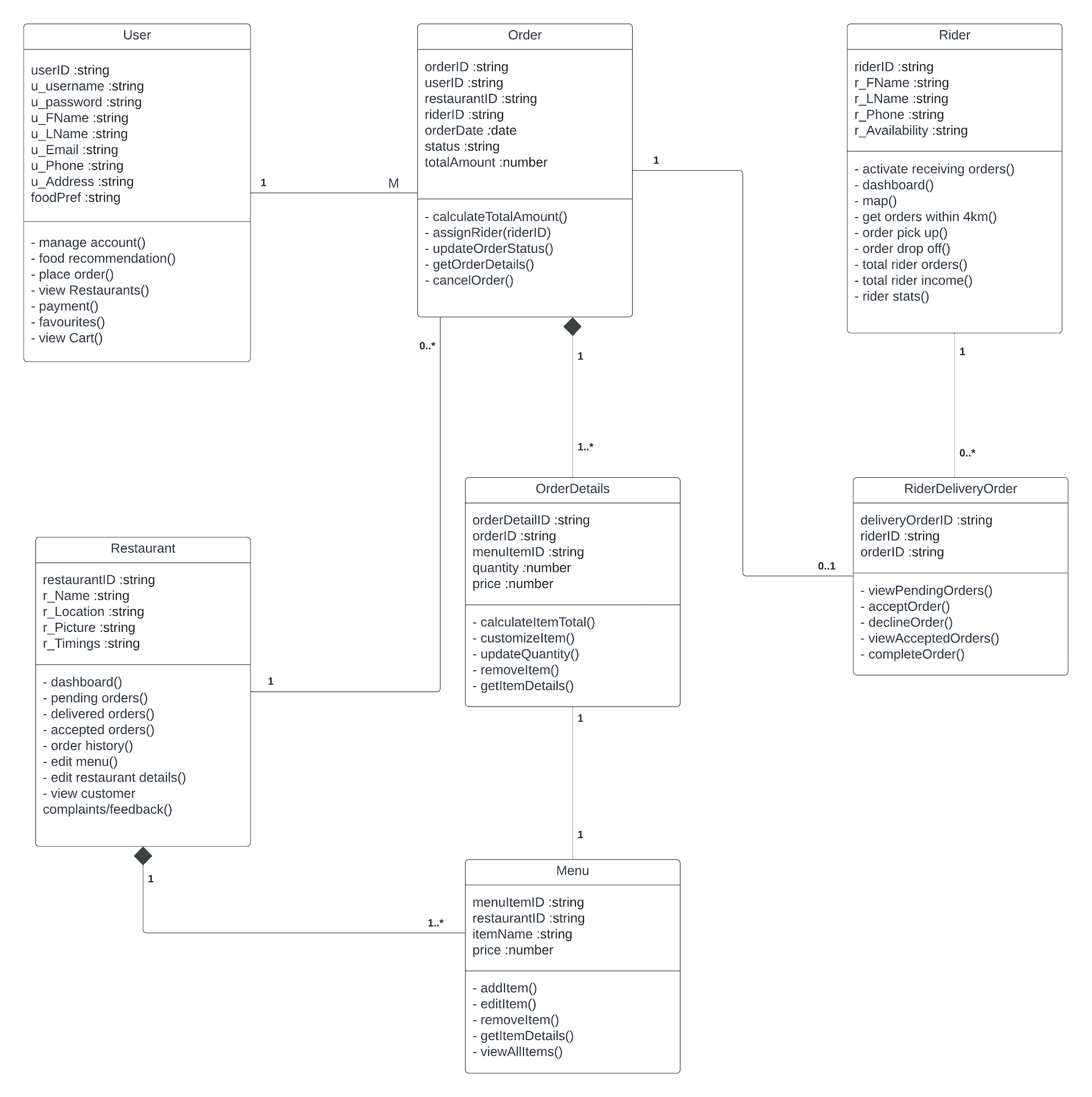
4.3.1 The level 1

The flow of information outside the system is defined in this level



###### *Figure 31 Level 1 DFD*

#### Class Diagram

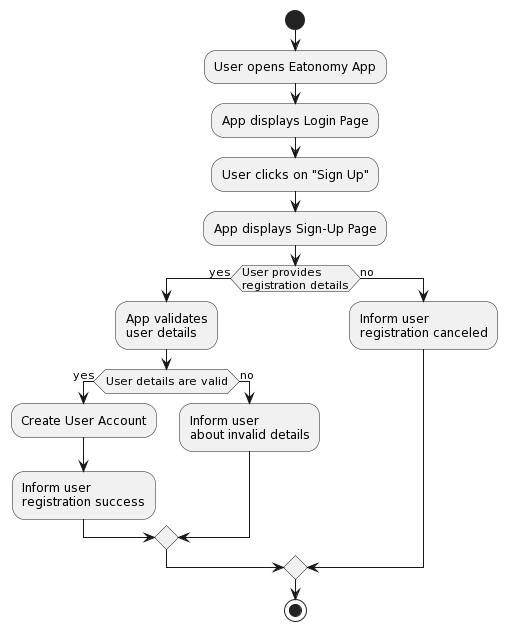
Describe the structure of a project by showing the systems classes, their attributes, operations (or methods), and the relationships among objects. 

###### *Figure 32 Class Diagram*

#### Activity Diagram

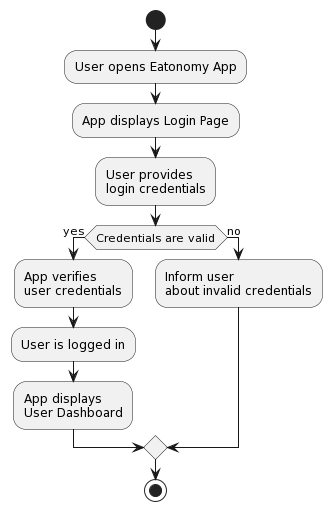
This diagram includes all the activity diagrams of the functional requirements of our project along with the aggregated activity diagram

##### 4.5.1 Activity diagram for create account



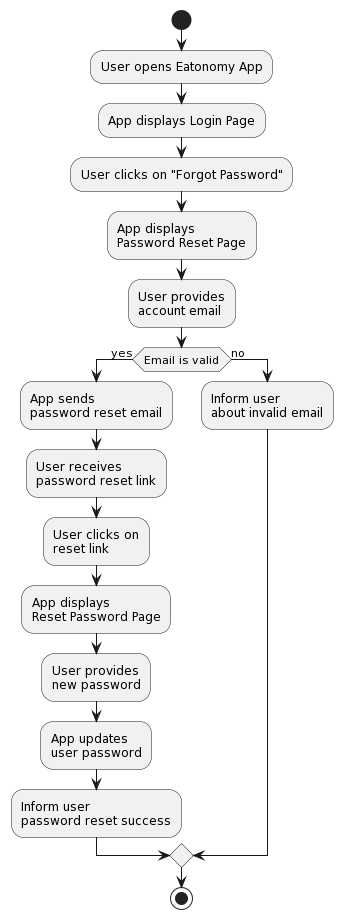
###### *Figure 33 Activity diagram Create Account*

##### 4.5.2 Activity diagram for login account



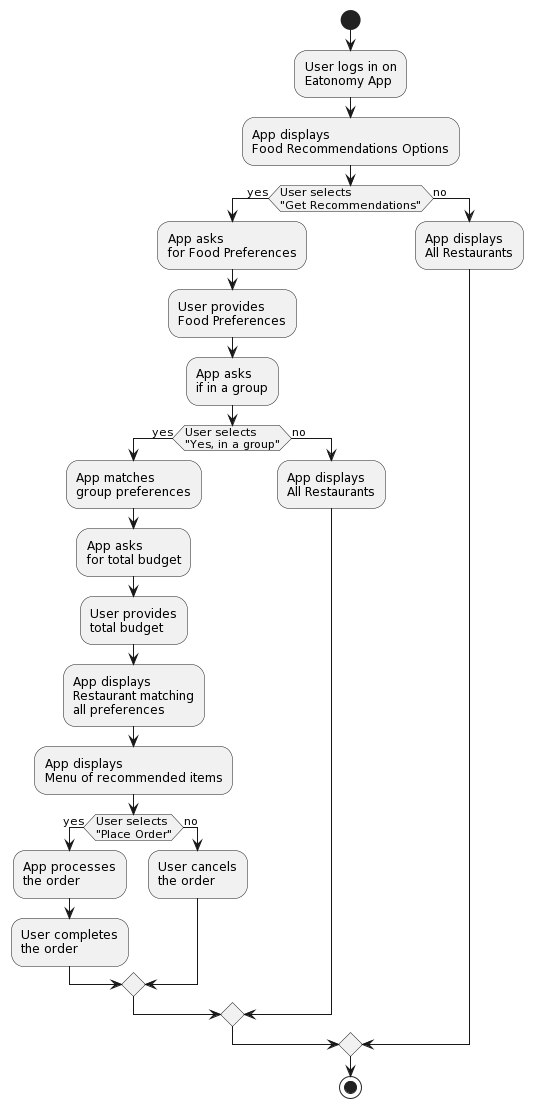
###### *Figure 34 Activity Diagram Login Account*

##### 4.5.3 Activity diagram for forgot password.



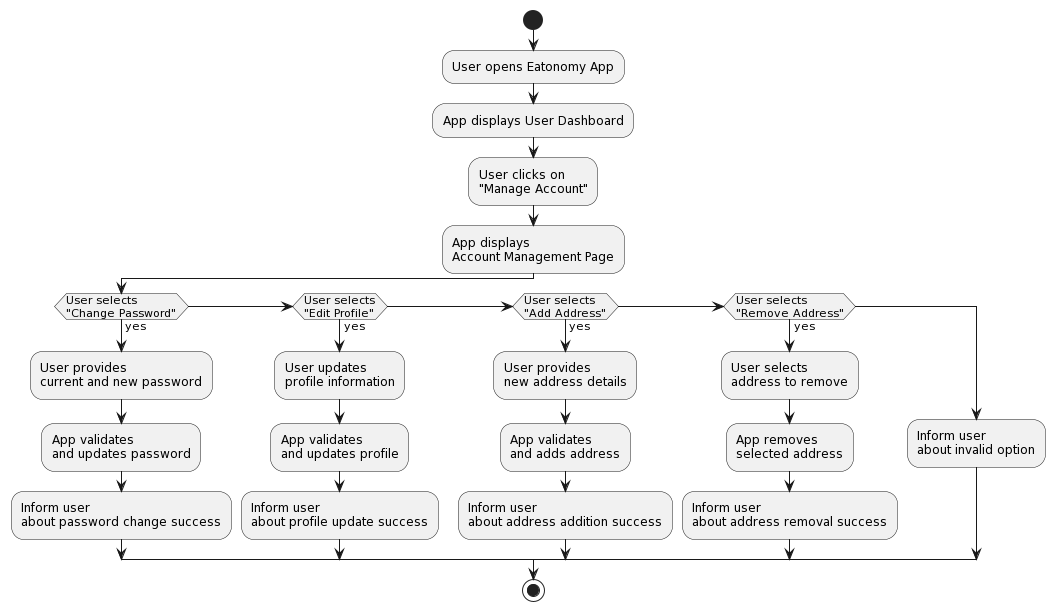
###### *Figure 35 Activity Diagram Forgot password*

##### 4.5.4 Activity diagram for Food Recommendation



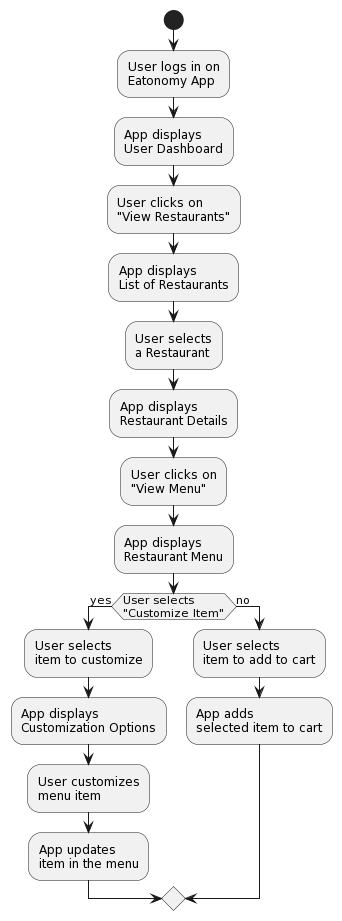
###### *Figure 36 Activity Diagram Food Recommendation*

##### 4.5.5 Activity diagram for account management



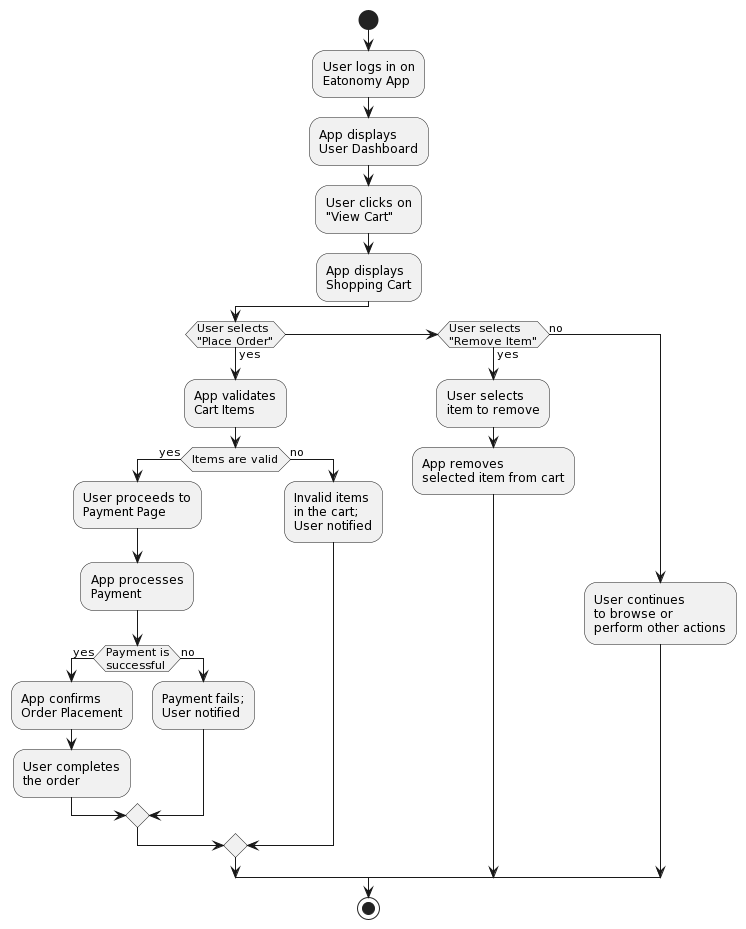
###### *Figure 37 Activity Diagram Account Management*

##### 4.5.6 Activity diagram for View Restaurant



###### *Figure 38 Activity Diagram View Restaurant*

##### 4.5.7 Activity diagram for View Cart

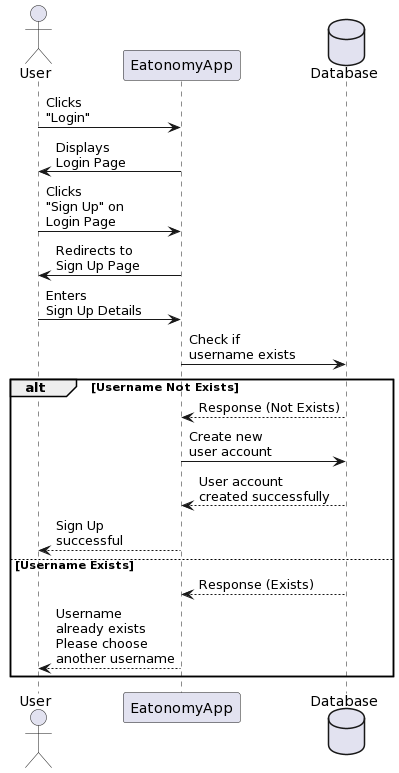


###### *Figure 39 Activity Diagram View Cart*

#### Sequence Diagram

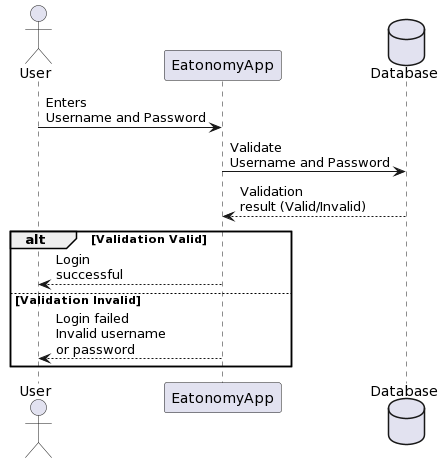
This diagram includes all the Sequence diagrams of the functional requirements of our project along with the aggregated Sequence diagram

##### 4.6.1 Sequence diagram for create account



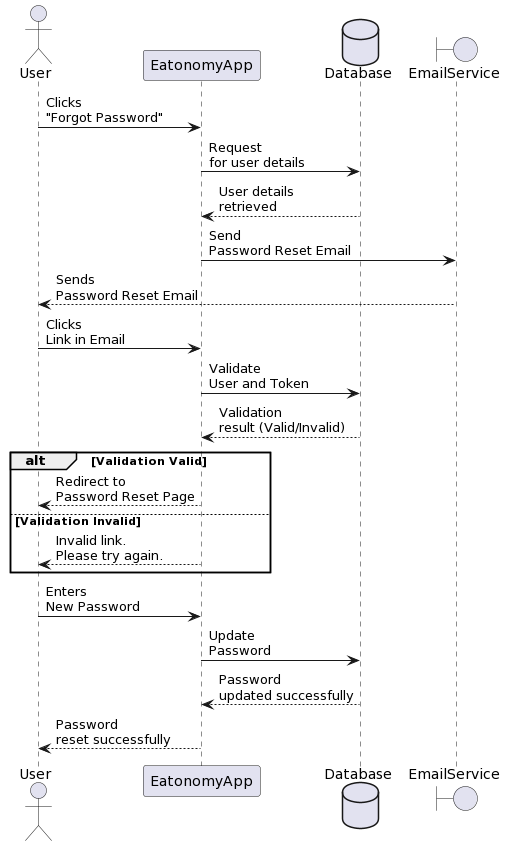
###### *Figure 40 Sequence Diagram Create Account*

##### 4.6.2 Sequence diagram for Login account



###### *Figure 41 Sequence Diagram Login Account*

##### 4.6.3 Sequence diagram for Forgot password



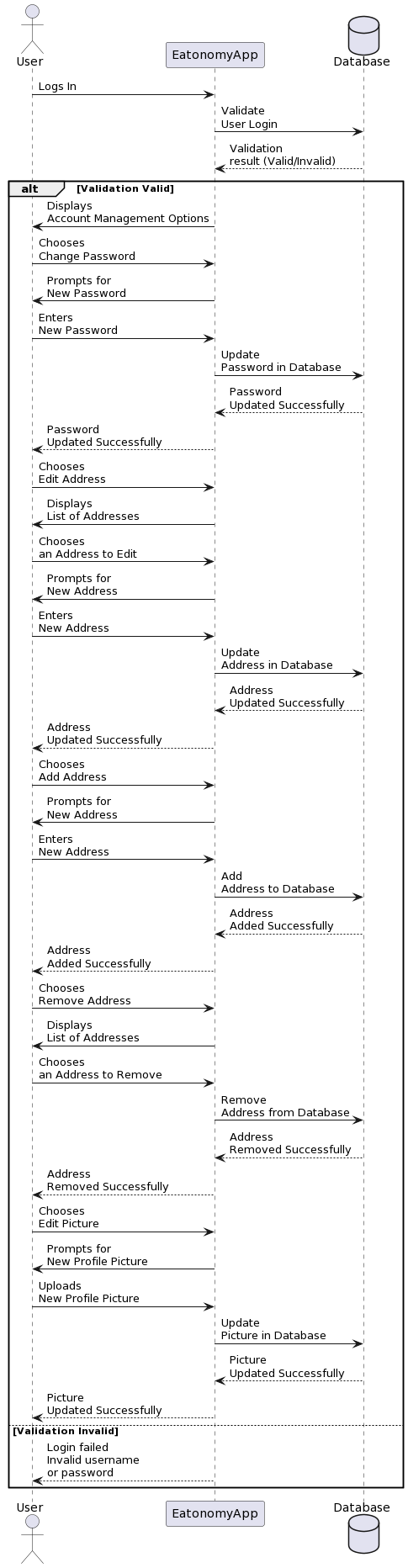
###### *Figure 42 Sequence Diagram Forgot Password*

##### 4.6.4 Sequence diagram for Food Recommender



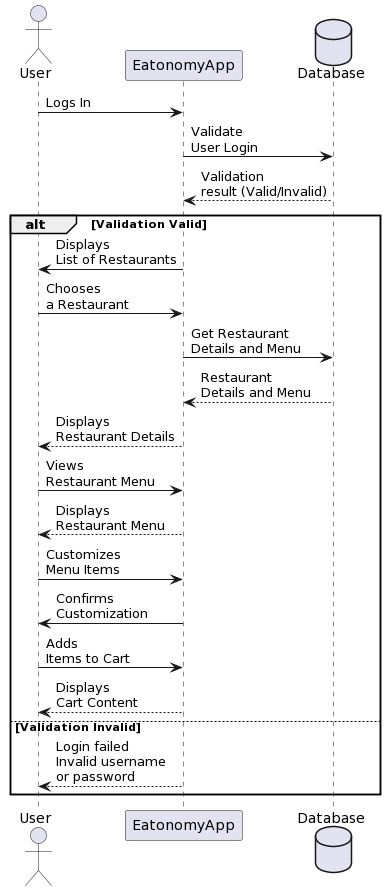
###### *Figure 43 Sequence Diagram Food Recommendation*

##### 4.6.5 Sequence diagram for manage account



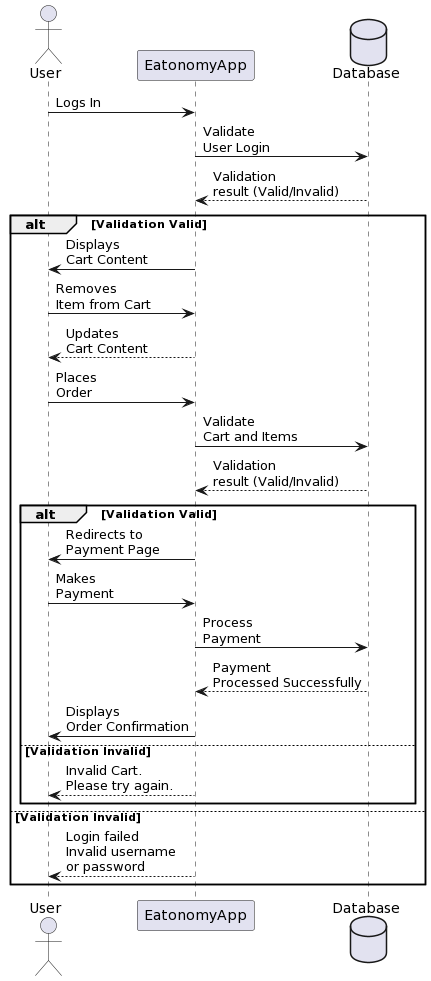
###### *Figure 44 Sequence Diagram manage account*

##### 4.6.6 Sequence diagram for View Restaurants



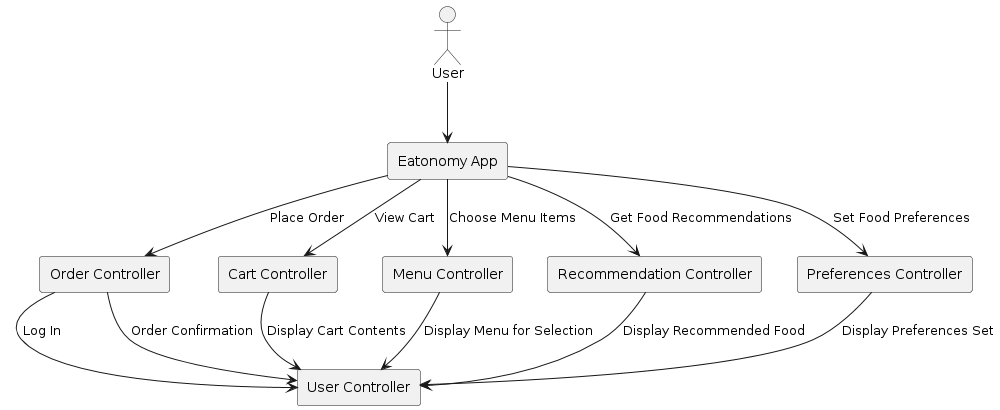
###### *Figure 45 Sequence Diagram View Restaurants*

##### 4.6.7 Sequence diagram for View Cart



###### *Figure 46 Sequence Diagram View Cart*

#### Collaboration Diagram

It shows the object organization as shown below.

###### *Figure 47 Collaboration Diagram*

#### State Transition Diagram

State Transition diagram is used to describe the states of different objects in its life cycle. So, the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately

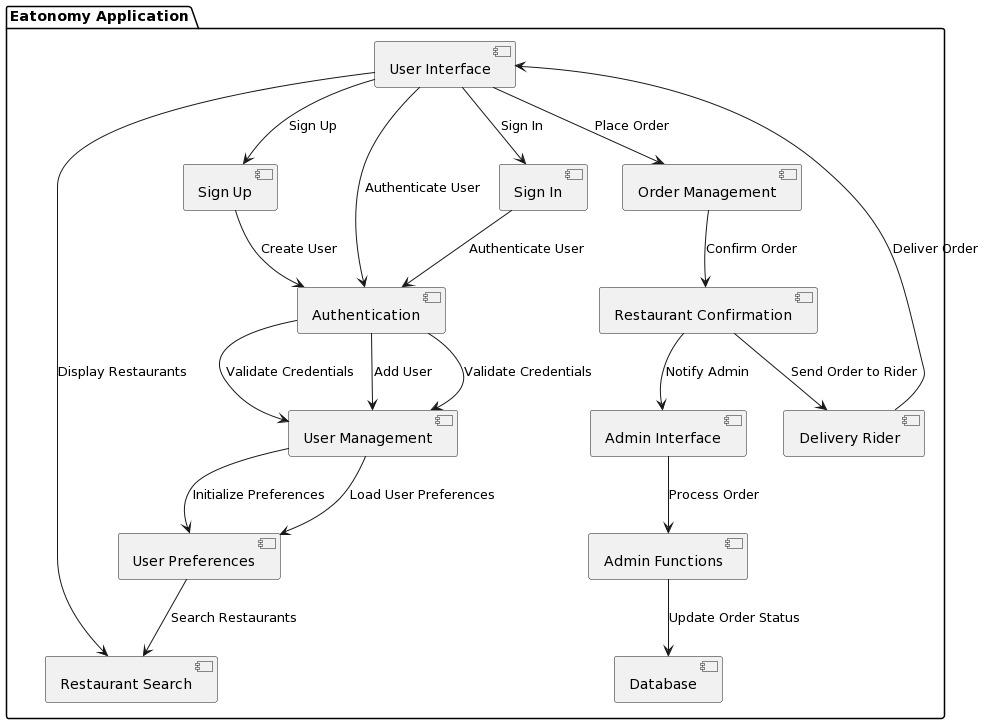


###### *Figure 48 State Transition Diagram*

#### Component Diagram

Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries etc.

So, the purpose of this diagram is different, Component diagrams are used during the implementation phase of an application. But it is prepared well in advance to visualize the implementation details.



###### *Figure 49 Component Diagram*

#### A screenshot of a computer Description automatically generatedDeployment Diagram

###### *Figure 50 Deployment Diagram*

# Chapter 5: Testing

#### Test Case Specifications

This Testing phase includes all the Test Cases of the functional requirements of your project

For example, your login account test case looks like this and you have to follow this template for writing your project test cases

#### Test Case for address

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_ADDRESS\_WIDGET\_SUCCESS** |
| **Priority** | High |
| **Description** | To verify that the address widget displays correctly and responds to taps. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Add AddressWidget to the screen.  B. Set a valid address and onTap function.  C. Tap on the AddressWidget. |
| **Input** | Valid address string and a mock onTap function. |
| **Expected result** | Address is displayed correctly and onTap function is called. |
| **Status** | Tested, passed. |

##### *Table 42: Positive Test Case – Address Widget*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_ADDRESS\_WIDGET\_FAILURE** |
| **Priority** | High |
| **Description** | To verify that the address widget handles invalid input gracefully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Add AddressWidget to the screen.  B. Set an invalid address and onTap function.  C. Tap on the AddressWidget. |
| **Input** | Invalid address (e.g., empty string) and a mock onTap function. |
| **Expected result** | Address widget should handle the invalid input gracefully without crashing. |
| **Status** | Tested, passed. |

##### *Table 43: Negative Test Case – Address Widget*

#### Test Case for forgot password

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FORGET\_PASSWORD\_SUCCESS** |
| **Priority** | High |
| **Description** | To verify that the password reset process works with valid email input. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Open Forget password.  B. Enter a valid email address.  C. Click "Reset Password". |
| **Input** | Valid email address (e.g., user@example.com). |
| **Expected result** | Password reset link is sent to the provided email address. |
| **Status** | Tested, passed. |

##### *Table 44: Positive Test Case – forgot password*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FORGET\_PASSWORD\_FAILURE** |
| **Priority** | High |
| **Description** | To verify that the password reset process handles invalid email input. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Open Forget password.  B. Enter an invalid email address.  C. Click "Reset Password". |
| **Input** | Invalid email address (e.g., invalid-email). |
| **Expected result** | Error message indicating invalid email address is displayed. |
| **Status** | Tested, passed. |

##### *Table 45: Negative Test Case – forgot password*

#### Test Case for login

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_LOGIN\_SUCCESS** |
| **Priority** | High |
| **Description** | To verify user authentication to the system. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access.  C. User must have active login credentials. |
| **Steps** | A. Open Login screen.  B. Enter valid email and password.  C. Click "Login". |
| **Input** | Valid email (e.g., user@example.com) and password (Password123). |
| **Expected result** | User is successfully logged in and navigated to the home screen. |
| **Status** | Tested, passed. |

##### *Table 46: Positive Test Case – Login*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_LOGIN\_FAILURE** |
| **Priority** | High |
| **Description** | To verify user authentication to the system. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Open Login screen.  B. Enter invalid email or password.  C. Click "Login". |
| **Input** | Invalid email (e.g., user@example.com) and password (wrongpassword). |
| **Expected result** | Error message indicating incorrect email or password is displayed. |
| **Status** | Tested, passed. |

##### *Table 47: Negative Test Case – Login*

#### Test Case for sign up

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_SIGNUP\_SUCCESS** |
| **Priority** | High |
| **Description** | To verify that the signup process works with valid details. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Open Signup screen.  B. Enter valid email, password, and confirm password.  C. Click "Sign Up". |
| **Input** | Valid email (e.g., newuser@example.com), password (Password123), and confirm password (Password123). |
| **Expected result** | User is successfully registered and navigated to the home screen. |
| **Status** | Tested, passed. |

##### *Table 48: Positive Test Case – sign up*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_SIGNUP\_FAILURE** |
| **Priority** | High |
| **Description** | To verify that the signup process handles invalid details. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Open Signup screen.  B. Enter invalid email, password, or mismatched confirm password.  C. Click "Sign Up". |
| **Input** | Invalid email (e.g., invalid email), password (Pass), or mismatched confirm password (Password456). |
| **Expected result** | Error message indicating invalid input or mismatched passwords is displayed. |
| **Status** | Tested, passed. |

##### *Table 49: Negative Test Case – sign up*

#### Test Case for cart

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_CART\_WIDGET\_SUCCESS** |
| **Priority** | High |
| **Description** | To verify that the cart widget displays items correctly and allows interaction. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access.  C. User has items in the cart. |
| **Steps** | A. Open the screen with cart.  B. View items in the cart.  C. Interact with the items (e.g., remove an item). |
| **Input** | Mock data for cart items. |
| **Expected result** | Items are displayed correctly, and interactions (e.g., removing an item) work as expected. |
| **Status** | Tested, passed. |

##### *Table 50: Positive Test Case – cart*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_CART\_WIDGET\_FAILURE** |
| **Priority** | High |
| **Description** | To verify that the cart widget handles empty cart gracefully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User has internet access. |
| **Steps** | A. Open the screen with CartWidget.  B. Ensure the cart is empty. |
| **Input** | No items in the cart. |
| **Expected result** | The cart widget displays a message indicating that the cart is empty. |
| **Status** | Tested, passed. |

##### *Table 51: Negative Test Case – cart*

#### Test Case for edit profile

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_EDIT\_PROFILE\_SUCCESS** |
| **Priority** | High |
| **Description** | To verify that the profile details can be edited successfully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Edit Profile screen.  B. Enter valid profile details.  C. Save the changes. |
| **Input** | Valid profile details (e.g., name, email). |
| **Expected result** | Profile details should be updated successfully. |
| **Status** | Tested, passed. |

##### *Table 52: Positive Test Case – edit profile*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_EDIT\_PROFILE\_FAILURE** |
| **Priority** | High |
| **Description** | To verify that the system handles invalid profile details gracefully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Edit Profile screen.  B. Enter invalid profile details (e.g., empty name).  C. Save the changes. |
| **Input** | Invalid profile details. |
| **Expected result** | Error message should be displayed indicating invalid profile details. |
| **Status** | Tested, passed. |

##### *Table 53: Negative Test Case – edit profile*

#### Test Case for favorite restaurant

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FAVORITE\_RESTAURANT\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that a restaurant can be added to the favorites list. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the restaurant details screen.  B. Tap the "Add to Favorites" button. |
| **Input** | None |
| **Expected result** | Restaurant should be added to the favorites list. |
| **Status** | Tested, passed. |

##### *Table 54: Positive Test Case – favorite restaurant*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FAVORITE\_RESTAURANT\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when adding a restaurant to the favorites list. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the restaurant details screen.  B. Simulate an error (e.g., no internet connection).  C. Tap the "Add to Favorites" button. |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to add the restaurant to the favorites list. |
| **Status** | Tested, passed. |

##### *Table 55: Negative Test Case – favorite restaurant*

#### Test Case for favorite item

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FAVORITE\_WIDGET\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the favorite items are displayed correctly. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Favorites screen.  B. View the list of favorite items. |
| **Input** | Mock data for favorite items. |
| **Expected result** | Favorite items should be displayed correctly. |
| **Status** | Tested, passed. |

##### *Table 56: Positive Test Case – favorite item*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FAVORITE\_WIDGET\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the favorite widget handles empty or null data gracefully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Favorites screen.  B. Ensure there are no favorite items. |
| **Input** | No favorite items. |
| **Expected result** | The favorites screen should display a message indicating no favorite items. |
| **Status** | Tested, passed. |

##### *Table 57: Negative Test Case – favorite item*

#### Test Case for food categories

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FOOD\_CATEGORIES\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that food categories are displayed correctly and can be selected. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Food Categories screen.  B. View the list of categories.  C. Select a category. |
| **Input** | Mock data for food categories. |
| **Expected result** | Food categories should be displayed correctly and the selected category should navigate to the corresponding screen. |
| **Status** | Tested, passed. |

##### *Table 58: Positive Test Case – food categories*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FOOD\_CATEGORIES\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when displaying food categories. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Food Categories screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load food categories. |
| **Status** | Tested, passed. |

##### *Table 59: Negative Test Case – food categories*

#### Test Case for food preferences

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FOOD\_PREFERENCES\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that food preferences can be set and saved successfully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Food Preferences screen.  B. Select food preferences.  C. Save the preferences. |
| **Input** | Valid food preference data. |
| **Expected result** | Food preferences should be saved and displayed correctly. |
| **Status** | Tested, passed. |

##### *Table 60: Positive Test Case – food preferences*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_FOOD\_PREFERENCES\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when setting food preferences. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Food Preferences screen.  B. Simulate an error (e.g., no internet connection).  C. Try to save the preferences. |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to save food preferences. |
| **Status** | Tested, passed. |

##### *Table 61: Negative Test Case – food preferences*

#### Test Case for popular items

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_HOME\_POPULAR\_ITEMS\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the popular items list displays correctly and allows interaction. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Home screen.  B. View the popular items list.  C. Select an item from the list. |
| **Input** | Mock data for popular items. |
| **Expected result** | Popular items list should display correctly and navigate to the selected item. |
| **Status** | Tested, passed. |

##### *Table 62: Positive Test Case – popular items*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_HOME\_POPULAR\_ITEMS\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when displaying the popular items list. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Home screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load popular items. |
| **Status** | Tested, passed. |

##### *Table 63: Negative Test Case – popular items*

#### Test Case for homescreen

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_HOME\_SCREEN\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the home screen loads and displays correctly. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Home screen.  B. View the content on the Home screen. |
| **Input** | Mock data for home screen content. |
| **Expected result** | Home screen should load and display content correctly. |
| **Status** | Tested, passed. |

##### *Table 64: Positive Test Case – homescreen*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_HOME\_SCREEN\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when loading the home screen. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Home screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load home screen content. |
| **Status** | Tested, passed. |

##### *Table 65: Negative Test Case – homescreen*

#### Test Case for recommended restaurant

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RECOMMENDED\_RESTAURANT\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the recommended restaurant list displays correctly and allows interaction. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the recommended restaurants screen.  B. View the list of recommended restaurants.  C. Select a restaurant from the list. |
| **Input** | Mock data for recommended restaurants. |
| **Expected result** | Recommended restaurants list should display correctly and navigate to the selected restaurant. |
| **Status** | Tested, passed. |

##### *Table 66: Positive Test Case – recommended restaurant*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RECOMMENDED\_RESTAURANT\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when displaying the recommended restaurant list. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the recommended restaurants screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load recommended restaurants. |
| **Status** | Tested, passed. |

##### *Table 67: Negative Test Case – recommended restaurant*

#### Test Case for no internet

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_NO\_INTERNET\_WIDGET\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the no internet widget displays correctly when there is no internet connection. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System has no internet connection. |
| **Steps** | A. Disable internet connection.  B. Open the app. |
| **Input** | None |
| **Expected result** | No internet widget should be displayed correctly. |
| **Status** | Tested, passed. |

##### *Table 68: Positive Test Case – no internet*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_NO\_INTERNET\_WIDGET\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the no internet widget handles errors gracefully. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System has no internet connection. |
| **Steps** | A. Corrupt the no internet widget data.  B. Open the app. |
| **Input** | Corrupted no internet widget data. |
| **Expected result** | App should handle the error gracefully without crashing. |
| **Status** | Tested, passed. |

##### *Table 69: Negative Test Case – no internet*

#### Test Case for orders

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_ORDERS\_WIDGET\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the orders widget displays orders correctly and allows interaction. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in.  C. User has placed orders. |
| **Steps** | A. Open the Orders screen.  B. View the list of orders.  C. Select an order from the list. |
| **Input** | Mock data for orders. |
| **Expected result** | Orders list should display correctly and navigate to the selected order. |
| **Status** | Tested, passed. |

##### *Table 70: Positive Test Case – orders*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_ORDERS\_WIDGET\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the orders widget handles errors gracefully when displaying orders. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Orders screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load orders. |
| **Status** | Tested, passed. |

##### *Table 71: Negative Test Case – orders*

#### Test Case for product details

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_PRODUCT\_DETAILS\_WIDGET\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the product details widget displays product information correctly. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Product Details screen.  B. View the product information. |
| **Input** | Mock data for product details. |
| **Expected result** | Product information should be displayed correctly. |
| **Status** | Tested, passed. |

##### *Table 72: Positive Test Case – product details*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_PRODUCT\_DETAILS\_WIDGET\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the product details widget handles errors gracefully when displaying product information. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Product Details screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load product details. |
| **Status** | Tested, passed. |

##### *Table 73: Negative Test Case – product details*

#### Test Case for profile

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_PROFILE\_WIDGET\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the profile widget displays user profile information correctly. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Profile screen.  B. View the profile information. |
| **Input** | Mock data for profile information. |
| **Expected result** | Profile information should be displayed correctly. |
| **Status** | Tested, passed. |

##### *Table 74: Positive Test Case – profile*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_PROFILE\_WIDGET\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the profile widget handles errors gracefully when displaying user profile information. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Profile screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load profile information. |
| **Status** | Tested, passed. |

##### *Table 75: Negative Test Case – profile*

#### Test Case for restaurant menu

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RESTAURANT\_MENU\_WIDGET\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the restaurant menu widget displays the restaurant's menu correctly. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Restaurant Menu screen.  B. View the restaurant's menu. |
| **Input** | Mock data for restaurant menu. |
| **Expected result** | Restaurant menu should be displayed correctly. |
| **Status** | Tested, passed. |

##### *Table 76: Positive Test Case – restaurant menu*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RESTAURANT\_MENU\_WIDGET\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the restaurant menu widget handles errors gracefully when displaying the menu. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Restaurant Menu screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load the restaurant menu. |
| **Status** | Tested, passed. |

##### *Table 77: Negative Test Case – restaurant menu*

#### Test Case for restaurants

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RESTAURANTS\_ALL\_VIEW\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that all restaurants are displayed correctly and allow interaction. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the All Restaurants screen.  B. View the list of restaurants.  C. Select a restaurant from the list. |
| **Input** | Mock data for restaurants. |
| **Expected result** | Restaurants list should display correctly and navigate to the selected restaurant. |
| **Status** | Tested, passed. |

##### *Table 78: Positive Test Case – restaurants*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RESTAURANTS\_ALL\_VIEW\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the system handles errors gracefully when displaying all restaurants. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the All Restaurants screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load restaurants. |
| **Status** | Tested, passed. |

##### *Table 79: Negative Test Case – restaurants*

#### Test Case for recommended restaurant

**Positive Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RECOMMENDED\_RESTAURANT\_LIST\_SUCCESS** |
| **Priority** | Medium |
| **Description** | To verify that the recommended restaurant list displays correctly and allows interaction. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Recommended Restaurants screen.  B. View the list of recommended restaurants.  C. Select a restaurant from the list. |
| **Input** | Mock data for recommended restaurants. |
| **Expected result** | Recommended restaurants list should display correctly and navigate to the selected restaurant. |
| **Status** | Tested, passed. |

##### *Table 80: Positive Test Case – recommended restaurants*

**Negative Test Case**

|  |  |
| --- | --- |
| **ID** | **TC\_RECOMMENDED\_RESTAURANT\_LIST\_FAILURE** |
| **Priority** | Medium |
| **Description** | To verify that the recommended restaurant list handles errors gracefully when displaying recommended restaurants. |
| **Reference** | Functional Requirement reference |
| **Users** | Customer |
| **Pre-requisites** | A. System is online.  B. User is logged in. |
| **Steps** | A. Open the Recommended Restaurants screen.  B. Simulate an error (e.g., no internet connection). |
| **Input** | None |
| **Expected result** | Error message should be displayed indicating the failure to load recommended restaurants. |
| **Status** | Tested, passed. |

##### *Table 81: Negative Test Case –* *recommended restaurants*

#### Black Box Test Cases

Black box testing also known as Behavioral Testing, is a [software testing method](http://softwaretestingfundamentals.com/software-testing-methods/) in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

* + 1. **Equivalence Partitions (EP)**

Equivalence class partitioning (EP) is a very widely used method to decrease the number of possible test cases that are required to test a system.

**Equivalence Partitions for Login**

**Username**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Username | 1. Valid email format (e.g., user@example.com). | 1. Incorrect email format (e.g., user@com  2. Empty Field. |

##### *Table 82: Valid Classes and Invalid Classes – Login username*

**Password**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Password | 1. Length greater than 5 characters.  2. Contains symbols, alphabets [a-z A-Z], and digits [0-9]. | 1. Length less than 5 characters.  2. Empty Field. |

##### *Table 83: Valid Classes and Invalid Classes – Login password*

**Equivalence Partitions for Sign Up**

**Email**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Email | 1. Valid email format (e.g., newuser@example.com). | 1. Incorrect email format (e.g., user@com,  2. Empty Field. |

##### *Table 84: Valid Classes and Invalid Classes - Sign Up Email*

**Password**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Password | 1. Length greater than 5 characters.  2. Contains symbols, alphabets [a-z A-Z], and digits [0-9]. | 1. Length less than 5 characters.  2. Empty Field. |

##### *Table 85: Valid Classes and Invalid Classes - Sign Up password*

**Confirm Password**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Confirm Password | 1. Matches Password. | 1. Does not match Password  2. Empty Field. |

##### *Table 86: Valid Classes and Invalid Classes - Sign Up confirm password*

**Equivalence Partitions for Forgot Password**

**Email**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Email | 1. Valid email format (e.g., user@example.com). | 1. Incorrect email format (e.g., user@com  2. Empty Field. |

##### *Table 87: Valid Classes and Invalid Classes - Forgot Password*

**Equivalence Partitions for Address**

**Address**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Address | 1. Valid address format (e.g., "123 Main St, City"). | 1. Empty Field. |

##### *Table 88: Valid Classes and Invalid Classes - Address*

**Equivalence Partitions for Cart**

**Cart Items**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Cart Items | 1. One or more items in the cart. | 1. Empty Cart. |

##### *Table 89: Valid Classes and Invalid Classes - Cart*

**Equivalence Partitions for Profile**

**Profile Details**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Profile Name | 1. Non-empty string. | 1. Empty Field. |
| Profile Email | 1. Valid email format (e.g., user@example.com). | 1. Incorrect email format (e.g., user@com  2. Empty Field. |

##### *Table 90: Valid Classes and Invalid Classes - Profile*

**Equivalence Partitions for Restaurant Menu**

**Menu Items**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Menu Items | 1. One or more items in the menu. | 1. Empty Menu. |

##### *Table 91: Valid Classes and Invalid Classes - Restaurant Menu*

**Equivalence Partitions for Orders**

**Orders**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Orders | 1. One or more orders placed. | 1. No orders placed. |

##### *Table 92: Valid Classes and Invalid Classes - Orders*

**Equivalence Partitions for Food Preferences**

**Food Preferences**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Food Preferences | 1. One or more food preferences selected. | 1. No preferences selected. |

##### *Table 93: Valid Classes and Invalid Classes - Food Preferences*

**Equivalence Partitions for Favorite Restaurant**

**Favorite Restaurant**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Favorite Restaurant | 1. One or more restaurants favorited. | 1. No restaurants favorited. |

##### *Table 94: Valid Classes and Invalid Classes - Favorite Restaurant*

**Equivalence Partitions for Popular Items**

**Popular Items**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Valid Classes** | **Invalid Classes** |
| Popular Items | 1. One or more popular items displayed. | 1. No popular items displayed. |

##### *Table 95: Valid Classes and Invalid Classes - Popular Items*

* + 1. **Boundary Value Analysis**

A boundary value is an input or output value on the border of an equivalence partition, includes minimum and maximum values at inside and outside boundaries. Normally Boundary value analysis is part of stress and negative testing.

**Boundary Value Analysis for Login**

**Username**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Valid email (e.g., user@example.com). |
| Outside | Invalid email (e.g., user@com, empty string). |

##### *Table 96: Boundary Value Analysis – Login username*

**Password**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Length of 6 characters.  Valid password format (e.g., Pass123). |
| Outside | Length of 5 characters or less.  Invalid password format (e.g., empty string). |

##### *Table 97: Boundary Value Analysis – Login password*

**Boundary Value Analysis for Sign Up**

**Email**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Valid email (e.g., newuser@example.com). |
| Outside | Invalid email (e.g., user@com, empty string). |

##### *Table 98: Boundary Value Analysis - Sign Up Email*

**Password**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Length of 6 characters.  Valid password format (e.g., Pass123). |
| Outside | Length of 5 characters or less.  Invalid password format (e.g., empty string). |

##### *Table 99: Boundary Value Analysis - Sign Up Password*

**Confirm Password**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Matches password (e.g., Pass123). |
| Outside | Does not match password (e.g., Pass456).  Empty string. |

##### *Table 100: Boundary Value Analysis - Sign Up Confirm Password*

**Boundary Value Analysis for Forgot Password**

**Email**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Valid email (e.g., user@example.com). |
| Outside | Invalid email (e.g., user@com, empty string). |

##### *Table 101: Boundary Value Analysis - Forgot Password*

**Boundary Value Analysis for Address**

**Address**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Valid address (e.g., "123 Main St, City"). |
| Outside | Empty string. |

##### *Table 102: Boundary Value Analysis - Address*

**Boundary Value Analysis for Cart**

**Cart Items**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | One or more items in the cart. |
| Outside | Empty cart. |

##### *Table 103: Boundary Value Analysis - Cart*

**Boundary Value Analysis for Profile**

**Profile Name**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Non-empty string. |
| Outside | Empty string. |

##### *Table 104: Boundary Value Analysis – Profile Name*

**Profile Email**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | Valid email (e.g., user@example.com). |
| Outside | Invalid email (e.g., user@com, empty string). |

##### *Table 105: Boundary Value Analysis – Profile Email*

**Boundary Value Analysis for Restaurant Menu**

**Menu Items**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | One or more items in the menu. |
| Outside | Empty menu. |

##### *Table 106: Boundary Value Analysis - Restaurant Menu*

**Boundary Value Analysis for Orders**

**Orders**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | One or more orders placed. |
| Outside | No orders placed. |

##### *Table 107: Boundary Value Analysis - Orders*

**Boundary Value Analysis for Food Preferences**

**Food Preferences**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | One or more food preferences selected. |
| Outside | No preferences selected. |

##### *Table 108: Boundary Value Analysis - Food Preferences*

**Boundary Value Analysis for Favorite Restaurant**

**Favorite Restaurant**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | One or more restaurants favorited. |
| Outside | No restaurants favorited. |

##### *Table 109: Boundary Value Analysis - Favorite Restaurant*

**Boundary Value Analysis for Popular Items**

**Popular Items**

|  |  |
| --- | --- |
| **Boundary Type** | **Boundary Value** |
| Inside | One or more popular items displayed. |
| Outside | No popular items displayed. |

##### *Table 110: Boundary Value Analysis - Popular Items*

* + 1. **Decision Table Testing**

Decision Table is a testing method, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.

**Decision Table for Login**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
| Valid email format | Y | Y | N | N |
| Valid password length | Y | N | Y | N |
| Action |  |  |  |  |
| Display success message | X |  |  |  |
| Display error for password |  | X |  |  |
| Display error for email |  |  | X |  |
| Display error for both |  |  |  | X |

##### *Table 111: Decision Table - Login*

**Decision Table for Sign Up**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** | **Rule 5** | **Rule 6** |
| Valid email format | Y | Y | N | N | Y | Y |
| Password length > 5 characters | Y | N | Y | N | Y | Y |
| Confirm password matches password | Y | Y | Y | Y | N | N |
| Action |  |  |  |  |  |  |
| Display success message | X |  |  |  |  |  |
| Display error for password |  | X |  |  |  |  |
| Display error for email |  |  | X |  |  |  |
| Display error for confirm password |  |  |  |  | X |  |
| Display error for both email & password |  |  |  | X |  |  |

##### *Table 112: Decision Table - Sign Up*

**Decision Table for Forgot Password**

|  |  |  |  |
| --- | --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** |
| Valid email format | Y | N | N |
| Email exists in the system | Y | Y | N |
| Action |  |  |  |
| Send reset link | X |  |  |
| Display error for email |  | X |  |
| Display error for non-exist email |  |  |  |

##### *Table 113: Decision Table - Forgot Password*

**Decision Table for Address**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Valid address format | Y | N |
| Action |  |  |
| Display address | X |  |
| Display error for address |  | X |

##### *Table 114: Decision Table - Address*

**Decision Table for Cart**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Cart contains items | Y | N |
| Action |  |  |
| Display cart items | X |  |
| Display empty cart message |  | X |

##### *Table 115: Decision Table - Cart*

**Decision Table for Profile**

|  |  |  |  |
| --- | --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** |
| Valid name format | Y | N | Y |
| Valid email format | Y | Y | N |
| Action |  |  |  |
| Display profile info | X |  |  |
| Display error for name |  | X |  |
| Display error for email |  |  | X |

##### *Table 116: Decision Table - Profile*

**Decision Table for Restaurant Menu**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Menu contains items | Y | N |
| Action |  |  |
| Display menu items | X |  |
| Display empty menu message |  | X |

##### *Table 117: Decision Table - Restaurant Menu*

**Decision Table for Orders**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Orders exist | Y | N |
| Action |  |  |
| Display orders | X |  |
| Display no orders message |  | X |

##### *Table 118: Decision Table - Orders*

**Decision Table for Food Preferences**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Preferences selected | Y | N |
| Action |  |  |
| Save preferences | X |  |
| Display error for no preferences |  |  |

##### *Table 119: Decision Table - Food Preferences*

**Decision Table for Favorite Restaurant**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Restaurant favorited | Y | N |
| Action |  |  |
| Add to favorites | X |  |
| Display error for no favorite |  |  |

##### *Table 120: Decision Table - Favorite Restaurant*

**Decision Table for Popular Items**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Popular items available | Y | N |
| Action |  |  |
| Display popular items | X |  |
| Display no popular items message |  |  |

##### *Table 121: Decision Table - Popular Items*

* + 1. **State transition Testing**

State Transition testing, a black box testing technique, in which outputs are triggered by changes to the input conditions or changes to 'state' of the system. In other words, tests are designed to execute valid and invalid state transitions.

**State Transition Table for Login**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| Logged Out | Enter valid credentials | Logged In | Success message |
| Logged Out | Enter invalid credentials | Logged Out | Error message |
| Logged Out | Click "Forgot Password" | Password Reset | Navigate to reset screen |
| Logged In | Click "Logout" | Logged Out | Navigate to login screen |

##### *Table 122: State Transition Table - Login*

**State Transition Table for Sign Up**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| No Account | Enter valid details | Account Created | Success message |
| No Account | Enter invalid details | No Account | Error message |
| No Account | Click "Already have an account?" | Logged Out | Navigate to login screen |
| Account Created | Click "Logout" | Logged Out | Navigate to login screen |

##### *Table 123: State Transition Table - Sign Up*

**State Transition Table for Forgot Password**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| Request Reset | Enter valid email | Reset Initiated | Reset link sent message |
| Request Reset | Enter invalid email | Request Reset | Error message |
| Request Reset | Click "Back to Login" | Logged Out | Navigate to login screen |
| Reset Initiated | Click "Back to Login" | Logged Out | Navigate to login screen |

##### *Table 124: State Transition Table - Forgot Password*

**State Transition Table for Address**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| Address Empty | Enter valid address | Address Set | Display address |
| Address Empty | Enter invalid address | Address Empty | Error message |
| Address Set | Click "Edit Address" | Address Edit | Navigate to edit screen |
| Address Edit | Enter valid address | Address Set | Display updated address |

##### *Table 125: State Transition Table - Address*

**State Transition Table for Cart**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| Cart Empty | Add items to cart | Cart Populated | Display items in cart |
| Cart Populated | Remove all items | Cart Empty | Display empty cart message |
| Cart Populated | Proceed to checkout | Checkout | Navigate to checkout page |
| Checkout | Complete payment | Order Placed | Order confirmation |
| Checkout | Cancel checkout | Cart Populated | Return to cart |

##### *Table 126: State Transition Table - Cart*

**State Transition Table for Profile**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| Profile View | Click "Edit Profile" | Edit Profile | Navigate to edit screen |
| Edit Profile | Enter valid details | Profile Updated | Display updated profile |
| Edit Profile | Enter invalid details | Edit Profile | Error message |
| Profile Updated | Click "Logout" | Logged Out | Navigate to login screen |

##### *Table 127: State Transition Table - Profile*

**State Transition Table for Restaurant Menu**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| Menu Empty | Add items to menu | Menu Populated | Display items in menu |
| Menu Populated | Remove all items | Menu Empty | Display empty menu message |
| Menu Populated | Click "Edit Menu" | Edit Menu | Navigate to edit screen |
| Edit Menu | Enter valid details | Menu Updated | Display updated menu |

##### *Table 128: State Transition Table - Restaurant Menu*

**State Transition Table for Orders**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| No Orders | Place new order | Orders Placed | Display orders |
| Orders Placed | Click "View Order" | Viewing Order | Display order details |
| Viewing Order | Cancel order | Order Cancelled | Order cancelled message |
| Viewing Order | Confirm order | Order Confirmed | Order confirmed message |

##### *Table 129: State Transition Table - Orders*

**State Transition Table for Food Preferences**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| No Preferences | Select preferences | Preferences Set | Display selected preferences |
| Preferences Set | Click "Edit Preferences" | Edit Preferences | Navigate to edit screen |
| Edit Preferences | Save changes | Preferences Updated | Display updated preferences |

##### *Table 130: State Transition Table - Food Preferences*

**State Transition Table for Favorite Restaurant**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| No Favorites | Add restaurant to favorites | Favorites Set | Display favorited restaurant |
| Favorites Set | Remove restaurant from favorites | No Favorites | Display no favorites message |
| Favorites Set | View favorite restaurants | Viewing Favorites | Display list of favorites |

##### *Table 131: State Transition Table - Favorite Restaurant*

**State Transition Table for Popular Items**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Input** | **Next State** | **Output** |
| No Items | Add popular items | Items Available | Display popular items |
| Items Available | Remove all items | No Items | Display no items message |
| Items Available | Select item | Item Selected | Display item details |

##### *Table 132: State Transition Table - Popular Items*

* + 1. **Use Case Testing**

Use Case Testing is a functional black box testing technique that helps testers to identify test scenarios that exercise the whole system on each transaction basis from start to finish.

**Use Case Testing for Login**

**Use Case: Login**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_LOGIN\_01 | Valid login | User has valid credentials | 1. Open login screen  2. Enter valid email and password  3. Click "Login" | User is logged in and navigated to home screen |
| UC\_LOGIN\_02 | Invalid login - incorrect password | User has valid email and invalid password | 1. Open login screen  2. Enter valid email and invalid password  3. Click "Login" | Error message displayed indicating incorrect password |
| UC\_LOGIN\_03 | Invalid login - incorrect email | User has invalid email and valid password | 1. Open login screen  2. Enter invalid email and valid password  3. Click "Login" | Error message displayed indicating incorrect email |
| UC\_LOGIN\_04 | Forgot password navigation | None | 1. Open login screen  2. Click "Forgot Password" | Navigated to password reset screen |

##### *Table 133: Use Case Testing - Login*

**Use Case Testing for Sign Up**

**Use Case: Sign Up**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_SIGNUP\_01 | Valid sign up | None | 1. Open sign up screen  2. Enter valid email, password, and confirm password  3. Click "Sign Up" | User is registered and navigated to home screen |
| UC\_SIGNUP\_02 | Invalid sign up - mismatched passwords | None | 1. Open sign up screen  2. Enter valid email, password, and different confirm password  3. Click "Sign Up" | Error message displayed indicating password mismatch |
| UC\_SIGNUP\_03 | Invalid sign up - invalid email | None | 1. Open sign up screen  2. Enter invalid email, password, and confirm password  3. Click "Sign Up" | Error message displayed indicating invalid email |
| UC\_SIGNUP\_04 | Navigation to login screen | None | 1. Open sign up screen  2. Click "Already have an account?" | Navigated to login screen |

##### *Table 134: Use Case Testing - Sign Up*

**Use Case Testing for Forgot Password**

**Use Case: Forgot Password**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_FORGOT\_PASSWORD\_01 | Valid password reset | None | 1. Open forgot password screen  2. Enter valid email  3. Click "Reset Password" | Password reset link sent to email |
| UC\_FORGOT\_PASSWORD\_02 | Invalid password reset - invalid email | None | 1. Open forgot password screen  2. Enter invalid email  3. Click "Reset Password" | Error message displayed indicating invalid email |
| UC\_FORGOT\_PASSWORD\_03 | Navigation to login screen | None | 1. Open forgot password screen  2. Click "Back to Login" | Navigated to login screen |

##### *Table 135: Use Case Testing - Forgot Password*

**Use Case Testing for Address**

**Use Case: Manage Address**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_ADDRESS\_01 | Add valid address | User is logged in | 1. Open manage address screen  2. Enter valid address  3. Click "Save" | Address saved and displayed |
| UC\_ADDRESS\_02 | Add invalid address | User is logged in | 1. Open manage address screen  2. Enter invalid address  3. Click "Save" | Error message displayed indicating invalid address |
| UC\_ADDRESS\_03 | Edit address | User is logged in and address is saved | 1. Open manage address screen  2. Edit address  3. Click "Save" | Address updated and displayed |

##### *Table 136: Use Case Testing - Manage Address*

**Use Case Testing for Cart**

**Use Case: Manage Cart**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_CART\_01 | Add items to cart | User is logged in | 1. Open cart screen  2. Add items to cart  3. View cart | Items displayed in cart |
| UC\_CART\_02 | Remove items from cart | User is logged in and items in cart | 1. Open cart screen  2. Remove items from cart  3. View cart | Items removed and cart updated |
| UC\_CART\_03 | Proceed to checkout | User is logged in and items in cart | 1. Open cart screen  2. Click "Proceed to Checkout" | Navigated to checkout screen |

##### *Table 137: Use Case Testing - Manage Cart*

**Use Case Testing for Profile**

**Use Case: Manage Profile**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_PROFILE\_01 | Edit valid profile details | User is logged in | 1. Open profile screen  2. Edit profile details  3. Click "Save" | Profile updated and displayed |
| UC\_PROFILE\_02 | Edit invalid profile details | User is logged in | 1. Open profile screen  2. Edit profile details with invalid data  3. Click "Save" | Error message displayed indicating invalid details |

##### *Table 138: Use Case Testing - Manage Profile*

**Use Case Testing for Restaurant Menu**

**Use Case: View Restaurant Menu**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_MENU\_01 | View menu with items | User is logged in | 1. Open restaurant menu screen  2. View menu | Menu items displayed |
| UC\_MENU\_02 | View empty menu | User is logged in | 1. Open restaurant menu screen  2. View menu | Empty menu message displayed |

##### *Table 139: Use Case Testing - View Restaurant Menu*

**Use Case Testing for Orders**

**Use Case: Manage Orders**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_ORDERS\_01 | View orders | User has placed orders | 1. Open orders screen  2. View orders | Orders displayed |
| UC\_ORDERS\_02 | Cancel order | User has placed orders | 1. Open orders screen  2. Select order  3. Click "Cancel" | Order cancelled message displayed |
| UC\_ORDERS\_03 | Confirm order | User has placed orders | 1. Open orders screen  2. Select order  3. Click "Confirm" | Order confirmed message displayed |

##### *Table 140: Use Case Testing - Manage Orders*

**Use Case Testing for Food Preferences**

**Use Case: Manage Food Preferences**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_PREFERENCES\_01 | Set valid food preferences | User is logged in | 1. Open food preferences screen  2. Select preferences  3. Click "Save" | Preferences saved and displayed |
| UC\_PREFERENCES\_02 | Set invalid food preferences | User is logged in | 1. Open food preferences screen  2. Select invalid preferences  3. Click "Save" | Error message displayed indicating invalid preferences |

##### *Table 141: Use Case Testing - Manage Food Preferences*

**Use Case Testing for Favorite Restaurant**

**Use Case: Manage Favorite Restaurant**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_FAVORITE\_RESTAURANT\_01 | Add to favorites | User is logged in | 1. Open restaurant screen  2. Click "Add to Favorites" | Restaurant added to favorites |
| UC\_FAVORITE\_RESTAURANT\_02 | Remove from favorites | User is logged in | 1. Open favorites screen  2. Select restaurant  3. Click "Remove from Favorites" | Restaurant removed from favorites |
| UC\_FAVORITE\_RESTAURANT\_03 | View favorites | User is logged in and has favorites | 1. Open favorites screen | List of favorite restaurants displayed |

##### *Table 142: Use Case Testing - Manage Favorite Restaurant*

**Use Case Testing for Popular Items**

**Use Case: View Popular Items**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| UC\_POPULAR\_ITEMS\_01 | View popular items | User is logged in | 1. Open popular items screen  2. View items | Popular items displayed |
| UC\_POPULAR\_ITEMS\_02 | View no popular items | User is logged in | 1. Open popular items screen  2. View items | No popular items message displayed |

##### *Table 143: Use Case Testing - View Popular Items*

#### White Box Test Cases

White box testing is a testing technique, that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

* + 1. **Cyclometric complexity**

Cyclometric complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Lower the Program's cyclometric complexity, lower the risk to modify and easier to understand.

**Cyclomatic Complexity for Login Function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Nodes (N)** | **Edges (E)** | **Connected Components (P)** | **Cyclomatic Complexity (CC)** |
| login | 4 | 5 | 1 | 3 |

##### *Table 144: Cyclomatic Complexity - Login Function*

**Cyclomatic Complexity for Sign Up Function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Nodes (N)** | **Edges (E)** | **Connected Components (P)** | **Cyclomatic Complexity (CC)** |
| signUp | 4 | 5 | 1 | 3 |

##### *Table 145: Cyclomatic Complexity - Sign Up Function*

**Cyclomatic Complexity for Cart Management Function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Nodes (N)** | **Edges (E)** | **Connected Components (P)** | **Cyclomatic Complexity (CC)** |
| manageCart | 4 | 5 | 1 | 3 |

##### *Table 146: Cyclomatic Complexity - Cart Management Function*

**Cyclomatic Complexity for Profile Update Function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Nodes (N)** | **Edges (E)** | **Connected Components (P)** | **Cyclomatic Complexity (CC)** |
| updateProfile | 3 | 4 | 1 | 3 |

##### *Table 147: Cyclomatic Complexity - Profile Update Function*

**Cyclomatic Complexity for Viewing Popular Items Function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Nodes (N)** | **Edges (E)** | **Connected Components (P)** | **Cyclomatic Complexity (CC)** |
| viewPopularItems | 5 | 6 | 1 | 3 |

##### *Table 148: Cyclomatic Complexity - Viewing Popular Items Function*

#### Performance testing

Performance testing, a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage.

Below are performance test cases for various components of the customer application.

**Test Case for Login Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_LOGIN\_PERF\_01** |
| Priority | High |
| Description | To verify that the login process is completed within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access.  C. User has valid credentials. |
| Steps | A. Open the login screen.  B. Enter valid email and password.  C. Click "Login". |
| Input | Valid email (e.g., user@example.com) and password (Password123). |
| Expected result | The login process should be completed within 2 seconds. |
| Status | Tested, passed. |

##### *Table 149: Performance Test Case – Login*

**Test Case for Sign Up Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SIGNUP\_PERF\_01** |
| Priority | High |
| Description | To verify that the sign-up process is completed within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access. |
| Steps | A. Open the sign-up screen.  B. Enter valid email, password, and confirm password.  C. Click "Sign Up". |
| Input | Valid email (e.g., newuser@example.com), password (Password123), and confirm password (Password123). |
| Expected result | The sign-up process should be completed within 3 seconds. |
| Status | Tested, passed. |

##### *Table 150: Performance Test Case – Sign Up*

**Test Case for Viewing Home Screen Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_HOME\_SCREEN\_PERF\_01** |
| Priority | High |
| Description | To verify that the home screen loads within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the home screen.  B. Measure the time taken to load all components of the home screen. |
| Input | Mock data for home screen content. |
| Expected result | The home screen should load within 3 seconds. |
| Status | Tested, passed. |

##### *Table 151: Performance Test Case – Home Screen*

**Test Case for Adding Item to Cart Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_CART\_ADD\_PERF\_01** |
| Priority | High |
| Description | To verify that adding an item to the cart is completed within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items to add to the cart. |
| Steps | A. Open the screen with the item.  B. Click "Add to Cart".  C. Measure the time taken for the item to appear in the cart. |
| Input | Mock data for item details. |
| Expected result | The item should be added to the cart within 1 second. |
| Status | Tested, passed. |

##### *Table 152: Performance Test Case – Adding Item to Cart*

**Test Case for Viewing Cart Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_CART\_VIEW\_PERF\_01** |
| Priority | Medium |
| Description | To verify that the cart screen loads within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items in the cart. |
| Steps | A. Open the cart screen.  B. Measure the time taken to load all items in the cart. |
| Input | Mock data for cart items. |
| Expected result | The cart screen should load within 2 seconds. |
| Status | Tested, passed. |

##### *Table 153: Performance Test Case – Viewing Cart*

**Test Case for Profile Update Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_PROFILE\_UPDATE\_PERF\_01** |
| Priority | Medium |
| Description | To verify that updating profile details is completed within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Edit Profile screen.  B. Enter valid profile details.  C. Click "Save".  D. Measure the time taken for the profile to be updated. |
| Input | Valid profile details (e.g., name, email). |
| Expected result | The profile update process should be completed within 2 seconds. |
| Status | Tested, passed. |

##### *Table 154: Performance Test Case – Profile Update*

**Test Case for Viewing Orders Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ORDERS\_VIEW\_PERF\_01** |
| Priority | Medium |
| Description | To verify that the orders screen loads within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has placed orders. |
| Steps | A. Open the Orders screen.  B. Measure the time taken to load all orders. |
| Input | Mock data for orders. |
| Expected result | The orders screen should load within 3 seconds. |
| Status | Tested, passed. |

##### *Table 155: Performance Test Case – Viewing Orders*

**Test Case for Viewing Restaurant Menu Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_RESTAURANT\_MENU\_PERF\_01** |
| Priority | Medium |
| Description | To verify that the restaurant menu screen loads within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Restaurant Menu screen.  B. Measure the time taken to load the menu. |
| Input | Mock data for restaurant menu. |
| Expected result | The restaurant menu should load within 2 seconds. |
| Status | Tested, passed. |

##### *Table 156: Performance Test Case – Viewing Restaurant Menu*

**Test Case for Viewing Popular Items Performance**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_POPULAR\_ITEMS\_VIEW\_PERF\_01** |
| Priority | Medium |
| Description | To verify that the popular items list loads within an acceptable time frame. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the popular items screen.  B. Measure the time taken to load all popular items. |
| Input | Mock data for popular items. |
| Expected result | The popular items screen should load within 2 seconds. |
| Status | Tested, passed. |

##### *Table 157: Performance Test Case – Viewing Popular Items*

#### Stress Testing

Stress testing a Non-Functional testing technique that is performed as part of performance testing. During stress testing, the system is monitored after subjecting the system to overload to ensure that the system can sustain the stress.

The recovery of the system from such phase (after stress) is very critical as it is highly likely to happen in production environment.

Here are stress test cases for various components of the customer application:

**Test Case for Login Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_LOGIN\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users attempt to login simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User has internet access.  C. User has valid credentials. |
| Steps | A. Simulate 30 users attempting to login simultaneously. |
| Input | Valid email (e.g., user@example.com) and password (Password123). |
| Expected result | The system should handle all login attempts without crashing and should respond within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 158: Stress Test Case – Login Under Heavy Load*

**Test Case for Sign Up Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SIGNUP\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users attempt to sign up simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User has internet access. |
| Steps | A. Simulate 30 users attempting to sign up simultaneously. |
| Input | Valid email (e.g., newuser@example.com), password (Password123), and confirm password (Password123). |
| Expected result | The system should handle all sign up attempts without crashing and should respond within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 159: Stress Test Case – Sign Up Under Heavy Load*

**Test Case for Viewing Home Screen Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_HOME\_SCREEN\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users access the home screen simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Simulate 30 users accessing the home screen simultaneously. |
| Input | Mock data for home screen content. |
| Expected result | The home screen should load for all users without crashing and within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 160: Stress Test Case – Viewing Home Screen Under Heavy Load*

**Test Case for Adding Items to Cart Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_CART\_ADD\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users add items to their cart simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items to add to the cart. |
| Steps | A. Simulate 30 users adding items to their cart simultaneously. |
| Input | Mock data for item details. |
| Expected result | The system should handle all add to cart operations without crashing and should respond within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 161: Stress Test Case – Adding Items to Cart Under Heavy Load*

**Test Case for Viewing Cart Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_CART\_VIEW\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users view their cart simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items in the cart. |
| Steps | A. Simulate 30 users viewing their cart simultaneously. |
| Input | Mock data for cart items. |
| Expected result | The cart screen should load for all users without crashing and within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 162: Stress Test Case – Viewing Cart Under Heavy Load*

**Test Case for Profile Update Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_PROFILE\_UPDATE\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users update their profiles simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Simulate 30 users updating their profiles simultaneously. |
| Input | Valid profile details (e.g., name, email). |
| Expected result | The system should handle all profile updates without crashing and should respond within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 163: Stress Test Case – Profile Update Under Heavy Load*

**Test Case for Viewing Orders Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ORDERS\_VIEW\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users view their orders simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has placed orders. |
| Steps | A. Simulate 30 users viewing their orders simultaneously. |
| Input | Mock data for orders. |
| Expected result | The orders screen should load for all users without crashing and within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 164: Stress Test Case – Viewing Orders Under Heavy Load*

**Test Case for Viewing Restaurant Menu Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_RESTAURANT\_MENU\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users view the restaurant menu simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Simulate 30 users viewing the restaurant menu simultaneously. |
| Input | Mock data for restaurant menu. |
| Expected result | The restaurant menu should load for all users without crashing and within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 165: Stress Test Case – Viewing Restaurant Menu Under Heavy Load*

**Test Case for Viewing Popular Items Under Heavy Load**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_POPULAR\_ITEMS\_VIEW\_STRESS\_01** |
| Priority | High |
| Description | To verify the application's performance when multiple users view popular items simultaneously. |
| Reference | Functional Requirement reference |
| Users | Multiple customers |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Simulate 30 users viewing popular items simultaneously. |
| Input | Mock data for popular items. |
| Expected result | The popular items screen should load for all users without crashing and within an acceptable time frame. |
| Status | Tested, passed. |

##### *Table 166: Stress Test Case – Viewing Popular Items Under Heavy Load*

#### System Testing

System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective.

System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased. It includes both functional and NonFunctional testing.

**Test Case for End-to-End Login and Navigation**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_LOGIN\_NAV\_01** |
| Priority | High |
| Description | To verify the end-to-end functionality of logging in and navigating to the home screen. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access.  C. User has valid login credentials. |
| Steps | A. Open the login screen.  B. Enter valid email and password.  C. Click "Login".  D. Verify navigation to the home screen. |
| Input | Valid email (e.g., user@example.com) and password (Password123). |
| Expected result | User is successfully logged in and navigated to the home screen. |
| Status | Tested, passed. |

##### *Table 167: System Test Case – Login and Navigation*

**Test Case for End-to-End Sign Up and Profile Update**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_SIGNUP\_PROFILE\_01** |
| Priority | High |
| Description | To verify the end-to-end functionality of signing up and updating the user profile. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access. |
| Steps | A. Open the sign-up screen.  B. Enter valid email, password, and confirm password.  C. Click "Sign Up".  D. Verify navigation to the home screen.  E. Open the Edit Profile screen.  F. Enter valid profile details.  G. Click "Save". |
| Input | Valid email (e.g., newuser@example.com), password (Password123), and confirm password (Password123).  Valid profile details (e.g., name, email). |
| Expected result | User is successfully registered, navigated to the home screen, and profile details are updated successfully. |
| Status | Tested, passed. |

##### *Table 168: System Test Case – Sign Up and Profile Update*

**Test Case for End-to-End Adding Items to Cart and Checkout**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_CART\_CHECKOUT\_01** |
| Priority | High |
| Description | To verify the end-to-end functionality of adding items to the cart and completing the checkout process. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items to add to the cart. |
| Steps | A. Open the screen with the item.  B. Click "Add to Cart".  C. Open the cart screen.  D. Verify the item is in the cart.  E. Click "Checkout".  F. Complete the payment process. |
| Input | Mock data for item details and payment information. |
| Expected result | Item is added to the cart, and the checkout process is completed successfully. |
| Status | Tested, passed. |

##### *Table 169: System Test Case – Adding Items to Cart and Checkout*

**Test Case for End-to-End Viewing and Removing Items from Cart**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_VIEW\_REMOVE\_CART\_01** |
| Priority | High |
| Description | To verify the end-to-end functionality of viewing and removing items from the cart. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items in the cart. |
| Steps | A. Open the cart screen.  B. View items in the cart.  C. Remove an item from the cart. |
| Input | Mock data for cart items. |
| Expected result | Items are displayed correctly, and the selected item is removed from the cart. |
| Status | Tested, passed. |

##### *Table 170: System Test Case – Viewing and Removing Items from Cart*

**Test Case for End-to-End Viewing and Adding Favorite Items**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_VIEW\_ADD\_FAVORITES\_01** |
| Priority | Medium |
| Description | To verify the end-to-end functionality of viewing and adding favorite items. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Favorites screen.  B. View the list of favorite items.  C. Add a new item to favorites. |
| Input | Mock data for favorite items. |
| Expected result | Favorite items are displayed correctly, and the new item is added to the favorites list. |
| Status | Tested, passed. |

##### *Table 171: System Test Case – Viewing and Adding Favorite Items*

**Test Case for End-to-End Viewing Order History**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_VIEW\_ORDER\_HISTORY\_01** |
| Priority | Medium |
| Description | To verify the end-to-end functionality of viewing order history. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has placed orders. |
| Steps | A. Open the Orders screen.  B. View the list of orders. |
| Input | Mock data for orders. |
| Expected result | Orders are displayed correctly in the order history. |
| Status | Tested, passed. |

##### *Table 172: System Test Case – Viewing Order History*

**Test Case for End-to-End Viewing Restaurant Menu**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_VIEW\_RESTAURANT\_MENU\_01** |
| Priority | Medium |
| Description | To verify the end-to-end functionality of viewing the restaurant menu. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Restaurant Menu screen.  B. View the restaurant's menu. |
| Input | Mock data for restaurant menu. |
| Expected result | Restaurant menu is displayed correctly. |
| Status | Tested, passed. |

##### *Table 173: System Test Case – Viewing Restaurant Menu*

**Test Case for End-to-End Viewing Popular Items**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_VIEW\_POPULAR\_ITEMS\_01** |
| Priority | Medium |
| Description | To verify the end-to-end functionality of viewing popular items. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Popular Items screen.  B. View the list of popular items. |
| Input | Mock data for popular items. |
| Expected result | Popular items are displayed correctly. |
| Status | Tested, passed. |

##### *Table 174: System Test Case – Viewing Popular Items*

**Test Case for End-to-End Handling No Internet Connection**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SYSTEM\_NO\_INTERNET\_01** |
| Priority | High |
| Description | To verify the end-to-end functionality when there is no internet connection. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System has no internet connection. |
| Steps | A. Disable internet connection.  B. Open the app.  C. Verify the no internet widget is displayed. |
| Input | None |
| Expected result | No internet widget should be displayed correctly. |
| Status | Tested, passed. |

##### *Table 175: System Test Case – Handling No Internet Connection*

#### Regression Testing

Regression testing a black box testing technique that consists of re-executing those tests that are impacted by the code changes. These tests should be executed as often as possible throughout the software development life cycle. Types of Regression Tests:

Final Regression Tests

A "final regression testing" is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers.

Regression Tests

A normal regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement.

* + 1. **Selecting Regression Tests**
* Requires knowledge about the system and how it affects by the existing functionalities.
* Tests are selected based on the area of frequent defects.
* Tests are selected to include the area, which has undergone code changes many a times.
* Tests are selected based on the criticality of the features.
  + 1. **Regression Testing Steps**

Regression tests are the ideal cases of automation which results in better **R**eturn **on** **I**nvestment (ROI).

* Select the Tests for Regression.
* Choose the apt tool and automate the Regression Tests
* Verify applications with Checkpoints
* Manage Regression Tests/update when required
* Schedule the tests
* Integrate with the builds
* Analyze the results

**Test Case for Login Functionality**

**Final Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_LOGIN\_REG\_FINAL\_01** |
| Priority | High |
| Description | To validate the login functionality before deployment. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access.  C. User has valid login credentials. |
| Steps | A. Open the login screen.  B. Enter valid email and password.  C. Click "Login". |
| Input | Valid email (e.g., user@example.com) and password (Password123). |
| Expected result | User is successfully logged in and navigated to the home screen. |
| Status | Tested, passed. |

##### *Table 176: Final Regression Test Case – Login*

**Normal Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_LOGIN\_REG\_NORMAL\_01** |
| Priority | High |
| Description | To verify that recent changes have not broken the login functionality. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access.  C. User has valid login credentials. |
| Steps | A. Open the login screen.  B. Enter valid email and password.  C. Click "Login". |
| Input | Valid email (e.g., user@example.com) and password (Password123). |
| Expected result | User is successfully logged in and navigated to the home screen. |
| Status | Tested, passed. |

##### *Table 177: Normal Regression Test Case – Login*

**Test Case for Sign Up Functionality**

**Final Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SIGNUP\_REG\_FINAL\_01** |
| Priority | High |
| Description | To validate the sign-up functionality before deployment. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access. |
| Steps | A. Open the sign-up screen.  B. Enter valid email, password, and confirm password.  C. Click "Sign Up". |
| Input | Valid email (e.g., newuser@example.com), password (Password123), and confirm password (Password123). |
| Expected result | User is successfully registered and navigated to the home screen. |
| Status | Tested, passed. |

##### *Table 178: Final Regression Test Case – Sign Up*

**Normal Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_SIGNUP\_REG\_NORMAL\_01** |
| Priority | High |
| Description | To verify that recent changes have not broken the sign-up functionality. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User has internet access. |
| Steps | A. Open the sign-up screen.  B. Enter valid email, password, and confirm password.  C. Click "Sign Up". |
| Input | Valid email (e.g., newuser@example.com), password (Password123), and confirm password (Password123). |
| Expected result | User is successfully registered and navigated to the home screen. |
| Status | Tested, passed. |

##### *Table 179: Normal Regression Test Case – Sign Up*

**Test Case for Adding Items to Cart**

**Final Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_CART\_ADD\_REG\_FINAL\_01** |
| Priority | High |
| Description | To validate the functionality of adding items to the cart before deployment. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items to add to the cart. |
| Steps | A. Open the screen with the item.  B. Click "Add to Cart".  C. Verify the item is added to the cart. |
| Input | Mock data for item details. |
| Expected result | Item is added to the cart successfully. |
| Status | Tested, passed. |

##### *Table 180: Final Regression Test Case – Adding Items to Cart*

**Normal Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_CART\_ADD\_REG\_NORMAL\_01** |
| Priority | High |
| Description | To verify that recent changes have not broken the functionality of adding items to the cart. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has items to add to the cart. |
| Steps | A. Open the screen with the item.  B. Click "Add to Cart".  C. Verify the item is added to the cart. |
| Input | Mock data for item details. |
| Expected result | Item is added to the cart successfully. |
| Status | Tested, passed. |

##### *Table 181: Normal Regression Test Case – Adding Items to Cart*

**Test Case for Viewing Orders**

**Final Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ORDERS\_VIEW\_REG\_FINAL\_01** |
| Priority | Medium |
| Description | To validate the functionality of viewing orders before deployment. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has placed orders. |
| Steps | A. Open the Orders screen.  B. View the list of orders. |
| Input | Mock data for orders. |
| Expected result | Orders are displayed correctly in the order history. |
| Status | Tested, passed. |

##### *Table 182: Final Regression Test Case – Viewing Orders*

**Normal Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ORDERS\_VIEW\_REG\_NORMAL\_01** |
| Priority | Medium |
| Description | To verify that recent changes have not broken the functionality of viewing orders. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in.  C. User has placed orders. |
| Steps | A. Open the Orders screen.  B. View the list of orders. |
| Input | Mock data for orders. |
| Expected result | Orders are displayed correctly in the order history. |
| Status | Tested, passed. |

##### *Table 183: Normal Regression Test Case – Viewing Orders*

**Test Case for Viewing Restaurant Menu**

**Final Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_RESTAURANT\_MENU\_REG\_FINAL\_01** |
| Priority | Medium |
| Description | To validate the functionality of viewing the restaurant menu before deployment. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Restaurant Menu screen.  B. View the restaurant's menu. |
| Input | Mock data for restaurant menu. |
| Expected result | Restaurant menu is displayed correctly. |
| Status | Tested, passed. |

##### *Table 184: Final Regression Test Case – Viewing Restaurant Menu*

**Normal Regression Test:**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_RESTAURANT\_MENU\_REG\_NORMAL\_01** |
| Priority | Medium |
| Description | To verify that recent changes have not broken the functionality of viewing the restaurant menu. |
| Reference | Functional Requirement reference |
| Users | Customer |
| Pre-requisites | A. System is online.  B. User is logged in. |
| Steps | A. Open the Restaurant Menu screen.  B. View the restaurant's menu. |
| Input | Mock data for restaurant menu. |
| Expected result | Restaurant menu is displayed correctly. |
| Status | Tested, passed. |

##### *Table 185: Normal Regression Test Case – Viewing Restaurant Menu*

# Chapter 6: Tools and Techniques

#### 6.1 Languages Used in Development

1. **Dart**: The primary programming language used for developing the Eatonomy app in Flutter. Dart is selected for its performance and ease of use in building cross-platform mobile applications.
2. **Swift** [**[8]**](#eight)**:** Used for iOS-specific functions, Swift [[8]](#eight) is a powerful and intuitive programming language for iOS, macOS, watchOS, and tvOS.
3. **Kotlin**: Employed for Android-specific functions, Kotlin is a modern language that offers enhanced code safety and interoperability with Java.
4. NoSQL: Utilized in Google Firebase [[5]](#five) for database management, NoSQL databases are chosen for their scalability and flexibility in handling unstructured data.

#### 6.2 Applications and Tools

1. **Flutter Flow** [**[1]**](#one)**:** A visual development platform used to design the UI of the Eatonomy app. Flutter Flow [[1]](#one) accelerates the development process by allowing designers to create UI components visually.
2. **Figma** [**[2]**](#two)**:** A collaborative interface design tool used to create the visual design and layout of the Eatonomy app. Figma [[2]](#two)'s collaborative features enable seamless teamwork among designers.
3. **Xcode** [**[9]**](#nine)**:** The integrated development environment (IDE) for macOS used for developing iOS applications. Used the latest version of Xcode [[9]](#nine) (Xcode 15), in conjunction with macOS Sonoma 14.5 [[10]](#ten) and the iOS 17.5 simulator.
4. **Android Studio** [**[11]**](#eleven)**:** The official IDE for Android development. The latest version, Android Studio Koala 2024.1.1, is used for developing and testing the Android version of the Eatonomy app on devices like the Samsung Galaxy S21 Ultra Exynos [[12]](#twelve) running Android 14 [[13]](#thirteen).
5. **PlantUML** [**[14]**](#fourteen) **and Lucidchart** [**[15]**](#fifteen)**:** Tools used for creating UML diagrams and class diagrams. These diagrams help in visualizing the architecture and design of the application.

#### 6.3 Libraries and Extensions

1. **Firebase** [**[5]**](#five) **Libraries**

* **firebase\_core**: Core Firebase [[5]](#five) SDK for initializing Firebase [[5]](#five) in the Flutter app.
* **firebase\_auth**: Provides Firebase [[5]](#five) authentication services.
* **cloud\_firestore**: Provides Cloud Firestore services for real-time database.
* **firebase\_storage**: Allows uploading and storing user-generated content.
* **firebase\_messaging**: Manages push notifications.
* **firebase\_analytics**: Collects analytics data.

2. **Google Maps API** [**[7]**](#seven) **Library**

* **google\_maps\_flutter**: Provides Google Maps services for Flutter applications.

**3. Other Libraries**

* **provider**: State management library for managing app state.
* **http**: Used for making HTTP requests to external APIs.
* **shared\_preferences**: Used for storing simple data persistently.
* **url\_launcher**: Launches URLs in a mobile browser.

**6.4 Designing the App UI**

1. **Using Figma** [**[2]**](#two)

* **Process**:
  + **Wireframing**: Initial wireframes were created to outline the basic structure and layout of the app.
  + **Prototyping**: Interactive prototypes were developed to visualize user flows and interactions.
  + **Design Systems**: A consistent design system was established to maintain visual consistency across the app.

**2. Using FlutterFlow**

* **Process**:
  + **Visual Design:** Using FlutterFlow's drag-and-drop interface, the UI components designed in Figma [[2]](#two) were translated into FlutterFlow.
  + Integration: Integrated the designed components with Firebase [[5]](#five) for backend functionality.
  + **Code Export**: Exported the generated Flutter code for further customization and integration.

# Chapter 7: Summary and Conclusion

**7.1 Summary**

The Eatonomy application project was conceived to address the common challenges users face when deciding where and what to eat. By leveraging advanced technologies and a user-centric design approach, the project aimed to deliver a seamless and personalized food recommendation experience. The following is a summary of the key components and accomplishments of the project:

**Objective**: The primary objective of Eatonomy was to develop a user-friendly application that provides personalized food recommendations based on user preferences, order history, and real-time data. The application also aimed to facilitate easy access to restaurant information, menu items, and user reviews.

**Development Approach**:

1. **Languages and Frameworks**:
   * **Dart**: Used with Flutter for cross-platform mobile application development.
   * Swift [[8]](#eight): Employed for iOS-specific functionalities.
   * **Kotlin**: Used for Android-specific functionalities.
   * NoSQL: Utilized in Firebase [[5]](#five) for database management.
2. **Tools and Technologies**:
   * Flutter Flow [[1]](#one): A visual development platform used for designing the UI.
   * Figma [[2]](#two): Used for creating detailed UI/UX designs.
   * Xcode [[9]](#nine): The latest version, Xcode 15[[9]](#nine), was used for iOS development on macOS Sonoma 14.5 [[10]](#ten).
   * **Android Studio**: The latest version, Android Studio Jellyfish 2024.1.1, was used for Android development.
   * PlantUML [[14]](#fourteen) and Lucidchart [[15]](#fifteen): Employed for creating UML and class diagrams to visualize the system architecture.
   * Google Maps API [[7]](#seven): Integrated for providing location-based services.
3. **Architectural Patterns**:
   * Implemented the Model-View-ViewModel (MVVM) architecture to separate concerns, improve maintainability, and enhance testability.
4. **Functionality**:
   * Personalized food recommendations based on user data.
   * User authentication and profile management.
   * Integration with Firebase [[5]](#five) for backend services.
   * Location-based services using Google Maps API [[7]](#seven).
   * A responsive and intuitive UI designed in Figma [[2]](#two) and implemented using Flutter.

**Challenges and Solutions**:

* **Complex UI Implementation**: Addressed by using Flutter’s flexible UI components and design principles.
* **Efficient Data Handling**: Leveraged NoSQL databases and optimized queries for performance.
* **Cross-Platform Consistency**: Achieved through the use of Flutter, ensuring a consistent experience on both iOS and Android platforms.

**7.2 Conclusion**

The Eatonomy project successfully achieved its goal of creating a comprehensive food recommendation application. Through careful planning, the use of advanced technologies, and a focus on user experience, the application was developed to provide a valuable service to its users.

Key accomplishments include:

* **User-Centric Design**: The application was designed with the user in mind, offering personalized recommendations and an intuitive interface.
* Technological Integration: Successfully integrated various technologies such as Flutter, Firebase [[5]](#five), and Google Maps API [[7]](#seven) to enhance functionality.
* **Scalability and Maintainability**: The use of MVVM architecture and reusable components ensured that the application is scalable and easy to maintain.

Future enhancements, as outlined in the project, will further improve the application's functionality and user experience. Overall, the Eatonomy project helped the group to use modern development practices and technologies in solving real-world problems.

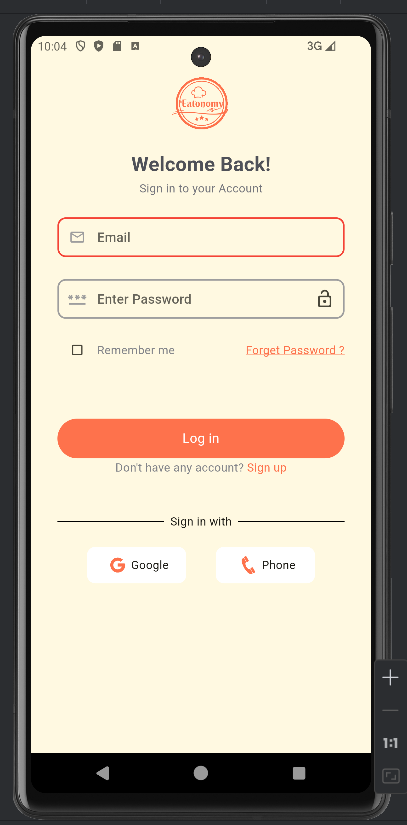
# Chapter 8: User Manual

### 8.1 Logging In

**Description:** The login screen is the first screen users encounter upon opening the Eatonomy app. It allows users to enter their credentials and access their accounts.

**Steps to Log In:**

1. Open the Eatonomy app.
2. Enter your email address in the "Email" field.
3. Enter your password in the "Password" field.
4. (Optional) Check the "Remember me" box to stay logged in.
5. Tap the "Log in" button.
6. If you don't have an account, tap "Sign up" to create one.
7. If you forgot your password, tap "Forget Password?" to reset it.

****

###### *Figure 51: Login Screen*

### 8.2 Signing Up

**Description:** The sign-up screen allows new users to create an account by providing necessary details.

**Steps to Sign Up:**

1. On the login screen, tap "Sign up."
2. Enter your name in the "Name" field.
3. Enter your email address in the "Email" field.
4. Enter a password in the "Enter Password" field.
5. Re-enter your password in the "Re-Enter Password" field.
6. (Optional) Check the box to accept the terms and conditions.
7. Tap the "Sign up" button to create your account.

##### *A screen shot of a phone Description automatically generated*

###### *Figure 52: Sign Up Screen*

### 8.3 Setting Food Preferences

**Description:** The food preferences screen allows users to select their food preferences for personalized recommendations.

**Steps to Set Food Preferences:**

1. Navigate to the Food Preferences screen.
2. Select your preferred food categories.
3. Tap "Continue" to save preferences.

##### *A screen shot of a phone Description automatically generated*

###### *Figure 53: Food Preferences Screen*

### 8.4 Browsing Categories

**Description:** The home screen displays various food categories and personalized recommendations.

**Steps to Browse Categories:**

1. Navigate to the Home screen.
2. Browse through the list of food categories.
3. Tap a category to view restaurants and dishes within that category.

##### *A screen shot of a phone Description automatically generated*

###### *Figure 54: Home Screen*

### 8.5 Viewing and Managing Cart

**Description:** The cart screen displays items that the user has added to their cart and allows them to proceed to checkout or modify cart contents.

**Steps to Manage Cart:**

1. Navigate to the Cart screen.
2. View the list of added items.
3. Adjust the quantity of each item or remove items from the cart.
4. Review the subtotal, delivery fee, and total amount.
5. Tap "Place Order" to proceed to payment.

##### *A screenshot of a phone Description automatically generated*

###### *Figure 55: Cart Screen*

### Editing Profile

**Description:** The profile screen allows users to view and edit their personal information.

**Steps to Edit Profile:**

1. Navigate to the Profile screen.
2. Tap "Edit Profile."
3. Update your first name, last name, email, or phone number.
4. Tap "Update" to save changes.

##### *A screenshot of a phone Description automatically generated*

###### *Figure 56: Edit Profile Screen*

### 8.7 Viewing Favorite Dishes

**Description:** The favorite dishes screen allows users to view and manage their favorite dishes.

**Steps to View Favorite Dishes:**

1. Navigate to the Favorite screen.
2. Select the "Dishes" tab to view a list of favorite dishes.

##### *A cell phone with a sandwich Description automatically generated*

###### *Figure 57: Favorite Dishes Screen*

### 8.8 Viewing Favorite Restaurants

**Description:** The favorite restaurants screen allows users to view and manage their favorite restaurants.

**Steps to View Favorite Restaurants:**

1. Navigate to the Favorite screen.
2. Select the "Restaurants" tab to view a list of favorite restaurants.

##### 

###### *Figure 58: Favorite Restaurants Screen*

### 8.9 Browsing Restaurant Menu

**Description:** The restaurant menu screen displays the selected restaurant's menu and allows users to add items to their cart.

**Steps to Browse Restaurant Menu:**

1. Navigate to a restaurant's details page.
2. View the menu items under different categories.
3. Tap on an item to view details and add it to your cart.

##### *A screen shot of a phone Description automatically generated*

###### *Figure 59: Restaurant Menu Screen*

### 8.10 Viewing Orders

**Description:** The orders screen allows users to track current and previous orders.

**Steps to View Orders:**

1. Navigate to the Orders screen.
2. View current orders under the "Current" tab.
3. Track order status or cancel an order if necessary.

##### *A screenshot of a food order Description automatically generated*

###### *Figure 60: Orders Screen*

### 8.11 Handling No Internet Connection

**Description:** The no internet screen informs users when their device is not connected to the internet.

**Steps to Handle No Internet Connection:**

1. If the no internet screen appears, check your internet connection.
2. Tap "Try Again" once your connection is restored.

##### *A screenshot of a phone Description automatically generated*

###### *Figure 61: No Internet Screen*

### 8.12 Managing Profile Settings

**Description:** The profile settings screen allows users to manage their account settings, including editing their profile, managing addresses, and logging out.

**Steps to Manage Profile Settings:**

1. Navigate to the Profile screen.
2. Tap on various options such as "Edit Profile," "Address," and "Notification Settings."
3. Update the relevant information or settings as needed.

##### *A screenshot of a phone Description automatically generated*

###### *Figure 62: Profile Settings Screen*

## Chapter 9: Lessons Learnt and Future Work

### 9.1 Lessons Learnt

Working on the Eatonomy project has been an enriching experience, filled with numerous lessons and insights that have significantly contributed to my growth as a developer and designer. Here are the key lessons I have learnt:

1. **Designing with Figma** [**[2]**](#two)**:**
   * Using Figma [[2]](#two) to design my application has been a particularly proud achievement. It allowed me to create a visually appealing and user-friendly interface. The ability to iterate quickly and collaborate effectively on designs was invaluable.
   * Lesson: Mastering a design tool like Figma [[2]](#two) enhances the ability to create cohesive and intuitive user interfaces, which is crucial for user satisfaction and engagement.
2. **Learning and Using Flutter Flow** [**[1]**](#one)**:**
   * Flutter Flow [[1]](#one) provided an efficient way to develop the application's frontend. It streamlined the process of creating complex UI components and integrating them seamlessly.
   * **Lesson**: Leveraging modern development tools can significantly reduce development time and increase productivity, allowing for rapid prototyping and iterative improvements.
3. **Data Integration with Python** [**[3]**](#three) **and Scrapy** [**[4]**](#four)**:**
   * Implementing a Python [[3]](#three) script using Scrapy [[4]](#four) to fetch restaurant data for our Firebase [[5]](#five) database was a crucial part of the project. This task improved my understanding of web scraping and data integration.
   * **Lesson**: Automating data collection processes not only saves time but also ensures that the data is comprehensive and up-to-date, which is essential for maintaining the relevance and accuracy of the app's content.
4. **Cross-Platform Development with Xcode and Swift** [**[8]**](#eight)**:**
   * Working with Xcode and learning Swift [[8]](#eight) to ensure cross-platform compatibility was a challenging yet rewarding experience. It broadened my technical skills and understanding of iOS development.
   * **Lesson**: Gaining proficiency in multiple programming languages and development environments is essential for creating versatile and robust applications that can reach a wider audience.
5. **Understanding and Implementing MVVM Architecture**:
   * Implementing the Model-View-ViewModel (MVVM) architecture helped in separating the UI from the business logic and data management, making the application more modular and easier to maintain.
   * **Lesson**: Using architectural patterns like MVVM enhances code maintainability, scalability, and testability, which are critical for long-term project sustainability.

**Efficient Use of Reusable Components**:

* + Throughout the development process, I learned the importance of creating and using reusable components to maintain consistency and reduce redundancy.
  + **Lesson**: Reusable components save development time, ensure consistency across the application, and make it easier to implement changes and new features.

1. **Data Structure Concepts**:
   * Developing the application reinforced my understanding of data structures and their efficient use in data extraction and representation on the screen.
   * **Lesson**: A strong grasp of data structures is crucial for optimizing application performance and ensuring smooth data handling and presentation.
2. **Database Query Optimization**:
   * Efficiently using database queries to extract and manipulate data was essential for ensuring the app's performance and responsiveness.
   * **Lesson**: Optimized database queries enhance the app's performance and provide a better user experience by reducing load times and ensuring data accuracy.
3. **Sparked Interest in Data Science and Mobile App Development**:
   * This project has ignited a keen interest in both data science and mobile app development. The integration of data scraping, data management, and mobile development has shown the vast potential of these fields when combined.
   * **Lesson**: Exploring interdisciplinary approaches can open up new avenues for innovation and personal growth, encouraging continuous learning and adaptation.

### 9.2 Future Work

While the Eatonomy project has achieved its primary objectives, there are several areas for future enhancement and exploration:

1. **Enhanced Data Analytics**:
   * Integrate advanced data analytics to provide users with more personalized recommendations and insights based on their preferences and behavior.
2. **Machine Learning Integration**:
   * Implement machine learning algorithms to predict user preferences, optimize search results, and improve the overall user experience.
3. **Advanced User Interaction Features**:
   * Develop features such as voice search, augmented reality for restaurant previews, and real-time order tracking to enhance user engagement.
4. **Expansion of Data Sources**:
   * Continuously expand the data sources for restaurant information to ensure comprehensive coverage and up-to-date details.
5. **Scalability and Performance Optimization**:
   * Focus on scaling the application to handle a larger user base and ensure optimal performance under increased load.
6. **Cross-Platform Enhancements**:
   * Further refine the cross-platform capabilities, ensuring seamless functionality and a consistent user experience across all devices.
7. **Community Features**:
   * Introduce community-driven features such as user reviews, ratings, and social sharing to foster a sense of community and enhance user engagement.

# References

[[1]](#one) A. (2024). "Flutter Flow." [Online]. Available: <https://flutterflow.io/>

[[2]](#two) Adobe. (2024). "Figma." [Online]. Available: <https://www.figma.com/>

[[3]](#three) G. (2024). "Python." [Online]. Available: <https://www.python.org/>

[[4]](#four) G. (2024). "Scrapy." [Online]. Available: <https://scrapy.org/>

[[5]](#five) Google. (2024). "Firebase." [Online]. Available: <https://firebase.google.com/>

[[6]](#six) Google. (2024). "NoSQL Database." [Online]. Available: <https://firebase.google.com/docs/firestore>

[[7]](#seven) Google. (2024). "Google Maps API." [Online]. Available: <https://developers.google.com/maps>

[[8]](#eight) Apple. (2024). "Swift." [Online]. Available: <https://swift.org/>

[[9]](#nine) Apple. (2024). "Xcode 15.0." [Online]. Available: <https://developer.apple.com/xcode/>

[[10]](#ten) Apple. (2024). "macOS Sonoma 14.5." [Online]. Available: <https://www.apple.com/macos/sonoma/>

[[11]](#eleven) Android. (2024). "Android Studio 2024.1.1." [Online]. Available: <https://developer.android.com/studio>

[[12]](#twelve) Samsung. (2024). "Samsung Galaxy S21 Ultra Exynos." [Online]. Available: <https://www.samsung.com/global/galaxy/galaxy-s21-ultra-5g/>

[[13]](#thirteen) Google. (2024). "Android 14." [Online]. Available: <https://developer.android.com/about/versions/14>

[[14]](#fourteen) PlantUML. (2024). "PlantUML." [Online]. Available: <http://plantuml.com/>

[[15]](#fifteen) Lucidchart. (2024). "Lucidchart." [Online]. Available: <https://www.lucidchart.com/>

[[16]](#sixteen) JetBrains. (2024). "Kotlin." [Online]. Available: <https://kotlinlang.org/>

[[17]](#seventeen) Google. (2024). "Dart." [Online]. Available: <https://dart.dev/>

[[18]](#eighteen) Google. (2024). "Flutter." [Online]. Available: <https://flutter.dev/>

# Appendix

* A section at the end of a document that includes information that is too detailed for the text of the document itself and would "burden the reader" or be "distracting," or "inappropriate" (APA, 2010, p. 38-9).
* The content in the appendices should be "easily presented in print format" (APA, 2010, p. 39).

**Appendix A**

**Demographic Details**

The study focuses on Pakistani students and adults, categorized into two age groups: 16-26 years and 40-65 years. Both male and female participants from urban areas were included. The participants possess a decent understanding of using mobile phones and the internet.

**Detailed Descriptions**

1. **Age Group 16-26:**
   * Students in this age group are primarily engaged in educational activities, often utilizing mobile applications for learning and social interaction.
   * They are proficient in using various mobile applications and internet services, including social media, educational platforms, and entertainment services.
2. **Age Group 40-65:**
   * Adults in this age group are generally employed or retired, with a focus on staying connected through mobile devices.
   * Their usage includes communication, news, social media, and various utility applications for daily tasks.

**Urban Area Focus:**

* The urban population has better access to mobile technology and internet services compared to rural areas.
* Participants from cities such as Lahore, Karachi, and Islamabad were primarily considered due to their higher exposure to technology.

**Knowledge and Usage:**

* Both groups have a good understanding of mobile technology, with the younger group being more adept at using advanced features.
* Adults, while less frequent users of new technologies, are competent in using smartphones for essential functions.

This appendix provides a detailed overview of the demographics and usage patterns of the study participants, highlighting their familiarity and interaction with mobile technology.